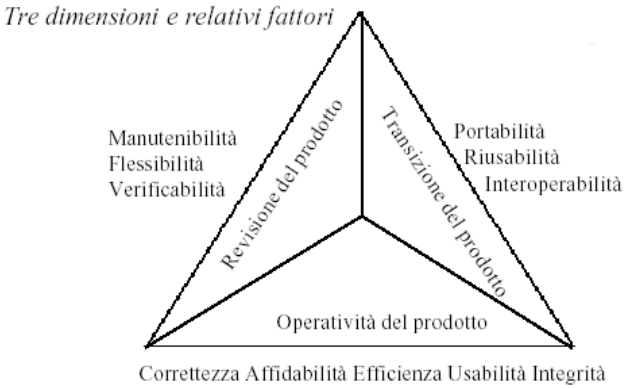
CAPITOLO 1-SW ENGINEER

- **Che cos'è il software?** E' un insieme di programmi e la loro documentazione. Esistono due fondamentali tipi di prodotti software:
  - SW GENERICI(sono i sistemi prodotti da un'organizzazione di sviluppo e venduti al mercato "aperto" per tutti i clienti che sono in grado di comprarli. Es: sw per databases, word, drawing…); ampio spettro di utilizzo
  - SW SPECIFICI(sw destinati e commissionati da una particolare clientela. Es: sistema di controllo di devices tecnologici o sistemi di controlli del traffico aereo.)
- **Quali sono le caratteristiche di un sw?**
  Mentre i costi sw aumentano i costi hw diminuiscono, in quanto c'è una difficoltà della stima dei costi sw derivanti a problemi quali l'incapacità di dimensionare accuratamente un progetto; l'incapacità di definire nel suo complesso il processo e l'ambiente operativo del progetto; effettuare una valutazione inadeguata del personale sia in quantità che in qualità e una sbagliata/inadeguata raccolta dei requisiti per la stima di attività specifiche all'interno del progetto.
  Mentre il tempo di sviluppo aumenta i costi per mantenere un software aumentano.
  Mentre gli errori sw incrementano i guasti hw diminuiscono, significa che quando l'hw funziona correttamente si possono effettuare le correzioni a livello software.
- **Quali sono gli attributi di un sw?** Le qualità di un sw possono essere INTERNE(riguardano caratteristiche di implementazione-non visibili all'utente finale) ed ESTERNE(riguardano le caratteristiche di funzionalità al momento dell'esecuzione-visibili all'utente finale). Ovviamente non è possibile ottenere le qualità esterne se non si dispone delle qualità interne.



  Gli attributi essenziali sono riassunti dal triangolo di McCall, tra questi troviamo:
  - MAINTAINABILITY: sw dev'essere scritto in modo che possa evolvere con il cambiamento delle esigenze dei consumatori.
  - DEPENDABILITY E SECURITY: Non deve creare danni fisici o economici nell'eventualità di "system failure". Utenti malintenzionati non devono poter accedere e danneggiare il sistema.
  - EFFICIENCY: sw non deve sprecare risorse
  - ACCEPTABILITY/USABILITY: il sw dev'essere accettabile dagli utenti per cui è stato sviluppato: comprendibile, usabile e compatibile con gli altri sistemi usati da loro.
- **Che cos'è l'ingegneria del sw?** L'ingegneria del sw è una disciplina che riguarda tutti gli aspetti sulla produzione di un sw. Ha il compito di creare e mantenere applicativi sw

usando tecnologie e tecniche derivanti sia dall'informatica che da altri campi come l'ing gestionale.

- **Qual è la differenza tra ingegneria del sw e informatica?** L'informatica riguarda la teoria e i fondamenti come gli algoritmi e i linguaggi, mentre l'ing del sw riguarda le pratiche di sviluppo e consegna di un sw di qualità.
- **Qual è la differenza tra ingegneria del sw e ingegneria dei sistemi?** L'ingegneria dei sistemi riguarda tutti gli aspetti dello sviluppo di un sistema basato su computer, inclusi aspetti hw, sw e di processo, mentre l'ing del sw è solo una parte di es
- **Cos'è il processo di produzione del sw?** Un set di attività con lo scopo di sviluppare o/ed evolvere il sw.
  Le attività generiche di tutti i processi di produzione sono:
  - SPECIFICA: Cosa deve fare il sistema e quali sono i vincoli di progettazione
  - SVILUPPO: Produzione del sistema sw
  - VALIDAZIONE: Verifica che il sistema sia coerente alle richieste del cliente (testing)
  - EVOLUZIONE: Modifica del sw in base alle nuove esigenze del cliente
- **Quali sono i problemi nel processo di sviluppo di un sw?**
  - Specifiche incomplete/incoerenti
  - Mancanza di distinzione tra specifica, progettazione ed implementazione
  - Assenza di un sistema di validazione
  - Il sw non si consuma: la manutenzione è la modifica del prodotto rispetto a nuove esigenze da parte del cliente.
- **Cos'è un modello di produzione del sw?** Una rappresentazione semplificata del processo di produzione, presentata da una specifica prospettiva. Esso include le attività che sono parte del processo, prodotti sw e il ruolo delle persone implicate in esso.
- **Quali sono i costi legati alla produzione di sw**? Il 60% dei costi è legato ai costi di sviluppo e il 40% alla verifica e validazione del sw; anche se questi costi variano a seconda del tipo di sistema che si deve realizzare. I costi di sviluppo sono così alti e sono difficili da stimare, perché comprendono anche i costi di evoluzione del sw.
  I costi variano a seconda del tipo di sistema che dev'essere sviluppato, alla performance e all'affidabilità.
  L'80% degli errori avviene durante la raccolta requisiti o la fase di design.
  Esistono 3 categorie importanti dei progetti sw: SUCCESS, ossia il progetto terminato con successo nei tempi previsti(16,2%), CHALLENGED, progetto consegnato in ritardo e con un aumento di costi (52,7%) E IMPAIRED, progetto cancellato per costi troppo alti e per un tempo di consegna non definito-troppo alto(31,1%).
- **Quali sono le principali sfide che l'ingegnere del sw deve affrontare?**
  -THE HETEROGENEY CHALLENGE: La sfida dello sviluppo di tecniche per la costruzione di sw sia affidabili che flessibili abbastanza per far fronte all'eterogeneità dei linguaggi di programmazione.
  -THE DELIVERY CHALLENGE: Ridurre i tempi di consegna per sistemi grandi e complessi senza compromettere la qualità dei sistemi.
  -THE TRUST CHALLENGE: Sviluppare tecniche che dimostrano la sicurezza del software agli utenti.

Le richieste di tempi di consegna ridotti e lo sviluppo di software affidabili, mantenendo legacy system eterogeni (compatibili ed evoluti sia a livello hw che sw all'era attuale). Queste sfide non sono indipendenti fra loro, per esempio per fare rapidi cambiamenti ad un sistema legacy, abbiamo bisogno di strumenti e tecniche innovativi che si ottengono usando metodi esistenti dell'ingegneria del sw.

CAPITOLO 2:

- **Complessità di un sistema: problema di scalabilità dei sottosistemi.**
  Un sistema è un insieme di componenti correlati che lavorano insieme per raggiungere un obiettivo. I sistemi che includono sw rientrano in due categorie:
  TECHNICAL COMPUTER-BASED SYSTEMS: sistemi che includono componenti hw e sw ma non le procedure e i processi; per esempio tv, cellulari e molti sw per pc.
  SOCIO-TECHNICAL SYSTEMS: includono uno o più technical system, ma includono anche la conoscenza di come il sistema potrebbe essere utilizzato per raggiungere degli obiettivi. Questi sistemi hanno definito processi operazionali, che includono persone come parte del sistema e sono governati da politiche organizzative e regole.
  Questi socio-technical systems hanno 3 essenziali caratteristiche:
  - Hanno delle proprietà emergenti (funzionali e non)
  - Sono spesso non deterministici, ossia che i comportamenti del sistema non sono sempre gli stessi, ma dipendono dalle persone che effettuano gli input.
  - La misura con cui il sistema supporta gli obiettivi organizzativi dipende dalla stabilità di questi obiettivi, ma anche dalle relazioni e conflitti tra gli obiettivi organizzativi e come le persone li interpretano.

  I sistemi sono spesso gerarchici e includono sottosistemi.(per esempio il comando di polizia e il suo sistema di controllo potrebbe includere un sistema di localizzazione geografico)

  I sottosistemi sono indipendenti, per cui possono essere utilizzati in differenti sistemi.

  Il numero di sottosistemi è dato dall'ampiezza dell'albero di decomposizione del sistema e dalla sua profondità. Data l'eterogeneità di tipo dei sottosistemi è difficile definire le relazioni tra di essi, quindi servono capacità scientifiche e manageriali per effettuare un socio-technical system (approccio multidisciplinare).

- **Modellazione di un sistema**
  Prima di modellare un sistema di valutano le caratteristiche qualitative o quantitative.
  - Si definisce una serie di processi che rappresentano l'entità della realtà fisica
  - Si definisce il comportamento di ciascun processo
  - Si definiscono i dati del sistema, che possono essere esogeni(che provengono dall'esterno) o endogeni(che il sistema scambia al proprio interno).

  Gli strumenti di modellazione e simulazione permettono di costruire sistemi con una diminuzione di errori, perché già modellati.

  Esistono modelli architetturali(mostrano in modo astratto la struttura in sottosistemi), gerarchici(organizzazione ad albero) e funzionali(rappresentano i flussi di informazione tra i vari sottosistemi).

- **Affidabilità di un sistema**

  L'affidabilità complessiva dipende da quella hw, sw e quella degli operatori. Per verificare l'affidabilità ci poniamo le seguenti domande:
  - Quanto è probabile un fallimento hw e quanto tempo è richiesto per ripararlo?
  - Quanto è probabile che una componente sw produca un output sbagliato?
  - Quanto è probabile che un operatore commetta errori?

  La maggior parte degli errori dipende dal software e l'interdipendenza delle componenti(le componenti di un sistema possono operare in modo indipendente, ma quando sono integrate in un sistema dipendono dalle altre componenti) fa sì che gli errori si propaghino per tutto il sistema, per cui molti sistemi complessi sono considerati un fallimento a causa software.

  Il sistema deve, quindi, essere RESILIENTE, ovvero continuare ad operare correttamente in presenza di fallimenti di uno o più componenti.

- **Quali sono le proprietà emergenti di un sistema?**

  Proprietà del sistema visto globalmente (non delle singole componenti), in quanto derivanti dall'integrazioni delle componenti.
  Esistono le proprietà funzionali, ovvero le proprietà che appaiono solo quando le varie componenti vengono integrate e cooperano per uno scopo comune e non-funzionali, ovvero le proprietà che riguardano il comportamento del sistema nel suo ambiente operativo, come reliability, performance, safety etc.
- **Quale relazione intercorre tra l'affidabilità delle componenti di un sistema?**

  L'ambiente in cui il sistema opera può influenzare la sua affidabilità, in quanto l'ambiente può condizionare il comportamento di un sistema.
  Es. i guasti hw possono causare segnali spuri nel sw e causare la produzione di segnali non corretti, mentre gli errori nel sw possono causare stress nell'operatore aumentando la sua propensione a commettere errori.
- **Acquisizione di un sistema, progettazione, sviluppo**

  Un sistema può essere costruito o acquisito, nel caso dell'acquisizione, è necessario sapere la specifica del sistema(cosa è richiesto al sistema) e l'architettura di progetto.
  PROCESSO DI ACQUISIZIONE:
  sistemi "off the shelf" (quali sistemi comprare)
  1- Analisi di mercato
  2- Adattare i requisiti
  3- Scegliere il sistema
  4- Richiedere i preventivi
  5- Scegliere il fornitore

  sistemi dedicati (quali sistemi sviluppare)
  1- Analisi di mercato
  2- Richiesta di offerte
  3- Selezionare le offerte
  4- Negoziare il contratto
  5- Contrattare lo sviluppo

Nella progettazione di un sistema c'è multidisciplinarietà, perché coinvolge vari tecnici di aree diverse(ing del sw, elettronico, meccanico, elettrico, civile, architetto etc); di solito segue un modello di sviluppo a cascata per poter sviluppare parallelamente le diverse componenti del sistema; dati gli alti costi di modifica c'è poco spazio per le iterazioni fra le diverse fasi; il sottosistema sw dev'essere il più flessibile (in quanto le modifiche hw sono più costose e complesse e dovrebbero essere compensate dal sw).

+ segnalibro pg.9

SVILUPPO DI UN SISTEMA:

1- Definizione dei requisiti
2- Progettazione in sottosistemi(disegno del sistema, come le funzionalità del sistema devono essere fornite dalle diverse componenti)
   -organizzare i requisiti, separandoli in gruppi collegati
   -identificare i sottosistemi (ogni sottosistema soddisfa un gruppo di requisiti)
   -assegnare i requisiti ai sottosistemi
   -specificare le funzionalità dei sottosistemi
   -definire le interfacce dei sottosistemi
3- Sviluppo dei sottosistemi implementare ciascuno dei sottosistemi individuati(può richiedere un nuovo processo di sviluppo)
4- Integrazione del sistema Mette insieme hw, sw e risorse umane. I sottosistemi vengono integrati uno alla volta per ridurre i costi di individuazione degli errori.
5- Installazione del sistema inserire il sistema completo nel suo sistema operativo. Questo può portare a vari problemi, quali
   -coesistenza dei due sistemi (nuovo + precedente)
   -l'ambiente finale può essere diverso da quello di sviluppo
   -resistenza al nuovo sistema da parte degli utilizzatori
   -problemi pratici(cablaggio, corrente …)
6- Mantenimento del sistema La durata di un sistema è legata alla sua dimensione
   ->l'evoluzione è costosa (interdipendenza sottosistemi, affidabilità etc)
7- Decomissioning

- **Come vengono definiti i requisiti?**
  I requisiti globali sono quelli funzionali (cosa il sistema deve fare), non-funzionali(proprietà del sistema + vincoli) e i requisiti che un sistema non deve avere.
- **Esempio: sistema d'allarme**

CAPITOLO 3:

- **Processo di produzione del sw: ciclo di risoluzione di un problema**
  Il processo di produzione di un sw è l'insieme di attività per la specifica, il progetto, l'implementazione e la verifica dei sistemi sw.
  - Status quo: stato della situazione attuale
  - Definizione del problema: individuo il problema da risolvere
  - Sviluppo-tecnico: risolvo il problema con una opportuna tecnologia
  - Integrazione: consegno i risultati al committente

Questo ciclo di risoluzione può essere effettuato sia a livello dell'intero sistema, che ai singolo sottosistemi o funzioni.

Process model	Process visibility
Waterfall model	Good visibility, each activity produces some deliverable
Evolutionary development	Poor visibility, uneconomic to produce documents during rapid iteration
Formal transformations	Good visibility, documents must be produced from each phase for the process to continue
Reuse-oriented development	Moderate visibility, it may be artificial to produce documents describing reuse and reusable components
Spiral model	Good visibility, each segment and each ring should produce some document

In seguito vengono elencati tutti i modelli di processo del sw:

- **Modello a cascata**

Il modello a cascata si compone delle seguenti 5 fasi:

1. Definizione dei requisiti: si lavora alla specifica dei requisiti dopo aver avuto un dialogo con il committente
2. Progettazione del sistema: si effettua la progettazione dell'architettura del sistema e delle sue componenti
3. Implementazione e testing delle singole componenti
4. Integrazione e testing delle singole componenti
5. Installazione e manutenzione: si installa il sistema e se ne si prende cura

È possibile passare da una certa fase a quella precedente, tuttavia il cambiamento dei requisiti in corso d'opera è estremamente oneroso.

Dal punto di vista della visibilità, il modello di sviluppo a cascata è buono poiché produce deliverables ad ogni fase.

Il rischio di questo modello di sviluppo dipende dalla familiarità del team nei confronti del problema da affrontare

- **Modello RAD (Rapid Application Development)**

Il modello RAD è particolarmente indicato nei casi in cui si deve sviluppare un progetto facilmente partizionabile in cui non vi sia la necessità di utilizzare interfacce appositamente definite. Inoltre non è il modello indicato qualora si utilizzassero tecnologie innovative con alto rischio di incontrare problemi in corso d'opera.

Le fasi che compongono questo modello sono:

1. Comunicazione
2. Pianificazione
3. Modellazione
4. Costruzione
5. Deployment

In particolare, la modellazione e la costruzione sono ad opera di ciascun team incaricato di lavorare su una partizione del progetto.

- **Modello evolutivo**

Lo sviluppo evolutivo dà importanza alla produzione di prototipi usa-e-getta a partire da definizioni dei requisiti di massima, ossia si dà molta importanza alla scrittura del codice, il quale evolve secondo le direttive del cliente. Le fasi sono:

1. Definizione di massima
2. Specifica, che produce la versione iniziale
3. Sviluppo, che produce la versione intermedia
4. Validazione, che produce la versione finale

La visibilità è scarsa a causa della produzione di documenti carente, il ché può elevare il livello di rischio (che però allo stesso tempo è ridotto grazie allo sviluppo di numerosi prototipi). È

fondamentale che il prodotto finale non sia un prototipo: una volta raggiunti gli obiettivi di sviluppo è opportuno riscrivere il software per la release. Questo modello si presta particolarmente per progetti di piccola dimensione o con una vita attesa corta.

- **Modello formale o trasformazionale** 4

  Il modello di sviluppo formale è adeguato nel momento in cui ci si prefigge, ad esempio, di codificare linguaggio matematico.

  1. Definizione dei requisiti
  2. Specifica formale
  3. Trasformazione formale
  4. Integrazione e test

  Il rischio è legato all'esperienza degli sviluppatori e dalla necessita di tecnologie avanzate. La visibilità è buona, ogni fase dovrebbe produrre documentazione affinché il processo di sviluppo possa svolgersi.

- **Modello basato sul riutilizzo** A API

  Il modello di sviluppo basato sul riutilizzo si basa sull'uso di componenti off-the-shelf già scritte e testate per la produzione del software.
  Le fasi di questo modello sono:

  1. Analisi dei componenti
  2. Adattamento dei requisiti
  3. Progettazione del sistema
  4. Integrazione

  La visibilità è buona, però può essere macchinoso scrivere documentazione per componenti off-the-shelf. Il rischio è ridotto, dato che si presuppone la correttezza dei componenti riutilizzati.
  Questo modello si presta molto allo sviluppo di software ad oggetti.

- **Modello a spirale** iterativo

  Il modello di sviluppo a spirale prevede i seguenti step, che si reiterano numerose volte fino alla release finale del software:

  1. Comunicazione con il cliente
  2. Progettazione del sistema
  3. Analisi dei rischi
  4. Strutturazione
  5. Costruzione e release
  6. Valutazione del cliente

  Questo modello è indicato per progetti di grandi dimensioni. La visibilità è buona perché ogni spicchio della spirale produce documentazione.
  I rischi sono minimizzati dalle numerose iterazioni, però la erronea valutazione dei rischi può ripercuotersi successivamente.

- **Modello RUP (Rational Unified Process)**

  Il Rational Unified Process è un modello che si rifà a UML per la progettazione di sistemi che prende elementi da vari modelli generici e fornisce buone prassi in fatto di progettazione e specifica. È particolarmente indicato per sistemi che fanno uso di cicli. Le sue fasi sono:

  1. Inception: il progetto ha inizio con il dialogo con il cliente, il quale fornisce i requisiti del sistema; si propone un'architettura di base e un piano per la natura iterativa e incrementale del progetto
  2. Elaboration: comunicazione con il cliente, modellazione, si ampliano gli use cases, e si espande la rappresentazione dell'architettura
  3. Construction: si sviluppa il software
  4. Transition: il software è pronto per essere installato nell'ambiente reale

  Queste milestones, se non producono risultati soddisfacenti, devono essere reiterate oppure si deve abortire il progetto.

- **Valutazione dei rischi nei vari modelli**

  impatto, conseguenze, probabilità che si verifichi.

  Per ogni rischio bisogna valutare l'impatto, le conseguenze e la probabilità che si verifichi una situazione problematica che impedisca al programma di funzionare adeguatamente. Non tutti i rischi, e i fallimenti che ne derivano sono uguali, per esempio il rischio di un fallimento corruttivo è molto più grave del rischio di un fallimento transiente recuperabile a parità di probabilità di occorrenza. In altre parole bisogna chiedersi "cosa si perde qualora il rischio si verificasse?"
  Il fattore di rischio si misura come prodotto tra probabilità che il rischio che si verifichi e la la sua gravità.

  gravità X prob. che si verifichi
  ↳ FATTORE DI RISCHIO

- **Metodologie di sviluppo agile (xp)-1999**

  Il modello di progettazione Agile, o Extreme Programming, è un modello molto recente e si basa su quattro punti:

  1. Le persone e le interazioni sono più importanti di software e processi
  2. Il software funzionante è più importante rispetto alla documentazione
  3. La comunicazione è fondamentale
  4. La capacità di risposta ai cambiamenti è più importante rispetto ad aderire al progetto

  Questo modello ha la particolarità di prevedere che lo sviluppo sia condotto da due sviluppatori, uno che scrive codice e l'altro che assiste, in maniera tale da favorire la concentrazione e la buona programmazione. Il testing(casi di prova) è obbligatorio, se il test non funziona si ferma lo sviluppo e si fissa il problema. Inoltre presenta costi dovuti al cambiamento dei requisiti molto inferiori rispetto a quelli del modello a cascata.

CAPITOLO 4: SW PROJECT MANAGMENT

- **Che cos'è un progetto?**
  Un progetto è un insieme ben definito di attività che hanno:
  -un inizio
  -una fine

-uno scopo
-viene portato avanti da un insieme di persone
-utilizza un insieme di risorse
-non è un lavoro di routine
Il project manager deve conciliare tempi, scopo e budget.

- **Quali sono le possibile problematiche?**
  Il sw è intangibile(intoccabile), Per valutare i progressi bisogna basarsi sulla documentazione. Non esiste uno standard per la produzione di un progetto perché ogni caso è a sé.
  Motivi per cui un progetto può essere in ritardo:
  - Deadline non realistica(impostata da un componente esterno allo staff tecnico)
  - Cambiamenti dei requisti imposti dal cliente
  - Stima troppo ottimistica
  - Rischi non presi in considerazione (difficoltà tecniche e umano imprevedibili)
  - Problemi di comunicazione nel gruppo
  - Incapacità di riconoscere un ritardo e mancata attuazione contromisure.
- **Qual è il caso peggiore?** WORST CASE
  In caso di riconoscimento ritardo ci sono 3 possibilità:
  1- Aumento budget e aggiungo risorse (possibile solo all'inizio)
  2- Elimino le funzionalità non essenziali -> sviluppo prototipo
  3- Non tengo conto dell'analisi e procedo fino al fallimento.
- **Come evitare la maggior parte dei problemi?**
  Seguire i principi fondamentali per la pianificazione:
  1- Un progetto dev'essere ripartito in attività e compiti di dimensioni ragionevoli.
  2- Il numero di persone coinvolte è definito a priori e non si deve accedere.
  3- Determinare le dipendenze tra attività e compiti
  4- Determinare quali compiti svolgere in parallelo e quali svolgere in sequenza
  5- Ogni compito deve avere una data di inizio e una di fine e deve essere associato a qualcuno, che deve produrre per ognuno un risultato predefinito.
  6- Ad ogni compito si deve associare almeno un punto di controllo, ossia verificare la qualità di uno o più compiti.
- **Quali sono gli attori sulla scena e di cosa si occupano?** 5

1 SENIOR MANAGERS-> definiscono gli aspetti economici del progetto
2 PROJECT MANGERS-> pianificano, organizzano e controllano lo sviluppo del progetto. Più precisamente si occupa della stesura del progetto, della stima dei costi, del planning e scheduling, del monitoraggio e revisione del progetto, della selezione dello staff tecnico e dell'assegnazione dei compiti e della stesura dei rapporti e delle presentazioni. Per questi motivi è una figura necessaria ed essenziale nel progetto.
3 PRATICTIONERS-> chi ha competenze tecniche per realizzare parti del progetto (sviluppatori)
4 CUSTOMERS-> cliente che definisce i requisiti del sw
5 END USERS-> chi utilizzerà il sw una volta sviluppato

- **Come organizzare le attività e lo scheduling?**
  Le attività devono produrre documentazione affinché i manager possano valutazre l'avanzamento del progetto. Le MILESTONES sono i punti finali di ogni attività e le DELIVERABLES sono i risultati del progetto consegnati al cliente.
  (intermedi) (di ogni fase)

Suddivisione in task
stime tempo e risorse necessarie per completare ogni task

Lo scheduling è la suddivisione in task del progetto e la stima del tempo e risorse necessarie per completare ogni compito.
Buona pratica è minimizzare le dipendenze tra task per evitare la propagazione dei ritardi a catena.
I problemi dello scheduling derivano da una difficoltà di stima dei costi di sviluppo e del tempo, dato che la produttività non è proporzionale al numero di persone che lavorano in un progetto.
Generalmente viene utilizzata la regola del 40-20-40

- 40% del tempo per l'analisi e la progettazione
- 20% del tempo per la scrittura del codice
- 40% per il collaudo

- **Possiamo generalizzare?**
  NO. A parità di obiettivi si hanno benefici maggiori impiegando un numero di persone inferiore per un periodo più lungo.
- **Creazione di un progetto:**
  1- Introduzione: definizione obiettivi e vincoli
  2- Organizzazione: definizione persone coinvolte e ruoli
  3- Analisi dei rischi
  4- Stime di costo e temporali per acquisire le risolse hw e sw richieste
  5- Suddivisione del lavoro in attività e identificazione deliverables e milestones
  6- Scheduling del progetto: identificazione dipendenze tra attività e stime tempo richiesto per milestones, assegnazione personale alle attività
  7- Controllo e rapporto sulle attività
- **Quali sono le tipologie di team?**
  Esistono 3 tipologie di team:
  - DEMOCRATICO DECENTRALIZZATO: team in cui non c'è un leader e serve un consenso di gruppo sulle soluzione e sull'organizzazione del lavoro, presenta una comunicazione orizzontale, ossia tra persone dello stesso livello → gestione della complessità
    E' difficile da imporre e non è scalabile, ma funziona bene per problemi difficili, come la ricerca e l'individuazione degli errori è più veloce.
  - CONTROLLATO CENTRALIZZATO: il team leader decide sulle soluzioni e sull'organizzazione, la comunicazione avviene in modo verticale tra i team leader e gli altri membri. project manager - staff tecnico - backup engineer - sw librarian
  - CONTROLLATO DECENTRALIZZATO: presenta un leader che coordina il lavoro e la risoluzione dei problemi è di gruppo, ma l'implementazione è gestita dai leader dei sottogruppi. Presenta quindi una comunicazione verticale con il leader e orizzontale tra i membri dei vari sottogruppi.
- **Quali sono i ruoli in un team controllato decentralizzato?**
  Project manager: pianifica, coordina e supervisiona le attività del team
  Technical staff: conduce analisi e sviluppo
  Backup engineer: supporta il project manager ed è responsabile della validazione
  Software librarian: si occupa di controllare la documentazione, i listati di codice e i dati.

- **Che cos'è la WBS(Work Breakdown Structure)-dividi e conquista?**
  Il problema viene risolto tramite la soluzione dei sottosistemi e l'integrazione delle soluzioni. Il WBS definisce quali sono le attività da eseguire e quali sono i rapporti gerarchici tra le attività. Grazie alla decomposizione dell'attività di pianificazione in attività più semplici, i tempi, i costi e gli scopi di realizzazione vengono stimati in modo più preciso.

- **Rapporto mese-uomo**
  Per valutare l'effort di un progetto in funzione del numero di sviluppatori è possibile ricorrere al "mese-uomo". Questa misura è efficace solamente in casi in cui il compito da svolgere è perfettamente partizionabile tra gli sviluppatori e non richiede comunicazione. Per questo motivo difficilmente la si può utilizzare per valutare il numero di sviluppatori richiesti per un task, infatti l'aggiunta di programmatore richiede che questi vengano adeguatamente formati da altri membri del personale, i quali dovranno rinunciare a parte della propria produttività.
- **Cosa sono i diagrammi di Gantt e a cosa servono?**
  Sono uno st5rumento utilizzabile sia in fase di pianificazione che in fase di monitoraggio. Sono dei diagrammi bidimensionali (un'asse per il tempo e un'asse per le attività di progetto)-l'origine degli assi è l'inizio del progetto.
  Permettono di avere tutte le informazioni di sintesi sul progetto e il suo andamento, infatti integrano info sui tempi, vincoli di precedenza nell'esecuzione delle attività, info sull'avanzamento e punti di controllo.
- **Cosa sono i diagrammi di Pert e a cosa servono?**
  I diagrammi di Pert contengono le stesse informazioni dei diagrammi di Gantt, ma si focalizzano sulla durata e sui vincoli di precedenza dell'esecuzione delle attività(scopo: pianificazione delle attività).
- **Algoritmi di trasformazione e partiche di ottimizzazione**
  Per trasformare il processo in piano di progetto inserisco nella prima colonna le attività che non dipendono da nulla e proseguo all'aumentare delle dipendenze.
  Per passare dal piano di progetto al piano esecutivo si ottimizzano le attività che eseguono in parallelo per uomini, tempi(es. analisi cammini critici) e costi(es. riduzione del personale ed out-sourcing(affido l'esecuzione di alcune attività a terzi a costi inferiori)): si valuta se renderle sequenziali(non ci sono abbastanza risorse) o lasciarle in parallelo.
- **Cosa si intende per piano di progetto e piano esecutivo**
  Il piano di progetto pianifica la sequenza potenziale dei compiti da svolgere per completare il processo che tiene conto solamente dei vincoli di precedenza tra i vari tasks, ignorando fattori come la durata di sviluppo e le risorse economiche.(tutte le attività+ piano temporale)
  Il piano esecutivo pianifica la sequenza effettiva delle attività elementari che tiene conto dei fattori economici, delle tempistiche di sviluppo e della gestione del personale.(piano di progetto+ limitazioni del mondo reale)
- **Analisi cammini critici**
  Il cammino critico è la successione di attività legate allo sviluppo del software per cui un ritardo su una qualsiasi delle attività interessate risulta in un ritardo nella consegna.

Per analisi dei cammini critici si intende lo studio della disposizione temporale dei compiti all'interno di un progetto volta ad ottimizzarla per mezzo di una rischedulazione dei compiti che tenga conto della disposizione del personale, delle dipendenze tra le attività (parallelismo) e dei tempi di consegna. Attraverso l'analisi dei cammini critici si può ridurre la durata di sviluppo del progetto.
Uno strumento utile per lo studio dei cammini critici è il diagramma di Pert.
È molto oneroso compiere quest'analisi quando il progetto è già in esecuzione.

CAPITOLO 5: INGEGNERIA DEI REQUISITI

- **Cos'è l'analisi dei requisiti?**
  L'analisi dei requisiti risponde alla domanda: che cosa deve fare l'applicativo?
  L'analisi dei requisiti è necessaria per comprendere precisamente cosa si richiede dal sistema, per comunicare questa comprensione agli sviluppatori e per controllare che il sistema soddisfi le specifiche.
  L'analisi dei requisiti coinvolge sicuramente lo staff tecnico, il committente e coloro che vengono chiamati stakeholders. Gli stakeholders sono tutte quelle persone che traggono vantaggio dal sistema sviluppato, per esempio manager, utenti finali, personale marketing, clienti esterni/interni, consulenti, ingegneri sw etc.
  Generalmente vengono coinvolte 5 figure:
  - Consumatori: spiegano cosa dev'essere consegnato
  - Managers: controllano scheduling e avanzamento del progetto
  - Designers: producono le specifiche di design
  - Programmatori: preparano liste di implementazioni accettabili/output
  - Testers: pensano ad una serie di test per la veridica, validazione e valutazione del progetto.
- **Qual è la differenza tra definizione e specifica dei requisiti?**
  Definizione dei requisiti: Descrizione delle funzionalità del sistema e dei vincoli operativi, orientata al cliente, in linguaggio naturale(ossia il linguaggio del dominio applicativo del cliente). Questo porta a problemi di chiarezza, confusione di requisiti(funzionali e non) e la descrizione di requisiti diversi insieme.
  I requisiti possono essere funzionali(descrivono le funzionalità che il sistema deve fornire) o non funzionali(vincoli sui servizi e funzionalità fornite o vincoli sull'ambiente e sul processo di sviluppo oppure proprietà che il sistema non deve avere). I requisiti non funzionali sono suddivisi in:
  - Requisiti di prodotto, quelli che riguardano il comportamento del prodotto finale(per esempio la velocità di esecuzione o l'affidabilità)
  - Requisiti organizzativi, sono conseguenze delle politiche organizzative del cliente(gli standard di processo per esempio)
  - Requisiti esterni al sistema e all'ambiente di sviluppo(per esempio quelli legati alla legislatura o all'interoperabilità)

  Specifica dei requisiti: documento strutturato e dettagliato dei servizi che il sistema deve fornire; di solito viene utilizzato come base di contratto tra committente e fornitore del sw. Può essere scritta anche con Linguaggi di descrizione di programmi, che è un linguaggio simile a quello di programmazione, ma più espressivo. Di solito si utilizza quando

un'operazione è specificata come sequenza di azioni e l'ordine è quindi importante e quando si devono specificare le interfacce hw e sw.
Specifica del software: una descrizione dettagliata del sw che serve come base per il design dell'architettura e per l'implementazione. E' quindi rivolta agli sviluppatore e agli architetti di sistema.

- **Come sono classificati i requisiti e come ottenerli?**
  Funzionali e non(vedi domanda precedenze-definizione dei requisiti). Si ottengono tramite delle "interviste" agli stakeholders ed analizzandole cercando di trovarne uno scopo. Esistono varie fasi del processo di analisi requisiti.
- **Perché i requisiti possono essere imprecisi?**
  Perché gli stakeholders possono non avere le idee chiare su cosa vogliono, esprimono i requisiti usando il loro linguaggio naturale(che può essere non compreso), ci possono essere requisiti contrastanti e fattori(generalmente politiche interne e di gestione) che influenzano i requisiti del sistema.
- **Come vengono utilizzati i requisiti raccolti?**
- **Quali sono le fasi di processo di ingegneria dei requisiti?**
  L'ingegneria dei requisiti specifica delle caratteristiche operative del sw(funzionalità, dati e comportamenti), indica un'interfaccia sw con gli altri elementi del sistema e stabilisce i vincoli a cui il sistema deve sottostare.
  Le fasi di processo sono:
  1- Studio della fattibilità: quali sono le necessità che il sistema può soddisfare con tecnologia e risorse disponibili.
  2- Analisi dei requisiti:
     - Comprensione del dominio applicativo-> avvio del processo
     - Raccolta dei requisiti, presenta 3 problemi principali, ossia problemi di scope (limiti del sistema mal definiti), di comprensione e di volatilità(evoluzione requisiti nel corso del tempo)-tecnica di raccolta più utilizzata: FAST
     - Classificazione dei requisiti(funzionali e non)
     - Risoluzione dei conflitti fra requisiti
     - Assegnazione delle priorità
     - Validazione dei requisiti, ossia la verifica che siano completi, consistenti e coincidano con i desideri del committente.--> Controllo **validità, consistenza, completezza, realismo, verificabilità, comprensibilità, tracciabilità e adattabilità** dei requisiti.
       Inoltre vengono effettuate delle revisioni dei requisiti periodiche con lo scopo di ottenere un'adeguata comunicazione fra committente, sviluppatore e utenti finali per individuare e risolvere i problemi appena si manifestano.
  3- Definizione dei requisiti: descrizione con linguaggio naturale dei servizi + vincoli.
  4- Specifica dei requisiti
  5- Specifica del software
- **Che cos'è FAST(Facilitated Application SpecificationTecnique)?**
  E' una tecnica che consiste nella formazione di un team misto di sviluppatori e clienti che collabori ad individuare e specificare il problema, proporre una soluzione, negoziare diverse strategie per valutare le soluzioni proposte e specificare un insieme preliminare di

requisiti(elenco consensuale per oggetti, servizi, vincoli e prestazioni + elenco dei criteri di validazione).

- **Cosa si fa in caso di evoluzione dei requisiti e perché evolvono?**
  I requisiti evolvono perché si scoprono le vere esigenze del cliente o perché cambiano gli obiettivi dell'azienda. Esistono requisiti durevoli(derivano dalla natura dell'attività del committente) e volatili(possono cambiare durante lo sviluppo o durante l'uso del sistema). Bisogna poter, quindi, identificare, tracciare e controllare i requisiti e i cambiamenti man mano che di prosegue allo sviluppo del progetto. Per far questo ad ogni requisito viene assegnato un identificatore univoco, si sviluppano delle tabelle di tracciabilità(indicano le relazioni tra requisito e funzionalità, l'origine di ogni requisito e le dipendenze/relazioni con altri requisiti)

CAPITOLO 6: UML 1

CAPITOLO 7: UML 2

CAPITOLO 8: ANALISI DEI REQUISITI

- **Modello di processo VORD(Viewpoint Oriented Requirement Definition)**
  Vieiwpoint: attori coinvolti nel sistema-> produttori/consumatori di dati, utenti/realizzatori di servizi …
  Il modello VORD è formato da 4 fasi:
  1- Individuzione dei viewpoints che ricevono servizi dal sistema e identificazione dei servizi che il sistema deve offrirgli.
  2- Strutturazione dei viewpoints in modo gerarchico
  3- Documentazione, ossia descrizione dei viewpoints e dei servizi identificati
  4- Mapping viewpoint-sistema, ossia la definizione degli oggetti che rappresentano i viewpoint(modello ad oggetti)

CAPITOLO 9: PROGETTAZIONE

- **Cos'è la progettazione?**
  La progettazione è il processo che porta alla definizione ingegneristica di quello che dev'essere realizzato. E' formato da due fasi:
  - Diversificazione: il progettista acquista il materiale grezzo del progetto per individuare le possibilità realizzative
  - Convergenza: il progettista sceglie e combina le componenti per arrivare al prodotto finale
- **Quali sono i 3 requisiti progettuali?**
  1- Il progetto deve soddisfare tutti i requisiti espliciti ed impliciti dettati dal cliente.
  2- Il progetto dev'essere una guida comprensibile e leggibile per chi si occuperà delle fasi di codifica, collaudo e manutenzione,
  3- Deve dare un quadro generale completo e coerente dei domini dei dati e funzionale e comportamentale dell'implementazione.
- **Quali sono le fasi della progettazione?**
  1- Comprensione del problema

1 - Comprensione del problema

2- Identificazione di una o più soluzioni
3- Descrivere astrazioni delle soluzioni→ l'astrazione è la descrizione del sistema in un certo livello trascurando i dettagli dei livelli sottostanti
4- Ripetere lo step per ogni astrazione identificata finché la progettazione non è espressa in termini primitivi.

- **Quali sono le fasi design?**
  - Architectural design: identifica e documenta i sottosistemi e le loro relazioni
  - Abstact Specification: specifica i servizi forniti e i vincoli
  - Interface Design: descriviamo l'interfaccia dei sottosistemi
  - Component design: alloca i servizi ai diversi componenti e definisce le interfacce di questi componenti
  - Data structure design: definire le strutture dati
  - Algorithm design: specifica degli algoritmi utilizzati
- **Quali sono le strategie di modularizzazione?**
  Progettazione top-down: il problema viene partizionato ricorsivamente in sottosistemi e si inizia a risolvere il problema dalla radice al livello più basso
  Progettazione bottom-up: composizione di soluzioni
  Progettazione sandwich: soluzione naturale
- **Come definire una dimensione ottimale dei moduli che minimizza i costi di sviluppo e integrazione?**
  Con i 5 criteri di Mayer(1988):
  1- Scomponibilità: riduzione complessità grazie alla scomposizione di grandi problemi in sottoproblemi
  2- Componibilità: assemblamento di componenti preesistenti per migliorare la produttività
  3- Comprensibilità: minimizza le interfacce di un modulo con altri-> facile costruzione e modifica
  4- Continuità: modifiche a singoli moduli -> facile controllo
  5- Protezione: effetti anomali non si propagano da un modulo ad un altro -> migliora la manutenibilità
- **Cos'è l'architettura del sw, quali sono le sue proprietà e quali modelli vengono utilizzati per presentarla?**
  L'architettura del sw descrive la struttura gerarchica dei moduli di un programma, come interagiscono e la struttura dei dati che manipolano.
  Ha proprietà strutturali(descrive le componenti del sistema e come sono assemblate), extrafunzionali(deve esplicitare il modo in cui vengono soddisfatti i vari requisiti -es prestazioni affidabilità etc) e di affinità(permette il riuso strutture e schemi per progetti simili)
  L'architettura può essere presentata da modelli:
  - Strutturali: mostra l'architettura come collezione organizzata di componenti
  - Dinamici: mostrano gli aspetti comportamentali dell'architettura
  - Schematici: vengono utilizzati per il riuso di schemi ricorrenti
  - Di processo: descrivono il processo tecnico o aziendale che il progetto deve supportare
  - Funzionali: descrivono le funzionalità di un sistema in modo gerarchico.
- **Qual è la struttura dei dati?**

La struttura dati definisce organizzazione, metodi di accesso, grado di associatività e alternative di elaborazione delle informazioni. Esistono varie strutture base cone il vettore sequenziale o lo spazio n-dimensionale o, ancora, la lista.

- **Cos'è l'information hiding e perché/quando viene utilizzato?**
  L'information hiding serve per identificare il minimo costo per la modularità del sistema. Richiede che ogni modulo sia definito in modo che le sue procedure ed informazioni non siano accessibili ad altri moduli. L'unico modo di interazione tra moduli deve avvenire tramite interfaccia.
- **Quali sono le strategie di progettazione?**
  Le strategie di progettazione sono quella funzionale, in cui lo stato del sistema è centralizzato e condiviso tra le funzioni che operano su quello stato e quella orientata agli oggetti, in cui il sistema è visto come un insieme di oggetti che interagiscono(il sitema è decentralizzato ed ogni oggetto ha il proprio stato; gli oggetti comunicano attraverso i metodi di classe).
- **Come si stabilisce la qualità della progettazione?**
  Dipende da specifiche priorità di tipo organizzativo.
  Attributi di mantenibilità di un progetto:
  - Coesione: un modulo esegue un numero di compiti limitato e coerente. Se ci sono cambiamenti al sistema, permette di mantenere il cambiamento locale ad una singola componente.
  - Accoppiamento: misura la forza di connessione tra i componenti di un sistema
  - Comprensibilità: legata a coesione, naming(nomi significativi), documentazione e complessità. Un'elevata complessità = scarsa comprensibilità.
  - Adattabilità: un progetto è adattabile quando le sue componenti sono debolmente accoppiante, è ben documentato, c'è una corrispondenza stretta tra i livelli di progettazione e per ogni componente c'è una forte coesione. L'eredità migliora molto l'adattabilità perché le componenti possono essere adattate senza cambiamenti e modifiche.
- **Quali sono le tipologie e i livelli di coesione?**
  Tipologie e livelli:
  - Coesione incidentale(debole): diverse parti di un componente raggruppate insieme ma non correlate
  - Coesione temporale(debole): raggruppamento componenti attivate nello stesso istante
  - Coesione logica(debole): raggruppamento componenti che svolgono azioni simili
  - Coesione procedurale(debole): raggruppate componenti che vengono attivati in sequenza uno dopo l'altro.
  - Coesione di comunicazione(media): raggruppamento componenti che sullo stesso input effettuano lo stesso output.
  - Coesione sequenziale(media): l'output di un componente è l'input di un altro.
  - Coesione funzionale(forte): ogni parte di componente è necessaria solo per una singola funzione
  - Coesione d'oggetto(forte): ogni operazione fornisce funzionalità per osservare o modificare gli attributi di un oggetto

- **Quali tipi di coupling esistono?**

Accoppiamento lasco: quando i cambiamenti di una componente non hanno effetto sulle altre; può essere ottenuto decentralizzando gli stati e realizzando la comunicazione tramite passaggio di parametri o di messaggi. Accoppiamento stretto: avviene quando le variabili sono condivise e c'è scambio di informazioni di controllo.

  - Accoppiamento basso: moduli diversi si scambiano parametri semplici, non strutture dati
  - Accoppiamento di controllo: un segnale di controllo viene scambiato tra due moduli
  - Accoppiamento comune: moduli diversi hanno dati in comune

- **Come si può migliorare la modularità di un progetto?**
  1- Studia la prima versione del progetto e cerca di ridurre accoppiamento e aumentare la coesione.
  2- Mantenere il campo di azione(insieme dei moduli che dipendono da una decisione presa localmente) di un modulo entro il suo campo di controllo(insieme dei moduli subordinati)
  3- Studiare le interfacce per ridurre l'accoppiamento
  4- Definire moduli con funzioni prevedibili
  5- Evitare collegamenti patologici(interni) tra moduli
- **Che cos'è la specifica di progetto?**
  E' il documento che descrive il progetto finale :
  1- Descrizione della portata globale del progetto ricavata dalla specifica dei requisiti
  2- Descrizione di dati di progetto, es struttura del db o file esterni
  3- Descrizione dell'architettura con riferimento ai metodi utilizzati per ricavarla e rappresentazione grafica dei moduli
  4- Progetto delle interfacce esterne ed interne e descrizione dettagliata delle interazioni fra utente e sistema con eventuale prototipo
  5- Descrizione procedurale dei singoli componenti in linguaggio natuale

CAPITOLO 10: PROGETTAZIONE ARCHITETTURALE

- **Perché la progettazione architetturale è importante?**
  E' importante per 3 motivi:
  - La rappresentazione dell'architettura serve per far comunicare le parti interessate allo sviluppo del sistema
  - L'architettura del sistema mette in evidenza le decisioni progettuali preliminari
  - L'architettura è un modello conciso e facilmente comprensibile di come il modello e i vari componenti interagiranno fra loro.
- **Quali sono i principi di progettazione dei dati?**
  - Analisi di funzionalità e comportamenti
  - Individuazione delle strutture dati e delle operazioni che ognuna deve svolgere
  - Inserire il class-diagram: definisce gli oggetti e le operazioni applicate
  - Raffinazione analisi requisiti(gerarchia delle decisioni e immissione nelle fasi di progettazione)

- Coupling e information hiding per i moduli
- Individuare possibili modularizzazioni e riutilizzi
- Implementare un adt in un linguaggio di programmazione

- **Che cos'è uno stile architetturale?**
  E' composto da un insieme di componenti che implementano le funzionalità richieste, da un insieme di connettori per la comunicazione dei componenti, da un insieme di vincoli per l'interazione fra di essi e da modelli semantici che permettono al progettista di comprendere il funzionamento del sistema sulla base delle sue componenti.
- **Quali sono i modelli di struttura di un sistema?**
  Repository: il sistema è centrato su un archivio dati che può essere attivo(quando notifica al client ogni variazione nei dati) o passivo(quando tutte le operazioni sono effettuate dal client). Le componenti accedono al client operando indipendentemente tra loro ,quindi si può intervenire su un componente senza che gli altri ne risentano. E' molto efficienti per grandi quantità di dati, ma presenta una bassa scalabilità e non c'è spazio vper politiche di accesso dati e modificare il sistema nel suo complessa diventa dispensioso.

  Pipe and Filter: Si tratta di un modello architetturale (tipico delle shell nei sistemi operativi tradizionali) che favorisce la manipolazione di input codice attraverso filtri. È un modello semplice ed efficace per gestire computazioni complesse attraverso la concatenazione di numerosi filtri semplici, tuttavia è poco adatto quando la struttura non è facilmente "linearizzabile"

  Client-server: Il modello client-server si basa sulla richiesta di servizi da parte del client nei confronti del server.

  Uno dei modelli principali è il three-tier architecture (di cui fanno parte i framework MVC), il quale si compone di tre livelli:

  - Presentation layer: si tratta dell'interfaccia grafica attraverso la quale l'utente interagisce con il sistema.
  - Application processing layer: si occupa di fornire le funzionalità del sistema
  - Data management layer: si occupa della comunicazione con il DBMS

  I vantaggi di questo modello risiedono nella sua semplicità ed efficacia in fatto di scalabilità e contenimento dei costi; gli svantaggi sono la necessità di replicare alcune componenti di gestione dei dati tra il DBMS e il Data management layer, e che la mancanza di un modello unico e condiviso può comportare un calo di prestazioni

  Macchina astratta: organizza il sistema in un insieme di strati, ognuno dei quali svolge un insieme di servizi; è usato per modellare l'interfaccia tra i sottosistemi e se si cambia l'interfaccia di un livello, solo quello adiacente ne risente, ma si possono incontrare restrizioni per l'interazione tra i livelli interni ed esterni
- **Qual è la differenza tra sottosistema e modulo?**
  Il sottosistema è un sistema di diritto formato da moduli e interfacce ben definite per comunicare con altri sottosistemi.
  Il modulo è una componente del sottosistema e fornisce uno o più servizi ad altri moduli. Non è indipendente.

- **Cosa si intende con scomposizione modulare, all'interno della fase di progettazione architetturale?**
  Per scomposizione modulare si i intende un raffinamento a livello strutturale per cui i sottosistemi sono scomposti in moduli: ciò comporta una riduzione del tempo di sviluppo, ma richiede che il sistema sia effettivamente partizionabile.
- **Quali sono i modelli di scomposizione modulare?**
  Il modello ad oggetti, dovei il sistema è scomposto in oggetti che interagiscono e i modelli data-flow, dove il sistema è scomposto in modelli funzionali che trasformano input in output (pipeline)
- **Quali sono i modelli di controllo?**
  Descrivono il modo in cui fluisce il controllo tra i sottosistemi:
  → **Modello centralizzato**: esiste un sottosistema che ha il controllo globale e che avvia e disattiva i sottosistemi.
  Modello call-return: gerarchie di procedure, ossia il controllo parte dalla routine iniziale e in base alla gerarchia si sposta verso il basso, utilizzabile su sistemi sequenziali.
  Modello manager: Un componente del sistema controlla l'avvio, l'interruzione e il coordinamento degli altri processi.
  → **Modello bastato su eventi:** dove ogni sistema può intervenire ad eventi generati da altri sottosistemi o dall'ambiente esterno.
  Modello broadcast: un evento viene trasmesso a tutti i sottosistemi e ogni sistema in grado di gestire l'evento può farlo
  Modello interrupt-driven: le interruzioni sono raccolte da un gestore che le passa al componente in grado di processarle

CAPITOLO 11: PROGETTAZIONE OO

- **Che cos'è la progettazione orientata agli oggetti?**
  E' rappresentata come un insieme di oggetti che interagiscono.
  - Classe: caratteristiche comuni di entità che hanno uno stato e un insieme di operazioni che operano sullo stato. Un OGGETTO è un'istanza di una classe.
  - Attributi: rappresentano lo stato di un oggetto
  - Metodi: operazioni definite sull'oggetto che offrono servizi ad altri oggetti
  - Incapsulamento
  - Ereditarietà
  - Specializzazione
  - Polimorfismo
- **Metodo OOD e caratteristiche**
  Esistono 3 fasi di sviluppo orientato agli oggetti:
  - Fase di analisi, che riguarda lo sviluppo del dominio di applicazione
  - Fase di progettazione, che riguarda la progettazione del modello orientato agli oggetti che deve soddisfare i requisiti richiesti
  - Fase di programmazione, che riguarda la realizzazione del progetto attraverso uno specifico linguaggio ad oggetti.

Noi ci troviamo nella fase di progettazione, che indica gli oggetti che sono derivati da ogni classe, come interagiscono fra loro(gerarchia di ereditarietà e aggregazioni), come si implementa il comportamento e come viene implementata la comunicazione tra oggetti.

Gli oggetti sono astrazioni di entità del mondo reale, indipendenti, che contengono informazioni sul proprio stato e sulla rappresentazione dei propri dati. LA comunicazione non avviene con memoria condivisa, ma gli oggetti comunicano tra loro tramite i metodi e passando parametri. Essi possono essere distribuiti ed eseguiti sia in sequenza che in parallelo.

- **Livelli di progettazione**
  - Progettare i sottosistemi
  - Progettare le classi
  - Progettare la comunicazione tra oggetti
  - Progettare le strutture dati e l'implementazione per gli attributi e i metodi di ciascun oggetto
- **Identificazione degli oggetti**
  Esistono vari approcci per l'identificazione degli oggetti come quello grammaticale:
  - I nomi corrispondono ad oggetti del sistema
  - I verbi corrispondono ad operazioni associate ad oggetti
  - I predicati corrispondono ad operazioni che restituiscono valori di tipo booleano

  Esistono anche altri approcci come quello comportamentale(identificazione degli oggetti per il modo in cui si comportano) oppure utilizzare il dominio di applicazione ed individuarne le entità.

- **Tipi di oggetti**
  - Entità interne: che producono o consumano informazione usata dal sistema
  - Entità esterne: parte del dominio informativo del sistema
  - Eventi: occorrenti nelle operazioni del sistema
  - Ruoli: persone che interagiscono con il sistema
  - Luoghi: stabiliscono il contesto del sistema
  - Strutture: costruite dai sistemi di entità correlate
- **OOD secondo Coad e Yourdon**
- **OOD secondo Rambaugh**

CAPITOLO 12: PROGETTAZIONE INTERFACCIA UTENTE

- **Principi della progettazione UI**

  Le tre regole della UI sono:

  1. Il controllo deve essere nelle mani dell'utente: l'utente deve avere un'interazione flessibile e quindi avere sempre la possibilità di interrompere o annullare le operazioni in corso; ciò non significa che all'utente debbano essere fornite informazioni di carattere tecnico
  2. L'utente deve ricorrere il meno possibile all'uso della memoria: (detto sopra)

implementazione applicazioni stesse regole
contesto in cui si trova l'utente devrebbe dare

L'interfaccia deve essere uniforme: il sistema deve avere un aspetto grafico consistente in ogni suo elemento; è inoltre opportuno che si rispettino canoni stilistici e di usabilità assodati, così come è importante che si mantengano modelli interattivi preesistenti → non apporre modifiche ai modelli già consolidati

- **Limiti del ricorso alla memoria**
  Affinché si possa raggiungere questo scopo, è importante che l'interfaccia grafica a cui è sottoposto l'utente sia di facile lettura e conforme a standard assodati in termini di estetica e usabilità, in maniera tale che non si tratti di un processo di apprendimento, bensì di riconoscimento. Inoltre l'UI dovrebbe sempre fornire all'utente informazioni circa le operazioni svolte in precedenza e in maniera progressiva. Un'altra ragione per ridurre la mode di informazioni che l'utente deve ricordare è che ad una maggiore memorizzazione corrisponde una maggiore propensione a commettere errori nelle interazioni con il sistema.
- **Modelli di utente**

  Esistono 3 tipi di utente:

  - Principiante: nessuna conoscenza sintattica(uso dell'interfaccia) e scarse conoscenze semantiche sull'uso del sistema(comprensione delle funzionalità)
  - Casuale: ragionevole conoscenza semantica dell'applicazione, mentre la conoscenza sintattica è limitata
  - Costante: buona conoscenza semantica e sintattica
- **Immagine e percezione**
  L'immagine di sistema comprende sia l'aspetto e il comportamento dell'interfaccia utente sia tutte le informazioni di supporto che descrivono sintassi e semantica del sistema.
  La percezione del sistema, invece, è l'immagine mentale che si forma nella mente dell'utente finale.
  Generalmente, si cerca di fare in modo che coincidano per soddisfare il cliente.
- **Processo progettazione UI**
  Il processo di progettazione e analisi è iterativo e segue un modello di sviluppo a spirale in cui troviamo 4 fasi:
  1- Analisi e modellazione degli utenti, delle operazioni e dell'ambiente:
  dobbiamo comprendere quali sono il tipo di utenti coinvolti la loro disponibilità nell'accettare il sistema, per ogni utente bisognerà, poi, individuare i requisiti dell'interfaccia e l'ambiente in cui verrà usata l'interfaccia, sia a livello fisico(luve, rumore, zona geografica etc), sia a livello comportamentale(in piedi/seduto, richieste ergonomiche etc).
  Per quanto riguarda l'analisi e la modellazione di operazione si effettuano varie fasi quali:
  - Utilizzo di use case nell'analisi per mostrare come l'utente svolge operazioni specifiche; da questi use case sono estratte le operazione, gli oggetti e il flusso generale delle interazioni.
  - Elaborazione delle operazioni tramite traduzioni di attività manuali oppure tramite lo studio delle specifiche di una soluzione computerizzata;
    generalmente per fare ciò si parte dallo use case e si estraggono gli oggetti per catalogarli in classi e definire ogni attributo di classe dalle azioni che si fanno sugli oggetti.

    - Analisi del workflow(sequenza di operazioni di lavoro), che viene effettuata se più utenti hanno ruoli differenti, per chiarirli tramite gli activity diagram.
    - Rappresentazione gerarchica, per associare le operazioni ad ogni ruolo.
  2- Progettazione dell'interfaccia
  3- Costruzione o implementazione dell'interfaccia
  4- Validazione dell'interfaccia
- **Design UI e problemi relativi**
  1- Il tempo di risposta dev'essere né troppo lungo né troppo corto e dovrebbe esserci una bassa variabilità
  2- Il sistema di help dell'utente non è disponibile per ogni funzione e in ogni fase dell'interazione
  3- I messaggi di errore dovrebbero descrivere il problema in un linguaggio comprensibile all'utente, fornire info per risolvere il problema e sottolineare quali effetti negativi ha portato/può portare l'errore
  4- Poche interfacce inseriscono l'uso della riga di comando, che viene solitamente richiesta da un utente esperto
  5- L'accessibilità è sempre più richiesta
  6- Spesso le interfacce sono pensate e progettate in una specifica lingua e poi adattate a tutte le altre, questo provoca un problema di internazionalizzazione
- **Presentazione delle informazioni**
- **Strumenti di implementazione**
  Sono strumenti che permettono la creazione di un prototipo di interfaccia.
- **Valutazione del prototipo**
  Esistono vari metodi per valutare un prototipo, quali il metodo informale, in cui un utente usa il sistema e offre le proprie considerazioni e quello formale, in cui gruppi di utenti usano il sistema e vengono utilizzati sistemi statistici per valutare le loro opinioni.
  Abbiamo vari indici di valutazione su cui basarci:
  - Difficoltà di apprendimento degli utenti: specifiche dei requisiti molto lunghe
  - Tempo di interazione e efficienza: il numero di operazioni degli utenti e il numero di azioni per ogni operazione fatta dall'utente.
  - Quanto l'utente ricorre alla memoria è proporzionale al numero di azioni e operazioni effettuato
  - La complessità dell'UI e del livello di accettazione da parte degli utenti dal modo di gestione degli errori e dall'utilizzo dell'help.

CAPITOLO 13: VERIFICA E VALIDAZIONE

Per validazione si intende il processo che risponde alla domanda "si sta costruendo il progetto giusto?", ossia si chiede se si sta implementando il prodotto in modo da soddisfare le richieste dei clienti. La validazione può essere solo dinamica, in quanto c'è la necessità di un prototipo per eseguire dei test di prova e osservare il comportamento.

La verifica, invece, risponde alla domanda "si sta costruendo il progetto nel modo giusto?", ossia si chiede se il prodotto è conforme alle specifiche imposte. Essa può essere statica, si analizza la rappresentazione statica del sistema per individuare problemi o dinamica, ossia si testa il prodotto con varie prove per vedere come risponde.

Principi di testing

1- Ogni singola prova dev'essere riconducibile ai requisiti del cliente
2- I collaudi vanno pianificati con largo anticipo: possono iniziare dopo la specifica dei requisiti(statici) e dopo la prototipazione(dinamici)
3- Principio di pareto: 80% degli errori è riconducibile al 20% dei componenti del programma, quindi serve isolare i componenti sospetti
4- I collaudi vengono effettuati prima sulle componenti e poi sull'intero sistema
5- E' impossibile effettuare collaudi esaurienti perché sono troppe tutte le possibili esecuzioni del programma
6- Il collaudo andrebbe effettuato da una persona esterna a chi scrive il codice

Che cos'è la collaudabilità?

La facilità con cui un test può essere provato. Dipende **dall'operabilità**, ossia da pochi errori che non impediscono l'esecuzione del collaudo, **dall'osservabilità**, ossia da input e output correlati e chiari, con visualizzazione degli stati e delle variabili durante l'esecuzione del collaudo e quando gli errori vengono subito identificati, **controllabilità**, tutto l'output può essere generalizzato ad un opportuno input, **scomponibilità**, ossia i moduli, essendo indipendenti, possono essere collaudati singolarmente, **semplicità funzionale**(funzionalità minime ed essenziali), **strutturale**(architettura scomponibile in moduli), **del codice**(standard unico di codifica), **stabilità**, meno modifiche si effettuano, minor volte devono essere eseguiti i collaudi e **comprensibilità** (chiarezza del progetto, dipendenze ben descritte e documenti e modifiche dettagliate e pubbliche).

Esistono due tipi di collaudo BLACK-BOX e WHITE-BOX.

WHITE-BOX: si intende il collaudo del software di cui si conosce il codice e per cui si controllano tutti i vari cammini indipendenti possibili interni ai moduli. Questo tipo di collaudo è estensivo ed è ad opera degli sviluppatori. Si studiano gli aspetti procedurali del programma piuttosto che mancanze dei requisiti.

BLACK-BOX: si intende il collaudo del software di cui non si conosce il codice, e dunque si valuta la conformità delle specifiche indicate nella specifica dei requisiti (disinteressandosi quindi della struttura interna). Si ricercano funzioni errate o mancanti, errori di interfaccia, nelle strutture dati o comportamenti erronei o problemi prestazionali, oppure errori nelle fasi di inizializzazione e terminazione. È complementare al collaudo white-box

Che cos'è la complessità ciclomatica? È la misura quantitativa della complessità logica di un sistema, ossia il numero di cammini indipendenti in un grafo.

Qual è la differenza fra testing e debugging?

Il testing viene utilizzato per confermare la presenza di errori, mentre il debugging viene utilizzato per localizzare risolvere gli errori identificati.

Le diverse strategie di testing sono:

- TOP-DOWN: si testa il modulo radice sostituendo quelli di livello inferiore con stub(versione semplificata dei moduli dei livelli inferiori) e man mano si includono i moduli superiori testati.

- BOTTOM-UP: si parte dal livello più basso sviluppando dei driver che usano e verificano il comportamento del sw fino alla radice, aggiungendo livelli. (appropriata per il linguaggio ad oggetti)
- COLLAUDO PER REGRESSIONE(incrementale): ad ogni introduzione di modulo vengono rieffettuate tutte le prove di testing per verificare che le modifiche/introduzioni di nuovi moduli/correzioni di bug non abbiamo portato effetti collaterali.
- STRESS TESTING: testa le prestazioni con carichi molto alti.
- BACK-TOBACK TESTING: testa le varie versioni del programma con lo stesso input e confronta i vari output, se sono diversi ci sono errori potenziali. Permette di ridurre i costi di esaminare il risultato per l'automatizzazione dei test di confronto.
  Si usa quando c'è una nuova versione di un sist4ema oppure quando un sistema viene sviluppato in più versioni.

Perché è difficile trovare le cause di un errore?

- Per la distanza tra la causa e il punto in cui si presenta l'errore
- Non c'è una causa specifica
- Può dipendere da un errore umano
- Può dipendere da problemi di tempo e non di elaborazione
- L'intermittenza di errore
- La dipendenza da cause distribuite su più processi

Revisione vs walkthrough

La revisione di progetto consiste in riunioni in cui si riesamina il codice a seguito dello sviluppo del processo, ricorrendo alla compilazione di checklists di errori comuni.

Il walkthrough, che è un processo meno rapido rispetto alla revisione, consiste in una lettura critica del codice con l'intento di simularne l'esecuzione; si tratta di un processo collaborativo che si basa sull'esperienza degli sviluppatori.

CAPITOLO 14: AFFIDABILITA'

Per affidabilità si intende la probabilità che il sistema funzioni senza errori per un dato intervallo di tempo, in un dato ambiente e per un determinato scopo. È una proprietà molto importante di ogni sistema, ma è molto difficile e costoso ottenere un'affidabilità tale da garantire la correttezza di ogni output per ogni possibile input, pertanto è più importante nella maggior parte dei casi diminuire il più possibile la possibilità che si verifichino fallimenti gravi piuttosto e implementare funzioni di recupero da fallimenti transienti(gli errori si verificano solo per determinati input) piuttosto che applicare una politica di fault avoidance(evitare guasti). Inoltre, molto spesso gli sforzi per l'aumento dell'affidabilità comportano un costo in prestazioni ed efficienza non indifferenti, senza avere la garanzia di non aggiungere errori non previsti.

Esistono numerosi metodi per misurare l'affidabilità:

- POFOD: misura la probabilità che si verifichi un fallimento a seguito di un input
- ROCOF(Tasso di occorrenza dei fallimenti) e MTTF: misura il tempo medio tra fallimenti

- AVAIL: misura il tempo necessario per ripristinare l'operatività del sistema a seguito di un fallimento

Alcuni passi per migliorare l'affidabilità del prodotto finito è applicare nelle fasi di progettazione e di sviluppo sani principi di buona programmazione strutturata, per cui evitare goto e fare un uso ragionato dell'ereditarietà e dell'allocazione in memoria.

DESIGN PATTERN:

- **Che cosa sono e quali sono i vantaggi e svantaggi?**

  I design pattern sono soluzioni consolidate (convalidata dal suo utilizzo con successo in più progetti) e accettate per un problema ricorrente in un determinato contesto.

  Possono essere strutturali (si concentrano come classi e oggetti e si combinano per formare strutture più grandi), comportamentali(identificano metodi di comunicazione tra oggetti e li realizzano) o creazionali (astraggono il processo di creazione di oggetti, riducono l'accoppiamento e aumentano la flessibilità e nascondono la conoscenza dei dettagli di creazione e composizione degli oggetti)

  La progettazione di software che segue design pattern favorisce il "concept reuse" e permette la scrittura di codice chiuso alle modifiche e aperto alle estensioni

  Vantaggi:

  - Permettono il riuso di architetture sw su larga scala
  - Aiutano a documentare e comprendere il sistema

  Svantaggi:

  - I pattern non portano al riuso diretto del codice
  - I pattern possono essere ingannevolmente semplici
  - C'è la possibilità di utilizzarne troppi
  - Non sono validati da un testing esaustivo, ma dall'esperienza e dal confronto
  - L'integrazione dei pattern in un processo di sviluppo di un sw richiede un'intensa attività umana