FRAGILITÀ DEI TEST PER APPLICAZIONI WEB TEMPLATE BASED TRAMITE GENERAZIONE DI MUTAZIONI: SPERIMENTAZIONE

Un’analisi empirica tramite Mutation Testing su architetture Single Page Application

CANDIDATO - Simone Liberti **RELATORE - Porfirio Tramontana**

CONTENUTI

01. Contesto e Problema

Panoramica sul testing E2E: concetti chiave, vantaggi e limitazioni.

Il Testing End-to-End (E2E): Definizione e Caratteristiche

Simulazione dell'Utente Reale

Valida il comportamento dell'applicazione replicando fedelmente le interazioni di un utente, dall'inizio alla fine del suo percorso.

Controllo dell'Intero Flusso

Testa l’intero flusso operativo. Assicura che azioni sul front-end generino elaborazioni corrette nel back-end e nei database.

Integrazione dei Sottosistemi

Garantisce che tutti i sottosistemi comunichino correttamente tra loro, incluse le interrogazioni esatte a servizi di terze parti.

Automazione e Ripetibilità

Sostituisce l'interazione manuale con script automatizzati che pilotano direttamente il browser.

Dinamiche di Fallimento dei Test E2E

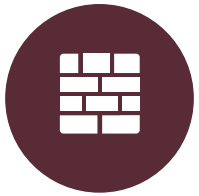
Obsolescenza

Si manifesta quando un test fallisce a causa di modifiche volute nella logica di business o nei requisiti funzionali. È un fallimento "giustificato": il test non è più allineato al sistema e deve essere aggiornato o rimosso.

Fragilità

Deriva dall'elevata sensibilità del test alle variazioni dell'interfaccia grafica. Modifiche marginali o estetiche del codice HTML causano il fallimento, generando falsi positivi e alti costi di manutenzione.

Il Mutation Testing: Paradigma Tradizionale vs Adattamento E2E



Il Paradigma Tradizionale

Inietta intenzionalmente difetti nel codice per creare "mutanti" e valutare l'efficacia dei test. Se il test fallisce, "uccide" il mutante dimostrando la sua validità; se il test passa, rivela una potenziale lacuna nel collaudo.

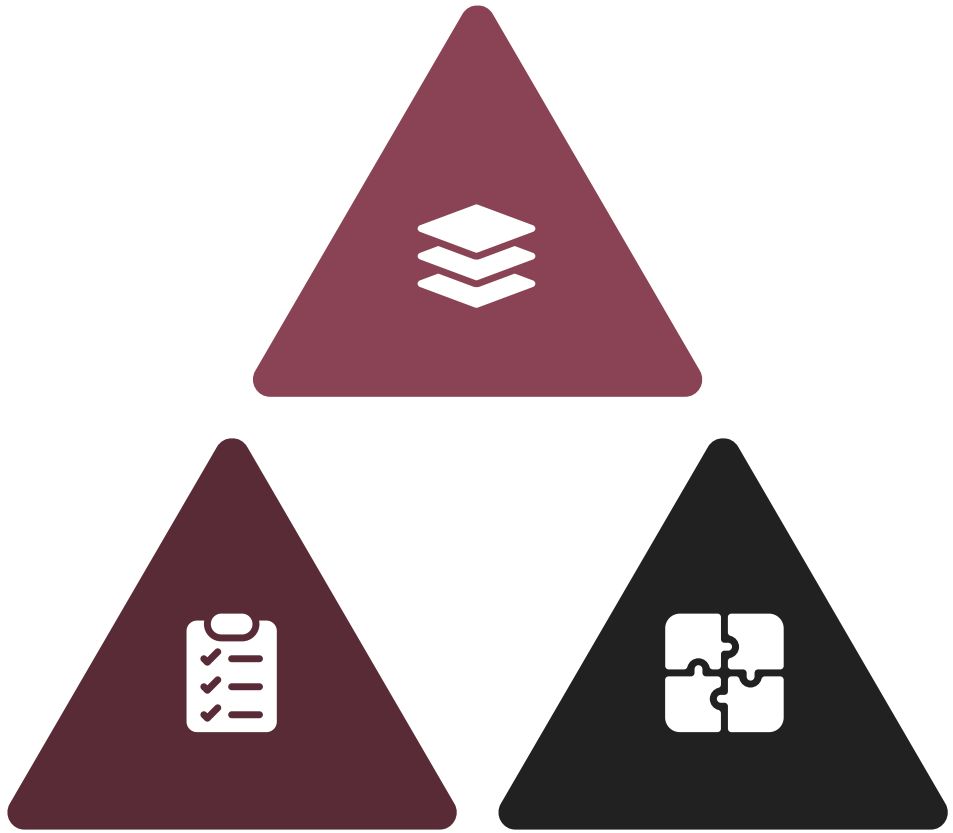
L'Adattamento E2E

Ribalta la logica applicando alterazioni strutturali all'interfaccia (DOM). L'obiettivo è misurare la resilienza dei locatori: se il test supera l'esecuzione, il locatore è robusto; un fallimento (test breakage) indica invece una fragilità.

02. Il Contributo Tecnologico

Descrizione e architettura dello strumento di mutazione e testing creato.

L'Architettura dello Strumento Proposto



Progettazione Modulare

Sviluppo in Java basato su Maven (moduli per locatori, iniezione, generazione, testing).

Analisi Statica Avanzata

Parsing del DOM effettuato staticamente tramite la libreria JSoup, garantendo velocità e isolamento.

Orchestrazione Agnostica

Un wrapper avvia l'app (Node.js), inietta i difetti, lancia i test e ripristina il clean state autonomamente.

Motore di Generazione

Indicizzazione Intelligente

Scansione preventiva dei template per supportare mutazioni inter-componente.

Configurazione Centralizzata

Flusso governato da un file JSON per massima flessibilità e riutilizzabilità.

Persistenza Sicura

Le alterazioni non sovrascrivono il file system originale, ma vengono tracciate e salvate strutturalmente in un database relazionale locale.

Tipologie di Mutazioni Applicate

01

Mutazioni sugli Attributi

- **[a]** Modifica del valore
- **[b]** Rimozione dell’attributo
- **[c]** Modifica dell’ID

02

Mutazioni sul Testo

- **[d]** Modifica del contenuto
- **[e]** Rimozione del contenuto

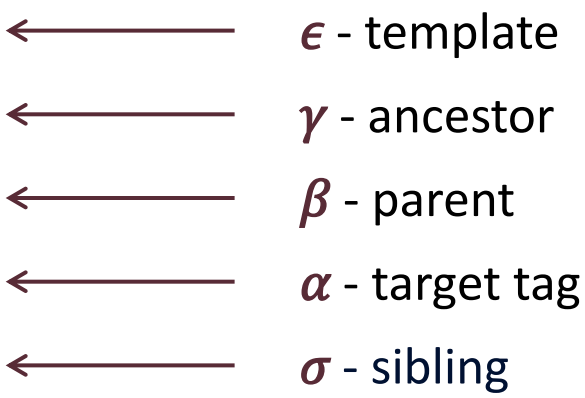
03

Mutazioni sui Tag

- **[f, g, h]** Spostamento nel DOM
- **[i]** Rimozione del tag
- **[j]** Modifica del tipo di tag
- **[k]** Inserimento di tag (sibling)

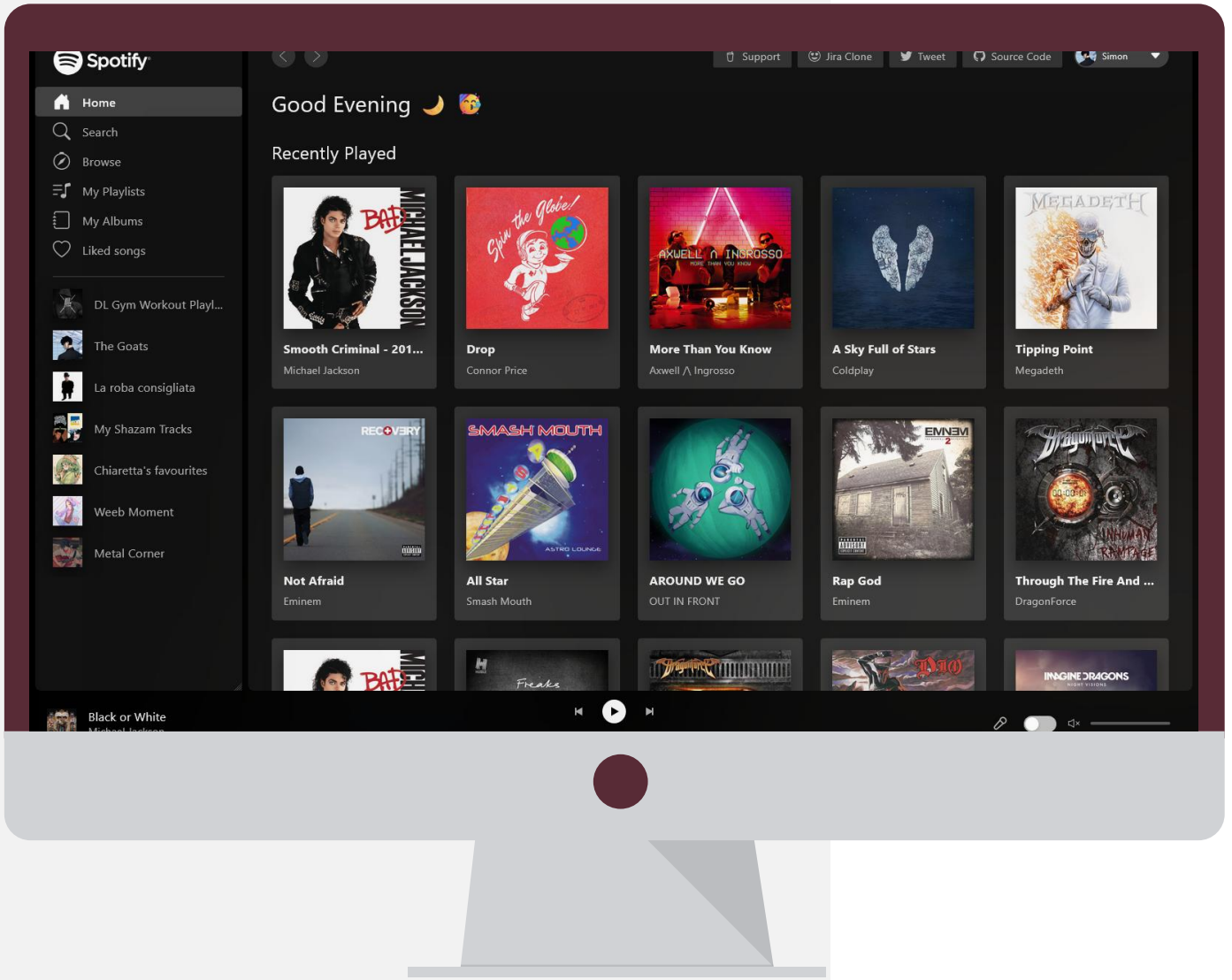
Gerarchia del DOM:
I Target delle Mutazioni

```
<template>
  <p>
    <div>
      <a onclick="go()">
      <a href="#">
    </div>
  </p>
</template>
```



03. Research Question e Sperimentazione

Domande di ricerca, definizione delle metriche e presentazione dell'ambiente di test e del tool sviluppato.



Oggetto dello Studio: Angular-Spotify

Rappresenta una vera e propria applicazione production-ready, caratterizzata da una complessità nettamente superiore rispetto ai comuni esempi didattici (toy applications). Implementa pattern di livello enterprise, come una gestione avanzata dello stato, interazioni asincrone con le API ufficiali di Spotify e una struttura del DOM particolarmente profonda e dinamica.

Strategie di Localizzazione Analizzate

Locatori Strutturali

- Absolute XPath
- Relative XPath

Strumenti Capture&Replay

- Selenium IDE
- Katalon Recorder

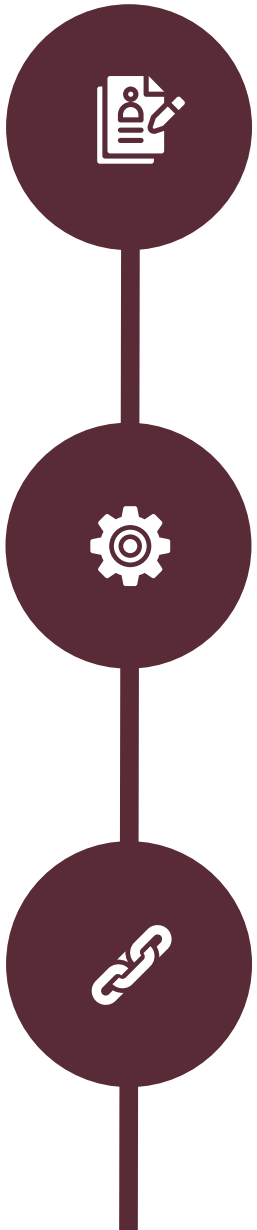
Euristiche Algoritmiche

- ROBULA
- ROBULA+

Strategia Intrusiva

- Approccio Hook-Based

Domande di Ricerca (RQs)



RQ1: Applicabilità

Qual è il grado di applicabilità pratica degli operatori di mutazione standard all'interno di un DOM reale e in che percentuale il framework AngularJS tollera tali alterazioni sintattiche senza incorrere in fallimenti di compilazione (Build Failure)?

RQ2: Robustezza

Quale strategia di localizzazione garantisce il minor tasso di fragilità (Fragility Rate) e il maggior tasso di sopravvivenza (Survival Rate) a fronte di modifiche strutturali ed estetiche utente?

RQ3: Correlazione

Esiste una correlazione tra specifiche tipologie di mutazione e il fallimento di determinate famiglie di locatori?

Le Metriche

RQ	Metrica	Descrizione	Formula
RQ1	Mutation Applicability Rate **(MAR)**	Percentuale di mutazioni teoriche effettivamente istanziate nel DOM	$MAR = \frac{N_{generatedOK}}{N_{attempted}} \times 100$
RQ1	Framework Tolerance Rate **(FTR)**	Percentuale di alterazioni che il compilatore AOT accetta come valide	$FTR = \left(1 - \frac{Not\_Compiled}{Total\_Applied}\right) \times 100$
RQ2	Success Rate **(SR)**	Percentuale di successo dei locatori a discapito della mutazione	$SR = \frac{Success}{Total - (NotCompiled + Obsolescent)} \times 100$
RQ2	Fragility Rate **(FR)**	Percentuale dei fallimenti dovuti alla rottura del locatore	$FR = \frac{Fragile}{Total - (NotCompiled + Obsolescent)} \times 100$
RQ3	Specific Fragility Rate **(SFR)**	Indice di fragilità diviso per variabili indipendenti (m – mutazioni ed elementi)	$SFR = \frac{Fragile_m}{Total_m - (NotCompiled_m + Obsolescent_m)} \times 100$

04. Analisi dei risultati e Conclusioni

Discussione dei risultati empirici, risposte alle domande di ricerca e riflessioni finali.

Esito RQ1: Applicabilità e Tolleranza

64.65%

Mutation Applicability Rate

891 mutanti generati su un pool teorico di 1378

64.76%

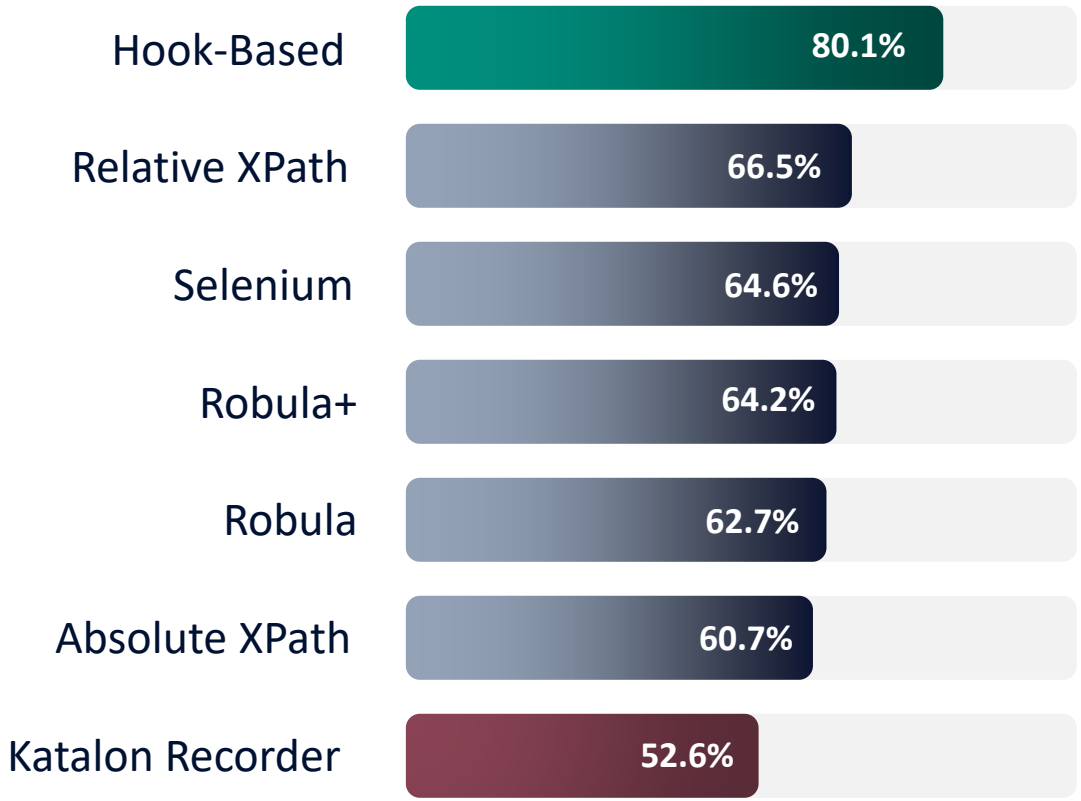
Framework Tolerance Rate

314 mutazioni non compilate su 891 generate

Risposta:

Lo strumento generatore si è dimostrato solido e affidabile, manipolando il DOM in modo sintatticamente valido. Tuttavia, l'architettura web a componenti sembra imporre un limite fisiologico, bloccando oltre un terzo delle mutazioni teoriche a causa dei vincoli strutturali del framework.

Esito RQ2: Robustezza dei Locatori



Risposta:

Nel caso di studio, la strategia Hook-Based risulta la più resiliente (**Survival Rate: 80.08%**), pur essendo un approccio **invasivo** limitato ai contesti in cui si ha il controllo del codice sorgente. Le euristiche non intrusive mostrano invece un'**efficacia variabile**, fortemente legata alla struttura del DOM: sorprendentemente, i locatori **Relative XPath** hanno registrato una tenuta superiore rispetto ad algoritmi come Robula+.

Esito RQ3: Correlazione

Specific Fragility Rate (SFR) valutato per tipologia di mutazione

Mutazione	Absolute	Hook	Katalon	Relative	Robula	Robula+	Selenium
a	88.33%	88.33%	88.33%	88.33%	88.33%	88.33%	90.00%
b	34.7%	28.26%	50.00%	32.61%	39.13%	39.13%	52.17%
d	11.67%	8.82%	50.00%	35.29%	35.29%	29.41%	26.47%
e	82.05%	82.05%	89.74%	87.18%	84.62%	84.62%	84.62%
f	28.30%	16.98%	49.06%	26.42%	37.74%	37.74%	35.85%
g	89.52%	88.46%	91.35%	88.57%	88.57%	88.57%	89.52%
h	85.86%	81.63%	88.78%	81.82%	88.00%	87.88%	85.00%
i	58.00%	45.54%	49.02%	41.00%	40.59%	38.61%	41.58%
j	69.49%	25.41%	55.93%	52.94%	48.31%	43.33%	47.11%
k	17.50%	4.17%	41.67%	20.83%	30.83%	33.33%	23.33%

Risposta:

L'analisi mostra una **chiara tendenza:** specifiche alterazioni del DOM mettono in crisi precise famiglie di locatori. L'affidamento a proprietà native e instabili è un fattore primario di fragilità. L'impiego di identificatori custom (**Hook-Based**) si conferma un'ottima strategia di mitigazione: **sebbene non esente da limiti**, riduce drasticamente l'accoppiamento tra gli script di collaudo e l'evoluzione estetica dell'applicazione.

Esito RQ3: Analisi per Tipo di Tag

Tipologia	Absolute	Hook	Katalon	Relative	Robula	Robula+	Selenium
alpha	58.55%	47.44%	65.09%	55.17%	58.50%	59.05%	57.26%
beta	67.87%	52.47%	67.87%	62.16%	60.36%	58.74%	64.13%
delta	49.02%	40.69%	65.20%	50.00%	58.82%	57.84%	52.45%
epsilon	63.48%	45.69%	58.12%	53.85%	52.14%	50.00%	51.69%

Specific Fragility Rate (SFR) valutato per tipologia di Tag

Nota: Assenza del tag 'gamma' dovuta alla mancanza di tale costrutto architetturale nel campione analizzato.

Minacce alla Validità

Construct Validity

L'approccio basato su analisi statica (JSoup) simula i refactoring sui template sorgente, ma potrebbe non catturare mutazioni dinamiche che avvengono esclusivamente a runtime tramite script complessi.

Internal Validity

Il processo di orchestrazione automatizzata (NpmWrapper e hot-reload) introduce potenziali minacce di timing e asincronicità, che possono generare fallimenti non imputabili al locatore ma alla sincronizzazione del framework Angular.

External Validity

Il metodo di misurazione basato sui log del TestRunnerEngine classifica i fallimenti in modo algoritmico. Una minaccia risiede nella distinzione automatica tra Fragilità e Obsolescenza, che abbiamo mitigato con una fase di validazione manuale dei mutanti generati.

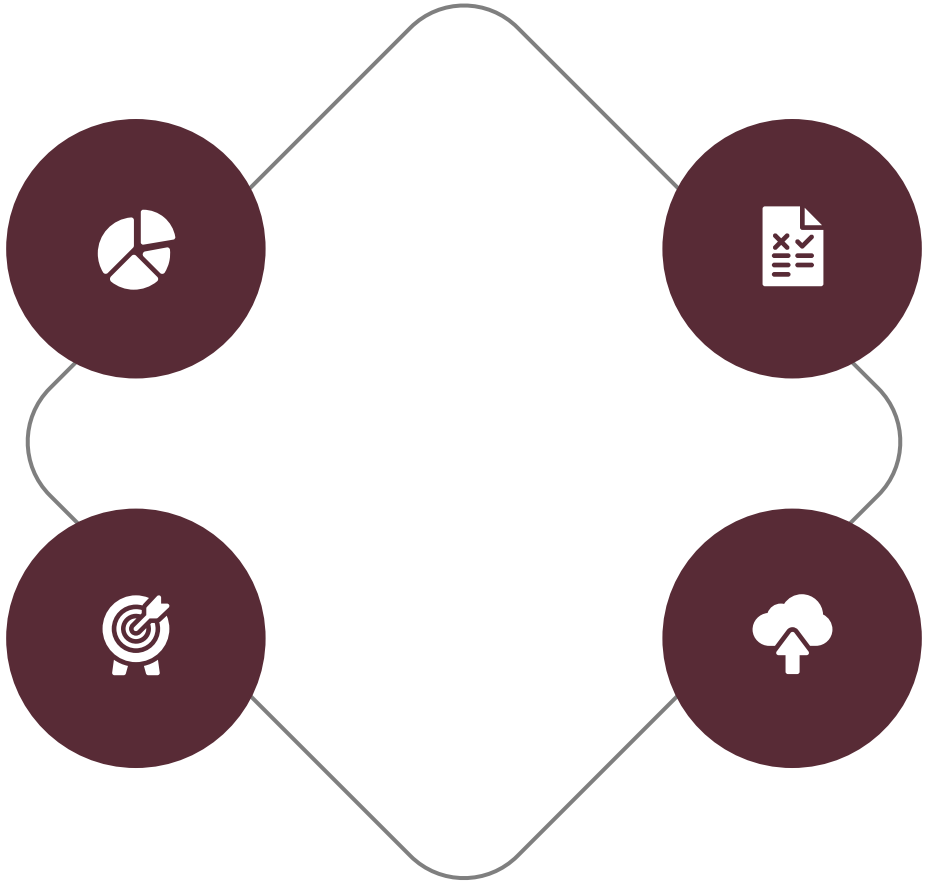
Conclusioni Finali

NUMERI DELLA SPERIMENTAZIONE

I dati raccolti su oltre **500 mutazioni valide** e un totale di più di **3000 esecuzioni empiriche** hanno confermato le ipotesi iniziali

DIRETTIVA CONSIGLIATA

I risultati suggeriscono che l'inserimento di metadati dedicati al collaudo sia un **compromesso altamente efficace** per disaccoppiare i test e ottimizzare i costi di manutenzione.



NESSUNA EURISTICA UNIVERSALE

Algoritmi e tool commerciali (ROBULA, Katalon) eccellono in aree specifiche ma **collassano in contesti dinamici** o a fronte di mutazioni gerarchiche

STANDARD HOOK-BASED

L'approccio Hook-Based emerge come il più resiliente (**Survival Rate > 80%**), tuttavia sconta un'elevata **invasività**: richiede pieno accesso al codice sorgente e un effort iniziale di configurazione.

Grazie per l’attenzione

Un’analisi empirica tramite Mutation Testing su architetture Single Page Application

CANDIDATO - Simone Liberti RELATORE - Porfirio Tramontana