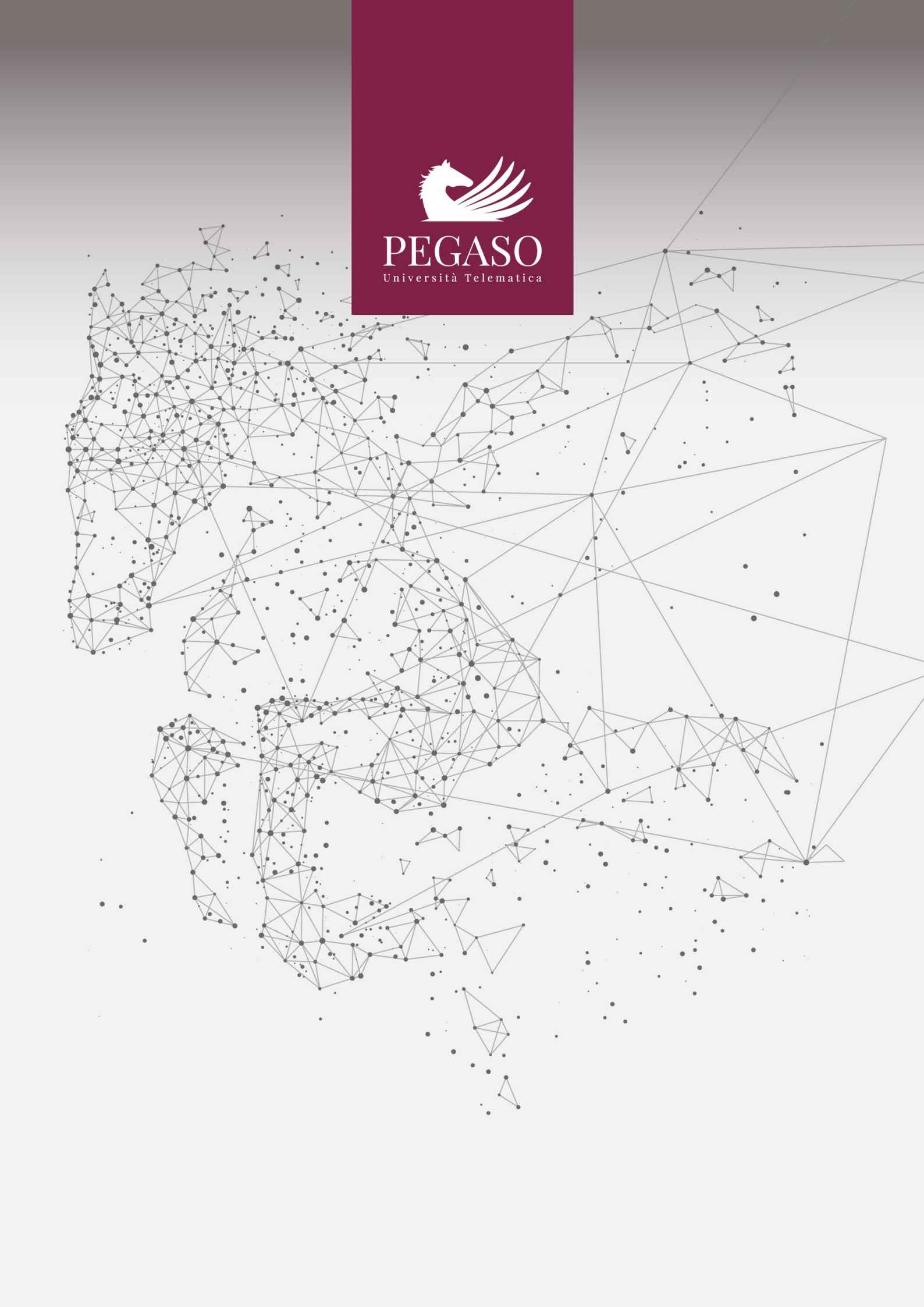
PEGASO
Università Telematica

# Indice

# 1. Premessa

Nel contesto dell'ingegneria del software, **il testing rappresenta una fase cruciale** del ciclo di vita di un sistema, con l'obiettivo primario di garantire che il prodotto finale soddisfi non solo i requisiti funzionali, ma anche le aspettative qualitative degli utenti e degli stakeholder. Non si tratta semplicemente di un'attività di "caccia al bug", bensì di un processo sistematico, metodico e iterativo che si estende trasversalmente lungo tutte le fasi dello sviluppo software. Il testing, se ben integrato nel processo di sviluppo, consente di prevenire difetti, migliorarne l'affidabilità e ridurre sensibilmente i costi di manutenzione post-rilascio. Inoltre, un sistema che non supera un'adeguata fase di testing può presentare anomalie difficili da individuare una volta in produzione, con conseguenze potenzialmente disastrose sul piano economico e reputazionale.

**L'obiettivo ultimo è fornire una panoramica approfondita delle attività di testing**, esplorando sia gli aspetti teorici sia le tecniche operative. Verranno analizzati i principali livelli di testing, dalle ispezioni del codice (code inspection), che rappresentano la forma più precoce di verifica, fino ai test di accettazione, passando per le verifiche di unità, di integrazione e di sistema. Saranno inoltre esaminati i test non funzionali, fondamentali per garantire la qualità globale del prodotto, con particolare riferimento alla performance, all'usabilità, alla sicurezza e all'affidabilità del sistema. La trattazione seguirà una progressione logica e coerente, rispecchiando l'ordine dei paragrafi presentati nelle slide del corso e supportata da esempi concreti e casi studio. Questo approccio intende offrire non solo una comprensione teorica ma anche una guida operativa per l'applicazione pratica dei concetti trattati, con lo scopo di formare professionisti consapevoli dell'importanza strategica del testing nel contesto del ciclo di sviluppo software.

# 2. Introduzione

**Il testing nel ciclo di vita del software non è un'attività opzionale**, ma rappresenta una fase fondamentale per assicurare che il sistema sviluppato sia conforme alle specifiche e pronto per essere utilizzato in ambienti reali. L'importanza del testing deriva dal fatto che nessun sistema software può essere considerato immune da difetti: l'errore umano, la complessità crescente delle applicazioni, e l'interazione con ambienti esterni eterogenei rendono inevitabili le imperfezioni. Tuttavia, attraverso un testing ben progettato e accuratamente eseguito, è possibile individuare e correggere tali imperfezioni prima che abbiano impatti gravi sull'utente finale.

Il testing non si limita alla verifica del comportamento funzionale del sistema, ma comprende anche l'analisi della qualità complessiva del prodotto, attraverso test di **performance**, **affidabilità**, **scalabilità** e **sicurezza**. Questi aspetti rivestono un'importanza crescente in un mercato in cui le aspettative degli utenti sono sempre più elevate, non solo in termini di funzionalità, ma anche di qualità dell'esperienza d'uso. Un sistema che risponde correttamente ai comandi ma si presenta lento, instabile o vulnerabile non può essere considerato accettabile.

In tal senso, il processo di testing deve essere inteso come un'attività trasversale, che inizia già dalle prime fasi della progettazione e prosegue fino alla messa in produzione e oltre, attraverso la manutenzione e l'evoluzione del software. Le pratiche di **continuous testing** e **test-driven development (TDD)** rappresentano oggi approcci consolidati per garantire che la qualità venga assicurata fin dalle prime iterazioni di sviluppo.

Le attività di testing assumono forme differenti a seconda del livello di astrazione su cui si opera. Dalla verifica del singolo modulo (unit testing) fino alla validazione dell'intero sistema (system testing), ogni fase presenta obiettivi specifici e richiede strumenti e competenze differenti. Inoltre, il testing può essere classificato in **funzionale** e **non funzionale**, a seconda che si valuti il rispetto delle funzionalità dichiarate o le caratteristiche qualitative del sistema.

Un aspetto particolarmente rilevante del testing moderno è il suo grado di automazione. L'utilizzo di framework per il testing automatico permette di velocizzare i controlli, aumentare la copertura dei casi di test e integrare il testing continuo nel flusso di integrazione e distribuzione continua (CI/CD). Tuttavia, resta imprescindibile il contributo umano, soprattutto nelle fasi di progettazione dei test e nell'interpretazione dei risultati, in particolare per i test di usabilità e quelli esplorativi. Inoltre, alcune tipologie di test, come quelli relativi all'esperienza utente o alla sicurezza, richiedono l'adozione di approcci qualitativi e l'intervento di esperti di dominio.

In sintesi, il testing rappresenta una garanzia non solo di correttezza, ma anche di **affidabilità operativa, sicurezza e qualità percepita**, elementi fondamentali per il successo di qualsiasi prodotto software in contesti produttivi reali. È per questo che una strategia di testing ben pianificata e integrata fin dall'inizio del ciclo di vita del software costituisce uno dei principali fattori critici per il successo dei progetti digitali.

# 3. Livello di Testing

Il processo di testing si articola su più **livelli distinti**, ciascuno dei quali svolge un ruolo specifico all'interno del ciclo di vita del software. Ciascun livello è caratterizzato da obiettivi, metodi e strumenti peculiari, e consente di affrontare diverse tipologie di difetti, intervenendo a gradi di astrazione e complessità crescenti. Comprendere a fondo queste differenze è fondamentale per costruire una strategia di verifica efficace ed efficiente. Inoltre, il corretto ordine di esecuzione dei test, così come la strategia adottata per ciascun livello, influisce direttamente sulla qualità finale del software e sull'efficienza del processo di sviluppo.

Il primo livello, detto **unit testing**, è focalizzato sulla verifica del corretto funzionamento di singole unità di codice, come metodi, funzioni o classi. In questa fase, le unità vengono testate in isolamento, simulando eventuali dipendenze mediante l'uso di *test stub* e *test driver*. Il vantaggio principale di questo approccio è la capacità di individuare precocemente i difetti, riducendo il costo complessivo della loro correzione. Inoltre, l'automazione del unit testing è ampiamente supportata da numerosi framework (come JUnit, NUnit, PyTest), il che rende questa fase facilmente integrabile nei processi di sviluppo agili e iterativi. Un altro beneficio rilevante è la possibilità di documentare in modo strutturato il comportamento atteso delle unità, facilitando la manutenzione futura e il refactoring.

Successivamente si passa al **integration testing**, che si occupa di verificare l'interazione tra due o più unità precedentemente validate individualmente. Questo livello consente di identificare difetti legati a errori di interfacciamento, incongruenze nei protocolli di comunicazione, o dipendenze non correttamente gestite. A seconda della strategia adottata, l'integrazione può essere eseguita in modo *bottom-up*, *top-down*, *big bang* o con l'approccio *sandwich*, ciascuno dei quali presenta vantaggi e svantaggi specifici. Ad esempio, l'approccio *bottom-up* permette una verifica approfondita dei componenti fondamentali, mentre quello *top-down* favorisce una validazione precoce delle funzionalità a maggiore visibilità per l'utente finale. Il metodo *sandwich* si propone come un compromesso, ma richiede un'accurata pianificazione per evitare lacune nei test dei moduli centrali.

Il terzo livello è il **system testing**, in cui viene valutato il comportamento del sistema nel suo complesso, confrontandolo con i requisiti funzionali e non funzionali specificati. In questa fase il sistema è testato come se fosse già in produzione, simulando scenari realistici e coinvolgendo tutte le sue componenti. Il system testing permette di effettuare una verifica end-to-end del software, evidenziando eventuali problemi che non emergono nei livelli precedenti. È anche il momento in cui si valuta la resilienza complessiva del sistema, la coerenza dell'interfaccia utente, la stabilità sotto carico e il rispetto dei vincoli

prestazionali. Spesso il system testing prevede l'impiego di ambienti di test che replicano fedelmente quello di produzione, per garantire l'attendibilità dei risultati.

Infine, si giunge al **acceptance testing**, ovvero la fase in cui il sistema viene validato dal cliente o dagli utenti finali. Questo tipo di test ha l'obiettivo di accertare che il prodotto sia conforme alle specifiche contrattuali e pronto per la distribuzione. Le modalità tipiche includono il *benchmark test*, il *competitor test* e lo *shadow test*. L'accettazione può avvenire in forma esplicita, tramite la firma di un verbale, oppure in forma condizionata, con l'individuazione di modifiche da implementare prima della consegna definitiva. Il valore strategico dell'acceptance testing risiede nella sua capacità di consolidare il rapporto di fiducia tra sviluppatori e cliente, rappresentando il momento cruciale in cui le aspettative vengono formalmente confrontate con i risultati.

In conclusione, i diversi livelli di testing costituiscono una struttura a strati che, se ben orchestrata, consente di affrontare progressivamente le criticità del software, assicurandone la robustezza, l'affidabilità e la qualità complessiva. L'integrazione tra i livelli, unita a una pianificazione oculata e all'uso degli strumenti appropriati, costituisce la base per un processo di verifica realmente efficace ed efficiente.

# 4. Testing Funzionale e Non Funzionale

Il **testing funzionale** si concentra sulla verifica che il software soddisfi i requisiti specificati, valutando il comportamento del sistema in relazione a determinati input e osservandone gli output attesi. Si tratta di un approccio di tipo *black-box*, che non richiede la conoscenza del codice sorgente, ma si basa sull'analisi delle specifiche funzionali e dei casi d'uso. Tra le principali tecniche utilizzate troviamo il *test di equivalenza*, il *test ai limiti* e il *test basato su casi d'uso*. L'obiettivo è quello di assicurarsi che ogni funzione implementata risponda correttamente alle attese, includendo anche la gestione di condizioni eccezionali. Il testing funzionale comprende inoltre la verifica di scenari positivi e negativi, la gestione degli errori, e la coerenza dell'interazione utente-sistema. È uno strumento prezioso per accertare la completezza delle funzionalità richieste e la loro corretta implementazione, con un focus particolare sulle aspettative esplicite del cliente.

Di contro, il **testing non funzionale** si occupa di verificare aspetti del sistema che non riguardano direttamente ciò che il sistema "fa", bensì *come* lo fa. Include categorie come il testing di performance (stress, volume, scalabilità), di sicurezza, di affidabilità, di manutenibilità e, soprattutto, di usabilità. Queste caratteristiche, pur non essendo descritte come funzionalità operative, incidono profondamente sulla qualità percepita del software da parte degli utenti e sulla sua accettazione nel contesto produttivo. Un sistema funzionalmente corretto ma lento, difficile da usare o vulnerabile, fallisce comunque nel suo scopo. Il testing non funzionale implica quindi anche un'analisi delle metriche di qualità, come il tempo medio di risposta, la tolleranza agli errori, la resilienza sotto carico e l'aderenza a standard di sicurezza. È spesso più complesso da automatizzare rispetto a quello funzionale, e richiede un approccio multidisciplinare che coinvolga anche esperti in ergonomia, sicurezza informatica e architettura del sistema.

Entrambi i tipi di testing sono imprescindibili e complementari. L'efficacia di un sistema software si misura infatti nella sua capacità di rispettare le funzionalità dichiarate e di fornire un'esperienza utente soddisfacente, sicura e coerente anche in condizioni impreviste o avverse. La pianificazione del testing non funzionale deve avvenire fin dalle prime fasi del progetto, in modo da individuare precocemente i criteri di accettazione qualitativa e predisporre ambienti di prova idonei a simularne le condizioni. La combinazione strategica di testing funzionale e non funzionale costituisce la chiave per la realizzazione di un prodotto solido, performante e pronto a soddisfare le esigenze di un mercato sempre più esigente.

# 5. Testing di Sistema e Accettazione

Il **testing di sistema** rappresenta una fase cruciale nella quale il software viene esaminato nella sua totalità, per valutare se soddisfa tutti i requisiti funzionali e non funzionali definiti nelle fasi precedenti dello sviluppo. Questo tipo di testing è eseguito in un ambiente che simula il più fedelmente possibile il contesto reale di produzione. L'obiettivo principale è quello di effettuare una verifica end-to-end dell'intero sistema, controllando l'interazione tra i diversi moduli software e le interfacce esterne. Durante questa fase, vengono utilizzati scenari realistici che coprono sia i flussi principali di utilizzo, sia quelli alternativi ed eccezionali. È anche in questo stadio che si possono individuare problemi legati all'integrazione tra moduli ben funzionanti singolarmente, ma che generano errori nel contesto globale. Questo fenomeno, noto come **faults emergenti dall'integrazione**, è spesso difficile da prevedere e sottolinea l'importanza di un testing di sistema ben strutturato.

Il testing di sistema include sia **test funzionali** (che valutano il comportamento del sistema rispetto ai casi d'uso) sia **test non funzionali** (che esaminano prestazioni, usabilità, sicurezza, ecc.). La sua efficacia dipende fortemente dalla qualità dei test case definiti e dalla copertura dei requisiti specificati. Inoltre, è fondamentale che venga effettuato in un ambiente isolato e controllato, per evitare interferenze che possano falsare i risultati. Gli strumenti di automazione possono essere utilizzati per simulare carichi realistici e per raccogliere metriche sulle performance del sistema. Tali strumenti permettono anche di ripetere i test in modo sistematico e di identificare regressioni che possono verificarsi in seguito a modifiche del codice. Un altro aspetto importante è l'adozione di **metriche quantitative**, come la percentuale di copertura del codice, il numero di errori rilevati per modulo, o il tempo medio di risposta, che aiutano a misurare oggettivamente il grado di maturità del sistema.

Una volta completato con successo il testing di sistema, si procede al **testing di accettazione**. Questa è l'ultima fase prima della consegna del software, in cui il prodotto viene validato dal cliente o dagli utenti finali. Lo scopo è verificare che il sistema soddisfi le aspettative concordate nel contratto o nei documenti di specifica. Le modalità più comuni includono:

- **Benchmark test**, in cui il cliente esegue casi di test predefiniti che rappresentano l'uso tipico del sistema;
- **Competitor test**, in cui il nuovo sistema viene confrontato con uno preesistente o con un prodotto concorrente;
- **Shadow test**, dove il nuovo sistema viene eseguito in parallelo a quello esistente per confrontarne il comportamento.

Esistono anche approcci più esplorativi come i **beta test**, in cui una versione quasi definitiva del prodotto viene distribuita a un gruppo ristretto di utenti reali, e i **test pilota**, che consistono nell'utilizzo controllato del sistema in un ambiente reale prima del rilascio ufficiale. Entrambe le strategie permettono di raccogliere **feedback autentici** da parte degli utenti finali e di valutare aspetti altrimenti difficili da simulare in ambiente di test.

L'esito del testing di accettazione può essere positivo (con il rilascio formale del sistema), condizionato (richiesta di modifiche minori), o negativo (richiesta di rielaborazioni significative). In ogni caso, rappresenta un momento di confronto cruciale tra le parti e un'opportunità per definire eventuali aggiornamenti ai requisiti, sulla base di un'esperienza d'uso diretta. È anche il punto in cui si consolidano le responsabilità contrattuali e si apre ufficialmente la fase di manutenzione e supporto.

Il testing di sistema e il testing di accettazione insieme garantiscono che il software non solo sia completo e conforme, ma anche pronto per l'utilizzo nel mondo reale. La cura nella definizione di scenari, criteri di accettazione e ambienti di esecuzione fa la differenza tra un rilascio fluido e uno problematico. Un investimento accurato in queste fasi finali del testing si traduce in maggiore soddisfazione del cliente, minor rischio di incidenti post-rilascio e miglioramento continuo del processo di sviluppo.

# 6. Conclusioni e sintesi

Il testing, lungi dall'essere una fase accessoria dello sviluppo software, costituisce un pilastro essenziale per garantire che i sistemi siano non solo funzionanti, ma anche affidabili, sicuri, usabili e performanti. Una strategia di testing ben strutturata consente di ridurre i costi di manutenzione, di migliorare l'esperienza utente e di prevenire problematiche che, se scoperte tardi, possono avere impatti economici e reputazionali devastanti. Il valore del testing non si limita all'ambito tecnico: rappresenta anche un atto di trasparenza e di fiducia nei confronti del cliente, un mezzo per dimostrare la qualità e l'impegno con cui è stato realizzato il prodotto software.

Abbiamo visto come i diversi livelli di testing, dall'unità fino all'accettazione, permettano di affrontare progressivamente i rischi legati allo sviluppo. Ogni livello ha un ruolo ben preciso e complementare: il testing di unità garantisce l'integrità delle singole componenti, quello di integrazione verifica la coerenza tra moduli, quello di sistema conferma il rispetto dei requisiti globali, mentre l'accettazione sancisce l'idoneità del prodotto al contesto reale. Abbiamo distinto i test funzionali, centrati sulla correttezza del comportamento, da quelli non funzionali, orientati alla qualità complessiva del sistema. Abbiamo anche esplorato le strategie di integrazione, gli strumenti di supporto e le tecniche per massimizzare l'efficacia del processo di verifica, inclusi gli approcci automatizzati, l'uso di metriche e la partecipazione attiva degli stakeholder.

**In sintesi, un software di qualità è quello che funziona correttamente, si comporta bene anche sotto stress, è facile da usare e difficile da compromettere.** È un software che ispira fiducia, facilita il lavoro dell'utente, resiste agli imprevisti e si adatta all'evoluzione dei bisogni. Il testing è la chiave per trasformare il codice in un prodotto affidabile e di valore, contribuendo a costruire soluzioni digitali sostenibili, durature e capaci di generare un impatto positivo nel contesto in cui sono adottate.

# Bibliografia


- Bruegge, B., & Dutoit, A. H. (2010). Object-oriented software engineering. Using UML.