Psicologia ed economicità del test

Ercole Colonese
Comitato direttivo “Qualità del Software e dei Servizi IT”. 2010
Pubblicato da AICQ su Qualità, N. 3 2010

L’attività di “software testing” rappresenta il processo con il quale si esegue e si valuta, manualmente e automaticamente, un programma, un prodotto o un sistema per verificare se soddisfa i requisiti specificati e per identificare le differenze tra i risultati attesi e quelli ottenuti (cioè per rilevare le anomalie ed i difetti).

Il test è un’attività generale che si svolge lungo l’intero ciclo di vita del software con enfasi e obiettivi diversi. Nel linguaggio comune dello sviluppo software i due termini “test” e “collaudo” sono utilizzati a volte in maniera distinta e a volte in maniera analoga. Il primo (test) indica le prove effettuate dal gruppo di sviluppo durante la realizzazione del progetto ed il secondo (collaudo) quelle effettuate dal cliente in fase di accettazione del prodotto finale. Generalmente ai due termini si da lo stesso significato: “collaudo del software” (o “test del software”) è l’espressione con cui si indicano sia i test eseguiti dallo sviluppo sia il collaudo vero e proprio eseguito dal cliente.

L’importante è stabilire (e concordare) sugli obiettivi del test. L’attività ha un duplice obiettivo: verificare che il software sviluppato indirizzi i requisiti specificati e scoprire gli errori in esso presenti (e consentire la rimozione) prima che sia rilasciato in esercizio.

Psicologia del test

A proposito del duplice obiettivo dei test di verificare l’aderenza ai requisiti e di rimuovere gli errori presenti nel software è interessante riportare quanto afferma Myers[1] che di test sicuramente se ne intende molto. Egli afferma che l’obiettivo del test dovrebbe essere principalmente quello di scoprire gli errori presenti nel software piuttosto che quello di validare l’aderenza ai requisiti e/o alle specifiche.

In tal senso, enfatizza il fatto che l’attività di test ha una componente psicologica molto importante.

Una persona agisce generalmente in base agli obiettivi che gli vengono assegnati. Un tester che ha l’obiettivo di scoprire gli errori affronta il proprio lavoro con una motivazione ed un impegno tutto profuso alla ricerca dei “bug”. Quindi l’aspetto psicologico influenza l’atteggiamento e di conseguenza il risultato. Trovare errori diventa prioritario. Con il tempo si consoliderà sempre di più tale atteggiamento, si affineranno le tecniche e miglioreranno i risultati.

Una persona, viceversa, con l’obiettivo di verificare l’aderenza del software ai requisiti e alle specifiche ha un atteggiamento mentale totalmente diverso. La sua psicologia lo porta a confermare la validità del software, a dimostrare l’aderenza ai requisiti, a trovare la corrispondenza alle specifiche. Il suo atteggiamento lo porterà con il tempo a migliorare tali capacità e a trovare sempre di più elementi che dimostrino l’aderenza agli obiettivi.

La psicologia del tester, dice ancora Myers, è quindi influenzata dagli obiettivi imposti: ricerca di errori o conferma dell’aderenza ai requisiti. Nel primo caso di parla atteggiamento “distruttivo” nel secondo caso di atteggiamento “costruttivo”. Il termine distruttivo è ovviamente da intendersi positivamente ai fini del progetto (“fare le pulci”, “scovare gli errori”, “dimostrare cosa non funziona”, ecc.).

Formazione, acquisizione di metodi e tecniche, utilizzo di strumenti sarà perciò guidata da tale approccio mentale.

E’ bene, quindi, tener presente tali fattori in fase di organizzazione del gruppo di test, in fase di assegnazione degli obiettivi del test ed in fase di progettazione dei casi di prova.

Economicità del test

Anche alla luce di quanto espresso in tema di psicologia del testing, si descrivono brevemente qui di seguito le problematiche più importanti che il testing è chiamato ad indirizzare (e risolvere).

Facciamo però, prima, alcune considerazioni di carattere tecnico (ed economico) specifiche del software.

Il software è il risultato di un’attività creativa ed umana allo stesso tempo soggetta, come ogni altra attività umana, ad errori. La prima considerazione da fare, quindi, è che ogni software contiene un numero più o meno grande di errori. Tecnicamente possiamo invece dire che il software è un insieme di istruzioni che, eseguite in sequenze ben determinate, permettono di elaborare dati (input) e produrre risultati (output).

Un software applicativo (un sistema, un componente, ecc.) è un insieme di migliaia di istruzioni (anche milioni) con un numero di combinazioni di percorsi possibili (miliardi e miliardi) che dipendono, a volte, da piccoli controlli e decisioni disseminati lungo il codice (istruzioni di controllo, loop, selezioni, impostazioni di valori, ecc.). Eseguire dunque dei test in numero sufficiente per coprire “tutte” le combinazioni possibili dei percorsi presenti nel codice è improponibile (oltre che irrealizzabile). Si pone allora il problema dell’economicità dei test: eseguire tanti test quanti possano assicurare obiettivi realistici, sia in termini di copertura dei requisiti e rimozione degli errori, sia in termini di impegno di risorse e di tempo.

Il test del software rappresenta quindi un problema (ma anche una necessità) sia di carattere psicologico (efficacia del test) sia di carattere economico (efficienza del test). In tal senso vanno interpretati gli elementi che seguono.

Selezione dei test

La selezione dei test permette di decidere quali casi di prova, all’interno di un insieme definito (*test suite*), sia opportuno eseguire. I criteri di selezione permettono di verificare il livello di copertura dei test rispetto ai requisiti, di ottimizzare i tempi ed i costi eliminando i casi di test ridondanti, e di decidere quando fermare le attività di testing.

Efficacia ed obiettivi dei test

L’attività di testing permette di osservare il comportamento del software quando viene eseguito in determinate condizioni (ambiente di test, dati di input, sequenza delle operazioni, ecc.). La scelta delle condizioni in cui eseguire i test dipende dagli obiettivi del testing: è solo alla luce degli obiettivi da perseguire, quindi, che si può valutare l’efficacia dei test.

Test per scoprire i difetti

I test per la ricerca degli errori hanno successo se sono capaci di indurre il sistema a fallire. Questo tipo di test è completamente differente da quello in cui si vuole verificare la corrispondenza del software alle sue specifiche, come si è detto a proposito della psicologia del testing. In questo secondo caso, infatti, il test ha successo se induce il software a completare la funzione attivata senza che si verifichi alcun errore.

Il problema dell’oracolo

Un oracolo è un agente (umano o meccanico) in grado di decidere se un programma si comporta correttamente durante l’esecuzione di un determinato test; di conseguenza, è in grado di emettere un verdetto di “ok” o “not ok” (*pass or fail*). Realizzare un oracolo automatico può risultare piuttosto difficile e costoso. Per questo motivo è poco frequente e si demanda alla valutazione umana (il tester) se l’esito delle prove sia stato positivo o meno.

Limiti teorici e pratici del testing

La teoria sul testing mette in guardia contro l’ingiustificata fiducia verso i risultati dei test. Ad esempio, un famoso aforisma di Dijkstra[2] dice che “il test di un programma può dimostrare la presenza di errori ma mai la loro assenza”[3]. La ragione di tale asserzione è che il test di un

[1] Myers G.J. & All, *The Art of Software Testing*, Second Edition, 2004.

[2] Edsger Wybe Dijkstra (Rotterdam, 11 maggio 1930 – Nuenen, 6 agosto 2002) è stato un informatico olandese di grande valore. E’ stato insignito nel 1972 del Turing Award (il Nobel dell’Informatica). Tra i suoi maggiori contributi all’informatica più conosciuti ci sono l’algoritmo che porta il suo nome (l’algoritmo permette di trovare i cammini minimi in un grafo), la tecnica del semaforo (struttura dati per la gestione di risorse condivise da un sistema) e la sua acerrima battaglia contro l’istruzione GOTO.

[3] Edsger Dijkstra, *The threats to computing science*, 1972.

software non può essere mai esaustivo. Da ciò deriva la necessità di considerare il test come un'attività nell'ambito della gestione del rischio e come tale essere considerata all'interno della strategia di *Risk Management*.

Il problema dei cammini non percorribili

I "cammini non percorribili" sono i flussi interni del software che non possono essere percorsi da alcuna combinazione di dati forniti in input al programma. Il problema è particolarmente rilevante quando si realizzino test automatici del codice. Una soluzione possibile è l'esecuzione di test unitari con la tecnica *White - box* in cui si interviene direttamente sul codice sorgente forzando il software a percorrere determinati cammini.

Testabilità del software

L'espressione "testabilità del software" ha due significati diversi: il primo si riferisce al livello di facilità con cui il software è capace di soddisfare i criteri di copertura dei test; il secondo si riferisce alla possibilità (probabilità), misurata statisticamente, che il software possa rivelare un difetto, se esso esiste, durante l'esecuzione dei test.

In termini più semplici, possiamo esprimere i due concetti come segue: 1) un software è detto testabile se possiamo immaginare almeno un caso di test in grado di esercitarlo; 2) un software è detto testabile se esiste la probabilità che esso possa fallire nell'eseguire almeno un caso di test.

Il primo significato è particolarmente importante in fase di definizione dei requisiti. Un requisito, infatti, deve essere "chiaro", "completo" e "testabile". Spesso, purtroppo, nei progetti i requisiti sono descritti in maniera poco chiara, confusa, ambigua, parziale, generica e "non testabile". Le conseguenze sono facilmente immaginabili. Il presente lavoro ha la speranza, in tal senso, di contribuire a creare un po' di buon senso nei nostri progetti.

Relazioni tra test e altre attività

Il test del software è strettamente legato ad altri temi: tecniche di controllo della qualità tramite attività di test statico (ispezioni e revisioni tecniche), debugging, programmazione, certificazione. Ciascuna di esse ha un legame stretto con il testing e con esso si integra per fornire un approccio completo; allo stesso tempo, ciascuna di esse, influisce sui costi del progetto. Il giusto compromesso tra costi e risultati è la formula corretta per decidere "cosa e quanto fare".

Le **ispezioni** (o **revisioni tecniche**) sono tecniche di controllo della qualità applicate al software "prima che sia stato sviluppato"; perciò sono anche note come "test statico"; si effettuano sui requisiti, sulla progettazione (architettura, specifiche tecniche), sul codice (inteso come documentazione), sulla documentazione utente (manuali, messaggi, ecc.), sulla pianificazione del progetto (piani) e anche sui test stessi (casi di prova, matrice di test). L'attività è propedeutica e complementare al testing "dinamico": scopre gli errori logici prima che questi possano proliferare in maniera incontrollata nel codice[4]. Permette di sottoporre a test un software possibilmente scremato dai tanti errori di interpretazione dei requisiti, di progettazione e di codifica.

Il **debugging** è la tecnica principe utilizzata dagli sviluppatori per analizzare i problemi, scoprire la causa dei difetti e correggere il codice. Essa è quindi strettamente legata al testing. In maniera efficace, anche se molto semplificata, possiamo dire che il test ci permette di scoprire gli errori, il debugging a correggerli!

La **programmazione** ed il test sono attività antitetiche ma imprescindibili l'una all'altra: si può testare solo un software sviluppato; non ha senso un software non testato. Ma la relazione è ancora più profonda. La strategia di test dipende strettamente dalla strategia di sviluppo e di integrazione del software. La bontà della programmazione ci dice quali e quanti test eseguire.

La **certificazione** è la formalizzazione dei risultati positivi del collaudo eseguito a fronte di specifiche definite.

Principi generali del test

Tornando agli aspetti psicologi del test, vediamo alcuni principi basilari dell'attività che influenzano anche l'aspetto organizzativo. Tali principi sono stati riassunti in maniera egregia da G.J. Myers[5] già nella versione del 1987 del suo libro. Qui ne diamo una nostra interpretazione, anche in base all'esperienza acquisita, in modo sintetico.

Il test non dovrebbe mai essere effettuato da chi sviluppa il software

Il responsabile del progetto (*Project Manager*) ed il gruppo di sviluppo non dovrebbero mai effettuare direttamente il test per due motivi: uno di carattere organizzativo ed uno psicologico. Lo sviluppo (il capo progetto in particolare) è fortemente orientato a garantire il rispetto dei tempi e dei costi del progetto, anche a discapito dell'affidabilità e della correttezza delle applicazioni sviluppate, cioè a discapito dei test. Ciò non è corretto ma è quello che spesso avviene nei progetti in difficoltà (ritardo nei tempi di consegna o superamento del budget). Inoltre, il programmatore ha un'attitudine mentale, un approccio psicologico, di tipo costruttivo (processo di sviluppo) e non distruttivo[6] (processo di test) come invece sarebbe necessario per scoprire il maggior numero di errori.

Ottimizzare il test in base agliobiettivi

Il software, anche quello più semplice, è una combinazione di istruzioni eseguite secondo innumerevoli percorsi diversi (*path*) che dipendono dai valori di variabili del programma, da condizioni verificate, ecc. Si può dimostrare che, anche per un semplice programma di poche decine di istruzioni, un loop e qualche istruzione condizionale, può richiedere un numero milioni e milioni di test: dunque impossibili da pianificare ed eseguire. Occorre dunque considerare il test come un numero limitato di tentativi di prova con cui coprire tutti i requisiti del software e scoprire il maggior numero possibile di errori. Si tratta dunque di "un'arte"[7] che mira ad ottimizzare i test in modo da ottenerne il massimo beneficio con un ragionevole impegno di tempo e risorse (costi). Dunque un'attività creativa basata sull'esperienza e sull'adozione di metodi e tecniche specifiche.

Un caso di test[8] deve sempre specificare i dati di input ed il risultato atteso

Nel progettare un test di un componente software, è necessario indicare precisamente i dati di input in quanto essi definiscono le circostanze in cui esso sarà utilizzato (cioè lo scenario di utilizzo del software). Ogni combinazione di dati di input rappresenta una specifico scenario di utilizzo, diverso da tutti gli altri utilizzi. Analogamente, occorre definire in anticipo i relativi dati di output. Questo principio, che può sembrare ovvio a prima vista, è invece di primaria importanza poiché se il risultato di un caso di test non è predefinito, anche un risultato errato potrebbe essere interpretato come corretto sull'onda psicologica di un test completato senza errori.

Eseguire i casi di test anche per condizioni non valide, al limite ed inattese

L'approccio psicologico di molti tester è quello di eseguire i casi di test che dimostrino la correttezza funzionale del programma, ignorando le condizioni non valide, inattese o al limite. Purtroppo gli utenti finali commetteranno sicuramente degli errori nell'utilizzare l'applicativo e quindi indurranno il software a percorrere cammini non testati. In questo caso si dovrà correggere il software in esercizio con costi maggiori.

Considerare i casi di test come un investimento

---

[4] E. Colonese, *Il controllo qualità a supporto del business*, articolo pubblicato su AICQ Qualità On-Line n° 3, novembre 2007. Nell'articolo è descritto il tema della propagazione degli errori durante il ciclo di vita del software e le relazioni con la qualità del software ed il testing.

[5] G. J. Myers, *The Art of Software Testing*, John Wiley & Sons, New York, 1979 e 2004 (Second Edition).

[6] Il termine "processo costruttivo" è utilizzato per indicare le attività di sviluppo del software (il processo, cioè, con il quale si "costruisce" il software); il termine "processo distruttivo" per l'attività di testing è utilizzato anch'esso in senso positivo per indicare che il processo tende a trovare il maggior numero possibile di errori nel software. L'approccio è descritto nel paragrafo precedente a proposito della psicologia del testing.

[7] Sempre G.J. Myers parla di "dark art", cioè di "arte oscura" in quanto ancora oggi non tutte le organizzazioni software possiedono conoscenza e specializzazione in tale attività.

[8] "Caso di test" è l'elemento unitario dei test. Dall'inglese "Test case" è detto anche "caso di prova".

Ogni volta che si modifica un programma occorre testarlo; il test di regressione è il tipico caso. Esso dovrà essere eseguito in maniera rigorosa come il test originario. E' bene quindi conservare i casi di test realizzati; se registrati in forma elettronica e rieseguiti in maniera automatica, ancora meglio.

La probabilità di trovare errori in un software è proporzionale al numero di errori già trovati

In maniera semplicistica il concetto si può esprimere così: "Più errori si trovano e più errori si troveranno; meno errori si trovano e meno se ne troveranno". Il fenomeno è illustrato nella figura che segue. Nelle prime fasi di test il numero di errori trovati tende a crescere per poi diminuire sempre più man mano che il "residuo" di errori diminuisce. Il valore limite (*asintoto*) verso cui tende la curva di rimozione degli errori è determinato statisticamente in base all'esperienza su progetti simili.

La costruzione della "curva di rimozione degli errori" rappresenta una tecnica efficace per individuare il numero di errori residui nel software ad ogni momento delle attività di testing.

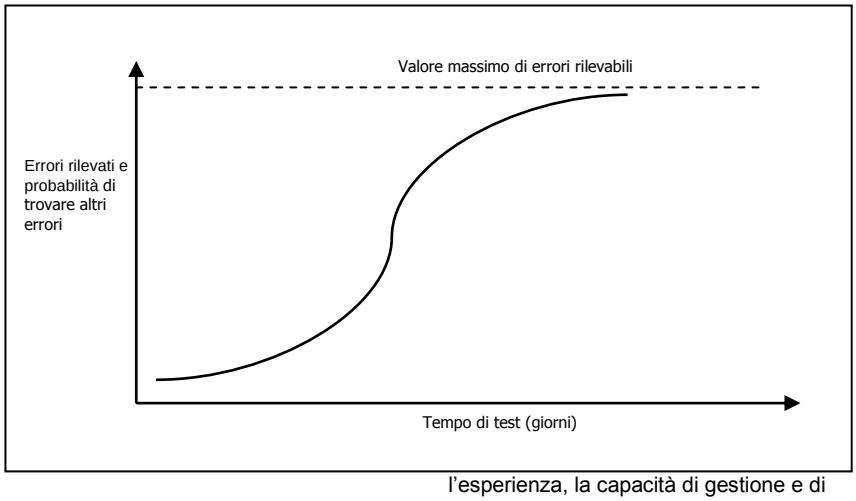

Figura 1. Relazione tra gli errori trovati e previsti.

Conclusioni finali

Riassumendo, possiamo concludere con due principi importanti:

1. Il test è un processo di esecuzione (vera o simulata) di un elemento software con l'obiettivo di verificare la conformità ai requisiti stabiliti e di scoprire gli errori presenti.
2. Il test è progettato per avere alta probabilità di scoprire gli errori presenti, anche quelli che risultano particolarmente nascosti finché il software non sarà utilizzato dagli utenti finali.

Composizione del gruppo di test

Un momento chiave nella gestione dei test è la fase di composizione del gruppo di lavoro. L'attività di testing (si è già detto) è molto diversa dalla programmazione e richiede competenze ed attitudini completamente diverse. Si tratta di due figure (il programmatore ed il tester) diverse e complementari. Solo apparentemente in conflitto, esse sono invece contrapposte perché è giusto che lo siano, è il ruolo che lo richiede. Alcuni pensieri circolanti nell'ambiente dello sviluppo software esprimono concetti negativi quali: il programmatore "crea" del software mentre il tester lo "distrugge". Esiste anche una corrente di pensiero, anche tra alti dirigenti, secondo la quale "il test è solo una spesa"; "essa non produce nulla"; "spende il proprio tempo solo per trovare errori senza mai costruire nulla", ecc.[9] Nulla di più sbagliato. Il presente lavoro, nel suo insieme, dovrebbe dissipare ogni dubbio e mostrare evidente l'importanza e la necessità del test.

Vediamo ora come è strutturato un gruppo di test. Come ogni gruppo di lavoro ha un suo responsabile (test manager, coordinatore dei test, team leader come meglio è opportuno chiamarlo) ed un insieme di persone preposte allo svolgimento delle attività. E' fuor di dubbio che il responsabile debba ricoprire il ruolo di coordinamento basando la propria leadership sulla competenza, l'esperienza, la capacità di gestione e di risoluzione delle situazioni critiche. Il gruppo di lavoro prevederà invece competenze diverse a seconda se si tratti di progettare i test piuttosto che predisporre gli ambienti di test oppure eseguire i casi di test.

Di seguito si fornisce una breve descrizione dei compiti, responsabilità e competenze richieste alle figure chiave di un gruppo di test. A seconda delle dimensioni e della complessità del progetto le figure riportate di seguito possono essere ricoperte da una o più persone a seconda delle necessità. Per esempio, in un gruppo di grandi dimensioni, esisteranno tutte le figure elencate, mentre in un progetto di piccole dimensioni la persona con maggiore competenza ed esperienza potrà ricoprire sia il ruolo di responsabile, sia quello di progettista, sia ancora quello di esecutore dei test. Tra questi due estremi potranno esserci tutte le variazioni necessarie ed adeguate al progetto.

Responsabile dei test (*Test Manager*)

Altri nomi usati per identificare la figura sono *Test Leader o Team Leader.* Ha la responsabilità di definire la strategia di test più adatta alle particolari necessità del progetto; di definire e coordinare le attività del gruppo di test, di valutare gli esiti dei test e lo stato di completamento fornendo report opportuni.

Le competenze principali richieste per ricoprire il ruolo sono:

- Project Management;
- Organizzazione / People Management;
- Metodologia di testing;
- Leadership;
- Comunicazione.

Progettista di test (*Test Designer*)

Altro nome utilizzato è *Specialista di test*. Ha il compito di progettare i test in base alla strategia, ai piani ed agli obiettivi di test, di produrre il materiale previsto (casi di test, matrice di test, dati di test, procedure di test), di fornire supporto al gruppo per i casi più complessi o critici (eseguendo, se necessario anche i casi di prova) e di valutare l'esito delle prove fornendo i dati necessari al Responsabile in fase di produzione della reportistica e di valutazione dei test.

Le competenze richieste dal ruolo sono molteplici:

- Conoscenza del business indirizzato dagli applicativi;
- Architetture tecnologiche e piattaforme applicative;
- Strategie di sviluppo e relative metodologie;
- Strategie di collaudo, ciclo di vita del test, metodi e strumenti a supporto del test;
- Pianificazione dei progetti;
- Ambienti di test e collaudo, hardware e software;
- Utilizzo di tecniche di analisi statistica;
- Gestione dei problemi;
- Gestione della configurazione.

Testatore (*Tester*)

Ha il compito di eseguire i casi di prova, di registrare l'esito delle prove ed i difetti e validarne le correzioni effettuate, di rapportare i risultati complessivi e di dettaglio dei test eseguiti.

[9] Joel Spolsky indica "Le cinque ragioni (sbagliate) per cui non hai un tester" – nel suo libro *Joel e il Software*, Apress - 2005. Le cinque ragioni (sbagliate) sono così espresse:

1. I bug sono conseguenza del lavoro di programmatori disattenti;
2. Il mio software è sul Web. Posso eliminare gli errori in un secondo momento;
3. I miei clienti faranno il test del prodotto al posto mio;
4. Chiunque sia qualificato come tester non intende lavorare come tale;
5. Non mi posso permettere un tester!

Possiede le seguenti competenze:

- Creazione di casi di test e di matrici di test;
- Esecuzione dei casi di test nell' ambiente di test e collaudo stabilito;
- Realizzazione degli ambienti di test secondo la progettazione disponibile;
- Registrazione dei difetti e gestione dell'iter di risoluzione degli errori;
- Utilizzo delle librerie negli ambienti di test e collaudo.

Specialista di strumenti (*Test Tool Expert*)

Ha il compito di predisporre e mantenere aggiornati gli ambienti di test con tutta la strumentazione (*debuggers, simulatori, oracoli, registratori, repository, ecc.*) prevista fornendo supporto al loro utilizzo.

Test e ingegneria del software

La IEEE Computer Society ha condotto uno studio sulle competenze richieste all'ingegneria del software. Lo studio ha individuato le caratteristiche principali della formazione professionale relativa all'ingegneria del software[10] ed i contenuti tecnici richiesti.

Sono previste dieci competenze tra cui quella che interessa più direttamente il nostro argomento, "Software testing". Lo studio in oggetto ha prodotto un manuale, disponibile sul Web, con il titolo "Software Engineering Body of Knowledge", anche conosciuto come SWEBOK[11]. La figura riportata di fianco rappresenta lo schema definito dal modello per i processi di sviluppo software e relative competenze.

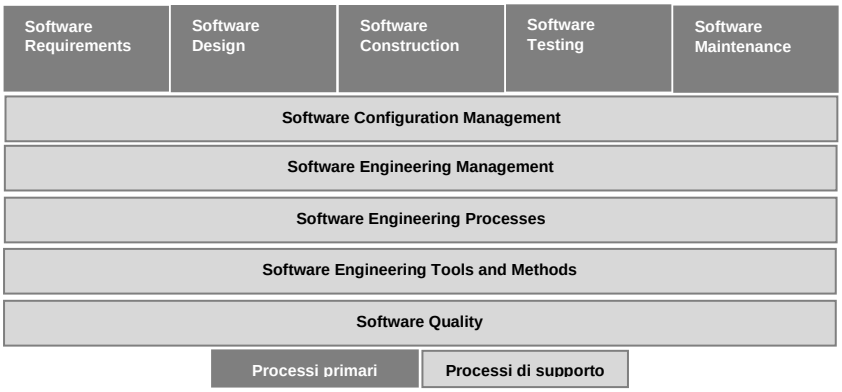

Figura 2. Processi di sviluppo software (ISO 12207:1995).

Codice etico e professionalità

Un codice di etica professionale per gli addetti all'ingegneria del software è stato sviluppato in cooperazione da ACM e IEEE. Di seguito è riportata una versione sintetica. La versione estesa del codice professionale aggiunge a questa maggiori dettagli e consistenza.

> ***Codice etico***
>
> *Gli sviluppatori software devono impegnarsi a rendere il proprio mestiere (l'analisi, la specifica, il disegno, lo sviluppo, il test e la manutenzione del software) una professione rispettata e dagli effetti benefici. Gli sviluppatori devono pertanto aderire alle seguenti regole:*
>
> 1. ***Pubblico**. Gli sviluppatori software devono agire in linea con l'interesse pubblico.*
> 2. ***Cliente e datore di lavoro**. Gli sviluppatori software devono agire in un modo conforme agli interessi del loro cliente e datore di lavoro, restando in accordo con l'interesse pubblico (punto 1).*
> 3. ***Prodotto**. Gli sviluppatori software devono assicurare che i loro prodotti e le modifiche che vi applicano siano a livello di standard professionale più elevato possibile.*
> 4. ***Giudizio**. Gli sviluppatori software devono mantenere integrità ed indipendenza nel loro giudizio professionale.*
> 5. ***Management**. Manager e leader degli sviluppatori devono sottoscrivere e promuovere un approccio etico alla gestione dello sviluppo e della manutenzione del software.*
> 6. ***Professione**. Gli sviluppatori software devono far progredire l'integrità e la reputazione della professione, restando in accordo con l'interesse pubblico (punto 1).*
> 7. ***Colleghi**. Gli sviluppatori software devono essere leali e di supporto nei confronti dei loro colleghi.*
> 8. ***Se stessi**. Gli sviluppatori software devono, per tutta la durata della loro attività lavorativa, continuare la propria formazione sulla pratica della professione, e devono promuovere un approccio etico ad essa.*

Ogni azienda ha un proprio codice etico legato principalmente al settore nel quale opera. Gli ingegneri hanno un proprio albo professionale che ne definisce, oltre ai contenuti tecnici, gli aspetti etici e professionali. Per quanto riguarda l'ingegneria del software, anche se si tratta di una disciplina giovane (ma poi non tanto), non ha un proprio riconoscimento giuridico formale e non esiste ancora un albo degli "Ingegneri del software" (tra l'altro, un ingegnere del software potrebbe non essere un "ingegnere" nel senso stretto del termine). Esistono diversi tentativi nel mondo di disciplinare la materia.

Formazione e addestramento

Un buon tester non s'improvvisa dall'oggi al domani. Anche un ottimo sviluppatore

non può svolgere efficacemente il compito di collaudatore senza che sia stato opportunamente formato.

La creazione di un gruppo di collaudo richiede quindi l'individuazione delle persone più adatte, la pianificazione del percorso formativo da seguire, l'organizzazione del lavoro corrente in modo da liberare le risorse che necessitano di assentarsi dai progetti per seguire i corsi di formazione, un adeguato tirocinio presso il gruppo di collaudo esistente.

La formazione teorica può essere formale in aula oppure, se la persona possiede già elementi di base, può consistere nello studio di un buon manuale (il presente quaderno può essere una buona base di inizio).

Dopo la preparazione teorica la persona segue una fase di tirocinio presso il gruppo di collaudo seguito da un tutor cui è affidata.

La partecipazione attiva alle fasi di collaudo di un progetto in corso e l'esecuzione di attività specifiche di progettazione dei casi di test e della matrice di test, nonché l'esecuzione di casi di prova in autonomia con registrazione dei difetti riscontrati, completano la preparazione del futuro "tester" che potrà essere inserito a pieno titolo nel team di collaudo.

Puntare su persone particolarmente adatte a tale mansione, garantire una formazione tecnica adeguata e fornire il gruppo di test con strumenti (tool) appropriati è il miglior investimento che un'azienda di software possa fare per la qualità dei propri prodotti e servizi.

Il livello di soddisfazione dei propri clienti costituisce una misura tangibile del maggiore successo che ne consegue.

Nel nostro paese non esistono corsi universitari che preparino i giovani a svolgere produttivamente le attività di test.

L'onere è lascito alle singole aziende, con gli investimenti che ciò comporta, e all'impegno personale che ciascuno pone per la propria crescita professionale. Il

[10] G. Ford and E. Gibbs, "*A Mature Profession of Software Engineering*" Software Engineering Institute, Carnegie Mellon University, Pittsburg, Pennsylvania, January 1996.

[11] Il documento in oggetto è diventato uno standard internazionale nel 2005 "ISO/IEC TR 19759:2005(E) IEEE *Guide to Software Engineering Body of Knowledge (SWEBOK)*".

risultato globale è comunque non adeguato alle esigenze del mercato. A livello europeo, ma ancora di più a livello nazionale, il "gap" esistente tra competenze richieste e competenze disponibili nell'ambito IT è ancora molto alto e tende, purtroppo, a crescere nel tempo. L'allarme dato dalla Comunità Europea a tale riguardo pone il problema dello "*Skill shortage*" come un freno alla competitività e alla crescita.