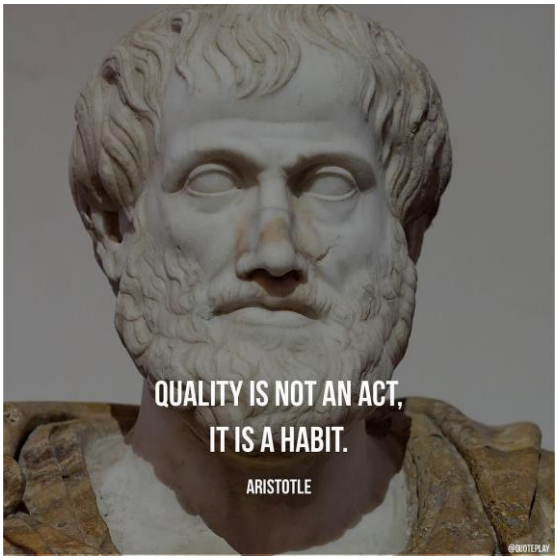
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Υλοποίηση στρατηγικής ελέγχου για τη διασφάλιση ποιότητας εφαρμογών ιστού και συγκριτική μελέτη εργαλείων αυτοματισμού ιστού»



Του φοιτητή
Κωνσταντίνου Χουλιάρα
Αρ. Μητρώου: 144353

Επιβλέπουσα
Ασδρέ Αικατερίνη
Ε.ΔΙ.Π.

Θεσσαλονίκη 07/01/2023

Τίτλος Π.Ε. «Υλοποίηση στρατηγικής ελέγχου για τη διασφάλιση ποιότητας εφαρμογών ιστού και συγκριτική μελέτη εργαλείων αυτοματισμού ιστού»
Κωδικός Π.Ε. 21379
Ονοματεπώνυμο φοιτητή Κωνσταντίνος Χουλιάρας
Ονοματεπώνυμο εισηγητή Αικατερίνη Ασδρέ
Ημερομηνία ανάληψης Π.Ε. 16-10-2021
Ημερομηνία περάτωσης Π.Ε. 10-9-2023

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Σε αυτούς που ήταν εκεί για εμένα…»

Πρόλογος

Τα συστήματα και οι εφαρμογές λογισμικού αποτελούν αναπόσπαστο μέρος της ζωής των επιχειρήσεων αλλά και των καταναλωτών. Είναι βέβαιο, ότι κάθε άνθρωπος στο παρελθόν, είχε τουλάχιστον μία εμπειρία κατά την διάρκεια αλληλεπίδρασής του με λογισμικό που δεν λειτούργησε όπως αναμενόταν. Η υλοποίηση στρατηγικής ελέγχου είναι ένας τρόπος αξιολόγησης και διασφάλισης της ποιότητας του λογισμικού που έχει ως αποτέλεσμα την μείωση του κινδύνου αστοχίας της εφαρμογής που βρίσκεται σε εκτέλεση.

Το αντικείμενο αυτής της πτυχιακής εργασίας είναι η ανάλυση του ευρύτερου τομέα της διασφάλισης ποιότητας του λογισμικού σε θεωρητικό αλλά και σε τεχνικό/πρακτικό επίπεδο καθώς και η σύγκριση πλαισίων (frameworks) και εργαλείων που χρησιμοποιούνται για την διεξαγωγή ελέγχου σε εφαρμογές ιστού με στόχο την περιγραφή των πλεονεκτημάτων και μειονεκτημάτων της κάθε τεχνολογίας.

Η επιλογή αυτού του θέματος προέκυψε κυρίως λόγω του έντονου ενδιαφέροντος στον συγκεκριμένο τομέα και στις διαφορετικές προσεγγίσεις και τεχνολογίες που χρησιμοποιούνται. Οι τεχνολογίες αυτές βρίσκονται σε διαρκή άνθιση και παράλληλα σε υψηλή ζήτηση στην αγορά εργασίας.

Το όφελος που εισέπραξα με την εκπόνηση αυτής της πτυχιακής εργασίας είναι αρκετά σημαντικό για την γενικότερη αντίληψη της διαδικασίας ανάπτυξης λογισμικού από την πλευρά των επιχειρήσεων με στραμμένη την προσοχή στο κομμάτι της διασφάλισης υψηλής ποιότητας των προϊόντων λογισμικού και του ελέγχου αυτών.

Περίληψη

Το αντικείμενο της παρούσας πτυχιακής εργασίας είναι η μελέτη της διασφάλισης της ποιότητας των προϊόντων λογισμικού, η οποία επιτυγχάνεται κυρίως μέσω της διεξαγωγής του ελέγχου. Υπογραμμίζεται ότι οι επιχειρήσεις και οι οργανισμοί πρέπει να λάβουν σοβαρά υπόψιν την ποιότητα των προϊόντων τους, εφόσον επιθυμούν τα προϊόντα τους να είναι ανταγωνιστικά στην αγορά. Τονίζεται ότι η διασφάλιση της ποιότητας λογισμικού και ο έλεγχος, είναι δύο πολύ ευρείς τομείς οι οποίοι περιέχουν αρκετούς ορισμούς, έννοιες, πρακτικές και διαδικασίες οι οποίες πρέπει να εφαρμοστούν σωστά στην πράξη προκειμένου να επιτευχθούν τα επιθυμητά επίπεδα ποιότητας. Στην αρχή μελετώνται οι εφτά βασικές αρχές, οι τρόποι, οι τύποι, τα επίπεδα, οι κατηγορίες του ελέγχου καθώς και ο κύκλος ζωής του ελέγχου λογισμικού και οι επιμέρους δραστηριότητές του. Εφόσον αναλυθεί το θεωρητικό κομμάτι του ελέγχου και της διασφάλισης της ποιότητας, τότε η πτυχιακή εμβαθύνεται μέσω της υλοποίησης του λειτουργικού ελέγχου για τον ιστότοπο του «https://inuportal.ihu.gr», χρησιμοποιώντας τα τρία δημοφιλέστερα εργαλεία αυτόματου ελέγχου (Selenium, Cypress και Playwright) για διαδικτυακές εφαρμογές. Επίσης, γίνεται η συγκριτική ανάλυση των εργαλείων αυτόματου ελέγχου που χρησιμοποιήθηκαν μελετώντας τα χαρακτηριστικά, τα πλεονεκτήματα και τα μειονεκτήματα του κάθε εργαλείου. Επισημαίνεται ότι οι επιχειρήσεις και οι οργανισμοί διεξάγουν έρευνα για την επιλογή του κατάλληλου test automation framework το οποίο καλύπτει τις ανάγκες τους. Ακόμη αναφέρεται ότι τα διάφορα πεδία του τομέα της διασφάλισης της ποιότητας και του ελέγχου λογισμικού ολοένα και αυξάνονται, απαιτώντας συνεχώς περισσότερη τεχνολογική κατάρτιση και μελέτη από τους ενδιαφερομένους. Γίνεται αντιληπτό ότι η ποιότητα των προϊόντων λογισμικού είναι ζωτικής σημασίας για τις επιχειρήσεις και τους οργανισμούς, εφόσον συνδράμει σημαντικά στην επιτυχία των προϊόντων τους. Καταλήγοντας η διασφάλιση της ποιότητας και ο έλεγχος των προϊόντων δεν πρέπει να συμπεριλαμβάνονται στην λίστα περικοπών μιας επιχείρησης ή ενός οργανισμού που επιθυμεί να είναι ανταγωνιστικός και επιτυχημένος στην αγορά.

«Test strategy and software quality assurance using web-based automation testing frameworks — comparison of various tools and frameworks»

Konstantinos Chouliaras

Abstract

The subject of this bachelor's thesis is the study of software quality assurance, which is primarily achieved through testing. It is emphasized that, business and organizations must take the quality of their products seriously if they want their products to be competitive in the market. It is highlighted that software quality assurance and testing are two very broad fields that contain several definitions, concepts, practices, and procedures that must be understood and then properly applied in practice to achieve the desired quality levels. The initial part of this thesis examines the seven fundamental principles, methods, types, levels, categories of testing, as well as the software testing life cycle and its individual activities. Once this theoretical foundation is acquired, the analysis deepens through the implementation of functional testing for the website of "https://uniportal.ihu.gr" using the three most popular automated testing frameworks (Selenium, Cypress, Playwright) for web applications. The comparative analysis of these automated testing frameworks by examining their features, advantages and disadvantages, guide businesses and organizations on selecting the appropriate framework for automating the functional testing for their web applications. Furthermore, a comparative analysis of the automation testing frameworks that have been used is conducted evaluating their features, advantages, and disadvantages. It is highlighted that business and organizations conduct research to select the appropriate test automation framework which is suitable for their needs. This thesis concludes by emphasizing that the quality of software products is crucial for the success of business and organizations. Quality assurance and testing should not be considered as areas for cost cutting, but rather as essential investments for competitiveness and success in the market.

Ευχαριστίες

Οφείλω να εκφράσω τις θερμές μου ευχαριστίες στους καθηγητές μου, οι οποίοι με βοήθησαν να αποκτήσω τις βασικές γνώσεις ώστε να μπορέσω να εκπονήσω την παρούσα εργασία αλλά και κατ' επέκταση να καταξιωθώ επαγγελματικά στον τομέα που έχω ζήλο.

Κατά κύριο λόγο, θα ήθελα να ευχαριστήσω την κυρία Ασδρέ Αικατερίνη που με τον υποστηρικτικό και συμβουλευτικό της χαρακτήρα, κατάφερα να ολοκληρώσω και να τελειοποιήσω την πτυχιακή μου εργασία και να αποκτήσω όλες τις απαραίτητες γνώσεις.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
QAE	Quality Assurance Engineers
DevOps	Development and Operations
SDLC	Software Development Life Cycle
CI/CD	Continuous Integration / Continuous Delivery
UI	User Interface
KPI	Key Performance Indicators
NORAD	North American Aerospace Defense Command
ΗΠΑ	Ηνωμένες Πολιτείες Αμερικής
STLC	Software Testing Life Cycle
TDD	Test Driven Development
BDD	Behavior Driven Development
ISTQB	International Software Testing Qualifications Board
ISCB	International Software Certifications Board
QC	Quality Control
QA	Quality Assurance
SUT	System Under Test
NFT	Non-Functional Testing
MTP	Master Test Plan
CR	Change Request
SAFe	Scaled Agile Framework
IDE	Integrated Development Environment
DBMS	Database Management System
VCS	Version Control System
APIs	Application Programming Interfaces
UAT	User Acceptance Testing
XSS	Cross Site Scripting
NRT	Non-Regression Testing
HTML	Hypertext Markup Language

JDK	Java Development Kit
JRE	Java Runtime Environment
TestNG	Test Next Generation
NPM	Node Package Manager
CLI	Command Line Interface
HTTP	Hypertext Transfer Protocol
MacOS	Mac Operating System
iOS	iPhone Operating System
SPA	Single Page Applications

Κεφάλαιο 1ο: Εισαγωγή

Η στρατηγική ελέγχου και η διασφάλιση της ποιότητας του λογισμικού είναι δύο έννοιες οι οποίες λαμβάνονται σοβαρά υπόψιν από το μέρος των επιχειρήσεων και των οργανισμών προκειμένου να παράγουν υψηλής ποιότητας και ανταγωνιστικά προϊόντα. Αυτές οι δύο έννοιες υλοποιούνται στις επιχειρήσεις και στους οργανισμούς μέσω της διεξαγωγής του ελέγχου και των επιμέρους δραστηριοτήτων και εργασιών του. Ο έλεγχος των προϊόντων λογισμικού αποτελεί ένα ευρύ τομέα ο οποίος εξελίσσεται συνεχώς καθώς οι εφαρμογές που διατίθενται γίνονται ολοένα και πιο σύνθετες και περίπλοκες. Τα frameworks, και τα εργαλεία που διατίθενται για τον έλεγχο συνεχώς αυξάνονται και γίνονται περισσότερο πολύπλοκα, στην προσπάθεια για κάλυψη περισσότερων λειτουργιών και την ανάπτυξη νέων χαρακτηριστικών που ενισχύουν τον έλεγχο.

Στην παρούσα πτυχιακή εργασία στο πρώτο κεφάλαιο, αναλύεται το γιατί η διασφάλισης της ποιότητας του λογισμικού και ο έλεγχος είναι ζωτικής σημασίας για τις επιχειρήσεις. Στο δεύτερο κεφάλαιο δόθηκε σημαντική βαρύτητα σε αρκετές έννοιες και ορισμούς της διασφάλισης της ποιότητας του λογισμικού και του ελέγχου προκειμένου να γίνει αντιληπτός ο κύκλος του ελέγχου λογισμικού. Επίσης αναλύεται εκτενώς το πως διεξάγεται ο έλεγχος στις επιχειρήσεις και στους οργανισμούς. Στην συνέχεια, στο τρίτο κεφάλαιο, υλοποιείται λειτουργικός έλεγχος στη ιστοσελίδα του Διεθνούς Πανεπιστημίου της Ελλάδος «https://uniportal.ihu.gr» με την χρήση των τριών δημοφιλέστερων automation testing frameworks (Selenium, Cypress, Playwright) για εφαρμογές ιστού. Τα παραρτήματα Α και Β περιέχουν το Master Test Plan και το Test Plan για την ιστοσελίδα «https://uniportal.ihu.gr» αντίστοιχα, ενώ στο παράρτημα Γ διατίθεται ο πηγαίος κώδικας και περισσότερες πληροφορίες σχετικά με τη δομή, την υλοποίηση και την εκτέλεση των tests για το κάθε automation testing framework. Έπειτα, στο τέταρτο κεφάλαιο, πραγματοποιείται συγκριτική μελέτη και αναδεικνύονται τα χαρακτηριστικά, τα πλεονεκτήματα και τα μειονεκτήματα του κάθε automation testing framework και αναφέρεται η διαδικασία επιλογής του κατάλληλου εργαλείου. Στο πέμπτο κεφάλαιο αναφέρονται τα συμπεράσματα και οι επεκτάσεις που θα μπορούσαν να μελετηθούν στην παρακάτω πτυχιακή εργασία. Οι κύριοι στόχοι και σκοποί της εργασίας είναι να γίνουν κατανοητά τα εξής αντικείμενα:

- Η σημαντικότητα της διασφάλισης της ποιότητας και του ελέγχου λογισμικού στις επιχειρήσεις και στους οργανισμούς
- Οι πολυάριθμοι ορισμοί και έννοιες της διασφάλισης της ποιότητας και του ελέγχου λογισμικού
- Οι βασικές αρχές του ελέγχου λογισμικού
- Οι κύκλος ζωής του ελέγχου λογισμικού και οι επιμέρους δραστηριότητές του
- Οι διαφορετικοί τομείς, επίπεδα, τύποι και τρόποι του ελέγχου λογισμικού
- Οι διάφορες τεχνικές και κατηγορίες του ελέγχου λογισμικού
- Η διαδικασία υλοποίησης του λειτουργικού ελέγχου για διαδικτυακές εφαρμογές
- Η χρήση των automation testing frameworks που χρησιμοποιήθηκαν
- Τα πλεονεκτήματα και τα μειονεκτήματα των automation testing frameworks που χρησιμοποιήθηκαν
- Τα συμπεράσματα της πτυχιακής εργασίας και οι πιθανές επεκτάσεις

1.1 Ανάγκες και απαιτήσεις της αγοράς στη βιομηχανίας λογισμικού

Με την πάροδο των χρόνων, ολοένα και περισσότερο, παρατηρείται ακμαία ανάπτυξη των επιχειρήσεων που πραγματεύονται με το λογισμικό και όπως αναμένεται αυτή η ανοδική πορεία πρόκειται να επικρατήσει για αρκετά χρόνια στο μέλλον. Οι ανάγκες και οι απαιτήσεις των επιχειρήσεων στον αχανή τομέα του λογισμικού συνεχώς αυξάνονται, γεγονός το οποίο θέτει υψηλό

ανταγωνισμό, ραγδαία ανάπτυξη των διαφορετικών τεχνολογιών και κατ' επέκταση περισσότερων θέσεων εργασίας αλλά και μη προϋφίσταντων τίτλων εργασίας. Όσο αυτές οι τεχνολογίες εξελίσσονται τόσο περισσότερη τεχνολογική κατάρτιση και εξειδίκευση των γνώσεων απαιτείται από την μεριά των εργαζομένων. Οι επιχειρήσεις αυτές, προκειμένου να ολοκληρώσουν έργα λογισμικού για τις ανάγκες τους (π.χ. ολοκλήρωση εφαρμογών για τους πελάτες τους, παράδοση εφαρμογών που τους έχουν ανατεθεί για outsourcing, κλπ.) χρειάζονται εξειδικευμένο προσωπικό με δεξιότητες σε ένα ή περισσότερα τμήματα του τομέα της πληροφορικής. Ένα project λογισμικού, ανεξαιρέτως από την μεθοδολογία που ακολουθείται για την διαχείριση του έργου (Agile, Waterfall, Kanban, κλπ.), πρέπει να «περάσει» από τέσσερα πολύ βασικά στάδια πριν παραδοθεί στον τελικό πελάτη-χρήστη. Το πρώτο στάδιο είναι ή ανάλυση των προδιαγραφών του λογισμικού, το δεύτερο είναι ο σχεδιασμός του, το τρίτο είναι η υλοποίηση και η ανάπτυξή του και το τελευταίο στάδιο είναι ο έλεγχος του λογισμικού πριν την παράδοσή (deployment) του στην αγορά. Το καθένα από αυτά τα τέσσερα στάδια απαιτεί το αντίστοιχο προσωπικό που θα αναλάβει τον κάθε ρόλο και αρμοδιότητα.

1.2 Αναγκαιότητα ελέγχου λογισμικού

Είναι ευρέως γνωστό σε όλους ότι ακόμη και το πιο απλό προϊόν απαιτείται να ελεγχθεί πριν την τελική παράδοση του προκειμένου να διασφαλιστεί η ποιότητα του και η εκπλήρωση των απαιτήσεων και των αναγκών του τελικού πελάτη-χρήστη. Χωρίς τον έλεγχο του προϊόντος δεν γίνεται να υπάρχει η σιγουριά και η επιβεβαίωση ότι εξυπηρετούνται οι ανάγκες του πελάτη-χρήστη το οποίο είναι λογικό ότι μπορεί να επιφέρει τραγικές συνέπειες για την ζωή και την φήμη μιας επιχείρησης αλλά ακόμη και για ανθρώπινες ζωές. Τα συστήματα λογισμικού ολοένα και γίνονται πιο σύνθετα και πολύπλοκα, έχοντας ως τελικό στόχο την αξιοποίηση νέων εφαρμογών και ως αντίκτυπο την εξέλιξη των τεχνολογιών. Όπως γίνεται ευκολά αντιληπτό, εφόσον τα έργα λογισμικού μετατρέπονται συνεχώς σε πιο πολύπλοκα, συνδεμένα και εξαρτημένα από άλλα επίσης πολύπλοκα συστήματα, ο έλεγχος τους καθίσταται αναγκαίος και αναπόφευκτος για τις επιχειρήσεις. Σε τελική ανάλυση, εάν δεν διασφαλίζεται η ποιότητα του προϊόντος λογισμικού διεξάγοντας εκτενείς ελέγχους τότε το προϊόν ίσως ουσιαστικά να μην υφίσταται, διότι είναι πιθανό να αποκλίνει από τις προδιαγραφές του ή/και να μην εκπληρώνει τους στόχους του. Μέσω της διασφάλισης της ποιότητας του λογισμικού επιτυγχάνεται ένα προϊόν το οποίο ελαχιστοποιεί τις πιθανότητες αποτυχίας στην αγορά. Η συνεισφορά του ελέγχου στην επιτυχία του προϊόντος λογισμικού προκύπτει με την συμμετοχή των μηχανικών διασφάλισης της ποιότητας από το αρχικό στάδιο ανάλυσης και σχεδίασης των static works products και του λογισμικού. Οι μηχανικοί διασφάλισης της ποιότητας του λογισμικού QAE (Quality Assurance Engineers) δουλεύουν σε ομάδες μαζί με τους designers, developers, DevOps (Development and Operations), product owners, managers και τους άλλους διάφορους stakeholders έτσι ώστε να επιτευχθεί η ποιότητα του προϊόντος σε όλα τα επίπεδα.

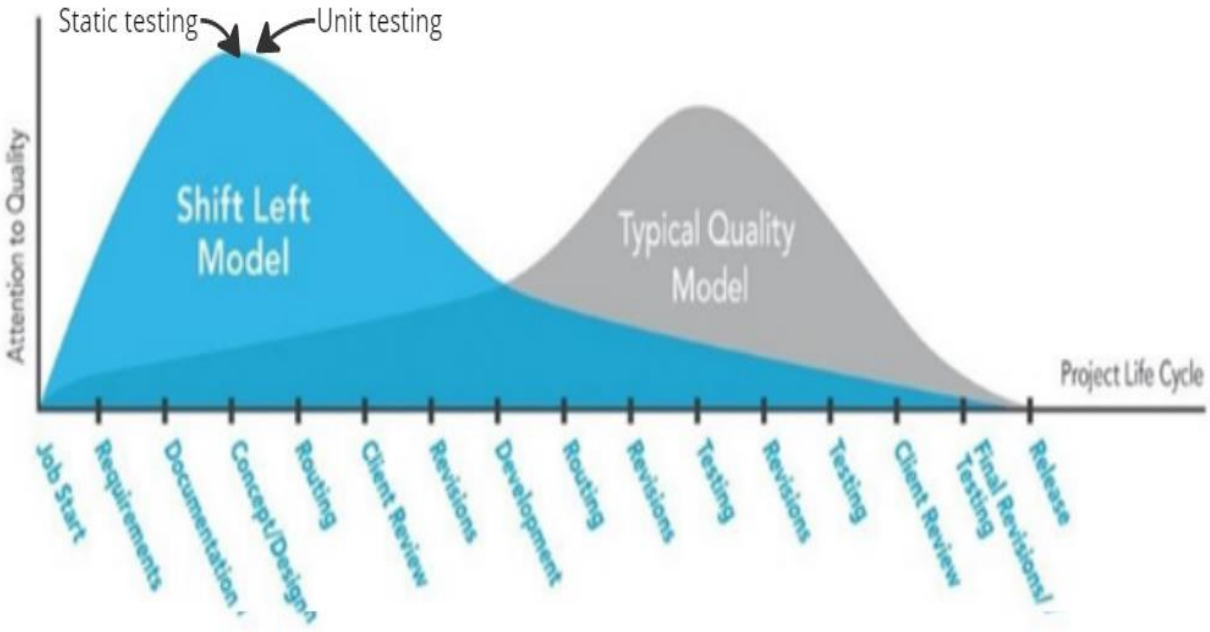
1.2.1 Τα οφέλη και η συμβολή του ελέγχου στην επιτυχία

«Σε όλη την ιστορία της πληροφορικής είναι αρκετά σύνηθες το λογισμικό και τα συστήματα να παραδίδονται σε λειτουργία και λόγω της παρουσίας ελαττωμάτων, να προκαλούνται αστοχίες ή/και να μην καλύπτονται οι ανάγκες των ενδιαφερομένων» [1] . Η διεξαγωγή του ελέγχου λογισμικού μπορεί να μειώσει δραστικά την παρουσία αυτών των προβλημάτων εάν και εφόσον ο έλεγχος διεξάγεται με τον σωστό και βέλτιστο τρόπο για κάθε προϊόν. Μέσω του software testing επιτυγχάνεται ο εντοπισμός σφαλμάτων και στην συνέχεια ακολουθεί από τους developers η διόρθωση τους με στόχο να αποφεύγονται οι δυσλειτουργίες του λογισμικού. Αυτά έχουν ως αποτέλεσμα την παράδοση προϊόντων

με υψηλή ποιότητα, την ικανότητα του προϊόντος να παραμένει ανταγωνιστικό και κερδοφόρο στην αγορά και διαβεβαίωση της ευχαρίστησης και την εμπιστοσύνης του πελάτη-χρήστη. Υπάρχουν αρκετοί τρόποι που το testing συνεισφέρει σημαντικά στην επιτυχία ενός έργου λογισμικού. Ένας από αυτούς είναι η συμμετοχή των μηχανικών διασφάλισης της ποιότητας σε σημαντικές δραστηριότητες του SDLC (Software Development Life Cycle) όπως για παράδειγμα οι εξής:

- Requirements and specification reviews
- User story refinements
- Sprint retrospective

Η συμμετοχή των μηχανικών ελέγχου στις παραπάνω δραστηριότητες – συζητήσεις οδηγεί στον έγκαιρο εντοπισμό σφαλμάτων στον σχεδιασμό των εφαρμογών πριν καν αναπτυχθεί το λογισμικό. Αυτό το γεγονός είναι εξαιρετικά σημαντικό για τις επιχειρήσεις διότι όσο το νωρίτερο έρθουν στην επιφάνεια τα τυχόν σφάλματα τόσο λιγότερο κοστοβόρα και χρονοβόρα θα είναι η επιδιόρθωση τους. Ο εντοπισμός σφαλμάτων σε δεύτερο χρόνο εξισώνεται με την εκθετική αύξηση του κόστους και του χρόνου που απαιτείται σε αντίθεση με το εάν το σφάλμα είχε εντοπιστεί σε προηγούμενο στάδιο της ανάπτυξης λογισμικού. Για παράδειγμα, αν σε ένα έργο λογισμικού εντοπιστεί ένα σφάλμα κατά τη διάρκεια της φάσης του σχεδιασμού του συστήματος, τότε οι αρμόδιοι αρχιτέκτονες του λογισμικού θα πρέπει απλώς να σχεδιάσουν έναν διαφορετικό τρόπο υλοποίησης της συγκεκριμένη λειτουργικότητας που ώστε το σφάλμα να αποφευχθεί. Όμως εάν, αυτό το σφάλμα εντοπίζονταν σε επόμενη φάση, τότε η αρχιτεκτονική του συστήματος θα έπρεπε μερικώς να ξανασχεδιαστεί και στην συνέχεια να υλοποιηθεί και να ελεγχθεί από την αρχή. Επίσης, θα ήταν πιθανό το συγκεκριμένο σφάλμα να επηρέαζε την λειτουργία άλλων components του συστήματος τα οποία με την σειρά τους θα έπρεπε να υποστούν αλλαγές, γεγονός που θα οδηγούσε σε επιπρόσθετη σπατάλη χρόνου, χρημάτων και πόρων. Μια γνωστή τακτική στον τομέα του ελέγχου λογισμικού που ονομάζεται «shift left» ενθαρρύνει τις επιχειρήσεις και τους υπεύθυνους του έργου λογισμικού να επικεντρώνονται στον πρώιμο testing, δηλαδή στον έλεγχο τους προϊόντος πριν καν αυτό αναπτυχθεί, εστιάζοντας στον έλεγχο των απαιτήσεων και των προδιαγραφών. Στο παρακάτω σχήμα αποτυπώνεται η εστίαση της προσοχής στο πρώιμο testing, κατά την διάρκεια του κύκλου ζωής ενός έργου λογισμικού, από την μεριά των developers με την χρήση του unit testing και από την μεριά των quality assurance engineers με την χρήση του static testing τα οποία θα αναλυθούν στην συνέχεια.



Σχήμα 1.1: Shift left

Το unit testing αναφέρεται στην υλοποίηση unit tests από την μεριά των developers έτσι ώστε να ελέγχεται από το μέρος τους η ομαλή λειτουργία της κάθε λογικής μονάδας (component) του κώδικα, ενώ το static testing αναφέρεται στον έλεγχο των προδιαγραφών και των απαιτήσεων του συστήματος λογισμικού από την μεριά των ανθρώπων που είναι υπεύθυνοι για τον έλεγχο.

Μερικά άλλα παραδείγματα της τακτικής του «Shift Left» είναι η συνεργασία των μηχανικών διασφάλισης της ποιότητας με τους σχεδιαστές του συστήματος και τους developers ώστε να γίνει ευκολότερα αντιληπτό και στους τρεις ρόλους το πώς θα λειτουργεί το εν λόγω σύστημα, το πώς μπορεί να αναπτυχθεί και το πως πρέπει να ελεγχθεί κατάλληλα. Η σωστή κατανόηση της λειτουργίας του συστήματος από όλες τις «πλευρές», πριν καν αυτό ξεκινήσει να αναπτύσσεται, μειώνει δραστικά τους κινδύνους για σφάλματα και οδηγεί στην πρώιμη ανάπτυξη ελέγχων που θα πρέπει να επικυρωθούν στην συνέχεια. Ο έλεγχος υποστηρίζει την ομαλή χρήση και λειτουργία των εφαρμογών που αναπτύχθηκαν και επαληθεύει ότι οι ανάγκες των πελατών-χρηστών εκπληρώνονται. Με όλα τα παραπάνω ενισχύονται οι προσπάθειες για την εξασφάλισης της καλής εμπειρίας χρήστη και οτιδήποτε άλλο συνδέεται άμεσα ή έμμεσα με αυτή (π.χ. φήμη της επιχείρησης, κριτικές, εμπιστοσύνη και ευχαρίστηση από τη μεριά του πελάτη, κλπ.), χαρακτηριστικά που είναι εξαιρετικά σημαντικά για τις επιχειρήσεις.

1.2.2 Η συμβολή του ελέγχου στη μέτρηση της ζήτησης στην αγορά

Ο έλεγχος του λογισμικού συμβάλει σημαντικά στην εκτίμηση της ζήτησης στην αγορά μέσω της αξιοποίησης διάφορων release strategies που χρησιμοποιούνται από τις επιχειρήσεις ώστε να μπορέσουν να θέσουν σε αυστηρότερο πλαίσιο τους στόχους τους. Οι παρακάτω δύο τύποι ελέγχου χρησιμοποιούνται για τη συμβολή του ελέγχου στη μέτρηση της ζήτησης στην αγορά:

- Ο σκοπός του A/B testing συνήθως είναι να εκτιμηθεί εάν μια έκδοση λογισμικού αρέσει περισσότερο στους πραγματικούς χρήστες της εφαρμογής σε σύγκριση με μία άλλη έκδοση. Κυρίως χρησιμοποιείται για εκδόσεις οι οποίες έχουν αρκετές διαφορές στο UI (User Interface) ή και στον τρόπο πλοήγησης της εφαρμογής. Μερικοί χρήστες λαμβάνουν διαφορετική έκδοση της εφαρμογής εν αγνοία τους και στην συνέχεια η επιχείρηση αναλύει τα στατιστικά της αλληλεπίδρασης των χρηστών για την κάθε έκδοση καθώς επίσης και πολύ σημαντικά KPI

(Key Performance Indicators) όπως για παράδειγμα ο ενεργός χρόνος χρήσης της εφαρμογής, η αποφυγή επανάληψης περιττών ενεργειών, ο χρόνος που χρειάστηκε για να επιτευχθούν οι κύριες ενέργειες, κλπ. Το ποσοστό των χρηστών που λαμβάνουν διαφορετική έκδοση λογισμικού μπορεί να διαφέρει αναλόγως με τις ανάγκες της επιχείρησης αλλά συνήθως είναι 50% προς 50% για λόγους αναλογίας. Μετά την περάτωση του A/B testing και την ανάλυση των KPI τα οποία συνδέονται άμεσα με την αξία της επιτυχίας της εφαρμογής, η επιχείρηση μπορεί βάσει των δεδομένων που συνέλεξε μέσω του A/B testing να έχει μια ρεαλιστική απάντηση για το ποια έκδοση του λογισμικού είναι πιο ελκυστική και προτιμούν οι χρήστες-πελάτες της.

- Το canary testing σε αντίθεση με το A/B testing είναι περισσότερο εστιασμένο στο πόσο σωστά λειτουργεί ένα καινούριο χαρακτηριστικό μιας εφαρμογής και με το οποίο οι χρήστες της εφαρμογής δεν έχουν ακόμη εξοικειωθεί. Το ποσοστό των χρηστών που θα λάβει την έκδοση της εφαρμογής η οποία ενσωματώνει το καινούριο αυτό χαρακτηριστικό ποτέ δεν είναι μεγάλο διότι σε περίπτωση που το χαρακτηριστικό δεν λειτουργεί σωστά οι πελάτες που χρησιμοποιούν αυτή την έκδοση θα δυσαρεστηθούν και θα έχουν μια κακή εμπειρία χρήσης. Ένα κύριο χαρακτηριστικό και διαφορά του canary testing σε αντίθεση με το A/B testing είναι ότι κατά την διεξαγωγή του canary testing επιλέγονται τυχαία οι χρήστες οι οποίοι θα λάβουν έκδοση λογισμικού που ενσωματώνει το νέο χαρακτηριστικό ενώ στο A/B testing η κατανομή των διαφορετικών εκδόσεων είναι εστιασμένη σε συγκεκριμένους χρήστες βάση συγκεκριμένων κριτηρίων που ορίζονται από την επιχείρηση και από την ομάδα που είναι υπεύθυνη για τον έλεγχο [3].

1.2.3 Επιπτώσεις ελλιπούς ή μηδαμινού ελέγχου λογισμικού

Οι επιπτώσεις του ελλιπούς ή μηδαμινού ελέγχου του λογισμικού μπορούν να γίνουν πολύ εύκολα αντιληπτές από την πρώτη στιγμή που μια εφαρμογή παραδοθεί στους πελάτες-χρήστες της εφόσον το λογισμικό δεν έχει ελεγχθεί διεξοδικά. Ένα προϊόν που έχει ελεγχθεί ελλιπώς ή καθόλου είναι σχεδόν σίγουρο ότι θα παρουσιάσει πολύ σημαντικά προβλήματα στην χρήση του ή ακόμη και στην ολοκληρωτική αποφυγή και εγκατάλειψη της χρήσεως του από τους αρχικά ενδιαφερόμενους. Καθώς το λογισμικό διαδραματίζει ολοένα και σημαντικότερο ρολό στις ζωές των ανθρώπων συνδέεται άμεσα με αυτούς. Οι επιπτώσεις της παράκαμψης του ελέγχου εκτός από τις τραγικές συνέπειες που μπορούν να επιφέρουν στην φήμη, την επιβίωση των επιχειρήσεων και την οικονομία, μπορούν να οδηγήσουν σε απώλεια ανθρώπινων ζωών. Αυτό φυσικά έχει σχέση με την φύση μιας εφαρμογής, αλλά εφόσον η παρουσία του λογισμικού είναι έντονη και πανταχού παρόν στην καθημερινότητα των ανθρώπων δεν χρειάζεται ιδιαίτερη φαντασία για να γίνουν αντιληπτές τέτοιες περιπτώσεις. Η τεχνολογία συνδράμει σημαντικά στον ανθρώπινο παράγοντα καθώς επίσης βοηθά να εκπληρωθούν οι πολύ βασικές ανάγκες του όπως αυτές ορίζονται στην ιεραρχική πυραμίδα των ανθρώπινων αναγκών του Abraham Maslow (υγεία, επιβίωση, αναπαραγωγή, ελευθερία, οικονομία, ανεξαρτησία, προσωπική ασφάλεια, αυτοβελτίωση, επιτυχία, κοινωνικοποίηση, κλπ.). Στο παρακάτω σχήμα παρουσιάζεται η ιεράρχηση των ανθρώπινων αναγκών σύμφωνα με τον Abraham Maslow [4].

Σχήμα 1.2: Η ιεράρχηση των ανθρώπινων αναγκών σύμφωνα με τον Abraham Maslow

Οι περισσότερες από θεμελιώδεις ανάγκες του ανθρώπου πλέον εκπληρώνονται με την βοήθεια λογισμικού και διαφόρων εφαρμογών. Ο έλεγχος του λογισμικού σίγουρα δεν ανήκει στην λίστα με στοιχεία που μια επιχείρηση πρέπει κάνει περικοπές προκειμένου να αποφύγει επιπλέον έξοδα αλλά αντιθέτως προκύπτει ότι ο έλεγχος αποτελεί αναπόσπαστο κομμάτι για κάθε εφαρμογή που αναπτύσσεται [5]. Αξίζει επίσης να αναφερθεί ότι αρκετά συμβόλαια παράδοσης λογισμικού μεταξύ επιχειρήσεων δηλώνουν ρητά την επιβολής χρηματικών ποινών σε περίπτωση που το προϊόν έχει ελλείψεις στην ποιότητα ή σε περίπτωση που η παράδοση καθυστερήσει, δηλαδή δύο προβλημάτων που μπορούν να ελαχιστοποιηθούν με τον έλεγχο.

1.2.4 Αιτίες ατελειών λογισμικού

Οι άνθρωποι είναι επιρρεπείς σε λάθη εκ φύσεως, γεγονός που οδηγεί στην παρουσίαση σφαλμάτων και ελαττωμάτων. Ένα μικρό λάθος στον κώδικα μιας εφαρμογής είναι πολύ πιθανό να κλιμακώσει το πρόβλημα και σε περαιτέρω εφαρμογές και συστήματα το οποία αλληλοεπιδρούν με την συγκεκριμένη εφαρμογή. Στην συνέχεια αναφέρονται οι κύριες αιτίες ελαττωμάτων που έχουν παρατηρηθεί στον τομέα του λογισμικού σύμφωνα με το ISTQB (International Software Qualifications Board).

- Πίεση χρόνου
- Ο άνθρωπος είναι επιρρεπής σε σφάλματα
- Άπειροι ή ανεπαρκώς ειδικευμένοι συμμετέχοντες στο έργο
- Κακή επικοινωνία μεταξύ των συμμετεχόντων στο έργο, συμπεριλαμβανομένης της κακής επικοινωνίας σχετικά με τις απαιτήσεις και το σχεδιασμό
- Η πολυπλοκότητα του σχεδιασμού, της αρχιτεκτονικής και του κώδικα του υποκείμενου προβλήματος που πρέπει να λυθεί ή/και των τεχνολογιών που χρησιμοποιούνται
- Λανθασμένη κατανόηση των ενδοσυστημικών (intra-system) και διασυστημικών (inter-system) διεπαφών, ειδικά όταν τέτοιες αλληλεπιδράσεις εντός και εντός συστήματος είναι μεγάλες σε αριθμό
- Νέες, άγνωστες τεχνολογίες [1].

1.2.5 Ιστορικά παραδείγματα αποτυχίας λογισμικού

Τα περιστατικά αποτυχίας λογισμικού στην καλύτερη περίπτωση επιφέρουν δυσαρέσκεια των χρηστών τους και δυσφήμιση των συγκεκριμένων οργανισμών και επιχειρήσεων που είναι υπεύθυνα για την ανάπτυξη αυτών των εφαρμογών. Για παράδειγμα ένα απλό software lag ή άρνηση εξυπηρέτησης από την πλευρά του server μιας διαδικτυακής εφαρμογής εμπορίου, συνήθως έχει μικρές συνέπειες. Δεν είναι όμως λίγες οι περιπτώσεις που σφάλματα και αποτυχίες λογισμικού οδήγησαν σε τραγικά περιστατικά τα οποία η ανθρωπότητα είναι δύσκολο να αγνοήσει, να προσπεράσει και να ξεχάσει.

«Το 1980, η NORAD (North American Aerospace Defense Command) ανέφερε ότι οι ΗΠΑ δέχονταν πυραυλική επίθεση. Το πρόβλημα προκλήθηκε από ένα ελαττωματικό κύκλωμα, μια πιθανότητα που το λογισμικό αναφοράς δεν είχε λάβει υπόψη.

Το 1983, ένας σοβιετικός δορυφόρος ανέφερε εισερχόμενους αμερικανικούς πυραύλους, αλλά ο υπεύθυνος αξιωματικός αποφάσισε να ακολουθήσει το ένστικτό του νιώθοντας ότι ήταν ψευδής συναγερμός και αποφάσισε να μην λάβει κάποια ενέργεια.

Ευτυχώς αυτές οι ψευδείς αναφορές δεν έγιναν ποτέ πράξη. Εάν είχε εξαπολυθεί μια αντεπίθεση σε κάθε περίπτωση, ένας πλήρης πυρηνικός πόλεμος θα ήταν γεγονός και ο κόσμος θα ήταν ένα πολύ διαφορετικό μέρος σήμερα» [6]. Το πιο πιθανό σενάριο εάν λαμβάνονταν υπόψιν η λανθασμένη έξοδος αυτών των προγραμμάτων και για τις δύο από τις προαναφερθέντες περιπτώσεις ήταν η ανθρωπότητα να οδηγούνταν στον τρίτο παγκόσμιο πόλεμο.

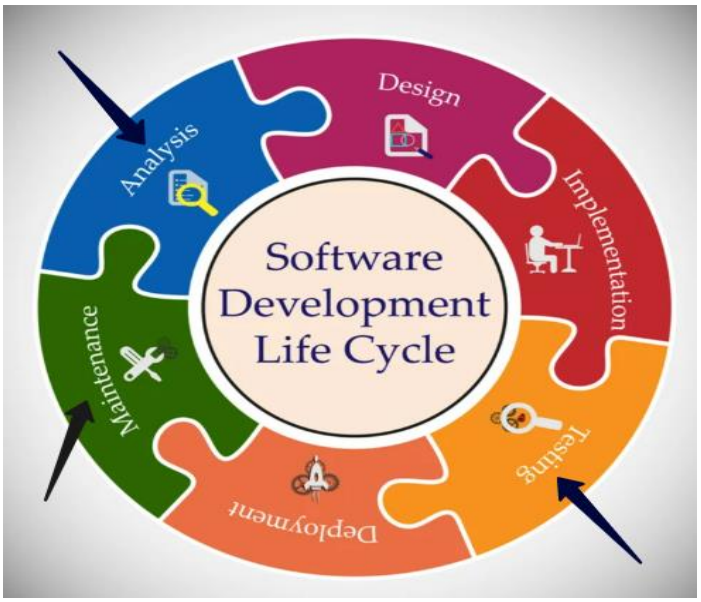
«Το 1994 στη Σκωτία, ένα ελικόπτερο Chinook συνετρίβη με αποτέλεσμα να σκοτωθούν οι 29 επιβαίνοντες. Ενώ αρχικά ο πιλότος κατηγορήθηκε για τη συντριβή, η απόφαση αυτή ανατράπηκε αργότερα, καθώς υπήρχαν στοιχεία ότι η πραγματική αιτία ήταν ένα σφάλμα του συστήματος λογισμικού». [6] Τα περιστατικά αποτυχίας λογισμικού μπορούν να βλάψουν και να καταστρέψουν ανθρώπινες ζωές και επιχειρήσεις και είναι δυνατόν να σχετίζονται μια διάφορες άλλες επιπτώσεις όπως πρόστιμα βάση ρυθμιστικών διατάξεων και νομοθεσιών έως και ποινές φυλάκισης για τους κύριους υπεύθυνους.

«Το 1996, ένας ευρωπαϊκός πύραυλος (Ariane 5) επρόκειτο να παραδώσει ένα ωφέλιμο φορτίο δορυφόρων στην τροχιά της Γης, αλλά προβλήματα με το λογισμικό έκαναν τον πύραυλο εκτόξευσης να απομακρυνθεί από την πορεία του μόλις 37 δευτερόλεπτα μετά την εκτόξευση. Καθώς άρχισε να διαλύεται, αυτοκαταστράφηκε (λόγω των μέτρων ασφαλείας). Το πρόβλημα ήταν αποτέλεσμα επαναχρησιμοποίησης κώδικα από τον προκάτοχο του συστήματος εκτόξευσης, το Ariane 4, το οποίο είχε πολύ διαφορετικές συνθήκες πτήσης από τον Ariane 5. Περισσότερα από 370 εκατομμύρια δολάρια χάθηκαν λόγω αυτού του σφάλματος.»

1.3 Η διασφάλιση της ποιότητας ως αναπόσπαστο κομμάτι του SDLC

Ο ορισμός που έχει επικρατήσει για λέξη ποιότητα είναι ο εξής «Η ποιότητα είναι ένα μέτρο αριστείας ή μια κατάσταση απαλλαγμένης από ελαττώματα, ελλείψεις και σημαντικές παραλλαγές, που προκαλείται από την αυστηρή και συνεπή τήρηση μετρήσιμων και επαληθεύσιμων προτύπων για την επίτευξη ομοιομορφίας της παραγωγής που ικανοποιεί συγκεκριμένες απαιτήσεις πελατών ή χρηστών». [7]

Η διασφάλιση της ποιότητας του λογισμικού επιτυγχάνεται μέσω των ενεργειών που λαμβάνουν μέρος κατά τον κύκλο ζωής ανάπτυξης του λογισμικού. Στο παρακάτω σχήμα παρουσιάζονται όλες οι φάσεις του SDLC (Software Development Life Cycle) και δίνεται εστίαση στις φάσεις τις οποίες εξελίσσεται το κομμάτι του ελέγχου [8].



Σχήμα 1.3: SDLC (Software Development Life Cycle)

Ο έλεγχος ξεκινάει από το στάδιο της ανάλυσης με την διεξαγωγή του static testing και τον έλεγχο των προδιαγραφών, των απαιτήσεων και των σχεδίων (π.χ. flow charts). Οι developers λαμβάνουν τις προδιαγραφές και τις απαιτήσεις επικυρωμένες από τους μηχανικούς διασφάλισης της ποιότητας ώστε να προχωρήσουν στην τεχνική ανάλυση και σχεδιασμό του λογισμικού. Καθώς οι developers προχωρούν στον τεχνικό σχεδιασμό και στην υλοποίηση του λογισμικού, ταυτόχρονα διεξάγουν έλεγχο στον κώδικα που ανέπτυξαν. Παράλληλα οι QAE προετοιμάζουν τα αντίστοιχα βήματα και διαδικασίες όσον αφορά τον έλεγχο, δηλαδή των σχεδιασμό και την ανάπτυξη των Test Cases. Όταν το λογισμικό είναι έτοιμο για να εκτελεστεί τότε τα άτομα του ελέγχου θα προχωρήσουν σε έλεγχο του συστήματος με στόχο να εντοπιστούν τα σφάλματα οποιουδήποτε τύπου και οι αποκλίσεις του προϊόντος από τις προδιαγραφές. Στην φάση του «Testing» στον κύκλο ζωής της ανάπτυξης λογισμικού λαμβάνουν χώρα οι κύριες διαδικασίες του STLC (Software Testing Life Cycle) ο οποίος θα αναλυθεί στην συνέχεια λεπτομερώς. Κατά το στάδιο του testing δίνεται ιδιαίτερη προσοχή, από την μεριά των μηχανικών διασφάλισης της ποιότητας, στην τήρηση όλων των διαδικασιών και την ολοκλήρωση των παραδοτέων που έχουν αποφασισθεί (από την ευρύτερη ομάδα) ότι πρέπει να λάβουν μέρος. Στην συνέχεια, τα άτομα (DevOps) που είναι υπεύθυνα για την παράδοση του λογισμικού στο τελικό χρήστη-πελάτη, φροντίζουν να ολοκληρωθούν οι σχετικές ενέργειες και διαδικασίες παράδοσης του προϊόντος. Αυτές οι ενέργειες και διαδικασίες συχνά ενσωματώνουν τον αυτόματο έλεγχο του λογισμικού μέσω συγκεκριμένων εργαλείων (π.χ. εργαλείων CI/CD Continuous Integration / Continuous Delivery). Τέλος ακολουθεί η φάση της συντήρησης «Maintenance» κατά την οποία διεξάγεται το maintenance testing. Το maintenance testing αναφέρεται στον έλεγχο της διατήρηση της ομαλής λειτουργίας του λογισμικού κατόπιν της παράδοσης του στον πελάτη-χρήστη. Όπως διαπιστώνεται, η διασφάλιση της ποιότητας και ο έλεγχος αποτελούν αναπόσπαστο κομμάτι της ζωής του λογισμικού από το πρώιμο στάδιο της ανάλυσης και σχεδίασης μέχρι το τελικό κομμάτι που είναι ή συνεχής επικύρωση και συντήρηση του ακόμη και μακροπρόθεσμα, σε μεταγενέστερο χρόνο από την παράδοση του στον πελάτη-χρήστη. Ο STLC κυλάει στο παρασκήνιο υπό την σκιά του SDLC ως βασικός παράγοντας και προϋπόθεση για τον ανάλυση, σχεδιασμό, ανάπτυξη, έλεγχο, παράδοση και τελικώς συντήρηση του λογισμικού.

1.4 Τι είναι ο έλεγχος λογισμικού;

Επικρατεί η άποψη πως το software testing συνήθως περιορίζεται στη εκτέλεση δοκιμών(tests) οι οποίες επικυρώνουν ότι το προϊόν λογισμικού δουλεύει σωστά χωρίς σφάλματα και ατέλειες καθώς και όπως απαιτείται να λειτουργεί από την μεριά του πελάτη-χρήστη. Οπότε η συνολική διαδικασία περιγράφεται λανθασμένα με τα εξής βήματα: Δημιουργία των Test Cases, Εκτέλεση των Test Cases και αναφορά των αποτελεσμάτων επικύρωσης.

Στην πραγματικότητα αυτό είναι εν μέρει αλήθεια αλλά περιγράφει μόνο ένα μικρό τμήμα της ευρύτερης διαδικασίας αφού είναι αρκετά πράγματα τα οποία πρέπει να γίνουν αντιληπτά προκειμένου να έχουμε μια καθαρή και συνολική εικόνα και απάντηση για την παραπάνω ερώτηση. Όπως προαναφέρθηκε, υπάρχει ένας μεγάλος κύκλος της ζωής για το Testing, ο οποίος ορίζεται ως STLC, πέρα από τον γνωστό κύκλο ζωής λογισμικού (SDLC). Ο STLC αποτελεί σημαντικό και αναπόσπαστο κομμάτι του SDLC για οποιοδήποτε software project οποιουδήποτε μεγέθους.

Το Testing ξεκινάει με το στατικό testing (static testing) κατά την διάρκεια της συλλογής των documentations, των specifications και οποιουδήποτε άλλου work product (ακόμη και κώδικα) έχει ετοιμαστεί κατά την διάρκεια του κύκλου ζωής του λογισμικού με στόχο να εξεταστούν και να γίνουν αντιληπτές όλες οι πληροφορίες πριν αυτές χρησιμοποιηθούν για την ανάπτυξη του προϊόντος. Αρκετές φορές αυτή εξέταση πραγματοποιείται μέσω συγκεκριμένων ειδών ανασκοπήσεων (reviews) αναλόγως με την φάση που βρίσκεται ένα work product. Τα παραπάνω work products ελέγχονται με σκοπό να εντοπιστούν ασυνέπειες, ασάφειες, αντιφάσεις, ανακρίβειες παραλείψεις και περιττές επαναλήψεις. Για αυτόν τον λόγο τουλάχιστον ένας testing genius είναι παρών από την φάση της συζήτησης των προαπαιτούμενων έτσι ώστε να ανακαλυφθούν το συντομότερο δυνατόν ελαττώματα και τυχόν ελλείψεις στο προϊόν λογισμικού που πρόκειται να αναπτυχθεί. Αυτό βοηθάει στην αποτροπή της εμφάνισης σφαλμάτων στον κώδικα στις επόμενες φάσεις γεγονός το οποίο σχετίζεται άμεσα με την εξοικονόμηση χρόνου, δυναμικού και προϋπολογισμού (budget). Στη συνέχεια, είναι πιθανό να διεξαχθεί στατική ανάλυση (static analysis) σε ορισμένα work products (κυρίως πραγματοποιείται στον κώδικα). Με την στατική ανάλυση ένα work product εξετάζεται και αξιολογείται με την βοήθεια συγκεκριμένων εργαλείων (π.χ. SonarQube, Synopsis Coverity, κλπ.) που έχουν αναπτυχθεί για αυτόν τον σκοπό. Τα εργαλεία αυτά συνήθως ελέγχουν για ατέλειες και πιθανές διορθώσεις στον πηγαίο κώδικα του προϊόντος χωρίς καν να τον εκτελέσουν. Όπως αρχίζει να γίνεται αντιληπτό και θα αναλυθεί στη συνέχεια, το testing δεν περιορίζεται στην δημιουργία, την εκτέλεση και την αναφορά αποτελεσμάτων (Test Progress Report) των test cases, αλλά περιβάλλει πολλές δραστηριότητες, διαφορετικές τεχνικές και συγκεκριμένες διαδικασίες προκειμένου να επιτευχθεί η ενίσχυση και το επιθυμητό επίπεδο της ποιότητας του συστήματος.

1.4.1 Προβλήματα που επιλύει το software testing

Ο έλεγχος λογισμικού αδιαμφισβήτητα προκύπτει ως λύση σε σημαντικά προβλήματα που μπορούν να προκύψουν κατά την διάρκεια της ανάπτυξης ενός έργου λογισμικού. Καθένα από τα παρακάτω προβλήματα είναι ζωτικής σημασίας για τις επιχειρήσεις.

Τα σημαντικότερα από αυτά τα προβλήματα είναι τα εξής:

- Διασφάλιση της ασφάλειας των ανθρώπινων ζωών (για σχετικές εφαρμογές)
- Αύξηση του κόστους και επιβολή προστίμων
- Διάφορες ποινές βάση νομοθεσίας
- Αύξηση του χρόνου για παράδοση (time to market)
- Η κακή φήμη και οι κακές κριτικές

- Κακή εμπειρία χρήστη, παύση χρήσης της εφαρμογής στο μέλλον
- Χαμηλή ποιότητα και επιδόσεις, αργή εκτέλεση των εφαρμογών
- Πολυπλοκότητα συστημάτων
- Αδυναμία επεκτασιμότητας και επαναχρησιμοποίησης του κώδικα
- Αδυναμία αποφυγής ρίσκων
- Εμφάνιση σφαλμάτων, αποκλίσεων των προδιαγραφών ή/και του κώδικα.

1.5 Συναφή επιτεύγματα της επιστήμης και της τεχνολογίας

Σύμφωνα με την βιβλιογραφία ο έλεγχος του λογισμικού χρονολογείται πίσω στις αρχές της δεκαετίας του 1970 αν και διαισθητικά είναι λογικό το testing να διαδραματίστηκε, έστω σε υψηλό επίπεδο, ταυτόχρονα με την εκτέλεση του πρώτου προγράμματος. Ο William Hetzel [9] χρονολογεί το πρώτο συνέδριο που αφιερώθηκε στον έλεγχο λογισμικού το 1972. «Το software testing χαρακτηρίστηκε ως μια "καταστροφική" διαδικασία για την εκτέλεση ενός προγράμματος με τον σκοπό την εύρεση σφαλμάτων που έρχονται σε αντίθεση με την αρχική σχεδίαση του λογισμικού. Από την δεκαετία του 1980 το software testing αντιστοιχίστηκε με την έννοια της μηχανικής λογισμικού και άρχισε να γίνεται αντιληπτό ότι δεν είναι απλώς μια διαδικασία ανακάλυψης σφαλμάτων και ότι το software testing αποτελεί μια πιο ολοκληρωμένη και σύνθετη διαδικασία για την πρόληψη λαθών στον κώδικα»[10]. Ο έλεγχος λογισμικού πλέον χαρακτηρίζεται ως μια ευρεία και συνεχής δραστηριότητα σε όλη την διαδικασία ανάπτυξης και έχει ως στόχο την μέτρηση και την αξιολόγηση των χαρακτηριστικών και των δυνατοτήτων του λογισμικού. Είναι αλήθεια ότι η έντονη έρευνα που πραγματοποιήθηκε με το που πρωτοεμφανίστηκε το software testing ως έννοια είχε ως αποτέλεσμα την "ωρίμανση" διαφόρων τεχνικών και εργαλείων τα οποία βοήθησαν η διαδικασία του σκεπτικού για την σχεδίαση δοκιμών (tests) να γίνει πιο συστηματική και να αποκτήσει πιο συνεργατικό χαρακτήρα όσον αφορά την διαδικασία της ανάπτυξης λογισμικού. Αρκετά μοντέλα της διαδικασίας ελέγχου έχουν προταθεί ως επιλογές για την βιομηχανία λογισμικού όπως το "V model" , και διάφορες έννοιες όπως το Test Driven Development (TDD) και το Behaviour Driven Development (BDD) τα οποία βοηθούν αποτελεσματικά τον έλεγχο και ακολουθούνται από τις περισσότερες επιχειρήσεις [10]. Ο έλεγχος λογισμικού πλέον έχει πάρα πολλούς διαφορετικούς τομείς και ρόλους οι οποίοι συνεχώς εξελίσσονται και απαιτούν περισσότερη τεχνολογική κατάρτιση από την μεριά των ενδιαφερόμενων και των επαγγελματιών του τομέα.

1.5.1 Συνεχής εξέλιξη του automation testing

Ο έλεγχος ιστορικά ξεκίνησε με την διεξαγωγή του manual testing δηλαδή με την απλή επικύρωση της εξόδου ενός προγράμματος μέσω της εκτέλεσης και την επαλήθευσης συγκεκριμένων και ατομικών βημάτων για την κάθε λειτουργία της εφαρμογής. Εάν για κάθε ατομικό βήμα, η έξοδος (actual result) ήταν ίδια με το αναμενόμενο αποτέλεσμα (expected result) χωρίς να υπάρχουν μη λειτουργικά προβλήματα (απόδοση, καθυστέρηση, κλπ.) τότε ο συγκεκριμένος έλεγχος ήταν επιτυχής (test passed) ενώ σε αντίθετη περίπτωση ήταν ανεπιτυχής (test failed). Αυτός ο τρόπος διεξαγωγής του ελέγχου απαιτεί συνεχή ανθρώπινη παρέμβαση για την διεξαγωγή του αποτελέσματος μιας δοκιμής και αρκετό χρόνο. Καθώς η επιστήμη του λογισμικού αναπτύσσεται και τα προγράμματα γίνονταν ολοένα και πιο σύνθετα με περισσότερες λειτουργίες, η έννοια των αυτοματοποιημένων δοκιμών (automation testing) ήρθε στην επιφάνεια. Το automation testing αυτοματοποιεί την παραπάνω διαδικασία και συγκεντρώνει τα αποτελέσματα των tests ώστε οι ενδιαφερόμενοι να έχουν μια καθαρή εικόνα της αξιολόγησης του συστήματος. Προφανώς η διαδικασία του automation testing είναι πολύπλοκη, απαιτητική και ακριβή σε χρόνο και κόστος. Επίσης απαιτεί την συμβολή εξιδεικευμένου προσωπικού. Παρόλα αυτά, το automation testing κυρίως κοστίζει εφάπαξ στις επιχειρήσεις διότι εφόσον αυτοματοποιηθούν τα tests

που ελέγχουν τις λειτουργίες της εφαρμογής, το μόνο πράγμα που κοστίζει μετά την σχεδίαση και ανάπτυξη των tests είναι η εκτέλεση, η συντήρηση τους και η δημοσίευση των αποτελεσμάτων στους ενδιαφερόμενους. Σαφώς, εφόσον οι περισσότερες επιχειρήσεις συνεχώς εξελίσσονται, ο κύκλος του STLC δεν σταματά να κυλάει και έτσι ο σχεδιασμός, η ανάπτυξη, η εκτέλεση και η δημοσίευση των αποτελεσμάτων των tests συνεχίζεται αδιάκοπα. Για τον λόγο αυτό, ένα από τα οράματα του ελέγχου λογισμικού αλλά και των επιχειρήσεων είναι ολοκληρωτική χρήση του automation testing για τον έλεγχο των εφαρμογών, πράγμα που σήμερα περισσότερο μοιάζει αδύνατο λόγω αρκετών αιτίων που αναλύονται στο υποκεφάλαιο 2.12. Υπάρχουν διαφορετικές κατηγορίες και τομείς για το automation testing που καλύπτουν διαφορετικές ανάγκες του ελέγχου εφαρμογών. Οι αναπτυσσόμενες τεχνολογίες τα frameworks, τα διάφορα εργαλεία αλλά και η ανάγκη που υπάρχει στην αγορά για το automation testing το καθιστούν σε υψηλή ζήτηση στις θέσεις αγοράς εργασίας.

1.6 Σημαντικοί παράγοντες στον τομέα διασφάλισης ποιότητας του λογισμικού

Η διασφάλιση της ποιότητας και ο έλεγχος αποτελούν έναν ευρύ τομέα στον οποίο υπάρχουν αρκετοί παράγοντες οι οποίοι διαδραματίζουν πολύ σημαντικό ρόλο. Ιδιαίτερη αναφορά πρέπει να γίνει σε οργανισμούς, πανεπιστήμια, και επιχειρήσεις οι οποίες προσπαθούν να διευρύνουν τις γνώσεις των ενδιαφερομένων και να εξελίξουν τον τομέα του software testing με διάφορους τρόπους.

1.6.1 International Software Testing Qualifications Board (ISTQB)

Ο σημαντικότερος παράγοντας που έχει ξεχωρίσει με διαφορά στον τομέα της διασφάλισης της ποιότητας του λογισμικού ονομάζεται International Software Testing Qualifications Board (ISTQB). Το ISTQB είναι ο παγκόσμιος οργανισμός πιστοποίησης προσόντων για το software testing που ιδρύθηκε στο Εδιμβούργο τον Νοέμβριο του 2002 και είναι ένας μη κερδοσκοπικός σύλλογος νόμιμα εγγεγραμμένος στο Βέλγιο. Επίσης θεωρείται η «βίβλος» από την ευρύτερη κοινότητα των μηχανικών διασφάλισης της ποιότητας λογισμικού και ως η εγκυρότερη και κατευθυντήρια εγκυκλοπαίδεια για αρχάριους αλλά και για εξαιρετικά έμπειρους στον τομέα. Το ISTQB αποτελεί κύρια πηγή αναφοράς της παρούσας πτυχιακής εργασίας για αυτόν τον λόγο υπάρχει εκτενής αναφορά στην εργασία.

Το σχήμα του ISTQB βασίζεται στις κύριες πηγές γνώσεων του Syllabus και του Glossary. Το Syllabus περιλαμβάνει την πηγή των γνώσεων για τους μηχανικούς διασφάλισης της ποιότητας ενώ το Glossary είναι το ευρετήριο για οποιοδήποτε σχετικό ορισμό, ακρωνύμιο ή έννοια. «Το ISTQB είναι το κορυφαίο παγκόσμιο σύστημα πιστοποίησης στον τομέα των δοκιμών λογισμικού. Έως τον Ιούλιο του 2022, η ISTQB έχει διαχειριστεί περισσότερες από 1,2 εκατομμύρια εξετάσεις και έχει εκδώσει περισσότερες από 845.000 πιστοποιήσεις σε περισσότερες από 130 χώρες. Αποτελείται από ένα εκτεταμένο δίκτυο διαπιστευμένων παρόχων εκπαίδευσης, συμβουλίων μελών και παρόχων εξετάσεων. Το ISTQB είναι ένα από τα μεγαλύτερα και πιο καθιερωμένα συστήματα επαγγελματικής πιστοποίησης ενδιαφερομένων στον κόσμο. Η ορολογία του ISTQB είναι παγκοσμίως αναγνωρισμένη από τον κλάδο της πληροφορικής. Αποτελεί την εξ ορισμού γλώσσα στον τομέα του ελέγχου και των δοκιμών λογισμικού (test) λογισμικού και συνδέει επαγγελματίες σε όλο τον κόσμο». [11]

Οι εξετάσεις για τα πιστοποιητικά του ISTQB εφαρμόζεται με συνέπεια σε οποιαδήποτε επιλέξιμη χώρα και η ύλη των εξετάσεων είναι διαθέσιμη δωρεάν σε αρκετές γλώσσες. Είναι σημαντικό να αναφερθεί ότι δεν υπάρχει κάποιο προαπαιτούμενο για τον ενδιαφερόμενο που θέλει να συμμετάσχει στις εξετάσεις του αρχικού επιπέδου το οποίο ονομάζεται «foundation level». Εάν ο ενδιαφερόμενος επιθυμεί να συμμετάσχει σε οποιοδήποτε ανώτερο επίπεδο, η κατάρτιση του foundation level είναι υποχρεωτική.

Το κόστος για τη συμμετοχή στις εξετάσεις για το foundation level είναι σχετικά μικρό (περίπου 180 ευρώ στην Ελλάδα). Επίσης μερικές επιχειρήσεις δίνουν έκπτωση στους εργαζόμενους τους για να συμμετέχουν στις εξετάσεις του ISTQB διότι οι επιχειρήσεις αυτές είναι ISTQB partners. Η εξετάσεις διεξάγονται είτε σε συγκεκριμένα εξεταστικά κέντρα διά ζώσης είτε απομακρυσμένα με την χρήση των κατάλληλων εργαλείων εξέτασης. Η εγκυρότητα των περισσότερων ISTQB certifications είναι εφ' όρου ζωής και δεν χρειάζεται καμία ανανέωση στο μέλλον.

Ο τύπος των εξετάσεων για το foundation level είναι ερωτήσεις πολλαπλής επιλογής (40) στις οποίες μόνο μια εκ των τεσσάρων είναι η σωστή απάντηση. Η διάρκεια της εξετάσεις είναι 60 (+15 λεπτά αν η εξέταση δεν είναι στην μητρική γλώσσα του εξεταζόμενου) οπότε ο εξεταζόμενος έχει περίπου ενάμιση λεπτό για να απαντήσει σωστά την κάθε ερώτηση.

Η συχνότητα που οι εξετάσεις λαμβάνουν χώρα εξαρτάται από το πλήθος των ενδιαφερομένων που εγγράφονται στις εξετάσεις. Η βάση επιτυχίας των εξετάσεων του foundation level είναι 65%. Ο εξεταζόμενος πρέπει να απαντήσει σωστά σε τουλάχιστον 26 σωστές ερωτήσεις για να επιτύχει στις εξετάσεις και να λάβει το πιστοποιητικό. Επίσης στον ιστότοπο του ISTQB υπάρχουν προσομοιώσεις εξετάσεων για εξάσκηση.

Η αποστολή του ISTQB περιγράφεται μερικώς στην παρακάτω λίστα:

- «Προώθηση της αξίας του ελέγχου λογισμικού ως επαγγέλματος σε άτομα και οργανισμούς.
- Βοήθεια στους μηχανικούς διασφάλισης της ποιότητας να είναι πιο αποδοτικοί και αποτελεσματικοί στην εργασία τους, μέσω της πιστοποίησης των ικανοτήτων.
- Δίνει την δυνατότητα στους μηχανικούς ελέγχου λογισμικού να προχωρήσουν στη σταδιοδρομία τους μέσω ενός επαγγελματικού κώδικα δεοντολογίας και ενός μονοπατιού πιστοποίησης πολλαπλών επιπέδων που τους παρέχει τις δεξιότητες και τις γνώσεις που χρειάζονται για να εκπληρώσουν τις αυξανόμενες ευθύνες τους και να επιτύχουν τον επιθυμητό επαγγελματισμό.
- Συνεχής εξέλιξη των κύριων πηγών γνώσεων αξιοποιώντας τις βέλτιστες διαθέσιμες πρακτικές του κλάδου και την πιο καινοτόμο έρευνα, παρέχοντας αυτή τη γνώση δωρεάν διαθέσιμη σε όλους.
- Ορισμός των κριτηρίων για τη διαπίστευση παροχών εκπαίδευσης, για να διασφαλιστεί η συνεπής παράδοση των κύριων πηγών γνώσεων παγκοσμίως» [12].

1.6.2 Διεύρυνση των γνώσεων στον έλεγχο του λογισμικού

Η διεύρυνση των γνώσεων στον έλεγχο λογισμικού μπορεί να ενισχυθεί με την απόκτηση σχετικών μεταπτυχιακών ή και πιστοποιητικών. Τα τελευταία χρόνια είναι αρκετά τα πανεπιστήμια που έχουν συνειδητοποιήσει την ευρύτητα του software testing, της αυτοματοποίησης και το αυξανόμενο ενδιαφέρον που υπάρχει για τον συγκεκριμένο τομέα. Ένα από τα πανεπιστήμια που οι ενδιαφερόμενοι μπορούν να αποκτήσουν μεταπτυχιακό τίτλο στον έλεγχο λογισμικού με διάρκεια φοίτησης 2 έτη είναι το πανεπιστήμιο του Radboud στην Ολλανδία [13]. Επίσης παρατηρείται ότι αρκετά πανεπιστήμια διαθέτουν μεταπτυχιακά προγράμματα σχετικά με τον ευρύτερο τομέα της αυτοματοποίησης ή οποία αποτελεί καινοτόμο και κυρίαρχο ρόλο στον τομέα του ελέγχου λογισμικού.

Εκτός από τα μεταπτυχιακά, οι ενδιαφερόμενοι έχουν μια πληθώρα επιλογών για την διεύρυνση των γνώσεων τους με την απόκτηση πιστοποιητικών και εκπαιδευτικών προγραμμάτων που στοχεύουν εστιασμένα σε συγκεκριμένα είδη / τύπους ελέγχου και σε συγκεκριμένες τεχνολογίες, εργαλεία και είδη εφαρμογών. Οι δύο παραπάνω δραστηριότητες λαμβάνουν μέρος είτε μέσω εξετάσεων είτε μέσω της ολοκλήρωσης παρακολούθησης εκπαιδευτικών προγραμμάτων από διάφορους οργανισμούς ή/και επιχειρήσεις που ασχολούνται με την κατάρτιση εξειδίκευσης σε ειδικές τεχνολογίες και εργαλεία. Τα

πιστοποιητικά αυτά μπορούν να αποτελέσουν την κύρια πηγή τεχνολογικής κατάρτισης και έτσι να διαδραματίσουν τον κυρίαρχο ρόλο για το γνωστικό υπόβαθρο για τον ενδιαφερόμενο στον εύρη τομέα του ελέγχου λογισμικού. Αξίζει να σημειωθεί ότι αυτά τα πιστοποιητικά και τα προγράμματα εκμάθησης είναι εξαιρετικά χρήσιμα και λαμβάνονται σοβαρά υπόψιν για την εύρεση εργασίας στον τομέα του ελέγχου λογισμικού. Όσοι μηχανικοί διασφάλισης της ποιότητας λογισμικού κατέχουν τέτοια πιστοποιητικά έχουν ιδιαίτερη ζήτηση στην αγορά εργασίας. Οι πιο γνωστοί από αυτούς τους οργανισμούς και επιχειρήσεις που πραγματεύονται με σχετικά πιστοποιητικά και εκπαιδευτικά προγράμματα είναι οι ακόλουθοι:

- ISTQB
- International Software Certifications Board (ISCB) [14]
- Ministry of Testing
- Art of Testing
- Edureka
- Udemy
- Coursera
- Chroma Campus [15]
- Codeacademy
- Skillshare
- Hitek computer school
- Linkedin Learning

Εκτός από αυτούς τους οργανισμούς και τις επιχειρήσεις, διάφορα πιστοποιητικά στον τομέα του ελέγχου λογισμικού και οι εξετάσεις τους διεξάγονται από πανεπιστήμια. Στην συνέχεια ακολουθεί μια λίστα μερικά πανεπιστήμια όπου οι ενδιαφερόμενοι μπορούν να αποκτήσουν τέτοια πιστοποιητικά.

- London Academy of IT — «Software Testing Course for beginners»
- London School of Informatics — «BSC/ISTQB Software Testing Foundation»
- The University of Oxford — «STE Software Testing Course» [16]

1.7 Επίλογος

Το πρώτο κεφάλαιο αποτελεί μια εισαγωγή και μια σύντομη περιγραφή των αναγκών και των απαιτήσεων της αγοράς στην βιομηχανία του λογισμικού. Επίσης αναφέρεται ότι ένα μεγάλο μέρος της διασφάλισης της ποιότητας πραγματοποιείται μέσω του ελέγχου του λογισμικού, ο οποίος έλεγχος επιτυγχάνεται κυρίως μέσω της εκτέλεση δοκιμών (tests). Στην συνέχεια διαπιστώνεται, μέσω επιχειρημάτων, ότι ο έλεγχος του λογισμικού είναι ζωτικής σημασίας και ότι προκύπτει ως ανάγκη για τις επιχειρήσεις και τους οργανισμούς καθώς συμβάλει στην επιτυχία των προϊόντων. Έπειτα αναφέρονται κάποιες περιπτώσεις ελλιπούς ή μηδαμινού ελέγχου, οι αιτίες ελαττωμάτων και κάποια ιστορικά γεγονότα που συνέβησαν λόγω αποτυχίας του λογισμικού. Ακόμη εξηγούνται οι λόγοι για τους οποίους η διασφάλιση της ποιότητας αποτελεί αναπόσπαστο κομμάτι του κύκλου ανάπτυξης του λογισμικού, τι είναι το testing και τα προβλήματα που αυτό επιλύει. Επιπλέον, αναφέρονται γενικά μερικά συναφή επιτεύγματα και διαπιστώσεις της επιστήμης και της τεχνολογίας όσον αφορά την διασφάλιση ποιότητας και του ελέγχου λογισμικού. Τέλος παρουσιάζονται οι σημαντικότεροι παράγοντες στον τομέα της διασφάλισης του λογισμικού και οι τρόποι διερεύνησης των γνώσεων στο πεδίο του ελέγχου λογισμικού. Η γνώση των παραπάνω εννοιών και γνώσεων έχουν ως στόχο μια γρήγορη εισαγωγή στο ευρύτερο τομέα της διασφάλισης ποιότητας του λογισμικού και του ελέγχου καθώς και για τα κεφάλαια που ακολουθούν.

Κεφάλαιο 2ο: Διασφάλιση ποιότητας του λογισμικού και έλεγχος

Η διασφάλιση της ποιότητας του λογισμικού και ο έλεγχος λογισμικού, αποτελούν δύο ευρείες τομείς οι οποίες περιέχουν αρκετούς ορισμούς και έννοιες. Αυτοί οι ορισμοί και οι έννοιες πρέπει να γίνουν αντιληπτές έτσι ώστε να αποκτηθεί το θεωρητικό υπόβαθρο έτσι ώστε να γίνει αντιληπτός ο κύκλος ζωής του ελέγχου λογισμικού (και των επιμέρους δραστηριοτήτων και εργασιών του) η οποία υλοποιείται στις επιχειρήσεις και στους οργανισμούς. Μερικές από αυτές τις σημαντικές έννοιες και ορισμούς είναι οι τρόποι, οι κατηγορίες, οι τεχνικές, τα επίπεδα, οι τύποι, οι υποτύποι, οι μετρικές, η στρατηγική και η προσέγγιση του ελέγχου.

2.1 Η ποιότητα ως κύριος στόχος

Η ποιότητα πρέπει να αποτελεί ευθύνη όλων των εργαζομένων σε μια επιχείρηση ανεξαρτήτως από τον ρόλο τους έτσι ώστε το προϊόν να πετύχει υψηλό βαθμό ποιότητας. Οι QAE πέρα από τις διαδικασίες ελέγχου που αναλαμβάνουν, προτείνουν τρόπους βελτίωσης της ποιότητα του λογισμικού στους συνεργάτες τους [17]. Αυτές οι προτάσεις αναλύονται από το scrum team και στην συνέχεια συζητούνται μεταξύ των υψηλών στελεχών της ποιότητας (Quality assurance managers, Quality assurance officers, κλπ.), των ατόμων που κατέχουν το προϊόν (product owners και business owners) και των αρχιτεκτόνων λογισμικού (tech architects) ώστε να αποφασιστεί αν τελικώς θα ληφθούν υπόψιν. Ο ρόλος και οι ευθύνες των ατόμων που είναι υπεύθυνοι για την διασφάλιση ποιότητας του λογισμικού είναι συγκεκριμένες και ποικίλουν.

2.2 Οι στόχοι του ελέγχου λογισμικού

Παρακάτω αναφέρεται το με τι πραγματεύεται ο έλεγχος λογισμικού και οι στόχοι του:

- Προσδιορισμός και ανακάλυψη όσο το δυνατόν νωρίτερα και όσο το δυνατών περισσότερων ατελειών πριν την παράδοση του λογισμικού στον πελάτη.
- Απόκτηση αυτοπεποίθησης για την ποιότητα. Αν η ομάδα που διεξάγει το testing σε ένα προϊόν λογισμικού δεν νιώθει αυτοπεποίθηση για το προϊόν (confidence) και κρίνει ότι το λογισμικό δεν μπορεί να παραδοθεί στον πελάτη, τότε το προϊόν δεν πρέπει να προχωρήσει στο επόμενο στάδιο εκτός και αν οι business owners ή οι product owners αποδεχτούν το συγκεκριμένο ρίσκο που έχει αναδειχθεί μέσω του ελέγχου. Η απόφαση για το «μπλοκάρισμα» της μετάβασης του προϊόντος στο επόμενο στάδιο μπορεί να συμβεί εάν το άτομο ή η ομάδα που κάνει το testing δεν έχει πετύχει ένα συγκεκριμένο ποσοστό ελέγχων (test coverage) για λειτουργίες του λογισμικού οι οποίες θεωρούνται κύριες (εμφάνιση ρίσκου) ή αν δεν έχουν διορθωθεί ατέλειες οι οποίες θεωρούνται κρίσιμες. Ένας quality assurance engineer πρέπει να έχει (και συνήθως έχει) τον τελευταίο λόγο για το αν ένα προϊόν πρέπει να προχωρήσει στο επόμενο στάδιο και/ή να παραδοθεί στον πελάτη καθώς αυτός είναι κατ' επέκταση ο κύριος σκοπός του ρόλου (ο έλεγχος της ροής του λογισμικού) που διαδραματίζει σε ένα scrum team και κατ' επέκταση σε μια επιχείρηση. Εάν δεν τηρείται αυτός ο θεμελιώδης κανόνας τότε το πιθανότερο είναι το προϊόν να αποτύχει μερικώς ή ολικώς στην αγορά γεγονός το οποίο επιφέρει τεράστιες επιπτώσεις στην οικονομία, στην φήμη της επιχείρησης ή ακόμη χειρότερα και στην απώλεια ανθρώπινων ζωών εάν το σφάλμα ή η έλλειψη του προϊόντος έχει άμεση επαφή με τον άνθρωπο (π.χ. λογισμικό απενεργοποίησης του αυτόματου πιλότου σε επιβατικό αεροσκάφος). Αυτή η άδεια, η οποία είναι γνωστή με τον όρο «sign off», δίνεται επισήμως εφόσον ο QAE έχει εκτιμήσει έπειτα από την εφαρμογή διάφορων διαδικασιών ελέγχου και διεξαγωγής δοκιμών ελέγχου ότι ένα συγκεκριμένο τμήμα του προϊόντος είναι έτοιμο για να προχωρήσει στο επόμενο στάδιο ή ακόμη και στην τελική παράδοση του στον πελάτη - χρήστη της εφαρμογής, χωρίς αυτή η καινούρια ανάπτυξη να επηρεάζει την ήδη υπάρχουσα και ομαλή λειτουργία του λογισμικού.

- Παροχή χρήσιμων πληροφοριών στους managers, stakeholders και στους product owners ώστε να πάρουν αποφάσεις σχετικές με την παράδοση. Οι QAE συνεισφέρουν σε αυτή την απόφαση παρέχοντας πληροφορίες για το ποιες λειτουργίες του λογισμικού έχουν ελεγχθεί μέχρι μια συγκεκριμένη χρονική στιγμή, τι ακόμη πρέπει να ελεγχθεί και ποια είναι τα πιθανά ρίσκα.
- Πρόληψη ατελειών. Οι QAE δίνουν πληροφορίες σχετικά με την επιθεώρηση των work products (reviews).
- Αξιολόγηση των static work products - Είναι ο τρόπος που διεξάγεται το static testing προκειμένου να βρεθούν ατέλειες πριν ξεκινήσει η διαδικασία ανάπτυξης του κώδικα.
- Επαλήθευση της εκπλήρωσης των απαιτήσεων. Σε τελική ανάλυση αυτό που ενδιαφέρει τον πελάτη είναι να έχει ένα προϊόν το οποίο οι χρήστες του μπορούν να το καλύπτει όλες τις ανάγκες τους και που μπορούν να το χρησιμοποιήσουν απρόσκοπτα χωρίς αμφιβολίες.
- Μείωση του επίπεδου ρίσκου - προσπάθεια για μετρίαση και εξάλειψη οποιουδήποτε ρίσκου πριν το λογισμικό φτάσει στην αγορά.
- Συμμόρφωση με συμβατικές, νομικές ή ρυθμιστικές απαιτήσεις και πρότυπα. Για παράδειγμα σε ένα safety critical system πρέπει να ληφθούν υπόψη οι ρυθμιστικές απαιτήσεις.

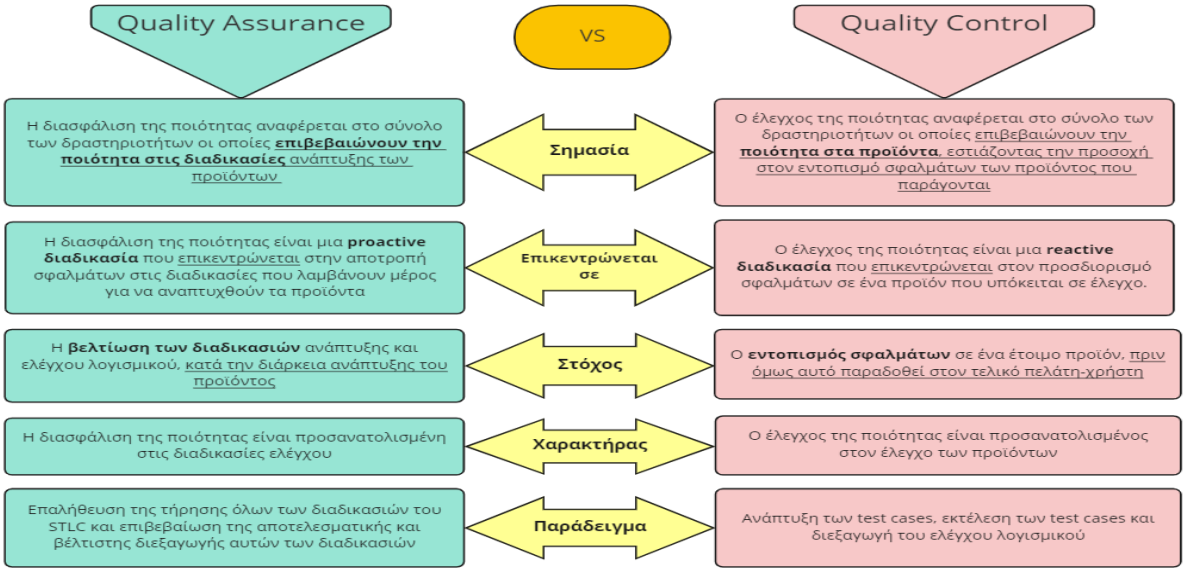
Έτσι λοιπόν, γίνεται αντιληπτό ότι ο έλεγχος λογισμικού δεν έχει να κάνει μόνο με τον έλεγχο των λειτουργικών απαιτήσεων μια εφαρμογής αλλά ότι σχετίζεται με πολλά διαφορετικά πράγματα σε μια επιχείρηση.

2.3 Quality Assurance vs Quality Control

Ο έλεγχος της ποιότητας Quality Control (QC) και η διασφάλιση της ποιότητας Quality Assurance (QA) συχνά είναι δύο έννοιες οι οποίες μπερδεύονται από το κοινό. Στην πραγματικότητα η ευρύτερη έννοια της διοίκησης της ποιότητας (Quality Management) αποτελεί μια «ομπρέλα» που καλύπτει το QA και το QC.

Ο έλεγχος της ποιότητας (QC) περιλαμβάνει διάφορες δραστηριότητες, συμπεριλαμβανομένων δοκιμαστικών δραστηριοτήτων που υποστηρίζουν την επίτευξη των κατάλληλων επιπέδων ποιότητας. Οι δραστηριότητες του ελέγχου αποτελούν μέρος της συνολικής διαδικασίας ανάπτυξης ή συντήρησης λογισμικού.

Με τον όρο QA ορίζονται οι διαδικασίες και τα βήματα τα οποία πρέπει να ακολουθήσει η ομάδα ώστε να επιτευχθεί η επιθυμητή ποιότητα στο προϊόν λογισμικού ενώ το QC περιλαμβάνει διάφορες δραστηριότητες (test case creation, test case execution, defect logging, κλπ.). Επίσης το QA διαξαγάγεται σε νωρίτερο χρόνο από το QC. Το QA αποτελεί υπερσύνολο του QC. Η διασφάλιση της ποιότητας λογισμικού ορίζεται από το ISTQB ως εξής «Οι δραστηριότητες που επικεντρώνονται στην απόκτηση αυτοπεποίθησης σχετική με την εκπλήρωση των απαιτήσεων της ποιότητας» [1]. Η διασφάλιση της ποιότητας του λογισμικού περιγράφεται ως το στάδιο της επιτυχούς ανάπτυξης των προϊόντων λογισμικού. Στο παρακάτω σχήμα μπορούν να γίνουν εύκολα αντιληπτές οι κύριες διαφορές του Quality Assurance και του Quality Control [18].



Σχήμα 2.1: Quality Assurance vs Quality Control

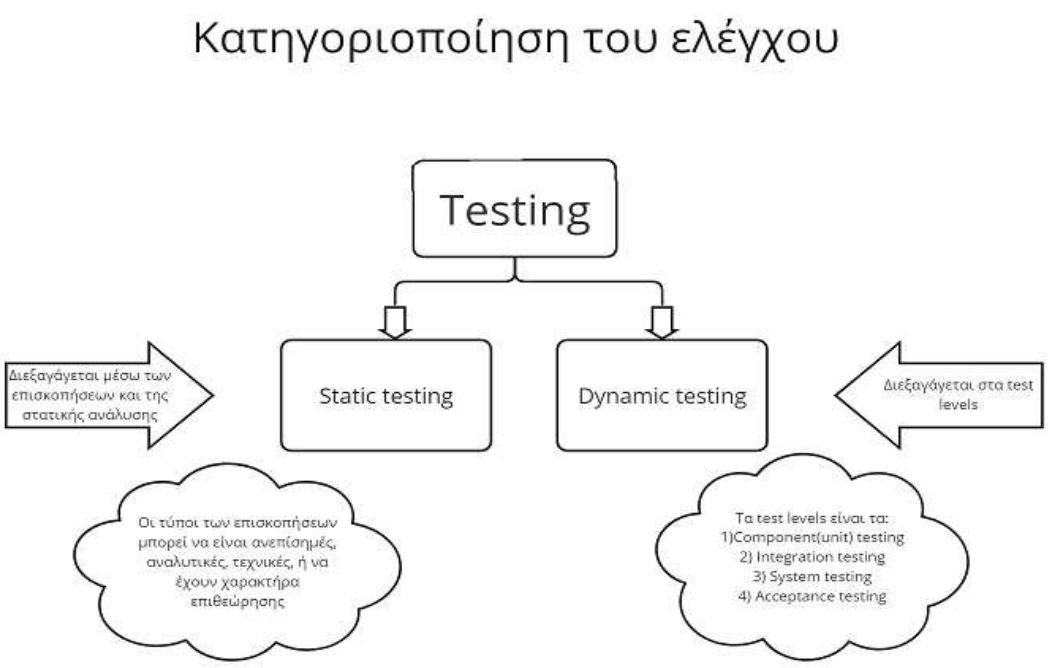
2.4 Κατηγοριοποίηση του ελέγχου

Υπάρχουν αρκετοί μύθοι για το ποιες είναι οι διαφορετικές κατηγορίες στο testing. Πολλοί αναφέρουν ότι το testing κατανέμεται στο White box και στο Black box testing, ενώ άλλοι αναφέρουν ότι το testing κατηγοριοποιείται στο το manual και στο automation testing. Στην πραγματικότητα οι δύο διαφορετικές κατηγορίες του software testing είναι το static και το dynamic testing.

Το static testing έχει να κάνει με τον έλεγχο των στατικών work products (documentations, specifications, requirements, prerequisites, user stories, designs, flow charts, control flow, regulations) και του κώδικα χωρίς αυτός να βρίσκεται σε εκτέλεση (static analysis). Για οτιδήποτε μη εκτελέσιμο work product γίνεται έλεγχος, τότε διαδραματίζεται static testing. Η αξία της συνεισφοράς των QAE στο πρώιμο επίπεδο και ο σκοπός του static testing είναι να βρεθούν ατέλειες και ελλείψεις από το αρχικό στάδιο της σχεδίασης λογισμικού πριν καν ξεκινήσει η ανάπτυξη ώστε να αποφευχθούν τα μειονεκτήματα που αναφέρθηκαν προηγουμένως. Όταν το static testing ολοκληρωθεί τότε τα static work products (γνωστά και ως «test basis») θα χρησιμοποιηθούν ώστε να δημιουργηθούν τα test cases για τα διάφορα επίπεδα του testing. Στην συνέχεια η εκτέλεση των test cases ορίζεται ως dynamic testing το οποίο λαμβάνει μέρος εκτελώντας το system under testing (δηλαδή εκτελώντας την εφαρμογή).

Γενικότερα το static testing διεξάγεται μέσω των reviews ή αλλιώς μέσω της στατικής ανάλυσης (static analysis) ενώ το dynamic testing διεξάγεται μέσω του ελέγχου των επιπέδων του testing (testing levels).

Στο παρακάτω σχήμα παρουσιάζονται οι κύριες διαφορές του static και του dynamic testing [19].



Σχήμα 2.2: Κατηγοριοποίηση του ελέγχου

2.5 Debugging vs testing

Συνήθως η κοινή γνώμη είναι ότι το debugging και το testing αναφέρονται ακριβώς στην ίδια διαδικασία αλλά στην πραγματικότητα πρόκειται για δύο διαφορετικές έννοιες και διαδικασίες.

Το testing σχετίζεται άμεσα με την εύρεση και τον προσδιορισμό σφαλμάτων σε μια εφαρμογή και είναι μια δραστηριότητα η οποία λαμβάνει μέρος από τους μηχανικούς διασφάλισης της ποιότητας του λογισμικού.

Στην συνέχεια το debugging αποτελεί μια διαδικασία που προκύπτει μετά από το testing και είναι μια δραστηριότητα που λαμβάνει μέρος κυρίως από τους developers στο στάδιο της ανάπτυξης του λογισμικού. Το debugging αποτελείται από τρεις ακόλουθες σειριακές ενέργειες:

1. Ανάλυση και κατανόηση του σφάλματος που αναφέρθηκε από τον QAE
2. Εύρεση της κύριας αιτίας (root cause) του σφάλματος
3. Αφαίρεση της κύριας αιτίας του σφάλματος.

Όπως γίνεται αντιληπτό το testing και οι μηχανικοί διασφάλισης της ποιότητας του λογισμικού δεν διαδραματίζονται στην επιδιόρθωση των σφαλμάτων, αλλά επικεντρώνονται αποκλειστικά στον προσδιορισμό των σφαλμάτων μιας εφαρμογής ή ενός συστήματος. Η επιδιόρθωση των σφαλμάτων γίνεται από τους developers. Στο automation testing, το οποίο θα αναλυθεί στην συνέχεια το debugging είναι μια διαδικασία η οποία μπορεί να εφαρμοστεί και από τον QA Engineer προκειμένου να λύσει σφάλματα που προκύπτουν κατά την διάρκεια της αυτοματοποίησης των Test Cases

2.6 Root Cause, Error, Defect, Failure, Effects

Στον τομέα του ελέγχου συχνά χρησιμοποιούνται αρκετοί όροι προκειμένου να προσδιοριστούν συγκεκριμένες έννοιες ή/και καταστάσεις. Δυστυχώς μερικές φορές υπάρχει μια μικρή σύγχυση μεταξύ

αυτών των όρων κυρίως από άτομα που θεωρούνται αρχάριοι. Μερικοί από τους όρους που συχνά χρησιμοποιούνται λάθος είναι απλές λέξεις ή φράσεις όπως «κύρια αιτία» (root cause), «λάθος» (error, mistake), «ατέλεια» (defect, bug, fault) «αποτυχία» (failure), «επιπτώσεις» (effects). Στην συνέχεια θα ακολουθήσει ένα παράδειγμα προκειμένου να γίνουν ευκολότερα αντιληπτοί αυτοί οι όροι. Σε μια εφαρμογή υπολογισμού μισθοδοσίας ένας developer μπορεί να κάνει ένα λάθος (error) στον υπολογισμό της πληρωμής τόκων το οποίο με την σειρά του θα οδηγήσει στην εμφάνιση μιας ατέλειας (defect, bug) στον κώδικα. Όταν ο εσφαλμένος κώδικας εκτελεστεί, θα προκληθεί μια αποτυχία (software failure) η οποία μπορεί (ή και όχι) να οδηγήσει στην κατάρρευση (crash) του προγράμματος. Οι επιπτώσεις (effects) στο συγκεκριμένο σενάριο θα είναι τα παράπονα των πελατών της επιχείρησης. Η κύρια αιτία (root cause) του σφάλματος αποτελεί τις πρώτες ενέργειες ή συνθήκες που συνέλαβαν στην δημιουργία του ελαττώματος. Στο συγκεκριμένο παράδειγμα η κύρια αιτία (root cause) του σφάλματος θα μπορούσε να είναι η έλλειψη κατανόησης του product owner για τον υπολογισμό της πληρωμής τόκων. Ο εσφαλμένος κώδικας που γράφτηκε για ένα User Story το οποίο ήταν ασαφής λόγω της έλλειψης κατανόησης του υπολογισμού της πληρωμής τόκων από την μεριά του product owner, οδήγησε στην εισαγωγή μιας ατέλειας ή οποία με την σειρά της προκάλεσε μία αποτυχία στον κώδικα που με την σειρά της η αποτυχία προκάλεσε ως επιπτώσεις τα παράπονα των πελάτων της επιχείρησης [1].

2.7 False positive vs False negative

Οι έννοιες του False positive και του False negative είναι δύο πολύ σημαντικές φαινόμενα τα οποία συνδέονται με το αποτέλεσμα της εκτέλεσης των Test cases. Συχνά, δεν αντιστοιχίζονται όλα τα μη αναμενόμενα αποτελέσματα των test με κάποιο bug.

Μπορεί να προκύψουν ψευδώς λανθασμένα αποτελέσματα λόγω σφαλμάτων που προέκυψαν στα tests που εκτελέστηκαν ή λόγω σφαλμάτων στα test data, στο test environment, στο testware ή για διάφορους άλλους λόγους [1]. Ένα παράδειγμα False positive θα μπορούσε να είναι το εξής σενάριο. Υποθέτοντας ότι πρέπει να ελεγχθεί ένα Test case το οποίο αναφέρει ότι ο χρήστης μπορεί να κάνει επιτυχώς log in στην εφαρμογή υπό έλεγχο. Κατά την διάρκεια εκτέλεσης, η σύνδεση του χρήστη με το διαδίκτυο διακόπτεται και το σύστημα εμφανίζει απροσδόκητα ένα μήνυμα λάθους υποδεικνύοντας μια λανθασμένη προσπάθεια για είσοδο στο σύστημα. Το σφάλμα αναφέρεται στην ευρύτερη ομάδα (π.χ. scrum team) αλλά μετά από λίγο διαπιστώνεται ότι το σφάλμα προκλήθηκε λόγω της διακοπής της σύνδεσης με το διαδίκτυο και όχι λόγω της αδυναμίας της λειτουργικότητας του log in. Το αποτέλεσμα της εκτέλεσης του παραπάνω test ήταν false positive επειδή το σύστημα υπέδειξε ένα πρόβλημα (γνωστό και ως false alert) το οποίο στην πραγματικότητα δεν υπήρχε στο SUT (System Under Test). Τα false positives αποτελέσματα συνδέονται με την καταγραφή «failed» αποτελεσμάτων και κατά συνέπεια αναφέρονται bugs τα οποία δεν θα έπρεπε να είχαν αναφερθεί, καθώς λανθασμένα θεωρείται ότι υπάρχει σφάλμα. Τα false positives αποτελέσματα δημιουργούν σύγχυση στην ευρύτερη ομάδα χωρίς να υπάρχει ουσιαστικός λόγος και σαφώς πρέπει να αποφεύγονται.

Μπορεί επίσης να καταγραφεί η αντίστροφη κατάσταση όπου παρόμοια σφάλματα οδηγούν στο φαινόμενο του false negative. Ένα παράδειγμα False negative θα μπορούσε να είναι το εξής σενάριο. Υποθέτοντας ότι διεξάγεται έλεγχος σε ένα website το οποίο θα χρησιμοποιηθεί σε ένα σχολείο για να πραγματοποιηθεί εξέταση πολλαπλών επιλογών. Έχει καταγραφεί στις προδιαγραφές ότι αυτό το σχολείο έχει μαθητές που χρησιμοποιούν αρκετά την οθόνη αφής. Έστω ότι η σωστή απάντηση είναι η πρώτη επιλογή. Ο χρήστης αντί να επιλέξει την πρώτη επιλογή κάνοντας κλικ, επιλέγει την πρώτη επιλογή μέσω της οθόνης αφής του φορητού υπολογιστή του. Το σύστημα καταγράφει λανθασμένα ότι ο χρήστης επέλεξε την τέταρτη επιλογή. Σε αυτή τη περίπτωση, το σύστημα δεν αναγνώρισε σωστά την επιλογή του χρήστη επειδή δεν πραγματοποιήθηκε σωστά έλεγχος για την επιλογή απάντησης. Σε

περίπτωση που είχε διεξαχθεί σωστά έλεγχος για την επιλογή μέσω οθόνης αφής, αυτό το σφάλμα θα είχε εντοπιστεί. Έπειτα το σφάλμα θα είχε διορθωθεί και δεν θα προέκυπτε. Το αποτέλεσμα της εκτέλεσης του παραπάνω test ήταν false negative επειδή δεν αναφέρθηκε σφάλμα από την ομάδα ελέγχου στην ευρύτερη ομάδα για την περίπτωση που ο χρήστης επιλέξει απάντηση χρησιμοποιώντας την οθόνη αφής. Τα False negative αποτελέσματα συνδέονται με τα tests τα οποία αδυνατούν να εντοπίσουν σφάλματα τα οποία θα έπρεπε να είχαν εντοπίσει. Επίσης τα false negative είναι πιο επικίνδυνα από τα false positives καθώς μπορούν να έχουν σημαντικές επιπτώσεις στο προϊόν, εφόσον τα tests που θα έπρεπε να είχαν καταγραφεί ως «failed» καταγράφονται ως «passed», ή δεν ελέγχονται καν, λόγω ελλιπούς ελέγχου ή έλλειψης προσοχής ή δυνατοτήτων για επαρκή έλεγχο από την μεριά της ομάδας ελέγχου.

2.8 Verification Vs Validation

Η επαλήθευση (verification) και η επικύρωση (validation) αποτελούν δύο κύριες βασικές έννοιες στον έλεγχο του λογισμικού οι οποίες συνδέονται με διαφορετικές διαδικασίες προκειμένου να επιτευχθεί η ποιότητα στο λογισμικό.

Η έννοια της επαλήθευσης είναι η διαδικασία ελέγχου που διεξάγεται προκειμένου να ελεγχθεί εάν το λογισμικό πετυχαίνει τον στόχο του χωρίς σφάλματα και μέσω αυτής απαντάται η ερώτηση «κατασκευάστηκε σωστά το σύστημα;» Επαληθεύει εάν αναπτυγμένο προϊόν πληροί τις προϋποθέσεις (requirements) και συνδέεται άμεσα με τον στατικό έλεγχο (static testing) που προαναφέρθηκε στο υποκεφάλαιο (2.4).

Η έννοια της επικύρωσης είναι η διαδικασία ελέγχου που διεξάγεται προκειμένου να ελεγχθεί εάν το λογισμικό είναι ικανοποιητικό και εάν καλύπτει τις υψηλού επιπέδου απαιτήσεις των χρηστών. Αποτελεί την διαδικασία ελέγχου της επικύρωσης του προϊόντος και απαντά στην ερώτηση «κατασκευάστηκε το σωστό σύστημα;». Η επικύρωση συνδέεται άμεσα με τον δυναμικό έλεγχο (dynamic testing) που προαναφέρθηκε στο υποκεφάλαιο (2.4). Στον παρακάτω πίνακα αναλύονται οι διαφορές των εννοιών της επαλήθευσης και της επικύρωσης [20].

Πίνακας 2.1: Verification Vs Validation

Verification	Validation
Περιλαμβάνει τον έλεγχο των εγγράφων, σχεδίων, κώδικα και προγραμμάτων	Περιλαμβάνει τον έλεγχο και την επικύρωση του προϊόντος που έχει αναπτυχθεί
Η επαλήθευση είναι ο στατικός έλεγχος	Η επικύρωση είναι ο δυναμικός έλεγχος
Δεν περιλαμβάνει την εκτέλεση κώδικα	Περιλαμβάνει την εκτέλεση κώδικα
Οι μέθοδοι που χρησιμοποιούνται στην επαλήθευση είναι κυρίως οι επισκοπήσεις (reviews)	Οι μέθοδοι-τεχνικές που χρησιμοποιούνται κατά την διαδικασία της επικύρωσης είναι το Black-Box Testing, το White-Box Testing και το (NFT) Non-Functional Testing
Ελέγχει εάν το λογισμικό συμμορφώνεται με τις προδιαγραφές	Ελέγχει εάν το λογισμικό πληροί τις απαιτήσεις και τις προσδοκίες ενός πελάτη
Μπορεί να ανακαλύψει ατέλειες στο πρώιμο στάδιο της ανάπτυξης λογισμικού	Μπορεί να ανακαλύψει ατέλειες που δεν μπορούν να βρεθούν μέσω της διαδικασίας της επαλήθευσης
Ο στόχος της επαλήθευσης είναι η αρχιτεκτονική και η προδιαγραφή εφαρμογών και λογισμικού	Ο στόχος της επικύρωσης είναι ένα σωστά λειτουργικό προϊόν

Η ομάδα της διασφάλισης της ποιότητας διεξάγει την διαδικασία της επαλήθευσης	Η επικύρωση εκτελείται στον κώδικα του λογισμικού με την βοήθεια της ομάδας που διεξάγει τον έλεγχο
Λαμβάνει μέρος πριν την διαδικασία της επικύρωσης	Λαμβάνει μέρος μετά την διαδικασία της επαλήθευσης
Αποτελείται από έλεγχο εγγραφών/αρχείων και εκτελείται από τον άνθρωπο	Αποτελείται από την εκτέλεση του προγράμματος και εκτελείται από τον υπολογιστή

2.9 Οι εφτά βασικές αρχές του ελέγχου λογισμικού

Υπάρχουν εφτά θεμελιώδεις αρχές στο software testing οι οποίες ορίζονται από το ISTQB που θα πρέπει αρχικά να γίνουν αντιληπτές από όλο το Scrum Team και στην συνέχεια να ακολουθούνται κατά την διάρκεια του κύκλου ζωής λογισμικού και του μοντέλου ανάπτυξης του. Οι παρακάτω αρχές θα πρέπει να λαμβάνονται σοβαρά υπόψιν και να λειτουργούν ως πρακτικές για τον τρόπο σκέψης για οποιονδήποτε ελέγχει ένα προϊόν λογισμικού.

2.9.1 Ο έλεγχος δείχνει την παρουσία σφαλμάτων αλλά όχι και την απουσία τους

Το testing φέρνει στην επιφάνεια σφάλματα του λογισμικού, αλλά αυτό δεν συνεπάγεται με το ότι το testing μπορεί να λειτουργήσει ως μια απόδειξη ορθότητας της εφαρμογής. Αυτό είναι λογικό διότι το testing μειώνει σημαντικά την εμφάνιση ατελειών στην εφαρμογή υπό έλεγχο αλλά ακόμη και αν οι μηχανικοί διασφάλισης της ποιότητας πιστεύουν πως έχουν βρει όλες τις πιθανές ατέλειες κατά την διάρκεια του testing, τότε ταυτόχρονα δεν υπάρχει καμία απόδειξη ότι η εφαρμογή λειτουργεί άψογα χωρίς ατέλειες οποιαδήποτε χρονική στιγμή. Έστω ότι γίνεται η παραδοχή πως μια συγκεκριμένη χρονική στιγμή δεν υπάρχουν ατέλειες, εάν έπειτα η ομάδα του testing διεξάγει παραπάνω έλεγχο και ανακαλύψει κάποιο σφάλμα που δεν είχε παρατηρήσει πριν την παραδοχή, τότε επιβεβαιώνεται η συγκεκριμένη αρχή. Όπως γίνεται αντιληπτό η εξής παραδοχή μπορεί να αποτελέσει μια αναδρομή η οποία πρέπει να αποφεύγεται διότι διαφορετικά πάντα θα καταλήγαμε σε επιβεβαίωση αυτής της αρχής [19].

2.9.2 Ο εξαντλητικός έλεγχος είναι αδύνατος

Το exhaustive testing αναφέρεται στον εξαντλητικό έλεγχο όλων των πιθανών συνδυασμών, εισόδων (inputs) και προαπαιτούμενων (prerequisites). Γενικότερα είναι μια τακτική η οποία τις περισσότερες φορές δεν είναι δυνατών να εφαρμοστεί επειδή κάτι τέτοιο θα σήμαινε την δημιουργία άπειρων Test Cases. Η πρακτική που πρέπει ακολουθείται είναι η δημιουργία Test Cases τα οποία είναι πολύ πιθανά να συμβούν από την μεριά του χρήστη. Αυτά τα Test Cases έχουν υψηλότερη προτεραιότητα διότι μέσω αυτών ελέγχονται οι κύριες λειτουργίες της εφαρμογής. Επειδή δεν υπάρχει πάντα καθορισμένη σειρά και πέρας των συνδυασμών των ενεργειών που λαμβάνουν χώρα σε μία εφαρμογή, πάντα θα υπάρχει χώρος να προστεθούν καινούρια test cases. Σε ένα project πάντα υπάρχει ένα χρονοδιάγραμμα και μια προθεσμία (deadline) τα οποία πρέπει να ακολουθούνται ώστε να παραδοθεί το προϊόν πριν μια ορισμένη ημερομηνία. Έτσι η δημιουργία και ο έλεγχος άπειρων σεναρίων ροής δεν συνάδει με την πραγματικότητα [1].

2.9.3 Ο πρώιμος έλεγχος σώζει χρόνο και χρήματα

Σε κάθε έργο λογισμικού συνιστάται η ομάδα ή τουλάχιστον ένας μηχανικός διασφάλισης της ποιότητας να είναι παρόν στις συζητήσεις από την στιγμή που συλλέγονται τα απαιτούμενα και από την στιγμή που τα προσχέδια των specifications, documentation και των designs θα είναι έτοιμα. Αυτό γενικά συνδράμει τη ομάδα του testing να διορθώσει τυχόν ατέλειες στα παραπάνω work products τα οποία θα βοηθήσουν τους stakeholders να καταλάβουν αν υπάρχουν τυχόν εμπόδια κατά την δημιουργία του τεχνικού σχεδιασμού του λογισμικού. Επίσης η συμμετοχή των μηχανικών διασφάλισης της ποιότητας σε αυτές τις συζητήσεις βοηθάει τους ίδιους να κατανοήσουν το προϊόν μελετώντας τα test basis (specifications, documentations, designs, preconditions, prerequisites, κλπ.) και κατ' επέκταση να δημιουργήσουν τα Test Cases καθώς και να επεκταθούν στο dynamic testing (δυναμικό έλεγχο). Εάν οι ατέλειες και οι ελλείψεις προσδιοριστούν από πρώιμο στάδιο τόσο πιο εύκολα, γρήγορα και λιγότερο δαπανηρά θα διορθωθούν διότι έτσι θα είχε αποφευχθεί η εμφάνιση αυτών των σφαλμάτων στον κώδικα. Το testing όταν το έργο λογισμικού βρίσκεται ακόμη σε πρώιμο στάδιο είναι γνωστό και με τον όρο "*Shift left*".

2.9.4 Τα ελαττώματα λογισμικού συγκεντρώνονται

Η πυκνότητα με την οποία εμφανίζονται τα σφάλματα δεν είναι πάντα καταμερισμένη σε όλα τα τμήματα (modules) του λογισμικού. Επίσης δεν είναι απίθανο πολλά σφάλματα να έχουν συγκεντρωθεί σε ένα συγκεκριμένο τμήμα (module) του λογισμικού. Τα παραπάνω σενάρια είναι πιθανά για πολλούς λόγους όπως το ότι μπορεί να έχει δοθεί υψηλότερη ή λιγότερη προτεραιότητα σε μερικά τμήματα (modules) της εφαρμογής. Ο έλεγχος πρέπει να γίνεται σε κάθε τμήμα της εφαρμογής ανεξάρτητα από τα αποτελέσματα ελέγχου των συναφών τμημάτων του [19].

2.9.5 Προσοχή στο pesticide paradox

Καταρχάς πρέπει να εξηγηθεί η σημασία του pesticide paradox. Ο Boris Beizer δήλωσε το εξής που στην συνέχεια έγινε γνωστό ως το "Pesticide paradox": «Κάθε μέθοδος που χρησιμοποιείται για την αποτροπή ή την εύρεση σφαλμάτων αφήνει ένα υπόλειμμα πιο λεπτομερών σφαλμάτων έναντι των οποίων αυτές οι μέθοδοι είναι αναποτελεσματικές». Έτσι, εάν οι ίδιες δοκιμές ελέγχου εκτελούνται ξανά και ξανά τότε μετά από ένα σημείο το σύνολο των δοκιμών (Test Cases) δεν θα βρίσκει κανένα νέο ελάττωμα. Αυτό ταυτόχρονα δεν σημαίνει ότι τα τμήματα της εφαρμογής που ελέγχονται δεν περιέχουν άλλα ελαττώματα αλλά σημαίνει ότι τα σενάρια δοκιμών δεν είναι πλέον ικανά να βρουν καινούρια σφάλματα ή ότι τα σενάρια είναι απαρχαιωμένα. Έτσι αντί να γίνεται η επανάληψη εκτέλεσης των ίδιων δοκιμών ξανά και ξανά πρέπει να δημιουργούνται καινούριες δοκιμές ελέγχου με στόχο να βρεθούν νέα σφάλματα. Μετά από κάποιο στιγμή τα test cases δεν είναι πλέον αποτελεσματικά στην εύρεση νέων ελαττωμάτων, όπως τα φυτοφάρμακα δεν είναι πλέον αποτελεσματικά στη θανάτωση των εντόμων [1].

2.9.6 Ο έλεγχος είναι σχετικός με το περιεχόμενο

Δύο διαφορετικές εφαρμογές δεν ελέγχονται με ακριβώς τις ίδιες προσεγγίσεις (π.χ. μια εφαρμογή λήψης μετρικών εξ' αποστάσεως χειρουργικής επέμβασης σε πραγματικό χρόνο σε αντίθεση με ένα ασύγχρονο turn-based παιχνίδι στρατηγικής). Σε αυτές τις δύο περιπτώσεις οι προσεγγίσεις που θα πρέπει να ακολουθηθούν είναι τελείως διαφορετικές λόγω της φύσης των εφαρμογών. Ο έλεγχος θα πρέπει να επικεντρωθεί σε διαφορετικά επίπεδα και τύπους για την κάθε περίπτωση καθώς είναι διαφορετικοί οι στόχοι της κάθε εφαρμογής. Στην περίπτωση της εφαρμογής χειρουργικής επέμβασης

θα πρέπει να δοθεί σημαντική βαρύτητα στα θέματα του non-functional testing (performance testing, stress testing, κλπ.) ενώ στην περίπτωση του ασύγχρονου παιχνίδι στρατηγικής θα πρέπει να δοθεί περισσότερη έμφαση στο functional testing(unit testing, component testing, etc). Η προσπάθεια που απαιτείται είναι διαφορετική για το κάθε project για αυτό οι προσεγγίσεις ελέγχου πρέπει να συμβαδίζουν με το κάθε έργο λογισμικού [19]. Οι διαφορετικές προσεγγίσεις, τα επίπεδα και οι τύποι του testing αναλύονται σε επόμενα κεφάλαια.

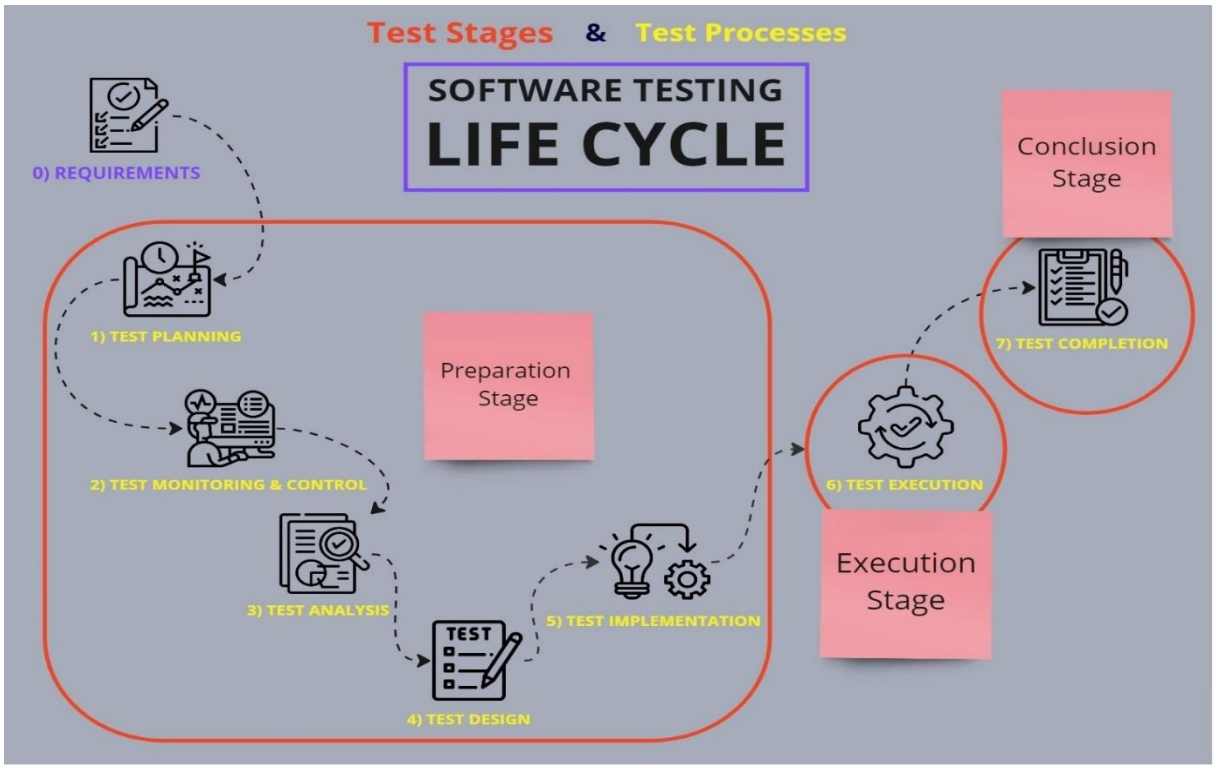
2.9.7 Η απουσία των σφαλμάτων είναι απάτη

Ακόμη και μετά την εύρεση και την διόρθωση πολλαπλών σφαλμάτων η εφαρμογή είναι πιθανό να αποτύχει στην αγορά γιατί μπορεί να μην είναι χρήσιμη στον πελάτη/χρήστη είναι να είναι υποδεέστερη των ανταγωνιστών. Έτσι ανεξάρτητα από το πόσα και πόσο σημαντικά σφάλματα έχουν ανακαλυφτεί και έχουν διορθωθεί, αν η εφαρμογή δεν "καλύπτει" τα απαιτούμενα, δεν θα είναι χρήσιμη για τον τελικό χρήστη. Η επικύρωση της κάλυψης των προαπαιτούμενων είναι εξίσου σημαντική για την επιτυχία μιας εφαρμογής με τον έλεγχο της λειτουργικότητας και της απόδοσης της εφαρμογής.

2.10 Ο κύκλος ζωής του ελέγχου λογισμικού (STLC)

Ανάλογα με την φύση της κάθε εφαρμογής υπάρχουν πολλοί παράγοντες οι οποίοι μπορούν να επηρεάσουν την διαδικασία του testing για μια επιχείρηση. Οι κύριοι λόγοι συνήθως αφορούν τον κύκλο ζωής του λογισμικού και τις μεθοδολογίες έργων λογισμικού που ακολουθεί οι επιχείρηση. Μερικοί επίσης σημαντικοί λόγοι που η διαδικασία του ελέγχου μπορεί να διαφέρει από επιχείρηση σε επιχείρηση έχει να κάνει με τον ευρύτερο τομέα τους και τα ρίσκα του προϊόντος. Λειτουργικοί περιορισμοί επίσης μπορούν να επηρεάσουν το κύκλο ζωής του ελέγχου (STLC) μιας επιχείρησης όπως για παράδειγμα ο διαθέσιμος προϋπολογισμός, το δυναμικό, η πολυπλοκότητα, τα πρότυπα, η υπάρχουσα κατάσταση στην αγορά, κ.ο.κ, ενός έργο λογισμικού. Οι παραπάνω παράγοντες θα πρέπει να ληφθούν υπόψιν έτσι ώστε να προσδιοριστούν με σαφήνεια τα επίπεδα (test levels) και οι τύποι (test types) που πρόκειται να διεξαχθεί ο έλεγχος. Έτσι η διαδικασία αυτή μπορεί να διαφέρει αρκετά για την κάθε επιχείρηση.

Προκειμένου να γίνει κατανοητή η διαδικασία του ελέγχου ή αλλιώς ο κύκλος ζωής του ελέγχου (STLC), η διαδικασία αυτή μπορεί να παρουσιαστεί στο παρακάτω σχήμα ως ένα σύνολο από 3 στάδια, από τα οποία το καθένα εξυπηρετεί έναν συγκεκριμένο σκοπό [21].



Σχήμα 2.3: Software Testing Life Cycle (STLC)

Το καθένα από αυτά τα τρία στάδια με την σειρά του περιλαμβάνει ένα σύνολο από δραστηριότητες που πρέπει να διεκπεραιωθούν για να επιτευχθούν διαφορετικοί στόχοι. Το πρώτο στάδιο ονομάζεται «Preparation stage» και σχετίζεται κυρίως με οτιδήποτε έχει να κάνει με την προετοιμασία της διεξαγωγής του ελέγχου. Το δεύτερο στάδιο ονομάζεται «Execution stage» και αφορά τις ενέργειες που πρέπει να πραγματοποιηθούν προκειμένου να εκτελεστούν οι δοκιμές (test cases). Τέλος το τελευταίο στάδιο ονομάζεται «Conclusion stage» και σχετίζεται με την διεξαγωγή των αποτελεσμάτων του ελέγχου. Η εμβάθυνση της εμφώλευσης στον κύκλο ζωής του ελέγχου δεν σταματάει εδώ καθώς, κάθε δραστηριότητα του κάθε σταδίου αποτελείται από πολλαπλές μεμονωμένες εργασίες, οι οποίες διαφέρουν για κάθε έργο λογισμικού ή για κάθε έκδοση του ίδιου λογισμικού. Τέλος με την ολοκλήρωση της κάθε μεμονωμένης εργασίας της κάθε δραστηριότητας που με την σειρά της ανήκει σε ένα από τα τρία στάδια του STLC, παράγεται κάποιο test work product ή αλλιώς Test deliverable. Το κάθε test work product με την σειρά του εξυπηρετεί έναν συγκεκριμένο απώτερο σκοπό και παράλληλα αποδεικνύει το αποτέλεσμα της δουλειάς που έγινε κατά την διάρκεια μιας από τις επτά δραστηριότητες του ελέγχου.

Αυτά τα στάδια και οι δραστηριότητες μπορεί να φαίνονται λογικά διαδοχικές άλλα πολλές φορές εφαρμόζονται επαναληπτικά και μερικώς παράλληλα. Η συντριπτική πλειοψηφία των επιχειρήσεων λογισμικού ακολουθεί agile μεθοδολογίες (π.χ. scrum, SAFe) οι οποίες περιλαμβάνουν μικρές επαναλήψεις (sprints, iterations αντίστοιχα για την κάθε μεθοδολογία) ανάλυσης και σχεδιασμού, ανάπτυξης και ελέγχου λογισμικού. Έτσι με την σειρά τους οι δραστηριότητες του ελέγχου συμβαίνουν επίσης σε επαναληπτική και συνεχή βάση. Ακόμη και σε πιο "παραδοσιακές" (ως προς την αντίληψη ανάπτυξης προϊόντων λογισμικού) επιχειρήσεις που ακολουθούν σειριακά μοντέλα ανάπτυξης λογισμικού (π.χ. Waterfall model) προκύπτει πως η βαθμιαία λογική ακολουθία των κύριων ομάδων δραστηριοτήτων αναγκαστικά περιλαμβάνει επικάλυψη, συνδυασμό, συγχρονισμό ή παράλειψη, άρα προσαρμογή.

2.10.1 Test planning

Το test planning είναι η αρχική δραστηριότητα του STLC και έχει στόχο την δημιουργία δύο πολύ σημαντικών Test deliverables, τα οποία ονομάζονται «(MTP) Master Test Plan» και «Test plan». Οι ορισμοί που δίνονται στο ISTQB για αυτά τα Test deliverables είναι οι εξής:

Το Master test plan είναι: «Έγγραφο σχεδίου ελέγχου που χρησιμοποιείται για τον συντονισμό πολλαπλών Test Levels και Test Types».

Το test plan είναι: «Τεκμηριωμένο έγγραφο που περιγράφει τους στόχους του ελέγχου που πρέπει να επιτευχθούν, τα μέσα και το χρονοδιάγραμμα για την επίτευξή τους καθώς και την οργάνωση για τον συντονισμό των δραστηριοτήτων του ελέγχου».

Τα δύο αυτά έγγραφα περιέχουν όλο το απαραίτητο περιεχόμενο και όλες τις απαραίτητες αναφορές που είναι σχετικές με το project (προδιαγραφές, flow charts, repositories, deadlines, κλπ.) και μια περίληψη (σε υψηλό και γενικό επίπεδο για το Master test plan και σε ποιο χαμηλό και ειδικό επίπεδο για το test plan) σχετική με το τι απαιτείται από το συγκεκριμένο project λογισμικού.

Ένα Master test plan δημιουργείται (για ένα Epic ή αλλιώς (CR) Change Request) περιέχει κυρίως γενικότερες πληροφορίες για όλα τα θέματα τα οποία αφορούν τον έλεγχο για ένα έργο λογισμικού και αποτελεί μια επισκόπηση του test approach. Επίσης περιγράφει την στρατηγική ελέγχου (test strategy) και την διαδικασία που επιλέγεται προκειμένου να πραγματοποιηθεί ο έλεγχος. Με την δημιουργία του αποσαφηνίζονται και δίνονται απαντήσεις σε πολύ βασικά ερωτήματα που έχουν να κάνουν με τον έλεγχο του λογισμικού. Μερικά από αυτά τα απλά αλλά πολύ σημαντικά ερωτήματα μπορούν να είναι τα εξής:

- Τι θα ελεγχθεί σε υψηλό επίπεδο;
- Ποια είναι η εμβέλεια του ελέγχου;
- Ποιοι είναι οι στόχοι του ελέγχου;
- Ποια στρατηγική ελέγχου θα ακολουθηθεί;
- Ποια άτομα ή ομάδες θα διεξάγουν τον έλεγχο (καταμερισμός ευθυνών);
- Πώς και πότε θα διεξαχθούν οι δραστηριότητες του ελέγχου;
- Ποια test levels και test types θα ελεγχθούν;
- Θα διεξαχθεί automation testing;
- Ποια testing tools και ποια testing frameworks θα χρησιμοποιηθούν;
- Ποια είναι τα entry και ποια είναι τα exit criteria;
- Ποια είναι τα Non-functional requirements αν και εφόσον υπάρχουν;
- Ποιες είναι οι μετρικές που πρέπει να ληφθούν υπόψιν για το monitoring and control και για τα KPI?
- Πώς θα ενσωματωθούν και πώς θα ευθυγραμμιστούν οι δραστηριότητες του STLC με τις δραστηριότητες του SDLC;
- Ποια είναι τα testing milestones (ορόσημα του ελέγχου);
- Πώς θα λάβουν μέρος οι επόμενες δραστηριότητες του STLC;
- Ποιες είναι (αν υπάρχουν) οι εξαρτήσεις (dependencies) για το συγκεκριμένο project;
- Ποιος είναι ο διαθέσιμος προϋπολογισμός για την διεξαγωγή του ελέγχου;
- Ποιοι είναι οι περιορισμοί όσον αφορά τον έλεγχο;
- Πόσο κρίσιμο είναι το συγκεκριμένο project?
- Πώς αξιολογείται το testability (η δυνατότητα να πραγματοποιηθεί έλεγχος) του project;
- Ποια είναι τα πιθανά ρίσκα;
- Θα διεξαχθεί exploratory testing; Αν ναι, πότε;

Ένα Test plan δημιουργείται για ένα χαρακτηριστικό (feature) – σύνολο λειτουργιών της εφαρμογής που με την σειρά τους περιέχουν προαπαιτούμενα και απαιτήσεις. Το test plan περιγράφει την

εξαντλητική λίστα όλων των σεναρίων (test cases) που θα ελεγχθούν. Οι ερωτήσεις που μπορεί να απαντάει ένα test plan είναι οι εξής:

- Σε ποιο test level ή test type αναφέρεται το συγκεκριμένο test plan;
- Ποια είναι τα επιμέρους σενάρια (test cases) που θα ελεγχθούν;
- Ποιες ημερομηνίες θα γίνει η «φόρτωση» (load) του feature (χαρακτηριστικού) σε κάθε περιβάλλον (environment);
- Ποιες ημερομηνίες θα γίνει η επισκόπηση του test plan από το scrum team;
- Μπορεί το συγκεκριμένο χαρακτηριστικό (feature) να ενεργοποιηθεί (activated) ή να απενεργοποιηθεί (deactivated); (αυτή η μέθοδος είναι γνωστή και με τον όρο «feature activation»). Αν ναι, με ποιόν τρόπο;
- Πώς θα γίνει το monitoring του feature (αναφορικά και μόνο);

Η συντριπτική πλειοψηφία των επιχειρήσεων-οργανισμών στις μέρες μας χρησιμοποιεί συγκεκριμένα templates τα οποία έχουν σχεδιαστεί από τους QAE για να δημιουργήσουν τα έγγραφα του Master Test Plan και του Test plan. Με αυτόν τον τρόπο οι επιχειρήσεις μπορούν να βασίσουν τον έλεγχο στις ανάγκες τους, σύμφωνα με τα προϊόντα λογισμικού που παράγουν. Το Master Test Plan και το Test plan είναι έγγραφα τα οποία μπορούν να ανανεώνονται και να συμπληρώνονται από τα άτομα που διεξάγουν το testing καθώς το project και ο έλεγχος προχωρούν διότι όλο και περισσότερες λεπτομέρειες γίνονται διαθέσιμες και μπορούν να συμπεριληφθούν σε αυτά. Επιπλέον και τα δύο αυτά έγγραφα πρέπει να εξεταστούν και να συζητηθούν όταν είναι έτοιμα με ολόκληρο το scrum team για feedback και τυχόν διορθώσεις προτού αυτά παρουσιαστούν στους stakeholders. Αυτή η μεμονωμένη εργασία ονομάζεται «Master Test Plan Review» και «Test Plan Review» αντίστοιχα για το κάθε έγγραφο.

Τα Test deliverables του test planning μπορούν να χωριστούν ανάλογα με τις ανάγκες του project. Συνήθως αυτό διαφέρει αρκετά σε κάθε επιχείρηση και project διότι διαφορετικές προσεγγίσεις πρέπει να ακολουθηθούν. Επιπλέον το test planning μπορεί να χωριστεί σε επιμέρους έγγραφα για καλύτερη οργάνωση. Μια συνηθισμένη τακτική είναι ή ακόλουθη. Σε κάθε ευρύτερο project, που σε όρους διοίκησης έργων λογισμικού μεταφράζεται ως «Epic», περιλαμβάνονται επιμέρους user stories το οποία εκφράζουν τις ανάγκες του πελάτη ή/και του χρήστη. Το κάθε user story δομείται από επιμέρους κριτήρια αποδοχής (acceptance criteria) που με την σειρά τους αναφέρονται σε προαπαιτούμενα και σε συγκεκριμένες λειτουργίες - χρήσης της εφαρμογής. Το κάθε Epic μπορεί να "καλυφθεί" από ένα master test plan το οποίο απαντάει τις παραπάνω ερωτήσεις και περιγράφει την συνολική εικόνα του ελέγχου. Στην συνέχεια το κάθε επιμέρους user story μπορεί να αντιστοιχηθεί με ένα Test plan ή με περισσότερα Test plans (αν περισσότερα από ένα test types πρέπει να ελεγχθούν). Το κάθε Test plan αντιστοιχίζεται με ένα user story και περιγράφει αναλυτικά τον έλεγχο ενός συγκεκριμένου Test Type. Τέλος η κύρια πληροφορία για κάθε Test plan είναι τα test cases τα οποία περιγράφουν λεπτομερώς τα σενάρια που πρόκειται να ελεγχθούν. Συνοψίζοντας, όταν ολοκληρωθεί η διαδικασία του Test planning, ως Test deliverables πρέπει να έχουν δημιουργηθεί ένα Master test plan που περιγράφει την συνολική εικόνα του ελέγχου για το έργο λογισμικού και ένα ή περισσότερα Test plans όπου το καθένα περιγράφει αναλυτικά τι και ποια test cases (σενάρια) θα ελεγχθούν για το κάθε Test type ή Test level που θα ελεγχθεί σε ένα user story.

2.10.2 Test monitoring and control

Η επόμενη δραστηριότητα του STLC ονομάζεται «test monitoring and control». Αυτή η δραστηριότητα επικεντρώνεται στην παρακολούθηση της προόδου του έργου λογισμικού στο οποίο διεξάγεται ο έλεγχος.

Σε αυτή τη δραστηριότητα του STLC γίνεται συνεχής σύγκριση των μετρικών των δοκιμών (test metrics) και της κάλυψης των δοκιμών (test coverage) που ορίστηκαν στο test plan (οι τιμές των οποίων πρέπει να επιτευχθούν), με τις μετρικές της χρονικής στιγμής που διεξάγεται ο έλεγχος. Επομένως αυτές οι κρίσιμες μετρικές συλλέγονται κατά την διάρκεια διεξαγωγής του ελέγχου και μεταφράζονται συνήθως σε ποσοστά. Οι test metrics επικεντρώνονται σε ποσοτικές μετρήσεις για την αξιολόγηση της διαδικασίας των δοκίμων (tests) και της προσπάθειας του ελέγχου. Για παράδειγμα μερικές σημαντικές μετρικές (test metrics) για τον έλεγχο θα μπορούσαν να ήταν οι ακόλουθες:

$$\text{Ποσοστό εκτέλεσης των test cases} = \frac{\text{αριθμός των εκτελεσμένων test cases}}{\text{αριθμός των test cases που είχαν σχεδιαστεί για εκτέλεση}} \quad (2.1)$$

$$\text{Ποσοστό επιτυχών των tests} = \frac{\text{αριθμός των tests που εκτελέστηκαν επιτυχώς}}{\text{αριθμός των tests που εκτελέστηκαν}} \quad (2.2)$$

$$\text{Ποσοστό των blocked test cases} = \frac{\text{αριθμός των test cases}}{\text{αριθμός των test cases που είναι αδύνατο να εκτελεστούν}} \quad (2.3)$$

$$\text{Ποσοστό των κρίσιμων σφαλμάτων} = \frac{\text{αριθμός των test cases } \textit{που απέτυχαν}}{\textit{αριθμός των test cases που αναφέρουν κρίσιμο σφάλμα}} \quad (2.4)$$

Ένα ακόμη σημαντικό είδος μετρήσεων που εκφράζεται σε ποσοστά και συχνά λαμβάνεται υπόψιν κατά την δραστηριότητα του test monitoring and control, είναι η «κάλυψη των δοκιμών (test coverage)». Μέσω του test coverage αξιολογείται η ποιοτική έκταση του ελέγχου και η πληρότητα των test cases σε σχέση με τις απαιτήσεις και τις λειτουργίες του λογισμικού. Το test coverage ορίζεται ως ο βαθμός στον οποίο τα καθορισμένα στοιχεία κάλυψης ασκούνται από μια σουίτα δοκιμών (test suite) [22]. Παρακάτω αναφέρονται μερικά παραδείγματα του test coverage.

$$\text{Ποσοστό κάλυψης των απαιτήσεων} = \frac{\text{αριθμός των απαιτήσεων που έχουν ελεγχθεί}}{\text{αριθμός των απαιτήσεων που έχουν προσδιοριστεί}} \quad (2.5)$$

$$\text{Ποσοστό κάλυψης των λειτουργιών} = \frac{\text{αριθμός των } \textit{λειτουργειών που έχουν ελεγχθεί}}{\text{αριθμός των λειτουργιών που έχουν προσδιοριστεί}} \quad (2.6)$$

$$\text{Ποσοστό κάλυψης του κώδικα} = \frac{\text{αριθμός γραμμών ή } \textit{blocks} \text{ του κώδικα που εκτελέστηκε}}{\text{αριθμός γραμμών ή } \textit{blocks κώδικα}} \quad (2.7)$$

Συνοψίζοντας, οι μετρικές ελέγχου (test metrics) χρησιμοποιούνται για την μέτρηση των ποσοστών ολοκλήρωσης, προόδου, αποτελεσματικότητας και ποιότητας των tests, ενώ η κάλυψη των δοκιμών (test coverage) χρησιμοποιείται για την μέτρηση του ποσοστού κάλυψης μιας συγκεκριμένης πτυχής του λογισμικού (κάλυψη του κώδικα, κάλυψη των λειτουργιών, κάλυψη των απαιτήσεων, κλπ).

Εκτός από την παρακολούθηση της προόδου (αναφορά στο test monitoring), η δραστηριότητα αυτή περιλαμβάνει την λήψη ενεργειών έτσι ώστε να επιτευχθούν οι στόχοι του ελέγχου (αναφορά στο test control). Το test control αναφέρεται στις ενέργειες που πρέπει να λάβουν μέρος σε περίπτωση που η πρόοδος του έργου λογισμικού αποκλίνει από αυτή που είχε εξαρχής σχεδιαστεί ή όταν το project αποκλίνει από τους στόχους του. Αυτές οι ενέργειες λειτουργούν ως εναλλακτική επιλογή στις παραπάνω περιπτώσεις και πρέπει να καθορίζονται ρητά στο test planning [19].

2.10.3 Test analysis

Μια εξαιρετικά σημαντική δραστηριότητα του STLC είναι η διαδικασία του Test analysis. Κατά την διάρκεια αυτής της διαδικασίας αναλύονται τα Test basis με σκοπό να προσδιοριστούν τα χαρακτηριστικά και οι λειτουργίες τις εφαρμογής που θα ελεγχθούν. Ως «Test basis» ορίζεται οποιαδήποτε τεκμηρίωση (documentation, specifications, designs, flow charts, κλπ.) χρησιμοποιείται ως πηγή αναφοράς για τον έλεγχο. «Με άλλα λόγια, στην δραστηριότητα του test analysis καθορίζεται «τι πρέπει να ελεγχθεί» υπό την έννοια των κριτήριων κάλυψης του ελέγχου» [1]. Ακόμη,

προσδιορίζονται οι συνθήκες του ελέγχου (test conditions) και η προτεραιότητά τους και γίνονται αντιληπτά τα χαρακτηριστικά της λειτουργίας, απόδοσης και της δομής της εφαρμογής. Επιπλέον, λαμβάνονται υπόψιν παράγοντες που σχετίζονται με την τεχνική αλλά και την επιχειρησιακή πλευρά της εφαρμογής, καθώς και ο αντίκτυπος των ρίσκων. Στην διαδικασία της ανάλυσης του ελέγχου, διεξάγεται static testing καθώς μέσω της ανάλυσης των test πραγματοποιείται επισκόπηση των test basis (γνωστή και ως «specification review») από την μεριά του QAE. Κατά την διάρκεια αυτής της επισκόπησης είναι πολύ πιθανό να εντοπιστούν διάφορα είδη σφαλμάτων όπως:

- Ασάφειες
- Παραλείψεις
- Ασυνέπειες
- Ανακρίβειες
- Αντιφάσεις
- Περιττές δηλώσεις

Η ανάλυση του ελέγχου επίσης είναι μια δραστηριότητα απόκτησης λειτουργικού υπόβαθρου του προϊόντος που θα ελεγχθεί και είναι η προετοιμασία της επόμενης δραστηριότητας του STLC.

2.10.4 Test design

Κατά την δραστηριότητα του Test design οι συνθήκες του ελέγχου (test conditions) που προσδιορίστηκαν στην προηγούμενη δραστηριότητα (test analysis) του STLC μεταφράζονται από τους υπευθύνους του ελέγχου σε test cases. Το κάθε test case, σε υψηλό επίπεδο αντιστοιχεί στον έλεγχο μια συγκεκριμένης λειτουργικότητας της εφαρμογής. Ένα test case έχει διάφορα πεδία όπως προϋποθέσεις, δεδομένα εισόδου, βήματα ενεργειών, αναμενόμενα αποτελέσματα και μετασυνθήκες (postconditions), καθένα των οποίον μπορεί να παίξει καθοριστικό ρόλο στο τελικό αποτέλεσμα (passed, failed, blocked, κλπ.) του test case. Τα βήματα ενεργειών του κάθε test case μπορεί να είναι ενέργειες όπως ένα κλικ ή η αφή σε κάποιο συστατικό μιας διεπαφής χρήστη ή η απευθείας κλήση κάποιου back-end. «Ενώ η δραστηριότητα του test analysis απαντά στο ερώτημα «τι θα ελεγχθεί;», η δραστηριότητα του test design απαντά στο ερώτημα «πώς θα ελεγχθεί;» »[1]. Η δραστηριότητα του test design περιλαμβάνει τις ακόλουθες ενέργειες:

- Σχεδίαση των test cases
- Προσδιορισμών συγκεκριμένων δεδομένων που θα υποστηρίξουν τα test conditions και τα test cases
- Προετοιμασία του περιβάλλοντος και των δεδομένων ελέγχου (test environment, test data) και προσδιορισμός της απαιτούμενης υποδομής για τον έλεγχο
- Συλλογή του bi-directional traceability matrix μεταξύ των test basis, test condition και test cases. (Το bi-directional traceability matrix είναι η αμφίδρομη αντιστοιχία που χρησιμοποιείται για να επιβεβαιωθεί ότι τα test basis έχουν μεταφραστεί σε test cases ή αντίστροφα ότι τα test cases καλύπτουν όλα τα test basis. [19]).

Η μετάφραση των test basis και των test conditions σε test cases συχνά συνδυάζεται με τεχνικές (test techniques) και μεθόδους-πρακτικές του ελέγχου οι οποίες αναλύονται σε επόμενα υποκεφάλαια. Επίσης κατά τον σχεδιασμό των test cases συνήθως χρησιμοποιούνται συγκεκριμένες μέθοδοι οι οποίες βοηθούν στην διεξαγωγή, ανάγνωση και κατανόηση των test cases. Οι μέθοδοι του BDD και του TDD είναι ευρέως γνωστές και χρησιμοποιούμενες από τους QAE και τις επιχειρήσεις προκειμένου να ενισχυθεί η συνολικότερη διαδικασία του ελέγχου. Με αυτές τις μεθόδους γίνονται ξεκάθαρα τα προαπαιτούμενα, η λειτουργία και το αποτέλεσμα του κάθε test case καθώς και κατά επέκταση το τι απαιτείται από το κάθε χαρακτηριστικό κάθε εφαρμογής. Επίσης μέσω της δραστηριότητας του Test

design, προκύπτει το testware υψηλού επιπέδου που στην συνέχεια θα χρησιμοποιηθεί για την διεξαγωγή manual ή automation testing.

Όπως και με την δραστηριότητα του test analysis, η δραστηριότητα του test design μπορεί επίσης να οδηγήσει στον εντοπισμό παρόμοιων τύπων ελαττωμάτων στα test basis, το οποίο οδηγεί σε σημαντικά πιθανά οφέλη [1].

2.10.5 Test implementation

Έπειτα από τις προηγούμενες απαραίτητες δραστηριότητες λαμβάνει χώρα η δραστηριότητα του Test implementation κατά την οποία ολοκληρώνεται το testware σε υψηλό επίπεδο αλλά και σε χαμηλό επίπεδο εάν πρόκειται να διεξαχθεί automation testing. «Έτσι, ενώ η δραστηριότητα του Test design απαντά στην ερώτηση «πως θα διεξαχθεί ο έλεγχος;», η δραστηριότητα του Test implementation απαντά στην ερώτηση «Είναι τα πάντα έτοιμα ώστε να εκτελεστεί ο έλεγχος;» » [1]. Η δραστηριότητα του test implementation περιλαμβάνει τις ακόλουθες ενέργειες:

- Ανάπτυξη και ανάθεση προτεραιότητας των test procedures, test cases και των test suites και σε περίπτωση διεξαγωγής automation testing δημιουργία των test scripts. (Test procedures αποκαλούνται όλες οι σχετικές ενέργειες που απαιτούνται ώστε να εκπληρωθούν τα προαπαιτούμενα και οποιεσδήποτε διαδικασίες ολοκλήρωσης μετά την εκτέλεση).
- Δημιουργία των test suites και οργάνωση τους με τρόπο ο οποίος οδηγεί σε μια αποτελεσματική εκτέλεση του ελέγχου. (Τα test suites αποτελούν σύνολα από test scripts που έχουν κοινά χαρακτηριστικά ή στόχους).
- Πραγματοποίηση των test procedures (προετοιμασία των test environments, των test harness και των υποδομών, προετοιμασία των emulators (εάν απαιτούνται), service virtualization, κλπ.). (Ο ορισμός «test harness» πρόκειται για μια συλλογή από stubs και από προγράμματα οδήγησης (drivers) που απαιτούνται ώστε να εκτελεστούν τα test cases).
- Προετοιμασία των test data και την επιβεβαίωση ότι έχουν μεταφορτωθεί (uploaded) με τον κατάλληλο τρόπο στα test environments.
- Επαλήθευση και ενημέρωση των bi-directional traceability matrix μεταξύ των test basis, test conditions, test cases, test procedures και των test suites.

Οι δραστηριότητες των Test design και Test implementation συχνά συνδυάζονται [1].Σε χαμηλό επίπεδο, ένα test case μπορεί να κωδικοποιηθεί με κάποια γλώσσα προγραμματισμού έτσι ώστε τα βήματα ενεργειών του test case να εκτελούνται αυτόματα και όχι μέσω ανθρώπινης παρέμβασης. Η μετάφραση των test cases από υψηλό και λειτουργικό επίπεδο σε χαμηλό και τεχνικό επίπεδο είναι κύρια διαδικασία των αυτοματοποιημένων δοκιμών (automation testing) και αποτελεί την κυρίαρχη ενέργεια στην δραστηριότητα του test implementation.

2.10.6 Test execution

Η δραστηριότητα του Test execution αποτελεί την πιο βασική δραστηριότητα του STLC καθώς μέσω αυτής εκτελούνται τα test cases, ώστε στην συνέχεια να γίνει η διεξαγωγή των αποτελεσμάτων για το αν το προϊόν λογισμικού λειτουργεί χωρίς σφάλματα και όπως αναμένεται. Το κάθε test case εκτελείται βάση των προαπαιτούμενων και των βημάτων ενεργειών που έχουν οριστεί σύμφωνα με την δραστηριότητα του Test plan και τη δραστηριότητα του Test design. Εάν τα actual results για όλα τα βήματα του test case είναι ίδια με τα expected results για όλα τα βήματα του test case, χωρίς να παρατηρούνται ατέλειες και οποιαδήποτε είδους ανωμαλίες στην εφαρμογή, τότε το τελικό αποτέλεσμα (status) του κάθε test case σημειώνεται ως «επιτυχές» (test passed). Ένα έστω και ένα actual result ενός βήματος δεν είναι ίδιο με το expected result του ίδιου βήματος, τότε το test case σημειώνεται ως «ανεπιτυχές» (test failed). Η δραστηριότητα του Test execution περιλαμβάνει τις εξής ενέργειες:

- Εγγραφή των αναγνωριστικών και των εκδόσεων των test items και των εργαλείων που χρησιμοποιήθηκαν για την εκτέλεση και του testware.
- Εκτέλεση των test cases χειροκίνητα ή αυτόματα
- Σύγκριση των αποτελεσμάτων (actual results) με τα αναμενόμενα αποτελέσματα (expected results)
- Ανάλυση των ανωμαλιών που παρατηρήθηκαν με σκοπό τον προσδιορισμό των πιθανών αιτιών
- Καταγραφή των αποτελεσμάτων της εκτέλεσης των test cases
- Αναφορά σφαλμάτων βάση των failed tests που καταγράφηκαν (passed, failed, blocked)
- Επανάληψη διαδικασιών του ελέγχου είτε λόγω ανωμαλιών που παρατηρήθηκαν είτε λόγω άλλων παραγόντων (π.χ. επανεκτέλεση ενός test που υποβλήθηκε σε διόρθωση, confirmation testing, regression testing, κλπ.)
- Επαλήθευση και ενημέρωση των bi-directional traceability matrix μεταξύ των test basis, test conditions, test cases, test procedures και των test results. [1]

2.10.7 Test completion

Η τελευταία δραστηριότητα του STLC ονομάζεται «Test completion» και προκύπτει ως αποτέλεσμα της διαδικασίας του Test execution. Κατά την διάρκεια του Test execution συλλέγονται τα δεδομένα και τα αποτελέσματα από τις ολοκληρωμένες ενέργειες που πραγματοποιήθηκαν σχετικά με τον έλεγχο ώστε να συνοψιστεί η συνολική προσπάθεια του ελέγχου. «Η δραστηριότητα του Test completion συνήθως πραγματοποιείται στα χρονικά σημεία τα οποία αποτελούν ορόσημα του έργου λογισμικού (π.χ. κυκλοφορία μιας εφαρμογής, ολοκλήρωση ή ακύρωση του ελέγχου για το έργο λογισμικού, τερματισμός ενός scrum sprint ή agile iteration, ολοκλήρωση ενός test level ή κυκλοφορία μια έκδοσης της ομάδας της συντήρησης (maintenance release))» [1]. Οι ακόλουθες βασικές ενέργειες προκύπτουν κατά την ολοκλήρωση του ελέγχου (test completion):

- «Έλεγχος εάν όλα τα σφάλματα που έχουν αναφερθεί έχουν επιδιορθωθεί και αναφορά αυτών που έχουν εξακολουθούν να παρουσιάζονται.» [1].
- Δημιουργία του Test Summary Report, με σκοπό την επικοινωνία της σύνοψης του ελέγχου, στους stakeholders, business owners, product owners, managers, QA Lead. (Το Test Summary Report περιέχει οργανωμένα τα αποτελέσματα (statuses) των εκτελεσμένων test cases και οποιαδήποτε άλλη πληροφορία κρίνει ο QAE ότι είναι χρήσιμη όπως για παράδειγμα τα εναπομείναντα σφάλματα που χρήζουν διόρθωση).
- «Ολοκλήρωση και αρχειοθέτηση των test environment, test data, test infrastructure, testware για πιθανή χρήση στο μέλλον.
- Παράδοση του testware στην ομάδα της συντήρησης (maintenance team), σε άλλες ομάδες της επιχείρησης ή/και στους stakeholders που μπορούν να ευνοηθούν από αυτό.
- Ανάλυση της γενικότερης εμπειρίας του ελέγχου για το συγκεκριμένο test object με σκοπό να οριστούν οι αλλαγές που χρειάζονται για τις μελλοντικές επαναλήψεις (agile iterations ή scrum sprints), κυκλοφορίες λογισμικού και τα μελλοντικά έργα λογισμικού.
- Χρήση των πληροφοριών που συγκεντρώθηκαν με στόχο να «ωριμάσει» η γενικότερη διαδικασία του ελέγχου» στο πλαίσιο της επιχείρησης [1].

2.11 Test work products

Για κάθε επιμέρους δραστηριότητα του STLC υπάρχει και ένα ή περισσότερα Test work products (γνωστά και ως «Test deliverables») τα οποία προκύπτουν ως αποτέλεσμα της κάθε εργασίας. Το κάθε Test work product μπορεί να θεωρηθεί και ως απόδειξη για το ότι πραγματοποιήθηκε η κάθε επιμέρους δραστηριότητα του STLC. Αξίζει να αναφερθεί ότι ο τρόπος με τον οποίο πραγματοποιείται ο STLC μπορεί να διαφέρει αρκετά από επιχείρηση σε επιχείρηση επομένως είναι αναμενόμενο και τα αντίστοιχα test work products να διαφέρουν ελαφρώς ή σημαντικά για κάθε επιχείρηση.

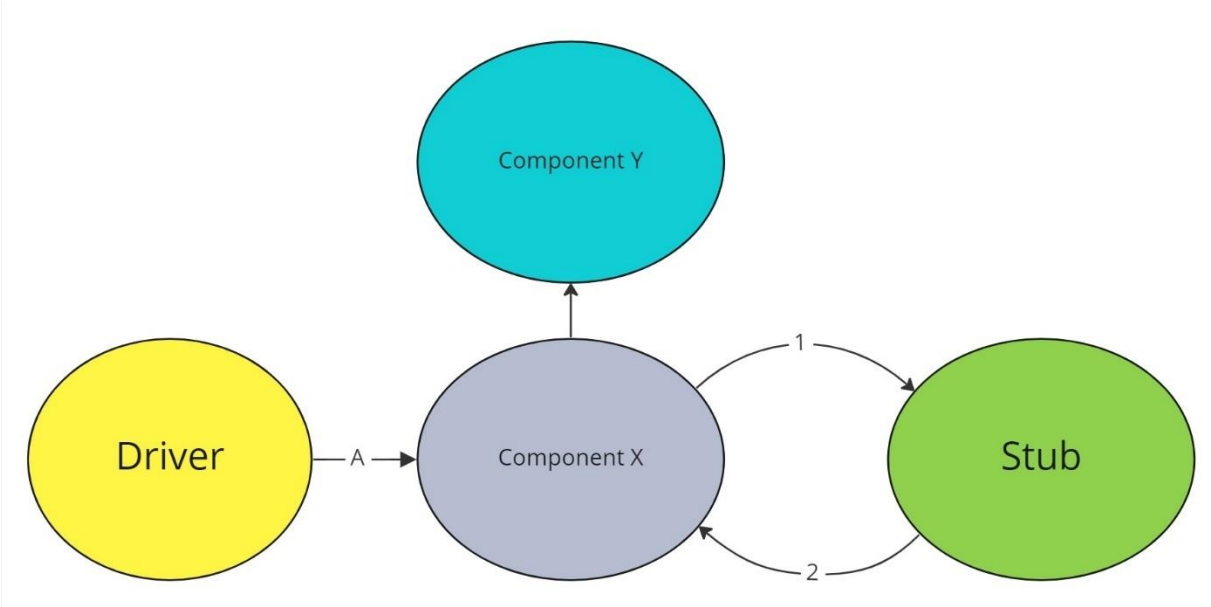
Ο STLC ξεκινάει με την δραστηριότητα του Test Planning. Τα test work products για το test planning μπορούν να είναι ένα Master Test Plan και ένα ή περισσότερα Test Plans για το αντίστοιχο Test Level ή Test Type που καλύπτει το κάθε Test Plan.

Στην συνέχεια λαμβάνει μέρος η διαδικασία του Test Monitoring and Control. Μετά την ολοκλήρωση αυτής της διαδικασίας παράγονται δύο συγκεκριμένα έγγραφα. Το πρώτο ονομάζεται «Test Progress Report» ενώ το δεύτερο ονομάζεται «Test Summary Report». Αυτά τα έγγραφα περιέχουν σχετικές πληροφορίες για την πρόοδο του testing για μια συγκεκριμένη ημερομηνία περιλαμβάνοντας τα αποτελέσματα των tests που εκτελέστηκαν καθώς και πληροφορίες σχετικές με το ποσοστό ολοκλήρωσης του project, την κατανομή πόρων και την προσπάθεια (εργατοώρες) που χρειάστηκε το project προκειμένου να αναπτυχθεί το project μέχρι την αναφέρουσα ημερομηνία. Το «Test Progress Report» αποστέλνεται σε τακτική βάση (π.χ. εβδομαδιαία) στους stakeholders και στον Test Manager του project από τους QAE, ενώ το «Test Summary Report» αποστέλνεται όταν το project πετυχαίνει συγκεκριμένα ορόσημα (milestones) ή κατά την ολοκλήρωση ενός iteration / sprint στους stakeholders και στον QA Lead του project κυρίως από τους Test Managers.

Έπειτα ακολουθεί η δραστηριότητα του Test Analysis όπου παράγονται και θέτονται σε προτεραιότητα οι συνθήκες του ελέγχου (test conditions). Οι συνθήκες ελέγχου ορίζονται ως μια πτυχή ενός component ή ενός συστήματος η οποία μπορεί να ελεγχθεί και προσδιορίζεται ως η βάση του testing [22]. Επίσης κατά την διάρκεια του Test Analysis μπορεί να λάβει μέρος Exploratory Testing το οποίο με την σειρά του μπορεί να οδηγήσει στην δημιουργία των Test Charters και ακόμη στην ανακάλυψη και αναφορά σφαλμάτων (bug reporting). Το Test Charter ορίζεται ως η τεκμηρίωση του στόχου για μια δοκιμαστική συνεδρία (test session) [22].

Ακολουθούν οι δραστηριότητες του Test Design και του Test Implementation. Τα test work products που προκύπτουν από την δραστηριότητα του Test Design περιλαμβάνουν τον σχεδιασμό των High Level Test Cases με σκοπό τον έλεγχο των test conditions που ορίστηκαν κατά την δραστηριότητα του Test Analysis. Επίσης μερικά ακόμη test work products που παράγονται κατά την διάρκεια του Test Design, είναι ο προσδιορισμός και ο σχεδιασμός των test data που θα χρειαστούν για να πραγματοποιηθεί ο έλεγχος, ο σχεδιασμός του test environment και ο προσδιορισμός των εργαλείων που θα χρησιμοποιηθούν και της υποδομής του συστήματος.

Στην συνέχεια τα Test deliverables για την δραστηριότητα του Test Implementation περιλαμβάνουν την παραγωγή και αλληλουχία των test procedures, την παραγωγή των Test Suites, των Low Level Test Cases και του Test Execution Schedule. Τα Test Suites ορίζονται ως το σύνολο των test scripts ή των test procedures που θα εκτελεστούν σε μια ένα συγκεκριμένο test run ενώ το Test Execution Schedule ορίζεται ως το πρόγραμμα για την εκτέλεση των Test Suites για ένα κύκλο δοκιμών (Test Cycle). Επίσης κατά την ολοκλήρωση του Test Implementation παράγονται τα automated Test Scripts, τα Test Data και δημιουργείται το Test Environment καθώς και τα Test Stubs, Test Drivers και η τεχνική του Service Virtualization (εάν χρειάζεται). Ένα Test Driver ορίζεται ως ένα προσωρινό στοιχείο λογισμικού (component) το οποίο αντικαθιστά ένα άλλο στοιχείο λογισμικού (Component) και ελέγχει ή καλεί ένα test item μεμονωμένα. Ένα Test Stub ορίζεται ως η υλοποίηση ενός Α στοιχείου λογισμικού το οποίο θα χρησιμοποιηθεί για να αναπτυχθεί ή να ελεγχθεί ένα Β στοιχείο λογισμικού το οποίο στοιχείο Β εξαρτάται ή «καλεί» το στοιχείο Α. Ένα Test Stub στην ουσία αντικαθιστά ένα «καλούμενο» στοιχείο [22]. Στο παρακάτω σχήμα παρουσιάζεται σε υψηλό επίπεδο η αλληλεπίδραση μεταξύ ενός Test Driver, ενός component και ενός Test Stub.



Σχήμα 2.4: Αλληλεπίδραση των Driver, Components και Stub

Ένα Driver μπορεί να θεωρηθεί ως μια παράμετρος για ένα στοιχείο λογισμικού καθώς προσομοιώνει ένα στοιχείο λογισμικού A το οποίο «οδηγεί» (drives) τις πληροφορίες σε ένα στοιχείο λογισμικού X. Στην συνέχεια το Stub θα «καταναλώσει» (consumes) την είσοδο του Component X και θα παρέχει πληροφορίες πίσω στο Component X από το οποίο «καλέστηκε».

Το service virtualization αποτελεί μια τεχνική που επιτρέπει την εικονική παράδοση υπηρεσιών οι οποίες μπορούν να αναπτυχθούν, να προσπελαστούν και να διαχειριστούν εξ αποστάσεως από ένα test item [22]. Οι παραπάνω έννοιες (Driver, Stub) αποτελούν τα επιμέρους συστατικά της τεχνικής του service virtualization που περιγράφεται στο σχήμα 2.4.

Μετά την ολοκλήρωση του Test Implementation λαμβάνει μέρος η επόμενη δραστηριότητα του STLC που ονομάζεται Test Execution. Τα test work products που παράγονται μέσω της δραστηριότητας του Test Execution αποτελούν την ενημέρωση της κατάστασης των Test Cases με την αντίστοιχη αντιπροσωπευτική αξία (Passed, Failed, Blocked, Retest, In progress κλπ.), την αναφορά των defects (bugs) που διαπιστώθηκαν στο SUT και την τεκμηρίωση των Test Cases, Test Items, του testware και των εργαλείων που χρησιμοποιήθηκαν για να διεξαχθεί ο έλεγχος.

Κατά την τελευταία δραστηριότητα του STLC που ονομάζεται Test Completion δημιουργείται το Test Summary Report το οποίο περιλαμβάνει πληροφορίες σχετικά με τις ενέργειες, τα αποτελέσματα, τις εναπομείναντες ατέλειες στο λογισμικό και οτιδήποτε άλλο αφορά το project από την οπτική γωνία του ελέγχου. Μερικά ακόμη test work products που μπορούν να προκύψουν από την δραστηριότητα του Test Completion είναι η καταγραφή ενεργειών (action items) και αιτημάτων αλλαγής (Change Requests) οι οποίες μπορούν να βελτιώσουν την διαδικασία του ελέγχου ή ακόμη και την διαδικασία ανάπτυξης του έργου λογισμικού. Τέλος, ένα ακόμη test work product που παράγεται είναι οριστικοποίηση του testware [1].

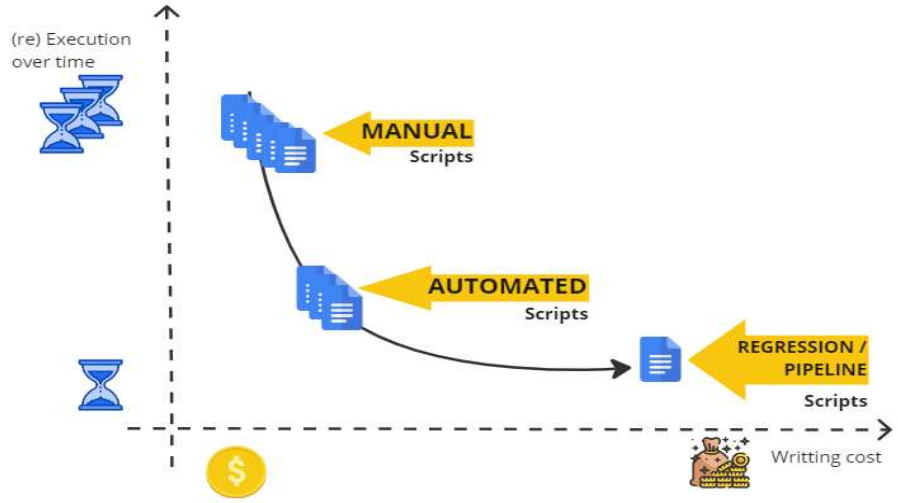
2.12 Manual vs Automation testing

Ο έλεγχος χωρίζεται και μπορεί να διεξαχθεί με δύο διαφορετικούς τρόπους οι οποίοι έχουν τον ίδιο σκοπό, την επίτευξη της ποιότητας του λογισμικού. Ο πρώτος τρόπος ονομάζεται manual testing και ο δεύτερος ονομάζεται automation testing. Το manual testing απαιτεί ανθρώπινη παρέμβαση για την

«χειροκίνητη» εκτέλεση των βημάτων του κάθε test καθώς δεν χρησιμοποιούνται εργαλεία για τον αυτοματισμό της εκτέλεσης. Το automation testing δεν απαιτεί την ανθρώπινη παρέμβαση κατά την εκτέλεση των βημάτων των tests διότι η εκτέλεση αυτή βασίζεται πάνω σε διάφορα εργαλεία και automation testing framework που αυτοματοποιούν την διαδικασία. Στο automation testing, τα βήματα για το κάθε test case, έχουν κωδικοποιηθεί χρησιμοποιώντας μια γλώσσα προγραμματισμού σε ένα ή παραπάνω test scripts και εκτελούνται μέσω ενός εργαλείο αυτοματισμού (automation tool).

Υπάρχουν αρκετές και σημαντικές διαφορές όπως επίσης και αρκετά πλεονεκτήματα και μειονεκτήματα μεταξύ του manual και του automation testing τα οποία πρέπει να ληφθούν υπόψιν από μία επιχείρηση και κατ' επέκταση για κάθε αναπτυσσόμενο έργο λογισμικού και ομάδα. Μία από τις κυριότερες διαφορές είναι ότι το manual testing είναι πιο χρονοβόρο μακροπρόθεσμα από το automation testing. Το manual testing είναι μια καλή βραχυπρόθεσμη λύση για προϊόντα λογισμικού τα οποία δεν είναι περίπλοκα και δεν έχουν αρκετές λειτουργίες και πιθανά user flows, καθώς για αυτές τις περιπτώσεις ο έλεγχος συνήθως δεν έχει μεγάλη διάρκεια. Επίσης το manual testing είναι αρκετά χρήσιμο όταν το λογισμικό βρίσκεται στο πρώιμο στάδιο ανάπτυξης του. Η διεξαγωγή του manual testing συνήθως δεν απαιτεί άτομα με ιδιαίτερες τεχνικές ικανότητες σε αντίθεση με το automation testing.

Δύο ακόμη πολύ σημαντικά κριτήρια τα οποία θα πρέπει να ληφθούν υπόψιν όσον αφορά την επιλογή του manual ή automation testing, είναι ο προϋπολογισμός και ο χρόνος που διατίθενται για έργο λογισμικού το οποίο θα πρέπει να ελεγχθεί. Στο παρακάτω σχήμα παρουσιάζεται μια γραφική παράσταση με άξονες τον χρόνο και το κόστος η οποία συγκρίνει το κόστος και την σπατάλη χρόνου μεταξύ του manual testing, του automation testing και του regression testing.



Σχήμα 2.5: Manual vs automation testing

Το regression testing μπορεί να θεωρηθεί ως η επέκταση του automated testing καθώς αποτελείται από μεγάλα και αυτοματοποιημένα test suites. Τα test suites είναι ένα σύνολο από scripts και από test procedures τα οποία οργανώνονται σε διαφορετικούς φακέλους και εκτελούνται για να ελεγχθεί ένα ή περισσότερα τμήματα του προϊόντος λογισμικού. Το regression testing διεξάγεται για να ελεγχθεί εάν η προσθήκη μια καινούριας λειτουργικότητας στο σύστημα ή η επιδιόρθωση μιας ατέλειας δεν έχει επιφέρει ακούσιες επιπτώσεις στο ίδιο το σύστημα ή σε άλλα συστήματα που αλληλοεπιδρούν με αυτό. Επίσης το regression testing χρησιμοποιείται για να ελεγχθεί εάν μια αλλαγή ή μία ενημέρωση σε μια έκδοση του λογισμικού επιφέρει ακούσιες επιπτώσεις στο λογισμικό. Τέτοιες αλλαγές ή ενημερώσεις περιλαμβάνουν την ενημέρωση ενός λειτουργικού συστήματος, γλώσσας προγραμματισμού, IDE

(Integrated Development Environment), DBMS (Database Management System), κλπ. Όπως παρατηρείται το manual testing απαιτεί λίγο χρόνο και χρήμα βραχυπρόθεσμα, αλλά αρκετό μακροπρόθεσμα και αυτό γιατί κάθε φορά που πρέπει να διεξαχθεί manual testing, τα βήματα ενός test εκτελούνται μέσω της παρέμβασης του ανθρώπου καθώς ή εκτέλεση των βημάτων των manual test scripts δεν είναι αυτοματοποιημένη. Από την άλλη πλευρά, το automation και το regression testing χρειάζονται βραχυπρόθεσμα αρκετό χρόνο προκειμένου να υλοποιηθούν και να συντηρηθούν τα automated και τα regression test scripts, αλλά μακροπρόθεσμα η επανεκτέλεσή τους κοστίζει λιγότερο σε χρόνο και σε κόστος από την εκτέλεση των manual test scripts.

Στον παρακάτω πίνακα αναφέρονται μερικά από τα σημαντικότερα πλεονεκτήματα και μειονεκτήματα των δύο τρόπων διεξαγωγής του ελέγχου [23].

Πίνακας 2.2: Πλεονεκτήματα και μειονεκτήματα του manual και του automation testing

Κριτήρια	**Manual testing**	**Automation testing**
Ακρίβεια	Το manual testing έχει χαμηλότερη ακρίβεια λόγω της πιθανότητας ανθρώπινων σφαλμάτων	Το automation testing έχει υψηλότερη ακρίβεια λόγω του ελέγχου βασισμένου σε υπολογιστή, εξαλείφοντας την πιθανότητα σφαλμάτων
Κλίμακα	Το Manual testing απαιτεί χρόνο όταν ο έλεγχος χρειάζεται να διεξαχθεί σε μεγάλη κλίμακα	Το automation testing εκτελεί εύκολα δοκιμές σε μεγάλη κλίμακα με την μέγιστη δυνατή αποτελεσματικότητα
Χρόνος ολοκλήρωσης	Το Manual testing απαιτεί περισσότερο χρόνο για την ολοκλήρωση ενός test cycle, επομένως ο χρόνος διεκπεραίωσης του ελέγχου είναι υψηλότερος	Το automation testing ολοκληρώνει έναν κύκλο δοκιμών εξαιρετικά γρήγορα. Έτσι, ο χρόνος διεκπεραίωσης του ελέγχου είναι πολύ χαμηλότερος
Απόδοση Κόστους	Το manual testing απαιτεί μεγαλύτερο κόστος καθώς περιλαμβάνει την πρόσληψη και την συνεχή απασχόληση των manual testers	Το automation testing εξοικονομεί το κόστος που αρχικά προέκυψε, από τη στιγμή που ενσωματώνεται η υποδομή του λογισμικού και μένει λειτουργικό για μεγάλο διάστημα με μερική συντήρηση
Εμπειρία χρήστη	Το manual testing εξασφαλίζει μια υψηλού επιπέδου εμπειρία στον τελικό χρήστη του προϊόντος, καθώς απαιτεί ανθρώπινη παρατήρηση και γνωστικές ικανότητες	Το automation testing δεν μπορεί να εγγυηθεί μια καλή εμπειρία χρήστη, καθώς οι υπολογιστές διαθέτουν ακόμη προηγμένη ανθρώπινη παρατήρηση και γνωστικές ικανότητες
Τομείς εξειδίκευσης	Για την παραγωγή των καλύτερων δυνατών αποτελεσμάτων το manual testing πρέπει να χρησιμοποιείται για την διεξαγωγή των exploratory testing, usability testing και ad-hoc testing	Το automation testing πρέπει να χρησιμοποιείται για την διεξαγωγή των regression testing, load testing, performance testing και για λόγους όπου χρειάζεται επαναλαμβανόμενη εκτέλεση των tests.

Δεξιότητες χρηστών του testing	Οι χρήστες του manual testing πρέπει να μπορούν να μιμούνται τη συμπεριφορά των χρηστών και να δημιουργούν test plans ώστε να καλύψουν όλα τα δυνατά σενάρια	Οι χρήστες του automation setting πρέπει να έχουν υψηλή εξειδίκευση στον προγραμματισμό και στο scripting για να δημιουργήσουν test cases και να αυτοματοποιήσουν όσο το δυνατών περισσότερα σενάρια
Εγκαθίδρυση	Η διαδικασία εγκαθίδρυσης του manual testing είναι συνήθως σύντομη ή μηδαμινή	Η διαδικασία εγκαθίδρυσης του automation testing είναι συνήθως χρονοβόρα και πιο περίπλοκη
Test reports	Τα αποτελέσματα του manual testing συνήθως σώζονται και ενημερώνονται σε αρχεία Excel ή σε test case management tools	Τα αποτελέσματα του automation testing συνήθως σώζονται και ενημερώνονται σε VCS (Version Control Systems) ή σε CI/CD tools ή σε Cloud Storage Services

Για την μεγαλύτερη αποτελεσματικότητα του ελέγχου στα προϊόντα λογισμικού, συνίσταται να χρησιμοποιούνται και οι δύο τρόποι διεξαγωγής καθώς η επίτευξης της σωστής ισορροπίας μεταξύ manual και automation testing είναι απαραίτητη [23]. Το automation testing δεν μπορεί να αντικαταστήσει εξ ολοκλήρου το manual testing. Το manual testing είναι χρήσιμο για πολύπλοκα test cases ενώ το automation testing είναι κατάλληλο για απλά και επαναλαμβανόμενα test cases. Το automation testing βοηθάει τους QAE να εκτελέσουν τα test cases πιο γρήγορα και πιο αξιόπιστα, αλλά δεν προβλέπεται ότι θα αντικαταστήσει πλήρως τους ανθρώπους που διεξάγουν τον έλεγχο [24].

Στην πραγματικότητα, ο ένας τρόπος δεν είναι γενικευμένα καλύτερος από τον άλλον, καθώς ο κάθε τρόπος προτιμάται να χρησιμοποιείται για διαφορετικά test types. Ο κάθε τρόπος έρχεται για να κομπλιμεντάρει τον άλλον, καλύπτοντας ότι δεν μπορεί ο άλλος. Συμπερασματικά, δεν είναι σωστό, ούτε βέλτιστο να χρησιμοποιείται μόνο ένας από τους δύο τρόπους. Ο σωστός συνδυασμός των δύο τρόπων ελέγχου απαιτείται και χρησιμοποιείται από τις επιχειρήσεις για τα προϊόντα λογισμικού.

2.13 Test management / Bug tracking

Στον κόσμο του λογισμικού το test management και το bug tracking είναι δύο σημαντικές έννοιες και διαδικασίες που σχετίζονται με την δοκιμή και την εποπτεία του λογισμικού.

Το test management είναι η διαδικασία που αφορά τον σχεδιασμό, την οργάνωση, την εκτέλεση και την παρακολούθηση των test cases. Η διαδικασία αυτή περιλαμβάνει τις ακόλουθες δραστηριότητες με την εξής σειρά:

1. Σχεδιασμός των test cases. Καθορισμός των test scenarios που θα εκτελεστούν για να ελεγχθεί το λογισμικό
2. Δημιουργία των test data. Δημιουργία αναγκαίων δεδομένων που θα χρησιμοποιηθούν κατά την διάρκεια του ελέγχου
3. Εκτέλεση των tests cases. Τα test cases εκτελούνται χειροκίνητα ή αυτοματοποιημένα και τα αποτελέσματά τους καταγράφονται
4. Αναφορά και παρακολούθηση. Τα αποτελέσματα της εκτέλεσης των test cases αναφέρονται και τα εκκρεμή ζητήματα παρακολουθούνται.

Υπάρχουν πάρα πολλά εργαλεία τα οποία χρησιμοποιούνται ώστε να διευκολυνθούν αυτές οι διαδικασίες. Τα περισσότερα από αυτά απαιτούν συνδρομή, ενώ μερικά από αυτά διατίθενται δωρεάν. Συνήθως οι, κυρίαρχες διαφορές μεταξύ αυτών που διατίθενται δωρεάν και αυτών που απαιτούν συνδρομή είναι η ευκολία στη χρήση, η δυνατότητα αυτοματοποίησης της αναφοράς αποτελεσμάτων και η δυνατότητα ενσωμάτωσης με εργαλεία διαχείρισης έργων λογισμικού (π.χ. Jira, Trello, Wrike, ClickUp, Notion) και με εργαλεία που διευκολύνουν την επικοινωνία και την συνεργασία (π.χ. Microsoft Teams, Slack, Google Workspace). Μερικά από τα γνωστότερα και τα πιο χρησιμοποιημένα εργαλεία στην αγορά για την διαχείριση των tests είναι τα:

- Jira (δωρεάν έως 10 χρήστες)
- Testlink (δωρεάν)
- Browser Stack (δωρεάν)
- QaManager (δωρεάν)
- Test Rail (με συνδρομή)
- Xray (με συνδρομή)
- Zephyr (με συνδρομή)
- ALM Octane (με συνδρομή)
- HP ALM (με συνδρομή)
- qTest (με συνδρομή)

Η παρακολούθηση σφαλμάτων, γνωστή και ως «bug tracking» ή «defect tracking», είναι η διαδικασία καταγραφής, παρακολούθησης και διαχείρισης των ανεπιθύμητων συμπεριφορών (ή αλλιώς σφαλμάτων) που εντοπίζονται κατά την διάρκεια των δοκιμών ή μετά την κυκλοφορία του λογισμικού. Θα μπορούσε να θεωρηθεί ως κομμάτι της ευρύτερης διαδικασίας του test management μιας και συνδέεται με την τελευταία δραστηριότητα του. Η παρακολούθηση των σφαλμάτων είναι σημαντική για τη βελτίωση της ποιότητας του λογισμικού και για τη διατήρηση ενός αποτελεσματικού κύκλου ανάπτυξης (SDLC).

Η καταγραφή, η παρακολούθηση και η διαχείριση των σφαλμάτων διευκολύνεται μέσω της χρήσης των αντίστοιχων εργαλείων. Όπως και για τη διαχείριση των test, έτσι και για την παρακολούθηση σφαλμάτων, τα αντίστοιχα εργαλεία διατίθενται δωρεάν ή με συνδρομή. Οι κυρίαρχες διαφορές των εργαλείων παρακολούθησης σφαλμάτων είναι ή διεπαφή χρήστη και η ευκολία χρήσης τους και η δυνατότητα προσαρμογής και ευελιξίας. Μερικές άλλες διαφορές είναι η δυνατότητα ενσωμάτωση με άλλα εργαλεία επικοινωνίας, συνεργασίας και ανάπτυξης καθώς και η δυνατότητα επεκτασιμότητας σχετικά με τους χρήστες, τις ομάδες και περίπλοκων workflows. Παρακάτω αναφέρονται μερικά από τα δημοφιλέστερα και από τα πιο χρησιμοποιημένα εργαλεία παρακολούθησης σφαλμάτων στην αγορά τα οποία χρησιμοποιούνται ευρέως για τα έργα λογισμικού:

- Jira (δωρεάν έως 10 χρήστες)
- Bugzilla (δωρεάν)
- Mantis Bug Tracker (δωρεάν)
- YouTrack (δωρεάν μέχρι 10 χρήστες)
- Readmine (δωρεάν)
- HP ALM (με συνδρομή)
- Zendesk (με συνδρομή)
- Test Rail (με συνδρομή)
- Xray (με συνδρομή)
- Zephyr (με συνδρομή)

Αξίζει να αναφερθεί ότι, τα τελευταία χρόνια επικρατεί μια τάση ραγδαίας εξέλιξης των test management και των bug tracking tools. Τα περισσότερα test management tools πλέον διαθέτουν

δυνατότητες και για το bug tracking και αντιστρόφως. Έτσι, κυρίως τα γνωστότερα εργαλεία, χρησιμοποιούνται και για το test management αλλά και για το bug monitoring. Αυτή η εξέλιξη έχει στρέψει τις περισσότερες επιχειρήσεις και οργανισμούς να χρησιμοποιούν ένα κοινό εργαλείο και για την διαχείριση των test αλλά και για την παρακολούθηση σφαλμάτων. Επίσης η εξέλιξη αυτή δικαιολογεί και τον λόγο όπου τα ίδια εργαλεία αναφέρονται και στις δύο παραπάνω λίστες (δημοφιλέστερα εργαλεία για test management και bug tracking).

2.14 Severity vs Priority

Η σοβαρότητα (severity) και προτεραιότητα (priority) είναι δύο βασικές έννοιες και χαρακτηριστικά ενός ελαττώματος (defect) το οποίο μπορεί να αναφερθεί κατά την διάρκεια του ελέγχου ή της συντήρησης ενός προϊόντος λογισμικού. Αυτές οι δύο παράμετροι βοηθούν τις ομάδες ανάπτυξης (συμπεριλαμβάνοντας και της ομάδες του ελέγχου) να προσδιορίσουν τον βαθμό για το πόσο σημαντικό και για το πόσο επείγον είναι να επιλυθεί ένα ελάττωμα στο λογισμικό. Σύμφωνα με τον ορισμό που δίνεται από το ISTQB, η σοβαρότητα είναι ο βαθμός επίπτωσης ενός ελαττώματος στην ανάπτυξη ή λειτουργία ενός στοιχείου (component) ή ενός συστήματος (system) λογισμικού. Επίσης ο ορισμός που δίνεται για την προτεραιότητα είναι το επίπεδο σημασίας (από την οπτική της επιχείρησης) που αποδίδεται σε ένα αντικείμενο (π.χ. ελάττωμα) [22].

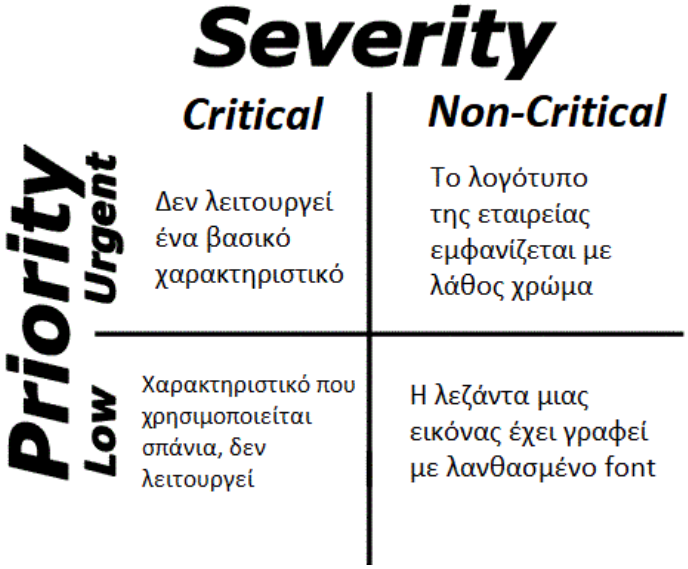
Η σοβαρότητα ενός ελαττώματος περιέχει τις παρακάτω τυπικές κατηγορίες από αύξων σε φθίνων βαθμό:

1. Critical → Ως «critical» κατηγοριοποιείται ένα ελάττωμα το οποίο οδηγεί στον τερματισμό του πλήρους συστήματος ή ενός ή περισσοτέρων στοιχείων (components) του συστήματος. Η αποτυχημένη λειτουργία δεν μπορεί να χρησιμοποιηθεί και δεν υπάρχει αποδεκτή εναλλακτική μέθοδος για την επίτευξη των απαιτούμενων αποτελεσμάτων.
2. High → Ως «high» κατηγοριοποιείται ένα ελάττωμα το οποίο οδηγεί στον τερματισμό του πλήρους συστήματος ή ενός ή περισσοτέρων στοιχείων (components) του συστήματος αλλά υπάρχει κάποια αποδεκτή μέθοδος για την επίτευξη των απαιτούμενων αποτελεσμάτων.
3. Medium/Moderate → Ως «medium» ή «moderate» κατηγοριοποιείται ένα ελάττωμα το οποίο δεν οδηγεί στον τερματισμό του συστήματος, αλλά επηρεάζει το σύστημα με συνέπεια αυτό να παράγει εσφαλμένα, ελλιπή ή ασυνεπή αποτελέσματα.
4. Low → Ως «low» κατηγοριοποιείται ένα ελάττωμα το οποίο δεν οδηγεί στον τερματισμό του συστήματος και δεν βλάπτει τη χρηστικότητα του εφόσον τα επιθυμητά αποτελέσματα μπορούν να επιτευχθούν, εύκολα, με κάποιον εναλλακτικό τρόπο [25].
5. Cosmetic/Non-critical → Ως «cosmetic» ή «non-critical» κατηγοριοποιείται ένα ελάττωμα το οποίο σχετίζεται με την βελτίωση του συστήματος και όπου συνήθως οι αλλαγές σχετίζονται με την εμφάνιση της εφαρμογής (στο UI).

Η προτεραιότητα ενός ελαττώματος περιέχει τις παρακάτω τυπικές κατηγορίες από αύξων σε φθίνων βαθμό:

1. High/Urgent → Ως «high» ή «urgent» κατατάσσεται ένα ελάττωμα το οποίο πρέπει να επιλυθεί το συντομότερο δυνατό, επειδή επηρεάζει σημαντικά το προϊόν ή την επιχείρηση. Το σύστημα συνήθως δεν μπορεί να χρησιμοποιηθεί μέχρι να ολοκληρωθεί η επισκευή.
2. Medium → Ως «medium» κατατάσσεται ένα ελάττωμα το οποίο πρέπει να επιλυθεί κατά την κανονική πορεία των δραστηριοτήτων της ανάπτυξης του λογισμικού. Επίσης η επιδιόρθωση ενός τέτοιου ελαττώματος μπορεί να παραμείνει μέχρι να δημιουργηθεί μια νέα έκδοση ή build του λογισμικού.
3. Low → Ως «low» κατατάσσεται ένα ελάττωμα το οποίο πρέπει να επισκευαστεί, αλλά η επισκευή του μπορεί να αναβληθεί έως ότου να επιδιορθωθούν ελαττώματα τα οποία έχουν υψηλότερη προτεραιότητα ή θεωρούνται πιο σημαντικά [25].

Συχνά έχει παρατηρηθεί ότι, κυρίως για τους αρχάριους QAE, προκαλείται σύγχυση μεταξύ των εννοιών της σοβαρότητας και της προτεραιότητας ενός σφάλματος/ελαττώματος. Στο παρακάτω σχήμα, αναφέρονται μερικά παραδείγματα για την σοβαρότητα και για την προτεραιότητα πιθανών σφαλμάτων για ένα προϊόν λογισμικού [26].



Σχήμα 2.6: Severity vs priority

Η προτεραιότητα καθορίζει ποιο ελάττωμα πρέπει να επιδιορθωθεί πρώτο. Τι γίνεται όμως σε περίπτωση που δύο ελαττώματα έχουν τον ίδιο βαθμό προτεραιότητας; Σε αυτή την περίπτωση, το δεύτερο κριτήριο που θα ληφθεί υπόψιν για το ποιο ελάττωμα θα επιδιορθωθεί πρώτο θα είναι αυτό που έχει τον υψηλότερο βαθμό σοβαρότητας. Σε εξαιρετικά σπάνιες περιπτώσεις μπορεί δύο διαφορετικά ελαττώματα να έχουν ακριβώς τον ίδιο βαθμό προτεραιότητας και σοβαρότητας. Τότε η επιλογή του ελαττώματος που θα πρέπει να επιδιορθωθεί πρώτο αφήνεται στην κρίση του QAE ή αποφασίζεται μέσω συνεννόησης μεταξύ των μελών της ευρύτερης ομάδας (π.χ. scrum team).

2.15 Functional vs Non-functional requirements

Οι απαιτήσεις που μπορεί να έχουν οι χρήστες ενός προϊόντος λογισμικού χωρίζονται σε 2 κατηγορίες. Η πρώτη κατηγορία ονομάζεται «λειτουργικές απαιτήσεις» (Functional requirements) και με αυτές ορίζεται το τι πρέπει να κάνει το σύστημα / λογισμικό, ποια είναι τα οφέλη του και ποιες είναι οι από άκρη σε άκρη (End to End) λειτουργίες του. Στην ουσία, είναι οι απαιτήσεις που ορίζονται από τον τελικό χρήστη και είναι ορατές στο τελικό προϊόν. Η δεύτερη κατηγορία ονομάζεται «μη-λειτουργικές απαιτήσεις» (Non-functional requirements) και με αυτές ορίζονται οι ποιοτικοί περιορισμοί που πρέπει να ικανοποιεί το σύστημα / λογισμικό σύμφωνα με τις προδιαγραφές. Οι Non-functional requirements περιγράφουν τις γενικές ιδιότητες ή αλλιώς τα χαρακτηριστικά ποιότητας του συστήματος / λογισμικού. Μερικά από τα βασικότερα από χαρακτηριστικά της ποιότητας για ένα σύστημα / λογισμικό μπορεί να είναι η απόδοση (performance), η ασφάλεια (security), η ευχρηστία (usability), η αξιοπιστία (reliability), η επεκτασιμότητα (scalability), η επαναχρησιμοποίηση (reusability), κλπ. [27]. Οι Functional requirements είναι επίσης γνωστές με τον όρο «λειτουργική συμπεριφορά (functional behavior)», ενώ οι Non-functional requirements ορίζονται και ως «μη λειτουργική συμπεριφορά (non-

functional behavior)». Όπως είναι λογικό, οι Functional requirements επαληθεύονται μέσω της διεξαγωγής του functional testing, ενώ οι Non-functional requirements επαληθεύονται μέσω της διεξαγωγής του Non-functional testing.

2.16 Test Levels

Τα επίπεδα του ελέγχου (Test Levels) αποτελούνται από δραστηριότητες ελέγχου (Test Activities) οι οποίες οργανώνονται και χρησιμοποιούνται ως ξεχωριστό σύνολο για το κάθε επίπεδο. Το κάθε επίπεδο είναι ένα στιγμιότυπο της διαδικασίας του ελέγχου που διεξάγεται σε ένα λογισμικό το οποίο βρίσκεται σε κάποιο στάδιο της ανάπτυξης [1]. Το στάδιο ανάπτυξης μπορεί να είναι πρώιμο, οπότε το λογισμικό έχει αποκλειστικά την μορφή μεμονωμένων στοιχείων ή πιο εξελιγμένο άρα το λογισμικό να έχει την μορφή ενός συστήματος ή ενός ευρύτερου συστήματος από συστήματα (system of systems). Τα επίπεδα του ελέγχου είναι τα εξής:

1. Component testing (γνωστό και ως «Unit testing» ή «Module testing»)
2. Integration testing
3. System testing
4. Acceptance testing

Το Component testing πρόκειται για τον έλεγχο ενός μεμονωμένου στοιχείου (π.χ. μεθόδου / συνάρτησης). Ο έλεγχος σε αυτό το επίπεδο γίνεται για το κάθε ξεχωριστό component του λογισμικού σε απομόνωση — δηλαδή χωρίς να υπάρχει καμία αλληλεπίδραση (interaction) από εξωτερικούς παράγοντες (π.χ. άλλες μεθόδους, συναρτήσεις, κλπ.). Το Component testing συνήθως διεξάγεται από την ομάδα ανάπτυξης (developers) του λογισμικού στο τοπικό τους περιβάλλον (local environment). Τα αντικείμενα του ελέγχου (test objects) στο επίπεδο των μεμονωμένων συστατικών του λογισμικού μπορούν να είναι κλάσεις, δομές δεδομένων, μέθοδοι, components βάσεων δεδομένων, κλπ. Τα πιο τυπικά bugs που προκύπτουν σε αυτό το επίπεδο είναι η λανθασμένη λειτουργικότητα και λογική, ο λανθασμένος κώδικας και προβλήματα ροής δεδομένων.

Ο έλεγχος ενσωμάτωσης (Integration testing) αποτελεί το δεύτερο επίπεδο του ελέγχου λογισμικού και μπορεί να χωριστεί σε δύο κατηγορίες. Η πρώτη κατηγορία είναι το «Component integration testing» και η δεύτερη κατηγορία είναι το «System integration testing». Με το Component integration testing διασφαλίζεται η ενσωμάτωση / σύνδεση των components, ελέγχοντας τις διεπαφές (interfaces), την σύνδεση και τις αλληλεπιδράσεις μεταξύ των components. Με το System integration testing διασφαλίζεται η ενσωμάτωση / σύνδεση των συστημάτων, ελέγχοντας τις διεπαφές (interfaces), την σύνδεση και τις αλληλεπιδράσεις μεταξύ των συστημάτων. Ως συστήματα μπορούν επίσης να εννοηθούν τα packages και τα microservices. Στο πλαίσιο του System integration testing μπορούν επίσης να ελεγχθούν οι συνδέσεις, οι διεπαφές και οι αλληλεπιδράσεις ενός συστήματος με άλλους οργανισμούς μέσω εξωτερικών συνδέσεων (π.χ. Web Services). Το Component integration testing πραγματοποιείται από την ομάδα ανάπτυξης (developers), ενώ το System integration testing διεξάγεται από την ομάδα του ελέγχου (QAE). Τα αντικείμενα του ελέγχου στο επίπεδο της ενσωμάτωσης (component με component και συστήματος με σύστημα) του λογισμικού μπορούν να είναι interfaces, APIs (Application Programming Interfaces), microservices, databases, infrastructure κλπ.

Το System testing αποτελεί το τρίτο επίπεδο του ελέγχου λογισμικού και επικεντρώνεται στον έλεγχο της συνολικής συμπεριφοράς και τις δυνατότητες ενός ολόκληρου συστήματος ή προϊόντος. Σε αυτό το επίπεδο, δοκιμάζονται οι λειτουργίες και η απόδοση του συστήματος, οι οποίες βασίζονται πάνω σε συγκεκριμένες προδιαγραφές και απαιτήσεις. Πολύ συχνά, οι επιχειρήσεις και οι οργανισμοί χρησιμοποιούν για το Non-functional testing, ένα ξεχωριστό περιβάλλον (που είναι απομίμηση του περιβάλλοντος παραγωγής – production environment), το οποίο έχει αντιπροσωπευτικό όνομα για το

κάθε χαρακτηριστικό ποιότητας που ελέγχεται (π.χ. Performance environment, Usability environment, κλπ.). Το System testing εκτελείται από την ομάδα ελέγχου (QAE) του scrum team ή από κάποια άλλη ανεξάρτητη ομάδα ελέγχου. Τα αντικείμενα του ελέγχου σε αυτό το επίπεδο μπορούν να είναι μια εφαρμογή, συστήματα λογισμικού ή υλισμικού (γνωστά και ως SUT), η διαμόρφωση του συστήματος και δεδομένα διαμόρφωσης.

Ο έλεγχος αποδοχής (Acceptance testing) αποτελεί το τελευταίο επίπεδο του ελέγχου. Ο έλεγχος επικεντρώνεται στην επικύρωση και στην απόδειξη ετοιμότητας του λογισμικού προκειμένου αυτό να προχωρήσει στο στάδιο του deployment, πράγμα που σημαίνει ότι το σύστημα ικανοποιεί τις επιχειρηματικές ανάγκες του τελικού χρήστη. Σε αυτό το επίπεδο, το λογισμικό πλέον έχει πάρει την μορφή ενός ολοκληρωμένου προϊόντος ή αλλιώς ενός ευρύτερου συστήματος που με την σειρά του αποτελείται από άλλα συστήματα (system of systems). Στην ιδανική περίπτωση ο έλεγχος αποδοχής πρέπει να εκτελείται σκόπιμα και από χρήστες. Ο έλεγχος αποδοχής κυρίως διεξάγεται από Product Owners, πελάτες, χρήστες της επιχείρησης και άλλους ενδιαφερόμενους. Επίσης, στον έλεγχο αποδοχής μπορεί να χρειαστεί έλεγχος όσον αφορά συμβατικές (contractual) και κανονιστικές (regulatory) απαιτήσεις. Οι κύριες μορφές του Acceptance testing είναι οι εξής:

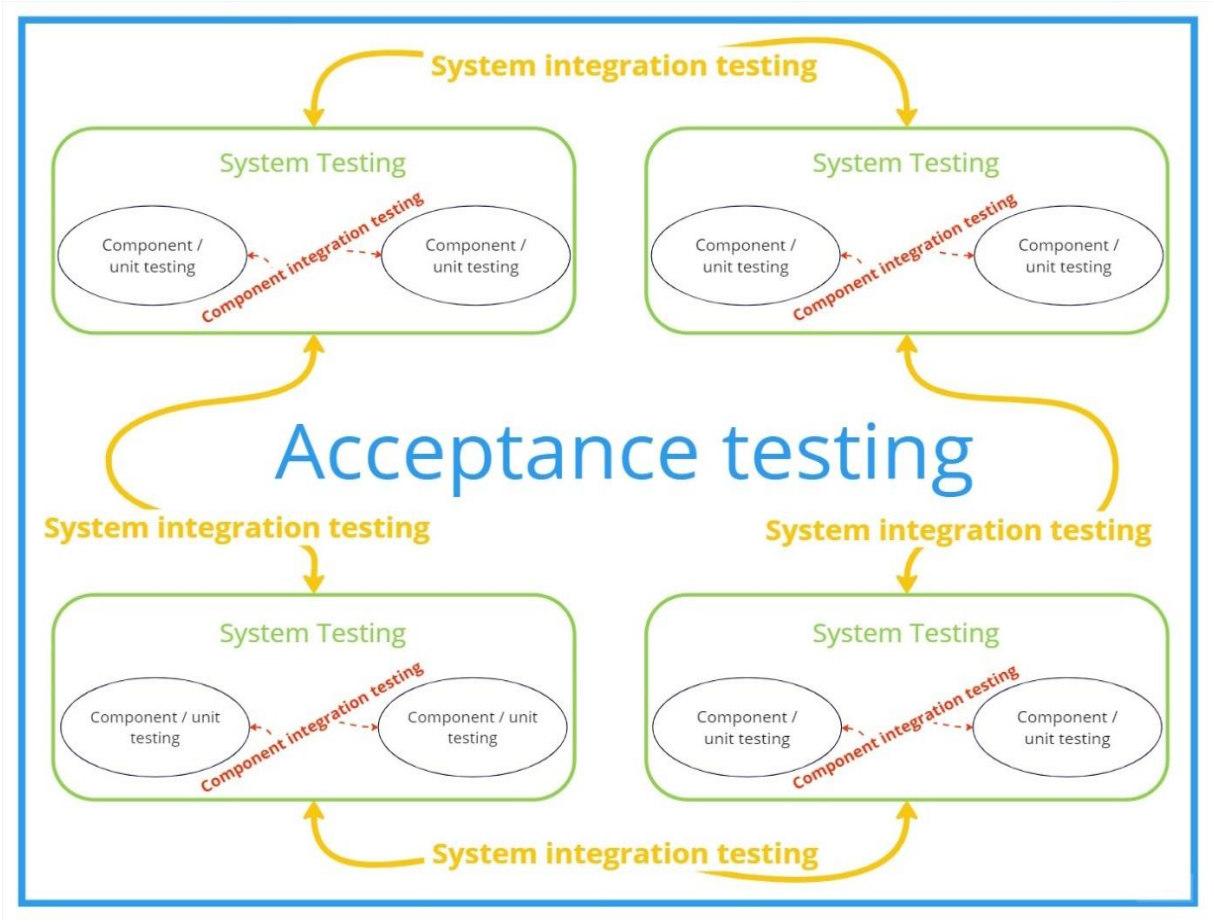
- UAT (User Acceptance Testing)
- Operational acceptance testing
- Contractual acceptance testing και Regulatory acceptance testing
- Alpha testing και Beta testing [28]

Τα αντικείμενα του ελέγχου σε αυτό το επίπεδο μπορούν να είναι ένα ολόκληρο προϊόν το οποίο αντιπροσωπεύεται από ένα σύστημα συστημάτων λογισμικού, επιχειρησιακές διαδικασίες για ένα πλήρως ολοκληρωμένο σύστημα, διαδικασίες λειτουργίας και συντήρησης, έντυπα, αναφορές και συστήματα ανάκτησης.

Οι βασικότεροι στόχοι του ελέγχου (test objectives) για το κάθε επίπεδο είναι ίδιοι με την μόνη αντίθεση ότι ο έλεγχος επικεντρώνεται σε κάποιο διαφορετικό αντικείμενο (test object). Οι κυρίαρχοι στόχοι του ελέγχου για κάθε επίπεδο είναι οι εξής:

- Αποτροπή της διαρροής σφαλμάτων (bug leakage) στα επόμενα επίπεδα και κυρίως στο production (ακολουθώντας πάντα την τακτική του shift left).
- Μείωση του ρίσκου αποτυχίας λειτουργικότητας ή αδυναμίας απόδοσης των test objects.
- Επιβεβαίωση(validation) για το εάν το σύστημα είναι πλήρες και ότι θα λειτουργήσει όπως αναμένεται και η επαλήθευση (verification) για το εάν η functional και η non-functional συμπεριφορά του συστήματος δουλεύει όπως έχει σχεδιαστεί και προσδιοριστεί.
- Η απόκτηση αυτοπεποίθησης από την πλευρά της ομάδας (scrum team) για την ποιότητα των test objects έτσι ώστε ο έλεγχος να διεξαχθεί στο αμέσως επόμενο επίπεδο [1].

Στο παρακάτω σχήμα αναπαρίστανται τα επίπεδα του ελέγχου.



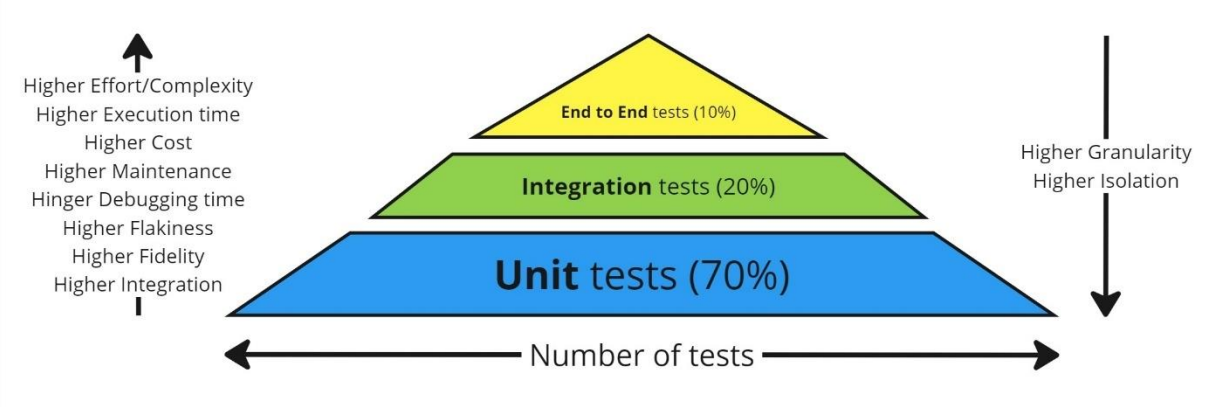
Σχήμα 2.7: Test Levels

Το κάθε διαφορετικό επίπεδο του ελέγχου έχει διαφορετικό χρώμα. Τα διαφορετικά σχήματα της εικόνας αναφέρονται σε μια οντότητα του λογισμικού (μεμονωμένο στοιχείο, σύστημα και προϊόν / σύστημα συστημάτων). Το component/unit testing, το system testing και το acceptance testing διεξάγονται μέσα στα παρακάτω τρία σχήματα. Οι διαφορετικές συνδέσεις / ζεύξεις μεταξύ δύο components όπως επίσης και μεταξύ δύο συστημάτων αναφέρονται στην ενσωμάτωση (integration) των οντοτήτων του λογισμικού. Το component integration testing διεξάγεται στις συνδέσεις μεταξύ component με component, ενώ το system integration testing διεξάγεται στις συνδέσεις μεταξύ συστήματος με σύστημα.

2.17 Test Pyramid

Η πυραμίδα του ελέγχου είναι μια έννοια η οποία γνωστοποιήθηκε το 2009 μέσω της έκδοσης του βιβλίου «Succeeding with Agile (2009)» το οποίο γράφτηκε από ένα ιδρυτικό μέλος της Agile Alliance και της Scrum Alliance, τον Mike Cohn [29]. Το Test Pyramid αποτελεί μια προσέγγιση του διαμερισμού των automated tests στα Test Levels, η οποία μέχρι και σήμερα θεωρείται ως η καθιερωμένη λύση για τα περισσότερα project λογισμικού. Ως βάση της πυραμίδας επιλέγονται τα Unit tests, στην μέση τα Integration tests και στην κορυφή της πυραμίδας τα End to End tests [30]. Τα automated tests έχουν αρκετά χαρακτηριστικά τα οποία λαμβάνονται σοβαρά υπόψιν από τους μηχανικούς ελέγχου μιας ομάδας προκειμένου να γίνει η αυτοματοποίηση με τον βέλτιστο δυνατό τρόπο. Οι τιμές αυτών των χαρακτηριστικών (π.χ. ταχύτητα, κόστος, πολυπλοκότητα, συντήρηση, κλπ.) είναι υψηλότερες ή χαμηλότερες για κάθε διαφορετικό επίπεδο της πυραμίδας. Στο παρακάτω σχήμα

απεικονίζεται η πυραμίδα του ελέγχου, καθώς και η αύξηση ή η μείωση της τιμής για τα βασικότερα χαρακτηριστικά των automated tests από επίπεδο σε επίπεδο [30].



Σχήμα 2.8: Test Pyramid

Τα End to End tests έχουν υψηλότερο κόστος και είναι πιο χρονοβόρα και περίπλοκα από τα Integration tests. Επίσης, η αστάθεια, αδυναμία πρόβλεψης και η χαμηλή αξιοπιστία των αποτελεσμάτων (Flakiness) των End to End tests είναι πιο υψηλές σε σύγκριση με αυτή των Unit tests ενώ τα Unit tests έχουν χαμηλότερη ικανότητας αντιπροσώπευσης της λειτουργικότητας του λογισμικού (Fidelity) από τα End to End tests. Ο βαθμός λεπτομέρειας (Granularity) και ο βαθμός απομόνωσης (Isolation) των Unit tests είναι υψηλότερος σε σύγκριση με τους αντίστοιχους βαθμούς για τα Integration tests και τα End to End tests. Όσο ψηλότερα στην πυραμίδα βρίσκεται ένα επίπεδο, τόσο πιο εκτενείς και λιγότερες θεωρητικά πρέπει να είναι οι δοκιμές (tests) για τα περισσότερα projects.

Το Απρίλη του 2015, η Google δημοσίευσε στο σχετικό Blog της για το testing την εξής ιδέα/πρόταση για την διαίρεση των δοκιμών, την οποία θεωρεί πως πρέπει να είναι και η κατευθυντήρια γραμμή. Το 70% των δοκιμών πρέπει να είναι Unit tests, το 20% πρέπει να είναι Integration tests και το υπόλοιπο 10% End to End tests. Το ακριβές ποσοστό μπορεί να διαφέρει, αναλόγως το λογισμικό και την ομάδα, αλλά γενικότερα θα πρέπει να διατηρείται το σχήμα της πυραμίδας και θα πρέπει να αποφεύγονται μοτίβα όπως αυτά της αναστραμμένης πυραμίδας (πολλά End to End tests, λίγα Integration tests και ακόμη λιγότερα Unit tests) και της κλεψύδρας (πολλά End to End tests, λίγα Integration tests και πολλά Unit tests). Η Google με αυτή τη δημοσίευση αναφέρει μερικά από τα σημαντικότερα γεγονότα και επιχειρήματα για την παραπάνω επιλογής της. Μερικά από αυτά αναφέρονται παρακάτω:

- Τα Unit tests, σε σύγκριση με τα υπόλοιπα, είναι ιδανικά για την αξιόπιστη διεξαγωγή και γρήγορη ανατροφοδότηση των αποτελεσμάτων των tests στους ενδιαφερόμενους.
- Τα Unit tests μπορούν να αναδείξουν με ακρίβεια και επάρκεια το σημείο του σφάλματος στον κώδικα.
- Τα End to End tests έχουν μεγαλύτερη πιθανότητα να αποτύχουν καθώς αντιπροσωπεύονται από σενάρια τα οποία έχουν μεγάλη εξάρτηση από προαπαιτούμενα βήματα - ενέργειες (prerequisites). Αυτά τα βήματα – ενέργειές βρίσκονται συχνά στην αρχή του σεναρίου (π.χ log in). Εάν ένα από αυτά τα βήματα αποτύχει τότε ο έλεγχος για ένα feature που βρίσκεται σε επόμενο στάδιο της εφαρμογής (π.χ. add to cart), δεν θα πραγματοποιηθεί και το test θα αναφέρεται ως failed χωρίς όμως να έχει εξεταστεί το αν λειτουργεί το feature [31].

Όπως διαπιστώνεται, υπάρχουν μερικοί καλοί λόγοι για την εφαρμογή της προσέγγισης της Google. Παρόλα αυτά, δεν συμφωνούν όλοι με αυτή την ιδέα καθώς τα εργαλεία ανάπτυξης των E2E tests, Integration tests και η χρήση εργαλείων CI/CD έχουν εξελιχθεί σημαντικά από τότε (2015) [32]. Το τελικό συμπέρασμα για την επιλογή του κατάλληλου διαμοιρασμού των tests, πρέπει να διαμορφώνεται

ανάλογα με τα διάφορα χαρακτηριστικά του project (κόστος, ανθρώπινο δυναμικό, προδιαγραφές, προθεσμίες, κλπ.).

2.18 Test Types

Υπάρχουν πολλοί διαφορετικοί τύποι ελέγχου (Test Types) οι οποίοι μπορούν να διεξαχθούν σε ένα έργο λογισμικού. Στην συνέχεια αναλύονται οι σημαντικότεροι τύποι ελέγχου οι οποίοι είναι ο λειτουργικός έλεγχος (functional testing) και ο μη λειτουργικός έλεγχος (non-functional testing). Το functional testing συνδέεται άμεσα με την έννοια των functional requirements, ενώ το non-functional testing συνδέεται με την έννοια των non-functional requirements όπως προαναφέρθηκαν στο υποκεφάλαιο 2.14.

Μέσω του functional testing αξιολογούνται οι λειτουργίες που πρέπει να εκτελεί το test object και μέσω αυτού του τύπου ελέγχου διασφαλίζονται οι από άκρη σε άκρη (End to End) λειτουργίες του component ή του συστήματος. Με το functional testing απαντάται η εξής ερώτηση: «τι πρέπει να κάνει το test object;». Οι κύριοι στόχοι του functional testing είναι ο έλεγχος της λειτουργικότητας, η ορθότητα των λειτουργιών και το ποσό κατάλληλες είναι αυτές οι λειτουργίες. Ο τύπος του functional testing περιλαμβάνει αρκετούς υποτύπους (subtypes) εκ των οποίων, οι κυριότεροι αναφέρονται παρακάτω:

- End to End testing: Ο έλεγχος από άκρη σε άκρη (End to End) είναι ένας υποτύπος του functional testing με τον οποίο επαληθεύεται ολόκληρη την εφαρμογή λογισμικού από την αρχή μέχρι το τέλος συμπεριλαμβανομένων όλων των components, συστημάτων και ενσωματώσεων που εμπλέκονται σε αυτή [33]. Ο επαλήθευση αυτής της αρμονικής συνεργασίας των τμημάτων του λογισμικού, διασφαλίζει την επίτευξη των επιδιωκόμενων επιχειρησιακών στόχων.
- UI testing: Ο έλεγχος για της διεπαφής χρήστη (UI) της εφαρμογής επικεντρώνεται στην επαλήθευση της ομαλής λειτουργίας των συστατικών του UI (buttons, textboxes, menu bars, dropdown lists, κλπ.), ώστε να διασφαλιστεί η χρηστικότητα της εφαρμογής και η συνολική ικανοποίηση των χρηστών. Το UI testing μπορεί επίσης να περιλαμβάνει τον έλεγχο της πλοήγησης (navigation) μιας εφαρμογής. Ο στόχος του UI testing είναι να επαληθεύσει ότι οι χρήστες μπορούν να έχουν εύκολη πρόσβαση σε διαφορετικές λειτουργίες (features) της εφαρμογής, χωρίς να αντιμετωπίζουν προβλήματα ή να «χαθούν» στην εφαρμογή.
- API testing: Το API testing περιλαμβάνει τον έλεγχο των αλληλεπιδράσεων και της ανταλλαγής δεδομένων μεταξύ των στοιχείων λογισμικού ή συστημάτων μέσω των APIs. Διασφαλίζει ότι τα APIs εκτελούν σωστά τις προβλεπόμενες λειτουργίες τους και ότι παράγονται τα προβλεπόμενα αποτελέσματα. Επίσης επικεντρώνεται στον έλεγχο της συμπεριφοράς του request-response, στην διαχείριση σφαλμάτων (error handling), στη μορφή των δεδομένων (data format) και στη συμμόρφωση του API με την τεκμηρίωση του API (API documentation).
- Database testing: Το Database testing περιλαμβάνει την επαλήθευση της ακρίβειας (accuracy), της ακεραιότητας (integrity) και της απόδοσης (performance) των δεδομένων που είναι αποθηκευμένα σε μια βάση δεδομένων. Διασφαλίζει ότι τα δεδομένα εισάγονται, ενημερώνονται, ανακτώνται και διαγράφονται σωστά και ότι οι λειτουργίες των βάσεων δεδομένων λειτουργούν όπως αναμένεται.
- Smoke testing: Το Smoke testing αποτελεί μια μορφή γρήγορου ελέγχου. Με αυτόν τον υποτύπο (subtype) του ελέγχου επικυρώνονται οι βασικές και κρίσιμες λειτουργίες του λογισμικού. Ο στόχος του είναι να προσδιορίσει αν οι κύριες λειτουργίες του λογισμικού λειτουργούν όπως αναμένεται μετά από μια νέα έκδοση του λογισμικού (νέο build) ή μετά από ένα νέο release με στόχο να αποφασιστεί αν το build ή το release θεωρείται έτοιμο ώστε να διεξαχθεί περαιτέρω έλεγχος. Αυτός ο υποτύπος ελέγχου καθορίζει εάν μια νέα έκδοση (build) του λογισμικού είναι αρκετά σταθερή για περαιτέρω και πιο εκτενή έλεγχο.
- Sanity testing: Το Sanity testing αποτελεί επίσης μια μορφή γρήγορου ελέγχου. Με την διεξαγωγή αυτού του υποτύπου ελέγχου επαληθεύεται, ότι τα στοιχεία του λογισμικού ή συγκεκριμένα τμήματα του λογισμικού δεν έχουν επηρεαστεί από μικρές αλλαγές ή διορθώσεις. Το Sanity testing έχει ως στόχο να διασφαλιστεί ότι μετά από αλλαγές ή διορθώσεις στο

λογισμικό, οι βασικές λειτουργίες ενός συγκεκριμένου μέρους της εφαρμογής, παραμένουν χωρίς σφάλματα – ανεπηρέαστες. Το Sanity testing μπορεί να θεωρηθεί ως ένα μικρό υποσύνολο του Regression testing. Επίσης η κυρίως διαφορά του Smoke testing και του Sanity testing είναι το scope. Το Smoke testing καλύπτει τις βασικότερες λειτουργίες ολόκληρου του συστήματος ή της εφαρμογής, ενώ το Sanity testing καλύπτει στοχευμένα συγκεκριμένες λειτουργίες ή components.

Μέσω του non-functional testing αξιολογούνται οι ιδιότητες ή αλλιώς τα χαρακτηριστικά της ποιότητας του test object. Το Non-functional testing δεν σχετίζεται με τον έλεγχο των λειτουργιών του test object και μέσω αυτού του τύπου ελέγχου απαντάται η ερώτηση «πόσο καλά συμπεριφέρεται και αποδίδει το test object;». Παρακάτω αναφέρονται κάποιοι από τους βασικότερους τύπους του non-functional testing:

- Performance testing: Ο έλεγχος της απόδοσης ενός προϊόντος λογισμικού αποτελεί έναν από τους πιο κοινώς ασκούμενους υποτύπους του non-functional testing ο οποίος με την σειρά του περιέχει αρκετούς ακόμη υποτύπους. Το Performance testing συνήθως διεξάγεται μέσω προσομοιώσεων των αιτήσεων (requests) των APIs και μέσω αυτού ανακαλύπτονται bugs τα οποία σχετίζονται με τον φόρτο (load) στο δίκτυο της εφαρμογής. Μερικοί από τους γνωστότερους υποτύπους του Performance testing είναι οι εξής:
    - Load testing: Με αυτόν τον υποτύπο ελέγχου η απόδοση της εφαρμογής μετριέται υπό την έγχυση (injection μέσω request) προσομοιωμένων φορτίων (loads) χρηστών. Ο στόχος είναι να προσδιοριστεί πως συμπεριφέρεται η εφαρμογή κάτω από τα αναμενόμενα και τα μέγιστα (peak) φορτία χρηστών ώστε να εντοπιστούν σημεία συμφόρησης και πιθανών ζητημάτων απόδοσης.
    - Stress testing: Το Stress testing αποτελεί την ακραία μορφή του Load testing πιέζοντας την εφαρμογή στα όριά της, συχνά περαιτέρω από την αναμενόμενη χωρητικότητα της. Στόχος είναι να προσδιοριστεί το σημείο θραύσης (breaking point) της εφαρμογής και να καθοριστεί η ικανότητά της στην διαχείριση απροσδόκητων αιχμών (spikes) στην κυκλοφορία (traffic) των δεδομένων [34].
    - Spike testing: Το Spike testing επικεντρώνεται σε ξαφνικές και ακραίες αυξήσεις του φορτίου (load) χρηστών. Στοχεύει στην αξιολόγηση της απόδοσης της εφαρμογής όταν υπάρχει μία ξαφνική αύξηση της κυκλοφορίας ή ένας σημαντικός αριθμός χρηστών έχει πρόσβαση ταυτόχρονα στην εφαρμογή. Η διαφορά του Spike testing με το Stress testing είναι ο τρόπος με τον οποίο προσομοιώνεται η αύξηση του load. Στο Spike testing ή αύξηση του load συμβαίνει ξαφνικά και κατευθείαν σε ένα ακραίο βαθμό (χωρίς να υπάρξει προοδευτική αύξηση μέχρι να επιτευχθεί ο ακραίος βαθμός του load), ενώ στο Stress testing ή αύξηση του φόρτου συμβαίνει προοδευτικά και κλιμακωτά μέχρι να φτάσει έναν ακραίο βαθμό.
    - Capacity testing: Με τον έλεγχο χωρητικότητας (capacity testing) ελέγχεται πόσους χρήστες μπορεί να διαχειριστεί το σύστημα πριν η απόδοση πέσει κάτω από τα αποδεκτά επίπεδα. Ο έλεγχος χωρητικότητας βοηθάει τους developers να προβλέψουν προβλήματα όσον αφορά την επεκτασιμότητα και την μελλοντική ανάπτυξη του ορίου των χρηστών [35].
- Security testing: Ο έλεγχος της ασφάλειας ενός προϊόντος λογισμικού αποτελεί έναν ακόμη υποτύπο του non-functional testing ο οποίος διεξάγεται από την συντριπτική πλειοψηφία των επιχειρήσεων και των οργανισμών. Η ασφάλεια είναι μία από τις ουσιαστικότερες πτυχές των προϊόντων λογισμικού και ο έλεγχός της περιλαμβάνει την αξιολόγηση της ικανότητας της εφαρμογής να προστατέψει δεδομένα, λειτουργίες και πόρους από πιθανές ευπάθειες και απειλές. Η προσπάθεια εντοπισμού και εξάλειψης των κενών ασφαλείας χρησιμοποιώντας σαρώσεις, δοκιμές και αξιολογήσεις και άλλα μέσα, επιτυγχάνεται με το security testing. Με την σειρά του, το security testing περιέχει αρκετούς υποτύπους. Μερικοί από τους γνωστότερους υποτύπους του είναι οι εξής:
    - Vulnerability scanning / Vulnerability assessment: Πρόκειται για τη σάρωση του συστήματος με σκοπό τον εντοπισμό γνωστών μοτίβων ευπάθειας με την βοήθεια εργαλείων λογισμικού και για την μετέπειτα αξιολόγηση των αποτελεσμάτων. Τέτοιου

είδους ευπάθειες μπορούν να εντοπιστούν στον κώδικα, στην διαμόρφωση (configuration) και στην υποδομή (infrastructure) της εφαρμογής.
    - Penetration testing / Pen testing: Η δοκιμή διείσδυσης ή αλλιώς γνωστή ως «pen testing» περιλαμβάνει την προσομοίωση πραγματικών επιθέσεων στην εφαρμογή / σύστημα (σε ξεχωριστό περιβάλλον από το Production) και έχει ως στόχο τον εντοπισμό των αδυναμιών ασφαλείας. Αυτές οι προσομοιώσεις επιθέσεων είναι γνωστές και με τον όρο «Ethical hacking» και εκτελούνται από εξειδικευμένο προσωπικό που είναι γνωστοί ως Ethical hackers ή αλλιώς penetration testers ή white hats. Τα άτομα αυτά υποδύονται τους κακόβουλους hackers και προσπαθούν να εντοπίσουν και να εκμεταλλευτούν πιθανά τρωτά σημεία στο σύστημα.
    - Network security testing: Ο έλεγχος της ασφάλειας του δικτύου εστιάζει στον εντοπισμό τρωτών σημείων στην υποδομή του δικτύου του συστήματος. Μπορεί να περιλαμβάνει τον έλεγχο των firewalls, routers και άλλων συσκευών δικτύου ώστε να προσδιοριστούν οι πιθανές ευπάθειες [36].
    - Input validation testing: Με τις δοκιμές της επικύρωσης της εισόδου, ελέγχεται εάν τα inputs που λαμβάνονται από την εφαρμογή συμμορφώνονται με το πρότυπο που έχει οριστεί στην εφαρμογή. Για παράδειγμα αυτή η δοκιμή μπορεί να ελέγχει ένα regular expression (π.χ. ότι ο χρήστης πληκτρολόγησε μόνο αριθμούς και συγκεκριμένο πλήθος αριθμών για να εισάγει την ημερομηνία γέννησής του). Επίσης αυτός ο υποτύπος ελέγχου μπορεί να προσδιορίσει αν αποτρέπονται κοινά ζητήματα ασφαλείας όπως επιθέσεις SQL injection και cross-site scripting (γνωστές και ως επιθέσεις XSS (Cross Site Scripting) [36],[37].
- Usability testing: Το Usability testing αποτελεί το χαρακτηριστικό ποιότητας με βάση το οποίο αξιολογείται πόσο εύκολο είναι να χρησιμοποιηθεί το User Interface της εφαρμογής από τους χρήστες.
- Accessibility testing: Το Accessibility testing αποτελεί την προσπάθεια επαλήθευσης της προσβασιμότητας της εφαρμογής σε χρήστες με ειδικές ανάγκες (π.χ. περιορισμένη όραση).
- Compatibility testing: Ο έλεγχος για την συμβατότητα μιας εφαρμογής, διασφαλίζει ότι το σύστημα λειτουργεί σωστά σε διαφορετικές συσκευές (Device compatibility testing) ή σε διαφορετικούς browsers (Browser compatibility testing).

Όλα τα test types, ανεξαρτήτως εάν ανήκουν στο functional ή στο non-functional testing, μπορούν να εφαρμοστούν σε όλα τα test levels. Παρ' όλα αυτά, η εστίαση θα είναι διαφορετική σε κάθε επίπεδο[1].

2.19 Application Environments

Ο έλεγχος στα περισσότερα από τα Test types, ακόμη και σε μερικά Test levels, διεξάγεται σε ξεχωριστά περιβάλλοντα τα οποία αποτελούν αντίγραφα του πραγματικού περιβάλλοντος της εφαρμογής. Η εκτέλεση διάφορων τύπων tests μπορούν να επιβαρύνουν, να επιβραδύνουν, ακόμη και να καταστρέψουν το περιβάλλον της εφαρμογής. Για αυτόν τον λόγο οι επιχειρήσεις και οι οργανισμοί επιλέγουν να δημιουργούν και να χρησιμοποιούν ξεχωριστά περιβάλλοντα (γνωστά και με την γενική ονομασία «Pre-Production environments») για ορισμένα επίπεδα ή τύπους του ελέγχου. Για παράδειγμα το περιβάλλον που διεξάγεται το Penetration testing (εξηγείται παρακάτω) είναι ένα ξεχωριστό περιβάλλον από το live περιβάλλον (γνωστό και ως Production environment) που χρησιμοποιεί η επιχείρηση για τους πελάτες-χρήστες της. Αυτή η ενέργεια αποτρέπει την επιβάρυνση, επιβράδυνση ή ακόμη και καταστροφή του Production environment (λόγω της διεξαγωγής του Penetration testing), έτσι ώστε να αποφευχθεί η αδυναμία χρήσης της εφαρμογής από τους πραγματικούς πελάτες-χρήστες του προϊόντος. Η έννοια του περιβάλλοντος μιας εφαρμογής αναφέρεται σε ένα σύνολο από μεθόδους (application functions) οι οποίες καλούνται μέσω requests και εκτελούνται στον χώρο μιας διεύθυνσης ενός διακομιστή (server). Κάθε περιβάλλον μιας εφαρμογής πρέπει να αντιπροσωπεύει και απευθύνεται σε ένα ορισμένο και ονομασμένο σύνολο server functions, οι οποίες απαιτούν πρόσβαση στις βιβλιοθήκες (libraries) του server για το συγκεκριμένο περιβάλλον [38]. Για παράδειγμα οι server

functions για το περιβάλλον του UAT μπορούν να έχουν συγκεκριμένο πρόθεμα (prefix) (π.χ. «UAT_addUserToDB») ή κατάληξη (suffix) (π.χ. «addUserToDB_UAT») στην ονομασία τους και το αντίστοιχο για το Production περιβάλλον ή το περιβάλλον που εκτελείται το Penetration testing. Αυτά τα συγκεκριμένα prefixes ή suffixes ορίζονται και πρέπει να ακολουθούν-συμμορφώνονται με τις πρακτικές που χρησιμοποιεί μια επιχείρηση ή ένας οργανισμός για την χρήση των ονομασιών των server functions και βιβλιοθηκών. Επίσης το κάθε περιβάλλον έχει την δικιά του ξεχωριστή βάση δεδομένων ή ιδανικά χρησιμοποιείται η ίδια βάση δεδομένων για όλα τα περιβάλλοντα [39]. Στην δεύτερη και ιδανική περίπτωση, η βάση δεδομένων έχει χωριστεί σε διαφορετικά instances, όπου το κάθε instance χρησιμοποιείται για ένα συγκεκριμένο περιβάλλον. Αυτός ο διαχωρισμός διασφαλίζει ότι οι αλλαγές που έγιναν σε ένα περιβάλλον δεν επηρεάζουν άλλα περιβάλλοντα, επιτρέποντας έτσι την απομόνωση των παρενεργειών των tests από το πραγματικό περιβάλλον που χρησιμοποιούν οι χρήστες. Η ευθυγράμμιση μεταξύ των test environments με το UAT και το production, όσον αφορά τα features και τα χαρακτηριστικά ποιότητας είναι κάτι το οποίο θα πρέπει να λαμβάνεται υπόψιν από τις επιχειρήσεις και τους οργανισμούς προκειμένου να μην δυσχεραίνεται ο έλεγχος. Αν λείπουν ορισμένα features ή test data ή external connections στα test environments τότε η προσπάθεια του ελέγχου είναι σημαντικά δυσκολότερη. Δεν είναι λίγες οι φορές όπου οι επιχειρήσεις αψηφούν εν μέρει την ευθυγράμμιση των test environments με το UAT και με το Production περιβάλλον. Αυτό μπορεί να συμβεί λόγω του κόστους και της αδιαφορίας από την πλευρά των επιχειρήσεων-οργανισμών για την χρηστικότητα των test environments και της επικέντρωσης της προσοχής τους στο UAT και κυρίως στο Production environment ή/και στην ανάπτυξη νεών features για το προϊόν.

2.20 Test Techniques

Οι τεχνικές του ελέγχου (Test techniques) απευθύνονται στους διαφορετικούς τρόπους με τους οποίους μπορεί να υλοποιηθεί ένα μεγάλο κομμάτι της διασφάλισης της ποιότητας του λογισμικού, το οποίο είναι η δημιουργία των δοκιμών (tests cases). Αυτές οι τεχνικές, βοηθούν τους μηχανικούς της διασφάλισης ποιότητας του λογισμικού στην ανάλυση (τι να ελεγχτεί στο λογισμικό;) των test cases και στον σχεδιασμό τους (πώς να ελεγχτεί το λογισμικό;). Οι δοκιμές του ελέγχου βοηθούν στην ανάπτυξη ενός σχετικά μικρού αλλά επαρκούς συνόλου tests με συστηματικό τρόπο [28] και στην ενίσχυση της ολικής προσπάθειας του ελέγχου (test effort). Μέσω των δοκιμών, εξασφαλίζεται η ισορροπία μεταξύ της αποτελεσματικότητας του ελέγχου και της δεύτερης αρχής του ελέγχου, η οποία ορίζει ότι λόγω διάφορων προθεσμιών (χρόνος, κόστος, ανάγκη στην αγορά) ο εξαντλητικός έλεγχος είναι αδύνατος. Οι τεχνικές του ελέγχου χωρίζονται σε τέσσερεις κατηγορίες οι οποίες αναλύονται συνοπτικά παρακάτω.

- Black box test techniques: Η πρώτη κατηγορία ονομάζεται «Black box test techniques» ή αλλιώς «Specification-based techniques». Αυτή η κατηγορία βασίζεται στην ανάλυση των προδιαγραφών του test object και έχει ως στόχο να ελεγχθεί η καθορισμένη συμπεριφορά του χωρίς καμία πληροφορία-αναφορά για την εσωτερική του δομή. Οι δοκιμές του ελέγχου σε αυτή τη κατηγορία είναι ανεξάρτητες από τον πηγαίο κώδικα του λογισμικού καθώς το λογισμικό ελέγχεται ως ένα μαύρο κουτί, στο οποίο τα άτομα που διεξάγουν τον έλεγχο δεν έχουν καμία γνώση του εσωτερικού σχεδιασμού. Ονομαστικά, οι γνωστότερες Black box techniques του ελέγχου λογισμικού είναι οι εξής:

    - Equivalence partitioning
    - Boundary value analysis
    - Decision table testing
    - State transition testing

- White Box test techniques: Η δεύτερη κατηγορία, των τεχνικών του ελέγχου λογισμικού, ονομάζεται «White box techniques» ή αλλιώς «Structure-based techniques». Αντιθέτως με την πρώτη κατηγορία, οι White Box τεχνικές βασίζονται στην ανάλυση της εσωτερικής δομής του test object και της εσωτερικής επεξεργασίας που λαμβάνει χώρα στο test object. Επίσης τα test cases σε αυτή τη κατηγορία εξαρτώνται από την αρχιτεκτονική και τον λεπτομερή σχεδιασμό του πηγαίου κώδικα. Αυτή η κατηγορία τεχνικών χρησιμοποιείται όταν η ομάδα του διεξάγει τον έλεγχο, έχει πλήρη γνώση του εσωτερικού σχεδιασμού του test object. Ονομαστικά, οι δημοφιλέστερες White box techniques του ελέγχου λογισμικού είναι οι εξής:

    - Statement testing
    - Branch testing

- Grey box test techniques: Μια ακόμη κατηγορία των τεχνικών ελέγχου λογισμικού ονομάζεται «Grey box». Αυτή η κατηγορία αποτελεί συνδυασμό των πτυχών των παραπάνω κατηγοριών (Black box και White box). Οι τεχνικές αυτές βασίζονται στην μερική γνώση της εσωτερικής δομής του test object. Ονομαστικά οι γνωστότερες Grey box test techniques είναι οι εξής:

    - Matrix testing
    - Pattern testing
    - Orthogonal array testing

- Experience-based test techniques: Η επόμενη κατηγορία ονομάζεται «Experience based test techniques» και βασίζεται στην εμπειρία του ατόμου ή της ομάδας που διεξάγει τον έλεγχο. Η εμπειρία αυτή αξιοποιείται ώστε να σχεδιαστούν, να αναπτυχθούν και να εκτελεστούν τα test cases. Με αυτή την κατηγορία τεχνικών, μπορούν να εντοπιστούν ατέλειες οι οποίες θα ήταν αδύνατο να εντοπιστούν μέσω της χρήσης των Black box και των White box τεχνικών. Για αυτόν τον λόγο οι Experience-Based τεχνικές του ελέγχου θεωρούνται ως συμπληρωματικές των πρώτων δύο κατηγοριών. Ονομαστικά, οι δημοφιλέστερες Experience-based techniques του ελέγχου λογισμικού είναι οι εξής:

    - Error guessing
    - Exploratory testing
    - Checklist-based testing [1], [28].

2.21 Test Strategy and Test Approach

Ο όρος στρατηγική ελέγχου (Test Strategy) παρέχει μια γενικευμένη περιγραφή της διαδικασίας του ελέγχου (Test Process) ενός προϊόντος λογισμικού ή ενός τμήματος μιας επιχείρησης-οργανισμού. Οι πιο κοινώς χρησιμοποιούμενες στρατηγικές ελέγχου είναι οι εξής:

- Analytical: Η analytical test strategy βασίζεται στην ανάλυση κάποιου παράγοντα (π.χ. προδιαγραφών ή ρίσκου). Ο Risk-based έλεγχος είναι ένα παράδειγμα της αναλυτικής στρατηγικής ελέγχου. Τα test cases σε αυτή τη στρατηγική σχεδιάζονται και προτεραιοποιούνται με βάση το επίπεδο του ρίσκου.
- Model-based: Σε αυτή τη στρατηγική ελέγχου τα test cases βασίζονται σε κάποια μοντέλα απαιτούμενων πτυχών του έργου λογισμικού (π.χ. λειτουργικότητα, εσωτερική δομή, διαδικασιών της επιχείρησης) ή ενός χαρακτηριστικού ποιότητας.
- Methodical: Η μεθοδική στρατηγική ελέγχου βασίζεται στη συστηματική χρήση μερικών προκαθορισμένων δοκιμών ή συνθηκών (π.χ. ταξινόμηση των συνηθισμένων τύπων ατελειών, προτύπων, συγκεκριμένου theming και χρήσης των προϊόντων της επιχείρησης).

- Process-compliant (ή Standard compliant): Αυτή η στρατηγική ελέγχου ενισχύει την σχεδίαση των test cases με βάση κανονισμών και παραγόντων οι οποίοι θεωρούνται εξωτερικοί της ομάδας που εκτελεί τον έλεγχο. Μερικοί τέτοιοι κανονισμοί και παράγοντες μπορεί να ορίζονται από πρότυπα του συγκεκριμένου είδους βιομηχανίας που σχετίζεται η επιχείρηση, ή από τεκμηρίωση συγκεκριμένων διαδικασιών ή από οποιουδήποτε πρότυπο ή κανονισμό επιβάλλεται στην επιχείρηση ή από εκτελεστικά στελέχη της επιχείρησης.
- Directed (ή consultative): Αυτή η στρατηγική ελέγχου καθοδηγείται από τις συμβουλές και τις εντολές από την μεριά των stakeholders (εξειδικευμένους αναλυτές, άτομα με τεχνική εξειδίκευση ή testing experts) στην ομάδα του ελέγχου. Οι stakeholders είναι έμπειροι επαγγελματίες που μπορεί να βρίσκονται εντός ή εκτός της ομάδας του ελέγχου.
- Regression-averse: Η χρήση της Regression-averse στρατηγικής ελέγχου έχει ως στόχο και αποτελεί παρακίνηση για την αποφυγή εντοπισμού των regression bugs (γνωστά και ως «regressions»). Τα regressions προκαλούν δυσλειτουργία των είδη ανεπτυγμένων και υπαρχόντων features του λογισμικού. Αυτή η στρατηγική περιλαμβάνει την εκτεταμένη χρήση και αυτοματοποιημένη εκτέλεση του υπάρχοντος testware (test cases και test data) με σκοπό των εντοπισμό των regressions.
- Reactive: Με τη χρήση αυτής της στρατηγικής, ο έλεγχος διεξάγεται βάση των γεγονότων που προκύπτουν κατά την διάρκεια της εκτέλεσης των δοκιμών (test cases). Σε αυτή τη στρατηγική ελέγχου, τα test cases δεν έχουν προσχεδιαστεί (σε αντίθεση με όλες τις άλλες στρατηγικές). Τα test cases σχεδιάζονται, αναπτύσσονται και εκτελούνται ως απάντηση με βάση τη γνώση που αποκτήθηκε από προηγούμενα αποτελέσματα δοκιμών. Το Exploratory testing είναι μια κοινή τεχνική που χρησιμοποιείται στην reactive στρατηγική ελέγχου[1].

Το πιο σύνηθες για την δημιουργία μιας κατάλληλης στρατηγικής ελέγχου είναι ο συνδυασμός πολλών διάφορων στρατηγικών, καθώς η μια χρησιμοποιείται για να συμπληρώνει τις άλλες. Ο συνδυασμός διάφορων στρατηγικών ελέγχου μπορεί να οδηγήσει στην αποτελεσματικότερη δημιουργία των Test cases [1].

Ενώ η στρατηγική ελέγχου παρέχει μια γενικευμένη περιγραφή της διαδικασίας του ελέγχου, η προσέγγιση του ελέγχου (Test approach) είναι το σημείο εκκίνησης για την επιλογή των Test techniques, Test levels, Test types και των Entry και Exit Criteria. Η προσαρμογή της στρατηγικής ελέγχου βασίζεται στις αποφάσεις που λαμβάνονται σε σχέση με την πολυπλοκότητα και τους στόχους του έργου λογισμικού που αναπτύσσεται, του προϊόντος και της ανάλυσης του ρίσκου (risk analysis). Η επιλεγμένη προσέγγιση του ελέγχου εξαρτάται από το περιεχόμενο και λαμβάνει υπόψιν διάφορους παράγοντες (π.χ. κινδύνους, ασφάλεια, διαθέσιμους πόρους και δεξιότητες, τεχνολογία, φύση του συστήματος, στόχους και κανονισμούς του ελέγχου, κλπ.) [1].

2.22 Regression testing vs Non-regression testing.

Είναι πιθανό μια αλλαγή σε ένα τμήμα του κώδικα (π.χ. διόρθωση, προσθήκη, βελτίωση, ή οποιαδήποτε αλλαγή) να επηρεάσει ακούσια την συμπεριφορά άλλων τμημάτων του κώδικα χωρίς απαραίτητα αυτά τα δύο τμήματα να ανήκουν στο ίδιο component ή σύστημα. Τέτοιες αλλαγές μπορούν ακούσια να δημιουργήσουν δυσλειτουργίες και bugs στο λογισμικό. Το Regression testing αναφέρεται στην επανεκτέλεση ενός ή περισσότερων test suites, για να επιβεβαιωθεί ότι οι παραπάνω αλλαγές δεν επηρέασαν την υπάρχουσα λειτουργικότητα του λογισμικού. Πιο συγκεκριμένα, η διεξαγωγή του NRT(Non-Regression Testing) έχει στόχο να εντοπιστούν τυχόν side-effects (γνωστά και ως «regressions»), εφόσον έχει πραγματοποιηθεί μια αλλαγή σχετική με το λογισμικό ή στο λογισμικό.

Το NRT διεξάγεται με στόχο να επαληθευτεί αν μια προσθήκη ή αλλαγή λειτουργεί σωστά, με την υπόθεση ότι όλες οι προηγούμενες σχετικές λειτουργίες δεν έχουν επηρεαστεί. Μέσω της διεξαγωγής του, εντοπίζονται τα non-regression bugs (ή αλλιώς non-regressions).

Είναι πολύ σημαντικό να αναφερθεί το γεγονός ότι, στον τομέα του ελέγχου λογισμικού, υπάρχει ή έννοια του «Regressions testing» η οποία συχνά και λανθασμένα αναφέρεται ως «Non-regression testing». Η έννοια του Regression testing είναι ξεκάθαρη και είναι αποδεκτή από τον παγκόσμιο οργανισμό του ελέγχου λογισμικού (ISTQB) σε αντίθεση με την έννοια του «Non-regression testing». Στην πραγματικότητα, πρόκειται για δυο διαφορετικές έννοιες καθώς για την κάθε μια, μπορούν να εντοπιστούν bugs για διαφορετικό λόγο. Τα regressions και τα non-regressions εντοπίζονται μετά από κάποια αλλαγή στον κώδικα η οποία μπορεί να έγινε για οποιοδήποτε λόγο (επιδιόρθωση, προσθήκη ενός feature, κλπ.). Η ουσιαστική διαφορά είναι το scope τους. Τα regressions είναι ατέλειες οι οποίες δεν αναφέρονται στο εύρος της αλλαγής αυτής καθ' αυτής, ενώ τα non-regressions είναι ατέλειες οι οποίες απευθύνονται στο εύρος της συγκεκριμένης αλλαγής. Ιδανικά και εφόσον δεν βρεθούν bugs μετά την διεξαγωγή του Non-regression και του Regression testing, τα non regression tests προστίθενται στην σουίτα των regression tests εφόσον η λειτουργικότητα των αλλαγών-βελτιώσεων και του ολικού συστήματος επιτυχώς μετά την αλλαγή έχει επικυρωθεί. Αυτή η προσθήκη (των Non-regression tests στην σουίτα των Regression tests) συμβαίνει με στόχο να ενισχυθεί η αποτελεσματικότητα και η εμβέλεια του Regression testing, ώστε κάθε φορά που εκτελείται Regression testing να υπάρχει πιθανότητα εύρεσης περισσότερων ατελειών στο λογισμικό.

2.23 Entry and Exit criteria

Δύο ακόμη πολύ σημαντικές έννοιες στον έλεγχο λογισμικού είναι τα Entry criteria, ή αλλιώς γνωστά και με την έκφραση «Definition of Ready» και τα Exit criteria τα οποία είναι γνωστά με την έκφραση «Definition of Done». Τα Entry criteria αναφέρονται στις διαδικασίες που πρέπει να εκπληρωθούν έτσι ώστε να μπορεί να ξεκινήσει ο έλεγχος, ενώ τα Exit criteria αναφέρονται στις διαδικασίες που πρέπει να πραγματοποιηθούν προκειμένου η δραστηριότητα του ελέγχου να ολοκληρωθεί. Παρακάτω αναφέρονται μερικά από τα πιο συνηθισμένα παραδείγματα αυτών των δύο εννοιών:

- Entry criteria:
    - Διαθεσιμότητα στα testable requirements (π.χ. user stories, προδιαγραφές, flow charts, κλπ.).
    - Διαθεσιμότητα στα test items που πληρούν τα Exit criteria για οποιοδήποτε test level.
    - Διαθεσιμότητα στα απαραίτητα εργαλεία, πόρους, test environments και test data.

- Exit criteria:
    - Τα σχεδιασμένα Test cases έχουν εκτελεστεί.
    - Το προκαθορισμένο ποσοστό του test coverage για ορισμένα αντικείμενα (π.χ. προδιαγραφές, user stories, acceptance criteria, ρίσκα, κώδικας) έχει επιτευχθεί.
    - Ο αριθμός των μη διορθωμένων ατελειών είναι αποδεκτός.
    - Η σοβαρότητα των μη διορθωμένων ατελειών είναι χαμηλή.
    - Το αξιολογούμενο επίπεδο της αξιοπιστίας, της απόδοσης, της ευχρηστίας, της ασφάλειας και άλλων σημαντικών χαρακτηριστικών ποιότητας είναι επαρκές [1].

Είναι επίσης σύνηθες οι δραστηριότητες του ελέγχου να περιοριστούν ή ακόμη και να σταματήσουν, χωρίς να πληρούνται όλα τα Exit criteria. Αυτό μπορεί να συμβεί λόγω της δαπάνης του προϋπολογισμού, λόγω της περάτωσης των προθεσμιών ή λόγω της πίεσης εισαγωγής του προϊόντος στην αγορά. Σε τέτοιες περιπτώσεις, οι stakeholders και οι business owners κάνουν επισκόπηση και τελικώς αποφασίζουν εάν το προϊόν μπορεί να εισαχθεί στην αγορά, παρ' όλο τον κίνδυνο και τα ρίσκα του μη ολοκληρωμένου ελέγχου [1].

2.24 Επίλογος

Το δεύτερο κεφάλαιο της πτυχιακής εργασίας αναφέρει και αναλύει αρκετούς σημαντικούς ορισμούς σχετικούς με την διασφάλισης της ποιότητας του λογισμικού, του ελέγχου και των δοκιμών. Αρχικά αναφέρονται οι στόχοι του ελέγχου λογισμικού, η κατηγοριοποίηση του ελέγχου και οι διαφορές και ομοιότητες πολλών εννοιών (π.χ. QC vs QA, Verification vs Validation, false positive vs false negative, κλπ.) οι οποίες είναι απαραίτητες για την κατανόηση του ευρύτερου τομέα της διασφάλισης της ποιότητας λογισμικού, της στρατηγικής ελέγχου και του πώς αυτές υλοποιούνται στις επιχειρήσεις και στους οργανισμούς. Στην συνέχεια αναφέρονται οι εφτά αρχές του ελέγχου λογισμικού και αναλύονται οι εφτά δραστηριότητες του STLC και των επιμέρους διαδικασίες του καθώς και των Test work products που προκύπτουν από αυτές. Έπειτα αναλύονται οι βασικές διαφορές αρκετών ακόμη σημαντικών εννοιών (π.χ. Manual vs Automation testing, Severity vs Priority, Functional και Non-functional requirements) και αναφέρονται το δημοφιλέστερα tools για τις διαδικασίες του Test management και του Bug tracking. Επίσης αναλύονται τα επίπεδα του ελέγχου, οι διάφοροι τύποι και υποτύποι του και διάφοροι ακόμη ορισμοί και έννοιες (π.χ. Test Pyramid, Application Environments). Επιπλέον, περιγράφονται οι διάφορες κατηγορίες τεχνικών του ελέγχου και δίνονται μερικά αναφορικά παραδείγματα για την κάθε κατηγορία. Στην συνέχεια, εξηγούνται η έννοια του Test Strategy περιγράφοντας μερικά παραδείγματα στρατηγικών ελέγχου που χρησιμοποιούνται από τις επιχειρήσεις και η έννοια του Test Approach. Τέλος αναφέρονται οι διαφορές του Regression και του Non-regression testing και αποσαφηνίζεται η σύγχυση μεταξύ των δύο αυτών εννοιών και αναλύονται τα Entry και τα Exit criteria δίνοντας μερικά παραδείγματα. Η κατανόηση όλων των παραπάνω γνώσεων και εννοιών αποτελεί προαπαιτούμενο για την κατανόηση της υλοποίησης της διασφάλισης της ποιότητας, του ελέγχου και της στρατηγικής ελέγχου για προϊόντα λογισμικού, καθώς και για την κατανόηση των επόμενων κεφαλαίων.

Κεφάλαιο 3ο: Διεξαγωγή των δραστηριοτήτων του STLC στον ιστότοπο «https://uniportal.ihu.gr»

Στην συνέχεια αναλύεται το πώς διεξάγεται ο έλεγχος σε διαδικτυακές εφαρμογές και παρουσιάζεται η εφαρμογή αρκετών από τις παραπάνω έννοιες στην πράξη. Η διαδικτυακή εφαρμογή που επιλέχθηκε για την διεξαγωγή του ελέγχου είναι ο ιστότοπος «https://uniportal.ihu.gr» που χρησιμοποιεί το τμήμα «Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων» του Διεθνούς Πανεπιστημίου της Ελλάδος. Επίσης διεξάγεται ο στατικός και ο δυναμικός έλεγχος και επιλέγεται η στρατηγική ελέγχου και η προσέγγιση του ελέγχου που θα υλοποιηθεί. Τέλος ακολουθεί η διεξαγωγή των δραστηριοτήτων του STLC με την επικέντρωση της προσοχής στα τρία δημοφιλέστερα test automation frameworks που χρησιμοποιήθηκαν για να υλοποιηθούν οι διαδικασίες του test implementation και του test execution.

3.1 Επιλογή των Test Strategy και Tet Approach

Η επιλογή της στρατηγικής ελέγχου που επιλέγεται για τον έλεγχο της διαδικτυακής εφαρμογής του «https://uniportal.ihu.gr» πρόκειται για τον συνδυασμό των Model-based, Methodical, Regression-averse και Reactive στρατηγικών. Η Model-based στρατηγική ελέγχου επιλέγεται διότι οι δοκιμές (tests) θα στοχεύσουν στην προσπάθεια διασφάλισης της ποιότητας των λειτουργιών του Test object. Επίσης η Methodical στρατηγική ελέγχου επιλέγεται διότι τα tests θα στοχεύσουν στον έλεγχο προκαθορισμένων δοκιμών οι οποίες θεωρούνται βασικές για τις σημαντικότερες λειτουργίες της εφαρμογής (π.χ. log in) και υψηλής προτεραιότητας. Τέλος, η Regression-averse στρατηγική ελέγχου επιλέγεται επίσης ως συστατικό της μίξης των στρατηγικών ελέγχου που επιλέγονται καθώς τα tests θα αυτοματοποιηθούν και θα είναι εύκολο να επανεκτελεστούν ανά πάσα στιγμή, ενισχύοντας έτσι την αποφυγή των regression bugs.

Η προσέγγιση του ελέγχου (Test approach) που θα ακολουθηθεί θα επικεντρωθεί και στα εξής Test levels:

- System integration testing
- System testing
- Acceptance testing

Ο έλεγχος θα διεξαχθεί στο Test Type του Functional Testing. Για τον τύπο του Functional testing θα γίνει έλεγχος για τους υποτύπους των End to End και User Interface testing. Οι τεχνικές του ελέγχου που επιλέγονται επικεντρώνονται κυρίως στο Black box testing και στο Experience based testing. Μερικά από τα Entry criteria για τον έλεγχο είναι η διαθεσιμότητα της διαδικτυακής εφαρμογής του «https://uniportal.ihu.gr» και η δυνατότητα εισόδου του χρήστη στην διαδικτυακή εφαρμογή (log in). Τέλος μερικά από τα Exit criteria είναι η εκτέλεση τουλάχιστον δέκα tests. Επίσης ένα ακόμη Exit criterion είναι η σοβαρότητα (Severity) των μη διορθωμένων ατελειών (σε περίπτωση που εντοπιστούν) να είναι χαμηλή.

3.2 Εφαρμογή του Static testing

Το Static testing αναφέρεται στην επαλήθευση των static work products της διαδικτυακής εφαρμογής. Στην συγκεκριμένη περίπτωση το μόνο διαθέσιμο static work product στον ιστότοπο του «https://uniportal.ihu.gr» είναι το εγχειρίδιο χρήσης της εφαρμογής. Το έγγραφο αυτό πρέπει να εξεταστεί για τον τυχόν εντοπισμό παραλήψεων, αντιφάσεων, ασυνεπειών, ανακριβειών και περιττών

δηλώσεων. Έπειτα από την παραπάνω εξέταση εντοπίστηκαν μερικά static bugs τα οποία αναφέρονται στην διεξαγωγής της δραστηριότητας του Test analysis (κεφάλαιο 3.3.1.3).

3.3 Διεξαγωγή των δραστηριοτήτων του STLC για Functional testing

Στα ακόλουθα υποκεφάλαια εφαρμόζονται οι δραστηριότητες του STLC για το Functional testing στον ιστότοπο του «https://uniportal.ihu.gr». Αναλύονται όλες οι επιμέρους δραστηριότητες του STLC καθώς και η επιμέρους εργασίες της κάθε δραστηριότητας. Τέλος παρουσιάζονται τα αναμενόμενα Test work products για την κάθε επιμέρους εργασίες κάθε δραστηριότητας του STLC.Οι δραστηριότητες του STLC θα επικεντρώσουν την προσοχή τους στους τύπους των End to End, και UI tests.

3.3.1 Test planning για End to End και UI test types

Το Test planning για την διεξαγωγή του ελέγχου στον ιστότοπο «https://uniportal.ihu.gr» αποτελείται από ένα Master test plan το οποίο συντονίζει τον έλεγχο των Functional tests στα επίπεδα του System Integration testing, System testing και Acceptance testing. Το test plan θα δημιουργηθεί για τους υποτύπους των End to End και UI tests. Η δημιουργία αυτών των εγγράφων, θα βασιστεί πάνω στην χρήση ενός template. Στο Παράρτημα Α παρουσιάζεται το Master Test Plan ενώ στο Παράρτημα Β το Test Plan για την διεξαγωγή του ελέγχου στον ιστότοπο του «https://uniportal.ihu»

3.3.2 Test monitoring and control για End to End και UI test types

Κατά την δραστηριότητα του Test monitoring and control πραγματοποιείται η συνεχής σύγκριση της κάλυψης των δοκιμών που ορίστηκαν στο Test plan (test pass ratio 80%). Η πρόοδος του ελέγχου εξελίσσεται φυσιολογικά τηρώντας την προθεσμία και δεν υπάρχει κάποια απόκλιση από τους στόχους της εφαρμογής οπότε δεν υπάρχει λόγος λήψης επεμβατικών ενεργειών.

3.3.3 Test analysis για End to End και UI test types

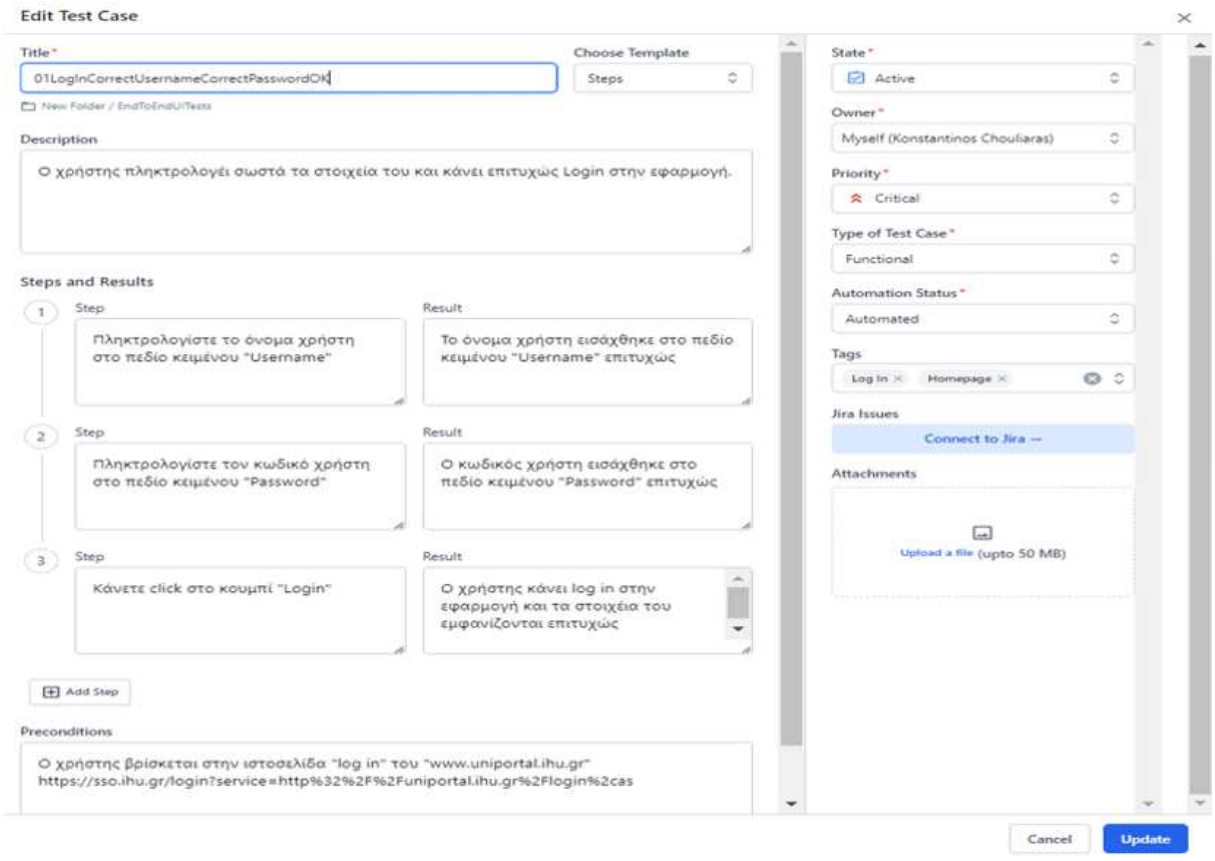
Κατά την δραστηριότητα του Test analysis για τα End to End και UI test types πραγματοποιείται μελέτη και έλεγχος στα test basis. Το test basis για τον ιστότοπο του «https://uniportal.ihu.gr» είναι το εγχειρίδιο χρήσης. Κατά την μελέτη του εγχειρίδιου χρήσης αποκτάται το λειτουργικό υπόβαθρο για την εφαρμογή. Ακόμη μέσω της μελέτης αυτής, γίνονται αντιληπτές οι λειτουργίες και τα χαρακτηριστικά της εφαρμογής που πρέπει να ελεγχθούν. Επίσης έπειτα από την διεξαγωγή του στατικού ελέγχου στο εγχειρίδιο χρήσης εντοπίστηκαν μερικά static bugs και πιο συγκεκριμένα, δύο αντιφάσεις. Η πρώτη αντίφαση που εντοπίστηκε είναι ότι στο εγχειρίδιο τα τελευταία νέα και οι ανακοινώσεις εμφανίζονται στην σελίδα του προφίλ του χρήστη, ενώ στην διαδικτυακή εφαρμογή ολόκληρο αυτό το κομμάτι λείπει. Η δεύτερη αντίφαση που εντοπίστηκε είναι ότι στο ημερολόγιο εξετάσεων το drop-down list του ακαδημαϊκού έτους λείπει από το screenshot του εγχειρίδιου ενώ στην διαδικτυακή εφαρμογή το drop-down list του ακαδημαϊκού έτους είναι διαθέσιμο. Αυτές οι αντιφάσεις ίσως είναι ορατές λόγω της έλλειψης ενημέρωσης του εγχειριδίου μετά από κάποιο νέο version ή release της διαδικτυακής εφαρμογής ή για άλλους λόγους όπως η χρήση ενός διαφορετικού ρόλου χρήστη (π.χ. admin user) για την λήψη των screenshots που ενσωματώθηκαν στο εγχειρίδιο. Σε κάθε περίπτωση τα static bugs πρέπει να αναφέρονται (επίσημα ή ανεπίσημα) στην ευρύτερη ομάδα (π.χ. Scrum team), προκειμένου να αποσαφηνιστεί η αιτία τους και να ληφθούν οι ανάλογες ενέργειες (π.χ. διόρθωση των static bugs στα test basis).

Το bi-directional traceability matrix είναι ορατό μεταξύ των ονομάτων των Test conditions, των Test cases και των Test scripts. Τα Test conditions αναφέρονται παρακάτω:

- Σωστό Username + σωστό Password : Login OK
- Σωστό Username + λάθος Password : Login KO
- Λάθος Username + Σωστό Password : Login KO
- Κενό String Username + σωστό Password: Login KO
- Σωστό Username + κενό String Password: Login KO
- Όλα τα hamburger menu items πρέπει να είναι ορατά όταν το hamburger menu βρίσκεται σε «collapsed» state
- Το ημερολόγιο των εξετάσεων πρέπει να εμφανίζεται σωστά για το επιλεγμένο ακ. έτος εφόσον ο φοιτητής έχει δώσει εξετάσεις το επιλεγμένο έτος
- Οι βαθμολογίες του φοιτητή είναι διαθέσιμες. Οι εγγραφές μπορούν να παρουσιαστούν ανάλογα με την επιλογή του φοιτητή
- Οι πρακτική/ες του φοιτητή είναι διαθέσιμες επιλέγοντας την αντίστοιχη επιλογή
- Τα προσωπικά δεδομένα του φοιτητή είναι διαθέσιμα
- Ο οδηγός χρήσης είναι διαθέσιμος
- Η αλλαγή γλώσσας λειτουργεί

3.3.4 Test design για End to End και UI test types

Για την δραστηριότητα του Test design χρησιμοποιήθηκε το test management tool «Browser Stack». Το Browser Stack προσφέρει την δυνατότητα της δημιουργίας και της επεξεργασίας/ενημέρωσης των test cases. Στο παρακάτω σχήμα παρουσιάζονται όλα τα απαραίτητα πεδία ενός test case (π.χ. test case για τον έλεγχο του επιτυχούς log in) το οποία χρησιμεύουν για επόμενη δραστηριότητα του STLC που είναι το test implementation.

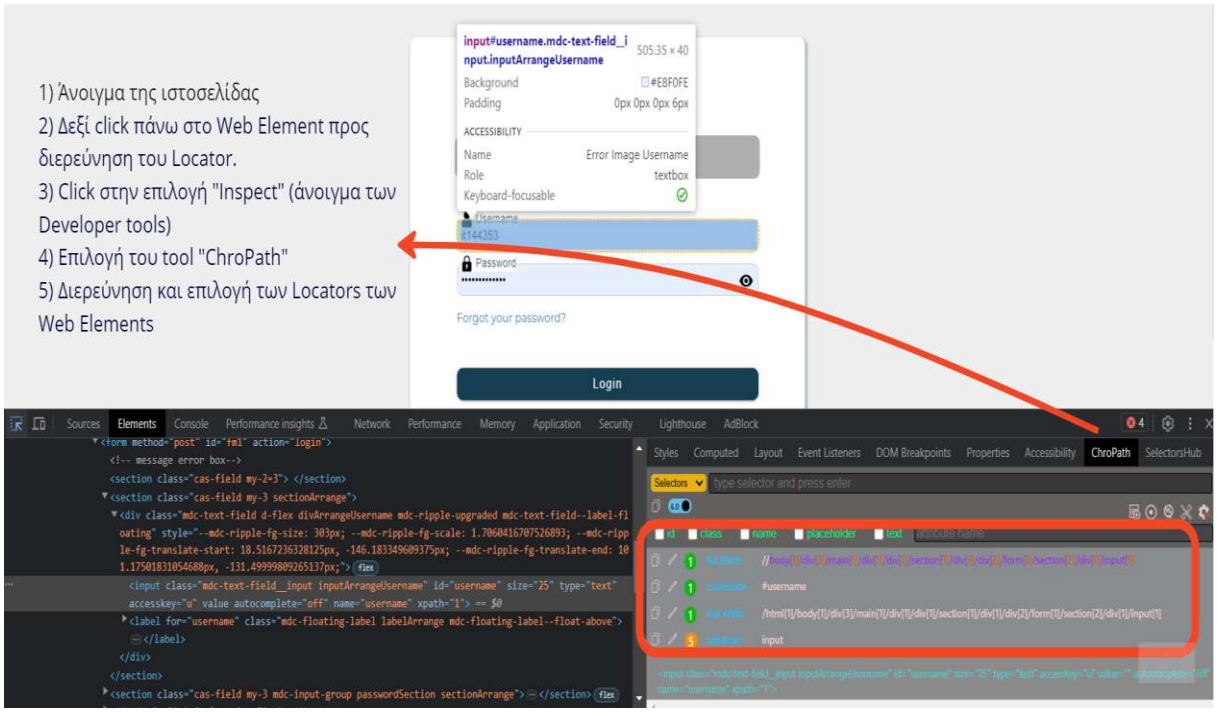


Σχήμα 3.1: Δημιουργία και επεξεργασία ενός test case μέσω του test management tool «Browser Stack»

Ο τίτλος, η περιγραφή, τα ατομικά βήματα, τα αναμενόμενα αποτελέσματα, οι προϋποθέσεις και άλλες λεπτομέρειες όπως η κατάσταση, ο δημιουργός, η προτεραιότητα, ο τύπος, η κατάσταση αυτοματοποίησης, οι λειτουργικές ετικέτες και οι σχετικές επισυνάψεις του κάθε test cases αναφέρονται σε αυτό το σημείο.

3.3.5 Test Implementation για End to End και UI test types

Τα automation testing frameworks έχουν αναδειχθεί ως απαραίτητα εργαλεία, εκσυγχρονίζοντας τη διαδικασία των tests και διασφαλίζοντας την αξιοπιστία του προϊόντος. Μεταξύ των κορυφαίων διεκδικητών σε αυτήν την αρένα, το Selenium, το Cypress και το Playwright έχουν αναδειχθεί. Για αυτόν τον λόγο, για το Test implementation των End to End και των UI test types χρησιμοποιήθηκαν τα τρία δημοφιλέστερα web automation tools (Selenium, Cypress και Playwright). Αυτά τα εργαλεία επιτρέπουν την αυτοματοποίηση των ενεργειών ενός υποτιθέμενου χρήστη πάνω στα Web elements και στην συνέχεια την δυνατότητα επικύρωσης με την χρήση των ισχυρισμών (assertions) . Τα παραπάνω εργαλεία (εκτός του Playwright που δεν απαιτείται testing framework) συνδυάστηκαν με testing frameworks και με διάφορες άλλες βιβλιοθήκες προκειμένου να διεξαχθεί η δραστηριότητα του Test Implementation. Τα automated tests δημιουργούνται μέσω του συνδυασμού των τριών εννοιών του End to End automation. Αυτές οι έννοιες είναι η εντόπιση (locate) των Web elements, η ενέργεια (action) και ο ισχυρισμός (assertion) που λαμβάνουν χώρα στα Web elements. Τα Web elements είναι τα στοιχεία μια ιστοσελίδας (text boxes, lists, buttons, κλπ.) και αναπαρίστανται μέσω των HTML(Hypertext Markup Language) elements. Επίσης χρησιμοποιήθηκαν τα browser extensions «Chropath», «Web Element Locator» και «SelectorsHub» προκειμένου να εντοπιστούν οι locators (π.χ. id, className, XPath, id, cssSelector, κλπ.) των Web elements, έτσι ώστε να μπορέσουν τα automation testing frameworks να αναφερθούν Web elements. Έπειτα από την αναφορά των Web elements, τα automation testing frameworks μπορούν να πράξουν συγκεκριμένες ενέργειες (actions) και ισχυρισμούς (assertions) στα Web elements προκειμένου να πραγματοποιηθεί η επικύρωση. Για παράδειγμα, υποθέτοντας ότι χρήστης έχει πραγματοποιήσει επιτυχώς την διαδικασία του «Log in» (κάνοντας ατομικές ενέργειες στα Web elements που ανήκουν στην ιστοσελίδα «Log in»), τότε πρέπει να κατευθυνθεί (redirect) στην ιστοσελίδα του «Homepage». Στο παρακάτω σχήμα παρουσιάζεται η διαδικασία εύρεσης των Locators για τα Web elements μέσω της χρήσης του browser extension «Chropath» στην ιστοσελίδα του «Log in» στον ιστότοπο «https://uniportal.ihu.gr».



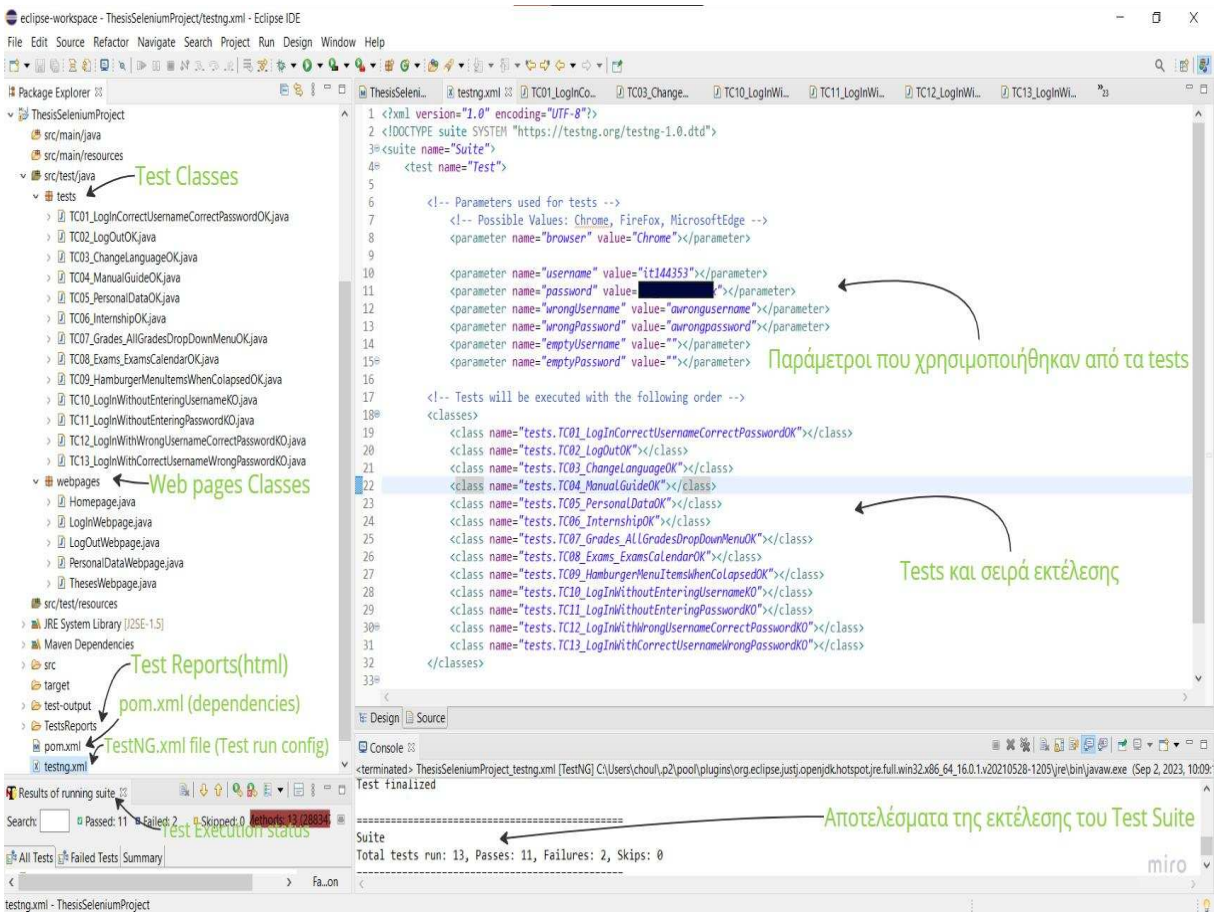
Σχήμα 3.2: Διαδικασία εύρεσης των Locators μέσω της χρήσης του εργαλείου «ChroPath»

Η αυτοματοποίηση των tests μέσω του Selenium automation framework πραγματοποιήθηκε με την γλώσσα προγραμματισμού Java την χρήση του Eclipse IDE. Κύρια πηγή αναφοράς για το Selenium project αποτέλεσε ο επίσημος ιστότοπος του Selenium [40]. Η επιλογή της γλώσσας Java λήφθηκε διότι τα περισσότερα projects που έχουν υλοποιηθεί σε Selenium χρησιμοποιούν αυτή τη γλώσσα. Το JDK(Java Development Kit) απαιτείται για την εκτέλεση αυτού του project. Το JDK περιέχει την γλώσσα προγραμματισμού Java καθώς και το JRE(Java Runtime Environment) και είναι διαθέσιμο στον επίσημο ιστότοπο της Oracle [41]. Δημιουργήθηκε ένα Maven project το οποίο στο αρχείο «pom.xml» περιέχει τις εξής βιβλιοθήκες για τους συγκεκριμένους σκοπούς:

- Selenium → Υποστήριξη αυτοματοποίησης των Web Browsers και μίμησης των ενεργειών ενός χρήστη
- jUnit → Δυνατότητα ανάπτυξης και εκτέλεσης των tests
- WebDriverManager → Δυνατότητα εύκολης διαχείρισης και εκτέλεσης των tests σε διαφορετικούς browsers
- TestNG(Test Next Generation) → Δυνατότητα εισαγωγής annotations, παραμέτρων και δυνατότητα εκτέλεσης των Test Suites
- ExtendReports → Δημιουργία των Test reports

Ακόμη εφαρμόστηκε η αρχή σχεδίασης (design principle) του Page Object Model. Αυτή η αρχή σχεδίασης αποτελεί προσπάθεια για την διαχώριση των locators και των actions για κάθε Web Element σε μια διαφορετική κλάση Java για κάθε ιστοσελίδα (η οποία έχει το αντιπροσωπευτικό όνομα αυτής της ιστοσελίδας), από τα assertions τα οποία λαμβάνουν μέρος στην κλάση Java του αρχείου για κάθε test (το Java class έχει αντιπροσωπευτικό όνομα για κάθε test). Η κάθε κλάση του Page Object Model, δεν είναι απαραίτητο να αντιπροσωπεύει μια συγκεκριμένη ιστοσελίδα του ιστότοπου, καθώς μπορούν να δημιουργηθούν επιπρόσθετες κλάσεις για τμήματα του ιστότοπου που συγκεντρώνουν κάποια λειτουργικότητα. Η υλοποίηση του Page Object Model μπορεί να διαφέρει από επιχείρηση σε επιχείρηση ακόμη και αν η υλοποίηση πραγματοποιηθεί στην ίδια ιστοσελίδα από διαφορετικές επιχειρήσεις / οργανισμούς. Η υλοποίηση του Page Object Model αφήνεται στην κρίση της ομάδας που υλοποιεί τον έλεγχο. Αυτή η αρχή σχεδίασης των tests συνίσταται διότι βοηθάει στην

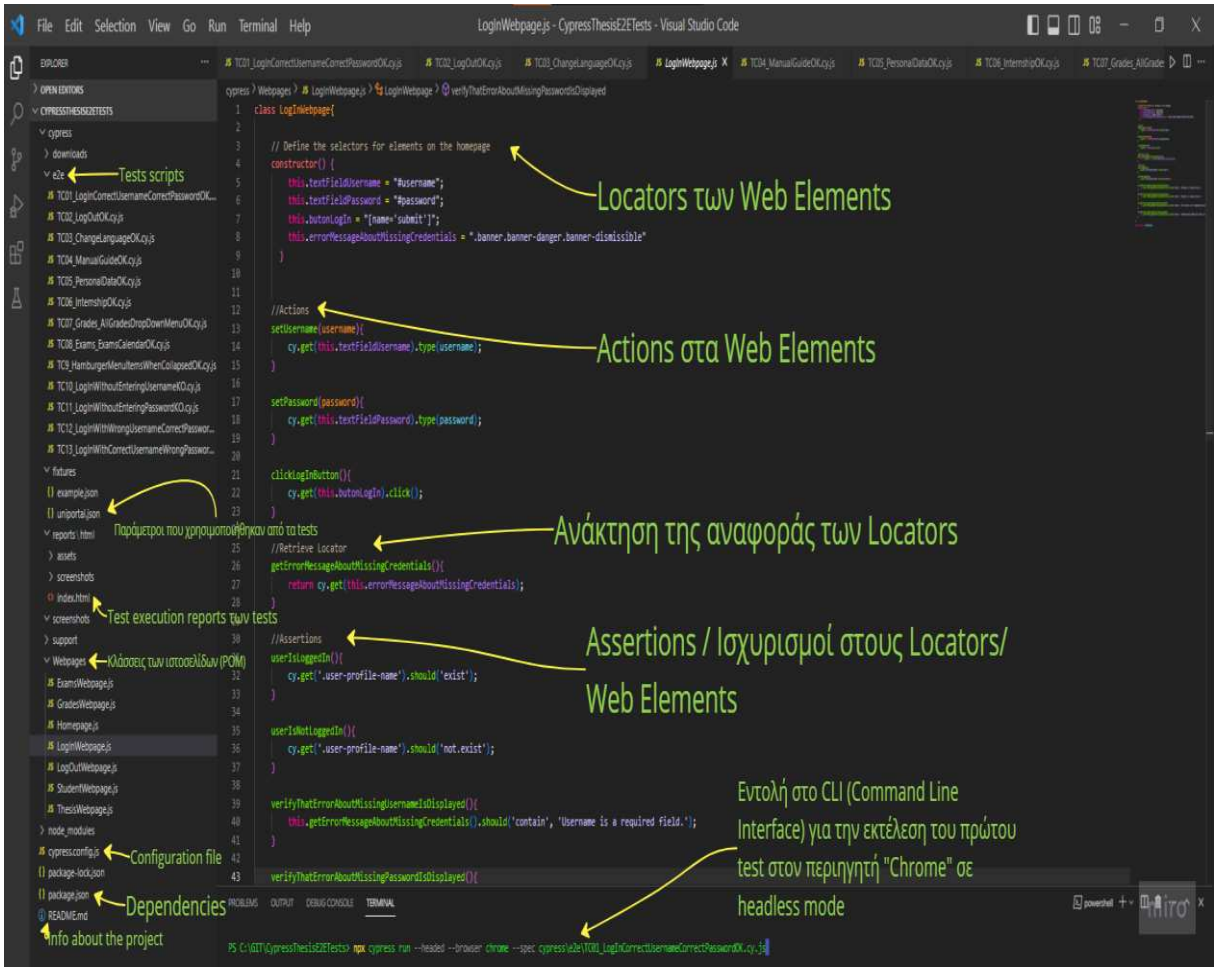
επαναχρησιμοποίηση και στις αλλαγές στον κώδικα. Με την χρήση του Page Object Model, ένας locator ενός Web Element μιας ιστοσελίδας, ορίζεται μόνο μία φορά και μπορεί να προσπελαστεί από οποιοδήποτε test δημιουργώντας ένα στιγμιότυπο αυτής της ιστοσελίδας. Για την υλοποίηση του project με τη χρήση του Selenium, οι κλάσεις που αντιπροσωπεύουν κάθε ιστοσελίδα βρίσκονται στον φάκελο «src/test/java/Webpages» ενώ τα tests βρίσκονται στον φάκελο «src/test/java/tests». Το αρχείο «readme.md» περιέχει περισσότερες πληροφορίες που είναι σχετικές με το project. Το testing framework που χρησιμοποιήθηκε σε συνδυασμό με το Selenium είναι το «TestNG». Το TestNG πρέπει να εγκατασταθεί στο Eclipse IDE μέσω της επιλογής «Install New Software…» της επιλογής «Help» του κεντρικού μενού. Έπειτα μέσω της επιλογής «Add» το «TestNG» προστίθεται ως το όνομα του repository text field και το «https:/beust.com.eclipse» προστίθεται στο text filed του location. Με την χρήση του TestNG αντί για «main()» μέθοδο χρησιμοποιούνται μερικά annotations για την εκτέλεση ενός test. Το annotation «@BeforeTest» χρησιμοποιείται στην μέθοδο που θα τρέξει πριν την μέθοδο του test και συνήθως περιέχει κάποιο set up, το annotation «@Test» που χρησιμοποιείται για την μέθοδο που περιέχει το test και το annotation «@AfterTest» το οποίο περιέχει ενέργειες που λαμβάνουν μέρος μετά την εκτέλεση του test. Επίσης το annotation «@Parameters» μπορεί να προηγηθεί μιας μεθόδου σε περίπτωση που αυτή η μέθοδος χρησιμοποιεί παραμέτρους. Στο παρακάτω σχήμα παρουσιάζεται η δομή του project για το End to End και UI test μέσω της χρήσης του Selenium.



Σχήμα 3.3: Selenium End to End και UI test implementation

Για την υλοποίηση των End to End / UI tests με την χρήση του automation testing framework «Cypress» χρησιμοποιήθηκε το IDE «Visual Studio Code» και η γλώσσα προγραμματισμού «JavaScript». Η επιλογή της γλώσσας JavaScript λήφθηκε διότι τα περισσότερα projects που έχουν υλοποιηθεί σε Cypress χρησιμοποιούν αυτή τη γλώσσα. Κύρια πηγή αναφοράς για το Cypress project αποτέλεσε ο

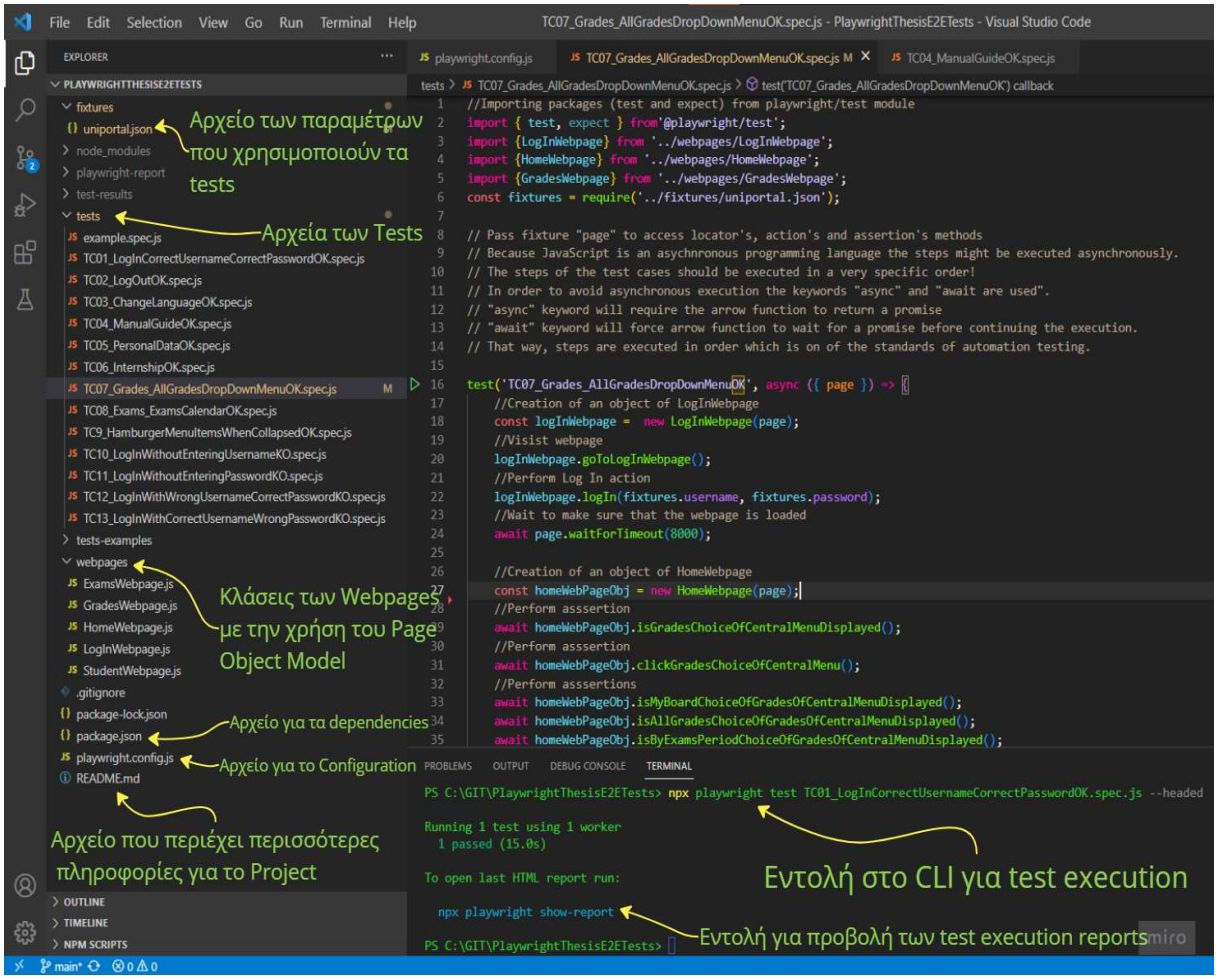
επίσημος ιστότοπος του Cypress [42]. Προκειμένου να εκτελεστούν τα End to End / UI tests με την χρήση του Cypress απαιτείται η εγκατάσταση της πλατφόρμας «node.js». Επίσης για την οργάνωση και την διαχείριση των dependencies του project απαιτείται και η εγκατάσταση του εξ' ορισμού NPM(Node Package Manager) του node.js. Ο NPM περιλαμβάνεται και εγκαθίσταται αυτόματα μέσω της εγκατάστασης του node.js. Το περιβάλλον του node.js είναι διαθέσιμο στην επίσημη ιστοσελίδα του [43]. Στον φάκελο «e2e» βρίσκονται όλα τα test suites και τα tests. Η υλοποίηση των tests με το Cypress, έγινε μέσω της χρήσης της αρχής σχεδίασης Page Object Model. Στον φάκελο «Webpages» ανήκουν όλες οι JavaScript κλάσεις οι οποίες αντιπροσωπεύουν κάθε ιστοσελίδα. Επίσης, τον φάκελο «reports» το αρχείο «index.html» περιέχει το test execution report των tests που εκτελέστηκαν στο τελευταίο execution run. Στον φάκελο «fixtures» το αρχείο «uniportal.json» περιέχει όλες τις παραμέτρους οι οποίες χρειάστηκαν για την εκτέλεση των tests. Τέλος στο αρχείο «cypress.config.js» ορίζεται το configuration του project, στο αρχείο «package.json» αναφέρονται τα dependencies ενώ στο αρχείο «readme.md» αναφέρονται περισσότερες πληροφορίες για το project. Στο παρακάτω σχήμα παρουσιάζεται η δομή του project για τα End to End / UI tests με την χρήση του Cypress.



Σχήμα 3.4: Cypress End to End και UI test implementation

Για την υλοποίηση των End to End / UI tests με την χρήση του automation testing framework «Playwright» χρησιμοποιήθηκε το IDE «Visual Studio Code» και η γλώσσα προγραμματισμού «JavaScript». Η επιλογή της γλώσσας JavaScript λήφθηκε διότι τα περισσότερα projects που έχουν υλοποιηθεί σε Playwright χρησιμοποιούν αυτή τη γλώσσα. Κύρια πηγή αναφοράς για το Playwright project αποτέλεσε ο επίσημος ιστότοπος του Playwright [43]. Προκειμένου να εκτελεστούν τα End to

End / UI tests με την χρήση του Playwrigh απαιτείται η εγκατάσταση της πλατφόρμας «node.js». Για την οργάνωση και την διαχείριση των dependencies του project απαιτείται και η εγκατάσταση του εξ' ορισμού NPM(Node Package Manager) του node.js. Ο NPM περιλαμβάνεται και εγκαθίσταται αυτόματα μέσω της εγκατάστασης του node.js. Το περιβάλλον του node.js είναι διαθέσιμο στην επίσημη ιστοσελίδα του [42]. Σχετικά με την δομή του project με την χρήση του Playwright, στον φάκελο «tests» βρίσκονται όλα τα tests. Η υλοποίηση των tests με το Playwright, έγινε μέσω της χρήσης της αρχής σχεδίασης Page Object Model. Στον φάκελο «webpages» ανήκουν όλες οι JavaScript κλάσεις οι οποίες αντιπροσωπεύουν κάθε ιστοσελίδα. Επίσης, τον φάκελο «reports» το αρχείο «index.html» περιέχει το test execution report των tests που εκτελέστηκαν στο τελευταίο execution run. Στον φάκελο «fixtures» το αρχείο «uniportal.json» περιέχει όλες τις παραμέτρους οι οποίες χρειάστηκαν για την εκτέλεση των tests. Τέλος στο αρχείο «playwright.config.js» ορίζεται το configuration του project, στο αρχείο «package.json» αναφέρονται τα dependencies ενώ στο αρχείο «readme.md» αναφέρονται περισσότερες πληροφορίες για το project. Στο παρακάτω σχήμα παρουσιάζεται η δομή του project για τα End to End / UI tests με την χρήση του Playwright.

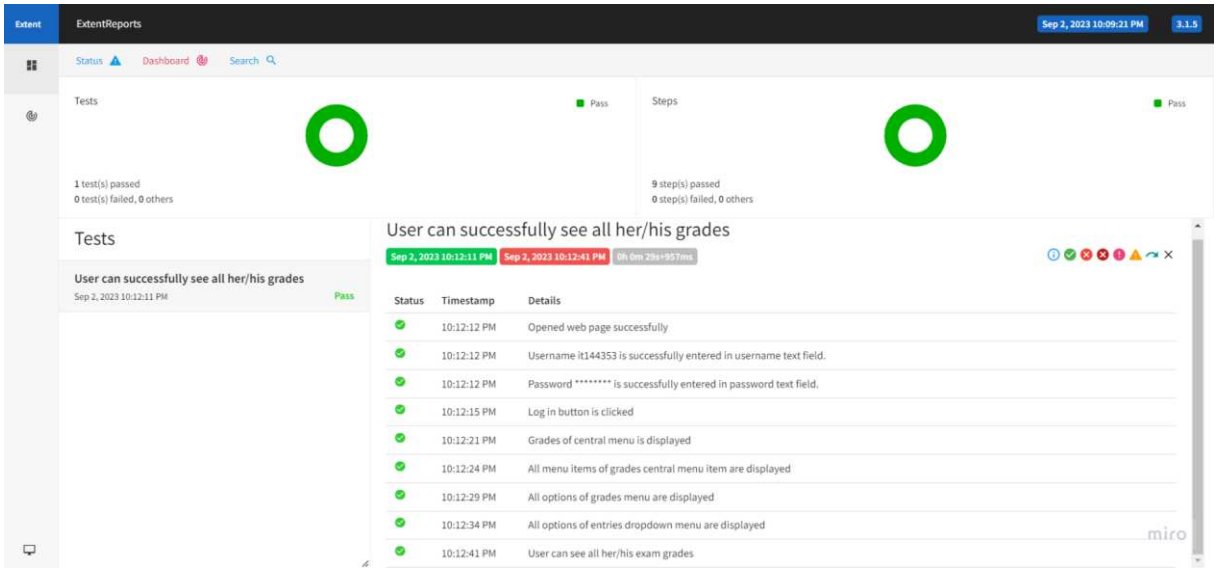


Σχήμα 3.5: Playwright End to End και UI test implementation

3.3.6 Test execution για End to End και UI test types

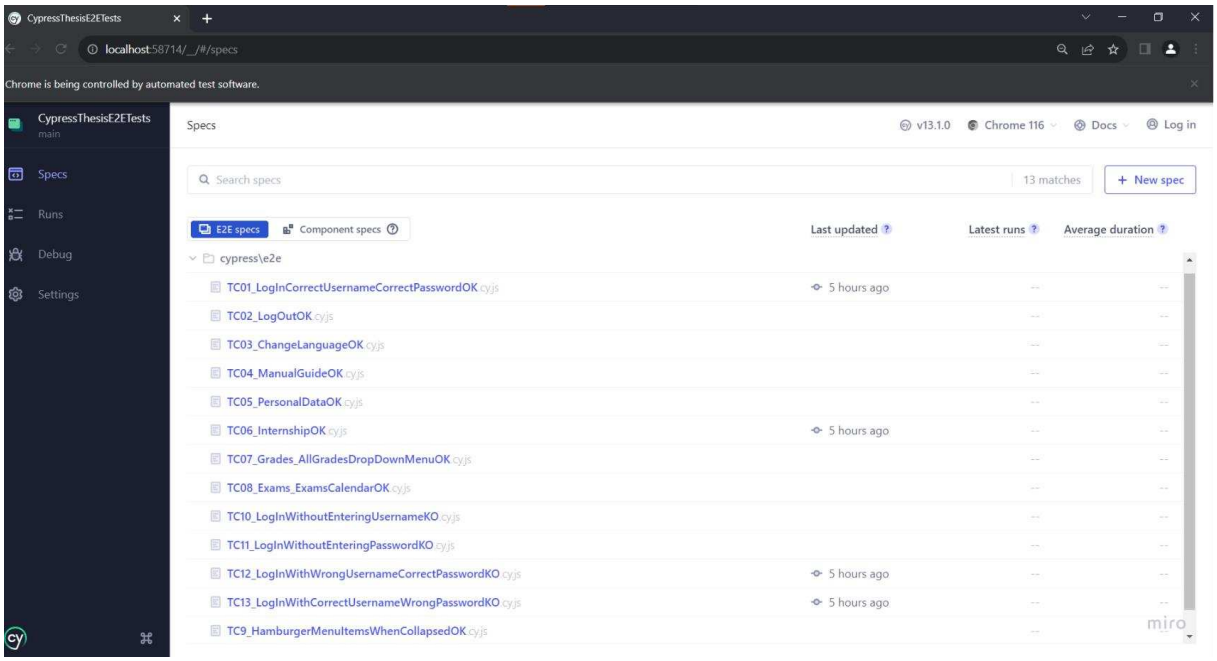
Η δραστηριότητα του Test Execution του Selenium project πραγματοποιήθηκε μέσω του test framework TestNG. Στο αρχείο «testng.xml», η οργάνωση της εκτέλεσης των tests μπορεί να πραγματοποιηθεί σε Test Suites. Επίσης η προτεραιότητα της εκτέλεσης του κάθε Test suite και του κάθε test μπορεί να

οριστεί στο αρχείο testng.xml, καθώς και επιπλέον επιλογές σχετικές με την εκτέλεση(π.χ. παράλληλη εκτέλεση των tests, Threads που θα χρησιμοποιηθούν, κλπ.). Για να εκτελεστούν τα End to End / UI tests πρέπει να εκτελεστούν μέσω του αρχείου «testing.xml» έχοντας ανοίξει το project στο Eclipse IDE, κάνοντας δεξί κλικ επάνω στο αρχείο «testing.xml», επιλέγοντας «Run as» και κάνοντας κλικ στην επιλογή «1 TestNG Suite». Η παράλληλη εκτέλεση όλων των tests πρέπει να αποφευχθεί καθώς μπορεί να επηρεάσει την λειτουργία του ιστότοπου «https://uniportal.ihu.gr». Η βιβλιοθήκη «extend reports» χρησιμοποιήθηκε για το Test Reporting. Τα αποτελέσματα του κάθε test εξάγονται στον φάκελο του project «TestReports» και είναι διαθέσιμα σε μορφή «.html». Η reporting library «extend reports» είναι ένα open-source reporting API διαθέσιμο για τις γλώσσες προγραμματισμού Java και .Net. Τα αρχεία «.html» που παράγονται για τα αποτελέσματα των tests είναι διαδραστικά και περιέχουν όλες τις απαραίτητες πληροφορίες για το κάθε βήμα και το τελικό αποτέλεσμα του κάθε test. Στο παρακάτω σχήμα παρουσιάζεται ως παράδειγμα το test execution report του test «TC07_Grades_AllGrades_DropDownMenuOK».



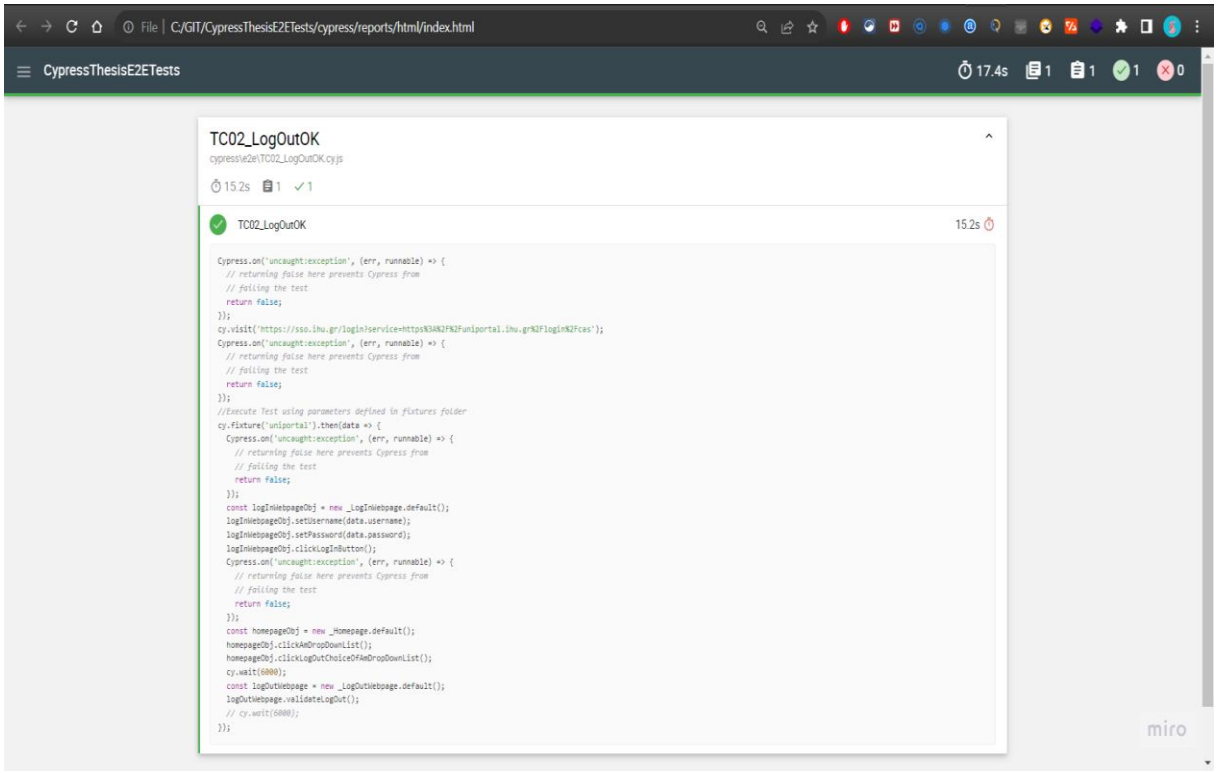
Σχήμα 3.6: Test execution report ενός test με την χρήση της βιβλιοθήκης «Extend Reports»

Η δραστηριότητα του Test Execution του Cypress project πραγματοποιήθηκε με την βοήθεια του test framework «Mocha». Η εκτέλεση των tests πραγματοποιείται μέσω της χρήσης του CLI(Command Line Interface) στον φάκελο του project. Η εκτέλεση των tests γίνεται μέσω της χρήσης της εντολής «npx cypress run όνομα_του_test_suite». Η εντολή της εκτέλεσης μπορεί να περιέχει διάφορα ορίσματα που αφορούν την επιλογή του browser που επιθυμείται να εκτελεστούν τα tests, ή επιλογή του τρόπου εκτέλεσης (headed ή headless). Το Cypress εξ' ορισμού εκτελεί τα tests χωρίς να είναι ορατός ο browser (headless mode). Ένα test ή ένα test suite μπορεί να εκτελεστεί ενώ παράλληλα να είναι ορατός ο browser μέσω της προσθήκης του ορίσματος «--headed» στην εντολή εκτέλεσης. Ένας άλλος τρόπος εκτέλεσης των tests στο project του Cypress είναι η εκτέλεση της εντολής «npx cypress open» στο CLI του directory του project, έτσι ώστε να ανοίξει η εφαρμογή του Cypress. Στο παρακάτω σχήμα παρουσιάζεται η εφαρμογή του Cypress και τα test suites διαθέσιμα για εκτέλεση.



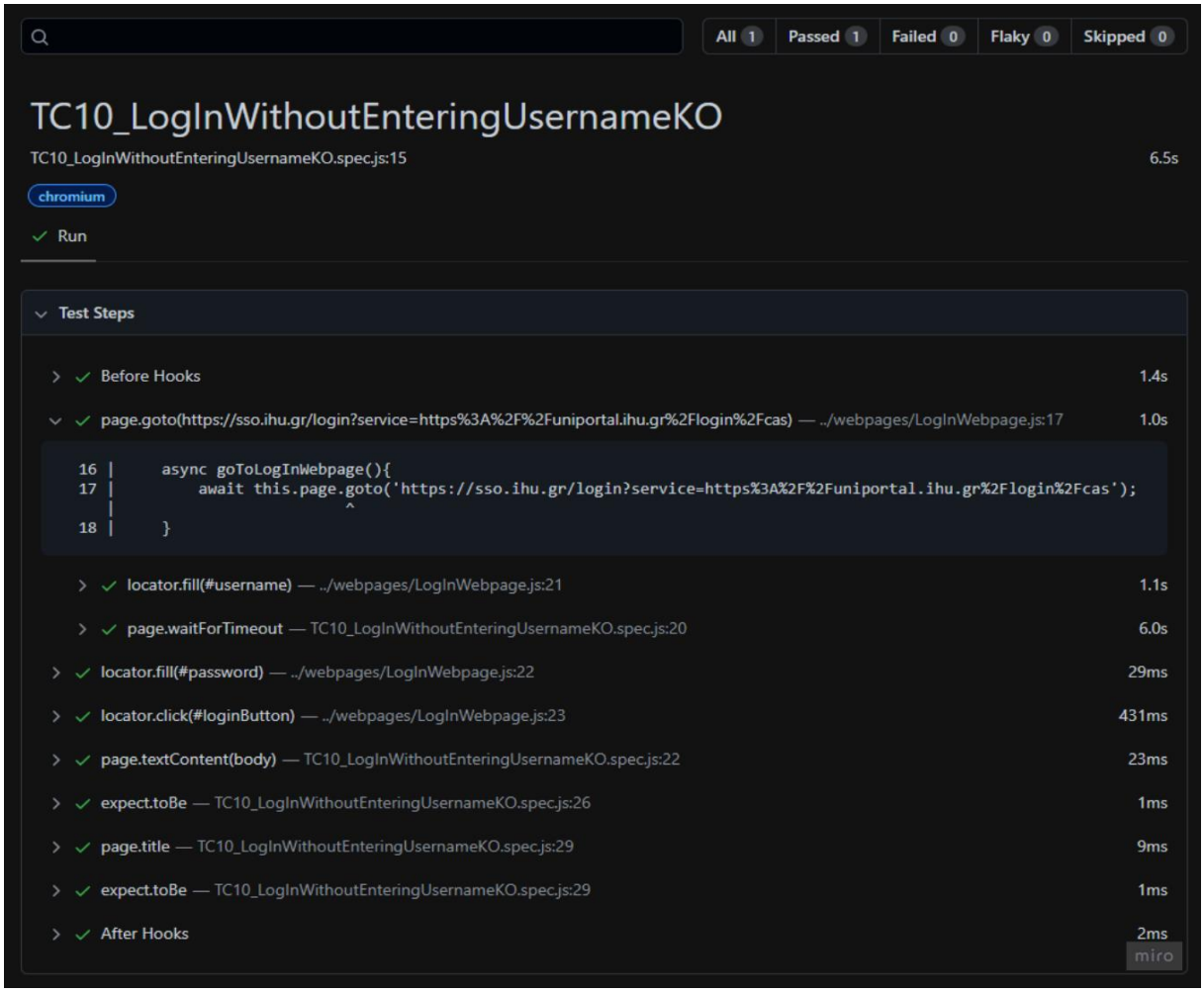
Σχήμα 3.7: Cypress End to End και UI test execution

Έπειτα μέσω των απαραίτητων επιλογών (π.χ. επιλογή browser εκτέλεσης), κάνοντας click πάνω σε ένα test suite ή εφαρμογή του Cypress θα τρέξει το επιλεγμένο test suite. Τέλος για την διεξαγωγή των αποτελεσμάτων της εκτέλεσης ενός Test ή Test Suite χρησιμοποιήθηκε η βιβλιοθήκη «mochawesome». Τα αποτελέσματα της εκτέλεσης ενός Test ή Test Suite είναι διαθέσιμα στο αρχείο «reports/index.html». Στο παρακάτω σχήμα παρουσιάζονται τα αποτέλεσμα της εκτέλεσης του test «TC02_LogOutOK».



Σχήμα 3.8: Cypress End to End και UI test execution report με την χρήση της βιβλιοθήκης «mochawesome»

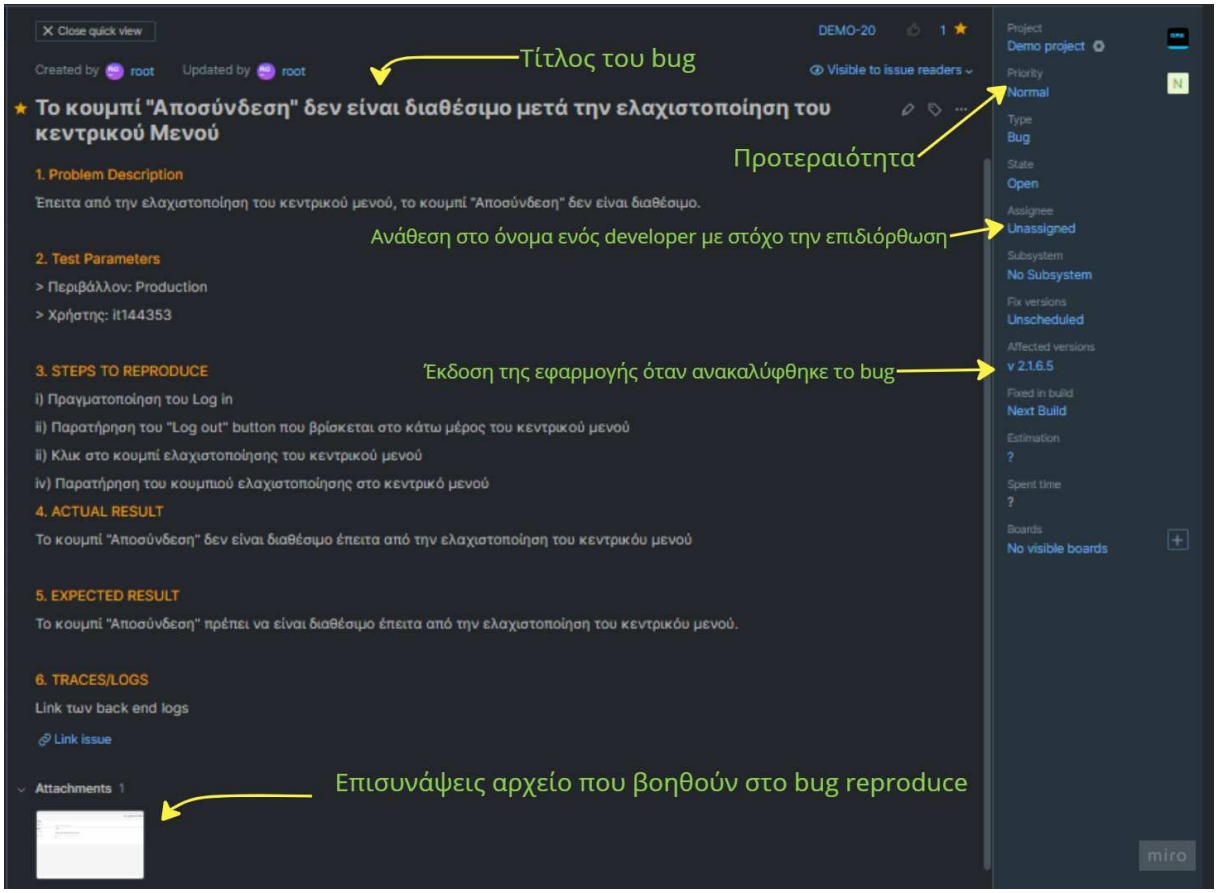
Η εκτέλεση των tests για τα tests που υλοποιήθηκαν με την βοήθεια του εργαλείου Playwright πραγματοποιείται μέσω της χρήσης του CLI(Command Line Interface) στον φάκελο του project. Η εκτέλεση των tests γίνεται μέσω της χρήσης της εντολής «npx playwright test όνομα_του_αρχείου_του_test». Η εντολή της εκτέλεσης μπορεί να περιέχει διάφορα ορίσματα που αφορούν την επιλογή του browser που επιθυμείται να εκτελεστούν τα tests, ή επιλογή του τρόπου εκτέλεσης (headed ή headless). Το Playwright εξ' ορισμού εκτελεί τα tests χωρίς να είναι ορατός ο browser (headless mode). Ένα test ή ένα test suite μπορεί να εκτελεστεί ενώ παράλληλα να είναι ορατός ο browser μέσω της προσθήκης του ορίσματος «--headed» στην εντολή εκτέλεσης. Ένας άλλος τρόπος εκτέλεσης των tests στο project του Playwright είναι η εκτέλεση τους μέσω του Visual Studio Code, πατώντας το πλήκτρο της εκτέλεσης που είναι διαθέσιμο δίπλα από το test function του αρχείου του test. Η εξαγωγή του test execution report πραγματοποιείται αυτόματα με την χρήση του εργαλείου Playwright. Τα αποτελέσματα της εκτέλεσης ενός Test ή Test Suite είναι διαθέσιμα στο αρχείο «playwright-report/index.html». Στο παρακάτω σχήμα παρουσιάζονται τα αποτέλεσμα της εκτέλεσης του test «TC10_LogInWithoutEnteringUsernameKO».



Σχήμα 3.9: Playwright End to End και UI test execution report

Έπειτα από την εκτέλεσης των tests μέσω οποιουδήποτε εργαλείου, εντοπίστηκαν δύο ατέλειες οι οποίες πρέπει να αναφερθούν. Η πρώτη ατέλεια σχετίζεται με την δυσλειτουργία του κεντρικού μενού, όταν αυτό βρίσκεται σε collapsed mode και έχει Medium/Moderate severity. Η σοβαρότητα της

δεύτερης ατέλειας, που σχετίζεται με την απουσία του κουμπιού αποσύνδεσης από το κεντρικό μενού, όταν το κεντρικό μενού βρίσκεται σε «collapsed» mode, ορίζεται επίσης ως Medium. Οι ατέλειες αυτές δημιουργήθηκαν μέσω της χρήσης του bug reporting tool «YouTrack».. Στην αναφορά των ατελειών «bugs», περιέχονται ορισμένα σημασιολογικά πεδία τα για κάθε bug. Το bug reporting tool «YouTrack», ελλιπώς δεν διαθέτει το πεδίο «Severity» για τις ατέλειες που αναφέρονται. Στο παρακάτω σχήμα παρουσιάζεται η αναφορά του πρώτου που εντοπίστηκε bug μέσω της χρήσης του bug reporting tool «YouTrack» καθώς και επεξηγούνται τα σημασιολογικά πεδία.



Σχήμα 3.10: Αναφορά σφάλματος με την χρήση του bug reporting tool «YouTrack»

Τα δύο bugs που εντοπίστηκαν, έπειτα από την αναφορά τους (η οποία γίνεται για λόγους ιχνηλάτησης), πρέπει να διορθωθούν από την ομάδα ανάπτυξης.

3.3.7 Test completion για End to End και UI test types

Η δραστηριότητα του Test completion για τα End to End και UI tests περιλαμβάνει την αποθήκευση και διάθεση των παραπάνω projects στον ιστότοπο του GitHub. Τα παραπάνω projects είναι διαθέσιμα στο Παράρτημα Γ.

Έπειτα από την εκτέλεση των tests με την χρήση οποιουδήποτε εργαλείου αυτοματισμού, έντεκα από τα δεκατρία tests εκτελέστηκαν χωρίς να εντοπίσουν κάποιο bug (Test Passed). Τα υπόλοιπα δύο από τα δεκατρία tests που εκτελέστηκαν, εντόπισαν δύο ατέλειες στον ιστότοπο του «https://uniportal.ihu.gr» (Test Failed). Το test pass ratio που ορίστηκε στο Test plan ως exit criterion ήταν μεγαλύτερο ή ίσο του 80% , εφόσον 11 από 13 tests που εκτελέστηκαν χωρίς να εντοπίσουν κάποια ατέλεια οδηγούν στο test pass ratio του 84.615%. Επίσης τα tests δεν εντόπισαν κάποια ατέλεια της οποίας η σοβαρότητα έχει οριστεί ως «Critical». Εφόσον και τα δύο exit criteria που ορίστηκαν στο

Test Plan πληρούνται, τότε μπορεί να θεωρηθεί ότι ο έλεγχος έχει ολοκληρωθεί καθώς και να σταματήσουν οι σχετικές δραστηριότητες του ελέγχου στην παρούσα χρονική στιγμή. Τέλος το test summary report πρέπει να αποσταλεί στους stakeholders του έργου λογισμικού ώστε να είναι ενήμεροι. Το Test summary report είναι ένα έγγραφο το οποίο πρέπει να αποτελεί το status (pass, failed, blocked, κλπ.) του κάθε test καθώς και να περιέχει μια λεπτομερή αναφορά για τις ατέλειες που εντοπίστηκαν.

3.4 Επίλογος

Στο τρίτο κεφάλαιο της πτυχιακής εργασίας αναλύεται το πώς πραγματοποιείται η διεξαγωγή των δραστηριοτήτων του STLC στον ιστότοπο «https://uniportal.ihu.gr». Αρχικά επιλέγεται και στη συνέχεια εφαρμόζεται η στρατηγική και η προσέγγιση του ελέγχου. Έπειτα διεξάγεται ο στατικός έλεγχος και αναφέρονται τα static bugs που εντοπίστηκαν. Ακόμη, πραγματοποιείται και αναλύεται η διεξαγωγή όλων των δραστηριοτήτων του STLC (και των επιμέρους εργασιών) για τον τύπο του Functional testing με την χρήση συγκεκριμένων εργαλείων για την κάθε δραστηριότητα. Τέλος, η έμφαση δίνεται στην χρήση των τριών δημοφιλέστερων automation testing frameworks (Selenium, Cypress και Playwright).

Κεφάλαιο 4ο: Σύγκριση των automation testing frameworks που χρησιμοποιήθηκαν

Οι επιχειρήσεις / οργανισμοί συγκρίνουν τα automation testing frameworks που υπάρχουν στην αγορά, προκειμένου να καταλήξουν στην επιλογή κάποιου εργαλείου αυτοματισμού των tests το οποίο αρμόζει στις ανάγκες τους και στον προϋπολογισμό που διαθέτουν. Τα automation testing frameworks με τα οποία υλοποιήθηκε η διεξαγωγή των δραστηριοτήτων του Test implementation και του Test execution ελαφρώς διαφέρουν σε μερικά χαρακτηριστικά τα οποία λαμβάνονται σοβαρά υπόψιν από κάθε επιχείρηση / οργανισμό. Επίσης αναφέρονται τα χαρακτηριστικά και αναλύονται τα ξεχωριστά χαρακτηριστικά, πλεονεκτήματα και μειονεκτήματα του κάθε automation testing framework. Η επιλογή του κατάλληλου test automation framework είναι μια κρίσιμη απόφαση για τις επιχειρήσεις που θέλουν να βελτιστοποιήσουν τις δραστηριότητες του ελέγχου στα προϊόντα τους και σε αυτό το κεφάλαιο, αναφέρονται τα χαρακτηριστικά, τα πλεονεκτήματα και τα μειονεκτήματα του κάθε εργαλείου και συγκρίνονται τα δυνατά και τα αδύνατά τους σημεία.

4.1 Χαρακτηριστικά των automation testing frameworks

Τα χαρακτηριστικά των automation testing frameworks παρουσιάζονται αναλυτικά στον παρακάτω πίνακα.

Πίνακας 4.1: Χαρακτηριστικά των automation testing frameworks

Χαρακτηριστικά	Selenium	Cypress	Playwright
Γλώσσες προγραμματισμού	Java, Python, C#, JavaScript, Kotlin, Ruby, Groovy	JavaScript, TypeScript	Java, Python, JavaScript, C#
Συμβατότητα με λειτουργικά συστήματα	Windows, MacOS, Linux	Windows, MacOS, Linux	Windows, MacOS, Linux
Browsers που υποστηρίζονται	Chrome, Firefox, Safari, Microsoft Edge, Internet Explorer	Chrome, Firefox, Microsoft Edge	Chromium, Firefox, Webkit (Safari)
Διαθεσιμότητα	Δωρεάν	Δωρεάν για τον Cypress Runner, με πληρωμή για το Cypress Dashboard	Δωρεάν
Testing frameworks	TestNG, Junit, NUnit, pyTest, κλπ.	Mocha, Jasmine	Jest, Mocha, Jasmine, κλπ.
Ταχύτητα	Μέτρια	Υψηλή	Υψηλή
Κοινότητα υποστήριξης	Μεγάλη	Μικρή	Μικρή
Open source	✓	✓	✓
Cross-browser testing	✓	✓	✓
Headless mode	✓	✓	✓
Παράλληλη εκτέλεση	✓	✓	✓
Auto-wait	X	✓	✓
Built-in debuggers	X	✓	✓

4.2 Ξεχωριστά χαρακτηριστικά του κάθε automation testing framework

Στην συνέχεια ακολουθούν μερικά ιδιαίτερα χαρακτηριστικά που διαθέτει το κάθε automation testing framework τα οποία λείπουν από τα υπόλοιπα. Καθώς αυτές οι τρεις συγκεκριμένες τεχνολογίες αναπτύσσονται διαρκώς, είναι πιθανόν κάποιο από τα παρακάτω χαρακτηριστικά να μην είναι διαθέσιμο στο μέλλον ή να διαφοροποιηθεί.

4.2.1 Selenium

Η αρχική έκδοση του Selenium αναπτύχθηκε από τον Jason Huggins το 2004 ως ένα εσωτερικό εργαλείο της επιχείρησης ToughtWorks. Είναι ένα αρκετά εξελιγμένο automation framework το οποίο παρέχει ένα ευρύ σύνολο εργαλείων και βιβλιοθηκών για την αυτοματοποίηση των browsers, καθώς επίσης υποστηρίζει τις περισσότερες γλώσσες προγραμματισμού. Παρακάτω αναφέρονται και εξηγούνται τα ξεχωριστά χαρακτηριστικά του automation framework Selenium σε συνδυασμό με το testing framework TestNG.

- Distributed testing (κατανεμημένες δοκιμές): Μέσω της χρήσης του εργαλείου Selenium Grid είναι δυνατόν να κατανεμηθούν οι δοκιμές σε πολλαπλές συσκευές και σε πολλαπλούς browsers ταυτόχρονα. Αυτό το ξεχωριστό χαρακτηριστικό είναι ιδιαίτερα πολύτιμο για την παράλληλη εκτέλεση δοκιμών μεγάλης κλίμακας, μειώνοντας σημαντικά τον χρόνο εκτέλεσης των tests. Ενώ το Cypress και το Playwright υποστηρίζουν την παράλληλη εκτέλεση, το Selenium Grid είναι μια αποκλειστική λύση για κατανεμημένες δοκιμές.
- Συγκεκριμένες ενέργειες των browsers: Το Selenium παρέχει πιο εκτεταμένη υποστήριξη για συγκεκριμένες ενέργειες και συμπεριφορές που διαθέτουν οι browsers. Για παράδειγμα είναι δυνατόν να αυτοματοποιηθούν οι αλληλεπιδράσεις για τα αναδυόμενα παράθυρα, τον χειρισμό πιστοποιητικών και τις επεκτάσεις του περιηγητή (browser extensions).
- Ισχυρό οικοσύστημα και ενσωματώσεις: Το Selenium μιας και είναι ένα από τα πρωτοεμφανιζόμενα automation frameworks διαθέτει ένα καλά εδραιωμένο οικοσύστημα ενσωματώσεων και εργαλείων τρίτων. Αυτό το οικοσύστημα περιλαμβάνει εργαλεία για την διαχείριση των δοκιμών, την αναφορά, τις ενσωματώσεις με εργαλεία CI/CD, κλπ.
- Selenium IDE: Το Selenium IDE είναι ένα εργαλείο το οποίο βοηθάει τα άτομα που αναπτύσσουν τα tests μέσω της δυνατότητας record-and-playback. Αυτή η δυνατότητα επιτρέπει αρχικά την εγγραφή των actions του χρήστη και έπειτα την δημιουργία του test script σε γλώσσα της επιλογής του χρήστη η οποία φυσικά υποστηρίζεται από το Selenium.

4.2.2 Cypress

Η αρχική έκδοση του Cypress ιδρύθηκε και κυκλοφόρησε στην αγορά του λογισμικού τον Σεπτέμβρη του 2017 από τον Brian Mann. Είναι ένα σύγχρονο test automation framework το οποίο έχει σχεδιαστεί ειδικά για τον έλεγχο των End to End tests για τον ιστό. Επίσης επικεντρώνεται στην απλότητα και στην εκτέλεση των tests σε πραγματικό χρόνο. Η εκτέλεση των tests σε πραγματικό χρόνο, διευκολύνει την ανάπτυξη των tests καθώς εμπεριέχει επιπλέον βοηθητικά χαρακτηριστικά. Παρακάτω αναφέρονται και εξηγούνται τα κυριότερα ξεχωριστά χαρακτηριστικά του automation testing framework Cypress.

- Επαναφόρτωση σε πραγματικό χρόνο (Real-time reload): Το Cypress παρέχει προεπισκόπηση του browser σε πραγματικό χρόνο και αυτόματη επαναφόρτωση όταν τα test scripts τροποποιούνται, καθιστώντας την αποσφαλμάτωση και την ανάπτυξη των tests ευκολότερη για τους QAE.
- Αποσφαλμάτωση με δυνατότητα επιλογής ενός σημείου της εκτέλεσης (Time-travel debugging): Οι QAE μπορούν να περιηγηθούν στην εκτέλεση των tests και να δουν την κατάσταση της εφαρμογής σε διάφορα σημεία στον χρόνο, γεγονός που επίσης βοηθάει στην αποσφαλμάτωση των test scripts.

- Αποσφαλμάτωση σε πραγματικό χρόνο (Real-time debugging): Το Cypress παρέχει τη δυνατότητα αποσφαλμάτωσης των δοκιμών σε πραγματικό χρόνο. Η εκτέλεση των δοκιμών μπορεί να διακοπεί ανά πάσα στιγμή έτσι ώστε να εξεταστεί η τρέχουσα κατάσταση της διαδικτυακής εφαρμογής, ενισχύοντας έτσι την αποσφαλμάτωση.
- Built-in test runner (Ενσωματωμένος Test Runner): Το Cypress περιλαμβάνει έναν built-in test runner ο οποίος παρέχει λεπτομερείς καταγραφές και βίντεο της εκτέλεσης των tests. Αυτό το χαρακτηριστικό είναι εξαιρετικά χρήσιμο για τους QAE σε περιπτώσεις που εντοπίζονται ατέλειες στο λογισμικό διότι το αρχείο του βίντεο της εκτέλεσης είναι απευθείας διαθέσιμο για αποστολή προς ολόκληρη την ομάδα.
- Διαδραστικός built-in test runner: Το Cypress παρέχει έναν διαδραστικό test runner, μέσω του οποίου η κατάσταση της εφαρμογής υπό έλεγχο είναι ορατή καθώς η εφαρμογή εκτελείται. Αυτή η δυνατότητα βοηθάει στην κατανόηση της συμπεριφοράς των tests, καθώς ο χρήστης του ελέγχου μπορεί να δει την κατάσταση της εφαρμογής υπό έλεγχο, έπειτα από την εκτέλεση του κάθε βήματος σε πραγματικό χρόνο.
- Network stubbing και Network Spying: Το Cypress παρέχει ενσωματωμένη υποστήριξη για stubbing και κατασκοπεία των αιτημάτων του διαδικτύου. Αυτό δίνει την δυνατότητα παρακολούθησης και διαχείρισης των HTTP (Hypertext Transfer Protocol) requests και των responses, διευκολύνοντας έτσι τον έλεγχο των πολύπλοκων tests.
- Αυτόματη επανάληψη εκτέλεσης των δοκιμών. Το Cypress μπορεί να επανεκτελέσει αυτόματα τα test που αποτυγχάνουν (test failed), βοηθώντας έτσι την εξάλειψη του flakiness που μπορεί να προκληθεί από παροδικά προβλήματα.

4.2.3 Playwright

Η αρχική έκδοση του Playwright ανακοινώθηκε και κυκλοφόρησε στην αγορά του λογισμικού τον Γενάρη του 2020 από την Microsoft. Είναι ένα σύγχρονο test automation framework το οποίο παρέχει μια ενιαία διεπαφή προγραμματισμού εφαρμογών (single API) για την αυτοματοποίηση των tests. Παρακάτω αναφέρονται τα κύρια χαρακτηριστικά του test automation framework Playwright.

- Ενιαία διεπαφή προγραμματισμού εφαρμογών (Single API): Το single API που παρέχει το Playwright χρησιμοποιείται για το automation testing των browser εφαρμογών που τρέχουν σε διάφορες πλατφόρμες. Αυτό σημαίνει ότι είναι δυνατόν να χρησιμοποιηθεί το ίδιο API για το automation testing των εφαρμογών που τρέχουν για browsers ανεξάρτητα από το εάν ο browser τρέχει σε κάποιο από τα λειτουργικά συστήματα Windows, MacOS και Linux ή από το εάν ο browser τρέχει σε Android ή iOS (iPhone Operating System). Αυτή η ενοποιημένη προσέγγιση καθιστά το Playwright ευέλικτο και ιδανικό για επιχειρήσεις και για τις αναπτυσσόμενες ομάδες που διεξάγουν τον έλεγχο, καθώς μπορούν να χρησιμοποιούν την ίδια τεχνολογία για την ανάπτυξη των tests για τον browser που μπορεί να τρέχει σε διάφορες πλατφόρμες. Μέσω του Single API, ο ίδιος πηγαίος κώδικας (source code) μπορεί να χρησιμοποιηθεί για τον έλεγχο των εφαρμογών που τρέχουν σε διάφορες πλατφόρμες, χωρίς να υπάρχει ανάγκη για σημαντικές αλλαγές στον κώδικα.
- Έλεγχος σε διάφορες πλατφόρμες (Cross-platform testing): Η διαδικασία του ελέγχου γίνεται σημαντικά πιο αποδοτική και η δυνατότητα επαναχρησιμοποίησης των tests επιτρέπει να ελεγχθούν εφαρμογές που έχουν αναπτυχθεί για να τρέξουν σε browsers που χρησιμοποιούνται από διαφορετικά λειτουργικά συστήματα και συσκευές.
- Browser contexts (στιγμιότυπα των περιηγητών): Δυνατότητα δημιουργίας μεμονωμένων στιγμιότυπων των browsers, όπου το καθένα μπορεί να περιέχει ξεχωριστά cookies, χώρο αποθήκευσης (storage) και δικαιώματα (permissions).
- Προηγμένες δυνατότητες αυτοματισμού: Το Playwright παρέχει προηγμένες δυνατότητες αυτοματισμού όπως η παρέμβαση σε δικτυακά αιτήματα (network requests).
- CodeGen: Αυτό το ξεχωριστό χαρακτηριστικό του Playwright επιτρέπει την καταγραφή των ενεργειών του χρήστη στον ιστότοπο και έπειτα την παραγωγή του αντίστοιχου κώδικα. Αυτό το χαρακτηριστικό είναι παρόμοιο με το Selenium IDE του Selenium, αλλά ελαφρώς διαφέρει.

4.3 Πλεονεκτήματα του κάθε automation testing framework

Μερικά από τα σημαντικότερα πλεονεκτήματα του Selenium σε αντίθεση με το Cypress και το Playwright είναι ότι το Selenium υποστηρίζει τις περισσότερες γλώσσες προγραμματισμού για την ανάπτυξη των tests καθώς και ότι τα tests μπορούν να εκτελεστούν σε μεγαλύτερο σύνολο περιηγητών. Το παραπάνω πλεονέκτημα καθιστά το Selenium πιο ευέλικτο. Ένα ακόμη βασικό πλεονεκτήματα του Selenium σε σχέση με το Cypress και το Playwright είναι ότι διαθέτει μια μεγάλη και ενεργή κοινότητα χρηστών με αποτέλεσμα να υπάρχουν εκτενείς τεκμηριώσεις, άρθρα και forums. Αυτό το γεγονός, βοηθάει σημαντικά τους χρήστες στην αντιμετώπιση προβλημάτων κατά την διάρκεια ανάπτυξης των tests. Επίσης ένα πλεονέκτημα του Selenium σε σύγκριση με το Cypress και το Playwright, είναι ότι το Selenium παρέχει εκτεταμένη υποστήριξη για παλαιότερα προγράμματα περιήγησης όπως για παράδειγμα ο Internet Explorer.

Για το test automation framework Cypress τα προτερήματα σε σύγκριση με το Selenium και το Playwright αποτελούν κυρίως τα ξεχωριστά χαρακτηριστικά του τα οποία αναφέρθηκαν παραπάνω. Το Cypress πλεονεκτεί σε σχέση με το Selenium σε αρκετά ακόμη χαρακτηριστικά τα οποία αναφέρονται παρακάτω.

Τα πλεονεκτήματα του test automation framework Playwright σε σύγκριση με το Selenium και το Cypress είναι τα ξεχωριστά χαρακτηριστικά που διαθέτει.

Τα κύρια πλεονεκτήματα του Cypress σε σύγκριση με το Playwright, εκτός από τα ξεχωριστά χαρακτηριστικά του, είναι ότι το Cypress έχει αναπτυχθεί εστιάζοντας στις μοντέρνες διαδικτυακές εφαρμογές και ότι το παρέχει μια φιλική εμπειρία ανάπτυξης προς τους προγραμματιστές (developer-friendly) των tests για SPA (Single Page Applications).

Το προνόμια του Cypress και του Playwright σε σύγκριση με το Selenium είναι ότι τα πρώτα δύο εργαλεία διαθέτουν πιο γρήγορη εκτέλεση των tests και ότι διαθέτουν την λειτουργία της αυτόματης αναμονής μεταξύ των βημάτων για κάθε φορά που αναμένονται να γίνουν διαθέσιμα τα web elements. Επιπλέον, η εγκατάσταση των Cypress και Playwright είναι πολύ πιο εύκολη σε σύγκριση με την περισσότερο πολύπλοκη εγκατάσταση του Selenium. Επίσης διαθέτουν built-in λειτουργίες αποσφαλμάτωσης των test scripts και native υποστήριξη για τα Chrome Devtools. Αυτό είναι επωφελές για τον εντοπισμό σφαλμάτων και την αντιμετώπιση προβλημάτων στα test scripts. Ένα ακόμη βασικό πλεονέκτημα του Cypress και του Playwright σε σύγκριση με το Selenium είναι ότι τα πρώτα δύο test automation frameworks διαθέτουν τις δυνατότητες καταγραφής βίντεο και στιγμιότυπων οθόνης του test προς εκτέλεση.

Ένα βασικό πλεονέκτημα του Playwright απέναντι στο Cypress, πέρα από τα ξεχωριστά χαρακτηριστικά του, είναι ότι υποστηρίζει περισσότερες γλώσσες προγραμματισμού για την ανάπτυξη των tests. Επίσης το Playwright υποστηρίζει την εκτέλεση των tests μέσω της παράλληλης εκτέλεσης των browsers (parallel browser testing) ενώ το Cypress δεν υποστηρίζει αυτήν τη δυνατότητα. Επιπλέον το Playwright υποστηρίζει τους XPath locators χωρίς την χρήση κάποιου plug-in. Τέλος οι μηχανισμοί της αυτόματης αναμονής και του network spying και network stubbing του Playwright είναι πιο προηγμένοι από αυτούς του Cypress [44].

4.4 Μειονεκτήματα του κάθε automation testing framework

Μερικά από τα σημαντικότερα μειονεκτήματα του Selenium σε σύγκριση με τα Cypress και Playwright είναι ότι η εκτέλεση των tests είναι πιο αργή και ότι η ανάπτυξη των συγκεκριμένων λειτουργιών είναι αρκετά πιο περίπλοκη καθώς απαιτείται η χρήση εξωτερικών βιβλιοθηκών (external libraries). Επίσης

η εξάρτηση από αρκετές εξωτερικές βιβλιοθήκες καθιστά το Selenium πιο περίπλοκο στην εγκατάσταση και λιγότερο σταθερό σε περίπτωση που κάποιο από τις εξωτερικές βιβλιοθήκες δυσλειτουργεί. Μια ακόμη αδυναμία του Selenium είναι ότι δεν διαθέτει την λειτουργία της αυτόματης αναμονής. Ένα ακόμη μειονέκτημα του Selenium είναι ότι δεν διαθέτει built-in debuggers. Τέλος η υποστήριξη πολλαπλών γλωσσών μπορεί σε συγκεκριμένες περιπτώσεις να καταστεί προβληματική. Μια τέτοια περίπτωση μπορεί να είναι για παράδειγμα μια ομάδα στην οποία ο κάθε QAE χρησιμοποιεί διαφορετική γλώσσα προγραμματισμού για την ανάπτυξη των tests. Η παραπάνω περίπτωση οδηγεί σε ασυνεπείς τακτικές της ανάπτυξης των tests (scripting) σε μία ομάδα ή ένα project που μπορεί να δυσχεραίνουν την ενημέρωση ή τροποποίηση των test scripts.

Για το test automation framework Cypress τα μειονεκτήματα σε σύγκριση με το Selenium και το Playwright αποτελούν την υποστήριξη σε λιγότερους browsers, την υποστήριξη σε λιγότερες γλώσσες προγραμματισμού και τη συνεργασία με πολλά testing frameworks. Επίσης η έλλειψη της δυνατότητας του ελέγχου σε περιηγητές κινητών συσκευών (mobile browser testing) αποτελεί μία ακόμη αδυναμία. Επιπλέον, το Cypress δεν έχει την δυνατότητα αυτόματης παραγωγής κώδικα σύμφωνα με τα actions του χρήστη, σε αντίθεση με το Selenium και το Playwright. Το Playwright διαθέτει το χαρακτηριστικό CodeGen ενώ το Selenium το χαρακτηριστικό Selenium IDE, τα οποία μπορούν να χρησιμοποιηθούν για την διευκόλυνση της ανάπτυξης των tests [45]. Τέλος το μικρό εύρος των ενσωματώσεων (integrations) και των εξωτερικών βιβλιοθηκών είναι ακόμη ένα μειονέκτημα του Cypress.

Μερικά από τα σημαντικότερα μειονεκτήματα του Playwright σε σύγκριση με το Cypress και του Selenium είναι ότι η περίοδος εκμάθησης του είναι αρκετά μεγάλη λόγω της πολυπλοκότητας και του εκτενούς εύρους των χαρακτηριστικών του, γεγονός το οποίο μπορεί να καθυστερήσει την ανάπτυξη των tests για ένα έργο λογισμικού. Μια ακόμη ατέλεια του Cypress, λόγω της ευρείας γκάμας των χαρακτηριστικών του, είναι ότι απαιτεί περισσότερους πόρους (resource-intensive) από το Selenium και το Playwright.

Τα κύρια μειονεκτήματα του Cypress σε σύγκριση με το Playwright είναι ότι το Cypress υποστηρίζεται μόνο από τις γλώσσες προγραμματισμού JavaScript και TypeScript. Επίσης το automation testing framework Cypress δεν υποστηρίζει το Safari browser. Μια ακόμη αδυναμία του Cypress είναι ότι η υποστήριξη του locator XPath υποστηρίζεται μόνο μέσω plug-in.

Οι αδυναμίες του Cypress και του Playwright απέναντι στο Selenium, είναι ότι τα πρώτα δύο test automation frameworks έχουν μικρότερη κοινότητα. Επίσης δεν υποστηρίζουν τόσες πολλές γλώσσες προγραμματισμού και browsers. Ακόμη το Cypress και το Playwright δεν μπορούν να χρησιμοποιήσουν τόσα testing frameworks όσα το Selenium.

4.5 Επιλογή του κατάλληλου test automation framework

Όπως διαπιστώνεται το κάθε test automation framework έχει τα δικά του δυνατά και αδύνατα σημεία. Επικρατούν αρκετές απόψεις για το ποιο test automation testing είναι το καλύτερο μεταξύ των τριών που χρησιμοποιήθηκαν. Στην πραγματικότητα δεν υπάρχει το τέλειο εργαλείο το οποίο πρέπει να προτιμηθεί από κάθε επιχείρηση ή οργανισμό. Η επικέντρωση της προσοχής από την μεριά των επιχειρήσεων και των οργανισμών πρέπει να στραφεί στην αναζήτηση του κατάλληλου εργαλείου ανάλογα με τις ανάγκες της, την αρχιτεκτονική και το περιεχόμενο του προϊόντος λογισμικού. Αρχικά πρέπει να γίνει μια ανάλυση για το ποια χαρακτηριστικά ή σύνολο χαρακτηριστικών των παραπάνω εργαλείων αρμόζουν και ενδιαφέρουν τις επιχειρήσεις και οργανισμούς σύμφωνα με τα τεχνικά χαρακτηριστικά των προϊόντων και των υπηρεσιών που προσφέρουν αλλά και με τους πόρους (ανθρώπινο δυναμικό, προϋπολογισμός, χρόνος για διάθεση του προϊόντος στην αγορά) που διαθέτουν.

Αυτή η ανάλυση πρέπει να λάβει σοβαρά υπόψιν και την δυνατότητα ενσωμάτωσης του automation testing framework με τα υπόλοιπα εργαλεία που χρησιμοποιεί για να υλοποιήσει τον έλεγχο αλλά και για να αναπτύξει και να εξελίξει τα προϊόντα λογισμικού της. Εφόσον λάβει μέρος αυτή η ανάλυση και η ιεράρχηση των προτεραιοτήτων, τότε μπορεί να ακολουθήσει η συγκριτική ανάλυση προκειμένου να ληφθεί η σωστή απόφαση για το ποιο test automation framework ταιριάζει περισσότερο στις ανάγκες μιας επιχείρησης ή ενός οργανισμού. Τέλος, αρκετές επιχειρήσεις (κυρίως αυτές που διαθέτουν πολύπλοκα προϊόντα), επιλέγουν να χρησιμοποιούν έναν συνδυασμό από test automation frameworks προκειμένου να καλύψουν όλες τις ανάγκες του ελέγχου για ένα προϊόν, ακόμη και όταν ο έλεγχος διεξάγεται στο ίδιο test type και στο ίδιο test level.

4.6 Επίλογος

Στο τέταρτο κεφάλαιο της πτυχιακής εργασίας αναφέρονται και συγκρίνονται τα χαρακτηριστικά των automation testing frameworks που χρησιμοποιήθηκαν για την διεξαγωγή ελέγχου. Επίσης αναφέρονται και αναλύονται τα ξεχωριστά χαρακτηριστικά τους, δηλαδή τα χαρακτηριστικά που είναι διαθέσιμα μόνο για ένα συγκεκριμένο εργαλείο. Επίσης αναφέρονται μερικές επιπρόσθετες πληροφορίες σχετικά με το κάθε test automation framework. Στην συνέχεια αναφέρονται αναλυτικά τα πλεονεκτήματα και τα μειονεκτήματα του κάθε εργαλείου σε σύγκριση με τα υπόλοιπα. Ακόμη, αναφέρεται ότι δεν υπάρχει ένα τέλειο test automation framework για κάθε επιχείρηση και προϊόν λογισμικού, καθώς το κάθε εργαλείο διαθέτει ένα διαφορετικό σύνολο χαρακτηριστικών. Τέλος, αποσαφηνίζεται η διαδικασία που πρέπει να ακολουθήσει μια επιχείρηση ή ένας οργανισμός, προκειμένου να προβεί στη σωστή επιλογή του κατάλληλου εργαλείου που αρμόζει στις ανάγκες, τους πόρους και τα προϊόντα της.

Κεφάλαιο 5ο: Συμπεράσματα και επεκτάσεις

Τα συμπεράσματα που προκύπτουν από την παρούσα πτυχιακή εργασία τα πεδία αξιοποίησης και οι πιθανές επεκτάσεις αναφέρονται αναλυτικά στο παρόν κεφάλαιο.

Στο πλαίσιο αυτής της πτυχιακής εργασίας μελετήθηκε εκτενώς η αναγκαιότητα της διασφάλισης της ποιότητας και του ελέγχου λογισμικού για τις επιχειρήσεις και τους οργανισμούς. Τα οφέλη του ελέγχου και η συμβολή του ελέγχου στην επιτυχία των προϊόντων λογισμικού διαδραματίζουν καθοριστικό ρόλο. Οι επιπτώσεις ελλιπούς ή μηδαμινού ελέγχου λογισμικού μπορούν να προκαλέσουν καταστροφικές συνέπειες για τις ανθρώπινες ζωές (για συγκεκριμένα είδη εφαρμογών), αλλά και για την ύπαρξη και την φήμη μιας επιχείρησης ή ενός οργανισμού. Για αυτόν τον λόγο οι επιχειρήσεις και οι οργανισμοί θα πρέπει να εστιάσουν την προσοχή και τους πόρους τους (προϋπολογισμός, ανθρώπινο δυναμικό, χρόνος, κλπ.) όχι μόνο στην παραγωγή αλλά και στην διασφάλιση της ποιότητας των προϊόντων λογισμικού που διαθέτουν. Η ποιότητα των προϊόντων λογισμικού επίσης πρέπει να αποτελείται στη λίστα των κύριων στόχων όλων των stakeholders που δουλεύουν σε ένα έργο λογισμικού ανεξαρτήτως από τον ρόλο τους.

Διαπιστώνεται ότι, οι κύριες αιτίες των ατελειών του λογισμικού βασίζονται στον ανθρώπινο παράγοντα ο οποίος είναι επιρρεπής σε σφάλματα, στην πίεση χρόνου ανάπτυξης του λογισμικού, στην πολυπλοκότητα του σχεδιασμού του λογισμικού ή ακόμη στην κακή επικοινωνία μεταξύ των συμμετεχόντων στο έργο. Επίσης η διασφάλιση της ποιότητας και ο έλεγχος του λογισμικού επιλύει σημαντικά προβλήματα τα οποία προκύπτουν κατά την διάρκεια ανάπτυξης του λογισμικού, τα οποία είναι κρίσιμο να αντιμετωπιστούν.

Είναι θετικό το γεγονός ότι, διάφοροι σημαντικοί παράγοντες στον τομέα της διασφάλισης ποιότητας λογισμικού (π.χ. ISTQB) έχουν δομήσει το θεωρητικό και το τεχνολογικό υπόβαθρο που απαιτείται προκειμένου οι QAE και οι υπόλοιποι stakeholders ενός έργου λογισμικού, να μπορούν να χρησιμοποιούν μια κοινή γλώσσα επικοινωνίας αλλά και να μπορούν να εξελίξουν τις γνώσεις τους πάνω στους τομείς της διασφάλισης ποιότητας του λογισμικού και του ελέγχου.

Επίσης το debugging και το testing πρόκειται για δύο διαφορετικές ενέργειες οι οποίες δεν πρέπει να συγχέονται μεταξύ τους. Το debugging είναι ενέργεια που πραγματοποιούν οι developers προκειμένου να βρουν το root cause μιας ατέλειας, ενώ το testing είναι ενέργεια που κυρίως πραγματοποιείται από τους QAE. Κατά την δραστηριότητα του test implementation του STLC, οι QAE μπορεί να προβούν σε ενέργειες debugging το οποίο θα λάβει μέρος στα test scripts που αναπτύσσουν. Σε εξαιρετικά σπάνιες περιπτώσεις σε μερικές επιχειρήσεις ή οργανισμούς, οι QAEs μπορούν να εμπλακούν και να βοηθήσουν τους developers με το debugging ακόμη και στο λογισμικό που αναπτύσσεται (δηλαδή όχι μόνο στα test scripts).

Οι εφτά βασικές αρχές του λογισμικού εστιάζουν στο γεγονός ότι όσο σωστός και πλήρης είναι ο έλεγχος στο λογισμικό, η ενέργεια της διεξαγωγής ελέγχου δεν πρέπει να λαμβάνεται υπόψιν ως απόδειξη ορθότητας εφόσον ο έλεγχος διεξάγεται για να εντοπίσει ατέλειες αλλά όχι για να αποδείξει την απουσία τους.

Ο έλεγχος μπορεί να διεξαχθεί με δύο τρόπους. Ο πρώτος τρόπος είναι να διεξαχθεί χειροκίνητα (manual testing) και ο άλλος τρόπος είναι να διεξαχθεί αυτοματοποιημένα (automation testing). Ο ένας τρόπος χρησιμοποιείται για να συμπληρώσει τον άλλον και έτσι και οι δύο τρόποι είναι απαραίτητοι.

Διαφορετικά εργαλεία και ρόλοι μεταξύ των μελών μια ομάδας δραστηριοποιούνται προκειμένου να διεξαχθεί ο έλεγχος στα διαφορετικά επίπεδα και είδη και υποείδη του ελέγχου.

Διάφορα εργαλεία μπορούν να χρησιμοποιηθούν για την διαχείριση των tests (test management) και για την αναφορά των ατελειών (bug reporting) που εντοπίζονται στο λογισμικό. Τα εργαλεία αυτά δεν έχουν σημαντικές διαφορές πέρα από τη δυνατότητα ενσωμάτωσης με άλλα εργαλεία που χρησιμοποιούνται για την συνεργασία των ατόμων της ομάδας και με εργαλεία που χρησιμοποιούνται για τη διαχείριση του έργου λογισμικού.

Η πυραμίδα του ελέγχου (test pyramid) αποτελεί κατευθυντήρια γραμμή για αρκετές επιχειρήσεις και οργανισμούς για το πώς πρέπει να μοιράζονται ποσοστιαία τα tests στα επίπεδα του ελέγχου. Παρ' όλα αυτά, η πυραμίδα του ελέγχου μπορεί να μην αποτελεί πάντα την κατάλληλη λύση για αρκετές επιχειρήσεις και οργανισμούς.

Οι διάφορες τεχνικές ελέγχου πρέπει να συνδυαστούν, έτσι ώστε να αντληθούν αποτελεσματικότερα τα test cases από τις προδιαγραφές τις τεκμηριώσεις και τα σχέδια που περιγράφουν τη λειτουργίας των προϊόντων.

Η σωστή επιλογή της στρατηγικής ή του συνόλου στρατηγικών ελέγχου πρέπει να ληφθεί υπόψιν βάση των χαρακτηριστικών και της φύσης των προϊόντων λογισμικού που διαθέτει μια επιχείρηση ή ένας οργανισμός.

Η δημιουργία ξεχωριστών περιβάλλοντων στα οποία πρέπει να διεξάγεται ο έλεγχος είναι απαραίτητα για τις επιχειρήσεις και τους οργανισμούς, έτσι ώστε να αποφευχθεί η πιθανότητα κατάρρευσης του πραγματικού περιβάλλοντος (production environment) που χρησιμοποιεί το προϊόν. Αυτά τα ξεχωριστά περιβάλλοντα, πρέπει να αποτελούν κλώνους-απομιμήσεις του πραγματικού περιβάλλοντος.

Οι διεξαγωγή των δραστηριοτήτων του STLC και των επιμέρους εργασιών του, απαιτεί μια μεγάλη προσπάθεια η οποία πρέπει να συγκεντρώνεται σε συγκεκριμένα επίπεδα και τύπους του ελέγχου. Οι δραστηριότητες του test design, του test implementation και του test execution αποτελούν τις πιο χρονοβόρες δραστηριότητες του κύκλου ζωής του ελέγχου καθώς σε αυτές τις δραστηριότητες χρησιμοποιείται το μεγαλύτερο πλήθος εργαλείων, θεωρητικών, τεχνικών αλλά και εμπειρικών γνώσεων.

Τα automation testing frameworks που είναι διαθέσιμα για το End to End testing ποικίλουν αλλά τα Selenium, Cypress και Playwright έχουν ξεχωρίσει, καθώς είναι τα δημοφιλέστερα λόγω της πληθώρας των χαρακτηριστικών του και των πλεονεκτημάτων τους απέναντι σε άλλα εργαλεία που είναι διαθέσιμα για την ίδια χρήση. Το καθένα από αυτά τα τρία automation testing frameworks έχει τα δικά του πλεονεκτήματα και μειονεκτήματα. Αυτά τα εργαλεία συνεχώς εξελίσσονται καθώς και συγκρίνονται μεταξύ τους για την θέση του καταλληλότερου εργαλείου που θα επιλεχθεί από τις επιχειρήσεις και από τους οργανισμούς προκειμένου να διεξάγουν τον έλεγχο στις διαδικτυακές τους εφαρμογές.

5.1 Πεδία αξιοποίησης της πτυχιακής εργασίας

Η παρούσα πτυχιακή εργασία μπορεί να χρησιμοποιηθεί ως πηγή γνώσης για τους φοιτητές που επιθυμούν να αποκτήσουν μια ολιστική εικόνα της διασφάλισης της ποιότητας του λογισμικού και του ελέγχου λογισμικού, αλλά και για τους φοιτητές που επιθυμούν να επενδύσουν πάνω στους συγκεκριμένους τομείς – δύο τομείς οι οποίοι έχουν υψηλή ζήτηση στην αγορά εργασίας.

Αξιοποίηση του πηγαίου κώδικα της εργασίας από την ομάδα ανάπτυξης της διαδικτυακής εφαρμογής του «https://uniprotal.ihu.gr» για μελλοντικές εκδόσεις. Η ομάδα ανάπτυξης της διαδικτυακής εφαρμογής «https://uniprotal.ihu.gr», μπορεί να χρησιμοποιήσει τα αναπτυγμένα tests προκειμένου να διεξάγει regression testing ή ακόμα και να προσθέσει καινούρια tests στο test suite.

5.2 Επεκτάσεις

- Υλοποίηση του ελέγχου στον ιστότοπο του «https://uniportal.ihu.gr» σε διαφορετικά επίπεδα, είδη και υποείδη του ελέγχου με την χρήση αντίστοιχων εργαλείων.
- Ανάπτυξη ακόμη περισσότερων test cases και test scripts με την χρήση των ίδιων test automation frameworks.
- Τα test cases που χρησιμοποιήθηκαν, μπορούν να αναπτυχθούν σε διαφορετικά test automation frameworks (πέρα από αυτών που χρησιμοποιήθηκαν στην παρούσα εργασία), για τα ίδια επίπεδα και ίδιους τύπους του ελέγχου. Τα χαρακτηριστικά, τα πλεονεκτήματα και τα μειονεκτήματα των νέων εργαλείων μπορούν να προστεθούν στη σύγκριση με τα εργαλεία που χρησιμοποιήθηκαν ήδη. Μερικά από αυτά τα εργαλεία μπορούν να είναι τα εξής: Protractor, Puppeteer, Robot framework, κλπ.
- Ενσωμάτωση (integration) των automation testing frameworks που χρησιμοποιήθηκαν με άλλα project management ή / και test management tools. Η εν λόγω ενσωμάτωση θα καταστήσει τα test automation frameworks που χρησιμοποιήθηκαν ως εργαλεία τα οποία μπορούν να εκτελέσουν και να βοηθήσουν διαδικασίες που σχετίζονται με την διαχείριση του έργου λογισμικού (π.χ. ορατότητα της προόδου του test planning ή του test implementation για ένα feature, απευθείας μέσω του στοιχείου ιχνηλάτησης / JIRA ticket). Μερικά από τα δημοφιλέστερα test management και project management tools με τα οποία μπορούν να ενσωματωθούν τα test automation frameworks που χρησιμοποιήθηκαν είναι τα εξής: JIRA, Xray, Zephyr, TestRail, TestCollab, qTest, PractiTest, κλπ.
- Ενσωμάτωση (integration) των automation testing frameworks με CI/CD tool. Με τη συγκεκριμένη ενσωμάτωση, η εκτέλεση των tests με την χρήση των εργαλείων Selenium, Cypress και Playwright θα αυτοματοποιηθεί κάθε φορά που θα πραγματοποιείται μια αλλαγή στον κώδικα του λογισμικού του προϊόντος από την ομάδα ανάπτυξης. Αυτό θα έχει ως αποτέλεσμα τον πολύ γρήγορο εντοπισμό σφαλμάτων και την γρήγορη ανατροφοδότηση (feedback) στους developers, επιτρέποντας τους να γνωρίζουν ανά πάσα στιγμή τα αποτελέσματα των tests που εκτελέστηκαν αυτόματα. Μερικά από τα δημοφιλέστερα CI/CD tools τα οποία μπορούν να ενσωματωθούν με automation testing frameworks που χρησιμοποιήθηκαν είναι τα εξής: Jenkins, Azure DevOps, TeamCity, GitHub Actions, GitLab CI/CD, Bamboo, CircleCI, TravisCI, κλπ.

5.3 Επίλογος

Στο τελευταίο κεφάλαιο, αναφέρονται τα συμπεράσματα, μερικές διαπιστώσεις και τα πεδία αξιοποίησης της παρούσας πτυχιακής εργασίας. Τέλος, αναφέρονται οι πιθανές επεκτάσεις της πτυχιακής εργασίας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Certified Tester Foundation Level (CTFL) Syllabus version 2018 v3.1.1, ISTQB. [Online]. Available: https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB-CTFL_Syllabus_2018_v3.1.1.pdf

[2] Shift-left and it's benefits, Isit. [Online]. Available: https://www.isit.fr/fr/article/shift-left-testing-et-ses-avantages.php

[3] Canary vs A/B strategy, StackOverflow. [Online]. Available: https://stackoverflow.com/questions/62092338/canary-vs-a-b-release-strategy

[4] Πυραμίδα του Maslow: Οι ανθρώπινες ανάγκες και τα κίνητρα για τις επιχειρήσεις, Greek business. [Online]. Available: https://greekbusiness.wixsite.com/news/post/pyramida-tou-maslow-stis-epixeiriseis

[5] Why is Software Testing Important for Your Business?, Medium. [Online]. Available: https://medium.com/@focaloidtechnologies/why-is-software-testing-important-for-your-business-d67cbf5a9f83

[6] 10 historical software bugs with extreme consequences, Pingdom . [Online]. Available: https://www.pingdom.com/blog/10-historical-software-bugs-with-extreme-consequences/

[7] Quality definition and meaning by Himanshu Rajak, Hmhub . [Online]. Available: https://hmhub.in/quality-definition-and-meaning/

[8] Everything you need to know about SDLC, Bleuwire . [Online]. Available: https://bleuwire.com/everything-need-to-know-about-sdlc/

[9] W. Hetzel. The Complete Guide to Software Testing, 2nd Edition. QED Inf. Sc., Inc., 1988.

[10] Antonia Bertolino – Software Testing Research: Achievements, Challenges, –IEEEXplore https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4221614

[11] Welcome, ISTQB. [Online]. Available: https://www.istqb.org/

[12] About us, ISTQB. [Online]. Available: https://www.istqb.org/about-us/who-we-are

[13] MSc in Software Science – About this course, Find A Masters. [Online]. Available: https://www.findamasters.com/masters-degrees/course/msc-in-software-science/?i196d8507c66267

[14] Our Certifications Programs, ISCB. [Online]. Available: https://www.softwarecertifications.org/

[15] Chroma Campus. [Online]. Available: https://www.cromacampus.com/

[16] Top 6 software testing courses, Software Testing News. [Online]. Available: https://www.softwaretestingnews.co.uk/top-5-software-testing-courses/

[17] What is Quality Assurance: A must or a waste of time?, Medium. [Online]. Available: https://mlsdev.medium.com/what-is-quality-assurance-a-must-or-a-waste-of-time-caba6f21ffac

[18] Quality Assurance vs Quality Control: Difference between them with definition and comparison chart, Key Differences. [Online]. Available: https://www.youtube.com/watch?v=zSyICkGZ6iM

[19] ISTQB Foundation Level 2018 – Introduction to ISTQB Certification. [Online]. Available: https://www.youtube.com/watch?v=bmdmZXMYrmo&list=PLj5VKaW115t1o1hk5ZbNWFr4sW5mBpvmv

[20] Differences between Verification and Validation, Geeks for geeks. [Online]. Available: https://www.geeksforgeeks.org/differences-between-verification-and-validation/

[21] What is Software Testing Life Cycle (STLC)? Step by step guide, QAWREK . [Online]. Available: https://qawerk.com/blog/software-testing-life-cycle-step-step-guide/

[22] Glossary, ISTQB. [Online]. Available: https://glossary.istqb.org/en_US/search?term=

[23] Manual vs Automation testing, BrowserStack. [Online]. Available: https://www.browserstack.com/guide/manual-vs-automated-testing-differences

[24] Manual Testing vs Automation Testing: Which One Should You Choose, Testsigma. [Online]. Available: https://testsigma.com/blog/manual-testing-vs-automation-testing-which-one-should-you-choose/

[25] What is the difference between Severity and Priority? With Examples, TryQa. [Online]. Available: http://tryqa.com/what-is-the-difference-between-severity-and-priority/

[26] What is the difference between Severity and Priority? With Examples, guru99. [Online]. Available: http://tryqa.com/what-is-the-difference-between-severity-and-priority/

[27] Functional vs Non Functional Requirements, Geeks for geeks. [Online]. Available: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/

[28] Certified Tester Foundation Level (CTFL) Syllabus v4, ISTQB. [Online]. Available: https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB_CTFL_Syllabus-v4.0.pdf

[29] Forgotten Layer of the Test Automation Pyramid, Mountain Goat Software. [Online]. Available: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

[30] The Testing Pyramid: How to Structure Your Test Suite, Semaphore. [Online]. Available: https://semaphoreci.com/blog/testing-pyramid

[31] Just Say No to More End-to-End Tests, Google Testing Blog. [Online]. Available: https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

[32] Test for value, not vanity. The testing “pyramid” is dead, Edward Burton. [Online]. Available: https://ed-burton.medium.com/test-for-value-not-vanity-the-testing-pyramid-is-dead-5abd75c0498a

[33] What is End To End Testing, BrowserStack.[Online]. Available: https://www.browserstack.com/guide/end-to-end-testing#:~:text=End%2Dto%2Dend%20testing%20is%20a%20type%20of%20testing%20that,and%20meets%20the%20user%20requirements.

[34] What is Stress Testing in Software Testing? , TutorialsPoint. [Online]. Available: https://www.tutorialspoint.com/what-is-stress-testing-in-software-testing

[35] 5 Types of Performance testing & Top Tools, The QA Lead. [Online]. Available: https://theqalead.com/test-management/everything-you-need-to-know-about-performance-testing/

[36] Software Testing | Security Testing, Geeks for geeks. [Online]. Available: https://www.geeksforgeeks.org/software-testing-security-testing/

[37] Input Validation, Science Direct. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/input-validation#:~:text=Input%20validation%20is%20the%20process,standard%20defined%20within%20the%20application.

[38] Application Environments, IBM. [Online]. Available: https://www.ibm.com/docs/en/zos/2.4.0?topic=tabs-application-environments

[39] The Question of Multiple Databases & Pre-Production Complexity, DBmaestro [Online]. Available: https://www.dbmaestro.com/blog/database-devops/multiple-databases

[40] Selenium automates browsers. That's it!, Selenium. [Online]. Available: https://www.selenium.dev/

[41] Java Downloads, Oracle. [Online]. Available: https://www.oracle.com/java/technologies/downloads/

[42] Test. Automate. Accelerate, Cypress.io. [Online]. Available: https://www.cypress.io/

[42] node.js, Nodejs. [Online]. Available: https://nodejs.org/en

[43] Playwright enables reliable end-to-end testing for modern web apps, Playwright.dev. [Online]. Available: https://playwright.dev/

[44] Is Playwright better than Cypress? Playwright vs Cypress, Medium. [Online]. Available: https://medium.com/geekculture/is-playwright-better-than-cypress-playwright-vs-cypress-151bd65a224f

[45] Playwright vs Selenium vs Cypress: A detailed comparison, Lamdatest . [Online]. Available: https://www.lambdatest.com/blog/playwright-vs-selenium-vs-cypress/

ΠΑΡΑΡΤΗΜΑ Α:

Master Test Plan για τον ιστότοπο του «https://uniportal.ihu.gr»

Εισαγωγή

Το εξής έγγραφο περιγράφει την συνολική στρατηγική ελέγχου για τον ιστότοπο του «https://uniportal.ihu.gr». Παρέχει μια επισκόπηση της συνολικής προσέγγισης του ελέγχου και του πεδίου των στόχων και του χρονοδιαγράμματος των δοκιμών.

Σκοπός

Ο σκοπός αυτού του Master Test Plan είναι να καθορίσει τις δραστηριότητες δοκιμής, τους πόρους και τις αρμοδιότητες που απαιτούνται για τη διασφάλιση της ποιότητας και της αξιοπιστίας του ιστότοπου «https://uniportal.ihu.gr»

Πεδίο εφαρμογής

Το πεδίο εφαρμογής της προσπάθειας των δοκιμών καλύπτει το σημαντικότερο μέρος του ιστότοπου «https://uniportal.ihu.gr», συμπεριλαμβάνοντας τον έλεγχο της front-end διεπαφής χρήστης αλλά και των back-end λειτουργιών του.

Στόχοι

Οι κύριοι στόχοι αυτής της προσπάθειας είναι:

- Επικύρωση της λειτουργικότητας και της διαδικτυακής εφαρμογής σύμφωνα με τις απαιτήσεις της.
- Τυχόν εντοπισμό και αναφορά ελαττωμάτων και ζητημάτων προς επίλυση.
- Διασφάλιση μιας συνεπούς και θετικής εμπειρίας χρήστη σε διαφορετικούς browsers

Υποθέσεις

- Οι δοκιμές θα πραγματοποιηθούν στο Production environment καθώς δεν υπάρχει διαθέσιμο κάποιο Test environment.
- Τα Test data είναι διαθέσιμα.
- Η ανάπτυξη έχει ολοκληρωθεί πριν ξεκινήσει ο έλεγχος.

Test Approach

Η προσέγγιση των δοκιμών για τον ιστότοπο του «https://uniportal.ihu.gr» θα πραγματοποιηθεί κατά κόρο μέσω Automation testing.

Automated Testing

Το Automation testing θα χρησιμοποιηθεί για την διεξαγωγή της προσπάθειας του ελέγχου ενισχύοντας έτσι το Regression testing.

Πρόγραμμα δοκιμών

Το πρόγραμμα δοκιμών ελέγχου για τον ιστότοπο «https://uniportal.ihu.gr» είναι το εξής:

Production	Ημερομηνία έναρξης	Ημερομηνία περάτωσης
Test Planning	14/05/2023	17/05/2023
Test Case Design	18/05/2023	22/05/2023
Test Implementation	23/05/2023	16/06/2023
Test Execution	17/06/2023	21/06/2023
Defect Reporting	22/06/2023	27/06/2023
Test Completion	28/06/2023	30/06/2023

Test Environment

Το περιβάλλον του ελέγχου αποτελείται από

- Λειτουργικό σύστημα: Windows 10 (64 bit)
- Browsers: Google Chrome [116.0.5845.96], Mozilla Firefox [99.0.1]
- IDEs: Eclipse, Visual Studio Code.
- Web automation Tools και γλώσσες προγραμματισμού: Selenium (Java), Playwright (JavaScript), Cypress (JavaScript).
- Test automation tools: TestNG, Mocha

Test Deliverables

Τα ακόλουθα Test deliverables θα παραχθούν κατά την διάρκεια της διεξαγωγής του ελέγχου:

- 1 Test Plan για το test type του functional testing. Τα test types των End to End και UI tests μπορούν να θεωρηθούν παρόμοια.
- 10 Test cases για το functional testing.

Ρόλοι και ευθύνες

- Κωνσταντίνος Χουλιάρας: Διεξαγωγή της ολικής προσπάθειας του ελέγχου.

Ρίσκα και μετριασμός ρίσκων

- Άρνηση εξυπηρέτησης του web server: Πιθανότητα άρνησης εξυπηρέτησης του web server λόγω της διεξαγωγής του End to End testing.
    - Μετρίαση: Προκειμένου να μετριαστεί το παραπάνω ρίσκο, η έγχυση των Requests θα περιοριστεί στο ελάχιστο.

Sign-off

Αυτό το Master Test plan έχει συμφωνηθεί μεταξύ των παρακάτω stakeholders

Ον/μο	Τίτλος	Ημερομηνία	Υπογραφή
Παύλος Παυλόπουλος	QA Lead	25/08/2023	
Δημήτρης Δημητριάδης	QA Manager	25/08/2023	
Γιάννης Γιαννόπουλος	Product Owner	26/06/2023	

Τα test types των End to End και UI testing μπορούν να θεωρηθούν ως ένα Test type, εφόσον αυτοί οι δύο τύποι ελέγχου μπορούν να συνδυαστούν. Για το test type του functional testing δημιουργείται το παρακάτω Test Plan.

ΠΑΡΑΡΤΗΜΑ Β:

Test plan για End to End και UI test types στον ιστότοπο «https://uniportal.ihu.gr»

Όνομα του έργου λογισμικού: Διαδικτυακή Πύλη Φοιτητολογίου

Αριθμός έκδοσης: 2.0.3

Ημερομηνία δημιουργίας: 17/05/2023

Δημιουργήθηκε από: Κωνσταντίνο Χουλιάρα

Ημερομηνία τελευταίας ενημέρωσης: 23/06/2023

Προθεσμία διεξαγωγής του ελέγχου (καταλληκτική): 30/09/2023

Εισαγωγή:

- Το συγκεκριμένο έγγραφο δημιουργήθηκε προκειμένου να επικυρωθούν οι βασικές λειτουργίες του ιστότοπου «https://uniportal.ihu.gr»

Στόχοι:

- Οι στόχοι του είναι να διασφαλιστούν οι βασικές End to End λειτουργίες και η διεπαφή χρήστη του ιστότοπου «https://uniprotal.ihu.gr»

Πεδίο των δοκιμών:

- Οι δοκιμές θα εκτελεστούν στις ιστοσελίδες του ιστότοπου «https://uniportal.ihu.gr»

Test Techniques:

- Για την διεξαγωγή του ελέγχου θα χρησιμοποιηθούν Black box testing και Experience based τεχνικές

Test Environment:

- Ο έλεγχος θα εκτελεστεί στο Production environment εφόσον δεν υπάρχει διαθέσιμο Test environment

Test Cases: Παρακάτω στην εικόνα παρουσιάζονται ονομαστικά τα test cases και οι σχετικές πληροφορίες τους για τα End to End και UI test types

EndToEndUITests ⓘ

EndToEndUITest folder

ID	TITLE	PRIORITY	OWNER	TAGS
TC-24	TC13_LogInWithCorrectUsernameWrongPasswordKO	Critical	Konstantinos Chouliaras	Log In
TC-23	TC12_LogInWithWrongUsernameCorrectPasswordKO	Critical	Konstantinos Chouliaras	Log In
TC-22	TC11_LogInWithoutEnteringPasswordKO	Critical	Konstantinos Chouliaras	Log In
TC-21	TC10_LogInWithoutEnteringUsernameKO	Critical	Konstantinos Chouliaras	Log In
TC-20	TC09_HamburgerMenuItemsWhenCollapsedOK	High	Konstantinos Chouliaras	HamburgerMenuItems
TC-19	TC08_Exams_ExamsCalendarOK	Medium	Konstantinos Chouliaras	Exams +1 tags
TC-17	TC07_Grades_AllGradesDropDownMenuOK	High	Konstantinos Chouliaras	Grades +1 tags
TC-16	TC06_InternshipOK	Medium	Konstantinos Chouliaras	Internship
TC-15	TC05_PersonalDataOK	High	Konstantinos Chouliaras	Personal Data
TC-14	TC04_ManualGuideOK	Low	Konstantinos Chouliaras	ManualGuide
TC-13	TC03_ChangeLanguageOK	Medium	Konstantinos Chouliaras	ChangeLanguage
TC-12	TC02_LogOutOK	Critical	Konstantinos Chouliaras	Log out
TC-11	TC01_LogInCorrectUsernameCorrectPasswordOK	Critical	Konstantinos Chouliaras	Log In +1 tags

Test Execution:

- Οι δοκιμές θα εκτελεστούν χρησιμοποιώντας τα εξής Web automation frameworks και τις ακόλουθες γλώσσες προγραμματισμού: Selenium (Java), Playwright (JavaScript) και Cypress (JavaScript).
- Οι δοκιμές θα εκτελεστούν με την βοήθεια των εξής testing frameworks: TestNG για το Selenium, Mocha για το Cypress. Με την χρήση του Playwright δεν χρειάζεται να χρησιμοποιηθεί κάποιο επιπλέον testing framework.
- Επίσης θα χρησιμοποιηθούν τα εξής IDEs: Eclipse (για Selenium) και Visual Studio Code (για Playwright και Cypress).

Defect Management:

- Για την αναφορά ατελειών θα χρησιμοποιηθεί το bug tracking tool «YouTrack».

Test Reporting:

- Το Test progress report θα αποστέλλεται το μεσημέρι κάθε Παρασκευής στο Scrum team και στους Stakeholders.

Risks and Mitigation:

- Άρνηση εξυπηρέτησης του web server: Πιθανότητα άρνησης εξυπηρέτησης του web server λόγω της διεξαγωγής του End to End testing.
    - Μετρίαση: Προκειμένου να μετριαστεί το παραπάνω ρίσκο, η έγχυση των Requests θα περιοριστεί στο ελάχιστο.

Entry Criteria

- Τα Test data είναι διαθέσιμα (username και password)
- Η ανάπτυξη έχει ολοκληρωθεί πριν ξεκινήσει ο έλεγχος.

Exit Criteria

- Test metrics: >=80% test pass ratio (τουλάχιστον το 80% των tests έχουν τρέξει επιτυχώς χωρίς να εντοπίσουν bugs)
- Δεν έχουν εντοπιστεί Bugs με «Critical» severity

Sign-Off :

- Τα συγκεκριμένο Test plan θεωρείται ολοκληρωμένο έπειτα από την διαδικασία του «Test Plan review» από τους stakeholders συμπεριλαμβανόμενου και του Test manager.

ΠΑΡΑΡΤΗΜΑ Γ: Selenium, Cypress και Playwright Project End to End και UI tests - https://github.com/CostasChou/