

A. Avellone, M. Cislighi, E. Colonese

COLLAUDO E QUALITÀ DEL SOFTWARE


Qualità del software e dei servizi IT

Editrice | UNI Service

A. Avellone, M. Cislighi, E. Colonese, *Collaudo e qualità del software*
Copyright© 2010 UNI Service – Trento
Via Verdi, 9/A – 38122 Trento
www.uni-service.it – editrice@uni-service.it

Prima edizione: dicembre 2010 – *Printed in Italy*
ISBN 978-88-6178-619-6

In copertina: *Global Communication* © TheSupe87 – Folia.com

Progetto grafico di copertina: 

Sponsor editoriale Engineering Ingegneria Informatica



I lettori che desiderano informazioni sui libri, quaderni e riviste pubblicate dalla nostra associazione (Aicq-ci) possono consultare il nostro sito Internet www.aicqci.it, richiedere informazioni tramite e-mail: info@aicqci.it, chiamare la segreteria al numero 06.4464132.



www.uni-service.it

Novità - Catalogo - Acquisti on-line

SOMMARIO

1	PRESENTAZIONE	1
2	INTRODUZIONE	9
2.1	Il collaudo del software	9
2.2	Il software	10
2.2.1	<i>Che cos'è il software</i>	10
2.2.2	<i>Lo sviluppo software</i>	10
2.2.3	<i>Quantità di errori immessi e rimossi nel software</i>	11
2.2.4	<i>Propagazione degli errori nel software</i>	13
2.2.5	<i>Tipologia di errori immessi e rimossi nel software</i>	14
2.2.6	<i>Costo di rimozione degli errori nel software</i>	15
2.3	Che cos'è il collaudo del software	16
2.3.1	<i>Generalità</i>	16
2.3.2	<i>Psicologia del tester</i>	16
2.3.3	<i>Problematiche relative al test del software</i>	17
2.3.4	<i>Relazioni tra test e altre attività</i>	19
2.3.5	<i>Principi generali del test</i>	20
2.4	Testing e qualità del software	23
2.4.1	<i>Che cos'è la qualità del software</i>	23
2.4.2	<i>Qualità di prodotto e di processo</i>	23
2.4.3	<i>Assicurazione qualità</i>	24
2.4.4	<i>Standard di verifica e validazione</i>	26
2.4.5	<i>L'approccio alla qualità del software</i>	26
2.4.6	<i>La qualità del prodotto e il ciclo di vita</i>	28
3	GUIDA ALLA LETTURA	31
3.1	Obiettivi e ambito	31
3.2	A chi è rivolto	31
3.3	Struttura	32

4	DEFINIZIONI, STANDARD E MODELLI	35
4.1	Acronimi	35
4.2	Definizione dei termini	37
4.3	Standard relativi al test del software	51
4.3.1	<i>ISO 9001 e “Verifica e Validazione”</i>	56
4.3.2	<i>ISO/IEC 12207 e “Verifica e Validazione”</i>	57
4.3.3	<i>ISO/IEC TR 19759:2005 IEEE, Guide to the Software Engineering Body Of Knowledge (SWEBOK) [IronMan version]</i>	59
4.3.4	<i>ISO/IEC 9126</i>	60
4.4	Modelli e test del software	62
4.4.1	<i>Capability Maturity Model (CMMI)</i>	62
4.4.2	<i>Modello di maturità per il testing (TMMI)</i>	66
4.4.3	<i>Aree di processo</i>	71
5	IL TEST NEL CICLO DI VITA DEL SOFTWARE	73
5.1	Il test nello sviluppo del software	73
5.2	Strategia e pianificazione dei test	79
5.3	Progettazione e preparazione dei test	80
5.4	Esecuzione dei test	82
5.4.1	<i>Test unitario</i>	83
5.4.2	<i>Test d'integrazione</i>	84
5.4.3	<i>Test di sistema</i>	86
5.4.4	<i>Test di accettazione</i>	89
5.5	Controllo, monitoraggio e reporting	92
6	ORGANIZZAZIONE DEL TEST	95
6.1	Il gruppo di test	95
6.1.1	<i>Composizione del gruppo di test</i>	95
6.1.2	<i>Sinergia con il gruppo di sviluppo</i>	98
6.2	Competenze e specializzazione	99
6.2.1	<i>Ruoli e responsabilità</i>	99
6.2.2	<i>Competenze dell'ingegneria del software e del software testing</i>	101
6.2.3	<i>Competenze e processi di sviluppo</i>	103
6.2.4	<i>Codice etico e professionalità</i>	103

6.3	Formazione, addestramento e certificazione	105
6.3.1	<i>Formazione e addestramento</i>	105
6.3.2	<i>Certificazione professionale</i>	106
7	TIPOLOGIE DI TEST	109
7.1	Modello a “V”	109
7.2	Livelli di testing	111
7.2.1	<i>Revisione tecnica/Ispezione (Technical Review/Inspection)</i>	111
7.2.2	<i>Test unitario (Unit Test)</i>	112
7.2.3	<i>Test d'integrazione (Integration Test)</i>	113
7.2.4	<i>Test di sistema (System Test)</i>	114
7.2.5	<i>Test di accettazione (Acceptance Test)</i>	115
7.3	Tipi di test	117
7.4	Test funzionali	118
7.4.1	<i>Test delle funzionalità</i>	118
7.4.2	<i>Test di gestione delle condizioni di errore</i>	119
7.4.3	<i>Test di operatività</i>	120
7.4.4	<i>Test d'installazione</i>	121
7.4.5	<i>Test di regressione</i>	122
7.4.6	<i>Test di parallelo</i>	123
7.4.7	<i>Test della documentazione</i>	124
7.4.8	<i>Test delle procedure</i>	125
7.4.9	<i>Test di usabilità</i>	126
7.5	Test strutturali	127
7.5.1	<i>Test di ripristino</i>	128
7.5.2	<i>Test di sicurezza</i>	129
7.5.3	<i>Test delle prestazioni</i>	130
7.5.4	<i>Test di carico</i>	131
8	TEST DI APPLICAZIONI E AMBIENTI SPECIFICI	133
8.1	Test di un'applicazione Web	133
8.1.1	<i>Punti di vista del “Web testing”</i>	133
8.1.2	<i>Performance, load e stress test</i>	133
8.1.3	<i>Strumenti software a supporto</i>	135
8.1.4	<i>Una lista di controllo (Checklist)</i>	136
8.2	Test di sicurezza	142

8.2.1	<i>Scenario</i>	142
8.2.2	<i>Obiettivi</i>	143
8.2.3	<i>Test di sicurezza dell'applicazione</i>	143
8.2.4	<i>Framework per Application Security Testing</i>	144
8.2.5	<i>Caratteristiche di un buon tool per il test di sicurezza</i>	154
	<i>Costi e ritorni della sicurezza</i>	156
9	STIMA DEI TEMPI E DEI COSTI DEL COLLAUDO	159
9.1	Una regola d'oro	159
9.2	Stimare i tempi e i costi del collaudo	162
9.3	Relazione tra tempi-costi del collaudo e livello di rischio del progetto	164
9.4	Scelte e gestione dei rischi	167
9.5	Controllo dei tempi e dei costi del collaudo	169
9.6	Economicità dei test	171
10	METODI E TECNICHE DI TEST	173
10.1	Revisione tecnica	173
	10.1.1 <i>Ispezione</i>	176
	10.1.2 <i>Walkthrough</i>	176
	10.1.3 <i>Revisione "informale"</i>	176
	10.1.4 <i>Revisione "formale" o "strutturata"</i>	178
10.2	Integrazione del software	182
	10.2.1 <i>Product Break-down Structure (PBS)</i>	182
	10.2.2 <i>Integrazione "Top-down"</i>	184
	10.2.3 <i>Integrazione "Bottom-up"</i>	186
	10.2.4 <i>Combinazione delle due tecniche</i>	187
10.3	Testing a scatola aperta (<i>White-box Testing</i>)	187
	10.3.1 <i>Ispezione del codice (Code Inspection)</i>	189
	10.3.2 <i>Valutazione della complessità del codice</i>	195
	10.3.3 <i>Correzione degli errori (Debugging)</i>	196
	10.3.4 <i>Copertura del codice (Code Coverage)</i>	199
10.4	Testing a scatola chiusa (<i>Black-box Testing</i>)	206
	10.4.1 <i>Partizionamento dei dati di input (Equivalence Partitioning)</i>	207
	10.4.2 <i>Valori al contorno (Boundary Value Analysis)</i>	208

10.4.3	<i>Previsione degli errori (Error Guessing)</i>	209
10.4.4	<i>Ricerca di errori casuali (Fuzz Test o Fuzzing)</i>	210
10.5	Testing a scatola grigia (<i>Grey-box Testing</i>)	211
10.5.1	<i>Iniezione di errori (Fault Injection)</i>	211
10.5.2	<i>Mutazione del codice (Mutation Testing)</i>	213
10.6	Validazione dell'usabilità	213
10.6.1	<i>Revisione euristica dell'usabilità</i>	214
10.6.2	<i>Test di usabilità</i>	216
10.6.3	<i>Valutazione dell'usabilità del software</i>	219
10.6.4	<i>Metriche di usabilità</i>	222
10.6.5	<i>Checklist di valutazione dei siti Web</i>	224
10.7	La previsione dei risultati (oracolo)	233
10.8	Valutazione e controllo dei test	236
10.8.1	<i>Profilo di qualità del prodotto</i>	236
10.8.2	<i>Matrice di test (Test Matrix)</i>	237
10.8.3	<i>Registrazione e risoluzione degli errori</i>	239
10.8.4	<i>Curva di rimozione degli errori</i>	239
11	MISURE DEL TESTING	243
11.1	Generalità	243
11.2	Origine dei dati	244
11.3	Misure relative al testing	244
11.4	Misure relative alle attività di test	245
11.4.1	<i>Difettosità dei prodotti di fase</i>	245
11.4.2	<i>Profilo di qualità del software</i>	246
11.4.3	<i>Difettosità residua del software</i>	248
11.4.4	<i>Copertura dei test</i>	249
11.4.5	<i>Efficacia ed efficienza dei test</i>	249
11.4.6	<i>Efficacia delle revisioni tecniche</i>	250
11.4.7	<i>Produttività dei test e delle revisioni tecniche</i>	250
11.4.8	<i>Ritorno dell'investimento delle ispezioni</i>	251
11.5	Misure relative alla qualità del software testato	252
11.5.1	<i>Modello di qualità del software</i>	252
11.5.2	<i>Complessità ciclomatica</i>	252
11.5.3	<i>Livello di accoppiamento (Coupling)</i>	253
11.5.4	<i>Livello di coesione (Coesion)</i>	255

11.5.5	<i>Usabilità</i>	257
11.5.6	<i>Prestazioni (Efficienza)</i>	258
11.5.7	<i>Robustezza</i>	258
11.5.8	<i>Recuperabilità</i>	259
11.5.9	<i>Manutenibilità</i>	259
12	STRUMENTI PER IL COLLAUDO	261
12.1	Automazione dei test	261
12.2	Strumenti di test	262
12.2.1	<i>Strumenti per la gestione degli errori</i>	263
12.2.2	<i>Strumento per la gestione delle modifiche al software</i>	265
12.2.3	<i>Strumento per la gestione e la progettazione dei test</i>	266
12.2.4	<i>Strumenti per l'analisi statica del codice</i>	267
12.2.5	<i>Strumento per l'esecuzione dei test unitari</i>	268
12.2.6	<i>Strumento per l'esecuzione automatica dei test</i>	269
12.2.7	<i>Strumento per la simulazione delle condizioni di carico</i>	270
12.2.8	<i>Strumenti per l'analisi delle prestazioni del software</i>	271
12.2.9	<i>Strumenti per l'analisi statica del codice</i>	272
12.3	Altri strumenti per il testing	272
13	TEST PARTICOLARI	275
13.1	Test "Agile"	275
13.1.1	<i>L'approccio "agile" al software</i>	276
13.1.2	<i>Il test nel Ciclo di vita "agile"</i>	279
13.1.3	<i>Conclusioni</i>	284
13.2	Continuous Integration and Test	285
13.2.1	<i>Lo scenario</i>	285
13.2.2	<i>L'integrazione continua ed il test continuo</i>	287
13.2.3	<i>Conclusioni e valutazioni</i>	289
13.3	Esempi di modulistica per i test	293
13.3.1	<i>Modello di Piano di test</i>	293
13.3.2	<i>Modelli di Specifiche di test</i>	297
13.3.3	<i>Modelli di Reportistica di test</i>	301
14	GLOSSARIO	305

15	BIBLIOGRAFIA	315
15.1	Pubblicazioni	315
15.1.1	<i>Testing</i>	315
15.1.2	<i>Continuous Integration and Testing</i>	316
15.1.3	<i>Security Testing</i>	316
15.1.4	<i>Software Engineering</i>	317
15.1.5	<i>Modelli di maturità</i>	318
15.1.6	<i>Standard internazionali</i>	319
15.1.7	<i>Pubblicazioni AICQ-CI</i>	321
15.1.8	<i>Pubblicazioni DigitPA (ex CNIPA)</i>	321
15.1.9	<i>Metriche del software</i>	321
15.1.10	<i>Sviluppo professionale</i>	322
15.1.11	<i>Usabilità del software</i>	322
15.1.12	<i>Qualità</i>	323
15.2	Siti Web	323
	NOTE SUGLI AUTORI	327

Figure

Figura 1. Curva di rimozione degli errori durante le fasi del ciclo di vita	12
Figura 2. Schema di propagazione degli errori (fonte: IBM System Scientific Institute, 1981)	13
Figura 3. Costo di rimozione degli errori nelle fasi (fonte: Pressman)	15
Figura 4. Qualità del prodotto basata sulla qualità del processo	24
Figura 5. Qualità del prodotto software nel ciclo di sviluppo (fonte: ISO/IEC 9126)	27
Figura 6. La qualità nel ciclo di vita del software (fonte: ISO/IEC 9126-1:2001)	29
Figura 7. Relazione causa-effetto nel software	39
Figura 8. Verifica e validazione nel ciclo di vita	41
Figura 9. Schema dei livelli di maturità del modello CMMI	63
Figura 10. TMMi e Aree di processo (fonte: TMMi Foundation)	67
Figura 11. Modello del processo di test del software	73
Figura 12. Collegamento tra piani di test (generale e di dettaglio)	74
Figura 13. Elementi del processo di test del software	76
Figura 14. Relazione tra livelli/tipi di test e documentazione	78
Figura 15. Competenze specifiche dell'Area di conoscenza Software Testing (fonte: SWEBOK, 2004 – IronMan Version)	102
Figura 16. Processi di sviluppo software (fonte: SWEBOK, 2004 – IronMan Version)	103
Figura 17. Schema dei livelli di testing (“V” Model)	109
Figura 18. Schema dei livelli di testing (“V” Model)	110
Figura 19. Soddisfazione dell'utente in base ai tempi di risposta	141
Figura 20. Processi per la sicurezza in un ciclo di sviluppo	146
Figura 21. Mappa delle funzionalità in base alla criticità e frequenza di utilizzo	167
Figura 22. Andamento delle attività di rilevazione e correzione degli errori	171
Figura 23. Sinergia tra revisione tecnica e testing	174

Figura 24. Revisioni tecniche durante il ciclo di vita	175
Figura 25. Scomposizione del software in moduli e grafo della relazione d'uso	183
Figura 26. Moduli Driver e Stub nell'integrazione del software	184
Figura 27. Schema di integrazione Top-down con utilizzo di "stub"	185
Figura 28. Schema di integrazione Bottom-up con utilizzo di un "driver"	186
Figura 29. Test a scatola aperta (White-box)	188
Figura 30. Complessità ciclomatica del software (fonte: Internet)	196
Figura 31. Ciclo di "debugging"	198
Figura 32. Operazioni primitive della programmazione strutturata	202
Figura 33. Test a scatola chiusa (Black-box)	206
Figura 34. Processo di revisione euristica dell'usabilità del software	215
Figura 35. Processo di test di usabilità del software	218
Figura 36. Schema di utilizzo di un oracolo	234
Figura 37. Curva "ideale" di rimozione degli errori in una fase di test	239
Figura 38. Costruzione "manuale" della curva di rimozione degli errori	241
Figura 39. Curva di rimozione degli errori (pianificata e attuale)	247
Figura 40. Curva per l'estrapolazione degli errori residui	248
Figura 41. Collocazione degli strumenti nello schema a "V" del testing	263
Figura 42. Strumento per la gestione dei difetti	264
Figura 43. Strumento per la gestione delle modifiche del software	266
Figura 44. Strumento per la gestione e la progettazione dei test	267
Figura 45. Registrazione ed esecuzione automatica dei test	270
Figura 46. Sviluppo di una User Story secondo il Test Driven Development (TDD)	282
Figura 47. Il test nel Ciclo di vita agile	283
Figura 48. Flusso del processo "Continuous Integration and Test"	286
Figura 49. Architettura (adattata da Duvall, 2007)	287
Figura 50. Continuous integration e testing nel Modello a V	290
Figura 51. I test in manutenzione (Sheikh, 2008)	291

Tabelle

Tabella 1. Tipologia di errori comunemente immessi nel software (fonte: R. S. Pressman)	14
Tabella 2. Standard di prodotto e di processo	25
Tabella 3. Modello di qualità del software (ISO/IEC 9126-1:2005)	61
Tabella 4. Codice etico per la professione di Ingegnere del software	104
Tabella 5. Sommario dei tipi di test specifici	134
Tabella 6. Costo di rimozione degli errori (fonte dei dati: R. S. Pressman, 2005)	145
Tabella 7. Confronto dei costi di rimozione “anticipata” e “posticipata” degli errori	157
Tabella 8. Regola d’oro della ripartizione degli impegni in un progetto software	159
Tabella 9. Profilo di durata e impegno secondo il modello RUP	159
Tabella 10. Livello di rischio per categoria	165
Tabella 11. Valutazione del rischio del progetto	166
Tabella 12. Stima dell’impegno per il test in base al livello di rischio	166
Tabella 13. Rappresentazione della criticità, frequenza e priorità	168
Tabella 14. Controllo del livello di completamento dei casi di test	170
Tabella 15. Controllo del livello di risoluzione degli errori	170
Tabella 16. Metriche relative al testing	172
Tabella 17. Ruoli coinvolti nelle revisioni tecniche “informali”	177
Tabella 18. Ruoli coinvolti nelle revisioni tecniche “formali”	178
Tabella 19. Produttività delle revisioni tecniche (fonte: Nocentini S.)	181
Tabella 20. Controlli da effettuare nelle ispezioni del codice (fonte: Ian Sommerville)	194
Tabella 21. Errori rilevati in funzione del numero di esperti coinvolti	214
Tabella 22. Ciclo di sviluppo di siti/applicazioni Web	217
Tabella 23. Valutazione dei casi d’uso da utilizzare per i test di usabilità	218
Tabella 24. Caratteristiche di usabilità del software (ISO/IEC 9126)	223

Tabella 25. Metriche di usabilità del software dedotte da standard “de facto”	224
Tabella 26. Modello per la valutazione dell’usabilità di un sito Web	225
Tabella 27. Modello per il Sommario delle valutazioni dell’usabilità di un sito Web	227
Tabella 28. Secondo modello per la valutazione dell’usabilità di un sito Web	229
Tabella 29. Modello di domande per la valutazione dell’usabilità di un sito Web	232
Tabella 30. Profilo di qualità del prodotto	237
Tabella 31. Matrice di tracciabilità dei requisiti da parte dei casi di test	238
Tabella 32. Differenze tra metodologie agili e tradizionali	277
Tabella 33. Il Manifesto Agile	278
Tabella 34. Prodotti open source integrabili in un ambiente di Continuous Integration	289

1 Presentazione

Perché un altro libro sul test e collaudo del software

L'attività di test nei progetti di sviluppo software riveste un'importanza notevole dato il ruolo cruciale che essa svolge nell'organizzazione del progetto: "Scoprire il maggior numero possibile di errori (e consentirne la correzione) prima che il prodotto software venga immesso sul mercato o sia rilasciato nell'ambiente di esercizio del cliente".

L'esperienza ci porta a constatare che, da sempre, l'attività di test viene sottovalutata e la sua pianificazione risulta inadeguata. Essendo l'ultima fase del ciclo di sviluppo, il test risente negativamente dei ritardi accumulati dal progetto e, necessariamente, vede ridursi i tempi di svolgimento e le risorse assegnate.

Il collaudo svolto dall'utente in ottica di accettazione del prodotto finale si svolge spesso in uno scenario diverso da quello previsto; da un ruolo istituzionale di "validazione dell'aderenza della soluzione sviluppata alle esigenze della Committenza" si passa ad uno di vero e proprio "test e collaudo dell'applicazione". Non è raro vedere collaudi di accettazione durare mesi e, qualche volta, non raggiungere l'obiettivo. Con ritardi nei tempi di rilascio in esercizio, dispendio di risorse e insoddisfazione da parte del cliente e degli utenti finali.

Se test esaustivi occorre fare, perché non pianificarli correttamente e svolgerli per tempo piuttosto che sotto lo stress provocato dall'incalzare del cliente a fronte di collaudi di accettazione falliti?

Le *best practice* dell'ingegneria del software attribuiscono alle attività di test la giusta importanza e suggeriscono di assegnare tempi e risorse a tale attività non inferiori a una percentuale pari a 30-40% dell'impegno totale del progetto." Percentuali esagerate!" è il commento più comune. La stima parte dalla constatazione che l'attività di testing "costa" quanto la programmazione. Generalmente l'attività di test stimata dai progetti si riferisce alla sola progettazione ed esecuzione dei casi di prova; si trascurava invece la stima concernente la pianificazione dei test, la definizione della strategia di test, la preparazione degli ambienti, l'esecuzione delle ispezioni, il controllo e la valutazione dei risultati, la modifica del lavoro già fatto, ecc. Includendo le attività appena menzionate le percentuali suggerite diventano più realistiche.

Un altro aspetto importante si riferisce all'esecuzione delle revisioni tecniche (ispezioni, walkthrough) nelle fasi alte del ciclo di vita. Il fenomeno della propagazione degli errori e l'aumentare dell'impegno richiesto per la loro correzione man mano che si procede nelle fasi di sviluppo, suggerisce di rimuovere gli errori il più presto possibile, già dalle prime fasi del ciclo di vita (Requisiti, Analisi e Disegno).

Altro fattore critico da prendere in considerazione è l'aspetto organizzativo del team: ruoli e responsabilità, competenze e attitudini, processo, metodi e tecniche, strumenti, standard e modelli. Sono componenti unici del test, diversi da quelli relativi alla programmazione. Progettare ed eseguire un test è diverso da progettare un software e sviluppare il codice. Competenza, mentalità e attitudine sono elementi specifici del testing, diversi da quelli richiesti per programmare.

In particolare, l'adozione di nuove tecnologie e strumenti automatici per la progettazione e la codifica del software fornisce un'opportunità per migliorare la qualità del codice sviluppato. Lo sviluppatore è supportato da strumenti, piattaforme e tecnologie (framework) che guidano il design, creano i flussi, controllano la sintassi, verificano la compatibilità, evidenziano gli errori nel codice, ecc. Le piattaforme di sviluppo offrono potenzialità per il miglioramento della qualità del software ma, ahimè, esso (il software) arriva ancora al collaudo utente con troppi errori. Il test è ancora insufficiente. Il prototipo diventa spesso il prodotto finale, senza ingegnerizzazione e senza adeguati test.

Il libro affronta tutti questi temi in maniera completa, semplice ma rigorosa. Con un occhio sempre rivolto a chi opera quotidianamente in ambienti non sempre favorevoli al cambiamento. Forse è per questo che considero questo lavoro "utile" a divulgare i concetti della qualità del software e dei servizi IT e a fornire, anche se in maniera modesta, un contributo al miglioramento.

Desidero perciò ringraziare tutti i componenti del team per l'ottimo lavoro svolto; in particolare desidero ringraziare il coordinatore del gruppo per la dedizione e la professionalità dimostrate.

Mario Cislaghi

Presidente Comitato AICQ per la Qualità del Software

Vice Presidente AICQ-CI

L'approccio di Engineering al test del software

Gabriele Ruffatti

Direttore Architetture e Consulenza, Ricerca e Innovazione di Engineering
gabriele.ruffatti@eng.it

L'esigenza di Engineering è quella di essere un'organizzazione efficiente che, tramite l'esecuzione ottimale dei processi interni, consentisse una continua crescita nel mercato con il mantenimento di assetti organizzativi stabili, anche se tatticamente adattabili all'evoluzione del contesto indotta dal mercato stesso. Per ottenere ciò ha ritenuto indispensabile riferirsi ad una cultura del processo intesa, nell'accezione di Myers¹, come *“insieme di assunzioni di base condivise che un gruppo di individui ha riconosciuto come mezzo di risoluzione dei suoi problemi o che ha aiutato il gruppo ripetutamente nel raggiungimento dei propri obiettivi”*.

Il tema da sempre critico da affrontare è quello relativo all'introduzione in un'organizzazione IT di processi efficaci supportati da un'infrastruttura efficiente per affrontare i processi dello sviluppo, in primis quello di *test*.

Prima di entrare nel merito dell'approccio di Engineering al test del software, è opportuno approfondire come l'azienda affronta in generale il tema della metodologia di sviluppo.

La metodologia di sviluppo in Engineering

Definire e implementare nella realtà di ogni giorno processi di sviluppo software non è semplice e non vi sono soluzioni univoche. L'aspetto importante è che i processi siano definiti e che le organizzazioni li adottino con azioni precise coinvolgendo diverse tipologie di attività, tra cui la gestione del processo, le politiche aziendali, il training, gli audit, la raccolta di misure.

L'attenzione di Engineering in questi anni è stata rivolta da un lato a rafforzare la “cultura del processo” nell'organizzazione, diffondendola nelle diverse unità operative e mantenendola viva anche con il supporto di un nuovo modello organizzativo; dall'altro a dare vita ad un sistema virtuoso che, attraverso l'utilizzo di approcci riconducibili a modalità uniformi di un sistema di misura e valutazione e di un feed-back efficace, consentisse di migliorare il processo produttivo aziendale per fornire risultati di qualità con un corrispondente contenimento dei costi, anche attraverso la riduzione dei costi della “non qualità”.

¹ C. Myers, *Ingredients of a successful improvement effort*, SEPG Conf. Proc., SEPG – 1996.

Engineering ha operato in modo che il suo processo di sviluppo fosse inserito nell'insieme dei modelli dell'organizzazione, per essere coerente con la caratterizzazione dell'azienda e la definizione dei suoi obiettivi. Da questo deriva la disponibilità di un sistema organizzativo e di procedure che sono facilmente “adattabili e contestualizzabili” rispetto alle esigenze delle diverse unità produttive dell'azienda e rispetto a quelle indotte dal mercato.

Relativamente agli elementi che influenzano lo sviluppo, Engineering ha individuato cinque dimensioni – le persone, il processo, il prodotto, la tecnologia e la creatività – quali fattori determinanti dei costi, dei tempi di produzione e della qualità del codice. Un buon metodo di sviluppo deve quindi individuare le “practice” di supporto che favoriscano l'interazione sinergica di tali elementi.

Il test del software in Engineering

Il test in Engineering è considerato un processo “autonomo” in quanto parallelo e pervasivo rispetto all'intero processo di sviluppo del software. È un approccio che procede in modo continuo dall'avvio del progetto fino alla sua conclusione.

Il test è l'attività più critica del processo di sviluppo software: l'accettazione da parte del committente del prodotto sviluppato rappresenta il momento in cui sono verificate le aspettative espresse o non espresse dal primo e quanto compreso e realizzato sulla base delle specifiche dei requisiti.

Secondo le norme ISO 9000 ed il modello CMMI, presupposto fondamentale per il buon esito del test è la *completezza e non ambiguità dei requisiti*; concetto rafforzato dalla mai abbandonata adozione di processi di sviluppo “tradizionali” come il modello a cascata che presuppone il consolidamento e l'approvazione dei requisiti prima di passare alla fase di progettazione e sviluppo².

Ma nella realtà degli sviluppi progettuali si incontrano diverse situazioni: i requisiti sono incompleti, variano nel corso del progetto e risultano spesso essere ambigui; il committente, pur avendo validato i requisiti, contesta a volte l'accettazione del prodotto; le specifiche dei requisiti o le specifiche tecniche da cui partire per disegnare i casi di test non sono sufficientemente dettagliate o aggiornate con le variazioni intercorse; la data di consegna del prodotto viene anticipata rispetto ai piani e si ripercuote sulle attività di test che vengono di conseguenza ritardate o ridotte quando non sono sottostimate.

² La principale motivazione del mai avvenuto abbandono di modelli a cascata o riconducibili ad esso risiede principalmente nelle modalità contrattuali che sovente presuppongono una dettagliata definizione di requisiti non più modificabile a valle.

Qualunque sia il ciclo di sviluppo adottato il test non è una fase del progetto ma un processo ben integrato con il processo di sviluppo che inizia dalla fase di analisi dei requisiti fino al momento dell'accettazione del prodotto da parte del committente.

Ovviamente il test non deve essere focalizzato solo sulla validazione dei requisiti utente, ma deve indirizzare anche altri elementi, quali i rischi, la disponibilità dell'applicazione, le prestazioni e l'usabilità.

Obiettivo del test non è verificare in modo esaustivo tutto il codice con tutti i possibili input al fine di trovare tutti i possibili errori, ma:

- ottenere sufficienti conferme sul fatto che il sistema risponde ai requisiti ed è “sicuro”
- dimostrare che il prodotto “opera correttamente” nell'ambiente di esercizio o in un ambiente ad esso conforme, nel rispetto di eventuali vincoli contrattuali e di specifiche caratteristiche del progetto.

Da qui discende che *l'obiettivo del processo di test* è:

- fornire visibilità dei tempi, dei costi e di eventuali criticità per la realizzazione dei test;
- definire gli obiettivi di test, al fine di progettare ed eseguire tutti i casi prova necessari per rendere il sistema conforme ai requisiti espressi dal committente e rimuovendo le anomalie rilevate;
- documentare il lavoro svolto come attestato verso il committente della qualità del prodotto;
- diminuire sensibilmente il rischio di esito negativo in fase di collaudo;
- misurare l'efficacia/efficienza del processo di test per attivare un continuo miglioramento dello stesso.

Ma è opportuno sottolineare che l'approccio più efficiente al test è dato non dal verificare i casi di prova che danno esito positivo, ma dal considerare le condizioni “negative”.

Scopo del test è dimostrare che il sistema NON funziona, ovvero che non soddisfa i requisiti (ovvero: verificare cosa il sistema non fa, non quello che fa).

Nell'approccio Engineering il test è suddiviso secondo vari livelli riferiti agli stadi di “aggregazione” in cui un sistema software, a partire dalle sue unità

elementari, deve essere verificato. Si tratta di un processo di test integrato in cui:

- si individuano ed attuano tutte le opportune attività di verifica nel corso del processo di sviluppo (verifica dei requisiti, ispezioni, *peer reviews*)
- si eseguono iterativamente tutti i livelli di test (unitario, di integrazione, di sistema);
- sono previste, per ognuno di questi, le fasi di pianificazione, progettazione, esecuzione;
- sono individuati gli opportuni tipi di test orientati alla verifica delle specifiche categorie di attributi (funzionali, prestazionali, di usabilità, ecc.).
- si individua un insieme completo di strumenti che supportino, per ogni piattaforma di sviluppo, i livelli ed i tipi di test definiti, che assicurino la piena ripetibilità dell'esecuzione dei test e che permettano, ove possibile, di automatizzare sia l'esecuzione che la produzione della reportistica di test;
- si individua un'organizzazione di test focalizzata sulla sua progettazione ed esecuzione, per assicurare competenze specifiche che garantiscano oggettività ed atteggiamento di autonomia operativa rispetto alle attività di sviluppo; questa può essere separata rispetto al gruppo di sviluppo (gruppo di test specifico) o costituita mediante assegnazione di responsabilità specifiche ad alcuni membri del gruppo di lavoro.

Il successo di un processo di test è dato non tanto dal condurre un test esaustivo per realizzare un software esente da difetti, ma dal riuscire ad ottenere, in tutte le sue componenti, il giusto equilibrio di economicità al fine di “ottenere il massimo risultato con il minimo sforzo”.

Non esiste quindi una regola generale in materia: è la capacità e l'esperienza del project manager che rispetto ad un processo di test definito sa equilibrare tutti gli ingredienti e li sa adattare al contesto del proprio progetto al fine di realizzare il software con l'adeguato livello di correttezza. L'unico aspetto prescrittivo risiede nel fatto che, a partire dalla definizione di un Master Test Plan, l'equilibrio deve essere trovato utilizzando almeno un po' di tutti gli ingredienti, ovvero evitando di saltare attività di definizione della strategia, pianificazione e progettazione per concentrarsi unicamente nell'esecuzione di un system test o, peggio, solo di un test funzionale.

Il processo di test in Engineering prevede quindi che:

- venga definito un Master Test Plan che contempli tutti gli aspetti rilevanti per il risultato di un buon test: l'analisi dei fattori determinanti nello specifico contesto, la definizione della strategia di test, l'individuazione delle diverse responsabilità e delle risorse coinvolte, la predisposizione di un'adeguata documentazione e la pianificazione del processo nelle sue fasi di progettazione, esecuzione e accettazione interna;
- siano individuati i diversi livelli in cui un test può essere eseguito, in funzione delle caratteristiche organizzative, tecnologiche e funzionali del progetto: test statico, unitario, di integrazione, di sistema, di integrazione dei sistemi e di accettazione interna;
- vengano utilizzate tecniche di verifica in tutto il ciclo di sviluppo del progetto, a partire dalla definizione dei requisiti, utilizzando anche tecniche quali *ispezioni e revisioni*, *walkthrough*, *peer review*;
- vi sia possibilità di scelta fra non meno di venti tipologie di test, molte delle quali descritte in questo libro, in modo da offrire un ampio spettro di scelta per corrispondere anche a progetti con caratteristiche non tradizionali,
- vi sia conoscenza diffusa di diverse tecniche di test, quali white-box, black-box, top-down, bottom-up, critical modules, risk-based test, exploratory test, test driven e test-first development;
- sia disponibile una libreria di *best practice*³ di test, appartenente alla *knowledge base* aziendale volta a consolidare le *lessons learned* dei progetti e a favorire il riuso di componenti, tecniche ed esperienze, che aiuti il project manager ed il gruppo di lavoro a prevenire le cause di errori;
- vi sia disponibilità di un insieme di strumenti per dare supporto al processo di test in tutte le sue fasi in modo automatico quando il contesto lo suggerisce o lo richiede.

A questo proposito, va sottolineata la presenza in Engineering di uno specifico Centro di Competenza per le Tecnologie della Qualità. Questa unità operativa, afferente alla Direzione Ricerca e Innovazione, è costituita da un gruppo di persone che si occupa di:

³ Con il termine *best practice* si intendono “approcci collaudati allo sviluppo software che, utilizzati nel loro insieme, prevengono le cause profonde dei problemi di sviluppo e contribuiscono al successo dello sviluppo di un sistema informatico”.

- consolidare e mantenere aggiornata la competenza su metodi e strumenti per la valutazione della qualità;
- raccogliere e diffondere *best practice* e utilizzare strumenti di gestione progetti e sviluppo software (*Application Lifecycle Management, Testing, Project Automation*);
- sviluppo di soluzioni per monitorare la qualità di prodotti, processi e servizi IT;
- fungere da osservatorio permanente per la valutazione e l'adozione di strumenti a supporto dell'intero ciclo di sviluppo del software, con un particolare orientamento verso soluzioni open source, non solo per le loro caratteristiche di economicità, ma soprattutto per l'elevato livello di flessibilità che offrono;
- progettare e implementare ambienti integrati a supporto dei processi di sviluppo e dei servizi IT.

In questo centro di competenza è presente una particolare esperienza sul *Project Automation* in generale e, in particolare, nell'utilizzo di soluzioni per il test automatico funzionale, di prestazione e di sicurezza, tanto da costituirsi anche come unità di intervento a supporto di specifiche esigenze progettuali nelle numerose attività dell'azienda.

Tale gruppo di lavoro è anche responsabile della piattaforma open source Spago4Q (www.spago4q.org), soluzione a supporto della raccolta di misure ed analisi di qualità di prodotti, processi di sviluppo e servizi.

In conclusione, è opportuno sottolineare che il test, come tutti gli altri processi di sviluppo, è un processo vivo che chiede un costante aggiornamento rispetto alle evoluzioni delle tecnologie e dei modelli di business dell'Information Technology. Proprio per questo, da qualche anno è stata data particolare attenzione alle problematiche specifiche del test delle Architetture a Servizi in coerenza con l'evoluzione del mercato che inizia a spostare l'attenzione verso modelli SaaS o Cloud che sposteranno, almeno in parte, l'attività dei produttori di soluzioni informatiche dallo sviluppo di applicazioni allo sviluppo di servizi ed alla loro integrazione in architetture complesse dove le problematiche di test si spostano dalla verifica di rispondenza del singolo servizio a quelle di integrazione in un'architettura interoperabile ed eterogenea.

2 Introduzione

2.1 Il collaudo del software

La fase di verifica e validazione – comunemente detta “test” o “collaudo” o “test e collaudo” – riveste un’importanza fondamentale all’interno del ciclo di sviluppo del software ai fini del raggiungimento della qualità del prodotto finale. Essa, infatti, permette di valutare il livello di qualità raggiunto dall’applicazione sviluppata verificandone la corrispondenza alle specifiche iniziali, rilevando gli errori e permettendone la correzione.

Ci sono molti libri che parlano di qualità del software, di test e di collaudo. Perché allora un nuovo testo sull’argomento?

Nonostante la maturità raggiunta dalla branca ingegneristica dello sviluppo software (Software Engineering) l’attività di test rimane ancora un’arte oscura, non ben conosciuta dalla maggior parte degli sviluppatori ed è poco insegnata nelle università. Ne consegue che i giovani informatici approdano al lavoro senza un’adeguata conoscenza della tecnica di testing. Ma anche nelle aziende di software la pratica non sempre è conosciuta in tutta la sua potenzialità e, anche quando conosciuta, è poco utilizzata come dovrebbe.

Questo lavoro vuole fornire un contributo alla divulgazione della pratica del collaudo del software fornendo un approccio integrato di temi importanti e tra loro correlati: la qualità del software ed una delle principali attività per il suo controllo, il testing appunto. La qualità è qui intesa come la caratteristica finale che valorizza il software sviluppato, il test è inteso come l’attività per valutare il grado di raggiungimento di tale qualità e per sancirne l’accettazione finale da parte del cliente.

Un’altra caratteristica del libro è quella di inquadrare l’attività di test nell’ambito degli standard esistenti e di fornire una panoramica della sua descrizione nei diversi modelli, inclusi quelli di maturità.

Un terzo fattore che caratterizza il libro è rappresentato da una serie di suggerimenti pratici per applicare le best practice del testing descritte in un’organizzazione permanente di testing (*Test Factory*) oppure all’interno di un singolo progetto di sviluppo software.

Il presente lavoro è frutto dell'esperienza di molti professionisti del settore e quindi riassume le migliori pratiche sviluppate negli anni, condite con suggerimenti pratici tratti dall'esperienza quotidiana.

2.2 Il software

2.2.1 Che cos'è il software

Il software è la traduzione tecnologica di una soluzione identificata a fronte di un'esigenza specificata o di un problema definito. Sappiamo anche che la qualità della soluzione sviluppata dipende da più fattori: la corretta definizione dell'esigenza (o del problema) da indirizzare, il corretto utilizzo della tecnologia più appropriata, la bontà del processo di sviluppo seguito e la competenza del gruppo di lavoro. L'ultimo fattore è di enorme importanza e su di esso si concentra quanto esposto nel libro.

Possiamo definire la qualità del software sviluppato – in ultima ed estrema sintesi⁴ – come il risultato finale del nostro lavoro valutato in base alla capacità di fornire una soluzione valida, in maniera completa e corretta, all'esigenza espressa dal committente. Se l'esigenza si configura come un problema (di business e/o organizzativo) la soluzione sviluppata costituisce un elemento critico di ulteriore rilevanza.

2.2.2 Lo sviluppo software

Un progetto di sviluppo software rappresenta un investimento importante per un'azienda. L'impegno di risorse economiche e il tempo atteso per ottenere il risultato finale sono giustificati solo dalla necessità di indirizzare un'esigenza ai fini del business aziendale. Se si potesse, a parità di costi, fare a meno di realizzare un progetto lo si eviterebbe molto volentieri. Risolvere il problema posto o indirizzare l'esigenza dichiarata è quindi l'unico aspetto che conta. E la qualità rappresenta “quanto correttamente” il problema sia stato risolto o l'esigenza indirizzata.

⁴ Nei paragrafi successivi del libro si fornisce una definizione più accurata e condivisa della qualità del software.

Lo sviluppo del software, in particolare, è un'attività intellettuale ad alto contenuto umano: una sintesi complessa tra creatività e utilizzo di tecnologie. La valutazione finale che il cliente dà della soluzione realizzata rappresenta un giudizio inappellabile. Il livello di soddisfazione indica la bontà del nostro lavoro. Decreta il successo o l'insuccesso del progetto.

Ogni attività umana è soggetta a errori ed anche lo sviluppo del software non fa eccezione; ne deriva che il software realizzato conterrà errori che dovranno essere rimossi prima della sua consegna al cliente. L'attività di testing ha dunque tale obiettivo: **scoprire gli errori e permetterne la correzione**. Possiamo concludere che l'attività di test può essere semplificata, ottimizzata ma non esclusa, mai!⁵

Gli errori immessi durante le varie fasi di sviluppo differiscono in quantità e tipologia secondo l'organizzazione (competenza, esperienza e motivazione del gruppo di lavoro), il processo di sviluppo adoperato (a cascata, iterativo, incrementale, agile, ecc.), le tecnologie adoperate (linguaggi, sistemi operativi, sottosistemi, connessioni, ecc.) e la specificità dell'applicazione (legacy, Object-Oriented, applicazione Web, real-time, ecc.).

Se si potesse conoscere il numero e la tipologia degli errori immessi durante l'interpretazione dei requisiti, la progettazione dell'applicativo e la codifica del software si potrebbe sicuramente ridurre l'attività di test esclusivamente all'individuazione e rimozione di tali errori. Nella quasi totalità dei progetti, purtroppo, non conosciamo tali elementi e dovremmo progettare e ottimizzare i test in modo da scoprire il maggior numero possibile di errori. Il test sarà quindi più efficace (cioè capace di trovare errori) ed efficiente (utilizzando il minimo di risorse) se sarà ben progettato, organizzato e condotto.

2.2.3 Quantità di errori immessi e rimossi nel software

Il software (a differenza dell'*hardware*) è unico e sempre diverso da tutti gli altri software realizzati, anche se ne duplica parti importanti o risolve le stesse problematiche.

L'immissione (involontaria) degli errori nel software dipende da vari fattori: competenza delle persone, maturità del processo di sviluppo, complessità del

⁵ L'osservazione (“--- l'attività di test può essere semplificata, ottimizzata ma non esclusa, mai!”) è riportata nel libro perché, anche se ovvia, rappresenta un tema discusso in quasi tutte le organizzazioni di sviluppo software alla perenne ricerca di quel “miracolo” mai realizzato di evitare le attività di test e ridurre, di conseguenza, il costo e i tempi di realizzazione del software.

problema affrontato e tecnologie adoperate. Il numero degli errori immessi varia secondo la fase del ciclo di vita; pochi errori sono generalmente immessi nelle fasi alte del ciclo di vita (analisi e disegno) mentre molti di più sono quelli immessi nelle fasi successive (progettazione di dettaglio e codifica). I primi, anche se pochi in numero, influenzano maggiormente la qualità del software finale sviluppato. Sono perciò da individuare il più presto possibile.

L'andamento della rimozione degli errori nelle diverse fasi del ciclo di vita del software segue il modello mostrato nella figura che segue. I valori sono normalizzati e riferiti ad una metrica dimensionale del software (1KLOC o 100FP)⁶. La rimozione degli errori nelle fasi alte del ciclo di vita (prima cioè che sia sviluppato il codice) è fatta tramite le ispezioni dei documenti tecnici (requisiti, specifiche, disegno, ecc.), mentre la rimozione dopo la produzione del codice è fatta tramite le attività di test. Le ispezioni sono revisioni tecniche dei documenti e rappresentano il “test statico”.

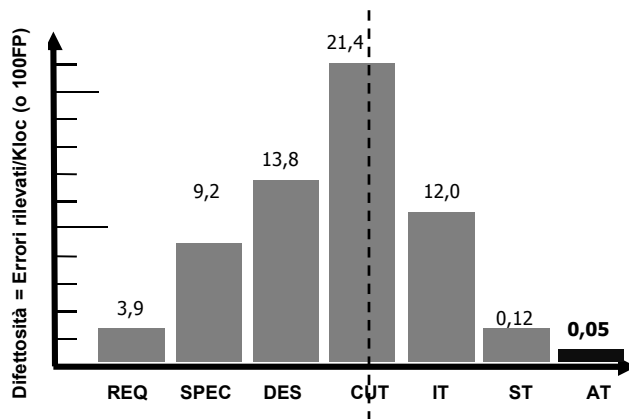


Figura 1. Curva di rimozione degli errori durante le fasi del ciclo di vita.

La rimozione degli errori effettuata contestualmente alla fase in cui si presuppone essi siano stati inseriti permette di ridurre drasticamente la loro propagazione degli errori alle fasi successive come descritto nell'articolo cui si fa riferimento nella nota a piè di pagina.

⁶ Si tratta di due metriche relative ad attributi diversi del software: la prima (KLOC) si riferisce al numero di linee di codice (migliaia di linee di codice, nel nostro esempio) ed il secondo alle funzionalità del software. L'equivalenza 1 FP = 100 LOC non è quindi corretta dal punto di vista delle metriche; viene generalmente utilizzata, in maniera impropria, come semplice strumento di paragone per chi è abituato da sempre a ragionare in termini di dimensione del codice sorgente.

2.2.4 Propagazione degli errori nel software

La teoria della propagazione degli errori nel software è stata presentata per la prima volta nel 1981 nell'IBM System Scientific Institute a fronte di una ricerca condotta sullo sviluppo del software nei maggiori laboratori della compagnia. La teoria è sintetizzata graficamente nella figura riportata di seguito.

Gli errori provenienti dalla fase precedente sono ereditati in quella attuale aggiungendosi a quelli immessi ex novo. Parte degli errori ereditati dalla fase precedente genera, a loro volta, altri errori secondo un fattore di amplificazione 1: x. Questo fattore dipende dalla fase del ciclo di sviluppo, dalla complessità del software e dalla tipologia di errore. L'esperienza ha insegnato, ad esempio, che alcuni errori insiti nell'interpretazione dei requisiti hanno un fattore di amplificazione fino a 1:5 e si propagano in tutte le fasi successive. Così un solo errore nell'interpretazione di alcuni requisiti chiave generava fino a 5 errori nelle specifiche, fino a 25 errori nel disegno e fino a 125 errori nel codice! Alcuni errori di progettazione, invece, possono avere un fattore di amplificazione anche 1:7 (esempio: un errore nella progettazione della base dati può generare un numero grandissimo di errori nei moduli che li gestiscono). Solo la registrazione degli errori rilevati e l'analisi delle loro caratteristiche può farci capire quali errori siano più dannosi e come evitarli. Il modello mostra quindi quanto sia importante ridurre il numero totale di errori trasmessi alla fase successiva e contenere, di conseguenza, il numero di errori ricevuti in eredità da quella precedente. A tale scopo si deve agire su due fronti: ridurre il numero di errori generati ex novo in ogni fase e aumentare il numero di errori rimossi in ogni fase⁷. La figura che segue mostra in maniera schematica il processo di propagazione degli errori nelle fasi successive.



Figura 2. Schema di propagazione degli errori
(fonte: IBM System Scientific Institute, 1981).

⁷ L'articolo "Il controllo della qualità al servizio del business" pubblicato sul numero 2 novembre 2007 della rivista Qualità On-line di AICQ descrive tale fenomeno (vedi Bibliografia).

Le “revisioni tecniche” (ispezioni) eseguite nelle fasi alte del processo di sviluppo sono quindi cruciali per ridurre il costo relativo alla correzione degli errori rilevati durante il test e l’esercizio.

2.2.5 Tipologia di errori immessi e rimossi nel software

Un ulteriore elemento da tenere in debita considerazione per quanto riguarda la rimozione degli errori è quello di conoscere le diverse tipologie di errori comunemente immessi (rilevati e rimossi) durante lo sviluppo del software. Conoscere tale elemento permette di intervenire adeguatamente per ridurre gli errori immessi (formazione del personale, predisposizione di checklist, ecc.). La tabella che segue riporta l’analisi effettuata su numerosissimi progetti⁸.

Tabella 1. Tipologia di errori comunemente immessi nel software
(fonte: R. S. Pressman)

Tipologia	Perentuale
Specifiche errate o incomplete (IES)	22%
Requisiti errati o incompleti (IER)	17%
Errori nella rappresentazione dei dati (EDR)	14%
Test incompleto o errato (IET)	10%
Inconsistenza nell’interfaccia (ICI)	6%
Errore nella traduzione del disegno nel linguaggio di programmazione (PLT)	6%
Errore logico di progettazione (EDL)	5%
Deviazione intenzionale dalle specifiche (IDS)	5%
Documentazione incompleta o imprecisa (IID)	4%
Violazione degli standard di programmazione (VPS)	3%
Interfaccia uomo-macchina (HCI) ambigua o inconsistente (HCI)	3%
Miscellanea di altri errori (MIS)	5%

⁸ Esistono diversi studi effettuati al riguardo e disponibili in letteratura. La tabella qui riportata si basa sui dati forniti da R.S. Pressman (vedi Bibliografia).

Dall'analisi dei dati riportati nella tabella precedente risulta che i primi cinque elementi costituiscono da soli il 70% del totale degli errori. Seguendo l'approccio suggerito da Pareto sarebbe sufficiente indirizzare con opportune azioni tali elementi per risolvere buona parte del problema discusso.

2.2.6 Costo di rimozione degli errori nel software

Un ultimo elemento di riferimento importante è quello relativo al costo unitario di rimozione degli errori nelle diverse fasi del ciclo di vita. I dati riportati nella tabella che segue mostrano quanto cresca il costo/l'impegno (in termini di numero di ore persona) man mano che si passa alle fasi successive. Rimuovere gli errori contestualmente alle fasi in cui si presuppone siano inseriti (di cui si è già detto a proposito della propagazione degli errori) è importante anche, e soprattutto, ai fini dell'economicità del progetto. La rappresentazione mostra l'andamento esponenziale dei costi di rimozione degli errori.

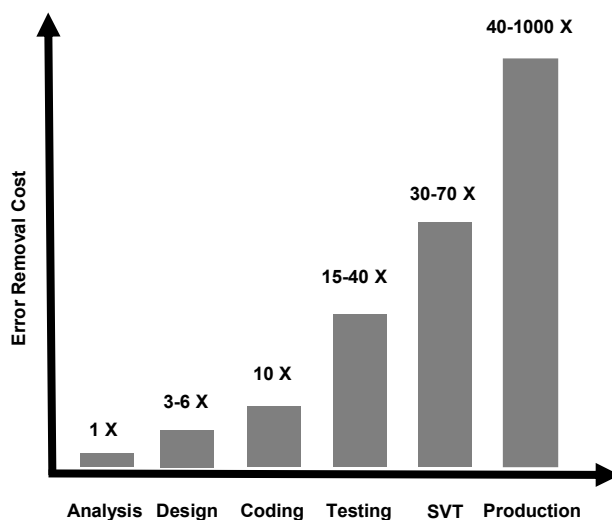


Figura 3. Costo di rimozione degli errori nelle fasi (fonte: Pressman)

Si vede bene come i fattori economici del progetto siano associabili alle attività di test e collaudo. Il costo relativo alla scoperta e rimozione di un errore tramite una revisione tecnica è nettamente inferiore al costo di rimozione dello stesso errore tramite l'esecuzione di un caso di prova e relativo debugging da parte dello sviluppatore. Riducendo il numero di errori previsti in fase di test si potrà ridurre di conseguenza l'impegno per l'attività di collaudo. Tutto ciò a parità di errori residui nel prodotto finale.

La conclusione è dunque semplice ma, purtroppo, ignorata nella maggior parte dei progetti. Ciò cui si assiste in moltissimi progetti, infatti, è quello di affidare alla sola fase di test la rimozione degli errori. Tale impegno risulta quindi oneroso (troppi errori si sono propagati fino alla fase di test e collaudo) e si arriva alla necessità di “tagliare” le attività di test con relativo impatto negativo sulla qualità del software finale.

La raccomandazione che ci sentiamo di fare è quella di pianificare in ogni progetto attività di “revisione tecnica” nelle fasi alte del ciclo di vita e di limitare di conseguenza le attività di test alle prove più significative con ottimizzazione dei costi e miglioramento della qualità del prodotto finale.

2.3 Che cos'è il collaudo del software

2.3.1 Generalità

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 per indicare, il primo, le prove effettuate dal gruppo di sviluppo (test) durante la realizzazione del progetto ed il secondo quelle effettuate dal cliente in fase di accettazione del prodotto finale (collaudo). In realtà i due termini hanno lo stesso significato: collaudo è il termine italiano corrispondente a quello inglese *test*. Spesso si usa anche il termine “collaudo interno”, in contrapposizione al termine “collaudo esterno”, per indicare il test effettuato dal gruppo di sviluppo.

2.3.2 Psicologia del tester

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 af-