Articolo

Adeguatezza dei test alla criticità del software

Prevenire è meglio che curare.

Ercole Colonese, 2009

Sommario

Introduzione

L'attività di test, confinata per sua natura come ultima fase del ciclo di vita, risente di tutte le perturbazioni che il progetto subisce durante l'esecuzione. Ritardi, taglio dei costi, risorse inadeguate ecc. Questi, e altri motivi ancora, costituiscono un ostacolo al buon completamento del progetto.

Non dedicare alla fase di testing il tempo necessario e le risorse adatte costituisce una lacuna i cui effetti negativi sono evidenziati dopo il rilascio in esercizio. L'applicazione sviluppata mostrerà allora tutti i suoi punti deboli: difettosità oltre l'aspettativa e funzionalità non esattamente corrispondenti ai requisiti.

Il testing, infatti, ha due obiettivi principali: 1) assicurare che i requisiti siano stati sviluppati correttamente e 2) scoprire gli errori presenti nel software prima del suo rilascio agli utenti. Da ciò si capisce bene, dunque, l'importanza di pianificare ed eseguire correttamente le attività di testing.

Caratteristiche del software

L'adeguatezza dei test dipende dalle caratteristiche del software sviluppato. Gli elementi importanti da prendere in considerazione per una corretta pianificazione dei test sono: a) dimensione del software; b) criticità del software per il business; c) complessità del software; d) vincoli e standard da rispettare; e) rischi del progetto.

Ciascun elemento richiede dei test appropriati alle caratteristiche esposte.

Le *dimensioni* del software richiedono un numero adeguato di casi di prova: un software di grandi dimensioni richiederà un numero maggiore che non un software di piccole dimensioni. I casi di prova da eseguire sono riportati in una tabella correlandoli con i requisiti che ciascuno di essi indirizzerà. La "Matrice di test" (Requisiti-Casi di prova) indicherà quali e quanti casi di prova prevedere. L'analisi della tabella permette così di ottimizzare l'impegno riducendo al minimo indispensabile i casi di prova richiesti a copertura totale dei requisti (per esempio, eliminando le duplicazioni). La tabella permette quindi di verificare anche il grado di copertura dei

requisiti da parte dei casi di prova (livello di copertura).

La *criticità* del software per il business può essere indirizzata tramite opportuni casi di prova progettati allo scopo. In ottica di contenimento dei costi (sempre richiesto nei progetti) èutile produrre uno "schema a quattro quadranti" dove riportare le funzionalità del prodotto (F1, F2, ... Fn) in base alla frequenza con cui ciascuna di esse sarà utilizzata e alla criticità che essa rappresenta per il business[1]. Come mostrato nell a Figura 1, nel quadrante in alto a destra sono riportate le funzionalità più critiche per il business e utilizzate con maggiore frequenza dagli utenti. Viceversa, nel quadrante in basso a sinistra, cadono le funzionalità meno critiche e meno utilizzate. Le prime richiederanno certamente un numero di casi di prova maggiore delle seconde.

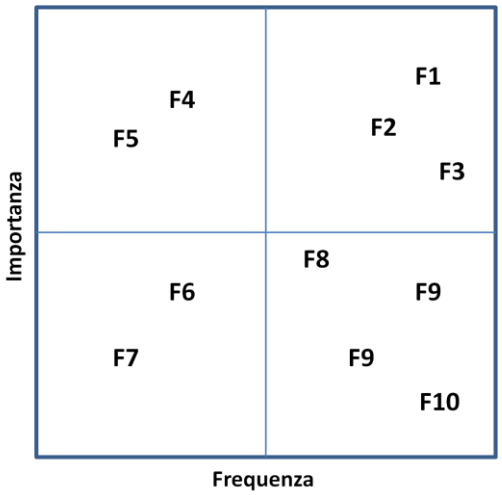

Figura 1 – Relazione tra frequenza di utilizzo e importanza delle funzioni.

La *complessità* del software è un elemento critico di per se e richiede un numero di casi di prova adeguato. Questi saranno progettati ad hoc per indirizzare i diversi aspetti della complessità: prestazioni, usabilità, gestione della memoria, algoritmi applicativi particolari, struttura della base dati, ecc.

I *vincoli* e *standard* da rispettare sono elementi da indirizzare con particolari casi di prova progettati anch'essi ad hoc.

I *rischi* del progetto sono infine indirizzati da appositi test (es.: tecnologie particolari utilizzate per lo sviluppo del software, indeterminatezza di alcune specifiche tecniche, ecc.).

In conclusione, il numero e i tipi di casi di prova dipenderà dai fattori elencati sopra e non, come spesso avviene, da quanto tempo abbiamo a disposizione o dalle risorse disponbili.

Pianificazione dei test

La pianificazione dei test è fondamentale per valutare l'impegno di tempo e risorse necessarie. Una buona pianificazione deve essere “realistica” ed “efficace”.

"Realistica" significa prevedere un numero "congruo" di casi di prova che possano essere realmente progettati ed eseguiti; la Matrice di test (**Figura 2**) e lo Schema a quattro quadranti sono due ottimi strumenti per selezionare quali e quanti casi di prova prevedere.

	CT-1	CT-2	CT-3	CT-4	CT-5	CT-6	CT-7	...	CT-n
REQ-1	x	x	x	x				x	
REQ-2	x		x				x	x	x
REQ-3		x		x					x
REQ-4		x		x	x	x			x
REQ-5			x	x	x	x			
...						x	x		
REQ-n							x	x	

Figura 2 – Matrice di test (corrispondenza caso di prova – requisiti).

"Efficace" significa progettare i casi di prova secondo la caratteristica del software da verificare: un requisito prestazionale richiederà un tipo di test (test di performance) diverso da una caratteristica funzionale (test funzionale). Occorre dunque esaminare tutte le caratteristiche del software e pianificare i diversi tipi di test da progettare ed eseguire (test funzionale, test prestazionale, test di usabilità, test di carico, test procedurale, ecc.).

La definizione dei tipi di test previsti è detta "Strategia di test". E da essa che occorre partire per una corretta pianificazione. La valutazione dei tempi e delle risorse necessarie è documentata nel "Piano di

[1] La matrice a quattro quadranti è stata già presentata nel primo articolo sulle best practice (*Analisi accurata dei requisiti*, 2009).

test" e richiede l'approvazione del management (sponsor, cliente, ecc.) per assicurare le risorse necessarie. E' bene ricordare che la pianificazione include la predisposizione degli ambienti di test (non sempre banale e a costo zero!).

Progettazione dei test

La progettazione dei casi di prova e degli ambienti di test determina l'efficacia dei test eseguiti. Essa deve essere affidata a personale competente sia dal punto di vista funzionale (competenza funzionale come quella posseduta dagli analisti) sia dal punto di vista dei metodi e delle tecniche del test (competenza degli esperti di test).

Se opportuno o necessario, si progettano dei test automatici tramite l'uso dei tool selezionati.

La documentazione dei casi di prova permette di accedere ai test anche in momenti successivi (es.: in fase di test di regressione durante, la manutenzione, per verificare che alcune funzionalità contigue a quelle modificate non siano state influenzate negativamente e continuino a funzionare correttamente).

La progettazione tiene conto dei "tipi" di test e della loro "profondità". I tipi di test, si è detto, indirizzano diversi aspetti del software: funzionalità, prestazioni, usabilità, carico, procedure, regressione, parallelo, ecc. La profondità dei test – o livello di test - si riferisce appunto al "livello" di dettaglio in cui si trova il software sviluppato: codifica, integrazione, sistema, collaudo. In corrispondenza a tali livelli, i relativi test sono detti unitari, d'integrazione, di sistema e di accettazione. Ogni tipo di test, così come ogni livello di test, ha caratteristiche proprie ed è progettato di conseguenza.

Tabella 1 – Corrispondenza tra livello di software e livello di test.

Software	Test
Requisiti	Test d'accettazione
Specifiche	Test di sistema
Disegno	Test d'integrazione
Codice	Test unitario

Il modello a "V" (**Figura 3**) costituisce un'ottima rappresentazione di tale concetto: la corrispondenza esistente tra il tipo di test da eseguire e il livello di completamento del software lungo il suo ciclo di sviluppo.

La parte sinistra della figura mostra come si passa, con attività di raffinamento successivo, dai requisiti alle specifiche, al disegno e infine al codice dei singoli moduli. La parte destra, invece, mostra come si procede, a ritroso, effettuando test man mano che il software è codificato, integrato, completate nel sistema disegnato e sottoposto all'accettazione degli utenti.

La corrispondenza tra ciascuna fase di sviluppo (parte sinistra della figura) e l'attività di test (parte destra) mette in luce la diversa tipologia di test da eseguire in base al livello di completamento del software.

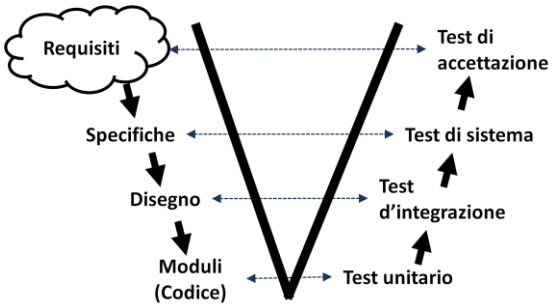

Figura 3 – Modello a "V" del testing.

Nessuna fase di test può essere dunque tralasciata perché considerata non importante: ciascuna di esse ha la sua importanza e deve essere pianificata, progettata ed eseguita con cura.

Esecuzione dei test

L'esecuzione dei test si porta a termine come definito nel Piano di test: secondo i tempi previsti, dalle risorse coinvolte, negli ambienti di test predisposti ed eseguendo i casi di prova progettati.

Gli errori rilevati durante l'esecuzione dei test sono registrati e notificati al gruppo di sviluppo perché li corregga. Il codice modificato è nuovamente sottoposto a test per verificare che l'errore sia stato rimosso. La registrazione degli errori e la loro rappre-

sentazione cumulativa giornaliera permettono di creare la "curva di rimozione degli errori" (detta anche "curva di saturazione degli errori") permettendo di verificare il momento in cui i test possono considerarsi esaustivi (**Figura 4**).

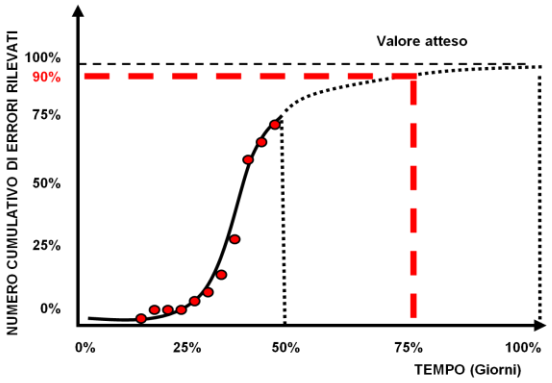

Figura 4 – Curva di rimozione degli errori (cumulativa).

L'andamento della curva dimostra che i test non possono essere interrotti quando sia stata raggiunta la data pianificata per la fine dei test. L'interruzione potrà avvenire solo quando la curva si sarà stabilizzata e il numero cumulativo di errori rilevati avrà raggiunto un valore percentuale vicino a quello ritenuto soddisfacente per gli obiettivi dei test (es.: quando il numero di errori rilevati sia prossimo al 90%).

Rapporto sullo stato dei test

Il risultato dei test è documentato in un apposito Rapporto di test (*Test Report*) dove si riportano le informazioni relative ai casi di prova (es.: numero di casi di prova completati, ancora in esecuzione, bloccati da errori, numero totale di casi di prova previsti dalla fase di test). Si riportano anche le informazioni concernenti gli errori rilevati in ciascuna fase di test (es.: numero totale di errori rilevati, corretti, testati). Le informazioni sugli errori si riferiscono alle diverse gravità con cui sono classificati (es.: errori bloccanti, gravi, lievi).

Chiusura dei test

Una fase di test si considera chiusa quando: 1) siano stati completati tutti i casi di prova previsti e 2) siano stati corretti e verificati tutti gli errori rilevati. Al termine della fase di test si emette un "Rapporto di fine test" in cui si dichiara quanto effettuato e i risultati conseguiti. Solo se sono stati raggiunti gli obiettivi pianificati per ciascuna fase di test queste potranno essere considerate chiuse.

Calcolo degli errori residui

Il calcolo degli errori residui nel software rilasciato può essere compiuto, se richiesto, registrando il numero di errori rilevati in ciascuna fase di test (unitario, d'integrazione, di sistema e d'accettazione).

L'andamento della curva di rilevazione degli errori riportata nella **Figura 5** permette di estrapolare il numero di errori residui.

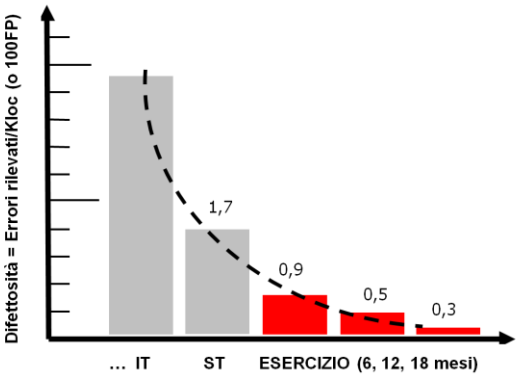

Figura 5 – Curva di estrapolazione degli errori residui.

La parte della figura riportata in colore rosso mostra i valori degli errori rilevabili in esercizio nei tre periodi corrispondenti a sei, 12 e 18 mesi di utilizzo del prodotto software.

La produzione di una tale curva richiede, come si vede, la registrazione del numero di errori rilevati nelle diverse fasi di test: cosa non difficile da realizzare ma, quasi sempre, disattesa nella maggior parte delle organizzazioni software..

Rischio associato a test carenti

La mancata esecuzione di quanto descritto in precedenza rappresenta un rischio grave per il progetto: alta probabilità di accadimento (per non dire certezza!) e alto impatto! Le conseguenze sono facili da prevedere e quantificare: difettosità alta del software rilasciato e funzionalità non sempre accettate dagli utenti. La ripercussione sui costi e l'immagine sono altrettanto evidenti: costo di manutenzione correttiva in garanzia alta (e conseguente riduzione del margine di profitto) e soddisfazione bassa degli utenti. Nei casi peggiori si può non superare il collaudo di accettazione e ritardare l'emissione delle fatture relative!

Eseguire test accurati è dunque una buona pratica del software (*Best Practice*).

Bibliografia

Testi generali di Software Engineering

[1] Sommerville, Ian. *Ingegneria del Software*. Addison Wesley, 7ª Edizione (2005).

[2] Pressman, Roger S. *Principi di ingegneria del software.* McGram-Hill (2005).

[3] Ghezzi, Jazayeri, Mandrioli. *Ingegneria del software*. Prentice Hall 2ª Edizione (2004).

[4] Binato, Fuggetta, Sfardini. *Ingegneria del software.* Addison Wesley (2006).

[5] Arlow J., Neustadt, I. *UML 2 e Unfied Process.* McGraw-Hill (2006).

[6] Zuser, W., Biffl, S., Kohle, M. *Ingegnria del software con UML e Unified Process.* McGraw-Hill (2004).

[7] Kruchten, Philippe. *Rational Unified Process - Introduzione.* Addison-Wesley (2000).

Testi specifici di Software Testing

[8] A. Avellone, M. Cislaghi, E. Colonese, *Qualità e collaudo del software*, 2010, Editrice Uni-Service.

[9] B. Beizer, Software System Testing and Quality Assurance, 1990, International Thomson Computer Press.

[10] G.A. Cignoni, P. De Risi, Il test e la qualità del software, 1998, Il Sole 24 Ore.

[11] Tom Gilb and Dorothy Graham, Software Inspection, 1993, Addison Wesley.

[12] McCabe T.J., Watson A.R., Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, 1996, NIST Special Publication 500-235.

[13] Daniel J. Mosley, Bruce A. Posey, Collaudo del Software (titolo originale: Just Enough Software Test Automation), 2003, McGraw Hill.

[14] Myers G. J., The art of Software Testing, 1979, John Wiley & Sons, New York.

[15] Myers G. J., The art of Software Testing – Second Edition, 2004, John Wiley & Sons, New York.

[16] Cem Kaner, Lames Bach, Bret Pettichord, Lessons Learned in Software Testing, A Context-Driven Approach, 2003, Wiley.

Autore

Ercole Colonese è consulente di direzionale e servizi IT. Opera, in particolare, nell'ambito delle metodologie di sviluppo software, dei servizi IT e della formazione.

La sua esperienza è maturata in moltissimi anni di lavoro presso i laboratori internazionali IBM di sviluppo software dove ha ricoperto ruoli tecnici, manageriali e dirigenziali.

E' socio APCO e membro del Consiglio Direttivo di AICQ-CI per il Comitato per la Qualità del Software e dei Servizi IT.

Ha realizzato Sistemi aziendali integrati di gestione per la Qualità, certificati ISO 9001, e Sistemi di gestione dei servizi IT, certificati ISO 20001:2005. Ha implementato modelli di eccellenza (EFQM, Malcom Baldrige, Six-Sigma). Ha condotto progetti di reingegne-rizzazione dei processi di sviluppo software in ottica di miglioramento delle performance e della qualità. Ha applicato il modello di maturità *Capability Maturity Model* (SEI-CMMI).

Come docente, tiene corsi di Software Engineering, Project Management e ITIL. In passato ha tenuto corsi sull'Ingegneria del software presso il Learning Center di IBM, l'Università Tor Vergata di Roma, il Politecnico di Vibo Valentia e l'Università del Sacro cuore di Roma.

Ha collaborato con CIRPS su progetti di ricerca tecnica.

Ha collaborato con CNIPA (ora DigitPA) sul tema del riuso del software applicativo.

Collabora alla pubblicazione dei Quaderni AICQ e articoli monotematici sulla qualità del software e dei servizi IT pubblicati sulla rivista Qualità On-Line di AICQ.

Articoli sull'Ingegneria del software sono pubblicati sul proprio sito all'indirizzo: (http://www.colonese.it/pubblicazioni/htm).

e-mail: ercole@colonese.it www.colonese.it