**INGEGNERIA DEL SOFTWERE**

1. **INTRODUZIONE:**

Affronta il problema di “costruire” un softwere, è fondamentale sapere cosa fare per progettare un softwere, perché ormai i softwere sono utilizzati nella vita di tutti i giorni, e inoltre chi sa progettare un softwere è molto ricercato nel mercato del lavoro. Da notare come ultimaamente si spenda molto di più nella creazione Sw che Hw. L’ingegneria del Sw studia metodologie e strumenti per creare e supportare Sw professionali (di tipo industriale). Un programma per essere professionale non deve solo funzionare correttamente ma anche essere scalabile e con un uso di risorse che sia ragionevole per il problema assegnato.

Scalabile: il softwere che viene progettato in riferimento a un certo insieme di oggetti deve poter essere esteso con facilità ad altri gruppi di oggetti anche più numerosi.

Esistono due macrocategorie in cui si divisono i Sw:

1. Prodotti Sw Generici: es. Sistemi Operativi, Office, videogiochi prodotti cioè di uso generico per più utenti non specifici, si fa un prdotto per una categoria vasta di persone che lo useranno come vogliono, ma è il programmatore che decide come crearlo.
2. Prodotti Sw specifici/personalizzati: es. gestire le condizioni dell’aria a verona, o gestire uno specifico reparto dell’ospedale prodotti cioè creati appostamente per una stretta cerchia di utenti e usati solo in quell’ambito. Cosa farà il softwere è deciso dal richiedente in tutti i suoi aspetti.

Note: da notare come le due categorie sempre più si intersecano tra loro.

1. **FAQ (Frequently Asked Question):**
2. Cos’è un Sw?

Un Sw è un insieme di programmi che svolgono un compito con annessa documentazione

1. Quali sono le caratteristiche di un buon Sw?
2. Mantenibilità: il softwere deve poter essere cambiato dal suo uso specifico, deve gestire errori anche imprevisti e deve adattarsi alle macchine su cui gira.
3. Affidabilità: il Sw deve fare quello per cui è stato progettato senza ledere la sicurezza o la privacy
4. Usabilità: è lo studio della semplicità di utilizzo, rendere facile l’uso di un Sw in modo corretto, in pratica devo mantenere le condizioni d’ uso del mio Sw il più possibile simili a quelli degli altri Sw già diffusi in modo da creare minore diffcoltà di utilizzo per chi si approccia al mio Sw per la prima volta (es. browser web).
5. Quali sono le principali attività dell’ingegneria del Sw?
6. Specifica del Sw: considerare i requisiti che ho (in qualsiasi linguaggio mi vengano proposti) specificare quindi cosa dovrà fare il mio programma e cosa l’utente si aspetta che faccia. Di questa area fa parte l’ ingegneria dei requisti.

Ingegneria dei requisti: considera i requisti degli utenti e li trasforma in altri requisti più formali, ottenendo così una descrizone scritta in linguaggio formale e non ambigua.

1. Realizzazione/ Sviluppo del Sw: Si appoggia sulle specifiche ottenute dall’ ingegneria dei requisiti; si divide a sua volta in due sottoparti:

* Progettazione: riguarda più aspetti (es. come il Sw si approccia all’esterno) e noi in particolare la useremo più orientata agli oggetti con Uml.
* Implementazione: è la parte pratica di scrittura del codice.

1. Validazione del Sw: consiste nella validazione di tutti le componenti in singolo del Sw, e successivamente del Sw stesso nella sua interezza, nella fasi finali spesso la validazione richiede l’uso di utenti esterni(Beta- tester).
2. Evoluzione del Sw: è una parte molto complicata ma fondamentale adattare il Sw a quanto di nuovo arriva sul mercato, uno dei principali problemi in questo campo è il fatto che l’ uomo è restio al cambiamento di natura e quindi tende sempre a tirare indietro.

Note: a seconda del Sw che voglio comunque non cambiano le fasi di produzione, ma possono cambiare le tecniche e la loro attuazione, e gli strumenti (i quali possono cambiare anche a causa del background del programmatore)

1. Differenza tra ingegneria del Sw e Informatica:

l’ ingegneria del Sw è l’applicazione delle teorie informatiche in ambito pratico.

1. Quali sono le attuali “sfide” dell’ ingegneria del Sw?
2. Sicurezza
3. Costi di produzione
4. Eterogeneità (le mie app devono girare ovunque e gestire diverse richieste)
5. Tempi (ultimamente i tempo sono spesso molto stretti e devono essere rispettati)7
6. Rispetto alle principali attività dell’ ingegneria del Sw dove sta la maggior parte dei costi?

Solitamente l’ evoluzione del Sw è il costo maggiore (vita media di un Sw 5-10 anni). Nel caso in cui il Sw non debba durare a lungo allora il costo si sposta in Realizzazione e Validazione. (Validazione è fondamentale è sarà obbligatoria nell’ elaborato dell’ esame).

1. **EFFICIENZA VS EFFICACIA**

Efficienza: uso solo tempo e memoria indispensabili cioè evito gli sprechi.

Efficacia: realizzare a pieno le specifiche dell’utente; quanto il Sw riesce ad avere successo nel trovare il risultato che voleva l’utente.

1. **TIPI DI APPLICAZIONI:**
2. Sistemi Stand-Alone: eseguiti su pc e non richiedono connesione a internet (ultimamente stanno sparendo)
3. Applicazioni Interattive basate su transazioni: si basano su scambi di dati e sul web (es. amazon e ebay)
4. Sistemi Embedded: Sistemi Sw messi dentro varie apparecchiature (bancomat, controllo treni, lavatrice) il Sw non è in un pc ma in un oggetto con vincoli architetturali e spesso presenta forti requisti in termini di tempo di reazione.
5. Applicazioni Batch: devono elaborare dati in grosse quantità, non c’è interazione con l’ utente esegue un compito molto grande da solo e in una sola volta.
6. Sistemi di Intrattenimento: come i videogiochi, netflix, siti di scommesse, il tutto spesso basato sul web.
7. Applicazioni di raccolta Dati: applicazioni che raccolgono dati in un unico punto, come i dati del meteo o dell’inquinamento.
8. Sistemi Di Sistemi: I Sw prevedono l’interazione tra vari sistemi, come amazon che se vuoi comprare da un sito presente su amzono ci pensa lu ad approcciarsi con tale sito.
9. **CARATTERISTICHE FONDAMENTALI DEL SVILUPPO DEL SW**

Metedologia e processo di sviluppo: deve esserci dall’inizio un percorso definito che si stacca dalla macchina in sé e dal programmatore stesso.

Metodologia: metodi e risorse che dovrò usare, sapere come “muoversi”

Processo: caratterizzare come ottengo tutti i requisti in vari passi

1. **SCOPO DEL CORSO**

Tra gli altri il più importante è dare le fasi fondamentali della metodologia per approccarsi ai problemi che mi verranno proposti; bisogna prestare particolare attenzione a:

* Affidabilità e prestazioni
* Specifiche e requisti, i quali devono essere compresi approfonditamente e in maniera non ambigua
* Riuso del Sw, sempre più si impone questa idea, cioè prendo piccoli Sw già fatti e cerco di assemblarli tra loro e usarli per ottenere il Sw che mi serve, da notare che per il riuso a volte i passaggi sopracitati non valgono più in quanto dovrò cercare di adattare i requisti al riuso di Sw invece di attare il Sw ai requisti.

Nota: Stakeholders, sono coloro che in qualsiasi modo si approcciano al mio sistema.

1. **RESPONSABILITÀ PROFESSINALI**

Che responsabilità ha il programmatore rispetto al suo Sw? Intanto ogni Sw è responsabilità del programmatore solo finchè viene usato nei modi da lui descritti come “corretti”. Inoltre prima di distribuire un Sw devo applicare tutte le tecniche di validazione e dimostrare così che è sicuro, è un aspetto fondamentale ma spesso sottovalutato. Chi decide in campo di Sw sono IEEE e ACM.

Aspetti Principali:

1. Confidenzialità
2. Competenza: garantire la massima competenza e la capacità di aggiornarsi, non promettere ciò che non si è in grado di fare.
3. Onestà etica: rispetto a quello che si sta facendo.
4. Proprietà intelletuale del Sw
5. Uso scorretto del Sw: non posso usare le mie competenze per usare in modo scorretto il Sw.
6. Giudizio e rapporto con i colleghi: ci deve essere collaborazione.
7. **DILEMMA ETICO**

* Disaccordo con le politiche aziendali: propongo soluzioni alternative.
* Dipendenti non si comportano in modo responsabile: riprendo il dipendente e lo controllo (controllo il prodotto).
* Sviluppo di strumenti potenzialmente pericolosi.
* Esempi 1-2-3 libro -> Sistema di infuzione di insulina, Sistema di cartelle clinche psichiatriche, Stazioni metereologiche in zone disabitate.

1. **PROCESSI SW (Intro):**

Il processo Sw è un insieme strutturato di attività richeste per lo sviluppo di un sistema Sw; il processo in sé è un insieme strutturato di attività (azioni, operazioni, divisio in 4 fasi che sono quelle considerate le principali attività dell’ingegneria del Sw).

Modelli di processo:

* Schemi / Modi di realizzare un Sw.
* Ogni modello si basa sulle 4 fasi già viste, ma ogni fase sarà specificata.

Nella progettazione di un Sw vanno considerate alcune cose:

* Pre / Post Condizioni: se applicare le condizioni prima di inziare o a prodotto finito
* Prodotti
* Ruoli: copetenze, compiti da svolgere
* Specifiche

1. **PROCESSI SW (App):**

Insieme di attività che opportunamente coordinate portano alla creazione di un buon Sw.

1. **APPROCCI:**
   1. Plan – Driven: tutti i processi sono specificati già dall’ inizio, verifico la progressione dell’ esecuzione delle attività.
   2. Agile: non c’ è la pianificazione iniziale delle attività, la pianificazione si ottiene durante lo svolgersi delle attività e può essere modificata.
2. **MODALITÀ DI SVILUPPO:**
   1. Sviluppo a cascata: plan-driven, approccio rigido distinguo diverse fasi che si susseguono per ogni fase uso il tempo che mi serve. L’ utente vede il prodotto finito.
      * + Definizione dei requisiti
        + Progettazione Sw e del sistema
        + Implementazione e test su Unità
        + Integrazione e validazione sistema
        + Operatività e gestione

Ogni fase fonrisce l’ input a quella successiva, dall’ultima si può anche tornare indietro.

* 1. Sviluppo incrementale: prevedo consegne intermedie, l’utente vede i primi semi-lavorati e può dare ulteriori specifiche; approccio agile o anche plan-driven.

Specifiche -> Sviluppo -> Validazione

Queste tre fasi sono comuni a tutte le modalità di sviluppo, però nell’ incrementale c’ è la possibilità di tornare indietro, costruisco pezzo per pezzo non tutto assieme. Ciò mi porta quindi a ottenere:

* Versione iniziale: di documentazione e del Sw
* Versione intermedia: di documentazione e del Sw (posso essere molteplici)
* Versione finale: di documentazione e del Sw

Pro e Contro lo Sviluppo Incrementale:

* + - * Pro: posso avere una gestione dinamica dei requisti, posso anche presentare un semi-lavorato per farmi spiegare meglio le specifiche, inoltre il committente si sente più coinvolto nel progetto, e posso avere una consegna più rapida in quanto alcune parti funzionano già prima che io concluda tutto il lavoro.
      * Contro: il processo di produzione non è visibile come nell’ approccio a cascata, inoltre non riesco a produrre una documentazione esplicita di tutti i passaggi. Da un punto di vista qualitativo spesso il Sw prodotto è inferiore a quello di un processo unico in quanto non riesco a ottimizzare al massimo facendo tutto in tempi separati.
  1. Sviluppo basato su riuso: nello sviluppo basato su riuso si aggiungono alcune fasi di produzione a quella sopracitate
     + - Analisi delle componenti Sw che voglio riusare
       - Modifica dei requisiti
       - Progettazione del Sw con riuso
       - Sviluppo del Sw con riuso
       - Integrazione con parti già esistenti

Da questi passaggi aggiuntivi ottengo un nuovo processo di produzione che sarà definito nle modo seguente:

1. Specifica dei requisti
2. Analisi delle componenti
3. Modifica dei requisti
4. Progettazione del Sw con riuso
5. Sviluppo/Integrazione
6. Validazione (sia componente per componente che totale )

Nota: Spesso questi approcci si intersecano tra loro.

1. **SPECIFICA DEL SW (requisti):**

Si divide principalmente in 4 fasi:

1. Studio di fattibilità: quello che mi viene richiesto che costi e risorse richiederà? È fattibile?
2. Elicitazione e analisi dei requisti: per capire cosa vuole realmente l’utente mi faccio dare i documenti principali della sua azienda per capire bene che cosa fanno e cosa in realtà gli serve ma non mi sta dicendo, in seguito è necessario anche intervistare il committente per tirargli fuori più informazioni possibile e soprattutto devo cercare di rendere la cosa il meno ambigua possibile.
3. Specifica dei requsiti: cerco di rendere tutti i requisti più formali usando appunto dei formalismi anche se magari non del tutto rigorosi
4. Validazione dei requisti: Controllo se il senso dei requisti dopo tutti i miei passaggi è rimasto lo stesso per capire come ho lavorato o se devo rifare quanto sopracitato.

Validazione dei requisiti

Specifica dei requisti

Studio di Fattibilità

Elicitazione e analisi dei requisti

Genero una documentazione per i requsiti

Documento di Validazione

Tratto i reqsuiti con l’utente

Rapporto di fattibilità

1. **PROGETTAZIONE E IMPLEMENTAZIONE DEL SW:**

Parto dal documento prodotto dalla fase precedente e devo eseguire

* + Progettazione del Sw: è divisa in varie parti perché non bisogna solo progettare il Sw ma anche come essa si approccerà con l’ esterno e in quanti sottoproblemi voglio dividerlo, o decidere quale sarà la parte principale. Il prodotto di questa fase è la documentazione della progettazione che di solito è scritta.
  + Implementazione dei Sw: in base al documento prodotto dalla fase precedente prima implemento tutte le parti e poi le unisco.

**12.1) PROGETTAZIONE DEL SW**

Per progettare il Sw servono:

* + informazioni sul sistema su cui deve girare
  + descrizione dei dati
  + specifiche dei requisti

Come ottenere ciò deriva dal corso di basi di dati

Progettazione della base di dati

Design pattern, cioè come farò i miei moduli

Progettazione dei componenti

Progettazione dell’ interfaccia

Decido come i moduli che creo si parlano tra loro. Interfaccia intesa nel senso di interfaccia di comunicazione tra moduli

Decido come scomporre il Sw cioè quanti moduli fare

Progettazione dell’ architettura

Nota: Una parte ormai fondamentale al giorno d’ oggi in un Sw è l’interfaccia grafica, non ci soffermiamo sull’ implementazione in quanto è la resa pratica di quanto ottenuto da questa fase.

1. **VALDIAZIONE DEL SW**

V e v = Verifica e validazione.

Verifica = controllo che il sw sia scritto bene e non sia ridondante .

Validazione = verifico che funzionino tutte le componenti del Sw e l’ unione di esse.

Solo alcune case produttrici lo fanno, faccio uscire una versione del Sw non definitiva ma quasi che può essere provato da qualsiasi utente che dovrà riportare la possibile presenza di bug

Beta-test

Fatto da uno o più utenti che effettuano una validazione con dati reali e derivanti dal loro background, chiamato anche alfa-test

Test di acettazione

Verifico che i componenti comunichino in modo corretto. Sempre fatto dai programmatori usando dati reali e il loro background

Mentre programmo controllo che funzionino le parti

Test del sistema

Test dei componenti

Nota: il test di acettazione gli altri vanno prima progettati.

1. **EVOLUZIONE / MANTENIMENTO DEL SW**

Evoluzione e sviluppo del Sw sono sempre più vicini e avvengono di continuo (vedi le interfacce dei browser web che cambiano continuamente senza una utilità reale)

* Tutti i cambiamenti richiedono un nuovo sistema?

Generalmente no, importante è cercare di prevedere alcuni cambiamenti, lasciare alcune parti non del tutto complete ma in parte configurabili di modo da potersi adattare se alcune cose dovessero cambiare.

* Come gestisco il cambiamento?

Devo gestire il costo di rifare una parte del sw, e va anche rifatta l’ analisi dei requisti.

Per ridurre i costi si può:

1. Evitare il cambiamento: durante il processo sw includo attività che possono anticipare il cambiamento, nella progettazione tengo presente che potrà esserci un cambiamento.
2. Tollero il cambiamento: progetto con un approccio incrementale in modo che se mi arriva un cambiamento in corso d’opera mi è più facile gestirlo, si usa anche la

prototipazione -> produzione di un prototipo che realizzi in parte le richieste dell’utente senza badare a tempo e memoria, per dare un idea al committente di cosa potrebbe fare. (viene usato il prototipo anche nell’ elicitazione dei requsiti).

1. **GESTIONE DEL CAMBIAMENTO: Prevenzione -> Prototipo**

Prototipo: versione inziale del sistema che sviluppa solo le idee più importanti e più delicate, inoltre permette anche di tenere “l’utente” più vicino al progetto, si usa in varie fasi:

* Analisi e gestione dei requsiti.
* Progettazione, per scegliere tra differrenti opzioni, è anche usato ampiamente per testare l’interfaccia grafica.
* Validazione, visto che ha le principali funzioni del sistema e le più delicate posso cominciare da lì a fare una prima validazione.
* A volte si usa anche nello studio di fattibilità

**PROCESSO DI GENERAZIONE DI UN PROTOTIPO:**

Stabilire gli obiettivi del prototipo

Definisco le funzionalità del prototipo

Sviluppo del prototipo

Valutazione del prototipo

“Contorno” del prototipo (cosa fa e cosa non fa)

Rapporto di valutazione

Sistema Sw prototipo

Piano della prototipazione

**DIFFERENZE CON LO SVILUPPO DEL SW**

* Non c’è la validazione (debugging)
* Non implementano tutte le funzionalità
* Controllo solo le cose più importanti
* Stanno sui requsiti funzionali

Nota: esistono linguaggio detti di FAST PROTOTIPING con cui risulta più facile creare un prototipo, inoltre i prototipi andrebbero abbandonati una volta svolta la loro funzione.

1. **GESTIONE DEL CAMBIAMENTO: Tolleranza -> Sviluppo e consegna incrementale**

Sviluppo e consegno varie parti del Sw, prima consegno quelle più importanti e poi quelle più piccole e secondarie. Ci sono alcuni punti da seguire:

* Per decidere cosa è importante lo chiedo al commitente all’inizio dei lavori
* Blocco i requisti della parte che ho deciso di fare mentre parallelamente va avanti l’ analisi dei requisti delle altre parti
* Prima di passare al prossimo elemento devo valutare quello appena composto
* I requisiti possono essere valutati in siuazioni reali

**PROCESSO DI SVILUPPO INCREMENTALE:**

Progettazione dell’architettura del sistema

Assegnare i requisti agli incrementi

Definire il contenuto dei requisti

IS

**VANTAGGI:**

* Le cose più importanti si possono usare fin da subito
* Riesco a capire meglio le cose successive
* Diminuisce il rischio di fallimento del Sw
* Essendo che comincio dalla parte critica del Sw essa verrà testata moltissime volte
* Rapporto più stretto con il committente che deve spesso essere presente.

**SVANTAGGI:**

* Potrei capire alla fine che ho implementato n volte alcune parti che potevano essere messe come comuni a tutti i moduli.

1. **MIGLIORAMENTO DEI PROCESSI SW -> Project management**

Il project management può rendersi necessario quando faccio qualcosa di male nella produzione o quando è presente un’ incompatibilità personale tra gli sviluppatori che lavorano al progetto; esistono due tipi di approccio principale per fare miglioramento del processo:

* Maturità del processo: mi prendo dei momenti prefissati in cui andare a ricontrollare il lavoro, molto statico
* Agile: agisco subito appena mi accorgo che qualcosa non va.

1. **SVILUPPO AGILE:**

volto a ridurre al minimo il tempo di produzione, presenta alcune caratteristiche:

1. Specifica, progettazione e implementazione non sono più fasi distinte
2. Il sistema si sviluppa in varie “sezioni”
3. Frequente consegna di nuove versioni
4. Uso di strumento ad hoc
5. Documentazione ridotta al minimo

**APPROCCIO NORMALE:**

Richiesta di modifica dei requsiti

Specifica dei requisti

**APPROCCIO AGILE:**

Progettazione e Sviluppo

Ingegneria dei requisti

|  |  |
| --- | --- |
| **PIÙ RICERCATO NELL’ AGILE** | **MENO RICERCATO NELL’AGILE** |
| Individui e interazione | Processi e strumenti |
| Softwere che funziona | Documentazione completa |
| Collaborazione del cliente | Stesura del contratto |
| Rispondere al cambiamento | Seguire un piano |

Nota: va bene solo se il progetto non è troppo grande

1. **XP -> Extreme Programming**

* È un tipo di programmazione nata per ridurre al minimo il tempo di produzione di Sw,
* Mostra incrementi al cliente ogni 2/4 settimane,
* Ogni componenete viene testato e quando ho terminato posso passare al prossimo
* I requisti vengono raccontati in “storie” già orientate alla produzione Sw
* Non è plan-driven ma faccio comunque una “Pianificazione incrementale”, pianifico in piccole parti
* La priortià è data all’utente
* Piccole release e progettazione semplice

1. **METODI ALERNATIVI:**

* Test-driven Development: prima progetto i test da fare, poi il modulo.
* Refactoring: continuo a rivedere il Sw già fatto e cerco di rendere miglore per ottenere anche un Sw completo migliore.

1. **XP E PROCESSI AGILE**

* Storie dell’utente: definite dall’utente finale, che è coinvolto nella progettazione, le storie vengono tradotte in sw composti da singoli task
* Refactoring: sw che passa il test non implica di fermarsi; è necessario continuare a migliorare il sw accertandosi che passi sempre i test
* Sviluppo Test-first: si usano strumenti sw per definire/creare i test. Gli sviluppatori creano il test e coinvolgono l’utente finale nella creazione. I test sono solitamente automatizzati
* Programmazione in coppie: gli sviluppatori lavorano in coppia e sviluppano un unico sw per ottenerne uno milgiore di quello che avrebbero ottenuto lavorando singolarmente. Approccio più critico e soluzioni più consolidate.

1. **PROBLEMI NELLO SCALARE PROCESSI AGILE:**

* Progetti medio-piccoli con gli sviluppatori che rsiedono nello stesso luogo
* Scaling UP: per progetti più ampi, più lunghi (durata), e più teeam distribuiti, sw più complessi
* Scaling OUT: quando si vuole introdurre l’approccio agile dentro organizzazioi che non seguono/conoscono questo approccio
* Aspetti concettuali: avere una specifica dettagliata del progetto finale non va d’accordo con lo sviluppo agile

1. **MANUTENZIONE VS SVILUPPO**

* Approccio agile e manutenzione del Sw -> vantaggio
* Cambiamento del team di sviluppo > svantaggio

1. **PLAN-DRIVEN O AGILE?**

1. **MODELLAZIONE DEI SISTEMI SW**

Modellazione del sistema, consiste nella rappresentazione astratta del sistema usando strumenti, vediamo alcuni di questi strumenti:

* Descrizione
* Progettazione
* Prospettiva esterna: rappresento il modello di sistema che interagisce con le parti esterne, definisco l’estremità, il bordo del sistema Sw
* Prospettiva legata all’ interazione: rappresento le interazioni delle singole componenti interne al sistema, come le diverse parti interagiscono tra loro
* Prospettiva strutturale: rappresento i componenti del sistema cioè la sua struttura, come sono fatte le diverse parti del sistema (es. schema di una base di dati)
* Prospettiva comportamentale/dinamica: racconto come evolve il sistema nel tempo
* Approccio a oggetti: UML (unified modelling langauge) insieme di strumenti formali per supportare la modellazione dei sistemi

1. **UML**

Unified modelling system, usato per Sw orientati ad oggetti, propone diversi tipi di diagrammi:

1. Activity Diagrams: raccontano delle attività che sono coinvolte nell’uso del Sw
2. Use-Case Diagrams: usati nella fase di raccolta dei reqsuiti, descrive la parte principale del sistema, chi sono gli attori (ad alto livello)
3. Sequence diagram: usata a due livelli diversi. Descrivono le operazioni nel tempo tra sistema esterno ed esterno, oppure descrivono come i componenti Sw interagisono tra loro.
4. Class Diagram: come è strutturato il sistema Sw e come sono collegati i componenti del del sistema (non interazione)
5. Object Diagram: come è lo stato del sistema Sw in un certo momento

Nota: diagrammi da descrivere all’esame obbligatoriamente, tutti i formalismi UML hanno poche cose obbligatorie e tante opzionali

1. **CLASS DIAGRAM**

* Concetti ∞
* Dettagli sui Cd
  + - * Oggetto: descrizione di un singolo elemento di interesse, è composto da:
        1. Object id (oid)
        2. Attributi + relazioni con altri oggetti (Privati)
        3. Metodi (pubblici)
        4. Gli oggetti sono descritti da un tipo(concetto intensionale -> Struttura comune a tutti gli oggetti -> classi in Java)
        5. Gli oggetti sono raccolti in classi (concetto estensionale)
      * Tipi Atomici
      * Costruttori di tipo: record\_of (tuple), set\_of, list\_of (liste ordinate),bag\_of(element appartenente a un insieme più di una volta)
      * Metodi: costruttori, disruttori, accessori, Trasformatori

DESCRIVE

Valore

HA

Oggetto

CONTIENE

HA

Tipo

Classe

1. **RAPPRESENTAZIONE GRAFICA DI UNO SCHEMA A OGGETTI**

Legenda: Classe

Possono esistere più oggetti con lo stesso stato e OID diverso, definsico quindi la cardinalità, cioè quanti oggetti possono essere associati a un altro ogetto

1, 1

0, N

…

…

Relazione ternaria

{(p1, pR1, s1), (…), …}

Link -> come relazioni in base di dati

{(p1,s1), (p2, s1), (p1, s2)}

LINK

nome: Nausea

data\_nascita: 03-10

P1: Persona

S1: Sintomo

SOFFRE

Sintomo

Nome

Descrizione

setNome()

Persona

Data.Nascita

Nome

cognome

1. **GERARCHIE DI GENERALIZZAZIONE**

Servono a descrivere due concetti diversi:

* Inheritance: dico una sola volta le caratteristiche comuni
* Subtyping: usare istanze di un tipo al posto di istanze di un altro tipo

Es. inheritance:

Inheritance: paziente pediatrico e cardiopatico ereditano da paziente e quindi da persona -> permette il riuso di codice

1. **GERARCHIE DI COMPONENTI**

Si usa un approccio oreintato agli oggetti:

* Oggetto
* Tipo -> descrive la struttura dell’oggetto ed è la sua firma
* Classe
* Class diagram
* Gerarchie di generalizzazione:
  + Riuso codice
  + Sub-typing ->

Contiene

Ereditato da due classi, attributi delle due classi che eredita possono avere lo stesso nome -> ridefinizione degli attributi in conflitto

* Overriding -> multipla definizione di metodi

Overloading del metodo

* Late Binding -> collegamento tra nome metodo e codice del metodo viene eseguito a run-time
* Polimorfismo -> ho lo stesso nome per un metodo a cui corrispondono diverse definizioni
* Co-varianza -> quando specializzo un metodo, faccio in modo che il tipo del parametro di ingresso e ritorno sia una sotto-classe
* Contro-varianza -> quando specializzo un metodo faccio in modo che il tipo del parametro in ingresso e ritorno sia una superclasse (cioè meno specializzato)

1. **CO-VARIANZA DELLE PROPRIETÀ (o degli attributi)**
   1. Esempio 1 in pseudo-codice:

class medico

type tuple(dati : tuple(co\_no: string))

method set\_data

end;

method body set data in class medico{

self -> dati\_anagrafici = tuple (co\_no=”dda”)

…

}

Class cardiologo inherits medico

Type tuple (dati\_anagrafici: tuple(co\_nd: string, indirizzo: string))

Metod read\_indirizzo: string;

End;

Method body read\_indirizzo in class cardiologo{

Return (self -> dati\_a)

}

M= new cardiologo

M -> set\_data

A -> read\_indirizzo

* 1. Esempio 2 in pseudo-codice:

Class paziente

Type tuple(cognomen: string, terapia: farmaco)

Method stessa\_ter(p:paziente): boolean

End;

class paziente\_cardio inherits paziente

method stessa\_ter(pc: paziente\_cardio): boolean

method pazienti\_simili(p: paziente) in class paziente

method body pazienti\_simili(p: paziente) in class paziente

return (p -> stessa\_ter(self) and…)

p1: paziente

pc1: paziente\_cardio

p1: pazienti\_simili (pc1)

1. **USIAMO UML PER DESCRIVERE I DIAGRAM**

**CLASSE:**

Attributi e operazioni ->lower case camel

* Attributi <vis> <nomeparam>:<tipo><molteplicità> = <default><vincoli>(opzionali)
* Operazioni <vis><nome>(<nomeParametro><tipoParametro>):< tipo p.>

Nome classe ->upper case camel

GetAge()

\_age

Person

Persona

Di solito lo stato è privato e i metodi sono pubblici:

+ public

- private

# protected -> visibile soltanto agli oggetti di quella classe e quelli discendenti

~ package -> visibilità solo alle classi dello stesso package

**ISTANZA DI UN OGGETTO:**

Age=20

Jim: persona

<<istance\_of>> -> capire a quale classe appartiene l’oggetto

**RELAZIONI TRA CLASSI:**

associazioni

generalizzazioni (punta vuota)

Realizzazione (punta vuota)

Dipendenza

Aggregazione (rombo vuoto)

Composizione



//

Employers

0..1

\*

person

Catalogo

0 cardinalità min, 1 cardinalità max

1, \*

X

anno

Giocatore

Ha giocato

La x indica l’essere vietata quella direzione

risultato

voto

Esame

0, N

0, N

Insegnamento

Studente

{(s1, i1), (s2, i2)…}

+ cerca()

<interface> riciclabile

Studente

<<use>>>

<interface> ricerca

Bibliotecario

Il rombo del 5 che è questo che unisce studente e docente con università è vuoto

1,N

1,N

0, N

Docente

Studente

Università

1,1

1,1

Il rombo del 6 che è questo che unisce dipartimento e centro amministrativo con università è pieno

1, N

1, N

Centro amministrativo

Dipartimento

1. **ESERCITAZIONI**

1..1

Carrello

**Esercizio n.3:**

1..N

0..1

1.. N

Catlogo

Strumenti usati

Strumento per scuola

Strumenti professionali

Titolare scuola di musica

Musicisti professionisti

Cliente

Vendita

Strumento

0..N

0..N

1.. N

1..1

1..1

Vendita strumenti usati

1..1

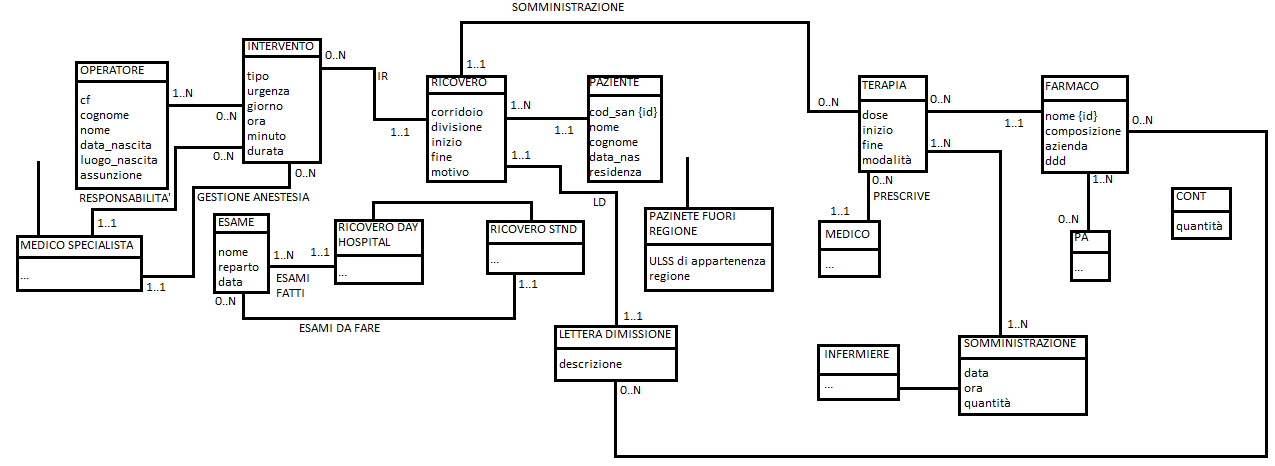
0..1

<- Mette in vendita

1..N

Dettaglio dello sconto

P.s. le freccie sarebbero tutte vuote.

**Esercizio n.2**

Nota:

* Ricordare che i vincoli di cardinalità sono opposti rispetto a quelli dell’ er
* Nomi delle classi sempre al singolare per evitare confusione
* Tutte le freccie sarebbero vuote

1. **USE-CASE DIAGRAM:**

vanno usati nell’analisi dei requisiti, descrivono l’interazione tra l’esterno del sistema e le caratteristiche del sistema, si compongono di varie parti:

* Attori -> si rappresentano con un omino stlizzato, posso fare generlaizzazione degli attori es. cardiologo generlaizzando diventa medico.
* <<include>> -> non è possbile fare una certa funzione se prima non ho svolto quelle funzione che essa include, es.non posso registrare un voto e prima non consulto la lista degli iscritti
* <<extend>> -> sono use-case che vengono usati all’interno di altri use-case solo in casi particolari

1. **DIAGRAMMI DELLE ATTIVITÀ**

Diagrammi che esplicitano attività in senso temporale, sono usati a diversi livelli di astrazione, le componeneti principali sono:

* Attività ->
* Oggetto ->
* Nodi di controllo:
* scelta ->
* biforcazione
* inizio di un insieme di attività
* Fine di tutto il processo
* Fine di una singola via

**Esempio n.1:**

SI’

NO

Pronto?

iscrizioneESAME

ESAME

Esercizi

Studio ESAME

Nota:

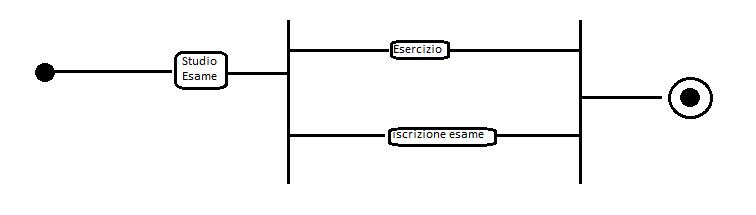
* Devo immaginare come se all’inizio partisse un token e seguisse tutti i percorsi che gli do, esegue l’attività e poi passa alla successiva, c’è un ordine temporale di esecuzione
* La prima sbarra è chiamata nodo di fork in pratica quando è raggiunta da un token ne genera uno per ogni uscita possbile
* La seconda sbarra è un nodo di join in pratica fa attendere l’arrivo di tutti i token per procedere
* Il primo rombo è un nodo di decisione quando gli arriva il token decide in che via farlo passare
* Il secondo rombo è un nodo di fusione attende che gli arrivi un token, il primo che arriva passa.

Linea temporale

Non di partenza

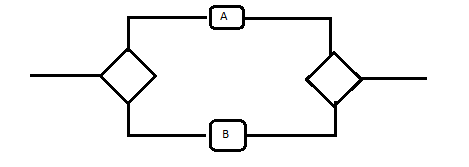
attività

StudIo ESAME



JOIN

FORK: attività parallele o concorrenti

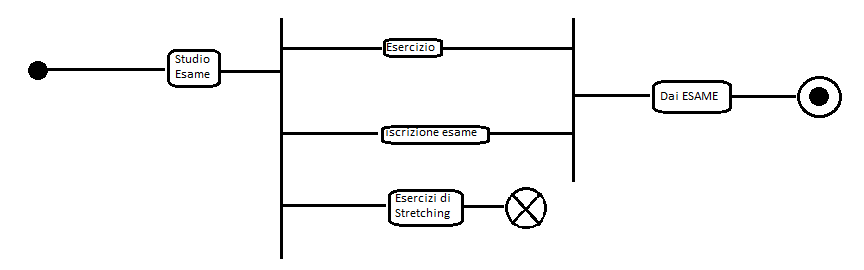


Nodo Di Fusione: il primo che arriva passa.

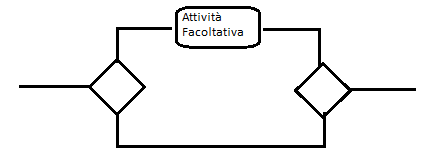
Nodo Decisione: deve esserci una condizione che in base alla sua valutazione dice da che parte andare, possono esserci decisione che hanno più di due uscite cmq tra di loro devono essere sempre mutuamente esclusive.

Nota: Usare sempre i diagrammi in maniera rigoroso, devono essere diagrammi STRUTTURATI, cioè a ogni FORK corrisponde un JOIN e a ogni DECISIONE corrisponde una FUSIONE.

Esempio di diagramma non strutturato bene:



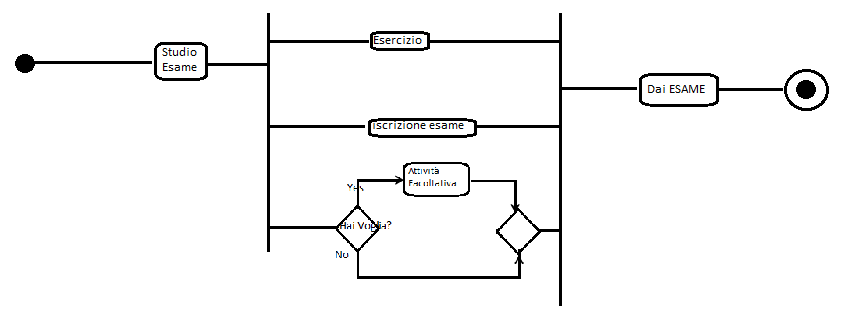
In questo esempio il problema è che, il tutto può finire senza che gli esericizi di stretching siano stati fatti, cioè se io svolgo esercizio e iscrizione esame non mi serve lo streatching per procedere, quindi lo streatching è facoltativo, ma così non si capisce non è intutitivo il fatto che sia opzionale per quanto mostrato si capisce solo che lo streatching è fine a sé stesso in quanto quando il token arriva su di lui procede giunge al nodo con la x che lo elimina, come concetto funziona ma è strutturato male, è meglio esplicitare la non obbligatorietà in altri modi esempio:



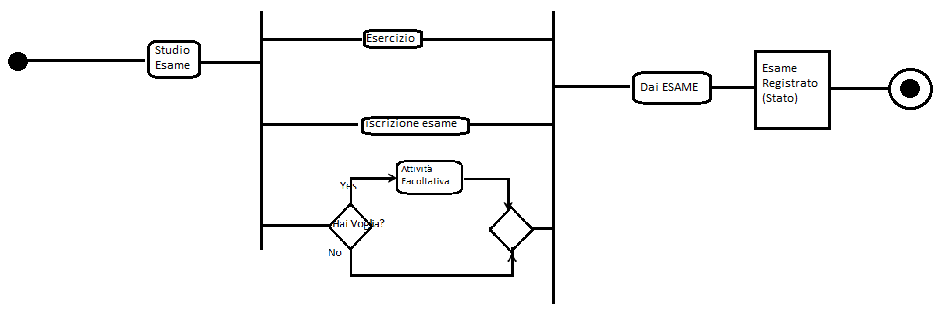
Yes

Hai Voglia?

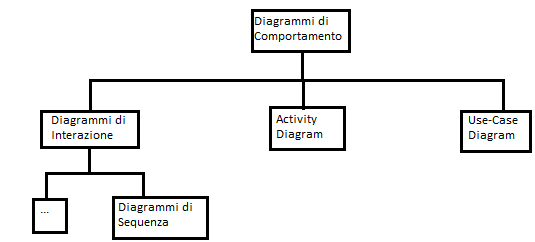
No



Nota: non verrà molto usato ma è possibile aggiungere al diagramma gli oggetti che vengono prodotti o richiesti usando un rettangolo, serve a capire cosa richiede un’attività o cosa produce. Vediamo un esempio:



1. **DIAGRAMMI DI SEQUENZA**

****

Reppresentano interazioni tra entità attraverso lo scambio nel tempo di messaggi, hanno aclune caratteristiche:

* Esplicitano come funziona un comportamento all’interno, come i comportamenti vengono realizzati
* Possono essere usati ad alto o basso livello, cioè o con i casi d’uso o con gli oggetti e i metodi delle classi.
* Dettagliano il comportamento di altri diagrammi
* Componenti principali:
  + Istanze di oggetti -> CLASSIFICATORI cioè iniseme di cose, come una classe o gli attori nello use-case diagram
  + Linee di vita -> CLASSIFICATORI come le istanze solo che questa volta l’oggeto sarà generico, es. persona: paolo è un’istanza, persona: venditore è una linea di vita.

Laureando: studente

(Linea di vita)

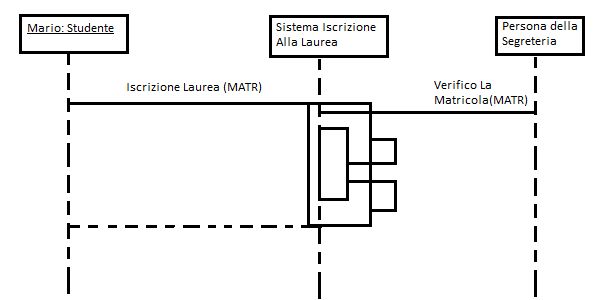
Mario: studente

(istanza)

* + Messaggi -> possono essere di due tipi, Sincrono e Asincrono, in quello sincrono quando invio un mex attendo la risposta per continuare a lavorare in quello asincrono no.

Sincrono

Asincrono

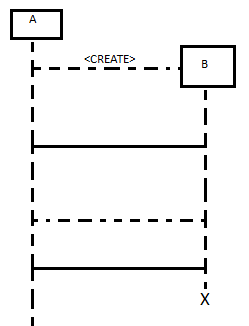


LEGENDA:

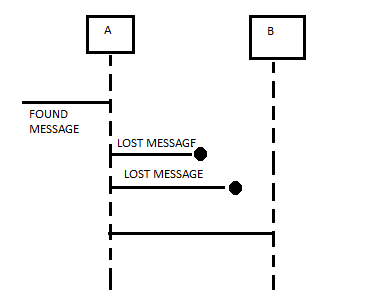
* Le frecce rappresentano i messaggi sincroni o asincrono
* (MATR) è quello che passo assieme al messaggio
* La freccia trattegiata è la risposta, chiamata anche risposta di chiamata, non sono obbligatorie
* Il rettangolo grande è chiamato rettangolo di attivazione ed è dove sta esguendo ora
* Nel rettangolo di attivazione c’è un altro rettangolo perché è come se quello che accade lì chiamasse una funzione di sé stesso nell’esempio con le classi si può pensare a un metodo pubblico come al rettangolo grande che chiama un metodo privato, che sarebbe il rettangolo piccolo
* Le linee trattegiate rappresentano lo scorrere del tempo, danno una linea temporale
* Importante notare che come nel caso delli use-case non posso legare due attori direttamente, Segretaria e studennte non dovranno parlarsi mai.

**ALTRI TIPI DI MESSAGGI:**

1. **Create/Destroy (usati soprattutto per le classi)**

****

Durante la sua vita A crea un oggetto B che avrà una sua linea temporale, quando non gli servirà pià A distruggerà B, create e destroy sono stereotipi e possono avere anche dei parametri in ingresso.

1. **FOUND/LOST MESSAGE**

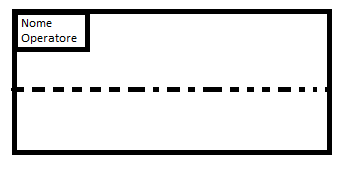
Anche questi non vengono usati nell use-case, il messaggio di FOUND dice che è stato trovato un messaggio che arriva dall’esterno, non so o comunque non mi importa chi ha inviato quel messaggio, invece il messaggio di LOST serve per rappresentare errori di trasmissione, indica che il messaggi ìo non arriva fino a un certo punto.

1. **FRAGMENTS**

Vengono usati nei casi d’uso più complessi, e servono per rappresentare if then, case, while, for… tramite i FRAMMENTI COMBINATI, sono composti da tre parti:

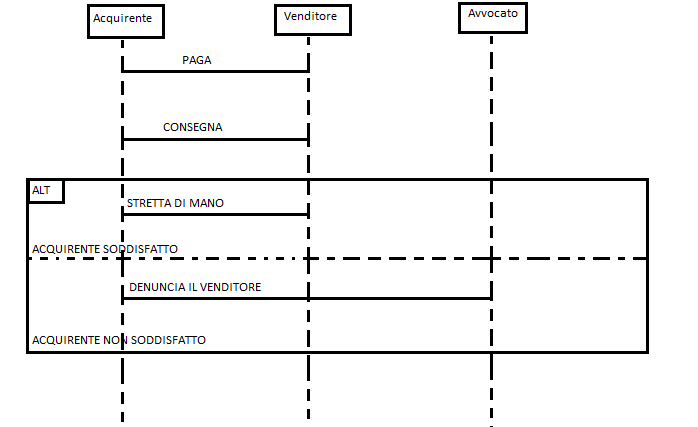
* Operatore: il tipo di frammento che sto usando
* Operandi
* Condizioni DI Guardia: le conidizioni per l attivazione del frammento che però dovranno essere mutuamente esclusive.

Es. di frammento:



* Metto una linea tratteggiata per ogni operando
* Le condizioni le metto tra quadre nella loro fascia
* Devono passarci tutte le linee temporali interessate

Es. concreto:

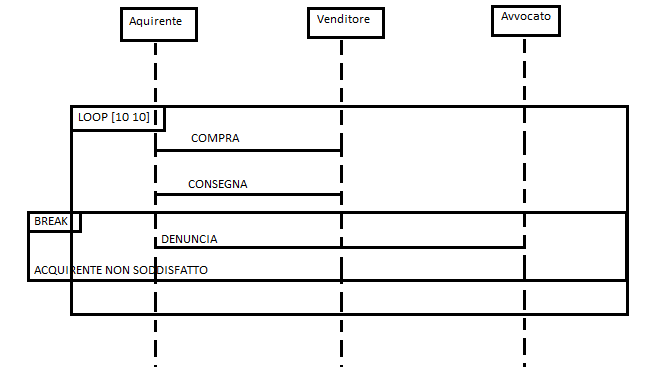


Nota: paga e consegna sono due messaggi diversi, consegna non è la risposta a paga, i segnali di risposta sono più come gli ack in reti.

**OPERATORI:**

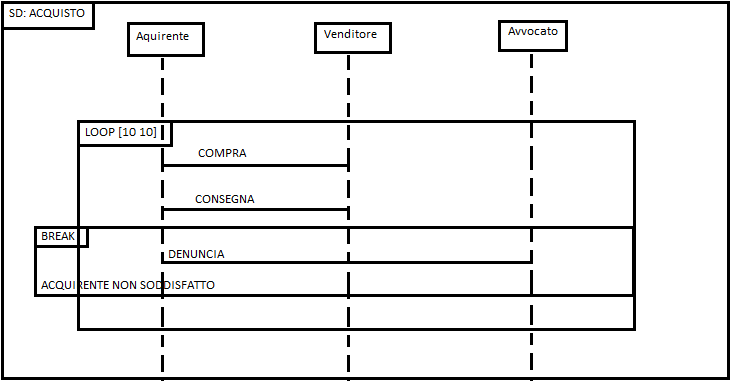
* OPT -> optional, cioè if… then (senza else), 1 operando e 1 guardia
* ALT -> alternative, cioè il case, che è quello usato sopra
* LOOP -> funziona come il for o while, è strutturato così: LOOP min max condizione, in presenza di uno solo dei tre gli altri diventano opzionali, in sintesi deve essercene almeno uno dei tre.
* BREAK -> esce da un ciclo

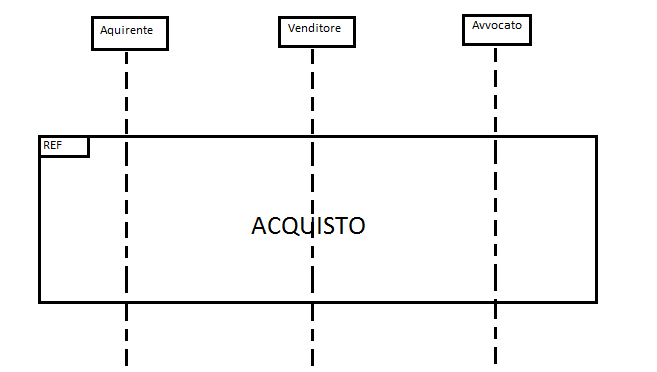
Es. loop e break:



Il loop dovrebbe ciclare 10 volte, continuando a comprare e consegnare, però se l’acquirente non è soddisfatto parte il break, che fa partire la denuncia e esce dal ciclo, per segnare che esce dal ciclo una parte del frammento di break deve sempre essere fuori dal ciclo loop da cui vuole uscire.

1. **REF**

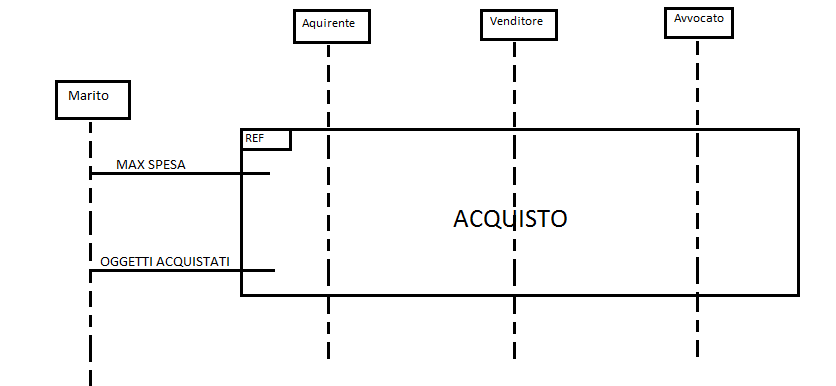
Ref sta per riferimento è un frammento partcolare che viene usato per indicare che in quel luogo verrà messo un frammento dichiarato in seguito, es:



In pratica io creo un riferimento a una cosa che creo da un’altra parte in questo caso faccio riferimento all’acquisto di pagine sopra, nell’oggeto dove c’è la vera implementazione del frammento va messo come operatore SD: nome\_frammento.

1. **GATE:**

hanno la partcolarità di poter partire da qualsiasi punto della cornice, e rappresentano la comunicazione dei frammenti con l’esterno, o verso l’esterno, servono quando si usano i REF anche per coordinare le attività



In questo esempio il marito mette una spesa massima all’acquirente, e poi riceve gli oggetti che vengono conegnati, grazie ai GATE i frammenti possono comunicare con il mondo esterno.

Nota: gli errori più comuni sono di far parlare gli attori tra di loro quando invece non è possibile come negli use-case, e un altro errore comune è di dettagliare troppo, mai dettagliare più del dovuto.

1. **PROGETTAZIONE ARCHITETTURALE**

Consiste nella progettazione di come va organizzato il Sw, la sua “ossatura” principale; a livello macro in pratica mostra quali sono le componenti del sistema e come interagisono, è molto importante anche se si usa un apporccio agile. Esistonono due diversi tipi di approccio alla progettazione architetturale:

1. In Grande: racconto l’architettura di un sistema che è un iniseme di Sw
2. In Piccolo: architetture legate al singolo programma, come esso è suddiviso.

Vado a raccontare le parti principali del Sw e come si parlano tra loro, lo faccio secondo varie viste:

* Vista logico/concettuale: mi concentro sulle parti più importanti e gli aspetti strutturali -> oggetti e classi principali
* Vista dinamica o di processo: va a completare la vista logico/concettuale, vado a vedere come i vari processi interagiscono, scambiano informazioni, mutano nel tempo.
* Vista fisica
* Vista di sviluppo: devo decidere come divido le componenti Sw (es. la componente x va assegnato al gruppo di lavoro y)

1. **ARCHITETTURE E CARATTERISTICHE DEI SISTEMI SW**

* Aspetti di Performance -> per farlo devo minimizzare la comunicazione tra sotto- parti, in pratiche cerco di suddividere in sotto-parti tra loro il più possibile indipendenti
* Security -> sicurezza nel senso di accesso permesso solo a persone accreditate, ci dovrà essere una parte dedicata solo a questo e dovrà essere il più possibile efficiente perché condivisa da tutti
* Safety -> il Sw non deve essere pericoloso, nel senso che non deve perdere o divulgare dei dati o in casi peggiori per robot o altro agire in maniera pericolosa.
* Avalability (Disponibilità)
* Mantenimento -> i singoli componeneti devono poter essere ripiazzati facilmente senza stravolgere ogni cosa.

1. **PATTERN ARCHITETTURALE**

Pattern= modello/ esempio di/ configurazione/ archetipo, in pratica è la soluzione a un problema ricorrente che invece di venire reimplementata ogni volta, viene riutilizzata

Pattern architetturale= soluzioni sintetizzate/ astratte che cerco di applicare nel contesto che sto considerando.

1. **MVC**

Model-view-controller è un pattern architetturale, divide ogni Sw in 3 parti che sono:

* Model -> ha a che fare con la base di dati, è la parte che contiene le informazioni
* View -> è la parte che interagisce con l’utente e con il model(tramite le query), inoltre il model notifica i cambi di stato nei dati al view
* Controller -> dice al sistema cosa va fatto, realizza la logica applicativa. Il controller può ricevere messaggi dalla view e agisce di conseguenza.

Browser

DBMS

Controller

View

Model

Nota: se aggiungo un browser come nell’esempio diventa un sistema Web MVC, uno svantaggio di questo sistema è che se ho operazioni anche molto semplici devo comunque passare per il model.

1. **ARCHITETTURA A STRATI (LAYARED)**

È un pattern architetturale; divido il Sw in vari strati Sw di cui ogni strato riceve servizi dallo strato inferiore e ne fornisce allo strato superiore.

1. **ARCHITETTURA A REPOSITORY**

È un pattern architetturale, rappresenta sistemi fortemente basati sui dati, le componenti non si parlano tra loro ma comunicano solo leggendo e scrivendo da un REPOSITORY centrale.

B

C

A

REPOSITORY

1. **ARCHITETTURA CLIENT-SERVER**

È un pattern architetturale, rappresenta un modello distribuito

Nota: lo svantaggio principale è che se il server va giù non funziona più nulla, inoltre da notare che questi pattern non sono tra loro mutuamente esclusivi, cioè in un’architettura client-server il server può essere implementato con MVC.

1. **ARCHITETTURA PIPE ANF FILTER**

È un pattern architetturale, ogni componente ha un tot di ingressi e deve produrre delle uscite, l’intero processo deve portare all’uscita del mio Sw.

…

Scelta

Filtro C

…

Filtro A

Dati

1. **ARCHITETTURA DI APPLICAZIONI**

È un pattern architetturale, vado a vedere le architetture di applicazioni differenti, posso verificare che funzioni siano supportate, posso capire che devo riservare alcune partie, esempi:

* Data processing system -> elaborazione dati
* Applicazioni d gestione delle transizioni basate su Web -> dati che supportano transizioni atomiche
* Sistemi di gestione di eventi
* Sistemi di elaborazione del linguaggio

1. **DESIGN PATTERN**

Questa parte è lasciata alle diapositive

1. **SW TESTING**

Verificare il programma creato, devo verificare che:

* Il programma fa quello per cui è stato scritto
* Devo scoprire eventuali difetti del Sw (anomalie di comportamento)

Nota: in questa fase lavora sempre il team di sviluppo, il testing certifica la presenza di errori, ma non l’assenza.

Il testing è una parte delle verifica e validazine; vediamo un esempio di testing

**-> DEFECT TESTING**

Anche qui c’è un sottoinsieme di dati dedicati ai comportamenti anomali

All’interno di input test ora c’è un sottoinsieme dei dati di input che contiene dati per comportamenti anomali

Sw System

1. **VERIFICA VALIDAZIONE SW**

**Verifica**: che il Sw sia scritto bene e documentato bene

**Validazione**: che il Sw faccia quello che deve fare

**Ispezione**: analisi statica del Sw (senza farlo girare), può contenere:

* i requisti
* la progettazione dell’architettura
* documenti UML
* Schemi base di dati
* Programma (il codice)

**Test**: controllo facendo girare il Sw stesso, contiene:

* Programmi
* Prototipo

Nota: l’ispezione può essere fatta anche per parti ma non mi permette di vedere caratteristiche non funzionali, il test invece deve sempre essere fatto sul programma o sul prototipo.

1. **TEST**

Parti principlai di test:

Record finale

Dati di test

Risulatati del test

Test case

Nota: alla fine controllo che i risultati ottenuti siano come quelli attesi

Esistono 3 diversi tipi di test:

* Test di sviluppo: eseguito dallo sviluppatore, cerco bug e problemi
* Test di release: do a un gruppo di lavoro diverso dal mio il mio Sw per fare dei test e verificarlo.
* Test con utenti: si divide in alfa-test che coinvolge gli utenti in modo ristretto cmq sempre all’interno dell’organizzazione, beta-test lascio verificare il Sw agli utenti generici.

1. **TEST DI SVILUPPO**

Si divide in:

* Unit test: controllo che funzioni bene le singole parti di un programma, ad esempio se parlo di classi controllo tutte le funzioni di una classe(RICHIESTO NELL’ELABORATO), bisogna controllare tutti i metodi, che le variabili siano giuste, e provare l’oggetto in tutti i suoi stati.
* Component test: controllo che l’interazione tra le componenti funzioni
* System test: controllo che tutto unito funzioni il mio Sw

Come funziona lo unit-test automatizzato:

* Setup -> dico al sistema cosa gli do in input e che output mi aspetto
* Call -> chiamo i vari metodi e oggetti
* Assertion -> capisco come è adnato il test

Nota: Nell’elaborato inserire dati reali.

Esistono due strategie principali di test:

* Test di partizioni: capisco se esistono gruppi di dati che presentano situazioni analoghe es studenti, studenti è una classe molto eterogenee, io per ogni partizione di studenti che riesco a fare (triennali, magistrali) faccio un test.
* Test basato su linee guida: metto all’inizio una serie di regole per generare i test in modo intelligente.

Nota: Per generare i test posso usufruire anche di use-case diagram o sequel diagram

1. **TEST DI RELEASE**

Questo test non è eseguito dallo sviluppatore ma da un altro team all’interno dell’azienda, viene usato un approccio a **BLACK-BOX,** cioè per loro il mio Sw è una scatola nera, non hanno idea di come sia implementato ma controllano che rispetti le specifiche e quanto è stato scritto nel documento dei requsiti; vengono controllati molto bug e malfunzionamenti.

1. **TEST DELL’UTENTE**

L’utente fa anch’esso il test usando dati reali e con le sue consocenze dell’ambiente in cui il Sw prodotto andrà utilizzato.

1. **EVOLUZIONE DEL SW**

Serve perché ormai il cambiamento è inevitabile per i seguenti motivi:

* Posso venir fuori nuovi requisti
* L’ambito organizzativo cambia
* Errori
* Nuove tecnologie
* Performance o affidabilità non sono sufficienti

Fasi per cui passa un Sw:

Gestione Del Servizio: fase in cui non accetto più cambiamenti del Sw ma solo correzzione di errori

Ritito del Sw: non gestisco più neanche gli errori il Sw è lasciato a sé stesso

Come capisco quando e come cambiare?

* La richiesta di cambiamento è come quella di nuovi requsiti, in pratica faccia una nuova anlisi dei requsiti, il problema è che chi gestisce l’evoluzione del Sw spesso non è chi l’ha prodotto, questo può generare problemi di comprensione soprattutto in mancanza di documentazione adeguata.

1. **SISTEMI LEGACY**

Vuol dire sistemi ereditati, in pratica sono sistemi di cui ho perso il controllo magari perché fatti con tecnolgia obsoleta e spesso sono anche senza documentazione, inoltre sono spesso considerati Sw SOCIO-TECNICI, che vuol dire in pratica che il Sw è diventato la parte fondamentale su cui è basata l’azienda, quello che dice il Sw è la verità.

Processi organizzativi

HW

Dati applicazioni

Regole e politiche

Sw di supporto

Applicazione Sw

Caratteristiche di un Sistemi Legacy:

* Sono molto costosi
* Sono vitali per l’organizzazione
* Manca documentazione
* Integrazione con processi organizzativi

Progettare nuovi sistemi costerebbe molto

Cerchiamo di evolvere il sistema:

come capisco se cambiare o no? Devo valutare la qualità del Sw sottostante e quanto tale Sw sia fonadamentale per l’azienda, in base a questi parametri decido se posso permettermi un cambaimento o meno. Inoltre posso fare una valutazione dell’ambiente cioè il produttore del mio sistema legacy esiste ancora? A livello Hw il mio sistema va bene?

E per mantenere il Sw?

È necessaria la riparazione di errori, adattamento ambientale(cambio di piattaforma), la modifica di funzionalità(gran parte dello sforzo).