**ciclo di vita del software**

si compone di diversi elementi, ognuno di questi richiede competenze specifiche

1. Analisi
2. Progettazione
3. Implementazione
4. Collaudo
5. Distribuzione
6. Manutenzione

Analisi

La fase di analisi definisce i bisogni, ossia specifica quali input e output deve gestire il software, inoltre deve definire il contesto dell’applicazione (sistema operativo di destinazione, tipo di utente, tipo di hardware)

L’analisi produce il documento di specifiche funzionali

Controllo di versione

È un software che consente di memorizzare gli stati di un file, ossia di tenere traccia di tutta la storia dell’evoluzione del file stesso

Il sistema è costituito da un’architettura client-server

L’architettura client-server consente di avere diversi vantaggi:

* Cloud (per il server)
* Ricostruzione di stati consistenti indipendentemente dalla posizione di sviluppo
* Rende semplice la scrittura collaborativa

Esistono diversi protocolli per il controllo di versione, i più diffusi sono GIT e SVN

Noi utilizzeremo github (server), tortoise (client) e visual studio code (client)

I server di controllo di versione raccolgono la storia dei file in contenitori chiamati repository

Un repository può essere pubblico o privato e l’accesso in lettura o scrittura può essere condiviso con altri utenti

I comandi principali sono:

* Clone, duplica sul computer locale lo stato attuale dei file presenti nel repostery
* Commit, crea una nuova versione dei file da mettere sul server
* Revert, annulla tutte le modifiche ai file apportate dopo l’ultima commit (torna all’ultima versione disponibile)

Progettazione

La progettazione del software trasforma il documento di specifiche funzionali in uno schema a blocchi che rappresenta le unità funzionali

Un’unità funzionale è un contenitore di codice che si può interfacciare con altre unità funzionali: un’unità funzionale riceve uno o più input e produce uno o più output

Il progetto è costituito da uno schema a più livelli, ogni livello può essere costituito da uno schema

La separazione della soluzione in unità funzionali consente di riutilizzare il codice già scritto

Spesso si utilizzano librerie software, ossia unità funzionali standard disponibili sul mercato o scritte dal team di sviluppo per altri progetti

Per ogni unità funzionale, il progettista deve indicare tutti gli input, tutti gli output e tutte le elaborazioni che portano alla produzione degli output

Esistono sistemi di codifica standardizzati per la produzione degli schemi, il più diffuso è il linguaggio UML (Unified Modeling Language)

Alcuni strumenti di progettazione producono (oltre allo schema concettuale) la struttura del codice sorgente del software, contenente le firme delle funzioni e la struttura delle unità

Controllo di versione

Il controllo PULL sincronizza la cartella locale con il server, scaricando l’ultima versione presente sul server e aggiornando i file locali

Commit/push

Il comando Commit rende permanenti le modifiche e genera una nuova versione

Il comando Push pubblica sul server le commit presenti in locale

Application password

Github non consente alle applicazioni di autenticarsi con le credenziali dell’utente: è necessario utilizzare una application password

Software compilato/interpretato

L’esecutore può eseguire solamente codice binario, mentre lo sviluppatore utilizza linguaggi di programmazione alfanumerici

Il codice eseguito dall’esecutore viene chiamato codice macchina, il codice scritto dallo sviluppatore iene chiamato codice sorgente

Per poter essere eseguito, il codice sorgente deve essere trasformato in codice macchina,

per eseguire la trasformazione esistono 2 metodi:

* Compilazione
* Interpretazione

Software compilato

Il codice sorgente viene trasformato in linguaggio macchina prima dell’esecuzione, generando un file eseguibile

Quando si compila un file, viene generato un file da distribuire separato dai sorgenti

Questo meccanismo rende facile la protezione del codice sorgente ma il file eseguibile funziona solamente sull’esecutore per il quale è stato compilato, le differenze sono dovute all’hardware (in particolare la cpu) e al sistema operativo sui quali dovrà funzionare il software

Il file eseguibile viene elaborato direttamente dall’esecutore, che utilizza poche risorse per la gestione dell’esecuzione

Ogni strato dello schema può dialogare con l’hardware solo con quello confinante, quindi il software applicativo non può dialogare con l’hardware

Software interpretato

Il codice sorgente viene tradotto in codice macchina durante l’esecuzione e riga per riga

La traduzione viene effettuata da un software chiamato interprete

Il software interpretato è indipendente dall’esecutore hardware e dal sistema operativo, ma richiede la presenza dell’interprete, questo genera un aumento della complessità dell’esecuzione

L’interprete è un software, come tale deve essere caricato in ram e deve utilizzare la cpu per essere eseguito il codice interpretato è indipendente dall’hardware e dal sistema operativo

Ide/editor

Per scrivere codice sorgente si utilizzano editor di testo (tipo blocco note), però esistono sistemi specializzati

Un editor avanzato (es: visual studio code) è un editor con funzioni aggiuntive che semplificano la scrittura del codice (es: color code)

Un ide (Integrated Development Environment) (es: visual studio) è un sistema software di supporto per la scrittura del software, comprende un editor avanzato e altri strumenti di sviluppo, tipicamente: debugger e compilatore

Cicli

Nello sviluppo si possono utilizzare cicli di vario tipo

Un ciclo è un meccanismo di controllo che cnsente di eseguire un blocco di codice un certo numero di volte

Esistono 2 tipi di ciclo:

* Enumerativo
* Con guardia

Ciclo enumerativo

Un ciclo enumerativo specifica quante volte devono essere eseguite le istruzioni

In pseudocodice si indica con per

Es:

per 42 volt

scrivere “ciao mondo!”

vai a capo

fine per

l’istruzione di controllo può essere gestita da una variabile o da una costante

es:

numero\_stampe = 42

per numero\_stampe volte

scrivi “ciao mondo!”

vai a capo

fine per

ciclo con guardia

un ciclo con guardia (finchè) è un ciclo che ha un’istruzione (guardia) che viene valutata per decidere se continuare ad eseguire le istruzioni all’interno del ciclo o se fermare la loro esecuzione

la guardia può essere prima o dopo le istruzioni

es:

termine = 10

finchè termine è positivo

stampa termine

termine = termine -1

fine finchè

il risultato sarebbe: 10 9 8 7 6 5 4 3 2 1 0

quando si utilizza un ciclo con guardia, il numero di iterazioni non è noto a priori

fai

stampa termine

termine = termine -1

finchè termine è positivo

il risultato sarebbe: 10 9 8 7 6 5 4 3 2 1 0

cpu

la cpu (central processing unit) è il componente dell’esecutore che si occupa di eseguire le istruzioni macchina del codice eseguibile

ogni istruzione viene eseguita in sequenza: un software è un insieme ordinato di istruzioni macchina, la cpu esegue queste istruzioni in ordine, a partire dalla prima

la cpu esegue un’istruzione (macchina) ad ogni tick dell’esecutore

un tick è un impulso generato da un orologio detto clock

il clock genera impulsi con frequenza costante, la frequenza si misura in herz (Hz), dato che le frequenze sono molto alte, normalmente si utilizza il moltiplicatore g (10^9): 1 ghz = 1 000 000 000 hz, ossia un miliardo di istruzioni al secondo

software/task/thread

un software è un insieme di istruzioni, queste istruzioni possono essere suddivise in task

un task è un unità funzionale che gira in maniera indipendente dal software, consentendo l’esecuzione parallela di più istruzioni

un thread è un unità funzionale di un task, che gira in maniera indipendente dallo thread che la gestisce

esecuzione di un operazione

per eseguire un operazione algebrica (x = a + b) la cpu deve:

* Leggere l’istruzione dalla ram (nb: l’istruzione è +)
* Leggere i valori degli operandi dalla ram (n b: gli operandi sono a, b)
* Calcolare il risultato (a + b)
* Scrivere il risultato nella ram

Dato che la ram è più lenta della cpu (legge e scrive ogni n tick), la cpu contiene delle memorie veloci dette cache che tengono una copia degli ultimi dati utilizzati

Le cache sono veloci e piccole, consentono di accedere alla ram solo se il dato non è già in cache, velocizzando il processo di calcolo

Sviluppo di un’unità funzionale

Un software è un sistema che trasforma dati. Un’unità funzionale è analoga ad un software, quindi è un sistema che trasforma dati. Un software è un sistema costituito da una o più unità funzionali che operano in cascata o in parallelo

Le unità funzionali hanno nomi specifici che dipendono dal linguaggio di sviluppo, i più omuni sono:

* Function
* Subroutine
* procedure

Firma

La firma di un’unità funzionale è costituita dal nome della unità funzionale, dall’elenco dei tipi di dato previsti per gli input e tipi di dato previsti per gli output.

Es: calcolaIban(stringa, stringa, stringa) 🡪 stringa

Es: scrivere una funzione (unità funzionale) che riceve in input 2 numeri interi e produce in output la somma dei 2

Variabile

Una variabile è un “pezzo” di memoria del computer al quale viene associato un nome, il nome viene utilizzato per leggere e scrivere il “pezzo” di memoria associato.

Una funzione è un sistema che riceve input (variabili) e modifica il valore di queste variabili, eventualmente creandone di nuove

Le variabili possono avere qualsiasi nome, le regole più comuni sono:

* non possono essere interamente numerici
* non possono contenere spazi o caratteri speciali

Le raccomandazioni sono:

* utilizzare solo lettere, numeri, \_
* utilizzare nomi parlanti (non ad esempio x=a+b, ma risultato=numero\_1+ numero\_2)

salto condizionale (if)

per verificare una condizione si utilizza il costrutto SE ALLORA ALTRIMENTI

questo consente di eseguire un ramo di codice o un altro a seconda del risultato di un confronto

es: SE a>b

ALLORA risultato=a+b

ALTRIMENTI risultato=a-b

FINE SE

Il ramo ALTRIMENTI non è obbligatorio, può essere presente solo ALLORA

Esecutore hardware

Un esecutore è un sistema in grado di eseguire codice

Qualsiasi esecutore ha una struttura di base uguale, alla quale possono essere aggiunti elementi specifici

La cpu, il bus la ram e lo storage sono tutti indispensabili per l’esecuzione di un software

Storage

Lo storage è il sistema che memorizza in maniera permanente i dati

Esistono molti sistemi di storage, ogni tipologia ha caratteristiche uniche e caratteristiche comuni, quelle comuni sono:

* memorizzano dai in maniera permanente
* memorizzano dati in forma binaria

codifica binaria

I computer utilizzano internamente la codifica binaria, questo significa che ogni informazione (numero, lettera, colore) deve essere tradotta in linguaggio binario.

Il linguaggio binario viene normalmente rappresentato con 0 e 1

Esistono sistemi semplici per trasformare un numero decimale in un numero binario e viceversa

Per codificare lettere si utilizzano tabelle di transcodifica, la più nota è la tabella ASCII

Per comodità di rappresentazione, i dati vengono raggruppati a gruppi di 8

Una singola (0/1) prende il nome di bit (BInary digiT). Un gruppo di 8 bit prende il nome di byte

NB: la rappresentazione dei dati è fondamentale per la rappresentazione dei medesimi, memorizzare il valore 15 come numero intero o come testo comporta differenze nell’elaborazione perché non è possibile utilizzare testi per operazioni algebriche

Le variabili memorizzano dati di tipo diverso a seconda di come vengono utilizzate

Es: numero=15

Parola=”15”

Esecutore/storage

Un hard disk è una periferica di storage di tipo magnetico

Un ssd (solidi state drive) è una periferica di storage di tipo elettronico

Gli hard disk generano 2 ritardi:

* seek: tempo necessario allo spostamento della testina
* latency: tempo necessario alla rotazione

gli ssd sono perferiche elettroniche permanenti, non generano ritardi

lo storage può limitare la velocità di esecuzione di un software perché ha una velocità di lettura e scrittura limitata e ha una capacità limitata

ram

la memoria ram (random access memory) è una memoria digitale, elettronica, volatile

la cpu può elaborare esclusivamente i dati presenti in ram

per poter funzionare, il codice di un software deve essere interamente caricato in ram

se il codice del software è più grande della ram disponibile, non può essere eseguito, a meno di utilizzare una memoria virtuale

la memoria virtuale è una tecnica che consente di scaricare una parte della ram salvandola sullo storage, per liberarla rendendola disponibile per altri dati

bus

il bus è un sistema che trasferisce dati fra i vari componenti dell’esecutore

il sistema non può utilizzare il bus per più di un trasferimento per volta: contesa del bus

Array

Un array è una variabile strutturata che può contenere più di un valore

I valori contenuti nell’array sono identificati da un numero (indice)

Es:

nomi(1)=’mario’

nomi(2)=’luigi’

nomi(3)=’stefania’

gli array si utilizzano per semplificare l’approccio alla soluzione di problemi che riguardano insiemi di dati, in particolare spesso vengono utilizzati all’interno di cicli

gli array possono avere una dimensione fissa o variabile, per i nostri fini li considereremo sempre variabili

es:

leggere 10 elementi chiedendogli all’utente e stamparli

funzione leggi\_stampa()

i=1

‘leggo i valori e popolo l’array elementi

per 10 volte

leggi elementi(i)

i=i+1

fine per

i=1

‘stampo i valori presenti nell’array elementi

Per 10 volte

Stampa elementi(i)

I=i+1

Fine per

Fine

Es:

scrivere la funzione positivi che riceve 100 valori e stampa tutti quelli positivi

funzione positivi(valori)

‘ciclo sull’array e verifico se ogni elemento è positivo

I=1

Per 100 volte

Se valori (i)>0

‘se il valore è positivo lo stampo

Stampa valori(i)

Fine se

I=i+1

Fine per

fine

gli array possono avere un indice (array monodimensionali) o più di un indice (array multidimensionali)

valori (i,j)

scomposizione funzionale

un problema può essere risolto utilizzando funzioni che collaborano fra loro

per collaborare con un’altra, una funzione può chiamarne un’altra

es:

funzione leggi()

i=1

‘leggo i valori e popolo l’array elementi

per 10 volte

leggi risultato(i)

i=i+1

fine per

restituisci risultato

fine

funzione stampa(valori)

i=1

‘stampo i valori presenti nell’array elementi

per 10 volte

stampa elementi(i)

i=i+1

fine per

fine

funzione Leggi\_stampa()

elementi=leggi()

stampa(elementi)

fine

la scomposizione funzionale consente di:

* riutilizzare il codice (chiamando più volte la stessa funzione)
* centralizzare la manutenzione (correggendo una sola funzione)
* suddividere il lavoro (assegnando funzioni diverse a sviluppatori diversi)

lo sviluppatore riceve dal progettista le specifiche dell’unità funzionale che deve implementare (nome, input, output, elaborazioni), per implementarla può scomporre il problema in più unità funzionali, rispettando le specifiche

collaudo

il collaudo consiste nel verificare il corretto funzionamento di un software

per verificare il corretto funzionamento è necessario verificare che gli input previsti generino gli output attesi, inoltre il software deve terminare l’elaborazione in tempi ragionevoli

per verificare la correttezza degli output in funzione degli input è importante stilare un protocollo di test

un protocollo di test è un elenco di input e output che il software deve produrre, se gli output non sono quelli attesi, il software è errato

per il collaudo si possono utilizzare diversi strumenti di supporto:

* debugger, consente di visualizzare lo stato interno dell’esecutore durante l’esecuzione del software
* issue tracker, consente di tenere traccia degli errori presenti nel software e dello stato di avanzamento della loro correzione
* test unit, consentono di automatizzare i test del protocollo e di riportare automaticamente gli errori e le performance

debugger, issue tracker e test unit possono essere utilizzate sia nella fase di sviluppo che nella fase di collaudo, in particolare gli issue tracker fanno da raccordo fra le sue fasi

cpu

la cpu può eseguire un solo software per volta, ma questo crea problemi in un mondo nel quale si usano contemporaneamente più applicazioni

per simulare l’esecuzione contemporanea di più applicazioni su una sola cpu si sfrutta il meccanismo del time sharing

il time sharing consiste nell’assegna la cpu ad un’applicazione per volta per un certo numero di cicli di clock a rotazione tutte le applicazioni hanno accesso alla cpu

bilanciamento dell’esecutore

un esecutore è costituito (almeno) da cpu, ram e storage

le performance di un software sono date dalle performance peggiori dei tre elementi: basta che uno dei tre sia sovraccarico perché il software risulti lento

le applicazioni possono caricare molto la cpu, ma possono sovraccaricare anche altri componenti, si parla di:

* cpu bound, se il componente più utilizzato è la cpu
* memory bound, se il componente più utilizzato è la ram
* i/o bound, se il componente più utilizzato è lo storage

in molti casi lo sviluppatore può spostare il carico di lavoro da un elemento all’altro

complessità di un software

la complessità di un software è l’insieme delle risorse necessarie ad seguire il software in tempi ragionevoli

la complessità può essere di risorse o di calcolo

la complessità di risorse studia quali elementi dell’esecutore (cpu, ram, storage, gpu, scheda di rete, …) sono sovraccarichi

la complessità di calcolo (complessità computazionale) studia quante istruzioni devono essere eseguite dalla cpu per completare l’esecuzione del software

stack

quando viene chiamata una funzione, l’esecutore gestisce la memoria come una pila (stack)

funzione 1()

funzione 2()

fine

funzione 2()

funzione 3()

fine

funzione 3()

funzione 4()

fine

funzione 1()

funzione 2()

funzione 3()

funzione 4()

fine

quando viene chiamata una funzione, l’esecuzione del codice della funzione chiamante viene sospesa fino a quando la funzione chiamata non ha terminato la sua esecuzione

l’esecutore può eseguire solo il codice della funzione presente nello strato in cima allo stack

l’esecutore ha accesso esclusivamente alle variabili contenute nella funzione presente nello strato in cima allo stack

funzioni ricorsive

una funzione ricorsiva è una funzione che richiama se stessa

le funzioni ricorsive consentono di risolvere problemi complessi con poche righe di codice

le funzioni ricorsive utilizzano pesantemente lo stack

es:

scrivere la funzione Fattoriale che riceve un numero intero e restituisce il suo fattoriale

4!=4\*3\*2\*1=4\*3!

3!=3\*2\*1

Funzione fattoriale(valore)

Se valore=1

Risultato= 1

Altrimenti

Risultato= fattoriale(valore-1)\*valore

Fine se

Restituisci risultato

Fine

Debugger

Il debugger è uno strumento di supporto allo sviluppo

Consente di vedere lo stato interno dell’esecutore (stack, variabili, funzioni, …)

Per php si può utilizzare xdebug

Programmazione ad oggetti

la programmazione ad oggetti si basa sui concetti di classe e di oggetto

un oggetto è una variabile strutturata, specializzata per modellare una realtà specifica (es: automobile)

una classe è la definizione di tipo: definisce la struttura dei dati, le regole di gestione dei dati e i metodi per elaborarli

la classe integra tutta la gestione dei dati, semplificando l’utilizzo di tipi complessi

es: classe automobile, definisce una generica automobile memorizzando:

* colore
* tipo di alimentazione
* numero di porte

un oggetto di tipo automobile è una variabile, quando viene utilizzata devono essere valorizzati tutti i dati necessari

variabile mia\_auto di tipo automobile

mia\_auto 🡪colore=grigio

mia\_auto🡪tipo di alimentazione=gasolio

mia\_auto🡪numero di porte=5

mia\_auto🡪quantità carburante=50

la classe memorizza internamente i valori (proprietà) utilizzando variabili che non sono manipolabili dall’esterno, questo consente di fare controlli di validità anche complessi

la classe contiene delle funzioni (metodi) che possono essere utilizzati per gestire i dati al suo interno

es:

mia\_auto->accendi(), accende il motore, ma solo se non è già acceso e se c’è carburante

la programmazione ad oggetti garantisce:

* ereditarietà
* polimorfismo
* incapsulamento

Ciclo di vita del software

La distribuzione è la parte del ciclo di vita che si occupa di recapitare il software agli utenti, spesso è la fase nella quale si monetizza il software

Si possono utilizzare diversi metodi di distribuzione, i più comuni sono:

* distribuzione di supporti fisici (dvd, cd, ecc)
* download tramite web
* pubblicazione su store dedicati
* fornitura come servizio (SAAS)

la fase di manutenzione è quella nella quale si producono e si distribuiscono gli aggiornamenti per il software

programmazione ad oggetti/incapsulamento

una classe deve garantire l’incapsulamento, cioè deve proteggere il suo stato interno (i valori delle sue variabili)

la classe interagisce con l’esterno tramite proprietà (variabili) e metodi (funzioni) pubblici, ma può utilizzare proprietà e metodi privati, che non possono essere utilizzati dall’esterno

ereditarietà

una classe può ereditare da un’altra classe (classe base), diventando una classe derivata

quando una classe eredita, ottiene automaticamente tutte le proprietà e i metodi della classe base, ma può aggiungere nuove proprietà, nuovi metodi e ridefinire proprietà e metodi della classe base

polimorfismo

una classe può definire più metodi che abbiano lo stesso nome, purchè gli argomenti siano di tipo o di numero diverso (la firma è diversa)

il polimorfismo consente di:

* definire funzioni che hanno lo stesso nome a argomenti diversi (overload)
* ridefinire funzioni della classe base (override)

debugging

il debugger consente di vedere lo stato interno dell’esecutore e di eseguire codice in tempo reale

impostando un breakpoint è possibile interrompere l’esecuzione del codice da parte dell’esecutore e passare il controllo al debugger