sweki

Giorgio Giuffrè

Indice

0 Sommario

sweki è un progetto che raccoglie i miei appunti dal corso di Ingegneria del Software tenuto dal prof. Tullio Vardanega nell'anno accademico 2016-2017. Gli appunti sono presi un po' dalle lezioni del professore e un po' dal materiale riportato in bibliografia.

Gli appunti sono disponibili su GitHub all'indirizzo `https://github.com/FIUP/sweki` in tre formati:

- .html — `https://github.com/FIUP/sweki`;
- .tex — `https://github.com/FIUP/sweki/tex/sweki.tex`;
- .pdf — qui.

la versione in LaTeX è derivata automaticamente da quella in XHTML tramite dei template XSLT e dei terribili script di shell (bash).

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; 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 delle 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:

- standard di processo — 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 dello standard di processo 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).

Esiste il processo perfetto? Esiste l'insieme ideale di processi? No: nessun progetto è identico a un altro, quindi i processi vanno selezionati e adattati in modo critico, in base al progetto e/o all'organizzazione a cui servono. Selezionare, adattare, applicare e migliorare (evolvere) i processi è compito degli amministratori di progetto.

I processi software, di per sé, non seguono un ordinamento. Le relazioni temporali tra essi sono fornite da un modello di ciclo di vita.

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;
  - gestione delle versioni e della configurazione;
  - accertamento della qualità;
  - qualifica: verifica e validazione ("V&V"), due processi distinti ma collegati;
  - revisioni congiunte con il cliente;
  - verifiche ispettive esterne;
  - risoluzione dei problemi.
- **Processi organizzativi** (l'"ambiente" del sistema):
  - gestione dei processi;
  - gestione delle infrastrutture;
  - miglioramento del processo;
  - formazione del personale.

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

2.5 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.

2.6 Efficienza ed efficacia di un processo

Un buon esempio di valutazione dell'efficienza e dell'efficacia è la misurazione di questi due aspetti a livello di processo. Efficienza di un processo, attività o compito è il rapporto tra le **risorse** realmente consumate e quelle che si era previsto venissero consumate; efficacia di un processo, attività o compito è il rapporto tra i **prodotti** realmente ottenuti a partire dal processo e i prodotti che si volevano ottenere.

3 Ciclo di vita del software

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

I processi software, di per sé, non seguono un ordinamento; le relazioni temporali e logiche tra essi sono fornite da un **modello** 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.

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.

In genere un modello del ciclo di vita di un *prodotto* include un modello del ciclo di vita dello *sviluppo* (più eventuali altri processi che riguardano fornitura, manutenzione, evoluzione...). Attenzione: il ciclo di vita dello sviluppo non deve per forza seguire lo stesso modello del ciclo di vita dell'intero prodotto!

Tra le qualità che contraddistinguono l'IS — sistematicità, disciplina, quantificabilità — i modelli di ciclo di vita nascono con l'obiettivo di perseguire la **quantificabilità**, che è la più difficile da soddisfare.

3.3 Il modello sequenziale

Nel 1970, grazie a Winston Royce, venne ideato il modello sequenziale (o a cascata), ispirato alle catene di montaggio. Questo è una successione di **fasi** rigidamente sequenziali. 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: ogni fase produce documenti che la concretizzano e devono essere approvati per il passaggio alla fase successiva. Nello specifico, ogni fase viene definita in termini di:

- attività previste;
- prodotti attesi in ingresso;
- prodotti attesi in uscita;
- ruoli coinvolti;
- scadenze di consegna.

Questo modello 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 correggere il modello creando un "ibrido", introducendo dei prototipi "usa e getta" oppure 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. Difatti, l'IS è un insieme di best practices che assembla cose già esistenti, più che crearle ex novo.

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 realizzativa);
- 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. In un certo senso, l'analista è la giuntura che collega gli utenti agli sviluppatori.
- **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.

Il primo passo da fare nel gestire un progetto dovrebbe essere la selezione di un modello di ciclo di vita per lo sviluppo del prodotto.

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, la progettazione è la definizione dell'architettura, dei componenti, delle interfacce e delle altre caratteristiche di un sistema o componente.

L'architettura di un software è, tipicamente, un insieme di **moduli** che si raggruppano in **unità**, che a loro volta si raggruppano in **componenti**, che vanno a formare un **sistema**. Acronimo tattico per ricordarsi: SCUM.

7.2 Progettazione architetturale e progettazione di dettaglio

Il processo di progettazione passa attraverso due attività, che si situano tra l'analisi dei requisiti e l'implementazione:

- **progettazione architetturale**, di alto livello, che descrive come il software viene organizzato in componenti;
- **progettazione di dettaglio**, 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 dai 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 pattern* e stili architetturali

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

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 una soluzione progettuale di alto livello; per la progettazione di dettaglio, invece, si ricorre ai design pattern, che si dividono in tre famiglie:

- design pattern creazionali, che cercano di rendere un sistema indipendente dall'implementazione concreta delle sue componenti;

- design pattern strutturali, che affrontano problemi riguardanti la composizione di classi e oggetti;
- design pattern comportamentali, che si occupano del comportamento degli oggetti e delle collaborazioni tra essi.

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 grana 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.

8.6 Manuale utente

Il manuale utente (MU) ha la funzione di spiegare all'utente come utilizzare il prodotto. Chi lo redige deve stare attento allo stile: deve preferire frasi brevi, paragrafi focalizzati, uso di liste, stile semplice, terminologia precisa e uso della forma attiva dei verbi. Può essere di diversi tipi, a seconda del grado di esperienza dell'utente (manuale introduttivo, manuale completo...) e in diversi formati (cartaceo, PDF, ipertestuale...).

8.7 Tracciamenti

In generale, i principali tracciamenti all'interno di un prodotto sono:

- tra requisiti utente (capitolato) e requisiti software (AR);
- tra requisiti software (AR) e descrizione dei componenti (ST);
- tra test di unità e moduli della progettazione di dettaglio (DP);

- tra test di integrazione e componenti architetturali (ST);
- tra test di sistema e requisiti software (AR);
- tra test di accettazione e requisiti utente (capitolato).

9 Progetto didattico

9.1 Motivazioni

Un corso di Ingegneria del Software sarebbe incompleto senza un progetto didattico: è bene che lo studente si scontri in prima persona con un progetto di gruppo che rispecchi la logica di relazione "cliente-fornitore". Nello specifico, le principali entità del progetto didattico sono:

- Il cliente o **committente**, cioè chi ha commissionato il prodotto; generalmente è un'azienda o una startup, che pubblica un capitolato d'appalto in cui viene spiegato qual è il prodotto da realizzare.
- Il **fornitore**, cioè chi si impegna a fornire il prodotto richiesto dal committente; è un gruppo di sei o sette studenti che scelgono di rispondere ad un particolare capitolato d'appalto. Il gruppo è un'entità coesa provvista di nome, logo e e-mail per le comunicazioni ufficiali.
- Il docente, che nel nostro caso funge da **proponente**; il docente verifica e valuta l'andamento dei vari progetti.

9.2 Revisioni di avanzamento

Il progetto si estende nell'arco di uno o due semestri. Il docente potrebbe scegliere tra due approcci: l'approccio "chiavi in mano" (i gruppi sono lasciati liberi e il docente torna il giorno della consegna del prodotto); l'approccio per revisioni (più interattivo). L'approccio adottato è il secondo, molto più didattico. Quindi la principale modalità d'interazione tra gruppi e docente sono le **revisioni di avanzamento**, che sono quattro:

1. Revisione dei Requisiti (RR);
2. Revisione di Progettazione (RP);
3. Revisione di Qualifica (RQ);
4. Revisione di Accettazione (RA).

È importante notare che la logica sequenziale delle revisioni di avanzamento non implica che il modello di sviluppo scelto dai gruppi debba essere anch'esso sequenziale!

Ricordiamo che ISO/IEC 12207 prevede due diversi tipi di processi di revisione: l'audit process e il joint review process. Alla prima classe appartengono le due revisioni esterne con effetto bloccante: RR e RA; alla seconda classe appartengono le due revisioni interne con valenza informativa: RP e RQ.

9.3 Documentazione

Tutto il progetto va documentato. In particolare, il fornitore deve documentare: le sue strategie di efficienza ed efficacia, nei documenti di strategia; le regole per attuare tali strategie, nelle Norme di Progetto. I documenti di strategia, che servono sia al fornitore sia al committente, sono:

- Il **Piano di Progetto**, che tratta delle strategie che il gruppo sceglie di adottare per essere efficiente; esso presenta l'organigramma dettagliato del fornitore, lo schema proposto per l'assegnazione e la rotazione dei ruoli di progetto, l'impegno complessivo previsto per ogni ruolo e per ogni individuo, e il conto economico preventivo di realizzazione del prodotto.
- Il **Piano di Qualifica**, che tratta dell'efficacia e garantisce che le attese verranno rispettate; esso illustra la strategia complessiva di verifica e validazione proposta dal fornitore.

Invece, le **Norme di Progetto** interessano solo il fornitore (e il docente) ma non il committente; esse servono a fissare il way of working.

In generale, i principali documenti del progetto didattico sono:

- Documenti gestionali:
  - Studio di Fattibilità;
  - Norme di Progetto;
  - Piano di Progetto;
  - Piano di Qualifica.
- Documenti tecnici:
  - Analisi dei Requisiti;
  - Specifica Tecnica;
  - Definizione di Prodotto;
  - Manuale Utente.

Possiamo suddividere i documenti anche tra interni (al gruppo) ed esterni. Documenti interni sono:

- Studio di Fattibilità;
- Norme di Progetto.

Documenti esterni sono:

- Piano di Progetto;
- Piano di Qualifica;
- Analisi dei Requisiti;
- Specifica Tecnica;
- Definizione di Prodotto;
- Manuale Utente.

9.4 Revisione dei Requisiti

Questa revisione ha la funzione di concordare con il cliente una visione condivisa del prodotto atteso. Tale visione è documentata nel documento di Analisi dei Requisiti, che studia i requisiti e i casi d'uso del prodotto da realizzare. I documenti valutati sono:

- Studio di Fattibilità;
- Norme di Progetto;
- Piano di Progetto;
- Piano di Qualifica;
- **Analisi dei Requisiti**.

9.5 Revisione di Progettazione

Questa revisione può avvenire in una (sola) delle seguenti modalità:

1. Revisione di Progettazione *minima*;
2. Revisione di Progettazione *massima*.

La RP minima riguarda l'architettura del sistema *ad alto livello*; essa ha la funzione di accertare la realizzabilità del prodotto. I documenti valutati sono:

- Norme di Progetto;
- Piano di Progetto;
- Piano di Qualifica;
- **Specifica Tecnica**.

La RP massima riguarda l'architettura *di dettaglio* del sistema; essa ha la funzione di accordarsi sulle caratteristiche del prodotto da realizzare. I documenti valutati sono:

- Norme di Progetto;
- Piano di Progetto;
- Piano di Qualifica;
- **Definizione di Prodotto**.

9.6 Revisione di Qualifica

Questa revisione ha la funzione di approvare l'esito finale delle verifiche e attivare quindi la validazione. I documenti valutati sono:

- Norme di Progetto;
- Piano di Progetto;
- Piano di Qualifica;
- Definizione di Prodotto;
- versione preliminare del Manuale Utente.

9.7 Revisione di Accettazione

Questa revisione ha la funzione di collaudare il sistema e di accertare il soddisfacimento di tutti i requisiti utente stabiliti nell'Analisi dei Requisiti. I documenti valutati sono le versioni definitive di:

- Piano di Progetto;
- Piano di Qualifica;
- Manuale Utente.

9.8 Ore di lavoro

Le ore di lavoro si contano solo *dopo* la valutazione della RR; fino a quel momento, il tempo speso è tempo d'investimento non rendicontato (eventualmente registrato, ma non rendicontato). Le ore totali impiegate a progetto da ciascuna persona sono circa 25-30 ore di lavoro; a questo proposito, è bene notare come l'efficienza è proporzionale al rapporto tra ore di lavoro e ore effettive. L'impegno totale di ore rendicontabili presentate a consuntivo da ogni componente di un gruppo dovrà situarsi fra un minimo di 85 e un massimo di 105 ore produttive. Le ore rendicontabili non includono le attività di auto-formazione.

10 Qualità del software

10.1 Definizione di qualità

Secondo ISO, la qualità è l'insieme delle caratteristiche di un'entità che ne determinano la capacità di soddisfare esigenze espresse e implicite. Valutare la qualità del software serve sia a chi lo ha realizzato (per migliorarlo), sia a chi lo usa (per avere garanzie), sia a chi valuta (per imparare). Tuttavia, quello di qualità è un concetto ambivalente: può riguardare la conformità ai requisiti (visione intrinseca della qualità), la soddisfazione del cliente (visione relativa) o la misura del livello di qualità (visione quantitativa); ad ogni modo, tutte queste sono aree in cui interviene il "sistema qualità".

10.2 Il sistema qualità

ISO definisce il "sistema qualità" come la struttura organizzativa, le responsabilità, le procedure, i procedimenti e le risorse messe in atto per il perseguimento della qualità. Questo sistema gestisce la qualità in tre ambiti:

- La **pianificazione**, cioè le attività del sistema qualità mirate a fissare gli obiettivi di qualità, con i processi e le risorse necessari per conseguire tali obiettivi; è una premessa al controllo della qualità.
- Il **controllo**, cioè le attività del sistema qualità pianificate e attuate affinché il prodotto soddisfi i requisiti attesi.
- Il **miglioramento continuo**, secondo lo schema PDCA.

Il sistema qualità è fissato nelle Norme di Progetto (il Piano di Qualifica fissa invece gli *obiettivi* di qualità: i piani fissano solo obiettivi e strategie per raggiungerli).

10.3 Modelli della qualità

Per uniformare la percezione e la valutazione della qualità, è bene che committenti e fornitori si accordino su un **modello** da seguire per la qualità. Esistono vari modelli della qualità; ognuno di essi definisce le caratteristiche rilevanti che deve avere un prodotto "di qualità" e organizza tali caratteristiche in una struttura logica.

Ad esempio, ISO/IEC 9126 definisce 7 caratteristiche principali (suddivise poi in 31 sotto-caratteristiche):

- funzionalità;
- affidabilità;
- efficienza;
- usabilità;
- manutenibilità;
- portabilità;
- qualità in uso.

Oltre a descriverle, il modello specifica anche come *misurarle*, cioè assegna ad ogni caratteristica una metrica (che è un modo per dare un significato a dei valori). Un modello è quindi uno strumento di definizione e di valutazione:

- **definizione** perché cataloga in modo sistematico delle caratteristiche rilevanti;
- **valutazione** perché definisce delle metriche per la misurazione di tali caratteristiche.

Sempre in ISO/IEC 9126, vengono proposte tre visioni della qualità:

- Visione esterna, relativa all'esecuzione del prodotto; è ciò che si osserva, quindi solo la punta dell'iceberg.
- Visione interna, relativa al prodotto *non* in esecuzione; è ciò che deriva dalle scelte di progettazione, codifica e verifica, quindi si può vedere solo attraverso una revisione critica del prodotto.
- Visione in uso, relativa alla percezione dell'utente finale.

Mentre le visioni interna ed esterna si riferiscono al prodotto software, la visione in uso si riferisce agli *effetti* di tale prodotto.

10.4 Qualità nel ciclo di vita

Possiamo individuare una catena di dipendenze tra le varie visioni della qualità: la **qualità di un processo** influenza gli attributi della **qualità interna**, che influenza gli attributi della **qualità esterna**, che influenza gli attributi della **qualità in uso**.

È bene che i seguenti aspetti siano il meno distante possibile tra loro:

1. qualità obiettivo;
2. qualità richiesta;
3. qualità progettata;
4. qualità stimata;
5. qualità consegnata.

Il committente ha responsabilità sulle prime due; il fornitore su tutte quante tranne la prima.

10.5 Metriche e misurazioni

Una **metrica** è un modo per dare un significato a dei valori, quindi è l'interpretazione di un sistema di unità di misura. Una **misurazione quantitativa** è l'uso di una metrica per assegnare un valore su una scala predefinita.

Le metriche software permettono di quantificare sia un prodotto (e quindi predire i suoi attributi) sia un processo (al fine di monitorarlo e migliorarlo).

L'uso di una metrica assume che:

- Si può misurare un certo attributo (o proprietà) del software.
- Esiste una relazione tra ciò che possiamo misurare e ciò che vogliamo sapere. Ad esempio, generalmente è possibile misurare solo attributi *interni* del software, mentre ciò che più spesso ci interessa sono i suoi attributi *esterni*: per capire il grado di manutenibilità di un software (attributo esterno), dobbiamo conoscere il numero di righe di codice, il numero di parametri per procedura, l'ampiezza del manuale utente... (attributi interni).
- La suddetta relazione è stata formalizzata e validata.

La selezione delle metriche (e la scelta dei criteri di accettazione) parte dall'analisi dei requisiti di qualità e dei vincoli di costo. Dopodiché:

1. una volta selezionate le metriche, si può procedere alla misurazione;
2. poi si passa alla valutazione (che è una verifica quantificata);
3. e infine viene l'accettazione — basandosi, appunto, sui criteri di accettazione scelti.

11 Qualità di processo

11.1 Risalire dal prodotto al processo

Da tubi sporchi non esce acqua pulita: per ottenere *prodotti* di qualità, è necessario avere *processi* di qualità. In altre parole, la **qualità di processo** sta a monte rispetto alla qualità di prodotto.

Un processo è una macchina che prende in input dei bisogni e dà in output dei prodotti, consumando risorse nel farlo. Come tutte le macchine, un processo va controllato; questo permette di assicurare che i prodotti che produce siano

quelli desiderati. Per controllare un processo, come per controllare una *self-driving car* in movimento, sono necessari dei sensori e degli attuatori: gli uni per conoscere lo stato del processo (la posizione della macchina), gli altri per modificare l'implementazione del processo (la direzione in cui va la macchina). In questo paragone il controllore del processo è la ***quality assurance***, i sensori sono delle **misurazioni** che vengono effettuate sul processo e gli attuatori sono **regole e decisioni** atte a migliorare il processo.

Tutto questo, per essere efficiente, va fatto con misurazioni preventive (e non a posteriori) il cui obiettivo è il continuo miglioramento del processo: la qualità di processo segue il ciclo PDCA.

11.2 ISO 9000

Il concetto di qualità di processo fu introdotto dai *big spender*, per avere delle classifiche oggettive di fornitori ed essere, quindi, più sereni. Nel 1987 nacque dunque ISO 9000, una famiglia di standard che riguardano i sistemi di gestione della qualità (QMS, Quality Management System). Questa famiglia di standard non si occupa solo di software ma astrae dal dominio di applicazione.

ISO 9000, come documento, introduce i fondamenti e un glossario; invece, ISO 9001 si occupa di calare la visione di ISO 9000 nei **sistemi produttivi**, introducendo dei requisiti ben specifici. Esistono enti che si occupano di certificare il rispetto di tali requisiti (ISO 9001 è quindi anche una *certificazione*). Infine, ISO 9004 è una guida al miglioramento dei risultati.

Questa famiglia di standard si basa su 7 princìpi di gestione della qualità (QMP):

- QMP 1 — customer focus
- QMP 2 — leadership
- QMP 3 — engagement of people
- QMP 4 — process approach
- QMP 5 — improvement
- QMP 6 — evidence-based decision making
- QMP 7 — relationship management

11.3 Sistema di gestione della qualità

La gestione della qualità, come funzione aziendale, dovrebbe essere al di sopra dei vari settori e dovrebbe riferire direttamente alla direzione; questa funzione deve infatti garantire la qualità in modo *trasversale* ai vari settori.

Il sistema di gestione della qualità è documentato tramite:

- Una **politica per la qualità**, cioè le motivazioni che stanno alla base delle scelte sulla qualità.
- Il **manuale della qualità**, che definisce il QMS di un'oraganizzazione. Adotta una visione *operativa* (orizzontale, ad alto livello).

- Il **piano della qualità**, una concretizzazione del manuale della qualità che adotta una visione *strategica* (verticale). Esso definisce gli elementi del QMS e le risorse che devono essere applicate in uno specifico caso (prodotto, processo o progetto). Può avere valenza contrattuale. Nel progetto didattico, il piano della qualità è integrato nel piano di qualifica.

11.4 Strumenti di valutazione

Alcuni strumenti per la valutazione della qualità di processo:

- SPY (Software Process assessment and Improvement)
- CMMI (Capability Maturity Model with Integration)
- SPICE (Software Process Improvement Capability dEtermination)

SPY si incentra sulla valutazione oggettiva dei processi di un'organizzazione. Esso fornisce un giudizio di maturità e individua azioni migliorative.

CMMI si basa su quattro termini:

1. Capability — misura di quanto è adeguato un singolo processo, rispetto agli scopi per cui è stato definito. Un processo ad alto livello di capability è seguito da tutti in modo disciplinato, sistematico e quantificabile.
2. Maturity — misura di quanto è governato l'intero sistema di processi dell'organizzazione. L'intelligenza dei processi di un'organizzazione si chiama *governance*.
3. Model — insieme di requisiti via via più stringenti per valutare il percorso di miglioramento dei processi dell'organizzazione.
4. Integration — architettura di integrazione delle diverse discipline (sistema, hardware, software) e tipologie di attività delle organizzazioni.

È bene notare che mentre la capability è caratteristica di un singolo processo, la maturity è caratteristica di un insieme di processi.

La governance di un'organizzazione ha interesse ad alzare il livello di maturity dell'organizzazione, dato che questo influenza direttamente la capability. CMMI definisce cinque livelli di maturità:

1. **initial** — i processi sono imprevedibili, poco controllati e poco reattivi;
2. **managed** — l'organizzazione applica la "D" di PDCA: i processi sono caratterizzati per i progetti e sono spesso reattivi;
3. **defined** — l'organizzazione applica la "P" di PDCA: i processi sono caratterizzati per l'organizzazione e sono proattivi;
4. **quantitatively managed** — l'organizzazione applica "PC" di PDCA: i processi sono misurati e controllati;
5. **optimized** — l'organizzazione applica PDCA in toto: l'organizzazione si concentra sul miglioramento dei processi.

Nota bene: la "P" di PDCA è pianificazione di *miglioramento*, non pianificazione di progetto.

Limitazioni di CMMI sono: la sua natura discreta (non continua: quindi potenzialmente frustrante per le organizzazioni); un'eccessiva concentrazione sulle pratiche (cosa si fa e come); un'insufficiente attenzione agli obiettivi (perché lo si fa).

SPICE è nato nel 1992 per armonizzare SPY con ISO/IEC 12207 e ISO 9001; nel 1998 è confluito in ISO/IEC 15504 (creato da ISO a partire da CMMI. Qui scompare il concetto di maturity e, al posto dei cinque gradini di maturità, si usa una scala più fine. Le aziende vengono misurate secondo il fatturato e il personale. La sua metodologia di valutazione è la seguente:

- identificazione dei portatori d'interesse (*stakeholders*: utenti, sviluppatori, valutatori...);
- scelta tra valutazione e miglioramento;
- definizione della portata (quali processi vanno inclusi nella valutazione?).

12 Verifica e validazione: introduzione

12.1 Definizione

Verifica e validazione sono due processi strettamente correlati. A questa coppia di processi si dà spesso il nome di "**qualifica**" o "V&V". Il loro obiettivo è assicurare la qualità del prodotto durante il suo ciclo di vita.

Questi due processi rispondono a due domande separate:

- Verifica: Did I build the system right?
- Validazione: Did I build the right system?

La **verifica** attiene alla coerenza, completezza e correttezza del prodotto. È un processo che si applica ad ogni "segmento" temporale di un progetto (ad ogni prodotto intermedio) per accertare che le attività svolte in tale segmento non abbiano introdotto errori nel prodotto.

La **validazione**, invece, non si applica ad un particolare segmento temporale ma è una conferma finale, una *self-fulfilling prophecy* che accerta la conformità di un prodotto alle attese; essa fornisce una prova oggettiva di come le specifiche del prodotto siano conformi al suo scopo e alle esigenze degli utenti. La validazione, a differenza della verifica, coinvolge sempre il committente.

12.2 Forme di verifica

La verifica è un processo analitico. Si può fare in due forme:

- analisi statica (senza eseguire il software);
- analisi dinamica (eseguendo il software o una sua parte).

L'analisi statica viene fatta prima di quella dinamica: quest'ultima necessita di (una parte di) software eseguibile, mentre l'analisi statica si può applicare già a *frammenti* di prodotto.

12.3 Test

L'analisi dinamica viene effettuata tramite prove (***test***). Un test, per essere utile, dev'essere facilmente ripetibile; per questo, bisogna sempre:

- tener conto dell'ambiente (non solo dell'input);
- specificare le pre-condizioni e i comportamenti attesi;
- descrivere le procedure per eseguire il test e per analizzarne i risultati.

Per fare ciò, servono tre strumenti:

- Un **logger**, che scriva l'esito della prova in un file.
- Un **driver**, cioè un "pilota" che guidi l'esecuzione del test. Ogni test di unità dev'essere chiamato da qualcuno: costui è il driver.
- Uno **stub**, cioè un "calco" che sostituisca del codice non ancora scritto: una componente passiva fittizia che simula una parte del sistema.

Lo stub è quindi il duale del driver: mentre il primo simula le dipendenze della procedura testata (quelle che ne accrescono il *fan-out*, per capirci), il secondo simula un chiamante di tale procedura (che contribuisce al *fan-in*).

12.4 Tipi di test

Distinguiamo quattro tipi di test, a seconda del livello a cui vengono eseguiti nell'architettura:

- collaudo o test di accettazione (stabilito durante l'analisi del capitolato d'appalto);
- test di sistema (stabiliti durante l'analisi dei requisiti);
- test di integrazione (stabiliti durante la progettazione logica);
- test di unità (stabiliti durante la progettazione di dettaglio).

Nell'architettura di un software, l'unità è la più piccola quantità di software che conviene verificare da sola. Un **test di unità** è un'attività di analisi dinamica che verifica la correttezza del codice. Può essere responsabilità del programmatore che ha implementato l'unità, di un verificatore indipendente oppure (meglio!) di un automa. Questi test vengono svolti con un alto grado di parallelismo (rispecchiando lo stesso parallelismo che caratterizza il lavoro dei programmatori).

I **test di integrazione** servono per verificare il sistema in modo *incrementale*. Questi test stanno a livello di componente e integrano il funzionamento di più unità.

Il **test di sistema** serve ad accertare la copertura dei requisiti. È un'attività interna all'organizzazione, mentre il **collaudo** è la corrispondente attività esterna (supervisionata dal committente). Al collaudo segue il rilascio del prodotto e la fine della commessa (con eventuale manutenzione).

Infine, esistono anche i **test di regressione**: a seguito di un test andato male e di una relativa correzione, i test di regressione sono l'insieme di test necessari ad accertare he la correzione non causi errori nelle parti del sistema che dipendono da essa. La regressione può essere complicata e dev'essere studiata *ad hoc*.

12.5 Forme di analisi statica

L'analisi statica si applica ad ogni prodotto intermedio (non solo software) per tutti i processi attivi nel progetto. Distinguiamo due tecniche principali per fare analisi statica:

- Metodi di lettura (*desk check*), svolti da un umano che controlla coerenza, completezza e correttezza; impiegati solo per prodotti semplici.
- Metodi formali, svolti da macchine.

Tra i metodi di lettura, due importanti sono *inspection* (cerco cose specifiche) e *walkthrough* (attraversamento a pettine: non so cosa cerco ma cerco ovunque), che si completano a vicenda.

Inspection consiste nell'eseguire una lettura mirata, alla ricerca di errori noti. Si basa sull'individuazione di errori presupposti e fa *profiling* del prodotto in esame. Si suddivide nelle seguenti attività: pianificazione; definizione di una *lista di controllo*; lettura; correzione dei difetti. È una tecnica molto più rapida del *walkthrough*. In termini statistici, *inspection* può generare falsi negativi.

Walkthrough, invece, consiste nell'eseguire una lettura critica del prodotto in esame. È una lettura "a largo spettro", senza l'assunzione di presupposti. Si articola nelle seguenti attività: pianificazione; lettura; discussione; correzione dei difetti. Rispetto all'*inspection*, richiede più attenzione. In termini statistici, *walkthrough* può generare falsi positivi.

12.6 *Quality assurance*

Analisi statica e analisi dinamica sono parte dell'attività di *quality assurance* (cioè garanzia di qualità). Questa attività è un controllo che non viene fatto dopo ma *prima* di fare, per assicurare la qualità tempestivamente; viene svolta sui processi con i quali si sviluppa il prodotto (non sul prodotto stesso) ma si basa su un modello di qualità di prodotto (ad esempio ISO/IEC 9126).

13 Verifica e validazione: analisi statica

13.1 Premessa

Un software di buona qualità deve possedere le capacità funzionali specificate nei requisiti e le caratteristiche non funzionali necessarie al buon funzionamento del sistema. Bisogna quindi verificare che il software posseggna determinate caratteristiche di costruzione (progettazione, codifica, integrazione), d'uso e di funzionamento.

Tanto più un linguaggio di programmazione è espressivo, tanto tanto meno è verificabile. Nel fissare il liguaggio di programmazione (e nel sceglierne i costrutti),

occorre quindi trovare il giusto compromesso tra **funzionalità** (potere espressivo) e **integrità** (costo di verifica). Inoltre, è bene adottare uno standard di codifica che tenga conto delle esigenze di verifica (per esempio, vietando alcuni costrutti): così facendo, si rende possibile una verifica che non sia retrospettiva ma che accompagna la codifica.

Infatti, il costo di rilevazione e correzione di un errore è tanto maggiore quanto più avanzato è lo stadio di sviluppo. Ecco perché ci interessa accompagnare la produzione con la verifica, invece di posticipare la verifica il più tardi possibile.

13.2 Tracciamento

Parte fondamentale dell'analisi statica è il **tracciamento**. Il tracciamento è una verifica atta a dimostrare due caratteristiche di una soluzione:

- **Completezza** della soluzione: tutti i requisiti sono soddisfatti; matematicamente, vuol dire che la soluzione è *condizione sufficiente* per il problema.
- **Economicità** della soluzione: nessuna funzionalità superflua, nessun componente ingiustificato; matematicamente, vuol dire che la soluzione è *condizione necessaria* per il problema.

Il tracciamento ha luogo su ogni passaggio dello sviluppo (ramo discendente del modello a "V") e su ogni passaggio della verifica (ramo ascendente).

13.3 Tipi di analisi statica

L'analisi statica si effettua con diversi metodi:

- analisi di flusso di controllo;
- analisi di flusso dei dati;
- analisi di flusso dell'informazione;
- esecuzione simbolica;
- verifica formale del codice;
- verifica di limite;
- analisi d'uso dello stack;
- analisi temporale;
- analisi d'interferenza;
- analisi del codice oggetto.

L'analisi statica costruisce modelli astratti del software in esame; questi modelli considerano ogni programma come un grafo orientato e ne studiano i cammini possibili.

14 Verifica e validazione: analisi dinamica

14.1 Definizione e problemi

L'analisi dinamica non è altro che l'esecuzione di ***test***, cioè prove su del codice in esecuzione. Purtroppo nello sviluppo di un software il codice nasce molto tardi; per questo, il *testing* dev'essere rapido (efficiente) ed efficace. Un altro problema dei test è la loro **non esaustività**: è possibile eseguirli solo su un insieme finito di casi, che non sono quasi mai tutti i casi possibili.

La pianificazione del *testing* deve quindi avvenire il prima possibile. Il primo momento utile è la progettazione del sistema.

Il debugging *non* è verifica: esso nasce da un errore che si è manifestato inaspettatamente, mentre la verifica viene pianificata a monte dello sviluppo.

14.2 Terminologia

Compito del test è trovare errori nel codice. Distinguiamo tre livelli di errore:

- **Malfunzionamento** (*failure*): l'esecuzione è difforme dalle attese. I malfunzionamenti riguardano il comportamento del software.
- **Errore** (*error*): stato del sistema che, se attivato, produce un malfunzionamento; se l'errore non viene attivato, rimane nascosto ma esiste. Gli errori sono quindi stati del sistema.
- **Guasto** o difetto (*fault*): causa dell'errore; può essere un guasto del computer o del software. I guasti possono essere malfunzionamenti di un sotto-sistema: i sistemi software sono sistemi gerarchici (annidati).

Quindi: un guasto cause un errore, il quale può produrre un malfunzionamento. (Nota: il glossario IEEE non concorda pienamente con le definizioni sopra.)

Organizziamo il *testing* nelle seguenti classi:

- **caso di prova** (*test case*) — una tripla <ingresso, uscita, ambiente>;
- **batteria di prove** (*test suite*) — un insieme di casi di prova;
- **prova** — una procedura di prova e una batteria di prove.

14.3 Compromesso

Esiste un compromesso, come tra efficienza ed efficacia, tra il numero di test sufficienti a verificare il prodotto e lo sforzo allocato a progetto. Sul *testing* vale infatti la "legge del **rendimento decrescente**": man mano che aumento lo sforzo, il rendimento cresce inizialmente ma poi diminuisce sempre più. Questo avviene, ad esempio, quando un produttore aumenta la produzione e, a un certo punto, oltrepassa la domanda: i profitti iniziano ad essere negativi. Così, arriva un tempo in cui fare altri test non aggiunge null, cioè non trova errori (e la funzione primaria dei test è trovare errori).

14.4 Criteri guida per i test

Oggetto di una prova può essere:

- il sistema nel suo complesso (per i test di sistema);
- parti del sistema in relazione funzionale, d'uso, di comportamento o di struttura (per i test di integrazione);
- singole unità (per i test di unità).

L'obiettivo di ogni prova dev'essere specificato in modo chiaro per ogni caso di prova (*test case*): un test è buono se è ripetibile. Per essere ripetibile, ogni test deve stare anche allo stato, all'ambiente. Il Piano di Qualifica specifica quali e quante sono le prove da effettuare. I test non sostituiscono la progettazione; piuttosto, sono speculari ad essa.

Un test deve cercare di far fallire il software: dev'essere "cinico"! I test che falliscono devono essere eseguiti sempre, dal momento in cui sono falliti e il software è stato corretto: Any failed execution must yield a test case, to be permanently included in the project's test suite (Bertrand Meyer).

All'origine, un test va specificato in fase di progettazione; dopo essere stato implementato ed eseguito, esso va tenuto in un archivio, come documentazione dell'attività di *testing*.

14.5 Test di unità

I test di unità sono naturalmente più numerosi dei test di integrazione e di unità: circa due terzi dei difetti rilevati tramite analisi dinamica sono dovuti ai test di unità.

Un concetto fondamentale nei test di unità è quello di **copertura** (*coverage*). Con questo termine si intende il la percentuale di codice che un caso di prova è in grado di eseguire, cioè quanto codice sorgente è stato effettivamente attraversato durante il caso di prova; ad esempio, una copertura del 100% indica che l'esecuzione di un test ha coperto tutti i casi possibili del codice in esame.

Alcuni criteri notevoli di copertura sono i seguenti:

- *function coverage* — quante funzioni (sottoprocedure) sono state eseguite?
- *statement coverage* — quante istruzioni (righe di codice, all'incirca) sono state eseguite?
- *branch coverage* — quanti rami del programma sono stati eseguiti?
- *condition coverage* — quante espressioni logiche hanno assunto entrambi i valori possibili?

Di queste, le più importanti sono lo *statement coverage* e, ancor più, il *branch coverage*. È sempre bene che i test di unità coprano il codice al 100% rispetto a questi ultimi due criteri; tuttavia, va ricordato che la copertura totale del codice non assicura l'assenza di difetti! Un criterio ancora più forte del *branch coverage* è il MC/DC (*Modified Condition/Decision Coverage*).

Distinguiamo due categorie di test di unità:

- Un **test funzionale** è un test a scatola chiusa (*black box*). Fa riferimento alla specifica di un'unità e osserva il suo comportamento dal di fuori; dati in ingresso che producono un medesimo comportamento funzionale costituiscono una classe di equivalenza e sono un caso di prova.
- Un **test strutturale** è un test a scatola aperta (*white box*): verifica la logica interna del codice dell'unità. Persegue la massima copertura del codice sorgente; dati in ingresso che attivano un medesimo percorso costituiscono un caso di prova.

Dobbiamo eseguire i test funzionali prima di quelli strutturali, se non vogliamo rischiare di analizzare una struttura che non svolge il compito giusto. I test funzionali vanno sempre integrati con test strutturali.

14.6 Test di integrazione

I test d'integrazione fanno parte di un processo più ampio che è quello dell'integrazione delle parti del sistema. Le parti vanno integrate secondo una strategia. Ad esempio è sempre bene assemblare le parti in modi incrementale (quindi reversibile), seguendo le dipendenze nell'architettura: aggiungendo una parte nuova ad un insieme ben verificato, i difetti rilevati in un test d'integrazione saranno probabilmente dovuti alla parte nuova, facilitando così la ricerca di quale parte sia da correggere.

Basandoci sul fatto che i sistemi software sono (al giorno d'oggi) sistemi gerarchici, possiamo individuare due principali **strategie d'integrazione**:

- Dal basso (bottom-up): si sviluppano e si integrano prima le parti con minore dipendenza funzionale (*fan-out*) e maggiore utilità (*fan-in*). Così facendo, si "risale" l'albero delle dipendenze. Si economizzano molti stub ma le funzionalità di alto livello compaiono più tardi.
- Dall'alto (top-down): si sviluppano prima le parti più esterne (di alto livello, con molte dipendenze) e poi si scende nell'albero delle dipendenze. Qui le funzionalità di alto livello vengono verificate sin da subito e si può mostrare al committente una bozza del sistema.

I test d'integrazione si applicano alle componenti specificate in progettazione architetturale; perciò, questi test rilevano difetti di progettazione. L'integrazione delle componenti costituisce il sistema completo.

Quanti test d'integrazione è bene fare? tanti quante sono le interfacce nell'architettura del sistema: i test d'integrazione devono accertare che i dati scambiati attraverso ciascuna interfaccia siano conformi alla propria specifica.

14.7 Test di sistema

I test di sistema verificano il comportamento del sistema rispetto ai suoi **requisiti**. Essi sono inerentemente funzionali: non hanno bisogno di conoscere la logica interna del sistema.

15 Metodi e obiettivi di quantificazione

15.1 Misurare

Misurare è un'attività fondamentale che compiamo quotidianamente: You can't control what you can't measure afferma Tom DeMarco, informatico e ingegnere del software. Misurare serve a conoscere, a valutare e a decidere; è un'attività che è alla base della cibernetica.

Purtroppo le misure rischiano di essere distorte, a causa di semplificazioni e imprecisioni. Per accertare la bontà di una misura essa dev'essere oggettiva, cioè deve garantire:

- ripetibilità
- confrontabilità
- confidenza

15.2 Definizioni

Definiziamo la **misurazione** come il processo che assegna numeri o simboli a entità del mondo reale. Risultato di una misurazione è una **misura**. Infine, definiamo una **metrica** come un insieme di regole atte all'interpretazione di un sistema di unità di misura; ogni metrica fornisce anche delle procedure per compiere misurazioni.

15.3 Misurazione del software

Nella produzione di software, le metriche sono strumenti di valutazione e controllo. Cosa ci interessa misurare in questo ambito? ci interessa valutare lo stato di:

- processi (per valutarne la qualità);
- progetti (per effettuare stime, preventivi e consuntivi);
- prodotti (per valutarne la qualità);
- risorse (per capirne il consumo).

Gli attributi a cui le misurazioni assegnano valori stanno in due categorie:

- gli attributi interni (riguardano la qualità interna) sono misurabili rispetto alle entità;
- gli attributi esterni (riguardano la qualità esterna) sono misurabili rispetto all'ambiente.

La misurazione rischia di essere retrospettiva; ci interessa, invece, avere misure che siano il più vicino possibile al tempo di correzione: si deve misurare in modo **proattivo**, non reattivo.

Anche per la misurazione, ISO ha uno standard: ISO/IEC 15939. Questo standard afferma, a riguardo dei **bisogni informativi**, the information needs are based on goals, constraints, risks, and problems which originate from the technical and mgmt process. CMMI suggerisce di determinare tali bisogni nelle seguenti aree:

- gestione dei requisiti;
- progettazione e implementazione;
- verifica e validazione;
- *quality assurance*;
- gestione della configurazione;
- gestione di progetto;
- analisi dei rischi e analisi delle decisioni;
- allenamento.

Ad esempio, il responsabile di progetto può essere interessato a misurare il *lead time* (cioè il tempo tra un ordine di sviluppo e la relativa consegna); e il *lead time* è inversamente proporzionale alla grandezza di ogni requisito, per cui un analista può essere interessato a misurare quanto sono grandi i requisiti, oltre a sapere quanto sono verificabili o quanto il prodotto soddisfa i requisiti.

Quando si trova un difetto, l'istinto sarebbe di correggerlo subito. Tuttavia sarebbe bene, prima, valutare il modo migliore per correggere il difetto con un buon rapporto beneficio/costi; questa è una misurazione che interessa chi si occupa della gestione dei cambiamenti (*change management*).

A Glossario

algoritmo Sequenza finita di passi per la risoluzione di un problema. Inglese: *algorithm*.

allocazione di risorse (per un progetto) Assegnare attività a ruoli e, poi, ruoli a persone.

ambiente di lavoro L'insieme di persone, di ruoli, di procedure e l'infrastruttura la cui qualità determina la produttività del progetto. Inglese: *work environment*.

amministratore di progetto (profilo professionale) Chi controlla 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. Inglese: *project administrator*.

analisi dei requisiti Definire cosa bisogna fare. Inglese: *requirements analysis*.

analisi dinamica Valutazione di un sistema (o di una sua componente) basata sul suo comportamento durante l'esecuzione. Inglese: *dynamic analysis*.

analisi statica Valutazione di un sistema (o di una sua componente) basata sulla sua forma, struttura, contenuto o documentazione. Inglese: *static analysis*.

analista (profilo professionale) Chi ha il compito di individuare, a partire dai bisogni del cliente, il problema da fornire ad un progettista; fa l'analisi dei requisiti. Inglese: *analyst*.

application logic La parte di un software che è specifica di quel software e non è intesa per essere riusata in altri software.

architettura La struttura organizzativa di un sistema o componente. Inglese: *architecture*.

attività Parte di un processo che dev'essere compiuta entro un determinato periodo di tempo. Inglese: *activity*.

baseline Nel ciclo di vita di un progetto, punto d'arrivo tecnico dal quale non si retrocede.

basso accoppiamento Minimizzazione delle dipendenze tra i vari componenti di un sistema. Inglese: *loose coupling*.

batteria di prove Insieme di casi di prova. Inglese: *test suite*.

best practice La prassi che, per esperienza e per studio, abbia mostrato di garantire i migliori risultati in circostanze note e specifiche.

bibliografia Elenco delle fonti di un documento. Inglese: *bibliografy*.

budget Tempo e denaro a disposizione.

business logic La parte di un software che ha a che fare con il dominio applicativo del software; questa parte è tipicamente riusabile e quindi condivisa tra diversi software che operano nello stesso dominio.

caos Contrario di organizzazione. Inglese: *chaos*.

caso d'uso Insieme di scenari che hanno in comune un obiettivo per un utente. Inglese: *use case*.

caso di prova Terna di valori (input, output, ambiente) che specifica il comportamento che un sistema (o parte di esso) deve avere in un caso specifico. Inglese: *test case*.

ciclo di vita (di un prodotto) Insieme degli stati che il prodotto assume, dal concepimento al ritiro. Inglese: *software product life cycle*.

ciclo di vita dello sviluppo (di un prodotto) Parte del ciclo di vita di un prodotto che riguarda il suo sviluppo. Inglese: *software development life cycle*.

ciclo PDCA (o ciclo di Deming) Schema iterativo di auto-miglioramento che consiste di quattro punti: Plan (individuare obiettivi di miglioramento), Do (eseguire ciò che si è pianificato), Check (verificare se ha funzionato) e Act (agire per correggersi). Inglese: *PDCA cycle*.

CoCoMo (Constructive Cost Model) Modello per la stima dei costi di un progetto, in tempo/persona. Inglese: *CoCoMo*.

coerenza L'esser composto da parti che non sono in disaccordo tra loro, cioè non affermano cose che si contraddicano. Inglese: *consistency*.

coesione Caratteristica di un sistema per la quale parti associate concorrono agli stessi obiettivi. Inglese: *cohesion*.

compito Parte di un'attività. Inglese: *task*.

componente Insieme di unità funzionalmente coese; parte di un sistema. (In Java, rappresentabile da uno o più package.) Inglese: *component*.

configuration item (CI) Parte della configurazione di un software. Inglese: *configuration item*.

configurazione Di quali parti si compone un prodotto e il modo in cui esse stanno assieme. Inglese: *configuration*.

controllore della qualità (profilo professionale) Funzione aziendale (e non ruolo di progetto) che accerta la qualità dei prodotti.

copertura del codice Percentuale di codice sorgente eseguito durante un caso di prova. Inglese: *code coverage*.

criticità Distanza troppo breve tra attività dipendenti.

design pattern Soluzione progettuale generale ad un problema ricorrente.

diagramma dei casi d'uso Grafo orientato che mostra gli attori, i casi d'uso e le relazioni tra essi: ogni nodo è un attore o un caso d'uso; ogni arco è una comunicazione tra un attore e un caso d'uso oppure una relazione (di estensione, inclusione o generalizzazione) tra due casi d'uso o tra due attori. Inglese: *use case diagram*.

diagramma di Gantt Diagramma che rappresenta la durata, la sequenzialità e il parallelismo delle attività di un progetto. Inglese: *Gantt diagram*.

diagramma PERT (Project Evaluation and Review Technique) Rete che rappresenta le dipendenze temporali (e le criticità) tra attività di un progetto. Inglese: *PERT diagram*.

diagramma WBS (Work Breakdown Structure) Diagramma che decompone in modo gerarchico le attività di un progetto in sotto-attività (coese ma non necessariamente sequenziali). Inglese: *WBS diagram*.

disciplinato Che segue le norme (anzi, la best practice). Inglese: *disciplined*.

documentazione Tutto ciò che documenta le attività di un progetto. Inglese: *documentation*.

efficacia Conformità alle attese. Inglese: *effectiveness*.

efficienza Contenimento dei consumi per raggiungere un obiettivo. Inglese: *efficiency*.

errore Stato del sistema che, se attivato, produce un malfunzionamento. Inglese: *error*.

fase (di un ciclo di vita) Durata temporale (che non si ripete) entro uno stato del ciclo di vita e un altro. Inglese: *phase*.

framework Struttura di supporto su cui un software può essere organizzato e progettato.

gestione della qualità L'insieme dei processi che assicurano che prodotti e implementazioni di processi rispettino gli obiettivi di qualità (di un'organizzazione) e soddisfino gli stakeholder. Inglese: *quality management*.

glossario Elenco dei significati dei termini più rilevanti di un documento. Inglese: *glossary*.

guasto Causa di un errore. Inglese: *fault*.

impegno Inglese: *commitment*.

incarico Inglese: *assignment*.

indice analitico Elenco ordinato delle corrispondenze tra particolari termini importanti di un documento e la loro ubicazione in esso. Inglese: *index*.

indice generale Elenco delle parti di un documento. Inglese: *table of contents*.

infrastruttura (di un progetto) Tutte le risorse hardware e software del progetto.

ingegneria L'applicazione di princìpi scientifici e matematici per scopi pratici. Inglese: *engineering*.

ingegneria del software Applicazione di un approccio sistematico, disciplinato e quantificabile allo sviluppo, al funzionamento e al mantenimento del software. Inglese: *software engineering*.

inspection Tecnica di analisi statica che consiste nell'eseguire una lettura mirata, alla ricerca di errori noti.

integrazione continua Pratica di sviluppo in cui i membri di un progetto integrano il loro lavoro frequentemente (quotidianamente) in modo automatizzato. Inglese: *continuous integration*.

LaTeX Sistema di composizione tipografica che utilizza TeX come motore.

legge di Parkinson Work expands to fill the time available. Inglese: *Parkinson's law*.

malfunzionamento Esecuzione di un software difforme dalle attese. Inglese: *failure*.

marcatore Istruzione che un programma deve eseguire per trattare nel modo specificato dall'utente una porzione di testo specificata. Inglese: *mark-up*.

metodo di lavoro Metodo di lavoro. Inglese: *way of working*.

metrica L'interpretazione di un sistema di unità di misura. Inglese: *metric*.

milestone Punto nel tempo associato ad un valore strategico.

misura Risultato di una misurazione.

misurazione quantitativa L'uso di una metrica per assegnare un valore su una scala predefinita. Inglese: *quantitative measurement*.

modello Astrazione della realtà. Inglese: *model*.

modulo L'elemento atomico dell'architettura di un software; tipicamente una classe o un interfaccia. Inglese: *module*.

organizzazione Aggregato di persone [?] che agiscono in modo sistematico, disciplinato e quantificabile; contrario di caos. Inglese: *organization*.

pianificazione Organizzare e controllare tempo, risorse e risultati. Inglese: *planning*.

prassi Modo di fare. Inglese: *practice*.

processo (ingegneristico) Insieme di attività correlate e coese che trasformano ingressi in uscite, consumando risorse nel farlo. Inglese: *process*.

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 obiettivi prefissati (processo calato nella realtà aziendale).

processo software Processo che porta ad un prodotto software. Inglese: *software process*.

produttività Rapporto tra valore e costo. Inglese: *productivity*.

profilo professionale 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. Inglese: *professional profile*.

progettazione Definizione dell'architettura, delle componenti, delle interfacce e delle altre caratteristiche di un sistema o componente. Inglese: *design*.

progettazione architetturale Definizione delle componenti e di come esse sono organizzate in un sistema. Inglese: *architectural design*.

progettazione di dettaglio Definizione del comportamento delle componenti di un sistema, con un livello di dettaglio tale per cui le componenti possano essere implementate. Inglese: *detailed design*.

progettista (profilo professionale) Chi sintetizza una soluzione a partire dalle specifiche di un problema già analizzato. Inglese: *designer*.

progetto Insieme di compiti da svolgere in modo collaborativo a fronte di un incarico (che diventa poi un impegno). Inglese: *project*.

programmatore (profilo professionale) Chi implementa una parte della soluzione dei progettisti. Inglese: *programmer*.

protocollo Accordo di interfacce. Inglese: *protocol*.

qualifica Verifica e validazione ("V&V"), cioè quei processi che assicurano la qualità di un prodotto durante il suo ciclo di vita.

qualità L'insieme delle caratteristiche di un'entità che ne determinano la capacità di soddisfare esigenze espresse e implicite. Inglese: *quality*.

quality assurance Insieme di attività che valutano i processi con i quali un prodotto viene sviluppato.

quantificabile Esprimibile in modo quantitativo. Inglese: *quantifiable*.

raccomandazione Norma di progetto suggerita, non sottoposta a verifica. Inglese: *recommendation*.

ramo (di un repository) Insieme di versioni di file sorgente in evoluzione. Inglese: *branch*.

regola Norma di progetto sottoposta a verifica. Inglese: *rule*.

rendimento decrescente, legge del Man mano che si aumenta lo sforzo, il rendimento cresce inizialmente ma poi diminuisce sempre più. Inglese: *diminishing returns*.

repository Base di dati centralizzata nella quale risiedono, individualmente, tutti i CI di ogni baseline nella loro storia completa.

requisito Bisogno da soddisfare o vincolo da rispettare. Inglese: *requirement*.

requisito di processo Vincolo sullo sviluppo del prodotto.

requisito di prodotto Bisogno o vincolo sul prodotto da sviluppare.

requisito di sistema Definizione formale e dettagliata di una funzione del sistema. Inglese: *system requirement*.

requisito funzionale (di un prodotto software) servizio che il prodotto deve fornire. Inglese: *functional requirement*.

requisito non funzionale (di un prodotto software) vincolo su uno o più servizi che il prodotto fornisce. Inglese: *non-functional requirement*.

requisito utente Richiesta generale, ad alto livello. Inglese: *user requirement*.

responsabile di progetto (profilo professionale) Chi pianifica il progetto, assegna le persone ai ruoli giusti e rappresenta il progetto presso il fornitore e il committente. Inglese: *project manager*.

rete Grafo orientato. Inglese: *network*.

revisione esterna Ispezione ufficiale di un prodotto condotta da un'organizzazione indipendente da chi ha sviluppato il prodotto. Inglese: *audit*.

revisione interna Ispezione di un prodotto interna all'organizzazione che lo sviluppa. Inglese: *joint review*.

rischio Opposto di opportunità. Inglese: *risk*.

ritiro (di un prodotto) Momento in cui il prodotto cessa di essere seguito dai suoi creatori. Inglese: *retirement*.

ruolo Funzione aziendale assegnata a progetto; identifica capacità e compiti. Inglese: *role*.

scenario Sequenza di passi che descrive un esempio di interazione con un sistema.

sistema Insieme di componenti organizzati per compiere una o più funzioni. Inglese: *system*.

sistematico Che abbraccia un metodo. Inglese: *systematic*.

slack time Quantità di tempo tra la data minima a partire da cui un evento può accadere e la data massima oltre la quale esso ritarda gli eventi successivi.

sommario Breve riassunto del contenuto di un documento. Inglese: *abstract*.

SQL (Structured Query Language Linguaggio di programmazione dichiarativo basato sull'algebra relazionale che serve a creare, manipolare e interrogare basi di dati relazionali. Inglese: *SQL*.

stakeholder Persona a vario titolo coinvolta nel ciclo di vita di un software che ha influenza sul prodotto o sul processo.

standard di processo 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. Inglese: *process standard*.

strumento Insieme di concetti e di metodi, con delle tecnologie di supporto. Inglese: *tool*.

tecnica Ricetta applicata agli strumenti disponibili; modo con cui si usa uno strumento. Inglese: *technique*.

tecnologia Strumento sul quale si opera. Inglese: *technology*.

test Attività di analisi dinamica che osserva i risultati dell'esecuzione di un sistema (o parte di esso) sotto determinate condizioni.

test di integrazione Test che verifica la correttezza dell'integrazione di un insieme di unità in una componente del sistema in esame. Inglese: *integration test*.

test di sistema Test che verifica la copertura dei requisiti da parte del sistema in esame. Inglese: *system test*.

test di unità Test che verifica la correttezza di una singola unità del sistema in esame. Inglese: *unit test*.

test di validazione Test che verifica il soddisfacimento del capitolato d'appalto da parte del sistema in esame. Inglese: *validation test*.

test funzionale Test di unità che analizza la logica interna del codice di un'unità. Inglese: *functional test*.

test strutturale Test di unità che analizza soltanto il comportamento dell'unità e non la sua logica interna. Inglese: *structural test*.

TeX Linguaggio formale di composizione tipografica.

UML (Unified Modelling Language) Famiglia di notazioni grafiche che si basano su un singolo meta-modello e servono a supportare la descrizione e il progetto dei sistemi software. Inglese: *UML*.

unità Insieme coeso di moduli, appaltabili in realizzazione a un singolo programmatore; non ha sempre un corrispondente diretto in un linguaggio di programmazione. È anche la più piccola quantità di software che conviene verificare da sola. Inglese: *unit*.

validazione La garanzia che un prodotto soddisfi i requisiti da cui è nato. Inglese: *validation*.

valutazione Verifica quantificata. Inglese: *evaluation*.

verifica Valutare se un prodotto soddisfa requisiti, regole o altre condizioni necessarie. Inglese: *verification*.

verificatore (profilo professionale) Chi verifica il lavoro dei programmatori.

versione (di un CI) Istanza identificata di un CI nel tempo. Inglese: *version*.

walkthrough Tecnica di analisi statica che consiste nell'eseguire una lettura critica, ad ampio spettro, senza l'assunzione di presupposti.

Riferimenti bibliografici

[1] Ian Sommerville, *Software Engineering* (9th ed.), Addison-Wesley, 2011.

[2] IEEE, *Guide to the Software Engineering Body of Knowledge* v3.0, ed. P. Bourque R. Fairley, 2014.

[3] Tullio Vardanega, Lucidi per le lezioni dell'A.A. 2016-2017.

[4] Riccardo Cardin, Lucidi per le lezioni dell'A.A. 2016-2017.

[5] Anonimo, *Riassunto IS*, reperito su MEGA.

[6] Martin Fowler, *UML Distilled*, reperito sul web.