



# Collaudo del software



*Disambiguazione* – "Testing" rimanda qui. Se stai cercando l'album di ASAP Rocky del 2018, vedi Testing (album).

**Questa voce o sezione sull'argomento ingegneria del software non cita le fonti necessarie o quelle presenti sono insufficienti.**

Il **collaudo del software** (anche **software testing** in lingua inglese), in informatica, indica un procedimento, che fa parte del ciclo di vita del software, utilizzato per individuare le carenze di correttezza, completezza e affidabilità delle componenti software in corso di sviluppo. Consiste nell'esecuzione del software da parte del collaudatore, da solo o in combinazione ad altro software di servizio, e nel valutare se il comportamento del software rispetta i requisiti. A volte il collaudo, che fa parte delle procedure di assicurazione di qualità, viene confuso con il debugging, con il profiling, o con il benchmarking.

## Descrizione generale

In generale, occorre distinguere i "malfunzionamenti" del software (in inglese, "failure"), dai "difetti" (o "errori" o "buchi") del software (in inglese, "fault" o "defect" o "bug"). Un *malfunzionamento* è un comportamento del software difforme dai requisiti espliciti o impliciti. In pratica, si verifica quando, in assenza di malfunzionamenti della piattaforma (hardware + software di base), il sistema non fa quello che l'utente si aspetta.

Un **difetto** è una sequenza di istruzioni, sorgenti o eseguibili, che, quando eseguita con particolari dati in input, genera un malfunzionamento. In pratica, si ha un malfunzionamento solo quando viene eseguito il codice che contiene il difetto, e solo se i dati di input sono tali da evidenziare l'errore. Per esempio, se invece di scrivere "a >= 0", il programmatore ha erroneamente scritto "a > 0" in una istruzione, si può avere un malfunzionamento solo quando viene eseguita quell'istruzione mentre la variabile "a" vale zero. Lo scopo del collaudo è di rilevare i difetti tramite i malfunzionamenti, al fine di minimizzare la probabilità che il software distribuito abbia dei malfunzionamenti nella normale operatività.

Nessun collaudo può ridurre a zero tale probabilità, in quanto le possibili combinazioni di valori di input validi sono enormi, e non possono essere riprodotte in un tempo ragionevole; tuttavia un buon collaudo può rendere la probabilità di malfunzionamenti abbastanza bassa da essere accettabile dall'utente. L'accettabilità di una bassa probabilità di malfunzionamento dipende dal tipo di applicazione.

Il software per cui è richiesta la massima qualità, è quello cosiddetto "life-critical" ("safety critical"), cioè in cui un malfunzionamento può mettere a rischio la vita umana, come quello per apparecchiature medicali o aeronautiche. Per tale software è accettabile solo una probabilità di malfunzionamento molto bassa, e pertanto il collaudo è molto approfondito e rigoroso. Per rilevare il maggior numero possibile di difetti, nel collaudo si *sollecita* il software in modo che sia eseguita la maggior quantità possibile di codice con svariati dati di input.

Può anche darsi che un errore nel codice sorgente generi un malfunzionamento solo se si utilizza un particolare compilatore o interprete, o solo se eseguito su una particolare piattaforma. Pertanto può essere necessario collaudare il software con vari ambienti di sviluppo e con varie piattaforme di utilizzo.

Le tecniche di collaudo possono essere classificate in molti modi. I principali sono i seguenti:

- Per modello di sviluppo: a cascata, guidato dal collaudo.
- Per livello di conoscenza delle funzionalità interne: a scatola bianca, a scatola nera.
- Per appartenenza o meno dei collaudatori all'organizzazione che modifica i sorgenti, nonché per fase di sviluppo: Alfa, Beta.
- Per grado di automazione: manuale, semi-automatizzato, completamente automatizzato.
- Per granularità: collaudo di modulo, collaudo di sistema.
- Per grado di formalità: da informale a formale.

## Verifica e validazione

La fase di **verifica** e di **validazione** serve ad accertare che il software ottemperi agli obiettivi.

Precisamente:

- la **verifica** serve a stabilire che il software rispetti i requisiti e le specifiche, quindi ad esempio che non ci siano requisiti mancanti o che le diverse prove (esecuzione, moduli/parti del sistema, unità, integrazione, etc) abbiano esito positivo, La verifica può essere eseguita in diverse sotto fasi o riguardare specifici aspetti (categorie di funzioni, sicurezza, architettura, ingegnerizzazione del sistema, prestazioni); spesso si esegue la verifica X per poter procedere alla successiva sottofase X+1; esistono test di verifica della progettazione (analisi funzionale) e altre più propriamente di sviluppo;
- invece, la **validazione** serve ad accertare che i requisiti di utilizzo (bisogni e aspettative dell'utente) siano anche soddisfatti nella maniera giusta. Il collaudo per antonomasia è quello finale, eseguito in condizioni che simulino il reale impiego applicativo. La validazione (validare = convalidare, rendere valido) segue sempre la verifica.

Questa fase, infatti, è molto delicata in quanto, se il software passa la verifica, ma non la validazione, dopo tutto il processo si può ottenere un software perfettamente funzionante, senza errori, ma del tutto inutile in quanto non rispecchia quanto era stato chiesto all'inizio (in tal caso non adempiendo all'insieme completo delle funzionalità previste e quindi non servendo allo scopo di progetto, può esserci il rischio che venga rigettato dal cliente). Oltre alla funzioni (tra cui quelle di sicurezza, sempre più rilevanti e spesso date per scontate), la validazione deve accertare anche il raggiungimento dei livelli prestazionali (in pratica, la velocità di esecuzione e l'efficienza del sistema).

Secondo il modello applicato questa fase si applica su stadi intermedi o su tutto il sistema software (ivi compresa l'ingegnerizzazione della piattaforma SW/HW quando prevista cioè l'ambiente in cui il software verrà eseguito e fruito). La verifica può essere *statica*, se effettuata su carta, cioè sul progetto, o *dinamica*, se effettuata attraverso il collaudo dello stesso software con dati di test.

# Modello di sviluppo

---

## Sviluppo a cascata

Il procedimento tradizionale di sviluppo del software, detto "a cascata" (in inglese "*waterfall*"), prescrive di iniziare il collaudo appena è completata la prima versione del prodotto. I risultati del collaudo vengono utilizzati per rivedere i requisiti, o il progetto, o il codice, e produrre così la versione successiva.

Questo procedimento è stato sottoposto a una severa critica in quanto ha i seguenti svantaggi:

- Se è giunta la data prevista per il completamento del prodotto, si tende a consegnarlo anche se il collaudo non è stato completato o ha dato esito negativo, il che significa che probabilmente si sta consegnando un prodotto scadente.
- Probabilmente, tra i requisiti non è stata inclusa la "collaudabilità" (in inglese, "*testability*"), cioè il progetto e il codice non contengono accorgimenti che agevolino il collaudo. In tal caso, il collaudo risulterà molto più costoso e meno efficace.
- Più tempo passa tra l'introduzione di un errore in un sistema e la segnalazione di un malfunzionamento dovuto a tale errore, più risulta difficile e costoso rimediare.
- Dato che il software in tale modello è previsto che venga sottoposto a test solo dopo aver completato la fase di sviluppo, l'eventuale feedback raccolto in fase di prova non può essere usato per alimentare tempestivamente l'apprendimento del team di sviluppo in modo tale che la qualità del codice possa essere migliorata come nei metodi iterativi ed incrementali (come può avvenire ad es. nei metodi "agili"). Pertanto nel modello di sviluppo in cascata le "lezioni apprese" possono essere utilizzate solo in eventuali successivi progetti di sviluppo, il che spesso ne limita l'effettivo valore aggiunto, in quanto la distanza nel tempo dalla fase di raccolta non agevola la applicazione.

## Sviluppo guidato dal collaudo



Lo stesso argomento in dettaglio: ***Test Driven Development***.

Un procedimento più recente, detto "guidato dal collaudo" (in inglese, "*test driven development*"), proposto a partire dall'inizio degli anni 1990, consiste nel considerare il collaudo una parte integrante del prodotto:

- Quando si analizzano i requisiti del software da produrre, si analizzano i requisiti del collaudo.
- Quando si progetta l'architettura del software da produrre, si progetta l'architettura del collaudo.
- Quando si scrive il codice del software da produrre, si scrive il codice delle routine di collaudo automatizzato o si preparano i dati e si scrivono le istruzioni per il collaudatore manuale.
- Quando si costruiscono gli eseguibili compilando i sorgenti, se la compilazione va a buon fine, vengono automaticamente eseguite le routine di collaudo automatizzato. Le


prove inerentemente interattive vengono tuttavia relegate a una procedura parzialmente manuale.

- Quando si archivia o si recupera una versione dei sorgenti, si archiviano o si recuperano tutti i suddetti documenti relativi al collaudo. Anzi, tali documenti devono essere intesi come parti integranti dei sorgenti.

## Fasi di distribuzione

---

### Il collaudo di tipo Alpha

 Lo stesso argomento in dettaglio: [Versione alpha](#).

Appena un software è stato costruito, prima di distribuirlo fuori dall'azienda, viene normalmente sottoposto a un collaudo interno all'azienda stessa.


Il collaudo di tipo Alpha può essere eseguito dagli stessi programmatori o da altro personale dell'azienda.

Spesso, il software prodotto per il collaudo Alpha viene arricchito di istruzioni di controllo a run-time che facilitano il rilevamento degli errori. Esempi di tali istruzioni sono:

- il controllo che non si acceda a indici non validi di array;
- il controllo che tutta la memoria allocata venga disallocata prima della terminazione;
- asserzioni dichiarative scritte dal programmatore.

In alcuni casi, in particolare per lo sviluppo di software di sistema o di software embedded, si utilizza un ambiente hardware di esecuzione che supporta specificamente il debugging e il testing.

### Il collaudo di tipo Beta

 Lo stesso argomento in dettaglio: [Versione beta](#).

Spesso, quando un prodotto ha superato il collaudo di tipo Alpha, viene distribuito all'esterno dell'azienda ad alcuni clienti selezionati o anche a tutti i clienti, avvertendo gli utenti che il prodotto distribuito potrebbe non essere di qualità elevata e probabilmente potrebbe richiedere ulteriori correzioni.

Gli utenti della versione Beta possono simulare l'utilizzo del software in casi realistici, e inviare al produttore resoconti dei malfunzionamenti riscontrati. Tale tipo di collaudo eseguito gratuitamente dagli utenti viene detto "collaudo Beta" (in inglese "Beta testing").

Possono esserci più versioni Beta, man mano che vengono corretti gli errori. Quando la frequenza delle segnalazioni d'errore diventa bassa, è il momento di distribuire la versione ufficiale.

Anche dopo che sono state distribuite delle versioni Beta, e quindi si è in fase di collaudo Beta, all'interno dell'azienda può continuare il collaudo Alpha.

## Automazione del collaudo

---

 Lo stesso argomento in dettaglio: [Automazione del collaudo del software](#).


Se il collaudo consiste nell'utilizzo del prodotto quasi come se fosse la normale operatività, si parla di "collaudo manuale".

Se il collaudo consiste nello sviluppo di apposito software che interagisce con il software da collaudare e fornisce un rapporto di qualità in assenza di personale, si parla di "collaudo automatizzato".

## Granularità

---

### Il collaudo di modulo (Unit Testing)

 Lo stesso argomento in dettaglio: *Unit testing*.

Quando si costruisce un prodotto complesso come un'automobile, non ci si limita a costruire e ad assemblare i pezzi e alla fine girare la chiave per vedere se tutto funziona. Questo per tre motivi:

- Alcuni difetti producono malfunzionamenti solo dopo un utilizzo prolungato, ma sono facilmente identificabili esaminando i singoli pezzi prima dell'assemblaggio.
- Se dopo aver girato la chiave la macchina non si accende, risulta molto difficile capire qual è il difetto.
- Se dopo aver girato la chiave la macchina non si accende, risulta molto costoso smontare la macchina, sostituire il pezzo difettoso e rimontarla.
- In alcuni casi potrebbero derivare dei rischi di arrecare danni a persone o cose dal funzionamento difettoso di un componente critico.

Ragionamenti analoghi valgono per il software. Pertanto, come i singoli pezzi di una macchina vengono sottoposti al controllo di qualità prima dell'assemblaggio, così è opportuno collaudare separatamente le singole componenti di un sistema software.

Le componenti collaudabili di un sistema software prendono il nome di "moduli" o "unità", per cui si parla di "collaudo di modulo" (in inglese, si usa l'espressione "unit testing").

Un modulo può avere una granularità variabile da una singola routine, a un insieme di alcune decine di routine, a un sottosistema comprendente migliaia di routine. Nella programmazione orientata agli oggetti la tipica componente da collaudare è la classe.

Siccome un modulo, per definizione, non è un programma completo, mentre il concetto stesso di collaudo richiede l'esecuzione di un programma, per eseguire il collaudo di modulo si deve costruire un apposito programma che contiene il modulo da collaudare.

In un programma completo, tutti i moduli, eccetto quelli di livello più basso, richiamano routine di altri moduli, e tutti i moduli, eccetto quelli di livello più alto, contengono routine a disposizione di altri moduli.

Per costruire un programma intorno a un modulo, è necessario fornire delle routine banali che possano essere chiamate dal modulo, finché non saranno pronte le routine complete. Tali routine banali sono dette stub (che in inglese significa "mozzicone", "moncherino").

Per mandare in esecuzione il codice contenuto nel modulo occorre invocare le routine. Viene pertanto

scritto un apposito programma che richiama le routine del modulo, passandole in sequenza vari parametri e verificando che le routine rendano i valori attesi. Tale programma viene detto "driver" (che in inglese significa "guidatore", "conduttore").

Normalmente, per rendere più flessibile e modificabile il collaudo, il driver preleva da un file di dati i parametri che dovrà passare alle routine del modulo da collaudare, e confronta i risultati ottenuti con il contenuto di un altro file di dati. Per ogni discrepanza riscontrata tra i dati ottenuti e quelli attesi, viene generato un messaggio d'errore in un terzo file di resoconto.

Con tale architettura, se si vuole aggiungere una combinazione di valori a quelle esistenti, basta modificare i file di dati.

## Il collaudo di sistema

Anche se i singoli moduli sono corretti, il sistema ottenuto integrandoli potrebbe non esserlo. Pertanto è sempre necessario collaudare il sistema completo.

Per collaudare un sistema completo si deve utilizzare il software in modo il più possibile simile a come verrà utilizzato nella normale operatività, ma con le seguenti differenze:

- Per il software interattivo, la normale operatività consiste in una persona che interagisce con il software. Questo si ottiene con il collaudo manuale, che è sempre utile, ma ha vari svantaggi. Il collaudo automatizzato di un programma interattivo è difficile da ottenere, se non rinunciando a parte del concetto di *normale operatività*.
- Nella normale operatività di un sistema software, per ogni singola installazione del sistema, l'utilizzo segue un certo andamento ripetitivo. Siccome nel collaudo si deve simulare il maggior numero possibile di situazioni, normalmente si rinuncia alla ripetitività tipica delle situazioni reali, e invece si simulano utilizzi molto variegati.

## Conoscenza delle funzionalità interne

---

### Il collaudo a scatola bianca

Il collaudo a scatola bianca, noto anche con le denominazioni inglesi *white box testing* o *clear/glass box testing* ("collaudo a scatola trasparente") o *structural testing* ("collaudo strutturale") è una forma di collaudo che verifica il funzionamento *interno* di un componente software, piuttosto che le sue funzionalità. Poiché richiede la conoscenza e la comprensione della struttura interna del software sotto test (eventualmente anche del suo codice sorgente), questa forma di testing è in genere a carico di un programmatore, spesso lo stesso che ha scritto il codice. Benché il concetto si possa applicare a diversi livelli, il collaudo a scatola bianca è tipicamente unitario. Il collaudo a scatola bianca può facilitare il compito di garantire una copertura esaustiva dei test case rispetto al codice sorgente; la sua controindicazione principale è che esso costituisce una deroga al principio dell'incapsulamento: in particolare, un test case esistente può fallire a seguito di una modifica strutturale di un componente (per esempio in seguito a refactoring) anche quando questa modifica preservi correttamente la funzionalità del componente stesso.

### Il collaudo a scatola nera

Il collaudo effettuato accedendo al software solamente tramite l'interfaccia utente, oppure tramite interfacce di comunicazione tra processi, viene detto "collaudo a scatola nera" (in inglese, "black box testing").

Può essere manuale oppure automatizzato, e tipicamente è utilizzato per il collaudo di sistema, in quanto per collaudare l'intero sistema normalmente non è necessario richiamare singole routine.

Se si tratta di un collaudo automatizzato, le routine di collaudo sono prevalentemente scritte in un linguaggio di livello più alto del linguaggio con cui è stata scritta l'applicazione, a volte in un linguaggio progettato appositamente per il collaudo.

Il software da collaudare e il software di collaudo costituiscono processi distinti comunicanti.

Per il collaudo a scatola nera manuale non è richiesta l'opera di un programmatore, e per quello automatizzato è sufficiente un programmatore con modeste competenze.

Spesso il collaudo a scatola nera è effettuato da persone che non fanno parte dell'organizzazione che sviluppa il software; si tratta degli utilizzatori stessi che effettuano il collaudo Beta.

Un esempio di una tecnica di collaudo a scatola nera automatizzata consiste nel registrare l'interazione dell'utente in un file, e poi ripetere la registrazione simulando il comportamento dell'utente.

Un vantaggio del collaudo a scatola nera sta, nei casi in cui il codice sorgente e la documentazione di progetto sono segreti industriali, nel fatto che tale collaudo può essere effettuato anche da persone esterne all'azienda.

Un altro vantaggio sta nel fatto che per tale tipo di collaudo non sono necessari programmatori esperti e che conoscano gli aspetti interni del software da collaudare. Pertanto, si possono trovare molti più collaudatori, senza dover investire nell'addestramento.

## **Tecniche di collaudo a scatola nera**

Ci sono tre tecniche principali:

- Tecnica della copertura delle classi di equivalenza
- Tecnica di analisi dei valori estremi
- Tecnica di copertura delle funzionalità

## **Collaudo formale e informale**

---

### **Il collaudo informale**

Nelle piccole organizzazioni, o per prodotti software di poco valore, il collaudo è normalmente informale, cioè non esiste a livello organizzativo una mansione riconosciuta come "collaudo del software".

Il programmatore, appena dopo aver apportato una modifica al software, manda in esecuzione il software modificato e verifica interattivamente se il funzionamento è quello atteso. Se il comportamento è insoddisfacente, apporta altre modifiche e reitera il procedimento.

Quando il programmatore è soddisfatto del comportamento del software, invia il software al suo superiore, che effettua un ulteriore rapido collaudo manuale. A questo punto, se non si tratta di modifiche che devono urgentemente essere rese operative, la nuova versione viene inviata agli utenti e al personale di assistenza ("help desk") come versione Beta. Gli utenti e il personale vengono addestrati alle nuove modifiche, e tale addestramento costituisce l'occasione per la rilevazione di ulteriori malfunzionamenti.

In tale procedimento informale di collaudo, la segnalazione di malfunzionamenti e di nuove versioni non segue un iter ben definito. Si usano comunicazioni di persona, telefoniche, e-mail, e appunti su biglietti.

## **Il collaudo formale**

Il collaudo del software importante in grandi organizzazioni segue invece iter procedurali più rigorosi, analoghi a quelli dei progetti ingegneristici.

Per eseguire un collaudo formale, cioè rigorosamente pianificato, si scrive un "piano di collaudo" (in inglese, "test plan"), in cui si descrive dettagliatamente come deve essere svolto il collaudo.

Ci sono due strategie fondamentali per organizzare il collaudo: la "batteria di prove" (in inglese "test suite"), e lo "scenario di collaudo" (in inglese "test scenario"). Spesso si utilizzano in combinazione, cioè si pianificano una o più batterie di prove e una serie di scenari di collaudo.

Una batteria di prove è un insieme di collaudi elementari, ognuno dei quali è detto "prova" (in inglese "test case"). Ogni prova ha almeno i seguenti attributi:

- Spesso ha un identificatore o numero progressivo.
- Talvolta ha una descrizione dello scopo della prova.
- Una sequenza di operazioni necessarie per portare il sistema nelle condizioni iniziali per la prova.
- Una sequenza di operazioni necessarie per riportare il sistema nelle condizioni di base dopo la prova.
- Le dipendenze, cioè gli identificatori dei casi di prova che devono aver avuto successo affinché abbia senso effettuare questa prova. Per esempio, se una prova consiste nell'aprire un file e chiuderlo, e un'altra prova consiste nel leggere lo stesso file, la seconda prova dipende dalla prima, in quanto non ha senso tentare di leggere un file che non è stato possibile aprire.
- Le operazioni che sollecitano il software da collaudare.
- Il risultato atteso.
- Talvolta, il tempo massimo ammissibile per ricevere il risultato.

Spesso vengono aggiunti altri attributi.

Nel caso di collaudo manuale, queste informazioni possono essere stampate e tenute come linea guida per il collaudatore. Nel caso di collaudo automatizzato, queste informazioni sono le specifiche per il programmatore che deve realizzare il software di collaudo.

Quando una prova viene eseguita, si registrano altre informazioni, tra cui le seguenti:

- Autore della prova.



- Data e ora di esecuzione.
- Identificatore della versione collaudata.
- Esito (successo o fallimento).
- In caso di fallimento, breve descrizione del tipo di fallimento.

Uno scenario di collaudo è un utilizzo realistico non banale del software da collaudare. Mentre le prove di collaudo considerano le funzionalità elementari, ogni scenario prende in considerazione una tipologia di utente e una situazione verosimile e complessa in cui tale utente può trovarsi. Il collaudo di scenario percorre tutti i passi che l'utente percorrerebbe in tale situazione. Il collaudo Beta è di fatto un collaudo di scenario, sebbene informale. Il collaudo di scenario è necessariamente un collaudo di sistema, e tipicamente è manuale o semiautomatico.

## Altri tipi di collaudo

---

### Collaudo Prestazionale (Performance Test)

Tra i requisiti di qualità di un'applicazione, non c'è solo la correttezza, ma anche l'efficienza, o comunque vengono definiti requisiti prestazionali il cui soddisfacimento deve essere verificato prima di utilizzare il software prodotto. Le attività finalizzate a determinare se un dato prodotto software soddisfa specifici requisiti prestazionali è chiamata "collaudo prestazionale" (o "performance testing"). Il suo scopo non è quindi rilevare errori nell'applicazione, ma verificare che l'applicazione soddisfi i requisiti prestazionali richiesti in fase di esplicitazione dei requisiti.

Solitamente, si definiscono per vari tipi di operazioni dei tempi massimi di esecuzione (ovvero, si definiscono delle "baseline") e si verifica che il prodotto software non superi tali tempi limite. Con l'evolvere del software e dell'hardware, anche le baseline possono venire modificate. Per esempio, se il software viene sviluppato adottando una libreria meno efficiente, si ammette che le varie operazioni possano essere un po' più lente; d'altra parte, se il collaudo viene fatto su un sistema multiprocessore, si richiede che, a parità di velocità del processore, le varie operazioni debbano essere più veloci. Le baseline possono essere semplicemente ottenute misurando i tempi di esecuzione di un sistema esistente. Da un punto di vista di test, queste attività sono tutte del tipo white-box, dove il sistema è ispezionato e controllato "dall'interno verso l'esterno" e da vari angoli. Una volta raccolte le misure e analizzate, come risultato, si effettua un tuning applicativo.

Tuttavia, a volte si usa anche un approccio black-box effettuando un test di carico sul sistema. Per una applicazione web, ad esempio, si usano tool che simulano un certo numero di utenti/conessioni http concorrenti e si misura il "response time".

Il collaudo prestazionale può essere integrato nel collaudo di regressione per verificare che le modifiche non abbiano introdotto rallentamenti.

### Collaudo di Carico/Volume (Load/Volume Test)

Questo tipo di test in genere è parte del performance testing e tuning. In tale contesto, significa aumentare costantemente il carico sul sistema tramite strumenti automatici. Per una applicazione web, ad esempio, il carico è il numero di utenti/conessioni HTTP concorrenti.

In letteratura, il termine load testing è di solito definito come il processo di esercizio del sistema sotto test alimentandolo in modo da fargli eseguire i task più grandi con cui esso può operare. Il test di carico è talvolta chiamato anche volume testing, o longevity/endurance testing.

Esempi di volume testing:

- test di un word processor editando un documento molto grande;
- test di una stampante inviandogli un job molto grande;
- test di un mail server con migliaia di mailbox utente;
- un caso specifico di volume testing è il cosiddetto “zero-volume testing”, dove il sistema viene alimentato con task “vuoti”.

Esempi di longevity/endurance testing:

- test di una applicazione client-server eseguendo il client in un loop di accessi alla componente per un periodo di tempo esteso.

Scopi del load testing:

- scoprire bug non emersi in fase di test, quali errori del tipo “memory management”, “memory leaks”, “buffer overflow”, ecc.;
- assicurare che l'applicazione soddisfa le prestazioni di “baseline” stabilite durante il “performance testing”. Questo viene fatto effettuando un test di regressione sull'applicazione con uno specifico carico massimo.

Nonostante il “performance testing” e il “load testing” possano sembrare simili, il loro scopo è differente. Da un lato, il “performance testing” utilizza tecniche e strumenti di “load testing” per la misurazione e allo scopo di effettuare misure di “benchmarking” utilizzando vari livelli di carico. Dall'altro, il “load testing” opera ad un predefinito livello di carico, di solito il massimo carico che il sistema può accettare continuando a funzionare regolarmente. Lo scopo non è di “rompere” il sistema sovraccaricandolo, ma invece provare a mantenere il sistema costantemente attivo come una “macchina ben oliata”. È necessario enfatizzare l'importanza di questo tipo di test. L'esperienza ha infatti mostrato che molti bug importanti non emergono fintanto che il sistema non ha a che fare con una mole di dati veramente vasta, come ad esempio migliaia di utenti in repository quali LDAP/NIS/Active Directory, migliaia di caselle di posta utente su server-mail, tabelle multi-gigabyte nei database, lunghissime gerarchie di file/directory sui file-system, ecc. In questi casi, quasi sempre c'è l'esigenza di utilizzare tool automatizzati che generino una tale mole di dati, ma fortunatamente con un qualunque buon linguaggio di scripting questo risulta un compito molto facile.

## **Collaudo di Stress (Stress Test)**

Lo “stress test” ha lo scopo di provare a “rompere” il sistema sotto test sovraccaricando le sue risorse o sottraendogli risorse (in tale caso viene anche chiamato “negative testing”). Lo scopo principale di queste attività è verificare che il sistema va in “fault” e successivamente (eventualmente) recupera in maniera “indolore” – questo aspetto qualitativo è noto come recoverability (sistemi fault-tolerant).

Mentre il “performance test” richiede un ambiente controllato e misure ripetibili, lo stress test provoca caos e imprevedibilità. Per rifare ancora un esempio per una applicazione web, alcuni modi in cui si può stressare il sistema:

- raddoppiare il numero di utenti/conessioni HTTP previste in baseline
- in maniera casuale spegnere e riavviare porte dei switch/router di rete che collegano i server (via comandi SNMP ad esempio)
- mettere offline il database, e farlo ripartire
- effettuare un rebuild di un array RAID mentre il sistema è funzionante
- eseguire processi che consumano risorse (CPU, memoria, disco, rete) sui web-server sui database-server.

La lista può essere ovviamente arricchita. Comunque, lo stress test non è fatto al puro scopo di far andare in crash il sistema, ma piuttosto deve servire per osservare come il sistema reagisce alle failure. Riesce a salvare il suo stato o va in crash nuovamente/continuamente? Si ferma improvvisamente, bloccandosi, o in maniera controllata? Al riavvio, è in grado di recuperare dall'ultimo stato corretto? Visualizza messaggi di errore comprensibili all'utente, o si limita a visualizzare incomprensibili elenchi di codice esadecimale? La sicurezza del sistema è compromessa a causa di un failure inatteso? E così via.

## Il collaudo di retrocompatibilità

---

Uno scopo del collaudo è verificare che i nuovi prodotti e le nuove funzionalità aggiunte a vecchi prodotti siano di alta qualità. Tuttavia, nel software capita spesso che l'introduzione di una nuova funzionalità a un vecchio prodotto comprometta la qualità di funzionalità preesistenti del prodotto stesso. Pertanto, per assicurarsi la qualità del prodotto bisognerebbe ripetere l'intero collaudo ad ogni modifica. Il collaudo di funzionalità preesistenti e già collaudate, eseguito per assicurare che modifiche al prodotto non ne abbiano compromesso la qualità, si chiama "collaudo di regressione" (in inglese, "regression testing", o anche "non-regression testing"<sup>[1]</sup>), in quanto si vuole verificare se la qualità sia *regredita*.<sup>[2]</sup>

Se sono stati predisposti dei collaudi automatizzati, tramite appositi strumenti software e script dedicati, il collaudo di regressione ha normalmente un basso costo, in quanto si tratta solo di lanciare le procedure di collaudo esistenti, eventualmente ritoccate e confrontare i risultati prodotti con quelli corretti. Un completo collaudo di regressione manuale sarebbe invece enormemente costoso, soprattutto se il sistema deve essere collaudato con molti utenti simultanei.

Per tale motivo normalmente il collaudo di regressione manuale è più sbrigativo del collaudo originale e anche molto più pericoloso.

## Metriche

---

Ci sono metriche che vengono sviluppate per misurare l'efficacia del collaudo.<sup>[2]</sup> La principale è l'analisi della copertura del codice ottenuta tramite un profiler. Il profiler indica quante volte è stata eseguita ogni istruzione durante il collaudo. Le istruzioni eseguite almeno una volta sono dette "coperte". L'obiettivo è coprire il maggior numero possibile di istruzioni.

Un'altra metrica utile è il rapporto tra il numero di difetti trovati in un modulo e il numero di istruzioni modificate. Se si trova che in un modulo sono state modificate molte istruzioni ma sono stati trovati pochi difetti, si può dubitare dell'efficacia del collaudo di tale modulo.

# Certificazioni

---

Il software testing sta assumendo sempre di più il ruolo centrale nell'ambito dell'ingegneria del software. È quindi sempre più importante riuscire a qualificare, in maniera oggettiva, professionisti nell'ambito del testing. Sono quindi nati a livello mondiale percorsi di certificazione delle conoscenze. L'International Software Testing Qualifications Board (ISTQB) è l'ente più importante a livello mondiale che si occupa di queste certificazioni.

I software critici sviluppati per gli aeromobili devono rispettare le norme DO-178, standard *de facto* del mondo dell'aviazione.

## Note

---

1. <sup>^</sup> Mauro Pezzè e Michal Young, *Software testing and analysis: process, principles, and techniques*, Wiley, 2008.
2. Anirban Basu, *Software Quality Assurance, Testing and Metrics*, PHI Learning, 2015, ISBN 978-81-203-5068-7.



## Voci correlate

---

- Ciclo di vita del software
- Modello a cascata
- Modello a spirale
- Rollout
- Software
- HP LoadRunner
- Test metamorfico

## Altri progetti

---

-  Wikiversità contiene risorse sul **collaudo del software**
-  Wikimedia Commons (<https://commons.wikimedia.org/wiki/?uselang=it>) contiene immagini o altri file sul **collaudo del software** ([https://commons.wikimedia.org/wiki/Category:Software\\_testing?uselang=it](https://commons.wikimedia.org/wiki/Category:Software_testing?uselang=it))

## Collegamenti esterni

---

- 
- **(EN)**  *testing*, su *Enciclopedia Britannica*, Encyclopædia Britannica, Inc.
- *The Test Management Guide - A to Z and FAQ Knowledgebase*, su *ruleworks.co.uk*. URL consultato il 5 novembre 2006 (archiviato dall'url originale il 6 dicembre 2006).

- *BS 7925-1 Vocabulary of terms in software testing - ver. 6.3*, su [testingstandards.co.uk](http://testingstandards.co.uk).
- *Global directory of software testing companies*, su [testcompanies.com](http://testcompanies.com).
- *Functional Testing Types Explained With Examples*, su [simform.com](http://simform.com).



**Portale Informatica:** accedi alle voci di Wikipedia che trattano di informatica

---

Estratto da "[https://it.wikipedia.org/w/index.php?title=Collaudo\\_del\\_software&oldid=144233667](https://it.wikipedia.org/w/index.php?title=Collaudo_del_software&oldid=144233667)"