sweki

Giorgio Giuffrè

Indice

1 Introduzione all'ingegneria del software

1.1 Cos'è l'ingegneria del software?

Mentre la scienza è un insieme di princìpi interpretativi della realtà, l'**ingegneria** è un'*applicazione* della scienza alla realtà (magari in modo originale, ingegnoso). Quindi l'ingegneria non crea nuova conoscenza, come invece fa la scienza. Il termine **software**, invece, si compone di due parti: ware significa "cosa inerte", oggetto, e soft aggiunge una connotazione astratta, dato che il software non è altro che una costruzione del pensiero. L'ingegneria del software è l'applicazione dell'ingegneria al software, cioè l'applicazione di un approccio **sistematico**, **disciplinato** e **quantificabile** allo sviluppo, al funzionamento e al mantenimento del software. Sistematico nel senso che abbraccia un metodo; disciplinato poiché segue le norme, anzi la best practice; quantificabile poiché si deve sapere a priori quanto si consumerà (in risorse). Le ingegnerie dell'hardware, del software e dei processi sono specializzazioni dell'ingegneria dei sistemi.

1.2 Cosa non è l'ingegneria del software?

L'ingegneria del software (d'ora in avanti "IS") non è un ramo dell'informatica; è una disciplina ingegneristica che fa affidamento solo in parte sull'informatica, allo stesso modo in cui l'ingegneria meccanica fa affidamento sulla fisica.

1.3 Software

Per software s'intende il programma con la sua documentazione. A causa dell'assenza di vincoli fisici, il software ha un potenziale illimitato; tuttavia, per lo stesso motivo, esso può velocemente diventare complesso, difficile da capire e costoso da cambiare. Difatti, l'IS si confronta con progetti così impegnativi da richiedere necessariamente un lavoro di gruppo.

1.4 Efficacia ed efficienza

I prodotti software devono presentare due qualità: **efficacia** (conformità alle attese) ed **efficienza** (contenimento dei costi per raggiungere un obiettivo). L'efficienza ha a che fare soprattutto con i bisogni e le risorse*; l'efficacia, invece, riguarda il prodotto finale. Tuttavia i due termini confliggono: l'efficacia sottintende il verbo "fare", mentre la massima efficienza è proprio il non fare nulla! Bisogna dunque badare a trovare un compromesso, possibilmente in modo sistematico. Ma sappiamo che esiste un ciclo virtuoso tra sistematizzazione ed esperienza. Ecco allora che si ricorre alla best practice, la prassi che per esperienza e per studio abbia mostrato di garantire i migliori risultati in circostanze note e specifiche.

1.5 Come studiare

Due sono i tipi di libri su cui si può studiare: i **libri teorici** espongono i princìpi ma non l'esperienza concreta; i **libri esperienziali** fanno l'opposto e servono soprattutto per risolvere dubbi. ingegneria del software va capita studiando con entrambi questi tipi di libri e, soprattutto, partecipando al **progetto didattico**. Inoltre, torna molto utile compilare un proprio **glossario** dei termini.

2 Processi software

2.1 Definizione

In ingegneria, secondo l'ISO, un **processo** è un insieme di attività correlate e coese che trasformano ingressi in uscite, consumando risorse nel farlo. In particolare, un processo software (d'ora in avanti solo "processo") porta ad un *prodotto* software.

2.2 Anatomia

Ogni processo si divide in più **attività**. Ogni attività si divide in **compiti**. Ogni compito si può svolgere usando qualche tecnica, cioè una sorta di ricetta applicata agli strumenti disponibili. Per strumento s'intende un insieme di concetti e di metodi, con delle tecnologie di supporto.

2.3 Processi software, aziende e progetti

Distinguiamo le seguenti tre categorie principali di processi:

- processo standard — riferimento di base generico usato come stile comune per lo svolgimento delle funzioni aziendali, pensato per una collettività di casi afferenti ad un certo dominio applicativo (quindi una sorta di template);
- processo definito — specializzazione del processo standard necessaria per adattarlo ad esigenze specifiche di progetto;
- processo di progetto — istanza di un processo definito che utilizza risorse aziendali per raggiungere obbiettivi prefissati (il processo viene calato nella realtà aziendale).

L'organizzazione aziendale si struttura verticalmente in settori (orientati alla specializzazione) e orizzontalmente in processi (che abbracciano più settori specializzati).

2.4 ISO/IEC 12207

Lo standard ISO/IEC 12207 è il più noto standard di processo. Esso divide i processi in tre famiglie.

- Processi primari:
  - acquisizione (gestione dei propri sotto-fornitori*);
  - fornitura (gestione dei rapporti con il cliente — controparte dell'acquisizione);
  - sviluppo — affrontato con approccio costruttivo, non correttivo; svolto anche tramite appalto esterno; *non* solo programmazione (che tra l'altro va affiancata dal testing)!
  - gestione operativa (installazione ed erogazione dei prodotti);
  - manutenzione (correzione, adattamento, evoluzione).

- Processi di supporto (delle specie di "sottoprocedure"):
  - documentazione;
  - accertamento della qualità;
  - gestione delle versioni e delle configurazioni;
  - qualifica: verifica e validazione ("V&V");
  - revisioni congiunte con il cliente;
  - verifiche ispettive interne;
  - risoluzione dei problemi.
- Processi organizzativi (l'"ambiente" del sistema):
  - gestione dei processi;
  - gestione delle infrastrutture;
  - miglioramento del processo;
  - formazione del personale;
  - riuso: l'IS è un insieme di best practices che assembla cose già esistenti, più che crearle ex novo.

Un ingegnere del software sa fare qualsiasi processo tra i suddetti.

2.5 Attività fondamentali di un processo software

Ogni processo dovrebbe includere le seguenti attività:

- definizione delle specifiche del software;
- progetto e implementazione del software;
- validazione del software;
- evoluzione del software.

2.6 Organizzazione di processo

Per essere disciplinati si ha bisogno di una forma di standardizzazione, per "tenere alta" la qualità di un lavoro ripetitivo che rischia continuamente di degradare. Ecco perché un buon processo si auto-migliora, in modo continuo, secondo lo **schema PDCA** (ciclo di Deming):

1. Plan — individuare obiettivi di miglioramento;
2. Do — eseguire ciò che si è pianificato;
3. Check — verificare se ha funzionato;
4. Act — agire per correggersi.

3 Ciclo di vita di un progetto

3.1 Definizione

Caratteristico di un prodotto di IS è il suo ciclo di vita, cioè l'insieme degli **stati** che il prodotto assume dal concepimento al ritiro*. Senza di esso non esisterebbe la figura dell'ingegnere del software. Conviene vedere il ciclo di vita come una macchina a stati, in cui gli stati sono il grado di maturazione del prodotto e gli archi rappresentano attività (suddivise in processi) che servono a far avanzare il prodotto nel suo grado di maturazione. La durata temporale entro uno stato del ciclo di vita e un altro è detta **fase**. Misura del successo di un prodotto è un ciclo di vita lungo, speso per lo più in manutenzione, magari con un buon feedback da parte degli utenti. Distinguiamo tre tipi di manutenzione:

- correttiva :(per correggere difetti;
- di adattamento :| per adattare il sistema a variazioni di requisiti;
- evolutiva ;) per aggiungere funzionalità al sistema.

Esempio di manutenzione evolutiva è Firefox.

3.2 Modelli di ciclo di vita

La qualità più difficile da soddisfare in IS è la quantificabilità; per questo nascono i **modelli** di ciclo di vita. Esistono diversi possibili cicli di vita, che si distinguono non per numero o significato degli stati, bensì per le transizioni tra essi e le loro regole di attivazione. Alcuni modelli di ciclo di vita sono:

- sequenziale — tipo catena di montaggio;
- incrementale — realizzazione in più passi, con numero crescente di funzionalità;
- evolutivo — con ripetute iterazioni interne;
- a spirale — contesto allargato e modello astratto;
- agile — dinamico, a cicli iterativi e incrementali.

In genere un modello del ciclo di vita di un *prodotto** include un modello del ciclo di vita dello *sviluppo**, eventualmente diverso dal primo. È bene tenere a mente che i vari modelli, per quanto differiscano tra di loro in questo o in quel dettaglio, si possono dividere in due grandi famiglie: quelli sequenziali e quelli iterativi; i modelli incrementale, evolutivo, a spirale e agile sono tutti esempi di modelli iterativi.

3.3 Il modello sequenziale

Nel 1970, grazie a Winston Royce, venne ideato il modello sequenziale (o a cascata), ispirato alle catene di montaggio. Il modello originale prevede che non si possa mai essere in due stati diversi allo stesso tempo e che non si possa tornare ad uno stato precedente. Il passaggio da una fase alla successiva è basato

sulla documentazione poiché ogni fase produce documenti che la concretizzano e devono essere approvati per il passaggio alla fase successiva. Ha il pregio di individuare fasi distinte e ordinate nelle quali decomporre il progetto. Suo difetto principale è l'eccessiva **rigidità**. Tuttavia questo approccio può funzionare se il cliente è consapevole (e abbastanza sicuro) di ciò che vuole, pur tenendo conto che il modello genera software vero e proprio molto tardi nel ciclo di vita. Allora, si pensò di creare un modello "ibrido", introducendo la possibilità di tornare ad uno stato precedente. Tuttavia risalire la cascata fa risalire il progetto nel tempo e genera iterazioni, non incrementi.

3.4 Il modello incrementale

Per superare le difficoltà del modello sequenziale ibrido, nacque il modello incrementale: in esso, i cicli non sono più iterazioni ma **incrementi** — con l'eccezione dell'analisi e della progettazione, che si affrontano all'inizio e non vengono ripetute. Il modello prevede rilasci multipli e successivi; ciascuno realizza un incremento di funzionalità, approssimando sempre meglio la soluzione. Un grande vantaggio è che le funzionalità più importanti vengono affrontante all'inizio. Questo modello è meno idealista ma più gentile.

3.5 Il modello evolutivo

Il modello evolutivo, che è incrementale, prevede che gli incrementi successivi siano versioni (prototipi) usabili dal cliente. Più versioni posso essere mantenute in parallelo e ogni fase ammette iterazioni multiple.

3.6 Il modello a spirale

Nel 1988 Barry Boehm propose il modello a spirale, che introduce il concetto di "rischio di progetto" (cercando di contenere tali rischi). Lo sviluppo procede a cicli sempre più lenti; difatti i cicli esterni sono così lenti che possono aderire, ognuno, ad un altro modello di ciclo di vita. Ad ogni ciclo si analizzano i rischi e si compiono simulazioni. Misura del successo di un progetto è il diametro della spirale. Questo modello viene usato solo da chi intraprende progetti sperimentali, che nessuno ha mai realizzato, e richiede forte forte interazione tra committente e fornitore. Un ciclo si articola generalmente in quattro fasi:

1. definizione degli obiettivi;
2. analisi dei rischi;
3. sviluppo e validazione;
4. pianificazione della successiva iterazione.

3.7 Il modello a componenti

Più pragmatico è il modello a componenti, che prevede l'integrazione di componenti già implementati. L'idea nasce dal fatto che molto di quel che ci serve fare è già stato fatto e molto di quel che faremo ci potrà servire ancora.

3.8 I metodi agili

I metodi agili nascono alla fine degli anni '90 come reazione all'eccessiva rigidità dei modelli allora in vigore. Si basano su quattro princìpi:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

4 Gestione di progetto

4.1 Progetto

Un progetto è un insieme di **compiti** da svolgere a fronte di un assignement. Alcune **attività** (intese come insiemi di compiti) possono essere svolte individualmente ma il progetto è sempre collaborativo. Tutti i compiti sono pianificati dall'inizio alla fine, secondo specifici obiettivi e vincoli; i vincoli sono dati dal tempo disponibile, le risorse utilizzabili e i risultati attesi.

4.2 Responsabile di progetto

La gestione di un progetto è compito del **responsabile di progetto*** e consiste di:

- istanziare processi nel progetto;
- stimare i costi e le risorse necessarie;
- pianificare le attività e assegnarle alle persone;
- controllare le attività e verificare i risultati.

4.3 Ruoli

Ogni persona, in un progetto, ha un ruolo (o funzione, in azienda*). Il ruolo può essere di quattro tipi:

- sviluppo (responsabilità tecnica e amministrativa);
- direzione (responsabilità decisionale);
- amministrazione (gestione dei processi);
- qualità (gestione della qualità).

Allocare le risorse per un progetto significa assegnare attività a ruoli e ruoli a persone.

4.4 Profilo professionale

Ogni persona ha un profilo professionale, cioè un insieme di competenze (tecnologiche e metodologiche) e un'esperienza (espressa in anni e partecipazione a progetti) che fanno da requisiti per l'assunzione di un ruolo in un progetto. Esistono vari profili professionali.

- **Analista** — a partire dal bisogno del cliente, individua il problema (di cui conosce il dominio) da fornire al progettista; solitamente non segue il progetto fino alla fine.
- **Progettista** — ha competenze tecniche e tecnologiche* aggiornate e ha vasta esperienza professionale; a partire dalle specifiche del problema fornitogli, sviluppa una soluzione e rimane finché la soluzione non è stata implementata; spesso si assume la responsabilità di gestione del progetto.
- **Programmatore** — implementa (una parte de) la soluzione del progettista; sta a lungo nel progetto poiché può essere coinvolto nella manutenzione. Ha competenze specifiche; visione e responsabilità circoscritte.
- **Verificatore** — verifica il lavoro prodotto dai programmatori.
- **Responsabile di progetto** — pianifica il progetto, assegna le persone ai ruoli giusti e rappresenta il progetto presso il fornitore e il committente.
- **Amministratore di progetto** — ruolo "orizzontale": deve controllare che ad ogni istante della vita del progetto le risorse (umane, materiali, economiche e strutturali) siano presenti e operanti; inoltre, gestisce la documentazione e controlla il versionamento e la configurazione.
- **Controllore della qualità** — funzione aziendale (e non ruolo di progetto) che accerta la qualità dei prodotti.

4.5 Pianificazione di progetto

Il ruolo più importante del responsabile di progetto è quello di pianificare. La pianificazione è l'identificazione del da farsi e di come farlo. È bene notare come lo stato di avanzamento di un prodotto sia rilevante solo se dà informazioni sulla pianificazione. Tre strumenti notevoli per la pianificazione di un progetto sono:

- I diagrammi WBS (Work Breakdown Structure) decompongono, in modo gerarchico, le attività in sottoattività; pur essendo fortemente coese, le sottoattività non sono necessariamente sequenziali.
- I diagrammi di Gantt sono ideali per rappresentare la durata, la sequenzialità e il parallelismo; si possono confrontare facilmente le stime con i progressi effettivi. Tuttavia, non sono particolarmente adatti per rappresentare le dipendenze tra attività.
- I diagrammi PERT (Project Evaluation and Review Technique) unificano le due tecniche precedenti e sono ideali per rappresentare le dipendenze temporali (e le criticità*) tra attività e, quindi, per ragionare sulle scadenze del progetto. Un tale diagramma è un grafo orientato dove gli archi

rappresentano le attività, mentre i nodi sono degli eventi. Ogni evento ha una data minima a partire da cui può accadere e una data massima oltre la quale esso ritarda gli eventi successivi; la differenza tra questi due tempi è detta slack time*.

4.6 Stima dei costi di progetto

Un'altro compito importante del responsabile di progetto è quello di stimarne i costi. In particolare, il responsabile deve stimare il tempo/persona*, unità di misura delle risorse umane. In questo, utile caveat è la legge di Parkinson, una critica alla regolamentazione fine a se stessa: Work expands to fill the time available. Uno strumento per la stima del tempo/persona è CoCoMo (Constructive Cost Model), una funzione matematica che produce in uscita un valore in tempo/persona e prende in ingresso alcuni parametri relativi al progetto (fattore di complessità del progetto C, misura in KDSI* della dimensione stimata del prodotto software D, fattore di complessità S e moltiplicatori di costo M): $x = C \cdot D^S \cdot M$, dove x è misurato in mesi-persona.

4.7 Rischi di progetto

I risultati di un progetto software possono portare costi eccessivi, non rispettare le scadenze o risultare insoddisfacenti. Un buon metodo per gestire i rischi è il seguente:

1. identificazione dei rischi;
2. analisi dei rischi (per ordinare i rischi secondo una priorità);
3. pianificazione di come evitare i rischi;
4. monitoraggio dei rischi e, eventualmente, ritorno al punto 2 per aggiornare le strategie.

5 Amministrazione di progetto

5.1 Amministratore di progetto

Scopo dell'amministrare un progetto è quello di evitare conflitti che si manifestano quando ci sono sovrapposizioni di ruoli e di respondabilità. L'amministratore di un progetto* non dirige (non compie scelte gestionali) ma deve far sì che l'**infrastruttura** di lavoro sia operante; attua le scelte tecnologiche concordate con i responsabili aziendali e di progetto e si assicura che vengano seguite dai membri del progetto.

5.2 Documentazione di progetto

Uno dei compiti dell'amministratore di progetto è quello di gestire la documentazione. I documenti devono essere chiaramente identificati, corretti nei contenuti, verificati, approvati, aggiornati (specificando la data) e dotati di versione. La loro diffusione dev'essere controllata: i destinatari devono essere chiaramente identificati e ogni documento ha una sua lista di distribuzione (oppure è pubblico).

La documentazione raccoglie **tutto ciò che documenta le attività** e si divide nelle seguenti due categorie.

- Documentazione di sviluppo:
  - documentazione fornita dal cliente;
  - diagrammi di progettazione;
  - codice;
  - piani di qualifica e risultati delle prove;
  - documentazione di accompagnamento del progetto.
- Documentazione di gestione del progetto:
  - documenti contrattuali;
  - piani e consuntivi delle attività;
  - piani di qualità.

Ogni documento contiene un "diario delle modifiche", in cui vengono riportate tutte le modifiche rispetto alla versione precedente del documento.

5.3 Ambiente di lavoro

L'amministratore di progetto si occupa dell'ambiente di lavoro, cioè l'insieme di persone, di ruoli, di procedure e l'infrastruttura* la cui qualità determina la produttività del progetto. L'ambiente di lavoro dev'essere:

- completo (offre tutto il necessario per svolgere le attività previste);
- ordinato (è facile trovare ciò che vi si cerca);
- aggiornato (il materiale obsoleto non deve causare intralcio).

5.4 Configurazione e versionamento di un prodotto

Oltre all'aspetto temporale (cioè il ciclo di vita), ogni prodotto ha anche un aspetto più "spaziale", in quanto si compone di parti. Quali esse sono e il modo in cui stanno assieme è detto "configurazione". E ogni sistema composto di parti va gestito con:

- controllo di configurazione;
- controllo di versione (versione non del prodotto ma di ogni *parte* della configurazione del prodotto).

Data la complessità di un prodotto software, la gestione della configurazione va automatizzata con strumenti adatti. Ogni parte della configurazione (configuration item, CI) dev'essere univocamente identificato (oltre ad avere nome, data, autore, registro delle modifiche e stato corrente). Due concetti centrali della gestione di configurazione sono i seguenti.

- Quello di **baseline** indica un punto d'arrivo tecnico dal quale non si retrocede; la baseline è fatta di elementi della configurazione e, poiché ogni parte è versionata, possiamo conoscere la differenza tra una baseline e l'altra. Una baseline è qualcosa di stabile — non usa e getta! — e sta in un repository*; serve da base per gli avanzamenti futuri e può essere cambiata solo tramite procedure di controllo di cambiamento.
- Il concetto di **milestone** indica un punto nel tempo associato ad un valore strategico. Ogni milestone di calendario è associata a uno specifico insieme di baseline. Ogni milestone dev'essere: specifica, raggiungibile, misurabile (per quantità d'impegno necessario), traducibile in compiti assegnabili e dimostrabile agli stakeholders*.

Anche il controllo di versione fa affidamento sul repository, per permettere di lavorare su vecchi e nuovi CI senza rischio di sovrascritture accidentali, di condividere il lavorato nello spazio comune e di poter verificare la bontà di ogni modifica di baseline. Ogni versione è una istanza di prodotto funzionalmente distinta dalle altre. Invece, si dice "variante" una istanza di prodotto funzionalmente identica ad altre ma diversa per caratteristiche non funzionali. Infine, si dice "rilascio" (release) una istanza di prodotto resa disponibile a utenti esterni.

5.5 Modifiche

Anche nel corso del suo sviluppo, un progetto non è esente da richieste di modifiche (dagli utenti, dagli sviluppatori o semplicemente per competizione). Le richieste di modifica vanno sottoposte a un rigoroso processo di analisi, decisione, realizzazione e verifica; di ogni richiesta va tenuta traccia.

5.6 Norme di progetto

Un progetto necessita di linee guida per le attività di sviluppo. Le norme — che vanno accertate dall'amministratore — comprendono:

- organizzazione e uso delle risorse di sviluppo;
- convenzioni sull'uso degli strumenti di sviluppo;
- organizzazione della comunicazione e della cooperazione;
- norme di codifica;
- gestione dei cambiamenti.

Le norme di progetto descrivono come dovrà essere il **way of working**. Individuiamo due categorie di norme: regole (sottoposte a verifica) e raccomandazioni (suggerimenti, senza verifica). Tra le norme di progetto, particolare rilevanza hanno le norme di codifica; queste hanno l'obiettivo di far sì che il codice sorgente sia leggibile (anche a distanza di tempo) e costituiscono una misura preventiva che garantisce verificabilità, manutenibilità e portabilità.

6 Ingegneria dei requisiti

6.1 Requisito

I requisiti di un sistema sono le descrizioni di cosa il sistema deve fare, cioè i servizi che offre e i vincoli sul suo funzionamento. Due definizioni un po' più formali sono:

- ponendoci dal punto di vista del bisogno, requisito è una condizione necessaria a un utente per risolvere un problema o raggiungere un obiettivo;
- dal punto di vista della soluzione, invece, requisito è una condizione che dev'essere soddisfatta da un sistema per adempiere a un obbligo.

I requisiti hanno a che vedere con il processo di sviluppo del software; tuttavia la loro gestione è qualcosa di costante, che viene iterato lungo *tutto* il ciclo di vita di un progetto.

6.2 Requisiti utente e di sistema

"Requisito" è un termine leggermente ambiguo, in quanto viene usato per indicare sia una richiesta generale (astratta, di alto livello) sia una definizione formale e dettagliata di una funzione del sistema. È bene separare questi differenti livelli di descrizione; per questo, distinguiamo tra requisiti utente (di alto livello) e requisiti di sistema (più dettagliati).

6.3 Requisiti di prodotto e di processo

È bene anche distinguere tra requisiti di prodotto (dei bisogni o dei vincoli sul software da sviluppare) e requisiti di processo (dei vincoli sullo sviluppo del software).

6.4 Requisiti funzionali e non

Un'ulteriore distinzione viene fatta tra:

- requisiti funzionali — i servizi che il sistema deve fornire (cioè la sua interfaccia);
- requisiti non funzionali — i *vincoli* sui servizi che il sistema fornisce (requisiti su prestazioni, manutenibilità, sicurezza, affidabilità...).

Ma tale distinzione non è sempre netta: ad esempio, il requisito di limitare l'accesso ai soli utenti autorizzati (apparentemente non funzionale) può essere sviluppato più in dettaglio fino a richiedere un servizio di autenticazione (requisito chiaramente funzionale)!

6.5 Piano di qualifica

Per garantire il rispetto dei requisiti entra in gioco il piano di qualifica, che consiste nel definire le strategie di **verifica** e scegliere metodi, tecniche e procedure da usare per la **validazione**; ha quindi a che fare con due processi:

- Verifica: accertare che l'esecuzione delle attività di processo non abbia introdotto errori (Did I build the system right?); rivolta ai processi (e svolta sui loro prodotti), per accertare il rispetto di norme e procedure.
- Validazione: accertare che il prodotto realizzato corrisponda alle attese (Did I build the right system?); rivolta ai prodotti finali.

La validazione dev'essere una *self fulfilling prophecy*, cioè bisogna essere certi che non fallirà; la verifica serve proprio a garantire questo. Infatti, il processo di verifica deve assicurarci che lavoriamo bene non a posteriori ma mentre lavoriamo. Se la verifica assicura i requisiti, la validazione li accerta. Il piano di qualifica nasce assieme ai requisiti.

6.6 Attività

Il processo di ingegneria dei requisiti raggruppa quattro attività:

1. **studio di fattibilità** (stabilire se il sistema in questione è redditizio);
2. acquisizione* e **analisi** dei requisiti;
3. **specifica** dei requisiti (cioè formalizzare i requisiti);
4. **validazione** dei requisiti.

Tale processo riguarda tutti gli stakeholders. In genere non è possibile soddisfare i requisiti di ognuno di essi, quindi bisogna quindi trovare un buon compromesso; questo presuppone quindi che gli stakeholders vengano identificati, "pesati" e interpellati.

6.7 Studio di fattibilità

Lo studio di fattibilità è uno studio breve e chiaro che consiste nel valutare **rischi**, **costi** e **benefici** legati al sistema da sviluppare: tale sistema contribuisce agli obiettivi generali dell'organizzazione? può essere sviluppato rispettando determinati vincoli economici, con la tecnologia corrente? può essere integrato con altri sistemi in uso? una risposta negativa in una qualunque tra queste domande inficia la fattibilità del sistema. Lo studio di fattibilità dovrebbe descrivere in modo chiaro gli **obiettivi** del progetto e valutare approcci alternativi, per capire se il progetto proposto è la migliore alternativa.

6.8 Acquisizione e analisi dei requisiti

Dopo aver compiuto uno studio di fattibilità, gli ingegneri del software devono lavorare assieme a clienti e utenti per individuare il dominio di applicazione e i requisiti del sistema. In generale, tutti gli stakeholders sono coinvolti in questa attività, che prende il nome di "acquisizione e analisi dei requisiti" e si svolge a grandi linee nel seguente modo.

1. Studio dei bisogni e delle fonti; si cerca di individuare un insieme (non strutturato) di requisiti. Per fare questo, si può:

   - interrogare gli stakeholders — con interviste chiuse (insieme predefinito di domande) o aperte;
   - discutere con gli stakeholders alcuni scenari del sistema (uno scenario è la descrizione di un esempio di interazione col sistema);
   - discutere con gli stakeholders i casi d'uso del sistema (tramite diagrammi che individuino le interazioni tra il sistema e i suoi utenti);
   - studiare un prototipo del sistema;
   - discutere in modo creativo, tramite brainstorming;
   - osservare il sistema in modo etnografico, concentrandosi sul suo funzionamento abituale.

   Interviste e scenari (oltre al capitolato d'appalto, chiaramente) sono fonte di requisiti espliciti; per ricavare, invece, i requisiti impliciti, gli ingegneri del software devono capire il dominio di applicazione del sistema (magari creando un glossario dei termini chiave del dominio).

2. Classificazione e organizzazione dei requisiti; l'insieme dei requisiti viene strutturato, dividendo i requisiti in gruppi che rispecchino l'architettura del software (qui, progettazione e analisi procedono spesso insieme).

3. Modellazione concettuale del sistema (ad esempio tramite un diagramma dei casi d'uso).

4. Assegnazione dei requisiti a parti distinte del sistema.

5. Negoziazione con il committente e con i sotto-fornitori: essendoci diversi stakeholders, è normale che alcuni requisiti siano in conflitto; bisogna dare una priorità ad ogni requisito e negoziare quelli incompatibili per trovare un compromesso.

I requisiti possono cambiare (a causa di condizioni esterne o anche solo perché l'analisi si è approfondita e ha introdotto nuovi requisiti); proprio per questo, è bene notare che la sequenza di passi riportata diventa spesso un ciclo che si ripete.

6.9 Specifica dei requisiti

I requisiti vanno specificati in un documento, usando un linguaggio formale o grafico. [?? ...] Vanno ordinati per priorità, classificati e identificati univocamente.

6.10 Validazione dei requisiti

Validare i requisiti vuol dire controllare che essi definiscano effettivamente il sistema che il cliente richiede. A partire dal documento generato durante la specifica dei requisiti, bisogna assicurarsi che questi siano:

- non ambigui;

- necessari — ogni requisito deve soddisfare qualche bisogno esplicito (dal capitolato di appalto);
- sufficienti — ogni bisogno dev'essere soddisfatto da qualche requisito del documento;
- coerenti;
- realistici — i requisiti devono essere implementabili con la tecnologia a disposizione;
- verificabili — si dev'essere in grado di dimostrare che il sistema soddisfa i requisiti.

7 Progettazione

7.1 Definizione

La risoluzione di un problema attraversa due fasi: la prima è **analitica**, la seconda **sintetica**. Nella fase analitica il problema viene decomposto, approfondito nel dettaglio per capire di quali parti è formato (approccio *top-down*); in quella sintetica, invece, si ricompongono i pezzi trovati al passo precedente e si sintetizza una soluzione per il problema (approccio *bottom-up*). Se la fase analitica ("qual è il problema?") corrisponde grosso modo all'attività di analisi dei requisiti, quella sintetica ("come risolvere il problema?") è proprio la progettazione. Formalmente, progettazione è il processo di definizione dell'architettura, dei componenti, delle interfacce e delle altre caratteristiche di un sistema o componente.

7.2 Progettazione architetturale e progettazione di dettaglio

Il processo di progettazione passa attraverso due attività:

- **progettazione architetturale**, di alto livello, che descrive come il software viene organizzato in componenti;
- **progettazione dettagliata**, che descrive il comportamento di tali componenti.

Nel piano di qualifica, se l'analisi è verificata dal test di sistema, la progettazione architetturale è verificata dal test d'integrazione e quella di dettaglio dai test di unità. (Questo è il cosiddetto "modello a V".)

7.3 Architettura

Obiettivo della progettazione è definire l'architettura del sistema. Per architettura s'intende la decomposizione di un sistema in componenti, l'organizzazione di tali componenti, le interfacce dei componenti e i loro paradigmi di composizione. In generale, una buona architettura deve rispettare i seguenti princìpi:

- sufficienza — deve soddisfare tutti i requisiti;
- comprensibilità — può essere capita dagli stakeholders;

- modularità — le sue parti sono chiare e distinte, non si sovrappongono;
- robustezza — è capace di gestire un'ampia classe di input diversi;
- flessibilità — permette modifiche a costo contenuto;
- riusabilità — alcune sue parti possono essere utilmente impiegate in altre applicazioni;
- efficienza.
- affidabilità — è altamente probabile che svolga bene il suo compito;
- disponibilità — necessita di poco tempo di manutenzione fuori linea;
- sicurezza rispetto ai malfunzionamenti.
- sicurezza rispetto a intrusioni.
- semplicità — ogni parte contiene solo il necessario e niente di superfluo;
- incapsulamento — nasconde i dettagli implementativi;
- coesione — parti associate concorrono agli stessi obiettivi (l'approccio a oggetti aiuta molto a ottenere coesione);
- basso accoppiamento — parti distinte dipendono il meno possibile le une dalle altre.

Tutte queste qualità devono essere misurabili. In particolare, l'accoppiamento è misurabile interpretando le componenti di un sistema come i nodi di un grafo orientato dove gli archi sono dipendenze di un componente nei confronti di un altro; il numero di archi entranti (*fan-in*) è indice di utilità, mentre il numero di archi uscenti (*fan-out*) è indice di dipendenza. Riguardo alla riusabilità, infine, è bene notare che costituisce un guadagno soltanto nel lungo termine, mentre nel breve termine è un puro costo.

7.4 Design patterns

Un *design pattern* è una soluzione progettuale a un problema ricorrente. Per la progettazione esistono design patterns di alto livello (gli stili architetturali) e di basso livello.

7.5 Progettazione architetturale

Esistono vari stili architetturali e aderire a uno di essi garantisce coerenza; alcuni stili sono:

- strutture generali (livelli, oggetti, pipes e filtri, blackboard);
- sistemi distribuiti (*client-server*, *three-tiers*, *peer-to-peer*, *broker*);
- sistemi interattivi (*Model-View-Controller*, *Presentation-Abstraction-Control*);
- sistemi adattabili (microkernel, riflessione);
- altri (batch, interpreti...).

7.6 Progettazione di dettaglio

Uno stile architetturale è un design pattern di alto livello; per la progettazione di dettaglio, invece, si ricorre a pattern di più basso livello (pattern creazionali, strutturali e comportamentali). La progettazione di dettaglio deve definire delle unità il cui carico di lavoro sia realizzabile da un singolo programmatore, in parallelo con le altre unità. Quanto più piccolo è il compito tanto più piccolo è il rischio.

8 Documentazione

8.1 Definizione

Documentazione è tutto il materiale che documenta le attività di un progetto (e i loro prodotti). wikibooks.org afferma che, in ingegneria, l'obiettivo primario della documentazione di un prodotto è la *replicabilità* di tale prodotto: You are not an engineer until others can replicate what you have done [...] without your presence [...] without your words [...] without your physical personality on the planet. I documenti di un progetto software e del prodotto che si sta sviluppando hanno diversi motivi di esistere:

- per i membri del progetto, sono un mezzo di comunicazione;
- per chi dovrà manutenere il prodotto, servono da repository di informazioni sul sistema;
- per l'amministrazione, forniscono informazioni che aiutano la pianificazione dello sviluppo;
- per gli utenti del prodotto, alcuni documenti sono utili per capire come interfacciarsi con il prodotto.

I principali documenti di progetto sono:

- analisi dei requisiti;
- specifica software (descrizione ad alto livello del sistema);
- specifica tecnica (architettura logica);
- definizione di prodotto (architettura di dettaglio);
- manuale utente.

8.2 Specifica software

La specifica software dev'essere una descrizione ad alto livello del sistema tramite rappresentazione gerarchica.

8.3 Specifica tecnica

La specifica tecnica (ST) segue l'analisi dei requisiti e descrive l'**architettura logica** del sistema, mostrando ciò che il sistema deve fare senza però fissarne i dettagli implementativi; difatti, la specifica tecnica descrive l'*interfaccia* di ogni componente del sistema, attraverso più livelli gerarchici di decomposizione. In particolare, di ogni componente vengono specificati:

- funzione svolta;
- tipo dei dati in ingresso;
- tipo dei dati in uscita;
- risorse necessarie per il suo funzionamento.

8.4 Definizione di prodotto

Dall'architettura logica procede l'**architettura di dettaglio**, descritta dalla definizione di prodotto (DP). Questo documento decompone l'architettura in moduli a granda più fine, finché ogni modulo ha dimensione, coesione, complessità e accoppiamento tali per cui i moduli possano essere sviluppati in parallelo dai programmatori. La DP deve fornire tutti i dettagli necessari alla codifica e alla verifica di ciascun modulo*. La DP consente di stimare costi e tempi di realizzazione.

8.5 Analisi dei requisiti

Una delle preoccupazioni maggiori nel documentare l'analisi dei requisiti (AR) è il **tracciamento dei requisiti**. Tracciare un requisito significa motivarne l'esistenza, spiegando qual'è l'origine di tale requisito (che magari non era esplicito e quindi è stato dedotto da requisiti più espliciti) e badando a garantire la necessità e la sufficienza di ogni requisito. Per fare tutto questo si può usare una matrice, un grafo o una qualsiasi struttura dati appropriata. [... -> LUCIDI 19-20]