Collaudo del software

Processo, metodi e tecniche, metriche e strumenti, guida all'applicazione in azienda

Ercole F. Colonese

Versione 2.1 – Giugno 2007

4

Note sulla versione 2.1

La presente versione (2.1) apporta alcune modifiche significative alla versione precedente (2.0) a seguito della revisione apportata sul campo e all'introduzione di un intero capitolo su come applicare la presente metodologia in azienda. La versione 2.0 rappresenta invece una rivisitazione completa della metodologia sperimentata presso il bacino di imprese informatiche italiane medio piccole. La nuova struttura risponde alle esigenze anche di progetti software più complessi e/o di dimensioni maggiori. Inoltre, riporta alcune correzioni ed aggiunte suggerite dall'utilizzo recente della metodologia in alcuni progetti di dimensioni e complessità elevata.
Le principali modifiche apportate alla nuova versione della metodologia riguardano:

- Nuova struttura del documento per facilitarne la lettura evidenziando gli elementi metodologici e quelli operativi;
- Inserimento di esempi che aiutano la comprensione dei vari temi;
- Aggiunta di un capitolo dedicato ai modelli (template) dei documenti di test da realizzare e di uno con le linee guida di come adottare la presente metodologia in azienda;
- Aggiornamento della Bibliografia.

Versione del documento

2000	2005	2006	2007	2008
v1.0	V1.1	V2.0	V2.1	

Ringraziamenti

Sentiti ringraziamenti vanno a Fabrizia Finamore che ha contribuito significativamente alla realizzazione della prima versione del documento, da cui è partita la stesura di questa nuova versione, anche se profondamente cambiata. Altrettanti ringraziamenti vanno ad Enzo Troiano che ha partecipato al progetto di divulgazione ed applicazione della metodologia presso il gruppo di piccole e medie imprese informatiche italiane. Un ultimo, ma non meno caloroso, ringraziamento va ad Anna Rita Laurenzi che ha utilizzato la metodologia nella sua versione attuale in un progetto reale presso un grande cliente pubblico. I commenti ed i suggerimenti ricevuti da più persone sono stati di grande aiuto ed a loro va il mio personale ringraziamento.

Manuali di sviluppo software

Il presente manuale fa parte di una serie di documenti, in parte già disponibili ed in parte in fase di realizzazione, che mirano a fornire una vista dei vari temi dello sviluppo software. Essi riportano l'esperienza pratica maturata dall'autore in molti anni di lavoro. L'elenco dei manuali e del relativo stato di completamento riportato qui di seguito si riferisce alla data in cui è rilasciata la presente versione del manuale in oggetto.

1.	Sviluppare software oggi	(in preparazione)
2.	Strategia di sviluppo e ciclo di vita del software	(in fase di completamento)
3.	Project Management	(disponibile)
4.	**Collaudo del software (testing)**	**(disponibile)**
5.	Collaudo del software in Rete	(in preparazione)
6.	Qualità del software	(disponibile)
7.	Modello ISO 9126 per la qualità del software	(disponibile)
8.	Metriche del software	(in fase di completamento)
9.	Usabilità del software	(disponibile)
10.	Usabilità dei siti Web	(disponibile)
11.	Metodi e tecniche dello sviluppo software	(in fase di completamento)
12.	Introduzione al modello CMMI	(in fase di completamento)
13.	Glossario dei termini dello sviluppo software	(disponibile)
14.	Bibliografia	(disponibile)

I manuali completati sono disponibili nel sito dell'autore (http://www.colonese.it/pubblicazioni/htm).

A chi sono rivolti i mauali

I manuali sono rivolti a

- Studenti universitari che seguono i corsi di ingegneria del software e che giudicano utile documenti che descrivano i temi trattati durante le lezioni in maniera più completa e strutturata;
- Capi progetto (Project Manager) che necessitano di una guida di riferimento per le attività di pianificazione e controllo delle attività di collaudo del software;
- Sviluppatori, collaudatori, personale dell'assicurazione qualità e consulenti di processo che necessitano di una guida di riferimento per approfondire i temi relativi al loro ruolo.

Sommario

Indice degli argomenti

Indice delle figure

Indice delle tabelle

1 Introduzione al testing

1.1 La metodologia proposta

La presente metodologia definisce i principi base del collaudo del software.

La letteratura sulle metodologie di test è vasta; i suoi principi fondamentali, al contrario, sono ancora ignorati da molte organizzazioni che operano nel campo dello sviluppo del software.

La metodologia proposta è particolarmente orientata alle organizzazioni software di piccole e medie dimensioni (PMI) che non abbiano definito una propria metodologia. La sua validità si estende anche ad organizzazioni di grandi aziende che sviluppino progetti complessi.

Pur basata su principi ormai consolidati da moltissimi anni, il presente lavoro può ritenersi originale in quanto frutto di una ricerca specifica che ha visto coinvolto un gruppo nutrito di piccole e medie aziende informatiche italiane.

Gli elementi più critici rilevati nell'ambito della ricerca condotta sulle aziende coinvolte sono riassunte nei seguenti punti:

- Le attività di test sono eseguite dagli stessi sviluppatori (inoltre risulta evidente l'assenza di una competenza specifica sulle metodologie di test).
- Poco spesso (o quasi mai) è definita una strategia dei test basata anche sui requisiti qualitativi (non funzionali).
- La pianificazione dei test (tempi e risorse) è fortemente condizionata dai tempi di rilascio, cioè dal tempo minimo disponibile tra la fine dello sviluppo e la consegna.
- Ogni ritardo nei tempi di sviluppo erode il tempo da dedicare alle attività di test.
- La progettazione dei test è spesso informale (non documentata) ed i test di regressione sono fatti "a memoria" da chi ha precedentemente condotto i test.
- Difficilmente è condotta una fase di test di regressione a seguito di modifiche al codice, anche per la mancanza di formalizzazione dei casi di test, oltre che per la mancanza di tempo e risorse disponibili.
- La copertura dei test non è sempre verificata (non è prodotta, per esempio, una matrice di copertura dei test o qualcosa di equivalente).

- La conclusione delle attività di test è determinata dalla data di consegna piuttosto che dai risultati raggiunti (stato di completamento dei test, copertura dei test eseguiti, curva di rimozione degli errori, numero di errori residui, ecc.).
- La registrazione delle anomalie riscontrate è carente (spesso assente) e comunque non strutturata. Ciò non permette di utilizzare dati storici che permettano di pianificare meglio i test dei progetti futuri (prevedere il tasso di difettosità residua, le tipologie di errori più frequenti, creare la curva di rimozione degli errori, verificare l'efficacia delle attività di rimozione degli errori, ecc.). Da ciò l'impossibilità di pianificare lo sforzo necessario alla fase di test per garantire il livello di qualità del software atteso.

Questi fattori di criticità sono stati indirizzati nella presente metodologia con opportune azioni, metodi e tecniche semplici ed efficaci.

1.2 Obiettivi della metodologia

Gli obiettivi principali della metodologia di test proposta sono quelli di assicurare metodi, tecniche e principi semplici ed efficaci che permettano ai progetti software di dare:

- ai Clienti l'attesa "garanzia di piena conformità del software sviluppato ai requisiti concordati";
- alle Aziende fornitrici "garanzia di efficacia dei metodi adottati" e previsione dei tempi e dei costi.

1.3 Riferimenti a standard di mercato

Esistono moltissimi modelli di collaudo del software, molti sono puramente teorici, altri definiscono requisiti per un sistema qualità in cui le attività di testing sono parte del processo di realizzazione del prodotto, altri ancora costituiscono delle linee guida da adottare con il buon senso all'interno del proprio processo di sviluppo software. Qui si vogliono ricordare solo due di questi modelli, in quanto considerati fra i più diffusi e conosciuti (ISO 9000 e CMM). Il primo (ISO 9000) è lo standard riconosciuto a livello mondiale. Il secondo (CMM – Capability Maturity Model) costituisce un modello per valutare il grado di maturità dei processi produttivi, tra cui è incluso il processo di Verifica e Validazione (Testing). Un terzo modello è riportato solo per amore di completezza in quanto costituisce l'evoluzione del modello CMM applicato specificatamente al testing: Testing Maturity Model (TMM).

1.3.1 ISO 9000

Lo standard ISO 9000 attuale (Vision 2000) ha compiuto un passo notevole verso il miglioramento continuo della qualità. Un'azienda software trova ora quanto necessario per portare la propria organizzazione verso un miglioramento concreto e tangibile.

Occorre però evitare di costruire cattedrali nel deserto, sistemi di qualità perfettamente aderenti alle norme ma poco utili dal punto di vista produttivo. Sistemi realizzati solo per risultare "conformi" (solo per ottenere il "bollino blu") sono molto costosi e non producono risultati economici che giustifichino gli investimenti fatti. Occorre andare alla sostanza, semplificare, condividere e poi ... agire. Con competenza, continuità e coerenza!

L'Aicq (Associazione italiana per la cultura della qualità), con il suo comitato per la qualità del software, ha realizzato un utilissimo quaderno per le organizzazioni software che descrive l'interpretazione delle norme e si suggeriscono modalità per la loro applicazione.

Il quadro normativo attuale è stato notevolmente semplificato ed è costituito da quattro documenti principali:

ISO 9000 - Fondamenti e terminologia, descrive i fondamenti dei sistemi di gestione per la qualità e ne specifica la terminologia adoperata;

ISO 9001 - Requisiti per i sistemi di gestione per la qualità, specifica i requisiti dei sistemi di gestione per la qualità che un'azienda deve soddisfare per dimostrare ai propri clienti, ed al mercato in generale, la propria capacità di fornire prodotti e servizi in linea con la qualità attesa;

ISO 9004 - Guida al miglioramento continuo, fornisce linee guida per adottare le norme ISO 9000 in maniera proficua, migliorando le prestazioni aziendali, sia in termini di efficacia che di efficienza, in modo continuativo (miglioramento continuo);

ISO 19011 - Guida alle verifiche ispettive dei sistemi di gestione per la qualità ed ambientali, fornisce una guida per le verifiche ispettive da condurre su sistemi di gestione per la qualità ed ambientali.

A queste si aggiungono norme specifiche per lo sviluppo del software, tra cui le più significative:

ISO 9126 - Le norme sulla qualità dei prodotti software;

ISO 12207 - Le norme sul ciclo di vita del software.

Ma vediamo cosa dice la norma ISO 9001 a proposito del collaudo.

Il punto della norma "**7. Realizzazione del prodotto**" indirizza la pianificazione, la progettazione, la realizzazione ed il collaudo del prodotto. In particolare, per le attività di verifica e validazione sono previsti i seguenti punti specifici.

7.3.4 Riesame della progettazione e dello sviluppo

In fasi opportune devono essere effettuati riesami sistematici della progettazione e dello sviluppo, in accordo con quanto pianificato, al fine di:

a) valutare la capacità dei risultati della progettazione e dello sviluppo di ottemperare ai requisiti,

b) individuare tutti i problemi e proporre le azioni necessarie.

A tali riesami devono partecipare rappresentati delle funzioni coinvolte nelle fasi di progettazione e di sviluppo oggetto del riesame. Le registrazioni dei risultati dei riesami e delle eventuali azioni necessarie devono essere conservate.

7.3.5 Verifica della progettazione e dello sviluppo

Devono essere effettuate verifiche, in accordo con quanto pianificato, per assicurare che gli elementi in uscita dalla progettazione e dallo sviluppo siano compatibili con i relativi requisiti in ingresso. Le registrazioni dei risultati delle verifiche e delle eventuali azioni necessarie devono essere conservate.

7.3.6 Validazione della progettazione e dello sviluppo

Deve essere effettuata la validazione dalla progettazione e dallo sviluppo in accordo con quanto pianificato, per assicurare che il prodotto risultante dalla progettazione e dallo sviluppo sia in grado di soddisfare i relativi requisiti per l'applicazione specificata o, dove conosciuta, per quella prevista. Dove applicabile, la validazione deve essere completata prima della consegna o dell'utilizzazione del prodotto. Le registrazioni dei risultati della validazione e delle eventuali azioni necessarie devono essere conservate.

7.3.7 Tenuta sotto controllo delle modifiche della progettazione e dello sviluppo

Le modifiche della progettazione e dello sviluppo devono essere identificate e le relative registrazioni conservate. Le modifiche devono essere riesaminate, verificate e validate, come opportuno, ed approvate prima della loro attuazione. Il riesame delle modifiche della progettazione e dello sviluppo deve comprendere la valutazione degli effetti che tali modifiche hanno sulle parti componenti e sui prodotti già consegnati. Le registrazioni dei risultati delle modifiche e delle eventuali azioni necessarie devono essere conservate.

Tutti questi elementi sono indirizzati in maniera opportuna ed operativa dalla presente metodologia. Quindi essa risulta essere aderente alla norma ISO 9001:2000.

1.3.2 Capability Maturity Model Integration (CMMI)

Il CMMI è un modello per il miglioramento dei processi di sviluppo prodotti e servizi. Consiste di "best practices" che indirizzano le attività di sviluppo e manutenzione e che coprono l'intero ciclo di vita, dal concepimento del prodotto al suo rilascio sul mercato e in produzione presso i clienti.

Realizzato dal Software Engineering Insitute (SEI), il modello di processo da indicazioni di quanto debba essere fatto per raggiungere determinati livelli di maturità[1] in un'organizzazione (nel nostro caso in un'organizzazione che sviluppa software).

Il modello si basa su cinque livelli di maturità crescenti (da 1 a 5).

L'attuale versione del modello (v1.2) definisce una serie di sottoprocessi (dette Aree di processo – KPA) da implementare a seconda del livello di maturità da raggiungere.

Livello di maturità 1: Eseguito. Nel primo livello in realtà non esistono processi definiti completamente o comunque questi non sono seguiti nella realizzazione dei progetti. La realizzazione è fatta in base alle competenze dei singoli ed i risultati, che in alcuni casi possono essere anche ottimi, non sono ripetibili in altri progetti realizzati da altre persone.

Livello di maturità 2: Gestito. Nel secondo livello si propongono i processi tipici gestionali: Gestione dei requisiti, Pianificazione del progetto, Monitoraggio e controllo del progetto, Gestione dei fornitori, Valutazione e analisi, Assicurazione qualità per i prodotti e per i processi, Gestione della configurazione.

Livello di maturità 3: Definito. Nel terzo livello si passa alla standardizzazione dei processi. Tutti i progetti utilizzano gli stessi processi che sono adattati alle diverse esigenze dei singoli progetti in base a regole stabilite. Le aree di processo indirizzate nell'attuale versione del modello sono ben 14, tra cui i processi di verifica e validazione (testing): Sviluppo dei requisiti; Soluzione tecnica; Integrazione del prodotto; ***Verifica***; ***Validazione***; Coinvolgimento nel processo aziendale; Definizione del processo aziendale; Addestramento; Gestione integrata del progetto; Gestione integrata dei fornitori; Valutazione dei rischi; Analisi e risoluzione delle decisioni; Ambiente aziendale per l'integrazione; Definizione integrata del team di sviluppo.

Livello di maturità 4: Gestito quantitativamente. Il quarto livello prevede la gestione dei progetti in base a risultati quantitativi, oltre che qualitativi. Tutto è misurato ed i processi sono valutati in base ai risultati ottenuti. Le aree di processo indirizzate sono: Prestazione del processo aziendale; Gestione quantitativa del progetto.

[1] In realtà il modello definisce due rappresentazioni diverse dello stesso modello: una basata su 6 Capability Levels (0 – 5) ed una basata su 5 Maturity Levels (1 – 5). Le due rappresentazioni sono equivalenti, anche se formalmente distinte. La seconda, forse più diffusa è quella descritta in questo capitolo.

Livello di maturità 5: Ottimizzato. Nell'ultimo livello i processi sono gestiti in ottica di miglioramento continuo. Le aree di processo indirizzate sono: Innovazione e spiegamento aziendale; Analisi e risoluzione causale.

Per ciascun livello di maturità sono definiti obiettivi generali ed obiettivi specifici da raggiungere. Inoltre sono proposte pratiche generiche e pratiche specifiche da seguire che possono essere interpretate come "best practices".

Esempio:

A titolo di esempio si riporta quanto previsto dal modello per il processo di Verifica e per quello di Validazione, corrispondenti al processo di collaudo (test statico e test dinamico).

Il modello definisce obiettivi specifici (**SG**) e pratiche specifiche (**SP**) per ciascun processo. Definisce anche obiettivi generici (**GG**), validi per tutti i processi e pratiche generiche (**GP**) anch'esse valide per tutti i processi.

Essi sono elencate qui di seguito. Per ovvi motivi di semplificazione non viene fornito alcun dettaglio sui singoli elementi, consigliando il lettore di approfondire il tema direttamente tramite il modello.

Verification (VER)

"Scopo del processo di verifica è quello di assicurare che i prodotti intermedi (es. documento) indirizzino i requisiti specificati".

SG1 Preparare le verifiche

I prodotti intermedi sono sottoposti a verifica.

SP 1.1 Selezionare i prodotti intermedi da verificare

SP 1.2 Stabilire l'ambiente per le verifiche

SP 1.3 Stabilire le procedure ed i criteri per le verifiche

SG2 Eseguire le revisioni tecniche (Peer Review)

I prodotti intermedi selezionati sono sottoposti a revisione tecnica.

SP 2.1 Preparare le revisioni tecniche

SP 2.2 Condurre le revisioni tecniche

SP 2.3 Analizzare i risultati delle revisioni tecniche

SG3 Verificare i prodotti intermedi selezionati

I prodotti intermedi sono verificati a fronte dei requisiti specificati.

SP 3.1 Effettuare le verifiche

SP 3.2 Analizzare i risultati delle verifiche

Validation (VAL)

"Scopo del processo di validazione è quello di dimostrare che il prodotto indirizza i requisiti richiesti quando esso è collocato (utilizzato) nell'ambiente specificato".

Obiettivi specifici (SGn) e pratiche specifiche (SPn.m).

SG1 Preparazione per la validazione

Preparare la validazione del prodotto.

SP 1.1 Selezionare il prodotto da validare

SP 1.2 Stabilire l'ambiente per la validazione

SP 1.3 Stabilire le procedure ed i criteri per la validazione

SG2 Validare il prodotto

Il prodotto è validato per assicurare che esso sia adatto ad essere utilizzato nell'ambiente per cui è previsto.

SP 2.1 Effettuare la validazione

SP2.2 Analizzare i risultati della validazione

Per entrambi i processi valgono i seguenti obiettivi generici e pratiche generiche.

GG3[2] Istituzionalizzare il processo definito

GP 2.1 Stabilire la Politica aziendale (per i processi in questione)

GP 2.2 Pianificare il processo (sia di verifica che di validazione)

GP 2.3 Assicurare le risorse

GP 2.4 Assegnare le responsabilità

GP 2.5 Formare il personale

GP 2.6 Gestire la configurazione

[2] La numerazione degli obiettivi e delle pratiche non è sequenziale in quanto essa si riferisce ad una notazione in cui sono presenti anche elementi che si riferiscono ad un'altra rappresentazione del modello, detta "continuous", diversa da quella presentata qui, detta "staged". Ma ciò ha poca importanza ai fini della discussione fatta nel nostro contesto.

GP 2.7 Identificare e coinvolgere tutte le parti interessate (stakeholders)

GP 2.8 Monitorare e controllare il processo

GP 2.9 Valutare l'aderenza in maniera oggettiva

GP 2.10 Rivedere lo stato con la direzione

GP 3.1 Stabilire il processo definito

GP 3.2 Collezionare le informazioni necessarie per il miglioramento

Nello specifico, il modello suggerisce (o richiede) per ciascuna delle voci sopra citate una serie di particolari che consentono di seguire l'approccio più corretto.

La metodologia presentata in questo documento indirizza in maniera pratica tutti gli elementi definiti dal modello CMMI per i processi di verifica e validazione.

1.3.3 Testing Maturity Model (TMM)

Gli autori[3] hanno definito un modello di maturità per il processo di testing (TMM) progettato per aiutare le organizzazioni che sviluppano software a valutare e migliorare il processo di collaudo. Il modello TMM è complementare al modello CMM di cui riprende e indirizza in maniera più dettagliata gli elementi critici per il responsabile dei collaudi (test manager), per gli specialisti di testing e per i coloro che si occupano della qualità del software. Il processo di collaudo come definito dal modello TMM indirizza in maniera più estesa tutte le attività inerenti la qualità del software. Gli autori dichiarano che l'utilizzo del modello al processo di testing ha un impatto positivo sulla qualità del prodotto finale, sulla produttività e sul tempo di realizzazione dei progetti.

Anche il modello TMM prevede cinque livelli di maturità del processo di collaudo. Ecco, a titolo di esempio, gli obiettivi dei singoli livelli.

Livello 1: Initial

Nessun obiettivo specifico.

Livello 2: Definition

Istituzionalizzare metodi e tecniche di base per il testing; Iniziare la pianificazione del processo di testing; Sviluppare gli obiettivi di testing e debugging.

Livello 3: Integration

Controllare e monitorare il processo di testing; Integrare il processo di testing nel ciclo di vita del software; Stabilire un programma di formazione tecnica; Stabilire un'organizzazione per il collaudo.

[3] Qui si fa riferimento al modello elaborato da Ilene Burnstein, Ariya Homyen, Tartip Suwanassart, Gary Safena e Rob Grom.

Livello 4: Management and Measurement

Sviluppare un processo per la valutazione della qualità del software; Stabilire un programma di misurazione del testing; Stabilire un programma di revisione per l'intera organizzazione.

Livello 5: Optimization/Defect Prevention and Quality Control

Ottimizzare il processo di testing; Implementare le misure della qualità del software; Applicare la prevenzione dei difetti basandosi sui dati disponibili.

Approfondendo la metodologia presentata in questo documento noterete che tutti questi principi sono indirizzati in maniera pratica.

1.4 Definizioni ed acronimi

1.4.1 Definizioni

La fase di verifica e validazione – comunemente detta di testing o collaudo - riveste un'importanza fondamentale all'interno del ciclo di sviluppo del software ai fini della qualità del prodotto finale. Essa, infatti, permette di valutare il livello di qualità raggiunto dall'applicazione sviluppata rilevando gli errori e permettendone la correzione.

Di seguito sono riportati i significati dei termini più comunemente usati per stabilire una comune base di partenza. Un elenco più completo è fornito nel Glossario disponibile sul sito dell'autore.

Verifica: Rappresenta l'attività volta a garantire che un determinato prodotto intermedio (prodotti di fase) operi come richiesto dal suo predecessore che rappresenta il suo input. Per esempio, la revisione di un documento di disegno verifica che esso sia aderente alle specifiche dichiarate per esso - specifiche funzionali, requisiti tecnici, standard documentali e di progettazione, ecc. -. La revisione di un modulo software (codice sorgente), a sua volta, verifica che esso sia coerente con il suo disegno tecnico e con gli standard di programmazione stabiliti. L'attività di verifica è svolta tramite "ispezioni", "revisioni", ecc.

Validazione: Rappresenta l'attività volta a garantire che un prodotto indirizzi i requisiti originali stabiliti per esso. Per esempio, la validazione di una transazione utente controlla che essa sia completata secondo le specifiche approvate, sia in termini funzionali che prestazionali. La validazione del software è svolta tramite attività di testing; la validazione del disegno può invece essere fatta tramite la valutazione di un prototipo, ecc.

Testing: Rappresenta il processo con il quale si esegue e si valuta, automaticamente o manualmente, 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.

Risultato atteso: Un insieme di requisiti o di specifiche cui devono conformarsi i risultati dei test ottenuti in output da un processo per essere accettati come validi. Un tipico esempio di risultato atteso è rappresentato dalle specifiche di una transazione on-line i cui tempi di risposta devono essere inferiori, per esempio, a 2 secondi.

Varianza: Rappresenta la discrepanza osservata tra i risultati ottenuti e quelli attesi durante i test. Può essere causata da diversi motivi: errori nel software, errori nella definizione dei risultati attesi, utilizzo di dati invalidi, ecc.

1.4.2 Acronini

AQ	Assicurazione Qualità
CCM	Change and Configuration Management
CMM	Capability Maturity Model©
CMMI	Capability Maturity Model® Integration
CT	Caso di test
DCR	Design Change Request
IEC	International Electrical Commission
ISO	International Organization of Standardization
IT	Integration Test (test d'integrazione)
JCL	Job Control Language
ODC	Orthogonal Defect Classification
PM	Project Manager
PMBOK	Project Management Body Of Knowledge
PMI	Piccole e Medie Imprese
	Project Management Institute
QA	Quality Assurance (Assicurazione qualità)
SAL	Stato Avanzamento Lavori
SCM	Software Configuration Management
SEI	Software Engineering Institute
SMART	Specifico, Misurabile, Attendibile, Realistico, Tempistico
ST	System Test (test di sistema)
SWEBOK	Software Engineering Body Of Knowledge
TMM	Testing Maturity Model
TR	Technical Report
UAT	User Acceptance Test (test di accettazione dell'utente)
UT	Unit Test (test unitario)

2 I principi e fondamenti del testing

2.1 Principi generali

I principi basilari dell'attività di test possono essere così riassunti:

- **Un'organizzazione di test non dovrebbe testare le proprie applicazioni**.

  Il responsabile del progetto (Project Manager) ed il gruppo di sviluppo non dovrebbero essere direttamente responsabili del test poiché il loro approccio psicologico è quello di 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).

- **Un programmatore dovrebbe evitare di effettuare il test delle applicazioni che ha sviluppato**.

  Il programmatore non accetta di esporre i propri errori di programmazione poiché la sua attitudine mentale è di tipo costruttivo (sviluppo) e non distruttivo[4] (processo di test).

- **Per ogni caso di test[5] è necessario specificare il risultato atteso**.

  Rispetto al componente testato, è necessario indicare precisamente i dati di input ed i relativi dati di output. Questo principio che può sembrare ovvio a prima vista è di primaria importanza poiché se il risultato di un caso di test non è predefinito, anche un risultato errato potrebbe essere interpretato come corretto.

- **I casi di test dovrebbero essere eseguiti anche per condizioni di input non valide ed inattese, oltre che per quelle valide e previste**.

  L'approccio psicologico di molti tecnici del test è quello di ignorare le condizioni di input dei test non valide ed inattese.

- **Ispezionare minuziosamente i risultati di ogni test eseguito**.

  E' bene verificare con cura il risultato del test eseguito e registrare, analizzare e correggere ogni errore rilevato.

---

[4] 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 ben descritto da G.J.Myers nel testo "The Art of Software Testing" riferito in Bibliografia.

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

- **L'analisi di un componente dovrebbe essere condotta per identificare anche gli effetti non desiderati**.

  Un software dovrà produrre un output sempre coerente con i valori di input immessi.

- **I casi di test rappresentano un investimento che dovrà essere preservato e riutilizzato in caso di riesecuzione dei test**.

  L'attività di test di regressione di un programma dovrà essere rigorosa come il test originario; per questo motivo è bene conservare i casi di test realizzati; se registrati in forma elettronica e rieseguiti in maniera automatica, ancora meglio.

- **La pianificazione dell'impegno per il test dovrà tenere conto anche degli errori rilevati nel corso del test**.

  Il responsabile di progetto deve sempre assumere che il test abbia come obiettivo quello di dimostrare la correttezza dei programmi sviluppati; quindi occorre prevedere l'impegno necessario per la rimozione degli errori trovati.

- **In un software, la probabilità di trovare errori è proporzionale al numero di errori già trovati**.

  Il fenomeno è illustrato nella **Figura 1**. 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.

  Nel paragrafo "Metodi, tecniche e strumenti di test" è descritto il modello di "Curva di saturazione degli errori" che rappresenta una tecnica efficace per individuare il numero di errori residui nel software ad ogni momento delle attività di testing.

- **L'attività di test è estremamente creativa**.

  Lo sviluppo del software è un'attività tecnica ed intellettuale ad alto contenuto umano. La creatività e la tecnica sono costantemente integrate per trovare soluzioni efficaci ed innovative. L'integrazione è spesso anche molto spinta. La creatività è perciò fortemente richiesta anche nella progettazione dei casi di test per assicurare che il software sia validato in tutte le sue componenti, caratteristiche e peculiarità, modalità e condizioni di utilizzo.

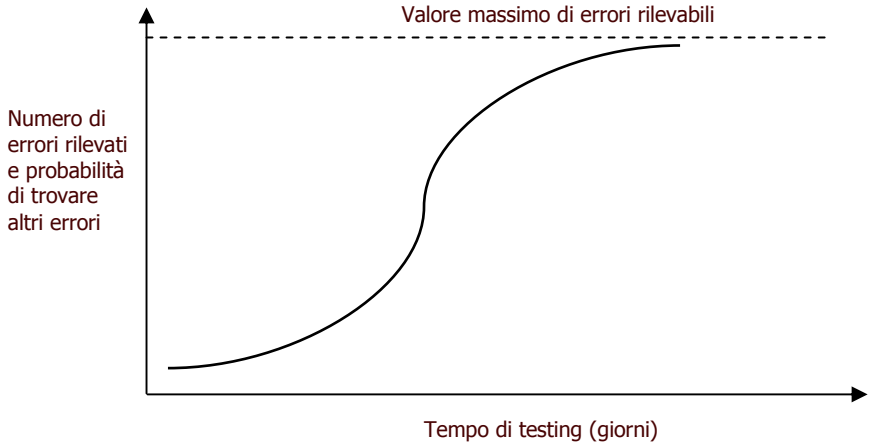

Figura 1. Relazione tra gli errori trovati e previsti.

Riassumendo, possiamo concludere che esistono tre principi importanti nel test:

1. Il test è un processo di esecuzione (vera o simulata) di un programma con l'obiettivo di scoprire gli errori presenti.
2. Un buon caso di test è progettato per avere la più alta probabilità di scoprire gli errori presenti, anche quelli che risultano particolarmente nascosti finché il software non è eseguito.
3. Un caso di test ha successo quando rileva almeno gli errori che ci si aspetta di trovare.

2.2 L'organizzazione di test

2.2.1 Composizione del gruppo di lavoro

Un momento chiave nella gestione dei test è la fase di composizione del gruppo di lavoro. L'attività di testing è molto diversa da quella dello sviluppo del software e richiede competenze ed attitudini completamente diverse[6]. 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, in quanto 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, specialmente tra alti dirigenti, secondo la quale "il test è

[6] G. J. Myers spiega molto bene tali peculiarità nel suo libro, The Art of Software Testing, John Wiley and Sons, New York (1979). Nonostante l'età, il libro rimane ancora una pietra miliare del test.

solo una spesa"; "essa non produce nulla"; "spende il proprio tempo solo a trovare errori senza mai costruire nulla"; "se i costi fossero devoluti ai gruppi di sviluppo si produrrebbe di più e con meno problemi"[7].

E' importante definire un responsabile dei test (Test Manager) con i seguenti compiti e responsabilità:

- Personalizzare la metodologia di test alle particolari necessità del progetto;
- Definire le responsabilità ed i compiti all'interno del gruppo di lavoro;
- Individuare le persone più adatte a svolgere i compiti stabiliti;
- Pianificare, coordinare, gestire e rapportare sull'esito dei test.

Il gruppo di test, a sua volta, sarà responsabile di:

- Produrre il materiale richiesto per il test (casi di test, matrice di test, dati di test, procedure di test);
- Eseguire i casi di test, registrare i difetti, validare le correzioni degli errori, rapportare sui risultati.

Le persone coinvolte nelle attività di test possono provenire dai seguenti reparti o organizzazioni:

- Sviluppo
- Utenti
- Gestione operativa
- Specialisti di test o gruppi indipendenti (es. sicurezza, performance, usabilità, ecc.).

Per una gestione e controllo efficace dei test è necessario definire, specialmente nei progetti di dimensioni medio-grandi, i coordinatori dei test (Test Team Leader) ed i singoli tester assegnando a ciascuno di essi le responsabilità definite. Per ciascun livello di test (test unitario, test d'integrazione, test di sistema, test d'accettazione) è bene chiarire quanto segue senza lasciare alcuna ambiguità:

- Chi crea i dati per i test?
- Chi esegue i test e inserisce i dati?

[7] 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;
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!

- Chi controlla i risultati?
- Chi prende le decisioni su quali correzioni eseguire?
- Chi effettua le correzioni?
- Chi predispone e mantiene gli ambienti di test?
- Chi opera sulle procedure di test?

2.2.2 Ruoli, responsabilità e competenze

Il gruppo di collaudo diventa efficace ed efficiente quando per esso siano state definiti: il ruolo del tester all'interno del progetto di sviluppo; le responsabilità associate al ruolo; le competenze necessarie per svolgere il ruolo assegnato.

Nell'ambito del gruppo di test sono identificati i seguenti ruoli:

- Responsabile del testing
- Esperto tecnico di testing
- Tester

Di seguito sono descritti i ruoli identificati e le relative responsabilità e competenze.

Responsabile del testing.

Per progetti molto grandi e complessi il ruolo è ricoperto da un manager (Test Manager). In progetti meno complessi e di dimensioni minori il ruolo è ricoperto da un coordinatore (Test Team Leader).

Il responsabile dei test coordina tutte le attività di testing (pianificazione, progettazione e preparazione, esecuzione, monitoraggio e reporting) all'interno del progetto.

E' responsabile di completare le attività di test pianificate secondo i tempi ed i costi previsti e raggiungendo gli obiettivi di qualità attesi. Altre responsabilità legate al ruolo sono l'emissione e l'approvazione dei piani di test (Piano di test generale e Piani di test specifici), la costituzione del gruppo di test, la gestione delle persone del gruppo di test e le relazioni con gli altri responsabili dei gruppi di lavoro del progetto. E' anche responsabile di comunicare ai membri del progetto lo stato di esecuzione dei test e lo stato di risoluzione dei difetti. E' inoltre responsabile di fornire il feedback necessario al miglioramento del processo di testing.

Le competenze principali richieste per ricoprire il ruolo sono:

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

Specialista di testing

Il ruolo è generalmente ricoperto da un analista esperto con grande esperienza nelle metodologie di collaudo. Ha molteplici responsabilità legate ai diversi aspetti del testing.

In fase di pianificazione, ha la responsabilità di definire la strategia di collaudo, definire le linee guida per lo svolgimento dei test, definire i criteri di completamento dei test ed identificare gli ambienti e gli strumenti necessari (test tool).

In fase di progettazione e realizzazione dei test, effettua l'analisi dei requisiti per identificare i criteri di testabilità degli stessi. In particolare, analizza i requisiti qualitativi per identificare le caratteristiche del prodotto da sottoporre a test delle prestazioni del prodotto (usabilità, performance, affidabilità, sicurezza, ecc.). Progetta gli scenari di business entro cui calare i casi di test in modo da indirizzare tutti gli aspetti critici dell'applicativo. Progetta i casi di test e partecipa alla loro realizzazione. Assicura la piena copertura dei requisiti, delle funzionalità, delle caratteristiche prestazionali e la gestione delle condizioni speciali e di errore da parte dei casi di test progettati. Supporta l'esperto di ambienti (a meno che non lo sia egli stesso) nella progettazione e realizzazione degli ambienti di test.

In fase di esecuzione, collabora nell'esecuzione dei singoli casi di test e nella valutazione dei risultati ottenuti rispetto a quelli attesi. Assicura la registrazione dei difetti riscontrati e la notifica al gruppo di sviluppo. Verifica la validazione delle modifiche apportate al software a fronte della correzione degli errori. I particolare, conduce i test più critici come quelli relative alle performance, all'affidabilità, all'installabilità, ecc.

Raccoglie e analizza i dati relativi ai test effettuati e supporta il responsabile del collaudo nella produzione dei rapporti periodici sullo stato dei test e degli errori.

Lo specialista esperto di ambienti realizza gli ambienti di test e collaudo secondo le specifiche di progettazione con il supporto tecnico dei sistemisti esperti nei singoli prodotti utilizzati.

Le competenze richieste dal ruolo sono molteplici:

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

Tester

Conosce la metodologia di testing ed applica il processo di testing definito in azienda. Effettua l'integrazione dei componenti di un applicativo nell'ambiente di test, sviluppa i casi di test sulla base della progettazione fatta dallo specialista, esegue i casi di test secondo il piano della attività, valuta i risultati ottenuti a fronte di quelli presiti, gestisce i difetti secondo la procedura stabilità. Opera nell'ambiente di test con i componenti hardware e software ivi disponibili. Operare sulle basi di dati. Utilizza i tool per l'automazione dei test e scrive i relativi script necessari. Scrive eventuali procedure di testing secondo la progettazione stabilità.

Possiede le seguenti competenze:

- Creazione di casi di test e di matrici di test
- Esecuzione dei casi di test nell'ambiente di 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.

2.2.3 Formazione del personale

Un buon tester non s'improvvisa dall'oggi al domani. Un buon 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 manuale si offre indegnamente a tale scopo).

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.

2.3 Testabilità del software

E' importante quanto difficile definire cosa si intenda per "testabilità" del software.

Per valutare se un determinato risultato raggiunge o meno il risultato atteso è necessario definire prima tale risultato in termini di testabilità, cioè capace di essere testato. Quindi, le caratteristiche rilevate a seguito dell'esecuzione di un test devono poter essere paragonate a caratteristiche definite a priori. Tali caratteristiche devono perciò poter essere paragonate senza alcuna ambiguità. L'esecuzione di una funzione di calcolo, per esempio, dovrà fornire un risultato numerico definito a priori: un numero ritenuto "corretto", mentre tutti gli altri risultati saranno considerati "errati".

Poiché il software implementa dei requisiti, occorre dunque che questi siano descritti in termini "testabili" perché i test possano verificarli e validarli. Purtroppo non tutti i requisiti sono definiti in maniera testabile. E' compito della fase di "analisi dei requisiti" raccogliere richieste generiche, elaborale e tradurle in specifiche chiare, complete e "testabili". Senza questa fase cruciale dell'analisi dei requisiti risulta impossibile assicurare l'efficacia dei test.

Concludendo, occorre garantire che i requisiti risultino[8]:

- Chiari (non ambigui)
- Completi
- Testabili.

Garantendo tali caratteristiche dei requisiti sarà possibile definire i risultati attesi dai test e quindi assicurare la "testabilità" del software.

[8] Nella letteratura specializzata, le caratteristiche di testabilità dei requisiti sono indicate con l'acronimo S.M.A.R.T., con il seguente significato:
S = Specifico;
M = Misurabile;
A = Attendibile;
R = Realistico;
T = Tempistico.

2.4 Adozione della metodologia in azienda

La metodologia di testing proposta può aiutare le aziende a sviluppare prodotti software di maggiore qualità e a migliorare la soddisfazione dei propri clienti.

Essa si presta ad essere utilizzata in progetti complessi ed in altri più semplici e di breve durata. Fornisce una descrizione dei tre elementi principali: la competenza delle persone, il processo di testing, i metodi, le tecniche e gli strumenti a supporto. Fa uso delle migliori pratiche disponibili ed è stata già sperimentata con successo presso un gruppo nutrito di piccole e medie aziende di software.

La sua applicazione in una struttura organizzativa con proprie abitudini e metodi consolidati non risulta agevole né immediata. Occorre una forte motivazione al cambiamento e la necessità di migliorare per competere sul mercato.

L'approccio suggerito è tratto dal modello CMM che indica le pratiche richieste per la corretta implementazione dei processi di sviluppo in generale, e di verifica e validazione in particolare, per il miglioramento del livello di maturità di un'organizzazione software.

2.4.1 Impegno della direzione

Il modello proposto acquista validità e diventa efficace quando sostenuto da un impegno della direzione aziendale. Tale impegno sarà quindi formale e sostanziale:

- **formale** in quanto dichiarato, documentato nella politica per la qualità e divulgato in tutta l'azienda;
- **sostanziale** in quanto tale politica sarà efficacemente implementata in azienda con il coinvolgimento della direzione stessa in fase di lancio ed in fase di sostegno e di controllo.

A tale scopo si suggerisce, come prima cosa, di modificare la politica per la qualità in modo da contenere i seguenti punti fondamentali, nel caso non li prevedesse già:

1. Competenza delle persone;
2. Gestione dei requisiti;
3. Disegno e codifica secondo un processo definito;
4. Ispezione e revisione tecnica dei output prodotti;
5. Test e collaudo accurato;
6. Utilizzo di tecniche e strumenti adeguati.

2.4.2 Applicazione della metodologia

Di seguito sono forniti dettagli e suggerimenti su come implementare tale politica.

1. **Assegnare le attività di sviluppo e test del software a personale qualificato con le competenze richieste dai vari ruoli.**

   In particolare, si suggerisce di:

   - Definire in azienda i ruoli, le responsabilità e le competenze necessarie. Si può utilizzare la definizione dei ruoli proposta nella metodologia adattandola alla realtà aziendale.
   - Assegnare a ciascuna persona il proprio ruolo. Ad una persona possono essere assegnati anche più di un ruolo (esempio: una persona tecnica può svolgere i ruoli di analista, programmatore e responsabile dei test, oppure capo progetto e analista, ecc.).
   - Pianificare la formazione necessaria per colmare eventuali lacune nelle competenze richieste.

2. **Gestire accuratamente i requisiti del committente per garantirne la completezza, la chiarezza ed il continuo aggiornamento. I requisiti, inoltre, sono usati come base principale per la progettazione del software e dei test, per la stima dei costi e dei tempi del progetto.**

   In particolare, si suggerisce di:

   - Assegnare la gestione dei requisiti del committente alla persona che in azienda ha il ruolo e le competenze giuste.
   - Assicurare che i requisiti del committente siano documentati, discussi e concordati con il committente.
   - Assicurare che le modifiche ai requisiti iniziali siano gestiti con coerenza (aggiornamento dei documenti, del disegno, dei test, delle stime dei costi e dei tempi).
   - Assicurare che i requisiti siano utilizzati come base sia per la progettazione che per la stima dei tempi e dei costi di progetto, incluse le attività di test.
   - Utilizzare, in fase di stima, la regola d'oro[9] che garantisce un giusto equilibrio tra le attività assegnando il 50% alle attività di analisi e disegno ed il restante 50% alle attività di codifica, controllo, test e collaudo: "Fatto 100 lo

[9] Si tratta della stessa regola menzionata in altra parte del manuale ed enunciata da Frederick P. Brooks nel suo famoso libro "*The Mithical Man-Month: Essays on Software Engineering – 20th Anniversary Edition, 1995,* Addison-Wesley, Reading(MA).

sforzo richiesto per lo sviluppo di un intero progetto, una giusta ripartizione di massima è: Analisi (25%), Disegno (25%), Codifica (10%), Ispezioni, Test e Collaudo (40%) ".

3. **Sviluppare il software secondo un ciclo di vita appropriato al tipo di progetto in corso con definizione delle fasi da seguire, delle attività da svolgere e dei risultati (output) da produrre. In particolare, arricchire l'attuale ciclo di vita con il processo di test descritto in questo manuale.**

   I particolare, si suggerisce di:

   - Assicurare che nel proprio ciclo di vita siano ben definite le attività di testing in modo da indirizzare gli elementi di base della metodologia proposta.
   - Mettere a disposizione delle persone impegnate nel progetto lo schema base personalizzato perché siano a conoscenza delle attività da svolgere.
   - Assicurare che le persone coinvolte nel progetto svolgano le attività previste e producano gli output richiesti utilizzando i modelli stabiliti.

4. **Verificare i prodotti realizzati (output), tramite revisioni tecniche che ne controllino il contenuto e la forma, e verifichino il soddisfacimento dei requisiti del committente.**

   Si consiglia di:

   - Pianificare le revisioni tecniche per tutti gli output più importanti (requisiti, disegno, codice, casi di test, manuali) per un totale pari almeno al 5% delle stime del progetto (utilizzare lo schema base fornito per il piano di qualità, il piano di progetto, il piano di test).
   - Assicurare che le revisioni tecniche siano eseguite; in particolare quelle dei documenti di analisi e disegno: il beneficio è immediato; si riduce la propagazione degli errori nel codice!
   - Produrre i rapporti delle revisioni tecniche effettuate utilizzando lo schema fornito.

5. **Pianificare accuratamente le attività di test e collaudo per assicurare che il software sviluppato indirizzi correttamente tutti i requisiti del committente e risulti privo di errori.**

   Si consiglia di:

   - Pianificare le attività di test necessarie per assicurare il livello di qualità atteso dal committente (utilizzare il modello fornito per il piano di test).

- Assegnare alle attività di test almeno il 30% del tempo totale di progetto; un valore più basso creerebbe impatti sulla qualità del prodotto rilasciato al committente.
- Assicurare che i casi di test indirizzino i requisiti del committente e verificare con la matrice di test il livello di copertura dei test e la possibilità di ottimizzare i test.
- Assicurare che l'ambiente di test sia adeguato (il più possibile simile all'ambiente del cliente ed, in particolare, che la base dati sia realistica e coerente).
- Controllare periodicamente lo stato di avanzamento dei test ed al termine redigere il rapporto di completamento dei test (utilizzare lo schema fornito).

6. Attivare una funzione di Assicurazione della Qualità (QA).

Si consiglia di:

- Assegnare formalmente la responsabilità della pianificazione della qualità di ciascun progetto direttamente al responsabile (capo progetto).
- Assegnare formalmente la responsabilità di sviluppare la qualità di ciascun progetto al proprio responsabile (capo progetto).
- Verificare in prima persona (la direzione), prima del rilascio, che ciascun progetto abbia raggiunto il livello di qualità pianificato.
- Discutere periodicamente, con tutti i capi progetto, il livello di maturità raggiunto dall'azienda se:
  - sono stati definiti obiettivi misurabili per ciascun progetto (vedi "Profilo di qualità");
  - gli obiettivi misurabili di qualità definiti per ciascun progetto sono stati raggiunti;
  - il processo di sviluppo e di test utilizzato in azienda è adeguato alle esigenze del business;
  - i documenti richiesti sono stati prodotti con sufficiente cura e completezza;
  - sono regolarmente tenute revisioni e ispezioni tecniche dei documenti più importanti;
  - sono pianificati test adeguati alla complessità e criticità dell'applicazione sviluppata;

- i progetti sono gestiti accuratamente (stime accurate e realistiche dei tempi e dei costi in fase di pianificazione, controllo dei consuntivi, gestione dei rischi).

- Discutere e pianificare azioni di miglioramento, in caso di insoddisfazione del modo attuale di operare, o in ottica di miglioramento continuo.
- Prendere nota (si utilizzi uno schema predisposto allo scopo) della valutazione fatta e delle azioni intraprese; valutare i risultati delle azioni intraprese.
- Utilizzare una lista di controllo per valutare il grado di miglioramento raggiunto (esempio: checklist di autovalutazione secondo il modello CMM fornita in allegato).

7. **Utilizzare metodi, tecniche e strumenti adeguati a supportare le attività svolte.**

   Si consiglia di:

   - Fornire una copia del presente manuale ad ogni persona coinvolta nelle attività di test del progetto.
   - Incoraggiare le persone a leggere il manuale ed a fare propri i concetti espressi.
   - Capire perché (se così fosse) la metodologia non sia letta, fatta propria ed applicata, ogni qualvolta si abbia l'evidenza, e suggerire azioni opportune per superare gli ostacoli.
   - Concordare con le persone interessate metodi, tecniche, metriche e strumenti più adatti ad essere utilizzati nei progetti realizzati in azienda. Quindi concordare la pianificazione del loro utilizzo a partire dal progetto designato.
   - Valutare i risultati dell'utilizzo dei metodi, tecniche, metriche e strumenti e decidere azioni opportune a fronte di eventuali situazioni non soddisfacenti.

3 Modello di processo di testing

3.1 Il modello proposto

La fase di test, nel ciclo di sviluppo, riveste un'importanza rilevante ai fini della correttezza del prodotto finale, consentendo di valutare il livello di qualità raggiunto, evidenziando gli errori e permettendone la loro correzione.

La fase di test ha il compito di verificare l'aderenza ai requisiti funzionali e prestazionali e la correttezza della progettazione e della codifica. I requisiti prestazionali, detti anche requisiti qualitativi, sono definiti nelle specifiche tecniche e sono indirizzati nel piano della qualità.

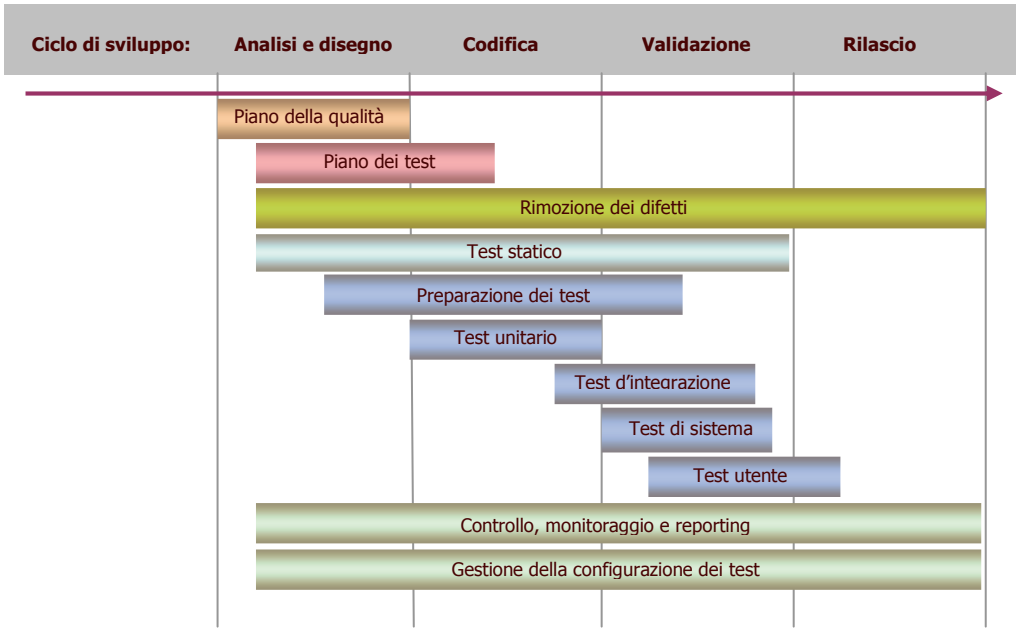

Figura 2. Modello del processo di test del software.

3.2 Descrizione del modello

La **Figura 2** mostra le macro attività previste dal processo di testing e descritte brevemente qui di seguito.

Piano della qualità

Rappresenta il documento principale che descrive gli elementi di qualità richiesti per il prodotto. La qualità raggiunta dal prodotto realizzato è verificata e validata tramite le attività di testing. Il piano indica gli obiettivi da raggiungere, le metriche di valutazione da adottare, i valori soglia, il processo di sviluppo da seguire e l'approccio al testing per la rimozione dei difetti.

Piano di test

Il piano di test è prodotto durante la fase di pianificazione del progetto. La pianificazione dei test consiste nell'analizzare i requisiti funzionali e prestazionali, identificare le caratteristiche del prodotto, stabilire la strategia di test (quali e quanti test eseguire), pianificare le attività ed individuare le risorse di test necessarie. L'attività di pianificazione è formalizzata nel Piano di test. Le caratteristiche del prodotto da verificare (usabilità, prestazioni, affidabilità, ecc.) sono documentate nel Piano della qualità insieme alle metriche da adottare ed ai valori di soglia da rispettare. La strategia di test consiste nello stabilire, partendo dalle caratteristiche del software, i tipi di test da eseguire, i livelli di profondità cui giungere, gli approcci all'integrazione da seguire, gli ambienti da predisporre. Per progetti complessi o di grandi dimensioni si realizza un Piano di test generale (Master Test Plan) e Piani di test di dettaglio. La relazione che intercorre tra i diversi piani è mostrata nella **Figura 3**.

Rimozione dei difetti

Rimuovere gli errori presenti nel software rappresenta uno dei due obiettivi dell'attività di testing (il secondo obiettivo è quello di "verificare l'aderenza del prodotto ai requisiti specificati"). La rimozione dei difetti è praticata durante l'intero ciclo di sviluppo: prima con le attività di revisione dei documenti tecnici (requisiti, specifiche, disegno, casi di test, documenti utente, ecc.), poi con le ispezioni del codice (test statico) ed infine con le attività di collaudo vero e proprio (test dinamico). Registrare i difetti riscontrati e la relativa correzione è considerata una "best practice". L'utilizzo di uno strumento software facilità la gestione del processo e ne migliora l'efficienza.

La **Figura 3** mostra il legame che intercorre tra il Piano di test generale (Master Test Plan) ed i singoli test di dettaglio relativi ai diversi livelli di testing.

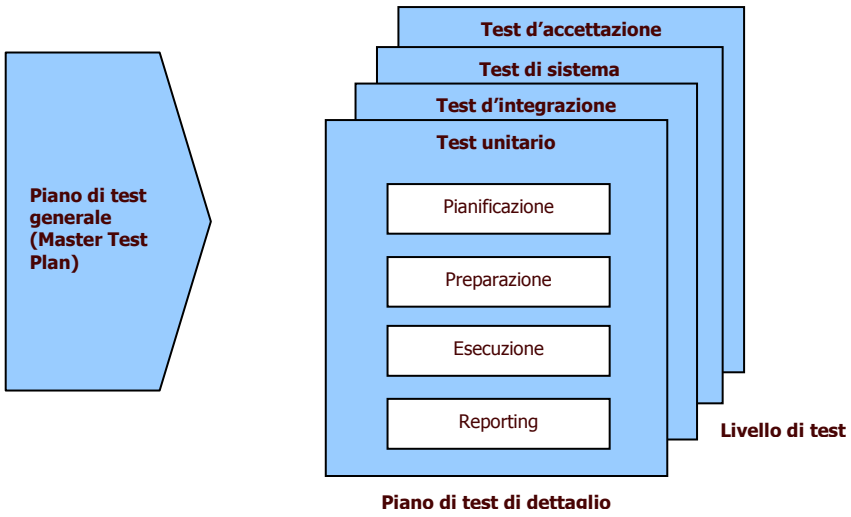

Figura 3. Collegamento tra piani di test (generale e di dettaglio).

Test statico

Nelle fasi alte del ciclo di sviluppo, Analisi e progettazione, la rimozione dei difetti è condotta tramite le attività di "Revisione tecnica" dei documenti con lo scopo di individuare e correggere gli errori presenti ed evitare la loro propagazione alla successiva fase di codifica. Tali attività sono dette "Test statico" in quanto non si esercita il codice (che ancora non esiste) ma si verifica la completezza e la correttezza della documentazione tecnica da cui si partirà per sviluppare il codice. Anche le "Ispezioni del codice" sono attività di test statico e permettono di rimuovere gli errori di codifica.

Preparazione dei test

L'attività include la progettazione dei casi di test e la predisposizione degli ambienti necessari.

La progettazione dei casi di test è fatta in base alle caratteristiche specificate del software (funzionalità, usabilità, performance, affidabilità, installabilità, ecc.). La progettazione riguarda i casi di test, la matrice di test, le procedure per l'esecuzione dei test, le basi dati ed i programmi e moduli "fantasma" (scaffolding: stub e driver). Prima della loro esecuzione i casi di test sono ispezionati per verificarne la loro completezza e correttezza.

Gli ambienti di test sono progettati e predisposti in base alle caratteristiche tecniche richieste. E' necessario che gli ambienti di test siano collaudati prima di iniziare l'esecuzione dei test per verificare la loro completezza e correttezza evitando che possano in qualche maniera influire sui risultati dei test eseguiti.

Esecuzione: Test unitario, Test d'integrazione, Test di sistema

I casi di test progettati sono eseguiti secondo la tempificazione e le modalità stabilite nel piano di test. Il collaudo del software è eseguito sui diversi livelli di aggregazione del software: prima a livello unitario (singoli moduli), poi a livello di moduli e componenti integrati, infine sull'intero sistema. L'utente collauderà il prodotto finale secondo i propri requisiti dichiarati. Il team di test installa il software negli ambienti predisposti ed esercita il prodotto eseguendo le procedure di test secondo le sequenze stabilite, registra gli errori rilevati e li notifica al gruppo di sviluppo. Una volta corretti gli errori segnalati, riprende l'esecuzione dei casi di test interrotti. Si tiene traccia dello stato di completamento dei casi di test e dello stato di risoluzione degli errori.

Controllo, monitoraggio e reporting dei test

Il controllo dell'esecuzione dei casi di test è costante ed è effettuato dal responsabile del test (Test Team Leader) che informa del risultato il responsabile del progetto. Eventuali problemi sono indirizzati prontamente per evitare ritardi ed inefficienze. Il monitoraggio dello stato di completamente dei casi di test e dello stato di risoluzione degli errori è anch'esso costante ed effettuato sempre dal responsabile dei test. Periodicamente – a livello settimanale – è prodotto un rapporto con lo stato di completamento dei casi di test e lo stato di risoluzione degli errori rilevati. Il rapporto è distribuito agli interessati (responsabili del progetto, del test, della qualità, direzione) nell'ambito del controllo periodico dello "stato di avanzamento delle attività" (SAL) del progetto.

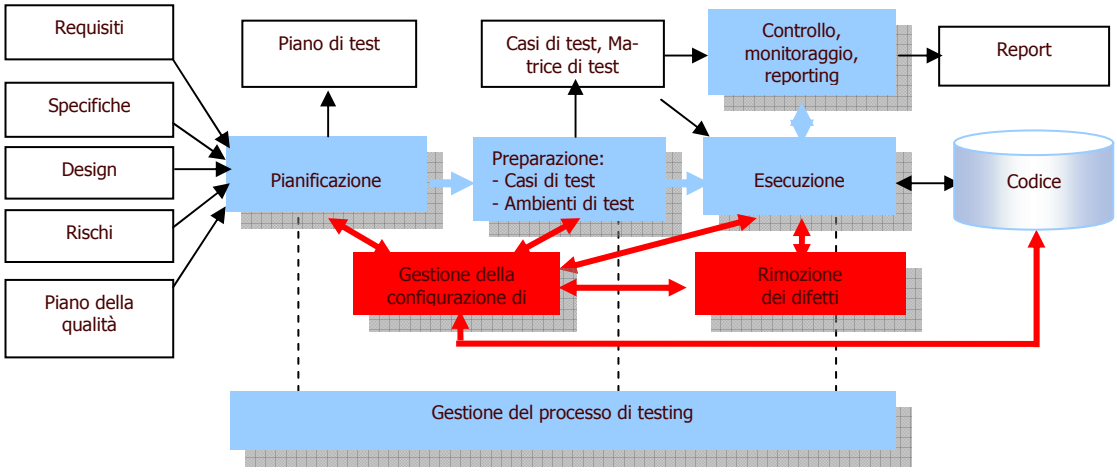

Figura 4. Elementi del processo di test del software.

La **Figura 4** mostra gli elementi di dettaglio che concorrono al processo di testing insieme all'attività di gestione del processo stesso. Sono anche indicati gli input e gli output delle singole fasi.

Le attività relative alla gestione del processo di testing sono descritte a seguire.

Gestione della configurazione di test

Consiste nel controllo della configurazione di tutti gli elementi che compongono il prodotto nella fase di testing: documenti del processo (piano di test, casi di test), ambienti di test, codice, modifiche al prodotto a fronte di richieste esterne o a fronte della rimozione dei difetti.

Il tema è ben noto e riferito come gestione della configurazione del software (*Software Configuration Management* - SCM).

Ad essa è strettamente legata la gestione delle modifiche (*Change Management*). Il test verifica e valida una determinata configurazione di software. Ogni modifica apportata (al software, agli ambienti o ai documenti tecnici) comporta una variazione nella configurazione in collaudo e quindi deve essere gestita opportunamente: modificare, se necessario, i test in corso per garantirne la copertura a fronte delle modifiche apportate.

L'insieme delle due tematiche è nota come Gestione delle modifiche e della configurazione (*Change and Configuration Management* - CCM).

Gestione del processo di test

Consiste nella verifica dell'efficacia del processo di test. L'efficacia è misurata adottando le metriche dei test stabilite: livello di copertura dei test pianificati, tasso di errori rilevati dai casi di test, curva di rimozione degli errori durante i test, produttività media del test, ecc. La verifica è condotta da una funzione esterna al gruppo di sviluppo e di test – per esempio, Assicurazione qualità (Quality Assurance) -. In un'organizzazione di medie o piccole dimensioni l'attività è svolta da una o più persone delegate dalla direzione a ricoprire tale ruolo. La misurazione dell'efficacia del processo di test è una "best practice" e permette di intervenire sul processo per migliorarlo, così come richiesta dallo standard ISO 9000 e dal modello di maturità CMMI.

4 Tipi di test

I test si dividono generalmente in due categorie distinte ma complementari tra di loro: test statici e test dinamici.

Test Statici

Sono le attività di verifica e validazione di tutto ciò che è prodotto durante l'intero ciclo di sviluppo del software (documenti dei requisiti, specifiche funzionali, documenti di disegno, piani, casi di prova, codice, tabelle, ecc.) senza la loro esecuzione nei computer. Generalmente queste attività sono conosciute con diversi termini: "ispezione", "revisione", "walkthrough", "peer review". La differenza sta nel livello di formalizzazione ed è descritta nel seguito. Costituiscono una "best practice" di valore grandissimo in quanto permettono di rimuovere gli errori il prima possibile evitando la loro propagazione e riducendo notevolmente i costi richiesti per la rimozione successiva.

Test Dinamici

Sono le attività comunemente conosciute come "test" e consistono nella verifica e validazione del codice sviluppato eseguendolo nei computer (Test unitario, di integrazione, di sistema, di accettazione). I test dinamici includono sia la verifica delle funzioni sviluppate sia la verifica delle caratteristiche del prodotto come, ad esempio, prestazioni, disponibilità, affidabilità, carico e scalabilità, ecc. Un caso particolare è rappresentato dalle applicazioni di e-business, che richiedono ai test di verificare caratteristiche particolari come, ad esempio, la tecnologia utilizzata, la sicurezza delle operazioni sulla rete, le prestazioni sotto particolari condizioni e periodi di carico, ecc.

4.1 Test statico

4.1.1 Che cosa sono le revisioni tecniche

L'attività è svolta lungo l'intero arco del ciclo di sviluppo (vedi **Figura 2**) e si applica a tutti i documenti prodotti durante lo svolgimento del progetto. I documenti sono revisionati per verificarne la completezza, la correttezza e l'aderenza ai requisiti, standard, regolamentazioni e normative. L'attività di verifica riguarda sia i contenuti che la forma. L'attività è di grande importanza in quanto rappresenta l'unico modo a disposizione per validare il prodotto nelle prime fasi del ciclo di sviluppo, prima cioè che sia realizzato il codice sorgente – è il motivo per cui tali attività sono anche dette "test statico" –. Si evita in questo modo che gli errori si propaghino alle fasi successive con notevole aumento dei costi di rimozione. L'utilizzo metodico delle revisioni tecniche costituisce una "best practice" di enorme valore ed è richiesta in maniera esplicita dal modello di

maturità CMMI (*Verification*), ma anche dalla norma ISO 9001:2000 (Verifica della progettazione e dello sviluppo).

I documenti da revisionare sono principalmente:

- documenti di progetto (piano di progetto, piano della qualità, piano di test);
- documenti di prodotto (requisiti, specifiche funzionalità, disegno, casi d'uso, casi di test, matrice di test, manuali);
- codice sorgente (inteso come documento).

In sintesi, una revisione tecnica verifica che un documento prodotto:

- sia aderente agli standard definiti (sia stato prodotto/redatto usando un modello stabilito);
- sia completo in tutte le parti richieste (così come previsto dallo schema base stabilito);
- sia corretto nei contenuti (es.: il disegno sia tecnicamente corretto);
- indirizzi i requisiti attesi (es.: il disegno indirizzi tutti i requisiti del prodotto stabiliti, espliciti ed impliciti).

La **Figura 5** mostra le revisioni tecniche previste durante l'intero ciclo di vita del progetto.

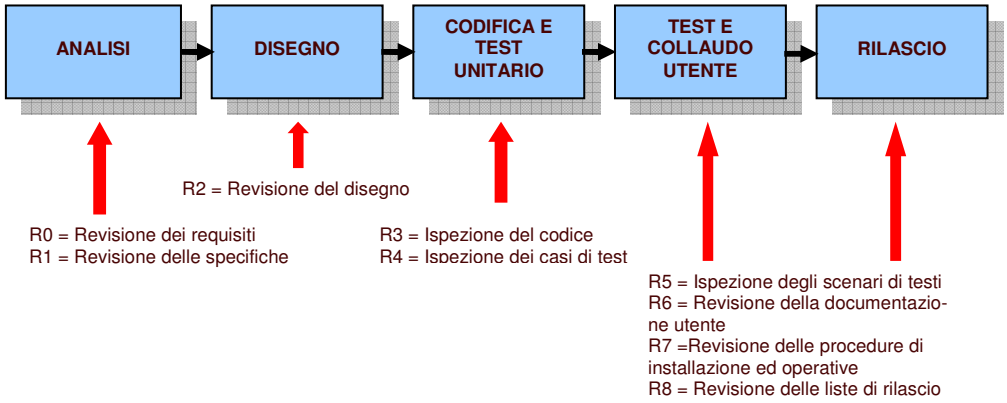

Figura 5. Revisioni tecniche previste nel ciclo di vita del progetto.

L'utilizzo dei termini "ispezione" e "revisione" è solo dettata dalla consuetudine maturata nei laboratori di sviluppo software.

4.1.2 Tipi di revisione tecnica

Le revisioni tecniche possono essere condotte con modalità diverse, a seconda dell'obiettivo: Ispezione o Walkthrough.

Ispezione

Consiste nella verifica di un documento da parte di un gruppo di persone (anche una sola persona) per verificarne, appunto, la completezza e la correttezza. Le persone coinvolte (detti ispettori o revisori) sono sempre persone competenti (colleghi esperti). Generalmente, la distinzione tra i due tipi sta solo nel livello di formalità con cui sono svolte: il primo tipo (ispezione) è eseguito in maniera "formale", mentre il secondo (revisione) in maniera "informale".

Walkthrough

Letteralmente "camminare attraverso", consiste nell'ispezionare documenti tecnici di rilievo (documento di disegno, codice, ecc.) in maniera particolare: i revisori, generalmente molto esperti della materia trattata (analisti o programmatori esperti), ripercorrono logicamente e tecnicamente il contenuto del documento simulando la sua esecuzione da parte del sistema. Per esempio, il disegno proposto è validato tecnicamente simulando il suo funzionamento come se fosse stato già tradotto in codice; oppure, un programma è ripercorso nei suoi rami logici per verificare che esso possa essere eseguito correttamente dal computer, ecc.

L'attività di revisione, come già detto sopra, può essere svolta in maniera formale o informale.

Formale

Le revisioni formali sono pianificate, preparate, eseguite e documentate secondo un formalismo stabilito (vedi "Revisione strutturata"). I risultati delle revisioni formali sono controllati dall'Assicurazione qualità.

Informale

Le revisioni informali sono eseguite da colleghi esperti quando l'autore di un documento lo ritiene necessario. La revisione dei documenti richiesti può essere fatta anche parzialmente, riunendosi direttamente con l'autore del documento, lasciando a questi la discrezionalità di prendere gli appunti necessari e senza produrre rapporti o verbali ufficiali. Non è richiesta la produzione del rapporto di revisione con i dati relativi all'attività svolta.

4.1.3 Elementi principali delle revisioni

Per essere realmente efficace occorre che una revisione tecnica segua alcuni principi di base.

- La revisione di un documento si effettua solo quando esso è completo;
- Se i tempi e le risorse a disposizione non consentono la revisione di tutto il documento, concentrarsi solo sulle parti più critiche: soluzioni tecniche particolari, introduzione di nuove tecnologie per l'azienda, algoritmi particolari, struttura dei dati, casi d'uso, elementi critici per la qualità, ecc.;
- La revisione è fatta sempre da persone competenti;
- La revisione è pianificata per consentire alle persone coinvolte di dedicare il tempo necessario alle attività di revisione (preparazione e riunione di revisione) compatibilmente con i propri impegni ed il paino di progetto (Piano delle revisioni);
- La revisione ha lo scopo di evidenziare i problemi e di scoprire gli errori nella progettazione prima di iniziare la codifica, e non quello di giudicare il lavoro o la persona.

Gli eventuali problemi riscontrati sono discussi e corretti opportunamente.

4.2 Test dinamico

4.2.1 Che cosa sono i test dinamici

I Test dinamici sono le attività di validazione del codice sviluppato tramite la sua esecuzione nei computer. Nella letteratura specialistica e nel linguaggio comune dello sviluppo software, sono chiamati più semplicemente "test". L'attività, quindi, consiste nell'eseguire il software in un ambiente di test fornendo degli input e verificando l'output prodotto.

Alcuni esempi di test dinamici eseguiti nelle diverse fasi del ciclo di sviluppo sono:

- Validazione del disegno tramite l'esecuzione di un prototipo in fase di analisi e disegno;
- Validazione delle funzionalità del prodotto tramite l'esecuzione dei casi di test funzionali in un ambiente di collaudo durante la fase di test d'integrazione;
- Validazione delle capacità di gestione delle condizioni di errore da parte del sistema tramite l'esecuzione di appropriati casi di test durante il test d'integrazione;

- Validazione dell'usabilità del sistema tramite la simulazione di scenari di utilizzo da parte di utenti in un test di usabilità eseguito in fase di test di sistema;
- Validazione delle prestazioni del sistema tramite l'utilizzo di un simulatore di più utenti contemporanei durante la fase di test di sistema;
- Validazione delle prestazioni del sistema tramite l'utilizzo di un simulatore del carico di lavoro durante l'esecuzione del test di sistema;
- Validazione della facilità e correttezza dell'installazione del prodotto negli ambienti previsti eseguito tramite un test di installabilità durante il test di sistema.

I test dinamici si caratterizzano per la loro capacità di rilevare gli errori del software a seconda del livello di aggregazione dei moduli e di integrazione del codice disponibile. Sono perciò individuati, nella letteratura specializzata, i seguenti "livelli di test" descritti in dettaglio nel capitolo successivo: test unitari, test d'integrazione, test di sistema, collaudo di accettazione.

4.2.2 Tipi di test dinamici

I test dinamici sono di tipo di verso a seconda del livello di profondità cui arrivano: singolo modulo (test unitario), integrazioni di più moduli in componenti (test d'integrazione), intero prodotto (test di sistema).

Test unitario

Si tratta dell'attività di verifica e validazione dei singoli moduli (oggetti, programmi, tabelle, ecc.) eseguiti dagli sviluppatori nei propri ambienti di sviluppo.

Test d'integrazione

Si tratta dell'attività di verifica e validazione dei componenti integrati (insieme di moduli integrati secondo l'architettura applicativa disegnata e la strategia di integrazione prevista) in un apposito ambiente di test.

Test di sistema

Si tratta dell'attività di verifica e validazione dell'intera applicazione. Il test di sistema è anche chiamato "Collaudo interno" ed eseguito in un ambiente di test simile, quando non sia possibile proprio quello di esercizio.

Test di accettazione

Si tratta del collaudo di accettazione eseguito dal committente per verificare la rispondenza del prodotto finale alle proprie necessità di business. Il test è anche detto "Collaudo utente" in quanto eseguito principalmente dagli utenti finali del prodotto. Il collaudo è eseguito in un ambiente perfettamente equivalente a quello di produzione.

Essi sono meglio descritti nel successivo "Capitolo 5. Livelli di test".

La tipologia di test è anche dipendente dalle caratteristiche del software verificate: funzionalità (test funzionale), prestazioni (test di usabilità, test di efficienza, test di robustezza, test di sicurezza, test di disponibilità e di affidabilità), operabilità (test di installazione, test di migrazione, test di parallelo, test di operabilità, test di manutenibilità, ecc.).

Essi sono descritti in dettaglio nel successivo "Capitolo 6. Tipologie di test".

Un caso particolare riveste il test delle applicazioni per il commercio elettronico o con forte interazioni con la Rete descritto nel successivo "Capitolo 7. Test delle applicazioni e-business".

5 Livelli di test

Rappresentano il livello di profondità cui arrivano i diversi test. I vari livelli sono:

- Test unitario (*Unit Test*)
- Test d'integrazione (*Integration Test*)
- Test di sistema (*System Test*)
- Test di accettazione (*Acceptance Test*)

Il livello di profondità è quindi rappresentato dal livello di integrazione del prodotto. La **Figura 6** riporta lo schema dei livelli di test in base al livello di aggregazione del software e la documentazione tecnica di progetto cui si riferiscono.

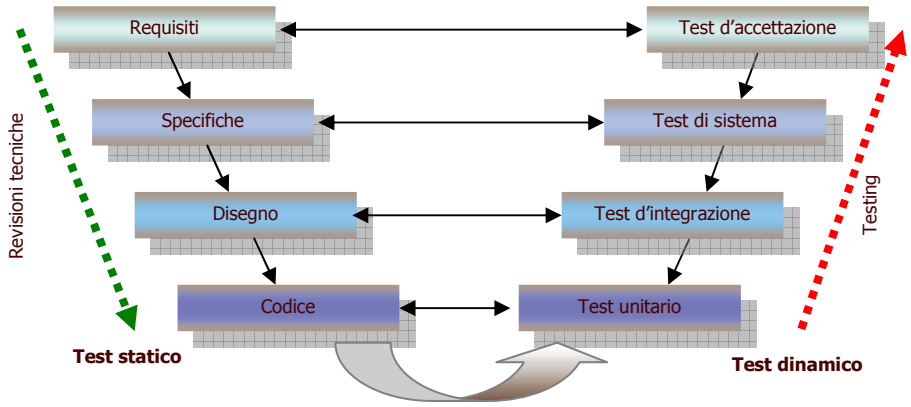

Figura 6. Schema dei livelli di testing ("V" Model).

Durante la costruzione del prodotto (lato sinistro dello schema rappresentato nella figura) si effettuano i test statici (revisioni tecniche) sui singoli prodotti di fase realizzati (requisiti, specifiche, disegno, codice). Le revisioni tecniche rappresentano le modalità operative per la conduzione delle verifiche di tali elementi. Una volta sviluppato il codice, si procede in senso contrario, dal basso verso l'alto (lato destro della figura). Man mano che il codice è integrato fino a completare il prodotto, si effettuano collaudi sempre più completi. Si parte quindi dal test unitario dei singoli moduli, si continua con il test d'integrazione, si completa il collaudo dell'intero prodotto con il test di sistema e si termina con il collaudo di accettazione da parte degli utenti.

Lo schema rappresentato mostra anche la corrispondenza del livello di test con il contenuto del prodotto: i requisiti sono verificati dall'utente tramite il collaudo di accetta-

zione, le specifiche sono verificate con il test di sistema, il disegno è validato con il test d'integrazione ed il codice con il test unitario.

Entriamo ora nel dettaglio dei singoli test.

5.1 Test unitario

Quello unitario (*Unit Test*) è il primo test dinamico del codice eseguibile, sia nella sua versione iniziale che in ogni sua modifica successiva. Esso verifica l'aderenza alle specifiche di programmazione dal punto di vista logico del programma; valida quindi la sua logica interna.

I programmi ed i moduli sono ispezionati manualmente prima di essere integrati per il test. La revisione del codice è fatta dai programmatori stessi tramite revisioni tecniche o attività di walkthrough (test statico).

I passi principali da compiere nel preparare un test unitario efficace sono:

- Determinare l'approccio all'integrazione e al testing (top-down, bottom-up o combinazione di entrambi);
- Determinare la tecnica di test da utilizzare (in questo caso "white-box") e le particolari sotto-tecniche che meglio si applicano al livello di testing (copertura delle linee di codice, copertura delle condizioni decisionali - if, select, ecc. -, copertura dei cammini, analisi delle condizioni di errore, ecc.);
- Sviluppare tutte le condizioni di test decisive con i casi di prova.

Tra i controlli da realizzare durante l'ispezione del codice ricordiamo:

- Tutte le variabili sono esplicitamente dichiarate ed inizializzate;
- L'inizializzazione è correttamente eseguita ad ogni ciclo di esecuzione e le aree acquisite sono opportunamente ripulite o rilasciate;
- I puntatori sono numerici e definiti all'interno dell'ambito (range) di validità;
- Le variabili referenziate sono correttamente "allocate";
- Le condizioni di errore sono gestite correttamente;
- Gli attributi dei file sono corretti;
- Le condizioni di "fine file" sono gestite correttamente.

La progettazione dei casi di test, eseguita contestualmente allo sviluppo del codice, costituisce una pratica consolidata con evidenti vantaggi in atto di copertura dei test ed individuazione delle condizioni da testare (sequenze, algoritmi, decisioni, errori, ecc.).

La documentazione relativa al test unitario include:

- Piano di test unitario;
- Casi di test;
- Rapporto finale di completamento del test unitario con dettagli sul livello di copertura dei test e sullo stato di risoluzione degli errori rilevati.

In sintesi, la qualità (efficacia) del test unitario è direttamente legata al livello di copertura dei test eseguiti e garantita dalle misurazioni effettuate tramite tool appositi.

5.2 Test d'integrazione

Il test d'integrazione (*Integration Test*) verifica la corretta esecuzione dei componenti (sottosistemi) dell'applicazione man mano che questi sono completati e resi disponbili. I moduli del sottosistema collaudato sono verificati, in modo integrato, nell'ambiente di test preparato, isolato e controllato. Le comunicazioni con altri sottosistemi non ancora pronti sono simulati utilizzando programmi "fantasma" detti scaffolding (driver, stub).

L'approccio all'integrazione del prodotto è quello stabilito nella strategia di sviluppo: Top-Down, Bottom-Up o combinazione di entrambe le tecniche.

5.2.1 Integrazione "Top-Down"

L'approccio "Top Down" è una strategia che parte con la costruzione dei moduli di livello più alto per poi integrare i moduli di livello inferiore all'interno della struttura. Per poter integrare e testare i moduli di livello più alto occorre costruire moduli fittizi che simulino le interfacce con i moduli di livello più basso. I programmi sono detti "stub" o "dummy".

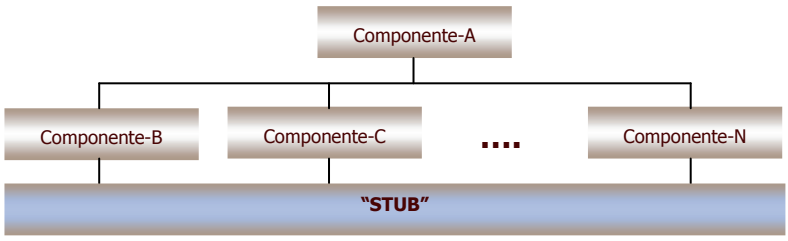

Figura 7. Schema di integrazione Top-Down con utilizzo di "stub".

Lo scopo di un programma "**stub**" o "dummy" è quello di simulare l'esistenza di un modulo o programma non ancora disponibile finché quello vero non sia pronto a sostituirlo. Questo tipo di utilizzo temporaneo di un codice rappresenta un esempio di "scaffolding". Ogni modulo integrato nel programma ha bisogno di uno stub che simuli i moduli chiamati, finché questi non saranno integrati nel programma. Il test può così

essere eseguito sull'intero sistema che include moduli veri, quelli di più alto livello, e moduli simulati, quelli di livello più basso. Il test prosegue sostituendo i moduli simulati con quelli veri man mano che questi ultimi sono disponibili.

Nella **Figura 7** è mostrato uno schema in cui è presente un modulo o programma "stub" per consentire l'integrazione di tipo Top-Down dei componenti disponibili da collaudare (nella figura i componenti A, B, C, …. N).

5.2.2 Integrazione "Bottom-Up"

Nell'approccio "Bottom Up" i moduli o programmi sono scritti e testati partendo da quelli di livello più basso. Successivamente, si aggiungono e si testano i moduli o programmi di livello superiore finché il sistema non è completo. Per eseguire questi test occorre sviluppare moduli o programmi in grado di simulare i componenti di livello superiore (driver) che richiamino le funzioni eseguite dai moduli da testare (quelli di livello inferiore).

Un "**driver**" è un programma che simula uno non ancora disponibile e che effettua le chiamate ai moduli e programmi sottostanti da testare. Per esempio, un driver può generare i dati necessari ai moduli chiamati per essere eseguiti nei test, oppure leggere un file di dati per i test, elaborali, formattarli e poi passarli ai moduli chiamati. Generalmente, viene creato un "file di log" o un "report di debug" con i dati passati ai moduli chiamati e quelli da essi ricevuti allo scopo di verificare la correttezza dei risultati. Questa tecnica è molto efficace quando il codice dei moduli testati è completo.

Nella **Figura 8** è mostrato uno schema in cui è presente un modulo o programma "driver" per consentire l'integrazione di tipo Bottom-Up dei componenti disponbili da collaudare (nella figura i componenti A, B, … N).

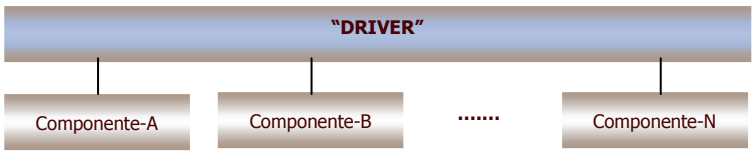

Figura 8. Schema di integrazione Bottom-Up con utilizzo di un "driver".

La documentazione relativa al test d'integrazione comprende:

- Piano di test d'integrazione;
- Casi di test e matrice di test;
- Rapporti periodici sullo stato di completamento del test d'integrazione con dettagli sul numero e lo stato dei casi di test, sul numero e lo stato dei difetti rilevati;
- Rapporto finale di completamento del test d'integrazione.

5.3 Test di sistema

I test di sistema verificano la corretta esecuzione dell'intera applicazione, incluse le interfacce con altre applicazioni. Si eseguono sia tipi di test funzionali che strutturali per verificare che il sistema sia corretto sia dal punto di vista funzionale che operativo. Il test di sistema include perciò i seguenti tipi di test:

5.3.1 Test funzionale (*Functional Test*)

Verifica che tutte le funzioni siano correttamente completate in uno scenario simile a quello degli utenti finali. Le funzioni non sono eseguite singolarmente (esse sono state già verificate durante il test d'integrazione), ma in una sequenza di operazioni che completi un tipico task dell'utente finale. I casi di test sono quindi eseguiti in una sequenza tale da completare uno o più "scenari di test".

5.3.2 Test di usabilità (*Usability Test*)

Verifica la facilità d'uso e la comprensibilità dell'applicativo da parte degli utenti finali. Il test è previsto quando ci siano requisiti relativi a caratteristiche di usabilità del sistema e siano indirizzati nel Piano della qualità. Tali caratteristiche di usabilità risultano in questo caso critiche per l'introduzione del prodotto software presso l'utenza finale. Il test di usabilità richiede competenze specifiche, utilizzo di tecniche proprie ed il coinvolgimento di utenti veri o di persone in grado di sostituirle.

5.3.3 Test delle prestazioni (*Performance Test*)

Verifica le prestazioni del sistema quali tempi di risposta e utilizzo delle risorse (memoria, linee di trasmissione, banche dati, altri componenti). Il test è eseguito quando le caratteristiche relative alle prestazioni costituiscono un fattore critico per il successo del prodotto. In questo caso sono stati definiti requisiti di prestazioni, stabiliti metriche e valori di soglia da rispettare ed il tutto è stato documentato nel Piano di qualità.

5.3.4 Test di affidabilità (*Reliability Test*)

Verifica l'affidabilità del sistema quando essa sia richiesta esplicitamente. L'affidabilità è definita come la capacità del sistema di operare per un intero arco di tempo (per esempio per 8 ore lavorative consecutive, oppure H24/7x7) senza che sia rilevato alcun difetto che ne interrompa l'operatività. In questo caso esistono requisiti specifici definiti sull'affidabilità, sono stabilite metriche e valori di soglia concordati ed il tutto è documentato nel Piano di qualità. Il test è eseguito con opportuni strumenti (tool) che simulano l'utilizzo del prodotto in modo continuativo e misura il tempo intercorso tra due difetti consecutivi. Tale tempo (detto Mean Time Between Failure - MTBF) rappresenta l'affidabilità del sistema.

In caso di caduta del sistema il test verifica la capacità del sistema di recuperare i dati e le transazioni eseguite.

5.3.5 Test di carico (*Stress/Load Test*)

Verifica la capacità del sistema di reggere un determinato carico di lavoro. Il test è eseguito a fronte di requisiti espliciti in materia, da cui sono tratte metriche per misurare le prestazioni, valori di soglia da rispettare. Il tutto è documentato nel Piano di qualità. Il test è eseguito utilizzando opportuni tool che simulano il carico possibile del sistema, misurando le prestazioni e verificando che non ci siano degradi nelle prestazioni o cadute del sistema.

5.3.6 Test di sicurezza (*Security Test*)

La verifica del livello di sicurezza fornito dal sistema è fatta in base a requisiti specifici che indicano tale livello, le modalità operative e le condizioni sottostanti. Il test è eseguito da esperti in grado di simulare attacchi al sistema e verificare la reattività del sistema stesso.

Il test di sistema è eseguito in un ambiente simile (quando non può essere lo stesso) a quello di esercizio in cui il prodotto sarà utilizzato dagli utenti.

La documentazione relativa al test di sistema comprende:

- Piano di test;
- Casi di test e scenari di test;
- Rapporti periodici sullo stato di completamento dei test e dello stato di risoluzione degli errori;
- Rapporti specifici sui risultati dei test di usabilità, performance, affidabilità, ecc.
- Rapporto finale di completamento del test di sistema.

5.4 Test di accettazione

Questo tipo di test è eseguito direttamente dal cliente per verificare la conformità del prodotto finale alle proprie esigenze di business, così come dichiarato nei requisiti. Il collaudo consiste nel verificare che i criteri di accettazione siano soddisfatti dal prodotto. I criteri di accettazione sono, solitamente, definiti formalmente nel contratto. L'esito del collaudo è formalizzato in un apposito documento: "Rapporto sull'esito del collaudo".

All'esito positivo del collaudo è solitamente legato il pagamento da parte del cliente. Il collaudo è effettuato da utenti che meglio rappresentano la tipologia più frequente o significativa dal punto di vista dell'utilizzo del prodotto. Il collaudo d'accettazione può essere progettato ed eseguito direttamente dal cliente (o da una società di consulenza delegata a fare ciò) o dal fornitore stesso su specifica richiesta del committente e poi da questi controllato ed approvato.

La documentazione di collaudo è composta da quattro elementi principali:

- Piano di collaudo;
- Criteri di accettazione;
- Casi di test e scenari di test;
- Verbale dell'esito del collaudo.

Sono quindi definite chiaramente i criteri secondo cui il cliente valuterà se accettare o meno il sistema realizzato. Si definiscono inoltre:

- I criteri per determinare la severità degli errori riscontrata durante il collaudo (solitamente sono definite tre livelli di gravità: errore bloccante del sistema, errore grave ma non bloccante del sistema, errore minore);
- Le modalità di gestione degli errori (stati e ciclo di risoluzione degli errori);
- I tempi di risoluzione degli errori secondo le gravità stabilite;
- I criteri di uscita dal test di accettazione (esempio: tutti i casi di test completati con successo; tutti gli errori di gravità massima risolti; tutti gli errori di gravità media risolti; accordo sul piano di risoluzione successiva degli errori minori).

> ***Nota**: Nel caso in cui il collaudo sia progettato e realizzato dal cliente, il fornitore deve comunque concordare quanto precedentemente elencato e fornire il supporto tecnico necessario ad eseguire il collaudo.*

6 Tipologie di test

I tipi di test definiscono quali aspetti del prodotto si vogliono validare ("cosa" si testa): le funzioni, la struttura, le prestazioni, l'installazione, l'usabilità, la capacità di gestire gli errori, ecc.

Le due tipologie principali di test sono:

- Test di tipo "funzionale" – questa tipologia di test verifica e valida il prodotto dal punto di vista "esterno" e del suo utilizzo e si concentra su caratteristiche esterne tipo: funzionalità, gestione degli errori, utilizzo della documentazione utente, usabilità, installazione, migrazione, regressione, gestione del parallelo, gestione delle interfacce esterne con altri prodotti, ecc.
- Test di tipo "strutturale" – questa tipologia di test tende a verificare e validare le caratteristiche "interne" del prodotto quali, ad esempio: architettura tecnica del prodotto, utilizzo delle tecnologie, funzioni di backup and recovery, sicurezza, performance, stress e volumi, gestione operatività, ecc.

Nello sviluppo di applicazioni per un cliente specifico, mediamente complesse e critiche per il business, sono generalmente definiti e condotti i due tipi di test descritti sopra:

Test funzionali

- Test delle funzionalità;
- Test di gestione delle condizioni d'errore;
- Test di operatività;
- Test d'usabilità.

Test strutturali

- Test di Backup and Restore;
- Test di sicurezza;
- Test di performance;
- Test di carico (stress e volumi).

Nel caso di modifiche sostanziali ad un software già esistente è di particolare importanza la verifica che le caratteristiche precedenti del software non siano state compromesse dalle modifiche. In questo caso è fondamentale l'esecuzione anche del tipo di test:

- Test di regressione.

Nel caso di prodotti programmi rivolti ad un mercato più ampio, ai test precedenti si aggiungono altri specifici di un prodotto programma quali:

- Test d'installazione;
- Test di migrazione;
- Test della documentazione e delle procedure.

Ci sono poi altri tipi di test molto particolari, utilizzati solo in determinate condizioni limite. Alcune di queste caratteristiche, se importanti, possono essere invece svolte all'interno di un test più comune come, ad esempio, quello strutturale o funzionale:

- Test delle transazioni;
- Test di conversione;
- Test di parallelo;
- Test di coesistenza/interfaccia tra prodotti.

La strategia di test identifica quali, fra tutti quelli descritti in precedenza, è necessario prevedere per validare le caratteristiche salienti del software in oggetto.

Ai tipi di test elencati, si può aggiungere un'ulteriore tipologia – anche se la letteratura non la tratta esplicitamente come tale - particolarmente significativa e relativa alle applicazioni "e-business", cioè le applicazioni per il commercio elettronico o quelle con forti componenti tecnologiche Web. Il test relativo è detto:

- Test e-business[10].

Di seguito è fornita una breve descrizione dei tipi di test più comunemente utilizzati nei progetti.

> ***Nota**: possiamo immaginare i "tipi di test" come un "insieme di casi di test" eseguiti durante le fasi del ciclo di sviluppo utilizzando tecniche diverse. Per esempio, la verifica delle funzioni del prodotto è eseguita in fase di disegno tramite "revisioni tecniche"; in fase di test d'integrazione eseguendo i casi di test funzionali; durante il test di sistema tramite scenari funzionali; durante il test d'accettazione tramite l'esecuzione da parte degli utenti di casi di test specifici.*

6.1 Test funzionali

Scopo principale dei test funzionali è quello di validare le caratteristiche esterne del software direttamente legate al suo utilizzo da parte degli utenti o dei gestori applicativi. Tra i vari tipi di test funzionali previsti si descrivono di seguito quelli più comunemente eseguiti nei progetti software:

[10] Il test per le applicazioni di commercio elettronico, e comunque di tute quelle applicazioni con una forte componente tecnologica basata sul Web è descritta nei dettagli in un apposito documento "e-Testing" disponibile nel sito dell'autore (www.colonese.it\pubblicazioni.htm).

- Test delle funzionalità;
- Test di gestione delle condizioni d'errore;
- Test di operatività;
- Test d'installazione;
- Test di regressione;
- Test di parallelo;
- Test di conversione;
- Test d'usabilità.

6.1.1 Test delle funzionalità

Descrizione

Il test verifica che ciascuna funzione richiesta dai requisiti sia correttamente e completamente progettata e sviluppata così come descritto nelle specifiche funzionali del prodotto. Nella fase di progettazione la funzionalità del prodotto è verificata tramite revisioni tecniche; in fase di test d'integrazione tramite i casi di prova progettati secondo le specifiche funzionali; in fase di test di sistema tramite gli scenari funzionali (scenari di test); in fase di collaudo utente tramite casi di test progettati ed eseguiti dagli utenti.

La matrice di test permette di valutare il livello di copertura dei requisiti e delle funzionalità da parte dei casi di test progettati.

Obiettivi

Obiettivo principale del test delle funzionalità è di assicurare che l'applicativo sviluppato:

- indirizzi correttamente tutti i requisiti funzionali espressi e le caratteristiche prestazionali previste;
- esegua le funzionalità in maniera coerente ed accurata;
- elabori le informazioni coerentemente con le politiche, gli standard e le procedure dell'organizzazione.

Esempi

Per garantire un buon test funzionale occorre progettare una serie di:

- casi di test che indirizzino tutti i requisiti funzionali stabiliti;
- scenari di test che permettano agli utenti di completare i loro task (un task è definito come una serie di funzioni eseguite in sequenza per completare un determinato lavoro).

6.1.2 Test di gestione delle condizioni di errore

Descrizione

Il test valuta le modalità con cui il sistema gestisce le condizioni di errore, sia quelle causate dall'utente tramite l'utilizzo improprio del prodotto, sia quelle generate dal sistema stesso. La completezza con cui sono gestite le condizioni di errore è una parte importante dell'usabilità del prodotto. Essa assicura che le transazioni errate saranno gestite in maniera controllata, senza procurare danni al sistema e tenendo sempre informato l'utente.

> ***Nota**: Il test relativo alla gestione delle condizioni di errore non è un test a se stante ma è incluso in tutti i livelli di test (test del disegno, del codice, del sistema).*

Obiettivi

Obiettivo principale del test di gestione delle condizioni di errore è di verificare ed assicurare che il sistema:

- intercetti e gestisca opportunamente tutte le condizioni ragionevoli di errore;
- gestisca in maniera appropriata le condizioni di errore intercettate e assicuri la continuità delle operazioni (solo se rigorosamente necessario, provvederà alla chiusura ordinata ed in sicurezza del sistema);
- esegua opportuni controlli a fronte delle correzioni eseguite.

Dal punto di vista logico, la gestione corretta delle condizioni di errore prevede i seguenti elementi:

- identificazione della condizione di errore;
- segnalazione della condizione di errore;
- esecuzione di azioni opportune per correggere la condizione anomala rilevata;
- registrazione delle informazioni necessarie per l'analisi del problema.

Esempi

Un buon test relativo della gestione delle condizioni di errore richiede di:

- progettare ed eseguire un numero sufficiente di casi di test che simulino tutte le condizioni di errore previste dalle specifiche funzionali;
- eseguire casi di test liberi (free test case) da parte di esperti applicativi.

6.1.3 Test di operatività

Descrizione

Tutti i prodotti rilasciati nell'ambiente di produzione devono ovviamente "performare" secondo i requisiti dichiarati. A funzionare secondo i requisiti non devono essere solo le caratteristiche funzionali, ma anche quelle operative in modo da assicurare ai responsabili della gestione applicativa di poter garantire il livello di servizio richiesto.

Le caratteristiche di operatività del sistema sono verificate durante il test d'integrazione e, principalmente, durante il test di sistema.

Obiettivi

Obiettivo principale del test di operatività è di verificare e assicurare che:

- tutti i componenti del sistema e le procedure siano disponibili ed operativi;
- tutte i comandi on-line siano disponibili e funzionino correttamente;
- tutte le JCL siano conformi agli standard ed operino correttamente;
- tutte le informazioni relative alla schedulazione dei lavori siano corrette;
- la documentazione operativa sia completa, chiara e precisa;
- tutti i lavori batch possano essere completati entro i tempi previsti.

Esempi

Il test di operatività può, per esempio:

- Eseguire un ciclo batch completo e verificare che esso si completi esattamente come descritto nella documentazione operativa disponibile.
- Sottoporre le JCL ad un controllo di aderenza agli standard stabiliti, ecc.

6.1.4 Test d'installazione

Descrizione

Ogni applicazione che deve essere eseguita (deve "girare") in un ambiente diverso da quello in cui è stato sviluppato ha bisogno di essere verificato e validato con un test di installazione. Maggiormente se l'applicazione deve essere eseguita in diverse locazioni remote. Per un pacchetto generalizzato,cioè destinato a utenze diverse, l'installabilità è vitale. Questo tipo di test è assolutamente necessario quando l'installazione dell'applicativo risulta essere complessa, critica, da completare in tempi brevi o essere facile come, ad esempio, per applicativi largamente diffusi (pacchetti su pc). Il test d'installazione dovrebbe essere eseguito da personale con competenza tecnica simile a quella di chi dovrà poi farlo nella realtà (specialista, sistemista, utente finale).

Obiettivi

Il test d'installazione ha l'obiettivo principale di assicurare o verificare che:

- il pacchetto d'installazione includa tutti i componenti richiesti;
- la procedura di installazione sia semplice e precisa;
- la documentazione di installazione sia completa ed accurata;
- il codice da installare sia nel formato corretto ed utilizzabile.

Esempi

Alcune modalità per eseguire il test d'installazione possono essere:

- Verificare i contenuti del package di installazione tramite una lista di controllo (checklist);
- Coinvolgere una persona al di fuori del gruppo di sviluppo e di test che esegua l'installazione utilizzando solo la documentazione fornita a supporto dell'installazione.

6.1.5 Test di regressione

Descrizione

Il test di regressione ha il compito di verificare che una modifica eseguita su una parte di prodotto non crei problemi ad altre parti non modificate.

Un test di regressione verifica se i risultati conseguiti dalle prove dopo le modifiche sono uguali a quelli attesi, cioè a quelli ottenuti prima delle modifiche con prove analoghe. Occorre perciò verificare che esista una base di casi di test con la definizione dei relativi risultati attesi, prima di eseguire le modifiche.

L'analisi delle differenze dei risultati dei test eseguiti prima e dopo le modifiche permette di valutare se l'applicazione si comporta correttamente o no. Situazioni di errore palese sono semplici da individuare, ma piccole variazioni, apparentemente non significative, potrebbero generare malfunzionamenti molto più gravi e più difficile da diagnosticare.

L'utilizzo di procedure di test automatizzate permettono di eseguire verifiche più accurate oltre a ridurre l'impegno di risorse umane in operazioni ripetitive.

E' bene utilizzare il test di regressione insieme al processo di gestione delle modifiche e della configurazione. Ogni modifica autorizzata richiede l'esecuzione di un test di regressione sulla configurazione oggetto di modifica.

Obiettivi

Il test di regressione ha l'obiettivo di verificare e assicurare che:

- la configurazione di un software continui a funzionare correttamente anche dopo le modifiche;
- le modifiche successive ad una versione consolidata non impattino negativamente sul sistema.

Esempi:

Alcune modalità per la conduzione dei test di regressione possono essere:

- Eseguire il test di regressione con i casi di test previsti dopo l'applicazione di una serie di modifiche al termine di una fase di test (per esempio, la correzione di errori minori lasciati per ultimo);
- Produrre una "build" giornaliera ed eseguire in maniera automatica una batteria di casi di test di regressione costituisce una pratica di grande valore (best practice) in un approccio di sviluppo avanzato.

6.1.6 Test di parallelo

Descrizione

Il test di parallelo consiste nell'eseguire le stesse funzioni su due sistemi diversi. Il test permette quindi di comparare i risultati ottenuti dalle prove sugli stessi dati elaborati da entrambi i sistemi (per esempio, il sistema "attuale" e quello "nuovo" che dovrà sostituirlo).

Il test di parallelo si esegue quando una nuova applicazione deve sostituire quella esistente. La verifica dei risultati di entrambe le prove permette di valutare l'opportunità di sostituire le applicazioni riducendo i rischi connessi.

Obiettivi

Il test di parallelo ha l'obiettivo principale di verificare e assicurare che:

- il nuovo sistema dia gli stessi risultati di quelli forniti dal sistema attuale (ovviamente si presuppone che il sistema attuale dia risultati corretti, anche se poco soddisfacenti da altri punti di vista);
- il nuovo sistema si differenzi da quello attuale (in questi casi si presuppone che il sistema attuale non risulti soddisfacente e quello nuovo superi tali limiti oppure si introducano nuove funzioni o caratteristiche).

Esempi:

Alcune modalità per eseguire un test di parallelo sono:

- Un operatore esegue sul nuovo sistema le stesse transazioni eseguite dai suoi colleghi durante la giornata (o un periodo della giornata);
- Si esegue la stessa procedura batch su entrambi i sistemi e si confrontano i risultati.

6.1.7 Test di usabilità

Descrizione

Il test di usabilità valuta se il prodotto finale sia facilmente utilizzabile dagli utenti nella loro operatività quotidiana. Mentre il test funzionale mira a verificare la correttezza con cui le funzioni sono state sviluppate, il test di usabilità mira a valutare la semplicità e la facilità del loro uso. Il test di usabilità è generalmente eseguito durante il test di sistema od il collaudo di accettazione.

Obiettivi

Obiettivo principale del test di usabilità è di verificare e assicurare che:

- il sistema risulti semplice, intuitivo e facile da adoperare ai vari utenti previsti (utenti finali, operatori, gestori, ecc.);
- le schermate e gli output in generale siano concisi e facili da interpretare; le funzioni di aiuto in linea (help on-line) risultino semplici, chiari, esaustivi e principalmente "utili!";
- l'immissione dei dati di input risulti semplice, intuitiva e naturale.

Esempi

Il test dell'usabilità di una determinata funzione può essere valutata verificando che:

- l'immissione dei dati di input di una transazione sia possibile seguendo le abitudini operative dell'utente nel proprio ambiente di lavoro quotidiano;
- l'acquisizione dei risultati ottenuti sia semplice, intuitiva, completa, accurata e secondo le modalità di presentazione normalmente utilizzate dall'utente (secondo gli output abitualmente ottenuti nel lavoro quotidiano);
- chiedendo a qualcuno non coinvolto con lo sviluppo del prodotto di eseguire le funzioni disponibili (o quelle più significative e frequenti) seguendo solo le indicazioni fornite (manuale utente, help on-line) la persona sia in grado di completare il task assegnato nei tempi previsti.

6.2 Test strutturali

Il test strutturale verifica la correttezza tecnica dell'architettura applicativa, la completezza e la correttezza tecnica dei singoli componenti. Verifica cioè che la progettazione sia corretta dal punto di vista tecnico e della sua gestione operativa.

Lo scopo principale dei test strutturali è di verificare che:

- la progettazione sia corretta dal punto di vista tecnico;
- le tecnologie utilizzate siano state implementate in modo corretto e coerente;
- i componenti siano stati progettati in modo da risultare coesi e disaccoppiati;
- le singole parti del sistema siano in grado di funzionare come un insieme unico una volta integrati.

Non è quindi obiettivo di questo test verificare la correttezza funzionale dell'applicativo.

I test strutturali indirizzano solitamente le seguenti caratteristiche dell'applicativo:

- Test di ripristino ("Backup and Recovery");
- Test di sicurezza;
- Test di prestazione ("Performance");
- Test di carico ("Stress" e volumi).

6.2.1 Test di ripristino (Backup and Recovery)

Descrizione

La funzione di ripristino ("recovery") è la capacità del sistema di ripristinare le condizioni iniziali, dopo una sua caduta procurata dal verificarsi di una condizione di errore o a seguito della chiusura forzata da parte del gestore dell'applicazione. Il processo di ripristino richiede una funzione di "memorizzazione" ("backup") dello stato del sistema - transazioni, dati, altre informazioni vitali - in diversi momenti del ciclo operativo. Lo stato del sistema memorizzato deve essere in grado di salvaguardare l'integrità del sistema in oggetto. Ogni stato memorizzato deve permettere di ripristinare le condizioni iniziali (prima della caduta) rieseguendo le transazioni interrotte senza perdita di dati.

Gli elementi critici da tenere presente in fase di progettazione delle funzioni di ripartenza sono molteplici, tra cui: la natura e la complessità dell'applicazione, il volume delle transazioni, la progettazione interna dell'applicativo per gestire il processo di ripartenza, la competenza e l'esperienza del personale preposto a gestire le procedure di ripartenza, la documentazione e le procedure messe a disposizione per la funzione di ripartenza.

Le singole funzioni di ripristino è bene eseguirle nell'ambito del test di sistema. La verifica completa e finale è invece eseguita durante il test di operabilità, quando i requisiti di continuità operativa sono cruciali per il sistema.

A determinare il livello di profondità ed estensione di questo tipo di test è sempre il livello di rischio associato alla caduta del sistema e relativa perdita di dati.

Obiettivi

Gli obiettivi principali del test di "backup and recovery" sono quelli di verificare e assicurare che:

- il sistema sia in grado di continuare ad operare dopo una caduta;
- i dati necessari a ripristinare il sistema siano memorizzati;
- i dati di backup siano accessibili e ripristinabili;
- le procedure di backup e di recovery siano ben documentate e disponibili;
- le persone responsabili di gestire le procedure di backup and recovery ricevano un'adeguata formazione.

Esempi

Alcune modalità di esecuzione del test di backup and recovery possono essere:

- Simulare un ambiente di test simile a quello di produzione in termini di dati e volumi;
- Provocare la caduta del sistema e verificare che le procedure di recovery siano adeguate a gestire la condizione verificatasi.

6.2.2 Test di sicurezza

Descrizione

La sicurezza di un'applicazione deve garantire la protezione delle informazioni del sistema in oggetto e di tutti gli altri sistemi ad esso connessi. La protezione delle informazioni riguarda la loro "perdita" e "utilizzo non consentito", sia intenzionale che accidentale.

Il livello di profondità ed estensione dei test di sicurezza dipendono dal livello di rischio associato alle informazioni gestite dal sistema. I test di sicurezza devono estendersi e limitarsi alle sole funzioni di sicurezza sviluppate, in accordo alle politiche di sicurezza ed in linea con le modalità operative previste. I test di sicurezza sono eseguiti durante la fase di test di sistema, continuano durante il collaudo di accettazione e si concludono con il test di operatività, se previsto.

I test di sicurezza richiedono competenze tecniche specifiche e quindi sono generalmente eseguiti da "esperti di sicurezza".

Obiettivi

Obiettivo principale del test di sicurezza è di verificare e assicurare che:

- i dispositivi di sicurezza sviluppati non siano elusi, alterati o distrutti;
- i rischi di sicurezza siano stati identificati, valutati ed indirizzati appropriatamente;
- la sicurezza sviluppata dal sistema funzioni correttamente.

Esempi

Alcuni test di sicurezza possono includere:

- Tentativo di logon al sistema senza alcuna autorizzazione (quando questa sia richiesta);
- Verifica che la password non sia visibile sui dispositivi di input/output (schermate, stampe);
- Tentativo di fornire l'autorizzazione a funzioni riservate da parte di noi stessi;
- Tentativo di accedere on-line a transazioni non autorizzate e verifica che il sistema sia in grado di proteggersi da tali intrusioni e di segnalare opportunamente i tentativi.

6.2.3 Test di performance

Descrizione

Il test di performance è progettato per verificare se il sistema sia in grado di mantenere le prestazioni richieste nell'ambiente di produzione. Le caratteristiche relative alle prestazioni riguardano i tempi di risposta, l'utilizzo delle risorse, i volumi elaborati o prodotti nell'unità di tempo ed altre considerazioni tecniche relative alla progettazione del sistema. I test di performance sono condotti nel sistema di produzione o simulandone uno simile.

L'attenzione alle prestazioni del sistema sono poste sin dalla fase di disegno. In questa fase sono stabiliti i criteri per l'implementazione e la verifica delle prestazioni. In questa fase possono essere costruiti modelli di prestazioni se richiesti dalle caratteristiche particolari del progetto.

Possono essere eseguite misure delle prestazioni del sistema non appena fossero disponibili parti rilevanti del prodotto, anche se non ancora prive di errore.

Obiettivi

Gli obiettivi principali del test di performance sono quelli di verificare e assicurare che:

- il sistema "performa" come atteso (tempi di risposta, utilizzo delle risorse, quantità di lavoro eseguito nell'unità di tempo, ecc.).

Esempi

Le prestazioni del sistema possono essere testate tramite:

- l'utilizzo di strumenti di simulazione e monitoraggio delle prestazioni del sistema;
- la memorizzazione dei tempi di risposta delle transazioni durante il test di sistema;
- utilizzando prodotti per il monitoraggio delle prestazioni identificando "colli di bottiglia", condizioni di stallo (deadlock) e transazioni particolarmente lente, sia in condizioni normali che durante picchi di lavoro.

6.2.4 Test di carico (Stress test)

Descrizione

Il test di carico è definito come l'esecuzione contemporanea di un numero rilevante di transazioni in un determinato periodo di tempo e con prestazioni definite. Esso tende a verificare che il sistema sia in grado di far fronte ad un picco di lavoro senza degradare le prestazioni (per esempio, elaborare un numero molto alto di transazioni contemporanee con tempi di risposta accettabili per ogni singola transazione).

I fattori di carico si riferiscono ai diversi aspetti del sistema: numero di transazioni eseguite, numero di output prodotti, numero di tabelle interne generate, numero di comunicazioni stabilite, capacità elaborativa del sistema, quantità di lavoro prodotto (throughput), spazio disco occupato, utilizzo dei sistemi di I/O, ecc.

Il test di carico non dovrebbe essere eseguito prima che tutte le funzioni siano state completamente testate e rese stabili. Le caratteristiche del test di carico sono definite nella fase di progettazione ed il test stesso può iniziare appena sono disponibili e stabili nell'ambiente operativo i componenti principali del sistema.

Non è quindi necessario che sia pronto l'intero sistema per poter iniziare il test di carico; è sufficiente che lo siano per parti principali cui è affidato lo smaltimento dei carichi di lavoro (gestione degli I/O, gestione delle code, gestione della memoria, gestione del mulltitasking, ecc.).

Obiettivi

Obiettivo principale del test di carico è di verificare ed assicurare che:

- il sistema di produzione sia in grado di elaborare il numero di transazioni previste nell'intervallo di tempo stabilito;
- l'architettura del sistema e la sua realizzazione siano in grado di elaborare il volume di dati previsto;
- i componenti hardware e software siano adeguati per elaborare i volumi previsti;
- il sistema abbia a disposizione le risorse adeguate a garantire i tempi di risposta previsti;
- le funzioni a supporto dell'elaborazione degli output (per esempio, le stampanti) siano in grado di supportare i volumi previsti.

Esempi

Alcune modalità per l'esecuzione del test di carico sono:

- Testare le condizioni di "overflow" del sistema caricandolo con volumi di lavoro superiori a quelli previsti dalle tabelle interne, dalle code delle transazioni, dalla memoria del sistema, ecc.

- Testare le linee di trasmissione dei dati caricandole con picchi prodotti da simulatori;
- Utilizzare generatori di dati e terminali multipli per stressare il sistema on-line per un periodo di tempo più lungo di quello previsto e per stressare il sistema batch sottomettendo più lavoro di quello previsto.

7 Test delle applicazioni e-business

L'e-business[11] è stata una vera e propria rivoluzione tecnologica ed industriale. Rappresenta il nuovo mercato dove gli affari si trattano sulla rete utilizzando le tecnologie Internet ed il "network computing". Essa richiede però la trasformazione dei processi aziendali per adattarli all'uso della rete (Intranet). La copertura è a 360 gradi, dalle relazioni con i partner ed i fornitori (attraverso l'uso delle reti Extranet) al rapporto con i clienti finali per l'acquisto di beni e servizi da casa o dall'ufficio (e-commerce).

L'e-business crea però anche molti problemi che, se sottovalutati o ignorati, possono rendere vani gli investimenti fatti. Tra quelli più strettamente legati allo sviluppo del software ricordiamo:

- Nel mercato globale i clienti sono dovunque e possono interagire in ogni momento richiedendo la "disponibilità" delle applicazioni su tutta la rete, 24 ore al giorno, 7 giorni a settimana.
- L'"indisponibilità" del servizio (causato dalla "caduta" del sistema – applicazione, rete, ecc. - o da altri fattori come tempi di risposta alti e scarsa usabilità dell'applicazione) possono recare gravi perdite al business.
- Il risultato di alcuni studi compiuti al riguardo da professionisti del settore e da enti di ricerca indicano che lunghi tempi di risposta e scarsa usabilità delle interfacce utente induco circa il 60% (bailout rate) ad abbandonare i siti visitati e, spesso, a non ritornarci più. Numericamente è stato fissato in 8 secondi (la così detta regole degli "8 secondi") il tempo massimo di attesa di una persona senza risposta prima che abbandoni definitivamente la transazione. La scarsa usabilità delle interfacce non fa altro che aggravare il problema.

7.1 Il valore del test e-business

I problemi menzionati sopra creano dei rischi che devono essere opportunamente valutati e mitigati attraverso azioni adeguate. I rischi identificati per le applicazioni di e-business sono relativi a:

- la tecnologia;

[11] Il termine e-Business è stato coniato per indicare gli affari condotti sulla Rete (Web) tramite l'utilizzo di tecnologie informatiche (elettroniche). Con il tempo ha assunto diversi significati legati ad utilizzi più specifici come, ad esempio, new economy, information technology, e-commerce, e-shopping e ogni altra declinazione del business on line.

- le applicazioni;
- la sicurezza.

La metodologia di test qui proposta cerca di fornire una risposta a tali problemi.

Definiamo prima alcuni elementi importanti della metodologia proposta e che possono essere riassunti in:

- Il valore del test è strettamente legato al rischio insito nell'applicazione e-business;
- La pianificazione di test deve includere test specifici per verificare la piena e completa disponibilità dei servizi offerti ovunque (worldwide) e sempre (24 ore / 7 giorni);
- L'investimento sui test deve essere commensurato al rischio economico causato dalle eventuali anomalie riscontrabili nell'applicazione nel suo insieme; per questo occorre prevedere test specifici di Availability, Performance (regola degli 8 secondi, bailout rate), Security, Usability, Scalability;
- La professionalità delle persone coinvolte nelle attività di test è la garanzia di una corretta gestione dei rischi: la capacità di identificare i rischi, di progettare adeguati test, di valutare l'impegno richiesto, di eseguire i test con cura, di controllare e valutare i risultati ottenuti.

Vediamo ora come la metodologia indirizza i rischi menzionati sopra.

7.1.1 Gestione dei rischi legati alla tecnologia

I professionisti del test valutano accuratamente nel disegno le seguenti componenti tecnologiche per valutare i rischi ad essi legati:

- Prestazioni (*Performance*);
- Sicurezza (*Security*);
- Disponbilità ed affidabilità (*Availability and Reliability*);
- Carico e scalabilità (*Stess and Scalability*);
- Usabilità (*Usabilità*).

Ciascun elemento è valutato accuratamente per identificare i potenziali rischi; quindi si progettano i test a fronte dei rischi e delle caratteristiche generali identificati.

7.1.2 Gestione dei rischi legati alle applicazioni

Le applicazioni e-business sono spesso costruite su quelle esistenti (legacy), con architetture multi-layer (host, client/server, utente Web), realizzate da fornitori diversi, basate su catene tecnologiche lunghe e complesse (Web Browser, Internet, Firewall, Web

Server, Application Server, Business Application Server, Host, Database distribuiti, ecc.).

Da tutto ciò nascono alcuni problemi importanti da indirizzare:

- Complessità del sistema;
- Più fornitori, spesso molto diversi per cultura ed approccio allo sviluppo, da integrare in una catena unica del valore;
- Tempi ristretti per lo sviluppo delle applicazioni;
- Budget disponibile spesso inferiore a quello richiesto dalla complessità del sistema.

Il personale di test qualificato deve possedere la professionalità richiesta per:

- Individuare i "colli di bottiglia" il più presto possibile in modo da ottimizzare i tempi e le risorse disponibili;
- Concentrare le attività di test sulle parti più critiche della catena in modo da garantire il completamento delle transazioni applicative;
- Testare anche le parti meno conosciute (sviluppate da altri) per evitare sorprese finali, quando il tempo e le risorse sono esaurite.

Al gruppo di test viene chiesto molto: competenza, professionalità, disponibilità, inventiva, spirito di iniziativa. Il supporto di tecniche, metodi e strumenti è fondamentale per migliorare l'efficacia e la produttività del collaudo e garantire il successo del progetto.

7.1.3 Gestione dei rischi legati alla sicurezza

Occorre contenere i rischi legati alle intrusioni di persone e programmi non autorizzate nella rete dei sistemi e creare danni economici e di immagine. Programmi come i virus possono recare danni considerevoli con gravi perdite economiche per le aziende.

I nemici (*hacker*) possiedono grandi competenze tecniche ed hanno obiettivi diversi: distruggere un sito Web, rubare informazioni, bloccare un sistema, dimostrare di essere capaci di eludere i più sofisticati sistemi di sicurezza, ecc.

La letteratura divide gli hacker in categorie diverse:

- Programmatori particolarmente esperti nei sistemi che aggrediscono. Tendono a scoprire i punti deboli di tali sistemi e sviluppano programmi software in grado di penetrarli, esplorarli e qualche volta a renderli instabili. Sono fondamentalmente mossi dal "piacere" di riuscire in qualcosa che si ritiene non possibile.
- Sistemisti esperti di reti (*System Administrator*, *Network Administrator*). Sono in possesso di molti strumenti (tool) ed in grado di utilizzali per scoprire i punti deboli dei sistemi. Si avvalgono della collaborazione di programmatori esperti

(sempre hacker) per realizzare i programmi che saranno lanciati per penetrare nei sistemi.

- "Script Kiddies". Si tratta di persone non specialiste in informatica, con poche competenze ma quanto basta per creare problemi, a volte anche seri. L'unico scopo di queste persone è quello di fare danni. Per questo motivo sono viste con disprezzo anche dal mondo degli stessi hacker. Tipicamente scaricano programmi da Internet e li eseguono anche nei propri uffici.

La soluzione a questi problemi è quella di assicurare la presenza di personale interno[12] qualificato ed affidabile (detti "Ethical Hacker") in grado di assistere le Aziende ed ai loro Clienti nel verificare il livello di sicurezza realizzato dai sistemi sviluppati evidenziandone le scoperture.

In sintesi, un test di sicurezza prevede le seguenti verifiche e controlli:

- Verifica dell'attendibilità della sicurezza;
- Verifica della configurazione;
- Verifica funzionale delle componenti di sicurezza;
- Verifica di intrusione.

[12] Quando non sia possibile assicurare personale interno qualificato nei tempi richiesti dal progetto occorre avvalersi della collaborazione esterna di consulenti qualificati, competenti ed affidabili.

8 Fasi del processo di testing

La complessità dei sistemi sviluppati oggi assegna grande importanza ad una buona pianificazione e realizzazione della fase di test. L'introduzione delle tecnologie legate alla Rete, inoltre, ha portato ad una sostanziale modifica dell'architettura applicativa e ad una più specifica e, spesso, più critica necessità di soddisfare le esigenze dell'utenza. Questa, diventata più eterogenea, vasta e globale (è il caso delle applicazioni di e-business) pone in luce nuovi aspetti tecnici e qualitativi del software: sicurezza delle transazioni, tempi di risposta della rete e dell'applicazione stessa, disponibilità h24 e 7/7 dei servizi offerti dall'applicazione, usabilità ed altre caratteristiche ancora (il tema è trattato in un capitolo specifico di questo manuale).

La metodologia di test identifica le seguenti fasi o macro attività (tra parentesi sono riportati i nomi inglese della figura):

- Pianificazione dei test (*Test Planning*);
- Preparazione e sviluppo dei test (*Test Preparation / Development*);
- Esecuzione dei test (*Unit Test, Integration Test, System Test, Acceptance Test*);
- Gestione dei difetti (*Defect Management*);
- Monitoraggio e reportistica dei test (*Test Monitoring and Reporting*);
- Gestione del processo di test (*Test Process Management*)
- Gestione della configurazione di test (*Test Configuration Management*).

8.1 Pianificazione dei test

L'attività di pianificazione si concretizza nella produzione del documento specifico "piano di test". Ricordarsi che una buona pianificazione dei test tiene conto della famosa regola d'oro richiamata già nel manuale. In particolare, la stima dell'impegno necessario per le attività di test sono discusse nel successivo paragrafo di questo capitolo e le regole sono riassunte nella **Tabella 1**.

Il piano è un documento formale che descrive le attività di test previste per la validazione del prodotto. Esso contiene la strategia di test, cioè quali tipi di test verranno effettuati, e la descrizione di ognuno di essi. In particolare, per ogni specifico tipo di test, descrive: gli obiettivi, i responsabili, i criteri di ingresso e di uscita, l'ambiente in cui verrà svolto, l'elenco dei casi di test, le banche dati di prova, gli eventuali strumenti (tool) da utilizzare, ecc.

Per progetti complessi o di grandi dimensioni è consigliabile produrre un Piano di test generale (Master Test Plan) che descriva la strategia di test che si intende adottare e le linee guida da seguire per i singoli test previsti. Sulla base di tale piano generale si producono uno o più piani di dettaglio per i singoli test previsti. In ciascun piano di dettaglio sono indicati i dettagli operativi quali, ad esempio, l'ambiente di test richiesto, la base dati da utilizzare, i casi di test da eseguire ed altri dettagli significativi. La **Figura 3** mostra la relazione tra i vari piani.

La strategia di test è definita sulla base di specifici criteri quali, ad esempio, la ricchezza funzionale e le caratteristiche del prodotto, la criticità dell'applicazione per il business, la complessità tecnica del disegno, la piattaforma tecnologica utilizzata in esercizio, le modalità di rilascio in esercizio dell'applicativo, la durata del progetto, i costi di realizzazione. In particolare, la strategia di test è definita sulla base alla strategia di sviluppo scelta per indirizzare i vari fattori critici del progetto.

> ***Esempio**: a fronte di uno sviluppo che preveda il rilascio periodico di componenti autoconsistenti in termini di funzionalità, la relativa strategia di testing deve prevedere il collaudo completo di ciascun rilascio e la successiva integrazione dei diversi rilasci.*
>
> *Analogamente, una strategia di sviluppo di tipo iterativo-evolutivo, in cui il prodotto si realizzi in modo ciclico aggiungendo di vota in volta funzionalità ad un corpo base, la corrispondente strategia di test dovrà prevedere un'integrazione continua di nuovi moduli e di moduli esistenti modificati man mano che lo sviluppo procede. Acquista notevole importanza, in questo caso, una strategia che preveda un test "continuativo" d'integrazione e di regressione.*

Il piano di test, documentato, revisionato ed approvato all'inizio del progetto, è mantenuto aggiornato a seguito di eventuali modifiche ai requisiti od altri elementi.

> ***Esempio**: al termine della progettazione, il piano di test è aggiornato con la lista dettagliata dei casi di test e con la matrice di test non ancora disponibili al momento della pianificazione del progetto*

I risultati dei test sono documentati in rapporti periodici che forniscono dettagli sui casi di test eseguiti, completati, in attesa di essere eseguiti o completati, interrotti e/o bloccati a causa di difetti rilevati, ecc. I rapporti forniscono quindi dettagli anche sullo stato di risoluzione dei difetti rilevati.

8.1.1 Stima del costo dei test

Il costo delle attività di test è direttamente legato al livello di rischio identificato per il progetto. Una delle cause di difficoltà di molti progetti è la sottostima del dimensionamento delle attività di test. Nonostante la regola d'oro[13] assegni alle attività di test e di

[13] Si tratta della regola d'oro definita da Frederick P. Brooks nel suo famoso libro "*The Mithical Man-Month: Essays on Software Engineering – 20th Anniversary Edition, 1995,* Addison-Wesley, Reading(MA)

gestione del progetto non meno del 50% del costo totale del progetto, la maggior parte dei progetti non pianifica più del 10-15%, salvo poi eseguire una serie di attività correttive non pianificate che portano il progetto a superare il budget.

La Tabella x fornisce una guida alla valutazione del costo delle attività di test in base al livello di rischio del progetto.

Tabella 1. Stima dell'impegno per il test in base al livello di rischio.

LIVELLO DI RISCHIO	IMPEGNO PER IL TEST
Rischio alto	Pianificare un impegno complessivo per le attività relative a tutti i livelli di test previsti pari a circa il **50-60%** dell'impegno totale previsto per l'intero progetto. I fattori che maggiormente impattano i rischi ed i costi del progetto sono: dimensioni del progetto, complessità, date dei rilasci previsti, tecnologie adoperate ed altri fattori di minore importanza.
Rischio medio	Pianificare un impegno complessivo per le attività relative a tutti i livelli di test previsti pari a circa il **20-50%** dell'impegno totale previsto per l'intero progetto.
Rischio basso	Pianificare un impegno complessivo per le attività relative a tutti i livelli di test previsti pari a circa il **10-20%** dell'impegno totale previsto per l'intero progetto.

I dati sono presi dalla letteratura disponibile sull'argomento e sull'esperienza pratica dell'autore.

8.2 Preparazione e sviluppo dei test

La progettazione dei casi di test tiene conto dei seguenti fattori:

- Scopo del test (test funzionale, test di performance, di usabilità, ecc.);
- Complessità dell'applicazione;
- Tempo a disposizione dei test.

I casi di test sono creati a partire dai requisiti e dalle specifiche tecniche. Da queste vengono estratti:

1. la lista delle funzioni disponibili ed i relativi parametri associati;
2. le indicazioni operative, cioè la sequenza delle operazioni che l'utente esegue nello svolgimento dei task (ad esempio l'immissione d'un ordine);

3. le caratteristiche tecniche del prodotto (usabilità, performance, ecc.).

Partendo da questi elementi (funzioni, operatività e caratteristiche tecniche) vengono realizzati i casi di test. Ciascun caso di test identifica una o più funzionalità da validare; Le funzionalità sono descritte utilizzando le modalità operative che l'utente deve seguire per completare un task, cioè un proprio lavoro.

La matrice di test mete in relazione i casi di test con le funzioni ed i parametri indirizzati. Essa permette di valutare il livello di copertura che i casi di test garantiscono rispetto alle funzioni ed ai requisiti del prodotto.

La matrice di test permette di individuare eventuali scoperture o duplicazioni nell'esercitare le funzioni. Permette anche di ottimizzare i test in termini economici. Ha quindi un duplice effetto positivo: sulla qualità e sull'economicità dei test. Offre garanzia di qualità assicurando che tutti i requisiti siano indirizzati e permette di ridurre i costi eliminando i casi di test duplicati.

I casi di test devono essere progettati utilizzando criteri economici (massimo risultato con minimo sforzo). Ciò è realizzato utilizzando la matrice di test con la quale si può procedere ad eliminare i casi di test ridondanti ed accorpare più casi di test in uno unico. In tal modo si è in grado di identificare la "batteria di test" minima più efficace.

I casi di test progettati sono archiviati in un archivio (*Test Cases Repository*). I casi di test potranno essere riutilizzati in ogni momento successivo per effettuare i test di regressione o validare le modifiche di manutenzione evolutiva, correttiva e migliorativa e adattativa.

L'ambiente dove si eseguono i test è importante ai fini dell'efficacia dei test stessi e dei risultati conseguiti. L'ambiente di test è opportunamente predisposto per simulare, quanto più possibile, le condizioni al contorno che influenzano l'applicazione da testare (sistemi hardware e software, altre applicazioni con cui interagire, banche dati, ecc.).

L'ambiente di test è "validato" prima di iniziare l'esecuzione dei test, in conformità a quanto previsto dalla normativa ISO 9000[14]. Tale validazione può essere effettuata utilizzando due modalità differenti:

- eseguendo sull'ambiente nuovo casi di test esistenti ed i cui risultati siano perfettamente noti (per esempio, eseguendo casi di test noti sulla versione precedete del prodotto);
- creando un caso di test 'ad hoc' il cui risultato sia certo; in tal caso è necessario che il software di riferimento sia privo di anomalie (cioè garantisca il risultato atteso).

[14] La validazione dell'ambiente, ed inclusi gli eventuali strumenti adoperati (tool), è richiesto per garantire che i risultati ottenuti dai test non siano in nessun modo influenzati/alterati dall'ambiente stesso.

In particolare, le banche dati di prova sono importanti ai fini dell'efficacia dei test in quanto condizionano fortemente i risultati dei test. Per simulare le condizioni finali nelle quali l'applicazione opererà quando sarà in esercizio, è opportuno che tutti i componenti, hardware, software e banche dati, siano il più possibile simili a quelli finali del committente. Ambienti particolari possono essere predisposti, invece, per eseguire test particolari come quelli di usabilità, di performance, di stress, ecc.

8.3 Esecuzione dei test

L'esecuzione dei test è fatta partendo dal basso: esercitando prima i singoli moduli, poi integrando i moduli in componenti e verificandone la loro consistenza, infine integrando i componenti fino a completare il sistema finale. I vari livelli di test sono quindi descritti in ordine di esecuzione:

1. Test unitario;
2. Test d'integrazione;
3. Test di sistema (o Collaudo interno);
4. Test di accettazione (Collaudo utente).

Ciascun test è descritto in termini di:

- Descrizione;
- Criteri di ingresso e di uscita della fase di test;
- Prodotti di fase realizzati (deliverable);
- Attività svolte;
- Controlli, verifiche e validazioni effettuate;
- Ruoli coinvolti;
- Tecniche adottate;
- Metriche applicate;
- Evidenze prodotte e mantenute.

8.3.1 Test unitario

Descrizione

Il test unitario è a carico dei programmatori; esso è descritto in questo documento per motivi di completezza della metodologia esposta.

La validazione dei singoli moduli (programmi, procedure, moduli, tabelle, ecc.) è fatta dal programmatore tramite l'esecuzione di test unitari nel proprio ambiente di sviluppo con l'ausilio di strumenti software di "debugging". La tecnica del test unitario (vedi metodo "white box") prevede l'esecuzione di tutti i rami logici del programma utilizzando dati che esercitino le funzioni sviluppate, le condizioni di errore, quelle particolari e quelle limite. Gli errori rilevati durante l'esecuzione dei test unitari sono corretti dal programmatore stesso. Una volta completati con successo i test unitari, i moduli sono promossi (in accordo con il piano di gestione della configurazione) nelle librerie di consolidamento dove saranno integrati con gli altri programmi e dove saranno eseguiti i test funzionali o di integrazione.

Criteri di ingresso e di uscita del test unitario

Il test unitario può iniziare quando siano disponibili i seguenti elementi:

- Piano di progetto approvato che includa i test unitari fra le attività di sviluppo;
- Codice sorgente da testare completamente sviluppato;
- Casi e/o condizioni di test unitario identificate;
- Ambiente di sviluppo in cui eseguire il test unitario corredato dei relativi tool di debugging e disponibilità dei vari moduli di scaffolging (driver e stub);

Il test unitario può ritenersi concluso quando siano verificate le seguenti condizioni:

- Tutti i casi di test e/o condizioni di test sono stati completati con successo;
- Tutti gli errori rilevati sono stati corretti;
- E' stato raggiunto il livello di copertura atteso (es. 100%).

Deliverable prodotti

Il test unitario produce i seguenti output:

- Moduli testati unitariamente (tutti gli errori corretti) e promossi nelle librerie apposite;
- Report di completamento del test unitario.

Attività svolte

Le attività propedeutiche all'esecuzione del test unitario:

- Pianificare le attività relative al test unitario nella fase di codifica;
- Progettazione dei casi di test da eseguire in base alle funzionalità dei singoli moduli ed alle diverse condizioni del codice sviluppate (le condizioni che portano ai diversi rami del codice sorgente).

L'esecuzione del test unitario prevede le seguenti attività di dettaglio:

- Esecuzione dei casi di test unitario identificati;
- Correzione degli errori rilevati;

- Promozione dei moduli testati con successo nelle librerie di sviluppo (in accordo alla gestione della configurazione).

Controlli e verifiche effettuate

L'attività di test unitario richiede che sia controllato e verificato quanto segue:

- Verifica dell'utilizzo degli standard di programmazione tramite ispezioni a campione;
- Revisione tecnica di eventuali algoritmi particolari o moduli complessi;
- Verifica del livello di copertura dei test (verifica che siano stati coperti tutti i rami e le condizioni di scelta all'interno del codice sorgente);
- Controllo dello stato di completamento dei test unitari previsti, dello stato di correzione degli errori rilevati e del livello di copertura raggiunto;
- Controllo della qualità del codice in termini di difettosità e di livello di aderenza agli standard di programmazione.

Ruoli coinvolti

L'attività di test unitario vede il coinvolgimento dei seguenti ruoli:

- *Programmatore*: il test unitario è eseguito dallo stesso sviluppatore dei moduli testati;
- *Altri programmatori*: nel caso siano eseguite ispezioni al codice sorgente (particolari algoritmi o parti più critiche del codice);
- *Assicurazione qualità*: verifica del completamento dei test unitari, della correzione degli errori rilevati, del livello di copertura dei test unitari, dell'utilizzo degli standard di programmazione (verifica a campione).

Tecniche adottate

Il test unitario è supportato dall'utilizzo delle seguenti tecniche e strumenti:

- Standard di programmazione generale e per i singoli linguaggi utilizzati da seguire per la scrittura del codice;
- Revsione tecnica (Peer Review) per l'ispezione del codice;
- Metodo white box per il test del codice sorgente;
- Tool di debugging per l'esecuzione dei test, l'immissione dei dati, il controllo dei risultati, la verifica della copertura dei rami e delle condizioni del codice sorgente, ecc.

Metriche applicate

L'efficacia del test unitario è valutata effettuando le seguenti misure:

- Livello di copertura dei test eseguiti (% di rami e di condizioni testate);

- Numero di errori rilevati e numero di errori corretti;
- Difettosità del codice testato (numero di errori per Kloc o per KFP);
- Indice di complessità essenziale del codice.

L'efficacia del processo di test unitario è valutata effettuando le seguenti misure:

- Livello di aderenza del software agli standard definiti;
- Difettosità del codice sorgente rispetto ai valori attesi;
- Livello di copertura dei test rispetto al valore target.

Evidenze prodotte e mantenute

Il processo prevede che in fase di test unitario siano prodotte e conservate le seguenti registrazioni della qualità:

- Rapporto di completamento dei test unitari;
- Rapporto di promozione del codice testato nelle librerie.

8.3.2 Test d'integrazione

Descrizione

Il test d'integrazione ha lo scopo di assicurare la correttezza dei diversi componenti quando questi siano integrati nel prodotto. Il test è eseguito nella libreria dove i componenti sono integrati man mano che sono completati dagli sviluppatori dopo il test unitario. Ciò consente di eseguire il test in parallelo allo sviluppo, riducendo il tempo di realizzazione del progetto. Il test può considerarsi, per questo motivo, come un test incrementale.

Il test d'integrazione verifica la completezza, la correttezza e l'aderenza ai requisiti de:

- le funzionalità sviluppate;
- la gestione delle condizioni d'errore e delle condizioni limite;
- le prestazioni dei vari componenti, se previste (performance, affidabilità, usabilità delle interfacce, integrabilità con altri sistemi, ecc.);

L'integrazione dei componenti e l'esecuzione dei relativi test d'integrazione adotta la tecnica "Bottom-up", "Top-down" o l'insieme delle due tecniche.

I componenti non ancora sviluppati sono simulati da oggetti fittizi (detti simulatori o "scaffolding"): "driver" o "stub". I simulatori sono sostituiti dai componenti "veri" man mano che questi sono resi disponibili dallo sviluppo.

Criteri di ingresso e di uscita del test d'integrazione

Il test d'integrazione può iniziare quando siano soddisfatte le seguenti condizioni:

- Piano di test approvato e risorse previste disponibili;

- Test unitario completato con successo;
- Moduli installati nell'ambiente d'integrazione;
- Casi di test progettati e disponibili;
- Ambiente di test disponibile secondo la configurazione prevista (software di base e a corredo, banche dati di prova, programmi e procedure, tool).

Il test d'integrazione può ritenersi concluso quando siano soddisfatte le seguenti condizioni:

- Tutti i casi di test completati con successo;
- Tutti gli errori rilevati, registrati e corretti.

Deliverable prodotti

Sono prodotti i seguenti output:

- Moduli integrati e verificati secondo i casi di test eseguiti;
- Report di completamento del test d'integrazione.

Attività svolte

Attività propedeutica all'esecuzione dei test (pianificazione e preparazione):

- Pianificazione del test d'integrazione e produzione del rispettivo Piano di test;
- Progettazione dei Casi di test e della Matrice di test con produzione della relativa documentazione;
- Revisione tecnica dei casi di test e della matrice di test per verificarne la completezza e la correttezza;
- Verifica della completezza e della correttezza dei simulatori previsti;
- Predisposizione e validazione dell'ambiente di test.

Attività specifiche dell'esecuzione del test d'integrazione:

- Esecuzione dei casi di test progettati;
- Registrazione dello stato di completamento dei casi di test eseguiti;
- Registrazione dei malfunzionamenti rilevati (e notifica agli sviluppatori nel caso non sia utilizzato uno strumento software che esegua la notifica automatica);
- Verifica della correzione degli errori rieseguendo i casi di test interrotti a seguito dei malfunzionamenti rilevati;
- Produzione del rapporto periodico sullo stato di completamento dei casi di test e lo stato di correzione degli errori;
- Chiusura della fase di test d'integrazione al completamento con successo delle attività previste.

Controlli e verifiche effettuate

L'esecuzione dei test d'integrazione è sottoposta ai seguenti controlli e verifiche:

- Revisione tecnica dei casi di test e della matrice di test per verificarne la completezza e la correttezza;
- Controllo dello stato d'avanzamento del test (numero di casi di test ancora da eseguire, eseguiti, bloccati in attesa di correzione degli errori, completati con successo);
- Controllo della risoluzione degli errori bloccanti e gravi (parte di quelli di gravità minore possono essere risolti in un secondo momento secondo un piano da specificare);
- Verifica della qualità del software raggiunta (numero di errori rilevati rispetto a quelli previsti).

Ruoli coinvolti

L'esecuzione dei test d'integrazione vede il coinvolgimento diretto dei seguenti ruoli:

- *Capo progetto*: pianifica e controlla lo stato di avanzamento del test; produce il rapporto periodico sullo stato di completamento dei test;
- *Tester*: progetta i casi di test e la matrice di test; prepara l'ambiente di test; realizza i simulatori; esegue i casi di test; rileva gli errori e li comunica allo sviluppo; verifica la correzione degli errori rieseguendo i casi di test interrotti; registra l'esecuzione dei casi di test, registra il completamento dei casi di test;
- *Programmatore*: corregge i malfunzionamenti segnalati;
- *Assicurazione qualità*: verifica i risultati delle revisioni tecniche, dei test eseguiti e dei malfunzionamenti corretti; valuta il livello qualitativo del software testato e del processo seguito.

Tecniche adottate

L'esecuzione dei test d'integrazione utilizza le seguenti tecniche e strumenti a supporto:

- Test Black-box;
- Test Bottom-up e/o Top-down;
- Ispezioni, Revisioni, Walkthrough;
- Scaffolding (Driver o Stub);
- Profilo di qualità del software;
- Curva di rimozione degli errori durante i test.

I documenti sono prodotti secondo i seguenti modelli disponibili:

- Piano di test;
- Casi di Test e Matrice di Test;
- Rapporto sullo stato di completamento del test e di correzione degli errori.

Metriche utilizzate

L'efficacia dei test d'integrazione è valutata tramite le seguenti metriche:

- Indice di qualità del software;
- Livello di copertura del test;
- Indice di efficienza del test.

Evidenze prodotte e mantenute

Sono prodotte e mantenute le seguenti registrazioni della qualità:

- Rapporto di revisione tecnica dei casi di test e della matrice di test;
- Rapporto di completamento del test.

8.3.3 Test di sistema

Descrizione

Scopo del test di sistema è assicurare che il prodotto finale, completo di tutti i componenti integrati, soddisfi i requisiti del committente (funzionalii e prestazionali) così come specificato nel documento "Specifiche funzionali" e nel "Piano di qualità". Questo tipo di test rappresenta il collaudo interno del prodotto ed assicura la direzione aziendale che le aspettative del committente sono state tutte correttamente indirizzate.

Il test di sistema è composto da diverse tipologie di test:

- Test "**funzionale**": sempre eseguito;
- Test di "**usabilità**": eseguito per applicazioni con forti interazioni con gli utenti, per applicazioni Web, per applicazioni destinate a utenti con scarse conoscenze informatiche, ecc.
- Test di "**performance**, **stress**, **affidabilità**": eseguiti quando all'applicazione sono richieste prestazioni particolari (es.: tempi di risposta particolari, utilizzo contenuto delle risorse, ecc.), condizioni di esercizio particolarmente stressanti (es.: 24 ore su 24 ore, 7 giorni su 7 giorni, ecc.), particolari requisiti di affidabilità (es.: assenza di errore per almeno 8 ore consecutive di esercizio);
- Test di "**sicurezza**": eseguiti quando all'applicazione sono richieste particolari requisiti di sicurezza (es.: protezione da attacchi e violazioni esterne intenzionali o meno, ecc.);
- Test di "**installazione**": eseguito sempre, se il prodotto deve essere distribuito a postazioni remote e deve essere installato in ambienti diversi, automaticamente o manualmente (es.: distribuzione e installazione automatica/manuale presso le sedi/agenzie periferiche di una banca, assicurazione, altra azienda).
- Test di "**portabilità**": eseguito se il prodotto deve funzionare su piattaforme differenti.

Un'altra tipologia di test significativa è:

- Test "**e-business**": eseguito per le applicazioni dette di "e-business", con forti componenti e tecnologie Web (il tipo di test è descritto in un paragrafo apposito della metodologia).

> ***Nota**: nel test di sistema i casi di test sono raggruppati in "**scenari di test**". Uno scenario di test è una sequenza di casi di test eseguiti in un ordine ben preciso che permette ad un potenziale utente del sistema di completare un task, cioè un proprio lavoro (esempio: apertura di un nuovo conto corrente, assegnazione del fido, modifica dei dati del conto, versamenti e prelievi sul conto, stampa dell'estratto conto, chiusura del conto). E' quindi chiaro cosa si intenda da ora in poi per scenario di test e la sua relazione con i casi di test.*

L'ambiente del test di sistema deve essere simile a quello finale di esercizio, sia nella configurazione del hardware e del software che della banche dati di prova e delle interfacce con altri sistemi con cui dovrà interagire.

E' opportuno che il test di sistema preveda tutti i casi test che saranno eseguiti nel collaudo di accettazione, in modo da prevenire eventuali problemi durante l'esecuzione del collaudo con gli utenti.

Criteri di entrata e di del test di sistema

Il test di sistema può iniziare quando siano verificati i seguenti criteri:

- Piano di test approvato e risorse previste disponibili;
- Test d'integrazione completato con successo;
- Manuali completi e disponibili;
- Casi di test, scenari di test e ambiente di test disponibili.

Il test di sistema può ritenersi completato quando siano verificati i seguenti criteri:

- Tutti i casi di test e scenari di test completati con successo;
- Tutti gli errori rilevati, registrati e corretti.

Deliverable prodotti

Il test di sistema produce i seguenti output:

- Prodotto completo testato con successo secondo i casi di test e scenari di test previsti;
- Rapporto di completamento del test di sistema.

Attività svolte

Il test di sistema prevede le seguenti attività propedeutiche (pianificazione e preparazione):

- Pianificazione del test, produzione e approvazione del Piano di test di sistema;
- Progettazione dei casi di test e degli scenari di test e produzione della relativa documentazione;
- Predisposizione e validazione dell'ambiente di test di sistema;
- Migrazione completa del software nell'ambiente del test di sistema (secondo la gestione della configurazione).

Attività specifiche di esecuzione del test di sistema:

- Esecuzione dei casi di test e degli scenari previsti;
- Registrazione dello stato di esecuzione e di completamento dei casi di test e degli scenari di test;
- Registrazione dei malfunzionamenti individuati e notifica al gruppo di sviluppo perché corregga gli eventuali errori;
- Correzione dei malfunzionamenti da parte del gruppo di sviluppo (programmatori);
- Verifica della correzione dei malfunzionamenti rieseguendo i casi di test e gli scenari di test coinvolti.
- Chiusura della fase di test di sistema ed emissione del Verbale di completamento del test di sistema.

Controlli e verifiche

Il processo prevede i seguenti controlli e verifiche:

- Revisione dei casi di test e degli scenari di test progettati;
- Controllo dello stato di completamento dei casi e scenari di test;
- Controllo dello stato di risoluzione degli errori;
- Controllo della qualità del prodotto finale secondo le specifiche dichiarate nel Piano di qualità (difettosità, copertura funzionale, usabilità, performance, ecc.).

Ruoli coinvolti

Il test di sistema vede il coinvolgimento dei seguenti ruoli:

- *Capo progetto*: pianifica e controlla l'avanzamento del test;
- *Tecnico di test*: progetta i casi di test e gli scenari di test, prepara l'ambiente di test, esegue i casi di test, registra l'esecuzione del test, effettua la revisione tecnica dei casi di test e degli scenari di test;
- *Programmatore*: corregge gli errori che rilevati durante l'esecuzione del test.

Tecniche adottate:

L'esecuzione del test di sistema utilizza le seguenti tecniche e strumenti a supporto:

- Test Black-box;
- Revisione tecnica dei casi di test e degli scenari di test;
- Profilo di qualità del software (curva di errori previsti per fase);
- Curva di saturazione degli errori rilevati.

I documenti sono prodotti utilizzando i seguenti modelli disponibili:

- Casi di Test e scenari di test;
- Piano di test di sistema;
- Rapporto sullo stato di completamento del test.

Metriche disponibili:

L'efficacia del test di sistema è valutato in base alle seguenti metriche:

- Indice di qualità del software (difettosità, usabilità, performance, ecc.);
- Indice di efficacia dei test (livello di copertura di test, numero di errori rilevati rispetto a quelli attesi);
- Indice di efficienza del test (tempo medio di esecuzione di un caso/scenario di test, produttività media del gruppo di test).

Evidenze prodotte e mantenute

Il processo prevede che siano prodotte e conservate le seguenti evidenze:

- Approvazione del piano di test;
- Rapporti di completamento dei test;
- Rapporti sui risultati dei test specifici (usabilità, performance, ecc.).

8.3.4 Collaudo utente o di accettazione

Descrizione

Questo tipo di test è eseguito direttamente dal cliente per "garantire la conformità" del prodotto finale alle proprie esigenze di business, così come dichiarato nei requisiti.

Il collaudo consiste nel definire i criteri di accettazione e nel verificare che essi siano soddisfatti. I criteri di accettazione sono, solitamente, definiti formalmente nel contratto e includono, tra l'altro, che:

- tutti i casi di test previsti (concordati) siano completati con successo nell'ambiente di collaudo opportunamente predisposto;
- tutti i malfunzionamenti rilevati siano stati corretti e verificati.

L'esito del collaudo è formalizzato nell'apposito "Verbale di collaudo".

All'esito positivo del collaudo è solitamente legato il completamento del progetto di sviluppo e l'eventuale pagamento del corrispettivo.

Il collaudo è generalmente eseguito in un ambiente apposito (ambiente di collaudo), copia esatta di quello in esercizio. Al termine del collaudo con esito positivo, il prodotto è messo in esercizio.

Il collaudo è solitamente effettuato da utenti che rappresentano le diverse tipologie di utenza previste (utenti che utilizzano specifiche funzionalità del prodotto in particolari scenari d'uso).

Il collaudo d'accettazione può essere progettato ed eseguito direttamente dal cliente (o da una società di consulenza di sua fiducia) o dal fornitore stesso con l'approvazione del cliente.

La documentazione di collaudo è composta da quattro elementi principali:

- Piano di collaudo;
- Criteri di accettazione;
- Casi di test e scenari di test;
- Rapporto di esito collaudo.

Dei criteri di accettazione si è detto sopra. La definizione del collaudo include l'accordo anche sui seguenti elementi:

- Severità degli errori riscontrati (solitamente sono definite tre livelli di severità o gravità: errore **bloccante** del sistema, errore **grave** ma non bloccante del sistema, errore **minore**, che non pregiudica la continuazione delle attività di collaudo sulle funzioni testate);

- Modalità di gestione degli errori (modalità per l'assegnazione del livello di gravità ai malfunzionamenti rilevati, tempi di risoluzione degli errori, modalità di registrazione, di notifica e di verifica delle correzioni).

> ***Nota**: Nel caso il collaudo sia progettato e realizzato dal cliente, il fornitore deve comunque concordare quanto previsto nelle attività e fornire il supporto tecnico per la realizzazione del collaudo.*

Criteri di ingresso e di uscita del collaudo d'accettazione

Il collaudo di accettazione può iniziare quando siano verificati i seguenti elementi:

- Piano di collaudo concordato;
- Criteri di accettazione concordati;
- Ambiente di collaudo predisposto e disponibile;
- Casi di test disponibili;
- Prodotto testato con successo a livello di sistema.

Il collaudo di accettazione può ritenersi completato quando siano stati verificati i criteri di uscita, e cioè:

- Tutti i casi di test completati con successo;
- Tutti i malfunzionamenti rilevati siano stati opportunamente corretti e verificati (almeno tutti gli errori classificati bloccanti e gravi e parte di quelli di gravità minore, mentre per i rimanenti può essere concordato un piano di risoluzione);
- Eventuali altri criteri definiti e concordati tra le parti.

Deliverable prodotti

Il collaudo di accettazione può ritenersi completato quando siano stati verificati i criteri di uscita, e cioè:

- Prodotto collaudato secondo i casi di test previsti;
- Verbale di collaudo.

Attività svolte

L'esecuzione del collaudo utente richiede alcune attività propedeutiche (pianificazione e preparazione):

- Pianificazione del collaudo con emissione e approvazione del Piano di collaudo;
- Progettazione dei casi di test, loro documentazione e approvazione;
- Predisposizione e validazione dell'ambiente di collaudo.

L'esecuzione del collaudo utente prevede le seguenti attività specifiche:

- Esecuzione dei casi di test previsti[15];
- Registrazione dell'esecuzione dei casi di test e del loro completamento;
- Registrazione dei malfunzionamenti e notifica al gruppo di sviluppo perché li corregga opportunamente;
- Accordo sul livello di gravità da assegnare al malfunzionamento;
- Correzione degli errori (da parte del gruppo di sviluppo);
- Verifica della correzione degli errori tramite la riesecuzione dei casi di test coinvolti dai malfunzionamenti;
- Emissione del Verbale di collaudo e chiusura della fase di collaudo.

Controlli e verifiche effettuate

Il processo prevede i seguente controlli e verifiche:

- Revisione dei casi di test con gli utenti (quando questi non siano stati progettati direttamente dall'utente);
- Controllo dello stato di completamento dei casi di test;
- Controllo dello stato di risoluzione degli errori riscontrati;
- Verifica dei criteri di completamento del collaudo e accordo su azioni necessarie per la risoluzione di eventuali situazioni critiche.

Ruoli coinvolti

Il collaudo utente vede il coinvolgimento dei seguenti ruoli:

- *Capo progetto del cliente e del fornitore*: pianificano il collaudo, controllano lo stato d'avanzamento delle attività e lo stato di risoluzione dei problemi, concordano i criteri di ingresso e di uscita del collaudo, concordano azioni necessarie per risolvere eventuali situazioni critiche, verificano i criteri di completamento e dichiarano la chiusura con successo del collaudo;
- *Utente*: progetta ed esegue i casi di test, registra l'esito dei test eseguiti ed i malfunzionamenti rilevati, verifica la risoluzione degli errori;
- *Tecnico di test qualificato (fornitore)*: supporta la progetta dei casi di test e la preparazione dell'ambiente di collaudo;
- *Programmatore (fornitore)*: corregge gli errori a fronte dei malfunzionamenti riscontrati.

[15] E' bene che i casi di test previsti dal collaudo di accettazione siano stati precedentemente verificati durante il test di sistema per evitare che siano riscontrati errori e malfunzionamenti che possano pregiudica lo svolgimento del collaudo stesso ed il suo completamento con successo.

Tecniche e strumenti adoperati

Il collaudo utente prevede l'utilizzo delle seguenti tecniche e degli strumenti:

- Test Black-Box;
- Revisione tecnica.

La documentazione è prodotta utilizzando i seguenti modelli disponibili:

- Piano di collaudo;
- Casi di test;
- Verbale di collaudo.

Metriche applicate

L'efficacia e l'efficienza del collaudo (lato fornitore) è valutata in base alle seguenti misurazioni:

- Livello di copertura di test eseguiti rispetto a quelli previsti;
- Numero totale di errori rilevati durante l'intera fase di collaudo e numero medio di errori rilevati per caso di test;
- Tempo medio di risoluzione degli errori (per livello di gravità);
- Rispetto dei tempi di completamento delle azioni concordate a fronte di eventuali criticità rilevate.

Evidenze prodotte e mantenute

Il processo prevede la produzione e conservazione delle seguenti evidenze:

- Approvazione del Piano di collaudo;
- Verbale di collaudo.

8.4 Gestione dei difetti

La gestione dei difetti rilevati durante l'esecuzione dei test è generalmente definita a livello di processo di test ed è seguita da tutti i progetti (a meno di richieste specifiche del cliente che andranno comunque concordate, documentate e conservate come evidenze). Ogni progetto ne fa menzione nel proprio documento di piano di test. Eventuali deviazioni saranno chiaramente descritte e quindi approvate insieme al piano. L'attività include la registrazione dei difetti, la loro segnalazione, la correzione degli errori, la verifica della correzione, la gestione dello stato delle segnalazioni, l'analisi statistica dei difetti.

8.4.1 Propagazione degli errori

La teoria della propagazione degli errori nel software è stata presentata per la prima volta nel 1981 sulla rivista specializzata *IBM System Journal* pubblicata dall'*IBM System Scientific Institute*. La teoria è rappresentata graficamente in maniera sintetica nella figura che segue.

Secondo la teoria, parte degli errori provenienti dalla fase precedente sono ereditati nella fase attuale così come sono, mentre altri sono amplificati con fattore che dipende dalla tipologia dell'errore, dalla complessità del software e dalla fase. Il fattore di amplificazione può assumere anche valori elevati (es.: 5 o 7). Dal modello rappresentato nella figura si evince chiaramente che i due fattori su cui occorre intervenire per ridurre drasticamente il numero totale di errori presenti nel software (e trasmessi alla fase successiva) sono: riduzione del numero di errori provenienti dalla fase precedente e aumento del numero di errori rilevati e rimossi nella fase attuale.

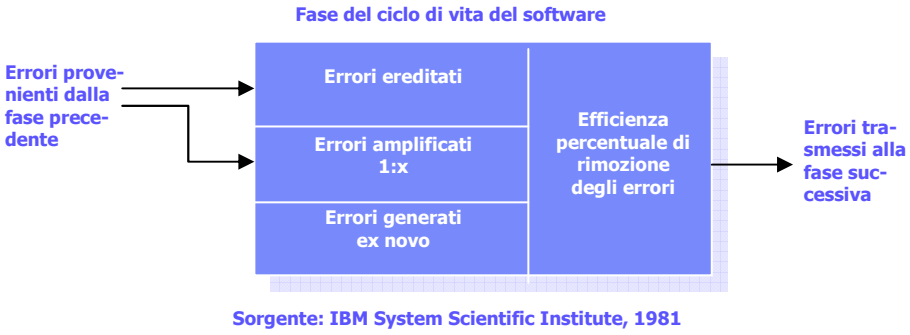

Figura 9. Teoria della propagazione degli errori nel software.

L'attività di "revisione tecnica" eseguita nelle fasi alte del processo di sviluppo è cruciale per migliorare la qualità del software e ridurre i costi relativi alla correzione degli errori rilevati durante il testing. L'utilizzo della tabella per la classificazione ortogonale dei difetti e la contemporanea attività di analisi degli errori più frequenti (Pareto) completa l'attività di controllo della qualità e dei costi. Dati statistici su progetti reali dimostrano una riduzione dei costi complessivi di rimozione dei difetti del 75%.

8.4.2 Alcune definizioni

Partiamo con alcune definizioni tratte dalla letteratura in modo da stabilire un comune linguaggio.

Malfunzionamento (*Failure*)

Un malfunzionamento è il comportamento del software difforme dai requisiti espliciti o impliciti. In pratica, si verifica quando nell'ambiente di test (hardware + software) il sistema non si comporta come ci si attende che faccia.

Anomalia (*Failure*)

E' utilizzato come sinonimo di "malfunzionamento" ed ha due definizioni: (1) Una condizione accidentale che provoca il malfunzionamento di un componente del prodotto nell'eseguire una sua funzione; (2) La manifestazione di un errore nel software.

Difetto (*Defect*)

E' una sequenza di istruzioni, sorgenti o eseguibili, che genera un malfunzionamento quando eseguita con particolari dati in input. In pratica, si ha un malfunzionamento solo quando viene eseguito il codice che contiene il difetto, e solo se i dati di input sono tali da evidenziare l'errore.

Errore (*Error, Bug*)

E' utilizzato come sinonimo di "difetto" ed ha due definizioni: (1) La discrepanza tra il risultato ottenuto dall'esecuzione di una funzione del software ed il risultato atteso, come previsto dalle specifiche; (2) Un'azione umana non corretta che provochi il malfunzionamento del software (errore dell'utente nell'utilizzo del prodotto).

Gravità (*Severity*)

(1) Livello di disagio procurato da un'anomalia verificatasi nel software; (2) Danno procurato dal verificarsi di un'anomalia nel software. A seconda del disagio o danno procurato la gravità può essere "alta", "media" o "bassa". In base alla gravità generalmente si utilizza classificare un'anomalia in "bloccante", "grave o non bloccante", "lieve o marginale".

8.4.3 Classificazione dei difetti

Le anomalie (errori o presunti tali) sono classificati in tre categorie distinte in base alla gravità attribuita ad esse come riassunto nella tabella che segue.

Tabella 1. Classificazione degli errori.

Gravità	Descrizione
Bloccante	Sono quegli errori che impediscono di proseguire qualsiasi altro caso di test in quanto il sistema risulta bloccato in tutte le sue funzioni primarie.
Grave	Sono quegli errori che bloccano la funzione oggetto del test in corso ma non impediscono di testare altre funzioni del prodotto.
Lieve	Sono quegli errori che non impediscono di proseguire i test in corso.

8.4.4 Procedura per la gestione dei difetti

Di seguito è riportata la procedura proposta per la gestione delle anomalie con individuazione dei ruoli coinvolti e attività assegnate.

Figura 10. Procedura di gestione dei difetti.

Attore	Attività
Tester	▪ Registra il malfunzionamento assegnando la gavità che ritiene più adatta al caso in oggetto; ▪ Informa il programmatore responsabile di correggere il difetto/errore (nel caso in cui non si utilizzi uno strumento informatico la modifica è automatica).
Programmatore	▪ Analizza il malfunzionamento, concorda la gravità assegnata (o negozia una più adatta secondo il proprio punto di vista); ▪ Accetta o rigetta il malfunzionamento a seconda che esso sia stato causato da un errore nel software, oppure sia stato originato dall'uso non corretto del prodotto da parte del tester, oppure sia la duplicazione di uno già rilevato e registrato; ▪ Identifica l'errore che ha causato il malfunzionamento e la correzione da apportare; ▪ Corregge il software e valida la correzione eseguendo dei test; ▪ Rende disponibile la nuova versione dei programmi modificati nella libreria dei programmi consolidati e notifica il test. **Nota**: Nel caso in cui il malfunzionamento sia stato generato da un errore concettuale (di disegno), è avvertito l'analista che avrà cura di trovare la soluzione logica dell'errore, modificare le specifiche funzionali ed il disegno, modificare/aggiungere i casi di test. Quindi il programmatore provvederà a correggere il software. In caso di urgenza o di particolare complessità, si provvede a fornire una soluzione provvisoria (*bypass*) che permetta al test di proseguire le attività; quindi si pianifica la correzione definitiva.
Tester	▪ Installa i programmi modificati nell'ambiente di test (l'attività dovrebbe essere svolta dal responsabile della configurazione, quando questa figura è presente nell'organizzazione); ▪ Verifica la correzione completando il caso di test interrotto dal malfunzionamento individuato ed effettua, se necessario, un test di regressione eseguendo altri casi di test complementari; ▪ In caso di completamento positivo dei casi di test eseguiti, il test chiude l'anomalia validandone la soluzione realizzata; ▪ In caso il malfunzionamento dovesse persistere si notifica lo sviluppo rassegnando la stessa segnalazione senza aprirne una nuova.

La gestione delle anomalie è generalmente svolta tramite l'utilizzo di uno strumento (tool). Sul mercato sono disponibili molti strumenti per lo scopo, alcuni anche come "freeware". L'utilizzo di una procedura per la gestione dei difetti e di uno strumento automatico è considerata una best practice di valore.

In caso di gestione manuale dei difetti, in Appendice è riportato un modello di scheda per la registrazione delle anomalie e relativa gestione degli stati.

8.4.5 Flusso di gestione dei difetti

La presenza di più stati dell'anomalia implica la definizione di un flusso come quello mostrato nella **Figura 10** (esempio generale).

La gestione degli stati è facilitata dall'utilizzo di uno strumento informatico.

La gestione manuale risulta comunque semplice e richiede l'utilizzo di un modello (scheda) per la registrazione dei difetti e la relativa risoluzione. In Appendice è riportato un esempio di modello da utilizzare per la registrazione manuale dei difetti e la relativa gestione dello stato di risoluzione, inclusi i rapporti periodici da produrre.

Segue la descrizione del flusso:

1. Apertura del problema.

   Colui che riscontra il problema (generalmente il tester) registra l'anomalia con lo stato "Aperto" e riporta le informazioni di base necessarie a gestire il problema (data e ora, nominativo, descrizione del difetto rilevato, tipo di attività svolta, gravità, fase di test, caso di test che ha rilevato il malfunzionamento, altre informazioni utili quali, ad esempio, documentazione disponibile, ecc.).
2. Notifica del problema.

   L'anomalia è notificata al responsabile della risoluzione dei problemi (generalmente il coordinatore del gruppo di sviluppo). La notifica è automatica quando si utilizzi uno strumento informatico, altrimenti dovrà essere eseguita manualmente. In ogni caso è registrato il tempo (data e ora) della notifica ai fini della rilevazione del tempo di risoluzione dei problemi.
3. Assegnazione del problema.

   Il responsabile della risoluzione dei difetti analizza il problema per verificarne la validità e può proseguire su due diverse linee di condotta:

   a) Accettazione del problema.

      Se il problema è ritento "valido", lo si accetta come tale e lo si assegna ad un membro del team di sviluppo per la sua risoluzione. Questi lo analizza nei dettagli, individua la soluzione più opportuna, individua i componenti da modificare, effettua le correzione, valida le correzioni eseguendo i casi di test opportuni, aggiorna i campi relativi alla risoluzione del problema ed aggiorna lo stato del problema come "Risolto"; i dati relativi alla risoluzione del problema (data e ora, tipo di problema identificato, tipo di soluzione adottata, elenco dei componenti modificati, tipo di test effettuato, risultato dei test effettuati) sono memorizzati per consentire l'analisi statistica dei problemi e poter intervenire per il miglioramento dei processi..

   b) Rifiuto del problema.

Se il problema è ritenuto "non valido", lo si rifiuta (in gergo, lo si "rigetta") e notifica di ciò chi ha aperto il problema. Costui, se d'accordo, pone l'anomalia in stato di "Chiuso" segnalando che il problema è stato "rigettato"; altrimenti innesca un processo di negoziazione per trovare un accordo. Anche in questo caso si registrano le informazioni relative allo stato (data e ora, motivazione del rifiuto, accettazione del rifiuto).

4. Disponibilità della soluzione.

   Lo sviluppo notifica il tester della disponibilità della soluzione marcando il problema come "Risolto". Anche in questo caso si registra la data e l'ora della notifica.

5. Validazione della soluzione.

   Il tester riesegue il caso di test che ha rilevato l'anomalia per verificare la correttezza della soluzione. A seconda dell'esito della verifica, può quindi proseguire secondo due diverse linee di condotta:

   a) Accettazione della soluzione.

      Se l'esito del test è positivo, accetta la soluzione e pone l'anomalia nello stato "Validato". Il responsabile dei test deciderà in seguito di passare lo stato dell'anomalia come "Chiuso". Anche in questo caso sarà registrata la data e l'ora del passaggio di stato.

   b) Rifiuto della soluzione.

      Se l'esito dei test è negativo, si rinvia il problema al gruppo di sviluppo per la risoluzione definitiva. In questo caso l'anomalia, dallo stato "Risolto" in cui l'aveva posta il programmatore ritorna allo stato di "Assegnato", come lo era prima della sua risoluzione. Anche in questo caso si registra la data e l'ora del passaggio di stato con un commento sulla motivazione del rifiuto. Quindi si ritorna al punto 3 del flusso, opzione a). Sarà infatti improbabile che si possa verificare l'opzione b).

Lo stato "chiuso" è quindi indotto da due diverse situazioni: "*Problema rigettato*" (perché non valido o duplicato) o "*Test con esito positivo*".

Come si vede dalla descrizione del flusso, ogni cambiamento di stato è registrato in termini di data e ora e informazioni che ne consentono la gestione.

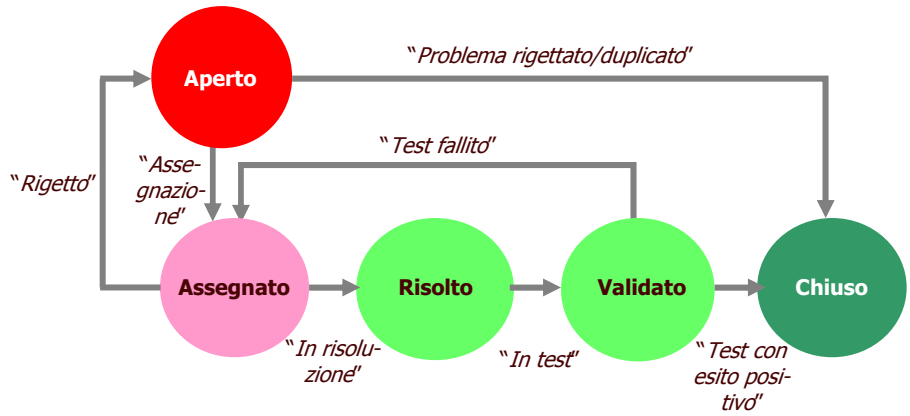

Figura 11. Flusso degli stati dell'anomalia.

8.4.6 Analisi statistica

La registrazione dei momenti di passaggio di stato permettono di calcolare i tempi di gestione dei problemi secondo i seguenti punti di vista:

Lato gruppo di test:

- Tempo medio di notifica dei problemi;
- Tempo medio di validazione della soluzione;

Lato gruppo di sviluppo:

- Tempo medio di risoluzione del problema;

Lato project management:

- Tempo medio di gestione dei problemi.

In quest'ottica valgono le seguenti definizioni.

Tempo medio di notifica dei problemi. Misura il tempo che intercorre tra l'apertura dell'anomalia e la sua notifica al gruppo di sviluppo. Permette di valutare la tempestività con la quale il gruppo di test notifica i problemi al gruppo di sviluppo per consentirne la risoluzione nel più breve tempo possibile.

Tempo medio di risoluzione del problema. Misura il tempo che intercorre tra la notifica del problema e la sua risoluzione, incluso il test effettuato da parte del gruppo di sviluppo. Permette di valutare l'efficienza del gruppo di sviluppo nel risolvere i problemi nel più breve tempo possibile. Il tempo di risoluzione dipende ovviamente dalla gravità dei problemi. Per cui la rilevazione di tale misure è fatta in base alla gravità assegnata ai problemi.

Tempo medio di validazione della soluzione. Misura il tempo che intercorre tra la notifica fatta dal gruppo di sviluppo della disponibilità della soluzione ed il completamento della verifica della sua correttezza fatta dal gruppo di test. Permette di valutare l'efficienza del gruppo di test nel valicare le soluzioni proposte.

Tempo medio di gestione dei problemi. Misura il tempo medio della risoluzione dei problemi calcolato come media dei tempi di apertura e di chiusura dei problemi. Permette di valutare l'efficienza generale del progetto (test + sviluppo) nella gestione dei problemi (individuazione + risoluzione + validazione).

La **Figura 11** mostra la rilevazione dei tempi nei vari momenti in cui l'anomalia cambia di stato (lo stato è indicato tra parentesi tra "virgolette"). L'utilizzo di uno strumento informatico permette una gestione automatica di tali misurazione e la produzione altrettanto automatica di rapporti statistici per la valutazione dell'efficacia del team di progetto.

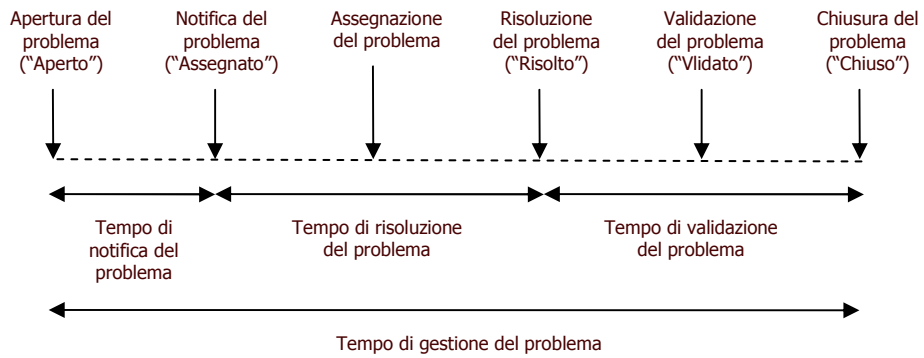

Figura 12. Rilevazione dei tempi di gestione delle anomalie.

La registrazione delle anomalie permette quindi di gestire la qualità del software prodotto e la capacità di risoluzione degli errori. Il rapporto periodico (*Test Report*) fornisce i valori sul numero dei difetti rilevati, degli errori risolti e di quelli in attesa di essere risolti. Un progressivo accumulo di difetti da analizzare e risolvere (*backlog*) indica, per esempio, che il gruppo di sviluppo non è in grado di far fronte al processo di rimozione degli errori. Analogamente un eccessivo accumulo di errori risolti e non ancora validati indica l'inadeguatezza (in termini di prestazioni) del gruppo di test.

Inoltre, la registrazione dei difetti permette di effettuare la l'analisi delle cause degli errori e di migliorare il processo di sviluppo.

Indicando la tipologia di errore corretto (per esempio, errore imputabile al codice, alla progettazione, ai dati, alle specifiche, ai requisiti, alla documentazione utente, al caso di test eseguito, all'ambiente, ecc.) è possibile creare delle statistiche sulle cause più fre-

quenti dei difetti ed intervenire sulle attività di sviluppo (migliorare la fase di analisi e progettazione, potenziare le ispezioni, imporre standard di programmazione, ecc.).

L'**analisi causale** dei difetti permette quindi il miglioramento del processo e si basa sulla disponibilità dei dati registrati.

La **Figura 13** mostra la rappresentazione grafica del tasso di errori per tipologia.

La **Tabella 3** riporta invece i valori percentuali delle tipologie di errori più frequenti.

8.5 Monitoraggio e reportistica dei test

L'esecuzione dei test è monitorata di continuo per verificare il completamento dei casi di prova ed il relativo esito, controllare la correzione degli errori e verificare la rimozione dei difetti rilevati. I risultati dei test eseguiti, lo stato di completamento delle prove e quello di risoluzione degli errori sono riportati in appositi "rapporti" periodici (*Test Report*). Essi riportano le informazioni sulle revisioni tecniche effettuate e sui test eseguiti.

Test statici

Il rapporto contiene informazioni sull'efficacia delle revisioni tecniche effettuate nelle fasi alte del ciclo di vita e sulla qualità della documentazione tecnica prodotta. In particolare, contiene informazioni su:

- *Revisioni tecniche*: numero di revisioni pianificate, completate, in corso, ancora da eseguire, da rieseguire;
- *Errori rilevati*: numero totale di errori rilevati, risolti, da risolvere, rigettati;
- *Efficacia rimozione errori*: numero di errori rilevati per fase di sviluppo (Analisi, Disegno, Codifica, Preparazione dei test, Documentazione utente).

Test dinamici

Il rapporto contiene informazioni sull'efficacia dei test effettuati e sulla qualità del software prodotto. In particolare, sull'andamento dei test eseguiti contengono informazioni su:

- *Casi di test*: numero di casi di test totali previsti, completati, in esecuzione, bloccati, ancora da eseguire;
- *Errori rilevati*: numero totale di errori rilevati (aperti), accettati/rigettati, corretti, validati (testati) e chiusi; il numero di errori nei vari stati è fornito anche per gravità (bloccanti, gravi, minore);
- *Efficacia rimozione errori*: numero di errori rilevati per fase di test e per tipo di gravità (bloccante, grave, minore);
- *Curva di saturazione errori*: andamento nel tempo del numero cumulativo di errori rilevati con evidenza dei punti di flesso e previsione del punto di saturazione.

I risultati del monitoraggio sono discussi nelle riunioni di controllo periodico dello stato di avanzamento del progetto e permettono di valutare lo stato di completamento dei test rispetto ai piani e l'efficacia del collaudo (numero di errori rilevati rispetto ai piani).

Nel paragrafo "Metodi, tecniche e strumenti di test" sono forniti alcuni modelli di riferimento per la produzione della reportistica e la rappresentazione grafica della curva di saturazione degli errori rilevati durante una fase di test.

8.6 Gestione del processo di test

Include l'insieme delle attività gestionali del capo progetto, della direzione e del responsabile della qualità, quando sia previsto in azienda, con l'obiettivo di assicurare che le attività di test siano svolte in accordo con il processo di test stabilito (per esempio, secondo quanto descritto in questa metodologia).

La gestione del processo di test comprende l'esecuzione delle seguenti attività di controllo e validazione previste dal processo stesso.

Test statici:

- Approvazione del Piano delle revisioni tecniche;
- Controllo periodico (fine di ciascuna fase di sviluppo) dello stato di completamento delle revisioni tecniche (revisioni completate, in corso, da eseguire) e degli errori rilevati (errori trovati, corretti, da correggere, testati, rigettati);
- Valutazione finale dei risultati delle revisioni tecniche effettuate e azioni necessarie per indirizzare eventuali situazioni critiche.

Test dinamici:

- Approvazione dei Piani di test;
- Validazione dell'ambiente di test;
- Revisione periodica (giornaliera/settimanale) dello stato di avanzamento dei test e della rimozione degli errori;
- Valutazione finale dei risultati dei test eseguiti e azioni necessarie per indirizzare eventuali situazioni critiche.

L'utilizzo dei modelli predisposti, o simili a quelli proposti in questa metodologia, aiuta il responsabile a seguire il processo definito. Il numero di controlli e la tempistica dipende dalle dimensioni e dalla criticità dell'applicativo sviluppato.

Per progetti medio-piccoli la cui durata dei test è di qualche settimana, è sufficiente effettuare un controllo formale di processo ogni settimana. Inizialmente si verifica l'approvazione del piano delle revisioni tecniche e quello di test. Al termine dei test si

effettua la valutazione dei risultati finali. Il controllo dello stato di completamento dei casi di test e di correzione degli errori è fatta invece giornalmente dal responsabile dei test con le persone del gruppo di test. I controlli sono generalmente informali (senza cioè produzione di un rapporto formale da distribuire agli interessati.

Per progetti più complessi e/o di durata superiore alle due settimane si effettueranno controlli intermedi sullo stato di avanzamento dei test con frequenza settimanale. Resta comunque inteso che il responsabile delle attività di test effettuerà un controllo giornaliero sullo stato dei test tramite incontri con i relativi coordinatori dei gruppi di test.

8.7 Gestione della configurazione di test

Perché i test producano un risultato attendibile (verifica e validazione della qualità del software realizzato e sua rispondenza ai requisiti dichiarati), è necessario che si conosca con precisione "cosa" si stia collaudato. Occorre, cioè, avere certezza che la configurazione sotto test sia quella corretta. La configurazione di test consiste nella definizione esatta sia dei componenti dell'ambiente di test, sia del software applicativo collaudato. I risultati dei test sono quindi sempre riferiti ad un'esatta configurazione (altre configurazioni potrebbero infatti fornire risultati diversi quando sottoposti agli stessi test).

La configurazione dell' "ambiente di test" descrive la sua composizione in termini di hardware, software di base, prodotti, tool e banche dati, tutti con gli attuali livelli di versione utilizzati.

La configurazione del "software applicativo" collaudato descrive invece la sua composizione (componenti integrati) e le relative versioni.

Ogni modifica a tali configurazioni deve essere registrata e valutata in termini di pianificazione dei test e di risultati attesi per le rispettive esecuzioni.

L'ambiente di test deve essere "validato[16]" prima di iniziare l'esecuzione dei test per garantire che esso non influenzi i risultati (l'ambiente di test deve risultare "trasparente" rispetto ai risultati). Eventuali modifiche alla configurazione iniziale dell'ambiente di test devono essere analizzate per verificare eventuali impatti sui risultati dei test pianificati e/o eseguiti. Se non sono evidenziati impatti negativi (i risultati dei test non sono influenzati dalle modifiche all'ambiente), si può proseguire nell'attività di collaudo secondo i piani. Se invece le modifiche alla configurazione dell'ambiente possono alterare i risultati attesi, allora occorre intervenire nei piani, modificare i casi di test e rischedulare

[16] Come già detto in altra parte del manuale, la norma ISO 9001:2000 (come d'altronde anche le versioni precedenti della norma) richiede che l'ambiente di collaudo sia validato prima di iniziare le prove per assicurare che esso non influenzi i risultati dei test.

l'esecuzione. Prima di proseguire con l'attività di test occorre "validare" nuovamente l'ambiente di test.

Le modifiche al software applicativo, sono gestite ugualmente tramite la configurazione software. Ogni modifica alla configurazione attuale del software richiederà che vengano rieseguiti i test già effettuati sugli stessi componenti. Alcune modifiche richiederanno la progettazione di nuovi casi di test.

La tematica discussa è oggetto della disciplina relativa alla gestione delle modifiche e della configurazione (*Change and Configuration Management*).

E' buona norma (best practice) gestire la configurazione in modo corretto utilizzando uno tra i vari strumenti disponibili per i diversi ambienti tecnologici.

La gestione delle modifiche richiede un processo definito. Il processo DCR (*Design Change Request*) è un possibile processo che permette di controllare le modifiche e gli impatti sulla configurazione del software. Tale pratica costituisce a sua volta un "best practice". Il processo DCR prevede che, a fronte di una richiesta/necessità di modifica, sia emesso un rapporto ufficiale che descriva in dettaglio la modifica, i razionali, gli impatti sul software attuale, i costi ed i tempi. Ogni modifica alla configurazione attuale deve essere approvata (nessuno può modificare la configurazione senza la docuta autorizzazione!).

Le informazioni minime richieste dal processo DCR sono:

- Identificazione della richiesta di modifica (nome del prodotto e versione per i quali è richiesta la modifica, nome del richiedente, data della richiesta, priorità/urgenza della richiesta);
- Descrizione della modifica e razionale;
- Impatto della modifica in termini di elenco di componenti da modificare e/o realizzare ex novo (moduli, documentazione tecnica, documentazione utente, procedure, ecc.);
- Costo delle modifiche (giorni-persona per ciascuna attività: analisi e disegno, codifica e test unitario, test d'integrazione, test di sistema);
- Piano di realizzazione delle modifiche (attività, durata e numero di risorse);
- Potenziale rischio legato alla realizzazione delle modifiche ed alla non realizzazione delle stesse;
- Approvazione/non approvazione della richiesta;
- Note (eventuali) di chiarimento delle decisioni prese.

In Appendice è riportato un modello di DCR per la gestione manuale degli stessi. L'utilizzo di uno strumento informatico faciliterà la gestione del processo.

9 Metodi e tecniche di testing

In questo capitolo sono elencate e descritti brevemente i metodi e le tecniche menzionati nel manuale. Se ne suggerisce l'utilizzo in quanto sperimentate con successo in vaie occasioni. Ogni realtà avrà cura di adattarle alle proprie necessità, ma la validità rimane immutata.

Tra parentesi sono riportati i nomi originali in inglese per consentire a chi le conoscesse già di farvi riferimento. L'ordine è per categoria, senza alcuna priorità. I modelli (template) sono presentati nella loro forma completa, come suggerito dagli standard vari. Le necessità specifiche di ciascuno permetteranno di modificarli opportunamente.

Si spera solo che risultino di qualche utilità pratica alla maggior parte dei lettori.

Modello della qualità del software:

- ISO/IEC 9126.

Tecniche per la progettazione dei casi di test:

- Condizioni generali;
- Transazioni on-line;
- Transazioni batch;
- Dati invalidi (tutti i tipi di transazioni);
- Interfacce esterne.

Tecniche per l'esecuzione del test:

- Revisione strutturata (*Peer Review*);
- Test a scatola aperta (*White-Box Test*);
- Test a scatola chiusa (*Black-Box Test*);
- Test a previsione di errori (*Error Guessing Test*);

Tecniche d'integrazione:

- Test gerarchico (*Top-Down Test*);
- Test subordinato (*Bottom-Up Test*);
- Combinazione delle due tecniche (*Top-Down* e *Bottom-Up*).

Tecniche di controllo della rimozione degli errori:

- Profilo di qualità del prodotto (*Quality Profile*);
- Classificazione ortogonale dei difetti (*Orthogonal Defect Classification*);
- Curva di rimozione degli errori (*Defect Removal Curve*).

9.1 Modello della qualità del software ISO/IEC 9126

Le norme ISO 9126 definiscono le caratteristiche e gli attributi del software[17].

9.1.1 Funzionalità

Definisce la ricchezza delle capacità funzionali del software nel soddisfare i requisiti, espliciti ed impliciti. "Il software fa quello che ci si aspetta debba fare".

I suoi attributi principali da prendere in considerazione sono:

- Completezza (delle funzioni offerte verso i requisiti);
- Accuratezza (con la quale le funzioni sono offerte);
- Interoperabilità (con gli altri sistemi presenti nell'ambiente nel quale la soluzione opererà);
- Aderenza (a normative, leggi, regole, standard, ecc.);
- Sicurezza (dei dati e delle persone).

9.1.2 Affidabilità

Definisce la capacità del software di mantenere i livelli di prestazione stabiliti nell'ambito di condizioni definite e per una durata stabilita. "Il software reagisce positivamente alle variazioni dell'ambiente circostante".

I suoi attributi principali da prendere in considerazione sono:

- Maturità (livello di assestamento del sistema);
- Tolleranza ai guasti (margini entro i quali le prestazioni sono fornite);
- Recuperabilità (capacità di ripristinare la situazione originale in caso di caduta o degrado).

9.1.3 Usabilità

Definisce l'impegno richiesto per usare il software da parte degli utenti stabiliti. "Il software gestisce l'interazione con gli utenti in modo ottimale" .

I suoi attributi principali da prendere in considerazione sono:

- Comprensibilità (facilità di capire e intuire le funzionalità disponibili nel sistema);

[17] Maggiori dettagli sul modello e sulla sua applicazione pratica nei progetti di sviluppo software sono forniti in un apposito documento dello stesso autore (La qualità del software secondo il modello ISO/IEC 9126) disponibile nel suo sito (www.colonese.it/pubblicazioni).

- Apprendibilità (facilità di imparare e ricordare le funzionalità offerte);
- Operabilità (facilità di adoperare le funzioni rese disponibili).

9.1.4 Efficienza

Definisce la capacità di erogare le funzioni offerte con il minor uso di risorse impiegate, in condizioni di funzionamento stabilite. Spesso è detta "performance". "Il software usa bene le risorse disponibili" .

I suoi attributi principali da prendere in considerazione sono:

- Rispetto al tempo (tempo di risposta);
- Rispetto alle risorse (utilizzo di risorse).

9.1.5 Manutenibilità

Definisce la facilità con cui è possibile eseguire modifiche al software esistente intese come evoluzioni, correzioni, adattamenti, ristrutturazioni, ecc. "Il software segue l'evoluzione dell'organizzazione" .

I suoi attributi principali da prendere in considerazione sono:

- Analizzabilità (facilità a diagnosticare i problemi, detta anche "problem determination");
- Modificabilità (facilità ad inserire nel software esistente le modifiche richieste);
- Stabilità (capacità del software esistente a rimanere stabile anche dopo le modifiche);
- Provabilità (facilità di eseguire test alle modifiche coinvolgendo poco le funzioni esistenti).

9.1.6 Portabilità

Definisce la facilità con la quale il software può essere portato da una piattaforma ad un'altra. "Il software segue l'evoluzione tecnologica" .

I suoi attributi principali da prendere in considerazione sono:

- Adattabilità (al nuovo sistema);
- Installabilità (facilità di installazione nel nuovo sistema) ;
- Conformità (alle regole del nuovo sistema);
- Sostituibilità.

9.2 Tecniche di progettazione dei casi di test

Di seguito sono riportati alcuni elementi importanti da prendere in considerazione quando si progettano i casi di test delle transazioni. Le considerazioni fanno riferimento al tipo di elaborazione (batch o on-line) ed al livello di testing (test unitario, di integrazione, di sistema, di accettazione). Si ritiene anche che:

- siano eseguite "Ispezioni" per verificare la completezza e la consistenza dei prodotti;
- si utilizzino "Strumenti di automazione" per la gestione, l'esecuzione, la regressione e l'operatività dei test.

Le condizioni di test da verificare si riferiscono alle seguenti tipologie:

- Condizioni generali;
- Transazioni on-line;
- Transazioni batch;
- Dati invalidi (tutti i tipi di transazioni);
- Interfacce esterne.

9.2.1 Condizioni generali

Le condizioni generali da verificare riguardano l'inserimento dei record, il loro aggiornamento, la cancellazione e la chiusura. Si tratta quindi di progettare casi di test per ciascuna di queste categorie.

Aggiunta/Inserimento di record:

- Aggiungere/inserire un nuovo record con dati validi;
- Aggiungere/inserire un record esistente.

Aggiornamento di record:

- Aggiornare un record con dati validi;
- Aggiornare un record che non esiste;
- Aggiornare un record chiuso;
- Aggiornare più volte lo stesso record.

Chiusura/Cancellazione di record:

- Chiudere (flag di cancellazione o spostamento ad un archivio "history") un record con dati validi;
- Ri-chiudere un record già chiuso;
- Chiudere un record che non esiste.

Tutti i record:

- Aggiungere, modificare e chiudere un record nello stesso caso di test;
- Utilizzare tutti i possibili metodi di inserimento dei dati.

9.2.2 Transazioni on-line

Per quanto attiene le transazioni on-line si tratta di verificare il comportamento a fronte dell'inserimento di valori nulli e del tentativo di interromperle. In particolare, si progettano casi di test per:

- Inserire dati "nulli" in tutte le transazioni;
- Tentare di interrompere la transazione senza che si possa completare.

9.2.3 Transazioni batch

Per le transazioni batch si tratta di progettare casi di test che ne verifichino il coretto comportamento a fronte di situazioni errate o imprevedibili. Di seguito un possibile elenco di condizioni da verificare.

- Provare sia transazioni singole che combinazioni di più transazioni;
- Sottomettere transazioni già eseguite;
- Provare combinazioni di dati validi e dati invalidi ed errori multipli;
- Sottomettere transazioni con sequenze di errori;
- Sottomettere processi batch con date errate e/o duplicate, testate errate e/o omesse, totali incorretti;
- Eseguire transazioni senza dati;
- Eseguire i dati relativi a due o più giorni in una unica esecuzione ("run").

9.2.4 Dati invalidi (Tutte i tipi di transazione)

La verifica del comportamento delle transazioni a fronte dell'inserimento di valori non corretti, non previsti o improbabili è molto importante e può comprendere le seguenti condizioni di test:

- Testare tutti i tipi di transazioni utilizzando dati validi ed invalidi;
- Impostare i record di test in modo da ottenere risultati con valori alti, bassi, zero, negativi e poi positivi;
- Provare transazioni a fronte di record chiusi e/o non esistenti;
- Verificare estensioni di valori (range) ampi e stretti, dentro e fuori dei limiti;
- Verificare i campi aritmetici che superano i limiti consentiti ("overflow"), che assumono valori negativi, troncati, divisi da zero, allineamento;

- Provare campi numerici con valori alfanumerici, sostituire valori validi con valori nulli (blank), utilizzare valori nulli (blank) per riempire gli spazi vuoti, inizializzare i campi;
- Provare i campi date con valori errati (giorni, mesi, anni);
- Provare caratteri speciali o "tasti chiave" (*, ?, ½, ¼, EOF, ecc.).

9.2.5 Interfacce esterne

I test delle interfacce esterne dell'applicazione sono importanti quanto numerosi. Di seguito sono proposte alcune condizioni di test fondamentali.

- Verificare il flusso dei pannelli (schermate) nell'eseguire le transazioni;
- Testare schermate che presentano un numero variabile di linee di output con un numero di linee pari a zero, al valore minimo, al valore minimo – 1, al valore massimo, al valore massimo + 1 (per esempio, una schermata che presenta un ordine con un numero variabile massimo di 20 elementi è provato con zero elementi, 19, 20 e 21 elementi);
- Provare che tutti i tasti funzionali previsti siano coretti in tutte le schermate;
- Verificare la consistenza in tutte le schermate dell'utilizzo dei formati, dei colori, delle posizioni dei campi, dei tasti funzionali, ecc.
- Verificare la presenza e la consistenza delle schermate di help;
- Verificare il corretto ritorno dalle schermate di help.

9.3 Tecniche per l'esecuzione del test

Le tecniche presentate per l'esecuzione dei test prevedono:

- Revisione strutturata ("Peer Review");
- Tecnica di test "White-Box";
- Tecnica di test "Black-Box" Test;
- Tecnica di test "Error Guessing".

9.3.1 Revisione strutturata ("Peer Review")

Partecipanti

E' una tecnica consolidata di grande efficacia per effettuare revisioni tecniche formali. Sono previsti i seguenti ruoli:

Autore: chi ha realizzato il materiale da sottoporre a revisione.

Moderatore: chi ha conoscenza ed esperienza della tecnica delle revisioni strutturate, oltre a conoscenze applicative e tecniche specifiche sull'argomento trattato. Svolge anche il ruolo di revisore/ispettore.

Revisore/Ispettore: chi ha competenza tecnica per poter eseguire la verifica del materiale revisionato.

Tutti i partecipanti sono considerati alla pari senza gerarchia di ruoli (da cui la denominazione "Peer Review"). L'attività è svolta tra colleghi con lo scopo di aiutarsi a vicenda per verificare la correttezza del materiale prodotto. Ogni collega, a turno, è "revisore" del lavoro di altri ed è anche "revisionato" nel proprio lavoro prodotto. In ottica di aiuto reciproco, è sconsigliata la presenza dei capi.

Modalità di esecuzione

Le modalità per l'organizzazione e la conduzione delle revisioni formali sono le seguenti:

Pianificazione delle revisioni

Il capo progetto pianifica le revisioni formali da eseguire nell'ambito del progetto (quali documenti ispezionare, quali persone coinvolgere come moderatore e revisori, quando effettuare le revisioni, quanto tempo dedicare a tali attività).

Distribuzione del materiale

L'autore distribuisce una copia del materiale da revisionare al moderatore ed ai revisori (in genere una o due persone), in un breve incontro (10 minuti), dove espone i contenuti del materiale fornito. Si concorda la data e l'ora della riunione in cui discutere i commenti.

Revisione del materiale

I revisori ed il moderatore, ognuno per proprio conto, leggono il documento fornito ed apportano sul materiale i propri commenti. L'attenzione è posta sugli eventuali errori, mancanze, particolari importanti e non sulle banalità. Lo scopo è quello di scoprire gli errori prima che si possano propagare nelle fasi successive ed infine nel codice sorgente.

Riunione di revisione

L'autore, il moderatore ed i revisori si riuniscono nel giorno concordato per esporre e commentare gli errori rilevati e le osservazioni fatte sui documenti. E' importante segnalare gli errori riscontrati senza entrare in discussioni circa la loro risoluzione. Sarà l'autore a decidere se e come risolverli. La riunione deve durare il tempo prefissato (in genere un paio d'ore). Se necessario, si pianifica una seconda riunione. Qui il moderatore gioca un ruolo importante. Egli deve garantire che la riunione sia efficace (siano realmente segnalati problemi ed errori), duri il tempo previsto (si vada al sodo e non si perda tempo in discussioni inutili) e sia svolta correttamen-

te (si aiuti l'autore a scoprire gli errori senza giudicare la sua capacità e quanto faccia bene il suo lavoro). Il moderatore produrrà una lista dei problemi segnalati che consegnerà all'autore al termine della riunione.

Correzione del materiale

In un tempo successivo, l'autore analizza i problemi segnalati (elencati nella lista) e provvede a risolverli secondo il proprio giudizio tecnico, compreso quello di non indirizzare un problema se ritenuto improprio.

Verifica delle correzioni

L'autore discute con il moderatore la lista dei problemi segnalati e le eventuali soluzioni apportate, senza entrare in un dettaglio tecnico eccessivo. Il moderatore decide se considerare risolti tutti i problemi e verbalizza la chiusura della revisione.

Chiusura della revisione e stesura del verbale

Il verbale di revisione è redatto utilizzando un modulo che contenga, al minimo, le seguenti informazioni: nome del progetto, titolo del documento ispezionato, data della riunione di revisione e partecipanti, elenco dei problemi riscontrati, data di chiusura della revisione.

9.3.2 Tecnica di test "White-Box"

"White-Box" è una tecnica di test focalizzata sulla struttura interna di un programma. In questo test si ha piena visibilità del codice. Per eseguire il test si deve seguire il percorso logico del programma. Il test white-box è quindi logic driven, cioè basato sulla logica interna del programma. E' un test strutturato, perché è effettuato seguendo la struttura del programma. Per sviluppare i casi di prova relativi, bisogna accedere la codice. E' richiesta, quindi, la conoscenza del linguaggio di programmazione utilizzato. Questi test, detti anche test unitari, sono progettati ed eseguiti dagli stessi sviluppatori che conoscono bene il proprio codice. Il codice è rappresentato mostrando i punti di decisione, le condizioni logiche ed i salti. I casi di test devono essere costruiti per verificare ciascuna condizione logica in tutti i suoi possibili valori (vero / falso). Devono essere verificati tutti i possibili cammini. Per definire quali e quanti test effettuare ci si può aiutare con il grafo del controllo dei flussi, utilizzando la complessità ciclomatica.

Vantaggi

- E' un test focalizzato;
- Verifica il flusso di controllo;
- Verifica gli algoritmi specifici;
- Verifica i confini / i casi limite del modulo (programma);
- E' l'approccio naturale per il test unitario.

Svantaggi

- Non verifica la corrispondenza del programma alle specifiche funzionali;
- Non verifica la soddisfazione dei requisiti del cliente;
- Non verifica i percorsi logici mancanti, ridondanti, inutili (codice inerte);
- Per migliorare i risultati occorre aumentare lo sforzo.

9.3.3 Tecnica di test "Black-Box"

"Black-Box" è una tecnica di test focalizzata sugli aspetti esterni della funzione da validare ed è quindi "data driven". La funzione è vista come una scatola nera dove sono noti solo i dati di input e di output (il codice non è visibile e quindi non è conosciuto il percorso seguito all'interno dei programmi). Non è quindi richiesta la conoscenza del codice interno eseguito. Esso riflette le necessità del business e per costruire i casi di prova è necessario avere conoscenza delle funzioni applicative previste e la relativa importanza rispetto al business.

I casi di prova sono quindi dedotti direttamente dalle specifiche funzionali. Le condizioni per la validazione esaustiva dell'applicazione sono quelle di provare tutte le combinazioni di input valide e provare tutte le concatenazioni logiche di sequenza dei dati. Poiché risulta oneroso eseguire un test esaustivo occorre, quindi, pianificare i test con criteri di economicità (massimo risultato con minimo sforzo) ottimizzando le prove da eseguire. L'efficacia del test Black-Box sta nell'efficacia dei casi di test. Questi dunque saranno progettati perché risultino significativi per il business, adeguati alla complessità delle funzioni da validare e saranno eseguiti da personale con adeguata competenza applicativa. La scelta dei casi di test significativi è effettuata tramite la costruzione di una matrice di test che mappa le funzionalità da validare con i casi di test di eseguire. La tecnica di test Black-Box è l'approccio naturale per i test funzionale (o d'integrazione), i test di sistema ed i collaudi utente (o d'accettazione).

9.3.4 Tecnica di test "Error Guessing" (previsione degli errori)

"Error Guessing" è una tecnica di test che si basa sull'esperienza pregressa, con la quale si possono creare dati di test e casi di test anticipare il rilevamento degli errori più comuni. Usando l'esperienza e le conoscenze applicative, si possono simulare gli errori più comuni che gli utenti finali tendono a commettere nell'utilizzo del prodotto per verificare che il software sia in grado di gestire correttamente tali situazioni.

Un tipico esempio di "Error Guessing" e':

- Utilizzare il tasto "Alt" invece del tasto "Ctrl" in un'applicazione basata su PC e vrificare il comportamento del software.

9.4 Tecniche di integrazione

Le tecniche di integrazione presentate sono quelle più importanti:

- Tecnica d'integrazione "Top-Down";
- Tecnica d'integrazione "Bottom-Up";
- Combinazione delle due tecniche.

9.4.1 Tecnica d'integrazione "Top-Down"

Il test "Top-Down" è complementare all'analisi e disegno Top-Down (o decomposizione funzionale).

I programmi/funzioni di più alto livello sono testate non appena sono stati implementati.

Le funzioni chiamate che ancora non sono state sviluppate sono simulati dagli "scaffolding". In questo modo si procede col testare la logica generale del prodotto. Le funzioni di livello più basso, man mano che si procede con la loro codifica, sono quindi integrate e testate.

Vantaggi

- Le funzioni di più alto livello sono testate prima e meglio;
- L'integrazione incrementale dei moduli/funzioni nel sistema rende più semplice determinare quale modulo ha problemi.

Svantaggi

- Le funzioni di basso livello sono testate di meno.

9.4.2 Tecnica d'integrazione "Bottom-Up"

Il test "Bottom-Up" consiste nel testare prima i moduli di livello più basso man mano che essi sono resi disponibili. Le Funzioni complesse sono combinazione di moduli elementari. I livelli più bassi dei programmi/funzioni sono testati man mano che vengono resi disponibili dal programmatore. I moduli base sono codificati per primi e vengono testati utilizzando i driver che provvedono a chiamare i moduli da testare, fornendo gli opportuni dati (parametri di input). Man mano che si procede con l'implementazione del prodotto, i moduli al livello immediatamente superiore vengono codificati e quindi integrati e testati.

Vantaggi

- Le funzioni di livello più basso non necessitano di essere ritestate dopo l'integrazione, è sufficiente testare la logica delle chiamate;

- Maggiore è il tasso di riuso e più il test Bottom-up risulta efficiente

Svantaggi

- La creazione dei driver potrebbe non essere semplice;
- I test più complessi sono eseguiti per ultimi.

9.4.3 Combinazione delle due tecniche

La combinazione dei due approcci, Top Down e Bottom Up, può risultare particolarmente efficace quando sia possibile realizzarla.

- L'approccio Top Down risulta efficace nelle strategie di sviluppo iterativo incrementale. L'approccio Bottom Up risulta invece efficace quando un numero di moduli comuni sono utilizzati da differenti sistemi o sottosistemi e sono sviluppati prima degli altri. E' questo il caso di un sottosistema che svolge servizi per tutti gli altri sottosistemi (esempio: gestione degli errori, gestione degli input o degli output, gestione delle interfacce con i dati, ecc.).
- La combinazione dei due approcci può dipendere quindi dalle considerazioni menzionate sopra.

9.5 Tecniche di controllo della rimozione degli errori

Le tecniche per il controllo della rimozione degli errori presentate sono:

- Profilo di qualità del prodotto;
- Classificazione ortogonale dei difetti (ODC);
- Curva di rimozione degli errori (nella fase di test);

9.5.1 Profilo di qualità del prodotto

Qualità del software

Il software è il prodotto di un'attività intellettuale creativa e tecnica allo stesso tempo, ad alto contenuto umano e quindi soggetta ad errori. La complessità del progetto, sia dal punto di vista tecnologico che applicativo, costituisce un ulteriore elemento che genera errori. Rimuovere gli errori costa, specialmente nelle fasi finali del ciclo di sviluppo, quando il software è stato già sviluppato; ma ancora di più costa correggere gli errori nel software messo in esercizio!

Inoltre, la teoria della propagazione degli errori descrive come gli errori residui in una fase del ciclo di sviluppo generi ulteriori errori nella fase successiva.

Un modo concreto per aumentare la qualità del software è quindi quello di ridurre il numero di errori immessi e aumentare quello degli errori rimossi!

Immissione degli errori

Il tasso di errori immessi nel codice dipende da fattori culturali, metodologici e tecnologici. I tre elementi principali su cui occorre agire per migliorare la qualità del software sono:

- competenza delle persone
- processi maturi
- metriche, metodi, tecniche e strumenti a supporto.

Un'organizzazione diventa sempre più matura e produce risultati sempre più qualitativi e di successo quanto più fa leva sui tre elementi menzionati in modo sinergico e bilanciato.

Uno strumento di sicuro valore in questo capo è proprio il "profilo della qualità del progetto" che stiamo descrivendo.

Rimozione degli errori

Le due tecniche più efficaci per la rimozione degli errori nel software sono:

1. *Revisione tecnica (ispezione)*: adoperata nelle fasi alte del ciclo di sviluppo, consiste nella revisione dei documenti prodotti con lo scopo di rimuovere gli errori nella stessa fase in cui sono "immessi" ed evitare che si propaghino nelle fasi successive;
2. *Testing*: utilizzata per collaudare il software prodotto con prove di tipo e profondità diverse: unitario, d'integrazione, di sistema, ecc.

Perché risultino efficaci è necessario che tali tecniche siano utilizzate correttamente. Entrambe le tecniche sono state discusse in dettaglio nei rispettivi capitoli. Qui si descrive il modo di controllarne l'efficacia. La competenza delle persone resta comunque la chiave di volta in tutti questi discorsi.

Errori residui

Gli errori rimasti nel software dopo la sua messa in esercizio (rilascio in produzione) creano problemi, a volte anche molto gravi, nell'operatività degli utenti finali. Inoltre, la loro rimozione ha un costo elevato. L'impatto è quindi negativo e fortemente penalizzante sia per il cliente che per il produttore del software. Occorre quindi poter calcolare il tasso di difettosità residua, cioè il numero di errori rimasti nel codice dopo la sua messa in esercizio.

Il "Profilo della difettosità del prodotto" è una tecnica che aiuta a stimare il tasso di difettosità residua.

Profilo della difettosità del prodotto

La tabella che segue mostra un modo molto semplice di costruire il profilo in oggetto.

L'analisi statistica prevede una difettosità residua nel primo periodo di esercizio pari a quella rilevata nell'ultima fase di collaudo. Tale valore di difettosità residua si riferisce ad un periodo di circa 18 mesi a partire dalla data di rilascio (nell'esempio, 2.7 errori per Kloc o per 100FP).

Tabella 2. Profilo di rimozione degli errori nel ciclo di sviluppo.

VALORI	REQUISITI	ANALISI - DISEGNO	CODIFICA – TEST UNITARIO	TEST D'INTEGRAZIONE	TEST DI SISTEMA
Valori attesi	2.2	3.8	13.3	5.1	1.2
Valori effettivi	**1.2**	**2.7**	**21.8**	**8.7**	**4.7**

Ogni colonna rappresenta una fase dell'intero ciclo di sviluppo.

La prima riga riporta i dati relativi al numero di errori che ci si aspetta di rimuove in ogni fase a seguito delle attività pianificate (revisioni tecniche e test). Tali valori sono de-

dotti dall'esperienza di progetti analoghi condotti in ambienti simili (competenza delle persone, processi adottati, tecnologie adoperate, complessità del prodotto ecc.). I valori sono normalizzati, cioè riferiti ad una unità di misura delle dimensioni del prodotto software (nell'esempio si riferiscono a 1KLoc (1000 linee di codice). Un'altra misura potrebbe riferirsi, per esempio, a 100 PF. Risulta evidente che non sia possibile realizzare tali previsioni in caso di mancanza di dati statistici cui fare riferimento. L'archivio statistico dei progetti cui si fa riferimento in un'altra sezione del documento ha proprio questo obiettivo. In tali casi occorre iniziare a registrare i dati di progetto in previsione di un utilizzo futuro (occorre pur iniziare una prima volta!).

La seconda riga riporta i valori effettivi degli errori rimossi in ciascuna fase.

La figura che segue mostra la rappresentazione grafica della curva di difettosità di un prodotto software come rilevata dalla rimozione degli errori nelle varie fasi di sviluppo.

Le barre di colore verde mostrano la rimozione degli errori nelle fasi alte del ciclo di sviluppo e quelle marroni la rimozione dal codice eseguibile; quelle rosse infine indicano gli errori trovati dagli utenti durante l'esercizio.La costruzione di tale curva, prima come andamento statistico costruito sulla base dei dati presenti nell'archivio storico dei progetti, e poi con i dati reali ricavati dalle attività di revisione e di test costituisce una "best practice" di grande utilità come descritto qui i seguito.

Coincidenza dei risultati - Qualora i valori effettivi siano paragonabili a quelli previsti, anche se approssimativamente, siamo nel caso in cui si conferma del modello statistico utilizzato (conferma della bontà dei dati dei progetti presenti nell'archivio).

Risultati inferiori alle attese - Nel caso in cui i valori effettivi siano inferiori a quelli previsti ci sono due possibili cause:

1. *Il software realizzato è qualitativamente migliore di quanto previsto*: gli elementi che contribuiscono a tale risultato sono l'accresciuta competenza delle persone e/o la complessità del lavoro è risultata essere nettamente inferiore a quella stimata. Se dall'analisi dettagliata dei dati si confermano i risultati, siamo di fronte ad un'organizzazione che è cresciuta in termini di efficacia; questo potrebbe essere il risultato di azioni precedentemente messe in opera per conseguire tali miglioramenti (si esclude comunque un miglioramento casuale e spontaneo e non generato da alcuna azione!). L'aggiornamento dei dati dell'archivio storico dei progetti produce si traduce anche nell'aggiornamento dei modelli statistici adoperati, ed i prossimi progetti potranno essere stimati tenendo conto dei miglioramenti realizzati.
2. *Le attività di revisione tecnica e di test sono poco efficaci*: gli elementi che contribuiscono a tale risultato sono la scarsa competenza o impegno delle persone che hanno eseguito le attività e/o la complessità del lavoro che è risultata essere superiore a quella stimata. Se dall'analisi dettagliata dei dati risultano confermati i risultati occorre intervenire su: competenza, efficacia e motivazione delle persone; metodi di con-

duzione delle revisioni tecniche; metodi di progettazione dei test; metodi di controllo dell'esecuzione dei test.

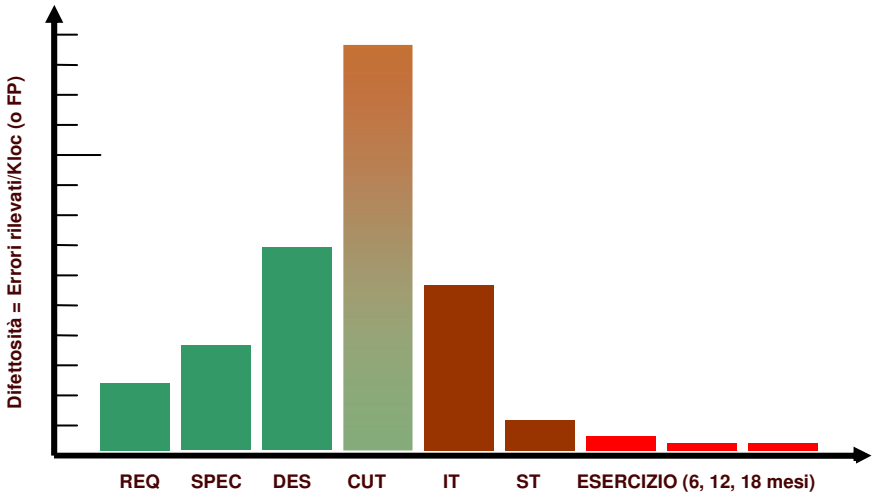

Figura 13. Profilo della difettosità del prodotto software.

Risultati superiori alle attese – Il caso in cui i valori effettivi sono superiori a quelli attesi (caso rappresentato nella tabella precedente) identifica un software di qualità inferiore a quella prevista. Le attività di revisione e di test rilevano un numero di errori superiore alla media dei progetti simili. La conseguenza diretta di una tale situazione è che la difettosità superiore si ripercuote negativamente alle fasi successive con aumento dei costi. L'intervento più opportuno, in questo caso, è quello di intervenire subito nelle prime fasi appena la deviazione è evidenziata. L'analisi approfondita di tale deviazione dai valori attesi dovrà identificare le cause (molto probabile che si tratti di una scarsa qualità delle attività svolte come). Intervenire subito con azioni di recupero (per esempio, la riesecuzione di alcune attività critiche) consente di recuperare e riportare il progetto verso un andamento più vicino a quello atteso e descritto dalla curva.

9.5.2 Classificazione ortogonale dei difetti

Si tratta di costruire una tabella con i difetti rilevati durante le varie fasi del ciclo di sviluppo. La tabella in oggetto (detta *Orthogonal Defect Classification*) è utilizzata per valutare l'efficacia e l'efficienza delle revisioni tecniche. Permette di individuare il profilo di immissione e quello di rimozione degli errori nel software. La tecnica è quella della valutazione dello scostamento reale rispetto alla curva ideale costruita sui dati statistici di progetti analoghi. La valutazione è riferita al tipo di progetto, alla tecnologia adoperata, al processo di sviluppo seguito, alla competenza delle persone, ecc.

In sintesi, la tecnica permette di proiettare la qualità finale del software sviluppato.

La tabella che segue ne mostra un esempio.

Tabella 3. Classificazione ortogonale dei difetti (*Orthogonal Defect Classification*).

		IMMISSIONE DEGLI ERRORI NELLE FASI DI SVILUPPO					
RIMOZIONE DEGLI ERRORI NELLE FASI DISVILUPPO		**Analisi**	**Disegno**	**Codifica**	**Test**	**Totale**	**%**
	Analisi	5	3	2	0	**10**	14%
	Disegno	--	6	6	11	**23**	32%
	Codifica	--	--	2	30	**32**	44%
	Test	--	--	--	7	**7**	10%
	Totale	**5**	**9**	**10**	**48**	**72**	100%
	%	7%	12%	14%	67%	100%	

I valori riportati nella tabella indicano, per esempio, che i 5 errori scoperti in fase di Analisi (tramite attività di revisione) sono tutti imputabili alla fase in oggetto. Dei 9 errori scoperti invece in fase di Progettazione (Design), 3 di essi sono da imputare alla fase precedente di Analisi mentre i restanti 6 sono da imputare all'attività di progettazione. Per finire, dall'analisi dei 48 errori rilevati durante l'esecuzione dei test (nel loro complesso), si scopre che nessuno di essi è da imputare alla fase di Analisi, 11 sono dovuti ad errori commessi in fase di progettazione e 30 ad errori di codifica.

Una prima valutazione porta a determinare che il tasso di "immissione" degli errori nelle diverse fasi di sviluppo è pari a: 10 errori nella fase di Analisi (14% del numero totale), 23 nella fase di Progettazione (32%), 32 nella fase di Codifica (44%) e 7 (10%) nella fase di Test.

Una seconda valutazione porta a determinare l'efficacia dell'attività di "rimozione" degli errori nelle diverse fasi dello sviluppo; essa è espressa come percentuale di errori rimossi rispetto al numero totale: Analisi (7%), Design (12%), Codifica (14%) e Test (67%).

Un ulteriore elemento di analisi è rappresentato dalla classificazione degli errori rimossi. La **Tabella 1** (Tipologia di errori rilevati con maggiore frequenza) riportata all'inizio del documento fornisce una classificazione generalmente utilizzata per lo scopo. Ogni organizzazione software adatterà la classificazione alla propria cultura e mentalità. In ogni caso, l'analisi delle percentuali di errori più frequenti permette di agire sulle aree più critiche e di migliorare le performance dell'organizzazione.

La rappresentazione delle percentuali di errori più frequenti secondo il modello di Pareto[18], per esempio, aiuta a definire le aree su cui intervenire per migliorare la qualità del software e le performance dell'intera organizzazione di sviluppo. I valori dei dati riportati nella figura sono tratti dalla letteratura disponibile in materia e confermati dall'esperienza dell'autore in diversi progetti di sviluppo software[19].

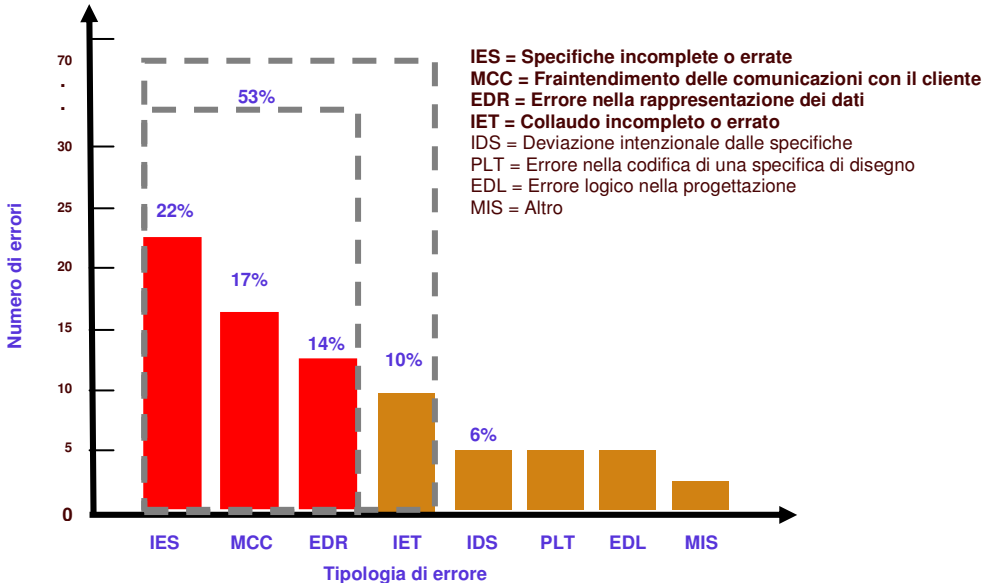

Figura 14. Classificazione dei difetti e rappresentazione secondo Pareto.

L'esempio mostrato di seguito mostra come la risoluzione delle cause che hanno generato i problemi relativi alle specifiche, alla comunicazione con i cliente, rappresentazione dei dati porta ad un abbattimento del 53% del totale degli errori immessi nel software.

La **Tabella 3** riporta i valori percentuali delle diverse tipologie di errori rilevati. I valori dei dati sono quelli riportati sulla letteratura disponibile.

[18] Applicando la teoria di Pareto all'analisi dei difetti, risulta che la maggior parte di essi è da imputare a pochi fattori. Quindi, agendo su una parte contenuta delle cause (30%) si risolve la maggior parte (70%) dei problemi. Per questo motivo la teoria è anche detta "70/30". In alcuni casi la teoria ha mostrato anche una percentuale più accentuata (80/20).

[19] L'autore è stato per molti anni responsabile della qualità dei prodotti software realizzati presso un centro di sviluppo prodotti di una importante azienda informatica.

Tabella 4. Valori percentuali delle tipologie di errori rilevati.

TIPOLOGIA DI ERRORE	%
Specifiche incomplete o errate (IES)	22
Fraintendimento delle comunicazioni del cliente (MCC)	17
Errore nella rappresentazione dei dati (EDR)	14
Collaudo incompleto o errato (IET)	10
Errore di codifica della progettazione (PLT)	6
Inconsistenza dell'interfaccia tra i componenti (ICI)	6
Errore logico nella progettazione (EDL)	5
Deviazione intenzionale dalle specifiche (IDS)	5
Documentazione imprecisa o incompleta (IID)	4
Violazione di standard di programmazione (VPS)	3
Interfaccia uomo-macchina ambigua o inconsistente (HCI)	3
Altro (MIS)	5
Totale	**100**

Le prime quattro tipologie di errore, da sole, costituiscono ben il 63% del totale di errori rilevati. Indirizzare queste tipologie di errore nel processo di sviluppo permette di ridurre drasticamente la difettosità del software e di ridurre i costi di rimozione degli errori.

9.5.3 Curva di rimozione degli errori (nella fase di test)

La rappresentazione del numero cumulativo di errori rilevati nel tempo durante lo svolgimento di una singola fase di test acquista la forma mostrata nella figura che segue.

Al completamento del 50% dei casi di prova eseguiti la curva tende ancora a salire, mentre al completamento del 75% dei casi di prova eseguiti la curva tende ad appiattirsi.

L'asintoto verso cui tende la curva rappresenta il numero massimo di errori che la fase di test è in grado di rilevare.

L'utilizzo del modello statistico ("profilo") consente di prevedere il valore atteso (l'asintoto).

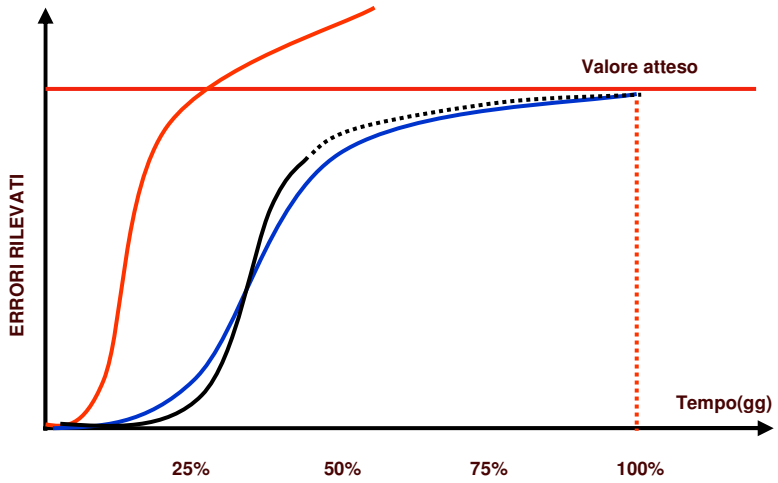

Figura 15. Singola fase di test (es. System Test)

L'utilizzo di entrambe le tecniche ("profilo" e "curva di rimozione dei test") permettono di controllare statisticamente l'efficacia dei test e la difettosità residua.

> ***Nota**: la curva rossa indica che il software ha una difettosità superiore a quella prevista. Occorrerà quindi prolungare la fase di test fino a vedere la curva appiattirsi verso un nuovo asintoto (di valore superiore a quello atteso).*

In questo caso il modello statistico risulta essere impreciso perché non sono state eseguite con cura le fasi precedenti e/o non sono stati registrati tutti gli errori rilevati.

Un modo pratico di costruire la curva di rimozione degli errori durante una fase di test consiste nel registrare ogni giorno il numero di errori rilevati e riportare il valore cumulativo su di un grafico come quello mostrato nella figura precedente.

La tecnica è quella di eseguire il numero massimo di casi di prova possibili fin dall'inizio.

Molti di questi si bloccheranno per via di errori gravi che non permettono il loro completamento ma, appena risolti gli errori rilevati, i casi di prova proseguiranno la loro esecuzione rilevando un numero sempre maggiore di errori.

L'appiattimento della curva indicherà il momento in cui si è esaurita la capacità dei casi di prova di rilevare altri errori (nell'esempio dopo 21 giorni si può considerare completata la fase di test in oggetto). L'appiattimento della curva prima del valore "atteso" è dovuto ad una di queste possibili cause: 1) il software è meno difettoso di quanto previ-

sto (poco probabile!); 2) i casi di prova non sono adeguati (in numero ed efficacia) a rilevare tutti gli errori; 3) il valore atteso non è corretto (il modello non è adeguato).

> ***Nota**: l'uso più efficace di tale tecnica è quella permette di valutare, al termine di una fase di test, se essa sia stata efficace oppure no. Generalmente una fase di test termina, nel migliore dei casi, quando tutti i casi di prova pianificati sono stati completati. La curva mostra, invece, se i casi di prova eseguiti abbiano rilevato tutti gli errori previsti!*

10 Produzione dei documenti di test

Qui di seguito è fornita una guida alla stesura dei documenti previsti dal processo di testing. Per alcuni documenti è fornito un esempio di modello (template) per la sua stesura pratica, oppure semplicemente l'elenco degli argomenti da trattare. Ogni organizzazione modificherà i modelli proposti in base ai propri standard.

1) Piani

- Piano di qualità del prodotto
- Piano di test generale
- Piano di test di dettaglio
- Piano delle revisioni tecniche
- Verbale di collaudo

2) Documenti tecnici

- Richiesta di modifica
- Elenco dei casi di test
- Caso di test
- Matrice dei casi di test
- Scheda rilevazione anomalia

3) Report

- Rapporto periodico sullo stato di completamento dei test
- Rapporto periodico sullo stato di risoluzione degli errori
- Rapporto di completamento dei test

10.1 Piani

10.1.1 Piano di qualità del prodotto (Quality Plan)

Il "Piano di qualità" è uno strumento di grande valore per la gestione della qualità del prodotto software. Esso è formalmente richiesto pressoché da tutte le metodologie presenti sul mercato (ISO 9000, CMMI, EFQM, ecc.).

Esso contiene gli obiettivi di qualità che il prodotto dovrà raggiungere estraendoli direttamente dai requisiti espressi dal cliente e dagli standard di mercato (per esempio dai prodotti analoghi esistenti sul mercato e considerati il riferimento del settore, oppure da standard definiti da normative di legge e/o dello specifico settore di industria, ecc.). Gli obiettivi di qualità saranno chiari e misurabili. Ad ogni obiettivo di qualità sarà quindi associata la relativa metrica per la verifica del livello di qualità raggiunto.

L'utilizzo del modello descritto dalla norma ISO/IEC 9126, che costituisce un riferimento interpretativo delle caratteristiche del software, risulta di grande utilità nel definire gli obiettivi di qualità; essa costituisce anche una valida lista di controllo (*Checklist*).

Perché un progetto sia credibile, occorre che siano descritte le azioni che l'organizzazione intende svolgere per garantire il raggiungimento degli obiettivi di qualità dichiarati. Il paino prevede quindi che siano descritti le metodologie adoperate per lo sviluppo del software, i processi di progettazione, realizzazione e collaudo, i prodotti di fase realizzati ed i controlli di qualità applicati ad essi, le modalità di gestione della documentazione e della configurazione del sistema, le modalità di gestione delle modifiche al sistema e la gestione degli errori, le azioni correttive e preventive che si intendono adottare a fronte di azioni necessarie, l'approccio al miglioramento continuo.

Il piano è approvato dai responsabili delle due parti interessate (cliente e fornitore) ed è distribuito a tutte le parti interessate al progetto (stakeholders).

Il piano è mantenuto aggiornato in base alle modifiche al progetto apportate durante il suo svolgimento. Il documento è sottoposto alla gestione della configurazione e le nuove versioni del documento seguiranno lo stesso iter di approvazione seguito dalla versione iniziale.

I livelli di qualità raggiunti dal prodotto finale sono documentati in un apposito rapporto (Rapporto sulla qualità del prodotto finale) per la sua valutazione ed approvazione.

Elenco dei contenuti del Piano di qualità

Data la complessità dell'argomento si riportano di seguito solo i titoli che corrispondono ai vari capitoli del documento. L'indice è tratto dal capitolato tecnico emesso da una Amministrazione Pubblica, conforme alla norma ISO 9000.

1. INTRODUZIONE
   - 1.1 Descrizione del progetto
   - 1.2 Scopo del piano di qualità
   - 1.3 Documenti applicabili e di riferimento
   - 1.4 Glossario dei termini
2. ORGANIZZAZIONE DEL PROGETTO
   - 2.1 Organigramma ed interfacce del progetto
   - 2.2 Ruoli e responsabilità
   - 2.3 Formazione e addestramento
3. PROFILO DELLA QUALITA'[20]
   - 3.1 Requisiti di qualità per il prodotto
     - 3.1.1 Qualità del software
     - 3.1.2 Qualità della documentazione
   - 3.2 Requisiti di qualità per i servizi
   - 3.3 Procedura per la valutazione della qualità
4. CICLO DI VITA
   - 4.1 Processi per la gestione del progetto
   - 4.2 Processi per la progettazione e lo sviluppo del software
   - 4.3 Processi per i test e i collaudi
   - 4.4 Processi per la produzione, gestione, conservazione e salvaguardia della documentazione

[20] Il profilo della qualità del prodotto definisce gli obiettivi della qualità misurabili. Esso è di vitale importanza per la definizione della strategia di test, la progettazione dei test, la predisposizione degli ambienti, l'esecuzione e la valutazione dei risultati. Per meglio capire la struttura di un profilo di qualità del prodotto, nel paragrafo successivo di questo capitolo è fornito un esempio concreto preso dalla realtà.

10.1.2 Profilo della qualità

Il profilo della qualità di un prodotto software è di fondamentale importanza per il collaudo. Esso definisce le caratteristiche di qualità del prodotto, stabilisce le metriche da utilizzare per la misurazione dette caratteristiche, stabilisce altresì i valori di soglia per ciascuna caratteristica/metrica. La tabella che segue mostra un esempio di profilo di qualità in alcune caratteristiche salienti.

Tabella 5. Profilo della qualità.

Caratteristica	Sottocaratteristica	Metrica	Valore soglia
Funzionalità	Adeguatezza	% funzioni sviluppate rispetto ai requisiti espressi nelle specifiche approvate.	100%
		% funzioni descritte nella documentazione rispetto a quelle sviluppate.	100%
	Accuratezza	Numero di casi di test da progettare in funzione delle dimensioni del software.	Stabilire secondo un algoritmo da definire
		% di test eseguiti in fase di test rispetto a quelli progettati nelle Specifiche di Test (pertinenti alla fase ed al prodotto di fase).	100%
		% di casi di test eseguiti con successo rispetto al numero di casi di test previsti nelle Specifiche di Test.	100%
Affidabilità	Tolleranza	Numero di malfunzionamenti (Bloccanti) che hanno causato una indisponibilità totale del prodotto, nei primi tre mesi di esercizio.	<= 2
	Maturità (Difettosità residua)	Numero di difetti del prodotto, emersi come malfunzionamenti con gravità massima (Bloccante o Grave), rilevati su periodi trimestrali a partire dal rilascio in esercizio.	<= 0,005/FP
Manutenibilità	Leggibilità	Percentuale di commenti sul volume totale di codice sorgente (in dipendenza della criticità del prodotto).	> 20%
Ecc.			

Le caratteristiche e sottocaratteristiche sono prese dal modello per la qualità del software definito dalle norme ISO/IEC 9126.

E' evidente dall'esempio riportato sopra la connessione tra il profilo della qualità e la strategia di testing.

10.1.3 Piano di test generale (Master Test Plan)

Il piano di test generale ha lo scopo principale di:

- descrivere la strategia di test che si intende adottare per verificare e valutare il livello con il quale sono indirizzati i requisiti da parte del prodotto realizzato e quale livello di qualità è stato raggiunto in termini di prestazioni (usabilità, performance, sicurezza, ecc.);
- fornire una descrizione sommaria dei tipi di test che si intendono eseguire (revisioni tecniche dei documenti e dei manuali utente, test unitario, test d'integrazione, test di sistema, test specifici);
- fornire una descrizione sommaria degli ambienti di test e relative risorse richieste;
- definire i piani di test di dettaglio che saranno prodotti.

Il piano di test generale fornisce inoltre informazioni sulla pianificazione generale delle attività e risorse di massima necessarie. Il piano costituisce una sorta di "accordo" tra lo sviluppo e le altre parti interessate (staheholders) sulla valutazione dei risultati raggiunti dal progetto. Esso descrive "che cosa" (ambito ed obiettivi), perché (scopo), "quando" (milestone) e "chi" (ruoli e responsabilità)per tutti i livelli di test.

Il piano di test generale è prodotto secondo un processo iterativo continuo. Si inizia a definirlo nelle prime fasi di studio di fattibilità e lo si raffina man mano che si ottengono maggiori informazioni sul progetto. In fase di pianificazione si produce la versione destinata ad essere approvata dalle parti interessate al progetto.

La strategia di test è definita in modo da indirizzare i requisiti qualitativi come descritti nel Piano di qualità (requisiti funzionali, requisiti prestazionali, standard applicabili, ecc.).

Il piano, una volta approvato, consente di produrre i piani di test di dettaglio in maniera coerente e sinergica.

Per progetti di dimensioni minori il piano di test generale può essere combinato in un unico documento con il piano di test di dettaglio.

Elenco dei contenuti del Piano di test generale

Di seguito i contenuti minimi richiesti per un tale piano.

- **Obiettivi dei test**: descrive gli obiettivi tangibili che si intende raggiungere con le attività di collaudo pianificate.
- **Rischi**: riporta la valutazione dei potenziali rischi di business derivanti dai requisiti funzionali e dalle scelte tecnologiche fatte dalla progettazione e dallo sviluppo.
- **Aree di test da focalizzare**: elenca (e descrive brevemente, se necessario) le aree da sottoporre a collaudo in base al progetto sviluppato ed agli obiettivi di qualità.
- **Livelli di test**: elenca (e descrive brevemente, se necessario) i livelli di test previsti dagli standard applicabili (test unitario, test d'integrazione, test di sistema, test prestazionali).
- **Tipi di test**: elenca (e descrive brevemente, se necessario) tipi di test legati alle caratteristiche funzionali (test funzionali) ed alle caratteristiche strutturali del prodotto.
- **Organizzazione**: elenca e descrive i ruoli e le responsabilità relative alle attività di collaudo.
- **Criteri di entrata e di uscita**: descrive chiaramente i criteri di entrata e quelli di uscita di ciascun tipo e livello di test.
- **Strategia per i dati di test**: descrive l'approccio per la creazione e l'aggiornamento dei dati di test.
- **Ambienti di test**: descrive i requisiti di alto livello per gli ambienti di tutti i tipi e livelli di test.
- **Strumenti (tool)**: descrive le specifiche, la selezione, la valutazione e le fasi per l'implementazione degli strumenti di test necessari.
- **Date principali (milestone)**: fornisce l'elenco delle principali date di test.
- **Procedure di test**: descrive le modalità di gestione delle attività di testing e del piano di test.
- **Reporting**: descrive come saranno rapportati gli stati di avanzamento dei test.

10.1.4 Piano delle revisioni tecniche (Inspection Plan)

Il "Piano delle revisioni tecniche" è un documento che riporta le attività di revisione tecnica previste nello svolgimento del progetto. Il documento è mantenuto aggiornato con i risultati delle revisioni effettuate. Può essere una semplice tabella che riporti almeno le seguenti informazioni di base:

- Identificativo del piano (titolo, autore, data di emissione, stato del piano, storia delle modifiche);
- Identificativo del progetto (nome e versione del progetto);
- Revisioni tecniche previste (documento da sottoporre a revisione, responsabile della revisione tecnica – generalmente l'autore oppure il responsabile dell'Assicurazione qualità – revisori, data prevista di revisione);
- Risultato delle revisioni effettuate (data di effettivo svolgimento, numero e gravità degli errori rilevati, risultato - revisione conclusa o da rieseguire -, eventuali note di chiarificazione).

L'utilizzo di uno strumento informatico faciliterà la gestione delle revisioni tecniche permettendo la produzione di report automatici con lo stato delle attività. Permetterà anche la produzione di report con l'analisi statistica degli errori rilevati per fase e per tipologia. Può essere documentato anche con una semplice tabella munita degli elementi necessari per la gestione del documento (titolo, autore, data di emissione, storia delle modifiche, ecc.).

Modello di un piano delle revisioni tecniche (Esempio)

Nella tabella che segue sono elencati i documenti che saranno sottoposti a revisione tecnica e le informazioni relative a ciascuna ispezione: autore, revisori, data di revisione, stato della revisione effettuata, numero di errori rilevati (con relativa gravità).

Documento da revisionare	Autore	Revisori	Data	Stato	Errori
Specifiche dei requisiti	Nome	Nome Nome	Gg/mm/aa	Completata/Da rieseguire	Numero (G)
Specifiche funzionali					
Design interfaccia					
Design DB					
Design componenti					
Prototipo					
Codice (a campione)					
Matrice di test					
Casi di test					
Manuale utente					

10.1.5 Piano di test di dettaglio (Detailed Test Plan)

Il piano di test di dettaglio è di fondamentale importanza per raggiungere gli obiettivi delle attività collaudo. Fornisce le risposte alle domande cruciali tipo:

- Cosa testare?
- Chi eseguirà i test?
- Quando saranno condotte le attività di test?
- Come saranno svolti i test?
- Come potremo sapere quando i test saranno finiti?
- Cosa sarà prodotto dalle attività di test?

Il piano di dettaglio di test è prodotto per ciascun livello di test. Per progetti più semplici è possibile realizzare un unico piano di test di dettaglio. Ciascun livello di test definisce il livello di integrazione del prodotto e gli obiettivi qualitativi che la soluzione sviluppata deve raggiungere. Per ciascun livello di test è definito: l'ambito del prodotto da collaudare, le funzioni ed i parametri da testare, i task da eseguire, il personale responsabile per ciascun task, i rischi di b business e tecnici da indirizzare.

Le caratteristiche del progetto – dimensioni, complessità, rischi e costi – determinano i livelli di test da previsti. I livelli di test comunemente utilizzati sono:

- Test unitario
- Test d'integrazione
- Test di sistema
- Test di accettazione (Collaudo utente)

Per ciascun livello di test sono specificati i tipi di test basandosi sugli attributi e le caratteristiche del prodotto. I tipi di test funzionali sono utilizzati per valicare le funzioni di business del prodotto; i test strutturali sono invece adoperati per validare i requisiti tecnici.

I tipi di test funzionali più comunemente eseguiti sono:

- Funzionalità del sistema
- Gestione degli errori
- Interfacce e connessione con altri sistemi
- Flusso delle transazioni
- Usabilità, operabilità.

I tipi di test strutturali più comunemente eseguiti sono:

- Backup an Restore
- Performance

- Stress e volume.

Per ciascun livello di test è specificato: la progettazione dei casi di test, i piani per l'esecuzione dei test, i risultati dei test attesi, i rapporti di test che descrivono i risultati conseguiti.

Elenco dei contenuti del Piano di test di dettaglio (Esempio)

- **Obiettivi del test**: descrive chiaramente gli obiettivi del test in oggetto.
- **Funzioni testate**: elenca (e descrive, se necessario) le funzioni oggetto del collaudo.
- **Attività previste**: elenca (e descrive, se necessario) le attività previste per la progettazione, l'approntamento e l'esecuzione dei test.
- **Criteri di entrata e di uscita**: descrive chiaramente i criteri che saranno utilizzati per valutare la possibilità di iniziare le attività di test e, ancora più importante, per concluderle.
- **Tecniche e strumenti di test (tool)**: descrive i requisiti necessari e le attività richieste per la selezione, l'acquisizione, l'installazione e l'utilizzo degli strumenti a supporto delle attività di collaudo negli ambienti di test.
- **Schedulazione delle attività**: descrive le attività di test previste e la relativa tempificazione (inizio, fine, durata, risorse necessarie).
- Ambienti di test: descrive i componenti hardware e software previsti per gli ambienti di collaudo, incluse le base di dati.
- **Rischi e precauzioni**: identifica i rischi del progetto, ne valuta gli impatti e definisce le azioni da intraprendere per mitigarli. Descrive inoltre le precauzioni da intraprendere.

10.1.6 Verbale di collaudo (Acceptance Test Report)

Il verbale di collaudo certifica l'esito del collaudo e, se positivo, permette di concludere l'attività e, di conseguenza, il progetto. L'importanza del verbale è anche dato dal fatto che alla sua emissione con esito positivo può partire il processo economico che vede l'emissione delle fatture di pagamento.

Da ciò anche l'importanza di un buon collaudo.

Il verbale di collaudo è un documento complesso i cui contenuti principali sono:

- **Generalità**. In questo capitolo è specificato il contratto, il progetto ed i componenti oggetto del collaudo. Specifica le modalità con cui si è svolto il collaudo, le date di inizio e fine, i prodotti consegnati, ecc.
- **Attività di collaudo**. Nel capitolo sono riepilogati i luoghi in cui sono svolte le attività di collaudo, gli ambienti ed eventuali note di rilievo. Specifica se si sono svolte tutte le attività previste e riporta le eventuali limitazioni intervenute. Sono indicati i principali documenti di riferimento utilizzati, le persone coinvolte ed i rispettivi ruoli e responsabilità, le date di inizio e fine previste ed effettive delle attività svolte, eventuali ritardi e relative conseguenze, ecc.
- **Esito inventario prodotti**. In questa parte del verbale è riportato l'esito dei controlli effettuati sui prodotti consegnati (documentazione, hardware e software) evidenziando eventuali mancanze e motivazioni delle stesse. E'm riportata la lista delle funzionalità sottoposte a collaudo, le modalità di controllo e gli esiti.
- **Esito attività di collaudo**. In questo capitolo sono riportati gli esiti delle verifiche tecniche e funzionali svolte durante l'attività di collaudo. Sono elencate anche le anomalie riscontrate, la severità, la data di rilevazione, quella di consegna del software modificato e quella di completamento delle verifiche di superamento delle anomalie.
- **Esito complessivo delle attività di collaudo**. Si riporta l'esito complessivo e globale del collaudo effettuato. In caso di non superamento è specificata la motivazione (anomalie riscontrate o inadeguatezza rispetto ai requisiti) e le azioni da intraprendere successivamente. In caso di conclusione positiva del collaudo, l'accettazione del prodotto è attestata tramite la sottoscrizione del verbale.

10.2 Documenti tecnici

10.2.1 Richiesta di modifica (DCR)

Le modifiche sono gestite secondo il processo di Design Change Request (DCR) descritto nel capitolo. Di seguito un esempio di scheda per la gestione manuale delle DCR.

Esempio di modulo di richiesta di modifica (DCR)

MODULO RICHIESTA MODIFICA (DCR)		
Progetto e versione:		
Data richiesta:		
Richiedente:		
Modifica richiesta (breve descrizione):		
Motivazione:		
Stima degli impatti:		
Elementi da modificare	**Impegno (gg/persona)**	
Requisiti:		
Specifiche:		
Disegno:		
Codifica:		
Testing:		
Documentazione utente:		
Note:		
Pianificazione:	**Inizio** (gg/mm/aaa):	**Durata** (gg):
Analisi e disegno:		
Codifica e test unitario:		
Test d'integrazione:		
Test di sistema:		
Note:		
Valutazione del rischio:		
Livello di rischio (Alto, Medio, Basso):		
Raccomandazione:		
Note:		
Approvazione:		
Chi approva (Nome / Reparto):		
Data di approvazione:		
Note:		

10.2.2 Lista dei casi di prova

L'elenco dei casi di test rappresenta un elemento importante in quanto semplifica la gestione e permette di svincolare i documenti formali (approvati) dalla necessità di aggiornamenti continui a seguito delle variazioni nell'elenco in oggetto.

Lista dei Casi di test (Esempio)

Identificativo	Obiettivo
CT1	Inserimento ordine
CT2	Modifica e stampa ordine
CT3	Modifica stato ordine
CT4	Visualizza e stampa rapporto degli ordini relativi allo specifico agente (rapporto 1)
CT5	Visualizza e stampa rapporto degli ordini inevasi (rapporto 2)
CT6	Visualizza e stampa rapporto degli articoli in giacenza in magazzino (rapporto 3)
CT7	Visualizza e stampa rapporto dello stato degli ordini (rapporto 4)
CT8	Visualizza e stampa rapporto degli ordini relativi allo specifico cliente (rapporto 5)
CT9	Cancella ordine
CT10	Spedizione dati ad altra applicazione
CT11	Spedizione dati ad altra applicazione
CT12	Ricerca ordine
CT13	Visualizza e stampa rapporto1 su due pagine
CT14	Visualizza e stampa rapporto2 su due pagine
CT15	Inserimento e stampa di un ordine con più di 20 articoli
CT16	Modifica e stampa di un ordine con più di 20 articoli
CT17	Sicurezza
CT18	Performance: tempi di risposta inserimento ordine e Stress test
CT19	Usabilità

10.2.3 Caso di test (Test Case)

I "Casi di prova (o Casi di test)" sono usati per verificare in un ambiente controllato (ambiente di test) le caratteristiche del prodotto, funzionali e non (efficienza, usabilità, sicurezza, installabilità, portabilità, performance, ecc.). L'obiettivo dei casi di test è descrivere e documentare le attività di validazione del software e renderle ripetibili anche da parte di persone diverse. I casi di test possono essere utilizzati come dettaglio delle condizioni di accettazione tra il cliente ed il fornitore. In ognuno dei casi di test sono descritte le attività di preparazione dei test, l'esecuzione degli stessi con i dati da immettere ed i risultati attesi.

In particolare, ogni caso di test contiene:

- Identificativo del caso di test (mnemonico o numero progressivo);
- Descrizione del test da eseguire;
- Prerequisiti;
- Dati di input del test;
- Risultati attesi.

L'esecuzione del caso di test è documentata tramite:

- Versione del sistema testato;
- Data e ora dell'esecuzione;
- Nome della persona che esegue il test;
- Differenze tra l'esecuzione attesa e quella reale, se ce ne sono;
- Differenze tra i valori dei dati di output attesi e quelli reali, se ce ne sono.

Una "batteria di test" (*Test Suite*) è un insieme di casi di test, spesso organizzati in sequenza, che formano un gruppo di controllo da eseguire sul prodotto. La batteria di test è utilizzata per certificare che il prodotto soddisfa i requisiti qualitativi richiesti, così come definiti nelle specifiche funzionali e nel piano della qualità.

Esempio di Caso di test

Caso di test	
Nome del progetto	AGORD
Identificativo del CT	CT1-P1a
Versione dell'applicazione	V1.0
Titolo del caso di test	Inserimento dell'ordine
Preparazione dei test (Pre-condizione)	Esiste un agente definito nel sistema abilitato all'inserimento. Il cliente Barbablù è definito nella tabella Clienti Sono definiti i seguenti prodotti con le corrispondenti quantità: PD01 (qtà 100), PD42 (qtà 100), PD57 (qtà 100)
Esecuzione del test	1. L'agente si collega al sistema 2. Il sistema richiede all'agente di autenticarsi 3. L'agente inserisce il proprio codice e la parola chiave 4. Il sistema autentica l'agente e consente l'accesso al sistema presentando la prima schermata. 5. L'agente richiede di immettere l'ordine 6. L'agente immette i dati relativi all'ordine 7. L'agente conferma l'inserimento 8. Il sistema notifica che l'inserimento è avvenuto correttamente 9. L'agente chiude il collegamento
Input (dati da inserire)	Cliente: Barbablù Data:12/05/2000 Prodotto: PD01 Quantità: 10 Prodotto: PD42 Quantità: 4 Prodotto: PD57 Quantità: 1
Risultato atteso	L'ordine è stato memorizzato nel database. È visualizzato il messaggio: "Inserimento dell'ordine relativo al cliente Barbablù è stato memorizzato con numero d'ordine 11"
Testatore	Luigi Bianchi
Data esecuzione	08/05/2000
Anomalie riscontrate (*)	13, 25
Durata (complessiva)	90 min.
Stato/Risultato finale	Da eseguire/In esecuzione/Completato con successo
Responsabile	Luigi Bianchi
Data chiusura	09/05/2000

(*) Indicare il progressivo con cui si è identificata l'anomalia nella tabella "Rilevazione Anomalia" nel paragrafo "Risultati" del documento "Piano di Test"

10.2.4 Matrice di test (Test Matrix)

La "Matrice di test" mette in rapporto le caratteristiche del prodotto (funzionali e non) con i casi di test definiti. Generalmente rappresentata con una tabella dove, sulle righe sono elencate le caratteristiche del prodotto: funzionalità, usabilità, manutenibilità, installabilità, sicurezza, performance, ecc. e sulle colonne sono indicati i casi di test definiti, con i parametri associati. Supposto, per esempio, che un caso di test debba essere eseguito passandogli in input tre parametri e che si voglia esercitare il caso di test con una terna di valori validi e con una terna di valori non validi, nella matrice di test si definiscono due colonne, ognuna relativa allo stesso caso di test ma associata alle due differenti terne di parametri.

La matrice di test consente d'identificare la copertura fornita dai test definiti, stimare i tempi/costi ed eliminare i casi di test ridondanti o poco importanti.

Infatti, avendo listate tutte le caratteristiche del prodotto (funzionali e non) è immediato dedurre dalla matrice se esiste almeno un test per ognuna delle caratteristiche, avendo indicazione della copertura fornita dalla batteria dei casi di test definiti.

Inoltre, avendo a disposizione la lista dei casi di test da eseguire e possedendo le opportune informazioni per quantificare i singoli test, si è in grado di poter dare una stima dei tempi richiesti per il test.

Qui di seguito sono evidenziati i criteri guida per la riduzione del numero di casi di test che devono essere esercitati:

- stesse funzioni testate da più casi di test (i casi di test sono ridondati);
- accorpamenti di più funzioni in un unico caso di test (purché il caso di test non diventi troppo complesso);
- le funzioni che gli utenti utilizzano con maggior frequenza devono essere sempre testate;
- le funzioni che risultano essere le più critiche per il business relativo devono essere sempre testate;
- completezza dei dati.

Segue un esempio di modello per la costruzione di una matrice di test.

Esempio di Matrice dei casi di test.

Funzionalità/ Condizioni (normali e di errore)	CT1–P1a	CT1-P1b	CT1-P1c	CT2-P2a	CT3	CT4	CT5	CT6	CT7	CT8	CT9	CT10	CT11	CT12-P12a	CT12-P12b	CT12-P12c	CT12-P12d	CT12-P12e	CT13	CT14	CT15	CT16	CT17	CT18	CT19
Inserimento ordine	X	X	X																X						
Modifica ordine				X																					
Modifica stato					X															X					
Stampa Ordine				X															X	X					
Visualizza rapporto1						X															X				
Visualizza rapporto2							X															X			
Visualizza rapporto3								X																	
Visualizza rapporto4									X																
Visualizza rapporto5										X															
Stampa rapporto1						X															X				
Stampa rapporto2							X															X			
Stampa rapporto3								X																	
Stampa rapporto4									X																
Stampa rapporto5										X															
Spedizione dati fatt.												X	X												
Ricerca Ordine														X	X	X	X	X							
Cancella Ordine											X														
Invio nota			X																						
Sicurezza	X																						X		
Performance																								X	
Usabilità	X																								X
Stress																								X	

Il rapporto tra il numero totale di "x" ed il numero totale di "funzioni" determina il "livello di copertura" dei test.
Copertura = Num. "x" / Num. Funzioni

Scheda rilevazione anomalia (Esempio)

SCHEDA RILEVAZIONE ANOMALIA
Identificativo dell'anomalia:
Stato dell'anomalia: Aperta/Assealidata/Chiusa
Titolo dell'anomalia:
Rilevazione del malfunzionamento: Responsabile del rilevamento (Tester): Data rilevamento: Descrizione del malfunzionamento: Test che ha rilevato l'anomalia: Documentazione disponibile: **Note**:
Risoluzione del problema (sviluppo): Responsabile (Sviluppatore): Data risoluzione: Descrizione della soluzione: Modifiche apportate: **Note**:
Validazione della soluzione (testing): Esito validazione: Data chiusura anomalia: **Note**:

10.3 Rapportistica

10.3.1 Rapporto di revisione/ispezione (Inspection Report)

E' utilizzato per tenere traccia delle revisioni eseguite e degli errori riscontrati.

Può essere il risultato dell'elaborazione (stampa) di uno strumento automatico (tool) oppure un semplice modulo cartaceo da inserire in seguito in un modello elettronico.

Le informazioni di base da rilevare sono divise in tre sezioni:

Informazioni generali. Nome dell'applicazione e del componente ispezionato, nome del documento oggetto della revisione, data di revisione, e data di chiusura della revisione, tipo di revisione effettuata (revisione di un documento, ispezione di un programma, walkthrough di simulazione di una esecuzione), data di chiusura, nome degli autori, dei revisori e del moderatore,

Identificazione degli Errori. Riporta l'elenco degli errori rilevati durante la revisione in termini di: numero progressivo per la sua identificazione, descrizione dell'errore, severità/gravità (grave, lieve), tipologia (errore di requisito, specifiche, disegno, codice, logica, banche dati di prova)

Risultato finale. Riporta le conclusioni della revisione in termini di commenti (esempio: è sufficiente correggere gli errori riscontrati per ritenere conclusa l'attività, oppure: occorre ripianificare una seconda revisione dopo la correzione egli errori in quanto si è fortemente modificato il disegno originale, ecc.), necessità o meno di ripetere la revisione, firma del moderatore.

Rapporto sull'esito di una revisione tecnica (Esempio)

RAPPORTO DI REVISIONE	
Applicazione:	**Data:**
Componente:	**Chiusura:**
Documento:	**Tipo Revisione/Ispezione:**
Autori:	
Ispettori/Revisori:	
Moderatore:	

N.ro	Descrizione	Sev.	Tipologia

Risultato:

Richiesta di nuova revisione: SI/NO	**Firma:**

Severità: G=Grave, M=Medio, L=Lieve
Tipologia: REQ: Requisiti, SPEC: Specifiche, DIS: Disegno, COD: Codice, DOC: documento, STD: Standard

10.3.2 Rapporto sullo stato di avanzamento dei test (Test Status Report)

E' utilizzato per rapportare lo stato di avanzamento dei test.

Può essere il risultato dell'elaborazione (stampa) di uno strumento automatico (tool) oppure un semplice modulo cartaceo da inserire in seguito in un modello elettronico.

Le informazioni di base da rilevare e riportare sono:

- data di inizio e di fine prevista dei test
- data della rilevazione dello stato dei test
- stato dei casi di test in termini di: Numero di casi di test pianificati, eseguiti, completati
- stato degli errori rilevati in termini di: numero totale per gravità (bloccante, grave, lieve)
- stato degli errori corretti in termini di: numero totale per gravità (bloccante, grave, lieve).

Rapporto sullo stato di completamento dei test (Esempio)

Rapporto sullo stato di completamento dei test									
				Inizio:			**Fine**:		
	Casi di test			**Anomalie trovate**			**Anomalie risolte**		
Data	**Pianifi-cati**	**Eseguiti**	**Completati**	B	G	L	B	G	L

Legenda:

B = Blocante, **G** = Grave, **L** = Lieve.

10.3.3 Rapporto di rilevazione errori (Error Removal Report)

E' utilizzato per la registrazione degli errori rilevati durante i test in assenza di uno strumento automatico (tool).

Le informazioni da registrare per ogni anomalia riscontrata è:

- Numero progressivo per l'identificazione univoca dell'errore
- Caso di test che ha generato l'errore
- Data di rilevazione
- Titolo/Descrizione dell'anomalia (i dettagli sull'anomalia sono fornito nella schema di rilevazione del difetto)
- Severità (Bloccante, Grave, Lieve)
- Data di risoluzione dell'errore

Rapporto sullo stato di risoluzione dei difetti (Esempio)

Lo stato di risoluzione delle anomalie è riassunto come segue.

Rilevamento delle anomalie					
N.ro	**ID Caso**	**Rilevato**	**Descrizione anomalia**	**Sev.**	**Risolto**
1	CT1	08/05/2000	Non accetta l'utente	B	08/05/2000
2	CT1	08/05/2000	Non memorizza dati sul DB	G	08/05/2000
3	CT1	09/05/2000	Messaggio errato	L	09/05/2000
4	CT2	09/05/2000	Campi non modificabili	G	10/05/2000
5	CT2	09/05/2000	Nella stampa i campi non sono incolonnati correttamente	L	09/05/2000
6	CT4	10/05/2000	Data di stampa rapporto errata	L	10/05/2000
….					
….					
….					
63	CT14	02/06/2000	……	G	02/06/2000

11 Strumenti di testing

Gli strumenti a supporto delle attività di testing sono:

- strumento per la "registrazione dei difetti";
- strumento per la "gestione delle modifiche"
- strumento per la "gestione e progettazione dei test";
- strumento per l' "esecuzione dei test unitari";
- strumento per l' "esecuzione automatica dei test";
- strumento per la "simulazione delle condizioni di carico";
- strumenti per l' "analisi statica del codice";
- altri strumenti specifici (es.: creazione delle basi di dati, creazione di scaffolding – driver e stub -, ecc.).

Segue una breve descrizione di ciascuna tipo di strumento.

11.1 Strumento per la registrazione dei difetti

Lo strumento permette di registrare i difetti rilevati durante l'esecuzione dei test e di gestire gli stati previsti. Le principali funzioni fornite sono:

Registrazione di un difetto e gestione degli stati previsti: il flusso di lavoro prevede, in sequenza, le seguenti attività: apertura del difetto ed assegnazione al gruppo di sviluppo, accettazione o rigetto del difetto da parte del gruppo di sviluppo, risoluzione dell'errore che ha causato il difetto (nel caso di accettazione) o registrazione delle motivazioni del rigetto (in caso di non accettazione), verifica della risoluzione (esecuzione del caso di test che ha rilevato il difetto), chiusura del difetto;

Inserimento e/o aggiornamento dei dati del difetto: i dati inseriti e/o aggiornati dipendono dalla fase di gestione del difetto e relativo stato del difetto:

- *Apertura ed assegnazione del difetto*: identificativo, data apertura, descrizione, caso di test, gravità, data assegnazione, nominativo/gruppo cui è assegnato il difetto. L'assegnazione è notificata all'assegnatario via e-mail;
- *Analisi e risoluzione del difetto*: descrizione della causa del difetto (individuazione dell'errore), tipo di errore (codice, dati, disegno, specifiche, requisiti, documentazione utente, altro), correzione (elemento componenti modificati), altre informazioni utili (eventuali aree contigue da testare, ecc.);
- *Verifica della risoluzione*: esito dell'esecuzione del caso di test che ha rilevato il difetto e verifica della sua scomparsa, data completamento della verifica;
- *Chiusura del difetto*: data di chiusura, motivazione.

Produzione di report statistici di sintesi e di dettaglio sui difetti e gli errori.

Figura 16. Strumento per la gestione dei difetti.

Lo strumento è essenziale nelle attività di testing e permette, nelle versioni più sofisticate di creare anche le curve di rimozione degli errori e di individuare il limite di saturazione (l'asintoto della curva).

Esistono diversi strumenti informatici per la gestione dei difetti, alcuni disponibili anche sulla Rete.

> ***Nota**: la funzione di "gestione dei difetti" è solitamente fornita dallo strumento insieme ad un'altra funzione importante come la "gestione delle feature", cioè la gestione delle modifiche a fronte di nuove funzionalità o modifiche di quelle esistenti.*
>
> *La descrizione di questa seconda funzione è descritta nel paragrafo successivo.*

11.2 Strumento per la gestione delle modifiche del software

> ***Nota**: la funzione di "gestione delle modifiche" è solitamente fornita dallo strumento insieme all'altra funzione importante di "gestione dei difetti" (descritta nel paragrafo precedente).*

Le funzioni principali relative alla gestione delle modifiche del software sono:

- *Registrazione delle modifiche*: si tratta di poter aprire una richiesta di modifica e di registrare le informazioni relative (identificativo, descrizione, data, priorità, applicazione cui si applica la modifica, altre informazioni di dettaglio);
- *Aggioramento delle informazioni legate al ciclo di vita della modifica*: analisi e valutazione dell'impatto, stato dello sviluppo, stato del test, chiusura della modifica;
- *Gestione del flusso (work-flow) e degli stati del processo*: apertura, analisi e stima degli impatti, approvazione, pianificazione, progettazione e sviluppo, testing, rilascio;
- *Produzione di report sullo stato delle modifiche*: report statistici co dati di sintesi e di dettaglio relativamente agli stati delle richieste di modifica (aperte, approvate, sviluppate, testate, rilasciate).

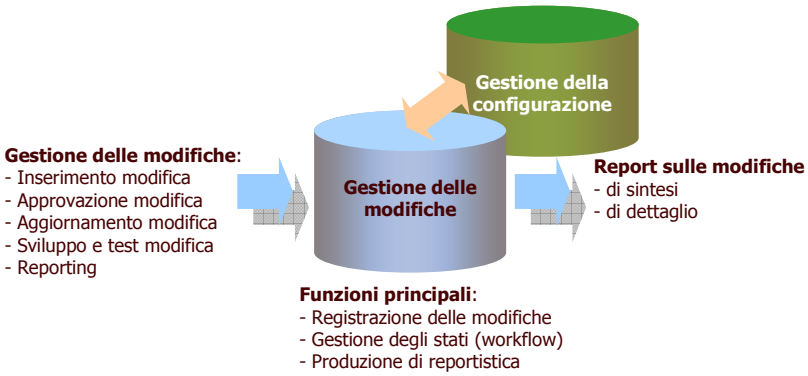

Figura 17. Strumento per la gestione delle modifiche del software.

La gestione delle richieste di modifica è strettamente legata alla gestione della configurazione del software e degli ambienti di test, come mostrato anche nella figura precedente.

11.3 Strumento per la gestione e la progettazione dei test

Si tratta di uno strumento informatico per pianificare le fasi di test, progettare i casi di test, registrare l'esito della loro esecuzione e produrre la reportistica necessaria.

Le funzionalità principali svolte dal responsabile dei test (Test Manager) e dai singoli tester con l'utilizzo di tale strumento sono:

- *Apertura di una fase di testing ed inserimento delle informazioni previste*: fase di test (unitario, d'integrazione, di sistema, di accettazione), responsabile, data di inizio e di fine previste;
- *Definizione dei casi di test previsti*: elenco dei casi di test previsti dalla fase;
- *Progettazione dei casi di test e registrazione delle informazioni*: funzionalità testate, requisiti indirizzati, attività da eseguire e/o input richiesti, risultati attesi, prerequisiti;
- *Aggiornamento dello stato dei casi di test*: aggiornamento dell'esito dei casi di test man mano che essi sono eseguiti, bloccati a causa di un difetto, ripresi ed infine completati con successo; di conseguenza è aggiornata la fase complessiva della fase di test;
- *Produzione della documentazione tecnica relativa ai casi di test*;
- *Produzione di report sullo stato dei casi di test* (completati, in esecuzione, bloccati, ancora da far partire).

Per quanto riguarda la progettazione dei casi di test, la descrizione delle attività che il tester dovrà svolgere nell'eseguire il caso di test può essere direttamente tratta dal "caso d'uso" (*Use Case*) nel caso in cui si utilizzi tale notazione e questa sia disponibile elettronicamente.

Durante l'esecuzione dei casi di test si rende necessario registrare i difetti rilevati tramite lo strumento descritto in un paragrafo precedente. Alcune soluzioni offrono strumenti integrati tra di loro: l'aggiornamento di un caso di test che ha rilevato un difetto attiva automaticamente il secondo strumento per la registrazione del difetto e viceversa.

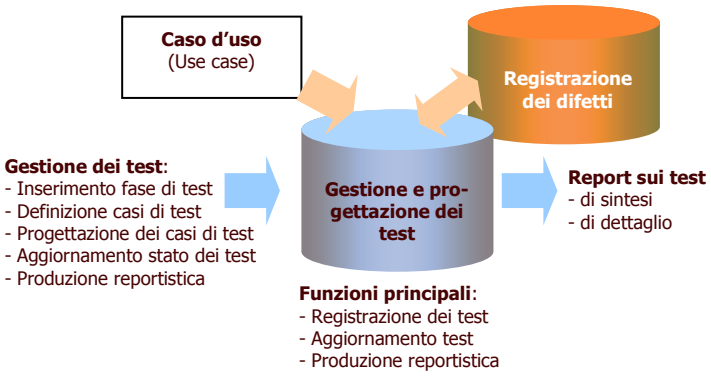

Figura 18. Strumento per la gestione e la progettazione dei test.

Tali strumenti sono spesso inseriti all'interno di una "suite" che comprende anche altri strumenti (vedi descrizione strumenti di test nei paragrafi successivi).

Esistono strumenti offerti da ditte specializzate nel settore ma esistono anche prodotti resi disponibili all'interno della politica "open source".

11.4 Strumento per l'esecuzione dei test unitari

Si tratta di strumenti, ambienti di sviluppo (*development framework*) e librerie integrati, a supporto dei test unitari, eseguiti generalmente dagli stessi sviluppatori nei propri ambienti di sviluppo (es. Unix e/o Windows) e sono generalmente legati al particolare linguaggio di sviluppo utilizzato (C, C++, Java, JavaScript, VB, Delphi, ecc.).

Permettono l'esecuzione principalmente di test di componente ("white-box"), ma anche di test "black-box", cioè di tipo funzionali.

Possono essere legati ad altri tool con funzioni particolari quali, ad esempio, l'analisi dei rami del codice e verificare il livello di copertura eseguito dai test eseguiti.

Alcuni strumenti per i test unitari consentono anche di realizzare codice "di simulazione" (*scaffolding* - *driver* e *stub* -) da compilare ed integrare nel codice da testare.

Le funzioni principali che questi strumenti permettono di svolgere sono:

- eseguire il codice "vedendo" direttamente le istruzioni eseguite, i dati elaborati, i risultati ottenuti, il contenuto della memoria;
- modificare, se necessario, i dati per modifica re e/o correggere l'esecuzione;
- avere una visione "grafica" dei percorsi eseguiti e del livello di copertura del codice.

La disponibilità di strumenti di questo tipo è molto ampia, sia per le piattaforme proprietarie sia per quelle open source.

11.5 Strumento per l'esecuzione automatica dei test

Gli strumenti di questo tipo permettono di registrazione l'esecuzione dei casi di test e di rieseguirli automaticamente comparando i risultati delle due esecuzioni ed evidenziando le differenze.

Un utilizzo tipico quanto efficace di questi strumenti è nei test di "regressione". Si tratta di quei casi in cui occorre rieseguire un insieme di casi di test (anche un numero molto alto) su di un sistema consolidato ogni volta che siano apportate delle modifiche e si vuole assicurare che il resto del sistema non sia stato alterato dai cambiamenti.

L'utilizzo di tali strumenti (record and play back) richiede che i casi di test siano progettati in modo particolare, lasciando inalterato l'ambiente così come lo era prima dell'esecuzione. Se, per esempio, un caso di test inserisce un nuovo record su cui testare delle funzioni applicative, occorre che al termine del test sia cancellato il record inserito. In questo modo, l'esecuzione successiva dello stesso caso di test eviterà di fermarsi immediatamente per "errore di duplicazione nell'inserimento del record".

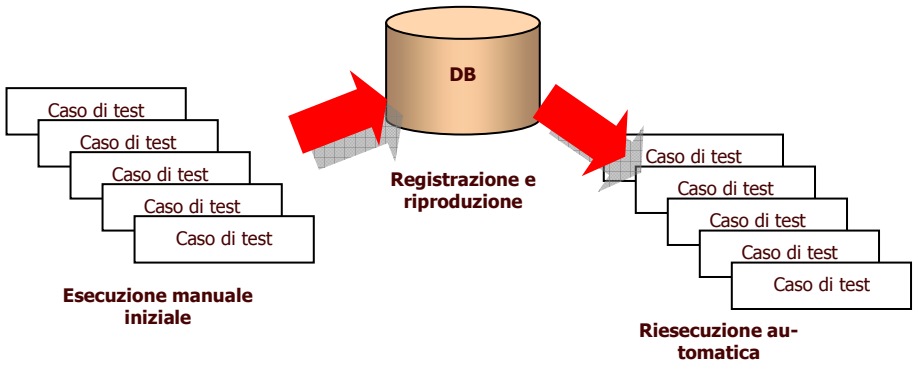

Figura 19. Registrazione e riesecuzione automatica dei test.

Anche non registrando test eseguiti precedentemente, è possibile utilizzare procedure apposite (scritte nei linguaggi forniti dai tool) per preparare o ripristinare gli ambienti e consentire l'esecuzione automatica di insiemi di casi di test anche molto sofisticati (test case suite).

Altri strumenti, molto più sofisticati, sono in grado di distinguere le variazioni nelle interfacce durante le diverse riesecuzioni (variazioni nelle dimensioni delle finestre, posizione di alcuni campi, colore e dimensioni delle schermate, ecc.).

11.6 Strumento per la simulazione delle condizioni di carico

Si tratta di strumenti specializzati nel simulare un carico alto sul sistema (specialmente sui sistemi client-server). Questi strumenti sono spesso comandati attraverso un'interfaccia grafica (GUI).

A seconda del tipo, dell'ambiente e dei linguaggi supportati gli strumenti possono essere utilizzati in modi e con obiettivi diversi.

Un modo di utilizzo, per esempio, è quello di simulare un numero alto di utenti contemporanei che accedono al sistema e di verificare quale sia il valore massimo supportato prima di mostrare un degrado nelle prestazioni.

Altri strumenti sono in grado di analizzare il codice (es. SQL) e di identificare i componenti che costituiscono un collo di bottiglia e riducono le prestazioni del sistema.

Altri strumenti, più sofisticati, permettono di testare il carico sulla rete e di predire il comportamento e le prestazioni dell'applicazione web. Sono in grado di testare l'intera architettura simulando un numero realistico di utenti connessi all'applicazione tramite la rete ed il carico che ne consegue. Alcuni sono in grado di identificare in tempo reale i componenti i crisi (colli di bottiglia) e le relative cause (velocità insufficiente, scarsa memoria, ecc.). Utilizzati come monitoraggio, alcuni di essi sono in grado di sorvegliare le transazioni e di registrare il contesto nel quale si generano eventuali problemi o incidenti (fermi, rallentamenti eclatanti, ecc.).

Altri strumenti, anche Open Source, sono molto specialistici e richiedono competenza specifica; in compenso permettono di generare carichi anche molto pesanti simulando l'attività anche di migliaia di utenti contemporanei virtuali. La funzione è realizzabile tramite l'architettura distribuita dello strumento nei diversi layer e componenti del sistema. Lo strumento visualizza anche in forma grafica i tempi di risposta degli utenti e l'utilizzo delle diverse risorse del sistema (Web Servers, Application Servers, Database Servers e Piattaforme operative su cui sono condotti i test).

Altri ancora sono specifici per i test di applicazioni che utilizzano apparecchiature wireless.

Molti generano anche report di dettaglio con le informazioni registrate.

11.7 Strumenti per l'analisi statica del codice

Si tratta di strumenti che analizzano i programmi senza eseguire il codice. L'analisi del codice sorgente riguarda caratteristiche come la "complessità", la "strutturazione", "la leggibilità", la "semplicità sintattica", ecc.

Ovviamente, questi tipi strumento dipendono fortemente dal linguaggio utilizzato per la scrittura del codice; non per tutti i linguaggi ci sono strumenti con le stesse funzionalità. Comunque, i linguaggi più utilizzati sono coperti.

I dati raccolti dall'analisi del codice sono elaborati e presentati in report con informazioni di sintensi e di dettaglio diversi.

Per quanto riguarda la complessità del software, gli strumenti calcolano quella "ciclomatica" e quella "esenziale" fornendo il valore calcolato per ogni singolo modulo e quello medio complessivo per l'intero parco applicativo analizzato.

Per quanto riguarda al "leggibilità", per esempio, viene effettuata l'analisi sintattica del codice sorgente in termini di

- lunghezza di ciascun modulo/funzione/blocco/if/while/switche;
- numero di strutture di controllo;
- numero di operatori nelle espressioni;
- livello di annidamento (nesting) dei blocchi;
- ecc.

Per alcuni linguaggi specifici (es. C/C++, Java, ecc.) è poi possibile adoperare strumenti ad hoc in grado di effettuare l'analisi del codice sorgente, individuare violazioni degli standard (alcuni strumenti hanno la capacità di verificare anche più di 300 standard) e permettere la correzione del codice direttamente dallo strumento che si posiziona sull'istruzione incriminata e suggerisce cosa e come cambiare.

Altri strumenti, simili a quelli precedenti, permettono di registrare i cammini percorsi dai casi di test eseguiti ed evidenziano gli eventuali percorsi duplicati seguiti da alcuni casi di test; in questo modo possibile eliminare i casi di test duplicati ed ottimizzare l'efficienza dei test.

E' indubbio l'aumento di produttività e di qualità che si ottiene con l'utilizzo di tali strumenti.

11.8 Altri strumenti specifici

Si tratta di strumenti specifici utilizzati nella fase di testing e con modalità e scopi diversi.

Alcuni, per esempio, permettono di creare delle basi di dati a partire da algoritmi e procedure specifiche. Possono generare intere basi di dati, specialmente se di grandissime dimensioni, con la generazione casuale (random) di dati specifici, oppure sequenziale (sequential) a partire da parametri definiti (da "a" a "z" con passo "x"). Sono utilissimi quando si tratti di generare archivi di grandi dimensioni con alcune chiavi sequenziali o casuali.

Altri strumenti permettono di costruire driver e stub a partire dal codice che li dovrà chiamare o da cui saranno chiamati.

Altri ancora, sono specifici per l'esecuzione di test di usabilità permettendo di registrare i comportamenti degli utenti a fronte di specifiche funzioni applicative adoperate. I comportamenti possono essere rappresentati dai tempi di risposta dell'utente nell'eseguire una transazione, oppure il tempo di reazione a fronte di una decisione da prendere, oppure ancora la correttezza nell'interpretare un messaggio del sistema, ecc. Altri comportamenti possono essere registrati a fronte dei "commenti" espressi dall'utente di fronte ad una funzione dell'applicativo. Gli strumenti relativi ai test di usabilità sono molto specifici e sono da utilizzare solo da parte di esperti in grado di valutare le reazioni degli utenti ed evitare qualsiasi tipo di equivoco, cattiva interpretazione o reazione negativa da parte dell'utente.

Molti altri strumenti esistono e sarebbe troppo lungo elencarli e descriverli.

12 Metriche di test

Di seguito sono elencate e descritte brevemente le metriche più comuni da adottare nel processo di test. In particolare, sono descritte:

Metriche relative alle attività di test

1. Errori rilevati dalle attività di test;
2. Difettosità residua del software;
3. Copertura dei test;
4. Efficacia dei test;
5. Efficacia delle revisioni;
6. Efficienza dei test;
7. Produttività dei test.

Metriche relative alla qualità del software e direttamente connesse alle attività di test

1. Complessità;
2. Livello di difettosità;
3. Usabilità;
4. Efficienza;
5. Robustezza;
6. Affidabilità;
7. Ricuperabilità;
8. Manutenibilità.

12.1 Metriche relative alle attività di test

12.1.1 Errori rilevati dalle attività di test

Rappresenta il numero di errori rilevati nel prodotto durante le diverse fasi di sviluppo. Così si parla di numero di errori rilevati nel disegno, nel codice, nella documentazione, oppure nei casi di test, ecc. Gli errori sono rilevati sulla documentazione si riferiscono alle attività di revisione e ispezione tecnica, quelli di codice alle attività di test. Sono importanti in quanto permettono di calcolare l'efficacia del processo di revisione e di test e, da questa, la curva di rimozione degli errori durante lo sviluppo. Dalla curva di rimozione degli errori è possibile prevedere, con tecniche più o meno sofisticate, il tasso di errori residuo, ovvero il numero di errori che gli utenti troveranno in fase di esercizio del prodotto. Il numero di errori rilevati è normalizzato rispetto alle dimensioni del progetto.

Numero di errori = Numero di errori rilevati / Dimensioni del progetto(KLOCs o FPs)

12.1.2 Difettosità residua del software

Rappresenta il numero di errori residui che si suppone abbia il software una volta rilasciato in esercizio. Il tasso di difettosità residuo è ovviamente una stima e potrà essere realmente misurata solo a consuntivo contando il numero di anomalie rilevate dagli utenti in un periodo di tempo stabilito (per esempio nei primi sei mesi di utilizzo oppure nel primo anno di esercizio). Al momento del rilascio del prodotto la stima degli errori residui può essere fatta con tecniche diverse più o meno sofisticate che qui non descriviamo. Il numero a consuntivo, invece, è normalizzato con le dimensioni del prodotto (numero di Klocs o di FPs).

Difettosità residua = Numero di anomalie rilevate dagli utenti nel periodo considerato / Dimensioni del prodotto

12.1.3 Copertura dei test

Rappresenta il livello di copertura che i test eseguiti forniscono rispetto alle funzionalità offerte dal prodotto. E' misurato come rapporto tra le funzionalità effettivamente verificate dai casi di test eseguiti ed il numero totale delle funzioni disponibili nel prodotto ed è espresso in punti percentuale. Una buona copertura è pari al 100%. Può essere calcolata anche come rapporto tra il numero totale di casi di test eseguiti ed il numero totale delle funzionalità offerte dal prodotto. In questo caso il valore può superare il 100%. Questa seconda misurazione evidenzia che alcune funzioni sono testate con più di un caso di test, e deve essere sempre accompagnata dalla prima (copertura) che dimostra quante funzioni sono state testate.

Copertura del test (1) = Numero di funzioni testate / Numero totale di funzioni disponibili
Copertura del test (2) = Numero di casi di test eseguiti / Numero totale di funzioni disponibili

12.1.4 Efficacia dei test

Rappresenta la capacità di rilevare errori nel prodotto tramite le attività di test. E' misurata come rapporto tra il numero di errori rilevati ed il numero di casi di test eseguiti. Essa è comunque una misurazione qualitativa e non quantitativa. Infatti, il numero di errori rilevati tiene conto solo degli errori effettivamente riconosciuti come tali e di cui si fornisce una correzione del codice. Sono perciò esclusi dal conteggio gli errori duplicati di altri errori, quelli dovuti ad una operazione sbagliata del testatore, quelli dovuti ad una errata impostazione del sistema (ambiente, banche dati di prova, ecc.), ecc. Per riferirsi a questa misurazione si usa il termine di "errori validi". In ambienti di test più evoluti si calcola anche il rapporto tra il numero di errori "validi" ed il numero totale di errori rilevati. Un'altra misurazione dell'efficacia del test è rappresentata dal rapporto tra il numero di errori validi rilevati in fase di test e le dimensioni del prodotto.

Efficacia del test (1) = Numero di errori validi / Numero di casi di test eseguiti
Efficacia del test (2) = Numero di errori validi / Numero totale di errori rilevati
Efficacia del test (3) = Numero di errori validi / Dimensioni del prodotto

12.1.5 Efficacia delle revisioni

Rappresenta la capacità di rilevare errori nell'analisi e nel disegno del prodotto tramite le attività di revisione ed ispezioni tecniche. E' misurata come rapporto tra il numero di errori rilevati durante la revisione e le dimensioni del prodotto. Ovviamente le dimensioni del prodotto in fase di analisi e disegno rappresenta la stima iniziale, oppure di pagine revisionate.

Efficacia delle revisioni (1) = Numero di errori / Dimensioni del prodotto
Efficacia delle revisioni (2) = Numero di errori / Numero di pagine ispezionate

12.2 Metriche relative alla qualità del software e direttamente connesse alle attività di test

12.2.1 Complessità del software

Rappresenta il livello di complessità rispetto ad un modello strutturato e rispetto a strutture logiche definite. Può essere rappresentato dai seguenti indicatori:

- Complessità ciclomatica;
- Livello di accoppiamento;
- Livello di coesione.

La complessità è valutata tramite attività di revisione del codice sorgente con l'utilizzo di tool specifici, in quanto l'attività manuale possibile (tramite revisioni del codice) è onerosa da fare per grandi applicazioni.

12.2.2 Complessità ciclomatica

Rappresenta il livello di complessità di un modulo (programma) calcolato rispetto ad un modello strutturato, cioè secondo le regole della programmazione strutturata che tutti i programmatori conoscono. E' calcolata sul grafo che rappresenta la logica interna del modulo dove si prendono in considerazione i cammini logici percorsi dal software all'interno del modulo ed i nodi presenti, cioè i punti decisionali del programma. Il numero ciclomatico è stato introdotto da Tom McCabe e perfezionato da altri. Dal numero ciclomatico puro è stata derivata una seconda misurazione pratica che è quella effettivamente usata: la complessità ciclomatica "essenziale" di un modulo che è la complessità ciclomatica base calcolata sul grafo dopo aver eliminato tutti i cammini strutturati. Mentre per la complessità ciclomatica base Tom McCabe raccomandava valori inferiori a 10, per la complessità ciclomatica essenziale oggi si raccomandano mediamente valori non superiori a 4.

Complessità ciclomatica di un modulo $v(G) = e - n + 2$ (dove e = cammino, n = nodo)

12.2.3 Livello di accoppiamento

Rappresenta il grado di conoscenza che un modulo/programma ha su di un altro modulo/programma chiamante o chiamato. Esso definisce, cioè, se la logica di un programma dipende dalla logica di un altro programma. E' bene quindi che il software abbia un "valore basso di accoppiamento".

Esiste una scala di valori di accoppiamento a 6 livelli così definiti:

1. Per dato (livello ottimale);
2. Per strutture dati;
3. Per controllo;
4. Per elementi esterni;
5. Per aree dati comuni;
6. Per contenuto.

12.2.4 Livello di coesione

Rappresenta il grado di dipendenza funzionale tra le parti che compongono un modulo/programma, cioè il grado di attinenza delle sue varie parti. Occorre, quindi, che i programmi abbiano un "alto valore di coesione".

Esistono 7 livelli di coesione, in ordine crescente:

1. Casuale;
2. Associazione logica;
3. Temporale;
4. Procedurale;
5. Comunicazione;
6. Sequenziale;
7. Funzionale (**livello ottimale**).

12.2.5 Usabilità

Rappresenta la facilità di utilizzo del prodotto. E' un giudizio soggettivo espresso da uno o più utenti sulla propria percezione della facilità d'uso delle funzionalità sperimentate. Per ridurre la soggettività della misura si calcola la media del giudizio espresso da almeno cinque utenti diversi. Il giudizio è espresso utilizzando una scala di valori predefinita (per esempio, su una scala a tre valori: 1 = poco usabile, 2 = mediamente usabile, 3 = molto usabile, oppure 1= bassa usabilità, 2 = media usabilità, 3= alta usabilità, ecc.). Il giudizio espresso dagli utenti coinvolti dipende da fattori personali (es.: cultura, coinvolgimento, conoscenza, ecc.) e dalle attività svolte per esprimere il giudizio (es.: test di usabilità, oppure revisione di un prototipo). Si raccomanda, quindi di scegliere gli utenti da coinvolgere in numero e rappresentatività adeguata e di eseguire i test simulando quanto più possibile una situazione reale di utilizzo del prodotto (es.: caso d'uso reale e concreto).

Usabilità = Valore medio della valutazione espressa dagli utenti coinvolti su una scala predefinita di valori di usabilità (per esempio su una scala a tre valori di usabilità: bassa, media, alta)

***Nota**: Per applicazioni e-business, dove le interfacce utente sono basate sulle tecnologie web, l'usabilità è più complessa di quella appena esposta e le relative metriche sono più sofisticate e precise. Maggiori dettagli su tali metriche sono forniti nel documento "Usabilità dei siti Web" dello steso autore e già disponibile in ICM.*

12.2.6 Efficienza

Rappresenta le prestazioni del prodotto, cioè la sua capacità di reazione alle richieste dell'utente. Sono conosciute più propriamente come "performance" del prodotto. E' una misura oggettiva e si riferisce, generalmente ai tempi di risposta di una transazione oppure all'utilizzo di risorse di sistema da parte dell'applicazione per eseguire le funzionalità richieste.

Performance (1) = Tempo medio di risposta di una transazione (in secondi)
Performance (2) = Numero di spazio richiesto su disco o su memoria centrale
Performance (3) = % di utilizzo delle linee di trasmissione impiegate

12.2.7 Robustezza

Rappresenta la capacità del prodotto di continuare a funzionare senza degrado nelle sue prestazioni anche in condizioni particolari (esempio: uso continuativo 24 ore su 24, 7 giorni su 7), oppure in condizioni limite (esempio: 500 utenti collegati contemporaneamente). Si misurano le prestazioni del prodotto (tempi di risposta ed utilizzo delle risorse) e si verifica che non ci sia degrado nelle prestazione. Valgono le stesse metriche definite per calcolare le prestazioni.

Performance (reattività) = Tempo medio di risposta di una transazione (in secondi)
Performance (utilizzo memoria) = Numero di spazio richiesto su disco o su memoria centrale
Performance (utilizzo linee) = % di utilizzo delle linee di trasmissione impiegate

12.2.8 Recuperabilità

Rappresenta la capacità del prodotto di recuperare una condizione anomala quando questa si presenti (esempio: ripristino dei dati di partenza nel caso di caduta durante l'esecuzione di una transazione). Le capacità di recupero possono essere automatiche, semi-automatiche, manuali, assenti. Ovviamente un prodotto senza capacità di recupero delle situazioni anomale è di bassa qualità e potrebbe non essere accettato. Le metriche sono quindi:

Recuperabilità = Automaticamente, Semiautomatica, Manuale, Assente

12.2.9 Manutenibilità

Rappresenta la facilità di eseguire la manutenzione (correttiva o evolutiva) da parte di personale qualificato ma non necessariamente esperto del prodotto specifico. Nel caso di manutenzione correttiva (quella che interessa questo processo specifico) la manutenibilità può essere misurata tramite i seguenti parametri:

- Tempo medio di risoluzione delle anomalie in base alla loro gravità;
- Numero di correzioni di anomalie non andate a buon fine (che richiedono, cioè, un secondo intervento correttivo) rispetto al numero totale di anomalie risolte

Manutenibilità (1) = Tempo risoluzione medio/ Totale anomalie risolte
Manutenibilità (2) = Anomalie risolte con riciclo / Totale anomalie risolte

Bibliografia

Di seguito è riportata la bibliografia utilizzata dall'autore e suddivisa per temi.

Test

[MOSLEY03] Daniel J. Mosley, Bruce A. Posey - 2003
Collaudo del Software (titolo originale: Just Enough Software Test Automation) - McGram Hill

[KANER03] Cem Kaner, Lames Bach, Bret Pettichord – 2003
Lessons Learned in Software Testing, A Context-Driven Approach - Wiley

[CICOGNI98] Cignoni G. A., De Risi P. - 1998
Il test e la qualità del software - Il Sole 24 Ore

[GILB93] Gilb Tom and Graham Dorothy - 1993
Software Inspection - Addison Wesley

[CABE96] Mc Cabe T.J., Watson A.R., 1996
Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric - Dolores R. Wallace, Editor
http://www.mccabe.com/iq_research_nist.htm

[BEIZER90] Beizer B. - 1990
Software System Testing and Quality Assurance- International Thomson Computer Press

[MYERS79] Myers G. J. - 1979
The art of Software Testing - John Wiley & Sons, New York

Qualità

[LEONARDI00 Erika Leonardi - 2000
Capire la qualità – ISO9000 – Tutto quello che occorre capire per applicare con profitto le nuove norme - Il Sole 24 Ore

[BARBARINO98] Filippo C. Barbarino, Erika Leonardi - 1998
ISO9000 Sistema Qualità e Certificazione - Il Sole 24 Ore

[PINDER96] Mark Pinder & Stuart McAdam - 1996
La consulenza interna - Jackson Libri

[MUNRO94] Lesly Munro-Faure, Malcom Munro-Faure - 1994
Qualità totale: tecniche di attuazione - Jackson Libri

[PALA94] Giorgio Pala - 1994
La qualità. Perché, per chi e come farla - Franco Angeli

[BANCI93] Alessandro banci, Giuseppe Iacono - 1993
La qualità nei progetti software - Franco Angeli

[NOCENTINI93] Stefano Nocentini - 1993
Il sistema di qualità del software - ETASLIBRI

[WHEELER93] Donald J. Wheeler - 1993
Understanding Variation – The Key To Managing Chaos - SPC Press

[ZEITHAML91] Valerie A. Zeithaml, A. Parasuraman, Leonard L. Berry - 1991
Servire qualità - McGraw-Hill

[RUMMLER90] Geary A. Rummler and Alan P. Brache - 1990
Improving performance - Jossey-Bass Publishers

[CROSBY86] Philip B. Crosby - 1986
La qualità non costa - McGraw-Hill

Metriche del software

[NATALE96] Domenico Natale - 1996
Qualità e quantità nei sistemi software – Teoria ed esperienze
Franco Angeli/Informatica

[PULFORD96] Kevin Pulford, Annie Kuntzmann, Stephen Shirlaw - 1996
A quantitative approach to Software Management – The AMI Handbook (AMI ESPRIT) - Addison-Wesley

Usabilità del software

[UCD04] User-Centered Design (sito IBM sull'usabilità)
http://ucdwasp1.torolab.ibm.com/pmp/pm

[CANTONI03] Lorenzo Cantoni, Nicoletta Di Blas, Davide Bolchini - 2003
Comunicazione, qualità, usabilità - Apogeo

[CORSO03] Corso pratico – 2003
L'artista digitale (tiolo originale: Design Companion for the Digital Artist) - Mondatori Informatica

[NORMAN00] Donald A. Norman - 2000
Il computer invisibile - Apogeo

[VISCIOLA00] Michele Visciola - 2000
Usabilità dei siti Web - Apogeo

[NIELSEN00] Jakob Nielsen - 2000
Web usabilità - Apogeo

[SKLAR00] Joel Sklar - 2000
Principi di Web Design (titolo originale: Priciples of Web Design) - Apogeo

[LYNCH99] Patrick J. Lynch, Sarah Horton – 1999
Web, Guida di stile (titolo originale: Web Style Guide – Basic Design Priciples for Creating Web Sites) - Apogeo

[TOFONI99] Graziella Tofoni - 1999
Il design della scrittura multimediale - CUEN

[MASSIRIONI98] Manfredo Massironi - 1998
Fenomenologia della percezione visiva
Il Mulino (collana: Aspetti della psicologia)

[MANTOVANI95] Giuseppe Mantovani - 1995
L'interazione uomo-computer
Il Mulino (collana: Aspetti della psicologia)

ISO9000

[ISO9001:2000] ISO/IEC 9001:2000
Quality Management Systems – Requirements

[ISO9000:2000] ISO/IEC 9000:2000
Quality Management Systems – Fundamentals and Vocabulary

[ISO9004:2000] ISO/IEC 9004:2000
Quality Management Systems – Guidelines for performance improvement

[ISO9003:2004] ISO/IEC 9003:2004
Software and Systems Engineering – Guidelines for the application of ISO 9001:2000 to computer software

[ISO9126-1:2001] ISO/IEC 9126-1:2001
Software Engineering – Product Quality Part 1: Quality Model

[ISO9126-2:2002] ISO/IEC 9126-2:2002
Software Engineering – Product Quality Part 2: External Metrics

[ISO9126-3:2002] ISO/IEC 9126-3:2002
Software Engineering – Product Quality Part 3: Internal Metrics

[ISO9126-4:2002] ISO/IEC 9126-4:2002
Software Engineering – Product Quality Part 4: Quality In Use Metrics

[ISO12207:95] ISO/IEC 12207:1995
Information Technology – Software Lifecycle Processes

[ISO12207:02-1] ISO/IEC 12207:1995/Amd.1:2002
Information Technology – Software Lifecycle Processes-Amendment 1

[ISO15271:98] ISO/IEC TR 15271:1998
Information Technology – Guide for ISO/IEC 12207 Software Lifecycle Processes)

[ISO16326:99] ISO/IEC TR 16326:1999
Software Engineering – Guide for the application of ISO/IEC 12207 in Project Management

Capability Maturity Model Integration

[SEI-CMMI02] CMMI® for Development, Version 1.2
Improving processes for better products – CMMI Product Team
August 2006

[SEI-SPM01] Software Project Management
SEI Curriculum Module SEI-CM-21-1.0
July 1989 (Draft For Public Review)

La fase di validazione nel ciclo di sviluppo del software ricopre un ruolo di estrema importanza per la qualità del prodotto finale. Essa, infatti, permette di valutare il livello di qualità raggiunto dall'applicazione sviluppata evidenziandone gli errori e permettendone la loro correzione.

In particolare, il testing ha il compito di verificare l'aderenza ai requisiti e la correttezza dell'implementazione. In altri termini, essa verifica, da un lato, l'aderenza ai requisiti così come definiti nelle specifiche tecniche e nel piano di qualità (requisiti funzionali e requisiti qualitativi) e, d'altro, la correttezza della progettazione e della relativa implementazione.

Il documento descrive una metodologia di test completa, che indirizza sia i test statici (revisioni tecniche, ispezioni e walkthrough), sia dinamica (testing vero e proprio). La descrizione delle fasi di test include la pianificazione, la progettazione dei test e la predisposizione degli ambienti, l'esecuzione ed il controllo e monitoraggio. I livelli di test previsti sono quelli tradizionali (unitario, di integrazione, di sistema, collaudo utente). E' fatto un accenno anche ai test delle applicazioni destinate al Web (Il tema dei test delle applicazioni di e-business è trattato in dettaglio in un altro documento dello stesso autore). Sono inoltre descritte le attività di gestione delle anomalie e di gestione della configurazione di test. Il documento riporta anche le metriche relative ai test ed i metodi più comuni (white-box e black-box, top-down e bottom-up, controllo statistico della rimozione degli errori, curva di saturazione degli errori rimossi, matrice OTR ecc.).

Ercole Franco Colonese svolge la sua attività di consulente essenzialmente nell'area delle metodologie per lo sviluppo del softwaree dei sistemi qualità. La sua esperienza è maturata in moltissimi anni di lavoro presso i laboratori internazionali IBM e organizzazioni AMS dove ha ricoperto ruoli tecnici, manageriali e dirigenziali. Ha realizzato numerosi sistemi qualità aziendali certificati ISO9000 presso piccole, medie e grandi aziende, pubbliche e private. Ha implementato modelli di eccellenza (EFQM, Malcom Baldridge, MDQ). Ha condotto diversi progetti di reingegnerizzazione dei processi di sviluppo software in ottica di miglioramento delle performance e della qualità. Ha applicato con successo i modelli di maturità dei processi SEI-CMM e CMMI. Come docente, tiene corsi e seminari sulla qualità del software e le metodologie di sviluppo presso aziende ed università.

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