POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Gamification per Exploratory Testing Multi-Utente di Applicativi Web

Relatori

Prof. Riccardo COPPOLA

Dott. Tommaso FULCINI

Candidato

Fabiano CAMPION

Aprile 2023

Sommario

Contesto: Il GUI Testing (Graphical User Interface Testing) di applicazioni web, ossia una pratica di testing volta a verificare le funzionalità di ogni componente di una pagina web, è uno step fondamentale del ciclo di sviluppo di un software. I costi elevati e l'assenza di feedback immediati hanno portato a concepire dei nuovi strumenti per la generazione di test case che riescono a replicare le azioni del tester e forniscono un supporto visivo aggiuntivo. L'applicazione della tecnica della gamification, che consiste nell'introduzione di elementi tipici del game design in contesti non ludici, ha permesso di rendere l'attività di testing più coinvolgente ed efficace.
Obiettivo: A partire da un ambiente di testing gamificato, questa tesi propone di integrare ad esso alcuni principi del crowdsourcing, ovvero la pratica volta ad ottenere le informazioni per completare un'attività o un progetto servendosi di una grande platea di persone. Con lo sviluppo dell'aspetto collaborativo della gamification si vuole incrementare la qualità dei risultati in ogni sessione di testing e i tester coinvolti per poter parallelizzare il lavoro.
Metodo: La tesi si concentra sulla teorizzazione di un modello degli stati di un'interfaccia web al fine di implementare le dinamiche di cooperazione. Si propone in seguito un algoritmo per la suddivisione del processo di testing in micro task che possono essere svolti in modo indipendente. Il contesto di applicazione comprende un prototipo di software di GUI Testing (Scout) e un plugin di gamification che verranno opportunamente modificati per poter inserire anche il nuovo plugin per il crowdsourcing.
Risultati: La fase di validazione è stata eseguita predisponendo una sessione di testing guidata e sottoponendola ad un campione di sei persone. Confrontando i risultati con quelli di una sessione equivalente ma con il plugin multi-utente disabilitato sono stati evidenziati miglioramenti in termini di coverage e di quantità di nuovi test case generati.
Conclusioni: Il modello degli stati individuato è molto semplice e necessita di ulteriori sviluppi per essere applicabile in qualsiasi contesto, e la modalità di validazione potrebbe non essere sufficiente per determinare il comportamento del plugin in un contesto reale in cui potrebbero esserci migliaia di persone che collaborano contemporaneamente. Nonostante le semplificazioni imposte e il campione di prova limitato, siamo riusciti ad applicare correttamente i nuovi meccanismi multi-utente proposti e la struttura teorizzata permette di raggiungere gli obiettivi prefissati per l'aumento della qualità dei risultati e la possibilità di suddividere il lavoro ad una maggiore quantità di tester.

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

GUI
: Graphical User Interface

SUT
: System Under Test

MVC
: Model View Controller

BMC
: Business Model Canvas

LGC
: Lean Gamification Canvas

IPO
: Input Process Output

JSON
: JavaScript Object Notation

VCS
: Version Control System

Capitolo 1

Introduzione

Un prodotto possiede un ciclo di vita che si estende dalla sua ideazione, alla distribuzione e infine alla sua messa fuori mercato. Prima di entrare nel mercato ogni prodotto deve passare una serie di test per essere conforme alle leggi e garantire che rispetti i requisiti senza generare effetti non voluti.
Il ciclo di vita di un software è molto simile al ciclo di vita di un prodotto materiale e quindi deve essere sottoposto anch'esso a dei test. Spesso questa fase è sottovalutata dai produttori di software poiché richiede molto tempo e risorse, a volte più della progettazione stessa e quindi è molto dispendiosa. Lo studio di (Maneela Tuteja e Gaurav Dubey, 2012) [1] stima che il costo del software testing sia attorno al 40-50% dei costi di sviluppo e sia quanto più dispendioso quanto sia necessaria l'affidabilità di un sistema.
Il testing dovrebbe essere parte di ogni fase di sviluppo di un software, ad ogni linea di codice dovrebbe corrispondere uno o più test che ne verifichino il corretto funzionamento, e ad ogni cambiamento apportato ne consegue l'aggiornamento del rispettivo test. La fase di testing però non si limita solo alla verifica delle funzionalità, ma vengono analizzati anche molti altri parametri, quali le prestazioni, la fruibilità, la conformità ai requisiti, la sicurezza e molto altro.
Questa tesi si svolge nell'ambito del GUI (Graphical User Interface) Testing, il testing delle interfacce grafiche di un software atto a verificare la funzionalità degli elementi presenti in ogni singola schermata. Questi elementi possono essere cliccabili e quindi comportare una risposta (bottoni, aree di testo, link ecc) oppure non cliccabili per i quali se ne può verificare solo la presenza.
La tipologia di testing considerata è in particolar modo importante nelle applicazioni web o mobile le quali sono incentrate nell'interazione con gli utenti e comprendono una grande quantità di schermate differenti. Piccole variazioni dell'interfaccia possono comportare la non validità di un test e la necessità di riformularlo.
Verrà preso in considerazione un software per il GUI testing già esistente (Scout) il cui codice non è modificabile direttamente ma presenta la possibilità di sviluppare

plugin per estenderne le funzionalità. Sarà installato anche un plugin per la gamification sviluppato da Cacciotto [2] che ha già ottenuto notevoli risultati in termini di stimolazione dei tester e di aumento della coverage. A partire da questo ultimo plugin, dopo aver studiato le problematiche e i vantaggi del crowdsourcing, verrà sviluppato un plugin per il testing cooperativo sempre mantenendo attivo l'aspetto di gamification e quindi per passare da una visione puramente competitiva ad una anche collaborativa.
Alégroth e Bauer [3] hanno provato a concettualizzare una procedura di testing collaborativa che li ha visti però costretti ad interrompere il lavoro iniziato per i vari problemi riscontrati. Dalle sfide incontrate nel loro studio continuerà quindi questa tesi, provando a risolvere i problemi quali la definizione di "stato", il contesto di applicazione e l'algoritmo di unione dei casi di test.

1.1 Motivazioni

Il lavoro attuale prevede l'aspetto puramente competitivo della gamification che è molto stimolante per i tester, ma non considerando anche l'aspetto collaborativo si rischia di generare test duplicati o comunque di non riuscire a raggiungere una copertura totale delle possibili vie che un comune utente percorre nella sua visita dell'applicazione web.
Aggiungendo l'aspetto collaborativo alle funzionalità presenti delle quali è già stata verificato l'impatto positivo si vuole provare a dimostrare che è possibile migliorare l'efficienza di un tester.
La concettualizzazione di uno stato di un'interfaccia grafica negli strumenti attuali di testing non viene affrontata in modo rigoroso e la maggior parte della ricerca nell'area del testing è focalizzata sui test di unità di basso livello. Le caratteristiche dei test di interfacce grafiche, ponendosi ad un livello di astrazione superiore, introducono ulteriori requisiti e complessità in un ambiente di testing collaborativo.

1.2 Obiettivi

Questa tesi si pone l'obiettivo di dimostrare che l'utilizzo di tecniche di crowdsourcing ha un impatto positivo sulla generazione di casi di test per interfacce grafiche, nello specifico per quelli di applicativi web. Le metriche con le quali si descriverà questo impatto saranno la coverage media delle pagine visitate e la quantità di test generati.
Si affronta quindi uno studio di fattibilità e l'implementazione di meccanismi gamificati di cooperazione nel contesto di un software per il testing di interfacce web già esistente.

Nello specifico:

- Identificazione delle problematiche del crowdtesting e della sua applicazione ad un contesto web
- Definizione di un ambito di applicazione
- Implementazione dei meccanismi di cooperazione
- Adeguamento del contesto gamificato attuale all'ambiente collaborativo
- Valutazione della fattibilità e benefici apportati

Capitolo 2

Background e stato dell'arte

In questo capitolo verranno affrontati i 3 temi principali su cui è basata questa tesi:

- **Software testing**
- **Gamification**
- **Crowdsourcing**

2.1 Software testing

Il software testing è l'atto volto all'esaminazione degli artefatti e il comportamento del software sotto test con l'ausilio di tecniche di validazione e verifica.

Lo scopo principale è di ricercare "guasti" nel software così da rilevare eventuali difetti del codice. Non essendo possibile testare un prodotto in tutti i campi possibili ci si sofferma a quelli effettivi di applicazione rendendo così il test set, cioè l'insieme di test generati, finito.

Una definizione tratta da (Pierre Bourque et al., 2014) [4] definisce il testing come la verifica dinamica che un programma si comporti come ci si aspetta su un insieme finito di test, correttamente selezioni da un dominio normalmente infinito. Quest'ultima e come altre definizioni nella letteratura si riferisce al testing eseguito con un approccio dinamico.

2.1.1 Approcci del testing

Esistono diversi tipi di approcci per eseguire il testing:

- **Statico, dinamico e passivo**
- **Esplorativo**
- **"Box"**

Statico, dinamico e passivo

Il testing di tipo statico può essere eseguito attraverso revisioni o percorsi guidati attraverso il codice scritto dai singoli programmatori. Spesso anche gli strumenti di programmazione possono performare test statici analizzando il codice sorgente, la sintassi e il flusso dei dati.

L'esecuzione di un set di test case è invece definito come testing dinamico. Solitamente quest'ultima tipologia viene eseguita solo con il programma in esecuzione, anche se specifiche funzioni possono essere eseguite prima del completamento dell'intero algoritmo.

Come definito anche in (Oberkampf et al., 2010) [5] possiamo quindi concludere che il testing statico prevede la verifica, mentre quello dinamico la validazione.

L'ultimo approccio, definito come statico, prevede di verificare il corretto funzionamento del sistema attraverso l'analisi prodotti del software quali log e tracce, quindi può essere effettuato anche con un sistema offline.

Esplorativo

Termine coniato da (Cem Kaner) [6], definisce il testing esplorativo come uno stile che enfatizza la libertà personale e la responsabilità del singolo tester per ottimizzare continuamente la qualità del proprio lavoro. Questo tipo di testing indica come attività utili lo studio dell'ambiente, del design e l'esecuzione dei test.

Box

Il punto di vista che il tester ha dell'ambiente di test viene paragonato ad una scatola (box). Possono esserci testing di tipo white box oppure black box.

Nel caso del white box o testing strutturale, il tester conosce tutti i meccanismi interni del software, quindi deve verificare che tutti i percorsi possibili all'interno del codice non presentino problemi.
È necessario introdurre il concetto di coverage, che nell'informatica si riferisce ad una misura della percentuale di codice sorgente che viene eseguita quando una particolare suite di test viene eseguita. Un programma con una coverage di test elevata suggerisce che abbia meno probabilità di contenere dei bug rispetto ad uno con coverage bassa.
I principali costrutti analizzati per calcolare la coverage sono i seguenti:

- **Funzioni**: è stata chiamata ogni funzione del programma?
- **Enunciati**: è stato eseguito ogni enunciato del programma?
- **"Vertici"**: è stato raggiunto ogni vertice del diagramma del flusso di controllo?

- **"Condizioni"**: è stata valutata ogni espressione booleana sia nel caso vero che falso?

Solitamente con white box ci si riferisce ai test di unità che verificano il funzionamento di piccoli componenti autonomi del programma, ma può anche riferirsi ai test di integrazione (comunicazione tra più unità) e di sistema. Il tester deve generare dell'opportuno codice per comunicare con i singoli componenti del programma, potendo così avere informazioni sulla quantità di linee di codice testate.

Per il black box testing o testing funzionale, il tester vede il sistema come una scatola nera, quindi non riesce a vedere i singoli pezzi di codice e deve verificare che si comporti secondo specifiche e requisiti senza sapere cosa succede al suo interno. Un vantaggio di questa tipologia di testing è che non viene richiesta alcuna esperienza di programmazione, mentre uno svantaggio è la mancanza di informazioni sulla completezza dei test che ha generato, non avendo a disposizione le linee di codice effettivamente testate. Questo approccio di testing può essere applicato a tutti i livelli di testing: unità, integrazione, sistema e accettazione.

2.1.2 GUI testing

Questa tesi affronterà l'argomento del testing di interfacce web quindi è necessario comprendere perché è importante e come può essere eseguito.

Il GUI testing è il processo che si occupa di verificare, tramite l'uso delle interfacce grafiche, che un programma o sistema soddisfi i requisiti.

Rispetto al testing di parti del codice qui si presenta il problema della sequenza. Come riportato da (Memon et al., 2001) [7] alcune funzionalità del sistema potrebbero essere verificate solo attraverso una specifica sequenza di eventi e quindi aumentare il numero di possibili operazioni incrementa esponenzialmente le sequenze possibili. Le sequenze di eventi sono concettualmente ad un livello di astrazione superiore rispetto al codice e non sempre possono essere ottenute da esso.
Le interfacce sono per loro natura gerarchiche e questa gerarchia può essere sfruttata per identificare gruppi di eventi che possono essere testati in isolamento. Questa decomposizione in unità di test permette di considerare la coverage per ogni singola unità e non per tutte le sequenze di eventi che possono teoricamente generarsi.
Alcune sequenze potrebbero non essere percorribili e sta al test designer capire se un certo criterio di coverage possa essere soddisfatto. Il problema della ricerca dei percorsi irrealizzabili è di tipo NP (non deterministico polinomiale) completo.

Un altro problema si presenta con i test di regressione, ovvero l'esecuzione dei test sviluppati per versioni precedenti del programma. Bastano piccoli cambiamenti dell'interfaccia per far fallire quei test, anche se il sistema sottostante rimane lo stesso.

Questi problemi e l'elevato costo del metodo di verifica manuale hanno portato il GUI testing verso l'automazione metodi del tipo capture & playback.

Inizialmente venne usato un semplice metodo di cattura delle coordinate del mouse alla pressione del tasto di selezione. Il caso di test consiste nella comparazione delle schermate dell'interfaccia durante la registrazione e all'esecuzione di quest'ultimo, le quali devono risultare identiche. Ovviamente questa metodologia incorre nel problema delle differenze puramente estetiche o per i piccoli cambiamenti tra versioni differenti del software.

Attualmente la cattura delle coordinate del mouse si è evoluta nella cattura degli eventi, cioè le interazioni che avvengono nel sistema di finestre (windowing system) sottostante alla GUI. Questi eventi, catturati sotto forma di log, sono disaccoppiati dall'aspetto dell'interfaccia. Può avvenire anche un filtraggio degli eventi che possono essere molto dettagliati ma non sempre sono rilevanti al problema.
Come spiegato in (Ruiz et al., 2008) [8], il primo passo per rendere il testing semplice è progettare un'applicazione che lo permetta. Viene consigliato di utilizzare il pattern Model View Controller (MVC) in modo da non coinvolgere la GUI per verificarne il comportamento. È anche possibile inserire un driver nella GUI tale che permetta l'invio degli eventi direttamente alla logica sottostante.

Una tecnica di testing detta Augmented Testing, nata per affrontare le problematiche della frequente e costosa manutenzione degli script di test, è stata introdotta nello studio di (Nass et al., 2019) [9]. Questa tecnica, di tipo capture & playback, prevede un'interfaccia aggiuntiva sovraimposta su quella del System Under Test (SUT) che fornisce all'utente suggerimenti e dati di test, evidenzia aree importanti o propone altri supporti oltre a permettere di osservare e registrare gli input.

L'Augmented GUI è l'unica interfaccia presentata al tester ed è possibile interagire solo con essa. Un driver si occupa di tradurre gli input ricevuti alla GUI del SUT. Gli input possibili comprendono check, issue e azioni aumentate che verranno salvate in un modello e riprodotte sull'interfaccia del SUT.

Il tool Scout è stato proposto per dimostrare i benefici dell'augmented testing ed è specifico per le applicazioni web. In Scout il modello in cui vengono salvate le interazioni del tester è rappresentato con un grafo pesato, dove ogni nodo rappresenta uno stato dell'applicazione. Ogni stato comprende tutte le dipendenze come la memoria interna, i dati salvati e le applicazioni esterne in un certo momento. I nodi sono interconnessi da azioni che sono le azioni del tester, ovvero click del mouse o input da tastiera.

Come risultato di questo studio si è dimostrato che l'Augmented Testing aumenta le prestazioni di ogni tester, diminuendo i costi per effettuare i casi di test. Viene anche semplificato il lavoro del tester rendendo il tutto più intuitivo, potendo allargare il lavoro del tester anche a personale meno qualificato.

2.2 Gamification

La gamification (ludicizzazione) é un tentativo strategico per il miglioramento di sistemi, servizi, organizzazioni e attività per motivare e coinvolgere gli utenti. Consiste nell'uso di elementi e tecniche tipiche del design dei giochi applicate a contesti non ludici. Ciò si discosta dal concetto di *serious games* che invece si riferisce a giochi veri e propri ma con finalità tipicamente educativa.
Huotari e Hamari in (Huotari and Hamari, 2012) [10] evidenziano il ruolo della gamification nell'invocare le stesse esperienze psicologiche che generalmente offrono i giochi. Deterding et al. in (Deterding et al., 2011) [11], d'altra parte enfatizzano che gli elementi implementati con la gamification devono essere gli stessi usati nei giochi senza preoccuparsi dei risultati.
Gadiyar in (Gadiyar, 2014) [12] specifica che la gamification può essere suddivisa in tre livelli (Figura 2.1):

- *Livello uno/Gamification tradizionale*: molte organizzazioni si affidano a sfide predefinite per motivare gli utenti e influenzare il loro comportamento, includendo meccaniche standard dei giochi quali punti, livelli, classifiche, stemmi. Nonostante possa rivelarsi un buon strumento inizialmente, non è possibile coinvolgere gli utenti in un modo sostenibile.

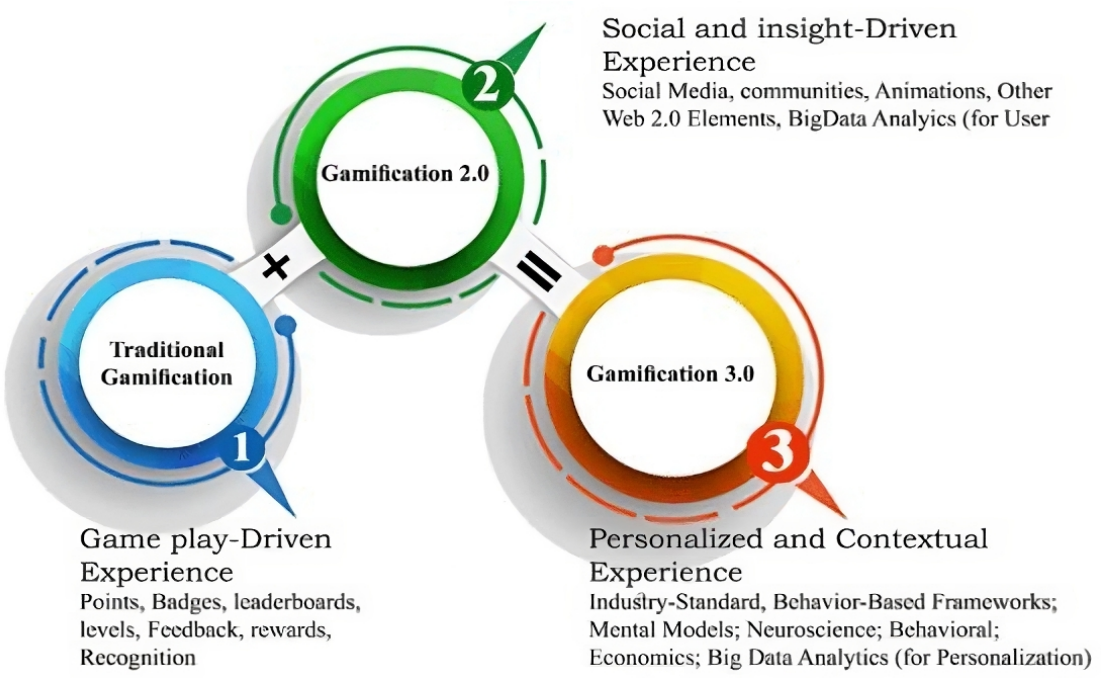


Figura 2.1: Livelli di gamification

- *Livello due/Gamification 2.0*: viene pianificata un'esperienza gamificata per gli utenti ma senza apprendere che tipologia di giocatore sia l'utente
- *Livello tre/Gamification 3.0*: il sistema è in grado di capire la personalità dei propri utenti e di proporre sfide e ricompense personalizzate

Essendo i giochi nati per intrattenere e attirare le attenzioni del giocatore, anche la gamification punta a migliorare le attività non ludiche portando alla produttività dell'utente, lo studio e il mantenimento delle conoscenze, facilità d'uso degli strumenti messi a disposizione, e molti altri benefici.
Nei risultati della ricerca di (Hamari et al., 2014) [13] si desume che la maggioranza degli studi identificano la gamification come qualcosa di positivo e che ha una buona influenza negli utenti di un sistema.

Il concetto di gamification è abbastanza recente e la sua nascita nel contesto del software risale al 2008. Come si può notare dalla Figura 2.2 vi è stata una diffusione tale che ha mantenuto alto l'interesse verso il fenomeno fino al giorno odierno con un picco nel 2022.

Figura 2.2: Interesse di ricerca rispetto al punto più alto del grafico del termine "gamification" sul motore di ricerca Google

Per riassumere con la definizione tratta da (Deterding et al., 2011) [11]: "gamification" si riferisce a

- *l'uso* (piuttosto dell'estensione) *di*
- *il design* (piuttosto della tecnologia alla base dei giochi)
- *caratteristiche per i giochi* (piuttosto di giochi a tutti gli effetti)
- *contesti non ludici* (senza preoccuparsi delle specifiche intenzioni di utilizzo o implementazioni)

2.2.1 Teorie sull'applicazione della Gamification

Non è semplice applicare gli elementi dei giochi in modo efficace in un ambiente non ludico e spesso può risultare controproducente. È quindi necessario introdurre

il concetto di gamification framework, ovvero un framework metodologico che si occupa del design e dell'integrazione della gamification nei processi in esame.

Consideriamo ora il "Processo di sviluppo 6D" introdotto in (Werbach e Hunter, 2015) [14] che è specifico per il mondo del business.
Vengono elencate 6D, ovvero dei consigli di design, che essendo tradotte in lingua italiana non sempre iniziano con la lettera "D":

- *Definisci i tuoi obiettivi di business*
- *Delinea le attitudini dei tuoi utenti*
- *Descrivi i tuoi giocatori*
- *Pianifica i cicli delle tue attività*
- *Non dimenticare il divertimento*
- *Utilizza gli strumenti appropriati*

Definisci i tuoi obiettivi di business

Bisogna prestare attenzione a mettere i propri obiettivi di business al primo posto. È possibile avere un sistema gamificato che riesca ad attirare a sé gli utenti, ma che non rispetta gli obiettivi del proprio business e quindi comporti una perdita di denaro. Gli obiettivi non devono essere solo monetari, ma bisogna considerare una necessità o qualcosa che l'utente vuole realizzare.

Si può scomporre questo punto in 4 passi pratici:

- Delinea tutti i possibili obiettivi
- Crea una graduatoria degli obiettivi
- Elimina gli obiettivi che sono un mezzo per arrivare ad altri fini
- Giustifica gli obiettivi e spiega perchè la tua organizzazione ne beneficerebbe

Delinea le attitudini dei tuoi utenti

Ora è necessario chiedersi le attività da proporre ai propri utenti e cosa gli piacerebbe fare. Anche qui vi sono alcuni passi da seguire:

- Fare brainstorming di tutte le azioni degli utenti, quali "scrivere un commento", "registrarsi", "sponsorizzare altri utenti", "acquistare monete virtuali" ecc.

- Creare una graduatoria per priorità delle azioni e mantenere solo alcuni dei primi in classifica
- Assegnare dei punti da 1 a 10 ad ogni azione in base all'importanza nell'ambito di riferimento
- Definire traguardi, ovvero combinazioni di azioni che possono comportare una ricompensa per gli utenti. Bisogna considerare sia traguardi giornalieri sia traguardi per i quali è necessario essere coinvolti per lunghi periodi. I traguardi di lungo periodo includono livelli, badge (ovvero elementi distintivi che indicano il raggiungimento di un traguardo) cumulativi e classifiche.

Descrivi i tuoi giocatori

Si passa quindi alla suddivisione dei giocatori in personalità. Le personalità individuate devono essere poche e comprendere almeno l'80% degli utenti. Creare un nome e un avatar per ogni personalità aiuta a motivare i giocatori.
Tipicamente vi sono tre livelli di abilità:

- Novizi: richiedono un aiuto passo per passo per iniziare a giocare
- Regolari: richiedono novità per continuare a giocare
- Esperti: richiedono sfide e aggiornamenti dello stato

Gli utenti possono essere suddivisi ulteriormente come riportato in (Bartle, 2008) [15]:

- Conquistatori: hanno bisogno di un senso di progressione e successo
- Esploratori: vogliono esplorare il mondo di gioco e imparare il più possibile
- Socializzatori: vogliono connettersi con altri utenti che hanno le stesse idee
- Killer: bramano le competizioni e il rango

Pianifica i cicli delle tue attività

Il progresso attraverso il sistema gamificato non dovrebbe essere visto come puramente lineare, ma dovrebbero esserci cicli e percorsi ramificati.
I cicli di coinvolgimento, di cui un esempio è riportato in Figura 2.3, forniscono feedback immediato e ulteriori azioni da compiere.

Per esempio:

- **Motivazione**: L'utente vuole salire di livello

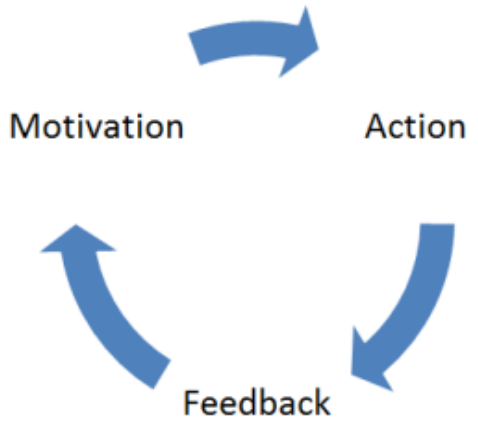


Figura 2.3: Ciclo di coinvolgimento

- **Azione**: L'utente clicca su un elemento
- **Feedback**: Il sistema risponde consegnando punti e riconoscimenti, con qualche sorta di feedback visuale e/o sonoro
- **Motivazione**: L'utente vuole salire di livello

Per immettere gli utenti in questi cicli possono essere usate delle comunicazioni del sistema, oppure può essere un loro amico a mandare una notifica. Una volta entrati in questi cicli, altri incentivi, come sfide giornaliere o avvisi inerenti alle classifiche possono stimolare un'azione da parte degli utenti.

Ovviamente il precedente esempio è molto semplice e servono cicli molto più complessi ed efficaci. I cicli si possono costruire attorno alle "trigger stories" ovvero uno strumento per evidenziare gli inneschi, gli obiettivi e i risultati che si vogliono ottenere.

Quando [innesco]
Voglio che [azione orientata al raggiungimento di un obiettivo]
Così posso [risultato desiderato]

Un altro utile strumento per inquadrare la macro prospettiva dell'intero viaggio del giocatore sono le "scale del progresso" (Figura 2.4).

Nella prima fase la difficoltà viene mantenuta bassa con frequente feedback positivo così gli utenti riescono ad imparare senza spiazzanti penalità.
Nella seconda fase l'utente ha acquisito le abilità basilari e la difficoltà di avanzamento deve essere aumentata gradualmente per mantenerli impegnati.
La terza fase è quella del riposo. La difficoltà non dovrebbe aumentare continuamente altrimenti il prossimo obiettivo verrebbe visto come troppo lontano da

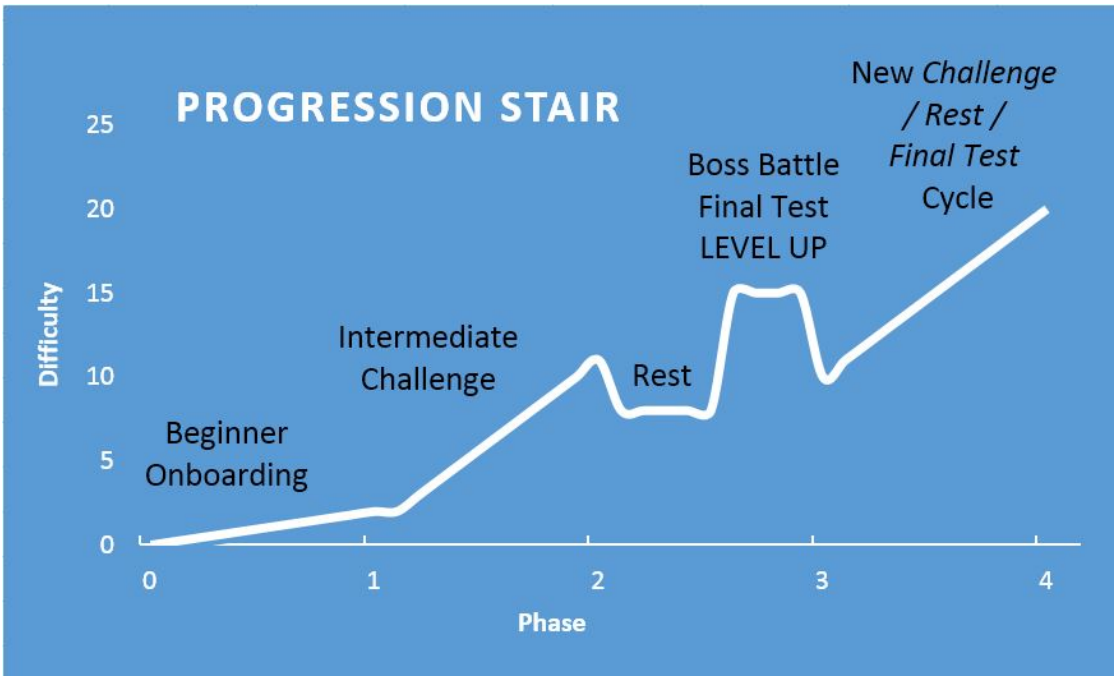


Figura 2.4: Scala del progresso

raggiungere. L'utente deve sentirsi potente prima di affrontare il prossimo test. L'ultimo step per far sentire l'utente appagato, prima di ripetere il ciclo, è la battaglia finale con un boss e l'avanzamento di grado.
Sono importanti ricompense casuali lungo il percorso che fanno sentire gli utenti ripagati delle loro fatiche.

Non dimenticare il divertimento

Un'importante euristica che rischia di essere dimenticata è che i giochi sono divertenti e quindi anche il sistema gamificato deve essere tale. Per approfondire quali sono i tipi di divertimento prendiamo in considerazione (Lazzaro, 2004) [16] di cui lo schema in Figura 2.5:

- **Divertimento forte**: divertimento per aver superato qualcosa
- **Divertimento leggero**: sfogarsi
- **Divertimento sperimentale**: provare nuove identità, imparare cose nuove
- **Divertimento sociale**: interazione con altri utenti

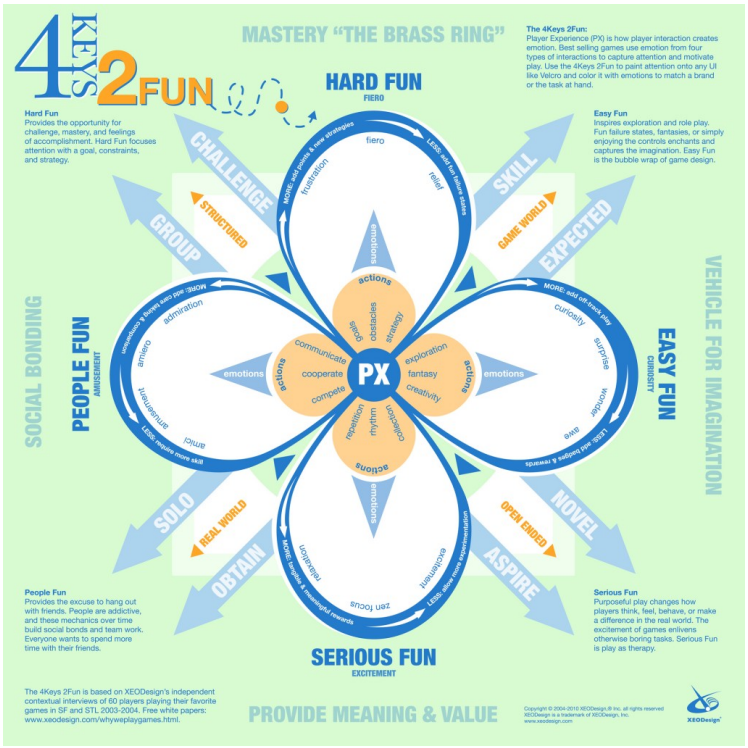


Figura 2.5: Descrizione analitica del divertimento proposta da Lazzaro

Utilizza gli strumenti appropriati

A livello pratico si rende necessario trovare cosa funziona attraverso un'analisi competitiva e iterativa. Aiuta analizzare piattaforme simili a quella che si vuole sviluppare e capire i punti di forza e quelli deboli.
La gamification non è solo una scienza che si può implementare sulla base dei dati, ma anche un'arte che cerca di capire gli umani e cosa li intrattenga. Per capire cosa funzionerebbe nel mondo reale è testare il tutto su soggetti reali e raccogliere i loro feedback, avanzando di iterazione in iterazione.

Un altro framework presentato da Marczewski nel 2012 e disponibile in (Marczewski, 2012) [17] si presenta come molto simile al 6D framework ma aggiunge alcuni importanti dettagli.

Il primo punto per cui si discosta dal framework identificato precedentemente è la divisione degli utenti per personalità e aggiunge 3 nuovi tipi:

- **Spiriti liberi**: sono motivati dall'autonomia e dall'espressione del loro essere
- **Filantropi**: sono motivati dallo scopo e il significato nelle cose. Sono altruisti e vogliono arricchire le vite di altre persone senza richiedere nulla in cambio

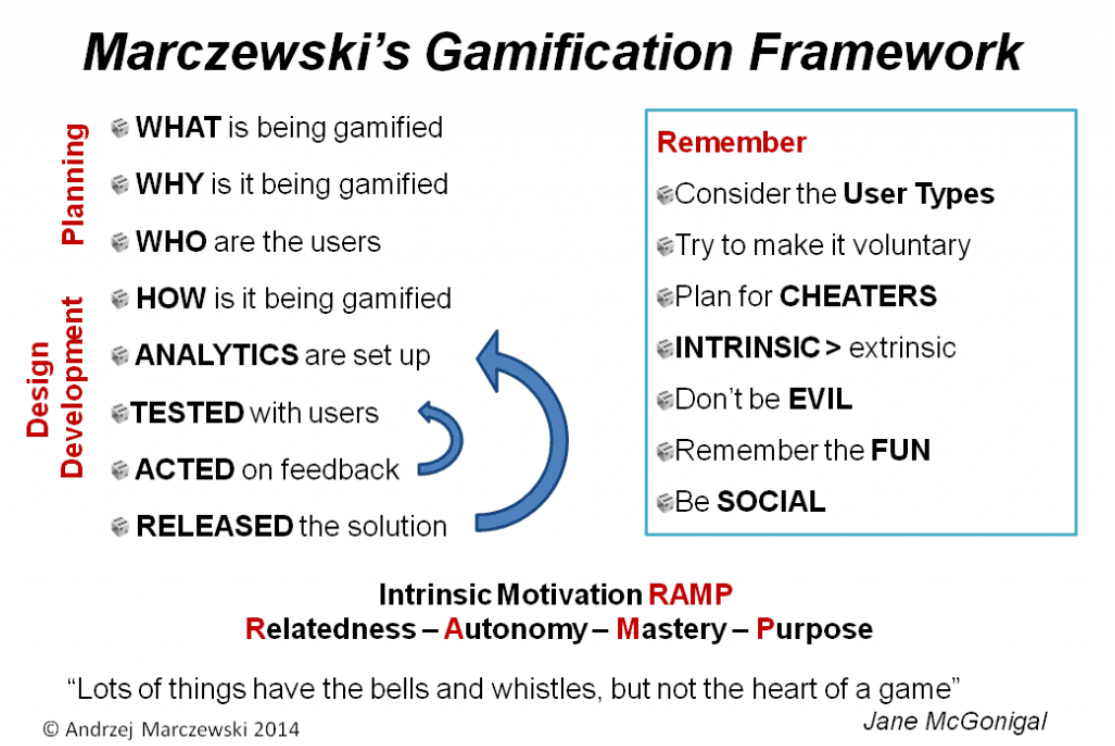


Figura 2.6: Framework di Marczewski

- **Distruttori**: sono volti al cambiamento. Desiderano la distruzione del sistema creato, sia direttamente oppure indirettamente attraverso altri utenti

Marczewski evidenzia l'importanza del testing continuo sugli utenti e la richiesta di feedback per poter migliorare continuamente il sistema. Vigono altri criteri basilari per creare un ambiente gamificato sano e durevole quali:

- provare a rendere la partecipazione degli utenti volontaria in quanto i volontari sono giocatori migliori
- prepararsi a chi vuole imbrogliare per arrivare subito a superare missioni difficili e ricevere ricompense
- non essere malvagi e usare le persone per i propri scopi perchè se ne accorgeranno e smetteranno di usare il sistema
- le meccaniche sociali sono essenziali per un coinvolgimento a lungo termine

Per realizzare un sistema gamificato di livello 3.0 è nato un framework che si basa sul concetto già esistente di Business Model Canvas (BMC). La BMC è uno

strumento molto utile per implementare o documentare idee del business, misurarne l'efficacia e identificare punti forti e debolezze. Si tratta di un diagramma con 9 liste a forma di blocco che devono essere riempite dal business designer per capire quali sono le attività rilevanti che saranno dedite alla generazione del profitto. Il modello dovrebbe aiutare gli sviluppatori ad allineare le loro attività con i potenziali obiettivi aziendali.
A partire dalla BMC è stata sviluppata la Lean Gamification Canvas (LGC) da parte di (Yousefi et al., 2020) [18], basata sull'idea di "gamification 3.0" ed è composta da 13 blocchi. Come si può vedere in Figura 2.7 si tratta di un modello molto semplice ma altrettanto efficace che mette in risalto i metodi per proporre valore e le necessità degli utenti. La LGC si può suddividere in quattro parti.
La parte destra si concentra sui giocatori/user e quella sinistra simula le idee e gli elementi che saranno realizzati nell'ambiente gamificato per produrre un valore. Nella sezione centrale vi sono influenze sia dalla parte destra che quella sinistra, quindi è necessario fondere il soggettivo e l'oggettivo per proporre un valore. La parte inferiore è specifica per gli aspetti manageriali come costi, risorse umane e in generale i criteri che si impongono sull'effettiva realizzazione del sistema.

Come ultimo framework e più completo viene ora introdotto Octalysis proposto

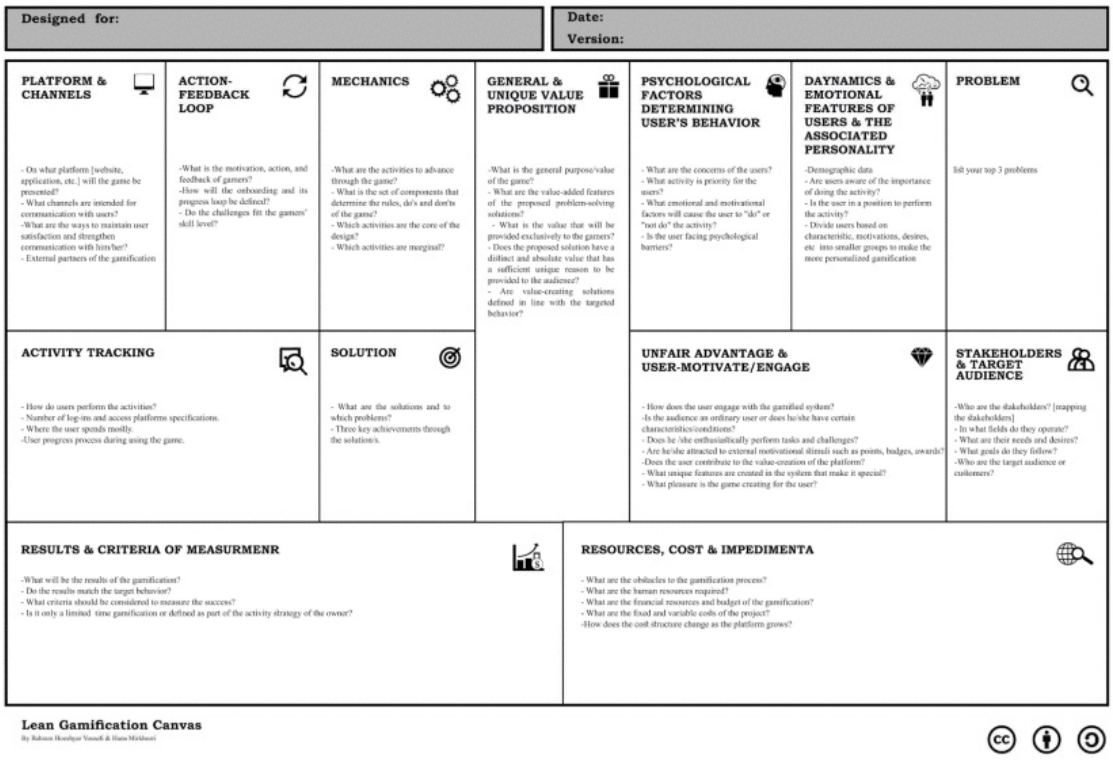


Figura 2.7: Lean Gamification Canvas

da Yu-kai Chou in [19]. L'autore considera il processo di gamification come un passaggio da un design funzionale, cioè che deve portare a termine il lavoro in breve tempo, ad un design focalizzato sull'essere umano che ha emozioni, insicurezze e motivi per fare o non fare qualcosa. I giochi hanno come scopo quello di gratificare chi li usa e sono stati progettati per aumentare la motivazione e il coinvolgimento del singolo.
Il risultato della ricerca ha portato a Octalysis, progettato come un ottagono con 8 elementi trainanti ai propri vertici (Figura 2.8):

- **senso epico e chiamata (epic meaning & calling)**: il senso epico rappresenta il desiderio di partecipare a qualcosa di più grande e di essere scelto per contribuire alla creazione di un bene collettivo. Questo elemento viene in gioco anche quando qualcuno pensa di avere la "fortuna del principiante" e pensa di ricevere come ricompensa qualcosa che altri non ricevono
- **progresso e senso di realizzazione (development & accomplishment)**: è l'elemento che si presenta con il progresso, aumentando le proprie abilità e

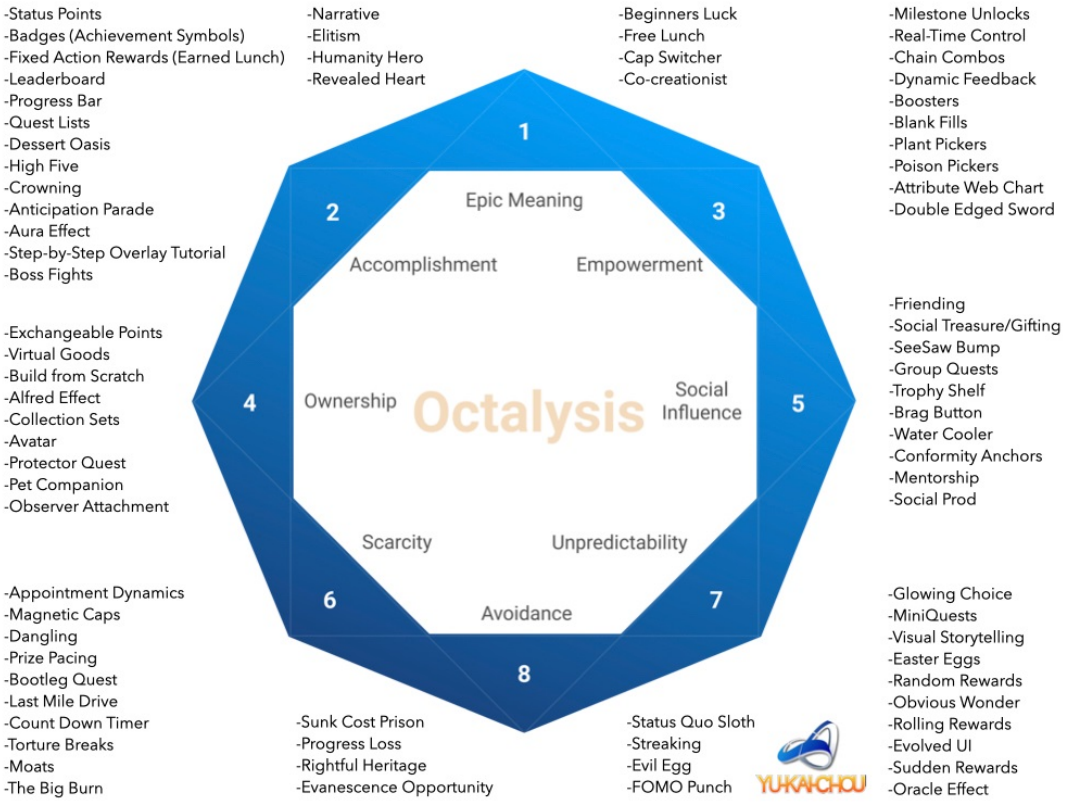


Figura 2.8: Octalysis framework

affrontando sfide. Come ricompensa allo sforzo fatto vi sono punti, badge e classifiche che indicano il progresso.

- **potenziamento della creatività e feedback (empowerment of creativity & feedback)**: vengono proposte sfide in cui l'utente deve esprimere la propria creatività per superare delle sfide e, vedendo i risultati delle proprie azioni ricevono immediatamente un feedback. Questo è un elemento fondamentale poiché essendo la creatività pressoché infinita una stessa sfida può comportare soluzioni sempre differenti ed è questo che mantiene le attività coinvolgenti.

- **proprietà e possesso (ownership & possession)**: gli utenti risultano motivati perchè si sentono in possesso di qualcosa che vogliono custodire con cura e aumentare nel tempo. Non si parla solo di oggetti e denaro, ma anche di semplici cose come un avatar in cui è stato investito molto tempo per essere reso di proprio gradimento.

- **influenza sociale e relazione (social pressure & relatedness)**: Questo elemento incorpora tutti gli elementi sociali che spingono le persone a compiere determinate azioni sotto l'influsso di fattori di pressione sociale come amicizia, competizione e invidia. Risulta uno degli elementi più utilizzati dalle organizzazioni.

- **scarsità e impazienza (scarcity & impatience)**: spesso le persone desiderano cose solo perché non possono averle. La dinamica degli appuntamenti (torna tra 2 ore per ricevere la ricompensa) viene spesso implementata proprio perché non potendo avere la ricompensa nell'immediato le persone tornano sulla piattaforma e ci pensano a quella cosa tutto il giorno.

- **imprevedibilità e curiosità (unpredictabilty & curiosity)**: l'interesse nell'ignoto è da sempre uno degli elementi che guida il nostro essere. Se non si sa cosa accadrà, la nostra testa ci rifletterà e inizierà a fare teorie di come possa andare.

- **paura della perdita ed evasione (loss & avoidance)**: la possibilità che accada qualcosa di negativo, ad esempio perdere qualcosa che si è costruito nel tempo, spinge le persone a continuare quello che stavano facendo anche se non è più di proprio gradimento. Anche le opportunità fanno parte di questo elemento poiché spingono le persone ad agire per non rischiare di non aver più l'opportunità di agire.

Gli elementi trainanti sulla parte destra dell'ottagono vengono riferiti alla parte destra del cervello ovvero quella relativa alla creatività, all'espressione di se stessi e agli aspetti sociali. Gli elementi sulla sinistra sono riferiti alla parte sinistra del

cervello e sono relativi alla logica, i calcoli e il possesso. La divisione del cervello in due parti non è una scienza vera e propria ma aiuta a rendere il framework più semplice ed efficace, dividendo aspetti della logica da quelli di tipo emozionale. Questa divisione è anche data dal fatto che gli elementi a sinistra sono motivatori estrinseci, ovvero motivano le persone perchè vogliono ottenere qualcosa, mentre gli elementi a destra sono intrinseci e non c'è bisogno di un obiettivo per volere qualcosa ma è direttamente l'attività a fare da ricompensa. I motivatori intrinseci sono molto più efficaci per un sistema sostenibile e duraturo.

Un'altra suddivisione dell'ottagono si ha fra parte superiore e inferiore. Gli elementi della parte superiore sono considerati motivatori molto positivi e chi li utilizza si avvale della "White Hat Gamification", al contrario la parte inferiore è considerata negativa e la gamification che ne deriva viene chiamata "Black Hat Gamification".

Se qualcosa è coinvolgente perchè permette di esprimere la propria creatività, sviluppare nuove abilità e ti fa sentire importante, allora l'utente si sentirà molto bene e potente. L'utente non si sentirà bene e resterà con "l'amaro in bocca" se è

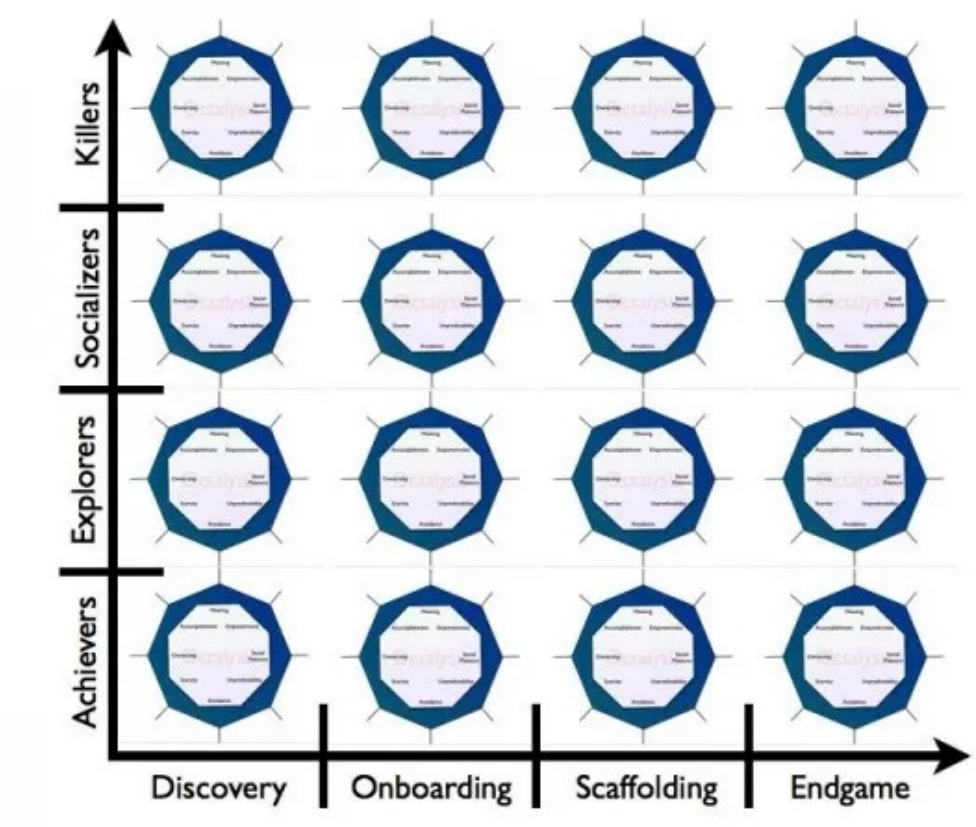


Figura 2.9: Octalysis a 3 livelli

continuamente con la sensazione di perdere qualcosa o se ci sono cose che non può avere.
Un buon esperto di gamification considererà tutti e 8 gli elementi trainanti per avere un sistema positivo e produttivo che riesca a soddisfare sia gli utenti che chi lo offre al pubblico. Per utilizzare Octalysis è necessario assegnare un punteggio ad ogni vertice dell'ottagono e verificare che i punteggi siano bilanciati e non si utilizzi solo uno degli elementi.

Considerare solo gli elementi forniti dall'ottagono è visto come il livello 1 di Octalysis. In livelli successivi vengono aggiunte altre dimensioni di analisi. Ad esempio con il secondo livello si introduce la dimensione temporale e sarà necessario prevedere come si comporterà il sistema gamificato nelle varie fasi, dalla scoperta di un utente, alla sua iniziazione, all'uso regolare del sistema e infine alla fine del gioco (come mantenere gli utenti che sono considerati esperti). Al terzo livello vengono introdotte le personalità degli utenti, che anche in questo caso si riferiscono a (Bartle, 2008) [15], ovvero conquistatori, esploratori, socializzatori e killer (Figura 2.9). Ulteriori livelli possono essere esplorati aggiungendo altre dimensioni e rendendo ancor più complicata la fase di progettazione.

2.2.2 Applicazione nell'istruzione

Negli anni vi sono stati molti tentativi di applicare la gamification non solo al mondo del business ma anche nell'istruzione. Soffermandoci solo sui corsi universitari dell'ingegneria del software, il documento di (Alhammad e Moreno, 2018) [20] fornisce una buona analisi di ciò che è stato fatto in un insieme di corsi universitari e i risultati ottenuti da tale applicazione.

La gamification è stata inserita in 4 contesti: lezioni, sessioni di laboratorio, compiti da svolgere in autonomia e progetti. L'ambito in cui è stata maggiormente applicata sono le lezioni con strumenti quali barre di progresso, sfide tramite quiz su piattaforme mobili, piccoli problemi da risolvere. I progetti degli studenti sono stati modificati in modo includere la competizione e ricompense per aver completato dei requisiti funzionali e aver scritto codice di buona qualità. Per quanto riguarda i compiti sono stati assegnati con le sembianze di sfide per le quali si potevano delineare definite fasi di progresso.

Gli obiettivi iniziali legati all'introduzione della gamification si possono dividere il quattro categorie: aumento del coinvolgimento, aumento delle performance e conoscenza, utilizzo delle "best practice" dell'ingegneria del software e infine il miglioramento della capacità di lavoro di gruppo. Degli studi trattati, solo quattro hanno riportato un impatto positivo sulle prestazioni degli studenti, mentre due hanno indicato un effetto negativo e uno non ha riportato alcun effetto.
Dai risultati di questi studi è possibile identificare alcuni punti sui quali è necessario lavorare:

- solo pochi casi di studio hanno seguito dei framework e si sono basati su idee nate dalla letteratura
- per riuscire a migliorare l'efficacia della gamification sono necessari più studi empirici, perché solo con tramite la sperimentazione si riescono a delineare gli elementi positivi e come inserirli nei contesti non ludici
- alcuni elementi denotati come negativi possono essere stati tali perchè hanno rappresentato un ulteriore carico di lavoro invece che facilitare le attività proposte, quindi piuttosto di introdurre qualcosa di nuovo devono essere applicati in simbiosi alle attività già esistenti

2.2.3 Applicazione nell'ingegneria del software

Consideriamo ora lo studio (Morschheuser et al., 2017) [21], volto all'identificazione di un metodo per progettare correttamente un software gamificato.
Inizialmente sono stati intervistati 25 esperti di gamification e dalle loro interviste e altri materiali raccolti si sono delineati 13 importanti principi di design. A seguire l'elenco dei principi formulati ai quali è stato assegnato un identificativo che verrà riproposto in seguito:

- **DP1.** Comprendere le necessità degli utenti, le motivazioni e i comportamenti, come anche le caratteristiche del contesto
- **DP2.** Identificare gli obiettivi del progetto e definirli chiaramente
- **DP3.** Testare le idee di gamification proposte quanto prima possibile
- **DP4.** Seguire un processo di design iterativo
- **DP5.** Avere una profonda conoscenza di game-design e psicologia umana
- **DP6.** Assicurarsi se la gamification sia la giusta scelta per raggiungere gli obiettivi
- **DP7.** Gli investitori e le organizzazioni devono aver comprende a fondo il concetto di gamification e supportarla in ogni modo
- **DP8.** Durante la fase di ideazione è necessario considerare i bisogni degli utenti
- **DP9.** Definire metriche per la valutazione e il monitoraggio del successo, così come gli effetti psicologici e comportamentali dovuti all'applicazione della gamification

- **DP10.** Controllo di eventuali imbroglioni o di chi vuole aggirare il funzionamento pianificato
- **DP11.** Ottimizzare continuamente il design della gamification
- **DP12.** Considerare limiti legali ed etici nella fase di design
- **DP13.** Includere gli utenti nella fase di ideazione e design

Dopo aver raccolto i metodi di sviluppo da oltre cento attività e i precedenti principi di design, gli autori hanno sviluppato un nuovo metodo di sviluppo di software gamificato. Alcuni principi di design possono apparire in più fasi del metodo in base alle attività del metodo stesso. In seguito la Tabella 2.1 che indica solo le macrofasi senza entrare nel dettaglio:

Fase del metodo	Principi di design utilizzati
1. Preparazione del progetto	DPs: 2, 6, 7, 9
2. Analisi del contesto e degli utenti	DP: 1
3. Ideazione	DPs: 8, 13
4. Design	DPs 3, 4, 5, 12, 13
5. Implementazione del design	DPs 4, 11, 13
6. Valutazione	DP: 9
7. Monitoraggio	DPs: 9, 10, 11

Tabella 2.1: Mappatura dei principi di design alle fasi del metodo di gamification

La valutazione del metodo sviluppato ha indicato che il metodo risulta essere comprensivo, completo e fornisce un'utilità pratica oltre che indirizzare punti cruciali che non sono stati d'interesse per studi precedenti.

2.2.4 Applicazione nel software testing

La fase di testing è spesso considerata noiosa, ripetitiva e non in grado di stimolare l'attenzione del tester. I test possono essere generati automaticamente ma è una pratica abbastanza limitante: quello che rende un test buono è spesso frutto di intuizione umana e la comprensione del contesto, mentre l'automazione potrebbe applicare metriche sbagliate (Inozemtseva and Holmes, 2014) [22]. Per tale motivo entra in gioco la gamification anche nel campo del software testing, promettendo coinvolgimento e competizione.

Un esempio di applicazione nel campo del testing è Code Defenders, ideato da (Rojas e Fraser, 2016) [23]. Questo tool è specifico per il mutation testing che consiste nella creazione di versioni leggermente modificate di uno stesso programma detti anche *mutanti*. L'attività di mutation testing ha come obiettivo la creazione di test efficaci e completi capaci di identificare i mutanti che non rispettano i requisiti del programma.

Code Defenders prevede due ruoli: gli attaccanti e i difensori. Gli attaccanti devono introdurre bug nel codice e creare i mutanti, mentre i difensori devono migliorare la test suite esistente per resistere a più mutanti possibili. Un mutante riesce a superare tutti i test della test suite esistente, quindi gli attaccanti partono col vantaggio della conoscenza di cosa possa essere un mutante.

Vi è un sistema a punti che ne assegna agli attaccanti in caso riescano a generare mutanti che non vengono individuati dai nuovi test. I difensori ricevono punti per ogni mutante correttamente individuato. Alcuni mutanti non garantiscono dei punti poiché sono semanticamente uguali al codice originale, ma per una differenza sintattica riescono ad eludere i test. I difensori, qualora sospettassero dell'equivalenza di un mutante, possono duellare l'attaccante che lo ha generato e l'attaccante dovrà proporre un test che individui il mutante per dimostrare la non equivalenza, con il rischio della perdita di punti nel caso non ci riuscisse.

Nello studio (Rojas et al., 2017) [24] il gioco Code Defenders è stato analizzato in uno scenario reale. I risultati positivi sono stati molteplici tra i quali la generazione di test più forti rispetto allo stato dell'arte dei tool di generazione automatica di test e la creazione di una varietà di mutanti più difficilmente individuabile.

Un concetto importante nel ciclo di sviluppo di un software è rappresentato dalla tracciabilità. Con questo concetto si intente la capacità di descrivere e seguire il ciclo di vita degli artefatti software, ed è descritto dalle connessioni di artefatti correlati. Il supporto alla tracciabilità si rende necessario per assistere gli ingegneri del software nella comprensione, sviluppo, e gestione efficace dei sistemi. Riuscire ad ottenere una tracciabilità economica può essere un fattore critico nel successo di un progetto e questo ha portato Parizi alla creazione di un framework gamificato per la tracciabilità in (Parizi, 2016) [25].

La tracciabilità svolge un ruolo importante anche nel testing con il concetto di test-to-code traceability. Fornire al testing la tracciabilità degli artefatti è utile per ricavare informazioni utili sul sistema, localizzare facilmente codice malfunzionante, analizzare l'impatto dei cambiamenti, trovare i test più efficaci e rimuovere quelli ridondanti.

Le ricerche di Parizi hanno poi portato ad un'implementazione pratica denominata *GamiTracify* utilizzando Microsoft Visual Studio 2013. Nel framework si fa pressione sugli sviluppatori affinché durante la scrittura dei test case vengano anche tracciati tutti gli artefatti coinvolti in quello specifico test e che qualora

qualcosa cambiasse anche le informazioni di tracciabilità vengano aggiornate. Per coinvolgere gli sviluppatori nel processo viene proposto un sistema gamificato con elementi tipici quali punti, avatar, indicatori di tempo trascorso, feedback, sfide.

Il framework è stato messo a confronto con uno equivalente e non ludicizzato chiamato SCOTCH (Slicing and Coupling based Test to Code trace Hunter) tramite una sessione sperimentale in cui hanno partecipato 18 candidati con esperienza nel mondo dello sviluppo del software. Le analisi dei risultati ottenuti indicano come GamiTracify porti ad una maggiore accuratezza del tracciamento degli artefatti di test con meno falsi positivi rispetto a SCOTCH oltre ad aumentare la motivazione dei soggetti coinvolti.

2.3 Crowdsourcing

Il crowdsourcing è lo sviluppo collettivo di un progetto da parte di una moltitudine di persone esterne all'azienda ideatrice. La parola deriva da "crowd" quindi folla e "outsourcing" ovvero il concetto di delegare alcuni processi interni di un'azienda ad una esterna. Il crowdsourcing rispetto all'outsourcing preferisce coinvolgere gruppi pubblici di partecipanti invece di aziende specifiche.

Avvalersi del crowdsourcing porta con sé alcuni vantaggi tra i quali costi inferiori, velocità e qualità produttive superiori, flessibilità e scalabilità del lavoro, promuovendo la diversità. Con l'avvento delle piattaforme digitali il crowdsourcing è diventato un utile e facilmente implementabile strumento in quanto è possibile avvalersi di una platea molto vasta di persone con una quantità di risorse limitata.

Si possono definire due tipi di crowdsourcing:

- **Crowdsourcing esplicito**: gli utenti collaborano per condividere e portare a termine specifici compiti. Gli utenti possono anche creare artefatti elaborando il lavoro di altre persone.
- **Crowdsourcing implicito**: gli utenti risolvono problemi come effetto collaterale di quello che stanno facendo.

Brabham in (Brabham, 2013) [26] evidenzia come ci possano essere quattro principali approcci per l'utilizzo del crowdsourcing dipendenti dal problema che si sta affrontando:

- Nel caso dei problemi di gestione delle informazioni in cui viene mobilitata la folla per cercare e assemblare informazioni vengono usate la scoperta e gestione della conoscenza. È ideale per creare risorse collettive.
- Per i problemi di gestione delle informazioni in cui l'organizzazione possiede un set di informazioni e mobilita la folla per processarle e analizzarle vengono

usati compiti distribuiti di intelligenza umana (Distributed Human Intelligence Tasking). È ideale per compiti di processamento di set di dati molto vasti che i computer non riuscirebbero a svolgere. Un esempio reale è Amazon Mechanical Turk.

- Per problemi di ideazione dove un'organizzazione si avvale del pubblico per trovare una soluzione che sia oggettiva e provabile, viene usata una ricerca broadcast. Ideale per problemi di natura scientifica.

- Nel caso di problemi di ideazione dove un'organizzazione si avvale del pubblico per trovare una soluzione che sia soggettiva o dipendente dal supporto pubblico, viene utilizzata una produzione creativa veicolata da pari.

Un fattore molto importante per il successo di un progetto che utilizza il crowdsourcing è la scelta del pubblico. Il concetto di base del crowdsourcing indica come possibile platea tutte le persone interessate al particolare lavoro, ma spesso ci si ritrova con partecipanti non qualificati e quindi grandi quantità di contributi inutilizzabili o di bassa qualità. Si rende necessario selezionare il pubblico adeguato ed questo è quanto considerato da iCrowd (Fan, 2015) [27], uno specifico framework per il crowdsourcing che analizzando i contributi di un utente riesce ad assegnare i task in modo adeguato per ottenere risultati migliori in termini di qualità e interesse del partecipante.
Alcuni metodi di controllo della qualità dei risultati si basano sulla ridondanza, ovvero assegnano un task a più utenti e derivano il risultato dall'aggregazione delle risposte. Un approccio molto basilare si basa sulla votazione e il risultato che è stato fornito dalla maggioranza degli utenti viene ritenuto corretto. Gli approcci più moderni si basano sulla strategia di aspettazione-massimizzazione o Expectation-Maximization (EM) e i lavoratori di "bassa qualità" per il lavoro scelto vengono eliminati. Il metodo per distinguere tra lavoratori buoni e cattivi prevede di assegnare dei test di qualifica per dare una prima valutazione delle qualità dell'utente. Un lavoratore che dà buone risposte ai test di qualifica non è detto che riesca a fornire risultati altrettanto buoni nei task assegnati quindi è necessario aggiornare adattivamente il profilo di un utente.
Il framework iCrowd riesce a stimare in corso d'opera l'accuratezza di un utente valutando le performance sul task appena completato e riesce a dedurre le sue performance su task simili. Quando un partecipante richiede un task, il framework gli assegna quello che è più appropriato al suo profilo mettendolo a confronto con tutti gli altri utenti online in quel momento. Il framework ha mostrato buoni risultati con performance degli utenti superiori del 10% - 20% rispetto agli altri strumenti considerati lo stato dell'arte.

Motivazione

Janzik in (Janzik, 2010) [28] sostiene che il successo di una piattaforma di crowdsourcing dipenda in gran parte dai propri membri e la loro motivazione nella partecipazione. La motivazione determina la qualità e la quantità dei contributi.

Alcuni studi di Amabile (Amabile, 1996) [29] [30] hanno rivelato che i motivatori intrinseci accelerano la creatività, mentre quelli estrinseci la inibiscono, quindi per compiti di tipo creativo è meglio affidarsi a quelli intrinseci. In alcuni casi la motivazione estrinseca si può trasformare in intrinseca. Ricompense per compiti di per sé interessanti possono aumentare ancor più la motivazione di un utente.

La ricerca di Mokter (Mokter, 2012) [31] si rivela molto utile per raggruppare tutti i principali fattori utili nella motivazione degli utenti di una piattaforma online.
Gli studi sulla motivazione della partecipazione in piattaforme online sono partiti dal campo dell'open source: programmatori di alto livello dedicano tempo volontariamente e come hobby alla creazione di software disponibile al pubblico.
La prima ragione che spinge una persona a partecipare ad una piattaforma online è per fornire aiuto alla comunità. La reputazione, il piacere, il renderlo un hobby e l'immagine di se stessi sono altri motivi dominanti. Nel caso di dipendenti pagati, è necessario rendere chiari i benefit in quanto sono la loro motivazione di partecipazione.
La motivazione è poi stata suddivisa da Mokter in due tipi (Tabella 2.2):

- **estrinseca**: devono esistere dei fattori esterni per motivare le persone a compiere qualcosa poiché si aspettano una ricompensa per il lavoro svolto. Alcuni motivatori estrinseci sono benefit, pagamenti monetari, premi, riconoscimento. Questo tipo di motivazione si può ulteriormente suddividere in tre gruppi: finanziaria, sociale e organizzazionale

- **intrinseca**: le persone fanno qualcosa e non si aspettano nulla come ricompensa. Alcuni esempi di motivatori intrinseci sono hobby, interessi

2.3.1 Teorie del crowdsourcing

Ghezzi in (Ghezzi et al, 2018) [32] ha descritto il processo di crowdsourcing attraverso un framework già esistente denominato Input Process Output (IPO) e creato da McGrath nel 1964. Il framework è stato introdotto per la collaborazione tra gruppi, ma è stato utilizzato più volte per studi nel campo del management perché riesce ad evidenziare gli antecedenti, componenti e risultati di un processo. Il nuovo framework costruito da Ghezzi (Figura 2.10) si divide in tre parti:

- **Input**: il problema o task che deve essere risolto dalla folla

Motivatori estrinseci		
Motivatori finanziari	**Motivatori sociali**	**Motivatori organizzazionali**
Benefit	Obbligo	Carriera
Soldi	Collaborazione	Marketing
Insoddisfazione	Ego	Prestigio professionale
Opportunità di lavoro	Esperienza	Reclutamento
Necessità	Frustrazione	Responsabilità
Pressione	Conoscenza	
Ricavi	Raccolta	
	Collegamenti	
	Riconoscimento tra pari	
	Forza	
	Privilegi	
	Reputazione	
	Sviluppo di abilità	
	Legami sociali	
	Interazioni sociali	
	Status	
Motivatori intrinseci		
Carità	Altruismo	Idee
Competenza	Autonomia	Verità
Desiderio di risolvere	Appartenenza	Apprendimento
Piacere	Spinte comunitarie	Passatempo
Divertimento	Determinazione personale	Orgoglio
Soddisfazione		Realizzazione

Tabella 2.2: Classificazione dei motivatori in una piattaforma di crowdsourcing

- **Processo**: scomponibile ulteriormente in quattro problemi: gestione della sessione di crowdsourcing, gestione delle persone, gestione delle conoscenze, tecnologie utilizzabili per gestire i processi di crowdsourcing
- **Output**: possono essere soluzioni definitive o che devono essere ulteriormente processate perché parte di un problema più grande

Input

Una delle dimensioni per classificare i progetti di crowdsourcing è il tipo di problema. Il problema si può descrivere attraverso tre parametri: le conoscenze richieste

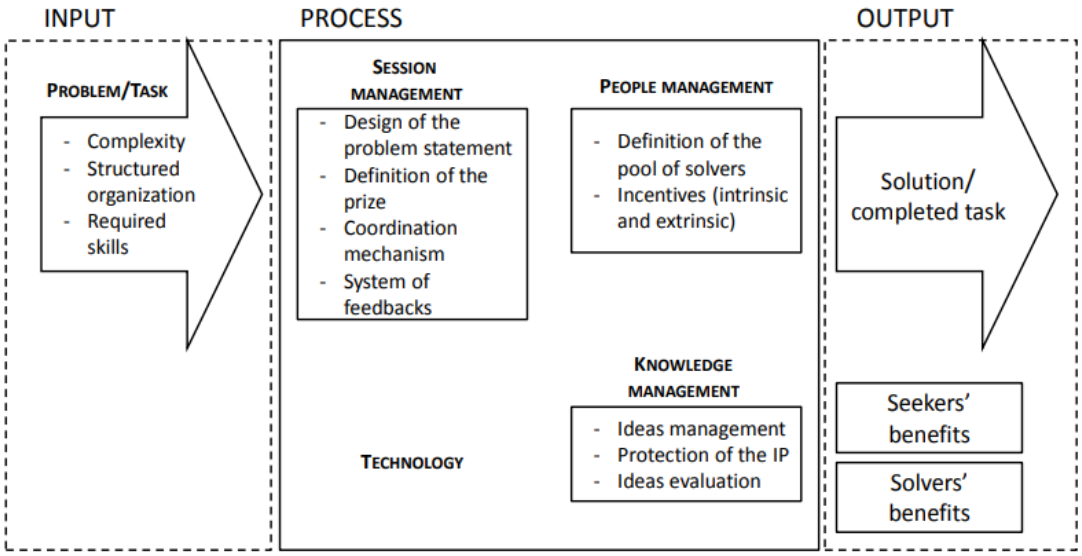


Figura 2.10: Input-Process-Output (I-P-O) framework per il crowdsourcing

affinché i partecipanti partecipino attivamente, la tipologia e la struttura. In termini generali i task/problemi dovrebbero essere semplici, facili da inquadrare, ben specificati e possibilmente modulari, quindi non deve esserci nulla di confidenziale né critico affinchè il partecipante riesca a capire le dinamiche di ciò su cui lavorerà.

Processo

Come prima cosa bisogna definire il problema. Come già detto il problema deve essere semplice, capibile, se necessario si può scomporre in problemi più semplici e la sua risposta deve essere non ambigua, facile da valutare e assimilare. Lo spazio delle possibili soluzioni deve essere limitato e consentire un risultato non parziale. Un'altra decisione chiave consiste nel decidere se l'iniziativa debba essere impostata come una collaborazione o una competizione. Nel caso della competizione bisogna definire un giusto premio che sia monetario o di altro tipo e per ciò ci si può affidare a framework già esistenti come (Singer e Mittal, 2013) [33]. I meccanismi di coordinazione sono utili per orchestrare i partecipanti e instaurare legami basati sulla responsabilità condivisa della creazione di un ecosistema. La sessione deve incoraggiare sia la cooperazione che la competizione dei partecipanti, in quanto questa strategia aumenta la qualità dei risultati sottomessi. Un buon modo per indurre il coinvolgimento e aumentare le performance durante la sessione è la condivisione da parte degli amministratori delle politiche dell'organizzazione, evidenziando gli obiettivi, le regole economiche in gioco e i principi sociali.
Le regole devono essere chiare per tutti i membri coinvolti e questi ultimi possono

anche prendere parte al cambiamento delle regole tramite discussioni e votazioni. Per garantire la trasparenza e creare fiducia sono utili obiettivi di percorso, con visibilità degli input e l'elaborazione dei dati ottenuti. Anche feedback da professionisti ed esperti aiutano a creare un progetto sostenibile ed efficace.
Conley e Tosti-Kharas in (Conley e Tosti-Kharas, 2014) [34], con riferimento ai micro task, affermano che utenti anonimi e seppur non esperti sono tanto affidabili e accurati quanto ricercatori con buone abilità.

Le iniziative di crowdsourcing garantiscono alle organizzazioni di acquisire conoscenza sia dalle singole persone che da altre organizzazioni. Ne consegue che le operazioni di gestione del sapere sono importanti per raccogliere, organizzare e sfruttare le conoscenze acquisite.
Il processo di gestione delle idee è cruciale sia durante che dopo lo svolgimento della sessione. Buona parte dei testi della letteratura propone tool automatizzati per l'analisi e il miglioramento della qualità dei risultati. Altre metodologie riescono a migliorare il livello di innovazione delle idee proposte attraverso tecniche per l'aggregazione che le rendono ancor più creative.
L'ultimo argomento nell'area della gestione delle conoscenze è la protezione della proprietà intellettuale. È un problema critico e controverso poiché molti siti di crowdsourcing impongono regole che proteggono quasi esclusivamente i diritti dell'organizzazione che pone il problema rispetto alle persone che hanno avuto l'idea innovativa.

La gestione delle persone è un elemento fondamentale per descrivere il processo di crowdsourcing. Come già detto in precedenza il crowdsourcing nasce come strumento aperto ad un pubblico non definito, l'avvento di nuovi strumenti permette ora di selezionare i partecipanti alle iniziative. Le caratteristiche dei partecipanti possono includere le loro abilità e come hanno performato in eventi precedenti. È riconosciuto come il successo in esperienze passate sia una predizione sul successo futuro in contesti simili. Bisogna però evidenziare come il precedente successo sia negativamente correlato al numero di idee diverse che potranno essere proposte.
La scelta del giusto numero di partecipanti va fatta ricordando che un numero troppo alto di partecipanti riduce la qualità dei risultati a causa di troppa competizione, però è anche vero che contrasti gli impatti negativi sulla qualità, specialmente in contesti di incertezza, grazie all'effetto dei "percorsi paralleli" e anche perché vengono fornite più soluzioni diverse.
Anche Ghezzi come Amabile evidenzia come sia fondamentale stimolare la motivazione delle persone coinvolte tramite motivatori estrinseci e intrinseci.

Output

L'output di un'iniziativa di crowdsourcing può essere un'idea innovativa o il completamento di un task. Le idee generate dai partecipanti, spesso inesperti, sono

spesso non implementabili in un contesto reale. Per ovviare a ciò le idee che sono più originali di quelle di esperti vengono aggregate e manipolate per formare un output che possa essere implementato.

I benefit di chi amministra l'iniziativa si riferiscono a:

- la conoscenza che hanno acquisito o i task che sono stati completati
- le relazioni che si instaurano con potenziali nuove persone interessate all'organizzazione
- la creazione di un processo esternalizzato di "innovazione e creazione" come un'alternativa ai processi tradizionali nel dipartimento interno di ricerca e sviluppo

I benefit dei partecipanti si riferiscono a ricompense monetarie e motivazioni sociali per garantirsi apprezzamento e riconoscimento sociale.

2.3.2 Applicazione all'ingegneria del software

In (Sarı et al., 2019) [35] si prende in considerazione la gestione e scomposizione dei task. La metodologia per la scomposizione dei task e variano in base a tre fattori: affidabilità, caratteristiche, tipo di processo che coinvolge.
Sull'affidabilità del task vi è la metodologia a scomposizione verticale per subtask dipendenti in cui l'output di un utente viene usato come punto di partenza per un altro utente. Per task indipendenti invece si usa la scomposizione orizzontale in cui i task vengono risolti contemporaneamente.
La seconda categoria comprende approcci per la decomposizione che riguardano le caratteristiche dei task che possono essere valutate in base a dimensione, creatività, qualità, contesto e requisiti di testing.
L'ultima categoria viene incluso il contesto lavorativo nel quale devono essere risolti i micro-task. I task potrebbero essere generati dinamicamente in risposta ai cambiamenti di stato degli artefatti. In un processo iterativo i micro-task vengono aggregati in unità sulle quali possono lavorare più partecipanti. I lavoratori possono fondare delle unità di lavoro con esperienza in un certo settore così da dividere anche i task in argomenti abbinabili ai team.

Differentemente dai modelli di sviluppo tradizionali, ad esempio il modello a cascata, il design, la scrittura del codice e i processi di distribuzione vengono fatti in parallelo alla ricezione di nuovi risultati del processo di crowdsourcing. La soluzione finale è costruita in modo iterativo durante tutto il ciclo di sviluppo del software. Questo nuovo modello di sviluppo è simile al modello agile per la sua natura iterativa, però in agile non ci sono più fasi di design, scrittura del codice e testing fatte in parallelo per molteplici soluzioni. Lo studio (Valentine et al., 2017) [36] propone una piattaforma collaborativa che supporta una coordinazione emergente

e adattiva di task più complessi. Gli autori sostengono che le piattaforme esistenti hanno utilizzato modelli di sviluppo che funzionano bene per micro-task e task modulari. Per gestire task più grandi e gruppi di persone con ruoli e responsabilità in costante cambiamento è necessaria l'introduzione di un nuovo modello. Il modello introdotto in questo studio è una fusione del modello a cascata e quello agile. È simile ad agile per la divisione dei task in micro-task che vengono assegnati agli sviluppatori (i partecipanti) sulla base delle loro conoscenze ed esperienza, quindi non ci sono ruoli predefiniti ed è sempre possibile sostituire uno sviluppatore con uno nuovo. La struttura gerarchica, d'altra parte, è simile al modello a cascata perché tutto il lavoro consegnato dalla folla deve essere approvato e portato nel ramo principale dagli amministratori del progetto. Gli amministratori hanno il compito di monitorare l'integrazione di micro-task in quella che risulterà essere la soluzione al task di partenza.

2.3.3 Applicazione al testing

Il testing con l'ausilio del crowdsourcing ha il vantaggio di poter usare anche comuni utenti senza esperienza specifica per eseguire i task di testing.
Nella ricerca (Mao et al., 2017) [37] vengono esposti alcuni esempi in cui è stato applicato il crowdsourcing. Le attività comprendono testing delle funzionalità, test delle performance, GUI testing, generazione automatica di casi di test e il problema dell'oracolo. Il testing funzionale è un lavoro che occupa una buona porzione del tempo dedicato al testing nel ciclo di sviluppo del software e può risultare molto costoso. Questa tipologia di testing, con l'ausilio del crowdsourcing, ha dimostrato la capacità di rivelare possibili problemi tanto quanto l'impiego di esperti. I vantaggi nell'uso del crowdsourcing si riflettono in ambito monetario, nella velocità di consegna del prodotto e all'accessibilità a questa pratica.
Il test delle performance può essere difficile da eseguire a causa dei vari scenari di esecuzione e le attitudini degli utenti. Nel caso del GUI testing la generazione di test automatizzati risulta difficile mentre il testing manuale è troppo dispendioso. Il crowdsourcing con la scomposizione in micro-task riesce ad ovviare a questi problemi riuscendo ad estendere la platea per effettuare i test e aumentare la coverage.
La generazione automatizzata di test è stata affrontata da molti studi ma la coverage ottenuta non sempre è risultata ottimale a causa di alcuni task che sono difficili da far eseguire ad una macchina ma che risultano essere semplici per una persona. I risultati di due progetti open source hanno mostrato un aumento della coverage del 7.0% e 5.8% rispetto ai metodi allo stato dell'arte.
Come ultimo ambito nel contesto del software testing si ha il problema dell'oracolo, ovvero la determinazione dell'output di un programma dato un certo input. La creazione di oracoli, essendo basati su input umani, è difficile da automatizzare.

Nello studio (Pastore et al., 2013) [38] è stato investigato l'argomento del crowdsourcing per mitigare il problema degli oracoli. I partecipanti all'iniziativa di crowdsourcing sono stati divisi in due gruppi: lavoratori qualificati (con esperienza nella programmazione) e lavoratori non qualificati scelti tramite la piattaforma Amazon Mechanical Turk. Ai partecipanti è stato chiesto di giudicare la correttezza delle asserzioni generate automaticamente da un tool e in caso di falsità, di correggerle. Il risultato di questa iniziativa ha indicato che il problema dell'oracolo può essere mitigato attraverso il crowdsourcing, ma viene comunque richiesta una buona esperienza nella programmazione.

GUI testing

I benefici del crowdsourcing si vedono particolarmente nell'ambito del GUI testing, in cui si rendono necessari la scalabilità per testare in modo completo ed efficace anche grandi sistemi e la capacità di usufruire di doti che solo le persone possiedono.

Nel lavoro di Wang et al. (Wang et al., 2019) [39] viene evidenziato un problema riguardante il crowdtesting che impatta notevolmente sull'efficienza del metodo: la presenza di report duplicati tra quelli consegnati dai vari utenti. Secondo gli autori circa l'82% dei report che vengono ricevuti sono duplicati e in termini di tempo la ricerca di duplicati su un insieme di 500 report richiede una giornata lavorativa di un tester.
Le motivazioni dietro a questo lavoro sono riposte in due scoperte:

- Le descrizioni testuali dei report possono essere mal interpretate. Con l'aiuto di informazioni visuali (screenshot) relative al contesto, i report duplicati possono essere rilevati più facilmente
- Gli screenshot forniti con i report dimostrano le informazioni del contesto. In uno specifico contesto, solo attraverso illustrazioni con una descrizione testuale si possono rilevare report duplicati

L'approccio proposto, denominato ScrEnshots and the TextUal descriptions (SETU), si compone di tre fasi: estrazione delle caratteristiche, calcolo delle somiglianze, e rilevamento dei duplicati.
Nella prima fase vengono estratte due caratteristiche dagli screenshot: caratteristiche strutturali e del colore. Per il testo invece vengono estratti le caratteristiche di frequenza dei termini (TF-IDF ovvero Term Frequency and Inverse Document Frequency) e la rappresentazione distribuita delle parole.
Nella seconda fase vengono calcolati dei punteggi di somiglianza tra screenshot e descrizioni di report diversi in base alle caratteristiche rilevate alla fase precedente. Viene impiegata la similarità del coseno per misurare la distanza tra le varie coppie messe a confronto.
Nell'ultima fase viene utilizzato un algoritmo di clustering gerarchico con il quale,

definendo una soglia di similarità tra screenshot, si classificano i report che la superano come di prima classe e quindi li si confronta anche per similarità della descrizione testuale. I report che non soddisfano la soglia minima vengono trattati come di seconda classe e li si classifica per la similarità combinata di screenshot e descrizione testuale.
Confrontando i risultati di questo metodo con altri ottenuti da strumenti definiti lo stato dell'arte nell'anno 2017 si ha che le performance di rilevamento di duplicati sono significativamente inferiori per tutti i metodi che si basano solo sulla classificazione tramite screenshot o solo su quella basata sulla descrizione testuale.

Una delle difficoltà del crowdtesting nel caso del GUI testing emerge quando volendo parallelizzare il lavoro di più persone su una stessa applicazione vengono a crearsi test duplicati. I test case delle interfacce grafiche consistono in una sequenza di eventi di input, che definiamo come percorsi di navigazione, e degli output, che sono gli stati della GUI. A causa della moltitudine di possibili percorsi che un utente può intraprendere, risulta difficile creare un insieme di set completo anche per gli scenari più semplici.
Nello studio (Chen et al., 2020) [40] viene proposto uno strumento per migliorare la coverage che assiste l'utente che effettua il test con informazioni sui percorsi di navigazione già intrapresi. Rispetto agli altri metodi già esistenti che si sono concentrati sulla rimozione dei duplicati, questo metodo si focalizza sulla prevenzione di duplicati, riuscendo a far risparmiare tempo che sarebbe impiegato in percorsi già intrapresi durante la sessione di testing e altro tempo che sarebbe stato utilizzato per la rimozione dei duplicati.
Nel tool realizzato, durante l'attività di testing è presente un aiuto grafico sulle parti dell'interfaccia da testare ed è possibile consultare un grafo interattivo del flusso di eventi. All'interfaccia sotto esame viene applicato un overlay che aggiunge informazioni agli elementi con cui è possibile interagire specificando quali percorsi sono già stati intrapresi e prevenendo quindi duplicati. Il grafo degli eventi viene costruito a partire dai test passati, rendendo quindi l'attività di testing fortemente cooperativa.
I risultati ottenuti da alcune sessioni di testing hanno evidenziato come questo tool elimini quasi completamente la presenza di test duplicati e aumenti la coverage ottenuta da tester indipendentemente dal loro livello di esperienza. Il limite principale di questo approccio è che funziona quasi esclusivamente con interfacce grafiche con un modello di oggetti (così che gli aiuti possano apparire sopra agli elementi) e uno spazio finito di stati (per poter rappresentare il grafo degli eventi). Le interfacce con uno spazio di stati non finito richiederebbe l'unione di più stati per rendere il grafo leggibile.

Il resto della tesi è strutturato come segue:

- **Capitolo 3**: fornisce una descrizione dettagliata riguardo alla scelta del modello degli stati e l'implementazione degli elementi di crowdsourcing

- **Capitolo 4**: descrive la sperimentazione e il processo di validazione del plugin
- **Capitolo 5**: presenta le riflessioni conclusive e suggerisce alcune possibilità di ampliamento del lavoro svolto.

Capitolo 3

Metodologia

In questo capitolo verrà presentato il tool utilizzato per effettuare le sessioni di testing e che permette di essere integrato con plugin che sviluppano altre funzionalità. Verranno poi proposte delle rappresentazioni per gli stati in cui può trovarsi l'interfaccia e una soluzione per il merge di stati in caso di una sessione di testing collaborativa. Il plugin di crowdsourcing sviluppato verrà integrato con quello già esistente sulla gamification sviluppando nuove metriche e puntando sull'aspetto collaborativo e non più competitivo della gamification.

3.1 Scout

Per applicare il crowdsourcing al GUI testing è stato selezionato il software Scout (Nass et. al, 2020)[9] che implementa una tecnica di Augmented Testing per le interfacce web. L'aggiunta di nuove funzionalità è facilmente ottenibile attraverso lo sviluppo di plugin che sono attivabili durante l'esecuzione del programma. Il codice di Scout non risulta invece modificabile ma è possibile interagire con le strutture dati del programma attraverso le funzioni di libreria che offre.

In Figura 3.1 si può vedere la struttura di Scout con il plugin di gamification introdotto da (Cacciotto, 2021) [2].

I moduli principali di Scout (non colorati) includono lo State Model che racchiude tutti gli stati del SUT attraversati durante le varie sessioni di testing di un utente. Il modello è implementato come un grafo pesato (Figura 3.2) in cui ogni nodo rappresenta uno stato dell'applicazione e gli archi sono le azioni del tester che comportano un cambiamento dello stato.
L'*Augmented GUI* è la sola ad essere visibile e con cui l'utente possa interagire. Tutti gli input sono raccolti da Scout e vengono poi trasmessi alla GUI originale dopo averli convertiti con un Application Driver nelle corrispettive azioni del SUT.

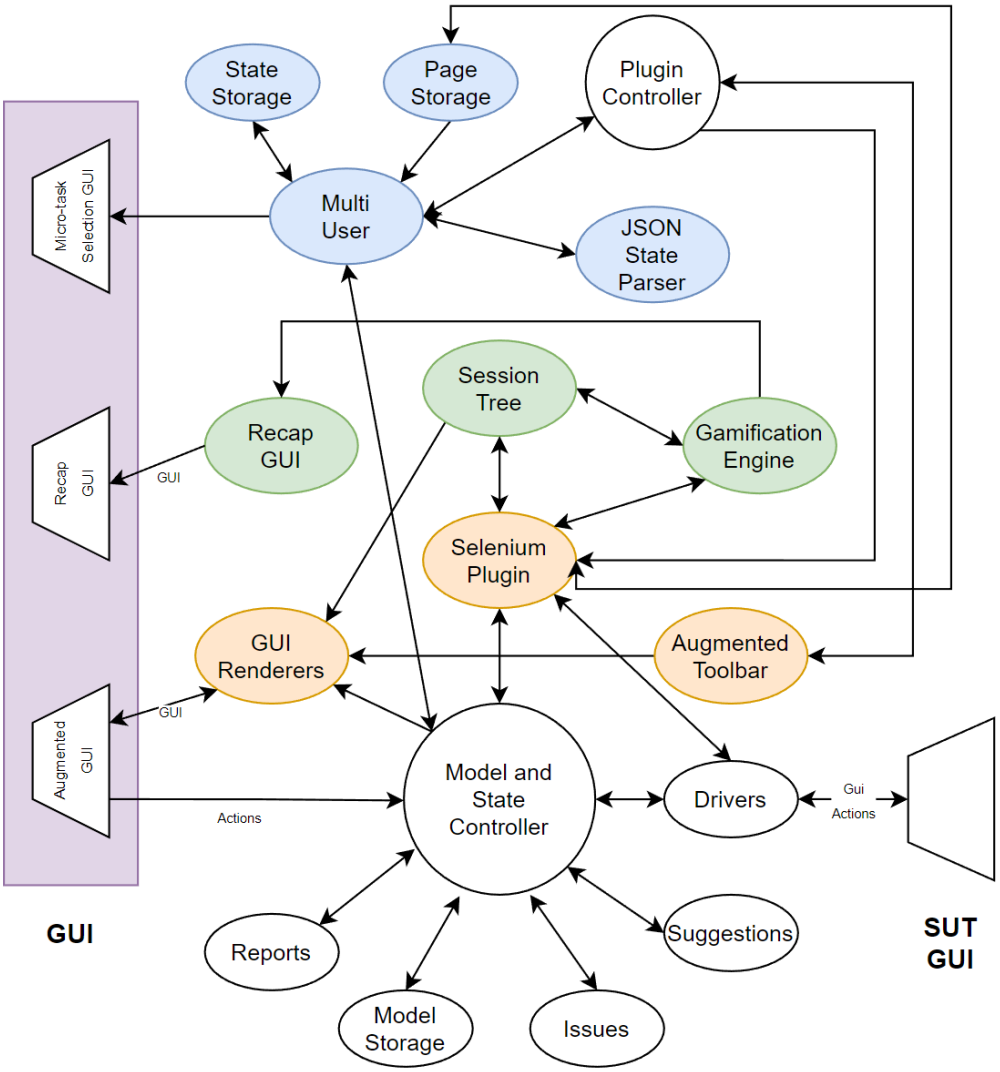


Figura 3.1: Scout con il plugin di gamification e crowdsourcing

L'interfaccia aumentata arricchisce la GUI del SUT con check, suggerimenti, issue e altre informazioni. Scout crea una variabile interna definita *widget* per ogni elemento interagibile della pagina, salvando come metadati ad esempio la posizione, le dimensioni e altre proprietà che permettono di distinguerlo da altri elementi nella pagina. L'utente vedrà dei rettangoli di differenti colori (che contengono i widget) in base all'interazione che è avvenuta nelle parti dello schermo che contengono gli elementi cliccabili: i check sono contornati in verde, i click in blu, gli issue in rosso e le espressioni che devono ancora essere valutate in giallo.

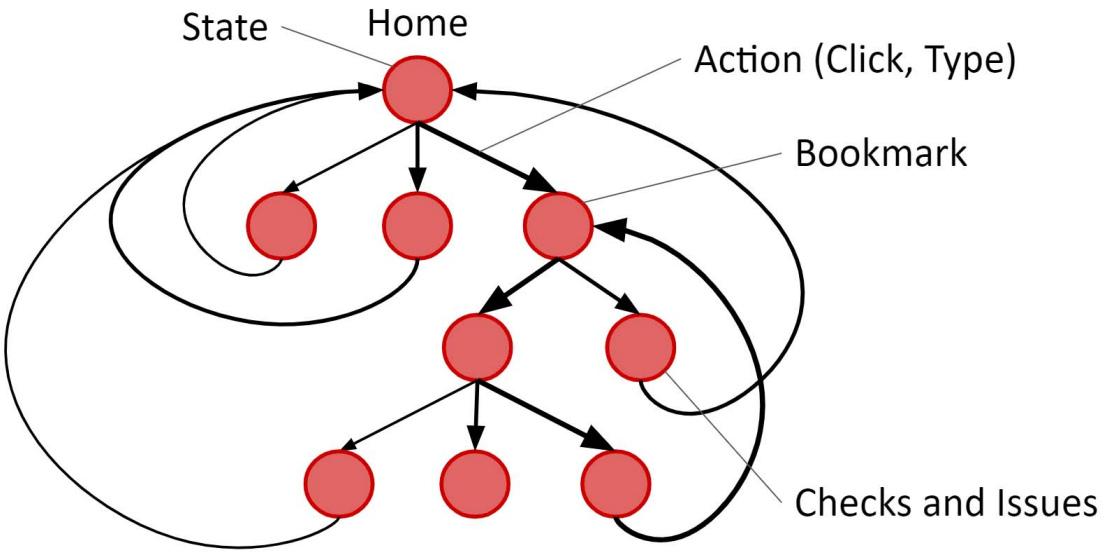


Figura 3.2: Scout State Model

L'utente può interagire anche con l'*Augmented Toolbar*, ovvero un menù che permette di gestire i plugin attivi, avviare una nuova sessione di testing o navigare il grafo degli stati per tornare ad uno salvato come segnalibro o allo stato iniziale.

Il plugin di gamification, di cui le componenti colorate in arancione intese come integrazione a quelle di Scout e in verde quelle create ex novo, introduce il *Session Tree* che registra tutte le interazioni con l'Augmented GUI e le tempistiche oltre alla coverage per ogni pagina. Il *Gamification Engine* assegna ad ogni "giocatore" i punteggi basandosi sulle interazioni descritte nel Session Tree, i quali vengono visualizzati a fine sessione nella Recap GUI.
L'albero degli stati creato dalla classe Session Tree non corrisponde perfettamente a quello generato da Scout, quindi abbiamo due rappresentazioni differenti e verranno utilizzate entrambe dal plugin di crowdsourcing con gli opportuni adattamenti. Essendo le nuove classi introdotte dal plugin di gamification strettamente legate a Selenium Plugin, sono state tutte opportunamente integrate con le modifiche per supportare anche una sessione collaborativa.
Il plugin di crowdsourcing (con componenti in azzurro) introduce due classi, Multi User e JSON State Parser, che si occupano del caricamento dello stato condiviso, la creazione dei micro-task e la risoluzione di conflitti nell'unione degli stati nel singolo stato condiviso. Gli stati di Scout e le pagine come intese dal gamification plugin vengono salvati nel State e Page storage. Ulteriori informazioni verranno aggiunte nel seguito.

3.2 Comunicazione sincrona vs asincrona

La comunicazione, ovvero lo scambio di informazioni tra persone o sistemi, è un requisito fondamentale per la collaborazione e può avvenire in modo sincrono o asincrono. La comunicazione sincrona avviene in tempo reale e richiede un coordinamento dispendioso tra le parti. La comunicazione asincrona invece non richiede che le parti siano coordinate ma che vi sia un modo per comprendere le informazioni ricevute e aggregarle a quelle che si hanno già. È stata scelta la comunicazione di tipo asincrono per il plugin collaborativo in quanto è più flessibile e semplice da realizzare in questo contesto.

Si può pensare alla cartella che contiene le informazioni condivise tra i vari tester come una repository di un VCS (Version Control System), ovvero una cartella condivisa della quale vengono mantenute le varie versioni che si vengono a creare con commit (sottomissioni) di modifiche. Un utente che avvia una sessione di testing possiede delle informazioni di stato condivise che possono essere associate ad una versione. Iniziando la sessione tutte le modifiche che apporterà allo stato saranno locali e quindi crea una versione delle informazioni condivise che si è allontanata da quelle che sono disponibili agli altri utenti, chiamata anche ramo (branch) di sviluppo. Finita la sessione di testing, per sottomettere le nuove modifiche, il sistema deve recuperare la versione più aggiornata dello stato condiviso e computare le differenze. Da queste differenze, con un'adeguata strategia risolutiva, è possibile costruire la versione finale dello stato che sarà poi possibile sottomettere e rendere disponibile agli altri utenti. Per riassumere, il plugin offrirà le funzionalità per creare, modificare o cancellare artefatti indipendentemente dalla versione corrente dello stato condiviso e di poter sincronizzare il tutto conseguentemente alla fine della sessione.

3.3 Rappresentazione degli stati

Per poter effettuare una sessione collaborativa è necessario definire il significato di stato nel plugin che verrà costruito.

Provando a formulare una definizione completa di stato di un'interfaccia grafica si può asserire a una serie di proprietà che rendono una certa schermata identica ad un'altra solo se tutte le proprietà che le caratterizzano sono equivalenti.
Di queste proprietà se ne può fare un elenco che varia in base al tipo di interfaccia, ma nel caso sotto esame ovvero le web GUI vengono proposte:

- **URL**: indirizzo specifico della pagina visualizzata nel dominio scelto
- **Xpath**: espressione che permette di identificare i singoli elementi all'interno di una pagina HTML. Attraverso quest'ultima si può verificare che vi siano gli stessi elementi e con la stessa disposizione nella struttura della pagina.

- **Variabili di stato**: possono essere associate alla singola pagina o alla sessione di navigazione in un sito. Verificarne l'uguaglianza può portare alla generazione di una quantità esponenziale di stati diversi man mano che si procede con la navigazione in profondità del dominio.

- **Elementi grafici**: l'uguaglianza estetica delle pagine web può essere utile in pochi casi (ad esempio nella contestualizzazione di bug report) in quanto non sarebbe possibile effettuare test con configurazioni diverse.

- **Stati adiacenti**: si guarda quindi al contesto con il quale si giunge alla pagina considerata, controllando quindi gli stati n-1 (i.e., lo stato che porta alla suddetta pagina) e n+1 (i.e., lo stato in cui si ricade interagendo con particolari componenti in quella pagina).

Il caso più semplice da esaminare si ha con siti web *stateless* che non portano con sé variabili globali da una pagina all'altra, quindi basta verificare che due URL uguali contengano anche gli stessi elementi con le stesse caratteristiche.
Considerando invece il caso di un utente che esegue un login e poi accede alla stessa pagina di un utente che non abbia fatto il login, non si può dire se la pagina visualizzata sia la stessa o meno, ma è sicuro che il sito web è di tipo *stateful* e conserva in qualche variabile o token le informazioni di accesso e quindi lo status di user loggato.
Anche nel caso di interfacce web che mantengono sempre lo stesso URL pur cambiando il contenuto della pagina risulta più difficile stabilire quali siano i parametri adatti a differenziare due pagine.

Nello studio di Yandrapally (Yandrapally et al., 2020) [41] vengono proposti una classificazione dei cambiamenti che possono esserci tra pagine web e un metodo per definire due pagine come "quasi-duplicati". Ciò ci consente di definire in maniera più chiara quali sono le proprietà per definire i vari stati dell'esplorazione di una pagina web.
Sono state ottenute le seguenti categorie:

- **Non correlati**. Data una differenza $\delta(e_i, e_j)$, né e_i né e_j non sono relazionati ad alcuna funzionalità offerta dall'applicazione web. Un esempio di queste differenze include cambiamenti nelle immagini di background, o widget relativi alle pubblicità.

- **Duplicati**. Data una differenza $\delta(e_i, e_j)$, e_i e e_j si riferiscono alle pagine originali p_i e p_j senza aggiungere nuove funzionalità ad alcuna pagina.

- **Nuovi**. Data una differenza $\delta(e_i, e_j)$, δ rappresenta una nuova funzionalità o un contenuto semanticamente differente.

Conseguentemente a queste classificazioni delle differenze, sono stati classificate coppie di stati da un punto di vista funzionale:

- **Clone funzionale**. Date due pagine p_1 e p_2, la coppia di stati (p_1, p_2), è un clone funzionale se non ci sono differenze semantiche, funzionali o percettibili tra loro.
- **Funzionalmente distinto**. Date due pagine p_1 e p_2, p_1 è funzionalmente distinto da p_2 se ci sono differenze semantiche o funzionali tra loro.
- **Funzionalmente quasi duplicato**. Date due pagine p_1 e p_2, la coppia di stati (p_1, p_2), p_1 è un quasi duplicato di p_2 se le differenze tra i due stati non cambiano la funzionalità esposta del sito web.
  I quasi duplicati possono essere ulteriormente divisi in 3 sottoclassi:
  - *Cosmetici*. Si ha quando i cambiamenti sono relativi all'estetica ma lasciano inalterata la funzionalità della pagina
  - *Dati dinamici*. Si ha quando entrambi gli stati della coppia di pagine sono generati dallo stesso modello e vengono popolati con dati dinamici ad esempio quelli di una query o delle funzionalità della pagina
  - *Duplicazione*. Si ha quando vi sono elementi addizionali in una pagina che sono interamente rappresentati all'interno dell'altra pagina in esame

3.3.1 Criterio utilizzato

La definizione di uno stato è necessaria poiché il meccanismo interno del nuovo plugin per il crowdsourcing ha come scopo l'unione degli stati di casi di test generati da più utenti per ottenere un solo grafo di esplorazione con un numero minimo di stati.

Nelle sezioni seguenti si spiegherà come si sia ricercato un modello degli stati semplificato per evitare il fenomeno dell'esplosione degli stati. Immaginiamo un utente che naviga su un sito di e-commerce e si ritrova nella pagina del carrello. Consideriamo come stato iniziale un utente che incontra per la prima volta quella pagina. Accettando l'acquisizione dei cookie da parte del sito cambia lo stato interno della pagina in quanto viene assegnato un id e verranno tracciate le interazioni di quel browser con quel sito. Con un'operazione di login la pagina si arricchisce di nuove informazioni quali eventuali prodotti aggiunti precedentemente al carrello, il nome, un'immagine o comunque altre informazioni dell'utente. Man mano che l'utente sceglie prodotti da aggiungere al carrello la pagina si arricchisce sempre più e alcune combinazioni di prodotti potrebbero attivare degli sconti speciali. Per una specifica pagina possono esistere un numero enorme di stati in cui si può trovare e la situazione si complica ancor più quando si considerano tutte le combinazioni che si

possono formare navigando tra le varie pagine e provenendo da stati differenti. Per questo motivo il modello utilizzato dalla versione di base di Scout non è sostenibile per la creazione di test multi-utente e si sono scelti modelli più semplificati.

Primo tentativo

Inizialmente è stato formulata la seguente definizione di stato: uno stato corrisponde ad una pagina web con un URL, che ha all'interno un determinato numero di widget e ognuno di questi widget sia distinguibile dagli altri attraverso i propri metadati, e che gli stati adiacenti siano gli stessi. Per verificare l'uguaglianza tra due stati si verifica che tutte le proprietà appena elencate combacino.
Per implementare una versione di Scout con tale rappresentazione è stata creata una nuova classe definita *HashifiedPage* che offre gli strumenti per creare un "riassunto" dei widget contenuti in una pagina e dei metodi statici per comparare le diverse rappresentazioni che potrebbero venirsi a creare. Queste pagine vengono raccolte in una HashMap che ha come chiave l'URL delle pagine e come valore una lista delle rappresentazioni delle pagine sotto forma di *HashifiedPage*. In figura 3.3 troviamo la mappa appena descritta che verrà anche salvata in memoria al termine di ogni sessione per poi essere ripristinata in una sessione successiva.

Con tale rappresentazione, se si dovessero salvare tutti i widget inclusi in una pagina con le relative proprietà, il dispendio in termini di memoria e prestazioni non sarebbe sostenibile. Per questo motivo è stato deciso di creare un "riassunto" di ogni widget che corrisponde ad un valore di tipo *Integer* tramite la funzione *hash()* che prende in input tutte le proprietà che riescono a distinguere due widget. In Figura 3.4 vengono raffigurate tutte le proprietà selezionate come essenziali.

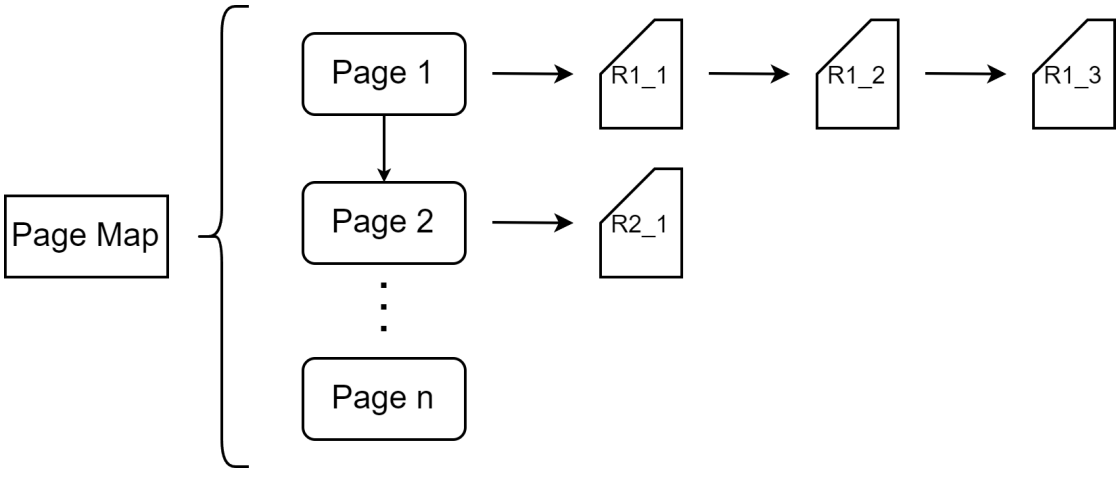


Figura 3.3: Mappa delle rappresentazioni delle pagine conosciute

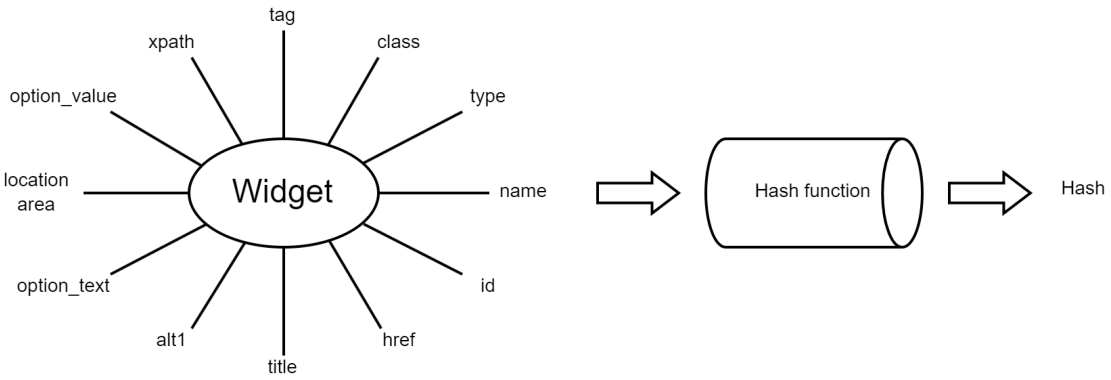


Figura 3.4: Processo conversione dei widget in hash

La classe *HashifiedPage* offre anche due metodi per verificare l'uguaglianza di due rappresentazioni di uno stato: *equals()* e *equalsPercentage()*. Entrambi sono volti a verificare l'uguaglianza di due istanze della classe *HashifiedPage*, ma il secondo metodo permette di ottenere il dato sulla percentuale di somiglianza che viene calcolato verificando quale percentuale dei widget della pagina che si sta analizzando corrispondano a quelli della pagina fornita come argomento.

Quando un utente clicca su un link, la pagina a cui si è indirizzati viene considerata come nuova e vengono istanziati anche tutti i widget della stessa. In seguito si crea il riassunto della pagina con il costruttore di *HashifiedPage* e si ricerca un riscontro nella lista delle rappresentazioni delle pagine per quell'URL. Se vi è un match viene caricato nuovamente lo stato già visitato. La percentuale di match può essere aggiustata per rendere la creazione di nuovi stati più o meno frequente con un guadagno in performance per percentuali inferiori al 100%.
Questa definizione ha portato con sé alcuni problemi non risolvibili anche se risulta semplificata rispetto alla definizione in cui due stati sono coincidenti solo se sono stati eseguiti gli stessi passi per arrivarci e tutto il contenuto della pagine sia coincidente. Per come è strutturato Scout, la fase elaborativa più complessa è quella del caricamento di una nuova pagina e la creazione dei widget contenuti in essa. Tutte le operazioni di inizializzazione possono richiedere anche alcuni secondi su un elaboratore di ultima generazione e questo tempo viene raddoppiato con l'aggiunta del nuovo algoritmo per la ricerca di uno stato già visitato, rendendo Scout quasi inutilizzabile in un contesto distribuito in cui tante persone devono eseguire le sessioni di testing sul proprio hardware.

Secondo tentativo

Nel plugin viene utilizzata una versione ridotta del modello degli stati: due stati vengono considerati coincidenti se combacia l'URL. È una semplificazione molto

forte, adottata anche nel plugin per la gamification, e che si può considerare come veritiera solo nelle pagine web statiche in cui se viene fatta una richiesta per uno stesso indirizzo si avrà sempre la stessa risposta. Tale semplificazione si rende necessaria per riuscire a costruire uno strumento che funzioni anche con poche risorse disponibili, in termini di hardware.
Il modello condiviso degli stati contiene solo i widget/percorsi effettivamente cliccati/intrapresi dagli utenti che partecipano alla definizione di test case per un particolare sito. I widget contengono informazioni sull'autore che ne ha fatto la prima scoperta, chi ne ha richiesto la cancellazione e chi ha apportato modifiche oltre ad alcuni metadati per descrivere cosa contengano.

3.3.2 Formato di salvataggio degli stati

Il formato in cui la libreria base di Scout salva i dati di esplorazione non è interpretabile, ad applicazione chiusa, da chi sviluppa i plugin.
Il modello della sessione di testing collaborativa, dovendo persistere alla chiusura dell'applicazione e in futuri lavori alla trasmissione su dispositivi collegati remotamente, deve essere salvato in un file per il quale è stato scelto un formato leggibile dall'uomo e facilmente interpretabile e manipolabile quale il JSON (JavaScript Object Notation). Le diminuzione delle prestazioni in un contesto di GUI testing sono trascurabili rispetto ai meccanismi per creare l'interfaccia aumentata, ma offre il vantaggio, nel contesto di una code review, di poter rilevare piccole modifiche a proprietà dei widget senza ispezionare il SUT con strumenti di testing basati sul modello.
Di seguito lo schema JSON che definisce le interazioni, i widget, gli utenti e l'albero di esplorazione generato da Scout:

```
1  {
2      "$schema": "http://json-schema.org/draft-07/schema",
3      "$id": "session-schema.json",
4
5      "definitions": {
6          "path": {
7              "type": "object",
8              "required": [],
9              "properties": {
10                 "product-version": { "type": "string" },
11                 "tester": { "type": "string" },
12                 "created-at": { "type": "string" },
13                 "id": { "type": "string" },
14                 "widgets": {
15                     "type": "array",
```

```
16                      "items": { "type": "string" }
17                  },
18                  "session-duration": { "type": "number" },
19                  "session-id": { "type": [ "string", "null" ]
    ↪  }
20              }
21          },
22          "widget": {
23              "type":"object",
24              "required": [],
25              "properties": {
26                  "visibility": { "type": "string" },
27                  "reported-text": { "type": [ "string",
    ↪ "null" ] },
28                  "weight": { "type": "number" },
29                  "meta-data": { "$ref":
    ↪ "#/definitions/meta-data" },
30                  "reported-by": { "type": [ "string", "null"
    ↪ ] },
31                  "type": { "type": "string" },
32                  "created-date-ms": {"type": "number"},
33                  "created-by-plugin": { "type": [ "string",
    ↪ "null" ] },
34                  "subtype": { "type": "string" },
35                  "comment": { "type": [ "string", "null" ] },
36                  "location": {
37                      "type": "object",
38                      "required": [],
39                      "properties": {
40                          "x": { "type": "string" },
41                          "width": { "type": "string" },
42                          "y": { "type": "string" },
43                          "height": { "type": "string" }
44                      }
45                  },
46                  "id": { "type": "string" },
47                  "text": { "type": [ "string", "null" ] },
48                  "created-by": { "type": [ "string", "null" ]
    ↪  },
49                  "reported-data": { "type": "string" },
50                  "status": { "type": "string" }
51              }
52          },
53          "meta-data": {
54              "type": "object",
```

```
55              "required": [],
56              "properties": {
57                  "matching_widget": { "type": "string" },
58                  "match_time": { "type": "string" },
59                  "match_score": { "type": "string" },
60                  "title": { "type": "string" },
61                  "match_percent": { "type": "string" },
62                  "contains": { "type": "string" },
63                  "xpath": { "type": "string" },
64                  "multi-user-merge-deleted-at":  { "type":
    ↪ "string" },
65                  "href":  { "type": "string" },
66                  "children": { "type": "string" },
67                  "execute_time": { "type": "string" },
68                  "id": { "type": "string" },
69                  "text": { "type": "string" },
70                  "tag": { "type": "string" },
71                  "class": { "type": "string" }
72              }
73          },
74          "state": {
75              "type": "object",
76              "required": [],
77              "properties": {
78                  "state-id": { "type": "string" },
79                  "bookmarks": { "type": "string" },
80                  "product-version": {
81                      "type": "array",
82                      "items": { "type":"string" }
83                  },
84                  "meta-data": {
85                      "type": "object",
86                      "required": [],
87                      "properties": {
88                          "product_id": { "type": "string" },
89                          "page_content": { "type": "string" }
    ↪ ,
90                          "no_clones": { "type": "string" },
91                          "no_widgets": { "type": "string" }
92                      }
93                  },
94                  "visible-widgets": {
95                      "type": "array",
96                      "items":  {
97                          "type": "object",
```

```
                        "required": [],
                        "properties": {
                            "next-state": {
                                "oneOf": [
                                    { "type": "null" },
                                    { "$ref":
    ↪ "#/definitions/state" }
                                ]
                            },
                            "id": { "type": "string" }
                        }
                    }
                }
            }
        },
        "type": "object",
        "properties": {
            "product": {
                "type": "string"
            },
            "paths": {
                "type": "array",
                "items":  { "$ref": "#/definitions/path" }
            },
            "all-widgets": {
                "type": "array",
                "items":  { "$ref": "#/definitions/widget" }
            },
            "state": { "$ref": "#/definitions/state" },
            "issues": {
                    "type": "array",
                    "items": { "type": "string" }
            }
        }
}
```

Per quanto riguarda invece il modello degli stati utilizzato dal plugin di gamification sono invece state salvate solamente le pagine visitate (come rappresentate nell'oggetto *Page*) con il riferimento al corrispettivo stato generato da Scout.
Si vuole specificare che alcuni stati salvati nel modello precedente non verranno mai raggiunti in quanto la navigazione viene volontariamente rediretta a stati già conosciuti dal plugin di gamification. Ciò avviene quando si clicca su un widget che contiene un link ad una pagina già visitata. Per Scout quest'ultima interazione

porta ad un nuovo stato, mentre per il modello semplificato considerato si ritorna ad uno stato precedente.
Di seguito lo schema JSON che definisce le pagine visitate:

```
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "$id": "pages-schema.json",

    "definitions": {
        "point": {
            "type": "object",
            "required": [],
            "properties": {
                "x": { "type": "number" },
                "y": { "type": "number" }
            }
        },
        "highscore": {
            "type": "object",
            "required": [],
            "properties": {
                "value": { "type": "number" },
                "key": { "type": "string" }
            }
        },
        "page": {
            "type": "object",
            "required": [],
            "properties": {
                "highlightedWidgets": { "type": "number" },
                "totalWidgets": { "type": "number" },
                "highlightedWidgetsId": {
                    "type": "array",
                    "items":  { "type": "string" }
                },
                "highscore": {
                    "type": "array",
                    "items":  { "$ref":
    ↪ "#/definitions/highscore" }
                },
                "hasEasterEgg": { "type": "boolean" },
                "easterEggStartPoint": { "$ref":
    ↪ "#/definitions/point" },
                "scoutState": { "type": "string" },
```

```
                "id": { "type": "string" },
                "sonWithEasterEgg": { "type": "string" },
                "isNew": { "type": "boolean" }
            }
        }
    },
    "type": "object",
        "properties": {
            "pages": {
                "type": "array",
                "items":  { "$ref": "#/definitions/page" }
            }
        }
}
```

3.3.3 Strategia di integrazione

Ogni sessione di testing di un utente comprende delle interazioni con widget che vengono memorizzate con il loro nuovo stato rispetto a quello nel grafo di esplorazione presente all'inizio della sessione di testing. Le azioni degli utenti comprendono:

- *Aggiunta* di un'interazione con un widget con conseguente modifica dello stato o transizione a stato successivo
- *Modifica* dell'interazione con un widget
- *Cancellazione* di un'interazione con cancellazione degli stati successivi

Facendo eseguire Scout contemporaneamente da utenti diversi nasce il problema dell'integrazione delle interazioni effettuate. Presupponiamo che due utenti Alberto e Bruno, chiamati per semplicità *A* e *B*, inizino la loro sessione di testing sullo stesso sito web. Avendo un unico database delle interazioni, in partenza entrambi si ritrovano con lo stesso albero degli stati, con alcune interazioni già eseguite da altri utenti. Si veda la Figura 3.5 per comprendere l'albero degli stati attuale e lo scenario presentato. A fine sessione *A* e *B* hanno eseguito un'interazione su alcuni widget, chiamati per semplicità con la lettera W seguita da un numero identificativo. *A* marca l'interazione con *W1* come eliminata e aggiunge un'interazione con *W2*. *B* aggiunge un'interazione con *W3* e *W1* che lo porta ad un altro stato. *B* nello stato derivato dall'interazione con *W1* aggiunge un'interazione con *W4*.
Nell'unione dei percorsi affrontati dai due utenti vi sono alcuni problemi facilmente risolvibili quali l'aggiunta di nuovi percorsi, mentre quello della cancellazione di un percorso che però viene esteso da un altro utente richiede una strategia ben

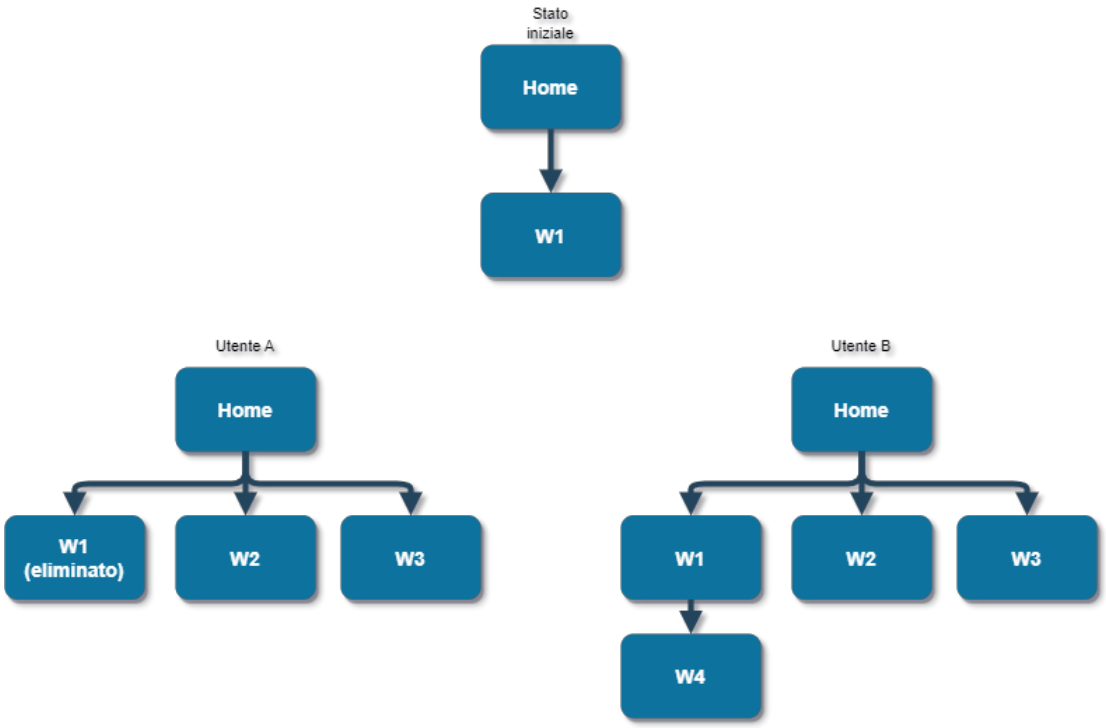


Figura 3.5: Scenario di sessione collaborativa

definita. Le varie casistiche con la corrispettiva azione risolutiva sono elencate nella Tabella 3.1 che è stata proposta da Alégroth e Bauer [3] e considerata applicabile anche in questo contesto.

Con la strategia risolutiva adottata si può ora completare l'esempio dei due utenti proposto precedentemente. In Figura 3.6 viene esposta la soluzione che prevede l'aggiunta dei widget W2 e W3 senza conflitti, mentre W1 e W4 vengono aggiunti ma l'intero percorso di esplorazione a partire da W1 verrà contrassegnato come eliminato.

Cambiamento A	Cambiamento B	Strategia di integrazione
Aggiunta	Aggiunta	Se entrambe le aggiunte sono date da un'interazione sullo stesso widget, allora unisco gli stati in uno solo, altrimenti li aggiungo entrambi
Aggiunta	Modifica	Accetto le modifiche ed eseguo una transizione al nuovo stato
Aggiunta	Cancellazione	Aggiungo una transizione ad un nuovo stato e marco lo stato e tutti i successivi come cancellati
Cancellazione	Modifica	Accetto le modifiche e marco lo stato e tutti i successivi come cancellati
Modifica	Modifica	Accetto le modifiche mantenendo solo le più recenti

Tabella 3.1: Mappatura dei principi adottati per la risoluzione dei conflitti nell'unione di test case

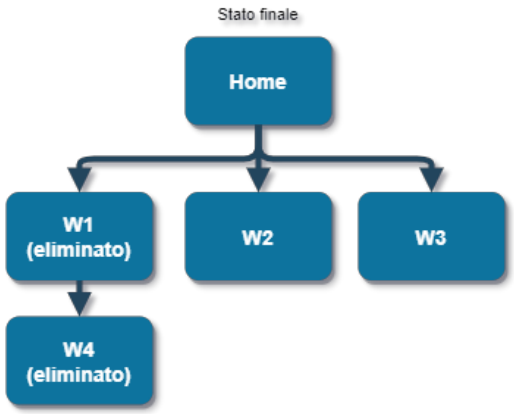


Figura 3.6: Risultato dello scenario di sessione collaborativa

3.4 Rappresentazione della sessione

Per avviare una sessione di testing, l'utente deve inserire il proprio ID, specificare il prodotto e la versione da testare, la pagina considerata come radice dell'albero di esplorazione (*Home Locator*) e il browser che si vuole emulare. Con l'aggiunta del plugin di crowdsourcing viene richiesto anche di scegliere un Micro Task da eseguire.

La rappresentazione della sessione è già stata implementata con il plugin di gamification e non è stata modificata se non per l'aggiunta di alcuni metodi per l'esportazione delle pagine visitate in formato JSON. In Figura 3.7 vediamo che le 4 classi sono: **Session**, **Node**, **Page** e **Timing**.

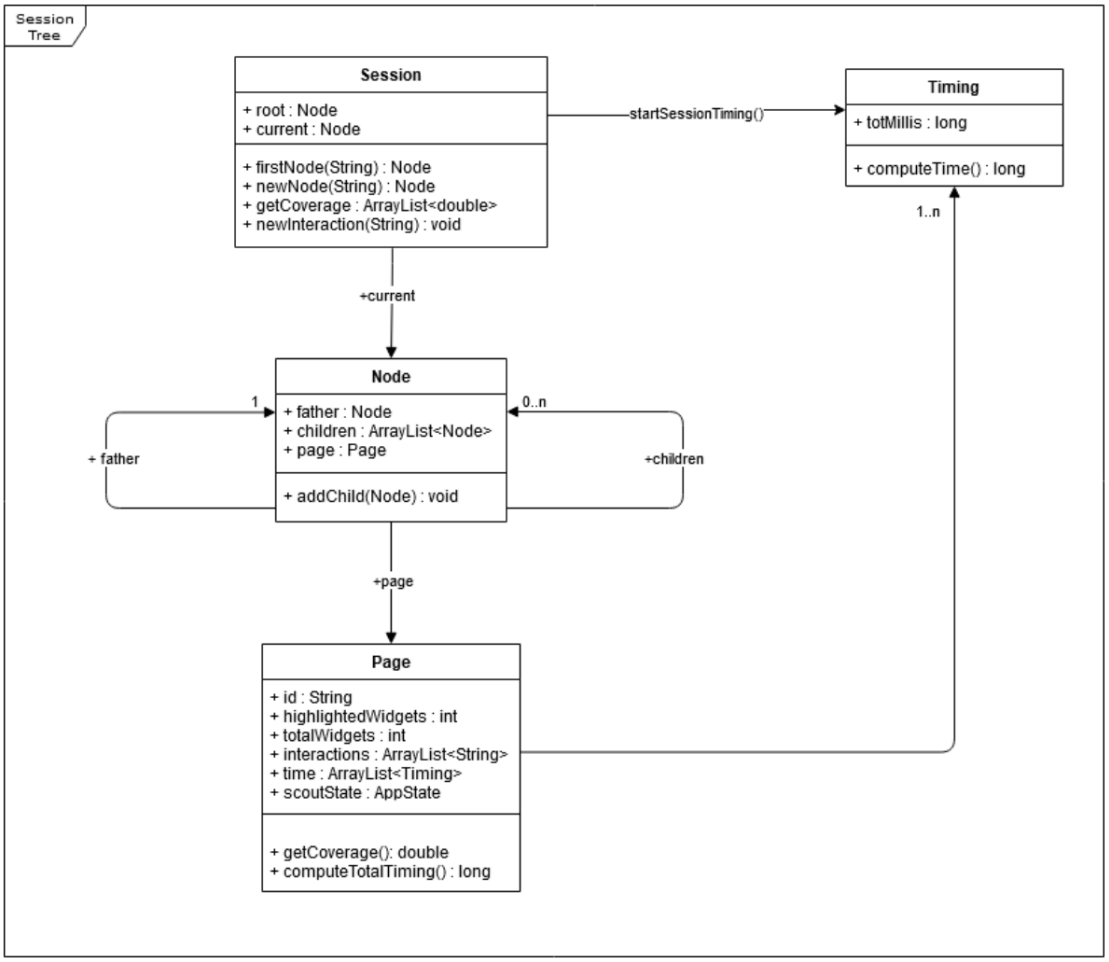


Figura 3.7: Classi Java esistenti utilizzate per rappresentare una sessione di testing

La sessione è rappresentata da un oggetto della classe *Session* che viene inizializzato all'inizio della sessione in *Selenium Plugin*. L'oggetto creato mantiene un riferimento *root* al nodo di partenza che rappresenta quindi la radice di una struttura dati ad albero non binario. Mantiene un riferimento *current* anche al nodo corrente in cui si trova il tester durante la sua sessione. L'aggiunta di nodi all'albero viene eseguita attraverso la classe Session e a partire dal nodo corrente. Altri medodi inclusi in questa classe offrono la possibilità di visitare l'albero ricorsivamente, richiedere la coverage di tutti i nodi visitati e il nuovo metodo per convertire le pagine visitate in una stringa JSON.

I nodi dell'albero sono modellati dalla classe *Node* che include un riferimento al nodo padre e a tutti i nodi figlio. Essendo ogni stato identificato univocamente grazie alla pagina visualizzata, in un oggetto Node è contenuto anche un oggetto *Page* che ha un identificatore che corrisponde all'URL della pagina, un riferimento al relativo stato nella rappresentazione di Scout, una lista di interazioni avvenute con i widget, il numero di widget visibili e totali e un meccanismo per calcolare il tempo trascorso in quella pagina attraverso un oggetto della classe *Timing*.

Le varie sessioni sono identificate dall'ID del tester e dall'istante temporale in cui si concludono. Le informazioni sulle sessioni vengono poi utilizzate dalla classe *Stats Computer* per calcolare statistiche sulle performance dei tester.

Per l'integrazione di questa visione della sessione con il plugin per il crowdsourcing si è reso necessario il salvataggio delle pagine visitate. Queste pagine, come già spiegato, sono in relazione 1 a 1 con gli stati del modello di Scout e sono necessarie per poter generare una nuova sessione che riesca poi ad essere confrontata con lo stato condiviso per integrare le differenze e assegnare un punteggio per la quantità e la qualità delle informazioni aggiunte.
Le pagine vengono salvate alla fine della sessione nel formato JSON e all'inizio di una nuova sessione di un qualsiasi utente vengono recuperate e rese disponibili in una variabile che viene passata alle funzioni di creazione della sessione e di creazione di nuovi nodi. Queste funzioni appositamente modificate, prima di generare una nuova pagina controllano se ne esista una nelle pagine recuperate dalla memoria condivisa e in caso positivo la inseriscono nel proprio albero di esplorazione.
Ogni sessione genera un albero dipendente dalle pagine dell'albero degli stati interagibile con la libreria di Scout, ma indipendente in quanto assume una propria forma che dipende solo dalle azioni del tester. Si viene quindi a generare una *foresta di alberi* per ogni micro task.

Tra i dati più rilevanti inclusi nelle 4 classi appena descritte e che vengono utilizzati nel plugin per il crowdsourcing troviamo:

- *totalWidgets*: il numero totale di widget presenti in una determinata pagina.
- *highlightedWidgets*: il numero di widget con i quali è avvenuta un'interazione in una determinata pagina.

- *id*: equivalente all'URL di una pagina.
- *scoutState*: di cui viene salvato solo l'id per procedere alla generazione dei Micro Task e ripristinare una pagina in una nuova sessione.

3.5 Micro Task

I micro task sono piccoli incarichi e vengono ampiamente utilizzati nell'ambito del crowdsourcing. Le conoscenze richieste per svolgerne uno sono minimali e solitamente si completano in un tempo breve. Gli utenti che completano i micro task ricevono un feedback sul loro lavoro e risultano più motivati per aver ottenuto dei risultati visibili in breve tempo.
Uno degli svantaggi principali nell'impiego di micro task è la mancanza di una visione completa dell'ambito in cui si lavora che può spesso aiutare a svolgere un compito in minor tempo comprendendo bene il contesto e la destinazione d'uso. Nel GUI testing un esempio di problematica potrebbe essere la presenza di un percorso dell'albero di esplorazione completamente testato e un nuovo utente a cui sia stato assegnato un task non avrebbe la memoria di quel percorso. Tutto ciò risulta in tempo utilizzato per comprendere meglio l'albero di esplorazione e non volto all'aumento della coverage in percorsi non testati. Si rende necessaria l'introduzione di alcuni strumenti, ad esempio suggerimenti sui widget cliccabili che indicano la completezza di un percorso o la visualizzazione dell'albero di esplorazione, con lo scopo di guidare un nuovo utente senza incombere in questa tipologia di problemi.

Nel caso della piattaforma di GUI testing i micro task possono essere delle missioni, ad esempio "Aggiungi un prodotto al carrello e completa il pagamento". Questi task sono volti a verificare delle specifiche funzionalità dell'interfaccia e sono difficili da creare in modo automatizzato.
Un altro tipo di micro task potrebbe essere "Parti dal nodo x e prova ad ottenere il massimo della coverage nei primi 2 livelli a partire da quel nodo". Questa tipologia di micro task è facilmente generabile da un algoritmo analizzando l'albero di esplorazione ed è quella che è stata scelta anche per semplicità, essendo volta alla dimostrazione dell'efficacia dei micro task nel GUI testing.

I task vengono generati al click del pulsante "START" che precedentemente serviva solo per iniziare una nuova sessione. L'utente deve selezionare un micro task che si presenta come un URL e l'indicazione sul numero di widget minimi con cui è possibile interagire iniziando una nuova sessione da quel nodo e considerando anche tutte le pagine già scoperte e direttamente collegate, quindi ad un livello di profondità dalla partenza.
Lo scopo di questi micro task, oltre a rappresentare un obiettivo facilmente raggiungibile per i tester, consiste nell'aumentare la coverage locale vicino al nodo di partenza. Il punteggio assegnato dal Gamification Engine dovrà includere anche

una penalizzazione nel caso in cui l'utente si "allontani" troppo dal nodo di partenza. Questo dovrebbe garantire che la coverage locale venga incrementata eliminando il micro task dalla lista iniziale.

L'elemento alla base dell'algoritmo per la ricerca dei micro task è la coverage delle pagine. Il set di pagine ottimale per la ricerca dei task è dato da tutte le pagine presenti all'interno di un certo dominio, ma avendo strumenti limitati e volendo offrire un prodotto usufruibile su larga scala con il crowdsourcing il set di pagine è ristretto a quelle già presenti nell'albero di esplorazione e quindi nello stato condiviso. La coverage di ogni pagina viene calcolata tramite il rapporto tra *highlightedWidgets* (elementi interagiti) e *totalWidgets* (elementi totali) di ogni pagina, a cui è possibile avendo recuperato lo stato condiviso delle pagine (shared-state-pages.json) e reso disponibile in *coverageMap* associando le due variabili precedentemente citate ad una chiave che rappresenta l'id dello stato di Scout. La ricerca di un candidato Micro Task si basa su una ricerca ricorsiva che parte dalla radice dell'albero (in questo caso quello generato da Scout) ed lo esplora completamente, generando micro task a partire dalle foglie. Ad ogni chiamata della funzione un nodo chiede ai propri figli un punteggio di coverage che comprende anche quello dei corrispettivi nodi figlio. Partendo dalle foglie, si ha un check che verifica che i nodi figlio incluso il punteggio di coverage del nodo stesso risultino superiori ad una certa soglia che ora chiamiamo **MIN_REQ_COV** e per la quale verrà analizzato successivamente il valore numerico. In caso positivo viene generato un micro task sul nodo risultato positivo e viene restituito al nodo chiamante un valore negativo di coverage che segnalando la scoperta di un micro task non farà includere questo nodo nel calcolo dei punteggi. In caso negativo viene restituita la somma della coverage disponibili del nodo stesso e il punteggio di coverage dei nodi figlio adeguata con un fattore **DST_FACT**.
La formula risultante è come segue:

$$P\_cov = \begin{cases} -1 & \text{se ci sono stati degli errori} \\ -2 & \text{se è stato scoperto un micro task} \\ (100 - Cov_{cur}) + \frac{\sum_{n=1}^{N} P\_cov_i}{DST\_FACT} & \text{se non ci sono stati degli errori} \end{cases} \tag{3.1}$$

dove Cov_{cur} è la coverage del nodo corrente, P_cov_i è il punteggio di coverage del figlio.
Il fattore che adegua il peso dei punteggi dei nodi figlio è stato introdotto per non far pesare troppo le coverage di nodi molto distanti dal nodo candidato e può essere modificato per regolare la dimensione di un micro task in quanto il valore del fattore è inversamente proporzionale alla dimensione del task.

Utilizzando la coverage come valore per la determinazione dei micro task potremmo avere task di dimensioni differenti rispetto ad altri in quanto nodi con la stessa coverage possono contenere un numero differente di widget interagibili. Per

indicare la dimensione di un task utilizziamo quindi la quantità di widget cliccabili e non il valore di quanto sia possibile incrementare la coverage a partire da un certo nodo. Va specificato che la quantità di widget viene intesa come quantità disponibile solo fino al livello dal quale viene scoperto un altro micro task, quindi si tratta di un dato puramente indicativo.
Come valori standard vengono proposti:

- MIN_REQ_COV = 90
- DST_FACT = 2

Con un valore MIN_REQ_COV pari a 90 viene offerta la possibilità di considerare un nodo come partenza di un micro task se la sua coverage è inferiore al 10%. Ciò porta alla generazione di molti task e quindi a test case ridondanti, ma aumenta notevolmente la possibilità di poter suddividere il lavoro a più persone, ognuna lo scopo di migliorare la coverage locale al nodo di partenza.
Con un valore pari a 2 per DST_FACT si vuole dimezzare il peso dei nodi figlio ad ogni livello di distanza dal nodo in cui si effettua il calcolo del punteggio e la verifica delle condizioni per generare un nuovo micro task. La verifica della soglia MIN_REQ_COV avviene prima di dimezzare il punteggio dei figli, quindi l'effetto del dimezzamento del punteggio dei nodi figlio avrà effetto solo sul nodo padre di quello in cui avviene la verifica. Per concludere con un esempio, consideriamo ora un albero a 4 livelli denominati come *L1, L2, L3, L4*. I punteggi dei figli di un nodo al *L3* contribuiscono al 100% alla verifica della soglia al *L3*, ma solo al 50% per il nodo al *L2* e al 25% per il nodo al *L1*.

3.6 Gestione dei dati condivisi

Il plugin per il crowdsourcing si occupa della gestione dei dati condivisi e della coordinazione tra i plugin esistenti affinché la sessione appaia come se i dati fossero di un solo utente. In figura 3.8 sono rappresentate le nuove classi introdotte dal plugin con i loro attributi e metodi principali e sono raffigurati anche i due file che rappresentano lo stato condiviso. All'avvio di scout e al caricamento dei plugin viene eseguita una chiamata a *loadstate()* per caricare nella memoria dell'applicazione l'albero degli stati da utilizzare. Nel file *shared-state.json* troviamo, come spiegato precedentemente, l'albero condiviso degli stati di Scout che viene salvato nella variabile *sharedStateFromSessionStart*. Una volta recuperato dalla memoria ci avvaliamo dei metodi nella classe JSONStateParser per convertire lo stato e widget dal formato JSON a quello di Scout che utilizza la classe AppState. Avviene anche un accesso al file *shared-state-pages.json* per creare una mappa che associa ad ogni stato dell'albero un oggetto della classe CoverageComputer che include indicazioni sul numero di widget cliccabili e quelli con cui è già avvenuta un'interazione, oltre

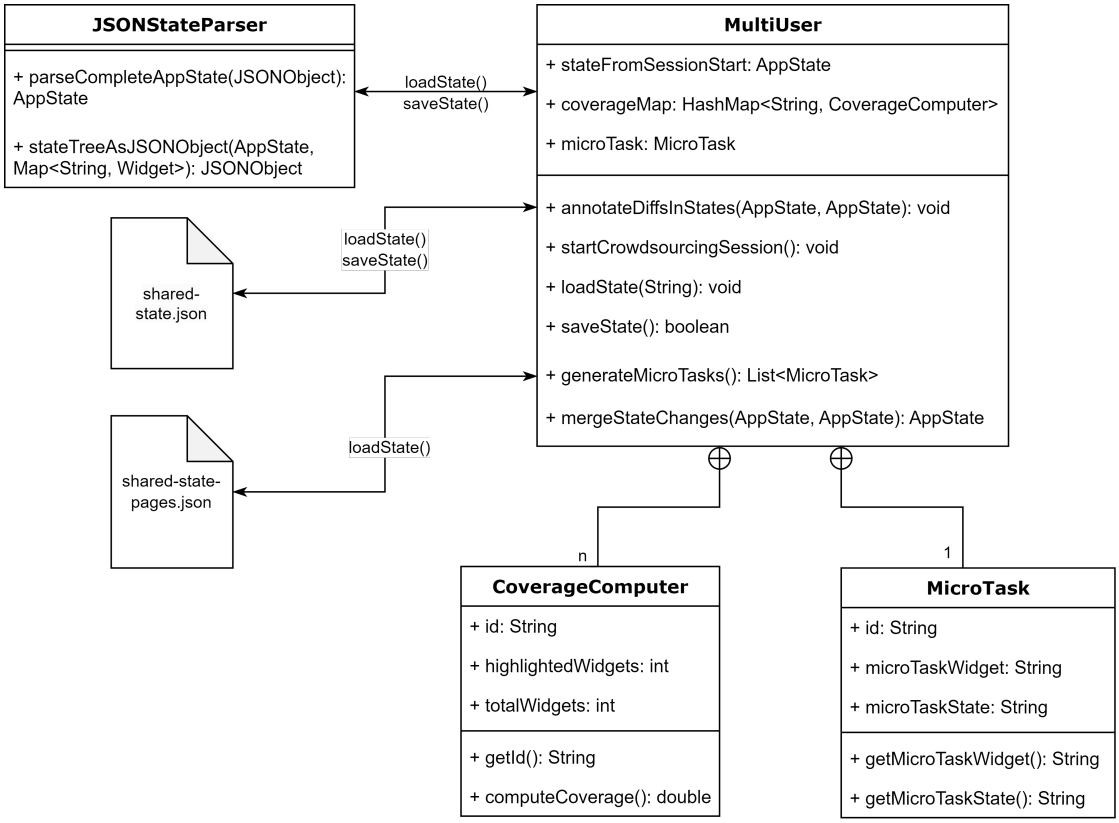


Figura 3.8: Classi che implementano le funzionalità di gestione degli stati condivisi

ad un metodo per calcolare la coverage.
La generazione dei micro task, che è stata spiegata nella sezione precedente, ci consente di disporre di un *id* di uno stato a cui "ripristinare" la sessione. Internamente il caricamento della sessione di crowdsourcing viene vista come un ripristino di una sessione precedente, ma l'utente vedrà l'avviarsi di una nuova sessione con un albero che dipende solo dalle nuove interazioni. Va specificato che l'albero di riferimento non è quello generato dal plugin *State Graph*, ma quello che verrà stampato in console a fine sessione.

Il plugin comunica con gli altri plugin attraverso un meccanismo ad azioni utilizzando la classe *Action* per generare un'azione e il metodo *performAction* di Scout che permette di inviare un'azione a tutti i plugin attivi. Attraverso questo canale di comunicazione è stato possibile rendere il plugin per il crowdsourcing responsabile dell'avvio della sessione e ordinare a *SeleniumPlugin* l'inizializzazione del browser e il caricamento della pagina richiesta dal micro task.

Al termine della sessione lo stato derivato dalla sessione attuale viene confrontato con lo stato iniziale attraverso la funzione *annotateDiffsInStates()* che aggiunge

agli stati un campo che ne delinea le differenze riscontrate. Le differenze vengono classificate in: *CREATED*, *NO_CHANGES* e *DELETED*. Al termine di questa procedura viene caricato nuovamente il file con lo stato condiviso, in quanto un altro utente potrebbe aver inserito nuovi test nel frattempo, e si procede alla risoluzione dei conflitti con lo stato locale. L'unione degli stati procede seguendo la politica di risoluzione esposta precedentemente nella tabella 3.1. Lo stato così generato viene salvato e reso disponibile ai prossimi utenti.
Il salvataggio file con le pagine condivise viene gestito da SeleniumPlugin che salva il vettore delle pagine scoperte. In quanto durante una sessione alcune pagine potrebbero essere eliminate o potrebbero diminuire i widget evidenziati, si mantiene sempre la versione di pagina che contiene il maggior numero di widget scoperti. In qualsiasi caso le pagine vengono aggiornate con i dati dello stato attuale appena vengono visitate nuovamente, recuperando dalle pagine conosciute nello stato condiviso solo l'URL e lo stato di Scout le rappresenta.

3.7 Gamification Engine

Le funzionalità relative alla gamification sono presenti nel package Gamification Engine. Le classi disponibili si occupano di raccogliere i dati della sessione, elaborare le varie metriche e renderle persistenti.
I tre metodi principali sono:

- *computeStats()*: questo metodo raccoglie tutti i dati grezzi salvati nelle strutture dati della sessione e li elabora per ottenere un oggetto della classe *Stats* contenente tutte le metriche relative alla gamification. Si occupa anche di aggiornare i valori di eventuali statistiche precedenti salvate su file.

- *computeScore()*: è il metodo principale che raggruppa tutte le metriche per generare un unico valore che è il punteggio assegnato al tester per quella sessione.

- *computeGrade()*: determina il voto del tester in base alle sue performance per quella sessione.

Il punteggio si ricava dalle metriche attraverso una formula che si compone di 3 componenti principali e 2 componenti bonus secondo la seguente formula:

$$P = a \cdot C + b \cdot EX + c \cdot EF + [d \cdot T] + [e \cdot PR] \tag{3.2}$$

Verranno ora spiegate brevemente le varie componenti così come sono state concepite per l'ambiente non collaborativo:

- **C**: rappresenta la componente di coverage e descrive il grado di copertura medio raggiunto dal tester nelle pagine visitate durante la sessione. C veniva calcolata come:

$$C = \frac{\sum_{\forall i \in P} pc_i}{|P|} \tag{3.3}$$

  dove al numeratore vi è la sommatoria delle coverage (pc_i) raggiunte nelle singole pagine nell'insieme P e al denominatore la cardinalità dell'insieme stesso.

- **EX**: rappresenta la componente esplorativa della sessione. Si basa su due fattori: il numero di pagine che il tester ha esplorato e non erano ancora state scoperte da nessun altro, e il numero di nuovi widget con cui sono state eseguite interazioni. EX_{comp}, veniva calcolata come:

$$EX = \frac{k}{b} \cdot \frac{p_{new}}{p_{tot}} + \frac{hw_{new}}{hw_{tot}} \cdot \frac{k}{k}, \quad k + h = b \tag{3.4}$$

  dove il primo rapporto rappresenta la sottocomponente relativa alle nuove pagine scoperte e il secondo quella relativa ai nuovi widget.

- **EF**: rappresenta la componente di efficienza, ovvero serve ad evitare che un utente generi interazioni multiple sullo stesso widget per aumentare il proprio punteggio. Veniva dunque calcolata come:

$$EF = \frac{w_{hl}}{w_{int}} \tag{3.5}$$

  dove w_{hl} è il numero totale di widget evidenziati della sessione, mentre w_{int} è il numero di interazioni effettuate sui widget (compresi i click effettuati su widget già evidenziati).

- **T**: rappresenta la componente temporale e viene assegnato un punteggio in base al confronto della durata della sessione con una scala in cui vengono premiate maggiormente sessioni lunghe.

- **PR**: rappresenta la componente legate alle problematiche individuate dal tester quali bug, issue e easteregg.

Per adeguare la formula di assegnazione del punteggio al crowdsourcing sono necessari alcuni adeguamenti alle metriche coinvolte nelle varie formule. Si ricorda che lo scopo dei micro task, oltre ad aumentare la parallelizzazione del lavoro, è anche quello di aumentare la coverage locale del nuovo nodo di partenza. Per stimolare l'innalzamento della coverage nel nodo di partenza e nei nodi vicini si introduce una penalità se si esplora troppo in profondità. La profondità limite al

di sopra della quale scatta la penalità viene stabilità in modalità hard-coded ad un valore di 3, che include il nodo di partenza fino ai figli dei figli. La penalità porta quindi a considerare le varie metriche in gioco già come un punteggio e non come qualcosa che abbia una correlazione 1 a 1 con gli oggetti realmente disponibili. Le funzioni per estrapolare le metriche sono state duplicate con apposite modifiche per calcolare i nuovi punteggi. Essendo queste ricorsive, si prestano bene all'introduzione della penalità in quanto per il calcolo si parte sempre dai nodi foglia dell'albero e il loro punteggio sale i livelli dell'albero venendo sommato a quello del nodo padre e così via per tutti i nodi fino a risalire alla radice. Tutto ciò si traduce in una diminuzione di una percentuale del valore del punteggio di ogni nodo figlio per il nodo in cui viene effettuato il calcolo, se quest'ultimo si trova oltre la soglia di profondità definita. Rispetto al dimezzato utilizzato nell'algoritmo per creare un micro task, la diminuzione del punteggio vuole essere più lieve per ogni livello di profondità poiché un widget o una pagina scoperti sono comunque un progresso nel testing della specifica applicazione. La penalità consigliata viene posta al 10%.

Vengono ora riformulate le varie componenti con la penalità:

- **C**: la percentuale di coverage di ogni pagina è ora intesa come aumento della percentuale di coverage rispetto a quella già presente. Una pagina che aveva già un 50% di coverage e che si ritrova ad avere un punteggio di coverage pari a 50, implica che la nuova coverage della pagina ammonti al 75% ovvero 50% iniziale più il 50% del valore iniziale. Si calcola quindi con la seguente formula:

$$pc_i = \frac{newW_{hl,i}}{w_{tot,i} - w_{ini,i}} \tag{3.6}$$

  dove $newW_{hl,i}$ sono il numero di nuovi widget evidenziati, $w_{tot,i}$ sono il numero widget totali e $w_{ini,i}$ sono il numero di widget già evidenziati all'inizio della sessione. La penalità viene applicata al punteggio.

- **EX**: non viene applicata alcuna penalità alla prima sottocomponente in quanto viene coinvolto il numero di nuove pagine scoperte. Una nuova pagina implica la possibilità di poter creare nuovi micro task e la possibilità di poter parallelizzare ulteriormente il lavoro. La seconda componente viene invece penalizzata per i livelli a cui si trova il nuovo widget individuato.

- **EF**: anche per questa componente valgono le stesse considerazioni delle precedenti. Il numero di widget evidenziati diventa il numero di nuovi widget evidenziati. Anche il numero di interazioni riguarda solo quelle eseguite in quella sessione.

In totale un tester può accumulare fino a 100 punti con le prime tre componenti (*componenti base*) e ricevere fino al 50% del punteggio base in punti bonus (ultime

due componenti) per un totale di 150 punti. Il nuovo punteggio con il plugin per il crowdsourcing è minore o uguale a quello teorizzato precedentemente, quindi non sarà possibile valutare l'utilità del nuovo plugin confrontando i punteggi con quello di gamification.
Le componenti appena descritte vengono moltiplicate per i rispettivi coefficienti a, b, c, d ed e. I valori possono essere modificati attraverso il file di configurazione ma vengono proposti i seguenti valori standard:

- $a = 0.6$
- $b = 0.3$
- $c = 0.1$
- $d = e = 0.25$

3.8 Adattamento del tool

Il punto di collegamento tra Scout e le nuove componenti introdotte è la classe *Selenium Plugin*. Questa classe che ha il ruolo di legante tra la logica degli stati di Scout e le invocazioni delle funzioni dei webdriver è stata arricchita con la logica di rappresentazione degli stati e di gamification ancor prima di questa tesi. Questo stretto legame ha portato a dover interagire e modificare parti delle funzioni di Selenium Plugin per adattarle alla presenza del plugin per il crowdsourcing. Tutte le modifiche apportate consentono l'utilizzo di Scout anche con il plugin per il crowdsourcing disattivato. In Figura 3.9 sono visibili i plugin modificati con le principali funzionalità e le interazioni tra essi.

Come già detto in precedenza, con il nuovo plugin abilitato l'avvio della sessione viene comandato dalla classe *MultiUser* che fornisce le direttive a Selenium Plugin per inizializzare il browser, l'albero degli stati, lo stato corrente e creare un nuovo oggetto della classe Session per consentire l'utilizzo della rappresentazione degli stati semplificata. Le pagine già visitate vengono caricate all'avvio della sessione per essere utilizzate per la verifica della creazione di un nuovo nodo, facendo da punto di collegamento tra gli stati di Scout e la loro nuova rappresentazione. Alla fine della sessione, le pagine vengono salvate nella memoria condivisa dopo aver integrato i cambiamenti provenienti dalla sessione o da altri utenti.

La classe preesistente per visualizzare nell'Augmented GUI l'albero degli stati, essendo legata alla rappresentazione di Scout è rimasta invariata, quindi sarà sempre visibile tutto l'albero. La navigazione dell'albero non deve però essere considerata in quanto la nuova rappresentazione porta lo State Controller ad eseguire dei salti tra un ramo e l'altro dell'albero in caso di pagine già visitate. Per rendere più agevole la navigazione viene aggiunto un segnalibro "Micro Task Home" allo stato

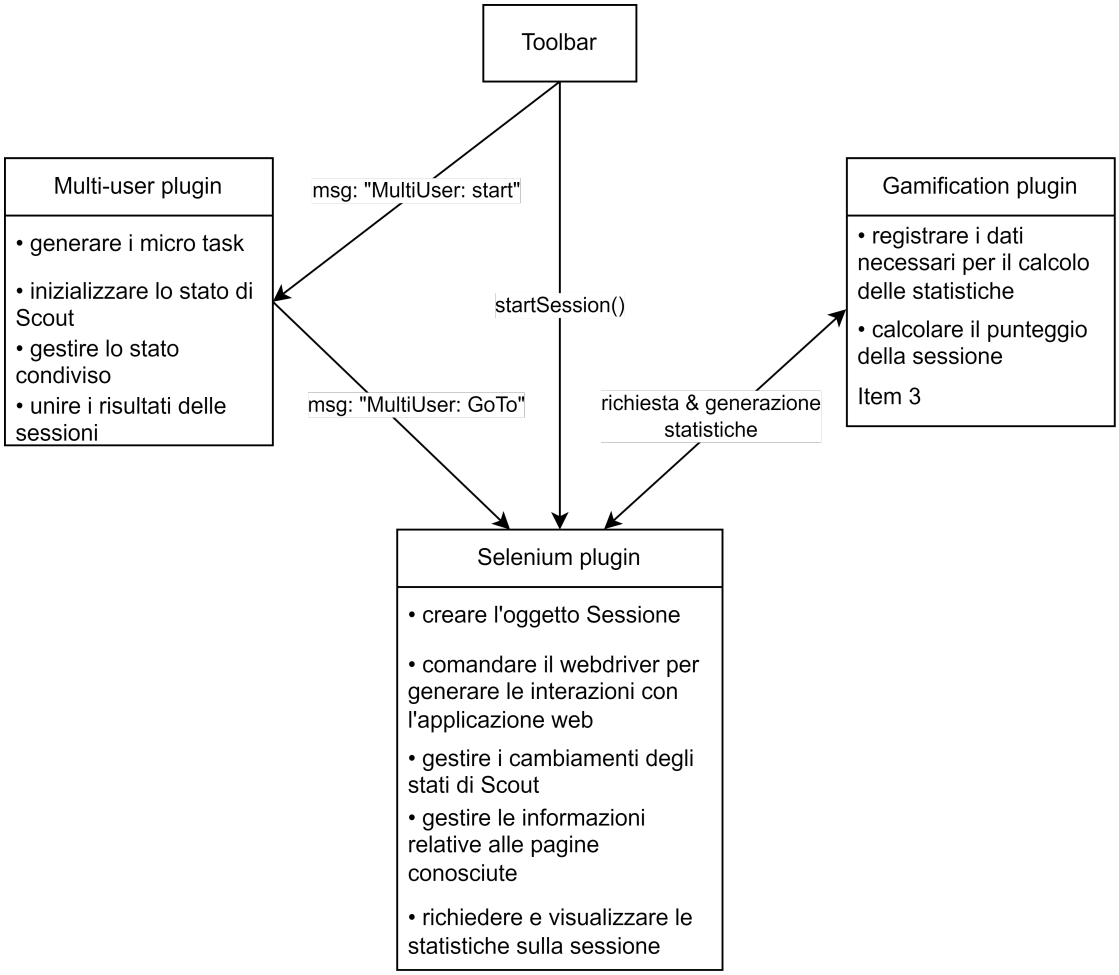


Figura 3.9: Interazioni tra i principali plugin e relative funzionalità

di partenza del micro task e un bottone "Go Micro Task Home" per tornarvi. A differenza del pulsante "Go Home" che torna alla pagina definita come *Home State* e aggiunge un widget alla pagina in cui è stato premuto, il pulsante per tornare alla pagina di partenza del micro task non crea un nuovo widget.

I widget sono arricchiti con nuovi metadati per consentire la collaborazione. I widget contrassegnati come eliminati vengono rappresentati come un widget visibile con il contorno blu, ma all'interno di esso è presente anche una cornice rossa.

3.9 Principi di gamification applicati

Per quanto riguarda il tema della gamification è stato seguito il framework *Octalysis* e sono stati individuati 2 vertici dell'ottagono in cui c'era la possibilità di espandere

gli elementi di gamification applicando tecniche di crowdsourcing. Dando la priorità agli elementi della "White Hat Gamification" abbiamo:

- **Progresso e senso di realizzazione**: l'utente che si ritrova ad eseguire test utilizzando Scout può essere motivato affrontando sfide che gli consentono di aumentare le proprie abilità. I micro task risultano essere delle sfide con lo scopo di aumentare la coverage locale del nodo di partenza. Il completamento di questi task determina la ricezione di una valutazione che indica quanto le abilità del giocatore siano sviluppate. Aggiungendo la penalità sui punti forniti dai widget e le pagine trovati oltre un certo livello di profondità nell'albero di esplorazione, si porta l'utente a massimizzare la coverage nei primi nodi. Una coverage al 100% è difficile da realizzare e implica una sessione molto approfondita che dovrebbe essere garantita dalla ricerca del progresso degli utenti.

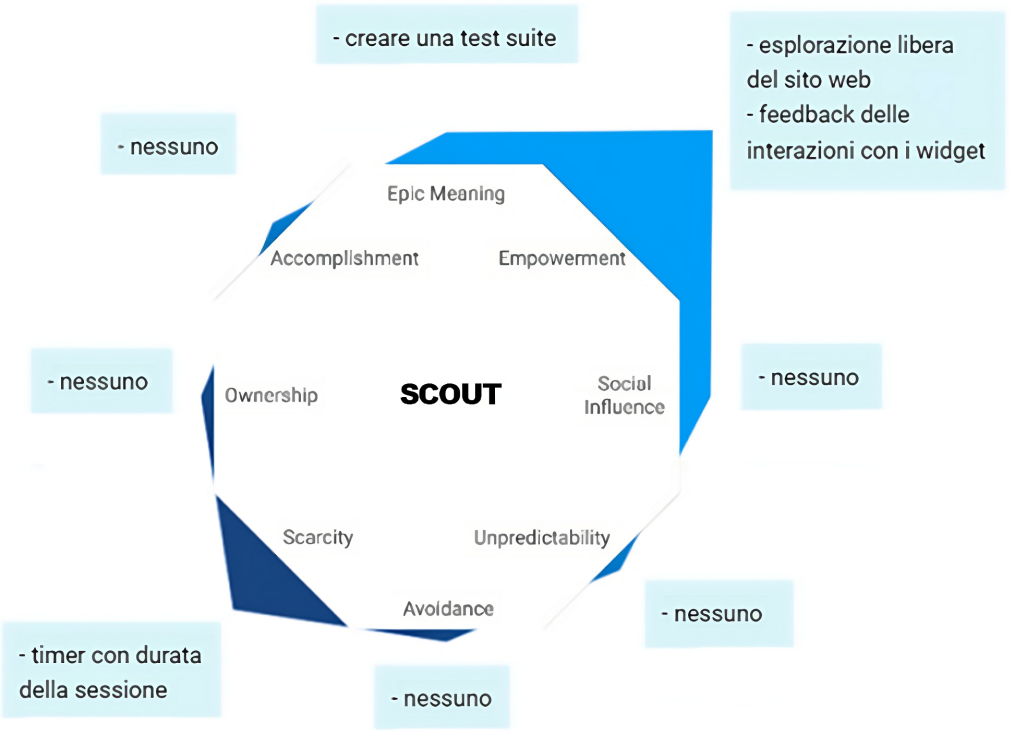


Figura 3.10: Framework Octalysis applicato a Scout

- **Influenza sociale e relazione**: è l'elemento alla base di tutto il processo di crowdsourcing. La collaborazione per creare un test case che copre tutte le interfacce del prodotto analizzato include lo svolgimento di sfide di gruppo che sono i micro task. Uno stesso task può essere svolto parallelamente da più utenti ma tutti contribuiscono allo sviluppo della test suite iniziale e si sentono coinvolti nel partecipare a qualcosa di più grande. Un buon risultato di coverage in un task implica che altri utenti non debbano rimediare andando a migliorare i test case e la scoperta di nuove pagine è molto utile alla creazione di nuovi task per altri utenti. Dalla pura competizione si passa quindi ad un mix tra competizione e collaborazione una serie di aiuti reciproci e status sociale che dovrebbero portare alla rapida creazione di nuovi micro task e al completamento dei task con un risultato ottimale.

In Figura 3.10 è rappresentato il framework Octalysis applicato a Scout in cui è stata data una misura quantitativa della presenza di elementi di gamification per ogni principio chiave. Tanto più è incisivo un certo meccanismo e tanto si espanderà

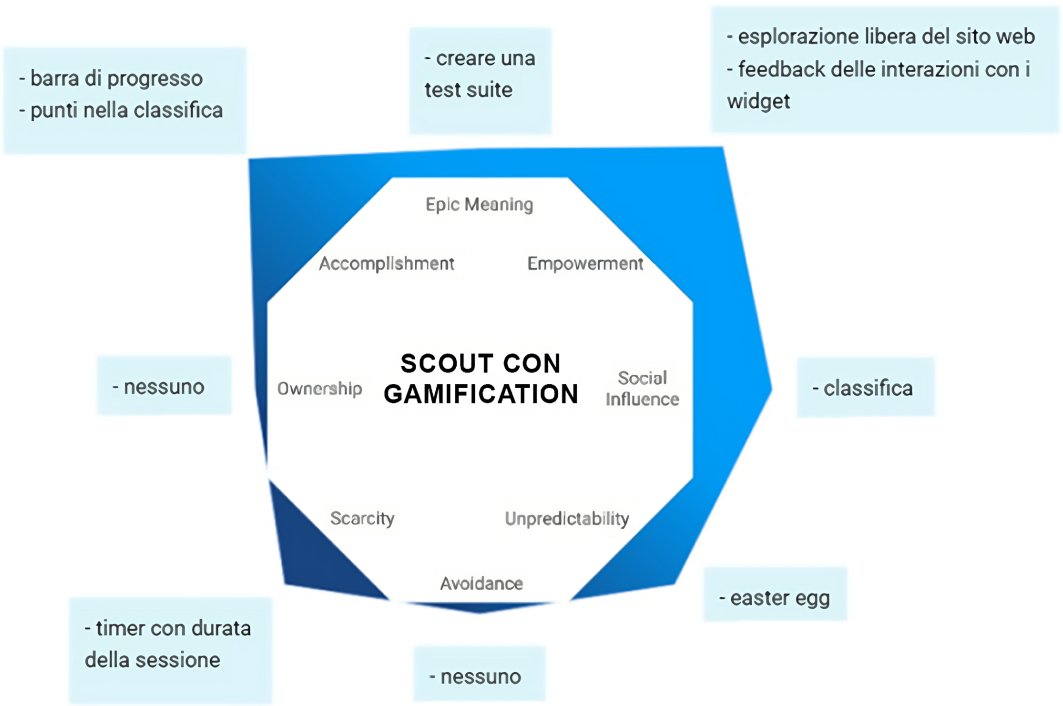


Figura 3.11: Framework Octalysis applicato a Scout con il plugin di gamification

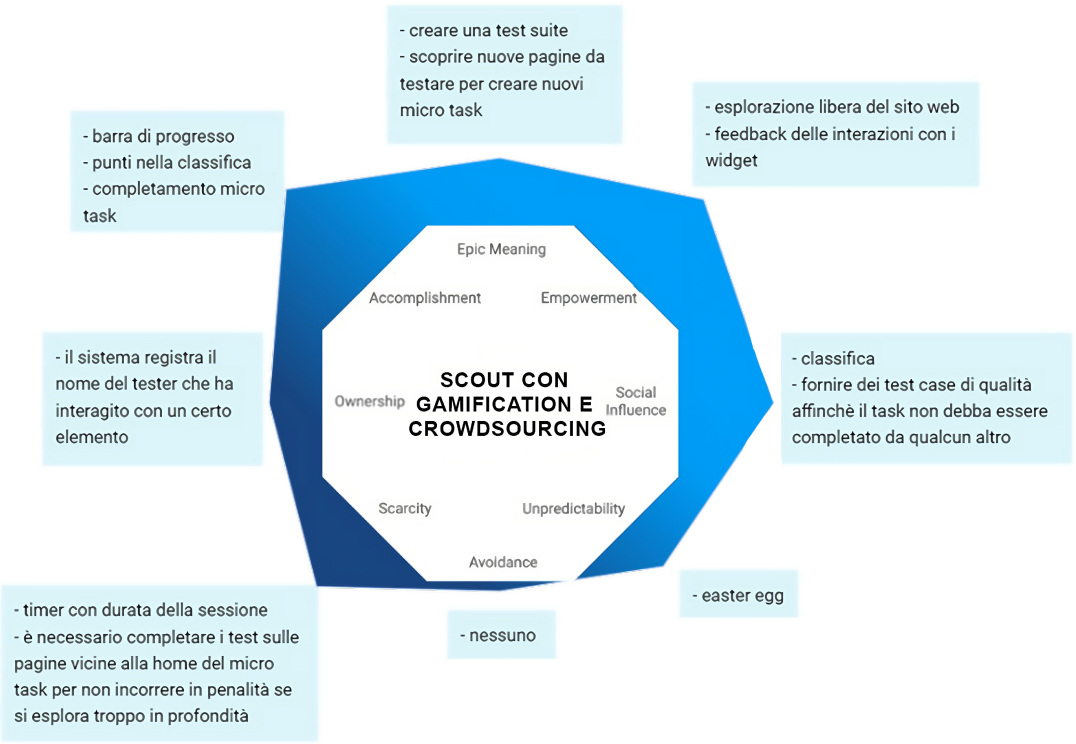


Figura 3.12: Framework Octalysis applicato a Scout con i plugin di gamification e crowdsourcing

il lato dell'ottagono al quale appartiene. Nelle Figure 3.11 e 3.12 si possono invece confrontare la versione solo gamificata e quella in cui è stato aggiunto il plugin multi-utente. Con il nuovo plugin l'ottagono risulta ben sviluppato in tutti quegli elementi appartenenti alla "White Hat Gamification" e che dovrebbero stimolare in modo positivo l'utente che farà utilizzo di questo strumento.

3.10 Elementi teorizzati

I micro task sono l'elemento chiave per eseguire sessioni multi utente che portano alla generazione di nuovi test case. L'idea teorizzata consiste nello spostamento della generazione dei task in uno strumento esterno, introducendo quindi un gestore dei micro task. Per come il plugin per il crowdsourcing sia stato implementato attualmente, i micro task vengono creati localmente all'avvio della sessione e

dipendono solamente dal particolare stato condiviso che è stato caricato prima dell'inizio della sessione. Si possono delineare due casi sfavorevoli che possono capitare con questa implementazione:

- Un'istanza di Scout genera uno specifico micro task che viene selezionato da un utente. Al termine della sessione di quell'utente il task risulta completo nel senso che non apparirà più in altre istanze di Scout poiché ha una coverage sufficientemente elevata. Se il task è stato completato solo da un utente si può considerare ciò come caso sfavorevole poiché avere due o più soluzioni per lo stesso task potrebbero incrementare la qualità del risultato sia in termini di coverage che di dati diversi utilizzati per testare i vari form.

- Per un task che viene generato da troppe istanze di Scout per utenti diversi ci si ritrova nel caso opposto rispetto al precedente. I test case forniti dalla moltitudine di utenti potrebbero sovrapporsi in gran parte e non portare ulteriori vantaggi, annullandosi a vicenda e sprecando forza lavoro.

La proposta per ovviare a questi problemi è l'introduzione di un "Micro task Manager" che sia indipendente dal funzionamento delle singole istanze di Scout ma sia raggiungibile da tutte queste. Questo nuovo applicativo deve aggiornare costantemente i micro task disponibili in base al nuovo stato condiviso e mantenere traccia di quelli precedentemente generati che sono stati assegnati agli utenti ma non ancora completati. Si introduce una soglia minima e massima di completamenti per un micro task che può essere fissa, ad esempio $S_{min} = S_{max} = 3$, oppure variabile che si adatti in base a delle metriche prefissate che indicano la completezza e la qualità delle soluzioni ricevute.

Attualmente i task vengono creati in modo automatico in base ad alcuni parametri per garantire un certo aumento di coverage il che li rende poco specifici. Il tester non si concentra sulla verifica della funzionalità di un insieme di azioni che potrebbero essere eseguite in una certa sequenza ma proverebbe solo ad aumentare la coverage della pagina. Si potrebbe introdurre il ruolo di un supervisore per i task che inserisce manualmente nuovi comportamenti da verificare ad esempio "Scegli dei prodotti da aggiungere al carrello e successivamente completa tutta la procedura di acquisto fino al pagamento" o "Utilizza la sezione di supporto per inviare un ticket per la risoluzione di problemi tecnici". Si introduce il problema della verifica del completamento dei task che dovrà essere anch'esso assegnato ad una persona dedicata.

Si prenda ora in considerazione un gruppo che deve collaborare per creare dei test per un certo prodotto. Questo tipo di collaborazione non è qualcosa di tipico del crowdsourcing, ma è più qualcosa rivolto alle funzionalità multi-utente. Per avvantaggiare il lavoro di gruppo serve l'introduzione della collaborazione in tempo reale. Se i test di tutti gli utenti che stanno svolgendo delle sessioni venissero

sincronizzati in tempo reale non sarebbe una cosa positiva in quanto un utente vedrebbe apparire molti widget sulla propria pagina e non capirebbe più cosa fare. Si propone invece di attivare la collaborazione in tempo reale solo per un particolare gruppo di persone a cui viene assegnato un micro task. La collaborazione sarebbe fruttuosa poiché può esserci una forma di coordinamento esterna che interviene e ogni utente comprenderebbe quello che sta accadendo e quale sia il suo compito.

Capitolo 4

Valutazione

4.1 Esempio di sessione

Per illustrare il funzionamento del plugin e la rispettiva rappresentazione interna della sessione verrà ora presentata un esempio di sessione di testing che un utente potrebbe svolgere su un sito web di tipo marketplace. Tale sito[1] è stato scelto per rimanere allineati agli esempi della tesi di Cacciotto [2] che ha fornito le classi relative alla rappresentazione della sessione e ai meccanismi di gamification. Il

[1] https://magi-giovanni-funghi-e-tartufi.webnode.it/

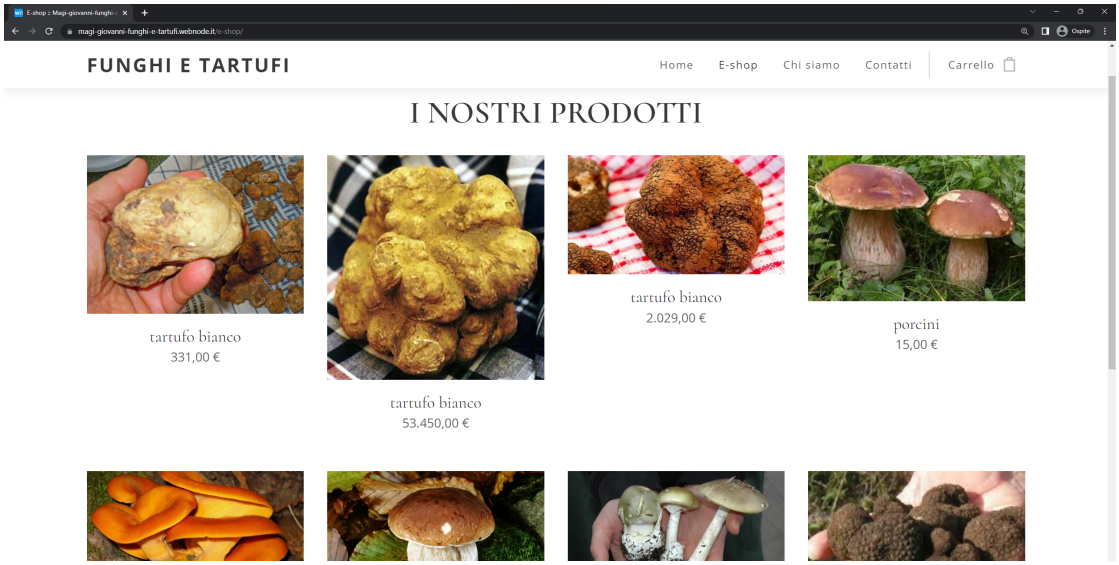


Figura 4.1: Pagina "E-shop" del sito scelto per l'esempio

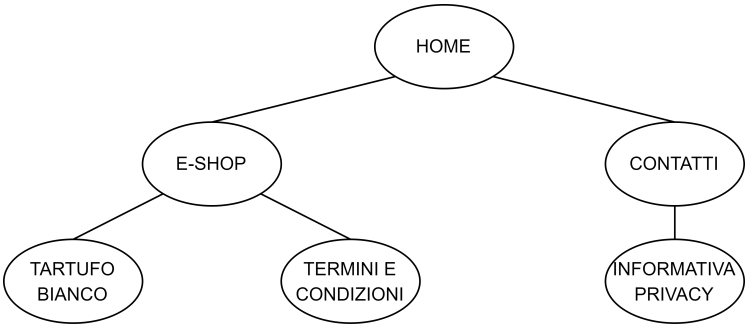


Figura 4.2: Stato condiviso presente all'avvio della sessione

sito, del quale viene rappresentata una schermata in Figura 4.1, è ottimale per questo esempio perché rispetta i requisiti di staticità in quanto ad ogni URL avremo sempre la stessa pagina senza la presenza di stati che cambino l'interfaccia, se non nel caso del carrello.

La sessione svolta avendo il plugin per il crowdsourcing abilitato, parte da uno stato condiviso non nullo per evidenziare il comportamento dell'applicativo in caso di un micro task. In figura 4.2 abbiamo rappresentato lo stato condiviso.

La sessione è stata svolta seguendo i passi elencati:

1. All'avvio della sessione è stato scelto uno specifico micro task [2].
2. Una volta caricata la pagina di partenza sono stati eseguiti alcuni *Check* su widget in quella pagina e poi è stato cliccato il widget che porta alla pagina "Chi siamo".
3. Come per lo step precedente sono avvenute delle interazioni con widget all'interno della pagina e poi ci si è spostati alla pagina dell'e-shop con il relativo widget.
4. Raggiunta la pagina dell'e-shop è stato creato un "*Bookmark*" per salvare la pagina e poi è stato scelto l'articolo "Tartufo Bianco" dal catalogo.
5. Dalla pagina attuale si decide di tornare alla precedente pagina attraverso il segnalibro appena creato.
6. Dalla pagina dell'e-shop si procede cliccando sull'icona del carrello.
7. Alla creazione della sessione viene registrato anche un segnalibro relativo alla pagina di partenza del micro task quindi dalla pagina del carrello si sceglie di tornare alla pagina di partenza attraverso l'apposito comando dalla toolbar.

[2] https://magi-giovanni-funghi-e-tartufi.webnode.it/contatti/

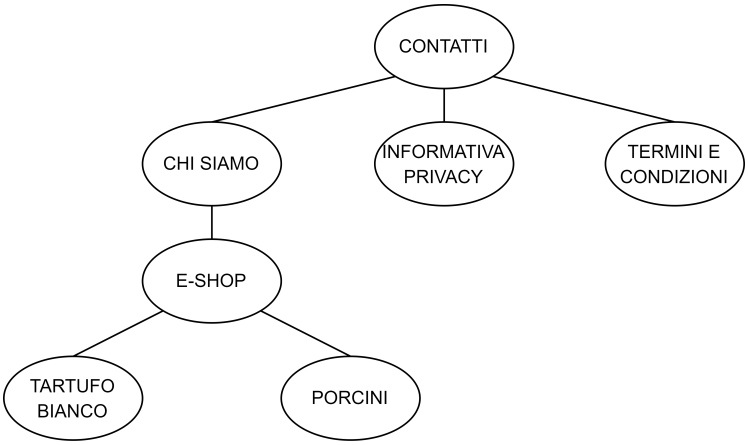


Figura 4.3: Stato relativo alla sessione appena conclusa

```
https://magi-giovanni-funghi-e-tartufi.webnode.it/contatti/
--->https://magi-giovanni-funghi-e-tartufi.webnode.it/chi-siamo/
--->--->https://magi-giovanni-funghi-e-tartufi.webnode.it/e-shop/
--->--->--->https://magi-giovanni-funghi-e-tartufi.webnode.it/p/tartufo-bianco/
--->--->--->https://magi-giovanni-funghi-e-tartufi.webnode.it/p/porcini/
--->https://magi-giovanni-funghi-e-tartufi.webnode.it/informativa-sulla-privacy/
--->https://magi-giovanni-funghi-e-tartufi.webnode.it/termini-e-condizioni/
```

Figura 4.4: Output della vista pre-order dell'albero della sessione

8. Allo stesso modo dei punti precedenti vengono esplorate altre due pagine: "Informativa sulla privacy" e "Termini e condizioni". Dopo aver raggiunto ogni pagina si torna alla pagina di partenza.

9. Infine si decide di terminare la sessione premendo il tasto Stop presente nella toolbar di Scout.

Come già descritto in precedenza, la forma dell'albero generato per ogni sessione è indipendente dallo stato condiviso, formando quindi una foresta di alberi per ogni micro task.
La sequenza di operazioni appena eseguita porta ad uno stato della sessione come raffigurato in Figura 4.3. Si noti come visite multiple su una stessa pagina non generino ulteriori nodi, ma reindirizzino lo stato di Scout allo stato visitato precedentemente. Attraverso il metodo *printTree()* fornito dalla classe *Session* è possibile ottenere una rappresentazione dell'albero in formato testuale (esempio in Figura 4.4). La visita dell'albero è effetuata con un algoritmo di visita *pre-order* in cui i livelli di profondità sono distinguibili dalla loro indentazione.

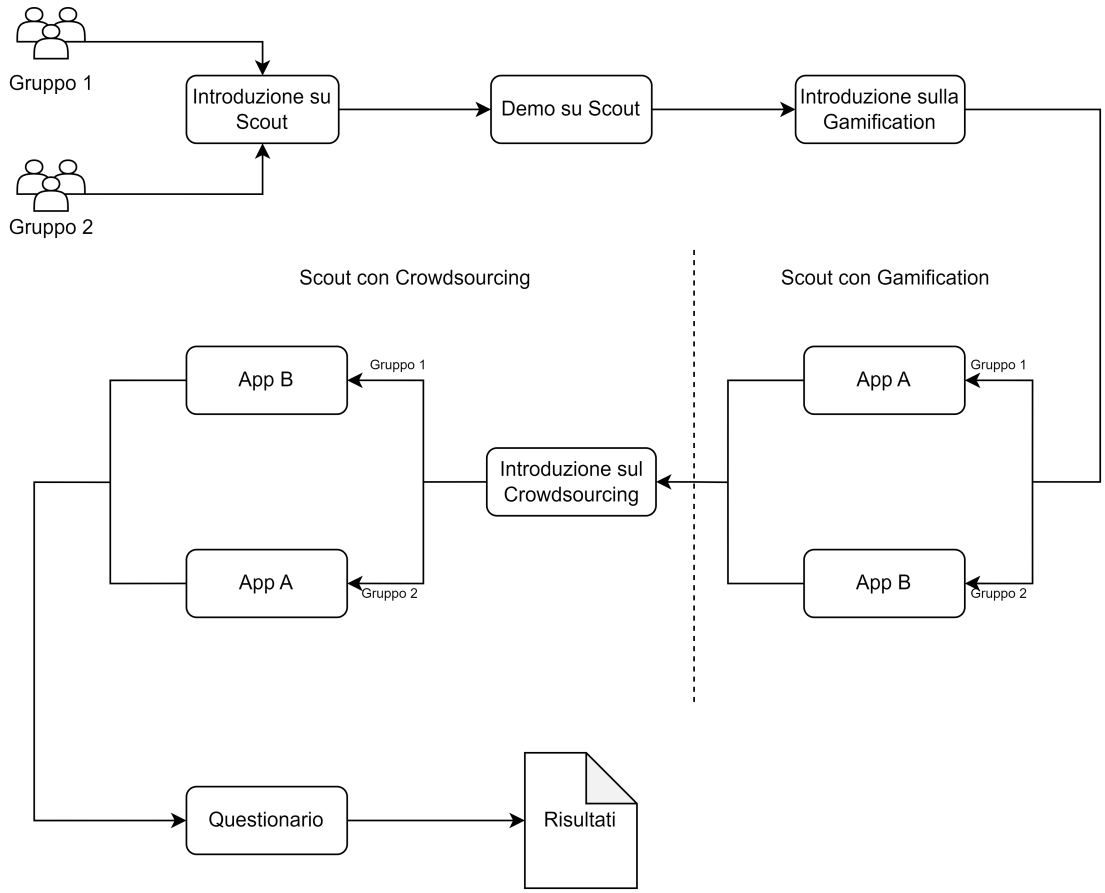


Figura 4.5: Fasi del processo di validazione

4.2 Descrizione dell'esperimento

La fase di validazione del tool è stata divisa in 3 macrofasi:

1. Presentazione di Scout e demo sull'utilizzo
2. Sperimentazione di Scout attivando prima il plugin di gamification e successivamente anche quello per il crowdsourcing
3. Raccolta delle informazioni sulle esperienze pregresse e opinioni dei partecipanti tramite un questionario

Come si può notare in Figura 4.5 il campione è stato diviso in due gruppi di eguali dimensioni in modo casuale per seguire uno schema di validazione AB-BA. L'utilizzo di questo schema fornisce ad ogni partecipante la possibilità di utilizzare il tool due volte, ognuna su una specifica applicazione: la prima volta sulla versione

gamificata di Scout e la seconda su quella multi-utente. L'esperienza in termini di conoscenza di un determinato sito web durante la prima sessione di testing può portare ad ottenere un risultato migliore durante una seconda sessione sullo stesso dominio, per questo motivo viene introdotto un altro dominio per ridurre questo fattore.
I due domini utilizzati per le sessioni di testing sono:

- Sito A: sito di tipo marketplace[3]
- Sito B: sito di un portale istituzionale[4]

La fase introduttiva prevede una rapida spiegazione di cosa sia il testing di interfacce web e come Scout sia uno strumento importante grazie all'interfaccia aumentata. Con l'ausilio di una dimostrazione pratica su un dominio ben conosciuto[5] vengono illustrate tutte le principali funzioni di Scout per eseguire correttamente una sessione di testing.

Prima di far eseguire la prima sessione di testing autonoma ai partecipanti viene affrontato il tema della gamification e tramite quali elementi ludici sia stata implementata nel primo plugin. Ogni persona di ogni gruppo ha a disposizione massimo quindici minuti per completare questa prima sessione nel dominio assegnatole.
Si passa poi ad una rapida spiegazione sui fondamenti del crowdsourcing e come questo porti alla creazione dei micro task e all'aggiunta di penalità in caso di una sessione che non pensi al lavoro che altri potrebbero svolgere successivamente. Per svolgere le sessioni di testing con il plugin multi-utente attivo, i partecipanti dei due gruppi verranno coordinati per ottenere dei risultati realistici.
Con l'introduzione del testing collaborativo si presentano molteplici possibilità di effettuare le sessioni:

- Sessioni temporalmente non coincidenti: il caso in cui si ha un risultato che comporta il massimo aumento di coverage e nessun conflitto
- Sessioni contemporanee scegliendo micro task differenti: un caso intermedio, in cui i percorsi di esplorazione dei vari utenti potrebbero coincidere in parte e quindi portare a conflitti in cui solo i test di un utente portano all'aumento della coverage
- Sessioni contemporanee scegliendo lo stesso micro task: il caso peggiore in termini di coverage poiché è molto probabile che i test dei vari utenti combacino in buona parte e riducano notevolmente i vantaggi delle sessioni multi-utente

[3] https://magi-giovanni-funghi-e-tartufi.webnode.it/

[4] https://www.polito.it/

[5] https://www.wikipedia.com/

Per riuscire a simulare un caso medio i membri del primo gruppo effettueranno delle sessioni disgiunte, mentre il secondo gruppo effettuerà delle sessioni coincidenti a partire dallo stesso micro task.

La fase finale dell'esperimento ha previsto la compilazione di un questionario su Google Form da parte di tutti i partecipanti le cui domande in Tabella 4.1. Le domande si dividono in 3 sezioni: la prima sezione con domande personali sull'anagrafica e esperienze pregresse in ambito informatico, la seconda sulle valutazioni dell'esperimento e di Scout con i vari plugin attivi, la terza su eventuali consigli, suggerimenti e problematiche riscontrate.

4.2.1 Selezione del campione

Il processo di testing di per sé risulta abbastanza intuitivo, si avvicina molto alla normale navigazione di una pagina web, con le differenze maggiori che consistono nell'interazione con l'interfaccia aumentata e nella ricerca di eventuali problematiche nel funzionamento. Uno degli obiettivi di questa tesi è stato quello di ampliare il più possibile il pubblico a cui sia destinato questo strumento e per questo tra le persone scelte per eseguire l'esperimento non sono stati scelti solo studenti di ingegneria informatica, ma anche altri giovani soggetti con l'unico requisito della conoscenza della navigazione Web.

Il campione selezionato è stato di sei persone, due con età compresa nella fascia 18-24 anni e quattro con età maggiore o uguale ai 25 anni. Nelle Figure 4.6 e 4.7 sono stati raffigurati i risultati del questionario delle domande da **D1.2** a **D1.7**. Si può notare come poco più della metà del campione ha avuto esperienze pregresse con linguaggi di programmazione e testing. Per quanto riguarda il tema della gamification solo 2 persone ne avevano già sentito parlare e ne conoscevano gli scopi, mentre il crowdsourcing è stato un tema totalmente nuovo.

4.2.2 Ambiente di svolgimento

Scout è un software che può risultare molto impegnativo da eseguire su computer di media potenza a causa dell'interfaccia aumentata e di tutti i meccanismi di gamification e crowdsourcing che sono stati aggiunti. Per tale motivo è stato deciso di eseguire l'esperimento sul computer in cui è stato sviluppato il nuovo plugin e i partecipanti vi avrebbero interagito o in loco o tramite collegamento con desktop remoto.

Le specifiche del computer di test sono le seguenti:

- CPU Intel Core i5-13600k 5.10GHz
- 32GB DDR4 Ram
- NVIDIA GeForce RTX 3060 12GB

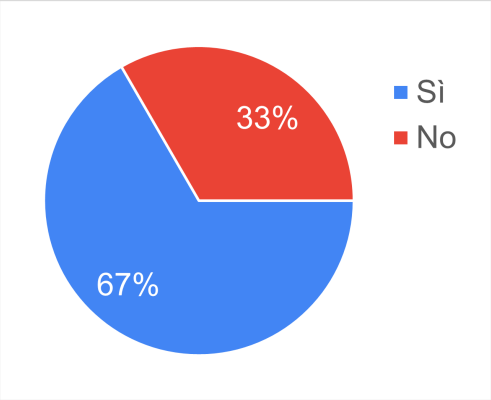
ID	Domanda (Tipologia)
1.1	Età (Scelta multipla)
1.2	Hai mai avuto esperienze con linguaggi di programmazione? (Scelta multipla)
1.3	Hai avuto esperienze con la programmazione in Java? (Scelta multipla)
1.4	Hai avuto esperienze con la programmazione Web? (Scelta multipla)
1.5	Hai avuto esperienze di testing di software? (Scelta multipla)
1.6	Prima dello svolgimento dell'esperimento conoscevi già la gamification? (Scelta multipla)
1.7	Prima dello svolgimento dell'esperimento conoscevi già il crowdsourcing? (Scelta multipla)
2.1	Hai compreso le funzioni di Scout e come utilizzarlo? (Likert)
2.2	Hai compreso la funzione dei micro task? (Likert)
2.3	Hai ritenuto comprensibile la schermata di selezione dei micro task? (Likert)
2.4	Ritieni utile l'aggiunta dei micro task? (Likert)
2.5	Conoscere l'esistenza di una penalità per sessioni poco focalizzate sulle pagine vicine ai micro task ti ha spinto ad esplorare al meglio la pagina di partenza? (Likert)
2.6	Comprendere che il tuo lavoro verrà continuato da un'altra persona ti ha spinto a migliorare le tue performance? (Likert)
2.7	Nel complesso, trovi migliore la versione di scout che utilizza il crowdsourcing rispetto a quella solo gamificata? (Likert)
2.8	Trovi che il tool utilizzato sia facilmente integrabile in un ambiente di testing esistente (lavorativo o di studio)? (Likert)
3.1	Hai riscontrato dei problemi nello svolgimento dell'esperimento? Se sì, descrivili brevemente (Risposta aperta)
3.2	Hai dei suggerimenti per migliorare il tool proposto? (Risposta aperta)

Tabella 4.1: Domande del questionario

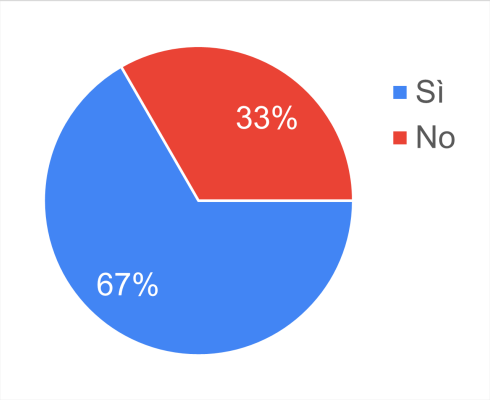
- Memoria NVMe M.2 PCIe 4.0

Dal punto di vista del software installato è presente il sistema operativo Windows 11 con Java 11 e Google Chrome 111. Scout è avviabile tramite il file *.jar* contenente tutte le dipendenze necessarie e con le cartelle per il salvataggio dei file per la gamification e il crowdsourcing già configurate.

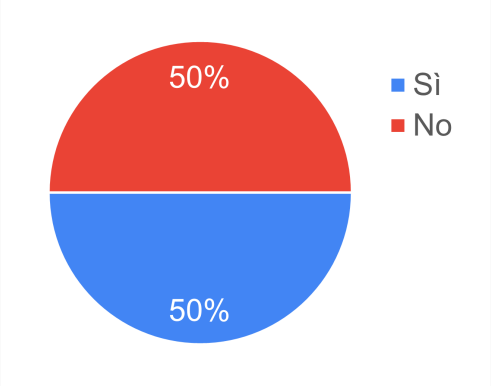
Lo svolgimento delle varie sessioni sullo stesso computer permette di controllare in



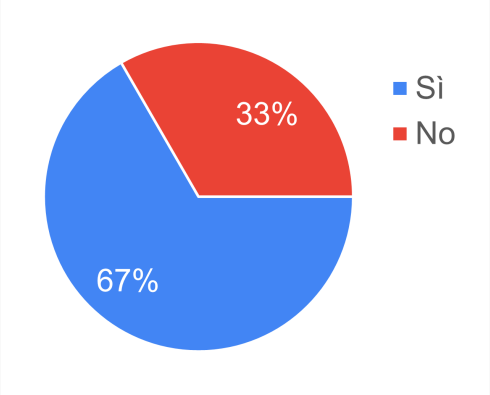
(a) D1.2: Precedenti esperienze con linguaggi di programmazione



(b) D1.3: Precedenti esperienze con la programmazione in Java



(c) D1.4: Precedenti esperienze con la programmazione Web



(d) D1.5: Precedenti esperienze con il testing di software

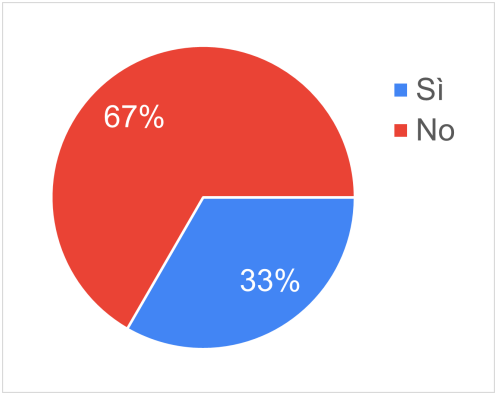
Figura 4.6: Precedenti esperienze del partecipante

maniera accurata l'ambiente di svolgimento sia in termini di risorse computazionali utilizzate che per lo stato in cui si ritrova Scout. L'esecuzione più sessioni in parallelo sullo stesso computer non comporta un problema in quanto Scout può essere avviato molteplici volte senza entrare in conflitto con gli altri processi attualmente in esecuzione.

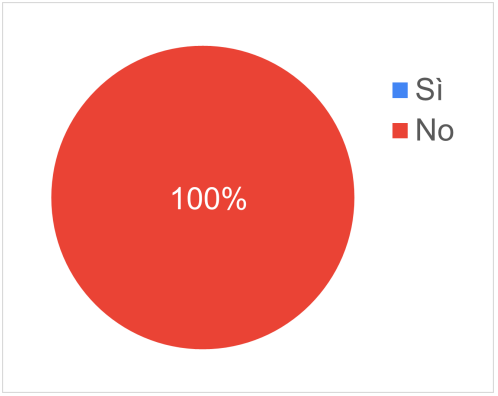
Lo stato di partenza non è nullo e comprende:

- un set di pagine già visitate
- un set di widget con cui sia stata effettuata un'interazione

La presenza delle pagine già visitate comporta la creazione di una serie di micro task che saranno disponibili quando verrà attivato il plugin per il crowdsourcing.



(a) D1.6: Conoscenza della gamification



(b) D1.7: Conoscenza del crowdsourcing

Figura 4.7: Conoscenza delle tematiche affrontate

4.3 Analisi dei risultati

Il plugin è stato appositamente modificato per raccogliere alcuni dati per valutare i risultati dei candidati con le varie configurazioni di test. In particolare vengono raccolte le informazioni sul numero di widget delle pagine visitate, il numero di widget con cui è stata effettuata un'interazione, la coverage finale della specifica pagina e l'aumento di coverage per quella pagina. Di seguito il file *.json* che viene generato.

```
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "$id": "validation-schema.json",

    "type": "object",
    "properties": {
        "scores": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "key": { "type": "string"},
                    "value": {
                        "type": "array",
                        "items": { "type": "number" }
                    }
                }
```

```
18                  }
19              }
20          }
21  }
```

I risultati di ogni pagina sono inseriti in una mappa che ha come chiave l'URL della pagina e come valore un vettore di valori numerici. Il vettore contiene un numero fisso di valori e si distinguono in base alla posizione nel vettore:

1. Numero totale di widget nella pagina
2. Numero di widget con cui è già stata effettuata un'interazione in una sessione precedente
3. Numero di widget con cui è stata effettuata un'interazione al termine della sessione (inclusi quelli di sessioni precedenti)
4. Coverage della pagina
5. Aumento percentuale di coverage rispetto allo stato iniziale della sessione

4.3.1 Coverage

I risultati più importanti dal punto di vista tecnico sono sicuramente quelli inerenti agli aumenti di coverage delle pagine visitate poiché permettono di comprendere l'efficacia dei nuovi elementi introdotti.

In Figura 4.8 e Figura 4.9 sono presenti dei confronti tra i risultati in termini di interazioni con nuovi widget, pagine scoperte, coverage media e incremento di coverage ottenuti nella prima e seconda parte dell'esperimento nei due domini scelti. Nel caso del sito A[6], che è stato selezionato per la sua semplicità, le metriche sono pressoché identiche se non una lieve discrepanza nel numero di widget e nuove pagine scoperte. Tale somiglianza è sicuramente data dalla scelta del micro-task che è stato forzatamente selezionato in modo tale che tutti e tre gli utenti avessero la stessa pagina di partenza e quindi fosse più difficile ottenere un aumento di coverage considerevole nelle pagine vicine alla loro "home".
Passando invece all'analisi dei risultati per il sito B[7] le differenze sono molto evidenti con un aumento di oltre il 35% di nuovi widget evidenziati, una coverage media quasi triplicata e un incremento di coverage medio più che duplicato. Le performance superiori in questo caso possono essere attribuite ad una esplorazione più attenta delle pagine visitate, infatti sono state scoperte meno pagine nuove che avrebbero sicuramente abbassato il valore di coverage e incremento medi.

[6] https://magi-giovanni-funghi-e-tartufi.webnode.it/"

[7] https://www.polito.it/

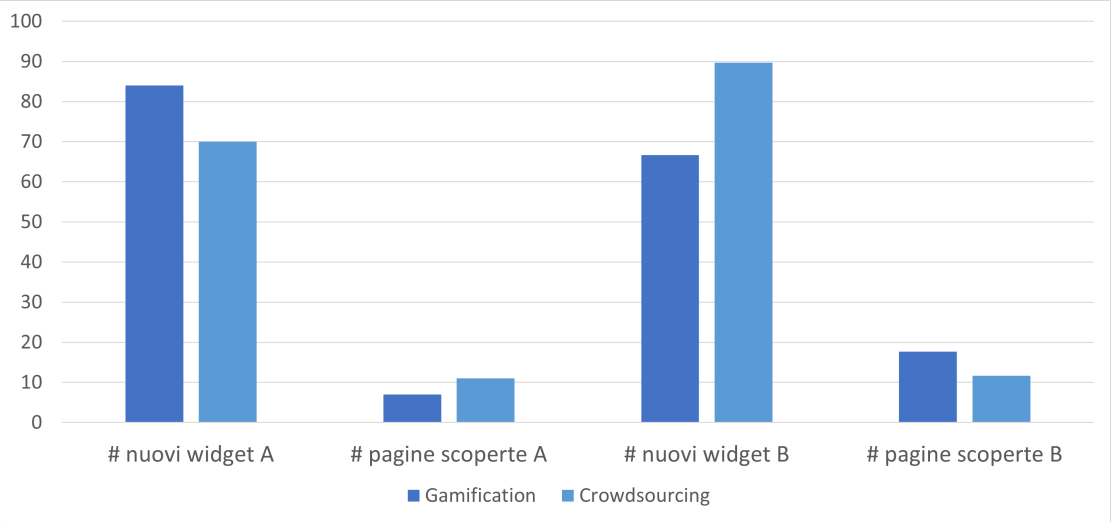


Figura 4.8: Risultati in termini di interazioni con widget e nuove pagine scoperte (A=https://magi-giovanni-funghi-e-tartufi.webnode.it/, B=https://www.polito.it/)

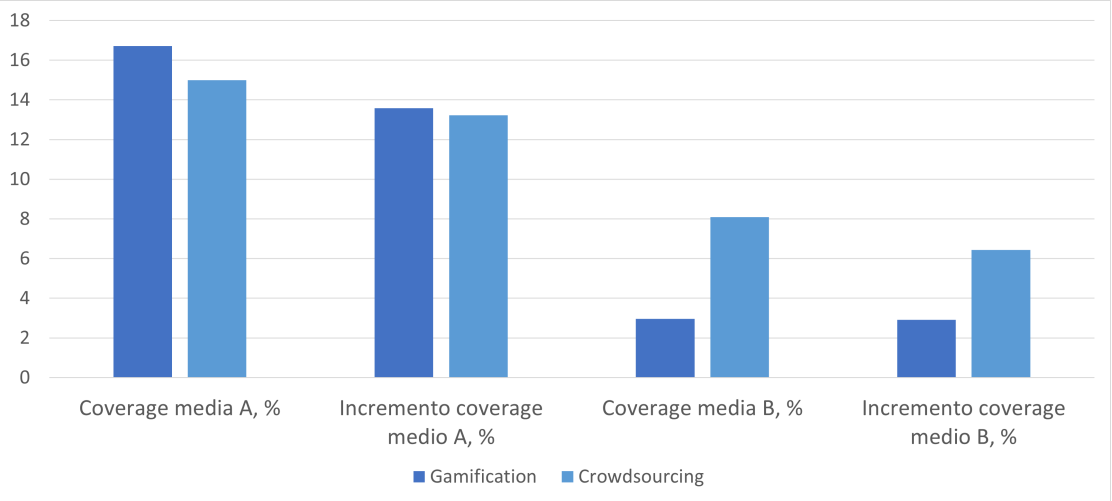


Figura 4.9: Risultati in termini di coverage media e incremento di coverage medio raggiunti (A=https://magi-giovanni-funghi-e-tartufi.webnode.it/, B=https://www.polito.it/)

Per interpretare al meglio i prossimi grafici ricordiamo che i partecipanti sono stati divisi in due gruppi e sono così formati:

- **Gruppo 1**: Utenti 1, 2 e 3. Hanno visitato prima il sito A e poi quello B

- **Gruppo 2**: Utenti 2, 3 e 4. Hanno visitato prima il sito B e poi quello A

In Figura 4.10 e Figura 4.11 vi sono rispettivamente i dati di coverage iniziale e incremento di coverage locali per le sessioni con il solo plugin per la gamification attivo e per le sessioni con entrambi i plugin abilitati. Ricordiamo che quando si parla di coverage locale si intende la coverage raggiunta da un utente nelle pagine a distanza minore di tre dal nodo di partenza, con distanza intesa come numero di link cliccati a partire dalla home del micro task per arrivare alla pagina considerata. In questo caso i dati di coverage locale si sono limitati all'analisi della pagina di partenza della sessione dato il poco tempo disponibile per l'esplorazione.

Per concludere si è fatto un confronto tra gli incrementi di coverage locale raggiunti dalle due parti dell'esperimento per ogni utente (Figura 4.12). Nel caso della versione gamificata gli incrementi di coverage sulla pagina di partenza sono pressoché nulli, infatti il pulsante per tornare alla home è stato premuto raramente e non è stata data importanza al raggiungimento di una coverage elevata nelle pagine quanta ne sia stata assegnata alla scoperta di nuovi widget. Quando invece si è passati alla sessione con il plugin multi-utente abilitato, il primo obiettivo è stato il raggiungimento di una buona coverage locale e come si nota dai risultati è particolarmente evidente per la pagina di partenza.

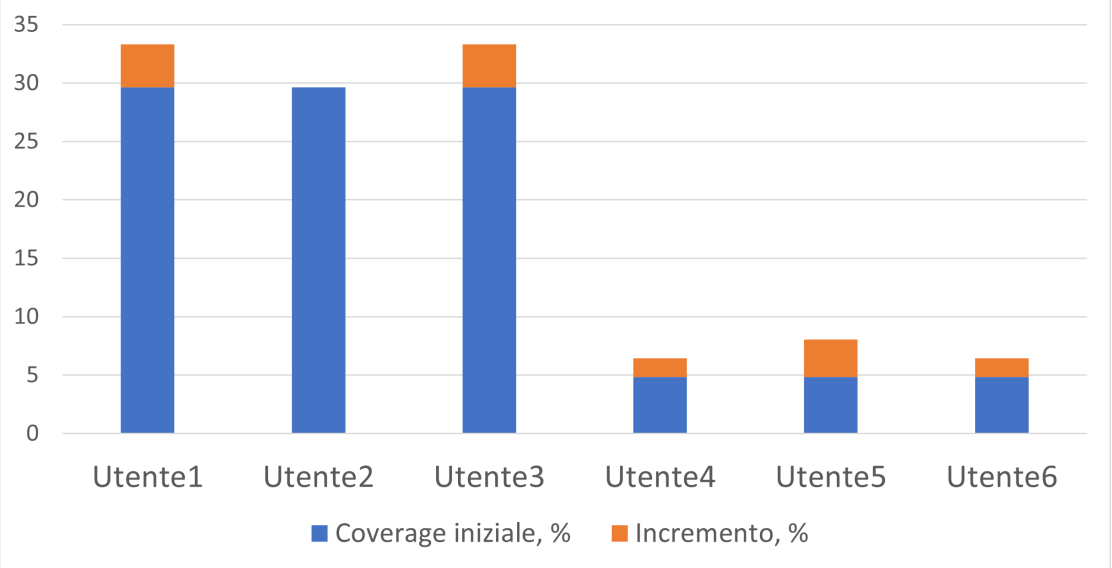


Figura 4.10: Coverage locale con il plugin di gamification attivo

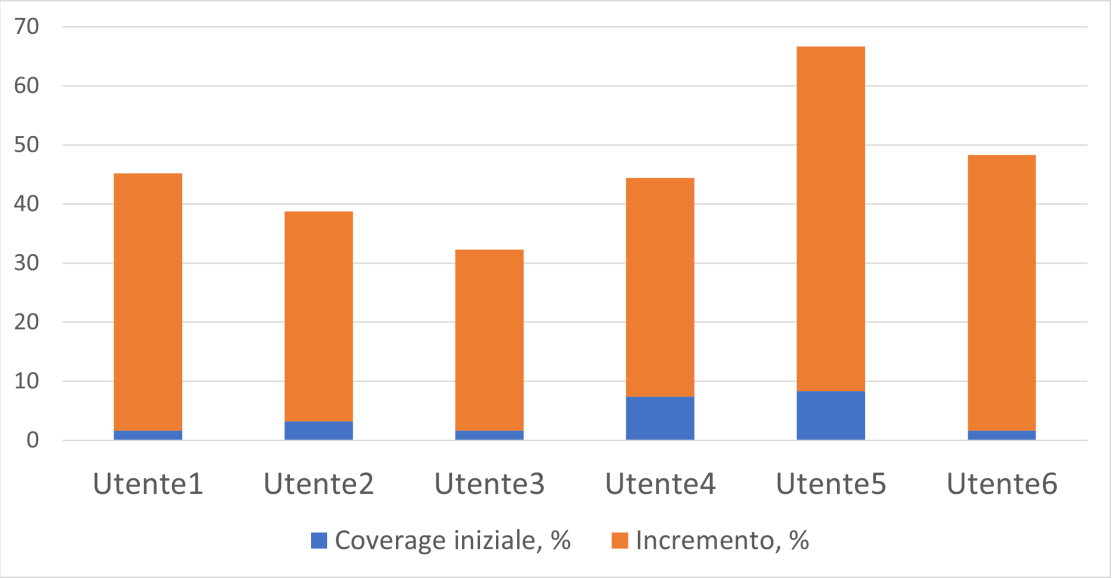


Figura 4.11: Coverage locale con i plugin di gamification e crowdsourcing attivi

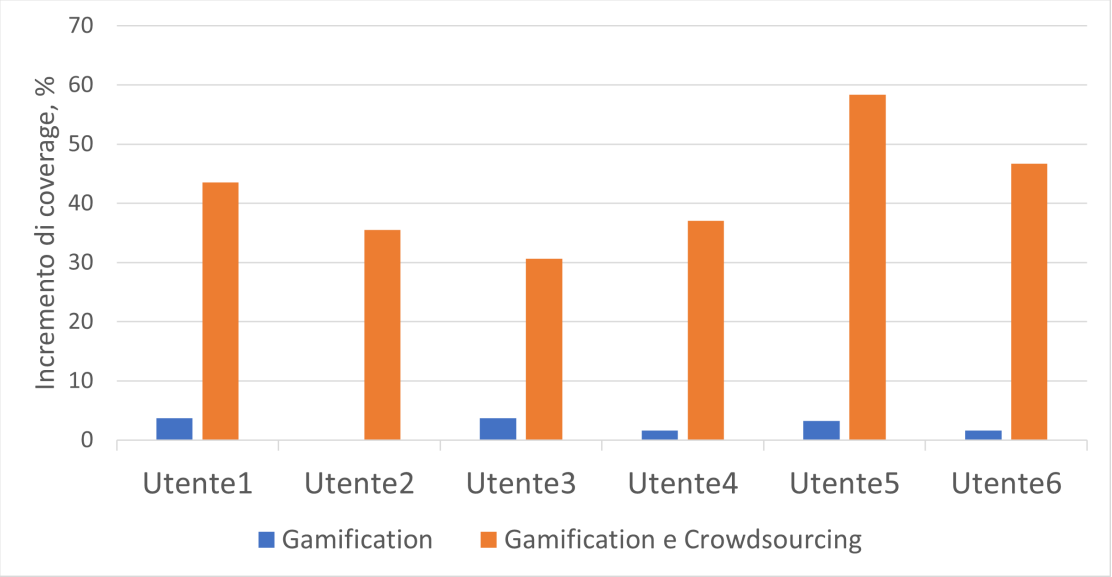


Figura 4.12: Confronto tra gli incrementi di coverage locale raggiunti da ogni utente nelle due fasi dell'esperimento

4.3.2 Nuove pagine scoperte

Dai risultati indicati in Figura 4.8 il numero di pagine scoperte non sembra essere influenzato dalla presenza o meno del plugin multi-utente. Gli utenti del primo

gruppo hanno aumentato il numero di pagine scoperte durante la seconda fase con un aumento in media di 4,67 pagine, mentre per gli utenti del secondo gruppo sono diminuite in media di 6,67. Si può dire che questo parametro dipenda solo dalla tipologia di sito visitato e dalle abilità del singolo tester.

Dai dati raccolti è stato evidenziato come le nuove pagine scoperte con il plugin multi-utente abbiano una percentuale di coverage inferiore tanto più sono situate in profondità rispetto al nodo di partenza poiché è stata preferenza comune tornare alla home del micro task piuttosto di esplorare le pagine scoperte. Questo è esattamente quello che si è voluto ottenere con l'introduzione della penalità e permette la generazione di micro-task di qualità ovvero per i quali sia possibile ottenere un buon incremento di coverage già sulla prima pagina.

4.3.3 Micro task

L'elemento principale introdotto con il nuovo plugin multi-utente sono i micro task. Le domande 2.2, 2.3, 2.4 e 2.5 si sono focalizzate su questo nuovo elemento con la possibilità di rispondere in modo positivo, neutro o negativo ad un determinato quesito. La scala di valutazione utilizzata è una scala Likert a cinque punti (Molto in disaccordo, Disaccordo, Né d'accordo né in disaccordo, D'accordo, Molto d'accordo).

In Figura 4.13 e 4.14 sono presenti rispettivamente le risposte alle domande 2.2 e 2.3. La spiegazione dell'elemento dei micro task e la loro funzione è stata ben recepita con ben cinque persone che hanno colto in ogni aspetto la necessità dell'introduzione di questo elemento. La schermata visibile poco prima dell'avvio della sessione è stata definita molto semplice ma efficace, con una buona percentuale di approvazione, in particolare sono stati due favorevoli e quattro molto favorevoli.

L'introduzione sul tema del crowdsourcing ha proposto anche degli esempi su scenari reali senza la presenza dei micro task e la perdita di efficienza delle sessioni di testing nel caso di domini molto grandi. Nella domanda 2.4 di cui i risultati in Figura 4.15 è stata chiesta una valutazione sull'utilità di questo elemento e cinque partecipanti lo hanno ritenuto molto utile.

Essendo lo scopo dei micro task quello di aumentare la coverage locale della pagina di partenza, l'introduzione della penalità è stata vista dai partecipanti come un fattore chiave per spingerli a testarle al meglio. Questo elemento ha ricevuto la piena approvazione come indicato in Figura 4.16. Ne è derivato che durante le sessioni multi-utente ci siano state molte più interazioni con la toolbar e il pulsante "Go Micro Task Home" rispetto al pulsante "Go Home" per il quale non vi è stata quasi alcuna interazione nelle sessioni con il solo plugin di gamification.

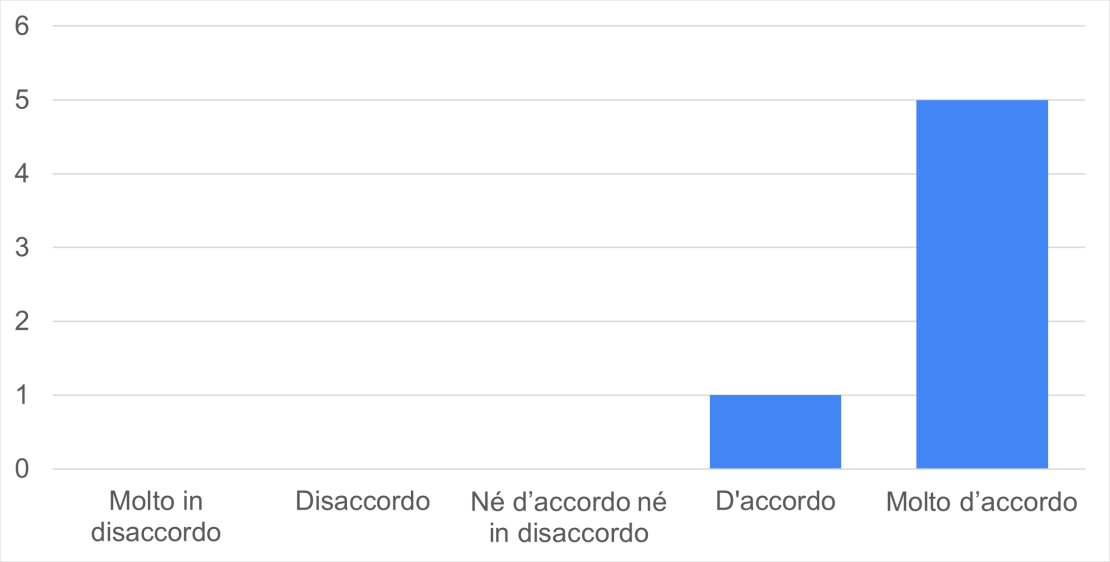


Figura 4.13: Comprensione delle funzioni dei micro task

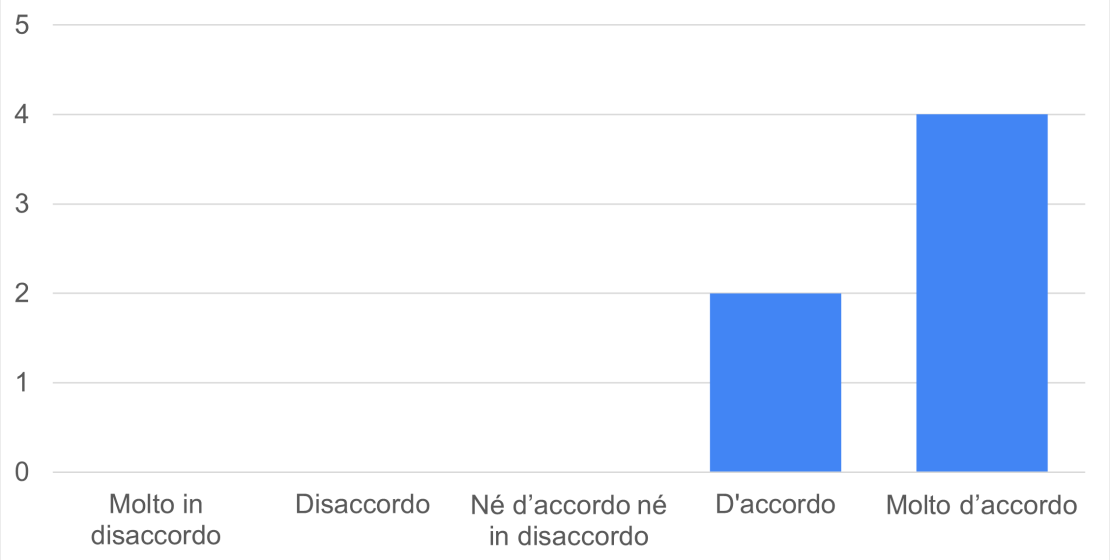


Figura 4.14: Valutazione della comprensibilità della schermata di selezione dei micro task

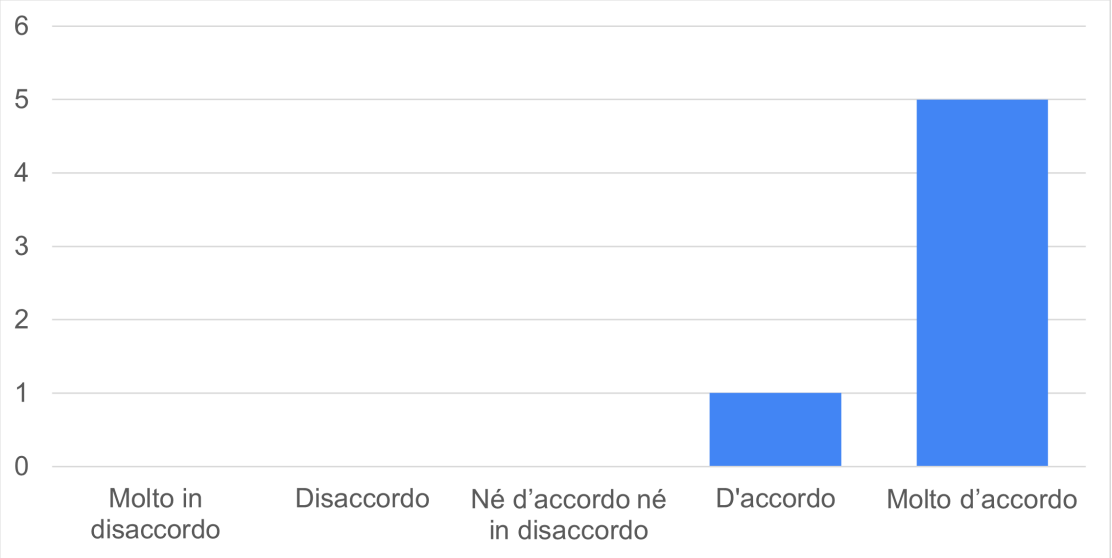


Figura 4.15: Valutazione dell'utilità dei micro task

Figura 4.16: Valutazione sull'efficacia dell'elemento di penalità

4.3.4 Giudizio complessivo

Nell'ultima parte delle domande a scala Likert a cinque punti, ai partecipanti è stato chiesto un parere su Scout e i nuovi plugin integrati ad esso.

I risultati della domanda 2.1 rappresentati in Figura 4.17 indicano che le funzionalità di Scout e gli obiettivi che si pone introducendo la tecnica di GUI testing di tipo record & replay sono stati ben compresi. Tre persone hanno indicato la comprensione totale del tool, mentre altri tre sono abbastanza convinti di aver inquadrato le sue funzioni con qualche piccolo dubbio sulle funzionalità più specifiche.

La domanda 2.6 (Figura 4.18) ha avuto l'obiettivo di valutare quanto la componente collaborativa del nuovo plugin possa aver influenzato le azioni degli utenti. La consapevolezza di lavorare in un ambiente collaborativo ha spinto cinque utenti ad impegnarsi nel fornire dei test accurati volti ad ottenere una coverage di pagina più elevata, mentre una persona non ha ritenuto tale aspetto stimolante.

La preferenza tra la versione di Scout ludicizzata e quella anche multi-utente (domanda 2.13), della quale i risultati sono osservabili in Figura 4.19, volge fortemente a favore della versione gamificata multi-utente del tool: cinque partecipanti sono "molto d'accordo" sul fatto che la versione multi-utente di Scout sia preferibile a quella solo ludicizzata, mentre uno è semplicemente "d'accordo". La preferenza non è data solo dallo stimolo aggiuntivo dell'ambiente cooperativo ma anche dai vantaggi che vengono offerti dai micro-task e la possibilità di svolgere sessioni in parallelo.

Nell'ultima domanda di questa sezione (domanda 2.8), è stato richiesto ai tester

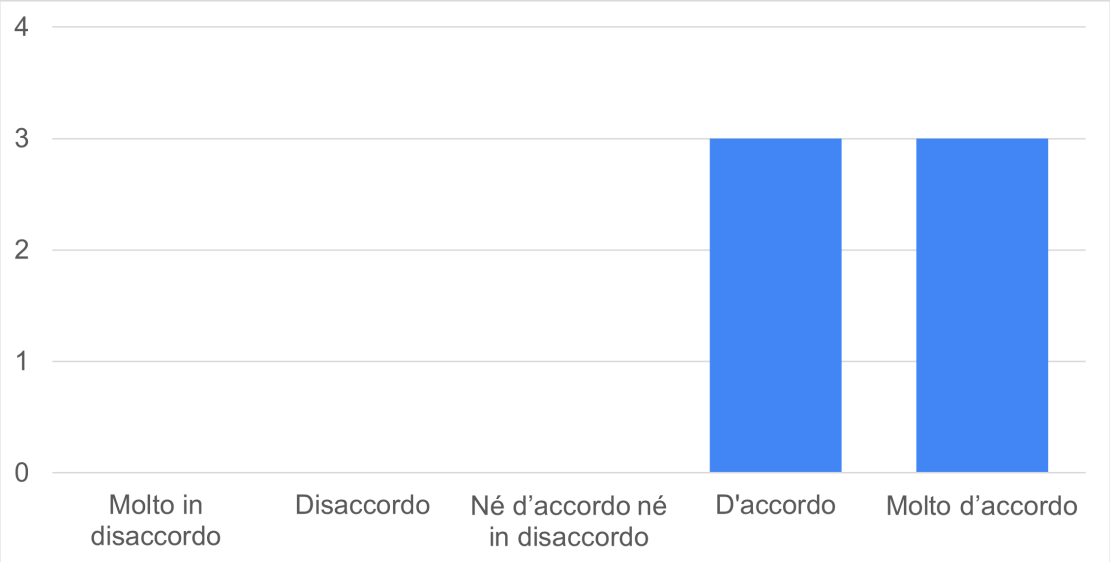


Figura 4.17: Comprensione delle funzionalità e utilizzi di Scout

se ritenessero che il tool con i due plugin attivi possa essere integrato facilmente nel contesto di un ambiente di testing già esistente (lavorativo o di studio). Le risposte (Figura 4.20) sono fortemente positive nella visione di una versione aggiornata.

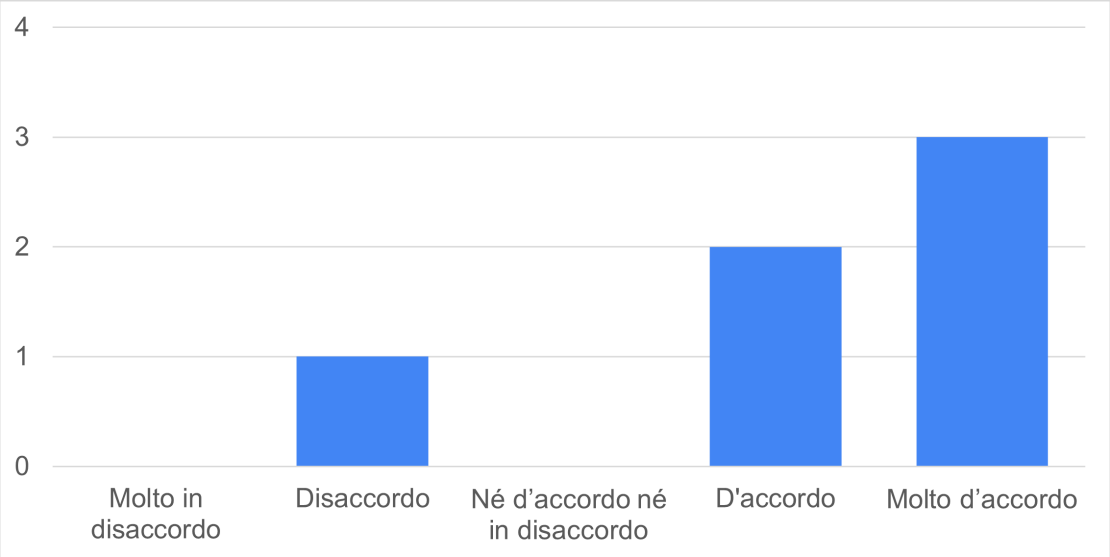


Figura 4.18: Valutazione dell'impatto della componente collaborativa del plugin multi-utente

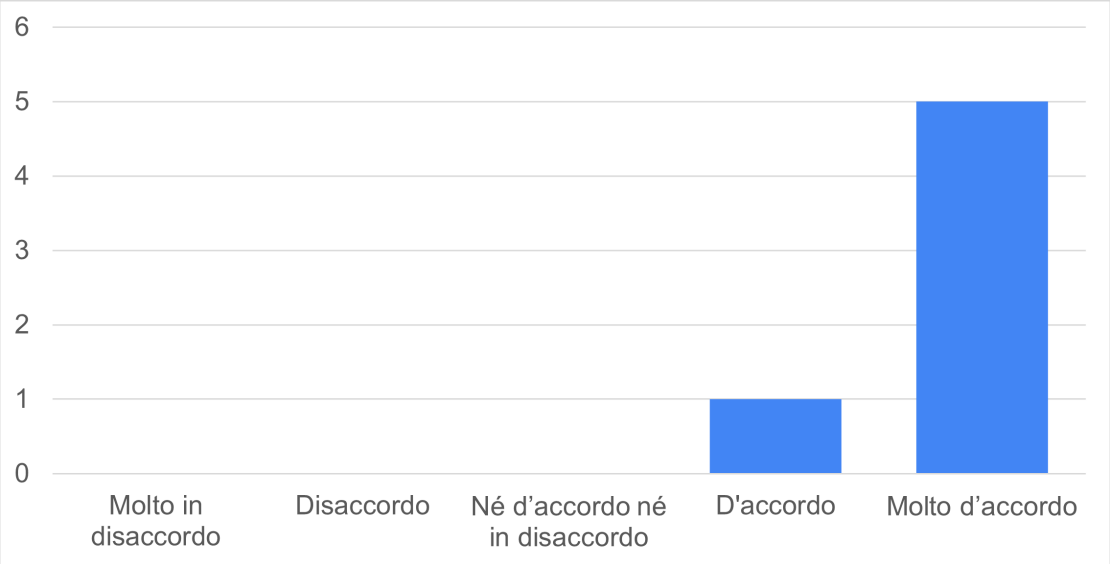


Figura 4.19: Valutazione sulla preferibilità della versione multi-utente di Scout rispetto a quella ludicizzata

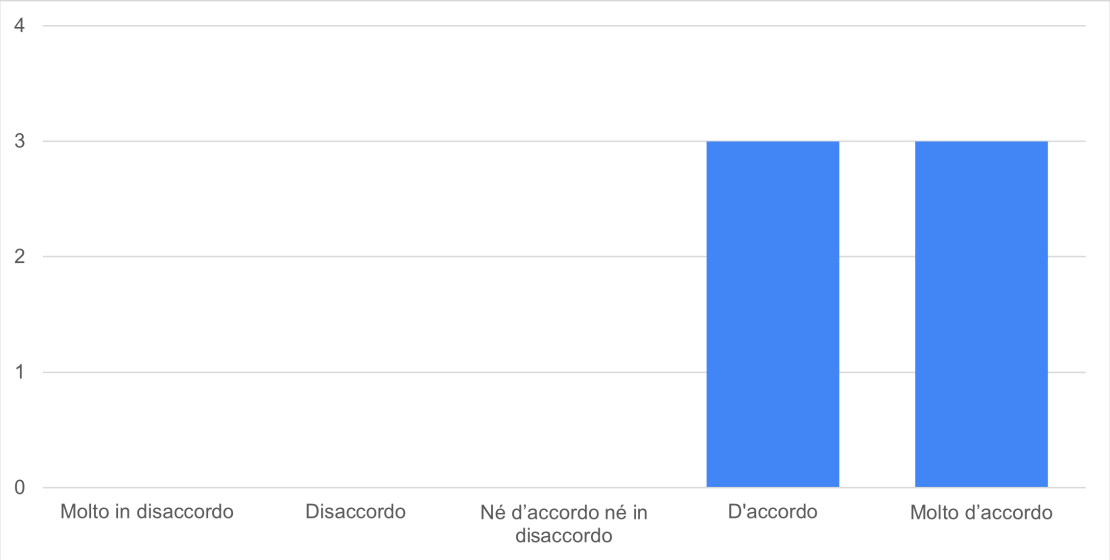


Figura 4.20: Valutazione della facilità di integrare il tool in un contesto di testing esistente

4.4 Problematiche e suggerimenti

Nella domanda 3.1 è stato chiesto un riscontro riguardo al verificarsi di problematiche tecniche durante l'esperimento. Tre persone hanno affermato che non vi è stato alcun problema. Due persone hanno individuato nella lentezza del caricamento delle pagine una problematica abbastanza grave che ha portato a non capire quando Scout stesse caricando una nuova pagina o se potessero già iniziare ad interagire con quella visualizzata. Un utente ha riportato la problematica legata agli easter egg che si posizionano sopra ai widget interagibili e che hanno istintivamente forzato la curiosità di provare a nasconderli cliccandoli senza portare ad alcun risultato.

Per la domanda 3.2, essendo legata al tema del crowdsourcing che è stato una cosa nuova per tutti i partecipanti, i suggerimenti proposti sono stati due.
Il nuovo plugin pone ancor più importanza nel testing con coverage elevata nelle pagine vicino alla radice del micro task. Un partecipante ha proposto l'inserimento di un pop-up di avvisa quando la pagina raggiunge una coverage del 100% e l'inserimento di un'indicazione numerica del numero di widget testati rispetto al totale di quelli presenti, non ritenendo sufficiente la barra di progresso.
Durante i test con la versione multi-utente tutti i partecipanti hanno interagito frequentemente con l'Augmented Toolbar per utilizzare il pulsante "Go Micro Task Home" e aumentare la coverage locale al nodo di partenza. La proposta è l'aggiunta di scorciatoie da tastiera per velocizzare questa azione ed eliminare il tempo di interazione con la toolbar.

Capitolo 5

Conclusioni

Il software testing viene spesso "demonizzato" per i costi elevati e l'apparenza di non portare benefici, ma come abbiamo visto può diventare coinvolgente attraverso l'utilizzo di nuove tecniche quali la gamification e il crowdsourcing. Il maggiore coinvolgimento dei tester si traduce in un aumento dell'efficienza e la qualità dei test case sottomessi e quindi una riduzione dei costi legati a questa attività.

Questa tesi si è posta come obiettivo quello di dimostrare l'applicabilità del crowdsourcing all'ambito dell'exploratory testing di interfacce web a partire dal software Scout, che implementa le funzionalità di base per effettuare un testing di tipo record & replay, e un plugin per l'introduzione degli elementi di gamification. Il contesto preesistente ha previsto l'impiego del solo aspetto competitivo della gamification, mentre gli intenti sono stati quelli di introdurre anche l'aspetto collaborativo fornendo la possibilità di eseguire sessioni multi-utente.

Per rendere il software Scout multi-utente è stato scelto un modello degli stati che permettesse l'unione dei test case dei singoli utenti, ampliando quindi il numero di persone che possono essere coinvolte nel testing di un dominio. Per riuscire a parallelizzare ulteriormente le attività di testing è stato introdotto anche il concetto di micro task, che innanzitutto riduce la probabilità di collisione di test case di utenti differenti e inoltre garantisce un incremento sicuro di coverage almeno per le pagine vicine a quella di partenza. Questi sono stati gli elementi fondamentali per arrivare ai risultati ottenuti.

L'esperimento che è stato condotto ha visto un piccolo campione di sei persone utilizzare Scout con i plugin di gamification e/o crowdsourcing. Nella sezione 4.3.1 viene descritto come vi sia stato un lieve aumento delle prestazioni complessive dei partecipanti con l'aggiunta del nuovo plugin. Il grande passo in avanti si ha avuto nei risultati dati dalla coverage locale, che ricordiamo sia stata intesa come coverage della pagina di partenza a causa della durata ridotta della sessione. Questi risultati vanno sicuramente interpretati poiché nelle sessioni effettuate lo stato preesistente comprendeva solo poche pagine visitate e poche interazioni con i widget.

In un contesto in cui il testing viene effettuato da un lungo periodo di tempo sullo stesso dominio, l'efficienza delle sessioni solo gamificate calerebbe drasticamente perché non si hanno indicazioni sulle pagine che necessitano di ulteriori test. Con il plugin multi-utente le sessioni di testing partirebbero già da una pagina in cui sia possibile ottenere un aumento di coverage migliorando l'efficienza generale dei tester. Le valutazioni personali dei partecipanti all'esperimento indicano che sono state ben comprese le funzionalità introdotte e i vantaggi che implicano in un contesto multi-utente, con grande approvazione verso i micro task.

Gli elementi tipici del crowdsourcing sono stati correttamente applicati confermando quindi l'applicabilità del crowdsourcing anche in questo settore con dei benefici non indifferenti.

5.1 Limiti di applicabilità

Il contesto in cui è stato sviluppato e testato il nuovo plugin per il crowdsourcing ha comportato l'introduzione di alcune semplificazioni e assunzioni che si traducono in un campo di applicabilità più ristretto.

5.1.1 Limiti di validità interna

La metodologia con la quale è stato condotto lo studio e la presenza di elementi non considerati possono aver portato durante la fase di validazione a risultati che si discostano da quelli effettivamente ottenibili. Una parte dei limiti è stata introdotta consapevolmente mentre altri sono legati al contesto preesistente che si è deciso di adottare.

Durante la fase di validazione tutto ciò che concerne la sincronizzazione dei dati tra vari utenti è stato gestito in modo automatico dal sistema per la maggior parte delle operazioni, mentre è stato gestito manualmente nel caso in cui si sono volute simulare delle sessioni parallele. Aggiungendo una limitazione di Windows 11 per cui la dimensione del nome di una cartella non può eccedere un numero di caratteri predefinito hanno reso difficile e rischiosa la copia di uno stato preesistente in una sessione in corso per simulare la sottomissione di risultati per sessioni con lo stesso stato di partenza.
Per semplicità ogni utente viene contraddistinto dal proprio ID testuale che viene scelto in fase di selezione del dominio da testare. La limitazione sorge nel momento in cui due utenti scegliessero lo stesso ID. I due utenti non verrebbero distinti dal sistema e si potrebbero avere anche casi di impersonificazione. Essendo il campione dell'esperimento ridotto e la sessione guidata non vi sono stati problemi correlati a questo fenomeno.

Una delle principali problematiche legate all'implementazione di Scout è sicuramente una visione limitata della pagina web da testare: le pagine risultano

zoomate e per su quelle più lunghe non è possibile scorrere fino in fondo, non permettendo quindi il test di tutti i widget presenti. Per evitare di riversare questo problema sul punteggio e sul calcolo della coverage vengono considerati solo i widget effettivamente visibili, ma in future versioni sarà sicuramente necessario risolvere questa problematica.
Ogni widget occupa una certa area e quando viene evidenziato la sua area viene evidenziata da un rettangolo colorato. Questi elementi aggiuntivi che sono parte dell'interfaccia grafica aumentata rischiano di interferire con altri widget sovrapponendosi e/o impossibilitando le interazioni con essi o comunque portando il tester a effettuare altre operazioni.
Scout è un software poco ottimizzato che necessita di una grande quantità di risorse computazionali per poter creare l'interfaccia aumentata e permettere tutte le operazioni di emulazione per interagire con il sito web. Spesso i tester, spinti dal voler ottenere un buon risultato, hanno eseguito sequenze di click molto vicine scambiando widget che richiedono una *check action* con widget che invece erano link. Queste sequenze vicine hanno portato a non considerare molte interazioni poiché dopo un'interazione con un link Scout deve caricare la nuova pagina e lo stato diventa inconsistente fino al completamento del caricamento dei nuovi widget dello stato.
Oltre al problema delle performance si sono venuti a creare widget in apparenza senza elemento corrispondente per elementi che hanno cambiato la propria posizione o la conformazione della pagina. Un esempio potrebbe essere un menù che si espande o ritrae al click oppure un elemento che ne genera altri in sovrimpressione. I widget "fantasma" assieme a widget che potrebbero aver cambiato coordinate nello schermo hanno portato molta confusione nel campione dell'esperimento.

Il plugin di gamification è stato sviluppato legando le proprie strutture dati e meccanismi alla classe Selenium plugin, il che ha forzato ad integrare ad essa anche i meccanismi del plugin multi-utente. Questo stretto legame porta alcuni svantaggi in quanto un futuro sviluppo di altri plugin o modifiche del comportamento base di Scout rischiano di diventare onerose e difficili da applicare come è già avvenuto per lo sviluppo del plugin di questa tesi.
La presenza degli easter egg in sovrimpressione sulle pagine selezionate diventa un problema nel momento in cui vanno a ricoprire completamente un widget poiché non sono removibili con un click e quindi non permetterebbero di raggiungere una coverage completa.

5.1.2 Limiti di validità esterna

Viene ora discussa l'applicabilità dei risultati al mondo reale quindi in un contesto diverso e più complesso di quello considerato durante la fase di validazione.

Si riprende ora il tema sulla scelta della definizione di stato nel modello introdotta nella sezione 3.3.1. La definizione adottata in cui ogni URL corrisponde ad uno stato implica notevoli semplificazioni che vedrebbero il tool di difficile applicazione in un contesto reale le cui pagine web sono sempre più dinamiche fino al punto in cui tutto il dominio potrebbe essere composto da una singola pagina (single page application). La scelta intermedia in cui si crea un "riassunto" (hash) degli elementi di una pagina potrebbe invece essere valida per un numero maggiore di pagine web e riuscirebbe a catalogare ulteriormente pagine con la soluzione adottata sarebbero ricadute nello stesso stato.

Le interazioni con tutti quegli elementi che rendono la pagina web dinamica non sono supportate da Scout. Ad ogni click dell'utente su un elemento viene effettuato un controllo sulla natura di quest'ultimo distinguendo quindi semplici link da elementi che contengono sottostringhe quali "javascript:", "ftp:", "mailto:", e che il link non contenga il riferimento ad un fragment ("#"). La presenza di script in una pagina comporta l'impossibilità di testare il comportamento dell'interezza degli elementi presenti in essa. Per limitare questa problematica i domini di prova dell'esperimento sono stati scelti per la presenza contenuta di elementi dinamici e quindi la possibilità di testarli quasi completamente.

L'utilizzo di Scout con il nuovo plugin abilitato potrebbe non aver portato problemi rilevanti in termini di memoria occupata durante l'esperimento, ma diventerebbe sicuramente un problema in un dominio reale che può contenere anche centinaia di migliaia di pagine differenti. Tutte le pagine conosciute e l'albero degli stati vengono caricati in memoria all'avvio della sessione, creando quindi gli oggetti Java corrispondenti. L'esperimento ha visto il caricamento di un set limitato di pagine dei due domini conosciuti e un utilizzo di memoria quasi nullo. In un contesto reale lo stato e le varie rappresentazioni delle pagine conosciute dovrebbero essere caricati dinamicamente all'occorrenza.

5.2 Lavori futuri

Alla luce dei risultati ottenuti e con la conoscenza degli attuali limiti imposti vengono ora elencati alcuni punti di miglioramento sui quali lavorare in eventuali lavori futuri:

- Miglioramento dell'efficienza di tutte quelle operazioni che richiedono un elevato carico computazionale per avere un'esperienza utente più fluida e meno errori involontari nelle interazioni.

- Individuazione di un modello degli stati che sia compatibile con le moderne applicazioni web dinamiche in modo tale da ampliare lo spettro di elementi testabili.

- Gestione dei dati in un database centralizzato per poter gestire sessioni di testing su più dispositivi e spostare il carico computazionale della fase di unione dei test case e generazione dei micro task.
- Introdurre un gestore dei micro task che riesca a definire i limiti di un task e ne comunichi il completamento all'utente a cui sia stato assegnato.

Bibliografia

[1] M. Tuteja e G. Dubey. «A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models». In: *International Journal of Soft Computing and Engineering (IJSCE)* 2 (3 lug. 2012), pp. 251–257 (cit. a p. 1).

[2] R. Coppola F. Cacciotto T. Fulcini e L. Ardito. «Gamification Applicata al GUI Testing di Applicazioni Web». Tesi di Laurea Magistrale. Torino, Italia: Politecnico di Torino, 2021 (cit. alle pp. 2, 37, 70).

[3] Andreas Bauer e Emil Alégroth. «We Tried and Failed: An experience report on a collaborative workflow for GUI-based testing». [Non pubblicato]. N.D. (Cit. alle pp. 2, 51).

[4] R. E. Fairley Pierre Bourque e IEEE Computer Society. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010, pp. 154–155. ISBN: 978-1-139-49176-1 (cit. a p. 5).

[5] C.J. Oberkampf W.L.; Roy. *Guide to the software engineering body of knowledge*. IEEE Computer Society, 2014. ISBN: 0-7695-5166-1 (cit. a p. 6).

[6] Cem Kaner. «A Tutorial in Exploratory Testing». In: *QUEST*. Chicago, USA, 2008 (cit. a p. 6).

[7] Atif M. Memon, Mary Lou Soffa e Martha E. Pollack. «Coverage Criteria for GUI Testing». In: *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ESEC/FSE-9. Vienna, Austria: Association for Computing Machinery, 2001, pp. 256–267. ISBN: 1581133901. DOI: 10.1145/503209.503244. URL: https://doi.org/10.1145/503209.503244 (cit. a p. 7).

[8] Alex Ruiz e Yvonne Wang Price. «GUI Testing Made Easy». In: *Testing: Academic & Industrial Conference - Practice and Research Techniques (taic part 2008)*. 2008, pp. 99–103. DOI: 10.1109/TAIC-PART.2008.11 (cit. a p. 8).

[9] Michel Nass, Emil Alégroth e Robert Feldt. «On the Industrial Applicability of Augmented Testing: An Empirical Study». In: *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2020, pp. 364–371. DOI: 10.1109/ICSTW50294.2020.00065 (cit. alle pp. 8, 37).

[10] Kai Huotari e Juho Hamari. «Defining Gamification: A Service Marketing Perspective». In: MindTrek '12. Tampere, Finland: Association for Computing Machinery, 2012. ISBN: 9781450316378. DOI: 10.1145/2393132.2393137. URL: https://doi.org/10.1145/2393132.2393137 (cit. a p. 9).

[11] Sebastian Deterding, Dan Dixon, Rilla Khaled e Lennart Nacke. «From Game Design Elements to Gamefulness: Defining "Gamification"». In: MindTrek '11. Tampere, Finland: Association for Computing Machinery, 2011, pp. 9–15. ISBN: 9781450308168. DOI: 10.1145/2181037.2181040. URL: https://doi.org/10.1145/2181037.2181040 (cit. alle pp. 9, 10).

[12] Anitha Rao Gadiyar. «Gamification 3.0: The power of personalization». In: *White paper. Cognizant's Global Technology* (2014) (cit. a p. 9).

[13] Juho Hamari, Jonna Koivisto e Harri Sarsa. «Does Gamification Work? – A Literature Review of Empirical Studies on Gamification». In: *2014 47th Hawaii International Conference on System Sciences*. 2014, pp. 3025–3034. DOI: 10.1109/HICSS.2014.377 (cit. a p. 10).

[14] Kevin Werbach e Dan Hunter. *The gamification toolkit: dynamics, mechanics, and components for the win*. University of Pennsylvania Press, 2015 (cit. a p. 11).

[15] Richard A Bartle. «Player types». In: *Jeannie Novak: Game Development Essentials* (2008), pp. 39–40 (cit. alle pp. 12, 21).

[16] Nicole Lazzaro. «Why we play games: Four keys to more emotion without story». In: *Game Developers Conference*. Vol. 8. 2004 (cit. a p. 14).

[17] *A Simple Gamification Framework / Cheat Sheet*. www.gamified.uk/gamification-framework/. Accessed: 15/11/2022 (cit. a p. 15).

[18] Bahram Hooshyar Yousefi e Hana Mirkhezri. «Lean Gamification Canvas : A New Tool for Innovative Gamification Design Process». In: *2020 International Serious Games Symposium (ISGS)*. 2020, pp. 1–9. DOI: 10.1109/ISGS51981.2020.9375297 (cit. a p. 17).

[19] Yu-Kai Chou. «Actionable gamification». In: *Beyond points, badges, and leaderboards* (2015) (cit. a p. 18).

[20] Manal M. Alhammad e Ana M. Moreno. «Gamification in software engineering education: A systematic mapping». In: *Journal of Systems and Software* 141 (2018), pp. 131–150 (cit. a p. 21).

[21] Benedikt Morschheuser, Lobna Hassan, Karl Werder e Juho Hamari. «How to design gamification? A method for engineering gamified software». In: *Information and Software Technology* 95 (2018), pp. 219–237 (cit. a p. 22).

[22] Laura Inozemtseva e Reid Holmes. «Coverage is Not Strongly Correlated with Test Suite Effectiveness». In: ICSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 435–445. ISBN: 9781450327565. DOI: 10.1145/2568225.2568271. URL: https://doi.org/10.1145/2568225.2568271 (cit. a p. 23).

[23] José Miguel Rojas e Gordon Fraser. «Code Defenders: A Mutation Testing Game». In: *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2016, pp. 162–167. DOI: 10.1109/ICSTW.2016.43 (cit. a p. 24).

[24] José Miguel Rojas, Thomas D. White, Benjamin S. Clegg e Gordon Fraser. «Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game». In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, pp. 677–688. DOI: 10.1109/ICSE.2017.68 (cit. a p. 24).

[25] Reza Meimandi Parizi. «On the gamification of human-centric traceability tasks in software testing and coding». In: *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*. 2016, pp. 193–200. DOI: 10.1109/SERA.2016.7516146 (cit. a p. 24).

[26] Daren C Brabham. *Crowdsourcing*. Mit Press, 2013 (cit. a p. 25).

[27] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan e Jianhua Feng. «ICrowd: An Adaptive Crowdsourcing Framework». In: SIGMOD '15. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 1015–1030. ISBN: 9781450327589. DOI: 10.1145/2723372.2750550. URL: https://doi.org/10.1145/2723372.2750550 (cit. a p. 26).

[28] Lars Janzik. «Contribution and participation in innovation communities: A classification of incentives and motives». In: *International Journal of Innovation and Technology Management* 7.03 (2010), pp. 247–262 (cit. a p. 27).

[29] Teresa M Amabile. «The Case for a Social Psychology of Creativity». In: *The Social Psychology of Creativity*. Springer, 1983, pp. 3–15 (cit. a p. 27).

[30] Teresa M Amabile. «Brilliant but cruel: Perceptions of negative evaluators». In: *Journal of Experimental Social Psychology* 19.2 (1983), pp. 146–156 (cit. a p. 27).

[31] Mokter Hossain. «Users' motivation to participate in online crowdsourcing platforms». In: *2012 International Conference on Innovation Management and Technology Research*. 2012, pp. 310–315. DOI: 10.1109/ICIMTR.2012.6236409 (cit. a p. 27).

[32] Antonio Ghezzi, Donata Gabelloni, Antonella Martini e Angelo Natalicchio. «Crowdsourcing: a review and suggestions for future research». In: *International Journal of Management Reviews* 20.2 (2018), pp. 343–363 (cit. a p. 27).

[33] Yaron Singer e Manas Mittal. «Pricing Mechanisms for Crowdsourcing Markets». In: WWW '13. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, pp. 1157–1166. ISBN: 9781450320351. DOI: 10.1145/2488388.2488489. URL: https://doi.org/10.1145/2488388.2488489 (cit. a p. 29).

[34] Caryn Conley e Jennifer Tosti-Kharas. «Crowdsourcing content analysis for managerial research». In: *Management Decision* 52.4 (2014), pp. 675–688 (cit. a p. 30).

[35] Aslı Sarı, Ayşe Tosun e Gülfem Işıklar Alptekin. «A systematic literature review on crowdsourcing in software engineering». In: *Journal of Systems and Software* 153 (2019), pp. 200–219 (cit. a p. 31).

[36] Melissa A Valentine, Daniela Retelny, Alexandra To, Negar Rahmati, Tulsee Doshi e Michael S Bernstein. «Flash organizations: Crowdsourcing complex work by structuring crowds as organizations». In: *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017, pp. 3523–3537 (cit. a p. 31).

[37] Ke Mao, Licia Capra, Mark Harman e Yue Jia. «A survey of the use of crowdsourcing in software engineering». In: *Journal of Systems and Software* 126 (2017), pp. 57–84 (cit. a p. 32).

[38] Fabrizio Pastore, Leonardo Mariani e Gordon Fraser. «CrowdOracles: Can the Crowd Solve the Oracle Problem?» In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. 2013, pp. 342–351. DOI: 10.1109/ICST.2013.13 (cit. a p. 33).

[39] Junjie Wang, Mingyang Li, Song Wang, Tim Menzies e Qing Wang. «Images don't lie: Duplicate crowdtesting reports detection with screenshot information». In: *Information and Software Technology* 110 (2019), pp. 139–155 (cit. a p. 33).

[40] Yan Chen, Maulishree Pandey, Jean Y Song, Walter S Lasecki e Steve Oney. «Improving crowd-supported gui testing with structural guidance». In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–13 (cit. a p. 34).

[41] Rahulkrishna Yandrapally, Andrea Stocco e Ali Mesbah. «Near-duplicate detection in web app model inference». In: *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 2020, pp. 186–197 (cit. a p. 41).

Ringraziamenti

A valle di questo lavoro vorrei ringraziare il professore Coppola e il dottore Fulcini che mi hanno seguito in questi mesi e hanno provveduto a fornirmi tutte le indicazioni necessarie per giungere ai risultati ottenuti. Ringrazio anche Andreas Bauer che ha dedicato del tempo per fornirmi i risultati di un lavoro precedente su Scout e risolvere alcune problematiche concettuali che si erano poste. Un grazie anche a tutti i compagni, amici e docenti che mi hanno accompagnato in questo percorso di studio.

Non ci sono parole che possano descrivere quanto la mia famiglia si sia impegnata per rendere tutto questo possibile, per appoggiarmi in qualsiasi decisione e sostenermi in tutti quei momenti di dubbio e difficoltà. Un grazie infinito a loro e alla mia ragazza Irene che ho conosciuto durante questo percorso e che mi ha dato la carica motivazionale necessaria prima di ogni esame.

E per ultima ma non per importanza ringrazio la città di Torino in cui ho trascorso quasi sette anni e non sarà dimenticata per tutte le esperienze che ha regalato.