UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

Corso di Laurea in Ingegneria Informatica

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Tesi di Laurea in

INGEGNERIA DEL SOFTWARE

Esperimenti a supporto dell'analisi del modello cognitivo di studenti durante la progettazione di casi di test

Candidato
GIUSEPPE GATTA
MATR: N46005093

ANNO ACCADEMICO 2022/2023

Abstract

La presente tesi si propone di condurre una approfondita analisi delle dinamiche cognitive coinvolte nella progettazione di casi di test da parte degli studenti. In un contesto sempre più complesso e dinamico, la capacità di progettare casi di test efficaci rappresenta una competenza critica per garantire la qualità e l'affidabilità del software.

La ricerca coinvolgerà una varietà di studenti Magistrali, ad oggi già Ingegneri Informatici triennali Federiciani per valutare come questi reagiscono al “Testing innato”, coprendo inevitabilmente differenti livelli di competenza e background, al fine di ottenere una rappresentazione completa delle abilità cognitive coinvolte nella progettazione di casi di test.

Gli esperimenti si concentreranno sulla valutazione della comprensione delle specifiche del software, l'identificazione di scenari critici da testare, le scelte di progettazione e le strategie risolutive adottate dagli studenti. I dati raccolti saranno analizzati, al fine di identificare tendenze comuni, sfide ricorrenti e opportunità di miglioramento.

Questo esperimento è attualmente in corso in Italia, Belgio, Portogallo e Spagna, coinvolgendo studenti di ingegneria ed anche di economia.

Sebbene questa parte dell’analisi vada oltre gli scopi di questa tesi, il progetto prevede di differenziare le risposte tra Ingegneri e coloro che si trovano al di fuori della "zona di pertinenza". Sarà altresì esaminato come le variazioni in termini di università, carriere e Paesi possano influenzare il processo di progettazione dei casi di test.

Le considerazioni emerse da questo studio potranno contribuire a:

- Ridefinire l'insegnamento nei corsi di studi universitari, tenendo conto delle differenze individuate.
- Analizzare come gli Ingegneri, con studi triennali, si avvicinano al testing e come affrontano il mondo del lavoro.
- Fornire un elenco di conoscenze che potrebbero migliorare le pratiche di esecuzione dei test da parte degli sviluppatori.
- Proporre implementazioni e indagini future sull'argomento, aprendo la strada a ulteriori sviluppi nel campo del testing software.

Indice

INTRODUZIONE

Il software non funziona mai come si vorrebbe: creare software significa specificare formalmente (usando un linguaggio di programmazione) una soluzione ad un problema.

Nonostante si definiscano i vari requisiti e la documentazione di progettazione, molto spesso si implementa questa soluzione in modo errato o, anche se si implementa la soluzione correttamente, la soluzione stessa potrebbe non essere ciò che il cliente esattamente desidera o potrebbe avere conseguenze impreviste nel comportamento del software. In tutte queste situazioni il software potrebbe produrre un comportamento imprevisto e non voluto.
L'errore che provoca questo comportamento inaspettato e non intenzionale è chiamato bug del software.
Il testing è il processo pianificato di esecuzione del software con l'intento di scoprire errori nel software: è importante rendersi conto che è un processo mirato e non è la scoperta accidentale di bug del software.
Il testing è il processo più utilizzato per garantire la qualità dei sistemi software.
Non c'è da stupirsi che aziende come Google, Microsoft e Facebook investano da tempo nei test del software e si assicurino che i loro sviluppatori padroneggino diverse tecniche.

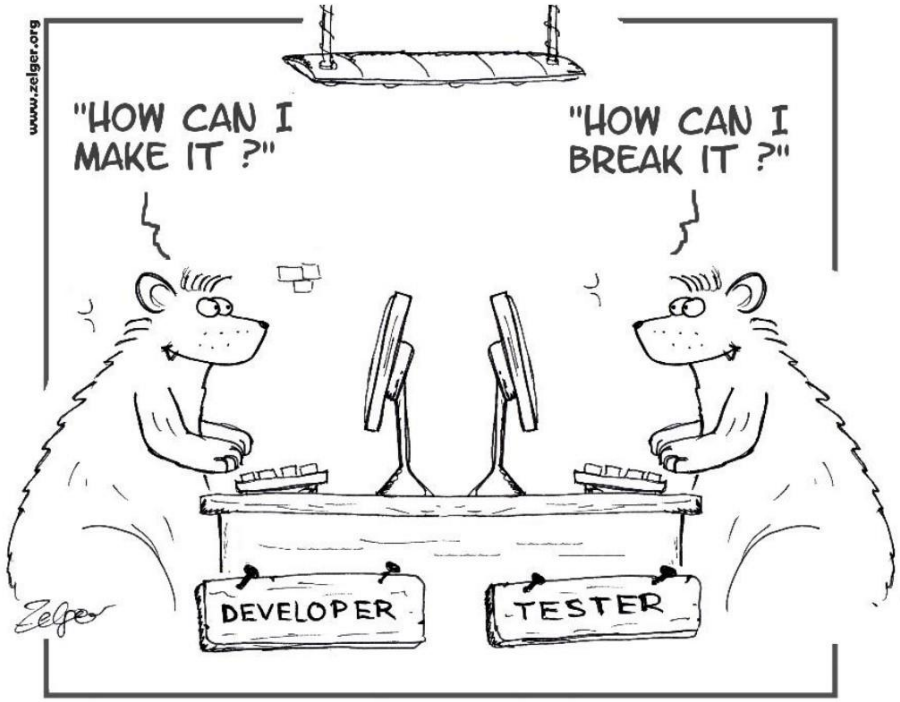
Scoprire gli errori nell'implementazione stessa del software è un aspetto importante del test del software, ma non è l'unico aspetto. I test possono essere utilizzati anche per scoprire quanto bene il software è conforme ai requisiti, inclusi quelli di prestazioni, di usabilità e altri.

Possono essere coinvolte molte persone diverse nello sviluppo di un particolare software nei suoi test, come gli sviluppatori, un team di tester indipendenti e persino i clienti stessi.

CAPITOLO 1

1.1 Perché testiamo il software

Mentre l'industria del software entra nella terza decade del XXI secolo, la qualità del software sta diventando sempre più essenziale per tutte le aziende e la conoscenza dei test del software sta diventando necessaria per tutti gli ingegneri del software. Oggi, il software definisce i comportamenti da cui dipende la nostra civiltà in sistemi come router di rete, motori di calcolo finanziario, reti di commutazione, Web, reti elettriche, sistemi di trasporto e servizi essenziali di comunicazione, comando e controllo. Negli ultimi due decenni, l'industria del software è diventata molto più grande, più competitiva e ha più utenti. Il software è un componente essenziale di applicazioni embedded insolite come aeroplani, astronavi e sistemi di controllo del traffico aereo, nonché di elettronica comune come orologi, forni, automobili, lettori DVD, apriporta di garage, telefoni cellulari e telecomandi. Le famiglie moderne hanno centinaia di processori e le nuove auto ne hanno più di mille; tutti eseguono software che i consumatori ottimisti presumono non falliranno mai. Sebbene molti fattori influenzino la progettazione di un software affidabile, tra cui, ovviamente, un'attenta progettazione e una corretta gestione del processo, il testing è il modo principale con cui l'industria valuta il software durante lo sviluppo.



They weren't so much different, but they had different goals

1.2 Principi di test del software

1.2.1 Il completamento del test del software

Il testing del software non viene mai completato. È un processo continuo il cui inizio coincide con quello del progetto stesso e continua fino a quando il progetto non viene più supportato. Nel corso della vita del software l'onere dei test si sposta lentamente: dagli sviluppatori durante la progettazione e la programmazione, ai team di tester indipendenti e infine ai clienti. Ogni azione che il cliente esegue con il software può essere considerata un test del software stesso.

Sfortunatamente, spesso intervengono vincoli di tempo e denaro: potrebbe non valere la pena spendere tempo o denaro da parte dello sviluppatore per correggere particolari errori software. Questo compromesso tra risorse spese e potenziali benefici può verificarsi facilmente per piccoli errori e per errori che non vengono riscontrati spesso dagli utenti del software.

È anche possibile (anche se difficile) essere statisticamente rigorosi quando si discute degli errori del software. Ad esempio, è possibile sviluppare un modello statistico del numero di guasti software previsti rispetto al tempo di esecuzione.

I tassi di errore su un dato periodo possono quindi essere specificati data una particolare probabilità. Quando tale probabilità è sufficientemente bassa, il test potrebbe essere considerato “completo”.

1.2.2 Scrivere software testabile

Un aspetto importante del test è garantire che il software scritto sia scritto in modo tale da poter essere facilmente testato.

Poiché il software diventa man mano più difficile da testare, verrà testato meno spesso.

Sebbene il cliente di solito non sia in grado di testare il software in modo approfondito come gli sviluppatori del software, questo sarà in grado di esaminare se il software soddisfa le sue esigenze, se i requisiti del software devono essere modificati e se sono presenti bug evidenti che gli sviluppatori hanno mancato.

Esistono numerose linee guida che gli ingegneri del software possono seguire per scrivere software che possa essere facilmente testato:

Operabilità: questa è in parte una qualità che si autoavvera: meno bug ha un sistema software, più facile sarà testare il software, poiché il test non progredirà in modo irregolare mentre si impiega tempo per riparare i bug. Chiaramente, maggiore sarà l'attenzione posta durante lo sviluppo del software per produrre codice privo di bug, più facili saranno i test che seguiranno.

Osservabilità: i test del software esaminano gli output prodotti dal software per input particolari. Ciò significa che il software è più facile da testare se gli input producono output distinti e prevedibili. Inoltre, può essere utile poter esaminare lo stato interno del software, soprattutto laddove potrebbero non esserci risultati prevedibili e soprattutto quando viene scoperto un errore.

Controllabilità: come appena accennato, il test del software implica l'esame degli output per determinati input. Ciò significa che quanto più facilmente si possono fornire input al software, tanto più facilmente si può testare il software. Ciò implica che il software è più testabile quando il tester ha la capacità di controllare facilmente il software per fornire gli input di test. Questa controllabilità si applica anche ai test: i test dovrebbero essere facilmente specificabili, automatizzati e riproducibili.

Scomponibilità: quando il software può essere scomposto in moduli indipendenti, questi moduli possono essere testati individualmente. Quando si verifica un errore in un singolo modulo, è meno probabile che l'errore richieda modifiche da apportare ad altri moduli o che il tester esamini anche più moduli.

Semplicità: chiaramente, più semplice è il software, meno errori avrà e più facile sarà testarlo. Esistono tre forme di semplicità che il software può dimostrare: **semplicità funzionale,** in cui il software non fa più di quanto sia necessario; **semplicità strutturale,** in cui il software è scomposto in piccole e semplici unità; e **la semplicità del codice,** in cui gli standard di codifica utilizzati dal team del software consentono una facile comprensione del codice.

Stabilità: se è necessario apportare modifiche al software, il test diventa più semplice se queste modifiche sono sempre contenute all'interno di moduli indipendenti (tramite, ad esempio, la scomponibilità), il che significa che il codice da testare rimane piccolo.

Comprensibilità: chiaramente, più i tester comprendono il software, più facile sarà testarlo.

Gran parte di ciò riguarda la buona progettazione del software, ma anche la cultura ingegneristica degli sviluppatori: la comunicazione tra progettisti, sviluppatori e tester ogni volta che si verificano modifiche nel software è importante, così come la capacità per tester e

sviluppatori di accedere facilmente a una buona documentazione tecnica. relativi al software (come le API per le librerie utilizzate e il software stesso).

1.3 Il Testing accademico – come viene insegnato il Testing

L'insegnamento del testing accademico svolge un ruolo cruciale nell'addestrare gli studenti di Ingegneria del Software a comprenderne i principi fondamentali e ad applicarli in scenari reali.

Esaminando il modo in cui viene insegnato il testing, troviamo due visioni contraddittorie, 2 paradigmi, per lo sviluppo del software nel suo insieme.

Da un lato c'è la scuola analitica (paradigma razionale) in cui l'enfasi è sulla creazione di casi di test utilizzando la precisione delle specifiche e molti tipi di modelli, cioè **test basati su modelli.** Questo paradigma ha molti sostenitori nel mondo accademico. D'altra parte, c'è la scuola guidata dal contesto (paradigma empirico) in cui l'enfasi è sulla creazione di casi di test che si adattano alle circostanze in cui il prodotto viene sviluppato e utilizzato attraverso l'esplorazione e la sperimentazione, tra cui: domande, studio, osservazione, inferenza, ecc., cioè **test esplorativi.**

Entrambe le scuole litigano da anni senza raggiungere un terreno comune. Ciò sicuramente non ha fatto nulla di buono per migliorare la formazione sul testing nelle università. Si ritiene che ciò sia dovuto al fatto che entrambe le scuole abbiano ragione: la progettazione dei casi di test dovrebbe essere sia basata su modelli che esplorativa. La progettazione dei casi di test richiede la creazione di senso che si ottiene attraverso l'esplorazione, le domande, lo studio, l'osservazione e la deduzione.

Successivamente, si può creare un modello di test che si può utilizzare per derivare casi di test e avere un'idea di ciò che si è (e non si è) testato.

Per definire e integrare adeguatamente materiali, pratiche e linee guida che aiutino gli studenti a progettare casi di test **basati su modelli esplorativi,** si ha bisogno di ottenere maggiori informazioni sui processi cognitivi e sulla creazione di senso da parte degli studenti che si occupano della progettazione dei casi di test.

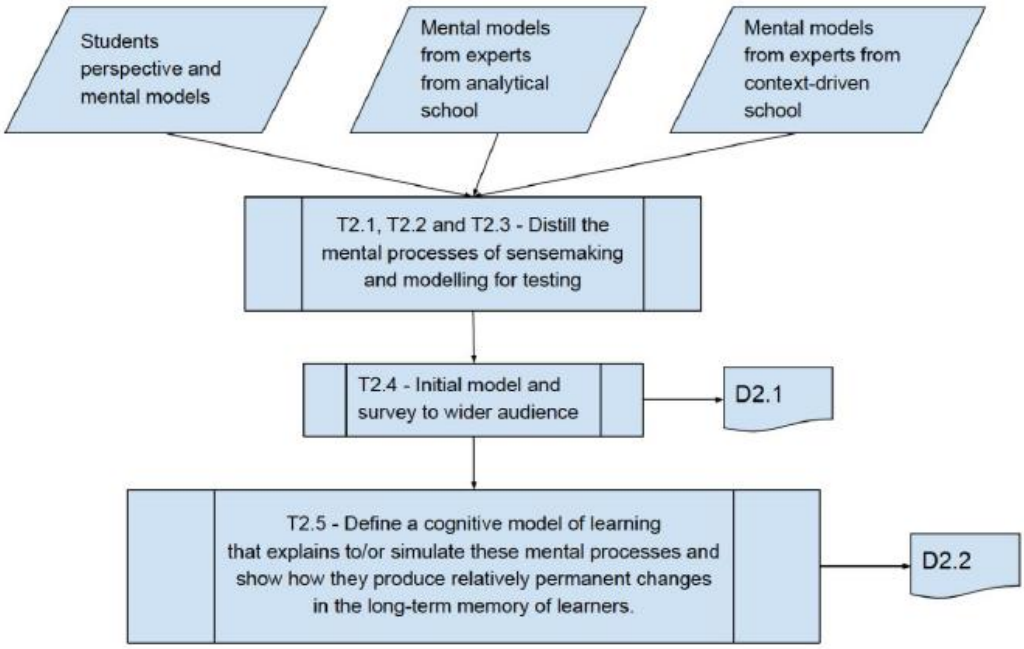
Il senso è una descrizione del processo di pensiero di qualcuno su come funziona qualcosa nel mondo reale. È una rappresentazione del mondo circostante, delle relazioni tra le sue varie parti e della percezione intuitiva di una persona riguardo ai propri atti e alle loro conseguenze.

Un modello cognitivo di apprendimento dovrebbe spiegare o simulare questi processi di creazione di senso e mostrare come producono cambiamenti relativamente permanenti nella memoria a lungo termine degli studenti.

Testare un sistema richiede un senso diverso rispetto a quello della programmazione del software.

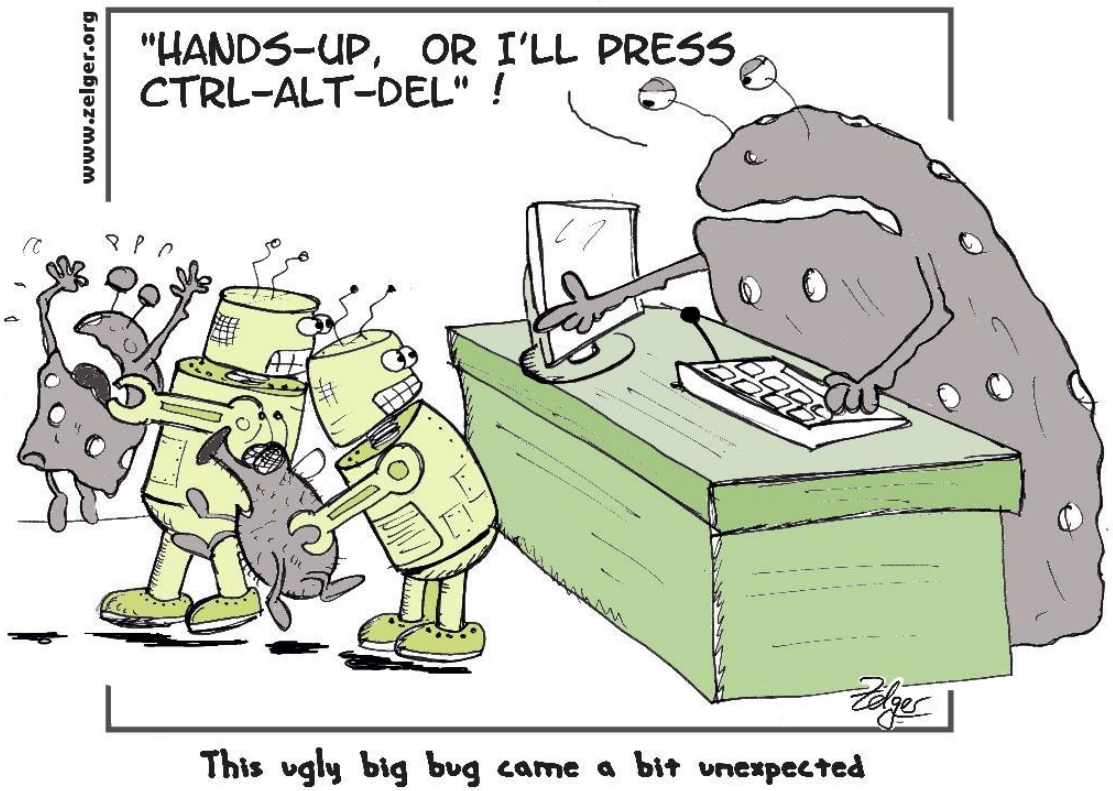
Bisogna combinare entrambi i mondi e prendere il meglio da ciascuno. Bisogna insegnare il lato tecnico dei test basato sul modello, senza ignorare la complessità e la creatività della parte relativa alla creazione di senso.

Si deve individuare il divario tra le tecniche di insegnamento e gli studenti, e allora occorre chiarire le competenze necessarie per eseguire con successo il processo di test come processo esplorativo basato su modelli. Per identificare il divario, si devono prima effettuare ricerche per comprendere i processi mentali e i processi di pensiero che gli esperti e gli studenti attraversano quando creano casi di test. Successivamente si definisce il modello cognitivo dell'apprendimento.



Lacks of Software Testing Courses

- **Software Testing courses are offered only by 30% of universities**
  - And they are always optional courses
- **On the other hand industries strongly requests for testing skills**
  - In particular, testing automation skills are requested



This ugly big bug came a bit unexpected

CAPITOLO 2

2.1 Come si è svolto l'esperimento

2.1.1 Introduzione

Contesto dell'Esperimento

L'esperimento descritto e sviscerato nel presente documento è stato condotto presso l'Università Degli Studi Di Napoli Federico II come parte del corso di Software Testing tenuto dal prof. Ing. Porfirio Tramontana nel semestre 1 dell'AA 2023/24 della Laurea Magistrale LM-32 in Ingegneria Informatica.

Il "Testing Innato" rappresenta un approccio particolare alla valutazione, enfatizzando la risposta naturale e spontanea dei soggetti di studio senza preavvisi o suggerimenti preliminari. In questa contestualizzazione, l'assenza di anticipazioni mira a catturare le reazioni autentiche e immediate dei partecipanti di fronte a stimoli o situazioni proposte.

© IMG [T.J. https://simply-the-test.blogspot.com]

Questa metodologia si basa sull'idea che il comportamento naturale e non guidato possa offrire insight preziosi sulla comprensione delle reazioni umane in determinati contesti, specialmente

nel contesto del testing. L'approccio senza preavvisi enfatizza la neutralità dell'ambiente di test, promuovendo una valutazione più pura e genuina delle capacità o risposte degli individui.

In sintesi, la scelta di adottare il "Testing Innato" sottolinea la volontà di esplorare le dinamiche spontanee e non influenzate dall'informazione preliminare, contribuendo a una comprensione più profonda delle reazioni dei soggetti nell'ambito specifico di studio.

Partecipanti

Il campione di partecipanti è stato composto da n.35 studenti iscritti al corso di Software Testing. Questi studenti sono per forza di cose tutti già Ingegneri Informatici triennali, ma il background di ognuno è comunque diverso, contribuendo così a una diversificazione delle esperienze e delle prospettive.

Competenze di Base

Gli studenti coinvolti nel presente esperimento hanno:

- una solida formazione nelle discipline di base: analisi matematica, fisica, geometria e algebra, informatica di base;
- una solida formazione nelle discipline affini e caratterizzanti tipiche del bagaglio culturale di un ingegnere dell'informazione: elettrotecnica, elettronica, telecomunicazioni, automazione e misure elettriche ed elettroniche.
- un'ampia e approfondita formazione nelle discipline dell'ingegneria informatica: architetture di elaborazione, ingegneria del software, progettazione e sviluppo di sistemi software, basi di dati, software di base e applicazioni su rete.

Prima di partecipare all'esperimento, quindi, gli studenti possedevano competenze di base comuni, comprese nozioni di programmazione in almeno un linguaggio di alto livello, la comprensione dei concetti di progettazione del software e la familiarità con le metodologie di sviluppo software. Queste competenze sono state considerate essenziali per garantire una base omogenea su cui basare l'esperimento di Software Testing.

2.1.2 Materiali e metodi

L'esperimento in questione, con l'obiettivo di essere implementato in tutta Europa, è attualmente in corso in Italia, Belgio, Portogallo e Spagna. In questa fase, ciascuno studente partecipante, senza interagire con gli altri, ha impiegato esclusivamente le proprie conoscenze pregresse e le competenze innate per sottoporre a prova le abilità di testing.

Il fine di questa iniziativa consiste nel raccogliere le strategie mentali e cognitive adottate dagli studenti durante l'esecuzione dei test. Sono stati condotti esperimenti in cui agli studenti è stato richiesto di riflettere ad alta voce, acquisendo, quando necessario, registrazioni dei loro schermi mentre affrontano compiti di modellazione dei test precedentemente assegnati.

Al termine di questa fase, si procederà all'analisi dei video, dei sottotitoli e dei documenti generati. Ciò consentirà di acquisire dati dettagliati sui processi cognitivi intrapresi dagli studenti durante la progettazione dei casi di test, mediante l'utilizzo di modellazione ed esplorazione.

Il risultato finale di questo compito sarà una relazione completa, volta a documentare in modo esauriente le strategie mentali e cognitive adoperate dagli studenti durante il processo di testing.

Gli esperimenti sono composti di 4 esercizi da svolgere, secondo due diverse sequenze:

-requisiti testuali – requisiti testuali – flow chart - sito web (schema ABD)

-requisiti testuali – requisiti testuali– sito web – flow chart (schema ACD)

Nello specifico l'esperimento, per questioni tempistiche e di facilità di esecuzione, è stato svolto a distanza utilizzando la piattaforma Microsoft Teams per l'iniziale somministrazione ed i successivi chiarimenti di eventuali dubbi; per il resto l'esperimento si è svolto in totale autonomia del singolo.

L'ambiente di esecuzione comprende un'applicazione di registrazione video e audio, consigliando ClipChamp per gli utenti Microsoft e suggerendo alternative equivalenti per i dispositivi macOS.

La sessione inizia con la verifica e il funzionamento degli strumenti di registrazione, inclusa la trascrizione del parlato e la registrazione dello schermo, che sarà necessaria in uno degli esercizi. Inoltre, ogni studente deve assicurarsi di poter inquadrare con la webcam solo le mani e il foglio su cui scrive, garantendo l'anonimato.

È richiesta una traccia stampata, carta, penna e un evidenziatore.

L'esperimento si suddivide in quattro esercizi, ognuno focalizzato su aspetti specifici del testing e della progettazione dei casi di test:

Esercizio 1: Testing a Partire dai Requisiti

- risoluzione presunta in 15 minuti
- utilizzo di ClipChamp o equivalenti per registrazione video, trascrizione automatica del parlato e quindi generazione dei sottotitoli
- documentazione dei casi di test cartacea

Esercizio 2: Testing a Partire dai Requisiti con Simulazione di Sito Web

- risoluzione presunta in 30 minuti
- utilizzo di ClipChamp o equivalenti per registrazione video, trascrizione automatica del parlato e quindi generazione dei sottotitoli
- documentazione dei casi di test cartacea

Esercizio 3/4: Testing a Partire dall'Osservazione del Sito Web

- risoluzione presunta in 30 minuti, estensione dell'esercizio precedente con osservazione di un sito Web di riferimento
- utilizzo di ClipChamp o equivalenti per registrazione schermo, trascrizione automatica del parlato e quindi generazione dei sottotitoli
- documentazione dei casi di test cartacea

Esercizio 4/3: Testing a Partire dal Flow Chart

- risoluzione presunta in 30 minuti, basato sui requisiti e con l'ausilio di un flow chart.
- utilizzo di ClipChamp o equivalenti per registrazione video, trascrizione automatica del parlato e quindi generazione dei sottotitoli
- documentazione dei casi di test cartacea

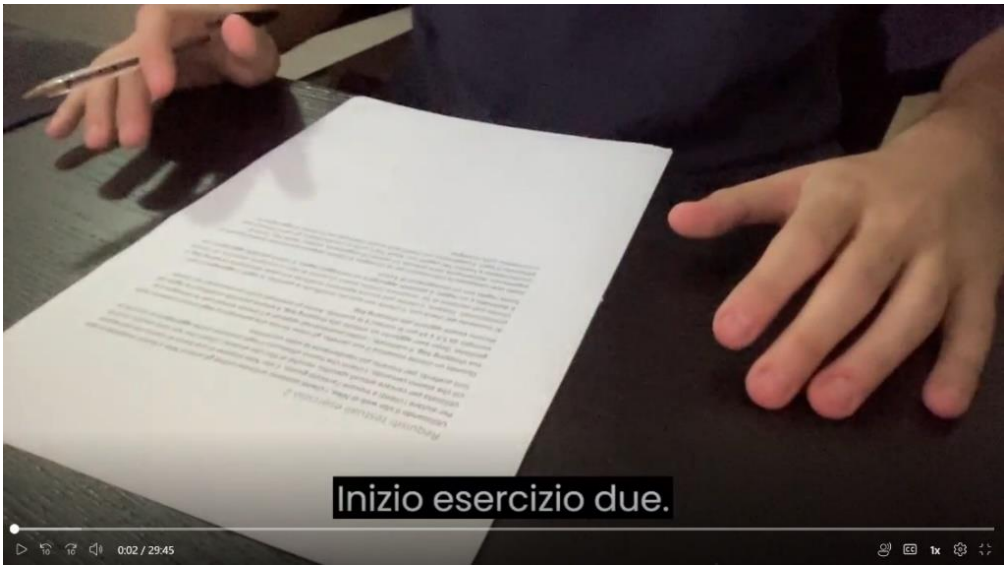
Al termine, è richiesta la consegna telematica dei filmati registrati, sottotitoli generati dai filmati e scansioni (o fotografie) dei fogli compilati insieme ai fogli della traccia che possono presentare annotazioni.

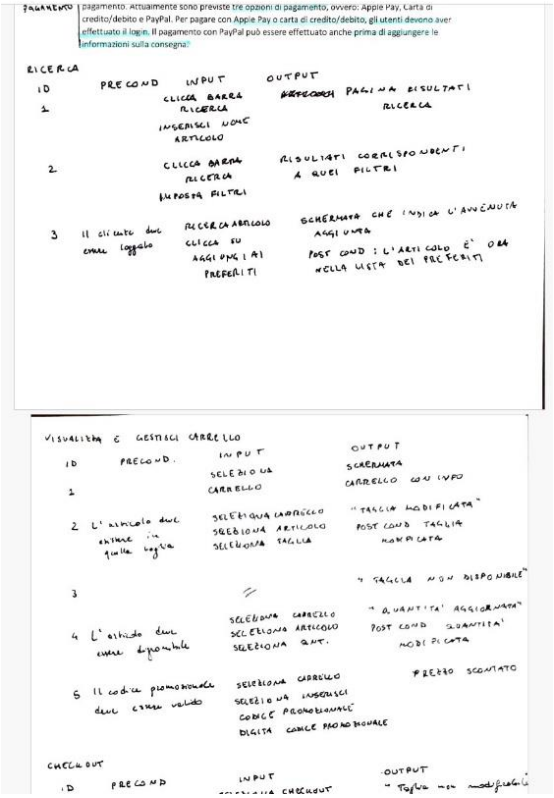
2.2 Costruzione e indagine del modello iniziale

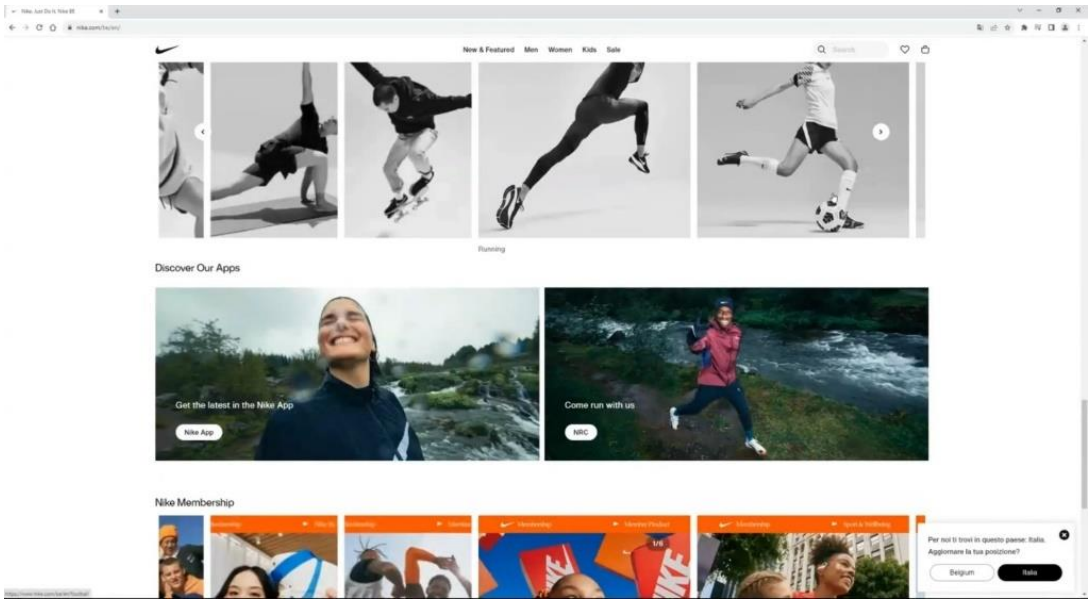
I dati raccolti nei compiti saranno analizzati in modo aggregato per costruire una comprensione iniziale dei modelli cognitivi durante le diverse attività di test.

Verranno studiate le difficoltà e la mancanza di competenze sperimentate dagli studenti, e in particolare i casi in cui gli studenti si discostano dalle strategie cognitive e dai modelli mentali dei tester esperti.

Sulla base di queste informazioni, verrà sviluppato un questionario che consentirà un'indagine su larga scala e la convalida di questa iniziale comprensione. Come risultato di questo compito, si avrà un rapporto con le intuizioni iniziali del modello cognitivo.




PAGAMENTO
pagamento. Attualmente sono previste tre opzioni di pagamento, ovvero: Apple Pay, Carta di credito/debito e PayPal. Per pagare con Apple Pay o carta di credito/debito, gli utenti devono aver effettuato il login. Il pagamento con PayPal può essere effettuato anche prima di aggiungere le informazioni sulla consegna.
RICERCA
ID
PRECOND
INPUT
OUTPUT
CLICCA BARRA RICERCA
INSERISCI NOME ARTICOLO
PAGINA RISULTATI RICERCA
CLICCA BARRA RICERCA
IMPOSTA FILTRI
RISULTATI CORRISPONDENTI A QUEI FILTRI
Il cliente deve essere loggato
RICERCA ARTICOLO
CLICCA SU AGGIUNGI AI PREFERITI
SCHERMATA CHE INDICA L'AVVENUTA AGGIUNTA
POST COND : L'ARTICOLO È ORA NELLA LISTA DEI PREFERITI
VISUALIZZA E GESTISCI CARRELLO
ID
PRECOND.
INPUT
OUTPUT
SELEZIONA CARRELLO
SCHERMATA CARRELLO CON LIVO
L'articolo deve entrare in quella taglia
SELEZIONA CARRELLO
SELEZIONA ARTICOLO
SELEZIONA TAGLIA
"TAGLIA MODIFICATA"
POST COND TAGLIA MODIFICATA
"TAGLIA NON DISPONIBILE"
SELEZIONA CARRELLO
SELEZIONA ARTICOLO
SELEZIONA QUANT.
"QUANTITA' AGGIORNATA"
POST COND QUANTITA' MODIFICATA
Il codice promozionale deve essere valido
SELEZIONA CARRELLO
SELEZIONA INSERISCI CODICE PROMOZIONALE
DIGITA CODICE PROMOZIONALE
PREZZO SCONTATO
CHECKOUT
ID
PRECOND
INPUT
OUTPUT


New & Featured
Men
Women
Kids
Sale
Running
Discover Our Apps
Get the latest in the Nike App
Nike App
Come run with us
NRC
Nike Membership
Per noi ti trovi in questo paese: Italia.
Aggiornare la tua posizione?
Belgium
Italia

2.3 Esercizi somministrati

Requisiti testuali esercizio 1

Amazon, un importante rivenditore online, ha stabilito diversi requisiti per il suo sito web. Chiede il vostro aiuto per verificare se tutti i requisiti sono soddisfatti. Spiegate la vostra strategia per sviluppare una serie di casi di test per verificare che l'applicazione soddisfi i seguenti requisiti.

1. I clienti membri hanno diritto alla spedizione gratuita e alla restituzione gratuita entro 30 giorni.
2. Per acquisti superiori a 100 euro la spedizione è gratuita, altrimenti il costo stimato per la consegna e la gestione è di 5 euro.
3. Se si dispone di un codice promozionale per una percentuale di sconto e l'acquisto (senza spese di spedizione) è superiore a 150 euro, il valore finale dell'acquisto (senza spese di spedizione) viene ridotto in base alla promozione.
4. Se si utilizza una carta regalo e il prezzo totale (tutto incluso) è superiore a 100 euro, il valore della carta regalo viene detratto dal prezzo totale finale.
5. L'output è il prezzo da pagare.

Requisiti testuali esercizio 2

Utilizzando il sito web di Nike, i clienti possono ordinare online gli articoli Nike e farseli recapitare a casa. Per aiutare i clienti a trovare l'articolo giusto, il sito Nike dispone di una barra di ricerca che può essere utilizzata per cercare articoli specifici, nonché di filtri che aiutano i clienti che non sono ancora sicuri di ciò che stanno cercando. I clienti che hanno effettuato il login possono anche aggiungere un articolo ai loro preferiti, per trovarlo più rapidamente la volta successiva.

Quando un cliente visualizza il suo carrello, gli viene fornita una panoramica degli articoli presenti nella sua shopping bag, il subtotale, i codici promozionali applicati e il prezzo stimato per la consegna e la gestione. Dopo aver aggiunto un articolo alla shopping bag, è ancora possibile modificare la taglia (ad esempio da 5,5 a 14 per le scarpe) e la quantità. Anche gli eventuali codici promozionali del cliente devono essere aggiunti alla shopping bag.

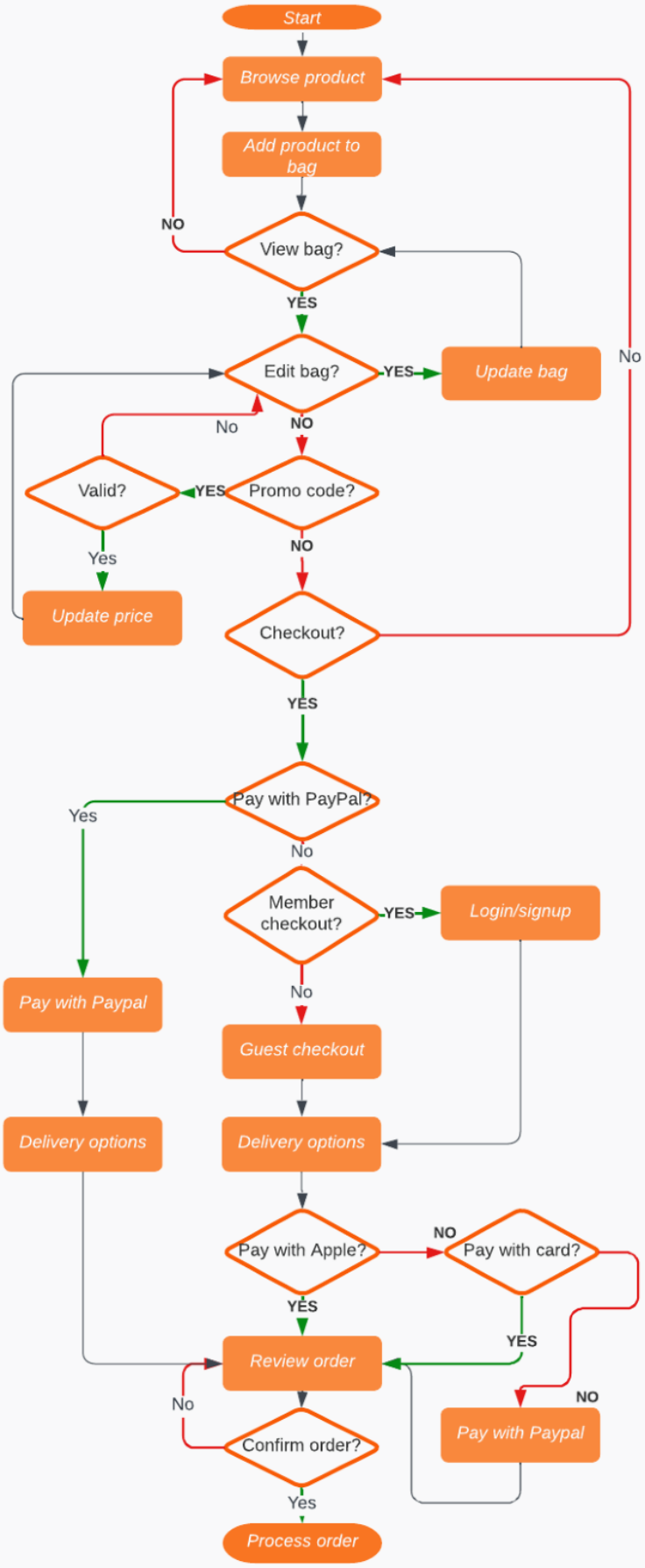
Al momento del check-out, il cliente non può più modificare la quantità, la taglia o aggiungere codici promozionali. Tuttavia, il cliente può ancora apportare modifiche tornando alla propria shopping bag. Il cliente può decidere di far recapitare l'ordine presso un punto di ritiro o al proprio domicilio. Se l'ordine è destinato a un regalo, è possibile aggiungere un messaggio regalo. È inoltre possibile aggiungere una busta regalo con un sovrapprezzo di 4 euro.

Dopo aver compilato le informazioni per la consegna, al cliente vengono presentate diverse opzioni di pagamento. Attualmente sono previste tre opzioni di pagamento, ovvero: Apple Pay, Carta di credito/debito e PayPal. Per pagare con Apple Pay o carta di credito/debito, gli utenti devono aver effettuato il login. Il pagamento con PayPal può essere effettuato anche prima di aggiungere le informazioni sulla consegna.

Sito Web di riferimento per il secondo esercizio

https://www.nike.com/be/en/

Flow chart Esercizio 2



2.4 Aspettative

Avendo sostenuto l'esame di Ingegneria del Software nel corso di laurea Triennale, che poi costituisce un importante pilastro per l'esame di Software Testing nel corso di laurea Magistrale, ci si aspetta un'idea di analisi ed una formalità di analisi derivante proprio da questi studi visto che l'esperimento viene somministrato ad Ingegneri Informatici Federiciani triennali.

Ci si aspetta un approccio alla risoluzione, una specifica dei casi di test inteso come l'insieme minimo di informazioni in grado di descrivere le specifiche di un caso di test, fatto di:

- Numero identificativo del caso di test (ID) e descrizione
- Condizioni iniziali (preconditions), cioè le ipotesi che devono verificarsi prima che il test è eseguito
- Valori input
- Valori output attesi
- Condizioni finali attese (postconditions), cioè le ipotesi che devono verificarsi dopo che il test è eseguito

E, nel caso del sito web, anche specifiche sul rapporto di esecuzione (execution report):

- Valori di output (in questo caso output osservati)
- Risultato, che può essere successo, fallimento o N/A

ID	Precond	Input	Expected Output	Output	PostCond	Result

2.4.1 Costruzione di modelli cognitivi

I dati dell'indagine verranno analizzati in modo da costruire una teoria sui modelli mentali e nelle strategie cognitive per i test utilizzati dagli studenti quando devono affrontare la progettazione dei casi di test.

Come risultato di questa attività, si avrà un rapporto con il modello cognitivo utilizzato per il test.

2.5 Raccolta delle strategie cognitive degli studenti durante la progettazione dei test case

Si costruisce un modello misurando l'occorrenza di eventi o l'applicazione delle strategie indicate di seguito dalla lettera C:

(M1) strategie cognitive chiave

Acquisire fiducia nella suite di test

Verificare la completezza dei casi di test

C1: Enumerazione del caso di test

Motivare un caso di prova

C2: Considerare la fattibilità del caso di test

C3: Fornire la motivazione dietro il caso di test

C4: Riformulazione del caso test precedente

Identificazione dei casi di test negativi

Definizione dei casi di test negativi

C5: Identificare il contesto del caso test negativo

C6: Identificazione del caso test negativo

Definizione degli esiti dei casi di test negativi

C7: Fornire il risultato atteso del caso test negativo

Identificazione dei casi di test positivi

Definizione dell'esito dei casi di test positivi

C8: Fornire il risultato atteso del caso di test positivo

Definizione di casi test positivi

C9: Identificare il contesto del caso test positivo

C10: Identificazione del caso test positivo

Identificazione dei casi di test tramite la conoscenza precedente

Identificazione di casi di test basati sull’esperienza

C11: Pensare fuori dagli schemi

C12: Utilizzare le conoscenze pregresse

Utilizzo dei casi di test precedenti per definire nuovi casi di test

C13: Estensione del test case precedente

C14: Identificare un nuovo caso di test dal caso di test precedente

Comprendere gli artefatti

Comprendere gli artefatti chiedendo aiuto

C15: Chiedere chiarimenti sull'esercizio

Comprendere gli artefatti da soli

C16: Lettura

C17: Pensare

(M2) modello mentale

Tecniche di progettazione delle prove

Basato sulle specifiche

C18: Boundary value analysis

C19: Cause effect graphing

C20: Classification tree method

C21: Combinatorial testing

C22: Decision table testing

C23: Equivalence partitioning

C24: Metamorphic testing

C25: Random testing

C26: Requirements-based testing

C27: Scenario testing

C28: Scenario testing

C29: Syntax testing

C30: Use case testing

Basato sulla struttura

C31: Branch condition combination testing

C32: Branch condition testing

C33: Branch testing

C34: Dataflow testing

C35: Decision testing

C36: Error guessing

C37: Experience based testing

C38: MCDC testing

C39: Statement testing

Livelli di prova

C40: Acceptance testing

C41: Integration testing

C42: System integration testing

C43: System testing

C44: Unit testing

Pratiche di prova

C45: AB testing

C46: Automated testing

C47: Back-to-back testing

C48: Experienced-based testing

C49: Exploratory testing

C50: Fuzz testing

C51: Keyword-driven testing

C52: Manual testing

C53: Mathematical-based testing

C54: Model-based testing

C55: Scripted testing

Tipi di test

C56: Accessibility testing

C57: Compatibility testing

C58: Conversion testing

C59: Disaster-recovery testing

C60: Functional testing

C61: Installability testing

C62: Interoperability testing

C63: Localization testing

C64: Maintainability testing

C65: Performance related testing

C66: Portability testing

C67: Procedure testing

C68: Reliability testing

C69: Security testing

C70: Usability testing

CAPITOLO 3

3.1 Risultati e discussione dei risultati

3.1.1 Analisi dei risultati

Al termine dell'esperimento, emerge chiaramente che non tutti i 70 elementi da misurare del modello originario possono essere osservati.

Ciò deriva dal fatto che gli studenti, in maniera differenziata, non hanno adottato tutte le strategie cognitive e i modelli mentali inizialmente auspicati.

In considerazione di ciò, poiché diversi campi mostrano valori netti di 0, il modello effettivo subisce una significativa riduzione e assume una configurazione più limitata.

3.1.2 Strategie di analisi

È possibile valutare i dati mediante:

-Istogrammi come rappresentazione grafica di una distribuzione di dati. È un tipo di grafico a barre che mostra la frequenza di osservazioni in intervalli di dati. Gli istogrammi sono comunemente utilizzati per visualizzare la distribuzione di un insieme di dati e comprendere la forma della distribuzione.

Gli istogrammi sono spesso utilizzati per visualizzare la distribuzione delle frequenze in dati statistici. Possono essere particolarmente utili per identificare modelli, simmetrie o asimmetrie nella distribuzione dei dati, nonché la presenza di valori anomali. Inoltre, gli istogrammi forniscono un'indicazione visiva della concentrazione o della dispersione dei dati.

Ecco alcuni elementi chiave di un istogramma:

Barre:

Le barre rappresentano gli intervalli o le classi di dati. Sono disposte in modo adiacente l'una all'altra e la loro altezza indica la frequenza delle osservazioni all'interno di ciascun intervallo.

Asse Orizzontale (Asse X):

L'asse orizzontale rappresenta la scala dei dati o delle categorie. Gli intervalli di dati sono disposti lungo questo asse.

Asse Verticale (Asse Y):

L'asse verticale rappresenta la frequenza delle osservazioni in ciascun intervallo. La scala dell'asse Y indica il numero di osservazioni o la percentuale di osservazioni.

Intervalli o Classi:

Gli intervalli o classi sono i segmenti in cui sono suddivisi i dati. Questi intervalli possono variare in dimensioni e sono scelti in modo che coprano l'intero intervallo di dati. La larghezza degli intervalli può influenzare l'interpretazione dell'istogramma.

-Valore Medio (o Media):

La media, o valore medio, rappresenta la somma di tutti i valori in un insieme diviso per il numero totale di elementi in quell'insieme. Questo valore è calcolato per ottenere una stima del "centro" dell'insieme di dati.

-Valore Mediano:

Il valore mediano è il valore che divide l'insieme di dati in due parti uguali. In altre parole, il 50% dei dati è inferiore al valore mediano, mentre il restante 50% è superiore ad esso. Il valore mediano è particolarmente utile quando si lavora con distribuzioni non simmetriche o quando ci sono valori estremi.

Calcolo (per un insieme di dati ordinato):

- Se il numero di dati è dispari, il valore mediano è il valore centrale.
- Se il numero di dati è pari, il valore mediano è la media dei due valori centrali.

-Conteggio dei Valori Maggiori di 0:

Il conteggio dei valori maggiori di 0 indica quante osservazioni o elementi in un insieme di dati sono positivi. Questo tipo di conteggio è spesso utilizzato per valutare la presenza o la frequenza di valori positivi in un insieme di dati.

3.2 Analisi approfondita

C1: Enumerazione del caso di test

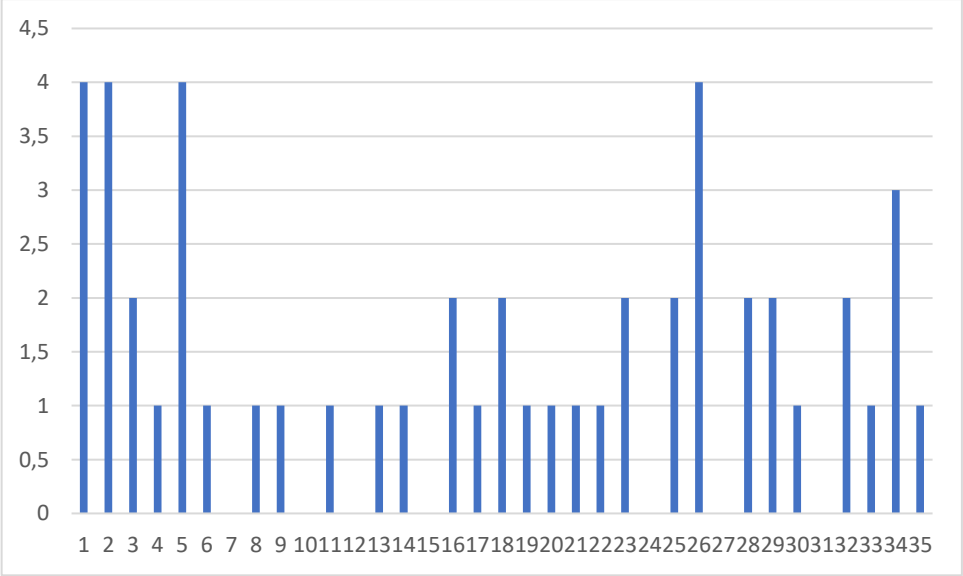
Lo studente enumera tutti i casi di test precedentemente definiti per verificare se ritiene di aver testato completamente il software.

Something about coverage

In questo processo di revisione, l'obiettivo è esaminare attentamente ciascun caso di test e valutarne l'efficacia nella copertura delle diverse funzionalità e scenari previsti dal software. Questo approccio mira a garantire una verifica esaustiva e meticolosa del sistema, consentendo allo studente di valutare la robustezza e la conformità del software agli standard prestabiliti.

Ad esempio "*ho testato il caso in cui l'acquisto è superiore a 100 euro inferiore a 100 euro e poi inferiore a 100 euro ma poi con la spedizione è superiore a 100 euro ma non ho testato il caso in cui è esattamente uguale a 100 euro ma sarebbe lo stesso del caso inferiore a 100 euro perché i requisiti sono superiori a 100 euro, altrimenti è appena inferiore o uguale a 100 euro"*

Si nota che quasi tutti gli studenti oggetto dell'esperimento procedono enumerando i vari casi di test, chi facendolo più volte, chi una volta sola. Infatti, i risultati risultano abbastanza allineati.



Valore medio (o media) C1: 50/28=1,471

Valore mediano C1: 1

Conteggio dei valori maggiori di 0 C1: 28

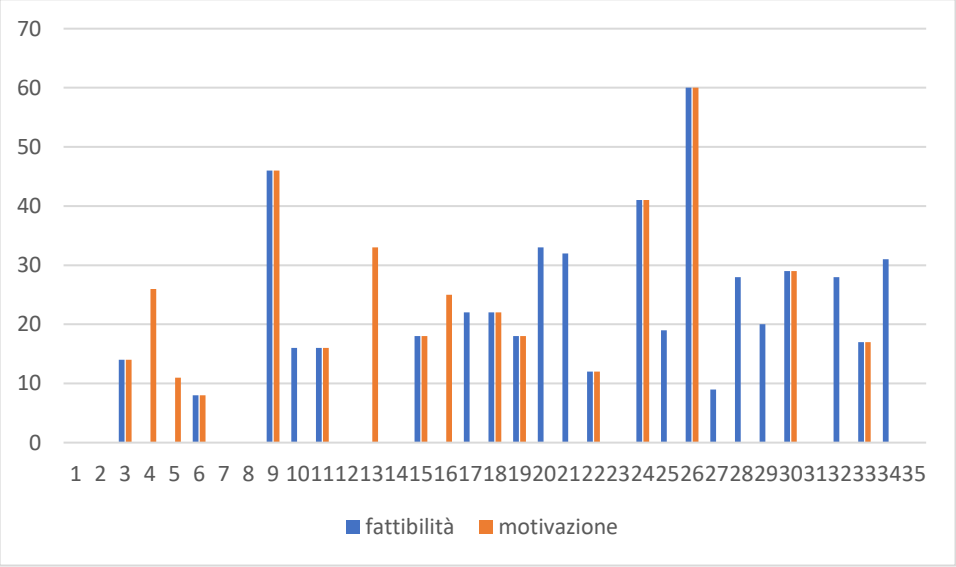
C2: Considerare la fattibilità del caso di test / C3: Fornire la motivazione dietro il caso di test

Lo studente considera la fattibilità di un caso di test da lui definito e motiva da solo perché un caso di test è buono o perché è stato realizzato.

Nel processo di revisione, lo studente valuta la fattibilità di un caso di test da lui definito e, in modo autonomo, fornisce una motivazione dettagliata sulla qualità del caso di test e sul motivo per cui ritiene che sia stato realizzato in modo efficace. In questa analisi, lo studente esamina attentamente le caratteristiche del caso di test, verificando se copre in modo adeguato gli scenari e le funzionalità previste. La motivazione autonoma del perché considera un caso di test valido può includere l'aderenza agli obiettivi specifici di testing, la completezza nella copertura dei requisiti, la chiarezza nell'esecuzione e la capacità di rilevare potenziali difetti nel software. Questo approccio permette allo studente di sviluppare un apprezzamento critico delle proprie attività di testing e di contribuire alla maturità della sua competenza nel campo del software testing.

Es .*C3 "Sì, possono pagare solo tramite PayPal perché applicano il pagamento e la carta di credito/debito è consentita solo ai clienti che hanno effettuato l'accesso"*

Si osserva che chi procede con un'analisi molto più completa, utilizzando il modello insegnato dal corso di Ingegneria del Software, considera sempre la fattibilità e la motivazione dietro al caso di test, chi ha un approccio invece molto più libero, non ingegneristico, tende a non farlo.



Valore medio (o media) C2: 17,97

Valore mediano C2: 17,5

Conteggio dei valori maggiori di 0 C2: 22

Valore medio (o media) C3: 13,66

Valore mediano C3: 11

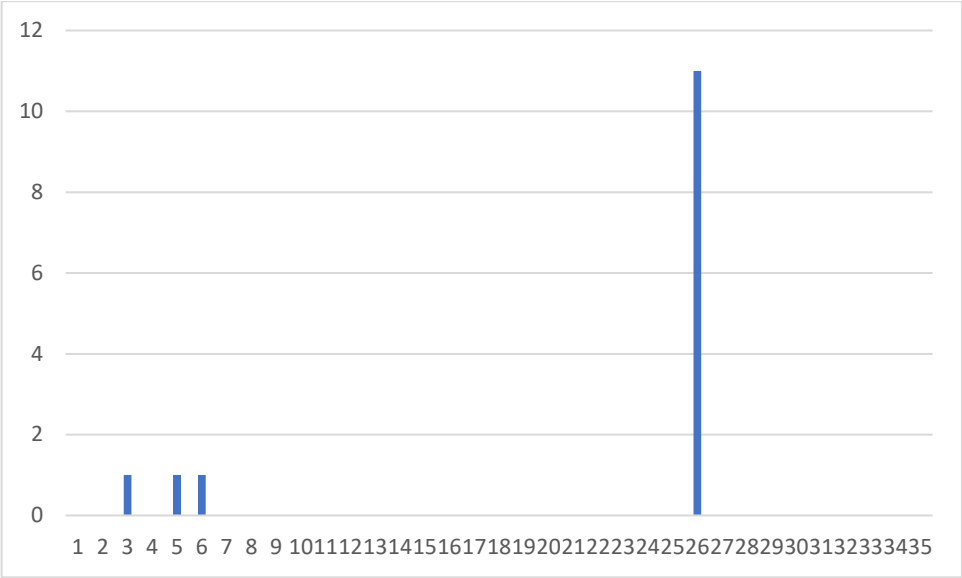
Conteggio dei valori maggiori di 0 C3: 16

C4: Riformulazione del caso test precedente

Lo studente riformula un caso di test che ha già definito in precedenza in modo diverso.

Questa pratica di riformulazione implica la revisione e la reinterpretazione del caso di test, mirando a esplorare nuove prospettive e modalità di verifica delle funzionalità software. Tale approccio può essere motivato da diverse ragioni, tra cui la ricerca di un linguaggio più chiaro, la considerazione di scenari alternativi o l'ottimizzazione dell'efficacia del test. La riformulazione dei casi di test rappresenta un processo dinamico che consente allo studente di affinare e migliorare continuamente le proprie abilità di progettazione dei test nel contesto del software testing.

Si osserva che in generale nessuno studente, tranne uno (nello specifico studente 26), che produce innumerevoli casi di test guidati da scenari, ha questo approccio.



Valore medio (o media) C4: 1,556

Valore mediano C4: 0

Conteggio dei valori maggiori di 0 C4: 4

C5: Identificare il contesto del caso test negativo / C6: Identificazione del caso test negativo / C7: Fornire il risultato atteso del caso test negativo

Lo studente definisce il contesto di un caso di test con un risultato negativo atteso, definisce un caso di test con un risultato negativo atteso e menziona il risultato atteso di un caso di test negativo.

Nel processo di progettazione del caso di test, lo studente inizia delineando il contesto in cui si verifica un risultato negativo atteso. Successivamente, formula il caso di test specifico, inserendo le condizioni o gli input che porterebbero a tale risultato indesiderato. Infine, esplicita chiaramente il risultato atteso associato al caso di test negativo, evidenziando la discrepanza tra il comportamento atteso e quello rilevato durante l'esecuzione del test. Questa pratica consente ai tester di identificare e valutare la gestione degli scenari indesiderati da parte del software,

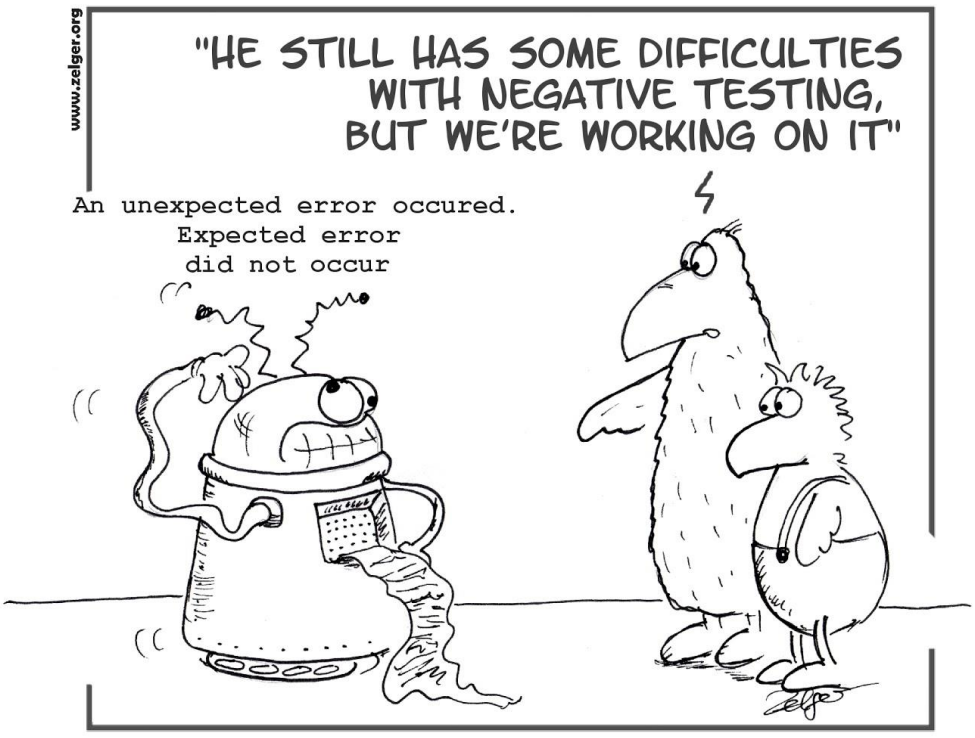
contribuendo così a migliorare la robustezza complessiva dell'applicazione nel rilevare e gestire situazioni anomale.

Esempio C5: *se il cliente è un membro e l'importo è di 100 EUR*

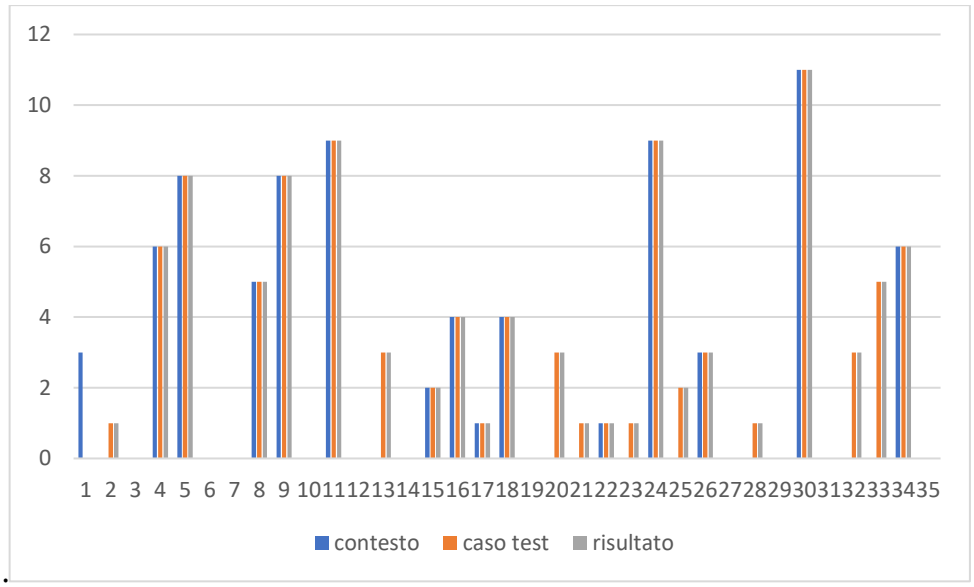
Esempio C6: *…forse per un valore negativo ma non penso che il sistema consentirebbe un valore negative*

Esempio C7*: verrà visualizzato come non valido e il prezzo rimarrà lo stesso*

Ci si aspetterebbe che C5, C6 e C7 debbano procedere di pari passo e con valori molto simili, se non uguali. Tuttavia, chi adotta una metodologia più libera, non conformandosi (o non conformandosi pienamente) ai principi dell'Ingegneria del Software, o semplicemente chi tralascia a turno elementi cruciali come input atteso, output atteso, pre-conditions e post-conditions, può generare anomalie significative.



Robot learning different testing techniques



Valore medio (o media) C5: 2,5

Valore mediano C5: 0

Conteggio dei valori maggiori di 0 C5: 15

Valore medio (o media) C6: 2,939

Valore mediano C6: 2

Conteggio dei valori maggiori di 0 C6: 23

Valore medio (o media) C7: 2,939

Valore mediano C7: 2

Conteggio dei valori maggiori di 0 C7: 23

C8: Fornire il risultato atteso del caso di test positivo / C9: Identificare il contesto del caso test positivo / C10: Identificazione del caso test positivo

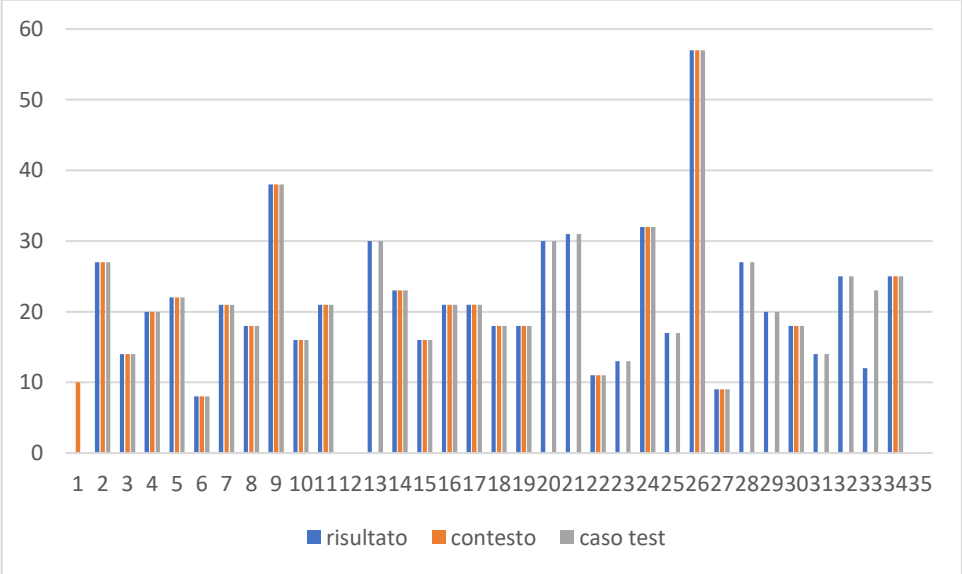
Lo studente menziona il risultato atteso di un caso di test positivo, definisce il contesto di un caso di test con un risultato positivo atteso e definisce un caso di test con un risultato positivo atteso.

Nel processo di progettazione del caso di test, lo studente inizia specificando chiaramente il risultato atteso di un caso di test positivo. Successivamente, definisce il contesto in cui si verifica un risultato positivo atteso, delineando le condizioni e gli input necessari per raggiungere questo esito desiderato. Infine, formula il caso di test, includendo le azioni o gli eventi che dovrebbero verificarsi e conducono al risultato positivo atteso. Questo approccio consente allo studente di valutare in modo accurato il comportamento del software sotto condizioni ottimali, verificando che risponda correttamente e in modo conforme alle aspettative quando sottoposto a input positivi. La definizione chiara di casi di test positivi è essenziale per garantire la validità e l'efficacia del processo di testing.

Esempio C8: *il cliente dovrebbe comunque ricevere la spedizione gratuita*

Esempio C9: *quando il cliente è membro e l'importo dell'acquisto è di 102 EUR*

Così come per i casi di test negativi: Ci si aspetterebbe che C8, C9 e C10 debbano procedere di pari passo e con valori molto simili, se non uguali. Tuttavia, chi adotta una metodologia più libera, non conformandosi (o non conformandosi pienamente) ai principi dell'Ingegneria del Software, o semplicemente chi tralascia a turno elementi cruciali come input atteso, output atteso, pre-conditions e post-conditions, può generare anomalie significative.



Valore medio (o media) C8: 20,38

Valore mediano C8: 20

Conteggio dei valori maggiori di 0 C8: 32

Valore medio (o media) C9: 14,67

Valore mediano C9: 16

Conteggio dei valori maggiori di 0 C9: 23

Valore medio (o media) C10: 20,71

Valore mediano C10: 20,5

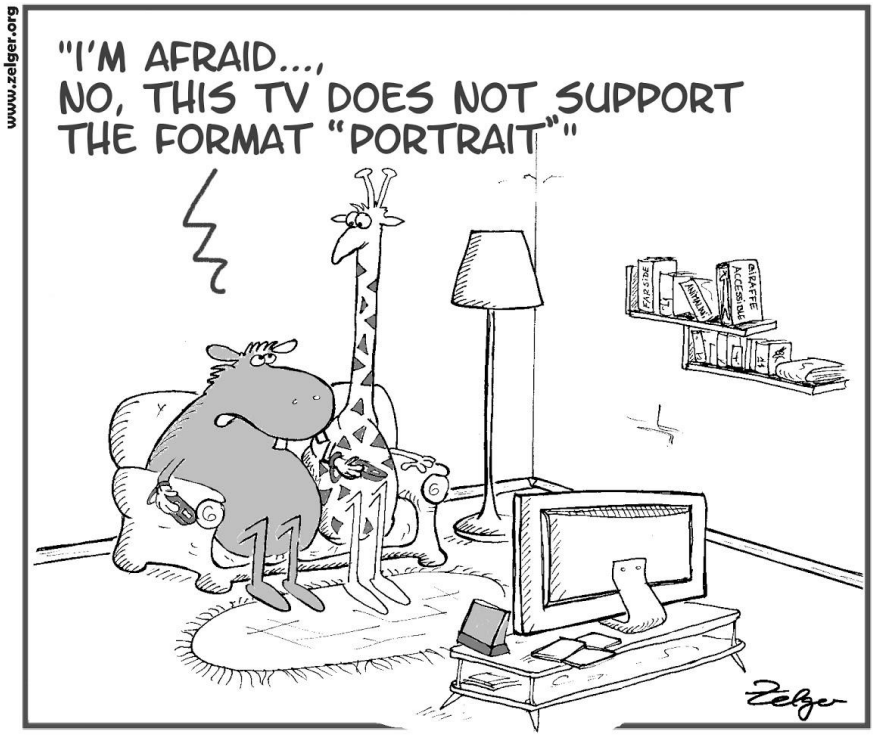
Conteggio dei valori maggiori di 0 C10: 32

C11: Pensare fuori dagli schemi

Lo studente pensa fuori dagli schemi quando definisce un caso di test

In questo contesto, il termine "pensare fuori dagli schemi" si riferisce alla capacità di andare oltre le soluzioni convenzionali o agli approcci tradizionali nel processo di progettazione dei test.

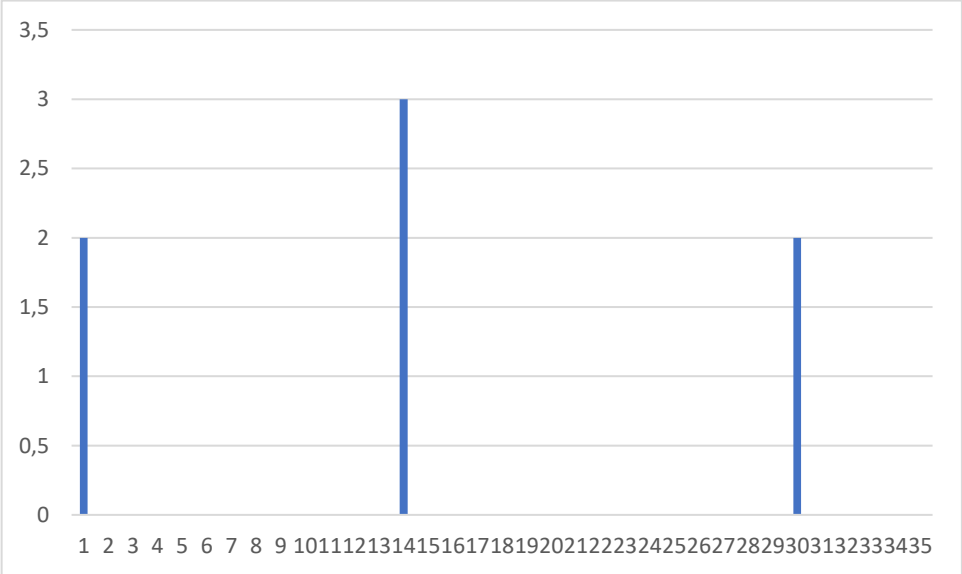
Lo studente non si limita a seguire procedure predefinite, ma piuttosto esplora e considera creativamente scenari, input o situazioni non convenzionali che potrebbero rivelare aspetti inattesi o criticità nel software. Questa abilità di pensiero innovativo arricchisce il processo di testing, contribuendo a identificare potenziali punti deboli o comportamenti non previsti, e dimostra una mentalità flessibile e aperta alla scoperta di nuove prospettive nel contesto del software testing.



Another purchase that just wasn't giraffe accessible

Ad esempio, *forse funziona, ma in realtà se lo aggiorno i dati scompariranno*

Pochi studenti pensano fuori dagli schemi, facendo quindi analisi poco convenzionali.



Valore medio (o media) C11: 0,318

Valore mediano C11: 0

Conteggio dei valori maggiori di 0 C11: 3

C12: Utilizzare le conoscenze pregresse

Quando uno studente si impegna nella definizione di un caso di test, integra le proprie conoscenze precedenti acquisite da varie fonti, come esperienze personali, contesti lavorativi precedenti o percorsi di studio pregressi. Questa pratica evidenzia la consapevolezza dello studente nell'utilizzare in modo strategico le sue competenze pregresse per arricchire il processo di progettazione dei test.

Le esperienze personali apportano una prospettiva unica, consentendo allo studente di comprendere le esigenze degli utenti finali e di valutare il software da una prospettiva più orientata agli utenti. Le esperienze lavorative precedenti forniscono una comprensione pratica delle sfide specifiche del settore e delle dinamiche operative, mentre gli studi passati contribuiscono alla solidificazione delle conoscenze teoriche.

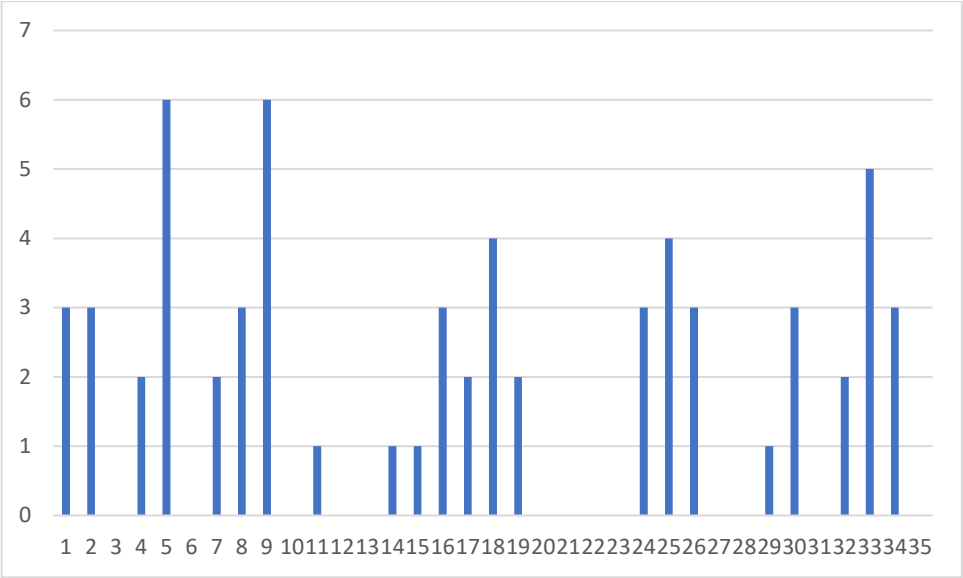
L'impiego di conoscenze pregresse durante la definizione dei casi di test non solo arricchisce la qualità degli stessi, ma dimostra anche la capacità dello studente di applicare in modo attivo e mirato le lezioni apprese in contesti precedenti.

Typical reading for aspiring killer bugs

Questo approccio riflette un'attitudine orientata alla risoluzione dei problemi e all'ottimizzazione delle attività di testing attraverso l'applicazione pratica delle esperienze accumulate nel corso del tempo.

Ad esempio, *avere dei contatori di intervallo che contano il passaggio di 30 giorni*

Si può accomunare a chi procede ad analizzare secondo schematiche ingegneristiche (schematiche insegnate da Ingegneria del Software) e chi meno: i risultati sono abbastanza frastagliati e divisi all'incirca a metà tra chi predilige prevenire problematiche riguardanti il coding, chi tenere d'occhio l'usabilità e l'efficienza del software e tenere conto della possibile soddisfazione dell'utente finale.



Valore medio (o media) C12: 1,8

Valore mediano C12: 2

Conteggio dei valori maggiori di 0 C12: 22

C13: Estensione del test case precedente

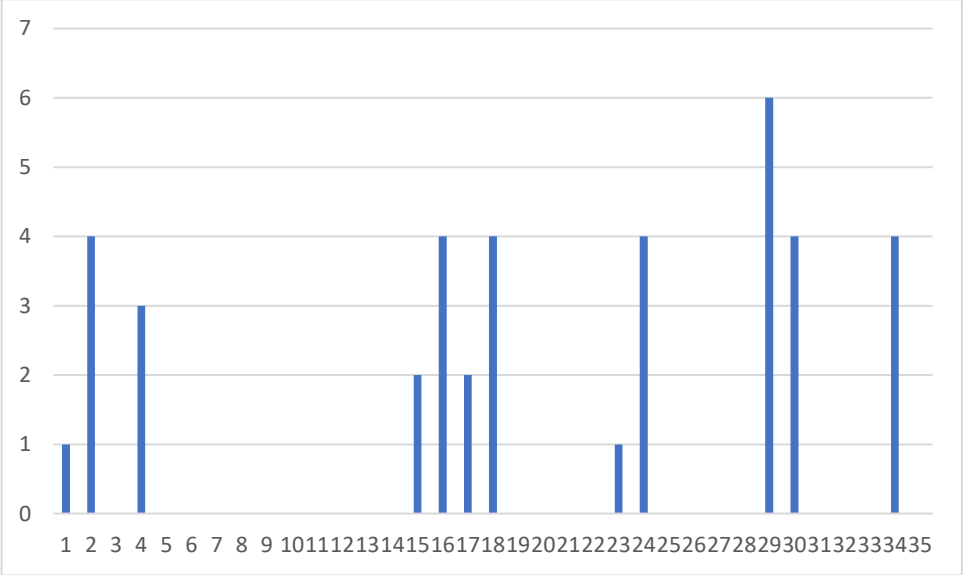
L'estensione di un test case precedente si riferisce al processo di modifica o espansione di un caso di test esistente al fine di coprire nuove funzionalità, scenari o requisiti. Questa pratica è comune quando vengono introdotte modifiche nel software o quando si desidera garantire una copertura più ampia del sistema.

Quando si estende un test case, è possibile apportare modifiche agli input, alle azioni o alle aspettative in modo da riflettere le nuove condizioni o funzionalità da testare. Questo assicura che il caso di test rimanga rilevante e in grado di rilevare eventuali difetti derivanti dalle modifiche apportate al software.

L'estensione di test case è fondamentale per mantenere la robustezza del set di test nel tempo, adattandolo alle evoluzioni del software e garantendo che i cambiamenti introdotti non compromettano la qualità complessiva del sistema. Inoltre, questa pratica contribuisce a

mantenere aggiornata la suite di test e a garantire una copertura completa delle funzionalità nel corso dello sviluppo del software.

In questo caso, oltre agli studenti che analizzano già tutto partendo dalla lettura dei requisiti, si nota che qualche studente ritorna sul cast case precedente estendendolo magari in negativo.



Valore medio (o media) C13: 3

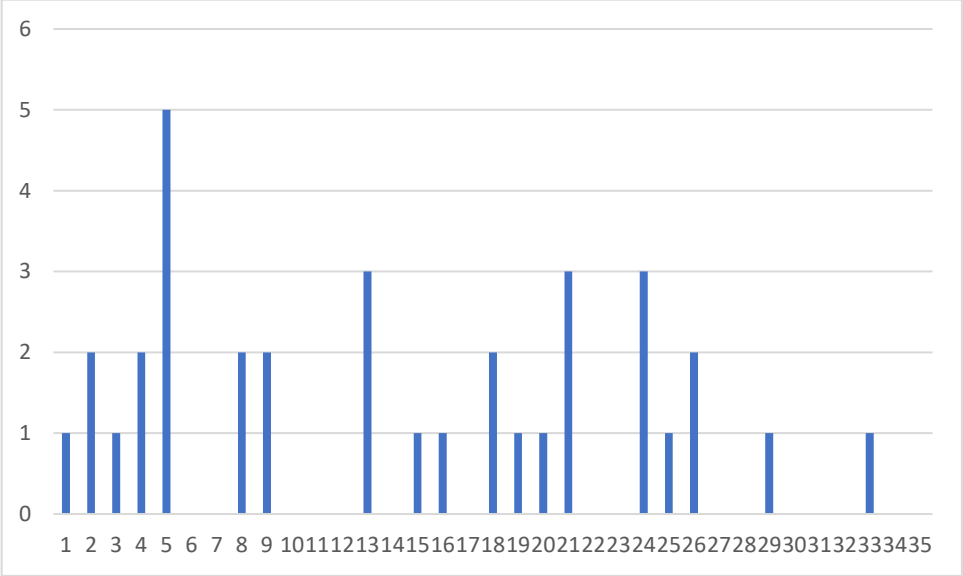
Valore mediano C13: 4

Conteggio dei valori maggiori di 0 C13: 12

C14: Identificare un nuovo caso di test dal caso di test precedente

Identificare un nuovo caso di test dal caso di test precedente può essere un processo importante nel migliorare la copertura del test e nell'assicurarsi che le nuove funzionalità o modifiche del software siano adeguatamente testate, o un rimedio per colmare delle mancanze e delle casistiche non valutate nel caso di test precedente.

A differenza del C13, dove lo studente modifica o estende un caso di test precedente al fine di coprire nuove funzionalità, qui lo studente procede con la creazione ed identificazione di un nuovo caso di test con implementazione autonoma.



Valore medio (o media) C14: 1

Valore mediano C14: 1

Conteggio dei valori maggiori di 0 C14: 19

FOCUS: Differenze e similitudini C13/C14

Estendere un caso di test precedente significa aggiungere nuove condizioni o scenari di test senza modificarne le caratteristiche principali. D'altra parte, identificare un nuovo caso di test potrebbe comportare la creazione di un test completamente nuovo basato sulle funzionalità o sugli scenari di test già definiti.

Ecco alcune differenze tra l'estensione di un caso di test precedente e l'identificazione di un nuovo caso di test:

C13: Estensione del test case precedente

1. **Coerenza:** L'estensione mantiene la coerenza con il caso di test precedente, ampliando semplicemente le condizioni o gli scenari di test.
2. **Utilizzo di Caso di Test Esistente:** Si parte da un caso di test esistente e se ne aggiungono nuovi elementi o scenari senza modificarne la struttura di base.
3. **Focus Sugli Stessi Requisiti:** L'estensione è utile quando si desidera testare funzionalità simili o correlate a quelle già definite nel caso di test precedente.
4. **Risparmio di Tempo:** L'estensione può essere più veloce da implementare poiché si basa su un caso di test esistente.

C14: Identificare un nuovo caso di test dal caso di test precedente

1. **Creazione da Zero:** Un nuovo caso di test viene creato da zero per testare funzionalità, requisiti o scenari unici o differenti.
2. **Indipendenza:** Non è legato a un caso di test precedente e può essere creato in modo indipendente.
3. **Copertura di Nuove Funzionalità:** L'obiettivo potrebbe essere quello di coprire funzionalità o requisiti che non sono stati precedentemente testati.
4. **Esplorazione di Nuovi Scenari:** Identifica nuovi scenari di test che potrebbero non essere coperti dai casi di test esistenti.

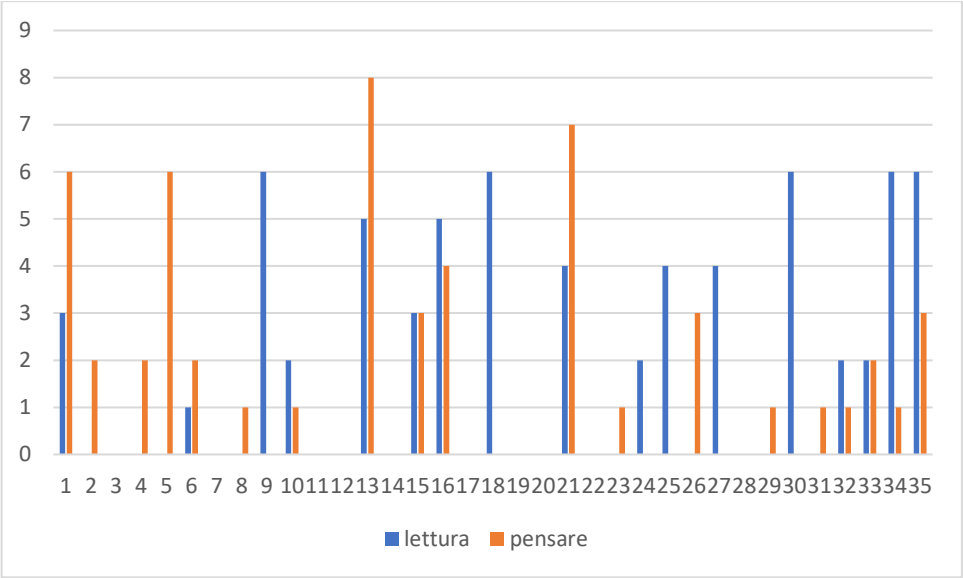
In sintesi, l'estensione di un caso di test precedente è una pratica comune per aggiungere dettagli o casi specifici senza reinventare la ruota, mentre l'identificazione di un nuovo caso di test è appropriata quando si desidera esplorare nuove funzionalità o scenari che richiedono una progettazione e implementazione autonoma. Entrambe le approcci sono utili a seconda del contesto e degli obiettivi di testing.

C16: Lettura

In generale, la lettura ad alta voce, è una pratica che potrebbe portare ad una migliore comprensione e ad un rilevamento precoce di potenziali errori.
Non tutti gli studenti leggono ad alta voce e tutto il testo prima di cominciare la loro analisi, alcuni ritornano su una lettura implicita, altri ritornano su una lettura esplicita.
Questa varietà di approcci sottolinea l'importanza di adottare strategie di studio personalizzate che rispecchino le preferenze individuali degli studenti. Nel contesto dell'insegnamento e dell'apprendimento, considerare e rispettare questi diversi stili di studio può contribuire a creare un ambiente più inclusivo e adattato alle esigenze degli studenti.

C17: Pensare

Ovviamente tutti gli studenti si suppone pensino, ma non tutti lo evidenziano con gesti o con suoni del tipo "*mh*" durante l'esperimento suggerendoci che, mentre è plausibile che gli studenti stiano riflettendo mentalmente durante l'esperimento, questa riflessione non è sempre manifesta attraverso espressioni verbali o gestuali come il suono "*mh*" o altri segnali fisici.
Questa constatazione sottolinea la diversità nelle modalità con cui gli individui comunicano il proprio pensiero durante un esperimento. Alcuni studenti potrebbero preferire esprimere le loro riflessioni attraverso gesti o suoni, indicando in modo più esplicito il processo di pensiero in corso. Al contrario, altri potrebbero mantenere il proprio pensiero più interno, senza manifestare esternamente segnali udibili o visibili.
È importante riconoscere che le modalità di espressione del pensiero possono variare ampiamente tra gli individui e che la mancanza di segnali udibili o visibili non indica necessariamente una mancanza di riflessione. Questa osservazione può influenzare la progettazione degli esperimenti o degli ambienti di testing, sottolineando l'importanza di considerare diverse modalità di comunicazione e di valutare il pensiero degli studenti in modo completo e inclusivo.



Valore medio (o media) C16: 1,971

Valore mediano C16: 0,5

Conteggio dei valori maggiori di 0 C16: 17

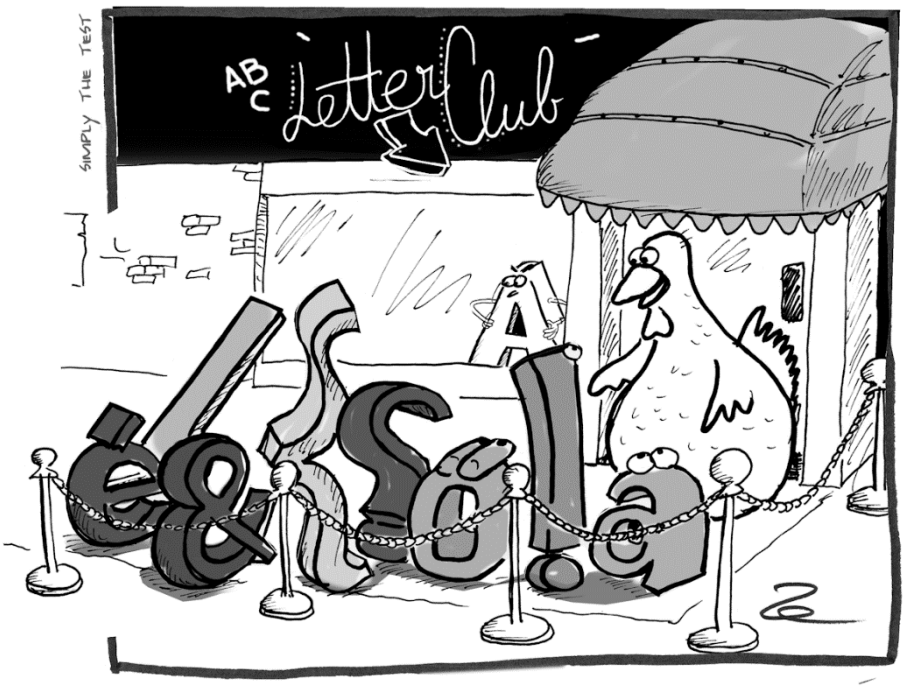
Valore medio (o media) C17: 1,618

Valore mediano C17: 1

Conteggio dei valori maggiori di 0 C17: 19

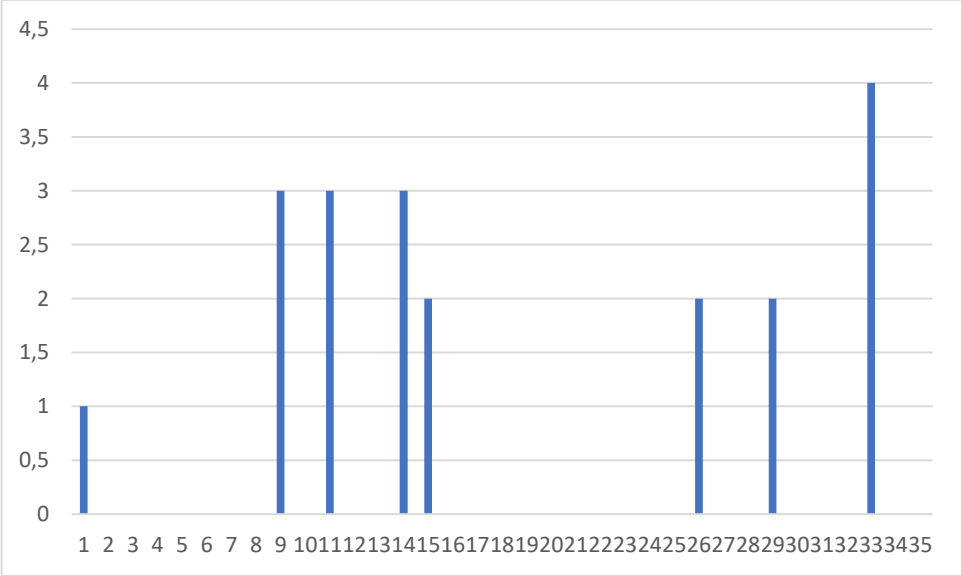
C18: Boundary Value Analysis

La Boundary Value Analysis (BVA), cioè l'analisi dei valori limite, fa parte delle tecniche di test Black Box ed è particolarmente utile quando il sistema di cui si sta valutando il funzionamento cambia il suo comportamento a seguito di ben specifici input.



Look, you've had your chance! Each time we let you in,
you are causing nothing but trouble"

8 studenti lo fanno e lo fanno anche più volte.



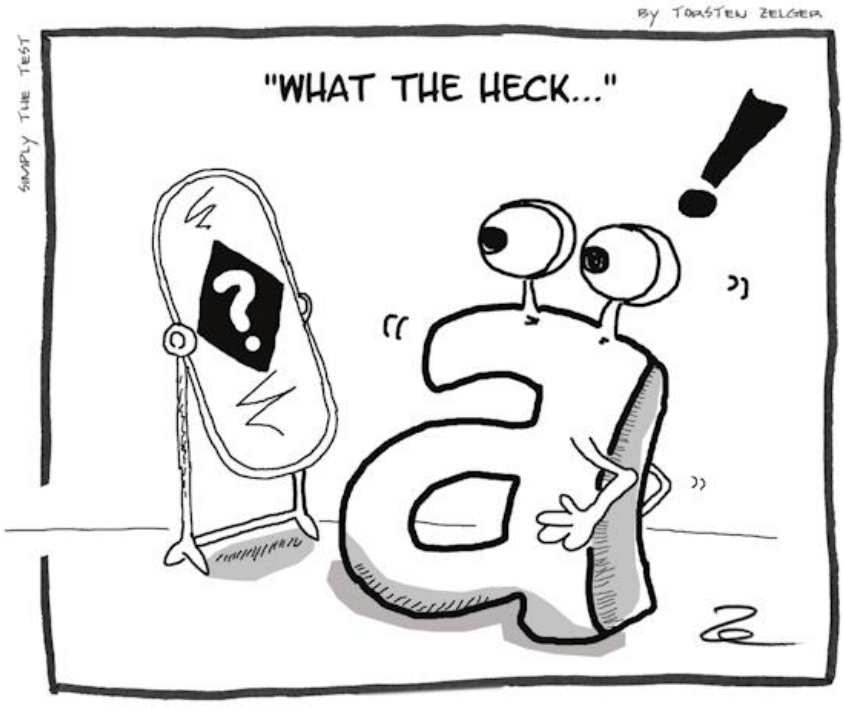
Valore medio (o media) C18: 2,5

Valore mediano C18: 2,5

Conteggio dei valori maggiori di 0 C18: 8

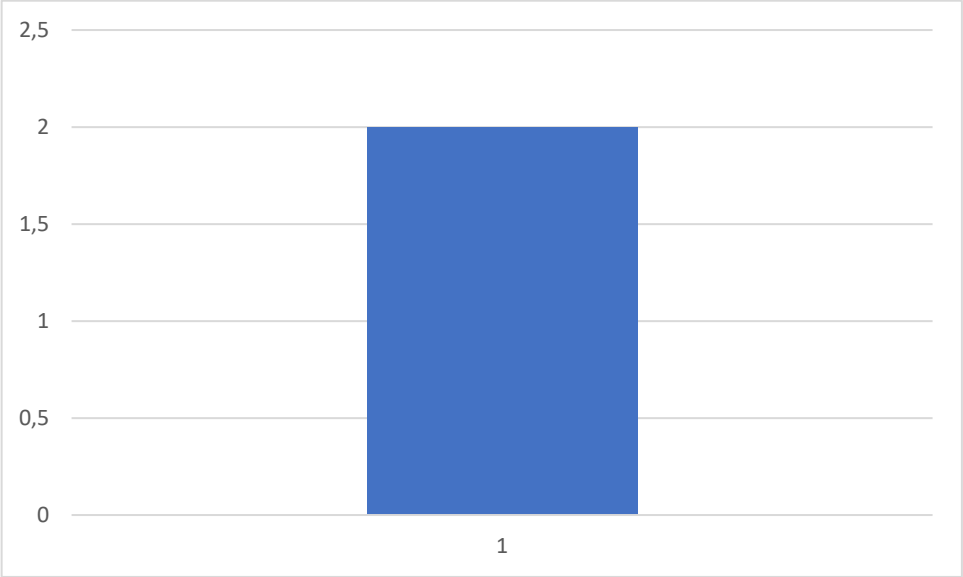
C21: Combinatorial testing

Il Combinatorial testing (o test combinatorio) nel software testing è una tecnica che si concentra sull'esplorazione efficiente di tutte le combinazioni uniche di input del sistema. L'obiettivo è identificare e testare le diverse interazioni tra variabili di input al fine di rilevare potenziali difetti o errori che possono verificarsi solo in specifiche combinazioni. Una strategia comune è il "Pairwise Testing," che mira a testare tutte le possibili coppie di valori di input per massimizzare l'efficienza del processo di testing.



All at once, the mirror did not recognize the Umlaut anymore

Soltanto uno studente, nello specifico lo studente 1, lo fa.



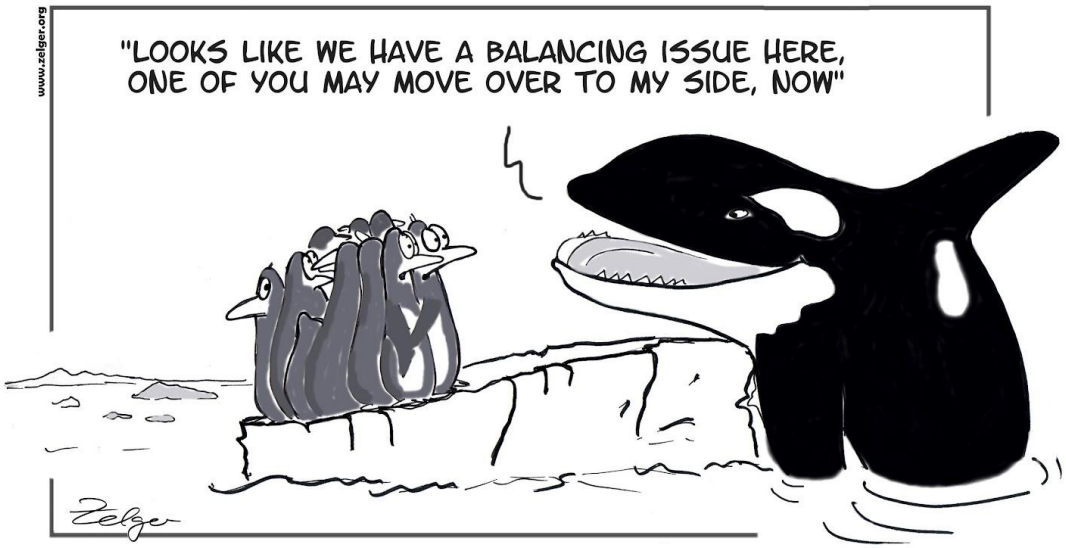
Valore medio (o media) C21: 2

Valore mediano C21: 2

Conteggio dei valori maggiori di 0 C21: 1

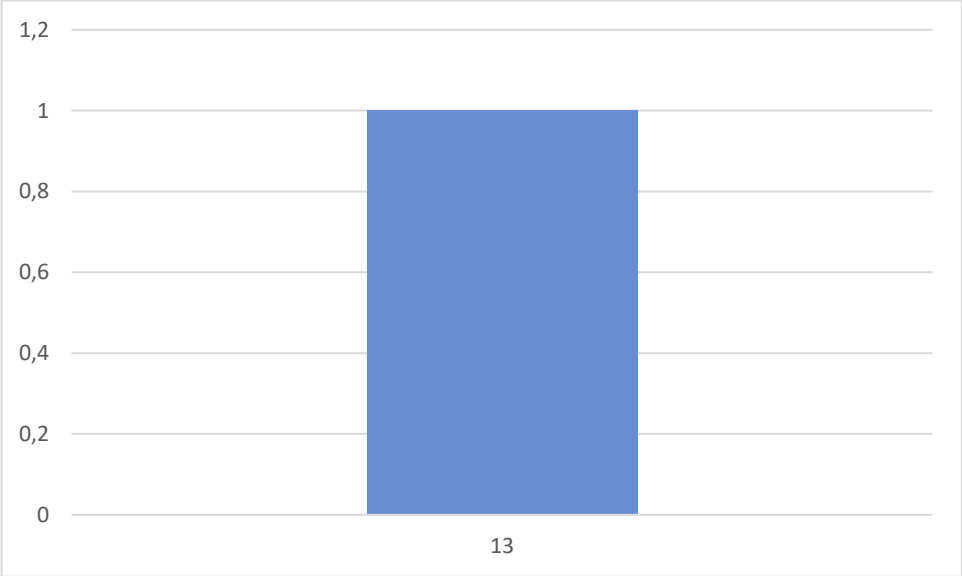
C22: Decision table testing

Una tabella decisionale, o tabella delle decisioni (conosciuta anche come Decision Table in inglese), è una rappresentazione organizzata di condizioni e azioni. Questa tabella semplifica e ordina le informazioni, presentando in modo chiaro percorsi logici alternativi di azione in diverse condizioni operative. Attraverso la visualizzazione strutturata di scelte e risultati possibili, una tabella decisionale agevola la riflessione su un problema, consentendo a un individuo di presentare in modo compatto la sua soluzione.



Everyday life at the Southern Hemisphere

Soltanto uno studente, nello specifico lo studente 13, lo fa.



Valore medio (o media) C22: 1

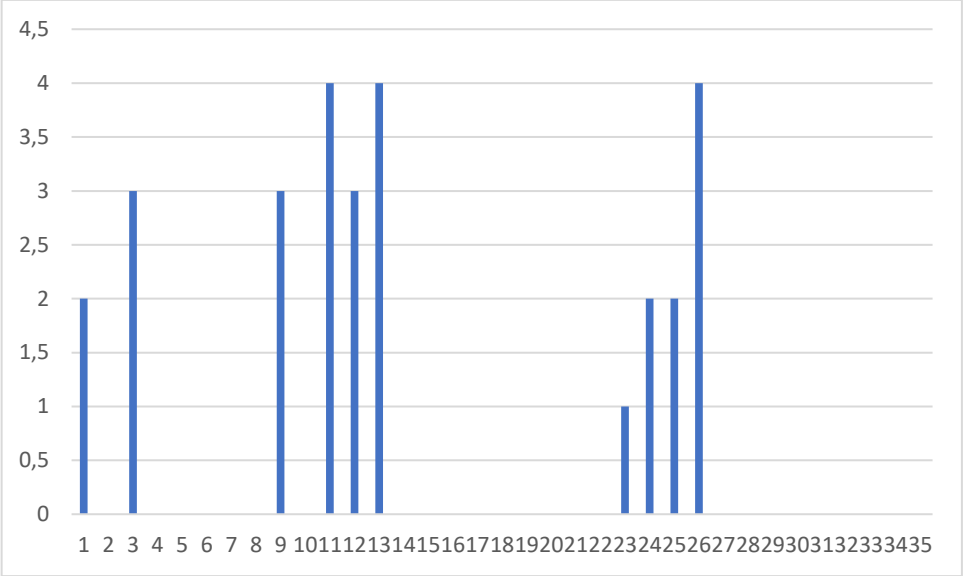
Valore mediano C22: 1

Conteggio dei valori maggiori di 0 C22: 1

C23: Equivalence Partitioning (EP)

Nell' Equivalence Partitioning (EP), facente sempre parte delle metodologie di testing Black Box, i dati inseriti nel software oggetto di test vengono suddivisi in partizioni di dimensioni omogenee. Per ciascuna partizione di dati, viene sviluppato almeno un caso di prova, mirando a esplorare in modo completo le varie classi del software. Questo approccio è finalizzato alla rilevazione di errori e bug potenziali all'interno del software. Ogni caso di test è specificamente progettato per verificare un tipo particolare di errore, contribuendo a accelerare il processo di individuazione degli errori attraverso una riduzione del numero complessivo di casi di test richiesti.

10 studenti lo fanno e lo fanno anche più volte.



Valore medio (o media) C23: 2,8

Valore mediano C23: 3

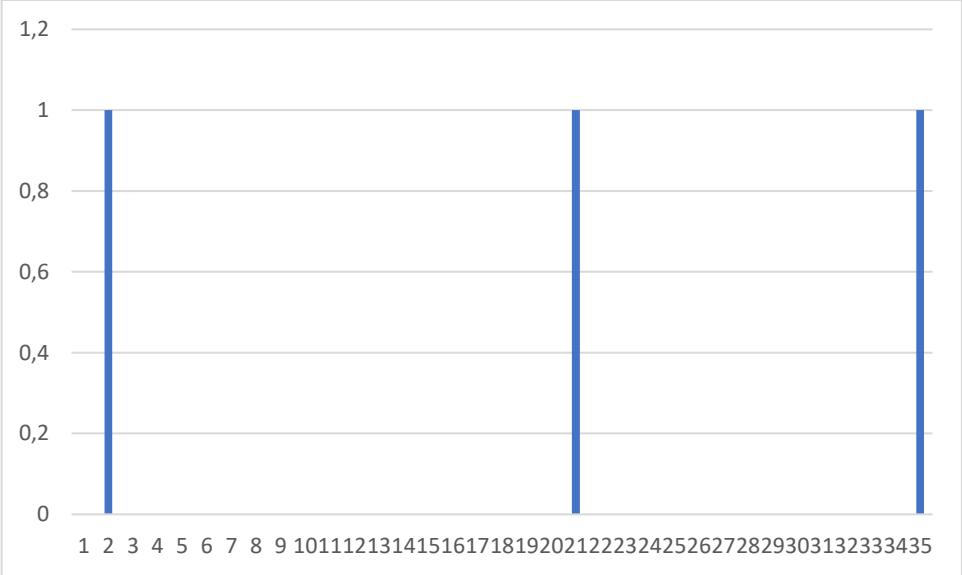
Conteggio dei valori maggiori di 0 C23: 10

C26: Requirements-based testing

Nel test basato sui requisiti, si progettano gli scenari di test prendendo come riferimento gli obiettivi del test e le condizioni che emergono dai requisiti. Questo processo mira a conseguire due obiettivi principali:

1. Verificare la validità, la completezza, l'assenza di ambiguità e la coerenza logica dei requisiti.
2. Definire un insieme minimo di scenari di test che garantisca la piena conformità del progetto e del codice ai requisiti stabiliti.

La maggior parte degli studenti procede con un'analisi dei requisiti comunque completa e varia, quindi in questo caso ho osservato studenti che si sono limitati ad elencare i requisiti, senza scendere in maggior dettaglio (come la maggior parte degli altri) nella progettazione di casi di test per gli scenari specifici.



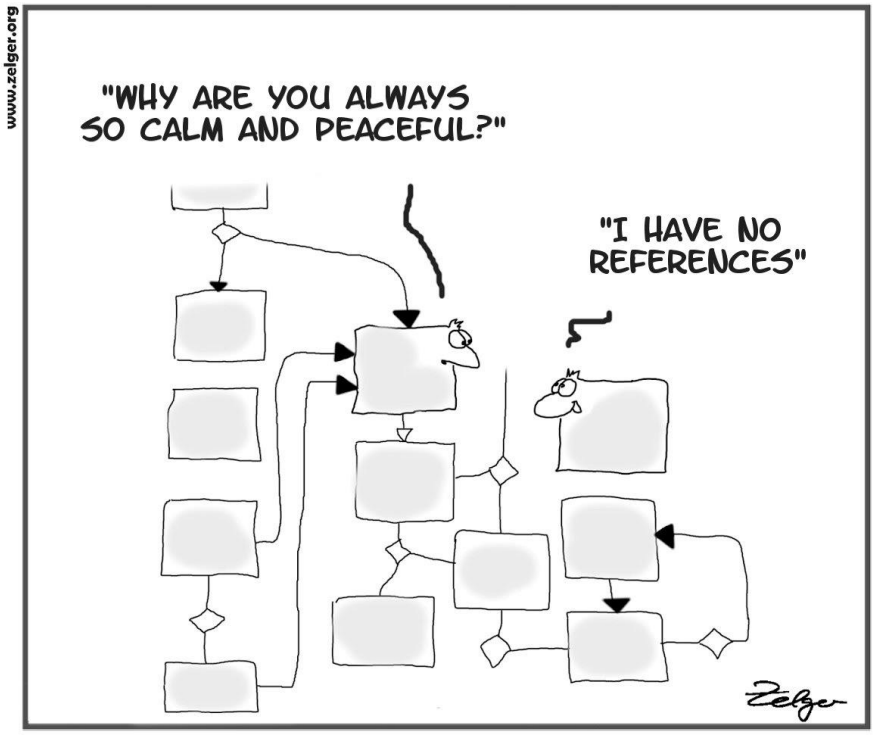
Valore medio (o media) C26: 1

Valore mediano C26: 1

Conteggio dei valori maggiori di 0 C26: 3

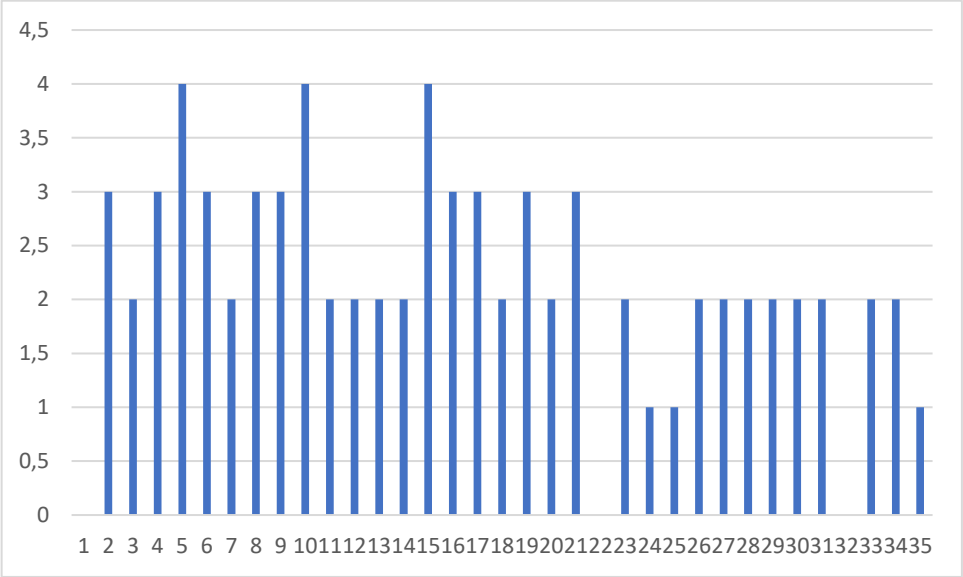
C27: Scenario testing

Lo Scenario testing viene impiegato per ottenere una visione completa del funzionamento complessivo del software. Un caso di test consiste in un insieme di istruzioni o passaggi eseguiti per verificare il comportamento di una specifica caratteristica o funzionalità di un'applicazione software. Diversamente, uno scenario di test rappresenta una descrizione di alto livello di una situazione di test specifica è cioè una descrizione più generale di ciò che deve essere testato.



FUNCTION TALKS

Viene fatto da quasi tutti gli studenti.



Valore medio (o media) C27: 2,375

Valore mediano C27: 2

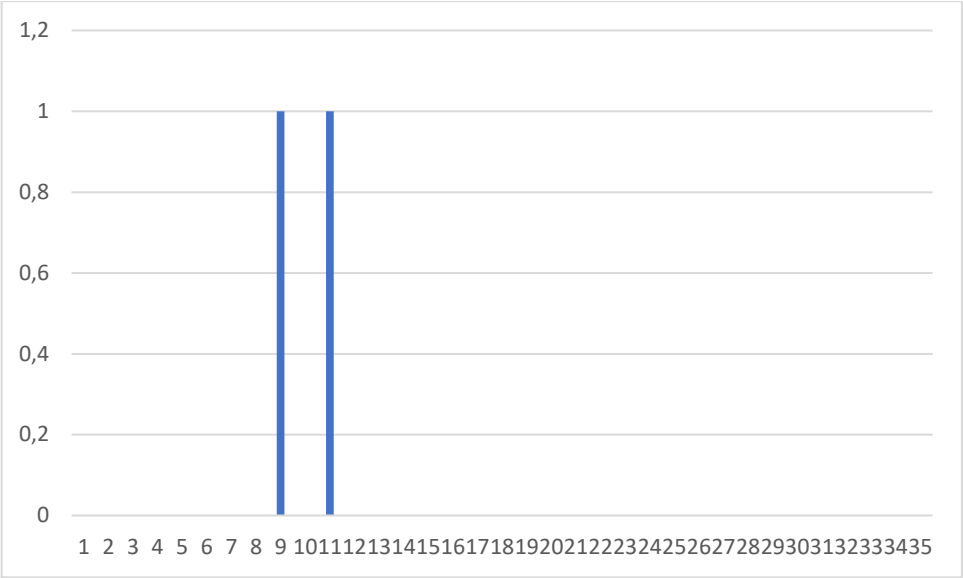
Conteggio dei valori maggiori di 0 C27: 32

C33: Branch testing

I Branch testing o test di copertura del ramo fanno parte delle tecniche di test white box.

I Branch testing assicurano che ogni ramo di un'applicazione sia coperto dai test unitari. Questo assicura che anche i percorsi di codice poco utilizzati siano convalidati correttamente.

La realizzazione di un test di copertura completa dei rami è cruciale poiché contribuisce a ottenere una copertura più dettagliata del codice e facilita l'individuazione di difetti legati alle decisioni del programma. Tuttavia, è importante notare che una copertura completa dei rami non implica automaticamente una copertura completa dei casi di test, e potrebbero essere necessarie altre tecniche di testing per garantire una copertura funzionale completa del software.



Valore medio (o media) C33: 1

Valore mediano C33: 1

Conteggio dei valori maggiori di 0 C33: 2

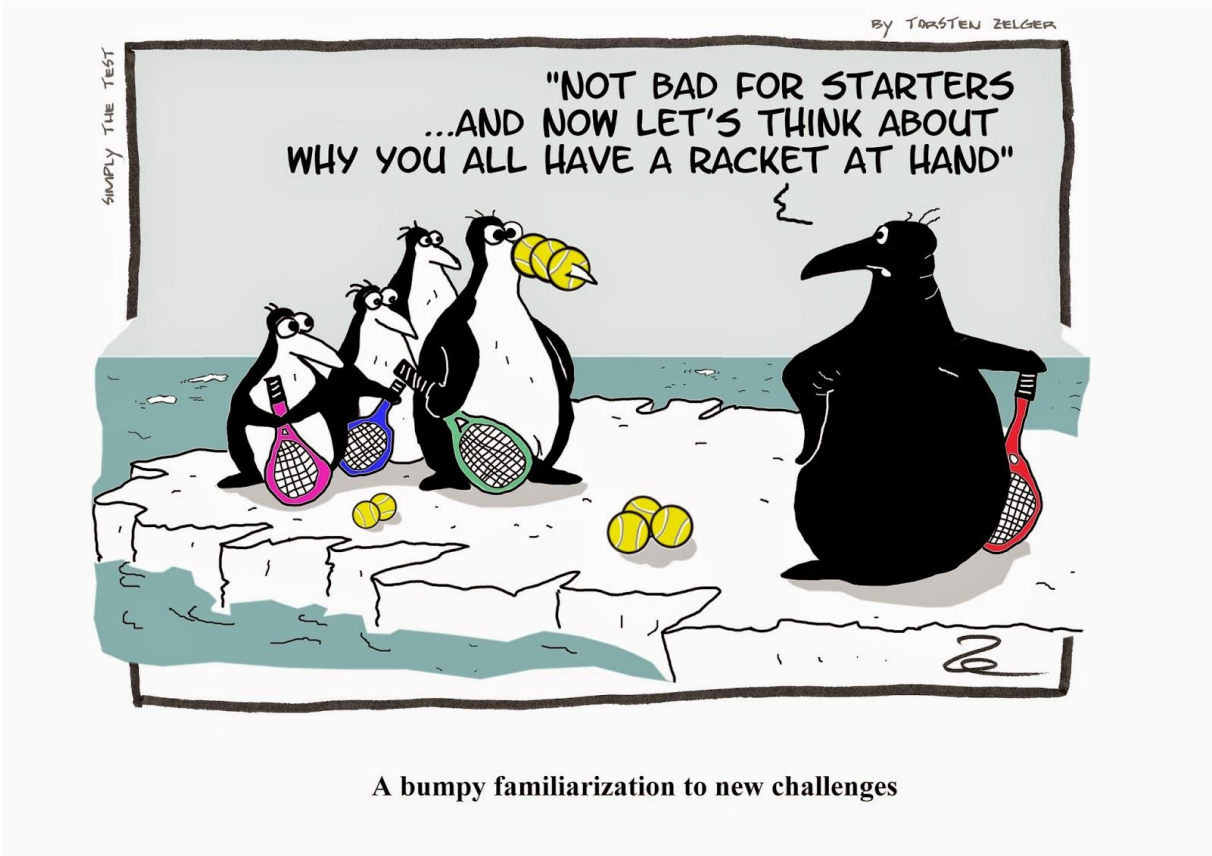
C36: Error guessing

L' Error guessing è una tecnica utilizzata per anticipare il verificarsi di errori, difetti e guasti, sulla base delle conoscenze del tester, che include:

• Come ha funzionato l'applicazione in passato

• Quali tipi di errori si tendono a commettere

• Guasti che si sono verificati in altre applicazioni

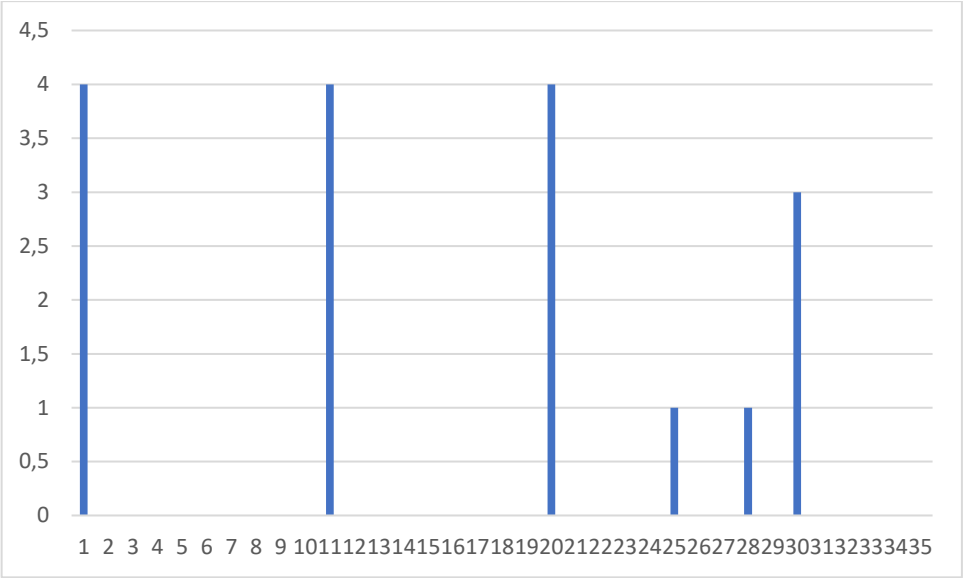


A bumpy familiarization to new challenges

Un approccio metodico alla tecnica di Error guessing consiste nel creare un elenco di possibili errori, difetti e fallimenti e progettare i test che genereranno questi fallimenti e i difetti che li hanno causati. Questi elenchi di errori, difetti e fallimenti possono essere creati sulla base dell'esperienza, dei dati su difetti e fallimenti, o della conoscenza diffusa delle ragioni per cui il software fallisce.

In pratica, consiste nell'effettuare (primi) test che, scommettiamo, possano mettere in difficoltà/portare al fallimento del sistema

L'efficacia della tecnica Error Guessing dipende dall'esperienza del tester.



Valore medio (o media) C36: 2,833

Valore mediano C36: 3,5

Conteggio dei valori maggiori di 0 C36: 6

C39: Statement testing o Statement (node) coverage

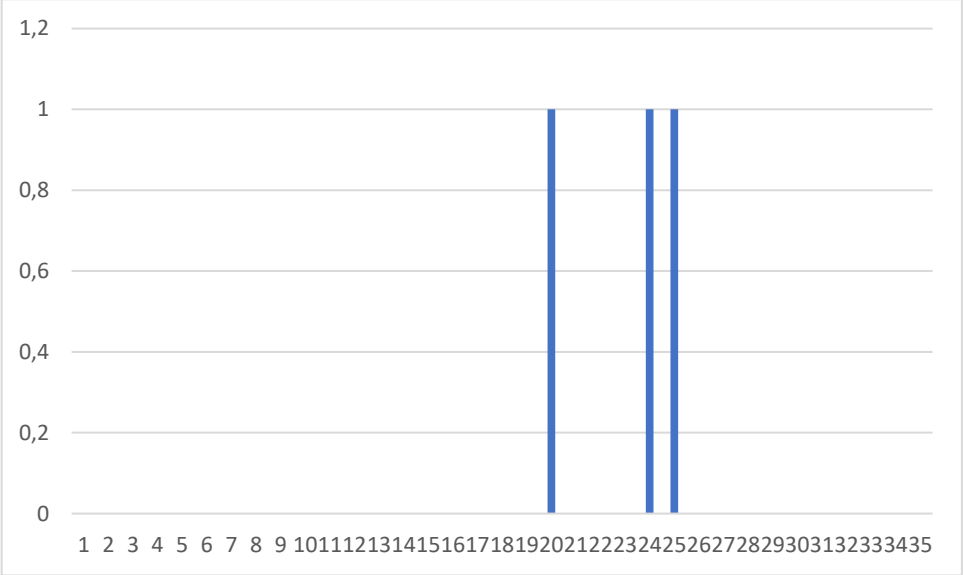
Lo Statement testing (o test delle istruzioni), facente parte dei test White Box, è una tecnica di test che richiede che ogni nodo nel grafo di flusso di controllo (CFG, Control-Flow Graph) venga eseguito almeno una volta durante i test. Il grafo di flusso di controllo rappresenta la struttura del programma attraverso i nodi che corrispondono alle istruzioni e gli archi che indicano i possibili percorsi di esecuzione.

Questa tecnica è considerata un criterio di copertura debole in quanto non assicura la copertura di entrambi i rami (true e false) di un predicato. In altre parole, mentre richiede che ogni istruzione nel programma venga eseguita almeno una volta, non garantisce che tutte le condizioni booleane siano testate sia con valori veri che falsi.

La copertura degli statement è utile per identificare le parti del codice che non sono state eseguite durante i test, ma non fornisce una copertura completa dei possibili percorsi di esecuzione attraverso il programma. Pertanto, è spesso utilizzata in combinazione con altre

tecniche di copertura, come la copertura di branch (ramo), per ottenere una valutazione più completa e accurata della qualità dei test effettuati su un software.

In questo specifico esperimento, quindi, è importante incentrare lo studio sui nodi del flow chart (se si fosse avuto il codice, lo studio si sarebbe incentrato sui comandi sulle righe di codice).



Valore medio (o media) C39: 1

Valore mediano C39: 1

Conteggio dei valori maggiori di 0 C39: 3

C49: Exploratory testing

Gli Exploratory testing (test esplorativi) hanno una lunga storia e in passato erano comunemente chiamati "test ad hoc". La denominazione formale "test esplorativo" è stata introdotta da Cem Kaner, un esperto di test del software, nel suo libro diventato un classico, Testing Computer Software.

Il test esplorativo emerge come un approccio efficace e affascinante alla tecnica Black-Box, spesso dimostrandosi più produttivo in determinate circostanze rispetto a un tradizionale test

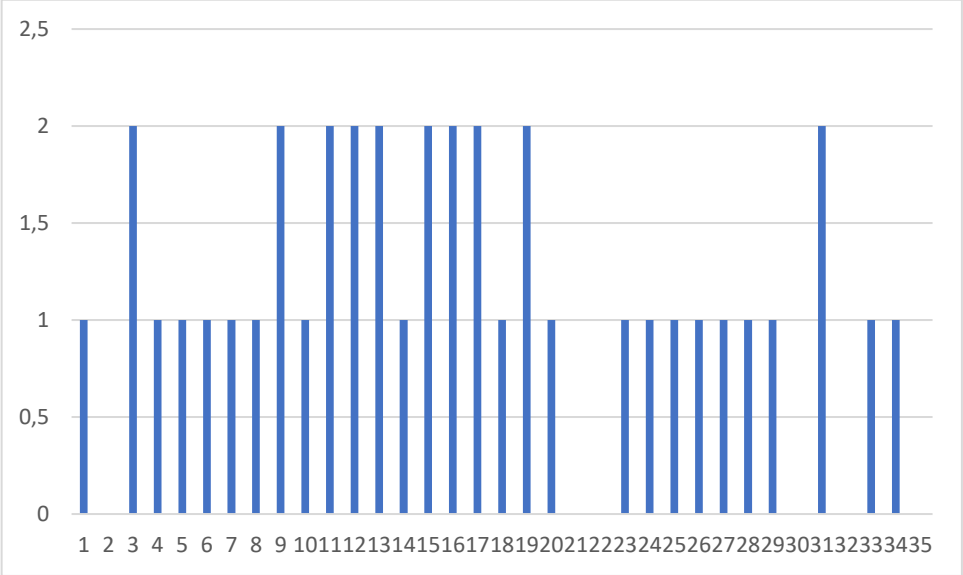
pianificato. Ogni tester, anche inconsciamente, incorpora occasionalmente il test esplorativo nella propria pratica. Tuttavia, nonostante ciò, il test esplorativo non gode di un ampio riconoscimento nel settore.

Una definizione canonica di test esplorativo lo concepisce come un'attività simultanea di progettazione e esecuzione del test. Questo si discosta dai test pianificati, che prevedono la progettazione dei casi di test prima della loro esecuzione pratica, che può avvenire manualmente o automaticamente. È evidente che non è possibile pianificare con precisione la durata e i risultati attesi di un'attività di test esplorativo. Tuttavia, un tester esplorativo esperto, basandosi sulla sua esperienza, è in grado di identificare rapidamente le idee e le strategie da adottare per testare un'applicazione specifica, contribuendo in qualche modo a definirne la durata.

Although the testers had some great testing ideas, they still struggled finding the right moment for their experiments

In generale, in questo specifico esperimento, ci si riferisce ai:

- test che vengono effettuati esplorando il sito web.
- casi nei quali lo studente progetta un po' di test a campione, senza adottare una strategia di analisi sistematica, ma facendosi guidare verso l'esplorazione delle funzionalità che evidentemente riteneva più importanti da testare



Valore medio (o media) C49: 1,345

Valore mediano C49: 1

Conteggio dei valori maggiori di 0 C49: 29

C54: Model-based testing

I Model-based testing (o test basati su modelli) sono una pratica nel campo del Software Testing che impiega modelli per guidare l'attività di testing. Questi modelli possono rappresentare diversi aspetti del sistema software, come il suo comportamento, la struttura, o i dati.

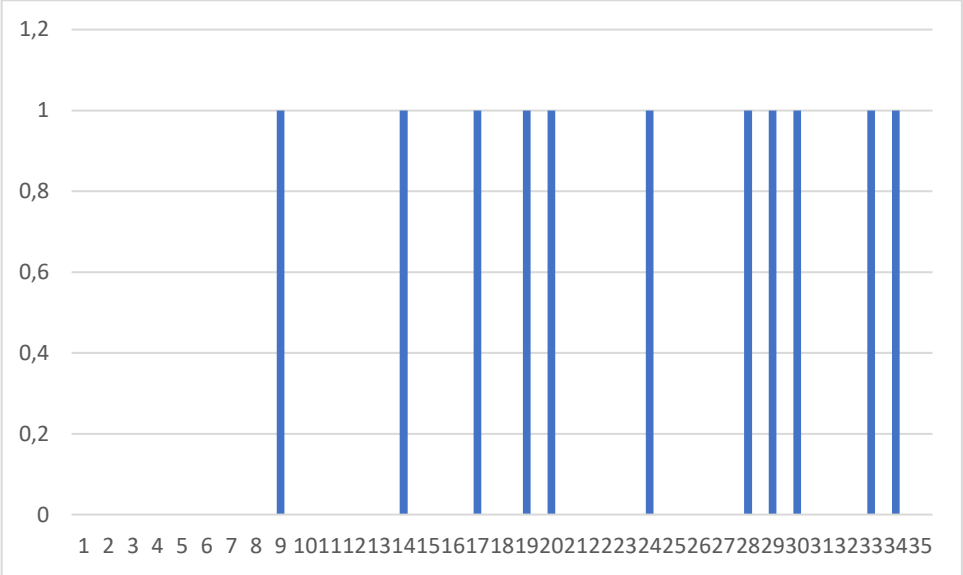
L'obiettivo principale è utilizzare questi modelli come base per la progettazione, l'esecuzione e la valutazione dei test del software.

Un test basato sul modello è quindi un'attività nella quale i test sono progettati osservando il modello e allo scopo di cercare di coprirlo in maniera esaustiva, ove possibile

In questo specifico esperimento, quindi, analizzo in maniera mirata i flow chart, in quanto unico modello a disposizione.

Questa metodologia è spesso utilizzata in progetti software complessi in cui è necessaria una comprensione dettagliata e una gestione accurata dei requisiti. L'uso di modelli contribuisce a fornire una rappresentazione più chiara e comprensibile del sistema, facilitando la progettazione e l'esecuzione dei test.

Più studenti lo fanno una singola volta.



Valore medio (o media) C54: 1

Valore mediano C54: 1

Conteggio dei valori maggiori di 0 C54: 11

CONCLUSIONI E SVILUPPI FUTURI

Tendenze generali

Approccio alle metodologie: Gli studenti sembrano adottare una varietà di approcci, da quelli guidati da scenari e classi di equivalenza a strategie più esplorative. Alcuni integrano anche l'analisi degli scenari con l'error guessing o l'esplorazione di nuovi casi dal modello.
Uso di strategie esplorative: Molti studenti hanno incorporato una strategia esplorativa nei loro test, cercando di individuare nuovi scenari o casi che potrebbero non essere stati precedentemente considerati.
Guida da modelli: Alcuni studenti sembrano essere più orientati all'utilizzo di modelli, mentre altri si affidano maggiormente all'analisi degli scenari.

Osservazioni specifiche sugli studenti

- *Studenti 1, 4, 6, 9, 13, 14, 16, 17, 20, 25, 26:* Hanno mostrato un mix di approcci, integrando sia l'analisi degli scenari che strategie esplorative.
- *Studenti 3, 7, 8, 11, 18, 21, 22, 23, 27, 28:* Si sono principalmente concentrati su approcci esplorativi, cercando di coprire scenari non precedentemente considerati.
- *Studenti 2, 5, 10, 12, 15, 19, 24, 29, 30, 31:* Hanno adottato approcci più guidati dagli scenari, concentrandosi su un'analisi dettagliata.

Suggerimenti

- Incorporare una varietà di approcci può essere vantaggioso. L'analisi degli scenari offre una base solida, ma l'esplorazione può portare a scoperte significative.
- Un'attenzione particolare dovrebbe essere posta sull'uso di strategie esplorative, specialmente quando si cercano di individuare scenari non precedentemente identificati.
- L'integrazione di modelli e analisi degli scenari può massimizzare la copertura del test e garantire un approccio completo.
- Promuovere il confronto tra studenti che hanno adottato strategie diverse può favorire la condivisione di conoscenze e best practices.

In generale, il processo di testing sembra riflettere una combinazione di metodologie strutturate e approcci più flessibili, il che è positivo nel garantire una copertura completa e creativa durante lo sviluppo del software.

Conclusioni

L’esperimento rappresenta un ottimo punto di partenza ed un’ottima analisi quantitativa e qualitativa riguardo all’approccio degli Ingegneri Federiciani al testing.

Alla luce dei risultati emersi dagli esperimenti condotti al fine di supportare l'analisi del modello cognitivo degli studenti durante la progettazione di casi di test, è possibile trarre alcune conclusioni di rilievo.

In primo luogo, l'approccio di esaminare le strategie mentali degli studenti durante le attività di testing ha fornito preziose intuizioni sulle dinamiche cognitive coinvolte in questo processo. L'analisi dettagliata dei dati ha consentito di identificare differenze significative nelle strategie adottate dagli studenti rispetto ai modelli mentali dei tester esperti.

In secondo luogo, le sfide e le lacune di competenza riscontrate negli studenti durante le attività di testing hanno evidenziato aree specifiche che richiedono un'attenzione particolare nell'insegnamento del testing software. Queste informazioni potrebbero informare la progettazione di futuri programmi di formazione, mirati a colmare tali lacune e migliorare le competenze degli studenti nell'ambito del testing.

Inoltre, l'analisi dei dati aggregati ha permesso di sviluppare un questionario strutturato, il quale potrebbe rivelarsi uno strumento utile per indagini su scala più ampia, estendendo così l'applicabilità delle conclusioni ottenute.

Infine, le intuizioni iniziali sul modello cognitivo degli studenti, derivanti dalla fase di analisi, costituiscono la base per ulteriori ricerche e approfondimenti nel campo del testing cognitivo. Questo studio pone le fondamenta per futuri sviluppi nell'educazione al testing software e nella comprensione dei processi mentali coinvolti nella progettazione dei casi di test.

In sintesi, questa ricerca fornisce una contribuzione significativa alla comprensione del modello cognitivo degli studenti nel contesto del testing software, offrendo spunti utili per il miglioramento delle pratiche didattiche e aprendo nuove prospettive per la ricerca futura in questo settore.

Implicazioni e Prospettive Future

- Orientamento alla Formazione: Le evidenze raccolte possono guidare la progettazione di programmi di formazione specifici per colmare le lacune identificate nelle competenze degli studenti in materia di testing software.
- Contributo alla Ricerca: La sintesi dei risultati fornisce una base solida per ulteriori ricerche nel campo del testing cognitivo. Gli spunti emersi possono orientare future indagini sulle diverse metodologie di progettazione dei casi di test.

Contributi alla Ricerca nel Campo del Testing Software

La ricerca condotta ha generato contributi significativi che non solo influenzano la transizione degli studenti verso il mondo del lavoro nel settore del testing software, ma offrono anche spunti per possibili miglioramenti delle pratiche di esecuzione dei test da parte degli sviluppatori e per future indagini nell'ambito.

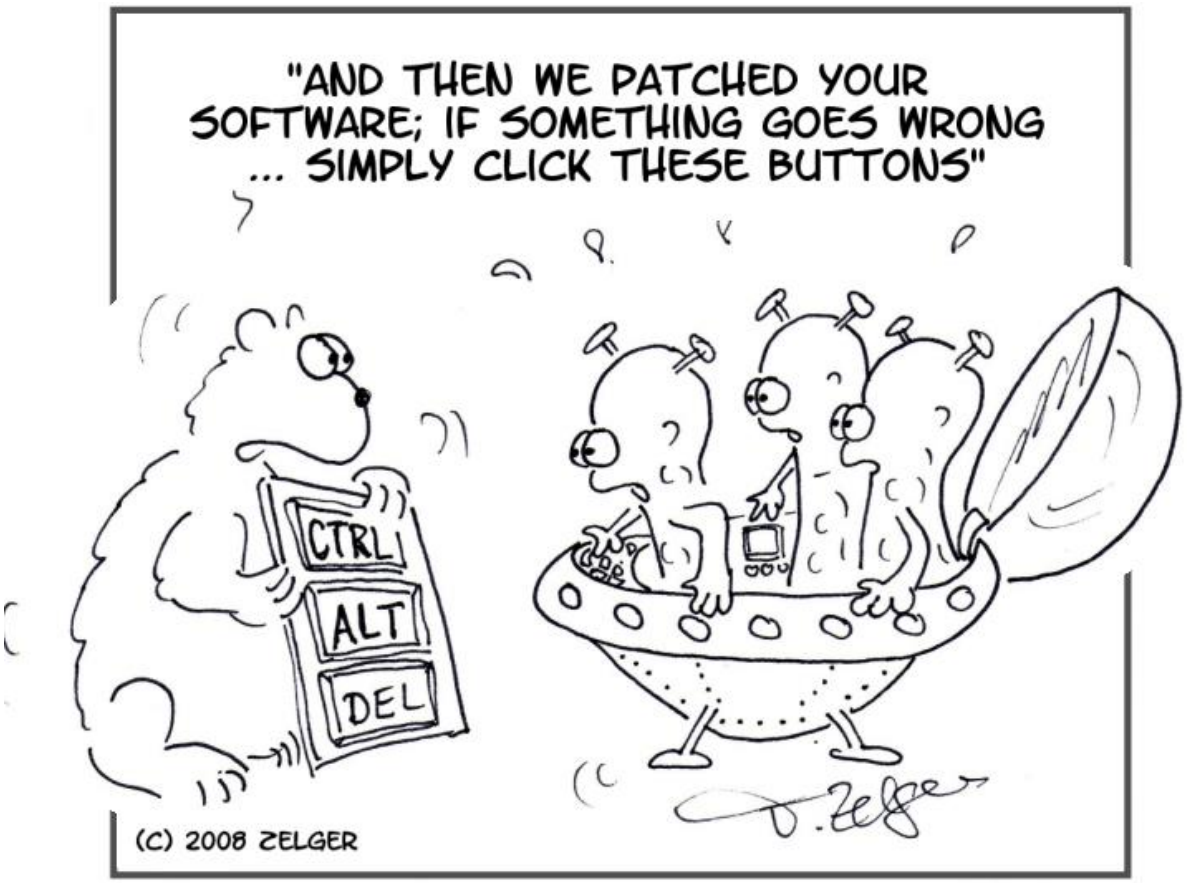
Impatto sulla Transizione verso il Mondo del Lavoro: Gli esiti della ricerca non solo forniscono una comprensione approfondita delle strategie cognitive degli studenti durante la progettazione dei casi di test, ma hanno anche un impatto diretto sulla preparazione degli studenti per il mondo del lavoro. L'identificazione di differenze chiave nelle strategie mentali offre agli studenti un insight prezioso per affrontare sfide professionali e svilupparsi in modo più efficace nel settore del testing software.

Possibili Implementazioni nell'Insegnamento e nella Formazione: Oltre alla progettazione di programmi didattici, la ricerca suggerisce l'introduzione di strumenti specifici per migliorare le pratiche di esecuzione dei test da parte degli sviluppatori. L'implementazione di ambienti di sviluppo integrati con funzionalità di testing automatizzato e la promozione di metodologie di test più agili potrebbero favorire una più rapida e efficiente esecuzione dei test durante il ciclo di sviluppo del software.

Possibili Strumenti per Migliorare le Pratiche di Esecuzione dei Test: L'integrazione di strumenti avanzati di testing automatico e la promozione di pratiche di Continuous Testing possono facilitare una maggiore automazione e riduzione dei tempi di esecuzione dei test. L'utilizzo di framework di test avanzati, come JUnit o Selenium, può migliorare l'efficienza e la copertura dei test, contribuendo così a garantire la qualità del software in modo più robusto.

Indagini Future sull'Argomento: Le ricerche future potrebbero concentrarsi sull'analisi approfondita delle migliori pratiche di esecuzione dei test nell'ambito dello sviluppo software. Investigare ulteriormente l'impatto di nuovi strumenti e metodologie potrebbe fornire indicazioni preziose per migliorare ulteriormente l'efficacia delle attività di testing. Esplorare le dinamiche cognitive in contesti più complessi e realistici potrebbe arricchire ulteriormente la nostra comprensione delle strategie mentali coinvolte nella progettazione dei casi di test.

In conclusione, questa ricerca offre un punto di partenza per implementazioni pratiche nel settore dell'insegnamento e della formazione, insieme a suggerimenti concreti per strumenti che potrebbero migliorare le pratiche di esecuzione dei test, oltre a costituire una base solida per future indagini nel campo del Testing Software.



Software made on Earth