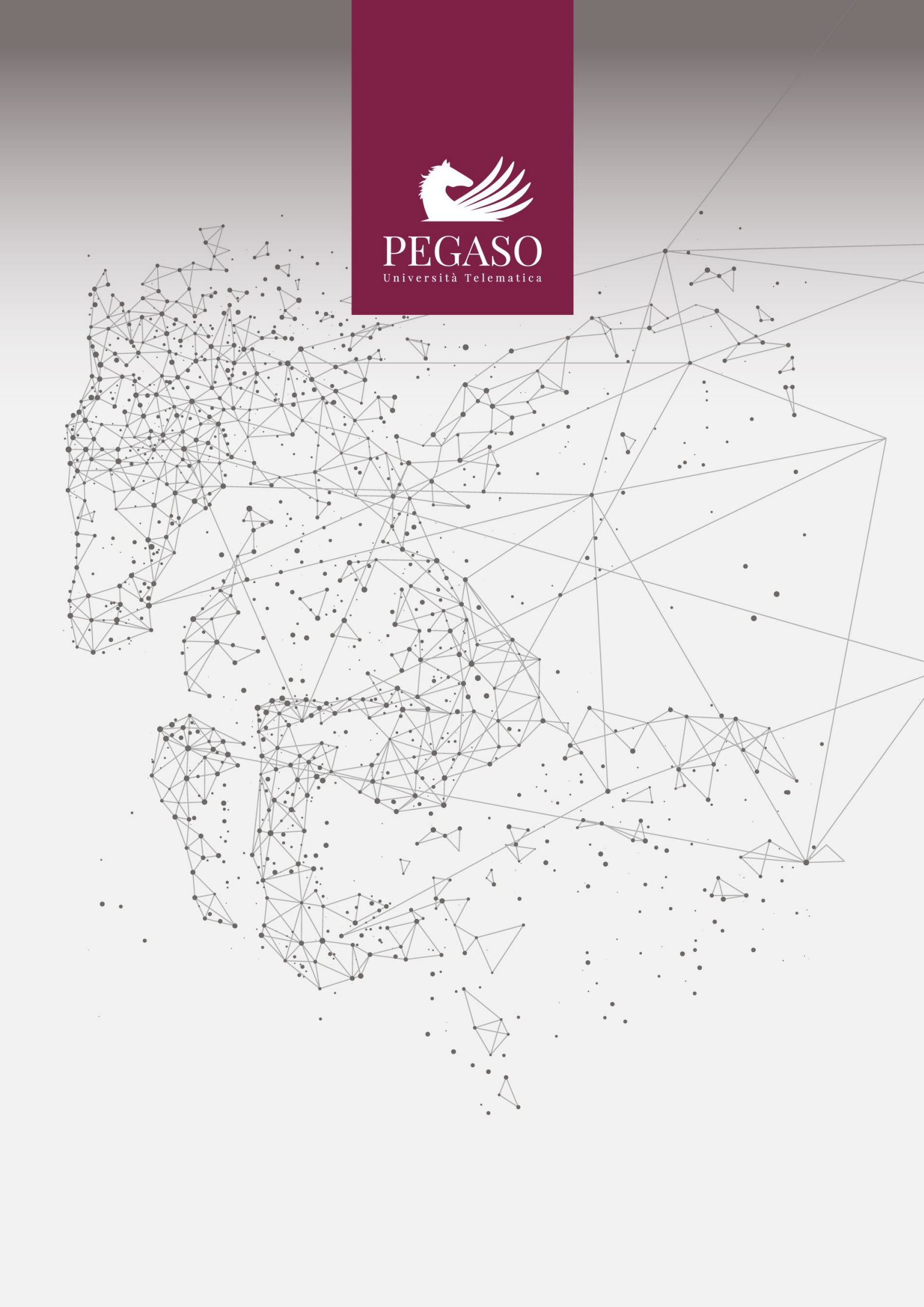
PEGASO
Università Telematica

Indice

1. Premessa

Nel contesto dell'ingegneria del software, **la selezione dei casi di test** rappresenta un'attività cruciale, poiché consente di garantire che il sistema in fase di sviluppo sia sottoposto a un processo di verifica sistematico e rappresentativo. La crescente complessità dei sistemi software moderni, unitamente all'aumento esponenziale delle combinazioni di input, stati e percorsi esecutivi, ha reso evidente che testare in modo **esaustivo ogni comportamento possibile** è, nella pratica, un'impresa irrealizzabile. Tale constatazione ha spinto la comunità scientifica e industriale a sviluppare e raffinare approcci metodologici che permettano di selezionare, in modo strutturato e razionale, un sottoinsieme di casi di test che sia sufficientemente significativo dal punto di vista della copertura e della rilevazione dei difetti.

La scelta accurata dei test da eseguire non solo incide direttamente sull'efficacia del processo di verifica, ma ha anche importanti ricadute in termini di costi, tempi di rilascio e affidabilità del prodotto finale. È fondamentale che i casi di test selezionati siano **rappresentativi** del comportamento atteso del sistema, in modo da massimizzare le probabilità di individuare difetti rilevanti e al contempo minimizzare la quantità di test eseguiti inutilmente. Questo obiettivo si raggiunge mediante l'applicazione di **tecniche sistematiche di selezione dei test**, basate su criteri logici, funzionali e strutturali.

In questa lezione analizzeremo le principali metodologie di selezione, concentrandoci su tecniche come le **classi di equivalenza**, la **boundary value analysis**, e il **category partition method**. Ognuna di queste strategie verrà presentata con una descrizione dettagliata, esempi pratici ed evidenziazione dei rispettivi punti di forza e limiti. Particolare attenzione verrà riservata alla loro applicabilità nei diversi contesti di sviluppo, alla possibilità di automazione e alla coerenza con i requisiti del sistema. Il nostro obiettivo è fornire una panoramica teorica e pratica utile a progettare casi di test in maniera **sistematica, efficiente e mirata alla copertura dei difetti** più significativi, ottimizzando così il rapporto tra il costo sostenuto e i benefici ottenuti in termini di qualità del software.

2. Introduzione

Il **testing** costituisce una delle attività fondamentali nello sviluppo del software. Non si tratta di un'attività secondaria o opzionale, bensì di una componente essenziale del ciclo di vita del software, il cui scopo è garantire che il prodotto realizzato soddisfi non solo i requisiti funzionali ma anche le aspettative di qualità, affidabilità e robustezza. La sua funzione primaria è verificare che il sistema rispetti le aspettative sia in termini di **correttezza del comportamento** che di **conformità ai requisiti**. In altre parole, un software che superi con successo i test è ritenuto idoneo a operare nei contesti reali per cui è stato progettato.

Testare un sistema non significa soltanto scovare difetti, ma anche aumentare la fiducia nella qualità del prodotto rilasciato. Infatti, il testing serve anche a fornire evidenze oggettive del corretto funzionamento del sistema, a supporto delle decisioni di rilascio e di accettazione. Esso contribuisce a consolidare la fiducia degli stakeholder, compresi i clienti finali, nella bontà e affidabilità del software.

Tra gli **obiettivi principali del testing** si annoverano la verifica dell'output rispetto agli input previsti, la validazione dei requisiti utente, l'individuazione di errori e comportamenti anomali, e la valutazione complessiva della qualità del sistema. A questi si aggiungono l'identificazione di colli di bottiglia prestazionali, l'analisi dell'usabilità, la verifica della compatibilità con sistemi esterni e l'osservazione del comportamento in condizioni di carico estremo.

Una delle idee apparentemente più intuitive, ma nella pratica irrealizzabili, è quella del **test esaustivo**, ovvero la verifica di tutte le possibili combinazioni di input e stati. Tale approccio fallisce di fronte alla **combinatorietà esplosiva** che caratterizza i sistemi reali. Per esempio, una funzione con tre parametri, ciascuno dei quali assume dieci valori differenti, implica 1.000 combinazioni da testare; aggiungendo un solo parametro, tale numero cresce a 10.000, rendendo rapidamente insostenibile ogni tentativo di verifica completa. In ambito industriale, l'adozione di strategie di testing esaustivo è praticamente assente, se non per porzioni molto limitate e critiche del sistema. L'alternativa più realistica consiste nel focalizzarsi su quelle aree del codice e del comportamento che presentano una maggiore probabilità di contenere difetti o che risultano essere più sensibili alle condizioni operative.

Di conseguenza, emerge la necessità di ricorrere a **tecniche di selezione dei test** che, guidate da criteri sistematici, permettano di concentrare lo sforzo nei punti più critici del sistema, ottimizzando l'uso delle risorse disponibili e garantendo al tempo stesso una **copertura efficace del comportamento atteso**. Tali tecniche non sono solo strumenti operativi, ma veri e propri modelli mentali e metodologie di supporto alla progettazione del processo di verifica. Essi consentono di trasformare un problema apparentemente

intrattabile — quello del testing completo — in un insieme gestibile e strutturato di attività, rendendo così possibile un approccio rigoroso, documentabile e ripetibile al controllo della qualità del software.

3. Concetti Chiave

Due approcci opposti ma complementari al testing sono il **random testing** e il **test sistematico**. Il primo si basa sulla generazione automatica di input casuali, tipicamente selezionati da un dominio di input definito ma senza una struttura logica predeterminata. Questo metodo è spesso impiegato per esplorare comportamenti inaspettati del sistema, specialmente in prossimità dei cosiddetti **border-case**, ovvero quegli input al margine tra condizioni lecite e illecite. Il **random testing** ha il vantaggio di essere estremamente semplice da implementare, soprattutto in ambienti che supportano l'automazione su larga scala. Può essere particolarmente efficace nel rivelare crash, malfunzionamenti sistemici o comportamenti erratici, risultando utile nei contesti di **fuzz testing** o nei sistemi che gestiscono input da fonti esterne non controllate. Tuttavia, questa tecnica soffre di importanti limitazioni, come la **bassa copertura mirata**, poiché gli input generati non sono progettati per coprire condizioni specifiche, e la **scarsa tracciabilità dei difetti**, rendendo difficile correlare un errore osservato a una determinata condizione funzionale del sistema.

Al contrario, il **test sistematico** si fonda su un processo rigoroso e strutturato di progettazione dei test, basato su informazioni disponibili sui requisiti funzionali (approccio **black-box**) o sulla struttura del codice sorgente (approccio **white-box**). Questo tipo di testing mira a garantire che ogni test abbia uno scopo ben definito e documentato, con l'obiettivo di ottenere una **copertura elevata** delle condizioni critiche del sistema. Tecniche come le **classi di equivalenza**, la **boundary value analysis**, la **decision table testing**, i **path coverage** e molte altre fanno parte di questo insieme metodologico. Il vantaggio principale del test sistematico risiede nella sua **ripetibilità e tracciabilità**, qualità che lo rendono estremamente utile per ambienti regolamentati o per il mantenimento della qualità in progetti di lunga durata. Esso consente anche una più agevole individuazione e diagnosi dei difetti, grazie al collegamento diretto tra ogni caso di test e il requisito o la struttura che intende verificare.

Nella pratica industriale, è raro affidarsi esclusivamente a uno di questi due approcci. Piuttosto, si adottano **strategie combinate**: test sistematici per coprire condizioni note e prevedibili, test casuali per esplorare comportamenti emergenti, e test esplorativi per intercettare anomalie che sfuggono ai modelli formali. Questa combinazione consente di bilanciare **efficacia, copertura e costo** del processo di verifica.

Un'ulteriore distinzione fondamentale è quella tra **black-box testing** e **white-box testing**. Il primo si focalizza sul comportamento esterno del sistema, ignorando completamente la sua implementazione interna. È particolarmente adatto nei contesti in cui il tester non ha accesso al codice sorgente, come nei test di accettazione o quando si lavora con componenti di terze parti. Il secondo, invece, parte dalla

conoscenza dettagliata del codice per garantire che ogni ramo, condizione e percorso logico sia testato almeno una volta. Entrambi gli approcci hanno vantaggi e limiti: il black-box garantisce un punto di vista utente-centrico e indipendente dall'implementazione, mentre il white-box offre una visibilità interna che consente di individuare errori nascosti nei meandri della logica di programmazione.

La scelta tra queste tecniche dipende da numerosi fattori, tra cui la **complessità del sistema**, la **criticità delle funzionalità testate**, il **tempo e le risorse disponibili** per l'esecuzione dei test e la **probabilità di difetti** legata a ciascuna area. Un approccio ben bilanciato e informato consente di ottimizzare l'efficacia complessiva del processo di testing, trasformandolo da un'attività onerosa a uno strumento strategico per la qualità del software.

4. Equivalence Class Testing

L'**Equivalence Class Testing (ECT)** è una tecnica sistematica di progettazione dei test che si basa sull'idea che è possibile suddividere il dominio di input di un sistema in gruppi omogenei, detti **classi di equivalenza**, all'interno dei quali ogni valore produce lo stesso comportamento osservabile. In altre parole, valori che rientrano nella stessa classe di equivalenza sono considerati intercambiabili ai fini del testing, poiché il sistema dovrebbe reagire a essi in modo analogo. Questa ipotesi consente di ridurre significativamente il numero di casi di test, mantenendo comunque un'elevata probabilità di identificare errori rilevanti.

Lo scopo dell'ECT è dunque quello di **ottimizzare la copertura funzionale** mediante una selezione razionale e rappresentativa dei dati di input. Non si tratta semplicemente di risparmiare tempo o risorse, ma di adottare un criterio formale e sistematico che guidi il processo di selezione, rendendolo ripetibile, documentabile e giustificabile anche in ambiti regolamentati. Anziché testare ogni singolo valore di input, si seleziona un rappresentante per ciascuna classe di equivalenza. Questo approccio permette di ottenere una copertura significativa del comportamento funzionale del sistema con un numero contenuto di test, riducendo la ridondanza e focalizzando l'attenzione sui comportamenti differenzianti del sistema.

Le **classi di equivalenza** si distinguono in **valide** e **non valide**. Le prime comprendono valori che il sistema dovrebbe accettare e gestire correttamente, mentre le seconde includono input errati, fuori specifica o malformati, che il sistema dovrebbe essere in grado di riconoscere e trattare in modo sicuro (es. generando messaggi di errore o rigettando l'input). Una progettazione accurata dei test implica quindi l'identificazione di entrambe le tipologie di classi, al fine di coprire sia i comportamenti funzionali attesi sia le reazioni del sistema a condizioni anomale. Ad esempio, per un campo di input che accetta numeri interi tra 1 e 100, le classi valide includono [1–100], mentre quelle non valide includono valori come -5, 0, 101, o stringhe alfanumeriche. In contesti più complessi, le classi possono essere derivate da specifiche funzionali, regole di business, condizioni contrattuali e altre fonti di requisiti.

Una variante dell'ECT è il **Weak Equivalence Class Testing (WECT)**, che si limita a testare un solo valore per ciascuna classe, ignorando le interazioni tra più parametri. Questo approccio è utile quando si ha a che fare con input indipendenti e consente una progettazione semplice e veloce, ideale per test preliminari, per la verifica di moduli isolati o per scenari a bassa criticità. Tuttavia, la sua semplicità è anche un limite, poiché non consente di scoprire difetti derivanti da combinazioni di input che, pur individualmente corretti, producono effetti indesiderati se valutati congiuntamente.

Al contrario, il **Strong Equivalence Class Testing (SECT)** combina classi appartenenti a più parametri per esplorare le **interazioni tra diversi input**. Questo metodo permette di ottenere una copertura funzionale molto più ricca e di rilevare difetti legati a combinazioni specifiche di valori, spesso non intercettabili con il WECT. Per esempio, in un modulo che richiede la combinazione di età compresa tra 18 e 65 e reddito mensile superiore a 1.000 euro, il SECT prevede test che verificano tutti gli incroci tra le classi valide e non valide di ciascun parametro. Sebbene più costoso in termini di tempo e risorse, questo approccio è essenziale nei sistemi in cui le **dipendenze tra input** rappresentano una fonte rilevante di rischio.

Una delle estensioni più significative del concetto di classi di equivalenza è la **Boundary Value Analysis (BVA)**, che si fonda sull'osservazione empirica che molti difetti si manifestano proprio nei valori ai **margini delle classi di input**. La BVA prevede la selezione di test che includano i valori estremi dei range validi e quelli immediatamente al di fuori. Questo tipo di analisi è particolarmente efficace per rilevare errori nei confronti logici, condizioni di overflow, e vulnerabilità legate a limiti mal gestiti. È una tecnica molto utilizzata nei sistemi numerici, nei form di input, nei campi con restrizioni sintattiche e semantiche.

Quando si hanno **più parametri di input**, la BVA può essere applicata in modo indipendente su ciascun parametro, mantenendo gli altri a valori centrali validi, oppure in combinazione, sebbene ciò comporti un aumento esponenziale del numero di casi da considerare. In ogni caso, l'uso congiunto di ECT e BVA fornisce una base solida per la costruzione di suite di test robuste, modulabili e orientate al rischio.

In sintesi, l'Equivalence Class Testing rappresenta una strategia essenziale per ottenere una **copertura funzionale sistematica e razionale**, riducendo la ridondanza dei test e migliorando l'efficienza complessiva del processo di verifica. L'uso combinato di WECT, SECT e BVA consente di adattare il livello di approfondimento dei test alle esigenze specifiche del sistema, migliorando la rilevazione precoce dei difetti e la qualità finale del prodotto. Questo approccio, se ben implementato, costituisce uno strumento potente per garantire la solidità del software e ridurre i costi legati ai difetti post-rilascio.

5. Category Partition Method

Il **Category Partition Method (CPM)** è una tecnica avanzata per la progettazione sistematica dei casi di test, particolarmente efficace nel contesto del **black-box testing**. Questa metodologia si propone di organizzare in modo strutturato lo spazio degli input e delle condizioni operative, al fine di identificare tutte le **variazioni significative** che un sistema potrebbe incontrare. A differenza di tecniche più semplici, come l'Equivalence Class Testing, il CPM prevede una fase iniziale di modellazione dettagliata delle **categorie di input**, che vengono poi suddivise in **partizioni** rappresentative. Ogni combinazione tra le partizioni delle diverse categorie costituisce un potenziale caso di test, aumentando in modo significativo la probabilità di rilevare difetti critici legati a condizioni complesse e interdipendenti.

L'approccio del CPM si articola in quattro fasi principali. La prima fase è l'**analisi dei requisiti**, durante la quale si identifica ciò che deve essere testato: una funzione, un modulo o un'interazione specifica. Si individuano anche i parametri di ingresso e le condizioni operative che influenzano il comportamento del sistema. La seconda fase consiste nella **costruzione dello schema di specifica**, in cui ogni parametro viene scomposto in categorie significative (ad esempio, formato dell'input, validità semantica, lunghezza, presenza di caratteri speciali, vincoli di dipendenza, configurazioni ambientali, ecc.). Ogni categoria viene poi suddivisa in **partizioni**, ovvero insiemi rappresentativi dei possibili valori, classificati ad esempio in valori validi, invalidi, borderline, opzionali, omessi, ecc. L'obiettivo è rappresentare in modo esaustivo ma gestibile l'intero spazio delle condizioni operative possibili.

Nella terza fase si procede con l'**applicazione di vincoli**, un passaggio cruciale per evitare la generazione di casi di test inutili o irrealizzabili. Alcune combinazioni di partizioni possono infatti risultare incompatibili, ridondanti o prive di significato nel contesto del sistema testato. Si introducono quindi **regole di esclusione o di priorità**, che possono derivare da vincoli di progettazione, da requisiti funzionali o da logiche di business. Queste regole non solo riducono il numero di test, ma aumentano la rilevanza dei casi selezionati, concentrando l'attenzione su combinazioni realistiche e potenzialmente problematiche.

Infine, nella quarta fase, si effettua la vera e propria **generazione dei casi di test**, che può essere **esaustiva** (combinando tutte le partizioni possibili) oppure **selettiva**, ad esempio tramite strategie **pairwise**, **3-wise** o **combinazioni basate sul rischio**. L'approccio selettivo consente di ridurre drasticamente il numero di test mantenendo una copertura adeguata delle interazioni tra le condizioni. Alcuni strumenti permettono anche l'automazione di questa fase, migliorando l'efficienza del processo e facilitando l'integrazione del CPM nei pipeline di continuous testing.

Un esempio pratico può chiarire meglio l'approccio. Supponiamo di voler testare un modulo di login con due campi: **username** e **password**. Le categorie identificate potrebbero includere: lunghezza dell'input, tipo di caratteri, presenza di simboli, rispetto di determinate regole (es. almeno un numero nella password). Le partizioni per la categoria "password" potrebbero essere: password valida, password troppo corta, password priva di numeri, password vuota, password con caratteri speciali. Combinando queste partizioni con quelle della categoria "username" (es. valido, vuoto, con simboli, troppo corto), si generano casi di test che coprono scenari realistici, comprese le condizioni anomale e le violazioni dei vincoli di formato.

Il principale punto di forza del CPM risiede nella sua **strutturazione esplicita** del processo di test. Ogni decisione presa durante la progettazione può essere documentata, tracciata e giustificata, favorendo così la revisione del lavoro, il riutilizzo delle specifiche e l'automazione della generazione dei test. Questa tracciabilità si traduce in una maggiore trasparenza nei processi di verifica e validazione, facilitando la certificazione di sistemi critici e la dimostrazione della copertura dei requisiti. Esistono strumenti software (come TSL, FAST, ACTS) che supportano il CPM e permettono di generare automaticamente i casi a partire da modelli formali, riducendo l'errore umano e aumentando la scalabilità del metodo.

Tuttavia, il Category Partition Method presenta anche alcune **criticità**. Innanzitutto, la **modellazione iniziale** richiede tempo, attenzione e competenze analitiche, in quanto la qualità dei test dipende direttamente dalla completezza e correttezza delle categorie e delle partizioni definite. Inoltre, con l'aumentare delle categorie e delle partizioni, può verificarsi un'**esplosione combinatoria** che rende necessario l'uso di tecniche di riduzione (come il pairwise testing), **filtri logici**, **pesature di rischio** o l'applicazione di algoritmi di generazione automatica. Infine, il metodo è fortemente dipendente dalla **qualità delle specifiche**: requisiti vaghi o ambigui compromettono l'efficacia dell'intera modellazione e rischiano di produrre test fuorvianti o non significativi.

Nonostante queste difficoltà, il CPM è estremamente **potente e flessibile**, adatto a sistemi complessi, configurazioni variabili, modalità operative diversificate, test di interfaccia, test di compatibilità e validazione di configurazioni software-hardware. L'adozione di questo metodo consente di affrontare il testing in modo **disciplinato, documentato e tracciabile**, elevando così il livello di maturità del processo di qualità del software e supportando l'introduzione di pratiche di testing avanzate in contesti aziendali di media e alta complessità.

6. Conclusioni e sintesi

In conclusione, le tecniche di selezione dei casi di test rappresentano strumenti fondamentali nel processo di verifica e validazione del software. Esse permettono di superare i limiti del test esaustivo e di affrontare in modo razionale la complessità intrinseca dei sistemi software moderni. L'evoluzione continua delle architetture software, la proliferazione delle piattaforme e l'aumento delle dipendenze tra moduli rendono impossibile un approccio naive alla verifica. È quindi necessario affidarsi a metodi formali, strutturati e scalabili, che garantiscano copertura, tracciabilità e ottimizzazione delle risorse impiegate nel processo di testing.

Attraverso l'applicazione di approcci sistematici come l'**Equivalence Class Testing**, la **Boundary Value Analysis** e il **Category Partition Method**, è possibile progettare suite di test che siano al contempo efficaci, efficienti e orientate al rischio. Ognuna di queste tecniche consente di focalizzarsi su aspetti specifici del comportamento del sistema: la rappresentatività delle classi di input, la sensibilità ai valori limite e la copertura combinatoria delle condizioni operative. Il risultato è un processo di testing più robusto, capace di anticipare i difetti prima che questi emergano in fase di produzione, riducendo così i costi post-rilascio e migliorando l'esperienza utente.

Uno dei principali risultati concettuali evidenziati è che **non è necessario testare ogni singolo comportamento del sistema per ottenere una verifica significativa**. Al contrario, selezionando un sottoinsieme ben scelto di casi rappresentativi, si può raggiungere una copertura logica o funzionale sufficiente per individuare i difetti più critici. L'ECT consente di raggruppare gli input in classi omogenee, riducendo la ridondanza. La BVA si concentra sui valori limite, che statisticamente sono i più soggetti a errori. Il CPM fornisce una struttura modulare per generare combinazioni significative a partire dai requisiti, con un'attenzione particolare alla tracciabilità e alla formalizzazione del processo.

Le tecniche analizzate, sebbene diverse tra loro per complessità e applicabilità, condividono un obiettivo comune: **aumentare l'efficacia del processo di test riducendo i costi e i tempi necessari per il suo svolgimento**. Questo obiettivo è cruciale in un'epoca in cui lo sviluppo agile e il deployment continuo richiedono cicli di verifica rapidi, precisi e possibilmente automatizzati. In contesti reali, queste metodologie non vengono utilizzate in modo isolato, bensì integrate in strategie combinate che includono test automatizzati, test esplorativi e strumenti di supporto alla progettazione. L'integrazione con strumenti software moderni permette inoltre di sfruttare al massimo le potenzialità dell'automazione, dell'analisi statica e della gestione del rischio, dando vita a pipeline di testing continue e intelligenti.

Le tecniche sistematiche, se correttamente adottate, favoriscono una maggiore maturità del processo di qualità, una migliore documentazione delle attività di verifica, e una maggiore affidabilità del prodotto software. Esse sono anche indispensabili nei contesti dove la **conformità normativa** o la **certificazione formale** richiedono evidenze chiare e tracciabili delle attività di test svolte. In ambiti quali la sanità digitale, l'aerospazio, l'automotive e i sistemi critici per l'infrastruttura pubblica, il rigore metodologico del testing è un requisito imprescindibile, e le tecniche di selezione giocano un ruolo determinante per garantire la sicurezza e la resilienza del sistema.

In sintesi, le tecniche di selezione dei casi di test rappresentano una risposta concreta al paradosso del testing nel mondo reale: **non possiamo testare tutto, ma possiamo testare meglio**. L'adozione di approcci metodologici basati su criteri razionali trasforma l'apparente impossibilità del test esaustivo in una strategia orientata al valore, che consente di concentrare gli sforzi dove realmente serve, aumentando la fiducia nella qualità e nella robustezza dei sistemi sviluppati

Bibliografia

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