1 - INTRODUZIONE

I software sono astratti e intangibili, non sono vincolati da materiali, processi di produzione o da leggi fisiche, ciò comporta una semplificazione dell'ingegneria del software poiché non impone limitazioni al potenziale del software. D'altra parte la mancanza di vincoli naturali indica che il software può facilmente diventare complesso, difficile da capire e costoso da modificare.

Molti dei "fallimenti del software" odierni sono dovuti a due fattori:

- Complessità crescente dei sistemi
- Fallimento nell'utilizzo dei metodi di ingegneria del software

Sviluppo professionale

Vi sono delle differenze tra il software individuale e quello professionale, quest'ultimo è destinato ad essere utilizzato da persone diverse dagli sviluppatori, inoltre è sviluppato da un team e non da una singola persona. L'ingegneria del software ha come scopo quello di supportare lo sviluppo di software professionale, anziché la programmazione individuale. Quando si parla di ingegneria del software non si fa riferimento solo ai programmi ma anche a tutta la documentazione associata: documentazione di sistema (descrive la struttura del sistema), documentazione per gli utenti (spiega come utilizzare il sistema), le librerie, i siti web di supporto e i dati di configurazione.

Esistono 2 tipi fondamentali di software:

- *Prodotti generici*: sistemi autosufficienti e disponibili sul mercato per i clienti che intendono acquistarli, questo tipo di software include anche applicazioni "verticali" progettate per uno specifico mercato.
- *Prodotti personalizzati* (o *su richiesta*): sistemi commissionati a un fornitore di software da uno specifico cliente.

Una differenza sostanziale tra questi due sta nel fatto che nei prodotti generici l'organizzazione che sviluppa il software controlla le sue specifiche mentre per i prodotti personalizzati gli sviluppatori devono attenersi alle specifiche del cliente. Sono in forte crescita i sistemi "ibridi", ne sono un esempio i sistemi ERP (Enterprise Resource Planning, pianificazione delle risorse aziendali), in cui un sistema grande e complesso viene adattao a una società.

Qui sotto una figura che riassume i caratteri qualitativi di un sistema software

Caratteristica del prodotto	Descrizione
Accettabilità	Il software deve essere accettabile per il tipo di utenti per i quali è progettato. Questo significa che deve essere comprensibile, usabile e compatibile con gli altri sistemi che essi usano.
Fidatezza e protezione	La fidatezza del software include varie caratteristiche, quali l'affidabilità, la protezione e la sicurezza. Un software fidato non dovrebbe causare danni fisici o economici nel caso di malfunzionamento del sistema. Il software deve essere protetto in modo tale che utenti malintenzionati non possano accedere o danneggiare il sistema.
Efficienza	Il software non dovrebbe fare un uso dispendioso delle risorse del sistema, come la memoria o i cicli di processore. L'efficienza include la risposta alle sollecitazioni, i tempi di elaborazione, l'uso della memoria ecc.
Mantenibilità	Il software dovrebbe essere scritto in modo da evolversi in rapporto alle nuove richieste dei clienti. Questo è un attributo critico perché la modifica del software è una conseguenza inevitabile della variabilità dell'ambiente aziendale.

Ingegneria del software

L'ingegneria del software è una disciplina ingegneristica che riguarda tutti gli aspetti della produzione software, dalle prime fasi della specifica alla manutenzione del sistema dopo che è entrato in uso. L'ingegneria aiuta a ottenere i risultati con la qualità richiesta ed entro i tempi e i costi prestabiliti. L'approccio sistematico che viene utilizzato viene detto anche **processo software**, ossia un insieme di attività che porta alla produzione di un prodotto software. Quattro attività fondamentali sono comuni a tutti i processi software:

1. *Specifiche del software*: clienti e ingegneri definiscono le funzionalità e i vincoli operativi del software da produrre.
2. *Sviluppo del software*: viene progettato e pianificato il software.
3. *Convalida del software*: il software è convalidato per garantire ciò che il cliente ha richiesto.
4. *Evoluzione del software*: il software viene modificato per soddisfare eventuali cambiamenti dei requisiti del cliente e del mercato.

L'ingegneria del software è correlata sia all'informatica sia all'ingegneria dei sistemi: la prima si occupa delle teorie e dei metodi alla base dei sistemi software (l'ingegneria del software si occupa dei problemi pratici relativi alla produzione del software), la seconda si occupa di tutti gli aspetti dello sviluppo e dell'evoluzione dei sistemi complessi dove il software gioca un ruolo chiave. Non esistono tecniche o metodi universali di ingegneria del software, tuttavia ci sono 4 aspetti correlati che influiscono su molti tipi di software differenti:

1. *Diversità* (o *eterogeneità*): il software deve poter essere eseguito, oltre che su un computer, anche su cellulari e tablet.
2. *Variabilità*: bisogna essere capaci di modificare il software esistente e svilupparne di nuovo nel minor tempo possibile.
3. *Fiducia e protezione*: occorre garantire che il software non possa essere attaccato da utenti malintenzionati e che sia sempre possibile assicurare la protezione delle informazioni.
4. *Scala*: il software deve essere sviluppato per un'ampia scala di sistemi.

Diversità nell'ingegneria del software

Il fattore più significativo per determinare quali tecniche e metodi siano più efficaci è il tipo di applicazione che si vuole sviluppare. Ci sono vari tipi di applicazioni differenti:

- *Applicazioni autonome*: applicazioni che possono essere eseguite su un PC o un dispositivo mobile, includono tutte le funzionalità necessarie e potrebbero non richiedere l'accesso a internet.
- *Applicazioni interattive basate sulle transazioni*: sono applicazioni che vengono eseguite su un computer remoto, spesso incorporano un grosso database che viene consultato e aggiornato a ogni transazione.
- *Sistemi di controllo integrati*: sistemi che controllano e gestiscono i dispositivi hardware.
- *Sistemi di elaborazione batch*: sistemi aziendali progettati per elaborare grandi blocchi di dati in sequenza.
- *Sistemi di intrattenimento*: sistemi per uso personale progettati per il divertimento dell'utente.
- *Sistemi per la modellazione e la simulazione*: sviluppati da scienziati e ingegneria per creare modelli di situazioni o processi fisici, che includono molti oggetti distinti che interagiscono tra di loro.
- *Sistemi per la raccolta e l'analisi dei dati*: raccolgono i dati dai loro ambienti e li inviano ad altri sistemi per l'elaborazione, il software può dover interagire con sensori.
- *Sistemi di sistemi*: sistemi composti da un certo numero di altri sistemi software.

Nonostante ciascun tipo di sistema richieda tecniche di ingegneria del software specializzate (poiché il software ha caratteristiche differenti) e quindi non esistono delle tecniche generali, esistono dei concetti fondamentali che si applicano a tutti i tipi di sistemi software:

1. I sistemi devono essere sviluppati utilizzando un processo di sviluppo chiaro e accuratamente pianificato.
2. La fidatezza e le prestazioni sono importanti per ogni tipo di sistema.
3. È importante capire e gestire la specifica e i requisiti del software (cosa deve fare il software).
4. Occorre utilizzare con efficienza le risorse esistenti (ove possibile riusare il software già sviluppato).

Ingegneria del software di Internet

L'avvento del Web ha portato a un drastico cambiamento del modo in cui viene organizzato il software aziendale, ora il software è molto distribuito (a volte in tutto il mondo), le applicazioni aziendali riutilizzano software e componenti già esistenti. Questo cambiamento nell'organizzazione del software ha avuto un significativo effetto sull'ingegneria del software per i sistemi basati sul Web. Per esempio:

- Il *riutilizzo di software* è diventato l'approccio dominante per costruire sistemi basati sul Web.
- *Sviluppo incrementale* e *agile* del software.
- Sistemi *orientati ai servizi.*
- *Interfacce ricche*.

Etica dell'ingegneria del software

Come le altre discipline ingegneristiche, l'ingegneria del software comprende responsabilità molto più ampie della semplice applicazione di capacità tecniche; gli ingegneri del software devono agire in maniera eticamente e moralmente responsabile se vogliono essere rispettati in quanto professionisti. Un comportamento etico va oltre il semplice seguire le leggi ma comprende che si seguano un insieme di principi moralmente corretti, legati più alla responsabilità del singolo individuo:

- *Riservatezza*: occorre rispettare la riservatezza dei propri datori di lavoro e clienti, a prescindere se sia stato firmato o meno un contratto;
- *Competenza*: non bisogna mentire sul proprio livello di competenza, né accettare lavori al di sopra delle proprie capacità;
- *Diritti di proprietà intellettuale*: è necessario conoscere le leggi locali che governano l'uso delle proprietà intellettuali, come brevetti e copyright, in modo da tutelare quelle del proprio datore di lavoro e dei clienti;
- *Uso improprio del computer*: non bisogna sfruttare le proprie capacità tecniche per un cattivo uso dei computer altrui, che sia banale (giocare con il computer di lavoro) o molto serio (disseminazione di virus);

Le società professionali negli Stati Uniti hanno cooperato per produrre un codice di condotta etico unico che viene sottoscritto dagli associati al momento dell'iscrizione. Questo "Codice" unificato sviluppato da ACM e IEEE, contiene 8 principi legati al comportamento e alle decisioni che vengono prese da sviluppatori software professionisti (ossia coloro che svolgono ruoli direttamente legati alla professione), ma anche educatori, manager, supervisori, decisori di alto livello, così come gli apprendisti e gli studenti della professione. I Principi identificano le relazioni significative dal punto di vista etico nelle quali posso prendere parti gl individui, i gruppi e le organizzazioni, e gli obblighi principali nell'ambito di tali relazioni.

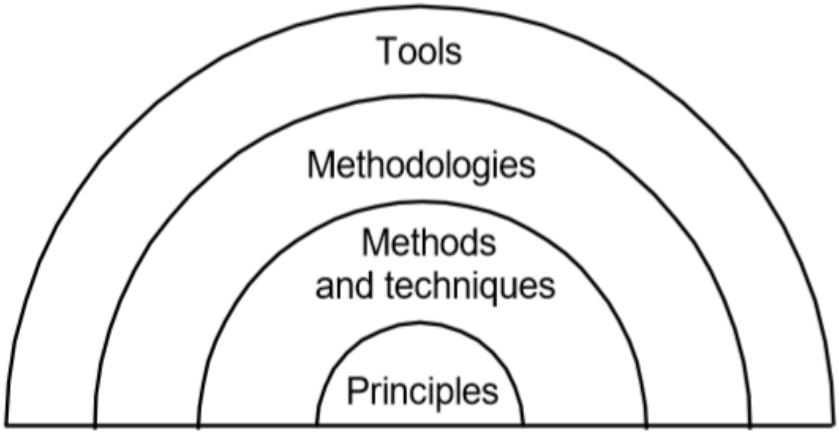
"Data l'importanza fondamentale degli sviluppatori nell'ambito dello sviluppo software, hanno opportunità significativa di fare del bene o di causare danni e di influenzare altri a fare altrettanto. Per assicurare, quanto possibile, che i loro sforzi verranno usati a fin di bene, gli sviluppatori, devono impegnarsi a rendere lo sviluppo software una professione rispettata e dagli effetti benefici. In accordo con questo impegno, gli sviluppatori software devono aderire al seguente Codice Etico e di pratica professionale."

Gli 8 Principi sono:

1. **Pubblico,** gli sviluppatori software devono agire in linea con l'interesse del pubblico.
2. **Cliente e datore di lavoro,** gli sviluppatori software devono agire in un modo conforme agli interessi del loro cliente e datore di lavoro, restando in accordo con l'interesse pubblico.
3. **Prodotto,** gli sviluppatori software devono assicurare che i loro prodotti e le modifiche che vi applicano siano al livello di standard professionale più elevato possibile.
4. **Giudizio,** gli sviluppatori software devono mantenere integrità e indipendenza nel loro giudizio professionale.
5. **Management,** manager e leader degli sviluppatori software devono sottoscrivere e promuovere un approccio etico al management dello sviluppo e della manutenzione del software.

6. **Professione,** gli sviluppatori software devono far progredire l'integrità e la reputazione della professione, restando in accordo con l'interesse pubblico.
7. **Colleghi,** gli sviluppatori software devono essere leali e di supporto nei confronti dei loro colleghi.
8. **Se stessi,** gli sviluppatori software devono, per tutta la durata della loro attività lavorativa, continuare la propria formazione sulla pratica della professione, e devono promuovere un approccio etico a essa.

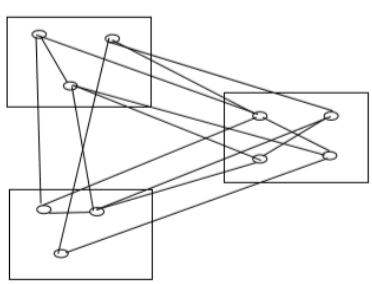
PRINCIPI dell'INGEGNERIA del SOFTWARE



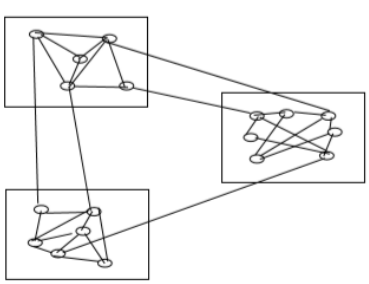
Nell'Ingegneria del Software vi sono dei principi di base per metodi, tecniche, metodologie e strumenti, questi 7 principi sono usabili in ogni fase dello sviluppo software. I principi sono applicati a processo e prodotto ma sono messi in pratica tramite metodi e tecniche: organizzati in metodologie e supportati da strumenti (tool).

Questi 7 principi chiave sono:

- *Rigore* e *formalità*: l'ingegneria del software è un'attività creativa ma deve essere esercitata sistematicamente, il Rigore è un complemento alla creatività (aumenta la fiducia in passi sviluppati) mentre la Formalità è un grado più alto di rigore (è un processo guidato e valutato con leggi matematiche), esempi:
  - Processo -> documentazione rigorosa dei passi di sviluppo per: gestione del progetto e valutazione tempestività (timeliness)
  - Prodotto -> Analisi matematica (formale) correttezza dei programmi – Derivazione sistematica (rigorosa) dati di test
  - Compilatore -> la costruzione di compilatori è area con metodi di progetto sistematico (formali)
- *Separation of concerns*: è un metodo per controllare la complessità, si separano le tematiche per concentrarsi su una sola di esse alla volta (Divide et impera), favorisce la parallelizzazione degli sforzi e la separazione delle responsabilità, esempi:
  - Processo -> procedere una fase alla volta (modello a cascata)permette di separare le attività nel tempo mentre stabilire un incremento (modelli incrementali o agili) indica su quali problemi concentrarsi
  - Prodotto -> mantenere i requisiti di prodotto separati: funzionalità, prestazioni, interfaccia utente e usabilità
  - Compilatore -> progettazione di algoritmi con allocazione ottimale in registri vs progettazione di messaggi diagnostici (runtime efficiency separata da user friendliness)
- *Modularità*: un sistema complesso viene suddiviso in parti più semplici detti *moduli* (sistema modulare)e favorisce la separation of concerns, in questo modo, mentre trattiamo un modulo, possiamo non curarci dei dettagli degli altri. Un concetto importante è la differenza tra *coesione* e *accoppiamento*, ogni modulo è altamente coeso (i componenti di un modulo sono strettamente correlati e possono essere interpretati come un'unica unità significativa) ma i moduli sono a basso accoppiamento (hanno bassa interazione tra di loro e possono essere compresi



Alto accoppiamento



Basso accoppiamento

separatamente) esempi:
    - Compilatore -> processo decomposto in fasi (analisi lessicale, analisi sintattica e generazione del codice) e ad ogni fase è associata un modulo
- *Astrazione*: è fondamentale identificare gli aspetti fondamentali di un fenomeno ignorando i dettagli, produce dei modelli, esempi:
    - Prodotto -> la semantica del linguaggio di una macchina astratta ignora i dettagli delle macchine reali per l'implementazione
    - Processo -> la stima dei costi è basata solo su fattori chiave, considera similarità con i sistemi recedenti, ignorando le differenze di dettaglio
    - Compilatore -> la sintassi astratta trascura i dettagli sintattici (begin … end vs {} per blocchi di istruzioni)
- *Anticipazione del cambiamento*: la capacità di supportare l'evoluzione del software richiede di anticipare i possibili cambiamenti futuri, è la base per l'evolvibilità del del software, esempi:
    - Compilatore -> si considerano i possibili cambiamenti di: linguaggi sorgente, processore bersaglio e dispositivi di I/O
- *Generalità*: spesso risulta più semplice risolvere un caso generale, quando si ha un problema si cerca di capire se è un caso particolare di un problema più generale e si procede alla risoluzione di quest'ultimo (la soluzione del caso generale inoltre è riusabile), vi è un bilancio tra generalità prestazioni/costi, esempi:
    - Compilatore -> parametrizzare rispetto a una macchina bersaglio (codice intermedio), si sviluppano strumenti per la generazione di compilatori e non di un compilatore singolo
- *Incrementalità*: un processo avanza a passi discreti detti *incrementi*, esempi:
    - Compilatore -> si sviluppa una versione base per un sottoinsieme di un linguaggio sorgente, andando a sviluppare sottoinsiemi progressivamente più grandi, consegnandoli con delle diagnostiche/ottimizzazioni limitate per consentire un aggiunta successiva

2 - PROCESSI SOFTWARE

Un processo software è un insieme strutturato di attività che porta alla creazione di un prodotto software, esistono vari tipi di sistemi software e, poiché non esiste un metodo unico di ingegneria del software, non esiste un processo universalmente applicabile. Benché ci siano vari processi tutti devono includere le 4 attività fondamentali:

1. *Specifica del software*: descrizione formale dei requisiti del committente;
2. *Sviluppo del software*: organizzazione e implementazione del software;
3. *Convalida del software*: si utilizzano vari metodi per verificare che il sistema faccia quanto previsto;
4. *Evoluzione del software*: il software deve essere in grado di evolversi per soddisfare le necessità e i cambiamenti richiesti dall'utente, dipende dai cambiamenti dell'esterno e dai cambiamenti delle esigenze.

Quando descriviamo e analizziamo i processi, parliamo delle attività svolte in questi; tuttavia quando li descriviamo è anche importante descrivere le persone coinvolte, cosa viene prodotto e le condizioni che influiscono sulla sequenza delle attività:

- Prodotti o consegne sono il risultato di un'attività di processo;
- I ruoli riflettono le responsabilità delle persone coinvolte nel processo;
- Pre e Postcondizioni sono le asserzioni che devono essere soddisfatte prima e dopo che un'attività di un processo è stata svolta o che un prodotto è stato realizzato.

I processi possono essere fondamentalmente di 2 diversi tipi: **Plan-driven** o **Agile**; i primi sono processi in cui ogni attività è pianificata in anticipo e il loro avanzamento è misurato rispetto a quanto previsto dal piano, i secondi prevedono una pianificazione incrementale e continua durante lo sviluppo del software (consente di gestire meglio i cambiamenti del committente). In generale per grandi sistemi occorre trovare una giusta via di mezzo tra questi due, partendo dal fatto che non esistono processi software giusti o sbagliati.

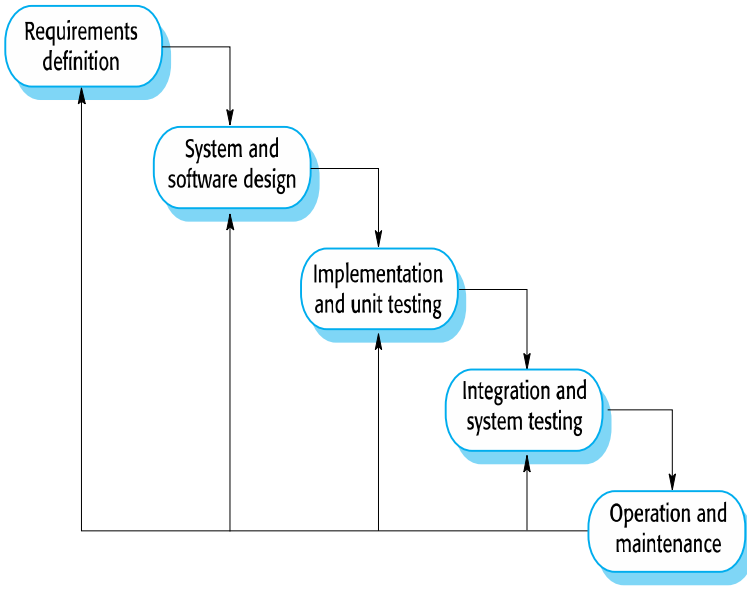
Modelli dei processi software

I modelli di processo sono 3:

- **Modello a cascata** *(plan-driven)*: le fasi di specifica e sviluppo sono separate e distinte;
- **Sviluppo incrementale** *(plan-driven o agile)*: specifica,sviluppo e validazione sono intrecciate;
- **Integrazione e configurazione** *(plan-driven o agile)*: il sistema è assemblato tramite componenti/sistemi riutilizzabili che vengono configurati per rispondere alle nuove esigenze.

Modello a cascata

Le fasi in questo modello sono distinte e ben separate, infatti la problematica maggiore di questo modello sono i cambiamenti poiché si può passare alla fase successiva solo una volta finita la precedente. I principali stadi del modello a cascata riflettono direttamente le attività di sviluppo fondamentali del software:



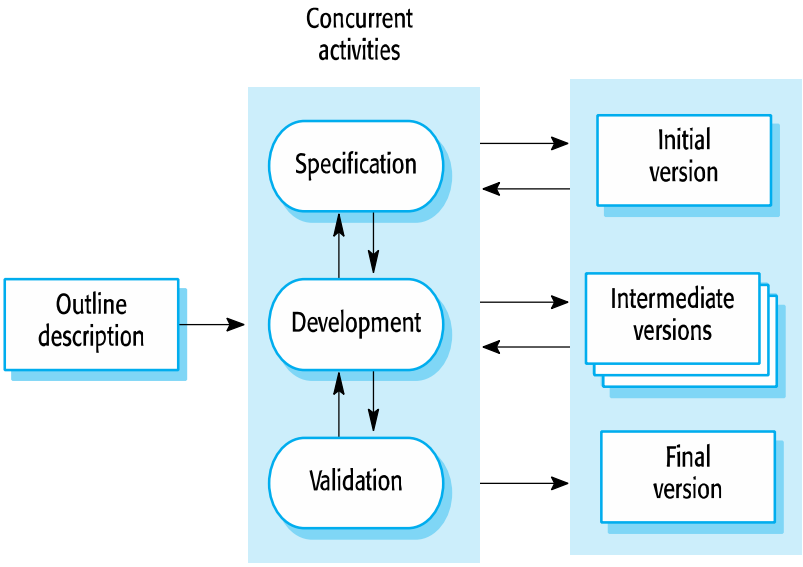
- *Analisi e definizione dei requisiti*: si determinano i servizi, vincoli e obiettivi del sistema;
- *Progettazione del sistema e del software*: si suddividono i requisiti sul sistema hardware o software e si stabilisce l'architettura generale del sistema;
- *Implementazione e test delle unità*: il progetto del software è realizzato come un insieme di programmi o unità del programma;
- *Integrazione e test del sistema*: si accorpano le unità e si testano come un sistema unico per accertarsi che i requisiti del software siano soddisfatti;
- *Operatività e manutenzione*: il sistema è installato, messo in opera e si correggono altri eventuali errori non emersi nei primi stadi del ciclo.

Questo modello presenta alcuni problemi primo trai quali la non flessibilità, il fatto di dover stabilire inizialmente gli obiettivi e di dover rilavorare il sistema quando vengono apportate le modifiche implica che il modello a cascata è appropriato solo per alcuni tipi di sistema (grandi sistemi che fanno parte di sistemi software più complessi, i sistemi integrati e i sistemi critici).

Sviluppo incrementale

Questo modello si basa sull'idea di sviluppare un'implementazione iniziale, esporla agli utenti e perfezionarla attraverso molte versioni, finché non si ottiene il sistema richiesto, questo porta a 3 vantaggi principali:

1. Il costo dell'implementazione delle modifiche dei requisiti è ridotto, l'analisi e la documentazione che devono essere ripetute sono significativamente minori rispetto al precedente modello;
2. È più facile ottenere il feedback del cliente sul lavoro di sviluppo, egli può fare commenti durante le dimostrazioni del software e vedere quanto è già stato realizzato;
3. È possibile consegnare in anticipo al cliente una versione utilizzabile del



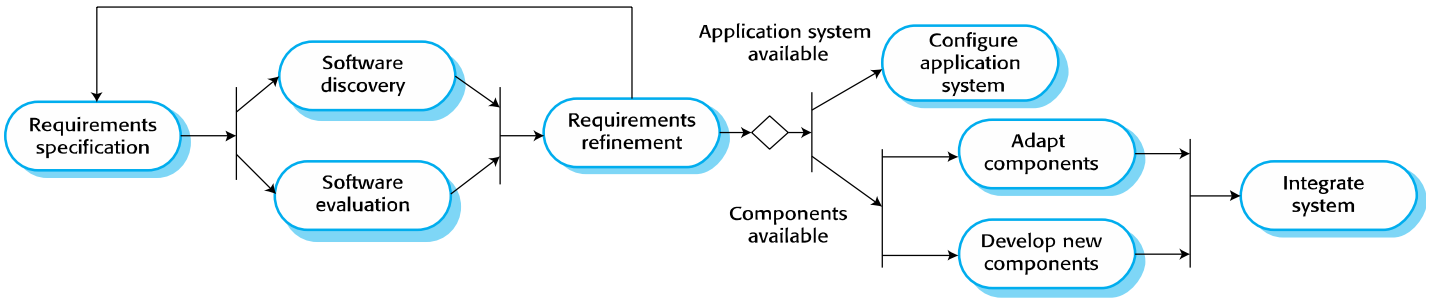
software, anche se non include tutte le funzionalità.

Dal punto di vista gestionale presenta però 2 problemi:

1. *Il processo non è visibile*, manager devono avere delle consegne regolari per misurare i progressi, se il sistema è sviluppato velocemente non è economico produrre una documentazione che rifletta ogni versione del sistema;
2. *La struttura dei sistemi tenda degradarsi quando vengono aggiunti nuovi incrementi*, ogni volta che si aggiunge una funzionalità il codice diventa sempre più complesso e diventa dunque più difficile e costoso aggiungerne di nuove. Per ridurre il livello di degrado , i metodi agili suggeriscono un costante *refactoring* (miglioramento della struttura del software) del software.

Integrazione e configurazione

Si basa sul riutilizzare il software già esistente, questo approccio si fonda su una larga base di componenti software riutilizzabili e su un frame work di integrazione per comporre questi componenti.



Tre tipi di componenti software sono frequentemente utilizzati:

1. Sistemi di applicazioni indipendenti (COTS, Commercial off the shelf systems) che sono configurati per essere utilizzati in un particolare ambiente;
2. Collezioni di oggetti che sono sviluppate come un componente o un pacchetto da integrare tramite un frame work di integrazione dei componenti;
3. Servizi web che sono sviluppati in conformità agli standard e che sono disponibili per essere utilizzati da siti remoti su Internet.

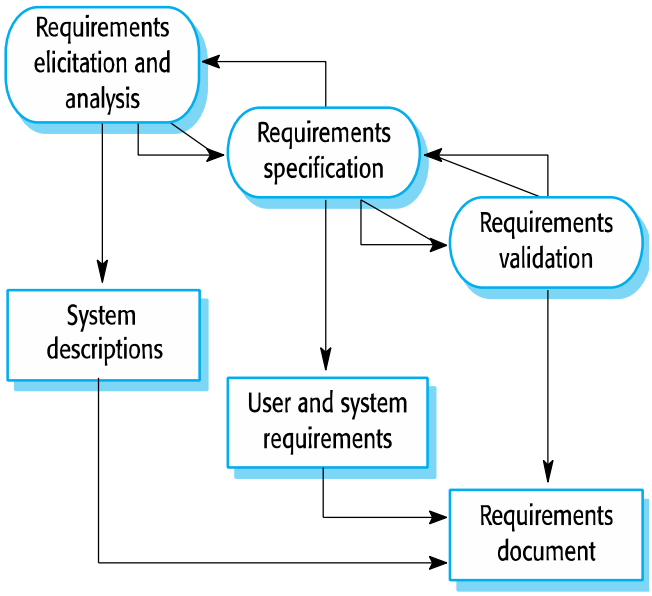
Questo modello presenta 5 fasi:

1. *Specifica dei requisiti*: vengono definiti i requisiti iniziali del sistema con una breve descrizione (non deve essere troppo elaborata);
2. *Ricerca e valutazione del software*: si ricercano i componenti e i sistemi che possono fornire le funzionalità richieste, vengono poi valutati per vedere se soddisfano i requisiti e se sono disponibili per il riutilizzo;
3. *Perfezionamento dei requisiti*: i requisiti sono perfezionati utilizzando le informazioni sui componenti trovati, vengono modificati in funzione dei componenti disponibili e la specifica del sistema viene ridefinita;
4. *Configurazione del sistema delle applicazioni:* se è disponibile un sistema delle applicazioni pronto all'uso che soddisfa i requisiti può essere configurato per creare il nuovo sistema;
5. *Adattamento e integrazione dei componenti:* se non è disponibile un sistema pronto all'uso, i singoli componenti possono essere modificati per sviluppare i nuovi componenti, che vengono poi integrati per creare il sistema.

Questo modello presenta, come i precedenti, vantaggi: ridurre la quantità di software da sviluppare (riducendo così costi e rischi) e conseguenti consegne più rapide; ma anche degli svantaggi: occorre andare incontro a dei compromessi nei requisiti che possono produrre un sistema che non soddisfi le reali necessità del cliente, inoltre si perde il controllo sull'evoluzione del sistema, poiché le nuove versioni dei componenti riutilizzati non sono sotto il controllo dell'organizzazione che li usa.

Attività di processo

I processi software reali sono sequenze intrecciate di attività tecniche, collaborative e manageriali con l'obiettivo comune di specificare, progettare, implementare e provare un sistema software. Le quattro attività fondamentali di specifica, sviluppo, convalida e evoluzione sono organizzate in modo diverso in diversi processi di sviluppo.



Specifica del software

La creazione delle specifiche del software (o ingegneria dei requisiti) è il processo per capire e definire quali servizi sono richiesti dal sistema, e per identificare i vincoli all'operatività e allo sviluppo del sistema, le fasi principali di questo processo sono 3:

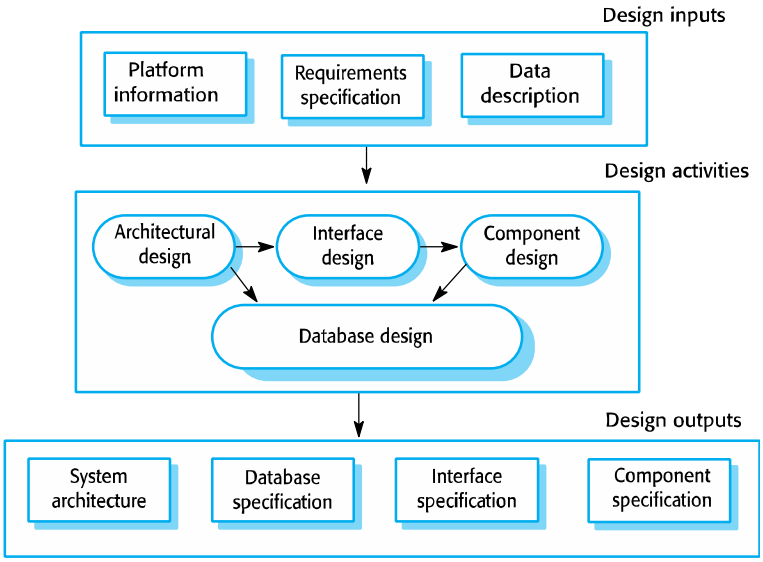
1. *Deduzione e analisi dei requisiti*: deduzione dei requisiti, discussione con i possibili utenti e acquirenti e analisi dei task (compiti);
2. *Specifica dei requisiti*: definizione dei requisiti nel dettaglio;
3. *Convalida dei requisiti*: si controlla la validità dei requisiti.

Progettazione e implementazione del software

Lo stadio di progettazione dello sviluppo del software è il processo di conversione delle specifiche in un sistema eseguibile da consegnare al cliente, progettazione e implementazione sono intrecciate se si applica un approccio agile nello sviluppo.

Sono 4 le attività che possono far parte del processo di progettazione per i sistemi informativi:

1. *Progettazione dell'architettura*: identificazione della struttura complessiva del sistema, dei componenti principali, delle loro relazioni e della loro distribuzione;
2. *Progettazione del database*: vengono progettate le strutture dei dati del sistema e come devono essere rappresentate in un database;
3. *Progettazione dell'interfaccia*: viene definita l'interfaccia tra le componenti del sistema;
4. *Progettazione e scelta dei componenti*: vengono ricercati i componenti riutilizzabili e, se non disponibili, ne vengono progettati di nuovi.



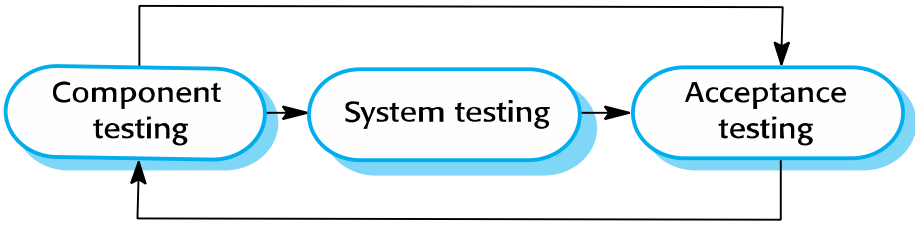
L'implementazione del sistema è realizzata tramite l'implementazione di uno o più programmi, oppure tramite la configurazione di un applicazione di sistema. Il design e lo sviluppo del programma sono attività intrecciata per la maggior parte dei sistemi. La programmazione è un'attività individuale e non c'è un processo generico che viene seguito abitualmente. Normalmente i programmatori testano il codice sviluppato in modo da mettere in luce i difetti che devono essere rimossi dal programma (**debugging**).

Convalida del software

La verifica (confronta ciò che abbiamo ottenuto con ciò che ci aspettavamo) e la convalida (confronta ciò che è stato implementato con ciò che era stato richiesto) servono a constatare che il sistema sia conforme alle specifiche e alle richieste del cliente. La convalida può richiedere anche processi di controllo, come le ispezioni e le revisioni, a ogni stadio del processo software, dalla definizione dei requisiti allo sviluppo del programma.

Ci sono tre stadi del processo di test:

1. *Test dei componenti (o delle unità)*: si testano i singoli componenti che possono essere entità semplici (come funzioni o classi di oggetti)o gruppi coerenti di queste unità;
2. *Test del sistema*: si uniscono le componenti e si testa il sistema completo (si cercano gli errori causati da interazioni tra le componenti e i problemi relativi alle interfacce), si provvede anche a convalidare la conformità dei requisiti, funzionali e non, e a verificare le proprietà significative del sistema;
3. *Test del cliente*: il sistema è testato dal cliente con i suoi dati (lavorando con dati reali e non di simulazione possono emergere errori dovuti l fatto che il sistema lavora in modo diverso con dati reali), possono emergere anche problemi con i requisiti, laddove le funzionalità del sistema non soddisfino il cliente.

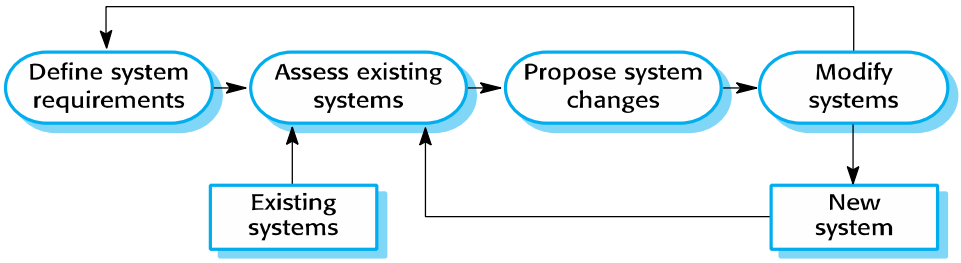


Evoluzione del software

La flessibilità dei sistemi software è una delle ragioni principali per cui sempre più software viene incorporato in sistemi grandi e complessi.

Il software può essere modificato in ogni momento (anche una grande modifica software è molto più economica di una qualsiasi modifica hardware); anche se c'è sempre stata una divisione tra i processi di sviluppo e di evoluzione (o manutenzione) del software ora, questa divisione, sta diventando sempre più irrilevante.

Far fronte ai cambiamenti



I cambiamenti sono inevitabili in tutti i grandi progetti software:

- Cambiamenti nel business portano a nuove priorità di gestione e modifica dei requisiti del sistema;
- Nuove tecnologie portano al cambiamento dei progetti e delle implementazioni.

I cambiamenti aggiungono costi allo sviluppo software perché prevedono una *rilavorazione* del progetto, per ridurre i costi della rilavorazione si usano due approcci correlati:

1. *Anticipazione dei cambiamenti*, quando il processo software include attività che possono anticipare o predire possibili variazioni prima che sia richiesta una significativa rilavorazione;
2. *Tolleranza ai cambiamenti*, quando il processo e il software sono progettati in modo che le modifiche possano essere facilmente apportate al sistema (implica solitamente qualche forma di sviluppo incrementale).

Vi sono poi due metodi per far fronte ai cambiamenti dei requisiti del sistema:

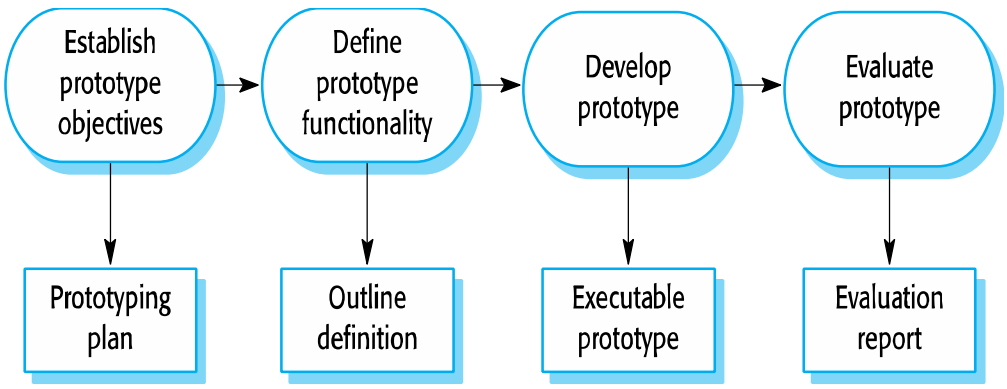
- *Prototipazione del sistema*: una versione del sistema o una sua parte sono sviluppate rapidamente per verificare i requisiti del cliente e la fattibilità delle scelte di progettazione, questo metodo supporta l'anticipazione dei cambiamenti;
- *Consegna incrementale*: gli incrementi del sistema sono consegnati al cliente per essere commentati e provati, supporta sia l'anticipazione dei cambiamenti che la tolleranza ad essi.

Prototipazione

Un **prototipo** è la versione iniziale di un sistema usata per dimostrare di concetti, provare opzioni di progettazione e scoprire più informazioni sui problemi e sulle possibili soluzioni. Un prototipo può essere utilizzato:

- Nel *processo di ingegneria dei requisiti*, può aiutare a identificare e convalidare i requisiti del sistema;
- Nel *processo di progettazione del sistema*, può essere usato per esplorare particolari soluzioni software e durante lo sviluppo di una UI;
- Nel *processo di testing del sistema*, per eseguire i test back-to-back.

I prototipi possono suggerire nuove idee per i requisiti e facilitare la scoperta dei punti di forza e debolezza del software, possono migliorare usabilità, progettazione e mantenimento del sistema così da ridurre l'impegno nello sviluppo. Possono essere basati su un rapido linguaggio di prototipazione o su tools però possono comportare l'omissione di funzionalità:



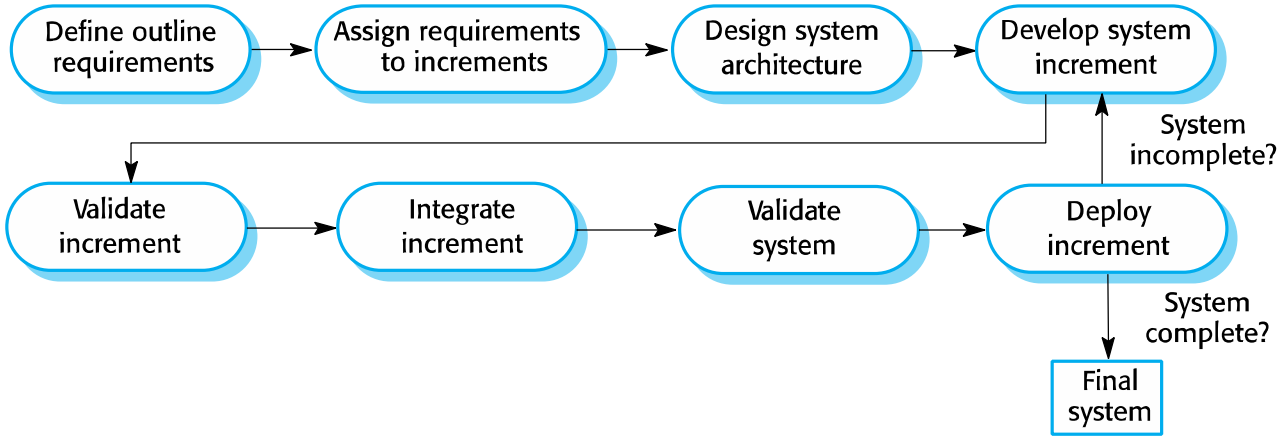
- I prototipi dovrebbero focalizzarsi sulle aree del prodotto che non sono state ben capite;
- La gestione degli errori potrebbe non essere inclusa per ridurne i costi;
- Dovrebbe Concentrarsi su requisiti funzionali piuttosto che non funzionali come affidabilità e sicurezza.

Dopo lo sviluppo e la dimostrazione un prototipo dovrebbe essere scartato poiché non è una buona base su cui avviare la produzione del sistema:

- Potrebbe essere impossibile regolare il sistema per soddisfare requisiti non funzionali;
- I prototipi sono normalmente privi di documenti;
- La struttura del prototipo viene solitamente degradata a causa di rapidi cambiamenti;
- Il prototipo probabilmente non soddisferà i normali standard di qualità organizzativa.

Consegna incrementale

La **consegna incrementale** è un approccio allo sviluppo software in cui, piuttosto che fornire il sistema come una singola consegna, lo sviluppo e la consegna vengono suddivisi in incrementi con ogni incremento che fornisce parte della funzionalità richiesta. I requisiti degli utenti hanno la priorità e i requisiti con la priorità più alta sono inclusi nei primi incrementi. Una volta avviato lo sviluppo di un incremento, i requisiti vengono congelati sebbene i requisiti per gli incrementi successivi possano continuare a evolversi.



Presenta diversi vantaggi:

1. Il valore per il cliente può essere fornito con ogni incremento in modo che la funzionalità del sistema sia disponibile prima.
2. I primi incrementi fungono da prototipo per aiutare a ottenere i requisiti per gli incrementi successivi ma, diversamente dai prototipi, sono parte reale del sistema;
3. Minor rischio di fallimento complessivo del progetto, conserva i benefici dello sviluppo incrementale;
4. I servizi di sistema con la priorità più alta tendono a ricevere il maggior numero di test.

Ma anche svantaggi:

1. La consegna iterativa è problematica quando un nuovo sistema deve sostituire un sistema esistente, gli utenti hanno bisogno di tutte le funzionalità del vecchio sistema e di solito sono riluttanti a sperimentare un sistema completamente nuovo;
2. Molti sistemi richiedono una serie di funzionalità di base che non sono utilizzate da parti differenti del sistema, che magari non sono ancora state implementate;
3. La specifica è sviluppata assieme al software cosa che è in conflitto con il modello di acquisizione di molte aziende in cui la specifica del sistema completo è parte del contratto di sviluppo software.

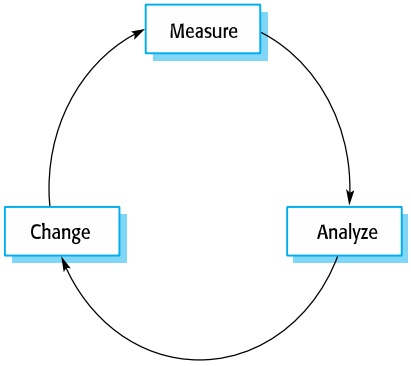
Miglioramento dei processi

Molte società di software si sono rivolte al miglioramento dei processi software come un modo per migliorare la qualità del loro software, ridurre i costi o accelerare i processi di sviluppo. Migliorare i processi significa comprendere i processi esistenti e modificare questi processi per aumentare la qualità del prodotto e / o ridurre i costi e i tempi di sviluppo.

Per cambiare e migliorare i processi si usano due differenti approcci:

- *Approccio di maturità del processo*: si concentra sul miglioramento della gestione dei processi e dei progetti e sull'introduzione di buone pratiche di ingegneria del software. Il livello di maturità del processo riflette la misura in cui è stata adottata una buona pratica tecnica e gestionale nei processi di sviluppo del software organizzativo.
- *Approccio agile*: si concentra sullo sviluppo iterativo e sulla riduzione dei costi generali nel processo software. Le caratteristiche principali dei metodi agili sono la consegna rapida di funzionalità e reattività alle mutevoli esigenze dei clienti.

Il processo generale di miglioramento dei processi che sta alla base dell'approccio di maturità è un'attività ciclica, gli stadi di questo processo sono:

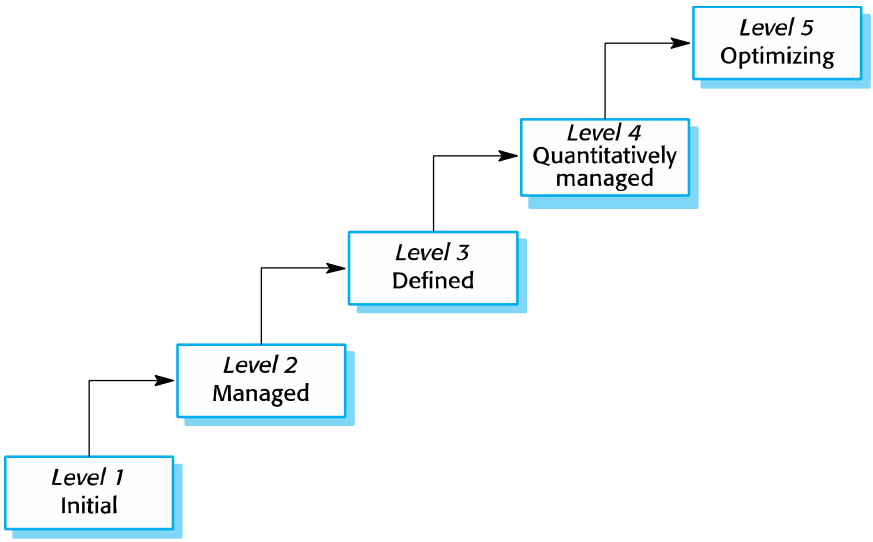


1. *Misurazione del processo*: si misurano uno o più attributi del processo o del prodotto software. Queste misurazioni costituiscono una linea di base che aiuta a decidere se i miglioramenti del processo sono stati efficaci.
2. *Analisi di processo*: Viene valutato il processo in corso e vengono identificati i punti deboli e le strozzature del processo. Possono essere sviluppati modelli di processo (a volte chiamati mappe di processo) che descrivono il processo.
3. *Modifica del processo*: Vengono proposte modifiche al processo per affrontare alcune delle debolezze del processo identificate. Questi vengono introdotti e il ciclo riprende per raccogliere dati sull'efficacia delle modifiche.

Ove possibile, dovrebbero essere raccolti dati quantitativi di processo, tuttavia, se le organizzazioni non hanno standard di processo chiaramente definiti, ciò è molto difficile in quanto non sai cosa misurare. Potrebbe essere necessario definire un processo prima che sia possibile una misurazione. Le misurazioni del processo dovrebbero essere utilizzate per valutare i miglioramenti del processo, ma questo non significa che le misurazioni debbano guidare i miglioramenti. Il driver di miglioramento dovrebbero essere gli obiettivi organizzativi.

I livelli del modello di maturità delle capacità dei processi sono:

1. *Iniziale:* Essenzialmente incontrollato;
2. *Ripetibile:* Procedure di gestione del prodotto definite e utilizzate;
3. *Definito:* Procedure e strategie di gestione dei processi definite e utilizzate;
4. *Gestito:* Strategie di gestione della qualità definite e utilizzate;
5. *Ottimizzazione:* Strategie di miglioramento dei processi definite e utilizzate.



5 - MODELLI di SISTEMA

La modellazione del sistema è il processo di sviluppo di modelli astratti di un sistema, con ogni modello che presenta una diversa visione o prospettiva di quel sistema, è ora arrivata a rappresentare un sistema utilizzando un qualche tipo di notazione grafica, che ora è quasi sempre basata su notazioni in **UML** (Unified Modeling Language). La modellazione del sistema aiuta l'analista a comprendere la funzionalità del sistema e i modelli vengono utilizzati per comunicare con i clienti. È possibile sviluppare modelli sia di sistemi esistenti sia di nuovi sistemi:

- I modelli del sistema esistente vengono utilizzati durante la progettazione dei requisiti. Aiutano a chiarire cosa fa il sistema esistente e possono essere usati come base per discutere i suoi punti di forza e di debolezza. Questi quindi portano ai requisiti per il nuovo sistema.
- I modelli del nuovo sistema vengono utilizzati durante l'ingegneria dei requisiti per aiutare a spiegare i requisiti proposti ad altre parti interessate del sistema. Gli ingegneri utilizzano questi modelli per discutere le proposte di progettazione e per documentare il sistema per l'implementazione.

In un processo di ingegneria basato sul modello, è possibile generare un'implementazione del sistema completa o parziale dal modello del sistema. È possibile sviluppare vari modelli per rappresentare il sistema da differenti prospettive:

- Una *prospettiva esterna*, in cui si modella il contesto o l'ambiente del sistema.
- Una *prospettiva di interazione*, in cui modellate le interazioni tra un sistema e il suo ambiente, o tra i componenti di un sistema.
- Una *prospettiva strutturale*, in cui si modella l'organizzazione di un sistema o la struttura dei dati elaborati dal sistema.
- Una *prospettiva comportamentale*, in cui si modella il comportamento dinamico del sistema e come risponde agli eventi.

Quando si sviluppano i modelli dei sistemi, occorre una certa flessibilità nel modo di utilizzare la notazione grafica, il dettaglio e il rigore di un modello dipendono dal modo in cui s'intende utilizzarlo. Ci sono tre modi in cui i modelli grafici sono comunemente utilizzati:

1. Come *mezzo per facilitare la discussione* su un sistema esistente o proposto, i modelli incompleti e errati vanno bene poiché il loro ruolo è quello di supportare la discussione;
2. Come un *modo per documentare un sistema esistente*, i modelli dovrebbero essere una rappresentazione accurata del sistema ma non è necessario che siano completi;
3. Come *descrizione dettagliata del sistema* che può essere utilizzata per generare un'implementazione del sistema, i modelli devono essere corretti e completi.

L'UML ha 13 tipi di diagrammi e quindi consente la creazione di vari tipi di modelli di sistemi, tuttavia si ritiene che siano 5 i tipi di diagrammi sufficienti per rappresentare gli elementi essenziali di un sistema:

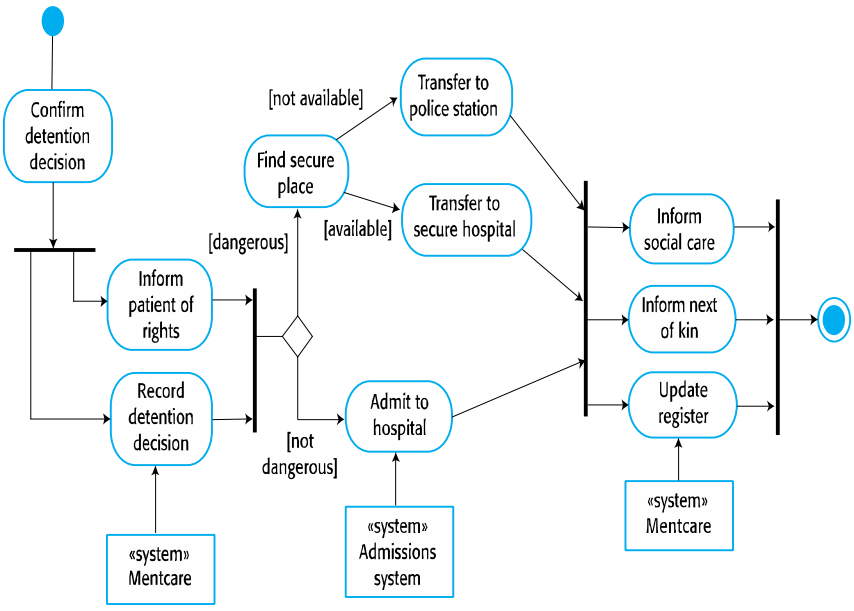
1. *Diagrammi di attività*, mostrano le attività coinvolte in un processo o nell'elaborazione dei dati;
2. *Diagrammi di casi d'uso*, mostrano le interazioni tra un sistema e il suo ambiente;
3. *Diagrammi di sequenza*, mostrano le interazioni tra gli attori e il sistema e tra i componenti del sistema;
4. *Diagrammi di classe*, che mostrano le classi di oggetti nel sistema e le associazioni tra queste classi;
5. *Diagrammi di stato*, che mostrano come il sistema reagisce agli eventi interni ed esterni.

Modelli contestuali

I modelli contestuali vengono utilizzati per illustrare il contesto operativo di un sistema: mostrano cosa si trova al di fuori dei confini del sistema. Le preoccupazioni sociali e organizzative possono influenzare la decisione su dove posizionare i confini del sistema. I modelli architettonici mostrano il sistema e la sua relazione con altri sistemi. I confini del sistema sono stabiliti per definire cosa vi è al suo interno e cos'è lasciato fuori, la posizione dei confini ha un profondo effetto sui requisiti di sistema. Non sempre però il confine del sistema è chiaro, ci sono volte in cui lo è relativamente e altre in cui, avendo più flessibilità, occorre decidere quale sia il confine tra il sistema e il suo ambiente durante il processo di ingegneria dei requisiti.

Il vantaggio di affidare informazioni ad altri sistemi è che si evita di duplicare i dati, allo stesso tempo però, l'accesso alle informazioni è più lento e, se gli altri sistemi non dovessero essere disponibili risulterebbe impossibile utilizzare tutto il sistema. La definizione dei confini di un sistema non è un giudizio senza valore: problemi sociali e organizzativi possono richiedere che la posizione di un confine sia determinata da fattori non tecnici.

I modelli di contesto mostrano semplicemente gli altri sistemi nell'ambiente, non il modo in cui il sistema in fase di sviluppo viene utilizzato in quell'ambiente. I modelli di processo rivelano come il sistema in fase di sviluppo viene utilizzato in processi aziendali più ampi. I diagrammi di attività UML possono essere utilizzati per definire modelli di processi aziendali; mostrano le attività in un processo e il flusso di controllo da un'attività all'altra.



- *Cerchio pieno*: inizio di un processo;
- *Cerchio in un cerchio*: fine del processo;
- *Rettangoli con spigoli arrotondati*: attività (specifici sottoprocessi che devono essere eseguiti);
- *[...]:* tipo di entità;
- *Freccia:* flusso di lavoro da un'attività all'altra, quando il flusso da più attività porta a una *barra piena*, allora tutte queste attività devono essere completate prima che il processo possa avanzare, se il flusso va da una barra piena verso più attività, possono essere eseguite in parallelo. Alle frecce si possono associare annotazioni tra parentesi quadre, che specificano quando tale flusso viene eseguito.

Modelli di interazione

Tutti i sistemi hanno un interazione di qualche tipo (con l'utente, con altri software o con componenti), la modellazione di questa interazione aiuta a identificare i requisiti dell'utente e evidenzia eventuali problemi di comunicazione. La modellazione dell'interazione tra i componenti aiuta a capire se la struttura proposta per un sistema sarà in grado di fornire la fidatezza e le prestazioni richieste, vi sono due approcci per la modellazione di queste:

- *Modellazione dei casi d'uso*, usata per modellare le interazioni tra il sistema e gli agenti esterni;
- *Diagrammi di sequenza*, usata per modellare le interazioni tra i componenti del sistema (in alternativa si possono usare i *Diagrammi di comunicazione*).

Modellazione dei casi d'uso

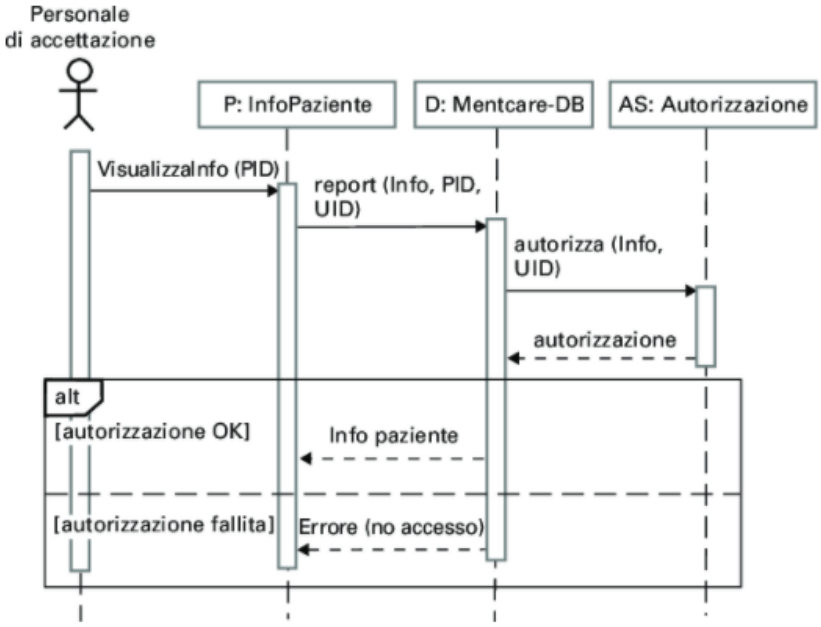
Sviluppata da Ivar Jacobsen negli anni '90, un caso d'uso può essere considerato come una semplice descrizione di ciò che un utente si aspetta da un sistema in tale interazione. Ciascun caso d'uso rappresenta un compito discreto che richiede un'interazione esterna con un sistema in cui:

- *ellisse*: caso d'uso;
- "*omini* ": attori (che possono essere persone o altri sistemi);
- *Frecce*: direzione del flusso dei messaggi.

Spesso occorre completare l'informazione fornita da un diagramma con una descrizione (tabellare, testuale o con un diagramma di sequenza) più dettagliata.

Diagrammi di sequenza

Sono usati per modellare le interazioni tra gli attori e gli oggetti di un sistema (o tra gli oggetti stessi), mostra la sequenza delle interazioni che si svolgono durante un particolare caso d'uso o una sua istanza. Gli oggetti e gli attori coinvolti sono elencati all'inizio del diagramma, con una linea tratteggiata verticale che parte da ciascuno di essi. Le interazioni tra oggetti sono denotate da frecce con dei commenti al di sopra che indicano le chiamate degli oggetti, i loro parametri e i valori di ritorno. I rettangoli sulle linee tratteggiate indicano la linea di vita dell'oggetto (ossia il tempo durante il quale l'istanza dell'oggetto prende parte al calcolo) e la sequenza delle interazioni va letta dall'alto verso il basso. In alcuni casi è presente un box denominato "alt" con le condizioni tra parentesi quadre delle possibili opzioni di interazione.



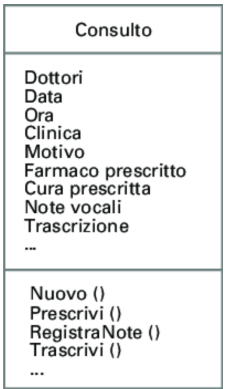
Modelli strutturali

Mostrano l'organizzazione del sistema in funzione dei componenti e delle relazioni tra di essi. Possono essere **statici** se rappresentano l'organizzazione del sistema; oppure **dinamici** se mostrano l'organizzazione del sistema durante la sua esecuzione. Questi modelli non sono la stessa cosa e possono differire di molto l'una dall'altra. Questi modelli sono creati quando descriviamo e progettiamo l'architettura del sistema.

Diagrammi di classe

Usati quando si sviluppa un modello di sistema orientato agli oggetti per mostrare le classi di un sistema e le loro associazioni. Una **classe** di oggetti può essere pensata come una definizione generale di un tipo di oggetti del sistema, un'**associazione** è un collegamento tra classi che indica che esiste qualche relazione tra queste.

Quando si sviluppano modelli durante le prime fasi del

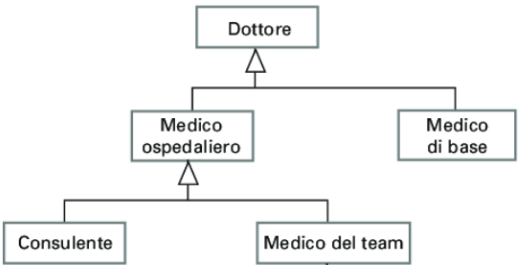


processo di ingegneria del software, gli oggetti rappresentano qualcosa del mondo reale: i *rettangoli* sono la classe, la presenza o meno di una linea che li collega ne indica la presenza di un'associazione tra le due, i *numeri* indicano la quantità esatta di oggetti di quella classe che sono coinvolti nell'associazione (* usato per una quantità indefinita).

Per aumentare il livello di dettaglio di una classe e sapere dunque le **operazioni** che può compiere e gli **attributi** (caratteristiche della classe), basta ampliare il rettangolo.

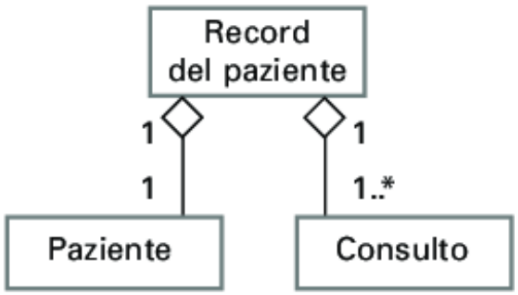
Generalizzazione

Tecnica usata per gestire la complessità in cui, piuttosto che conoscere le caratteristiche dettagliate di tutto ciò che incontriamo, ci limitiamo a conoscere soltanto le classi **generali** e le loro caratteristiche; le definizioni generali si applicano a tutti i membri della classe (ad esempio tutti i roditori hanno i denti, un topo essendo un membro della classe generale roditori ha i denti); le sottoclassi (ossia le classi a livello più basso rispetto a quella/e generale) oltre ad ereditare le operazioni e gli attributi della superclasse, aggiungono a questi anche le proprie andando a creare classi più specifiche e dettagliate. È un procedimento simile all'ereditarietà delle classi in Java ed è rappresentata da una freccia che punta alla classe più generale.



Aggregazione

Gli oggetti del mondo reale sono spesso composti da parti diverse, nei diagrammi UML questo è reso da un tipo speciale di associazione detta **aggregazione,** in cui un oggetto (intero) è composto da altri oggetti (parti). Per definirla viene aggiunto un rombo al collegamento vicino alla classe che rappresenta l'intero.



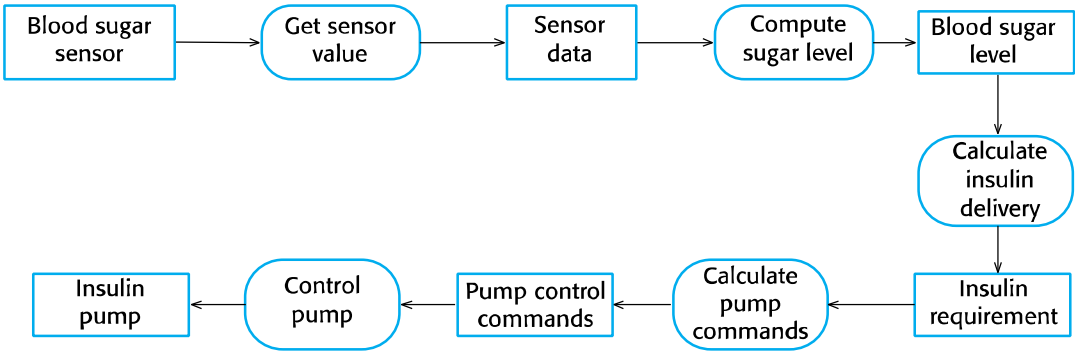
Modelli comportamentali

Sono modelli del comportamento dinamico di un sistema durante la sua esecuzione, mostrano cosa accade o cosa dovrebbe accadere quando il sistema risponde a stimoli provenienti dal suo ambiente. Questi stimoli possono essere:

- **Dati:** i dati che arrivano e devono essere processati dal sistema (sistemi aziendali);
- **Eventi:** un evento innesca l'elaborazione del sistema, possono essere associati ai dati(sistemi real-time).

Modelli guidati dai dati

Molti sistemi aziendali sono sistemi di elaborazione dei dati, che sono controllati prettamente dall'input di dati, con una modesta elaborazione di eventi esterni. L'elaborazione dei dati consiste in una sequenza di azioni sui dati e nella generazione di un output.



Sono particolarmente utili durante l'analisi dei requisiti poiché mostrano l'elaborazione dall'inizio alla fine in un sistema (tracciando e documentando come i dati associati a un particolare processo si spostano all'interno di un sistema), ossia l'intera sequenza delle azioni svolte dal momento della ricezione dell'input alla generazione dell'output. Nei diagrammi di flusso dei dati: i passi di elaborazione sono

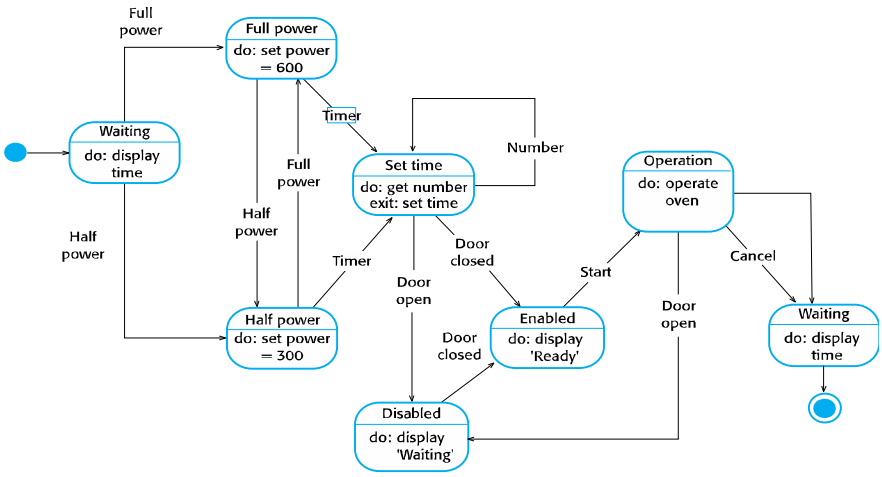
rappresentati come attività (rettangoli a spigoli smussati) mentre i dati che fluiscono tra questi passi sono rappresentati come oggetti (rettangoli).

Un modo alternativo di mostrare la sequenza dell'elaborazione in un sistema consiste nell'usare i diagrammi di sequenza UML. La differenza tra i diagrammi di flusso dei dati e i diagrammi di sequenza è che: i primi mettono in evidenza le operazioni o attività del sistema, mentre i secondi evidenziano gli oggetti del sistema.

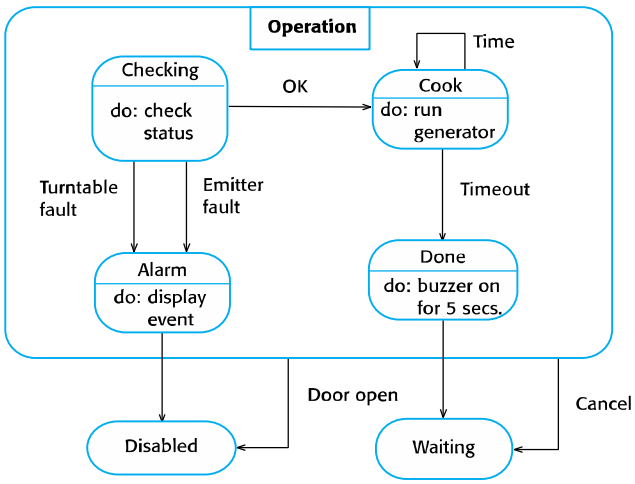
Modelli guidati dagli eventi

Mostrano come un sistema risponde agli eventi esterni ed interni, si basa sull'ipotesi che un sistema ha un numero finito di stati e che gli eventi (stimoli) possono causare una transazione da uno stato all'altro. Questo modo di vedere il sistema è particolarmente appropriato per i sistemi real-time (un sistema che controlla una valvola che passa dallo stato "valvola aperta" a quello "valvola chiusa" quando riceve lo "stimolo" ossia l'evento da un operatore). L'UML supporta questa modellazione tramite i diagrammi di stato, questi mostrano gli stati e gli eventi che causano le transazioni da uno stato all'altro; non mostrano il flusso dei dati all'interno del sistema, ma possono includere informazioni addizionali sui calcoli da eseguire in ciascuno stato. In questi diagrammi:

- i rettangoli a spigoli arrotondati rappresentano gli stati del sistema, possono includere una breve descrizione delle azioni svolte in quello stato;
- le frecce con le annotazione rappresentano gli stimoli che forzano la transazione da uno stato all'altro;
- gli stati iniziale e finale sono rappresentati da dei cerchi pieni come nei diagrammi di attività.



Il problema della modellazione basata sugli stati è che il numero dei possibili stati aumenta rapidamente, di conseguenza è necessario nascondere i dettagli nei modelli di grandi sistemi. Un modo per farlo è usando il concetto di superstato che comprende una serie di stati separati (un unico stato che viene poi espanso per mostrare maggiori dettagli su un diagramma separato).



Questi modelli forniscono una panoramica dell'elaborazione degli eventi; di solito occorre estendere questi modelli con una descrizione più dettagliata degli stimoli e degli stati del sistema.

Ingegneria guidata da modelli

Sviluppata dal concetto di architettura guidata da modelli, l'ingegneria guidata dai modelli (MDE, Model-Driven Engineering) è un approccio allo sviluppo dei sistemi software dove i modelli, anziché i programmi, sono il principale output del processo di sviluppo. I programmi che sono eseguiti su piattaforme hardware/software sono generati automaticamente dai modelli. I sostenitori della MDE ritengono che così si alza il livello di astrazione nell'ingegneria del software, poiché gli ingegneri non devono più preoccuparsi dei dettagli dei linguaggi di programmazione o delle specifiche delle piattaforme di esecuzione. È ancora ad uno stato iniziale di sviluppo e non si sa ancora se avrà o meno effetti significativi sull'ingegneria del software nel pratico:

- *Pro*:
  - permette ai sistemi di essere considerati a un livello più alto di astrazione;

- la generazione automatica di codice comporta un minor costo nell'adattamento dei sistemi a nuove piattaforme.

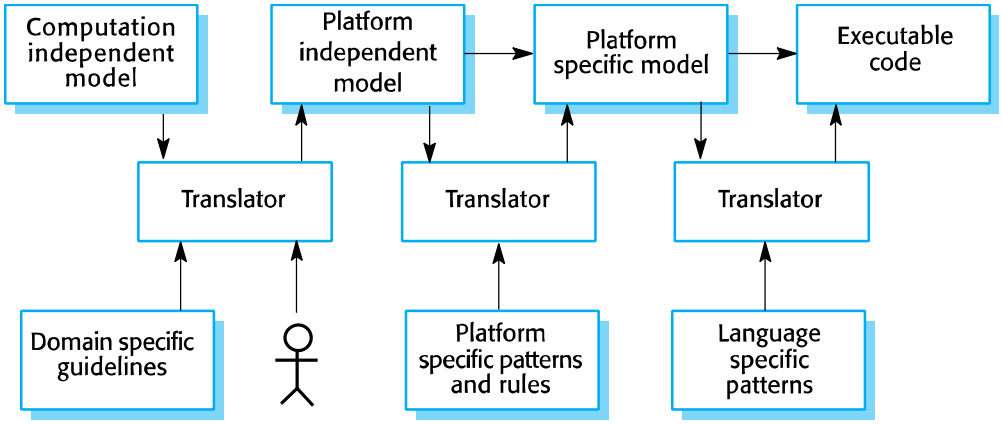
- *Contro:*
  - i modelli usati per l'astrazione non sono necessariamente giusti per l'implementazione;
  - il risparmio derivante dalla generazione del codice potrebbe essere sormontato dal costo dello sviluppo di traslatori per le nuove piattaforme.

Architettura guidata da modelli

La MDA (Model-Driven Architecture) è stata la madre di ingegnerie guidate da modelli più generali, è una tecnica basata sui modelli per progettare e implementare software che usa un sottoinsieme di modelli UML per descrivere un sistema. Vengono creati modelli a diversi livelli di astrazione, da un modello di alto livello (indipendente dalla piattaforma) teoricamente è possibile generare un programma funzionante senza interventi manuali.

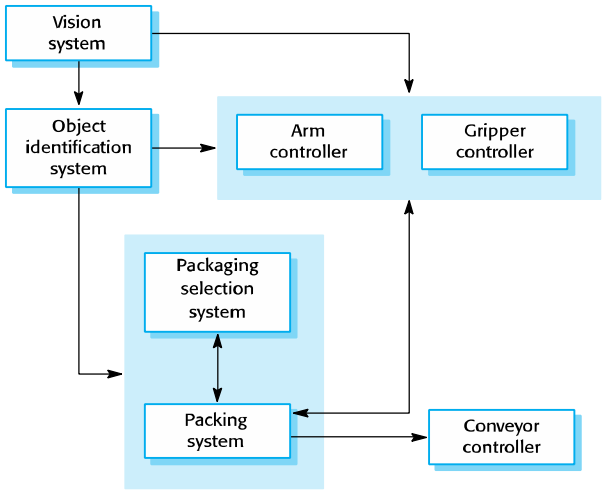
Questo metodo consiglia di produrre tre diversi tipi di modelli astratti per un sistema:

1. *Modelli indipendenti dal calcolo **(CIM)***: si basano sulle principali astrazioni dei domini utilizzate in un sistema e, per questo, ogni tanto sono chiamati *modelli dei domini;*
2. *Modelli indipendenti dalle piattaforme **(PIM)***: si basano sul funzionamento del sistema, senza riferimento alla sua implementazione. Descritto solitamente tramite i modelli UML che mostrano la struttura statica di un sistema e come questa risponde agli eventi interni ed esterni (event-driven models);
3. *Modelli specifici delle piattaforme **(PSM)***: sono trasformazioni dei PIM in cui ogni piattaforma di applicazioni ha uno specifico PSM. Possono esserci più strati di PSM, dove ogni strato aggiunge qualche dettaglio specifico per una piattaforma.



Gli sviluppatori della MDA affermano che è destinato a supportare un approccio iterativo allo sviluppo e quindi può essere usato all'interno di metodi agili. La nozione di modellazione estensiva iniziale, tuttavia, contraddice le idee fondamentali che si trovano nel manifesto del metodo agile. Se le trasformazioni possono essere completamente automatizzate e i programmi completamente generati un PIM allora, in linea di principio, l'MDA potrebbe essere utilizzato nel processo di sviluppo agile senza alcuna codifica separata necessaria.

6 - PROGETTAZIONE ARCHITETTURALE



La progettazione architetturale si occupa di capire come un sistema software dovrebbe essere organizzato e della progettazione della sua struttura complessiva (è il primo stadio del processo di progettazione del software). È il collegamento critico tra progettazione e ingegneria dei requisiti, in quanto identifica i principali componenti strutturali di un sistema e le relazioni tra di essi. L'output è un modello architetturale che descrive come il sistema è organizzato in funzione dei componenti di comunicazione.

Nei processi agili è generalmente accettato che una fase iniziale del processo sia dedicata alla progettazione dell'architettura complessiva del sistema.

La rifattorizzazione dei componenti in risposta ai cambiamenti è un compito relativamente facile ma solitamente costoso, questo perché ha effetto su molti componenti del sistema. Il diagramma che si utilizza mostra un modello astratto dell'architettura, con i componenti del sistema e i loro collegamenti. È possibile progettare l'architettura del software a due livelli di astrazione:

- *Architettura in piccolo*: riguarda l'architettura dei singoli programmi, a questo livello ci interessa il modo in cui un singolo programma è scomposto nei suoi componenti;
- *Architettura in grande*: riguarda l'architettura si complessi sistemi aziendali che includono altri sistemi, programmi e componenti di programmi. Questi sistemi possono essere distribuiti su diversi computer, i quali potrebbero essere di proprietà o gestiti da società differenti.

L'architettura del software è importante poiché influisce sulle prestazioni, robustezza, distribuzione e manutenzione di un sistema. La progettazione e documentazione esplicita dell'architettura del software comporta tre vantaggi :

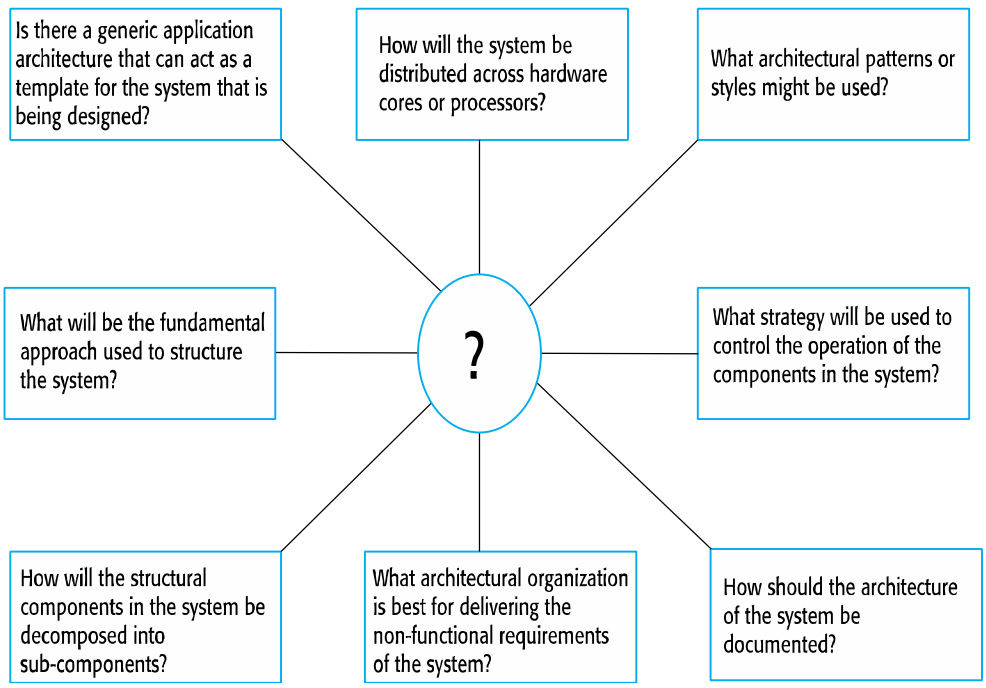
1. *Comunicazione tra gli stakeholder*: l'architettura è una presentazione di alto livello del sistema che potrebbe essere usata per focalizzare i temi di discussione tra vari stakeholder;
2. *Analisi del sistema*: analizza se il sistema può o meno raggiungere i requisiti non funzionali;
3. *Riutilizzo su vasta scala*: l'architettura del sistema è spesso la stessa per sistemi con requisiti simili, può quindi supportare il riutilizzo del software su vasta scala.

Le architetture dei sistemi sono spesso modellate in modo informale utilizzando semplici diagrammi a blocchi in cui ciascun box rappresenta un componente, un box dentro l'altro indica che il componente è stato scomposto in sottocomponenti e le frecce indicano che i dati o i segnali di controllo passano da un componente all'altro nella direzione della freccia. Questi diagrammi offrono una vista di alto livello della struttura del sistema, tuttavia sono stati criticati poiché non mostrano la semantica, il tipo delle relazioni tra le componenti e neanche le proprietà esternamente visibili dei componenti. Ci sono due modi di utilizzare il modello architetturale di un programma:

1. *Come un modo di incoraggiare le discussioni sulla progettazione del sistema;*
2. *Come un modo di documentare un'architettura che è stata progettata.*

Decisioni di progettazione architetturale

La progettazione architetturale è un processo creativo in cui si progetta un'organizzazione che soddisfa i requisiti funzionali e non di un sistema. Questo processo dipende dal tipo di sistema che si sta sviluppando, dal background, dall'esperienza dell'architetto e dai requisiti specifici del sistema; sarebbe quindi più giusto parlare di una **serie di decisioni** da prendere piuttosto che una serie di attività. Per quanto ogni sistema sia unico, i sistemi nello stesso dominio di applicazione hanno spesso architetture simili che ne riflettono i concetti fondamentali. L'architettura di un sistema software può basarsi su un particolare *schema* o *stile architetturale*, è una descrizione dell'organizzazione del sistema (ad esempio client-server o a strati); cattura l'essenza di un architettura e può essere instanziato in modi differenti.



A causa della stretta relazione tra le caratteristiche non funzionali del sistema e l'architettura del software, la scelta dello schema architetturale e della struttura dipende dai requisiti non funzionali del sistema.

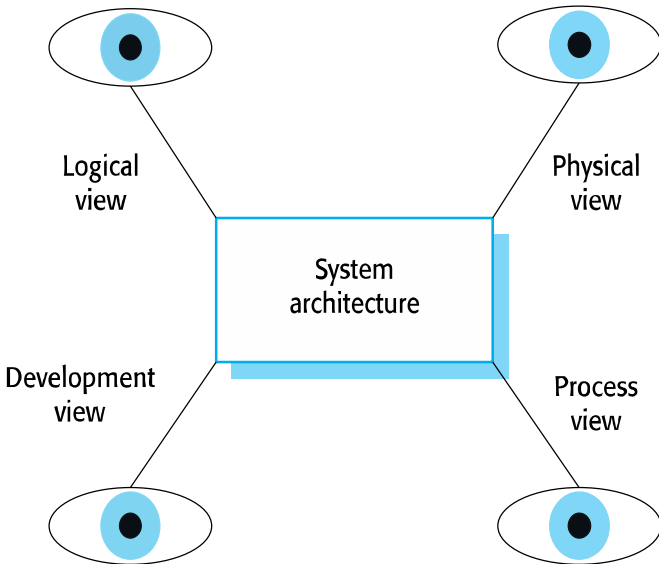
1. *Prestazioni*: se le prestazioni sono un requisito critico, localizza le operazioni critiche e minimizza le comunicazioni, a tal fine si utilizzano componenti grandi al posto di moduli più piccoli;
2. *Protezione*: se la protezione è un requisito critico, si usa un'architettura a strati, con le risorse critiche nei livelli più interni (alto livello di convalida della protezione ad ogni livello);
3. *Sicurezza*: se la sicurezza è un requisito critico, bisogna localizzare tutte le operazioni relativa alla sicurezza in un piccolo numero di componenti (si riducono costi e problemi di convalida);
4. *Disponibilità*: se la disponibilità è un requisito critico, l'architettura dev'essere progettata per includere componenti ridondanti, in modo tale da poterli sostituire o aggiornare senza bloccare il sistema (sono architetture ad alta resistenza ai guasti);
5. *Mantenibilità*: se la mantenibilià è un requisito critico, l'architettura dovrebbe essere progettata utilizzando componenti piccoli, autonomi e sostituibili;

Viste architetturali

Abbiamo detto all'inizio come i modelli architetturali possono essere usati per focalizzare un discussione o per documentare un progetto, quali viste o prospettive sono utili quando si progetta e si documenta l'architettura di un sistema? Quali notazioni andrebbero usate per la descrizione di modelli architetturali?

Ogni modello architetturale mostra solo una vista o prospettiva del sistema, **Krutchen** nel 1995 con il suo modello "**4+1 viste** dell'architettura del software", suggerisce quattro viste architetturali fondamentali che possono essere collegate tramite scenari o casi d'uso comuni:

1. *Vista logica:* mostra le astrazioni chiave nel sistema come oggetti o classi di oggetti (utile per mettere in relazione i requisiti del sistema con le entità);
2. *Vista dei processi:* mostra come, a runtime, il sistema è composto da processi interattivi (utile per valutare le caratteristiche non funzionali del sistema, come prestazioni e disponibilità);
3. *Vista di sviluppo:* mostra come il software viene scomposto per lo sviluppo nei singoli componenti (utile ai programmatori e manager);
4. *Vista fisica:* mostra l'hardware del sistema e come i componenti software sono distribuiti tra i processori nel sistema (utile agli ingegneri che pianificano l'installazione dei sistemi);
5. *(+1) Casi d'suo o scenari collegati;*



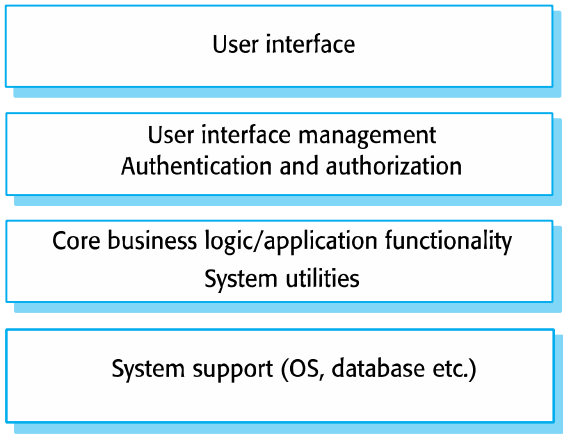
Alcune persone sostengono che l'UML sia una notazione appropriata per la descrizione e la documentazione dell'architettura dei sistemi, secondo me il linguaggio UML non dispone delle astrazioni appropriate per la descrizione di sistemi ad alto livello. Linguaggi di descrizione architetturale (ADL) sono stati sviluppati ma non sono ancora largamente usati.

Schemi architetturali

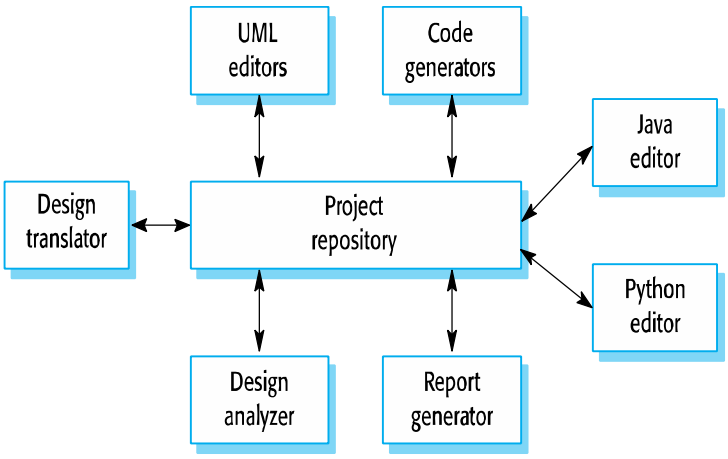
L'idea degli schemi è quella di rappresentare, condividere e riusare le conoscenze dei sistemi software. Uno schema architetturale è una descrizione stilizzata di una buona pratica progettuale, che è stata provata e testata in ambienti differenti. Gli schemi dovrebbero includere informazioni riguardanti quando sono utili e quando no e i dettagli sui suoi vantaggi e svantaggi. Possono essere rappresentati tramite descrizioni tabellari o con grafici.

Architettura a strati

Usata per modellare l'interfacciamento di sottosistemi, organizza le funzionalità del sistema in strati (o macchine astratte) separati, ciascuno dei quali si basa sui servizi e le funzioni offerti dallo strato sottostante. Supporta lo sviluppo incrementale dei sistemi, inoltre se si modificano le interfacce di uno strato, solo lo strato adiacente ne viene influenzato (con questa architettura abbiamo un'organizzazione concettuale del sistema).



Architettura repository



Mostra come una serie di componenti interattivi possono condividere dati, questo può avvenire in due modi:

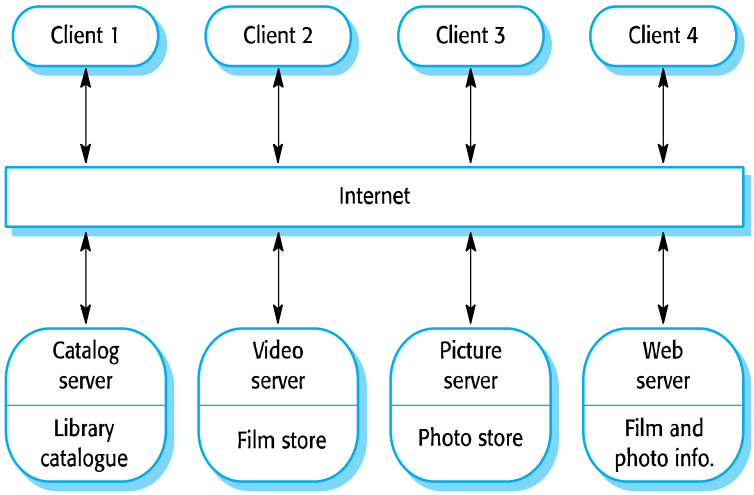
- I dati condivisi sono mantenuti in un database centrale o repository e tutti i componenti possono farvi accesso;
- Ogni componente ha il proprio database e passa i dati esplicitamente a tutti gli altri componenti del sistema;

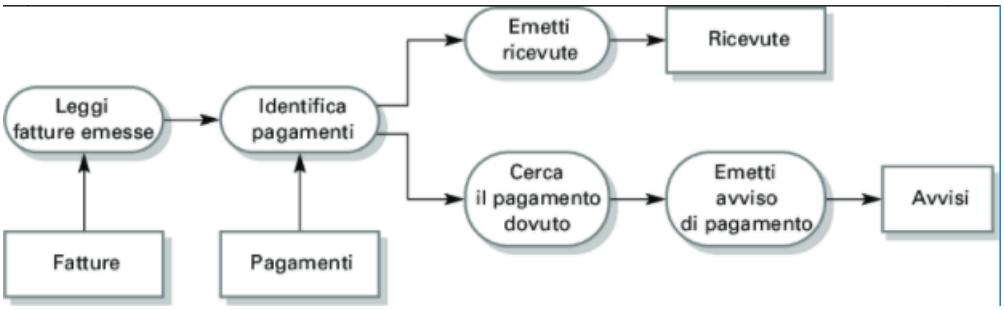
Il modello di scambio tramite repository è comunemente utilizzato quando grandi quantità di dati devono essere condivise poiché è un meccanismo di scambio molto efficiente (riguarda la struttura statica di un sistema, non mostra la sua organizzazione a runtime).

Architettura client-server

Modello di sistema distribuito che mostra come i dati e l'elaborazione sono distribuiti attraverso una gamma di componenti (può essere implementato su un singolo computer). I principali componenti di questo modello sono:

- Un insieme di server stand-alone che offrono specifici servizi quali stampa, gestione dei dati, ecc;
- Un insieme di clienti che richiedono i servizi offerti dai server;
- Una rete che permette ai clienti di accedere a questi servizi;





Architettura pipe and filter

Rappresenta l'organizzazione a runtime di un sistema dove le trasformazioni funzionali elaborano i loro input per produrre dell'output, i dati passano da una trasformazione all'altra attraverso una sequenza. Il nome deriva dalla terminologia usata nei processi Unix in cui è possibile collegare processi tramite "pipe". Varianti di questo sistema sono molto comuni; quando le trasformazioni sono sequenziali con i dati

elaborati in blocco si parla di modello batch sequenziale. Molto usato nei sistemi di processo dati, tuttavia non molto adatto per i sistemi interattivi.

Architetture applicative

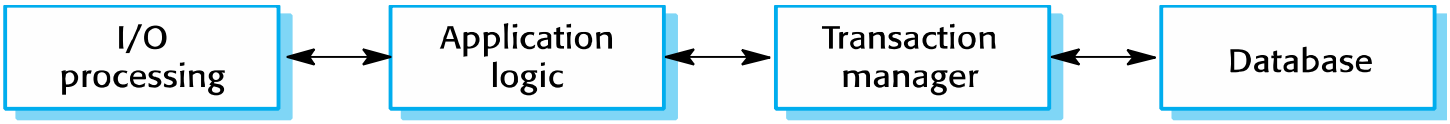
I sistemi applicativi sono ideati per soddisfare necessità aziendali o organizzative. Poiché le aziende hanno molto in comune, anche i loro sistemi applicativi tendono ad avere un'architettura comune che riflette i requisiti dell'applicazione. Un'architettura applicativa generica è un'architettura per un tipo di sistema software che può essere configurato e adattato per creare un sistema che soddisfi requisiti specifici. Un progettista può utilizzare questi modelli di architetture delle applicazioni in vari modi.

- Come punto di partenza per il processo di progettazione architetturale;
- Come una lista di verifica per il progetto;
- Come un modo per organizzare il lavoro per il team di sviluppo;
- Come mezzo per valutare la riusabilità dei componenti;
- Come vocabolario per discutere delle applicazioni;

Ci sono vari tipi di sistemi applicativi che potrebbero sembrare molto diversi, tuttavia, applicazioni che all'apparenza sono molto diverse, hanno in comune molti aspetti e quindi possono condividere una struttura astratta. Vi sono diversi tipi di applicazioni:

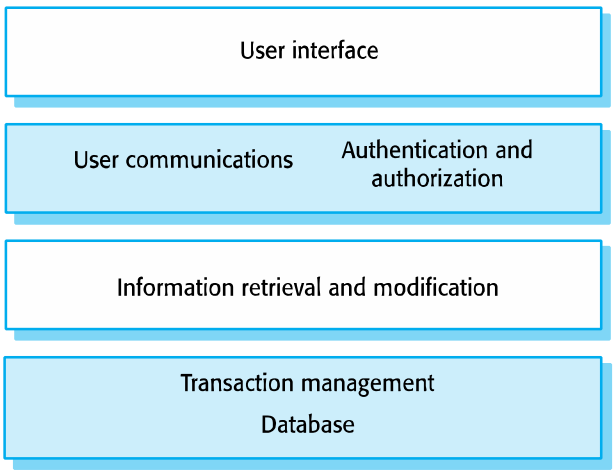
1. *Applicazioni per l'elaborazione dei dati:* basate sui dati che elaborano i dati in batch senza l'intervento esplicito dell'utente durante l'elaborazione;
2. *Applicazioni di elaborazione delle transazioni:* incentrate sui dati che elaborano le richieste degli utenti e aggiornano le informazioni in un database di sistema;
3. *Sistemi di elaborazione degli eventi:* applicazioni in cui le azioni del sistema dipendono dall'interpretazione degli eventi dall'ambiente del sistema;
4. *Sistemi di elaborazione del linguaggio:* applicazioni in cui le intenzioni degli utenti sono specificate in un linguaggio formale che viene elaborato e interpretato dal sistema.

Sistemi di elaborazione delle transazioni



Progettati per elaborare le richieste dell'utente di ricevere informazioni da un database o di aggiornarlo. Dal punto di vista di dell'utente una **transazione** è una qualsiasi sequenza coerente di operazioni che permettono di raggiungere un obiettivo, ad esempio "trovare gli orari dei voli da Londra a Parigi". Gli utenti effettuano richieste asincrone per servizi che sono processate da un gestore delle transazioni.

Sistemi informativi



Hanno una struttura generica che può essere organizzata tramite un'architettura a strati. Sono sistemi basati sulle transazioni in quanto le interazioni con questi sistemi genericamente coinvolgono le transazioni del database. Gli strati includono:

- *Interfaccia utente;*
- *Comunicazione con l'utente* (autenticazione e autorizzazione);
- *Recupero e modifica delle informazioni;*
- *Gestione delle transazioni* (Database);

I sistemi di gestione delle informazioni e delle risorse ora sono

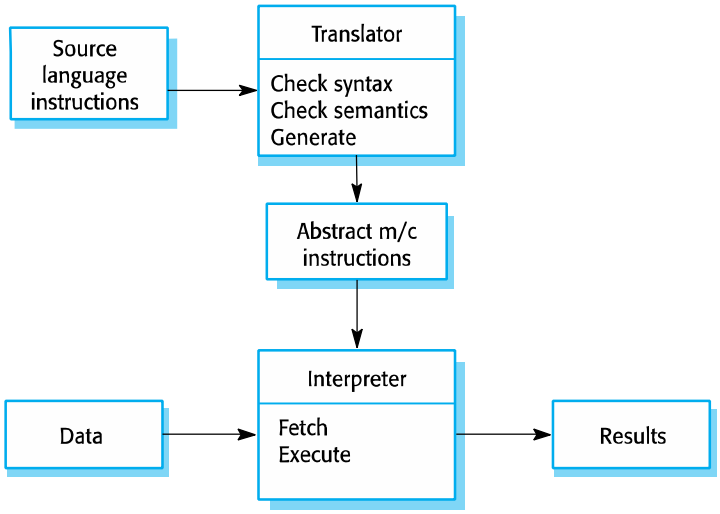
solitamente sistemi **basati sul Web** in cui le interfacce utente vengono implementate utilizzando un browser. Ad esempio, i sistemi di e-commerce sono sistemi di gestione delle risorse basati su Internet che accettano ordini elettronici di beni o servizi e quindi organizzano la consegna di questi beni o servizi al cliente. In un sistema di e-commerce, il livello specifico dell'applicazione include funzionalità aggiuntive che supportano un "carrello della spesa" in cui gli utenti possono inserire un numero di articoli in transazioni separate, quindi pagarli tutti insieme in un'unica transazione.

Questi sistemi sono spesso implementati come architetture client-server multilivello:

- Il server web è responsabile di tutte le comunicazioni dell'utente, con l'interfaccia utente implementata utilizzando un browser web;
- Il server delle applicazioni è responsabile dell'implementazione della logica specifica dell'applicazione, nonché dell'archiviazione delle informazioni e delle richieste di recupero;
- Il server del database sposta le informazioni da e verso il database e gestisce la gestione delle transazioni.

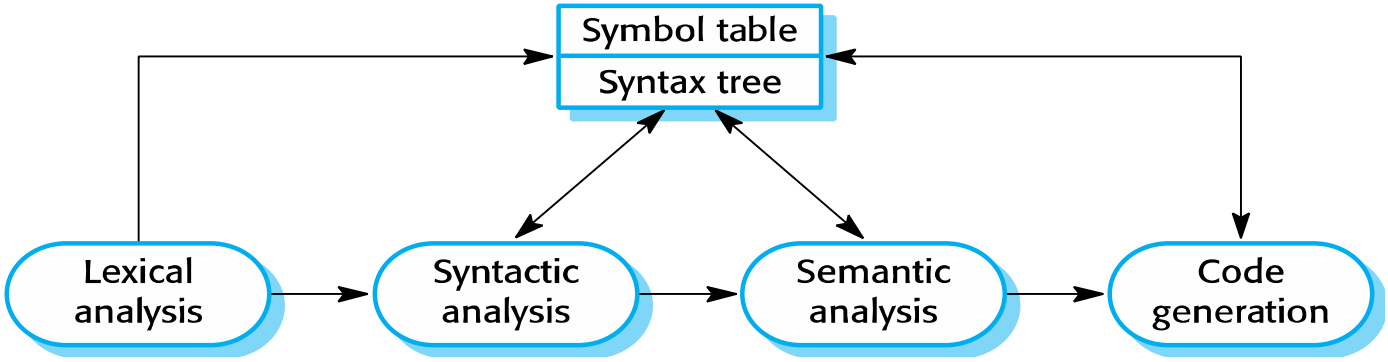
Sistemi di elaborazione dei linguaggi

Accettano un linguaggio, naturale o artificiale, come input per generare in output una rappresentazione alternativa del linguaggio e, per i linguaggi di programmazione, possono anche eseguire il codice risultante. Possono includere un interprete per agire sulle istruzioni del linguaggio che stanno venendo processate. Sono utilizzati in situazioni in cui la maniera più semplice per risolvere un problema è quella di descrivere un algoritmo o i dati del sistema.



I **compilatori** traducono un linguaggio di programmazione nel codice macchina, per molti di essi l'interprete è un'unità hardware che elabora le istruzioni macchina (e la macchina astratta è un vero processore); tuttavia per i linguaggi digitati dinamicamente (Python o Ruby), l'interprete è un componente software. I compilatori dei linguaggi di programmazione che fanno parte di un ambiente di programmazione generale hanno un'architettura generica che include i seguenti componenti:

1. Un analizzatore lessicale, che prende i *token* del linguaggio di input e li converte in una forma interna;
2. Una tabella dei simboli, che contiene informazioni riguardo i nomi delle entità (variabili, nomi delle classi e degli oggetti, ecc) utilizzate nel testo che si sta traducendo;
3. Un analizzatore sintattico, che controlla la sintassi del linguaggio che si sta traducendo, utilizza una grammatica definita del linguaggio e costruisce un albero sintattico;
4. Un albero sintattico, ovvero una struttura interna che rappresenta il programma da compilare;
5. Un analizzatore semantico, che utilizza le informazioni dell'albero sintattico e la tabella dei simboli per verificare la correttezza semantica del testo in input;
6. Un generatore di codice, che "cammina" sull'albero e genera il codice della macchina astratta;



7 - PROGETTAZIONE E IMPLEMENTAZIONE

La progettazione e l'implementazione del software sono le fasi in un processo di ingegneria del software in cui viene sviluppato un sistema software eseguibile. Queste due attività sono inevitabilmente intrecciate: la **progettazione** è un'attività creativa in cui sono identificate le componenti software del sistema e le loro relazioni in base alle richieste di un cliente, l'**implementazione** è il processo che realizza il progetto sotto forma di programma.

Una delle decisioni più importanti di implementazione che devono essere prese nella fase iniziale di un progetto è se conviene creare o acquistare (*build or buy*) il software dell'applicazione. Per molti tipi di applicazioni adesso è possibile acquistare sistemi off-the-shelf (***COTS***) che possono essere adattati alle richieste dell'utente. Quando si sviluppa un sistema applicativo riutilizzando un prodotto COTS, il processo di progettazione su come configurare il prodotto per soddisfare i requisiti dell'applicazione.

Progettazione orientata agli oggetti tramite UML

Un sistema orientato agli oggetti è formato da **oggetti interagenti** che mantengono il proprio stato locale e forniscono le operazioni su tale stato, la progettazione di tali sistemi implica lo sviluppo di un numero di modelli di sistema differenti. Questi modelli richiedono molto impegno per lo sviluppo e il mantenimento, cosa che potrebbe risultare non molto efficiente da un punto di vista economico per sistemi piccoli. Tuttavia, per grandi sistemi sviluppati da gruppi diversi, i modelli di progettazione sono un importante meccanismo di comunicazione.

Ci sono una varietà di processi di progettazione differenti orientate agli oggetti che dipendono dall'organizzazione di tale processo. Per sviluppare la progettazione di un sistema da un concetto astratto a progetto dettagliato (orientato agli oggetti), occorre:

1. *Capire e definire il contesto e le interazioni esterne con il sistema;*
2. *Progettare l'architettura del sistema;*
3. *Identificare i principali oggetti del sistema;*
4. *Sviluppare i modelli di progettazione;*
5. *Specificare le interfacce;*

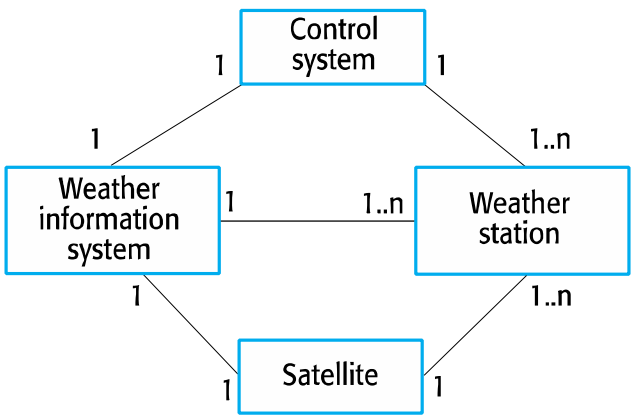
Vedremo la progettazione software orientata agli oggetti, sviluppando un progetto per la parte del software integrato in una stazione meteorologica.

Contesto del sistema e interazioni

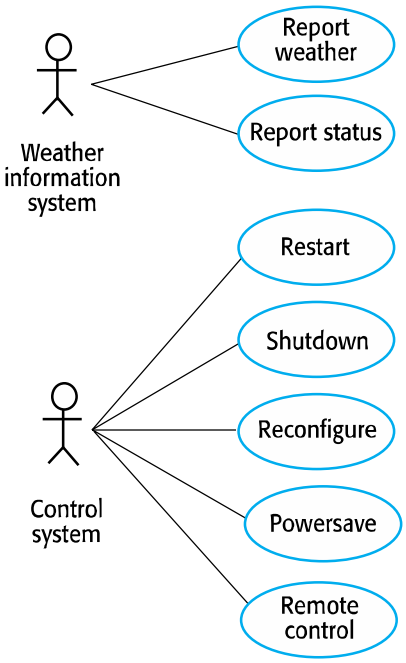
Il primo stadio in ogni processo di progettazione del software è sviluppare la **comprensione delle relazioni** tra il software che si sta progettando e il suo ambiente esterno. Questa comprensione è necessaria per decidere come fornire le funzionalità richieste per il sistema e come strutturare la comunicazione tra il sistema e il suo ambiente. Stadio contemporaneo a questo è **capire il contesto**, questo ti permette di definire i confini del sistema. Definire i confini di un sistema aiuta a scegliere quali caratteristiche devono essere implementate nel sistema che si sta sviluppando e quali invece si trovano in altri sistemi associati.

Un modello del contesto del sistema e i modelli di interazione rappresentano delle viste complementari delle relazioni tra un sistema e il suo ambiente:

- Un **modello del contesto** (1) di un sistema è un modello strutturale che descrive gli altri sistemi nell'ambiente del sistema che si sta sviluppando (rappresentato tramite associazioni che mostrano le relazioni tra entità associate);

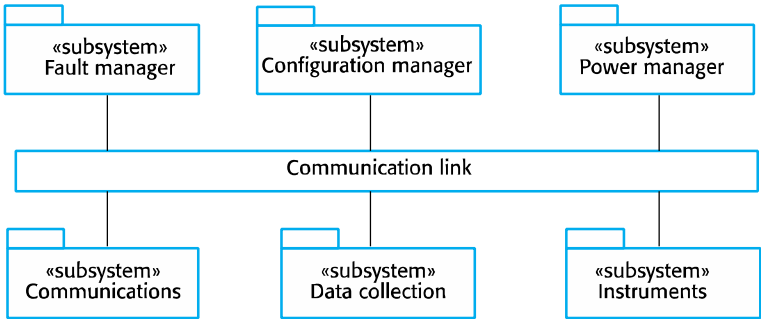


- Un **modello di interazione** è un modello dinamico che descrive come il sistema interagisce con il suo ambiente mentre viene utilizzato (rappresentato tramite un modello dei casi d'uso, ogni caso d'uso rappresenta un'interazione con il sistema, gli ellissi sono il tipo di interazione);



Progettazione architetturale

Una volta definite le interazioni tra il sistema e l'ambiente del sistema si possono utilizzare queste informazioni come base per progettare l'architettura del sistema. Occorre identificare i principali componenti che formano il sistema e le loro interazioni, poi è possibile progettare l'organizzazione del sistema utilizzando uno schema architetturale, come un modello a strati o client-server. La stazione meteorologica è composta da sottosistemi indipendenti che comunicano trasmettendo messaggi su un infrastruttura comune, indicata con *Link di comunicazione*. Ciascun sottosistema ascolta i messaggi su quella infrastruttura e raccoglie quelli che lo riguardano (questo "modello di ascolto" è comunemente utilizzato nei sistemi distribuiti). Il vantaggio chiave di questa architettura è che è facile supportare diverse configurazioni di sottosistemi, in quanto il trasmettitore di un messaggio non ha bisogno di indirizzare il messaggio a un particolare sottosistema.



Identificazione delle classi di oggetti

Solitamente l'identificazione delle classi di oggetti è la parte complicata della progettazione orientata agli oggetti. Non vi è alcuna regola o "formula magica", bisogna fare affidamento sulle proprie skills, sull'esperienza e la conoscenza del dominio del sistema. È un processo iterativo solitamente difficile da azzeccare alla prima botta. Tuttavia sono state sviluppate alcune tecniche di identificazione che ci semplificano il lavoro:

- Utilizzare *l'analisi grammaticale* di una descrizione in linguaggio naturale del sistema da costruire, gli oggetti e gli attributi sono nomi mentre le operazioni o i servizi sono verbi;
- Utilizzare *entità tangibili* (cose) del dominio di applicazione;
- Utilizzare un *approccio comportamentale* e identificare gli oggetti in base a cosa partecipa e con che comportamento;
- Utilizzare un'*analisi basata sugli scenari* (caso particolare di use-case) in cui i vari scenari di utilizzo del sistema sono identificati e analizzati uno alla volta;

Ad esempio l'identificazione delle classi di oggetto in una stazione meteorologica si basa su elementi hardware tangibili del sistema. Ho cinque classi di oggetti:

- *TermometroSuolo*, *Anemometro* e *Barometro*, oggetti del dominio applicativo che sono oggetti "hardware" in relazione diretta con gli strumenti del sistema;
- *StazioneMeteorologica*, l'interfaccia di base della stazione meteo con il suo ambiente, le sue operazioni riflettono le relazioni i casi d'uso;
- *DatiMeteorologici*, contiene i dati raccolti dagli strumenti ed elabora il comando che genere il rapporto meteorologico;

In base alle proprie competenze e conoscenze riguardo al dominio applicativo, si possono identificare altri oggetti, attributi e servizi.

Modelli di progettazione

Mostrano gli oggetti, le classi di oggetti e le relazioni tra queste entità. Vi ne sono due tipi:

- I *modelli strutturali*, descrivono la struttura statica del sistema tramite le classi di oggetti e le loro relazioni (le più importanti sono generalizzazione/ereditarietà, relazioni usa/usato - da e relazioni di composizione);
- I *modelli dinamici*, descrivono la struttura dinamica del sistema e le interazioni tra gli oggetti;

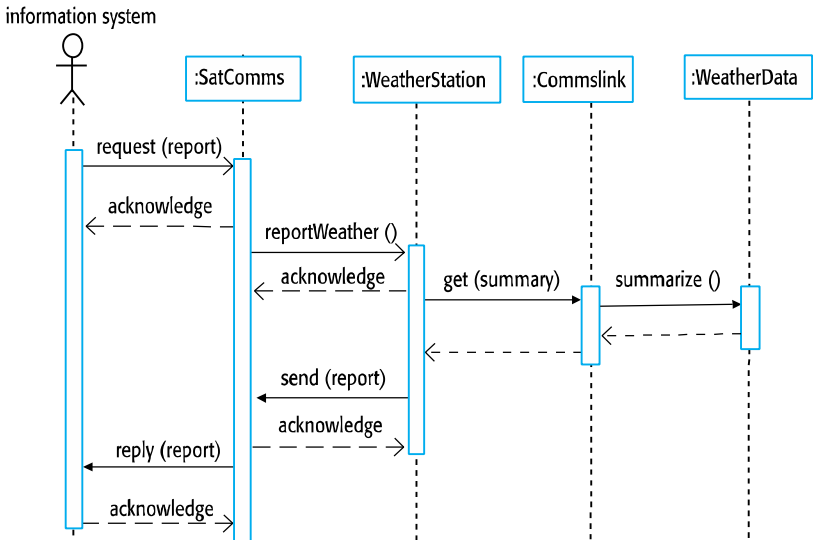
Sono tre i tipi di modelli UML particolarmente utili per aggiungere dettagli ai modelli architetturali e dei casi d'uso:

1. I *modelli di sottosistema*, che mostrano il raggruppamento logico degli oggetti in sottosistemi coerenti;
2. I *modelli di sequenza*, che mostrano la sequenza delle interazioni tra gli oggetti (focalizzano l'ordine);
3. I *modelli di macchine a stati,* che mostrano come i singoli oggetti cambiano il loro stato in risposta agli eventi (forniscono una visione interna e come ogni oggetto cambia a seguito delle interazioni con altri modelli);
4. Altri modelli che includono modelli dei casi d'uso, modelli di aggregazione, modelli di generalizzazione, ecc...;

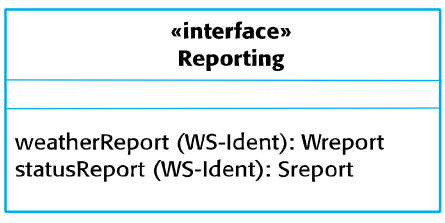
Un **modello di sottosistema** è un modello statico utile che mostra come un progetto è organizzato in gruppi di oggetti logicamente correlati. In UML sono mostrati usando i packages, un costrutto di incapsulamento. Questi sono modelli logici, l'attuale organizzazione degli oggetti nel sistema potrebbe essere differente.

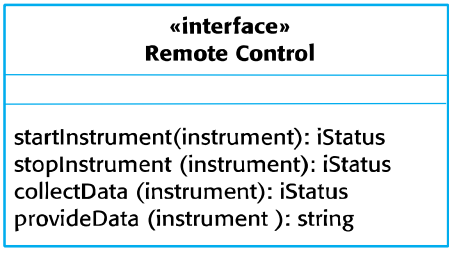
Un **modello di sequenza,** rappresentati dai diagrammi di sequenza, è un modello dinamico che descrive, per ciascuno modalità di interazione, la sequenza delle interazioni che avvengono tra gli oggetti:

- Gli *oggetti* sono disposti orizzontalmente nella parte superiore;
- Il *tempo* è rappresentato verticalmente in modo che i modelli vengano letti dall'alto verso il basso;
- Le *interazioni* sono rappresentate da frecce etichettate, diversi stili di freccia rappresentano diversi tipi di interazione;
- Un sottile rettangolo in una linea di vita di un oggetto rappresenta il *momento* in cui l'oggetto è l'oggetto di controllo nel sistema.



I **modelli di macchine a stati**, rappresentati tramite diagrammi di stato, sono usati per mostrare come gli oggetti rispondono a richieste di servizi diversi e come cambiano gli stati in funzione delle richieste. I diagrammi di stato sono utili in quanto sono modelli ad alto livello che mostrano il comportamento di sistemi o oggetti durante la loro esecuzione. Di solito non è necessario un diagramma di stato per tutti gli oggetti nel sistema. Molti degli oggetti in un sistema sono relativamente semplici e un modello a stati aggiunge dettagli non necessari al progetto.





Specifica delle interfacce

Una parte importante di ogni processo di progettazione è la specifica delle interfacce tra i componenti del progetto. Le interfacce degli oggetti devono essere

specificate in modo che gli oggetti e gli altri componenti possano essere progettati in parallelo. I progettisti dovrebbero evitare di progettare la rappresentazione dell'interfaccia, ma dovrebbero nasconderla nell'oggetto stesso. Gli oggetti possono avere diverse interfacce che sono punti di vista sui metodi forniti. UML utilizza i diagrammi delle classi per la specifica dell'interfaccia, ma può essere utilizzato anche Java.

Design pattern (schemi di progettazione)

I **design pattern** derivano dalle idee proposte da Christopher Alexander, e sono un modo di riutilizzare la conoscenza astratta riguardo un problema e la sua soluzione. Uno schema è la descrizione di un problema e l'essenza della sua soluzione, in modo che la soluzione possa essere riutilizzata in diverse impostazioni. Non è una specifica dettagliata ma più una descrizione delle conoscenze ed esperienze accumulate, una soluzione sicura a un problema comune. La descrizione degli schemi fa solitamente uso delle caratteristiche legate agli oggetti quali l'ereditarietà e il polimorfismo.

"Gli schemi e i linguaggi di schematizzazione sono modi di descrivere le migliori procedure e i progetti più validi, e di acquisire esperienza in modo che possa essere riutilizzata da altri"

I quattro elementi fondamentali degli scemi di progettazione sono:

1. Un *nome* significativo per riferirsi allo schema;
2. Una *descrizione del problema*, o meglio della sua area per sapere quando lo schema può essere applicato;
3. Un *descrizione della soluzione*, non è una descrizione concreta del progetto, bensì un modello per una soluzione che può essere instanziata in diversi modi;
4. Le *conseguenze*, i risultati e i compromessi dell'applicazione dello schema;

Per utilizzare i pattern nel proprio progetto, bisogna riconoscere che qualsiasi problema di progettazione che si sta affrontando potrebbe avere un pattern associato che può essere applicato. Ad esempio

- Indica a diversi oggetti che lo stato di qualche altro oggetto è cambiato (modello Observer).
- Riordinare le interfacce a una serie di oggetti correlati che sono stati spesso sviluppati in modo incrementale (modello di facciata).
- Fornire un modo standard per accedere agli elementi in una raccolta, indipendentemente da come tale raccolta è implementata (modello Iteratore).
- Consentire la possibilità di estendere la funzionalità di una classe esistente in fase di esecuzione (motivo Decorator).

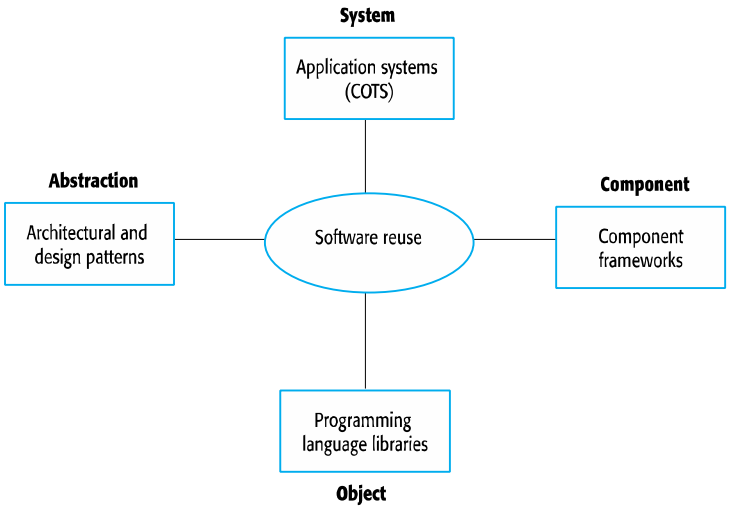
Problemi di implementazione

Non fa riferimento alla programmazione, ovviamente anch'essa è importante, bensì riguarda altri problemi di implementazione che spesso non sono trattati nei libri di programmazione; i principali aspetti dell'implementazione sono:

- *Riutilizzo,* la maggior parte dei sistemi software moderni è costruita riutilizzando componenti esistenti o interi sistemi. Quando si sviluppa un software, bisognerebbe utilizzare quanto più codice esistente possibile;
- *Gestione della configurazione,* durante il processo di sviluppo vengono create varie versioni di ciascuno componente software. Bisogna tener traccia di queste diverse versioni di ogni componente per non rischiare di includere versioni errate di un componente nel sistema;
- *Sviluppo host-target,* il software prodotto non viene solitamente eseguito sullo stesso computer su cui è stato sviluppato; solitamente viene sviluppato su un computer (host) ed eseguito su un altro (target);

Riutilizzo

Dagli anni '60 agli anni '90, la maggior parte dei nuovi software è stata sviluppata da zero, scrivendo tutto il codice in un linguaggio di programmazione ai alto livello. L'unico riutilizzo o software significativo era il riutilizzo di funzioni e oggetti nelle librerie del linguaggio di programmazione. I costi e la pressione sui tempi fanno sì che questo approccio sia diventato sempre più impraticabile, soprattutto per i sistemi commerciali e basati su Internet. È emerso un approccio allo sviluppo basato sul riutilizzo del software esistente, oggi generalmente utilizzato per il software aziendale e scientifico.



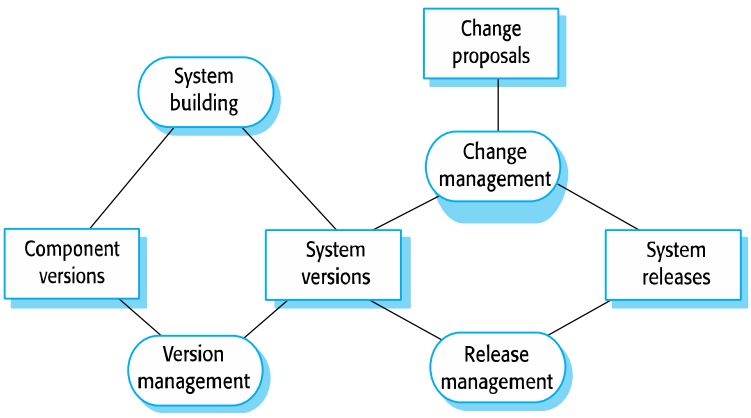
Il riutilizzo del software è possibile a vari livelli:

1. *Livello di astrazione*, a questo livello non viene riutilizzato direttamente il software ma le conoscenze di astrazione nella progettazione software;
2. *Livello degli oggetti*, a questo livello si riutilizzano direttamente gli oggetti da una libreria anziché scrivere il codice;
3. *Livello dei componenti*, i componenti sono raccolte di oggetti e classi di oggetti che operano insieme per fornire funzioni e servizi, quando si vanno a riutilizzare molte volte bisognerà adattare ed estendere il componente con un po' di nostro codice;
4. *Livello del sistema*, a questo livello si riutilizzano interi sistemi applicativi;

Tuttavia anche il riutilizzo comporta dei costi :

- Il tempo impiegato per cercare il software da riutilizzare e stabilire se soddisfa le nostre esigenze;
- Il costo di acquisto del software riutilizzabile, per grandi sistemi off-the-shelf potrebbe essere elevato;
- I costi per adattare e configurare i componenti o i sistemi del software riutilizzabile per soddisfare i requisiti del sistema che sta venendo sviluppato;
- I costi per integrare gli elementi del software riutilizzabile tra loro e con il nuovo codice sviluppato;

Gestione della configurazione



Gestione della configurazione è il nome dato al processo generale di gestione di un software che cambia. Lo scopo della gestione della configurazione è supportare il processo di integrazione del sistema in modo che tutti gli sviluppatori possano accedere al codice e ai documenti del progetto in modo controllato, trovare le modifiche che sono state apportate, compilar e collegare i componenti per creare un sistema. Sono quattro le attività fondamentali nel processo di gestione della configurazione:

1. *Gestione delle versioni*, dove viene fornito il supporto per tenere traccia delle diverse versioni dei componenti software. I sistemi di gestione delle versioni includono funzionalità per coordinare lo sviluppo da parte di diversi programmatori;

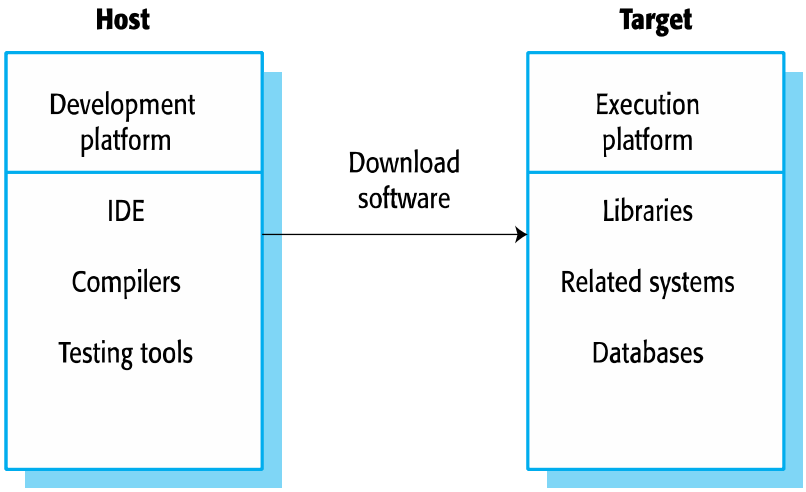
2. *Integrazione del sistema*, dove viene fornito supporto per aiutare gli sviluppatori a definire quali versioni dei componenti vengono utilizzate per creare ciascuna versione di un sistema. Questa descrizione viene quindi utilizzata per costruire automaticamente un sistema compilando e collegando i componenti richiesti;

3. *Monitoraggio dei problemi*, dove viene fornito supporto per consentire agli utenti di segnalare bug e altri problemi e per consentire a tutti gli sviluppatori di vedere chi sta lavorando su questi problemi e quando vengono risolti;

4. *Gestione delle release*, dove le nuove versioni di un sistema software vengono rilasciate ai clienti, si occupa di pianificare le funzionalità delle nuove release e di organizzare la distribuzione del software;

Sviluppo host-target

La maggior parte del software è sviluppata su un computer (host), ma viene eseguita su una macchina separata (target). Più in generale, possiamo parlare di una piattaforma di sviluppo e di una piattaforma di esecuzione. Una piattaforma è più di un semplice hardware, include il sistema operativo installato più altro software di supporto come un sistema di gestione del database o, per le piattaforme di sviluppo, un ambiente di sviluppo interattivo.



La piattaforma di sviluppo di solito ha un software installato diverso dalla piattaforma di esecuzione; queste piattaforme possono avere architetture differenti. Una piattaforma di sviluppo del software dovrebbe fornire una serie di strumenti che supportano i processi di ingegneria del software, come ad esempio:

- Un *compilatore integrato* e un *sistema di editing* guidato dalla sintassi che consente di creare, modificare e compilare il codice;
- Un *sistema di debugging* del linguaggio;
- *Strumenti di editing grafico*, come quelli per modificare il modelli UML;
- *Strumenti di test*, come JUnit, che possono eseguire automaticamente una serie di test sulla nuova versione di un programma;
- *Strumenti che supportano il re factoring* e la visualizzazione dei programmi;
- *Strumenti per la gestione della configurazione* per gestire le versioni del codice e per integrare e compilare i sistemi;

Gli odierni strumenti di sviluppo del software, di solito, sono installati all'interno di un ambiente di sviluppo integrato (*Integrated Development Environment, **IDE***). Un IDE è un insieme di strumenti software che supporta diversi aspetti dello sviluppo del software, all'interno di un framework e di un'interfaccia utente comuni. Gli IDE vengono creati per supportare lo sviluppo in un linguaggio di programmazione specifico come Java. L'IDE del linguaggio può essere sviluppato appositamente o può essere un'istanza di un IDE generico, con strumenti di supporto linguistico specifici.

Come parte del processo di sviluppo bisogna decidere come il software sviluppato sarà rilasciato nella piattaforma target. Questo e semplice nel caso di sistemi integrati, dove la piattaforma target è solitamente un singolo computer; per sistemi distribuiti bisognerà scegliere le piattaforme specifiche sulle quali saranno rilasciati i componenti. I temi da affrontare per prendere queste decisioni sono:

1. *Requisiti hardware e software di un componente*, se un componente è progettato per un'architettura hardware specifica o si basa su un altro sistema software, deve ovviamente essere distribuito su una piattaforma che fornisce il supporto hardware e software richiesto;

2. *Requisisti di disponibilità del sistema*, i sistemi ad alta disponibilità possono richiedere la distribuzione dei componenti su più di una piattaforma. Ciò significa che, in caso di guasto della piattaforma, è disponibile un'implementazione alternativa del componente;

3. *Comunicazione tra i componenti,* se c'è un alto livello di traffico di comunicazioni tra i componenti, di solito ha senso distribuirli sulla stessa piattaforma o su piattaforme fisicamente vicine l'una all'altra. Ciò riduce il ritardo tra il tempo in cui un messaggio viene inviato da un componente e ricevuto da un altro.

12 - INGEGNERIA DELLA SICUREZZA

La **sicurezza** è una proprietà di un sistema che riflette la capacità del sistema di funzionare, normalmente o in modo anomalo, senza il pericolo di causare lesioni alle persone o morte e senza danni all'ambiente del sistema. È importante considerare la sicurezza del software poiché la maggior parte dei dispositivi il cui guasto è critico ora incorporano sistemi di controllo basati su software.

I software nei sistemi a sicurezza critica svolge un duplice ruolo nella realizzazione della sicurezza:

1. Il sistema può essere controllato dal software in modo che le decisioni prese dal software e le azioni successive siano critiche per la sicurezza. Pertanto, il comportamento del software è direttamente correlato alla sicurezza complessiva del sistema.

2. Il software è ampiamente utilizzato per controllare e monitorare altri componenti critici per la sicurezza in un sistema. Ad esempio, tutti i componenti dei motori degli aeromobili vengono monitorati da un software alla ricerca di indicazioni precoci di guasto dei componenti. Questo software è fondamentale per la sicurezza perché, se si guasta, altri componenti potrebbero guastarsi e causare un incidente.

L'**affidabilità** riguarda la conformità a una determinata specifica e l'erogazione del servizio. La **sicurezza** si occupa di garantire che il sistema non possa causare danni indipendentemente dal fatto che sia conforme o meno alle sue specifiche. L'affidabilità di un sistema software è necessaria ma non sufficiente, questo perché sicurezza e affidabilità sono due aspetti connessi ma distinti (un sistema affidabile non è necessariamente un sistema sicuro e viceversa).

I sistemi software che sono affidabili potrebbero non essere sicuri per quattro motivi:

1. In un sistema possono esserci *errori dormienti* che non vengono rilevati per molti anni e che si verificano solo raramente;

2. *Errori di specifica*, se le specifiche del sistema non sono corrette, il sistema può comportarsi come specificato ma comunque causare un incidente;

3. *Errori hardware* che generano input spuri, difficile da anticipare nelle specifiche;

4. *Comandi sensibili al contesto*, ovvero l'emissione del comando giusto al momento sbagliato, spesso il risultato di un errore dell'operatore;

Sistemi a sicurezza critica

I **sistemi a sicurezza critica** sono sistemi nei quali è essenziale che le operazioni siano sempre sicure, in altre parole il sistema non dovrebbe mai arrecare danni alle persone o all'ambiente. Possono essere classificati in due categorie:

1. *Software primario a sicurezza critica*, software integrato come controllore in un sistema il cui malfunzionamento può provocare un malfunzionamento dell'hardware che potrebbe danneggiare le persone e l'ambiente (esempio il sistema della pompa di insulina);

2. *Software secondario a sicurezza critica*, questo software può provocare danni in modo indiretto, ossia in cui il guasto si traduce in guasti in altri sistemi (socio-tecnici), che possono quindi avere conseguenze sulla sicurezza. Ad esempio, il sistema Mentcare è fondamentale per la sicurezza poiché il fallimento può portare alla prescrizione di un trattamento inappropriato. Anche i sistemi di controllo dell'infrastruttura sono sistemi secondari critici per la sicurezza.

I rischi (**hazards**) sono stati del sistema che possono portare a un incidente. Lo sviluppo di sistemi a sicurezza critica utilizza tecniche guidate dai rischi (*hazard-driven*), che considerano i potenziali incidenti che possono capitare a un sistema software:

- Evitare i rischi (*hazard avoidance*), il sistema è progettato in modo da evitare i rischi (quelli più semplici e prevedibili);

- Identificare ed eliminare i rischi (*hazard detection and removal*), il sistema è progettato in modo che i rischi sono individuati e rimossi prima che possano provocare incidenti;

- *Limitazione dei danni*, il sistema potrebbe includere delle funzioni di protezione che minimizzano i danni che un incidente potrebbe causare.

Term	Definition
Accident (or mishap)	An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment. An overdose of insulin is an example of an accident.
Hazard	A condition with the potential for causing or contributing to an accident. A failure of the sensor that measures blood glucose is an example of a hazard.
Damage	A measure of the loss resulting from a mishap. Damage can range from many people being killed as a result of an accident to minor injury or property damage. Damage resulting from an overdose of insulin could be serious injury or the death of the user of the insulin pump.
Hazard severity	An assessment of the worst possible damage that could result from a particular hazard. Hazard severity can range from catastrophic, where many people are killed, to minor, where only minor damage results. When an individual death is a possibility, a reasonable assessment of hazard severity is 'very high'.
Hazard probability	The probability of the events occurring which create a hazard. Probability values tend to be arbitrary but range from 'probable' (say 1/100 chance of a hazard occurring) to 'implausible' (no conceivable situations are likely in which the hazard could occur). The probability of a sensor failure in the insulin pump that results in an overdose is probably low.
Risk	This is a measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the hazard severity, and the probability that the hazard will lead to an accident. The risk of an insulin overdose is probably medium to low.

Gli incidenti in sistemi complessi raramente hanno una sola causa poiché questi sistemi sono progettati per resistere a un singolo punto di guasto. Progettare sistemi in modo che un singolo punto di guasto non provochi un incidente è un principio fondamentale della progettazione di sistemi sicuri. Quasi tutti gli incidenti sono il risultato di combinazioni di malfunzionamenti piuttosto che di singoli guasti. È probabile che sia impossibile anticipare tutte le combinazioni di problemi, specialmente nei sistemi controllati da software, quindi è impossibile ottenere la sicurezza completa. Gli incidenti sono inevitabili.

Sebbene il controllo software possa introdurre più modi in cui un sistema può fallire, tuttavia esso consente un monitoraggio e una protezione migliori, quindi contribuisce a migliorare la sicurezza nei sistemi.

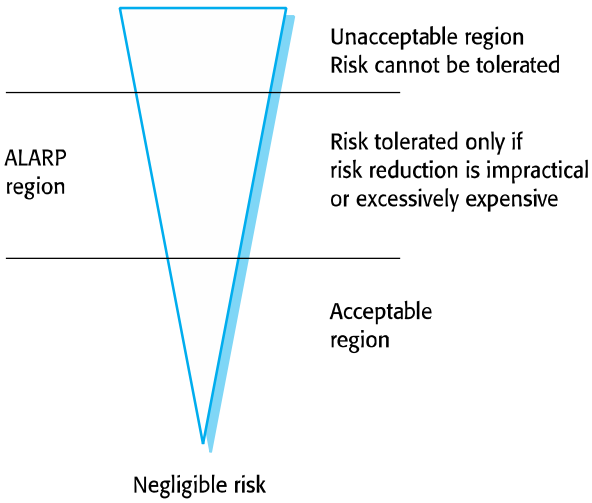
Requisiti

L'obiettivo dell'ingegneria dei requisiti di sicurezza è identificare i requisiti di protezione che garantiscono che i guasti del sistema non causino lesioni o morte o danni ambientali. I requisiti di sicurezza possono essere requisiti "non devono", ovvero definiscono situazioni ed eventi che non dovrebbero mai verificarsi, ossia il comportamento che non è accettabile da parte del sistema. Questi requisiti “non deve” non possono essere implementati direttamente, devono essere scomposti in requisiti funzionali più specifici. I requisiti di sicurezza funzionale definiscono:

- Controllo e ripristino delle funzionalità che dovrebbero essere incluse in un sistema;
- Funzionalità che forniscono protezione contro errori di sistema e attacchi esterni;

La specifica dei requisiti basata sui rischi è un approccio generale in cui vengono identificati non solo i rischi cui sono sottoposti i sistemi, ma anche i requisiti per evitare o ridurre tali rischi. Il processo di specifica della sicurezza guidata dai rischi è composto da quattro attività:

1. *Identificazione dei rischi*, identificare i potenziali rischi di un sistema che vengono poi registrati in un apposito registro, che è un documento contenente le analisi e le valutazioni sulla sicurezza. Vi sono diversi tipi di rischi: fisici, elettrici, biologici, fallimento dei servizi e molti altri;
2. *Valutazione dei rischi*, questa attività decide quali rischi sono più pericolosi e/o frequenti. I rischi possono essere categorizzati come:
    a. *Intollerabili*, sono rischi che minacciano la vita umana, il sistema deve dunque essere progettato per evitare il verificarsi di questi rischi o, dove questo non fosse possibile, minimizzare al massimo i danni;
    b. *Basso o con conseguenze accettabili (ALARP)*, sono rischi con conseguenze lievi oppure gravi ma con bassa probabilità di realizzazione;
    c. *Accettabile*. Le conseguenze del rischio sono accettabili e non dovrebbero essere sostenuti costi aggiuntivi per ridurre la probabilità di rischio;



    L’accettabilità di un rischio è determinata da considerazioni umane, sociali e politiche, dunque la pericolosità dei rischi è “soggettiva”. Poiché la pericolosità dei rischi è soggettiva, la cosa importante da fare è cercare il più possibile di ridurre al minimo la frequenza di questi;
3. *Analisi dei rischi*, è il processo che analizza le cause principali dei rischi e gli eventi che possono concretizzarli. Per farlo si possono applicare tecniche deduttive top-down (partono dal rischio e cercano di identificare i possibili fallimenti del sistema, albero dei guasti) o tecniche induttive bottom-up (partono da un possibile fallimento del sistema e cercano di identificare quali rischi potrebbero derivare da tale fallimento);
4. *Riduzione dei rischi,*questo processo si basa sul risultato dell’analisi dei rischi e permette di stabilire i requisiti di sicurezza. Questi requisiti definiscono le tecniche per evitare i rischi o impedire che questi degenerino in un incidente o, in caso avvenga, ridurre al minimo i danni derivanti da quell’incidente. Ci sono tre possibili strategie:
    a. *Evitare i rischi*, il sistema è progettato in modo che il rischio non possa presentarsi;
    b. *Individuazione e rimozione dei rischi*, il sistema è progettato in modo che i rischi siano individuati e neutralizzati prima che possano causare incidenti;

c. *Limitazione dei danni*, il sistema viene progettato in modo da minimizzare le conseguenza di un incidente;

Normalmente i progettisti di sistemi critici usano un misto di queste strategie.

Processi

I processi utilizzati per sviluppare il software a sicurezza critica si basano sui processi utilizzati nell'ingegneria dell'affidabilità del software. Hanno un approccio basato sul piano con revisioni e controlli in ogni fase del processo, l'obiettivo generale è quello di prevenire e rilevare i guasti. Bisogna includere anche revisioni sulla sicurezza e identificazione e monitoraggio espliciti dei pericoli.

Alcuni sistemi a sicurezza critica sono **regolamentati,** gli enti di regolamentazione richiedono prove dettagliate sulla sicurezza dei sistemi e che il processo di ingegneria della sicurezza sia stato usato durante lo sviluppo del sistema. Queste prove possono includere:

- La specifica del sistema che è stato sviluppato e le registrazioni dei controlli effettuati su quella specifica.
- Prova dei processi di verifica e validazione che sono stati effettuati e dei risultati della verifica e validazione del sistema.
- Prova che le organizzazioni che sviluppano il sistema hanno processi software definiti e affidabili che includono revisioni della garanzia di sicurezza. Devono anche esserci registrazioni che dimostrano che questi processi sono stati attuati correttamente.

I metodi agili non vengono solitamente utilizzati per l'ingegneria dei sistemi critici per la sicurezza. Per la regolazione del sistema è necessaria un'ampia documentazione di processo e di prodotto. Contrasta l'attenzione dei metodi agili sul software stesso. È importante un'analisi dettagliata della sicurezza di una specifica di sistema completa, contrasta lo sviluppo interlacciato di una specifica di sistema e di un programma. È possibile utilizzare alcune tecniche agili come lo sviluppo basato sui test.

La garanzia del processo implica la definizione di un processo affidabile e la garanzia che questo processo venga seguito durante lo sviluppo del sistema. La garanzia di processo si concentra su:

- Abbiamo i processi giusti? I processi sono appropriati per il livello di affidabilità richiesto. Dovrebbe includere la gestione dei requisiti, la gestione delle modifiche, le revisioni e le ispezioni, ecc.
- Stiamo eseguendo correttamente i processi? Questi processi sono stati seguiti dal team di sviluppo.

La garanzia del processo genera documentazione, i processi agili sono quindi usati raramente per i sistemi critici.

Processi per assicurare la sicurezza

La garanzia del processo è importante per lo sviluppo di sistemi critici per la sicurezza, gli incidenti sono eventi rari, quindi i test potrebbero non trovare tutti i problemi, inoltre, i requisiti di sicurezza a volte sono requisiti "non devono" quindi non possono essere dimostrati mediante test. Le attività di garanzia della sicurezza possono essere incluse nel processo del software che registra le analisi che sono state effettuate e le persone responsabili di queste. La responsabilità personale è importante in quanto i guasti del sistema possono portare a successive azioni legali.

Alcuni esempi di attività di garanzia della sicurezza sono:

- *Creazione di un sistema di registrazione e monitoraggio dei pericoli;*
- *Nomina di ingegneri per la sicurezza del progetto che hanno responsabilità esplicita per la sicurezza del sistema;*
- *Ampio uso di revisioni sulla sicurezza;*

- *Creazione di un sistema di certificazione della sicurezza in cui la sicurezza dei componenti critici è formalmente certificata;*
- *Gestione dettagliata della configurazione;*

L'analisi dei rischi è una parte essenziale dello sviluppo dei sistemi a sicurezza critica, essa richiede l'identificazione dei rischi e delle loro cause. Dovrebbe esserci una chiara tracciabilità dai rischi identificati attraverso la loro analisi fino alle azioni intraprese durante il processo per garantire che questi rischi siano stati coperti. Un registro dei rischi può essere utilizzato per tener traccia dei rischi durante il processo.

I revisori della sicurezza dovrebbero essere guidati dal registro dei rischi, per ogni rischio identificato, un team di revisori esamina il sistema e giudica se esso può affrontare un rischio in modo sicuro.

Quando si incorporano dei componenti esterni in un sistema a sicurezza critica c'è bisogno della certificazione della garanzia della qualità.

Verifica

I metodi formali per lo sviluppo del software si basano su un modello formale del sistema che opera come una specifica del sistema. Questi metodi formali possono essere usati quando è prodotta una specifica matematica del sistema. Sono la tecnica di verifica statica definitiva che può essere utilizzata in diverse fasi del processo di sviluppo:

- Una specifica formale può essere sviluppata e analizzata matematicamente per verificarne la coerenza. Questo aiuta a scoprire errori e omissioni nelle specifiche.
- Possono essere sviluppati argomenti formali che dimostrano che un programma è conforme alle sue specifiche matematiche. Questo è efficace per scoprire errori di programmazione e progettazione.

I metodi formali presentano dei pro:

1. La produzione di una specifica matematica richiede un'analisi dettagliata dei requisiti e questo potrebbe rivelare errori;
2. I sistemi concorrenti possono essere analizzati per scoprire condizioni di competizione che potrebbero portare a un deadlock. Il test per tali problemi è molto difficile;
3. Possono rilevare errori di implementazione prima del test quando il programma viene analizzato insieme alle specifiche;

Ma allo stesso tempo presentano delle problematiche:

1. Richiedono notazioni specializzate che non possono essere comprese dagli esperti di dominio;
2. Molto costoso sviluppare una specifica e ancora più costoso dimostrare che un programma soddisfa tale specifica;
3. Le prove possono contenere errori;
4. Potrebbe essere possibile raggiungere lo stesso livello di fiducia in un programma in modo più economico utilizzando altre tecniche di V&V.

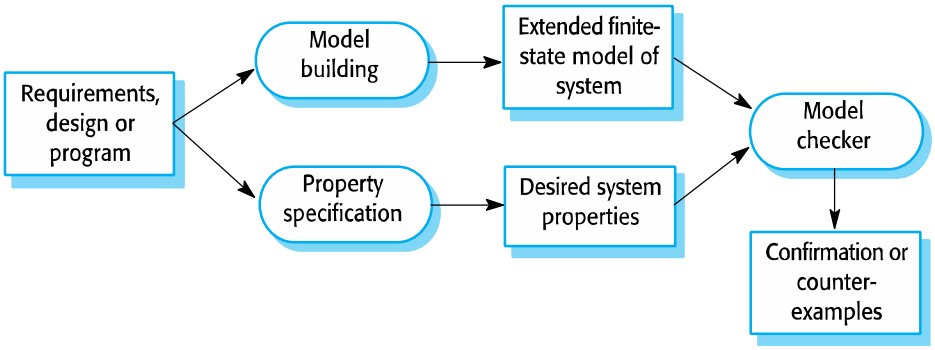
Tuttavia, i metodi formali non possono garantire la sicurezza del sistema:

- *La specifica potrebbe non riflettere i requisiti reali degli utenti del sistema.* Gli utenti raramente comprendono le notazioni formali, quindi non possono leggere direttamente le specifiche formali per trovare errori e omissioni;

- *La prova può contenere errori*. Le prove di programma sono grandi e complesse, quindi, come i programmi grandi e complessi, di solito contengono errori;

- *La prova può fare supposizioni errate sul modo in cui viene utilizzato il sistema*. Se il sistema non viene utilizzato come previsto, il comportamento del sistema esula dall'ambito della dimostrazione;

La **verifica dei modelli** implica la creazione di un modello a stati finiti esteso di un sistema e, utilizzando un sistema specializzato (un verificatore di modelli), il controllo di errori in quel modello. Il controllo del modello esplora tutti i possibili percorsi attraverso il modello e controlla che una proprietà specificata dall'utente sia valida per ogni percorso,inoltre è particolarmente utile per verificare i sistemi concorrenti, che sono difficili da testare. Sebbene il controllo del modello sia molto costoso dal punto di vista computazionale, ora è pratico utilizzarlo nella verifica di sistemi critici di piccole e medie dimensioni.



Analisi di sicurezza

Gli analizzatori statici automatici sono strumenti software che esaminano il testo sorgente di un programma e rilevano i possibili errori e anomalie. Analizzano il testo sorgente e cercano di scoprire condizioni potenzialmente errate e di portarle all'attenzione del team di V&V. Sono molto efficaci come ausilio alle ispezioni: sono un supplemento ma non un sostituto delle ispezioni. L'obiettivo dell'analisi statica automatica è quello di focalizzare l'attenzione di un lettore di codice sulle anomalie di un programma (variabili usate senza essere state inizializzate, variabili inutilizzate, ecc).

Sono tre i livelli di verifica che possono essere implementati negli analizzatori statici:

1. *Verifica degli errori caratteristici*, l'analizzatore statico può analizzare il codice per cercare schemi che sono tipici di un particolare problema (realizzato dai programmatori utilizzando quel linguaggio) e li segnala al programmatore;

2. *Verifica degli errori definiti dall'utente,* gli utenti di un linguaggio di programmazione definiscono schemi di errori da rilevare, andando ad ampliare i tipi di errore che possono essere individuati. Questi tipi di errore sono correlati al dominio delle applicazioni oppure si basano sulla conoscenza del particolare sistema che si sta sviluppando;

3. *Verifica delle asserzioni*, gli sviluppatori includono asserzioni formali nei loro programmi che stabiliscono le relazioni che devono essere rispettate in quel punto del programma. L'analizzatore esegue simbolicamente il codice e mette in evidenza le istruzioni nelle quali tale asserzione non è vera;

L'analisi statica è efficace nel trovare gli errori nei programmi, ma di solito genera un gran numero di falsi positivi (sezioni di codice senza errori ma vengono rilevati possibili errori a causa delle regole dell'analizzatore). Particolarmente utile quando viene utilizzato un linguaggio come il C che ha una digitazione debole e quindi molti errori non vengono rilevati dal compilatore. È particolarmente utile per il controllo della sicurezza: l'analizzatore statico può rilevare aree di vulnerabilità come overflow del buffer o input non controllati. L'analisi statica viene ora utilizzata di routine nello sviluppo di molti sistemi critici per la sicurezza.

Casi si sicurezza

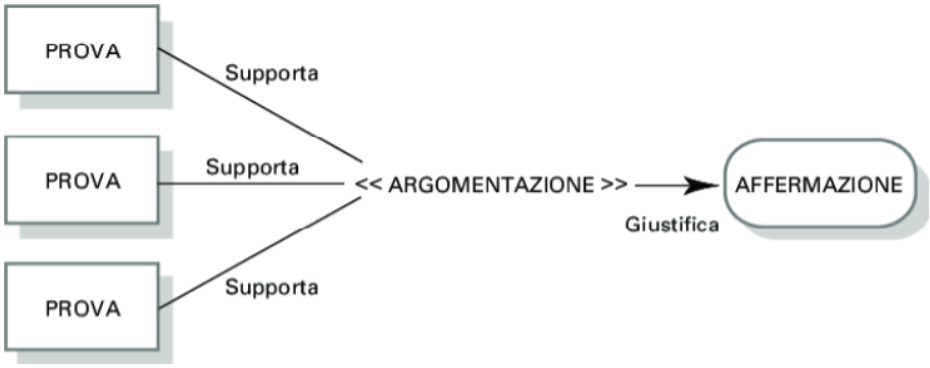
I casi di sicurezza e affidabilità sono documenti strutturati che espongono argomenti dettagliati e prove che è stato raggiunto un livello richiesto di sicurezza o affidabilità. Normalmente sono richiesti dalle autorità di regolamentazione prima che un sistema possa essere certificato per l'uso operativo. La responsabilità del

regolatore è verificare che un sistema sia sicuro o affidabile per quanto pratico. Regolatori e sviluppatori lavorano insieme e negoziano ciò che deve essere incluso in un caso di sicurezza / affidabilità del sistema.

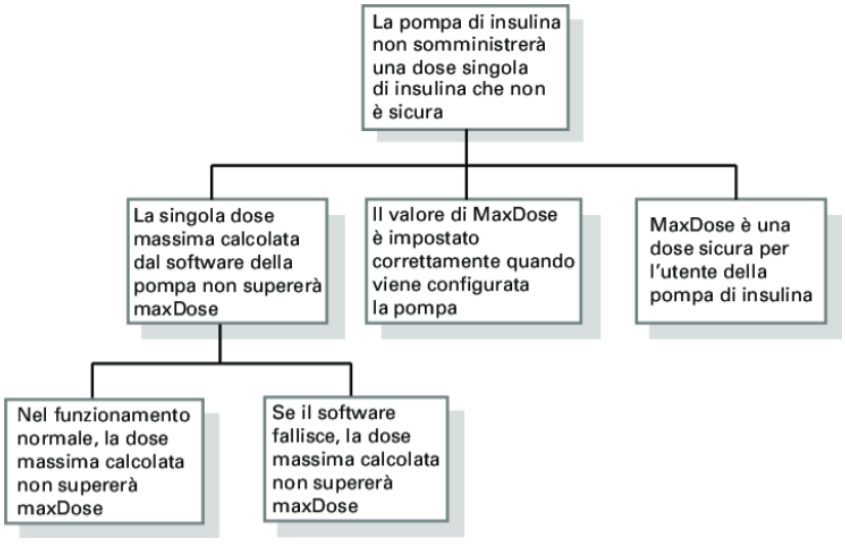
Un **caso di sicurezza** è un insieme di prove documentate, che fornisce un argomento convincente e valido che un sistema è adeguatamente sicuro per una data applicazione in un dato ambiente. Gli argomenti in un caso di sicurezza possono essere basati su prove formali, motivazioni di progettazione, prove di sicurezza, ecc. (possono essere inclusi anche fattori di processo). Un caso di sicurezza del software fa solitamente parte di un caso di sicurezza del sistema più ampio che tiene conto dei problemi hardware e operativi. Essendo documenti estesi e complessi sono anche costosi da produrre e da mantenere e questo obbliga gli sviluppatori a tenere in considerazione i requisiti del caso di sicurezza durante il processo di sviluppo. Le modifiche del sistema e le conseguenti revisioni aumentano i costi dei casi di sicurezza, questo perché quando viene apportata una modifica al software o hardware del sistema bisogna dimostrare che la sicurezza del sistema non è stata influenzata dalla modifica.

Argomentazioni

Per stabilire se un sistema è sicuro da utilizzare, ci si deve basare su **argomentazioni logiche**, queste dovrebbero dimostrare che le prove presentate supportano le affermazioni sulla sicurezza e sulla fidatezza di un sistema. Queste affermazioni possono essere *assolute* o *probabilistiche*, un'argomentazione collega la prova all'affermazione. Un'argomentazione è una relazione tra ciò che si pensa sia il caso (affermazione) e le prove che sono state raccolte; spiega essenzialmente perché l'affermazione (che è un'asserzione sulla sicurezza/fidatezza del sistema) può essere dedotta dalle prove disponibili.



Gli argomenti strutturati dimostrano che un sistema soddisfa i propri obblighi di sicurezza, non è necessario dimostrare che il programma funziona come previsto; l'obiettivo è semplicemente quello di dimostrare la sicurezza. Di norma si fanno molte affermazioni sulla sicurezza di un sistema; spesso la validità di un'affermazione dipende dalla validità di altre affermazioni. È possibile organizzare le affermazioni in modo gerarchico. Inizi dalle foglie della gerarchia e dimostri sicurezza, ciò implica che le affermazioni di livello superiore sono vere.

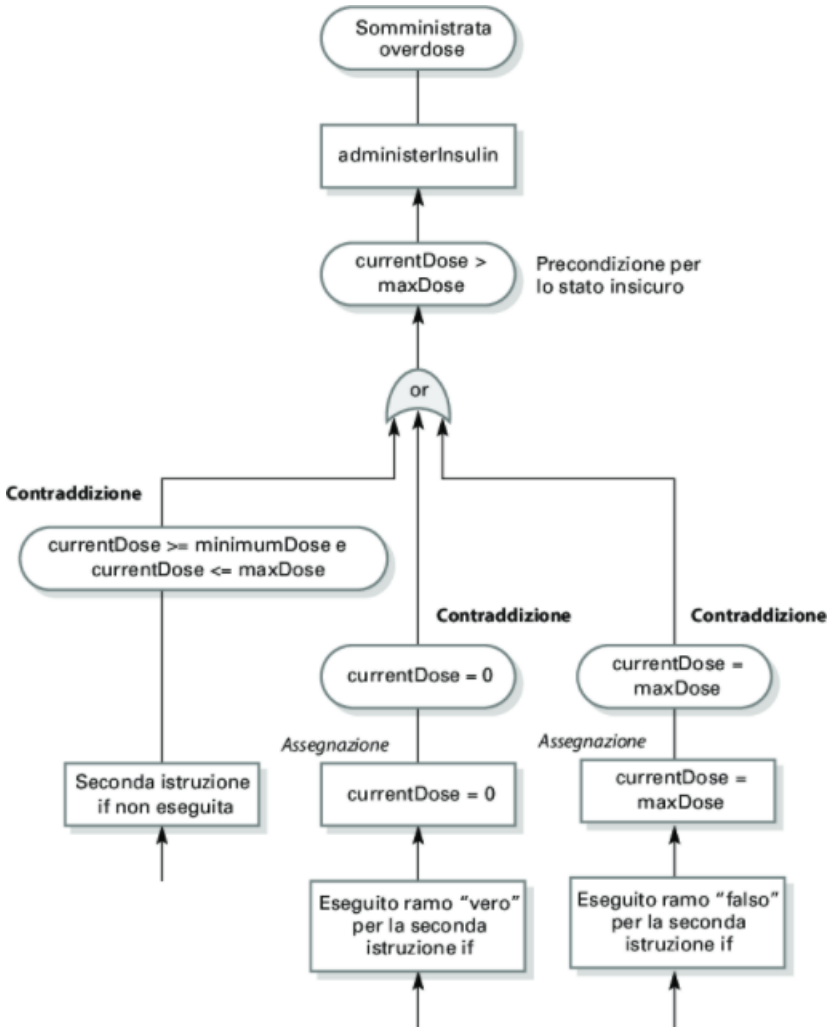


Le argomentazioni sulla sicurezza del software sono un tipo di argomentazioni strutturate che dimostrano che un programma soddisfa i requisiti della sicurezza. In un argomentazione sulla sicurezza non è necessario dimostrare che il programma non opera come previsto; è sufficiente dimostrare che l'esecuzione di un programma non potrà portare il sistema in uno stato potenzialmente non sicuro. Queste sono più deboli di quelle sulla correttezza dei sistemi che devono mostrare che il codice di sistema è conforme alle sue specifiche.

Sono generalmente basate su prove per contraddizione/ per assurdo:

1. Supponiamo che sia possibile raggiungere uno stato pericoloso;

2. Si scrive un predicato (espressione logica) che definisce questo stato non sicuro;
3. Si dimostra che, per tutti i percorsi che conducono a tale stato, la condizione finale di questi percorsi (anche questa definita predicato) contraddice il predicato dello stato non sicuro. Se questo è vero, si può dedurre che l'ipotesi iniziale non è vera;
4. Si ripete questa procedura per tutti i rischi che sono stati identificati e allora si ha la prova che il sistema è sicuro;



Può essere sviluppato un modello grafico dell'argomento sicurezza.

13 - IGNEGNERIA DELLA PROTEZIONE

L'ingegneria della protezione è un insieme di strumenti, tecniche e metodi per supportare lo sviluppo e la manutenzione di sistemi in grado di resistere ad attacchi dannosi intesi a danneggiare un sistema basato su computer o i suoi dati. È un sottocampo del più ampio campo della sicurezza informatica.

Per progettare sistemi protetti occorre tenere in considerazione tre fattori essenziali:

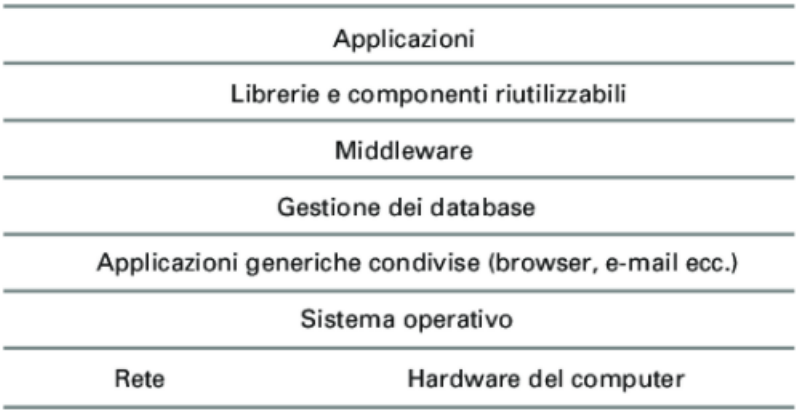
1. *Riservatezza*, la informazioni in un sistema potrebbero diventare accessibili a persone o programmi che non sono autorizzati ad aver accesso a tali dati;
2. *Integrità*, la informazioni in un sistema possono danneggiarsi, diventando inutilizzabili o inaffidabili;
3. *Disponibilità*, l'accesso a un sistema o ai suoi dati potrebbe diventare impossibile;

Questi tre fattori sono strettamente correlati. Da un punto di vista organizzativo , la protezione deve essere considerata a tre livelli:

1. *Protezione dell'infrastruttura*, che si occupa di mantenere la sicurezza di tutti i sistemi e le reti che forniscono un'infrastruttura e una serie di servizi condivisi all'organizzazione;

2. *Protezione delle applicazioni*, che riguarda la sicurezza dei singoli sistemi applicativi o dei relativi gruppi di sistemi;

3. *Protezione operativa*, che riguarda il funzionamento e l'uso sicuri dei sistemi dell'organizzazione;



Questo schema rappresenta i livelli di un sistema dove può essere compromessa la protezione. La maggior parte degli attacchi sono rivolti alle infrastrutture software dei sistemi in quanto, le loro componenti (come i browser), sono disponibili a tutti. La **protezione delle applicazioni** è un problema di ingegneria del software in cui il sistema è progettato per resistere agli attacchi. La **protezione dell'infrastruttura** è un problema di gestione dei sistemi in cui l'infrastruttura è configurata per resistere agli attacchi. Il fulcro di questo capitolo è la sicurezza delle applicazioni piuttosto che la sicurezza dell'infrastruttura.

La gestione della protezione dei sistemi include varie attività:

- *Gestione degli utenti e delle autorizzazioni*, aggiunta e rimozione di utenti dal sistema e impostazione delle autorizzazioni appropriate per gli utenti;

- *Distribuzione e manutenzione del software*, installazione di software applicativo e middleware e configurazione di questi sistemi in modo da evitare le vulnerabilità;

- *Monitoraggio, rilevamento e ripristino degli attacchi*, monitoraggio del sistema per accessi non autorizzati, progettazione di strategie per resistere agli attacchi e sviluppo di strategie di backup e ripristino;

La protezione delle operazioni è principalmente un problema umano e sociale, perché bisogna garantire che le persone che usano il sistema non si comportino in modo da compromettere il sistema (condividere la password, lasciare un account collegato, ecc). Spesso gli utenti si comportano in modo poco sicuro per svolgere il loro lavoro con più efficienza, c'è dunque un trade-off tra la protezione del sistema e efficacia di questo. Il termine protezione cibernetica, oggi comunemente usato quando si parla di protezione dei sistemi software, riguarda tutti gli aspetti della protezione dei cittadini, delle aziende e delle infrastrutture contro le minacce che possono nascere utilizzando i computer e Internet.

Protezione e affidabilità

La **protezione di un sistema** è una proprietà del sistema che riflette la capacità del sistema di proteggersi da attacchi esterni accidentali o intenzionali. La protezione è essenziale poiché la maggior parte dei sistemi è collegata in rete in modo che sia possibile l'accesso esterno al sistema tramite Internet. La protezione è un prerequisito essenziale per disponibilità, affidabilità e sicurezza. Se un sistema è collegato alla rete ed è insicuro, le dichiarazioni sulla sua affidabilità e sicurezza sono inaffidabili. Queste istruzioni dipendono dal sistema in esecuzione e dal sistema sviluppato che sono gli stessi. Tuttavia, l'intrusione può modificare il sistema in esecuzione e/o i suoi dati. Pertanto, la garanzia di affidabilità e sicurezza non è più valida. Dunque un sistema molto sicuro è un sistema non collegato alla rete, ma questa fornisce vantaggi enormi, bisogna capire se ne vale la pena .

Le vulnerabilità di un sistema possono nascere per problemi della definizione dei requisiti, nella progettazione o nell'implementazione del software o derivare da errori umani, sociali o aziendali. Esistono quattro tipi di minacce alle protezione:

Termine	Definizione
Risorsa	Qualcosa che merita di essere protetta. La risorsa può essere lo stesso sistema software o i dati utilizzati dal sistema.
Attacco	Un hacker sfrutta una vulnerabilità del sistema con lo scopo di provocare qualche danno alle risorse del sistema. Gli attacchi possono essere portati dall'esterno del sistema (attacchi esterni) o da utenti autorizzati del sistema (attacchi interni).
Controllo	Una misura protettiva che riduce la vulnerabilità del sistema. La crittografia può essere un esempio di controllo che riduce le vulnerabilità di un sistema che ha un controllo degli accessi debole.
Esposizione	Possibile perdita o danno a un sistema di calcolo. Il termine può riferirsi alla perdita o al danneggiamento dei dati oppure al tempo e alle energie spese per ripristinare le funzionalità del sistema dopo una violazione della protezione.
Minaccia	Circostanza che ha il potenziale di causare perdite o danni. Il termine identifica una vulnerabilità del sistema che è sottoposto a un attacco.
Vulnerabilità	Un punto debole in un sistema informatico che può essere sfruttato per arrecare perdite o danni al sistema.

- *Minacce di intercettazione*, consentono a un hacker di accedere a una risorsa. Una possibile minaccia per il sistema Mentcare potrebbe essere una situazione in cui un hacker ottiene l'accesso ai record di un singolo paziente.
- *Minacce di interruzione,* che consentono a un hacker di rendere non disponibile una parte del sistema. Una possibile minaccia potrebbe essere un attacco Denial of Service (DoS) su un server di database di sistema in modo che le connessioni al database diventino impossibili.
- *Minacce di modifica*, consentono a un hacker di manomettere una risorsa di sistema. Nel sistema Mentcare, una minaccia di modifica sarebbe quando un aggressore altera o distrugge una cartella clinica del paziente.
- *Minacce di fabbricazione*, consentono a un hacker di inserire informazioni false in un sistema. In un sistema bancario potrebbero essere aggiunte transazioni false al sistema che trasferisce denaro sul conto bancario dell'autore del reato.

I controlli che potremmo mettere in atto per migliorare la protezione del sistema si basano su tre concetti fondamentali: *avoidance, detection* e *recovery*.

- *Evitare le vulnerabilità* (avoidance), il sistema è progettato in modo che non si verifichino vulnerabilità. Ad esempio, se non esiste una connessione di rete esterna, l'attacco esterno è impossibile;
- *Identificazione e neutralizzazione degli attacchi* (detection), il sistema è progettato in modo che gli attacchi alle vulnerabilità vengano rilevati e neutralizzati prima che si traducano in un'esposizione. Ad esempio, i programmi antivirus trovano e rimuovono i virus prima che infettino un sistema;
- *Limitazione dell'esposizione e ripristino del sistema* (recovery), il sistema è progettato in modo da ridurre al minimo le conseguenze negative di un attacco riuscito. Ad esempio, una policy di backup consente il ripristino delle informazioni danneggiate;

La protezione è strettamente correlata con gli attributi della fidatezza: *affidabilità, disponibilità, sicurezza* e *resilienza*.

- *Protezione e affidabilità*, se un sistema viene attaccato e il sistema o i suoi dati vengono danneggiati in conseguenza di tale attacco, ciò potrebbe causare errori di sistema che compromettono l'affidabilità del sistema. Gli errori durante lo sviluppo di un sistema possono provocare delle falle nei meccanismi di protezione (se un sistema non rifiuta un input imprevisto, un hacker potrebbe usare questa debolezza per guadagnare l'accesso al sistema);
- *Protezione e disponibilità,* un attacco comune a un sistema basato sul Web è un attacco Denial of Service, in cui un server Web viene invaso da richieste di servizio da una serie di origini diverse. Lo scopo di questo attacco è rendere il sistema non disponibile (una variante è la minaccia di questo attacco con richiesta di riscatto a un sito economicamente redditizio);
- *Protezione e sicurezza,* un attacco che danneggia il sistema o i suoi dati significa che le ipotesi sulla sicurezza potrebbero non essere valide. I controlli di sicurezza si basano sull'analisi del codice sorgente del software

critico per la sicurezza e presumono che il codice in esecuzione sia una traduzione completamente accurata di quel codice sorgente. In caso contrario, possono essere indotti guasti relativi alla sicurezza e il caso di sicurezza realizzato per il software non è valido. Analogamente alla sicurezza non possiamo dare un valore numerico alla protezione di un sistema, non potremo mai dimostrare che un sistema è sicuro o protetto ma solo che non lo è;

- *Protezione e resilienza*, la resilienza è una caratteristica del sistema che riflette la sua capacità di resistere e riprendersi da eventi dannosi. L'evento dannoso più probabile sui sistemi software in rete è un attacco informatico di qualche tipo, quindi la maggior parte del lavoro ora svolto sulla resilienza è finalizzato a scoraggiare, rilevare e recuperare da tali attacchi;

Protezione e organizzazione

Realizzare sistemi protetti è un'operazione costosa e incerta, è impossibile prevedere i costi di un fallimento della protezione, per questo è importante prendere decisioni sulla protezione in modo conveniente. Le organizzazioni utilizzano un approccio basato sul rischio per supportare il processo decisionale sulla sicurezza e dovrebbero disporre di una policy di sicurezza definita basata sull'analisi dei rischi per la sicurezza. L'analisi dei rischi per la sicurezza è un problema economico piuttosto che tecnico.

Le politiche di protezione dovrebbero definire strategie generali di accesso alle informazioni che dovrebbero essere applicate a tutta l'organizzazione. Lo scopo delle politiche di protezione è informare tutti in un'organizzazione sulla protezione, quindi questi non dovrebbero essere documenti tecnici lunghi e dettagliati. Dal punto di vista dell'ingegneria della protezione, la politica di protezione definisce, in termini generali, gli obiettivi di protezione dell'organizzazione. Il processo di ingegneria della protezione riguarda l'implementazione di questi obiettivi. Affinché la gestione dei rischi possa essere efficace, le società dovrebbero adottare politiche di protezione basate sui seguenti criteri:

- *Le risorse che devono essere protette*, non è conveniente applicare rigorose procedure di protezione a tutte le risorse dell'organizzazione. Molte risorse non sono riservate e possono essere rese disponibili gratuitamente;

- *Il livello di protezione richiesto per diversi tipi di risorse (asset)*, per le informazioni personali sensibili è richiesto un elevato livello di protezione; per altre informazioni, le conseguenze della perdita possono essere minori, quindi un livello di protezione inferiore è adeguato;

- *Le responsabilità dei singoli utenti*, manager e società, la politica di protezione dovrebbe stabilire cosa ci si aspetta dagli utenti, ad es. password complesse, disconnessione dai computer, protezione dell'ufficio, ecc;

- *Procedure e tecnologie di protezione esistenti che dovrebbero essere mantenute*, per motivi di praticità e costo, può essere essenziale continuare a utilizzare gli approcci esistenti alla protezione anche laddove questi presentano limitazioni note;

Valutazione e gestione dei rischi

La valutazione e la gestione dei rischi si occupa di valutare le possibili perdite che potrebbero derivare da attacchi al sistema e di bilanciare queste perdite con i costi delle procedure di sicurezza che possono ridurre tali perdite. La gestione del rischio dovrebbe essere guidata da una politica di sicurezza dell'organizzazione; essa comporta:

- *Valutazione preliminare del rischio,* lo scopo di questa valutazione è identificare i rischi generici applicabili al sistema e decidere se è possibile ottenere un livello di sicurezza adeguato a un costo ragionevole. La valutazione del rischio dovrebbe concentrarsi sull'identificazione e l'analisi dei rischi di alto livello per il sistema. I risultati del processo di valutazione del rischio vengono utilizzati per identificare i requisiti di protezione;

- *Valutazione del rischio del ciclo di vita,* questa valutazione del rischio avviene durante il ciclo di vita di sviluppo del sistema e si basa sulle decisioni tecniche di progettazione e implementazione del sistema. I risultati della valutazione possono portare a modifiche ai requisiti diprotezione e all'aggiunta di nuovi requisiti. Le

vulnerabilità note e potenziali vengono identificate e questa conoscenza viene utilizzata per scegliere le funzionalità del sistema e stabilire come il sistema deve essere implementato, testato e distribuito;

- *Valutazione del rischio operativo,* Questo processo di valutazione del rischio si concentra sull'uso del sistema e sui possibili rischi che possono derivare dal comportamento umano. La valutazione del rischio operativo dovrebbe continuare dopo che un sistema è stato installato per tener conto di come il sistema viene utilizzato. Cambiamenti organizzativi possono significare che il sistema viene utilizzato in modi diversi da quelli originariamente pianificati. Questi cambiamenti portano a nuovi requisiti di protezione che devono essere implementati con l'evolversi del sistema;

Requisiti di protezione

La specifica di protezione ha qualcosa in comune con la specifica dei requisiti di sicurezza: in entrambi i casi, la preoccupazione è evitare che accada qualcosa di brutto. Analogamente a quelli di sicurezza, anche i **requisiti di protezione** sono di tipo "non deve", che definiscono un comportamento inaccettabile del sistema, anziché una funzionalità richiesta.

La protezione , però, è un problema più difficile da affrontare per diversi motivi:

- I problemi di sicurezza sono accidentali: il software non funziona in un ambiente ostile. Nella protezione, è necessario presumere che gli aggressori siano a conoscenza dei punti deboli del sistema;
- Quando si verificano problemi di sicurezza, è possibile cercare la causa principale o la debolezza che ha portato al guasto. Quando il fallimento risulta da un attacco intenzionale, l'aggressore può nascondere la causa del fallimento;
- L'arresto di un sistema può evitare un guasto relativo alla sicurezza. L'obbiettivo di un attacco potrebbe essere l'arresto;
- Gli eventi relativi alla sicurezza non sono generati da un avversario intelligente. Un aggressore può sondare le difese nel tempo per scoprire i punti deboli;

Dunque i requisiti di protezione devono essere più estesi dei requisiti di sicurezza. La sicurezza richiede un insieme di requisiti funzionali che assicurano la protezione contro eventi ed errori che potrebbero provocare fallimenti correlati alla sicurezza. I requisiti di protezione, invece, riguardano vari tipi di minacce che potrebbe subire un sistema.

Firesmith ha identificato 10 tipi di requisiti di protezione che possono essere inclusi nella specifica di un sistema:

1. *Requisiti di identificazione*, specificano se un sistema deve identificare i suoi utenti prima di interagire con loro;
2. *Requisiti di autenticazione*, specificano come identificare gli utenti;
3. *Requisiti di autorizzazione,* specificano i privilegi e le autorizzazioni di accesso degli utenti identificati;
4. *Requisiti di immunità,* specificano come un sistema dovrebbe proteggersi da virus, worm e minacce simili;
5. *Requisiti di integrità,* specificano come evitare di danneggiare i dati;
6. *Requisiti di rilevamento delle intrusioni,* specificano quali meccanismi utilizzare per scoprire gl attacchi portati al sistema;
7. *Requisiti di non ripudio,* specificano che una parte interessata in una transazione non può negare il proprio coinvolgimento;
8. *Requisiti di riservatezza,* specificano come deve essere mantenuta la riservatezza delle informazioni;

9. *Requisiti di controllo della protezione,* specificano come può essere controllato l'utilizzo di un sistema;

10. *Requisiti di protezione per la manutenzione del sistema,* specificano come un'applicazione può impedire che siano apportate la modifiche autorizzate da un accidentale annullamento dei meccanismi di protezione;

Ovviamente non tutti i sistemi richiedono tutti questi requisiti, dipendono dal tipo di sistema, di utenti e dalle condizioni d'uso.

Lo scopo della valutazione preliminare dei rischi è identificare i rischi generici della protezione per un sistema e i suoi dati, è un input importante per il processo di ingegneria dei requisiti della protezione. Questi requisiti possono essere proposti per supportare le strategie generali della gestione dei rischi: *evitare, identificare* e *mitigare* i rischi.

- I *requisiti di prevenzione* dei rischi stabiliscono i rischi che dovrebbero essere evitati progettando il sistema in modo che questi rischi semplicemente non possano sorgere;
- I *requisiti di rilevamento* del rischio definiscono meccanismi che identificano il rischio se si presenta e neutralizzano il rischio prima che si verifichino perdite;
- I *requisiti di mitigazione* del rischio stabiliscono come il sistema dovrebbe essere progettato in modo che possa recuperare e ripristinare le risorse del sistema dopo che si è verificata una perdita;

Il processo di valutazione preliminare dei rischi per definire i requisiti di protezione è composto dalle seguenti fasi:

1. *Identificazione delle risorse*, si identificano le risorse (o servizi) chiave del sistema che devono essere protette;
2. *Valutazione del valore patrimoniale*, si stima il valore delle risorse identificate;
3. *Valutazione dell'esposizione*, si valuta le potenziali perdite associate a ciascuna risorsa;

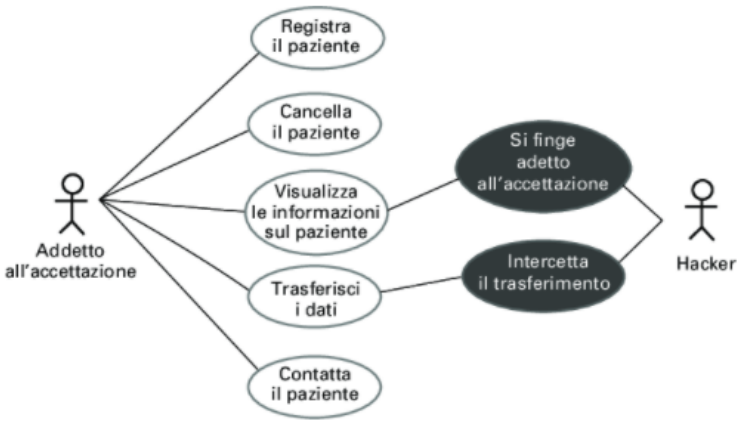
4. *Identificazione delle minacce*, si identificano le minacce più probabili per le risorse di sistema;
5. *Valutazione degli attacchi*, si decompone ciascuna minaccia nei vari possibili attacchi che potrebbe subire il sistema e nei modi in cui possono verificarsi;
6. *Identificazione dei controlli*, si definiscono i controlli che possono essere messi in atto per proteggere le risorse del sistema;
7. *Valutazione di fattibilità*, si valuta la fattibilità tecnica e il costo dei controlli proposti;
8. *Definizione dei requisiti di protezione*, una volta svolti i passi precedenti è possibile definire i requisiti di protezione del sistema. Questi possono essere requisiti di sistema dell'infrastruttura o dell'applicazione;

Casi d'uso improprio

I **casi d'uso improprio** sono scenari che rappresentano interazioni dannose con un sistema, possono essere usati per discutere ed identificare possibili minacce e, quindi, determinare i requisiti di protezione in un sistema. I casi d'uso improprio sono associati alle istanze dei casi d'uso e rappresentano minacce o attacchi correlati a questi casi d'uso, ma devono avere anche una descrizione più dettagliata e completa.

Come i casi d'uso anche questi possono essere descritti in vari modi, è utile scriverli come supplemento alla descrizione originale dei casi d'uso.

È meglio avere un formato flessibile per i casi d'uso improprio, in quanto ci sono vari tipi di attacchi che possono essere descritti in modi differenti:



- *Minacce di intercettazione*, l'hacker ottiene l'accesso a una risorsa;
- *Minacce di interruzione*, l'hacker rende indisponibile parte di un sistema;
- *Minacce di modifica*, una risorsa di sistema se manomessa;
- *Minacce di fabbricazione*, informazioni false vengono aggiunte a un sistema;

Progettazione di un sistema protetto

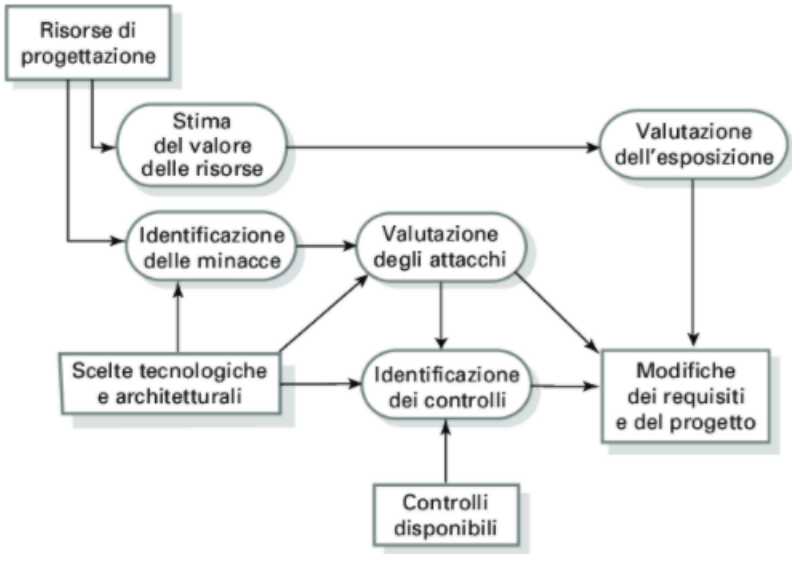
È molto difficile proteggere un sistema dopo che è stato implementato, pertanto occorre tenere in considerazione i problemi della protezione durante lo sviluppo di un sistema e fare scelte appropriate di progettazione in modo da migliorare la protezione del sistema. Due argomenti importanti per la progettazione di sistemi protetti sono:

- *Progettazione architetturale*, come influiscono le scelte di progettazione sulla protezione di un sistema?
- *Buone pratiche di progettazione*, quali sono le buone pratiche da adottare durante la progettazione di un sistema protetto?

Ogni applicazione è diversa da un'altra, la progettazione della protezione deve tenere conto dello scopo, della criticità e dell'ambiente operativo di ciascuna applicazione. La progettazione di un sistema protetto richiede inevitabilmente dei compromessi: aggiungere funzionalità per aumentare la protezione di un sistema influisce negativamente sulle prestazioni complessive del sistema stesso. Dobbiamo dunque trovare un compromesso per massimizzare le *prestazioni* e *l'usabilità*.

Valutazione dei rischi

La valutazione dei rischi di protezione durante l'ingegneria dei requisiti identifica una serie di requisiti di protezione di alto livello per il sistema. Durante la progettazione e l'implementazione del sistema, le scelte architetturali e tecnologiche possono influire sulla protezione di un sistema. Queste scelte generano nuovi requisiti che potrebbero richiedere la modifica dei requisiti esistenti.

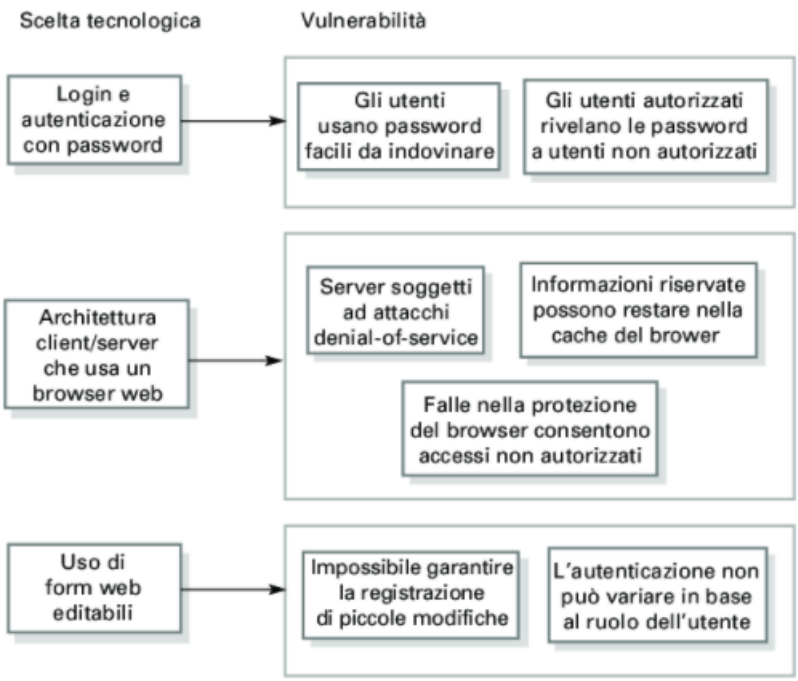


La valutazione dei rischi e la progettazione del sistema sono processi intrecciati, ad ogni evoluzione del sistema occorre operare una nuova valutazione dei rischi, questo processo termina solo una volta che il sistema è ultimato e i rischi restanti sono accettabili. Quando si valutano i rischi durante la progettazione e l'implementazione, si hanno maggiori informazioni su cosa deve essere protetto e sulle vulnerabilità del sistema; alcune di queste vulnerabilità riguardano le scelte di progettazione. La differenza principale tra l'analisi preliminare dei rischi e la valutazione dei rischi di progettazione è

che, nella fase di progettazione, si hanno informazioni sulla rappresentazione e distribuzione dei dati e sull'organizzazione del database per le risorse di alto livello che devono essere protette; inoltre, si conoscono le scelte importanti di progettazione.

Quando si sviluppa un'applicazione usando un sistema esistente, occorre accettare le scelte di progettazione che sono state fatte dagli sviluppatori di quel sistema, come ad esempio:

- gli utenti del sistema sono autenticati utilizzando una combinazione nome / password;
- l'architettura del sistema è client-server con i client che accedono al sistema tramite un browser web standard;
- le informazioni vengono presentate come un modulo web modificabile;



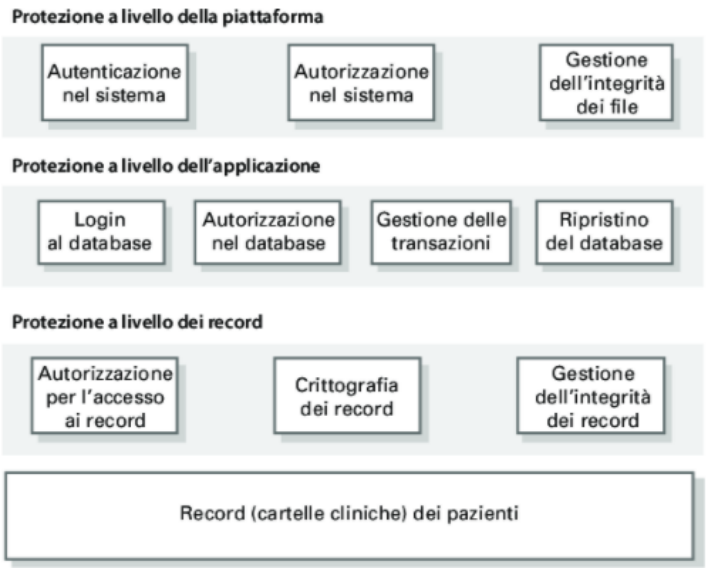
Una volta identificate le vulnerabilità, occorre studiare i rimedi per ridurre i rischi associati. Questo spesso richiede nuovi requisiti di protezione del sistema o la modifica delle modalità di utilizzo del sistema. Rifacendoci al caso soprastante, esempi di requisiti sono:

- dovrà essere messo a disposizione un programma che controlla la password giornalmente, gli utenti con password deboli dovranno essere segnalati agli amministratori del sistema;
- sarà consentito l'accesso al sistema solamente ai client che sono stati approvati e registrati dagli amministratori del sistema;
- su ogni computer client potrà essere installato un solo browser approvato;

Progettazione dell'architettura

Quando si progetta l'architettura di un sistema che garantisce la protezione, occorre considerare due punti fondamentali:

1. *Protezione,* in che modo si deve organizzare il sistema per far si che le risorse critiche possano essere protette da attacchi esterni?
2. *Distribuzione*, come devono essere distribuite le risorse all'interno del sistema per ridurre al minimo le conseguenze di un attacco che ha successo?



Questi sono però potenzialmente in conflitto: se le risorse vengono distribuite, è più costoso proteggerle ma, se le risorse sono protette, i requisiti di usabilità e prestazioni potrebbero essere compromessi. Per assicurare la protezione, si può progettare il sistema con un architettura a strati con le risorse critiche protette al livello più basso del sistema (il sistema Mentcare è così organizzato).

Per accedere ai record e modificarli, un hacker deve penetrare tre strati di **protezione** del sistema:

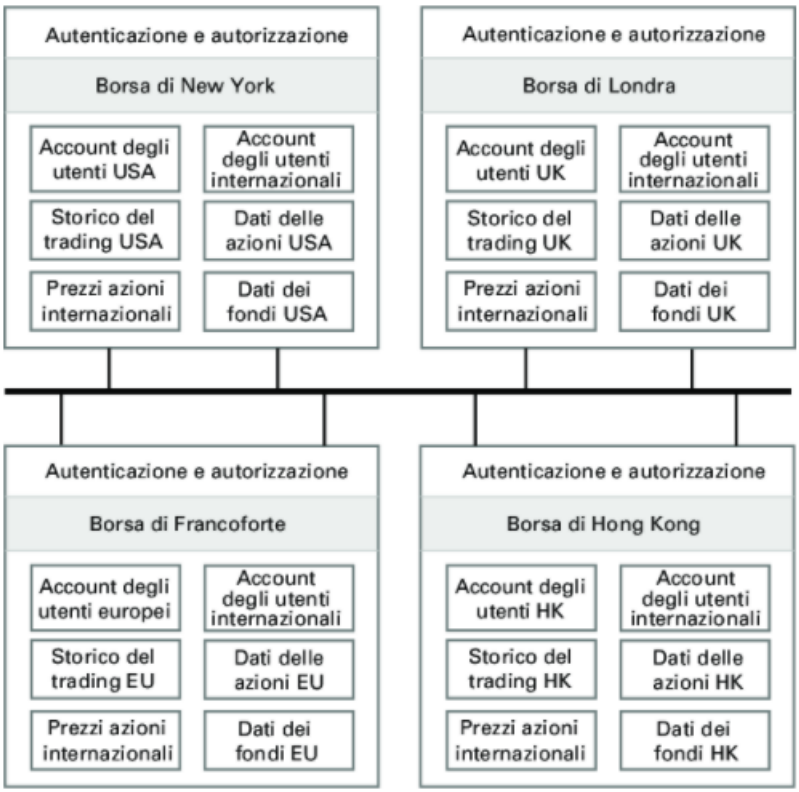
1. *Protezione a livello della piattaforma*, controlli di primo livello sulla piattaforma su cui viene eseguito un sistema (l'utente deve collegarsi con un particolare computer);

2. *Protezione a livello di applicazione*, meccanismi di protezione specifici integrati nell'applicazione stessa (protezione tramite password aggiuntiva);

3. *Protezione a livello di record*, protezione che viene invocata quando viene richiesto l'accesso a informazioni specifiche (un sistema di crittografia);

Il numero di strati di protezione necessari dipende dalla criticità dei dati.

Distribuire risorse significa che gli attacchi a un sistema non portano necessariamente alla completa perdita del servizio di sistema. Ogni piattaforma ha funzionalità di protezione separate e può essere diversa dalle altre piattaforme in modo che non condividano una vulnerabilità comune. La distribuzione è particolarmente importante se il rischio di attacchi Denial of Service è elevato.



Linee guida per la progettazione

Non ci sono metodi facili per garantire la protezione di un sistema, tipi differenti di sistemi richiedono misure di protezione differenti per raggiungere un livello di protezione accettabile per il proprietario del sistema. È possibile, tuttavia, indicare delle **linee guida** che includono alcune buone pratiche di progettazioni per l'ingegneria dei sistemi protetti. Queste linee guida hanno due scopi principali:

1. Aumentare la consapevolezza dei problemi di protezione in un team di ingegneri del software. La protezione è considerata quando vengono prese le decisioni di progettazione;

2. Possono essere utilizzati come base per una checklist di revisione che viene applicata durante il processo di convalida del sistema.

Le linee guida di progettazione, a volte, sono principi molto generali ("proteggere la connessione più debole del sistema"), vi sono però 10 direttive di progettazione più pratiche, estrapolate da diverse fonti:

1. *Basare le decisioni su una politica di sicurezza esplicita*, definire una politica di sicurezza per l'organizzazione che stabilisca i requisiti di sicurezza fondamentali che dovrebbero applicarsi a tutti i sistemi organizzativi;

2. *Evitare un singolo punto di errore*, assicurati che un errore di sicurezza possa verificarsi solo quando c'è più di un errore nelle procedure di sicurezza. Ad esempio, disporre di password e autenticazione basata su domande;

3. *Fallire in modo sicuro*, quando i sistemi si guastano, per qualsiasi motivo, assicurarsi che le informazioni sensibili non siano accessibili da utenti non autorizzati anche se le normali procedure di sicurezza non sono disponibili;

4. *Bilanciare sicurezza e usabilità*, cerca di evitare procedure di sicurezza che rendono il sistema difficile da usare. A volte è necessario accettare una sicurezza più debole per rendere il sistema più utilizzabile;

5. *Registrare le azioni dell'utente*, mantieni un registro delle azioni dell'utente che può essere analizzato per scoprire chi ha fatto cosa. Se gli utenti conoscono un registro di questo tipo, è meno probabile che si comportino in modo irresponsabile;

6. *Usare la ridondanza e la diversità per ridurre il rischio*, conserva più copie dei dati e utilizza un'infrastruttura diversificata in modo che una vulnerabilità dell'infrastruttura non possa essere l'unico punto di errore.

7. *Specificare il formato di tutti gli input di sistema,* se i formati di input sono noti, è possibile verificare che tutti gli input rientrino nell'intervallo in modo che input imprevisti non causino problemi;

8. *Compartimentalizzare le tue risorse*, organizzare il sistema in modo che le risorse siano in aree separate e gli utenti abbiano accesso solo alle informazioni di cui hanno bisogno anziché a tutte le informazioni di sistema;

9. *Progettare per la distribuzione*, progettare il sistema per evitare problemi di distribuzione;

10. *Progettare per la recuperabilità*, progettare il sistema per semplificare la recuperabilità dopo un attacco riuscito;

Programmare

Oltre a garantire la protezione in fase di progettazione, è anche importante considerare la protezione quando si programma un sistema software. Le vulnerabilità sono spesso specifiche dei linguaggi (in C non c'è alcun controllo automatico sui limiti degli array), inoltre le vulnerabilità della protezione sono strettamente correlate all'affidabilità dei programmi. Come per la progettazione, anche in questo caso abbiamo delle linee guida per una programmazione fidata:

1. *Limitare la visibilità delle informazioni in un programma;*
2. *Verificare la validità di tutti gli input;*
3. *Fornire un gestore per tutte le eccezioni;*
4. *Ridurre al minimo l'uso di costrutti inclini a errori;*
5. *Fornire funzionalità di riavvio;*
6. *Verificare i limiti dell'array;*
7. *Includere timeout quando si chiamano componenti esterni;*
8. *Assegnare un nome a tutte le costanti che rappresentano i valori del mondo reale;*

Test e garanzie di protezione

Testare la misura in cui il sistema può proteggersi da attacchi esterni sta diventando sempre più importante, tuttavia ci sono dei problemi con i test di sicurezza:

- I requisiti di protezione, come quelli di sicurezza, sono requisiti "non devono", ovvero specificano cosa non dovrebbe accadere. Di solito non è possibile definire i requisiti di protezione come semplici vincoli che possono essere verificati dal sistema;
- Le persone che attaccano un sistema sono intelligenti e cercano vulnerabilità. Possono sperimentare per scoprire punti deboli e scappatoie nel sistema che i progettisti non sono riusciti a rendere protetti;

Possiamo usare una combinazione di test, analisi supportate da strumenti e verifiche formali per controllare e analizzare la protezione di un sistema di applicazioni:

1. *Test basati sull'esperienza*, il sistema viene riesaminato e analizzato rispetto ai tipi di attacchi noti al team di convalida. Si può definire una lista di controllo per i problemi noti della protezione;

2. *Test di violazione*, viene istituito un team il cui obiettivo è violare la protezione del sistema simulando attacchi al sistema;

3. *Analisi supportata da strumenti*, vari strumenti di protezione come il controllo delle password vengono utilizzati per analizzare il sistema in funzione;

4. *Verifica formale*, si formulano delle argomentazioni matematiche formali che dimostrano che il sistema soddisfa i suoi requisiti formali di protezione;

19 - GESTIONE DELLA PROGETTAZIONE

La gestione dei progetti software è una parte essenziale, si occupa dunque di garantire che il software sia consegnato nei tempi e nei tempi previsti e in conformità con i requisiti delle organizzazioni che sviluppano e acquistano il software. La gestione del progetto è necessaria perché lo sviluppo del software è sempre soggetto ai vincoli di budget e di pianificazione stabiliti dall'organizzazione che sviluppa il software. Una buona gestione non può garantire il successo di un progetto, di contro una cattiva gestione ne determina il fallimento.

I criteri per il successo variano da progetto a progetto , tuttavia per la maggior parte dei progetti, gli obiettivi più importanti sono:

- *Consegnare il software entro i termini concordati;*
- *Mantenere i costi entro il budget;*
- *Consegnare un software che soddisfa le richieste del cliente;*
- *Mantenere una squadra di sviluppo affiatata e collaudata;*

L'ingegneria del software è diversa dalle altre ingegnerie per varie ragioni che rendono la gestione del software particolarmente impegnativa. Alcune differenze sono:

1. *Il prodotto è intangibile*, il software non può essere visto o toccato. I project manager del software non possono vedere i progressi semplicemente guardando il manufatto che viene costruito;

2. *Molti progetti software sono progetti "unici"*, i progetti software di grandi dimensioni sono generalmente diversi in qualche modo dai progetti precedenti. Anche i manager che hanno molta esperienza precedente potrebbero avere difficoltà ad anticipare i problemi;

3. *I processi software sono variabili e specifici dell'organizzazione*, non siamo ancora in grado di prevedere in modo affidabile quando un particolare processo software potrebbe portare a problemi di sviluppo;

I sistemi software sono spesso nuovi, molto complessi e tecnicamente innovativi, è per questo impossibile fornire una descrizione dei compiti standard per un manager di progetto . I compiti variano notevolmente in funzione dell'azienda e del tipo di software che si sta sviluppando. Alcuni dei più importanti fattori che influiscono sulla gestione dei progetti software sono:

1. *Dimensione dell'azienda*, piccole aziende operano con gestione e comunicazioni informali, nelle grandi aziende devono essere rispettate le gerarchie gestionali, report e budget formali e procedure di approvazione;

2. *Clienti di software*, se il cliente è interno le comunicazioni possono essere informali, se il cliente è esterno bisogna raggiungere un accordo sui canali di comunicazione formali;

3. *Dimensioni del software*, i piccoli sistemi software possono essere sviluppati da piccoli team, per grandi sistemi software c'è bisogno di un team che può richiedere più team di sviluppo (anche di diversi paesi o agenzie). Il manager si occupa di coordinare le squadre;

4. *Tipo di software,* se è un software a sicurezza critica è necessario che tutte le decisioni sulla gestione dei progetti siano registrate e giustificate in quanto possono influire sulla sicurezza del sistema;

5. *Cultura aziendale,* alcune aziende preferiscono incentivare il lavoro individuale, altre il lavoro di gruppo. Alcune aziende sono più propense al rischio mentre altre hanno una cultura avversa ai rischi;

6. *Processi di sviluppo software*, i processi agili operano con una gestione "leggera", processi più formali richiedono un monitoraggio;

Questi fattori significano che i project manager in diverse organizzazioni possono lavorare in modi molto diversi. Tuttavia vi sono attività comuni a tutte le aziende:

1. *Pianificazione dei progetti*, i project manager sono responsabili della pianificazione della stima e della tempistica dello sviluppo dei progetti, inoltre devono assegnare le persone alle attività;

2. *Gestione dei rischi*, i project manager valutano i rischi che possono influenzare un progetto, monitorano questi rischi e agiscono in caso di problemi;

3. *Gestione del personale*, i project manager devono scegliere le persone per il proprio team e stabilire modalità di lavoro che portino a prestazioni di squadra efficaci;

4. *Reporting*, i project manager sono solitamente responsabili di riferire sullo stato di avanzamento di un progetto ai clienti e ai manager dell'azienda che sviluppa il software;

5. *Scrittura delle proposte*, la prima fase in un progetto software può comportare la scrittura di una proposta per vincere un contratto per eseguire un elemento di lavoro. La proposta descrive gli obiettivi del progetto e come sarà realizzato;

Gestione dei rischi

I rischi possono minacciare un progetto, il software in via di sviluppo o l'organizzazione. La **gestione dei rischi** si occupa di identificare i rischi e di elaborare piani per minimizzare il loro effetto su un progetto. La gestione del rischio del software è importante a causa delle incertezze inerenti allo sviluppo del software. Queste incertezze derivano da requisiti vagamente definiti, modifiche dei requisiti dovute a cambiamenti nelle esigenze dei clienti, difficoltà nella stima del tempo e delle risorse necessarie per lo sviluppo del software e differenze nelle competenze individuali. Bisogna anticipare i rischi, comprendere l'impatto di questi rischi sul progetto, sul prodotto e sul business e adottare misure per evitare questi rischi.

I rischi possono essere classificati in due modi: in base alla loro natura (*tecnica, organizzativa, ecc.*) oppure in base all'oggetto dei loro effetti:

- *Rischi per il progetto*, influiscono sulle risorse e sulla tempistica del progetto (un esempio è la perdita di un progettista esperto);

- *Rischi per il prodotto*, influiscono sulla qualità o sulle prestazioni del software che si sta sviluppando (un esempio è un componente che non funziona come previsto);

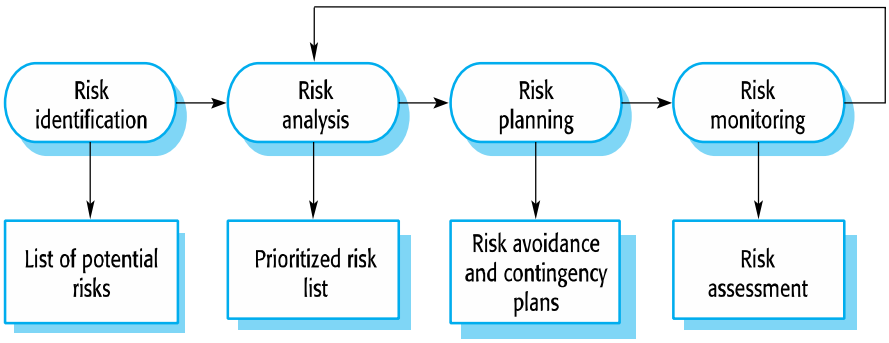
- *Rischi per l'azienda*, influiscono sull'azienda che sta sviluppando o acquistando il software (un esempio è un'azienda concorrente che sviluppa un nuovo prodotto);

Ovviamente questi tipi di rischi possono sovrapporsi. Uno schema generico della gestione dei rischi rappresenta varie fasi:

1. *Identificazione dei rischi*, identificare i rischi di progetto, prodotto e business;

2. *Analisi dei rischi, v*alutare la probabilità e le conseguenze di questi rischi;

3. *Pianificazione dei rischi,* elaborare piani per evitare o ridurre al minimo gli effetti



del rischio;

4. *Monitoraggio dei rischi*, monitorare i rischi durante tutto il progetto;

Per grandi progetti si dovrebbero documentare i risultati del processo di gestione dei rischi in un apposito piano di gestione. Il processo di gestione dei rischi è un processo iterativo che continua per tutta lo sviluppo del progetto. Lo sviluppo agile riduce alcuni rischi come quelli derivanti dalla modifica dei requisiti, tuttavia presenta degli svantaggi a causa della sua dipendenza dal personale, della mancanza di documentazione e comunicazioni formali.

Identificazione dei rischi

Potrebbe essere un processo di squadra o, in alternativa, il manager potrebbe fare affidamento alla sua esperienza sugli errori in precedenti progetti. Come punto di partenza si può utilizzare una lista che elenca i vari tipi di rischi:

1. *Rischi di stima*, derivano dalle stime delle risorse che dovrebbero servire al sistema che si sta costruendo;
2. *Rischi aziendali,* derivano dall'ambiente aziendale in cui sta avvenendo lo sviluppo software;
3. *Rischi riguardanti il personale,* associati alle persone del team;
4. *Rischi dei requisiti,* derivano dalla modifica dei requisiti del cliente e del processo di gestione dei requisiti;
5. *Rischi tecnologici,* derivano dalle tecnologie hardware o software utilizzate per sviluppare il sistema;
6. *Rischi degli strumenti,* derivano dal software di supporto e dagli strumenti software utilizzati per lo sviluppo;

Una volta terminato questo processo si dovrebbe avere una lista di potenziali rischi.

Analisi dei rischi

Bisogna considerare i singoli rischi individuati e valutarne probabilità e gravità. Non è possibile fare delle valutazioni numeriche precise, perciò associamo ogni rischio a una *banda di rischio*:

- *Probabilità*, può essere stimata in molto bassa, bassa, moderata, alta, o molto alta;
- *Effetti*, possono essere stimati come catastrofici, gravi, tollerabili o insignificanti;

Ovviamente la probabilità e la gravità di un rischio non restano costanti, variano assieme al progetto e possono cambiare quando si ottengono nuove informazioni. Una volta che i rischi sono stati classificati occorre valutare quali sono quelli più significativi (combinazione di gravità e frequenza).

Pianificazione dei rischi

Bisogna sviluppare per ogni rischio una strategia per gestirlo, bisogna considerare le azioni che potrebbero essere svolte per minimizzare gli effetti dannosi sul progetto. Occorre porsi delle domande e, in base alle risposte, si definiscono le strategie di gestione dei rischi. Queste strategie possono essere classificate in tre categorie:

1. *Strategie per evitare i rischi,* ridurre la probabilità che un rischio si concretizzi;
2. *Strategie di minimizzazione dei rischi,* ridurre l'impatto del rischio sul progetto o sul prodotto;
3. *Piani di emergenza,* prepararsi al peggio e avere soluzioni pronte a gestire situazioni molto gravi;

Vi è un analogia con le strategie adottate nei sistemi critici per garantire affidabilità, protezione e sicurezza.

Monitoraggio dei rischi

Occorre valutare regolarmente ogni rischio identificato per decidere se sta diventando meno o più probabile e se gli effetti del rischio sono cambiati. Ogni rischio chiave dovrebbe essere discusso durante le riunioni di avanzamento della gestione.

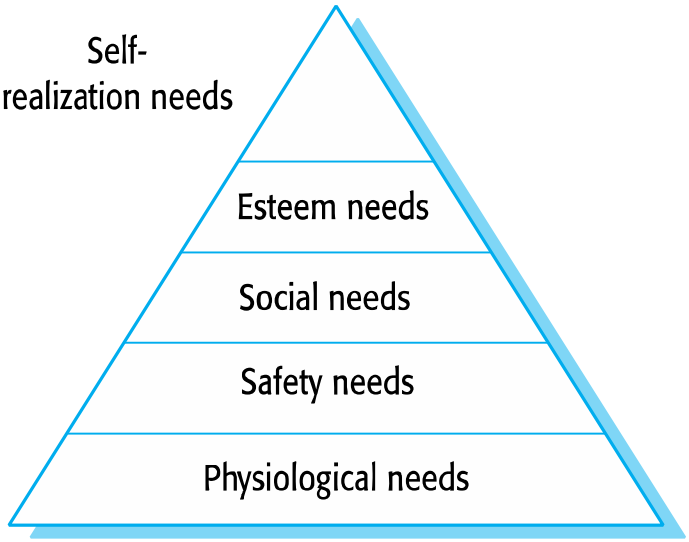
Gestione del personale

Le persone sono le risorse più importanti di una società, per questo motivo i compiti di un manager sono essenzialmente orientati alle persone. A meno che non ci sia una certa comprensione delle persone, la gestione non avrà successo. Una cattiva gestione delle persone è un importante contributo al fallimento del progetto. Vi sono quattro fattori critici che influiscono sulle relazioni tra un manager e il suo personale:

1. *Coerenza*, i membri del team dovrebbero essere tutti trattati in modo comparabile senza favoriti o discriminazioni;
2. *Rispetto,* i diversi membri del team hanno abilità diverse e queste differenze dovrebbero essere rispettate;
3. *Considerazione*, coinvolgere tutti i membri del team e assicurarsi che le opinioni delle persone siano prese in considerazione;
4. *Onestà/Obiettività,* I manager dovrebbero sempre essere onesti su ciò che sta andando bene e cosa sta andando male in un progetto;

Un ruolo importante di un manager è motivare le persone che lavorano a un progetto. Motivazione significa organizzare il lavoro e l'ambiente di lavoro per incoraggiare le persone a lavorare in modo efficace. Se le persone non sono motivate, non saranno interessate al lavoro che stanno facendo. Lavoreranno lentamente, avranno maggiori probabilità di commettere errori e non contribuiranno agli obiettivi più ampi del team o dell'organizzazione. Per Maslow le persone sono motivate quando sono soddisfatte le loro necessità, che possono essere classificate in una serie di livelli di una piramide:

- Alla base abbiamo i *bisogni primari* (es. Cibo, sonno, ecc.);
- Seguono i *bisogni personali* (es. Rispetto, autostima);
- Successivamente abbiamo i *bisogni sociali* (ad es. Essere accettati come parte di un gruppo);
- Al penutlimo abbiamo il *bisogno di stima* (sentirsi rispettati dagli altri);
- In cima alla piramide c'è il *bisogno della autorealizzazione* (riguarda lo sviluppo della propria personalità);



Nei gruppi di sviluppo software, le esigenze fisiologiche e di sicurezza di base non sono un problema. Per soddisfare le necessità sociali bisogna: fornire strutture comuni e consentire comunicazioni informali, (ad es. tramite social network). Per soddisfare le necessità di stima occorre: riconoscere i risultati dei membri del gruppo, assegnando anche delle ricompense adeguate. Infine per soddisfare le esigenze di autorealizzazione occorre responsabilizzare le persone nello svolgimento del lavoro, assegnando compiti difficili, ma non impossibili, offrendo loro l'opportunità di partecipare a corsi di addestramento e formazione per sviluppare le loro capacità.

Il modello delle motivazioni di Maslow non tiene conto in modo adeguato del fatto che le persone sentono di far parte di un'organizzazione, di un gruppo professionale o di una o più culture. Sentirsi membro di un gruppo coeso è altamente motivante per una persona. Un manager deve dunque motivare le persone come gruppo , non solo individualmente. Le difficoltà personali influiscono molto spesso sulle motivazioni, in quanto le persone non riescono a concentrarsi sul lavoro. Altro fattore che influenza la motivazione è il tipo di personalità, Bass e Dunteman hanno classificato i professionisti in tre categorie:

1. *Persone orientate ai compiti*, motivate dal lavoro che svolgono;
2. *Persone orientate alle interazioni*, motivate dalla presenza e dalle azioni dei colleghi;

3. *Persone orientate a se stesse*, motivate principalmente dal successo e dal riconoscimento personale, interessate allo sviluppo del software come mezzo per raggiungere i propri obiettivi;

Le motivazioni individuali sono costituite da elementi di ogni classe, ma un solo tipo di motivazione di solito è dominante. L'equilibrio può cambiare a seconda delle circostanze personali e degli eventi esterni.

Lavoro di squadra

La maggior parte dell'ingegneria del software è un'attività di gruppo, il programma di sviluppo per la maggior parte dei progetti software non banali è tale che non possono essere completati da una persona che lavora da sola. Un buon gruppo è coeso e ha uno spirito di squadra. Le persone coinvolte sono motivate dal successo del gruppo e dai propri obiettivi personali. L'interazione di gruppo è un fattore determinante per le prestazioni del gruppo, tuttavia la flessibilità nella composizione del gruppo è limitata (i manager devono fare del loro meglio con le persone disponibili).

In un gruppo coeso, i membri considerano il gruppo più importante di qualsiasi individuo in esso. I vantaggi di avere un team coeso sono:

- *Il gruppo può definire i suoi standard di qualità*, poiché definiti di comune accordo è più facile che siano rispettati;
- *I membri del team imparano gli uni dagli altri e si supportano a vicenda*, ogni membro;del gruppo impara lavorando insieme agli altri membri, quindi le inibizioni causate dall'ignoranza sono ridotte.
- *Le conoscenze sono condivise*, la continuità del lavoro può essere mantenuta anche se un membro del gruppo lo abbandona;
- *Il refactoring e il miglioramento continuo sono incoraggiati,* i membri del gruppo lavorano collettivamente per fornire risultati di alta qualità e risolvere i problemi, indipendentemente dalle persone che hanno originariamente creato il progetto o il programma;

Un project manager dovrebbe trattare i membri del gruppo come persone responsabili e degne di fiducia, lasciando loro un libero accesso alle informazioni. L'efficienza di un gruppo dipende dalla natura del progetto e dalla società che sta svolgendo il lavoro. Dato un ambiente di lavoro stabile, sono tre i fattori che maggiore influenza sulla squadra di sviluppo:

1. *Le persone nel gruppo,* si ha bisogno di un mix di persone in un gruppo di progetto poiché lo sviluppo del software coinvolge diverse attività come la negoziazione con i clienti, la programmazione, i test e la documentazione;
2. *L'organizzazione del gruppo, u*n gruppo dovrebbe essere organizzato in modo che gli individui possano contribuire al meglio delle loro capacità e le attività possano essere completate come previsto;
3. *Comunicazioni tecniche e gestionali, è* essenziale una buona comunicazione tra i membri del gruppo e tra il team di ingegneri del software e altri stakeholder del progetto;

Selezione dei membri del gruppo

Il compito di un manager o di un team leader è creare un gruppo coeso e organizzare il proprio gruppo in modo che possano lavorare insieme in modo efficace. Ciò comporta la creazione di un gruppo con il giusto equilibrio di abilità tecniche e personalità e l'organizzazione di quel gruppo in modo che i membri lavorino insieme in modo efficace. Potrebbe non essere possibile nominare le persone ideali per lavorare su un progetto:

- Il budget del progetto potrebbe non consentire l'utilizzo di personale altamente retribuito;
- Il personale con l'esperienza appropriata potrebbe non essere disponibile;

- Un'organizzazione potrebbe voler sviluppare le competenze dei dipendenti su un progetto software;

I manager devono lavorare entro questi vincoli, soprattutto quando c'è carenza di personale qualificato. Comporre un gruppo con membri che condividono le stesse motivazioni potrebbe essere problematico, un gruppo efficace è bilanciato con tutti i tipi. Potrebbe essere molto difficile da ottenere perche gli ingegneri del software sono spesso persone orientate ai compiti. Delle persone molto importanti in un gruppo sono quelle orientate all'interazione, questo perché possono individuare e disinnescare le tensioni che si presentano.

Organizzazione del gruppo

Il modo in cui un gruppo è organizzato influenza le decisioni che vengono prese da quel gruppo, i modi in cui le informazioni vengono scambiate e le interazioni tra il gruppo di sviluppo e gli stakeholder esterni del progetto. Alcune delle domande chiave che un manager dovrebbe porsi sono:

- Il project manager dovrebbe essere il capo tecnico del gruppo?
- Chi sarà coinvolto nel prendere decisioni tecniche critiche e come verranno prese?
- Come verranno gestite le interazioni con gli stakeholder esterni e l'alta dirigenza aziendale?
- Come possono i gruppi integrare persone che non si trovano nella stessa sede?
- Come è possibile condividere la conoscenza in tutto il gruppo?

Piccoli gruppi di ingegneria del software sono generalmente organizzati in modo informale senza una struttura rigida. Per progetti di grandi dimensioni, potrebbe esserci una struttura gerarchica in cui diversi gruppi sono responsabili di diversi sottoprogetti. Lo sviluppo agile si basa sempre su un gruppo informale sul principio che la struttura formale inibisce lo scambio di informazioni. Nei gruppi informali il gruppo agisce nel suo insieme e raggiunge un consenso sulle decisioni che interessano il sistema. Il capogruppo funge da interfaccia esterna del gruppo ma non assegna elementi di lavoro specifici. Piuttosto, il lavoro viene discusso dal gruppo nel suo insieme e i compiti vengono assegnati in base alle capacità e all'esperienza. Questo approccio ha successo per i gruppi in cui tutti i membri sono esperti e competenti.

Comunicazione tra i membri del gruppo

Una buona comunicazione è essenziale per un lavoro di gruppo efficace. Devono essere scambiate informazioni sullo stato del lavoro, sulle decisioni di progettazione e sulle modifiche alle decisioni precedenti. Una buona comunicazione rafforza anche la coesione del gruppo poiché promuove la comprensione. Sono diversi i fattori che influiscono sull'efficacia e sull'efficienza della comunicazione:

Dimensione del gruppo, più grande è il gruppo, più difficile è per le persone comunicare con gli altri membri del gruppo;

Struttura del gruppo, la comunicazione è migliore in gruppi strutturati in modo informale che in gruppi strutturati gerarchicamente;

Composizione del gruppo, la comunicazione è migliore quando ci sono diversi tipi di personalità in un gruppo e quando i gruppi sono misti piuttosto che single;

L'ambiente di lavoro fisico, una buona organizzazione del posto di lavoro può aiutare a incoraggiare la comunicazione;

I canali di comunicazione, bisogna sfruttare le tecnologie di interazione, ci sono molte forme di comunicazione (faccia a faccia, messaggi, e-mail, documenti formali, telefono, social network e wiki);

20 – PIANIFICAZIONE DELLA PROGETTAZIONE

La pianificazione del progetto prevede la suddivisione del lavoro in parti e l'assegnazione di queste ai membri del team di progetto, l'anticipazione dei problemi che potrebbero sorgere e la preparazione di soluzioni provvisorie a tali problemi. Il piano del progetto, che viene creato all'inizio di un progetto, viene utilizzato per comunicare come verrà svolto il lavoro al team di progetto e ai clienti e per aiutare a valutare i progressi del progetto. La pianificazione della progettazione si svolge in tre fasi nel ciclo di vita di un progetto:

1. *Proposta*, si fa un'offerta per un contratto per lo sviluppo o la fornitura di un sistema software;
2. *Avviamento*, si deve pianificare chi lavorerà al progetto, come il progetto sarà suddiviso in incrementi, come verranno allocate le risorse nella tua azienda, ecc;
3. *Revisioni*, si rivede periodicamente il progetto, aggiornando il piano di sviluppo alla luce dell'esperienza acquisita e delle informazioni derivanti dal monitoraggio dello stato di avanzamento dei lavori;

Potrebbe essere necessaria una pianificazione con solo requisiti software di massima. Lo scopo della pianificazione in questa fase è fornire informazioni che verranno utilizzate per stabilire un prezzo per il sistema ai clienti. Il prezzo del progetto implica la stima di quanto costerà il software per lo sviluppo, tenendo conto di fattori come i costi del personale, i costi dell'hardware, i costi del software, ecc. Per calcolare i costi di sviluppo di un progetto occorrono tre parametri:

- Costi delle attività svolte dai manager e degli ingegneri;
- Costi dell'hardware e del software, quali manutenzione del primo e supporto al secondo;
- Costi per i viaggi e l'addestramento del personale;

Una volta che il contratto di sviluppo di un sistema è stato assegnato, deve essere ridefinito lo schema iniziale del progetto in modo da creare un piano di avviamento per lo sviluppo . In questa fase, sai di più sui requisiti di sistema ma non hai informazioni sulla progettazione o sull'implementazione. Crea un piano con dettagli sufficienti per prendere decisioni sul budget e sul personale del progetto; questo piano è la base per l'allocazione delle risorse del progetto. Il piano di avvio dovrebbe anche definire i meccanismi di monitoraggio del progetto. È ancora necessario un piano di avvio per lo sviluppo agile per consentire l'allocazione delle risorse al progetto.

Il piano del progetto si evolve in continuazione durante il processo di sviluppo, a causa delle modifiche apportate ai requisiti, alle innovazioni tecnologiche e ai problemi di sviluppo. Per questo motivo il piano del progetto dovrebbe essere modificato regolarmente man mano che il progetto procede e si conosce di più sul software e sul suo sviluppo. Assieme alla pianificazione del progetto, anche la stima dei costi e i rischi devono essere rivisti regolarmente.

Prezzo del software

In teoria il prezzo di un sistema software sviluppato per un cliente è semplicemente il costo di sviluppo più il profitto per la società, in pratica non è così semplice. Per calcolare il prezzo occorre tenere in conto di fattori più ampi, di tipo organizzativo, economico, politico e aziendale. Questi aspetti potrebbero richiedere una revisione del prezzo verso il basso o verso l'alto:

- *Basso*, un'azienda può sottostimare un sistema per ottenere un contratto che consenta loro di trattenere il personale per opportunità future o può sottovalutare un sistema per ottenere l'accesso a una nuova area di mercato;
- *Alto*, il prezzo può essere aumentato quando un acquirente desidera un contratto a prezzo fisso e quindi il venditore aumenta il prezzo per consentire rischi imprevisti;

Esiste una strategia di approccio al prezzo del software chiamato "*pricing to win*", in cui i il prezzo del software dipende da quanto lo sviluppatore del software ritiene che l'acquirente sia disposto a pagare. Se questo è inferiore

ai costi di sviluppo, la funzionalità del software può essere ridotta di conseguenza in vista di ulteriori funzionalità da aggiungere in una versione successiva. Possono essere aggiunti costi aggiuntivi al variare dei requisiti e questi possono essere prezzati a un livello più alto per compensare il deficit del prezzo originale.

Sviluppo guidato da piani

Lo sviluppo plan-driven o plan-based è un approccio all'ingegneria del software in cui il processo di sviluppo è pianificato in dettaglio. Lo sviluppo pianificato si basa su tecniche di gestione dei progetti ingegneristici ed è il modo "tradizionale" di gestire grandi progetti di sviluppo software. Viene creato un piano di progetto che registra il lavoro da fare, chi lo farà, il programma di sviluppo ei prodotti di lavoro. I manager utilizzano il piano per supportare il processo decisionale del progetto e come metodo per misurare i progressi.

Gli argomenti a favore di un approccio guidato da piani sono che la pianificazione anticipata consente di tenere attentamente conto delle questioni organizzative (disponibilità di personale, altri progetti, ecc.) E che i potenziali problemi e le dipendenze vengono scoperti prima dell'inizio del progetto, piuttosto che una volta avviato il progetto.

L'argomento principale contro lo sviluppo basato sul piano è che molte delle prime decisioni devono essere riviste a causa dei cambiamenti nell'ambiente in cui il software deve essere sviluppato e utilizzato.

Piano di progettazione

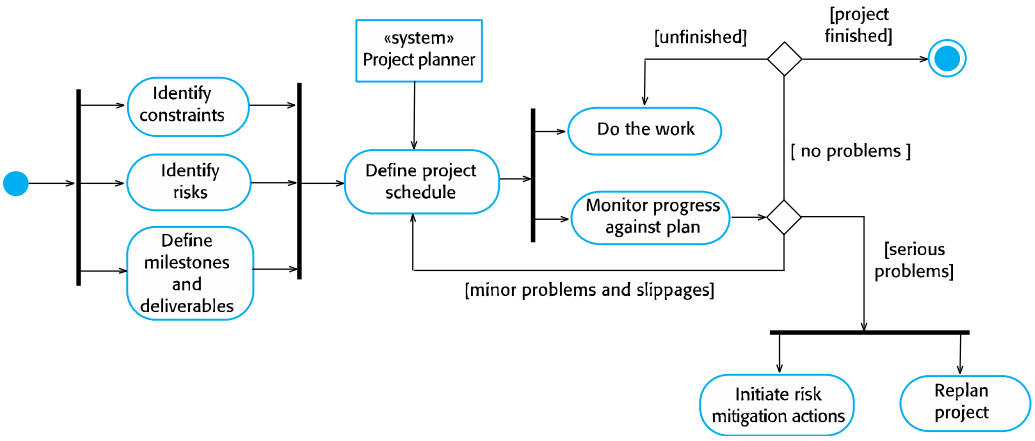
In un progetto di sviluppo basato sul piano, un piano di progetto definisce le risorse disponibili per il progetto, la suddivisione del lavoro e un programma per l'esecuzione del lavoro. I dettagli di un piano di progettazione variano in funzione del tipo di progetto e organizzazione, solitamente però includono le seguenti parti:

- *Introduzione*, descrive brevemente gli obiettivi del progetto e definisce i vincoli (budget, tempistica, ecc) che influiscono sulla gestione del progetto;
- *Organizzazione del progetto,* descrive il modo in cui il team di sviluppo è organizzato, le persone coinvolte e i loro ruoli;
- *Analisi del rischio,* descrive i possibili rischi del progetto, la loro probabilità e strategie per limitarli;
- *Requisiti delle risorse hardware e software,* specifica l'hardware e il software di supporto richiesti per eseguire lo sviluppo. Se occorre acquistare nuove unità hardware, dovrebbero essere incluse anche le stime dei costi e di consegna;
- *Suddivisione del lavoro,* descrive la ripartizione delle attività del progetto e definisce input e output per ogni attività;
- *Tempistica del progetto,* mostra le dipendenze tra le varie attività, il tempo stimato per raggiungere ciascun traguardo (milestone), e l'allocazione delle persone alle singole attività;
- *Meccanismi di monitoraggio e comunicazione,* definisce la documentazione da produrre sulla gestione, i suoi tempi di produzione e i meccanismi di monitoraggio del progetto;

Processo di pianificazione

La pianificazione del progetto è un processo iterativo che inizia quando si crea un piano di progetto iniziale durante la fase di avvio del progetto. I cambiamenti di piano sono inevitabili. Man mano che maggiori informazioni sul sistema e sul team di progetto diventano disponibili durante il progetto, è necessario rivedere regolarmente il piano per riflettere i requisiti, la pianificazione e le modifiche ai rischi. Cambiare gli obiettivi aziendali porta anche a cambiamenti nei piani di progetto. Man mano che gli obiettivi aziendali cambiano, ciò potrebbe influenzare tutti i progetti, che potrebbero dover essere riprogrammati.

Quando si definisce un piano di progetto, è necessario formulare ipotesi realistiche anziché ottimistiche. Problemi di qualche descrizione sorgono sempre durante un progetto e questi portano a ritardi del progetto. Le tue ipotesi iniziali e la pianificazione dovrebbero quindi tenere conto dei problemi imprevisti. Bisognerebbe includere gli imprevisti nel piano in modo che se le cose vanno male, il programma di consegna non viene seriamente interrotto.

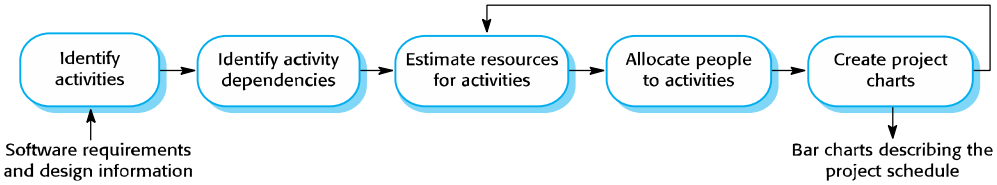


Se, durante lo sviluppo del progetto, si presentano seri problemi che possono portare a ritardi significativi, è necessario avviare azioni di mitigazione del rischio per ridurre i rischi di fallimento del progetto. Insieme a queste azioni, bisogna anche ripianificare il progetto. Ciò può comportare la rinegoziazione dei vincoli e dei risultati del progetto con il cliente. Deve inoltre essere stabilito e concordato con il cliente un nuovo programma per il completamento del lavoro.

Tempistica dei progetti

La tempistica di progetto è il processo per decidere come il lavoro in un progetto sarà organizzato come attività separate e quando e come queste attività verranno eseguite. Si stima il tempo di calendario necessario per completare ogni attività, lo sforzo richiesto e chi lavorerà sulle attività che sono state identificate. È inoltre necessario stimare le risorse necessarie per completare ogni attività, come lo spazio su disco richiesto su un server, il tempo richiesto su hardware specializzato, come un simulatore, e quale sarà il budget di viaggio. Questa tempistica viene affinata e modificata durante lo sviluppo del progetto. Sia i metodi guidati da piani sia i metodi agili richiedono una pianificazione iniziale della progettazione, sebbene un metodo agile richiede meno dettagli.

Nella definizione della tempistica si suddivide il progetto in attività e si valuta il tempo e le risorse necessari per completare ciascuna attività. Si organizza le attività contemporaneamente per fare un uso ottimale della forza lavoro (si ottimizza il tempo). Bisogna ridurre al minimo le dipendenze delle attività per evitare ritardi causati da un'attività in attesa del completamento di un'altra. È dunque un processo che dipende dall'intuizione e dall'esperienza dei project manager.



Quando si definisce la tempistica di un progetto, occorre mettere in conto la possibilità che qualcosa possa andare storto. Stimare la difficoltà dei problemi e quindi il costo dello sviluppo di una soluzione non è un compito facile. La produttività non è proporzionale al numero di persone che lavorano su un'attività, questo è dovuto anche al fatto che non tutte le persone lavorano allo stesso ritmo e hanno la stessa efficienza. La soluzione per un progetto in ritardo non può essere l'aggiunta di persone, questo perché l'aggiunta di persone a un progetto in ritardo lo farà tardare ancora di più a causa delle spese generali di comunicazione. L'imprevisto accade sempre, bisogna dunque sempre consentire gli imprevisti nella pianificazione. Per farlo, una buona regola pratica consiste nel fare le stime come se tutto dovesse andare bene e, poi, aumentare tali stime per tenere conto dei problemi prevedibili, delle attività di emergenza e degli imprevisti.

Presentazione della tempistica

Solitamente per illustrare la pianificazione del progetto si utilizzano le notazioni grafiche, questi mostrano la suddivisione del progetto in attività. Le attività non dovrebbero essere troppo piccole. Dovrebbero volerci circa una o due settimane. Le rappresentazioni grafiche più comuni sono due:

1. *Diagrammi a barre basati su calendario,* mostrano per ogni compito, le persone responsabili, la durata prevista e il tempo di inizio e di fine;

2. *Reti di attività,* mostra le dipendenze tra le varie attività che formano un progetto;

Le attività di progetto (compiti) sono gli elementi di pianificazione di base. Ogni attività ha:

- una durata in giorni o mesi di calendario;
- una stima dello sforzo, che mostra il numero di giorni-persona o mesi-persona per completare il lavoro;
- una scadenza entro la quale l'attività dovrebbe essere completata;
- un punto finale definito, che potrebbe essere un documento, lo svolgimento di una riunione di revisione, l'esecuzione con successo di tutti i test, ecc;

Quando si pianifica un progetto, si può decidere di definire i traguardi (milestone) da raggiungere. I traguardi sono punti della pianificazione rispetto ai quali è possibile valutare i progressi, ad esempio il passaggio di consegne del sistema per i test. I traguardi possono essere associati a un singolo compito o a gruppi di attività correlate. I deliverable sono prodotti di lavoro che vengono consegnati al cliente, ad es. un documento dei requisiti per il sistema.

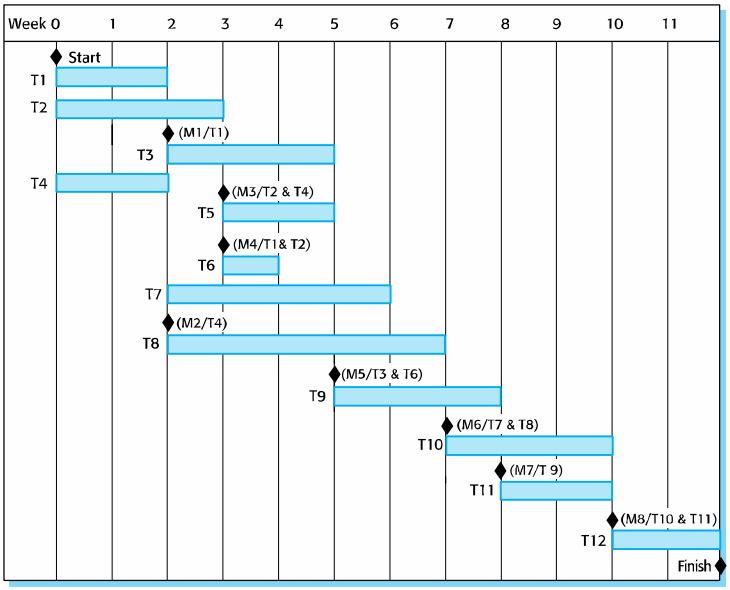


Diagramma a barre delle attività di un progetto

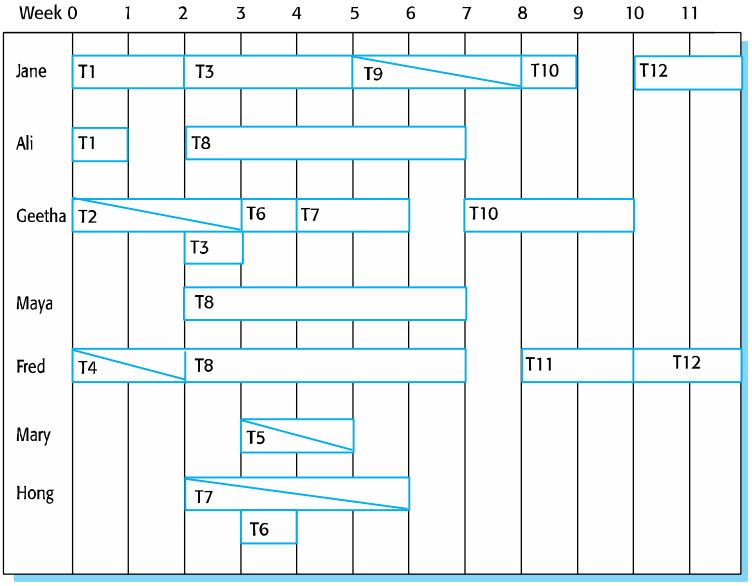


Diagramma di allocazione del personale

Pianificazione agile

I metodi agili di sviluppo del software sono approcci iterativi in cui il software viene sviluppato e consegnato ai clienti in incrementi. A differenza degli approcci basati sul piano, la funzionalità di questi incrementi non è pianificata in anticipo ma viene decisa durante lo sviluppo. La decisione su cosa includere in un incremento dipende dal progresso e dalle priorità del cliente. Le priorità e i requisiti del cliente cambiano, quindi ha senso disporre di un piano flessibile in grado di accogliere questi cambiamenti.

I metodi di sviluppo agile hanno un approccio in due fasi alla pianificazione, che corrisponde alla fase di avviamento nello sviluppo guidato da piani e nella pianificazione dello sviluppo:

1. *Pianificazione delle release*, si fanno previsioni per vari mesi e si scelgono le funzionalità da includere in una release del sistema;

2. *Pianificazione delle iterazioni,* si fa una previsione a breve termine e si decide la pianificazione del successivo incremento del sistema, di solito richiede da 2 a 4 settimane di lavoro da parte del team di sviluppo;

Vi sono diversi approcci, quello Scrum si basa sulla gestione di un backlog di progetto (cose da fare) con revisioni quotidiane di progressi e problemi.