INGEGNERIA DEL SOFTWARE

Alessandro Pioggia

19 gennaio 2022

Indice

Capitolo 1

Analisi dei requisiti

L'analisi dei requisiti è la fase che permette, attraverso la modellazione della realtà, di redigere la specifiche dei requisiti, documento che rappresenterà l'input per le successive fasi di progettazione. Per rendere possibile la creazione un coadiuvato è necessario, da parte dell'analista, intervistare il produttore, cercando di assimilare al meglio tutte le informazioni riguardanti il dominio applicativo.
In questa fase è necessario fare in modo che il documento sia chiaro, non ambiguo, accessibile e privo di contraddizioni.
L'analisi è incrementale e deve comunicare gli aspetti statici, dinamici e funzionali del progetto software.

1.1 Analisi orientata agli oggetti

In questa tipologia di analisi vengono curati principalmente gli aspetti statici, ovvero vengono definiti gli oggetti e le relazioni presenti fra essi. Definire gli oggetti significa individuare tutte le informazioni e proprietà ad essi connesse, esse tendono a rimanere invariate nel tempo, per questo viene definito un approccio **statico**.

1.2 Analisi funzionale

L'analisi funzionale si pone il problema di definire le specifiche prendendo come riferimento le funzioni, ovvero viene studiata solo ed esclusivamente la relazione presente fra dati in ingresso e dati in uscita. Vengono presi in considerazione i flussi informativi, che verranno poi modificati da processi.

1.3 Analisi orientata agli stati

L'analisi orientata agli stati analizza i vari stati evolutivi del prodotto, in funzione di ciò, vengono analizzati i comportamenti e successivamente redatta la documentazione delle specifiche dei requisiti.

1.4 Astrazione

I meccanismi di astrazione utilizzati sono:

- classificazione
- ereditarietà (is-a)
- aggregazione (part-of)
- associazione

1.5 Linguaggi utilizzati per la specifica dei requisiti

- informale (linguaggio naturale, molto contradditorio e poco chiaro, sconsigliato)
- semiformale (utilizzo di diagrammi secondo un preciso standard, che però è semplice ed intuitivo (e/r, dfd))
- formale (linguaggio tecnico, difficile da comprendere e inutilmente complicato, indipendentemente dal contesto)

Una volta definito il tipo di linguaggio da utilizzare è necessario effettuare una distinzione fra formalismi dichiarativi e operazionali. I primi definiscono il problema indicando le proprietà che esso deve avere, i secondi ne descrivono il comportamento, nella maggior parte dei casi attraverso un modello. I formalismi operazionali sono i più indicati in questo contesto, in quanto più comprensibili e modellabili.

Capitolo 2

Progettazione

Se la fase di analisi vuole sincerarsi sul "che cosa" sviluppare, nella parte di progettazione ci si chiede "come" svilupparlo, dunque si ha una sorta di ponte fra analisi e codifica.

2.1 Modalità

Nella progettazione si suddivide il problema iniziale in più sottoproblemi, il più possibile indipendenti fra loro, questo dà la possibilità di fare gestire il lavoro a più team, anche in parallelo. Una fase di progettazione ben realizzata consente un notevole risparmio di risorse, dal momento che un eventuale errore in fase di produzione o peggio, durante la manutenzione comporta un costo molto più elevato per l'azienda.

2.2 Approccio

L'approccio varia in base alla strategia adottata, ne conosciamo due:

- generale
- specifico

Il primo permette di mantenere una buona elasticità, può essere modellato anche nelle fasi successive. Il secondo invece è rigido, non contempla modifiche nelle fasi successive, questo semplifica il passaggio dalla progettazione alla codifica.

2.3 Il buon progettista

Il buon progettista è colui che ha una buona conoscenza di tutto ciò che riguarda lo sviluppo software, sa anticipare i cambiamenti ed ha un buon grado di esperienza. Inoltre, è colui che si pone come obiettivi della progettazione l'affidabilità, la modificabilità, la comprensibilità e la riusabilità del software.

Capitolo 3

Paradigma ad oggetti

La programmazione OO offre un nuovo e potente modello per scrivere programmi, gli oggetti sono scatole nere che mandano e ricevono messaggi. Questo sistema velocizza l'approccio software e migliora la riusabilità, la modificabilità e il mantenimento, richiedendo però un maggiore sforzo in fase di progettazione. I concetti fondamentali sono : oggetto, astrazione, classe, incapsulamento, ereditarietà, polimorfismo, late-binding e delegazione.

3.1 Oggetto

Un oggetto è una entità del dominio applicativo caratterizzata da un identificatore (unico), delle proprietà (attributi) e da un comportamento (definito da un insieme di operazioni). Gli oggetti possono essere aggregati fra loro per poter poi formare oggetti complessi.

3.2 Operazione

Ogni operazione è caratterizzata da un nome, dai parametri che prende come argomento e dal tipo di ritorno (signature). L'insieme di tutte le signature delle operazioni di un oggetto sono dette interfaccia dell'oggetto.

3.3 Classe

Una classe è la realizzazione di un tipo di dati astratto (tipo di dati astratto : insieme di oggetti simili, caratterizzati da dati e da un insieme di operazioni associate agli oggetti), cioè una implementazione per i metodi ad esso associati.

3.4 Incapsulamento

L'incapsulamento permette di **proteggere** dati e implementazione delle operazioni di un oggetto, attraverso una interfaccia pubblica. L'unico modo per modificare lo stato dell'oggetto (o ottenere informazioni riguardanti gli attributi) è attraverso l'utilizzo dei metodi presenti nell'interfaccia. Questo meccanismo favorisce la diminuzione della quantità di errori commessi (es:modifica di uno stato manualmente), una maggiore sicurezza, l'utilizzo (per utilizzare una classe è sufficiente conoscerne l'interfaccia pubblica) ed infine la modifica dell'implementazione di un metodo di una classe (non si ripercuote sull'applicazione).

3.5 Metodi

Un metodo cattura l'implementazione di una operazione, possono essere classificati in: costruttori, distruttori, accessori e trasformatori. Un metodo può essere o privato, o pubblico o protetto.

3.6 Ereditarietà

Il meccanismo di ereditarietà permette di basare la definizione dell'implementazione di una classe su quella di altre classi. Una classe può **ereditare** da una superclasse ed essere **estesa** da una sottoclasse. Le sottoclassi ereditano attributi e metodi della superclasse.
In certi contesti è ammessa l'ereditarietà multipla, fenomeno per il quale una sottoclasse eredita **contemporaneamente** da due superclassi.

3.7 Polimorfismo

Per polimorfismo si intende la capacità di assumere forme molteplici, si verifica quando:

- in una classe posso definire due metodi con stessa intestazione, a patto che abbiano una diversa signature (parametri in ingresso)
- posso ridefinire un metodo attraverso il meccanismo di override

3.8 Late binding

Il late binding, o istanziamento dinamico, permette a ciascun oggetto di rispondere a uno stesso messaggio in modo appropriato a seconda della classe da cui deriva.

3.9 Delegazione

Si parla di delegazione quando un oggetto A contiene un riferimento ad un oggetto B, in questo modo A può delegare delle funzioni alla classe a cui appartiene B. Questo è il meccanismo fondamentale per quanto riguarda l'implementazione dell'associazione fra classi.

Capitolo 4

Ingegneria del software

L'ingegneria del software è la disciplina che, attraverso un approccio sistematico (tecnico e preciso), si pone come obiettivo quello di gestire l'operabilità, la manutenzione e l'eventuale ritiro dal commercio del software. Questa disciplina, oltre all'aspetto tecnico, comprende un aspetto manageriale, il quale si occupa della gestione dei flussi, dei tempi e delle risorse dell'azienda.

4.1 Qualità del software

Un software che funzioni non è sufficiente, è necessario che sia di qualità, esse possono essere:

- interne (visibili solamente dagli sviluppatori)
- esterne (visibili dall'utente finale)
- di prodotto (guardano la qualità del prodotto)
- di processo (guardano la qualità del processo produttivo)

4.1.1 Qualità esterne, di prodotto e di processo

Robustezza (si comporta bene anche su aspetti non menzionati nelle specifiche di progetto)

4.1.2 Qualità esterne e di prodotto

Correttezza (funziona?), affidabilità (un software è affidabile se posso dipendere da esso), Efficienza (tempi e prestazioni), Facilità d'uso, portabilità e interoperabilità (utilizzo di software unito ad altri, ad esempio word e excel).

4.1.3 Qualità interne, di prodotto e di processo

Verificabilità (il sw è certificabile, si osserva la correttezza, performance, ecc.)

4.1.4 Qualità interne e di prodotto

Riusabilità e facilità di manutenzione (aperto a modifiche correttive, perfettive ed adattive).

4.1.5 Qualità esterne e di processo

Produttività, tempestività e trasparenza.

4.2 Software design

Il sw design rappresenta il processo che trasforma le specifiche utente in un insieme di specifiche utilizzabili dai programmatori, il risultato del processo di design è l'architettura software.

4.2.1 Principi di progettazione (good practices)

- **Formalità** (suggerisce l'uso di formalismi, ad esempio uml)
- **Anticipazione dei cambiamenti** (un sw robusto deve essere aperto ai cambiamenti, che siano noti o meno a priori. Le modifiche possono riguardare : hw, sw, algoritmi e dominio di applicazione)
- **Separazione degli argomenti** (suggerisce di spezzare il problema in sottoproblemi, in funzione del: tempo, livello di qualità (approccio incrementale), vista (in fase di analisi curare aspetti statici, dinamici e funzionali), livello di astrazione e dimensione)
- **Modularità** (divisione del sistema in moduli, un modulo è un componente di base di un sistema software, che permette di scomporre un sistema complesso in più componenti semplici. Tutti i servizi strettamente connessi devono appartenere allo stesso modulo e inoltre devono essere indipendenti fra loro, i programmatori devono poter operare su un modulo con una conoscenza minima del contenuto degli altri. Un modulo inoltre deve definire una interfaccia, la quale mostra le informazioni relative ad esso ma nasconde l'implementazione, si tratta di information hiding, dunque è suff. indicare una linea guida su come

utilizzare i servizi. Essi interagiscono fra loro, attraverso le dipendenze [uses], composizione [part-of] e in base al tempo.)

- **Astrazione**
- **Generalità** (Ogni volta che occorre risolvere un problema, si cerca di capire quale è il problema più generale che vi si nasconde dietro.)

4.3 Misurazione

La misurazione si occupa di prevedere e stimare tempi di consegna e qualità di lavorazione del software. Non è una misurazione esatta, in quanto è molto difficile quantificare, si tratta di una stima, eseguibile attraverso strumenti di diverso genere.

4.3.1 Fasi in cui è opportuno eseguire la stima

Scopi:
La stima può avvenire in diversi stadi di lavorazione, può essere utile per prevedere le caratteristiche che avrà il software in una fase successiva alla valutazione (utile per definire azioni correttive) oppure per stimare le caratteristiche nello stadio attuale, utile per capire il costo adeguato di un software.
Fasi
Bisogna inoltre definire in quale momento effettuare la misurazione, essa può avvenire in:

- fase di progettazione (serve a prevedere la manutenibilità e prevenire problemi)
- fase di collaudo/test (eseguire un confronto con le specifiche)
- fase dopo il rilascio in esercizio (serve a misurare l'impatto del software, in modo da assegnarli eventualmente un costo adeguato)

4.3.2 Costi

Prima di stimare i costi occorre effettuare una classificazione, dividendo le fonti di costo (costi per le attività generatrici) dai fattori di costo (hanno incidenza sul costo).

Fonti di costo

- Costo delle risorse per lo sviluppo del sw (costo diretto)
  - personale tecnico
  - personale di supporto (bidelli, donne delle pulizie, manutentori)
  - risorse informatiche
  - materiali di consumo
  - costi generali della struttura
- Costo per l'indisponibilità di un'applicazione (costo indiretto)

Fattori di costo

- Lines of code (LOC)
- bravura del team (un team bravo, anche se lo pago molto, mi fa risparmiare sul processo produttivo)
- complessità del programma
- stabilità dei requisiti (se cambiano idea spesso è difficile costruire un sw solido e in poco tempo)
- caratteristiche dell'ambiente di sviluppo

4.3.3 Dimensioni del software

Le dimensioni del software si possono stimare in funzione di due tipi di metriche:

- dimensionali (si basano sul numero di istruzioni del programma)
- funzionali (si basano sulle caratteristiche funzionali del programma)

4.3.4 Il metodo function points

Il metodo FP è empirico (basato sull'esperienza) e misura la dimensione di un sw in termini delle funzionalità offerte all'utente. Può essere utilizzato a partire dalla prima fase dello sviluppo per poi ripetere la misura nel caso le specifiche siano cambiate, è inoltre indipendente dall'ambiente tecnologico in cui si sviluppa il progetto.

Può essere utilizzato per:

- capire che beneficio un sw porterà alla mia organizzazione
- controllare che non si ladri alla vendita di un sw, c'è un fattore di proporzionalità (non posso vendere un sw di hello kitty a 3 milioni di euro)

4.3.5 Conteggio dei function points

In ordine:

- individuazione del tipo di conteggio
  - sviluppo software o manutenzione?
- individuazione dei confini
  - tutto il sistema o una sola componente?
- conteggio dei FP non pesati
  - funzioni di tipo dati
    - file interni logici (informazioni tenute all'interno dei confini dell'applicazione)
    - file esterni di interfaccia (dati a disposizione dell'applicazione ma mantenuti dentro i confini di un'altra applicazione)
  - funzioni di tipo transizione
    - input esterno (dati provenienti dall'esterno elaborati all'interno dei confini dell'applicazione, es: un form)
    - output esterno (manda dati di controllo fuori dai confini dell'applicazione, richiede almeno una formula matematica, es:una stampa)
    - interrogazioni esterne (manda dati o informazioni fuori dai confini dell'applicazione, non richiede una formula matematica)

- calcolo dei FP pesati, in funzione del fattore di aggiustamento(varia da 0.65 a 1.35, con incidenza max sul totale del 35%)

fattore di influenza $= 0.65 + (TDI \cdot 0.01)$
Il TDI è il total degree of influence e somma i gradi per diverse caratteristiche (es : riusabilità, comunicazione dati, prestazioni, facilità di installazione, complessità elaborativa, ecc.) ed assume valori fra 0 e 5.

4.4 Il numero ciclomatico

Mentre il calcolo dei FP è empirico, in questo contesto simo teorici e molto specifici, in particolare si quantifica la complessità del flusso di controllo (cicli, if, switch, ecc.). Il numero ciclomatico cattura solo in parte ciò che è la complessità del flusso di controllo. Il numero ciclomatico viene calcolato su un grafo fortemente connesso (ogni nodo può raggiungere indirettamente qualsiasi altro nodo). In termini di conti, il suo valore è determinabile attraverso:

- $e - n + 1$ (se il grafo è già fortemente connesso)
- $e - n + 2$ (se ho un grafo normale, lo posso trasformare in fortemente connesso collegando il nodo iniziale al nodo terminale)
- $e - n + 2p$ (se il programma ha procedure al suo interno, il numero ciclomatico dell'intero grafo è dalla somma dei numeri ciclomatici dei singoli grafi indipendenti. p è il numero di grafi (procedure) indipendenti)

4.5 Cocomo (COnstructive COst MOdel)

Si calcola una stima iniziale dei costi di sviluppo in base alla dimensione del software da produrre, poi la si migliora sulla base di un insieme di parametri.

4.5.1 Stima della dimensione del software

Questa stima la si opera contando il numero di linee di codice scritte (KDSI), il conto lo si può fare sfruttando i FP, assegnati diversamente in funzione del linguaggio.

4.5.2 Determinazione della classe del software

Ci sono differenti classi software, determinate in funzione del tempo e personale richiesto, ogni classe presenta un calcolo diverso per la determinazione del costo.

- Organic $M_{nom} = 3.2 \cdot KDSI$
- Semi-detatched $M_{nom} = 3 \cdot KDSI$
- Embedded $M_{nom} = 2.8 \cdot KDSI$

4.5.3 Applicazione degli estimatori di costo

Gli estimatori di costo c_i sono dei coefficienti che sono determinati in funzione di :

- Proprietà del prodotto (affidabilità, complessità, ecc.)
- Caratteristiche hardware (efficienza, memoria, ecc.)
- Caratteristiche del team (esperienza e capacità del team)
- Caratteristiche del progetto (Modernità del processo di sviluppo, ecc.)

La formula esatta è la seguente:

$M = M_{nom} \cdot \prod_{i=1}^{15} c_i$