Ingegneria del software

Introduzione

L'ingegneria del software è la parte di progettazione di un sistema software in cui vengono soddisfatte le richieste dei committenti attraverso delle fasi ben precise.
Essa è importante perché tutt'oggi le economie dei paesi industrializzati dipendono dai sistemi informatici, o meglio dai sistemi software, un insieme di componenti il cui scopo è informatizzare una dat6a attività. Ogni componente del sistema è a sua volta suddiviso in sottocomponenti, ognuno con una funzione specifica. la realizzazione di un sistema software, a differenza di un programma , ha bisogno di un gruppo di lavoro complesso (in cui ognuno ha le proprie funzioni) e di un tempo nell'ordine delle settimane/mesi/giorni.

Nascita

L'ingegneria del software è nata negli anni '60 come "soluzione" alla crisi del software, al tempo infatti i computer non potevano far girare cose complesse e i sistemi software erano realizzati con la stessa logica dei programmi, causando requisiti non richiesti, difficoltà da parte dell'utente, molti difetti ecc.

Fasi di realizzazione

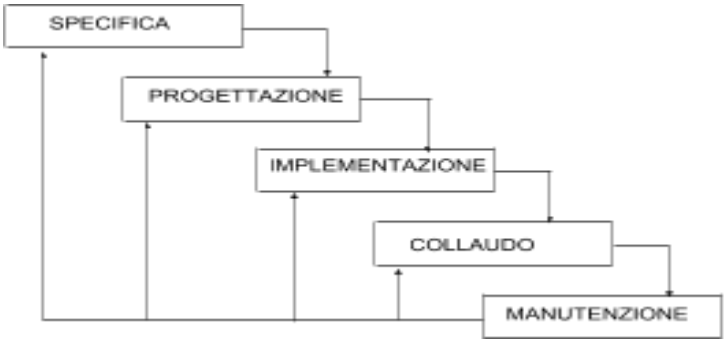
- Specifica: definisce i requisiti funzionali e non, ovvero i servizi che il sistema deve fornire e che danno qualità al prodotto;
- Progettazione: si determina il tutto tramite diagrammi, tra cui l'architettura, il controllo, il comportamento delle componenti e i vari algoritmi;
- Implementazione: scrittura del codice e interrogazione dei moduli;
- Collaudo: ispezionamento del codice riga per riga o tramite tester per controllare la presenza di eventuali bug (verifica) o se tutti i requisiti sono soddisfatti (validazione);
- Manutenzione: correzione di eventuali bug non visti nella fase di collaudo, aggiunta o miglioramenti di nuovi servizi oppure adattamenti del sistema alla piattaforma su cui gira

Modelli di processo software

I modelli di processo definiscono l'ordine in cui vengono svolte le fasi, il più utilizzato è il modello a cascata, in esso vi sono fasi distinte e sequenziali che devono essere completate per passare alla successiva.

Strumenti CASE (Computer-Aided Software Engineering)

Gli strumenti case sono tool che semplificano il processo software, fanno parte di questo gruppo gli editori e i linguaggi che supportano le 5 fasi.



Management

Un processo parallelo a quello software è in management, esso si occupa di gestire le risorse a disposizione, i costi e i tempi di sviluppo e vi è anche l'analisi dei rischi, la persona che si preoccupa di tutto questo è il project manager.

Persone chiave nel processo software

- Committente: persona che richiede il sistema software, decidendo i requisiti del prodotto, valuta i relativi costi e verifica che quest'ultimo lo soddisfi.
- Project Manager: persona che si occupa del management del prodotto, negoziando tempi e costi col committente e pianificando e supervisionando il progetto stesso.
- Sviluppatori: persone che realizzano le basi di progetto.

Tipi di sistemi software

- Sistemi generici: sistemi in commercio il cui obiettivo è vendere più copie possibili, i requisiti vengono scelti in base alle tendenze di mercato e tutti i costi sono a carico dell'azienda che lo produce.
- Sistemi customizzati: sistemi creati ad-hoc per un committente e quindi per il solo utilizzo esclusivo, tutti i requisiti vengono definiti da quest'ultimo e quindi bisogna cercare di soddisfarlo.

Acquisizione di un sistema

Prima di scegliere un tipo di sistema, si sonda il mercato per verificare se vi sono sistemi software simili.
Se è così si adottano i requisiti e si sceglie il tipo di sistema, in seguito si richiedono i prezzi e si sceglie il fornitore.
In caso contrario si inoltrano le richieste ai produttori e nel caso ci siano più interessati, si stipula un contratto con uno di essi e si avvia lo sviluppo.
I due “percorsi” dettati prima si riferiscono rispettivamente all'acquisizione di un sistema generico o customizzato.

Modello fornitore/ sotto-fornitore

Il committente che vuole un sistema software con date richieste si rivolge a un produttore principale quest'ultimo può subappaltare parte del lavoro a sotto produttori esterni a esso.

Computer science e ingegneria del software

Da sola, la computer science non è sufficiente per realizzare un sistema software, oltre a essa serve anche l'ingegneria del software, la conoscenza del dominio applicativo del sistema e capacità creativa per trovare soluzioni ad-hoc.

Ingegneria tradizionale vs Ingegneria del software

Aspetti comuni:

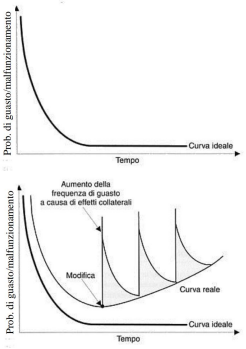
- Acquisizione del sistema,
- Modello fornitore / subfornitore,
- 2 tipi di requisiti,
- Processi di sviluppo;
- Modelli di specifica e progettazione,

Differenze:

- Il software viene sviluppato e non fabbricato;
- Il software richiede strumenti per l'implementazione ;
- Le fasi del processo software sono meno distinte ;
- Il software ha un costo marginale nullo e tutte le copie hanno gli stessi difetti;
- Il software è soggetto a modifiche, la manutenzione è più costosa ed è più complessa e frequente.

Curva dei guasti del software

In teoria la curva dei guasti di un software dovrebbe essere una linea che “si raddrizza” nel corso del tempo. Nel caso reale, però, la correzione degli errori e l'implementazione di nuovi servizi causano a loro volta nuovi possibili errori, causando così il cosiddetto “deterioramento” del software.



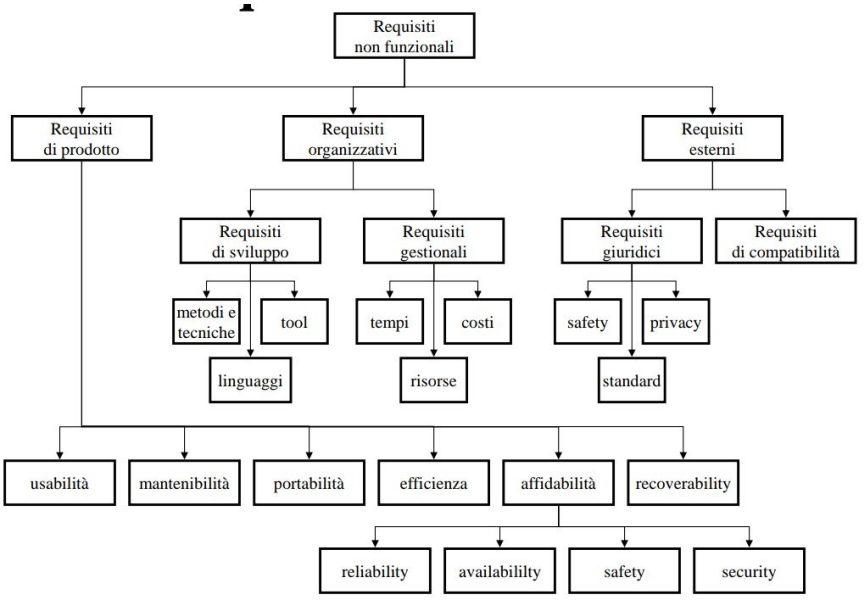
Specifica

La specifica descrive i requisiti funzionali e non con diversi livelli di dettaglio: una descrizione ad alto livello è utile per essere comprensibile da tutte le persone chiave mentre quella a basso livello è adatta a chi ci lavora direttamente dato che contiene specifiche formali e/o matematiche.
Per i requisiti funzionali si descrive cosa accade durante l'interazione utente-sistema, cosa accade in seguito a un input e cosa in particolari situazioni.
Quando il sistema è considerato un'unica entità non si descrive il funzionamento interno del sistema dal momento che si fa durante la progettazione .
I requisiti non funzionali descrivono le qualità del sistema, esse vengono dette proprietà complessive quando riguardano il sistema nel suo complesso e non entra nel dettaglio, oppure emergenti quando " emergono" dal funzionamento dopo l'implementazione.
I requisiti non funzionali possono essere:

- Di prodotto: definiscono la qualità, l'efficienza, l'affidabilità, ecc…di un sistema;
- Organizzativi: possono essere di sviluppo o di management, i primi riguardano la parte di sviluppo specificando i tool e le tecniche da utilizzare mentre i secondi descrivono la parte di management, definendo i tempi, i costi e le risorse.
- Esterni: dipendono da fattori esterni, quelli giuridici riguardano la privacy e la sicurezza mentre i secondi si riferiscono alla piattaforma su cui gira il sistema .

Requisiti di prodotto

- Usabilità: indica il grado di facilità dell'utente nella comprensione di un sistema, l'interfaccia utente deve essere intuitiva e curata dato che un utente è solito valutarla "a vista", l'uso del sistema deve essere documentato e ci può essere bisogno di training per adattare gli utenti e quindi migliorare l'usabilità.



- Manutenibilità: grado di facilità per la manutenzione in modo da fixare bug, implementare nuove feature o semplicemente adottarlo a nuovi sistemi operativi;
- Portabilità: capacità di migrazione da un ambiente a un altro;
- Recoverability: capacità di ripristino dello stato e dei dati del sistema dopo un crash;

- Efficienza: livello di prestazione del sistema, si misura attraverso i tempi di risposta, il numero medie di richieste risolte per unità di tempo e la quantità di risorse consumate;
- Affidabilità: grado di fiducia nel funzionamento corretto di un'azione del sistema, vi sono differenti misure:

    - Reliability: capacità di fornire servizi come definiti nella specifica;
    - Availability: capacità di fornire servizi nel momento richiesto;
    - Safety: capacità di operare senza causare danni;
    - Security: capacità di proteggersi dalle intrusioni

Sistemi Critici

Un sistema è critico quando il suo funzionamento non corretto può portare a conseguenze disastrose alle persone e all'ambiente (Safety Critical System) oppure a perdite economiche (Business Critical System) in questi sistemi l'affidabilità è importante.

Costo dell'affidabilità

Il costo dell'affidabilità cresce esponenzialmente al grado richiesto, vi sono infatti tecniche di sviluppo e di tolleranza ai guasti più costose per soddisfare il grado richiesto, tutto questo è giustificato dal fatto che vengono ridotti i rischi di danno dovuto a funzionamenti scorretti. Alcuni requisiti come affidabilità ed efficienza potrebbero andare in conflitto tra loro, bisogna quindi trovare dei compromessi.

Processo di specifica

La specifica è a sua volta organizzata in fasi:

- Studio di fattibilità: raccolta di informazioni per indicare se una richiesta è fattibile o meno, anche in base a tempi e costi, da essa si produce il rapporto di fattibilità;
- Deduzione dei requisiti: raccolta di informazioni per permettere la definizione dei requisiti, in questa fase si studia il dominio applicativo e si interrogano il committente e gli sviluppatori che hanno lavorato a progetti simili, come output si ottiene la definizione ad alto livello dei requisiti.
  Un altro modo per ottenere (operazioni)* è lo stakeholder, ovvero si raccolgono informazioni dalle persone riguardanti il progetto, gli unici problemi sono che alcuni potrebbero non saper indicare cosa vogliono, omettendo informazioni troppo ovvie e uno stesso requisito potrebbe essere definito in modi differenti.
  Come scritto in precedenza questa fase restituisce una descrizione ad alto livello dei requisiti, tuttavia dato che è scritta in linguaggio naturale (è comunque possibile l'utilizzo dei diagrammii) vi possono essere delle incomprensioni dovute a confusione e mescolanza dei requisiti.
- Analisi dei requisiti: vengono classificati e organizzati i requisiti assegnandogli una priorità, in questa fase è possibile individuare conflitti tra requisiti, requisiti non realizzabili e vengono effettuati dei compromessi per risolvere i problemi citati prima.

Inoltre i requisiti vengono modellati analiticamente attraverso diagrammi e notazioni che li descrivono nel dettaglio.
I requisiti di sistema, dato che sono alla base della progettazione, devono essere riscritti con un linguaggio specifico onde evitare la lunghezza e l'ambiguità del linguaggio naturale.
I requisiti non funzionali vengono invece valutati per misure di efficienza, affidabilità e usabilità, quali i tempi di risposta e la memoria occupata, il tempo e la probabilità di malfunzionamento e il tempo di addestramento degli utenti.
Come output questa fase restituirà un documento dei requisiti contenenti un' introduzione, un glossario e i requisiti utente e di sistema, esso verrà letto da committente dal project manager, dai progettisti, dai tester e dai manutentori;

- Validazione dei requisiti: controlla che tutte le proprietà siano state soddisfatte dai requisiti, infatti per questi ultimi la stabilità è importante.
  Inoltre questa fase serve ad evitare errori di specifica durante le fasi successive.
  Le proprietà sono:
  - Completezza: tutti i requisiti richiesti dal committente devono essere documentati;
  - Coerenza: la specifica dei requisiti non deve contenere informazioni contraddittorie;
  - Precisione: l'implementazione di un requisito deve essere unica e non ambigua;
  - Realismo: i requisiti possono essere implementati date le risorse disponibili;
  - Tracciabilità della sorgente dei requisiti e del progetto, attraverso una matrice avente dipendenze forti (D) e deboli (R). La tracciabilità della sorgente è utile per reperire la fonte delle informazioni su un dato requisito, quella dei requisiti per l'individuazione degli stessi e quella del progetto per individuare i componenti per realizzare un requisito.

Tecniche di validazione

- Revisione: un gruppo di revisori controlla i requisiti per individuare anomalie e omissioni, solitamente appartengono al team di sviluppo o all'azienda committente.
- Costruzione di prototipi: vengono generati e mostrati al committente uno o più prototipi per verificare la comprensione dei requisiti.
  I problemi risolti ripetono le fasi precedenti alla validazione, tuttavia è raro scoprirli tutti.

Progettazione

La progettazione è la fase in cui vengono definite tutte le cose che verranno implementate nella fase successiva, essa è suddivisa in:

- Progettazione architetturale: definisce la struttura dei componenti, i metodi di condivisione dati e controllo degli stessi e si modella il loro comportamento, ovvero tutte le interazioni con lo scopo di svolgere una certa funzione;
- Progettazione delle strutture dati: vengono definite tutte le strutture che mantengono i dati del sistema;
- Progettazione degli algoritmi;
- Progettazione della GUI.

La progettazione viene solitamente rappresentata attraverso diagrammi UML, modelli ER, DataFlow e diagrammi di flusso.
I componenti del sistemi possono essere sottosistemi o moduli, i primi sono parti del sistema che svolgono una certa attività, i secondi invece sono parti dedicate a funzioni legate a quest'ultima. I moduli di una sottosistema sono formati da un'interfaccia, contenente i servizi che esso offre, e un corpo in cui vi è l'implementazione. Nei moduli si pratica l'information hiding, ovvero si nasconde il corpo e lo si rende accessibile attraverso l'interfaccia, in questo modo si evita che eventuali modifiche abbiano effetto sul resto del sistema.

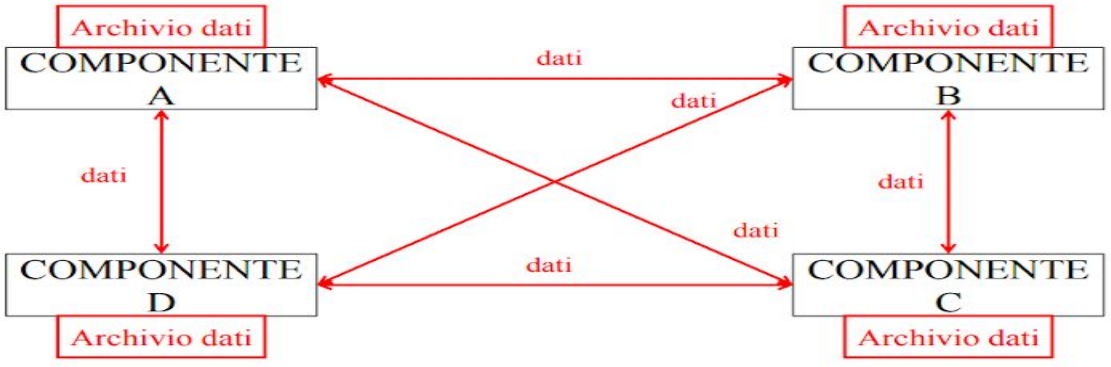
Stili architetturali

Gli stili architetturali sono modelli standardizzati che fungono da riferimento per la progettazione architetturale.

Condivisione dati

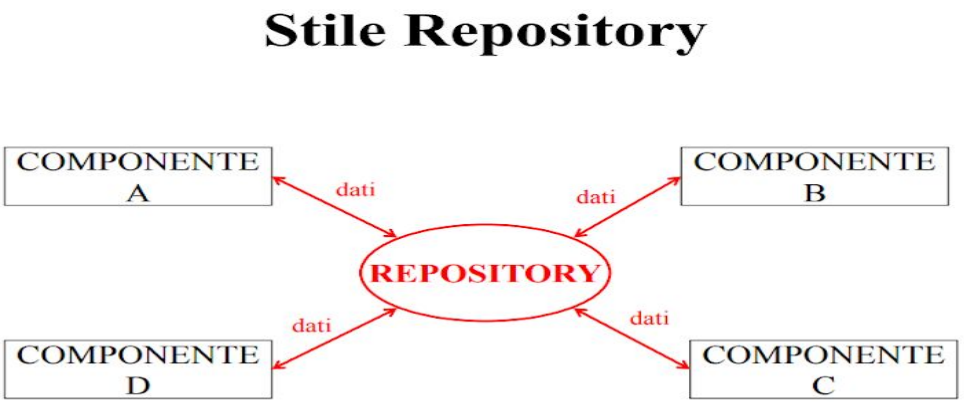
La condivisione dati permette lo scambio di dati tra componenti, esso può avvenire in due modi:

- Passaggio di dati: ogni componente ha un proprio archivio, da essa vengono prelevati i dati quando richiesti da ulteriori componenti esterne;



- Repository: I dati sono condivisi in un database centrale in cui ogni componente può accedere. Essa è efficiente per grandi quantità di dati e la loro gestione è gestita dalla repository stessa, tuttavia la struttura dati deve essere accettata da tutte le

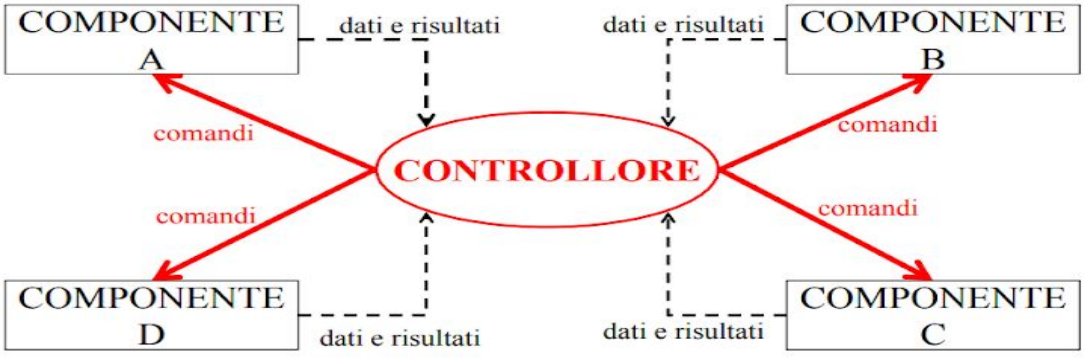
componenti (creando compromessi e/o condividendo le stesse politiche di sicurezza, backup e recovery) e avere un alto grado di affidabilità, dato che il fallimento del repository impedirebbe l'accesso ai dati.



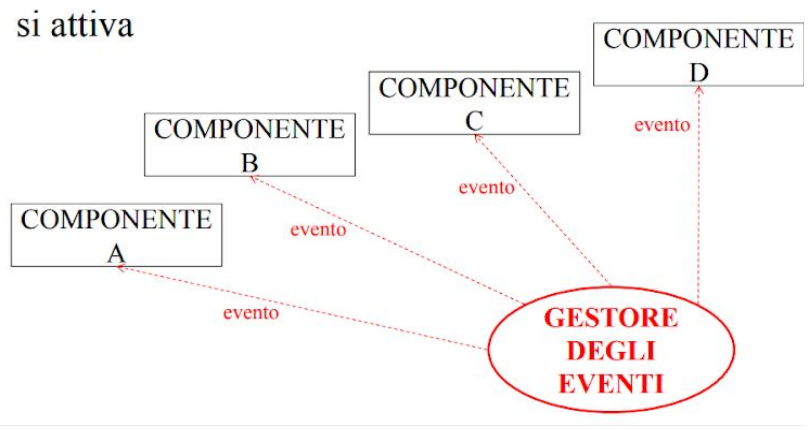
Controllo

I componenti devono essere controllati affinché forniscano il loro servizio al momento giusto, esso può essere:

- Centralizzato: un componente, detto controllore, ha il compito di controllare tutte le altre attivandole e disattivandole. Il controllore esegue periodicamente il control loop: il controller può controllare se le componenti hanno prodotto dati, attivarle/disattivarle in base a essi, passarle ulteriori dati oppure raccogliere i risultati;



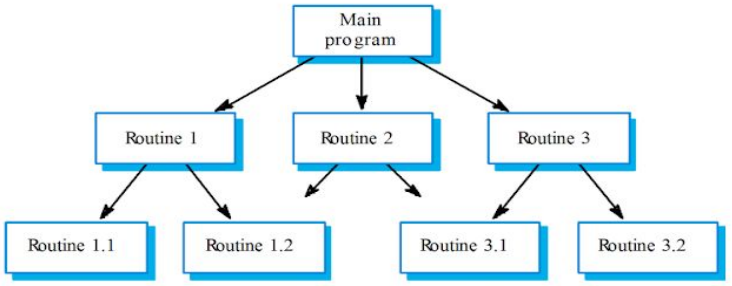
- Basato su eventi: ogni componente si attiva reagendo a eventi determinati dall'ambiente o da altre componenti. Ogni componente è dedicata a specifici eventi, quando si verificano, essa si attiva. Gli eventi vengono gestiti dal gestore degli eventi, esso lo rileva e lo notifica a tutte le componenti in modo selettivo o meno, ogni componente è il controllore di sè stesso, infatti decide se un evento è rilevante o meno e si comporta di conseguenza;



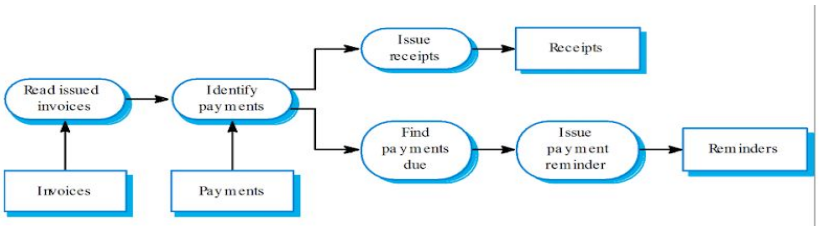
- Call-return: una componente principale attiva delle componenti che a loro volta ne attivano altre.

Modellazione del comportamento

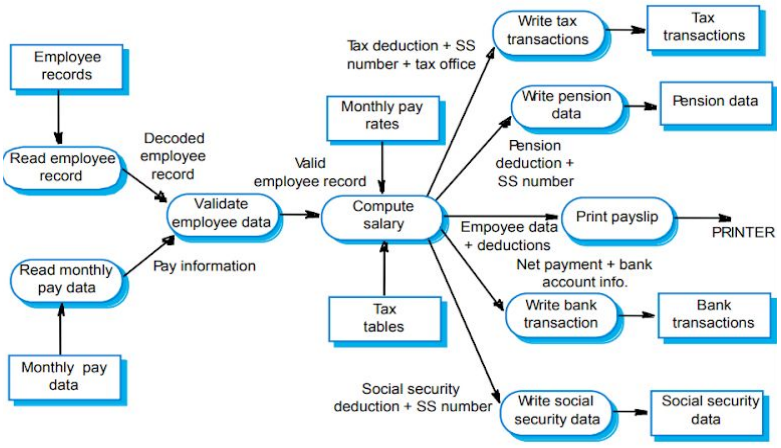
Il comportamento delle componenti per lo svolgimento di un dato servizio può essere modellato attraverso due stili:



- Modello data-flow: vi è una componente filter che, dato un input, genera un output e una componente pipe che trasferisce i dati tra filter, creando flussi di dati. I flussi e l'elaborazione dei dati possono essere sequenziali o in parallelo. Questo modello non prevede la gestione degli errori, essi infatti possono determinare l'interruzione di un flusso di dati;
  Esempio:



- Modello a oggetti: il sistema viene strutturato in oggetti aventi attributi e operazioni, essi si scambiano dei messaggi, ovvero delle invocazioni di operazioni e valori di ritorno. Gli oggetti vengono creati come istanze di una classe, esse infatti riflettono le entità del mondo reale;

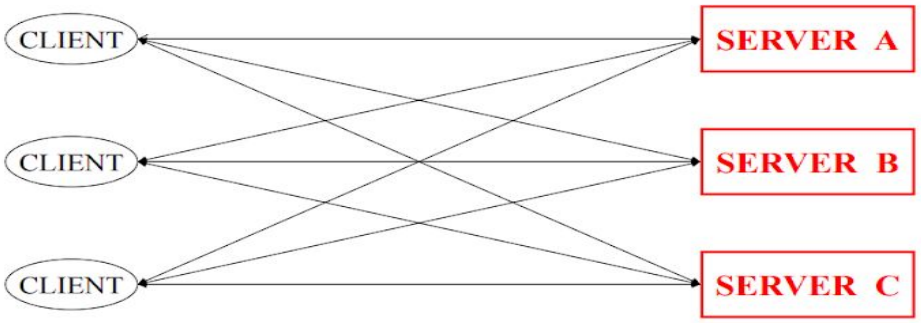


Strutturazione

La strutturazione indica il come i vari componenti sono distribuiti all'interno del sistema

Modello Client-Server

Il modello client-server è composto da componenti che forniscono servizi (i server) e componenti che ne usufruiscono (i client) posti nella stessa macchina quando il sistema non è distribuito, i client non comunicano tra loro direttamente ma utilizzano i server come intermediario. La comunicazione tra client e server avviene tramite un protocollo Request/Reply: i client richiedono un servizio al server attraverso una chiamata di una funzione e questi ultimi rispondono.



Modello stratificato

In questo modello, il sistema è organizzato in livelli, ognuno di essi fornisce dei servizi a quelli adiacenti.

Sistemi distribuiti

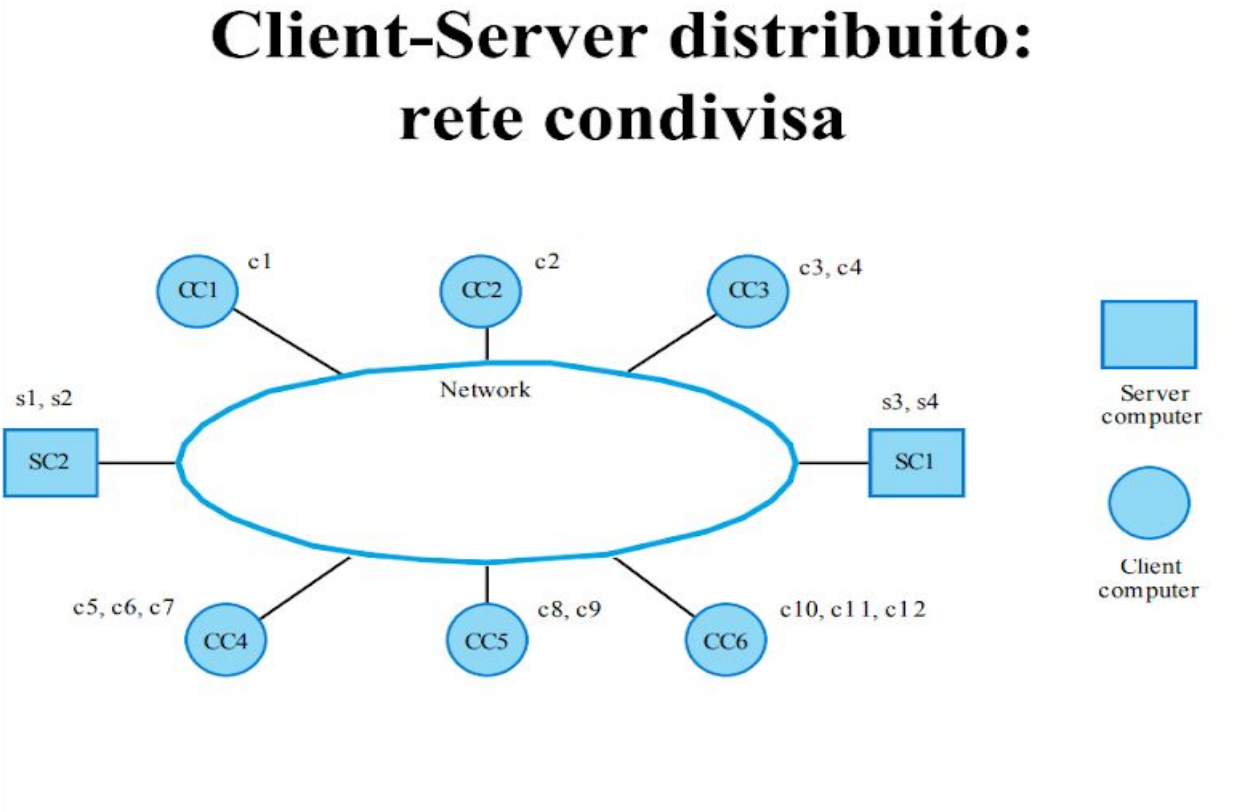
I sistemi distribuiti sono sistemi in cui l'elaborazione è appunto distribuita su più macchine, anche di varie architetture. Le caratteristiche di questi sistemi sono:

- La condivisione delle risorse hardware e software;
- L'apertura, ovvero l'utilizzo di componenti di vari produttori;
- Più processi sono eseguiti simultaneamente su più macchine;
- La scalabilità, si possono quindi aggiungere componenti, quando necessario, per aumentare la capacità di calcolo;
- Tolleranza agli errori: attraverso la condivisione di dati e servizi, è possibile continuare a operare nonostante si verifichino dei fallimenti.

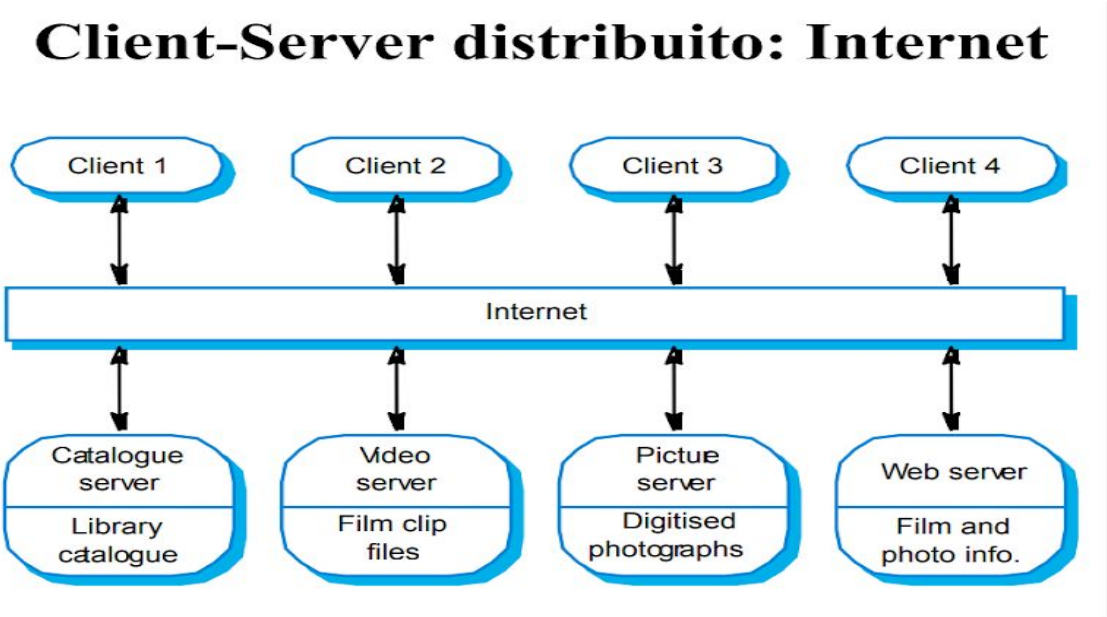
I sistemi distribuiti tuttavia non mancano di problemi, la loro natura, infatti, li rende più complessi e vulnerabili rispetto a quelli centralizzati, in più i tempi di risposta non sono prevedibili e, a causa dell'utilizzo di hardware e sistemi operativi differenti tra loro, vi è una maggior complessità nella gestione (risolvibile coi middleware, software che permettono la comunicazioni tra parti eterogenee nel sistema distribuito).

Modello Client-Server distribuito

Rispetto a quello non distribuito, client e server si trovano tutti in differenti macchine e comunicano tra loro attraverso la rete, infatti i primi sanno localizzare i secondi e conoscono i servizi che erogano, questi ultimi non hanno bisogno di conoscere l'identità del client e quanti di questi sono presenti. Un modello di questo tipo può funzionare attraverso una rete



condivisa oppure tramite Internet: nel primo caso ogni macchina è connessa una rete locala e tramite essa comunicano coi server, nel secondo si fa la stessa cosa ma si utilizza internet.



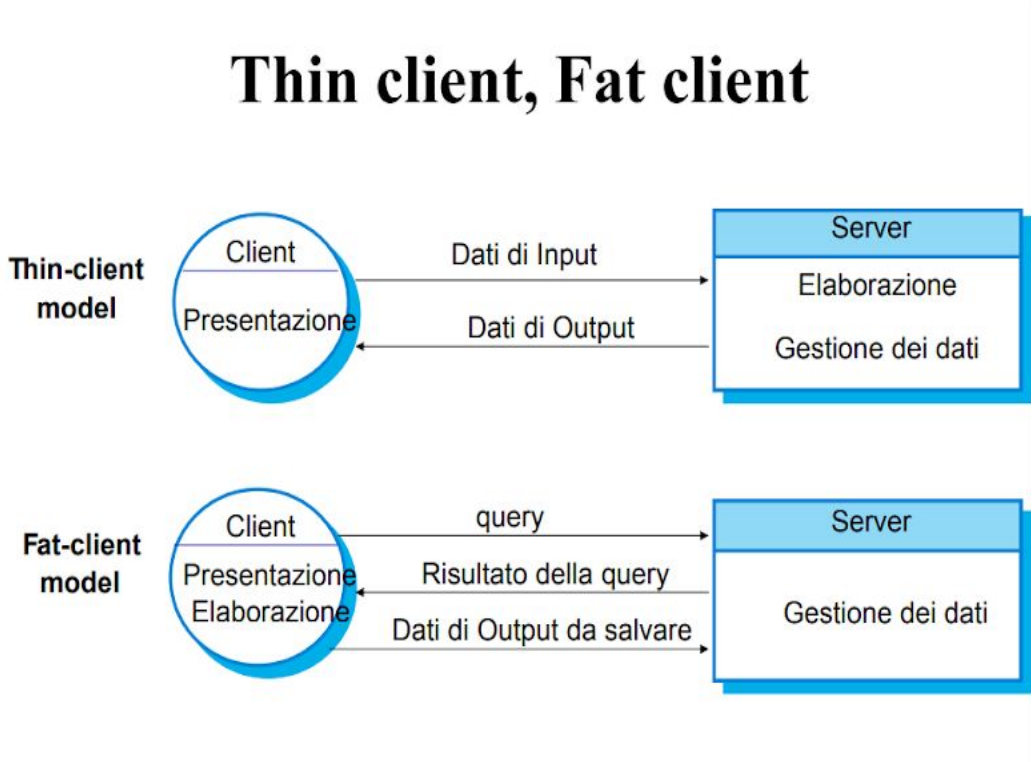
Modello stratificato distribuito

Ogni livello del sistema stratificato viene dato a una specifica macchina, ad esempio:

- La gestione dei dati viene effettuata da una server Database;
- L'elaborazione viene effettuata da un server normale;
- La presentazione viene fatta dalle macchine utente inviando input ai server e visualizzando gli output che ricevono;

Questo modello può essere:

- 2-tiers: i livelli sono divisi in macchine client e macchine server, nell'esempio precedente si può dare l'elaborazione e la gestione dei dati ai server e solo la presentazione ai client (soluzione thin client) oppure si può si può dare al server la gestione dati e lasciare al client il resto (soluzione fat client). I thin client, dal momento che devono solo presentare i dati all'utente, sprecano capacità di calcolo dal momento che non è utilizzata, a loro volta i fat client hanno bisogno di continui aggiornamenti per continuare a lavorare. Per ovviare ciò si utilizza il codice mobile: parte dell'elaborazione viene eseguita dai client mentre l'altra dal server.
  <slide 45>

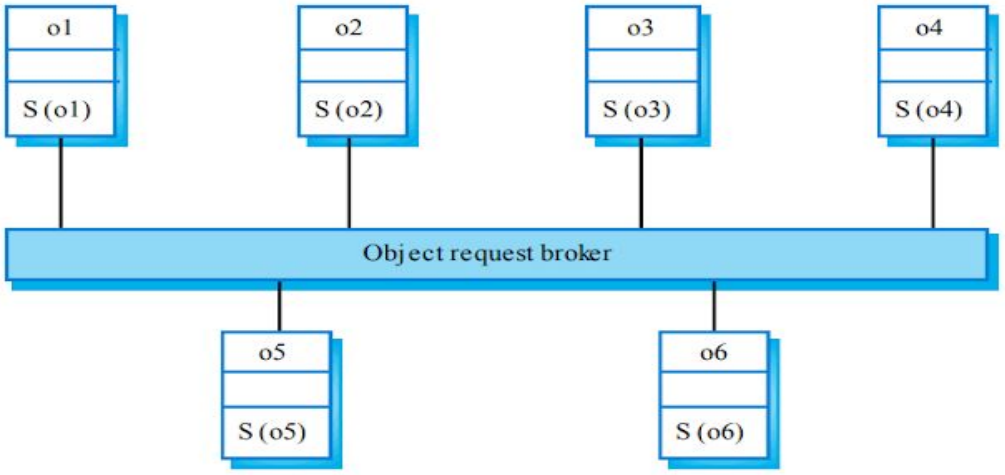


- 3-tier e n-tier: utilizzano lo stesso ragionamento del punto precedente con la differenza che i livelli sono rispettivamente suddivisi in 3 macchine oppure molteplici.

Architettura a oggetti distribuiti

In questo tipo di architettura, il sistema è composto da un insieme di oggetti (distribuiti su un insieme di macchine e realizzati anche con linguaggi differenti, quindi possono girare su piattaforme eterogenee), ognuno di essi può richiedere o fornire servizi. La comunicazione tra oggetti avviene tramite il middleware ORB (Object Request Broker).
<slide 53>

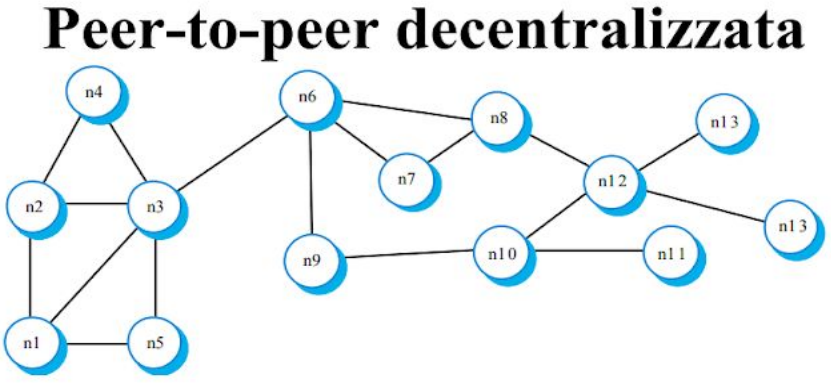
Architettura ad oggetti distribuiti



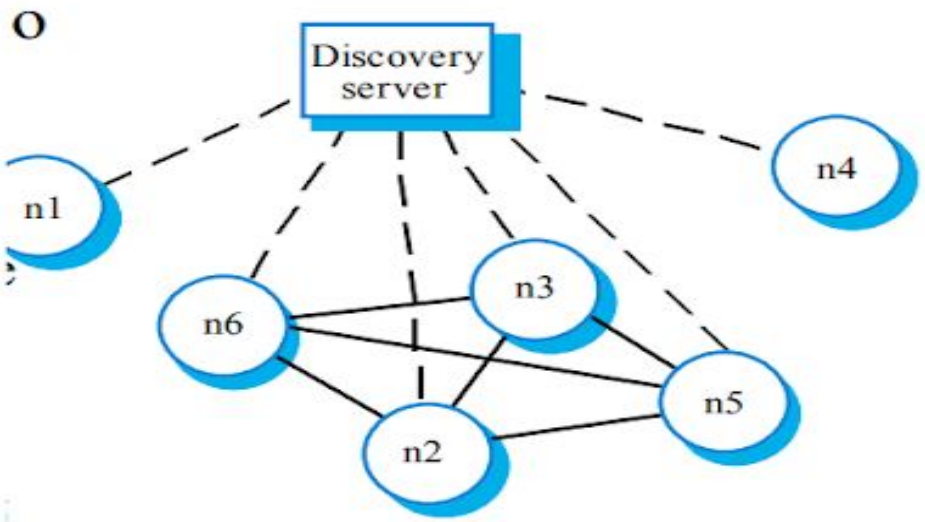
Architettura P2P

Nell'architettura P2P si trae vantaggio della capacità di calcolo e della memoria delle macchine connesse alla rete, tutti i nodi hanno lo stesso ruolo, fanno girare la stessa applicazione e possono rilevare/connettersi con altri nodi. P2P può essere:

- Decentralizzata: ogni nodo comunica solo coi vicini, questi ultimi inoltrano le richieste quando non possono soddisfarle, la risposta percorrerà lo stesso percorso ma al contrario.
  <slide 57>

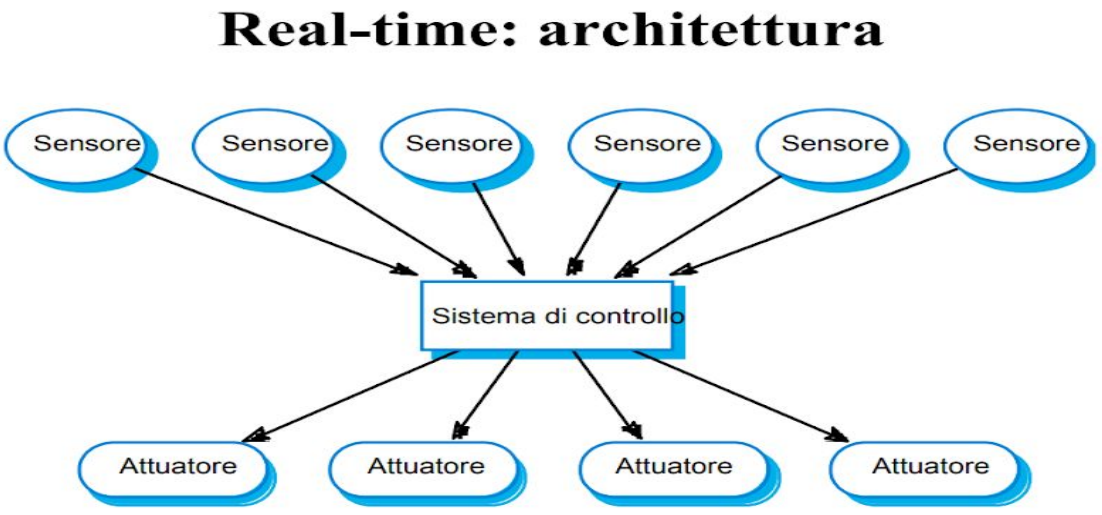


- Semi-centralizzata: vi è un nodo server che indica a tutti gli altri i servizi offerti di ogni nodo, successivamente richiedente e fornitore si contatteranno senza chiamare il server.
  <slide 58>



Sistemi real-time

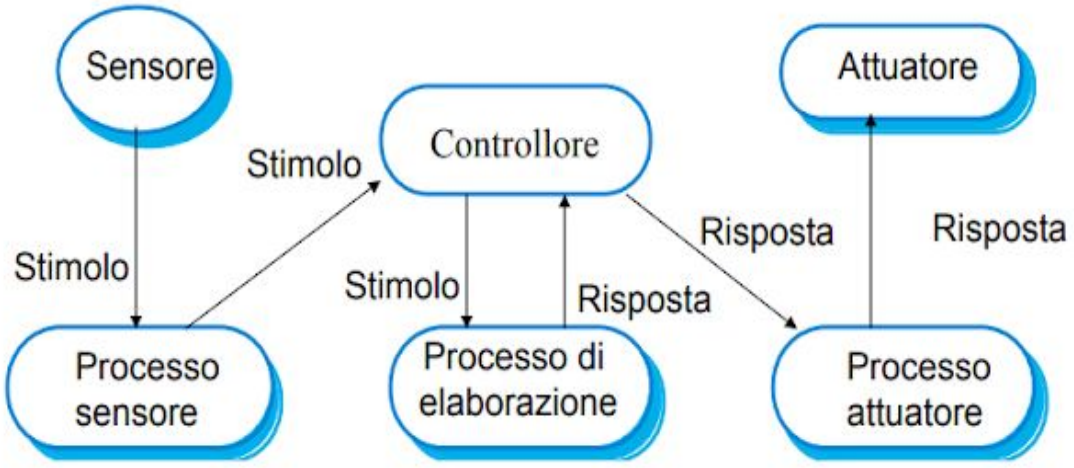
I sistemi real-time controllano l'ambiente in cui operano attraverso sensori (componenti che raccolgono dati dall'ambiente) e attuatori (componenti che modificano l'ambiente), il tutto controllato da specifiche componenti software. Un sistema di questo tipo funziona correttamente quando i risultati sono corretti e vengono restituiti entro un certo tempo. Un sistema real-time funziona reagendo agli stimoli, essi possono essere periodici (si verificano a intervalli regolari) oppure aperiodici, a ogni stimolo corrisponde una risposta.
<slide 62>



Il sistema di controllo di P2P è composto da:

- Processi sensore: gestiscono i sensori;
- Processi attuatori: gestiscono gli attuatori;
- Processi di elaborazione: generano risposte elaborando gli stimoli;
- Interfaccia utente;
- Gestore fallimenti;
- Controllore: invia comandi ai processi;

Attraverso queste componenti, il sistema di controllo può raccogliere dati dall'ambiente utilizzando i sensori, elaborarli e inviare la risposta ottenuta agli attuatori i quali la realizzano.
<slide 64>



Implementazione

L'implementazione è la parte in cui si scrive il codice del sistema, i sottosistemi vengono tipicamente in parallelo assegnando un team a ognuno, per farlo si possono utilizzare componente commerciali, il loro costo è infatti inferiore rispetto a una componente ad-hoc. La composizione è la fase finale in cui tutti i sottosistemi vengono integrati in un unico sistema, in essa si possono scoprire problemi di interfacciamento e di conseguenza risolverli. L'implementazione dei sottosistemi può avvenire in due modi:

- Nell'approccio "Big Bang" i sottosistemi vengono implementati in contemporanea, quindi devono essere completati nello stesso termine temporale (cosa molto difficile) e c'è più difficoltà a localizzare gli errori;
- Nell'approccio incrementale i sottoinsiemi vengono implementati uno alla volta, di conseguenza la consegna deve essere coordinata (e non contemporanea) e, in presenza di errori, vi è molta probabilità che riguardi l'ultimo implementato.

Collaudo

La fase di collaudo è la fase in cui si ricercano e correggono i difetti del sistema implementato, più precisamente si esegue:

- La verifica: si controlla che ogni funzione del prodotto non abbia malfunzionamenti (bug) e , nel caso ci siano, correggerli. I malfunzionamenti sono manifestazioni visibili dei bachi che causano comportamenti imprevisti, essi vengono causati dall'errore umano;
- La validazione: si controlla che ogni funzione soddisfi i requisiti del committente, lo scopo principale è trovare le anomalie (servizi non forniti nel modo previsto) e le omissioni(servizi previsti ma non implementati).

La causa di un difetto è solitamente un errore che, a partire da una precedente fase, si è propagato. La ricerca di un errore può essere:

- Statica (Ispezione): si cerca l'errore ispezionando il codice e la documentazione, il sistema in questo caso non viene avviato;
- Dinamica (Testing): si cerca l'errore mandando in esecuzione il sistema osservandone il comportamento con dei test-case.

Ispezione

A volte si preferisce l'ispezione perchè il testing può essere costoso, dato che si ha bisogno di molti test-case per collaudare ogni parte del codice. I test-case possono correggere uno o più difetti, tuttavia ce ne possono essere di nascosti: non si sa se i comportamenti anomali successivi sono causati dalla correzione o da altri difetti. Oltre a ciò, l'ispezione permette la correzione del codice anche quando il (sotto) sistema è incompleto, cosa non possibile con il testing. Portabilità e mantenibilità sono requisiti non funzionali che possono essere collaudati solo con l'ispezione, infatti essi dipendono dal tipo di linguaggio e dalla strutturazione del codice, tutti gli altri non sono collaudabili in questo modo. Una strategia comune è effettuare prima un'ispezione e in seguito un testing, in questo modo più difetti si scoprono nella prima fase, meno test-case bisogna definire ed eseguire. L'ispezione è effettuata da un team che analizza il codice e segnala i possibili difetti, ognuno si concentra su un determinato aspetto da collaudare e col supporto della documentazione e di una check-list, quest'ultima deve essere compilata durante la fase per segnare gli errori relativi ai dati, il controllo, l'I/O, eccetera.

Classe di errore	Controllo di ispezione
Errori nei dati	Tutte le variabili del programma sono inizializzate prima che il loro valore sia utilizzato? Tutte le costanti hanno avuto un nome? Il limite superiore degli array è uguale alla loro dimensione o alla loro dimensione -1? Se si utilizzano stringhe di caratteri, viene assegnato esplicitamente un delimitatore? Ci sono possibilità di buffer overflow?
Errori di controllo	Per ogni istruzione condizionale la condizione è corretta? È certo che ogni ciclo sarà ultimato? Le istruzioni composte sono correttamente messe fra parentesi? Se è necessario un break dopo ogni caso nelle istruzioni case, è stato inserito?
Errori di Input/Output	Sono utilizzate tutte le variabili di input? A tutte le variabili di output viene assegnato un valore prima che siano restituite? Input imprevisti possono causare corruzione?
Errori di interfaccia	Tutte le chiamate a funzione e a metodo hanno il giusto numero di parametri? Il tipo di parametri formali e reali corrisponde? I parametri sono nel giusto ordine? Se i componenti accedono alla memoria condivisa, hanno lo stesso modello di struttura per questa?
Errori nella gestione della memoria	Se una struttura collegata viene modificata, tutti i collegamenti sono correttamente riassegnati? Se si utilizza la memoria dinamica, lo spazio è assegnato correttamente? Lo spazio viene esplicitamente deallocato quando non è più richiesto?
Errori di gestione delle eccezioni	Sono state prese in considerazione tutte le possibili condizioni d'errore?

Processo di ispezione

<slide 12>

Il processo di ispezione è formato da sei fasi:

- Pianificazione: In questa fase viene selezionato il team e si controlla se il materiale (documentazione e codice) sia completo;
- Introduzione: Il moderatore organizza una riunione in cui l'autore del codice descrive lo scopo del programma, viene visionato il materiale e si stabilisce la check-list;
- Preparazione individuale: viene studiato il materiale e si cercano i difetti nel codice in base alla check-list e all'esperienza personale;
- Riunione di ispezione: vengono dichiarati i difetti individuati dagli ispettori;
- Rielaborazione: il programma viene modificato dall'autore correggendo i difetti;
- Prosecuzione: il moderatore decide se è necessario un altro processo di ispezione e, se così fosse, effettuarlo.

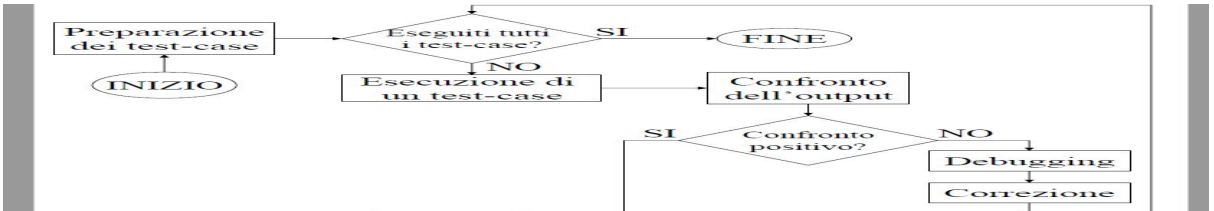
Durante l'ispezione il codice può essere analizzato con gli strumenti CASE, precisamente con:

- l'analisi del flusso: controllo che il flusso dei dati sia corretto, pulito e che non ci sia codice irraggiungibile;
- l'analisi dell'uso dei dati: si controllano se le variabili sono inizializzate e utilizzate nel modo corretto;
- l'analisi delle interfacce: si controlla la consistenza delle funzioni, se vengono tutte invocate e che il loro risultato sia utilizzato;
- Analisi della gestione della memoria: si controlla che tutti i puntatori siano assegnati e se ci sono errori relativi a essi;
- eccetera;

Alcuni di questi strumenti sono inclusi nei compilatori e forniscono informazione di supporto all'individuazione dei difetti, il loro utilizzo non è tuttavia sufficiente dato che possono trovare, ad esempio, inizializzazioni mancanti ma non quelle errate.

Testing

Come già detto, il testing valuta il comportamente del sistema in esecuzione attraverso dei test-case, tuttavia questi possono trovare dei difetti ma non è detto che tutto il sistema non li



abbia in quanto posso aver controllato una parte del sistema esente da difetti, mentre un'altra potrebbe averli. Come per l'ispezione, vi sono delle fasi:

- Preparazione dei test: vengono preparati i test-case inserendo input e output;
- Viene eseguito un test case utilizzando gli input dati;
- confronto il risultato con gli output dati, se non vi è corrispondenza, vi è un errore e quindi lo si va a correggere, altrimenti passa al prossimo test-case;
- Il ciclo continua finchè ci sono test-case da eseguire.

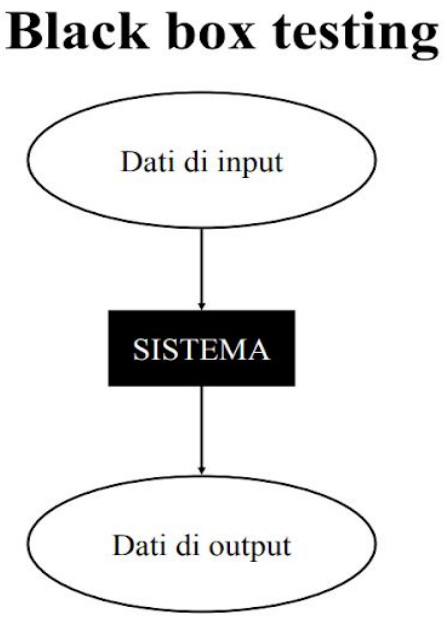
In caso di errore, per prima cosa si effettua il debugging, ovvero si controlla l'esecuzione del programma controllando ogni istruzione e i valori delle variabili per individuare l'errore, dopo si passa alla correzione e infine vi è il testing di regressione, in questa fase si ripete l'ultimo test-case per verificare che sia giusto e tutti quelli precedenti per controllare se le modifiche non abbiano impattato anche su essi.

Vi possono essere due possibili approcci per il testing:

- Black box: sono visibili solo input e output del sistema, l'elaborazione non lo è e la scelta dei test-case è basata sulla conoscenza di input e output;
- White box: l'elaborazione è visibile e la scelta dei test-case è basata sul codice;

Un test-case è formato dagli input e dagli output attesi, essi possono essere infiniti dal momento che anche i possibili input lo possono essere. Tuttavia il sistema può essere testato un numero finito di volte, di conseguenza bisogna utilizzare un criterio che permetta di effettuare il testing in modo efficace.

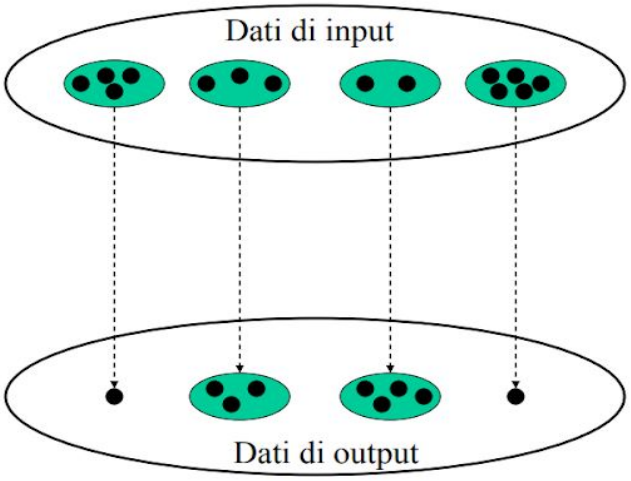
Black box testing



Il black box testing testa il sistema suddividendo gli input in partizioni di equivalenza da cui vengono ricavati i test-case, esse vengono scelte in modo da ottenere lo stesso risultato in output oppure quando esso appartiene a una delle partizioni.

Solitamente il sistema si comporta allo stesso modo per tutti gli elementi di uno stesso sottoinsieme. Per questo tipo di testing si utilizza il seguente ragionamento: se un test è vero/falso per alcuni elementi di un sottoinsieme, allora lo è anche per tutti gli altri della stessa partizione.
<slide 27>

Partizioni di equivalenza (2)



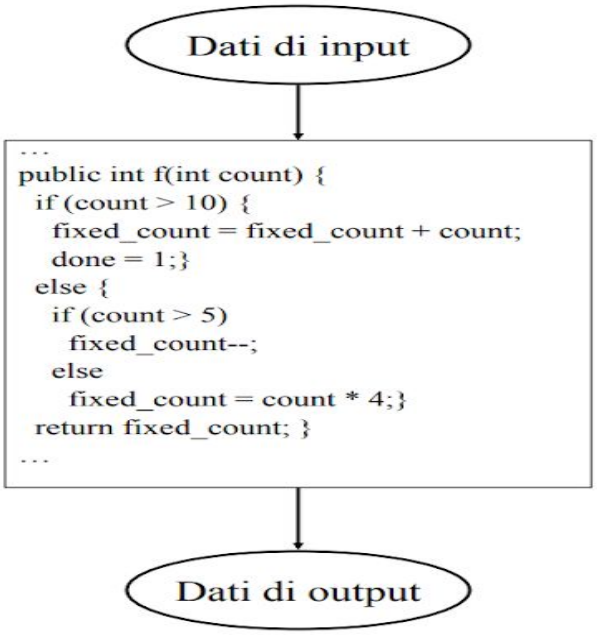
Per individuare le partizioni, si identificano i possibili output ed eventualmente li si dividono in sottoinsiemi, per ognuno di essi si individuano le partizioni e da esse si ricavano i test-case. Il numero di partizionamenti è direttamente proporzionale al numero di test-case e di conseguenza all'efficacia del testing.
Esempio: La funzione f(a,b) dà come output 1 se il valore assoluto di a è maggiore di quello di b, 2 se è minore e 0 se è uguale. L'insieme degli input è ogni possibile coppia di numeri reali {(a,b) ∈ RxR}, quello degli output è {0, 1, 2}, output non sono divisibili in sottoinsiemi e quindi vi è una partizione per ogni valore:
Out1: {(a, b) : |a| > |b|}
Out2: {(a, b) : |a| < |b|}
Out0: {(a, b) : |a| = |b|}
Di seguito alcuni possibili test-case per:
Out1: (3,2), (-3,-2),(2,1),...
Out2: (2,3), (-2,-3),(1,2),...
Out0: (3,3), (-3,3),(1,1),...
Sempre in questo esempio, per ogni output sono possibili più partizioni:
Per Out1:

- {(a, b) : a>0 b>0 a>b}→(10, 1), (20, 10), (30, 5), …
- {(a, b) : a<0 b<0 a>b}→(-1, -10), (-10, -20), (-5, -30), …
- eccetera;

Per Out2: …
Per Out0: …

White Box Testing

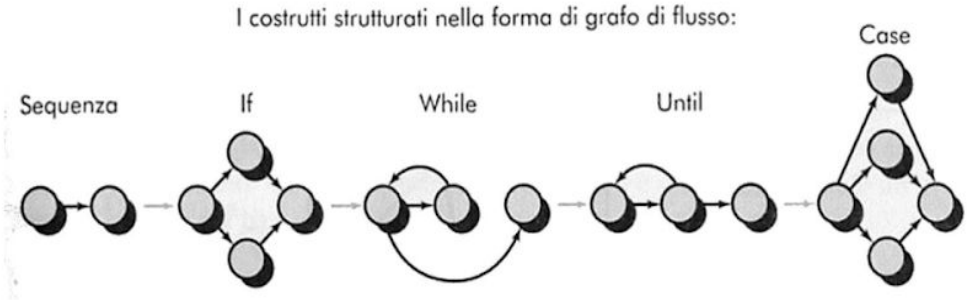
Nel White box testing la struttura del sistema e il codice sono conosciuti, quindi i test-case vengono ricavati da lì per testarne ogni parte, ogni istruzione deve essere eseguita almeno una volta e ogni condizione deve essere testata sia quando soddisfatta sia quando non lo è.
<slide 35>



Per fare ciò si utilizzano i grafi di flusso, un grafo in cui i nodi rappresentano le istruzioni e gli archi il flusso, esso può essere ricavato manualmente o con l'ausilio di strumenti CASE, un cammino è indipendente quando introduce un numero/sequenza di istruzioni/condizioni e/o se attraversa un arco non ancora percorso da altri, l'insieme di questi forma la base dei cammini. La cardinalità della base dei cammini è data dalla complessità ciclomatica (CC) del grafo di flusso.

I costrutti strutturali nel grafo si rappresentano nel seguente modo:
<slide 40>

Costruzione del flow-graph



La CC è il numero di cammini indipendenti nel grafo e può essere ricavata in tre modi:

- #archi + #nodi + 2;
- #regioni (aree delimitate da archi e nodi oppure l'area esterna del grafo)
- #nodiPredicato +1 (nodi in cui l'istruzione è una if)

CC indica il numero minimo di test-case da eseguire per poter testare ogni parte del codice.

Testing con integrazione incrementale

Il testing incrementale riguarda ogni componente del (sotto)sistema e permette di testare essi e la loro integrazione.

Testing dei componenti

Questo tipo di testing riguarda le soli componenti, ognuna di esse viene testata in modo isolato, può essere preceduto da un'ispezione dato che il codice non è solitamente lungo, per lo stesso motivo può essere white box e ha lo scopo di verificare la componente ma non la sua validazione (dal momento che si considera solamente essa) e viene di solito svolta dal programmatore che l'ha implementata.

Testing di integrazione

Questo tipo di testing viene svolto quando le componenti del (sotto)sistema vengono integrati, esse possono essere sviluppate ad-hoc, COTS oppure componenti riutilizzate, l'obiettivo è verificare che l'interfacciamento sia corretto controllando che ogni funzione sia invocata correttamente e che restituisca i risultati attesi. Quando una nuova componente viene aggiunta al (sotto)sistema, si esegue un test di regressione per evitare che si verifichino malfunzionamenti in quelli testati in precedenza, oltre che a nuove test-case.
Il testing di integrazione riguarda sempre la verifica e a volte anche la validazione se vi è un numero sufficiente di componenti, esso può essere white box o black box.

Release Testing

Il release testing riguarda release del sistema da consegnare al committente, è di tipo black box e l'obiettivo è la sua validazione, esso è differente in base al tipo di sistema:

- Nei sistemi customizzati, il committente controlla se i requisiti sono soddisfatti e si corregge ciò che è sbagliato finchè esso non è soddisfatto, nel momento in cui il committente è soddisfatto, il sistema viene consegnato;
- Nei sistemi generici invece vi sono due ulteriori fasi: l'alpha testing, in cui il sistema viene collaudato da sviluppatori tester o utenti esperti per poi correggere gli errori, e il beta testing, in cui una versione beta viene data a un gruppo di possibili utenti per collaudarlo ed eventualmente correggerlo, quest'ultima fase viene ripetuta finché non si ha la versione completa che viene messa sul mercato.

Stress testing

Lo stress testing server per verificare le performance e l'affidabilità del sistema, ovvero rispettivamente quando elabora carichi superiori a quello previsto e quando non fallisce, esso viene svolto quando è completamente integrato dato che queste proprietà sono complessive ed emergenti. Se il sistema supera lo stress test, dovrebbe in teoria resistere anche a carichi di lavoro normali, altrimente si possono individuare difetti non emersi nei test precedenti.

Back-to-back testing

Il back-to-back testing è utilizzato quando vi sono varie versione del sistema, solitamente prototipo, versioni per sistemi operativi/hardware diversi oppure nuove versioni con funzioni in comune con quelle precedenti. Per farlo si effettua lo stesso test su tutte le versioni e si confrontano gli output, se vi sono differenze, vi sono dei difetti in qualche versione, altrimenti dovrebbe essere corretto (oppure tutte hanno lo stesso errore).

Test di usabilità

Il test di usabilità valuta la facilità di utilizzo del sistema con diverse categorie di utenti che lo utilizzano la prima volta, da essi si raccolgono commenti, critiche e suggerimenti.

Manutenzione

La manutenzione è la fase in cui il sistema software viene mantenuto dopo la consegna, solitamente correggendo bug non emersi nel collaudo, migliorando i requisiti esistenti o aggiungendone di nuovi, riadattarlo alla piattaforma, eccetera. Questa fase è una sorta di evoluzione, il sistema viene infatti mantenuto aggiungendo nuove feature o correggendo i bug fino alla sua dismissione, questo gli permette di sopravvivere e quindi non perdere utilità e valore economico. La manutenzione può essere:

- correttiva: prevede la correzione di eventuali difetti, essi possono essere di implementazione, progettazione o specifica: i primi sono i più facili da correggere mentre gli ultimi sono invece più difficile dato che potrebbe essere necessario riprogettare il sistema, i secondi sono una via di mezzo perchè richiedono “solo” la modifica delle componenti del sistema;
- adattativa: il sistema viene adattato a cambiamenti di piattaforma (nuovo hardware o sistema operativo), i requisiti funzionali non vengono cambiati;
- migliorativa: miglioramento di vecchi requisiti (funzionali e non) o aggiunta di nuovi secondo le richieste del committente/mercato.

Processo di manutenzione

Il processo di manutenzione del sistema parte con l’identificazione dell’intervento: in questa fase si controlla se vi sono richieste per nuovi requisiti, correzione di errori, adattamenti, eccetera. In seguito vi è l’analisi, ovvero che in base alla richiesta si individuano requisiti, componenti, operazioni e test-case interessati dalla modifica, in base al tipo di manutenzione, potrebbe essere necessaria una revisione di una o più delle precedenti fasi.

Costo della manutenzione

Il costo della manutenzione è uguale o superiore al costo di sviluppo del sistema ed è influenzato da differenti fattori:

- La dipendenza dei componenti, la modifica di uno potrebbe ripercuteorsi sugli altri;

- I linguaggi di programmazione utilizzati, quelli ad alto livello sono più facili da capire e quindi da mantenere;
- La struttura del codice, la manutenzione è infatti più facile se il codice è ben strutturato e documentato;
- un collaudo ben approfondito riduce il numero di difetti successivi alla consegna;
- una documentazione di alta qualità facilita la comprensione del sistema da mantenere;
- La stabilità dello staff, i costi si riducono se i manutentori sono stati gli sviluppatori del sistema;
- L'età, il costo infatti tende a crescere con essa (specie se scritto con linguaggi superati);
- La stabilità della piattaforma, eventuali cambiamenti della piattaforma potrebbero richiedere manutenzione del software, anche se è raro che succeda.

Il tipo di manutenzione più costosa è quella migliorativa dato che bisogna "ripartire da zero", gli altri invece hanno costo minore.

Change Request Form (CRF)

Il CRF è un documento formale che descrive una modifica del sistema, esso è compilato dal proponente della modifica, inserendo la sua proposta e un valore di priorità, e dallo staff, il quale indica i componenti influenzati,la modalità di implementazione e la valutazione in termini di costi, tempi e benefici. Ultima ma non meno importante è l'approvazione/rifiuto della modifica da parte del project manager, dello staff e del committente.

Versioni e release

Un sistema software può essere distinto in versioni o release, le prime sono istanze del sistema stesso che differiscono di alcuni aspetti quali i requisiti, la correzioni degli errori o la piattaforma, le seconde invece sono particolari versioni che vengono distribuite al committente/client, di norma ci sono più versioni che release dato che possono essere create versioni intermedie durante lo sviluppo e la distribuzione può avvenire tramite un formato fisico oppure un download. Una release contiene il programma eseguibile e di installazione, la documentazione e i file di dati (necessari per il funzionamento) e di configurazione(indicano come il sistema deve essere configurato). Per identificare una versione, si utilizza un numero o gli attributi stessi, quest'ultima è più precisa ma ha il problema dell'unicità: gli attributi scelti devono poter rappresentare in modo univoco la versione (una possibile soluzione è l'utilizzo di entrambi i metodi).
Il rilascio di una release varia a seconda del tipo di sistema: nei sistemi customizzati avviene quando i bug segnalati sono corretti, vengono aggiunti nuovi requisiti oppure si cambia piattaforma, in quelli generici invece la frequenza di rilascio può avere effetti sul mercato: troppe release potrebbero scoraggiare i clienti ad acquistare l'ultima uscita mentre troppe poche potrebbe farli passare a prodotti più aggiornati.

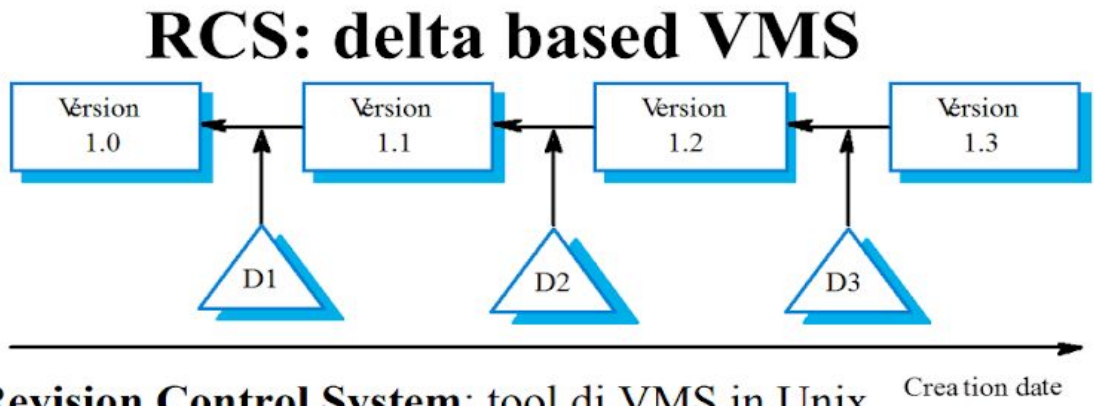
Version Management System

La gestione delle versioni avviene attraverso il Version Management System (VMS), uno strumento CASE che permette di creare una repository contenente documentazione e il codice di componenti e sistema, estrarne una versione (check-out) e salvarne una copia

nella directory di lavoro, al completamento del lavoro la nuova versione viene caricata nella repository (check-in). Il VMS, oltre a poter identificare versioni e release, permette di gestire tutte le modifiche non sovrascrivendole e differenziandole in base all'autore e/o rispetto alla versione master, in più ne permette la condivisione tra più sviluppatori e la tracciabilità dei contributi di questi ultimi. Un tool di VMS importante è il Revision Control System (RCS) in cui:

- l'ultima versione caricata diventa quella master;
- quella precedente viene sostituita con una delta, ovvero una specifica delle differenze rispetto a quella master;
- Riduce lo spazio occupato memorizzando solo la versione master più eventuali delta;
- Ogni versione è caratterizzata da una data, un numero e un autore e se ne può richiedere una in base a esse, applicando il delta corrispondente e usando file ASCII per il codice;
- i delta sono specificati come comandi di editing da applicare al master.

<slide 22>



- **Revision Control System**: tool di VMS in Unix
- La versione più recente del codice è quella master
- Quando una nuova versione viene creata, quella precedente viene cancellata e sostituita da un delta
- **Delta**: specifica delle differenze tra la versione corrente e quella precedente
- RCS memorizza una sola versione del codice e un insieme di delta, riducendo lo spazio occupato.

Configurazione software

Una configurazione software è un insieme di informazioni prodotte da un processo software, esse sono:

- La documentazione, contenente tutti i documenti e modelli creati durante il processo, i test-case, le informazione del committente, gli strumenti CASE utilizzati, eccetera;
- Il programma;
- le strutture dati utilizzate.

Ogni configurazione software di una release di un sistema è memorizzata in un apposito database, esso fornisce informazioni utili per la manutenzione e può essere integrato con degli strumenti CASE tra cui VMS.

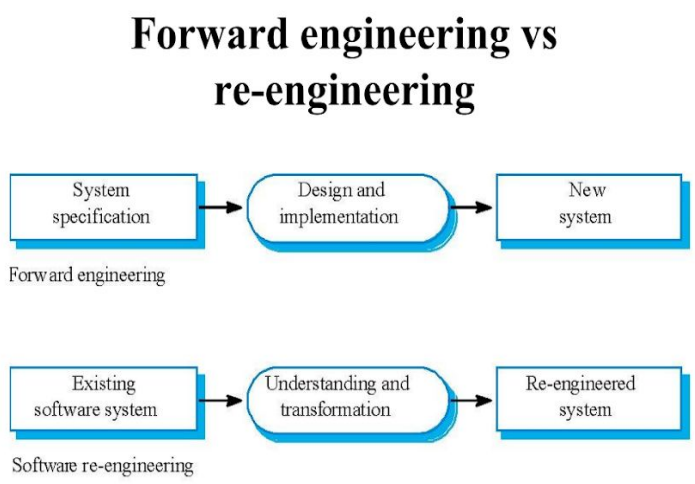
Sistemi ereditati

Un sistema si dice ereditato quando è vecchio e deve essere mantenuto nel tempo, esso presenta tecnologia obsoleta e sono costosi da mantenere, una loro dismissione potrebbe essere rischiosa perchè questi sistemi potrebbero essere critici, contengono informazioni che bisogna conservare oppure hanno servizi molto perfezionati. I problemi principali di questo tipo di sistema sono il fatto di essere poco strutturati, manca la documentazione, gli sviluppatori originali sono difficili da reperire e sono sviluppati su vecchie piattaforme.
Oltre alla normale manutenzione, altre strategie per gestire questi sistemi sono il cambio della propria organizzazione in modo da non renderlo più necessario e quindi dismetterlo, ristrutturarlo per renderlo più mantenibile oppure reimplementarlo, la scelta dipende dalla qualità del codice e dalla sua utilità; A seconda del grado di qualità e utilità, si possono essere quattro categorie di sistemi ereditati:

- Qualità/utilità bassa: hanno un alto costo di manutenzione e uno scarso ritorno economico, la dismissione in questo caso è la scelta migliore;
- Qualità/Utilità alta:bassi costi di manutenzione e buon ritorno economico, conviene quindi mantenere un sistema di questo tipo;
- Qualità bassa e Utilità alta: costo alto di manutenzione ma buon ritorno economico, in tal caso si può reimplementare o reingegnerizzare;
- Qualità alta e utilità bassa: basso costo di manutenzione e scarso ritorno economico, si può provare a mantenere oppure si dismette quando non ha più utilità oppure quando l’hardware si guasta.

Reingegnerizzazione

La reingegnerizzazione non è altro che la trasformazione di un vecchio sistema in uno nuovo con lo scopo di rinnovarlo o renderlo più mantenibile, i requisiti funzionali non cambiano e il comportamento non viene alterato, rispetto al normale processo software è meno probabile commettere errori e costa meno.



Il processo di reingegnerizzazione è diviso in tre fasi:

- Traduzione del codice: si traduce il codice sorgente del vecchio sistema in un altro linguaggio di programmazione non obsoleto oppure una versione più recente dello stesso, le ragioni di questa fase sono la mancanza di compilatori adeguati per i nuovi hardware, la mancanza di conoscenza del linguaggio e politiche organizzative che impongono un dato linguaggio come standard, essa può essere fatta attraverso strumenti CASE anche se sono comunque necessari interventi umani;
- Ristrutturazione del codice: si rende il codice più comprensibile raggruppando le parti correlate, eliminando le ridondanze, aggiungendo commenti, eccetera. Questa fase può avvenire manualmente oppure col supporto di tool, tuttavia è un processo costoso e quindi limitata solo alla parti in cui sono presenti fallimenti, molte modifiche,con del codice molto complesso, eccetera;
- Ristrutturazione dei dati: le strutture dati vengono cambiate e si migrano i dati in un DBMS più moderno.

Il costo della reingegnerizzazione è influenzato diversi fattori come la qualità del software e della documentazione , la quantità di codice e dati da ristrutturare e la disponibilità dello staff: questo processo può richiedere molto tempo e lavori e quindi si tende a impiegare personale solo per quello.
A supporto della reingegnerizzazione vi è l'ingegneria inversa, ovvero che si genera la documentazione a partire dal codice, per questo vi sono strumenti CASE adeguati.

Reimplementazione

La reimplementazione è un'alternativa alla reingegnerizzazione e si attua quando vi sono cambiamenti troppo radicali che non permettono di utilizzare quest'ultima, essa dismette il vecchio sistema e ne crea uno nuovo partendo dagli stessi requisiti, vi sono rischi e costi maggiori al prezzo di un sistema nuovo e più mantenibile.

Gestione del progetto

In parallelo al processo di ingegneria, vi è anche la gestione del progetto, esso si occupa della parte più burocratica/gestionale, ovvero l'assegnazione delle risorse umane, tecnologiche e finanziarie a ogni attività e il calcolo delle stime dei tempi, dei costi e dei rischi di ognuna, l'obiettivo è permettere la consegna al committente nei tempi previsti e cercare di contenere il costo finale sotto a quello previsto. In questa fase si incontrano tutti dello sviluppo di un software, può capitare infatti che vi siano risorse o tempo limitato avendo come limite a livello finanziario il costo deciso dal committente, un progetto infatti fallisce quando il prodotto viene consegnato in ritardo, costa di più di quanto stimato e/o non soddisfa i requisiti. La persona che si occupa di questa fase è il project manager, esso ha il compito di negoziare tempi e costi col committente e quello di pianificare e supervisionare il progetto stimando i costi e i tempi di sviluppo e assegnando le risorse, si può quindi considerare come l'anello mancante tra committente e sviluppatori.

Qualità del manager

La qualità del manager è la capacità di incoraggiare gli sviluppatori a dare il meglio, l'assumersi ogni responsabilità, la capacità di trasferire l'esperienza di progetti passati in quelli correnti e, cosa più importante, riuscire a vedere un possibile prodotto a partire da un'idea. Una cosa importante della qualità del manager è il lavoro di squadra distribuendo adeguatamente gli incarichi, garantendo la parità di trattamento, fiducia reciproca, eccetera, una squadra è infatti molto più produttiva di un normale gruppo e il risultato finale è la somma dei singoli contributi.

Attività di gestione

Nella gestione del progetto vi sono varie fasi:

- Scrittura delle proposte: permette l'ottenimento di un contratto o di vincere una gara d'appalto, in essa vi è una valutazione preliminare dei costi, dei tempi e degli obiettivi;
- Stima del costo;
- Pianificazione: si scompone il processo software in task di tipo milestone o derivabile, si calcola la tempistica e si assegnano le risorse a ognuna, in quest'ultima sottofase può essere necessario richiedere nuovo personale o acquistando nuovo hardware/software;
- Gestione dei rischi;
- Monitoraggio: si valutano i progressi e si confrontano con la pianificazione, se vi sono problematiche si esegue una revisione della pianificazione e sui costi e si discute col committendo di ciò.
- Presentazioni e scrittura rapporti: si informa il committente sull'avanzamento del progetto.
-

Difficoltà di gestione

Sfortunatamente una buona gestione di un progetto non ne garantisce il successo, una cattiva gestione però di solito lo fa fallire, le possibili difficoltà che possono venire fuori sono:

- Software intangibile: non si può vedere materialmente il progresso di un progetto, risolvibile con la produzione di documentazione ben fatta;
- Dal momento che ogni progetto è diverso dall'altro, non si può fare sempre riferimento all'esperienza;
- Incertezza: conseguenza previste o meno di decisioni non certe;
- Irreversibilità: tempo e risorse non recuperabili;
- Complessità: bisogna coordinare le attività distribuite in base alle risorse, ai tempi e ai costi.

Piano di progetto

Il piano di progetto è il documento che definisce l'attività di gestione del progetto, esso è formato da:

- un'introduzione contenente obiettivi e vincoli;
- l'organizzazione delle persone nei team e il loro ruolo;
- L'analisi dei rischi, dei possibili effetti e delle strategie da seguire quando necessario;
- Risorse hardware e software;
- Divisione del lavoro con tutti i possibili milestone e derivabili;
- Tempistica: dipendenze tra attività, tempi per raggiungere ogni milestone e allocazione delle risorse alle rispettive attività;
- Rapporti: tipi di rapporto prodotti durante il progetto.

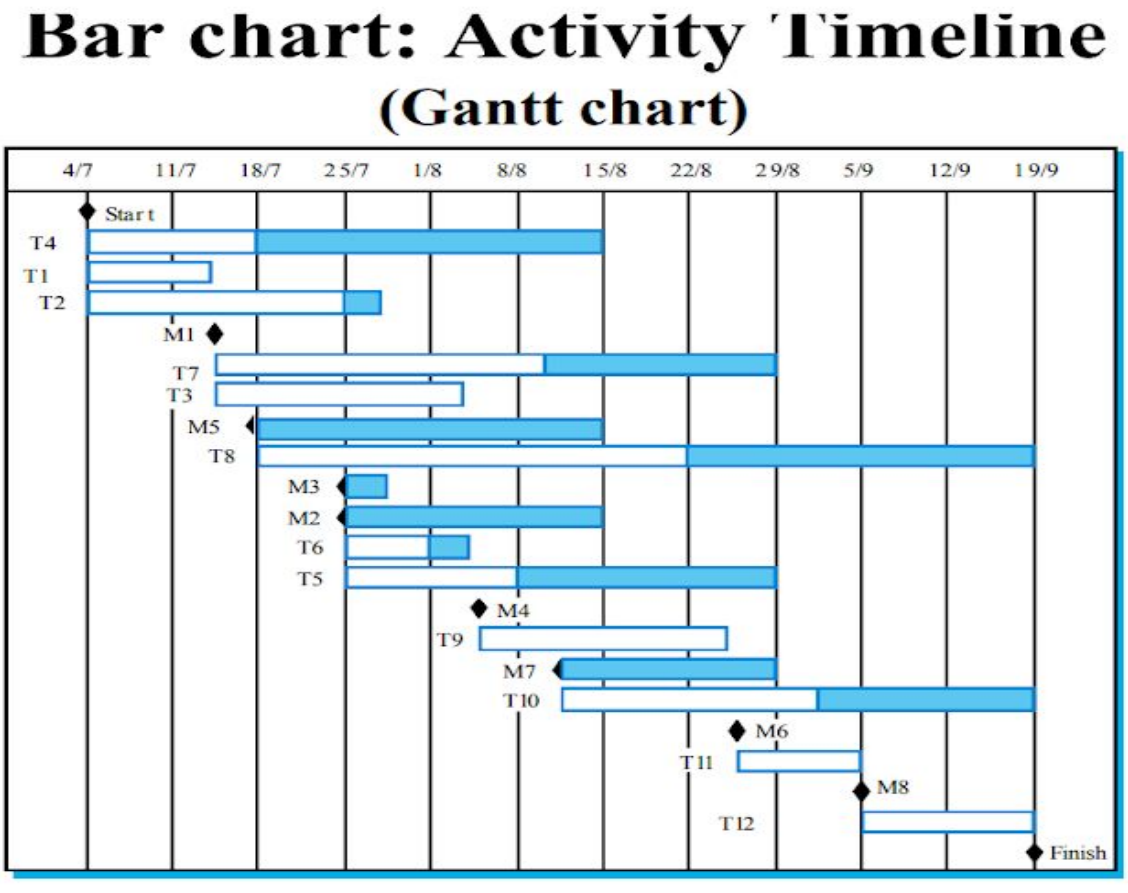
Pianificazione

La fase di pianificazione è divisa in diversa fasi, la prima è la scomposizione del progetto in task con conseguente identificazione delle dipendenze, un task infatti può iniziare solamente quando tutti quelli da cui esso dipende sono ultimati, vengono inoltre definiti milestone e derivabili. La tempistica è la stima dei tempi che indica la durata di ogni task e dell'intero progetto, la regola che si utilizza in pratica è il calcolo della stima come se tutto andasse bene più un 50 % per i problemi, rispettivamente il 30% per quelli previsti e un 20% per quelli che non lo sono. Infine vi è l'assegnazione delle risorse a ogni task. La pianificazione segue i seguenti principii:

- Ripartizione: un progetto deve essere suddiviso in task;
- Dipendenza: vengono definite le dipendenze tra task;
- Assegnazione del tempo: a ogni task è assegnato un tempo specificando la dati di inizio e fine;
- Responsabilità definite: ogni task deve essere assegnato a una persona/team;
- Risultati definiti: ogni task deve produrre un risultato;

- Punti di controllo: si definiscono momenti precisi in cui si verifica l'avanzamento del progetto.

La pianificazione si basa sulle informazioni, esse sono inizialmente scarse e in più bisogna prevedere problemi quali assenteismo, licenziamento o guasti, essa evolve all'aumentare delle informazioni stesse, rivedendo regolarmente i piani ed effettuando un processo ciclico durante il progetto. Come detto in precedenza, la pianificazione definisce i milestone e i derivabili, essi producono risultati intermedi per valutare l'avanzamento del progetto, i primi infatti indicano la terminazione di un task o un insieme di questi, producendo un rapporto sullo stato del progetto, i secondi invece sono particolari milestone da consegnare al commitente.
Milestone troppo frequenti determinano uno spreco di tempo per la preparazione dei rapporti, al contrario invece non permettono di valutare l'avanzamento del progetto. Per quanto riguarda la tempistica, si imposta la dati di inizio del progetto, in questa fase si utilizzano gli activity network, dei grafi DAG in cui i nodi rappresentano i task/milestone e gli archi le dipendenze tra essi, ogni nodi è assegnato un numero corrispondente al tempo di completamento del task, la durata del progetto viene calcolato col cammino critico, esso è il cammino di task in cui la somma dei tempi è massima, tuttavia possibili ritardi su altri cammini possono creare un'altro cammino critico. Per indicare la durata di ogni task si utilizza il bar chart, un diagramma che permette di calcolare anche il ritardo massimo consentito per non modificare il cammino critico.
<slide 21>

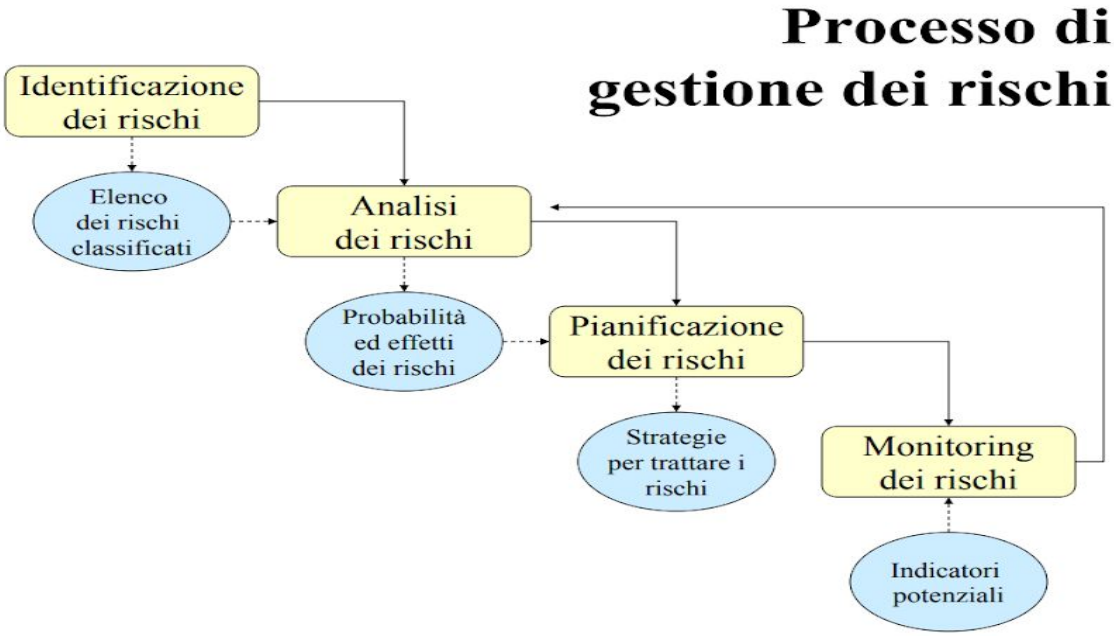


Rischi

I rischi sono circostanze avverse caratterizzate dall'incertezza, ovvero la probabilità con cui si verifica, e la perdita, le sue conseguenza che influenzano la tempistica e la qualità in negativo.
La gestione dei rischi è un processo il cui compito è quello di identificarli, valutarne l'impatto e da esso pensare a delle strategie per evitarli oppure per risolverli:

- Identificazione: si cercano i possibili rischi e la loro natura;
- Analisi: si determina la probabilità del rischio e la gravità delle conseguenze;
- Pianificazione: piani per evitare i rischi analizzati oppure per minimizzarli;
- Monitoring: si controlla che la probabilità e la gravità dei rischi non siano cambiate.

<slide 26>



I rischi possono essere:
in base alla causa:

- tecnologici: derivano dall'hardware e/o dal software utilizzati;
- riguardanti il personale: derivano dalle persone del team;
- organizzativi: derivano dall'organizzazione aziendale;
- strumentali: derivano dagli strumenti CASE utilizzati;
- dei requisiti: derivano dal cambiamento dei requisiti;
- di stima: derivano dalla valutazione delle caratteristiche del sistema.

In base all'effetto:

- di progetto: colpiscono tempistica e risorse;
- di prodotto: colpiscono la qualità del prodotto;
- di business: colpiscono il committente o lo sviluppatore sul piano economico.

Come già detto prima, l'analisi dei rischi è la stima della probabilità di un rischio e della gravità dei suoi effetti, la valutazione si basa sulla quantità di informazione sul progetto, sul processo, sul team, eccetera e sul giudizio e l'esperienza del manager. La probabilità assegnata a ogni rischio può essere molto bassa (minore del 10%) oppure (andando per gradi) molto alta (75% o superiore), i relativi effetti possono essere insignificanti, tollerabili, gravi oppure catastrofici.
Le strategie per limitare l'insorgere dei rischi possono essere:

- preventive: si cerca di ridurre la probabilità con cui essi avvengono;
- reattive: si cerca di ridurre il loro effetto nel caso avvengano;

Il monitoring non è altro che una verifica su ogni rischio identificato, controllando che la sua probabilità e la gravità dei suoi effetti non siano cambiati.

Modelli di processo

Un modello di processo è formato dalle cinque fasi viste in precedenza ed è specificato dai seguenti attributi:

- Visibilità: capacità di stabilire lo stato di avanzamento del processo, individuando la fase corrente e quante ne mancano alla fine;
- Affidabilità: gli errori devono essere scoperti durante il processo e non durante l'utilizzo del prodotto;
- Robustezza: il processo deve essere in grado di continuare anche in presenza di problemi, in particolare al cambiamento dei requisiti;
- Rapidità: durata del processo a partire dalla specifica, indica il tempo necessario per la consegna.

Purtroppo non è possibile ottimizzare tutti gli attributi, cose come rapidità e visibilità sono in contrasto dato che la produzione di documenti con i risultati di ogni fase rallenta il processo, essi dipendono molto dal modello utilizzato. Un modello di processo è il tipo di organizzazione delle fasi utilizzato, esso può essere a cascata, iterativo, a spirale, eccetera.

Modello a cascata

Nel modello a cacata le fasi sono distinte e sequenziali, prima di passare alla successiva bisogna prima completare quella corrente, esso è caratterizzato dall'essere strettamente vicino all'ingegneria tradizionale, richiede un definizione dei requisiti già dettagliata all'inizio (onde evitare errori per incompletezza o mal interpretazione), ogni fase produce un documento coi risultati ottenuti e il committente può controllare il prodotto solamente dopo il collaudo. Nonostante vi sia una buona visibilità, questo modello presenta difficoltà nell'effettuare cambiamenti in corsa ai requisiti, si cerca quindi di stabilizzarli il più possibile.
<slide 7>
Il modello a cascata è un processo black-box: il committente è solamente coinvolto nella specifica e può visionare il prodotto dopo il collaudo.
<slide 10>
Le fasi del modello a cascata hanno lunga durata, questo per ottenere una documentazione più completa possibile e di conseguenza evitare errori, alcuni componenti del team potrebbero attendere la fine dell'attività di un collega, andando così in stato bloccante.

Nella realtà le fasi non sono sempre sequenziali, infatti se durante una fase si presentano problemi che coinvolgono quella precedente, si torna lì a correggerli. In più non è sempre possibile eseguire una specifica completa e dettagliata già dall'inizio dato che i committenti potrebbero cambiare idea nel corso del tempo e/o esprimersi male. Per andare incontro a questi problemi si utilizzano i prototipi, essi sono versioni preliminari del sistema che realizzano un sottoinsieme dei requisiti. un'esempio di prototipo è il modello usa e getta, il suo scopo è quello di comprendere un'insieme di requisiti funzionali poco chiari per poi mostrarli al committente per una valutazione, esso non evolverà nel sistema finale ma verrà gettato via.
<slide 14>
Il prototipo usa e getta è anche pensato per il training degli utenti, la sperimentazione di soluzioni software e il back-to-back testing, ovvero un tipo di testing in cui si eseguono gli stessi test sul sistema e su un prototipo alternativo, essi devono dare gli stessi risultati.

Modelli iterativi

I modelli iterativi prevedono la ripetizione di alcune fasi del processo, il sistema è un prototipo evolutivo, ovvero che il prototipo non viene gettato ma evolve nel sistema finale, ogni fase infatti produce una nuova istanza del prototipo e periodicamente ne viene rilasciata una versione per farla valutare dal committente. I modelli iterativi sono processi white box: il committente partecipa a ogni fase del processo.
<slide 18>
Rispetto al modello a cascata, il costo delle modifiche nei modelli iterativi è logaritmico e non esponenziale in relazione alla quantità di tempo passata dall'inizio del progetto.

Sviluppo evolutivo

Nello sviluppo evolutivo, si genera un primo prototipo avente un sottoinsieme dei requisiti, di essi si eseguono tutte le fasi fino al collaudo e, in presenza di problemi, è possibile ritornare a quella precedente per risolverli. Infine si genera una versione/release del sistema che verrà valutata dal committente e, in base al suo feedback, si aggiorna il sistema. In seguito si generano altri prototipi da altri sottoinsiemi di requisiti procedendo all stesso modo, si continua così finchè non vi sono più requisiti da gestire.
<slide 20>
Il processo dello sviluppo evolutivo incomincia dai requisiti funzionali più chiari e con maggior priorità, quelli on funzionali vengono invece trattati in seguito, il committente partecipa valutando il sistema indicando quali requisiti sono soddisfatti o meno, chiarendo quelli già esistenti e introducendone di nuovi. I vantaggi di questo sviluppo vengono dati dal prototipaggio: specifica e progettazione non vengono infatti definite in modo completo già dall'inizio del processo, il cambiamento dei requisiti è più semplice, grazie ai controlli è possibile sapere se vi sono equivoci con il committente e/o se vi sono requisiti incompleti, inconsistenti o mancanti e vi è la disponibilità anticipata di un sistema funzionante. Al contrario, il processo è poco visibile dato che la documentazione non è sempre aggionata e non si sa quante versioni mancano prima della fine, i prototipi devono essere generati rapidamente per non far aspettare troppo il committente dato che deve fornire un feedback (e non sempre è disponibile), è un modello costoso e rischia di essere poco strutturato a causa dei cambiamenti continui all'implementazione.

Sviluppo incrementale

Lo sviluppo incrementale mantiene i vantaggi di quello evolutivo permettendo però una maggior visibilità, esa consiste in una fase iniziale in cui si fa una specifica ad alto livello dei requisiti, si indicano il numero di versioni da fare durante lo sviluppo e si progetta il sistema, le restanti fasi sono cicliche, ovvero che si specificano i requisiti sempre più nel dettaglio, implementando, collaudando e generando a ogni iterazione una release da mostrare al committente (se necessario, è possibile ripetere la fase per una stessa versione).
<slide 27>
Questo tipo di sviluppo è meno soggetto all'incertezza dato che l'architettura e il numero di versioni sono stabiliti fin dall'inizio.

Sviluppo rapido

A causa della concorrenza, tutt'oggi il software deve essere consegnato rapidamente, seguendo questo punto di vista si preferisce quindi lo sviluppo evolutivo/incrementale a causa della lentezza di quello a cascata. Nei team ristretti e con progetti limitati un modello di gesto tipi potrebbe generare overhaed dato che sono troppo vincolanti e si porta via tempo utile all'implementazione. Proprio da questa filosofia è nato l'approccio agile, esso è adatto ai piccoli team, è pronto al cambiamento dei requisiti ed è flessibile: il modello di processo si adatta alle esigenze delle persone. Lo scopo è la consegna rapida del prodotto concentrandosi sulla fase di implementazione, tutte le altre vengono eseguite ad alto livello.

Extreme programming (XP)

XP è una variante dello sviluppo evolutivo che segue l'approccio agile, infatti la rapidità è l'obiettivo fondamentale, rendendo in questo modo i tempo di consegna ristretti e rendendo l'implementazione la fase principale del processo. In questo modello vi sono numerose release del sistema, vi è il cambiamento dei requisiti e il committente è particolarmente coinvolto.
<slide 35>
Il processo parte con una specifica ad alto livello dei requisiti, essi sono rappresentati sottoforma di scena non strutturati e ordinati per priorità, ognuno di essi viene poi suddiviso in incrementi. La seconda fase è la pianificazione, in essa si stabiliscono le versioni da realizzare, i tempi di consegna e i test della release successiva insieme al committente. La progettazione semplice definisce o aggiorna l'architettura ad alto livello lasciando i dettagli all'implementazione. Nell'implementazione si utilizza la programmazione a coppie: ogni linea del codice è scritta e visionata da due persone, aumentando le probabilità di rilevare errori, in questo modo si ha una condivisione della conoscenza del codice dato che le coppie cambiano dinamicamente a seconda della versione e, quando uno sviluppatore cambia compagno, può spiegargli in breve cosa fa il codice. Finita l'implementazione si passa a uno sviluppo dei test iniziale e al possesso collettivo, in quest'ultima fase un programmatore può modificare ogni parte del codice. La fase di collaudo presenta tre tipi di testing:

- testing sulla versione: viene testata la corrente versione del sistema;
- testing di integrazione: tutti i test vengono eseguiti su ogni versione;
- testing di accettazione; il committente controlla se i requisiti sono soddisfatti.

Gli strumenti CASE permettono di ridurre i tempi di collaudo, aumentando la rapidità.

In XP il committente è molto più coinvolto rispetto ai modelli precedenti, egli infatti assegna le priorità ai requisiti, decide il contenuto delle release e le relative scadenze, partecipa alla definizione dei requisiti e dei test-case (e alla loro esecuzione) e si trova nella stessa sede degli sviluppatori. Per un modello di questo tipo è opportuno avere un team di poche persone e il sistema deve essere di dimensioni medio-piccole, i programmatori possono lavorare a coppie su ogni parte di esso. Il committente è on-site, ovvero che è presente a tempo pieno nel team di sviluppo. La rapidità è ottenuta eseguendo specifica e progettazione ad alto livello, dagli strumenti CASE per il collaudo e dalle release frequenti (ognuna realizza poche versione e quindi richiede poco tempo). La documentazione non è in forma classica ma è invece rappresentata dagli scenari, dal codice e dai test-case. La qualità è invece mantenuta dallo sviluppo iniziale dei test, dalla programmazione a coppie e sempre dal numero frequente di release dal momento che vi sono tanti collaudi su piccole versione (è più facile trovare un difetto), in più il coinvolgimento del committente ne aumenta i benefici. La fase di refactioring è la modifica del codice per includere aggiornamenti, effettuare test, mantenerlo pulito oppure per il testing di regressione, esso non modifica in alcun modo il comportamento di un sistema ma ne migliora la struttura interna.

Processo unificato

Il processo unificato è un modello creato dagli stessi autori di UML ed è organizzato in quattro fasi:

- L'avvio permette la definizione dei requisiti e dei rischi prelimiari, permette lo studio di fattibilità degli stessi, eccetera;
- L'elaborazione genera un primo prototipo di tipo evolutivo;
- La costruzione permette il passaggio dal prototipo al sistema finale;
- La transizione prepara tutto il necessario per la consegna, dall'adattamento hardware/software al manuale, eccetera.

Ogni fase del progetto richiede l'esecuzione di workflow, essi sono gli stessi per tutte le fasi, essi sono la definizione e l'analisi dei requisiti, la progettazione, l'implementazione e il testing. Ogni fase può essere eseguita più volte e produce una milestone e un prototipo eseguibile. Questo modello di processo è sequenziale dato che le fasi vengono eseguite in sequenza, iterativo perchè ogni fase può essere fifatta ripetendo così gli stessi workflow ed è incrementale dato che viene prodotto un prototipo evolutivo a ogni fase. UML nel processo unificato è utilizzato all'avvio per la deduzione dei requisiti, all'elaborazione per l'analisi degli stessi e la strutturazione del sistema e per modellare il comportamento nella fase di costruzione.

Modello a spirale

Nel modello a spirale le attività sono distribuite in vari settori del piano, il processo avviene attraversando varie volte i settori, formando appunto una spirale, una prima versione prevede il rilascio del sistema alla fine del processo mentre una seconda ne permette lo sviluppo tramite prototipi. Questo modello include anche le attività di gestione e ogni loop rappresenta un task, esso è diviso in quattro settori:

- Determinazione di obiettivi e alternative: si pongono obiettivi specifici e delle soluzioni per il task corrente;

- Valutazione di alternative e rischi: vi è la gestione dei rischi e la costruzioni di prototipi per valutare le soluzioni;
- Sviluppo e convalida: si sviluppano modelli e test per valutare i prototipo, si sceglie la soluzione in base a essa e si sviluppa e verifica l'attività a seconda della soluzione scelta;
- Pianificazione: Revisione del progetto più la pianificazione della fase successiva.

<slide 69> (if you know what i mean)
Il modello a spirale è un processo ciclico in cui vi è una release a ogni loop, i settori di questi ultimi rappresentano le fasi (ben distinte) e include le attività di gestione.

Sviluppo basato sul riutilizzo

Un modello chiamato CBSE si basa sul riutilizzo dei componenti già esistenti, essi infatti vengono riciclati inserendoli nel sistema corrente, essi possono essere di vecchi progetti oppure commerciali, questo approccio è sempre più frequente a causa dell'affermarsi degli standard.
<slide 75>
Questo modello è formato dalle seguenti fasi:

- Specifica;
- Analisi dei componenti: si valutano le possibili componenti da riutilizzare;
- Adattamento dei requisiti: i requisiti vengono adattati alle componenti scelte;
- Progettazione con riuso: viene definita l'architettura del sistema composta sia da componenti vecchie sia da quelle ad-hoc;
- Implementazione: codifica dei componenti ad-hoc e integrazione di quelli esistenti;
- Collaudo;

Grazie al riutilizzo dei componenti, questo modello permette consegne più veloci e una riduzione dei costi e dei rischi, tuttavia si devono trovare compromessi per quanto riguarda i requisiti, i componenti infatti potrebbero non essere perfettamente adatti a soddisfarli.

Design Pattern

Un altro metodo per velocizzare il processo è l'utilizzo dei design pattern, esse sono soluzioni generiche adattabili a ogni tipo di problema. I design pattern possono essere:

- Creazionali: riguardano la creazione di oggetti;
- Comportamentali: riguardano l'interazione tra classi e oggetti e le relative responsabilità;
- Strutturali: riguardano la struttura di classi e oggetti;

Un esempio di design pattern comportamentale è Observer, esso realizza delle relazioni 1-N tra oggetti osservabili e oggetti osservatori, nel caso in cui i primi cambiano il proprio stato, tutti quelli appartenenti alla seconda categoria vengono notificati e di conseguenza aggiornati. Il funzionamento avviene attraverso delle specifiche operazioni:

- Subscribe: un'oggetto chiede a un subject di diventare osservatore;
- Registered: un'oggetto diventa osservatore;
- Update: un'oggetto cambia il proprio stato e lo comunica all'osservatore;
- remove: un'osservatore chiede a un subject di togliersi;
- removed: l'oggetto non è più osservatore;