

# Qualità del software

*Linee guida per pianificare,  
realizzare e controllare la qualità del  
software utilizzando le migliori  
pratiche disponibili*

Ercole Colonese

Versione 2.0 - Dicembre 2005

### Nota relativa alla versione attuale

Questa seconda versione del documento è aggiornata in base all'esperienza maturata da alcune piccole e medie organizzazioni software che hanno adottato il modello proposto nella prima versione. Alcune aziende hanno potuto sperimentare molte delle tecniche descritte nel documento, mentre altre solo parte di esse. Tutte, comunque, hanno dichiarato l'importanza del coinvolgimento della direzione nella definizione del tema della qualità. Altrettanto importante è stato dichiarato il modello proposto basato sui tre elementi chiave:

1. *Competenza delle persone;*
2. *Maturità dei processi;*
3. *Metriche, metodi, tecniche e strumenti a supporto.*

Essendo tutte le organizzazioni certificate ISO9001, ciascuna ha aggiornato il proprio sistema di gestione per la qualità per indirizzare i seguenti punti qualificanti:

- Definizione più dettagliata dei ruoli e delle responsabilità, delle competenze richieste e predisposizione di un piano di formazione più mirato;
- Maggiore attenzione da parte della direzione alle tematiche della qualità ed in particolare alle attività dell'Assicurazione Qualità;
- Aggiornamento del processo di sviluppo software con riferimento alle metriche, metodi e tecniche, e strumenti selezionati. In particolare, sono state definite metriche semplici e dedotte dagli obiettivi di business aziendali;
- Adozione di metriche, metodi e tecniche, e strumenti a supporto dello sviluppo più semplici ed efficace. In particolare, si è deciso di fare un uso molto frequente delle revisioni tecniche "informali".

Alcune tecniche di particolare efficacia sono state sperimentate con successo ed incluse nei propri processi standard: Matrice di tracciabilità dei requisiti, Revisione tecnica non formale, Controllo della copertura dei test, Definizione di standard di codifica, Revisione di qualità (Assicurazione Qualità).

### Versione del documento

2001	2002	2003	2004	2005	2006
V1.0	v1.1	--	--	V2.0	

## Indice

Qualità del software .....	5
Introduzione.....	5
Analisi, progettazione e codifica.....	9
Un approccio metodologico concreto.....	12
I principi della qualità del software .....	15
Responsabilità per la qualità .....	15
Ambito della qualità del software .....	17
Assicurazione Qualità (Software Quality Assurance - SQA).....	18
Gestione dei requisiti (Requirements Management).....	19
Realizzazione del software (Software Engineering).....	20
Gestione dei progetti (Project Management).....	21
Gestione della configurazione (Configuration Management).....	22
Misurazioni del software (Software Measurement).....	22
Alcuni esempi di metriche del software.....	23
Le attività di gestione della qualità.....	25
Pianificazione della qualità del progetto.....	26
Controllo della qualità del progetto.....	27
Revisioni dell'Assicurazione della qualità (QA Review) .....	27
Revisione dell'offerta (QA1).....	29
Revisione della pianificazione (QA2).....	29
Revisione del rilascio (QA3) .....	29
Revisioni tecniche (Technical Reviews).....	30
Alcuni elementi importanti delle revisioni.....	32
Checklist per le revisioni tecniche .....	32
Collaudi (Testing).....	34
La qualità nella manutenzione del software.....	35
Metodi, tecniche e strumenti a supporto .....	36
ISO9126.....	38
Revisione strutturata (Peer Review) .....	39
Profilo di difettosità del prodotto.....	40
Classificazione ortogonale dei difetti.....	44
Curva di rimozione degli errori nei test (curva di saturazione).....	46
Propagazione degli errori.....	48
Archivio storico dei progetti .....	49
Analisi causale.....	52
Glossario.....	54
Riferimenti bibliografici .....	55



# Qualità del software

*Linee guida per pianificare,  
realizzare e controllare la qualità del  
software utilizzando le migliori  
pratiche disponibili*

## Introduzione

Il tema della qualità del software è stato ampiamente trattato nella letteratura specializzata ed è ricchissima la disponibilità di materiale al riguardo. Questo documento non vuole essere quindi l'ennesimo lavoro sul tema e ancor meno una duplicazione di quanto già esistente. Vuole invece essere (sperando di riuscirci) un contributo pratico per le piccole e medie imprese di software che vogliano approfondire la tematica dal punto di vista pratico, in maniera semplice ed efficace, ed adottarla nel proprio ambiente di sviluppo.

Iniziamo con la definizione di qualità del software per stabilire un comune punto di partenza.

*“La qualità del software è la sua capacità di soddisfare le aspettative di quanti direttamente interessati (stakeholders) al software in oggetto; in altre parole la capacità di indirizzare correttamente i requisiti, espliciti ed impliciti, e di aderire agli standard applicabili utilizzando un processo di sviluppo definito ed ottimizzato”.*  
*(Definizione liberamente tratta dai testi di ingegneria del software).*

Lo sviluppo del software è un'attività creativa e tecnica allo stesso tempo; un'attività ad altissimo contenuto intellettuale e quindi con un grande margine di errore umano. In estrema sintesi, si tratta di tradurre esigenze (spesso non ben identificate) in una soluzione tecnica deterministicamente corretta. Quello che possiamo fare è, da un lato, ridurre il numero di errori<sup>1</sup> inevitabilmente immessi nelle singole fasi e, dall'altro, potenziare la rimozione degli errori immessi individuandoli il più presto possibile, cioè nella fase stessa in cui questi sono stati introdotti.

Ci sono quindi due fasi cruciali per la qualità del software:

1. l'**immissione di errori** nella fase di realizzazione (analisi, disegno e codifica);
2. la **rimozione degli errori**, prima attraverso la revisione dei requisiti e della progettazione, poi attraverso la verifica (test) del codice sviluppato.

Un errore commesso in una fase iniziale del ciclo di sviluppo software (analisi, disegno e codifica) genera altri errori (da 3 a 5) nella fase successiva, e così via<sup>2</sup>. Un singolo errore di interpretazione di un requisito può generare anche 5 errori nelle specifiche successive, fino a 25 errori di disegno e fino a 125 errori di codifica.

L'analisi condotta sugli errori rilevati nel codice, inoltre, ha dimostrato che essi sono imputabili solo in minima parte ad una errata codifica (vedi tabella di seguito) mentre la maggior parte di essi è da imputare a specifiche e a requisiti incompleti o errati, a errori nel disegno dei dati, nella progettazione o nei test.

**Tabella 1. Tipologia di errori rilevati con maggiore frequenza.**

Tipologia	Perentuale
Specifiche errate o incomplete (IES)	22%
Requisiti errati o incompleti (IER)	17%
Errori nella rappresentazione dei dati (EDR)	14%
Test incompleto o errato (IET)	10%

---

<sup>1</sup> In letteratura, con il termine "errore" è indicata la causa di un difetto. Un errore è commesso a livello concettuale (ad esempio, l'errata comprensione di un requisito del cliente) oppure operativo (ad esempio, errata produzione del codice sorgente). "Difetto", invece, indica un vizio nel sistema o componente del sistema che determina il fallimento del sistema / componente nell'eseguire la funzionalità richiesta. Nel presente documento i due termini (errore e difetto) sono utilizzati con lo stesso significato non tanto per mancanza di precisione quanto per adeguarsi al comune utilizzo dei termini nella quotidianità facilitandone la comprensione al lettore.

<sup>2</sup> Vedere la parte successiva del documento in cui si descrive la teoria della "propagazione degli errori nel software".

## Sviluppo di Software Applicativo

---

Inconsistenza nell'interfaccia (ICI)	6%
Errore nella traduzione del disegno nel linguaggio di programmazione (PLT)	6%
Errore logico di progettazione (EDL)	5%
Deviazione intenzionale dalle specifiche (IDS)	5%
Documentazione incompleta o imprecisa (IID)	4%
Violazione degli standard di programmazione (VPS)	3%
Interfaccia uomo-macchina (HCI) ambigua o inconsistente (HCI)	3%
Miscellanea di altri errori (MIS)	5%

**Fonte:** Roger S. Pressman, *Principi di ingegneria del software* [PRESSMAN05].

Una prima conclusione (ovvia, ma poi non così tanto!) è che un buon livello di qualità si raggiunge con:

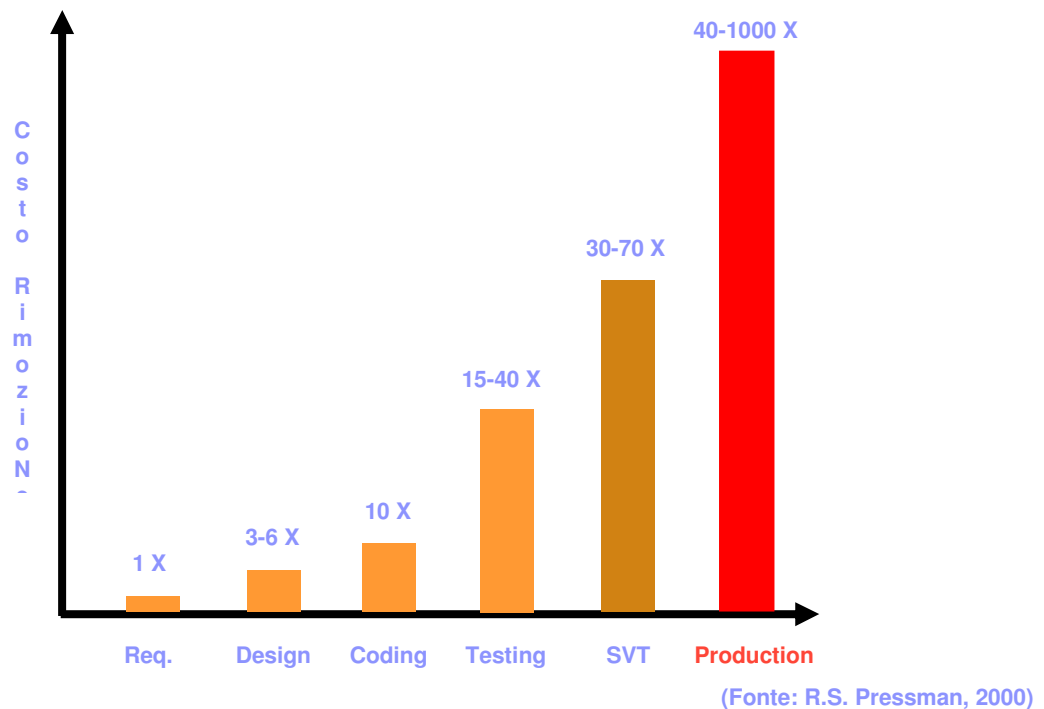
1. la *riduzione degli errori immessi* in fase di analisi, progettazione e codifica, e
2. il *potenziamento della rimozione degli errori* introdotti.

Basterebbe infatti introdurre il minor numero possibile di errori e poi, sapendo quanti e quali errori siano stati inseriti ed in quali documenti o programmi, correggerli!.

Purtroppo, il bilancio di queste due attività, nella pratica quotidiana, è sempre negativo ed un considerevole numero di errori (difetti) sfugge alle attività di correzione e sono quindi rilasciati in produzione insieme al codice: sarà allora l'utente a scoprirli e noi a correggerli, con suo disappunto e nostro affanno.

Ma il bilancio è ancora più negativo dal punto di vista economico. Il costo per la rimozione dei difetti in produzione è molto più alto di quello necessario per rimuoverli nelle fasi di sviluppo (da 40 anche a 1000 volte), così come emerge dagli studi statistici condotti per qualche decennio su numerosi progetti e rappresentato nella figura riportata di seguito.

Già da alcuni anni il mondo dello sviluppo software ha dirottato la propria attenzione dalla problematica della gestione degli errori nel software, a quella dei costi e tempi di sviluppo. La sfida oggi sembra essere quella di “produrre software a costi inferiori ed in tempi brevi.”.



**Figura 1. Costo della rimozione degli errori nel software.**

Un altro elemento che è cresciuto notevolmente negli ultimi decenni è la complessità del software e la conseguente difficoltà ad effettuarne la manutenzione. In pratica, il codice è continuamente modificato per rispondere a diverse esigenze: nuovi requisiti, adeguamento a nuove normative, miglioramento di aspetti funzionali, prestazionali e gestionali, ecc. Intervenire sul codice esistente può risultare, a volte, difficile e costoso quando esso non sia stato già progettato e realizzato correttamente per facilitare la sua manutenzione, sia correttiva che evolutiva e migliorativa. La qualità della progettazione è quindi un elemento fondamentale per agevolare l'evoluzione del software.

Cosa possiamo fare, allora, per risolvere il problema?

Una ricetta, ovvia, semplice ed efficace è sicuramente la seguente: *“capire bene le esigenze, progettare una soluzione corretta e modulare, produrre un codice robusto e facile da mantenere, eseguire test esaustivi ed efficaci”*.

L'ingegneria del software fornisce tecniche e metodi (*best practice*) di sicura efficacia per indirizzare queste problematiche.

### Analisi, progettazione e codifica

#### Analisi

La produzione di software di qualità presuppone una fase di analisi in cui i requisiti, espliciti ed impliciti, siano collezionati, analizzati, valutati, documentati e discussi con gli utenti ed i responsabili della progettazione. L'analisi deve risultare completa ed esaustiva, fare chiarezza sulle esigenze ed eliminare ogni forma di ambiguità. Una buona norma è quella di verificare che ogni requisito risulti “*chiaro, esaustivo e testabile*”, cioè non crei ambiguità nella sua descrizione e sia indirizzato da un set di casi di prova per la sua verifica e validazione. La “*Revisione dei requisiti con gli utenti*” e la creazione della “*Matrice di tracciabilità dei requisiti*” sono due tecniche che le organizzazioni software hanno sperimentato con enorme successo ed utilità.

Per quanto riguarda la definizione delle specifiche funzionali e tecniche si è dimostrato di grande utilità l'utilizzo di un qualche formalismo. Alcuni diagrammi UML sono risultati molto importanti per la qualità dell'analisi e sono applicabili anche in ambienti di sviluppo più tradizionale, cioè non object-oriented. Tra questi, sono risultati particolarmente utili i seguenti diagrammi: Casi d'uso, Diagramma di stato, Diagramma delle attività, Diagramma delle componenti, Diagramma dei rilasci (ovviamente, non tutti i diagrammi si applicano a tutti gli sviluppi software, fatta eccezione per i casi d'uso applicabili in tutti i casi).

Per verificare e valicare comuni modi di vedere ed interpretare le soluzioni tra utenti e progettisti, si sono utilizzate con successo le tecniche di simulazione (*Realizzazione di prototipi*) e tecniche organizzative (*Riunioni congiunte tecnici-utenti*). Le tecniche hanno permesso di migliorare le comunicazioni tra progettisti e utenti e di ridurre, se non proprio eliminare, le ambiguità e le inconsistenze.

#### Progettazione

La progettazione richiede competenza, esperienza e creatività. La definizione delle architetture applicative e la scelta delle tecnologie più appropriate da utilizzare sono attività fondamentali che richiedono competenza ed esperienza; interpretare le richieste con soluzioni innovative e di successo richiede spirito di iniziativa e creatività. Già ai tempi dello sviluppo tradizionale, alla base di uno sviluppo di qualità c'erano tecniche consolidate: *programmazione strutturata e information hiding*, tecniche validissime ancora oggi, anche in ambienti tecnologicamente avanzati. La progettazione basata sui componenti (*Component-Based Design*) e l'utilizzo (*Reuse*) di parti esistenti costituiscono elementi di sicuro valore sia intermini di qualità che di produttività. Una buona progettazione definisce comunque alcune regole fondamentali da seguire con assoluta precisione: strutturazione top-down, alta coesione e basso

accoppiamento. La tecnica *Top-Down* richiede che la progettazione parta dalla definizione del contesto applicativo su cui operare (definizione di tutti gli altri componenti esterni con cui interagire) per poi definire ad alto livello l'architettura complessiva dell'applicazione. Con successivi approfondimenti si passa a livelli di maggiore dettaglio, fino a giungere alla definizione delle singole componenti e moduli. Durante tali attività può risultare necessario approfondire alcuni elementi di dettaglio prima di passare alla scomposizione di componenti di maggiore dettaglio. In questo caso si dice che si adotta un approccio *Bottom-Up*. La scomposizione di un sistema in componenti e sottocomponenti, fino a giungere agli elementi unitari deve tener conto di due regole fondamentali: alta coesione e basso accoppiamento. La *coesione* di un componente rappresenta il grado di "coerenza funzionale" delle varie parti che lo compongono. Un'alta coesione significa che tutte le parti sono progettate per realizzare una coesione funzionale, cioè realizzare un'unica funzione. Funzioni diverse sono realizzate da componenti diversi. L'*accoppiamento* di un componente rappresenta il grado di "conoscenza logica interna" che esso ha di un altro componente. Un basso accoppiamento indica che ogni componente non ha assolutamente bisogno di conoscere alcuna logica interna ad altri componenti per svolgere le proprie funzioni. Tali caratteristiche si applicano sia ai componenti di alto livello (es.: componente di interfaccia esterna, componente di servizio, componente di trasmissione, ecc.) sia a componenti di dettaglio, fino ai singoli moduli (tale caratteristica è enfatizzata dalla tecnologia object-oriented).

### Codifica

Scrivere codice di qualità non è un'impresa difficile; richiede particolari tecniche ormai consolidate, anche se molte di esse sono state "dimenticate" con il tempo. La conoscenza approfondita dei vari *linguaggi di programmazione* è un primo prerequisito imprescindibile. L'esperienza maturata nell'utilizzo dei linguaggi adoperati è un patrimonio di grande valore sia per le persone e che per l'organizzazione stessa. Un secondo elemento di qualità è l'utilizzo di *standard di programmazione*. Questi sono un insieme di regole di comprovata utilità ed efficacia che garantiscono l'uniformità del codice scritto da persone e gruppi di persone diverse. Alcuni sono *standard relativi ai singoli linguaggi di programmazione* e la loro conoscenza è quindi inclusa nelle competenze relative ai linguaggi stessi. Altri sono *standard relativi alla documentazione del codice*. Essi definiscono cosa, come e quanto documentare all'interno del codice stesso. La documentazione del codice è importante perché fornisce le informazioni necessarie per poter intervenire sul codice in momenti successivi alla sua prima scrittura. Le informazioni riguardano la stesura originaria del codice (nome del programmatore, data di scrittura della prima versione), la struttura del codice (funzionalità realizzate, dati e struttura degli input richiesti, dati e struttura degli output prodotti, controlli effettuati, particolari algoritmi utilizzati, ogni altra informazione necessaria a capire come sia stato strutturato il codice), le successive modifiche (nome del programmatore, data e motivo delle modifiche successive). Una buona documentazione del codice richiede che essa sia circa il

30% del codice stesso. Altri elementi da tenere in considerazione è il controllo del *codice inerte* (“codice morto”) che deve essere assolutamente assente e rimosso qualora sia individuato. Tali regole definiscono spesso anche le dimensioni dei singoli moduli. Una dimensione di circa 10 linee di codice costituiva una buona regola. Altre progettazioni di singoli moduli sono tecniche ancora validissime per produrre codice di qualità. La *programmazione strutturata*, nata negli anni ‘70 e poi inglobata dalla tecnica Object-Oriented (OO), consiste in alcune regole semplici ed efficaci che permettono di scrivere codice di qualità. Tali regole, insieme a standard di documentazione del codice e principi di progettazione dei singoli moduli, costituiscono le fondamenta dell’attività di codifica per la produzione di codice di qualità.

### **Verifica e validazione del software sviluppato**

Per quanto riguarda il controllo della qualità prodotta, le revisioni tecniche ed i collaudi del software rimangono le due tecniche principali per verificare il livello qualitativo raggiunto e ricercare il maggior numero di errori presenti nella progettazione e nel codice.

Le *revisioni tecniche* si eseguono sui singoli oggetti (documenti, codice, casi di prova, ecc.) prodotti nelle diverse fasi del ciclo di sviluppo (analisi, progettazione, codifica e test). Aiutano a scoprire e a rimuovere gli errori concettuali nelle fasi alte del ciclo di sviluppo ed evitano la loro propagazione nel codice prodotto. Le revisioni tecniche verificano, per ogni oggetto ispezionato, il livello di uniformità agli standard stabiliti per le singole categorie di oggetti prodotti, la completezza, la correttezza e la coerenza con i prodotti della fase precedente.

I *collaudi* ai vari livelli (test unitario, test d’integrazione, test di sistema, test utente) rimuovono gli errori dal codice prima della sua messa in esercizio (rilascio in produzione). L’efficacia dei test dipende dal livello di copertura dei requisiti funzionali e delle caratteristiche qualitative (sicurezza, interoperabilità, usabilità, prestazioni, robustezza, ecc.). I test devono verificare non solo la correttezza delle funzionalità sviluppate ma anche la gestione delle condizioni di errore e delle condizioni di eccezione. La progettazione dei test deve includere la predisposizione di ambienti di test adeguati (componenti tecnologiche e basi di dati) il più possibile simili a quelle di esercizio.

Le attività di revisione tecnica e di collaudo del software, quindi, devono essere pianificate, eseguite e controllate per verificarne l’efficacia.

### **Assicurazione Qualità**

Perché un’organizzazione pianifichi ed esegua correttamente tutte le attività descritte sopra occorre che esse siano previste nel processo di sviluppo del software. L’elemento che assicura la qualità del software, quindi, è il controllo che i processi siano eseguiti completamente e correttamente dai progetti e che i

livelli qualitativi attesi siano stati raggiunti. A fronte dell'analisi dei risultati raggiunti si valuta quanto i processi sono seguiti dall'organizzazione e l'efficacia dei processi; quindi si interviene per rimuovere eventuali ostacoli organizzativi o errori nei processi.

Tutto ciò è compito dell'assicurazione della qualità (*SQA - Software Quality Assurance*) descritto in dettaglio nel seguito del documento.

*Queste tecniche, utilizzate dalle organizzazioni software che hanno partecipato alla sperimentazione in maniera semplice ma estesa, si sono dimostrate di grande aiuto nel raggiungimento degli obiettivi dei progetti (rispetto dei tempi e dei costi e raggiungimento del livello di qualità).*

## Un approccio metodologico concreto

Lo sviluppo del software è un'attività creativa e tecnica allo stesso tempo, perciò ad alto contenuto umano e tecnologico. La componente umana è quindi la chiave di lettura principale. La definizione dei processi permette di standardizzare le operazioni, prevedere i risultati, misurarli e migliorarli. L'utilizzo di strumenti a supporto permette di aumentare la produttività e la qualità. Questi tre elementi determinano il grado di maturità di un'organizzazione software e, da questo, il livello della qualità del software prodotto.

La qualità del prodotto software è quindi direttamente legata alla maturità dell'organizzazione che lo realizza. I fattori che determinano la maturità di un'organizzazione software sono:

1. **Competenza ed esperienza delle persone:** professionalità e motivazione delle persone che operano in un ambiente culturale aperto, innovativo, stimolante, gratificante, che premia appunto la professionalità;
2. **Maturità dei processi adoperati per lo sviluppo del software:** processi ben definiti, utilizzati da tutti e migliorati di continuo;
3. **Utilizzo di metriche, metodi, tecniche e strumenti** a supporto delle attività di sviluppo che risultino utili, semplici ed efficaci.

Ciascun elemento ha la sua importanza ed è direttamente legato agli altri due. Essi devono crescere in maniera coerente e supportarsi a vicenda. La crescita contemporanea, sinergica e coerente delle tre componenti rappresenta anche la crescita della maturità dell'organizzazione.

La figura che segue riporta graficamente i tre elementi secondo una rappresentazione comune a molta letteratura.



**Figura 2. Elementi che determinano la maturità di un'organizzazione software.**

### **Persone e ambiente culturale**

Il coinvolgimento di persone con le competenze e l'esperienza richieste è fondamentale per la riuscita dei progetti di sviluppo software. E' necessario definire per ciascun ruolo previsto le responsabilità associate e le competenze richieste. Ciò per assicurare che ciascuno sappia cosa fare, come farla e conosca ed accetti le proprie responsabilità: svolga cioè il proprio ruolo con piena consapevolezza e capacità. Ma le persone devono anche essere proattive, entusiaste del proprio lavoro ed essere riconosciute nel loro valore. Occorre quindi un ambiente culturale in cui la qualità sia un fondamento, nasca dalla volontà dell'alta direzione e coinvolga l'intera organizzazione. La qualità si sviluppa solo in ambienti adatti a promuoverla, ad accorglierla, a valorizzarla: in tutti gli altri casi è un fattore casuale, legato solo a fatti contingenti e comunque non replicabile.

### **Processi maturi**

Un processo è "maturo" quando è ben definito, utilizzato da tutta l'organizzazione e migliorato di continuo per renderlo sempre più efficace e adatto alle esigenze dei gruppi di lavoro. Un buon processo è semplice ed efficace. Esso è completo perché descrive le fasi da seguire, le attività da svolgere, le modalità e le tecniche da adottare, gli output da produrre, i ruoli da coinvolgere, le verifiche ed i controlli da eseguire, le metriche da usare e le evidenze da produrre. Un processo maturo è quindi ben documentato, conosciuto ed utilizzato dall'intera organizzazione, valutato nella sua efficacia e migliorato di continuo. Non è complesso; la semplicità è la sua migliore qualità. Un processo è efficace quando è compreso, utilizzato agevolmente e consente di

raggiungere gli obiettivi prefissati. Un processo complesso e rigido è in contrasto con la competenza, l'esperienza e la professionalità richieste alle persone coinvolte. Esso deve essere una guida che ben si adatta all'ambiente culturale nel quale si colloca. La competenza e l'esperienza delle persone permettono di utilizzarlo con profitto nelle diverse situazioni in cui ci si trova ad operare.

### **Metriche, metodi, tecniche e strumenti**

Un detto popolare dice che “un buon artigiano lo si vede dagli strumenti che adopera”. Analogamente possiamo asserire che la qualità del lavoro svolto da un ingegnere del software dipende anche dalla qualità delle metriche, dei metodi e delle tecniche che adopera, dagli ambienti su cui opera e dai tool utilizzati. L'ingegneria del software ha proposto nel tempo innumerevoli metriche, metodi e tecniche, strumenti a supporto dello sviluppo del software. Alcuni metodi e tecniche costituiscono delle *best practice* grazie ai risultati positivi dimostrati sul campo; di metriche ne esistono in grande quantità: basta sceglierne le più semplici ed efficaci. Gli strumenti (*tool*) realizzati a supporto dello sviluppo del software permettono di automatizzare le attività ripetitive, semplificare alcune attività complesse, controllare la qualità raggiunta. Un'organizzazione matura si dota di strumenti adatti al proprio ambiente di lavoro ed integrati con i processi utilizzati. A questo scopo, si fornisce nel seguito del documento una breve panoramica di alcune metriche, metodi, tecniche e strumenti da utilizzare durante le diverse fasi del ciclo di sviluppo.

*L'esperienza maturata dalle organizzazioni software che hanno partecipato alla sperimentazione di questa metodologia hanno prodotto:*

- 1. un documento firmato dalla direzione e portato a conoscenza dell'intera organizzazione che definisce le linee guida da seguire da parte di tutti i progetti (politica);*
- 2. un documento con l'elenco dei ruoli, delle responsabilità e delle competenze minime richieste; il documento è stato utilizzato per valutare il livello di competenza posseduto da ciascuna persona in base al ruolo svolto; dall'analisi della valutazione delle competenze è stato prodotto un piano di formazione per coprire le lacune evidenziate; l'implementazione del piano di formazione (in parte con attività di formazione in aula, in parte con attività di affiancamento (training on the job), in parte con attività autodidattica; la formazione è diventata un obiettivo della direzione stessa e di ciascuna persona;*
- 3. un documento con la descrizione del processo di sviluppo software completo di tutte le informazioni necessarie (fasi, attività da svolgere, input richiesti e output da produrre, ruoli coinvolti, tecniche e metodi da utilizzare, metriche da adoperare, controlli e verifiche da eseguire, strumenti da adoperare, evidenze da produrre);*
- 4. un documento con la descrizione delle metriche, i metodi e le tecniche, gli strumenti (tool e modelli) da adoperare in tutti i progetti;*
- 5. un documento con gli standard aziendali in tema di sviluppo software (standard documentale, di progettazione e di programmazione);*
- 6. un documento con la descrizione dell'organizzazione e del processo di assicurazione della qualità.*

## I principi della qualità del software

Il processo che ha l'obiettivo di assicurare la qualità in un'organizzazione software è detta "Assicurazione Qualità (*Software Quality Assurance - SQA*)". Il processo, adottato da tutte le grandi organizzazioni software come strumento di controllo e miglioramento della propria capacità produttiva, dovrebbe essere utilizzato anche dalle piccole e medie imprese con opportune semplificazioni. Il miglioramento che se ne può conseguire è in termini di:

- Aumento della qualità del software realizzato;
- Riduzione dell'intero ciclo di sviluppo;
- Aumento della produttività e conseguente riduzione dei costi.

Per raggiungere tali obiettivi si agisce su:

- *Pianificazione della qualità del prodotto software* (definizione degli obiettivi da raggiungere e pianificazione delle attività per raggiungerli);
- *Controllo della qualità del software realizzato* (controllo del livello di qualità del software sviluppato rispetto agli obiettivi definiti);
- *Controllo dell'adeguatezza (qualità) del processo utilizzato per lo sviluppo del software* (controllo del livello di aderenza al processo definito durante l'intero ciclo di sviluppo e valutazione dell'efficacia del processo adoperato).

## Responsabilità per la qualità

La qualità del software realizzato è responsabilità di tutti, nessuno escluso. Chiunque sia direttamente coinvolto in un progetto software contribuisce con il proprio lavoro a costruire (o a non costruire) la qualità del prodotto finale. Essa (la qualità) è infatti una caratteristica intrinseca del software e non il risultato miracoloso ottenuto con qualche alchimia. La si costruisce giorno per giorno con la cura e l'attenzione che poniamo nello svolgere il nostro compito.

*Questo è un principio fondamentale da cui derivano tutte le iniziative che seguono. Nell'esperienza cui si fa riferimento, la direzione di ciascuna azienda è stata promotrice di tale principio ed ha incluso nel documento relativo all'organizzazione anche il ruolo che essa deve svolgere in tema di qualità e le responsabilità associate al ruolo stesso (vedi tabella che segue).*

La tabella di seguito riporta le responsabilità associate ad ogni ruolo.

**Tabella 2. Responsabilità definite per la qualità.**

Ruolo	Responsabilità
<b>Direzione</b>	<ul style="list-style-type: none"> <li>• Assicurare che il processo di sviluppo software sia stato ben definito, compreso da tutti ed utilizzato in tutti i progetti;</li> <li>• Assicurare che il processo sia valutato in maniera oggettiva ed opportunamente migliorato per renderlo sempre più semplice, efficace e rispondente alle necessità;</li> <li>• Assegnare le responsabilità relative all'assicurazione qualità a persone indipendenti dallo sviluppo che verifichi, validi e valuti la qualità raggiunta (in organizzazioni di piccole dimensioni, può essere necessaria una sola persona appartenente, per esempio, al gruppo di test, indipendente dall'organizzazione di sviluppo);</li> <li>• Assicurare la definizione del processo per la gestione della qualità che ne preveda una fase di pianificazione ed una di controllo;</li> <li>• Assicurare la formazione dell'intera popolazione sui temi della qualità che ne faccia capire l'importanza e le relative responsabilità di ciascuno;</li> <li>• Assicurare le risorse (umane e tecnologiche) necessarie ai singoli progetti secondo la pianificazione e le stime approvate;</li> <li>• Rivedere periodicamente i risultati delle attività di QA e intraprendere opportune azioni che ne correggano le deviazioni e ne migliorino i risultati.</li> </ul>
<b>Gruppo QA</b>	<ul style="list-style-type: none"> <li>• Supportare il responsabile del progetto nella pianificazione della qualità;</li> <li>• Eseguire le revisioni di QA secondo il piano definito;</li> <li>• Rapportare la direzione nella valutazione dei risultati delle revisioni di QA;</li> <li>• Supportare l'implementazione delle azioni correttive e preventive pianificate per migliorare i processi.</li> </ul>
<b>Capo progetto</b>	<ul style="list-style-type: none"> <li>• Pianificare la qualità del progetto assicurando la disponibilità delle risorse necessarie sia realizzative che di verifica e validazione;</li> <li>• Controllare l'esecuzione di tutte le attività pianificate che assicurino il raggiungimento del livello qualitativo atteso secondo il piano della qualità prodotto;</li> <li>• Collaborare con la funzione QA in modo da agevolare le attività di revisione;</li> <li>• Concordare azioni opportune per indirizzare eventuali deviazioni dagli obiettivi di qualità attesi.</li> </ul>
<b>Gruppo di collaudo</b>	<ul style="list-style-type: none"> <li>• Assicurare una strategia di test in grado di verificare il reale livello qualitativo raggiunto in linea con la criticità e complessità dell'applicazione;</li> <li>• Pianificare i diversi livelli di collaudo (test) in base ai requisiti ed alla progettazione realizzata, assicurando il massimo della copertura con il minimo di risorse necessarie (ottimizzazione dei test);</li> <li>• Progettare scenari, casi di prova, ambienti di collaudo e basedati che garantiscano l'efficacia dei test;</li> <li>• Tracciare gli errori rilevati e verificarne la corretta rimozione;</li> <li>• Controllare la corretta esecuzione dei test verificandone lo stato di completamento e riportando periodicamente;</li> <li>• Cooperare con il gruppo di supporto coinvolgendolo attivamente nel progetto.</li> </ul>

- Gruppo di progetto**
- Definire e documentare i requisiti funzionali e quelli qualitativi (non funzionali);
  - Indirizzare nelle specifiche tecniche e funzionali anche i requisiti di qualità;
  - Realizzare la progettazione in modo da indirizzare completamente, correttamente ed efficacemente anche i requisiti di qualità;
  - Eseguire le revisioni tecniche dei documenti (requisiti, specifiche, progettazione) realizzati nelle fasi alte del ciclo di sviluppo;
  - Sviluppare il codice secondo i documenti di disegno e con alto livello di qualità;
  - Aderire a tutti gli standard applicabili (di progettazione, di programmazione, documentali, tecnologici, ecc.);
  - Tracciare gli errori rilevati in ciascuna fase del ciclo di sviluppo e risolverli nella stessa fase.

- Gruppo di supporto**
- Acquisire le competenze necessarie per utilizzare efficacemente le tecnologie necessarie per lo sviluppo dei progetti;
  - Realizzare gli ambienti di test e collaudo secondo le specifiche definite dai progetti ed in tempo utile per il loro utilizzo (secondo il piano di progetto);
  - Fornire il supporto necessario ai progetti per risolvere eventuali problemi relativi all'utilizzo delle tecnologie e degli ambienti di sviluppo e di test realizzati;
  - Partecipare proattivamente allo sviluppo dei progetti sentendosi parte integrante dei gruppi di lavoro e mai al di fuori di essi.

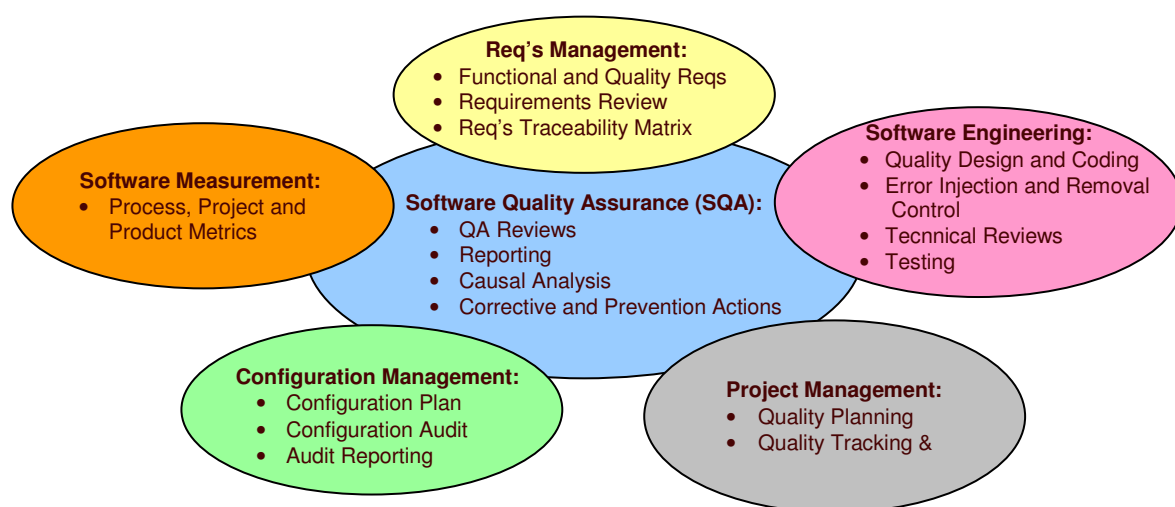
### Ambito della qualità del software

La qualità di un prodotto software coinvolge diverse aree e competenze. Ciascuna attività, se eseguita correttamente, contribuisce alla costruzione del risultato finale. Ovviamente è vero anche il contrario (Ciascuna attività, se **non** eseguita in maniera corretta, contribuisce ad inserire un numero variabile di errori che pregiudicano la qualità finale del risultato, a volte anche in maniera irreparabile!).

La figura che segue mostra le diverse aree di competenza che hanno una diretta influenza nella gestione della qualità del software:

- *Assicurazione qualità* (controllo e verifica dell'aderenza ai processi, della bontà della conduzione del progetto, del livello qualitativo raggiunto dal prodotto sviluppato);
- *Gestione dei requisiti* (raccolta, analisi, selezione, documentazione dei requisiti espliciti ed impliciti, gestione delle modifiche ai requisiti iniziali, tracciatura del livello di implementazione dei requisiti);
- *Ingegnerizzazione del software* (progettazione, sviluppo e test);

- *Gestione del progetto* (pianificazione e controllo del progetto)
- *Gestione della configurazione* (identificazione degli elementi da sottoporre alla gestione della configurazione; pianificazione delle modalità di gestione della configurazione; verifica e controllo dell'aderenza al piano; reporting);
- *Misurazioni* (definizione delle metriche da utilizzare in base agli obiettivi da raggiungere, misurazione e valutazione dei risultati).



**Figura 3. Ambito della gestione della qualità del software.**

L'area centrale rappresenta il processo per l'assicurazione della qualità mentre le aree intorno rappresentano gli altri processi tipici dello sviluppo del software.

La sovrapposizione delle aree indica che i relativi processi contribuiscono in maniera determinante alla qualità del software finale. Di seguito si fornisce una descrizione sintetica delle aree; nel capitolo successivo sono forniti maggiori dettagli e riferimenti.

### ***Assicurazione Qualità (Software Quality Assurance - SQA)***

Quest'area rappresenta il processo principale per l'assicurazione della qualità. Il processo relativo prevede le seguenti attività principali ai fini della qualità:

- Revisioni della qualità (*Quality Reviews*) dei processi, dei progetti e dei prodotti tramite attività di verifica lungo l'intero ciclo di sviluppo; le revisioni tendono a verificare il livello di aderenza ai processi, di conformità agli standard ed efficacia delle attività di revisione e collaudo del software;

- Rapporto alla direzione sull'esito delle revisioni (*Reporting*) con evidenza delle non conformità rilevate;
- Analisi causale dei progetti in modo da identificare azioni che possano migliorare i progetti attuali e futuri agendo in modo preventivo (*Causal Analysis*);
- Azioni correttive e preventive (*Corrective and Preventive Actions*) per indirizzare le non conformità rilevate, prevenirne il loro ripetersi e migliorare le performance dell'organizzazione e la qualità dei prodotti realizzati.

In sintesi, l'area di processo ha l'obiettivo di controllare che i processi definiti siano seguiti, i progetti siano pianificati e controllati, i prodotti siano realizzati per raggiungere gli obiettivi di qualità attesi. Un gruppo indipendente esegue questi controlli e riporta i risultati alla direzione. A fronte dei risultati delle revisioni sono intraprese azioni correttive, preventive e migliorative per aumentare le performance dell'organizzazione e la qualità dei prodotti. L'analisi causale permette di identificare le cause che generano i problemi e di intervenire per rimuoverli.

### ***Gestione dei requisiti (Requirements Management)***

L'area di processo rappresenta le attività necessarie per la gestione corretta dei requisiti. Il processo relativo prevede le seguenti attività importanti ai fini della qualità:

- Definizione dei requisiti funzionali e non-funzionali, detti anche requisiti di qualità (usabilità, performance, robustezza, ecc.)<sup>3</sup>;
- Validazione dei requisiti tramite incontri con gli utenti, realizzazione di prototipi, simulazioni, ecc.;
- Assicurare che i requisiti siano alla base della progettazione della soluzione, dei test per la verifica e validazione della soluzione, delle stime con cui si pianifica il progetto;
- Gestire le modifiche ai requisiti iniziali assicurando che siano di conseguenza modificate anche la progettazione, l'eventuale codice prodotto alla data, i test, le stime su cui si è pianificato il progetto;
- Costruzione e aggiornamento della Matrice di tracciabilità dei requisiti che permetta di controllare costantemente l'implementazione dei requisiti nel prodotto e massimizzare la copertura e l'efficacia dei test.

---

<sup>3</sup> La definizione completa delle caratteristiche e degli attributi di qualità del software è data dallo standard ISO 9126 ed è riportata in un successivo capitolo di questo documento.

In sintesi, la qualità del prodotto finale dipende dalla qualità con cui i requisiti sono raccolti e interpretati, sviluppati, verificati e validati.

### ***Realizzazione del software (Software Engineering)***

L'area di processo rappresenta le attività di sviluppo vero e proprio del software che include le fasi di analisi e progettazione, codifica e test, messa in esercizio (rilascio in produzione). In particolare, la progettazione e la codifica sono i momenti in cui si realizza il prodotto e si immettono innumerevoli errori, sia logici che tecnici. La fase di rimozione degli errori immessi include le attività di revisione tecnica e di testing.

In particolare, la qualità del codice deve essere controllata appena se ne inizia la stesura. Una tecnica molto valida è quella di ispezionare i primi programmi sviluppati da ciascun programmatore, verificare il livello qualitativo (strutturazione, aderenza agli standard, commenti, dimensioni, coesione, accoppiamento, ecc.) e correggere subito eventuali impostazioni errate. L'ispezione di particolari parti di codice critico (algoritmi, performance, modelli di dati, ecc.) permettono infine di verificare la correttezza della progettazione prima che le performance globali del prodotto ne possano risentire).

Il processo prevede le seguenti attività cruciali per la qualità del prodotto:

- Progettazione di un software di qualità che indirizzi in maniera esaustiva, corretta ed efficace tutti i requisiti definiti, sia quelli funzionali che qualitativi; per una progettazione di qualità si utilizzano le migliori pratiche che l'ingegneria del software definisce e consiglia per tale attività (*structured design, information hiding, top-down e bottom-up, component coesion and coupling, component-based design, reuse*);
- Produzione di codice di qualità, efficace e robusto, utilizzando le migliori pratiche che l'ingegneria del software definisce e mette a disposizione (*coesion and coupling, standards for coding, testing, and documentation, etc.*) ed aderendo a tutti gli standard applicabili (tecnologici, di programmazione, documentali);
- Controllo dell'immissione e della rimozione degli errori durante le fasi di progettazione e codifica (*Error Injection and Removal Control*) tramite l'utilizzo dei "profili di difettosità" del software relative alle diverse fasi del ciclo di sviluppo e l'utilizzo delle "curve di rimozione e saturazione dei difetti" nelle fasi di testing; i profili di difettosità e le curve di rimozione sono dedotte dai dati statistici di progetti analoghi. Nel capitolo successivo si forniscono dettagli e riferimenti su tali tecniche;
- Verifica e validazione dei prodotti intermedi realizzati nelle diverse fasi del ciclo di sviluppo (specifiche funzionali, disegno, documentazione di prodotto, codice, casi/scenari/matrici di test) tramite revisioni tecniche

interne (*Technical Reviews, Code Inspections*) e secondo il Piano di revisioni previsto; Nel capitolo che segue sono forniti dettagli e riferimenti su tale tecnica;

- Collaudo del software (*Testing*) tramite attività di verifica e validazione del codice eseguibile in ambienti di collaudo opportunamente progettati e predisposti per indirizzare con diversi livelli di test (unitario, d'integrazione, di sistema, di accettazione) le diverse caratteristiche del software, funzionali e qualitative (performance, usabilità, robustezza, installabilità, ecc.); il Piano di test e collaudo dirige le attività e le risorse necessarie. Nel capitolo seguente sono forniti dettagli e riferimenti al processo di testing.

### ***Gestione dei progetti (Project Management)***

L'area di processo indirizza la pianificazione ed il controllo del progetto. Essa prevede le seguenti attività inerenti la gestione della qualità:

- Pianificazione della qualità e stesura del *Piano della qualità* del progetto in linea con gli standard definiti al riguardo. La pianificazione prevede la definizione degli obiettivi di qualità, le metriche da applicare, i valori di soglia (*target*) da raggiungere e le attività per la realizzazione, verifica e validazione della qualità attesa (*Quality Design Techniques, Technical Reviews, Testing*);
- Pianificazione delle revisioni e stesura del *Piano delle revisioni*<sup>4</sup> con la definizione delle componenti da sottoporre a revisione (revisione di documenti, ispezione di parti di codice, revisione di scenari e casi di prova, ecc.);
- Pianificazione dei test e stesura del *Piano di test e collaudo* partendo dalla strategia di test che definisce i livelli di test (test unitari, d'integrazione, di sistema) e le tipologie di test (funzionali, di regressione, di integrazione, di usabilità, di performance, di stress, ecc.);
- Controllo periodico dello stato della qualità del prodotto tramite il controllo delle attività di progetto pianificate allo scopo e verifica dei risultati attesi rispetto a quelli attesi;
- Registrazione dei dati di progetto (*Archivio storico dei progetti*) a livello di pianificazione e di consuntivo (dimensioni, stime, indicatori, revisioni tecniche, test, ecc.). L'interpretazione e l'utilizzo dei dati permette una migliore valutazione dei fatti e decisioni più efficaci (*Fact-based*

---

<sup>4</sup> Il Piano delle revisioni può essere parte integrante (un capitolo) del Piano di test e collaudo oppure un piano separato.

*Decision*). Costituisce una base per le stime successive di progetti analoghi. E' un patrimonio di enorme valore per l'organizzazione.

### ***Gestione della configurazione (Configuration Management)***

L'area di processo rappresenta la gestione della configurazione. Il processo relativo prevede le seguenti attività:

- Identificazione degli elementi da sottoporre al controllo della configurazione, definizione delle modalità di gestione della configurazione e stesura del *Piano della configurazione*;
- Registrazione dello stato degli elementi da tenere sotto il controllo della configurazione (codice e documentazione) utilizzando gli strumenti messi a disposizione (*Giornale della configurazione* realizzato manualmente o con uno strumento specifico);
- Revisioni periodiche (*Audit*) dello stato della configurazione e stesura dei rapporti (*Reporting*) da inoltrare ai responsabili dei progetti ed alla direzione per opportune valutazioni e azioni.

### ***Misurazioni del software (Software Measurement)***

L'area di processo indirizza le misurazioni relative all'organizzazione, i processi, i progetti ed i prodotti realizzati. Il tema delle misurazioni è molto critico e reso complesso dall'adozione, molto spesso, di troppe metriche, alcune anche poco utili ai fini pratici del controllo della qualità. Occorre perciò utilizzare poche metriche, semplici, efficaci e quindi "utili". Un secondo aspetto altrettanto importante riguarda il momento in cui le misurazioni sono effettuate. Occorre misurare la qualità man mano che si sviluppa il prodotto (qualità dei requisiti e delle specifiche, qualità della progettazione, qualità del codice, copertura ed efficacia dei test). Misurare a posteriore la qualità del prodotto finale è poco utile in quanto è spesso impossibile rimediare ad eventuali livelli non soddisfacenti.

Nel paragrafo seguente sono presentate alcune metriche utilizzate con successo presso le aziende che hanno partecipato alla sperimentazione della metodologia. Il processo relativo alle misurazioni prevede le seguenti attività:

- Definizione delle *metriche* necessarie per valutare la qualità dei processi, dei progetti e dei prodotti partendo dagli *obiettivi* e selezionando gli indicatori più efficaci;
- Pianificare il *programma di misure* da realizzare (cosa, come, chi e quando misurare) e le modalità di reporting e valutazione. Il piano delle misure è documentato nel Piano della qualità;

- Esecuzione delle misurazioni durante il ciclo di sviluppo, produzione della reportistica, valutazione dei risultati e predisposizione di eventuali azioni necessarie in caso di scostamento dai valori attesi.

### Alcuni esempi di metriche del software

*Le aziende che hanno partecipato alla sperimentazione della metodologia hanno sperimentato con successo un insieme di metriche semplici ed efficaci. Le metriche sono state scelte partendo dagli obiettivi di business della direzione:*

<b>Obiettivo</b>	<b>Descrizione</b>
O1	Capacità dell'organizzazione di far fronte alle esigenze dei progetti garantendo processi, mezzi e risorse adeguati;
O2	Capacità dei progetti di rispettare i tempi, i costi ed il livello di qualità atteso;
O3	Capacità di pianificare i progetti con stime corrette e realistiche;
O4	Capacità di controllare puntualmente lo stato di andamento dei progetti e di reagire prontamente nei casi di deviazione dai piani;
O5	Capacità dell'organizzazione di sviluppare prodotti con la qualità attesa;
O6	Capacità di migliorare i processi per aumentare le capacità descritte sopra.

La tabella che segue elenca l'insieme di metriche sperimentate dalle aziende che hanno partecipato alla sperimentazione. A queste si possono aggiungere altre di particolare necessità nei contesti diversi.

**Tabella 1. Metriche del software.**

<b>Categoria</b>	<b>Metrica</b>	<b>Cosa si misura</b>
<b>Metriche relative all'organizzazione (O1)</b>	Adeguatezza della struttura organizzativa alle esigenze dei progetti (risorse)	Disponibilità delle risorse pianificate (numero e permanenza nel progetto).
		Adeguatezza delle risorse disponibili (competenza, esperienza, produttività).
		Rilievi fatti dal cliente sulle persone.
<b>Metriche di Processo (O6)</b>	Aderenza al processo (da parte dei progetti)	Numero di non conformità relative ai progetti che non seguono i processi stabiliti.
	Efficacia dei processi	Numero di non conformità relative ad obiettivi dei progetti non raggiunti e da addebitare ai processi stabiliti (es.: rigidità dei processi e scarsa flessibilità ad adattarsi alle esigenze specifiche del progetto, metodi di stima non adeguati, dati statistici incompleti o errati, checklist incomplete o errate, template

## Sviluppo di Software Applicativo

---

incompleti o errati, ecc.).

<b>Metriche di Progetto</b> (O2, O4)	Rispetto dei tempi di consegna	Numero di giorni di ritardo rispetto ai piani delle consegne <sup>5</sup> .
(O3, O4)	Rispetto dei costi	Scostamento percentuale dei costi a consuntivo rispetto ai preventivi <sup>6</sup> .
(O5, O4)	Raggiungimento della qualità	Rilascio dei deliverable previsti (documenti, piani, formazione, manuali, ecc.).  Qualità dei deliverable (numero di difetti riscontrati durante le revisioni tecniche e rilevate dai clienti alla consegna).
<b>Metriche di Prodotto</b> (O5)	Completezza e correttezza dell'interpretazione dei requisiti	Numero di errori rilevati durante l'intero ciclo di sviluppo ed imputabili ad una incompleta o non corretta interpretazione dei requisiti.
- <i>Analisi</i>		
- <i>Progettazione</i>	Qualità della progettazione	Livello di strutturazione dell'architettura applicativa.  Livello di coesione dei componenti applicativi.  Livello di accoppiamento dei componenti applicativi.  Livello di strutturazione dei dati ( <i>information hiding</i> ).  Livello di aderenza agli standard tecnologici richiesti (OO, CORBA, SOA, SOAP, ecc.).
- <i>Codifica</i>	Qualità del software	Livello di aderenza del codice agli standard di codifica (strutturazione, commenti, dimensioni, ecc.).
- <i>Rilascio</i>	Difettosità residua (del software rilasciato in esercizio)	Numero di difetti rilasciati in produzione <sup>7</sup> .

<sup>5</sup> E' importante analizzare le cause dei ritardi per verificare se il problema è stato generato da stime iniziali non corrette, da una pianificazione non realistica o da una conduzione non adeguata.

<sup>6</sup> Nel caso di scostamento dei costi consuntivi rispetto ai piani occorre distinguere se esso (lo scostamento) sia dovuto ad una errata stima iniziale oppure ad uno "sforamento" vero e proprio del budget.

## Sviluppo di Software Applicativo

---

- <i>Tutte le fasi</i>	Qualità della documentazione (tecnica e manuali) prodotta	Completezza della documentazione (documenti prodotti rispetto a quelli previsti).  Aderenza agli standard documentali applicabili (numero di anomalie rilevate).  Correttezza della documentazione (numero di anomalie rilevate).
- <i>Test e collaudo</i>	Efficacia dei test	Livello di copertura dei test rispetto ai requisiti (dalla Matrice di tracciabilità dei requisiti).  Numero di errori rilevati rispetto al numero previsto (ricavato dalla curva di rimozione degli errori).  Numero di errori rilevati per caso di test.

Se richieste, saranno rese disponibili le descrizioni di dettaglio delle metriche e delle modalità di utilizzo.

## Le attività di gestione della qualità

Di seguito sono descritte le attività principali per la gestione della qualità del software:

- Pianificazione della qualità del progetto;
- Controllo della qualità del progetto;
- Revisioni dell'Assicurazione della qualità (*QA Review*);
- Revisioni tecniche (*Peer Reviews*);
- Collaudi (*Testing*).

---

<sup>7</sup> Il numero dei difetti (errori) residui nel software messo in produzione e rilasciato in esercizio è calcolato estrapolandolo dall'andamento della curva di rimozione dei difetti nelle varie fasi del ciclo di sviluppo, con le revisioni tecniche prima ed i test dopo. L'andamento della curva mostra infatti quale sarebbe il numero di errori rimossi nella fase successiva a quella dei test e del collaudo finale: tale fase coincide proprio con l'utilizzo del software da parte degli utenti. Presupposto per l'utilizzo delle curve è quella della registrazione dei difetti rilevati in tutte le fasi del ciclo di sviluppo ed una corretta progettazione ed esecuzione dei test.

### Pianificazione della qualità del progetto

Scopo dell'attività di "pianificazione della qualità" è quello di assicurare che ogni progetto abbia definiti i suoi obiettivi di qualità, le relative metriche, i valori di soglia (valori target da raggiungere), le attività necessarie per l'implementazione ed il controllo della qualità attesa. L'attività include la definizione delle responsabilità di ciascun ruolo coinvolto nel progetto.

La formalizzazione è fatta nel "Piano della qualità del progetto". Esistono diversi standard per la stesura del piano di qualità (il CNIPA, ad esempio, ha definito lo standard per i progetti sviluppati nella Pubblica Amministrazione).

*Nella sperimentazione le aziende hanno deciso di optare per la creazione di un documento snello da produrre separatamente o da includere nel Piano di progetto come una sua parte (capitolo "Qualità").*

*In un successivo capitolo si fornisce una descrizione della tabella dello schema del documento (o del capitolo da includere nel Piano di progetto).*

In fase di analisi sono identificati e descritti anche i requisiti non funzionali (detti anche requisiti di qualità) per il prodotto da sviluppare. L'attività consiste nel definire, per ogni requisito non funzionale, una descrizione chiara, l'importanza e la priorità, l'impatto sul progetto, le attività necessarie per l'implementazione ed i controlli da eseguire. Ad ogni requisito non funzionale è associata una metrica ed un valore soglia. Per la definizione di quali requisiti non funzionali siano applicabili al singolo progetto è di aiuto la norma *ISO 9126* riportata in un apposito capitolo del presente documento. Essa definisce le *caratteristiche* e gli *attributi* della qualità del software (funzionalità, affidabilità, usabilità, efficienza, manutenibilità, portabilità). Tra le attività specifiche per implementare la qualità del software in un progetto ricordiamo: la validazione dei requisiti, la revisione del disegno, l'utilizzo di prototipi per le interfacce utente e l'esecuzione di test specifici (funzionali, di integrazione, di sistema: installabilità, performance, affidabilità, usabilità, portabilità).

Gli obiettivi di qualità, ricavati direttamente dai requisiti non funzionali come visto al punto precedente, sono documentati dal capo progetto nel "Piano della qualità" del progetto.

*Nella sperimentazione le aziende hanno scelto di documentare con una tabella gli obiettivi di qualità, le metriche, i valori di soglia, le attività per l'implementazione della qualità, le modalità di misurazione e di rendicontazione.*

*In un successivo capitolo si fornisce una descrizione della tabella.*

Il Piano della qualità di progetto è rivisto dal gruppo di QA ed approvato dalla direzione quale garante verso l'esterno (i clienti ed il mercato).

Le attività definite per il raggiungimento del livello di qualità richiesto sono assegnate alle singole persone del progetto e controllate accuratamente dal capo

progetto che ne dovrà rendere conto alla direzione in fase di valutazione prima del rilascio del prodotto.

### **Controllo della qualità del progetto**

Scopo dell'attività è quello di verificare il livello di qualità realmente raggiunto dai progetti. Il controllo è svolto direttamente dal capo progetto, durante l'intero ciclo di sviluppo, con il supporto delle persone coinvolte nel progetto stesso.

Di seguito sono descritte le singole attività con i ruoli coinvolti.

Il capo progetto controlla i risultati delle attività di verifica e validazione della qualità realizzata (validazione dei requisiti, revisione delle specifiche e del disegno, validazione dei prototipi delle interfacce, risultati dei test) e li documenta aggiornando il Piano della qualità (o l'apposita sezione del Piano di progetto).

La registrazione dei risultati delle revisione e dei test permette di controllare statisticamente il progressivo andamento del livello di qualità e di prevedere in ogni momento la tendenza ed i valori finali. A tale scopo sono utilizzate le curve di rimozione degli errori che, costruite sui dati storici dei vari progetti, evidenziano gli scostamenti e permettono di fare previsioni.

Durante le fasi del ciclo di sviluppo e al termine del progetto (e comunque sempre prima del rilascio del prodotto/applicazione al mercato/cliente) il gruppo di QA valuta il livello di qualità finale raggiunto e concorda con il capo progetto eventuali azioni correttive in caso di scostamenti dai valori attesi. In ogni caso, è sempre la direzione ad autorizzare il rilascio del software in produzione (o al cliente/mercato).

### **Revisioni dell'Assicurazione della qualità (QA Review)**

Scopo delle revisioni di QA (*QA Review*) è quello di garantire che i vari progetti software abbiano definito i propri piani, adoperino efficacemente il processo di sviluppo definito in azienda e misurino i risultati di qualità raggiunti rispetto agli obiettivi attesi. L'attività prevede anche la valutazione del livello qualitativo finale raggiunto dal prodotto software realizzato. Il risultato delle revisioni di QA è documentato in un rapporto alla direzione. La direzione potrà così valutare periodicamente il livello qualitativo dei progetti ed intervenire con opportune azioni. Anche l'efficacia del processo è valutata ed il processo migliorato, se necessario. Segue una breve descrizione delle attività principali ed i ruoli coinvolti.

La direzione, in linea con la “Politica per la qualità” definita a livello aziendale, assicura la definizione di un “Processo di sviluppo software” in azienda in tutti i suoi dettagli (ruoli, procedure, strumenti), la sua documentazione, la diffusione e le modalità per la valutazione del grado di utilizzo nei progetti e la sua efficacia. A tale scopo, costituisce un gruppo di QA, indipendente dall’organizzazione di sviluppo, dotato di un proprio processo e con la responsabilità di assicurare la qualità del software prodotto in azienda e di rapportare la direzione sull’esito di tali attività.

Ai capi progetto è assegnata, invece, la responsabilità di seguire tutti i processi di sviluppo definiti in azienda. In particolare, di pianificare la qualità per i propri progetti e di controllarne la realizzazione, seguendo gli standard ed il processo definiti in azienda.

*Le aziende che hanno sperimentato la metodologia hanno concordato la seguente modalità per la gestione delle eventuali deviazioni dal processo, dagli standard e dai livelli di qualità attesi.*

- *Il capo progetto discute le deviazioni richieste con il gruppo di QA per valutarne gli impatti e concordare le modalità di implementazione;*
- *La direzione approva le deviazioni;*
- *Il gruppo di QA verifica il presentarsi di eventuali impatti negativi sul progetto causati dalle deviazioni durante le fasi del ciclo di sviluppo; in questo caso si analizzano le conseguenze e si concordano opportune azioni correttive;*
- *Al termine del progetto di valuta globalmente l’impatto delle deviazioni e si decide se evitarle in futuro.*

La direzione valuta la qualità finale raggiunta da ciascun progetto e ne autorizza il rilascio. Periodicamente (almeno due volte all’anno) valuta anche le attività di revisione della qualità svolte dal gruppo di QA e sulla base delle risultanze decide quali azioni intraprendere per migliorare il processo di sviluppo del software.

Le iniziative intraprese, documentate dai verbali delle revisioni, sono controllate per verificarne l’efficacia (i verbali sono conservati come “documenti di registrazione della qualità”).

*Le aziende che hanno partecipato alla sperimentazione, tutte organizzazioni particolarmente semplici per dimensione e struttura organizzativa, hanno previsto le revisioni della qualità (QA Review) elencate di seguito e descritte a seguire:*

- *Revisione dell’offerta (QA1);*
- *Revisione della pianificazione (QA2);*
- *Revisione del rilascio (QA3);*

*In base alle verifiche effettuate si valuta il rischio del progetto (basso, medio, alto), si identificano le azioni necessarie per contenerlo e si decide il proseguimento del progetto nelle fasi successive.*

### ***Revisione dell'offerta (QA1)***

In questa fase si controlla che

- l'offerta sia stata redatta utilizzando le modalità e gli schemi previsti,
- i requisiti del clienti siano stati definiti e verificati,
- la soluzione proposta sia fattibile tecnicamente,
- il piano di massima sia ragionevole (tempi, attività principali, *milestone*),
- gli aspetti economici (prezzo, costi e profitto) siano in linea con le politiche (*policy*) aziendali e calcolati secondo le procedure di stima definite,
- le risorse siano state individuate in linea di massima,
- i fornitori esterni da utilizzare siano stati individuati,
- i rischi siano stati individuati, valutati e opportunamente indirizzati.

### ***Revisione della pianificazione (QA2)***

In questa fase si controlla che

- i piani (di progetto, della qualità, delle revisioni, dei test, della configurazione) siano stati redatti secondo le procedure definite (modalità e modelli),
- i piani siano ragionevoli (tempi, attività, *milestone*, dipendenze, ecc.),
- le risorse identificate siano disponibili quando richieste,
- sia stata verificata la reale disponibilità dei fornitori esterni da coinvolgere,
- i rischi individuati siano stati ulteriormente indirizzati.

### ***Revisione del rilascio (QA3)***

In questa fase si controlla che

- le attività previste dal piano di progetto siano state completate con successo,

- i prodotti intermedi previsti (codice, documentazione) siano stati realizzati secondo gli standard e tenuti sotto il controllo della configurazione,
- gli obiettivi di qualità definiti e documentati nel Piano della qualità siano stati raggiunti.

Ogni *QA Review* è documentata in un rapporto secondo le modalità ed i modelli definiti.

*Le aziende hanno concordato un Rapporto di QA con le seguenti informazioni: data e tipo di revisione, nome di progetto, responsabili di progetto, revisori QA, oggetti revisionati, esito della revisione, valutazione del rischio, azioni concordate, data prevista di completamento delle azioni concordate.*

Alle scadenze previste si verificano le attività concordate per il contenimento del rischio; l'esito delle verifiche è registrato nel rapporto originale o in uno nuovo rapporto.

I rapporti delle Revisioni QA sono registrati e mantenuti dal gruppo di QA.

Il Sommario dell'esito delle Revisioni QA sono oggetto di presentazione alla direzione.

### **Revisioni tecniche (Technical Reviews)**

Le revisioni tecniche (dette anche *ispezioni*) si applicano a tutti gli artefatti prodotti (documenti e codice) durante lo svolgimento del progetto per verificarne la completezza e la correttezza, sia nei contenuti che nella forma. Questa attività è di grande importanza in quanto è l'unica tecnica disponibile per verificare e validare il prodotto man mano durante la sua realizzazione. Essa consente di individuare e di rimuovere gli errori introdotti direttamente nelle singole fasi di sviluppo evitando così che essi si propaghino alle fasi successive. Per quanto concerne il codice, la tecnica tende a verificare il suo livello qualitativo in termini di aderenza a standard (standard di programmazione) e qualità intrinseca (strutturazione, leggibilità, quantità e qualità dei commenti, livello di modularità, di coesione e di accoppiamento, ecc.). Un altro scopo è quello di verificare la correttezza di parti di codice particolarmente critiche (per esempio, realizzazione di algoritmi particolari, ecc.). Essendo onerosa, l'attività di ispezione del codice è eseguita su pochi programmi, scritti da ciascun programmatore, per verificarne l'omogeneità all'interno dei diversi gruppi di sviluppo; in altri casi per validare parti di codice particolarmente critiche. Le attività di revisione sono anche dette di *test statico* (cioè senza che ci sia del codice da eseguire).

L'utilizzo metodico di tale pratica costituisce una *best practice* di enorme valore.

I documenti da revisionare sono principalmente:

- documenti di progetto (piano di progetto, piano di qualità, piano di test, piano delle revisioni, piano di gestione della configurazione);
- documenti di prodotto (requisiti, specifiche funzionalità, disegno, casi d'uso, scenari e casi di test, matrice di test);
- il codice (inteso come documento);

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

- sia aderente agli standard definiti (es.: sia stato scritto usando un formato stabilito);
- sia completo in tutte le sue parti richieste (così come previsto dallo schema base disponibile);
- sia corretto nei contenuti (es.: il disegno sia tecnicamente corretto);
- indirizzi i requisiti attesi (es.: il disegno indirizzi i requisiti del progetto e non altri).

Esistono diversi tipi di revisioni tecniche, a seconda dell'obiettivo. I principali sono:

- *Revisione/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). Particolarmente efficace è la tecnica della "revisione strutturata" (o *Peer Review*) descritta in dettaglio di seguito.
- *Walkthrough*: consiste nell'ispezione di documenti tecnici di rilievo (documento di disegno, codice, ecc.) fatta in maniera particolare: i revisori, generalmente molto esperti della materia trattata (analisti esperti 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 verificarne che esso possa essere eseguito correttamente dal computer.

Le revisioni possono essere eseguite in maniera:

- *Informale*: le revisioni sono eseguite dalle persone tecniche competenti quando l'autore di un documento lo ritiene necessario, ispezionando i

documenti richiesti, anche parzialmente, spesso riunendosi direttamente con l'autore del documento, lasciando all'autore la discrezionalità di prendere gli appunti necessari e senza produrre rapporti o verbali ufficiali;

- *Formale*: le revisioni sono pianificate dal capo progetto, sono preparate, eseguite e documentate secondo un formalismo stabilito (vedi *Revisione Strutturata*) ed i risultati sono controllate dal capo progetto per verificarne l'efficacia.

*Le aziende coinvolte nella sperimentazione hanno deciso di adottare in maniera estesa la tecnica delle revisioni "informali" a tutti i documenti prodotti e alcuni programmi a campione scritti da ciascun programmatore. La decisione di utilizzare revisioni "informali" tende a compensare il sovraccarico di lavoro previsto per tali attività non eseguite precedentemente.*

**Nota:** la tecnica particolare delle revisioni paritetiche (*Peer Review*) è in linea con il modello di maturità CMMI.

### ***Alcuni elementi importanti delle revisioni***

Per garantire l'efficacia delle revisioni è assolutamente necessario applicare le seguenti regole e seguire i suggerimenti dati:

- 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 sulle parti maggiormente 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 da personale competente;
- La revisione deve essere pianificata per consentire alla persona coinvolta di dedicare il tempo necessario alle attività di revisione (preparazione e riunione di revisione) compatibilmente con i suoi impegni;
- La revisione ha lo scopo di scoprire eventuali problemi nella progettazione prima di iniziare lo sviluppo e non quello di giudicare il lavoro o la persona;
- Gli eventuali problemi riscontrati sono discussi con l'autore che provvede a correggerli, se opportuno.

### ***Checklist per le revisioni tecniche***

Ogni organizzazione (o gruppo di lavoro) ha caratteristiche proprie ben definite e legate a diversi fattori:

- ambiente culturale nel quale opera (*livello di formalizzazione dei processi, attitudine all'innovazione, controllo del lavoro, livello di delega, motivazione delle persone, ecc.*),
- proprie capacità (*competenza ed esperienza*),
- ciclo di vita adoperato (*a cascata, iterativo-incrementale, prototipale, RAD ecc.*),
- dimensione e complessità del progetto (*piccola, media, alta*),
- ambienti di sviluppo e tecnologie adoperate (*sistemi legacy, sistemi multi-tier, sistemi real-time, sistemi Web, tecnologie OO, linguaggi di nuova generazione ecc.*).

L'insieme di tutti questi fattori fa sì che un gruppo di lavoro tenda ad operare nel tempo in maniera sempre più conforme alle proprie attitudini ed abitudini. Si vengono così a creare “mentalità” specifiche che si traducono in “esperienza” e poi in “competenza”; questa a sua volta determina “scelte” progettuali di sicuro successo perché corroborate dagli esempi positivi.

**Aspetti positivi:** tale processo culturale crea “uniformità, tendenza, certezza, coesione” nel gruppo e questa, a sua volta, produce risultati di qualità.

**Aspetti negativi:** tale contesto consente di ripetere (e a volte perseverare in) gli errori senza accorgersene.

Le “liste di controllo (*checklist*)” costituiscono un valido strumento che fa tesoro di entrambi i punti di vista. Esse riportano ciò che si deve perché considerato positivo e cosa è da evitare perché considerato negativo in base all'esperienza maturata nel tempo. Le *checklist* sono quindi realizzate dagli esperti dei processi in base all'esperienza e mantenute aggiornate nel tempo man mano che le “abitudini” dei gruppi di lavoro cambiano. Con l'adozione delle *checklist* i gruppi di lavoro imparano a non ripetere più alcuni errori ricorrenti ma ne commettono altri legati all'adozione di nuovi processi e tecnologie.

Nell'ambito delle revisioni tecniche le *checklist* acquistano un'importanza fondamentale in quanto suggeriscono ai revisori quali controlli effettuare. Esistono perciò diverse *checklist* adatte alle diverse fasi del ciclo di sviluppo e relative alla revisione di diversi documenti.

In linea di principio, le *checklist* utilizzate nelle revisioni tecniche elencano gli elementi da verificare secondo tre tipologie diverse di controllo:

- *Controlli formali:* si verifica il livello di completezza del documento e la sua aderenza agli standard applicabili (documentali, di programmazione, tecnologici);

- *Controlli di congruità*: si verifica che il documento sia congruente con quanto lo precede o lo affianca nella fase (congruenza con altri documenti cui di fa o si deve fare riferimento, congruenza con criteri di ingresso o uscita di fase ecc.);
- *Controlli di correttezza*: si verifica la chiarezza, la completezza e la correttezza dei concetti tecnici (e non) espressi nel documento. Per esempio, se si tratta della revisione del documento dei requisiti, si verifica che tutti i requisiti documentati risultino “chiari, esaustivi e testabili”; se si tratta invece di un documento di progettazione, si verifica che essa sia corretta dal punto di vista tecnico, che utilizzi correttamente le tecnologie, che indirizzi tutti i requisiti dichiarati ecc.

Gli elementi inseriti nelle liste dipendono da caso a caso e sono aggiornati in base all'esperienza.

*Le aziende che hanno partecipato alla sperimentazione hanno costituito un gruppo di lavoro per la redazione delle liste di controllo, partendo da checklist esistenti e adattandole alle proprie esigenze in base alle proprie esperienze.*

### Collaudi (Testing)

Non vogliamo qui descrivere la metodologia di test ma evidenziare un aspetto molto importante relativo alla gestione della qualità di cui si sta discutendo.

Il test è la seconda attività fondamentale per scoprire gli errori nel codice. Dalla sua efficacia dipende la possibilità di rilasciare il software con il livello di qualità atteso. Una buona progettazione dei test permette di rimuovere almeno il 50% degli errori presenti nel codice. Occorre quindi una strategia di test in sequenza (unitari, d'integrazione e di sistema) per ridurre drasticamente gli errori presenti al termine della fase di codifica. Partendo, per esempio, da un tasso di difettosità presunta del codice pari a 10 errori per mille linee di codice (corrispondenti a circa 100 punti funzione), il test unitario può dimezzare il tasso portandolo al valore di 5 errori, il test di integrazione a 2,5 ed il test di sistema a 1,25.

Ciò presuppone una corretta strategia di test ed una efficace progettazione degli stessi. In particolare, il test di sistema includerà test specifici per verificare le diverse caratteristiche di qualità definite (usabilità, performance, stress, ecc.) in relazione ai valori target degli obiettivi.

Tutto ciò, come si intuisce facilmente, è dispendioso in termini di risorse e di tempo. Occorre, dunque, che le attività di revisione tecnica eseguite nelle fasi progettuali precedenti siano le più efficaci in modo da ridurre quanto più possibile la soglia di difettosità da cui partono le attività di test. L'esecuzione di

revisioni tecniche poco efficaci che dovessero produrre un codice con tasso di errore superiore a quello ipotizzato nell'esempio numerico precedente (per es.: 20 errori per Klocs invece di 10) richiederebbero uno sforzo maggiore (e quindi costi maggiori) per le attività di test. Nel caso in cui il piano di test non sia aggiornato con i risultati delle revisioni tecniche, si arriverebbe a rilasciare in produzione codice con difettosità nettamente superiore e con costi di manutenzione successiva ancora maggiori.

La domanda che sorge spontanea è la seguente: “Perché non condurre revisioni tecniche efficaci, sempre?”. La risposta spesso manca!

## La qualità nella manutenzione del software

Una volta rilasciato in esercizio (messo in produzione), il software è soggetto ad una serie di modifiche richieste per diversi motivi: rimuovere gli errori rimasti nel codice e rilevati dagli utenti durante le loro attività operative (*manutenzione correttiva*); adeguare il software a nuove caratteristiche tecniche, di mercato, norme di legge ecc. (*manutenzione adeguativa*); migliorare le caratteristiche particolari del software, come le performance, la robustezza, ecc. (*manutenzione migliorativa o perfettiva*); sviluppare, infine, nuove esigenze funzionali (*manutenzione evolutiva*).

Una buona manutenzione applicativa (correttiva o evolutiva, perfettiva che sia) ha come prerequisito che siano disponibili:

- una buona documentazione del software da mantenere,
- standard di manutenzione della documentazione e del codice,
- competenze tecniche adeguate al livello di qualità desiderato (linguaggi di programmazione usati, piattaforme applicative e di sviluppo, sistemi delle base dati, tecniche e metodologie di sviluppo e manutenzione del software),
- procedure e strumenti per la gestione delle richieste di modifica, delle modifiche stesse e della configurazione,
- procedure e strumenti per la gestione delle anomalie.

**Nota:** Il tema della qualità nella manutenzione del software sarà trattato nel dettaglio in una versione successiva del documento.

## Metodi, tecniche e strumenti a supporto

Diversi sono i metodi, le tecniche e gli strumenti che dovrebbero essere sempre disponibili (ed utilizzati!) in ogni organizzazione software. In particolare, per quanto concerne la qualità del software, ricordiamo:

### Modelli standard per la qualità:

- Modelli per la stesura dei documenti di progetto (Piano della qualità del progetto, Piano delle revisioni, Piano dei test, Piano per la gestione della configurazione, Piano delle Revisioni SQA);
- Modelli per la rendicontazione dei risultati delle attività di controllo della qualità (Rapporto delle revisioni tecniche, Rapporti sull'esito dei test, Rapporto sull'esito delle revisioni (*Audit*) di configurazione, Rapporto sull'esito delle revisioni SQA);
- Modelli per la stesura dei documenti tecnici (Requisiti, Specifiche funzionali e tecniche, Progettazione, Casi di prova, Scenari di prova, Matrice di test, Matrice di tracciabilità dei requisiti);
- Modelli per la stesura dei manuali previsti (Manuale utente, Manuale di installazione, Manuale operativo ecc.);

### Checklist per la qualità:

- Checklist di revisione tecnica (una per ogni tipo di revisione);

### Tecniche e metodi per la qualità:

- Revisione tecnica dei documenti;
- Registrazione degli errori rilevati durante tutte le fasi del ciclo di sviluppo (revisioni tecniche nelle fasi alte e test del codice eseguibile);
- Registrazione degli errori rilevati nelle fasi di test e loro stato di risoluzione;
- Profilo della qualità del progetto (curva del tasso di rimozione degli errori durante l'intero ciclo di sviluppo);
- Curva di rimozione degli errori durante i test (detta anche curva di saturazione dei test);
- Tabella "ortogonale" degli errori rilevati nelle fasi di sviluppo;

- Archivio storico dei progetti, con registrazione delle informazioni di dettaglio relative ai progetti per permettere l'analisi statistica di alcune metriche;
- Analisi causale dei risultati dei progetti (e di alcune attività critiche) e delle attività di revisione e controllo allo scopo di identificare gli elementi di valore dei processi e le carenze da indirizzare con opportune azioni.

### **Standard per la qualità:**

- Standard ISO9126;
- Standard di nomenclatura degli oggetti (documenti, codice, procedure, elementi di configurazione ecc.);
- Standard documentali (modalità di stesura, revisione, approvazione, aggiornamento, emissione e ritiro della documentazione);
- Standard di progettazione (progettazione strutturata, progettazione OO, coesione e accoppiamento delle componenti progettate ecc.);
- Standard di programmazione (strutturazione del codice, commenti all'interno del codice, dimensione dei programmi, utilizzo di nomenclature interne ecc.);
- Standard di usabilità (standard interni ed esterni di usabilità);
- Standard tecnologici vari ecc.(UML, OO, SOA/SOAP, CORBA ecc.);

Ogni azienda deve produrre ed utilizzare i propri standard e mantenerli aggiornati con l'evolversi delle tecnologie e dei propri processi produttivi e di controllo.

Di seguito sono descritti alcune tecniche e metodi di facile utilizzo e grande efficacia per quanto riguarda il controllo della qualità del software.

## ISO9126

Le norme ISO 9126 stabiliscono le caratteristiche e gli attributi del software.

**Tabella 3. Caratteristiche ed attributi del software (ISO9126).**

Caratteristica	Attributi
<p><b>Funzionalità</b>  <i>“Il software fa quello che ci si aspetta debba fare”</i></p>	<ul style="list-style-type: none"> <li>• Completezza: delle funzioni offerte verso i requisiti;</li> <li>• Accuratezza: con la quale le funzioni sono offerte;</li> <li>• Interoperabilità: con gli altri sistemi presenti nell’ambiente nel quale la soluzione opererà;</li> <li>• Aderenza: a normative, leggi, regole, standard, ecc.;</li> <li>• Sicurezza: dei dati e delle persone.</li> </ul>
<p><b>Affidabilità:</b>  <i>“Il software reagisce positivamente alle variazioni dell’ambiente circostante”</i></p>	<ul style="list-style-type: none"> <li>• Maturità: livello di assestamento del sistema;</li> <li>• Tolleranza ai guasti: margini entro i quali le prestazioni sono fornite;</li> <li>• Recuperabilità: capacità di ripristinare la situazione originale in caso di caduta o degrado.</li> </ul>
<p><b>Usabilità:</b>  <i>“Il software gestisce l’interazione con gli utenti in modo ottimale”</i></p>	<ul style="list-style-type: none"> <li>• Comprensibilità: facilità di capire e intuire le funzionalità disponibili nel sistema;</li> <li>• Apprendibilità: facilità di imparare e ricordare le funzionalità offerte;</li> <li>• Operabilità: facilità di adoperare le funzioni rese disponibili.</li> </ul>
<p><b>Efficienza:</b>  <i>“Il software usa bene le risorse disponibili”</i></p>	<ul style="list-style-type: none"> <li>• Rispetto al tempo: tempo di risposta;</li> <li>• Rispetto alle risorse: utilizzo di risorse.</li> </ul>
<p><b>Manutenibilità:</b>  <i>“Il software segue l’evoluzione della organizzazione”</i></p>	<ul style="list-style-type: none"> <li>• Analizzabilità: facilità a diagnosticare i problemi, detta anche “problem determination”;</li> <li>• Modificabilità: facilità ad inserire nel software esistente le modifiche richieste;</li> <li>• Stabilità: capacità del software esistente a rimanere stabile anche dopo le modifiche;</li> <li>• Provabilità: facilità di eseguire test alle modifiche coinvolgendo poco le funzioni esistenti.</li> </ul>
<p><b>Portabilità:</b>  <i>“Il software segue l’evoluzione tecnologica”</i></p>	<ul style="list-style-type: none"> <li>• Adattabilità: al nuovo sistema;</li> <li>• Installabilità: facilità di installazione nel nuovo sistema;</li> <li>• Conformità: alle regole del nuovo sistema;</li> <li>• Sostituibilità: di un prodotto/versione precedente.</li> </ul>

- Qualità in uso:** • Efficacia: completezza nel raggiungimento degli obiettivi;  
“*Il software adempie alle esigenze degli utenti*” • Produttività: impegno delle risorse umane durante l'utilizzo;  
• Sicurezza: limitazione dei rischi alle persone;  
**Nota:** in attesa di pubblicazione • Soddisfazione: capacità di soddisfare gli utilizzatori.

### Revisione strutturata (*Peer Review*<sup>8</sup>)

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 specifiche sull'argomento trattato. Svolge anche il ruolo di revisore;
- *Revisore/Ispettore:* chi ha competenza tecnica per poter eseguire la verifica del materiale da revisionare.

Tutti i partecipanti sono considerati alla pari senza gerarchia di ruoli (da cui la denominazione *Peer Review*). L'attività è svolta tra colleghi dove ognuno di loro agisce nell'ottica di aiutarsi a vicenda a verificare la correttezza del materiale prodotto. Ogni collega, a turno è revisore del lavoro dei altri ed è anche revisionato per quanto da lui prodotto. In ottica di aiuto reciproco, è sconsigliata la presenza dei capi.

Le modalità di organizzazione e 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). Ciò è richiesto in quanto l'attività ha un suo costo che incide sul progetto.
- *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.

---

<sup>8</sup> Il termine “Peer Review” è adoperato nel modello CMM per indicare le revisioni tecniche del software fatte da personale tecnico di pari livello (escludendo cioè il management).

- *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 essi siano tradotti in codice nelle fasi successive.
- *Riunione di revisione:* l'autore, il moderatore ed i revisori si riuniscono nel giorno concordato per esporre e commentare le osservazioni fatte sui documenti. E' importante che si segnalino solo gli errori riscontrati senza entrare in discussioni circa la loro risoluzione. Sarà l'autore a decidere se e come risolvere gli eventuali problemi segnalati. 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 correttamente (si aiuti l'autore a scoprire gli errori senza giudicare la sua capacità tecnica di fare 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 (eventualmente 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.

## Profilo di difettosità 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*: 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 e progettazione	Codifica e Test unitario	Test d'integrazione	Test di sistema
<b>Valori attesi</b>	2.2	3.8	13.3	5.1	1.2
<b>Valori effettivi</b>	<b>1.2</b>	<b>2.7</b>	<b>21.8</b>	<b>8.7</b>	<b>4.7</b>

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 rimuovere in ogni fase a seguito delle attività pianificate (revisioni tecniche e test). Tali valori sono dedotti 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.

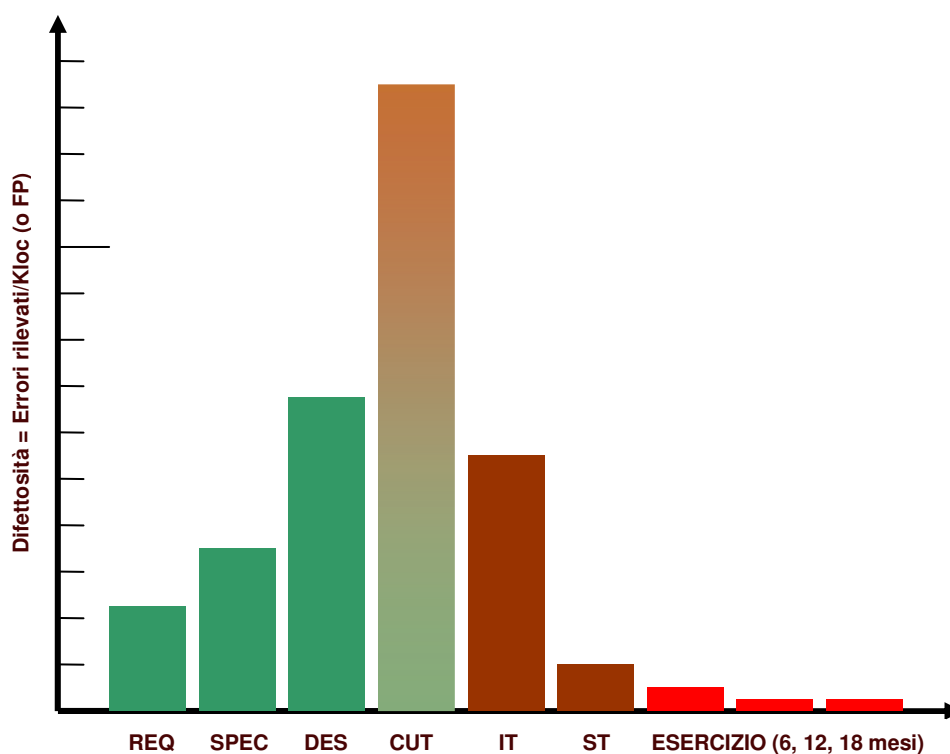


Figura 4. Profilo della difettosità del prodotto software.

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

**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.

## Classificazione ortogonale dei difetti

La tabella in oggetto (*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 di sviluppo		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, 3 di essi sono da imputare alla fase di analisi mentre i restanti 6 sono da imputare alla progettazione. Per finire, dall'analisi dei 48 errori rilevati durante l'esecuzione dei test, 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<sup>9</sup>, per esempio, aiuta a definire le aree su cui intervenire per migliorare la qualità del software e le performance dell'intera organizzazione di sviluppo.

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

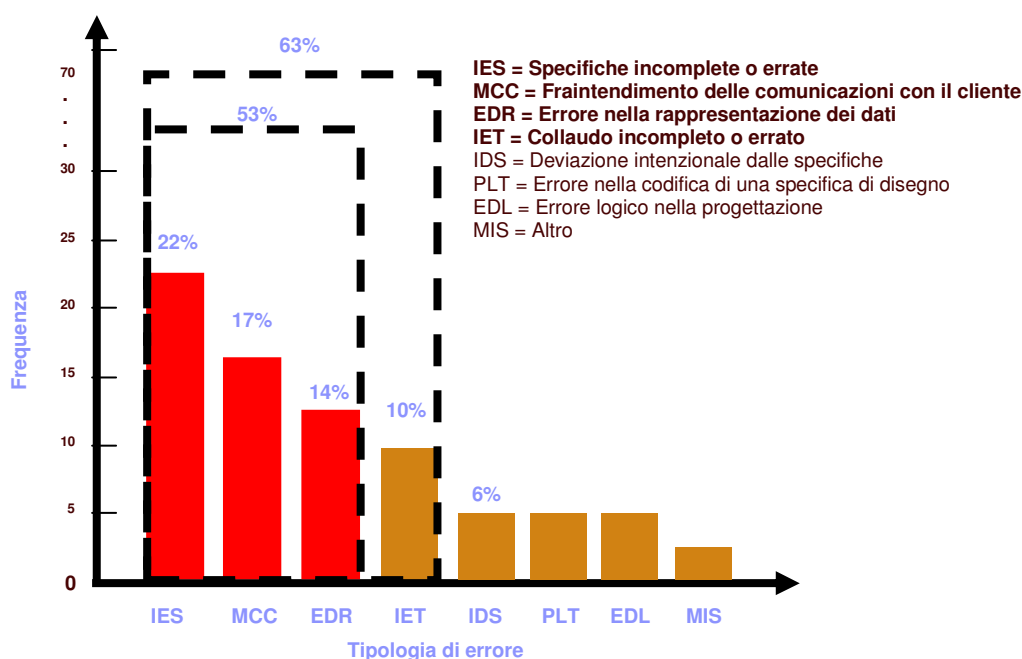


Figura 5. Classificazione dei difetti e rappresentazione secondo Pareto.

### Curva di rimozione degli errori nei test (curva di saturazione)

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.

<sup>9</sup> 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).

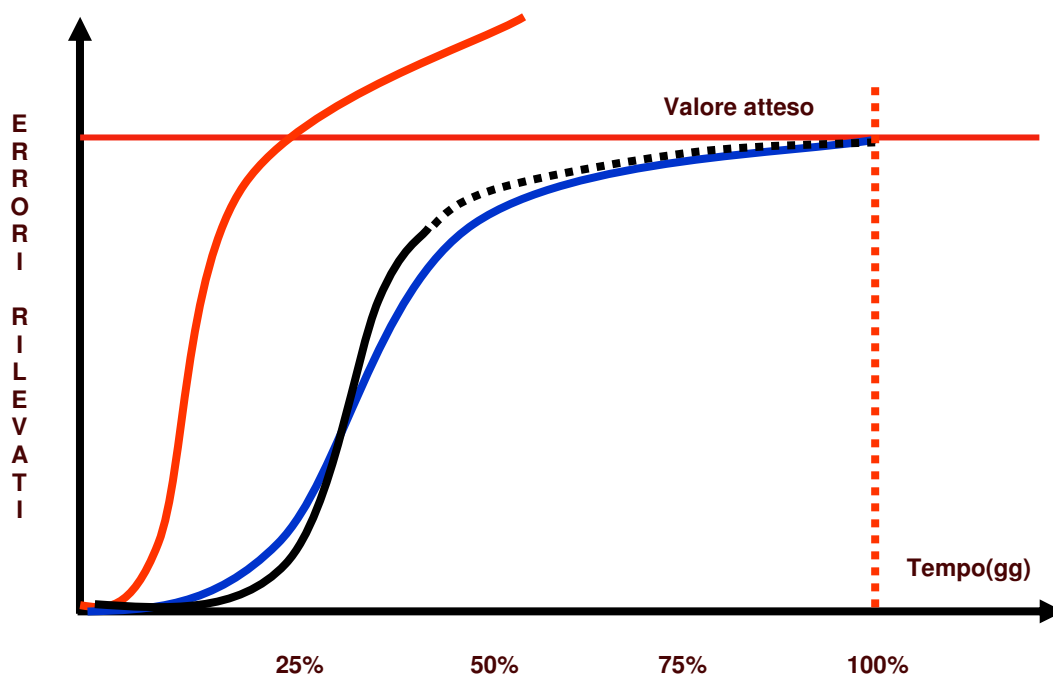


Figura 6. Singola fase di test (es. System)

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).

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.

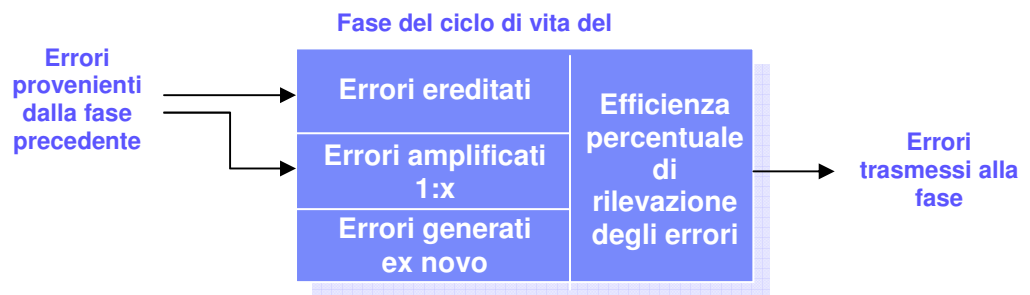
L'appiattimento della curva indicherà il momento in cui si è esaurita la capacità dei casi di prova di rilevare altri errori (nell'esempio dopo 21 giorni si può considerare completata la fase di test in oggetto). L'appiattimento della curva prima del valore "atteso" è dovuto ad una di queste possibili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

### 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.



Source: IBM System Scientific Institute,

**Figura 7. 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%.

## Archivio storico dei progetti

L'Archivio storico dei progetti è uno strumento prezioso per l'azienda. Esso contiene la memoria storica dei progetti aziendali, è uno strumento fondamentale per effettuare analisi statistiche e permette il controllo quantitativo dei progetti.



In esso sono memorizzate le informazioni dei progetti e relative a:

- Caratteristiche dei progetti; Produttività dei gruppi di lavoro; Qualità del software realizzato.

Le informazioni sono memorizzate secondo:

- un tracciato definito; le metriche identificate.

L'archivio costituisce quindi la memoria storica dell'azienda ed è una rappresentazione fedele dei dati relativi ai progetti.

E' uno strumento indispensabile per:

- Prendere decisioni con cognizione di causa (es. su nuovi progetti, acquisiti, ecc.); Controllare efficacemente le risorse (interne ed esterne).

Consente, tra le altre cose, di:

- Stimare e pianificare con maggiore cura le fasi del progetto; Effettuare l'analisi approfondita della produttività dei progetti e della qualità dei prodotti; Costruire modelli di profilo statistico che fanno da guida alle attività di controllo; Prevedere la difettosità in esercizio e pianificarne l'impegno di risorse e l'impatto economico; Confrontare le tipologie di problemi emersi in esercizio con le mappe di errori rilevate in fase di verifica e validazione, migliorare il processo di rimozione degli errori e ridurre il numero, la gravità e l'impatto sugli utenti; Valutare l'efficacia del processo di verifica e validazione (revisioni tecniche e test); Creare una mappa statistica dei problemi più frequenti immessi nelle fasi di analisi e disegno del software; Pianificare il miglioramento dei processi con beneficio sulla produttività delle risorse e la qualità del prodotto;

### **Reporting:**

Alcuni contenuti dei report statistici sono:

- *Profilo di difettosità del software:* Profilo della rimozione degli errori nelle diverse fasi di un progetto e globale; Difettosità residua del software (numero di errori rimasti dopo il suo rilascio in esercizio); Profilo di tipologia di errori rimossi;
- *Profilo di efficacia delle fasi di test:* Profilo di rimozione degli errori in una fase di test; Profilo di saturazione degli errori in una fase di test;
- *Profilo della produttività di un progetto:* Globale (su tutto il progetto); per Fase (Analisi, disegno, codifica, test, collaudo); per Attività (Ispezione);
- *Profilo delle stime nell'arco del ciclo di sviluppo:* Dimensione del prodotto, produttività, difettosità residua, ecc.

### **Esempio di struttura dei dati:**

I dati relativi ad un **prodotto** possono essere:

- *Identificativo:* nome, rilascio, data di rilascio;
- *Dimensioni:* KLoc o FP;

- *Tipologia*: software applicativo, prodotto programma, ecc.
- *Caratteristiche tecniche*: tecnologia, linguaggi, ecc.
- *Qualità*: profilo qualitativo (difettosità media del ciclo di sviluppo, delle singole fasi di sviluppo, della fase di manutenzione, ecc.) ecc.
- *Caratteristiche prestazionali*: performance, robustezza, ecc.
- *Documentazione a supporto*: Documentazione tecnica (completa/parziale, numero di documenti e di pagine); Documentazione utente (completa / parziale, numero di manuali e di pagine);
- **Note**: ogni informazione utile per la comprensione del prodotto.

I dati relativi al **progetto** possono invece essere:

- *Durata*: numero di mesi (pianificati, effettivi, ritardo);
- *Staff*: numero di persone impiegate per fase e per ruolo (pianificate ed effettive);
- *Stime*: dimensione del software (stima iniziale, a fine progettazione, a fine progetto);
- *Produttività*: per ciascuna fase e globale per l'intero progetto, in KLoc/MP o FP/MP, (stima iniziale e a fine progetto);
- *Ciclo di vita del software adoperato*: standard/ridotto (secondo lo standard aziendale);
- *Difettosità rilevata (normalizzata)*: per ciascuna fase e globale per l'intero progetto;
- *Ispezioni*: numero di ispezioni eseguite, numero di errori rilevati, numero di ore spese, numero di documenti ispezionati (e numero di pagine revisionate)
- *Test*: numero di test eseguiti (ipologie), numero di casi di prova eseguiti, numero di errori rilevati, numero di persone impiegate, numero di giorni persona impiegati
- *Tracciabilità dei requisiti*: numero di requisiti indirizzati, sviluppati e testati, livello di copertura dei test rispetto ai requisiti
- **Note**: ogni informazione utile per la comprensione dei dati.

### Statistiche:

Basandosi sui dati contenuti nell'Archivio storico dei progetti, la funzione di Assicurazione Qualità, elabora i dati e produce statistiche periodiche di controllo sui progetti. Il controllo statistico è basato sui dati disponibili e la sua attendibilità dipende da:

- *numerosità* dei dati disponibili (numero di progetti memorizzati);
- *dettaglio* dei dati disponibili (numero di informazioni memorizzate);
- *qualità* dei dati disponibili (affidabilità delle informazioni registrate).

Ogni progetto memorizzato ha un effetto di raffinamento dei dati (ne convalida o corregge l'andamento).

Ogni progetto "nuovo" può attingere ai dati statistici per pianificare il proprio andamento. La possibilità di confrontarsi con modelli esistenti (progetti simili) permette di effettuare stime più attendibili e realistiche. E questa è sicuramente un "best practice".

Le statistiche hanno due finalità:

1. Finalità di **controllo**: Controllo della qualità dei prodotti; Controllo della produttività dei progetti; Controllo dell'operatività dei fornitori (eventuali); Controllo della capacità di stima; Controllo della bontà dell'investimento.
2. Finalità **statistica**: Costruzione di modelli statistici per il controllo dei processi; Costruzione di modelli statistici di controllo della qualità dei prodotti; Costruzione di modelli statistici per il controllo della gestione dei progetti (pianificazione e conduzione); Costruzione di modelli statistici per il controllo della bontà dell'investimento.

### Analisi causale

Lo scopo principale dell'analisi causale è quello di migliorare la qualità dei progetti e le performance dell'organizzazione tramite un'attività *preventiva*. Elaborando l'esperienza maturata nei progetti ed analizzando i dati contenuti nell'archivio storico dei progetti è possibile risalire alle cause principali dei problemi rilevati e agire di conseguenza con azioni preventive che migliorino gli aspetti di fondo dell'organizzazione e, da questi, dei progetti e dei prodotti realizzati.

L'analisi causale si effettua per ogni singolo "errore grave" (*major error*) riscontrato nel corso del progetto. L'analisi può rilevare che un problema possa essere stato generato da:

- un *errato utilizzo dei processi, metodi e tecniche esistenti*. In questo caso occorre intervenire sulle competenze delle persone;
- *l'adozione di un processo, metodo o tecnica che indice all'errore*. In questo caso si interverrà sui processi, metodi o tecniche.

L'attività di analisi causale è condotta al termine di ogni progetto o, se necessario, al termine di una o più fasi di sviluppo particolarmente critiche per il proseguo del progetto. Una sessione di analisi non dovrà durare più di 5 o 6 ore analizzando nel complesso non più di 5 o 6 major error (spendendo cioè circa un'ora per ogni problema).

Per garantire la massima efficacia di tale attività occorre che essa sia condotta in maniera formale per evitare dispersione, lungaggine e mancato raggiungimento dell'obiettivo finale. Le sessioni sono quindi:

- pianificate (da parte del capo progetto a conclusione del progetto stesso);
- condotte sotto la responsabilità di una persona (generalmente SQA) e con il coinvolgimento di poche persone esperte della materia (non più di due o tre persone);
- documentazione dei risultati (major error, causa principale certa, azione correttiva proposta, efficacia dell'azione proposta, impegno, durata e costo dell'azione, beneficio atteso, priorità, altre informazioni necessarie);
- revisione da parte della direzione e pianificazione delle azioni suggerite (selezione delle azioni da implementare, definizione delle priorità, assegnazione delle risorse necessarie all'implementazione delle azioni);
- valutazione dell'efficacia delle azioni (generalmente da parte del gruppo SQA);
- inclusione delle azioni nei processi formali dell'organizzazione e divulgazione dei processi stessi (generalmente gruppo SQA).

## Glossario

Termine	Descrizione
CI	Configuration Item
CMM	Capability Maturità Level
CUT	Code and Unit Test
EF	Emergency Fix
FP	Function Point
FVT	Functional Verification Test
IQP	Inventory Quality Plan
ISO	International Organization for Standardization
IBM	International Business Machines
IT	Integration Test
LOC	Line od Code
MAC	Manutenzione Correttiva
MEV	Manutenzione Evolutiva
PQP	Piano di Qualità del Progetto
Kloc	Kilo Loc
KPA	Key Process Area
KPI	Key Process Indicator
SCM	Software Configuration Management
SEI	Software Engineering Institute
SQA	Software Quality Assurance
SVT	System Verification Test
UAT	User Acceptance Test

## Riferimenti bibliografici

### Qualità

- [LEGA06] Riccardo Lega - 2006  
*La Patente Europea della Qualità (EQDL)*  
FrancoAngeli
- [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
- [BANCII93] 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

### 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*

[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 (CMM e CMMI)

[SEI-CMMI02] Capability Maturity Model ® Integration (CMMI<sup>SM</sup>), V1.1  
CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1), Staged Representation – Improving processes for better products – CMMI Product Team, March 2002

[SEI-SPM01] Software Project Management  
SEI Curriculum Module SEI-CM-21-1.0  
July 1989 (Draft For Public Review)

[CMM93] Software Engineering Institute, SEI - 1993  
*Capability Maturity Model for Software, Version 1.1*

### Sviluppo professionale

- [SWEBOK00] IEEE - 2004  
*Guide to the Software Engineering Body of Knowledge (SWEBOK)*  
A project of the IEEE Computer Society, Professional Practices Committee  
- IEEE
- [MCCONNELL03] Steve McConnell - 2003  
*Professional Software Development*  
Addison-Wesley

### Ingegneria del software

- [FUGGETTA06] Annalisa Binato, Alfonso Fuggetta, Laura Sfardini - 2006  
*Ingegneria del software*  
Pearson – Addison Wesley
- [JOEL05] Joel Spolsky - 2005  
*Joel e il Software*  
Apress
- [SOMMERV05] Ian Sommerville - 2005  
*Ingegneria del software*  
PEARSON Addison-Wesley
- [PRESSMAN05] Roger S. Pressman – 2005  
*Principi di Ingegneria del Software (titolo originale: Software Engineering, A Practitioner's Approach, Sixth Edition - 2004)*  
McGram-Hill
- [GHEZZI04] Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrili - 2004  
*Ingegneria del software*  
PEARSON Prentice Hall
- [KUCHTEN99] Philippe Kruchten - 1999  
*Unified Process Introduction*  
Assidon-Wesley
- [CICOJNI96] Vincenzo Ambriola, Giovanni Cicogni - 1996  
*Laboratorio di Programmazione*  
Jackson Libri
- [BROOKS95] Brooks Frederick P. - 1995  
*The Mythical Man-Month: Essays on Software Engineering – 20th Anniversary Edition*  
Addison-Wesley, Reading(MA)
- [MARTIN90] James Martin - 1990  
*Information Engineering Books: I (Introduction), II (Planning and Analysis), III (Design and Construction)*  
Prentice-Hall

[NATO68] *Atti della conferenza Nato tenuta nel 1968*

### **Analisi, disegno, coding e Object Technology**

- [FOWLER04] Martin Fowler – 2004  
*UML Distilled, Terza edizione*  
Addison-Wesley
- [SCOTT01] Kendall Scott – 2001  
*UML Explained, Prima edizione italiana*  
Addison-Wesley
- [TKACH96] Daniel Tkach, Richard Puttick - 1996  
*Object Technology in Application Development*  
Addison-Wesley
- [YOURDON96] Edward Yourdon – 1996  
*Rise & Resurrection of the American Programmer*  
Prentice Hall PTR
- [BOOCH94] Gardy Booch - 1994  
*Object-Oriented Analysis and Design with Applications, 2nd edition*  
Addison-Wesley
- [JACOBSON92] Ivar Jacobson - 1992  
*Object-Oriented Software Engineering – a Use Case-Driven Approach*  
Addison-Wesley
- [RUMBAUGH91] James Rumbaugh - 1991  
*Object-Oriented Modeling and Design*  
Prentice-Hall
- [LEDGARD75] Henry F. Ledgard - 1975  
*Programming Proverbs*  
Hayden Book

### **Project Management**

- [PMBOK00] Project Management Institute (PMI) – 2000  
*A Guide to the Project Management Body of Knowledge (PMBOK Guide)*  
PMI – Newtown Square, Pennsylvania USA
- [KLIEM04] Ralph L. Kliem – 2004  
*Leading High-Performance Projects*  
J. Ross Publishing
- [ARCHIBALD97] Russel D. Archibald – 1997  
*Project Management, La gestione di progetti e programmi complessi*  
Franco Angeli

- [MEREDITH95] Jack R. Meredith, Samuel J. Mantel Jr. – 1995  
*Project Management. A Managerial Approach, Third Edition*  
John Wiley & Sons

### Testing

- [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
- [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

### Metriche del software

- [GUFPI06] GUFPI – ISMA  
*Metriche del software – Esperienze e ricerche*  
FrancoAngeli
- [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>

## Sviluppo di Software Applicativo

---

- [CANTONI03] Lorenzo Cantoni, Nicoletta Di Blas, Davide Bolchini - 2003  
*Comunicazione, qualità, usabilità*  
Apogeo
- [CORSO03] Corso pratico – 2003  
*L'artista digitale (titolo originale: Design Companion for the Digital Artist)*  
Mondadori Informatica
- NORMAN00] Donald A. Norman - 2000  
*Il computer invisibile*  
Apogeo
- [VISCIOLO00] Michele Visciola - 2000  
*Usabilità dei siti Web*  
Apogeo
- [NIELSEN00] Jakob Nielsen - 2000  
*Web usability*  
Apogeo
- [SKLAR00] Joel Sklar - 2000  
*Principi di Web Design (titolo originale: Principles of Web Design)*  
Apogeo
- [LYNCH99] Patrick J. Lynch, Sarah Horton – 1999  
*Web, Guida di stile (titolo originale: Web Style Guide – Basic Design Principles 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)



*Sviluppare software è un'attività creativa e tecnica allo stesso tempo. Richiede competenza ed esperienza, attitudine e capacità, metodi e tecniche, processi e strumenti. E' un'attività ad alto contenuto intellettuale e quindi soggetta anche ad errori. Quelli immessi nelle prime fasi di sviluppo generano altri errori nelle fasi successive secondo la teoria della propagazione degli errori nel software. Occorre quindi rimuovere gli errori il prima possibile, man mano che si procede nello sviluppo. Gli errori presenti nel software rilasciato in esercizio causano problemi agli utenti finali e generano alti costi di rimozione. La qualità del software quindi è realizzata giorno per giorno durante l'intero ciclo di sviluppo. Misurare la qualità raggiunta al termine del progetto serve a poco se non si può più intervenire per migliorarla.*

*Il documento descrive le problematiche tipiche della qualità del software e fornisce alcuni elementi di riflessione sull'approccio tipico dei progetti. Propone l'esperienza pratica maturata dall'applicazione del metodo illustrato presso un gruppo di piccole e medie imprese software. I temi trattati spaziano lungo l'intero ciclo di sviluppo: dalla qualità dell'analisi dei requisiti alla qualità della progettazione, dalla qualità del codice all'efficacia dei test, all'assicurazione qualità. Si propongono le migliori pratiche (best practice) che l'ingegneria del software ha definito ed il mercato ha confermato come valide.*



Ercole Colonese svolge la sua attività di consulente essenzialmente nell'area dello sviluppo applicativo. La sua esperienza è maturata in moltissimi anni di lavoro presso i laboratori internazionali IBM di sviluppo prodotti software dove ha coperto ruoli tecnici e manageriali. Ha realizzato numerosi sistemi qualità aziendali certificati ISO9000 presso piccole, medie e grandi aziende, pubbliche e private. Ha implementato modelli di eccellenza (EFQM, Malcom Bauldrige ecc.). 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 come il *Capability Maturity Model* del *Software Engineering Institute* (SEI-CMM e CMMI). Come docente, ha tenuto corsi sulla qualità del software e le metodologie di sviluppo presso il Learning Center IBM. Ha pubblicato diversi articoli sulle tematiche dello sviluppo software ed il ruolo consulenziale. Attualmente è professore aggiunto di Ingegneria del software presso l'Università di Roma Tor Vergata.

e-mail: [e.colonese@virgilio.it](mailto:e.colonese@virgilio.it)

[www.colonese.it](http://www.colonese.it)