Ingegneria del Software

Cristian Mercadante
UNIMORE | ULTIMO AGGIORNAMENTO: 14/06/2019

Introduzione

Scrivere un programma

Requisiti: affermazioni che definiscono e qualificano cosa deve fare il programma.

Vincoli di design: affermazioni che vincolano il modo in cui il software deve essere costruito ed implementato.

Ogni requisito ha un costo e il cliente potrebbe decidere che non ne ha veramente bisogno dopo averne compreso i relativi costi di implementazione. Ci sono requisiti da avere ed altri che sarebbe bello avere:

- Requisiti funzionali: ciò che deve fare il programma;
- Requisiti non funzionali: il modo in cui i requisiti funzionali devono essere implementati (es. performance, usabilità, manutenibilità, ecc.).

Alcuni vincoli di design sono spesso considerati requisiti non funzionali (es. linguaggio di programmazione, GUI, ecc.). I requisiti sono stabiliti dal cliente con l'aiuto dell'ingegnere del software.

Testing: deve essere fatto durante lo sviluppo del programma e dopo la sua ultimazione. Può essere di due tipi:

- Black box testing: eseguito dal cliente, ovvero chi utilizza veramente il software;
- White box testing: eseguito dal programmatore per testare specifici blocchi di codice.

È necessario anche fare una stima dei costi e dei tempi per la produzione di un software, sia per motivi organizzativi che per il costo finale del prodotto. È un compito difficile: il tempo stimato tende generalmente ad essere accurato dopo aver diviso il lavoro in parti, ma nonostante ciò il risultato non sarà accurato. Il problema è dato dal fatto che la stima precede il progetto e non si è a conoscenza di un gran numero di informazioni.

Riguardo l'implementazione, è necessario essere consistenti nella scelta dei nomi, librerie e altre convenzioni di programmazione.

Riassumendo:

1. Capire il problema: analisi dei requisiti funzionali e non funzionali.
2. Fare design, basandosi sui requisiti: organizzare le funzionalità tramite diagrammi, pensando ai vari vincoli e algoritmi da adottare.
3. Implementare: trasformare il design in codice.
4. Testare e validare il codice: controllare gli output in base a dei set di input predefiniti, debuggare, correggere e ritestare.

Altri punti da considerare sono:

- Quanto tempo occorre per terminare il lavoro?
- Quanto tempo e persone sono state necessarie?
- L'esito risolve il problema?

Costruire un sistema

Esistono problemi correlati alla costruzione di un sistema che contiene molteplici componenti: l'aumento del loro numero causa una maggiore complessità e richiede una maggiore analisi. La produzione di software diventa sempre più complessa e agli ingegneri del software è richiesto di risolvere problemi semplici e complessi e comprendere le differenze fra essi. I problemi complessi hanno:

- Problemi di ampiezza: funzioni più complicate, funzionalità specifiche, interfacce verso sistemi esterni, utilizzo simultaneo da parte di più utenti, differenti tipi di dati e strutture dati, ecc.
- Problemi di profondità: condivisione dei dati e collegamenti fra di essi (gerarchie, loop, ecc.).

Cambiare l'approccio verso problemi complessi può essere difficile, ma la politica più diffusa è "dividi e conquista": occorre dividere il problema principale in moduli meno complessi e risolverli.

Scegliere il linguaggio di programmazione può diventare un problema nella scrittura di software grandi e complessi. Infatti, molti programmatori collaborano nella scrittura del programma e devono concordare sul linguaggio e strumenti di sviluppo. A questi si aggiungono database, network, middleware ecc.

Per sviluppare il codice singolarmente occorre comprendere il problema ed i requisiti, strutturare una soluzione, implementarla e testarla con un utente. Ma sorgono spesso problemi riguardo la scarsa documentazione, non necessaria in quanto non vi sono altri programmatori. Software complessi richiedono lo sviluppo in team e pertanto la coordinazione di un gruppo di persone. Ciò significa dividere i compiti e l'ordine delle operazioni, decidere gli input e gli output dei singoli pezzi, con precondizioni e post-condizioni, ecc.

I cinque blocchi principali in cui si divide lo sviluppo del software sono: Analisi dei requisiti, Design, Codifica, Test, Supporto e Manutenzione

Questi processi devono essere indipendenti, ma se sono coinvolte molte persone, allora bisogna avere un'idea chiara della loro sequenza, sovrapposizione e condizioni di inizio di ognuna. Infatti, è difficile stabilire il momento in cui passare da un modulo all'altro.

Coordinare il lavoro per progetti facili fra poche persone è semplice, perché i requisiti sono pochi. Ma per progetti complessi che coinvolgono team di programmatori è assai difficile capire i requisiti e stimare tempo e costo della produzione: questo molto spesso porta al fallimento del software. Infatti, è assai difficile comunicare sia col cliente che fra i programmatori e la probabilità di un errore di comunicazione è alta e questo porta al ritardo della consegna del progetto.

Al livello della programmazione il design è tradotto in codice. Ciò necessita alcune precisazioni sul layout, linguaggio, interfacce, accesso ai dati, convenzioni varie, gestione degli errori, documentazione, ecc.

Ciascun modulo dovrà poi essere testato e ciascun problema dovrà essere corretto, reinserito e ritestato. Bisogna tenere conto che chi utilizzerà il software non sarà chi l'ha programmato. Il sistema dovrà infine essere testato per intero.

Una volta rilasciato il software, è necessario fornire assistenza e training al cliente interessato. Ciò richiede abilità nella comunicazione e presentazione. E per fornire una migliore assistenza è necessario preparare anche il personale al supporto: chi risponderà al cliente e chi correggerà i problemi. Se è prevista una lunga vita del software il personale di supporto coinciderà col team di sviluppo o sarà grande quanto quello. Infine, tale software dovrà essere correttamente documentato e fornire manuali d'uso per il futuro.

L'ingegneria del software

L'ingegneria del software si pone l'obiettivo di imporre "disciplina" nella programmazione e di fare qualcosa in più rispetto alla scrittura di codice, ovvero modellare, analizzare e migliorare in questo ambito. Molti degli errori di programmazione non risultano infatti risolti e vengono spesso ripetuti. Inoltre, la maggior parte dei progetti si rivela un fallimento, perché incompleti, fuori tempo di consegna o fuori budget.

Fattori di successo per un software sono: il coinvolgimento dell'utente, il supporto ai piani alti del committente, una chiara analisi dei requisiti e una adeguata pianificazione.

Fattori di insuccesso sono: lo scarso contributo dell'utente, requisiti non chiari o il cambiamento di questi, la mancanza di risorse. Gli errori possono essere di codifica, di design, di documentazione, di requisiti o di cattiva correzione degli errori. Il coinvolgimento dell'utente e i suoi requisiti sono un punto chiave per il successo: le probabilità di successo sono minime se non si comprende cosa deve essere sviluppato. Gli errori di requisiti si propagano su tutte le fasi successive, con un costo assai elevato, e spesso si riscontrano solo dopo la release del software. D'altra parte, un prototipo del progetto può essere conveniente, ma costoso. Pertanto, è necessaria una buona coordinazione per evitare il fallimento del progetto, fra software house e committente.

Il termine "Ingegneria del software" fu introdotto dalla NATO nel 1968, quando aumentò la potenza di calcolo e capacità di memoria e il software risultava essere mal funzionante: erano necessarie alcune formalizzazioni. L'ingegneria del software è la costruzione multi-persona di un programma multifunzione. È una disciplina incentrata sullo sviluppo cost-effective di software di alta qualità. Il software è astratto ed intangibile. L'ingegneria del software si occupa di tutti gli aspetti della produzione, dall'analisi dei requisiti al servizio post-vendita.

Il processo di sviluppo e supporto del software richiede molte attività distinte che devono essere svolte da persone diverse, in una certa sequenza: infatti se gli ingegneri del software agissero secondo la loro volontà, ognuno avrebbe un punto di vista e un modo di lavorare diverso dagli altri, e l'inconsistenza causa fallimenti. L'obiettivo è quello di coordinare e controllare i vari lavori verso

un fine comune. Infatti, per progetti di grandi dimensioni è necessario stabilire e chiarire i requisiti, testare le funzionalità, fare un buon design, utilizzare tool esistenti e coinvolgere le persone.

Diversi modelli di progettazione sono:

- **Waterfall:** i task si susseguono uno dopo l'altro, con l'output di uno come input del successivo. Ha di buono la semplicità e la tracciabilità, ma comunica poco col cliente (all'inizio e alla fine).
- **Incremental waterfall:** richiede il waterfall su ogni componente, in modo che si possa al termine collegare i pezzi e testare il sistema per intero.
- **Multiple-release incremental:** richiede lo sviluppo di release che comprendono diversi requisiti, che vengono implementati sulla base della release precedente.

Tutti questi modelli utilizzano l'approccio "dividi e conquista".

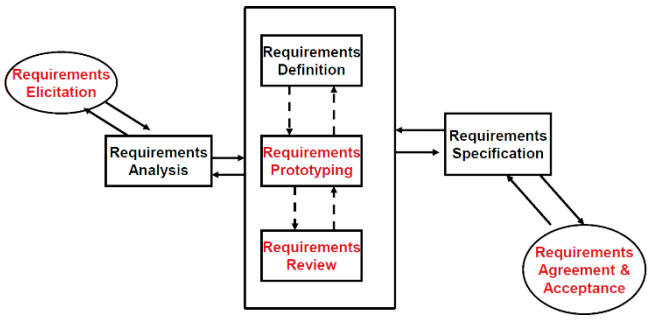
Specifica dei requisiti

Ingegneria dei requisiti

I requisiti sono un elenco di affermazioni che descrivono ciò di cui il cliente ha bisogno e desidera. Questi devono essere chiari e compresi dagli ingegneri del software per sviluppare il progetto. L'analisi dei requisiti può rivelarsi un fattore di insuccesso o di successo nello sviluppo del software. Pertanto, con un'accurata analisi ed un adeguato coinvolgimento del cliente si può raggiungere l'obiettivo con successo, al di là dell'implementazione.

Esistono diverse fasi comprese nella specifica dei requisiti, che possono essere incluse o approfondite a livelli diversi a seconda della tipologia di software: elicitazione, documentazione, specifica, prototipazione, analisi, validazione e accettazione.

Prima di iniziare l'analisi dei requisiti è necessario pianificare le risorse, la metodologia e il tempo necessario da spendere in questa fase cruciale. Il piano deve essere visionato ed accettato da tutti gli attori. Infatti, in requisiti non devono essere oggetto di immaginazione del designer o dello sviluppatore, ma il cliente deve essere incluso, perché i requisiti rappresentano i suoi bisogni. Anche il management deve essere incluso, perché le sue risorse sono necessarie per svolgere le attività. Pertanto, la tabella di marcia deve essere accordata fra tutti i partecipanti, fra cui gli ingegneri del software, che devono mostrarsi flessibili e aperti.



Dopo l'organizzazione preliminare, ingegneri esperti nell'analisi dei requisiti devono raccogliere i requisiti. Sono necessarie ottima capacità di comunicazione, conoscenze industriali e tecniche: sono pertanto persone preparate a questo compito.

Successivamente il piano deve essere approvato da tutti i partecipanti. Ogni modifica, cambiamento, richiesta deve essere controllata e gestita per evitare che il progetto cresca senza il controllo di nessuno (problema dello **scope-creeping**).

Non spendere tempo nella specifica dei requisiti può rivelarsi dannoso e costoso, perché potrebbero esserci requisiti con scarsa documentazione su cui basare test, training e supporto, oppure requisiti non accordati. Tuttavia, prototipazione e documentazione risultano essere operazioni molto costose.

L'analisi dei requisiti è effettuata da persone competenti con esperienza in coding, design e testing, capacità comunicative e di interpretazione. Il cliente inoltre è sempre incluso nello sviluppo del software per chiarire continuamente i requisiti, ma allo stesso tempo risulta difficile includere l'utente per lo sviluppo di grandi progetti software (per motivi di costo).

Esistono due livelli di **elicitazione** dei requisiti: ad alto livello ci si impegna a comprendere il motivo per cui si realizza il software, il fine ultimo; a basso livello si comunica con l'utente, colui che utilizzerà il software, per capire le sue necessità. L'elicitazione deve essere preparata dall'analista e può avvenire tramite colloquio o in forma scritta, tramite questionario: questo potrebbe essere molto vincolante, in quanto l'utente non è libero di esprimersi. È pertanto raccomandato un colloquio orale e diretto. L'analista deve anche raccogliere documentazione esistente da parte dell'azienda per comprendere al meglio l'obiettivo. Per raccogliere requisiti di alto livello, gli analisti devono recarsi ai piani alti dell'azienda committente, che stabilisce:

- **Obiettivi e necessità:** solitamente un problema di business, non un problema tecnico. Stabilisce i limiti e l'obiettivo del progetto software;
- **Vincoli principali:** solitamente vincoli di budget (importante per poter dare la priorità ad un requisito, piuttosto che ad un altro: contribuisce a differenziare ciò che si necessita e ciò che sarebbe bello avere) oppure vincoli di tempo;
- **Funzionalità principali:** servono come linee guida per la raccolta dei requisiti nelle varie aree di business;
- **Fattori di successo:** devono essere accordati da entrambe le parti.

Per raccogliere requisiti di basso livello è necessario che analisti tecnici contattino gli utenti per poter risolvere problemi specifici. Tenere conto che se esiste già un sistema software utilizzato dal cliente, questo baserà le proprie necessità sul software precedente. Requisiti di questo tipo sonno essere:

- Funzionalità individuali: sono le più naturali e quelle da cui si parte per l'elicitazione. Spesso è utile conoscere come l'utente svolge il proprio lavoro;
- Business workflow;
- Dati e formati di dati: per questioni di input e output del sistema;
- Interfacce utente: come vengono rappresentati input e output del software. Spesso può essere utile prototipare le interfacce e chiedere all'utente il suo parere;
- Interfacce software: trasferimento di controllo, trasferimento di dati, ricezione di risposte, errori, ecc.;
- Affidabilità, performance, sicurezza e adattabilità, trasportabilità, manutenibilità.

I **requisiti** si classificano in **funzionali**, ovvero le interazioni fra il sistema e l'ambiente di implementazione, cioè l'utente o qualsiasi altro sistema nello spazio di lavoro; oppure **non funzionali**, ovvero aspetti del sistema che non sono direttamente correlato alle funzionalità o al comportamento del sistema.

Il modello FURPS+ contiene alcuni parametri riguardo la qualità del software: funzionalità, usabilità, affidabilità, performance, supporto, e altro (design, implementazione, ecc.).

Dopo l'elicitazione dei requisiti è necessario organizzare il materiale e le richieste tramite categorie e priorità. Ci sono diversi modi per categorizzare i requisiti, ma l'importante è la consistenza e la completezza di essi. Ad esempio, possiamo mettere un prefisso indicante l'ambito del requisito, seguito da un numero che identifica una tipologia di attività e altri numeri per scendere del dettaglio.
Ad esempio: IF-1.3 = requisito relativo ad una funzionalità individuale, relativa all'attività 1, numero 3.

Esiste un altro modo per organizzare i requisiti detto VORD: Viewpoint-Oriented Requirements Definition che si basa sul fatto che i requisiti non sono visti allo stesso modo da tutti gli attori, che chi paga il software ha diversi punti di vista rispetto a chi lo utilizzerà, e che ciascun utente darà enfasi a un componente, piuttosto che ad un altro. La metodologia VORD si divide in tre fasi: identificare i punti di vista, documentarli e mapparli nel sistema.

Occorre precisare che spesso alcuni requisiti non possono essere implementati per limitatezza di risorse, tempo o impossibilità tecniche. Pertanto, è necessario dar delle priorità ad alcune funzionalità: le altre potranno essere rilasciate in release successive. Il criterio di priorità è discutibile e può basarsi sulla domanda di un requisito, necessità immediate, ecc. Per far ciò entrano in scena esperti del mercato e dell'industria.

Un ultimo approccio all'organizzazione dei requisiti può essere l'AHP: Analytical Herarchy Process, che usa più rigore. Ogni requisito è confrontato con un altro e un valore di priorità è assegnato ad uno sull'altro. Il requisito con punteggio più alto sarà il più importante.

I requisiti devono essere tracciabili, ovvero deve essere possibile risalire, dopo lo sviluppo, e verificare che tutti i requisiti siano stati implementati, testati, impacchettati e consegnati. Esistono quattro tipi di **tracciabilità**:

- Backward from traceability: collega il requisito al documento o alla persona che l'ha creato;
- Forward from traceability: collega il requisito al design e all'implementazione;
- Backward to traceability: collega il design e l'implementazione al requisito;
- Forward to traceability: collega il documento al requisito.

È possibile che alcuni requisiti abbiano prerequisiti e/o post-requisiti. In tal caso si generano matrici di requisiti.

La definizione, la prototipazione e la rivisitazione dei requisiti sono tre attività differenti, ma in pratica si sovrappongono. La definizione avviene attraverso una tabella requirementNumber-input-process-output, in cui in ogni riga è inserito un requisito con i corrispettivi input, output e processi. Altre notazioni assai utilizzate sono i diagrammi UML, che di fatto sono forme di

prototipazione, che mostrano le funzionalità principali del software al cliente e al team di sviluppo. Le interfacce grafiche si possono prototipare tramite disegni o interfacce reali, ma vuote. Infine, la review dei requisiti è un'attività fondamentale per evitare che un singolo requisito sbagliato provochi una serie di errori. Ogni modifica o correzione deve passare per la definizione dei requisiti. Nonostante sia meglio non essere troppo burocratici, è meglio documentare ogni cambiamento in ogni fase della progettazione.

Finito ciò si scrive e firma il documento **SRS (Software Requirement Specification)**, più o meno dettagliato in base a diversi fattori, come ad esempio l'ampiezza del progetto oppure l'esperienza dei programmatori. Una volta firmato si chiude la fase dell'analisi dei requisiti e ogni cambiamento futuro deve essere controllato per evitare danni o addirittura il fallimento del progetto.

Design

Architettura e metodologia

Una volta compresi i requisiti di progetto, comincia la trasformazione dei requisiti in design. Il design si occupa di come il software debba essere strutturato, ovvero quali componenti e come debbano essere collegati fra di loro. Per grandi sistemi si divide in due parti: quella architetturale è di livello più alto e si occupa di definire i componenti principali, le loro proprietà e relazioni fra di essi, mentre la parte di design vera e propria va a descrivere in modo più dettagliato i componenti e i requisiti funzionali. Tradizionalmente l'ideale sarebbe attuare design fino al più basso livello, nel modo più dettagliato possibile, in modo che il design possa essere semplicemente tradotto in codice. Con alcune metodologie dette Agile si lascia un ruolo importante al programmatore, a cui è affidato il design dettagliato, ovvero di basso livello. Per specificare il design si usa o una rappresentazione grafica oppure linguaggi appositi come l'**UML (Unified Modeling Language)**.

L'**architettura software** di un programma è la "struttura delle strutture del sistema", che comprende elementi software, proprietà esternamente visibili di tali elementi e le relazioni fra di essi. Tutti i sistemi hanno un'architettura. Un software può essere visto sotto diverse prospettive:

- logica: rappresenta la decomposizione orientata agli oggetti di un sistema, ovvero le classi e le relazioni fra di essi. Rappresentata dal Class Diagram UML;
- processuale: rappresenta il comportamento dei componenti durante i processi e come comunicano fra di loro;
- della decomposizione in sottosistemi: rappresenta i blocchi logici e le loro relazioni;
- dell'architettura fisica: rappresenta la corrispondenza fra Software e Hardware. Si presume un sistema che lavora su una rete di computer.

Sono valide anche altre prospettive (che non approfondisco).

Gli ingegneri del software comparando sistemi architetturali esistenti e descrivendoli in termini di similarità e differenze hanno fornito una **conoscenza meta-architetturale**: modelli di architetture valide e funzionanti. Si suddivide in stili, tattiche e architetture di riferimento. Lo scopo di questa conoscenza è fornire un punto d'inizio per una particolare architettura software, risparmiando lavoro e fornendo una guida per la soluzione del problema. È un efficiente meccanismo per la progettazione di alto livello.

Gli stili:

- **architettura Pipe & Filters:** è un sistema di flusso di dati che non richiede molto lavoro in termini di implementazione, ma ha diverse funzionalità.
    - pipe: collegamenti unidirezionali d'informazione;
    - filtri: componenti autonomi, indipendenti da altri filtri, creati per uno specifico lavoro (produttori, trasformatori, tester, consumer, ecc.).

  Le pipe quindi collegano filtri che elaborano i dati in ingresso e li conducono all'uscita. I vantaggi di questo sistema riguardano la semplicità e la riutilizzabilità dei componenti. Il sistema sarà inoltre facile da manutenere ed aggiornare. Gli svantaggi sono collegati alla difficoltà nel debug del sistema nel complesso. Inoltre, questo sistema è adatto solo a sistemi di processamento in batch, con scarsa interazione con l'utente.

- **architettura Client-Server:** Il client è il processo che richiede un servizio o un'informazione. Il server è il processo che accetta le richieste del client, processa, raccoglie le informazioni e genera ed invia la soluzione/risposta alle richieste del client, per poi ricevere altre richieste. Il mezzo di comunicazione definisce il modo in cui le due parti comunicano. Quest'architettura è influenzata dai cambiamenti dell'hardware e dal costo: infatti tramite terminali meno potenti è