

Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica

A.a. 2025/2026

Sessione di laurea Settembre 2026

Aggiunta di Meccanismi di Gamification in uno Strumento di Test Esplorativo per App Mobili

Relatori:
Marco Torchiano
Lorenzo Laudadio

Candidato:
Alessandro Costantino

Sommario

Nel panorama contemporaneo della digitalizzazione, il software mobile riveste un ruolo infrastrutturale cruciale, rendendo il testing delle interfacce grafiche utente un processo fondamentale per garantire l'affidabilità delle applicazioni. Sebbene l'automazione sia ampiamente adottata, il collaudo mobile dipende ancora fortemente dal testing esplorativo manuale, un approccio black-box ed end-to-end guidato dall'intuito umano che eccelle nell'individuare difetti complessi di usabilità. Tuttavia, sessioni prolungate di testing manuale tendono a diventare ripetitive e alienanti, riducendo la motivazione e la soglia di attenzione del collaudatore. Per mitigare questo declino e stimolare la produttività, la ricerca e l'industria si stanno orientando verso la Gamification, applicando meccaniche di game design a contesti non ludici.

Questo lavoro di tesi presenta la progettazione, l'implementazione e la valutazione sperimentale di GAppium+, un'estensione "gamificata" originale dello strumento di test esplorativo Appium Inspector. Basandosi sul modello teorico dell'Octalysis Framework, GAppium+ arricchisce l'esperienza di collaudo attraverso un insieme di meccaniche ludiche consolidate, tra cui trofei, classifiche, livelli, tracciamento della copertura dell'interfaccia o personalizzazione del profilo utente. Da un punto di vista più tecnico, l'estensione utilizza algoritmi per una gestione selettiva dei widget interattivi dell'interfaccia e sfrutta l'Indice di Jaccard per misurare la similarità strutturale tra le schermate esplorate.

Inoltre, l'efficacia del sistema è stata testata con una valutazione preliminare in-vivo su un campione di dottorandi e assegnisti di ricerca, i quali hanno testato l'applicativo in condizioni controllate. I risultati dei questionari post-sessione indicano un'accoglienza generalmente positiva: l'adozione della gamification nel testing esplorativo è stata ampiamente condivisa e considerata un approccio vantaggioso. Nello specifico, il tracciamento della copertura e i trofei si sono rivelati le meccaniche di maggior successo e con il più alto impatto motivazionale.

Nonostante i risultati incoraggianti, il lavoro ha anche evidenziato limiti che potrebbero guidare i futuri sviluppi verso una riprogettazione dell'interfaccia per facilitarne l'utilizzo e verso l'integrazione di ulteriori meccaniche di gamification.

Ringraziamenti

Esprimo la mia più profonda gratitudine a tutti coloro che mi hanno supportato durante questo percorso universitario.

Ringrazio il mio relatore, il prof.Marco Torchiano, per la sua disponibilità e per il suo supporto durante la stesura della tesi. Un ulteriore ringraziamento va a Lorenzo Laudadio, per l'enorme disponibilità e, in particolar modo, per il supporto dimostrato durante lo sviluppo e la valutazione sperimentale del software.

Ringrazio infinitamente la mia famiglia, in particolare mia madre Claudia, mio padre Domenico, e mia nonna Nicolina per tutti i sacrifici che hanno fatto. A loro dedico questo traguardo e i futuri. Grazie per essermi stati vicino, per avermi supportato e sopportato nonostante le tante difficoltà, per avermi sempre spronato a cercare di meglio e di non accontentarmi mai, e per tutto l'amore che mi dimostrate ogni giorno.

Un ringraziamento speciale va a tutti i miei più cari amici, che sono rimasti al mio fianco nonostante la lontananza causata da questo percorso di studi, e ai miei colleghi universitari che, condividendo con me sfide complesse e grandi soddisfazioni, hanno reso indimenticabile questa esperienza. In particolare, ringrazio Domenico e Gaetano, compagni di viaggio e coinquilini fantastici, ma anche Giorgio, Davide e Giuseppe. Insieme abbiamo condiviso la maggior parte delle giornate di studio, e li ringrazio di cuore per averle rese più semplici e, sicuramente, più divertenti.

Infine, voglio ringraziare tutte le persone con le quali ho condiviso una parte di me e del mio percorso, e che hanno contribuito alla mia crescita personale.

Indice

Elenco delle figure v

1 Introduzione 1
1.1 Obiettivo . . . 2
1.2 Struttura della tesi . . . 3

2 Background 4
2.1 Software Testing . . . 4
2.1.1 Piramide del Testing . . . 5
2.1.2 Verifica e Validazione dei test . . . 7
2.2 Mobile Testing . . . 8
2.2.1 Frammentazione e instabilità . . . 8
2.2.2 Tecniche e strumenti . . . 10
2.3 Exploratory Testing . . . 12
2.4 Octalysis Framework . . . 14
2.4.1 Epic Meaning & Calling . . . 15
2.4.2 Development & Accomplishment . . . 16
2.4.3 Empowerment of Creativity & Feedback . . . 16
2.4.4 Ownership & Possession . . . 17
2.4.5 Social Influence & Relatedness . . . 18
2.4.6 Scarcity & Impatience . . . 19
2.4.7 Unpredictability & Curiosity . . . 19
2.4.8 Loss & Avoidance . . . 20
2.4.9 Left & Right Brain . . . 21
2.4.10 White & Black Hat . . . 21

3 Strumenti utilizzati 23
3.1 Stack tecnologico . . . 23
3.1.1 JavaScript . . . 23
3.1.2 React . . . 24
3.1.3 Npm . . . 25

3.2 Appium . . . 26
3.2.1 Architettura . . . 26
3.2.2 Client WebDriverIO . . . 27
3.2.3 Automation Drivers . . . 28
3.3 Appium Inspector . . . 30
3.3.1 Session Builder . . . 30
3.3.2 Session Inspector . . . 31
3.4 GAppium . . . 36

4 Descrizione del progetto 38
4.1 Quality of life . . . 39
4.1.1 Gestione dei widgets interattivi . . . 39
4.1.2 Algoritmo di similarità delle pagine . . . 41
4.1.3 Scelte in evidenza . . . 47
4.2 Livelli, punteggi e feedback . . . 49
4.3 Profilo utente . . . 50
4.4 Badges e funzionalità sbloccabili . . . 51
4.5 Classifica . . . 53
4.6 Gestione dell'impazienza e *FOMO* . . . 54
4.7 Imprevedibilità delle ricompense . . . 57
4.8 Moltiplicatori di punteggio . . . 59
4.9 Conclusione e copertura Octalysis . . . 63

5 Risultati e Valutazione 65
5.1 Struttura del questionario . . . 66
5.2 Analisi dei risultati . . . 68

6 Conclusioni 71
6.1 Limiti e sviluppi futuri . . . 72

Bibliografia 74

Elenco delle figure

2.1 Piramide del Testing [10] 6
2.2 Verifica e Validazione nel software testing [11] 7
2.3 Frammentazione dei sistemi operativi Android [13] 9
2.4 Panoramica di alcuni AIG esistenti per Android [14] 11
2.5 Octalysis Framework [19] 22

3.1 Vantaggi del framework React [33] 25
3.2 Architettura di Appium [35] 27
3.3 Spec Reporter di WebDriverIO [39] 29
3.4 Allure Reporter di WebDriverIO [40] 29
3.5 Session Builder di Appium Inspector 31
3.6 Header del Session Inspector di Appium Inspector 32
3.7 Screenshot Panel del Session Inspector di Appium Inspector 33
3.8 Source Tab del Session Inspector di Appium Inspector 33
3.9 Commands Tab del Session Inspector di Appium Inspector 34
3.10 Gestures Tab del Session Inspector di Appium Inspector 34
3.11 Recorder Tab del Session Inspector di Appium Inspector 35
3.12 Session Information Tab del Session Inspector di Appium Inspector 35
3.13 Overview della Gamification Tab di GAppium 36
3.14 Valutazione di GAppium basata sull'Octalysis Framework 37

4.1 Overview della Gamification Tab nella nuova estensione realizzata . 38
4.2 Funzioni di gestione dei widgets interattivi 41
4.3 Indice di Jaccard 42
4.4 Funzioni di identificazione dei widgets 44
4.5 Funzione di calcolo della similarità 44
4.6 Funzioni di identificazione delle pagine 45
4.7 Funzione principale dell'algoritmo di similarità 46
4.8 Funzionalità *Scelte in evidenza* attivata 47
4.9 Funzione di rendering per la funzionalità *Scelte in evidenza* 48
4.10 Sezione di gestione del profilo utente 50

4.11 Funzionalità di rivelazione dei badge non ancora ottenuti disattivata/attivata . . . 52
4.12 Parte della funzione di gestione della coverage delle pagine . . . 54
4.13 Parte della funzione di gestione dei livelli . . . 55
4.14 Notifica contestuale basata sulla coverage della pagina . . . 56
4.15 Funzione di gestione delle ricompense bonus al passaggio di livello . 58
4.16 Funzione di gestione delle ricompense bonus al completamento della coverage . . . 59
4.17 Notifica animata di assegnazione del bonus casuale . . . 59
4.18 Funzione di schedule per determinare quando si attiverà il booster . 60
4.19 Funzione di attivazione del booster . . . 60
4.20 Funzione di gestione del tempo trascorso nella sessione corrente, utilizzata anche per attivare il booster . . . 61
4.21 Funzione di gestione del booster nell'interfaccia utente . . . 61
4.22 Booster attivato . . . 62
4.23 Valutazione di GAppium+ basata sull'Octalysis Framework . . . 64

5.1 Background accademici dei partecipanti . . . 65
5.2 Esperienza dei partecipanti nello sviluppo mobile (a sinistra) e nel software testing (a destra) . . . 66
5.3 Distribuzione delle risposte Likert sulle domande SUS . . . 68
5.4 Distribuzione delle risposte Likert sulle domande di gamification . . 69
5.5 Distribuzione delle risposte Likert sulle domande relative alle meccaniche di gamification . . . 69

Capitolo 1

Introduzione

Il software è diventato una componente essenziale della vita moderna, supportando l'intera infrastruttura della società contemporanea. Gran parte delle nostre professioni è oggi automatizzata e, di conseguenza, facilitata da programmi che, in questo scenario di digitalizzazione crescente, diventano sempre più complessi per potersi adattare a ogni contesto e supportare un ventaglio di attività in continua espansione.

Tuttavia, questa crescita esponenziale in termini di complessità determina inevitabilmente un aumento delle potenziali anomalie legate allo sviluppo del codice. Per garantire il corretto funzionamento delle applicazioni e assicurare un'alta affidabilità, la fase di *software testing* risulta un processo imprescindibile. Essa permette di individuare e mitigare gli errori durante il ciclo di vita dello sviluppo, elevando la qualità del prodotto finale e consolidando la fiducia del cliente verso l'azienda. Sebbene tale attività richieda un investimento significativo in termini di tempo e risorse, la letteratura e la storia dell'industria informatica dimostrano come l'assenza di test, o una loro gestione superficiale, possa portare a conseguenze disastrose, non solo sul piano economico, ma, in alcuni casi, anche su quello della sicurezza [1].

Nel panorama odierno, il testing assume un'importanza ancora più critica nel settore dei dispositivi mobile, che rappresentano lo strumento di accesso principale ai servizi digitali per la maggior parte degli utenti. In un'applicazione mobile, l'*Interfaccia Grafica Utente* (GUI) e l'*Esperienza Utente* (UX) sono fattori determinanti che spesso sanciscono il successo o il fallimento dell'applicazione stessa sul mercato.

Quindi nasce il *GUI Testing*, una pratica fondamentale, ma ricca di sfide.
L'automazione dei test delle interfacce utente, pur essendo necessaria nei progetti di grandi dimensioni per velocizzare lo sviluppo e ridurre l'errore umano, soffre

spesso di problematiche dovute alla miriade di dispositivi mobile esistenti, con hardware e sistemi operativi eterogenei, unita alla rapidità con cui le interfacce vengono aggiornate [2].

Per superare i limiti della pura automazione, una delle tecniche più efficaci e adottate è il *Testing Esplorativo* della GUI. Questo approccio si basa sulla creatività, l'intuizione e l'esperienza umana dei tester, i quali esplorano liberamente l'applicazione in fase di collaudo (*Application Under Test* - AUT) attraverso la sua interfaccia. Si tratta tipicamente di test black-box ed end-to-end: il tester non ha accesso al codice sorgente e non ha una conoscenza dettagliata dell'architettura di sistema, quindi si concentra sui requisiti funzionali, sui comportamenti del software e persino sulle sue prestazioni, simulando a tutti gli effetti le azioni dell'utente finale [3].

Tuttavia, nonostante la sua innegabile efficacia nel trovare errori critici che gli script automatizzati potrebbero ignorare, il testing esplorativo nasconde un'insidia significativa: l'esecuzione prolungata e sistematica di queste verifiche manuali può trasformarsi rapidamente in un'attività alienante, noiosa e ripetitiva, causando cali di attenzione e compromettendo l'efficacia stessa del processo [4].
Una soluzione innovativa e promettente a questo problema è rappresentata dalla *Gamification*, ovvero l'applicazione di elementi, meccaniche e dinamiche tipiche del game design all'interno di contesti non ludici [5].
I recenti progressi teorici e pratici nell'ambito della gamification hanno definito strumenti specifici, quali l'assegnazione di punteggi, la creazione di classifiche, il conseguimento di obiettivi e l'introduzione di dinamiche competitive o cooperative. L'adozione di questi meccanismi ha suscitato un crescente interesse nel campo dell'ingegneria del software: la letteratura dimostra infatti che l'introduzione di elementi ludici è in grado di trasformare compiti ripetitivi in sfide coinvolgenti, aumentando significativamente la motivazione intrinseca e, di conseguenza, le prestazioni operative di sviluppatori e tester [6].

1.1 Obiettivo

Partendo da queste premesse, il presente lavoro di tesi si pone l'obiettivo di progettare, implementare e valutare empiricamente un approccio "gamificato" applicato specificamente al testing esplorativo delle interfacce grafiche utente in ambito mobile.
Il fulcro pratico della ricerca consiste nell'estensione di *GAppium*, un prototipo originale che consente di implementare meccanismi di gamification all'interno di *Appium Inspector*, un noto strumento di test esplorativi per applicazioni mobile

basato sul consolidato framework open-source *Appium* [7]. Attraverso l'integrazione di specifiche meccaniche ludiche all'interno del flusso di lavoro dell'applicazione, lo studio mira a dimostrare come sia possibile incrementare la motivazione dei tester, migliorando al contempo la qualità e la quantità dei difetti individuati durante le sessioni di testing esplorativo.

1.2 Struttura della tesi

La tesi è strutturata nei seguenti capitoli:

- **Background**: introduce lo stato dell'arte relativo al software testing, approfondendo in particolare quello dell'interfaccia utente nel contesto mobile, e i fondamenti teorici e pratici della gamification.
- **Strumenti utilizzati**: analizza le tecnologie utilizzate in questo studio, con un'analisi approfondita sul framework Appium e sull'architettura originale dello strumento GAppium.
- **Descrizione del progetto**: descrive nel dettaglio la progettazione e l'implementazione dell'estensione "gamificata" realizzata, illustrando e motivando le meccaniche introdotte.
- **Risultati e Valutazione**: presenta i risultati e una valutazione sperimentale dello strumento.
- **Conclusioni**: riassume il lavoro svolto, ne discute i limiti e i possibili miglioramenti e sviluppi futuri.

Capitolo 2

Background

L'obiettivo di questo capitolo è discutere del software testing, in particolare del testing delle applicazioni mobile, e dei principi teorici e applicativi della gamification, approfondendo l'*Octalysis Framework*, ideato da Yu-kai Chou [8].

2.1 Software Testing

Il collaudo del software, o software testing, costituisce un pilastro imprescindibile all'interno del ciclo di vita dello sviluppo applicativo (*Software Development Life Cycle* - SDLC). Il suo obiettivo primario consiste nell'accertare che il prodotto sia rigorosamente allineato ai vincoli progettuali e architetturali definiti a monte. Attraverso l'identificazione e la successiva risoluzione delle anomalie, il testing mira a minimizzare drasticamente il rischio che il software rilasciato in ambiente di produzione presenti criticità operative.
Oltre a garantire la correttezza tecnica, questa fase assume anche una valenza strategica per le aziende: individuare tempestivamente le vulnerabilità abbatte i costi di manutenzione e sviluppo, determinando il successo commerciale del prodotto e garantendo un elevato grado di soddisfazione e di utilizzo da parte dell'utente finale. Con l'affermazione dei paradigmi di sviluppo moderni, come la metodologia *Agile*, il collaudo non è più relegato alla fine del ciclo produttivo, ma si intreccia in modo continuo con lo sviluppo stesso, supportando rilasci frequenti e assicurando standard qualitativi e prestazionali costanti.

Nel campo dell'ingegneria informatica, è importante distinguere la terminologia legata alle problematiche del codice:

- **Difetto (Bug)**: anomalia intrinseca a livello di codice, ovvero una sequenza di istruzioni errate scritta durante la fase di programmazione.

- **Malfunzionamento (Failure)**: manifestazione tangibile di un difetto durante l'esecuzione del programma. Si verifica quando il comportamento effettivo del sistema diverge dalle aspettative dell'utente o dai requisiti [9].

Sebbene l'obiettivo ideale del testing sia la rimozione totale dei difetti, la teoria dell'ingegneria del software dimostra come l'esplorazione esaustiva di ogni singola permutazione di input e stato del sistema sia matematicamente e temporalmente irrealizzabile. L'obiettivo pratico è dunque mitigare il rischio fino a raggiungere una soglia di affidabilità accettabile per il rilascio. Tale soglia varia radicalmente a seconda del dominio di applicazione: nei sistemi cosiddetti *life-critical* (dove un errore informatico può tradursi in un pericolo per la vita umana, come nel settore medico o aerospaziale), la propensione al rischio deve essere praticamente nulla, richiedendo protocolli di validazione eccezionalmente rigorosi, prolungati ed esaustivi.

2.1.1 Piramide del Testing

Per strutturare una strategia di collaudo efficace, l'industria adotta comunemente il modello della *Piramide del Testing* (figura 2.1), un approccio gerarchico che organizza le verifiche in base al livello di granularità e isolamento. Questa struttura impone un rigore sequenziale, in cui i test di basso livello, essendo più numerosi, rapidi ed economici da mantenere, costituiscono le fondamenta [10]:

- **Unit Testing**: costituiscono la base della piramide e si concentrano sulla verifica delle porzioni di codice più piccole ed elementari (come ad esempio i metodi e le classi nella programmazione orientata agli oggetti). L'obiettivo è validare la logica interna di ciascuna unità in totale isolamento dal resto del sistema. Costruire basi solide in questa fase evita pesanti aumenti dei costi di refactoring e debug nelle fasi successive.
- **Integration Testing**: livello intermedio che esplora le interazioni tra i singoli moduli precedentemente validati. L'attenzione si sposta dallo stato interno del modulo ai protocolli di comunicazione, al passaggio dei dati e alle interfacce. Questa fase è cruciale per garantire che le diverse componenti architettoniche collaborino in modo sinergico senza generare conflitti o regressioni.
- **End-to-End (E2E) Testing**: questi test valutano l'architettura nella sua interezza. Simulando scenari d'uso reali dalla prospettiva dell'utente finale, attraversano l'intero stack applicativo (database e servizi back-end compresi). È la verifica definitiva per accertare che il sistema sia idoneo al rilascio in ambiente di produzione.
- **UI Testing**: collocati al vertice della piramide, questi test rappresentano il livello di astrazione più elevato e si concentrano in modo esclusivo sulla

validazione della GUI. Mentre i test E2E si occupano primariamente di verificare l'integrità dei flussi operativi e la corretta comunicazione con i sistemi di back-end, i test di UI si focalizzano sull'aspetto puramente visivo e interattivo dell'applicazione. Il loro scopo è accertare che gli elementi grafici (quali layout, pulsanti, form di inserimento e stili) siano renderizzati in modo impeccabile sui diversi dispositivi e che rispondano fluidamente alle azioni dirette dell'utente finale. Poiché strettamente legati ai design, soggetti a cambiamenti rapidi e frequenti, i test di UI sono i più fragili, lenti e costosi da mantenere. Perciò, l'architettura a piramide ne raccomanda un uso limitato ai soli casi d'uso necessari, affidando la verifica della logica di business e dell'architettura ai livelli inferiori, più rapidi e stabili [9].

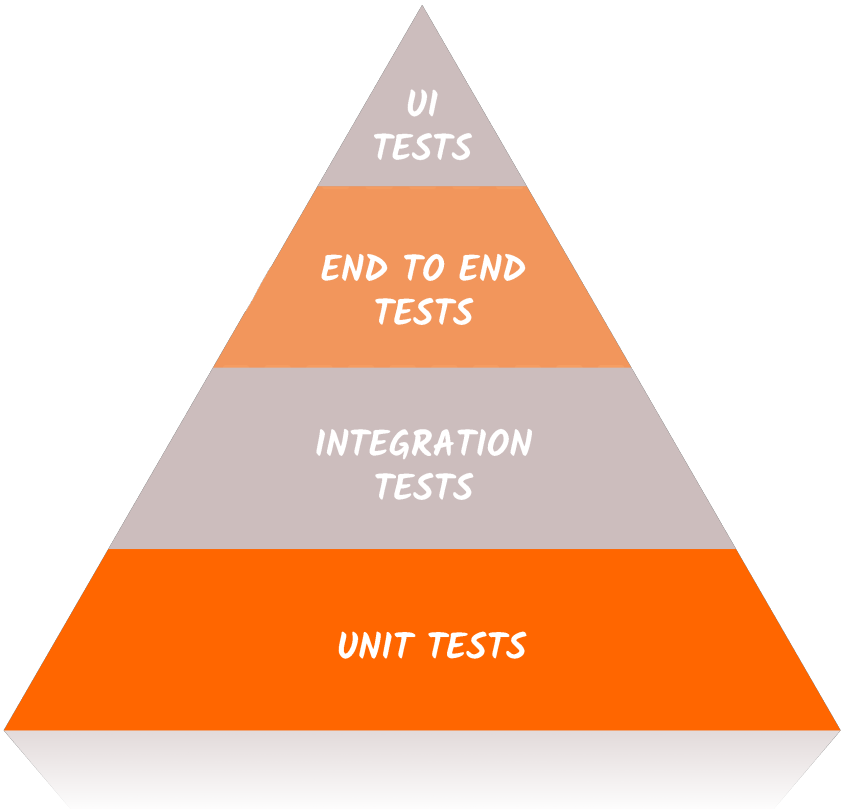


Figura 2.1: Piramide del Testing [10]

2.1.2 Verifica e Validazione dei test

Per valutare la qualità di un sistema informatico in modo completo, la fase principale di collaudo si divide in due processi complementari ma distinti (figura 2.2):

- **Verifica**: affidato al team di Quality Assurance (QA), si accerta che il software sia costruito nel modo corretto, rispettando rigorosamente le specifiche tecniche e di design. Tipicamente, coinvolge test statici che non richiedono l'esecuzione del programma.
- **Validazione**: affidato a un team di testing dedicato, si assicura di aver costruito il software giusto, ovvero che il prodotto finale risponda ai bisogni reali e all'uso previsto dall'utente. Richiede test dinamici, ovvero l'esecuzione effettiva del codice per valutare i requisiti (funzionali e non funzionali) [9].

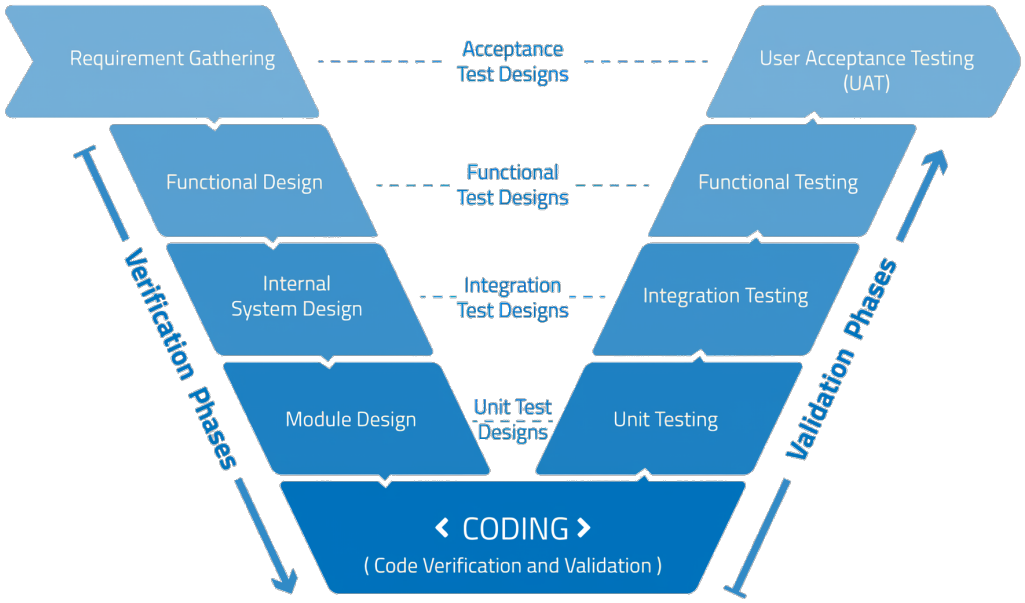


Figura 2.2: Verifica e Validazione nel software testing [11]

2.2 Mobile Testing

L'ecosistema degli smartphone ha vissuto una crescita straordinaria, diventando uno dei comparti tecnologici più capillari e profittevoli su scala mondiale. Questo sviluppo su larga scala è sostenuto da un costante avanzamento dell'hardware e dall'espansione dei mercati digitali (app store), che mettono a disposizione software per qualunque esigenza. In questo paradigma, le dimensioni compatte dei dispositivi hanno imposto il touch-screen come veicolo primario di interazione, rendendo l'*Interfaccia Grafica Utente* (GUI) il fulcro dell'esperienza utente. Considerando la saturazione del mercato, dove innumerevoli applicazioni offrono funzionalità sovrapponibili, l'usabilità, il design e la fluidità dell'interfaccia diventano i principali fattori per l'adozione e il successo commerciale di un prodotto. Di conseguenza, il collaudo rigoroso della componente grafica assume un'importanza strategica.
Il testing delle interfacce grafiche utente nelle applicazioni mobile mira a validare l'intero stack software, inclusi i servizi di back-end. Trattandosi tipicamente di un approccio black-box ed end-to-end, il tester valuta il rispetto dei requisiti funzionali e il comportamento generale del sistema senza accedere al codice sorgente. Tale pratica fornisce inoltre preziose indicazioni sulla robustezza e sulle prestazioni percepite dell'applicativo.
Tuttavia, nonostante la forte spinta verso l'automazione, le limitazioni strutturali degli attuali framework rendono il collaudo mobile un processo ancora fortemente dipendente dall'intervento manuale. Questa dipendenza si traduce in un notevole dispendio di risorse aziendali, causato dalle intrinseche difficoltà operative dell'ambiente mobile, che accetta input eterogenei non solo dall'utente, ma anche da una vasta gamma di sensori e componenti hardware (es. GPS, accelerometri, fotocamere), difficilmente emulabili in ambienti di test strettamente controllati [2].

2.2.1 Frammentazione e instabilità

Uno degli ostacoli più critici nel mobile testing è la *frammentazione*, ovvero l'estrema eterogeneità del mercato. Validare un'app richiede la gestione di una matrice di test multidimensionale, generata dalla combinazione di innumerevoli dispositivi fisici, diverse risoluzioni dello schermo e molteplici versioni dei sistemi operativi (figura 2.3).
Questo fenomeno è particolarmente marcato nell'ecosistema Android, caratterizzato da un'architettura aperta e da un vasto numero di produttori hardware (OEM), a differenza dell'ambiente iOS, dove il controllo centralizzato del produttore sull'hardware e la rapida adozione degli aggiornamenti software da parte degli utenti mitigano il problema. Per affrontare la frammentazione, le aziende si affidano spesso a infrastrutture cloud (come ad esempio le *Device Farms*) che permettono di testare il software su dispositivi fisici o virtualizzati da remoto. Sebbene efficaci,

queste soluzioni comportano costi di licenza elevati e introducono latenze di rete che dilatano i tempi di esecuzione dei test, entrando spesso in conflitto con i principi di *Continuous Integration* tipici delle metodologie *Agile* e *DevOps*.

Un'altra problematica ingente è rappresentata dai test instabili (*Flakiness*), definiti come script automatizzati che producono risultati non deterministici in esecuzioni successive, in assenza di modifiche al codice sorgente. Questa instabilità mina la fiducia nell'infrastruttura di automazione e genera ingenti costi, richiedendo sessioni prolungate di debugging per isolarne le possibili cause. Queste ultime possono includere problemi di concorrenza (race conditions), dipendenze da sistemi esterni, uso di dati dinamici non sufficientemente isolati, asserzioni non adeguate e persino errori all'interno degli stessi test [12].

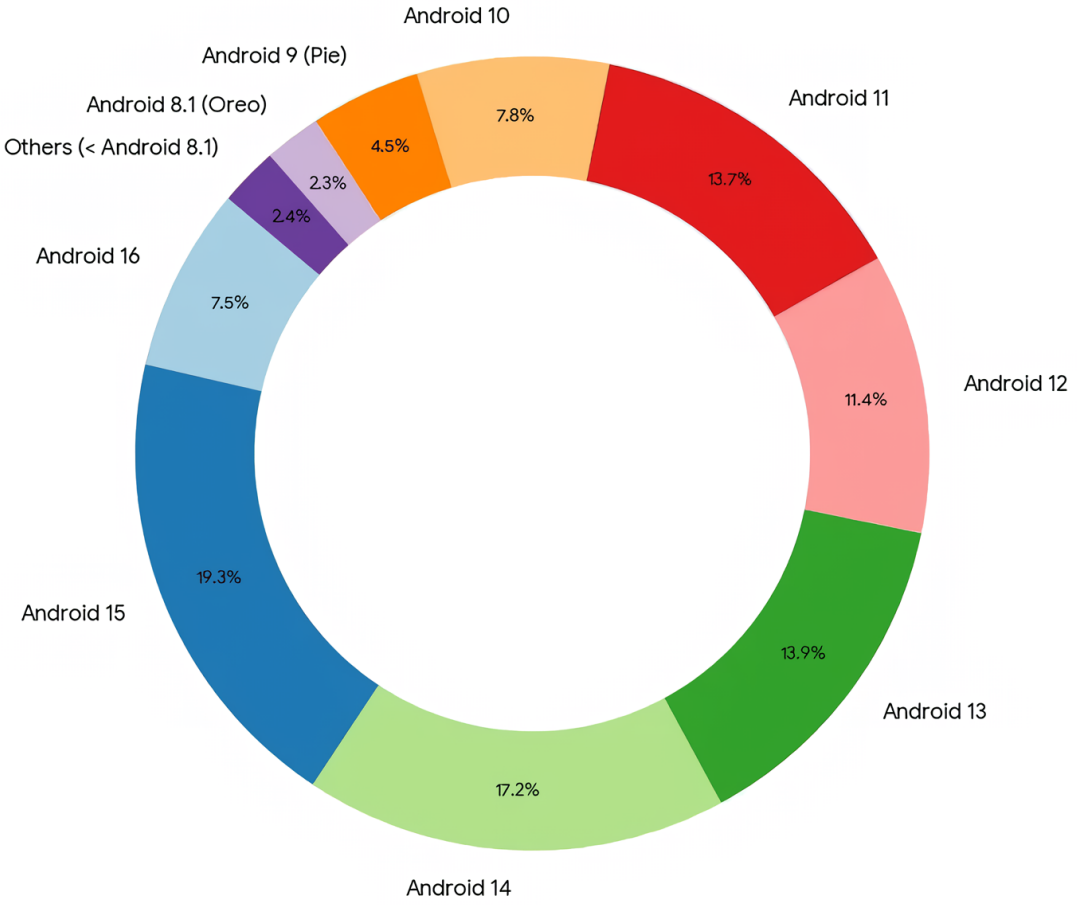


Figura 2.3: Frammentazione dei sistemi operativi Android [13]

2.2.2 Tecniche e strumenti

Per interagire programmaticamente con le GUI mobile, gli sviluppatori si avvalgono di *framework di automazione* che espongono specifiche API. Con il passare del tempo, questi strumenti si sono evoluti tre volte:

- **Coordinate**: inizialmente interagivano con lo schermo usando coordinate spaziali, ma risultavano estremamente fragili e bastava una minima variazione della densità di pixel o della risoluzione del display per causare il fallimento dei test.
- **Widget**: attualmente interrogano l'albero dei widget (componenti) dell'interfaccia, agganciandosi a identificativi univoci (ID, tag, classi, ecc.) forniti dal framework di sviluppo. Pur essendo più robusti, soffrono quando gli sviluppatori modificano l'architettura della UI o rinominano gli identificatori tra un rilascio e l'altro.
- **VGT**: l'approccio più recente è il *Visual GUI Testing*, che sfrutta algoritmi di computer vision per riconoscere gli elementi a schermo confrontandoli con screenshot di riferimento. Ne consegue che, pur risultando più resiliente ai cambiamenti strutturali del codice, è destinato inevitabilmente a non funzionare nel momento in cui un asset grafico viene modificato.

Esistono anche applicativi detti *Record and Replay* (R&R) che consentono di registrare le interazioni manuali di un tester, traducendole automaticamente in script eseguibili. Tuttavia, pur abbassando drasticamente la barriera d'ingresso all'automazione e velocizzando la scrittura dei test, non generano automaticamente le asserzioni logiche. Senza l'intervento umano, lo script si limita a riprodurre le azioni sullo schermo senza verificarne l'esito reale [12].

Un'ulteriore categoria è costituita dalle tecniche di *Automated Test Input Generation* (AIG), il cui obiettivo è generare automaticamente sequenze di input per esplorare l'applicazione, diminuendo il lavoro manuale richiesto e incrementando la copertura del codice (figura 2.4). Le sequenze possono essere ottenute seguendo tre approcci principali:

- **Model-based**: genera input basandosi su modelli formali del sistema (come ad esempio diagrammi o modelli matematici) per navigare logicamente le transizioni di stato.
- **Systematic**: tenta un'esplorazione metodica e combinatoria dello spazio degli input per massimizzare la copertura strutturale.
- **Random**: stressa l'interfaccia con input casuali allo scopo di causare crash imprevedibili, identificando vulnerabilità ed eccezioni non gestite.

Nonostante le potenzialità dell'AIG, permangono sfide quali: la gestione non nativa di eventi di sistema imprevisti; l'elevato costo computazionale del ripristino dello stato iniziale delle applicazioni tra un test e l'altro; la difficoltà nel sintetizzare input semantici complessi per form specifici; e i limiti nella comunicazione tra processi con servizi esterni [12].

Per compensare il deficit di informazioni lasciato dalle recensioni generiche degli utenti negli store digitali, si integrano nel codice sorgente strumenti di telemetria e di crash reporting. Questi SDK intercettano le eccezioni a runtime, raccogliendo log di sistema e registrazioni video delle azioni che precedono il crash, mettendo così a disposizione dei tester i dati diagnostici necessari per la risoluzione degli errori [12].

Come alternativa scalabile all'automazione, i servizi *crowd-based* affidano le sessioni di test a una rete distribuita di tester umani retribuiti a cottimo per ogni anomalia validata. Sfruttando la logica della gig economy, questa pratica garantisce l'accesso a un parco dispositivi reale ed eterogeneo. Le campagne possono essere focalizzate su indagini funzionali (ricerca di bug specifici), valutazioni euristiche di usabilità, testing per la sicurezza, o collaudi di localizzazione per verificare l'adeguatezza linguistica e culturale dell'interfaccia [12].

Name	Available	Instrumentation		Events		Exploration strategy	Needs source code	Testing strategy
		Platform	App	UI	System			
Monkey	✓	×	×	✓	×	Random	×	Black-box
Dynodroid	✓	✓	×	✓	✓	Random	×	Black-box
DroidFuzzer	✓	×	×	×	×	Random	×	Black-box
IntentFuzzer	✓	×	×	×	×	Random	×	White-box
Null IntentFuzzer	✓	×	×	×	×	Random	×	Black-box
GUIRipper	✓	×	✓	✓	×	Model-based	×	Black-box
ORBIT	×	×	×	✓	×	Model-based	✓	Grey-box
A^{3}E -Depth-first	✓	×	✓	✓	×	Model-based	×	Black-box
SwiftHand	✓	×	✓	✓	×	Model-based	×	Black-box
PUMA	✓	×	✓	✓	×	Model-based	×	Black-box
A^{3}E -Targeted	×	×	✓	✓	×	Systematic	×	Grey-box
EvoDroid	×	×	✓	✓	×	Systematic	×	White-box
ACTEve	✓	✓	✓	✓	✓	Systematic	✓	White-box
JPF-Android	✓	×	×	✓	×	Systematic	✓	White-box

Figura 2.4: Panoramica di alcuni AIG esistenti per Android [14]

2.3 Exploratory Testing

A fronte delle limitazioni intrinseche degli approcci rigidamente prescrittivi (come gli script di automazione) o puramente sequenziali, il *Testing Esplorativo* (ET) si è progressivamente affermato come metodologia cruciale per la validazione del software, risultando particolarmente performante nell'ambito delle interfacce grafiche utente mobile. Definito storicamente dai pionieri del settore Cem Kaner e James Bach, il testing esplorativo va inteso come un approccio cognitivo strutturato in cui l'apprendimento delle logiche di sistema, la progettazione dei casi di test e la loro esecuzione avvengono in modo simultaneo e iterativo [15].

Nel contesto del mobile testing, l'operatore esplora liberamente l'applicazione in fase di collaudo (AUT) attraverso la sua interfaccia, ma è guidato da specifici obiettivi. Svincolato dalla necessità di seguire pedissequamente passi prestabiliti, il tester sfrutta la propria intuizione, creatività e conoscenza del dominio per assumere la prospettiva dell'utente finale. Questa libertà d'azione permette di stressare l'interfaccia sondando "casi limite" e scenari di interazione complessi, come gesture impreviste o interruzioni hardware. Di conseguenza, l'approccio esplorativo risulta eccellente nello scovare anomalie sfuggenti, difetti di usabilità e malfunzionamenti legati all'esperienza utente che i test automatizzati tradizionali, privi di pensiero euristico, inevitabilmente ignorerebbero [3].

Tuttavia, l'implementazione sistematica dell'ET presenta delle innegabili sfide. In primo luogo, la mancanza di script formali rende complessa la riproducibilità dei difetti: ricreare l'esatta sequenza di gesture che ha innescato un'eccezione a runtime richiede un elevato sforzo di reverse engineering del flusso mentale del tester, a meno che non si utilizzino strumenti avanzati di registrazione video e log a supporto.
In secondo luogo, l'esito dell'attività è strettamente correlato al bagaglio di competenze e di esperienza della singola persona.
Infine, fattore di assoluta rilevanza, quando la validazione manuale esplorativa viene condotta per sessioni prolungate, l'attività rischia di trasformarsi in una procedura alienante e ripetitiva. La conseguente flessione della soglia di attenzione e motivazione del collaudatore finisce spesso per compromettere la profondità dell'esplorazione, riducendo l'efficacia stessa del processo [4].

Per arginare il declino fisiologico dell'attenzione e ripristinare un alto livello di coinvolgimento operativo, sia la ricerca accademica che l'industria del software stanno orientando i propri sforzi verso l'adozione della **Gamification**.

In questo contesto, ci riferiamo dunque al testing esplorativo "gamificato" (*Gamified Exploratory Testing*), che consiste nell'integrare dinamiche e meccaniche proprie del game design all'interno degli strumenti e delle pipeline di collaudo [5].
Attraverso l'integrazione di componenti ludiche (come quelle che verranno illustrate nelle pagine successive), l'attività di ricerca dei bug viene ripensata. Da compito gravoso e di routine, il testing si trasforma in una vera e propria sfida intellettuale. Le evidenze empiriche mostrano che fornire al tester feedback costanti e adeguati riconoscimenti non solo aumenta sensibilmente la sua produttività, ma favorisce anche un'esplorazione più ampia e variegata della GUI, portando a una copertura superiore del codice e all'identificazione di vulnerabilità latenti più insidiose [6].

Con il passare del tempo, sono stati ideati e validati numerosi strumenti basati sulla gamification da integrare nelle pratiche di testing del software:

- **Gamekins**: fa uso di meccaniche di gioco collegate ai commit nei repository software, permettendo agli sviluppatori di ottenere punti completando sfide e di gareggiare con i colleghi tramite classifiche. Il sistema si è rivelato efficace nello stimolare gli sviluppatori a scrivere un numero maggiore di casi di test all'interno di una pipeline CI/CD [16].

- **IntelliGame**: un plugin di gamification per IntelliJ IDEA, è pensato in modo specifico per le attività di test condotte direttamente nell'IDE, e la sua capacità di incrementare motivazione e produttività è stata evidenziata in uno studio con quasi 200 partecipanti in un contesto formativo [17].

- **Scout**: una versione "gamificata" di questo strumento per il testing esplorativo su dispositivi mobile ha influenzato positivamente il modo in cui i tester hanno progettato casi di test per applicazioni web, portando a una maggiore copertura dei test e a una maggiore efficienza nell'individuazione dei bug [6].

Come sottolineato da recenti revisioni della letteratura multidisciplinare, l'uso di meccanismi di gamification nel testing in ambito mobile risulta tuttora meno diffuso rispetto al testing delle applicazioni web [18], circostanza che giustifica l'adozione dello strumento sviluppato in questo lavoro di tesi come mezzo per contribuire a colmare tale lacuna nella letteratura.

2.4 Octalysis Framework

Uno degli obiettivi principali della tesi era quello di progettare e implementare efficacemente i principi della gamification nell'ambito del testing esplorativo, basandosi su un framework teorico solido e ampiamente riconosciuto. Per questo motivo, è stato scelto l'*Octalysis Framework*, ideato da Yu-kai Chou [8], che rappresenta una delle teorie più influenti e complete nel campo della gamification.

Yu-kai ha definito la gamification come un approccio incentrato sulla motivazione umana, che utilizza elementi tipici dei giochi e li applica ad attività del mondo reale, ossia contesti non ludici.
Ha inoltre definito due metodi di implementazione della gamification:

- **Esplicito**: utilizza giochi veri e propri per raggiungere gli obiettivi. Questo garantisce certamente la creazione di un ambiente più divertente, offrendo al contempo ai designers una grande libertà creativa, poiché coinvolge esplicitamente dei veri e propri giochi. Tuttavia, gli svantaggi risiedono nella possibilità di non essere presi sul serio e nel bisogno di una maggiore quantità di risorse.

- **Implicito**: consiste in una forma di progettazione che sfrutta abilmente le tecniche e gli elementi tipici dei giochi, inserendoli nell'esperienza dell'utente con la massima discrezione possibile. Questo approccio è tecnicamente più facile da adottare e nella maggior parte dei casi è il più appropriato, ma la sua stessa praticità può spesso portare a progetti superficiali e poco curati se non si adottano le precauzioni necessarie.

Si tratta quindi di un *Human-Focused Design*, in contrapposizione al *Function-Focused Design*: un processo che ottimizza le motivazioni e il coinvolgimento delle persone all'interno di un sistema, piuttosto che puntare esclusivamente all'efficienza. Il motivo per cui la gamification viene definita come *Human-Focused Design* è che il primo settore in grado di sviluppare competenze in questo campo è stato quello videoludico, dove i game designers hanno dedicato anni a capire come mantenere i propri clienti costantemente coinvolti.
Quasi tutti i giochi sono divertenti perché fanno leva su determinati meccanismi motivazionali che guidano ogni attività umana: alcuni sono fonte di ispirazione e crescita personale, mentre altri sono manipolatori e ossessivi [19].

Il framework si chiama Octalysis perché può essere rappresentato come un diagramma dalla forma ottagonale (figura 2.5) che si basa sui seguenti otto *Core Drives* (in altre parole, i motivatori psicologici fondamentali).

2.4.1 Epic Meaning & Calling

Il primo Core Drive si attiva quando le persone si sentono ispirate e quando sono entusiaste di far parte di uno scopo o di un piano più grande di loro (in ambito videoludico, spesso il mondo sta per essere distrutto e, per qualche motivo, il giocatore è l'unico in grado di salvarlo). Si possono ottenere utenti attivi e si può ampliare il bacino di utenza grazie alla loro passione e al loro coinvolgimento nella causa. Solitamente viene introdotto durante la *Fase di Scoperta*, ossia quando gli utenti iniziano a interagire con l'applicazione.
Alcune tecniche utili ad implementare questo Core Drive in modo da creare impegno, motivazione e lealtà sono le seguenti:

- **Narrazione**: una narrazione coinvolgente permette di presentare una storia che fornisca agli utenti un contesto per attribuire un significato più profondo al loro rapporto con l'applicazione.

- **Eroe dell'umanità**: integrando una missione globale nell'applicazione, è possibile ottenere un coinvolgimento ancora maggiore durante il processo di onboarding. Il meccanismo consiste nel collegare le azioni che gli utenti devono compiere a qualcosa che renderà il mondo un posto migliore.

- **Elitarismo**: consente agli utenti di formare un gruppo orgoglioso basato su etnia, credenze o interessi comuni che li fa sentire parte di una causa più grande. Il gruppo cerca anche di ostacolare i propri rivali, il che può portare entrambi a migliorarsi ulteriormente per battere la concorrenza.

- **Fortuna del principiante**: gli utenti si sentono come se fossero stati tra i pochi scelti per agire, il che li rende molto più propensi a farlo. Ad esempio, se un giocatore, il primo giorno in cui gioca, ottiene casualmente una delle armi più potenti disponibili, è probabile che non smetterà presto di giocare. Inoltre, se il game designer aggiunge anche il *Core Drive 6: Scarcity & Impatience* dicendo all'utente che potrà equipaggiare quell'arma solo dopo aver superato un livello particolarmente difficile, lui diventerà ossessionato e cercherà di capire come superarlo il più rapidamente possibile.

- **Omaggio**: offrire agli utenti regali che solitamente non sono gratuiti può farli sentire speciali e invogliarli a continuare ad utilizzare l'applicazione.

Tuttavia, è importante tenere presente che si rischia davvero di disinteressare gli utenti se si dà l'impressione di non essere sinceri nel tentativo di realizzare questo Core Drive [20].

2.4.2 Development & Accomplishment

Il secondo Core Drive si attiva quando le persone sentono di progredire, di migliorare e di raggiungere risultati significativi. Questo è uno dei motivatori più potenti, poiché tutti amano sentirsi competenti e realizzati. È anche uno dei più facili da implementare, infatti molte aziende si focalizzano esclusivamente su di esso quando intendono implementare tecniche di gamification. Le più utilizzate sono le seguenti:

- **Punteggi**: il metodo più semplice per fornire feedback immediato agli utenti sui loro progressi e per riconoscere i loro sforzi. I punti solitamente possono essere guadagnati completando attività o raggiungendo obiettivi specifici.
- **Trofei/Obiettivi**: rappresentano un riconoscimento visivo dei risultati raggiunti dagli utenti. Possono essere sbloccati al completamento di determinate sfide o al raggiungimento di traguardi specifici, e fungono da simbolo di status e di competenza all'interno della comunità degli utenti.
- **Classifiche**: creano competizione tra gli utenti, mostrando le loro prestazioni rispetto a quelle degli altri. Possono essere costruite su numerosi criteri di valutazione, e incentivano gli utenti a migliorarsi per scalarle.

Tuttavia, il solo fatto di vedere progressi non significa che ci si senta realizzati. La chiave sta nell'assicurarsi che gli utenti stiano superando sfide di cui possono essere orgogliosi. È quindi importante concentrarsi su come si vuole che gli utenti si sentano, piuttosto che su quali elementi ludici si vogliono utilizzare [21].

2.4.3 Empowerment of Creativity & Feedback

Il terzo Core Drive pone molta enfasi sul lato ludico. Si attiva quando gli utenti possono mettere continuamente alla prova la propria creatività e inventare un numero infinito di nuovi modi di utilizzare l'applicazione. In tal modo, il game designer non ha più bisogno di creare costantemente nuovi contenuti per mantenere vivo l'interesse, poiché è la mente stessa dell'utente a costituire quel contenuto che catturerà continuamente la sua attenzione.
Alcune tecniche utili ad implementare questo Core Drive sono le seguenti:

- **Boosters/Potenziamenti**: elementi che, in misura e/o per un periodo di tempo limitati, aiutano gli utenti a raggiungere più facilmente il risultato desiderato (la sensazione di acquisire nuovi potenziamenti, per quanto limitati, è esaltante e costituisce un forte stimolo verso il raggiungimento dell'obiettivo).
- **Sblocco di funzionalità**: una delle meccaniche di gioco di maggior successo, che sblocca una possibilità entusiasmante precedentemente inesistente una volta raggiunto un determinato obiettivo.

- **Percezione della scelta**: molti studi hanno dimostrato che alle persone piace di più qualcosa quando viene data loro la possibilità di scegliere, anche se le opzioni non sono così allettanti rispetto a una singola alternativa migliore. La chiave è che la scelta in sé non è necessariamente significativa, ma dà agli utenti la sensazione di avere il potere di decidere. Inoltre, se le scelte sono significative, tanto meglio.

Purtroppo è difficile sfruttare appieno questo potenziale perché questo è uno dei Core Drives più complicati da implementare: infatti, richiede all'utente di dedicargli una certa dose di attenzione, il che non è affatto scontato nella società odierna, caratterizzata da carenza di attenzione e sovraccarico di informazioni [22].

2.4.4 Ownership & Possession

Il quarto Core Drive si basa sul principio che, quando si possiede qualcosa, si desidera migliorarla, proteggerla e averne ancora di più. Inoltre, a un livello più astratto, se si è investito molto tempo per personalizzare qualcosa secondo i propri gusti, oppure se un sistema ha costantemente appreso le proprie preferenze adattandosi fino a diventare parte integrante della propria identità, in genere si inizia a provare un maggiore senso di appartenenza nei suoi confronti.
Alcune tecniche che utilizzano molto questo Core Drive per interagire con gli utenti sono le seguenti:

- **Costruire da zero**: quando le persone realizzano qualcosa partendo da zero, provano un senso di appartenenza verso di esso e potrebbero non sentirsi così coinvolte diversamente.
- **Set da collezione**: assegnando agli utenti alcuni badges/trofei e spiegando loro che questi fanno parte di un set da collezione legato a un tema specifico, si stimola in loro il desiderio di collezionarli tutti e completare la collezione.
- **Alfred Effect**: si verifica quando gli utenti sentono che un prodotto/servizio è talmente personalizzato in base alle loro esigenze che non riescono a immaginare di utilizzare un altro. Quando un utente ha la sensazione che un sistema abbia imparato tutto su di lui e si sia adattato alle sue esigenze, anche se un altro servizio offrisse tecnologie, funzioni o prezzi migliori, l'utente tenderà comunque a rimanere fedele a quel sistema, perché ora è in modo univoco "il suo".
- **Protector Quest**: è un concetto basato sul fatto che le persone iniziano a sviluppare un rapporto con qualcosa che stanno proteggendo.

Si tratta di un Core Drive fondamentale in contesti in cui vengono utilizzati elementi quali le valute o i beni virtuali, nonché nell'ambito delle attività di ricerca sui collezionabili [23].

2.4.5 Social Influence & Relatedness

Il quinto Core Drive incorpora elementi sociali che stimolano le persone, tra cui il tutoraggio, l'accettazione, le reazioni sociali, il senso di compagnia, così come la competizione e l'invidia. Comprende anche il nostro desiderio di avvicinarci a persone, luoghi o eventi con cui possiamo identificarci. Se utilizzato correttamente, può rappresentare una delle motivazioni più forti e durature che spingono le persone a connettersi e a impegnarsi. Grazie ai social media e alle reti internet, molti servizi cercano di sfruttare questa spinta sin dalle prime fasi (*scoperta* e *onboarding*) incoraggiando l'interazione sociale, invitando gli amici, ecc. Ma limitarsi a chiedere agli utenti di invitare gli amici non è sufficiente: i contenuti e il contesto devono essere davvero degni di essere condivisi.
Le tecniche utili ad implementare questo Core Drive sono numerose:

- **Tutoraggio**: abbinare utenti esperti a nuovi arrivati aiuta nell'inserimento, offre sostegno emotivo e, inoltre, conferisce prestigio ai veterani.
- **Obiettivi di gruppo**: compiti che richiedono uno sforzo collettivo per sbloccare ricompense.
- **Brag buttons & Touting flags**: meccanismi che consentono agli utenti di mostrare i propri risultati agli altri (come ad esempio la condivisione di punteggi, trofei, o elementi di status visibili).
- **Tesori sociali**: oggetti o ricompense ottenibili solo tramite amici o gesti sociali (come ad esempio regali o azioni di riconoscenza).
- **Ancora di conformità**: mostrare cosa stanno facendo i propri pari è una prova sociale utile a guidare il comportamento degli utenti.
- **Luoghi di ritrovo**: spazi (come ad esempio forum, chat, ecc.) dove gli utenti possono interagire socialmente in modo informale aiuta a costruire una comunità.
- **Stimoli sociali**: interazioni sociali che richiedono poco sforzo (come ad esempio i "mi piace"). Sono metodi semplici per permettere alle persone di socializzare senza impegno.

Tuttavia, è importante tenere presente che l'influenza sociale può essere controproducente: se la riprova sociale o la comunità degli utenti risultano deboli, oppure se i contenuti non sono coinvolgenti, il messaggio può sembrare privo di significato; inoltre, un uso eccessivo di funzioni come "invita gli amici" può infastidire gli utenti se gli incentivi non sono sufficientemente allettanti [24].

2.4.6 Scarcity & Impatience

Il sesto Core Drive si attiva quando le persone desiderano qualcosa semplicemente perché è difficile da ottenere, raro o non immediatamente disponibile. Questo fenomeno attinge a impulsi emotivi e irrazionali: quando qualcosa è appena fuori dalla nostra portata, spesso lo desideriamo ancora di più. Se usato bene, incoraggia le persone ad agire subito, a impegnarsi di più o a rimanere coinvolte, soprattutto per il timore che, se non agiscono, potrebbero perdere l'occasione.
Alcune tecniche utili ad implementare questo Core Drive sono le seguenti:

- **Limiti virtuali**: stabilire dei limiti che creano un senso di scarsa disponibilità (ad esempio, limitare la quantità di valuta virtuale che un utente può possedere lo spingerà a darsi da fare).
- **Dinamiche di disponibilità**: rendere alcuni contenuti disponibili solo in determinati momenti (ad esempio, con eventi ricorrenti o offerte a tempo limitato).
- **Torture Breaks**: costringere gli utenti a sospendere (intenzionalmente) l'attività. I tempi di attesa dopo aver raggiunto un limite creano impazienza e spesso portano a riflettere maggiormente sull'applicazione e a tornare ad utilizzarla.
- **Interfaccia utente evoluta**: tenere nascoste alcune funzionalità o complessità all'inizio dell'utilizzo dell'applicazione e sbloccarne gradualmente altre mano a mano che l'utente progredisce fa percepire alcune funzionalità come più preziose, poiché non sono immediatamente disponibili.

Tuttavia, è importante tenere presente che occorre trovare il giusto equilibrio (se le regole sono troppo rigide o i tempi di attesa troppo lunghi, gli utenti potrebbero perdere la pazienza o sentirsi penalizzate). Ciò che funziona per stimolare azioni nel breve termine potrebbe compromettere la fedeltà a lungo termine se gli utenti si sentono manipolati o costantemente ostacolati [25].

2.4.7 Unpredictability & Curiosity

Il settimo Core Drive si attiva quando le persone affrontano situazioni incerte o sorprendenti e, di conseguenza, non sanno cosa potrebbe accadere in seguito, ma sono interessate a scoprirlo. Gli esseri umani sono curiosi per natura, e la sorpresa è profondamente legata a ciò che rende le esperienze coinvolgenti e divertenti. Funziona particolarmente bene nelle situazioni a lungo termine: quando le novità svaniscono, l'imprevedibilità può mantenere vivo l'interesse.
Alcune tecniche che utilizzano questo Core Drive per coinvolgere maggiormente gli utenti sono le seguenti:

- **Ricompense variabili/casuali**: offrire ricompense variabili o casuali al posto della stessa ricompensa per il completamento di un'azione aumenta l'imprevedibilità e stimola la curiosità.
- **Easter Eggs/Ricompense a sorpresa**: ricompense da assegnare senza preavviso (in particolare, si tratta di premi aggiuntivi concessi in seguito a eventi imprevedibili).
- **Ricompense ricorrenti/Lotterie**: offrire ricompense che si verificano periodicamente, in modo tale che continuare ad utilizzare l'applicazione offra la possibilità di vincere premi.
- **Scelte in evidenza**: guidare gli utenti verso una possibile azione successiva tramite indizi visivi per suscitare curiosità. Questa è una delle tecniche più efficaci in assoluto poiché evita che gli utenti si sentano persi mantenendoli comunque coinvolti.

Tuttavia, è importante tenere presente che un eccesso di imprevedibilità/casualità può risultare opprimente o ansiogeno, quindi è necessario trovare un equilibrio con un certo grado di prevedibilità [26].

2.4.8 Loss & Avoidance

L'ottavo e ultimo Core Drive motiva le persone attraverso il timore di perdere qualcosa o il desiderio di evitare esiti negativi. Ciò può riguardare beni in cui hanno già investito o opportunità che potrebbero svanire se non agissero. Tendiamo infatti ad essere più motivati a evitare di perdere qualcosa piuttosto che ad acquisirla.
Anche questo Core Drive può essere implementato attraverso numerose tecniche:

- **Eredità legittima**: far sentire agli utenti che possiedono già qualcosa o che se lo meritano. In tal modo, il rischio di perdere ciò che si sono "guadagnati" li motiverà a impegnarsi per mantenerlo.
- **Opportunità effimere e Countdown timers**: opportunità che svaniscono se non vengono colte immediatamente. I timer con il conto alla rovescia sottolineano visivamente il tempo limitato rimanente, suscitando un senso di urgenza.
- **Pigrizia dello status quo**: gli utenti tendono a procrastinare o ad attenersi a ciò che conoscono, quindi, riconoscere questo stato di inerzia aiuta a progettare modi per superarlo.
- **Fear of Missing Out (FOMO)**: mostrare ciò che gli utenti potrebbero perdersi se non agiscono, al fine di provocarne una reazione.

- **Prigione dei costi irrecuperabili**: se un utente ha investito molto tempo, impegno o/e risorse, potrebbe continuare anche se non è più divertente per evitare di avere la sensazione che il suo investimento passato sia andato sprecato.

Tuttavia, è importante tenere presente che, per funzionare efficacemente, questo Core Drive deve essere accompagnato da indicazioni chiare e concrete, in modo che gli utenti sappiano come comportarsi per evitare eventuali perdite. Altrimenti, motivare gli utenti solo attraverso la paura diventa controproducente [27].

2.4.9 Left & Right Brain

Inoltre, nell'Octalysis Framework, i Core Drives sulla sinistra sono quelli dell'emisfero sinistro, più legati alla logica, al ragionamento e al possesso. I Core Drives sulla destra sono quelli dell'emisfero destro, più associati alla creatività, alla capacità di esprimersi e ai rapporti sociali [28].
Nello specifico:

- **Left Brain**: comprende i *Core Drives 2, 4, 6*. Sono caratterizzati da una motivazione *estrinseca* che deriva dal desiderio di raggiungere un obiettivo o di ottenere una ricompensa. Ciò implica che l'attività in sé non sia necessariamente interessante o piacevole. I possibili vantaggi consistono nel miglioramento della concentrazione su attività di routine monotone e nella nascita di un interesse iniziale per tali attività. D'altra parte, gli svantaggi risiedono nella riduzione delle capacità creative e sociali, della curiosità e del pensiero fuori dagli schemi.

- **Right Brain**: comprende i *Core Drives 3, 5, 7*. Sono caratterizzati da una motivazione *intrinseca* che deriva dal divertimento e dal coinvolgimento legati alle attività in sé. I possibili vantaggi consistono nell'interesse per l'attività stessa, dovuto alla creatività, ai rapporti sociali e alla curiosità, mentre il principale svantaggio possibile risiede nella mancanza di interesse iniziale per tali attività. Inoltre, Yu-kai sostiene che un'esperienza, per essere maggiormente intrinseca, dovrebbe essere modificata in modo da fare leva principalmente sulla socialità, sull'imprevedibilità e sulla casualità. Per di più, dovrebbero essere aggiunte scelte significative che riflettano lo stile e le preferenze degli utenti.

2.4.10 White & Black Hat

Infine, nell'Octalysis Framework, i Core Drives superiori sono considerati positivi, mentre quelli inferiori sono considerati negativi [29].
Nello specifico:

- **White Hat**: comprende i *Core Drives 1, 2, 3*. Fanno sentire le persone potenti, appagate e soddisfatte, dando loro la sensazione di avere il controllo della propria vita e delle proprie azioni. I possibili vantaggi consistono in un aumento continuo del coinvolgimento, la fidelizzazione a lungo termine e la costruzione di un rapporto di fiducia, mentre lo svantaggio principale risiede nell'assenza di un senso di urgenza.
- **Black Hat**: comprende i *Core Drives 6, 7, 8*. Fanno sentire le persone ossessionate, ansiose e dipendenti, dando loro l'impressione di non avere il controllo sulle proprie azioni. I vantaggi consistono nell'aumento dei ricavi a breve termine e nell'aumento iniziale del coinvolgimento delle persone. D'altra parte, il possibile svantaggio risiede nella limitata sostenibilità a lungo termine dovuta alla perdita di controllo provata dalle persone quando sono oggetto di tecniche manipolative.

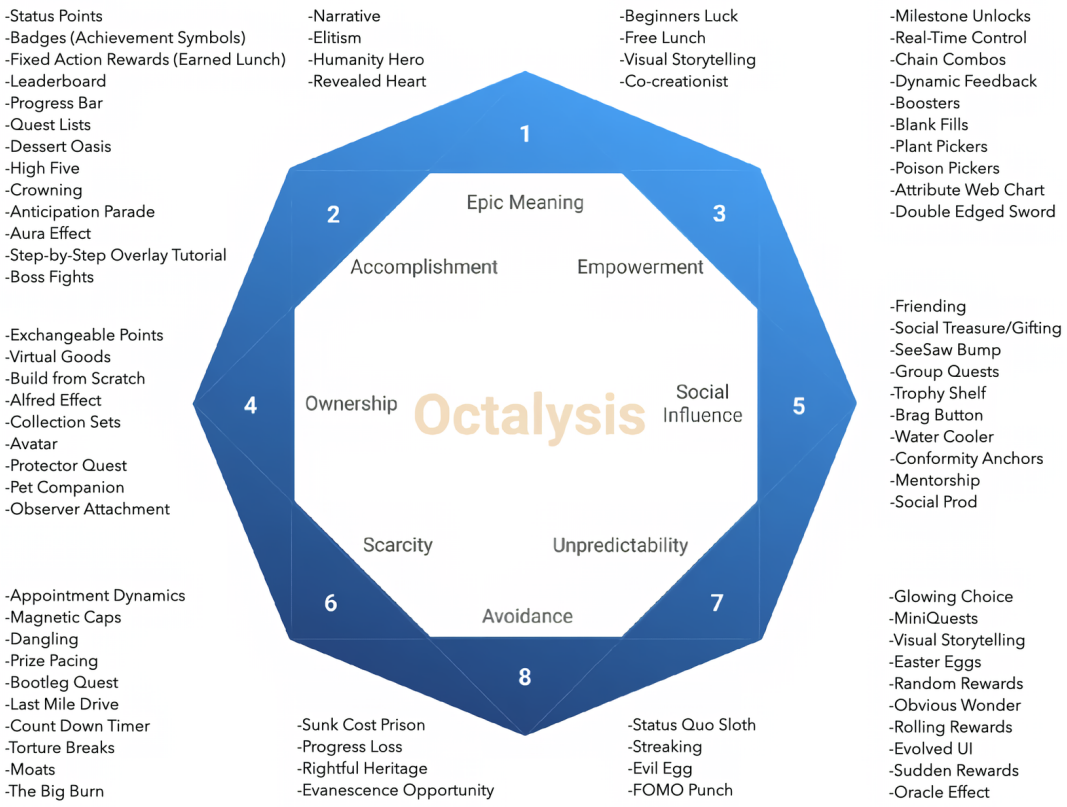


Figura 2.5: Octalysis Framework [19]

Capitolo 3

Strumenti utilizzati

Il presente capitolo ha lo scopo di illustrare lo stack tecnologico adottato, basato su un ecosistema moderno e ampiamente consolidato nel panorama del web development, il cui fulcro è rappresentato dal linguaggio JavaScript.
Attorno a questo nucleo sono presenti framework e gestori di pacchetti specifici, essenziali per ottimizzare la creazione dell'interfaccia utente e per l'orchestrazione delle dipendenze software.
Successivamente, verranno approfonditi il framework Appium e il prototipo originale GAppium, che costituiscono la base su cui è stata realizzata l'estensione "gamificata" oggetto di questo lavoro di tesi.

3.1 Stack tecnologico

3.1.1 JavaScript

JavaScript è un linguaggio di programmazione ad alto livello, tipicamente compilato in modalità *just-in-time* (JIT), che costituisce la tecnologia fondante per la costruzione di applicazioni web. Tra le sue caratteristiche peculiari figurano la tipizzazione dinamica, l'orientamento agli oggetti basato su prototipi e l'impiego di funzioni di prima classe. Si tratta di un linguaggio spiccatamente multi-paradigma, in grado di supportare nativamente stili di programmazione orientati agli eventi, funzionali e imperativi. JavaScript è inoltre corredato da un ricco set di API per la manipolazione di testi, date, espressioni regolari e strutture dati, e per l'interazione con il *Document Object Model* (DOM).
Da un punto di vista architetturale, JavaScript è un linguaggio single-threaded che basa la propria esecuzione sul paradigma dell'*event loop*. Durante l'esecuzione (runtime), il sistema elabora i messaggi provenienti da una coda processandoli sequenzialmente; per ogni nuovo messaggio viene invocata la funzione associata,

creando un frame all'interno dello stack di chiamate. Tale frame contiene gli argomenti della funzione e le relative variabili locali, espandendosi o contraendosi dinamicamente in base alle necessità computazionali. Una volta che lo stack di chiamate si svuota e la funzione termina il suo ciclo, JavaScript procede con il messaggio successivo presente nella coda.
L'esecuzione del codice è demandata a un componente software specifico, ossia l'*engine* di JavaScript. Tali componenti forniscono il sistema di runtime essenziale per abilitare l'interazione con l'ambiente circostante. Il runtime include le API necessarie per le operazioni di input/output, come le comunicazioni di rete, l'archiviazione dati, la renderizzazione grafica e le funzionalità di importazione di script esterni [30].

3.1.2 React

Per la progettazione e lo sviluppo dell'interfaccia web si è fatto ricorso a React, una libreria JavaScript open-source orientata alla creazione di interfacce utente basate su componenti. React offre un vantaggio prestazionale e architetturale significativo: essa effettua il re-rendering esclusivamente delle porzioni di pagina che hanno subito modifiche. Ciò permette di evitare inutili aggiornamenti degli elementi DOM invariati grazie all'impiego del *Virtual DOM*, una struttura dati che funge da cache per calcolare e applicare solo le differenze risultanti al DOM reale. Il framework aderisce fedelmente al paradigma della programmazione dichiarativa: lo sviluppatore definisce le *viste* (views) corrispondenti a ciascuno stato dell'applicazione, e React si occupa autonomamente di gestire l'aggiornamento e la renderizzazione dei componenti al variare dei dati sottostanti.
L'architettura di un'applicazione React è intrinsecamente stratificata in molteplici livelli di componenti modulari e riutilizzabili. Questi vengono inseriti nella radice del DOM mediante l'utilizzo della libreria accessoria React DOM. La stesura dei componenti avviene abitualmente avvalendosi di *JSX*, un'estensione sintattica del linguaggio JavaScript che permette di scrivere markup direttamente all'interno della logica applicativa.
Una funzionalità cardine introdotta nelle versioni più recenti di React è rappresentata dagli *Hooks*, funzioni che consentono agli sviluppatori di "agganciarsi" allo stato interno di React e ai metodi del ciclo di vita direttamente dai componenti di tipo funzionale. Gli Hooks sottostanno a regole rigorose (ad esempio, possono essere invocati solo al livello superiore del codice e solo dall'interno di funzioni React) e la libreria ne fornisce alcuni predefiniti di fondamentale importanza, come *useState* e *useEffect* [31].
Oltre ai già citati componenti riutilizzabili, al Virtual DOM e alla programmazione dichiarativa, React presenta ulteriori punti di forza che ne hanno garantito il successo, come il supporto di un'ampia e attiva comunità, il riutilizzo del codice su

più piattaforme (ad esempio tramite *React Native*) e, soprattutto, l'adozione di un flusso di dati unidirezionale, che rende la gestione dello stato dell'applicazione più prevedibile e facile da debuggare (figura 3.1).

3.1.3 Npm

Per garantire una corretta gestione delle librerie e dei plugin è stato utilizzato Npm (Node Package Manager), il gestore di pacchetti d'elezione per il linguaggio JavaScript, nonché il package manager predefinito per l'ambiente di runtime *Node.js*. Il suo ecosistema si divide in due parti principali: un'interfaccia a riga di comando (CLI) e un vasto database online contenente pacchetti sia pubblici che privati, denominato *npm registry*.
Tramite la CLI, gli sviluppatori possono integrare e distribuire moduli JavaScript, esplorabili agevolmente tramite il portale web dedicato. Impiegato come gestore delle dipendenze per un progetto locale, Npm permette di installare, attraverso un singolo comando, tutte le dipendenze dichiarate all'interno del file di configurazione *package.json*. In questo file, per ciascuna dipendenza, è possibile specificare un intervallo di versioni valide (sfruttando il *semantic versioning*): tale meccanismo consente l'aggiornamento automatico dei pacchetti in uso, proteggendo le applicazioni dall'introduzione accidentale di modifiche incompatibili. Oltre al file principale, Npm genera in automatico il documento *package-lock.json*, la cui funzione è quella di "congelare" l'albero delle dipendenze registrando le versioni esatte scaricate durante l'ultima installazione, garantendo così l'assoluta riproducibilità dell'ambiente di sviluppo [32].

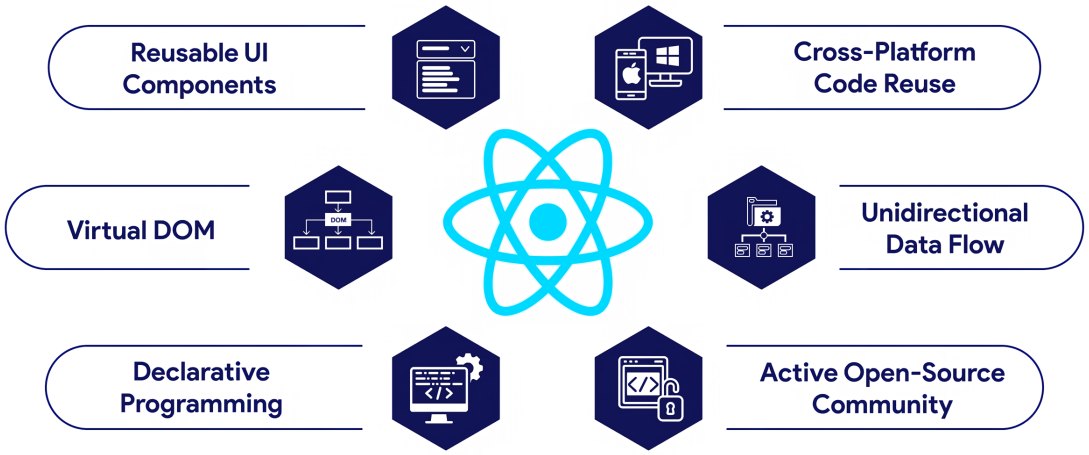


Figura 3.1: Vantaggi del framework React [33]

3.2 Appium

Nel panorama dell'ingegneria del software e della garanzia di qualità (QA), l'automazione dei test su dispositivi mobili rappresenta una delle sfide più complesse a causa della frammentazione dei sistemi operativi e della varietà di hardware disponibili. In questo contesto, Appium si è affermato come uno dei framework open-source più rilevanti per l'automazione di interfacce utente.
La peculiarità di Appium risiede nella sua filosofia *cross-platform* e *language-agnostic*. Esso permette di implementare script di automazione per diverse piattaforme, tra cui mobile (Android e iOS), web e desktop, utilizzando diversi linguaggi di programmazione, quali JavaScript, Java e Python. Questa flessibilità deriva dalla scelta progettuale di non vincolare lo sviluppatore a un linguaggio specifico o a un SDK proprietario, rendendolo uno strumento estremamente versatile rispetto a soluzioni native [34].

3.2.1 Architettura

L'architettura di Appium è ispirata al modello *client-server* e poggia su tre pilastri fondamentali: l'*Appium Core*, i *Client* e i *Driver*.

Il Core definisce le API del framework, traendo le proprie origini da *Selenium*, software pioniere nell'automazione web. Il team di Appium, in collaborazione con i principali vendor di browser e il *consorzio W3C* (World Wide Web Consortium), ha adottato la WebDriver specification come protocollo di comunicazione standard. L'adozione di tale specifica è motivata dalla profonda somiglianza strutturale tra le interfacce dei browser e quelle delle applicazioni mobili: sebbene le modalità di interazione differiscano, la gerarchia degli elementi (DOM o UI Tree) segue logiche analoghe. Da un punto di vista tecnico, il protocollo WebDriver si basa su richieste *HTTP RESTful*.
Questo approccio garantisce *modularità* (server e client possono risiedere su macchine diverse), *interoperabilità* (qualsiasi linguaggio dotato di librerie HTTP può interagire con il server) e *standardizzazione* (l'automazione segue regole universali definite a livello internazionale).

Il processo di automazione (figura 3.2) si articola attraverso l'interazione coordinata di quattro componenti chiave:

- **Client**: la libreria utilizzata dallo sviluppatore per scrivere i test (in questo lavoro di tesi, *WebDriverIO*).
- **Server**: un server *Node.js* che riceve le richieste HTTP dal client, le interpreta e le instrada al driver corretto.

- **Automation Driver**: il modulo specifico che traduce i comandi astratti di WebDriver in comandi nativi comprensibili dal sistema operativo.
- **Dispositivo**: l'emulatore, il simulatore o il dispositivo fisico su cui viene eseguita l'applicazione in fase di collaudo (*Application Under Test* - AUT) [34].

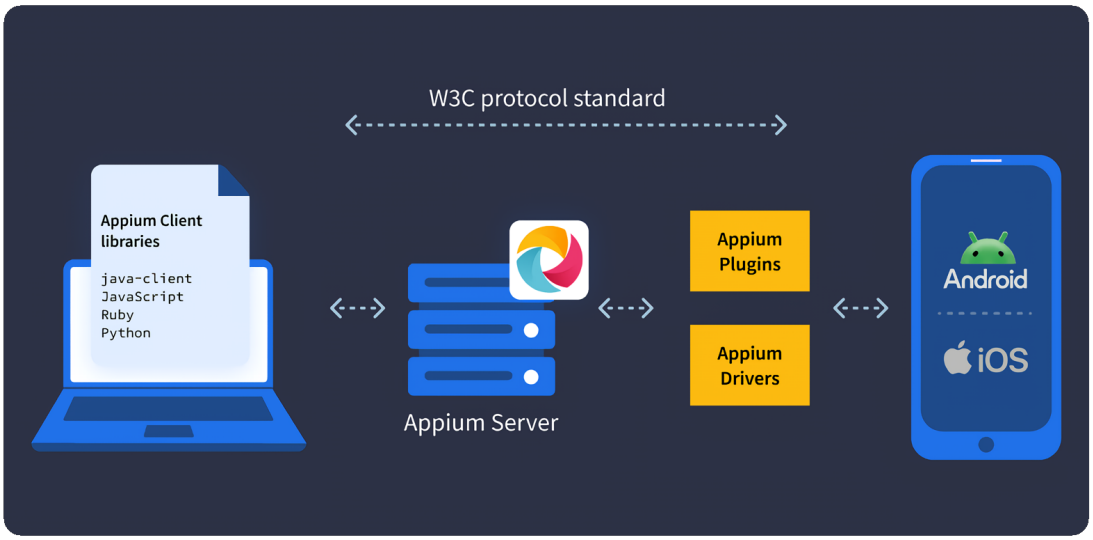


Figura 3.2: Architettura di Appium [35]

3.2.2 Client WebDriverIO

Il client ha il compito di inviare comandi al server e ricevere risposte riguardanti lo stato dell'applicazione o l'esito delle operazioni. Sebbene il protocollo WebDriver operi tramite endpoint HTTP (come ad esempio, una richiesta POST verso */session/:sessionid/element* per individuare un componente), le librerie client astraggono questa complessità, permettendo allo sviluppatore di interagire con il framework tramite funzioni native del linguaggio scelto [36].
Il client WebDriverIO, sebbene non sia mantenuto direttamente dal core team di Appium, è considerato uno standard de facto nell'ecosistema JavaScript per la sua perfetta integrazione e facilità di configurazione.

Tra le funzionalità distintive di WebDriverIO si annoverano:

- **Services**: moduli che estendono le capacità del client. Nello specifico, l'utilizzo del *@wdio/appium-service* permette di automatizzare la gestione del server Appium, avviandolo e chiudendolo in sincrono con l'esecuzione dei test, eliminando processi manuali soggetti a errori.
- **Reporters**: strumenti essenziali per la fase di debugging e analisi dei risultati. Il client ne mette a disposizione diversi tra cui scegliere, come ad esempio *Spec*, il reporter predefinito per un feedback immediato e testuale nel terminale durante lo sviluppo (figura 3.3), o *Allure*, uno dei reporters più completi che permette di generare report grafici avanzati, utili a fornire una visione d'insieme agli stakeholder tramite diagrammi e screenshot (figura 3.4) [37].

3.2.3 Automation Drivers

I driver rappresentano lo strato di astrazione finale. Essi sono implementati come classi *Node.js* che estendono una superclasse denominata *BaseDriver*. Il loro ruolo è convertire le richieste standard WebDriver in azioni sulla tecnologia di automazione proprietaria della piattaforma:

- Su **Android**: il driver comunica solitamente con il framework *UiAutomator2* di Google.
- Su **iOS**: il driver si interfaccia con la libreria *XCUITest* di Apple.

Ogni driver deve implementare i metodi previsti dal protocollo WebDriver, ma il contenuto logico di tali metodi è strettamente dipendente dalle API fornite dal produttore del sistema operativo, garantendo così che l'automazione avvenga nel modo più fedele possibile all'esperienza utente reale [38].

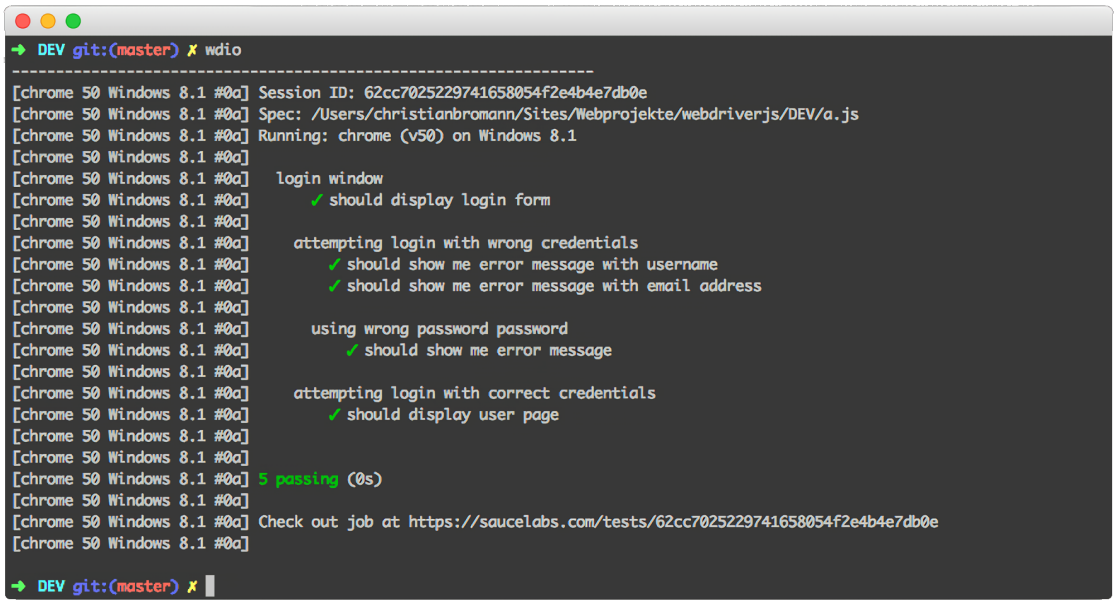


Figura 3.3: Spec Reporter di WebDriverIO [39]



Figura 3.4: Allure Reporter di WebDriverIO [40]

3.3 Appium Inspector

Lo strumento di test esplorativi per applicazioni mobile utilizzato in questo lavoro di tesi è Appium Inspector, un client grafico basato su WebDriverIO progettato specificamente per l'identificazione precisa e affidabile degli elementi dell'interfaccia grafica utente dell'*Application Under Test*.
Appium Inspector richiede una rigorosa catena di inizializzazione che rispecchia l'architettura client-server descritta precedentemente. Nello specifico, è necessario avviare in sequenza: il server Appium, il dispositivo (emulatore, simulatore o device fisico) e, in ultimo, la sessione del client Inspector. Questo ordine vincolante è dettato dal fatto che il client necessita di un server in ascolto e di un dispositivo pronto a ricevere i comandi di automazione per poter instaurare correttamente la sessione.
L'ambiente di lavoro di Appium Inspector è logicamente e visivamente suddiviso in due macro-aree: il *Session Builder* e il *Session Inspector* [41].

3.3.1 Session Builder

Il Session Builder (figura 3.5) è il modulo preposto all'avvio e alla configurazione delle sessioni di test, sia in ambiente locale che remoto (su cloud). L'instaurazione di una sessione richiede la definizione di un set di parametri di configurazione denominati *Desired Capabilities*.
Le capabilities sono coppie chiave-valore inviate dal client al server sotto forma di oggetto JSON durante la richiesta HTTP iniziale. Esse istruiscono il server Appium sulle caratteristiche dell'ambiente di test desiderato. I parametri possiedono una tipologia specifica (come ad esempio testo, booleano, numero o oggetto JSON) e includono attributi fondamentali e obbligatori per il routing corretto dei comandi, tra cui:

- La **piattaforma target**, ossia Android o iOS.
- Il **driver di automazione**, ossia UiAutomator2 o XCUITest.
- Il **percorso o l'identificativo dell'applicazione** da analizzare.
- Il **nome o l'identificativo del dispositivo** fisico o emulato su cui eseguire i test.

Al fine di ottimizzare i tempi di configurazione nei progetti multi-piattaforma, il Session Builder offre la funzionalità *Saved Capability Sets*, che consente di salvare e richiamare rapidamente diverse configurazioni di test a seconda dell'ambiente operativo che si intende verificare.

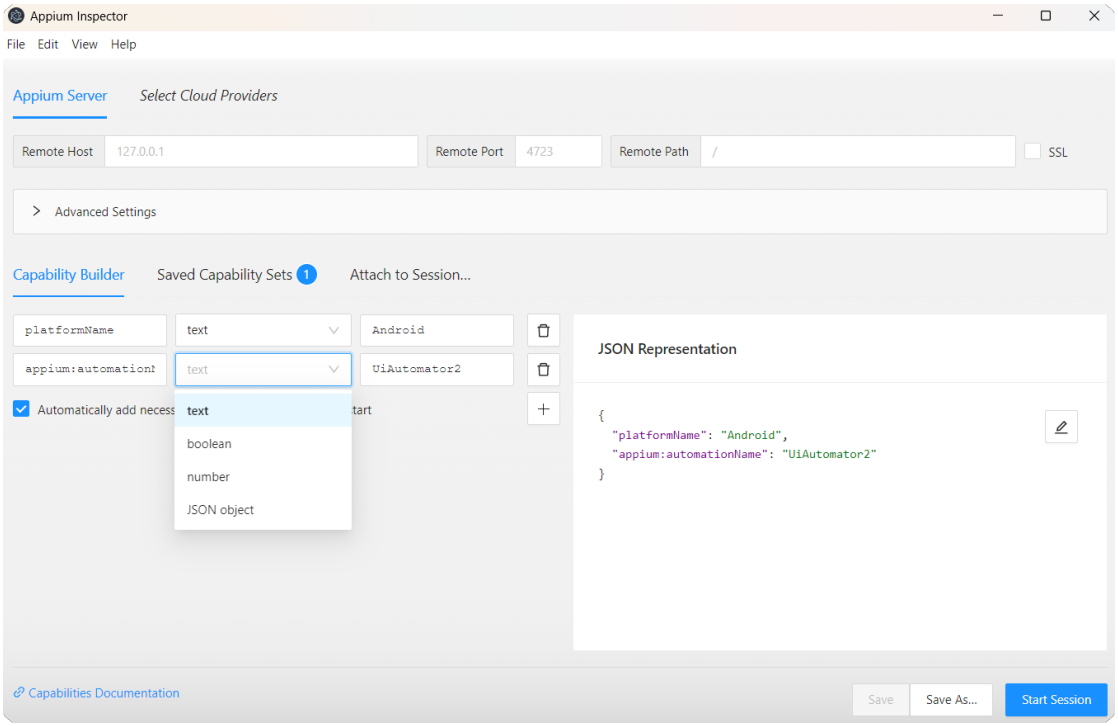


Figura 3.5: Session Builder di Appium Inspector

3.3.2 Session Inspector

Una volta che il server Appium ha accettato le capabilities e avviato la sessione sul dispositivo, l'utente accede al Session Inspector. Quest'area rappresenta il fulcro operativo dell'applicativo ed è organizzata in sezioni modulari, ciascuna adibita a una specifica funzionalità di analisi o interazione:

- **Header** (figura 3.6): raggruppa i comandi globali per gestire il flusso del dispositivo (come ad esempio il refresh dell'interfaccia) e per navigare la struttura XML. Include anche strumenti e comandi avanzati per cercare elementi specifici nella struttura XML (utilizzando selettori tra cui XPath, ID, e Accessibility ID), per la registrazione delle azioni dell'utente, e per scaricare un report della sessione corrente.

- **Screenshot Panel** (figura 3.7): fornisce una rappresentazione visiva in tempo reale della schermata corrente dell'*Application Under Test*. Inoltre, il pannello è interattivo: cliccando sui componenti grafici renderizzati, l'applicazione sincronizza e isola automaticamente le informazioni strutturali corrispondenti negli altri pannelli.

- **Source Tab** (figura 3.8): costituisce il pannello più rilevante ai fini dell'identificazione degli elementi grafici (definiti anche come *widgets*). Si suddivide in due ulteriori pannelli: l'*Application Source Panel*, che mostra l'albero gerarchico della GUI, rappresentato sotto forma di codice XML, e il *Selected Element Panel*, che dettaglia gli attributi del nodo XML selezionato, suggerendo i selettori più idonei per rintracciare univocamente il componente tramite codice.
- **Commands Tab** (figura 3.9): consente di inviare istruzioni dirette al driver in uso per interrogare lo stato dell'applicazione o forzare azioni specifiche. Poiché molti comandi riflettono le API native, alcune funzionalità potrebbero essere specifiche per Android e non supportate su iOS, o viceversa.
- **Gestures Tab** (figura 3.10): un'interfaccia dedicata alla costruzione di interazioni tattili complesse. Sfruttando la specifica API *W3C Actions*, permette di concatenare primitive come *Pointer Down*, *Move*, *Pause* e *Pointer Up* per orchestrare gesture avanzate, quali *swipe*, *scroll* o *pinch-to-zoom*, testandone l'effetto in tempo reale.
- **Recorder Tab** (figura 3.11): strumento che traccia le interazioni manuali dell'utente sull'interfaccia grafica e le converte automaticamente in script di automazione, offrendo l'esportazione del codice in vari linguaggi di programmazione. Sebbene la logica prodotta possa necessitare di refactoring, la registrazione riduce drasticamente i tempi di scrittura dei test iniziali, rivelandosi particolarmente vantaggiosa in ambienti dove la GUI è soggetta a iterazioni frequenti.
- **Session Information Tab** (figura 3.12): riepiloga i metadati della connessione in corso, mostrando diverse informazioni, tra cui l'ID univoco della sessione, gli endpoint utilizzati o il payload delle desired capabilities effettivamente applicate.



Figura 3.6: Header del Session Inspector di Appium Inspector

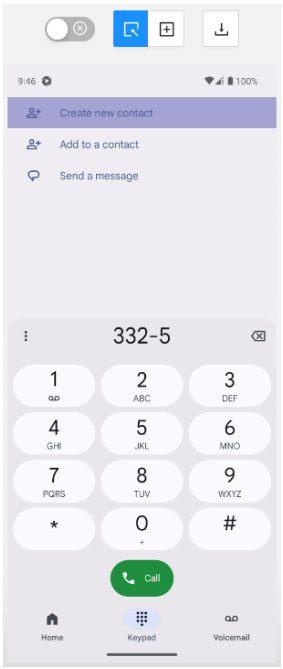


Figura 3.7: Screenshot Panel del Session Inspector di Appium Inspector

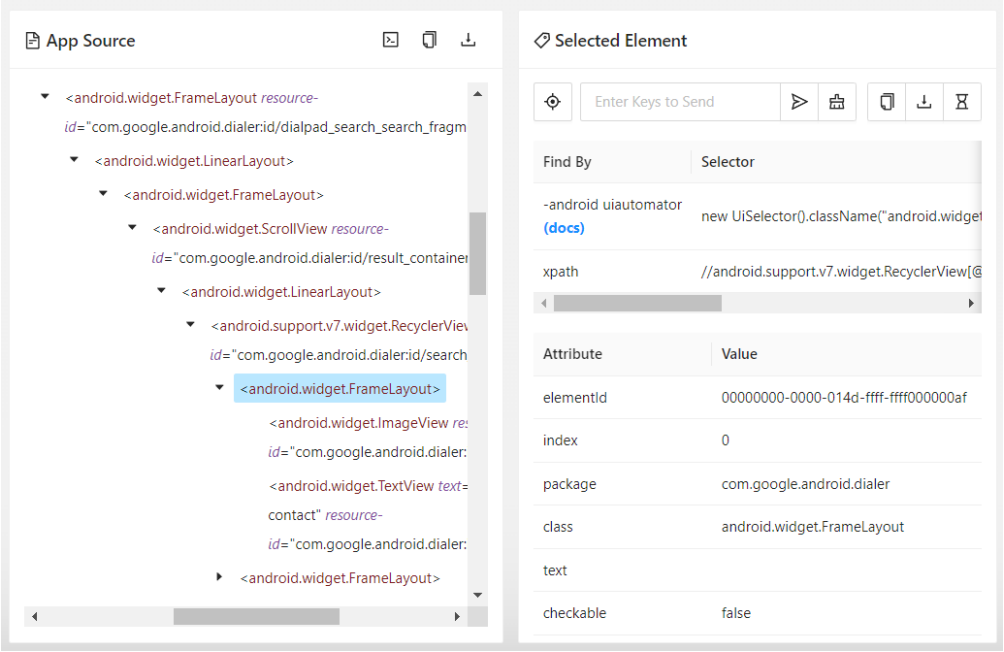


Figura 3.8: Source Tab del Session Inspector di Appium Inspector

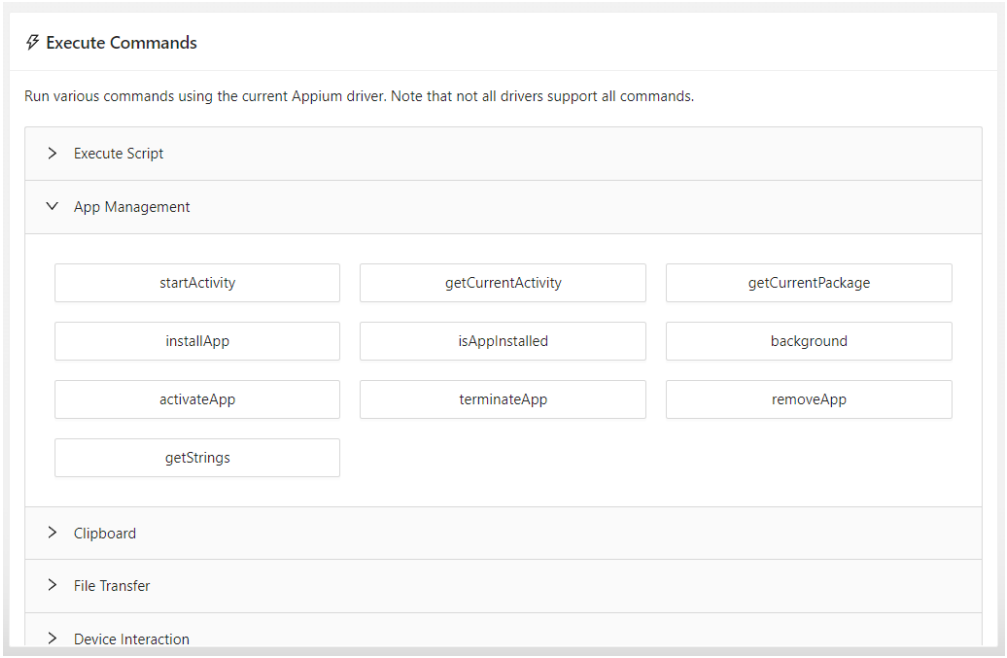


Figura 3.9: Commands Tab del Session Inspector di Appium Inspector

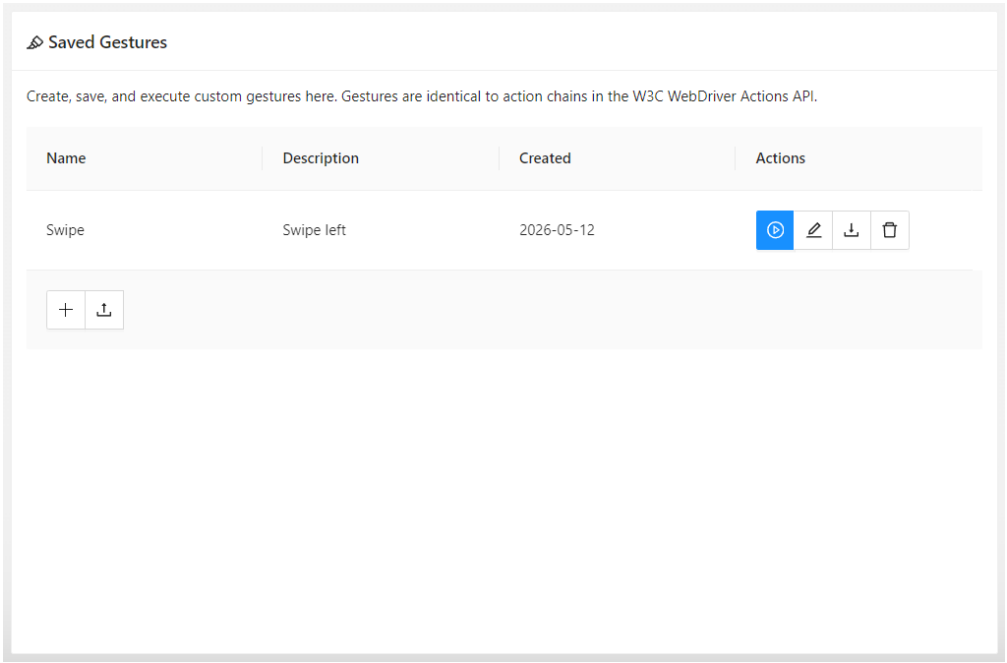


Figura 3.10: Gestures Tab del Session Inspector di Appium Inspector

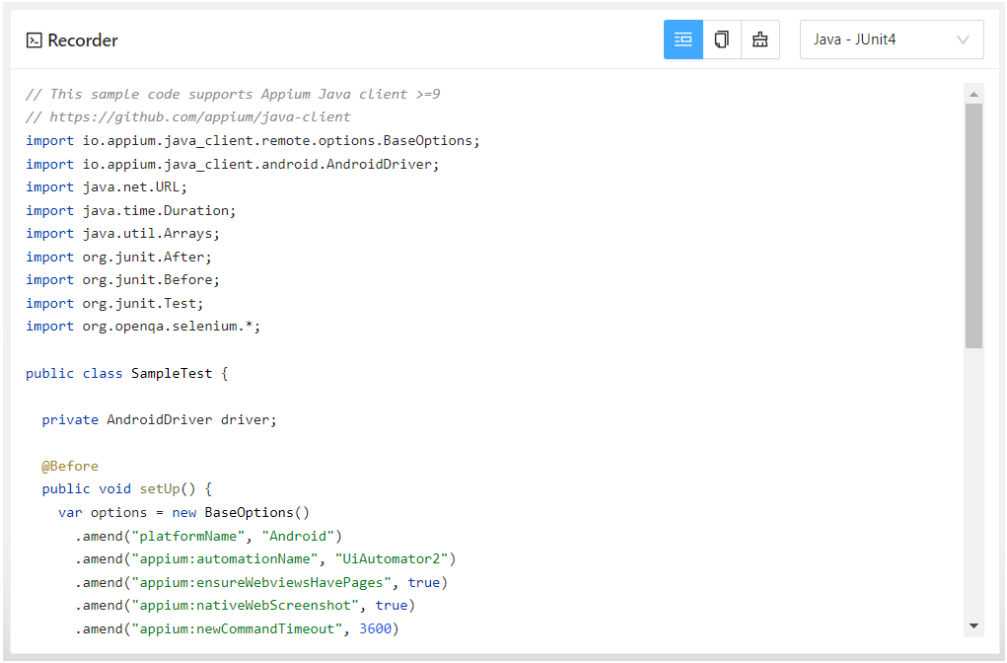


Figura 3.11: Recorder Tab del Session Inspector di Appium Inspector

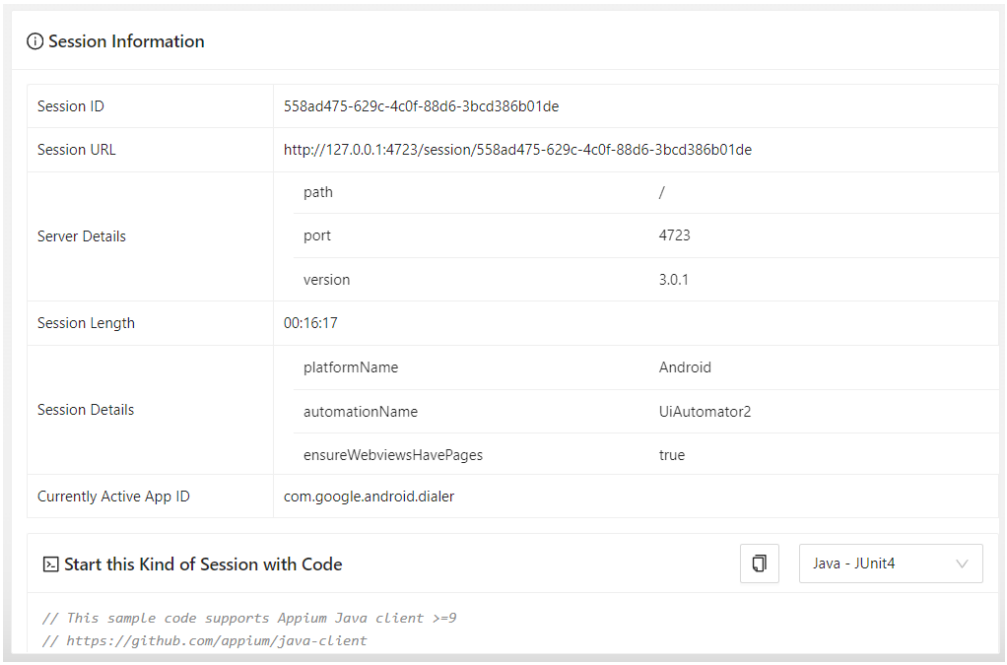


Figura 3.12: Session Information Tab del Session Inspector di Appium Inspector

3.4 GAppium

GAppium è un prototipo di strumento per il testing esplorativo "gamificato" su dispositivi mobile, sviluppato originariamente da Lorenzo Laudadio et al. [7]. Nello specifico, esso estende le funzionalità esistenti di Appium Inspector aggiungendo un'ulteriore sezione, ossia la *Gamification Tab* (figura 3.13). Quest'ultima introduce meccaniche di gioco all'interno dell'ambiente di test, con l'obiettivo di aumentare il coinvolgimento e la motivazione dei tester durante l'esplorazione dell'interfaccia grafica.

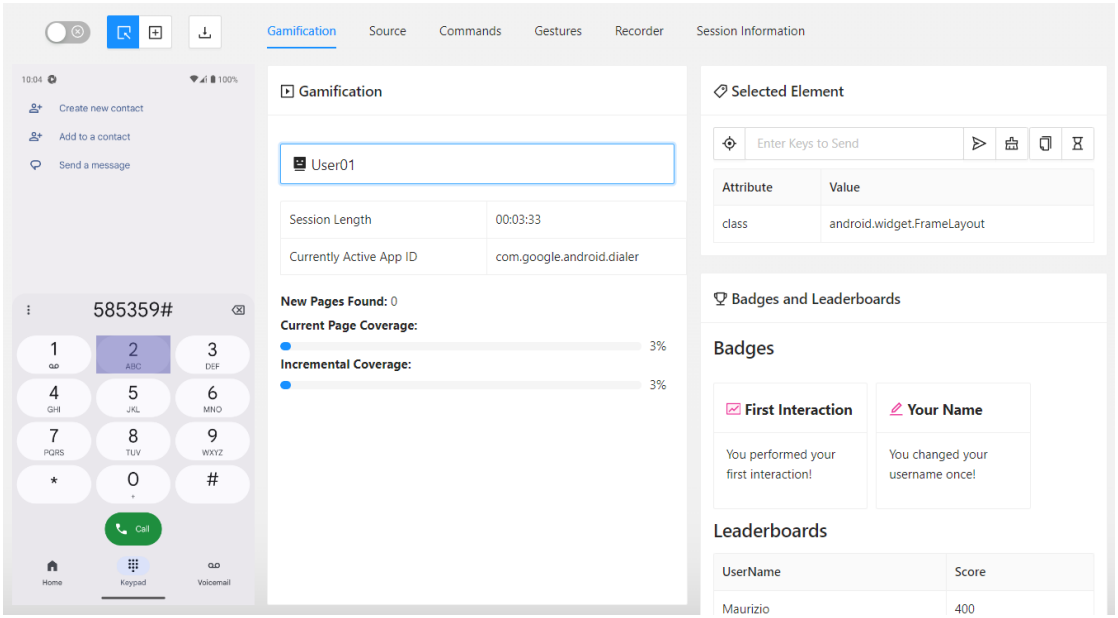


Figura 3.13: Overview della Gamification Tab di GAppium

Tuttavia, essendo GAppium un prototipo sperimentale, la sua implementazione è limitata alle seguenti meccaniche ludiche di base:

- **Selezione di un username** per mezzo di un input di testo.
- **Assegnazione di punti** interagendo con i vari *widgets* (gli elementi grafici della GUI) per la prima volta, o scoprendo nuove **"pagine"** (ossia le diverse schermate dell'*Application Under Test*).
- **Ottenimento di badges** (trofei), come ad esempio all'interazione con il primo widget o dopo aver esplorato un certo numero di "pagine".
- **Prototipo di una classifica**, che mostra solo un paio di utenti preimpostati con punteggi fittizi.

- **Barra di avanzamento** per mostrare la percentuale di *widgets* interagiti nella "pagina" corrente, e la percentuale di *widgets* interagiti in totale.
- **Contatore** del numero di "pagine" scoperte durante la sessione corrente.

Queste semplici funzionalità possono già influenzare positivamente il modo in cui i tester progettano i casi di test, portando a una maggiore copertura dei test e a una maggiore efficienza nell'individuazione dei bug [6].
Ciò nonostante, è importante sottolineare che queste meccaniche presentano diverse limitazioni, tra cui la mancanza di un sistema di progressione più articolato o la limitata varietà di badges disponibili. Inoltre, ulteriori meccaniche possono essere implementate seguendo i principi teorici e applicativi della gamification, approfonditi nel capitolo precedente.
Questa marcata parzialità è visivamente evidente analizzando il seguente diagramma (figura 3.14). A differenza dell'ottagono regolare, espanso e bilanciato teorizzato da Yu-kai Chou (figura 2.5), il quale rappresenta un'esperienza gamificata in grado di stimolare l'utente a 360 gradi, quello di GAppium risulta fortemente sbilanciato e asimmetrico. Infatti, il limite più significativo di GAppium è rappresentato dal fatto che copre, parzialmente, solo il *Core Drive 2: Development & Accomplishment*, nell'assegnazione di punti o badges, e il *Core Drive 4: Ownership & Possession*, nella possibilità di selezionare un username, dell'Octalysis Framework.

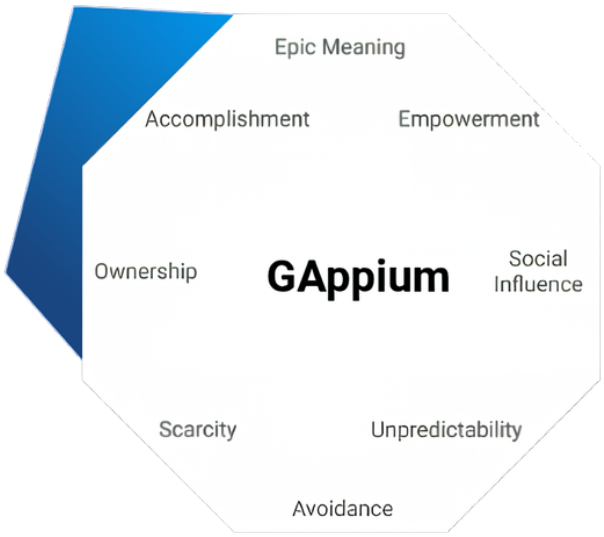


Figura 3.14: Valutazione di GAppium basata sull'Octalysis Framework

Capitolo 4

Descrizione del progetto

Alla luce delle premesse teoriche delineate e, in particolare, partendo dalle limitazioni intrinseche del prototipo sperimentale GAppium evidenziate precedentemente, questo capitolo costituisce il fulcro operativo e pratico del presente lavoro di tesi. Se da un lato le funzionalità ludiche introdotte inizialmente hanno dimostrato il potenziale della gamification nell'influenzare positivamente la progettazione e la copertura dei casi di test, dall'altro la loro natura acerba ha lasciato inesplorate ampie aree motivazionali dell'Octalysis Framework.

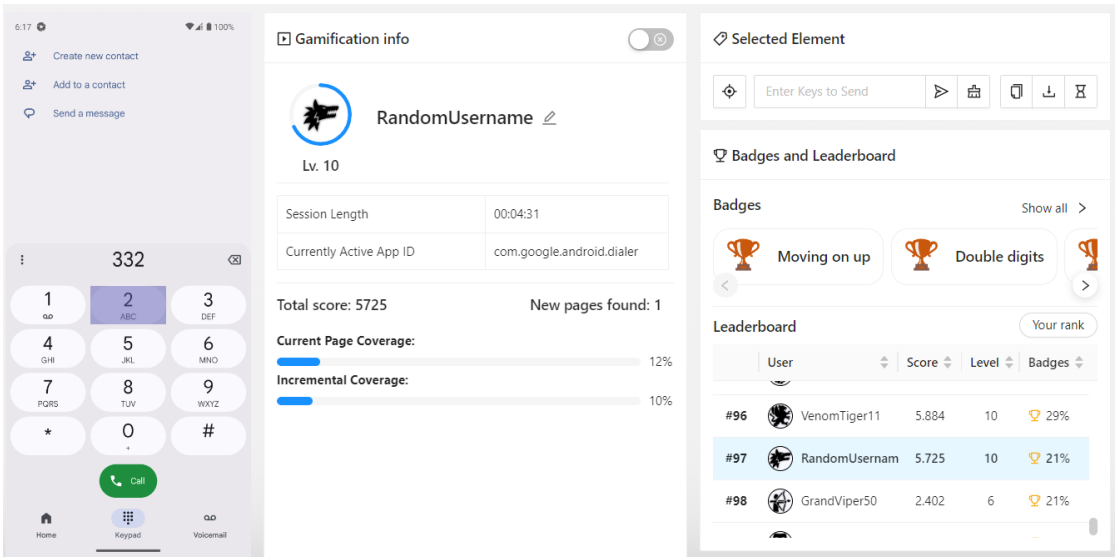


Figura 4.1: Overview della Gamification Tab nella nuova estensione realizzata

Nelle sezioni che seguono verrà descritta nel dettaglio la progettazione e l'implementazione della nuova estensione "gamificata" realizzata (figura 4.1), illustrando come il prototipo iniziale sia stato espanso e perfezionato per offrire un'esperienza più completa e stimolante per i tester.

4.1 Quality of life

Il primo obiettivo perseguito è stato quello di migliorare la qualità dell'esperienza utente all'interno dell'applicazione. A tal fine, è stata introdotta una nuova funzionalità volta a semplificare e ottimizzare il flusso di lavoro dei tester, e sono state migliorate alcune funzionalità esistenti che presentavano delle problematiche/limitazioni.

4.1.1 Gestione dei widgets interattivi

Un aspetto cruciale per il corretto funzionamento delle dinamiche di gamification all'interno dell'estensione riguarda la classificazione e la gestione degli elementi grafici della GUI, comunemente definiti *widgets*.
L'interfaccia utente di un'applicazione mobile, così come interpretata e restituita dal server Appium, è strutturata come un albero gerarchico in formato XML o JSON di elevata complessità. Tale albero è intrinsecamente eterogeneo: esso comprende non solo gli elementi con cui l'utente può effettivamente interagire (come ad esempio pulsanti, campi di testo o menù a tendina), ma anche una vasta maggioranza di nodi puramente strutturali, invisibili e, di conseguenza, non cliccabili. Questi ultimi (come ad esempio i *LinearLayout* o i *ViewGroup*) sono privi di qualsiasi semantica esplorativa per i tester.

Nella prima versione sperimentale di GAppium, il calcolo delle metriche ludiche teneva conto dell'intero ammontare dei *widgets* presenti nell'albero della GUI. Questo approccio generava due problematiche fondamentali:

- **Diluizione delle metriche**: se, ad esempio, una schermata conteneva 100 *widgets* totali ma solo 5 pulsanti interagibili, la barra di avanzamento calcolata non avrebbe mai potuto raggiungere il 100%, generando frustrazione nei tester e vanificando l'effetto motivazionale della gamification.

- **Assegnazione impropria di punteggi**: un sistema non adeguatamente filtrato avrebbe potuto premiare interazioni accidentali o tecnicamente impossibili su elementi strutturali.

Per risolvere queste criticità, il sistema di gestione dei *widgets* è stato ricalibrato implementando una logica di filtraggio selettivo. L'algoritmo attuale analizza ricorsivamente la struttura della "pagina" e scarta a priori tutti i nodi statici, concentrando il tracciamento e la memorizzazione esclusivamente sui *widgets* interattivi.
Da un punto di vista tecnico, l'estensione identifica la rilevanza di un *widget* verificando, per mezzo delle funzionalità di Appium, la presenza e il valore dell'attributo *clickable*.

Ciò nonostante, poiché nella GUI sono presenti numerosi *widgets* strutturali non interagibili, quando i tester cliccano su un'icona o su un testo attraverso lo Screenshot Panel di Appium Inspector, il sistema seleziona il nodo figlio specifico che probabilmente non risulterà *clickable*, impedendo quindi l'assegnazione dei punti o l'attivazione di altre meccaniche di gamification, sebbene l'azione simulata sia teoricamente valida. Per superare questo limite architetturale, è stata implementata una logica di ricerca ascendente (*bottom-up*) nell'albero JSON, gestita dalle seguenti funzioni (figura 4.2):

- ***getParentPath***: analizza la stringa che rappresenta il percorso gerarchico del nodo corrente (ad esempio 0.1.4) e, manipolandola, estrae il percorso del nodo genitore immediato (ad esempio 0.1). Questo meccanismo permette di risalire la catena gerarchica partendo da qualsiasi punto della foglia verso la radice.

- ***getNearestClickable***: quando il tester seleziona un nodo, la funzione avvia un ciclo iterativo. In prima istanza, verifica se il nodo selezionato è direttamente interattivo. Se lo è, la ricerca termina con successo e restituisce il nodo stesso. Altrimenti, la funzione invoca *getParentPath* per ricavare il riferimento al nodo padre. Utilizzando una mappa pre-indicizzata dell'albero (*nodesMap*), recupera l'oggetto nodo corrispondente al genitore e reitera il controllo sull'attributo *clickable*.

Questa riprogettazione della gestione dei *widgets* ha un impatto diretto e fondamentale sul calcolo delle metriche relative alla gamification, garantendo un'esperienza utente equa e gratificante. Inoltre, conservare in memoria e analizzare solo gli elementi interattivi riduce la dimensione dei dati da elaborare, migliorando la reattività dell'estensione durante le sessioni di testing.

```
/**
 * Gets the parent path from a node path string
 */
function getParentPath(path) : ... { Show usages  Alessandro Costantino
  if (path === '')
    return null;   // Root node has no parent

  if (path.lastIndexOf('.') === -1)
    return '';     // Parent is the root node

  return path.substring(0, path.lastIndexOf('.'));
}

/**
 * Finds the nearest clickable node (starting from startNode and moving up the sourceJSON tree)
 */
export function getNearestClickable(nodesMap, startNode) : any | null { Show usages  Alessandro Costantino
  let current = startNode;

  while (current) {
    if (current.attributes && (current.attributes.clickable === "true" || current.attributes.clickable === true))
      return current;

    const parentPath : string | any = getParentPath(current.path);

    if (parentPath === null)
      return null;   // We reached the root

    current = nodesMap.get(parentPath);
  }

  return null;   // No clickable parent found
}
```

Figura 4.2: Funzioni di gestione dei widgets interattivi

4.1.2 Algoritmo di similarità delle pagine

Nella prima versione sperimentale di GAppium è emersa una criticità significativa legata al tracciamento dell'esplorazione: qualsiasi minima modifica all'albero degli elementi della schermata corrente (come ad esempio, l'apertura di un menù a tendina, la comparsa di un indicatore di caricamento, o il cambio di stato di una checkbox) veniva erroneamente interpretata dal sistema come l'individuazione di una nuova "pagina". Questo approccio, eccessivamente sensibile alle variazioni del DOM, generava un elevato numero di falsi positivi, alterando sia il calcolo del punteggio sia la percezione del progresso da parte dei tester.
Per risolvere questa anomalia e garantire una mappatura più fedele delle *Applications Under Test*, è stato implementato un algoritmo di similarità strutturale in grado di distinguere in modo piuttosto accurato e robusto i reali cambi di "pagina" dalle semplici transizioni di stato interne a una medesima schermata.

L'algoritmo implementato si basa sull'**Indice di Jaccard** (noto anche come *Coefficiente di Jaccard* o *Intersection over Union*), una metrica statistica consolidata per misurare la similarità e la diversità tra due insiemi finiti. Da un punto di vista matematico, l'indice è definito come il rapporto tra la dimensione dell'intersezione e la dimensione dell'unione di due insiemi A e B [42].

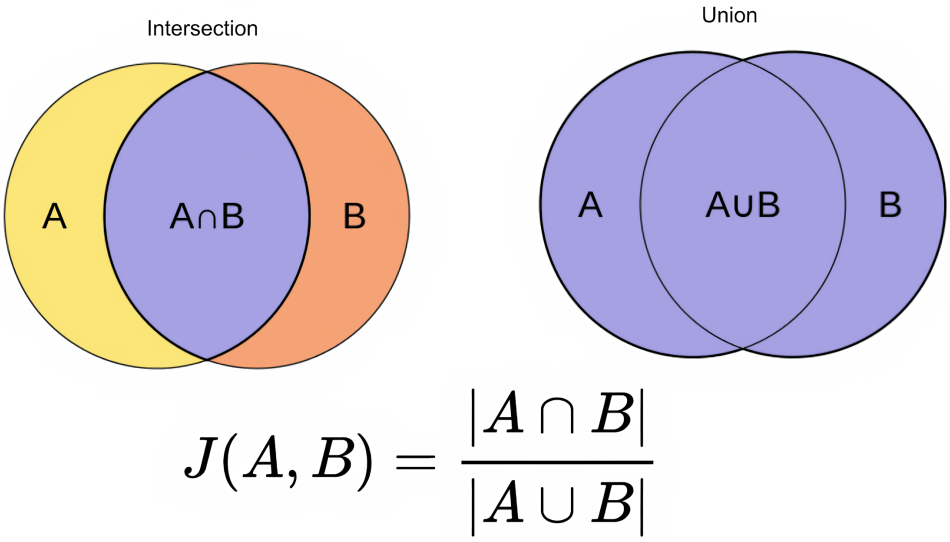


Figura 4.3: Indice di Jaccard

In questo caso, gli insiemi A e B rappresentano due collezioni di attributi e proprietà estratte dai *widgets*. L'insieme A racchiude le proprietà (nello specifico, ID univoci, classi e descrizioni testuali) della schermata precedente all'interazione del tester, mentre l'insieme B contiene le proprietà della schermata risultante successiva all'interazione. Confrontando queste due collezioni, viene calcolato un valore di similarità compreso tra 0 (insiemi completamente disgiunti) e 1 (insiemi identici). Se il valore $J(A, B)$ supera o eguaglia una soglia di tolleranza definita empiricamente, l'algoritmo deduce che i cambiamenti strutturali sono minimi. Di conseguenza, la schermata viene classificata come uno stato aggiornato della "pagina" corrente e non come una nuova "pagina" scoperta.

L'implementazione pratica dell'algoritmo si articola in diverse funzioni che analizzano l'albero gerarchico della GUI restituito da Appium Inspector:

- **Identificazione dei widgets** (figura 4.4): per evitare che modifiche irrilevanti sfalsino la rilevazione, la funzione *buildWidgetId* costruisce un identificativo univoco per ogni *widget* basandosi sui suoi attributi, scartando proprietà volatili come le coordinate spaziali (*bounds*). Parallelamente, le funzioni

collectWidgets e *countWidgets* ispezionano ricorsivamente l'albero per isolare e contare esclusivamente i *widgets* interattivi.

- **Identificazione delle pagine** (figura 4.6): per poter confrontare due schermate, è necessario astrarne la struttura. La funzione *fingerprint* genera un identificativo dell'albero concatenando gerarchicamente i *resource-id* dei nodi. Inoltre, per popolare gli insiemi richiesti dall'Indice di Jaccard, la funzione *collectStructuralHistogram* estrae da ogni nodo una combinazione univoca di attributi primari (come *class* e *resource-id*) o secondari (come *text* e *content-desc* se l'ID è assente), inserendoli in un costrutto dati.

- **Calcolo della similarità** (figura 4.5): la funzione *treeSimilarity* esegue il calcolo matematico vero e proprio. Ricevendo in input i due alberi JSON (pre e post interazione del tester), ne estrae gli insiemi strutturali e ne calcola l'intersezione e l'unione. Il rapporto tra questi due valori restituisce la percentuale di similarità.

- **Gestione delle pagine** (figura 4.7): Il cuore logico dell'algoritmo risiede nella funzione *buildPageId*. Essa valuta il valore di similarità rispetto alla soglia predefinita. Se la similarità è superiore o uguale alla soglia, la "pagina" esistente viene aggiornata, aggiungendo eventuali nuovi *widgets* scoperti a seguito del cambio di stato. Altrimenti, se la similarità è inferiore alla soglia, viene istanziata una nuova "pagina" (purché non sia una già visitata, e di conseguenza creata, in precedenza).

La scelta dell'Indice di Jaccard per la misurazione della similarità strutturale della GUI è supportata dal suo pregresso successo nella letteratura del software testing [43]. Le ragioni principali che hanno guidato questa decisione progettuale includono:

- **Efficienza computazionale**: l'algoritmo presenta una complessità temporale lineare pari a $O(N)$, aspetto fondamentale in uno strumento di testing esplorativo reattivo poiché il calcolo avviene in tempo reale e non deve introdurre fastidiose latenze tra un'azione e l'altra [44].

- **Semplicità e Interpretabilità**: rispetto a modelli basati su reti neurali o ad algoritmi di Computer Vision, il calcolo insiemistico offre risultati trasparenti, predicibili e facilmente interpretabili in fase di debugging [44].

- **Confronto con alternative più onerose**: esistono algoritmi di similarità indubbiamente più precisi, come il *Tree Edit Distance*, che calcola il numero minimo di operazioni (inserimenti, cancellazioni, sostituzioni) necessarie per trasformare un albero JSON in un altro. Tuttavia, esso richiede generalmente una complessità temporale polinomiale (spesso $O(N^3)$), un costo prestazionale sproporzionato rispetto alle reali necessità di calcolo dell'estensione [45]. Anche

altre metriche, come la *Cosine Similarity*, pur essendo efficaci, risultano meno dirette per insiemi di attributi testuali o categorici tipici dell'albero della GUI rispetto all'immediatezza dell'intersezione insiemistica offerta dall'Indice di Jaccard [46].

```
// Build widget ID from its attributes
export const buildWidgetId = (attributes) : string  => {  Show usages  Alessandro Costantino
  if (!attributes)
    return "";

  const entries : ([string, unknown])[]  = Object.entries(attributes)
    .filter(([k]) : boolean  => k !== "bounds")
    .sort( compareFn: ([a : string ], [b : string ]) => a.localeCompare(b));

  return entries.map(([k : string , v]) : string  => `${k}=${v}`).join("|");
};
// Recursively collect all clickable widget IDs
  const collectWidgets = (json, widgetList : any[]  = []) : any[]  => {  Show usages  Alessandro Costantino
    if (!json || !json.attributes)
      return widgetList;

    if (json.attributes && (json.attributes.clickable === "true" || json.attributes.clickable === true))
      widgetList.push(buildWidgetId(json.attributes));

    (json.children || []).forEach((c) : any[]  => collectWidgets(c, widgetList));
    return widgetList;
  };
// All widgets may be interacted with Appium Inspector, but we are only interested in the clickable ones
  const countWidgets = (json) : number  => {  Show usages  Alessandro Costantino *
    let sum : number  = 0;
    if (json.attributes && (json.attributes.clickable === "true" || json.attributes.clickable === true)) {
      sum += 1;
    }
    if (json.children) {
      for (const child of json.children) {
        sum += countWidgets(child);
      }
    }
    return sum;
  };
```

Figura 4.4: Funzioni di identificazione dei widgets

```
// Compute similarity between two json trees (Jaccard index => intersection over union of class + resource-id counts)
  const treeSimilarity = (treeA, treeB) => {  Show usages  Alessandro Costantino
    if (!treeA || !treeB) {
      return 0; // One or both trees are empty
    }

    // Get the set of unique structural elements from each tree
    const signatureA : Set<any>  = collectStructuralHistogram(treeA);
    const signatureB : Set<any>  = collectStructuralHistogram(treeB);

    // Calculate Jaccard Index (Intersection over Union) for the sets
    let intersectionSize : number  = 0;
    for (const elementSig of signatureA) {
      if (signatureB.has(elementSig)) {
        intersectionSize++;
      }
    }

    const unionSize : number  = signatureA.size + signatureB.size - intersectionSize;

    if (unionSize === 0) {
      // Both sets are empty, so they are identical
      return 1;
    }

    return intersectionSize / unionSize;
  };
```

Figura 4.5: Funzione di calcolo della similarità

```
// Build a structural fingerprint with the 'resource-ids' to utilize as the pageId
  const fingerprint = (json) : ... => {  Show usages  Alessandro Costantino
    if (!json) {
      return [];
    }

    const attributes : {} = json.attributes || {};
    const children : any[] = json.children || [];
    const childrenFingerprints : unknown[] = children.flatMap( callback: child => fingerprint(child));

    const id : string = attributes['resource-id'];

    if (id) {
      // This node has an id
      childrenFingerprints.sort(); // Sort for order independence
      const nodeString : string = id + "(" + childrenFingerprints.join(",") + ")";
      return [nodeString];
    } else {
      // This node has no id
      return childrenFingerprints;
    }
  };
// Collect class + resource-id counts from json tree
  const collectStructuralHistogram = (tree, signatureSet : Set<any> = new Set()) : Set<any> => {  Show usages  Alessandro Cos
    if (!tree || !tree.attributes) {
      return signatureSet;
    }

    const className : string = tree.attributes.class || "unknown";
    const resourceId : string = tree.attributes['resource-id'] || "no-id";
    let elementSignature;

    if (resourceId !== "no-id") {
      elementSignature = `${className}|${resourceId}`;
    } else {
      const text : string = tree.attributes.text || "";
      const contentDesc : string = tree.attributes['content-desc'] || "";
      elementSignature = `${className}|${resourceId}|${text}|${contentDesc}`;
    }
    signatureSet.add(elementSignature);

    // Recurse through children
    if (tree.children && Array.isArray(tree.children)) {
      tree.children.forEach(child => collectStructuralHistogram(child, signatureSet));
    }

    return signatureSet;
  };
```

Figura 4.6: Funzioni di identificazione delle pagine

```
// Algorithm to build a page (screen) id from its content and check similarity with last page
  const buildPageId = (json) : unknown  => {  Show usages  Alessandro Costantino
    if (!json)
      return "unknown_page";

    const widgetsId : any[]  = collectWidgets(json);
    const pageId : string  = fingerprint(json).join(",");

    if (lastTree === null) {
      // First screen
      setLastTree(json);
      setLastPageId(pageId);
      const newPage : {...}  = {
        pageId: pageId,
        widgetsId: widgetsId,
        widgetIndex: Object.fromEntries(widgetsId.map((w) : [any, boolean]  => [w, true])),
        instances: [pageId],
        nInteractableWidgets: countWidgets(json),
        nInteractedWidgets: 0,
        interactedWidgetsId: []
      };
      addPage(newPage);
      return pageId;
    }

    const matchedPageId = pages.find(p => p.instances.includes(pageId))?.pageId;
    if (matchedPageId) {
      // Already visited screen
      setLastTree(json);
      setLastPageId(matchedPageId);
      return matchedPageId;
    }
    const sim = treeSimilarity(lastTree, json);
    if (sim >= PAGE_SIMILARITY_THRESHOLD) {
      // Same screen
      setLastTree(json);
      const page = pages.find(p => p.pageId === lastPageId);
      for (const w of widgetsId) {
        if (!page.widgetIndex[w])
          updatePageWidgets(w, lastPageId);
      }
      updatePageInstances(lastPageId, pageId);
      return lastPageId;
    } else {
      // Different new screen
      setLastTree(json);
      setLastPageId(pageId);
      const newPage : {...}  = {
        pageId: pageId,
        widgetsId: widgetsId,
        widgetIndex: Object.fromEntries(widgetsId.map((w) : [any, boolean]  => [w, true])),
        instances: [pageId],
        nInteractableWidgets: countWidgets(json),
        nInteractedWidgets: 0,
        interactedWidgetsId: []
      };
      addPage(newPage);
      return pageId;
    }
  };
```

Figura 4.7: Funzione principale dell'algoritmo di similarità

4.1.3 Scelte in evidenza

Nelle sessioni prolungate di testing esplorativo, uno dei principali ostacoli per i tester è rappresentato dal sovraccarico cognitivo. All'aumentare della complessità dell'interfaccia e della profondità di navigazione, diventa sempre più difficile memorizzare quali elementi grafici siano già stati esaminati e quali, invece, siano stati trascurati. Per risolvere questa criticità e ottimizzare il flusso di lavoro, è stata implementata una nuova funzionalità di assistenza visiva direttamente integrata nell'interfaccia grafica dell'estensione.
La funzionalità si concretizza nell'applicazione di un overlay grafico sullo Screenshot Panel di Appium Inspector. Il sistema, sfruttando la mappatura degli elementi interattivi precedentemente descritta, genera dinamicamente dei marcatori visivi (specificamente, dei riquadri dai bordi di colore arancione) che circondano il perimetro spaziale (*bounding boxes*) dei *widgets* con i quali l'utente non ha ancora interagito.

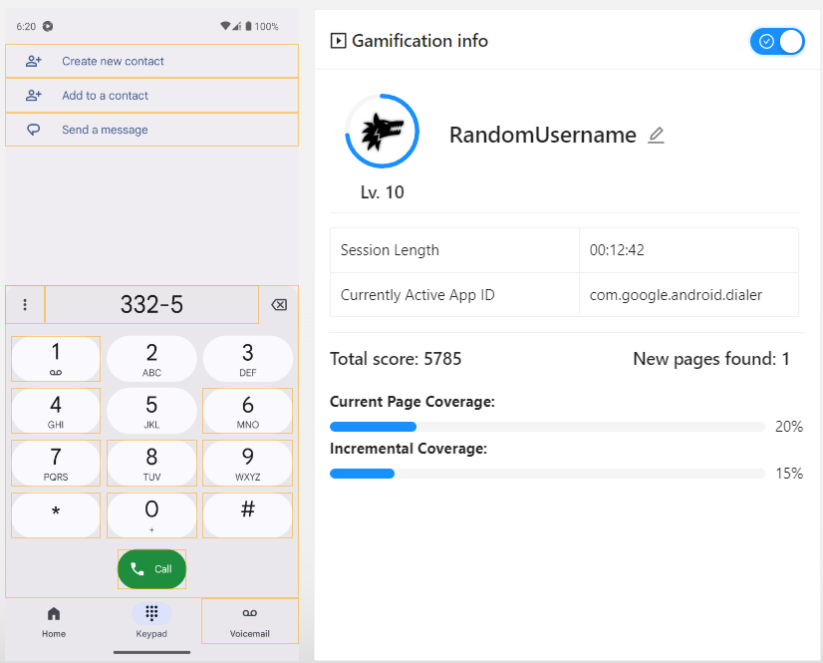


Figura 4.8: Funzionalità *Scelte in evidenza* attivata

Dal punto di vista implementativo, la creazione di questi marcatori è gestita da uno dei componenti di Appium Inspector dedicato al rendering dei riquadri di evidenziazione quando gli utenti interagiscono con lo Screenshot Panel (figura 4.9). Il codice utilizza le proprietà spaziali dei *widgets* (width, height, left, top) per mappare e posizionare i contenitori colorati esattamente sopra i *widgets* esaminati dal tester nello screenshot dell'*Application Under Test*.

Il cuore logico della funzionalità risiede nell'assegnazione condizionale delle classi di stile (CSS). L'algoritmo valuta specifiche espressioni booleane per determinare se applicare o meno i marcatori, compreso questo relativo alla gamification. Affinché la classe *orange-border-active* venga applicata, l'utente deve aver abilitato la relativa funzionalità di gamification, il *widget* analizzato deve essere effettivamente un elemento interattivo, e l'identificativo univoco del *widget* non deve essere presente all'interno della collezione che tiene traccia degli elementi già esplorati nella schermata corrente.

```
const HighlighterRectForElem = (props) => {  Show usages  Alessandro Costantino
  const {width, height, left, top, clickable, wId} = dimensions;
  const key = element.path;
  let highlighterClasses : any[]  = [InspectorCSS['highlighter-box']];
  // Add class + special classes to hovered, selected elements, and clickable elements
  if (hoveredElement.path === element.path) {
    highlighterClasses.push(InspectorCSS['hovered-element-box']);
  }
  if (selectedElement.path === element.path) {
    highlighterClasses.push(InspectorCSS['inspected-element-box']);
  }
  const currentPage = pages.find(p => p.pageId === currentPageId);
  if (
    showClickable &&
    String(clickable) === 'true' &&
    !currentPage.interactedWidgetsId.some((wid : T ) : boolean  => wid === wId)
  ) {
    highlighterClasses.push(InspectorCSS['orange-border-active']);
  }

  return (
    <div
      className={highlighterClasses.join(' ').trim()}
      onMouseOver={() => selectHoveredElement(key)}
      onMouseOut={unselectHoveredElement}
      onClick={() : any  => (key === selectedElement.path ? unselectElement() : selectElement(key))}
      key={key}
      style={{left: left || 0, top: top || 0, width: width || 0, height: height || 0}}
    >
      <div></div>
    </div>
  );
};
```

Figura 4.9: Funzione di rendering per la funzionalità *Scelte in evidenza*

Oltre agli evidenti vantaggi pratici, l'introduzione di questi marcatori visivi rappresenta, secondo i principi dell'Octalysis Framework, l'implementazione diretta della meccanica di gamification *Scelte in evidenza*, che attiva il *Core Drive 7: Unpredictability & Curiosity*. Nello specifico, questa meccanica, evidenziando esplicitamente un percorso inesplorato, cattura l'attenzione e stimola la curiosità dell'utente.

4.2 Livelli, punteggi e feedback

Nel prototipo iniziale di GAppium, l'assegnazione dei punti risultava essere una meccanica prettamente quantitativa e transazionale, priva di un reale senso di evoluzione a lungo termine. Per superare questo limite e trasformare l'esplorazione da un'attività monotona a un'esperienza gratificante e continuativa, l'architettura logica dell'estensione è stata rinnovata attraverso l'implementazione di un sistema di progressione strutturato e di meccanismi di feedback dinamico.
Il fulcro di questa evoluzione è l'introduzione di un sistema a *livelli*, che permette all'utente di progredire dal livello 1 fino al livello 50, in base ai punti accumulati. Parallelamente, la logica di calcolo del punteggio è stata ricalibrata per premiare in modo proporzionale la diversità e la qualità delle azioni di testing. Ad esempio, è possibile ottenere punti scoprendo e interagendo con nuovi *widgets*, individuando nuove "pagine" e sbloccando badges, ma anche completando attività/obiettivi specifici. Questa struttura è stata progettata per massimizzare l'attivazione del *Core Drive 2: Development & Accomplishment*. Suddividendo l'obiettivo finale in piccoli obiettivi incrementali, il sistema restituisce ai tester una percezione tangibile della propria crescita e del superamento delle sfide.

Oltre all'infrastruttura logica dei punti, un'attenzione particolare è stata dedicata all'interfaccia e all'esperienza utente dell'estensione. Sono state infatti introdotte delle specifiche *animazioni* visive che si attivano istantaneamente nel momento in cui il tester guadagna punti, sblocca un obiettivo o avanza di livello.
L'implementazione di queste animazioni svolge una funzione cruciale nel ciclo di gratificazione dell'utente. Fornendo risposte immediate, chiare e personalizzate alle azioni del tester, il sistema crea un feedback loop positivo (un circolo virtuoso) che rafforza il comportamento desiderato. Quando l'utente sperimenta una nuova strategia esplorativa e viene immediatamente ricompensato con un'animazione a schermo, si sente incoraggiato a reiterare e variare ulteriormente le proprie interazioni. Questa dinamica di azione-risposta innesca il *Core Drive 3: Empowerment of Creativity & Feedback*. Il tester è stimolato a utilizzare l'intuizione e la creatività per scovare i percorsi nascosti dell'applicazione, sapendo che il sistema è reattivo e pronto a validare visivamente le sue scoperte.

4.3 Profilo utente

Nel contesto di un'applicazione volta a stimolare la motivazione intrinseca e la partecipazione attiva, la rappresentazione dell'identità dell'utente riveste un ruolo primario. Per tale motivo, l'estensione è stata arricchita con una sezione dedicata interamente alla gestione del *profilo utente*. Rispetto al prototipo iniziale, il nuovo modulo offre un grado di personalizzazione superiore, permettendo ai tester non solo di definire liberamente il proprio username, ma anche di selezionare un'icona (avatar) che lo rappresenti all'interno dell'estensione, della classifica e dei report di sessione.

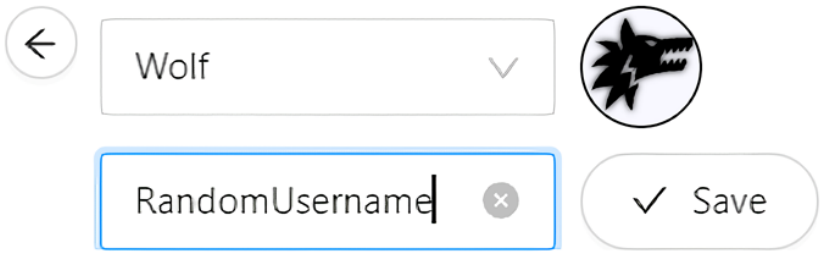


Figura 4.10: Sezione di gestione del profilo utente

Dal punto di vista del game design e dell'Octalysis Framework, questa funzionalità è uno strumento per innescare esplicitamente il *Core Drive 4: Ownership & Possession*. Nello specifico, essa implementa una nota dinamica di gamification definita *Costruire da zero*.
Questo catalizzatore motivazionale si basa sul principio psicologico per cui gli individui, nel momento in cui sentono di possedere qualcosa o di averne il pieno controllo, sono naturalmente spinti a migliorarla, proteggerla e dedicarvi maggiore attenzione. Nel contesto severo e rigoroso del software testing, fornire all'utente la sovranità sulla propria identità digitale trasforma Appium Inspector in uno spazio di lavoro personale e personalizzato.

La letteratura scientifica, supportata da concetti consolidati dell'economia comportamentale (come ad esempio il noto *IKEA effect*), dimostra che quando gli utenti investono tempo, energia o creatività per plasmare un elemento fin dal principio, sviluppano un forte senso di appartenenza e di attaccamento verso di esso. Se a un tester venisse assegnato un profilo semplificato e immutabile, il suo coinvolgimento ne risulterebbe fortemente compromesso, percependo il sistema come freddo e distante. Al contrario, chiedendo al tester di *costruire da zero* la propria identità virtuale prima di iniziare l'attività esplorativa, si consolida un

tacito impegno psicologico. Questo investimento emotivo è fondamentale, poiché funge da ancoraggio cognitivo che fidelizza il tester all'applicazione, rendendolo meno incline ad abbandonare la sessione e più propenso a investire ulteriore sforzo per veder crescere il profilo che egli stesso ha creato.

4.4 Badges e funzionalità sbloccabili

I trofei virtuali (*badges*) rappresentano una delle meccaniche più iconiche ed efficaci nel panorama della gamification. Nel prototipo iniziale del progetto, il sistema di conseguimento dei badge risultava limitato a poche interazioni basilari. Per elevare la qualità dell'esperienza e fornire stimoli diversificati, il comparto degli obiettivi è stato rielaborato e ampliato, introducendo maggiori traguardi che riflettono in modo più fedele la complessità del testing esplorativo.
Attualmente, il sistema offre badge in base a diversi criteri, tra cui: il raggiungimento di determinati livelli utente, il numero di "pagine" scoperte, la percentuale di *widgets* coverage ottenuta sulla GUI, la posizione raggiunta nella classifica competitiva, fino al conseguimento di obiettivi speciali legati ad azioni particolari sull'interfaccia (ad esempio, un badge si può ottenere riuscendo a raddoppiare il proprio punteggio durante il periodo di attivazione di un *booster*).

Un elemento cardine della nuova interfaccia è la visualizzazione esplicita del numero totale di badge disponibili nel sistema, inclusi quelli non ancora ottenuti. Questa scelta progettuale non è casuale, ma è mirata a stimolare intensamente il *Core Drive 4: Ownership & Possession*, sfruttando la tecnica del *Set da collezione*.
La psicologia umana è intrinsecamente incline alla categorizzazione. Mostrare all'utente una collezione parzialmente vuota genera un sottile ma potente desiderio di raccogliere tutti gli elementi mancanti. Il tester è quindi spinto a esplorare l'applicazione non solo per dovere metodologico, ma per appagare il bisogno psicologico di completare la propria collezione di trofei, aumentando di conseguenza il tempo e l'attenzione dedicati al collaudo delle applicazioni.

Un'innovazione significativa riguarda la gestione delle informazioni associate ai badge non ancora ottenuti. Il sistema prevede la possibilità di rivelare il loro grado di rarità e una descrizione che indica come ottenerli. Tuttavia, per prevenire un approccio eccessivamente guidato e per valorizzare la scoperta, questa funzionalità non è accessibile ai nuovi utenti: è necessario raggiungere almeno il livello 10 per poterla utilizzare (figura 4.11).
Questa restrizione deliberata è un'applicazione pratica della tecnica dell'*Interfaccia utente evoluta*, strettamente legata al *Core Drive 6: Scarcity & Impatience*. L'occultamento iniziale di certe funzionalità o complessità impedisce il sovraccarico

cognitivo del tester novizio. Al contempo, il fatto che una funzione sia inaccessibile la rende intrinsecamente più desiderabile. Trattenere funzionalità e sbloccarle gradualmente man mano che l'utente acquisisce esperienza fa sì che quest'ultimo percepisca le nuove opzioni come un bene prezioso e una conquista sudata.
Infine, il momento esatto in cui il tester raggiunge il traguardo del decimo livello innesca la dinamica dello *Sblocco di funzionalità*. Considerata una delle meccaniche di gamification di maggior successo, lo sblocco di un'opportunità prima inaccessibile rivitalizza l'esperienza utente.
Questa tecnica si collega direttamente al *Core Drive 3: Empowerment of Creativity & Feedback*. Fornire al tester nuovi indizi su badge meno comuni proprio quando possiede ormai una solida padronanza dello strumento, gli permette di utilizzare le nuove informazioni per pianificare strategie creative e mirate.

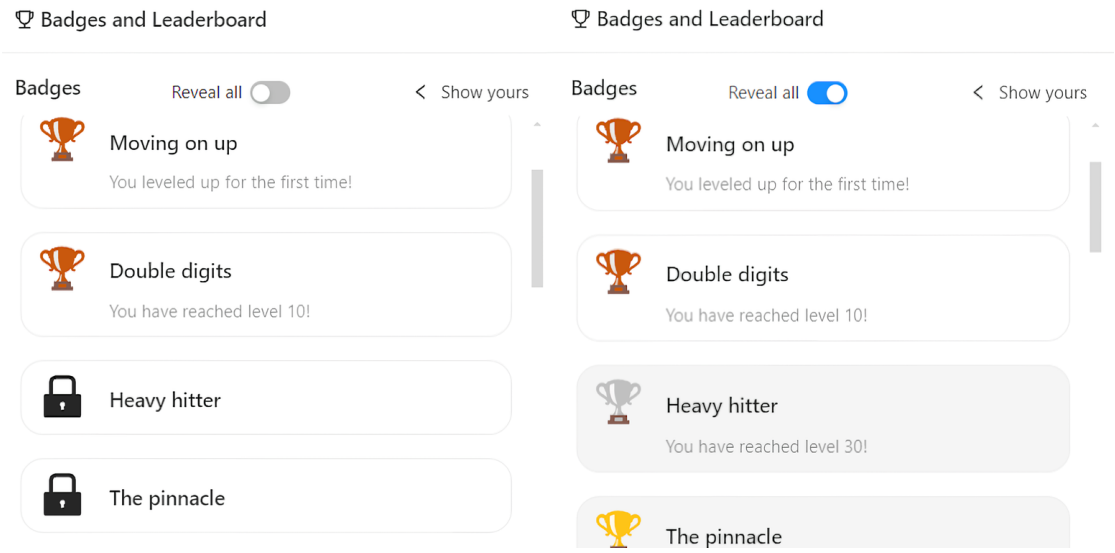


Figura 4.11: Funzionalità di rivelazione dei badge non ancora ottenuti disattivata/attivata

4.5 Classifica

L'introduzione di punteggi, livelli e trofei perde parte della sua efficacia motivazionale se gli utenti non hanno la possibilità di contestualizzare i propri traguardi all'interno di una comunità. Per stimolare una sana competitività tra i tester, l'infrastruttura della classifica del prototipo iniziale è stata ridisegnata e arricchita di nuove funzionalità (in basso a destra nella figura 4.1).
Mentre la versione originale offriva unicamente un abbozzo statico con punteggi fittizi, la nuova implementazione presenta una tabella interattiva e multidimensionale. Sebbene per le attuali finalità di prototipazione e collaudo dell'estensione i profili visualizzati siano simulati (dati hard-coded), l'architettura è stata predisposta per accogliere dati reali da un backend condiviso.

L'interfaccia della classifica non si limita più a mostrare un singolo valore numerico, ma espone un quadro completo dei profili utente, includendo la posizione assoluta in graduatoria (*ranking*), il livello raggiunto, il punteggio totale accumulato, e il numero di badge ottenuti.
Un'ulteriore aggiunta fondamentale è la possibilità di ordinare dinamicamente la classifica in base a ciascuno di questi parametri. Questa flessibilità garantisce all'utente un maggiore senso di controllo e gli permette di individuare metriche alternative in cui eccelle (ad esempio, potrebbe non essere il primo in termini di punteggio, ma potrebbe risultare il tester con il maggior numero di badge rari scoperti).

Da un punto di vista psicologico, la presenza di una classifica non attiva soltanto il *Core Drive 2: Development & Accomplishment*, ma anche il *Core Drive 5: Social Influence & Relatedness*, seppure leggermente. Nel contesto del software testing, l'introduzione di un elemento di confronto sociale spinge il tester a incrementare le proprie performance per raggiungere o superare i colleghi, innescando dinamiche di competizione virtuosa.
Le classifiche tradizionali nascondono tuttavia un'insidia psicologica ben nota nella gamification: l'effetto del calo motivazionale. Quando un nuovo utente accede al sistema e visualizza gli utenti in vetta alla classifica con punteggi attualmente irraggiungibili, rischia di provare un senso di impotenza e abbandonare immediatamente la competizione, percependo l'obiettivo finale come matematicamente precluso.
Per neutralizzare questo rischio, quando l'utente accede al sistema, la vista della classifica viene riposizionata e centrata su di lui. Inoltre, è stato implementato uno specifico pulsante dedicato a riattivare questa funzionalità (*Your rank* button).
Questa semplice meccanica di navigazione è l'applicazione diretta della tecnica di gamification definita *Ottimismo Urgente*. Centrando la vista sul proprio profilo, il sistema pone in risalto i diretti avversari posizionati immediatamente sopra

e sotto il tester. Questo ridimensionamento del contesto trasforma una sfida impossibile in un obiettivo tangibile e a portata di mano: l'utente realizza che gli bastano pochi punti o la scoperta di una singola "pagina" aggiuntiva per superare il collega che lo precede. L'ottimismo urgente genera quindi una spinta motivazionale istantanea, incoraggiando il tester a compiere quello sforzo immediato che lo separa dal traguardo successivo.

4.6 Gestione dell'impazienza e *FOMO*

Durante le sessioni di testing esplorativo, è possibile che il tester decida di abbandonare una schermata quando manca poco al completamento della coverage della stessa. Per mitigare questo fenomeno e sostenere l'ingaggio dell'utente fino al completamento assoluto delle attività, il sistema è stato dotato di un meccanismo di messaggistica proattiva.
L'estensione non si limita a calcolare passivamente il progresso nella barra di avanzamento, ma interviene attivamente generando avvisi contestuali (alerts/notifiche visive) basati sullo stato di avanzamento del tester. Questi messaggi sono progettati per fornire micro-obiettivi a breve termine. Ad esempio, l'algoritmo monitora costantemente il rapporto tra i *widgets* totali e quelli interagiti nella "pagina" corrente. Se rileva che tale differenza è minima (come ad esempio, con 9 *widgets* scoperti su 10 totali), il sistema mostrerà un incoraggiamento del tipo: "Continua a testare, manca un solo elemento con cui interagire!" (figure 4.12 e 4.14).
Similmente, vengono inviate notifiche strategiche quando il punteggio accumulato è a ridosso della soglia necessaria per sbloccare un trofeo o per raggiungere un livello importante (figura 4.13).

```
const getCurrentPageCoverage = () :number  => {  Show usages  Alessandro Costantino
  if (currentPageId === null)
    return 0;
  const currentPage = pages.find(p => p.pageId === currentPageId);
  if (!currentPage)
    return 0;
  const coverage :number  = Math.round( x: 100 * currentPage.nInteractedWidgets / currentPage.nInteractableWidgets);
  // FOMO coverage
  if ((currentPage.nInteractedWidgets === currentPage.nInteractableWidgets - 1) && !fomoCalled.current) {
    message.info( content: {
      content: "KEEP TESTING...! In this page you are missing only one widget to interact with",
      key: "fomo",
      duration: 5,
      style: {marginTop: '15vh'},
    }).then( onfulfilled: fomoCalled.current = true);
  }
```

Figura 4.12: Parte della funzione di gestione della coverage delle pagine

```
useEffect( effect: () : function(): void | undefined => {
  if (calculatedLevel === 9 && calculatedProgress < 50) {
    prevProgressRef.current = calculatedProgress;
  } else if (calculatedLevel === 9 && prevProgressRef.current < 50 && calculatedProgress >= 50) {
    prevProgressRef.current = calculatedProgress;
    message.info( content: {
      content: "KEEP TESTING! You are almost at level 10",
      key: "level10",
      duration: 5,
      style: {marginTop: '15vh'},
    });
  } else if (calculatedLevel === 29 && calculatedProgress < 66) {
    prevProgressRef.current = calculatedProgress;
  } else if (calculatedLevel === 29 && prevProgressRef.current < 66 && calculatedProgress >= 66) {
    prevProgressRef.current = calculatedProgress;
    message.info( content: {
      content: "KEEP GOING! You are not far from level 30",
      key: "level30",
      duration: 5,
      style: {marginTop: '15vh'},
    });
  } else if (calculatedLevel === 49 && calculatedProgress < 75) {
    prevProgressRef.current = calculatedProgress;
  } else if (calculatedLevel === 49 && prevProgressRef.current < 75 && calculatedProgress >= 75) {
    prevProgressRef.current = calculatedProgress;
    message.info( content: {
      content: "ONE LAST PUSH! You have almost reached the max level",
      key: "levelmax",
      duration: 5,
      style: {marginTop: '15vh'},
    });
  }
```

Figura 4.13: Parte della funzione di gestione dei livelli

L'integrazione di questi avvisi puntuali innesca parzialmente il *Core Drive 6: Scarcity & Impatience*. Nella gamification, mostrare a un utente che è a un passo dal raggiungimento di un obiettivo genera una potente frizione psicologica e un senso di impazienza costruttiva. La ricompensa appare imminente, inducendo il tester a compiere quello sforzo esplorativo finale per "chiudere il cerchio", piuttosto che rimandare l'azione al futuro.

L'aspetto motivazionale predominante di questi avvisi risiede tuttavia nell'attivazione del *Core Drive 8: Loss & Avoidance*. Questo principio sfrutta la naturale tendenza umana a modificare il proprio comportamento per evitare conseguenze negative o per non sprecare il lavoro già svolto.

Nello specifico, le notifiche implementano una variante della celebre tecnica *Fear Of Missing Out* (FOMO). Avvisare il tester in procinto di abbandonare una schermata di ciò che sta per lasciarsi alle spalle capovolge la sua percezione: l'interruzione anticipata non è più vista come un risparmio di tempo, ma come una potenziale

perdita. Il timore di perdere l'opportunità di sbloccare una ricompensa quasi garantita agisce come un forte deterrente contro l'abbandono prematuro. Questa spinta emotiva si traduce in un vantaggio tangibile per la qualità del software, poiché costringe l'utente a perseverare nella ricerca di eventuali bug nascosti in quei componenti dell'interfaccia meno evidenti.

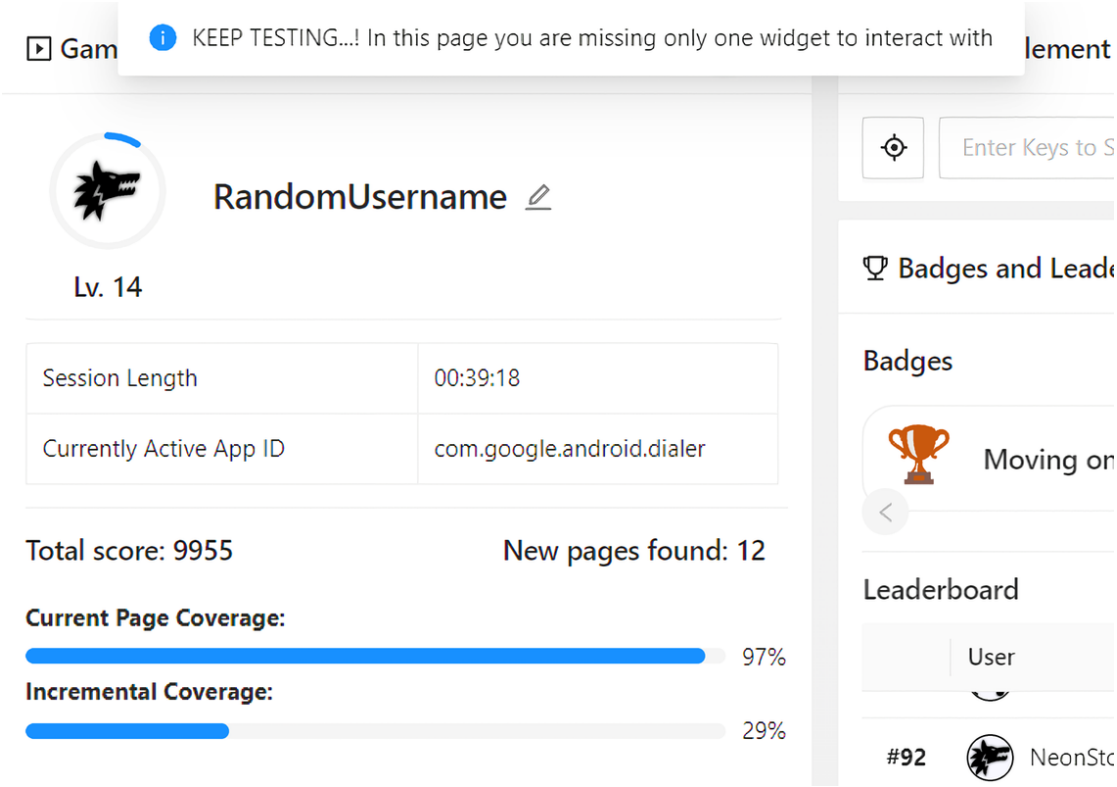


Figura 4.14: Notifica contestuale basata sulla coverage della pagina

4.7 Imprevedibilità delle ricompense

In un sistema di gamification, l'erogazione di ricompense fisse e prevedibili (come ad esempio, un ammontare standard di punti per ogni interazione di una certa tipologia) è fondamentale per stabilire le regole di base, ma nel lungo periodo rischia di generare assuefazione. Quando il cervello umano impara a prevedere l'esatto esito di un'azione, il rilascio di dopamina diminuisce e l'attività si trasforma rapidamente in una routine meccanica. Per contrastare questo fenomeno e mantenere elevato l'ingaggio dei tester, il sistema è stato potenziato con l'introduzione di ricompense bonus, caratterizzate da un valore variabile e da un'erogazione apparentemente casuale.

La nuova logica implementata nel prototipo prevede l'assegnazione di punti extra al verificarsi di specifiche condizioni di successo. Ad esempio, quando un tester raggiunge il 100% della coverage in una "pagina", il sistema può erogare un bonus di punti aggiuntivo il cui valore è generato casualmente entro un certo intervallo (da 500 a 3000 punti), oppure, al passaggio di alcuni livelli, il tester potrebbe ricevere un bonus proporzionale al traguardo raggiunto, creando un incentivo scalabile (figure 4.15, 4.16 e 4.17).

L'inserimento di questa variabilità è progettata per innescare profondamente il *Core Drive 7: Unpredictability & Curiosity*. La letteratura sulla gamification dimostra che le ricompense di entità variabile rappresentano uno degli stimoli più potenti per la reiterazione di un comportamento. Nello specifico, l'estensione adotta diverse declinazioni di questa meccanica:

- *Ricompense variabili/casuali*: il tester non sa mai esattamente quanti punti extra otterrà, il che mantiene viva la curiosità.
- *Ricompense a sorpresa*: l'erogazione del bonus si manifesta senza preavvisi invadenti, generando un effetto sorpresa positivo.
- *Ricompense a catena*: raggiungere un traguardo può innescare la possibilità di ottenerne immediatamente un altro. Ad esempio, è possibile che il bonus di punti per aver completato la coverage di una "pagina" permetta al tester di superare la soglia e passare di livello, innescando un secondo bonus.

Sapere che la perseveranza nel testing offre la possibilità, seppur non garantita, di vincere premi cospicui, spinge l'utente a rimanere coinvolto nel ciclo di testing più a lungo rispetto a un sistema con punteggi fissi.
Inoltre, oltre a stimolare la curiosità continua, il sistema delle ricompense bonus è stato calibrato per intervenire in modo strategico soprattutto durante le prime fasi di utilizzo del software (fasi di *scoperta* e *onboarding*).

Questa manipolazione controllata delle probabilità di ricompensa è finalizzata ad attivare il *Core Drive 1: Epic Meaning & Calling*, e in particolare la meccanica nota come *Fortuna del principiante*.
Quando un nuovo tester sperimenta un successo inaspettato e ottiene una ricompensa bonus considerevole dopo aver iniziato a usare lo strumento da poco, sviluppa un'illusione positiva. Inconsciamente, l'utente si sente "scelto", particolarmente fortunato o dotato di un talento innato per il testing esplorativo di quella specifica applicazione. Ciò consolida un forte attaccamento emotivo alla piattaforma, neutralizzando l'attrito iniziale legato all'apprendimento di un nuovo strumento e rendendo il tester più propenso a continuare l'attività di testing.

```
useEffect( effect: () : function(): void | undefined => {
  // Bonus level rewards
  if (prevLevelRef.current === 4 && calculatedLevel === 5) {
    const bonusScore : number = [250, 500, 750, 1000][Math.floor( x: Math.random() * 4)];
    message.warning( content: {
      content: <BonusRoulette level={true} bonusScore={bonusScore} possibleValues={[250, 500, 750, 1000]}/>,
      key: "roulette5",
      duration: 5,
      style: {marginTop: '15vh'},
    }).then( onfulfilled: () => addScore(bonusScore));
  } else if (prevLevelRef.current === 19 && calculatedLevel === 20) {
    const bonusScore : number = [5000, 7500, 10000, 12500, 15000][Math.floor( x: Math.random() * 5)];
    message.warning( content: {
      content: <BonusRoulette level={true} bonusScore={bonusScore} possibleValues={[5000, 7500, 10000, 12500, 15000]}/
      key: "roulette20",
      duration: 5,
      style: {marginTop: '15vh'},
    }).then( onfulfilled: () => addScore(bonusScore));
  } else if (prevLevelRef.current === 39 && calculatedLevel === 40) {
    const bonusScore : number = [30000, 50000, 70000][Math.floor( x: Math.random() * 3)];
    message.warning( content: {
      content: <BonusRoulette level={true} bonusScore={bonusScore} possibleValues={[30000, 50000, 70000]}/>,
      key: "roulette40",
      duration: 5,
      style: {marginTop: '15vh'},
    }).then( onfulfilled: () => addScore(bonusScore));
  }
```

Figura 4.15: Funzione di gestione delle ricompense bonus al passaggio di livello

```
if (coverage >= 100 && !coveragePages.current.includes(currentPageId)) {
  coveragePages.current.push(currentPageId);
  if ([0, 1][Math.floor( x: Math.random() * 2)] === 1) {  // 50% chance to trigger bonus score
    const bonusScore : number  = [500, 1000, 1500, 2000, 2500, 3000][Math.floor( x: Math.random() * 6)];
    message.warning( content: {
      content: <BonusRoulette level={false} bonusScore={bonusScore} possibleValues={[500, 1000, 1500, 2000, 2500, 30
      key: "roulette",
      duration: 10,
      style: {marginTop: '15vh'},
    }).then( onfulfilled: () => addScore(bonusScore));
  }
}
```

Figura 4.16: Funzione di gestione delle ricompense bonus al completamento della coverage

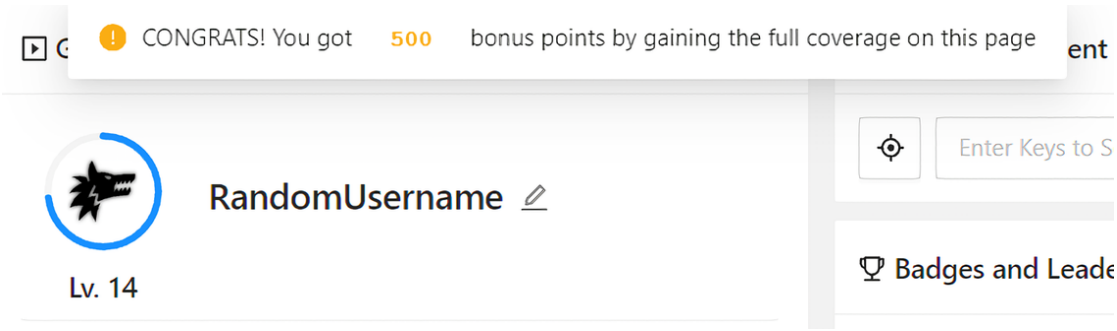


Figura 4.17: Notifica animata di assegnazione del bonus casuale

4.8 Moltiplicatori di punteggio

Nelle fasi più avanzate di una sessione di testing esplorativo, quando l'utente ha già familiarizzato con l'interfaccia, è fisiologico riscontrare un calo dell'attenzione e del ritmo di interazione. Per contrastare questa naturale flessione e per stimolare ulteriormente il tester, l'estensione è stata arricchita con una meccanica dinamica di moltiplicazione del punteggio, definita *Booster*.
Il sistema implementa un algoritmo che, in modo pseudocasuale e dopo aver superato determinati intervalli di tempo, attiva un booster temporaneo (figura 4.18). Durante questa finestra di attivazione, tutti i punti guadagnati dal tester subiscono un incremento esponenziale (ad esempio, possono essere raddoppiati o addirittura triplicati).
Cruciale per l'efficacia di questa meccanica è la sua natura rigidamente circoscritta nel tempo: il booster ha una durata variabile, al massimo di due minuti (figura 4.19). Nell'interfaccia utente, l'attivazione viene segnalata in modo evidente attraverso un feedback visivo accompagnato da un timer per il conto alla rovescia, che traccia in tempo reale i secondi rimanenti prima dell'esaurimento dell'effetto (figura 4.22).

```
// Score boosters setup
  const schedule :any[] = useMemo( factory: () :any[] => {
    const timestamps :any[] = [];
    let cursor :number = 60;  // Time, in seconds, from the session start to when the boosters can be activated

    // Still time left in the session
    while (cursor < (SESSION_DURATION * 60)) {
      const randomVariance :number = Math.floor( x: Math.random() * 300); // 0 to 5 min variance
      const nextStart :number = cursor + (BOOSTERS_GAP * 60) + randomVariance;

      if (nextStart > ((SESSION_DURATION * 60) - 120))  // 120 = 2 min = max possible booster duration
        break;

      timestamps.push(nextStart);

      cursor = nextStart + 120; // 120 = 2 min = max possible booster duration
    }

    return timestamps;
  }, deps: []);
```

Figura 4.18: Funzione di schedule per determinare quando si attiverà il booster

```
const triggerBooster = () :void => {  Show usages  Alessandro Costantino
  const boosterDuration :number = Math.floor( x: (Math.random() * (120 - 30)) + 30);  // 30s to 2min
  const boosterValue :number = [1.5, 2, 3][Math.floor( x: Math.random() * 3)];        // 1.5x - 2x - 3x
  setBoosterDuration(boosterDuration);
  setBoosterValue(boosterValue);
  setBoosterEndTime( value: Date.now() + (boosterDuration * 1000));
  setBoosterActive(true);
  // Saves the score at the moment a booster starts
  scoreBoosterStart.current = scoreRef.current;

  // Animations
  setBoosterAnimationClass(InspectorStyles['animate-pop-in']);
  // Wait for booster to finish
  setTimeout( handler: () :void => {
    setBoosterAnimationClass(InspectorStyles['animate-pop-out']);
  }, (boosterDuration * 1000));

  message.warning( content: {
    content: (
      <span>
        <span className={InspectorStyles['lightning-anim']}> ϟ </span>
        {` ${boosterValue}x SCORE BOOSTER ACTIVATED! `}
        <span className={InspectorStyles['lightning-anim']}> ϟ </span>
      </span>
    ),
    key: "booster",
    style: { marginTop: '15vh' }
  });
};
```

Figura 4.19: Funzione di attivazione del booster

```
function updateTime() : void {  Show usages  Alessandro Costantino
  const timeDiff : number  = Date.now() - startTime.current;
  const hours : number  = timeDiff / 3600000;
  const minutes : number  = (hours - Math.floor(hours)) * 60;
  const seconds : number  = (minutes - Math.floor(minutes)) * 60;
  const showTime = (time) : string  => String(Math.floor(time)).padStart(2, '0');  Show usages  Alessandro Costantino
  const timeString : string  = `${showTime(hours)}:${showTime(minutes)}:${showTime(seconds)}`
  setCurrentTime(timeString);

  if (schedule.includes(Math.floor( x: hours * 3600)))
    triggerBooster();
}
```

Figura 4.20: Funzione di gestione del tempo trascorso nella sessione corrente, utilizzata anche per attivare il booster

```
/* score booster ui logic */
useEffect( effect: () : function(): void | undefined  => {
  let interval;
  if (boosterActive) {
    interval = setInterval( handler: () : void  => {
      // UI logic
      const remaining : number  = boosterEndTime - Date.now();
      if (remaining <= 0) {
        setBoosterProgress( value: 0);
        setTimeout( handler: () : void  => {
          setBoosterActive(false);
        }, timeout: 300);
        // Score booster badge
        if (
          (scoreBoosterStart.current !== 0) &&
          (scoreRef.current >= (scoreBoosterStart.current * 2)) &&
          (badges.every(b => b.id !== "points-booster"))
        ) {
          addBadge(GAMIFICATION_BADGES.find((b) : boolean  => b.id === 'points-booster'), 'silver');
        }
      } else {
        const percent : number  = (remaining / (boosterDuration * 1000)) * 100;
        setBoosterProgress(percent);
      }
    }, timeout: 100);
    return () : void  => clearInterval(interval);
  }
}, deps: [boosterActive, boosterEndTime]);
```

Figura 4.21: Funzione di gestione del booster nell'interfaccia utente

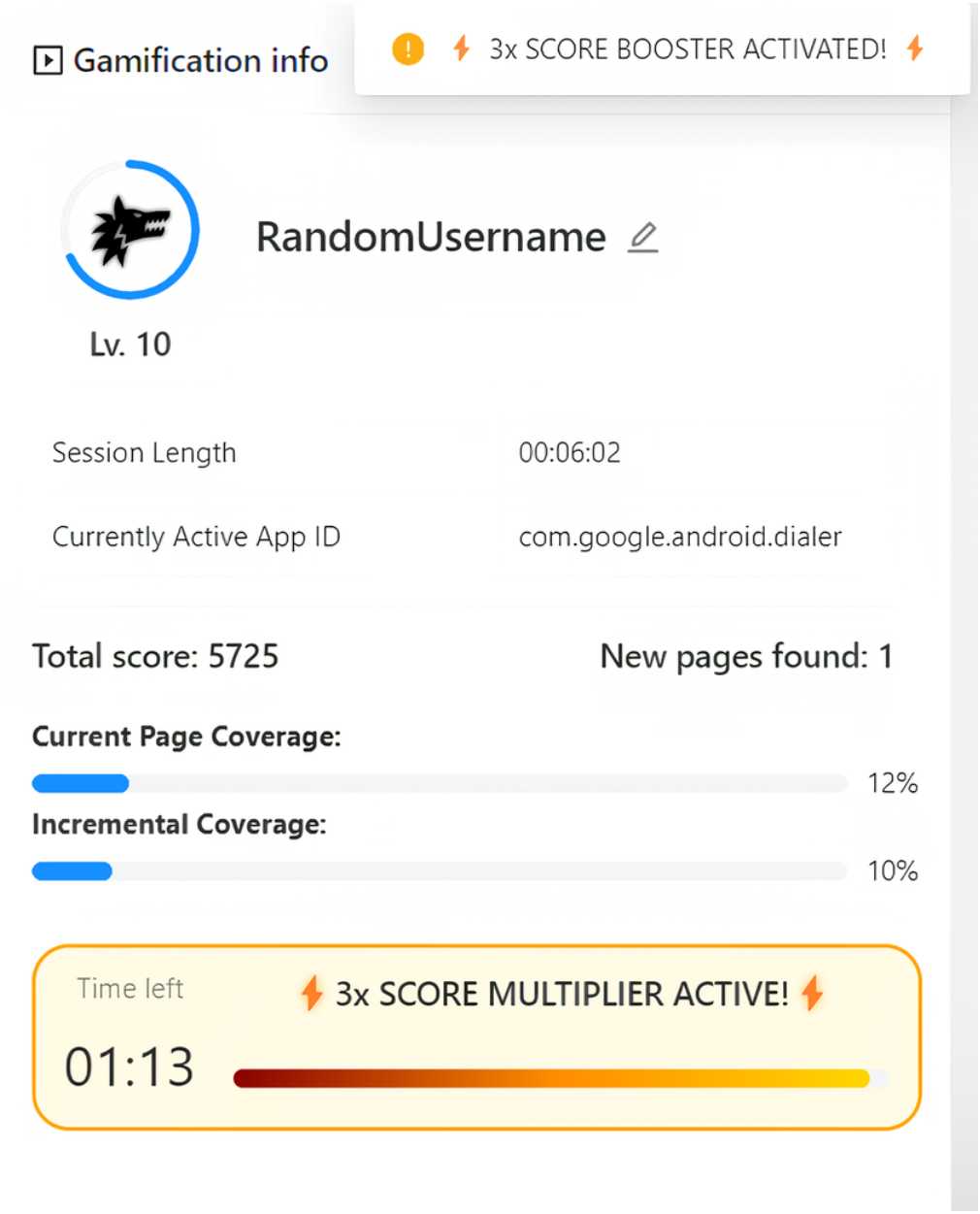


Figura 4.22: Booster attivato

L'introduzione di questa meccanica mira in primo luogo ad attivare il *Core Drive 3: Empowerment of Creativity & Feedback*, sfruttando la specifica tecnica denominata *Boosters/Potenziamenti*.
Fornire agli utenti la sensazione di essere improvvisamente "potenziati" per un periodo limitato rappresenta un motivatore comportamentale estremamente forte. Il tester percepisce che le sue azioni standard hanno ora un peso e un impatto nettamente superiori sulla scalata della classifica o l'avanzamento di livello. Questa percezione di efficacia amplificata stimola un approccio tattico: l'utente potrebbe, ad esempio, intuire l'imminenza di un booster e iniziare a mappare mentalmente le aree inesplorate, per poi testare rapidamente non appena il moltiplicatore si attiva. In questo caso, il collaudo cessa di essere una procedura lineare per trasformarsi in una strategia di massimizzazione del rendimento.

Infine, la presenza inesorabile del conto alla rovescia svolge un ruolo psicologico forse ancora più determinante, ancorandosi al *Core Drive 8: Loss & Avoidance*. In questo contesto, il sistema applica la tecnica delle *Opportunità effimere*.
Un moltiplicatore permanente o privo di scadenza visibile verrebbe rapidamente dato per scontato dall'utente. Al contrario, il timer scandisce visivamente un'opportunità eccezionale che presto svanirà. L'ansia di sprecare questo enorme vantaggio infonde un profondo senso di urgenza.
Di fronte all'inesorabile scorrere del tempo, il tester è indotto ad accantonare qualsiasi esitazione, aumentando la concentrazione e la velocità con cui interagisce con il sistema, per capitalizzare al massimo il bonus prima che scada. Questa urgenza artificiale può neutralizzare la noia e la procrastinazione e, di conseguenza, può migliorare l'efficienza della copertura dei test.

4.9 Conclusione e copertura Octalysis

L'architettura, le implementazioni algoritmiche e le nuove meccaniche introdotte in questa evoluzione dell'estensione permettono di superare nettamente i limiti del prototipo iniziale di GAppium, trasformando l'attività di testing esplorativo in un'esperienza stimolante e coinvolgente.
Attraverso l'integrazione sinergica di dinamiche ludiche avanzate, quali il sistema di progressione a livelli, la personalizzazione dell'identità digitale, il confronto sociale, le sollecitazioni proattive e l'imprevedibilità delle ricompense, il sistema riesce a stimolare una complessa rete di leve psicologiche nel tester.
A differenza della prima versione sperimentale, che offriva una copertura marginale limitata ai soli *Core Drives 2 e 4* (figura 3.14), il nuovo ecosistema abbraccia ora pressoché l'intero spettro del modello teorico di riferimento. L'estensione orchestra un equilibrio raffinato tra motivatori estrinseci ed intrinseci, combinando

l'accumulo di punti e trofei con il piacere della scoperta, la creatività esplorativa e il senso di unicità del proprio percorso, e tra White Hat e Black Hat gamification, alternando dinamiche che infondono significato e controllo, a dinamiche che generano concentrazione e urgenza.

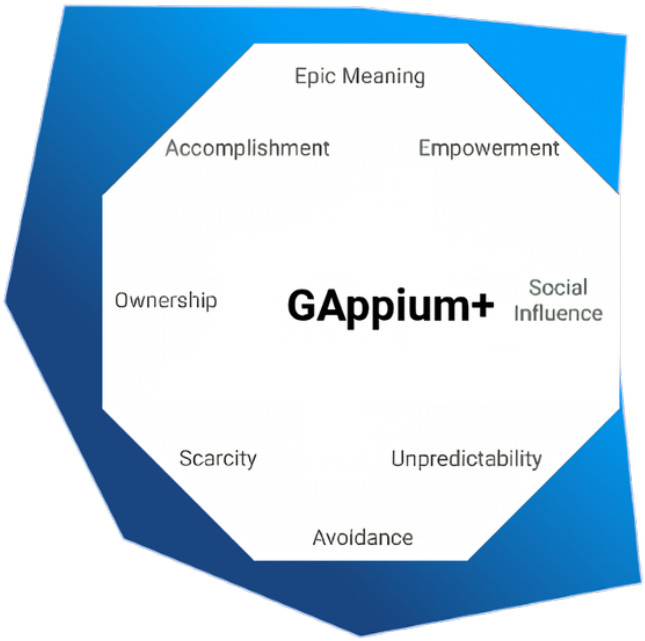


Figura 4.23: Valutazione di GAppium+ basata sull'Octalysis Framework

Come si evince dal nuovo diagramma (figura 4.23), l'estensione (rinominata **GAppium+**) garantisce ora un'espansione volumetrica significativa e una distribuzione più bilanciata sui diversi vertici dell'ottagono. In particolare, il *Core Drive 2: Development & Accomplishment* e il *Core Drive 7: Unpredictability & Curiosity* sono quelli attivati in maggior misura, seguiti dal *Core Drive 3: Empowerment of Creativity & Feedback* e dal *Core Drive 4: Ownership & Possession.*
Questa rinnovata armonia tra gli otto Core Drives consolida il successo progettuale dell'approccio proposto, ponendo le basi teoriche e pratiche ottimali per sviluppi e implementazioni future.

Capitolo 5

Risultati e Valutazione

Al fine di validare l'efficacia delle implementazioni descritte, è stata condotta una valutazione preliminare interattiva di GAppium+. L'esperimento ha coinvolto un campione volontario di dieci partecipanti, composto da dottorandi e assegnisti di ricerca operanti presso i laboratori del Politecnico di Torino. La selezione del campione è avvenuta su base di disponibilità durante la fase sperimentale, garantendo al contempo una certa eterogeneità nei background accademici di provenienza.

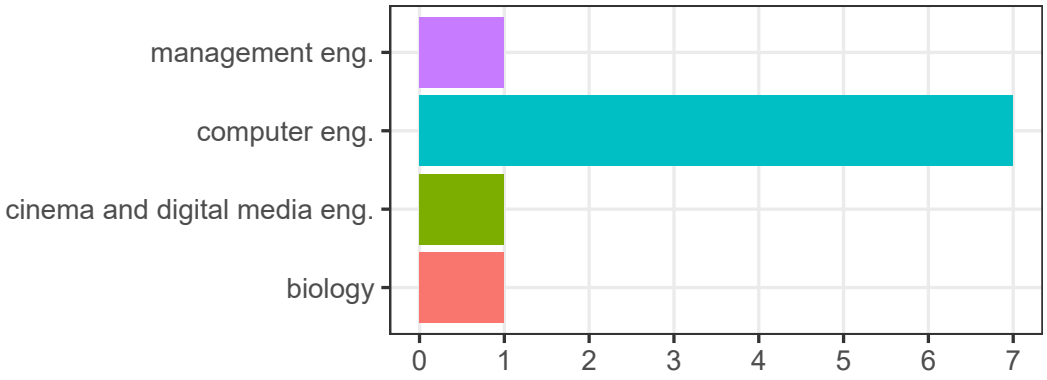


Figura 5.1: Background accademici dei partecipanti

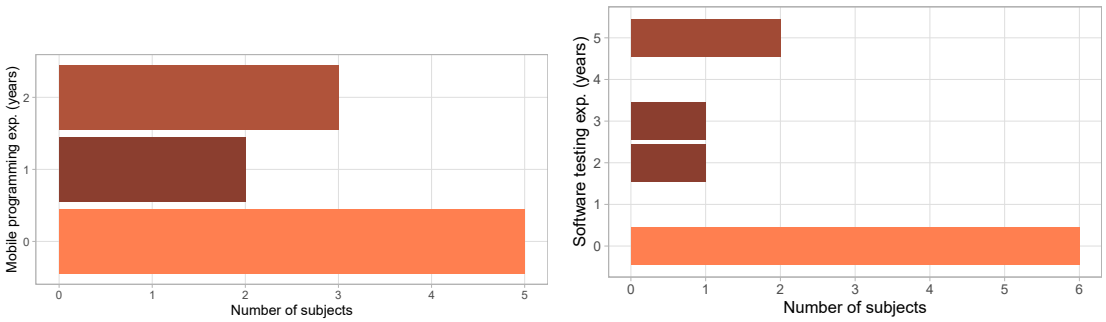


Figura 5.2: Esperienza dei partecipanti nello sviluppo mobile (a sinistra) e nel software testing (a destra)

La metodologia ha previsto una valutazione sperimentale in vivo in condizioni controllate, finalizzata a simulare uno scenario di utilizzo reale. Ai partecipanti è stato assegnato il compito di collaudare usando *OmniNotes* [1] (un'applicazione open-source per la gestione di appunti ampiamente diffusa), agendo a tutti gli effetti come tester esplorativi, senza l'imposizione di script o vincoli di navigazione specifici.
I soggetti hanno anche ricevuto una breve spiegazione preliminare sull'utilizzo tecnico di GAppium+; tuttavia, per evitare bias cognitivi, non sono stati informati preventivamente della presenza di meccanismi di gamification, permettendo loro di scoprirli e interagirvi in modo spontaneo durante l'esplorazione.

5.1 Struttura del questionario

Al termine della sessione di testing, della durata di 15 minuti, i partecipanti sono stati invitati a compilare un questionario [2] per raccogliere un feedback quantitativo e qualitativo immediato. Il questionario è stato strutturato utilizzando una scala *Likert* a 5 punti (1 = *Completamente in disaccordo*, 2 = *In disaccordo*, 3 = *Neutro*, 4 = *D'accordo*, 5 = *Completamente d'accordo*) per facilitare la misurazione statistica e il confronto delle risposte soggettive.

La tabella presente nella pagina seguente elenca le domande del questionario.

[1] `https://github.com/federicoiosue/Omni-Notes`

[2] La lista completa di domande e risposte può essere trovata anche all'indirizzo: `https://doi.org/10.5281/zenodo.19825435`

Tabella 5.1: Lista delle domande del questionario

ID	Testo
Domande relative alla System Usability Scale	
SUS1	In futuro potrei utilizzare questo tipo di applicazione per le mie attività di testing.
SUS2	L'applicazione è risultata più complessa del necessario.
SUS3	L'applicazione e' risultata facile da utilizzare.
SUS4	Penso che avrei bisogno del supporto tecnico di una persona esterna per utilizzare l'applicazione.
SUS5	Le varie funzionalità dell'applicazione erano ben integrate.
SUS6	Penso che ci fosse troppa inconsistenza nell'applicazione.
SUS7	Penso che la maggior parte delle persone potrebbe imparare ad utilizzare l'applicazione rapidamente.
SUS8	Ho avuto bisogno di acquisire tante informazioni prima di poter utilizzare l'applicazione.
Domande specifiche relative all'esperienza di utilizzo	
S1	Ho trovato le spiegazioni degli istruttori abbastanza chiare.
S2	Era chiaro per me capire quali percorsi esplorare per testare l'applicazione.
Domande relative alla gamification	
G1	Gli elementi di gamification erano chiari e comprensibili.
G2	Se dovessi effettuare exploratory testing in futuro, sarei a favore dell'utilizzo di tecniche di gamification.
G3	Credo che sia una buona idea utilizzare la gamification per l'exploratory testing.
G4	Penso che gli aspetti di gamification potrebbero migliorare la mia produttività.
Domande relative alla Gameful Experience	
GMX1	Mi sono divertit* nell'utilizzare l'applicazione.
GMX2	Mentre utilizzavo l'applicazione, ho perso la cognizione del tempo.
GMX3	Ho trovato l'utilizzo dell'applicazione frustrante.
GMX4	Mi sono sentit* autonom* nell'utilizzo dell'applicazione.
GMX5	Mi sono sentit* sicur* di me nell'utilizzo del sistema.
Meccaniche specifiche di gamification *(x :: B = badges, L = leaderboard, C = personalizzazione, P = progresso, V = livelli)*	
Gx1	Penso che la feature x sia utile.
Gx2	Penso che la feature x abbia influenzato il mio utilizzo dell'applicazione.
Gx3	Ho apprezzato la feature x.
Gx4	La feature x mi ha motivato nell'exploratory testing.

Come è possibile osservare anche nella tabella, l'indagine è stata organizzata in diverse aree tematiche chiave:

- **Profilo demografico e Background (D1-D4)**: finalizzata a inquadrare la formazione accademica dei partecipanti e a quantificare i loro anni di esperienza nello sviluppo mobile, nel software testing generale e nell'impiego pregresso di strumenti per il testing esplorativo.
- **Usabilità del sistema (SUS1-SUS8)**: un adattamento della metrica standard *System Usability Scale* (SUS) volto a misurare la facilità d'uso, l'intuitività dell'interfaccia, la percezione di complessità e la curva di apprendimento del prototipo.
- **Chiarezza e Comprensibilità (S1-S2)**: mirata a valutare la chiarezza delle istruzioni fornite all'inizio del test e la comprensibilità dei percorsi esplorativi.

- **Atteggiamenti verso la gamification (G1-G4)**: volta a indagare la chiarezza degli elementi ludici, l'impatto percepito sulla produttività e l'intenzione futura di adottare strumenti "gamificati" nelle proprie procedure di collaudo.
- **Esperienza utente e Coinvolgimento (GMX1-GMX5)**: strutturata secondo la scala di valutazione *GAMEX* (Gameful Experience) [47], per misurare aspetti psicologici ed emotivi dell'esperienza, quali il divertimento, il flow (ossia la perdita della cognizione del tempo), l'autonomia, la sicurezza nell'uso e i livelli di frustrazione.
- **Valutazione delle meccaniche specifiche**: una sezione analitica in cui quasi ogni singola funzionalità ludica (nello specifico Badges/Trofei, Classifica, Barra di Avanzamento, Personalizzazione di Avatar/Username e Sistema a Livelli) è stata valutata trasversalmente secondo quattro criteri: l'utilità percepita, l'influenza sul testing, l'apprezzamento personale e l'impatto motivazionale.
- **Feedback qualitativo (SUGG)**: uno spazio aperto conclusivo dedicato alla raccolta di commenti e suggerimenti non strutturati.

5.2 Analisi dei risultati

I risultati emersi dalla valutazione quantitativa, le cui distribuzioni sono illustrate in dettaglio nelle seguenti figure, hanno fornito indicazioni preziose sull'efficacia dell'approccio e sul gradimento delle singole meccaniche.

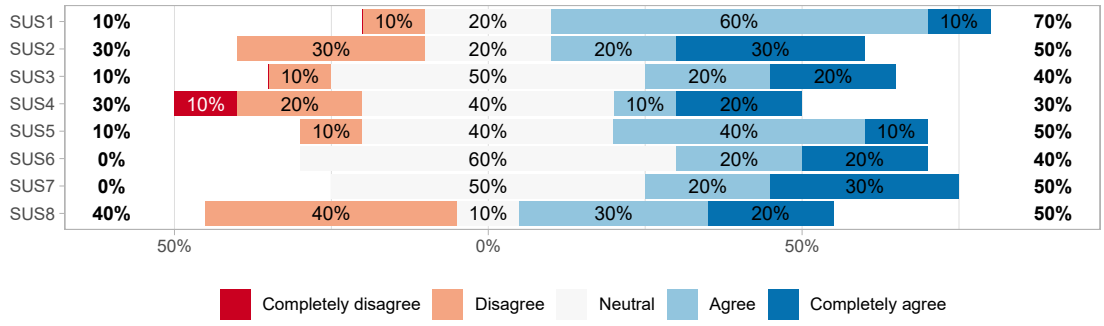


Figura 5.3: Distribuzione delle risposte Likert sulle domande SUS

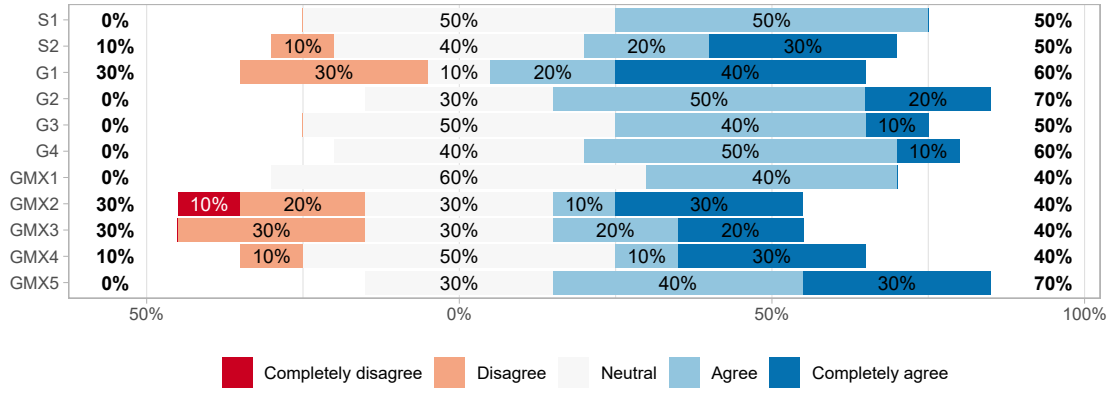


Figura 5.4: Distribuzione delle risposte Likert sulle domande di gamification

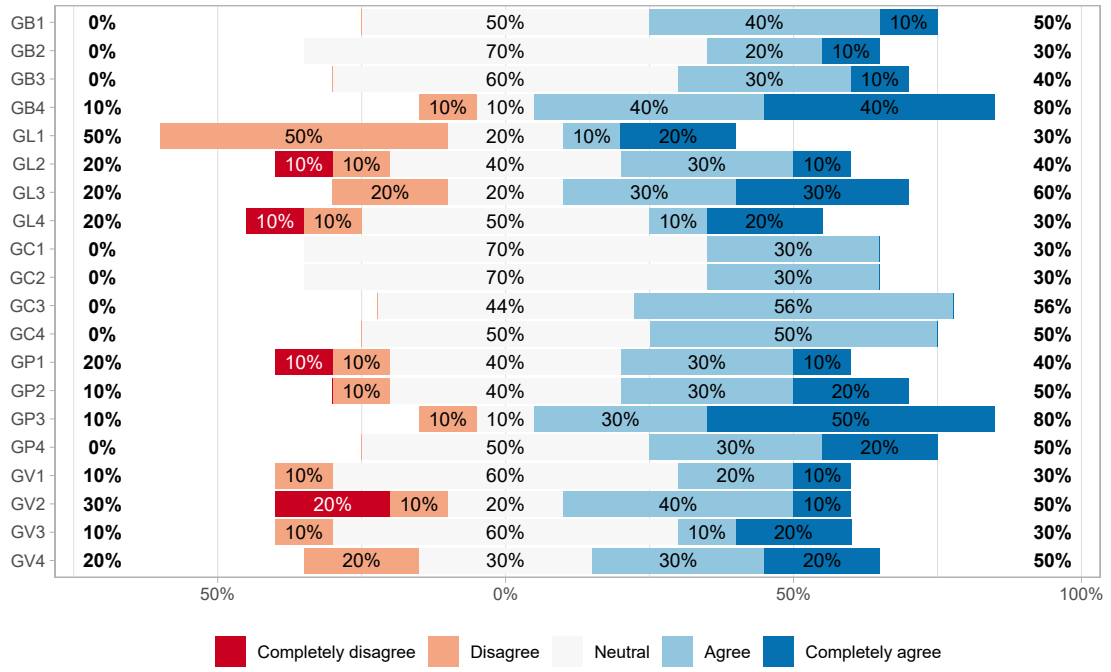


Figura 5.5: Distribuzione delle risposte Likert sulle domande relative alle meccaniche di gamification

In linea generale, l'atteggiamento dei partecipanti nei confronti della gamification (G1-G4) è risultato molto positivo, registrando una media di 3,95 su 5. I punteggi più elevati in questa sezione hanno riguardato l'adeguatezza percepita dell'impiego della gamification nel testing (G3: 4,30) e la propensione per un'adozione futura (G2: 4,30).

Analizzando l'impatto delle singole funzionalità ludiche, la Barra di Avanzamento (Coverage) è emersa come la caratteristica più forte e apprezzata in assoluto (con una media di 4,55 su 5). I partecipanti l'hanno ritenuta estremamente utile (GC1: 4,70) e capace di influenzare in modo diretto ed efficace il loro comportamento di esplorazione (GC2: 4,70). Anche il sistema dei Badges ha registrato un forte consenso, mantenendo un punteggio solido e coerente in tutti gli aspetti indagati (media 4,12 su 5).
Di contro, la Classifica si è rivelata una meccanica meno impattante nel breve lasso di tempo del test (media 3,23 su 5). Il dato più critico è tuttavia emerso per la funzione di Personalizzazione utente (avatar e username), che ha registrato il punteggio più basso dell'intera indagine (media 2,77 su 5). Tale flessione è imputabile alla scarsa influenza percepita sulle azioni pratiche (GP2: 2,10) e al debole impatto motivazionale (GP4: 2,30); questo suggerisce che la personalizzazione puramente estetica rivesta una rilevanza marginale in un ambiente intrinsecamente tecnico e orientato agli obiettivi come il software testing.
Sul fronte puramente tecnico, l'usabilità di GAppium+ è stata valutata come pienamente adeguata (SUS1-SUS8: 3,66 su 5)[3].

Infine, analizzando le metriche legate all'esperienza utente (GAMEX), la voce relativa alla percezione di Frustrazione (GMX3: 2,30) rappresenta l'unico segnale tendenzialmente negativo, confermandosi come il principale aspetto di attrito in un'esperienza d'uso che è risultata per il resto fluida e coinvolgente.

[3]Poiché le domande SUS2, SUS4, SUS6 e SUS8 erano formulate negativamente, i rispettivi punteggi sono stati invertiti (calcolati come *6 - punteggio*) prima di estrapolare la media finale.

Capitolo 6

Conclusioni

Il presente lavoro di tesi si è posto l'obiettivo di progettare, implementare e valutare empiricamente un approccio "gamificato" per il testing esplorativo delle interfacce grafiche utente in ambito mobile. Partendo dalle fondamenta del prototipo originale, è stato sviluppato GAppium+, uno strumento che integra in modo sistematico meccaniche ludiche all'interno di Appium Inspector, basandosi sui solidi principi teorici dell'Octalysis Framework.

L'estensione ha arricchito l'esperienza di testing attraverso l'introduzione di diverse funzionalità, tra cui un sistema di progressione a livelli, una collezione di trofei virtuali/badges, un migliorato tracciamento in tempo reale della coverage, una personalizzazione del profilo utente, o diverse altre tecniche/meccaniche di gamification utili a trasformare un'attività intrinsecamente ripetitiva e soggetta a cali di attenzione in un compito più stimolante, incrementando la motivazione intrinseca e l'ingaggio dei tester.

Inoltre, una valutazione preliminare in vivo ha mostrato un'accoglienza complessivamente positiva verso l'adozione dello strumento. In particolare, la *barra di avanzamento* per il tracciamento della coverage e il sistema dei *badges* si sono rivelate le meccaniche più apprezzate in assoluto, registrando i punteggi più alti in termini di utilità percepita e capacità di influenzare positivamente il comportamento esplorativo. Al contrario, la *classifica* e la personalizzazione del *profilo* si sono rivelate meno efficaci.

Nel caso della classifica, ciò è verosimilmente imputabile alle dimensioni ridotte del gruppo di partecipanti, che non ha permesso l'instaurarsi di una vera e propria dinamica competitiva, mentre per quanto riguarda la personalizzazione dell'identità digitale, i risultati suggeriscono che le modifiche puramente estetiche ricoprono un peso motivazionale marginale all'interno di un contesto altamente tecnico e orientato agli obiettivi come quello del software testing.

6.1 Limiti e sviluppi futuri

Sebbene i risultati ottenuti siano incoraggianti, il lavoro presenta alcuni limiti tecnici e metodologici che delineano chiari obiettivi per i futuri sviluppi dell'estensione.

Dal punto di vista implementativo, l'algoritmo di similarità delle pagine attualmente impiegato si basa sull'*Indice di Jaccard* che, pur garantendo un'ottima efficienza computazionale e tempi di risposta lineari indispensabili per l'utilizzo in tempo reale, presenta delle limitazioni: fondandosi esclusivamente sull'intersezione insiemistica delle proprietà dei nodi, ignora in parte la profondità strutturale e la gerarchia spaziale dell'albero della GUI. Questo può comportare una minore precisione rispetto a metriche più complesse e computazionalmente onerose (come la *Tree Edit Distance*), specialmente nell'identificare cambiamenti semantici sottili o nel distinguere interfacce visivamente simili ma strutturalmente dissimili.

Un ulteriore limite tecnico riguarda l'attuale gestione e classificazione dei widgets interattivi. Al momento, il sistema isola gli elementi con cui l'utente può interagire basandosi prevalentemente sulla verifica dell'attributo *clickable* fornito dall'analisi del server Appium. Una necessaria miglioria futura consiste nell'espandere questa logica di filtraggio, integrando il controllo di un maggior numero di attributi (come, ad esempio, *scrollable*, *checkable* o *focusable*) a seconda delle specifiche necessità di testing. Questa gestione granulare permetterebbe all'algoritmo di adattarsi in modo ancora più fedele alle diverse tipologie di interazione previste dai moderni framework di sviluppo mobile.

Inoltre, una fondamentale area di miglioramento riguarderebbe l'espansione e il bilanciamento delle meccaniche di gamification, con particolare attenzione al quinto Core Drive dell'Octalysis Framework: *Social Influence & Relatedness*. Allo stato attuale, l'implementazione di questo motivatore all'interno di GAppium+ è piuttosto carente ed esclusivamente circoscritta alla classifica competitiva la quale, come emerso dalla valutazione preliminare, ha mostrato un'efficacia molto limitata nel breve termine e in gruppi ristretti. Per di più, il sistema attualmente non offre funzionalità capaci di stimolare l'interazione sociale in senso cooperativo. Per ovviare a questo limite, i futuri sviluppi dello strumento potrebbero prevedere l'introduzione di meccaniche di collaborazione, quali gli *obiettivi di gruppo* (sfide collettive in cui più tester possono raggiungere un determinato obiettivo comune nell'applicazione unendo le forze), o i *brag buttons* (per consentire la condivisione dei traguardi o dei bug più rari scoperti).
L'integrazione di queste dinamiche relazionali non solo arricchirebbe il profilo motivazionale del sistema bilanciandone meglio il lato competitivo, ma favorirebbe la creazione di una vera e propria community, incrementando l'ingaggio, la coesione

e la fidelizzazione dei tester nel lungo periodo.

Infine, per quanto riguarda la valutazione sperimentale, i risultati devono essere interpretati con cautela a causa della dimensione ridotta del campione e della sua forte omogeneità. Inoltre, è anche importante considerare che le sessioni di testing sono state limitate a 15 minuti, ma che, come emerso dai questionari GAMEX, lo strumento non è stato facile da utilizzare.

Alla luce di ciò, i potenziali lavori futuri si possono riassumere nei seguenti punti:

- **Miglioramento dell'interfaccia e Usabilità**: interventi mirati di riprogettazione dell'interfaccia utente per risolvere i problemi di usabilità emersi e ridurre i livelli di frustrazione riscontrati durante l'utilizzo dello strumento.
- **Ampliamento della valutazione**: indagine empirica estesa coinvolgendo una popolazione di partecipanti significativamente più ampia e diversificata.
- **Efficacia della Gamification**: implementare ulteriori meccaniche di gamification (relative principalmente al quinto Core Drive) e indagare in modo quantitativo la relazione diretta tra il livello di ingaggio generato dallo strumento e l'effettiva efficacia del testing. Infatti, l'obiettivo primario consisterebbe nel verificare se la gamification si traduca effettivamente in misurabili miglioramenti oggettivi, quali un aumento percentuale della copertura del codice e un tasso superiore di identificazione dei difetti.

Bibliografia

[1] Wikipedia contributors. *Software bug*. https://en.wikipedia.org/w/index.php?title=Software_bug&oldid=1348956867. 2026 (cit. a p. 1).

[2] Michel Nass, Emil Alegroth e Robert Feldt. «Why many challenges with GUI test automation (will) remain». In: *Information and Software Technology* 138 (mag. 2021), p. 106625. DOI: 10.1016/j.infsof.2021.106625 (cit. alle pp. 2, 8).

[3] Ragini kumari. *Why is Manual Testing Important in the World of Automation?* https://www.testriq.com/blog/post/why-is-manual-testing-important-in-the-world-of-automation. 2026 (cit. alle pp. 2, 12).

[4] Philipp Straubinger e Gordon Fraser. «A Survey on What Developers Think About Testing». In: *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. Ott. 2023, pp. 80–90. DOI: 10.1109/ISSRE59848.2023.00075. URL: https://ieeexplore.ieee.org/abstract/document/10301252 (cit. alle pp. 2, 12).

[5] Sebastian Deterding, Rilla Khaled, Lennart E Nacke, Dan Dixon et al. «Gamification: Toward a definition». In: *CHI 2011 gamification workshop proceedings*. Vol. 12. Vancouver. 2011, pp. 1–79 (cit. alle pp. 2, 13).

[6] Riccardo Coppola, Tommaso Fulcini, Luca Ardito, Marco Torchiano e Emil Alègroth. «On effectiveness and efficiency of gamified exploratory GUI testing». In: *IEEE Transactions on Software Engineering* 50.2 (2023), pp. 322–337 (cit. alle pp. 2, 13, 37).

[7] Lorenzo Laudadio, Riccardo Coppola, Marco Torchiano e Tommaso Fulcini. «GAppium: A Framework to Enact Gamification Mechanics in Appium Inspector». In: *2025 IEEE/ACM Workshop on Gamification in Software Development, Verification, and Validation (Gamify)*. IEEE. 2025, pp. 9–12 (cit. alle pp. 3, 36).

[8] Yu-kai Chou. *Actionable gamification: Beyond points, badges, and leaderboards*. Packt Publishing Ltd, 2019 (cit. alle pp. 4, 14).

[9] Wikipedia contributors. *Software testing*. https://en.wikipedia.org/w/index.php?title=Software_testing&oldid=1345102655. 2026 (cit. alle pp. 5–7).

[10] Diego Sozio. *Una piramide per la qualità*. https://antreem.com/it/blog/una-piramide-per-la-qualita. 2020 (cit. alle pp. 5, 6).

[11] *Verifica e Validazione nel software testing*. https://www.technolush.com/uploads/2019-04-05/banner/verification-validation.png (cit. a p. 7).

[12] Mario Linares Vasquez, Kevin Moran e Denys Poshyvanyk. *Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing*. 2018. arXiv: 1801.06267 [cs.SE]. URL: https://arxiv.org/abs/1801.06267 (cit. alle pp. 9–11).

[13] *Frammentazione dei sistemi operativi Android*. https://www.cellulare-magazine.it/wp-content/uploads/2026/01/google-anrdroid-version i.jpg (cit. a p. 9).

[14] Shauvik Roy Choudhary, Alessandra Gorla e Alessandro Orso. *Automated Test Input Generation for Android: Are We There Yet?* 2015. arXiv: 1503.07217 [cs.SE]. URL: https://arxiv.org/abs/1503.07217 (cit. a p. 11).

[15] Cem Kaner e James Bach. *The Nature of Exploratory Testing*. https://kaner.com/pdfs/NatureOfExploratoryTest.pdf. The original citation for this work was "Exploratory & Risk-Based Testing". 2004 (cit. a p. 12).

[16] Philipp Straubinger e Gordon Fraser. «Gamekins: Gamifying software testing in jenkins». In: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. 2022, pp. 85–89 (cit. a p. 13).

[17] Philipp Straubinger, Tommaso Fulcini, Giacomo Garaccione, Luca Ardito e Gordon Fraser. «Gamifying Testing in IntelliJ: A Replicability Study». In: *Proceedings of the ACM on Software Engineering* 2.ISSTA (2025), pp. 2407–2429 (cit. a p. 13).

[18] Tommaso Fulcini, Riccardo Coppola, Luca Ardito e Marco Torchiano. «A review on tools, mechanics, benefits, and challenges of gamified software testing». In: *ACM Computing Surveys* 55.14s (2023), pp. 1–37 (cit. a p. 13).

[19] Yu-kai Chou. *The Octalysis Gamification Framework*. https://yukaichou.com/gamification-examples/octalysis-gamification-framework/ (cit. alle pp. 14, 22).

[20] Yu-kai Chou. *Core Drive 1: Epic Meaning and Calling*. https://yukaichou.com/gamification-study/8-core-drives-gamification-1-epic-meaning-calling/ (cit. a p. 15).

[21] Yu-kai Chou. *Core Drive 2: Development and Accomplishment*. https://yukaichou.com/gamification-study/8-core-drives-gamification-2-development-accomplishment/ (cit. a p. 16).

[22] Yu-kai Chou. *Core Drive 3: Empowerment of Creativity and Feedback*. https://yukaichou.com/gamification-study/8-core-drives-gamification-3-empowerment-creativity-feedback/ (cit. a p. 17).

[23] Yu-kai Chou. *Core Drive 4: Ownership and Possession*. https://yukaichou.com/gamification-study/8-core-drives-4-ownership/ (cit. a p. 17).

[24] Yu-kai Chou. *Core Drive 5: Social Influence and Relatedness*. https://yukaichou.com/gamification-study/8-core-drives-of-gamification-5-social-influence-relatedness/ (cit. a p. 18).

[25] Yu-kai Chou. *Core Drive 6: Scarcity and Impatience*. https://yukaichou.com/gamification-study/8-core-drives-gamification-6-scarcity-impatience/ (cit. a p. 19).

[26] Yu-kai Chou. *Core Drive 7: Unpredictability and Curiosity*. https://yukaichou.com/gamification-study/the-8-core-drives-of-gamification-7-unpredictability/ (cit. a p. 20).

[27] Yu-kai Chou. *Core Drive 8: Loss and Avoidance*. https://yukaichou.com/gamification-study/8-loss-and-avoidance/ (cit. a p. 21).

[28] Yu-kai Chou. *Left Brain vs Right Brain Core Drives*. https://yukaichou.com/gamification-study/left-brain-extrinsic-brain-intrinsic-core-drives-gamification/ (cit. a p. 21).

[29] Yu-kai Chou. *White Hat vs Black Hat Gamification*. https://yukaichou.com/gamification-study/white-hat-vs-black-hat-gamification/ (cit. a p. 21).

[30] Wikipedia contributors. *JavaScript*. https://en.wikipedia.org/w/index.php?title=JavaScript&oldid=1351077540. 2026 (cit. a p. 24).

[31] Wikipedia contributors. *React (software)*. https://en.wikipedia.org/w/index.php?title=React_(software)&oldid=1350071220. 2026 (cit. a p. 24).

[32] Wikipedia contributors. *Npm*. https://en.wikipedia.org/w/index.php?title=Npm&oldid=1351640960. 2026 (cit. a p. 25).

[33] *Vantaggi del framework React*. https://www.acte.in/wp-content/uploads/2025/06/Major-Advantages-of-React-JS-Acte-e1761639259825.png (cit. a p. 25).

[34] *Appium Documentation*. https://appium.io/docs/en/latest/intro/appium/ (cit. alle pp. 26, 27).

[35] *Architettura di Appium*. https://mobikul.com/wp-content/uploads/2023/09/1-1.png (cit. a p. 27).

[36] *Appium Clients*. https://appium.io/docs/en/latest/intro/clients/ (cit. a p. 27).

[37] *WebDriverIO Documentation*. https://webdriver.io/docs/gettingstarted (cit. a p. 28).

[38] *Appium Drivers*. https://appium.io/docs/en/latest/intro/drivers/ (cit. a p. 28).

[39] *Spec Reporter di WebDriverIO*. https://webdriver.io/assets/images/spec-3477bb7cd9114bf21a5902bc39542c9c.png (cit. a p. 29).

[40] *Allure Reporter di WebDriverIO*. https://webdriver.io/assets/images/allure-bb6c9b036b07594235a5aca5aff5ac43.png (cit. a p. 29).

[41] *Appium Inspector Documentation*. https://appium.github.io/appium-inspector/latest/ (cit. a p. 30).

[42] Wikipedia contributors. *Jaccard Index*. https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=1352343238. 2026 (cit. a p. 42).

[43] Han Hu, Haolan Zhan, Yujin Huang e Di Liu. *Pairwise GUI Dataset Construction Between Android Phones and Tablets*. 2023. arXiv: 2310.04755 [cs.SE]. URL: https://arxiv.org/abs/2310.04755 (cit. a p. 43).

[44] Sam Fletcher e Md Zahidul Islam. «Comparing sets of patterns with the Jaccard index». In: *Australasian Journal of Information Systems* 22 (mar. 2018). DOI: 10.3127/ajis.v22i0.1538. URL: https://ajis.aaisnet.org/index.php/ajis/article/view/1538 (cit. a p. 43).

[45] Kaizhong Zhang e Dennis Shasha. «Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems». In: *SIAM J. Comput.* 18 (dic. 1989), pp. 1245–1262. DOI: 10.1137/0218082 (cit. a p. 43).

[46] Santhi.K Kanishkaa.S. «A comparative analysis of Jaccard and Cosine Similarity for plagiarism detection». In: *IJIREEICE.* 13 (mar. 2025). URL: https://ijireeice.com/wp-content/uploads/2025/03/IJIREEICE.2025.13309.pdf (cit. a p. 44).

[47] René Eppmann, Magdalena Bekk e Kristina Klein. «Gameful Experience in Gamification: Construction and Validation of a Gameful Experience Scale [GAMEX]». In: *Journal of Interactive Marketing* 43 (2018), pp. 98–115. ISSN: 1094-9968. DOI: https://doi.org/10.1016/j.intmar.2018.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1094996818300124 (cit. a p. 68).