# ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ


# Σύγκριση Μεταξύ των Πλαισίων Δοκιμής Ιστού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της


Χαμακιώτη Ελένης

**Επιβλέπων: Κρητικός Κυριάκος**

**Μέλη εξεταστικής επιτροπής: Κοκολάκης Σπύρος - Χαραλαμπίδης Γιάννης**


Σάμος, Οκτώβριος, 2025

**© 2025**

**της**

**Χαμακιώτη Ελένης**

**Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ**

# Ευχαριστίες

Θα ήθελα να εκφράσω την εκτίμησή μου σε όλους όσους συνέλαβαν στην ολοκλήρωση αυτής της διπλωματικής μου εργασίας, προσφέροντας τη βοήθεια, τις γνώσεις ή την υποστήριξή τους. Ιδιάιτερες ευχαριστίες απευθύνω στον επιβλέποντα καθηγητή μου, κύριο Κρητικό Κυριάκο, για την καθοδήγηση, τις πολύτιμες συμβουλές και την άμεση ανταπόκρισή του καθ' όλη την διάρκεια της εργασίας. Ευχαριστώ επίσης την οικογένεια μου και τους φίλους μου για την στήριξή τους όλα αυτά τα χρόνια.

# Περίληψη

Η παρούσα διπλωματική εργασία εστιάζει στην μελέτη και σύγκριση των κύριων πλαισίων δοκιμής εφαρμογών ιστού που χρησιμοποιούνται ευρέως από εταιρείες λογισμικού. Ειδικότερα, σκοπός της εργασίας είναι να αναλύσει τα χαρακτηριστικά, τα πλεονεκτήματα και τα μειονεκτήματα των πιο δημοφιλών πλαισίων δοκιμής, όπως είναι το Selenium και το Cypress. Η ανάλυση αυτή εστιάζει κυρίως στη λειτουργικότητα, την ευκολία χρήσης, την υποστήριξη για διάφορους φυλλομετρητές και τις δυνατότητες αυτοματοποίησης των πλαισίων αυτών. Μέσω της ανάλυσης αυτής, παρέχονται χρήσιμες πληροφορίες και κατευθύνσεις για τους επαγγελματίες του χώρου της τεχνολογίας λογισμικού, ώστε να επιλέξουν το κατάλληλο πλαίσιο δοκιμής για τις ανάγκες ανάπτυξης των εφαρμογών τους. Η εργασία φιλοδοξεί να λειτουργήσει ως οδηγός επιλογής πλαισίου, βάσει αντικειμενικών κριτηρίων αξιολόγησης. Πέραν της ανάλυσης των υφιστάμενων εργαλείων, η εργασία εντοπίζει ερευνητικά κενά στον τομέα των αυτοματοποιημένων δοκιμών, όπως η έλλειψη εμπειρικών δεδομένων για τη χρήση πλαισίων σε περιβάλλοντα CI/CD, και η απουσία κριτηρίων που αντανακλούν σύγχρονες τεχνολογικές τάσεις, όπως το AI testing και η ενσωμάτωση με cloud περιβάλλοντα. Ως εκ τούτου, προτείνονται συγκεκριμένες μελλοντικές κατευθύνσεις για την επέκταση της έρευνας και την περαιτέρω βελτίωση των μεθόδων αξιολόγησης.

**Λέξεις-κλειδιά:** πλαίσια δοκιμών, αυτοματοποίηση, εφαρμογές ιστού, ανάλυση, αξιολόγηση, οδηγός επιλογής πλαισίων

# Abstract

This thesis focuses on the study and comparison of the main web application testing frameworks widely used by software companies. Specifically, the aim of the thesis is to analyse the features, advantages, and disadvantages of the most popular testing frameworks, such as Selenium and Cypress. This analysis primarily emphasizes functionality, ease of use, support for various browsers, and the automation capabilities of these frameworks. Through this analysis, valuable insights and guidance are supplied to software technology professionals to assist them in selecting the most suitable testing framework for their application development needs. This thesis ambitions to operate as a guide towards selecting web application testing frameworks based on objective evaluation criteria. In addition to the analysis, the thesis identifies existing research gaps in the field of automated testing, such as the lack of empirical performance data in CI/CD environments and the absence of modern evaluation criteria reflecting current trends like AI-assisted testing and cloud integration. As a result, it proposes targeted future research directions to address these gaps and further enhance the reliability and relevance of testing framework evaluations.

**Keywords:** test frameworks, test automation, web applications, analysis, evaluation, framework selection guide

# Πίνακας Περιεχομένων

1. Εισαγωγή ..... 10
2. Δοκιμές Λογισμικού ..... 14
2.1. Ορισμός ..... 14
2.2 Τα είδη δοκιμών λογισμικού ..... 14
2.3 Αυτοματοποίηση δοκιμών ..... 16
2.4 Πλαίσια Δοκιμών ..... 18
2.4 Μεθοδολογίες και Μοτίβα Σχεδίασης Πλαίσιων Δοκιμών ..... 23
3. Σχετικές εργασίες ..... 26
3.1 A comparative study of automation testing tools for web applications ..... 26
3.2 A Study of Automated Software Testing: Automation Tools and Frameworks ..... 27
3.3 Comparison Analysis of Android GUI Testing Frameworks by Using an Experimental Study ..... 30
3.4 Taxonomy of Automated Software Testing Tools ..... 35
3.5 Web App Security - A Comparison and Categorization of Testing Frameworks ..... 36
3.6 Η δική μας συνεισφορά ..... 36
4. Ανάλυση Πλαισίων Δοκιμής Εφαρμογών Ιστού ..... 38
4.1 Ανάλυση Κριτηρίων και Μεθοδολογία Βαθμολόγησης ..... 39
4.2 Το Junit ..... 45
4.3 Το TestNG ..... 46
4.4 Το PyTest ..... 48
4.5 Το Selenium ..... 50
4.6 Το Cypress ..... 53
4.7 Το Playwright ..... 56
4.8 Το TestCafe ..... 57
4.9 Το Appium ..... 59
4.10 Το Cucumber ..... 61
4.11 Το Gauge ..... 63
4.12 Το Robot ..... 64
4.13 Σύνοψη και Συμπεράσματα ..... 65
4.13.1 Διαχωρισμός των Πλαισίων σε Ομάδες ..... 67
4.13.2 Ομάδα 1: Πλαίσια Δοκιμών Μονάδας και Ενοποίησης ..... 68
4.13.3 Ομάδα 2: Πλαίσια Δοκιμών UI / Συστήματος / End-to-End ..... 68
4.13.4 Ομάδα 3: Πλαίσια BDD / Δοκιμών Αποδοχής ..... 68
4.13.5 Καθολικά Συμπεράσματα ..... 69

5. Ερευνητικά Κενά και Μελλοντικές Κατευθύνσεις ..... 72
5.1 Κενό 1: Έλλειψη εμπειρικών δεδομένων από περιβάλλοντα CI/CD ..... 72
5.2 Κενό 2: Απουσία διαδραστικών εργαλείων υποστήριξης επιλογής ..... 72
5.3 Κενό 3: Περιορισμός στην τεχνική – όχι εμπειρική – αξιολόγηση ..... 73
5.4 Κενό 4: Ανάγκη εμπλουτισμού του μοντέλου αξιολόγησης ..... 73
5.5 Εσωτερικές Επεκτάσεις της Παρούσας Εργασίας ..... 74
6. Συμπεράσματα ..... 76
6.1 Συνεισφορά της Εργασίας και Πλεονεκτήματα ..... 76
6.2 Δεξιότητες και εμπειρία που αποκομίσθηκαν ..... 77
Βιβλιογραφία ..... 79

# Λίστα Πινάκων


| **Αριθμός Πίνακα** | **Όνομα Πίνακα** | **Ενότητα** | **Σελίδα** |
|---|---|---|---|
| Πίνακας 1 | Σύγκριση πλαισίων δοκιμών | 3.2 | 28 |
| Πίνακας 2 | Συνοπτική παρουσίαση της αξιολόγησης πλαισίων αυτοματοποίησης δοκιμών σε γενικά και τεχνικά κριτήρια | 3.3 | 31 - 34 |
| Πίνακας 3 | Παρουσίαση των αποτελεσμάτων αξιολόγησης βάσει κριτηρίων σύγκρισης για κάθε πλαίσιο | 4.13 | 66 |

# Λίστα Εικόνων


| **Αριθμός Εικόνας** | **Όνομα Εικόνας** | **Ενότητα** | **Σελίδα** |
|---|---|---|---|
| Εικόνα 1 | Η αρχιτεκτονική δομή ενός πλαισίου δοκιμής εφαρμογών, με απεικόνιση των βασικών και προαιρετικών επιπέδων καθώς και των μεταξύ τους διασυνδέσεων | 2.4 | 22 |
| Εικόνα 2 | Η ιεραρχική δομή των διαστάσεων και κριτηρίων αξιολόγησης των πλαισίων δοκιμής εφαρμογών ιστού. | 4.1 | 44 |
| Εικόνα 3 | Διάγραμμα ροής – οδηγός επιλογής κατάλληλων πλαισίων | 4.13.2 | 71 |

# 1. Εισαγωγή

Το 1990, ο Sir Timothy John Berners-Lee, επιστήμονας του CERN, δημιούργησε ένα πρόγραμμα ονομαζόμενο ως WorldWideWeb που αποτελούσε συνάμα τόσο μια πρώτη εφαρμογή ιστού (web application) όσο και έναν φυλλομετρητή (browser) [1]. Αυτή η διττή φύση του προγράμματος οφειλόταν στην ικανότητά του όχι μόνο να περιηγείται σε ηλεκτρονικά έγγραφα, τα οποία συνδέονταν μεταξύ τους μέσω υπερσυνδέσμων (hyperlinks), αλλά και να τα τροποποιεί. Το πρόγραμμα αυτό απετέλεσε το πρώτο βήμα για τη δημιουργία του Παγκόσμιου Ιστού (World Wide Web), όπως τον γνωρίζουμε σήμερα [1]. Τρία έτη αργότερα, το 1993, ο Marc Andreessen δημιούργησε τον «Mosaic», το δεύτερο πρόγραμμα περιήγησης (browser), και έτσι το ευρύ κοινό απέκτησε πρόσβαση στον «ιστό» [2]. Από αυτό το σημείο και έπειτα, η ανάπτυξη εφαρμογών ιστού (web applications) έγινε πιο εκτεταμένη.

Οι εφαρμογές ιστού έχουν ενσωματωθεί στην καθημερινότητά μας και αποτελούν αναγκαία εργαλεία για τον καθένα μας [3]. Έχουν επιφέρει αλλαγές στον τρόπο που εργαζόμαστε, επικοινωνούμε, εκπαιδευόμαστε κ.α.. Οι οργανισμοί τις χρησιμοποιούν για τις εσωτερικές τους λειτουργίες, τη διαχείριση του πελατολογίου, την πώληση προϊόντων και υπηρεσιών, καθώς και για διαφήμιση [4]. Συμβάλλουν αποτελεσματικά στο εκπαιδευτικό έργο, παρέχοντας διαδικτυακές πλατφόρμες μάθησης, εκπαιδευτικά εργαλεία, προσομοιώσεις και υλικό προς μελέτη. Η κοινωνική δικτύωση, μέσω της οποίας η επικοινωνία και η ενημέρωση είναι γρηγορότερες και πιο άμεσες από ποτέ, οφείλει την ύπαρξή της σε εφαρμογές ιστού [4]. Η τραπεζική, είναι ακόμα ένας κλάδος που έχει ανθίσει μέσω τέτοιων εφαρμογών. Ακόμα και ο τομέας της υγείας εξελίσσεται με τη χρήση υπηρεσιών και εργαλείων πρόληψης ασθενειών, την τηλεϊατρική και άλλα είδη εφαρμογών (ιστού).

Εφ’ όσον, όμως, ο ρόλος και η σημασία των εφαρμογών ιστού είναι τόσο σημαντικοί, είναι λογικό και οι εταιρείες λογισμικού ή οι ομάδες προγραμματιστών που παράγουν τις εφαρμογές αυτές να επιδιώκουν το βέλτιστο δυνατό αποτέλεσμα, δηλαδή την παραγωγή ποιοτικών και αξιόπιστων εφαρμογών που ικανοποιούν τις

απαιτήσεις που έχουν τεθεί για αυτές [5]. Κάθε εφαρμογή αποτελεί υλοποίηση μιας ιδέας ή/και εξυπηρέτηση μίας ή περισσότερων αναγκών. Αναπτύσσεται για συγκεκριμένο σκοπό ή σκοπούς ενώ το αποτέλεσμα της ανάπτυξης αυτής δεν είναι πάντα το επιθυμητό ή το βέλτιστο.

Οι κυριότεροι λόγοι για τους οποίους συμβαίνει αυτό είναι οι ασαφείς απαιτήσεις που καθορίζονται κατά τη σχεδίαση, η έλλειψη επικοινωνίας μεταξύ των μελών της ομάδας ανάπτυξης και του πελάτη ή και των χρηστών, η κακή ή λανθασμένη σχεδίαση, η χαμηλή ποιότητα κώδικα που οδηγεί σε σφάλματα (bugs) και η αντίσταση σε αλλαγές από την πλευρά των χρηστών, οι οποίοι ίσως είναι απρόθυμοι να χρησιμοποιήσουν μία εφαρμογή εάν εκείνη δεν ανταποκρίνεται επακριβώς στις συνήθειές τους. Επίσης, σε προσωποποιημένα έργα λογισμικού, οι περιορισμοί στον προϋπολογισμό και στην ημερομηνία παράδοσής του λογισμικού μπορεί να είναι στενοί, οδηγώντας πολλές φορές σε χαμηλή ποιότητα όλων των τεχνημάτων ανάπτυξης, λόγω της πίεσης που δημιουργείται στην ομάδα έργου και της έλλειψης επαρκών πόρων για την ανάπτυξη.

Ήταν λοιπόν εύλογο να δημιουργηθούν εργαλεία δοκιμών που ελέγχουν τη λειτουργικότητα και συμπεριφορά των εφαρμογών (με αυτοματοποιημένο ή ήμι-αυτοματοποιημένο τρόπο) καθώς και τα μη λειτουργικά τους χαρακτηριστικά και έπειτα τροφοδοτούν με τα σχετικά αποτελέσματα – δεδομένα ελέγχου την ομάδα έργου, έτσι ώστε αυτή να προχωρήσει σε κατάλληλες τροποποιήσεις ή διορθώσεις στην υλοποίηση (ευθύνη των προγραμματιστών) και ενδεχομένως και στη σχεδίαση της εφαρμογής (ευθύνη των αναλυτών/σχεδιαστών και των αρχιτεκτόνων) [5]. Καθένα από τα εργαλεία αυτά διαφέρει και μπορεί να υπερέχει έναντι των υπολοίπων για συγκεκριμένα είδη εφαρμογών ή για συγκεκριμένα είδη απαιτήσεων ή προτιμήσεων. Για αυτό τον λόγο, αρκετές μελέτες έχουν επικεντρωθεί στην ανάλυση διαφόρων πλαισίων δοκιμών λογισμικού, ειδικότερα ως προς τα κύρια χαρακτηριστικά και πλεονεκτήματά τους, όμως αποτυγχάνουν να κατευθύνουν στην επιλογή του κατάλληλου πλαισίου ανάλογα με τις ανάγκες και προτιμήσεις του εκάστοτε οργανισμού.

Η παρούσα αναφορά έρχεται να καλύψει αυτό το κενό, προσφέροντας μια συστηματική και λεπτομερή ανάλυση των πλαισίων δοκιμών λογισμικού, με ιδιαίτερη έμφαση στα εργαλεία που απευθύνονται στις εφαρμογές ιστού. Η βασική μας

συνεισφορά έγκειται στην παροχή ενός δομημένου οδηγού επιλογής, ο οποίος υπερβαίνει την απλή περιγραφή χαρακτηριστικών. Συγκεκριμένα, αναλύουμε τα πλεονεκτήματα και μειονεκτήματα κάθε πλαισίου, συσχετίζοντάς τα άμεσα με πραγματικές ανάγκες και προτιμήσεις οργανισμών (όπως το κόστος, η ευκολία εκμάθησης, η υποστήριξη συγκεκριμένων τεχνολογιών και η κοινότητα).

Το κύριο όφελος για τον αναγνώστη είναι ότι, μετά τη μελέτη της αναφοράς, θα έχει στη διάθεσή του τα απαραίτητα κριτήρια και την τεκμηριωμένη σύγκριση για να λάβει μια ενημερωμένη απόφαση σχετικά με το βέλτιστο πλαίσιο δοκιμών για το δικό του έργο ή οργανισμό. Αυτή η προσέγγιση στοχεύει στη μείωση του χρόνου και του κόστους που απαιτείται για την επιλογή και την υιοθέτηση εργαλείων δοκιμών, συμβάλλοντας τελικά στην παραγωγή υψηλότερης ποιότητας εφαρμογών ιστού.

Μέσω αυτής της αναφοράς, αρχικά (Κεφάλαιο 2) θα περιγράψουμε τα είδη των δοκιμών λογισμικού και θα αναφέρουμε τα πλεονεκτήματα της αυτοματοποίησης αυτών και έπειτα θα αναφερθούμε στα πλαίσια δοκιμών και στα επίπεδα που αυτά στοχεύουν. Στη συνέχεια (Κεφάλαιο 3) θα αναλύσουμε υπάρχουσες εργασίες που εστιάζουν στην παρουσίαση και σύγκριση πλαισίων δοκιμών. Έπειτα, θα προσδιορίσουμε τον τρόπο αναζήτησης και σύγκρισης των πλαισίων δοκιμών (Κεφάλαιο 4) ενώ κατόπιν (Κεφάλαιο 5) θα αναφερθούμε στα εντοπισμένα πλαίσια δοκιμών, θα περιγράψουμε τα ειδικά χαρακτηριστικά καθενός, καθώς και τα πλεονεκτήματα ή τα μειονεκτήματα που έχουν έναντι των άλλων. Στόχος μας είναι αυτή η αναλυτική παρουσίαση να αποτελέσει οδηγό επιλογής κατάλληλου πλαισίου δοκιμών από οποιοδήποτε ενδιαφερόμενο με βάση τις δικές του ανάγκες και προτιμήσεις. Τέλος, θα περιγράψουμε ορισμένα γενικά συμπεράσματα και θα παρέχουμε ορισμένες κατευθύνσεις προς μελλοντική έρευνα αλλά και επέκταση της τρέχουσας εργασίας στο τελευταίο κεφάλαιο της τρέχουσας αναφοράς.

# 2. Δοκιμές Λογισμικού

## 2.1. Ορισμός

Η δοκιμή λογισμικού αποτελεί εκείνο το στάδιο στη διαδικασία ανάπτυξης του λογισμικού, όπου το λογισμικό υποβάλλεται σε δοκιμές (tests) με σκοπό την αξιολόγηση της λειτουργικότητας και της ποιότητάς του, καθώς και της συμμόρφωσής του με τις προδιαγραφές που είχαν τεθεί αρχικά για αυτό. Από τον ορισμό αυτό φαίνεται ότι η δοκιμή λογισμικού δεν αρκείται στον εντοπισμό και στη διόρθωση τυχόν σφαλμάτων (bugs) κατά τη διάρκεια της ανάπτυξης, αλλά έμφαση δίνεται και στην εισαγωγή βελτιώσεων, ώστε η συνολική ποιότητα να αυξηθεί σε κατάλληλα πλαίσια πριν την τελική παράδοση της εφαρμογής στο χρήστη [5]. Οι τρόποι εκτέλεσης των δοκιμών λογισμικού διακρίνονται σε δύο κατηγορίες. Από τη μία έχουμε την *αυτοματοποιημένη εκτέλεση*, μέσω της οποίας οι δοκιμές έχουν την μορφή κώδικα και εκτελούνται αυτόματα με τη χρήση ειδικών βιβλιοθηκών (πχ. JUnit για την Java), ενώ από την άλλη έχουμε *χειροκίνητη εκτέλεση* δοκιμών από την ομάδα έργου (ή από ειδική ομάδα διασφάλισης ποιότητας της εταιρείας λογισμικού) [6][7].

## 2.2 Τα είδη δοκιμών λογισμικού

Τα είδη δοκιμών λογισμικού μπορούν να χωριστούν σε δύο κατηγορίες, στις λειτουργικές και στις μη λειτουργικές δοκιμές [8].

**Λειτουργικές δοκιμές**:

Οι λειτουργικές δοκιμές εστιάζουν στον έλεγχο της σωστής λειτουργίας και συμπεριφοράς του λογισμικού σύμφωνα με τις (λειτουργικές) απαιτήσεις και τις προδιαγραφές του. Ειδικότερα, αξιολογούν κατά πόσο το σύστημα (λογισμικού) εκτελεί τις προδιαγεγραμμένες λειτουργίες του σωστά και χωρίς σφάλματα [8]. Οι σημαντικότερες από αυτές τις δοκιμές είναι οι εξής:

- Δοκιμή Μονάδας (Unit Testing): Ελέγχονται μεμονωμένες μονάδες ή τμήματα του κώδικα (π.χ. μέθοδοι) για να διασφαλιστεί ότι λειτουργούν σωστά.

- Δοκιμή Ενσωμάτωσης (Integration Testing): Ελέγχεται η σωστή αλληλεπίδραση μεταξύ διαφορετικών μονάδων ή συστατικών που συνδυάζονται για να σχηματίσουν ένα μέρος του συστήματος.
- Δοκιμή Συστήματος (System Testing): Ελέγχεται ολόκληρο το σύστημα για να διασφαλιστεί ότι πληροί τις προκαθορισμένες (λειτουργικές) απαιτήσεις.
- Δοκιμή Παλινδρόμησης (Regression Testing): Ελέγχεται το κατά πόσο νέες αλλαγές (που πρόκειται να υποβληθούν) στο σύστημα θα επηρεάσουν αρνητικά τις υπάρχουσες λειτουργίες του (όπως αυτές έχουν υλοποιηθεί μέχρι στιγμής).
- Δοκιμή Αποδοχής (Acceptance Testing): Γνωστή και ως User Acceptance Testing (UAT) (Δοκιμή Αποδοχής Χρηστών). Αποσκοπεί στη διασφάλιση ότι το σύστημα πληροί τις καθορισμένες απαιτήσεις και τις προσδοκίες των χρηστών, καθώς και ότι είναι φιλική ως προς αυτούς. Εδώ αποφασίζεται και το αν το σύστημα είναι έτοιμο για κυκλοφορία και άρα μπορεί να χρησιμοποιηθεί από τους χρήστες/πελάτες του.

Σημειώνεται ότι οι δοκιμές μονάδας και ενσωμάτωσης στοχεύουν κυρίως στις λειτουργικές απαιτήσεις, ωστόσο σε ορισμένες περιπτώσεις μπορεί να καλύψουν και μη λειτουργικά χαρακτηριστικά. Αντίστοιχα, οι δοκιμές συστήματος και αποδοχής, εκτός από λειτουργικές απαιτήσεις, αξιολογούν και μη λειτουργικά χαρακτηριστικά του λογισμικού, όπως η απόδοση, η ασφάλεια και η χρηστικότητα, καθώς εφαρμόζονται στο λογισμικό ως σύνολο.

**Μη λειτουργικές δοκιμές**:

Σε αντίθεση με τις λειτουργικές δομικές, εδώ ο στόχος είναι η αξιολόγηση του συστήματος ως προς τις μη λειτουργικές του πτυχές, όπως για παράδειγμα την απόδοση, την ασφάλεια και τη χρηστικότητα [8]. Οι σημαντικότερες μη λειτουργικές δοκιμές είναι οι εξής:

- Δοκιμές Απόδοσης (Performance): Αυτές οι δοκιμές εστιάζουν στον έλεγχο της απόδοσης του λογισμικού όσον αφορά την ρυθμό διεκπεραίωσης (throughput), την αποκριτικότητα (responsiveness) και τον χρόνο εκτέλεσής του (execution time) υπό ποικίλες αλλά κανονικές συνθήκες φόρτου (stress test).

- Δοκιμή Αντοχής (Stress Testing): Εδώ το σύστημα υποβάλλεται σε φόρτο που ξεπερνά τις φυσιολογικές συνθήκες λειτουργίας του, συχνά προσεγγίζοντας ή ξεπερνώντας τα μέγιστα όρια χωρητικότητάς. Αυτό επιτυγχάνεται με τη χρήση εξαιρετικά μεγάλων όγκων δεδομένων και τη δημιουργία μεγάλου αριθμού ταυτόχρονων αιτημάτων. Σκοπός της δοκιμής είναι ο εντοπισμός σημείων αποτυχίας, ο καθορισμός των ορίων αντοχής του συστήματος, η αξιολόγηση της ανταπόκρισής του υπό πίεση και η βελτίωση της ανθεκτικότητάς του, ώστε να μπορέσει να αντέξει μεγαλύτερες πιέσεις στο μέλλον.
- Δοκιμές Συμβατότητας (Compatibility): Αυτές οι δοκιμές εστιάζουν στο να ελέγξουν αν το λογισμικό λειτουργεί κανονικά σε διάφορα περιβάλλοντα, τα οποία μπορούν να περιλαμβάνουν διαφορετικές διαμορφώσεις υλικού, πλατφόρμες, λειτουργικά συστήματα, βάσεις, δίκτυα και συσκευές.
- Δοκιμές Ασφάλειας (Security): Στόχος είναι ο έλεγχος της ασφάλειας του λογισμικού, των εξαρτήσεών του και των διαμορφώσεών του. Εντοπίζονται ευπάθειες (σε αυτά τα τρία προαναφερθέντα μέρη/στοιχεία) και απειλές (πχ., ευκαιρίες ή δυνητικά σημεία εισβολής (hotspots)) ώστε έπειτα να ληφθούν κατάλληλα προστατευτικά μέτρα ασφαλείας.
- Δοκιμές Ευχρηστίας (Usability Testing): Τέλος, έχουμε τις δοκιμές ευχρηστίας με τις οποίες επιχειρούμε να ελέγξουμε το λογισμικό από τη σκοπιά του τελικού χρήστη και όχι από εκείνη του δημιουργού, με σκοπό να αποτιμηθεί πόσο εύχρηστο και διαισθητικό είναι το λογισμικό όταν χρησιμοποιοιείται από τους τελικούς του χρήστες. Είναι σύνηθες οι δοκιμές αυτές να πραγματοποιούνται από τους ίδιους τους χρήστες. Εκείνοι ανατροφοδοτούν την ομάδα έργου με προτάσεις βελτίωσης, έτσι ώστε το λογισμικό να ανταποκρίνεται στις ανάγκες τους. Η ανατροφοδότηση μπορεί να αφορά προβλήματα με την διεπαφή χρήσης (user interface) ή την εμπειρία χρήστη (user experience), την λειτουργικότητα που προσφέρεται ή την μη ικανοποίηση ορισμένων αναγκών των χρηστών.

## 2.3 Αυτοματοποίηση δοκιμών

Όπως αναφέρθηκε και στον ορισμό στην Ενότητα 2.1, οι δοκιμές λογισμικού μπορούν να εκτελούνται είτε αυτοματοποιημένα, είτε χειροκίνητα. Η διαφορά φυσικά είναι ότι στη χειροκίνητη εκτέλεση δοκιμών, απαιτείται η συμμετοχή του ανθρώπινου

παράγοντα, ο οποίος θα πρέπει τόσο να παράγει όσο και να εκτελέσει χειροκίνητα τις δοκιμές με δεδομένα που κατασκευάζει ο ίδιος, τα οποία μοιάζουν με τα πραγματικά. Αυτό προφανώς οδηγεί σε σημαντική καθυστέρηση στην φάση/στάδιο της δοκιμής του λογισμικού. Εκ τούτου, η αυτοματοποίηση δοκιμών είναι ένας σημαντικός πυλώνας στη σύγχρονη διαδικασία ανάπτυξης λογισμικού. Επιτρέπει στις ομάδες ανάπτυξης να επιτύχουν υψηλή ποιότητα και ευελιξία στον κύκλο ζωής του λογισμικού ενώ συνάμα να επιταχύνουν την φάση της δοκιμής.

Τα πλεονεκτήματα της αυτοματοποιημένης εκτέλεσης δοκιμών είναι κατά βάση τέσσερα. Το πρώτο αφορά στην αύξηση της αποδοτικότητας και της ταχύτητας εκτέλεσης των δοκιμών. Είναι φυσικό μία αυτόματη διαδικασία που χρειάζεται ελάχιστη αλληλεπίδραση με τον άνθρωπο, να μπορεί να εκτελείται γρηγορότερα από την αντίστοιχη χειροκίνητη, επιτρέποντας έτσι τη διεξαγωγή περισσότερων δοκιμών σε μικρότερο χρονικό διάστημα. Αυτό είναι ιδιαίτερα χρήσιμο σε μεγάλα έργα λογισμικού, όπου οι χειροκίνητες δοκιμές θα απαιτούσαν πολύ χρόνο και πόρους.

Το δεύτερο πλεονέκτημα είναι εκείνο της ακρίβειας και της συνέπειας. Μία αυτοματοποιημένη δοκιμή μπορεί να εκτελείται πάντα με τον ίδιο τρόπο και έτσι εξαλείφεται η πιθανότητα ύπαρξης ανθρώπινου λάθους (human error), λόγω παράλειψης, κόπωσης ή άλλων παραγόντων.

Επίσης, οι αυτοματοποιημένες δοκιμές μπορούν να εκτελεστούν επανειλημμένα με τα ίδια δεδομένα εισόδου, εξασφαλίζοντας έτσι συνεπή αποτελέσματα και επιτρέποντας τη σύγκριση των αναμενόμενων και παραγόμενων αποτελεσμάτων για τον εντοπισμό λανθάνουσας συμπεριφοράς στο λογισμικό.

Το τρίτο πλεονέκτημα αφορά την καλύτερη κάλυψη και το βάθος [8]. Καθώς η αυτοματοποίηση επιτρέπει την εκτέλεση περισσότερων δοκιμών σε λιγότερο χρόνο, καλύπτεται μεγαλύτερο μέρος του κώδικα και έτσι εντοπίζονται σφάλματα που μπορεί να μην είχαν εντοπιστεί μέσα από χειροκίνητες δοκιμές.

Τέλος και σε συνδυασμό των τριών ανωτέρω πλεονεκτημάτων, έρχεται η μακροπρόθεσμη μείωση του κόστους [10]. Παρόλο που η αρχική επένδυση για την ανάπτυξη αυτοματοποιημένων δοκιμών μπορεί να είναι υψηλή, η μακροπρόθεσμη εξοικονόμηση χρόνου και πόρων αντισταθμίζει αυτό το κόστος. Η αυτοματοποίηση

μειώνει το χρόνο που απαιτείται για τις δοκιμές, επιταχύνοντας την ανάπτυξη και τη διάθεση του λογισμικού στην αγορά. Αυτό μπορεί να μειώσει το συνολικό κόστος του έργου και να αυξήσει την ανταγωνιστικότητα της επιχείρησης.

## 2.4 Πλαίσια Δοκιμών

Για την υποστήριξη της αυτοματοποίησης των δοκιμών λογισμικού, έχουν αναπτυχθεί σαφώς καθορισμένα και επεκτάσιμα πλαίσια (frameworks) δοκιμής. Τα πλαίσια αυτά αποτελούνται από εργαλεία και πρακτικές που έχουν σχεδιαστεί για τη διεξαγωγή δοκιμών λογισμικού [9]. Καθορίζουν τόσο τον τρόπο με τον οποίο οι δοκιμές εφαρμογών πρέπει να οργανωθούν και να εκτελεστούν, όσο και τον τρόπο με τον οποίο παράγουν και αναφέρουν τα αποτελέσματα των ελέγχων. Η αρχιτεκτονική των πλαισίων δοκιμών διαφέρει ανάλογα με τον ειδικό σκοπό και τις λειτουργίες που υποστηρίζει το κάθε ένα από αυτά. Ωστόσο, υπάρχουν τρία βασικά επίπεδα, τα οποία αποτελούν μέρος της αρχιτεκτονικής κάθε πλαισίου:

. .

- Επίπεδο Διαχείρισης Δοκιμών (Test Management Layer): Ασχολείται με τον προγραμματισμό, την οργάνωση και τον έλεγχο της διαδικασίας δοκιμών και περιλαμβάνει συστατικά όπως διαχείριση test cases, οργάνωση σε test suites, έλεγχο εκδόσεων (version control) και μηχανισμούς για τη δημιουργία/φόρτωση δεδομένων δοκιμής. Στόχος είναι ο καθορισμός των στόχων και του πεδίου εφαρμογής των δοκιμών, η κατάρτιση χρονοδιαγραμμάτων και ο καθορισμός των φάσεων των δοκιμών, η παρακολούθηση αυτών και η λήψη διορθωτικών μέτρων όπου χρειάζεται. Ειδικότερα, ορίζονται οι περιπτώσεις και τα σενάρια δοκιμών, είτε χειρωνακτικά είτε με τη χρήση ενός εργαλείου διαχείρισης δοκιμών. Στη συνέχεια οι περιπτώσεις δοκιμών οργανώνονται σε σουίτες ή ομάδες, ώστε να γίνεται καλύτερα η διαχείριση της εκτέλεσής τους. Διατίθεται και εφαρμόζεται έλεγχος εκδόσεων έτσι ώστε οι δοκιμές να εξελίσσονται παράλληλα με το λογισμικό και γίνεται διαχείριση των δεδομένων των δοκιμών, τα οποία είτε παράγονται χειρωνακτικά είτε αντλούνται από πηγές, όπως είναι οι βάσεις δεδομένων, αρχεία ή APIs.
- Επίπεδο Εκτέλεσης Δοκιμών (Test Execution Layer): Αφορά στην πραγματική εκτέλεση των δοκιμών σύμφωνα με τα προκαθορισμένα σενάρια και τα σχέδια

δοκιμών είτε κατά απαίτηση είτε στα πλαίσια εκτέλεσης ενός αγωγού CI/CD (Continuous Integration / Continuous Delivery). Σε αυτό το επίπεδο συναντάμε δομικά στοιχεία όπως οι μηχανισμοί εκτέλεσης δοκιμών (test runners), τα εκτελέσιμα scripts, οι μονάδες προσομοίωσης εξαρτήσεων (mock/stub) και η υποδομή ρύθμισης του περιβάλλοντος εκτέλεσης. Περιλαμβάνει την εκτέλεση, συνήθως με αυτοματοποιημένο τρόπο, των προκαθορισμένων σεναρίων και περιπτώσεων δοκιμών καθώς και την καταγραφή των αποτελεσμάτων των δοκιμών, συμπεριλαμβανομένων των επιτυχιών και αποτυχιών. Επίσης, περιλαμβάνει τη διαμόρφωση του περιβάλλοντος δοκιμής σε επίπεδο εκτέλεσης, δηλαδή την αρχικοποίηση και ρύθμιση των φυλλομετρητών, των βάσεων δεδομένων και των εξυπηρετητών εφαρμογών (application servers) που απαιτούνται για την υλοποίηση των δοκιμών. Κατά την εκτέλεση, το πλαίσιο μπορεί να κάνει mock ή stub εξωτερικές εξαρτήσεις ώστε να προσομοιώσει τη συμπεριφορά τους και να απομονώσει καλύτερα τα μέρη του συστήματος που δοκιμάζονται. Τέλος, η εκτέλεση των δοκιμών μπορεί να γίνεται σε διαφορετικά modes, ειδικότερα όσον αφορά τη δοκιμή της διεπαφής χρήστη (user interface) του λογισμικού.

- Επίπεδο Ανάλυσης και Δημιουργίας Αναφορών (Test Analysis and Reporting Layer): Επικεντρώνεται στην ανάλυση των αποτελεσμάτων των δοκιμών και στη δημιουργία αναφορών για την αξιολόγηση της ποιότητας του λογισμικού. Το επίπεδο αυτό βασίζεται σε υποσυστήματα όπως οι μηχανισμοί δημιουργίας αναφορών, η καταγραφή σφαλμάτων και logs, η αποθήκευση stack traces και στιγμιότυπων οθόνης, καθώς και η μονάδα υπολογισμού ποσοτικών δεικτών αξιολόγησης. Περιλαμβάνει την ανάλυση των αποτελεσμάτων των δοκιμών για τον εντοπισμό σφαλμάτων και αδυναμιών, τη δημιουργία αναφορών που περιγράφουν τα ευρήματα των δοκιμών, τα προβλήματα που εντοπίστηκαν και τις προτάσεις για βελτιώσεις, καθώς και την παροχή πληροφοριών στους προγραμματιστές, για τη λήψη αποφάσεων. Εκτός από τη σύλληψη της εξόδου της κάθε δοκιμής, μπορεί να καταγράφονται και άλλα στοιχεία, όπως μηνύματα λάθους, ίχνη στοίβας (stack traces), στιγμιότυπα οθόνης (screenshots) και καταχωρήσεις (logs). Επίσης, μπορεί να παρέχονται και ποσοτικές μετρήσεις που μπορεί να αφορούν τον χρόνο εκτέλεσης, τον αριθμό των περασμένων/αποτυχημένων δοκιμών, το ποσοστό κάλυψης του κώδικα και

άλλα. Σε ορισμένες περιπτώσεις, τα πλαίσια μπορεί να παράγουν γραφήματα για την οπτικοποίηση των αποτελεσμάτων δοκιμής, των ποσοστών κάλυψης και τις τάσεις αποτυχιών (failure trends).

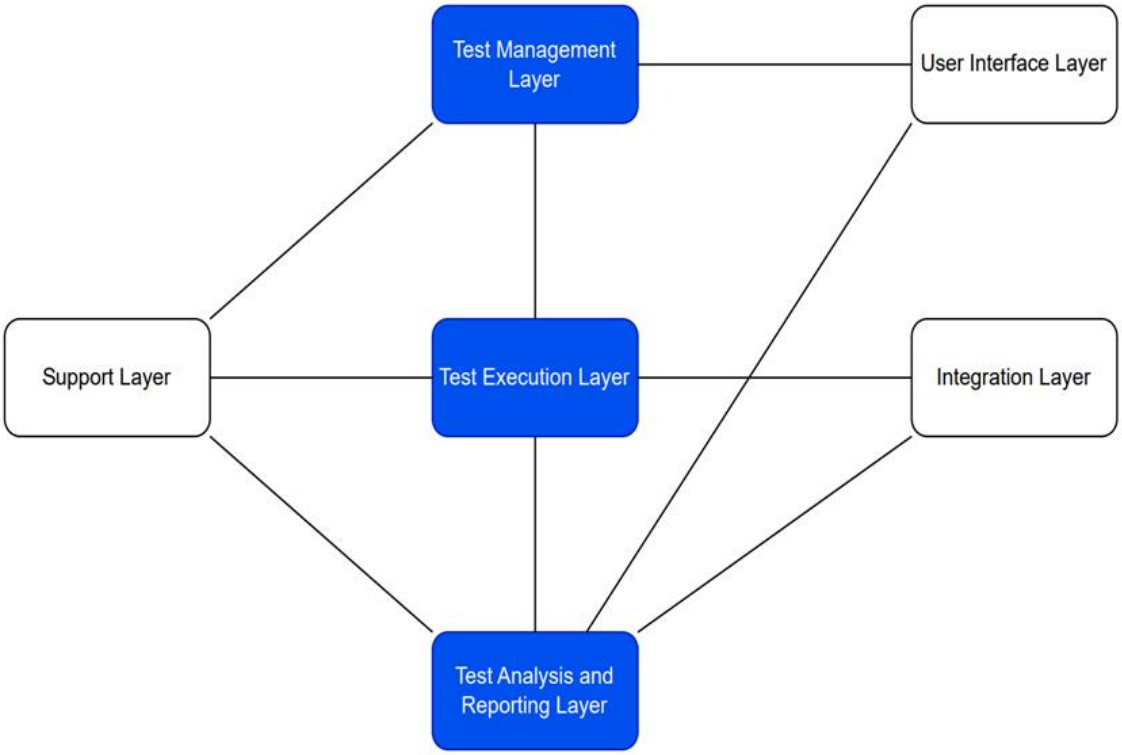
Εκτός από τα ανωτέρω τρία βασικά επίπεδα, τα πλαίσια δοκιμών μπορούν κατά περίπτωση να ενσωματώνουν στην αρχιτεκτονική τους και τα εξής προαιρετικά επίπεδα:

- Επίπεδο Ενοποίησης (Integration Layer): Το επίπεδο ενοποίησης εξασφαλίζει τη σωστή συνεργασία των συστατικών του συστήματος και την ενσωμάτωσή τους σε ευρύτερα περιβάλλοντα ανάπτυξης. Αποτελείται από συστατικά όπως εργαλεία συνεχούς ενσωμάτωσης και παράδοσης (CI/CD), συστήματα ελέγχου έκδοσης (π.χ. Git), υπηρεσίες εντοπισμού σφαλμάτων (bug tracking) και συνδέσεις με υποδομές νέφους (cloud platforms). Υποστηρίζει τη διαδικασία της δοκιμής ενοποίησης, παρέχοντας τους απαραίτητους μηχανισμούς ώστε οι επιμέρους μονάδες να συνδυάζονται λειτουργικά και να προετοιμάζονται για εκτέλεση δοκιμών. Η υποστήριξη CI επιτρέπει την αυτόματη ενεργοποίηση δοκιμών κάθε φορά που υποβάλλεται νέα αλλαγή στον κώδικα, ενώ η ενσωμάτωση με εξωτερικά συστήματα ενισχύει την ευελιξία και την επεκτασιμότητα του πλαισίου. Ειδικότερα, περιλαμβάνονται: εργαλεία συνεχούς ενσωμάτωσης που εκκινούν τις δοκιμές με κάθε υποβολή κώδικα, εργαλεία συνεχούς παράδοσης που μεταφέρουν αυτόματα τον κώδικα σε κατάλληλα περιβάλλοντα μετά από επιτυχείς δοκιμές, διασύνδεση με αποθετήρια όπως το Git για την άντληση αλλαγών προς έλεγχο, bug trackers για τη δημιουργία εισιτηρίων σε περίπτωση αποτυχιών, καθώς και cloud υποδομές που επιτρέπουν την αυτόματη δημιουργία και κλιμάκωση περιβαλλόντων δοκιμής.
- Επίπεδο Υποστήριξης (Support Layer): Το επίπεδο υποστήριξης παρέχει τις απαραίτητες υπηρεσίες και λειτουργίες υποστήριξης των δοκιμών χωρίς να αλληλεπιδρά άμεσα με την κύρια διαδικασία εκτέλεσης. Περιλαμβάνει μηχανισμούς διαχείρισης περιβαλλόντων, mock υπηρεσίες, παραμετροποίηση μεταβλητών και δυνατότητα αυτόματης παραγωγής δεδομένων δοκιμών. Εστιάζει στη μακροπρόθεσμη διαχείριση και συντήρηση των περιβαλλόντων δοκιμών, περιλαμβάνοντας τη δημιουργία, παραμετροποίηση και διατήρηση

διαφορετικών διαμορφώσεων περιβάλλοντος για επαναχρησιμοποίηση. Περιλαμβάνει επίσης την ενοποίηση με υπηρεσίες τρίτων μερών που χρησιμοποιούνται για λόγους δοκιμής (όπως βάσεις δεδομένων, υπηρεσίες mock, πύλες πληρωμών), τη διαχείριση των διαμορφώσεων (όπως μεταβλητές περιβάλλοντος και διαπιστευτήρια), την αυτόματη παραγωγή δεδομένων δοκιμής με βάση κανόνες, καθώς και την εγκαθίδρυση και διαμόρφωση mock/stub μηχανισμών.

- Επίπεδο Διεπαφής Χρήστη (User Interface Layer): Το επίπεδο διεπαφής χρήστη παρέχει το μέσο, μέσω του οποίου οι χρήστες αλληλεπιδρούν με το πλαίσιο δοκιμών. Η διεπαφή χρήσης αυτή, μπορεί να είναι είτε γραφική, είτε γραμμής εντολών. Το επίπεδο επιτρέπει τη δημιουργία, τη διαχείριση και την εκτέλεση των δοκιμών, καθώς και την οπτικοποίηση των αποτελεσμάτων τους. Περιλαμβάνει διεπαφή ορισμού δοκιμών, η οποία διευκολύνει τη δημιουργία και διαχείριση των δοκιμών και των δεδομένων τους. Παρέχει έλεγχο εκτέλεσης δοκιμών, που επιτρέπει την πυροδότηση της εκτέλεσης, τη διαμόρφωση παραμέτρων εκτέλεσης (περιβάλλον, φυλλομετρητής, συσκευή) και την παρακολούθηση της προόδου της εκτέλεσης. Επιπλέον, ενσωματώνει ένα dashboard, το οποίο προσφέρει ανατροφοδότηση σε πραγματικό χρόνο, ανάλυση τάσεων ή σφαλμάτων, παρακολούθηση παρελθοντικών αποτελεσμάτων και οπτικοποίηση αυτών. Τέλος, περιλαμβάνει εργαλεία αποσφαλμάτωσης, τα οποία βοηθούν τους χρήστες να παρατηρήσουν καταχωρήσεις, μηνύματα λάθους και άλλες πληροφορίες αποσφαλμάτωσης για τη διευκόλυνσή τους στην επίλυση προβλημάτων.

Η αρχιτεκτονική διάρθρωση και οι διασυνδέσεις μεταξύ των επιπέδων αποτυπώνονται στην Εικόνα 1.




*Εικόνα 1: Η αρχιτεκτονική δομή ενός πλαισίου δοκιμής εφαρμογών, με απεικόνιση των βασικών και προαιρετικών επιπέδων καθώς και των μεταξύ τους διασυνδέσεων.*

Όπως φαίνεται στην Εικόνα 1, η αρχιτεκτονική ενός πλαισίου δοκιμών βασίζεται σε τρία κύρια επίπεδα: το Test Management Layer, το Test Execution Layer και το Test Analysis and Reporting Layer. Τα επίπεδα αυτά συγκροτούν τον βασικό κορμό του πλαισίου και συνδέονται λογικά μεταξύ τους κατά τη ροή της διαδικασίας ελέγχου. Το Support Layer λειτουργεί εγκάρσια, παρέχοντας κρίσιμες υποδομές στα τρία βασικά επίπεδα, όπως η διαχείριση περιβάλλοντος, mock υπηρεσίες και η δημιουργία δεδομένων δοκιμής. Το User Interface Layer συνδέεται με το Test Management Layer και το Test Analysis and Reporting Layer, επιτρέποντας στον χρήστη να αλληλεπιδρά με το πλαίσιο για τη δημιουργία, παραμετροποίηση και παρακολούθηση δοκιμών μέσω διεπαφής χρήσης (GUI ή CLI). Τέλος, το Integration Layer συνδέεται με το Test Execution Layer και το Test Analysis and Reporting Layer, προσφέροντας υποστήριξη για CI/CD pipelines, σύνδεση με εργαλεία εντοπισμού σφαλμάτων και αποθετήρια κώδικα, διευκολύνοντας έτσι την αυτοματοποίηση και την ενοποίηση με εξωτερικά συστήματα.

## 2.4 Μεθοδολογίες και Μοτίβα Σχεδίασης Πλαίσιων Δοκιμών

Εκτός από την αρχιτεκτονική διάρθρωση ενός πλαισίου δοκιμών, εξίσου κρίσιμος είναι και ο τρόπος με τον οποίο οργανώνονται τα ίδια τα σενάρια δοκιμής στο εσωτερικό του πλαισίου. Οι προσεγγίσεις αυτές διακρίνονται σε μεθοδολογίες σχεδιασμού δοκιμών και σε μοτίβα σχεδίασης δοκιμών, τα οποία συχνά συνδυάζονται για την οικοδόμηση αποτελεσματικών και επαναχρησιμοποιήσιμων αυτοματοποιημένων δοκιμών.

**Μεθοδολογίες σχεδιασμού δοκιμών**

Οι μεθοδολογίες καθορίζουν τον τρόπο με τον οποίο σχεδιάζονται και γράφονται τα σενάρια δοκιμών, ειδικότερα ως προς την πηγή και τη ροή των δεδομένων, την αλληλεπίδραση με τον κώδικα εφαρμογής και τον βαθμό ευκολίας ή επαναχρησιμοποίησης. Οι κυριότερες μεθοδολογίες περιλαμβάνουν:

- *Linear (γραμμική)*: Η πιο απλή και πρώιμη μεθοδολογία, όπου κάθε σενάριο καταγράφεται και εκτελείται με σταθερή σειρά ενεργειών. Δεν υπάρχει επαναχρησιμοποίηση κώδικα και κάθε σενάριο αποτελεί μια αυτόνομη ροή, καθιστώντας τη μέθοδο κατάλληλη για απλά έργα ή αρχική φάση αυτοματοποίησης.
- *Data-driven* (οδηγούμενη από δεδομένα): Εδώ διαχωρίζεται η λογική των δοκιμών από τα ίδια τα δεδομένα, τα οποία αποθηκεύονται σε εξωτερικές πηγές (π.χ. Excel, CSV, SQL). Ένα κοινό σενάριο μπορεί να εκτελεστεί επαναλαμβανόμενα με διαφορετικά σύνολα δεδομένων, προσφέροντας υψηλό επίπεδο παραμετροποίησης και κάλυψης.
- *Keyword-driven* (οδηγούμενη από λέξεις-κλειδιά): Παρόμοια με την προηγούμενη, η προσέγγιση αυτή στηρίζεται στον ορισμό λέξεων-κλειδιών που περιγράφουν επαναλαμβανόμενες ενέργειες. Τα σενάρια δημιουργούνται με χρήση αυτών των λέξεων-κλειδιών, οι οποίες είναι αποθηκευμένες εξωτερικά και ερμηνεύονται κατά την εκτέλεση, μειώνοντας την ανάγκη για προγραμματισμό.

- *Behavior-driven development* (BDD): Εστιάζει στη συγγραφή των δοκιμών με φυσική γλώσσα, συνήθως βάσει του προτύπου Given–When–Then. Χρησιμοποιείται συχνά σε περιβάλλοντα όπου η επικοινωνία μεταξύ αναλυτών, προγραμματιστών και δοκιμαστών είναι κρίσιμη.
- *Mixed* (μεικτή προσέγγιση): Συνδυάζει περισσότερες από μία από τις παραπάνω μεθοδολογίες, ανάλογα με τις ανάγκες και το επίπεδο πολυπλοκότητας της εφαρμογής.

**Μοτίβα σχεδίασης δοκιμών**

Τα μοτίβα σχεδίασης αναφέρονται στο πώς οργανώνεται και δομείται ο κώδικας των δοκιμών, ώστε να υπάρχει επεκτασιμότητα, ευκολία συντήρησης και επαναχρησιμοποίηση.

- *Modular design* (αρθρωτός σχεδιασμός): Οι ενέργειες που σχετίζονται με την εφαρμογή αποσπώνται σε ξεχωριστές μονάδες ή συναρτήσεις. Το κάθε σενάριο χτίζεται μέσω της επαναχρησιμοποίησης αυτών των επιμέρους ενοτήτων, επιτυγχάνοντας χαμηλή σύζευξη και ευκολότερη συντήρηση.
- *Library architecture* (αρχιτεκτονική βιβλιοθηκών): Παρόμοιο με τον αρθρωτό σχεδιασμό, αλλά εδώ οι κοινές λειτουργίες συλλέγονται σε βιβλιοθήκες επαναχρησιμοποιούμενων συναρτήσεων, οι οποίες μπορούν να καλούνται σε διάφορα σενάρια. Το πλεονέκτημα είναι η μείωση του πλεονάζοντος κώδικα και η βελτίωση της ευελιξίας.
- *Page Object Model* (POM): Ιδιαίτερα διαδεδομένο για UI δοκιμές. Διαχωρίζει τη λογική δοκιμών από την παρουσίαση της εφαρμογής. Για κάθε σελίδα ή τμήμα του UI δημιουργείται μια αντίστοιχη κλάση που περιέχει τα στοιχεία και τις ενέργειες που σχετίζονται με αυτή. Με αυτόν τον τρόπο διευκολύνεται η συντήρηση και η ανθεκτικότητα των σεναρίων στις αλλαγές του UI.
- *Layered or MVC-style separation*: Εφαρμόζεται συχνά στις δοκιμές υπηρεσιών ή πολύπλοκων αρχιτεκτονικών. Ο κώδικας των δοκιμών οργανώνεται σε επίπεδα (π.χ. data layer, service layer, presentation layer), ώστε να ακολουθεί την αρχιτεκτονική του ίδιου του συστήματος υπό δοκιμή.

Η κατανόηση και ο διαχωρισμός των παραπάνω μεθοδολογιών και μοτίβων είναι καθοριστικής σημασίας κατά τη φάση του σχεδιασμού ενός πλαισίου δοκιμών. Η ορθή επιλογή μεθοδολογίας και μοτίβου εξαρτάται από την πολυπλοκότητα της εφαρμογής, το επίπεδο εμπειρίας της ομάδας δοκιμών, την απαίτηση για επαναχρησιμοποίηση και το επιθυμητό βάθος του ελέγχου. Οι προσεγγίσεις αυτές δεν αποκλείουν η μία την άλλη, αλλά μπορούν να συνδυαστούν, συνθέτοντας ένα υβριδικό πλαίσιο που αξιοποιεί τα πλεονεκτήματα περισσότερων από ενός μοντέλου.

# 3. Σχετικές εργασίες

Η ενότητα αυτή είναι αφιερωμένη σε προηγούμενες εργασίες που επικεντρώθηκαν πάνω στο αντικείμενο της διπλωματικής εργασίας μας. Θα αναφέρουμε και θα αναλύσουμε ορισμένες από αυτές, ειδικότερα όσον αφορά το σκοπό της κάθε μίας και τη συνεισφορά της, καθώς και θα τις συγκρίνουμε με την δική μας εργασία.

Τονίζουμε πως ο εντοπισμός, η συλλογή και μελέτη αυτών των σχετικών εργασιών μας επέτρεψε να σχηματίσουμε μία βιβλιοθήκη ιδεών – απόψεων που αφορούν στη δοκιμή λογισμικού και τα πλαίσια δοκιμών. Σε αυτή τη βάση χτίστηκε εν μέρει η δική μας συνεισφορά για το ζήτημα της αξιολόγησης, σύγκρισης και επιλογής του κατάλληλου πλαισίου δοκιμής.

## 3.1 A comparative study of automation testing tools for web applications

Η εργασία [26] των Pelivani & Cico (2021), μετά από μία σύντομη αναφορά στη σημασία της δοκιμής στον κύκλο ζωής και ανάπτυξης του λογισμικού, εξηγεί τα πλεονεκτήματα της μετάβασης από τη «χειροκίνητη» δοκιμή στην αυτοματοποιημένη.

Στην εργασία [26] γίνεται αναφορά σε δύο εργαλεία δοκιμών, το Katalon Studio[1] και το Selenium[2], τα οποία εστιάζουν κυρίως σε δοκιμές εφαρμογών ιστού, χωρίς όμως να περιορίζονται σε αυτές. Υποστηρίζουν δοκιμές εφαρμογών κινητών & ιστού, διεπαφής χρήστη (πχ. image-based δοκιμές για την επικύρωση της ορθότητας των γραφικών στοιχείων της διεπαφής χρήστη), συστήματος & αποδοχής, παλινδρόμησης (regression), απόδοσης και ασφάλειας, ενώ υποστηρίζουν εργαλεία συνεχούς ενσωμάτωσης (CI). Μετά την παρουσίαση των βασικών χαρακτηριστικών τους, τα δύο πλαίσια συγκρίνονται με βάση τα εξής κριτήρια: την ευκολία στην εγκατάσταση και χρήση, τις γλώσσες προγραμματισμού που το καθένα υποστηρίζει, το χρόνο που απαιτείται για τη δημιουργία σεναρίων δοκιμών, το κόστος χρήσης, τη δυνατότητα δημιουργίας και διαχείρισης αναφορών και τη ταχύτητα εκτέλεσης των δοκιμών.

[1] https://www.katalon.com
[2] https://www.selenium.dev

Τα αποτελέσματα της σύγκρισης έδειξαν ότι όσον αφορά την ευκολία χρήσης, το Katalon Studio είνια πιο φιλικό προς το χρήστη, έχοντας εύκολη εγκατάσταση και λειτουργίες, όπως καταγραφή και αναπαραγωγή ενεργειών, ενώ το Selenium απαιτεί κατάλληλες προγραμματιστικές γνώσεις για τη ρύθμισή του. Το Katalon Studio έχει ενσωματωμένα εργαλεία για γρήγορη δημιουργία δοκιμών, σε αντίθεση με το Selenium, το οποίο καθυστερεί σημαντικά σε σύγκριση με το πρώτο. Όμως, το Selenium υποστηρίζει περισσότερες γλώσσες προγραμματισμού από το Katalon Studio, όπως Java, Python, C#, Ruby και PHP, προσφέροντας, έτσι, μεγαλύτερη ευελιξία. Ενώ το Katalon Studio περιορίζεται στη χρήση Java. Επίσης, καθώς το Katalon Studio «φορτώνει» πρόσθετες βιβλιοθήκες κατά την εκτέλεσή του, γίνεται και πιο αργό. Όσον αφορά την παραγωγή αναφορών, το Katalon Studio διαθέτει ενσωματωμένα και φιλικά προς το χρήστη εργαλεία για τη δημιουργία αναφορών, ενώ το Selenium απαιτεί πρόσθετα, όπως για παράδειγμα το Cucumber, για τη δημιουργία αναφορών. Τέλος, το Selenium είναι ανοικτού κώδικά ενώ το Katalon Studio είναι κλειστού όπου παρέχεται μια έκδοσή του με περιορισμένες δυνατότητες χωρίς χρέωση.

Το συμπέρασμα της εργασίας υπό ανάλυση είναι ότι η επιλογή ανάμεσα στα Katalon Studio και Selenium κρίνεται από την εμπειρία του χρήστη και συγκεκριμένα το πρώτο ενδείκνυται για τον αρχάριο χρήστη που αναζητά την ευκολία σε εγκατάσταση και ταχύτητα στη δημιουργία δοκιμών, ενώ το δεύτερο αφορά κυρίως σε προχωρημένους και έμπειρους χρήστες που ψάχνουν την ευελιξία και την εξατομίκευση.

## 3.2 A Study of Automated Software Testing: Automation Tools and Frameworks

Οι Mubarak Albarka Umar και Chen Zhanfang (2019) δημοσίευσαν μελέτη, μέσω της οποίας παρουσιάζονται εργαλεία και πλαίσια αυτοματοποίησης δοκιμών λογισμικού. Η μελέτη επιδιώκει να εξηγήσει τις διάφορες κατηγορίες αυτοματοποιημένων δοκιμών, να περιγράψει τα είδη προσεγγίσεων αυτοματοποίησης που εφαρμόζονται στα πλαίσια δοκιμών (όπως αυτά αναλύθηκαν και στην Ενότητα 2.5), και να παρέχει μια σύγκριση δημοφιλών εργαλείων αυτοματοποίησης. Σκοπός της μελέτης είναι να διευκολύνει την επιλογή κατάλληλων μεθόδων και εργαλείων για

την επιτυχή και αποτελεσματική αυτοματοποίηση των δοκιμών λογισμικού. Η ταξινόμηση των πλαισίων που παρουσιάζουν οι Umar και Zhanfang εστιάζει σε μεθοδολογικές προσεγγίσεις, όπως: Linear Automation Framework (Γραμμική Αυτοματοποίηση), Data-Driven Framework (Οδηγούμενο από Δεδομένα), Keyword-Driven Framework (Οδηγούμενο από Λέξεις-Κλειδιά), όπως επίσης και στα ακόλουθα αρχιτεκτονικά μοτίβα Σχεδίασης: Modular Framework (Αρθρωτό Πλαίσιο), Library Architecture Framework (Αρχιτεκτονική Βιβλιοθηκών). Όπως αναφέρεται και στην Ενότητα 2.5, οι παραπάνω προσεγγίσεις δεν είναι αλληλοαποκλειόμενες, αλλά μπορούν να συνδυαστούν δημιουργώντας υβριδικά πλαίσια, ανάλογα με τις ανάγκες της ομάδας και τις δυνατότητες των εργαλείων.

Στη συνέχεια περιγράφονται τέσσερα προγράμματα αυτοματοποίησης, το Selenium, το Unified Functional Testing (UFT), το Katalon Studio και το TestComplete. Στον παρακάτω πίνακα συγκρίνονται τα πλαίσια αυτά με βάση συγκεκριμένα κριτήρια (πάντα στο πλαίσιο της εργασίας υπό μελέτη).

*Πίνακας 1: Σύγκριση πλαισίων δοκιμών [25]*

| **Χαρακτηριστικά** | **Katalon Studio** | **Selenium** | **UFT** | **TestComplete** |
|---|---|---|---|---|
| *Πλατφόρμα/ΛΣ ανάπτυξης* | Πολλαπλές | Πολλαπλές | Μόνο Windows | Πολλαπλές |
| *Εφαρμογές που δοκιμάζονται* | Web, mobile και API | Μόνο web | Web, mobile, API και desktop | Web, mobile, API και windows |
| *Γλώσσες προγραμματισμού* | Java/Groovy | Java, C#, Perl, Python, JavaScript, | VBScript | JavaScript, Python, VBScript, JScript, |

| | | | | |
|---|---|---|---|---|
| | | Ruby και PHP | | Delphi, C++ και C# |
| *Κόστος* | Δωρεάν / Επιπλέον δυνατότητες επί πληρωμή | Δωρεάν | Απαιτεί πληρωμή | Απαιτεί πληρωμή |
| *Μεθοδολογίες & Αρχιτεκτονικά Μοτίβα Πλαισίου* | Υβριδική (Data-Driven, Keyword-Driven, BDD) | Κυρίως Data-Driven & Keyword-Driven | Keword-Driven & Modular προσεγγίσεις | Έμφαση σε Keyword-Driven μοντέλο |

Τελικά, η μελέτη καταλήγει ότι δεν υπάρχουν καλά και κακά πλαίσια δοκιμών και ότι η επιλογή του κατάλληλου εξαρτάται σε μεγάλο βαθμό από το σύστημα που θα δοκιμαστεί, αλλά και από το επιθυμητό βάθος της δοκιμής, τις ικανότητες του οργανισμού ή της ομάδας δοκιμών και το συνολικό κόστος. Όπως είδαμε και στην προηγούμενη σχετική εργασία (Ενότητα 3.1), έτσι και τώρα το Selenium παρουσιάζεται ως ένα εξαιρετικό εργαλείο για λεπτομερείς δοκιμές εφαρμογών ιστού, επιτρέποντας υψηλό βαθμό αυτοματοποίησης και προσαρμοστικότητας, το οποίο όμως απαιτεί προχωρημένες γνώσεις προγραμματισμού και αυξημένη εμπειρία. Αντίθετα, το Katalon Studio είναι εκείνο με τη μεγαλύτερη ευκολία στη χρήση, όμως το βάθος των δοκιμών είναι περιορισμένο σε σύγκριση με το Selenium.

Τα UFT και TestComplete, ναι μεν παρουσιάζονται ως κατάλληλα για χρήση σε πολύπλοκα σενάρια δοκιμών εφαρμογών desktop και API, όμως το εκάστοτε κόστος (και για τα δύο) και η περιορισμένη υποστήριξη σε λειτουργικά συστήματα του πρώτου (μόνο Windows), τα περιορίζει και δεν τα καθιστά ιδανικά για όλες τις περιπτώσεις. Ειδικά για τις ικανότητες του οργανισμού/χρήστη και τη σχετική εμπειρία, θα λέγαμε ότι μία ομάδα με υψηλή εξειδίκευση στις γλώσσες που υποστηρίζει το Selenium, θα

μπορέσει να αξιοποιήσει πλήρως τις δυνατότητές του, σε αντίθεση με μια ομάδα της οποίας η εμπειρία προγραμματισμού είναι περιορισμένη και είναι πολύ πιθανό αυτή να θελήσει να αποφύγει το Selenium και να αναζητήσει τη λύση στις δυνατότητες του keyword-driven testing του TestComplete ή την ευκολία χρήσης του Katalon Studio.

Από την άποψη της αρχιτεκτονικής πλαισίου και των μεθοδολογιών που υποστηρίζουν, τα εργαλεία διαφοροποιούνται ως εξής:

- Ως προς τις μεθοδολογίες πλαισίου (Framework Methodologies), το Selenium υποστηρίζει κυρίως τις προσεγγίσεις data-driven και keyword-driven. Το Katalon Studio προσφέρει την πιο ευρεία κάλυψη, με υβριδική υποστήριξη για data-driven, keyword-driven και BDD (Behavior-Driven Development) frameworks. Τόσο το TestComplete όσο και το UFT δίνουν έμφαση στο keyword-driven μοντέλο, με το TestComplete να διευκολύνει ιδιαίτερα την εργασία μη προγραμματιστών.
- Ως προς τα αρχιτεκτονικά μοτίβα σχεδίασης (Design Patterns), το UFT είναι το εργαλείο που ενσωματώνει επιπλέον modular (αρθρωτές) προσεγγίσεις στην αρχιτεκτονική του πλαισίου.

## 3.3 Comparison Analysis of Android GUI Testing Frameworks by Using an Experimental Study

Το συγκεκριμένο άρθρο δημοσιεύθηκε το 2018 [17] και αποτέλεσε μία σύγκριση μεταξύ τεσσάρων πλαισίων δοκιμών GUI για το Android: Espresso, Appium, Calabash και UI Automator, μέσω μιας πειραματικής μελέτης. Τα κριτήρια σύγκρισης ήταν τόσο γενικά, όπως η απλότητα και η ευχρηστία, η ποιότητα του συστήματος καταγραφής και η δυνατότητα αυτοματοποίησης στη δημιουργία σεναρίων για εύρεση σφαλμάτων, όσο και τεχνικά, όπως η υποστήριξη πολλαπλών εκδόσεων Android, η δυνατότητα συνέχισης μίας εφαρμογής που έχει τεθεί σε αναστολή λειτουργίας, η δυνατότητα αλλαγής του προσανατολισμού της οθόνης κ.α..

Ο παρακάτω πίνακας παρουσιάζει το αποτέλεσμα της αξιολόγησης των πλαισίων αυτών για τα κριτήρια που επιλέχθηκαν. Αξίζει να διευκρινιστεί ότι τα κριτήρια αξιολόγησης αποτυπώνονται με διαφορετικά είδη τιμών. Για ορισμένα

κριτήρια (όπως η δυνατότητα συγχρονισμού ή η υποστήριξη συγκεκριμένων δοκιμών) χρησιμοποιούνται λογικές τιμές (ναι ή όχι), οι οποίες δηλώνουν καθαρά την ύπαρξη ή απουσία της δυνατότητας. Σε άλλα κριτήρια όμως (όπως η ανθεκτικότητα ή η υποστήριξη CI/CD, ο χειρισμός σφαλμάτων) χρησιμοποιούνται ποιοτικές περιγραφές που αποτυπώνουν είτε τον βαθμό/επίπεδο υποστήριξης, είτε την απόδοση του ίδιου του εργαλείου Οι ποιοτικές αυτές τιμές είναι: "αυτόματη", "εκτενής", "μέτρια", "περιορισμένη", "μερικός" και "γρήγορη". Ειδικότερα, ο χαρακτηρισμός "αυτόματη" υποδηλώνει την πλήρως ενσωματωμένη και χωρίς παρεμβάσεις υποστήριξη. Οι χαρακτηρισμοί "εκτενής", "μέτρια" και "περιορισμένη" αποτυπώνουν τον αντίστοιχο βαθμό κάλυψης ή δυνατοτήτων (π.χ., Εκτενής = υψηλός βαθμός, Μέτρια = ικανοποιητικός, Περιορισμένη = υποστήριξη με σημαντικούς περιορισμούς). Ο χαρακτηρισμός "μερικός" (όπως στον Χειρισμό Σφαλμάτων) σημαίνει ότι η δυνατότητα υπάρχει, αλλά καλύπτει μόνο ένα μέρος των απαιτούμενων λειτουργιών ή είναι ελλιπής. Τέλος, ο χαρακτηρισμός "γρήγορη" χρησιμοποιείται για την αξιολόγηση της υψηλής απόδοσης ή ταχύτητας εκτέλεσης του πλαισίου. Αντίθετα, η λογική τιμή "Ναι" απλώς δηλώνει την ύπαρξη της δυνατότητας, όχι όμως απαραίτητα τον τρόπο, την ποιότητα ή την ταχύτητα με την οποία υλοποιείται.

*Πίνακας 2: Συνοπτική παρουσίαση της αξιολόγησης πλαισίων αυτοματοποίησης δοκιμών σε γενικά και τεχνικά κριτήρια [17]*

| Κριτήρια | Espresso | UI Automator | Appium | Calabash |
|---|---|---|---|---|
| Συνέχιση εφαρμογής (Resume application) ( Τεχνικό) | Όχι | Ναι | Όχι | Όχι |
| Αναστολή εφαρμογής (Suspend application) (Τεχνικό) | Όχι | Ναι | Ναι | Ναι |

| | | | | |
|---|---|---|---|---|
| Ολοκλήρωση δραστηριότητας (Finish activity)<br>(Τεχνικό) | Ναι | Ναι | Ναι | Ναι |
| Τροποποίηση ρυθμίσεων συσκευής (Change device settings)<br>(Τεχνικό) | Όχι | Όχι | Ναι | Όχι |
| Κύλιση (Scrolling)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |
| Ταυτόχρονες πληκτρολογήσεις (Simultaneous keypresses)<br>(Τεχνικό) | Όχι | Όχι | Ναι | Όχι |
| Υποστήριξη καθυστέρησης (Delay support)<br>(Τεχνικό) | Αυτόματη | Ναι | Ναι | Ναι |
| Δοκιμή βάσει συνθηκών (Condition-based testing)<br>(Τεχνικό) | Ναι | Ναι | Ναι | Ναι |
| Εισαγωγή κειμένου (Type text field) | Ναι | Ναι | Ναι | Ναι |

| | | | | |
|---|---|---|---|---|
| (Γενικό) | | | | |
| Πάτημα κουμπιού (Click button)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |
| Πάτημα ραδιο-κουμπιού (Click radio button)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |
| Πάτημα κουμπιού επιλογής (Click checkbox)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |
| Επιλογή συλλέκτη (Select picker)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |
| Επιλογή στροβιλιστή (Select spinner)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |
| Σύρσιμο (Swiping)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |
| Επιλογή τίτλου tab (Click tab title)<br>(Γενικό) | Ναι | Ναι | Ναι | Ναι |

| | | | | |
|---|---|---|---|---|
| Αλλαγή προσανατολισμού (Orientation change) (Γενικό) | Όχι | Ναι | Ναι | Ναι |
| Χειρισμός σφαλμάτων (Error handling) (Τεχνικό) | Μερικός | Μερικός | Ναι | Μερικός |
| Ταχύτητα εκτέλεσης (Execution speed) (Τεχνικό) | Γρήγορη | Μέτρια | Μέτρια | Αργή |
| Υποστήριξη εκδόσεων Android (Android versions supported) (Τεχνικό) | Περιορισμένη | Εκτενής | Εκτενής | Περιορισμένη |

Το άρθρο δεν καταλήγει ξεκάθαρα σε κάποιο πλαίσιο «νικητή».Ο συνδυασμός γενικών και τεχνικών κριτηρίων αναδεικνύει ορισμένα πλεονεκτήματα και μειονεκτήματα για κάθε πλαίσιο. Τα πιο σημαντικά σημεία που προκύπτουν από τον πίνακα είναι τα εξής:

- **Espresso:** Υπερέχει σε ταχύτητα εκτέλεσης και υποστηρίζει τις περισσότερες βασικές λειτουργίες (π.χ. τύπος δεδομένων, κλικ κουμπιών), αλλά υστερεί σε προηγμένες δυνατότητες (αναστολή εφαρμογής, αλλαγή ρυθμίσεων συσκευής). Επίσης, δεν υποστηρίζει πολλές εκδόσεις Android καθώς και παρέχει περιορισμένη υποστήριξη στον χειρισμό σφαλμάτων. Λόγω της απλότητας της χρήσης του, το Espresso είναι ιδιαίτερα κατάλληλο για αρχάριους χρήστες.
- **UI Automator:** Ξεχωρίζει ως προς υποστήριξη περισσότερων εκδόσεων Android, όλων των βασικών λειτουργιών καθώς και των περισσοτέρων

προηγμένων λειτουργιών/δυνατοτήτων (πχ. αναστολή εφαρμογής, αλλαγή προσανατολισμού), όμως είναι πιο αργό από το Espresso και η διαχείριση σφαλμάτων που διαθέτει είναι περιορισμένη.

- **Appium:** Φαίνεται να είναι το πιο ισχυρό πλαίσιο, καθώς υποστηρίζει όλες σχεδόν τις λειτουργίες (εκτός από την αναστολή εφαρμογής), τόσο στα γενικά όσο και στα τεχνικά κριτήρια. Παρότι δεν είναι τόσο γρήγορο όσο το Espresso, η ευελιξία και η πληρότητά του το καθιστούν, ίσως, την κορυφαία επιλογή.
- **Calabash:** Καλύπτει όλες τις βασικές λειτουργίες, όμως υστερεί σημαντικά σε τομείς, όπως η ταχύτητα εκτέλεσης, η υποστήριξη εκδόσεων Android και ο χειρισμός λαθών. Για τους λόγους αυτούς, δε θα λέγαμε ότι είναι τόσο ανταγωνιστικό έναντι του UI Automator και του Appium. Αλλά βρίσκεται σε παρόμοιο επίπεδο με αυτό του Espresso (υποστηρίζει ναι μεν περισσότερες λειτουργίες από το Espresso αλλά είναι πιο αργό).

Τελικά, το Espresso προτείνεται ως κατάλληλο για αρχάριους επειδή είναι πιο απλό στη χρήση. Αναλύοντας τον πίνακα με τα κριτήρια, εξάγεται ότι το Appium καλύπτει τις περισσότερες δυνατότητες σε σύγκριση με τα υπόλοιπα πλαίσια, γεγονός που το καθιστά ισχυρό υποψήφιο για την πιο ολοκληρωμένη λύση. Δυστυχώς, όμως, και τα τέσσερα πλαίσια παρουσιάζουν ελλείψεις στην αυτόματη δημιουργία γεγονότων, πράγμα το οποίο δεν φαίνεται από τον παραπάνω πίνακα.

## 3.4 Taxonomy of Automated Software Testing Tools

Στόχος της έρευνας [23] ήταν η ταξινόμηση και η σύγκριση πλαισίων δοκιμής που διατίθενται στην αγορά, μέσω των οποίων εξασφαλίζεται η ποιότητα των προϊόντων λογισμικού.

Αρχικά, γίνεται αναφορά στις προκλήσεις που μπορεί να αντιμετωπίσει μία ομάδα προγραμματιστών ή ένας οργανισμός κατά τη χρήση των εργαλείων αυτών. Ακολουθεί η περιγραφή βασικών χαρακτηριστικών των πλαισίων και η σύγκριση και ταξινόμηση αυτών με βάση τα εξής κριτήρια: η υποστήριξη φυλλομετρητών, η άδεια χρήσης, το κόστος, οι υποστηριζόμενες πλατφόρμες, οι υποστηριζόμενες γλώσσες προγραμματισμού και το λειτουργικό σύστημα. Συνολικά αναφέρονται στην έρευνα 32 εργαλεία αυτοματοποίησης δοκιμών, τα οποία, τελικά, συγκρίνονται με βάση τα 6 ανωτέρω χαρακτηριστικά, σε μορφή πίνακα.

Η έρευνα δεν καταλήγει σε κάποιο ή κάποια πλαίσια που ξεκάθαρα υπερέχουν έναντι άλλων (ως προς όλα τα κριτήρια). Όμως, εάν εστιάσουμε σε λιγότερα κριτήρια (5), όπως για παράδειγμα η άδεια και το κόστος χρήσης, η υποστήριξη σε φυλλομετρητές, οι γλώσσες προγραμματισμού και λειτουργικά συστήματα, παρατηρούμε ότι το Selenium «δείχνει» να υπερτερεί έναντι των υπολοίπων.

## 3.5 Web App Security - A Comparison and Categorization of Testing Frameworks

Σύμφωνα με το άρθρο αυτό [24], η ευπάθεια των διαδικτυακών εφαρμογών ολοένα και αυξάνεται και έτσι οι χρήστες των εφαρμογών θέτονται σε κίνδυνο. Για την αντιμετώπιση του προβλήματος αυτού, προτείνεται η χρήση πλαισίων δοκιμών ασφαλείας.

Το άρθρο επικεντρώνεται στην κατηγοριοποίηση και σύγκριση τέτοιων πλαισίων, καθώς οι προγραμματιστές συχνά αντιμετωπίζουν δυσκολίες στην επιλογή και στη χρήση των διαθέσιμων πλαισίων, λόγω της αυξημένης πολυπλοκότητάς τους. Συνολικά, ταξινομούνται 26 πλαίσια δοκιμών ασφαλείας με βάση 27 κριτήρια, από τα οποία, τα πιο βασικά είναι τα εξής: οι τύποι των επιθέσεων που καλύπτονται από το πλαίσιο, ο βαθμός της αυτοματοποίησής του, η γλώσσα προγραμματισμού, εάν είναι ανοικτού κώδικα ή όχι και το είδος των εφαρμογών που μπορεί το πλαίσιο να δοκιμάσει (Web, mobile κ.α.).

Το άρθρο καταλήγει στα 10 καλύτερα πλαίσια δοκιμών ασφαλείας: OWASP Enterprise Security API, Python - Django – based framework, w3af, Hdiv, Spring Security, Samurai Web Testing Framework, SPIKE Proxy, Yii, Jboss και YSO Mobile Security Framework. Όμως εδώ πρέπει να σημειωθεί ότι ορισμένα από τα ανωτέρω εργαλεία, όπως το Django-based framework, το Yii και το JBoss, παρέχουν κυρίως μηχανισμούς ασφαλούς διαμόρφωσης και προστασίας, χωρίς να αποτελούν καθεαυτού εργαλεία ελέγχου ευπαθειών ή αυτοματοποιημένων δοκιμών ασφάλειας.

## 3.6 Η δική μας συνεισφορά

Η παρούσα εργασία διαφοροποιείται ουσιαστικά από τις σχετικές μελέτες που παρουσιάστηκαν παραπάνω, τόσο ως προς την πολυπλοκότητα της ανάλυσης, όσο

και ως προς τη συστηματική και δομημένη προσέγγιση της σύγκρισης των πλαισίων δοκιμών εφαρμογών ιστού. Σε αντίθεση με τις περισσότερες εργασίες που εξετάζουν 2 έως 4 εργαλεία είτε εστιάζουν μόνο λίγα κριτήρια σύγκρισης, η δική μας μελέτη θα αξιολογήσει 9 διαφορετικά πλαίσια χρησιμοποιώντας 15 στοχευμένα κριτήρια, τα οποία θα αφορούν τόσο σε τεχνικές δυνατότητες (όπως η υποστήριξη παράλληλων δοκιμών, δυνατότητα αναφορών, CI/CD ενσωματώσεις κ.α.), όσο και πρακτικές παραμέτρους (όπως το κόστος, ευκολία χρήσης και βαθμός επεκτασιμότητας).

Επιπλέον, στην παρούσα εργασία, πέρα από την ποσοτική ανάλυση, θα ενσωματωθούν ποιοτικά σχόλια και, κυρίως, προτάσεις χρήσης ανά περίπτωση, βασισμένα στις ανάγκες και τα χαρακτηριστικά διαφορετικών ομάδων ανάπτυξης λογισμικού. Σε αντίθεση με τις προγενέστερες εργασίες που εστιάζουν είτε στη λειτουργικότητα είτε σε επιμέρους χαρακτηριστικά (όπως απόδοση ή ασφάλεια), εδώ επιχειρείται μια σφαιρική αξιολόγηση, η οποία ενσωματώνει και τις αρχιτεκτονικές διαστάσεις των πλαισίων (π.χ. επίπεδα υποστήριξης, εκτέλεσης, αναφορών κ.λπ.), που σε καμία από τις αναφερόμενες μελέτες δεν αποτυπώνονται ολοκληρωμένα.

Τέλος, η μελέτη θα περιλαμβάνει οδηγό επιλογής πλαισίου (διάγραμμα ροής), ο οποίος συνθέτει τα αποτελέσματα της ανάλυσης σε μορφή χρηστικού εργαλείου για ομάδες ανάπτυξης. Έτσι, δεν περιορίζεται σε μια θεωρητική αξιολόγηση, αλλά παρέχει και ένα βοήθημα απόφασης, κάτι που απουσιάζει από τις προηγούμενες εργασίες.

# 4. Ανάλυση Πλαισίων Δοκιμής Εφαρμογών Ιστού

Στο κεφάλαιο αυτό θα παρουσιάσουμε εννιά πλαίσια δοκιμών: το JUnit, το Selenium, το TestNG, το PyTest, το Cypress, το Playwright, το Appium, το Cucumber και το TestCafe. Η επιλογή αυτών των πλαισίων βασίστηκε τόσο στη δημοτικότητά τους, όσο και στην ποικιλία λειτουργιών που προσφέρουν, καλύπτοντας ανάγκες όπως αυτοματοποίηση, υποστήριξη πολλαπλών γλωσσών προγραμματισμού, δυνατότητα ενσωμάτωσης με CI/CD εργαλεία/πλαίσια κ.α.. Η αναζήτηση και ο εντοπισμός των σχετικών πηγών πραγματοποιήθηκε χρησιμοποιώντας τις μηχανές αναζήτησης Google και Google Scholar. Συγκεκριμένα, χρησιμοποιήθηκαν λέξεις-κλειδιά, όπως "web testing frameworks", "comparison of test automation tools" και "popular software testing frameworks". Μέσω αυτής της αναζήτησης, εντοπίστηκαν και αντλήθηκαν επιστημονικά άρθρα (μέσω Google Scholar) που αντιστοιχούν στις σχετικές εργασίες που αναλύθηκαν στο προηγούμενο κεφάλαιο, γκρίζα βιβλιογραφία (διατριβές, κεφάλαια βιβλίων, τεχνικές αναφορές), τεχνικές αναφορές από τις επίσημες ιστοσελίδες και τα αποθετήρια (repositories) των ίδιων των εργαλείων, ιστοσελίδες σύγκρισης εργαλείων και κοινότητες προγραμματιστών (όπως το Stack Overflow). Από όλες αυτές τις πηγές αντλήθηκε σχετικό περιεχόμενο, όπως τεκμηρίωση, σκορ (π.χ., στο GitHub), και μετρικές δημοτικότητας. Αρχικά εντοπίστηκαν περισσότερα πλαίσια από αυτά που παρουσιάζονται, αλλά έγινε τελική επιλογή με βάση το πόσο διαδεδομένα είναι και το αν προσφέρουν διαφορετικές δυνατότητες, ώστε να υπάρχει ποικιλία στη σύγκριση. Θα αναφέρουμε στη συνέχεια τόσο τα πλεονεκτήματα και τα μειονεκτήματά τους, καθώς και ό,τι τα κάνει ξεχωριστά.

Είναι κρίσιμο να γνωρίζουμε αν το εργαλείο είναι δωρεάν ή αν απαιτεί άδειες χρήσης ή συνδρομές. Η ευκολία στη χρήση αποτελεί επίσης ένα βασικό κριτήριο, αφού η καμπύλη εκμάθησης και η καθημερινή εμπειρία χρήσης παίζουν καθοριστικό ρόλο στην υιοθέτηση και αξιοποίηση του πλαισίου. Οι περισσότερες ομάδες ανάπτυξης επιθυμούν το πλαίσιο δοκιμής να υποστηρίζει την εκτέλεση πολλαπλών δοκιμών ταυτόχρονα, ώστε να μειώνεται ο συνολικός χρόνος δοκιμής και κατ’ επέκταση της ανάπτυξης. Η συχνότητα ενημέρωσης είναι σημαντική για τη διατήρηση της ασφάλειας, της σταθερότητας και της συμβατότητας του πλαισίου με νέες τεχνολογίες. Η συμβατότητα με διαφορετικά λειτουργικά συστήματα εξασφαλίζει την ευελιξία

χρήσης του πλαισίου σε ετερογενή περιβάλλοντα ανάπτυξης. Αντίστοιχα, η ικανότητα δοκιμής εφαρμογών ιστού είναι απαραίτητη για ομάδες που αναπτύσσουν τέτοιες εφαρμογές, ενώ η υποστήριξη πολλών φυλλομετρητών διευκολύνει τις δοκιμές συμβατότητας και τη διασφάλιση της εμπειρίας χρήστη σε διαφορετικές πλατφόρμες.

Η δυνατότητα παραγωγής αναφορών επιτρέπει την τεκμηρίωση των αποτελεσμάτων των δοκιμών και την αξιολόγηση της ποιότητας του λογισμικού, ενώ οι υποστηριζόμενες ενοποιήσεις με άλλα εργαλεία (όπως CI/CD, bug tracking κ.α.) συμβάλλουν στην αυτοματοποίηση και επιτάχυνση των διαδικασιών ελέγχου ποιότητας. Τέλος, η ύπαρξη γραφικής διεπαφής (GUI) ή γραμμής εντολών (CLI) επηρεάζει την αλληλεπίδραση του χρήστη με το εργαλείο, ανάλογα με τις προτιμήσεις ή τις ανάγκες του, ενώ η δυνατότητα αυτόματης παραγωγής δεδομένων δοκιμής, καθώς και η δυνατότητα αυτόματης παραγωγής δοκιμών προσφέρουν μεγαλύτερη ευκολία και ταχύτητα στη δημιουργία και εκτέλεση σεναρίων ελέγχου.

Παίρνοντας υπόψιν τα παραπάνω, τα κριτήρια με βάση τα οποία θα γίνει η σύγκριση είναι τα εξής: Κόστος χρήσης, ευκολία στη χρήση, υποστήριξη CI/CD, υποστήριξη mocking/stubbing, υποστήριξη bug tracking, υποστήριξη παράλληλων δοκιμών, συχνότητα ενημέρωσης, συμβατότητα με λειτουργικά συστήματα, ικανότητα δοκιμής εφαρμογών ιστού, συμβατότητα με φυλλομετρητές, δυνατότητα παραγωγής αναφορών, ύπαρξη διεπαφής χρήσης GUI ή/και CLI, δυνατότητα αυτόματης παραγωγής δεδομένων δοκιμής και δυνατότητα αυτόματης παραγωγής δοκιμών.

## 4.1 Ανάλυση Κριτηρίων και Μεθοδολογία Βαθμολόγησης

Για τη σύγκριση των πλαισίων δοκιμής εφαρμογών ιστού, καθορίστηκαν είκοσι (20) κριτήρια αξιολόγησης, τα οποία καλύπτουν τεχνικές, λειτουργικές και πρακτικές πτυχές της χρήσης των πλαισίων σε πραγματικά περιβάλλοντα ανάπτυξης. Παρακάτω περιγράφονται οι διαφορετικές τιμές που μπορεί να λάβει κάθε πλαίσιο για καθένα από τα κριτήρια αξιολόγησης και τι ακριβώς αυτές σημαίνουν. Για τη συστηματική σύγκριση των πλαισίων δοκιμής εφαρμογών ιστού, τα είκοσι (20) επιμέρους κριτήρια αξιολόγησης ομαδοποιούνται σε επτά (7) κύριες Διαστάσεις Κατηγοριοποίησης. Αυτή η ιεραρχική δομή διευκολύνει την ανάλυση και επιτρέπει την κατηγοριοποίηση των πλαισίων με βάση τα βασικά χαρακτηριστικά τους. Κάθε διάσταση αποτελεί έναν άξονα βάσει του οποίου αξιολογούνται τα πλαίσια.

## 1. Τεχνική Εμβέλεια και Είδη Δοκιμών

Αυτή η διάσταση αξιολογεί το εύρος των δοκιμών που μπορεί να εκτελέσει το πλαίσιο και την τεχνολογική συμβατότητά του.

### Είδη δοκιμών που υποστηρίζονται

Αφορά τα είδη δοκιμών που μπορεί να υποστηρίξει κάθε πλαίσιο. Εξετάστηκαν οι εξής βασικές κατηγορίες: δοκιμές μονάδας (unit testing), δοκιμές ενσωμάτωσης (integration testing), δοκιμές συστήματος (system testing), δοκιμές διεπαφής χρήστη (UI testing), δοκιμές απόδοσης (performance testing), δοκιμές αποδοχής (acceptance testing), δοκιμές από άκρο σε άκρο (end-to-end), και δοκιμές API.

Για κάθε πλαίσιο, αξιολογήθηκε αν η υποστήριξη για το εκάστοτε είδος δοκιμής παρέχεται εγγενώς, δηλαδή ως μέρος των βασικών δυνατοτήτων του πλαισίου ή αν απαιτείται ενσωμάτωση με τρίτα εργαλεία ή βιβλιοθήκες. Στις περιπτώσεις όπου η υποστήριξη προσφέρεται μόνο μέσω πρόσθετων ή εξωτερικών εργαλείων, αυτό σημειώνεται ως «εν μέρει», ενώ όπου δεν υπάρχει καμία σχετική υποστήριξη σημειώνεται ως «όχι». Να σημειωθεί ότι δεν πρόκειται τόσο για κάποιο κριτήριο αξιολόγησης, αλλά για ένα τρόπο κατηγοριοποίησης των πλαισίων που κρίνεται χρήσιμο να αναφέρεται και να συνοδεύει τα 15 κατωτέρω κριτήρια αξιολόγησης.

### Ικανότητα δοκιμής εφαρμογών ιστού

Αξιολογείται αν το εργαλείο μπορεί να χρησιμοποιηθεί για τη δοκιμή εφαρμογών που λειτουργούν σε περιβάλλον ιστού. Η αξιολόγηση δεν περιορίζεται μόνο σε μεμονωμένο έλεγχο διεπαφής χρήστη (UI) ή API, αλλά αφορά κυρίως την υποστήριξη ολοκληρωμένων end-to-end σεναρίων, στα οποία ελέγχεται συνολικά η λειτουργικότητα της εφαρμογής σε συνθήκες που προσομοιώνουν την πραγματική χρήση από τον τελικό χρήστη. Η τιμή «Ναι» αποδίδεται όταν το εργαλείο υποστηρίζει τέτοιου είδους δοκιμές, ενώ σε αντίθετη περίπτωση η τιμή «Όχι».

### Συμβατότητα με φυλλομετρητές

Εξετάζεται η δυνατότητα εκτέλεσης δοκιμών στους φυλλομετρητές Chrome, Firefox, Edge και Safari. Οι δυνατές τιμές είναι «Ναι» όταν υποστηρίζονται και οι 4 ανωτέρω φυλλομετρητές, «εν μέρει» όταν υποστηρίζονται 1-3 από αυτούς ή «Όχι» όταν δεν υπάρχει κάποια υποστήριξη.

### Υποστήριξη παράλληλων δοκιμών

Εξετάζεται αν το εργαλείο υποστηρίζει την ταυτόχρονη εκτέλεση πολλών δοκιμών (multithreading, distributed testing). Η τιμή «Ναι» αποδίδεται όταν η παράλληλη

εκτέλεση υποστηρίζεται άμεσα ή με επίσημες προσθήκες που παρέχονται από τους δημιουργούς του εργαλείου. Η τιμή «Εν μέρει» χρησιμοποιείται όταν η υποστήριξη είναι περιορισμένη ή ισχύει μόνο σε συγκεκριμένες περιπτώσεις. Τέλος, η τιμή «Όχι» δίνεται όταν η δυνατότητα αυτή δεν υποστηρίζεται καθόλου.

**2. Μοντέλο Πρόσβασης και Υποστήριξη**

Αυτή η διάσταση εστιάζει στο κόστος, την ανοικτότητα του κώδικα, τη δημοφιλία και την υποστήριξη της κοινότητας/δημιουργών.

**Κόστος χρήσης και ανοικτότητα κώδικα**

Αξιολογείται το αν το πλαίσιο είναι πλήρως δωρεάν, αν απαιτεί συνδρομή ή αν η δωρεάν του έκδοση έχει περιορισμούς στη χρήση. Τα πλαίσια που είναι ελεύθερα και επεκτάσιμα χαρακτιρίζονται ως «open-source», εκείνα που είναι δωρεάν αλλά κλειστού κώδικα «freeware», όσα είναι δωρεάν με περιορισμούς αλλά κλειστού κώδικα «freemium» και όσο είναι επί πληρωμή και κλειστού κώδικα «proprietary».

**Συχνότητα ενημέρωσης**

Εξετάζεται πόσο συχνά ενημερώνεται κάθε πλαίσιο, δηλαδή αν συνεχίζει να υποστηρίζεται ενεργά από τους δημιουργούς του. Τα πλαίσια που λαμβάνουν τακτικές ενημερώσεις (με νέες εκδόσεις, διορθώσεις και βελτιώσεις) χαρακτηρίζονται ως «Συχνή». Όσα ενημερώνονται πιο αραιά ή έχουν μειωμένη δραστηριότητα τα τελευταία χρόνια λαμβάνουν την ένδειξη «Μέτρια». Τέλος, αν η ανάπτυξη έχει ουσιαστικά σταματήσει χαρακτηρίζονται ως «Σπάνια».

**Σκορ / Αστέρια στο GitHub**

Αξιολογείται η δημοφιλία του πλαισίου βάσει των "αστεριών" (stars) που έχει λάβει στο GitHub. Αυτή η τιμή αποτελεί ένδειξη της δραστηριότητας της κοινότητας, της ευρείας υιοθέτησης και του ενδιαφέροντος για το έργο. «Υψηλό» όσα έχουν πάνω από 15 χιλ., «Μέτριο» όσα έχουν πάνω από 5 χιλ. και «Χαμηλό» όσα έχουν έως 5 χιλ..

**Συχνότητα Αναφοράς / Υιοθέτησης**

Αξιολογείται η γενική αναφορά, η χρήση σε άρθρα, έρευνες και επαγγελματικά έργα. Υποδηλώνει την αποδοχή του πλαισίου ως βιομηχανικό πρότυπο. Μπορεί να πάρει τις τιμές «Υψηλή», «Μέτρια», «Χαμηλή».

**3. Μεθοδολογίες Σχεδίασης Δοκιμών**

Αξιολογεί τη δυνατότητα του πλαισίου να υποστηρίζει σύγχρονες μεθοδολογίες και δομικές τεχνικές για τη συγγραφή δοκιμών.

**Υποστήριξη Μεθοδολογιών Σχεδίασης Δοκιμών**

Εξετάζεται αν το πλαίσιο υποστηρίζει ή ενθαρρύνει συγκεκριμένες μεθοδολογίες σχεδίασης δοκιμών, όπως το Behavior-Driven Development (BDD) ή το Keyword-Driven Testing. Ανατίθεται η τιμή «Ναι» όταν υπάρχει πλήρης/εγγενής υποστήριξη, η τιμή «Μέσω πρόσθετων» όταν υπάρχει υποστήριξη από εξωτερικά εργαλεία και «Όχι» όταν δεν υπάρχει σχετική υποστήριξη.

**Υποστήριξη Μοτίβων Σχεδίασης Δοκιμών**

Εξετάζεται η ευκολία με την οποία μπορούν να εφαρμοστούν τυποποιημένα μοτίβα (patterns) σχεδίασης, όπως το Page Object Model (POM), για τη βελτίωση της επαναχρησιμοποίησης και της συντηρησιμότητας του κώδικα δοκιμής. Ανατίθεται η τιμή «Υψηλή» όταν ενθαρρύνει/περιλαμβάνει εργαλεία για POM, «Μέτρια» όταν είναι δυνατή η εφαρμογή, αλλά όχι εγγενής υποστήριξη και «Χαμηλή» όταν απαιτεί σημαντική χειροκίνητη προσπάθεια για την εφαρμογή μοτίβων.

**4. Υποστηριζόμενες Γλώσσες Προγραμματισμού**

Εξετάζει το φάσμα των γλωσσών με τις οποίες μπορεί να αναπτυχθεί ο κώδικας των δοκιμών.

**Υποστήριξη Γλωσσών Προγραμματισμού**

Καταγράφονται οι γλώσσες προγραμματισμού στις οποίες μπορεί να γραφεί ο κώδικας των δοκιμών.Αναφέρεται ρητά η/οι γλώσσα/ες: π.χ. JavaScript/TypeScript, Python, Java, C#, κ.λπ.

**5. Είδη Προσφερόμενων Διεπαφών**

Εξετάζει τις διαθέσιμες διεπαφές για την αλληλεπίδραση του χρήστη με το πλαίσιο.

**Διεπαφή GUI**

Αξιολογείται αν το εργαλείο προσφέρει πλήρες γραφικό περιβάλλον για τη συγγραφή, εκτέλεση και διαχείριση των δοκιμών. Η τιμή «Ναι» δίνεται όταν υπάρχει ολοκληρωμένο GUI μέσω του οποίου ο χρήστης μπορεί να δημιουργεί σενάρια δοκιμών, να βλέπει αποτελέσματα και να παραμετροποιεί τις εκτελέσεις δοκιμών. Η τιμή «Εν μέρει» δίνεται όταν υπάρχει κάποια γραφική διεπαφή, αλλά με περιορισμένες δυνατότητες, όπως για παράδειγμα, το πλαίσιο να επιτρέπει μόνο την επισκόπηση αποτελεσμάτων ή την παρακολούθηση της εκτέλεσης, ή όταν η συγγραφή και διαχείριση των δοκιμών γίνεται μέσω ολοκληρωμένης ενσωμάτωσης σε IDE. Η τιμή «Όχι» αποδίδεται όταν η χρήση του εργαλείου γίνεται αποκλειστικά μέσω γραμμής εντολών (CLI), χωρίς επίσημο γραφικό περιβάλλον.

**Διεπαφή CLI**

Αξιολογείται κατά πόσο το εργαλείο υποστηρίζει την εκτέλεση και διαχείριση των δοκιμών μέσω εντολών τερματικού (CLI). Η τιμή «Ναι» αποδίδεται όταν μέσω CLI παρέχεται πλήρης έλεγχος (π.χ. εκτέλεση δοκιμών, ρύθμιση παραμέτρων και προβολή αποτελεσμάτων). Η τιμή «Εν μέρει» δίνεται όταν υπάρχουν βασικές μόνο λειτουργίες CLI (π.χ. απλή εκκίνηση) και απουσιάζουν δυνατότητες διαχείρισης ή παραμετροποίησης. Η τιμή «Όχι» αποδίδεται όταν το εργαλείο δεν προσφέρει CLI ή απαιτεί αποκλειστικά γραφικό περιβάλλον.

**6. Είδη Αυτοματοποίησης**

Εστιάζει στις δυνατότητες του πλαισίου να παράγει αυτόματα κώδικα δοκιμής ή δεδομένα εισόδου.

**Αυτόματη παραγωγή δεδομένων δοκιμής**

Αξιολογείται αν το εκάστοτε πλαίσιο υποστηρίζει τη δημιουργία δεδομένων εισόδου για τις δοκιμές χωρίς να απαιτείται η παρέμβαση του χρήστη. Η τιμή «Ναι» θα δίνεται όταν το εργαλείο παρέχει εγγενώς κάποιον τέτοιο μηχανισμό για την αυτόματη παραγωγή δεδομένων, όπως γεννήτριες τυχαίων τιμών ή βιβλιοθήκες για τη δημιουργία ψευδο-δεδομένων. Η τιμή «Μέσω πρόσθετων» δίνεται όταν η λειτουργία υποστηρίζεται μόνο με τη χρήση εξωτερικών βιβλιοθηκών και plugins. Η τιμή «Όχι» αποδίδεται όταν η παραγωγή δεδομένων πρέπει να γίνει αποκλειστικά χειροκίνητα από τον ίδιο το χρήστη.

**Αυτόματη παραγωγή δοκιμών**

Εξετάζεται αν το πλαίσιο υποστηρίζει την αυτόματη δημιουργία σεναρίων δοκιμής, για παράδειγμα μέσω καταγραφής ενεργειών του χρήστη ή με άλλες τεχνικές. Η τιμή «Ναι» αποδίδεται όταν η δυνατότητα αυτή προσφέρεται εγγενώς. Η τιμή «Μέσω πρόσθετων» χρησιμοποιείται όταν η λειτουργία υποστηρίζεται μόνο με εξωτερικά εργαλεία ή plugins, ενώ η τιμή «Όχι» δίνεται όταν το εργαλείο δεν προσφέρει καμία σχετική δυνατότητα.

**7. Πρακτική Υλοποίηση και Ενοποίηση**

Αξιολογεί την πρακτική ευκολία υιοθέτησης και ενσωμάτωσης του πλαισίου στο περιβάλλον ανάπτυξης.

**Ευκολία στη χρήση**

Αξιολογείται το πόσο εύκολο είναι για τον χρήστη να εγκαταστήσει, να ρυθμίσει και να χρησιμοποιήσει το κάθε πλαίσιο. Τα πλαίσια που παρέχουν γραφικό περιβάλλον,

οδηγούς εγκατάστασης και γενικά δεν απαιτούν προηγούμενη εμπειρία στον προγραμματισμό χαρακτηρίζονται ως «Εύκολη». Όσα απαιτούν βασικές τεχνικές γνώσεις και λίγη εξοικείωση με τον κώδικα λαμβάνουν την ένδειξη «Μέτρια». Τέλος, όσα απαιτούν αυξημένη εμπειρία ή πολύ καλή γνώση προγραμματισμού χαρακτηρίζονται ως «Δύσκολη».

**Συμβατότητα με λειτουργικά συστήματα**

Αξιολογείται σε ποια λειτουργικά συστήματα μπορεί να εγκατασταθεί και να λειτουργήσει κανονικά το κάθε πλαίσιο. Όσα υποστηρίζουν χωρίς περιορισμούς τα τρία βασικά λειτουργικά συστήματα (Windows, macOS και Linux) λαμβάνουν την ένδειξη «Πλήρης». Αν η υποστήριξη περιορίζεται, τότε θα αναφέρεται ρητά. Για παράδειγμα, «Μόνο Windows» ή «Μόνο macOS».

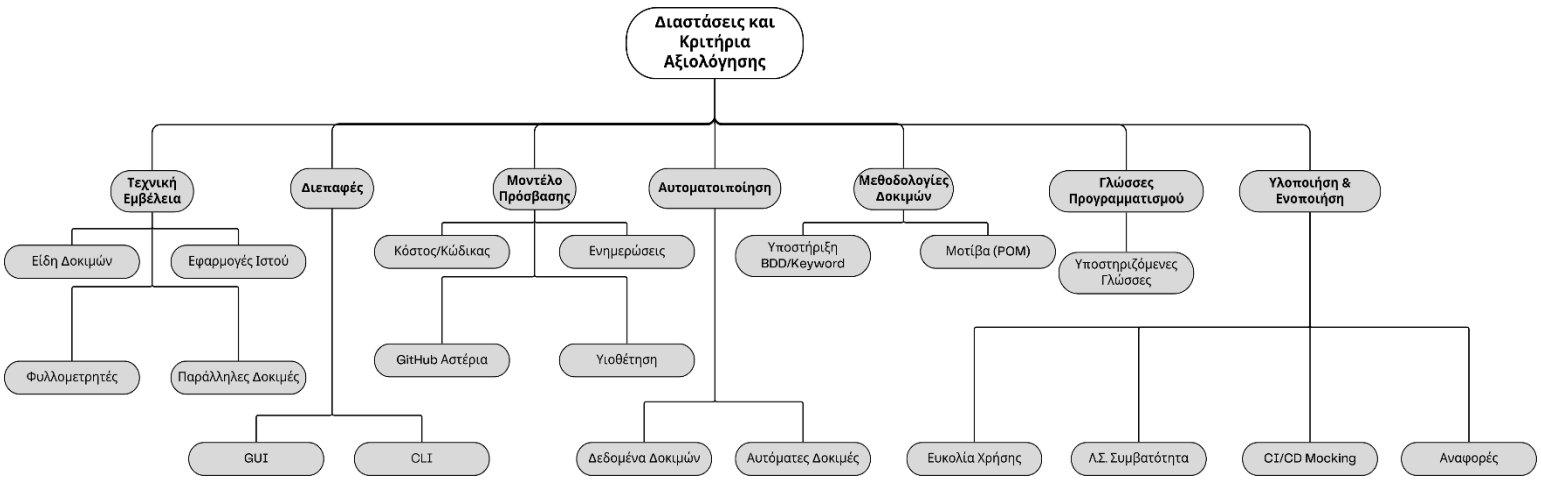
**Υποστήριξη CI/CD ,mocking/stubbing, bug tracking**

Εδώ εξετάζεται αν το κάθε πλαίσιο υποστηρίζει τις δυνατότητες αυτές εγγενώς, μέσω πρόσθετων ή καθόλου. Άρα ανατίθεται η τιμή «εγγενώς» για την πρώτη περίπτωση καθώς και οι τιμές «Μέσω πρόσθετων» και «Καθόλου» για τις υπόλοιπες δύο περιπτώσεις, αντίστοιχα.

**Δυνατότητα παραγωγής αναφορών**

Αξιολογείται κατά πόσο το εργαλείο μπορεί να παρέχει αναφορές με τα αποτελέσματα των δοκιμών. Η τιμή «Πλήρης» αποδίδεται όταν προσφέρονται έτοιμες, αναλυτικές και ευανάγνωστες αναφορές χωρίς την ανάγκη εξωτερικών παρεμβάσεων. Η τιμή «Μέσω πρόσθετων» δίνεται όταν η δυνατότητα παραγωγής αναφορών υλοποιείται με τη χρήση εξωτερικών εργαλείων με τα οποία συνεργάζεται το πλαίσιο.

*Εικόνα 2: Η ιεραρχική δομή των διαστάσεων και κριτηρίων αξιολόγησης των πλαισίων δοκιμής εφαρμογών ιστού.*



## 4.2 Το Junit

Το JUnit δημιουργήθηκε από τους Kent Beck και Erich Gamma στα τέλη της δεκαετίας του 1990 και έχει εξελιχθεί μέσα από πολλές εκδόσεις ώστε να υποστηρίζει τις σύγχρονες ανάγκες δοκιμών και ανάπτυξης λογισμικού. Ίσως είναι το πιο δημοφιλές πλαίσιο δοκιμών μονάδων για εφαρμογές που είναι γραμμένες σε Java [11]. Δεν έχει κόστος απόκτησης και χρήσης διότι είναι ανοικτού κώδικα. Είναι εύκολο στη χρήση από προγραμματιστές Java, καθώς όπως αναφέραμε είναι σχεδιασμένο για αυτή τη γλώσσα. Ωστόσο, μπορεί να φανεί περίπλοκο σε αρχάριους χρήστες σε σχέση με άλλα πλαίσια δοκιμών που θα δούμε στη συνέχεια. Όσον αφορά την επεκτασιμότητα, το Junit υποστηρίζει μεγάλες σουίτες δοκιμών και μπορεί να ενσωματωθεί σε άλλα εργαλεία, όπως το TestNG.

Η τελευταία έκδοση του JUnit είναι η «JUnit 5» που εισήγαγε πολλές βελτιώσεις σε σχέση με την παλαιότερη έκδοση «JUnit 4», ενώ η κοινότητα συνεχίζει να υποστηρίζει ενεργά το εργαλείο δίνοντας πολύτιμη ανατροφοδότηση και προτείνοντας αλλαγές [11]. Επίσης, από την έκδοση 5 και μετά, η υποστήριξη εκτέλεσης παράλληλων δοκιμών είναι εγγενής, κάτι που στην προηγούμενη έκδοση 4, μπορούσε να επιτευχθεί με τη χρήση πρόσθετων (ειδικότερα του surefire[3]). Το πλαίσιο μπορεί να «τρέξει» σε όλα τα λειτουργικά συστήματα που υποστηρίζουν τη Java, δηλαδή σε Windows, macOS και Linux και έχει τη δυνατότητα παραγωγής πολύ βασικών αναφορών των δοκιμών σε μορφή XML, ενώ για προηγμένες αναφορές θα πρέπει να συνδυαστεί με κάποιο πρόσθετο (plugin), όπως το JUnit Report Plugin[4] ή το Allure[5] [11]. Υποστηρίζει δοκιμές μονάδας (unit testing), δοκιμές ενσωμάτωσης (integration testing) και δοκιμές παλινδρόμησης (regression testing). Επιπρόσθετα, υποστηρίζει και unit performance testing, μια πολύ περιορισμένη μορφή δοκιμών επιδόσεων που εστιάζει στο επίπεδο της μονάδας. Η χρήση του σε συνδυασμό με άλλα εργαλεία, όπως το Selenium και το Cucumber που θα δούμε παρακάτω, μπορεί να εκτελέσει και δοκιμές API, διεπαφής χρήστη (user interface testing), συστήματος (system testing) και αποδοχής (acceptance testing).

[3] https://maven.apache.org/surefire/maven-surefire-plugin/
[4] https://plugins.jenkins.io/junit/
[5] https://allurereport.org/docs/

Πρόκειται για ένα εξαιρετικά ευέλικτο πλαίσιο και μπορεί να συνεργαστεί με εργαλεία συνεχούς ενσωμάτωσης και παράδοσης (CI/CD συστήματα), όπως για παράδειγμα το δημοφιλές Jenkins[6], καθώς και με συστήματα ελέγχου εκδόσεων (version control), όπως το Git. Αν και το JUnit δεν υποστηρίζει bug tracking και υπηρεσίες stub/mock, η λειτουργία αυτή επιτυγχάνεται με τη χρήση πρόσθετων, όπως το Jira[7] και το Mockito[8], αντίστοιχα. Επίσης, το JUnit δεν παρέχει δυνατότητα αυτόματης παραγωγής δεδομένων, όμως το πρόβλημα λύνεται με τη χρήση του JUnitParams[9] ή του Faker[10]. Επιπρόσθετα, το JUnit δεν διαθέτει δυνατότητα αυτόματης παραγωγής δοκιμών και η δημιουργία των σεναρίων γίνεται εξ ολοκλήρου χειροκίνητα από τον προγραμματιστή. Το ίδιο το πλαίσιο δεν προσφέρει γραφική διεπαφή χρήσης (GUI), ωστόσο μπορεί να χρησιμοποιηθεί μέσα από IDEs, όπως το Eclipse[11] ή το IntelliJ[12], τα οποία προσφέρουν γραφικές δυνατότητες για τη δημιουργία, εκτέλεση και παρακολούθηση δοκιμών. Τέλος, υπάρχει υποστήριξη για γραμμή εντολών (CLI) μέσω του JUnit Platform Console Launcher, που επιτρέπει την εκτέλεση δοκιμών εκτός IDE .

Συνοψίζοντας, το JUnit είναι ιδανικό για μονάδες δοκιμών Java. Είναι δωρεάν, εύχρηστο και επεκτάσιμο. Θα λέγαμε ότι η κύρια χρήση του γίνεται μέσα από περιβάλλοντα ανάπτυξης (IDE – π.χ. NetBeans), καθώς το ίδιο δεν διαθέτει γραφική διεπαφή.

## 4.3 Το TestNG

Το TestNG δημιουργήθηκε το 2004 από τον Cedric Beust της Sun Microsystems. Το «NG» του ονόματός του σημαίνει «Next Generation». Εμπνεύστηκε από το JUnit που αναλύσαμε προηγουμένως, όμως είχε και πρόσθετα χαρακτηριστικά με στόχο να γίνει πιο ευέλικτο και ισχυρό όταν χρησιμοποιείται για τη δοκιμή περίπλοκων εφαρμογών [13]. Όπως και το Junit, έτσι και το TestNG είναι βασισμένο

[6] https://www.jenkins.io/
[7] https://www.atlassian.com/software/jira
[8] https://site.mockito.org/
[9] https://github.com/Pragmatists/JUnitParams
[10] https://github.com/DiUS/java-faker
[11] https://www.eclipse.org/
[12] https://www.jetbrains.com/idea/

στη Java και δεν έχει κόστος απόκτησης διότι είναι ανοικτού κώδικα [13]. Οι χρήστες που έχουν εμπειρία πάνω στο JUnit, θα βρουν πολύ εύκολη τη χρήση του TestNG.

Το πλαίσιο υποστηρίζει δοκιμές μονάδας και ενσωμάτωσης, ενώ μπορεί να επεκταθεί με τη χρήση εργαλείων όπως το Selenium για την υποστήριξη δοκιμών διεπαφής χρήστη και συστήματος. Δεν παρέχει εγγενώς υποστήριξη για δοκιμές API, αποδοχής ή απόδοσης. Ωστόσο, τέτοιες δοκιμές μπορούν να υλοποιηθούν με τη χρήση εξωτερικών βιβλιοθηκών ή εργαλείων, όπως για παράδειγμα η ενσωμάτωση με BDD εργαλεία τύπου Cucumber, που επιτρέπει την πραγματοποίηση δοκιμών αποδοχής με σενάρια σε μορφή φυσικής γλώσσας. Ίσως το μεγαλύτερο πλεονέκτημα του TestNG είναι η δυνατότητα εκτέλεσης παράλληλων δοκιμών σε πολλαπλά νήματα (threads) (αν και πλέον αυτό το χαρακτηριστικό υποστηρίζεται και από το JUnit στην νέα του έκδοση 5.). Δυστυχώς, δεν ενδείκνυται για εκτέλεση δοκιμών (συστήματος) σε εφαρμογές ιστού και δεν είναι συμβατό με φυλλομετρητές, όμως εάν συνδυαστεί με το Selenium, αποτελεί ένα δυνατό εργαλείο δοκιμής εφαρμογών ιστού [13]. Βέβαια, το ίδιο ισχύει και για το JUnit και εν γένει για πλαίσια δοκιμών μονάδων που ενσωματώνονται με το Selenium.

Σε αντίθεση με το JUnit, το TestNG έχει την ικανότητα παραγωγής αναφορών των δοκιμών που εκτελεί. Οι αναφορές αυτές μπορεί να είναι της μορφής HTML ή XML και περιέχουν πληροφορίες σχετικά με τις επιτυχείς, αποτυχείς και μη εκτελεσμένες (skipped) δοκιμές [13]. Είναι επεκτάσιμο και έχει τη δυνατότητα ενσωμάτωσης προσθέτων (plugins), όπως για παράδειγμα το ExtentReports[13], μέσω του οποίου οι αναφορές που παράγονται είναι ακόμα πιο πλούσιες, αναλυτικές και επαγγελματικές, εισάγοντας γραφήματα και διαγράμματα για την παρουσίασή τους. Το TestNG δεν διαθέτει δυνατότητα αυτόματης δημιουργίας δοκιμών. Όλα τα σενάρια δοκιμών δημιουργούνται μέσω χειροκίνητης συγγραφής κώδικα. Ενημερώνεται τακτικά από την κοινότητα, η οποία ζητά νέες δυνατότητες, όσο αυξάνονται και οι αντίστοιχες ανάγκες. Τέλος, είναι συμβατό με όλα τα λειτουργικά συστήματα που υποστηρίζουν την Java (Windows, macOS, Linux).

Συνοψίζοντας, το TestNG διαθέτει περισσότερες ενσωματωμένες δυνατότητες σε σχέση με το JUnit (ειδικά σε σχέση με την προηγούμενη έκδοσή του), όπως είναι η

[13] https://extentreports.com/

εγγενής υποστήριξη παράλληλης εκτέλεσης δοκιμών και η παραγωγή καλύτερων αναφορών HTML/XML χωρίς τη χρήση προσθέτων. Αν και τα δύο πλαίσια «δείχνουν» ισοδύναμα, το TestNG προσφέρει επιπλέον δυνατότητες που δεν έχουν περιληφθεί ρητά στα παραπάνω κριτήρια αξιολόγησης, καθώς η ανάλυση επικεντρώνεται κυρίως σε χαρακτηριστικά ευρύτερης εφαρμογής. Για παράδειγμα, το TestNG υποστηρίζει την ομαδοποίηση των δοκιμών, τη διαχείριση εξαρτήσεων μεταξύ τους και την ενσωματωμένη υλοποίηση δοκιμών οδηγούμενων από δεδομένα (data-driven testing). Αυτά τα χαρακτηριστικά καθιστούν το TestNG πιο κατάλληλο για μεγάλα και σύνθετα έργα, ενώ, από την άλλη, το JUnit υπερτερεί σε απλότητα και ευκολία χρήσης, γεγονός που το καθιστά πιο κατάλληλο για αρχάριους χρήστες ή για έργα μικρότερης πολυπλοκότητας.

## 4.4 Το PyTest

Όπως φαίνεται και από το «Py» στο όνομά του, το PyTest είναι ένα πλαίσιο δοκιμών για εφαρμογές Python. Δημιουργήθηκε το 2004 από τον Holger Krekel και σχεδιάστηκε για να αποτελέσει την αναβάθμιση του ήδη υπάρχοντος πλαισίου δοκιμών της Python, του unittest [14]. Είναι ένα ακόμα δωρεάν πλαίσιο, ανοικτού κώδικα και μπορεί να χρησιμοποιηθεί τόσο για προσωπικά, όσο και για εμπορικά, έργα λογισμικού.

Είναι εύκολο στην εγκατάσταση και χρήση κι αυτό οφείλεται σε μεγάλο βαθμό στην απλότητα της σύνταξης των εντολών, αλλά και στο μικρότερο, συγκριτικά με άλλες αντικειμενοστραφείς γλώσσες, μέγεθος του κώδικα (δοκιμής που παράγεται) [14]. Υποστηρίζει δοκιμές μονάδων, ενσωμάτωσης, συστήματος, API, παλινδρόμησης, διεπαφής χρήστη, αποδοχής και απόδοσης (είτε εγγενώς όπως στην περίπτωση των δοκιμών μονάδων, ενσωμάτωσης, παλινδρόμησης και API είτε με τη βοήθεια προσθέτων/εργαλείων όπως στην περίπτωση δοκιμών αποδοχής, απόδοσης & συστήματος). Είναι ένα επεκτάσιμο εργαλείο και όπως είδαμε στα Junit και TestNG, προκειμένου να μπορεί να υποστηρίξει και άλλα είδη δοκιμών (π.χ. εφαρμογές ιστού), ενσωματώνεται με εξωτερικές βιβλιοθήκες και πρόσθετα, όπως το Selenium. Συνεπώς, μόνο με αυτό τον τρόπο μπορεί να υποστηρίξει την πλήρη δοκιμή εφαρμογών ιστού. Επιπλέον, ενοποιείται με συστήματα CI/CD και συστήματα stub – mock μέσω της βιβλιοθήκης unittest.mock που είναι ενσωματωμένη στην Python. Ένα

ενδιαφέρον χαρακτηριστικό του PyTest είναι πως με τη χρήση του pytest-xdist[14] plugin, μπορεί να εκτελεί δοκιμές παράλληλα, σε πολλαπλές K.M.E. (C.P.U.), ακόμα και σε διαφορετικούς κόμβους ενός δικτύου [14]. Επιπλέον, το πλαίσιο παρέχει τη δυνατότητα της αυτόματης παραγωγής των δεδομένων δοκιμής, μέσω του pytest-factoryboy[15] plugin, ενώ η βιβλιοθήκη hypothesis[16] προσφέρει τη δημιουργία δυναμικών δοκιμών, παράγοντας αυτόματα διαφορετικά δεδομένα εισόδου. Επιπλέον, μέσω του pytest-html[17] παρέχεται η δυνατότητα παραγωγής αναφορών των δοκιμών που εκτελούνται, σε μορφή HTML. Το πλαίσιο ενημερώνεται τακτικά από το δημιουργό, με νέες δυνατότητες και διορθώσεις σφαλμάτων, ενώ η κοινότητα το υποστηρίζει με συνεισφορά σε πρόσθετα και συνεχή ανατροφοδότηση. Μόνο μέσα στο 2024, έχουν κυκλοφορήσει 14 νέες εκδόσεις του πλαισίου. Είναι συμβατό με όλα τα λειτουργικά συστήματα που υποστηρίζουν την Python (Windows, macOS, Linux). Δυστυχώς, το PyTest δεν διαθέτει το δικό του GUI και προτείνεται η χρήση του μέσα από IDEs, όπως το PyCharm που έχει ενσωματωμένη υποστήριξη για το πλαίσιο. Παρέχει διεπαφή μέσω γραμμής εντολών (CLI). Επιπλέον, δεν παρέχει δυνατότητα αυτόματης παραγωγής δοκιμών. Η δημιουργία των σεναρίων γίνεται από τον χρήστη, μέσω Python, χωρίς μηχανισμό record & playback ή κάποιας μορφής αυτόματης συγγραφής κώδικα.

Συμπερασματικά, θα λέγαμε πως το PyTest είναι ένα εύκολο στη χρήση πλαίσιο δοκιμής μονάδας, κατάλληλο και ιδανικό για εφαρμογές Python ενώ δεν περιορίζεται από το μέγεθος του project, οπότε μπορεί να χρησιμοποιηθεί και για επαγγελματική χρήση σε μεγάλα & πολύπλοκα έργα λογισμικού.Παρότι το PyTest δεν υποστηρίζει εγγενώς όλα τα είδη δοκιμών που απαιτούνται για εφαρμογές ιστού, όπως δοκιμές διεπαφής χρήστη, μπορεί να καλύψει με ιδιαίτερη επιτυχία δοκιμές μονάδας και παλινδρόμησης, ακόμα και σε μεγάλα επαγγελματικά έργα.

Σε σύγκριση με το JUnit και το TestNG, υπερτερεί σε ευκολία χρήσης καθώς μειώνει τον boilerplate κώδικα, επιτρέπει την αυτόματη ανακάλυψη δοκιμών και παρέχει συνεπή και ακριβή παραμετροποίηση, καθώς και στον βαθμό αυτοματοποίησης δοκιμών (αυτόματη και εγγενής παραγωγή δεδομένων δοκιμής και

[14] https://pypi.org/project/pytest-xdist/
[15] https://factoryboy.readthedocs.io/en/stable/
[16] https://hypothesis.readthedocs.io/
[17] https://pypi.org/project/pytest-html/

αυτόματη παραγωγή δοκιμών μέσω εξωτερικών εργαλείων) στη γλώσσα Python. Το JUnit και το TestNG, από την άλλη, είναι πιο κατάλληλα για περιβάλλοντα Java (σε αντίθεση με το Python-centric PyTest) και υπερτερούν ως προς την ευκολία υποστήριξης δοκιμών διεπαφής χρήστη (UI) και δοκιμών αποδοχής (acceptance testing). Επιπλέον, το TestNG υπερτερεί έναντι του JUnit και του PyTest προσφέροντας μεγαλύτερη ευελιξία στη διαχείριση σύνθετων σουιτών δοκιμών και στην παραλληλοποίηση. Συνεπώς, η επιλογή ανάμεσα στα τρία πλαίσια εξαρτάται από τη γλώσσα του έργου, τις ανάγκες παραμετροποίησης και την πολυπλοκότητα των δοκιμών.

## 4.5 Το Selenium

Όπως και το JUnit, έτσι και το Selenium αποτελεί πλαίσιο δοκιμών ανοικτού κώδικα. Δημιουργήθηκε το 2004 από τον Jason Huggins. Αποτελεί ίσως το πιο δημοφιλές πλαίσιο δοκιμών για εφαρμογές ιστού [12]. Ειδικότερα, το Selenium υποστηρίζει την εκτέλεση δοκιμών διεπαφή χρήσης (UI testing) και δοκιμών από-σε τελικό σημείο (end-to-end testing). Συνεπώς, καλύπτει την δοκιμή μιας εφαρμογής ιστού από την πλευρά του χρήστη, οπότε είναι ιδανικό για δοκιμές χρηστών (user testing) και δοκιμές αποδοχής (acceptance testing). Όμως, μπορεί να ενσωματώσει κι άλλα πλαίσια δοκιμών για να υποστηρίξει και άλλα είδη δοκιμών. Για παράδειγμα, μπορεί να ενσωματώνεται με πλαίσια δοκιμών μονάδων, όπως το JUnit, και με πλαίσια δοκιμών APIs, όπως είναι το rest-assured[18]. Επιπλέον, αν και το Selenium δεν έχει σχεδιαστεί ειδικά για δοκιμές ενσωμάτωσης, είναι δυνατόν να τις υποστηρίξει, εν μέρει, μέσω της αυτοματοποίησης UI workflows, δηλαδή των βημάτων που ακολουθεί ο χρήστης στη διεπαφή της εφαρμογής. Μέσα από την προσομοίωση αυτών των ροών, το Selenium μπορεί να επαληθεύσει τη σωστή συνεργασία μεταξύ διαφόρων υπομονάδων του συστήματος σε πραγματικές συνθήκες χρήσης.

Θα λέγαμε ότι το Selenium στην πραγματικότητα είναι μία συλλογή από τρία βασικά εργαλεία [12]:

A. Selenium WebDriver: : Επιτρέπει την αλληλεπίδραση με φυλλομετρητές, παρέχοντας ένα API που επιτρέπει την εκτέλεση εντολών και την

[18] https://rest-assured.io/

αλληλεπίδραση με τα στοιχεία κάποιας σελίδας [12]. Μέσω του API αυτού μπορούν να συγγραφούν δοκιμές σε κώδικα διαφορετικών γλωσσών προγραμματισμού, όπως Java, Python, Javascript και C#. Υποστηρίζονται όλοι οι δημοφιλείς φυλλομετρητές, όπως ο Google Chrome, ο Mozilla Firefox, ο Safari, ο Edge της Microsoft, ο ξεπερασμένος πλέον Internet Explorer, ο Opera και ο Brave, καθώς και οι εξής γλώσσες προγραμματισμού: JavaScript, Python, Java, C#, και Ruby [12]. Η αλληλεπίδραση με τον φυλλομετρητή γίνεται μέσω σχετικών οδηγών (πχ. ChromeDriver για τον φυλλομετρητή Google Chrome) ενώ είναι δυνατή η εκτέλεση των δοκιμών χωρίς τη χρήση γραφικής διεπαφής. Σε σύγκριση με το Selenium IDE[19], ο Selenium Web Driver υποστηρίζει πιο πλούσιες αλληλεπιδράσεις με τον φυλλομετρητή (πχ. χειρισμό popup & alert παραθύρων) ενώ η ικανότητα συγγραφής προγραμματιστικού κώδικα επιτρέπει περισσότερη ευελιξία και την κάλυψη πιο προηγμένων σεναρίων δοκιμής. Τέλος, υποστηρίζει ενοποιήσεις με άλλα πλαίσια δοκιμών, όπως τα JUnit & TestNG, καθώς και με εργαλεία αυτοματοποίησης κατασκευής, όπως είναι το Maven[20].

B. Selenium IDE: Είναι ένα πρόσθετο (plugin) για τους φυλλομετρητές Mozilla Firefox και Google Chrome [12]. Μέσω αυτού μπορεί ο χρήστης να δημιουργήσει και να εκτελέσει γρήγορα απλές δοκιμές στην ιστοσελίδα, χωρίς να απαιτούνται γνώσεις προγραμματισμού [12]. Οι δοκιμές παράγονται με βάση την καταγραφή των ενεργειών και αλληλεπιδράσεων του χρήστη με την ιστοσελίδα. Έπειτα, μπορούν να αποθηκευθούν σε αρχεία προγραμμάτων δοκιμών (test scripts) και να οργανωθούν σε σουίτες. Με αυτό τον τρόπο, μπορούν να εκτελούνται μαζί και να πυροδοτούνται από αλλαγές κώδικα. Τα test scripts μπορούν να εξαχθούν σε μορφή κάποιας γλώσσας προγραμματισμού (πχ. Java, Python, Javascript, C#) για περισσότερη προσωποποίηση και επέκταση. Με αυτό τον τρόπο, μπορεί κάποιος να ενσωματώνει assertions (ισχυρισμούς), όπως αν κάποιο στοιχείο της ιστοσελίδας είναι ορατό, αν ένας σύνδεσμος URL είναι σωστός και άλλα. Εφόσον οι δοκιμές παίρνουν την μορφή test scripts, μπορούν να εκτελούνται και με διαφορετικά σύνολα δεδομένων. Τέλος, αξίνει να αναφέρουμε τη

[19] https://www.selenium.dev/selenium-ide/
[20] https://maven.apache.org/

δυνατότητα για οπτική αποσφαλμάτωση μέσω εισαγωγής breakpoints και ελέγχου στα σημεία όπου σταματά η εκτέλεση των δοκιμών για να εξεταστεί η τρέχουσα κατάσταση της εφαρμογής.

C. Selenium Grid: Επιτρέπει την κατανομή των δοκιμών κατά μήκος πολλών υπολογιστών και φυλλομετρητών έτσι ώστε οι δοκιμές να μπορούν να εκτελούνται παράλληλα σε διαφορετικά περιβάλλοντα ή συνθήκες [12]. Υποστηρίζεται master-slave διαμόρφωση όπου ο έλεγχος και η εκκίνηση εκτέλεσης των δοκιμών γίνεται από τον master κόμβο/μηχάνημα, όπου αυτός διανέμει τις δοκιμές προς παράλληλη εκτέλεση στους slave κόμβους (κάθε κόμβος μπορεί να στηρίζεται σε διαφορετικά λειτουργικά συστήματα και φυλλομετρητές). Τέλος, υποστηρίζεται ενοποίηση με CI/CD αγωγούς έτσι ώστε κάθε φορά που γίνεται υποβολή κώδικα και εκτελείται ένας αγωγός, αυτός να επικοινωνεί με τον master κόμβο του Selenium Grid ώστε να εκκινηθεί η εκτέλεση των δοκιμών.

Αν και η εγκατάσταση και ρύθμιση του Selenium (είτε Selenium Web Driver είτε Selenium IDE) είναι εύκολη, το βασικότερο μειονέκτημα του Selenium είναι πως απαιτείται χρόνος για την κατανόηση και χρήση του περιβάλλοντος, ειδικότερα όσον αφορά το Selenium Web Driver & Selenium Grid που ενδείκνυνται για επαγγελματική χρήση. Επομένως, η καμπύλη εκμάθησης είναι αρχικά υψηλή. Φυσικά, η χρήση του πλαισίου γίνεται ευκολότερη μέσω των οδηγών που υπάρχουν και την υποστήριξη της κοινότητας, όμως ένας αρχάριος χρήστης πιθανότατα να δυσκολευτεί, οπότε ίσως αρχικά να καταφύγει στη λύση του Selenium IDE (ώστε με ορισμένες αλληλεπιδράσεις μέσω της χρήσης αυτού του εργαλείου να παρατηρεί το περιεχόμενο των test scripts που παράγονται για να προχωρήσει σε περαιτέρω διαμόρφωση μέσω του Selenium Web Driver). Ένα ακόμα μειονέκτημα είναι το γεγονός ότι το Selenium από μόνο του δεν παράγει αναφορές αποτελεσμάτων των δοκιμών. Ωστόσο, μπορεί να ενσωματωθεί με άλλα εργαλεία (όπως το JUnit που είδαμε ή το TestNG που ακολουθεί), τα οποία προσφέρουν αυτή τη δυνατότητα. Όσον αφορά στην επεκτασιμότητα, το Selenium μπορεί τόσο να ενσωματώσει εργαλεία (π.χ. Jenkins για αυτοματοποίηση), όσο και να επεκταθεί με custom τρόπο (εφόσον είναι ανοικτού κώδικα), ώστε να δώσει λύσεις σε ειδικές συνθήκες ή περιβάλλοντα [12]. Η κοινότητα του Selenium είναι ενεργή και συνεχώς βελτιώνει το πλαίσιο, το οποίο είναι συμβατό με Windows, macOS και Linux.

Ως συμπέρασμα θα λέγαμε ότι το Selenium είναι ένα ισχυρό και δωρεάν πλαίσιο, το οποίο αποτελεί ένα «δυνατό χαρτί» για κάθε δημιουργό/ελεγκτή εφαρμογών ιστού, εφόσον όμως συνδυαστεί και με άλλα εργαλεία που το συμπληρώνουν.

## 4.6 Το Cypress

Το Cypress είναι ακόμα ένα πλαίσιο δοκιμών που εξειδικεύεται σε εφαρμογές ιστού, όπως το Selenium. Δημιουργήθηκε από την εταιρεία Cypress το 2014 και είναι επίσης ανοικτού κώδικα και άρα δωρεάν (ειδικότερα ως προς την κύρια έκδοσή του).

Σε αντίθεση με το Selenium, του οποίου η εγκατάσταση και χρήση απαιτεί περισσότερες γνώσεις, το Cypress είναι πολύ πιο εύκολο και φιλικό προς το χρήστη. Ειδικότερα, η εγκατάστασή του πραγματοποιείται με μια μόνο εντολή.

Το Cypress υποστηρίζει εγγενώς διάφορα είδη δοκιμών που σχετίζονται με εφαρμογές ιστού. Ειδικότερα, προσφέρει πλήρη υποστήριξη για δοκιμές ενσωμάτωσης, συστήματος, παλινδρόμησης, διεπαφής χρήστη (UI) και API. Οι δοκιμές αυτές εκτελούνται στον ίδιο τον φυλλομετρητή, προσφέροντας πλούσια πληροφόρηση για κάθε βήμα της εκτέλεσης και άμεση ορατότητα των αποτελεσμάτων. Όσον αφορά τις δοκιμές μονάδας, το Cypress τις υποστηρίζει εν μέρει, μέσω της λειτουργίας component testing, κυρίως για frameworks ανάπτυξης frontend, όπως το React και το Vue. Ωστόσο, δεν ενδείκνυται για γενικές δοκιμές μονάδας ανεξάρτητου κώδικα (π.χ. λογικές συναρτήσεις χωρίς UI). Αντίστοιχα, οι δοκιμές αποδοχής μπορούν να υλοποιηθούν μέσω σεναρίων ελέγχου με φυσική γλώσσα, αλλά δεν υποστηρίζονται εγγενώς BDD εργαλεία, όπως το Cucumber. Συνεπώς, και αυτές οι δοκιμές χαρακτηρίζονται ως εν μέρει υποστηριζόμενες. Τέλος, το Cypress δεν υποστηρίζει δοκιμές απόδοσης (performance testing), καθώς δεν έχει σχεδιαστεί για benchmarking, stress-testing ή αξιολόγηση των χρόνων απόκρισης κάτω από φορτίο. Για τον σκοπό αυτό χρειάζονται άλλα εξωτερικά εργαλεία όπως το Lighthouse [21]ή το JMeter[22]. Συνεπώς, και σε αυτή την περίπτωση, οι δοκιμές απόδοσης υποστηρίζονται εν μέρει.

[21] https://developer.chrome.com/docs/lighthouse/
[22] https://jmeter.apache.org/

Το περιβάλλον του Cypress διαθέτει ενσωματωμένο φυλλομετρητή και dashboard για να κάνει την παρακολούθηση των δοκιμών ακόμα πιο εύκολη. Σημειώνεται όμως ότι το dashboard απαιτεί συνδρομή επί πληρωμή [16] (οπότε δεν είναι ανοικτού κώδικα). Οι δοκιμές εκτελούνται σε πραγματικό χρόνο, απευθείας στον φυλλομετρητή και το πλαίσιο παρέχει πληροφορίες για κάθε βήμα της δοκιμής που εκτελείται [16]. Ακόμη, υποστηρίζει την αυτοματοποιημένη αναμονή για την εκτέλεση των εντολών ή των ισχυρισμών, έχει ένα απλοποιημένο συντακτικό σε σχέση με το προσφερόμενό του API, κατέχει ισχυρά χαρακτηριστικά αποσφαλμάτωσης (ενοποίηση με Chrome DevTools, επιθεώρηση στοιχείων, οπτικοποίηση καταχωρήσεων κονσόλας και εισαγωγή breakpoints) ενώ επιδεικνύει και πλούσια ενσωματωμένα χαρακτηριστικά, όπως είναι το ταξίδι μέσα στο χρόνο (πλοήγηση σε προηγούμενες εντολές για τον έλεγχο της τότε κατάστασης της εφαρμογής) καθώς και η λήψη screenshots και βίντεο.

Το πλαίσιο ενημερώνεται τακτικά από το δημιουργό, έχοντας λάβει περισσότερες από 15 ενημερώσεις από την αρχή του έτους. Υποστηρίζει πρόσθετα (plugins), όμως όχι σε τόσο μεγάλο βαθμό, όσο ο αντίπαλός του, το Selenium. Ένα αρχικό μειονέκτημα του Cypress ήταν το γεγονός ότι υποστήριζε κυρίως φυλλομετρητές που βασίζονται σε Chrome (π.χ. Google Chrome, Chromium, Edge) [16]. Όμως, από την έκδοση 4.0 κι έπειτα, υποστηρίζεται και ο Mozilla Firefox. Επιπλέον, παρέχεται περιορισμένη υποστήριξη για τον Safari μέσω της χρήσης του WebKit. Δυστυχώς, ο παλιός Internet Explorer δεν υποστηρίζεται καθόλου [16] αν και αυτό δεν θεωρείται σημαντικό μειονέκτημα διότι ο εν λόγω φυλλομετρητής θεωρείται πλέον αρκετά ξεπερασμένος. Όσον αφορά τα λειτουργικά συστήματα, το Cypress υποστηρίζει Windows, macOS και Linux [16].

Η εκτέλεση παράλληλων δοκιμών υποστηρίζεται κανονικά κατά μήκος διαφορετικών περιβαλλόντων, όμως απαιτείται η κατάλληλη διαμόρφωση του Cypress Dashboard για το σκοπό αυτό. Δεν είναι κάτι δύσκολο προς επίτευξη και υπάρχει σχετική, καλογραμμένη τεκμηρίωση. Η υποστήριξη εκτέλεσης παράλληλων δοκιμών κατά μήκος διαφορετικών φυλλομετρητών δεν υποστηρίζεται άμεσα διότι κάθε εκτέλεση μπορεί να αφορά μόνο έναν φυλλομετρητή. Θα πρέπει να διαμορφωθεί ένας αγωγός CI/CD ώστε να εκτελεί παράλληλα διαφορετικές δοκιμές κατά μήκος φυλλομετρητών. Εναλλακτικά, μπορεί να γίνει η χρήση εργαλείων όπως είναι το

BrowserStack[23], το οποίο επιτρέπει την εκτέλεση δοκιμών κατά μήκος διαφορετικών φυλλομετρητών.

Το Cypress συνεργάζεται άψογα με συστήματα ελέγχου εκδόσεων (version control) και διαθέτει ενσωματωμένη υποστήριξη mock-stub (cy.stub και cy.spy). Επίσης, μπορεί να ενσωματωθεί με συστήματα παρακολούθησης σφαλμάτων (bug tracking), αν και το πλαίσιο το ίδιο παρέχει μία βασική καταγραφή σφαλμάτων με screenshots και logs. Σε αντίθεση με το Selenium, εδώ υποστηρίζεται η αυτόματη παραγωγή δυναμικών δεδομένων, μέσω της χρήσης JavaScript. Επιπλέον, το Cypress παρέχει ενσωματωμένη δυνατότητα αυτόματης παραγωγής δοκιμών μέσω του γραφικού περιβάλλοντος Test Runner[24]. Οι ενέργειες του χρήστη καταγράφονται και μετατρέπονται σε βήματα δοκιμής, ενώ παρέχεται και δυνατότητα αναπαραγωγής και τροποποίησης αυτών.

Σε γενικές γραμμές, το Cypress εμφανίζει σημαντικά πλεονεκτήματα έναντι του Selenium σε αρκετά σημεία, σύμφωνα και με τα αποτελέσματα της σύγκρισης βάσει των κριτηρίων. Ειδικότερα, το Cypress προσφέρει ενσωματωμένη υποστήριξη για δοκιμές διεπαφής χρήστη, συστήματος, API, και παλινδρόμησης, χωρίς την ανάγκη χρήσης εξωτερικών πρόσθετων. Παράλληλα, υποστηρίζει την αυτόματη παραγωγή δοκιμών και δεδομένων δοκιμής, παρέχει γραφική διεπαφή (GUI) μέσω του Test Runner, διαθέτει μηχανισμούς mocking και bug tracking, και ενοποιείται άμεσα με εργαλεία CI/CD. Η ευκολία στη χρήση, η φιλικότητα προς τον προγραμματιστή και η έμφαση στην ταχύτητα ανάπτυξης, το καθιστούν ιδιαίτερα κατάλληλο για σύγχρονα έργα λογισμικού που αναπτύσσονται με ευέλικτες μεθοδολογίες (Agile). Σε αντίθεση με το Selenium, το οποίο είναι προτιμότερο για περιπτώσεις που απαιτούν ευρεία υποστήριξη φυλλομετρητών και προηγμένη παραμετροποίηση, το Cypress υπερέχει σε εμπειρία χρήσης, όταν πρόκειται για web εφαρμογές σύγχρονης αρχιτεκτονικής. Ωστόσο, θα πρέπει να σημειωθεί ότι ορισμένες προηγμένες λειτουργίες, όπως είναι η χρήση του Cypress Dashboard για αποθήκευση αποτελεσμάτων και εκτέλεση παράλληλων δοκιμών στο νέφος (cloud), προσφέρονται μόνο στην «επί πληρωμή»

[23] https://www.browserstack.com/
[24] https://docs.cypress.io/guides/core-concepts/test-runner

έκδοση, γεγονός που καθιστά τη δωρεάν έκδοσή του μη κατάλληλη επιλογή για πολύπλοκα έργα.

## 4.7 Το Playwright

Το Playwright αναπτύχθηκε από τη Microsoft και παραδόθηκε η πρώτη του έκδοση την 31 Ιανουαρίου 2020. Πρόκειται για ακόμα ένα δωρεάν στη χρήση πλαίσιο δοκιμών εφαρμογών ιστού και ανοικτού κώδικα, κάτι που σίγουρα το καθιστά ελκυστικό [18].

Σε γενικές γραμμές, υποστηρίζει τα ίδια είδη δοκιμών με το Cypress που «είδαμε» πιο πάνω (με μια μικρή υπεροχή ως προς την εγγενή υποστήριξη σε δοκιμές μονάδων). Ωστόσο, το Playwright φαίνεται να υπερτερεί ελαφρώς σε ορισμένα σημεία, όπως η εγγενής υποστήριξη για δοκιμές μονάδας μέσω της επίσημης βιβλιοθήκης δοκιμών, ενώ το Cypress παρέχει πιο φιλικό και διαδραστικό περιβάλλον αποσφαλμάτωσης. Το Cypress υποστηρίζει πολλαπλές γλώσσες προγραμματισμού, όπως Java, Javascript, Python και C# [18]. Χάρη στην καθαρή σύνταξη του API του, είναι εύκολο στη χρήση και υποστηρίζει τη δυνατότητα δημιουργίας δοκιμών μέσα από εντολές καταγραφής [18]. Το εργαλείο αυτό ονομάζεται Playwright Codegen[25] και πρόκειται στην ουσία για μία εφαρμογή ιστού, στην οποία ο χρήστης αλληλοεπιδρά (με μια ιστοσελίδα) και καταγράφει ενέργειες (click, συμπλήρωση πεδίων κ.α.) [18]. Οι ενέργειες αυτές μεταφράζονται αυτόματα σε κώδικα και σενάρια δοκιμών [18]. Αυτό όμως δε σημαίνει ότι δεν απαιτούνται ορισμένες βασικές γνώσεις προγραμματισμού και έτσι, θα λέγαμε, πως δεν είναι το πιο φιλικό προς τους αρχάριους χρήστες πλαίσιο, συγκριτικά με προαναφερόμενα, όπως για παράδειγμα το Cypress. Σε καμία περίπτωση όμως δεν «αγγίζει» τη δυσκολία σε χρήση του Selenium.

Το Playwright είναι επεκτάσιμο και μπορεί να συνδεθεί με συστήματα CI/CD, bug tracking και version control, ενώ η υποστήριξη stub/mock είναι ενσωματωμένη [18]. Όσον αφορά τις διεπαφές, το πλαίσιο βασίζεται κυρίως στη διεπαφή μέσω γραμμής εντολών (CLI) για τη δημιουργία, τη διαχείριση και την εκτέλεση των δοκιμών, όμως, όπως αναφέραμε παραπάνω, μέσω του Playwright Codegen παρέχεται μία πιο οπτική εμπειρία χρήσης. Χωρίς όμως να μπορούμε να το θεωρήσουμε ως ένα πλήρες

[25] https://playwright.dev/docs/codegen

GUI σύστημα και θα λέγαμε πως πρόκειται περισσότερο για ένα υποστηρικτικό εργαλείο [18]. Όπως και το Cypress, έτσι και το Playwright, έχει τη δυνατότητα αυτόματης παραγωγής δυναμικών δεδομένων μέσα από τη χρήση της JavaScript. Επίσης, με το Codegen υποστηρίζεται η αυτόματη δημιουργία δοκιμών. Επιπλέον, υποστηρίζεται πλήρως η εκτέλεση πολλαπλών δοκιμών παράλληλα, ενώ μέσω του playwright test --reporter html λαμβάνουμε και αναφορές των δοκιμών σε μορφή html [18].

Η κοινότητα υποστηρίζει ενεργά το Playwright, παρέχοντας οδηγούς εγκατάστασης, χρήσης και παραμετροποίησης του πλαισίου [18], το οποίο «μετρά» περισσότερες από 40 αναβαθμίσεις με την τελευταία έκδοση να είναι η 1.49. Είναι συμβατό με όλα τα «μεγάλα» λειτουργικά συστήματα (Windows, Linux και macOS), όπως επίσης, υποστηρίζονται όλοι οι δημοφιλείς φυλλομετρητές (Chrome, Edge, Safari και Firefox).

Τελικά, το Playwright είναι ένα μοντέρνο και «φρέσκο» πλαίσιο που «κάθεται» ανάμεσα στην ευκολία και ταχύτητα του Cypress και την ευρεία υποστήριξη και παραμετροποίηση του Selenium. Σε σύγκριση με το Cypress, υπερτερεί κυρίως στην υποστήριξη παράλληλων δοκιμών και δοκιμών μονάδας, στο μηδενικό κόστος χρήσης και στη συμβατότητα με όλους τους βασικούς φυλλομετρητές. Ωστόσο, το Cypress παραμένει ισχυρότερο σε άλλους τομείς, όπως η ευκολία χρήσης, η ύπαρξη ολοκληρωμένου γραφικού περιβάλλοντος, η παραγωγή αναφορών και η εγγενής υποστήριξη CI/CD & bug tracking.

## 4.8 Το TestCafe

Η DevExpress παρέδωσε μια πρώτη έκδοση του TestCafe τον Νοέμβριο του 2016 [19]. Επρόκειτο για ένα πλαίσιο δοκιμής εφαρμογών ιστού, ανοικτού κώδικα (ως προς την κύρια εκδοχή του) και πλήρως δωρεάν για το χρήστη [19]. Έκτοτε, το πλαίσιο έχει αναβαθμιστεί με νέες εκδόσεις, οι οποίες διόρθωσαν προηγούμενα σφάλματα, με την τελευταία έκδοση να είναι η 3.7.2 που δημοσιεύθηκε το Φεβρουάριο του 2025 [19], όμως φαίνεται ότι σε σύγκριση με άλλα πλαίσια δοκιμών, το πλαίσιο δε δέχεται τόσο συχνά ενημερώσεις.

Το TestCafe υποστηρίζει τις δοκιμές διεπαφής χρήστη (UI Testing), συστήματος (System Testing), παλινδρόμησης (Regression Testing) και API, καθιστώντας το κατάλληλο για εκτεταμένες δοκιμές λειτουργικότητας σε εφαρμογές ιστού. Ωστόσο, δεν υποστηρίζει δοκιμές μονάδας (Unit Testing), καθώς δεν έχει σχεδιαστεί για να επικεντρώνεται σε τμήματα εσωτερικού κώδικα. Οι δοκιμές ενσωμάτωσης και αποδοχής μπορούν να υλοποιηθούν μόνο μέσω προσεκτικού σχεδιασμού των σεναρίων, αλλά δεν υποστηρίζονται πλήρως ή εγγενώς από το πλαίσιο. Επίσης, το πλαίσιο δεν υποστηρίζει δοκιμές απόδοσης (Performance Testing), κάτι που περιορίζει τη χρήση του σε σενάρια που δεν απαιτούν μέτρηση ταχύτητας ή φορτίου. Είναι πολύ φιλικό προς τον αρχάριο χρήστη, καθώς δεν απαιτεί εξειδικευμένες γνώσεις προγραμματισμού ή γνώσεις ρύθμισης φυλλομετρητών και εξωτερικών οδηγών. Είναι συμβατό με Windows, Linux και macOS, ενώ υποστηρίζει όλους τους σύγχρονους φυλλομετρητές (Chrome, Firefox, Safari και Edge κ.α.) [19]. Επίσης, είναι ικανό να πραγματοποιεί δοκιμές παράλληλα και η έκδοση «TestCafe Studio[26]» διαθέτει ένα φιλικό προς το χρήστη GUI [19]. Αντίθετα, η δωρεάν έκδοση υποστηρίζει διεπαφή μέσω χρήσης γραμμής εντολών (CLI). Επιπλέον, η «Studio» έκδοση προσφέρει μία παραγωγή εμπλουτισμένων αναφορών δοκιμών μέσω του GUI, ενώ η δωρεάν έκδοση μία πολύ βασική, η οποία μπορεί να επεκταθεί μόνο με τη χρήση εξωτερικών εργαλείων, εφόσον αυτό είναι ζητούμενο (πχ. ενδεχομένως σε μεγάλα έργα) [19]. Τέλος, το TestCafe δεν προσφέρει πλήρως ανεπτυγμένο μηχανισμό αυτόματης παραγωγής δοκιμών. Υποστηρίζει όμως δυνατότητες οπτικής παρακολούθησης και τροποποίησης των δοκιμών, γεγονός που μπορεί να διευκολύνει τη δημιουργία τους, χωρίς ωστόσο να παρέχεται μία πλήρης λειτουργία record & playback.

Όσον αφορά στην επεκτασιμότητα του TestCafe, υποστηρίζεται η σύνδεσή του με CI/CD (π.χ. Jenkins), version control (π.χ. Git), bug tracking (π.χ. Jira) και stub/mock συστήματα [19]. Να τονίσουμε, εδώ, ότι καμία από αυτές τις λειτουργίες δεν παρέχεται από το πλαίσιο και η σύνδεσή του με τρίτα εργαλεία είναι ο μόνος τρόπος για την εκτέλεση τέτοιων λειτουργιών, ενώ στο Cypress που αναλύσαμε προηγουμένως, οι λειτουργίες bug tracking και stub/mock είναι ενσωματωμένες στο πλαίσιο. Τέλος, όπως αναφέραμε και στα Cypress και Selenium, έτσι κι εδώ δεν

[26] https://www.devexpress.com/products/testcafestudio/

περιλαμβάνεται στις δυνατότητες του TestCafe η αυτόματη παραγωγή δεδομένων δοκιμών εισόδου και δοκιμών. Επομένως, και το TestCafe θα πρέπει να συνδυαστεί με τρίτα εργαλεία, εφόσον ο χρήστης αναζητά τέτοιες δυνατότητες.

Συμπερασματικά, το TestCafe κατατάσσεται στην ίδια ομάδα πλαισίων με το Cypress, το Playwright και το Selenium. Σε σύγκριση, όμως, φαίνεται να πλησιάζει περισσότερο στο Cypress όσον αφορά τη φιλικότητα προς τον χρήστη, την συμβατότητα με φυλλομετρητές και την διεπαφή GUI. Επίσης, όπως το Playwright και το Selenium ξεχωρίζει ως προς την ικανότητα υποστήριξης παράλληλων δοκιμών. Υστερεί όμως σε σημεία όπως η ενσωματωμένη υποστήριξη για mocking και bug tracking και η αυτόματη παραγωγή δοκιμών και δεδομένων δοκιμών. Η δωρεάν έκδοσή του περιορίζεται κυρίως σε CLI και βασικές αναφορές, ενώ η έκδοση "TestCafe Studio" καλύπτει περισσότερες ανάγκες, προσφέροντας GUI και εμπλουτισμένες αναφορές. Συνεπώς, παρότι δεν προσφέρει το εύρος δυνατοτήτων του Selenium ή την πληρότητα του Playwright, αποτελεί μια αξιόλογη επιλογή για ομάδες που αναζητούν ένα απλό, γρήγορο και εύχρηστο εργαλείο για λειτουργικές δοκιμές ιστού, χωρίς ιδιαίτερες ανάγκες παραμετροποίησης ή επέκτασης.

## 4.9 Το Appium

Το Appium είναι ένα ακόμα ανοικτού κώδικα πλαίσιο δοκιμών για mobile και web εφαρμογές, το οποίο δημιουργήθηκε το 2011 από τον Dan Cuellar, αρχικά ως ένα προσωπικό έργο που στόχευε στη βελτίωση της διαδικασίας δοκιμών σε εφαρμογές iOS [20]. Σύντομα προσέλκυσε την προσοχή της κοινότητας και εξελίχθηκε, με τη συμβολή των Sauce Labs που ανέλαβαν και τη διαχείρισή του [20]. Από το 2013, το Appium έχει καθιερωθεί ως ένα από τα πιο δημοφιλή εργαλεία για την εκτέλεση δοκιμών σε κινητές συσκευές, υποστηρίζοντας εφαρμογές Android και iOS, καθώς και υβριδικές εφαρμογές που βασίζονται σε τεχνολογίες ιστού. Η αρχιτεκτονική του βασίζεται στο πρότυπο WebDriver, όπως είδαμε και στο το Selenium [20]. Η τελευταία έκδοση του Appium είναι η 2.17.1, η οποία κυκλοφόρησε τον Μάρτιο του 2025. Υποστηρίζεται ενεργά από μια μεγάλη κοινότητα και λαμβάνει συχνές ενημερώσεις, με σταθερές βελτιώσεις και νέες δυνατότητες που ακολουθούν την εξέλιξη των mobile λειτουργικών συστημάτων.

Το Appium υποστηρίζει δοκιμές αποδοχής, συστήματος, παλινδρόμησης και διεπαφής χρήστη για κινητές εφαρμογές, ενώ μέσω κατάλληλων επεκτάσεων μπορεί να χρησιμοποιηθεί και για δοκιμές APIs και ενσωμάτωσης. Παρέχει υποστήριξη για δοκιμές σε πραγματικές συσκευές, σε εξομοιωτές και προσομοιωτές (emulators/simulators), καθώς και σε cloud περιβάλλοντα μέσω ενσωμάτωσης με συστήματα όπως το BrowserStack. Δεν απαιτεί τροποποίηση του κώδικα της εφαρμογής, γεγονός που αποτελεί ένα από τα βασικά πλεονεκτήματά του. Όσον αφορά τη συμβατότητα με γλώσσες προγραμματισμού, το Appium υποστηρίζει Java, Python, JavaScript, Ruby, C#, Kotlin και PHP και φυσικά είναι πλήρως συμβατό με τα λειτουργικά συστήματα Windows, macOS και Linux [20]. Επίσης, ενσωματώνεται εύκολα σε CI/CD pipelines, όπως Jenkins, GitLab CI[27] και CircleCI[28][20].

Η εγκατάστασή του είναι σχετικά εύκολη, όμως η χρήση του απαιτεί κάποιες βασικές γνώσεις προγραμματισμού και κατανόηση της αρχιτεκτονικής WebDriver, γεγονός που το καθιστά πιο απαιτητικό για κάποιο αρχάριο χρήστη. Παρόλα αυτά, η ενεργή κοινότητα και η πλούσια τεκμηρίωσή του βοηθούν στη γρήγορη εξοικείωση με το εργαλείο. Είναι επεκτάσιμο μέσω plugins και υποστηρίζει την παράλληλη εκτέλεση δοκιμών σε πολλές συσκευές [20]. Επίσης, επιτρέπει την αυτόματη παραγωγή δεδομένων δοκιμών μέσω της χρήσης εξωτερικών βιβλιοθηκών (π.χ. Faker) και την παραγωγή αναφορών είτε μέσω CLI είτε με την ενσωμάτωση σε εργαλεία όπως το Allure ή το ExtentReports που είδαμε προηγουμένως [20]. Η βασική έκδοση του Appium δεν υποστηρίζει εγγενώς την αυτόματη δημιουργία δοκιμών. Ωστόσο, μέσω του εξωτερικού εργαλείου Appium Studio[29] παρέχεται δυνατότητα καταγραφής ενεργειών και παραγωγής σεναρίων, γεγονός που προσφέρει περιορισμένη υποστήριξη στο συγκεκριμένο κριτήριο. Τέλος, δεν διαθέτει ενσωματωμένο GUI και υποστηρίζει τη χρήση διεπαφής γραμμής εντολών (CLI), ενώ για οπτικοποίηση μπορεί να συνδυαστεί με άλλα εργαλεία.

Συνοψίζοντας, αν και το Appium ακολουθεί παρόμοια αρχιτεκτονική με εκείνη του Selenium, καθώς και τα δύο βασίζονται στο WebDriver, οι δυνατότητες των δύο δεν είναι παρόμοιες. Το Appium υποστηρίζει τη δοκιμή native, υβριδικών και κινητών

[27] https://docs.gitlab.com/ci/
[28] https://circleci.com/
[29] https://digital.ai/products/continuous-testing/appium-studio/

εφαρμογών ιστού. Αυτό σημαίνει ότι καλύπτει τρία υπο-είδη εφαρμογών και όχι μόνο ένα, ενώ ο ίδιος κώδικας δοκιμής μπορεί να εφαρμόζεται σε όλα τα υποστηριζόμενα είδη, καθιστώντας τα κριτήρια σύγκρισης με τα Cypress, Playwright και TestCafe εξίσου σημαντικά για πολλές λειτουργίες. ο Appium απευθύνεται σε ένα εντελώς διαφορετικό πεδίο εφαρμογής και προσφέρει λύσεις που σχετίζονται με τη δοκιμή λογισμικού σε φορητές συσκευές. Η χρήση του σε έργα μεγάλης κλίμακας είναι εφικτή, καθώς διαθέτει τα απαραίτητα εργαλεία και επεκτάσεις που απαιτούνται για να ενσωματωθεί του σε επαγγελματικές ροές εργασίας. Συμπερασματικά, ενώ η απόδοσή του (performance) είναι χειρότερη σε σύγκριση με τα άλλα πλαίσια της ίδιας κατηγορίας σε πολλά κριτήρια, ξεχωρίζει χάρη στην εστίασή του στις κινητές εφαρμογές και στην εγγενή υποστήριξη σε δοκιμές αποδοχής.

## 4.10 Το Cucumber

Το Cucumber παραδόθηκε στην κοινότητα το 2008 από τον Aslak Hellesøy και μία μικρή ομάδα (προγραμματιστών) ως δωρεάν και ανοικτού κώδικα λογισμικό [22]. Αρχικά, σχεδιάστηκε ως εργαλείο δοκιμών βασισμένο στη γλώσσα Ruby [22]. Στη συνέχεια, στόχος της ομάδας κατέστη η δημιουργία ενός πλαισίου που να βασίζεται στη μεθοδολογία Behavior-Driven Development (BDD), χρησιμοποιώντας μία φυσική γλώσσα σε αγγλικά, τη Gherkin[30], για τη δημιουργία των δοκιμών [22]. Ακολούθησε η υποστήριξη και άλλων γλωσσών προγραμματισμού, πέρα από τη Ruby, όπως Java και Python και έτσι το εργαλείο απέκτησε γρήγορα δημοτικότητα [22]. Η τελευταία έκδοση του πλαισίου κυκλοφόρησε την 15η Νοεμβρίου 2024 ενώ η κοινότητα συνεισφέρει στη συνεχή βελτίωση και προσαρμογή του στις σύγχρονες ανάγκες. Όμως, σε σύγκριση με τα προηγούμενα πλαίσια, φαίνεται πως η συχνότητα των ενημερώσεων είναι μικρότερη και οι νέες εκδόσεις αφορούν κυρίως τη σταθερότητα του συστήματος.

Χρησιμοποιείται εγγενώς για δοκιμή αποδοχής, και παλινδρόμησης ενώ υποστηρίζει δοκιμές συστήματος, ενσωμάτωσης, UI, API και απόδοσης μέσω ενσωμάτωσης με άλλα πλαίσια (όπως JUnit, Rest-assured, Selenium, Gatlin[31] &

[30] https://cucumber.io/docs/gherkin/
[31] https://gatling.io/

JMeter[32]). Χρησιμοποιούνται λέξεις όπως Given, When και Then για να περιγράψουν διαφορετικά «κομμάτια» ενός σεναρίου, που αφορούν στη δράση που επιτελείται και το αναμενόμενο αποτέλεσμα [21]. Η διατύπωση αυτή καθιστά τις δοκιμές που σχεδιάζονται εύκολες στην κατανόηση από όλους τους εμπλεκόμενους. Στη συνέχεια, τα σενάρια που δημιουργούνται μετατρέπονται από το Cucumber σε εκτελέσιμο κώδικα [21]. Πρόκειται για ένα επεκτάσιμο και ευέλικτο πλαίσιο που μπορεί να ενσωματωθεί με το Selenium για έλεγχο διεπαφής χρήστη, το Appium για δοκιμές εφαρμογών Android και iOS καθώς και με εργαλεία, όπως το Rest-Assured για δοκιμές API και το JMeter ή Gatlin για δοκιμές απόδοσης.

Υποστηρίζει τη σύνδεση με εργαλεία CI/CD για αυτόματη εκτέλεση δοκιμών μετά από αλλαγές στον κώδικα [21]. Οι αναφορές που παράγονται από το πλαίσιο είναι είτε σε μορφή κειμένου στην κονσόλα (CLI), είτε της μορφής HTML με τη χρήση του πρόσθετου Cucumber Reporting[33] [21]. Οι δεύτερες είναι εμπλουτισμένες με γραφικά και στατιστικά, καθώς και λεπτομέρειες των αποτυχιών. Τέλος, μπορεί να λειτουργήσει σε Windows, Linux και macOS, ενώ υποστηρίζει όλους τους μοντέρνους φυλλομετρητές, εφ' όσον ενσωματωθεί, ενδεικτικά, με το Selenium ή το Playwright.

Ενώ το πλαίσιο δε διαθέτει από μόνο του κάποιο bug tracking και stub/mock σύστημα, μπορεί να ενσωματωθεί με τρίτα εργαλεία για το σκοπό αυτό. Δυστυχώς, δε διαθέτει GUI (θα πρέπει να χρησιμοποιηθεί κάποιο άλλο εργαλείο, όπως το εμπορικό Cucumber Studio προς αυτό τον σκοπό) και υποστηρίζει μόνο τη χρήση με γραμμή εντολών (CLI). Τέλος, δεν υποστηρίζεται η αυτόματη παραγωγή δεδομένων δοκιμών ή η αυτόματη παραγωγή δοκιμών. Επομένως, κι εδώ απαιτείται η χρήση εξωτερικών εργαλείων (π.χ. Faker) γι αυτή τη δυνατότητα, ενώ η παράλληλη εκτέλεση δοκιμών επιτυγχάνεται μόνο με το συνδυασμό του, ενδεικτικά, με TestNG ή Selenium Grid.

Συνοψίζοντας, το Cucumber είναι ένα πλαίσιο, ιδιαίτερα χρήσιμο σε περιβάλλοντα όπου η συνεργασία μεταξύ διαφορετικών ρόλων είναι κρίσιμη. Γεφυρώνει το χάσμα επικοινωνίας ανάμεσα στους προγραμματιστές, τους ελεγκτές (testers) και γενικότερα στους εμπλεκόμενους σε ένα έργο. Επιπλέον, υποστηρίζεται η συγγραφή των δοκιμών σε φυσική γλώσσα (οπότε έχουμε ανεξαρτησία από τη

[32] https://jmeter.apache.org/
[33] https://github.com/damianszczepanik/cucumber-reporting

γλώσσα προγραμματισμού), ενώ παράλληλα διευκολύνεται η ανάπτυξη έργων λογισμικού εφαρμόζοντας τη νεωτεριστική, ευέλικτη μεθοδολογία BDD.

## 4.11 Το Gauge

Το Gauge είναι ένα ακόμα ανοικτού κώδικα πλαίσιο δοκιμών, που σχεδιάστηκε από την εταιρεία ThoughtWorks για να υποστηρίξει σενάρια δοκιμών αποδοχής και BDD (Behavior Driven Development)[27]. Εστιάζει στη δημιουργία δοκιμών σε μορφή φυσικής γλώσσας, οι οποίες γράφονται σε αρχεία Markdown (.spec), καθιστώντας τα εύκολα κατανοητά ακόμα και από μη τεχνικούς χρήστες.

Οι εντολές των δοκιμών υλοποιούνται στον κώδικα, σε γλώσσες όπως Java, JavaScript, Python, C#, Kotlin και Ruby, και «συνδέονται» με τις γραπτές προτάσεις του σεναρίου. Το Gauge υποστηρίζει τις εξής κατηγορίες δοκιμών: δοκιμές αποδοχής (acceptance testing), δοκιμές διεπαφής χρήστη (UI testing), δοκιμές ενσωμάτωσης και παλινδρόμησης, μέσω της ενοποίησης με εργαλεία όπως το Selenium ή το Appium[28]. Δεν υποστηρίζει δοκιμές μονάδας ή απόδοσης, ενώ οι δοκιμές API μπορούν να υλοποιηθούν μέσω της σύνδεσης με βιβλιοθήκες όπως το REST-assured ή το HTTPClient.

Επιτρέπει την εκτέλεση παράλληλων δοκιμών, υποστηρίζει ενοποίηση με CI/CD αγωγούς (όπως Jenkins, GitHub Actions, GitLab CI) και παρέχει αναφορές εκτέλεσης σε HTML ή JSON μορφή μέσω plugins. Η παραγωγή των δοκιμών δεν γίνεται αυτόματα – ο χρήστης δημιουργεί τόσο τα σενάρια όσο και τη σύνδεση με τον κώδικα χειροκίνητα. Δεν διαθέτει δική του γραφική διεπαφή (GUI), αλλά μπορεί να χρησιμοποιηθεί μέσα από IDEs (όπως το VS Code) με υποστήριξη σχετικών plugins. Επιπλέον, δεν προσφέρει μηχανισμό record & playback[28].

Όσον αφορά την επεκτασιμότητα, το Gauge υποστηρίζει την ενσωμάτωση με τρίτα εργαλεία όπως το Selenium, Appium, Docker και Git, καθώς και με συστήματα mocking και bug tracking (π.χ. μέσω συνδυασμού με WireMock, Jira κ.ά.). Η κοινότητα είναι ενεργή και συνεχώς εμπλουτίζει το πλαίσιο με νέα plugins και βιβλιοθήκες. Είναι συμβατό με όλα τα βασικά λειτουργικά συστήματα (Windows, macOS και Linux).

Συνοψίζοντας, το Gauge είναι ένα ελαφρύ και ευέλικτο εργαλείο για αποδοχή και BDD δοκιμές, ιδανικό για συνεργατικά περιβάλλοντα ανάπτυξης λογισμικού στα οποία συμμετέχουν τόσο τεχνικά όσο και μη τεχνικά μέλη της ομάδας.

## 4.12 Το Robot

Το Robot Framework είναι ένα ισχυρό πλαίσιο αυτοματοποίησης ανοικτού κώδικα για δοκιμές αποδοχής και ATDD (Acceptance Test Driven Development)[29]. Δημιουργήθηκε το 2008 και είναι γραμμένο σε Python, αν και υποστηρίζει και άλλες γλώσσες μέσω βιβλιοθηκών. Βασίζεται στη φιλοσοφία των λέξεων-κλειδιών (keyword-driven testing), όπου οι δοκιμές γράφονται με προκαθορισμένα keywords που περιγράφουν τις ενέργειες προς εκτέλεση[30].

Οι δοκιμές δημιουργούνται σε αρχεία .robot ή .txt, είναι απλές στην ανάγνωση και εύκολα επεκτάσιμες. Το πλαίσιο υποστηρίζει δοκιμές αποδοχής (acceptance testing), διεπαφής χρήστη (UI testing), παλινδρόμησης, συστήματος, ενσωμάτωσης και API, με χρήση βιβλιοθηκών όπως η SeleniumLibrary, AppiumLibrary, RESTinstance και άλλες. Δεν υποστηρίζει δοκιμές απόδοσης εγγενώς, αλλά μπορεί να συνδυαστεί με εργαλεία όπως το JMeter για τον σκοπό αυτό. Η υποστήριξη δοκιμών μονάδας είναι περιορισμένη, καθώς δεν έχει σχεδιαστεί για unit testing[30].

Το Robot Framework ενσωματώνεται άψογα με CI/CD εργαλεία (όπως Jenkins, GitHub Actions, Azure DevOps) και συστήματα version control όπως το Git. Υποστηρίζει την παραγωγή αναφορών σε HTML και log αρχείων με λεπτομέρειες εκτέλεσης, ενώ μέσω της χρήσης εργαλείων όπως το Allure ή το ReportPortal μπορεί να εμπλουτιστεί με διαδραστικές αναφορές. Επίσης, μπορεί να εκτελεί δοκιμές παράλληλα μέσω του εργαλείου Pabot[30].

Η δημιουργία δοκιμών γίνεται χειροκίνητα, όμως η απλότητα της σύνταξης με keywords καθιστά το εργαλείο ιδιαίτερα προσβάσιμο. Υπάρχει διαθέσιμο GUI εργαλείο (RIDE – Robot Framework IDE) για πιο οπτική συγγραφή και διαχείριση δοκιμών, αν και πολλές ομάδες χρησιμοποιούν το Robot μέσα από text editors ή IDEs όπως το VS Code ή το PyCharm.

Το πλαίσιο διαθέτει ενεργή κοινότητα και υποστηρίζεται από το Robot Framework Foundation, με τακτικές κυκλοφορίες νέων εκδόσεων, προσθήκες βιβλιοθηκών και τεκμηρίωση. Είναι πλήρως συμβατό με Windows, macOS και Linux.

Συνοψίζοντας, το Robot Framework είναι ένα ευέλικτο εργαλείο αποδοχής και end-to-end δοκιμών, με έμφαση στην αναγνωσιμότητα και την επεκτασιμότητα, κατάλληλο για ομάδες που επιθυμούν keyword-driven testing χωρίς απαραίτητα να διαθέτουν έντονες προγραμματιστικές γνώσεις.

Σε σύγκριση με τα υπόλοιπα πλαίσια της ίδιας κατηγορίας (Cucumber και Gauge), το Robot Framework ακολουθεί διαφορετική προσέγγιση μέσω της φιλοσοφίας των λέξεων-κλειδιών (keyword-driven), σε αντίθεση με το Cucumber και το Gauge που βασίζονται στη συμπεριφορική ανάπτυξη (BDD) και τη χρήση φυσικής γλώσσας. Και τα τρία πλαίσια επιδιώκουν τη συνεργασία μεταξύ τεχνικών και μη τεχνικών χρηστών, προσφέροντας υψηλή αναγνωσιμότητα στα σενάρια δοκιμών. Όσον αφορά τις δυνατότητες επεκτασιμότητας, και τα τρία ενσωματώνονται με εργαλεία όπως το Selenium και το Appium για έλεγχο UI, ενώ υποστηρίζουν τη σύνδεση με CI/CD αγωγούς και παραγωγή HTML αναφορών. Το Robot Framework διαθέτει GUI μέσω του RIDE, σε αντίθεση με τα άλλα δύο που χρησιμοποιούνται κυρίως μέσω CLI ή IDEs. Αν και κανένα από τα τρία πλαίσια δεν υποστηρίζει εγγενώς την αυτόματη παραγωγή δοκιμών ή δεδομένων, όλα μπορούν να επεκταθούν με εξωτερικές βιβλιοθήκες για να καλύψουν αυτές τις ανάγκες. Τέλος, η παράλληλη εκτέλεση δοκιμών υποστηρίζεται και από τα τρία, μέσω εργαλείων όπως το Pabot (Robot), Selenium Grid (Cucumber) ή plugins (Gauge), επιτρέποντας την κλιμάκωση των δοκιμών σε μεγαλύτερα έργα.

## 4.13 Σύνοψη και Συμπεράσματα

Έχοντας παρουσιάσει τα γενικά και ειδικά χαρακτηριστικά κάθε προς αξιολόγηση πλαισίου δοκιμής εφαρμογών, μένει να αξιολογήσουμε την ικανότητα καθενός να ικανοποιήσει τα επιμέρους κριτήρια που επιλέξαμε. Ο παρακάτω πίνακας συνοψίζει τα αποτελέσματα της αξιολόγησης αυτής.

*Πίνακας 3: Παρουσίαση των αποτελεσμάτων αξιολόγησης βάσει κριτηρίων σύγκρισης για κάθε πλαίσιο*

| Πλαίσια / Κριτήρια | Junit | TestNG | PyTest | Selenium | Cypress | Playwright | TestCafe | Appium | Cucumber | Gauge | Robot |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Δοκιμές μονάδας (Unit Testing) | Ναι | Ναι | Ναι | Όχι | Εν μέρει | Ναι | Όχι | Όχι | Όχι | Όχι | Όχι |
| Δοκιμές ενσωμάτωσης (Integration Testing) | Ναι | Ναι | Εν μέρει | Εν μέρει | Ναι | Ναι | Εν μέρει | Εν μέρει | Εν μέρει | Εν μέρει | Εν μέρει |
| Δοκιμές συστήματος (System Testing) | Εν μέρει | Εν μέρει | Εν μέρει | Εν μέρει | Ναι | Ναι | Ναι | Ναι | Εν μέρει | Εν μέρει | Εν μέρει |
| Δοκιμές παλινδρόμησης (Regression Testing) | Ναι | Ναι | Ναι | Εν μέρει | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι |
| Δοκιμές διεπαφής χρήστη (UI Testing) | Εν μέρει | Εν μέρει | Όχι | Ναι | Ναι | Ναι | Ναι | Ναι | Εν μέρει | Εν μέρει | Εν μέρει |
| Δοκιμές API | Εν μέρει | Εν μέρει | Εν μέρει | Εν μέρει | Ναι | Ναι | Ναι | Εν μέρει | Εν μέρει | Εν μέρει | Εν μέρει |
| Δοκιμές αποδοχής (Acceptance Testing) | Εν μέρει | Εν μέρει | Όχι | Ναι | Εν μέρει | Εν μέρει | Εν μέρει | Ναι | Ναι | Ναι | Ναι |
| Δοκιμές απόδοσης (Performance Testing) | Εν μέρει | Εν μέρει | Εν μέρει | Όχι | Εν μέρει | Εν μέρει | Όχι | Όχι | Εν μέρει | Εν μέρει | Εν μέρει |
| Ικανότητα δοκιμής εφαρμογών ιστού | Όχι | Όχι | Όχι | Ναι | Ναι | Ναι | Ναι | Όχι | Όχι | Ναι | Ναι |
| Συμβατότητα με φυλλομετρητές | Όχι | Όχι | Όχι | Ναι | Εν μέρει | Ναι | Ναι | Όχι | Όχι | Εν μέρει | Εν μέρει |
| Υποστήριξη παράλληλων δοκιμών | Ναι | Ναι | Ναι | Ναι | Εν μέρει | Ναι | Ναι | Ναι | Όχι | Ναι | Εν μέρει |
| Κόστος χρήσης και ανοικτότητα κώδικα | open-source | open-source | open-source | open-source | open-source | Δωρεάν | open-source | open-source | open-source | open-source | open-source |
| Συχνότητα ενημέρωσης | Συχνή | Συχνή | Συχνή | Συχνή | Συχνή | Συχνή | Μέτρια | Συχνή | Συχνή | Μέτρια | Μέτρια |
| Σκορ / Αστέρια στο GitHub | Μέτριο | Χαμηλό | Μέτριο | Υψηλό | Υψηλό | Υψηλό | Μέτριο | Υψηλό | Μέτριο | Χαμηλό | Μέτριο |
| Συχνότητα Αναφοράς / Υιοθέτησης | Υψηλή | Μέτρια | Υψηλή | Χαμηλή | Μέτρια | Μέτρια | Μέτρια | Μέτρια | Μέτρια | Χαμηλή | Μέτρια |
| Υποστήριξη Μεθοδολογιών Σχεδίασης Δοκιμών | Ναι | Ναι | Ναι | Ναι | Μέσω πρόσθετων | Ναι | Ναι | Μέσω πρόσθετων | Ναι | Ναι | Ναι |
| Υποστήριξη Μοτίβων Σχεδίασης Δοκιμών | Υψηλή | Υψηλή | Υψηλή | Υψηλή | Χαμηλή | Υψηλή | Υψηλή | Υψηλή | Χαμηλή | Μέτρια | Μέτρια |
| Υποστήριξη Γλωσσών Προγραμματισμού | Java | Java | Python | Java, Python, C#, κ.ά. | JavaScript, TypeScript | Java, Python, C#, κ.ά. | JavaScript, TypeScript | Java, Python, C#, κ.ά. | Java, Python, Ruby, κ.ά. | Java, C#, JavaScript, κ.ά. | Java, Python |
| Διεπαφή GUI | Εν μέρει | Όχι | Εν μέρει | Εν μέρει | Ναι | Εν μέρει | Ναι | Εν μέρει | Όχι | Εν μέρει | Εν μέρει |
| Διεπαφή CLI | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι |
| Αυτόματη παραγωγή δεδομένων δοκιμής | Μέσω πρόσθετων | Μέσω πρόσθετων | Ναι | Μέσω πρόσθετων | Ναι | Ναι | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων |
| Αυτόματη παραγωγή δοκιμών | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Ναι | Ναι | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων |
| Ευκολία στη χρήση | Μέτρια | Δύσκολη | Εύκολη | Δύσκολη | Εύκολη | Μέτρια | Εύκολη | Δύσκολη | Μέτρια | Εύκολη | Εύκολη |
| Συμβατότητα με λειτουργικά συστήματα | Πλήρης | Πλήρης | Πλήρης | Πλήρης | Πλήρης | Πλήρης | Πλήρης | Πλήρης | Πλήρης | Πλήρης | Πλήρης |
| Υποστήριξη CI/CD | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Εγγενώς | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Εγγενώς | Εγγενώς |
| Υποστήριξη bug tracking | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Εγγενώς | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων |
| Υποστήριξη mocking/stubbing | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Εγγενώς | Εγγενώς | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων | Μέσω πρόσθετων |
| Δυνατότητα παραγωγής αναφορών | Μέσω πρόσθετων | Πλήρης | Μέσω πρόσθετων | Μέσω πρόσθετων | Πλήρης | Πλήρης | Πλήρης | Μέσω πρόσθετων | Πλήρης | Πλήρης | Πλήρης |

Αρχικά, παρατηρούμε ότι όλα τα πλαίσια που εξετάζονται παρουσιάζουν δύο βασικά κοινά χαρακτηριστικά: την εξαιρετική επεκτασιμότητα και τη συμβατότητα με όλα τα δημοφιλή λειτουργικά συστήματα (Windows, Linux, macOS).

Ωστόσο, η ουσιαστική διάκριση ξεκινά από την ωριμότητα και την υιοθέτηση στην αγορά:

- *Ωριμότητα και Σταθερότητα*: Τα Selenium και JUnit εμφανίζουν, βάσει των κριτηρίων Σκορ / Αστέρια στο GitHub και Συχνότητα Αναφοράς / Υιοθέτησης, την υψηλότερη ιστορική υιοθέτηση και σταθερότητα, γεγονός που εγγυάται πλούσια τεκμηρίωση και ένα μεγάλο, ώριμο οικοσύστημα.
- *Δυναμική και Εξέλιξη*: Τα Playwright και Cypress παρουσιάζουν υψηλή Συχνότητα Ενημέρωσης και ταχεία ανάπτυξη, αντανακλώντας τη στροφή της αγοράς προς σύγχρονες τεχνολογίες με έμφαση στην ταχύτητα και την ευκολία χρήσης.

### 4.13.1 Διαχωρισμός των Πλαισίων σε Ομάδες

Ο διαχωρισμός των πλαισίων σε τρεις διακριτές ομάδες βασίζεται πρωτίστως στο πρωταρχικό επίπεδο εστίασης της δοκιμής και, δευτερευόντως, στη μεθοδολογία σχεδίασης δοκιμών που υποστηρίζουν εγγενώς.

1. Ομάδα 1: Unit Testing Based (JUnit, TestNG, PyTest): Ταξινομούνται εδώ επειδή ο πρωταρχικός τους στόχος είναι οι Δοκιμές μονάδας (Unit Testing) και η Δοκιμές ενσωμάτωσης (Integration Testing). Δεν παρέχουν εγγενώς πλήρη υποστήριξη για Δοκιμές διεπαφής χρήστη (UI Testing) ή εφαρμογών ιστού.
2. Ομάδα 2: UI/System/End-to-End Testing (Selenium, Appium, Cypress, Playwright, TestCafe): Περιλαμβάνει τα πλαίσια που εστιάζουν στην αυτοματοποίηση της συμπεριφοράς του συστήματος από την πλευρά του χρήστη. Ο διαχωρισμός βασίζεται στην Ικανότητα δοκιμής εφαρμογών ιστού και στις Δοκιμές διεπαφής χρήστη (UI Testing).
3. Ομάδα 3: BDD/Acceptance Testing (Cucumber, Gauge, Robot Framework): Ομαδοποιούνται εδώ πλαίσια που υποστηρίζουν εγγενώς τη σύνδεση των επιχειρηματικών απαιτήσεων με τα σενάρια μέσω μιας φυσικής γλώσσας. Ο

διαχωρισμός βασίζεται στην Υποστήριξη Μεθοδολογιών Σχεδίασης Δοκιμών (BDD/Keyword-Driven) και στις Δοκιμές αποδοχής (Acceptance Testing).

### 4.13.2 Ομάδα 1: Πλαίσια Δοκιμών Μονάδας και Ενοποίησης

Εντός της ομάδας των JUnit, TestNG, PyTest, η κύρια διάκριση γίνεται με βάση τη γλώσσα και την πολυπλοκότητα του έργου. Το PyTest αποτελεί την de facto επιλογή για Python. Στα έργα Java, το TestNG υπερτερεί έναντι του JUnit. Παρότι και τα δύο υποστηρίζουν πλήρως τις Δοκιμές μονάδας, το TestNG προσφέρει πλήρη υποστήριξη μοτίβων σχεδίασης δοκιμών και πιο προηγμένα χαρακτηριστικά διαχείρισης δοκιμών (π.χ. παράλληλη εκτέλεση και αναφορές), καθιστώντας το καταλληλότερο για πολύπλοκα σενάρια.

### 4.13.3 Ομάδα 2: Πλαίσια Δοκιμών UI / Συστήματος / End-to-End

Στην ομάδα αυτή, η σύγκριση αφορά την ισορροπία μεταξύ ευελιξίας, ταχύτητας και Υποστήριξης Γλωσσών Προγραμματισμού. Το Selenium προσφέρει τη μέγιστη Συμβατότητα με φυλλομετρητές και την ευρύτερη γλωσσική υποστήριξη, αλλά είναι αρχιτεκτονικά πιο σύνθετο. Το Cypress είναι ταχύτερο και πιο φιλικό προς τον προγραμματιστή, αλλά περιορίζεται κυρίως σε JavaScript/TypeScript. Το Playwright συνδυάζει τα πλεονεκτήματα και των δύο: προσφέρει υψηλή ταχύτητα, πλήρη υποστήριξη φυλλομετρητών (συμπεριλαμβανομένων του WebKit) και ευρεία Υποστήριξη Γλωσσών Προγραμματισμού (όπως Java, Python, .NET), αναδεικνυόμενο ως ο πιο ολοκληρωμένος ανταγωνιστής. Τέλος, το Appium αποτελεί τη μοναδική λύση στη συστάδα για την αυτοματοποίηση δοκιμών σε εγγενείς εφαρμογές κινητών.

### 4.13.4 Ομάδα 3: Πλαίσια BDD / Δοκιμών Αποδοχής

Η ομάδα των Cucumber, Gauge, Robot Framework εστιάζει στην αναγνωσιμότητα και την εφαρμογή μεθοδολογιών. Το Cucumber είναι το πιο διαδεδομένο για το BDD με τη σύνταξη Gherkin. Το Robot Framework ξεχωρίζει λόγω της αρχιτεκτονικής του βασισμένης σε keywords (Keyword-Driven Testing) και είναι

ιδιαίτερα ελκυστικό λόγω της πλήρους υποστήριξης γλωσσών προγραμματισμού (μέσω βιβλιοθηκών). Το Gauge προσφέρει μια σύγχρονη και καθαρή προσέγγιση BDD (based on simple prose). Και τα τρία πλαίσια υποστηρίζουν πλήρως τις Δοκιμές αποδοχής, με την επιλογή να εξαρτάται από την προτίμηση της ομάδας ως προς τη σύνταξη και την ανάγκη για ευρεία γλωσσική υποστήριξη.

### 4.13.5 Καθολικά Συμπεράσματα

Συνοψίζοντας την ανάλυση ανά ομάδα, πρέπει να τονιστεί ότι δεν υπάρχει κάποιος απόλυτος "νικητής" ανά κατηγορία. Η καταλληλότητα ενός πλαισίου εξαρτάται από μια σειρά στρατηγικών παραγόντων, όπως η πολυπλοκότητα του έργου, το διαθέσιμο budget (κόστος), η εμπειρία της ομάδας και οι απαιτήσεις συμβατότητας. Η επιλογή είναι πάντα συνάρτηση των εκάστοτε αναγκών.

Για την καλύτερη κατανόηση, παρακάτω παρατίθενται ενδεικτικές προτάσεις, με βάση τα κύρια κριτήρια κάθε ομάδας:

<u>Για Δοκιμές Μονάδας και Ενοποίησης (Unit Testing):</u>

Σε περιβάλλοντα που εστιάζουν στην ορθότητα της εσωτερικής λογικής του κώδικα, η επιλογή εξαρτάται κυρίως από τη γλώσσα προγραμματισμού.

- *Ενδεικτική Πρόταση (Python):* Για έργα γραμμένα σε Python, το PyTest αποτελεί τη βέλτιστη επιλογή. Προσφέρει καθαρή σύνταξη και μεγάλη ευελιξία, επιτρέποντας στους προγραμματιστές να γράφουν αποτελεσματικά και σύντομα σενάρια δοκιμών.
- *Ενδεικτική Πρόταση (Java/Πολυπλοκότητα):* Για έργα Java υψηλής πολυπλοκότητας, το TestNG είναι πιο κατάλληλο. Παρόλο που το JUnit είναι ευρέως διαδεδομένο, το TestNG προσφέρει ανώτερες δυνατότητες για Υποστήριξη Μοτίβων Σχεδίασης Δοκιμών, διαχείριση παραμέτρων, δημιουργία αναφορών και ομαδοποίηση δοκιμών.

<u>Για Δοκιμές UI / Συστήματος / End-to-End (E2E Testing):</u>

Σε αυτή την ομάδα, η επιλογή καθορίζεται από την ισορροπία μεταξύ ταχύτητας, γλωσσικής υποστήριξης και εμπειρίας προγραμματιστή.

- *Ενδεικτική Πρόταση (Σύγχρονα, Πολυγλωσσικά Projects):* Το Playwright είναι η ενδεδειγμένη επιλογή. Προσφέρει υψηλή ταχύτητα εκτέλεσης και ευρεία Συμβατότητα με φυλλομετρητές (Chromium, Firefox, WebKit), ενώ ταυτόχρονα υποστηρίζει πολλαπλές γλώσσες (Python, Java, C#, JavaScript), γεγονός που μειώνει το κόστος εκμάθησης για ετερογενείς ομάδες.
- *Ενδεικτική Πρόταση (JavaScript/Developer Experience):* Αν το έργο είναι αποκλειστικά σε JavaScript και η προτεραιότητα είναι η βέλτιστη Developer Experience και το γρήγορο feedback, το Cypress είναι προτιμότερο.

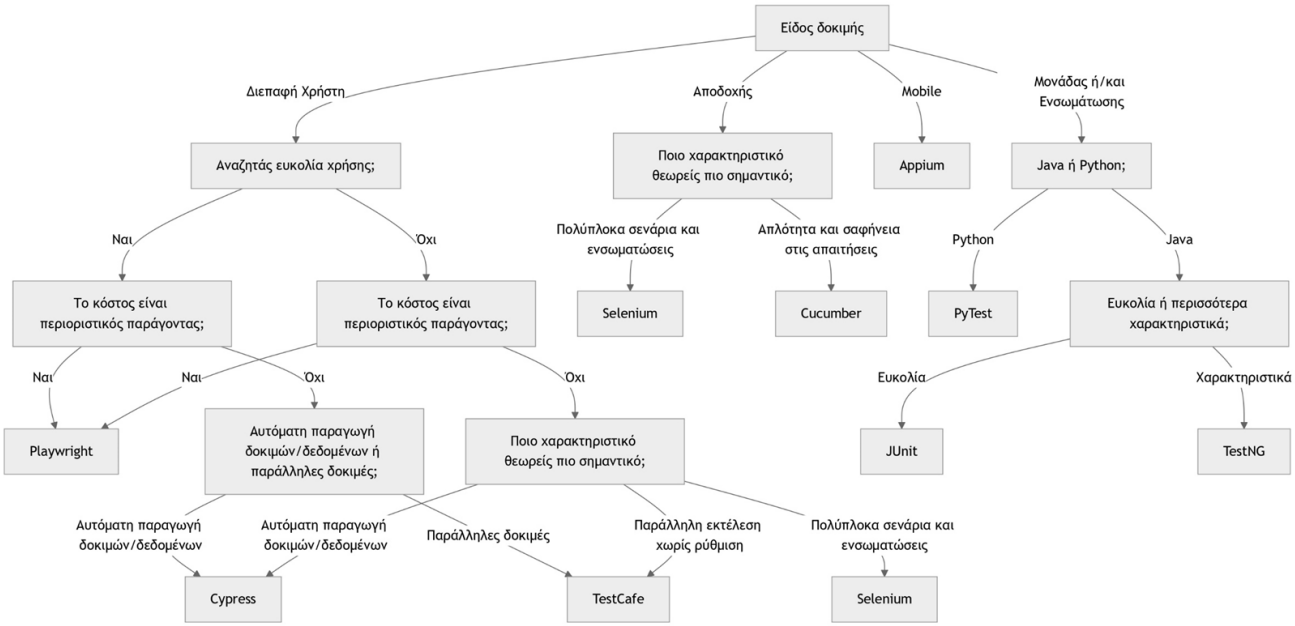
Για Δοκιμές BDD / Αποδοχής (Acceptance Testing):

Σε περιβάλλοντα που δίνουν έμφαση στη συνεργασία και τη σαφήνεια των απαιτήσεων, η επιλογή εξαρτάται από τη μεθοδολογία σύνταξης και την ευελιξία.

- *Ενδεικτική Πρόταση (Keyword-Driven Approach):* Το Robot Framework αποτελεί μια ισχυρή, ευέλικτη επιλογή. Υποστηρίζει την Keyword-Driven Testing μεθοδολογία, η οποία είναι εύκολα κατανοητή από μη τεχνικά μέλη. Επιπλέον, λόγω της πλήρους Υποστήριξης Γλωσσών Προγραμματισμού (μέσω βιβλιοθηκών), μπορεί να ενσωματωθεί άψογα σε διαφορετικά τεχνολογικά stacks.

Οδηγός Επιλογής

Έχοντας υπ’ όψη τα παραπάνω, παρουσιάζεται στη συνέχεια ένας οδηγός απόφασης με τη μορφή διαγράμματος ροής, ο οποίος διευκολύνει την επιλογή του καταλληλότερου πλαισίου δοκιμών. Μέσα από διαδοχικά ερωτήματα, ο χρήστης κατευθύνεται σε μονοπάτια που αντανακλούν τις ανάγκες του έργου και τα τεχνικά του χαρακτηριστικά, με στόχο την κατάληξη σε ένα κατάλληλο πλαίσιο. Το διάγραμμα αυτό αποτελεί ένα συμπαγές εργαλείο υποστήριξης απόφασης, χρήσιμο τόσο σε επαγγελματίες όσο και σε ομάδες ανάπτυξης, καθώς συνοψίζει ευδιάκριτα τη λογική επιλογής, αναδεικνύοντας τα βασικά κριτήρια που επηρεάζουν την επιλογή εργαλείων δοκιμών. Αξίζει να σημειωθεί ότι για τη δημιουργία του διαγράμματος έχουν ληφθεί υπόψη μόνο οι εγγενείς δυνατότητες κάθε πλαισίου, δηλαδή όσες προσφέρονται ενσωματωμένα, και όχι τυχόν λειτουργίες που μπορούν να επιτευχθούν μέσω εξωτερικών εργαλείων ή προσθηκών.




*Εικόνα 3: Διάγραμμα ροής – οδηγός επιλογής κατάλληλων πλαισίων*

# 5. Ερευνητικά Κενά και Μελλοντικές Κατευθύνσεις

Η ανάλυση που πραγματοποιήθηκε στην παρούσα εργασία ανέδειξε τόσο τα πλεονεκτήματα όσο και τους περιορισμούς των πλαισίων δοκιμών που εξετάστηκαν. Παρά τον συστηματικό χαρακτήρα της συγκριτικής αξιολόγησης, παραμένουν ερευνητικά και πρακτικά κενά στον τομέα, τα οποία μπορούν να αποτελέσουν τη βάση για μελλοντική επιστημονική διερεύνηση, εφαρμοστική υλοποίηση ή εμπλουτισμό της υπάρχουσας εργασίας.

## 5.1 Κενό 1: Έλλειψη εμπειρικών δεδομένων από περιβάλλοντα CI/CD

Αν και τα περισσότερα πλαίσια δηλώνουν υποστήριξη για περιβάλλοντα συνεχούς ενσωμάτωσης και παράδοσης, η βιβλιογραφία αποκαλύπτει περιορισμένες εμπειρικές μελέτες που να αξιολογούν την αποτελεσματικότητα και την απόδοσή τους σε πραγματικά pipelines, με πολλαπλούς κύκλους ανάπτυξης, αυτοματοποιήσεις και περιπτώσεις σφαλμάτων.

*Μελλοντική κατεύθυνση:* Η μελέτη της απόδοσης των πλαισίων σε συνθήκες συνεχούς ενσωμάτωσης και παράδοσης (CI/CD). Η αξιολόγηση των εργαλείων υπό πραγματικές συνθήκες παραγωγής (σε pipelines με έντονο ρυθμό ανάπτυξης), θα μπορούσε να αναδείξει πλεονεκτήματα ή/και αδυναμίες που δεν είναι ορατές σε απομονωμένο περιβάλλον.

## 5.2 Κενό 2: Απουσία διαδραστικών εργαλείων υποστήριξης επιλογής

Η εργασία προτείνει ένα διάγραμμα ροής για την υποστήριξη επιλογής πλαισίου, όμως δεν υπάρχουν στοχευμένα ψηφιακά εργαλεία που να αξιοποιούν τέτοια μοντέλα απόφασης και να προσφέρουν δυναμική καθοδήγηση βασισμένη στις ανάγκες του χρήστη.

*Μελλοντική κατεύθυνση:* Η δημιουργία ενός διαδραστικού εργαλείου (π.χ. εφαρμογής ή ιστοσελίδας) που θα επιτρέπει σε χρήστες να επιλέγουν τις απαιτήσεις

του έργου τους και να λαμβάνουν προτάσεις για πλαίσια δοκιμών, βασισμένες στο μοντέλο αξιολόγησης της παρούσας εργασίας. Μία τέτοια εφαρμογή θα μπορούσε να αποτελέσει χρήσιμο βοήθημα για μικρές ομάδες ή startups χωρίς εμπειρία στον χώρο των πλαισίων δοκιμών.

## 5.3 Κενό 3: Περιορισμός στην τεχνική – όχι εμπειρική – αξιολόγηση

Η παρούσα εργασία βασίζεται κυρίως σε τεκμηριωμένα τεχνικά χαρακτηριστικά. Ωστόσο, η υποκειμενική εμπειρία των επαγγελματιών (testers, developers) είναι επίσης κρίσιμη για την πρακτική αποδοχή ενός εργαλείου: δυσκολία εγκατάστασης, ποιότητα τεκμηρίωσης, υποστήριξη κοινότητας, καμπύλη μάθησης κ.ά.

*Μελλοντική κατεύθυνση:* Η ποιοτική αξιολόγηση των πλαισίων, λαμβάνοντας υπόψη την ανατροφοδότηση από επαγγελματίες ελεγκτές (testers) ή προγραμματιστές (developers) που χρησιμοποιούν καθημερινά τα εν λόγω εργαλεία. Μία τέτοια εμπλουτισμένη προσέγγιση θα συνδύαζε αντικειμενικά τεχνικά δεδομένα με εμπειρικές παρατηρήσεις.

## 5.4 Κενό 4: Ανάγκη εμπλουτισμού του μοντέλου αξιολόγησης

Το υπάρχον μοντέλο καλύπτει βασικά και γενικά αποδεκτά κριτήρια. Όμως η ταχύτατη πρόοδος της τεχνολογίας έχει αναδείξει νέες απαιτήσεις:

- Υποστήριξη AI testing: Πρόκειται για τεχνολογίες που επιτρέπουν στα εργαλεία να δημιουργούν ή να προτείνουν αυτόματα σενάρια δοκιμών, να εντοπίζουν περιοχές υψηλού ρίσκου στον κώδικα ή να βελτιστοποιούν την κάλυψη μέσω ανάλυσης δεδομένων. Η αξιοποίηση της τεχνητής νοημοσύνης μπορεί να συμβάλλει στη μείωση του κόστους και του χρόνου συντήρησης των δοκιμών, ιδιαίτερα σε δυναμικά ή μεγάλης κλίμακας έργα.

- Ενσωμάτωση με cloud περιβάλλοντα: Η ενσωμάτωση με cloud περιβάλλοντα αναδεικνύεται ως κρίσιμο χαρακτηριστικό, δεδομένης της αυξανόμενης μετάβασης των οργανισμών σε υποδομές cloud. Η δυνατότητα εκτέλεσης

δοκιμών σε κατανεμημένα περιβάλλοντα, η αποθήκευση και ανάλυση αποτελεσμάτων στο cloud, καθώς και η σύνδεση με cloud-native pipelines καθιστούν ένα πλαίσιο πιο ευέλικτο και κατάλληλο για σύγχρονα, αποκεντρωμένα έργα ανάπτυξης.

*Μελλοντική κατεύθυνση:* Η περαιτέρω αναθεώρηση και ενίσχυση του μοντέλου αξιολόγησης, με τη συμπερίληψη νέων τεχνολογικών κριτηρίων (όπως η υποστήριξη AI testing και ενσωμάτωση με cloud εργαλεία), θα μπορούσε να καταστήσει τον πίνακα ακόμα πιο χρήσιμο για σύγχρονες απαιτήσεις. Με αυτόν τον τρόπο, η εργασία δεν μένει στατική, αλλά αποκτά δυναμικό χαρακτήρα και μετατρέπεται σε βάση συνεχούς ενημέρωσης

## 5.5 Εσωτερικές Επεκτάσεις της Παρούσας Εργασίας

Πέρα από τα ευρύτερα ερευνητικά κενά που εντοπίστηκαν στον τομέα των πλαισίων δοκιμών λογισμικού, υπάρχουν και σημαντικές δυνατότητες εσωτερικής ενίσχυσης της παρούσας μελέτης, οι οποίες σχετίζονται τόσο με την έκταση όσο και με το βάθος της ανάλυσης.

Μία προφανής προέκταση αφορά τη συμπερίληψη επιπλέον πλαισίων δοκιμών, τα οποία, αν και δημοφιλή ή λειτουργικά, δεν περιλήφθηκαν στην αρχική αξιολόγηση λόγω περιορισμού έκτασης ή εστίασης. Παραδείγματα αποτελούν το Serenity BDD (με ιδιαίτερη εστίαση στο behavior-driven development), το Katalon Studio (που προσφέρει low-code/no-code περιβάλλον), και το TestProject (που βασίζεται στο cloud και υποστηρίζει συνεργατική εργασία). Η προσθήκη αυτών των εργαλείων θα εμπλουτίσει το φάσμα της σύγκρισης και θα επιτρέψει την κάλυψη επιπλέον αναγκών και περιπτώσεων χρήσης.

Παράλληλα, κρίνεται ωφέλιμη η επαναξιολόγηση του πίνακα αξιολόγησης με βάση εξειδικευμένες κατηγορίες έργων, όπως είναι οι mobile εφαρμογές, τα embedded συστήματα ή τα cloud-native προϊόντα. Οι απαιτήσεις αυτών των έργων διαφέρουν σημαντικά από τα γενικά έργα λογισμικού, και συνεπώς η απλή χρήση ενός ενιαίου πίνακα ενδέχεται να μην αποτυπώνει επαρκώς τις διαφοροποιήσεις. Μια τέτοια προσαρμογή θα επιτρέψει πιο στοχευμένες και αξιόπιστες συστάσεις, ανάλογα με το τεχνολογικό και λειτουργικό πλαίσιο κάθε έργου.

Επιπροσθέτως, θα μπορούσε να εξεταστεί η εξειδίκευση των διαγραμμάτων απόφασης που παρουσιάζονται στην εργασία, ώστε να ανταποκρίνονται σε συγκεκριμένες τεχνολογίες ή κατηγορίες χρήσης. Για παράδειγμα, ένα εξειδικευμένο διάγραμμα για επιλογή εργαλείων αποκλειστικά για UI testing, ή για projects βασισμένα σε Python, θα μπορούσε να προσφέρει μεγαλύτερη ακρίβεια και χρηστικότητα για τις αντίστοιχες ομάδες χρηστών. Αυτή η προσέγγιση θα ενίσχυε τη χρησιμότητα του εργαλείου υποστήριξης απόφασης που προτείνεται.

Τέλος, ιδιαίτερα σημαντική κρίνεται η μετατροπή της παρούσας εργασίας σε ψηφιακό εργαλείο υποστήριξης απόφασης, είτε υπό μορφή διαδικτυακής εφαρμογής είτε ως ελαφριά mobile λύση. Μέσω αυτής της μετατροπής, ο χρήστης θα μπορεί να εισάγει τις παραμέτρους του έργου του (π.χ. γλώσσα προγραμματισμού, τύπος δοκιμών, απαίτηση για CI/CD, κ.ά.) και να λαμβάνει άμεσα προτάσεις εργαλείων βασισμένες στο μοντέλο αξιολόγησης της μελέτης. Μια τέτοια εφαρμογή θα μετατρέψει τη θεωρητική ανάλυση σε πρακτικό και προσβάσιμο εργαλείο λήψης αποφάσεων, επεκτείνοντας σημαντικά τον αντίκτυπο της εργασίας σε ευρύτερο κοινό πέραν του ακαδημαϊκού χώρου.

# 6. Συμπεράσματα

Η παρούσα εργασία εστιάζει σε ένα κρίσιμο αλλά συχνά υποτιμημένο ζήτημα της ανάπτυξης λογισμικού: την επιλογή του κατάλληλου πλαισίου αυτοματοποιημένων δοκιμών. Μέσα από τη θεωρητική διερεύνηση της σχετικής βιβλιογραφίας, την αποτύπωση των τεχνικών χαρακτηριστικών έντεκα δημοφιλών πλαισίων και τη διαμόρφωση ενός συγκριτικού πίνακα αξιολόγησης, επιδιώχθηκε η κάλυψη ενός υπαρκτού κενού, αυτού της έλλειψης συστηματικής και πρακτικά αξιοποιήσιμης πληροφόρησης για επαγγελματίες και ομάδες ανάπτυξης. Η εργασία ακολούθησε μεθοδολογική προσέγγιση με σαφή και τεκμηριωμένα κριτήρια, και ενισχύθηκε με τη δημιουργία ενός οπτικού εργαλείου υποστήριξης απόφασης. Το τελικό αποτέλεσμα συνδυάζει θεωρητική εγκυρότητα και πρακτική χρηστικότητα.

## 6.1 Συνεισφορά της Εργασίας και Πλεονεκτήματα

Η εργασία συνέβαλε καταλυτικά στη συγκριτική μελέτη έντεκα ευρέως χρησιμοποιούμενων πλαισίων δοκιμών λογισμικού, καλύπτοντας ένα ευρύ φάσμα εργαλείων που ανταποκρίνονται σε διαφορετικές ανάγκες ανάπτυξης, τεχνολογίες και περιβάλλοντα χρήσης. Τα πλαίσια αξιολογήθηκαν με βάση αντικειμενικά, διαφανή και πρακτικά κριτήρια, προσφέροντας έτσι έναν πίνακα αξιολόγησης που μπορεί να αξιοποιηθεί ως σημείο αναφοράς για προγραμματιστές, testers, αρχιτέκτονες λογισμικού και decision-makers. Η συνεισφορά της εργασίας δεν εξαντλείται στην αποτύπωση τεχνικών χαρακτηριστικών, αλλά επεκτείνεται στη μεθοδολογική διαμόρφωση ενός προσαρμοσμένου μοντέλου αξιολόγησης, που μπορεί να επεκταθεί ή τροποποιηθεί σε μελλοντικές εφαρμογές.

Επιπλέον, η εργασία ανέπτυξε έναν οδηγό απόφασης με τη μορφή διαγράμματος ροής, ο οποίος διευκολύνει τη λήψη τεκμηριωμένων αποφάσεων για την επιλογή πλαισίου, βάσει των ειδικών απαιτήσεων κάθε έργου. Μέσα από διαδοχικά ερωτήματα και λογική δόμηση, το διάγραμμα προσφέρει ένα οπτικοποιημένο εργαλείο που μειώνει την αβεβαιότητα και την πολυπλοκότητα της επιλογής.

Συνεπώς, η παρούσα εργασία συνιστά μια πλήρη και ισορροπημένη συμβολή στον χώρο των πλαισίων αυτοματοποιημένων δοκιμών, καθώς ενσωματώνει στοιχεία θεωρητικής ανάλυσης, πρακτικής εφαρμογής και υποστήριξης απόφασης. Μπορεί να λειτουργήσει ως βάση για επαγγελματική χρήση, ως βοηθητικό υλικό για εκπαιδευτικά προγράμματα, αλλά και ως αφετηρία για μελλοντική ερευνητική διερεύνηση ή τεχνολογική επέκταση.

## 6.2 Δεξιότητες και εμπειρία που αποκομίσθηκαν

Κατά την εκπόνηση της εργασίας, αποκτήθηκαν σημαντικές τεχνικές και ερευνητικές δεξιότητες. Αναπτύχθηκε βαθύτερη κατανόηση του τοπίου των πλαισίων δοκιμών λογισμικού και ενισχύθηκε η ικανότητα αναζήτησης, αξιολόγησης και σύνθεσης τεκμηριωμένου υλικού από επίσημες και ανεξάρτητες πηγές. Επίσης, ενισχύθηκαν δεξιότητες τεκμηρίωσης, οργάνωσης και παρουσίασης τεχνικού περιεχομένου, ενώ παράλληλα εξοικειώθηκα με τη χρήση εργαλείων μορφοποίησης, ανάλυσης και αξιολόγησης σε περιβάλλοντα λογισμικού.

Κατά την εκπόνηση της εργασίας, αποκτήθηκαν σημαντικές τεχνικές, μεθοδολογικές και αναλυτικές δεξιότητες, σχετικές τόσο με τον τομέα των δοκιμών λογισμικού όσο και με τη διαδικασία της ερευνητικής τεκμηρίωσης. Εμβάθυνα στην αρχιτεκτονική, τις δυνατότητες και τα μειονεκτήματα διαφορετικών εργαλείων, ενισχύοντας την ικανότητά μου να αξιολογώ τεχνολογικές λύσεις με κριτικό και τεκμηριωμένο τρόπο.

Παράλληλα, αναπτύχθηκαν δεξιότητες στη συλλογή, επιλογή και σύνθεση πληροφοριών από επίσημες και ανεξάρτητες πηγές, καθώς και στην τεχνική συγγραφή και παρουσίαση σύνθετου περιεχομένου με σαφήνεια και δομή. Μέσα από τη διαδικασία αξιολόγησης εργαλείων, καλλιεργήθηκε επίσης η ικανότητα σύγκρισης εναλλακτικών λύσεων βάσει συγκεκριμένων κριτηρίων, ενώ η δημιουργία του πίνακα και του διαγράμματος υποστήριξης απόφασης με βοήθησε να συνδυάσω αναλυτική σκέψη με εργαλειακές δεξιότητες (π.χ. μορφοποίηση, απεικόνιση, συστηματοποίηση πληροφορίας).

Η εμπειρία αυτή ενίσχυσε συνολικά την ικανότητα ολιστικής κατανόησης του κύκλου ζωής των εργαλείων δοκιμών, προσφέροντας εφόδια που μπορούν να αξιοποιηθούν σε μελλοντικά επαγγελματικά ή ακαδημαϊκά περιβάλλοντα.

# Βιβλιογραφία

1. The World Wide Web: Past, present, and future (1996). Tim Berners-Lee.

2. NCSA Mosaic - The first graphical web browser (2021).

3. What is Web 2.0: Design patterns and business models for the next generation of software (2005). Tim O'REILLY.

4. Service Oriented Architecture for E-learning Systems: The Web2.0 Approach (2015).

5. The Art of Software Testing (2012). John Wiley & Sons, Inc.

6. Software Testing: A Craftsman's Approach 4th edition (2014). CRC Press.

7. Introduction to Software Testing (2016). Cambridge University Press.

8. ISTQB Foundation Level Syllabus v3.1.1 (2018). International Software Testing Qualifications Board.

9. A comparative study of automation testing tools for web applications (2021). Elis Pelivani, Betim Cico.

10. Best Test Automation Frameworks (Updated 2023). GH and Akshay Badka.

11. JUnit: A Complete Guide (2022). Shinji Kanai.

12. Pros and Cons of Selenium In Automation Testing: A Comprehensive Assessment (2024). Rohith Ramesh.

13. TestNG Framework Tutorial: A Comprehensive Guide (2023). Sarah Elson.

14. Effective Python Testing With Pytest (2024). Dane Hillard.

15. Selenium Pytest Tutorial: A Comprehensive Guide, with Examples & Best Practices.

16. Cypress Documentation - Cross Browser Testing.

17. Comparison Analysis of Android GUI Testing Frameworks by Using an Experimental Study (2018). Meilianaa, Irwandhi Septian, Ricky Setiawan Alianto, Daniel.

18. Playwright Documentation - https://playwright.dev.

19. TestCafe Documentation - https://testcafe.io/documentation

20. Appium Documentation - https://appium.io/docs/en/2.0/intro/history/

21. Cucumber Documentation - https://cucumber.io/

22. BDD Tool Cucumber is 10 Years Old: Q&A with its Founder Aslak Hellesøy (2018).

23. Taxonomy of Automated Software Testing Tools - Kamran Shaukat, Usman Shaukat, Faran Feroz, Shahraiz Kayani, Ali Akbar (2015)

24. Web App Security A Comparison and Categorization of Testing Frameworks - Satish M. Srinivasan and Raghvinder S. Sangwan (2017)

25. A Study of Automated Software Testing: Automation Tools and Frameworks - Mubarak Albarka Umar, Chen Zhanfang (2019)

26. A comparative study of automation testing tools for web applications - Elis Pelivani, Betim Cico (2021)

27. Test Automation with the Gauge Framework: Experience and Best Practices. In: Gervasi, O., *et al.* Computational Science and Its Applications - Garousi, V., Keleş, A.B., Balaman, Y., Güler, Z.Ö. (2020).

28.Behavior Driven Development using Gauge Framework. International Journal for Research in Applied Science and Engineering Technology - Shamanth, M.. (2022).

29.Generating a Robot Framework Test Script for Web Application Based on Database Constraints - Rattasan Aekaukkharachawakit, Taratip Suwannasart (2024)

30.Usage of Robot Framework in Automation of Functional Test Regression - Stresnjak, Stanislav & Hocenski, Zeljko. (2011)