ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Διπλωματική Εργασία

Τεχνικές εξασφάλισης ποιότητας σε συστήματα ειδικού σκοπού (Bluetooth Low Energy) - Αυτοματοποίηση ελέγχου ποιότητας λογισμικού

ΜΟΣΧΟΠΟΥΛΟΥ ΕΙΡΗΝΗ
Α.Μ. 235588

Επιβλέπων
Καθηγητής Μιχάλης Ξένος

Μέλος Επιτροπής Αξιολόγησης
Καθηγητής Ιωάννης Γαροφαλάκης

Πάτρα, 2018

Ευχαριστίες

Ιδιαίτερα ευχαριστώ στον επιβλέποντα καθηγητή, κύριο Μιχάλη Ξένο για την πολύτιμη επικουρεία του και το έμπρακτο ενδιαφέρον σε κάθε στάδιο εκπόνησης της παρούσας εργασίας. Μεγάλο ευχαριστώ επίσης στην εταιρεία Dialog Semiconductor Hellas AE και ιδιαίτερα στους κυρίους Ιωάννη Παπανίκο, Θεόδωρο Βασιλείου και Κωνσταντίνο Θεοδωρόπουλο, όπου χωρίς την βοήθεια αυτών πιθανότατα να μην είχε υπάρξει αυτή η εργασία που κρατάτε στα χέρια σας. Κλείνοντας θα ήθελα να ευχαριστήσω την οικογένεια μου, τους γονείς μου Βασίλη και Αγγελική, τα αδέρφια μου Βαγγέλη και Γεράσιμο, την δίδυμη αδερφή μου Μαρία για την αμέριστη ψυχολογική, υλική και κάθε είδους στήριξη σε όλη αυτή την κοπιώδη προσπάθεια που κατεβλήθη τα τελευταία έξι χρόνια της ζωής μου. Χωρίς αυτούς δεν θα είχα φτάσει ως εδώ.

Στους γονείς μου, Βασίλη και Αγγελική.
Στα αδέλφια μου, Βαγγέλη, Γεράσιμο, Μαρία.

Περιεχόμενα

Λίστα Εικόνων

Λίστα Πινάκων

Απόδοση Όρων

Acceptance level testing	Έλεγχος επιπέδου αποδοχής
Accessibility	Προσβασιμότητα
Advertising channels	Κανάλια αποστολής
Agile model	Επιθετικό μοντέλο ελέγχου
Automated testing	Αυτοματοποιημένος έλεγχος
Automated regression tests	Αυτοματοποιημένες δοκιμές παλινδρόμησης
Basic tests	Βασικές δοκιμές
Beta products	Προϊόντα υπό δοκιμή
Beta testing	Δοκιμαστικός έλεγχος
Beta users	Δοκιμαστικοί χρήστες
Black box testing	Έλεγχος μαύρου κουτιού
BLE controller software	Ελεγκτής λογισμικού BLE
Bluetooth device address	Διεύθυνση συσκευής Bluetooth
Bluetooth special interest group (SIG)	Ειδική ομάδα ενδιαφέροντος Bluetooth
Branch	Διακλάδωση
Broadcast data	Δεδομένα μετάδοσης
Business acceptance testing	Έλεγχος αποδοχής από μια άλλη εταιρεία
Commit	Υποβολή
Compile	Μεταγλώττιση
Compatibility	Συμβατότητα
Concurrency	Συγχρονισμός
Connection event	Συμβάν σύνδεσης
Console output	Οθόνη εξόδου
Continuous integration	Συνεχής ολοκλήρωση
Criteria	Κριτήρια
Dashboard	Πίνακας εργαλειών
Data-driven scripts	Προγράμματα αυτοματοποιημένου ελέγχου που βασίζονται σε δεδομένα
Debugging	Αποσφαλμάτωση
Defect	Ελάττωμα
Defect report	Αναφορά ελαττωμάτων
Designer	Σχεδιαστής
Documentation tests	Δομικές τεκμηρίωσης
Driver	Πρόγραμμα οδήγησης
Dynamic analysis	Δυναμική ανάλυση
Efficiency	Αποτελεσματικότητα
Engineer	Μηχανικός
Error	Σφάλμα
Failure	Αποτυχία
Fault	Βλάβη
Feedback	Ανάδραση
Frequency hopping spread spectum	Εύρος διασποράς συχνότητας
Functionality	Λειτουργικότητα

Functionality tests	Δοκιμές λειτουργικότητας
Grey box testing	Έλεγχος γκρίζου κουτιού
Incremental development model	Βαθμιαία αυξανόμενο μοντέλο ανάπτυξης
Installability	Δυνατότητα εγκατάστασης
Integration testing	Έλεγχος ολοκλήρωσης
Interoperability tests	Δοκιμές διαλειτουργικότητας
Iterative development model	Επαναληπτικό μοντέλο ανάπτυξης
Kernel scheduler	Προγραμματιστής πυρήνα
Keyword-driven scripts	Προγράμματα αυτοματοποιημένου ελέγχου που βασίζονται σε λέξεις-κλειδιά
Linear scripting	Γραμμικά οργανωμένα προγράμματα αυτοματοποιημένου ελέγχου
Link layer	Επίπεδο σύνδεσης
Localizability	Χωρική και χρονική Προσαρμοστικότητα
Low power mode	Λειτουργία χαμηλής κατανάλωσης
Maintainability	Ευκολία Συντήρησης
Main program	Κύριο πρόγραμμα
Manager	Υπεύθυνος Διαχείρισης
Manual testing	Χειρωνακτικός έλεγχος
Manufacturing view	Η οπτική της βιομηχανίας παραγωγής λογισμικού
Master	Κεντρική συσκευή
Metadata	Μεταδεδομένα
Metric	Μετρική
Operation system	Λειτουργικό σύστημα
Performance	Απόδοση
Performance tests	Δοκιμές απόδοσης
Peripheral driver	Οδηγός για περιφερειακές συσκευές
Portability	Μεταφερσιμότητα
Product view	Η οπτική για το ίδιο το προϊόν
Programmer	Προγραμματιστής
Prototyping	Μέθοδος ανάπτυξης με τη δημιουργία πρωτοτύπων
Quality	Ποιότητα
Quality factors	Παράγοντες ποιότητας
Radio module	Ραδιοσυσκευή
Rapid application development	Ταχεία ανάπτυξη εφαρμογών
Regression tests	Δοκιμές παλινδρόμησης
Regularity tests	Δοκιμές συμμόρφωσης με τις νομοθετικές και κανονιστικές μεταρρυθμίσεις μιας χώρας
Reliability	Αξιοπιστία
Reliability tests	Δοκιμές αξιοπιστίας
Report	Αναφορά
Robustness tests	Δοκιμές ανθεκτικότητας
Run	Εκτέλεση (προγράμματος)
Reusability	Επαναχρησιμοποιησιμότητα
Scalability	Ευελιξία
Scalability tests	Δοκιμές κλιμάκωσης
Security	Ασφάλεια
Server disk	Δίδκος διακομιστή
Sequential development model	Διαδοχικ;ο μοντέλο ανάπτυξης
Shared scripts	Διαμοιραζόμενα προγράμματα αυτοματοποιημένου ελέγχου
Slave	Περιφερειακή συσκευή

Software	Λογισμικό
Software bugs	Αδυναμίες λογισμικού
Software development life cycle	Κύκλος ζωής ανάπτυξης λογισμικού
Software engineer	Μηχανικός ανάπτυξης λογισμικού
Software errors	Λάθη λογισμικού
Software Project	Έργο λογισμικού
Software Project Manager	Υπεύθυνος διαχείρισης εργου λογισμικού
Software testing	Έλεγχος ποιότητας λογισμικού
Software quality	Ποιότητα λογισμικού
Software quality assurance	Εξασφάλιση ποιότητας λογισμικού
Spiral model	Μοντέλο ελέγχου σπιράλ
Sprint	Επαναληπτική μέθοδος εργασίας
Stability tests	Δοκιμές σταθερότητας
Staging area	Ενδιάμεσο στάδιο
Static analysis	Στατική ανάλυση
Stress tests	Δοκιμές υπό ακραίες συνθήκες
Structured scripting	Δομημένα προγράμματα αυτοματοποιημένου ελέγχου
System testing	Έλεγχος συστήματος
Testability	Ελεγξιμότητα
Test case	Στατική δοκιμή
Test Engineer	Μηχανικός ελέγχου
Test logs	Αρχεία καταγραφής ελέγχου
Test scripts	Κώδικας ελέγχου
Test suite	Σουΐτα δοκιμών
Timers	Χρονοδιακόπτες
Transcendental view	Εμπειρική άποψη
Trigger	Ώθηση
Unit testing	Έλεγχος μονάδων
User	Χρήστης
User view	Η οπτική του χρήστη
User acceptance testing	Έλεγχος επιπέδου αποδοχής
Usability	Ευχρηστία
User manual	Εγχειρίδιο χρήσης
Value based view	Θεώρηση βάσει της αξίας
V model	Μοντέλο ελέγχου σχήματος V
Waterfall model	Μοντέλο ελέγχου καταρράκτη
White box testing	Έλεγχος λευκού κουτιού
Workflow	Ροή εργασίας
Working directory	Κατάλογος εργασίας
World wide web	Παγκόσμιος ιστός

1

ΕΙΣΑΓΩΓΗ

1.1 ΣΗΜΑΣΙΑ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Η ραγδαία εξέλιξη των συστημάτων λογισμικού τις τελευταίες δεκαετίες έχει πυροδοτήσει πλήθος συζητήσεων και ερευνών σχετικά με την ποιότητά τους. Από την έρευνα του McCall, έως και σήμερα έχουν γίνει πολλές αλλαγές στον τρόπο που μελετάμε και μετράμε την ποιότητα λογισμικού, ωστόσο υπάρχουν κάποιες βασικές αρχές που θα πρέπει να ακολουθεί κάθε κύκλος ζωής ανάπτυξης λογισμικού εάν θέλει να είναι επιτυχημένος. Μέχρι και πριν δυο δεκαετίες, οι μηχανικοί ελέγχου είχαν επικεντρωθεί στους χειρωνακτικούς τρόπους ελέγχου λογισμικού, οι οποίοι όμως είναι χρονοβόροι, ακριβοί και πολλές φορές μη αποτελεσματικοί. Οι αυξανόμενες απαιτήσεις της σύγχρονης τεχνολογίας και τα αυστηρά χρονοδιαγράμματα στην έκδοση των λογισμικών έχουν αναδείξει την αυτοματοποίηση του ελέγχου ως αναπόσπαστο κομμάτι της διαδικασίας ανάπτυξης λογισμικού. Μάλιστα, οι αυτοματοποιημένοι έλεγχοι εξετάζουν και αξιολογούν κάθε πτυχή του συστήματος λογισμικού, σε κάθε βήμα του κύκλου ζωής ανάπτυξης του λογισμικού.

1.2 ΣΤΟΧΟΙ ΕΡΓΑΣΙΑΣ

Στόχος της παρούσας διπλωματικής εργασίας είναι η ανάλυση και η ανάδειξη της τέχνης της ποιότητας λογισμικού από την στιγμή που πρωτοεμφανίστηκαν οι υπολογιστές μέχρι σήμερα. Στα επόμενα κεφάλαια θα βρείτε μια εκτενή ανάλυση των ποιοτικών χαρακτηριστικών λογισμικού, των μετρικών ποιότητας λογισμικού, των επιπέδων και των μεθόδων ελέγχων. Επιπλέον, περιγράφονται λεπτομερώς τα μοντέλα ανάπτυξης λογισμικού και ο τρόπος που ο έλεγχος εντάσσεται σε αυτά. Η παρούσα διπλωματική εργασία, επίσης, αναφέρει τις βασικές αρχές του αυτοματοποιημένου ελέγχου λογισμικού και παρουσιάζει την προσωπική ενασχόληση με την αυτοματοποίηση ελέγχου χρησιμοποιώντας συγκεκριμένα εργαλεία ανάπτυξης κώδικα και εργαλεία αυτοματοποιημένου ελέγχου, τα οποία αναφέρονται παρακάτω.

1.3 ΜΕΘΟΔΟΛΟΓΙΑ ΠΡΟΣΕΓΓΙΣΗΣ

Τα επόμενα κεφάλαια έχουν αναπτυχθεί με αφορμή την προσωπική εργασιακή και θεωρητική εμπειρία που έχει αποκτηθεί τον τελευταίο ενάμιση χρόνο, αφενός από τις πολύτιμες γνώσεις που αντλήθηκαν κατά τη διάρκεια της πρακτικής άσκησης στην εταιρεία Dialog Semiconductor το χρονικό διάστημα 01/04/2017 έως 1/07/2017, έχοντας τον τίτλο Quality Assurance Junior Test Engineer, αφετέρου δε από προσωπική ενασχόληση με τα θέματα ποιότητας λογισμικού παράλληλα και με την υλοποίηση εκτενούς έρευνας πάνω στις καλύτερες σύγχρονες μεθόδους και τακτικές χειρωνακτικού αλλά και αυτοματοποιημένου ελέγχου λογισμικού.

1.4 ΣΥΝΕΙΣΦΟΡΑ

Σκοπός του παρόντος κείμενου είναι η σχεδίαση και ανάπτυξη προγραμμάτων αυτοματοποιημένου ελέγχου λογισμικού σε ρεαλιστικές συνθήκες υλοποίησης. Οι έλεγχοι σχεδιάστηκαν και αναπτύχθηκαν με κύριο στόχο την γρήγορη, απλή και άμεση εξαγωγή αποτελέσμάτων. Μετά από ενδελεχή έρευνα και ανάλυση πάνω στους χειρωνακτικούς τρόπους ελέγχων, σχεδιάστηκε ένα πλάνο ελέγχου που αυτοματοποιεί κάθε ξεχωριστό βήμα αυτών. Οι κώδικες αυτοματοποιημένων ελέγχων ενώνουν όλα τα παραπάνω βήματα και είναι σε θέση να τα υλοποιούν μέσα σε λίγα δευτερόλεπτα πατώντας απλώς ένα κουμπί. Τα προγράμματα αυτοματοποιημένου ελέγχου έχουν σχεδιαστεί με τέτοιο τρόπο έτσι ώστε να εμφανίζουν τα αντίστοιχα αποτελέσματα κάθε ξεχωριστού είδους ελέγχου σχεδόν αμέσως στην οθόνη του μηχανικού που τα υλοποιεί. Τα αποτελέσματα περιέχουν επίσης τις απαραίτητες πληροφορίες και παραμέτρους των ελέγχων καθώς και την ακριβή ημερομηνία και ώρα εκτέλεσης κάθε ξεχωριστής δοκιμής. Τέλος, στον κύριο κώδικα κάθε ξεχωριστού ελέγχου,

υπάρχει αναλυτική περιγραφή για τον τρόπο αλλαγής των στοιχείων του προγράμματος έτσι ώστε ο έλεγχος να εφαρμόζεται εύκολα και σε άλλα συστήματα λογισμικού.

1.5 ΔΙΑΡΘΡΩΣΗ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ

Στο κεφάλαιο 2 αναλύονται οι βασικές έννοιες: λογισμικό, ποιότητα και ποιότητα λογισμικού, έτσι ώστε ο αναγνώστης να είναι σε θέση να κατανοήσει πλήρως τις διαδικασίες ελέγχου ποιότητας λογισμικού. Στο τρίτο κεφάλαιο περιγράφονται λεπτομερώς τα ποιοτικά χαρακτηριστικά του λογισμικού, οι μετρικές ποιότητας καθώς και ο χρόνος εντοπισμού των λαθών. Στο κεφάλαιο 4 γίνεται μια εκτενής και πλήρης περιγραφή των επιπέδων ελέγχου, των μεθόδων ελέγχων και των μοντέλων ανάπτυξης λογισμικού. Στο κεφάλαιο 5 αναλύεται ο κύκλος ζωής αυτοματοποιημένου ελέγχου και περιγράφονται οι βασικές αρχές τις αυτοματοποιήσης αλλά και ο τρόπος ένταξης της αυτοματοποίησης στον κύκλο ζωής ανάπτυξης του λογισμικού. Μετά από το έκτο κεφάλαιο, ο αναγνώστης θα είναι σε θέση να γνωρίζει τι είναι το Buetooth Low Energy, ποια είναι η εταιρεία Dialog Semiconductor και ποιος είναι ο σκοπός της. Στο κεφάλαιο 7, αναλύονται τα βασικά εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη των προγραμμάτων αυτοματοποιημένου ελέγχου σε προϊόντα της εν λόγω εταιρείας. Επιπλέον, στο κεφάλαιο 7 περιγράφεται λεπτομερώς το πλάνο και τα δομικά στοιχεία κάθε ξεχωριστού είδους αυτοματοποιημένου ελέγχου που αναπτύχθηκε, ο τρόπος εκτέλεσής του καθώς και τα αντίστοιχα αποτελέσματα. Τέλος, στο παράρτημα Α ο αναγνώστης μπορεί να βρει τα καλύτερα σύγχρονα εργαλεία αυτοματοποιημένου ελέγχου. Στο παράρτημα Β αναφέρονται οι σύγχρονες θέσεις εργασίας σχετικές με τον αυτοματοποιημένο έλεγχο.

2

ΛΟΓΙΣΜΙΚΟ ΚΑΙ ΠΟΙΟΤΗΤΑ ΛΟΓΙΣΜΙΚΟΥ

2.1 ΛΟΓΙΣΜΙΚΟ

ΟΡΙΣΜΟΣ ΛΟΓΙΣΜΙΚΟΥ – ΙΕΕΕ

Λογισμικό (software) είναι:
Προγράμματα υπολογιστών, διαδικασίες, σχετικά έγγραφα που περιέχουν τεκμηρίωση των παραπάνω και δεδομένα σχετικά με την λειτουργία ενός υπολογιστικού συστήματος[3].

Και τα τέσσερα παραπάνω στοιχεία είναι απαραίτητα προκειμένου να εξασφαλιστεί η ποιότητα διαδικασίας ανάπτυξης λογισμικού. Αρχικά, τα προγράμματα υπολογιστών είναι προφανώς αναγκαία διότι θέτουν τις απαραίτητες εφαρμογές ενός υπολογιστή σε λειτουργία. Οι διεργασίες είναι αναπόσπαστο κομμάτι του λογισμικού, καθώς είναι αυτές που θα καθορίσουν την σειρά εκτέλεσης των παραπάνω προγραμμάτων, όπως επίσης και τον υπεύθυνο ανάπτυξης λογισμικού. Επιπλέον, διαφορετικά είδη εγγράφων, όπως αναφορές (reports), τεχνικές οδηγίες, περιγραφή της δομής του κώδικα και άλλα, είναι εκείνα που τελικά θα κρίνουν την αποδοτική συνεργασία και τον μεθοδικό συντονισμό μεταξύ των μελών μιας ομάδας ανάπτυξης λογισμικού. Βεβαίως και το εγχειρίδιο χρήσης (user manual) προς τον τελικό χρήστη περιέχει ακριβή περιγραφή των αντίστοιχων εφαρμογών, τη σωστή μέθοδο εγκατάστασης και τις οδηγίες χρήσης τους. Το εγχειρίδιο χρήσης για τον προγραμματιστή κρίνεται εξίσου σημαντικό, εφόσον περιλαμβάνει όλες τις απαιτούμενες πληροφορίες για τον κώδικα, τη δομή του και τα αναμενόμενα τελικά αποτελέσματα της εκάστοτε εφαρμογής. Οι εν λόγω πληροφορίες χρησιμοποιούνται συνεχώς από τους επαγγελματίες για να εντοπίσουν τις πιθανές αιτίες σφαλμάτων αλλά και να επιφέρουν τις επιθυμητές αλλαγές στον κώδικα. Τέλος, η ομάδα δεδομένων περιέχει παραμέτρους, κώδικες και ονόματα λιστών που προσαρμόζονται κάθε φορά στις ανάγκες του επαγγελματία προγραμματιστή.

2.2 ΠΟΙΟΤΗΤΑ ΛΟΓΙΣΜΙΚΟΥ

Το λογισμικό είναι σαν την εντροπία. Είναι δύσκολο να το καταλάβεις, δεν έχει βάρος και υπακούει στο δεύτερο νόμο της Θερμοδυναμικής, δηλαδή, πάντα αυξάνεται.

Norman Ralph Augustine,
Ιδρυτής της IN-Q-Tel

2.2.1 ΠΟΙΟΤΗΤΑ

Οι άνθρωποι, ανέκαθεν, αναζητούσαν την ποιότητα (quality) σε κάθε προϊόν που παρήγαγαν. Η έννοια της ποιότητας, λοιπόν, είναι τόσο παλαιά όσο η ανθρώπινη προσπάθεια μαζικής παραγωγής προϊόντων και αντικειμένων μεγάλου μεγέθους. Τις τελευταίες δύο δεκαετίες έχει ξεσπάσει θα λέγαμε, μία επανάσταση της ποιότητας. Η ραγδαία ανάπτυξη του διαδικτύου πυροδότησε, την τελευταία πεντηκονταετία, αξιοσημείωτες έρευνες και αναζητήσεις με στόχο την ποιότητα. Ο παγκόσμιος ανταγωνισμός, η εξωτερική ανάθεση εργασιών, η εξωστρέφεια και οι αυξανόμενες απαιτήσεις των πελατών, έχουν αναδείξει την ποιότητα ως ένα από τα μείζονα αντικείμενα έρευνας κατα την παραγωγή οποιουδήποτε προϊόντος. Επιπλέον, θα πρέπει να προστεθεί, πως για κάθε εταιρεία, η ανάπτυξη ποιοτικών προϊόντων στα πλαίσια αυστηρότερων χρονοδιαγραμμάτων είναι κρίσιμη και είναι εκείνη που μακροπρόθεσμα την ξεχωρίζει και την αναδεικνύει.
Από την απαρχή της, η τέχνη της εξασφάλισης ποιότητας προϊόντων, επικεντρωνόταν στην ανίχνευση και τη διόρθωση ελαττωμάτων και ατελειών. Αντιθέτως, η σύγχρονη προσέγγιση, επιβάλλει την παρακολούθηση της ποιότητας του προϊόντος καθ' όλη τη διάρκεια ανάπτυξής του. Κάθε βήμα, κατά τη διάρκεια παραγωγής, θα

πρέπει να ακολουθεί κατά γράμμα τα υψηλότερα πρότυπα και να ανταποκρίνεται στις μέγιστες προσδοκίες. Έτσι λοιπόν, μία σύγχρονη, αποτελεσματική, ποιοτική διαδικασία παραγωγής θα πρέπει να χαρακτηρίζεται από [4]:

- Επικέντρωση στις απαιτήσεις του πελάτη
- Καθημερινή Προσπάθεια Βελτίωσης ποιότητας
- Ενσωμάτωση τω διαδικασιών μέτρησης και εκτίμησης κατά τη διάρκεια ανάπτυξης του προϊόντος
- Εισαγωγή της ποιότητας, σαν έννοια, ακόμη και στα χαμηλότερα επίπεδα του οργανισμού.
- Εξάλειψη των σφαλμάτων μέσω της συνεχούς βελτίωσης

2.2.2 ΠΟΙΟΤΗΤΑ ΛΟΓΙΣΜΙΚΟΥ

ΟΡΙΣΜΟΣ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ – IEEE

Ποιότητα λογισμικού είναι:

1. **Ο βαθμός κατά τον οποίο, ένα σύστημα, ένα στοιχείο ή μια διαδικασία ανταποκρίνεται σε συγκεκριμένες απαιτήσεις.**
2. **Ο βαθμός κατά τον οποίο, ένα σύστημα, ένα στοιχείο ή μια διαδικασία ανταποκρίνεται στις απαιτήσεις και τις προσδοκίες του πελάτη ή του χρήστη [3].**

Οι παραπάνω δύο προσεγγίσεις έχουν έναν κοινό προορισμό: τον βαθμό ικανοποίησης του τελικού χρήστη ή της επαγγελματικής ομάδας προγραμματιστών. Ο ορισμός της ποιότητας λογισμικού επιδέχεται πολλών διαφωνιών και συζητήσεων και εξαρτάται από την σκοπιά από την οποία τον προσεγγίζει κανείς. Η εν λόγω έννοια προσεγγίζεται διαφορετικά από διαφορετικούς ανθρώπους. Φυσικά, η διαφοροποίηση αυτή πηγάζει από τις διαφορετικές ανάγκες του καθενός.

Οι κύριες προσδοκίες που έχουν οι περισσότεροι, έτσι ώστε να χαρακτηρίσουν ποιοτικό ένα λογισμικό είναι οι παρακάτω[5]:

- Προσβασιμότητα (Accessibility)
  Ο βαθμός κατά τον οποίο, ένα λογισμικό μπορεί να χρησιμοποιηθεί άνετα από ένα τεράστιο εύρος ανθρώπων.
- Συμβατότητα (Compatibility)
  Η καταλληλόλητα του λογισμικού να εγκαθίσταται και να εκτελείται σε διαφορετικά περιβάλλοντα, όπως σε διαφορετικά λειτουργικά συστήματα, σε διαφορετικούς περιηγητές κλπ.
- Συγχρονισμός (Concurrency)
  Η ικανότητα του λογισμικού να εξυπηρετεί διαφορετικά αιτήματα από διαφορετικές πηγές, την ίδια χρονική στιγμή.
- Αποτελεσματικότητα (Efficiency)
  Η βεβαιότητα πως το λογισμικό εκτελείται σωστά και αποδίδει τα αναμενόμενα αποτελέσματα χωρίς περιττή απώλεια ενέργειας, πηγών, προσπάθειας, χρόνου και χρήματος.
- Λειτουργικότητα (Functionality)
  Η ικανότητα του λογισμικού να φέρνει εις πέρας τις επιθυμητές λειτουργίες.
- Δυνατότητα Εγκατάστασης (Installability)
  Η ευχέρεια του λογισμικού να εγκαθίσταται σε ένα ειδικό, απαιτητικό περιβάλλον.
- Χωρική και Χρονική προσαρμοστικότητα (Localizability)
  Η ευκολία προσαρμογής του λογισμικού στις διαφορετικές γλώσσες, στις διαφορετικές ζώνες ώρας.
- Ευκολία Συντήρησης (Maintainability)
  Η ευκολία με την οποία μπορεί να τροποποιηθεί το λογισμικό (προσθήκη χαρακτηριστικών, βελτίωση κώδικα, αποκατάσταση σφαλμάτων)
- Απόδοση (Performance)
  Η ταχύτητα με την οποία το λογισμικό εκτελείται, υπό κανονικές αλλά και υπό ακραίες συνθήκες εφαρμογής.
- Μεταφερσιμότητα (Portability)
  Η ικανότητα του λογισμικού να μεταφέρεται εύκολα από μια διαφορετική τοποθεσία σε μια άλλη.
- Αξιοπιστία (Reliability)
  Η ιδιότητα του λογισμικού να εκτελεί μια απαιτούμενη λειτουργία υπό καθορισμένες συνθήκες, για την επιθυμητή χρονική περίοδο, χωρίς σφάλματα.
- Ευελιξία (Scalability)
  Το μέτρο της ικανότητας του λογισμικού να αυξάνει ή να μειώνει την απόδοση, σε αναλογία με τις μεταβαλλόμενες απαιτήσεις κάθε εφαρμογής.

- Ασφάλεια (Security)
  Το αίσθημα εμπιστοσύνης πως το λογισμικό είναι προστατευμένο από πιθανή μη εξουσιοδοτημένη πρόσβαση, ανεπιθύμητη εισβολή στα ιδιωτικά δεδομένα που περιέχει, κλοπή του ίδιου του κώδικα, απώλεια δεδομένων κλπ.
- Ελεγξιμότητα (Testability)
  Η ευκολία που παρέχει το λογισμικό να ελέγχεται.
- Ευχρηστία (Usability)
  Ο βαθμός κατά τον οποίο, το λογισμικό είναι δυνατό να χρησιμοποιηθεί.

Συνεπώς, όταν τίθεται η ερώτηση εάν κάποιο λογισμικό έχει ποιότητα, η απάντηση ποικίλει, ανάλογα με το ποιες και πόσες από τις παραπάνω ιδιότητες αναζητούνται σε αυτό.

2.2.3 ΈΛΕΓΧΟΣ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ

ΟΡΙΣΜΟΣ ΕΛΕΓΧΟΥ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ - IEEE

Ο έλεγχος ποιότητας λογισμικού (software testing) είναι:
Ένα σύνολο δραστηριοτήτων σχεδιασμένες έτσι ώστε να αξιολογούν την ποιότητα ενός ανεπτυγμένου ή κατασκευασμένου προϊόντος [3].

Ο έλεγχος ποιότητας λογισμικού (software testing) υπάρχει σαν έννοια από τότε που πρωτοεμφανίστηκαν οι υπολογιστές. Η προσπάθεια που απαιτείται έτσι ώστε να εκτιμηθεί η ποιότητα του λογισμικού, καταλαμβάνει ουσιαστικά μεγάλο ποσοστό -σχεδόν το μισό- του χρόνου που προορίζεται για την παραγωγή του. Επιπλέον, πολλές φορές, το περιβάλλον, οι συνθήκες και η αναγκαιότητα παραγωγής ενός λογισμικού, επιβάλλουν την άριστη εφ' όρου ζωής ποιότητά του. Δεδομένου ότι οι εκδόσεις ενός λογισμικού συνεχώς ανανεώνονται και οι εσωτερικές αλλαγές σε αυτό, διαδέχονται η μία την άλλη, επιβάλλεται να υπάρξει εκ νέου έλεγχος και εκτίμηση της ποιότητας σε κάθε εκδοχή του. Στη σύγχρονη εποχή, η αδιάκοπα αυξανόμενη παραγωγή και πώληση νέων εφαρμογών και ψηφιακών υπηρεσιών συνδυάζεται με την εξάλειψη λαθών και παραλείψεων του εκάστοτε προϊόντος, πριν αυτό διατεθεί στην αγορά. Ο έλεγχος ποιότητας λογισμικού χαρακτηρίζεται λοιπόν, ως ένας αρκούντως σημαντικός τομέας που αναπτύσσεται παράλληλα με την επιστήμη των υπολογιστών.
Το εύρος της περιοχής γνώσης που καλύπτει η ποιότητα λογισμικού είναι τεράστιο και συμπεριλαμβάνει πολλά διαφορετικά αντικείμενα και υποκατηγορίες. Ένα από τα πιο κρίσιμα ζητήματα του ελέγχου, το οποίο κρίνει και την απόδοσή του είναι η εξάλειψη των λαθών, των ατελειών, των ασαφειών του, πριν αυτό εκτεθεί δημόσια και πωληθεί. Ο έλεγχος κρίνεται απαραίτητος καθ' όλη τη διάρκεια ανάπτυξης λογισμικού. Ισοδυναμεί με την παρατήρηση της διαδικασίας παραγωγής ενός συστήματος λογισμικού προκειμένου να μειωθεί η απόσταση του τελικού και του αναμενόμενου τελικού αποτελέσματος. Μάλιστα, η άμεση ή και η ταυτόχρονη εξέταση του λογισμικού με την ανάπτυξή του, οδηγεί με ακρίβεια σε μια πραγματική εκτίμηση των αδυναμιών του[6]. Ο έλεγχος ποιότητας λογισμικού χαρακτηρίζεται ως ένα από τα πιο κρίσιμα κομμάτια της διαδικασίας σχεδίασης και παραγωγής του, για τους εξής λόγους[7]:

- Εντοπίζει και σημειώνει τις ατέλειες και τα λάθη που ίσως συνέβησαν κατά την ανάπτυξη του λογισμικού.
- Βεβαιώνει την αξιοπιστία του λογισμικού καθώς και την ευχρηστία του.
- Εξασφαλίζει την ποιότητα του τελικού προϊόντος.
- Δοκιμάζει ακόμη και στις πιο ακραίες συνθήκες την αντοχή και την απόδοση του λογισμικού.

Υπάρχουν έξι βασικές αρχές που διέπουν την επιστήμη του ελέγχου αξιοπιστίας και ποιότητας του λογισμικού. Κάθε νέος μηχανικός ελέγχου (test engineer) ποιότητας θα πρέπει να τις γνωρίζει και να τις ακολουθεί:

1. Σημαντικό βήμα της διαδικασίας ελέγχου είναι η μελέτη και η εφαρμογή, όσων περισσότερων γίνεται, πιθανών καταστάσεων στις οποίες μπορεί να οδηγηθεί το λογισμικό.
2. Η λεπτομερής και η εξαντλητική εξέταση είναι αδύνατη. Πρώτον, πολλές φορές υπάρχει περιορισμός χρόνου και δεύτερον είναι αδύνατον να εξεταστούν όλα τα πιθανά σενάρια που μπορεί να συμβούν μετά την έκδοση του λογισμικού.
3. Τα λάθη (errors) και τα σφάλματα (bugs) του λογισμικού επιβάλλεται να εντοπιστούν όσο πιο νωρίς γίνεται, έτσι ώστε να υπάρχει διαθέσιμος χρόνος για τη διόρθωσή τους. Επιπλέον, όσο πιο νωρίς εντοπίζεται μια αστοχία, τόσο πιο απλή είναι η επίλυσή της, διότι ο κώδικας βρίσκεται ακόμη σε αρχική δομή και κατάσταση.
4. Η μεγαλύτερη συγκέντρωση ατελειών εντοπίζεται σε μικρό αριθμό ενοτήτων μέσα σε ένα σύστημα.
5. Θα πρέπει να γίνει κατανοητό από τους μηχανικούς ελέγχους ποιότητας λογισμικού, πως όσες φορές επαναλαμβάνεται η ίδια τακτική ελέγχου, υπό τις ίδιες συνθήκες εφαρμογής, τόσο πιο όμοια γίνονται

τα αποτελέσματα των δοκιμών. Συνεπώς, θα πρέπει, συνεχώς να σχεδιάζονται και να καταγράφονται νέες υποθέσεις εφαρμογής του λογισμικού, για να εντοπίζονται νέα λάθη και αστοχίες.

6. Τέλος, ο έλεγχος θα πρέπει να προσαρμόζεται στις εκάστοτε ανάγκες που καλύπτει το αντίστοιχο λογισμικό. Διαφορετικές τεχνικές και μεθοδολογίες ελέγχου εφαρμόζονται για διαφορετικά είδη λογισμικών.

Ο έλεγχος είναι ίσως μια από τις πιο ακριβές διαδικασίες κατά την παραγωγή λογισμικού. Εκτός από τα άμεσα και προφανή κόστη που δημιουργεί, είναι υπεύθυνος και για άλλα, εξίσου σημαντικά, έμμεσα[8]. Το κόστος που προκύπτει από την χαμηλή ποιότητα, οφείλεται στην κακή πρόβλεψη των ατελειών από την πλευρά των ελέγχων. Όταν μάλιστα, τα λάθη εντοπίζονται από τον τελικό χρήστη, το κόστος διπλασιάζεται για τον φορέα παραγωγής του λογισμικού. Ως απόρροια των προαναφερθέντων, συνεπάγεται πως εντείνοντας τον έλεγχο σε κάθε επίπεδο στάδιο παραγωγής του λογισμικού, μειώνεται σημαντικά το κόστος υλοποίησης βραχυπρόθεσμα, αλλά και μακροπρόθεσμα. Μια καθοριστική πρόταση και λύση για την αύξηση της αποδοτικότητας των μεθόδων ελέγχου, είναι η αυτοματοποίηση κάποιου ποσοστού αυτών. Ιδανική θα ήταν η αυτοματοποίηση όλης της διαδικασίας ελέγχου. Τότε, οι ομάδες ελέγχου θα έχουν χρόνο να ασχοληθούν με σημαντικότερα ζητήματα που προκύπτουν και να χτίσουν πολυπλοκότερες συνθήκες ελέγχου. Μπορούμε να διακρίνουμε, λοιπόν, τη διεργασία ελέγχου σε δύο κατηγορίες:

i. Χειρωνακτικός έλεγχος (Manual testing)
ii. Αυτοματοποιημένος Έλεγχος (Automated testing)

2.2.4 ΕΞΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ

ΟΡΙΣΜΟΣ ΔΙΑΣΦΑΛΙΣΗΣ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ – IEEE

Εξασφάλιση Ποιότητας Λογισμικού (Software quality Assurance) είναι:

1. **Ένα καλά σχεδιασμένο πρότυπο που περιλαμβάνει όλες τις ενέργειες εκείνες που κρίνονται απαραίτητες προκειμένου να εξασφαλιστεί η πεποίθηση πως ένα αντικείμενο ή ένα προϊόν συμμορφώνεται στις καθιερωμένες τεχνικές απαιτήσεις.**
2. **Ένα σύνολο δραστηριοτήτων σχεδιασμένες έτσι ώστε να αξιολογούν τις διαδικασίες σύμφωνα με τις οποίες τα προϊόντα σχεδιάζονται ή παράγονται[3].**

Ο κύριος ρόλος της διασφάλισης ποιότητας είναι να ελαχιστοποιεί το κόστος της διαδικασίας που απαιτείται έτσι ώστε να εγγυηθεί η ποιότητα, σε όλες τις φάσεις εφαρμογής της. Η εξασφάλιση ποιότητας αφορά όλα τα στάδια παραγωγής λογισμικού. Τα βήματα που ακολουθεί, μειώνουν την πιθανότητα να εμφανιστούν λάθη όταν είναι αργά, καθώς βοηθούν στον εντοπισμό και τη διόρθωση αυτών νωρίτερα. Επομένως, οι διεργασίες διασφάλισης ποιότητας, μακροπρόθεσμα, περιορίζουν σημαντικά το πλήθος των ελαττωματικών προϊόντων, συμβάλλοντας ταυτόχρονα στην ελάττωση του κόστους εκτίμησης ποιότητας[9].

2.3 Ο ΕΛΕΓΧΟΣ ΛΟΓΙΣΜΙΚΟΥ ΣΗΜΕΡΑ

Ο έλεγχος λογισμικού είναι ένα ευρύς όρος που περιλαμβάνει ένα ευμέγεθες φάσμα διαφορετικών δραστηριοτήτων, από την εξέταση ενός μικρού κομματιού κώδικα από τον προγραμματιστή, έως την αξιολόγηση ενός μεγάλου όγκου πληροφοριών από τον πελάτη.

Τα ποικίλα και διαφορετικά στάδια του ελέγχου μπορούν να διακριθούν, ανάλογα με τον σκοπό που προορίζονται να εκπληρώσουν. Ο ρόλος που τους διακρίνει μπορεί να είναι η μέτρηση συγκεκριμένων χαρακτηριστικών, η απόδοση, η χρηστικότητα, η εκτίμηση της αξιοπιστίας και τα λοιπά. Οι διαφορετικές προσεγγίσεις στον ρόλο του ελέγχου, γεννά πλήθος ορισμών του, γεγονός που δημιουργεί ποικίλες ερευνητικές προκλήσεις[6]. Ο γενικός ορισμός του ελέγχου ποιότητας λογισμικού, ο οποίος χαρακτηρίζεται κοινά αποδεκτός φαίνεται παρακάτω:

Έλεγχος ποιότητας λογισμικού είναι το σύνολο των διαδικασιών ή των μεθόδων εύρεσης λάθους (ή λαθών) σε μια εφαρμογή λογισμικού, έτσι ώστε οι λειτουργίες της να ανταποκρίνονται στις απαιτήσεις και τις προσδοκίες του χρήστη[7].

Μπορεί επίσης να οριστεί ως μια διαδικασία αξιολόγησης και επιβεβαίωσης πως ένα πρόγραμμα λογισμικού ή μια εφαρμογή ή ένα προϊόν ανταποκρίνεται στις επιχειρηματικές και τεχνικές απαιτήσεις, στις οποίες οφείλεται ο σχεδιασμός και η ανάπτυξή του. Τα τρία βασικά βήματα που ακολουθεί ο έλεγχος ποιότητας είναι ο σχεδιασμός, η προετοιμασία και η αξιολόγηση. Κατά τον σχεδιασμό, καταγράφεται το είδος των ελέγχων που θα εφαρμοστούν και το οργανόγραμμα που θα ακολουθήσουν. Στον σχεδιασμό συμπεριλαμβάνεται και η παρακολούθηση της συνολικής πορείας και προόδου των ελέγχων μέχρι και την έκδοση του τελικού προϊόντος.

Η προετοιμασία περιέχει τη διαδικασία επιλογής του είδους δοκιμών που θα εφαρμοστούν. Η επιλογή αυτή εξαρτάται από τις συνθήκες και από το είδος ελέγχου που έχει προγραμματιστεί κατά την φάση του σχεδιασμού.

Κατά τη διάρκεια της αξιολόγησης, ελέγχονται τα αποτελέσματα και αξιολογείται το λογισμικό βάσει των κριτηρίων ολοκλήρωσης των δοκιμών. Συνεπώς, σε αυτό το στάδιο γίνεται ξεκάθαρο εάν ο έλεγχος έχει τελειώσει καθώς επίσης και εάν το λογισμικό ανταποκρίθηκε όπως αναμενόταν. Τέλος, παράλληλα με τη δοκιμή του κώδικα και την καταγραφή του βαθμού ικανοποίησης των προδιαγραφών σχεδιασμού, κρίνεται απαραίτητη η συγγραφή σχετικών εγγράφων όπως οι οδηγίες λειτουργίας και εγκατάστασης και οι οδηγίες χρήσης προς τον τελικό χρήστη.

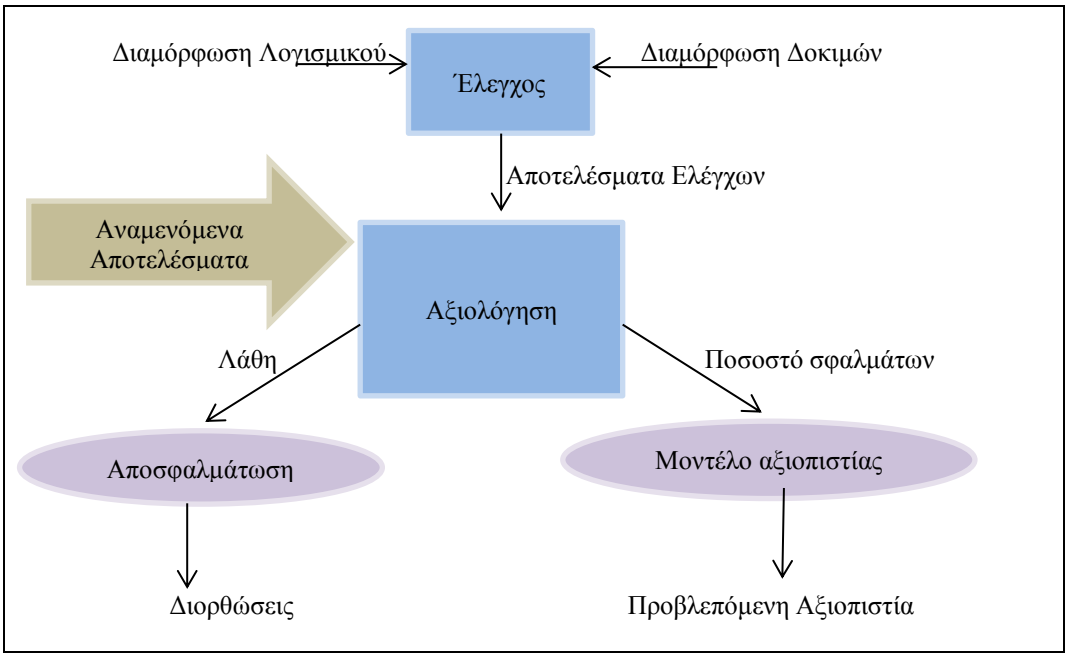
Οι Voas και Friedman έχουν ορίσει τον έλεγχο λογισμικού ως μια διαδικασία επαλήθευσης και βελτιστοποίησης της ποιότητας λογισμικού **[10].**

2.3.1 ΠΕΤΥΧΗΜΕΝΟΣ ΕΛΕΓΧΟΣ ΛΟΓΙΣΜΙΚΟΥ

Ο έλεγχος λογισμικού επικεντρώνεται στην προσπάθεια εύρεσης λαθών κατά τη διάρκεια εκτέλεσης (run) ενός προγράμματος. Αλλά, εάν ο σκοπός των μηχανικών εκτίμησης ποιότητας λογισμικού είναι να διασφαλίσουν πως ένας κώδικας δεν έχει λάθη, τότε οδηγούνται κατευθείαν σε αυτόν τον στόχο. Δηλαδή, εάν στόχος αυτών εντοπίζεται μόνο στο να βρουν κάποιο λάθος (error) στον κώδικα, τότε επιλέγουν συγκεκριμένες τακτικές ελέγχου, που οδηγούν, με μαθηματική ακρίβεια, σε μικρή πιθανότητα λάθους. Αντιθέτως, θα πρέπει να σκέφτονται οδηγούμενοι από την αντίστροφη ψυχολογία. Θα πρέπει να θεωρήσουν εξαρχής πως ο κώδικας που έχουν απέναντί τους έχει λάθη. Και μάλιστα πολλά[11]. Τότε, η εργασία τους ξεκινά σε διαφορετική βάση. Σκοπός τους πλέον είναι να σχεδιάσουν τις κατάλληλες τακτικές εντοπισμού λαθών και να τις ακολουθούν. Με αυτόν τον τρόπο, εξασφαλίζουν την έγκαιρη εύρεση του μεγαλύτερου μέρους αστοχιών και σφαλμάτων. Συνήθως, οι άνθρωποι τείνουν να ενώνουν αντικείμενα ώστε να συνεργάζονται και να αποδίδουν συνολικά όλα μαζί. Οι μηχανικοί ελέγχου ποιότητας λογισμικού θα πρέπει να ακολουθούν ακριβώς την αντίθετη τακτική.

- Να αποδομούν ένα πρόγραμμα, να το εξετάζουν υπό οποιαδήποτε συνθήκη και σε κάθε πιθανό περιβάλλον.
- Να δημιουργούν τις πιο απίθανες, τις πιο ακραίες συνθήκες που μπορεί να επέλθει η εφαρμογή ή το πρόγραμμα, έτσι ώστε να είναι σε θέση να εντοπίζουν κάθε πιθανή αστοχία.

Επιτυχημένη δοκιμή λογισμικού χαρακτηρίζεται εκείνη που έχει ελέγξει τον κώδικα σε κάθε περιβάλλον, υπό οποιαδήποτε συνθήκη, ενώ ταυτόχρονα εφαρμόζει σε αυτό τις πιο απίθανες μεταβλητές εισόδου. Ένας αποτελεσματικός έλεγχος λογισμικού είναι εκείνος που έπειτα από την υλοποίηση μιας καλώς σχεδιασμένης και δομημένης δοκιμής βρίσκει λάθη που μπορούν να διορθωθούν. Ταυτόχρονα, θα πρέπει να είναι σε θέση να διαβεβαιώσει πως δεν υπάρχουν περαιτέρω αστοχίες. Ο μόνος, μη επιτυχημένος, έλεγχος λογισμικού είναι αυτός που δεν εξέτασε προσεκτικά τον κώδικα, τις δυνατότητες και τις αδυναμίες του. Γενικώς στην πλειονότητα των περιπτώσεων, μία δοκιμή που, ως επί το πλείστον, δεν εντοπίζει καμία ατέλεια θεωρείται ανεπιτυχής καθώς λογισμικό χωρίς αστοχίες μπορεί να υπάρξει μόνο στην πλασμένη ουτοπία των μηχανικών υπολογιστών.



Εικόνα 1: Ροή πληροφοριών ελέγχου

Συμπερασματικά, μια διαδικασία ελέγχου που μπορεί να εντοπίσει τουλάχιστον ένα σφάλμα είναι επιτυχημένη. Αντιθέτως, μία αποτυχημένη εξέταση λογισμικού είναι η εκτέλεση του, υπό τις κατάλληλες συνθήκες, έτσι ώστε να φέρνει, σχεδόν στοχευμένα, πάντα τα σωστά και αναμενόμενα θετικά αποτελέσματα, χωρίς ποτέ να αναδεικνύει κάποια αστοχία. Το διάγραμμα ροής ελέγχου φαίνεται στην Εικόνα 1. Όπως φαίνεται στο σχήμα, ο έλεγχος ξεκινά ταυτόχρονα με την ανάπτυξη του λογισμικού και εφαρμόζεται καθ' όλη τη διάρκεια ζωής του. Τα αποτελέσματα των ελέγχων συγκρίνονται με τα αναμενόμενα που ορίζουν οι προδιαγραφές. Αν υπάρξουν λάθη, αυτά αναφέρονται στους προγραμματιστές προκειμένου να διορθωθούν. Το ποσοστό σφαλμάτων είναι αυτό που κρίνει την αξιοπιστία του λογισμικού.

2.3.2 ΣΤΟΧΟΣ ΤΟΥ ΕΛΕΓΧΟΥ

Ο κύριος σκοπός του ελέγχου λογισμικού είναι:

- Να επιβεβαιωθεί η ποιότητα των συστημάτων λογισμικού μέσω των συχνών και στοχευμένων δοκιμών και προσομοιώσεων του, κάτω από προσεχτικά ελεγχόμενες συνθήκες.
- Να διαβεβαιώσει τους σχεδιαστές, προγραμματιστές και τους υπεύθυνους διαχείρισης έργων λογισμικού (Software Project Managers) πως παράγουν προϊόντα με την μέγιστη δυνατή ποιότητα[12].

2.3.3 ΡΟΛΟΣ ΤΟΥ ΕΛΕΓΧΟΥ

Με τον έλεγχο λογισμικού βελτιώνεται η ποιότητα των προϊόντων, εφόσον όσες περισσότερες φορές επαναλαμβάνεται ο κύκλος: | Έλεγχος → Εύρεση λαθών → Διόρθωση |, τόσο μειώνεται η πιθανότητα να υπάρξουν αστοχίες στην τελική έκδοση του προϊόντος. Επιπλέον, αξιολογείται η απόδοση του συστήματος λογισμικού, με την ενδελεχή πραγματοποίηση πλήθους δοκιμών πριν την προώθηση του. Οι δραστηριότητες που συνιστούν τον έλεγχο χωρίζονται σε δύο μεγάλες κατηγορίες[13]:

- **Στατική ανάλυση (Static analysis)**

Βασίζεται στην ανάλυση πλήθους εγγράφων, τα οποία περιέχουν τις απαιτήσεις, τις πηγές του κώδικα, τα μοντέλα και τα είδη δοκιμών που εφαρμόζονται. Η δομή της στατικής ανάλυσης αποτελείται από:

- Απεικόνιση του κώδικα
- Επιθεώρηση των λειτουργιών του
- Ενδελεχή αλγοριθμική ανάλυση
- Απόδειξη της ορθότητας του

Δεν περιλαμβάνει εκτέλεση του ίδιου του κώδικα που βρίσκεται σε φάση ανάπτυξης. Αντιθέτως, εξετάζει τον κώδικα και αιτιολογεί τις πιθανές συμπεριφορές που ενδέχεται να παρουσιάσει κατά την εκτέλεση του. Οι προτάσεις και τα σφάλματα που προκύπτουν από την μεταγλώττιση (compile) ενός κώδικα είναι ένα αντιπροσωπευτικό παράδειγμα στατικής ανάλυσης.

- **Δυναμική Ανάλυση (Dynamic analysis)**

Η δυναμική ανάλυση ενός συστήματος λογισμικού περιέχει την πραγματική εκτέλεση του κώδικα, προκειμένου να φανερωθούν οι πιθανές αποτυχίες και παραλείψεις του. Η συμπεριφορά, η απόδοση, ο χρόνος εκτέλεσης και οι ιδιότητες του λογισμικού αξιολογούνται σε αυτήν την φάση. Οι μεταβλητές εισόδου που επιλέγονται είναι οι αναμενόμενες και οι μη. Οι τελευταίες επιλέγονται προσεκτικά και θεωρούνται ακραίες, όμως είναι απαραίτητες για να αξιολογηθεί η συμπεριφορά του συστήματος σε όλες τις διαφορετικές συνθήκες εισόδου. Η προσεκτική διαλογή του τελικού συνόλου δοκιμών είναι κρίσιμη και επηρεάζει την ασφαλή εξαγωγή συμπερασμάτων.

Και τα δύο παραπάνω είδη ανάλυσης του λογισμικού, είναι εξίσου σημαντικά και απαραίτητα για την εξασφάλιση ποιότητας λογισμικού. Επιπροσθέτως, η επανάληψη αυτών, οδηγεί σε πιο βέβαια και σίγουρα αποτελέσματα. Οι μηχανικοί ελέγχου ποιότητας και οι ερευνητές καλούνται να παραβλέψουν τις διαφορές ανάμεσα στη δυναμική και την στατική ανάλυση και να δημιουργήσουν μια νέα, υδριβική, που θα τις ενώνει και θα εκμεταλλεύεται τα προτερήματά τους [14].

2.4 ΕΙΔΗ ΕΛΕΓΧΩΝ ΛΟΓΙΣΜΙΚΟΥ

Οι εργασίες ελέγχου λογισμικού χωρίζονται σε δύο μεγάλες κατηγορίες:

- **Χειρωνακτικός Έλεγχος Λογισμικού**
- **Αυτοματοποιημένος Έλεγχος Λογισμικού**

Ο αυτοματοποιημένος έλεγχος συνήθως χρησιμοποιείται για εργασίες επαναλαμβανόμενης εκτέλεσης (*UNIT testing και Regression testing*), όπου οι διαφορετικές καταστάσεις ελέγχου (*test cases*) εφαρμόζονται κάθε φορά που υπάρχει μια αλλαγή στον κώδικα. Οι τυπικές εργασίες του αυτοματοποιημένου ελέγχου περιλαμβάνουν συγγραφή και εκτέλεση κώδικα (*scripts*) που εξετάζει αυτόματα το εκάστοτε λογισμικό. Σε αντίθεση με τον χειρωνακτικό έλεγχο, η αυτοματοποιημένη μέθοδος εξέτασης λογισμικού δεν προτείνεται να εφαρμοστεί σε συστήματα που μεταβάλλονται σπάνια. Ο χειρωνακτικός έλεγχος είναι εξ΄ορισμού κατάλληλος για τέτοιου

είδους λογισμικά. Ο αυτοματοποιημένος έλεγχος είναι μια ακριβή και δύσκολη διαδικασία για μία εταιρεία. Για τον λόγο αυτό, εφαρμόζεται μόνο στις περιπτώσεις που το λογισμικό, που ελέγχεται, είναι σχεδιασμένο να αλλάξει ή να αναβαθμιστεί στο μέλλον. Η προκείμενη μεταβολή μπορεί να οφείλεται είτε στην αλλαγή του περιβάλλοντος υλοποίησης του λογισμικού, είτε στην εκούσια βελτιστοποίηση του κώδικα, είτε στην αλλαγή της εμφάνισης της εφαρμογής και γενικά στην προσαρμογή του λογισμικού στις απαιτήσεις του τελικού χρήστη [8]. Και τα δύο είδη ελέγχου υπάρχουν και χρησιμοποιούνται για να εκπληρώσουν έναν κοινό στόχο. Την έγκαιρη, οικονομική παραγωγή του λογισμικού. Για τον λόγο αυτό, αποτελούν ένα αναπόσπαστο κομμάτι του κύκλου ζωής του λογισμικού, από την πρώτη ημέρα έως την τελευταία.

2.4.1 ΧΕΙΡΩΝΑΚΤΙΚΟΣ ΕΛΕΓΧΟΣ ΛΟΓΙΣΜΙΚΟΥ

Ο χειρωνακτικός έλεγχος λογισμικού αποτελεί τον ακρογωνιαίο λίθο της διαδικασίας ελέγχου. Όλοι οι προγραμματιστές, οι μηχανικοί ελέγχου ποιότητας και γενικά οι μηχανικοί υπολογιστών εκτελούν, σχεδόν καθημερινά χειρωνακτικές δοκιμές σε οποιοδήποτε κώδικα αναπτύσσουν, ελέγχουν, βελτιώνουν. Είναι ένα βασικό και δομικό στοιχείο οποιουδήποτε ελέγχου και οποιασδήποτε αξιολόγησης[9]. Η εγκυρότητα των αποτελεσμάτων των χειρωνακτικών δοκιμών λογισμικού εξαρτάται από πλήθος παραγόντων. Ο δια χειρός έλεγχος λογισμικού είναι ένα απαραίτητο και σημαντικό μέρος της διαδικασίας ανάπτυξης λογισμικού, ακόμη και σήμερα. Στην χειρωνακτική αναζήτηση λαθών ο μηχανικός ελέγχου παίρνει τον ρόλο του τελικού χρήστη και δοκιμάζει το προϊόν ή την εφαρμογή. Εξετάζει, με το χέρι, εάν κάθε λειτουργία του λογισμικού, εκτελείται όπως αναμένονταν. Ο μηχανικός ελέγχου εκτελεί χειρωνακτικά, διάφορα είδη ελέγχων χωρίς να χρησιμοποιεί κάποια αυτοματοποιημένη μέθοδο. Σχεδιάζει, προετοιμάζει και καταγράφει το πλάνο ελέγχου, μέσα στο οποίο περιγράφει τις λειτουργίες που θα ελέγξει και την μέθοδο που θα ακολουθήσει με συγκεκριμένα βήματα. Στο τέλος, καταγράφει τα αποτελέσματα, θετικά ή όχι. Αν οι έλεγχοι δεν φέρουν τα επιθυμητά αποτελέσματα, ο μηχανικός (tester) επιστρέφει μια λεπτομερή αναφορά στους αρχικούς μηχανικούς ανάπτυξης (programmers) του εν λόγω προγράμματος. Αυτοί με την σειρά τους, θα μελετήσουν την αναφορά και θα προσπαθήσουν να διορθώσουν τις ατέλειες που αναφέρονται σε αυτή. Ο στόχος του χειρωνακτικού ελέγχου λογισμικού είναι να διασφαλιστεί πως η τελική εφαρμογή ή πρόγραμμα ή προϊόν, δεν έχει αστοχίες και ελαττώματα. Τότε μόνο, μπορεί να προωθηθεί στον τελικό χρήστη-πελάτη. Ο δια χειρός έλεγχος λογισμικού είναι κατάλληλος για μικρότερους οργανισμούς και εταιρείες με μικρές οικονομικές απαιτήσεις[15].

Τα κυριότερα μειονεκτήματα του χειρωνακτικού ελέγχου αναφέρονται παρακάτω[16]:

- Είναι χρονοβόρος καθώς κάθε διαφορετικό είδος δοκιμών θα πρέπει να εφαρμοστεί για κάθε νέο χαρακτηριστικό του προγράμματος αλλά και για κάθε νέα έκδοση της εφαρμογής.
- Δεν είναι προσαρμοστικός. Όσο πολυπλοκότερη γίνεται μια εφαρμογή και συνεπώς απαιτεί προσεκτικότερο έλεγχο, τόσο περισσότερο χρόνο απαιτεί η κάθε δοκιμή σε κάθε λειτουργία της. Επιπλέον όσο πιο περίπλοκο γίνεται ένα πρόγραμμα τόσο μειώνεται το ποσοστό χειρωνακτικού ελέγχου σε αυτό. Η πολυπλοκότητα του δεν επιτρέπει στον μηχανικό να επέμβει και να ελέγξει κάθε κομμάτι αυτού.
- Κάθε είδος ελέγχου που εφαρμόζεται επαναλαμβανόμενα από τον ίδιο μηχανικό, θα φέρνει πάντα διαφορετικά αποτελέσματα. Η κατάληξη της ίδιας δοκιμής στο ίδιο λογισμικό, συχνά, θα είναι διαφορετική λόγω του ανθρώπινου λάθους και της απόδοσης του συστήματος τη δεδομένη στιγμή.
- Η έλλειψη κατάρτισης του προσωπικού είναι ένα κοινό πρόβλημα. Οι μηχανικοί ελέγχου θα πρέπει να έχουν εκπαιδευτεί αρκετά καλά έτσι ώστε να είναι σε θέση να εξετάζουν και να αξιολογούν τον σχεδιασμό, την εκτέλεση και τα αποτελέσματα ενός λογισμικού.
- Οι σύγχρονες πρακτικές ανάπτυξης λογισμικού είναι καλά δομημένες, αλλά εάν δεν είναι εξίσου καλά ανεπτυγμένοι οι μέθοδοι ελέγχου, είναι αρκετά δύσκολο να συμβαδίσουν μεταξύ τους.
- Οποιαδήποτε καθυστέρηση στην λήψη του λογισμικού από τους προγραμματιστές στην ομάδα ελέγχου, μπορεί να οδηγήσει σε σημαντικές απώλειες χρόνου και συνεπώς αύξηση του κόστους παραγωγής.
- Οι σύγχρονες απαιτήσεις για γρήγορη παραγωγή οικονομικών και ταυτόχρονα ποιοτικών προϊόντων καθιστούν τον χειρωνακτικό έλεγχο αδύναμο να ανταποκριθεί.

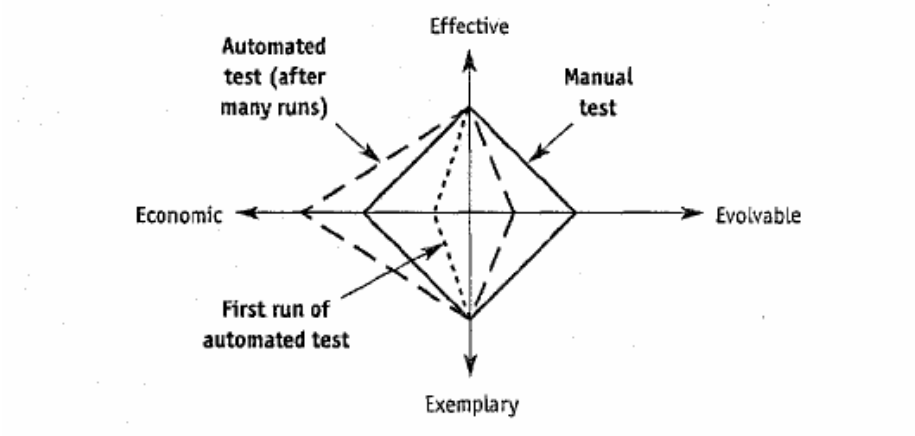
2.4.2 ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΣ ΕΛΕΓΧΟΣ ΛΟΓΙΣΜΙΚΟΥ

Ο αυτοματοποιημένος έλεγχος (Automated software testing) λογισμικού είναι ακριβότερος από τον χειρωνακτικό. Εντούτοις, τα οφέλη του για μια εταιρεία που τον χρησιμοποιεί, είναι τεράστια. Πρέπει να γίνει κατανοητό πως, παρόλο που μια διαδικασία αυτοματοποίησης όταν εφαρμόζεται για πρώτη φορά, κοστίζει χρονικά και οικονομικά, όταν αυτή επαναλαμβάνεται πολλές φορές, τότε γίνεται, μακροπρόθεσμα, οικονομικότερη σε σχέση με την αντίστοιχη, μη αυτόματη διαδικασία.

Πλεονεκτήματα του αυτοματοποιημένου ελέγχου:

- Εκτέλεση ήδη υπαρχόντων δοκιμών σε μια καινούρια έκδοση του προγράμματος.
  Σε ένα περιβάλλον, όπου συχνά πολλά προγράμματα επιδέχονται αλλαγές και τροποποιήσεις, κρίνεται απαραίτητος ο έλεγχος έπειτα από κάθε ανανέωση του περιεχομένου του λογισμικού. Υποθέτοντας πως προϋπάρχει μια διεργασία ελέγχου προηγούμενης έκδοσης του λογισμικού, είναι εύκολο για κάποιον ειδικό με χειρωνακτική εργασία λίγων λεπτών, να προσαρμόσει το υπάρχον τεστ στη νέα έκδοση του λογισμικού.
- Ένα προφανές πλεονέκτημα της αυτοματοποίησης ελέγχου είναι η εφαρμογή περισσότερων τεστ στον ίδιο χρόνο, ή και πολλές φορές σε μικρότερο χρόνο, σε σχέση με τον χειρωνακτικό έλεγχο. Συνεπώς, γίνεται πλέον εφικτή η επανάληψη κάποιων εξ΄αυτών αλλά και η εξοικονόμηση χρόνου για δημιουργία νέων μεθόδων δοκιμών.
- Υλοποίηση εξιδεικευμένων ελέγχων, που θα ήταν αδύνατο να πραγματοποιηθούν με το χέρι.
- Αυτοματοποιώντας κάποιες διαδικασίες, αυτές γίνονται πιο προσιτές σε οποιοδήποτε μη τεχνικό επιτελείο ελέγχου που επιθυμεί να τις χρησιμοποιήσει οποιαδήποτε στιγμή.
- Καλύτερη αξιοποίηση των πηγών και των ανθρώπων.
  Κάποια είδη ελέγχων που είναι αντικειμενικά βαρετά και επαναλαμβανόμενα, εκτελούνται σκοπίμως αυτοματοποιημένα. Έτσι, το προσωπικό έχει την ευκαιρία να ασχοληθεί και να αναπτύξει νέα και πιο εξιδεικευμένα είδη εξέτασης του λογισμικού. Επιπλέον, υπάρχουν συστήματα ελέγχου που απαιτούν πολύωρη εκτέλεση προκειμένου να εξάγουν κάποιο συμπέρασμα. Για αυτό το λόγο, προφανώς, πρέπει εξαρχής να προγραμματιστούν να τρέχουν αυτόματα.
- Συνέπεια και επανάληψη παρόμοιων ελέγχων.
  Οι έλεγχοι που επαναλαμβάνονται αυτόματα, παρουσιάζουν συνέπεια στο περιεχόμενό τους. Αυτό παρέχει σταθερότητα στις δοκιμές, κάτι που είναι αρκετά δύσκολο να επιτευχθεί στις περιπτώσεις όπου παρεμβαίνει ο άνθρωπος.
- Οι ίδιοι έλεγχοι μπορούν να εφαρμοστούν σε διαφορετική δομή υλικού (hardware), σε διαφορετικά λειτουργικά συστήματα (operation systems), χρησιμοποιώντας κάθε φορά διαφορετική βάση δεδομένων. Με αυτόν τον τρόπο, η εταιρεία εξασφαλίζει πως το τελικό προϊόν ή λογισμικό μπορεί να εφαρμοστεί σε διαφορετικές πλατφόρμες. Η προσαρμοστικότητα που πετυχαίνει ο αυτοματοποιημένος έλεγχος είναι σχεδόν αδύνατο να επιτευχθεί από τους χειρωνακτικούς ελέγχους.
- Σημαντική εξοικονόμηση χρόνου.
  Εφόσον ένα σημαντικό ποσοστό ελέγχων ποιότητας έχει αυτοματοποιηθεί, ο συνολικός χρόνος παραγωγής μειώνεται κατά πολύ. Η τελική διάθεση του προϊόντος στην αγορά γίνεται σε πολύ μικρότερο χρόνο.
- Αύξηση Αυτοπεποίθησης.
  Γνωρίζοντας πως πλήθος αυτοματοποιημένων ελέγχων έχουν εφαρμοστεί, με θετικά αποτελέσματα, η ομάδα ανάπτυξης λογισμικού και οι υπεύθυνοι διαχείρισης έργων, έχουν την αυτοπεποίθηση πως το τελικό προϊόν είναι τέλειο και είναι έτοιμο να πωληθεί.

Η αυτοματοποίηση, προσφέρει ενδελεχή και διεξοδικό έλεγχο, με λιγότερη κόπωση, αυξάνοντας παράλληλα την ποιότητα και την παραγωγικότητα[17]. Θα πρέπει να σημειωθεί σε αυτό το σημείο, πως η αυτοματοποίηση έχει κάποια όρια. Δεν αντικαθιστά τον παραδοσιακό χειρωνακτικό τρόπο ελέγχου. Η αυτοματοποίηση, επικεντρώνεται στην μελέτη και αξιολόγηση των τελικών αποτελεσμάτων, ωστόσο η εξέταση του λογισμικού από τον άνθρωπο, εντοπίζει περισσότερα λάθη. Τα εργαλεία που χρησιμοποιούνται για αυτοματοποίηση δεν διαθέτουν την ανθρώπινη φαντασία και διαίσθηση. Όμως, η σωστή τακτική αυτοματοποίησης μπορεί να αυξήσει σημαντικά την ποιότητα και την παραγωγικότητα των ελεγκτικών διαδικασιών.
Η αποτελεσματικότητα κάθε δοκιμής μπορεί να απεικονιστεί γραφικά στην *Εικόνα 2*, όπου συσχετίζονται τα τέσσερα βασικά χαρακτηριστικά της, χρησιμοποιώντας το διάγραμμα Kiviat: Αποδοτικότητα, οικονομία, επιδραστικότητα και δυνατότητα εξέλιξης[18]. Όσο πιο πολύ βελτιώνεται κάθε χαρακτηριστικό για κάθε έλεγχο, τόσο μεγαλύτερη είναι η περιοχή που περικλείεται στις κάθετες γραμμές σύνδεσης και τόσο καλύτερη είναι η μέθοδος ελέγχου.



Εικόνα 2: Διάγραμμα Kiviat για την αποτελεσματικότητα κάθε ελέγχου

Συγκεντρωτικά, υπάρχει πλήθος λόγων για τους οποίους προτιμάται η αυτοματοποίηση:

- Ο χειρωνακτικός έλεγχος απαιτεί πολύ χρόνο
- Οι χειροκίνητες διαδικασίες είναι επιρρεπείς σε σφάλματα
- Η αυτοματοποίηση δίνει κίνητρο στους ανθρώπους να εργαστούν με αφοσίωση
- Οι αυτοματοποιημένες δοκιμές παλινδρόμησης (automated regression tests) επιστρέφουν συχνή και έγκυρη ανατροφοδότηση
- Οι έλεγχοι συνοδεύονται από τεκμηρίωση
- Η αυτοματοποίηση μπορεί να αποτελέσει μια καλή επιστροφή επι της επένδυσης

2.4.2.1 ΠΡΟΣΔΟΚΙΕΣ ΤΗΣ ΑΥΤΟΜΑΤΟΠΟΙΗΣΗΣ

Χρησιμοποιώντας ένα πλήρες εργαλείο αυτοματοποίησης, ο αυτοματοποιημένος έλεγχος αναμένεται να εξοικονομήσει χρόνο, να τηρήσει τα πρότυπα και να εξασφαλίσει την ποιότητα των προϊόντων. Επιπλέον, όσο οι ικανότητες και η εμπειρία των μηχανικών ελέγχου λογισμικού μεγεθύνονται, τόσο τα ποσοστά αυτοματοποίησης θα αυξάνονται[19]. Ο κύκλος ζωής ανάπτυξης λογισμικού (Software development life cycle) επηρεάζει άμεσα την τεχνική αυτοματοποίησης. Συνήθως, οι προγραμματιστές επεμβαίνουν συνεχώς στον κώδικα για αλλαγές, βελτιώσεις και αναβαθμίσεις. Το εργαλείο δοκιμών θα πρέπει να είναι σχεδιασμένο έτσι ώστε να εντοπίζει τις μεταβολές αυτές και να πραγματοποιεί από την αρχή έλεγχο, κάθε φορά, σε όλα τα κομμάτια του λογισμικού. Όσο η τεχνολογία εξελίσσεται, η διαδικασία ανάπτυξης λογισμικού γίνεται ακόμη πιο εύκολη και σύντομη. Ένα εργαλείο αυτοματοποιημένου ελέγχου, θα πρέπει και αυτό να είναι σύντομο, ακριβές και ευμετάβλητο, για να χαρακτηριστεί αποδοτικό. Θα πρέπει, δηλαδή, να προσαρμόζεται στις συνεχώς μεταβαλλόμενες συνθήκες και απαιτήσεις του περιβάλλοντος.

2.4.2.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΕΝΟΣ ΕΠΙΤΥΧΗΜΕΝΟΥ ΕΡΓΑΛΕΙΟΥ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΩΝ ΕΛΕΓΧΩΝ ΛΟΓΙΣΜΙΚΟΥ

Ο απώτερος σκοπός ενός πετυχημένου αυτοματοποιημένου εργαλείου ελέγχου λογισμικού η μείωση, χρονικά, του κύκλου ζωής δοκιμών και ταυτόχρονα η εξέταση του μεγαλύτερου μέρους του δοθέντος λογισμικού. Παράλληλα θα πρέπει να εξοικονομεί χρήματα στην εταιρεία διατηρώντας πάντα την αίσθηση ασφάλειας πως τα παραγόμενα προϊόντα είναι πρωτίστως ποιοτικά.

Προκειμένου να επιτευχθούν τα παραπάνω, το πρόγραμμα αυτοματοποιημένου ελέγχου θα πρέπει να έχει τα εξής χαρακτηριστικά [16]:

1. Να αξιοποιεί πλήρως το ταλέντο και τους διαθέσιμους πόρους.
2. Να μεγιστοποιεί την αίσθηση της ομάδας και της συνεργασίας.
3. Να εκμεταλλεύεται και να βελτιστοποιεί και τους τέσσερις βασικούς τομείς των χειρωνακτικών ελέγχων: ανάπτυξη πλήθους ξεχωριστών μεθόδων ελέγχων (test cases), εφαρμογή αυτών, ανάλυση αποτελεσμάτων και συγγραφή αναφορών (reports).
4. Να ελαχιστοποιεί την ανάγκη δημιουργίας νέων κομματιών κώδικα ελέγχου (test scripts).
5. Να παράγει μεθόδους ελέγχων που μπορούν επαναχρησιμοποιηθούν στο μέλλον.
6. Να αξιοποιεί πλήρως το ήδη υπάρχον σύνολο ελέγχων.
7. Να είναι ευμετάβλητο και επεκτάσιμο.

3

ΑΞΙΟΛΟΓΗΣΗ ΠΟΙΟΤΗΤΑΣ

Η ποιότητα δεν αποτελεί ποτέ ένα ατύχημα· είναι πάντα το αποτέλεσμα μιας ιδιοφυούς προσπάθειας.
John Ruskin

3.1 ΠΟΙΟΤΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΛΟΓΙΣΜΙΚΟΥ

Τις τελευταίες δυο δεκαετίες, το μέγεθος και ο βαθμός της πολυπλοκότητας των συστημάτων λογισμικού σημειώνουν ραγδαία αύξηση. Όταν τα συστήματα αυτά τεθούν σε λειτουργία είναι πολύ πιθανό να εμφανίσουν δυσλειτουργίες. Οι ζημίες αυτές μπορούν να οδηγήσουν σε οικονομικές απώλειες, δυσφήμηση στην εταιρεία παραγωγής λογισμικού, ακόμη και χρεοκοπία. Επιπλέον, η εξοικείωση των καθημερινών ανθρώπων με τους υπολογιστές, η αξιοσημείωτη πρόοδος στα δίκτυα υπολογιστών, ο παγκόσμιος ιστός(World Wide Web) σε συνδυασμό με την ανάπτυξη υπολογιστικών συστημάτων φιλικών προς τον χρήστη, έχουν αυξήσει κατακόρυφα τις προσδοκίες των καθημερινών χρηστών. Παρότι που πλήθος ανταγωνιστικών προϊόντων προσφέρουν παρόμοιες λειτουργίες, εντούτοις εκείνα που τελικώς επιβιώνουν είναι τα προϊόντα χαμηλότερου κόστους που ταυτόχρονα καταφέρνουν τα έχουν τα καλύτερα ποιοτικά χαρακτηριστικά[20]. Αναγνωρίζοντας αυτό το ρίσκο, οι μηχανικοί παραγωγής λογισμικού, παγκοσμίως, έχουν συμφωνήσει να δίνουν έμφαση σε εκείνες τις τεχνικές ανάπτυξης λογισμικού που σκοπό έχουν να ελαχιστοποιούν τον αριθμό των ατελειών του, να εντοπίζουν και να διορθώνουν τα λάθη, όπου και αν αυτά εμφανιστούν. Ακόμη καλύτερα να προβλέπουν και να αποτρέπουν την εμφάνισή τους. Ταυτόχρονα θα πρέπει, οι εν λόγω τεχνικές να γίνουν όσο πιο οικονομικές γίνεται και να παράγουν ποιοτικά λογισμικά[21]. Ο έγκαιρος εντοπισμός και η άμεση διόρθωση των λαθών είναι άκρως σημαντικές διαδικασίες. Η προσπάθεια και το κόστος που απαιτείται για την απαλοιφή των λαθών σε ένα σύστημα λογισμικού, αυξάνονται κατακόρυφα όταν αυτά εντοπίζονται προς το τέλος του κύκλου ζωής του. Πόσο μάλλον όταν αυτά εντοπίζονται πολύ αργότερα, από τους πελάτες. Τα προγράμματα λοιπόν, θα πρέπει να αξιολογούνται, κάθε μέρα. Η αξιολόγηση της ποιότητας λογισμικού επέκτεινεται σε πολλά διαφορετικά παρακλάδια και δεν περιορίζεται μόνο στην εξασφάλιση λειτουργίας του λογισμικού. Διαφορετικοί άνθρωποι έχουν διαφορετικές προσδοκίες από ένα προϊόν λογισμικού. Η διαφοροποίηση αυτή οφείλεται στους διαφορετικούς ρόλους, ευθύνες και προσεγγίσεις της κάθε ομάδας ανθρώπων[22]. Πλήθος παραγόντων επηρεάζουν την παραγωγή και την αγορά προϊόντων λογισμικού. Οι παράγοντες αυτοί πηγάζουν από τις ανάγκες και τις προσδοκίες του εκάστοτε αγοραστή, την υπόληψη της εταιρείας, τα προσδοκόμενα χαρακτηριστικά κάθε προϊόντος και την αντικειμενική αξία του λογισμικού. Προκειμένου να γίνουν κατανοητοί οι παράγοντες που επηρεάζουν την ποιότητα, θα πρέπει η ποιότητα να μελετηθεί από μια πιο ευρεία σκοπιά. Ο D.Garvin[23] έχει αναλύσει πώς η ποιότητα γίνεται αντιληπτή με διαφορετικούς τρόπους σε διαφορετικούς τομείς. Σύμφωνα με τον Garvin λοιπόν, η ποιότητα διαχωρίζεται και αναλύεται από πέντε διαφορετικές σκοπιές:

- Εμπειρική οπτική (transcendental view)
  Κατά την εμπειρική θεώρηση, η ποιότητα είναι κάτι το οποίο μπορεί να οριστεί μέσω της εμπειρίας. Η ποιότητα θεωρείται κάτι το ιδανικό, το οποίο είναι πολύ περίπλοκο να προσδιοριστεί επακριβώς. Ωστόσο, ένα καλό αντικείμενο ποιότητας ξεχωρίζει και αναγνωρίζεται εύκολα. Λόγω της φιλοσοφικής φύσης της, η υπερβατική άποψη δεν μπορεί να εκφραστεί χρησιμοποιώντας συγκεκριμένες μετρήσεις.
- Η οπτική του τελικού χρήστη (user view)
  Η ποιότητα θα πρέπει να είναι κατάλληλη ώστε να ικανοποιεί τον σκοπό και τις ανάγκες και τις προσδοκίες των χρηστών. Η ποιότητα δεν εξετάζεται μόνο ως προς τις δυνατότητες που δίνει ένα προϊόν στον χρήστη του, αλλά και αν αυτό είναι ευρείας χρήσης. Δηλαδή, για τον τελικό χρήστη, ένα προϊόν είναι καλής ποιότητας εάν ικανοποιεί τις ανάγκες ενός μεγάλου αριθμού πελατών. Θα πρέπει να προσδιοριστούν εδώ, ποια ακριβώς χαρακτηριστικά θεωρούν οι χρήστες σημαντικά. Πολλά από αυτά,

κρίνονται υποκειμενικά ανάλογα τις συνθήκες. Παραδείγματα υποκειμενικών χαρακτηριστικών είναι η χρηστικότητα, η αξιοπιστία, η αποδοτικότητα, η δυνατότητα δοκιμής κλπ.

- Η οπτική της βιομηχανίας παραγωγής λογισμικού (manufacturing view)
  Για τις εταιρείες, ποιότητα σημαίνει, συμμόρφωση με τα πρότυπα της διαδικασίας. Η άποψη της βιομηχανίας για την ποιότητα, γεννάται ιστορικά, από τις κατασκευαστικές εταιρείες, όπως οι βιομηχανίες αυτοκινήτων, ηλεκτρονικών κοκ. Στην οπτική αυτή, ποιοτικό κρίνεται ένα προϊόν μόνο εάν συμμορφώνεται με τις προκαθορισμένες απαιτήσεις. Οποιαδήποτε απόκλιση από τους προγραμματισμένους στόχους κοστίζει και μειώνει την ποιότητα. Τα προϊόντα θα πρέπει να κατασκευαστούν, από την πρώτη φορά, σωστά, έστι ώστε το κόστος ανάπτυξης και συντήρησης να είναι το ελάχιστο δυνατό. Ωστόσο, δεν υπάρχει καμία εγγύηση, ότι η συνέπεια στα πρότυπα της διαδικασίας θα οδηγήσει με βεβαιότητα στην παραγωγή καλών προϊόντων. Η ποιότητα του προϊόντος μπορεί να ενισχυθεί διαδοχικά με την συνεχή βελτίωση της διαδικασίας. Η συνεχής προσπάθεια βελτίωσης, είναι μια τακτική που χρησιμοποιείται ευρέως σήμερα, από την πλειονότητα των βιομηχανιών.
- Η οπτική για το ίδιο το προϊόν (product view)
  Η κεντρική ιδέα πίσω από αυτή την άποψη είναι πως, εάν ένα προϊόν έχει καλές εσωτερικές ιδιότητες, τότε θα έχει και καλές εξωτερικές ιδιότητες. Η άποψη αυτή, χρήζει έρευνας διότι δημιουργεί τηνευκαιρεία να εξερευνηθεί η σχέση μεταξύ των εσωτερικών και εξωτερικών χαρακτηριστικών ενός προϊόντος. Κατά την οπτική αυτή, το τρέχον επίπεδο ποιότητας ενός προϊόντος, υποδηλώνει την παρουσία ή την απουσία κάποιων εσωτερικών μετρήσιμων χαρακτηριστικών. Η οπτική αυτή, αξιολογεί την ποιότητα βάσει αντικειμενικών στοιχείων.
- Η αξιόλογηση της ποιότητας βάσει της αξίας (value based view)
  Η παρούσα άποψη, συγχωνεύει δύο ανεξάρτητες έννοιες, την αριστεία και την τιμή. Η ποιότητα είναι ένα μέτρο αριστείας και η αξία είναι ένα μέτρο της τιμής. Η κεντρική ιδέα στην άποψη που βασίζεται στην αξία είναι το πόσο ένας πελάτης είναι διατεθειμένος να πληρώσει για ένα συγκεκριμένο επίπεδο ποιότητας. Η πραγματικότητα είναι πως η ποιότητα δεν έχει νόημα αν ένα προϊόν δεν έχει λογική τιμή. Ουσιαστικά η εν λόγω οπτική, βασίζεται στην προσπάθεια να υπάρξει ισορροπία μεταξύ ποιότητας και κόστους.

3.1.1 ΤΑΞΙΝΟΜΗΣΗ ΤΩΝ ΑΠΑΙΤΗΣΕΩΝ ΛΟΓΙΣΜΙΚΟΥ

Η ανάγκη για ορισμό και ταξινόμηση των απαιτήσεων λογισμικού πηγάζει από την απαίτηση κάλυψης όλων των χαρακτηριστικών του λογισμικού και των πτυχών του. Μερικές πτυχές είναι η χρηστικότητα, η δυνατότητα επαναχρησιμοποίησης, η συντηρησιμότητα και ούτω καθεξής. Όλες οι παραπάνω ιδιότητες αποβλέπουν σε ένα κοινό σκοπό, τη διασφάλιση της ικανοποίησης του τελικού χρήστη. Η μεγάλη ποικιλία ζητημάτων που σχετίζονται με τις διάφορες ιδιότητες του λογισμικού, την χρήση και τη διατήρησή του, μπορούν τα ταξινομηθούν σε ομάδες διαφορετικού περιεχομένου, τους παράγοντες ποιότητας (Quality factors)[20]. Η ομάδα που είναι υπεύθυνη για τον ορισμό των απαιτήσεων ενός συστήματος λογισμικού, είναι υπεύθυνη και να ταξινομήσει τις απαιτήσεις στις ομάδες παραγόντων. Τα έγγραφα στα οποία καταγράφονται οι προσδοκώμενες απαιτήσεις, διαφέρουν ως προς τους παράγοντες που αναλύουν και ουσιαστικά αντανακλούν τις βασικές διαφορές μεταξύ των προγραμμάτων λογισμικού. Έτσι, επιτυγχάνεται η παγκόσμια κάλυψη όλων των παραγόντων σε όλα τα έγγραφα απαιτήσεων. Οι πέντε βασικές οπτικές, που αναπτύχθηκαν στην παράγραφο 2.1, μέσα από τις οποίες μελετάται η ποιότητα, βοηθούν στην κατανόηση των διαφορετικών απόψεων πάνω στην επιστήμη της ποιότητας. Από την άλλη μεριά, οι αντικειμενικές μετρήσεις δίνουν μια ποσοτική άποψη της έννοιας της ποιότητας. Παρακάτω εξηγούνται οι λόγοι για τους οποίους, δημιουργήθηκε η ανάγκη να οριστούν τρόποι ποσοτικής απεικόνισης της ποιότητας των συστημάτων λογισμικού[24]:

- Οι μετρήσεις, επιτρέπουν στους προγραμματιστές να γνωρίζουν το ελάχιστο επίπεδο ποιότητας που πρέπει να προσφέρουν, προκειμένου να γίνει ένα λογισμικό δεκτό.
- Τα μοντέλα παραγωγής λογισμικού εμπεριέχουν την συνεχόμενη προσπάθεια βελτίωσης του λογισμικού αλλά και της διαδικασίας παραγωγής του. Οι εταιρείες θα πρέπει να γνωρίζουν πως η βελτίωση της ποιότητας επιτυγχάνεται με ένα συγκεκριμένο κόστος. Αυτή η αλληλοεξαρτώμενη σχέση ποιότητας-κόστους θα πρέπει εξαρχής να είναι γνωστή στους υπεύθυνους διαχείρισης έργων λογισμικού. Κάποιες φορές αξίζει η επένδυση στην ποιότητα, άλλες φορές επιβάλλεται η αποφυγή κόστους. Τις περισσότερες φορές, επιδιώκεται η ισορροπία μεταξύ των δύο.
- Το τρέχον επίπεδο ποιότητας ενός προϊόντος πρέπει να αξιολογείται, έτσι ώστε να καθορίζεται έγκαιρα η ανάγκη βελτίωσής του.

Η οπτική του χρήστη για την ποιότητα περικλείει πλήθος παραγόντων που επηρεάζουν την ποιότητα, όπως η λειτουργικότητα, η αξιοπιστία και η χρηστικότητα. Είναι εύκολο να υπολογίσει κάποιος, πόσες από τις παραπάνω ιδιότητες έχει ένα προϊόν λογισμικού, εφαρμόζοντας τουλάχιστον μια δοκιμαστική συνθήκη ελέγχου

για κάθε μία από αυτές. Ένα προϊόν ενδέχεται να χρειαστεί πολλαπλές δοκιμαστικές περιπτώσεις εκτέλεσης, σε διαφορετικά περιβάλλοντα εφαρμογής, για το ίδιο χαρακτηριστικό. Ο λόγος του πλήθους επιτυχημένων δοκιμαστικών ελέγχων προς τον συνολικό αριθμό των δοκιμαστικών ελέγχων είναι αποτελεί μέτρο των συνολικών λειτουργιών που παρέχεται από το λογισμικό. Ανάμεσα στους παράγοντες που πηγάζουν από την οπτική του πελάτη, συμπεριλαμβάνεται η έννοια της αξιοπιστίας, η οποία έχει προσεγγίσει το μεγαλύτερο ερευνητικό ενδιαφέρον. Η οπτική της ποιότητας από την πλευρά των δημιουργών λογισμικού, επικεντρώνεται κυρίως σε δύο σημεία ενδιαφέροντος: α)Πόσα ελαττώματα έχουν εντοπιστεί και β)Πόσο κοστίζει η αποκατάστασή των ελαττωμάτων. Η καταγραφή των ελαττωμάτων δίνει μια εμπεριστατωμένη εικόνα της ποιότητας της εργασίας. Η εν λόγω μέτρηση είναι σημαντική μόνο όταν γίνεται αξιοποιήσιμη. Δηλαδή, μόνο εάν μπορεί να γίνει κάτι έτσι ώστε να βελτιωθεί η διαδικασία παραγωγής λογισμικού, λαμβάνοντας υπ'όψιν τα καταγγεγραμένα λάθη. Τότε μόνο, στα επόμενα έργα λογισμικού οι ατέλειες θα είναι λιγότερες[25]. Για κάθε ελάττωμα προσδιορίζεται η φάση ανάπτυξης στην οποία εισήχθη και η φάση στην οποία ανακαλύφθηκε. Για παράδειγμα αν εντοπιστεί ένα λάθος που προέρχεται από την φάση συλλογής απαιτήσεων, εξάγεται το συμπέρασμα πως η φάση ανάλυσης απαίτησης δεν εκτελέστηκε επαρκώς και συγχρόνως, οι φάσεις που ακολούθησαν δεν ήταν υψηλού επιπέδου. Εάν εντοπιστεί πλήθος ελαττωμάτων κατά τη διάρκεια λειτουργίας του συστήματος, εξάγεται το συμπέρασμα πως κατά τη διάρκεια εκτέλεσης δοκιμών σε αυτό, δεν δόθηκε η απαιτούμενη προσοχή. Τα ελαττώματα που ενδεχομένως εντοπιστούν χωρίζονται σε ενότητες. Υποθέτοντας πως κάθε ενότητα είναι συνεκτική, ξεχωρίζοντας τις ενότητες που περιέχουν τα περισσότερα ελαττώματα, είναι εύκολο να εντοπιστεί η πηγή των λαθών. Για παράδειγμα, εάν εντοπιστεί μεγάλος αριθμός ελαττωμάτων στην κατηγορία επικοινωνία σε μια εφαρμογή, αυτό σημαίνει πως περισσότεροι πόροι θα πρέπει να διατεθούν για την βελτίωση του συστήματος επικοινωνίας. Επιπλέον, για να επιτευχθεί σωστή σύγκριση και να εξαχθούν βέβαια συμπεράσματα, οι μηχανικοί ελέγχου λογισμικού, θα πρέπει να ταξινομούν τα ελαττώματα βάσει του μεγέθους του τελικού προϊόντος. Ταυτόχρονα, δεν θα πρέπει ποτέ να ξεχνούν την αναγκαιότητα επίτευξης μέγιστης ποιότητας του προϊόντος που ελέγχουν, συνυπολογίζοντας τον προορισμό και τον ρόλο του.

3.1.2 ΠΑΡΑΓΟΝΤΕΣ ΠΟΙΟΤΗΤΑΣ

Ένας παράγοντας ποιότητας αντιπροσωπεύει μια χαρακτηριστική συμπεριφορά ενός συστήματος. Μερικά παραδείγματα παραγόντων ποιότητας υψηλού επιπέδου είναι η ορθότητα, η αξιοπιστία, η αποτελεσματικότητα, η φορητότητα και η δυνατότητα επαναχρησιμοποίησης. Οι παράγοντες ποιότητας είναι εξωτερικά χαρακτηριστικά ενός συστήματος λογισμικού. Οι τελικοί χρήστες, οι προγραμματιστές, μηχανικοί εξασφάλισης ποιότητας ενδιαφέρονται με διαφορετικό τρόπο για κάθε έναν από τους παράγοντες. Για παράδειγμα, οι πελάτες απαιτούν ένα αποδοτικό και αξιόπιστο λογισμικό, χωρίς να ενδιαφέρονται ιδιαίτερα για την φορητότητα. Οι προγραμματιστές προσπαθούν να ανταποκριθούν στις ανάγκες των χρηστών κάνοντας το σύστημα τους αποτελεσματικό και αξιόπιστο, ενώ ταυτόχρονα έχουν ως στόχο την φορητότητα και το χαμηλό κόστος ανάπτυξης. Η ομάδα διασφάλισης ποιότητας λογισμικού, ενδιαφέρεται περισσότερα για τη δυνατότητα δοκιμής του λογισμικού, έτσι ώστε η παρουσία ή η απουσία άλλων παραγόντων, όπως η ευχρηστία και η αποτελεσματικότητα, να μπορεί εύκολα εντοπιστεί μέσω των δοκιμών.

3.1.3 ΜΟΝΤΕΛΑ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ

Ποικίλα μοντέλα των παραγόντων ποιότητας λογισμικού έχουν προταθεί και έχουν εφαρμοστεί στην ιστορία των υπολογιστικών συστημάτων. Κάθε μοντέλο αποτελείται από ποιοτικά χαρακτηριστικά ή παράγοντες ποιότητας. Τα χαρακτηριστικά αυτά χρησιμοποιούνται για να διαμορφωθεί μια ολοκληρωμένη άποψη για την ποιότητα του λογισμικού ή προϊόντος[26]. Σύμφωνα με τον Wallmüller[27], «ένα από τα παλιότερα και πιο συχνά εφαρμοσμένο μοντέλο παραγόντων ποιότητας είναι αυτό του McCall». Το μοντέλο McCall σχεδιάστηκε και αναπτύχθηκε το 1977 από το τμήμα ηλεκτρονικών συστημάτων αεροσκαφών των ΗΠΑ, το αεροδρόμιο της Ρώμης και την General Electric, με κοινό στόχο την βελτίωση της ποιότητας των προϊόντων λογισμικού. Το μοντέλο παραγόντων ποιότητας του McCall, είναι ένα από τα πιο γνωστά πρότυπα ομαδοποίησης χαρακτηριστικών λογισμικού. Ο κύριος σκοπός του McCall ήταν να καταστεί η ποιότητα μετρήσιμη. Το αρχικό μοντέλο περιέχει έντεκα παράγοντες. Μεταγενέστερα μοντέλα, τα οποία περιέχουν δώδεκα με δεκαπέντε παράγοντες, προτάθηκαν από τους Deutcsh και Willis περί το 1988[28] και από τους Evans και Marciniak το 1987[29]. Και τα δύο αυτά μοντέλα δεν διαφέρουν κατά πολύ από του McCall. Η δομή του McCall για τους παράγοντες ποιότητας, παρότι διανύει ηλικιακά την πέμπτη δεκαετία ύπαρξής της, συνεχίζει να παρέχει τις βάσεις εκείνες που βοηθούν τους σύγχρονους μηχανικούς ελέγχου ποιότητας να ταξινομούν τις απαιτήσεις και συνεπώς τα χαρακτηριστικά των λογισμικών. Άλλα πρότυπα μοντέλα των παραγόντων ποιότητας είναι αυτά που έχουν αναπτυχθεί από τους Boehm[30], Dromey[31] και φυσικά δεν θα πρέπει να παραληφθεί το πρότυπο ISO 9126 το οποίο εξελίχθηκε σε διεθνές πρότυπο από το διεθνή οργανισμό τυποποίησης ISO.

3.1.3.1 ΤΟ ΜΟΝΤΕΛΟ MCCALL ΓΙΑ ΤΟΥΣ ΠΑΡΑΓΟΝΤΕΣ ΠΟΙΟΤΗΤΑΣ

Στην παράγραφο αυτή παρουσιάζεται το μοντέλο ποιότητας κατά McCall, ο οποίος ξεκίνησε με 55 χαρακτηριστικά που επιδρούν στην ποιότητα και καλούνται παράγοντες. Για λόγους απλότητας, ο McCall αργότερα μείωσε τον αριθμό των παραγόντων και κατέληξε σε έντεκα και είναι οι ακόλουθοι:

Πίνακας 1: Οι παράγοντες ποιότητας κατά McCall

ΠΑΡΑΓΟΝΤΕΣ	ΣΗΜΑΣΙΑ
Αποδοτικότητα (Efficiency)	Ο βαθμός κατά τον οποίο χρησιμοποιεί ένα σύστημα λογισμικού πόρους, όπως η ισχύς του υπολογιστή, η μνήμη, ο χώρος στο δίσκο, η ενέργεια και ο όγκος του κώδικα. Ένα σύστημα λογισμικού πρέπει να χρησιμοποιεί τους ελάχιστους δυνατούς πόρους για την εκτέλεση των λειτουργιών του.
Ακεραιότητα (Integrity)	Η ικανότητα του συστήματος να είναι ασφαλές σε πιθανές επιθέσεις. Δηλαδή, η ακεραιότητα αναφέρεται στον βαθμό κατά τον οποίο η πρόσβαση σε λογισμικό ή δεδομένα από μη εξουσιοδοτημένα άτομα ή προγράμματα, είναι ελεγχόμενη.
Ευχρηστία (Usability)	Ένα σύστημα λογισμικού θεωρείται εύχρηστο, εάν το βρίσκουν οι χρήστες εύκολο να το χρησιμοποιήσουν. Χωρίς ένα καλώς δομημένο περιβάλλον χρήσης, ένα σύστημα λογισμικού μπορεί να εξαφανιστεί, ακόμη και αν διαθέτει πολλές επιθυμητές ιδιότητες.
Ορθότητα (Correctness)	Ένα σύστημα λογισμικού θα πρέπει να ανταποκρίνεται στις καθορισμένες λειτουργικές απαιτήσεις και να μην εμφανίσει ποτέ τις μη αναμενόμενες. Εάν ένα λογισμικό πληροί τις λειτουργικές απαιτήσεις, λέγεται πως είναι ορθό.
Συντηρησιμότητα (Maintainability)	Η ικανότητα των προϊόντων να διατηρούνται ακέραια, παρά της χρόνιας συνεχιζόμενης χρήσης τους. Επίσης, η συντηρησιμότητα αναφέρεται στο πόσο εύκολα και οικονομικά μπορούν να εκτελεστούν οι εργασίες συντήρησης.
Ελεγξιμότητα (Testability)	Η ικανότητα επαλήθευσης ικανοποίησης των απαιτήσεων σχεδιασμού και ανάπτυξης. Σε κάθε στάδιο ανάπτυξης του λογισμικού, είναι απαραίτητη η εξέταση του λογισμικού για το εάν ικανοποιεί τις προδιαγραφές χρήσης και λειτουργίας χωρίς λάθη και ατέλειες.
Ευελιξία (Flexibility)	Η ευελιξία αντικατοπτρίζει το κόστος τροποποίησης ενός συστήματος λογισμικού. Αν το αρχικό σύστημα δεν είναι ευέλικτο, είναι πιθανόν οι μεταγενέστερες αλλαγές να εφαρμόζονται με όλο και περισσότερο κόστος. Η ευελιξία ενός συτήματος είναι η απάντηση στο ερώτημα: Πόσο εύκολα μπορεί να προσθέσει κάποιος ένα νέο χαρακτηριστικό στο σύστημα;
Επαναχρησιμοποιησιμότητα (Reusability)	Η δυνατότητα ενός λογισμικού, να επαναχρησιμοποιηθεί σε άλλο προϊόν, ίσως με κάποιες μικρές τροποποιήσεις. Η επαναχρησιμοποιησιμότητα εξοικονομεί κόστος και χρόνο, με την έννοια πως δεν χρειάζεται η ανάπτυξη μέρους του κώδικα, που χρησιμοποιείται αυτούσιο σε διαφορετικό περιβάλλον ή προϊόν.
Μεταφερσιμότητα (Portability)	Η ικανότητα ενός συστήματος λογισμικού να προσαρμόζεται και να λειτουργεί σε διαφορετικό περιβάλλον εκτέλεσης. Το περιβάλλον είναι ένας ευρύς όρος που περιλαμβάνει την πλατφόρμα υλικού (hardware system), το λειτουργικό σύστημα, το οργανωτικό περιβάλλον κ.α.
Διαλειτουργικότητα (Interoperability)	Τα σύγχρονα συτήματα λογισμικού συνδέονται μεταξύ τους μέσω των δικτύων. Διαλειτουργικότητα είναι η δυνατότητα του λογισμικού να αλληλεπιδρά με ένα ή περισσότερα προεγκατεστημένα ή προκαθορισμένα συστήματα.
Αξιοπιστία (Reliability)	Η αξιοπιστία είναι η δυνατότητα ενός προϊόντος να συνεχίζει να επιτελεί τον προβλεπόμενο σκοπό του σε μια μεγάλη χρονική περίοδο εντός προκαθορισμένων συνθηκών. Είναι ένα σύνολο χαρακτηριστικών που είναι φορείς της δυνατότητας του λογισμικού να διατηρεί το επίπεδο απόδοσής του σε καθορισμένες συνθήκες και για προκαθορισμένη χρονική περίοδο.

Στον Πίνακας 1, στη δεύτερη στήλη σημειώνεται και η έννοια κάθε χαρακτηριστικού το οποίο κρίνει την ποιότητα. Χρησιμοποιώντας αυτή την ομαδοποίηση, ο McCall προσπάθησε να γεφυρώσει το κενό μεταξύ των χρηστών και των προγραμματιστών, δίνοντας έμφαση σε εκείνους τους παράγοντες ποιότητας που συνυπολογίζουν και τις ανάγκες των χρηστών αλλά και τις προτεραιότητες των προγραμματιστών[1]. Η δομή του εν λόγω προτύπου, κατηγοριοποιεί τους παράγοντες σε τρεις ξεχωριστές ομάδες:

- Λειτουργία προϊόντος (Product operation)
- Αναθεώρηση προϊόντος (Product revision)
- Μεταφορά προϊόντος (Product transition)

Κάθε ένας από αυτούς τους παράγοντες, έχει μια συλλογή από κριτήρια ποιότητας και κάθε κριτήριο ποιότητας μπορεί να δείχνει σε μια ή σε περισσότερες μετρικές ποιότητας. Το μοντέλο αυτό με τις αντίστοιχες ομαδοποιήσεις φαίνεται σχηματικά στον Πίνακας 2. Από τα ονόματα των τριών επιπέδων κατηγοριοποίησης το μοντέλο αυτό ονομάστηκε FCM (Factors-Criteria-Metrics). Ο κύριος σκοπός του McCall είναι η δομή κατηγοριοποίησης παραγόντων να παρέχει μια ολοκληρωμένη εικόνα της ποιότητας λογισμικού. Οι μετρικές ποιότητας υπολογίζονται απαντώντας «ναι» ή «όχι» σε δεδομένες ερωτήσεις. Τα κριτήρια ποιότητας που φαίνονται στον Πίνακας 2, είναι χαρακτηριστικά ενός παράγοντα ποιότητας, τα οποία σχετίζονται με την ανάπτυξη λογισμικού[22] και μπορούν να μετρηθούν άμεσα με μετρικές (metrics). Όπως φαίνεται στον Πίνακας 2, ο McCall πρότεινε 11 παράγοντες ποιότητας, 25 κριτήρια και 41 μετρικές. Το μοντέλο αυτό αποτέλεσε ένα από τα πιο ολοκληρωμένα μοντέλα για την εποχή του, έγινε παράδειγμα για πολλούς και αποτέλεσε την βάση νέων, εναλλακτικών μοντέλων όπως το διεθνές πρότυπο ISO 9126. Αρκετές επιχειρήσεις σήμερα, βασίζονται στην λογική του McCall και ο βασικότερος λόγος είναι ότι είναι πιο αναλυτικό και πιο προσαρμοστικό από τα αντίστοιχα σύγχρονα.

Πίνακας 2: Μοντέλο FCM (Factors-Criteria-Metrics)

Κατηγορίες Ποιότητας	Παράγοντες Ποιότητας	Ευρείς Στόχοι	Κριτήρια Ποιότητας	
Λειτουργία Προϊόντος	Ορθότητα	Κάνει ό,τι θέλει ο πελάτης;	Traceability Completeness Consistency	
	Αξιοπιστία	Το κάνει με ακρίβεια συνεχώς;	Consistency Accuracy Error tolerance	
	Αποδοτικότητα	Επιλύει γρήγορα το πρόβλημα που προκύπτει ;	Execution efficiency Storage efficiency	
	Ακεραιότητα	Είναι ασφαλές;	Access control Access audit	
	Ευχρηστία	Μπορώ να το τρέξω;	Operability Training Communicativeness	
				ΜΕΤΡΙΚΕΣ

Αναθεώρηση Προϊόντος	Συντηρησιμότητα	Μπορεί να διορθωθεί;	Simplicity Conciseness Self-descriptiveness Modularity	ΠΟΙΟΤΗΤΑΣ
	Ελεγξιμότητα	Μπορεί να δοκιμαστεί;	Simplicity Instrumentation Self-descriptiveness Modularity	
	Ευελιξία	Μπορεί να αλλάξει;	Self-descriptiveness Expandability Generality	
Μεταφορά Προϊόντος	Μεταφερσιμότητα	Μπορεί να χρησιμοποιηθεί σε άλλο μηχάνημα;	Self-descriptiveness Machine independence S/W system independence	
	Επαναχρησιμοποιησιμότητα	Μπορούν τα μέρη του να επαναχρησιμοποιηθούν;	Generality Self-descriptiveness Modularity Machine independence S/Wsystem independence	
	Διαλειτουργικότητα	Μπορεί να διασυνδέεται με άλλο σύστημα;	Modularity Comms commonality Data commonality	

3.1.3.2 ΤΟ ΠΡΟΤΥΠΟ ISO 9126

Το 1991, η ISO δημοσιεύει ένα διεθνώς αποδεχόμενο πρότυπο για την εκτίμηση των χαρακτηριστικών ποιότητας λογισμικού· το πρότυπο αυτό ονομάστηκε «Λογισμικό Αξιολόγησης προϊόντος – Χαρακτηριστικά ποιότητας και κατευθυντήριες γραμμές για την χρήσης τους»[26]. Η τρέχουσα έκδοση της σειράς ISO 9126 αποτελείται από ένα διεθνές πρότυπο και τρεις τεχνικές αναφορές:

1. ISO IS 9126-1: Μοντέλο ποιότητας[ISO,2001]
2. ISO TR 9126-2: Εξωτερικές μετρήσεις [ISO, 2003].
3. ISO TR 9126-3: Εσωτερικές μετρήσεις [ISO, 2003].
4. ISO TR 9126-4: Μετρικές ποιότητας κατά την χρήση [ISO,2004].

Το πρώτο έγγραφο της σειράς ISO 9126 –Μοντέλο Ποιότητας- περιλαμβάνει δύο μέρη για την μελέτη της ποιότητας λογισμικού:

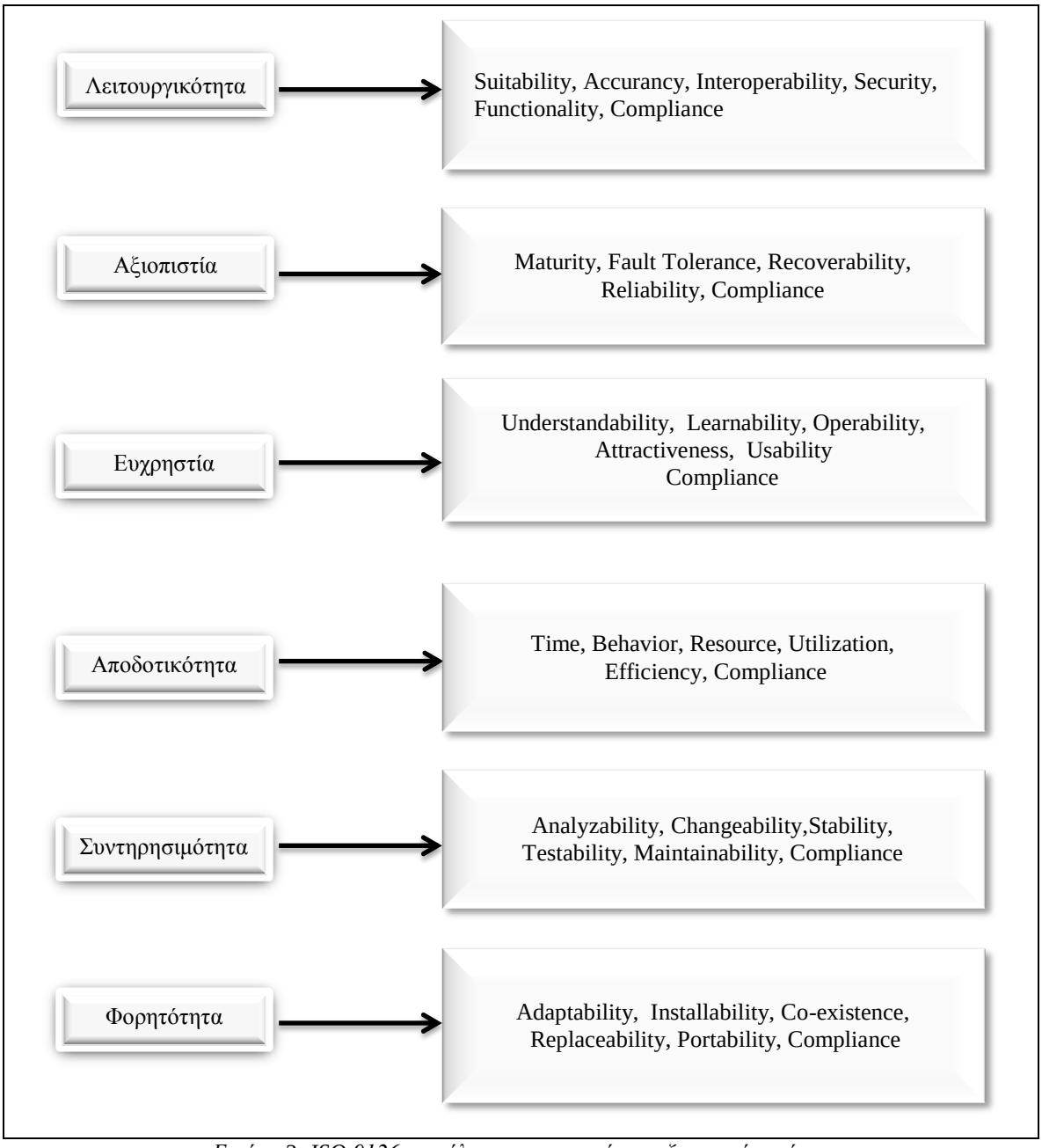
1. Εσωτερικό και εξωτερικό μοντέλο ποιότητας.
2. Μοντέλο ποιότητας κατά την χρήση.

Το πρώτο από τα παραπάνω μοντέλα, προσδιορίζει έξι χαρακτηριστικά τα οποία υποδιαιρούνται σε είκοσι-επτά χαρακτηριστικά για εσωτερική και εξωτερική ποιότητα, όπως φαίνεται στην Εικόνα 3. Το δεύτερο μέρος του μοντέλου δείχνει σε τέσσερα χαρακτηριστικά ποιότητας κατά την χρήση όπως φαίνεται στην Εικόνα 4. Επειδή το μοντέλο ποιότητας ISO/IEC 9126-1 είναι γενικό, μπορεί να εφαρμοστεί σε οποιοδήποτε προϊόν λογισμικού. Επιπλέον, ορίζει πλήθος μετρικών για την αξιολόγηση των υπο-χαρακτηριστικά του[32]. Στο ISO/IEC 9126, ικανοποίηση έρχεται μόνο εάν ένα προϊόν λογισμικού είναι σε θέση να ανταποκρίνεται στις προσδοκίες των χρηστών. Το πρότυπο ISO 9126 είναι ένα χαρακτηριστικό παράδειγμα μοντέλου παραγόντων ποιότητας, το οποίο αποσυνθέτει τα ποιοτικά χαρακτηριστικά σε πιο εξειδικευμένα χαρακτηριστικά. Για παράδειγμα το βασικό χαρακτηριστικό συντηρησιμότητα έχει διαιρεθεί σε τέσσερα συγκεκριμένα χαρακτηριστικά γνωρίσματα, τη δυνατότητα ανάλυσης, την μεταβλητότητα, την σταθερότητα και την ελεγξιμότητα. Τα είκοσι εξειδικευμένα γνωρίσματα ποιότητας, τα οποία φαίνονται στην Εικόνα 3, ορίζονται ως εξής[33]:

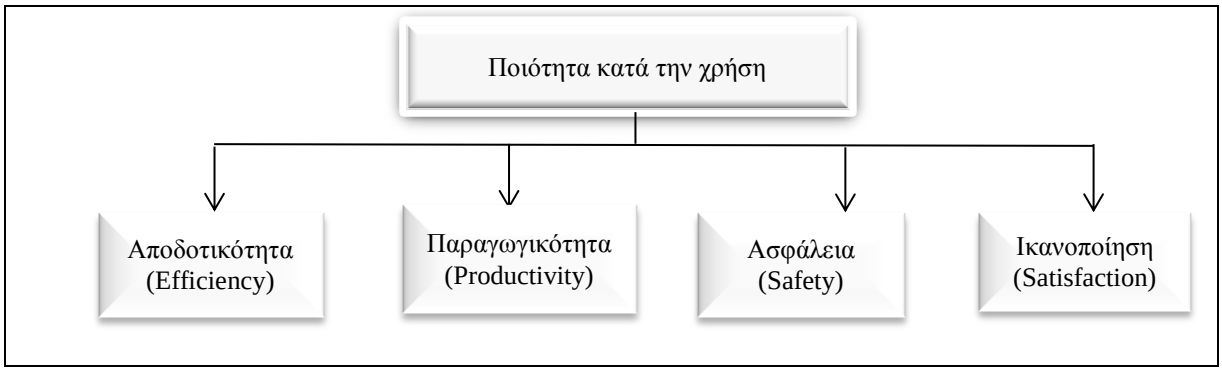
- Καταλληλότητα (Suitability)
  Η ικανότητα του λογισμικού να παρέχει επαρκές σύνολο λειτουργιών για συγκεκριμένες εργασίες και στόχους του χρήστη.
- Ακρίβεια (Accuracy)
  Η δυνατότητα του λογισμικού να συμφωνεί με τα αποτελέσματα ή τις επιπτώσεις
- Διαλειτουργικότητα (Interoperability)
  Η ικανότητα του λογισμικού να αλληλεπιδρά με ένα ή περισσότερα συστήματα και περιβάλλοντα.
- Ασφάλεια (Safety)
  Η δυνατότητα του λογισμικού να αποτρέψει την ακούσια πρόσβαση και να αντιστέκεται σε σκόπιμες επιθέσεις που αποσκοπούν στη μη εξουσιοδοτημένη πρόσβαση σε εμπιστευτικές πληροφορίες ή στις μη εξουσιοδοτημένες τροποποιήσεις πληροφοριών στο πρόγραμμα.
- Ωριμότητα (Maturity)
  Η ικανότητα του λογισμικού να αποφεύγει την αποτυχία λόγω κάποιων ελαττωμάτων που πιθανόν να παρουσιαστούν σε αυτό.
- Αντοχή στο σφάλμα (Fault Tolerance)
  Η ικανότητα του λογισμικού να διατηρεί ένα καθορισμένο επίπεδο της απόδοσης σε περίπτωση βλάβης του λογισμικού ή παραβίασης των προδιαγραφών διεπαφής του.
- Ανάκτηση (Recoverability)
  Η δυνατότητα του λογισμικού να αποκαταστήσει το επίπεδο απόδοσής του και να ανακτήσει τα άμεσα επηρεαζόμενα δεδομένα σε περίπτωση αποτυχίας.
- Κατανοησιμότητα (Understability)
  Η δυνατότητα του προϊόντος λογισμικού να επιτρέπει στον χρήστη να κατανοεί εαν το λογισμικό είναι κατάλληλο και πώς μπορεί να χρησιμοποιηθεί για συγκεκριμένα καθήκοντα υπό τις κατάλληλες συνθήκες χρήσης.
- Μάθηση (Learnability)
  Η ικανότητα του προϊόντος λογισμικού να επιτρέπει στο χρήστη να μάθει τις λειτουργίες του.
- Λειτουργικότητα (Operability)
  Η δυνατότητα του προϊόντος λογισμικού να επιτρέπει στον χρήστη να το λειτουργεί και να το ελέγχει.
- Ελκυστικότητα (Attractiveness)
  Η ικανότητα του προϊόντος λογισμικού να προτιμάται από τον χρήστη, ανάμεσα σε άλλα παρόμοια.
- Χρονική Συμπεριφορά (Time behavior)
  Η ευελιξία του λογισμικού να παρέχει κατάλληλες απαντήσεις, να εκτελείται σύντομα και να έχει τα καλύτερα ποσοστά απόδοσης, εξοικονομώντας χρόνο.
- Χρήση πόρων (Resource Utilization)
  Η ικανότητα του λογισμικού να παρέχει τις κατάλληλες πηγές, την κατάλληλη στιγμή, όταν εκτελεί τις λειτουργίες του, υπό καθορισμένες συνθήκες.
- Αναλυσιμότητα (Analyzability)
  Η ικανότητα του προϊόντος λογισμικού να διαγνωστεί για ελλείψεις ή αιτίες αποτυχιών ή για τα εξαρτήματα που πρόκειται να χρησιμοποιηθούν τροποποιηθεί για να ταυτοποιηθεί.
- Μεταβλητότητα (Changeability)
  Η δυνατότητα του προϊόντος λογισμικού να ενεργοποιήσει μια καθορισμένη τροποποίηση που πρέπει να εφαρμοστεί.
- Σταθερότητα (Stability)
  Η ικανότητα του λογισμικού να ελαχιστοποιεί τις απροσδόκητες επιπτώσεις από τροποποιήσεις που πιθανόν συμβούν σε αυτό.
- Ελεγξιμότητα (Testability)
  Η δυνατότητα του λογισμικού να επιτρέπει την τροποποίησή του, προκειμένου να επικυρωθεί η εύρυθμη λειτουργία του.
- Προσαρμοστικότητα (Adaptability)
  Η ικανότητα του λογισμικού να τροποποιηθεί για να εφαρμοστεί σε περιβάλλον διαφορετικών προδιαγραφών.
- Δυνατότητα εγκατάστασης (Installability)
  Η ικανότητα του λογισμικού να εγκατασταθεί σε ένα καθορισμένο περιβάλλον.
- Συνύπαρξη (Coexistence)
  Η ικανότητα του λογισμικού να συνυπάρχει με διαφορετικές, ανεξάρτητες πηγές λογισμικού σε ένα κοινό περιβάλλον που μοιράζεται κοινούς πόρους.
- Δυνατότητα Αντικατάστασης (Replaceability)

Η ευελιξία του λογισμικού να χρησιμοποιείται στην θέση άλλων συγκεκριμένων λογισμικών, στα δικά τους περιβάλλοντα εφαρμογής.

Οι οργανισμοί θα πρέπει να ορίζουν τα δικά τους ποιοτικά χαρακτηριστικά και υπο-χαρακτηριστικά έπειτα από διεξοδική ανάλυση των δικών τους αναγκών. Με άλλα λόγια, οι εταιρείες θα πρέπει να προσδιορίζουν το επίπεδο των διάφορων ποιοτικών χαρακτηριστικών που πρέπει να πληρούν στο πλαίσιο της ανάπτυξης του δικού τους λογισμικού. Προκειμένου να φθάσουν σε ένα ιδανικό επίπεδο ποιότητας, ξεκινώντας από το μηδέν, οι εταιρείες θα πρέπει να κατανοήσουν πως θα περάσουν από δύσκολα και επίπονα στάδια. Το αποτέλεσμα όμως, θα τις δικαιώσει. Το πρότυπο ISO 9126 είναι αυτή την στιγμή, το πιο επιδραστικό στην κοινότητα των μηχανικών λογισμικού, οι οποίοι το προσαρμόζουν στις δικές τους ανάγκες και τα δικά τους περιβάλλοντα εφαρμογών[20].



Εικόνα 3: ISO 9126 μοντέλο για εσωτερική και εξωτερική ποιότητα



Εικόνα 4: ISO 9126 μοντέλο ποιότητας κατά την χρήση

3.2 ΜΕΤΡΙΚΕΣ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ

Ένα λογισμικό αξιολογείται έτσι ώστε:

- Να εκτιμηθεί η ποιότητα του τρέχοντος προϊόντος ή διαδικασίας
- Να προβλεφθεί η ποιότητα των μελλοντικών προϊόντων
- Να βελτιωθεί η ποιότητα των διαδικασιών παραγωγής λογισμικού
- Να καθοριστεί η κατάσταση του έργου σε σχέση με τον προϋπολογισμό και το χρονοδιάγραμμα

Η αξιολόγηση του λογισμικού είναι ένα από τα πιο βασικά συστατικά της διαδικασίας παραγωγής του. Πολλοί από τους καλύτερους προγραμματιστές αξιολογούν τα χαρακτηριστικά του λογισμικού που οι ίδιοι παρήγαγαν, προκειμένου να σχημτίσουν μια πλήρη και αντικειμενική άποψη για αυτά. Οι διαχειριστές έργων λογισμικού αναλύουν και βαθμολογούν τα χαρακτηριστικά των διαδικασιών και προϊόντων προκειμένου να αποφανθούν εάν το λογισμικό είναι έτοιμο να διατεθεί για παράδοση. Οι οργανισμοί χρησιμοποιούν διαδικασίες αξιολόγησης της διαδικασίας για να επιλέξουν το τελικό λογισμικό που θα αγοράσουν από ένα σύνολο προμηθευτών. Οι ενημερωμένοι πελάτες κρίνουν τις πτυχές και τα χαρακτηριστικά του τελικού προϊόντος για να διαπιστώσουν αν ανταποκρίνονται στις απαιτήσεις που έχουν. Επίσης, οι διαχειριστές πρέπει να είναι σε θέση να αξιολογούν το τελικό προϊόν έτσι ώστε να διαπιστώσουν το θα πρέπει να βελτιώσουν ή να αναβαθμίσουν[34]. Οι μετρήσεις ποιότητας, συνεπώς, χαρακτηρίζονται ως ένα αναπόσπαστο κομμάτι της διαδικασίας παραγωγής οποιουδήποτε προϊόντος.

Μέτρηση (Measurement) είναι:
Η διαδικασία κατά την οποία αριθμοί ή σύμβολα αντιστοιχούνται σε χαρακτηριστικά οντοτήτων του πραγματικού κόσμου, με τέτοιο τρόπο έτσι ώστε να περιγράφονται σύμφωνα με καθορισμένους κανόνες.
Norman Fenton

Για την επιστήμη των υπολογιστών, οι οντότητες που περιγράφουν την ποιότητα των στοιχείων ενός λογισμικού, λέγονται μετρικές. Οι μετρικές ποιότητας λογισμικού έχουν γίνει αναπόσπαστο κομμάτι της παραγωγής λογισμικού. Οι τιμές τους κρίνουν την τελική απόδοση του προϊόντος και την προώθηση αυτού[35]. Κατά συνέπεια, πλήθος μετρικών έχουν προταθεί κατά καιρούς, όπως επίσης και πληθώρα εργαλείων υπολογισμού αυτών. Λαμβάνοντας υπόψιν τις διαφορετικές σκοπιές απ΄τις οποίες βλέπουν οι διαφορετικοί άνθρωποι που συμμετέχουν στον σχεδιασμό και στην ανάπτυξη ενός λογισμικού την ποιότητα, είναι κατανοητό γιατί η ποιότητα αξιολογείται σε διαφορετικά επίπεδο λεπτομέρειας από το κάθε ξεχωριστό τμήμα. Ο πρακτικός υπολογισμός της ποιότητας αποτελεί μια πρόκληση εξαιτίας i)της ανάγκης συνδυασμού διαφορετικών μετρικών, όπως συνιστάται από το μοντέλο ποιότητας παραγόντων FCM και ii) λόγω της ανάγκης απόκτησης γνώσεων σχετικών με την ποιότητα ενός ολόκληρου συστήματος, λόγου χάρη τις τιμές που λαμβάνονται για τις μετρικές αξιολόγησης στοιχείων χαμηλότερου επιπέδου. Η ποιότητα λογισμικού μπορεί να υπολογιστεί:

1. Εσωτερικά με στατικές μετρήσεις του κώδικα
2. Εξωτερικά, παρατηρώντας και καταγράφοντας την συμπεριφορά του κώδικα όταν εκτελείται

Για παράδειγμα, η αξιοπιστία μπορεί να μετρηθεί εξωτερικά παρατηρώντας το πλήθος των λαθών σε ένα δεδομένα χρονικό διάστημα εκτέλεσης, κατά τη διάρκεια ενός δοκιμαστικού ελέγχου του λογισμικού. Από την άλλη, η ποιότητα υπολογίζεται εσωτερικά, όταν για παράδειγμα επιθεωρούνται οι λεπτομερείς προδιαγραφές και οι πηγές του κώδικα.

Σύμφωνα με τον Gilb[36] «οποιοδήποτε χαρακτηριστικό μπορεί να μετρηθεί κατά κάποιο τρόπο· αυτός ο τρόπος θα είναι ανώτερος από το να μην μετρηθεί καθόλου το συγκεκριμένο χαρακτηριστικό». Οι μετρικές ποιότητας λογισμικού εκτιμούν την επιτυχία μια διαδικασίας παραγωγής λογισμικού. Θεωρητικά, οι μετρικές

μπορούν να βοηθήσουν στη βελτίωση της διαδικασίας ανάπτυξης και παρέχουν στις εταιρίες τις κατάλληλες πληροφορίες προκειμένου τα μελλοντικά τους έργα να είναι πιο ακριβή, πιο αποδοτικά κ.τ.π.

3.2.1 ΤΙ ΕΙΝΑΙ ΜΙΑ ΜΕΤΡΙΚΗ ΠΟΙΟΤΗΤΑΣ

Μετρική(metric) είναι:
Μια εμπειρική αντικειμενική αντιστοίχηση ενός αριθμού σε μια οντότητα με στόχο να χαρακτηρίσει ένα συγκεκριμένο χαρακτηριστικό της οντότητας αυτής[34].

Μια μετρική είναι μια εκτίμηση του βαθμού σύμφωνα με το οποίο ένα χαρακτηριστικό ανήκει σε ένα σύστημα, ένα προϊόν ή μια διαδικασία. Για παράδειγμα, το πλήθος των λαθών σε έναν κώδικα, είναι μια μετρική. Οι μετρικές λογισμικού ταξινομούνται σε τρεις κατηγορίες[37]:

- Μετρικές για τα προϊόντα (Product metrics)
- Μετρικές εκτίμησης διεργασιών (Process metrics)
- Μετρικές έργων (Project metrics)

Οι μετρικές για τα προϊόντα περιγράφουν τα χαρακτηριστικά των προϊόντων όπως το μέγεθος, η πολυπλοκότητα, τα σχεδιαστικά χαρακτηριστικά, η απόδοση και το επίπεδο ποιότητας. Οι μετρικές εκτίμησης διεργασιών μπορούν να χρησιμοποιηθούν για την βελτίωση της διαδικασίας ανάπτυξης και συντήρησης λογισμικού. Παραδείγματα μετρικών εκτίμησης διεργασιών είναι η προσπάθεια αφαίρεσης των λαθών κατά την ανάπτυξη, το πόσο γρήγορα εντοπίζεται ένα λάθος κατά τον έλεγχο και ο χρόνος που απαιτείται για τη διόρθωση του. Οι μετρικές έργων περιγράφουν τα χαρακτηριστικά του έργου και την εκτέλεσή του. Τέτοιες μετρικές είναι για παράδειγμα, ο αριθμός των προγραμματιστών, το σύνολο των υπαλλήλων που ασχολήθηκαν με το λογισμικό κατά τη διάρκεια του κύκλου ζωής του, το κόστος, η απόκλιση από το χρονοδιάγραμμα και η παραγωγικότητα της ομάδας ανάπτυξης. Ορισμένες μετρικές ανήκουν σε παραπάνω από μια κατηγορία. Οι μετρικές ποιότητας λογισμικού είναι ένα υποσύνολο των μετρικών λογισμικού που εστιάζει στις πτυχές ποιότητας ενός προϊόντος, μιας διαδικασίας, ενός έργου. Γενικά, οι μετρικές ποιότητας λογισμικού, σχετίζονται περισσότερο με το τελικό προϊόν και τη διαδικασία ανάπτυξής του παρά με την αξιολόγηση του συνολικού έργου. Παραταύτα, οι παράγοντες που επηρεάζουν το έργο, όπως το πλήθος των προγραμματιστών, τα επίπεδα δεξιοτήτων τους, το χρονοδιάγραμμα, το μέγεθος και η δομή του οργανισμού, ασφαλώς επηρεάζουν την ποιότητα του τελικού προϊόντος. Οι μετρικές ποιότητας λογισμικού μπορούν να διαχωριστούν περαιτέρω σε μετρικές ποιότητας τελικού προϊόντος και μετρικές ποιότητας κατά τη διάρκεια της διαδικασίας ανάπτυξης [38]. Η ουσία της επιστήμης εκτίμησης ποιότητας λογισμικού είναι η διερεύνηση των σχέσεων μεταξύ των εσωτερικών μετρικών, των χαρακτηριστικών του έργου και της ποιότητας του τελικού προϊόντος. Επιπλέον, θα πρέπει να αξιολογούμε την ποιότητας λαμβάνοντας υπόψιν κάθε κομμάτι του συνολικό κύκλου ζωής ενός λογισμικού. Έτσι προκύπτει μια νέα κατηγορία μετρικών, εκείνη που εκτιμά το επίπεδο ποιότητας της διαδικασίας συντήρησης του λογισμικού. Στον Πίνακας 3, παρακάτω, φαίνονται οι βασικές μετρικές που χρησιμοποιούνται για την αξιολόγηση των ποιοτικών χαρακτηριστικών σύμφωνα με το πρότυπο της IEEE(1988,1996) [5]. Φυσικά μια εταιρεία μπορεί να μην χρησιμοποιεί όλη τη λίστα ή να υπολογίζει τα χαρακτηριστικά βάσει δικών της μετρικών. Άλλες εταιρείες, υιοθετούν δικές τους τεχνικές εκτίμησης. Στον παραπάνω πίνακα, ωστόσο φαίνονται οι μετρικές που χτίζουν την βάση για κάθε εταιρεία, έτσι ώστε να βελτιώσει την ποιότητα, να μειώσει το κόστος και εν τέλει να αναπτύξει τα δικά της λογισμικά εκτίμησης ποιότητας λογισμικού.

Πίνακας 3: Προτεινόμενες μετρικές από την IEEE

Πυκνότητα Σφάλματος (Fault density)	**Επίπεδο καθαρότητας λογισμικού** (Software purity level)
Πυκνότητα Ελαττωμάτων (Defect density)	**Εκτιμώμενος αριθμός ελαττωμάτων** (Estimated number of faults remaining)
Συγκεντρωτικό προφίλ αποτυχίας (Cumulative failure profile)	**Προϋποθέσεις συμμόρφωσης** (Requirements compliance)
Αριθμός ημερών βλάβης (Fault-days number)	**Δοκιμή κάλυψης** (Test coverage)

Λειτουργική ή αρθρωτή κάλυψη δοκιμών (Functional or modular test coverage)	**Πολυπλοκότητα δεδομένων ή ροής πληροφοριών** (Data or information flow complexity)
Σχέση αιτίου και αποτελέσματος (Cause and effect graphing)	**Λειτουργία αύξησης αξιοπιστίας** (Reliability growth function)
Ιχνηλασιμότητα απαιτήσεων (Requirements traceability)	**Υπολογισμός υπολειπόμενης βλάβης** (Residual fault count)
Δείκτες ανεπάρκειας (Defect indices)	**Χρόνος ανάλυσης αποτυχιών** (Failure analysis elapsed time)
Κατανομή σφαλμάτων (Error distribution(s))	**Ορισμός κατάλληλων δοκιμών** (Testing sufficiently)
Δείκτης ωριμότητας λογισμικού (Software maturity index)	**Μέσος χρόνος έως την εμφάνιση αποτυχίας** (Mean time to failure)
Προσωπικές ώρες ενασχόλησης για τον εντοπισμό μείζονος ελαττώματος (Person-hours per major defect detected)	**Ποσοστό αποτυχίας** (Failure rate)
Αριθμός αντιφατικών απαιτήσεων (Number of conflicting requirements)	**Τεκμηρίωση λογισμικού και καταχώρηση πηγής** (Software documentation and source listing)
Αριθμός εισόδων και εξόδων ανα ενότητα (Number of entries and exits per module)	**Αξιοπιστία του απαιτούμενο λογισμικού** (Rely-required software reliability)
Εκτίμηση της σχέσης λογισμικού και κανόνων επιστήμης (Software science measures)	**Ετοιμότητα διάθεσης λογισμικού στο ευρύ κοινό** (Software release readiness)
Θεωρητικό γράφημα για την πολυπλοκότητα της αρχιτεκτονικής (Graph-theoretic complexity for architecture)	**Πληρότητα** (Completeness)
Κυκλική πολυπλοκότητα (Cyclomatic complexity)	**Ακρίβεια δοκιμής** (Test accuracy)
Καθορισμός ελάχιστων περιπτώσεων δοκιμών μονάδων (Minimal unit test case determination)	**Αξιοπιστία απόδοσης συστήματος** (System performance reliability)
Εκτέλεση αξιοπιστίας (Run reliability)	**Ανεξάρτητη διαδικασία εκτίμησης αξιοπιστίας** (Independent process reliability)
Δομή σχεδιασμού (Design structure)	**Διαθεσιμότητα υλικού και λογισμικού συστήματος** (Combined hardware and software (system) availability)
Μέσος χρόνος εντοπισμού επόμενων Κ-σφαλμάτων (Mean time to discover the next K-faults)	

3.2.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΑΠΟΔΟΤΙΚΩΝ ΜΕΤΡΙΚΩΝ ΠΟΙΟΤΗΤΑΣ

Πριν αναλυθεί το κατά πόσο μια μετρική χαρακτηρίζεται καλή, θα πρέπει να σημειωθεί εδώ, πότε μια διαδικασία μέτρησης χαρακτηρίζεται επιτυχημένη. Μια διαδικασία μέτρησης χαρακτηριστικών ποιότητας επιβάλλεται να ακολουθεί πέντε βασικές αρχές[39]:

- Διατύπωση
  Η εξαγωγή μετρήσεων και μετρικών λογισμικού θα πρέπει να είναι κατάλληλη, έτσι ώστε να συμπεριληφθεί στην αναπαράσταση του λογισμικού που εξετάζεται.
- Συλλογή
  Πρόκειται για τον μηχανισμό που απαιτείται για την ακριβή καταγραφή των τιμών που παίρνουν οι μετρικές.
- Ανάλυση
  Ο υπολογισμός των μετρικών και η εφαρμογή των μαθηματικών εργαλείων.
- Ερμηνεία
  Η αξιολόγηση των μετρικών οδηγεί σε κατανόηση της ποιότητας της παρουσίασης.
- Ανατροφοδότηση
  Οι συστάσεις που προκύπτουν από την ερμηνεία των μετρικών αξιολόγησης του προϊόντος μεταδίδονται στην ομάδα ανάπτυξης λογισμικού.

Οι μετρικές λογισμικού θα είναι χρήσιμες και θα αποδεικνύουν την αξία τους μόνο εάν μπορούν να χαρακτηριστούν έγκυρες και αποτελεσματικές. Οι παρακάτω αρχές αποτελούν πρότυπο για τις μετρικές:

- Μια μετρική πρέπει να έχει τις επιθυμητές μαθηματικές ιδιότητες. Δηλαδή, το πεδίο ορισμού της τιμής μιας μετρικής θα πρέπει να είναι ορισμένο.
- Όταν μια μετρική αντιπροσωπεύει ένα χαρακτηριστικό λογισμικού το οποίο αυξάνεται όταν φθάνει τις επιθυμητές τιμές ή μειώνεται σε αντίθετη περίπτωση, η τιμή τής θα πρέπει να αυξομειώνεται με τον ανάλογο τρόπο.
- Κάθε μετρική, θα πρέπει να επικυρώνεται εμπειρικά πριν δημοσιευθεί ή χρησιμοποιηθεί για την λήψη αποφάσεων.
- Μια μετρική θα πρέπει να λαμβάνει υπόψιν τον παράγοντα ενδιαφέροντος, ανεξάρτητα από τους άλλους παράγοντες.
- Θα πρέπει να είναι ευέλικτη. Δηλαδή θα πρέπει να κλιμακώνεται για μεγάλα συστήματα και να λειτουργεί σωστά για πλήθος γλωσσών προγραμματισμού

Παρόλο που η διατύπωση, ο χαρακτηρισμός και η επικύρωση είναι κρίσιμες ιδιότητες για μια μετρική, η συλλογή και η ανάλυση είναι δραστηριότητες που οδηγούν στη διαδικασία μέτρησης. Οι βασικές αρχές για τις παραπάνω δραστηριότητες είναι 1) όταν είναι δυνατό, η συλλογή και η ανάλυση δεδομένων θα πρέπει να αυτοματοποιούνται, 2) πρέπει να εφαρμόζονται έγκυρες στατιστικές τεχνικές για να υπάρξει επιτυχημένη συσχέτιση μεταξύ των εσωτερικών ιδιοτήτων του προϊόντος και των εξωτερικών ποιοτικών χαρακτηριστικών και 3) θα πρέπει να θεσπιστούν ερμηνευτικές οδηγίες και συστάσεις για κάθε μετρική[40]. Εκατοντάδες μετρικές έχουν προταθεί για το λογισμικό, αλλά δεν προσφέρουν όλες πρακτική βοήθεια σε έναν μηχανικό υπολογιστών. Κάποιες μετρικές είναι πολύπλοκες, άλλες είναι τόσο εξιδεικευμένες που μόνο λίγοι έμπειροι επαγγελματίες μηχανικοί μπορούν να τις καταλάβουν, ενώ άλλες παραβιάζουν τις βασικές διαισθητικές έννοιες για το τι πραγματικά είναι ένα λογισμικό υψηλής ποιότητας.

Παρακάτω παρουσιάζονται τα χαρακτηριστικά που όλες οι μετρικές θα πρέπει να έχουν, προκειμένου να χαρακτηριστούν αποδοτικές[41]:

- Απλότητα και αξιοπιστία
  Η εκμάθηση και η εφαρμογή των μετρικών θα πρέπει να είναι απλές, γρήγορες και εύκολες διαδικασίες.
- Πειστικότητα
  Οι μετρικές θα πρέπει να ικανοποιούν τη διαίσθηση του μηχανικού σχετικά με τα αποτελέσματα που αναμένει να αντικρίσει από την αξιολόγηση του υπο-εξέταση χαρακτηριστικού.
- Συνέπεια και αντικειμενικότητα
  Τα αποτελέσματα των μετρικών θα πρέπει να είναι ξεκάθαρα και αναπαραγώγιμα. Ένα ανεξάρτητο τρίτο εργαλείο αξιολόγησης, θα πρέπει να αντλεί παρόμοιες τιμές με τις μετρικές.
- Συνέπεια μονάδων ή διαστάσεων
  Οι μαθηματικοί υπολογισμοί των μετρικών, επιβάλλεται να οδηγούν σε αποτελέσματα που είναι λογικά και δεν οδηγούν σε παράξενους συνδυασμούς ανεξάρτητων μονάδων.
- Ανεξάρτητη γλώσσα προγραμματισμού
  Οι μετρικές θα πρέπει να βασίζονται στις απαιτήσεις του μοντέλου σχεδιασμού ή στη δομή του ίδιου του προγράμματος. Είναι αναγκαίο να είναι ανεξάρτητες των ιδιοτροπιών της εκάστοτε γλώσσας προγραμματισμού και της σημασιολογίας.

- Δυνατότητα ανατροφοδότησης
  Τα αποτελέσματα των μετρικών θα πρέπει να παρέχουν κατανοητές και χρήσιμες πληροφορίες στον μηχανικό ανάπτυξης του λογισμικού, έτσι ώστε να είναι σε θέση να αναπτύξει στο μέλλον προϊόντα υψηλότερης ποιότητας.

3.3 ΛΑΘΗ ΚΑΙ ΕΚΔΟΣΕΙΣ ΛΟΓΙΣΜΙΚΟΥ

Η διαδικασία αξιολόγησης λογισμικού είναι ένα φίλτρο για τις διεργασίες ανάπτυξης του λογισμικού. Οι έλεγχοι που εφαρμόζονται κατά τη διάρκεια του κύκλου ζωής του λογισμικού χρησιμεύουν στον εντοπισμό σφαλμάτων και ελαττωμάτων που μπορούν μετέπειτα να αφαιρεθούν. Τα λογισμικά επιθεώρησης και ελέγχων είναι αναπόσπαστο κομμάτι της διαδικασίας ελέγχου καθώς μερικές κατηγορίες σφαλμάτων ξεφεύγουν της ανθρώπινης παρατηρητικότητας. Ο ρόλος που διαδραματίζει ο έλεγχος κατά τη διάρκεια σχεδίασης και ανάπτυξης του λογισμικού είναι πολυδιάστατος. Επισημαίνει τις απαραίτητες βελτιώσεις, επιβεβαιώνει την μη αναγκαιότητα εφαρμογής διορθώσεων όταν αυτές δεν είναι απαραίτητες και επιτυγχάνει την βέλτιστη ποιότητα των τεχνικών εργασιών. Μια τεχνική ανάλυσης της ποιότητας των στοιχείων ενός συστήματος λογισμικού αποτελεί το αποτελεσματικότερο μέσο για την ανίχνευση ατελειών και συνεπώς την βελτίωση της ποιότητας του λογισμικού. Οι ατέλειες που ενδεχομένως εντοπιστούν σε ένα σύστημα λογισμικού μπορεί να είναι:

1. Αποτυχία (failure)
2. Σφάλμα (error)
3. Βλάβη (fault)
4. Ελάττωμα (defect)

Αν και οι έννοιες των όρων αποτυχία (failure), σφάλμα (error), βλάβη (fault) και ελάττωμα (defect) σχετίζονται μεταξύ τους, διαφέρουν αρκετά. Παρακάτω, παρουσιάζονται οι πρώτες τρεις έννοιες:

- Αποτυχία παρουσιάζεται όταν η εξωτερική συμπεριφορά ενός συστήματος δεν συμμορφώνεται με τις προκαθορισμένες προδιαγραφές του.
- Το σφάλμα είναι μια κατάσταση του συστήματος. Ελλείψει διορθωτικών ενεργειών, μια κατάσταση σφάλματος θα μπορούσε να οδηγήσει σε μια αποτυχία.
- Βλάβη είναι η κρίσιμη αιτία σφάλματος.

Μια βλάβη ενδεχομένως να παραμείνει για μεγάλο χρονικό διάστημα μη ανιχνεύσιμη, έως ότου ενεργοποιηθεί από κάποιο συμβάν. Όταν αυτό συμβεί, το πρόγραμμα αρχικά βρίσκεται σε ενδιάμεση κατάσταση σφάλματος. Εάν ο υπολογισμός μπορεί να συνεχιστεί, παρά την τρέχουσα κατάσταση σφάλματος, το πρόγραμμα τελικώς οδηγείται σε αποτυχία[20]. Υπάρχουν υπολογιστές ανεκτικοί σε σφάλματα, οι οποίοι λαμβάνουν προληπτικά μέτρα έτσι ώστε μια κατάσταση σφάλματος να μην οδηγήσει το σύστημα σε αποτυχία. Η διαδικασία που πιθανών να οδηγήσει σε αποτυχία αποτελείται από τρία μέρη που συνδέονται σαν αλυσίδα: Βλάβη → Σφάλμα → Αποτυχία[42]. Η παραπάνω αλυσίδα μπορεί να επαναληφθεί, δηλαδή η αποτυχία ενός στοιχείου του συστήματος, μπορεί να οδηγήσει στην αποτυχία ενός άλλου. Είναι δύσκολο να δοθούν ακριβείς ορισμοί στις έννοιες αποτυχία, σφάλμα και βλάβη λόγω του ανθρώπινου παράγοντα που σχετίζεται με την γενικότερη αποδοχή ενός συστήματος. Σύμφωνα με τον Ram Chillarege[43], η σύγχρονη προσέγγιση θέλει την αποτυχία λογισμικού, ως μια έννοια που επηρεάζεται από έναν μοναδικό παράγοντα: την ικανοποίηση των προσδοκιών του πελάτη, δηλαδή εάν το προϊόν είναι χρήσιμο στον τελικό χρήστη, τότε δεν μπορεί να χαρακτηριστεί αποτυχημένο. Το ελάττωμα (defect) είναι μια κατάσταση στην οποία μπορεί να έλθει ένα λογισμικό όταν α) δεν πληρεί τις προδιαγραφές και τις απαιτήσεις σχεδιασμού και ανάπτυξης και β) όταν δεν ανταποκρίνεται στις προσδοκίες του τελικού χρήστη. Με άλλα λόγια, ένα ελάττωμα μπορεί να είναι ένα σφάλμα του κώδικα, μία λογική που προκαλεί δυσλειτουργία του προγράμματος ή παράγει λανθασμένα και απροσδόκητα αποτελέσματα. Οι μηχανικοί ελέγχου λογισμικού είναι υποχρεωμένοι να εντοπίζουν και να καταγράφουν τα παραπάνω ελαττώματα νωρίς και παράλληλα να επιστρέφουν μια αναφορά ελαττωμάτων (Defect Report) στους μηχανικούς ανάπτυξης λογισμικού, όπου θα περιγράφουν λεπτομερώς τα σφάλματα που ανακάλυψαν. Οι μηχανικοί ελέγχου χρησιμοποιούν ειδικά λογισμικά εντοπισμού ελαττωμάτων, ενώ η διαδικασία εύρεσής τους, είναι γνωστή ως αποσφαλμάτωση (debugging). Η διαδικασία εκούσιας εισαγωγής σφαλμάτων στο πρόγραμμα, προκειμένου να αξιολογηθεί η ταχύτητα και η επιτυχία των εφαρμογών εντοπισμού αυτών, είναι συνήθης τακτική από τις ομάδες εξασφάλισης ποιότητας λογισμικού.

Στο σημείο αυτό θα πρέπει να διαχωριστούν με σαφήνεια οι έννοιες σφάλμα και ελάττωμα. Το σφάλμα είναι ένα πρόβλημα του λογισμικού που εντοπίζεται πριν αυτό διατεθεί στους τελικούς χρήστες. Το ελάττωμα είναι ένα πρόβλημα στο λογισμικό που εντοπίζεται αφού αυτό διατεθεί στους τελικούς χρήστες[39]. Επίσης ελάττωμα κατά τη διαδικασία βελτίωσης, υπάρχει όταν εντοπίζεται ένα πρόβλημα στην ποιότητα κατά τη διάρκεια εκτέλεσης συγκεκριμένης εργασίας, ωστόσο αυτό μεταφέρεται ανέγγικτο στην επόμενη διεργασία της παραγωγικής ροής. Ο διαχωρισμός της σημασίας του σφάλματος και του ελαττώματος είναι απαραίτητος. Τα δύο αυτά είδη ατελειών λογισμικού επιδρούν διαφορετικά στην οικονομία, στην επιχείρηση, στην ψυχολογία

και στον άνθρωπο. Οι μηχανικοί υπολογιστών έχουν ως στόχο την ανακάλυψη και τη διόρθωση, όσων περισσοτέρων σφαλμάτων και ελαττωμάτων γίνεται, πριν τα εντοπίσει ο τελικός χρήστης. Η χρονική στιγμή εντοπισμού του προβλήματος είναι ίσως από τα πιο σημαντικά ζητήματα που έχουν να αντιμετωπίσουν οι μηχανικοί ανάπτυξης και ελέγχου λογισμικού.

3.3.1 ΔΙΟΡΘΩΣΗ ΕΛΛΑΤΩΜΑΤΩΝ ΠΡΙΝ ΚΑΙ ΜΕΤΑ ΤΗΝ ΕΚΔΟΣΗ ΛΟΓΙΣΜΙΚΟΥ

Οι μηχανικοί εξασφάλισης ποιότητας προσφέρουν εναλλακτικές λύσεις για να είναι αποτελεσματική μια συντονισμένη προσπάθεια αντιμετώπισης σφαλμάτων, ελαττωμάτων, ή αποτυχιών, προκειμένου να επιτευχθεί ο κοινός στόχος διασφάλισης και βελτίωσης της ποιότητας. Οι δραστηριότητες πρόληψης και μείωσης ελαττωμάτων ασχολούνται με τις ανταγωνιστικές διαδικασίες ακούσιας και εκούσιας προσθήκης ελαττωμάτων κατά τη διάρκεια της διαδικασίας ανάπτυξης λογισμικού[44]. Επηρεάζουν τα περιεχόμενα των ελαττωμάτων, ή τον αριθμό σφαλμάτων στα τελικά προϊόντα λογισμικού, προσπαθώντας να μειώσουν τα ελαττώματα πριν την έκδοσή του ή να αφαιρέσουν όσα ελαττώματα είναι πιθανό να συμβούν μετά την έκδοσή τους. Τα ελαττώματα που παραμένουν, παρά την προσπάθεια, μετά τη διάθεση του λογισμικού στην αγορά, λέγονται «αδρανή ελαττώματα», επειδή μπορούν να παραμένουν κρυφά και αδρανή για κάποιο χρονικό διάστημα, αλλά ενδεχομένως να προκαλέσουν στο μέλλον προβλήματα στους χρήστες[22].

Όταν τα ελαττώματα εντοπίζονται νωρίς στη διαδικασία ανάπτυξης λογισμικού, λίγα από αυτά θα παραμείνουν στο τελικό προϊόν. Ταυτόχρονα, αν έχει γίνει σωστή πρόληψη ελαττωμάτων, πριν την έκδοση, το κόστος διόρθωσης των σφαλμάτων που εντοπίζονται νωρίτερα είναι πολύ μικρότερο, από εκείνο που θα δαπανήσει μια εταιρεία να διορθώσει λάθη που εντοπίζει ο τελικός χρήστης[45]. Ένας σημαντικός λόγος για τον οποίο γίνεται ακριβότερη η διόρθωση στα τελικά στάδια της ανάπτυξής είναι πως ο κώδικας που περιέχει ένα σφάλμα εμπλουτίζεται καθημερινά, γίνεται ακόμη πιο σύνθετος και δυσανάγνωστος. Συνεπώς, η διαδικασία αποσύνθεσης του κώδικα, για τον εντοπισμό του λάθους που προκαλεί αποτυχία, γίνεται ολοένα και πιο δαπανηρή, ειδικά εάν κώδικας βρίσκεται σε τελικό στάδιο. Τα χρήματα που δαπανά μια εταιρεία προκειμένου να εντοπίσει λάθη νωρίτερα, είναι και τα πιο αποδοτικά. Η διόρθωση ελαττωμάτων σε μεμονωμένα στοιχεία του προϊόντος, πριν αυτό διατεθεί ολοκληρωμένο στην αγορά, στοιχίζει πολύ λιγότερο από το να ανακαλυφθούν τα συγκεκριμένα λάθη από τον πελάτη, στο τελικό ολοκληρωμένο προϊόν. Ένα ελάττωμα που εντοπίζεται μετά την έκδοση του προϊόντος δεν αντανακλά μόνο μια αδυναμία στις διαδικασίες δοκιμής, αλλά και μια μη-οργανωμένη και μη-αποδοτική παραγωγή. Γι' αυτό λοιπόν, είναι πολύ σημαντικό για τις εταιρείες να γνωρίζουν τον ακριβή αριθμό των λαθών που εντοπίστηκαν μετά την έκδοση, το επίπεδο που επηρεάζουν την συνολική λειτουργία του προϊόντος, για να αποφαίνονται τελικώς με σιγουριά πως και ποια χρονική στιγμή, κατά τη διάρκεια παραγωγής, θα μπορούσαν να είχαν αποφευχθεί[18]. Συνεπώς, αμέσως μόλις ανακαλυφθεί ένα ελάττωμα, το επόμενο βήμα να εντοπιστεί ο χρόνος και ο τρόπος εμφάνισης του. Αυτό δεν συμβαίνει για να αποδοθούν ευθύνες, αλλά για να επιτευχθεί το προφανές ζητούμενο: η συνεχής βελτίωση της διαδικασίας. Εάν δεν κατανοηθεί πλήρως γιατί γεννήθηκε ένα πρόβλημα, θα επαναληφθεί με μαθηματική ακρίβεια και στις επόμενες εκδόσεις του λογισμικού. Πέρα από το υψηλό κόστος διόρθωσης ενός σφάλματος το οποίο εντοπίζεται από τους πελάτες μετά την έκδοση, η καταστροφή της φήμης του πωλητή λογισμικού είναι τεράστια. Η ελεγχόμεη δοκιμή ή αλλιώς «Δοκιμή Beta (Beta testing)», προτείνεται για να αποφεύγονται αυτά τα περιστατικά.

3.4 ΧΡΗΣΤΕΣ BETA

Αν και οι στατιστικές δοκιμές που λαμβάνουν δεδομένα από τους πελάτες, μπορούν να αξιοποιηθούν με ασφάλεια έτσι ώστε να διαπιστωθεί η αξιοπιστία ενός προϊόντος, υπάρχουν διάφοροι περιορισμοί στην ακρίβεια και στην πρακτικότητα αυτών των τεχνικών. Μια εναλλακτική πρόταση είναι η ελεγχόμενη απελευθέρωση του λογισμικού, έτσι ώστε αυτό να εκτεθεί σε μικρό ποσοστό του γενικότερου πληθυσμού. Με αυτόν τον τρόπο, το προϊόν θα μπορεί να αξιολογηθεί και να διορθωθεί πριν την επίσημη έκδοσή του προς όλους. Αυτό το μοντέλο αναφέρεται ως δοκιμαστικοί έλεγχοι (beta testing). Το όφελος του ελέγχου από τους δοκιμαστικούς χρήστες (beta users) είναι μεγάλο, ειδικά για προϊόντα ευρείας κυκλοφορίας[22]. Ωστόσο το κόστος ενός τέτοιου προγράμματος δοκιμών είναι επίσης αξιοσημείωτο και οφείλεται στους εξής δύο λόγους:

- Πρέπει να βρεθεί ένα αρκετά μεγάλο και αντιπροσωπευτικό δείγμα πελατών, ενώ η σχέση μαζί τους πρέπει να καλλιεργείται για μεγάλο χρονικό διάστημα προκειμένου να εμπιστευτούν την εταιρεία παραγωγής του λογισμικού.
- Οι εταιρείες θα πρέπει να είναι προετοιμασμένες για το έμμεσο κόστος που αναπόφευκτα προκύπτει από τις καθυστερήσεις έκδοσης των προϊόντων υπό δοκιμή (beta products) σε συνδυασμό με τον χρόνο που χρειάζονται οι δοκιμαστικοί χρήστες για να εξοικειωθούν με το προϊόν.

Συνεπώς, αν μια εταιρεία επιθυμεί να χρησιμοποιήσει το πρόγραμμα δοκιμαστικού ελέγχου θα πρέπει να είναι ενήμερη για τα παραπάνω. Τα οφέλη υλοποίησης του ωστόσο είναι αρκετά καθώς η άμεση συμμετοχή πλήθους

αριθμού χρηστών, οι οποίοι αναφέρουν τα προβλήματα και κάνουν κριτική για το τελικό προϊόν, μπορεί να αποτελεί βαρυσήμαντη εγγύηση για την ποιότητα του[46].

4

ΒΑΣΙΚΕΣ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΕΙΣ ΤΟΥ ΕΛΕΓΧΟΥ ΛΟΓΙΣΜΙΚΟΥ

4.1 ΕΠΙΠΕΔΑ ΕΛΕΓΧΟΥ

Η γνώση αποκτά αξία μόνο όταν εφαρμοστεί.

Anton Chekhov

Υπάρχουν πολλές στρατηγικές που μπορούν να χρησιμοποιηθούν για τον έλεγχο λογισμικού. Μια άποψη υποστηρίζει αναμονή έως ότου ολοκληρωθεί η κατασκευή του συστήματος και έπειτα να γίνουν οι έλεγχοι στο ολοκληρωμένο σύστημα, με την ελπίδα να βρεθούν λάθη. Μια άλλη άποψη, υποστηρίζει τη διεξαγωγή δοκιμών σε καθημερινή βάση σε κάθε μέρος του συστήματος που βρίσκεται υπό κατασκευή. Η τελευταία προσέγγιση είναι και η πιο αποτελεσματική για πολλούς, καθώς είναι και η πιο αποδοτική[39]. Μια στρατηγική ελέγχου που επιλέγεται από τις περισσότερες ομάδες λογισμικού συνδυάζει τις δυο παραπάνω μεθόδους. Σύμφωνα με αυτή, οι έλεγχοι λογισμικού πραγματοποιούνται σε διαφορετικά επίπεδα που εμπλέκονται όλο το σύστημα ή τμήματά του, καθ' όλη τη διάρκεια του κύκλου ζωής ενός προϊόντος λογισμικού[20]. Ένα σύστημα λογισμικού περνά από τέσσερα στάδια δοκιμής πριν την τελική φάση ανάπτυξής του. Οι τέσσερις στρατηγικές εφαρμόζονται με την ακόλουθη σειρά:

- Έλεγχος μονάδων (Unit testing)
- Έλεγχος ολοκλήρωσης (Integration testing)
- Έλεγχος συστήματος (System testing)
- Έλεγχος επιπέδου αποδοχής (Acceptance level testing)

Τα τρία πρώτα είδη ελέγχου εκτελούνται από φορείς εμπλεκόμενους με την ανάπτυξη του λογισμικού, σε αντίθεση με το τελευταίο που εκτελείται από τους πελάτες. Και τα τέσσερα διαφορετικά είδη μοντέλων συνδέονται στενά με διαφορετικές τεχνικές δοκιμών και λεπτομέρειες μοντελοποίησης. Ωστόσο, τα παρακάτω γενικά βήματα και δραστηριότητες ακολουθούνται από κάθε μοντέλο δοκιμής[22]:

1. Ταυτοποίηση πηγής πληροφοριών και συλλογή δεδομένων
   Οι πληροφορίες και τα δεδομένα γενικά επηρεάζονται από τις απαιτήσεις κάθε μοντέλου και από το περιβάλλον κάθε έργου.
2. Ανάλυση και αρχική κατασκευή μοντέλου
   Οι πληροφορίες και τα δεδομένα που συλλέχθηκαν, αναλύονται για την κατασκευή δοκιμαστικών μοντέλων. Η εξειδίκευση και η εξοικείωση με τις συγκεκριμένες τεχνικές και μοντέλο είναι απαραίτητες για τους ανθρώπους που εκτελούν αυτό το βήμα. Το στάδιο αυτό αυτοματοποιείται δύσκολα, καθώς προϋποθέτει την συνεισφορά της ανθρώπινης νοημοσύνης και εμπειρίας.
3. Επικύρωση μοντέλου και σταδιακή βελτίωση
   Πρόκειται για ένα εξαιρετικά σημαντικό βήμα, ειδικά για μεγάλα αντικείμενα ή για λειτουργίες που σχετίζονται με εξωτερικούς πελάτες. Η επανάληψη της διαδικασίας μπορεί να είναι απαραίτητη για την διόρθωση ανακριβειών και άλλων προβλημάτων που εντοπίστηκαν στον αρχικό μοντέλο.

Εφόσον τα μοντέλα δοκιμών έχουν κατασκευαστεί και επικυρωθεί, χρησιμοποιούνται για τη δημιουργία δοκιμαστικών περιπτώσεων, οι οποίες στην συνέχεια εκτελούνται ακολουθώντας μια προκαθορισμένη διαδικασία. Στο σημείο αυτό, είναι απαραίτητο να οριστούν και να διακριθούν οι παρακάτω έννοιες:

- Στατική περίπτωση δοκιμών(test case)
  Η στατική περίπτωση δοκιμών είναι μια συλλογή που αποτελείται από οντότητες και σχετικές πληροφορίες που επιτρέπουν την εφαρμογή ενός ελέγχου.
- Δυναμική περίπτωση δοκιμών (test run)

Μια δυναμική περίπτωση δοκιμών είναι μια μονάδα συγκεκριμένων ελέγχων στην συνολική ακολουθία ελέγχων ενός αντικειμένου. Ονομάζεται επίσης απόπειρα δοκιμής.

Σε κάθε στατική περίπτωση δοκιμών αντιστοιχούν πολλαπλές δυναμικές περιπτώσεις δοκιμών. Οι πληροφορίες που περιλαμβάνονται σε μια περίπτωση δοκιμών πρέπει να ορίζουν το σημείο εκκίνησης και το σημείο ολοκλήρωσης μιας απόπειρας δοκιμής. Για τα περισσότερα είδη ελέγχων, η εκκίνηση και η ολοκλήρωση συμπίπτουν με την έναρξη και τον τερματισμό των εργασιών ολόκληρου του συστήματος. Υπάρχουν όμως εξαιρέσεις όπως τα λειτουργικά συστήματα και οι ψηφιακές τηλεπικοινωνίες, όπου η συνεχής λειτουργία χωρίς διακοπή είναι αναμενόμενο γεγονός.

Ένα βασικό στοιχείο των πληροφοριών δοκιμής είναι ο καθορισμός των εισόδων στο τμήμα του λογισμικού, το οποίο τίθεται σε λειτουργία. Επιπλέον, οι πληροφορίες που συνοδεύουν τις περιπτώσεις δοκιμών, συμπεριλαμβάνουν συνήθως λεπτομέρειες σχετικές με την αναμενόμενη απόδοση. Οι εν λόγω πληροφορίες σε συνδυασμό με τις ειδικές εισόδους και τον καθορισμό του αναμενόμενου χρόνου εκτέλεσης, καθορίζουν και την συμπεριφορά του προγράμματος στην συγκεκριμένη απόπειρα δοκιμής. Λαμβάνοντας υπόψιν τα παραπάνω, γίνεται κατανοητό πως η κατασκευή μια περίπτωσης δοκιμών γίνεται αναθέτοντας συγκεκριμένες τιμές εισόδου σε μια προγραμματισμένη απόπειρα δοκιμής. Η ανάθεση αυτή γίνεται συνήθως από μοντέλα δοκιμών που καθορίζονται στον προηγούμενο στάδιο σχεδιασμού και προετοιμασίας των ελέγχων.

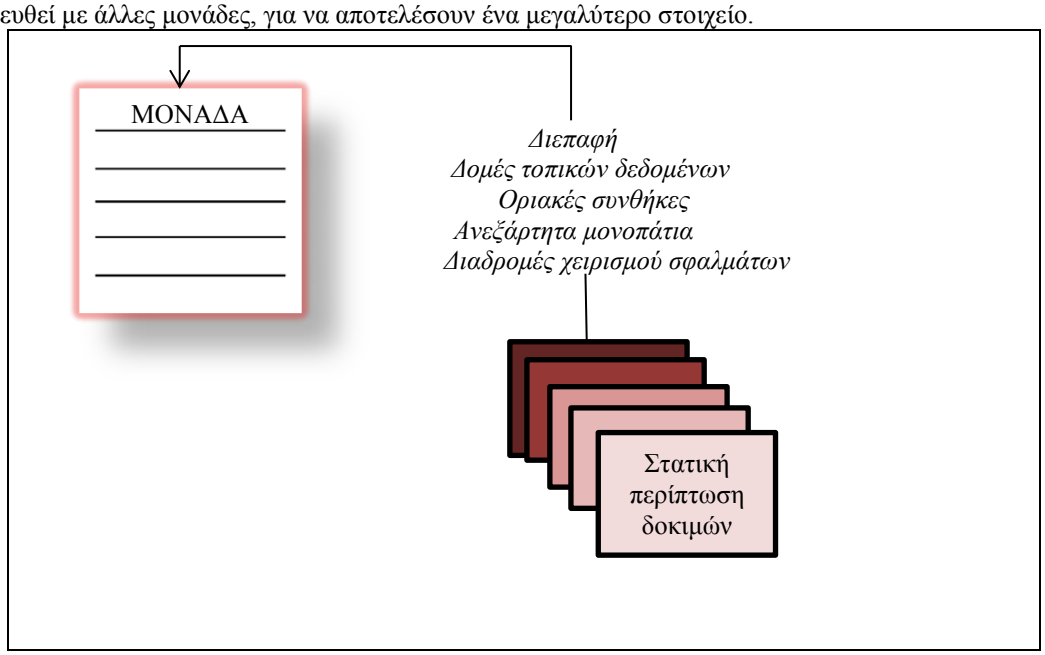
4.1.1 ΕΛΕΓΧΟΣ ΜΟΝΑΔΩΝ

Ο έλεγχος μονάδων είναι μια διαδικασία ελέγχου μεμονωμένων υποπρογραμμάτων, υπορουτίνων, κλάσεων ή διαδικασιών σε ένα πρόγραμμα[47]. Πιο συγκεκριμένα, ο έλεγχος επικεντρώνεται πρώτα στα μικρότερα δομικά στοιχεία του προγράμματος και ύστερα στο πρόγραμμα σαν σύνολο. Δεδομένου ότι μια μονάδα προγράμματος υλοποιεί μια λειτουργία, είναι φυσικό ο έλεγχος να επικεντρώνεται σε αυτήν και έπειτα το σύστημα να ενσωματώνει τις άλλες μονάδες. Έτσι δοκιμάζεται μια μονάδα μεμονωμένα, δηλαδή με αυτόνομο τρόπο. Τα κίνητρα εφαρμογής της συγκεκριμένης στρατηγικής ελέγχου είναι τρία[11]. Πρώτον, ο έλεγχος μονάδων είναι ένας τρόπος διαχείρισης και συνδυασμού των στοιχείων ελέγχου, εφόσον η προσοχή επικεντρώνεται αρχικά σε μικρότερες μονάδες του προγράμματος. Δεύτερον, διευκολύνει τον εντοπισμό και τη διόρθωση ενός ανακαλυφθέντος σφάλματος, καθώς αυτό θα βρεθεί σε συγκεκριμένη γνωστή ενότητα η οποία μάλιστα βρίσκεται σε πρώιμο στάδιο. Τέλος, ο έλεγχος δοκιμών εισάγει την έννοια της παραλληλοποίησης, αφού δίνει τη δυνατότητα να δοκιμαστούν ταυτόχρονα πολλές ενότητες. Το τελευταίο φαίνεται γραφικά στην *Εικόνα 5*. Ο έλεγχος μονάδων θεωρείται κανονικά ως συμπλήρωμα της κωδικοποίησης. Ο σχεδιασμός των δοκιμών μονάδων μπορεί να γίνει πριν αρχίσει η κωδικοποίηση ή εφόσον έχει δημιουργηθεί ο κώδικας. Μια ανασκόπηση των πληροφοριών σχεδιασμού του κώδικα, παρέχει οδηγίες προκειμένου να καθοριστούν συγκεκριμένες περιπτώσεις δοκιμών που είναι πιθανόν να αποκαλύψουν μη-φανερά σφάλματα. Κάθε δοκιμαστική περίπτωση συνδυάζεται με ένα σύνολο αναμενόμενων αποτελεσμάτων. Επειδή ένα στοιχείο του συστήματος δεν είναι αυτόματο πρόγραμμα, θα πρέπει για κάθε είδος ελέγχου του στοιχείου αυτού, να αναπτυχθεί ένα ξεχωριστό πρόγραμμα οδήγησης (driver). Το πρόγραμμα οδήγησης ή οδηγός δεν είναι τίποτα παραπάνω από ένα κύριο πρόγραμμα (main program) που δέχεται σαν είσοδο δεδομένα δοκιμαστικών περιπτώσεων, τα μεταβιβάζει στο στοιχείο που ελέγχεται και εκτυπώνει τα αποτελέσματα. Οι οδηγοί χρησιμεύουν για την αντικατάσταση μονάδων που είναι δευτερεύουσες του στοιχείου που ελέγχεται[39]. Το πρόγραμμα οδήγησης χρησιμοποιεί το περιβάλλον υλοποίησης της μονάδας, μπορεί να κάνει ελάχιστη επεξεργασία δεδομένων, εκτυπώνει την επαλήθευση της εισόδου και επιστρέφει τον έλεγχο στην μονάδα που βρίσκεται υπό δοκιμή. Διαισθητικά ένας προγραμματιστής, ελέγχει μια μονάδα ακολουθώντας τα παρακάτω:

- Εκτέλεση κάθε γραμμής του κώδικα
  Ο προγραμματιστής χρειάζεται να γνωρίζει τι συμβαίνει όταν εκτελείται μια γραμμή κώδικα. Όταν δεν είναι σε θέση να γνωρίζει το αποτέλεσμα κάθε γραμμής, τότε οι εκπλήξεις σε μεταγενέστερο στάδιο μπορεί να είναι δαπανηρές.
- Εκτέλεση κάθε συνθήκης στην μονάδα και κατηγοριοποίηση αυτών σε δυο ομάδες: αλήθεια,ψέμα.
- Ελέγχει αν η μονάδα εκτελεί την προβλεπόμενη λειτουργία χωρίς να παρατηρούνται σφάλματα.

Ο έλεγχος μονάδων εκτελείται από τον προγραμματιστή που γράφει την μονάδα προγράμματος, διότι είναι εξοικειωμένος με τις εσωτερικές λεπτομέρειες κάθε μονάδας. Ο στόχος για τον προγραμματιστή, είναι να εξασφαλίσει πως η μονάδα λειτουργεί όπως αναμενόταν. Έτσι, κάθε προγραμματιστής γίνεται υπόλογος για την ποιότητα της εργασίας του, η οποία ενδέχεται να περιλαμβάνει είτε νέο κώδικα είτε τροποποιήσεις σε ήδη υπάρχοντα. Ο σκοπός του ελέγχου μονάδων είναι να εισάγει την έννοια της ποιότητας στα χαμηλότερα επίπεδα οργάνωσης και να εξουσιοδοτεί κάθε προγραμματιστή να είναι υπεύθυνος για την ποιότητα των δικών του τμημάτων κώδικα. Τα ελαττώματα που διαπιστώθηκαν κατά τη διάρκεια ελέγχου μονάδων είναι εσωτερικά αναφερόμενα στην ομάδα ανάπτυξης λογισμικού και διορθώνονται σε αυτό το επίπεδο ιεραρχίας. Ο έλεγχος μονάδων πραγματοποιείται στο μικρότερο στοιχείο του έργου, οπότε ο αριθμός των δοκιμαστικών και των δεδομένων υπό δοκιμή είναι μικρός και δεν είναι πάντοτε να ελεγχθούν όλα τα σενάρια λειτουργίας της

εφαρμογής λογισμικού. Υπάρχουν πολλές περιπτώσεις δοκιμών που μπορούν να εκτελεστούν αφού η μονάδα συγχωνευθεί με άλλες μονάδες, για να αποτελέσουν ένα μεγαλύτερο στοιχείο.



Εικόνα 5: Έλεγχος μονάδων

4.1.2 ΕΛΕΓΧΟΣ ΟΛΟΚΛΗΡΩΣΗΣ

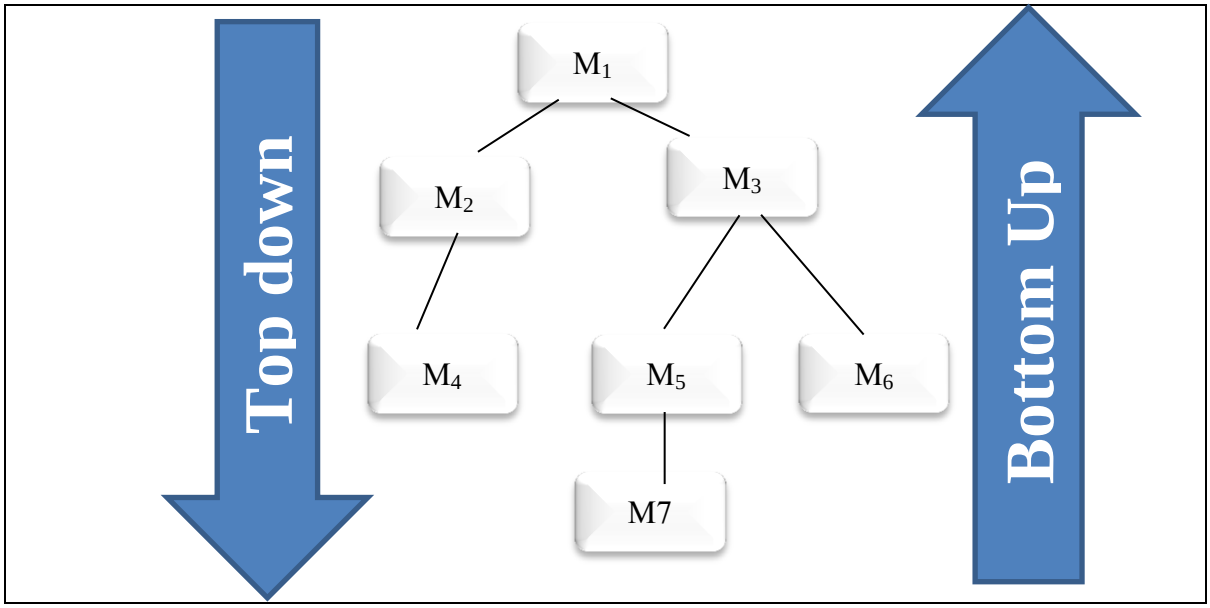
Στο πρώτο επίπεδο ελέγχου, τον έλεγχο μονάδων, το σύστημα είναι χωρισμένο σε κομμάτια υπό τον έλεγχο των προγραμματιστών. Στο επόμενο βήμα, οι μονάδες ομαδοποιούνται με σκοπό την κατασκευή ενός πλήρους συστήματος. Η κατασκευή ενός ολοκληρωμένου συστήματος εργασίας, ξεκινώντας από κομμάτια, είναι δύσκολη λόγω των πολυάριθμων σφαλμάτων που προκύπτουν στις διεπαφές. Ακόμη και η κατασκευή ενός λογικά σταθερού συστήματος με την ένωση απλών τμημάτων κώδικα περιλαμβάνει πολλές δοκιμές. Η διαδρομή από τον έλεγχο στοιχείων έως την κατασκευή ενός παραδοτέου συστήματος περιέχει δύο κύριες φάσεις δοκιμών, τον έλεγχο ολοκλήρωσης και τον έλεγχο του συστήματος. Ο πρωταρχικός στόχος του ελέγχου ολοκλήρωσης είναι η εύλογη συναρμολόγηση μονάδων, δημιουργώντας έτσι ένα σταθερό σύστημα σε εργαστηριακό περιβάλλον, το οποίο θα μπορεί να φέρει εις πέρας τις δοκιμές που συμβαίνουν στο πραγματικό περιβάλλον του συστήματος. Η σημασία του ελέγχου ολοκλήρωσης επιβεβαιώνεται από τους παρακάτω τρεις λόγους[29]:

- Οι διάφορες ενότητες δημιουργούνται από ομάδες διαφορετικών προγραμματιστών. Οι προγραμματιστές μπορεί να εργάζονται σε διαφορετικές τοποθεσίες. Παρότι μπορεί να αφιερώνεται χρόνος στην προσπάθεια σχεδιασμού και τεκμηρίωσης του συστήματος, στην πραγματικότητα τα λάθη, οι παρερμηνείες και οι παραβιάσεις είναι αναπόφευκτες καταστάσεις. Τα σφάλματα διασύνδεσης που μεταξύ ενοτήτων που αναπτύχθηκαν από διαφορετικούς προγραμματιστές θα είναι αναπόφευκτα πολλά.
- Ο έλεγχος μονάδων των επιμέρους ενοτήτων πραγματοποιείται σε ελεγχόμενο περιβάλλον χρησιμοποιώντας δοκιμαστικά προγράμματα οδήγησης, τα οποία επιστρέφουν προκαθορισμένες τιμές. Μια μονάδα υπο δοκιμή, στην πραγματικότητα, χρησιμοποιεί πολλές άλλες μονάδες για την αποτελεσματική λειτουργία της. Επομένως με τη δοκιμή της μονάδας χρησιμοποιώντας προγράμματα οδήγησης τα οποία αναλαμβάνουν τον ρόλο των συνεργαζόμενων μονάδων, είναι αρκετά δύσκολο να προβλεφθεί η συμπεριφορά της μονάδας στο πραγματικό της περιβάλλον.
- Ορισμένες μονάδες είναι περισσότερο επιρρεπείς σε σφάλματα, λόγω της πολυπλοκότητάς τους. Συνεπώς, κρίνεται σημαντικός ο εντοπισμός εκείνων των μονάδων που προκαλούν περισσότερα σφάλματα σε σύγκριση με τις υπόλοιπες.

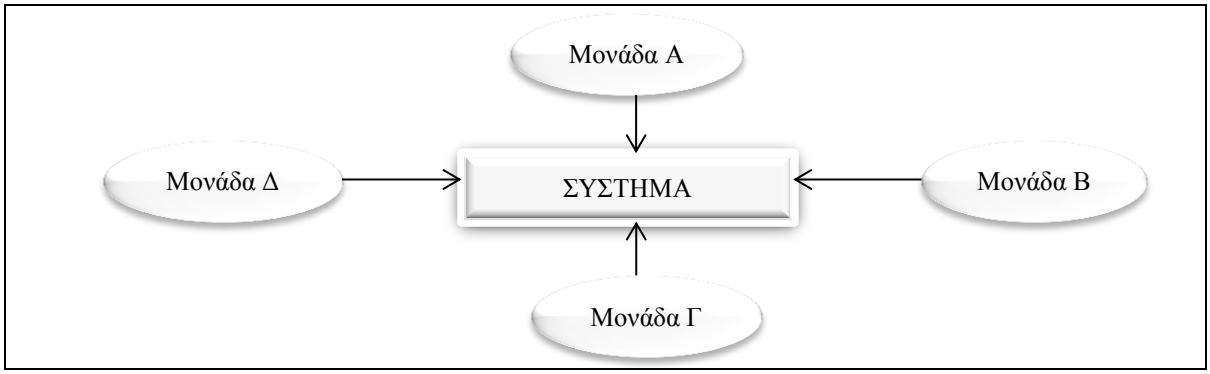
Ο στόχος του ελέγχου ολοκλήρωσης είναι να δημιουργηθεί μια λειτουργική έκδοση του συστήματος μέσα από την σταδιακή συναρμολόγηση των ξεχωριστών μονάδων και την εξασφάλιση πως οι ενότητες που προστίθενται λειτουργούν όπως αναμένεται χωρίς να διαταράσσουν τις λειτουργίες των μονάδων που έχουν ήδη ενωθεί. Με άλλα λόγια, ο έλεγχος ολοκλήρωσης είναι μια συστηματική τεχνική για την κατασκευή της αρχιτεκτονικής του λογισμικού ενώ ταυτόχρονα διεξάγει δοκιμές για την αποκάλυψη σφαλμάτων που σχετίζονται με διασύνδεση. Το συγκεκριμένο επίπεδο ελέγχου κατασκευάζει και δοκιμάζει μικρές ομάδες μονάδων, όπου τα σφάλματα είναι

ευκολότερο να εντοπιστούν. Επιπλέον, ελέγχει τις διεπαφές και την συνολική λειτουργία μιας ομάδας μονάδων. Ο έλεγχος ολοκλήρωσης είναι πλήρης όταν έχει ενσωματώσει πλήρως όλες τις μονάδες στο σύστημα, έχουν εκτελεστεί όλες οι περιπτώσεις δοκιμών, όλα τα σφάλματα έχουν εντοπιστεί και το σύστημα είναι σε θέση να επανεξεταστεί. Υπάρχουν τέσσερις κύριες στρατηγικές ενσωμάτωσης των μονάδων στο στάδιο ελέγχου ολοκλήρωσης[48]:

- Έλεγχος ολοκλήρωσης από πάνω προς τα κάτω (top-down integration testing)
  Η μέθοδος αυτή προσεγγίζει διαισθητικά την πραγματική αρχιτεκτονική δομή του λογισμικού. Οι μονάδες ελέγχονται σύμφωνα με την εξής ιεραρχία ελέγχου: ο έλεγχος ξεκινά από την μονάδα που περιέχει το κύριο πρόγραμμα, ενώ οι υπόλοιπες υποδεέστερες μονάδες ενσωματώνονται σταδιακά σε αυτό, είτε ακολουθώντας τον αλγόριθμο επιλογής μονοπατιού κατά-βάθος (depth-first), είτε ακολουθώντας τον αλγόριθμο επιλογής μονοπατιού κατά πλάτος (breadth-first). Στην Εικόνα 6 φαίνεται και γραφικά η συγκεκριμένη στρατηγική. Το κύριο πρόγραμμα είναι η μονάδα M_1, ενώ γενικά η ιεραρχία μονάδων αποτελείται από επτά μονάδες και τέσσερα επίπεδα. Σύμφωνα με την επιλογή μονάδων κατά βάθος, πρώτα θα ελεγχθεί η μονάδα M_1 και στην συνέχεια (υποθέτοντας πως επιλέγεται το αριστερό μονοπάτι) οι μονάδες M_2, M_4 κλπ. Σε κάθε βήμα, αρχικά ενσωματώνεται η μονάδα στον σύνολο των ήδη ελεγμένων μονάδων και έπειτα ελέγχεται αν λειτουργεί σε συνδυασμό με τις υπόλοιπες. Αν επιλεγούν οι μονάδες με την τακτική κατά πλάτος, έπειτα από τον έλεγχο της κύριας μονάδας M_1, ενσωματώνονται και ελέγχονται (επιλέγοντας φορά από αριστερά προς τα δεξιά) οι μονάδες M_2 και M_3 κλπ.
- Έλεγχος ολοκλήρωσης από κάτω προς τα πάνω (bottom-up integration testing)
  Κατά την εν λόγω τεχνική, ο έλεγχος ξεκινά από τις ατομικές μονάδες, δηλαδή τα στοιχεία που βρίσκονται στο κατώτερα επίπεδα της δομής προγράμματος και κατευθύνεται σταδιακά προς τα πάνω. Δηλαδή, σύμφωνα με την Εικόνα 6, θα ελεγχθεί πρώτα η μονάδα M_7, στην συνέχεια αυτή θα ενσωματωθεί με τον γονέα της, την μονάδα M_5 και θα ελεγχθεί ο συνδυασμός αυτών, έστω $M_α$. Στην συνέχεια θα ελεγχθεί η ατομική μονάδα M_6. Η M_6 θα ενωθεί με την $M_α$ και θα ελεγχθεί ο συνδυασμός τους κλπ.
- Έλεγχος ολοκλήρωσης όλων των μονάδων ταυτόχρονα (big bang integration testing)
  Η συγκεκριμένη τακτική κάνει ακριβώς ότι περιγράφει το όνομά της. Όλες οι μονάδες ενώνονται και ελέγχονται την ίδια στιγμή. Επειδή ο έλεγχος γίνεται σε όλα τα στοιχεία του προγράμματος την ίδια στιγμή, σε περίπτωση αποτυχίας είναι δύσκολο να εντοπιστεί η πηγή από την οποία προήλθε το λάθος. Η προσέγγιση ελέγχου εικονικά φαίνεται στην Εικόνα 7. Ο εν λόγω τρόπος ελέγχου προτείνεται για συστήματα μικρού μεγέθους.
- Υβριδικός έλεγχος ολοκλήρωσης (hybrid integration testing)
  Κατά την υβριδική μέθοδο, ο έλεγχος συνδυάζει τις μεθόδους από κάτω προς τα πάνω και από πάνω προς τα κάτω. Ένα ιεραρχικό σύστημα αποτελείται από τρία στρώματα. Το κάτω στρώμα περιέχει όλες τις μονάδες που χρησιμοποιούνται συχνά. Σε αυτές, εφαρμόζεται η μέθοδος από κάτω προς τα πάνω. Το ανώτατο στρώμα περιέχει μονάδες που υλοποιούν σημαντικές αποφάσεις σχεδιασμού. Σε αυτές εφαρμόζεται η μέθοδος από πάνω προς τα κάτω. Οι υπόλοιπες μονάδες τοποθετούνται στο μεσαίο στρώμα. Οι μονάδες αυτές συνήθως ελέγχονται ταυτόχρονα, έπειτα ενσωματώνονται με την ανώτατη στρώση και μετά με την κατώτερη[47].



Εικόνα 6: Έλεγχος ολοκλήρωσης Top-down & bottom-up



Εικόνα 7: Έλεγχος ολοκλήρωσης Big Bang

4.1.3 ΕΛΕΓΧΟΣ ΣΥΣΤΗΜΑΤΟΣ

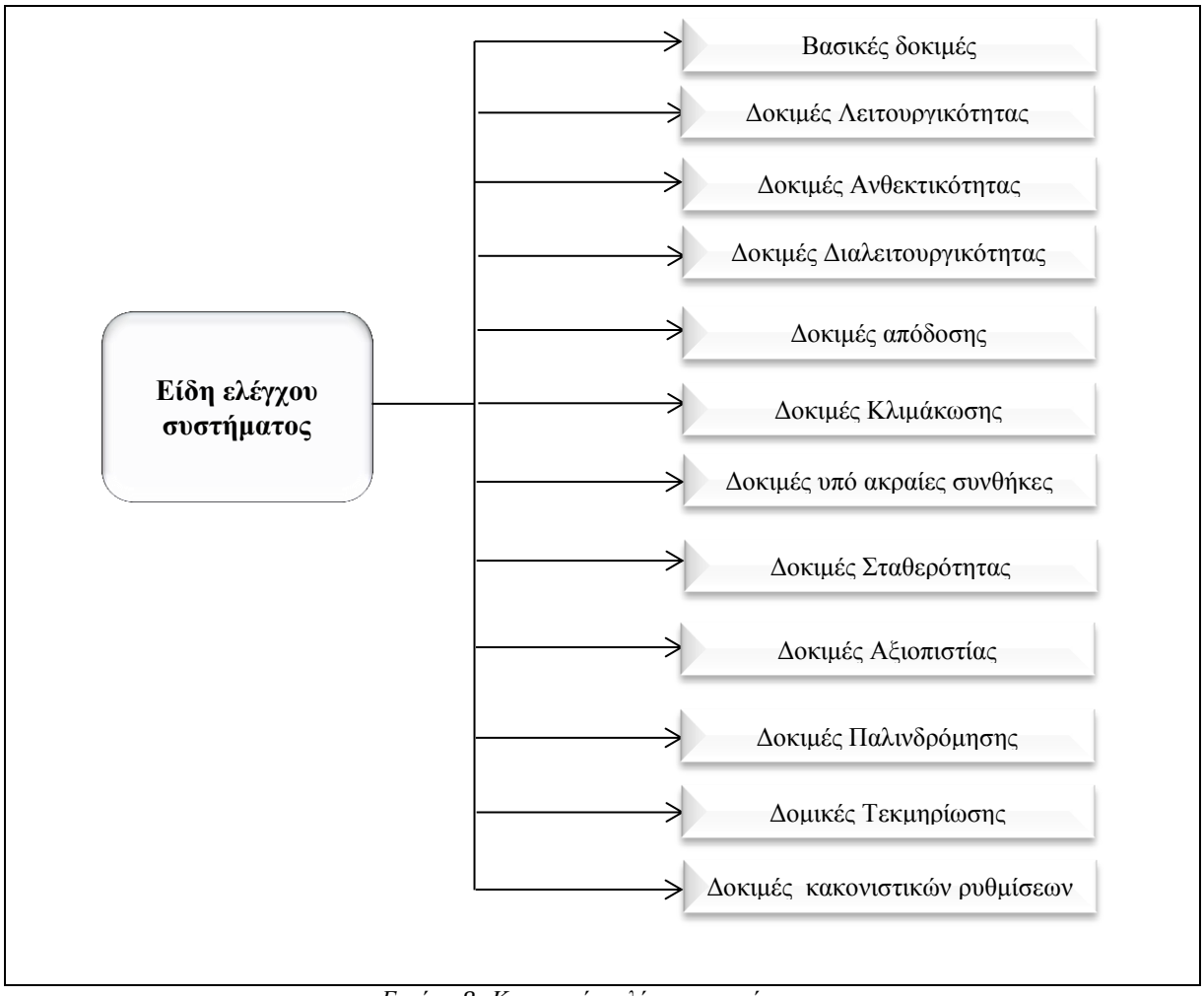
Ο έλεγχος συστήματος είναι η πιο πολυσυζητημένη και η πιο δύσκολη διαδικασία δοκιμών και έχει ένα συγκεκριμένο σκοπό: να συγκρίνει το σύστημα ή το πρόγραμμα με τους αρχικούς του στόχους. Ο έλεγχος συστημάτων δεν περιορίζεται στα συστήματα. Εάν το αντικείμενο εξέτασης, είναι ένα πρόγραμμα, ο έλεγχος συστήματος είναι μια προσπάθεια απόδειξης πως το πρόγραμμα στο σύνολό του, δεν ανταποκρίνεται στους αρχικούς στόχους του. Επιπλέον, ο έλεγχος του συστήματος, εξ ορισμού, είναι αδύνατος εάν δεν υπάρχουν καταγεγραμμένες μετρήσιμες αξιολογήσεις της προσέγγισης στο στόχο. Αναζητώντας διαφορές μεταξύ του τελικού προγράμματος και των αρχικών στόχων και προδιαγραφών ανακαλύπτονται τα σφάλματα μετάφρασης που έγιναν κατά τη διαδικασία σχεδιασμού των εξωτερικών προδιαγραφών. Το συγκεκριμένο βήμα είναι συνήθως και το πιο επιρρεπές σε λάθη[11].

Ο στόχος του ελέγχου σε επίπεδο συστήματος, είναι η εξασφάλιση πως μια εφαρμογή συμμορφώνεται με τις απαιτήσεις των πελατών. Χρειάζεται αρκετή προσπάθεια προκειμένου να εξασφαλιστεί πως το προϊόν λογισμικού ανταποκρίνεται στις απαιτήσεις των πελατών και το σύστημα είναι αποδεκτό. Κατά τον έλεγχο του συστήματος εφαρμόζονται πλήθος περιπτώσεων δοκιμών, για να ελαχιστοποιηθεί και η παραμικρή πιθανότητα αποτυχίας και το σύστημα να γίνει αποδεκτό[47]. Τα αντικείμενα που βρίσκονται υπό έλεγχο, δηλαδή τα συστήματα, είναι πολύπλοκα καθώς αποτελούνται από υλικό και λογισμικό. Συνεπώς, δημιουργείται η ανάγκη κάλυψης πολλαπλών πιθανών συμπεριφορών του συστήματος, από το μικρότερο στοιχείου υλικού έως το πολυπλοκότερο σύστημα. Ο έλεγχος συστήματος διακρίνεται σε διαφορετικές ομάδες δοκιμών ανάλογα το αντικείμενο του συστήματος που βρίσκεται υπό εξέταση. Η εν λόγω κατηγοριοποίηση απεικονίζεται στην *Εικόνα 8*. Η παραπάνω κατηγοριοποίηση έχει τρία βασικά πλεονεκτήματα:

- Οι μηχανικοί ελέγχου μπορούν να εστιάσουν με ακρίβεια σε διαφορετικές πτυχές ενός συστήματος, ελέγχοντας μία κάθε φορά.
- Οι μηχανικοί ελέγχου μπορούν να δώσουν προτεραιότητα στις εργασίες τους, βασιζόμενοι στις κατηγορίες δοκιμών. Για παράδειγμα, ο έλεγχος της αντοχής ενός συστήματος, θα γίνει αφού έχει ολοκληρωθεί ο έλεγχος λειτουργίας του.
- Προγραμματίζοντας τον έλεγχο συστήματος, βασιζόμενος στην κατηγοριοποίησή του, ο μηχανικός ελέγχου έχει την ευκαιρία να σχεδιάζει μία καλά ισορροπημένη σουΐτα δοκιμών (test suite). Έτσι, η σουίτα δοκιμών θα περιέχει ελέγχους που θα εξετάζουν ισόβαθμα κάθε πτυχή του συστήματος.

Παρακάτω περιγράφονται οι κατηγορίες δοκιμών που συνιστούν τον έλεγχο συστήματος:

- Βασικές δοκιμές (Basic tests)
  Οι βασικές δοκιμές παρέχουν μια ένδειξη πως το σύστημα μπορεί να εγκατασταθεί, να διαμορφωθεί και να τεθεί σε λειτουργία.
- Δοκιμές λειτουργικότητας (Functionality tests)
  Παρέχουν πλήθος ολοκληρωμένων ελέγχων σε όλο το εύρος απαιτήσεων του συστήματος.
- Δοκιμές Ανθεκτικότητας (Robustness tests)
  Καθορίζουν την ικανότητα του συστήματος να ανακτάται μετά από διάφορα σφάλματα εισόδου και άλλες καταστάσεις αποτυχίας.
- Δοκιμές διαλειτουργικότητας (Interoperability tests)
  Εξετάζουν τη δυνατότητα του συστήματος να λειτουργεί σε συνδυασμό με άλλα προϊόντα τρίτων κατασκευαστών.
- Δοκιμές απόδοσης (Performance tests)
  Μετρούν τα χαρακτηριστικά απόδοσης του συστήματος, όπως τον χρόνο απόκρισης του συστήματος κάτω από ακραίες συνθήκες.
- Δοκιμές κλιμάκωσης (Scalability tests)
  Καθορίζουν τα όρια κλιμάκωσης του συστήματος υπό την οπτική του χρήστη, την γεωγραφική κλίμακα και την κλιμάκωση πόρων.
- Δοκιμές υπό ακραίες συνθήκες (Stress tests)
  Εξετάζουν το σύστημα σε κάθε πιθανή ακραία συνθήκη και όταν αυτό αποτύχει, καθορίζουν την αιτία της αποτυχίας. Ακραίες συνθήκες μπορεί να είναι η λειτουργία του συστήματος για πολλές μέρες, η χρήση μη λογικών μεταβλητών ως είσοδο στο σύστημα, η χρήση του συστήματος υπό ακραίες θερμοκρασίες κλπ.
- Δοκιμές σταθερότητας (Stability tests)
  Παρέχουν την γνώση πως το σύστημα μπορεί να παραμείνει σταθερό για μεγάλο χρονικό διάστημα, χρησιμοποιώντας στο μέγιστο κάθε λειτουργία του.
- Δοκιμές αξιοπιστίας (Reliability tests)
  Μετρούν την ικανότητα του συστήματος να συνεχίζει να λειτουργεί για μεγάλο χρονικό διάστημα χωρίς να αποτυγχάνει.
- Δοκιμές παλινδρόμησης (Regression tests)
  Ελέγχουν την ικανότητα του συστήματος να παραμένει σταθερό, όσο οι μονάδες ενσωματώνονται σε αυτό και όσο διαρκούν οι διαδικασίες συντήρησης.
- Δοκιμές τεκμηρίωσης (Documentation tests)
  Διασφαλίζουν πως οι οδηγίες χρήσης του συστήματος είναι κατανοητές και προσβάσιμες.
- Δοκιμές συμμόρφωσης με τις νομοθετικές και κανονιστικές μεταρρυθμίσεις (Regulatory tests)
  Εξασφαλίζουν ότι το σύστημα πληροί τις απαιτήσεις των νομοθετικών και κανονιστικών μεταρρυθμίσεων της χώρας όπου θα εφαρμοστεί.



Εικόνα 8: Κατηγορίες ελέγχου συστήματος

4.1.4 ΕΛΕΓΧΟΣ ΕΠΙΠΕΔΟΥ ΑΠΟΔΟΧΗΣ

Ένα προϊόν είναι έτοιμο να παραδοθεί στον πελάτη, εφόσον περάσει η ομάδα ελέγχου συστήματος είναι ικανοποιημένη με την απόδοση του προϊόντος, διενεργώντας δοκιμές σε επίπεδο συστήματος. Οι πελάτες, έπειτα εκτελούν ελέγχους επιπέδου αποδοχής βασιζόμενοι στις δικές τους προσδοκίες και απαιτήσεις[37]. Οι υπηρεσίες που προσφέρει ένα επιτυχημένο προϊόν λογισμικού, συνήθως χρησιμοποιούνται από εκατομμύρια χρήστες. Ο έλεγχος επιπέδου αποδοχής καθορίζει εάν το σύστημα ικανοποιεί τα κριτήρια αποδοχής προκειμένου το λογισμικό να γίνει αποδεκτό από τους πελάτες. Βοηθά τον πελάτη να αποφασίσει εάν δέχεται ή όχι το σύστημα λογισμικού[49]. Ο πελάτης διατηρεί το δικαίωμα να αρνηθεί να δεχτεί την παραλαβή του προϊόντος εάν δεν είναι επιτυχημένος ο έλεγχος αποδοχής. Υπάρχουν δύο κατηγορίες ελέγχου επιπέδου αποδοχής:

- Έλεγχος αποδοχής από τον χρήστη (User Acceptance Testing-UAT)
- Έλεγχος αποδοχής από μια άλλη επιχείρηση (Business Acceptance Testing-BAT)

Η πρώτη κατηγορία ελέγχου διεξάγεται από τον πελάτη για να εξασφαλίσει ότι το σύστημα ικανοποιεί τα συμβατικά κριτήρια αποδοχής προκειμένου να διαβεβαιώσει πως το προϊόν ικανοποιεί τις ανάγκες των χρηστών. Ωστόσο, ο πραγματικός σχεδιασμός και η εκτέλεση των δοκιμών ελέγχου αποδοχής δεν χρειάζεται να πραγματοποιηθούν άμεσα από τον πελάτη. Συχνά συμβουλευτικές εταιρείες προσφέρουν τις υπηρεσίες τους για να αξιολογήσουν το προϊόν από την πλευρά του χρήστη, βασιζόμενες πάντα στα κριτήρια αποδοχής που έχει καθορίσει ο πελάτης. Οι έλεγχοι αποδοχής από μια επιχείρηση αποτελούν μια πρόβλεψη των ελέγχων αποδοχής από τον χρήστη, στις εγκαταστάσεις της εταιρείας παραγωγής λογισμικού. Συνεπώς τον ρόλο του πελάτη σε αυτό το βήμα ελέγχου, μπορεί να τον αναλάβει, εκτός από τον τελικό χρήστη, μια εταιρεία που χρησιμοποιεί το λογισμικό και το αξιολογεί. Τα κριτήρια αποδοχής πρέπει να καθοριστούν και να συμφωνηθούν μεταξύ του προμηθευτή και του πελάτη για να αποφευχθούν τυχόν παρερμηνείες στο μέλλον. Η συμβουλευτική εταιρεία που θα αναλάβει τους ελέγχους είτε ο χρήστης, σχεδιάζουν και υλοποιούν τις περιπτώσεις δοκιμών που αυτοί επιθυμούν. Η αποδοχή εγγράφου που περιέχει τα κριτήρια αποδοχής αποτελούν μέρος της σύμβασης που

υπογράφεται μεταξύ των δύο. Τα βήματα των δοκιμών ελέγχου αποδοχής φαίνονται στον Πίνακας 4. Οι κύριοι στόχοι του ελέγχου επιπέδου αποδοχής είναι οι εξής:

- Διαβεβαιώνει πως το σύστημα συμμορφώνεται με τα καθορισμένα κριτήρια αποδοχής.
- Εντοπίζει και επιλύει τις διαφορές και τις αποκλίσεις με τα κριτήρια, εάν υπάρχουν.
- Προσδιορίζει την ετοιμότητα του συστήματος για ξαφνική διακοπή ενεργών λειτουργιών.
- Η ομάδα ελέγχου αποδοχής συγγράφει μια αναφορά που περιέχει τις συνθήκες αποδοχής.

Πίνακας 4: Έλεγχος αποδοχής

Βήμα 1 **Ανάλυση Απαιτήσεων**	**Βήμα 2** **Πλάνο ελέγχου**	**Βήμα 3** **Εκτέλεση ελέγχου**
-Ορθότητα και πληρότητα λειτουργίας -Ακρίβεια -Ακεραιότητα δεδομένων -Μετατροπή δεδομένων -Αντίγραφα ασφαλείας και αποκατάσταση -Ανταγωνιστικότητα -Ευχρηστία	-Όνομα έργου ελέγχου -Ιστορικό αναθεώρησης -Ορολογίες και ορισμοί -Όνομα της ομάδας ελέγχου -Ημερομηνία αποτελεσμάτων -Επισκόπηση του πλάνου ελέγχου -Αναφορές	-Εκπαίδευση πελάτη για την χρήση του λογισμικού -Επικοινωνία προγραμματιστών και πελατών -Επίλυση λαθών
-Εκτέλεση -Αντοχή σε δύσκολες συνθήκες -Αξιοπιστία και διαθεσιμότητα -Συντήρηση -Αντοχή και έλεγχος ανθεκτικότητας -Διαχρονικότητα -Διαθεσιμότητα και ασφάλεια -Συμβατότητα και διαλειτουργικότητα -Εγκατάσταση και αναβάθμιση -Ευελιξία -Τεκμηρίωση	Επίσης για κάθε ξεχωριστό κριτήριο, στο πλάνο ελέγχου προστίθενται τα παρακάτω: 1. Εισαγωγή 2. Κατηγορία ελέγχου αποδοχής 3. Περιβάλλον λειτουργίας 4. Λεπτομέρειες (ID,Τίτλος,Αντικείμενο, Διαδικασία τρέχοντος ελέγχου) 5. Χρονοδιάγραμμα 6. Ανθρώπινο δυναμικό	Σε περίπτωση εύρεσης λάθους ή μη συμμόρφωσης με κάποιο κριτήριο, ο μηχανικός ελέγχου αποδοχής κάνει μια αλλαγή στα κριτήρια κάνοντας τα εξής: 1. Καταγράφει το μοναδικό νούμερο της αλλαγής, τα κριτήρια που επηρεάστηκαν 2. Περιγράφει το πρόβλημα και την αλλαγή που πρέπει να γίνει 3. Σημειώνει την αλλαγή στο σύστημα και στην χρήση 4. Σημειώνει το όνομα των μηχανικών ελέγχου αποδοχής και της ομάδας έγκρισης της αλλαγής.
Βήμα 4 **Τεκμηρίωση**		
Το έγγραφο τεκμηρίωσης του ελέγχου αποδοχής περιέχει: 1. Ημερομηνία 2. Τελική κατάσταση ελέγχου (PASS\FAIL) 3. Λεπτομέρειες Σφαλμάτων (ID, περιγραφή) 4. Νούμερο αλλαγής κριτηρίου 5. Κατάσταση της δοκιμής (πλήθος δοκιμαστικών περιπτώσεων που εκτελέστηκαν, πόσες είναι επιτυχείς, πόσες ανεπιτυχείς, πλήθος δοκιμαστικών περιπτώσεων που δεν έχουν εκτελεστεί ακόμα) 6. Σύνοψη όλων των ελέγχων 7. Αξιολόγηση και προτάσεις 8. Έγκριση		

4.2 ΜΕΘΟΔΟΙ ΕΛΕΓΧΟΥ ΛΟΓΙΣΜΙΚΟΥ

Υπάρχει μια διαρκής συζήτηση μεταξύ των μηχανικών λογισμικού για το κατά πόσο οι εκροές της λειτουργίας ενός λογισμικού είναι αρκετές προκειμένου να αξιολογηθεί η απόδοσή του. Ορισμένοι ισχυρίζονται πως η εσωτερική δομή του λογισμικού και οι υπολογισμοί που εκτελεί στις εσωτερικές μεθόδους του, θα πρέπει να ελεγχθούν ικανοποιητικά σε συνδυασμό με τα εξωτερικά χαρακτηριστικά του προϊόντος λογισμικού[9]. Συνεπώς έχουν αναπτυχθεί, δύο αντιτιθέμενες προσεγγίσεις της ποιότητας λογισμικού:

- Έλεγχος μαύρου κουτιού (Black-box testing)
  Ονομάζεται αλλιώς έλεγχος λειτουργικότητας. Προσδιορίζει τα σφάλματα μόνο παρατηρώντας τα αποτελέσματα της λειτουργίας του λογισμικού. Σε περίπτωση που οι έξοδοι υλοποίησης του λογισμικού είναι οι αναμενόμενες, ο έλεγχος μαύρου κουτιού αγνοεί τις εσωτερικές μετρήσεις και τους εσωτερικούς υπολογισμούς του λογισμικού.
- Έλεγχος λευκού κουτιού (White-box testing)
  Ονομάζεται αλλιώς έλεγχος εσωτερικής δομής. Εξετάζει τις εσωτερικές διαδρομές των υπολογισμών που εφαρμόζει το λογισμικό, προκειμένου να εντοπίσει την πηγή των σφαλμάτων. Ο βασικός σκοπός του ελέγχου λευκού κουτιού, ή αλλιώς του ελέγχου γυάλινου κουτιού, είναι η διερεύνηση της ορθότητας της δομής του κώδικα.

Η κύρια διαφορά μεταξύ ελέγχου μαύρου και λευκού κουτιού εντοπίζεται στην οπτική του καθενός. Οι έλεγχοι λειτουργικότητας επικεντρώνονται στην εξωτερική συμπεριφορά ενός συστήματος λογισμικού χωρίς να επιτρέπουν τον έλεγχο εσωτερικά. Από την άλλη πλευρά, οι έλεγχοι εσωτερικής δομής επικεντρώνονται στην αξιολόγηση των εσωτερικών μεθόδων του λογισμικού, ενώ ταυτόχρονα επιτρέπουν την προσέγγιση στα εσωτερικά στοιχεία του προγράμματος[22]. Μια σφαιρική εικόνα των δύο μεθόδων ελέγχου φαίνεται στον Πίνακας 5, όπου μπορούν να εντοπιστούν εύκολα οι διαφορές τους.

4.2.1 ΕΛΕΓΧΟΣ ΜΑΥΡΟΥ ΚΟΥΤΙΟΥ

ΟΡΙΣΜΟΣ ΕΛΕΓΧΟΥ ΜΑΥΡΟΥ ΚΟΥΤΙΟΥ-IEEE

1. **Έλεγχος μαύρου κουτιού (black-box testing) είναι η δοκιμή που αγνοεί τον εσωτερικό μηχανισμό ενός συστήματος ή ενός στοιχείου και εστιάζει αποκλειστικά στις εξόδους που παράγονται ως απόκριση σε προκαθορισμένες εισόδους και συνθήκες εκτέλεσης.**
2. **Έλεγχος μαύρου κουτιού είναι η διεξαγωγή δοκιμών για την αξιολόγηση της συμμόρφωσης ενός συστήματος ή ενός στοιχείου με συγκεκριμένες λειτουργικές απαιτήσεις.**

Ο Έλεγχος μαύρου κουτιού επιτρέπει την αξιολόγηση του λογισμικού στους εξής τομείς[50]:

- Ορθότητα
- Τεκμηρίωση
- Χρόνος αντίδρασης στις δοκιμές
- Αξιοπιστία
- Αντοχή του λογισμικού να λειτουργεί κάτω από ακραίες συνθήκες
- Ασφάλεια
- Ευχρηστία
- Συντηρησιμότητα
- Ελεγξιμότητα
- Ευελιξία
- Φορητότητα
- Επαναχρησιμοποιησιμότητα
- Διαλειτουργικότητα λογισμικού
- Διαλειτουργικότητα εξοπλισμού

Εκτός από τον έλεγχο της ορθότητας της εξόδου και της συντηρησιμότητας του προϊόντος, οι περισσότερος από τις άλλες δοκιμές εκτελούνται αποκλειστικά και μόνο από τη διαδικασία ελέγχου μαύρου κουτιού. Άρα, αυτό εξηγεί πως είναι απαραίτητο και αναντικατάστατο κομμάτι της ελεγκτικής διαδικασίας ενός λογισμικού και δεν μπορεί να αντικατασταθεί πλήρως από τον έλεγχο λευκού κουτιού[51]. Σε αντίθεση, με τον έλεγχο λευκού κουτιού ο οποίος εκτελείται νωρίς στη διαδικασία δοκιμής, ο έλεγχος λειτουργικότητας εφαρμόζεται κατά τα μεταγενέστερα στάδια των δοκιμών του συστήματος. Ο έλεγχος μαύρου κουτιού επιτρέπει την εκτέλεση των περισσότερων ειδών ελέγχου ενώ από τις δοκιμές που γίνονται αποκλειστικά κατά την εφαρμογή του, ιδιαίτερης σημασίας κρίνονται οι έλεγχοι απόδοσης του συστήματος, οι δοκιμές φορτίου και διαθεσιμότητας. Συγχρόνως, για τις δοκιμές οι οποίες μπορούν να γίνουν και με την μέθοδο του μαύρου κουτιού αλλά και του λευκού, ο

έλεγχος μαύρου κουτιού υπερτερεί καθώς απαιτεί λιγότερους πόρους για το ίδιο σύστημα λογισμικού. Ωστόσο, η μέθοδος του μαύρου κουτιού εγκυμονεί τον κίνδυνο πως μέσα από έναν τυχαίο συνδυασμό πολλών λαθών μπορεί να προκύψει κατά λάθος η σωστή έξοδος του συστήματος, με αποτέλεσμα να μην εντοπιστούν ποτέ αυτά τα λάθη. Επιπλέον, οι έλεγχοι μαύρου κουτιού μπορεί να μην εξετάζουν σημαντικό κομμάτι του κώδικα ή να το αγνοούν, λόγω ανεπάρκειας των μηχανικών ελέγχου να καλύψουν κάθε πιθανή γραμμή του, μέσω των δοκιμαστικών περιπτώσεων. Τέλος, πολλές φορές παρατηρείται η έλλειψη συμμόρφωσης με τα πρότυπα της κωδικοποίησης, δηλαδή αδυναμία της μεθόδου του μαύρου κουτιού να ελέγξει την ποιότητα της κωδικοποίησης[25].

4.2.2 ΕΛΕΓΧΟΣ ΛΕΥΚΟΥ ΚΟΥΤΙΟΥ

ΟΡΙΣΜΟΣ ΕΛΕΓΧΟΥ ΛΕΥΚΟΥ ΚΟΥΤΙΟΥ-IEEE
Έλεγχος λευκού κουτιού (white-box testing) είναι η δοκιμή που λαμβάνει υπόψιν τον εσωτερικό μηχανισμό ενός συστήματος ή ενός στοιχείου.

Ο έλεγχος λευκού κουτιού είναι μια φιλοσοφία σχεδιασμού δοκιμαστικών περιπτώσεων που α) εγγυάται πως έχουν εκτελεστεί όλα τα ανεξάρτητα στοιχεία μέσα σε μια ενότητα τουλάχιστον μια φορά, β) δίνει και τις δυο τιμές σε όλες τις λογικές συνθήκες ενός κώδικα, γ) εκτελεί όλους τους βρόχους εντός των ορίων λειτουργίας τους και δ) εξετάζει τα δεδομένα εισόδου για να διασφαλίσει την εγκυρότητά τους[52]. Η μέθοδος λευκού κουτιού επιτρέπει την εκτέλεση της επεξεργασίας δεδομένων και των δοκιμών ορθότητας υπολογισμών. Επιπλέον, στα πλαίσια του ελέγχου λευκού κουτιού, εξετάζονται η ποιότητα του λογισμικού, η συντηρησιμότητά του και η δυνατότητα επαναχρησιμοποίησής του. Προκειμένου να διεξαχθούν οι έλεγχοι ορθότητας και επεξεργασίας δεδομένων, θα πρέπει να εξεταστεί κάθε υπολογιστική λειτουργία καθώς και η ακολουθία των πράξεων που εκτελούνται στα πλαίσια μια δοκιμαστικής περίπτωσης. Η παραπάνω διαδικασία αποτελεί και το μοναδικό τρόπο επαλήθευσης πως όλες διαδικασίες επεξεργασίας δεδομένων προγραμματίστηκαν σωστά. Όσον αφορά την πιστοποίηση του λογισμικού, ο έλεγχος λευκού κουτιού εστιάζει στην εξέταση του κώδικα λογισμικού (συμπεριλαμβανομένων και των σχολίων), προκειμένου να αποφανθεί εάν συμμορφώνεται με τα πρότυπα κωδικοποίησης. Η αξιολόγηση της συντηρησιμότητας του λογισμικού, με την μέθοδο λευκού κουτιού, επικεντρώνεται σε ειδικά χαρακτηριστικά όπως αυτά που έχουν εγκατασταθεί με σκοπό τον εντοπισμό της αποτυχίας, τις βελτιώσεις του λογισμικού κλπ. Οι έλεγχοι για την επαναχρησιμοποίηση του λογισμικού, εξετάζουν το ποσοστό συμμόρφωσης του λογισμικού με το υλικό και τις αλλαγές που έγιναν προκειμένου αυτό να μπορεί να χρησιμοποιηθεί σε μελλοντικά πακέτα λογισμικού[53]. Ένα πλεονέκτημα του ελέγχου λευκού κουτιού είναι η διεξοδική ανάλυσή του και η εστίασή του στον παραγόμενο κώδικα. Επειδή υπάρχει η γνώση της εσωτερικής δομής του κώδικα εξαρχής, η λογική, τα λάθη ή η σκόπιμη αναστάτωση εκ μέρους ενός προγραμματιστή εντοπίζονται πολύ πιο εύκολα. Ένα μειονέκτημα της μεθόδου είναι ότι δεν επαληθεύει την ορθότητα των προδιαγραφών. Επίσης, η μέθοδος δεν έχει την ικανότητα να εντοπίζει διαδρομές που λείπουν και σφάλματα που είναι ευαίσθητα σε δεδομένα. Τέλος, ο έλεγχος λευκού κουτιού δεν είναι σε θέση να εκτελέσει όλες τις πιθανές λογικές διαδρομές, καθώς αν το έκανε, θα έπρεπε να εκτελέσει ένα άπειρο αριθμό δοκιμών[54]. Τα χαρακτηριστικά του ελέγχου λευκού κουτιού περιορίζουν την χρήση του σε ενότητες λογισμικού με υψηλό κόστος αποτυχίας, συνεπώς με πολύ υψηλό ρίσκο. Σε αυτές τις περιπτώσεις είναι εξαιρετικά κρίσιμος ο εντοπισμός και η διόρθωση σφαλμάτων νωρίς, σε επίπεδο κώδικα.

Πίνακας 5: Σφαιρική εικόνα white box testing & black box testing

	Έλεγχος λευκού κουτιού	**Έλεγχος μαύρου κουτιού**
Περιγραφή Μεθόδου	Έλεγχος εσωτερικής δομής που εστιάζει στα εσωτερικά χαρακτηριστικά ενός προγράμματος.	Λειτουργικός έλεγχος που εστιάζει στα εξωτερικά χαρακτηριστικά, βασιζόμενος στις απαιτήσεις ή τις προδιαγραφές.
Αντικείμενα ελέγχου	Σχεδιασμός μονάδων, εκτέλεση, έλεγχος συμμόρφωσης των εσωτερικών συναρτήσεων και δομικών στοιχείων με τις σχεδιαστικές προδιαγραφές, ανάλυση τρόπου εκτέλεσης του κώδικα	Λειτουργικότητα, απαιτήσεις, χρήση, πρότυπα χρήσης, ορθότητα, τεκμηρίωση, έλεγχος στο σύστημα εάν ανταποκρίνεται στις απαιτήσεις της επιχείρησης
Επίπεδα ελέγχου	Έλεγχος μονάδων Έλεγχος ολοκλήρωσης	Έλεγχος συστήματος Έλεγχος επιπέδου αποδοχής

Τεχνικές	Εισαγωγή σφάλματος, δοκιμή συμβολοσειράς εισόδου, διαχείριση σφαλμάτων, κάλυψη καταστάσεων, κάλυψη απόφασης, κάλυψη συνθήκης, κάλυψη διαδρομής, κάλυψη ροής δεδομένων, έλλειψη μνήμης, κυκλική πολυπλοκότητα	Ανάλυση οριακής τιμής, δοκιμές απόδοσης, γράφημα αίτιου-αποτελέσματος, έλεγχος λειτουργίας, τυχαίος έλεγχος, έλεγχος ασφάλειας, εικασία λάθους, διαχωρισμός ισοδυναμίας, έλεγχος επιχειρησιακής ετοιμότητας, έλεγχος λειτουργίας υπό ακραίες συνθήκες, έλεγχος αποδοχής από τον χρήστη, δοκιμή αναπαραγωγής, συμβατότητα/δοκιμή μετατροπής, έλεγχος ακεραιότητας δεδομένων, συγκριτική αξιολόγηση, δημιουργία αντιγράφων ασφαλείας και ανάκτησης δεδομένων, έλεγχος διαμόρφωσης, έλεγχος BETA
Προσωπικό	Προγραμματιστές, μηχανικοί ελέγχου	Προγραμματιστές, μηχανικοί ελέγχου, τελικοί χρήστες

4.2.3 ΕΛΕΓΧΟΣ ΓΚΡΙΖΟΥ ΚΟΥΤΙΟΥ

Ο έλεγχος μαύρου κουτιού εστιάζει στην λειτουργικότητα του προγράμματος σε σχέση με τις προδιαγραφές. Ο έλεγχος λευκού κουτιού επικεντρώνεται στις λογικές διαδρομές του κώδικα. Ο έλεγχος γκρίζου κουτιού (grey-box testing) είναι ο συνδυασμός των δύο παραπάνω μεθόδων ελέγχου. Σύμφωνα με την μέθοδο γκρίζου κουτιού, ο μηχανικός ελέγχου μελετά τις απαιτούμενες προδιαγραφές και επικοινωνεί με τον προγραμματιστή προκειμένου να κατανοήσει την εσωτερική δομή του συστήματος[55]. Το κίνητρο χρήσης της μεθόδου γκρίζου κουτιού είναι η απαλοιφή αμφισβητήσιμων προδιαγραφών και η κατανόηση του κώδικα, πίσω από τις προφανείς διαδικασίες, για να σχεδιαστούν χρήσιμες περιπτώσεις δοκιμών. Για παράδειγμα, όταν παρατηρείται πως μια συγκεκριμένη λειτουργία του συστήματος επαναχρησιμοποιείται καθ' όλη τη διάρκεια εκτέλεσης της εφαρμογής, τότε ο μηχανικός ελέγχου θα επικοινωνήσει με τον προγραμματιστή, θα κατανοήσει την εσωτερική δομή του κώδικα, με αποτέλεσμα να είναι σε θέση να απαλείψει πολλές δοκιμαστικές περιπτώσεις, αφού μια αρκεί για να εξεταστεί η προαναφερθείσα λειτουργία[54]. Ο έλεγχος γκρίζου κουτιού παρέχει συνδυασμένα τα οφέλη των ελέγχων μαύρου και του λευκού κουτιού, συνδυάζει δηλαδή την συμβολή των προγραμματιστών και των μηχανικών ελέγχου και έτσι βελτιώνει την συνολική ποιότητα των προϊόντων. Επιπλέον, μειώνει την επιβάρυνση από την μακρά διαδικασία ελέγχου λειτουργικών και μη λειτουργικών χαρακτηριστικών του λογισμικού. Τέλος, παρέχει αρκετό χρόνο στον προγραμματιστή να διορθώσει σφάλματα, ενώ ασφαλώς ο έλεγχος γίνεται ακολουθώντας την οπτική του τελικού χρήστη.

4.3 ΜΟΝΤΕΛΑ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ

Τα μοντέλα λογισμικού προτάθηκαν αρχικά για να φέρουν τάξη στο χάος που επικρατεί στη διαδικασία ανάπτυξης λογισμικού. Περιγράφουν και οργανώνουν ένα σύνολο από στοιχεία της διαδικασίας: δραστηριότητες πλαισίου, δράσεις των μηχανικών υπολογιστών, δραστηριότητες διασφάλισης ποιότητας, μηχανισμούς αλλαγής διαδικασιών για κάθε έργο. Κάθε μοντέλο καθορίζει επίσης την ροή εργασίας (workflow), δηλαδή τον τρόπο με τον οποίο συνδέονται τα στοιχεία της διαδικασίας παραγωγής μεταξύ τους[48]. Κάθε μοντέλο διαδικασίας ανάπτυξης λογισμικού είναι μια αφηρημένη αναπαράσταση ενός συνόλου διεργασιών. Παρουσιάζει και περιγράφει τον τρόπο με τον οποία θα γίνουν τα τέσσερα βασικά μέρη μιας διαδικασίας ανάπτυξης λογισμικού: α)Καθορισμός προδιαγραφών, β)Σχεδιασμός, γ)Επικύρωση, δ)Εξέλιξη. Τα βασικά μοντέλα περιγραφής των διαδικασιών ανάπτυξης λογισμικού είναι τέσσερα:

- Μοντέλο Καταρράκτη (Waterfall model)
- Μοντέλο σχήματος V (V-model)
- Επιθετικό μοντέλο (Agile model)
- Μοντέλο σπιράλ (Spiral model)

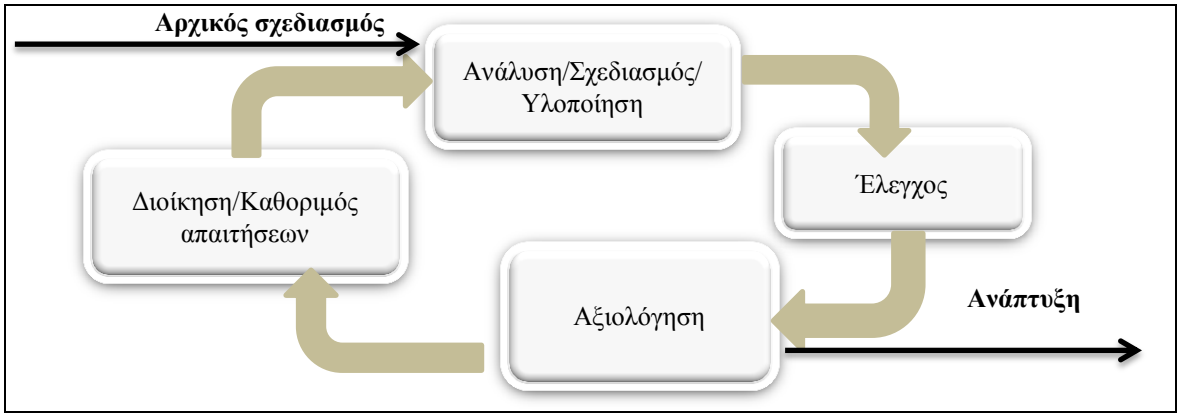
Τα παραπάνω μοντέλα επιλέχθηκαν να αναπτυχθούν καθώς τα χαρακτηριστικά τους αντιστοιχούν στα περισσότερα προγράμματα ανάπτυξης λογισμικού. Τα μοντέλα ανάπτυξης λογισμικού μπορούν να κατηγοριοποιηθούν σε τρεις βασικές ομάδες[12]:

- Διαδοχικό μοντέλο ανάπτυξης (Sequential model)
  Οι δραστηριότητες εκτελούνται διαδοχικά, η μια μετά την άλλη, με μοναδικό σκοπό την επίτευξη των προκαθορισμένων στόχων.
- Επαναληπτικό μοντέλο ανάπτυξης (Iterative development model)

Οι δραστηριότητες εκτελούνται επαναληπτικά έως ότου επιτευχθεί το απαιτούμενο επίπεδο ποιότητας. Τα μοντέλα αυτά χρησιμοποιούν εκτεταμένα τις δοκιμές παλινδρόμησης (regression testing). Στα επαναληπτικά μοντέλα ανάπτυξης, το έργο χωρίζεται σε μικρά κομμάτια. Αυτό επιτρέπει στην ομάδα ανάπτυξης να απεικονίζει τα αποτελέσματα νωρίτερα κατά τη διαδικασία και να λαμβάνει πολύτιμη ανατροφοδότηση από τους χρήστες του συστήματος. Κάθε ξεχωριστή επανάληψη των μοντέλων που ανήκουν σε αυτή τη κατηγορία, είναι μια εφαρμογή των διαδοχικών μοντέλων ανάπτυξης. Τα βήματα που ακολουθεί το επαναληπτικό μοντέλο ανάπτυξης φαίνονται στην *Εικόνα 9*.

- Βαθμιαία αυξανόμενο μοντέλο (Incremental model)
  Συνδυάζει τα δύο προηγούμενα μοντέλα.

Ο έλεγχος είναι πάντα παρών στο κύκλο ανάπτυξης λογισμικού και κάθε φορά υλοποιείται διαφορετικά, αναλόγως το μοντέλο ανάπτυξης. Ωστόσο, οι βασικές αρχές του, όπως αναλύθηκαν στις προηγούμενες δύο παραγράφους, ακολουθούνται πάντα.



Εικόνα 9: Επαναληπτικό μοντέλο ανάπτυξης

4.3.1 ΤΟ ΜΟΝΤΕΛΟ ΚΑΤΑΡΡΑΚΤΗ

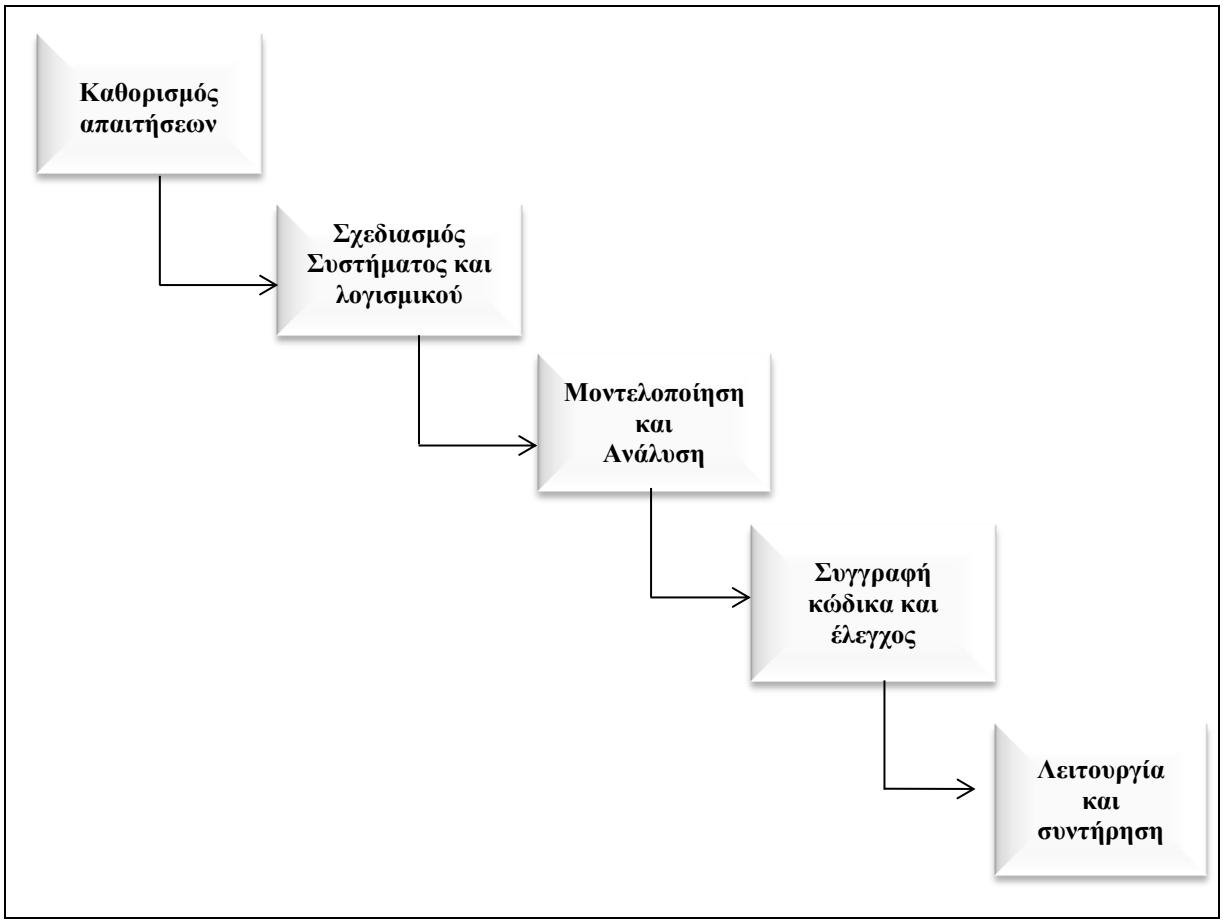
Το μοντέλο καταρράκτη είναι το κλασσικό μοντέλο της επιστήμης μηχανικών υπολογιστών. Ανήκει στην κατηγορία των διαδοχικών μοντέλων και χρησιμοποιείται ευρέως σε κυβερνητικά έργα και σε μεγάλες εταιρείες[33]. Το μοντέλο καταρράκτη δίνει έμφαση στην σχεδίαση των εργασιών σε πρώιμο στάδιο, εξασφαλίζει την εξάλειψη αρκετών σχεδιαστικών ελαττωμάτων πριν αυτά εμφανιστούν. Επιπλέον, το έγγραφο καταγραφής πλάνου και ο αναλυτικός σχεδιασμός του, το κάνουν ιδανικό για να εφαρμοστεί σε έργα λογισμικού όπου ο έλεγχος ποιότητας είναι από τις πιο σημαντικές διεργασίες. Το μοντέλο καταρράκτη, ή αλλιώς ο κλασσικός κύκλος ζωής λογισμικού, υποστηρίζει τη διαδοχική προσέγγιση στην ανάπτυξη λογισμικού, η οποία αρχίζει με τις απαιτήσεις του πελάτη, συνεχίζει με τον σχεδιασμό, την μοντελοποίηση, την κατασκευή, την ανάπτυξη και τέλος την συνεχή υποστήριξη του ολοκληρωμένου λογισμικού. Η παραπάνω διαδικασία φαίνεται στην *Εικόνα 10*. Παρακάτω περιγράφονται λεπτομερώς τα βήματα που ακολουθεί το μοντέλο:

1. Καθορισμός απαιτήσεων συστήματος
   Καθορίζει τα στοιχεία που απαιτούνται για την κατασκευή του συστήματος, συμπεριλαμβανομένων του υλικού, των εργαλείων λογισμικού, της γλώσσας προγραμματισμού, βάσεις δεδομένων κλπ.
2. Καθορισμός απαιτήσεων λογισμικού
   Προσδιορίζει τις προσδοκίες από την λειτουργία του λογισμικού και αναφέρει ποιες από τις απαιτήσεις συστήματος επηρεάζουν το λογισμικό.
3. Σχεδιασμός αρχιτεκτονικής
   Καθορίζει το πλαίσιο πάνω στο οποίο αναπτύσσεται το λογισμικό προκειμένου να ανταποκρίνεται στις προαναφερθείσες λειτουργίες. Σε αυτή τη φάση, ορίζονται τα συστατικά στοιχεία και αλληλεπίδραση μεταξύ τους, χωρίς να περιγράφεται η ακριβής εσωτερική δομή κάθε συστατικού. Οι διεπαφές και τα εργαλεία με την βοήθεια των οποίων εκτελείται το σύστημα λογισμικού, προσδιορίζονται σε αυτό το βήμα.
4. Λεπτομερής σχεδιασμός
   Εξετάζει τα συστατικά του λογισμικού που έχουν καθοριστεί στο βήμα 3 και επιλέγει τις κατάλληλες προδιαγραφές για να εφαρμοστεί κάθε ξεχωριστό συστατικό.
5. Κωδικοποίηση
   Εφαρμόζει τις λεπτομερείς προδιαγραφές σχεδιασμού μέσα από την συγγραφή κώδικα.
6. Έλεγχος

Εξετάζει εάν το λογισμικό πληροί τις προκαθορισμένες απαιτήσεις και εντοπίζει τυχόν σφάλματα που υπάρχουν στον κώδικα.

7. Συντήρηση
   Αντιμετωπίζει προβλήματα και αιτήματα βελτίωσης του συστήματος, μετά την κυκλοφορία του λογισμικού.

Σε μερικούς οργανισμούς, υπάρχει μια επιτροπή ελέγχου αλλαγής, που ελέγχει της ποιότητα του προϊόντος, εξετάζοντας κάθε αλλαγή που γίνεται στο στάδιο συντήρησης. Σε κάθε βήμα του μοντέλου, έγγραφα τεκμηρίωσης εξηγούν τα αντικείμενα και περιγράφουν τις απαιτήσεις κάθε φάσης του μοντέλου. Στο τέλος κάθε φάσης, συγγράφεται μια αναφορά που εξηγεί με επιχειρήματα εάν το έργο είναι σε θέση να συνεχίσει στο επόμενο στάδιο ανάπτυξης. Η μέθοδος του καταρράκτη δεν απαγορεύει την επιστροφή σε κάποιο από τα προηγούμενα βήματα, για παράδειγμα την επιστροφή από την φάση του σχεδιασμού στην φάση προσδιορισμού των απαιτήσεων. Ωστόσο, αυτό συνεπάγεται και τα ανάλογα κόστη[56], για παράδειγμα οι παραλήψεις που γίνονται στην φάση καθορισμού των απαιτήσεων, κοστίζουν αρκετά αν διορθωθούν αργότερα. Κάθε ολοκληρωμένη φάση προϋποθέτει επίσημη αναφορά και εκτεταμένο κείμενο τεκμηρίωσης. Επειδή η πραγματική ανάπτυξη του λογισμικού γίνεται στα τελευταία στάδια της διαδικασίας, τα αποτελέσματα λειτουργίας του λογισμικού δεν διαφαίνονται νωρίς. Επιπλέον, το μέγεθος της τεκμηρίωσης που απαιτεί κάθε φάση του μοντέλου προκειμένου να ολοκληρωθεί σωστά, γίνεται πολλές φορές υπερβολικό και μη ευέλικτο για την συνολική διαδικασία ανάπτυξης. Αν και το μοντέλο καταρράκτη έχει τις αδυναμίες του, αποτελεί πρότυπο για τα υπόλοιπα μοντέλα και αυτό γιατί δίνει έμφαση σε αρκούντως σημαντικά στάδια της διαδικασίας ανάπτυξης λογισμικού[57].



Εικόνα 10: Μοντέλο καταρράκτη

Τα πλεονεκτήματα του μοντέλου καταρράκτη είναι τα εξής:

- Είναι κατανοητό και εύκολα εφαρμόσιμο.
- Χρησιμοποιείται από τους περισσότερους και είναι ευρέως γνωστό.
- Ενθαρρύνει τον ακριβή προσδιορισμό των απαιτήσεων πριν τον σχεδιασμό και τον αναλυτικό σχεδιασμό πριν από την συγγραφή του κώδικα.

- Προϋποθέτει τη δημοσίευση εγγράφων τεκμηρίωσης.
- Βοηθά τις αδύναμες ομάδες να αναπτύσσουν ένα σωστό προϊόν λογισμικού.

Τα μειονεκτήματα του μοντέλου καταρράκτη είναι τα εξής:

- Πολλές φορές πρόκειται για ένα μοντέλο που εφαρμόζεται υπό ιδανικές συνθήκες, οι οποίες συχνά δεν ανταποκρίνονται στην πραγματικότητα.
- Δεν έχει επαναληπτικό χαρακτήρα.
- Δεν είναι πάντα εύκολο να προσδιορίζονται οι ακριβείς απαιτήσεις τόσο νωρίς.
- Το λογισμικό αναπτύσσεται αργά και δεν υπάρχει αρκετός χρόνος για τον εντοπισμό σοβαρών σφαλμάτων.
- Δεν ενσωματώνει στη διαδικασία ανάπτυξης της διαχείριση σφαλμάτων.
- Είναι δύσκολο και ακριβό να γίνουν αλλαγές στα έγγραφα.
- Απαιτεί σημαντικές διοικητικές δαπάνες, οι οποίες είναι δαπανηρές για μικρές ομάδες και έργα.

4.3.2 ΤΟ ΜΟΝΤΕΛΟ V

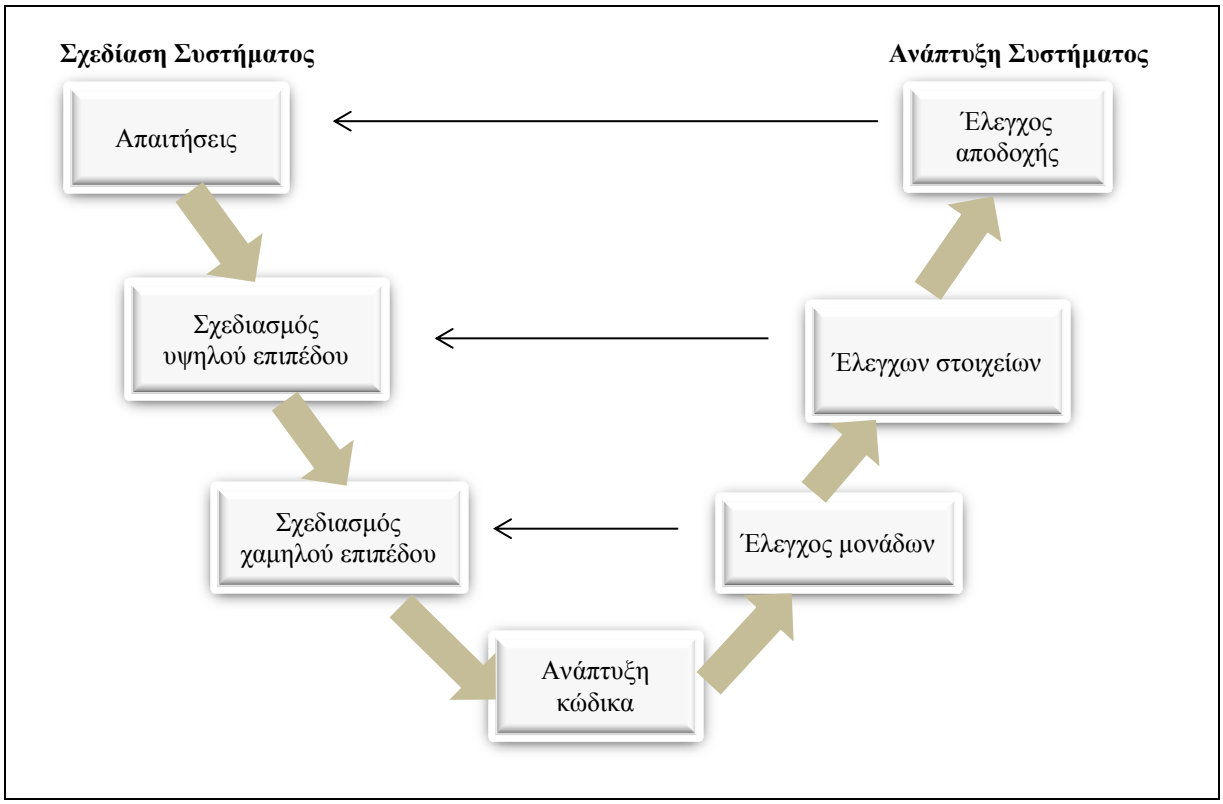
Μια παραλλαγή του μοντέλου ανάπτυξης καταρράκτη είναι το μοντέλο V. Το μοντέλο V απεικονίζει την σχέση μεταξύ του τμήματος διασφάλισης ποιότητας και της τμημάτων που δραστηριοποιούνται με την επικοινωνία, την μοντελοποίηση και τις μεθόδους ανάπτυξης. Όπως και το μοντέλο καταρράκτη, το μοντέλο V ανήκει στην κατηγορία των διαδοχικών μοντέλων ανάπτυξης. Κάθε φάση πρέπει να ολοκληρωθεί πριν ξεκινήσει η επόμενη. Στο συγκεκριμένο μοντέλο δίνεται περισσότερη έμφαση στον έλεγχο, για αυτό το λόγο οι διαδικασίες ελέγχου αναπτύσσονται στα πρώιμα στάδια του κύκλου ζωής του λογισμικού πριν οποιαδήποτε κωδικοποίηση[58]. Ο καθορισμός των απαιτήσεων είναι το πρώτο βήμα ανάπτυξης όπως και στο καταρράκτη. Πριν ξεκινήσει η ανάπτυξη κώδικα, οργανώνεται το σχέδιο ελέγχου του συστήματος, το οποίο επικεντρώνεται στην ικανοποίηση των απαιτήσεων όσον αφορά την λειτουργικότητα. Η φάση σχεδιασμού υψηλού επιπέδου επικεντρώνεται στην αρχιτεκτονική και το σχεδιασμό του συστήματος. Στη φάση αυτή, δημιουργείται και ο τρόπος ελέγχου της ολοκλήρωσης προκειμένου μετέπειτα να ελεγχθεί εάν τα ξεχωριστά τεμάχια του λογισμικού μπορούν να λειτουργήσουν μαζί. Στη φάση σχεδιασμού χαμηλού επιπέδου, το μοντέλο επικεντρώνεται στον σχεδιασμό των πραγματικών δομικών στοιχείων του λογισμικού, τις μονάδες. Συνεπώς, σε αυτό το βήμα σχεδιάζονται και οι τρόποι ελέγχου των συγκεκριμένων μονάδων. Η φάση της υλοποίησης εκτελείται όταν ο κώδικας είναι έτοιμος, οπότε και τα σχέδια ελέγχου που οργανώθηκαν νωρίτερα μπορούν να εφαρμοστούν, όπως φαίνεται στην Εικόνα 11. Τα πλεονεκτήματα του μοντέλου ανάπτυξης V είναι τα παρακάτω[56]:

- Εύχρηστο και εύκολο στην χρήση.
- Κάθε φάση έχει συγκεκριμένα παραδοτέα έγγραφα.
- Έχει μεγαλύτερη πιθανότητα επιτυχίας σε σύγκριση με το μοντέλο καταρράκτη, λόγω της παράλληλης ανάπτυξης σχεδίων ελέγχου με τη διαδικασία ανάπτυξης.
- Όταν εφαρμόζεται σε μικρά έργα λογισμικού, έχει υψηλή απόδοση.

Τα μειονεκτήματα του μοντέλου ανάπτυξης V είναι τα εξής:

- Δεν είναι ευέλικτο και έτσι είναι δαπανηρό.
- Το λογισμικό αναπτύσσεται αργά, κατά τη διάρκεια της φάσης υλοποίησης.
- Δεν παρέχει έναν τρόπο ανάκαμψης από τυχόν σφάλματα που θα εντοπιστούν κατά τη διάρκεια των ελέγχων.

Η γραμμική φύση του κλασσικού κύκλου ζωής του λογισμικού, την οποία ακολουθούν τα μοντέλα καταρράκτη και V, οδηγεί καταστάσεις δέσμευσης, όπου τα μέλη μιας ομάδας, περιμένουν τα μέλη μιας άλλης ομάδας να ολοκληρώσουν τις εργασίες τους, για να προχωρήσουν στα δικά τους καθήκοντα[59]. Μερικές φορές ο χρόνος αναμονής μπορεί να υπερβεί το χρόνο που αφιερώνεται στην πραγματική διαδικασία ανάπτυξης. Ειδικά σήμερα, που το λογισμικό δομείται γρήγορα και υπόκειται μια ατέρμονη ροή αλλαγών, τα διαδοχικά μοντέλα ανάπτυξης είναι σχεδόν μη εφαρμόσιμα. Ωστόσο χρησιμεύουν ως πρότυπο της διαδικασίας σε περιπτώσεις όπου αυτή πρέπει να ολοκληρωθεί σε συγκεκριμένο χρόνο, με γραμμικό τρόπο.



Εικόνα 11: Το μοντέλο V

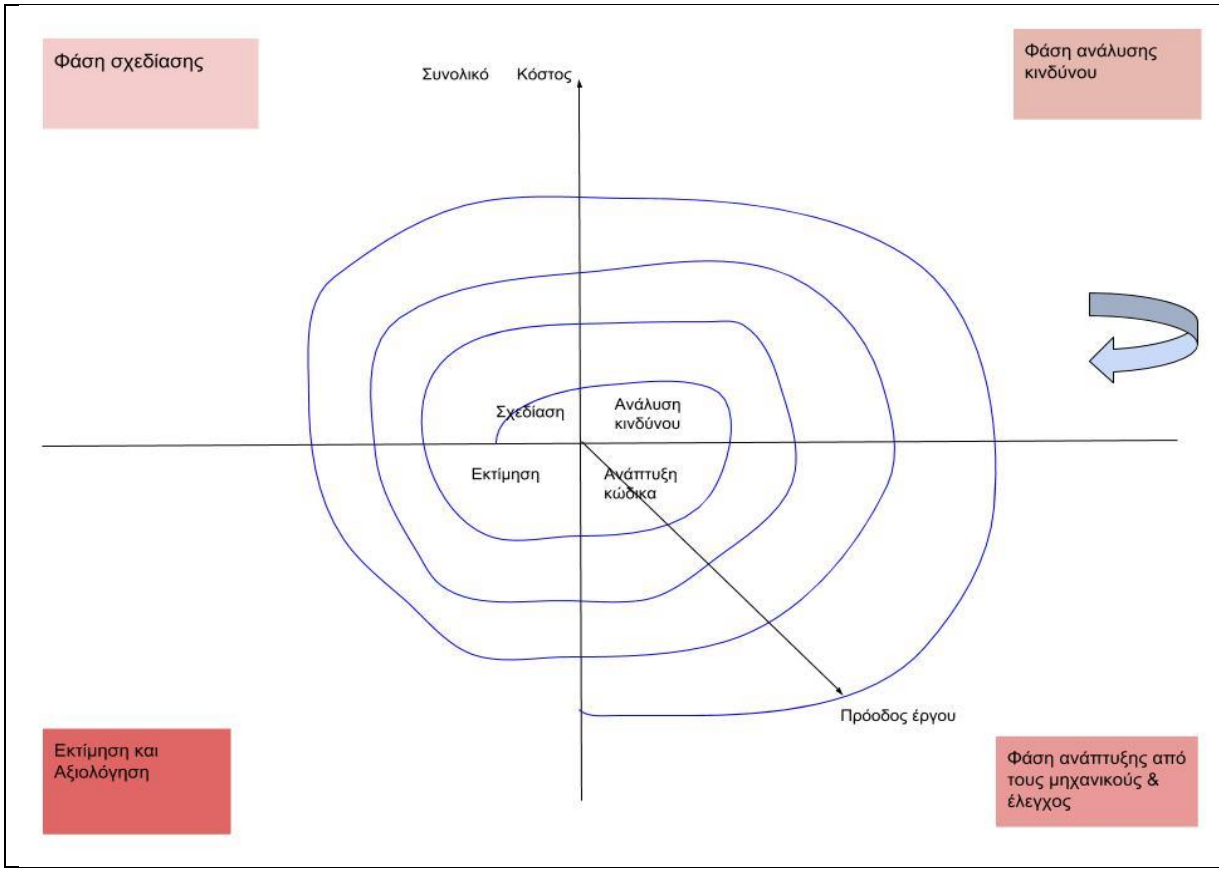
4.3.3 ΤΟ ΜΟΝΤΕΛΟ ΣΠΙΡΑΛ

Η ανάγκη ανάπτυξης του μοντέλου σπιράλ, δημιουργήθηκε παρατηρώντας την συμπεριφορά του μοντέλου καταρράκτη σε πολύ μεγάλα έργα με αυξημένες απαιτήσεις και σύντομες προθεσμίες. Το μοντέλο ανάπτυξης σπιράλ ανήκει στην κατηγορία των βαθμιαία αυξανόμενων μοντέλων και δίνει περισσότερη έμφαση στην ανάλυση κινδύνου. Ο κύκλος ζωής του μοντέλου αποτελείται από τέσσερις φάσεις: σχεδιασμός, ανάλυση κινδύνου, εργασία μηχανικών και εκτίμηση. Ένα έργο λογισμικού περνά επανειλημμένα από αυτές τις φάσεις. Κάθε ολοκληρωμένος κύκλος ζωής του μοντέλου, δηλαδή οι τέσσερις φάσεις, σχηματίζει και από ένα κομμάτι του σπιράλ. Όσες περισσότερες φορές χρησιμοποιείται το μοντέλο από το έργο λογισμικού τόσο μεγεθύνεται το σπιράλ που σχηματίζεται[60]. Το αρχικό σπιράλ, ξεκινά από την φάση του σχεδιασμού, συγκεντρώνει τις απαιτήσεις και αξιολογεί το ρίσκο. Κάθε επόμενο σπιράλ ξεκινά πάλι από την φάση ένα, τον σχεδιασμό. Στην φάση ανάλυσης κινδύνου, καταγράφονται και επεξεργάζονται οι κίνδυνοι και οι εναλλακτικές λύσεις. Ένα πρωτότυπο, πρώιμο σχέδιο ανάπτυξης παράγεται στο τέλος της φάσης ανάλυσης κινδύνου. Κατά την τρίτη φάση, οι μηχανικοί αναλαμβάνουν την παραγωγή του λογισμικού και τον έλεγχο αυτό με τις τεχνικές που έχουν προαναφερθεί παραπάνω. Στην τελευταία φάση αξιολόγησης, ο πελάτης αξιολογεί το έργο μέχρις ότου ξεκινήσει η διαδικασία του επόμενου σπιράλ. Στο μοντέλο σπιράλ, η γωνιακή συνιστώσα, η οποία κινείται με φορά από δεξιά προς αριστερά, αντιπροσωπεύει την πρόοδο του έργου, στον ανάλογο κύκλο σπιράλ. Η ακτίνα κάθε σπιράλ αντιπροσωπεύει το κόστος της μεθόδου[33]. Η διαδικασία που ακολουθεί το μοντέλο σπιράλ απεικονίζεται στην Εικόνα 12. Τα πλεονεκτήματα της μεθόδου είναι[61]:

- Υψηλή ανάλυση κινδύνου.
- Είναι ιδανικό για μεγάλα και σημαντικά έργα.
- Το λογισμικό παράγεται νωρίς, στις αρχές του κύκλου ζωής του.

Τα μειονεκτήματα του μοντέλου σπιράλ είναι τα παρακάτω:

- Είναι ακριβό.
- Η ανάλυση κινδύνου προϋποθέτει ειδική εμπειρογνωμοσύνη.
- Η επιτυχία του έργου εξαρτάται σε μεγάλο βαθμό από την φάση ανάλυσης κινδύνου.
- Δεν είναι αποδοτικό όταν εφαρμόζεται σε μικρότερα έργα λογισμικού.



Εικόνα 12: Το μοντέλο Σπιράλ

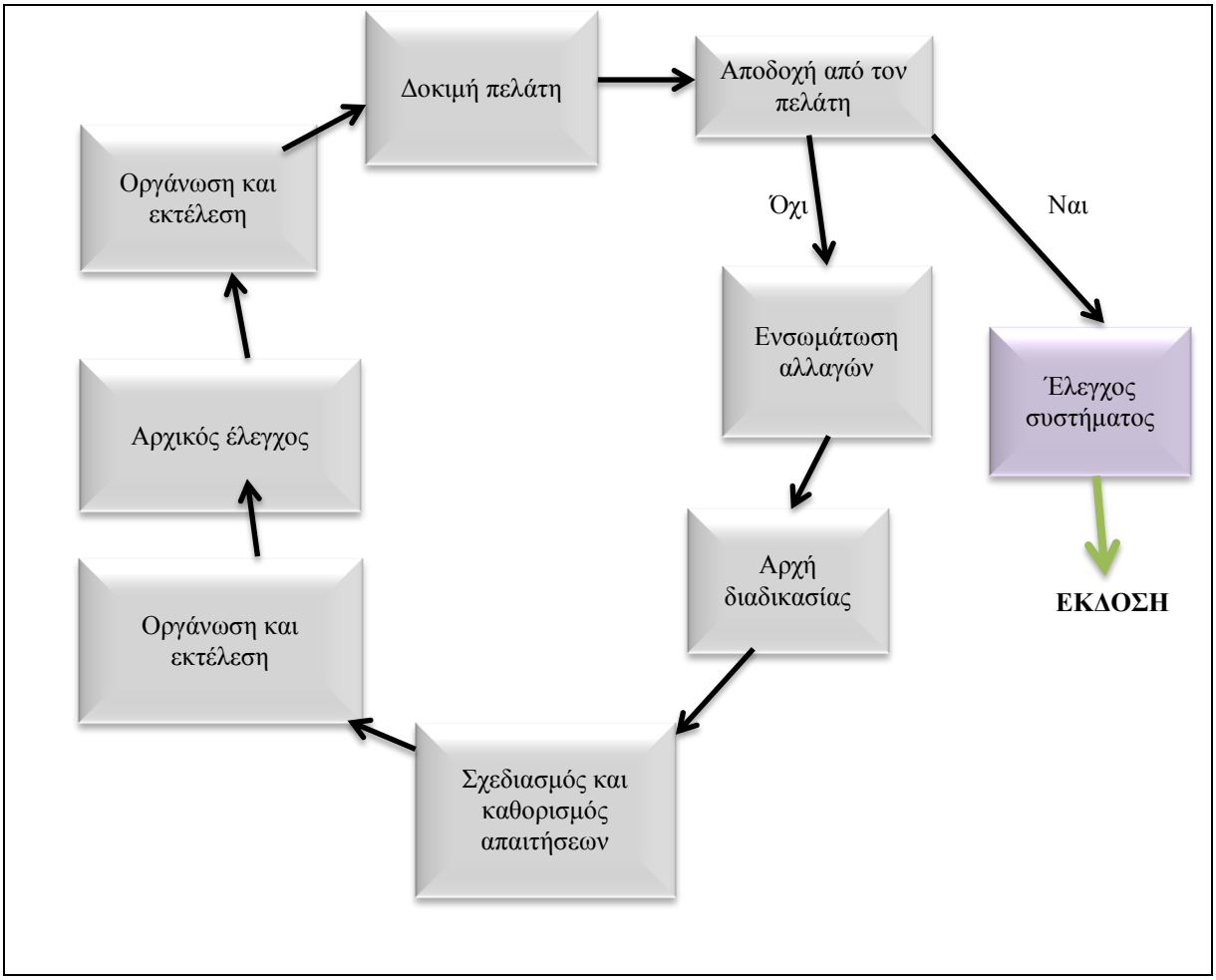
4.3.4 ΤΟ ΕΠΙΘΕΤΙΚΟ ΜΟΝΤΕΛΟ ΑΝΑΠΤΥΞΗΣ

Το επιθετικό μοντέλο προσεγγίζει την ανάπτυξη από την σκοπιά του προγραμματισμού και βασίζεται στην συνεχή βελτίωση του κώδικα και την συμμετοχή των τελικών χρηστών στην ομάδα ανάπτυξης. Η μέθοδος αυτή βοηθά τις ομάδες ανάπτυξης να ανταποκριθούν στις ξαφνικές αλλαγές που πιθανόν να γίνουν στον κώδικα. Χαρακτηρίζεται ως ένα βαθμιαία αυξανόμενο μοντέλο που χρησιμοποιεί επαναληπτικές μεθόδους εργασίας (sprints)[21]. Το επιθετικό μοντέλο ανάπτυξης, είναι ένα πακέτο που καλύπτει διάφορους τομείς της παραγωγικής διαδικασίας, επιτρέποντας στις ομάδες να ανταποκρίνεται άμεσα στις μεταβαλλόμενες απαιτήσεις και στον απρόβλεπτη ανάδραση (feedback) από τον πελάτη. Η ανάγκη δημιουργίας του επιθετικού μοντέλου, προέκυψε από το γεγονός πως οι κλασσικές διαδοχικές μέθοδοι ανάπτυξης δεν μπορούσαν να συμβαδίσουν με τις μεταβαλλόμενες απαιτήσεις και προτεραιότητες της παραγωγικής διαδικασίας. Το επιθετικό μοντέλο ανάπτυξης άρχισε να εμφανίζεται στις αρχές της δεκαετίας του 1990 ταυτόχρονα με την ραγδαία εμφάνιση όλο και περισσότερων εταιριών λογισμικού[62]. Το μανιφέστο που ορίζει και περιγράφει το επιθετικό μοντέλο ανάπτυξης και υπογράφηκε από πλήθος μηχανικών τον Φεβρουάριο του 2001[63], περιγράφει τις τέσσερις βασικές αξίες που θα πρέπει να ακολουθεί κάθε ομάδα ανάπτυξης που ακολουθεί το νέο επιθετικό μοντέλο:

- Οι αλληλεπιδράσεις των ατόμων με τις διαδικασίες και τα εργαλεία.
- Πλήρη τεκμηρίωση των λειτουργιών του λογισμικού.
- Συνεργασία με τους πελάτες κατά τη διαπραγμάτευση των συμβάσεων.
- Άμεση ανταπόκριση στην αλλαγή ακολουθώντας συγκεκριμένο σχέδιο.

Οι ρόλοι των διαφορετικών ομάδων ανάπτυξης κατά την μέθοδο του επιθετικού μοντέλου διαφέρουν. Η διαδικασία ξεκινά πάντα με τον καθορισμό των χρηστών και την συγγραφή αναλυτικών εγγράφων τεκμηρίωσης που περιλαμβάνουν το εύρος των προβλημάτων που πρέπει να αντιμετωπιστούν και τις ευκαιρίες που προκύπτουν. Στην συνέχεια, η εταιρεία καταγράφει το πλάνο ανάπτυξης και προσπαθεί να το υλοποιήσει. Πιο συγκεκριμένα[64], το επιθετικό μοντέλο ανάπτυξης λαμβάνει υπόψιν τα διαφορετικά είδη αναγκών και συμπεριφορών κάθε πελάτη, προκειμένου να προγραμματίσει και να καθορίσει τους διαφορετικούς ρόλους κάθε ομάδας κατά τη διάρκεια του κύκλου ζωής του. Η διαδικασία ξεκινά με κάποιον που αντιπροσωπεύει τον πελάτη και εκφράζει την άποψη του συνυπολογίζοντας τις κριτικές και την ανατροφοδότηση που έχει δεχτεί η εταιρεία μέχρι εκείνη την στιγμή. Έτσι, δημιουργείται ένα πρότυπο προϊόντος, από το οποίο η ομάδα ανάπτυξης

μπορεί να βγάλει χρήσιμα συμπεράσματα και να σχεδιάσει μια στρατηγική σε αρχικό στάδιο. Το άτομο που παίρνει τον ρόλο του πελάτη, περιγράφει το πρόβλημα και τι θα ήθελε αυτός να έχει το καινούριο προϊόν λογισμικού. Η ομάδα ανάπτυξης λογισμικού του επιθετικού μοντέλου ανάπτυξης είναι διεπιστημονικές, αποτελούνται από άτομα με διαφορετικές δεξιότητες που αναλαμβάνουν την ανάπτυξη του προϊόντος από την πιο μικρή λεπτομέρεια έως την πιο πολύπλοκη λειτουργία. Συνεπώς διαφορετικές ομάδες αναλαμβάνουν ταυτόχρονα το χτίσιμο της βάσης δεδομένων, τη διοίκηση, τη δημιουργία της διεπαφής χρήστη, τον έλεγχο κλπ. Οι παραπάνω διαφορετικές ομάδες ανάπτυξης, πρέπει να συζητούν συχνά για να βεβαιωθούν πως όλοι έχουν κατανοήσει επακριβώς τον ρόλο του καθενός και την τρόπο ανάπτυξης του προϊόντος λογισμικού. Εκτός από τους προγραμματιστές, οι ομάδες ανάπτυξης περιλαμβάνουν και τους μηχανικούς διασφάλισης ποιότητας, μηχανικούς με διαφορετικό αντικείμενο εργασίας (όπως η δημιουργία βάσης δεδομένων), σχεδιαστές και αναλυτές. Ο κύκλος ζωής του επιθετικού μοντέλου ανάπτυξης φαίνεται στην Εικόνα 13.



Εικόνα 13: Επιθετικό μοντέλο ανάπτυξης

Τα πλεονεκτήματα του επιθετικού μοντέλου ανάπτυξης είναι τα εξής[61]:

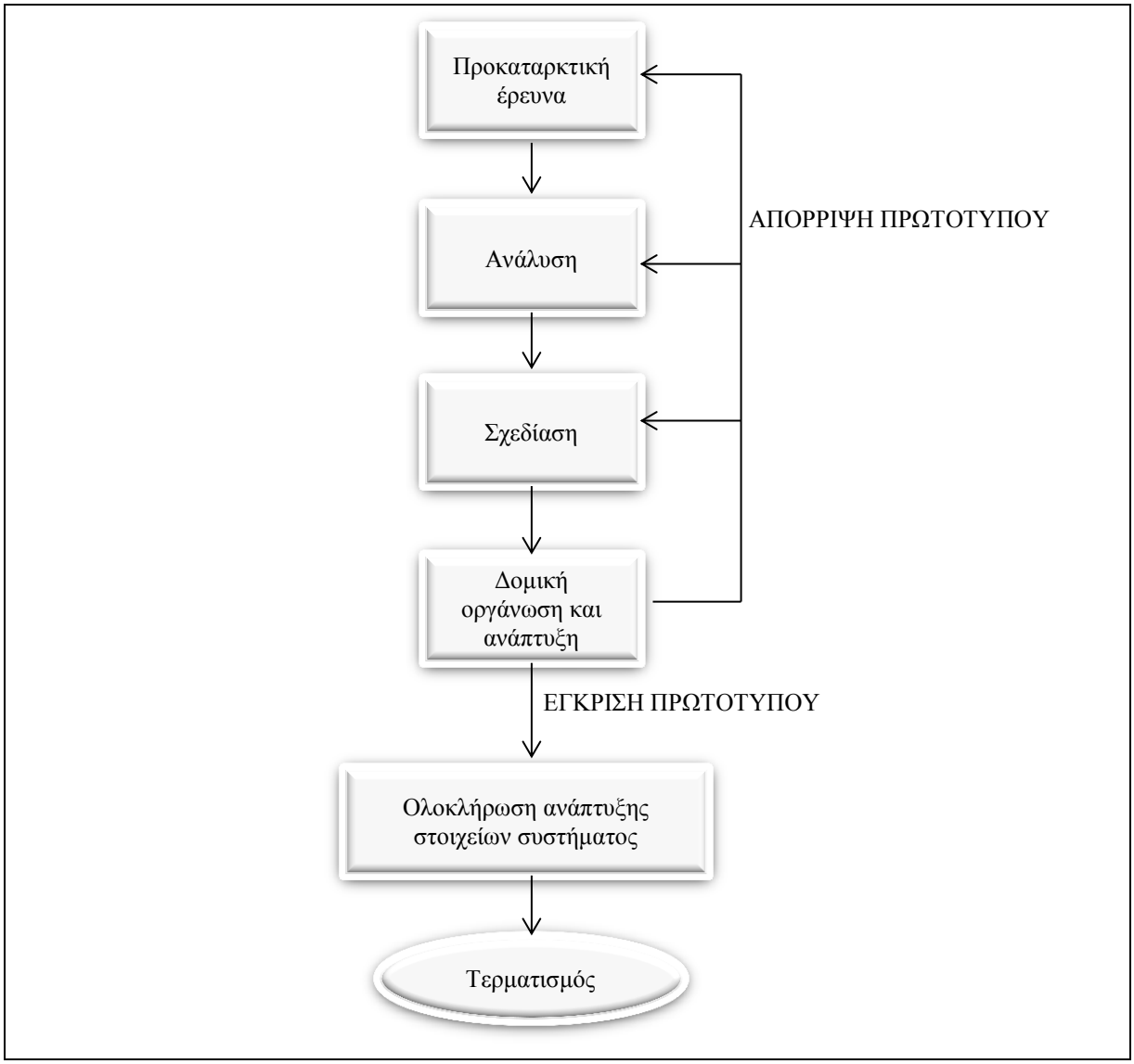
- Λειτουργεί αρκετά καλά σε έργα μικρού και μεσαίου μεγέθους.
- Ενισχύει την συνοχή των ομάδων.
- Δίνει έμφαση στο τελικό προϊόν καθ' όλη τη διάρκεια ζωής του.
- Είναι επαναληπτικό, συνεπώς ενθαρρύνει την συνεχή βελτίωση.
- Περιέχει περιπτώσεις δοκιμών που επικεντρώνονται στις απαιτήσεις του πελάτη και στην ποιότητα

Τα μειονεκτήματα της μεθόδου είναι τα παρακάτω:

- Είναι δύσκολο να εφαρμοστεί σε έργα μεγάλου μεγέθους, όπου η τεκμηρίωση είναι απαραίτητη.
- Απαιτεί εμπειρία και ικανότητες προκειμένου να εφαρμοστεί σωστά.
- Η δημιουργία δοκιμαστικών περιπτώσεων είναι δύσκολη και ακριβή διαδικασία.

4.3.5 ΑΝΑΠΤΥΞΗ ΜΕ ΔΗΜΙΟΥΡΓΙΑ ΠΡΩΤΟΤΥΠΩΝ

Μια καλύτερη προσέγγιση της διαδικασίας ανάπτυξης λογισμικού, όπου οι ενδιάμεσες αλλαγές γίνεται κανόνας και όχι εξαίρεση είναι η δημιουργία πρωτοτύπων. Σύμφωνα με αυτήν, η ανάπτυξη ενός συστήματος λογισμικού περνά μέσα από σειρά επαναληπτικών μεθόδων, μέχρις ότου το καλυφθεί το σύνολο των απαιτήσεων[65]. Στην Εικόνα 14 φαίνονται τα στάδια που ακολουθεί το μοντέλο ανάπτυξης λογισμικού σύμφωνα με την μέθοδο των πρωτοτύπων. Τα πρώτα τέσσερα στάδια επαναλαμβάνονται έως ότου ο χρήστης αποδεχτεί το πρωτότυπο. Τα ονόματα διαφέρουν σε σύγκριση με την κλασσικό κύκλο ζωής ανάπτυξης, αλλά η λογική είναι η ίδια. Το μοντέλο ξεκινά με τον σχεδιασμό και καταλήγει στην εφαρμογή. Σε ορισμένα έργα λογισμικού, ο χρήστης δεν είναι σε θέση να προσδιορίσει ακριβώς πότε και πως το σύστημα επιτυγχάνει. Σε αυτές τις περιπτώσεις, οι μηχανικοί ελέγχου χρησιμοποιούν πρωτότυπα, τα οποία προσομοιώνουν τις ανάγκες του χρήστη. Αυτό το είδος πρωτοτύπου ονομάζεται πρωτότυπο απαιτήσεων[66]. Το πρωτότυπο είναι συνήθως ένα μη λειτουργικό κέλυφος του προβλεπόμενου και προσχεδιασμένου συστήματος. Χρησιμεύει ως υπόδειγμα για την κλασσικό κύκλο ζωής ανάπτυξης λογισμικού και χρησιμοποιείται σε πολλά έργα ανάπτυξης όπως στην ανάπτυξη λογισμικών διαχείρισης δεδομένων. Ένα πρωτότυπο είναι μια αρχική προσέγγιση του τελικού συστήματος λογισμικού ή προϊόντος που αναπτύσσεται, ελέγχεται και επανασχεδιάζεται όσες φορές χρειαστεί έως ότου συγκεντρωθούν όλες οι απαραίτητες πληροφορίες για να ξεκινήσει η ανάπτυξη του ολοκληρωμένου τελικού προϊόντος.



Εικόνα 14: Ο κύκλος ζωής ανάπτυξης λογισμικού με τη δημιουργία πρωτοτύπων

Το πρωτότυπο προϊόντος συνήθως εκτελεί τις βασικές λειτουργίες και καλύπτει τις βασικές γνωστές απαιτήσεις του πελάτη. Όσες περισσότερες φορές επανασχεδιάζεται και αναπτύσσεται τόσο πιο πολύ επιτρέπει στους

μηχανικούς ανάπτυξης να προσδιορίζουν όλες ή σχεδόν όλες τις ανάγκες του τελικού χρήστη. Η συγκεκριμένη μέθοδος ανάπτυξης είναι ιδανική για μεγάλα και πολύπλοκα συστήματα, όπου δεν προϋπάρχει κάποιο παρόμοιο σύστημα που να καθορίζει τις απαιτήσεις και ανάγκες των πελατών. Η ανάπτυξη λογισμικού με δημιουργία πρωτοτύπων χρησιμοποιείται σε συστήματα που απαιτούν συνεχή επικοινωνία με τους τελικούς χρήστες καθ' όλη τη διάρκεια του κύκλου ζωής ανάπτυξής τους. Για παράδειγμα, εφαρμογές και ιστοσελίδες χρησιμοποιούν την συγκεκριμένη μέθοδο ανάπτυξης[67].

5

ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΣ ΕΛΕΓΧΟΣ ΛΟΓΙΣΜΙΚΟΥ

Στην σύγχρονη εποχή, οι διαχειριστές έργων λογισμικού και οι προγραμματιστές βρίσκονται αντιμέτωποι καθημερινά με προκλήσεις και προβλήματα που θα πρέπει να λύσουν σύντομα και με ελάχιστους διαθέσιμους πόρους. Τα χρονοδιαγράμματα συρρικνώνονται συνεχώς, ενώ ο ανταγωνισμός είναι τεράστιος. Προκειμένου λοιπόν να ανταποκρίνονται στους ταχείς ρυθμούς εξέλιξης και τις συνεχώς αυξανόμενες απαιτήσεις των πελατών, οι οργανισμοί θέτουν ως έναν από τους κύριους στόχους τους, τον σύντομο αλλά ταυτοχρόνως περιεκτικό έλεγχο των προϊόντων τους. Αυτό μπορεί να επιτευχθεί μέσω της αυτοματοποίησης μέρους των ελέγχων, δεδομένου πως δεν υπάρχει πλέον η χρονική πολυτέλεια πολλών δοκιμών με το χέρι, όπως η προσομοίωση εκτέλεσης μιας εφαρμογής από 1000 εικονικούς χρήστες[13]. Η δυνατότητα αυτοματοποίησης όλο και περισσότερων δοκιμών συμβαδίζει με τη συνεχώς αυξανόμενη δημοτικότητα της μεθόδου ταχείας ανάπτυξης εφαρμογών (Rapid Application Development –RAD), η οποία επικεντρώνεται στην ελαχιστοποίηση του χρονοδιαγράμματος ανάπτυξης, παρέχοντας ταυτόχρονα όλο και μεγαλύτερο αριθμό λογισμικών. Ο στόχος της μεθόδου ταχείας ανάπτυξης εφαρμογών είναι να εμπλακεί ο τελικός χρήστης, όσο πιο νωρίς γίνεται, στον σχεδιασμό και στην ανάπτυξη κάθε έκδοσης λογισμικού ούτως ώστε να διασφαλιστεί πως οι λειτουργίες του συμβαδίζουν με τις προτιμήσεις του χρήστη και ικανοποιούν τις ανάγκες του. Σε ένα τέτοιο περιβάλλον, όπου οι καθημερινές αλλαγές και προσθήκες στο λογισμικό κρίνονται απαραίτητες και ο καθορισμός των απαιτήσεων κρίνουν την ανάπτυξή του, ο έλεγχος λαμβάνει εξορισμού επαναληπτικό χαρακτήρα. Κάθε νέα έκδοση του λογισμικού, συνοδεύεται από πλήθος νέων ελέγχων άλλα και επανάληψη παλαιών περιπτώσεων δοκιμών προσαρμοσμένων στα νέα δεδομένα. Έτσι λοιπόν, η αυτοματοποίηση του ελέγχου πλέον κρίνεται απαραίτητη και μάλιστα οι μηχανικοί λογισμικού προσπαθούν να την εντάξουν σε όσο το δυνατόν περισσότερα βήματα του κύκλου ζωής ελέγχου.

5.1 ΚΥΚΛΟΣ ΖΩΗΣ ΤΟΥ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΥ ΕΛΕΓΧΟΥ

Η χρήση εργαλείων αυτοματοποιημένων ελέγχων για την στήριξη της διαδικασίας δοκιμών αποδεικνύεται ωφέλιμη και κρίνεται απαραίτητη προκειμένου να επιτευχθεί η μέγιστη δυνατή ποιότητα προϊόντος στον ελάχιστο δυνατό χρόνο[19]. Για την καλύτερη δυνατή οργάνωση και εκτέλεση της αυτοματοποίησης, είναι αναγκαίος ο σχεδιασμός και ο προγραμματισμός των αυτοματοποιημένων ελέγχων όσο πιο νωρίς γίνεται. Δηλαδή, ταυτόχρονα με τον καθορισμό των απαιτήσεων και το σχεδιασμό πλάνων για την ανάπτυξη του προϊόντος θα πρέπει, από την πρώτη μέρα, να καθορίζονται οι τρόποι και τα εργαλεία αυτοματοποίησης, οι μηχανικοί που θα αναλάβουν τους συγκεκριμένους ελέγχους, καθώς και τα όρια της αυτοματοποίησης. Το τελευταίο μέρος ανάλυσης ειδικά θα πρέπει να είναι σαφές από την αρχή σε όλους, για να μην υπάρχουν παρεκκλίσεις και απομάκρυνση από τα χρονικά όρια του ελέγχου. Η αυτοματοποίηση δεν αντικαθιστά τους χειρωνακτικούς ελέγχους και μερικοί έλεγχοι είναι πιο αποτελεσματικοί όταν γίνονται από κάποιον άνθρωπο. Ωστόσο, με τις κατάλληλες τεχνικές, ο αυτοματοποιημένος έλεγχος μπορεί να αυξήσει σημαντικά την ποιότητα και την παραγωγικότητα των μεθόδων ελέγχου.

Τα εργαλεία ελέγχου δεν περιέχουν μαγεία -αλλά, με την κατάλληλη οργάνωση, μπορούν να κάνουν θαύματα.

Hayes,1995[68]

Η εμπλοκή των διαδικασιών αυτοματοποιημένων ελέγχων νωρίς κατά τη διαδικασία ανάπτυξης είναι σημαντική διότι οι λειτουργικές προδιαγραφές του συστήματος ή της εφαρμογής ενδέχεται να αναθεωρηθούν από την ομάδα ελέγχων. Όταν ένας οργανισμός δεν έχει εντάξει πότε ξανά την αυτοματοποίηση στις διαδικασίες ελέγχου θα περάσει κατά κανόνα από πέντε επίπεδα οργάνωσης της αυτοματοποίησης[13]. Στο αρχικό στάδιο, ένας οργανισμός, εφόσον δεν έχει ξανά εφαρμόσει ποτέ την αυτοματοποίηση, θα πρέπει να χρησιμοποιήσει τα

εργαλεία αυτοματοποιημένου ελέγχου πειραματικά. Οι κώδικες που αυτοματοποιούν τον έλεγχο (scripts) καταγράφονται για πρώτη φορά και εκτελούνται υπό συγκεκριμένες συνθήκες και για στοχευμένους ελέγχους. Συνήθως, τα πρώτα κομμάτια κώδικα που χρησιμοποιούνται για αυτοματοποίηση, δεν είναι σε θέση να επαναχρησιμοποιηθούν, έστω και με κάποια τροποποίηση γιατί δεν ακολουθούν κανένα πρότυπο σχεδιασμού και ανάπτυξης, ειδικά εάν αναπτύσσονται από κάποιον άπειρο μηχανικό. Ωστόσο, στο δεύτερο επίπεδο αυτοματοποίησης, ο έλεγχος γίνεται μια ρουτίνα λόγω της πείρας που οι μηχανικοί ελέγχου έχουν κερδίσει από τα λάθη τους κατά την προσπάθεια τους να αυτοματοποιήσουν κάποια είδη δοκιμών[69]. Στο επίπεδο αυτό, τα κομμάτια κώδικα αυτοματοποίησης είναι σε θέση να τεθούν υπο επεξεργασία αλλά δεν ακολουθούν κάποιο πρότυπο τεκμηρίωσης. Επιπλέον, τα χρονοδιαγράμματα και οι απαιτήσεις ελέγχου δεν ικανοποιούνται και η αυτοματοποίηση σε αυτό το επίπεδο μπορεί να φέρει τα αντίθετα αποτελέσματα, δηλαδή την σπατάλη χρόνου και πόρων. Τα εργαλεία που χρησιμοποιούνται σε αυτό το στάδιο είναι εργαλεία σχεδίασης έργου, προσομοιωτές, μεταγλωττιστές κώδικα. Στο τρίτο επίπεδο, οι διαδικασίες αυτοματοποιημένου ελέγχου είναι πλέον καλά δομημένες και οργανωμένες. Οι απαιτήσεις δοκιμών και ο τρόπος συγγραφής κομματιών κώδικα που ελέγχουν αυτοματοποιημένα κάποιες από τις λειτουργίες του λογισμικού, προκύπτουν εύκολα από τον αρχικό σχεδιασμό απαιτήσεων του λογισμικού. Ο κώδικας αυτοματοποίησης βασίζεται σε συγκεκριμένα πρότυπα σχεδιασμού και ανάπτυξης, ενώ οι περιπτώσεις δοκιμών ελέγχου μπορούν να επαναχρησιμοποιηθούν και να συντηρηθούν. Ωστόσο, οι ομάδες ελέγχου, δεν μπορούν να ελέγξουν και να αξιολογήσουν τις διαδικασίες αυτοματοποιημένου ελέγχου. Στο τέταρτο επίπεδο, ο οργανισμός είναι σε θέση να εντάξει τον αυτοματοποιημένο έλεγχο σχεδόν σε όλες τις φάσεις ανάπτυξης λογισμικού. Πλέον η ομάδα αυτοματοποιημένου ελέγχου αξιολογεί και καταγράφει τα λάθη που εντόπισε και τα μεταφέρει πίσω στην ομάδα ανάπτυξης, μαζί με ένα έγγραφο τεκμηρίωσης. Έτσι, πλέον η αυτοματοποίηση ξεκινά νωρίς, ούτως ώστε να είναι λιγότερο δαπανηρή η διόρθωση πιθανών σφαλμάτων. Σε αυτό το επίπεδο, η ομάδα αυτοματοποίησης ελέγχου χρησιμοποιεί και εργαλεία πρόληψης και παρακολούθησης, εργαλεία δημιουργίας νέων ειδών ελέγχων καθώς και εργαλεία αξιολόγησης κώδικα. Στο τελευταίο στάδιο οργάνωσης της αυτοματοποίησης ελέγχων, η διαδικασία έχει πλέον οργανωθεί και έχει χωριστεί σε έξι βασικά βήματα:

1. Καθορισμός ορίων αυτοματοποίησης.
2. Συγκέντρωση εργαλείων αυτοματοποιημένου ελέγχου.
3. Εισαγωγή στη διαδικασία αυτοματοποιημένου ελέγχου.
4. Οργάνωση, σχεδίαση και ανάπτυξη των αυτοματοποιημένων δοκιμών.
5. Εκτέλεση και διαχείριση των αυτοματοποιημένων δοκιμών.
6. Αναθεώρηση και αξιολόγηση της διαδικασίας.

Η παραπάνω ολοκληρωμένη μεθοδολογία, αναδεικνύει εκείνες τις δραστηριότητες ελέγχου που μπορούν να αυτοματοποιηθούν καθώς και το κατάλληλο εργαλείο αυτοματοποίησης για κάθε περίπτωση. Επίσης, καθορίζει και διαχειρίζεται τα δεδομένα που τίθενται υπό δοκιμή, το περιβάλλον ελέγχου καθώς και τον ακριβή τρόπο ανάπτυξης των εγγράφων τεκμηρίωσης[70]. Τα εργαλεία που χρησιμοποιούνται σε αυτό το επίπεδο, είναι όλα τα προαναφερθέντα, αλλά και εργαλεία παραγωγής τυχαίων δεδομένων εισόδου στους ελέγχους. Ακόμη, στο τελικό στάδιο οργάνωσης της αυτοματοποίησης, ο οργανισμός διαθέτει εργαλεία συλλογής και υπολογισμού μετρικών ποιότητας και λογισμικά στατιστικής ανάλυσης των αποτελεσμάτων που εξάγονται από τους αυτοματοποιημένους ελέγχους.

5.1.1 ΣΤΡΑΤΗΓΙΚΗ ΑΥΤΟΜΑΤΟΠΟΙΗΣΗΣ

Η αυτοματοποίηση είναι μια συνεχόμενη διαδικασία που δεν πρόκειται να ολοκληρωθεί ποτέ και ούτε θα πρέπει να σταματήσει ποτέ. Είναι μια διαδικασία ελέγχου που εξελίσσεται και βελτιώνεται ανάλογα τις συνθήκες, τις ανάγκες του οργανισμού. Παρακάτω καταγράφονται τα βασικά βήματα που θα πρέπει να ακολουθεί μια ομάδα ελέγχου για να πετύχει την βέλτιστη στρατηγική αυτοματοποιημένου ελέγχου[18]:

1. Συγγραφή λεπτομερούς πλάνου ελέγχου πριν ξεκινήσει το οτιδήποτε. Θα πρέπει να είναι ξεκάθαρη η στρατηγική ελέγχου τόσο προς την ομάδα ελέγχου όσο και στους διαχειριστές των έργων λογισμικού.
2. Σχεδιασμός ενός κοινού πλαισίου διαχείρισης δοκιμαστικών περιπτώσεων, έτσι ώστε κάθε μηχανικός ελέγχου να οργανώνει τα είδη δοκιμών που του ανατίθενται σύμφωνα με τα προσυμφωνημένα πρότυπα.
3. Μείωση των μονάδων συγγραφής και συντήρησης κώδικα, έτσι ώστε όλοι να χρησιμοποιούν κοινά προγράμματα ανάπτυξης αυτοματοποιημένων ελέγχων.
4. Καταγραφή αρχείων ελέγχων (test logs) καθώς επίσης και συγγραφή πλήρους αναφοράς για τη διαδικασία, τα προγράμματα, τον κώδικα, τις εισόδους και τα τελικά αποτελέσματα.
5. Συγγραφή δοκιμών που έχουν τη δυνατότητα να τρέξουν χωρίς επίβλεψη και να ανακάμψουν από πιθανή αποτυχία.

6. Συγγραφή δοκιμών σε πολλές διαφορετικές γλώσσες προγραμματισμού και εκτέλεση αυτών σε διαφορετικές πλατφόρμες, με διαφορετικές ρυθμίσεις.
7. Εισαγωγή τυχαίων δεδομένων εισόδου στους κώδικες αυτοματοποιημένου ελέγχου.
8. Έναρξη διαδικασίας με απλές δοκιμές που όμως είναι επιτυχείς.
9. Μέτρηση αποτελεσματικότητας της διαδικασίας αυτοματοποίησης ανάλογα το πλήθος και το ποσοστό σφαλμάτων που εντόπισε.
10. Χρήση αυτοματοποίησης για προσομοιώσεις ακραίων καταστάσεων λειτουργίας.
11. Εκτέλεση δοκιμών έως ότου το λογισμικό αποτύχει.

5.2 ΒΑΣΙΚΕΣ ΑΡΧΕΣ ΤΗΣ ΑΥΤΟΜΑΤΟΠΟΙΗΣΗΣ

Δεδομένου πως ο έλεγχος λογισμικού είναι μια ακριβή και απαιτητική διαδικασία, είναι θεμιτό μέρος αυτού να αυτοματοποιείται. Οι οργανισμοί που επιλέγουν την αυτοματοποίηση σαν τακτική ελέγχου θα πρέπει να γνωρίζουν τα πραγματικά οφέλη αλλά και τα κόστη μιας τέτοιας διαδικασίας. Η αυτοματοποίηση ελέγχου λογισμικού έχει τρία βασικά πλεονεκτήματα[68]:

- Οι αυτοματοποιημένοι έλεγχοι μπορούν να εκτελεστούν πολλές φορές, προσφέροντας κάθε φορά έγκυρα αποτελέσματα. Συνεπώς με την αυτοματοποίηση ένας οργανισμός εξοικονομεί χρόνο με την προϋπόθεση πως όλες οι δοκιμές εκτελούνται χωρίς της επέμβαση ανθρώπου.
- Οι μηχανικοί ελέγχου έχουν τη δυνατότητα, χρησιμοποιώντας τα εργαλεία αυτοματοποιημένου ελέγχου, να εκτελέσουν δοκιμές που θα ήταν αδύνατο να εκτελεστούν με χειρωνακτικούς ελέγχους.
- Οι εφαρμογές αλλάζουν και γίνονται πιο πολύπλοκες με την πάροδο του χρόνου, συνεπώς οι έλεγχοι που θα πρέπει να γίνουν σε αυτές γίνονται ακόμη πιο πολύπλοκοι και χρονοβόροι. Οι αυτοματοποιημένες δοκιμές έχουν το πλεονέκτημα πως μπορούν να τροποποιηθούν και να προσαρμοστούν εύκολα στις νέες συνθήκες και απαιτήσεις της εκάστοτε εφαρμογής.

Η αυτοματοποίηση είναι κάτι παραπάνω από έναν απλό έλεγχο λογισμικού. Απαιτεί ένα ολοκληρωμένο σύστημα και περιβάλλον για τη δημιουργία και την καταγραφή των ελέγχων, την οργάνωση και την συντήρησή τους. Στην συνέχεια ο οργανισμός θα πρέπει να διαθέτει ειδικά λογισμικά για την εκτέλεση των αυτοματοποιημένων ελέγχων καθώς και την στατιστική ανάλυση των αποτελεσμάτων. Το σίγουρο είναι πώς κάθε ξεχωριστός αυτοματοποιημένος έλεγχος θα πρέπει να συνοδεύεται από λεπτομερή τεκμηρίωση της διαδικασίας και των αποτελεσμάτων. Θα πρέπει να σημειωθεί εδώ πως δεν επαρκεί ένα καλό εργαλείο αυτοματοποίησης που θα δίνεται σε μεμονωμένους μηχανικούς ελέγχου. Αυτό θα προκαλέσει αναπόφευκτα την σύγκρουση διότι κάθε μηχανικός θα ακολουθεί διαφορετική μέθοδο ελέγχου, ίσως και διαφορετικές γλώσσες προγραμματισμού, με διαφορετική δομή για τον ίδιο έλεγχο. Αντι αυτού, η αυτοματοποίηση θα πρέπει να οργανώνεται από διαφορετικούς ανθρώπους με κοινό παρανομαστή. Θα πρέπει δηλαδή, από την πρώτη στιγμή, να συμφωνηθεί το περιβάλλον και η πλατφόρμα δοκιμών, η γλώσσα που θα χρησιμοποιηθεί, το πλήθος δοκιμών που θα εκτελεστούν κλπ.

5.2.1 ΠΟΤΕ ΔΕΝ ΠΡΕΠΕΙ ΝΑ ΕΠΙΛΕΓΕΤΑΙ Η ΑΥΤΟΜΑΤΟΠΟΙΗΣΗ

Ο ακρογωνιαίος λίθος της αυτοματοποίησης ελέγχων λογισμικού είναι πως η σωστή λειτουργία της εξεταζόμενης εφαρμογής είναι γνωστή. Όταν δεν είναι, η αυτοματοποίηση δεν είναι η καλύτερη επιλογή. Υπάρχουν εφαρμογές που είναι ασταθής από την φάση του σχεδιασμού, όπως για παράδειγμα ένα σύστημα που προβλέπει τον καιρό. Τέτοιου είδους συστήματα, τα οποία δεν είναι σταθερά και δεν εμφανίζουν συνέπεια στα αποτελέσματα, είναι αδύνατο να ελεγχθούν αυτοματοποιημένα. Η επένδυση που απαιτεί η αυτοματοποίηση δοκιμών μιας τέτοιας εφαρμογής είναι μεγάλη, ενώ ταυτόχρονα τα οφέλη της αυτοματοποίησης είναι λίγα[15]. Επιπλέον, εάν οι μηχανικοί ελέγχου που επιλέγονται για αυτοματοποίηση δεν έχουν την ανάλογη πείρα, τότε υπάρχει ο κίνδυνος αφενός να μην γίνει σωστά η τεχνική, αφετέρου δε να επιλεχθούν προς έλεγχο ασήμαντες λειτουργίες της εφαρμογής ή ακόμη λειτουργίες που μπορούν να ελεγχθούν εύκολα και γρήγορα με το χέρι. Επίσης, η αυτοματοποίηση δεν θα πρέπει να επιλέγεται στις περιπτώσεις όπου, η σύσταση της ομάδας ελέγχου έχει προσωρινό χαρακτήρα, διότι η επένδυση της εταιρείας σε μια τέτοια τεχνική είναι τόσο μεγάλη που δεν θα πρέπει να αξιοποιηθεί για λίγο. Τέλος, μια επιτυχημένη τεχνική ελέγχου απαιτεί αρκετό χρόνο και πλήθος πόρων. Θα πρέπει να υπάρχει υπομονή και επιμονή στην κατάρτιση των εργαζομένων και στην εξαγωγή αποτελεσμάτων. Εάν δεν υπάρχει αρκετός χρόνος, προτείνεται η επιλογή χειρονακτικών ελέγχων λογισμικού.

5.2.2 ΔΙΑΔΙΚΑΣΙΑ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΥ ΕΛΕΓΧΟΥ

Σε έναν ιδανικό κόσμο, ο κύκλος ζωής του ελέγχου θα λάμβανε χώρα συγχρόνως με τον κύκλο ζωής ανάπτυξης του λογισμικού. Ωστόσο, οι διαδικασίες ελέγχου δεν ξεκινούν πάντα, στο αρχικό στάδιο ανάπτυξης του λογισμικού[36]. Παρακάτω αναλύονται οι κανόνες και οι τακτικές που θα πρέπει να ακολουθήσει ένας οργανισμός για επιτευχθεί ο εν λόγω συγχρονισμός και να πετύχει την βέλτιστη αυτοματοποίηση ελέγχου.

5.2.2.1 Η ΟΜΑΔΑ ΕΛΕΓΧΟΥ

Η ομάδα ελέγχου που επιλέγεται να αυτοματοποιήσει τις δοκιμές θα πρέπει να είναι αφοσιωμένη στον στόχο της και να είναι σε θέση να λαμβάνει βοήθεια από άλλες ομάδες της εταιρείας. Είναι σημαντικό η εταιρεία να γνωρίζει τις ικανότητες κάθε ατόμου και να του αναθέτει τις αντίστοιχες αρμοδιότητες. Το είδος δεξιοτήτων που απαιτεί η αυτοματοποίηση διαφέρει ανάλογα τον τρόπο που επιλέγει η εταιρεία να αυτοματοποιήσει. Εάν, ο έλεγχος μιας εφαρμογής προϋποθέτει την συγγραφή τμημάτων κώδικα, τότε η εταιρεία θα επιλέξει το άτομο με το ανάλογο τεχνικό υπόβαθρο. Τα προτεινόμενα μέλη της ομάδας ελέγχου και οι αντίστοιχες αρμοδιότητές τους περιγράφονται παρακάτω[71]:

- Αρχηγός ομάδας ελέγχου
  Ο αρχηγός της ομάδας ελέγχου είναι υπεύθυνος για την ανάπτυξη ενός λεπτομερούς σχεδίου αυτοματοποίησης καθώς και την ανάθεση εργασιών στα μέλη της ομάδας ανάλογα τις ικανότητες του καθενός. Πρέπει να έχει την εξουσία να δίνει καθήκοντα και να αξιολογεί την εργασία των μελών της ομάδας.
- Προγραμματιστές αυτοματοποιημένων ελέγχων
  Οι προγραμματιστές αυτοματοποιημένων ελέγχων ειδικεύονται στον έλεγχο της λειτουργικότητας της εφαρμογής. Είναι υπεύθυνοι για την ανάπτυξη κώδικα που ελέγχει αυτοματοποιημένα διάφορες λειτουργίες της εφαρμογής. Εκτελούν τον κώδικα, αναλύουν και καταγράφουν τα αποτελέσματα.
- Προγραμματιστές πιθανών δοκιμαστικών σεναρίων
  Οι προγραμματιστές πιθανών δοκιμαστικών σεναρίων (test scripts) θα πρέπει να είναι ειδικοί στον χειρισμό του εργαλείου ελέγχου που έχει επιλέξει η εταιρεία, ενώ θεμιτό είναι να γνωρίζουν προγραμματισμό. Είναι υπεύθυνοι για την ανάπτυξη και την συντήρηση σεναρίων ελέγχων.

5.2.3 ΟΔΗΓΙΕΣ ΚΑΙ ΑΝΑΛΥΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Οι οδηγίες για το λογισμικό αυτοματοποιημένου ελέγχου μπορεί να είναι είτε σχόλια μέσα στον κώδικα που εξηγούν την λειτουργία του είτε έγγραφα που εξηγούν τον τρόπο εκτέλεσής τους, τις προτεινόμενες εισόδους, ακόμη και τον τρόπο με τον οποίο ένας τρίτος χρήστης μπορεί να επέμβει στον κώδικα[8]. Μόνο το πρόσωπο το οποίο έχει δημιουργήσει τον κώδικα ελέγχου μπορεί να γνωρίζει πως αυτός τρέχει, πότε τα αποτελέσματα είναι θετικά και πότε όχι. Για αυτόν το λόγο ο κώδικας πρέπει να συνοδεύεται με τις κατάλληλες πληροφορίες και οδηγίες χρήσης, έτσι ώστε όταν κάποιος άλλος μηχανικός ελέγχου τον χρησιμοποιήσει, να είναι σε θέση να τον τρέξει, να κατανοεί τον τρόπο λειτουργίας του και φυσικά να εξάγει ασφαλή συμπεράσματα παρατηρώντας τα αποτελέσματα. Πάντα θα πρέπει ο κώδικας αυτοματοποίησης να συνοδεύεται από λεπτομερή τεκμηρίωση και περιγραφή του τι αναμένεται να ελέγξει, το είδος και το μέγεθος της εισόδου που μπορεί να δεχτεί και τα ακριβή σημεία όπου μπορεί να τροποποιηθεί. Σε κάθε περίπτωση, οι μηχανικοί ελέγχου θα πρέπει έχουν κατά νου πως το λογισμικό αυτοματοποιημένου ελέγχου που δημιουργούν θα πρέπει ιδανικά να είναι κατανοητό και ευανάγνωστο ακόμη και από κάποιον απλό χρήστη. Οι οδηγίες χρήσης και η ανάλυση του κώδικα θα πρέπει να ακολουθούν αυτήν τη λογική. Επιπλέον, πρέπει εξαρχής να καθορίζεται και να αναλύεται πότε ένα λογισμικό έχει περάσει με επιτυχία τον έλεγχο και πότε όχι, πόσο ήταν το ποσοστό επιτυχίας. Επιπλέον, τα έγγραφα τεκμηρίωσης θα πρέπει να συνοδεύονται από πρόσθετες πληροφορίες που μπορούν να βοηθήσουν στη διάγνωση των πιθανών αποτυχιών. Ακόμη και ο ίδιος ο κώδικας, θα πρέπει να έχει αναπτυχθεί με τέτοιο τρόπο έτσι ώστε να εμφανίζει μηνύματα προς τον μηχανικό ελέγχου για το είδος του σφάλματος που παρουσιάστηκε, την ώρα που συνέβη και τις συνθήκες υπό τις οποίες βρισκόταν το λογισμικό που εξετάζεται εκείνη την στιγμή[72]. Γενικά όσο πιο λεπτομερής και αναλυτική είναι η περιγραφή του κώδικα αυτοματοποιημένου ελέγχου και των αποτελεσμάτων που αναμένονται από την εκτέλεση αυτού, τόσο πιο εύκολο είναι να επαναχρησιμοποιηθεί είτε στο βραχυπρόθεσμο είτε στο μακροπρόθεσμο μέλλον. Επίσης, θεμιτό είναι τα αποτελέσματα των αυτοματοποιημένων ελέγχων να συνοδεύονται και από στατιστικές αναλύσεις. Έτσι, η ομάδα ελέγχου και γενικότερα η ομάδα ανάπτυξης του λογισμικού μπορεί να εντοπίσει πιο εύκολα την πηγή του λάθους και επιπλέον εάν αυτό συμβαίνει όταν εκτελείται μεγάλο πλήθος δοκιμών ή συνέβη τυχαία και μια φορά.

Οι μηχανικοί αυτοματοποιημένων ελέγχων θα πρέπει να είναι ενημερωμένοι και την πιθανότητα εξαγωγής ανακριβών αποτελεσμάτων που δεν δείχνουν την αληθινή κατάσταση της δοκιμής. Υπάρχουν τρία είδη ανακριβών αποτελεσμάτων[47]:

- Ψευδείς αποτυχίες
  a) Ψευδείς αποτυχίες από το περιβάλλον ελέγχου: Πρόκειται για σφάλματα που προκύπτουν για άλλους λόγους, πέρα από την λειτουργία του λογισμικού που εξετάζεται. Τα σφάλματα αυτά μπορεί να προέρχονται από την κατάσταση της βάσης, από τις λάθος ρυθμίσεις και παραμέτρους που έχουν επιλεγεί για το περιβάλλον στο οποίο τρέχει ο έλεγχος και γενικότερα από ένα λάθος που προκάλεσε την αποτυχία του ελέγχου αλλά δεν έχει καμία σχέση με το λογισμικό που ελέγχεται.

b) Ψευδείς αποτυχίες που προκύπτουν από τις αλλαγές στην εφαρμογή: Ένας άλλος τύπος ψευδής αποτυχίας μπορεί να συμβεί εάν προστεθεί ένα χαρακτηριστικό ή μια λειτουργία στο λογισμικό που εξετάζεται και προκαλέσει αποτυχία στο σενάριο ελέγχου που ήδη εκτελείται.

- Διπλές αποτυχίες
  Μια διπλή αποτυχία οφείλεται σε ένα λάθος που έχει εντοπιστεί σε προηγούμενο είδος ελέγχου αλλά δεν έχει διορθωθεί με αποτέλεσμα να παρατηρείται το ίδιο λάθος επαναλαμβανόμενα σε κάθε εκτέλεση του αυτοματοποιημένου ελέγχου. Αυτό προκαλεί διαστρεβλώσεις στα τελικά αποτελέσματα και στον υπολογισμό των μετρικών.
- Ψευδείς επιτυχίες
  a) Ψευδείς επιτυχίες από λάθη που συμβαίνουν κατά τον έλεγχο: Πρόκειται για λάθος αποτελέσματα που προκύπτουν όταν ο κώδικας ελέγχου, εξετάζει ελάχιστες από τις πτυχές και λειτουργίες του λογισμικού αγνοώντας τις υπόλοιπες. Κάτι τέτοιο μπορεί να συμβεί και καταλάθος, εάν για παράδειγμα μια συνθήκη μέσα στον κώδικα ελέγχου γίνει ψευδής αντί για αληθής και το πρόγραμμα δεν καταφέρει να τρέξει το συγκεκριμένο κομμάτι κώδικα που ελέγχει και άλλα μέρη του εξεταζόμενου λογισμικού. Το συγκεκριμένο σφάλμα στον κώδικα ελέγχου μπορεί να εντοπιστεί και από τον χρόνο εκτέλεσης του κώδικα. Εάν είναι ανεξήγητα μικρός, τότε κάποια κομμάτια του ελέγχου πιθανόν δεν έχουν τρέξει.
  b) Ψευδείς επιτυχίες από λάθος που δεν έχει εντοπιστεί: Πρόκειται για λάθος αποτελέσματα που προκύπτουν εάν το σενάριο δοκιμής επικεντρώνεται σε συγκεκριμένη λειτουργία της εφαρμογής, που δεν χαρακτηρίζεται κρίσιμη και αμελεί σημαντικότερες.

5.3 ΕΡΓΑΛΕΙΑ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΥ ΕΛΕΓΧΟΥ

Τα εργαλεία αυτοματοποιημένου ελέγχου διαδραματίζουν κυρίαρχο ρόλο στην επιτυχία της ομάδας ελέγχου. Είναι μια συλλογή προϊόντων λογισμικού ειδικά σχεδιασμένων να βοηθούν τους μηχανικούς και διαχειριστές αυτοματοποιημένων ελέγχων να επεκτείνουν την εργασία τους και να καλύψουν πλήθος διαφορετικών δοκιμαστικών περιπτώσεων. Υπάρχει πληθώρα τέτοιων εργαλείων στην αγορά, κάθε ένα εκ των οποίων βοηθά τον μηχανικό ελέγχου να οργανώνει, να προγραμματίζει και να μεταγλωττίζει εύκολα τους κώδικές του.
Τα πρώτα εργαλεία αυτοματοποιημένου ελέγχου εμφανίστηκαν περί τα 1980. Τα συγκεκριμένα εργαλεία ήταν απλά, εκτελούσαν απλούς ελέγχους με επίκεντρο την σωστή λειτουργία ενός λογισμικού. Πολλές από τις πρώτες αυτοματοποιημένες εργασίες ελέγχου ήταν επέκταση των κλασσικών μεταγλωττιστών κώδικα. Στην συνέχεια, τα εργαλεία που συνδύαζαν τη διοίκηση και την λειτουργία του ελέγχου άρχισαν να παίρνουν προβάδισμα στην αγορά, ώσπου το 1990, οι πωλητές των εργαλείων ελέγχου ξεκίνησαν να ενώνουν τα διάφορα μικρά εργαλεία που εκτελούσαν ξεχωριστούς ελέγχους και δημιούργησαν μεγαλύτερα τα οποία και χαρακτηρίστικαν πιο αποδοτικά[71]. Γύρω στο 1995, ένα νέο είδος εργαλείου αυτοματοποιημένου εργαλεία εμφανίστηκε, το οποίο υπολόγιζε τα ποιοτικά χαρακτηριστικά του λογισμικού, όπως ακριβώς ορίστικαν στο κεφάλαιο 2. Έπειτα, όσο πιο περίπλοκα γινόντουσαν τα συστήματα λογισμικού, τόσο πιο περίπλοκη έγινε και η εκτίμηση της ποιότητάς του. Πόσο μάλλον όταν αυτή μετράτε αυτοματοποιημένα. Διακρίνονται δύο είδη εργαλείων: τα εργαλεία ελέγχου λειτουργικότητας και τα εργαλεία εκτίμησης της απόδοσης ενός λογισμικού. Τα πρώτα επικεντρώνονται στην σύγκριση των φαινομενικών και των αναμενόμενων αποτελεσμάτων κάθε ελέγχου χαρακτηρίζοντάς τα ως pass ή fail, ενώ τα εργαλεία αυτοματοποιημένου ελέγχου που ανήκουν στη δεύτερη κατηγορία επικεντρώνονται στον χρόνο ανταπόκρισης του λογισμικού σε κάθε πιθανή συνθήκη ελέγχου. Γύρω στο 2000, οι δημιουργοί των εν λόγω εργαλείων, άρχισαν να συγκεντρώνουν τα διάφορα εργαλεία ποιοτικού ελέγχου, δημιουργώντας τις λεγόμενες σουίτες δοκιμών (test suites), οι οποίες παρέχουν τη δυνατότητα εύκολης διαχείρισης του ελέγχου ενώ συνδυάζουν και τα δύο παραπάνω είδη ελέγχου. Δηλαδή, μέσα από τα νέα εργαλεία, θα μπορεί κανείς να ελέγξει αυτοματοποιημένα την λειτουργία και την απόδοση του εκάστοτε λογισμικού. Τα αδιαμφησβήτητα πλεονεκτήματα των αυτοματοποιημένων εργαλείων, προκάλεσαν την ουσιαστική κυριαρχία τους στην αγορά εργαλείων ελέγχων[69]. Εάν ένα συγκεκριμένο σενάριο αυτοματοποιημένου ελέγχου εκτελεστεί πολλές φορές, το εργαλείο εξοικονομεί χρήματα. Η προσπάθεια, οι πόροι και το κόστος που απαιτείται για τη δημιουργία ενός σεναρίου αυτοματοποιημένου ελέγχου από το μηδέν είναι σαφώς υψηλότερο από το να γίνει ο ίδιο έλεγχος με το χέρι για μια φορά. Τη δεύτερη φορά που θα εκτελεστεί το ίδιο σενάριο αυτοματοποιημένου ελέγχου δεν θα κοστίσει καθόλου, ενώ αντιθέτως τη δεύτερη φορά που θα γίνει χειροκίνητα ο έλεγχος θα έχει ακριβώς το ίδιο κόστος όπως την πρώτη φορά. Συνεπώς, παρότι η αυτοματοποίηση κοστίζει την πρώτη φορά που επιχειρείται, παραταύτα σε βάθος χρόνου είναι πολύ πιο οικονομική από τους παραδοσιακούς τρόπους ελέγχου. Τα καλύτερα εργαλεία αυτοματοποιημένου ελέγχου όπως έχουν διαμορφωθεί, για το 2018, αναφέρονται στο παράρτημα Α στο τέλος του παρόντος κειμένου.

5.4 ΣΧΕΔΙΑΣΗ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (TEST AUTOMATION DEVELOPMENT)

Ο μηχανικός ελέγχου που αναπτύσσει και προγραμματίζει τα σενάρια αυτοματοποιημένου ελέγχου θα πρέπει να έχει καλή γνώση προγραμματισμού καθώς ο προγραμματισμός κομματιών κώδικα που εκτελεί αυτοματοποιημένο έλεγχο είναι μια σύνθετη και δύσκολη διαδικασία που βαρύνει με ευθύνες τον μηχανικό, ίδιες με εκείνες που βαρύνει τον προγραμματιστή της εφαρμογής που εξετάζεται. Ενώ οι χειρωνακτικοί έλεγχοι είναι μια διαδικασία που συνήθως λαμβάνει χώρα στο τέλος του κύκλου ζωής ανάπτυξης του λογισμικού, οι αυτοματοποιημένες δοκιμές θα πρέπει να ενσωματωθούν στη διαδικασία ανάπτυξης του λογισμικού, από την αρχή. Οι μηχανικοί που αναπτύσσουν τους κώδικες αυτοματοποιημένου ελέγχου, χρησιμοποιούν εργαλεία που διευκολύνουν την ανάπτυξη του κώδικα και ταυτόχρονα δημιουργούν σενάρια ελέγχου που εξετάζουν το λογισμικό για την λειτουργία και την απόδοσή του. Ο κώδικας ελέγχου μπορεί να προγραμματιστεί με γλώσσες όπως BASIC, C, C++, Fortran, Python κλπ, ενώ ταυτόχρονα πρέπει να μπορεί να τροποποιηθεί και προσαρμοστεί για μελλοντικά σενάρια ελέγχων.

5.4.1 ΚΑΝΟΝΕΣ ΣΥΓΓΡΑΦΗΣ ΤΩΝ TEST SCRIPTS

Ένας καλά δομημένος κώδικας αυτοματοποιημένου ελέγχου θα πρέπει[73]:

- Να περιέχει σχόλια για να καθοδηγεί τόσο τον χρήστη όσο και αυτόν που το συντηρεί.
- Να είναι λειτουργικός και να είναι επαναχρησιμοποιούμενος.
- Να είναι καλά οργανωμένος, ευανάγνωστος, κατανοητός.
- Να συνοδεύεται από την ανάλογη τεκμηρίωση, η οποία θα περιλαμβάνει και τα επιθυμητά αποτελέσματα που αναμένεται να εξάγει ο έλεγχος.

Οι παραπάνω βασικές αρχές θα πρέπει να τηρούνται ανεξάρτητα της τεχνικής και της γλώσσας που χρησιμοποιούνται για τον προγραμματισμό των σεναρίων ελέγχου.

5.4.2 ΤΕΧΝΙΚΕΣ SCRIPTING

Παρακάτω αναλύονται οι διαφορετικές τεχνικές οργάνωσης και συγγραφής προγραμμάτων που αυτοματοποιούν τον έλεγχο. Οι εν λόγω τεχνικές συνήθως δεν χρησιμοποιούνται αυτόνομα, αλλά συνδυάζονται πετυχαίνοντας την παραγωγή αξιόλογων μεθόδων ελέγχου. Κάθε τεχνική, όπως αναλύεται παρακάτω, έχει τα πλεονεκτήματά και τα μειονεκτήματά της, ενώ απαιτεί συγκεκριμένο χρόνο και προσπάθεια για την σωστή ανάπτυξη και συντήρηση των προγραμμάτων ελέγχων[8]. Κάθε τέτοιο πρόγραμμα αυτοματοποιημένου ελέγχου αντιστοιχεί σε δεκάδες δοκιμαστικές περιπτώσεις, αναλόγως τις μεταβλητές που δέχεται σαν είσοδο, το περιβάλλον υλοποίησης, τη διάρκεια εκτέλεσης και γενικότερα τις συνθήκες που επιλέγει η ομάδα ελέγχου, κάτω από τις οποίες θα εκτελέσει τον αντίστοιχο έλεγχο. Όταν έχει συγκεντρωθεί ο επιθυμητός αριθμός δοκιμαστικών περιπτώσεων, για κάθε πρόγραμμα ελέγχου, έχουν εκτελεστεί και έχουν φέρει τα αντίστοιχα αποτελέσματα, η ομάδα είναι υπεύθυνη να καταγράψει, να οργανώσει και αναλύσει τα αποτελέσματα, είτε θετικά είτε αρνητικά. Επίσης, η ομάδα ελέγχου είναι υπεύθυνη να περιγράφει τη διαδικασία που ακολούθησε από την πρώτη στιγμή έως την τελευταία, σε τεκμηριωμένα έγγραφα που θα είναι κατανοητά από όλους. Τέλος, εάν η ομάδα ελέγχου θέλει να πετύχει τα βέλτιστα αποτελέσματα μέσα από τη δουλεία της, η οποία θα είναι χρήσιμη, θα πρέπει να συμπεριλάβει στον κώδικα της, τις ανάλογες εντολές και μεθόδους που θα λαμβάνουν το αποτέλεσμα από κάθε ξεχωριστό έλεγχο και θα χτίζουν σταδιακά μια βάση δεδομένων γεμάτη από αποτελέσματα. Στην συνέχεια, θα κάνει μια στοιχειώδη στατιστική ανάλυση πάνω στα δεδομένα, η οποία θα είναι χρήσιμη γιατί θα καταδεικνύει τις δοκιμαστικές περιπτώσεις με τα περισσότερα αρνητικά αποτελέσματα κάνοντας έτσι πιο εύκολη τη διόρθωση πιθανών λαθών στο λογισμικό. Οι τεχνικές συγγραφής και ανάπτυξης των προγραμμάτων που αυτοματοποιούν τον έλεγχο χωρίζονται σε πέντε κατηγορίες[74]:

- Γραμμικά οργανωμένα προγράμματα αυτοματοποιημένου ελέγχου (linear scripting)
- Δομημένα προγράμματα αυτοματοποιημένου ελέγχου (structured scripting)
- Διαμοιραζόμενα προγράμματα αυτοματοποιημένου ελέγχου (shared scripts)
- Προγράμματα αυτοματοποιημένου ελέγχου που βασίζονται στα δεδομένα (data-driven scripts)
- Προγράμματα αυτοματοποιημένου ελέγχου που βασίζονται σε λέξεις-κλειδιά (keyword-driven scripts)

5.4.2.1 ΓΡΑΜΜΙΚΑ ΟΡΓΑΝΩΜΕΝΑ ΠΡΟΓΡΑΜΜΑΤΑ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΥ ΕΛΕΓΧΟΥ

Τα γραμμικά οργανωμένα προγράμματα αυτοματοποιημένου ελέγχου είναι αυτά που αναπτύσσονται όταν κάποιος παρατηρήσει και προσπαθήσει να αντιγράψει ακριβώς τους χειροκίνητους ελέγχους λογισμικού. Περιέχουν, όλες εκείνες τις κινήσεις που θα έκανε κάποιος με το χέρι αν θα ήθελε να ανοίξει μια εφαρμογή με το χέρι. Δηλαδή, η τεχνική αυτή προσπαθεί να μεταφράσει σε γλώσσα μηχανής και αυτοματοποιήσει τις φυσικές κινήσεις κάποιου που θα χρησιμοποιούσε την εφαρμογή που βρίσκεται υπό εξέταση. Οι κινήσεις αυτές μπορεί να είναι για παράδειγμα πληκτρολόγηση κάποιων δεδομένων εισόδου, είτε αυτά είναι γράμματα είτε

αριθμοί, «πάτημα» κάποιου πλήκτρου βέλους, επιλογή κάποιας λειτουργίας της εφαρμογής και πολλά άλλα. Επίσης, η συγκεκριμένη τεχνική εισάγει και συγκρίσεις, όπως ο έλεγχος εάν εμφανίζεται το αναμενόμενο μήνυμα λάθους μετά από πιθανή διακοπή λειτουργίας, ή αν για παράδειγμα εμφανίζεται η ανάλογη έξοδος στην οθόνη όταν όλα βαίνουν καλώς και ο χρήστης έχει εισάγει κάποια δεδομένα. Όσο πιο περίπλοκο είναι το σύστημα που εξετάζεται, τόσο περισσότερο χρόνο απαιτεί η μεταφορά του ελέγχου των λειτουργιών του σε κώδικα. Τα πλεονεκτήματα της συγκεκριμένης τεχνικής συγγραφής αυτοματοποιημένων ελέγχων είναι τα εξής:

- Δεν απαιτείται εργασία ή σχεδιασμός εκ των προτέρων, εφόσον προϋποθέτει παρατηρητικότητα και ενδελεχή καταγραφή όλων των ελέγχων που θα είχαν εκτελεστεί χειροκίνητα, εάν δεν είχε επιλεγεί η αυτοματοποίηση. Έτσι, ξεκινά γρήγορα η ανάπτυξή τους.
- Ο χρήστης δεν χρειάζεται να είναι προγραμματιστής. Οι συγκεκριμένοι κώδικες είναι συνήθως απλοί και κατανοητοί.
- Οι κώδικες που παράγονται με την συγκεκριμένη τεχνική, είναι ιδανικοί για επίδειξη της εξεταζόμενης εφαρμογής, εφόσον προσομοιάζουν ακριβώς τις λειτουργίες της.

Σχεδόν κάθε επαναλαμβανόμενη ενέργεια του συστήματος λογισμικού που ελέγχεται, μπορεί να αυτοματοποιηθεί χρησιμοποιώντας την γραμμική μέθοδο και μάλιστα σε κάποιες περιπτώσεις αποτελεί την βέλτιστη επιλογή. Τα γραμμικά οργανωμένα προγράμματα αυτοματοποιημένου ελέγχου είναι ιδανικά για περιπτώσεις όπου η ομάδα ελέγχου θέλει να κάνει μια επίδειξη λειτουργίας ενός λογισμικού, γιατί προσφέρουν τη δυνατότητα να επιτευχθεί ακριβής προσομοίωση των λειτουργιών του και μάλιστα μπορεί να τις επαναλάβει όσες φορές επιθυμεί η ομάδα. Επιπλεόν, η συγκεκριμένη τεχνική μπορεί να χρησιμοποιηθεί για τη δημιουργία προγραμμάτων που θα εισάγουν αυτοματοποιημένα, ρυθμίσεις και νέα δεδομένα σε ήδη υπάρχοντα προγράμματα αυτοματοποιημένου ελέγχου. Τέλος, η τεχνική αυτή χρησιμοποιείται σε περιτπώσεις, όπου έπειτα από μια μικρή αλλαγή στο λογισμικό, αυτή μπορεί να ελεγχθεί επεμβαίνοντας ελάχιστα στον κώδικα του ελέγχου. Δεν χρειάζεται να επαναπρογραμματιστεί νέος έλεγχος. Εάν έχει οργανωθεί ο σωστά ο γραμμικός αυτοματοποιημένος έλεγχος της προηγούμενης έκδοσης του εξεταζόμενου λογισμικού, τότε δεν θα είναι δύσκολο για έναν μηχανικό ελέγχου να εντάξει την αλλαγή στο πρόγραμμα ελέγχου προσθέτοντας εκ νέου κομμάτι κώδικα που θα ελέγχει τις νέες ρυθμίσεις. Τα γραμμικά οργανωμένα προγράμματα αυτοματοποιημένου ελέγχου έχουν κάποια μειονεκτήματα:

- Η διαδικασία απαιτεί χρόνο που συνήθως υπερβαίνει κατά πολύ τον χρόνο που θα σπαταλούσε η ομάδα ελέγχου για να εξετάσει το λογισμικό με το χέρι.
- Οι διεργασίες ξεκινούν από το μηδέν, ενώ τα δεδομένα εισόδου και οι συγκρίσεις προγραμματίζονται δύσκολα, ειδικά εάν ο μηχανικός ελέγχου είναι άπειρος.
- Δεν προσφέρουν τη δυνατότητα επαναχρησιμοποίησης. Κάθε τέτοιο πρόγραμμα αντιστοιχεί στον έλεγχο συγκεκριμένου λογισμικού.
- Τα προγράμματα που συγγράφονται με την γραμμική μέθοδο είναι ευάλωτα στις αλλαγές του λογισμικού.
- Έχουν υψηλό κόστος αλλαγής και συντήρησης.
- Τα εν λόγω προγράμματα αποτυγχάνουν εύκολα, ακόμη και αν το εξεταζόμενο λογισμικό λειτουργεί καλά. Παράγοντες περιβάλλοντος τα επηρεάζουν εύκολα.

Τα παραπάνω μειονεκτήματα καθιστούν τα γραμμικά οργανωμένα προγράμματα αυτοματοποιημένου ελέγχου, μη ιδανικά για την εκτέλεση πλήθους μακροχρόνιων δοκιμών.

5.4.2.2 Δομημενα προγραμματα αυτοματοποιημενου ελεγχου

Τα δομημένα προγράμματα αυτοματοποιημένου ελέγχου αναπτύσσονται παράλληλα με το δομημένο προγραμματισμό και χρησιμοποιούν εξιδεικευμένες οδηγίες για τον έλεγχο της εκτέλεσης κάθε ξεχωριστής δοκιμής, οι οποίες λέγονται δομές ελέγχου. Υπάρχουν τρεις βασικές δομές ελέγχου που υποστηρίζονται από όλες τις γλώσσες προγραμματισμού προγραμμάτων αυτοματοποιημένου ελέγχου. Η πρώτη ονομάζεται δομή ακολουθίας και είναι ακριβώς η ίδια τεχνική με αυτή που αναλύθηκε στην παράγραφο 5.4.2.1. Η πρώτη οδηγία εκτελείται πρώτα, έπειτα η δεύτερη και ούτω καθεξής. Τα άλλα δύο είδη δομών ελέγχου είναι οι δομές επιλογής και οι δομές επανάληψης. Η δομή επιλογής δίνει τη δυνατότητα στο πρόγραμμα να επιλέγει και να παίρνει αποφάσεις. Η πιο γνωστή δομή επιλογής είναι η «if», η οποία αξιολογεί μια συνθήκη και την χαρακτηρίζει ψευδή ή αληθή. Για παράδειγμα, χρησιμοποιώντας την συγκεκριμένη δομή ελέγχου, ένα πρόγραμμα μπορεί να ελέγξει εάν έχει εμφανιστεί το μήνυμα που έπρεπε στην οθόνη. Αν ναι, ο έλεγχος συνεχίζεται, αν όχι ο έλεγχος σταματά εμφανίζοντας το ανάλογο μήνυμα που δικαιολογεί τη διακοπή. Η δομή ελέγχου επανάληψης δίνει στο πρόγραμμα ελέγχου τη δυνατότητα να επαναλάβει μια ή περισσότερες οδηγίες όσες φορές χρειάζεται. Η διαδικασία επανάληψης ονομάζεται βρόχος και η ακολουθία των οδηγιών μπορεί να οριστεί έτσι ώστε να επαναλαμβάνεται όσες φορές πρέπει έως ότου ικανοποιηθεί μια συνθήκη. Για παράδειγμα, εάν για ένα είδος αυτοματοποιημένου ελέγχου απαιτείται η ανάγνωση ενός αρχείου κειμένου, η εντολή ανάγνωσης κάθε γραμμής θα πρέπει να επαναληφθεί τόσες φορές όσες και οι γραμμές του κειμένου. Η χρήση των δομημένων

προγραμμάτων αυτοματοποιημένου ελέγχου καθιστά τους κώδικες κατανοητούς και επαναχρησιμοποιήσιμους, ενώ ταυτόχρονα καταφέρνει να αυξήσει τη δυναμικότητα και την ευελιξία των προγραμμάτων ελέγχου. Η καλή χρήση των δομών ελέγχου οδηγεί στην εύκολη διατήρηση και προσαρμογή των προγραμμάτων ελέγχου, γεγονός που υποστηρίζει γενικότερα την αυτοματοποίηση ελέγχου, εφόσον οι κώδικες αυτοματοποίησης γίνονται διαχειρίσιμοι, ευανάγνωστοι και αποτελεσματικοί. Ωστόσο, η συγκεκριμένη τεχνική προϋποθέτει καλή γνώση προγραμματισμού. Επιπροσθέτως, ένα πρόγραμμα αυτοματοποιημένου ελέγχου μπορεί, μέσα από τον κώδικά του, να καλέσει ένα άλλο πρόγραμμα αυτοματοποιημένου ελέγχου να εκτελέσει μια συγκεκριμένη δοκιμή και όταν αυτή ολοκληρωθεί το πρώτο πρόγραμμα να συνεχίσει την εκτέλεσή του από το σημείο που διακόπηκε προσωρινά. Αυτός ο μηχανισμός χρησιμοποιείται για να διαιρεί τα μεγάλα προγράμματα αυτοματοποιημένου ελέγχου σε μικρότερους και πιο εύκολα διαχειρίσιμους κώδικες.

5.4.2.3 ΔΙΑΜΟΙΡΑΖΟΜΕΝΑ ΠΡΟΓΡΑΜΜΑΤΑ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΥ ΕΛΕΓΧΟΥ

Τα διαμοιραζόμενα προγράμματα αυτοματοποιημένου ελέγχου είναι προγράμματα που χρησιμοποιούνται από πολλά είδη ελέγχου και πλήθος δοκιμαστικών περιπτώσεων. Η επιτυχής λειτουργία τους προϋποθέτει την ικανότητα ενός προγράμματος αυτοματοποιημένου ελέγχου να καλεί ένα άλλο ή να καλείται από ένα άλλο. Η βασική ιδέα πίσω από την συγκεκριμένη τεχνική συγγραφής προγραμμάτων αυτοματοποιημένου ελέγχου είναι η ανάπτυξη κώδικα που εκτελεί κάποιον έλεγχο επαναλαμβανόμενα και έπειτα όταν μια συγκεκριμένη δοκιμαστική περίπτωση χρειαστεί να εκτελέσει τον ίδιο έλεγχο, να καλεί τον εν λόγω κώδικα που θα είναι ήδη έτοιμος. Τα πλεονεκτήματα της τεχνικής των διαμοιραζόμενων προγραμμάτων είναι τα εξής:

- Εξοικονομούν χρόνο γιατί δεν χρειάζεται να συγγράψει κάποιος έναν κώδικα από την αρχή προκειμένου να ελέγξει κάποιο ή κάποια χαρακτηριστικά που ήδη ελέγχονται στο διαμοιραζόμενο πρόγραμμα.
- Σε περίπτωση αλλαγής κάποιας εκ των ενεργειών που ελέγχονται επαναλαμβανόμενα, θα χρειαστεί να υπάρξει αλλαγή μόνο στο συγκεκριμένο πρόγραμμα αυτοματοποιημένου ελέγχου που εξετάζει τις λειτουργίες που αλλάζουν.
- Η συγγραφή παρόμοιων προγραμμάτων ελέγχου είναι γρηγορότερη και ευκολότερη.
- Το κόστος συντήρησης είναι μικρότερο σε σύγκριση με το αντίστοιχο των γραμμικών προγραμμάτων ελέγχου.
- Εξαλείφουν τις παρόμοιες επαναλήψεις, εξοικονομώντας χρόνο.
- Έχουν τη δυνατότητα να εντάξουν σύγχρονες τεχνικές ελέγχου στα ήδη υπάρχοντα προγράμματα.

Η τεχνική διαμοιραζόμενων προγραμμάτων είναι κατάλληλη για έλεγχο λειτουργιών σε μικρά συστήματα ή μόνο όταν ένα μικρό μέρος λειτουργιών ενός μεγάλου και σταθερού συστήματος μπορεί να ελεγχθεί με μικρό αριθμό ελέγχων. Θα πρέπει να σημειωθεί εδώ πως τα διαμοιραζόμενα προγράμματα αυτοματοποιημένου ελέγχου επικεντρώνονται σε εξιδεικευμένο μέρος του εξεταζόμενου λογισμικού.

Ένα από τα κύρια χαρακτηριστικά των σύγχρονων εργαλείων ανάπτυξης λογισμικού είναι η ευκολία με την οποία τα γραφικά των περιβαλλόντων ανάπτυξης μπορούν να μεταβάλουν τη διεπαφή του χρήστη σε ένα σύστημα. Ωστόσο, όσο πιο ελκυστικό και εύχρηστο γίνεται ένα σύστημα λογισμικού για τον τελικό χρήστη τόσο πιο δύσκολο είναι για την ομάδα ελέγχου να αναπτύξει κώδικες που θα ελέγχουν αυτοματοποιημένα τα καινούρια χαρακτηριστικά και ιδιότητες της διεπαφής χρήστη. Τα διαμοιραζόμενα προγράμματα αυτοματοποιημένου ελέγχου αποτελούν ένα βήμα προς την κατεύθυνση ανάπτυξης κώδικα αυτοματοποιημένου ελέγχου που θα μπορεί να ανταποκριθεί στις ανάγκες του σύγχρονου συνεχώς μεταβαλλόμενου λογισμικού. Υπάρχουν δύο μεγάλες κατηγορίες διαμοιραζόμενων προγραμμάτων αυτοματοποιημένου ελέγχου: τα προγράμματα που μπορούν να χρησιμοποιούνται από κοινού για έλεγχο διαφορετικών συστημάτων λογισμικού και τα προγράμματα που μπορούν να χρησιμοποιηθούν από τύπους δοκιμών που ελέγχουν το ίδιο σύστημα λογισμικού. Η πρώτη κατηγορία προγραμμάτων είναι πιο χρήσιμη μακροπρόθεσμα. Η επιτυχία των διαμοιραζόμενων προγραμμάτων αυτοματοποιημένου ελέγχου κρίνεται, σε μεγάλο ποσοστό, από τα έγγραφα τεκμηρίωσης που τα συνοδεύουν. Οι κώδικες θα πρέπει να είναι τεκμηριωμένοι με τέτοιο τρόπο ώστε να είναι ξεκάθαρος ο ρόλος και ο σκοπός του καθενός καθώς και ο τρόπος χρήσης τους.

5.4.2.4 ΠΡΟΓΡΑΜΜΑΤΑ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΥ ΕΛΕΓΧΟΥ ΠΟΥ ΒΑΣΙΖΟΝΤΑΙ ΣΤΑ ΔΕΔΟΜΕΝΑ

Η τεχνική ανάπτυξης προγραμμάτων αυτοματοποιημένου ελέγχου με γνώμονα τα δεδομένα βασίζεται στην αποθήκευση των παραμέτρων εισόδου σε ξεχωριστό αρχείο. Έτσι όταν εκτελείται ένα είδος ελέγχου, λαμβάνει τα δεδομένα εισόδου από το συγκεκριμένο αρχείο και όχι από το πρόγραμμα αυτοματοποίησης. Ένα σημαντικό πλεονέκτημα της εν λόγω τεχνικής είναι η δυνατότητα που προσφέρει να εκτελεστεί κάθε σενάριο ελέγχου, όσες φορές χρειαστεί, με διαφορετική είσοδο. Για παράδειγμα, συστήματα λογισμικού που δέχονται σαν είσοδο πλήθος στοιχείων από διαφορετικούς χρήστες, θα πρέπει να ελεγχθούν και να αξιολογηθούν για διαφορετικά δεδομένα με διαφορετικές ιδιότητες. Επιπλέον, η χρήση της συγκεκριμένης τεχνικής βοηθά την ομάδα ελέγχου να σχεδιάζει και να υλοποιεί διαφορετικά σενάρια δοκιμών, εφόσον η εκτέλεση του ίδιου προγράμματος με

διαφορετική είσοδο κάθε φορά, είναι στην πραγματικότητα εκτέλεση πολλών και διαφορετικών δοκιμαστικών περιπτώσεων. Ένα κύριο πλεονέκτημα της τεχνικής ανάπτυξης προγραμμάτων οδηγούμενων από τα δεδομένα, είναι η ευελιξία που δίνεται σε κάθε μηχανικό ελέγχου να προσαρμόσει το πρόγραμμα στις δικές του απαιτήσεις. Κάθε μηχανικός ελέγχου, θα μπορεί να εισάγει τις εισόδους που επιθυμεί και αυτές θα έχουν την μορφή και τη διάταξη που ο μηχανικός επιλέγει, χωρίς να επηρεάζεται το κυρίως πρόγραμμα ελέγχου. Επίσης, εκτός από τα δεδομένα εισόδου, τα δεδομένα εξόδου μπορούν με τον ίδιο τρόπο να αποθηκεύονται σε ξεχωριστό αρχείο με την ίδια λογική. Με την τακτική αυτή, τα δεδομένα εξόδου εύκολα μπορούν να συγκριθούν με τα αναμενόμενα αποτελέσματα, αναλόγως την είσοδο και τις συνθήκες εκτέλεσης του ελέγχου. Κάθε αναμενόμενο αποτέλεσμα μπορεί να συνδεθεί με τα αντίστοιχα δεδομένα εισόδου, κάνοντας εύκολη την σύγκριση και τον καθορισμό της απόκλισης με τα πραγματικά αποτελέσματα. Πολλά εργαλεία ανάπτυξης και εκτέλεσης αυτοματοποιημένων ελέγχων ενθαρρύνουν την τεχνική αυτή και παρέχουν εξαρχής τη δυνατότητα αποθήκευσης των δεδομένων εισόδου και εξόδου σε ξεχωριστά αρχεία. Θα πρέπει να σημειωθεί εδώ η συγγραφή αυτοματοποιημένων ελέγχων με την τεχνική ανάπτυξης οδηγούμενη από τα δεδομένα προϋποθέτει πείρα στον προγραμματισμό. Επιπλεόν, η ανάπτυξη προγραμμάτων με αυτή την μέθοδο, την πρώτη φορά απαιτεί χρόνο και προσπάθεια, ωστόσο η προσπάθεια αυτή δικαιώνεται σύντομα καθώς με την τεχνική αυτή η ομάδα ελέγχου θα είναι σε θέση να τρέξει εκατοντάδες διαφορετικές περιπτώσεις δοκιμών σε μικρό χρονικό διάστημα, μεταβάλλοντας απλώς κάθε φορά το αρχείο εισόδου στο πρόγραμμα. Η τεχνική ανάπτυξης προγραμμάτων αυτοματοποιημένου ελέγχου με γνώμονα τα δεδομένα είναι ιδανική για μεγάλα και πολύπλοκα συστήματα λογισμικού με μεγάλη διάρκεια ζωής.

5.4.2.5 Προγραμματα αυτοματοποιημενου ελεγχου που βασιζονται σε λεξεις-κλειδια

Η τεχνική ανάπτυξης προγραμμάτων αυτοματοποιημένων ελέγχων που βασίζονται σε λέξεις-κλειδιά είναι η λογική επέκταση της τεχνικής ανάπτυξης προγραμμάτων με γνώμονα τα δεδομένα. Ο κύριος περιορισμός που βασίζεται στα δεδομένα είναι πως οι ενέργειες που ακολουθεί το πρόγραμμα ελέγχου πρέπει να είναι ίδιες για κάθε διαφορετικό αρχείο δεδομένων δηλαδή για κάθε διαφορετικό σενάριο δοκιμών. Επιπλέον, τα αρχεία εισόδου και τα προγράμματα ελέγχου πρέπει συγχρονίζονται άριστα εφόσον η εκτέλεση κάθε ξεχωριστής περίπτωσης ελέγχου εξαρτάται και από τα δύο ταυτόχρονα. Μια καλύτερη μέθοδος, είναι να αφαιρέσουμε τις οδηγίες εκτέλεσης του ελέγχου από το κυρίως πρόγραμμα και να τις προσθέσουμε στο αρχείο εισόδου. Η τεχνική ανάπτυξης προγραμμάτων αυτοματοποιημένων ελέγχων που με γνώμονα λέξεις-κλειδιά συνδυάζει την μέθοδο της ανάπτυξης με γνώμονα τα αρχεία εισόδου με την ικανότητα να εξειδικεύει εύκολα κάθε σενάριο ελέγχου χωρίς να χρειάζεται να προσδιορίσει κάθε ξεχωριστή λεπτομέρεια. Επεκτείνει το αρχείο δεδομένων εισόδου, με σκοπό να μετατραπεί σε μια λεπτομερή περιγραφή κάθε ξεχωριστής δοκιμαστικής περίπτωσης χρησιμοποιώντας ένα σύνολο από λέξεις –κλειδιά που καθορίζουν επακριβώς τις εργασίες που θα πρέπει να γίνουν. Στην συνέχεια το πρόγραμμα ελέγχου θα πρέπει να είναι σε θέση να ερμηνεύσει τις λέξεις κλειδιά εκτός του κυρίου κώδικα, δημιουργώντας έτσι ένα ξεχωριστό επιπλέον επίπεδο ελέγχου, ξεχωριστό από την κύρια υλοποίηση. Το αρχείο δεδομένων πλέον, είναι μια λεπτομερής περιγραφή κάθε διαφορετικού σεναρίου ελέγχου. Στο ενδιάμεσο επίπεδο, ή αρχείο ελέγχου όπως λέγεται διαβάζει τις λέξεις –κλειδιά που δόθηκαν από το αρχείο εισόδου και καλεί το αντίστοιχο πρόγραμμα αυτοματοποιημένου ελέγχου που μπορεί να εκτελέσει το ζητούμενο σενάριο ελέγχου. Στο ενδιάμεσο επίπεδο, καθορίζεται ποια δοκιμαστική περίπτωση θα εκτελεστεί, όχι πως θα εκτελεστεί, δηλαδή περιγράφει επακριβώς ποια λειτουργία θα ελέγξει το πρόγραμμα και για ποια δεδομένα. Η τεχνική ανάπτυξης προγραμμάτων αυτοματοποιημένου ελέγχου με γνώμονα λέξεις-κλειδιά προσομοιάζει ουσιαστικά την περιγραφή που θα δινόταν σε έναν μηχανικό ελέγχου προκειμένου να εκτελέσει ένα σενάριο ελέγχου με το χέρι. Έτσι, είναι δυνατή η περιγραφή δοκιμαστικών περιπτώσεων χωρίς να επηρεάζεται το κυρίως πρόγραμμα και δίνεται η δυνατότητα σε κάποιον που δεν είναι προγραμματιστής να συμμετέχει στη διαδικασία ελέγχου χτίζοντας τα αρχεία εισόδου με τις λέξεις-κλειδιά, και ταυτόχρονα οι προγραμματιστές έχουν στη διάθεσή τους περισσότερο χρόνο να ασχοληθούν με την ανάπτυξη του κώδικα αυτοματοποιημένου ελέγχου. Φυσικά η συνεργασία μεταξύ των δύο είναι απαραίτητη, εφόσον η εργασία του ενός εξαρτάται από τις διαδικασίες που ακολουθεί ο άλλος. Τα οφέλη από την χρήση της τεχνικής ανάπτυξης που βασίζεται σε λέξεις-κλειδιά είναι πολλαπλά. Αρχικά, το πλήθος των προγραμμάτων αυτοματοποιημένου ελέγχου που θα πρέπει να αναπτυχθούν για να ελεγχθεί το λογισμικό στο σύνολό του εξαρτάται από την πολυπλοκότητα του ίδιου του λογισμικού και όχι από το πλήθος των δοκιμαστικών περιπτώσεων που θα πρέπει να εφαρμοστούν. Αυτό μειώνει κατά πολύ το κόστος δημιουργίας και συντήρησης των προγραμμάτων ελέγχου και επιταχύνει τον χρόνο του κύκλου ζωής του αυτοματοποιημένου ελέγχου. Επιπλέον, η αλλαγή του εργαλείου αυτοματοποιημένου ελέγχου που χρησιμοποιείται μπορεί να γίνει πολύ εύκολα γιατί τα δεδομένα που χρησιμοποιεί είναι ανεξάρτητα του περιβάλλοντος υλοποίησης του ελέγχου. Όλα τα δεδομένα και οι οδηγίες που χρειάζεται το πρόγραμμα για να εκτελέσει κάποιο είδος ελέγχου βρίσκονται σε ξεχωριστή θέση και δεν επηρεάζονται από την οποιαδήποτε αλλαγή στο εργαλείο αυτοματοποιημένου ελέγχου. Συνεπώς, τα δεδομένα

και η περιγραφή του ελέγχου είναι ασφαλή και μπορούν να χρησιμοποιηθούν σε οποιαδήποτε πλατφόρμα και περιβάλλον υλοποίησης. Κάποια εργαλεία αυτοματοποιημένου ελέγχου δίνουν τη δυνατότητα εισαγωγής δεδομένων και οδηγιών για κάθε ξεχωριστή περίπτωση ελέγχου από ξεχωριστή πηγή.
Συνοψίζοντας, οι τεχνικές συγγραφής προγραμμάτων αυτοματοποιημένου ελέγχου είναι παρόμοιες με τις κλασσικές τεχνικές προγραμματισμού. Μια καλή οργανωμένη σύγχρονη μέθοδος προγραμματισμού παράγει λογισμικά που συντηρούνται εύκολα και μπορούν να επαναχρησιμοποιηθούν. Το ίδιο ισχύει και για τις τεχνικές ανάπτυξης κώδικα αυτοματοποιημένου ελέγχου, οι οποίες εφόσον επιλεχθούν και οργανωθούν προσεχτικά, μπορούν να οδηγήσουν σε βέλτιστες μεθόδους ελέγχου[75]. Τα προγράμματα αυτοματοποιημένου ελέγχου περιέχουν δεδομένα και οδηγίες για το εργαλείο ελέγχου όπως πληροφορίες για τον συγχρονισμό και την σύγκριση με το πρόγραμμα ελέγχου, οδηγίες για την ανάγνωση και την αποθήκευση των δεδομένων καθώς και σημαντικές πληροφορίες για τη δομή του προγράμματος. Κάθε πρόγραμμα ελέγχου θα πρέπει να αντιστοιχεί σε πλήθος σεναρίων ελέγχου, αναλόγως τα δεδομένα εισόδου και τις οδηγίες που περιέχονται στο αρχείο εισόδου. Ένα καλό σενάριο ελέγχου θα πρέπει να περιέχει σχόλια, να είναι κατανοητό και πλήρως τεκμηριωμένο, ενώ θα πρέπει να είναι σε θέση να επαναχρησιμοποιηθεί είτε βραχυπρόθεσμα είτε μακροπρόθεσμα. Ο τρόπος με τον οποίο σχεδιάζονται οι δοκιμαστικές περιπτώσεις (test cases) επηρεάζει τα σενάρια ελέγχου (test scripts), ωστόσο η δομή των σεναρίων ελέγχου ενδέχεται να είναι διαφορετική από τη δομή κάθε ελέγχου. Το πιο σημαντικό και ουσιαστικό σε κάθε τέτοια διαδικασία είναι το πρόγραμμα ελέγχου να είναι κατανοητό, όπως επίσης και το έγγραφο τεκμηρίωσης που το συνοδεύει να έχει νόημα. Το πρότυπο τεκμηρίωσης θέλει τις πληροφορίες ελέγχου όπως το όνομα του σεναρίου ελέγχου, τα στοιχεία του προγραμματιστή, την ημερομηνία δημιουργίας ή επεξεργασίας του κώδικα, τον σκοπό, τις παραμέτρους εισόδου και εξόδου καθώς και το ιστορικό αλλαγών στον κώδικα, να τοποθετούνται στο πάνω μέρος του προγράμματος.

6

BLUETOOTH LOW ENERGY(BLE) / DIALOG SEMICONDUCTOR

6.1 BLUETOOTH LOW ENERGY

Ο κόσμος των ασύρματων συσκευών εξαπλώνεται ραγδαία μέρα με την ημέρα. Από την εφεύρεση των ραδιοφωνικών συσκευών, διάφορες εταιρείες και ερευνητικά κέντρα έχουν ένα κοινό στόχο: τη δημιουργία της πιο αποδοτικής, της πιο επιτυχημένης και συνεπώς της πιο ανταγωνιστικής ραδιοσυσκευής (radio module). Τα χαρακτηριστικά που έχει κάθε προϊόν ραδιοσυχνοτήτων εξαρτώνται από την λειτουργία του και τα προβλήματα που καλείται να επιλύσει. Σε ορισμένες περιπτώσεις, πρέπει να αποσταλούν μεγάλα κομμάτια δεδομένων μέσω ασύρματου δικτύου, οπότε οι οργανισμοί δεν ενδιαφέρονται για την κατανάλωση ενέργειας. Σε άλλες περιπτώσεις, οι οργανισμοί αναπτύσσουν μικρές φορητές συσκευές οι οποίες χρησιμοποιούν μπαταρία για τροφοδοσία και ως εκ τούτου οι συσκευές ραδιοσυχνοτήτων θα πρέπει να είναι όσο το δυνατόν πιο αποδοτικές ενεργειακά[76]. Στο σύνολο των ενεργειακά αποδοτικών μονάδων ραδιοσυχνοτήτων, οι οποίες αποστέλλουν το επιθυμητό ποσό δεδομένων καταναλώνοντας την μικρότερη δυνατή ενέργεια, ξεχωρίζει μια τεχνολογία ραδιοσυχνοτήτων: Bluetooth Low Energy. Όταν η Bluetooth[77] κυκλοφόρησε το είδος ραδιοσυχνοτήτων Bluetooth 4.0 core, εισήγαγε έναν νέο παίκτη στο πεδίο των μονάδων ραδιοσυχνοτήτων, την τεχνολογία Bluetooth Low Energy ή BLE ή αλλιώς Bluetooth Smart. Το BLE ξεκίνησε σαν ένα project από την εταιρεία Nokia, με το όνομα "Wibree", ενώ το 2006 εισήχθη στην αγορά με το κανονικό του όνομα. Το 2010, η ειδική ομάδα ενδιαφέροντος Bluetooth (Bluetooth Special Interest Group-SIG), συγχώνευσε το WIbree με το πρότυπο Bluetooth ως μέρος προδιαγραφής της έκδοσης Bluetooth 4.0 core. Παρότι που αποτελεί μέρος της ίδιας προδιαγραφής, το BLE δεν είναι συμβατό με το bluetooth και επομένως δεν μπορεί να αναλυθεί ακολουθώντας το ίδιο πρότυπο. Αυτό που κάνει το Ble τόσο ξεχωριστό είναι ότι μπορεί να επικοινωνήσει με ένα μεγάλο αριθμό κινητών συσκευών που κυκλοφορούν στην αγορά σήμερα: κινητά τηλέφωνα ή tablets που χρησιμοποιούν Android, OS X, Windows Phone, iOS kai BlackBerry καθώς επίσης και υπολογιστές με λειτουργικό σύστημα Linux ή Windows 8. Το BLE έχει αναπτυχθεί με τέτοιο τρόπο έτσι ώστε να έχει τη δυνατότητα να εφαρμοστεί και να διευκολύνει την επικοινωνία διαφορετικών συσκευών. Το BLE δημιουργήθηκε με τέτοιο τρόπο έτσι ώστε οποιοσδήποτε επιθυμεί να στείλει κάποια δεδομένα, να μπορεί να το χρησιμοποιεί όποτε και όπως θέλει, κάτι που δεν συμβαίνει με την τεχνολογία Bluetooth η οποία σχεδιάστηκε για να εφαρμόζεται σε ειδικές περιπτώσεις χρήσης. Συνοπτικά το BLE είναι η κατάλληλη επιλογή για ραδιοεπικοινωνία χαμηλής ενέργειας καθώς μπορεί να στείλει οποιαδήποτε δεδομένα, είναι ενεργειακά αποδοτική τεχνολογία και μπορεί να συνδεθιεί με όλες τις μεγάλες πλατφόρμες της αγοράς. Στις επόμενες παραγράφους αναλύεται η λειτουργία και ο τρόπος εφαρμογής του BLE.

6.1.1 ΔΥΝΑΜΙΚΟΤΗΤΑ ΚΑΙ ΕΥΡΟΣ ΔΕΔΟΜΕΝΩΝ

Ο ρυθμός μεταφοράς δεδομένων του Bluetooth Low Energy καθορίζεται από τις προδιαγραφές στην σταθερή ταχύτητα 1 Mbps. Αυτό είναι το θεωρητικό άνω όριο, ωστόσο στην πραγματικότητα αποστέλλονται 5-10 KB ανά δευτερόλεπτο, αναλόγως τους περιορισμούς των συσκευών που χρησιμοποιούνται. Όσον αφορά την εμβέλεια, το BLE επικεντρώνεται στην επικοινωνία συσκευών σε μικρή απόσταση. Είναι δυνατή η δημιουργία και διαμόρφωση μιας BLE συσκευής που μπορεί να μεταδίδει αξιόπιστα δεδομένα σε απόσταση 30 μέτρων ή περισσότερο, ωστόσο ένα τυπικό εύρος λειτουργίας είναι πιθανότητα ανάμεσα στα 2 με 5 μέτρα. Φυσικά όσο μεγαλύτερη είναι η εμβέλεια, τόσο μεγαλύτερη είναι η κατανάλωση ενέργειας. Ο στόχος των εταιρειών παραγωγής συσκευών BLE είναι η αποστολή δεδομένων με την μέγιστη δυνατή απόσταση μεταξύ των συσκευών αλλά ταυτόχρονα την κατανάλωση της ελάχιστης δυνατής ενέργειας.

6.1.2 ΔΟΜΙΚΑ ΣΤΟΙΧΕΙΑ BLE

Το BLE, όπως όλες οι τεχνολογίες ασύρματης επικοινωνίας οργανώνεται σε επίπεδα. Κάθε επίπεδο έχει ξεχωριστό σκοπό και παίζει σημαντικό ρόλο στην ανάπτυξη και την σωστή λειτουργία της συσκευής BLE. Το BLE είναι δομημένο σε τρεις κύριες δομικές μονάδες: εφαρμογή, host και controller[78], όπως φαίνεται και στην Εικόνα 15. Στο επίπεδο της εφαρμογής, είναι η εφαρμογή χρήστη που διασυνδέεται με την στοίβα πρωτοκόλλου του Bluetooth. Η εφαρμογή ανήκει στο ανώτερο επίπεδο οργάνωσης της συσκευής BLE και είναι υπεύθυνη για τη διεπαφή χρήστη και την επεξεργασία δεδομένων που σχετίζονται με την τελική λειτουργία της εφαρμογής καθώς και την εμφάνισή της. Συνδέεται με το επόμενο επίπεδο, τον host, μέσω του πρωτοκόλλου API. Η δομή της εφαρμογής εξαρτάται από τον τελικό σκοπό της συσκευής BLE. Ο host καλύπτει τα ανώτερα στρώματα της στοίβας πρωτοκόλλου και επικοινωνεί με την μονάδα BLE μέσω μιας προσθήκης που ονομάζεται Host controller interface (HCI). Ο σκοπός του HCI είναι η διασύνδεση του host με τον controller και συνήθως χρησιμοποιεί την μνήμη SPI για να τον επιτύχει. Ο controller καλύπτει τα κατώτερα στρώματα της συσκευής BLE. Αναλυτικά, ο host περιέχει τα παρακάτω επίπεδα:

- Generic Access Profile (GAP)
  Το GAP ελέγχει τη διαδικασία advertising, δηλαδή τη διαδικασία κατά την οποία μια συσκευή δηλώνει διαθέσιμη για σύνδεση και αποστολή πακέτων, καθώς και τις συνδέσεις. Προσδιορίζει τον τρόπο που η συσκευή εκτελεί τις διαδικασίες ελέγχου όπως την ανακάλυψη συσκευών (device discovery), την σύνδεση, την εγκατάσταση δεδομένων ασφαλείας κλπ. Το GAP επικεντρώνεται στους ρόλους των συσκευών αλλά και την αλληλεπίδραση μεταξύ τους, στις ξεχωριστές λειτουργίες και τον τρόπο μετάβασης σε αυτές, στις διαδικασίες επίτευξης συνεπούς και διαλειτουργικής επικοινωνίας, στις πτυχές ασφαλείας καθώς και στις πρόσθετες μορφές δεδομένων, πέρα από τα δεδομένα που σχετίζονται με τα πρωτόκολλα.
- Generic Attribute Profile (GATT)
  Το GATT βρίσκεται ακριβώς από πάνω από το ATT στην ιεραρχία. Προσθέτει ένα μοντέλο δεδομένων, ιεραρχεί και καθορίζει τον τρόπο οργάνωσης και ανταλλαγής των δεδομένων. Τα δεδομένα στο GATT οργανώνονται σε services. Κάθε service περιέχει ένα ή περισσότερα χαρακτηριστικά και κάθε χαρακτηριστικό διαχωρίζεται σε descriptors, όπως φαίνεται στην Εικόνα 16. Μαζί με το GAP, το GATT, αποτελεί την κύρια διασύνδεση σε μια στοίβα πρωτοκόλλου BLE. Οι υπηρεσίες του GATT είναι οργανωμένες σε κάτι που αποκαλείται προφίλ GATT και κάθε προφίλ μπορεί να περιέχει πολλές υπηρεσίες. Οι υπηρεσίες διακρίνονται μεταξύ τους χρησιμοποιώντας ένα μοναδικό UUID των 16 δυαδικών ψηφίων[77]. Κάθε χαρακτηριστικό έχει επίσης ένα μοναδικό UUID.
- Logical Link Control and Adaptation Protocol (L2CAP)
  Το επίπεδο L2CAP είναι υπεύθυνο για δύο εργασίες. Παίρνει πολλαπλά πρωτόκολλα από τα ανώτερα στρώματα και τα ενσωματώνει στην τυπική μορφή των πακέτων BLE (και αντιστρόφως). Παίρνει μεγάλα πακέτα από τα ανώτερα στρώματα και τα χωρίζει σε κομμάτια που χωρούν στο μέγεθος του μέγιστου μεγέθους πακέτων BLE στην πλευρά της μετάδοσης και αντιστρόφως λαμβάνει πολλαπλά πακέτα που έχουν κατακερματιστεί και τα ενώνει σε ένα μεγάλο πακέτο που θα αποσταλεί σε ανώτερο επίπεδο. Το επίπεδο L2CAP είναι υπεύθυνο για τη δρομολόγηση δύο κύριων πρωτοκόλλων: το attribute protocol (ATT) και το security manager protocol (SMP).
- Attribute Protocol (ATT)
  Αποτελεί την βάση ανταλλαγής δεδομένων μεταξύ εφαρμογών BLE. Είναι ένα απλό πρωτόκολλο client/server που βασίζεται στα χαρακτηριστικά που παρουσιάζει μια συσκευή. Ένας client ζητά δεδομένα από τον server και ο server στέλνει δεδομένα στους clients. Εάν εκκρεμεί μια αίτηση, δεν μπορούν να σταλούν περαιτέρω αιτήματα έως ότου απαντηθεί. Κάθε server περιέχει δεδομένα οργανωμένα με την μορφή χαρακτηριστικών, κάθε ένα εκ των οποίων προσδιορίζεται με ένα μοναδικό UUID, δηλαδή έναν αριθμό 16 δυαδικών ψηφίων, ένα σύνολο δικαιωμάτων και μια τιμή. Η εν λόγω τιμή είναι απλώς ένα αναγνωριστικό που χρησιμοποιείται για την πρόσβαση σε ένα χαρακτηριστικό. Το UUID χρησιμοποιείται για τον προσδιορισμό του τύπου και της φύσης δεδομένων. Ο πελάτης στέλνει τις κατάλληλες εντολές εγγραφής ή ανάγνωσης και ο server ανταποκρίνεται σύμφωνα με αυτά. Όταν ο πελάτης θέλει να διαβάσει ή να γράψει τιμή σε χαρακτηριστικά του server στέλνει μια αίτηση ανάγνωσης ή εγγραφής και στην συνέχεια ο sever ανταποκρίνεται με την τιμή χαρακτηριστικού ή μια απάντηση αναγνώρισης αντιστοίχως. Κατά τη διαδικασία ανάγνωσης, ο πελάτης αναλύει την τιμή που λαμβάνει και κατανοεί τον τύπο δεδομένων με βάση το UUID που διαβάζει. Κατά τη διαδικασία εγγραφής, ο πελάτης γράφει δεδομένα σε κάποιο χαρακτηριστικό και ο server έχει τη δυνατότητα αποδοχής ή απόρριψης της εγγραφής. Το σύνολο των λειτουργιών που εκτελούνται μέσω του πρωτοκόλλου ATT είναι: Error Handling, Server Configuration, Find Information, Read Operations, Write Operations, Queued Writes, Server Initiated.
- Security Manager protocol (SMP)

Παρέχει ένα πλαίσιο για τη δημιουργία και τη διανομή κλειδιών ασφαλείας.

- Host Controller Interface (HCI), Host side

Ο controller περιέχει τα παρακάτω επίπεδα:

- Host Controller Interface (HCI), Controller side
  Το επίπεδο HCI επιτρέπει σε πιο ισχυρούς επεξεργαστές να ελέγχουν την συσκευή BLE μέσω μιας σειριακής διεπαφής, συνήθως UART ή USB. Ένα τυπικό παράδειγμα περιλαμβάνει τα περισσότερα smartphones, tablets και υπολογιστές, όπου το επίπεδο του host υλοποιείται στην κύρια CPU, ενώ ο controller βρίσκεται σε ξεχωριστό chip, το οποίο συνδέεται με τον host μέσω UART ή USB.
- Link Layer (LL)
  Συνδέεται άμεσα με το physical layer και συνήθως υλοποιείται ως συνδυασμός προσαρμοσμένου υλικού (hardware) και λογισμικού. Το συγκεκριμένο επίπεδο ορίζει τους ακόλουθους ρόλους για κάθε συσκευή:
  a) Advertiser
     Είναι η συσκευή που στέλνει πακέτα advertising, δηλαδή δεδομένα που δηλώνουν πως η συσκευή είναι έτοιμη για σύνδεση.
  b) Scanner
     Είναι η συσκευή που σαρώνει τα πακέτα advertising.
  c) Master
     Η συσκευή που ξεκινά και διαχειρίζεται την σύνδεση.
  d) Slave
     Η συσκευή που δέχεται αίτημα σύνδεσης και ακολουθεί το χρονοδιάγραμμα του master.

  Το link layer ασχολείται και με τον ορισμό της διεύθυνσης της συσκευής Bluetooth (Bluetooth Device Address), δηλαδή καθορίζει έναν αριθμό που αποτελείται από 48 ψηφία και είναι μοναδικός για κάθε συσκευή. Ο αριθμός αυτός είναι κάτι σαν τη διεύθυνση MAC σε μια IP. Ένα παράδειγμα διεύθυνσης είναι: 80:24:DB:03:DA:00. Το Link layer είναι επίσης υπεύθυνο για τη δημιουργία συνδέσεων, το φιλτράρισμα πακέτων έτσι ώστε να μεταφέρονται πακέτα που αντιστοιχούν στην σωστή διεύθυνση και καθορισμό του χρονικού ορίου διασύνδεσης. Το χρονικό όριο διασύνδεσης είναι ο χρόνος μεταξύ δύο διαδοχικών συμβάντων σύνδεσης. Το link layer μπορεί επίσης να ρυθμίσει την κρυπτογράφηση δεδομένων, λειτουργία ιδιαίτερα σημαντική, ειδικά σε περιοχές όπου συγκεντρώνονται πολλές συσκευές ταυτόχρονα.
- Physical Layer (PHY)
  Περιέχει το κύκλωμα αναλογικών επικοινωνιών που χρησιμοποιείται για τη διαμόρφωση και την αποδιαμόρφωση των αναλογικών σημάτων και τη μετατροπή τους σε ψηφιακά σύμβολα. Το BLE μπορεί να επικοινωνήσει πάνω από 40 κανάλια από 2.400 GHz έως 2.4834 GHz, όπου τα 37 από αυτά τα κανάλια χρησιμοποιούνται για δεδομένα σύνδεσης, ενώ τα τρια τελευταία (37,38,39) χρησιμοποιούνται για την περιγραφή των καναλιών αποστολής (advertising channels), τις ρυθμίσεις σύνδεσης και την αποστολή των δεδομένων μετάδοσης (broadcast data). Το BLE χρησιμοποιεί μια τεχνική που ονομάζεται εύρος διασποράς συχνότητας (frequency hopping spread spectrum), όπου τα ραδιοκύματα μεταπηδούν μεταξύ των καναλιών κατά τη διάρκεια κάθε σύνδεσης. Σε κάθε σύνδεση χρησιμοποιούνται διαφορετικά ραδιοκύματα και έτσι μειώνεται η πιθανότητα να συμβεί οποιαδήποτε παρεμβολή.

6.1.3 ΤΟΠΟΛΟΓΙΑ ΔΙΚΤΥΟΥ

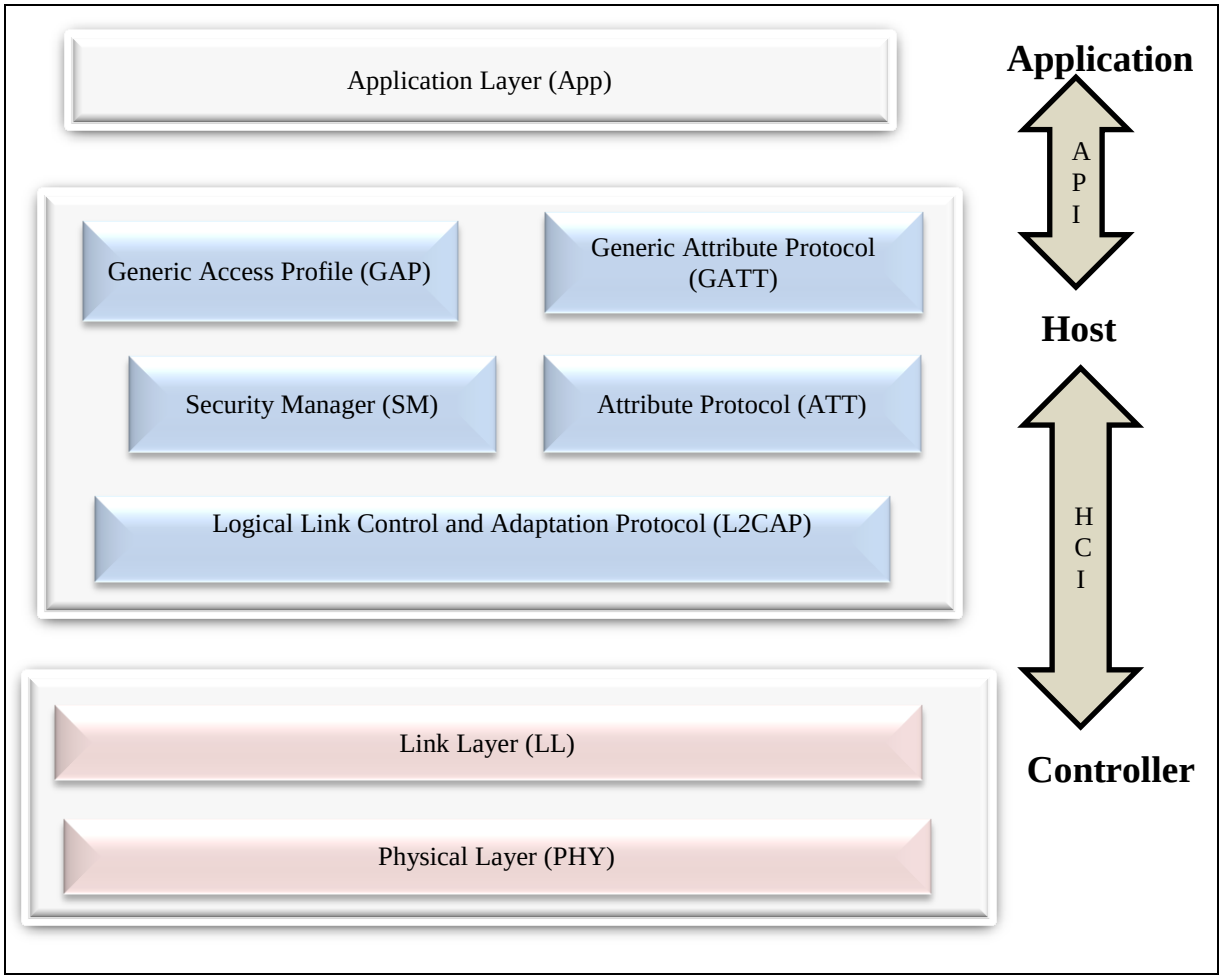
6.1.3.1 BROADCAST TOPOLOGY

Οι συσκευές BLE μπορούν να έχουν δύο διαφορετικούς ρόλους, είτε ως κεντρικές συσκευές, είτε ως περιφερειακές συσκευές. Οι κεντρικές συσκευές είναι συνήθως κινητά τηλέφωνα ή υπολογιστές με υψηλότερη ισχύς επεξεργασίας στην CPU. Οι περιφερειακές συσκευές είναι συνήθως μερικοί αισθητήρες ή συσκευές χαμηλής ισχύος που συνδέονται με την κεντρική συσκευή. Μια συσκευή BLE μπορεί να στείλει δύο τύπους δεδομένων: πακέτα δεδομένων advertising και πακέτα δεδομένων scan response. Τα πακέτα δεδομένων advertising είναι απαραίτητα και μεταδίδονται συνεχώς από μια περιφερειακή συσκευή προκειμένου να κάνουν αισθητή την παρουσία της παρούσας συσκευής στις υπόλοιπες. Όταν κάποια συσκευή λάβει τα δεδομένα advertising, μπορεί να ζητήσει επιπλέον δεδομένα από την περιφερειακή συσκευή, η οποία στην συνέχεια στέλνει τα πακέτα δεδομένων scan response. Μια συσκευή BLE μπορεί να επικοινωνήσει με τις κοντινές συσκευές με δύο τρόπους: με broadcasting και με connections. Ο πρώτος τρόπος επικοινωνίας γίνεται με αποστολή δεδομένων από μια συσκευή προς όλες τις υπόλοιπες που αντιλαμβάνονται την παρουσία της. Κατά τον τρόπο επικοινωνίας broadcasting, η συσκευή μπορεί να έχει δύο ρόλους: broadcaster ή observer. Ο broadcaster αποστέλλει περιοδικά μη συνδεδεμένα advertising πακέτα σε οποιονδήποτε είναι διατεθειμένος να

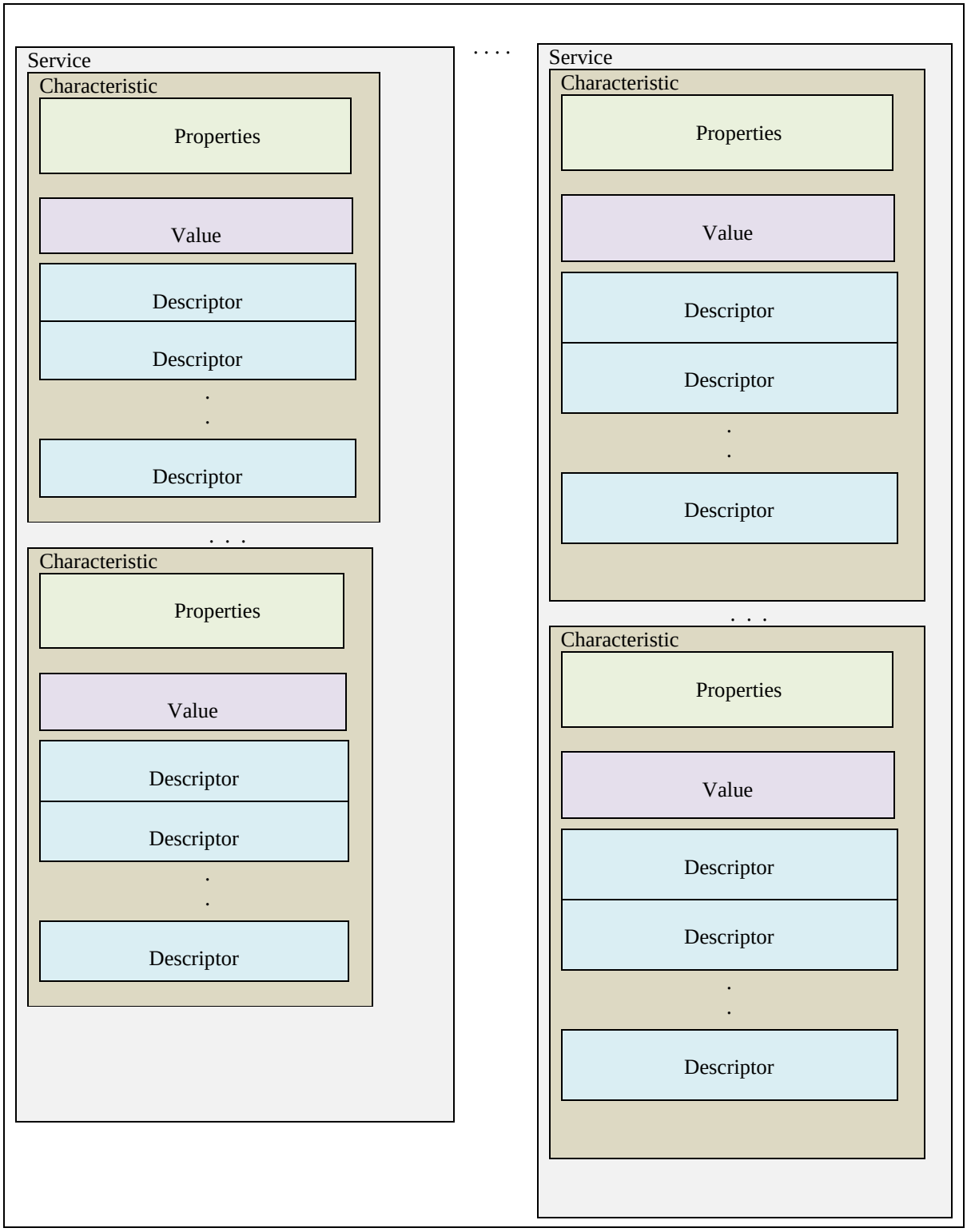
τα δεκτή. Ο observer σαρώνει επανειλημμένα την περιοχή προκειμένου να λάβει πακέτα. Στη συνέχεια, όταν κάποιος observer λάβει ένα advertising πακέτο, μπορεί να ζητήσει να του αποσταλούν τα scan response data. Είναι σημαντικό να σημειωθεί εδώ πως η μέθοδος επικοινωνίας broadcasting είναι ο μόνος τρόπος με τον οποίο μια συσκευή μπορεί να μεταδώσει δεδομένα σε περισσότερες από μια κοντινές συσκευές, κάθε φορά που το επιχειρεί.

6.1.3.2 CONNECTED TOPOLOGY

Μια σύνδεση είναι μια μόνιμη, περιοδική ανταλλαγή πακέτων δεδομένων μεταξύ δύο συσκευών. Η κεντρική συσκευή (master) σαρώνει τις συχνότητες κοντινών advertising πακέτων και όταν είναι εφικτό, ενεργοποιεί μια σύνδεση. Μόλις δημιουργηθεί μια σύνδεση, η κεντρική συσκευή διαχειρίζεται το χρονισμό και ξεκινά τις περιοδικές ανταλλαγές δεδομένων. Η περιφερειακή συσκευή (slave), στέλνει περιοδικά advertising πακέτα και δέχεται εισερχόμενες συνδέσεις. Μόλις ενεργοποιηθεί μια σύνδεση, η περιφερειακή συσκευή ακολουθεί το χρονισμό της κεντρικής συσκευής και ανταλλάσσει ανα τακτά χρονικά διαστήματα και για όσο υπάρχει σύνδεση, δεδομένα μάζι του. Όταν οι δύο συσκευές συνδεθούν, ορίζουν ένα σημείο έναρξης ανταλλαγής δεδομένων που λέγεται συμβάν σύνδεσης (connection event). Ένα συμβάν σύνδεσης είναι μια περιοδική ανταλλαγή δεδομένων σε συγκεκριμένα χρονικά σημεία. Αυτό αποτελεί και το βασικό όφελος των συσκευών BLE· δύο συσκευές μπορούν να συνδεθούν, να ανταλλάξουν δεδομένα και στην συνέχεια να ενεργοποιούν την λειτουργία sleep mode μέχρι το επόμενο συμβάν σύνδεσης, εξοικονομώντας έτσι ενέργεια.



Εικόνα 15: Δομικά στοιχεία Bluetooth Low Energy



Εικόνα 16: Ιεραρχία GATT

6.2 DIALOG SEMICONDUCTOR

Η τεχνολογία Bluetooth Low Energy (BLE) είναι η ευφυής, χαμηλής κατανάλωσης έκδοση της κλασσικής ασύρματης τεχνολογίας Bluetooth. Η αποδοτικότητα ισχύος που πετυχαίνει η τεχνολογία BLE, την κάνουν ιδανική για συσκευές που καταναλώνουν ελάχιστη μπαταρία όταν δεν χρησιμοποιούνται με τον εξής τρόπο: η συσκευή μπαίνει σε λειτουργία sleep mode όταν δεν χρησιμοποιείται για μεγάλο χρονικό διάστημα. Το μεγάλο πλεονέκτημα των συσκευών BLE είναι η ικανότητά τους να επικοινωνούν με πλήθος smarthphones και tablets που ήδη υπάρχουν στην αγορά. Η τεχνολογία αυτή έχει την ικανότητα να συνδέει δισεκατομμύρια φορητές συσκευές και συσκευές που τροφοδοτούνται με μπαταρίες, με τον κόσμο του Internet of Things. Τέλος, η τεχνολογία BLE τροφοδοτεί πολλά είδη σύγχρονων συσκευών όπως wireless human interface devices, wearables και smart home.

Η εταιρεία Dialog Semiconductor[2] είναι μια εταιρεία ημιαγωγών που επικεντρώνεται κυρίως στην ανάπτυξη ολοκληρωμένων προϊόντων μικτού σήματος για τα ηλεκτρονικά είδη ευρείας κατανάλωσης. Οι τεχνολογίες που έχει αναπτύξει η εταιρεία, συμβάλλουν στην επέκταση της διάρκειας ζωής των μπαταριών σε φορητές συσκευές, στην γρήγορη και ασφαλή φόρτιση των μπαταριών και στην παροχή αποτελεσματικής σύνδεσης στις εφαρμογές IoT (Internet of Things).

6.2.1 DIALOG SEMICONDUCTOR SMARTBOND™ PRODUCTS

Η οικογένεια προϊόντων SMARTBOND™ της εταιρείας Dialog Semiconductor[79], παρέχει τη διαδρομή για τα πιο εύχρηστα Bluetooth Low Energy προϊόντα τα οποία εξοικονομούν βέλτιστα την ενέργεια της μπαταρίας. Στον Πίνακας 6: Παρουσίαση προϊόντων της Dialog Semiconductor αναφέρονται τα προϊόντα της εταιρείας καθώς και οι ιδιότητες κάθε προϊόντος.

Πίνακας 6: Παρουσίαση προϊόντων της Dialog Semiconductor

Προϊόν	**Μέγεθος και ιδιότητες μνήμης**	**Διαστάσεις**	**Χαρακτηριστικά**	**Εφαρμογές**
DA14580	ROM 84 kB OTP 32 kB RAM 50 kB Support for 1 MB Flash memory	WL-CSP34 size 2.5x2.5x0.5mm, pitch 0.4mm, QFN40 size 5x5x0.9mm, pitch 0.4mm	Bluetooth 4.2 Cortex M0 application processor Power supply 0.9 -3.6 V Single pin RF I/O Rich set of analog and digital peripherals Over The Air update	Beacon & proximity Health & Fitness HID Smart Home
DA14581	ROM 84 kB OTP 32 kB RAM 50 kB Support for 1 MB Flash memory	WL-CSP34 size 2.5x2.5x0.5mm, pitch 0.4mm QFN40 size 5x5x0.9mm, pitch 0.4mm	Bluetooth 4.2 Cortex M0 application processor Power supply 0.9 -3.6 V Single pin RF I/O Rich set of analog and digital peripherals 8 connections Optimized boot time Over The Air update	Wireless charging (A4WP) HCI
DA14582	ROM 84 kB OTP 32 kB RAM 50 kB	QFN56 size 8x8x0.9mm, pitch 0.5mm	Bluetooth 4.2 Cortex M0 application processor Power supply 2.35 - 3.3 V Single pin RF I/O Rich set of analog and digital peripherals Wideband analog codec input and output Compatible with FR1-type PCB Over The Air update	Remote controls with voice commands over BLE

DA14583	FLASH 1 Mb ROM 84 kB OTP 32 kB RAM 50 kB	QFN40 size 5x5x0.9mm, pitch 0.4mm	Bluetooth 4.2 Cortex M0 application processor 1 Mb Flash Power supply 2.35 - 3.3 V Single pin RF I/O Rich set of analog and digital peripherals Over The Air update	Applications requiring software upgrade Over The Air (OTA)
DA14585	FLASH ROM OTP RAM	WL-CSP34 QFN40	Complies to Bluetooth 5 core specification Extended user RAM (96 kB) Wide operating range (0.9 V to 3.6 V) I2S and PDM audio interfaces	Remote controls Proximity tags and trackers Beacons Connected medical devices Smart home Human Interface Devices VR controllers Connected sensors Wireless charging
DA14586	FLASH ROM OTP RAM	QFN40	Complies to the Bluetooth 5 core specification Integrated 2 Mb Flash Extended user RAM (96 kB) Low operating voltage (1.8 V to 3.6 V) I2S and PDM audio interfaces	Remote controls Proximity tags and trackers Beacons Connected medical devices Smart home Human Interface Devices VR controllers Connected sensors Wireless charging
DA14680	FLASH 8 Mb RAM 144 kB OTP 64 kB ROM 128 kB	AQFN60 size 6x6x0.9mm, pitch 0.5mm	Bluetooth 4.2 Cortex M0 application processor Dedicated HW crypto engine Integrated battery and system PMU Power supply 1.7 -5.5 V Single pin RF I/O Rich set of analog and digital peripherals 8 Mbit executable Flash	Wearables Smart Home HID Other rechargeable devices

DA14681	RAM 144 kB OTP 64 kB ROM 128 kB	WL-CSP53 size 3.4x3.1x0.5mm, pitch 0.5mm AQFN60 size 6x6x0.9mm, pitch 0.5mm	Bluetooth 4.2 Cortex M0 application processor Dedicated HW crypto engine Integrated battery and system PMU Power supply 1.7 -5.5 V Single pin RF I/O Rich set of analog and digital peripherals QSPI interface	Wearables Smart Home HID Other rechargeable devices

6.3 DA14585 SOFTWARE PLATFORM OVERVIEW

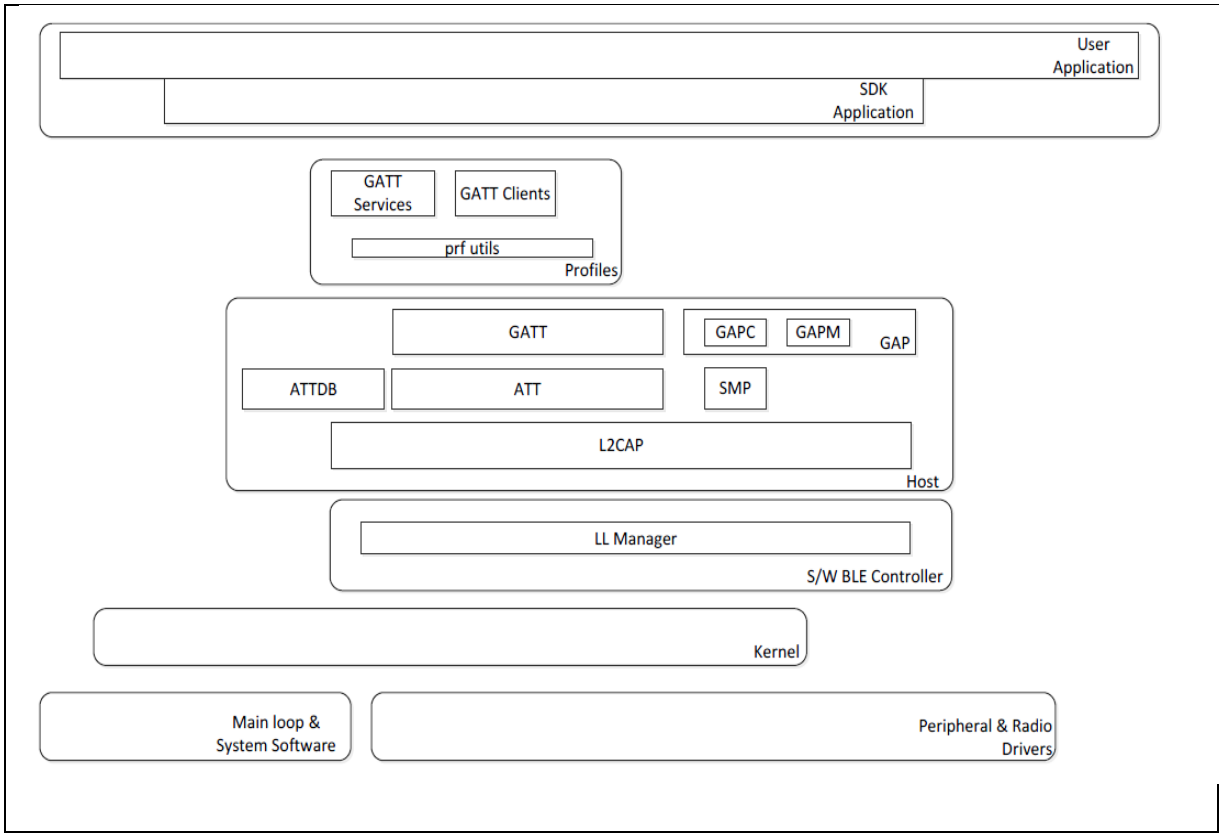
Στην παράγραφο αυτή παρουσιάζεται ολοκληρωμένα η πλατφόρμα και η αρχιτεκτονική του λογισμικού που υποστηρίζει το chip DA14585. Ένα διάγραμμα που περιγράφει σχηματικά την αρχιτεκτονική λογισμικού και τα επίπεδα του προϊόντος φαίνεται στην Εικόνα 17. Παρακάτω, δίνεται μια περιγραφή της λειτουργίας κάθε επιπέδου.

6.3.1 PERIPHERALS ΚΑΙ RADIO DRIVERS

Το σύστημα λογισμικού ενσωματώνει έναν αριθμό προγραμμάτων οδήγησης (drivers) για τις περιφερειακές συσκευές, τις συχνότητες και ορισμένα μπλοκ υλικού hardware του BLE. Οι προγραμματιστές εφαρμογών είναι σε θέση να χρησιμοποιούν τους οδηγούς για τις περιφερειακές συσκευές (peripheral drivers), ωστόσο, τα υπόλοιπα προγράμματα οδήγησης μπορούν να χρησιμοποιηθούν μόνο από το λογισμικό πυρήνα και τον ελεγκτή BLE (BLE controller software).

6.3.2 REAL TIME KERNEL

Η πλατφόρμα λογισμικού DA14585 χρησιμοποιεί ένα μικρό και αποδοτικό πυρήνα πραγματικού χρόνου. Σχεδόν η πλήρης στοίβα BLE και το μεγαλύτερο μέρος της εφαρμογής κάνουν χρήση των υπηρεσιών που προσφέρει ο πυρήνας πραγματικού χρόνου. Ο πυρήνας προσφέρει εργασίες(tasks), μηνύματα(messages), γεγονότα(events) και ικανότητες δυναμική μνήμης (dynamic memory capabilities). Οι εργασίες επικοινωνούν με μηνύματα που ωθούνται σε μια ουρά κάθε φορά που μια εργασία προσπαθεί να στείλει μήνυμα σε μια άλλη εργασία. Οι χρονοδιακόπτες (timers) και άλλα συμβάντα που προέρχονται από το υλικό ωθούνται επίσης σε μια ουρά συμβάντων. Όταν ο προγραμματιστής πυρήνα καλείται από τον κύριο βρόχο, εμφανίζονται μηνύματα και γεγονότα από τις ουρές με την σειρά που έχουν ταξινομηθεί στο εσωτερικό κάθε ουράς. Έπειτα καλείται το αντίστοιχο κομμάτι κώδικα που χειρίζεται το αντίστοιχο συμβάν και εκτελείται. Η εκτέλεση συνεχίζεται μέχρις ότου αδειάσουν οι ουρές.



Εικόνα 17: Αρχιτεκτονική λογισμικού DA14585

6.3.3 BLUETOOTH LOW ENERGY SOFTWARE

Το λογισμικό BLE εφαρμόζει το πρωτόκολλο χαμηλής ενέργειας Bluetooth® Low Energy όπως καθορίζεται στην έκδοση 4.2 του Bluetooth® και συμμορφώνεται πλήρως με αυτό το πρότυπο. Πρόκειται για μία εφαρμογή BLE μονής λειτουργίας συνεπώς δεν υπάρχει υποστήριξη για το πρωτόκολλο Basic Rate / Enhanced Data Rate protocol (BR/EDR). Τα βασικά μέρη του λογισμικού BLE αναπτύχθηκαν στην παράγραφο 6.1.2. Κάθε επίπεδο (ATT, GATT, GAP κλπ) της στοίβας αποτελείται από πολλές διαφορετικές εργασίες. Τα συμβάντα που γίνονται στον αέρα καταγράφονται από του drivers και εισάγονται στο λογισμικό χαμηλού επιπέδου του ελεγκτή BLE. Οι χειριστές (handlers) δημιουργούν μηνύματα που ενεργοποιούν τα επίπεδα του πρωτοκόλλου. Ένα υπάρξει συμβάν από την πλευρά του χρήστη στέλνετε μια ακολουθία μηνυμάτων στην εφαρμογή και στο profile, όπως φαίνεται στην Εικόνα 17. Στην αντίστροφη κατεύθυνση, εάν η εφαρμογή χρήστη θέλει να εκτελέσει μια συγκεκριμένη λειτουργία, στέλνει μηνύματα στα κατώτερα επίπεδα, ξεκινώντας με μια ακολουθία μηνυμάτων που μπορούν να φτάσουν παρακάτω, αναλόγως την λειτουργία, τον ελεγκτή BLE και το radio. Στην κορυφή του host παρέχεται μια βιβλιοθήκη profile έτοιμων προς χρήση και ορισμένα βοηθητικά προγράμματα σχετικά με το profile, με σκοπό την απλούστευση της ανάπτυξης εφαρμογής. Οι προγραμματιστές θα πρέπει να αλληλοεπιδρούν με το profile, τις μονάδες GATT ΚΑΙ GAP προκειμένου να υλοποιήσουν όλες τις λειτουργίες που απαιτούνται από μια εφαρμογή BLE.

6.3.4 APPLICATION SOFTWARE

Παρόλο που τα μηνύματα API (Application Programming Interface) που παρέχονται είναι ολοκληρωμένα και πλήρη, υπάρχουν κάποιοι περιορισμοί στο κατά πόσο γρήγορα και εύκολα μπορεί να κατασκευαστεί μια εφαρμογή. Για την αντιμετώπιση αυτού του προβλήματος έχει δημιουργηθεί ένα νέο επίπεδο στη δομή του DA14585, το SDK(software development Kit), το οποία συλλέγει μέρος λειτουργιών που απαιτούνται για την κατασκευή μιας εφαρμογής. Οι λειτουργίες αυτές συνιστούν μια βιβλιοθήκη που περιέχει μια σειρά από APIs και βοηθητικές συναρτήσεις. Σκοπός της τεχνικής αυτής είναι η απόκρυψη από τον χρήστη, όσο περισσότερων γίνεται εργασιών και μηνυμάτων που απαιτούνται για τη διαχείριση της εφαρμογής.

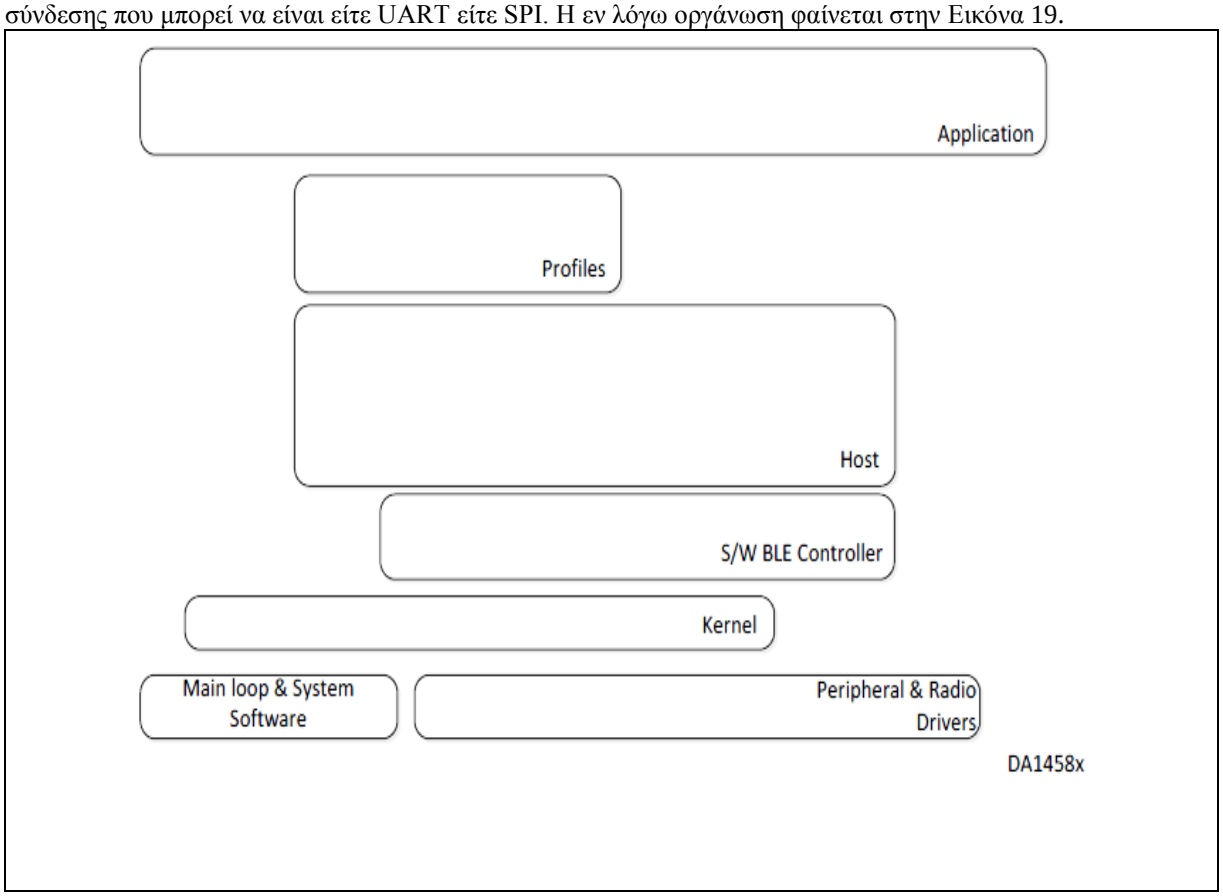
6.3.5 SUPPORTED HARDWARE CONFIGURATIONS

6.3.5.1 ΕΝΣΩΜΑΤΩΜΕΝΟΣ ΕΠΕΞΕΡΓΑΣΤΗΣ

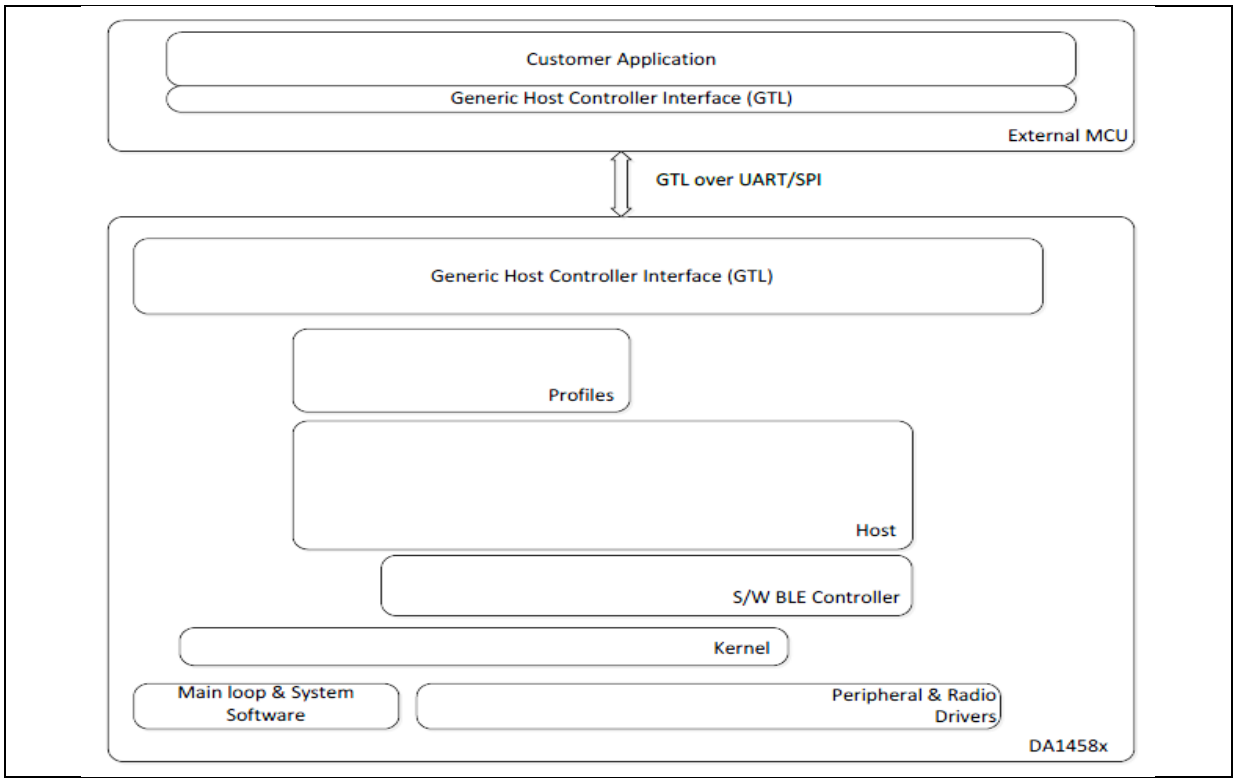
Η απλή προσέγγιση στην ανάπτυξη εφαρμογών με το DA14585 είναι η ολοκληρωμένη διαμόρφωση επεξεργαστή. Όλα τα στοιχεία λογισμικού, τα κατώτερα επίπεδα (controller), τα ανώτερα επίπεδα (host), τα profiles και η ολοκληρωμένη εφαρμογή λειτουργούν μόνο με το DA14585 chip. Η επιλογή αυτή, δηλαδή όλα να λειτουργούν και να τρέχουν από ένα chip, κάνει το DA14585 κατάλληλο για πολλές εφαρμογές χαμηλής έως μέσης πολυπλοκότητας. Η εν λόγω οργάνωση φαίνεται στην Εικόνα 18.

6.3.5.2 ΕΞΩΤΕΡΙΚΟΣ ΕΠΕΞΕΡΓΑΣΤΗΣ

Σε περιπτώσεις που υπάρχει εξωτερικός επεξεργαστής ή για εφαρμογές μεσαίας και υψηλής πολυπλοκότητας, το DA14585 μπορεί να χρησιμοποιηθεί ως διεπαφή BLE που ελέγχεται από εξωτερικό επεξεργαστή μέσω του πρωτοκόλλου Generic Layer Transport (GTL). Το DA14585 μπορεί να φιλοξενήσει το επίπεδο σύνδεσης (link layer) καθώς και τα πρωτόκολλα του host, τα profiles και ο εξωτερικός επεξεργαστής τα οποία είναι υπεύθυνα για την λειτουργία της εφαρμογής. Τα δύο στοιχεία θα επικοινωνούν μέσω GTL με την βοήθεια σειριακής σύνδεσης που μπορεί να είναι είτε UART είτε SPI. Η εν λόγω οργάνωση φαίνεται στην Εικόνα 19.



Εικόνα 18: Διαμόρφωση (hardware) ενσωματωμένου επεξεργαστή DA14585



Εικόνα 19:Διαμόρφωση (hardware) εξωτερικού επεξεργαστή DA14585

6.3.6 DEVELOPMENT ENVIROMENT

Τα έργα λογισμικού βασίζονται και υποστηρίζονται από τα προγράμματα: ARM Keil μVision IDE / Debugger, ARM C / C ++ Compiler και βασικά συστατικά του middleware, Keil IDE και τα εργαλεία Keil[80]. Το DA14585 μπορεί να συνδέεται με καλώδια ART Segger JTAG, τα οποία υποστηρίζονται πλήρως από το περιβάλλον Keil. Προκειμένου να επιτευχθεί η βέλτιστη αξιολόγηση και προγραμματισμός του DA14585 στο μικρότερο δυνατό χρονικό διάστημα, η Dialog Semiconductor παρέχει το γραφικό εργαλείο SmartSnippets Toolbox. Το πρόγραμμα SmartSnippets Toolbox ενεργοποιεί την σύνδεση στις συσκευές DA14585 μέσω UART ή JTAG για να προγραμματίσει την μνήμη OTP που βρίσκεται πάνω στο chip ή σε εξωτερική Flash/EEPROM. Ένα πολύ ισχυρό εργαλείο που περιλαμβάνεται στην εργαλειοθήκη του SmartSnippets Toolbox με χρήση του προτύπου SDK της Dialog Semiconductor, είναι το Power profiler, με το οποίο ο χρήστης μπορεί να παρακολουθεί σε πραγματικό χρόνο την κατανάλωση ενέργειας και την συσχέτιση αυτής με τα διάφορα συμβάντα που συμβαίνουν κατά την εκτέλεση του λογισμικού.

7

QUALIFICATION ANALYSIS AUTOMATED SOFTWARE TESTING

7.1 GIT

Μέχρι στιγμής, το πιο ευρέως χρησιμοποιημένο σύγχρονο σύστημα ελέγχου εκδόσεων λογισμικού είναι το Git. Το Git είναι ένα πρόγραμμα ελέγχου και συντήρησης ανοικτού κώδικα, το οποίο αναπτύχθηκε αρχικά το 2005 από τον Linus Torvalds, το διάσημο δημιουργό του λειτουργικού συστήματος Linux[81]. Ένας τεράστιος αριθμός έργων λογισμικού βασίζονται στο Git για έλεγχο της έκδοσης, συμπεριλαμβανομένων των εμπορικών έργων και των έργων ανοιχτού κώδικα. Οι προγραμματιστές που χρησιμοποιούν το Git ξεχωρίζουν και έχουν τη δυνατότητα να προγραμματίσουν σε ένα ευρύ φάσμα λειτουργικών συστημάτων και IDEs (Integrated Development Enviroments). Έχοντας μια κατανεμημένη αρχιτεκτονική, το Git είναι ένα παράδειγμα ενός DVCS (Distributed Version Control System). Αντί να υπάρχει σε μία μόνο θέση το πλήρες ιστορικό έκδοσης του λογισμικού, όπως συμβαίνει στο CVS[82] ή στο Subversion[83](επίσης γνωστό ως SVN), στο Git, κάθε αντίγραφο κάθε κώδικα κάθε δημιουργού αποθηκεύεται ξεχωριστά και ονομάζεται repository. Το repository περιέχει το πλήρες ιστορικό όλων των αλλαγών και κάθε έκδοση. Το Git έχει σχεδιαστεί με γνώμονα την απόδοση, την ασφάλεια και την ευελιξία. Το Git είναι η καλύτερη επιλογή για τις περισσότερες ομάδες λογισμικού σήμερα. Τα χαρακτηριστικά του git είναι πολύ ισχυρά και το κάνουν να ξεχωρίζει σε σύγκριση με άλλες εναλλακτικές λύσεις. Το Git βλέπει τα δεδομένα ως μια σειρά στιγμιότυπων ενός μικροσκοπικού συστήματος αρχείων (file system). Κάθε φορά που υποβάλλεται ή αποθηκεύεται η κατάσταση του έργου, το Git παίρνει μια φωτογραφία της εικόνας των αρχείων τη δεδομένη στιγμή και αποθηκεύει μια αναφορά σε αυτό το στιγμιότυπο. Για να είναι πιο αποδοτικό, το Git δεν ξανά αποθηκεύει ένα αρχείο, εφόσον αυτό δεν έχει τροποποιηθεί. Αντί αυτού, αποθηκεύει έναν σύνδεσμο για το προηγούμενο όμοιο αρχείο το οποίο έχει ήδη αποθηκεύσει. Το Git λοιπόν αντιλαμβάνεται τα δεδομένα περισσότερο σαν μία ροή από στιγμιότυπα. Οι περισσότερες λειτουργίες στο Git χρειάζονται μόνο τοπικά αρχεία και πόρους για να εκτελεστούν. Γενικά, δεν χρειάζεται να μεταφερθεί πληροφορία από κάποιον άλλο υπολογιστή του ίδιου δικτύου. Οι περισσότερες λειτουργίες στο Git φαίνονται σχεδόν στιγμιαίες, κυρίως λόγω του γεγονότος ότι το πλήρες ιστορικό του έργου βρίσκεται στον τοπικό δίσκο. Για παράδειγμα, για την περιήγηση στο ιστορικό του έργου, το Git δεν χρειάζεται να λάβει το ιστορικό από το διακομιστή και να το εμφανίσει στον χρήστη — χρειάζεται απλά να το διαβάσει από την τοπική βάση δεδομένων. Αυτό σημαίνει ότι υπάρχει η δυνατότητα προβολής του ιστορικού του έργου άμεσα. Για την προβολή των αλλαγών που εισήχθησαν ανάμεσα στην τρέχουσα έκδοση ενός αρχείου και στην έκδοσή τους πριν ένα μήνα, ένα σύστημα ελέγχου εκδόσεων μπορεί να ρωτήσει το διακομιστή για τις διαφορές των δύο αρχείων ή να έλξει την παλιά έκδοση του αρχείου και να υπολογίσει τις διαφορές τοπικά. Αντί αυτού, το Git μπορεί επί τόπου τοπικά να αναζητήσει την έκδοση του αρχείου που δημιουργήθηκε ένα μήνα πριν και να κάνει τον υπολογισμό των διαφορών τους. Οτιδήποτε υπάρχει στο Git περνά από έλεγχο αθροίσματος (check-summed) πριν αποθηκευτεί. Αυτό σημαίνει ότι είναι αδύνατο κάποιος να αλλάξει το περιεχόμενο ενός αρχείου ή ενός καταλόγου χωρίς το Git να γνωρίζει για αυτό. Η λειτουργικότητα αυτή είναι ενσωματωμένη στο Git στα χαμηλότερα επίπεδά και είναι αναπόσπαστη της φιλοσοφίας του. Είναι αδύνατο να χαθεί πληροφορία κατά τη μεταφορά της ή να φθαρεί κάποιο αρχείο χωρίς το Git να το ανιχνεύσει. Οι βασικές λειτουργίες του Git είναι: commit, add, branch, merge, pull, push, diff και αναλύονται παρακάτω. Τα αρχεία στο Git μπορούν να βρίσκονται σε τρεις κύριες καταστάσεις: υποβεβλημένο, τροποποιημένο και καταχωρημένο.

- Υποβεβλημένο είναι ένα αρχείο όταν τα δεδομένα του είναι αποθηκευμένα με ασφάλεια στην τοπική βάση δεδομένων. Το αρχείο τότε βρίσκεται στον κατάλογο εργασίας (working directory).
- Τροποποιημένο είναι ένα αρχείο όταν έχει αλλάξει, αλλά δεν έχει υποβληθεί στη βάση δεδομένων ακόμα. Το αρχείο βρίσκεται στο ενδιάμεσο στάδιο (staging area).

- Καταχωρημένο είναι ένα τροποποιημένο αρχείο της τρέχουσας έκδοσης όταν έχει επισημανθεί για βρίσκεται στο επόμενο υποβεβλημένο στιγμιότυπο. Το αρχείο βρίσκεται στον κατάλογο .git. (.git directory)

Ο κατάλογος του Git είναι το μέρος όπου το Git αποθηκεύει τα μεταδεδομένα (metadata) και τη βάση δεδομένων του έργου. Αυτό είναι το πιο σημαντικό μέρος του Git και είναι αυτό που αντιγράφεται όταν ο χρήστης επιθυμεί να αντιγράψει ένα repository από ένα άλλο υπολογιστή. Ο κατάλογος εργασίας είναι απλά ένα στιγμιότυπο μίας έκδοσης του έργου. Τα αρχεία αυτά ανασύρονται από τη συμπιεσμένη βάση δεδομένων του καταλόγου του Git και τοποθετούνται στον τοπικό δίσκο ώστε να μπορεί ο χρήστης να τα χρησιμοποιήσει ή να τα τροποποιήσει. Το ενδιάμεσο στάδιο είναι ένα αρχείο το οποίο γενικά περιλαμβάνεται στον κατάλογο του Git, στο οποίο είναι αποθηκευμένες πληροφορίες σχετικά με το τι θα περάσει στην επόμενη υποβολή (commit). Το τεχνικό όνομα του ενδιάμεσου σταδίου στην ορολογία του Git είναι "ευρετήριο"(index), αλλά το όνομα "ενδιάμεσο στάδιο" είναι επίσης σύνηθες. Η βασική ροή εργασίας του Git είναι:

1. Τροποποίηση κάποιων αρχείων στον κατάλογο εργασίας.
2. Καταχώρηση των αρχείων, προσθέτοντας στιγμιότυπά τους στο ενδιάμεσο στάδιο.
3. Υποβολή, η οποία θα πάρει τα αρχεία όπως είναι στο ενδιάμεσο στάδιο και αποθηκεύει αυτό το στιγμιότυπο μόνιμα στον κατάλογο του Git.

Αν μια συγκεκριμένη έκδοση ενός αρχείου βρίσκεται στον κατάλογο του Git, ονομάζεται υποβεβλημένη. Αν έχει τροποποιηθεί και έχει προστεθεί στο ενδιάμεσο στάδιο, ονομάζεται καταχωρημένη έκδοση. Επίσης αν έχει τροποποιηθεί από τότε που ελέγχθηκε τελευταία φορά αλλά δεν έχει καταχωρηθεί τότε λέμε ότι πρόκειται για τροποποιημένη έκδοση. Σε αντίθεση με άλλα λογισμικά ελέγχου έκδοσης, το git δεν μπερδεύει τα ονόματα αρχείων κατά την αποθήκευση όλων των εκδόσεων και δημιουργεί δέντρου που παρουσιάζει τις διάφορες εκδόσεις από τους διαφορετικούς δημιουργούς στα κλαδιά του. Το Git επικεντρώνεται στο περιεχόμενο κάθε αρχείου, ενώ τα αντικείμενα κάθε ξεχωριστού repository αποθηκεύονται χρησιμοποιώντας την κωδικοποίηση δέλτα, συμπιέζονται και αποθηκεύονται. Με αυτόν τον τρόπο εξασφαλίζεται η ασφάλεια των δεδομένων ενώ ταυτόχρονα γίνεται εύκολος ο εντοπισμός των διαφορών μεταξύ δύο εκδόσεων του ίδιου κώδικα.

7.1.1 ΕΝΣΩΜΑΤΩΣΗ ΕΝΟΣ ΝΕΟΥ REPOSITORY

Κάθε εντολή που αφορά το Git, γράφεται στο ειδικό bash του git (Εικόνα 20) και ανοίγει με την εξής διαδικασία:1) Εύρεση φακέλου όπου θα αποθηκευτεί το repository, 2) δεξί κλικ και επιλογη «Git bash here». Προκειμένου να ξεκινήσει και να δημιουργηθεί ένα καινούριο έργο στο Git υπάρχει η επόμενη εντολή:

```
$ git init
```

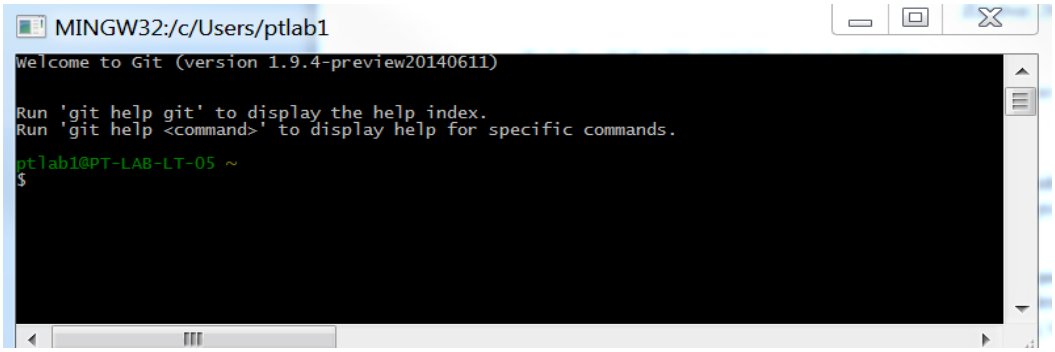
Η εντολή αυτή δημιουργεί έναν νέο υπο-κατάλογο με το όνομα `.git` ο οποίος περιέχει όλα τα απαραίτητα αρχεία για το repository. Ουσιαστικά ο υπο-κατάλογος αυτός αποτελεί ένα σκελετό για το repository. Για οτιδήποτε δημιουργείται από εδώ και πέρα στο συγκεκριμένο directory, το Git θα κρατά πλήρες ιστορικό. Στην συνέχεια, για να επιτραπεί στο git να αποθηκεύσει όλα τα δεδομένα, όλων των αρχείων κάτω από το ίδιο directory υπάρχει η επόμενη εντολή:

```
$ git add .  ή  $ git add test.txt
```

Η παραπάνω εντολή προσθέτει όλες τις αλλαγές στο repository, με την προσωρινή αποθήκευση αυτών στη staging area. Πλέον, τα δεδομένα έχουν αποθηκευτεί σε μια προσωρινή περιοχή που το Git ονομάζει ενδιάμεσο στάδιο ή index. Για να αποθηκευτούν τα περιεχόμενα του index μόνιμα σε ένα repository υπάρχει η επόμενη εντολή:

```
$ git commit
```

Η εντολή commit ισοδυναμεί με την λήψη στιγμιότυπου από το τρέχον repository, τη στιγμή που εκτελείται η εντολή, με σκοπό την επιστροφή στον εν λόγω στιγμιότυπο όταν υπάρχει ανάγκη. Στην συνέχεια θα εμφανιστεί ένα μήνυμα επιβεβαίωσης του commit. Με τον παραπάνω τρόπο αποθηκεύεται η πρώτη έκδοση ενός project στο Git.



Εικόνα 20: Git bash

7.1.2 ΕΦΑΡΜΟΓΗ ΑΛΛΑΓΩΝ

Σε περίπτωση που αλλάξει το περιεχόμενο σε κάποιο ή κάποια από τα αρχεία, η εντολή add θα εντάξει τις αλλαγές στην προσωρινή περιοχή index:

```
$ git add file1 file2 file3
```

Στην συνέχεια, κατά τα γνωστά, πρέπει να γίνει commit. Για να εμφανιστούν τα αρχεία που βρίσκονται στην προσωρινή περιοχή και πρέπει να γίνουν commit υπάρχει η εξής εντολή:

```
$ git diff --cached
```

Χωρίς την επέκταση --cached, η εντολή $ git diff, εμφανίζει ακριβώς όλες τις αλλαγές που έχουν γίνει, γραμμή προς γραμμή, αλλά δεν έχουν γίνει ακόμη add στο index. Στην συνέχεια εφόσον εκτελεστούν οι εντολές $git add και $ git commit, υπάρχει η δυνατότητα επιβεβαίωσης του commit με τον εξής τρόπο: ξανά εκτελείται η εντολή $git diff και αυτή την φορά δεν θα πρέπει να επιστρέφει τίποτα. Με την παρακάτω εντολή εμφανίζονται οι διαφορές των αρχείων που υπάρχουν στο staging area και των αρχείων που υπάρχουν στο repository:

```
$git diff --staged
```

Με την παρακάτω εντολή διαγράφεται το επιθυμητό αρχείο:

```
$ git rm «όνομα αρχείου»
```

Επιπλέον, υπάρχει η δυνατότητα εμφάνισης της κατάστασης των αρχείων εκτελώντας την επόμενη εντολή:

```
$ git status
```

Με την εντολή $git status, εμφανίζονται στο bash οι εξής πληροφορίες, δηλαδή τα αρχεία για τα οποία δεν έχει γίνει commit. Μετά τις επιθυμητές αλλαγές και προσθήκες, πρέπει να εκτελεστεί η ακολουθία εντολών:

```
$ git add «όνομα αρχείου»
$ git commit –m «όνομα αλλαγής»
```

Προτείνεται, η προσθήκη τίτλου σε κάθε commit μετά από κάθε αλλαγή, γιατί μετά από πλήθος αλλαγών, είναι δύσκολο για τον χρήστη να τις ξεχωρίσει ή ακόμη και να τις θυμηθεί. Η εντολή commit αποθηκεύει όλες τις αλλαγές του reporitory σε μορφή δέντρου. Με αυτόν τον τρόπο δεν χάνεται καμία αλλαγή και ο χρήστης μπορεί να ανακτήσει οποιαδήποτε παλιά ή καινούρια έκδοση των αρχείων επιθυμεί. Εναλλακτικά, η παρακάτω εντολή θα αναγνωρίσει αυτόματα κάθε αλλαγή σε παλιά αρχεία και θα κάνει σε ένα βήμα add και commit:

```
$ git commit –a
```

Η εντολή $ git add, για νέα αρχεία, αλλά και για ήδη υπάρχοντα στα οποία γίνεται κάποια αλλαγή. Και στις δύο περιπτώσεις, τα αρχεία μεταβαίνουν στην ενδιάμεση stage area και είναι έτοιμα για το επόμενο commit. Με την παρακάτω εντολή, μεταφέρεται το αρχείο από την staging area πίσω στο working directory και χρησιμοποιείται όταν το αρχείο δεν είναι έτοιμο ακόμη για commit αλλά έχει εκτελεστεί η εντολή $git add:

```
$ git reset HEAD «όνομα αρχείου»
```

7.1.3 ΠΡΟΒΟΛΗ ΙΣΤΟΡΙΚΟΥ ΑΛΛΑΓΩΝ

Οποιαδήποτε στιγμή μπορεί να εμφανιστεί το ιστορικό των commit με την εντολή:

```
$ git log
```

Επιπλέον, η εμφάνιση όλων των αλλαγών σε κάθε βήμα γίνεται με την εντολή:

```
$ git log -p
```

7.1.4 ΔΙΑΚΛΑΔΩΣΕΙΣ

Ένα απλό repository του Git μπορεί να περιέχει πολλές διαφορετικές διακλαδώσεις(branches). Κάθε διακλάδωση είναι και μια διαφορετική έκδοση του κεντρικού repository. Για να δημιουργηθεί μια καινούρια διακλάδωση χρησιμοποιείται η εντολή:

```
$ git branch «όνομα διακλάδωσης»
```

Με την ακόλουθη εντολή εμφανίζεται η λίστα που περιέχει όλες τις διακλαδώσεις. Η διακλάδωση που έχει * πριν το όνομά της, είναι και η τρέχουσα διακλάδωση εργασίας.

```
$ git branch
```

Η διακλάδωση master είναι η διακλάδωση που δημιουργείται αυτόματα. Προκειμένου να αλλάξει η τρέχουσα διακλάδωση εργασίας υπάρχει η εντολή:

```
$ git checkout «όνομα διακλάδωσης που επιθυμεί ο χρήστης να πάει»
```

Οι αλλαγές που γίνονται σε κάποιο παρακλάδι του repository, δεν φαίνονται στα υπόλοιπα. Προκειμένου συγχωνευθούν οι αλλαγές που κάποια διακλάδωση, στη διακλάδωση master, χρησιμοποιείται η εντολή:

```
$ git merge
```

Η διαγραφή κάποιας διακλάδωσης του έργου γίνεται με την εντολή:

```
$ git branch -d «όνομα διακλάδωσης»
```

7.1.5 ΚΛΩΝΟΠΟΙΩΝΤΑΣ ΕΝΑ ΥΠΑΡΧΩΝ REPOSITORY

Σε περίπτωση που ο χρήστης επιθυμεί να αντιγράψει τοπικά κάποιο υπάρχων repository στο Git, πρέπει να χρησιμοποιήσει την εντολή:

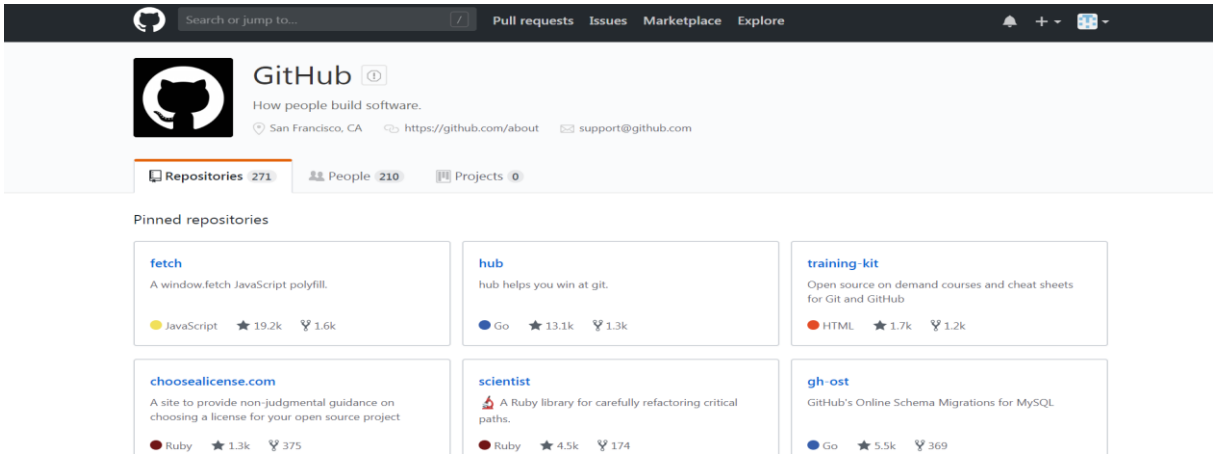
```
$ git clone «url του repository»
```

Με την εντολή git clone όλες οι εκδόσεις κάθε αρχείου του έργου αποθηκεύονται τοπικά. Στην πραγματικότητα, αν ο δίσκος του διακομιστή (server disk) αλλοιωθεί, με την εντολή αυτή δίνεται η δυνατότητα σε οποιονδήποτε από τους κλώνους του να θέσει το διακομιστή στην κατάσταση που ήταν όταν κλωνοποιήθηκε.

7.2 GITHUB

Το Github είναι ένα online εργαλείο που περιέχει δημόσια repositories και δίνει τη δυνατότητα στους χρήστες να εργαστούν μαζί σε κοινά repositories, από οπουδήποτε[81]. Η ιστοσελίδα του Github φαίνεται στην Εικόνα 21. Κάθε χρήστης του Github εκτελεί τα παρακάτω βήματα με την σειρά που καταγράφονται:

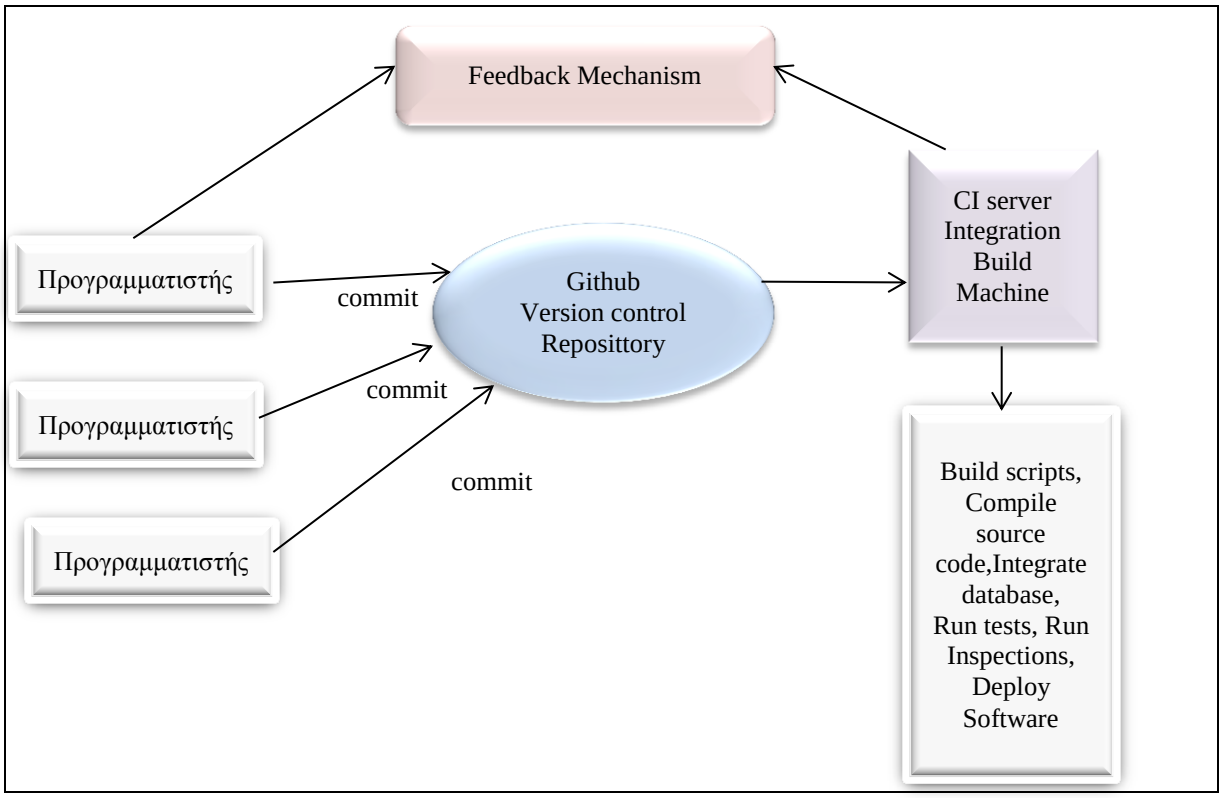
1. Δημιουργία repository
2. Δημιουργία διακλάδωσης (branch)
   Με την τεχνική αυτή οι χρήστες δουλεύουν σε διαφορετικές εκδόσεις του ίδιου repository, την ίδια στιγμή. Εξαρχής, το repository έχει μια κύρια διακλάδωση με την ονομασία master. Οι διακλαδώσεις δίνουν τη δυνατότητα στους χρήστες να πειραματίζονται και να κάνουν αλλαγές, χωρίς να επηρεάζουν τη διακλάδωση master, δηλαδή το αρχικό repository.
3. Στάδιο commit
   Στο Github οι αποθηκευμένες αλλαγές λέγονται commits. Κάθε commit συνοδεύεται και από ένα μήνυμα που υποδηλώνει συνήθως το είδος της εκάστοτε αλλαγής.
4. Αίτημα Pull
   Όταν υπάρχει ένα αίτημα pull από την πλευρά του χρήστη, σημαίνει πως ο ίδιος έχει επέμβει στο repository, έχει κάνει κάποιες αλλαγές και προτείνει αυτές να ενταχθούν στη διακλάδωση master. Η ενσωμάτωση των αλλαγών στη διακλάδωση master ονομάζεται merge.
5. Αίτημα Push
   Εφόσον εγκριθεί το αίτημα pull, οι αλλαγές ενσωματώνονται στον master μέσω την εντολής push.



Εικόνα 21:Github site

7.3 ΜΕΘΟΔΟΙ ΣΥΝΕΧΟΥΣ ΟΛΟΚΛΗΡΩΣΗΣ

Η συνεχής ολοκλήρωση (Continuous Integration ή CI) αποτελεί τον ακρογωνιαίο λίθο των σύγχρονων διαδικασιών ανάπτυξης λογισμικού. Στην πραγματικότητα, όταν εισάγεται η λειτουργία της συνεχής ολοκλήρωσης σε έναν οργανισμό, αλλάζει ριζικά τον τρόπο σκέψης των ομάδων σχετικά με την αναπτυξιακή διαδικασίας[84]. Έχει τη δυνατότητα να ενεργοποιεί και να ωθεί (trigger) μια σειρά βαθμιαίων αυξανόμενων μοντέλων βελτίωσης, ξεκινώντας κάνοντας build σε μια απλή αυτοματοποιημένη διαδικασία ελέγχου έως την τελική παράδοση στην ομάδα ανάπτυξης. Μια καλή υποδομή συνεχούς ολοκλήρωσης διευκολύνει την αναπτυξιακή διαδικασία με τον γρήγορο εντοπισμό και την άμεση διόρθωση σφαλμάτων, παρέχοντας ταυτόχρονα έναν χρήσιμο πίνακα εργαλείων προς τους προγραμματιστές και μη. Ο κύριος στόχος των μεθόδων συνεχούς ολοκλήρωσης είναι να βοηθούν τις ομάδες ανάπτυξης λογισμικού να παρέχουν πιο αξιόπιστα και επαγγελματικά προϊόντα στον τελικό χρήστη. Κάθε σύγχρονη εταιρεία, εάν θέλει να εξελίσσεται και να είναι ανταγωνιστική, θα πρέπει να εντάσσει την συνεχή ολοκλήρωση στη διαδικασία ανάπτυξης και ελέγχου[85]. Ο Martin Fowler έχει ορίσει την συνεχή ολοκλήρωση ως[86]: μια πρακτική ανάπτυξης λογισμικού όπου τα μέλη μιας ομάδας ενσωματώνουν το έργο τους συχνά (κάθε άτομο συνήθως ενσωματώνει το έργο του μια φορά την ημέρα τουλάχιστον), οδηγώντας σε πολλαπλές ενσωματώσεις την ημέρα. Κάθε ενσωμάτωση επαληθεύεται από μια διαδικασία αυτοματοποιημένου build, συμπεριλαμβανομένου του αυτοματοποιημένου ελέγχου, με σκοπό την γρηγορότερη ανίχνευση σφαλμάτων. Πριν την ανάπτυξη των τεχνικών συνεχούς ολοκλήρωσης, ο χρόνος και η ενέργεια των ομάδων ανάπτυξης λογισμικού αφιερωνόταν, κατά μεγάλο ποσοστό, στην φάση της ενσωμάτωσης. Κατά τη διάρκεια αυτής της φάσης, κάθε αλλαγή που έκανε κάθε προγραμματιστής στο κώδικα του λογισμικού, έπρεπε να ενσωματωθεί στο κυρίως project και ταυτόχρονα το σύνολο των αλλαγών που προέρχονταν από ξεχωριστά άτομα να λειτουργούν αρμονικά και αποδοτικά. Ένα μεγάλο μειονέκτημα της παραπάνω τακτικής εντοπίζεται στο γεγονός πως είναι αρκετά δύσκολο να προβλεφθούν τα ζητήματα που θα προκύψουν και ακόμη πιο δύσκολο να διορθωθούν, καθώς κάτι τέτοιο συνεπάγεται την χρονοβόρα ανασκόπηση σε κώδικα που έχει αναπτυχθεί, ίσως και μήνες πριν[87]. Αυτή η επώδυνη διαδικασία, συχνά οδηγούσε σε σημαντικές καθυστερήσεις στην παράδοση και μη αναμενόμενα κόστη. Η συνεχής ολοκλήρωση, δημιουργήθηκε προκειμένου να αντιμετωπίσει τα παραπάνω ζητήματα. Τα στοιχεία ενός συστήματος συνεχούς ολοκλήρωσης φαίνονται στην Εικόνα 22.



Εικόνα 22:Τα στοιχεία ενός συστήματος συνεχούς ολοκλήρωσης

Η συνεχής ολοκλήρωση, περιλαμβάνει ένα εργαλείο εμφάνισης αλλαγών μέσω του συστήματος ελέγχου. Κάθε φορά που εντοπίζεται μια αλλαγή, το εργαλείο κάνει αυτόματα compile και ελέγχει το σύστημα λογισμικού. Εάν κάτι πάει στραβά, ειδοποιεί τους προγραμματιστές ώστε να διορθώσουν αμέσως το ζήτημα. Ωστόσο, η συνεχής ολοκλήρωση μπορεί να κάνει πολλά περισσότερα. Βοηθά τον χρήστη να κρατά καρτέλες της κατάστασης του κώδικα, παρακολουθεί αυτόματα την ποιότητα του κώδικα υπολογίζοντας ειδικές μετρικές και συμβάλλει στη διατήρηση του τεχνικού κόστους αλλά και του κόστους συντήρησης, σε χαμηλά επίπεδα. Οι δημοφιλείς μετρικές ποιότητας κώδικα μπορούν επίσης να ενθαρρύνουν τους προγραμματιστές να υπερηφανεύονται για την ποιότητα του κώδικα τους και να προσπαθούν να το βελτιώσουν. Συνδυασμένο με αυτοματοποιημένες δοκιμές αποδοχής, το CI μπορεί επίσης να λειτουργήσει ως εργαλείο επικοινωνίας, καθώς έχει την ικανότητα να δημοσιεύει μια σαφή εικόνα της τρέχουσας κατάστασης των προσπαθειών ανάπτυξης. Απλοποιεί και επιταχύνει την διαδικασία παράδοσης των εκδόσεων λογισμικού, αφού αυτοματοποιεί τη διαδικασία ανάπτυξης της τελευταίας έκδοσης μιας εφαρμογής. Στην ουσία, η συνεχής ενσωμάτωση αφορά στη μείωση του κινδύνου με την παροχή ταχύτερης ανάδρασης. Πρώτα απ' όλα, έχει σχεδιαστεί για να βοηθήσει στον εντοπισμό και την επίλυση των προβλημάτων ολοκλήρωσης και παλινδρόμησης, με αποτέλεσμα την ομαλότερη, ταχύτερη παράδοση λογισμικού με λιγότερα σφάλματα. Παρέχοντας καλύτερη προβολή τόσο για τα τεχνικά όσο και για τα μη τεχνικά μέλη της ομάδας σχετικά με την κατάσταση του project, η συνεχής ολοκλήρωση μπορεί να ανοίξει και να διευκολύνει τους δίαυλους επικοινωνίας μεταξύ των μελών της ομάδας και να ενθαρρύνει τη συνεργατική επίλυση προβλημάτων και τη βελτίωση της διαδικασίας. Προκειμένου να αξιοποιηθεί στο έπακρο η συνεχής ολοκλήρωση, η ομάδα ανάπτυξης πρέπει να αποκτήσει μια νοοτροπία συνεχούς ολοκλήρωσης. Αυτό σημαίνει πως θα πρέπει τα έργα να ελέγχονται από μια αξιόπιστη, επαναληπτική και αυτοματοποιημένη διαδικασία ελέγχου στην οποία δεν θα επεμβαίνει ο άνθρωπος. Η διόρθωση των μη επιτυχημένων builds θα πρέπει να είναι προτεραιότητα και η διαδικασία ανάπτυξης θα πρέπει να αυτοματοποιείται στο μεγαλύτερο ποσοστό της. Η επιτυχία των μεθόδων συνεχούς ολοκλήρωσης εξαρτάται σε μεγάλο βαθμό από την ποιότητα των ελέγχων. Η ομάδα θα πρέπει να δώσει μεγάλη έμφαση στην σχεδίαση και στην εφαρμογή τεχνικών αυτοματοποιημένων ελέγχων υψηλής ποιότητας, έτσι ώστε σε συνδυασμό με το κατάλληλο εργαλείο συνεχούς ολοκλήρωσης να πετυχαίνει τα βέλτιστα αποτελέσματα.

7.3.1 ΣΥΝΕΧΗΣ ΟΛΟΚΛΗΡΩΣΗ ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ JENKINS

Το Jenkins είναι ένα εργαλείο συνεχούς ολοκλήρωσης, το οποίο έχει αναπτυχθεί με την γλώσσα Java και χρησιμοποιείται από ομάδες όλων των μεγεθών, για έργα που έχουν αναπτυχθεί από τις μεγαλύτερες γλώσσες

προγραμματισμού. Είναι εύκολο στην χρήση, το περιβάλλον εργασίας χρήστη είναι απλό και διαισθητικά ελκυστικό, ενώ ταυτόχρονα δεν είναι δύσκολο για κάποιον που δεν το έχει ξανά χρησιμοποιήσει να το μάθει. Είναι εξαιρετικά ευέλικτο και προσαρμόζεται εύκολα στις προτιμήσεις κάθε διαφορετικού χρήστη. Επιπλέον, εκατοντάδες επεκτάσεις ανοικτού κώδικα, που συνεχώς ανανεώνονται, καλύπτουν σχεδόν κάθε ανάγκη οποιουδήποτε προγραμματιστή[88]. Οι επεκτάσεις εγκαθίστανται πολύ εύκολα και βοηθούν τον χρήστη σε πληθώρα θεμάτων όπως τις εκδόσεις ελέγχου συστήματος, τα εργαλεία build, τις μετρικές ποιότητας κώδικα, τις ειδοποιήσεις για κάθε build κλπ. Το Jenkins είναι το αποτέλεσμα ενός οραματιστή προγραμματιστή, του Kohsuke Kawaguchi, ο οποίος ξεκίνησε το έργο ως χόμπι υπό την επωνυμία Hudson στα τέλη του 2004, ενώ εργαζόταν στην εταιρεία Sun. Καθώς το Hudson εξελίχθηκε με τα χρόνια, υιοθετήθηκε από όλο και περισσότερες ομάδες μέσα στην Sun για τα δικά τους έργα. Στις αρχές του 2008, η Sun αναγνώρισε την ποιότητα και την αξία του εργαλείου και ζήτησε από τον Kohsuke να δουλεύει πάνω στο Hudson με πλήρες ωράριο, αρχίζοντας να παρέχει επαγγελματικές υπηρεσίες και υποστήριξη γύρω από το εργαλείο Hudson. Μέχρι το 2010, το Hudson είχε γίνει το κορυφαίο εργαλείο συνεχής ολοκλήρωσης με μερίδιο αγοράς άνω του 70%. Το 2009, η Oracle αγόρασε την Sun. Προς το τέλος του 2010, προέκυψαν εντάσεις μεταξύ τους, ώσπου τον Ιανουάριο του 2011, η κοινότητα προγραμματιστών Hudson ψήφισε και μετονόμασε το εργαλείο σε Jenkins[88].

7.3.1.1 USING THE JENKINS BUILD SERVER

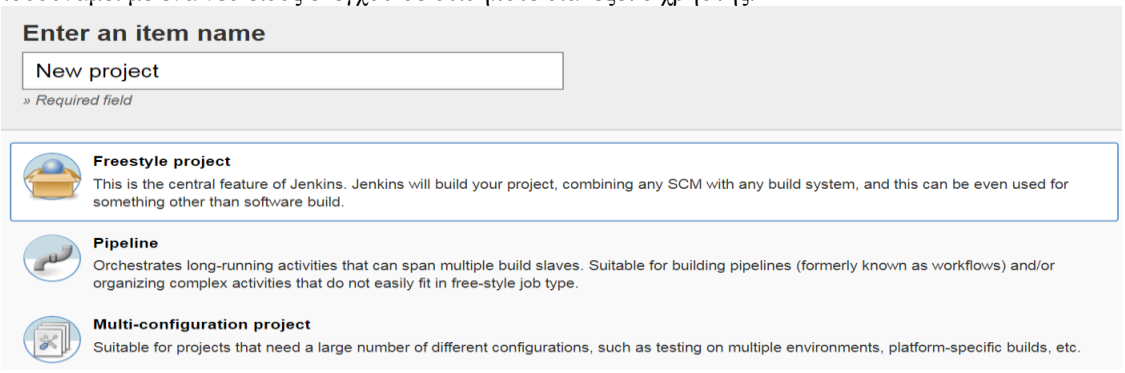
Η συνεχής ολοκλήρωση είναι μια διαδικασία κατά την οποία όλα τα παράγωγα των αναπτυξιακών διαδικασιών ενσωματώνονται όσο το δυνατόν νωρίτερα. Οι μονάδες που προκύπτουν ελέγχονται αυτόματα, επιτρέποντας έτσι τον γρήγορο εντοπισμό σφαλμάτων, συνεπώς την άμεση διόρθωσή τους. Το Jenkins είναι ένα δημοφιλές εργαλείο ανοιχτού κώδικα που υλοποιεί την μέθοδο της συνεχούς ολοκλήρωσης μέσω των αυτοματοποιημένων builds. Πιθανές λειτουργίες του Jenkins:

- Εκτέλεση και build κώδικα χρησιμοποιώντας συστήματα όπως Apache Maven ή Gradle.
- Εκτέλεση ενός shell script.
- Εμφάνιση λεπτομερών αποτελεσμάτων για κάθε build.
- Εκτέλεση αυτοματοποιημένων ελέγχων λογισμικού

Το εργαλείο Jenkins δίνει τη δυνατότητα στον χρήστη να ορίσει την στιγμή εκτέλεσης ενός κώδικα ή ακόμη και τη διάρκεια εκτέλεσης. Επιπλέον, παρακολουθεί την πορεία κάθε βήματος εκτέλεσης εμφανίζοντας τα αποτελέσματα σε ένα ειδικό log σε πραγματικό χρόνο (console output) και επιτρέπει στον χρήστη τον τερματισμό της διαδικασίας build όποια στιγμή αυτός το επιθυμεί. Επίσης μπορεί να στέλνει ειδοποίηση σε περίπτωση επιτυχίας ή αποτυχίας, αλλά και να εμφανίζει στατιστική ανάλυση των αποτελεσμάτων, αν ο χρήστης το επιλέξει. Η αρχική σελίδα του Jenkins φαίνεται παρακάτω στην Εικόνα 23.

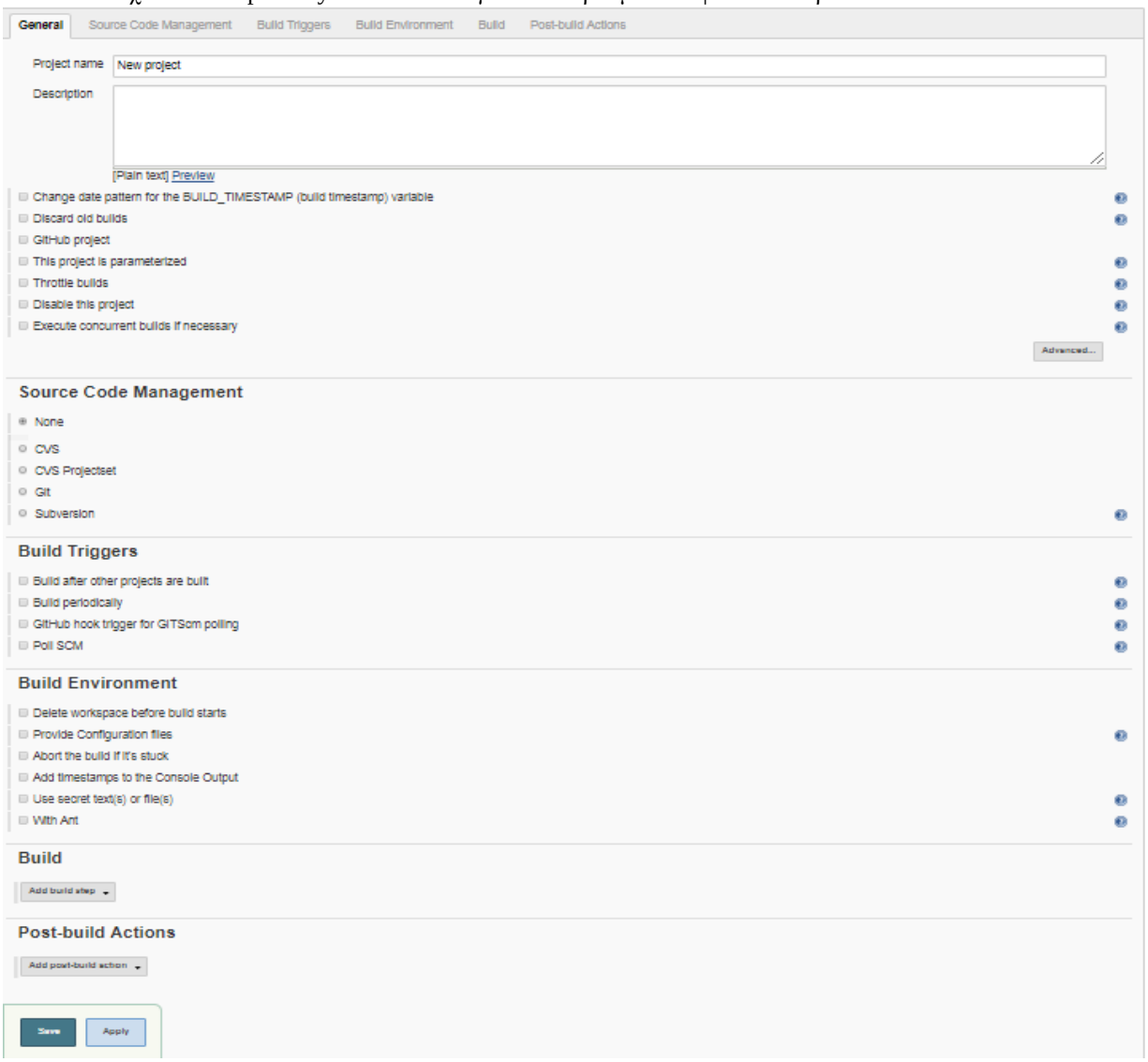
7.3.1.2 SETTING UP A JENKINS JOB

Επιλέγοντας New item (πάνω αριστερά), ο χρήστης δημιουργεί ένα καινούριο job στον Jenkins, το οποίο μπορεί να κάνει build επιλέγοντας τις επιθυμητές ρυθμίσεις όπως φαίνεται στην παρακάτω εικόνα. Κάθε νέο job ισοδυναμεί με ένα νέο είδος ελέγχου σε οτιδήποτε διαλέξει ο χρήστης.

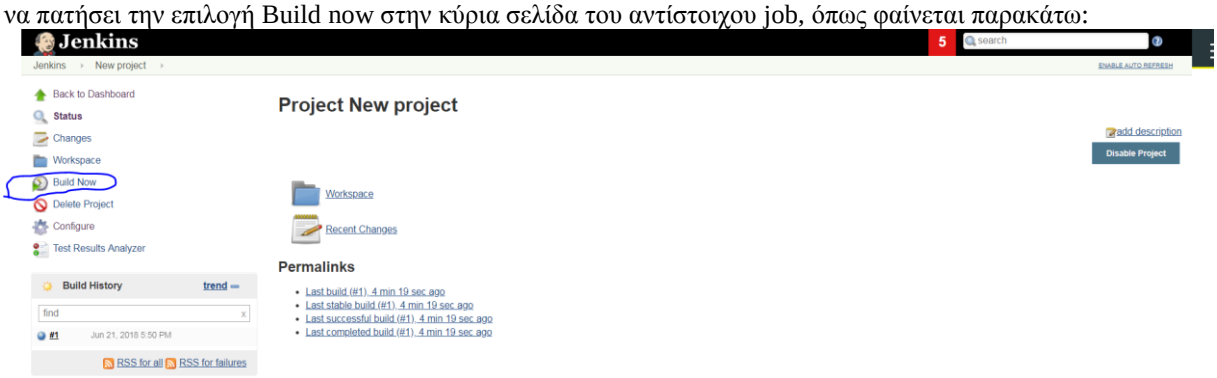


Ο χρήστης θα πρέπει να συμπληρώσει ένα όνομα για το Job, να επιλέξει το είδος του project που επιθυμεί και στην συνέχεια να πατήσει OK. Στην συνέχεια θα εμφανιστεί η σελίδα επιλογής παραμέτρων για το νέο job. Υπάρχει και η δυνατότητα αντιγραφής ρυθμίσεων από παλιότερο job. Στην σελίδα επιλογής παραμέτρων ο χρήστης έχει τη δυνατότητα να ορίσει γενικές πληροφορίες για τον κώδικα, τον τρόπο εισαγωγής του κώδικα για τον οποίο μετέπειτα ο Jenkins θα κάνει build, τις ιδιότητες κάθε build όπως την συχνότητά του, τον τρόπο εμφάνισης των αποτελεσμάτων κάθε build αλλά και τις ενέργειες που θα πρέπει να κάνει ο Jenkins μετά από κάθε build. Εάν επιλεγεί ο κώδικας να εισάγεται από κάποιο repository του Github, θα πρέπει ο χρήστης να

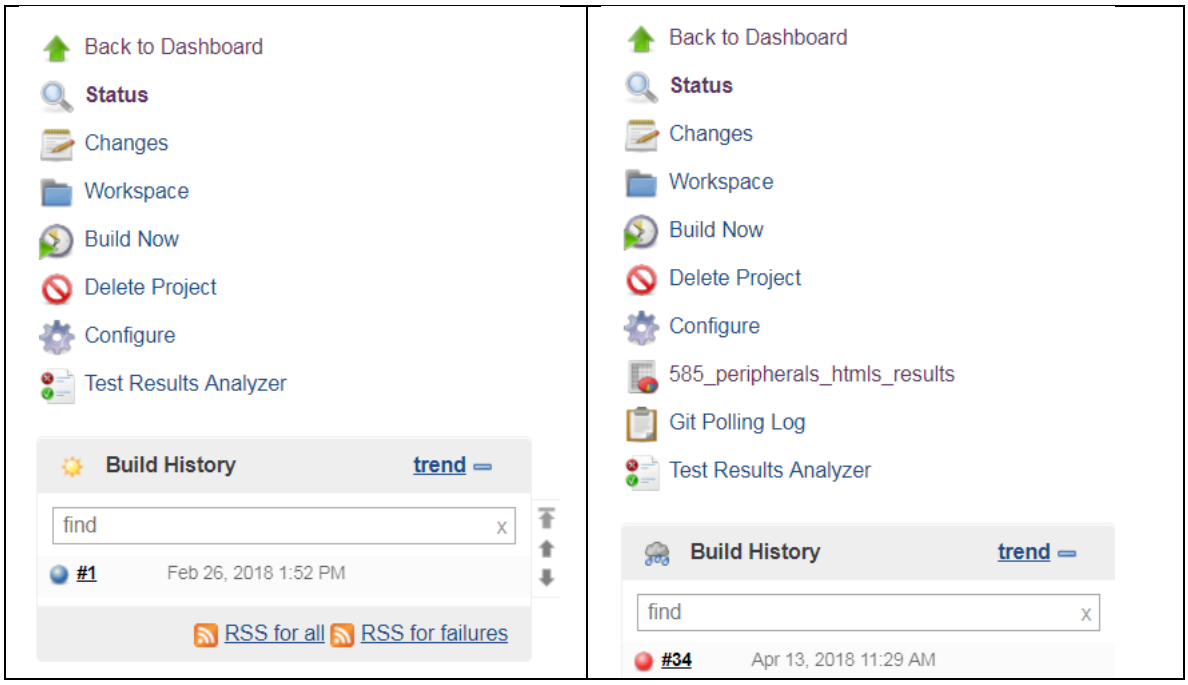
εισάγει το URL του επιθυμητού Git repository, στο πεδίο Source Code Management, κάτω από την επιλογή Git, στο αντίστοιχο πεδίο Repository URL. Η εν λόγω σελίδα ρυθμίσεων φαίνεται παρακάτω:

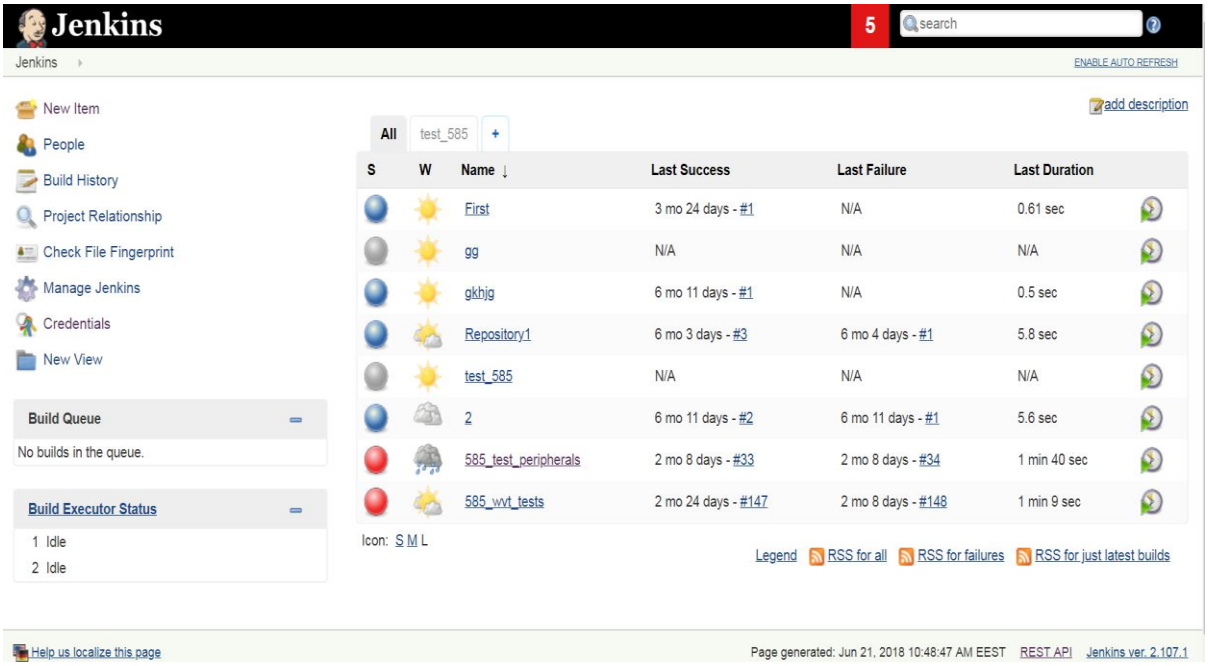


Εφόσον ο χρήστης επιλέξει τις κατάλληλες ρυθμίσεις και συμπληρώσει τα αντίστοιχα πεδία, στην συνέχεια, θα πρέπει να κάνει save (κάτω αριστερά). Στην συνέχεια, για να κάνει build στον κώδικα που εισήγαγε θα πρέπει να πατήσει την επιλογή Build now στην κύρια σελίδα του αντίστοιχου job, όπως φαίνεται παρακάτω:



Μετά από λίγο, εάν τα αποτελέσματα του job είναι θετικά, το αντίστοιχο build θα γίνει μπλε. Αλλιώς, θα γίνει κόκκινο. Και οι δύο περιπτώσεις φαίνονται παρακάτω:


Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
Test Results Analyzer
Build History
trend
find
#1
Feb 26, 2018 1:52 PM
RSS for all
RSS for failures
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
585_peripherals_htmls_results
Git Polling Log
Test Results Analyzer
Build History
trend
find
#34
Apr 13, 2018 11:29 AM



Εικόνα 23: Αρχική σελίδα Jenkins

7.3.2 TESTING SUITE DEVELOPMENT

Στις επόμενες παραγράφους παρουσιάζεται μια πραγματική σύνδεση όλων των εργαλείων που προαναφέρθηκαν, σε ρεαλιστικές συνθήκες για λογαριασμό της εταιρείας Dialog Semiconductor. Παρακάτω παρουσιάζεται ο αυτοματοποιημένος έλεγχος συγκεκριμένων χαρακτηριστικών του DA14585 σε ρόλο advertiser και με την βοήθεια του DA14683 σε ρόλο scanner. Χρησιμοποιείται η έκδοση Git 1.9.4 και η έκδοση Jenkins ver 2.107.1. Η σύνδεση στον Jenkins γίνεται από την θύρα 9999 χρησιμοποιώντας τον localhost. Για την συγγραφή των script αυτοματοποιημένου ελέγχου χρησιμοποιείται η γλώσσα Python, στην έκδοση 2.7.14. Παρακάτω στην Εικόνα 24 φαίνεται σχηματικά η υλοποίηση των ελέγχων, κάθε κομμάτι της οποίας αναλύεται ως εξής:

- Μηχανικός ελέγχου

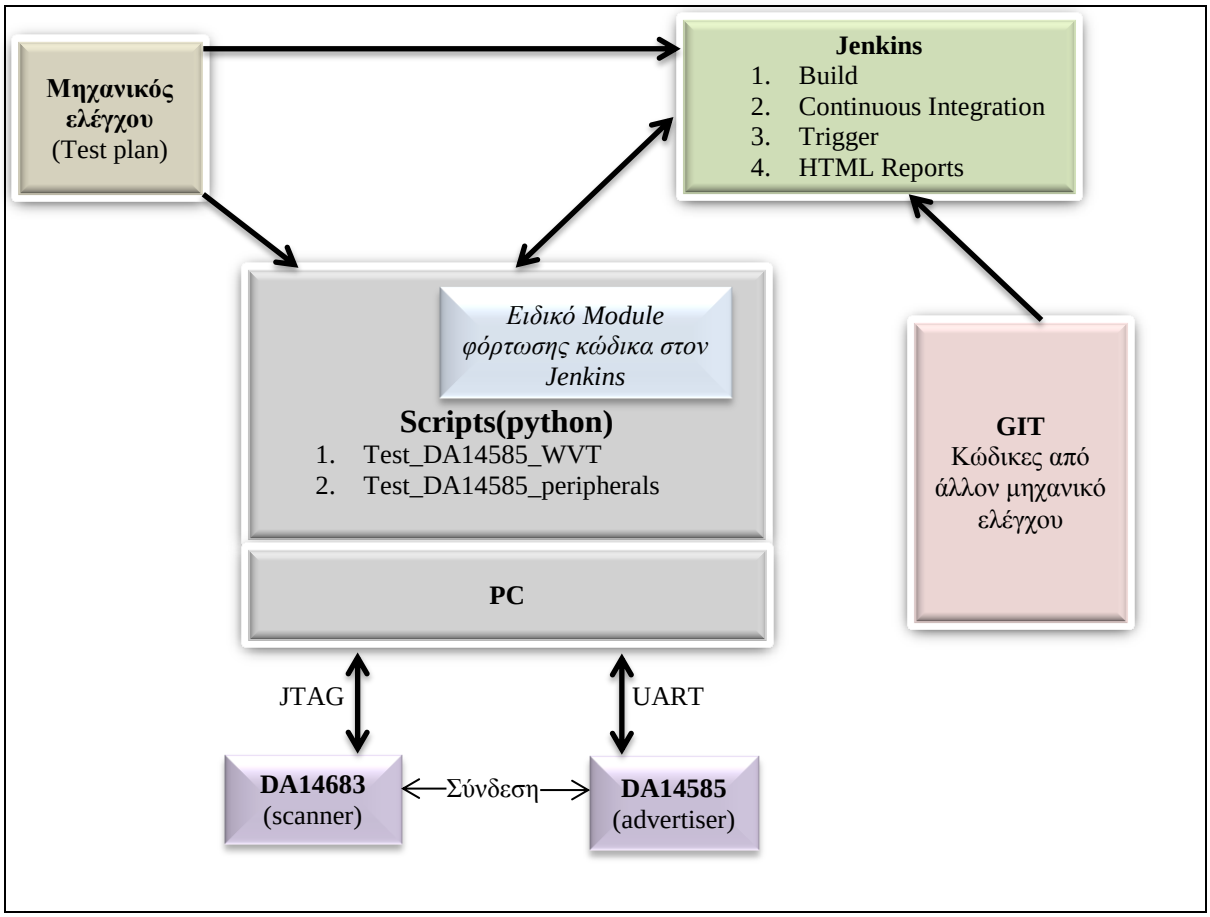
Αρχικά ο μηχανικός ελέγχου σχεδίασε και ανάπτυξε ένα test plan αυτοματοποιημένου ελέγχου. Το test plan περιέχει δύο είδη ελέγχων. Το πρώτο επικεντρώνεται στον έλεγχο του DA14585 έχοντας τον ρόλο του advertiser, ενώ το δεύτερο επικεντρώνεται στον έλεγχο των περισσότερων περιφερειακών του DA14585. Ο μηχανικός επίσης είναι υπεύθυνος για την ρύθμιση των παραμέτρων στον Jenkins έτσι ώστε να κάνει build σωστά σε κάθε test script και να εμφανίζει τα αποτελέσματα ξεχωριστά σε σελίδες HTML.

- Jenkins

Ο Jenkins έχει ρυθμιστεί από τον μηχανικό ελέγχου έτσι ώστε να συνδέεται με τον private λογαριασμό Git της εταιρείας (δίνοντας τα κατάλληλα στοιχεία εισόδου) και να λαμβάνει από εκεί το repository που αφορά το SDK (software development kit) του DA14585. Επιπλέον, ο Jenkins έχει ρυθμιστεί με τέτοιο τρόπο έτσι ώστε να εμφανίζει ξεχωριστά σε σελίδες HTML τα αποτελέσματα του τελευταίου build κάθε ξεχωριστού job, όπου επίσης φαίνεται η ώρα και η διάρκεια υλοποίησης κάθε test.

- Python scripts

Τα scripts αυτοματοποιημένου ελέγχου, τα οποία έχουν αναπτυχθεί σε python υλοποιούν δύο είδη ελέγχων. Το πρώτο είδος αφορά τον έλεγχο του DA14585 στον ρόλο του advertiser και το δεύτερο ελέγχει τα peripherals του DA14585. Περισσότερες πληροφορίες για τα test scripts φαίνονται στην παράγραφο 7.3.3. Επιπλέον για κάθε ξεχωριστό έλεγχο αναπτύχθηκε με τις γλώσσες Python και HTML, script που οπτικοποιεί τα τελικά αποτελέσματα σε σελίδα η οποία περιέχει επίσης links σε άλλες σελίδες που παρουσιάζουν επακριβώς τις λεπτομέρειες κάθε ξεχωριστού ελέγχου. Τα συγκεκριμένα scripts χρησιμοποιούνται από τον Jenkins για την παρουσίαση των αποτελεσμάτων.



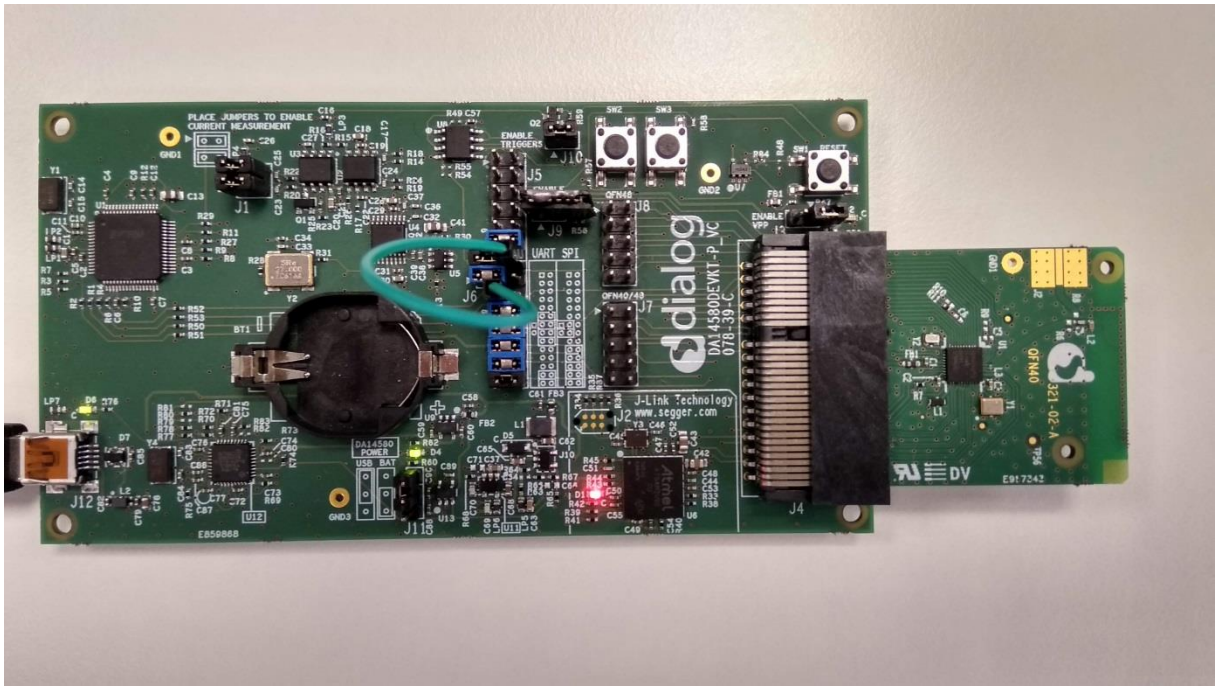
Εικόνα 24:Σχηματική απεικόνιση αυτοματοποιημένων ελέγχων για το chip DA14585

- DA14585

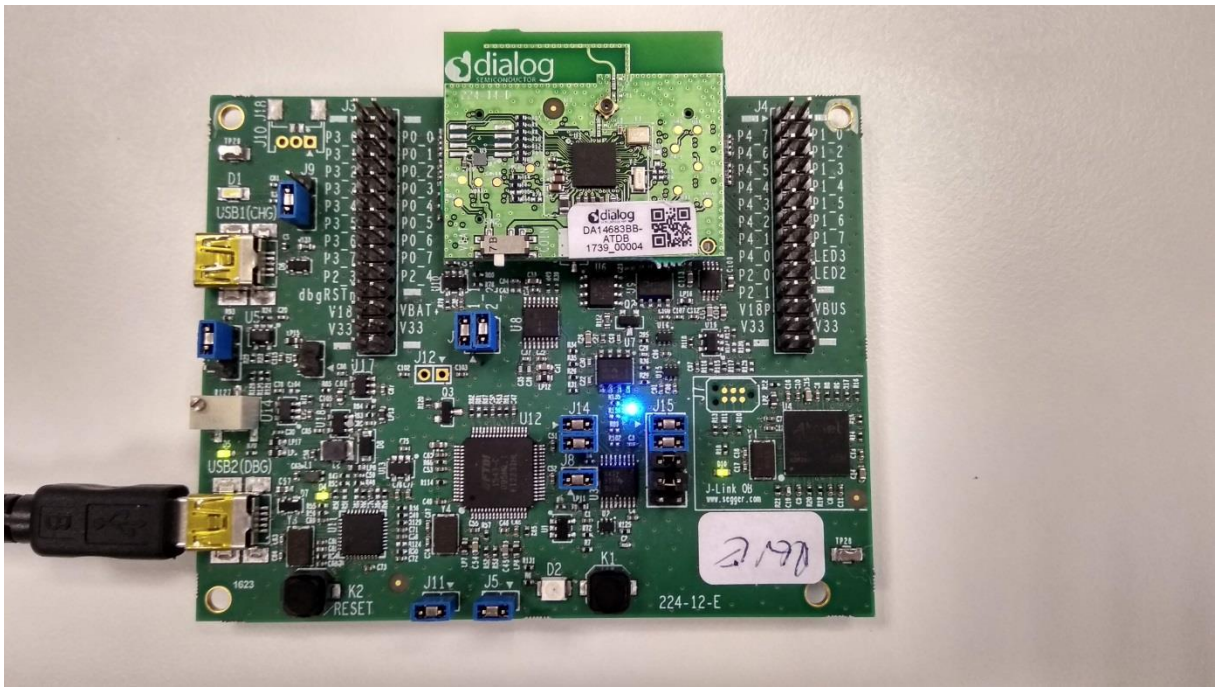
Το chip DA14585 συνδέεται με τον υπολογιστή σειριακά μέσω UART, ενώ κάθε φορά που ξεκινά ένα νέο test, το chip επαναπρογραμματίζεται μέσω του κώδικα SDK που κατεβαίνει μέσω του Jenkins από το Git repository. Ο κώδικας αυτός ουσιαστικά προγραμματίζει το DA14585 να λειτουργεί ως advertiser και να στέλνει advertising data σε όποιον τα απαιτήσει. Η παραπάνω διαδικασία γίνεται μέσω των εργαλείων Keil και SmartSnippets. Το πρώτο μεταγλωττίζει τον κώδικα που κατεβαίνει από τον Jenkins και το δεύτερο περνά τον κώδικα μέσω UART πάνω στην OTP του chip, δηλαδή προγραμματίζει το chip. Το DA14585 φαίνεται στην Εικόνα 25.

- DA14683

Το DA14683 συνδέεται με τον υπολογιστή σειριακά μέσω JTAG και χρησιμοποιείται για την πρώτο είδος ελέγχων. Αναλαμβάνει τον ρόλο του scanner, συνδέεται με το DA14585 και διαβάζει την βάση του. Το DA14683 φαίνεται στην Εικόνα 26.



Εικόνα 25:DA14585 chip



Εικόνα 26:DA14683 chip

- Git

Το Git παρέχει μέσω Jenkins τους απαραίτητους κώδικες και SDKs για την υλοποίηση των ελέγχων, σαν σύνολο από ένα repository. Οι κώδικες προέρχονται από άλλον μηχανικό ελέγχου (ή μηχανικούς), ο οποίος έχει κάνει commit, pull και push κάθε πιθανή αλλαγή, στο κεντρικό repository. Οι εν λόγω κώδικες είναι πιθανό να αλλάξουν, γι αυτό το λόγο κάθε φορά που ο Jenkins κάνει κάποιο νέο build σε ένα job, τους ξανα κατεβάζει από την αρχή αυτόματα και τους δίνει στα scripts αυτοματοποιημένου ελέγχου, τα οποία με την σειρά τους αυτόματα θα μεταγλωττίσουν τον νέο κώδικα και θα τον περάσουν μέσω UART στο DA14585.

7.3.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΩΝ SCRIPTS

7.3.3.1 DA14585_WVT TESTS

Τα scripts του πρώτου είδους ελέγχου αυτοματοποιούν την λειτουργία του WVT, το οποίο είναι ένα BLE εργαλείο ελέγχου του DA14585 ανεπτυγμένο σε python. Ουσιαστικά, έχουν αναπτυχθεί ειδικά scripts σε python για το εργαλείο WVT και στην συνέχεια το script αυτοματοποιημένου ελέγχου καλεί το WVT να τρέξει αυτόματα τα scripts. Με τον τρόπο αυτό εκτελούνται οι παρακάτω ελέγχοι αυτόματα ο ένας μετά τον άλλον (Πίνακας 7), χωρίς την επέμβαση ανθρώπου, τρέχοντας απλά ένα μόνο script:

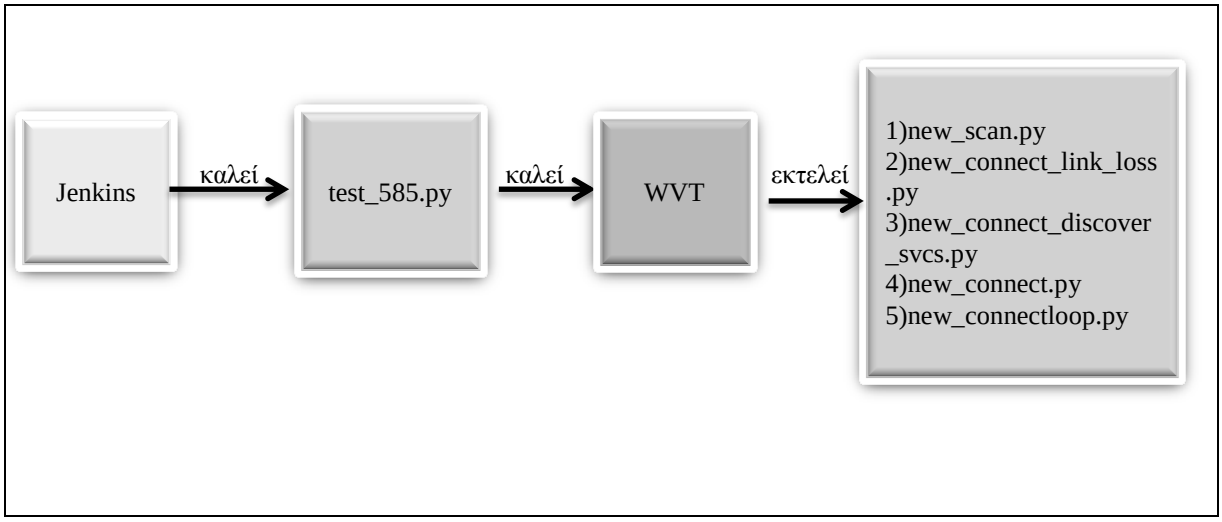
Πίνακας 7: DA14585 tests_WVT

new_scan.py	Στο script αυτό, το chip DA14683 σε ρόλο scanner, σκανάρει την γύρω περιοχή για κοντινά devices, έως ότου βρει τη διεύθυνση του DA14585. Εάν δεν το βρεί και ταυτόχρονα έχουν περάσει αρκετά λεπτά, εμφανίζει το μήνυμα 'Error advertiser not found'. Εάν το βρει, το σκανάρισμα σταματά. Όλα τα αποτελέσματα, μαζί με τις συσκευές που εντόπισε εγγράφονται στο αρχείο κειμένου log_[new_scan].txt, το οποίο ανανεώνεται σε κάθε νέα εκτέλεση του script.
new_connect.py	Στο script αυτό, εφόσον ο scanner έχει βρεί το chip DA14585 στο προηγούμενο βήμα, συνδέονται οι δύο συσκευές (DA14683 με DA14585). Περνά κάποιο εύλογο χρονικό διάστημα (10 sec) και στην συνέχεια γίνεται disconnect. Η παραπάνω διαδικασία είναι επιτυχής μόνο εάν γίνει σωστή σύνδεση και αποσύνδεση και η σύνδεση διατηρείται κατά το χρονικό διάστημα που έχει ορίσει ο χρήστης. Τα αποτελέσματα μαζί με τις παραμέτρους της σύνδεσης φαίνονται στο αρχείο κειμένου log_[new_connect].txt, το οποίο ανανεώνεται σε κάθε νέα εκτέλεση του script.
new_connectloop.py	Στο script αυτό, οι δύο συσκευές συνδέονται και αποσυνδέονται 50 φορές. Κάθε σύνδεση διαρκεί 60 δευτερόλεπτα. Εάν όλες οι συνδέσεις και οι αποσυνδέσεις έχουν γίνει σωστά, ο έλεγχος είναι επιτυχής. Αν όχι, ο έλεγχος είναι ανεπιτυχής. Τα αποτελέσματα μαζί με τις παραμέτρους της σύνδεσης για κάθε σύνδεση, φαίνονται στο αρχείο κειμένου log_[new_connectloop].txt, το οποίο ανανεώνεται σε κάθε νέα εκτέλεση του script.
new_connect_discover_svcs.py	Στο αρχείο αυτό, εφόσον συνδεθούν οι δύο συσκευές, ο master (DA14683) διαβάζει τις services του slave (DA14585). Το script ελέγχει αν ο slave έχει τις services που του αντιστιχούν. Εάν όχι, το αποτέλεσμα του ελέγχου είναι αρνητικό. Τα αποτελέσματα μαζί με τις παραμέτρους σύνδεσης φαίνονται στο αρχείο κειμένου log_[new_connect_dicover_svsv].txt, το οποίο ανανεώνεται σε κάθε νέα εκτέλεση του script. Μέσα στο αρχείο αποτελεσμάτων φαίνονται και οι λεπτομέρειες κάθε service καθώς και το ανάλογο μήνυμα εύρεσης του service.
new_connect_link_loss.py	Στο αρχείο αυτό, εφόσον υπάρξει επιτυχημένη σύνδεση, ο master διαβάζει την βάση του slave για να εντοπίσει την service Link Loss. Εάν υπάρχει η συγκεκριμένη service, ο master κάνει discover την characteristic και διαβάζει την value της characteristic Link Loss. Εάν όλη η παραπάνω διαδικασία γίνει χωρίς λάθος και χωρίς να γίνει disconnect, ο έλεγχος είναι επιτυχημένος, αλλιώς όχι. Τα αποτελέσματα μαζί με τις παραμέτρους σύνδεσης και τα στοιχεία του service που διαβάζονται φαίνονται στο αρχείο κειμένου log_[new_connect_linkloss].txt, το οποίο ανανεώνεται σε κάθε νέα εκτέλεση του script.

Το script που καλεί αυτόματα τους παραπάνω ελέγχους λέγεται test_585.py. Μέσα σε αυτό το scripts γίνεται import ένα άλλο script που λέγεται wvttests.py, το οποίο καλεί αυτόματα το wvt να τρέξει τους παραπάνω ελέγχους. Όλα τα αποτελέσματα αποθηκεύονται και αξιοποιούνται αργότερα από τον Jenkins, όπως θα αναλυθεί

παρακάτω. Η οργάνωση και ο τρόπος εκτέλεσης του συγκεκριμένου ελέγχου φαίνεται σχηματικά στην Εικόνα 27. Το python script που καλεί το wvt ώστε να εκτελέσει αυτόματα τους παραπάνω ελέγχους είναι:

```
1  import os
2
3
4
5  def wvt_testcases(wvt_path):
6      path=os.chdir(wvt_path)
7      print os.getcwd()
8      os.system("python wvt.py -l   ./PXP_REPORTER_TC_LIST     ")
9
```



Εικόνα 27:Σχηματική απεικόνιση του Project DA14585_WVT_Tests

7.3.3.1.1 Ρυθμιση Jenkins για την υλοποιηση των ελεγχων DA14585_WVT_tests

Στην σελίδα configure του job 585_wvt_tests, ο Jenkins έχει ρυθμιστεί έτσι ώστε να λαμβάνει όλους τους κώδικες του master branch του repository SDK_585.git και να προσθέτει αυτόματα στην οθόνη εξόδου (console output) την ημερομηνία και ώρα του εκάστοτε build. Στις παραμέτρους που αφορούν το build έχει προστεθεί το directory του python script test_585.py καθώς και το directory του python αρχείου results_585.py που οπτικοποιεί τα αποτελέσματα σε καινούρια HTML σελίδα. Στις παραμέτρους που αφορούν τις post-build actions έχουν προστεθεί το directory των HTML, στον οποίο ο Jenkins θα βρεί τον κώδικα HTML για κάθε ξεχωριστό έλεγχο:

- results_585.html
- results_585_scanning.html
- results_585_connect.html
- results_585_connect_multipletimes.html
- discover_services.html
- discover Link Loss service.html
- results_Keil.html

Με την εφαρμογή των παραπάνω ρυθμίσεων ο Jenkins είναι έτοιμος για το πρώτο build στο job με όνομα 585_wvt_tests. Ο χρήστης δεν χρειάζεται να γνωρίζει την εσωτερική δομή κάθε script και τον τρόπο που εκτελείται και γενικώς πως γίνεται εσωτερικά ο κάθε έλεγχος. Το μόνο που χρειάζεται να κάνει ο χρήστης, για να πάρει τα αποτελέσματα των ελέγχων, είναι να πατήσει μόνο το κουμπί build στον Jenkins. Μετά από λίγα λεπτά, τα αποτελέσματα, θετικά ή αρνητικά, θα εμφανιστούν στην οθόνη.

7.3.3.1.2 Αποτελεσματα DA14585_WVT_tests

Τα αποτελέσματα του πρώτου είδους ελέγχων εμφανίζονται παρακάτω στην Εικόνα 28. Ο Jenkins έπειτα από κάθε build, δημιουργεί όπως ο χρήστης έχει ορίσει τις παρακάτω ιστοσελίδες HTML:

- Results_585

Πρόκειται για την κύρια σελίδα των αποτελεσμάτων όπου φαίνεται η τελική κατάσταση κάθε ελέγχου σε πίνακα: PASS για επιτυχή έλεγχο, FAIL για ανεπιτυχή έλεγχο. Επιπλέον, πάνω αριστερά αναγράφεται η ακριβής ημερομηνία και ώρα των ελέγχων, ενώ δίνονται πληροφορίες και για την συνολική διάρκεια των ελέγχων αλλά για τη διάρκεια κάθε ελέγχου ξεχωριστά. Κάτω αριστερά, ο χρήστης μπορεί εάν το επιθυμεί, να πατήσει πάνω στα link προκειμένου να διαβάσει τα αναλυτικά αποτελέσματα κάθε ξεχωριστού ελέγχου.

- Results_585_scanning

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται μέσω του python script new_scan.py.

- Results_585_connect

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται μέσω του python script new_connect.py.

- Results_585_connect_multipletimes

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται μέσω του python script new_connectloop.py.

- Discover_services

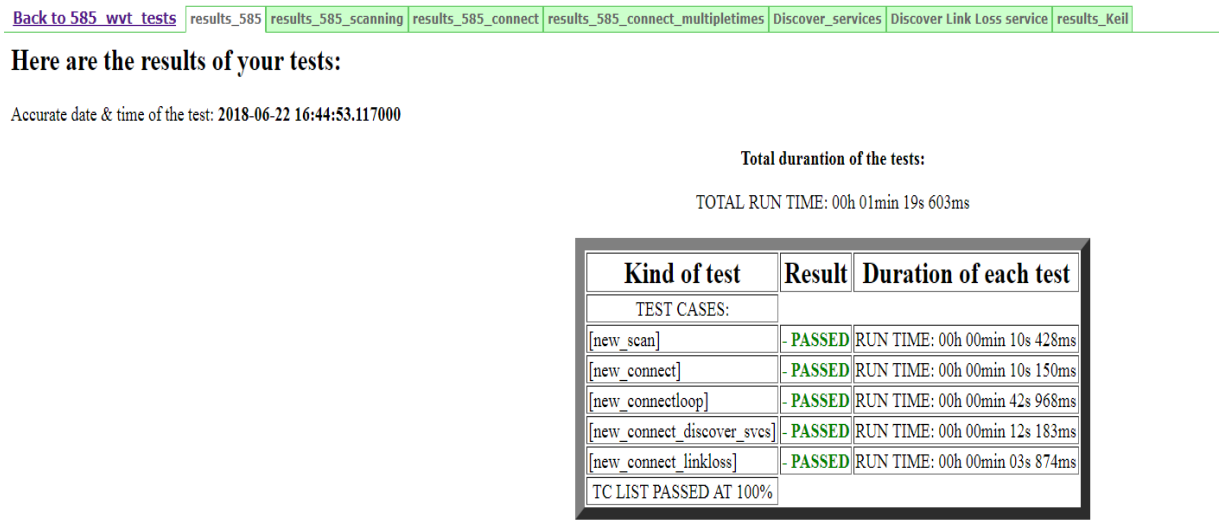
Πριέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται μέσω του python script new_connect_discover_svcs.py.

- Discover Link Loss service

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται μέσω του new_connect_link_loss.py.

- results_Keil

Περιέχει τα αποτελέσματα του αυτόματου compile που έγινε στον κώδικα που προήλθε από το GIT repository SDK_585.


Back to 585_wvt_tests | results_585 | results_585_scanning | results_585_connect | results_585_connect_multipletimes | Discover_services | Discover Link Loss service | results_Keil

Here are the results of your tests:

Accurate date & time of the test: **2018-06-22 16:44:53.117000**

Total durantion of the tests:

TOTAL RUN TIME: 00h 01min 19s 603ms

Kind of test	Result	Duration of each test
TEST CASES:		
[new_scan]	- PASSED	RUN TIME: 00h 00min 10s 428ms
[new_connect]	- PASSED	RUN TIME: 00h 00min 10s 150ms
[new_connectloop]	- PASSED	RUN TIME: 00h 00min 42s 968ms
[new_connect_discover_svcs]	- PASSED	RUN TIME: 00h 00min 12s 183ms
[new_connect_linkloss]	- PASSED	RUN TIME: 00h 00min 03s 874ms
TC LIST PASSED AT 100%		

Click or right Click(new tab) any of the following links in order to view the logs in details:

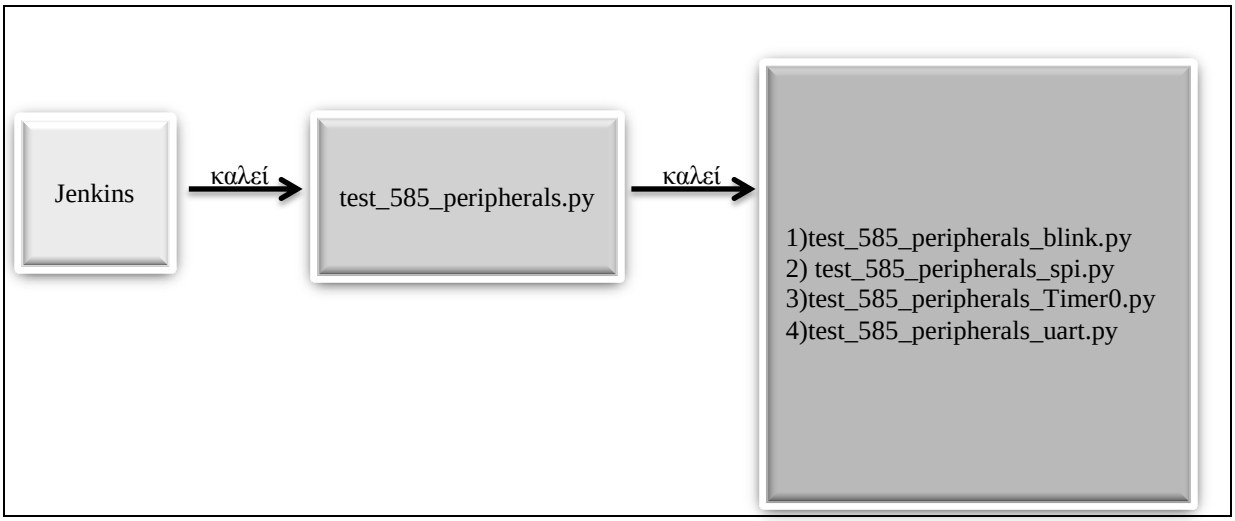
Compile log(Keil)
Scanning log
Connect log
Connect(loops) log
Discover_Services_log
Discover Characteristics of Link Loss Service

Εικόνα 28:Results_DA14585_WVT_tests

7.3.3.2 DA14585_ PERIPHERALS_TESTS

Τα scripts του δεύτερου είδους αυτοματοποιημένου ελέγχου εξετάζουν διάφορες περιφερειακές συσκευές (peripherals) του DA14585. Για κάθε ξεχωριστό peripheral έχει αναπτυχθεί ειδικό python script που εξετάζει την λειτουργία του. Όλα τα scripts καλούνται και τρέχουν αυτόματα από το κεντρικό python script test_585_peripherals.py. Στο σημείο αυτό θα πρέπει να σημειωθεί πως για κάθε ξεχωριστό peripheral που εξετάζεται, το DA14585 προγραμματίζεται μέσω του προγράμματος SmartSnippets Toolbox, αυτόματα, έτσι

ώστε να υλοποιεί τον επιθυμητό έλεγχο στο αντίστοιχο peripheral. Δηλαδή πριν από κάθε ξεχωριστό έλεγχο peripheral, ο Jenkins έχει ρυθμιστεί ώστε να λαμβάνει όλους τους κώδικες του master branch του repository SDK_585.git, έπειτα κάθε ξεχωριστό script ελέγχου κάποιου peripheral, λαμβάνει από τον φάκελο SDK_585 το κώδικα ελέγχου που το αφορά. Στην συνέχεια ο κώδικας αυτός, μεταγλωττίζεται από τον Keil και περνά μέσω UART στην OTP του DA14585, όπου τέλος μέσω του Smartsnippets το chip προγραμματίζεται και έχει τον συγκεκριμένο κώδικα στην μνήμη του. Το chip, τότε είναι έτοιμο για τον έλεγχο του αντίστοιχου peripheral. Το συγκεκριμένο είδος αυτοματοποιημένου ελέγχου έχει σχεδιαστεί έτσι ώστε, το κεντρικό python script να καλεί τα scripts ελέγχου κάθε ξεχωριστού peripheral, το ένα μετά το άλλο, αυτόματα. Η οργάνωση και ο τρόπος εκτέλεσης του συγκεκριμένου ελέγχου φαίνεται σχηματικά στην Εικόνα 29. Με τον παρακάτω τρόπο εκτελούνται οι έλεγχοι που φαίνονται στον Πίνακας 8, με την σειρά που απεικονίζονται.



Εικόνα 29:Σχηματική απεικόνιση του Project DA14585_PERIPHELAS_tests

Πίνακας 8: DA14585_Peripherals_tests

test_585_peripherals_blink.py	Στο script αυτό, ελέγχεται η λειτουργία της περιφερειακής συσκευής blink. Συγκεκριμένα, η συσκευή προγραμματίζεται έτσι ώστε να εκτελέσει τον κώδικα ελέγχου του blinky (που προέρχεται από το Git) αυτόματα. Αν όλα λειτουργούν σωστά, ο χρήστης παρατηρεί το φωτάκι του DA14585 να αναβοσβήνει για λίγα δευτερόλεπτα, για όσο διαρκεί το test δηλαδή.
test_585_peripherals_spi.py	Στο script αυτό, ελέγχεται η λειτουργία της περιφερειακής συσκευής spi. Πρόκειται για την μνήμη του chip και η συσκευή προγραμματίζεται ώστε να εκτελέσει τον κώδικα ελέγχου της μνήμης SPI(που προέρχεται από το Git) αυτόματα. Ο έλεγχος χαρακτηρίζεται επιτυχής, μόνο εάν διαβάζονται κάποιες προκαθορισμένες γραμμές στην μνήμη.
test_585_peripherals_Timer0.py	Στο script αυτό, ελέγχεται αυτόματα ο timer της συσκευής. Η συσκευή προγραμματίζεται ώστε να εκτελεί αυτόματα τον κώδικα ελέγχου του Timer0 (που προέρχεται από το Git).
test_585_peripherals_uart.py	Στο script αυτό, ελέγχεται αυτόματα η UART. Συγκεκριμένα, ελέγχεται εάν στα περιεχόμενα εμφανίζεται ένα προκαθορισμένο μήνυμα, όπως ορίζεται από τον κώδικα ελέγχου. Η συσκευή προγραμματίζεται ώστε να εκτελεί αυτόματα τον κώδικα ελέγχου της UART (που προέρχεται από το Git).

Όλα τα αποτελέσματα αποθηκεύονται και αξιοποιούνται αργότερα από τον Jenkins, όπως θα αναλυθεί παρακάτω**.**

7.3.3.2.1 ΡΥΘΜΙΣΗ JENKINS ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΤΩΝ ΕΛΕΓΧΩΝ DA14585_TESTS_PERIPHERALS

Στην σελίδα configure του job 585_test_peripherals, ο Jenkins έχει ρυθμιστεί έτσι ώστε να λαμβάνει όλους τους κώδικες του master branch του repository SDK_585.git και να προσθέτει αυτόματα στην οθόνη εξόδου (console output) την ημερομηνία και ώρα του εκάστοτε build. Στις παραμέτρους που αφορούν το build έχει προστεθεί το directory του python script test_585_peripherals.py καθώς και το directory του python αρχείου results_585_peripherals.py που οπτικοποιεί τα αποτελέσματα σε καινούρια HTML σελίδα. Στις παραμέτρους που αφορούν τις post-build actions έχουν προστεθεί το directory των HTML, στον οποίο ο Jenkins θα βρεί τον κώδικα HTML για κάθε ξεχωριστό έλεγχο:

- results_585_Peripherals.html
- results_Keil_blink.html
- results_Keil_SPI.html
- results_Keil_TIMER0.html
- results_Keil_uart.html
- results_BLINK.html
- results_TIMER0.html
- results_SPI.html
- results_uart.html

Με την εφαρμογή των παραπάνω ρυθμίσεων ο Jenkins είναι έτοιμος για το πρώτο build στο job με όνομα 585_test_peripherals. Ο χρήστης δεν χρειάζεται να γνωρίζει την εσωτερική δομή κάθε script και τον τρόπο που εκτελείται και γενικώς πως γίνεται εσωτερικά ο κάθε έλεγχος. Το μόνο που χρειάζεται να κάνει, για να πάρει τα αποτελέσματα των ελέγχων, είναι να πατήσει μόνο το κουμπί build στον Jenkins. Μετά από λίγα λεπτά, τα αποτελέσματα, θετικά ή αρνητικά, θα εμφανιστούν στην οθόνη.

7.3.3.2.2 ΑΠΟΤΕΛΕΣΜΑΤΑ DA14585_TESTS_PERIPHERALS

Τα αποτελέσματα του δεύτερου είδους ελέγχων εμφανίζονται παρακάτω στην Εικόνα 30. Ο Jenkins έπειτα από κάθε build, δημιουργεί όπως ο χρήστης έχει ορίσει τις παρακάτω ιστοσελίδες HTML:

- Results_585_Peripherals

Πρόκειται για την κύρια σελίδα των αποτελεσμάτων όπου φαίνεται η τελική κατάσταση κάθε ελέγχου σε πίνακα: PASS για επιτυχή έλεγχο, FAIL για ανεπιτυχή έλεγχο. Επιπλέον, πάνω αριστερά αναγράφεται η ακριβής ημερομηνία και ώρα των ελέγχων. Κάτω αριστερά, ο χρήστης μπορεί εάν το επιθυμεί, να πατήσει πάνω στα link προκειμένου να διαβάσει τα αναλυτικά αποτελέσματα κάθε ξεχωριστού ελέγχου.

- Results_BLINK

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται από το python script test_585_peripherals_blink.py.

- Results_TIMER0

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται από το python script test_585_peripherals_Timer0.py.

- Results_SPI

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται από το python script test_585_peripherals_spi.py.

- Results_uart

Περιέχει αναλυτικά αποτελέσματα του ελέγχου που γίνεται από το python script test_585_peripherals_uart.py.

- results_Keil_blink

Περιέχει τα αποτελέσματα του αυτόματου compile που έγινε στον κώδικα ελέγχου 'blinky.uvprojx' που προήλθε από το GIT repository SDK_585.

- results_Keil_SPI

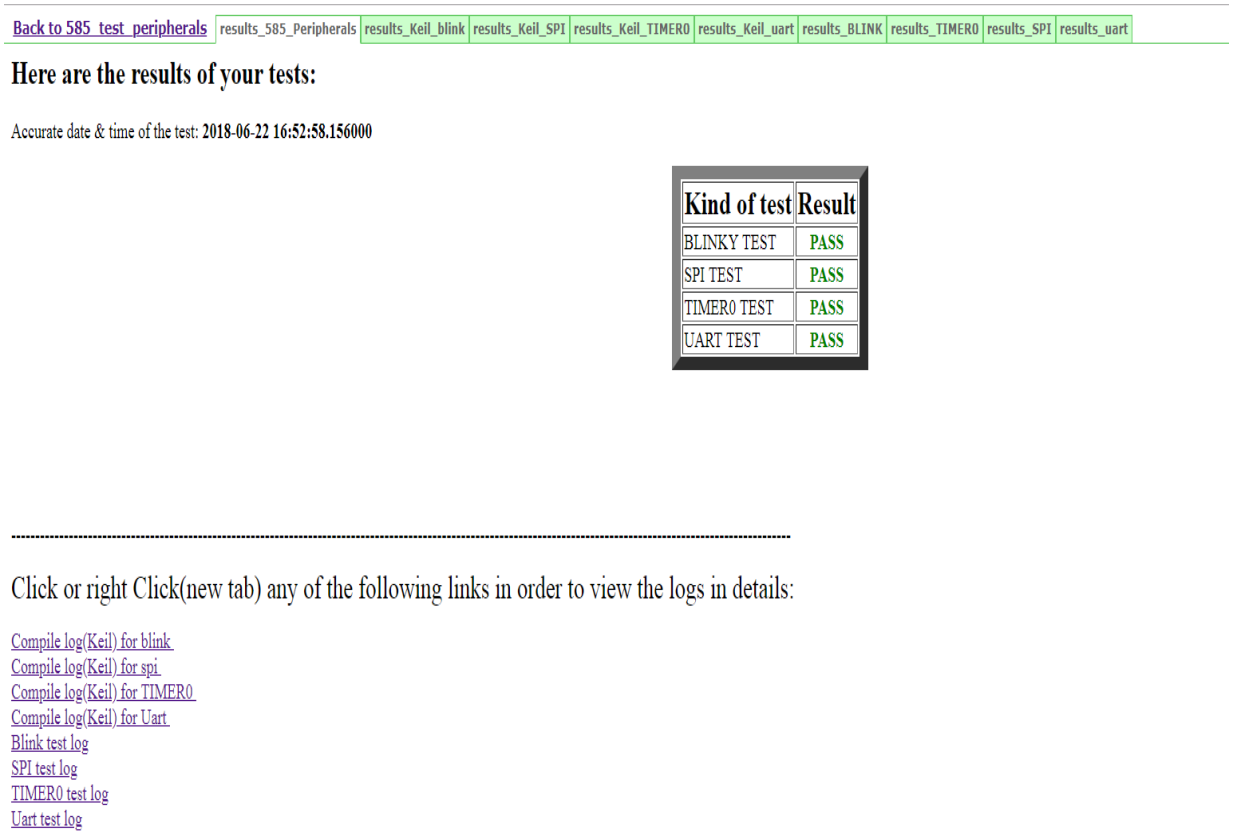
Περιέχει τα αποτελέσματα του αυτόματου compile που έγινε στον κώδικα ελέγχου 'spi_flash.uvprojx' που προήλθε από το GIT repository SDK_585.

- results_Keil_TIMER0

Περιέχει τα αποτελέσματα του αυτόματου compile που έγινε στον κώδικα ελέγχου 'timer0_general.uvprojx' που προήλθε από το GIT repository SDK_585.

- results_Keil_uart

Περιέχει τα αποτελέσματα του αυτόματου compile που έγινε στον κώδικα ελέγχου 'uart.uvprojx' που προήλθε από το GIT repository SDK_585.



Εικόνα 30:Results DA14585_tests_peripherals

7.3.4 PRE-CONFIGURATION

Οι αυτοματοποιημένοι έλεγχοι που αναλύθηκαν στις παραγράφους 7.3.3.1 και 7.3.3.2, έχουν σχεδιαστεί με τέτοιο τρόπο έτσι ώστε να προσαρμόζονται στις ανάγκες κάθε μηχανικού ελέγχου που ενδέχεται να τους χρησιμοποιήσει. Μπορεί τα αποτελέσματα των ελέγχων να αφορούν το DA14585 chip αλλά με τις κατάλληλες αλλαγές, τα scripts μπορούν να χρησιμοποιηθούν για τον έλεγχο οποιουδήποτε άλλου chip από τον Πίνακας 6, με οποιοδήποτε είδος JTAG ή UART. Επιπλέον, ο χρήστης έχει τη δυνατότητα αλλαγής του directory κάθε εργαλείου που χρησιμοποιείται.

7.3.4.1 PRE-CONFIGURATION DA14585_WVT_TESTS

Για το πρώτο είδος ελέγχων, στο κύριο python script με όνομα test_585.py υπάρχει ειδικό κομμάτι του κώδικα, πριν ακόμη κληθεί το wvt, όπου οι κύριες μεταβλητές και παράμετροι του ελέγχου έχουν οριστεί σαν global, προκειμένου να χρησιμοποιούνται από κάθε συνάρτηση και από κάθε έλεγχο. Παράλληλα έχει προστεθεί κατάλληλο σχόλιο που εξηγεί στον χρήστη ακριβώς την λειτουργία κάθε παραμέτρου, έτσι ώστε να είναι σε θέση να την αλλάξει εάν το επιθυμεί. Τέλος, επειδή η σύνδεση με το DA14585 γίνεται μέσω σειριακής σύνδεσης UART, είναι εύκολο να υπάρξει σύγχυση για την θύρα που θα χρησιμοποιηθεί. Για αυτό τον λόγο αυτό, έχει προστεθεί κατάλληλο σχόλιο που εξηγεί αναλυτικά στον χρήστη τι θα πρέπει να κάνει για να αλλάξει την θύρα που χρησιμοποιεί το WVT. Το πεδίο πιθανών αλλαγών στο είδος ελέγχων DA14585_WVT_tests φαίνεται στην Εικόνα 31.

7.3.4.2 PRECONFIGURATION DA14585_PERIPHERALS_TESTS

Για το δεύτερο είδος ελέγχων, στο κύριο python script με όνομα test_585_peripherals, υπάρχει ειδικό κομμάτι κώδικα, πριν ακόμη κληθούν τα scripts αυτοματοποιημένου ελέγχου κάθε ξεχωριστού peripheral, όπου οι κύριες μεταβλητές και παράμετροι έχουν οριστεί ως global προκειμένου να χρησιμοποιούνται από κοινού από κάθε script ελέγχου. Επιπλέον, έχουν προστεθεί κατάλληλα σχόλια που εξηγούν στον χρήστη την λειτουργία κάθε παραμέτρου, έτσι ώστε να είναι σε θέση να την αλλάξει εάν το επιθυμεί. Τέλος, επειδή η σύνδεση με το DA14585 γίνεται μέσω σειριακής σύνδεσης UART, είναι εύκολο να υπάρξει σύγχυση για την θύρα που θα

χρησιμοποιηθεί. Έτσι, στον κώδικα υπάρχει ειδικός έλεγχος, που ελέγχει την θύρα. Το πεδίο πιθανών αλλαγών στο είδος ελέγχων DA14585__PERIPHERALS_Tests αλλά και ο έλεγχος για την θύρα φαίνεται στην Εικόνα 32.

```
##########################################################################################
#global values for keil
#path of the programm UV4.exe
global keil_path
keil_path='C:\Keil_v5\UV4\UV4.exe'
#path of the prox_reporter.uvprojx file
global prox_reporter_uvprojx_path
prox_reporter_uvprojx_path='C:\\Users\\ptlab1\\585\\SDK_585\\projects\\target_apps\\ble_examples\\prox_reporter\\Keil_5\\prox_reporter.uvprojx'
# path of the resutls txt file
global path_txt
path_txt='C:\Users\ptlab1\Desktop\\585\Results_Keil_585.txt '
##########################################################################################
#global values for SmartSnippets
global SmartSnippetsToolbox_path
SmartSnippetsToolbox_path = 'C:\DiaSemi\SmartSnippetsStudio\Toolbox\SmartSnippetsToolbox.exe'
#number CHIP
global type_of_chip
type_of_chip='DA14585-00'
#number JTAG
global type_of_JTAG
type_of_JTAG='480066802'
#path of the .hex file
global hex_path
hex_path='C:\Users\ptlab1\\585\SDK_585\projects\\target_apps\\ble_examples\prox_reporter\Keil_5\out_585\prox_reporter_585.hex'
##########################################################################################################################
#define the path of wvt
global wvt_path
wvt_path='C:\Users\ptlab1\Desktop\\rw-ble-app-v8_1_3\\tool\wvt\src'
##define the path of logs of wvt
#you can change the path of the logs of wvt at python script: Results_585-->line:21 .Be careful: this is important if you re going to change the entire path of wvt
##########################################################################################################################
# for change of the port go to C:\Users\ptlab1\Desktop\rw-ble-app-v8_1_3\tool\wvt\config, open the dev_plf.txt file and line 19, column 7 -->
# --> and change the COM PORT NAME.
```

Εικόνα 31:Πεδίο αλλαγών για το DA14585_WVT_Tests

```
#############################################################################################


#global values for the port,Uart
global COM_PORT_NAME
COM_PORT_NAME='COM48'
global BAUDRATE
BAUDRATE=115200
global TIME_OUT
TIME_OUT=1
#global address for DA14585
global ADDRESS
ADDRESS= ['\x80', '\xea', '\xca', '\x70', '\x00', '\x06']

try:
    ser = serial.Serial(COM_PORT_NAME, BAUDRATE)
    check=0
    ser.close()
except serial.serialutil.SerialException:
    check=1
    print "The device is Not connected..the port name is propably wrong"
```

Εικόνα 32:Πεδίο αλλαγών για το DA14585_PERIPHERALS_Tests

ΠΑΡΑΡΤΗΜΑ Α

ΤΑ ΚΑΛΥΤΕΡΑ ΣΥΓΧΡΟΝΑ ΕΡΓΑΛΕΙΑ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟΥ ΕΛΕΓΧΟΥ

Εργαλείο αυτοματοποιημένου ελέγχου	Selenium	Katalon Studio	Unified Functional Testing	Test Complete	Watir
Έτος κυκλοφορίας	2004	2015	1998	1999	2008
Εφαρμογές και συστήματα ελέγχου όπου εφαρμόζεται	Web apps	Web(UI & API) Mobile apps	Web(UI & API) Mobile, Desktop, Packaged apps	Web(UI & API), Mobile, Desktop apps	Web apps
Τιμή	Δωρεάν	Δωρεάν	$$$$	$$	Δωρεάν
Πλατφόρμες υλοποίησης	Windows, Linux, OS X	Windows, Linux, OS X	Windows	Windows	Windows, Linux, OS X
Γλώσσες συγγραφής κώδικα αυτοματοποίησης	Java, C#, Perl, Python, Javascript, Ruby, PHP	Java/Groovy	VBScript	JavaScript, Python, VBScript, Jscript, Delphi, C++,C#	Ruby
Ικανότητες προγραμματισμού	Προηγμένες δεξιότητες προγραμματισμού	Δεν απαιτούνται δεξιότητες προγραμματισμού	Δεν απαιτούνται δεξιότητες προγραμματισμού	Δεν απαιτούνται δεξιότητες προγραμματισμού	Προηγμένες δεξιότητες προγραμματισμού
Επίπεδο δυσκολίας στην εγκατάσταση και χρήση	Προυποθέτει εμπειρία και αυξημένες ικανότητες για εγκατάσταση και χρήση	Εύκολο στην εγκατάσταση και την χρήση	Πολύπλοκη εγκατάσταση, προϋποθέτει ειδική εκπαίδευση για την χρήση του	Εύκολη εγκατάσταση, προϋποθέτει ειδική εκπαίδευση για την χρήση του	Προϋποθέτει δεξιότητες και αυξημένες ικανότητες για χρήση και εγκατάσταση

ΠΑΡΑΡΤΗΜΑ Β

ΣΥΓΧΡΟΝΕΣ ΘΕΣΕΙΣ ΕΡΓΑΣΙΑΣ ΣΧΕΤΙΚΕΣ ΜΕ ΤΟΝ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΟ ΕΛΕΓΧΟ

Τίτλος θέσης εργασίας	**Περιγραφή θέσης εργασίας**
Junior Test Engineer	Μια εισαγωγική θέση για κάποιον με πτυχίο στην επιστήμη των υπολογιστών ή για κάποιον με μια μικρή εμπειρία σε χειρωνακτικούς ελέγχους λογισμικού. Αρμοδιότητες: Προγραμματισμός κώδικα ελέγχου και εξοικείωση με τον κύκλο ζωής ελέγχου λογισμικού καθώς και με τις τεχνικές ελέγχου.
Test Engineer/ Programmer Analyst	Ένας μηχανικός ελέγχου ή προγραμματιστής που διαθέτει εμπειρία δύο χρόνων. Αρμοδιότητες: Προγραμματισμός κώδικα αυτοματοποιημένου ελέγχου και εισαγωγή στον ρόλο του ηγέτη στη διαδικασία ανάπτυξης προγραμμάτων ελέγχου. Προϋποθέσεις: εμπειρία στον προγραμματισμό, στα λειτουργικά συστήματα, στα δίκτυα και τις βάσεις δεδομένων.
Senior Test Engineer/ Programmer Analyst	Ένας μηχανικός ελέγχου ή προγραμματιστής που διαθέτει τρία έως τέσσερα χρόνια εμπειρίας. Αρμοδιότητες: Ανάπτυξη ή συντήρηση προγραμμάτων ελέγχου, αξιολογεί και ενεργεί ως μέντορας σε άλλους πιο νέους μηχανικούς ελέγχου ή προγραμματιστές. Συνεχίζει να αναπτύσσει δεξιότητες στον προγραμματισμό, στα λειτουργικά συστήματα, στα δίκτυα και στις βάσεις δεδομένων.
Team Lead	Ένας μηχανικός ελέγχου ή προγραμματιστής που διαθέτει τέσσερα έως έξι χρόνια εμπειρίας. Αρμοδιότητες: Είναι υπεύθυνος και επιτηρεί μέχρι και τρείς μηχανικούς ελέγχου ή προγραμματιστές. Ευθύνεται για χρονικές προθεσμίες και κάνει εκτιμήσεις κόστους και προσπάθειας. Επικεντρώνεται περισσότερο σε τεχνικές δεξιότητες.
Test/ Programming Lead	Ένας μηχανικός ελέγχου ή προγραμματιστής που διαθέτει έξι έως και δέκα χρόνια εμπειρίας. Αρμοδιότητες: Είναι υπεύθυνος και επιτηρεί από τέσσερις έως και οκτώ μηχανικούς ελέγχου ή προγραμματιστές. Ευθύνεται για χρονικές προθεσμίες και κάνει εκτιμήσεις κόστους και προσπάθειας και την παράδοση προϊόντος εντός προθεσμίας, με το λιγότερο κόστος. Είναι υπεύθυνος για τον σχεδιασμό και την ανάπτυξη των τεχνικών χαρακτηριστικών του έργου, παρέχει και υπηρεσίες εξυπηρέτησης πελατών και κάνει παρουσιάσεις. Αναπτύσσει τεχνική εξειδίκευση σε συγκεκριμένους τομείς.
Test/ QA / Development (Project) Manager	Ένας μηχανικός ελέγχου ή προγραμματιστής που διαθέτει παραπάνω από δέκα χρόνια εμπειρίας. Αρμοδιότητες: Είναι υπεύθυνος για περισσότερους από οκτώ υπαλλήλους που απασχολούνται σε περισσότερα από ένα έργα. Ευθύνεται για τον κύκλο ζωής ανάπτυξης του λογισμικού στο τομέα ελέγχου. Έρχεται σε επαφή συχνά με πελάτες και κάνει πολλές παρουσιάσεις. Σχεδιάζει και προγραμματίζει τις διαδικασίες ελέγχου και ευθύνεται για την στελέχωση της ομάδας.
Program Manager	Εμπειρία πάνω από δεκαπέντε χρόνια σε δραστηριότητες ανάπτυξης και υποστήριξης ενεργειών που σχετίζονται με τον αυτοματοποιημένο έλεγχο. Αρμοδιότητες: Είναι υπεύθυνος για το προσωπικό που εκτελεί πλήθος εργασιών σε όλο τον κύκλο ανάπτυξης τους λογισμικού. Αναλαμβάνει την οργάνωση κάθε project και έχει την ευθύνη για κάθε πιθανή επιτυχία ή αποτυχία.

ΒΙΒΛΙΟΓΡΑΦΙΑ- ΑΝΑΦΟΡΕΣ

[1] McCall James A., "Quality Factors," *Encyclopedia of Software Engineering*, Jan. 2002.

[2] "Dialog Semiconductor overview." [Online]. Available: https://www.dialog-semiconductor.com/sites/all/themes/dialog/ir-overview-2018/. [Accessed: 19-Jun-2018].

[3] "IEEE Standard for Software Quality Assurance Processes," *IEEE Std 730-2014 (Revision of IEEE Std 730-2002)*, pp. 1–138, Jun. 2014.

[4] L. Luo, "Software testing techniques," *Institute for software research international Carnegie mellon university Pittsburgh, PA*, vol. 15232, no. 1–19, p. 19, 2001.

[5] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, 2002.

[6] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*, 2007, pp. 85–103.

[7] S. Singh, G. Singh, and S. Singh, "Software testing," *International Journal of Advanced Research in Computer Science*, vol. 1, no. 3, 2010.

[8] J. Kasurinen, O. Taipale, and K. Smolander, "Software Test Automation in Practice: Empirical Observations," *Advances in Software Engineering*, 2010. [Online]. Available: https://www.hindawi.com/journals/ase/2010/620836/abs/. [Accessed: 23-Mar-2018].

[9] S. Duncan, "Software Quality Assurance: From Theory to Implementation," *Software Quality Professional*, vol. 7, no. 3, p. 42, 2005.

[10] J. M. Voas and K. W. Miller, "Software testability: The new verification," *IEEE software*, vol. 12, no. 3, pp. 17–28, 1995.

[11] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[12] B. Homès, *Fundamentals of software testing*. John Wiley & Sons, 2013.

[13] E. Dustin, J. Rashka, and J. Paul, *Automated software testing: introduction, management, and performance*. Addison-Wesley Professional, 1999.

[14] M. D. Ernst, "Static and dynamic analysis: Synergy and duality," in *WODA 2003: ICSE Workshop on Dynamic Analysis*, 2003, pp. 24–27.

[15] R. Ramler and K. Wolfmaier, "Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost," in *Proceedings of the 2006 International Workshop on Automation of Software Test*, New York, NY, USA, 2006, pp. 85–91.

[16] H. Q. Nguyen, M. Hackett, and B. K. Whitlock, *Happy About Global Software Test Automation: A Discussion of Software Testing for Executives*. Happy About, 2006.

[17] J. Pan, "Software testing," *Dependable Embedded Systems*, vol. 5, p. 2006, 1999.

[18] M. Fewster and D. Graham, *Software test automation: effective use of test execution tools*. ACM Press/Addison-Wesley Publishing Co., 1999.

[19] K. Li and M. Wu, *Effective software test automation: developing an automated software testing tool*. John Wiley & Sons, 2006.

[20] K. Naik and P. Tripathy, *Software testing and quality assurance: theory and practice*. John Wiley & Sons, 2011.

[21] A. Cockburn, *Agile software development*, vol. 177. Addison-Wesley Boston, 2002.

[22] J. Tian, "Software quality engineering : testing, quality assurance and quantifiable improvement," 2005.

[23] D. A. Garvin, "What Does 'Product Quality' Really Mean?," *Sloan management review*, vol. 25, 1984.
[24] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]," *IEEE software*, vol. 13, no. 1, pp. 12–21, 1996.
[25] L. Briand and Y. Labiche, "Empirical studies of software testing techniques: Challenges, practical strategies, and future research," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, pp. 1–3, 2004.
[26] R. E. Al-Qutaish, "Quality models in software engineering literature: an analytical and comparative study," *Journal of American Science*, vol. 6, no. 3, pp. 166–175, 2010.
[27] E. Wallmuller, *Software quality assurance : a practical approach = Software-Qualitatssicherung in der Praxis. English*. New York: Prentice Hall, 1994.
[28] M. S. Deutsch and R. R. Willis, *Software quality engineering: a total technical and management approach*. Prentice-Hall, Inc., 1988.
[29] M. W. Evans and J. J. Marciniak, *Software quality assurance and management*. Wiley, 1987.
[30] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, and M. J. Merrit, *Characteristics of Software Quality. Vol. 1 Vol. 1*. Amsterdam; Lausanne; New York: Elsevier, 1978.
[31] R. G. Dromey, "A model for software product quality," *IEEE Transactions on software engineering*, vol. 21, no. 2, pp. 146–162, 1995.
[32] Ho-Won Jung, Seung-Gweon Kim, and Chang-Shin Chung, "Measuring Software Product Quality: A Survey of ISO/IEC 9126," *IEEE Softw. IEEE Software*, vol. 21, no. 05, pp. 88–92, 2004.
[33] N. M. A. Munassar and A. Govardhan, "A comparison between five models of software engineering," *IJCSI*, vol. 5, pp. 95–101, 2010.
[34] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.
[35] S. L. Pfleeger, "Software metrics: Progress after 25 years?," *IEEE Software*, vol. 25, no. 6, pp. 32–34, 2008.
[36] T. Gilb and S. Finzi, *Principles of software engineering management*, vol. 11. Addison-Wesley Reading, MA, 1988.
[37] S. H. Kan, *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.
[38] N. VanSuetendael and D. Elwell, "Software Quality Metrics," COMPUTER RESOURCE MANAGEMENT INC PLEASANTVILLE NJ, 1991.
[39] R. S. Pressman and B. R. Maxim, *Software engineering : a practitioner's approach*. New York: McGraw-Hill Education, 2015.
[40] J. M. Roche, "Software metrics and measurement principles," *ACM SIGSOFT Software Engineering Notes*, vol. 19, no. 1, pp. 77–85, 1994.
[41] L. O. Ejiogu, *Software engineering with formal metrics, 1991*. QED Technical Publishing Group: McGraw-Hill Book Company, Boston, Massachusetts.
[42] B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, *Predictably dependable computing systems*. Springer Science & Business Media, 2013.
[43] R. Chillarege, "What is software failure?," *IEEE Transactions on Reliability*, vol. 45, no. 3, p. 354, 1996.
[44] W. S. Humphrey, *A discipline for software engineering*. Addison-Wesley Longman Publishing Co., Inc., 1995.
[45] K. Edward, *Software Testing in the real world*. Pearson Education India, 1995.

[46] E. S. Raymond, *The cathedral and the bazaar-musings on Linux and open source by an accidental revoltionary (rev. ed.)*. O'reilly Beijing, 2001.
[47] B. Beizer, *Software system testing and quality assurance*. Van Nostrand Reinhold Co., 1984.
[48] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.
[49] IEEE Computer Society. and Standards Coordinating Committee., *IEEE standard glossary of software engineering terminology*. New York: Institute of Electrical and Electronics Engineers, 1991.
[50] S. Nidhra and J. Dondeti, "Black box and white box testing techniques-a literature review," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012.
[51] B. Beizer and J. Wiley, "Black box testing: Techniques for functional testing of software and systems," *iEEE Software*, vol. 13, no. 5, p. 98, 1996.
[52] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.
[53] A. H. Watson, D. R. Wallace, and T. J. McCabe, *Structured testing: A testing methodology using the cyclomatic complexity metric*, vol. 500. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996.
[54] W. E. Lewis, *Software testing and continuous quality improvement*. Auerbach publications, 2000.
[55] A. C. Coulter, "Graybox software testing methodology: embedded software testing technique," in *Digital Avionics Systems Conference, 1999. Proceedings. 18th*, 1999, vol. 2, pp. 10–A.
[56] D. Hutchison *et al.*, "Quality of Software Architectures. Models and Architectures 4th International Conference on the Quality of Software-Architectures, QoSA 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings," 2008. .
[57] S. Gupta and V. Dewan, "A Comparison between Five Models of Software Engineering."
[58] S. Easterbrook, "Software Lifecycles," *University of Toronto Department of Computer Science*, 2001.
[59] M. G. Bradac, D. E. Perry, and L. G. Votta, "Prototyping a process monitoring experiment," in *Proceedings of the 15th international conference on Software Engineering*, 1993, pp. 155–165.
[60] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.
[61] V. Rastogi, "Software Development Life Cycle Models-Comparison, Consequences," *International Journal of Computer Science and Information Security*, vol. 6, no. 1, pp. 168–172, 2015.
[62] M. Huo, J. Verner, L. Zhu, and M. A. Babar, "Software quality and agile methods," in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, 2004, pp. 520–525.
[63] K. Beck *et al.*, "Manifesto for agile software development," 2001.
[64] L. Crispin and J. Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams," 2009.

[65] D. T. Pham and R. S. Gault, "A comparison of rapid prototyping technologies," *International Journal of machine tools and manufacture*, vol. 38, no. 10–11, pp. 1257–1287, 1998.
[66] S. Vegas and V. Basili, "A characterisation schema for software testing techniques," *Empirical Software Engineering*, vol. 10, no. 4, pp. 437–466, 2005.
[67] M. Auguston, J. . Michael, M. Shing, and 16th IEEE International Workshop on Rapid System Prototyping (RSP'05), "Test Automation and Safety Assessment in Rapid Systems Prototyping," pp. 188–194, 2005.
[68] L. G. Hayes, *The Automated Testing Handbook*. Software Testing Institute, 2004.
[69] S. Berner, R. Weber, and R. K. Keller, "Observations and lessons learned from automated testing," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 571–579.
[70] I. Burnstein, *Practical software testing: a process-oriented approach*. Springer Science & Business Media, 2006.
[71] G. D. Everett and R. McLeod Jr, *Software testing: testing across the entire software development life cycle*. John Wiley & Sons, 2007.
[72] B. Hetzel, *The Complete Guide to Software Testing, Wellesley, MA: QED Information Sciences*. Inc, 1988.
[73] B. J. LeSuer and K. A. Adams, "Automated software testing," May-2007.
[74] E. Collins, A. Dias-Neto, and V. F. de Lucena Jr, "Strategies for agile software testing automation: An industrial experience," in *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, 2012, pp. 440–445.
[75] K. Karhu, T. Repo, O. Taipale, and K. Smolander, "Empirical observations on software testing automation," in *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*, 2009, pp. 201–209.
[76] "Bluetooth Low Energy - Part 1: Introduction To BLE." [Online]. Available: https://www.mikroe.com/blog/bluetooth-low-energy-part-1-introduction-ble. [Accessed: 18-Jun-2018].
[77] "Bluetooth Technology Website." [Online]. Available: https://www.bluetooth.com/. [Accessed: 18-Jun-2018].
[78] K. T. Davidson Carles Cufí, Akiba, Robert, *Getting Started with Bluetooth Low Energy*. .
[79] "Guide: Bluetooth low energy products | Dialog Semiconductor customer support." [Online]. Available: https://support.dialog-semiconductor.com/guide/bluetooth-low-energy. [Accessed: 19-Jun-2018].
[80] "Keil (company)," *Wikipedia*. 22-Mar-2018.
[81] J. D. Blischak, E. R. Davenport, and G. Wilson, "A quick introduction to version control with Git and GitHub," *PLoS computational biology*, vol. 12, no. 1, p. e1004668, 2016.
[82] "Concurrent Versions System - Summary [Savannah]." [Online]. Available: http://savannah.nongnu.org/projects/cvs. [Accessed: 20-Jun-2018].
[83] "Apache Subversion." [Online]. Available: https://subversion.apache.org/. [Accessed: 20-Jun-2018].
[84] J. F. Smart, *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. O'Reilly Media, Inc., 2011.
[85] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Software*, vol. 32, no. 2, pp. 50–54, 2015.
[86] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, vol. 122, p. 14, 2006.

[87] A. L. Jenkins, J. S. Harsany, and R. Perugini, "Test definition tool," Dec-1999.
[88] A. Berg, *Jenkins Continuous Integration Cookbook*. Packt Publishing Ltd, 2012.
[89] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 235–245.
[90] D. St\a ahl and J. Bosch, "Modeling continuous integration practice differences in industry software development," *Journal of Systems and Software*, vol. 87, pp. 48–59, 2014.
[91] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in GitHub," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 805–816.
[92] S. Stolberg, "Enabling agile testing through continuous integration," in *Agile Conference, 2009. AGILE'09.*, 2009, pp. 369–374.
[93] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.
[94] F. Slomka, M. Dorfel, R. Munzenberger, and R. Hofmann, "Hardware/software codesign and rapid prototyping of embedded systems," *IEEE Design & Test of Computers*, vol. 17, no. 2, pp. 28–38, 2000.
[95] K. Petersen, C. Wohlin, and D. Baca, "The waterfall model in large-scale development," in *International Conference on Product-Focused Software Process Improvement*, 2009, pp. 386–400.
[96] J. Tretmans, "Model-Based Testing," 2010.
[97] R. D. Beatty, "Computerlore: The Bit Bucket," *New York Folklore*, vol. 2, no. 3, p. 223, 1976.
[98] V. Pulkkinen, "Continuous deployment of software," in *Proc. of the Seminar*, 2013, vol. 58312107, pp. 46–52.
[99] I. Molyneaux, *The art of application performance testing: Help for programmers and quality assurance*. O'Reilly Media, Inc., 2009.
[100] Y. Bassil, "A simulation model for the waterfall software development life cycle," *arXiv preprint arXiv:1205.6904*, 2012.
[101] M. A. Jackson, "Problem Frames: Analysing and Structuring Software Development Problems pdf," 2001.
[102] Z. Reszela *et al.*, "Bringing Quality in the Controls Software Delivery Process," 2015.
[103] B. Potter and G. McGraw, "Software security testing," *IEEE Security & Privacy*, vol. 2, no. 5, pp. 81–85, 2004.
[104] A. Arcuri, M. Z. Iqbal, and L. Briand, "Black-box system testing of real-time embedded systems using random and search-based testing," in *IFIP International Conference on Testing Software and Systems*, 2010, pp. 95–110.
[105] P. Koopman, A. Alimarine, J. Tretmans, and R. Plasmeijer, "Gast: Generic automated software testing," in *Symposium on Implementation and Application of Functional Languages*, 2002, pp. 84–100.
[106] G. Candea, S. Bucur, and C. Zamfir, "Automated software testing as a service," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 155–160.
[107] A. Krstic, W.-C. Lai, K.-T. Cheng, L. Chen, and S. Dey, "Embedded software-based self-test for programmable core-based designs," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 18–27, 2002.

[108] S. H. Edwards, "A framework for practical, automated black-box testing of component-based software," *Software Testing, Verification and Reliability*, vol. 11, no. 2, pp. 97–111, 2001.
[109] B. J. LeSuer and K. A. Adams, *Automated software testing*. Google Patents, 2007.
[110] S. H. Edwards, "A framework for practical, automated black-box testing of component-based software," *Software Testing, Verification and Reliability*, vol. 11, no. 2, pp. 97–111, 2001.
[111] Brian, "Best Automation Testing Tools for 2018 (Top 10 reviews)," *Medium*, 26-Oct-2017.
[112] "autotest.pdf." .
[113] J. F. Smart, *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. O'Reilly Media, Inc., 2011.
[114] J. Pan, *18-849b Dependable Embedded Systems*. .
[115] K. Petersen, C. Wohlin, and D. Baca, "The waterfall model in large-scale development," in *International Conference on Product-Focused Software Process Improvement*, 2009, pp. 386–400.
[116] "Definition of Test," *Software Testing Fundamentals*, 19-Dec-2010. .
[117] J. Voas and J. Payne, "Dependability certification of software components," *Journal of Systems and Software*, vol. 52, no. 2–3, pp. 165–172, 2000.
[118] H. Pham, *System Software Reliability*. Springer Science & Business Media, 2007.
[119] "100+ Best Software Testing Tools - (Research Done for You!)," *QASymphony*, 18-Oct-2016. [Online]. Available: https://www.qasymphony.com/blog/100-plus-best-software-testing-tools/. [Accessed: 25-Apr-2018].
[120] "Dimensions of Software Quality," *Software Testing Fundamentals*, 03-Sep-2011. .
[121] E. Kit, *Software testing in the real world: improving the process*. Addison-wesley, 1995.
[122] M. Fewster, "Common mistakes in test automation," in *Proceedings of Fall Test Automation Conference*, 2001.
[123] A. Hartman, M. Katara, and A. Paradkar, "Domain specific approaches to software test automation," in *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers*, 2007, pp. 621–622.
[124] T. Koomen and M. Pol, *Test process improvement: a practical step-by-step guide to structured testing*. Addison-Wesley Longman Publishing Co., Inc., 1999.
[125] R. Fantina, *Practical software process improvement*. Artech House, Inc., 2005.
[126] R. Black, "Managing The Testing Process," 1999.
[127] R. Black, "Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing," 2011.
[128] P. Fitness, "Software Quality Metrics."
[129] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.
[130] S. C. Ntafos, "A comparison of some structural testing strategies," *IEEE transactions on software engineering*, vol. 14, no. 6, pp. 868–874, 1988.
[131] A. P. Mathur, "Performance, effectiveness, and reliability issues in software testing," in *Computer Software and Applications Conference, 1991. COMPSAC'91., Proceedings of the Fifteenth Annual International*, 1991, pp. 604–605.
[132] W. E. Perry, *Effective methods for software testing: Includes complete guidelines, Checklists, and Templates*. John Wiley & Sons, 2007.

[133] I. Jovanović, "Software testing methods and techniques," *The IPSI BgD Transactions on Internet Research*, vol. 30, 2006.

Σύντομο Βιογραφικό Συγγραφέα

Ονομάζομαι Μοσχοπούλου Ειρήνη, γεννήθηκα στην Πάτρα και κατάγομαι από την Κεφαλονιά. Σπούδασα στο τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής, στο Πολυτεχνείο Πατρών (2012-2018). Έκανα πρακτική άσκηση στην εταιρεία Dialog Semiconductor έχοντας τον τίτλο Quality Assurance Junior Test Engineer. Έχω εξειδικευθεί σε θέματα εξασφάλισης ποιότητας λογισμικού με επαγγελματική εμπειρία σε τεχνικές χειρωνακτικού και αυτοματοποιημένου ελέγχου λογισμικού. Είμαι εξοικειωμένη με διάφορες γλώσσες προγραμματισμού όπως Python, Javascript, HTML, Java και C. Μιλώ άπταιστα Αγγλικά.