



Σχολή Θετικών Επιστημών και Τεχνολογίας (ΣΘΕΤ)
Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα
(ΠΛΣ)

Διπλωματική Εργασία

Συγκριτική αξιολόγηση εργαλείων και τεχνολογιών αυτομάτου
ελέγχου λογισμικού για εφαρμογές διαδικτύου

Ελένη Χαραμή

Επιβλέπων καθηγητής: Ρηγόπουλος Γεώργιος

Πάτρα, Ιούλιος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του/της φοιτητή/φοιτήτριας («συγγραφέας/δημιουργός») που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.



Συγκριτική αξιολόγηση εργαλείων και τεχνολογιών αυτομάτου
ελέγχου λογισμικού για εφαρμογές διαδικτύου

Ελένη Χαραμή

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής/Επιβλέπουσα
Καθηγήτρια:

Ρηγόπουλος Γεώργιος

ΣΕΠ, Ελληνικό Ανοικτό Πανεπιστήμιο

Συν-Επιβλέπων Καθηγητής/Συν-
Επιβλέπουσα Καθηγήτρια:

Καστανιώτης Δημήτριος

ΣΕΠ, Ελληνικό Ανοικτό Πανεπιστήμιο

Πάτρα, Ιούλιος 2026

Περίληψη

Η παρούσα διπλωματική εργασία πραγματεύεται τη συγκριτική αξιολόγηση τριών εργαλείων αυτοματοποιημένου ελέγχου λογισμικού για εφαρμογές διαδικτύου: Selenium WebDriver, Playwright και Cypress. Η αξιολόγηση πραγματοποιήθηκε μέσω εμπειρικής μελέτης, κατά την οποία σχεδιάστηκαν και υλοποιήθηκαν εννέα σενάρια δοκιμών - τέσσερα βασισμένα σε τεχνικές σχεδιασμού δοκιμών κατά ISTQB (Equivalence Partitioning, Boundary Value Analysis, Decision Table, State Transition) και τέσσερα σενάρια αξιολόγησης απόδοσης - σε ιστότοπο ηλεκτρονικού εμπορίου (saucedemo.com).

Η σύγκριση διεξήχθη ως προς τρεις ποσοτικούς άξονες: ταχύτητα εκτέλεσης (1.000 επαναλήψεις τοπικά), συνολικό κόστος CI pipeline (5 διαδοχικές εκτελέσεις στο GitHub Actions), και πολυπλοκότητα κώδικα (γραμμές κώδικα ανά σενάριο). Συμπληρωματικά, τεκμηριώθηκαν ποιοτικά ευρήματα σχετικά με την εμπειρία ανάπτυξης (Developer Experience), τη λειτουργική υποδομή που απαιτεί κάθε εργαλείο, και τη συμπεριφορά σε διαφορετικά περιβάλλοντα εκτέλεσης.

Τα αποτελέσματα ανέδειξαν σημαντικές διαφορές μεταξύ των εργαλείων. Το Playwright ήταν το ταχύτερο σε τοπικό περιβάλλον (μέσος χρόνος 6,9s), ακολουθούμενο από το Selenium (8,7s) και το Cypress (25,4s). Ωστόσο, η κατάταξη αντιστράφηκε πλήρως στο CI, όπου το Selenium ήταν ταχύτερο (52,6s) και το Playwright αργότερο (76,8s), λόγω του κόστους εγκατάστασης browser. Σε επίπεδο κώδικα, το Selenium απαίτησε κατά μέσο όρο 3,1× περισσότερες γραμμές από το Playwright για ταυτόσημη λογική ελέγχου. Σε 1.000 εκτελέσεις, μόνο το Cypress εμφάνισε flakiness (0,3%), ενώ τα Playwright και Selenium πέτυχαν μηδενικό ποσοστό αποτυχίας.

Η εργασία καταλήγει στο συμπέρασμα ότι η επιλογή εργαλείου αυτοματισμού δεν μπορεί να βασιστεί σε μεμονωμένο κριτήριο ταχύτητας ή δημοφιλίας, αλλά πρέπει να λαμβάνει υπόψη το περιβάλλον ανάπτυξης, το αρχιτεκτονικό μοντέλο κάθε εργαλείου, και τις πραγματικές απαιτήσεις της ομάδας.

Λέξεις – Κλειδιά

αυτοματοποιημένος έλεγχος λογισμικού, selenium, playwright, cypress, ISTQB, CI/CD, web testing

Comparative evaluation of automated software testing tools and technologies for web applications

Eleni Charami

Abstract

This thesis presents a comparative evaluation of three automated software testing tools for web applications: Selenium WebDriver, Playwright, and Cypress. The evaluation was conducted through an empirical study in which nine test scenarios were designed and implemented - four based on ISTQB test design techniques (Equivalence Partitioning, Boundary Value Analysis, Decision Table, State Transition) and four performance benchmarking scenarios - against an e-commerce website (saucedemo.com).

The comparison was performed across three quantitative dimensions: execution speed (1,000 local iterations), total CI pipeline cost (5 consecutive GitHub Actions runs), and code complexity (lines of code per test scenario). Additionally, qualitative findings regarding Developer Experience, operational overhead per framework, and cross-environment behaviour were documented.

Results revealed significant differences between the tools. Playwright was the fastest locally (mean 6.9s), followed by Selenium (8.7s) and Cypress (25.4s). However, rankings fully inverted on CI, where Selenium was fastest (52.6s) and Playwright slowest (76.8s), due to browser installation overhead. In terms of code, Selenium required on average 3.1x more lines than Playwright for identical test logic. Across 1,000 runs, only Cypress exhibited flakiness (0.3%), while Playwright and Selenium achieved zero failure rates.

The thesis concludes that the selection of an automation tool cannot rely on a single criterion such as speed or popularity, but must account for the deployment environment, the architectural model of each tool, and the actual requirements of the development team.

Keywords

automated software testing, Selenium, Playwright, Cypress, ISTQB, CI/CD, benchmarking, web testing

Περιεχόμενα

Περίληψη.....	vi
Abstract.....	vii
Περιεχόμενα.....	viii
Κατάλογος Εικόνων / Σχημάτων.....	x
Κατάλογος Πινάκων.....	xi
Συντομογραφίες & Ακρωνύμια.....	xii
1. Εισαγωγή.....	1
1.1 Σκοπός της Εργασίας.....	2
1.2 Μεθοδολογική Προσέγγιση.....	2
1.3 Δομή της Εργασίας.....	3
2. Θεωρητικό Υπόβαθρο και Βιβλιογραφική Ανασκόπηση.....	3
2.1 Αυτοματοποιημένος Έλεγχος Λογισμικού.....	5
2.2 Τεχνικές Σχεδιασμού Δοκιμών (ISTQB).....	5
2.2.1 Equivalence Partitioning (EP).....	6
2.2.2 Boundary Value Analysis (BVA).....	6
2.2.3 Decision Table Testing.....	6
2.2.4 State Transition Testing.....	6
2.3 Αρχιτεκτονική Εργαλείων Αυτοματισμού.....	7
2.3.1 Selenium WebDriver.....	7
2.3.2 Playwright.....	8
2.3.3 Cypress.....	8
2.4 Εργαλεία Ελέγχου με Τεχνητή Νοημοσύνη (AI-Powered Testing).....	9
2.5 Βιβλιογραφική Ανασκόπηση Συγκριτικών Μελετών.....	10
2.6 Ερευνητικά Ερωτήματα.....	10
3. Μεθοδολογία.....	11
3.1 Σχεδιασμός Πειράματος.....	13
3.2 Περιβάλλον Δοκιμών.....	14
3.2.1 Τοπικό Περιβάλλον.....	14
3.2.2 Περιβάλλον CI.....	14
3.2.3 Εφαρμογή υπό Δοκιμή (AUT).....	15
3.3 Δομή Αποθετηρίου.....	15
3.4 Σενάρια Δοκιμών.....	16
3.4.1 Σουίτα Black-Box (εκτελείται μία φορά).....	16
3.4.2 Σουίτα Benchmarking (εκτελείται 1.000 φορές).....	17
3.4.3 Σχεδιαστική Σημείωση: Ρόλος των Τεχνικών ISTQB.....	17
3.5 Μηχανισμός Benchmark.....	18
3.5.1 Αυτοματοποιημένος Runner (benchmark.js).....	18
3.5.2 Οπτικοποίηση (charts.html).....	18
3.6 Στατιστική Μεθοδολογία.....	19
3.6.1 Μέγεθος Δείγματος.....	19
3.6.2 Μετρικές.....	19
3.6.3 Σχετική Τυπική Απόκλιση (RSD).....	20
3.7 Ανάλυση Γραμμών Κώδικα (LOC).....	20
3.8 Εργαλεία και Τεχνολογίες.....	21
3.9 Χρήση Εργαλείων Τεχνητής Νοημοσύνης στην Ανάπτυξη.....	21

4. Αποτελέσματα και Συζήτηση.....	22
4.1 Αποτελέσματα Τοπικού Benchmark (n=1.000).....	23
4.1.1 Ανάλυση Ταχύτητας.....	24
4.1.2 Ανάλυση Flakiness.....	25
4.2 Αποτελέσματα CI Pipeline (n=5).....	26
4.3 Ανάλυση Γραμμών Κώδικα (LOC).....	27
4.4 Σταθεροποίηση Selenium - Σωρευτικό Λειτουργικό Κόστος.....	29
4.5 Συζήτηση Αποτελεσμάτων.....	30
Απάντηση στα Ερευνητικά Ερωτήματα.....	30
5. Συμπεράσματα.....	30
5.1 Σύνοψη Κύριων Αποτελεσμάτων.....	31
5.2 Περιορισμοί.....	33
5.3 Προτάσεις για Μελλοντική Έρευνα.....	34
5.4 Συνεισφορά.....	35
Βιβλιογραφικές Αναφορές.....	36
Παράρτημα Α: Αποθετήριο Κώδικα.....	38
A.1 Selenium.....	38
A.2 Playwright.....	40
A.3 Cypress.....	41
A.4 benchmark.js.....	41
A.5 runner.js.....	46
A.6 benchmarking-runner.js.....	48
Παράρτημα Β: Ερευνητικό Ημερολόγιο.....	50
B.1 Βασικές Πληροφορίες Συστήματος.....	50
B.2 Χρονολόγιο Ανάπτυξης.....	50
B.3 Σύνοψη Κρίσιμων Συμβάντων.....	55
B.4 Χρόνοι CI Pipeline.....	58
B.5 Κατάλογος Ευρημάτων.....	58
Παράρτημα Γ: Οπτικοποίηση Αποτελεσμάτων.....	61

Κατάλογος Εικόνων / Σχημάτων

<u>Εικόνα Γ.1: Μέσος χρόνος εκτέλεσης ανά εργαλείο</u>	<u>61</u>
<u>Εικόνα Γ.2: Χρόνος εκτέλεσης για n=1000</u>	<u>61</u>
<u>Εικόνα Γ.3: Κατανομές-Min-Mean-Max</u>	<u>62</u>
<u>Εικόνα Γ.4: Ποσοστό Flakiness</u>	<u>62</u>
<u>Εικόνα Γ.5: Δομή Αποθετηρίου</u>	<u>63</u>

Κατάλογος Πινάκων

<u>Πίνακας 3.1: Τεχνολογίες που χρησιμοποιήθηκαν</u>	<u>21</u>
<u>Πίνακας 4.1: Αποτελέσματα τοπικού benchmark</u>	<u>24</u>
<u>Πίνακας 4.2: Λεπτομέρειες αποτυχιών Cypress</u>	<u>25</u>
<u>Πίνακας 4.3: Χρόνοι εκτέλεσης CI pipeline</u>	<u>26</u>
<u>Πίνακας 4.4: Γραμμές κώδικα ανά σενάριο</u>	<u>28</u>

Συντομογραφίες & Ακρωνύμια

AUT - Application Under Test (Εφαρμογή υπό Δοκιμή)

BVA - Boundary Value Analysis (Ανάλυση Οριακών Τιμών)

CDP - Chrome DevTools Protocol

CI/CD - Continuous Integration / Continuous Delivery (Συνεχής Ολοκλήρωση / Συνεχής Παράδοση)

CLT - Central Limit Theorem (Κεντρικό Οριακό Θεώρημα)

DOM - Document Object Model

DT - Decision Table (Πίνακας Αποφάσεων)

EP - Equivalence Partitioning (Διαχωρισμός Κλάσεων Ισοδυναμίας)

ISTQB - International Software Testing Qualifications Board

LLM - Large Language Model (Μεγάλο Γλωσσικό Μοντέλο)

LOC - Lines of Code (Γραμμές Κώδικα)

RSD - Relative Standard Deviation (Σχετική Τυπική Απόκλιση)

SPA - Single Page Application

ST - State Transition (Μετάβαση Καταστάσεων)

UC - Use Case (Σενάριο Χρήσης)

1. Εισαγωγή

Ο αυτοματοποιημένος έλεγχος λογισμικού αποτελεί αναπόσπαστο μέρος της σύγχρονης διαδικασίας ανάπτυξης εφαρμογών διαδικτύου. Η ταχεία υιοθέτηση μεθοδολογιών Agile και DevOps, σε συνδυασμό με τη διαρκή αύξηση της πολυπλοκότητας των web εφαρμογών και τη μείωση των κύκλων αποδέσμευσης (release cycles), καθιστούν τον χειροκίνητο έλεγχο ανεπαρκή τόσο σε ταχύτητα όσο και σε αξιοπιστία. Σε ένα περιβάλλον όπου οι εφαρμογές αποδεσμεύουν νέες εκδόσεις πολλαπλές φορές ημερησίως, η δυνατότητα αυτόματης εκτέλεσης δοκιμών σε κάθε commit - μέσω pipelines Συνεχούς Ολοκλήρωσης (CI) - δεν αποτελεί πλέον πολυτέλεια αλλά προϋπόθεση.

Η αγορά εργαλείων αυτοματισμού ελέγχου βρίσκεται σε φάση σημαντικής μεταβατικότητας. Το Selenium WebDriver, που κυριάρχησε για πάνω από μία δεκαετία ως το de facto πρότυπο, αντιμετωπίζει πλέον ανταγωνισμό από νεότερα εργαλεία όπως το Playwright (Microsoft, 2020) και το Cypress (2017), τα οποία υιοθετούν θεμελιωδώς διαφορετικές αρχιτεκτονικές προσεγγίσεις στην αλληλεπίδραση με τον browser. Το Selenium επικοινωνεί μέσω του πρωτοκόλλου WebDriver (W3C standard), το Playwright μέσω του Chrome DevTools Protocol (CDP), και το Cypress εκτελείται εντός του ίδιου του browser μέσω ενός Electron wrapper.

Παράλληλα, η εμφάνιση εργαλείων που αξιοποιούν τεχνικές τεχνητής νοημοσύνης (AI-powered testing) - όπως τα Testim, Mabl, Applitools και testRigor - εισάγει νέες δυνατότητες: αυτο-ίαση locators που αλλάζουν μεταξύ εκδόσεων, οπτική αναγνώριση αλλαγών στο UI, και αυτόνομη δημιουργία δοκιμών από περιγραφές σε φυσική γλώσσα. Η ταυτόχρονη αξιοποίηση μεγάλων γλωσσικών μοντέλων (LLMs) - όπως τα Claude και Gemini - ως βοηθημάτων ανάπτυξης αντιπροσωπεύει μια ακόμη διάσταση ενσωμάτωσης AI στον κύκλο ζωής του ελέγχου λογισμικού.

Δομή του Κεφαλαίου

Η ύλη του εισαγωγικού αυτού κεφαλαίου διαρθρώνεται σε τρεις κεντρικούς άξονες:

- Σκοπός της Εργασίας (Ενότητα 1.1): Προσδιορισμός της κεντρικής στοχοθεσίας της διατριβής, η οποία εστιάζει στην πολυδιάστατη, αντικειμενική αξιολόγηση των

Selenium, Playwright και Cypress, με στόχο την παροχή ενός data-driven οδηγού για τις ομάδες ανάπτυξης.

- Μεθοδολογική Προσέγγιση (Ενότητα 1.2): Συνοπτική παρουσίαση των σταδίων της πειραματικής διαδικασίας, από τον σχεδιασμό των 9 σεναρίων (ISTQB) και την εκτέλεση του benchmark των 1.000 επαναλήψεων, έως τις μετρήσεις σε περιβάλλον CI και την ποσοτική ανάλυση της πολυπλοκότητας του κώδικα.
- Δομή της Εργασίας (Ενότητα 1.3): Μια γενική επισκόπηση των πέντε κεφαλαίων που συνθέτουν το σώμα της παρούσας μελέτης, καθορίζοντας τη λογική αλληλουχία και τη ροή της ανάλυσης.

1.1 Σκοπός της Εργασίας

Σκοπός της εργασίας είναι η συγκριτική αξιολόγηση τριών εργαλείων αυτοματοποιημένου ελέγχου - Selenium WebDriver, Playwright και Cypress - μέσω ελεγχόμενου πειράματος σε πραγματικό περιβάλλον. Η αξιολόγηση πραγματοποιείται ως προς πολλαπλές διαστάσεις: ταχύτητα εκτέλεσης, αξιοπιστία (flakiness), πολυπλοκότητα κώδικα, απαιτούμενη υποδομή σταθεροποίησης, και συμπεριφορά σε περιβάλλον Συνεχούς Ολοκλήρωσης (CI).

Η εργασία δεν επιδιώκει να ανακηρύξει ένα εργαλείο ως «καλύτερο» - μια τέτοια δήλωση θα ήταν αβάσιμη χωρίς πλαίσιο χρήσης. Αντίθετα, στοχεύει στην παραγωγή τεκμηριωμένων δεδομένων που επιτρέπουν σε ομάδες ανάπτυξης να λάβουν ενημερωμένες αποφάσεις βάσει των δικών τους αναγκών, περιορισμών, και περιβαλλόντων εκτέλεσης.

1.2 Μεθοδολογική Προσέγγιση

Για την επίτευξη των παραπάνω στόχων, ακολουθήθηκε η εξής μεθοδολογία: α) Σχεδιασμός εννέα σεναρίων δοκιμών βασισμένων σε τεχνικές ISTQB Foundation Level (Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, State Transition Testing) και σεναρίων αξιολόγησης απόδοσης (Use Case Testing, ασύγχρονες μεταβάσεις DOM, μέτρηση χρόνων μέσω Performance API και Navigation Timing API). β) Υλοποίηση κάθε σεναρίου

και στα τρία εργαλεία, με ταυτόσημη λογική ελέγχου, ώστε οι διαφορές στα αποτελέσματα να αντικατοπτρίζουν τη συμπεριφορά του εργαλείου και όχι τη σχεδίαση του σεναρίου. γ) Εκτέλεση benchmark 1.000 επαναλήψεων τοπικά με πλήρη στατιστική ανάλυση (μέσος όρος, τυπική απόκλιση, διάμεσος, εύρος, ποσοστό αστοχίας). δ) Ανάλυση χρόνων CI pipeline από 5 διαδοχικές εκτελέσεις στο GitHub Actions. ε) Μέτρηση γραμμών κώδικα ανά σενάριο και ανά εργαλείο. στ) Τεκμηρίωση ποιοτικών ευρημάτων μέσω δομημένου ερευνητικού ημερολογίου (RESEARCH_LOG.md).

1.3 Δομή της Εργασίας

Η εργασία διαρθρώνεται σε πέντε κεφάλαια. Το Κεφάλαιο 1 (παρόν) εισάγει το αντικείμενο, τον σκοπό και τη μεθοδολογία. Το Κεφάλαιο 2 παρουσιάζει το θεωρητικό υπόβαθρο και τη βιβλιογραφική ανασκόπηση, συμπεριλαμβανομένων των τεχνικών ελέγχου ISTQB, της αρχιτεκτονικής των τριών εργαλείων, μιας επισκόπησης εργαλείων AI-powered testing, και των ερευνητικών ερωτημάτων. Το Κεφάλαιο 3 περιγράφει αναλυτικά τη μεθοδολογία - το περιβάλλον δοκιμών, τα σενάρια, τον μηχανισμό benchmark, τη στατιστική προσέγγιση, και τη χρήση AI εργαλείων. Το Κεφάλαιο 4 παρουσιάζει και αναλύει τα αποτελέσματα, απαντά στα ερευνητικά ερωτήματα, και συγκρίνει τα ευρήματα με τη βιβλιογραφία. Το Κεφάλαιο 5 συνοψίζει τα συμπεράσματα, αναλύει τους περιορισμούς, και προτείνει κατευθύνσεις για μελλοντική έρευνα.

2. Θεωρητικό Υπόβαθρο και Βιβλιογραφική Ανασκόπηση

Η ραγδαία εξέλιξη των σύγχρονων διαδικτυακών εφαρμογών και η ανάγκη για άμεση, αξιόπιστη παράδοση κώδικα έχουν καταστήσει τη διασφάλιση ποιότητας (Quality Assurance) κρίσιμο πυλώνα στην ανάπτυξη λογισμικού. Η παραδοσιακή χειροκίνητη δοκιμή, αν και απαραίτητη για την εξερευνητική φάση, αδυνατεί πλέον να συμβαδίσει με τους έντονους ρυθμούς των σύγχρονων ροών εργασίας. Στο πλαίσιο αυτό, ο αυτοματοποιημένος έλεγχος αναδεικνύεται σε αναγκαία συνθήκη για τη διατήρηση της σταθερότητας και της ποιότητας των εφαρμογών.

Ωστόσο, η επιτυχής μετάβαση στον αυτοματισμό δεν βασίζεται στην τυχαία επιλογή εργαλείων, αλλά στον συνδυασμό δομημένων θεωρητικών μεθοδολογιών και στη βαθιά κατανόηση της υποκείμενης τεχνολογίας. Η αποτελεσματικότητα μιας δοκιμής εξαρτάται εξίσου από τον σωστό σχεδιασμό των σεναρίων της (test design) και από την αρχιτεκτονική ικανότητα του εκάστοτε εργαλείου να αλληλεπιδρά αξιόπιστα με το περιβάλλον εκτέλεσης. Σκοπός του παρόντος κεφαλαίου είναι να θέσει το απαραίτητο θεωρητικό και τεχνικό υπόβαθρο που τεκμηριώνει την πειραματική διαδικασία της εργασίας, καταλήγοντας στον προσδιορισμό των κεντρικών ερευνητικών ζητούμενων.

Δομή του Κεφαλαίου

Η ανάλυση διαρθρώνεται σε πέντε θεματικούς άξονες:

- Αυτοματοποιημένος Έλεγχος Λογισμικού (Ενότητα 2.1): Ορισμός του αυτοματισμού, ανάλυση των πλεονεκτημάτων του και των ιδιαίτερων προκλήσεων που εισάγουν οι σύγχρονες web εφαρμογές (SPAs, δυναμικό DOM).
- Τεχνικές Σχεδιασμού Δοκιμών κατά ISTQB (Ενότητα 2.2): Παρουσίαση του διεθνούς πλαισίου black-box δοκιμών και ανάλυση των τεσσάρων τεχνικών (Equivalence Partitioning, Boundary Value Analysis, Decision Table, State Transition) που εφαρμόστηκαν στην εργασία.
- Αρχιτεκτονική Εργαλείων Αυτοματισμού (Ενότητα 2.3): Εμβάθυνση στις διαφορετικές αρχιτεκτονικές φιλοσοφίες των επικρατέστερων open-source εργαλείων της αγοράς (Selenium WebDriver, Playwright, Cypress).
- Εργαλεία Ελέγχου με Τεχνητή Νοημοσύνη (Ενότητα 2.4): Χαρτογράφηση των αναδυόμενων τεχνολογιών AI (Self-healing, Visual AI, LLMs) που μεταμορφώνουν τη συντήρηση και την ανάπτυξη δοκιμών.
- Ερευνητικά Ερωτήματα (Ενότητα 2.5): Συγκεκριμενοποίηση των ερευνητικών αξόνων της παρούσας διατριβής με βάση τα εντοπισμένα κενά της βιβλιογραφίας.

2.1 Αυτοματοποιημένος Έλεγχος Λογισμικού

Ο αυτοματοποιημένος έλεγχος λογισμικού αναφέρεται στη χρήση εξειδικευμένων εργαλείων για την εκτέλεση δοκιμών χωρίς ανθρώπινη παρέμβαση. Σε αντίθεση με τον χειροκίνητο έλεγχο, ο αυτοματοποιημένος έλεγχος επιτρέπει την επαναλαμβανόμενη εκτέλεση δοκιμών, τη μείωση του ανθρώπινου λάθους, και την ενσωμάτωση στη ροή εργασίας Συνεχούς Ολοκλήρωσης / Συνεχούς Παράδοσης (CI/CD) (Garousi, Felderer, Kuhrmann & Herkiloğlu, 2017).

Σύμφωνα με τους Dustin κ.ά. (2009), τα κύρια πλεονεκτήματα του αυτοματοποιημένου ελέγχου περιλαμβάνουν: ταχύτητα εκτέλεσης, επαναληψιμότητα, κάλυψη μεγάλου εύρους σεναρίων, και δυνατότητα ενσωμάτωσης σε CI pipelines. Ωστόσο, η αυτοματοποίηση δεν αποτελεί πανάκεια. Απαιτεί σημαντική αρχική επένδυση χρόνου και εξειδίκευσης, συντήρηση κώδικα δοκιμών σε κάθε αλλαγή του UI, και βαθιά κατανόηση τόσο του εργαλείου αυτοματισμού όσο και της εφαρμογής υπό δοκιμή.

Στον τομέα των web εφαρμογών, ο αυτοματοποιημένος έλεγχος αντιμετωπίζει επιπρόσθετες προκλήσεις που δεν υπάρχουν σε desktop ή mobile εφαρμογές. Η ασύγχρονη φύση των Single Page Applications (SPA), οι δυναμικές αλλαγές DOM, οι μεταβάσεις σελίδας χωρίς πλήρη ανανέωση (client-side routing), και η ποικιλία browsers δημιουργούν ένα απαιτητικό τοπίο για τα εργαλεία αυτοματισμού. Η ικανότητα ενός εργαλείου να διαχειρίζεται αξιόπιστα αυτές τις συνθήκες - μέσω μηχανισμών αυτόματης αναμονής (auto-wait) και επαναλαμβανόμενης εκτέλεσης assertions (auto-retry) - αποτελεί κρίσιμο παράγοντα στην επιλογή εργαλείου.

2.2 Τεχνικές Σχεδιασμού Δοκιμών (ISTQB)

Το International Software Testing Qualifications Board (ISTQB) ορίζει ένα πλαίσιο τεχνικών σχεδιασμού δοκιμών στο Foundation Level Syllabus (ISTQB, 2024). Οι τεχνικές αυτές ανήκουν στην κατηγορία black-box testing: ο ελεγκτής σχεδιάζει δοκιμές βάσει της εξωτερικής συμπεριφοράς του συστήματος, χωρίς γνώση της εσωτερικής δομής του κώδικα. Στην παρούσα εργασία εφαρμόστηκαν τέσσερις τεχνικές:

2.2.1 Equivalence Partitioning (EP)

Η τεχνική EP διαιρεί τον χώρο εισόδου σε κλάσεις ισοδυναμίας (equivalence classes), υποθέτοντας ότι οποιαδήποτε τιμή εντός μιας κλάσης θα προκαλέσει ταυτόσημη συμπεριφορά στο σύστημα. Αντί να δοκιμαστούν εξαντλητικά όλες οι πιθανές τιμές - πρακτικά αδύνατο στα περισσότερα συστήματα - επιλέγεται ένας αντιπρόσωπος από κάθε κλάση (ISTQB, 2024). Η τεχνική αυτή μειώνει δραστικά τον αριθμό των απαιτούμενων δοκιμών διατηρώντας παράλληλα ικανοποιητική κάλυψη. Στην παρούσα εργασία, η EP εφαρμόστηκε στη λειτουργία ταξινόμησης προϊόντων κατά τιμή (σενάριο EP-01).

2.2.2 Boundary Value Analysis (BVA)

Η BVA αποτελεί συμπλήρωμα της EP, εστιάζοντας στις οριακές τιμές κάθε κλάσης ισοδυναμίας. Η εμπειρία δείχνει ότι τα σφάλματα λογισμικού συγκεντρώνονται δυσανάλογα στα σύνορα μεταξύ κλάσεων - εκεί όπου η συμπεριφορά αλλάζει. Δοκιμάζονται η ελάχιστη τιμή, η μέγιστη τιμή, και οι αμέσως γειτονικές τιμές (ISTQB, 2024). Στην εργασία, η BVA εφαρμόστηκε στον μέγιστο αριθμό προϊόντων στο καλάθι: η μέγιστη τιμή 6 (boundary) αντιπροσωπεύει το άνω σύνορο (σενάριο BVA-01).

2.2.3 Decision Table Testing

Ο πίνακας αποφάσεων (Decision Table) αντιστοιχίζει συστηματικά συνδυασμούς συνθηκών εισόδου σε αναμενόμενα αποτελέσματα. Κάθε στήλη του πίνακα αντιπροσωπεύει έναν κανόνα - δηλαδή έναν μοναδικό συνδυασμό True/False τιμών για κάθε συνθήκη - και η αντίστοιχη γραμμή αποτελέσματος καθορίζει την αναμενόμενη συμπεριφορά. Η τεχνική είναι ιδιαίτερα αποτελεσματική σε φόρμες εισαγωγής δεδομένων με πολλαπλές συνθήκες επικύρωσης (ISTQB, 2024). Στην εργασία, ο κανόνας R2 (πρώτο όνομα: παρόν, επώνυμο: παρόν, ταχυδρομικός κώδικας: απών) εφαρμόστηκε στη φόρμα checkout (σενάριο DT-R2).

2.2.4 State Transition Testing

Η τεχνική STT μοντελοποιεί τη συμπεριφορά ενός συστήματος ως μηχανή πεπερασμένων καταστάσεων (Finite State Machine). Κάθε κατάσταση αντιπροσωπεύει μια συγκεκριμένη

κατάσταση του συστήματος, και κάθε μετάβαση (transition) ενεργοποιείται από ένα γεγονός (event). Η δοκιμή επαληθεύει ότι κάθε μετάβαση παράγει την αναμενόμενη νέα κατάσταση (ISTQB, 2024). Στην εργασία, η STT εφαρμόστηκε στο καλάθι αγορών: η μετάβαση empty→full→empty ελέγχθηκε μέσω προσθήκης και αφαίρεσης του μοναδικού προϊόντος (σενάριο ST-01).

2.3 Αρχιτεκτονική Εργαλείων Αυτοματισμού

Η κατανόηση της αρχιτεκτονικής κάθε εργαλείου αποτελεί τη βάση για την ερμηνεία κάθε εμπειρικού αποτελέσματος. Τα τρία εργαλεία που εξετάζονται στην παρούσα εργασία ανήκουν σε τρεις θεμελιωδώς διαφορετικές αρχιτεκτονικές κατηγορίες.

2.3.1 Selenium WebDriver

Το Selenium αποτελεί το παλαιότερο και πλέον διαδεδομένο εργαλείο αυτοματισμού ελέγχου web εφαρμογών. Η πρώτη του έκδοση κυκλοφόρησε το 2004 (αρχικά ως Selenium RC), ενώ η σύγχρονη αρχιτεκτονική WebDriver εισήχθη το 2011 και τυποποιήθηκε ως W3C standard. Η αρχιτεκτονική βασίζεται σε ένα μοντέλο τριών επιπέδων: ο κώδικας δοκιμής αποστέλλει εντολές μέσω HTTP requests σε έναν browser-specific driver (π.χ. ChromeDriver), ο οποίος τις μεταφράζει σε native browser commands (Selenium Project, 2024).

Αυτή η αρχιτεκτονική προσφέρει εξαιρετική ευελιξία - υποστηρίζει πολλαπλές γλώσσες προγραμματισμού (Java, JavaScript, Python, C#, Ruby) και browsers (Chrome, Firefox, Safari, Edge) - αλλά εισάγει μετρήσιμο latency σε κάθε εντολή λόγω του HTTP round-trip. Επιπλέον, το Selenium είναι βιβλιοθήκη (library) και όχι πλαίσιο (framework): δεν περιλαμβάνει ενσωματωμένο test runner, assertion library, ή reporters, αφήνοντας τον χρήστη να τα επιλέξει ή να τα κατασκευάσει (Colantonio, 2021). Κρίσιμη απουσία αποτελεί ο μηχανισμός αυτόματης επανάληψης (auto-retry): κάθε assertion εκτελείται μία μόνο φορά, και αν το element δεν είναι ακόμα διαθέσιμο, η δοκιμή αποτυγχάνει αμέσως.

2.3.2 Playwright

Το Playwright αναπτύχθηκε από τη Microsoft και κυκλοφόρησε το 2020. Η ομάδα ανάπτυξής του - που περιλαμβάνει πρώην μηχανικούς του Puppeteer - σχεδίασε ένα εργαλείο ειδικά για σύγχρονες web εφαρμογές. Η αρχιτεκτονική βασίζεται στο Chrome DevTools Protocol (CDP), ένα πρωτόκολλο αμφίδρομης επικοινωνίας μέσω WebSocket. Σε αντίθεση με το HTTP-based WebDriver, το CDP επιτρέπει ταυτόχρονη αποστολή εντολών και λήψη γεγονότων, εξαλείφοντας τον overhead του request-response κύκλου.

Το Playwright παρέχει πλήρες framework «out of the box»: ενσωματωμένο test runner, auto-wait μηχανισμό (κάθε ενέργεια περιμένει αυτόματα μέχρι το element να γίνει actionable), parallel execution μέσω worker processes, trace viewer για debugging, και reporters σε πολλαπλές μορφές (Playwright Documentation, 2024). Βασικό χαρακτηριστικό είναι η δυνατότητα auto-retry σε assertions: αν μια αναμενόμενη κατάσταση δεν ισχύει ακόμα, το Playwright επαναλαμβάνει αυτόματα μέχρι να ικανοποιηθεί ή να εκπνεύσει το timeout.

2.3.3 Cypress

Το Cypress κυκλοφόρησε το 2017 και υιοθέτησε μια ριζικά διαφορετική αρχιτεκτονική: εκτελείται μέσα στον ίδιο τον browser (in-browser execution model) μέσω ενός Electron wrapper ή ενσωματωμένου Chrome instance. Αυτό σημαίνει ότι ο κώδικας δοκιμής τρέχει στο ίδιο execution context με την εφαρμογή - έχει άμεση πρόσβαση στο DOM, στα network requests, στο τοπικό state, και στα cookies χωρίς ενδιάμεσο πρωτόκολλο (Cypress Documentation, 2024).

Αυτή η αρχιτεκτονική προσφέρει μοναδικές δυνατότητες debugging: Time Travel (δυνατότητα «γυρίσματος στον χρόνο» μέσω snapshots σε κάθε βήμα), αυτόματα screenshots σε κάθε εντολή, και network stubbing χωρίς εξωτερικά εργαλεία. Ωστόσο, εισάγει σημαντικό startup overhead: η εκκίνηση του Electron runner και η επεξεργασία του εσωτερικού command queue απαιτούν χρόνο που δεν υπάρχει στα out-of-process εργαλεία. Επίσης, η αρχιτεκτονική αυτή περιορίζει τη γλωσσική υποστήριξη αποκλειστικά σε JavaScript/TypeScript.

2.4 Εργαλεία Ελέγχου με Τεχνητή Νοημοσύνη (AI-Powered Testing)

Παράλληλα με τα παραδοσιακά εργαλεία αυτοματισμού, έχει αναδυθεί μια νέα κατηγορία εργαλείων που αξιοποιούν τεχνικές τεχνητής νοημοσύνης και μηχανικής μάθησης στη διαδικασία ελέγχου λογισμικού. Τα εργαλεία αυτά μπορούν να κατηγοριοποιηθούν σε τέσσερις κατηγορίες:

Αυτο-ίαση (Self-healing): Εργαλεία όπως τα Testim και Mabl χρησιμοποιούν αλγορίθμους μηχανικής μάθησης για την αυτόματη ενημέρωση locators όταν αλλάζει η δομή του UI. Αντί η δοκιμή να αποτύχει επειδή ένα κουμπί άλλαξε CSS class, το εργαλείο αναγνωρίζει τη σημασιολογία του element και ανανεώνει τον locator (Wang κ.ά., 2021).

Οπτικός Έλεγχος (Visual AI): Το Applitools Eyes χρησιμοποιεί αλγορίθμους computer vision για τη σύγκριση screenshots μεταξύ εκδόσεων, εντοπίζοντας οπτικές αλλαγές που δεν θα ανιχνεύονταν από παραδοσιακό DOM-based testing (Applitools, 2024).

Αυτόνομη Δημιουργία Δοκιμών: Εργαλεία όπως το testRigor επιτρέπουν τη δημιουργία δοκιμών σε φυσική γλώσσα («κάνε login και πρόσθεσε ένα προϊόν στο καλάθι»), μεταφράζοντας αυτόματα τις περιγραφές σε εκτελέσιμο κώδικα (testRigor, 2024).

AI ως Βοήθημα Ανάπτυξης: Τα μεγάλα γλωσσικά μοντέλα (LLMs) όπως τα Claude (Anthropic) και Gemini (Google) χρησιμοποιούνται ως βοηθήματα κατά τη συγγραφή κώδικα δοκιμών, τον εντοπισμό σφαλμάτων, και την τεκμηρίωση. Η χρήση αυτή - που εφαρμόστηκε και στην παρούσα εργασία - αντιπροσωπεύει μια διαφορετική διάσταση: το AI δεν εκτελεί τις δοκιμές, αλλά επιταχύνει τη διαδικασία ανάπτυξής τους.

Τα AI-powered εργαλεία προσφέρουν πλεονεκτήματα σε συγκεκριμένες περιπτώσεις χρήσης - ιδιαίτερα σε εφαρμογές με συχνές αλλαγές UI, όπου το κόστος συντήρησης locators είναι δυσανάλογα υψηλό. Ωστόσο, παρουσιάζουν σημαντικούς περιορισμούς: εξάρτηση από εξωτερικές υπηρεσίες cloud, λιγότερο granular έλεγχο σε σύγκριση με χειρόγραφες δοκιμές, κλειστό κώδικα, και σημαντικά υψηλότερο κόστος αδειοδότησης (Leotta κ.ά., 2026). Η παρούσα εργασία εστιάζει στα παραδοσιακά εργαλεία ανοιχτού κώδικα, καθώς αυτά αποτελούν τη βάση πάνω στην οποία οικοδομούνται τα AI-powered εργαλεία.

2.5 Βιβλιογραφική Ανασκόπηση Συγκριτικών Μελετών

Ο Tymoshchuk (2025) πραγματοποίησε συγκριτική ανάλυση Selenium, Playwright και Cypress με διαδοχική εκτέλεση σεναρίων διαφορετικής πολυπλοκότητας, καταγράφοντας μέσο χρόνο εκτέλεσης, τυπική απόκλιση και συντελεστή μεταβλητότητας. Τα αποτελέσματά του υποδεικνύουν ότι το Playwright παρουσιάζει την καλύτερη απόδοση σε δυναμικές web εφαρμογές. Ωστόσο, η μελέτη βασίστηκε σε περιορισμένο αριθμό εκτελέσεων και δεν περιελάμβανε cross-environment ανάλυση (τοπικό vs CI).

Ο Brahmhatt (2023) αξιολόγησε πλαίσια αυτοματισμού για ιστότοπο ηλεκτρονικού εμπορίου στο πλαίσιο μεταπτυχιακής εργασίας, παρέχοντας χρήσιμες ποιοτικές παρατηρήσεις σχετικά με την εμπειρία εγκατάστασης και ανάπτυξης, αλλά χωρίς ποσοτικά benchmarks μεγάλης κλίμακας.

Οι Almabruk κ.ά. (2025) συνέκριναν την αξιοπιστία (uptime) Selenium και Playwright σε 24ωρη παρακολούθηση, εντοπίζοντας ελαφρώς υψηλότερη σταθερότητα στο Playwright. Η μελέτη δεν περιελάμβανε το Cypress και επικεντρώθηκε σε reliability μέσω uptime, όχι σε execution speed.

Οι Garousi και Felderer (2017) ανέδειξαν το χάσμα μεταξύ ακαδημαϊκής έρευνας και βιομηχανικής πρακτικής στον έλεγχο λογισμικού, τονίζοντας ότι η ακαδημαϊκή κοινότητα δεν παρέχει επαρκώς αναπαραγώγιμα πειραματικά δεδομένα - ένα κενό που η παρούσα εργασία επιχειρεί να καλύψει μέσω δημόσιου αποθετηρίου κώδικα.

Οι Leotta κ.ά. (2026) διεξήγαγαν εκτεταμένη έρευνα (survey) σε επαγγελματίες χρήστες Selenium, τεκμηριώνοντας ότι το Playwright αποτελεί πλέον την πιο δημοφιλή εναλλακτική, ενώ τα βασικά προβλήματα του Selenium (flakiness, brittleness) παραμένουν.

Κοινό χαρακτηριστικό αυτών των μελετών είναι η απουσία cross-environment ανάλυσης: η σύγκριση πραγματοποιείται σε ένα μόνο περιβάλλον (τοπικό ή CI). Η παρούσα εργασία αντιμετωπίζει αυτόν τον περιορισμό.

2.6 Ερευνητικά Ερωτήματα

Με βάση την ανασκόπηση της βιβλιογραφίας και τα εντοπισμένα κενά, η παρούσα εργασία επιχειρεί να απαντήσει στα ακόλουθα ερευνητικά ερωτήματα:

EE1: Πώς συγκρίνονται τα Selenium, Playwright και Cypress ως προς ταχύτητα εκτέλεσης δοκιμών σε μεγάλο δείγμα επαναλήψεων ($n=1000$);

EE2: Η κατάταξη των εργαλείων ως προς ταχύτητα παραμένει σταθερή μεταξύ τοπικού περιβάλλοντος και περιβάλλοντος CI;

EE3: Πόσο διαφέρει η πολυπλοκότητα κώδικα μεταξύ των εργαλείων για ταυτόσημη λογική ελέγχου, και πώς κλιμακώνεται ανάλογα με τον τύπο σεναρίου;

EE4: Ποιες αρχιτεκτονικές διαφορές εξηγούν τις παρατηρούμενες αποκλίσεις σε ταχύτητα, αξιοπιστία, και πολυπλοκότητα;

EE5: Ποιο εργαλείο παρουσιάζει τη μεγαλύτερη αξιοπιστία (χαμηλότερο ποσοστό flakiness) σε παρατεταμένη εκτέλεση;

3. Μεθοδολογία

Η μετάβαση από τη θεωρητική ανάλυση των εργαλείων αυτοματισμού στην πρακτική τους αξιολόγηση απαιτεί τη διαμόρφωση ενός αυστηρού, αντικειμενικού και επαναλήψιμου πειραματικού πλαισίου. Καθώς η υφιστάμενη βιβλιογραφία συχνά περιορίζεται σε ποιοτικές εκτιμήσεις ή μεμονωμένες μετρήσεις μικρού δείγματος, η παρούσα εργασία υιοθετεί μια αμιγώς ποσοτική, εμπειρική προσέγγιση. Μέσω ενός ελεγχόμενου πειράματος μεγάλης κλίμακας, επιχειρείται η απομόνωση κάθε εξωτερικής μεταβλητής, διασφαλίζοντας ότι οι καταγεγραμμένες αποκλίσεις αντανακλούν αποκλειστικά τις εγγενείς αρχιτεκτονικές ιδιαιτερότητες των εξεταζόμενων εργαλείων.

Στο παρόν κεφάλαιο περιγράφεται αναλυτικά ο σχεδιασμός του πειράματος και οι προδιαγραφές των περιβαλλόντων εκτέλεσης (τοπικού και CI). Παρουσιάζεται η αρχιτεκτονική οργάνωσης του κώδικα, η σκοπιμότητα πίσω από την επιλογή της δοκιμαζόμενης React εφαρμογής, καθώς και η ανατομία των εννέα (9) σεναρίων δοκιμών που αναπτύχθηκαν. Επιπλέον, αναλύονται οι αυτοματοποιημένοι μηχανισμοί συλλογής, οπτικοποίησης και στατιστικής επεξεργασίας των δεδομένων, η μεθοδολογία ποσοτικοποίησης του κώδικα, καθώς και το τεχνολογικό και συμβουλευτικό πλαίσιο (AI tools) που υποστήριξε την ανάπτυξη της έρευνας.

Δομή του Κεφαλαίου

Η ύλη του μεθοδολογικού αυτού κεφαλαίου διαρθρώνεται σε εννέα κεντρικούς άξονες:

- Σχεδιασμός Πειράματος (Ενότητα 3.1): Παρουσίαση των βασικών αρχών του ελεγχόμενου πειράματος και των πέντε (5) αξιολογικών αξόνων (ταχύτητα, αξιοπιστία, κόστος CI, πολυπλοκότητα, υποδομή) γύρω από τους οποίους δομήθηκε η έρευνα.
- Περιβάλλον Δοκιμών (Ενότητα 3.2): Αναλυτική καταγραφή των τεχνικών χαρακτηριστικών του τοπικού υλικού (hardware) και των δικτυακών παραμέτρων, της αρχιτεκτονικής απομόνωσης των runners στο GitHub Actions (CI), καθώς και των κριτηρίων επιλογής της εφαρμογής υπό δοκιμή (Application Under Test - AUT).
- Δομή Αποθετηρίου (Ενότητα 3.3): Ανάλυση της αρχιτεκτονικής σταθμών εργασίας (station-based architecture), η οποία εφαρμόστηκε στο αποθετήριο του κώδικα για τη διασφάλιση της πλήρους απομόνωσης μεταξύ των Selenium, Playwright και Cypress.
- Σενάρια Δοκιμών (Ενότητα 3.4): Ανατομία των 9 σεναρίων, χωρισμένων στη σουίτα Black-Box (βάσει των τεχνικών ISTQB: EP, BVA, DT, STT) και στη σουίτα Benchmarking (ροές χρήστη, ασύγχρονες μεταβάσεις DOM και μετρήσεις Performance API).
- Μηχανισμός Benchmark (Ενότητα 3.5): Περιγραφή του custom αυτοματοποιημένου runner (benchmark.js) που αναπτύχθηκε για τη σειριακή εκτέλεση των δοκιμών, καθώς και του συστήματος οπτικοποίησης των αποτελεσμάτων μέσω της Chart.js.
- Στατιστική Μεθοδολογία (Ενότητα 3.6): Τεκμηρίωση της επάρκειας του μεγέθους του δείγματος ($n = 1.000$) βάσει του Κεντρικού Οριακού Θεωρήματος και ανάλυση των μετρικών αξιολόγησης (Μέσος, Τυπική Απόκλιση, Διάμεσος, Flakiness Rate και Σχετική Τυπική Απόκλιση - RSD).
- Ανάλυση Γραμμών Κώδικα - LOC (Ενότητα 3.7): Προσδιορισμός των κριτηρίων καταμέτρησης του μεγέθους του κώδικα με στόχο την ποσοτικοποίηση του αρχιτεκτονικού overhead και του boilerplate κάθε εργαλείου.

- Εργαλεία και Τεχνολογίες (Ενότητα 3.8): Συγκεντρωτική παράθεση των εκδόσεων και του ρόλου όλων των τεχνολογιών και των browsers που συνθέτουν την πειραματική υποδομή.
- Χρήση Εργαλείων Τεχνητής Νοημοσύνης (Ενότητα 3.9): Παρουσίαση του πλαισίου αξιοποίησης των LLMs ως συμβουλευτικών βοηθημάτων κατά τη διαδικασία αποσφαλμάτωσης, βελτιστοποίησης και τεκμηρίωσης.

3.1 Σχεδιασμός Πειράματος

Η μεθοδολογική προσέγγιση της εργασίας βασίζεται σε εμπειρική σύγκριση τριών εργαλείων αυτοματοποιημένου ελέγχου - Selenium WebDriver, Playwright και Cypress - μέσω ελεγχόμενου πειράματος. Για να εξασφαλιστεί η εγκυρότητα των αποτελεσμάτων, η σύγκριση πραγματοποιήθηκε υπό ελεγχόμενες συνθήκες: τα ίδια σενάρια δοκιμών υλοποιήθηκαν και στα τρία εργαλεία, εκτελέστηκαν στο ίδιο hardware, κατά την ίδια χρονική περίοδο, και ενάντια στην ίδια εφαρμογή υπό δοκιμή. Με αυτόν τον τρόπο, οποιαδήποτε μετρούμενη διαφορά στα αποτελέσματα αντικατοπτρίζει τη συμπεριφορά του εργαλείου και όχι εξωτερικούς παράγοντες.

Η αξιολόγηση δομήθηκε γύρω από πέντε αξιολογικούς άξονες:

1. Ταχύτητα εκτέλεσης (μέσω 1.000 τοπικών επαναλήψεων)
2. Αξιοπιστία (ποσοστό flakiness σε 1.000 εκτελέσεις)
3. Κόστος CI pipeline (5 εκτελέσεις στο GitHub Actions)
4. Πολυπλοκότητα κώδικα (γραμμές κώδικα ανά σενάριο)
5. Λειτουργική υποδομή (αριθμός απαιτούμενων ρυθμίσεων/workarounds)

3.2 Περιβάλλον Δοκιμών

3.2.1 Τοπικό Περιβάλλον

Οι τοπικές δοκιμές εκτελέστηκαν σε σύστημα με τα ακόλουθα χαρακτηριστικά: λειτουργικό σύστημα Windows 11, επεξεργαστής AMD Ryzen 5 5600H with Radeon Graphics, μνήμη RAM 16GB, και Node.js v20.20.2. Η επιλογή Node.js v20 (αντί της πλέον πρόσφατης v24) ήταν σκόπιμη, ώστε το τοπικό περιβάλλον να αντιστοιχεί ακριβώς στην έκδοση που χρησιμοποιείται στο CI (GitHub Actions, Node.js 20 LTS). Η απόφαση αυτή καταγράφηκε στο ερευνητικό ημερολόγιο ως Finding #6 και ελήφθη αφού μια αρχική ασυμβατότητα εκδόσεων (v24 τοπικά vs v20 στο CI) προκάλεσε αδιάγνωστες αποκλίσεις στα αποτελέσματα.

Η σύνδεση δικτύου ήταν οικιακό Wi-Fi με μετρημένη ταχύτητα download 117,36 Mbps, upload 20,87 Mbps, και ping 7ms. Η ταχύτητα δικτύου καταγράφηκε διότι η εφαρμογή υπό δοκιμή (saucedemo.com) φιλοξενείται εξωτερικά και η απόδοση δικτύου αποτελεί σταθερή μεταβλητή σε όλες τις εκτελέσεις.

3.2.2 Περιβάλλον CI

Οι δοκιμές Συνεχούς Ολοκλήρωσης (CI) εκτελέστηκαν στο GitHub Actions με runner ubuntu-latest και Node.js 20. Το CI workflow (.github/workflows/ci.yml) ορίστηκε με τρία παράλληλα jobs - ένα ανά εργαλείο - ώστε κάθε εργαλείο να εκτελείται σε ανεξάρτητο, πρωτογενές περιβάλλον (fresh runner) χωρίς αλληλεπιδράσεις μεταξύ τους. Η εγκατάσταση browser γινόταν ξεχωριστά σε κάθε job: το Playwright κατέβαζε τα δικά του binaries (~437MB με caching), το Cypress χρησιμοποιούσε τον ενσωματωμένο Electron browser, και το Selenium αξιοποιούσε τον Chrome που εγκαθίστατο μέσω browser-actions/setup-chrome@v2.

Η τεχνική αυτή απομόνωσης (isolation) εξασφαλίζει ότι τα CI αποτελέσματα αντιπροσωπεύουν τη «from-scratch» εμπειρία κάθε εργαλείου, συμπεριλαμβανομένου του κόστους εγκατάστασης browser.

3.2.3 Εφαρμογή υπό Δοκιμή (AUT)

Ως εφαρμογή υπό δοκιμή επιλέχθηκε το saucedemo.com, μια πλατφόρμα ηλεκτρονικού εμπορίου σχεδιασμένη ειδικά για αυτοματοποιημένες δοκιμές από τη Sauce Labs. Η εφαρμογή υλοποιεί πλήρη ροή αγοράς: σύνδεση χρήστη, κατάλογο προϊόντων, καλάθι αγορών, φόρμα checkout, και επιβεβαίωση παραγγελίας. Η συγκεκριμένη επιλογή βασίστηκε σε μια σειρά από κρίσιμους παράγοντες που διασφαλίζουν την αξιοπιστία και την εγκυρότητα της πειραματικής διαδικασίας.

Αρχικά, η επαναληψιμότητα των δοκιμών διασφαλίζεται καθώς η εφαρμογή παραμένει αμετάβλητη μεταξύ των εκτελέσεων, εξαλείφοντας τον κίνδυνο εμφάνισης αστάθειας από απρόβλεπτες ενημερώσεις λογισμικού (deployment updates). Παράλληλα, εξασφαλίζεται σταθερή διαθεσιμότητα, καθώς το περιβάλλον δεν επηρεάζεται από περιόδους συντήρησης (maintenance windows), διακοπές λειτουργίας (downtime) ή περιορισμούς στη συχνότητα αιτημάτων (rate limiting).

Επιπλέον, η χρήση ειδικών αναγνωριστικών (data-test attributes) εγγυάται μια προβλέψιμη και γνωστή συμπεριφορά, προσφέροντας αξιόπιστους εντοπιστές (locators) και ελαχιστοποιώντας τις αποτυχίες που οφείλονται σε δυναμικές αλλαγές στη δομή του DOM. Τέλος, παρά τον ελεγχόμενο χαρακτήρα της, η εφαρμογή διακρίνεται για τον ρεαλισμό της, καθώς αποτελεί μια πλήρη εφαρμογή μοναδικής σελίδας (React SPA) με πραγματικές ασύγχρονες μεταβάσεις. Το γεγονός αυτό επιτρέπει την αποτελεσματική αξιολόγηση του τρόπου με τον οποίο το εκάστοτε εργαλείο διαχειρίζεται τις μεταβολές κατάσταση (DOM transitions) σε συνθήκες πραγματικής λειτουργίας.

Κατά τη διάρκεια της εργασίας, εντοπίστηκε ένα πραγματικό ελάττωμα στην εφαρμογή (αποδοχή whitespace ως έγκυρη είσοδο στη φόρμα login) μέσω σεναρίου Error Guessing (EG-01). Το σενάριο αυτό τεκμηριώθηκε ως defect report και απομονώθηκε στον φάκελο _sandbox/ του αποθετηρίου.

3.3 Δομή Αποθετηρίου

Ο κώδικας οργανώθηκε σε δομή απομόνωσης ανά εργαλείο (station-based architecture, πβ. Εικόνα Γ.5: Δομή Αποθετηρίου). Κάθε station περιέχει 9 αρχεία δοκιμών (1 login + 4 black-
Διπλωματική Εργασία

box + 4 benchmarking), σύνολο 27 αρχεία. Η δομή αυτή εξασφαλίζει πλήρη απομόνωση: αλλαγές στο Selenium δεν επηρεάζουν τα Playwright ή Cypress, και κάθε station μπορεί να εκτελεστεί ανεξάρτητα.

3.4 Σενάρια Δοκιμών

Σχεδιάστηκαν 9 σενάρια δοκιμών, χωρισμένα σε δύο σουίτες. Κάθε σενάριο υλοποιήθηκε και στα τρία εργαλεία με ταυτόσημη λογική ελέγχου, ώστε οι διαφορές στα αποτελέσματα να αντικατοπτρίζουν τη συμπεριφορά του εργαλείου.

3.4.1 Σουίτα Black-Box (εκτελείται μία φορά)

EP-01 (Equivalence Partitioning): Ταξινόμηση προϊόντων κατά τιμή «χαμηλή→υψηλή» μέσω του dropdown φίλτρου και επαλήθευση ότι η πρώτη εμφανιζόμενη τιμή είναι \$7.99 (φθηνότερο) και η τελευταία \$49.99 (ακριβότερο). Η τεχνική EP εφαρμόστηκε στο πεδίο ταξινόμησης: η κλάση «χαμηλή-υψηλή» αντιπροσωπεύει μία αύξουσα κλάση ισοδυναμίας με αναμενόμενη σειρά τιμών.

BVA-01 (Boundary Value Analysis): Προσθήκη και των 6 διαθέσιμων προϊόντων στο καλάθι (μέγιστο boundary value) και επαλήθευση ότι το badge αριθμός εμφανίζει «6». Η τεχνική BVA εφαρμόστηκε στο πλήθος προϊόντων: η μέγιστη τιμή (6 από 6) αντιπροσωπεύει το άνω όριο. Αυτό είναι το σενάριο με τη μεγαλύτερη αλληλεπίδραση UI (6 κουμπιά add-to-cart), γεγονός που αναδεικνύει τη διαφορά μεταξύ εργαλείων στη διαχείριση πολλαπλών γεγονότων click.

DT-R2 (Decision Table, Rule 2): Πλοήγηση μέχρι τη φόρμα checkout, συμπλήρωση πρώτου ονόματος και επωνύμου (παρόντα), αλλά παράλειψη ταχυδρομικού κώδικα (απών), και επαλήθευση εμφάνισης μηνύματος σφάλματος «Error: Postal Code is required». Ο κανόνας R2 αντιπροσωπεύει μία γραμμή του πλήρους πίνακα αποφάσεων: [T, T, F] → σφάλμα.

ST-01 (State Transition): Προσθήκη ενός προϊόντος στο καλάθι (μετάβαση: κενό → ένα αντικείμενο), πλοήγηση στο καλάθι, αφαίρεση του μοναδικού αντικειμένου (μετάβαση: ένα

αντικείμενο → κενό), και επαλήθευση ότι το badge εξαφανίζεται εντελώς. Η τεχνική STT εφαρμόστηκε σε δύο μεταβάσεις: empty→full→empty.

3.4.2 Σουίτα Benchmarking (εκτελείται 1.000 φορές)

UC-01 (Use Case): Πλήρης ροή αγοράς (end-to-end): login → επιλογή προϊόντος → add to cart → checkout → συμπλήρωση φόρμας → finish → επαλήθευση μηνύματος επιτυχίας «Thank you for your order!». Αυτό είναι το πιο ρεαλιστικό σενάριο και αντιπροσωπεύει μια τυπική ροή χρήστη σε e-commerce εφαρμογή.

ASYNC-01 (Ασύγχρονες Μεταβάσεις DOM): Τρεις διαδοχικές μεταβάσεις σελίδας - inventory → product detail page → πίσω στο inventory - χωρίς login. Σχεδιάστηκε ειδικά για τη μέτρηση ευστάθειας κάθε εργαλείου σε ασύγχρονες μεταβάσεις DOM. Στο saucedemo.com (React SPA), οι μεταβάσεις δεν φορτώνουν νέα σελίδα αλλά ανανεώνουν το DOM δυναμικά, αποτελώντας stress test για τους auto-wait μηχανισμούς.

PERF-01 (Performance - Login Timing): Μέτρηση χρόνου από το click στο κουμπί login μέχρι τη φόρτωση της σελίδας inventory. Στο Cypress χρησιμοποιήθηκε performance.mark(), ενώ στα Playwright και Selenium χρησιμοποιήθηκε Date.now(). Η διαφορά μεθόδου ήταν αναγκαία λόγω αρχιτεκτονικών περιορισμών: το Cypress εκτελείται εντός του browser (πρόσβαση στο Performance API), ενώ τα Playwright/Selenium εκτελούνται εκτός (χρήση εξωτερικού χρονομέτρου).

PERF-02 (Performance - Navigation Timing API): Μέτρηση χρόνου απόκρισης δικτύου μέσω Navigation Timing API (performance.timing.responseEnd – performance.timing.requestStart) σε όλα τα εργαλεία. Σε αντίθεση με το PERF-01 που μετρά end-user perception, το PERF-02 μετρά network-level latency, παρέχοντας μια δεύτερη διάσταση μέτρησης χρόνων.

3.4.3 Σχεδιαστική Σημείωση: Ρόλος των Τεχνικών ISTQB

Οι τεχνικές ISTQB (EP, BVA, DT, STT) εφαρμόστηκαν με συγκεκριμένο σκοπό: τη δημιουργία ρεαλιστικών σεναρίων δοκιμών που αντικατοπτρίζουν τυπικές πρακτικές QA, όχι την εξαντλητική λειτουργική κάλυψη της εφαρμογής. Κάθε τεχνική εφαρμόστηκε με έναν αντιπρόσωπο ανά κλάση (π.χ. μία κλάση ισοδυναμίας στο EP-01, μία γραμμή πίνακα

αποφάσεων στο DT-R2), διότι ο σκοπός δεν ήταν η ανεύρεση ελαττωμάτων στο saucedemo.com - η εφαρμογή είναι σταθερή και γνωστής συμπεριφοράς - αλλά η σύγκριση εργαλείων.

Η χρήση τεχνικών ISTQB αντί αυθαίρετων σεναρίων παρέχει τρία πλεονεκτήματα: πρώτον, δομημένη σχεδίαση δοκιμών βάσει αναγνωρισμένου διεθνούς πλαισίου, δεύτερον, ποικιλία αλληλεπιδράσεων UI (από απλή ταξινόμηση dropdown έως πολλαπλά click events σε 6 κουμπιά), και τρίτον, αναπαραγωγικότητα - κάθε σενάριο μπορεί να υλοποιηθεί από τρίτο ερευνητή βάσει της περιγραφής του.

3.5 Μηχανισμός Benchmark

3.5.1 Αυτοματοποιημένος Runner (benchmark.js)

Αναπτύχθηκε αυτοματοποιημένος runner στη ρίζα του αποθετηρίου (benchmark.js), ο οποίος εκτελεί τη σουίτα benchmarking επαναλαμβανόμενα και παράγει στατιστικά δεδομένα. Ο runner δέχεται ως παράμετρο τον αριθμό επαναλήψεων (π.χ. node benchmark.js 1000) και σε κάθε επανάληψη εκτελεί διαδοχικά:

1. npm run run-cypress (Cypress benchmarking suite)
2. npx playwright test (Playwright benchmarking suite)
3. node selenium-station/benchmarking-runner.js (Selenium benchmarking suite)

Σε κάθε εκτέλεση καταγράφονται: αριθμός εκτέλεσης, timestamp, κατάσταση (passed/failed), και χρόνος σε milliseconds ανά εργαλείο. Τα δεδομένα αποθηκεύονται σε δύο μορφές: results/benchmark-data.json (ακατέργαστα δεδομένα) και results/benchmark-data.js (JavaScript module για χρήση στο charts.html).

3.5.2 Οπτικοποίηση (charts.html)

Δημιουργήθηκε σελίδα οπτικοποίησης (results/charts.html) με χρήση της βιβλιοθήκης Chart.js, η οποία φορτώνει αυτόματα τα δεδομένα benchmark και αποδίδει τέσσερα

γραφήματα: μέσος χρόνος εκτέλεσης ανά εργαλείο (πβ. Παράρτημα Γ, Εικόνα Γ.1) χρονοσειρά 1.000 εκτελέσεων (πβ. Παράρτημα Γ, Εικόνα Γ.2), κατανομή min/mean/max (πβ. Παράρτημα Γ, Εικόνα Γ.3), και ποσοστό flakiness (πβ. Παράρτημα Γ, Εικόνα Γ.4). Τα χρώματα ακολουθούν τα brand colors των εργαλείων: Cypress #04C38E (πράσινο), Playwright #E2574C (κόκκινο), Selenium #F8C300 (κίτρινο).

3.6 Στατιστική Μεθοδολογία

3.6.1 Μέγεθος Δείγματος

Πραγματοποιήθηκαν 1.000 διαδοχικές εκτελέσεις μέσω του benchmark runner. Η εκτέλεση ξεκίνησε στις 19 Μαΐου 2026 στις 15:49 και ολοκληρώθηκε στις 20 Μαΐου στις 03:12, με συνολική διάρκεια περίπου 11 ωρών.

Ο αριθμός $n=1.000$ υπερβαίνει κατά πολύ τον ελάχιστο αριθμό $n \geq 30$ που απαιτεί το Κεντρικό Οριακό Θεώρημα (CLT) για στατιστική εγκυρότητα, παρέχοντας εξαιρετικά στενά διαστήματα εμπιστοσύνης. Επιπλέον, ο μεγάλος αριθμός επιτρέπει τον εντοπισμό σπάνιων φαινομένων (rare events) όπως clusters αποτυχιών, τα οποία δεν θα ήταν ορατά σε μικρότερο δείγμα.

3.6.2 Μετρικές

Για κάθε εργαλείο υπολογίστηκαν:

Η στατιστική ανάλυση της συμπεριφοράς και της απόδοσης του εκάστοτε εργαλείου βασίστηκε στον υπολογισμό συγκεκριμένων δεικτών κεντρικής τάσης, διασποράς και αξιοπιστίας.

Αρχικά, ο μέσος αριθμητικός χρησιμοποιήθηκε ως μέτρο κεντρικής τάσης για τον προσδιορισμό του αναμενόμενου χρόνου σε μια τυπική εκτέλεση, ενώ η τυπική απόκλιση επιστρατεύθηκε ως μέτρο διασποράς, όπου οι χαμηλές τιμές της υποδηλώνουν μια σταθερή και προβλέψιμη συμπεριφορά. Παράλληλα, εξετάστηκε η διάμεσος, η οποία προσφέρει ένα μέτρο κεντρικής τάσης ανθεκτικό στην επίδραση ακραίων τιμών (outliers), αποτυπώνοντας

με μεγαλύτερη ακρίβεια τα δεδομένα σε περιπτώσεις ασύμμετρων κατανομών. Το εύρος των παρατηρήσεων καθορίστηκε από την ελάχιστη (Min) και τη μέγιστη (Max) τιμή, οι οποίες είναι κρίσιμες για την κατανόηση της ακραίας συμπεριφοράς του συστήματος στη χειρότερη δυνατή περίπτωση (worst-case scenario). Τέλος, η αξιοπιστία του κάθε εργαλείου ποσοτικοποιήθηκε μέσω του ποσοστού αστοχίας (flakiness rate), το οποίο ορίζεται ως ο λόγος των αποτυχημένων εκτελέσεων προς το συνολικό πλήθος των δοκιμών, εκπεφρασμένος επί τοις εκατό.

3.6.3 Σχετική Τυπική Απόκλιση (RSD)

Για τη σύγκριση συνέπειας μεταξύ εργαλείων με πολύ διαφορετικούς μέσους χρόνους (6,9s vs 25,4s), χρησιμοποιήθηκε η σχετική τυπική απόκλιση ($RSD = \text{StdDev} / \text{Mean} \times 100\%$). Η RSD εκφράζει τη μεταβλητότητα ως ποσοστό του μέσου, επιτρέποντας δίκαιη σύγκριση: ένα εργαλείο με StdDev 3.375ms σε mean 25.440ms (RSD 13,3%) είναι πιο σταθερό από ένα με StdDev 1.361ms σε mean 6.923ms (RSD 19,7%), παρότι η απόλυτη τιμή StdDev είναι μικρότερη στο δεύτερο.

3.7 Ανάλυση Γραμμών Κώδικα (LOC)

Πραγματοποιήθηκε μέτρηση γραμμών κώδικα ανά αρχείο δοκιμής. Μετρήθηκαν μόνο οι ουσιαστικές γραμμές (εξαιρούνται κενές γραμμές και σχόλια). Συγκεκριμένα, η μέτρηση συμπεριλαμβάνει: δηλώσεις import/require, wrappers describe/test/it, τη λογική ελέγχου (assertions, navigation, click events), τον boilerplate κώδικα (driver setup, Chrome flags, try/catch, retry loops) και την κλήση εκκίνησης (π.χ. runUC01()). Εξαιρούνται αποκλειστικά κενές γραμμές και γραμμές σχολίων. Ο σκοπός δεν ήταν η εύρεση ενός απόλυτου μέτρου «ποιότητας» - η αρχή «λιγότερες γραμμές = καλύτερο» δεν ισχύει πάντα - αλλά η ποσοτικοποίηση του αρχιτεκτονικού overhead: πόσο κώδικα πρέπει να γράψει ο χρήστης για ταυτόσημη λογική ελέγχου;

Η μέτρηση περιλαμβάνει τόσο τη λογική ελέγχου (assertions, navigation) όσο και τον «boilerplate» κώδικα (driver setup, Chrome flags, try/catch, retry loops) που απαιτεί κάθε εργαλείο.

3.8 Εργαλεία και Τεχνολογίες

Στον κάτωθι Πίνακα (Πίνακας 3.1) παρουσιάζονται συγκεντρωτικά οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν.

Τεχνολογία	Έκδοση	Ρόλος
Node.js	v20.20.2	Περιβάλλον εκτέλεσης
Selenium WebDriver	4.18.0	Εργαλείο αυτοματισμού
Playwright	1.42.0	Εργαλείο αυτοματισμού
Cypress	15.12.0	Εργαλείο αυτοματισμού
GitHub Actions	ubuntu-latest	CI pipeline
Chart.js	4.4.0	Οπτικοποίηση αποτελεσμάτων
Chrome	Τελευταία σταθερή	Browser (Selenium/Cypress)
Chromium	Bundled by PW	Browser (Playwright)

Πίνακας 3.1: Τεχνολογίες που χρησιμοποιήθηκαν

3.9 Χρήση Εργαλείων Τεχνητής Νοημοσύνης στην Ανάπτυξη

Κατά τη διάρκεια της ανάπτυξης χρησιμοποιήθηκαν τα AI conversational tools Claude (Anthropic) και Gemini (Google) ως βοηθήματα. Η χρήση τους ήταν αυστηρά συμβουλευτική και εντοπίζεται σε τρεις κατηγορίες:

Εντοπισμός σφαλμάτων: Όταν τα σενάρια αποτύγχαναν (ιδιαίτερα σε CI), η έξοδος τερματικού μοιραζόταν με τα AI tools για διάγνωση. Παράδειγμα: η ανακάλυψη ότι το Selenium σε headless mode χρησιμοποιεί viewport 800×600 (αντί του αναμενόμενου 1920×1080) εντοπίστηκε μέσω Claude.

Βελτιστοποίηση κώδικα: Προτάσεις για Chrome flags, retry patterns, και JavaScript click injection αξιολογήθηκαν και δοκιμάστηκαν. Δεν εφαρμόστηκαν όλες: η πρόταση για χρήση explicit sleep() αντί retry loop απορρίφθηκε (καταγράφεται στο RESEARCH_LOG ως anti-pattern).

Τεκμηρίωση: Δομή ερευνητικού ημερολογίου, ακαδημαϊκή τεκμηρίωση ευρημάτων, και σύνταξη κειμένων.

Κάθε πρόταση αξιολογήθηκε εμπειρικά - εκτελέστηκε τοπικά και στο CI - και εγκρίθηκε ή απορρίφθηκε βάσει αποτελεσμάτων. Η φοιτήτρια φέρει πλήρη ευθύνη για τον τελικό κώδικα.

4. Αποτελέσματα και Συζήτηση

Στο παρόν κεφάλαιο παρουσιάζονται και αναλύονται τα εμπειρικά δεδομένα που προέκυψαν από την πειραματική διαδικασία. Η αξιολόγηση των εργαλείων βασίζεται στα ποσοτικά αποτελέσματα των 1.000 τοπικών επαναλήψεων και στις μετρήσεις από την εκτέλεση σε περιβάλλον Συνεχούς Ολοκλήρωσης (CI).

Η σύγκριση πραγματοποιείται με βάση την ταχύτητα εκτέλεσης, τη συνέπεια των χρόνων απόκρισης, την εμφάνιση ασταθών δοκιμών (flakiness), την πολυπλοκότητα του κώδικα (Lines of Code) και το λειτουργικό κόστος σταθεροποίησης. Τα δεδομένα αυτά συσχετίζονται με την ιδιαίτερη αρχιτεκτονική του κάθε εργαλείου, επιτρέποντας την τεκμηριωμένη απάντηση στα ερευνητικά ερωτήματα της διατριβής.

Δομή του Κεφαλαίου

Η ύλη του κεφαλαίου των αποτελεσμάτων διαρθρώνεται στους ακόλουθους κεντρικούς άξονες:

- Αποτελέσματα Τοπικού Benchmark (Ενότητα 4.1): Παρουσίαση των συγκεντρωτικών στατιστικών στοιχείων των 1.000 εκτελέσεων.
- Ανάλυση Ταχύτητας (Υποενότητα 4.1.1): Αξιολόγηση του μέσου χρόνου εκτέλεσης και ερμηνεία του startup overhead της in-browser αρχιτεκτονικής.

- Ανάλυση Συνέπειας (Υποενότητα 4.1.2): Εξέταση της μεταβλητότητας των εργαλείων με βάση τη Σχετική Τυπική Απόκλιση (RSD).
- Ανάλυση Flakiness (Υποενότητα 4.1.3): Καταγραφή και ποιοτική ανάλυση των consecutive clusters αποτυχίας που παρατηρήθηκαν.
- Αποτελέσματα CI Pipeline (Ενότητα 4.2): Παρουσίαση των μετρήσεων από το περιβάλλον του GitHub Actions και ανάλυση του φαινομένου της πλήρους αντιστροφής της κατάταξης απόδοσης μεταξύ τοπικού περιβάλλοντος και CI (Υποενότητα 4.2.1).
- Ανάλυση Γραμμών Κώδικα - LOC (Ενότητα 4.3): Ποσοτική σύγκριση του μεγέθους του κώδικα ανά σενάριο και ανά κατηγορία (Black-box έναντι Benchmarking), αναδεικνύοντας πώς η πολυπλοκότητα κλιμακώνεται ανάλογα με τον αριθμό των UI αλληλεπιδράσεων.
- Σταθεροποίηση Selenium - Σωρευτικό Λειτουργικό Κόστος (Ενότητα 4.4): Ποιοτική και ποσοτική χαρτογράφηση των 13 συγκεκριμένων ρυθμίσεων και custom workarounds (Chrome flags, retry loops, JS injections) που απαιτήθηκαν για την εξίσωση της αξιοπιστίας του Selenium με τα σύγχρονα frameworks.
- Συζήτηση Αποτελεσμάτων (Ενότητα 4.5): Συστηματική και data-driven απάντηση στα πέντε αρχικά ερευνητικά ερωτήματα (EE1 έως EE5) της εργασίας, με βάση τα συνδυαστικά ευρήματα όλων των προηγούμενων ενοτήτων.

4.1 Αποτελέσματα Τοπικού Benchmark (n=1.000)

Πραγματοποιήθηκαν 1.000 διαδοχικές εκτελέσεις της σουίτας benchmarking σε τοπικό περιβάλλον. Η εκτέλεση ξεκίνησε στις 19/05/2026 στις 15:49 και ολοκληρώθηκε στις 20/05/2026 στις 03:12, με συνολική διάρκεια περίπου 11 ωρών. Τα αποτελέσματα παρουσιάζονται συγκεντρωτικά στον κάτωθι πίνακα (Πίνακας 4.1) :

	Cypress	Playwright	Selenium
Mean	25.440ms	6.923ms	8.683ms
StdDev	3.375ms	1.361ms	648ms
RSD	13,3%	19,7%	7,5%
Min	19.146ms	5.544ms	7.314ms
Median	25.263ms	6.937ms	8.681ms
Max	85.147ms	38.291ms	13.761ms
Flakiness	0,3% (3/1000)	0%	0%

Πίνακας 4.1: Αποτελέσματα τοπικού benchmark (1.000 εκτελέσεις, Windows 11, AMD Ryzen 5 5600H, Node.js v20)

4.1.1 Ανάλυση Ταχύτητας

Το Playwright ήταν το ταχύτερο εργαλείο τοπικά, με μέσο χρόνο 6.923ms (6,9 δευτερόλεπτα) - περίπου 3,7× ταχύτερο από το Cypress και 1,3× από το Selenium. Το Selenium κατέλαβε τη δεύτερη θέση με μέσο χρόνο 8.683ms (8,7 δευτερόλεπτα). Το Cypress ήταν σημαντικά αργότερο, με μέσο χρόνο 25.440ms (25,4 δευτερόλεπτα).

Αυτή η κατάταξη αντιβαίνει στη συμβατική αντίληψη ότι το Selenium είναι το πιο αργό εργαλείο. Η εξήγηση εντοπίζεται στο startup overhead: ο Electron runner του Cypress απαιτεί σημαντικά περισσότερο χρόνο εκκίνησης και επεξεργασίας του command queue σε σύγκριση με τον lean custom runner του Selenium. Με άλλα λόγια, η αρχιτεκτονική in-browser του Cypress - πλεονέκτημα για debugging - αποτελεί μειονέκτημα για raw execution speed.

Αξιοσημείωτο είναι ότι η διαφορά Mean vs Median είναι ελάχιστη για το Selenium (8.683 vs 8.681) και το Playwright (6.923 vs 6.937), υποδεικνύοντας σχεδόν συμμετρική κατανομή. Αντιθέτως, το Cypress παρουσιάζει Mean > Median (25.440 vs 25.263), υποδεικνύοντας θετική ασυμμετρία (right skew) λόγω σποραδικών αργών εκτελέσεων.

Η απόλυτη τυπική απόκλιση (StdDev) δεν είναι δίκαιο μέτρο σύγκρισης μεταξύ εργαλείων με πολύ διαφορετικούς μέσους χρόνους. Για τον λόγο αυτό υπολογίστηκε η σχετική τυπική

απόκλιση (RSD). Το Selenium παρουσίασε τη χαμηλότερη RSD (7,5%), καθιστώντας το πιο προβλέψιμο εργαλείο σε τοπικό περιβάλλον. Η χαμηλή διασπορά εξηγείται από την αρχιτεκτονική του WebDriver: κάθε εντολή ακολουθεί ένα σταθερό HTTP round-trip pattern χωρίς ασύγχρονες βελτιστοποιήσεις που θα μπορούσαν να εισάγουν μεταβλητότητα.

Η υψηλότερη RSD του Playwright (19,7%) οφείλεται σε σποραδικές ακραίες τιμές: ο maximum χρόνος (38.291ms) είναι $\sim 5,5\times$ ο μέσος, ενώ στο Selenium ο maximum (13.761ms) είναι μόλις $\sim 1,6\times$ τον μέσο. Αυτό υποδηλώνει ότι το CDP (Chrome DevTools Protocol) του Playwright, αν και ταχύτερο κατά μέσο όρο, είναι πιο ευαίσθητο σε περιστασιακά GC pauses ή system load spikes.

4.1.2 Ανάλυση Flakiness

Σε 1.000 εκτελέσεις, μόνο το Cypress εμφάνισε αποτυχίες: ακριβώς 3, στις εκτελέσεις #289, #290 και #291 - τρεις διαδοχικές αποτυχίες.

Run	Cypress	Playwright	Selenium	Timestamp
#288	24.598ms ✓	6.160ms ✓	7.652ms ✓	19:04:49
#289	23.624ms ✗	6.225ms ✓	7.801ms ✓	19:05:27
#290	24.555ms ✗	5.853ms ✓	7.580ms ✓	19:06:05
#291	23.814ms ✗	6.459ms ✓	7.901ms ✓	19:06:43
#292	21.376ms ✓	5.877ms ✓	7.674ms ✓	19:07:21

Πίνακας 4.2: Λεπτομέρειες αποτυχιών Cypress

Η ανάλυση αυτών των αποτυχιών (πβ. Πίνακα 4.2) αποκαλύπτει σημαντικά μοτίβα:

Αρχικά, οι αποτυχίες είναι διαδοχικές (consecutive cluster), όχι τυχαία κατανεμημένες. Αυτό αποκλείει τον τυχαίο θόρυβο (random flakiness) και υποδεικνύει ένα παροδικό περιβαλλοντικό γεγονός (transient environmental event) - πιθανώς ένα burst δραστηριότητας στο σύστημα ή στο δίκτυο - που διήρκεσε περίπου 2 λεπτά (19:05 έως 19:07).

Εξάλλου, κατά τη διάρκεια αυτών ακριβώς των εκτελέσεων, τα Playwright και Selenium ολοκλήρωσαν επιτυχώς, ενάντια στην ίδια εφαρμογή (saucedemo.com) και στο ίδιο δίκτυο. Αυτό υποδηλώνει ότι η in-browser αρχιτεκτονική του Cypress είναι πιο ευαίσθητη σε εξωτερικές διαταραχές σε σύγκριση με τις out-of-process αρχιτεκτονικές των Playwright (CDP) και Selenium (WebDriver).

Συνεπώς, οι χρόνοι εκτέλεσης κατά τις αποτυχίες (23-24 δευτερόλεπτα) είναι εντός φυσιολογικών ορίων - δεν πρόκειται για timeouts. Αυτό υποδεικνύει αποτυχία assertion (π.χ. ένα element δεν φόρτωσε εγκαίρως) και όχι κατάρρευση (crash) του εργαλείου.

Σε μεγάλη κλίμακα (1.000 εκτελέσεων), τα Playwright και Selenium πέτυχαν τέλει 0% flakiness. Το Cypress εμφάνισε 0,3% - αριθμός χαμηλός αλλά μη μηδενικός, με μοτίβο που υποδεικνύει περιβαλλοντική ευαισθησία της in-browser αρχιτεκτονικής.

4.2 Αποτελέσματα CI Pipeline (n=5)

Συλλέχθηκαν χειροκίνητα χρόνοι εκτέλεσης από 5 διαδοχικά επιτυχή workflow runs (#55–#59) στο GitHub Actions. Για διευκόλυνση παρουσιάζονται συγκεντρωτικά στον κάτωθι πίνακα (Πίνακας 4.3):

	Selenium	Cypress	Playwright
Mean	52,6s	62,8s	76,8s
StdDev	2,7s	7,5s	27,8s
Min	49s	53s	58s
Median	52s	63s	66s
Max	57s	72s	132s

Πίνακας 4.3: Χρόνοι εκτέλεσης CI pipeline

Σημείωση: Τα CI δεδομένα (n=5) παρουσιάζονται ως ενδεικτικά και όχι ως στατιστικά έγκυρα.

Ένα ιδιαίτερα ενδιαφέρον εύρημα της παρούσας εργασίας αποτελεί η αντιστροφή της κατάταξης των εργαλείων ως προς την απόδοσή τους όταν αυτά μεταφέρονται από το τοπικό περιβάλλον ανάπτυξης στο περιβάλλον Συνεχούς Ενσωμάτωσης (CI). Η διαφοροποίηση αυτή εντοπίζεται κατά κύριο λόγο στο κόστος εγκατάστασης του προγράμματος περιήγησης (browser installation overhead). Συγκεκριμένα, το Playwright απαιτεί τη λήψη και εγκατάσταση δικών του δυαδικών αρχείων (περίπου 437MB ακόμη και με τη χρήση προσωρινής μνήμης), γεγονός που σε περιπτώσεις χωρίς caching (όπως στην εκτέλεση #56) οδήγησε σε κατακόρυφη αύξηση του χρόνου στα 132 δευτερόλεπτα. Αντίθετα, το Selenium αξιοποιεί τον Chrome μέσω της ελαφριάς διαδικασίας setup-chrome (περίπου 10 δευτερόλεπτα), ενώ το Cypress χρησιμοποιεί τον ενσωματωμένο Electron browser μέσω της εντολής npm ci, παρουσιάζοντας ένα μέτριο και σταθερό χρονικό κόστος. Σημειώνεται ότι στο τοπικό περιβάλλον το εν λόγω κόστος εκμηδενίζεται, καθώς οι απαραίτητοι browsers είναι ήδη εγκατεστημένοι.

Η παρατήρηση αυτή υπογραμμίζει ότι η επιλογή ενός εργαλείου αυτοματοποιημένου ελέγχου αποκλειστικά με βάση τοπικές μετρήσεις (benchmarks) ενδέχεται να οδηγήσει σε μη βέλτιστες αποφάσεις. Καθώς η πλειονότητα των δοκιμών σε επαγγελματικές ομάδες ανάπτυξης εκτελείται σε περιβάλλον CI, η πραγματική απόδοση των εργαλείων σε αυτό το πλαίσιο διαφοροποιείται σημαντικά από εκείνη του τοπικού περιβάλλοντος, καθιστώντας απαραίτητη τη συνεκτίμηση των υποδομών εκτέλεσης κατά τη διαδικασία της αξιολόγησης.

4.3 Ανάλυση Γραμμών Κώδικα (LOC)

Το Selenium απαίτησε κατά μέσο όρο $3,1\times$ περισσότερες γραμμές κώδικα από το Playwright για ταυτόσημη λογική ελέγχου. Τα Cypress και Playwright παρουσίασαν παρόμοια πολυπλοκότητα (πβ. Πίνακα 4.4)

Η ανάλυση ανά κατηγορία σεναρίου αποκαλύπτει ένα σαφές μοτίβο:

Σενάρια με πολλαπλές αλληλεπιδράσεις UI (BVA-01: 6 click events, DT-R2: multi-step form, ASYNC-01: τρεις μεταβάσεις): ο πολλαπλασιαστής κυμαίνεται μεταξύ $3,3\times$ και $3,4\times$. Εκεί χρειάζονται retry loops, JavaScript click injection, explicit waits - δηλαδή κώδικας που τα Cypress/Playwright χειρίζονται αυτόματα (auto-retry, auto-wait).

Σενάρια μέτρησης χρόνων (PERF-01, PERF-02): ο πολλαπλασιαστής μειώνεται σε 2,0×–2,3×. Η λογική αυτών των σεναρίων είναι κυρίως `Date.now()` και `assertions` - η αρχιτεκτονική διαφορά δεν επηρεάζει τόσο, αφού δεν υπάρχουν πολλαπλά UI events. Σημείωση: Το PERF-01 χρησιμοποιεί διαφορετική μέθοδο μέτρησης ανά εργαλείο (`performance.mark()` στο Cypress, `Date.now()` στα Playwright/Selenium) λόγω αρχιτεκτονικών περιορισμών (βλ. §3.4.2). Τα δεδομένα PERF-01 δεν είναι απολύτως συγκρίσιμα μεταξύ εργαλείων και πρέπει να ερμηνεύονται με αυτή την επιφύλαξη. Το PERF-02 χρησιμοποιεί κοινό Navigation Timing API και είναι πλήρως συγκρίσιμο.

Εύρημα: Η πολυπλοκότητα κώδικα στο Selenium δεν είναι σταθερή - κλιμακώνεται ανάλογα με τον αριθμό αλληλεπιδράσεων UI, αντικατοπτρίζοντας την απουσία `auto-retry`.

Σενάριο	Cypress	Playwright	Selenium	Sel/PW
Login	23	15	50	3,3×
EP-01	20	19	65	3,4×
BVA-01	18	21	71	3,4×
DT-R2	33	32	106	3,3×
ST-01	22	21	70	3,3×
UC-01	32	30	96	3,2×
ASYNC-01	33	30	103	3,4×
PERF-01	28	20	45	2,3×
PERF-02	26	23	45	2,0×
ΣΥΝΟΛΟ	235	211	651	3,1×
M.O./σενάριο	26,1	23,4	72,3	-

Πίνακας 4.4: Γραμμές κώδικα ανά σενάριο

4.4 Σταθεροποίηση Selenium - Σωρευτικό Λειτουργικό Κόστος

Ένα από τα σημαντικότερα ποιοτικά ευρήματα της εργασίας αφορά τον αριθμό ρυθμίσεων (workarounds) που απαίτησε το Selenium για σταθερή λειτουργία τόσο τοπικά όσο και στο CI. Αυτές οι ρυθμίσεις, τις οποίες τα Cypress και Playwright παρέχουν εγγενώς, ποσοτικοποιούνται ως «σωρευτικό λειτουργικό κόστος»:

1. Chrome flags (7): `--headless=new`, `--no-sandbox`, `--disable-dev-shm-usage`, `--window-size=1920,1080`, `--disable-background-networking`, `--disable-sync`, `--incognito`
2. JavaScript click injection: Αντικατάσταση της φυσικής μεθόδου `.click()` με `executeScript('arguments[0].click()', element)` λόγω αστοχιών στη διαχείριση React event batching. Η φυσική μέθοδος `click` του WebDriver στέλνει ένα native OS event (mouse click) στο element, ενώ η JavaScript μέθοδος εκτελεί απευθείας τον click handler. Σε React SPA, η native μέθοδος αποτύγχανε σποραδικά επειδή το event batching του React δεν προλάβαινε να επεξεργαστεί το click πριν ο driver προχωρήσει στην επόμενη εντολή.
3. Retry loops: Βρόχοι επανάληψης ανά click (for loop, 10 iterations, 300ms sleep) με επαλήθευση μέσω `badge`, για αντιστάθμιση της απουσίας auto-retry.
4. Negative assertions: Υλοποίηση αρνητικών assertions μέσω `findElements().length===0` αντί μιας απλής `.should('not.exist')` (Cypress) ή `expect().toHaveLength(0)` (Playwright).
5. Viewport ρύθμιση: Χειροκίνητη ρύθμιση 1920×1080 λόγω default 800×600 σε headless mode.
6. Custom runner.js: Ανάπτυξη runner για εκτέλεση δοκιμών λόγω απουσίας ενσωματωμένου test runner.
7. Custom benchmarking-runner.js: Ξεχωριστός runner αποκλειστικά για τη σουίτα benchmark, ώστε να μην εκτελούνται τα black-box σενάρια σε κάθε benchmark iteration.

Σύνολο: ~13 ρυθμίσεις που τα σύγχρονα εργαλεία παρέχουν εγγενώς. Αυτό δεν σημαίνει ότι το Selenium είναι «κακό» εργαλείο - αντίθετα, η ευελιξία του επιτρέπει αυτές τις ρυθμίσεις. Σημαίνει ωστόσο ότι η χρήση του Selenium απαιτεί σημαντικά μεγαλύτερη εμπειρία και χρόνο εγκατάστασης.

4.5 Συζήτηση Αποτελεσμάτων

Απάντηση στα Ερευνητικά Ερωτήματα

EE1 (Ταχύτητα εκτέλεσης σε $n=1000$): Το Playwright είναι το ταχύτερο τοπικά (mean 6,9s), ακολουθούμενο από Selenium (8,7s) και Cypress (25,4s). Η κατάταξη παραμένει σταθερή σε 1.000 εκτελέσεις και αντιβαίνει στη συμβατική αντίληψη ότι το Selenium είναι αργό. Η διαφορά εξηγείται από το startup overhead του Electron runner (Cypress) vs τον lean custom runner (Selenium) vs το CDP (Playwright).

EE2 (Σταθερότητα κατάταξης μεταξύ περιβαλλόντων): Όχι. Η κατάταξη αντιστρέφεται πλήρως στο CI: Selenium 1ο (52,6s), Cypress 2ο (62,8s), Playwright 3ο (76,8s). Η αιτία είναι το κόστος εγκατάστασης browser. Αυτό αποτελεί εύρημα που δεν έχει τεκμηριωθεί στην υπάρχουσα βιβλιογραφία.

EE3 (Πολυπλοκότητα κώδικα): Το Selenium απαιτεί $3,1\times$ περισσότερες γραμμές κώδικα από το Playwright. Τα Cypress και Playwright είναι παρόμοια ($1,1\times$). Το χάσμα κλιμακώνεται ανάλογα με τον αριθμό αλληλεπιδράσεων UI.

EE4 (Αρχιτεκτονική αιτιολόγηση): Κάθε παρατηρούμενη διαφορά εξηγείται από τη θεμελιώδη αρχιτεκτονική: το WebDriver protocol (Selenium) εισάγει HTTP overhead αλλά προβλεψιμότητα, το CDP (Playwright) παρέχει ταχύτητα αλλά μεταβλητότητα, και η in-browser εκτέλεση (Cypress) προσφέρει debugging αλλά startup overhead.

EE5 (Αξιοπιστία σε παρατεταμένη εκτέλεση): Τα Playwright και Selenium πέτυχαν τέλει 0% flakiness σε 1.000 εκτελέσεις. Το Cypress εμφάνισε 0,3% (3 διαδοχικές αποτυχίες), υποδεικνύοντας ελαφρώς μεγαλύτερη ευαισθησία σε περιβαλλοντικές μεταβολές.

5. Συμπεράσματα

Στο πέμπτο και καταληκτικό κεφάλαιο της παρούσας εργασίας ανακεφαλαιώνονται τα κύρια συμπεράσματα που προέκυψαν από την εμπειρική αξιολόγηση των Selenium, Playwright και Cypress. Παράλληλα, προσδιορίζονται οι περιορισμοί που διέπουν την πειραματική διαδικασία, διατυπώνονται συγκεκριμένες προτάσεις για τη μελλοντική επέκταση της

έρευνας και καταγράφεται η επιστημονική και πρακτική συνεισφορά της διατριβής στο πεδίο του αυτοματοποιημένου ελέγχου.

Δομή του Κεφαλαίου

Η ύλη του κεφαλαίου διαρθρώνεται στους ακόλουθους άξονες:

- Σύνοψη Κύριων Αποτελεσμάτων (Ενότητα 5.1): Συγκεντρωτική αποτίμηση των ευρημάτων σχετικά με τον ρόλο της αρχιτεκτονικής, το φαινόμενο της αντιστροφής της κατάταξης μεταξύ των περιβαλλόντων, την ανταγωνιστικότητα του Selenium και τα ποσοστά flakiness.
- Περιορισμοί (Ενότητα 5.2): Αναγνώριση των παραμέτρων που περιορίζουν τη γενίκευση των αποτελεσμάτων, όπως η χρήση μίας μόνο δοκιμαζόμενης εφαρμογής, η συγκεκριμένη διάταξη υλικού, το μέγεθος του δείγματος στο CI και ο περιορισμός σε μία γλώσσα προγραμματισμού.
- Προτάσεις για Μελλοντική Έρευνα (Ενότητα 5.3): Καθορισμός μελλοντικών ερευνητικών κατευθύνσεων, συμπεριλαμβανομένης της δοκιμής σε πολλαπλά frontend frameworks, της μέτρησης κατανάλωσης πόρων και της πρακτικής αξιολόγησης AI-powered εργαλείων.
- Συνεισφορά (Ενότητα 5.4): Παρουσίαση της προστιθέμενης αξίας της εργασίας, η οποία εστιάζει στη στατιστική τεκμηρίωση μεγάλης κλίμακας, στην παροχή ενός αναπαραγωγίμου αποθετηρίου κώδικα και στη δημιουργία ενός πρακτικού οδηγού για ομάδες ανάπτυξης.

5.1 Σύνοψη Κύριων Αποτελεσμάτων

Η παρούσα εργασία πραγματοποίησε εκτεταμένη εμπειρική σύγκριση τριών εργαλείων αυτοματοποιημένου ελέγχου web εφαρμογών - Selenium WebDriver, Playwright και Cypress - σε κλίμακα 1.000 τοπικών εκτελέσεων, 5 εκτελέσεων CI, και 27 αρχείων δοκιμών. Τα ευρήματα τεκμηριώθηκαν επίσης σε δομημένο ερευνητικό ημερολόγιο (RESEARCH_LOG.md). Τα κυριότερα συμπεράσματα συνοψίζονται ως εξής:

Πρώτον, η αρχιτεκτονική αποτελεί τη ρίζα κάθε παρατηρούμενης διαφοράς. Οι τρεις αρχιτεκτονικές - WebDriver protocol (Selenium), Chrome DevTools Protocol (Playwright), in-browser proxy (Cypress) - εξηγούν κάθε μετρήσιμη απόκλιση: στην ταχύτητα, στη σταθερότητα, στην πολυπλοκότητα κώδικα, και στη συμπεριφορά σε CI. Δεν πρόκειται για τυχαίες διαφορές αλλά για αναπόφευκτες συνέπειες σχεδιαστικών αποφάσεων. Το WebDriver εισάγει HTTP overhead αλλά προβλεψιμότητα (RSD 7,5%), το CDP παρέχει ταχύτητα αλλά σποραδική μεταβλητότητα (RSD 19,7%), και η in-browser εκτέλεση προσφέρει εξαιρετικό debugging αλλά startup overhead (3,7× αργότερο σε ωμή ταχύτητα).

Δεύτερον, δεν υπάρχει «καλύτερο» εργαλείο. Κάθε εργαλείο υπερέχει σε διαφορετική διάσταση: το Playwright στην ταχύτητα εκτέλεσης (mean 6,9s τοπικά), το Selenium στη σταθερότητα CI (52,6s mean, RSD 7,5%) και στην προβλεψιμότητα, το Cypress στην εμπειρία debugging (Time Travel, in-browser snapshots). Η ιδέα ενός «νικητή» προϋποθέτει μία διάσταση αξιολόγησης, ενώ η πραγματικότητα περιλαμβάνει πολλαπλές.

Τρίτον, η αντιστροφή κατάταξης μεταξύ τοπικού περιβάλλοντος και CI αποτελεί το κρισιμότερο πρακτικό εύρημα. Το Playwright, ταχύτερο τοπικά κατά 3,7× σε σχέση με το Cypress, γίνεται αργότερο στο CI κατά 1,2×. Το Selenium, δεύτερο τοπικά, γίνεται πρώτο στο CI. Η αιτία - κόστος εγκατάστασης browser - δεν είναι προφανής κατά την τοπική αξιολόγηση. Αυτό σημαίνει ότι ομάδες που επιλέγουν εργαλείο βάσει τοπικών benchmarks μπορεί να λάβουν εσφαλμένη απόφαση. Η αντιστροφή αυτή δεν έχει τεκμηριωθεί στην υπάρχουσα βιβλιογραφία. Σημειώνεται ότι η τάση αυτή τεκμηριώθηκε με $n=5$ CI εκτελέσεις (ενδεικτικά, όχι στατιστικά εγκύρα) και αποτελεί υπόθεση εργασίας για μελλοντική επαλήθευση σε μεγαλύτερο δείγμα.

Τέταρτον, το Selenium - παρά την ηλικία του (2004) - παραμένει ανταγωνιστικό σε raw execution speed, αλλά απαιτεί σημαντικά μεγαλύτερη επένδυση σε υποδομή και εξειδίκευση. Οι ~13 ρυθμίσεις σταθεροποίησης και οι 3,1× περισσότερες γραμμές κώδικα ανά σενάριο αντανakλούν το αρχιτεκτονικό χάσμα μεταξύ μιας βιβλιοθήκης (library) και ενός πλαισίου (framework). Η ωρίμανση των νεότερων εργαλείων δεν καθιστά το Selenium αργό - καθιστά τα νεότερα εργαλεία πιο αποδοτικά ως προς τον χρόνο ανάπτυξης.

Πέμπτον, σε παρατεταμένη εκτέλεση 1.000 runs, τα Playwright και Selenium πέτυχαν μηδενικό ποσοστό αστοχίας, ενώ το Cypress εμφάνισε 0,3% - τρεις διαδοχικές αποτυχίες (#289–#291) που υποδεικνύουν ευαισθησία της in-browser αρχιτεκτονικής σε παροδικές

περιβαλλοντικές μεταβολές. Ο αριθμός είναι χαμηλός αλλά μη μηδενικός, και η σημασία του αυξάνει σε μεγάλες σουίτες δοκιμών σε production περιβάλλον.

5.2 Περιορισμοί

Η εργασία υπόκειται σε αρκετούς περιορισμούς που πρέπει να ληφθούν υπόψη κατά τη γενίκευση των αποτελεσμάτων:

Μονοεφαρμογή: Τα αποτελέσματα βασίζονται σε μία εφαρμογή υπό δοκιμή (saucedemo.com). Η εφαρμογή αυτή, αν και πλήρης React SPA, παρουσιάζει χαρακτηριστικά που ευνοούν τον αυτοματισμό: σταθερή δομή DOM, αξιόπιστα data-test attributes ως locators, απουσία δυναμικά παραγόμενων αναγνωριστικών (dynamic IDs), απουσία iframes και shadow DOM, και απλή αυθεντικοποίηση (username/password χωρίς OAuth, CAPTCHA ή πολυπαραγοντική επαλήθευση). Επιπλέον, η εφαρμογή χρησιμοποιεί στατικά, προγνωστά διαπιστευτήρια (standard_user / secret_sauce) χωρίς session timeouts ή rate limiting, συνθήκες που σε πραγματικές εφαρμογές αποτελούν συνηθισμένη πηγή flakiness - ιδιαίτερα σε μεγάλης κλίμακας επαναλαμβανόμενες εκτελέσεις. Σε εφαρμογές με πολύπλοκα SPA frameworks (Angular, Vue), server-side rendering, δυναμικά παραγόμενα class names (CSS Modules, Styled Components), iframes, shadow DOM, ή ροές αυθεντικοποίησης OAuth - συνθήκες που αυξάνουν σημαντικά την πολυπλοκότητα των locators και τις ασύγχρονες αλληλεπιδράσεις - τα αποτελέσματα ενδέχεται να διαφέρουν, ιδιαίτερα ως προς τα ποσοστά flakiness και τη σχετική πολυπλοκότητα κώδικα.

Ένα hardware configuration: Οι τοπικές μετρήσεις αντιπροσωπεύουν ένα συγκεκριμένο σύστημα (AMD Ryzen 5 5600H, Windows 11). Διαφορετικό hardware, λειτουργικό σύστημα, ή φόρτος εργασίας μπορεί να αλλάξει τους απόλυτους χρόνους - αν και η σχετική κατάταξη αναμένεται να παραμείνει σταθερή.

CI δείγμα: Τα CI δεδομένα ($n=5$) δεν ικανοποιούν το threshold $n \geq 30$ του Κεντρικού Οριακού Θεωρήματος. Παρουσιάζονται ως ενδεικτικά μιας σαφούς τάσης, αλλά δεν μπορούν να θεωρηθούν στατιστικά εγκύρα.

Γλώσσα προγραμματισμού: Και τα τρία εργαλεία χρησιμοποιήθηκαν αποκλειστικά με JavaScript. Η απόδοση του Selenium σε Java ή Python - τις πιο δημοφιλείς γλώσσες χρήσης του - μπορεί να διαφέρει.

AI-powered tools: Τα εργαλεία αυτοματισμού με AI (Testim, Mabl, Applitoools) αναλύθηκαν μέσω βιβλιογραφίας, χωρίς πρακτική υλοποίηση σεναρίων. Μια μελλοντική εργασία θα μπορούσε να τα συμπεριλάβει ως τέταρτη κατηγορία σύγκρισης.

5.3 Προτάσεις για Μελλοντική Έρευνα

Τα ευρήματα της παρούσας εργασίας ανοίγουν αρκετές κατευθύνσεις για μελλοντική έρευνα:

Επέκταση σε πολλαπλές εφαρμογές υπό δοκιμή, συμπεριλαμβανομένων πραγματικών production sites με διαφορετικά frontend frameworks, ώστε να ελεγχθεί η γενικευσιμότητα των αποτελεσμάτων.

Εκτέλεση CI benchmark σε κλίμακα $n \geq 30$ για στατιστικά εγκύρα αποτελέσματα, ιδανικά σε πολλαπλούς CI providers (GitHub Actions, GitLab CI, Jenkins, CircleCI).

Πρακτική αξιολόγηση AI-powered εργαλείων (Testim, Mabl, testRigor) στα ίδια σενάρια, ώστε να μετρηθεί η πραγματική αποτελεσματικότητα self-healing locators σε σύγκριση με data-test attributes.

Μέτρηση κατανάλωσης πόρων (CPU, μνήμη, δίκτυο) κατά την εκτέλεση δοκιμών, ώστε να ποσοτικοποιηθεί το κόστος υποδομής κάθε εργαλείου.

Σύγκριση σε πολλαπλές γλώσσες προγραμματισμού (Java, Python) για τα εργαλεία που τις υποστηρίζουν (Selenium, Playwright).

Αξιολόγηση παράλληλης εκτέλεσης (parallelisation) και της επίδρασής της στον συνολικό χρόνο CI pipeline.

5.4 Συνεισφορά

Η παρούσα εργασία συνεισφέρει στο πεδίο με τους ακόλουθους τρόπους:

Παρέχει στατιστικά τεκμηριωμένη σύγκριση τριών εργαλείων σε κλίμακα 1.000 εκτελέσεων - σημαντικά μεγαλύτερη από τις υπάρχουσες μελέτες στη βιβλιογραφία.

Τεκμηριώνει για πρώτη φορά την αντιστροφή κατάταξης μεταξύ τοπικού περιβάλλοντος και CI - ένα εύρημα με άμεσες πρακτικές επιπτώσεις για ομάδες ανάπτυξης.

Ποσοτικοποιεί το «σωρευτικό λειτουργικό κόστος» του Selenium (~13 ρυθμίσεις, 3,1× γραμμές κώδικα), παρέχοντας μετρήσιμο πλαίσιο για την αξιολόγηση του trade-off μεταξύ ευελιξίας και ευκολίας χρήσης.

Παρέχει πλήρως αναπαραγώγιμο αποθετήριο κώδικα (<https://github.com/Menguin/thesis-automation-lab>), ανταποκρινόμενη στο εντοπισμένο κενό αναπαραγωγιμότητας στην υπάρχουσα βιβλιογραφία (Garousi & Felderer, 2017).

Αποτελεί πρακτικό οδηγό για ομάδες ανάπτυξης που αξιολογούν εργαλεία αυτοματισμού, παρέχοντας δεδομένα σε πολλαπλές διαστάσεις αντί μονοδιάστατων συγκρίσεων.

Βιβλιογραφικές Αναφορές

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

Almabruk, S., Abdalhamid, S., & Almabruk, T. (2025). Comparative reliability analysis of Selenium and Playwright: Evaluating automated software testing tools. *Asian Journal of Research in Computer Science*, 18(1), 34–44. <https://doi.org/10.9734/ajrcos/2025/v18i1546>

Applitools. (2024). Visual AI testing documentation. <https://applitools.com/docs/>

Brahmbhatt, K. H. (2023). Comparative analysis of selecting a test automation framework for an e-commerce website [Μεταπτυχιακή εργασία, Tallinn University of Technology]. TalTech Digital Collection. <https://digikogu.taltech.ee/>

Colantonio, J. (2021). Selenium vs Cypress vs Playwright: A comparison. TestGuild. <https://testguild.com/>

Cypress Documentation. (2024). Why Cypress? <https://docs.cypress.io/app/get-started/why-cypress>

Dustin, E., Garrett, T., & Gauf, B. (2009). Implementing automated software testing. Addison-Wesley.

García, B., Ricca, F., Leotta, M., & Munoz-Organero, M. (2026). Test automation with Selenium: A survey. *Information and Software Technology*, 194, Article 107621. <https://doi.org/10.1016/j.infsof.2026.107621>

Garousi, V., & Felderer, M. (2017). Worlds apart: A comparison of industry and academic focus areas in software testing. *IEEE Software*, 34(5), 38–45.

Garousi, V., Felderer, M., Kuhrmann, M., & Herkiloğlu, K. (2017). What industry wants from academia in software testing? Hearing practitioners' opinions. Στο Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17) (σσ. 65–69). ACM. <https://doi.org/10.1145/3084226.3084264>

GitHub Documentation. (2024). GitHub Actions. <https://docs.github.com/en/actions>

ISTQB. (2024). Certified tester foundation level syllabus (v4.0.1). https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf

Playwright Documentation. (2024). Getting started. <https://playwright.dev/docs/intro>

Selenium Project. (2024). WebDriver documentation.

<https://www.selenium.dev/documentation/webdriver/>

testRigor. (2024). AI test automation. <https://testrigor.com/>

Tymoshchuk, A. (2025). The evolution of test automation: From Selenium to Playwright. A comparison of automation tools: Selenium vs. Playwright vs. Cypress. International Journal of Computer (IJC), 54(1), 37–45.

<https://ijcjournal.org/InternationalJournalOfComputer/article/view/2355>

Παράρτημα Α: Αποθετήριο Κώδικα

Ο πλήρης κώδικας της εργασίας είναι διαθέσιμος στο:

<https://github.com/Menguin/thesis-automation-lab>

Για λόγους οικονομίας χώρου παρατίθενται τρία αντιπροσωπευτικά σενάρια - ένα ανά εργαλείο - που υλοποιούν ταυτόσημη λογική ελέγχου (σενάριο login – happy path). Ο πλήρης κώδικας όλων των 27 αρχείων δοκιμών είναι διαθέσιμος στο αποθετήριο GitHub.

Παρατίθενται επίσης και τα custom runners, καθώς και η σύντομη περιγραφή τους.

A.1 Selenium

```
const { Builder, By, until } = require('selenium-webdriver');
const chrome = require('selenium-webdriver/chrome');

async function ecommerceLoginFlowValidation() {

  // Start the timer
  const startTime = Date.now();

  // Configure Chrome to run headless (required for CI environments with no display)
  let options = new chrome.Options();
  options.addArguments('--headless=new');
  options.addArguments('--no-sandbox');
  options.addArguments('--disable-dev-shm-usage');

  // 1. Launch a headless Chrome instance
  const driver = await new Builder()
    .forBrowser('chrome')
    .setChromeOptions(options)
    .build();

  try {

    // 2. Navigate to the target web application
    await driver.get('https://www.saucedemo.com/');

    // 3. Locate the username input field and fill it with valid credentials
    await driver.findElement(By.id('user-name')).sendKeys('standard_user');

    // 4. Locate the password input field and fill it
    await driver.findElement(By.id('password')).sendKeys('secret_sauce');

    // 5. Locate and click the login submission button
    await driver.findElement(By.id('login-button')).click();
```

```
// 6. Wait for the page to change, then grab the current URL
await driver.wait(until.urlContains('inventory'), 5000);
const currentURL = await driver.getCurrentUrl();

// 7. Assertion: Verify the URL contains 'inventory'
if (currentURL.includes('inventory')) {
    console.log('TEST PASSED: Login successful, redirected to inventory
page');
} else {
    console.log('TEST FAILED: Login did not redirect correctly');
}

// 8. Assertion: Verify the page title
const title = await driver.getTitle();
if (title.includes('Swag Labs')) {
    console.log('TEST PASSED: Page title confirms correct
application');
} else {
    console.log('TEST FAILED: Page title does not match');
}

// 9. Stop the timer and log execution time
const endTime = Date.now();
const duration = (endTime - startTime) / 1000;
console.log(`Execution Time: ${duration}s`);

} finally {
    // 10. Always close the browser when done, even if the test fails
    await driver.quit();
}
}
ecommerceLoginFlowValidation();
```

A.2 Playwright

```
const { test, expect } = require('@playwright/test');

test('E-commerce Login Flow Validation', async ({ page }) => {

  // 1. Navigate to the target web application using the Base URL
  await page.goto('/');

  // 2. Locate the username input field and fill it with valid
  // credentials
  await page.locator('#user-name').fill('standard_user');

  // 3. Locate the password input field and fill it
  await page.locator('#password').fill('secret_sauce');

  // 4. Locate and click the login submission button
  await page.locator('#login-button').click();

  // 5. Assertion: Verify the login was successful by confirming the URL
  // changed to the inventory page
  await expect(page).toHaveURL(/.*inventory/);

  // 6. Assertion: Verify the page title confirms we are on the correct
  // application
  await expect(page).toHaveTitle(/Swag Labs/);

});
```

A.3 Cypress

```
describe('E-commerce Login Flow Validation', () => {

  it('should log in successfully with valid credentials', () => {

    // 1. Navigate to the target web application using the Base URL
    cy.visit('/');

    // 2. Locate the username input field and fill it with valid
    credentials
    cy.get('#user-name').type('standard_user');

    // 3. Locate the password input field and fill it
    cy.get('#password').type('secret_sauce');

    // 4. Locate and click the login submission button
    cy.get('#login-button').click();

    // 5. Assertion: Verify the URL contains 'inventory'
    cy.url().should('include', 'inventory');

    // 6. Assertion: Verify the page title confirms correct application
    cy.title().should('include', 'Swag Labs');

  });
});
```

A.4 benchmark.js

Το script αυτοματοποίησης που αναπτύχθηκε εκτελεί επαναλαμβανόμενα τα τρία frameworks ελέγχου (Cypress, Playwright, Selenium) και καταγράφει τον χρόνο εκτέλεσης εκάστου, με σκοπό τη συγκριτική αξιολόγηση της απόδοσής τους. Ο αριθμός των επαναλήψεων ορίζεται παραμετρικά, με προκαθορισμένη τιμή τις τριάντα εκτελέσεις.

Σε κάθε επανάληψη εκτελούνται διαδοχικά και τα τρία frameworks με σταθερή σειρά. Για κάθε framework καταγράφεται η διάρκεια εκτέλεσης σε χιλιοστά του δευτερολέπτου, ενώ η επιτυχής ή μη ολοκλήρωση καταγράφεται μέσω μηχανισμού διαχείρισης εξαιρέσεων, επιτρέποντας την παρακολούθηση τυχόν αστάθειας (flakiness). Τα αποτελέσματα κάθε

επανάληψης αποθηκεύονται άμεσα στον δίσκο, ώστε σε περίπτωση διακοπής να μην απόλλυνται τα έως τότε καταγεγραμμένα δεδομένα.

Μετά την ολοκλήρωση, υπολογίζονται περιγραφικά στατιστικά μεγέθη ανά framework (μέση τιμή, τυπική απόκλιση, ελάχιστη, μέγιστη και διάμεση τιμή, καθώς και ποσοστό αστάθειας) και αποθηκεύονται σε μορφή JSON για πρόσβαση στα ακατέργαστα δεδομένα, αλλά και σε μορφή JavaScript για την οπτικοποίησή τους μέσω φυλλομετρητή.

```
const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

const runs = parseInt(process.argv[2]) || 30;
const resultsDir = path.join(__dirname, 'results');
if (!fs.existsSync(resultsDir)) fs.mkdirSync(resultsDir);

const benchmarkData = {
  metadata: {
    timestamp: new Date().toISOString(),
    totalRuns: runs,
    node: process.version,
    platform: process.platform
  },
  runs: [],
  statistics: {}
};

console.log(`\n Starting Benchmark - ${runs} run(s) per framework\n`);

for (let i = 1; i <= runs; i++) {
  console.log(`\n RUN ${i} of ${runs}`);
  console.log('-----');
```

```
const runResult = {
  runNumber: i,
  timestamp: new Date().toISOString(),
  cypress: { passed: false, durationMs: 0 },
  playwright: { passed: false, durationMs: 0 },
  selenium: { passed: false, durationMs: 0 }
};

// Cypress
console.log('\n  Cypress...');
const cypressStart = Date.now();
try {
  execSync('npm run run-benchmark-cypress', { stdio: 'inherit' });
  runResult.cypress.passed = true;
} catch (e) {
  runResult.cypress.passed = false;
}
runResult.cypress.durationMs = Date.now() - cypressStart;

// Playwright
console.log('\n  Playwright...');
const playwrightStart = Date.now();
try {
  execSync('npm run run-benchmark-playwright', { stdio: 'inherit' });
  runResult.playwright.passed = true;
} catch (e) {
  runResult.playwright.passed = false;
}
runResult.playwright.durationMs = Date.now() - playwrightStart;

// Selenium
console.log('\n  Selenium...');
const seleniumStart = Date.now();
```

```
try {
  execSync('npm run run-benchmark-selenium', { stdio: 'inherit' });
  runResult.selenium.passed = true;
} catch (e) {
  runResult.selenium.passed = false;
}
runResult.selenium.durationMs = Date.now() - seleniumStart;

benchmarkData.runs.push(runResult);

// Save after each run so progress isn't lost if interrupted
saveData(benchmarkData);

console.log(`\n Run ${i} complete`);
}

// Calculate statistics
['cypress', 'playwright', 'selenium'].forEach(fw => {
  const durations = benchmarkData.runs.map(r => r[fw].durationMs);
  const passes = benchmarkData.runs.filter(r => r[fw].passed).length;
  const sum = durations.reduce((a, b) => a + b, 0);
  const mean = sum / durations.length;

  const variance = durations.reduce((acc, val) => acc + Math.pow(val - mean, 2), 0) / durations.length;
  const stdDev = Math.sqrt(variance);
  const sorted = [...durations].sort((a, b) => a - b);

  benchmarkData.statistics[fw] = {
    runs: durations.length,
    passed: passes,
    failed: durations.length - passes,
    flakinessRate: parseFloat(((durations.length - passes) / durations.length * 100).toFixed(2)),
```

```
    meanMs: Math.round(mean),
    stdDevMs: Math.round(stdDev),
    minMs: Math.min(...durations),
    maxMs: Math.max(...durations),
    medianMs: sorted[Math.floor(sorted.length / 2)]
  };
});

saveData(benchmarkData);

console.log('\n=====');
console.log(' BENCHMARK COMPLETE - STATISTICS');
console.log('=====');
['cypress', 'playwright', 'selenium'].forEach(fw => {
  const s = benchmarkData.statistics[fw];
  console.log(`\n${fw.toUpperCase()}:`);
  console.log(`    Passed:  ${s.passed} /  ${s.runs}      (Flakiness:  ${s.flakinessRate}%)`);
  console.log(`    Mean:  ${s.meanMs}ms   StdDev:  ${s.stdDevMs}ms`);
  console.log(`    Min:   ${s.minMs}ms    Median:  ${s.medianMs}ms    Max:  ${s.maxMs}ms`);
});
console.log('\n=====');
console.log(' Data saved to: results/benchmark-data.json');
console.log(' Open results/charts.html in a browser to view charts');
console.log('=====');

function saveData(data) {
  // Write JSON for raw data access
  fs.writeFileSync(
    path.join(resultsDir, 'benchmark-data.json'),
    JSON.stringify(data, null, 2)
  );
}
```

```
// Write JS-loadable version so charts.html works without a web server
fs.writeFileSync(
  path.join(resultsDir, 'benchmark-data.js'),
  `const benchmarkData = ${JSON.stringify(data, null, 2)};`
);
}
```

A.5 runner.js

Ο εκτελεστής (runner) της σουίτας ελέγχων του Selenium αναλαμβάνει τον εντοπισμό και τη διαδοχική εκτέλεση του συνόλου των αρχείων ελέγχου. Αρχικά συγκεντρώνονται όλα τα αρχεία JavaScript από τον ριζικό κατάλογο - με εξαίρεση το ίδιο το αρχείο του εκτελεστή, καθώς και από τους υποκαταλόγους black-box και benchmarking, τα οποία ενοποιούνται σε έναν ενιαίο κατάλογο εκτέλεσης.

Στη συνέχεια, κάθε αρχείο ελέγχου εκτελείται διαδοχικά ως ανεξάρτητη διεργασία. Η επιτυχής ολοκλήρωση καταγράφεται ως επιτυχία, ενώ η εμφάνιση σφάλματος καταγράφεται ως αποτυχία μέσω μηχανισμού διαχείρισης εξαιρέσεων. Μετά την ολοκλήρωση του συνόλου των ελέγχων, εμφανίζεται συγκεντρωτική αναφορά με το πλήθος των επιτυχιών και των αποτυχιών. Εφόσον καταγραφεί τουλάχιστον μία αποτυχία, η διεργασία τερματίζεται με κωδικό εξόδου διάφορο του μηδενός, ώστε να σηματοδοτείται η αποτυχία της σουίτας στο περιβάλλον συνεχούς ενσωμάτωσης (CI).

```
const fs = require('fs');
const path = require('path');
const { execSync } = require('child_process');

const rootFiles = fs
  .readdirSync(__dirname)
  .filter(f => f.endsWith('.js') && f !== 'runner.js')
  .map(f => path.join(__dirname, f));
```

```
const blackBoxFiles = fs
  .readdirSync(path.join(__dirname, 'black-box'))
  .filter(f => f.endsWith('.js'))
  .map(f => path.join(__dirname, 'black-box', f));

const benchmarkingFiles = fs
  .readdirSync(path.join(__dirname, 'benchmarking'))
  .filter(f => f.endsWith('.js'))
  .map(f => path.join(__dirname, 'benchmarking', f));

const allFiles = [...rootFiles, ...blackBoxFiles, ...benchmarkingFiles];

let passed = 0;
let failed = 0;

console.log(`\n Starting Selenium Suite: Found ${allFiles.length}
test(s)...\n`);

for (const file of allFiles) {
  const fileName = path.relative(__dirname, file);
  console.log(` Running: ${fileName}...`);
  try {
    execSync(`node "${file}"`, { stdio: 'inherit' });
    passed++;
  } catch (error) {
    console.error(`\n FAILED: ${fileName}`);
    failed++;
  }
  console.log('-----\n');
}
```

```
console.log(` SELENIUM SUITE RESULTS:`);  
console.log(` Passed: ${passed}`);  
console.log(` Failed: ${failed}\\n`);  
  
if (failed > 0) {  
    process.exit(1);  
}
```

A.6 benchmarking-runner.js

Ο εκτελεστής (runner) της σουίτας benchmarking του Selenium αναλαμβάνει την εκτέλεση αποκλειστικά των αρχείων ελέγχου που αφορούν τη μέτρηση απόδοσης. Συγκεκριμένα, συγκεντρώνονται όλα τα αρχεία JavaScript από τον υποκατάλογο benchmarking, καθένα από τα οποία εκτελείται διαδοχικά ως ανεξάρτητη διεργασία.

Η επιτυχής ολοκλήρωση κάθε ελέγχου καταγράφεται ως επιτυχία, ενώ η εμφάνιση σφάλματος καταγράφεται ως αποτυχία μέσω μηχανισμού διαχείρισης εξαιρέσεων. Μετά την ολοκλήρωση του συνόλου των ελέγχων, εμφανίζεται συγκεντρωτική αναφορά με το πλήθος των επιτυχιών και των αποτυχιών, ενώ εφόσον καταγραφεί τουλάχιστον μία αποτυχία, η διεργασία τερματίζεται με κωδικό εξόδου διάφορο του μηδενός, ώστε να σηματοδοτείται η αποτυχία της σουίτας στο περιβάλλον συνεχούς ενσωμάτωσης (CI).

Αυτός ο runner είναι ουσιαστικά μια στενότερη παραλλαγή του προηγούμενου, με τη διαφορά ότι περιορίζεται στον υποκατάλογο benchmarking. Κάλυψε την ανάγκη απομόνωσης των ελέγχων μέτρησης απόδοσης από τους υπόλοιπους ελέγχους. Ο πλήρης runner (runner.js) εκτελεί τα πάντα: ριζικό κατάλογο, black-box και benchmarking. Αυτό είναι κατάλληλο για επαλήθευση ορθότητας, αλλά ακατάλληλο για μέτρηση χρόνου. Αν ο κεντρικός μηχανισμός benchmarking (που τρέχει 1000 φορές) καλούσε ολόκληρη τη σουίτα, τότε ο μετρούμενος χρόνος durationMs του Selenium θα περιλάμβανε και τους black-box ελέγχους ορθότητας - εργασία άσχετη με τη σύγκριση απόδοσης. Αυτό θα είχε δύο συνέπειες: θα νόθευε τη μέτρηση με χρόνο που δεν αφορά το benchmarking, και θα καθιστούσε τη σύγκριση μεταξύ των τριών frameworks μη ισοδύναμη, αφού δεν θα μετρούσαν όλα τον ίδιο, καθαρά οριοθετημένο φόρτο εργασίας.

Επομένως, χρειαζόταν ένας εκτελεστής περιορισμένος αποκλειστικά στον υποκατάλογο benchmarking, ώστε ο χρόνος που καταγράφει ο κεντρικός μηχανισμός να αντιστοιχεί σε ένα συνεπές και συγκρίσιμο σύνολο ελέγχων για κάθε framework. Πρακτικά, αυτός είναι ο εκτελεστής που καλείται από την εντολή `npm run run-benchmark-selenium` μέσα στον βρόχο μέτρησης, σε αντιδιαστολή με τον πλήρη runner που χρησιμοποιείται για την επαλήθευση ορθότητας ολόκληρης της σουίτας.

```
const fs = require('fs');
const path = require('path');
const { execSync } = require('child_process');

const benchmarkingDir = path.join(__dirname, 'benchmarking');
const files = fs
  .readdirSync(benchmarkingDir)
  .filter(f => f.endsWith('.js'))
  .map(f => path.join(benchmarkingDir, f));

let passed = 0;
let failed = 0;

console.log(`\n Selenium Benchmarking Suite: Found ${files.length}
test(s)...\n`);

for (const file of files) {
  const fileName = path.relative(__dirname, file);
  console.log(` Running: ${fileName}...\n`);
  try {
    execSync(`node "${file}"`, { stdio: 'inherit' });
    passed++;
  } catch (error) {
    console.error(`\n FAILED: ${fileName}\n`);
    failed++;
  }
  console.log('-----\n');
}

console.log(` SELENIUM BENCHMARK RESULTS:`);
console.log(` Passed: ${passed}`);
console.log(` Failed: ${failed}\n`);

if (failed > 0) process.exit(1);
```

Παράρτημα Β: Ερευνητικό Ημερολόγιο

Το παρόν παράρτημα παρουσιάζει το δομημένο ερευνητικό ημερολόγιο που τηρήθηκε καθ' όλη τη διάρκεια εκπόνησης της εργασίας. Κάθε καταχώρηση τεκμηριώνει συμβάντα (incidents), ορόσημα (milestones), και αποφάσεις σχεδιασμού που προέκυψαν κατά την υλοποίηση. Η πλήρης, μη επεξεργασμένη έκδοση στα Αγγλικά βρίσκεται στο αρχείο RESEARCH_LOG.md του αποθετηρίου (github.com/Menguin/thesis-automation-lab).

B.1 Βασικές Πληροφορίες Συστήματος

Λειτουργικό σύστημα: Windows 11

Επεξεργαστής: AMD Ryzen 5 5600H with Radeon Graphics

Μνήμη RAM: 16GB

Node.js: v20.20.2 (αρχικά v24.14.0 - υποβαθμίστηκε σκόπιμα για συμβατότητα με το CI, βλ. B.3.2)

Δίκτυο: Οικιακό Wi-Fi (Download 117,36 Mbps, Upload 20,87 Mbps, Ping 7ms, Download Latency 106ms, Upload Latency 220ms)

B.2 Χρονολόγιο Ανάπτυξης

Φάση 1 - Αρχική Υλοποίηση, Διαλειτουργικότητα και Υποδομή CI (16 Μαρτίου 2026)

Εγκατάσταση και διαλειτουργικότητα εργαλείων: Και τα τρία stations (Playwright, Cypress, Selenium) λειτούργησαν επιτυχώς με βασικό login test. Το Selenium εκτέλεσε σε headed mode με ελαφρώς αργότερη εκκίνηση σε σύγκριση με το Playwright.

Αρχιτεκτονική συμμετρία: Δημιουργήθηκε playwright.config.js για δυναμική διαχείριση του baseURL, αντικαθιστώντας hardcoded URLs. Ενεργοποιήθηκε ο Playwright JSON Reporter (results/playwright.json). Λόγω απουσίας εγγενούς reporter στο Selenium, ενσωματώθηκε

χειροκίνητο χρονόμετρο `Date.now()` στο `login-test.js` - πρώτη ένδειξη της αρχιτεκτονικής ασυμμετρίας μεταξύ βιβλιοθήκης (Selenium) και πλαισίου (Cypress, Playwright).

Δημοσίευση σε GitHub: Δημιουργήθηκε δημόσιο αποθετήριο με αυστηρή πολιτική `.gitignore`. Αποκλείστηκαν τα `node_modules/`, `test-results/`, `playwright-report/`, `.env`, και αρχεία λειτουργικού συστήματος. Ο φάκελος `results/` διατηρήθηκε σκόπιμα στο repository για διαφάνεια και αναπαραγωγικότητα - η απόφαση αυτή αποτελεί μεθοδολογική επιλογή.

Ανάπτυξη CI pipeline: Σχεδιάστηκε `.github/workflows/ci.yml` με τρία παράλληλα, απομονωμένα jobs σε `ubuntu-latest`. Τα Playwright και Cypress ανίχνευσαν αυτόματα το CI περιβάλλον και μετέβησαν σε `headless mode` χωρίς ρύθμιση. Το Selenium απαίτησε 3 system-level Chrome flags: `--headless=new` (αόρατη εκτέλεση), `--no-sandbox` (απαιτείται σε root Linux), `--disable-dev-shm-usage` (αποφυγή crash σε shared memory). Τα αποτελέσματα ανέβηκαν ως CI artifact.

Εντοπίστηκε προειδοποίηση Node.js deprecation. Αναβαθμίστηκαν όλα τα GitHub Actions σε v5/v6 (`checkout@v5`, `setup-node@v5`, `upload-artifact@v6`). Σταθερή καθήλωση (pinning) στην έκδοση Node 20 LTS αντί `lts/*` - η χρήση δυναμικής έκδοσης θα εισήγαγε κίνδυνο σιωπηλών αλλαγών περιβάλλοντος μεταξύ benchmark runs.

Βελτιστοποίηση Playwright στο CI: Αρχικός χρόνος 1m 39s λόγω λήψης πλήρων browser binaries (~437MB). Εφαρμόστηκε caching (`actions/cache@v5`) με κλειδί βασισμένο στο `package-lock.json` - μείωση σε 1m 7s. Ποσοτικοποίηση σταθερού λειτουργικού κόστους browser management.

Φάση 2 - Ανάπτυξη Υποδομής Selenium (4 Μαΐου 2026)

Custom test runner: Καθώς η σουίτα δοκιμών επεκτάθηκε πέρα από το αρχικό `login-test.js`, το Selenium απαίτησε μηχανισμό εκτέλεσης πολλαπλών αρχείων - ισοδύναμο του `specPattern` του Cypress και του `directory scanning` του Playwright. Αναπτύχθηκε ο `selenium-station/runner.js` (~30 γραμμές): αυτόματη ανακάλυψη αρχείων `.js` εντός του φακέλου, απομόνωση σφαλμάτων ανά αρχείο (`try/catch`), δομημένη σύνοψη `pass/fail`, και σωστά exit codes για το CI (`process.exit(1)` σε αποτυχία).

Φάση 3 - Σενάρια Black-Box, ISTQB Ευθυγράμμιση και Αποτυχίες CI (5 Μαΐου 2026)

Refactor EP-01: Η αρχική υλοποίηση εξήγαγε όλες τις τιμές προϊόντων μέσω DOM scraping (el.innerText), μετατροπή string-to-float, και μαθηματική ταξινόμηση - fragile προσέγγιση εξαρτημένη από context switching (jQuery objects στο Cypress, evaluateAll() στο Playwright). Για αυστηρή ευθυγράμμιση με ISTQB Equivalence Partitioning (επαλήθευση αντιπροσωπευτικών τιμών αντί εξαντλητικού ελέγχου), το σενάριο απλοποιήθηκε σε έλεγχο οριακών τιμών ταξινόμησης (\$7.99 πρώτο, \$49.99 τελευταίο) - δύο γραμμές standard API assertions.

Αρχιτεκτονικές διαφορές στη διαχείριση κατάστασης (BVA-01): Η ανάγκη για iteration σε πολλαπλά κουμπιά αποκάλυψε τρεις διαφορετικές στρατηγικές. Cypress: .each((\$btn) => cy.wrap(\$btn).click()) - η cy.wrap() επανεισάγει το raw DOM element στην αλυσίδα εντολών Cypress. Στη συνέχεια αντικαταστάθηκε από .click({ multiple: true }) - atomic operation χωρίς loop, χωρίς callback, χωρίς jQuery. Playwright: for loop με .nth(i) - αμυντική στρατηγική που επανερωτά (re-query) το DOM σε κάθε iteration, αποφεύγοντας stale element references. Selenium: for...of loop με findElements() - λειτούργησε στο saucedemo αλλά χωρίς ενσωματωμένη προστασία από stale elements.

Προσθήκη DT-R2 (Decision Table): Πρώτο multi-transition σενάριο στη σουίτα. Ο αριθμός χειροκίνητων waits ανά εργαλείο αποτέλεσε άμεσο μετρήσιμο δείκτη: Cypress 0, Playwright 1, Selenium 3 driver.wait() calls. Κάθε μετάβαση σελίδας στο Selenium απαιτεί δικό της explicit wait.

Προσθήκη ST-01 (State Transition): Η ανάγκη για αρνητικό assertion (απουσία badge μετά αφαίρεση προϊόντος) αποκάλυψε κρίσιμη διαφορά. Cypress: .should('not.exist') - μία assertion, αυτόματο polling. Playwright: .not.toBeVisible() - μία assertion, αυτόματο polling. Selenium: findElement() ρίχνει fatal exception αν λείπει το element. Η εναλλακτική findElements() επιστρέφει κενό array αλλά αξιολογεί τη σελίδα στιγμιαία χωρίς polling - απαιτεί driver.sleep(300) πριν τον έλεγχο. Η σκληρά κωδικοποιημένη αναμονή αποτελεί αντι-πρότυπο (anti-pattern): πλασματικά αυξάνει τον χρόνο εκτέλεσης και προκαλεί flakiness όταν η πραγματική ταχύτητα του runner αποκλίνει.

Προσθήκη EG-01 (Error Guessing): Εισαγωγή whitespace σε όλα τα πεδία checkout. Η αναμονή Selenium αυξήθηκε σε 500ms - η αλλαγή διάρκειας μεταξύ δύο σεναρίων στην ίδια σουίτα (300ms στο ST-01, 500ms στο EG-01) αποδεικνύει ότι η «σωστή» τιμή μπορεί μόνο

να μαντευτεί, όχι να υπολογιστεί. Το σενάριο αποκάλυψε πραγματικό ελάττωμα στο saucedemo.com (αποδοχή whitespace ως έγκυρη είσοδο), τεκμηριώθηκε ως defect report, και απομονώθηκε στον φάκελο _sandbox/.

Αποτυχίες χρονισμού στο CI (Selenium): Αποτυχίες σε τρία αρχεία: dt-r2.js (timeout στο checkout), st-01.js (badge ακόμα ορατό μετά sleep), bva-01.js (badge εμφάνισε 4 αντί 6 - clicks εκτελέστηκαν ταχύτερα από όσο μπορούσε να επεξεργαστεί ο CI runner). Αιτία: όλοι οι hardcoded χρονισμοί βαθμονομήθηκαν σε τοπικό Windows 11 - ο ubuntu-latest runner τρέχει σε διαφορετική ταχύτητα. Η πρόβλεψη ότι τα sleeps θα αποτύγχαναν σε διαφορετικό περιβάλλον (βλ. ST-01) επαληθεύτηκε αμέσως.

Διορθώσεις: dt-r2.js - timeouts 5000 → 10000, st-01.js - sleep 300 → 1000, bva-01.js - sleep 500ms ανά click. Το τεχνητό overhead ανά εκτέλεση: BVA-01 +3000ms (6×500ms), ST-01 τριπλασιασμός, DT-R2 διπλασιασμός timeouts. Τα Cypress και Playwright δεν απαίτησαν καμία αλλαγή.

Φάση 4 - Καταστολή Browser, Viewport Bugs, React Event Batching και Benchmarking (6 Μαΐου 2026)

Παρεμβολή Chrome Autofill (DT-R2): Το TensorFlow Lite Autofill του Chrome αναγνώρισε τη φόρμα checkout και συμπλήρωσε αυτόματα τον ταχυδρομικό κώδικα που είχε αφεθεί σκόπιμα κενός. Η δοκιμή πέρασε εσφαλμένα - η φόρμα υποβλήθηκε χωρίς σφάλμα. Ο Chrome εκτελούσε κάτι που ο προγραμματιστής δεν ζήτησε και δεν γνώριζε ότι συμβαίνει. Παράλληλα, το Google Cloud Messaging (GCM) προσπαθούσε να εγγράψει τον headless browser ως συσκευή (PHONE_REGISTRATION_ERROR), καταναλώνοντας πόρους.

Διόρθωση: 3 Chrome flags (--disable-background-networking, --disable-sync, --disable-features=AutofillServerCommunication,AutofillEnableAccountStorageForAddresses) + incognito mode + explicit .clear() στο postal code ως backup. Τα Cypress και Playwright δεν επηρεάστηκαν: το Cypress εκτελείται εντός του browser μέσω proxy (οι background services δεν το φτάνουν), και το Playwright δημιουργεί απομονωμένη περίοδο λειτουργίας (isolated session) με autofill απενεργοποιημένο.

Headless viewport rendering (BVA-01): Σε headless mode χωρίς explicit window size, ο Chrome εκκινούσε σε ανάλυση 800×600 (mobile layout). Η sticky header του saucedemo κάλυπτε τα κουμπιά - το Selenium εντόπιζε σωστά κάθε κουμπί αλλά το click
Διπλωματική Εργασία

προσγειωνόταν στη header λόγω υψηλότερου CSS z-index. Κανένα exception δεν ρίχνονταν: το click event εκτελέστηκε κάπου στο DOM, απλώς όχι στο σωστό στοιχείο - σιωπηλή αποτυχία (silent failure).

Τα Cypress και Playwright διαθέτουν εγγενείς μηχανισμούς ελέγχου actionability - πριν εκτελέσουν click, ελέγχουν εάν κάποιο άλλο στοιχείο επικαλύπτει τον στόχο (element occlusion) και ρυθμίζουν αυτόματα τη θέση κύλισης. Το Selenium εκτελεί click στις υπολογισμένες συντεταγμένες X/Y χωρίς κανέναν τέτοιο έλεγχο. Διόρθωση: --window-size=1920,1080 σε κάθε αρχείο Selenium.

React event batching και dropped clicks: Τα 6 διαδοχικά clicks του Selenium στο BVA-01 εκτελούνταν με ταχύτητα JavaScript (όσο γρήγορα επιτρέπει η CPU), χωρίς ενημερότητα για το εάν η εφαρμογή ολοκλήρωσε την επεξεργασία του προηγούμενου event. Το Virtual DOM του React ομαδοποίησε τα state updates - μόνο ένα click καταχωρήθηκε. Badge: 1 αντί 6.

Σταθεροποίηση: Ολόκληρη η σουίτα black-box του Selenium επανεγγράφηκε με JavaScript click injection (driver.executeScript('arguments[0].click()', button)) - παρακάμπτει το WebDriver routing και εκτελεί click απευθείας στο DOM - και per-click retry loops: μετά κάθε click, βρόχος ελέγχει εάν το badge αυξήθηκε πριν προχωρήσει στο επόμενο κουμπί (μέχρι 10 επαναλήψεις). Η τεχνική αυτή αναπαράγει χειροκίνητα τη retryability που τα Cypress και Playwright παρέχουν εγγενώς.

Υλοποίηση σεναρίων benchmarking: Σχεδιάστηκαν 4 σενάρια: UC-01 (πλήρης ροή αγοράς end-to-end), ASYNC-01 (τρεις μεταβάσεις DOM - inventory→detail→inventory), PERF-01 (χρόνος login-to-inventory μέσω Date.now()), PERF-02 (χρόνος απόκρισης δικτύου μέσω Navigation Timing API).

Αποτυχία network interception (PERF-02): Τόσο η page.waitForResponse() (Playwright) όσο και η cy.intercept() (Cypress) εξέπνευσαν (timeout) - περίμεναν HTTP request που δεν υπήρξε ποτέ. Το saucedemo.com είναι SPA: μετά την αρχική φόρτωση, η πλοήγηση γίνεται client-side μέσω React Router χωρίς HTTP request. Και τα τρία εργαλεία αναβαθμίστηκαν ώστε να χρησιμοποιούν το Navigation Timing API του browser (performance.getEntriesByType('navigation')[0]), μετρώντας τον χρόνο απόκρισης αρχικής φόρτωσης.

Benchmark runner και οπτικοποίηση: Αναπτύχθηκε ο κεντρικός runner (benchmark.js στη ρίζα) που δέχεται αριθμό επαναλήψεων ως παράμετρο (π.χ. `node benchmark.js 1000`). Σε κάθε iteration εκτελεί διαδοχικά: `npm run run-benchmark-cypress`, `npx playwright test`, `node selenium-station/benchmarking-runner.js`. Καταγράφει timestamp, κατάσταση (pass/fail), και χρόνο ανά εργαλείο. Αποθήκευση σε `benchmark-data.json` (ακατέργαστα δεδομένα + στατιστικά) και `benchmark-data.js` (JS module για τα charts). Δημιουργήθηκε `results/charts.html` με Chart.js - 4 γραφήματα: μέσος χρόνος, χρονοσειρά, ποσοστό flakiness, κατανομή min/mean/max.

Αρχική επαλήθευση (1-run): Cypress 67.507ms, Playwright 25.112ms, Selenium 11.359ms - αντιφάσκει με τη διαδεδομένη αντίληψη ότι το Selenium είναι αργότερο. Η αιτία είναι δομική: το Electron runner του Cypress έχει σημαντικό startup overhead, ενώ ο lean custom runner του Selenium εκτελεί ταχύτερα.

B.3 Σύνοψη Κρίσιμων Συμβάντων

Αποτυχία εκκίνησης GUI Cypress

Ημερομηνία: 16/03/2026

Πρόβλημα: Λευκή οθόνη κατά εκκίνηση Cypress Test Runner - αποτυχία αρχικοποίησης Electron frame. Επαναλήφθηκε κατά δεύτερη απόπειρα, επιβεβαιώνοντας ότι ήταν σταθερό πρόβλημα περιβάλλοντος.

Αιτία: Κατεστραμμένη τοπική binary cache Cypress σε συνδυασμό με ασυμβατότητα GPU/Windows.

Επίλυση: `npx cypress cache clear` → `npx cypress install` → `npm install cypress@latest --save-dev` → `npx cypress install`. Μετά τον καθαρισμό, το GUI λειτούργησε κανονικά. Μόνιμη μετάβαση σε headless mode για benchmark.

Δευτερεύον ζήτημα: Η αναβάθμιση σε `cypress@latest` (μέσω caret ^ versioning) εισήγαγε νέα ρύθμιση ασφαλείας (`allowCypressEnv`). Διορθώθηκε στο `cypress.config.js`. Η χρήση ^ versioning εισάγει κίνδυνο αναπαραγωγικότητας.

Ασυμβατότητα εκδόσεων Node.js

Ημερομηνία: 16/03/2026

Πρόβλημα: Deprecation warning για Node.js 20 → 24 migration.

Απόφαση: Υποβάθμιση τοπικού Node.js από v24.14.0 σε v20.20.2 ώστε τοπικό και CI περιβάλλον να ταυτίζονται. Επιπλέον, ο `driver.wait(async () => {...})` pattern συμπεριφερόταν διαφορετικά στο Node 24 vs Node 20 - αντικαταστάθηκε με explicit while loops.

Υψηλός χρόνος Playwright στο CI

Ημερομηνία: 16/03/2026

Πρόβλημα: Playwright CI job ~1m 39s vs Cypress/Selenium ~38-40s.

Αιτία: Λήψη πλήρων browser binaries (Chromium, Firefox, WebKit) + system dependencies.

Επίλυση: GitHub Actions caching (actions/cache@v5, κλειδί: package-lock.json hash). Μείωση σε 1m 07s.

Σημασία: Η αρχιτεκτονική trade-off: μεγαλύτερος έλεγχος περιβάλλοντος (version consistency, cross-browser) σε αντάλλαγμα υψηλότερου setup overhead.

Chrome autofill interference

Ημερομηνία: 06/05/2026

Πρόβλημα: DT-R2 πέρασε εσφαλμένα - ο Chrome συμπλήρωσε αυτόματα τον postal code.

Αιτία: TensorFlow Lite autofill model αναγνώρισε τη φόρμα checkout.

Επίλυση: 3 Chrome flags + `.clear()` + incognito mode.

Σημασία: Αποτυχία που δεν αφορά τον κώδικα δοκιμής αλλά τη συμπεριφορά του browser - αόρατη χωρίς λεπτομερή διερεύνηση.

Headless viewport και silent click interception

Ημερομηνία: 06/05/2026

Πρόβλημα: Μόνο 1 από 6 clicks καταχωρήθηκε στο BVA-01.

Αιτία: Viewport 800×600 → mobile layout → sticky header κάλυπτε κουμπιά → clicks σιωπηλά απορροφούνταν.

Επίλυση: --window-size=1920,1080.

Σημασία: Τρίτη κατηγορία αποτυχίας μοναδικής στο Selenium (μετά timing και autofill). Τα Cypress/Playwright ελέγχουν element occlusion πριν κάθε click.

React event batching

Ημερομηνία: 06/05/2026

Πρόβλημα: 6 clicks στο BVA-01 - μόνο 1 καταχωρήθηκε.

Αιτία: Selenium εκτελεί clicks με ταχύτητα CPU χωρίς ενημερότητα για React state updates. Το Virtual DOM ομαδοποίησε τα events.

Επίλυση: JavaScript click injection + per-click retry loop με badge verification.

Σημασία: Η μη ενημερότητα του Selenium για την ετοιμότητα της εφαρμογής (application readiness) - σε αντίθεση με τα Cypress ({ multiple: true }) και Playwright (re-query ανά iteration) - αποτελεί θεμελιώδη αρχιτεκτονική διαφορά.

SPA routing ακυρώνει network interception

Ημερομηνία: 06/05/2026

Πρόβλημα: page.waitForResponse (Playwright) και cy.intercept (Cypress) timeout.

Αιτία: Το saucedemo.com πλοηγεί client-side μέσω React Router - δεν παράγεται HTTP request μετά την αρχική φόρτωση.

Επίλυση: Μετάβαση στο Navigation Timing API (framework-agnostic).

Σημασία: Ακόμα και τα πιο προηγμένα εργαλεία (CDP, network proxy) δεν μπορούν να παρακολουθήσουν αυτό που το δίκτυο δεν παράγει.

B.4 Χρόνοι CI Pipeline

Εργαλείο - Χρόνος - Μηχανισμός Browser

Selenium - 40s - Προεγκατεστημένος Chrome μέσω setup-chrome@v2

Cypress - 38s - Ενσωματωμένος Electron μέσω npm ci

Playwright (με cache) - 1m 07s - Browser binaries ~437MB μέσω cache restore

Playwright (χωρίς cache) - 1m 39s - Πλήρης λήψη binaries + system deps

Στοιχεία cache:

playwright-a29110c79baf5b230d371746582d78f428d6d85744d27be9ebf4a252011c, μέγεθος ~437MB (458.013.712 bytes). Εξοικονόμηση ~32s ανά run.

Σημείωση: Η διαφορά Playwright (1m 07s) vs Selenium (40s) στο CI αντικατοπτρίζει αποκλειστικά τη διαφορά στο κόστος εγκατάστασης browser - όχι στην ταχύτητα εκτέλεσης δοκιμών. Αυτή η παρατήρηση αποτελεί τη βάση για το εύρημα αντιστροφής κατάταξης (Κεφάλαιο 4.2).

B.5 Κατάλογος Ευρημάτων

Στον παρακάτω κατάλογο συνοψίζονται τα 14 ευρήματα που προέκυψαν εμπειρικά κατά την ανάπτυξη. Κάθε εύρημα αντιπροσωπεύει μια αρχιτεκτονική ή λειτουργική διαφορά που τεκμηριώθηκε μέσω πρακτικής υλοποίησης - όχι μέσω θεωρητικής ανάλυσης.

Εύρημα 1 - Η αρχιτεκτονική είναι η ρίζα κάθε διαφοράς. Κάθε παρατηρούμενη διαφορά ανάγεται στον τρόπο επικοινωνίας με τον browser: Cypress εκτελείται εντός του browser μέσω proxy layer, Playwright επικοινωνεί μέσω CDP (Chrome DevTools Protocol), Selenium εκκινεί τον Chrome ως κανονικό browser και τον οδηγεί εξωτερικά μέσω WebDriver protocol.

Εύρημα 2 - Το χάσμα του test runner. Τα Cypress και Playwright παρέχουν πλήρη υποδομή: ανακάλυψη αρχείων, παράλληλη εκτέλεση, απομόνωση σφαλμάτων, δομημένη αναφορά, CI exit codes. Το Selenium δεν παρέχει τίποτα - απαιτήθηκε ανάπτυξη runner.js (~30 γραμμές).

Εύρημα 3 - Το sleep anti-pattern είναι δομική συνέπεια, όχι επιλογή. Τα Cypress/Playwright κάνουν poll μέχρι η συνθήκη να ικανοποιηθεί. Το Selenium ελέγχει μία φορά σε εκείνο το millisecond. Οι αποτυχίες CI επαλήθευσαν ότι τα sleeps που λειτουργούσαν τοπικά αποτυγχάνουν σε runner διαφορετικής ταχύτητας - η «σωστή» διάρκεια δεν μπορεί ποτέ να είναι γνωστή εκ των προτέρων.

Εύρημα 4 - Τα αρνητικά assertions αποκαλύπτουν ωριμότητα αυτοματισμού. Cypress: `.should('not.exist')` (polling). Playwright: `.not.toBeVisible()` (polling). Selenium: `findElements().length===0` (στιγμιαίο, χωρίς polling - απαιτεί sleep).

Εύρημα 5 - Ο Chrome εκτελείται χωρίς τροποποίηση και αυτό προκαλεί αποτυχίες. Το TensorFlow Lite autofill αναγνώρισε τη φόρμα checkout και συμπλήρωσε αυτόματα πεδία. Τα Cypress/Playwright δεν εκθέτουν τον Chrome σε αυτήν τη συμπεριφορά.

Εύρημα 6 - Το headless viewport δεν ρυθμίζεται αυτόματα. Chrome headless → 800×600 → mobile layout → sticky header intercepts clicks. Τα Cypress/Playwright διαχειρίζονται το viewport εσωτερικά.

Εύρημα 7 - Το React event batching απορρίπτει clicks σιωπηλά. 6 γρήγορα clicks → React ομαδοποιεί → μόνο 1 καταχωρείται. Cypress: `{ multiple: true }` (atomic). Playwright: `query` ανά iteration. Selenium: χωρίς ενημερότητα εφαρμογής.

Εύρημα 8 - Η αυτόματη επανάληψη (auto-retry) είναι η πιο θεμελιώδης διαφορά. Τα Playwright/Cypress παρακολουθούν τη λειτουργία μετά κάθε αλληλεπίδραση - αν ένα click δεν παράξει αποτέλεσμα, επαναλαμβάνουν. Το Selenium εκτελεί μία φορά - αν ο browser είναι απασχολημένος, το event χάνεται.

Εύρημα 9 - Η JavaScript click injection είναι απαραίτητη για αξιοπιστία. Η `button.click()` του Selenium δρομολογείται μέσω WebDriver routing - μπορεί να απορροφηθεί υπό φορτίο CI. Η `executeScript('arguments[0].click()')` εκτελεί click απευθείας στο DOM, παρακάμπτοντας το WebDriver. Γνωστό αντισταθμιστικό workaround - η αναγκαιότητά του αποτελεί εύρημα.

Εύρημα 10 - Η έκδοση Node επηρεάζει τη συμπεριφορά. Ο `driver.wait(async () => {...})` pattern λειτουργούσε στο Node 24 αλλά timeout στο Node 20. Αντικαταστάθηκε με explicit

while loops. Τα Cypress/Playwright αφαιρούν (abstract) τους εσωτερικούς async μηχανισμούς τους.

Εύρημα 11 - Ο SPA routing ακυρώνει το network interception. Μετά την αρχική φόρτωση, η πλοήγηση γίνεται client-side χωρίς HTTP request. Τα `page.waitForResponse` και `cy.intercept` δεν μπορούν να παρακολουθήσουν αυτό που το δίκτυο δεν παράγει. Μετάβαση στο Navigation Timing API.

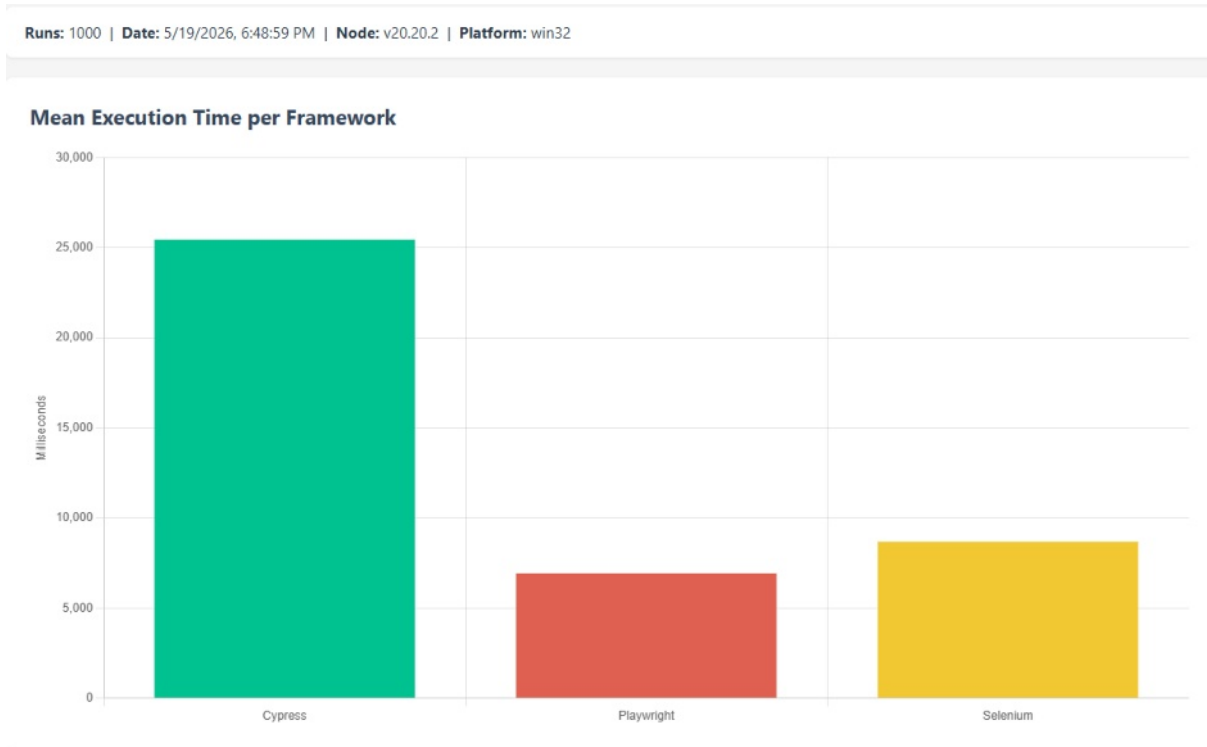
Εύρημα 12 - Σωρευτικό λειτουργικό κόστος. Πλήρης κατάλογος ρυθμίσεων που προστέθηκαν στο Selenium: χειροκίνητο browser launch/teardown, explicit waits ανά μετάβαση, hardcoded sleep values, Chrome background service flags, autofill suppression flags, explicit field clearing, window size ανά αρχείο, per-click React synchronisation loops, JavaScript click injection, per-click action retryability loops, custom test runner, Node version standardisation, incognito mode. Σύνολο: ~13 στοιχεία. Κάθε ένα είναι μηδέν γραμμές κώδικα στα Cypress και Playwright.

Εύρημα 13 - Ο χρόνος CI αντικατοπτρίζει κόστος υποδομής, όχι μόνο ταχύτητα. Οι χρόνοι Selenium στο benchmark πλασματικά αυξάνονται λόγω sleeps, wait timeouts, και retry polling - δεν μετρούν πόσο γρήγορα επεξεργάζεται ο WebDriver, αλλά πόση τεχνητή καθυστέρηση χρειάζεται για αξιόπιστη λειτουργία.

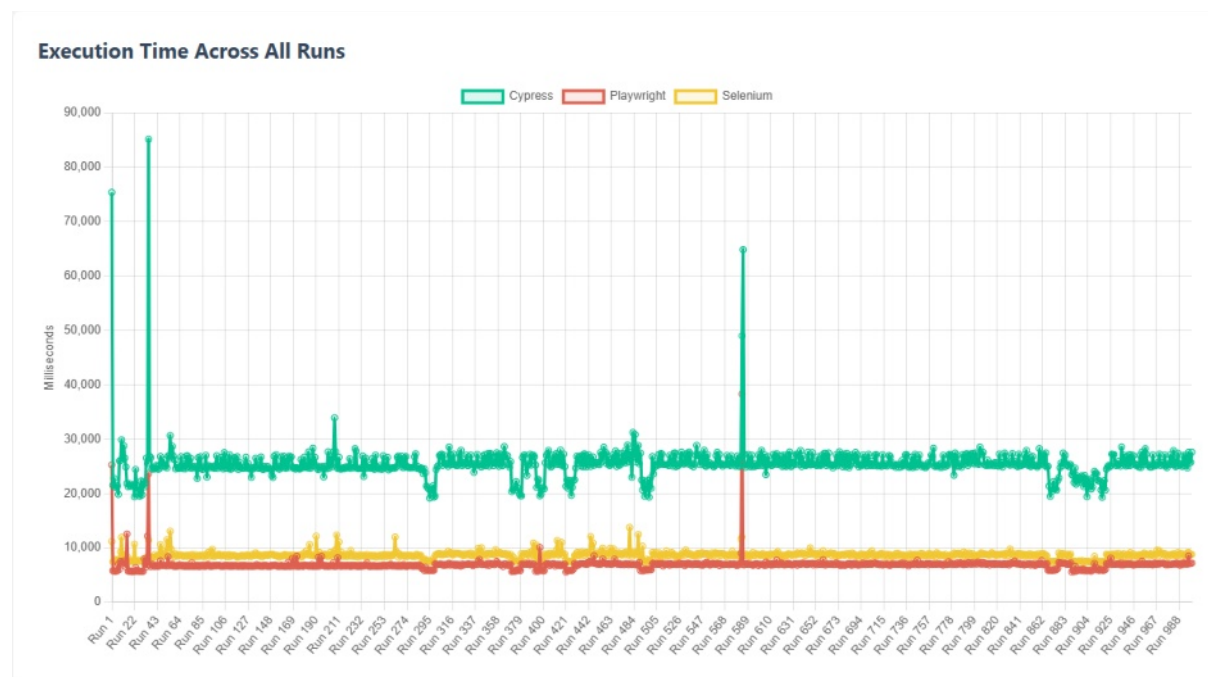
Εύρημα 14 - Τα εργαλεία διαφέρουν σε εύρος (scope), όχι μόνο σε σύνταξη. Τα Cypress και Playwright είναι πλήρη testing frameworks (test runner, assertion library, network interception, auto-wait, reporting, CI integration). Το Selenium είναι browser automation library - παρέχει πρωτόκολλο επικοινωνίας με τον browser. Όλα τα υπόλοιπα είναι ευθύνη του προγραμματιστή. Η επιλογή δεν είναι ερώτημα προτίμησης αλλά αρχιτεκτονικό ερώτημα: πόση υποδομή είναι διατεθειμένη να κατασκευάσει και να συντηρήσει η ομάδα;

Παράρτημα Γ: Οπτικοποίηση Αποτελεσμάτων

Αποτελέσματα n=1000 runs, οπτικοποιημένα αξιοποιώντας το charts.html

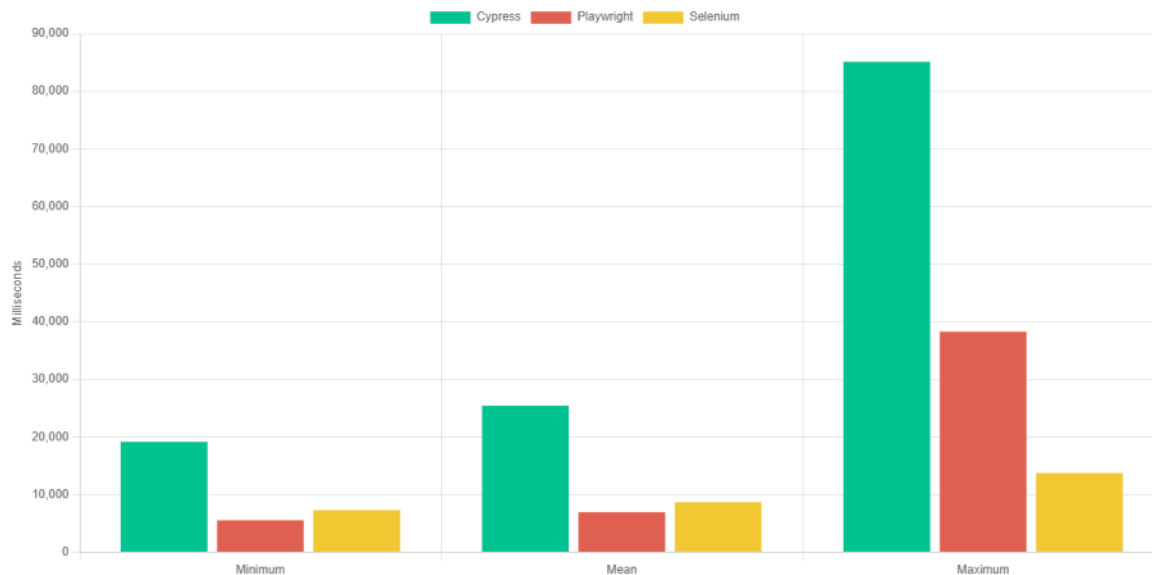


Εικόνα Γ.1: Μέσος χρόνος εκτέλεσης ανά εργαλείο



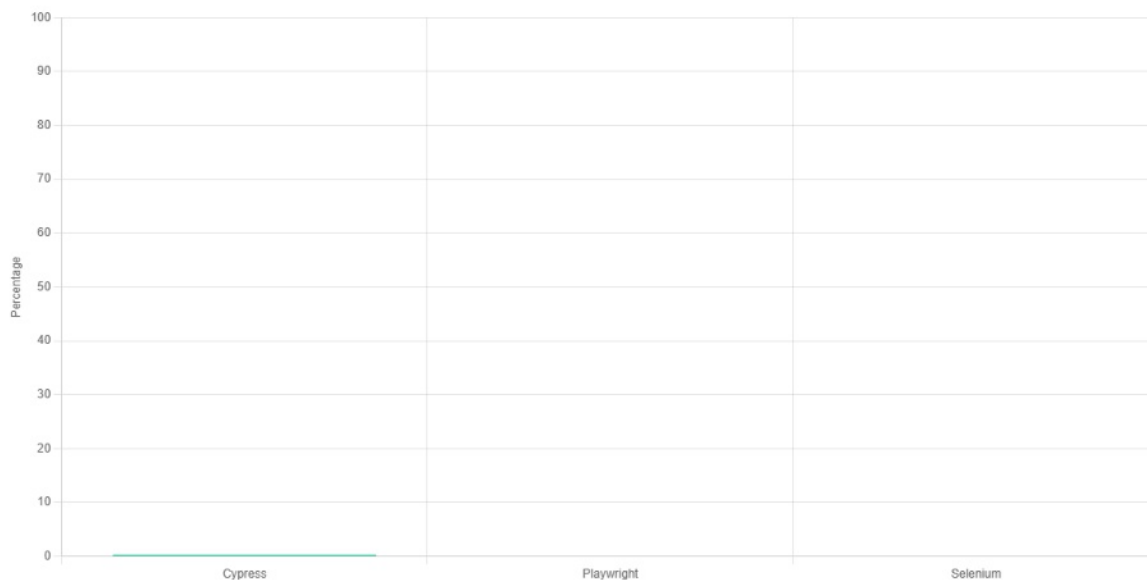
Εικόνα Γ.2: Χρόνος εκτέλεσης για n=1000

Distribution — Min / Mean / Max

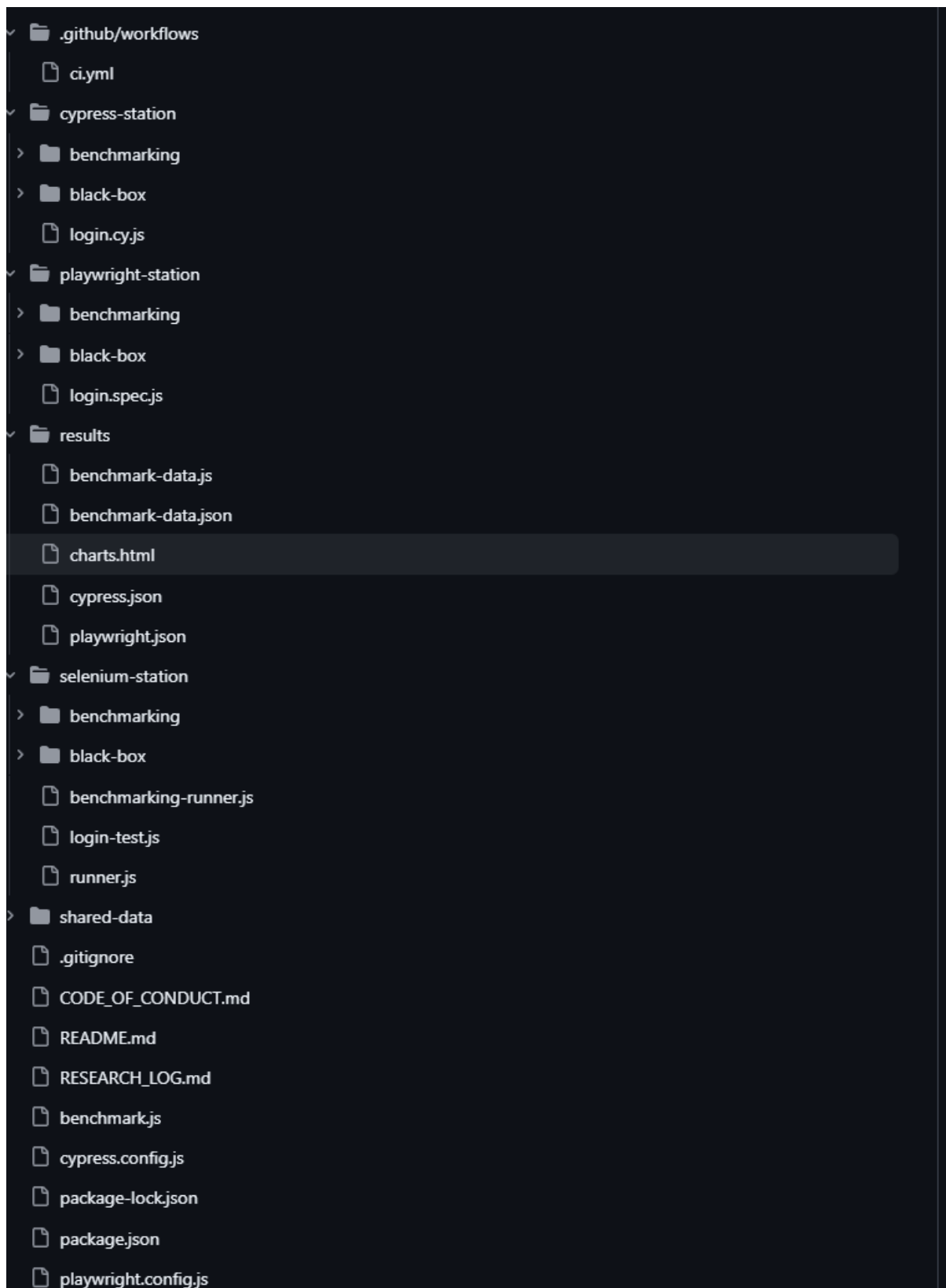


Εικόνα Γ.3: Κατανομές Min / Mean / Max

Flakiness Rate per Framework (% Failed Runs)



Εικόνα Γ.4: Ποσοστό Flakiness



Εικόνα Γ.5: Δομή Αποθετηρίου

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.