

Modulo didattico di INGEGNERIA DEL SOFTWARE

“IL TESTING”

---

INTRODUZIONE

Indipendentemente dalle tecniche di sviluppo adottate, la produzione di software, come ogni altra attività umana, è un processo non immune da errori: non è infatti possibile garantire che il prodotto finale si comporti sempre come richiesto. Per questa ragione, in ogni campo applicativo, tale prodotto viene sottoposto ad opportuni collaudi affinché sia verificata la sua conformità ai requisiti e tale attività di verifica viene ripetutamente effettuata durante il progetto e lo sviluppo dell'opera.

Il testing, inteso come l'analisi e la valutazione della correttezza di una implementazione, costituisce una delle più importanti e concrete vie da percorrere per il raggiungimento della qualità del software.

Risalgono agli anni '70 i primi seri tentativi di fornire fondamenti teorici ed approcci sistematici al testing. Allora si vagheggiava la possibilità di un testing esaustivo, che dimostrasse la correttezza dei programmi e l'assoluta mancanza di errori. In seguito si verificò che il testing può rilevare la presenza ma non l'assenza di difetti (Tesi di Dijkstra) e ci si concentrò maggiormente sul miglioramento dei processi di testing e sulle nuove sfide legate ai nuovi paradigmi di programmazione, produzione, manutenzione ed evoluzione del software. Il testing si delineò comunque fin da subito come un processo estremamente complesso ed oneroso (60% del costo totale di produzione).

CONVALIDA E VERIFICA: DEFINIZIONE DELLA TERMINOLOGIA

Il problema di controllare se un software sia corretto è un problema diverso a seconda che si considerino requisiti (informali) espressi dall'utente o specifiche (rigorose/formali) prodotte dall'analista a partire dai requisiti iniziali.

**Convalida** è l'attività volta ad esaminare se sia costruito il sistema che risolve il problema applicativo che ha motivato la realizzazione; **verifica** è l'attività volta ad esaminare se il software prodotto è corretto rispetto alle specifiche dell'analista. Convalida e verifica sono attività complementari. Paradossalmente, è possibile verificare che un software sia conforme alle specifiche, senza poter stabilire se queste ultime corrispondano ai requisiti formulati dall'utente. Convalida e verifica sono attività estremamente complesse ed articolate, che molto spesso nella pratica sono sottovalutate ed affidate all'esperienza ed al buon senso del programmatore. Inoltre, l'identificazione degli errori prima di passare a fasi successive, evita che questi ultimi si propaghino e si amplifichino durante le varie fasi di sviluppo. Per questo motivo la

progettazione, la pianificazione e l'implementazione del processo di verifica deve iniziare con i primi passi di un qualsiasi processo di produzione, manutenzione ed evoluzione del software.

Per quanto concerne la terminologia adottata, secondo lo standard IEEE è definito **malfunzionamento (failure)** un comportamento del codice non conforme alle specifiche, un evento osservabile percepito dall'esterno come una mancata prestazione del servizio atteso, ad esempio un programma che deve sommare due numeri con in input i dati 4 e 3 e produce 12; con il termine di **difetto (fault)** si indica invece la causa di una failure, ossia l'elemento del programma che non corrisponde alle aspettative, nell'esempio precedente l'operatore di prodotto al posto dell'operatore di somma. E' definito invece **errore**, il fattore (umano) che causa una deviazione tra il software prodotto ed il programma ideale (uno o più errori possono produrre uno o più difetti nel codice), ad esempio errori di analisi dei requisiti, di progetto, di battitura.

## ANALISI STATICA ED ANALISI DINAMICA

Il problema di determinare la presenza di anomalie in un programma può essere affrontato in due modi diversi ma complementari. Un primo modo consiste nell'osservare il codice sorgente ed in questo caso si parlerà di **analisi statica** in quanto non si richiede l'esecuzione del programma. Un'altro metodo, detto di **analisi dinamica**, si basa su tecniche di controllo del funzionamento del programma con particolari dati di ingresso. Queste tecniche consentono di rilevare un maggior numero di malfunzionamenti rispetto alle tecniche di analisi statica, le quali però sono generalmente meno costose. Tuttavia, le tecniche di analisi dinamica presentano problemi di infinitezza: il numero di possibili esecuzioni di un programma è tale da non permettere un'analisi esaustiva. Per questo motivo è necessario selezionare un insieme finito di casi di test modulato in funzione del costo di tale attività alla luce dei requisiti del programma e dell'utente. La selezione dei casi di test può avvenire sulla base di diversi criteri. Il più semplice, anche se non il più efficace, prevede una selezione casuale dei dati. E' evidente che, in condizioni particolari, la possibilità di selezionare casualmente dati che rivelano malfunzionamenti è molto limitata. Per questo motivo i dati di test non sono quasi mai selezionati in modo completamente casuale, ma si utilizzano le conoscenze sul problema e sul programma per indirizzare la scelta. E' possibile definire due classi di criteri di selezione di test: criteri di selezione **funzionali** (detti anche a scatola nera o **Black-Box**) che derivano i casi di test dall'analisi delle specifiche dei requisiti e di progetto; e criteri di selezione **strutturali** (detti anche a scatola bianca o **White-Box**) che ricavano i casi di test dall'analisi del codice del programma.

## TEST STRUTTURALE

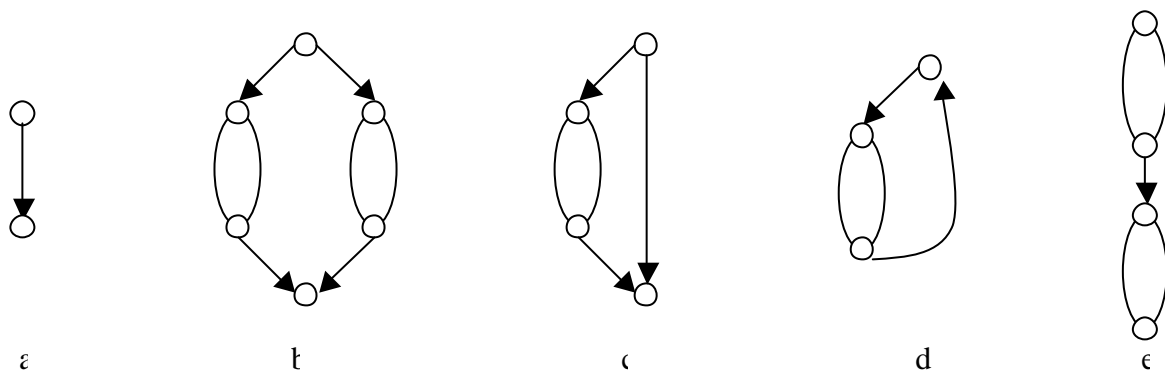
Nell'ambito del testing strutturale, esistono diversi criteri di copertura che guidano la scelta dei casi di test. Essi sono stati definiti in modo tale da imporre vincoli via via più stringenti e quindi aumentare il numero di casi di test necessari per verificare un

programma. Al crescere del grado di copertura, aumenta la qualità dell'attività di test, ma aumentano anche i costi necessari alla sua attuazione.

**Criterio di copertura dei comandi:** l'insieme dei casi di test deve essere selezionato in modo che ogni singola istruzione del codice (statement) venga eseguita almeno una volta. Tale criterio presenta alcuni limiti ad esempio nel caso di una struttura di selezione in cui manca la parte else. In questa circostanza il caso di test derivato tipicamente andrà a stimolare il programma eseguendo il ramo dell'if senza però verificare il comportamento del programma in una situazione molto importante, cioè quando la condizione dell'if è falsa. Per questo motivo si introduce un nuovo criterio di copertura.

**Criterio di copertura delle decisioni:** l'insieme dei casi di test deve essere selezionato in modo che ogni singolo ramo del grafo di controllo del programma venga attraversato almeno una volta.

Per applicare tale criterio è necessario costruire una rappresentazione grafica bidimensionale del flusso di controllo del programma da testare detta grafo di controllo.



- a) Grafo di un'istruzione di I/O, assegnamento o chiamata a procedura
- b) Grafo di un costrutto *if-then-else*
- c) Grafo di un costrutto *if-then*
- d) Grafo di un ciclo *while*
- e) Grafo di due istruzioni sequenziali

Anche questo criterio presenta degli inconvenienti. Infatti può non permettere di rilevare un errore dovuto all'uso di condizioni composite (che utilizzano più condizioni connesse da operatori logici). Per ovviare al problema è necessario introdurre un criterio più stringente.

**Criterio di copertura delle condizioni:** l'insieme dei casi di test deve essere selezionato in modo che ogni singolo ramo del grafo di controllo del programma venga attraversato almeno una volta e che tutti i possibili valori delle sottoespressioni costituenti le condizioni composte vengano valutati almeno una volta.

Per questa estensione i casi di test aumentano in modo considerevole dato che in ogni ciclo che presenti una condizione complessa si deve entrare più volte ed particolare per tale operazione devono essere generati  $2^n$  input, dove  $n$  è il numero delle condizioni atomiche che compongono quella complessa. Il limite di tale criterio è dato

dal fatto che i casi di test selezionati permettono di attraversare tutti gli archi ma non tutte le possibili combinazioni di archi.

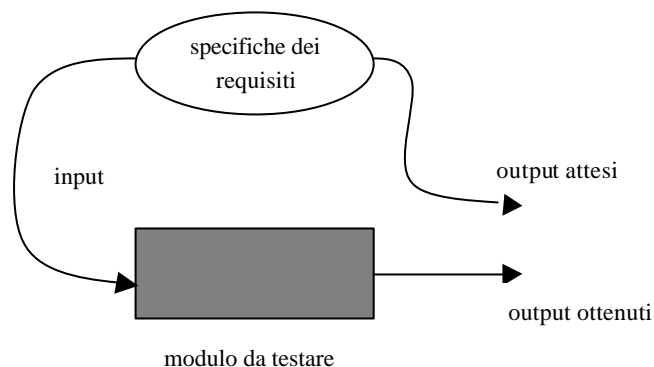
**Criterio di copertura dei cammini:** l'insieme dei casi di test deve essere selezionato in modo che ogni possibile percorso che porti dal nodo iniziale al nodo finale nel grafo di controllo del programma venga attraversato almeno una volta.

Con tale criterio vengono superate notevolmente le limitazioni rilevate nei criteri precedentemente descritti, ma si introducono un numero di casi di test spesso troppo elevato perché possa essere realmente applicabile nella pratica anche con piccole porzioni di codice (si pensi che la complessità della ricerca di tutti i possibili cammini è esponenziale).

L'osservazione precedente sottolinea ancora una volta che i vari criteri di White-Box testing qui descritti presentano un evidente trade-off tra rapidità di esecuzione (e quindi minor costo) ed affidabilità del codice una volta testato. Il criterio di copertura dei comandi presenta tra i criteri esposti una minore qualità, ma certamente a confronto con il criterio di copertura dei cammini una miglior fattibilità. In generale il White-Box testing, viene utilizzato, indipendentemente dal criterio di copertura scelto, solo per piccole porzioni di codice ed in particolari per quelle parti del codice che vengono considerate critiche per il corretto funzionamento del sistema.

## TEST FUNZIONALE

Il testing funzionale è basato sulle specifiche che descrivono il comportamento del programma (o di una sua porzione) anziché analizzarne la struttura.



Come esemplificato in figura, il programma viene considerato come una scatola nera di cui sono visibili i risultati per particolari dati d'ingresso ma di cui si ignora la struttura e qualsiasi informazione sul codice. I casi di test vengono identificati studiando la specifiche del programma. Questo fatto comporta che la pianificazione dei test può iniziare abbastanza presto nel processo di sviluppo del software. Se le specifiche sono espresse in un linguaggio formale è possibile definire criteri di test rigorosi supportati da strumenti automatici. Esistono infatti linguaggi di specifica (per esempio TRIO) per i quali si è in grado di generare in modo (semi)automatico casi di test ovvero si derivano i casi di test dalla specifica degli scenari d'uso (si pensi agli

Use Case Diagram dell'UML). Al contrario, se le specifiche sono espresse in un linguaggio informale è necessario l'intervento umano a cui sono demandate scelte che possono compromettere il risultato finale. Infatti, le specifiche vengono scomposte in un insieme di casi rilevanti per ciascuno dei quali vengono creati dei dati di test ed in quest'attività di individuazione delle funzioni da scomporre è di fondamentale rilevanza l'abilità e l'esperienza delle persone a cui è affidata questa fase. Al fine di selezionare dati di test critici per il sistema si costruisce spesso un grafo definito causa-effetto, in cui le specifiche iniziali, ridotte a condizioni booleane, vengono messe in relazione fra loro per indicare come fatti elementari corrispondenti a dati di ingresso possano essere causa di fatti elementari corrispondenti a dati di uscita. Altre volte si utilizzano criteri di selezione basati su specifiche algebriche.

### TEST "IN GRANDE"

Sono tecniche di test che consentono di verificare il sistema nel suo complesso. Finora abbiamo considerato il testing di singole unità (moduli) di un sistema software (**test di unità**).

Il **test di integrazione** viene applicato ad un aggregato di due o più unità di un sistema software con l'obiettivo di rilevare gli errori nella integrazione fra le unità e nelle funzioni che l'aggregato deve assolvere. Le unità da integrare vengono ricavate in base a criteri funzionali derivabili dall'architettura del sistema. Sono possibili due approcci all'integrazione: top-down e bottom-up. Supponendo di avere un modulo A che richiama i moduli B, C e D, se volessimo testare il modulo A prima ancora di aver realizzato B, C e D (approccio top-down), dovremmo utilizzare dei moduli in grado di simulare il comportamento di B, C e D. Tali moduli sono detti fittizi (stub). Uno stub ha la stessa interfaccia del modulo simulato, ma è più semplice: può ad esempio restituire solo valori costanti ovvero risultati scelti dal testatore. Nel caso di approccio bottom-up, occorre invece simulare l'ambiente chiamante. A tale scopo occorre realizzare moduli guida (drivers). Si parla di test di integrazione big bang quando tutti i moduli (precedentemente sottoposti a test di unità) sono integrati in un sol colpo, si parla invece di test di integrazione di tipo incrementale quando i moduli sono integrati via via che vengono prodotti ed esercitati singolarmente.

Il **test di sistema** è applicato sul sistema software completo ed integrato, con l'obiettivo di valutare l'adesione del sistema ai requisiti specificati che non consistono soltanto nelle funzionalità esterne ma anche in requisiti di qualità e di prestazioni, stabiliti sulla base di un modello di qualità del prodotto opportunamente istanziato. Le qualità che vengono analizzate in questa fase sono lo **stress**, ossia l'affidabilità in condizione di carico limite; la **sicurezza**, cioè l'invulnerabilità del sistema rispetto ad accessi non autorizzati; la **robustezza**, intesa come la capacità del sistema a cadute e la **configurazione**, ovvero un funzionamento corretto per tutte le configurazioni richieste.

Il **test di accettazione** viene effettuato sull'intero sistema sulla base di procedure approvate dal cliente affinché quest'ultimo possa decidere se accettare il prodotto.

Viene normalmente effettuato prima un test interno, detto  $\alpha$ -test da persone diverse dagli sviluppatori del sistema, ma ancora interne alla ditta produttrice del software. Una volta terminata la fase di  $\alpha$ -test viene rilasciata una versione non definitiva del prodotto che viene proposta per una fase di  $\beta$ -test (**test esterno**) da parte di veri utenti (detti anche **utenti pilota**) in un ambiente reale.

Un'ultima forma di test che è utile in questa sede ricordare è il cosiddetto **test di regressione**, che ha lo scopo di verificare compatibilità e differenze di una nuova versione di un programma rispetto alla precedente. Esso consiste nell'eseguire la nuova versione con gli stessi dati di test della vecchia confrontando i risultati con quelli precedentemente registrati.

Dispensa a cura di  
Barilaro Rosamaria