Il collaudo di un sistema informatico

a cura di Ing. Massimo Di Carlo

1. Nell'ambito delle attività tipiche della professione di Ingegnere un importante aspetto è lo svolgimento di collaudi di opere di ingegneria per verificarne la corrispondenza ai requisiti definiti dal committente o derivati da normative e regolamenti (tra cui per esempio i requisiti di sicurezza).
Anche nel campo dell'Ingegneria del software il collaudo riveste un'importanza decisiva per garantire la qualità del prodotto.
I sistemi software sono parte integrante della vita, dalle applicazioni aziendali (ad es. bancarie), ai prodotti di consumo (ad es. le automobili).

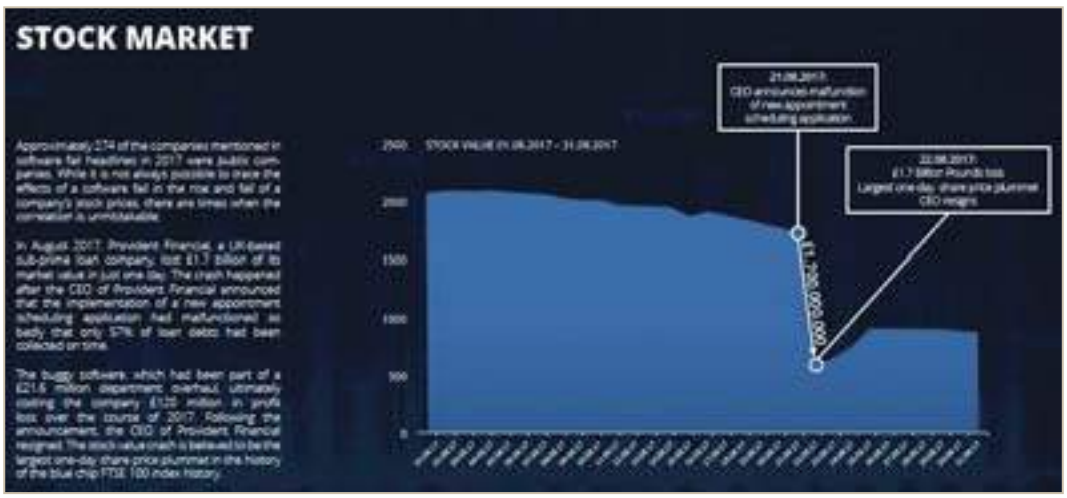
Il software è ormai presente in tutte le attività quotidiane e ci affidiamo di continuo a sistemi il cui nucleo centrale è formato da applicazioni software ed è difficile immaginare di vivere senza di esso in molte delle nostre attività professionali o nella vita di tutti i giorni. Pensiamo per esempio ai rapporti bancari oggi utilizzati per lo più tramite ATM e da casa via computer o con app su smartphone, oppure all'acquisto di biglietti per viaggi o per spettacoli.
Il software è diventato così pervasivo che quando ha un malfunzionamento le nostre attività giornaliere ne sono altamente impattate. Se questo vale a livello personale ancora più drammatici sono

gli effetti di anomalie per il business delle aziende: la perdita di denaro, di tempo, di reputazione aziendale e persino infortuni o morte (pensiamo ad esempio ai sistemi di guida e controllo dei veicoli aerei, ai sistemi di controllo delle centrali di produzione di energia, ai sistemi di sicurezza e di controllo delle automobili).
Dai dati relativi al 2017 riportati dal Software Fail watch (https://www.tricentis.com/software-fail-watch/) - redatto in base alle notizie di failure del software apparse in articoli in lingua inglese e facendo una stima conservativa (si basa solo sulle notizie conosciute, quindi senza contare incidenti non venuti alla luce) - le perdite causate da errori nel software nel 2017 sono pari a 1,715,430,778,504 dollari.

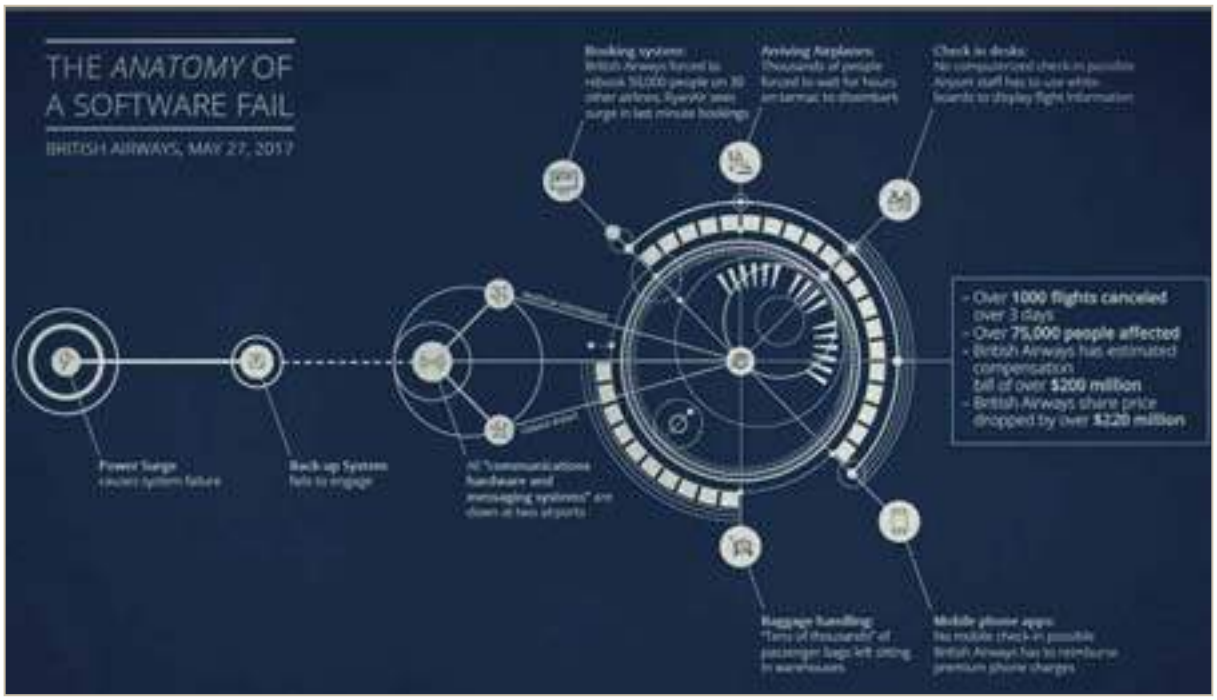
FAILURE=un evento in cui un componente o un sistema non esegue una richiesta funzionalità all'interno di limiti determinati (dal glossario ISTQB).

Nel report ci sono degli esempi rilevanti di impatti sul business delle aziende.
Circa 274, tra le aziende menzionate nei titoli relativi a failure del software, sono public company. Anche se non è sempre possibile tracciare gli effetti di un incidente provocato dal software sull'andamento dei titoli in borsa c'è un esempio in cui tale correlazione è chiara.
Nell'agosto 2017, Provident Financial, un'azienda britannica di prestiti sub-prime, ha perso 1,7 miliardi di sterline del suo valore di mercato in un solo giorno. Il crollo è avvenuto dopo che il CEO dell'azienda ha annunciato che l'implementazione di una nuova applicazione di gestione degli appuntamenti aveva funzionato così male che solo il 57% dei debiti era stato raccolto entro la scadenza.
Il software difettoso, che faceva parte di una ristrutturazione dal costo di 21,6 milioni di sterline, è costato complessivamente alla società 120 milioni di perdite di profitto nel corso del 2017. Dopo l'annuncio, il CEO di Provident Financial si è dimesso. Il crollo del valore azionario si crede sia stato il tonfo più grande in un solo giorno della storia dell'indice FTSE 100 blu chip.

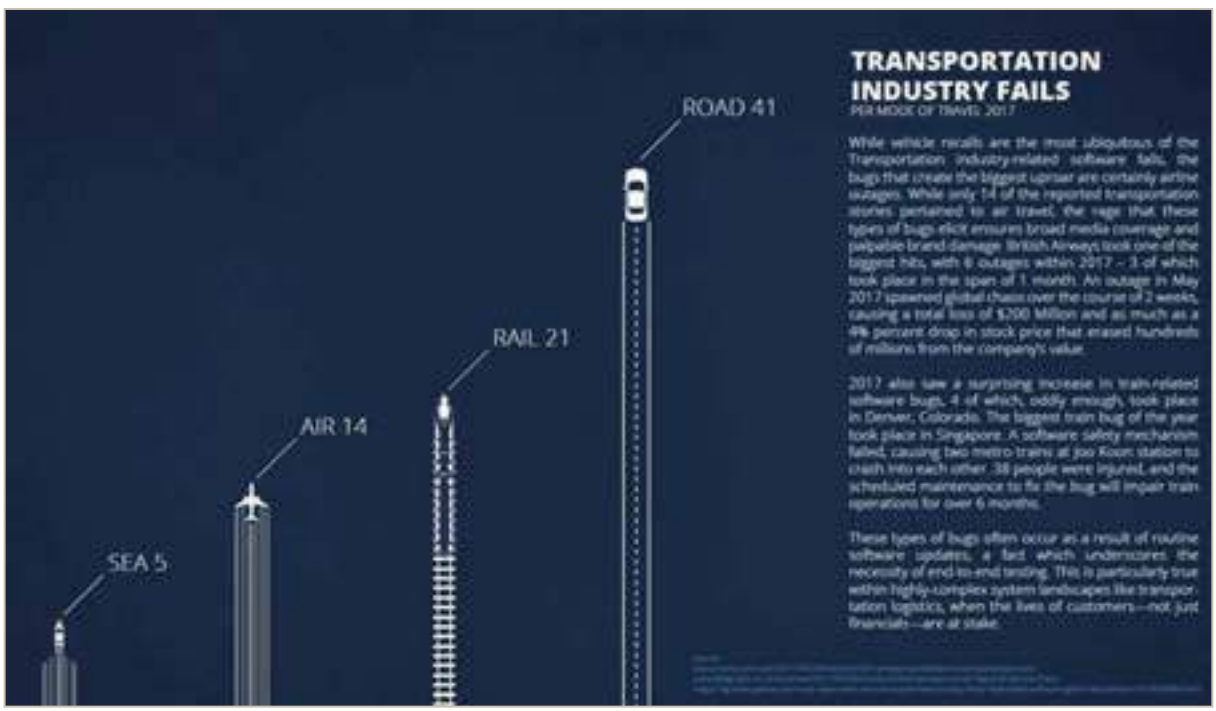
Un altro esempio di un pesante impatto sul business ha coinvolto la compagnia aerea British Airways



nel maggio 2017: un difetto nel software ha comportato la cancellazione di 100 voli in 3 giorni, ha coinvolto 75.000 persone e una perdita di 200 milioni di sterline solo legati ai rimborsi. I dati completi nella figura seguente.



Nella figura seguente si può inoltre vedere una sintesi degli incidenti nel settore dei trasporti.



Ci sono anche dei fallimenti storici nel campo dei voli spaziali:
Il razzo Ariane 5 ha riutilizzato la stessa piattaforma inerziale del precedente Ariane 4. Ma la maggiore accelerazione orizzontale dell'Ariane 5 ha provocato un crash dei sistemi a causa del tentativo di inserire un dato a 64 bit in una variabile a 16 bit portando a valori errati nel sistema di guida con conseguente perdita del vettore.
Nella sua missione verso Marte nel 1998 il Climate Orbiter spacecraft si è in pratica perso nello spazio a causa della mancata conversione da unità di misure inglesi verso il sistema metrico. L'errore ha provocato la mancata entrata in orbita della sonda costata 125 milioni di dollari che si è andata a perdere in un'orbita intorno al sole.
La sonda Mariner 1, in missione verso Venere nel 1962, era appena partita da Cape Canaveral quando un baco nel software ha provocato una virata che rischiava di far schiantare il razzo sulla Terra con conseguente invio del comando di autodistruzione da parte del centro di controllo per non provocare danni gravi. Successivamente si è visto che il problema era stato provocato dalla mancanza di un trattino nelle istruzioni del codice. Il costo del razzo è stato indicato come 18 milioni di dollari dell'epoca.
Si possono trovare questi e altri esempi a questo link: https://raygun.com/blog/costly-software-errors-history/

2. Per ridurre la probabilità di incorrere in incidenti come quelli riportati e ridurre di conseguenza le perdite in termini economici – oltreché evidentemente di vite umane - è necessario applicare, anche nell'ambito della produzione del software, standard e metodologie che migliorino la qualità dell'intero ciclo di sviluppo del software dalla fase di analisi e progettazione, alla realizzazione, al collaudo.
Se le fasi alte di progettazione e realizzazione sono viste come fasi in cui sono necessari standard e professionalità elevati per migliorare la qualità, la fase di collaudo purtroppo è quella che ancora troppo spesso non riceve la giusta importanza e di cui non si comprende quanto sia fondamentale per migliorare la qualità dei prodotti. In realtà nel campo dello sviluppo software quello che manca è proprio una cultura diffusa del collaudo: scopo, obiettivi, metodi e tecniche.
Negli anni si sono fatti passi avanti, ma vediamo cosa dice **Glenford J. Myers** - uno dei maggiori esperti sul tema del collaudo del software - nell'introduzione del suo libro che è un po' il testo sacro del software testing ("The Art of Software Testing" - 2011):
"Given these facts, you might expect that by

this time program testing would have been refined into an exact science. This is far from true. In fact, less seems to be known about software testing than about any other aspect of software development. Furthermore, testing has been an out-of-vogue subject—it was true when this book was first published (*1979 n.d.a.*) and, unfortunately, it is still true today (*2011 n.d.a.*). Today there are more books and articles about software testing, meaning that, at least, the topic has greater visibility than it did when this book was first published. But testing remains among the "dark arts" of software development".
Quindi nel mondo dell'ingegneria del software non si è ancora consolidata in modo pervasivo una cultura del collaudo del software, né una diffusione adeguata delle necessarie professionalità. C'è ancora scarsa attenzione anche in ambito universitario. Così scrive ancora Myers: "At various times, we have heard professors and teaching assistants say, "Our students graduate and move into industry without any substantial knowledge of how to go about testing a program. Moreover, we rarely have any advice to provide in our introductory courses on how a student should go about testing and debugging his or her exercises." So, the purpose of this updated edition of *The Art of Software Testing* is the same as it was in 1979: to fill these knowledge gaps for the professional programmer and the student of computer science".
Lo stesso concetto è espresso da Rex Black nel suo libro "Managing the Testing Process" (2011): "When I first started working as a test engineer and test project manager, I was a testing ignoramus - … -Unfortunately, my university professors didn't teach about testing, even though Boris Beizer, Bill Hetzel, and Glenford Myers had all published on the topic prior to or during my college career. As software engineering enters its sixth decade, that has begun to change. However, even at prestigious universities the level of exposure to testing that most software-engineers-in-the-making receive remains too low".
Viste le problematiche e i relativi pesanti costi che possono derivare dagli insuccessi è decisivo che penetri in modo esteso la cultura del testing come parte essenziale del processo di sviluppo e che si formino professionisti del test che conoscano ed applichino metodologie e tecniche del test adatte allo specifico ambito.
Vediamo di seguito un rapido excursus sugli scopi, i principi, i metodi e le tecniche principali del software testing, partendo, anzitutto, da cosa si intende per collaudo del software:
"Software testing is a process, or a series of processes, designed to make sure computer code does what it was designed to do and that it does

not do anything unintended. Software should be predictable and consistent, offering no surprises to users" (Myers, cit.).
In sintesi, il testing di un Sistema Informatico è un processo di controllo che consente di evidenziare errori indotti dal processo produttivo, anticipando e minimizzando l'impatto degli interventi correttivi.
È quindi un modo per valutare la qualità del software e per ridurre il rischio di suoi malfunzionamenti.
Il collaudo del software si pone due obiettivi: verifica e validazione:

- Verifica: risponde alla domanda: "stiamo sviluppando il sistema correttamente?"
  Il software deve soddisfare le specifiche
- Validazione: risponde alla domanda: "stiamo sviluppando il prodotto corretto?"
  Il software deve fare quello che l'utente realmente richiede.

È importante definire esattamente lo scopo del test in quanto una delle cause principali di un testing non ben fatto consiste nell'approccio che si ha anche dal punto di vista psicologico se si parte da false definizioni del testing come ad esempio:

- "Il testing è un processo per dimostrare che non sono presenti errori"
- "Obiettivo del testing è mostrare che un programma esegua correttamente le funzioni per cui è stato realizzato"
- "Il testing è un processo per stabilire la confidenza che un programma fa quello che deve fare."

Queste definizioni sono da rovesciare perché altrimenti l'approccio mentale sarebbe nel senso di cercare di dimostrare quanto affermato, cioè in sintesi che il sistema funziona come previsto. Inoltre, la prima frase "…non sono presenti errori" è impossibile da realizzare perché non è possibile eseguire il software in tutte le molteplici condizioni di funzionamento.
Quindi possiamo definire che **scopo del collaudo** *è trovare i difetti per ridurre il rischio di malfunzionamenti durante l'uso in esercizio, tenendo conto che un Testing che elimina tutti gli errori è impossibile.*
In un mondo ideale, si vorrebbe poter testare ogni possibile permutazione dei dati di input di un programma, ma in pratica non è mai possibile. Anche un semplice programma può avere centinaia o migliaia di possibili combinazioni di

input e output. Creare dei casi di test per tutte queste combinazioni è impraticabile. Testare completamente applicazioni complesse richiederebbe tempi troppo lunghi e troppe risorse per essere economicamente sostenibile.
Pertanto, l'unico obiettivo che ci si può porre è ridurre il rischio e fissare degli obiettivi per il test in modo tale da minimizzare i rischi e i costi.
Si deve quindi porre come obiettivo del test il raggiungimento di un livello di rischio accettabile, valutando l'impatto sugli utenti di un possibile malfunzionamento.
L'impatto in caso di occorrenza di un evento negativo è spesso inteso come la gravità degli effetti sugli utenti, sui clienti o sugli altri stakeholder: in altre parole, come rischio di business.
Per valutare il livello di rischio accettabile si deve identificare e valutare il potenziale impatto sul dominio di business o sugli utenti per ogni elemento di rischio.
Tra i fattori che influenzano il rischio di business ci sono:

- Frequenza di utilizzo delle funzionalità interessate
- Dimensioni della perdita di business
- Potenziali responsabilità o perdite finanziarie, ecologiche o sociali
- Sanzioni legali civili o penali
- Preoccupazioni sulla sicurezza
- Visibilità della funzione del sistema impattata
- Visibilità del difetto che induce una pubblicità negativa e un potenziale danno d'immagine
- Potenziale perdita di clienti.

In base all'impatto potenziale del rischio residuo si può valutare quanto investire nel collaudo per cercare di raggiungere un punto di massima riduzione del rischio e di minimizzazione dei costi.
Nella figura seguente è esemplificato questo concetto; la curva del costo delle conformità rappresenta il costo di tutte le attività eseguite per aumentare la qualità del prodotto. La curva del costo delle non conformità rappresenta il costo degli effetti del rischio residuo. Più alta è la qualità minore è il rischio residuo del prodotto. L'ottimo si ottiene quando si ha il massimo della riduzione del rischio al minor costo.

Definito il livello di rischio si devono definire dei criteri di arresto del collaudo in modo da raggiungere gli obiettivi prestabiliti.
I criteri d'arresto sono definiti per il progetto a livello del piano di qualità o di un più specifico piano di test e collaudo, che predetermina l'ottimale ripartizione delle risorse economiche, umane e strumentali fra le differenti entità da testare, fra le differenti attività e nell'acquisto degli strumenti.
Un criterio di arresto consiste per esempio nello stimare il numero di errori attesi e confrontare tale valore con il numero di quelli riscontrati, basando il criterio d'arresto su questo e non solo sul rispetto di tempi e budget.
Molto spesso quindi si definisce una soglia di accettabilità oltre la quale si passa a testare altre entità. In casi particolari, quando si testano moduli critici per il buon funzionamento del sistema informatico, l'assenza di malfunzionamenti riscontrati non va interpretata come assenza di difetti, ma come scarsa idoneità od efficacia delle tecniche di test utilizzate (ovvero come mancanza di padronanza delle stesse). Si rende quindi necessaria la ripetizione di attività di progettazione e preparazione dei casi di test che si ritenevano concluse e non della sola esecuzione, onde arricchire e rendere più probante il superamento di una sessione.
Quanto sopra indicato e gli altri principi base sul software testing si possono sintetizzare nei 7 principi del testing (dal Syllabus Livello "Foundation" – versione 2018 – ITASTQB).

Principio 1
Il testing mostra la presenza di difetti
Il testing può mostrare la presenza di difetti, ma non può provarne l'assenza. Il testing riduce la probabilità della presenza di difetti non rilevati nel software, ma il fatto che nessun difetto venga trovato, non è una prova di correttezza, ovvero di assenza di difetti.

Principio 2
Il testing esaustivo è impossibile
Testare tutto (tutte le combinazioni di input e precondizioni) non è possibile tranne che per casi banali. Piuttosto che tentare di testare in modo esaustivo, è necessario utilizzare l'analisi del rischio, le tecniche di test e le priorità per concentrare gli effort di testing.

Principio 3
Il testing anticipato permette di risparmiare tempo e denaro
Per individuare tempestivamente i difetti è necessario avviare quanto prima le attività di testing statico e dinamico nel ciclo di vita dello

sviluppo software. Testare all'inizio del ciclo di vita dello sviluppo del software aiuta a ridurre o eliminare costose modifiche successive.

Principio 4
I difetti tendono a formare cluster
Un numero limitato di moduli solitamente contiene la maggior parte dei difetti scoperti durante i test prima del rilascio o è responsabile della maggior parte delle failure operative. I cluster dei difetti previsti e quelli effettivamente osservati in fase di test o di esercizio sono un input importante per un'analisi del rischio al fine di concentrare l'effort di testing (come menzionato nel principio 2).

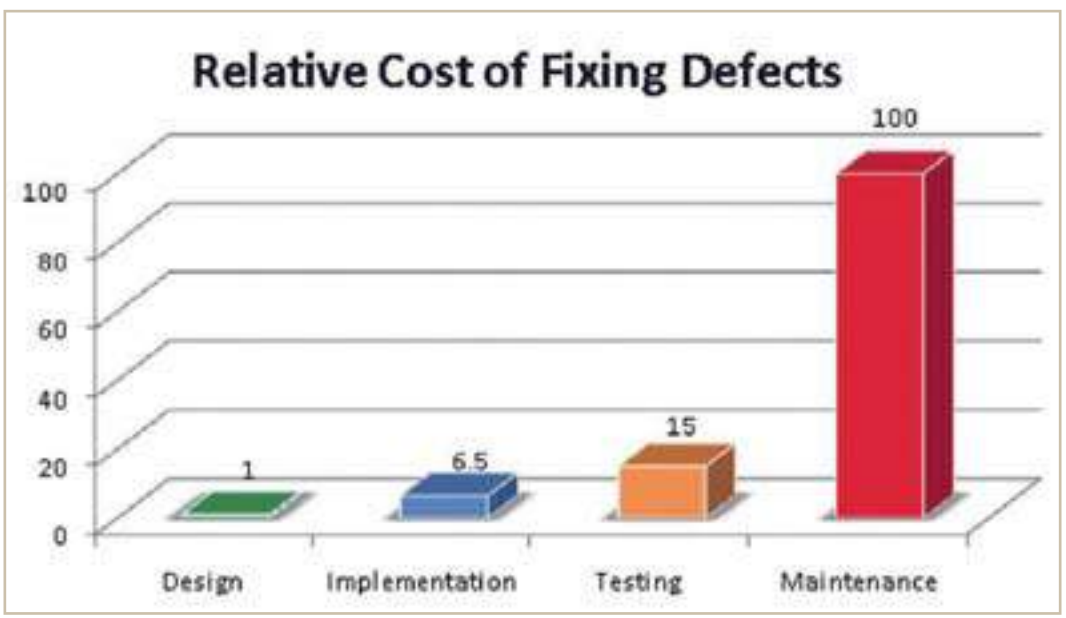
Principio 5
Attenzione al paradosso pesticida
Se gli stessi test sono ripetuti più e più volte, presumibilmente tali test non troveranno nessun nuovo difetto. Per rilevare nuovi difetti, i test esistenti e i dati di test potrebbero necessitare di modifiche e potrebbe essere necessario progettare nuovi test (i test non risultano più efficaci nel trovare difetti, proprio come i pesticidi non risultano più efficaci nell'uccidere gli insetti dopo un utilizzo protratto nel tempo).

Principio 6
Il testing è dipendente dal contesto
Il testing viene eseguito in modo differente in contesti differenti. Per es. un software safety-critical viene testato in modo differente da una app mobile di commercio elettronico. Come altro esempio, il testing in un progetto Agile viene condotto in modo diverso rispetto al testing di un progetto con ciclo di vita sequenziale.

Principio 7
L'assenza di errori è una falsa credenza
Alcune organizzazioni si aspettano che i tester possano eseguire tutti i test possibili e trovare tutti i possibili difetti, ma i principi 2 e 1 ci dicono che questo è impossibile. Inoltre, non è corretto aspettarsi che il solo trovare e risolvere un gran numero di difetti garantisca il successo di un sistema. Ad es. testare accuratamente tutti i requisiti specificati e correggere tutti i difetti rilevati potrebbe ancora produrre un sistema che sia difficile da usare, che non soddisfi i bisogni e le aspettative degli utenti o che risulti meno valido rispetto ad altri sistemi concorrenti.

3. È importante sottolineare quanto sopra indicato dal principio 3.
Le attività di testing devono iniziare quanto prima nel corso del ciclo di vita dello sviluppo del software per poter intercettare il prima possibile un difetto. Infatti, un difetto – o una non conformità rispetto al comportamento atteso – può derivare da una specifica dei requisiti utente errata, ambigua o incompleta, oppure dalla mancata comprensione di un requisito e quindi nella sua traduzione in una specifica funzionale errata, oppure da errori inseriti nel codice.
I difetti possono essere introdotti in qualsiasi punto del ciclo di vita ed in qualsiasi prodotto software correlato. Pertanto, ogni fase del ciclo

Fig 1 IBM System Science Institute



di vita dello sviluppo software dovrebbe includere attività per rilevare e rimuovere potenziali difetti. Ad esempio, le tecniche di test statici (per es. revisioni e analisi statica) possono essere utilizzate sulle specifiche dei requisiti, le specifiche di progettazione e il codice software, prima di consegnare questi prodotti per le attività successive. Quanto prima un difetto viene rilevato e rimosso, tanto minore è il costo complessivo della qualità per il sistema. Il costo della qualità è minimizzato quando ogni difetto viene rimosso nella stessa fase in cui è stata introdotto, cioè quando il processo di software raggiunge il livello di *contenimento di fase*.
Un esempio potrebbe essere quello di un requisito errato che è identificato durante il riesame dei requisiti e lì viene corretto. Questo non è solo un uso efficiente della revisione dei requisiti, ma impedisce anche che il difetto causi del lavoro in più che lo renderebbe più costoso. Infatti, se un requisito non corretto sfugge al riesame dei requisiti e viene successivamente implementato dagli sviluppatori, testato senza essere scoperto e rilevato dall'utente durante il test di accettazione, tutto il lavoro svolto su tale requisito costituisce uno spreco di tempo e di risorse.
Nella figura 1 si può vedere la proporzione tra i costi di risoluzione di un difetto nelle varie fasi del ciclo di vita del software posto a 100 il costo di rimozione nella fase di manutenzione quindi quando il software è già stato rilasciato in esercizio. (https://www.jrothman.com/articles/2000/10/what-does-it-cost-you-to-fix-a-defect-and-why-should-you-care/).
Per i motivi su elencati, è importante che un professionista nel campo del test del software abbia familiarità con i modelli del ciclo di vita dello sviluppo software, in modo che per ogni fase possano essere condotte le attività di test più adeguate.
In ogni modello del ciclo di vita dello sviluppo software, ci sono diversi punti da considerare per un testing efficace:

- Per ogni attività di sviluppo, esiste una corrispondente attività di test
- Ogni livello di test ha obiettivi di test specifici per quel livello
- L'analisi e la progettazione per un determinato livello di test iniziano durante la corrispondente attività di sviluppo
- Le persone del gruppo di collaudo partecipano agli incontri per definire e perfezionare i requisiti e la progettazione e sono coinvolti nella revisione dei prodotti di lavoro (ad es. requisiti, progettazione, user story, ecc.) non appena siano disponibili delle loro bozze.

I comuni modelli del ciclo di vita dello sviluppo del software si possono classificare come:

- Modelli di sviluppo sequenziali
- Modelli di sviluppo iterativi e incrementali.

Un modello di *sviluppo sequenziale* descrive il processo di sviluppo del software come un flusso lineare e sequenziale di attività. Ciò significa che qualsiasi fase del processo di sviluppo deve iniziare quando la precedente fase è completata (figura 2).
Lo *sviluppo incrementale* comporta la definizione di requisiti, la progettazione, lo sviluppo e il testing di un sistema per parti, il che significa che le funzionalità del software crescono in modo incrementale. La dimensione di questi incrementi di funzionalità varia, con alcuni metodi che prevedono incrementi più grandi e altri più piccoli. Gli incrementi di funzionalità possono essere anche molto piccoli come una singola modifica a una schermata dell'interfaccia utente o una nuova opzione di query.
Lo *sviluppo iterativo* si adotta quando gruppi di funzionalità sono specificati, progettati, sviluppati e testati insieme, in una serie di cicli spesso di durata fissa. Le iterazioni possono

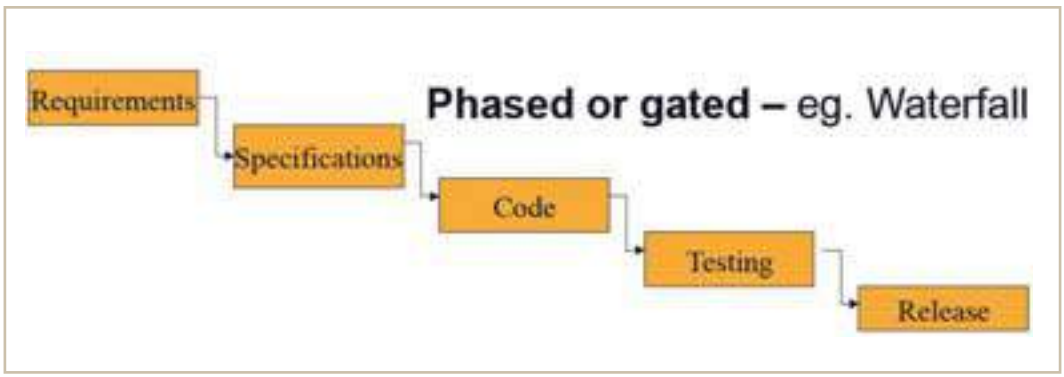


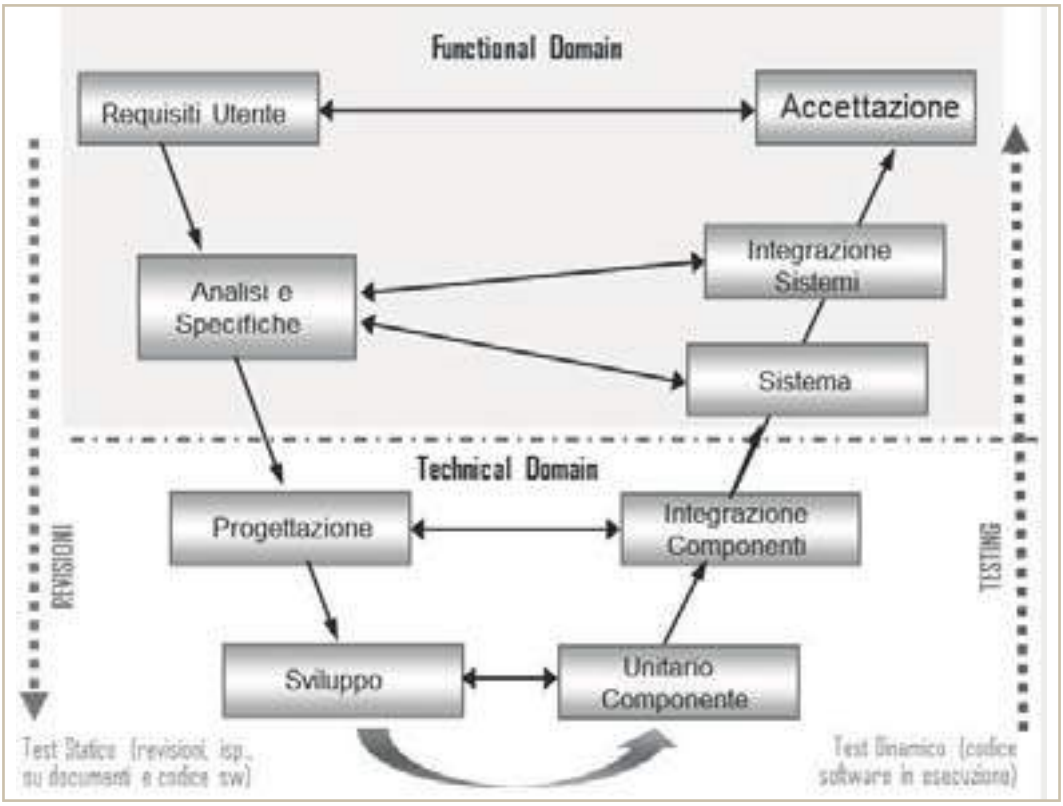
Fig 2

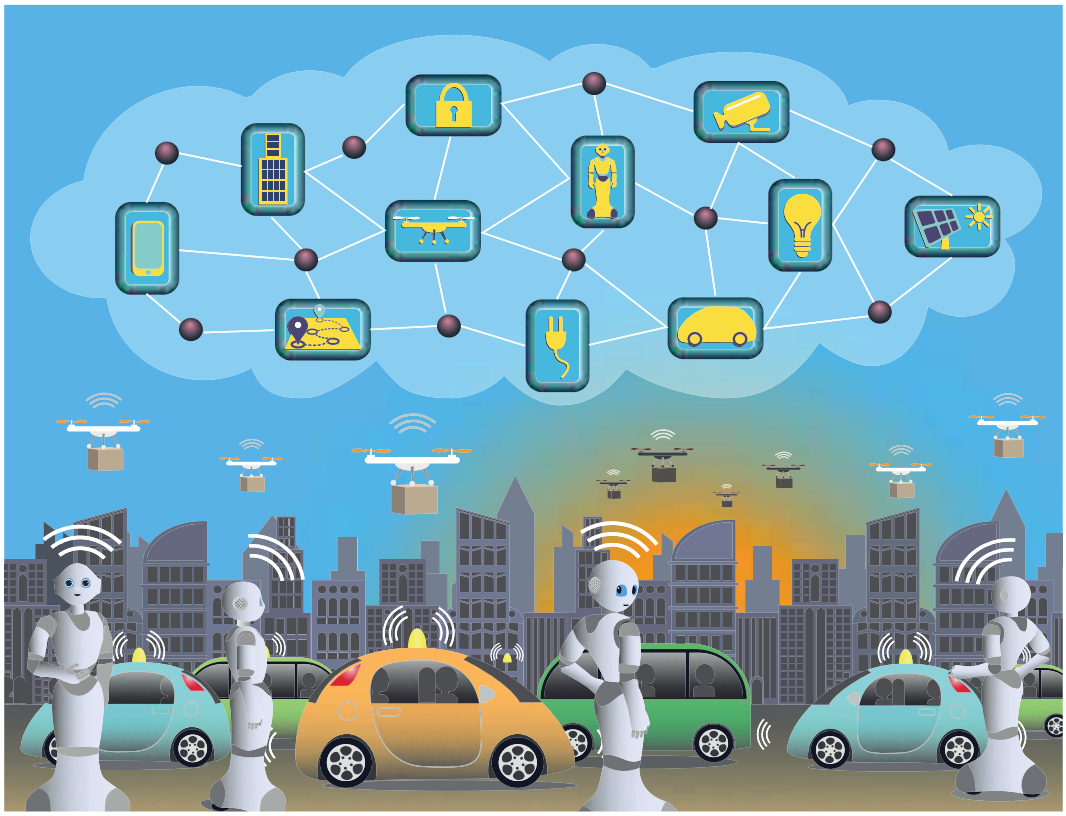
comportare modifiche alle funzionalità sviluppate nelle iterazioni precedenti, in linea con le modifiche nell'ambito del progetto. Ogni iterazione consegna un software funzionante, che è un sottoinsieme crescente dell'insieme complessivo delle funzionalità, fino al rilascio del software finale o all'interruzione dello sviluppo. Un esempio è lo sviluppo Agile (figura 3).

Fig 3

Tipici **modelli sequenziali** sono il **modello Waterfall** in cui le attività di sviluppo (ad es. analisi dei requisiti, progettazione, codifica, testing) sono completate una dopo l'altra. In questo modello le attività di test sono svolte solo dopo che tutte le altre attività di sviluppo sono state completate.
A differenza del modello Waterfall, il **modello a V** integra il processo di test durante tutto il processo di sviluppo, attuando il principio del testing anticipato. Inoltre, il modello a V include livelli di test associati a ciascuna fase di sviluppo corrispondente, supportando ulteriormente il testing anticipato. In questo modello l'esecuzione dei test associati a ciascun livello di test procede in sequenza, anche se alcuni casi si verificano sovrapposizioni (in figura 4 uno schema del modello a V).
Il modello mostra la relazione tra ogni fase del ciclo di vita dello sviluppo del software e la sua fase di test. Ad ogni fase corrisponde un livello di test. Il prodotto della fase di sviluppo è l'input per la progettazione del test del corrispondente livello. Nel progettare il test si analizza il prodotto dello sviluppo riuscendo ad evidenziare eventuali problemi.
Per esempio per la progettazione dei test di accettazione si parte dai requisiti utente, in questo caso il tester può evidenziare l'ambiguità o la mancanza di un requisito se non riesce a

Fig 4 Schema del modello a V





comprenderlo o non ha sufficienti informazioni per scrivere un test consistente, individuando quindi un difetto prima che si propaghi alle fasi successive.

4. Abbiamo parlato di livelli di test, vediamo cosa si intende specificamente per Livello di Test.

Livelli di Test

I livelli di test sono gruppi di attività di test che sono organizzati e gestiti insieme. Ogni livello di test è un'istanza del processo di test con le relative attività svolte in relazione al software a un determinato livello di sviluppo (da singole unità o componenti a sistemi completi o a sistemi di sistemi).

I livelli di test di norma utilizzati sono:

- Testing di componente: vengono testati i singoli moduli
- Testing di integrazione: si integrano man mano i moduli per trovare eventuali problemi che nascano dalla loro interazione
- Testing di sistema: è il test di tutto il sistema completo quando tutti i componenti sono stati integrati
- Testing di accettazione: è il collaudo finale di solito svolo con il client o dal cliente stesso per verificare la corrispondenza con i requisiti.

Ci sono vari tipi di test che si possono svolger ai vari livelli.

Tipi di Test

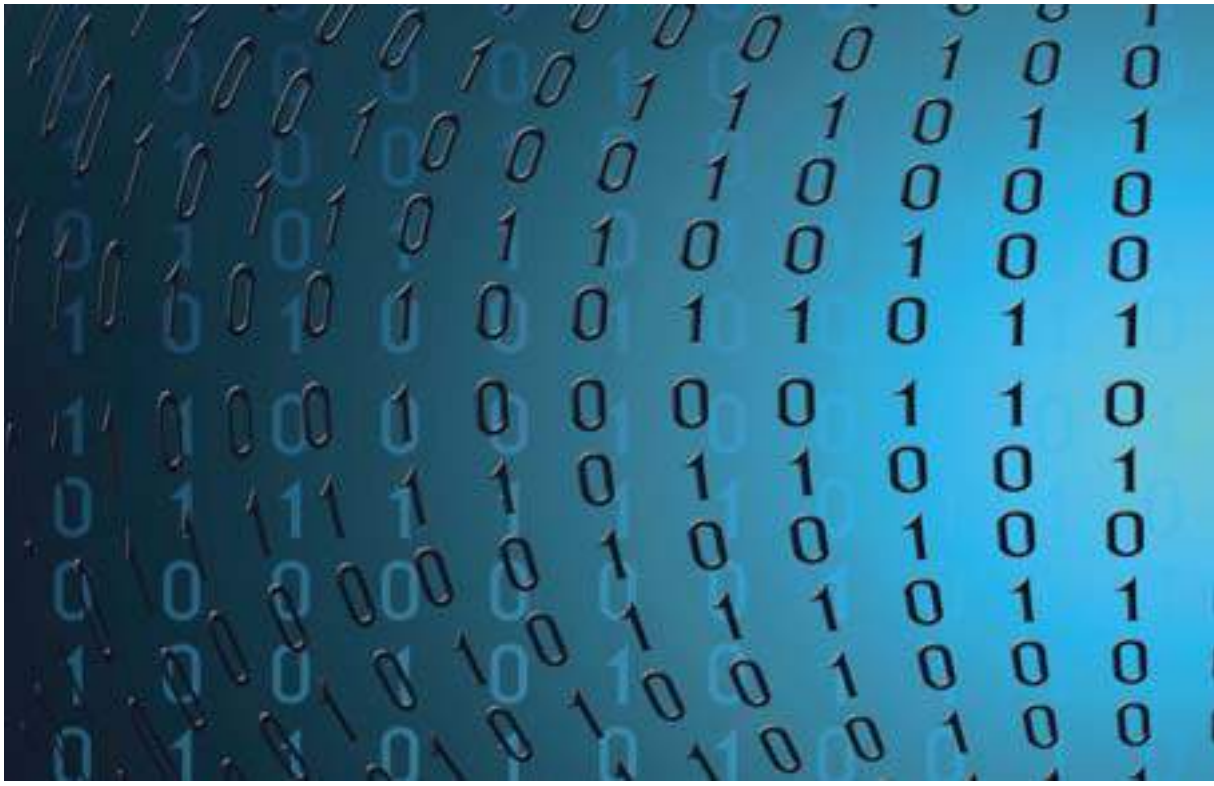
Un tipo di test è un gruppo di attività di test volte a testare caratteristiche specifiche di un sistema software o di una sua parte, sulla base di obiettivi di test specifici. Tali obiettivi includono:

- Valutare le caratteristiche di qualità funzionale, quali completezza, correttezza e appropriatezza
- Valutare le caratteristiche di qualità non-funzionali, quali affidabilità, efficienza prestazionale, sicurezza, compatibilità e usabilità
- Valutare gli effetti delle modifiche, ad es. confermando che i difetti sono stati corretti (testing confermativo) e cercando alterazioni involontarie nel funzionamento risultanti da modifiche software o di ambiente (testing di regressione).

Di norma si possono dividere in test funzionali e test non funzionali.

Testing Funzionale

I test funzionali di un sistema valutano le funzioni che il sistema deve eseguire. I requisiti funzionali possono essere descritti tramite specifiche dei requisiti di business, user story, casi d'uso o specifiche funzionali, oppure possono anche



non essere documentati. I test funzionali verificano "cosa" il sistema deve fare. I test funzionali dovrebbero essere eseguiti a tutti i livelli di test (ad es. i test di componente possono essere basati su una specifica dei componenti), sebbene l'ambito sia diverso per ogni livello.
Il testing funzionale considera il comportamento del software, quindi è possibile utilizzare tecniche black-box (vedi oltre) per derivare condizioni di test e casi di test relativi alla funzionalità del componente o del sistema.
Testing Non-Funzionale
Il test non-funzionali di un sistema valutano le caratteristiche di software e sistemi, come usabilità, efficienza prestazionale o sicurezza. Il testing non-funzionale è il testing di "quanto bene" si comporta il sistema.
Contrariamente alle comuni percezioni errate, il testing non-funzionale può, e spesso deve, essere svolto a tutti i livelli di test e il prima possibile. La scoperta tardiva di difetti non-funzionali può essere estremamente pericolosa per il successo di un progetto.
Le tecniche black-box possono essere utilizzate per ricavare condizioni di test e casi di test anche per il testing non-funzionale. Ad es. l'analisi al valore limite può essere utilizzata per definire le condizioni di stress per i test prestazionali.

Per le varie tipologie di test sono applicabili vari tecniche che hanno lo scopo di aiutare l'identificazione delle condizioni di test, dei casi di test e dei dati di test.

Tecniche di testing

Le tecniche di testing sono classificate come black-box, white-box o basate sull'esperienza.
Le tecniche di testing *black-box* (chiamate anche tecniche comportamentali o basate sul comportamento) si basano su un'analisi della base di test (ad es. documenti relativi a requisiti formali, specifiche, casi d'uso, user story o processi aziendali) e si concentrano sugli input e gli output dell'oggetto del test, senza riferimenti alla sua struttura interna.
Le tecniche di test *white-box* (chiamate anche tecniche strutturali o basate sulla struttura) si basano su un'analisi dell'architettura, della progettazione di dettaglio, della struttura interna o del codice dell'oggetto del test.
Le tecniche di test *basate sull'esperienza* sfruttano l'esperienza di sviluppatori, tester e utenti per progettare, implementare ed eseguire i test. Queste tecniche sono spesso combinate con il testing black-box e white-box.
Alcune tecniche sono meglio applicabili in determinate situazioni e a certi livelli di test, altre sono applicabili a tutti i livelli di test.
Tra le tecniche black-box abbiamo:

Copertura delle classi di equivalenza
La tecnica di Copertura delle classi di equivalenza consente di selezionare casi di test che, a fronte di una stessa condizione, sollecitano in maniera differente l'entità testata.
Per utilizzare la tecnica di Copertura delle classi di equivalenza bisogna suddividere l'insieme di tutti i possibili casi di input in "classi di equivalenza".
Dato un prodotto software e l'insieme di tutti i possibili valori di un suo dato di input, una "classe di equivalenza" è un sottoinsieme di questi valori tale che il comportamento previsto del prodotto è il medesimo per ciascun elemento del sottoinsieme.
Analisi ai Valori Limite
L'analisi ai valori limite è un'estensione del partizionamento di equivalenza, ma può essere utilizzata solo quando la partizione è ordinata, costituita da dati numerici o sequenziali. I valori minimo e massimo (o primo e ultimo valore) di una partizione sono i suoi valori limite. Questa tecnica nasce dal fatto che l'esperienza mostra quanto sia facile scrivere una istruzione di decisione e salto condizionato con la condizione limite esclusa ovvero inclusa erroneamente.
Testing della Tabella delle Decisioni
Le tecniche di test combinatorie sono utili per testare l'implementazione di requisiti di sistema che specificano come diverse combinazioni di condizioni si traducano in risultati diversi. Un approccio a tale metodo di test è il testing della tabella delle decisioni.
Testing delle Transizioni di Stato
I componenti o i sistemi possono rispondere in modo diverso a un evento a seconda delle attuali condizioni o dello stato in cui sono a seguito di precedenti elaborazioni. Un diagramma di transizione di stato mostra i possibili stati del software e le transizioni da uno stato all'altro. Il cambiamento di stato può comportare l'azione del software (ad es. l'emissione di un calcolo o di un messaggio d'errore).
I test possono essere progettati per coprire una sequenza tipica di stati, per esercitare tutti gli stati, per esercitare ogni transizione, per esercitare specifiche sequenze di transizioni o per testare transizioni non valide.
Testing dei Casi d'Uso
I test possono essere derivati da casi d'uso, che sono un modo specifico di progettare le interazioni con elementi del software, incorporando i requisiti funzionali espressi dai casi d'uso stessi. I casi d'uso sono associati ad attori (utenti umani, hardware esterno, altri componenti o sistemi) e soggetti (il componente o il sistema a cui il caso d'uso è applicato).
I casi di test progettati con le varie tecniche

sono utilizzati per sollecitare l'oggetto del test (componente, sistema) eseguendolo nelle varie condizioni di test (*test dinamico*).
Nelle fasi alte dei cicli di sviluppo quando non è disponibile il software eseguibile oppure per effettuare verifiche particolari è possibile effettuare un *test statico*.

Test statico

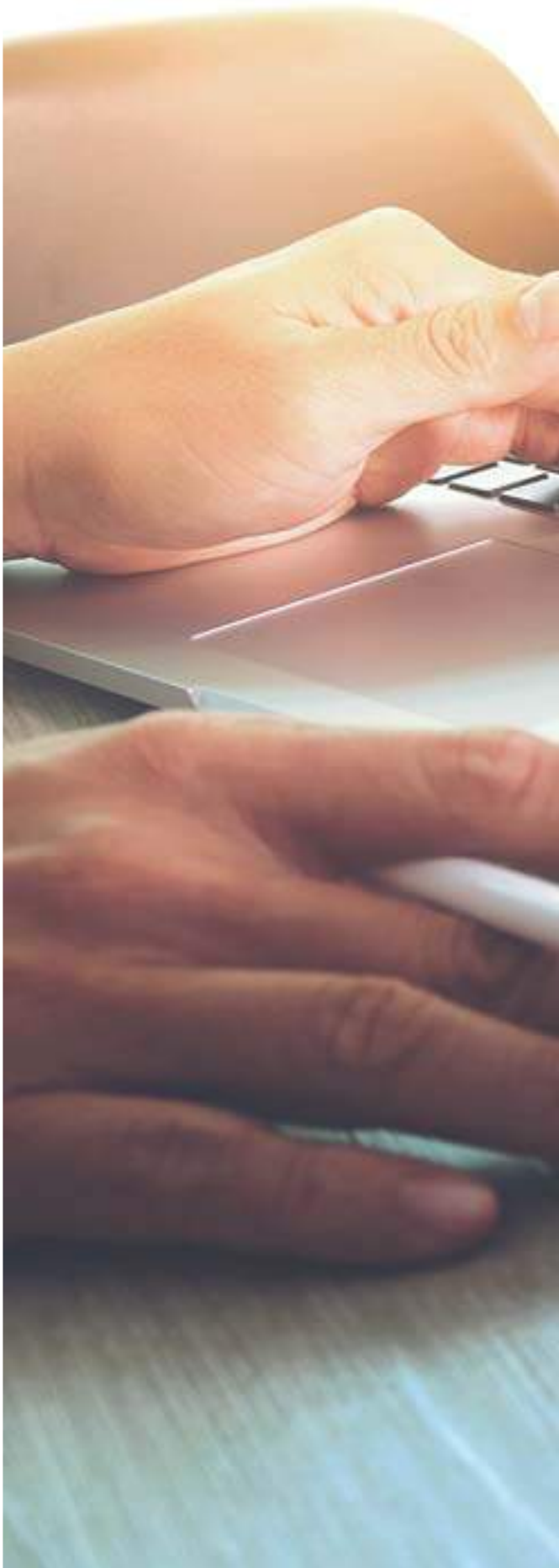
Contrariamente ai test dinamici, che richiedono l'esecuzione del software in fase di test, i test statici si effettuano tramite l'esame manuale dei prodotti di lavoro (revisioni) o sulla valutazione del codice da parte di strumenti (analisi statica). Entrambi i tipi di test statici valutano il codice o altri prodotti di lavoro da verificare senza eseguire effettivamente il codice o il prodotto di lavoro stesso.
L'analisi statica è importante per i sistemi informatici critici per la sicurezza (ad es. software avionico, medicale o nucleare), ma è diventata importante e comune anche in altre situazioni. Per es. l'analisi statica è una parte importante del testing di sicurezza.
Quasi tutti i prodotti di lavoro possono essere esaminati utilizzando test statici (revisioni e/o analisi statica), per es.:

- Specifiche come requisiti di business, requisiti funzionali e requisiti di sicurezza
- Specifiche di architettura e progettazione
- Codice
- Testware: piani di test, casi di test, procedure di test e script di test automatizzati
- Guide per l'utente
- Pagine Web.

Differenze fra Testing Statico e Dinamico

I test statici e i test dinamici possono avere gli stessi obiettivi: fornire una valutazione della qualità dei prodotti di lavoro e identificare i difetti il prima possibile. Il testing statico e il testing dinamico si completano a vicenda trovando diversi tipi di difetti.
Una distinzione principale è che i test statici individuano direttamente i difetti nei prodotti di lavoro, piuttosto che identificare le failure causate da difetti durante l'esecuzione. Un difetto può essere presente per molto tempo senza causare una failure se il percorso in cui si trova è raramente esercitato o difficile da raggiungere, quindi non sarà facile costruire ed eseguire un test dinamico che lo rilevi. I test statici possono essere in grado di trovare il difetto con molto meno sforzo.
Un'altra distinzione è che i test statici possono essere utilizzati per migliorare la coerenza e la qualità interna di prodotti di lavoro, mentre i test dinamici si concentrano in genere su comportamenti visibili esternamente.
Rispetto ai test dinamici, i difetti tipici che è più facile e meno costoso da trovare e risolvere tramite i test statici comprendono:

- Difetti nei requisiti (ad es. incongruenze,



ambiguità, contraddizioni, omissioni, inesattezze, e ridondanze)
- Difetti di progettazione (ad es. algoritmi o strutture di database inefficienti)
- Difetti di codifica (ad es. variabili con valori non definiti, variabili dichiarate ma mai utilizzate, codice non raggiungibile, codice duplicato)
- Deviazioni dagli standard (ad es. mancanza di aderenza agli standard di codifica)
- Specifiche di interfaccia errate (ad es. diversità delle unità di misura utilizzate dal sistema chiamante rispetto al sistema chiamato)
- Vulnerabilità della sicurezza (ad es. esposizione ai buffer overflow)
- Lacune o imprecisioni nella tracciabilità o copertura della base di test (ad es. test mancanti per i criteri di accettazione).

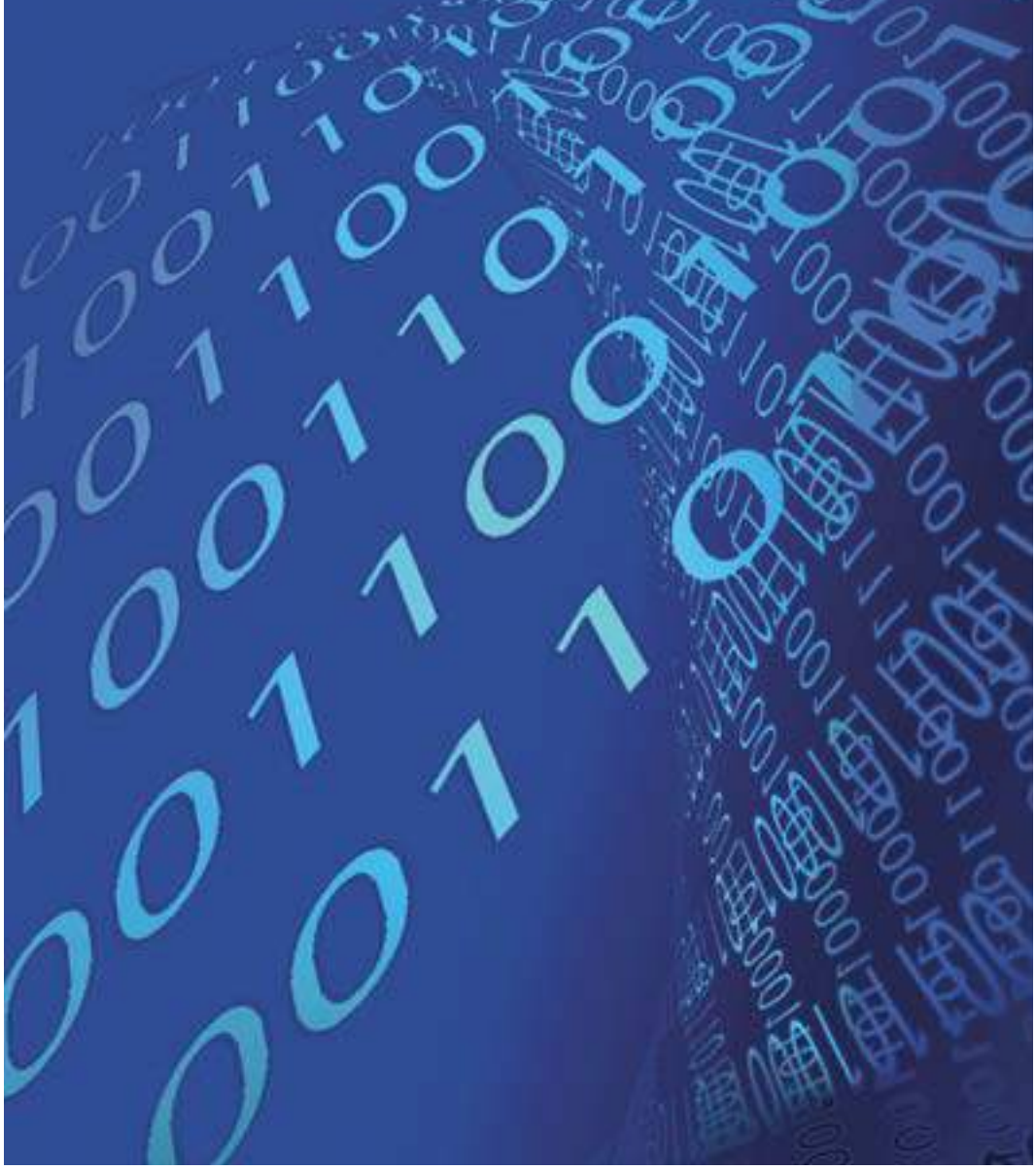
Inoltre, la maggior parte dei tipi di difetti di manutenibilità possono essere rilevati solo mediante test statici (ad es. impropria modularizzazione, scarsa riusabilità dei componenti, codice difficile da analizzare e modificare senza introdurre nuovi difetti).

5. In conclusione, considerato che il software è oggi presente in ogni nostra attività, un suo malfunzionamento può comportare costi enormi in termini economici, oltreché danni materiali e perdita di vite umane. È decisivo quindi - per ridurre il rischio di malfunzionamento - investire e dare importanza alle fasi di collaudo del software sin dalle prime fasi del ciclo di vita di sviluppo del software.

In questa prospettiva, è fondamentale sia aumentare la consapevolezza dell'importanza del collaudo software da parte di chi è responsabile dei progetti e del top management sia formare tecnici che conoscano e sappiano applicare al contesto le metodologie e le tecniche del software testing. È necessario che la formazione dei professionisti del collaudo cominci sin dalle università.

Va comunque riconosciuto che oggi già esistono diverse pubblicazioni su questi temi (cfr. bibliografia in calce): esse dimostrano che la consapevolezza in merito alla necessità di professionalità nel campo del software testing sta gradualmente aumentando.

Se vediamo i numeri di ISTQB® (International Software Testing Qualifications Board - https://www.istqb.org/) - che ha creato lo schema di certificazione di maggiore successo nel mondo per certificare le competenze nel campo del software testing - le certificazioni sono in costante aumento (vedi figura). Fino a dicembre 2017, sono stati sostenuti circa 785.000 esami di certificazione ISTQB in circa 120 paesi.



Siti Web

ISTQB® (International Software Testing Qualifications Board) - https://www.istqb.org/
ITA-STQB (Board italiano di ISTQB) - https://www.ita-stqb.org/index.php/it/

Libri

Black, R. (2009) Managing the Testing Process (3e), John Wiley & Sons: New York NY
Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY

Documenti ISTQB *(versione italiana da https://www.ita-stqb.org/index.php/it/)*

ISTQB Glossary
ISTQB Foundation Level Overview 2018
ISTQB-ATA Advanced Level Test Analyst Syllabus
ISTQB-ATM Advanced Level Test Manager Syllabus
ISTQB-AT Foundation Level Agile Tester Extension Syllabus
ISTQB-ATA Advanced Level Test Analyst Syllabus
ISTQB-ATM Advanced Level Test Manager Syllabus