Operazione Rif. PA 2017-7537/RER Approvata con deliberazione di Giunta Regionale n. 953 del 28/06/2017 e cofinanziata con risorse del Fondo sociale europeo e della Regione Emilia-Romagna

Corso IFTS

"TECNICO PER LA PROGETTAZIONE E LO SVILUPPO DI APPLICAZIONI INFORMATICHE SPECIALIZZATO IN SOLUZIONI WEB ORIENTED"

Periodo di svolgimento: novembre 2017 – giugno 2018

Durata: 800 ore

DISPENSE DIDATTICHE

Modulo n°: 7

Titolo modulo: FONDAMENTI DI PROGRAMMAZIONE E DI INGEGNERIA DEL SOFTWARE

Docente: Claudio Rossi

Fondazione En.A.I.P. S. Zavatta Rimini
Viale Valturio, 4 47923 Rimini
Tel. 0541.367100 – fax. 0541.784001
www.enaiprimini.org; e-mail: info@enaiprimini.org

Ciclo di vita del software

Il ciclo di vita del software è la scomposizione delle attività legate alla produzione di un software.
Le fasi principali che caratterizzano il ciclo di vita di un software sono:

- analisi
- progettazione
- implementazione
- collaudo
- rilascio
- manutenzione

Analisi del software

L'analisi del software è la prima attività dello sviluppo e consiste nello studio del contesto applicativo nel quale opererà il software e delle caratteristiche che questo deve possedere.
Una parte fondamentale dell'attività di analisi è l'interazione con il cliente per definire le caratteristiche che il software dovrà possedere, a tal fine è necessario realizzare una o più interviste con il personale del cliente al fine di definire ogni possibile aspetto rilevante (dall'aspetto estetico delle interfacce agli aspetti di usabilità richiesti).
Il fine del processo di analisi è quello di creare un documento (detto *documento di specifiche funzionali*) che contiene tutte le caratteristiche individuate durante l'analisi, normalmente questo documento viene sottoposto al committente e fatto firmare per l'accettazione delle richieste software, al fine di evitare contestazioni successive.

Progettazione del software

La progettazione serve a definire la struttura del software.
La struttura del software è costituita da un elenco o da uno schema grafico dei moduli che lo comporranno.
Per ogni modulo devono essere indicate le specifiche funzionali e le interfacce.
Le specifiche funzionali indicano quali funzionalità deve implementare il modulo, le interfacce indicano quali proprietà e metodi il modulo deve esporre all'esterno e quali firme devono avere le proprietà e i metodi.
Esempio
l'analisi produce un documento nel quale indica di dover produrre due moduli:
Un modulo che si chiama DBInterface che si occupa di leggere e scrivere sul database, il modulo deve esporre il metodo FindEmailAdresses che accetta due parametri di tipo stringa e restituisce un vettore di stringhe, i due parametri dovranno contenere rispettivamente il nome ed il cognome del destinatario, il metodo dovrà cercare nella tabella Contatti del database tutti coloro che hanno nome e cognome corrispondenti, costruire un vettore di stringhe contenente i relativi indirizzi email e restituirlo all'esterno.
Un modulo che si chiama MailServices che si occupa di fornire servizi email, il modulo deve esporre il metodo SendEmail che accetta tre parametri di tipo stringa contenenti rispettivamente il nome, il cognome del destinatario ed il corpo della email, il metodo dovrà utilizzare il metodo FindEmailAdresses del modulo DBInterface passando nome e cognome, se il metodo ritorna un vettore vuoto dovrà mostrare un messaggio per avvisare l'utente dell'impossibilità di trovare i dati, se restituisce un vettore non vuoto dovrà inviare una email contenente i dati scritti nel terzo parametro per ogni indirizzo indicato nel vettore.

Implementazione del software

L'implementazione è la fase di programmazione, ossia della stesura del codice in uno o più linguaggi di programmazione.
È compito del team di sviluppo definire i metodi privati del modulo, gli unici vincoli ai quali devono sottostare sono quelli delle interfacce e delle specifiche funzionali.
Eventuali metodi privati possono essere aggiunti secondo le necessità che si presenteranno durante la stesura del codice.
Esempio
Durante l'implementazione del metodo FindEmailAdresses nel modulo DBInterface gli sviluppatori decidono di creare un metodo privato CheckAddress che accetta un parametro di tipo stringa e restituisce un parametro di tipo boolean: se il valore passato nel parametro è un indirizzo valido restituirà Vero, altrimenti restituirà Falso.

Controllo di versione

Il controllo di versione è un sistema che consente di gestire in maniera organica più versioni del software che si sta sviluppando.
Il sistema si basa su un'architettura client-server che può essere locale, distribuita in LAN o in WAN.
Il server archivia i dati in un *repository*, questo può essere un file, un insieme di cartelle o un database, a seconda del prodotto che si utilizza.
Ogni modifica al progetto viene archiviata nel repository e le viene assegnato un numero progressivo univoco (detto numero di versione) che consentirà di identificarla in maniera semplice da quel momento in poi.
In qualunque momento del lavoro è possibile:

- sincronizzare la propria copia locale del progetto con l'ultima versione presente nel repository
- tornare ad una versione precedente
- confrontare due diverse versioni
- verificare quale elemento del gruppo di lavoro ha prodotto una particolare riga di codice o modifica di progetto
- generare branch di sviluppo
- integrare branch di sviluppo

Continuous testing

Il continuous testing è un sistema di verifica automatica che consente di verificare il corretto funzionamento del codice scritto o in fase di scrittura.
Si basa su un software che analizza il codice sorgente, esegue dei test predeterminati (detti classi di test) e fornisce i risultati dell'esecuzione del codice (presenza di errori, tempo di esecuzione ed altre informazioni), il tutto mentre lo sviluppatore continua a lavorare.

Collaudo

Il collaudo consiste nella verifica di quanto il software realizzato rispetti (totalmente o parzialmente) i requisiti stabiliti in fase di analisi.
Qualora le specifiche stilate e sottoscritte in fase di analisi non siano state rispettate, viene prodotto un documento contenente l'elenco delle mancanze e degli errori riscontati ed il software torna alla fase di implementazione.

Bug Tracking

il Bug Tracking System è un sistema con architettura client-server che consente al team si sviluppo di tenere traccia di segnalazioni bug, richieste di nuove funzionalità e altre informazioni.
In alcuni casi si lascia agli utenti del software la possibilità di inserire direttamente le segnalazioni sul server di bugtracking, in modo da consentire la concentrazione delle segnalazioni in un unico punto e da rendere agevole la gestione della correzione dei bug.
Gli elementi che costituiscono un Bug Tracking System sono

- un server contenente il database con tutte le segnalazioni
- un client per l'inserimento delle segnalazioni e l'aggiornamento delle stesse

Spesso come client viene utilizzato un comune Web Browser in modo da semplificare l'accesso alle informazioni.
Le informazioni principali che vengono registrate sono:

- data della segnalazione
- tipo di segnalazione (bug, richiesta nuova funzionalità, miglioramento di funzionalità esistente e altre)
- sviluppatore al quale assegnare la gestione della segnalazione
- ore di lavoro stimate per la chiusura della segnalazione
- ore di lavoro effettivamente svolte
- stato di chiusura (bug corretto, situazione non riproducibile, funzionalità aggiunta e altre)

Rilascio

La fase di rilascio consiste nell'installazione del software nell'infrastruttura di utilizzo (detta *ambiente di produzione*).
Il rilascio può essere costituito dalla semplice copia di un file così come da una procedura estremamente complessa che richiede l'intervento di tecnici appositamente formati.

Manutenzione

La manutenzione consiste nel supporto al software già distribuito tramite la creazione e la distribuzione di patch che correggono errori sfuggiti alle fasi di implementazione e collaudo, aggiunge supporto per nuovi ambienti hardware o software, aggiunge nuove funzionalità al prodotto.

Architettura del software

Lo standard ANSI/IEEE Std 1471-2000 definisce l'architettura del software come:

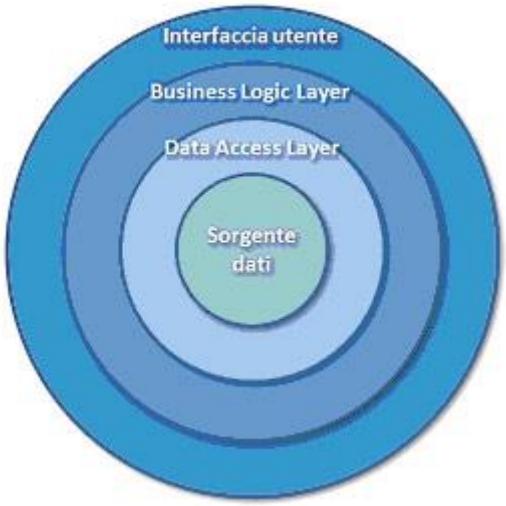
> l'organizzazione basilare di un sistema, rappresentato dalle sue componenti, dalle relazioni che esistono tra di loro e con l'ambiente circostante, e dai principi che governano la sua progettazione ed evoluzione.

Detto in termini semplici, l'architettura di un software è l'organizzazione strutturale del sistema: definisce i componenti software[1], le proprietà visibili di questi componenti e le relazioni fra gli elementi. L'architettura di un software non comprende solamente la sua struttura ma anche le modalità con cui le diverse parti si integrano e interagiscono, gli aspetti legati all'interoperabilità con altri sistemi, il livello con cui l'applicazione soddisfa i requisiti funzionali, le caratteristiche orientate a favorire l'evoluzione nel tempo del sistema a fronte dei suoi cambiamenti strutturali e in relazione all'ambiente in cui esso è inserito (scalabilità, performance, manutenibilità, sicurezza, affidabilità, ecc.).
Ciascun componente entra a far parte dell'architettura in funzione del ruolo che esso ricopre. Ogni componente presenta caratteristiche peculiari (nella definizione denominate "proprietà visibili esternamente") che influenzano il modo con cui ciascuna parte del sistema comunica e interagisce con le altre. L'architettura considera gli aspetti che sono inerenti la comunicazione tra le parti, si focalizza sulle modalità di interazione, tralasciando i dettagli di funzionamento interni.
L'architettura è una rappresentazione che permette all'architetto di analizzare l'efficacia del progetto per rispondere ai requisiti stabiliti, di considerare e valutare le alternative strutturali in una fase in cui i cambiamenti abbiano ancora un impatto relativo sull'andamento del progetto e sul risultato finale e di gestire in modo appropriato i rischi che sono collegati alla progettazione e alla realizzazione del software.
Nei moderni sistemi software le parti interagiscono tra loro per mezzo di interfacce che suddividono in modo netto ciò che non è direttamente accessibile dall'esterno da ciò che è pubblico. L'architettura si concentra unicamente sul secondo aspetto, tralasciando i dettagli interni che in generale non influenzano (o quasi) il modo con cui i componenti si relazionano tra loro. Le interazioni fra i componenti possono essere semplici (come una chiamata a funzione) oppure complesse (come un protocollo di comunicazione o un meccanismo di serializzazione).

Stratificazione del software

Questo stile di architettura prevede la strutturazione logica di un sistema software in strati sovrapposti (layer), tra loro comunicanti, ciascuno caratterizzato da una forte omogeneità funzionale.
La forma più nota e usata riguarda l'architettura a tre livelli composta da: *User Interface* (UI) o strato di presentazione, dove vengono gestite le interazioni dell'utente col sistema, *Business Logic Layer* (BLL) o strato di business, dove sono presenti i servizi applicativi, e *Data Access Layer* (DAL) o strato di accesso ai dati, dove sono gestite le interazioni con il sistema di persistenza delle informazioni.

[1] Per componente software si intende qualsiasi entità facente parte di un sistema: dal semplice modulo applicativo (es: una classe in un'applicazione basata sul paradigma ad oggetti) al sottosistema complesso (per esempio, un DBMS).


Interfaccia utente
Business Logic Layer
Data Access Layer
Sorgente dati

Programmazione object oriented

I sistemi informatici hanno una struttura interna complessa e sono composti da molti moduli ciascuno dei quali può contenere migliaia di istruzioni, pertanto diventa essenziale la corretta comunicazione tra moduli e la garanzia di funzionamento di ognuno di questi.
La complessità di un sistema aumenta in maniera esponenziale con l'aumentare dei componenti ed il comportamento del sistema può essere inatteso, considerato il numero enorme di possibili stati in cui può trovarsi, la scomposizione di un sistema in funzioni risolve parzialmente il problema, con una tendenza al degrado delle prestazioni conseguentemente ad una modifica dei requisiti.
Per ovviare a tutti questi problemi è nata la Programmazione Object Oriented (OOP).
La programmazione orientata agli oggetti[2] (o programmazione ad oggetti) è un paradigma di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi, è particolarmente adatta nei contesti nei quali si possono definire delle relazioni fra i moduli che costituiscono il software.
La programmazione ad oggetti prevede di raggruppare in una classe[3] la dichiarazione delle strutture dati e delle procedure che operano su di esse.
La OOP presenta molti vantaggi, i principali sono:

- fornisce un supporto naturale alla modellazione software degli oggetti del mondo reale o del modello astratto
- aiuta la suddivisione del lavoro all'interno di un team di sviluppo
- permette una facile gestione e manutenzione di progetti di grandi dimensioni
- l'organizzazione del codice sotto forma di classi favorisce la modularità e il riuso di codice
- il codice prodotto è compatto perché riutilizza al meglio componenti di piccole dimensioni

Il progetto finale si ottiene tramite la realizzazione degli oggetti nelle loro componenti strutturali (classi ed oggetti) e comportamentali (proprietà e metodi).
La codifica è basata sulla cooperazione delle classi che sono state progettate per svolgere un compito assegnato.
Le classi costituiscono dei modelli astratti, che durante l'esecuzione del codice vengono invocate per instanziare oggetti software relativi alla classe invocata, questi oggetti sono dotati di proprietà e metodi secondo quanto definito dalle rispettive classi.
Le classi sono caratterizzate da:

- proprietà
- metodi
- costruttore
- distruttore

Una classe si utilizza in maniera simile ad un tipo di dato: si dichiara una variabile del tipo definito dalla classe e, se necessario, si istanzia la variabile (si invoca il costruttore della classe) dopodiché si utilizzano i metodi e le proprietà che la classe espone.
Ad esempio: disponendo di una classe Automobile, è possibile dichiarare una variabile miaAuto di tipo Automobile, istanziarla, verificare la proprietà LivelloCarburante e chiamare i metodi Accendi e Accelera.
Un linguaggio di programmazione è definito ad oggetti quando la sua sintassi nativa permette di implementare tre meccanismi:

- incapsulamento
- ereditarietà

---

[2] La programmazione orientata agli oggetti viene spesso indicata dalla sigla OOP, Object Oriented Programming
[3] Nella programmazione orientata agli oggetti una classe è una zona isolata del codice sorgente

- polimorfismo

Incapsulamento

Si definisce incapsulamento la possibilità di nascondere il funzionamento interno di una parte di programma (tipicamente di una classe), questo consente di isolare le modifiche apportate al codice in modo da ottenere moduli che lavorano assieme ma che sono modificabili in maniera completamente indipendente.
Un'altra conseguenza positiva dell'incapsulamento è che questo rende possibile la modifica dell'implementazione di uno o più oggetti senza dover modificare il codice che li utilizza, in pratica le modifiche effettuate al codice di una classe non richiedono modifiche al codice delle altre classi.
Il permesso di accedere ai dati interni all'oggetto viene gestito interamente dall'oggetto stesso, in questo modo i dati sono protetti da possibili modifiche effettuate dall'esterno e si ha sempre la garanzia che gli stati interni di una classe siano al sicuro.

Ereditarietà

L'ereditarietà è un meccanismo che consente di derivare nuove classi (dette sottoclassi o classi figlie) a partire da classi esistenti (dette superclassi o classi padre), creando una gerarchia di classi.
Quando una sottoclasse eredita da una superclasse mantiene i metodi e gli attributi della classe da cui deriva, inoltre può definire i propri metodi o attributi e ridefinire il codice di alcuni dei metodi ereditati.
Ad esempio ipotizziamo di avere una classe *automobile* con le sue proprietà (colore, cilindrata, ...) e i suoi metodi (accendi, spegni, ...); se creiamo una classe *utilitaria* derivata dalla classe automobile, la classe utilitaria erediterà automaticamente proprietà e metodi di *automobile*.

Polimorfismo

Il polimorfismo è una caratteristica che garantisce che le istanze di una sottoclasse possono essere utilizzate al posto di istanze della superclasse, detto in altri termini è la possibilità di utilizzare lo stesso nome per funzioni diverse applicate a classi diverse.
Ad esempio consideriamo una automobile, questa ha dei comportamenti diversi a seconda del modello.
In particolare per accenderla (se si trattasse di un programma invocando il metodo Accendi() della classe Automobile) il comportamento effettivo del motore è diverso a seconda che si tratta di una Ferrari monoposto di formula uno, o una utilitaria per andare alla spesa ma in ogni caso entrambe si accendono.
In pratica ad una richiesta che è abbastanza standard (accendere la macchina), la specifica automobile sa come comportarsi.

Proprietà

Una proprietà identifica lo stato di un oggetto, può far riferimento ad una o più variabili appartenente alla classe.
Le variabili di una classe possono essere viste dall'esterno solamente tramite le proprietà che la classe stessa mette a disposizione del programma chiamante, questo garantisce che nessuno possa modificare in maniera impropria lo stato di un oggetto.
Tornando all'esempio dell'automobile: la proprietà colore indica di che colore è l'automobile, leggendolo da una o più variabili interne della classe.

Metodo

Un metodo è una funzione appartenente ad una classe, può essere esposto all'esterno o essere privato della classe.
I metodi (sia pubblici che privati) hanno pieno accesso alle variabili interne della classe, pertanto possono leggerle o modificarle liberamente.
Ancora dall'esempio dell'automobile: la proprietà colore potrà essere modificata solamente utilizzando un apposito metodo implementato dalla classe automobile (ad es. Dipingi) e non direttamente, questo consente a chi scrive la classe automobile di decidere quando e come sia possibile cambiarne il colore.

Costruttore

In una classe possono essere presenti uno o più metodi (tra loro si differenziano per tipo o numero di argomenti) che vengono invocati automaticamente quando un oggetto viene istanziato e solitamente servono per inizializzare le proprietà dell'oggetto.
Se una classe ha uno o più padri (perché eredita da altre classi) anche i costruttori dei padri vengono automaticamente invocati.

Distruttore

In una classe può essere presente un solo metodo distruttore che viene invocato quando un oggetto viene distrutto e che consente di gestire la pulizia delle variabili interne.
Solitamente viene utilizzato per deallocare la memoria occupata dall'oggetto, altri utilizzi tipici sono la chiusura delle connessioni aperte (verso database, file o entità di rete).
Molti linguaggi orientati agli oggetti eseguono automaticamente l'operazione di deallocazione della memoria.

Overloading

Con il termine overloading si indica la definizione, in una stessa classe, di più funzioni con stesso nome ma argomenti differenti.
Il termine overloading si utilizza solamente per indicare una definizione di un metodo con firma (numero e tipo dei parametri) diversa.

Overriding

Una classe figlio può ridefinire (quindi usando stesso nome metodo, numero e tipo argomenti) un metodo definito nella classe padre, se questo metodo viene invocato verrà utilizzato quello della classe figlio, come se nella classe padre non esistesse.

Sommario