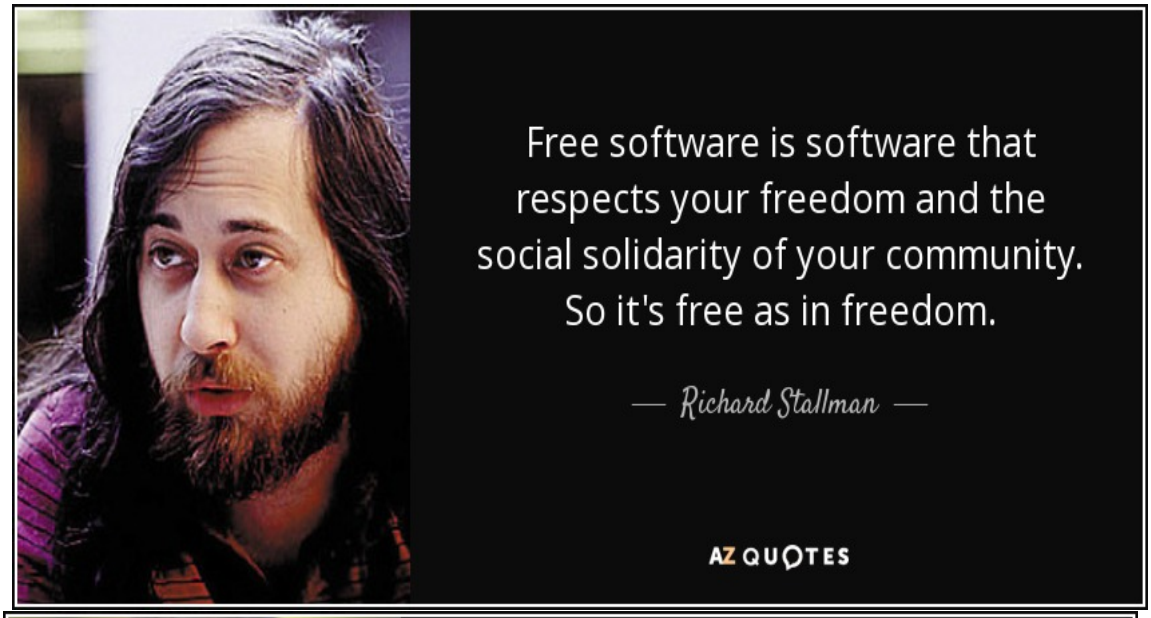
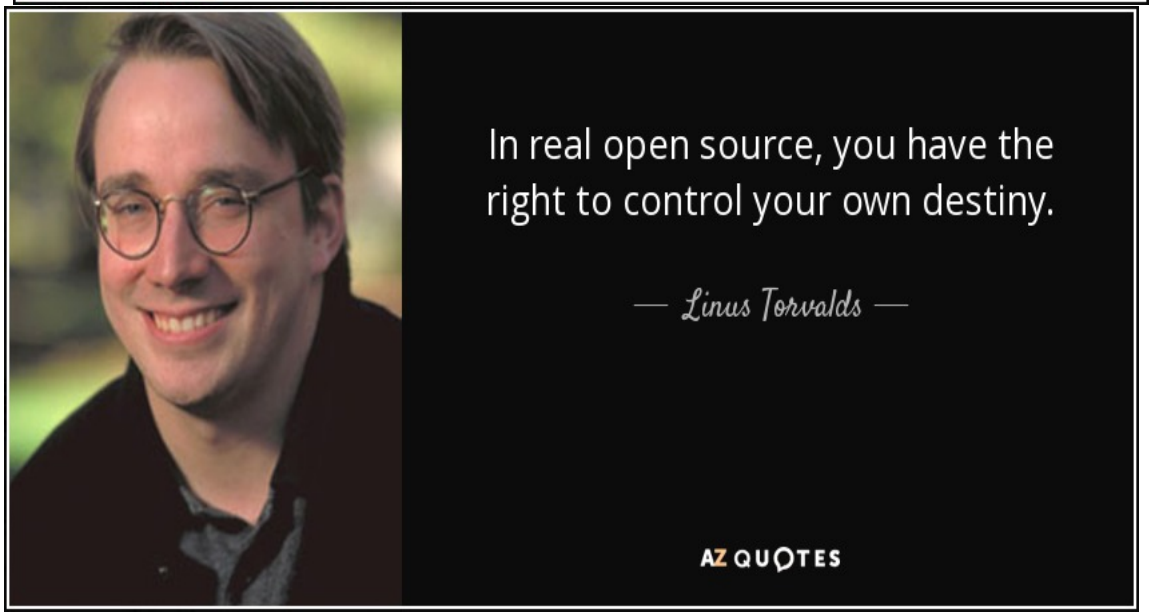
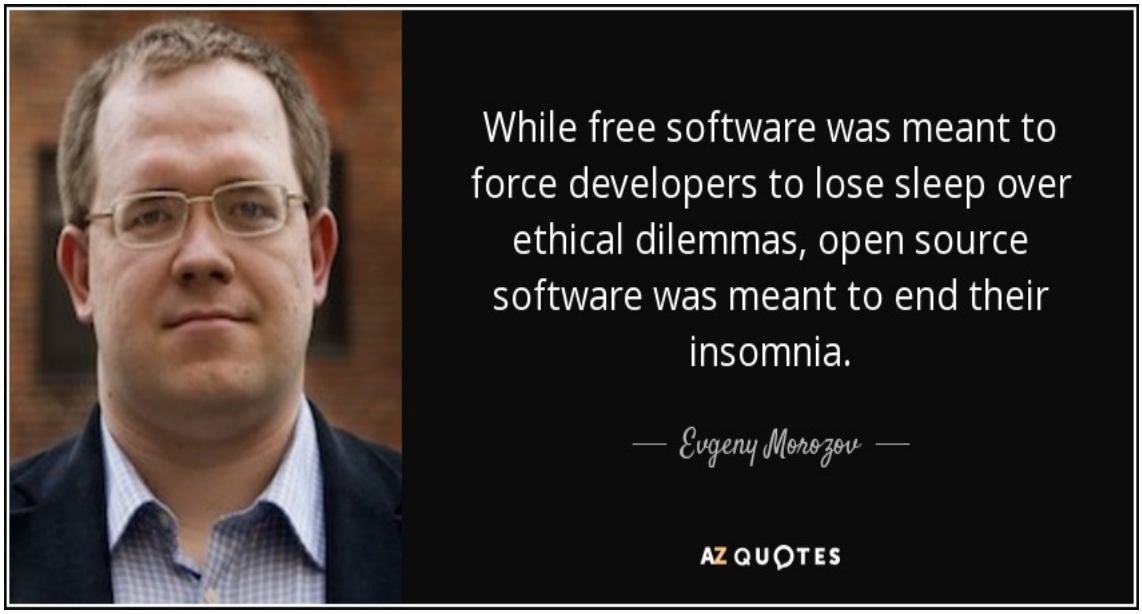

Free software is software that respects your freedom and the social solidarity of your community. So it's free as in freedom.
— Richard Stallman —
AZ QUOTES


In real open source, you have the right to control your own destiny.
— Linus Torvalds —
AZ QUOTES


While free software was meant to force developers to lose sleep over ethical dilemmas, open source software was meant to end their insomnia.
— Evgeny Morozov —
AZ QUOTES

OPEN SOURCE SOFTWARE QUALITY

Κωνσταντίνος Βασιλόπουλος•3180018○*git*
Πέτρος Χάνας•3170173○*git*

Μπορείτε να βρείτε τον πηγαίο μας κώδικα Latex στο Github

2022

Περιεχόμενα

1 Τι είναι το λογισμικό ανοιχτού κώδικα; 5

2 Ιστορική Αναδρομή 5
2.1 Τι είναι το Linux . . . 6
2.1.1 Ubuntu . . . 6
2.1.2 Debian . . . 7
2.2 GPL . . . 7
2.3 Mozilla Firefox . . . 7

3 Ποιότητα Λογισμικού 8
3.1 Ορισμός . . . 8
3.2 Τρόποι Ελέγχου Ποιότητας . . . 8
3.3 Εργαλεία Ελέγχου Ποιότητας . . . 9
3.3.1 Χρήσιμες Μετρικές Ποιότητας Ελέγχου . . . 9
3.3.2 Εκτίμηση Αριθμού Σφαλμάτων . . . 9
3.3.3 Μοντέλα Ποιότητας Ελέγχου . . . 11
3.3.4 Δείκτης Πραγματικής Συντηρησιμότητας(Maintainability Index) . . . 12
3.3.5 ARiSA Compendium Model . . . 13
3.3.6 SQALE MODEL . . . 15
3.4 Απειλές στην Εγκυρότητα Λογισμικού . . . 16
3.5 Σύγχρονοι Μέθοδοι Ανάπτυξης Ανοιχτού Λογισμικού . . . 16
3.5.1 Διαδικασία Ανάπτυξης Λογισμικού Ανοιχτού Κώδικα . . . 16
3.5.2 Χαρακτηριστικά Διαδικασίας Ανάπτυξης Λογισμικού Ανοιχτού Κώδικα . . . 17
3.6 Μοντέρνος Έλεγχος Ποιότητας . . . 20
3.6.1 Λογισμικό Ελέγχου Πηγαίου Κώδικα . . . 20
3.6.2 Αποφυγή Κακόβουλου Λογισμικού . . . 20

4 Ένα Αναλυτικό Παράδειγμα 21
4.1 Λόγοι και Αφορμές . . . 21
4.2 Αρχίζοντας το project ή μία συνεισφορά . . . 21
4.3 Συμβατικός Έλεγχος . . . 22

4.3.1 Έλεγχος μονάδας 22
4.4 Γενικότεροι Έλεγχοι 23
4.4.1 Έλεγχος Λειτουργίας 23
4.4.2 Έλεγχος Εκτέλεσης 23
4.4.3 Έλεγχος Αποδοχής 24
4.4.4 Έλεγχος Εγκατάστασης 24

5 Επίλογος 24

6 Βιβλιογραφία 24

Περίληψη

Σε αυτήν την εργασία θα αναφερθούμε στους τρόπους ανάπτυξης ελεύθερου λογισμικού και στα ενδεχόμενα προβλήματα τα οποία προκύπτουν ως προς τον προγραμματισμό και το debugging αυτού καθ'όλη τη διάρκεια ζωής του. Επίσης θα μελετήσουμε τους τρόπους αξιολόγησης του λογισμικού από τρίτους παράγοντες όπως εταιρείες, δημόσιοι οργανισμοί αλλά και από τους ίδιους τους χρήστες. Ουσιαστικα, η ποιότητα του ανοιχτού-ελεύθεροου λογισμικού συνίσταται από κάποιους βασικούς πυλώνες άμεσα εξαρτώμενος από το ίδιο το λογισμικό όπως η ταχύτητά του, η αξιοπιστία του, η ευκολία στην χρήση του αλλά και την περιήγησή του και πολλά άλλα. Εν ολίγοις, θα αναλύσουμε όλες τις πτυχές ανάπτυξης και αξιολόγησης της ποιότητας λογισμικού ανοιχτού κώδικα διανθίζοντας όλα τα επιμέρους στοιχεία που συγκροτούν αλλά και καθιστούν το λογισμικό έτοιμο προς διάθεση στο ευρύ κοινό.

1 Τι είναι το λογισμικό ανοιχτού κώδικα;

Συχνά θα ακούσουμε ανθρώπους της πληροφορικής να αναφέρονται σε συγκεκριμένα προγράμματα ως «ανοιχτού κώδικα» ή «ελεύθερου λογισμικού». Εάν ένα πρόγραμμα είναι ανοιχτού κώδικα, ο πηγαίος κώδικας του είναι ελεύθερος διαθέσιμος στους χρήστες του. Οι χρήστες του - και οποιοσδήποτε άλλος - έχουν τη δυνατότητα να λάβουν αυτόν τον πηγαίο κώδικα, να τον τροποποιήσουν και να διανείμουν τις δικές τους εκδόσεις του προγράμματος. Οι χρήστες έχουν επίσης τη δυνατότητα να διανέμουν όσα αντίγραφα του αρχικού προγράμματος θέλουν. Οποιοσδήποτε μπορεί να χρησιμοποιήσει το πρόγραμμα για οποιονδήποτε σκοπό. δεν υπάρχουν χρεώσεις αδειοδότησης ή άλλοι περιορισμοί στο λογισμικό. Για παράδειγμα, το Ubuntu Linux είναι ένα λειτουργικό σύστημα ανοιχτού κώδικα. Μπορείτε να κατεβάσετε το Ubuntu, να δημιουργήσετε όσα αντίγραφα θέλετε και να τα δώσετε στους φίλους σας. Μπορείτε να εγκαταστήσετε το Ubuntu σε απεριόριστο αριθμό υπολογιστών. Μπορείτε να δημιουργήσετε αντίγραφα του δίσκου εγκατάστασης του Ubuntu και να τα διανείμετε. Εάν είχατε ιδιαίτερα κίνητρα, θα μπορούσατε να κατεβάσετε τον πηγαίο κώδικα για ένα πρόγραμμα στο Ubuntu και να το τροποποιήσετε, δημιουργώντας τη δική σας προσαρμοσμένη έκδοση αυτού του προγράμματος - ή του ίδιου του Ubuntu. Όλες οι άδειες ανοιχτού κώδικα σάς επιτρέπουν να το κάνετε αυτό, ενώ οι άδειες κλειστού κώδικα επιβάλλουν περιορισμούς σε εσάς.

2 Ιστορική Αναδρομή

Κατά τη διάρκεια των πρώτων δεκαετιών της πληροφορικής, ήταν ο κανόνας και όχι η εξαίρεση η κοινή χρήση αναγνώσιμου πηγαίου κώδικα από τον άνθρωπο. Στις δεκαετίες του 1950 και του 1960 σχεδόν όλο το λογισμικό που παρήχθη από ακαδημαϊκούς και εταιρικά ερευνητικά εργαστήρια, όπως τα Bell Labs της ATT, εργάστηκαν σε συνεργασία. Όλοι είχαν μακροχρόνιες παραδόσεις ανοιχτού χαρακτήρα και συνεργασίας, άτυπες για τον ακαδημαϊκό χώρο, και ως εκ τούτου, ακόμη κι αν το λογισμικό δεν ήταν επίσημα διαθέσιμο στο κοινό, ο πηγαίος κώδικας τους ήταν ευρέως κοινός. Οι εταιρείες ηλεκτρονικών υπολογιστών διένειμαν επίσης τον πηγαίο κώδικα του λογισμικού που έστειλαν μαζί με το υλικό, για δύο λόγους. Πρώτον, δεν έβλεπαν το λογισμικό ως εμπόρευμα προς πώληση μέσω αδειοδότησης. Δεύτερον, οι χρήστες τροποποιούσαν συχνά οι ίδιοι το λογισμικό επειδή δεν θα λειτουργούσε σε διαφορετικό υλικό ή λειτουργικό σύστημα χωρίς κάποια παρέμβαση στον κορμό του, καθώς και για να διορθώσουν σφάλματα ή να προσθέσουν νέες λειτουργίες.

Το να έχουν διαθέσιμο τον πηγαίο κώδικα ήταν νευραλγικής σημασίας για τους κατασκευαστές, καθώς η δημιουργία διαφορετικών binary files (ή μεταγλωττισμένου κώδικα) για διαφορετικό υλικό δεν ήταν καθόλου πρακτική και συχνά δημιουργούσε μεγάλες επιπλοκές που οδηγοούσαν σε λογισμικό μή προσπελάσιμο λογισμικό, δλδ μη ικανό να περαιωθούν οι λειτουργίες του. Ορισμένα πανεπιστήμια είχαν ακόμη και μια πολιτική που απαιτούσε όλα τα λογισμικά που είναι

εγκατεστημένα στους υπολογιστές στα εργαστήριά τους να συνοδεύονται από δημοσιευμένους πηγαίους κώδικες.

Το 1953, το τμήμα UNIVAC του Remington Rand ανέπτυξε την πρώτη παρουσία ελεύθερου λογισμικού ανοιχτού κώδικα που ονομάζεται A-2 (Arithmetic Language v2 system), το οποίο κυκλοφόρησε στους πελάτες τους μαζί με τον πηγαίο κώδικα. Προσκλήθηκαν επίσης να στείλουν πίσω τις βελτιώσεις τους. Αργότερα, το πρώτο λειτουργικό σύστημα της IBM, ο κώδικας του IBM 704 διανεμήθηκε με όλους τους μεγάλους υπολογιστές τους. Οργανισμοί όπως η IBM, η DEC και η General Motors δημιουργούν ομάδες χρηστών για να διευκολύνουν την κοινή χρήση κώδικα μεταξύ των χρηστών, ακαδημαϊκών και άλλων παραγόντων του κλάδου.

Με την πάροδο των δεκαετιών και την συνεχώς αναπτυσσόμενη βιομηχανία της τεχνολογίας ποικίλοι παράγοντες απαιτούσαν την διανομή, σε μεγαλύτερο βεληνεκές, ανοιχτού κώδικα λογισμικού για την κάλυψη των ήδη υπαρχουσών αναγκών. Ο ανοιχτός κώδικας απεδείχθη ως το κύριο πλεονέκτημα μικρών ομάδων προγραμματιστών για την ανάπτυξη μεγάλων και ευρέου βεληνεκούς, τρόπον τινά, εφαρμογών και λογισμικού. Χαρακτηριστικότερο ίσως παράδειγμα αυτής της αυτοδυναμίας ανάπτυξης αποτελεί ο Linus Torvalds που στις αρχές της δεκαετίας του 1990 κατάφερε να προγραμματίσει έναν καινούριο kernel βασισμένο εκ των προτέρων αλλά όχι εξόλοκλήρου στο λειτουργικό UNIX. Έτσι, η κοινότητα του ελεύθερου λογισμικού έλαβε το πρώτο πλήρες δωρεάν λειτουργικό σύστημα με τον πυρήνα του Linus Torvalds σε συνδυασμό με το λειτουργικό σύστημα GNU. Το Debian, που ιδρύθηκε από τον Ian Murdock το 1993, δεσμεύτηκε στις αρχές GNU και FSF του ελεύθερου λογισμικού. Μεταγενέστερα παραδείγματα ελεύθερου λογισμικού αποτελούν οι διανομές ή distros του Linux και πληθώρα πακέτων και προγραμμάτων διαθέσιμα ενδολειτουργικά όπως π.χ. το inkcape, το gcalculator, τα KDE πακέτα και πολλά άλλα.

2.1 Τι είναι το **Linux**

Το Linux είναι μια από τις δημοφιλείς εκδόσεις του λειτουργικού συστήματος UNIX. Είναι ανοιχτού κώδικα καθώς ο πηγαίος κώδικας του είναι δωρεάν διαθέσιμος. Το Linux σχεδιάστηκε λαμβάνοντας υπόψη τη συμβατότητα UNIX. Η λίστα λειτουργιών του είναι αρκετά παρόμοια με αυτή του UNIX. Ακριβώς όπως τα Windows, iOS και Mac OS, το Linux είναι μια από τις πιο δημοφιλείς πλατφόρμες στον πλανήτη. Το Android, τροφοδοτείται από το λειτουργικό σύστημα Linux και σχεδόν καθόλολκληρίαν όλα τα λειτουργικά υπεύθυνα για την διαχείριση και την λειτουργία servers δουλεύουν πάνω σε Linux.

Η υιοθέτηση του Linux στην μαζική παραγωγή, αντί της πρότερης χρήσης αποκλειστικά και μόνο από ερασιτέχνες, άρχισε να απογειώνεται για πρώτη φορά στα μέσα της δεκαετίας του 1990 στην κοινότητα των υπερυπολογιστών, όπου οργανισμοί όπως η NASA άρχισαν να αντικαθιστούν τις ολοένα και πιο ακριβές μηχανές τους με ομάδες φθηνών εμπορευματικών υπολογιστών με Linux. Η εμπορική χρήση ξεκίνησε όταν η Dell και η IBM, ακολουθούμενη από τη Hewlett-Packard, άρχισαν να προσφέρουν υποστήριξη Linux για να ξεφύγουν από το μονοπώλιο της Microsoft στην αγορά λειτουργικών συστημάτων επιτραπέζιων υπολογιστών.

Σήμερα, τα συστήματα Linux χρησιμοποιούνται σε όλους τους υπολογιστές, από τα ενσωματωμένα συστήματα έως σχεδόν όλους τους υπερυπολογιστές, και έχουν εξασφαλίσει μια θέση σε εγκαταστάσεις διακομιστή όπως η δημοφιλής στοίβα εφαρμογών LAMP. Η χρήση των διανομών Linux σε οικιακούς και εταιρικούς επιτραπέζιους υπολογιστές έχει αυξηθεί. Οι διανομές Linux έχουν γίνει επίσης δημοφιλείς στην αγορά νετβοοκ, με πολλές συσκευές να αποστέλλονται με προσαρμοσμένες διανομές Linux εγκατεστημένες και η Google να κυκλοφορεί το δικό της Chrome OS σχεδιασμένο για netbook.

2.1.1 **Ubuntu**

Ίσως η διασημότερη διανομή (distro) Linux μέχρι σήμερα με χιλιάδες εφαρμογές σε desktop, server και ακόμη και mobile περιβάλλοντα. Ονομάστηκε έτσι από την διάλεκτο των Xhosa των Nguni, μίας φυλής στην Νότιο Αφρική. Κυριολεκτικά, σημαίνει ανθρωπότητα και θέλει να επιδείξει τον

καθ'θπερβολή τον 'πανανθρώπινο' χαρακτήρα και φιλοσοφία του λειτουργικού που είναι ελεύθερο και δωρεάν από όλους προς όλους. Έχοντας αυτό υπ'όψιν, το Ubuntu δεν αναπτύσσεται ούτε συντηρείται από την κοινότητα των χρηστών αλλά από την εταιρεία Canonical που αποτελούν και τους ιδρυτές του. Δεν μπορεί όμως να θεωρηθεί καθόλοκληρίαν proprietary software καθώς ο Kernel εξακολουθεί να είναι ο Linux. Δίχως αμφιβολία το Ubuntu είναι το διασημότερο distro χάρη στην ευκολία χρήσης του, την απλότητα της γραφικής του διεπαφής και τον άμεσο παραλληλισμό του με την αντίπερα όχθη των πλέον συνηθισμένων Windows.

2.1.2 **Debian**

Το Debian αναπτύσσεται από εθελοντές από όλο τον κόσμο. Δεν είναι ένα proprietary λογισμικό, το οποίο υποστηρίζεται από εταιρείες όπως πολλές άλλες διανομές Linux. Εν τοις πράγμασι, κανένας δεν μπορεί να ισχυριστεί πως του άνήκει' το Debian καθώς υπεύθυνη για την συντήρηση του και τις εκάστοτε αναβαμίσεις του είναι η κοινότητα χρηστών του. Αυτή η ουσιώδης διαφορά του από τα Ubuntu το καθιστούν ως πυλώνα της κίνησης του Ελεύθερου Λογισμικού και ως ένα από τα διασημότερα και πιο εμπορικά Linux Distros. Το Debian ξεκίνησε τον Αύγουστο του 1993 από τον Ian Murdock, τότε προπτυχιακό στο Πανεπιστήμιο Purdue όπου και χρηματοδοτήθηκε από το έργο GNU του Ιδρύματος Ελεύθερου Λογισμικού, του οργανισμού που ξεκίνησε ως ιδέα του Richard Stallman και σχετίζεται με τη Γενική Δημόσια Άδεια (GPL).

2.2 **GPL**

Η άδεια GPL αναφέρεται στη Γενική Δημόσια Άδεια του GNU που χρησιμοποιείται ευρέως για λογισμικό ανοιχτού κώδικα, που γράφτηκε για πρώτη φορά από τον Richard Stallman το 1989. Οι προγραμματιστές και οι οργανισμοί τη χρησιμοποιούν για να εμποδίσουν το λογισμικό να γίνει ιδιόκτητο. Ενοποίησε παρόμοιες άδειες που ήταν διαθέσιμες εκείνη την εποχή, αλλά κατά κύριο λόγο αποτελεί υλοποίηση της φιλοσοφίας του Ιδρύματος Ελεύθερου Λογισμικού (FSF) και της αντίληψης του Stallman για το copyleft για την ανάπτυξη και διανομή λογισμικού.

Με άδεια GPL (ή απλώς GPL), ένας συγκεκριμένος χρήστης μπορεί ελεύθερα να χρησιμοποιεί, να τροποποιεί ή να αναδιανέμει λογισμικό χωρίς περιορισμούς. Ένα δημοφιλές παράδειγμα λογισμικού που χρησιμοποιεί GPL είναι το WordPress, που σημαίνει ότι ο καθένας μπορεί να χρησιμοποιήσει, να τροποποιήσει ή να επεκτείνει τον πηγαίο κώδικα όπως επιθυμεί.

Η άδεια δίνει στους χρήστες την δικαιδοσία να εκτελέσουν τα κάτωθι:

- Μπορούν να κατεβάσουν και να εκτελέσουν το λογισμικό ελεύθερα
- Μπορούν να αλλάξουν τον πηγαίο κώδικα του λογισμικού
- Μπορούν να αναδιανείμουν αντίγραφα του λογισμικού αυτούσια ή ακόμα και να τα τροποποιήσουν και να τα διανείμουν εξυπηρετώντας καλύτερα μία ανάγκη του ενδεχομένως να έχει προκύψει

Η ουσία αυτής της άδειας, και όλων των παρεμφερών, είναι να διασφαλίζουν κατά το μέγιστο δυνατό την ελεύθερη ανάπτυξη, συντήρηση και διανομή του λογισμικού ανοιχτού κώδικα.

2.3 **Mozilla Firefox**

Άλλο ένα σπουδαίο παράδειγμα λογισμικού ανοιχτού κώδικα είναι το πρόγραμμα περιήγησης διαδικτύου Mozilla Firefox. Ο Firefox είναι ένα δωρεάν πρόγραμμα περιήγησης ιστού που κυκλοφόρησε για πρώτη φορά σε έκδοση βετα στις 23 Σεπτεμβρίου 2002, ως ¨Πρόγραμμα περιήγησης Mozilla, αν και εσωτερικά είχε την κωδική ονομασία "Phoenix". Ο Firefox 1.0 κυκλοφόρησε επίσημα στις 9 Νοεμβρίου 2004. Ο σκελετός του φυλλομετρητή είναι ανοιχτού κώδικα και προσβάσιμος στο ευρύ κοινό.

Ο Firefox έγινε δημοφιλής εναλλακτική του Microsoft Internet Explorer 6.0 όταν οι χρήστες αναζήτησαν ένα πρόγραμμα περιήγησης που θα μπορούσε να τους προστατεύει καλύτερα από λογισμικό υποκλοπής spyware και κακόβουλους ιστοτόπους. Από το 2017, είναι το τέταρτο πιο δημοφιλές πρόγραμμα περιήγησης μετά το Google Chrome, το Apple Safari και το UC Browser.

Το Mozila χρησιμοποιεί την Gecko Rendering Engine για να προβάλλει τους ιστοτόπους δημιουργώντας έτσι Real-time απεικόνιση των ιστοσελιδών χωρίς σχεδόν καθόλο intermediate lag. Το Gecko είναι επίσης μια μηχανή περιήγησης που αναπτύχθηκε από την Mozilla που συγκροτεί το Firefox, το mail client Thunderbird και πολλά άλλα components. Όπως σχεδόν όλα τα projects της Mozilla έτσι και το Gegko είναι και αυτό open-source.

3 Ποιότητα Λογισμικού

3.1 Ορισμός

Η ποιότητα λογισμικού είναι μια αφηρημένη έννοια που γίνεται αντιληπτή και ερμηνευόμενη διαφορετικά με βάση τις προσωπικές απόψεις και τα ενδιαφέροντά του. Για να λυθεί αυτή η ασάφεια, το ISO/IEC9126 (Διεθνής Οργανισμός Τυποποίησης 2001) παρέχει ένα πλαίσιο για την αξιολόγηση της ποιότητας του λογισμικού. Ορίζει έξι χαρακτηριστικά ποιότητας λογισμικού, συχνά αναφέρονται ως ποιοτικά χαρακτηριστικά:

- **Λειτουργικότητα**: Εάν το λογισμικό εκτελεί τις απαραίτητες λειτουργίες.
- **Αξιοπιστία**: Η ικανότητα του λογισμικού να παράγει επανειλημένα έγκυρα αποτελέσματα άνευ προβλημάτων.
- **Χρηστικότητα**: Ο βαθμός ευκολίας του λογισμικού από το μέσο χρήστη.
- **Αποτελεσματικό**: Κατά πόσο το επιθυμητό αποτέλεσμα επιτυγχάνεται μετά το πέρας της λειτουργίας του λογισμικού.
- **Συντηρησιμότητα**: Η ευκολία με την οποία αναβαθμίζεται το λογισμικό στα πιο σύγχρονα πρότυπα.
- **Φορητότητα**: Η δυνατότητα του λογισμικού να λειτουργεί σε πολλά και διαφορετικά λειτοργικά συστήματα.

3.2 Τρόποι Ελέγχου Ποιότητας

Ο έλεγχος του λογισμικού περιστρέφεται πρακτικά γύρω από 2 θεμελιώδεις έννοιες-της επαλήθευσης και της επικύρωσης. Έτσι, γίνεται βέβαιο ότι το λογισμικό συναντά τις ανάγκες του εκάστοτε χρήστη δίχως παρεκκλίσεις από τις απαιτήσεις του. Οι 2 βασικές ερβτήσεις που τίθενται στον developer είναι:

- Αναπτύσσω το σωστό προϊόν;(Επαλήθευση)
- Αναπτύσσω σωστά/ορθά το προϊόν;(Επικύρωση)

Η ουσία των 2 αυτών παραγώγων συναρτάται άμεσα με την ποιότητα της ανάπτυξης του λογισμικού και όχι μόνο. Αυτό συμβαίνει, διότι χρησιμοποιώντας αυτούς του 2 πυλώνες βεβαιώνω πως ο τρόπος ανάπτυξης του λογισμικού βρίσκεται σε παραλληλία με τις απαιτήσεις του χρήστη. Σε ιδεατό επίπεδο, οι μέθοδοι ελέγχου ποιότητας παράγουν ένα μαθηματικό μοντέλο που μας δίνει την ΄ποιότητα΄ του λογισμικού που παρήχθη πάντα συναρτόμενη με τις απαιτήσεις του χρήστη/πελάτη και με τους διαθέσιμους πόρους. Ενίοτε, αυτό το πόρισμα δεν είναι πάντα τόσο ακριβές και σαφές και εν προκειμένω δίνεται μία πιο γενική παράθεση του προβλήματος όχι σε τόσο φορμαλιστικά πλαίσια.

Επι παραδείγματι, θα ασχοληθούμε με την έννοια του κόστους στον έλεγχο της ποιότητας του λογισμικού. Η έννοια αυτή αφορά ποικίλους τομείς της διαδικασίας ανάπτυξης του λογισμικού που αναλύονται στους επιμέρους παράγοντες:

- Κόστος Πρόληψης
- Κόστος Επιθεώρησης
- Κόστος Αξιολόγησης
- Κόστος Βλαβών

Αυτές οι 4 βασικές κατηγορίες κόστους συνιστούν σαν σύνολο την έννοια του ΄κόστους΄ ανάπτυξης λογισμικού. Ο λόγος για τον οποίο αναφέρεται στον τρόπο ελέγχου ποιότητας δεν είναι παρα για να τεθεί η μεταβλητή ΄κόστος΄ στην συνάρτηση ΄ποιότητα΄. Είναι προφανές, πως όσο και αν θέλω να αυξήσω την ποιότητα του λογισμικού μου έαν τίθεται περιορισμός στο κόστος ανάπτυξης τότε η προαναφερθείσα ποιότητα θα λάβει ένα άνω όριο συναρτήσει του διαθέσιμου κεφαλαίου. Αξίζει να σταθούμε για λίγο στον τομέα των λαθων.

3.3 Εργαλεία Ελέγχου Ποιότητας

Σε αυτόν τον τομέα θα αναφερθούμε στα αυτοματοποιημένα εργαλεία που μας παράσχουν metrics και ενδελεχείς αναλύσεις περί της γενικότερης ΄ποιότητας΄ αλλά και τους επιμέρους παράγοντες που την συγκροτούν.

Μπορούμε να πλαισιώσουμε την γενικότερη έννοια της ποιότητας του λογισμικού ρωτώντας κάποιες απλές αλλά νευραλγικής σημασίας ερωτήσεις που ενοποιούν το πεδίο ποιότητα και το καθιστούν εύκολα κατηγοριοποιήσιμο.

3.3.1 Χρήσιμες Μετρικές Ποιότητας Ελέγχου

Αξιοπιστία λογισμικού (software reliability) είναι η πιθανότητα το πρόγραμμα να λειτουργεί ικανοποιητικά για μία δεδομένη χρονική περίοδο.

$$R = \frac{MXMB}{1 + MXMB}$$

Όπου $MQMB$ είναι ο μέσος χρόνος μεταξύ βλαβών.

Διαθεσιμότητα λογισμικού (software availability) είναι η πιθανότητα ένα πρόγραμμα να λειτουργεί επιτυχώς (ικανοποιητικά) σύμφωνα με τις απαιτήσεις λογισμικού σε μια δεδομένη χρονική στιγμή

$$A = \frac{MXMB}{MXMB + MQES}$$

Όπου $MQES$ μέσος χρόνος επιδιόρθωσης σφάλματος και MXMB μέσος χρόνος μεταξύ βλαβών.

3.3.2 Εκτίμηση Αριθμού Σφαλμάτων

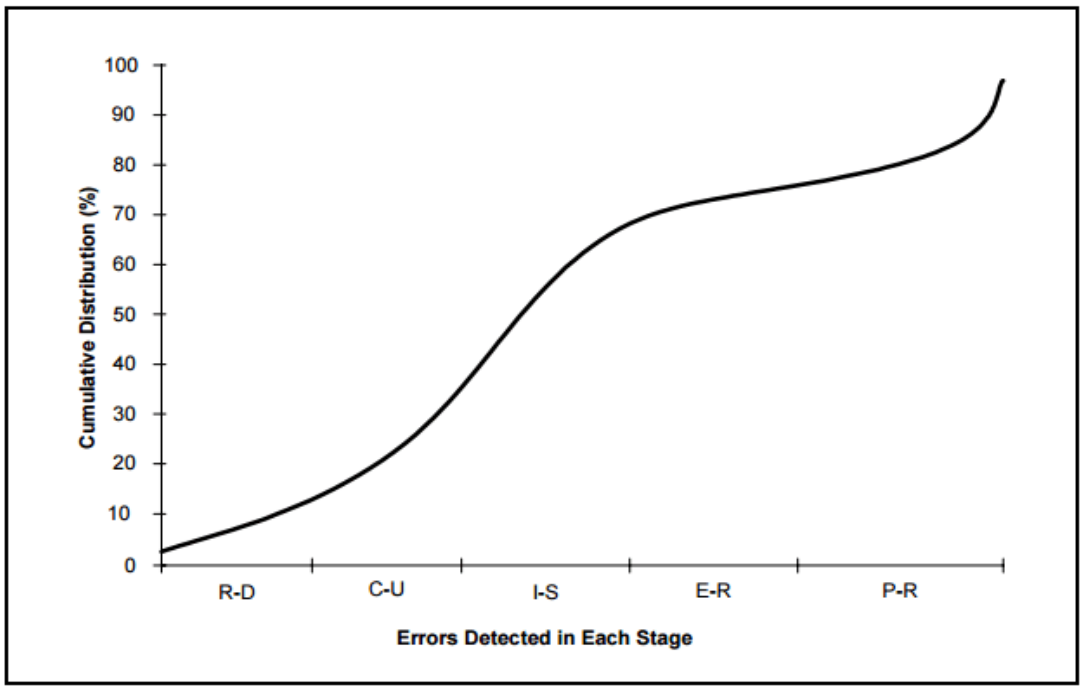
Εάν σε κάποιο λογισμικού διασπείρουμε εσκεμμένα Σ σφάλματα τότε μπορούμε να προβούμε σε εκτίμηση των πραγματικών σφαλμάτων παρατηρώντας το ποσοστό επιτυχίας στην επισήμανση των διεσπαρμένων σφαλμάτων.

$$N = \frac{Sn}{s}$$

Όπου:
-S σύνολο διεσπαρμένων σφαλμάτων
-n επισημανθέντα μη-διεσπαρμένα σφάλματα
-s επισημανθέντα διεσπαρμένα σφάλματα
-N σύνολο πραγματικών μη-διεσπαρμένων σφαλμάτων

Γράφημα συσχέτισης μεταξύ υπάρξης σφαλμάτων και του πλήθους των ήδη εντοπισμένων

Figure 5-2. Typical Cumulative Distribution of Error Detection



Legend:
R-D: Requirements Gathering and Analysis/Architectural Design
C-U: Coding/Unit Test
I-S: Integration and Component/RAISE System Test
E-R: Early Customer Feedback/Beta Test Programs
P-R: Post-product Release

Οι μετρικές που αναγράφονται εξηγούνται παρακάτω

- Εκτίμηση αριθμού σφαλμάτων με χρήση δύο διαφορετικών ομάδων ελέγχου.
- Η αποτελεσματικότητα κάθε ομάδας ελέγχου μπορεί να μετρηθεί, υπολογίζοντας το κλάσμα των σφαλμάτων που βρέθηκαν από κάθε ομάδα.
- Η αποτελεσματικότητα E(1) της ομάδας 1 μπορεί να εκφραστεί ως: $E(1) = x/n$
- Η αποτελεσματικότητα της ομάδας 2 ως $E(2) = y/n$
- $n = q/(E(1)E(2))$, όπου q ο αριθμός των κοινών σφαλμάτων που βρήκαν οι ομάδες 1 και 2.

Έτσι, καταλήγουμε στην έννοια της εμπιστοσύνης του λογισμικού όπου φορμαλιστικά είναι εν γένει η πιθανότητα να μην υπάρχει σφάλμα στο λογισμικό και δίνεται από τον παρακάτω τύπο

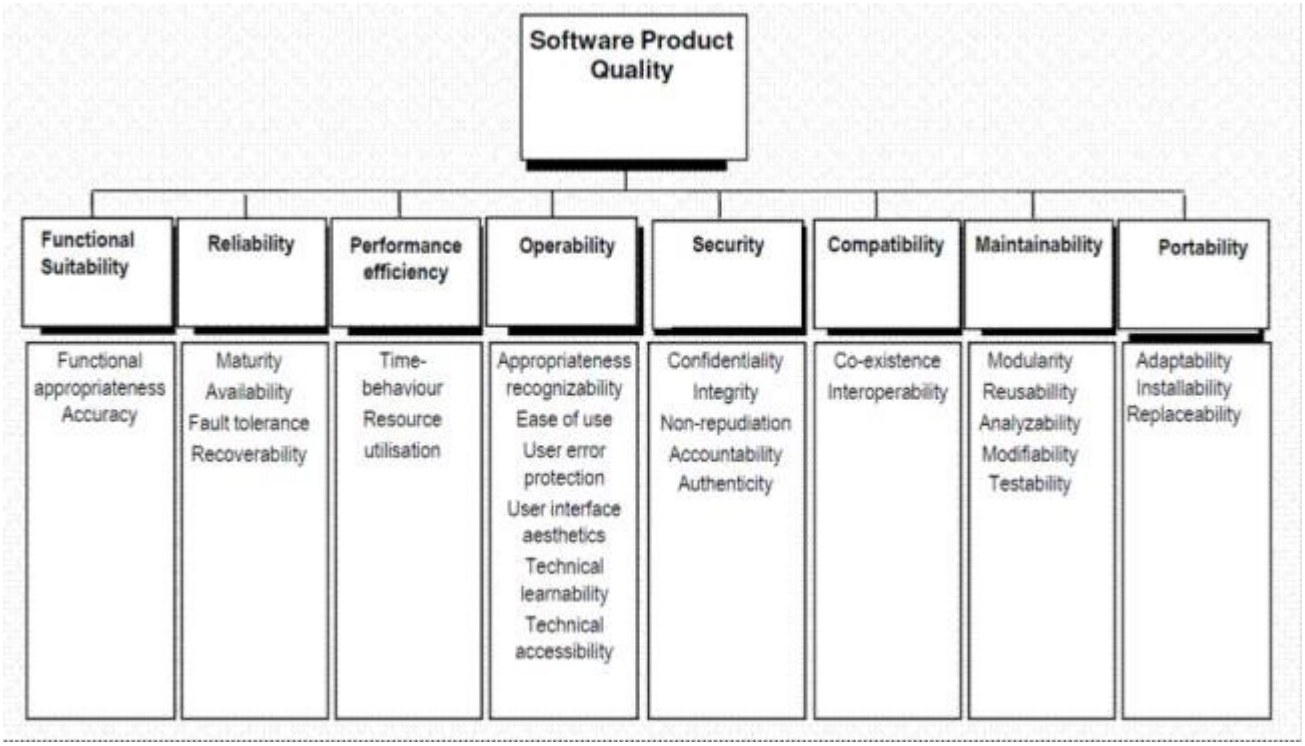
$$C_n = \begin{cases} 1 & \text{αν } n \geq N \\ \frac{S}{S-N+1} & \text{αν } n \leq N \end{cases}$$

Ο παραπάνω τύπος εκφράζει την διασπρά σφαλμάτων και αναλύεται ως εξής:

1. S αριθμός διεσπαρμένων σφαλμάτων
2. N αριθμός πραγματικών σφαλμάτων
3. n ο αριθμός των πραγματικών σφαλμάτων που επισημάνθηκαν

3.3.3 Μοντέλα Ποιότητας Ελέγχου

Η σημασία της ποιότητας του λογισμικού εξακολουθεί να αποτελεί βασικό ενδιαφέρον για τις ακαδημαϊκές και βιομηχανικές κοινότητες μετά από περισσότερα από 50 χρόνια έρευνας. Επιπλέον, όπως ο αριθμός των πολύπλοκων δικτυωμένων συστημάτων και οι υποδομές ζωτικής σημασίας που στηρίζονται σε αυτές αυξάνονται, αναμένεται να παραμείνει θέμα συνεχούς ενδιαφέροντος για έρευνα και συνεχή ανάπτυξη. Προηγούμενη έρευνα για την ποιότητα του λογισμικού είχε ως αποτέλεσμα μεγάλο αριθμό ποιότητας μοντέλα. Τα περισσότερα από αυτά περιγράφουν ένα σύνολο βασικών ιδιοτήτων που προσπαθούν να χαρακτηρίσουν τις πολλαπλές πτυχές ενός συστήματος λογισμικού από την πλευρά του πελάτη και από την πλευρά του προγραμματιστή αμφώτερα.



Παραπάνω παρουσιάζεται το ιεραρχικό μοντέλο: ISO25010

Η σημασία της ποιότητας του λογισμικού εξακολουθεί να αποτελεί βασικό ενδιαφέρον και για τα δύο τις ακαδημαϊκές και βιομηχανικές κοινότητες μετά από περισσότερα από 50 χρόνια έρευνας και πρακτική. Επιπλέον, όπως ο αριθμός των πολύπλοκων δικτυωμένων συστημάτων και οι υποδομές ζωτικής σημασίας που στηρίζονται σε αυτές αυξάνονται, αναμένεται να παραμείνει α θέμα συνεχούς ενδιαφέροντος για έρευνα, ανάπτυξη και συντήρηση λογισμικού. Προηγούμενη έρευνα για την ποιότητα του λογισμικού είχε ως αποτέλεσμα μεγάλο αριθμό ποιότητας μοντέλα. Τα περισσότερα από αυτά περιγράφουν ένα σύνολο βασικών ιδιοτήτων που προσπαθούν να το κάνουν χαρακτηρίζουν τις πολλαπλές πτυχές ενός συστήματος λογισμικού από ένα εσωτερικό

(προγραμματιστή- προσανατολισμένη), εξωτερική (πελατοκεντρική) ή και των δύο προοπτικών. Η εισαγωγή του πρώτου μοντέλου ποιότητας λογισμικού αποδίδεται στον McCall ιν 1976, ακολουθούμενο από το μοντέλο Dromey που το βελτίωσε. Αργότερα, οι συνεισφορές έγιναν μέρος του προτύπου ISO 9126, το οποίο εξέφραζε λογισμικό ποιότητας χρησιμοποιώντας ένα ιεραρχικό μοντέλο 5 τομέων ελέγχου που αποτελούνται από επιμέρους χαρακτηριστικά. Το μοντέλο ISO 25010 που απεικονίζεται στο άνωθι σχήμα αντιπροσωπεύει την τρέχουσα έκδοση και θεωρεί τη δυνατότητα συντήρησης ως το σύνολο 5 βασικών πυλώνων τροποποίησης:

1. Αρθρωτότητα
2. Επαναχρησιμοποίηση
3. Αναλυσιμότητα
4. Τροποποίηση
5. Δοκιμαστικότητα

Η ουσία αυτών των χαρακτηριστικών δεν είναι απαραίτητα να βελτιώνουν τον κώδικα ενός προγράμματος ή μια δομής προγραμμάτων (λογισμικού) αλλά να παράσχουν σχολαστικές λεπτομέρειες σχετικά με την δυνατότητα της περαιτέρω ανάπτυξης αυτού.

Τα σημαντικά αποτελούμενα κομμάτια ενός λογισμικού εν γένει ονομάζονται metrics. Βασικές metrics όπως γραμμές κώδικα, αριθμός συναρτήσεων ή αριθμός πακέτων και κλάσεων οι ενότητες έχουν χρησιμοποιηθεί ευρέως και με τη σειρά τους, αντικαταστάθηκαν από την εισαγωγή του αντικειμενοστραφούς μοντέλου και του σχετικού συνόλου μετρήσεων. Στις μέρες μας βρίσκουμε ένα πλήθος αντικειμενοστρεφών metrics όπως π.χ. αριθμός πακέτων και κλάσεων που ορίζονται και χρησιμοποιούνται για την ανίχνευση κομματιών κώδικα(προβληματικού), ελαττωματικού σχεδιασμού ή για τη βελτίωση της συντηρησιμότητας. Αυτές οι μετρήσεις είναι επίσης που απαιτούνται από τους ερευνητές για την αξιολόγηση της ποιότητας του λογισμικού σε ευρύτερο πλαίσιο. Ωστόσο, αυτό το μοντέλο φτάνει πια στην λήξη της λειτουργίας του στην αγορά και αντικαθίσταται πια από νέα που δίνουν έμφαση σε ποικίλους άλλους παράγοντες όπως η ταχύτητα του λογισμικού και η συμβασιμότητα αυτού με εργαλεία τεχνητης νοημοσύνης (AI) για την πιο υψηλού επιπέδου περάτωση και επίλυση ενός προβλήματος.

3.3.4 Δείκτης Πραγματικής Συντηρησιμότητας(**Maintainability Index**)

Είχαν γίνει σημαντικές προσπάθειες για τη σωστή εκτίμηση των απαιτούμενων πόρων για τη συντήρηση συστημάτων λογισμικού. Ορισμένος από τις αρχές της δεκαετίας του ′70, η υποθετική φόρμουλα για τον Δείκτη Συντηρησιμότητας (MI) ολοκληρώθηκε το 1992. Ο τύπος λαμβάνει υπόψη το μέγεθος του πηγαίου κώδικα, μετρούμενο συναρτήσει των παραλλαγών των γραμμών κώδικα και δύο παραμέτρους πολυπλοκότητας που εκφράζονται σε όρους του αρθρωτού παραδείγματος. Το αρθρωτό παράδειγμα είναι ο αριθμός των λειτουργιών και των χρηστών, γνωστός και ως όγκος Halstead που εκφράζει αριθμό των πιθανών διαδρομών και εναλλακτικών εκτέλεσης που δημιουργούνται από υπάρχουσες εντολές συνθηκών και βρόχους. Ο τύπος εκφράζεται ώς:

$$MI = 1715.2ln(aveV)0.23aveG16.2ln(aveSTAT)$$

όπου: V απεικονίζει τον όγκο του Halstead
$aveG$ ο αριθμός των πιθανών μονοπατιών εκτέλεσης
$aveSTAT$ ο μέσος όρος των δηλώσεων μεταβλητών

3.3.5 **ARiSA Compendium Model**

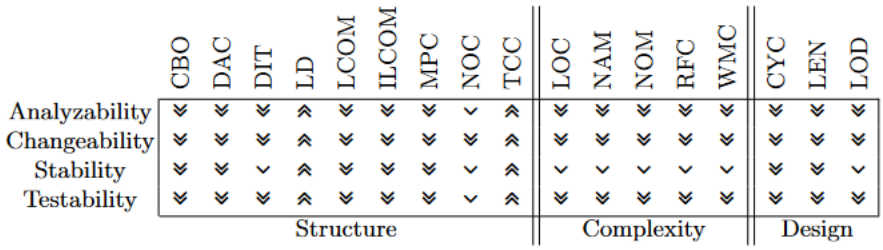
Η Σύνοψη Προτύπων Ποιότητας Λογισμικού και Μετρικών δημιουργήθηκε από ARiSA και ερευνητές από το Πανεπιστήμιο Linnaeus. Αποσκοπεί στη μελέτη της εκ νέου σχέση μεταξύ χαρακτηριστικών ποιότητας λογισμικού και μετρικών τιμών λογισμικού. Η Compendium μοντελοποιεί την ποιότητα λογισμικού σύμφωνα με το ISO 9126, μια παλαιότερη έκδοση στην οικογένεια ISO για πρότυπα ποιότητας λογισμικού. Όπως και το προηγούμενο έτσι και το εν λόγω μοντέλο αποτελείται από κάποια χαρακτηριστικά-πυλώνες που αναλύονται σε κάποια βασικά επιμέρους. Τα βασικά χαρακτηριστικά της Arisa είναι τα κάτωθι:

- Αναλυσιμότητα
- Μεταβλητότητα
- Συμμόρφωση,
- Σταθερότητα
- Δοκιμαστικότητα

Συχνά, παρατηρούμε πως ειδικα στο λογισμικό του ανοιχτού κωδικα παρατηρούνται στο στάδιο της δόμησης κώδικα προτού καταλήξει αυτός στο αποθετήριο ενός συστήματος ελέγχου υπάρχουν σοβαρά σφάλματα στην δομή του κωδικα πράγμα που τον καθιστά πιο δυσανάγνωστο και μερικές φορές ακόμα πιο πολύπλοκο χρονικά και χωρικά αντίστοιχα. Χαρακτηριστικό ήταν το παράδειγμα που σε μια μεγάλη αναβάθμιση του Debian στα πολύ πρώιμα βήματά του στα μέσα της δεκαετίας του ′90 το περιβάλλον Desktop που τότε χρησιμοποιούσε Xfce κολλούσε συνεχώς και παρουσίαζε σημαντικά προβλήματα στην διεπαφή του με το σύτημα αρχειοθέτησης του Kernel. Το πρόβλημα αρκετά αργότερα αποδείχθηκε ότι δεν ήταν σφάλμα του κωδικα συντακτικό ή λογικό αλλά υπέρογκο κομμάτι κώδικα που δεν μπορούσαν να διαχειριστούν πιο αδύναμοι υπολογιστές έτσι καθιστώντας το λειτουργικό σύστημα ακατάλληλο για τον σκοπό που είχε δημιουργηθεί. Έτσι, με ένα απλό μοντέλο εγκυρότητας και επαληθευσιμότητας ο κώδικας θα μπορούσε να έχει γίνει debloated από αχρείαστα metrics και συνθήκες που αποτελούσαν βάρος πολυπλοκότητας στην εκτέλεση του προγράμματος.

Για κάθε μετρική επιρροή, η compendium περιγράφει λεπτομερώς την κατεύθυνση και την ισχύ του ως προς την ροή του κώδικα. Αυτά τα στατιστικά ονομάζονται chevrons και χρησιμοποιούνται να διευκρινίσουν εάν οι αυξημένες τιμές στην πολυπλοκότητα της περαίωσης για τη δεδομένη μέτρηση οδηγούν σε βελτίωση ή υποβάθμιση της συντηρησιμότητας. Ο αριθμός των chevron αντιπροσωπεύει τη δύναμη αυτής της συσχέτισης, με δύο chevrons να αντιπροσωπεύουν μια ισχυρότερη σχέση όσον αφορά την συντηρησιμότητα που στην περίπτωση του λογισμικού ανοιχτού κώδικα είναι ιδιαίτερα σημαντική για κάθε επόμενο update. Πολλές φορές σε αποθετήρια ανοιχτού κώδικα υπάρχει ο κανόνας να εμφανίζεται η δομή του κώδικα αλλά και snippets που προσδιορίζουν benchmarks-κλειδιά αλλά και ο τρόπος που αυτός ο κώδικας συντηρείται και αναπτύσσεται με την συνοδεία πάντα σχολίων και οδηγιών εξωτερικά.

Παρακάτω παρουσιάζεται αναλυτικά ο πίνακας των επιρροών της συντηρησιμότητας στο μοντέλο ARISA Compendium



- CBO : coupling between objects
- DAC : data abstraction coupling
- DIT : depth of inheritance tree
- LD : Locality of data
- LCOM : lack of cohesion in methods
- ILCOM : improved LCOM variant
- MPC : message pass coupling
- NOC : number of children
- TCC : tight class cohesion
- LOC : Lines of code
- NAM : number of attributes and methods
- NOM : number of methods
- RFC : response for class
- WMC : weighted method count
- CYC : number of classes in cycle
- LEN : length of names
- LOD : Lack of Documentation

Αυτά τα metrics αποτελούν την σύνθεση του μοντέλου έτσι ώστε να παραχθεί το απαιτούμενο αποτέλεσμα για την καταταξη της συντηρησιμότητας του λογισμικού.
Σε σύγκριση με το MI, το μοντέλο ARiSA χρησιμοποιεί μια ευρύτερη επιλογή από μετρήσεις. Εκτός από τη μέτρηση LOC που χρησιμοποιείται συνήθως, χρησιμοποιεί επίσης το WMC ως μέτρηση πολυπλοκότητας, μαζί με πολλά γνωστά αντικειμενοστραφή μοντέλα , καλύπτοντας προβλήματα όπως η συνοχή, η σύζευξη και η κληρονομικότητα. Ενώ ο MI μπορεί να υπολογιστεί σε διάφορα επίπεδα ευαισθησίας και ευπάθειας κώδικα, το μοντέλο ARiSA περιορίζεται σε επίπεδο τάξης. Για να το κλιμακώσουμε σε επίπεδο συστήματος, εμείς να υπολογίσουμε τη γεωμετρική του μέση τιμή σε όλες τις κατηγορίες συστημάτων.

3.3.6 **SQALE MODEL**

Το SQALE(Software Quality Assessment Based on Lifecycle Expectations) εισήχθη για πρώτη φορά από τον J.L. Letouzey ως μέθοδος αξιολόγησης για την ποιότητα του πηγαίου κώδικα της εφαρμογής, με τρόπο ανεξάρτητο από τον γλώσσα προγραμματισμού ή οποιοδήποτε άλλο εργαλείο ανάλυσης. Ίσως το πιο γνωστό τέτοιο εργαλείο είναι η πλατφόρμα κώδικα SonarQube που βασίζει την φιλοσοφία της στο μοντέλο SQALE. Το λογισμικό είναι ανοιχτού κώδικα και ελεύθερο προς χρήση. Η υποστήριξη ανάλυσης παρέχεται μέσω plugins για συγκεκριμένη γλώσσα. Υποστήριξη για πρόσθετες γλώσσες ή λειτουργίες μπορεί να αναπτυχθεί με τη μορφή πρόσθετων plugins φτιαγμένων από εκάστοτε χρήστες. Για παράδειγμα, ο κώδικας C++ μπορεί να αναλυθεί χρησιμοποιώντας μια δωρεάν προσθήκη που έχει αναπτυχθεί από την κοινότητα. Τα plugins συνήθως περιλαμβάνουν έναν αριθμό από κανόνες, έναντι των οποίων ελέγχεται το δέντρο αφηρημένης σύνταξης του πηγαίου κώδικα κατά τη διάρκεια ανάλυσης. Ουσιαστικά, το λογισμικό διαθέτει μεθόδους που ελέγχουν snippets κώδικα του λογισμικού προς εξέταση ποιότητας και παράγουν μια εκτενή αναφορά με τυχόν ευπάθειες ή ευάλωτα σημεία στον κώδικα που δόθηκε. Κάθε κανόνας χαρακτηρίζεται από τη γλώσσα προγραμματισμού στην οποία εφαρμόζεται, τον τύπο της και τον τρόπο διερμηνείας της. Ο τύπος της κάθε γλώσσας είναι αυτός της συντήρησης, αξιοπιστίας (πιθανότητα σφάλματος) ή ασφάλειας (τρωτότητα του προγράμματος). Το μοντέλο χρησιμοποιεί ¨ετικέτες' για να ταυτοποιήσει το εκάστοτε πρόβλημα με κάθε ετικέτα να σχετίζεται με μία ή περισσότερες περιπτώσεις όπως αχρησιμοποίητος κώδικας, απόδοση ή υπερφόρτωση εγκεφάλου (π.χ. όταν η πολυπλοκότητα του κώδικα είναι πολύ υψηλή) ή ακόμα και λογικά λάθη στον κώδικα με την βοήθεια της τεχνητής νοημοσύνης σε πρόσφατα plugins. Η παραβίαση μιας ετικέτας οδηγεί σε ένα ζήτημα, το οποίο κληρονομεί τα χαρακτηριστικά του από την ετικέτα που παραβιάστηκε. Για παράδειγμα, η ετικέτα Java S106 δηλώνει ότι ¨οι εκφράσεις δεν πρέπει να είναι πολύ περίπλοκες». Η σωρεία ετικετών δημιουργεί κρίσιμα ζητήματα σοβαρότητας που επισημαίνονται με υπερφόρτωση εγκεφάλου για εκφράσεις που περιλαμβάνουν περισσότερους από 3 τελεστές. Η ωρα εκτιμάται ότι για να διορθωθεί το πρόβλημα είναι ένας σταθερός χρόνος 5 λεπτών στον οποίο προστίθεται 1 λεπτό για κάθε πρόσθετο χειριστή πάνω από το όριο. Αυτό πρακτικά σημαίνει ότι έχω αθοιστική πολυπλοκότητα όχι μόνο στο δεβυγγινγ του κώδικα καθάυτού αλλά και στην ανάλυση της ποιότητας του λογισμικού μου ιδιαίτερα σε περιπτώσεις που έχω conflicts ετικετών. Η συνολική τεχνική ωφέλεια μιας εφαρμογής υπολογίζεται ως το άθροισμα του εκτιμώμενου χρόνου που απαιτείται για την επίλυση όλων των προβλημάτων που εντοπίστηκαν. Άρα, η έννοια της ¨τεχνικής ωφέλειας' είναι αυτή που εν τοιαύτη περιπτώσει συγκροτεί το μοντέλο και μαθηματικοποιεί τα παραγόμενα αποτελέσματά του.

Το SonarQube χρησιμοποιεί τον ακόλουθο μαθηματικό τύπο για να παράξει μια ακόλουθη προσέγγιση της τεχνικής ωφέλειας του δοσμένου λογισμικού.
Η συνολική τεχνική ωφέλεια μίας εφαρμογής ή ενός λογισμικού συγκροτείται από τον ακόλουθο μαθηματικό τύπο:

$$TDR = \frac{TD}{DevTime}$$

όπου TD η συνολική τεχνική ωφέλεια του προγράμματος εκφρασμένη σε λεπτά και $DevTime$ ο συνολικός χρόνος που απαιτείται από τους προγραμματιστές έτσι ώστε να αναπτύξουν το λογισμικό.

Ενώ το SonarQube και παρόμοια εργαλεία παρέχουν ποσοτικά μοντέλα λογισμικού ποιότητα, η υπάρχουσα έρευνα επισήμανε επίσης ορισμένες υπάρχουσες παγίδες. Συγγραφείς κώδικα σε project ανοιχτού κώδικα μεγάλης κλίμακας έδειξαν ότι πολλά από τα αναφερόμενα ζητήματα παρέμειναν μη διορθωμένα το οποίο θα μπορούσε να είναι το αποτέλεσμα αυτών των εργαλείων που αναφέρουν πολλά ψευδώς θετικά αποτελέσματα, ή αποτελέσματα χαμηλής σημασίας. Μια μελέτη των προεπιλεγμένων κανόνων του SonarQube έδειξε επίσης τα περισσότερα από αυτά έχουν περιορισμένη τάση σε σφάλματα. Αυτό σημαίνει ότι λογισμικά όπως το SonarQube ή παρόμοια μπορούν να χρησιμοποιηθούν στην ανάπτυξη λογισμικού χωρίς όμως αυτό να προϋποθέτει πως δεν πρέπει να υπάρχουν συμπληρωματικά μοντέλα ελέγχου που προκαθορίζουν τα

πλαίσια ανάπτυξης προς αποφυγή περαιτέρω σφαλμάτων.

3.4 Απειλές στην Εγκυρότητα Λογισμικού

Έτσι ώστε να διατηρείται η εγκυρότητα σε ένα κομμάτι λογισμικού οφείλουν να τηρούνται 3 βασικές αρχές

- Δομική Εγκυρότητα: Η απειλή στην δομική εγκυρότητα ενός προγράμματος απορρέει από εσφαλμένα βαθμονομημένες μετρικές κώδικα, δηλαδή από κακώς ορισμένα και αριθμημένα στοιχεία κώδικα που συντελούν σε διαρροή στον έλεγχο εγκυρότητας, π.χ. δεν λαμβάνονται υπ΄όψιν οι ανάφορές @param ως μετρική.

- Εξωτερική Εκγυρότητα: Εδώ εστιάζουμε σε συγκεκριμένους εξωτερικούς παράγοντες δηλαδή εκτός περιβάλλοντος ανάπτυξης του λογισμικού όπως κάποια εσφαλμένη διαχείριση στους servers που είναι υπεύθυνοι για την διαχείριση storage των δεδομένων του λογισμικού. Μπορούμε να ελαχιστοποιήσουμε αυτήν την άπειλη χρησιμοποιώτας τα πρωτόκολλα διαχείρισης του OSI: Open Source Initiative

- Εγκυρότητα συμπερασμάτων : Η απειλή αφορά τα συμπεράσματα που έγιναν για την οικοδόμηση των μακροεντολών. Με ένα απλό παράδειγμα όταν πραγματοποιούμε έλεγχο με JUnit και δεν αξιολογούμε διακριτά όλα τα επιμέρους κομμάτια του λογισμικού δημιοργείται σφάλμα μακροεντολής δηλαδή αδυναμία του λογισμικού να περαιώσει μια τάδε λειτουργία λόγω ενδεχόμενου λάθους σε αυτή.

3.5 Σύγχρονοι Μέθοδοι Ανάπτυξης Ανοιχτού Λογισμικού

Προτού προχωρήσουμε στις σύχρονες μεθόδους ελέγχου της ποιότητας του ανοιχτού λογισμικού είναι απαραίτητο να γνωρίζουμε πως αναπτύσσεται το ανοιχτό λογισμικό, έτσι ώστε να διατυπωθεί μια πιο ολοκληρωμένη εικόνα της διαδικασίας.

Αρχικά, η σχεδίαση και η υλοποίηση λογισμικού ανοιχτού κώδικα διαφέρει σημαντικά από την παραδοσιακή μέθοδο ανάπτυξης λογισμικού. Η μεθοδολογία ανάπτυξης τέτοιου λογισμικού δεν είναι ξεκάθαρη και είναι δύσκολο να διαπιστωθούν τα μεμονωμένα τμήματα από τα οποία αποτελείται. Για αυτό το λόγο, project ανοιχτού κώδικα έχουν δεχτεί σκληρή κριτική για την αδαφάνεια της διαδικασίας ανάπτυξης τους.

Αυτό είναι και το μαγαλύτερο εμπόδιο που πρέπει να υπερβεί κανείς καθώς εισέρχεται στον κόσμο της σχεδίασης ανοιχτού λογισμικού. Εφόσον δεν υπάρχει μία ξεκάθαρα δομημένη διαδικασία, την οποία ένας νέος προγραμματιστής μπορεί να ακολουθήσει, απαιτείται πειραματισμός και απόκτηση έμπρακτης εμπειρίας για να κατανοήσει κανείς τις διαδικασίες και τις φάσεις της ανάπτυξης του ανοιχτού λογισμικού.

3.5.1 Διαδικασία Ανάπτυξης Λογισμικού Ανοιχτού Κώδικα

Αυτή η ενότητα ασχολείται με τις σύγχρονες, τελευταίας γενιάς μεθόδους σχεδίασης και ανάπτυξης που αφορούν το λογισμικό ανοιχτού κώδικα(στο εξής OSS - Open Source Software). Πιο συγκεκριμένα, αναφέρονται ονομαστικά τα βήματα που ακολουθάει ένας προγραμματιστής κατά την ανάπτυξη OSS. Συνοπτικά, τα βήματα που ακολουθούνται έχουν εξής:

1. Έρευνα απαιτήσεων
2. Ανακάλυψη προβλήματος ή ένταξη νέας λειτουργίας
3. Επιλογή η ανάθεση προβλήματος/λειτουργίας
4. Αναζήτηση λύσης

5. Ανάπτυξη κώδικα
6. Προσθήκη κώδικα στο αποθετήριο
7. Έλεγχος μονάδας
8. Ανασκόπηση κώδικα
9. Σύνταξη αναφοράς γνώμεων
10. Περαιτέρω βήματα με σκοπό την προσθήκη του νέου κώδικα στην τελική έκδοση του λογισμικού

Το τελευταίο βήμα, το οποίο αποσκοπεί στην ένταξη των αλλαγών στην τελευταία εκδοχή του κώδικα, είναι ουσιαστικά μία σειρά από βήματα τα οποία εξαρτόνται από το είδος του λογισμικού, το μέγεθος της ομάδας και την κατάσταση του project. Για παράδειγμα, μια μικρή ομάδα ενδέχεται να μην συζητάει και να μην αναλύει κάθε μεμονωμένη προσθήκη κώδικα ή να μην διαθέτει το λογισμικό στους χρήστες σε pre-Alpha έκδοση. Τέτοιοι παράγοντες επηρεάζουν τα τελευταία βήματα της ανάπτυξης και για αυτό δεν μπορούν να παρατεθούν τα τελικά βήματα εν συντομία, εντός της παραπάνω λίστας.

Επιπλέον, ορισμένες ομάδες προγραμματιστών έχουν επιλέξει μία πρακτική συνεχόμενης ενσωμάτωσης κώδικα (continuous integration practice). Σύμφωνα με αυτή την πρακτική, νέος κώδικας ενσωματώνεται συνέχεια και με γρήγορους ρυθμούς στον κεντρικό σκελετό του λογισμικού. Αυτή η διαδικασία καθιστά αναγκαία την υιοθέτηση μεθόδων που επιτρέπουν την άμεση και άνευ καθυστερήσεων ανασκόπηση του νέου κώδικα. Αυτό που συνήθως υιοθετείται είναι αναφορές από τις γνώμες και τις απόψεις των μελών της υπόλοιπης ομάδας μέχρις ότου ο νέος κώδικας να τηρεί ορισμένα κριτήρια που θέτει η ομάδα.

3.5.2 Χαρακτηριστικά Διαδικασίας Ανάπτυξης Λογισμικού Ανοιχτού Κώδικα

Η ειδοποιός διαφορά ανάμεσα στην ανάπτυξη λογισμικού ανοιχτού κώδικα και στην ανάπτυξη του παραδοσιακού λογισμικού εντοπίζεται στο ότι οι προγραμματιστές OSS δεν περνούν κάποια επίσημη διαδικασία προτού είναι σε θέση να προσφέρουν κώδικα. Οι προσθήκες κώδικα γίνονται κατά δύο τρόπους: την δημιουργία νέων χαρακτηριστικών και την διόρθωση λαθών(bugs).

Τα νέα χαρακτηριστικά ορίζονται συνήθως από τους κύριους προγραμματιστές του OSS, οι οποίοι ελέγχουν και την δομή του πρότζεκτ. Παρόλα αυτά, ¨μικρότεροι' προγραμματιστές ενδέχεται να δημιουργήσουν και αυτοί νέα χαρακτηριστικά ή να δημιουργήσουν ομάδες προγραμματιστών με σκοπό την δημιουργία ενός νέου feature.

Αφού τα νέα, επιθυμητά χαρακτηριστικά οριστούν ή αφού ήδη υπάρχοντα προβλήματα εντοπιστούν δημιουργόυνται τα λεγόμενα issues, συνήθως σε μια ιστοσελίδα όπως το Github που φιλοξενεί τον κώδικα του OSS, και μετέπειτα αναλαμβάνονται από προγραμαματιστές. Αυτή η ανάληψη εργασιών γίνεται από τον ίδιο τον προγραμματιστή χωρίς το issue να αναθέτεται σε αυτών από κάποιον άλλο. Ακόμα, σε πολλές εφαρμογές υπάρχουν τμήματα κώδικα που ¨ανήκουν' σε ορισμένα άτομα. Αυτά τα άτομα διατηρούν και αναπτύσσουν αυτό το συγκεκριμένο τμήμα του project. Αυτό δεν σημαίνει πως άλλοι προγρμαματιστές δεν μπορούν να συνεισφέρουν εξαιτίας κάποιας απαγόρευσης. Αντ' αυτού, υπάρχει σεβασμός προς αυτά τα μέλη της κοινότητας και για αυτό οι υπόλοιποι προγραμματιστές δεν ασχολόνται με τα δικά τους κομμάτια.

Έπειτα, έχοντας αναλάβει ένα συγκεκριμένο κομμάτι κώδικα(το issue δηλαδή), ο προγραμματιστής αναζητά να βρει τον τρόπο υλοποίησης του νέου χαρακτηριστικού ή επίλυσης του bug. Ουσιαστικά, το άτομο αναζητά πολλαπλές λύσεις και από αυτές επιλέγει συνήθως αυτή που προσωπικά θεωρεί ως καλύτερη έναντι των άλλων. Σε μερικές περιπτώσεις, όπου χρησιμοποιούνται προγράμματα επικοινωνίας ανάμεσα στους developers, το εν λόγω άτομο μπορεί να προωθήσει τις λύσεις που έχει βρει σε άλλα μέλη της ομάδας προκειμένου να ζητήσει μία δεύτερη γνώμη.

Κατά αυτόν τον τρόπο φτάνει ο προγραμματιστής στο κομμάτι της υλοποίησης του νέου feature ή της επίλυσης του προβλήματος. Σε αυτό το σημείο συγγράφεται ο νέος κώδικας που υλοποιεί το ζητούμενο αποτέλεσμα. Δεδομένου πως οι προγραμματιστές δουλεύουν εθελοντικά, είναι ασφαλές να υποθέσουμε πως οι περισσότεροι από αυτούς είναι και ¨παθιασμένοι' με αυτό που κάνουν. Για αυτό τον λόγο το λογισμικό ανοιχτού κώδικα παράγει και πιο ποιοτικό αποτέλεσμα από ότι το ιδιοταγές.

Ως επόμενο βήμα της διαδικασίας, ο νέος κώδικας προστίθεται στο αποθετήριο κώδικα. Κάθε project έχει την δική του πολιτική για την προσθήκη νέου κώδικα. Κάποιες ομάδες αφήνουν μόνο πεπειραμένους προγρμματιστές να προσθέτουν νέο κώδικα στην βάση. Οι προσθήκες των υπόλοιπων προγρμματιστών ελέγχονται τακτικά από έμπιστα μέλη έως ότου αυτοί να θεωρηθούν έμπιστα μέλη της ομάδας, όπου και τους δίνεται το ελεύθερο να ανεβάζουν νέο κώδικα μέσω commits κατά βούληση. Ακόμα, είναι αναγκαίο να τονίσουμε πως το άτομο που ανεβάζει τον νέο κώδικα δεν είναι απαραίτητα και ο συγγραφέας του. Ορισμένες ομάδες έχουν άτομα αποκλειστικά υπεύθυνα για τα commits και άλλους που μόνο προγραμματίζουν.

Ο νεοσύστατος όμως κώδικας δεν μπορεί να ενταχθεί απλά στο κεντρικό κλαδί του αποθετηρίου. Αυτό, διότι ενδέχεται η προσθήκη του να προκαλεί σφάλματα σε άλλα τμήματα του λογισμικού. Σε αυτό το σημείο έρχονται και τα χαρακτηριστικά της συντηρησιμότητας και της αρθρωτότητας του κώδικα, τα οποία κρίνουν κατά πόσο νέος κώδικας θα επηρεάσει το υπόλοιπο λογισμικό στο σύνολο του. Εν πάση περιπτώση και ανεξαρτήτως του πόσο καλά τετμημένο είναι το πρότζεκτ ή πόσο καλά προσεγμένος είναι ο νέος κώδικας, η ομάδα οφείλει να ελέγξει την λειτουργικότητα του project συνολικά, τρέχοντας test κώδικα.

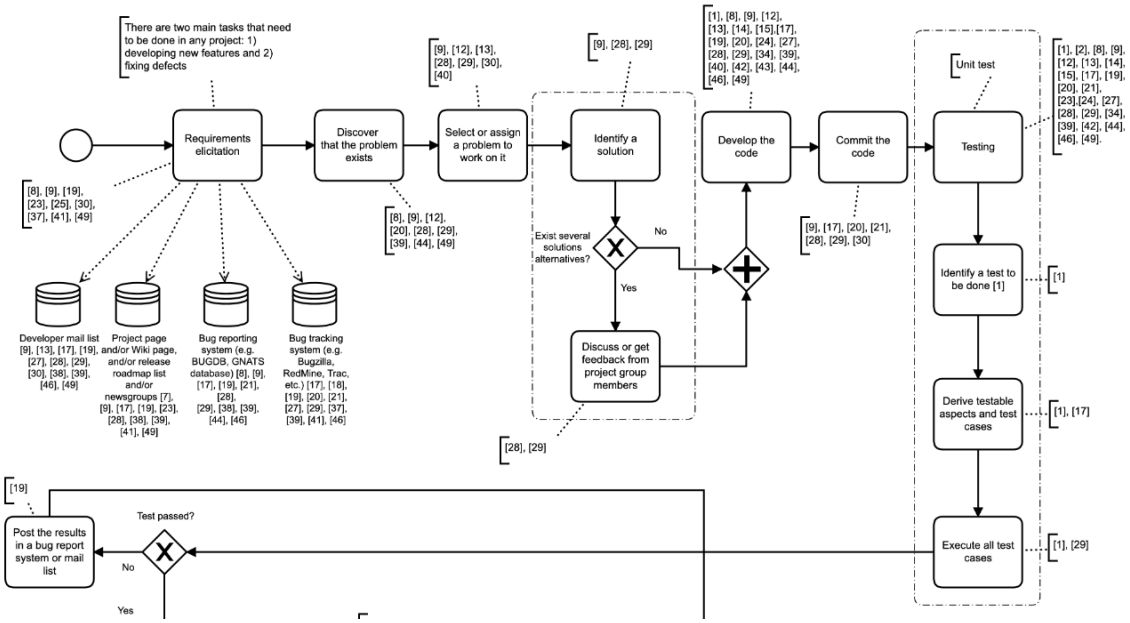
Η πιο συνηθισμένη μορφή τεστ είναι αυτή του ελέγχου μονάδας(unit testing). Δηλαδή, έλεγχοι συγκεκριμένων τμημάτων κώδικα. Η διαδικασία με την οποία δημιουργούνταιτα οι παραπάνω έλεγχοι συνήθως έχει ως εξής:

1. Εύρεση τμήματος που πρέπει να ελεγχθεί. Το κάθε τμήμα επιλέγεται με βάση το ενδιαφέρον του προγραμματιστή για το κάθε feature. Έτσι, δεν υπάρχει κάποια προτεραιότητα, η οποία καθορίζει το ποια τμήματα επιλέγονται για έλεγχο.

2. Δημιουργία ελέγχων για το συγκεκριμένο τμήμα.

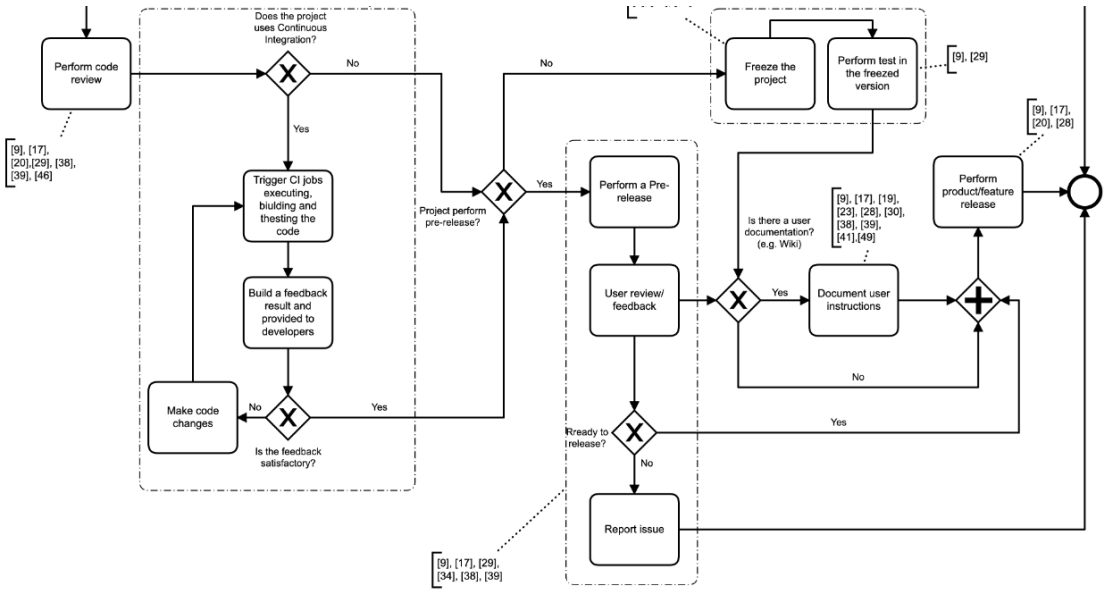
3. Εκτέλεση ελέγχων. Ο έλεγχος γίνεται στο τοπικό κλαδί του κώδικα.

Μετά την επιτυχημένη εκτέλεση των τεστ, ο νέος κώδικας προστήθεται στο αποθετήριο ή μία αναφορά των αλλαγών κοινοποιείται για τα υπόλοιπα μέλη της ομάδας. Στην περίπτωση όπου ο κώδικας δεν καταφέρει να περάσει όλους τους ελέγχους επιτυχώς, ο προγραμματιστής οφείλει να επιστρέψει στο προηγούμενο βήμα της ανάπτυξης OSS και να επεξεργαστεί τον κώδικα του έτσι ώστε οι έλεγχοι τμήματος να τερματίζουν άνευ προβλημάτων.



Σε αυτό το σημείο κάθε ομάδα ανάπτυξης OSS έχει τον δικό της τρόπο ένταξης του νέου κώδικα στο κεντρικό κλαδί του αποθετηρίου. Οι πιο μικρές ομάδες, με σχετικά μικρά σε σκέλος πρότζεκτ, ενδέχεται να εντάσσουν νέες προσθήκες κατευθείαν στο κεντρικό κλαδί. Όμως, μία τέτοια πρακτική θα ήταν μη βιώσιμη για μία μεγαλύτερη ομάδα και θα παραβίαζε όλες τις αρχές συγγραφής ¨καθαρού' κώδικα.

Μία συνήθης μέθοδος δημοσιοποίησης του λογισμικού είναι αυτή της pre-release έκδοσης. Ουσιαστικά, μία αρχική, αλλά λειτουργική έκδοση, του προγράμματος δημοσιοποιείται και είναι διαθέσιμοι για όλους. Κατά αυτόν τον τρόπο, περισσότεροι χρήστες δοκιμάζουν το λογισμικό πριν την τελική του έκδοση. Σε μια παραδοσιακή έκδοση, οι μόνοι που δρουν ως beta testers του λογισμικού είναι οι ίδιοι οι προγραμματιστές που το αναπτύσσουν. Αντιθέτως, σε μία pre-release έκδοση το πρόγραμμα αποκτά περισσότερους χρήστες και κατά συνέπεια περισσότερες ευκαιρίες για feedback.



3.6 Μοντέρνος Έλεγχος Ποιότητας

Η παρακάτω ενότητα ασχολείται με τους σύγχρονους τρόπους ελέγχου της ποιότητας του λογισμικού. Ως επί το πλείστον, τα περισσότερα προγράμματα ανοιχτού κώδικα σχεδιάζονται και υλοποιούνται με την βοήθεια λογισμικού ελέγχου πηγαίου κώδικα(version control systems), τα οποία επιτρέπουν σε μεγάλο αριθμό προγραμματιστών να συνεργάζονται πάνω στο ίδιο πρότζεκτ. Για αυτό το λόγο, η επόμενη υποενότητα αναλύει τα προγράμματα ελέγχου πηγαίου κώδικα.

3.6.1 Λογισμικό Ελέγχου Πηγαίου Κώδικα

Ο έλεγχος πηγαίου κώδικα ορίζεται ως η πρακτική σύμφωνα με την οποία παρακολουθούνται και διαχειρίζονται οι αλλαγές στον κώδικα του λογισμικού. Τα εργαλεία ελέγχου πηγαίου κώδικα είναι εξειδικευμένο λογισμικό, το οποίο συνεισφέρει στην διαχείριση του κώδικα από ομάδες προγραμματιστών. Ουσιαστικά, τα εν λόγω προγράμματα κρατούν κάθε αλλάγη στον πηγαίο κώδικα εντός μίας ειδικής βάσης δεδομένων. Εάν συμβεί κάποιο λάθος, οι προγραμματιστές μπορούν να επαναφέρουν τον πηγαίο κώδικα σε κάποια προηγούμενη έκδοση, αναιρώντας το πρόβλημα.

Επιπλέον, ο κώδικας ενός πρότζεκτ, μιας εφαρμογής ή ενός τμήματος λογισμικού είναι τυπικά οργανωμένος σε φακέλους. Έτσι, ένας προγραμματιστής μπορεί να δουλεύει ένα κομμάτι του κώδικας καθώς κάποιος άλλος αφαιρεί ένα bug από ένα άλλο κομμάτι. Τα προγράμματα ελέγχου πηγαίου κώδικα συνεισφέρουν σε αυτή την διαδικασία καθώς καταγράφουν κάθε μεμονομένη αλλάγή από κάθε προγραμματιστή ξεχωριστά. Αλλαγές σε ένα κομμάτι του κώδικα ενδέχεται να μην είναι συμβατές με αυτές σε ένα άλλο τμήμα. Τέτοια προβλήματα πρέπει να αναγνωριστούν και να απομονωθούν δίχως να παρεμποδίζουν την ανάπτυξη του υπόλοιπου λογισμικού. Τα προγράμματα ελέγχου πηγαίου κώδικα μπορούν να διατηρούν πολλά κλαδιά(branches) με διαφορετικές εκδόσεις του λογισμικού σε κάθε κλαδί, επιτρέποντας έτσι την εύκολη αντιμετώπιση των παραπάνω προβλημάτων.

Μακράν το πιο γνωστό λογισμικό ελέγχου πηγαίου κώδικα είναι το Git. Το Git είναι πλέον ενσωματωμένο σε πολλές πλατφόρμες και μπορεί να χρησιμοποιηθεί με πολλούς τρόπους. Γνωστά IDE όπως το Visual Studio της Microsoft, το Android Studio της Google και πολλά άλλα, περιέχουν προεγκατεστημένο το Git για πιο γρήγορο version control. Υπάρχει φυσικά και η παραδοσιακή μέθοδος μέσω του terminal ή και η επιλογή μίας γραφικής διεπαφής. Ακόμα, παρέχονται και διαδικτυές υπηρεσίες, όπως το Github, το Gitlab και το Bitbucket που προσφέρουν γενικευμένες λύσεις στον τομέα του ελέγχου πηγαίου κώδικα. Άλλα λογισμικά ελέγχου πηγαίου κώδικα είναι το Mercurial και το CVS(Concurrent Versions System).

3.6.2 Αποφυγή Κακόβουλου Λογισμικού

Το κύριο πρόβλημα ασφαλείας που αντιμετωπίζουν οι ομάδες ανάπτυξης ελεύθερου λογισμικού προκύπτει από το ότι ο καθένας είναι ελεύθερος να προσθέσει κώδικα στο project. Κάποιος κακόβουλος προγραμματιστής θα μπορούσε να εισάγει λογισμικό το οποίο βλάπτει το πρόγραμμα ή τον χρήστη του. Για αυτό το λόγο, οι ομάδες που αναπτύσσουν τέτοιου είδους λογισμικό, σχεδόν πάντα, δεν επιτρέπουν την ενσωμάτωσει νέου κώδικα άνευ ελέγχου από κάποιο έμπειρο και έμπιστο μέλος της ομάδας. Ακόμα, ο νέος κώδικας συνηθίζεται να προστίθεται σε ξεχωριστά από το κεντρικό κλαδία του λογισμικού ελέγχου πηγαίου κώδικα. Έτσι, πριν την συγχώνευση των κλαδιών στο κεντρικό τμήμα, ο κώδικας επανελέγχεται για τυχόν επικίνδυνα κομμάτια κώδικα που διέφυγαν.

Ως αποτέλεσμα, ο λόγος για τον οποίο το λογισμικό ανοιχτού κώδικα θεωρείται και ασφαλείες είναι η ύπαρξη αυτής της ¨ομάδας ματιών' που συνεχώς κοιτάει τις νέες προσθήκες. Κάτι τέτοιο

μπορεί να επιτευχθεί μόνο σε λογισμικό το οποίο είναι και διαθέσιμο άνευ πληρωμής, καθώς έτσι μπορεί να προσελκύσει μεγαλύτερο πλήθως ενδιαφερομένων.

Παρόλα αυτά, πρέπει να σημειωθεί πως τα περισσότερα πρότζεκτ τα διαχειρίζονται μικρές, ολιγομελείς ομάδες. Έτσι, δεν είναι ιδιαίτερα απίθανη η διείσδυση κακόβουλου κώδικα στο λογισμικό, καθώς και ο αριθμός των ατόμων που τακτικά επιθεωρούν τις νέες προσθήκες είναι μικρότερος. Επιπλέον, εκ φύσεως το ανοιχτό λογισμικό δεν αναγκάζει τους συνδρομητές ενός project να το συντηρήσουν. Οπότε, η δυνατότητα συντήρησης ενός πρότζεκτ μειώνεται καθώς οι προγραμματιστές που συνέσφεραν κώδικα φεύγουν και νέοι προγραμματιστές δεν είναι διατεθιμένοι να ανατρέξουν στον παλαιό κώδικα για να τον ανανεώσουν. Αυτό σημαίνει πως νέες αδυναμίες που ανακαλύπτονται με την πάροδο του χρόνου δεν διορθώνονται και το λογισμικό παραμένει επιρρεπής σε γνωστές πλέον επιθέσεις.

Κατά αυτό τον τρόπο έχουν υπάρξει ορισμένες περιπτώσεις όπου λογισμικό ανοιχτού κώδικα παρέμενε ευάλωτο σε επιθέσεις για πολλά χρόνια χωρίς αυτό να ήταν γνωστό στην ομάδα που το διαχειρίζοταν. Για παράδειγμα, στις 13 Μαΐου του 2008 ανακαλύφθηκε πως η έκδοση Debian του λειτουργικού συστήματος Linux διέθεται μία γεννήτρια τυχαίων αριθμών, της οποίας η συμπεριφορά μπορούσε να προβλεθεί. Αυτό έμπρακτα σήμαινε, πως χιλίαδες πακέτα είχα κρυπτογραφηθεί κατά τρόπο προβλέψιμο και έτσι η ακεραιότητα των εν λόγω πακέτων τέθηκε υπό αμφισβήτηση.

Άλλη μία γνωστή περίπτωση, όπου λογισμικού ανοιχτού κώδικα χτυπήθηκε και παρέμεινε μολυσμένο για μεγάλο χρονικό διάστημα είναι αυτή του Webmin. Το Webmin είναι λογισμικό διαχείρισης και ελέγχου διακομιστών τύπου Unix. Ένας άγνωστος hacker κατάφερε να εμφυτέψει ένα πρόγραμμα απομακρυσμένης πρόσβασης (backdoor) εντός της επίσημης έκδοδης του Webmin. Ως αποτέλεσμα, ο κακόβουλος χρήστης μπορούσε να αποκτήσει πρόσβαση σε κάθε υπολογιστή που κατέβαζε και εγκαταστούσε το Webmin. Αργότερα, αναγνωρίστηκε από την ομάδα σύνταξης του προγράμματος πως η συγκεκριμένη αδυναμία είχε παραμείνει κρυμμένη για περισσότερο από ένα χρόνο.

4 Ένα Αναλυτικό Παράδειγμα

4.1 Λόγοι και Αφορμές

- **Συνεργασία** Τα έργα ανοιχτού κώδικα μπορούν να δεχτούν αλλαγές από οποιονδήποτε στον κόσμο. Το Exercism, για παράδειγμα, είναι μια πλατφόρμα ασκήσεων προγραμματισμού με περισσότερους από 350 συντελεστές.

- **Υιοθέτηση και ανάμιξη**: Τα έργα ανοιχτού κώδικα μπορούν να χρησιμοποιηθούν από οποιονδήποτε για σχεδόν οποιονδήποτε σκοπό. Οι άνθρωποι μπορούν ακόμη και να το χρησιμοποιήσουν για να χτίσουν άλλα λογισμικά. Το WordPress, για παράδειγμα, ξεκίνησε ως διχάλα ενός υπάρχοντος έργου που ονομάζεται b2.

- **Διαφάνεια**: Οποιοσδήποτε μπορεί να επιθεωρήσει ένα έργο ανοιχτού κώδικα για σφάλματα ή ασυνέπειες. Η διαφάνεια έχει σημασία για κυβερνήσεις όπως η Γερμανια ή οι Ηνωμένες Πολιτείες, για τις ανάλογες βιομηχανίες και υπηρεσίες όπως ο τραπεζικός τομέας και η υγειονομική περίθαλψη.

4.2 Αρχίζοντας το **project** ή μία συνεισφορά

Για να μπορέσουμε να συμβάλλουμε σε ένα προθεςτ ανοιχτού κώδικα πρέπει να ακολουθήσουμε κάποιους βασικούς κανόνες που επιβάλλει η κοινότητα όσον αφορά την κύρια ροή στα requests των υποψήφιων developers. Ο έλεγχος ποιότητας μπορεί να πραγματοποιηθεί και από εμάς αλλά τον τελευταίο λόγο θα τον έχει η ομάδα των moderators του λογισμικού.

Πριν κάνουμε οτιδήποτε, κάνουμε έναν γρήγορο έλεγχο για να βεβαιωθούμε ότι η ιδέα μας δεν έχει συζητηθεί αλλού. Παρακάμπτουμε το README του έργου, τα requests (open closed), και τη λίστα των logs. Δεν χρειάζεται να ξοδεύουμε ώρες εξετάζοντας τα πάντα, αλλά μια γρήγορη αναζήτηση για μερικούς βασικούς όρους είναι πολύ σημαντική.

Εάν δεν μπορούμε να βρούμε την ιδέα μας αλλού, είμαστε έτοιμοι να κάνουμε μια κίνηση. Εάν το έργο βρίσκεται στο GitHub, πιθανότατα θα επικοινωνήσουμε με την ομάδα των προγραμματιστών ανοίγοντας ένα ισσυε ή ένα πυλλ ρεχυεστ.

Τα ισσυες είναι σαν να ξεκινάς μια συζήτηση όπου τα pull requests αφορούν την έναρξη εργασιών για μια λύση Για χαλαρή επικοινωνία, όπως διευκρινιστική ερώτηση, χρησιμοποιούμε Stack Overflow, IRC, Slack ή ά ά μί.

Προτού ανίξουμε ένα issue ή pull request, ελέγχουμε τα συνεισφέροντα έγγραφα του έργου (συνήθως ένα αρχείο που ονομάζεται CONTRIBUTING ή στο README), για να δούμε εάν χρειάζεται να συμπεριλάβουμε κάτι συγκεκριμένο. Για παράδειγμα, μπορεί να μας ζητηθεί να ακολουθήσουμε ένα πρότυπο ή να απαιτηθεί η χρήση δοκιμών.

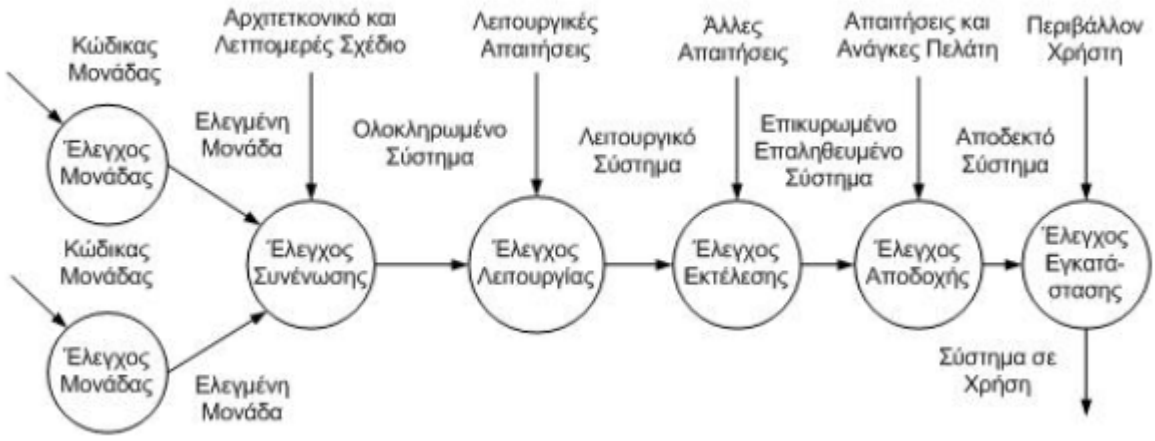
4.3 Συμβατικός Έλεγχος

Προφανώς, προτού παραδόσουμε ένα λογισμικό προς χρήση οφείλουμε να προβούμε σε κάποιους ελέγχους που θα καθορίσουν την ποσόστωση της λειτουργικότητας του και το άν θα γίνει public η όχι. Στην περίπτωση του ανοιχτού κώδικα ισχύουν ακριβώς οι ίδιοι παράγοντες με τα ιδιοταγή λογισμικά.

Ύπάρχουν τα κάτωθι στάδια ελέγχου:

- Ο έλεγχος λογισμικού έχει μία ροή από το «μικρό» (έλεγχος μονάδας) προς το «μεγάλο» (έλεγχος συστήματος)
- Έλεγχος μονάδας (unit testing). Ο έλεγχος μίας μονάδας λογισμικού (μία μέθοδος ή μία κλάση).

Στην παρακάτω εικόνα αναφέρεται σε αντίστοιχο σχεδιάγραμμα τα βήματα των σταδίων ελέγχου



4.3.1 Έλεγχος μονάδας

Η δοκιμή μονάδας είναι η διαδικασία συγγραφής και αυτόματης εκτέλεσης δοκιμών για να διασφαλιστεί ότι οι συναρτήσεις που κωδικοποιείτε λειτουργούν όπως αναμένεται. Παρόλο που

μπορεί να φαίνεται σαν περισσότερη δουλειά, στην πραγματικότητα πρόκειται για τη λήψη προληπτικών μέτρων για την εξάλειψη των σφαλμάτων πριν αυτά εμφανιστούν.

- Οι έλεγχοι μονάδας πραγματοποιούνται συνήθως από τους προγραμματιστές και εστιάζουν στον έλεγχο της ορθότητας του λογισμικού.
- Συνήθως είναι έλεγχοι ανοιχτού κουτιού (ωηιτε βοξ τεστινγ) επειδή ο έλεγχος γίνεται έχοντας γνώση της εσωτερικής δομής των μονάδων που ελέγχονται.
- Μπορεί όμως να είναι και έλεγχοι κλειστού κουτιού (βλαςκ βοξ) χωρίς δηλαδή να λαμβάνουμε υπόψη την εσωτερική δομή των μονάδων λογισμικού.

4.4 Γενικότεροι Έλεγχοι

Η σωστή εκσφαλμάτωση του λογισμικού πραγματοποιείται ανά στάδια και αποτελεί διαδικασία ενός συνόλου ελέγχων που συγκροτούν τον έλεγχο του συστήματος

4.4.1 Έλεγχος Λειτουργίας

Ο έλεγχος λειτουργίας εστιάζει στην ορθή λειτουργικότητα και μπορεί να αυτοματοποιηθεί με κατάλληλα εργαλεία (π.χ. Selenium).

4.4.2 Έλεγχος Εκτέλεσης

- Έλεγχοι πίεσης (stress tests). Αξιολογούν το σύστημα όταν δουλεύει υπό πίεση, στα όρια του, σε μια σύντομη χρονική περίοδο.
- Έλεγχοι χωρητικότητας (volume tests). Δείχνουν πως το σύστημα χειρίζεται μεγάλες ποσότητες δεδομένων.
- Έλεγχοι διάταξης (configuration tests). Αναλύουν τις διάφορες διατάξεις λογισμικού και υλικού που προσδιορίστηκαν από τις απαιτήσεις.
- Έλεγχοι συμβατότητας (compatibility tests). Εξετάζεται αν οι λειτουργίες των διεπαφών εκτελούνται σύμφωνα με τις απαιτήσεις.
- Έλεγχοι παλινδρόμησης (regression tests). Οι έλεγχοι παλινδρόμησης εγγυώνται ότι η εκτέλεση του νέου συστήματος είναι τουλάχιστον τόσο καλή όσο και του παλαιού
- Έλεγχοι ασφάλειας (security tests). Διαβεβαιώνουν ότι οι απαιτήσεις ασφάλειας έχουν ικανοποιηθεί.
- Έλεγχοι χρονισμού (timing tests). Αποτιμούν τις απαιτήσεις που ασχολούνται με χρόνους απόκρισης και χρόνους εκτέλεσης μιας λειτουργίας
- Περιβαλλοντικοί έλεγχοι (environmental tests). Εξετάζουν την ικανότητα του συστήματος να λειτουργεί στο χώρο εγκατάστασης.
- Έλεγχοι ποιότητας (quality tests). Αποτιμούν τα ποιοτικά χαρακτηριστικά του λογισμικού.
- Έλεγχοι ανάκαμψης (recovery tests). Ασχολούνται με την απόκριση του συστήματος όταν υπάρχουν σφάλματα ή όταν χαθούν δεδομένα ή όταν δεν λειτουργούν συσκευές ή όταν πέσει η ισχύς.
- Έλεγχοι συντήρησης (maintenance tests). Επαληθεύουμε την ύπαρξη και σωστή λειτουργία αυτών των βοηθητικών μέσων συντήρησης.

- Έλεγχοι τεκμηρίωσης (documentation tests). Επιβεβαιώνουν ότι έχουν γραφτεί τα απαιτούμενα έγγραφα τεκμηρίωσης.
- Έλεγχοι ανθρώπινων παραγόντων (human factors tests). Ερευνούν τις απαιτήσεις που σχετίζονται με την διεπαφή του χρήστη με το σύστημα

4.4.3 Έλεγχος Αποδοχής

- benchmark test: Ο έλεγχος που σχετίζεται με την δοκιμασία του δοθέντος λογισμικού στις περιπτώσεις εξαίρεσης της πλαισιωμένης λειτουργίας του και στην διαχείριση των σφαλμάτων του.
- πιλοτικός έλεγχος (pilot test)
- alpha test
- beta test
- parallel testing: Παράλληλος έλεγχος με συναφή λογισμικά.

4.4.4 Έλεγχος Εγκατάστασης

Έλεγχοι που αφορούν το περιβάλλον εκτέλεσης το λογισμικού. Συνήθως, αυτοί οι έλεγχοι πραγματοποιούνται μετά την πλήρη ανάπτυξη του λογισμικού αλλά και την εκσφαλμάτωσή του και αφορούν την λειτουργία σε διαφορετικά περιβάλλοντα όπως (π.χ. λειτουργικά συστήματα).

5 Επίλογος

Σε αυτήν την εργασία αποπειραθήκαμε να δώσουμε μία οπτική στην διαδικασία ανάπτυξης, εκσφαλμάτωσης, εκτέλεσης και ελέγχου λογισμικού ανοιχτού κώδικα. Ο στόχος μας στα πλάισια του μαθήματος "Επαλήθευση, Επικύρωση και Συντήρηση λογισμικού" ήταν να μπορέσουμε να εστιάσουμε πρωτίστως στο κομμάτι του ελέγχου του λογισμικού και για αυτόν ακριβώς τον λόγο παραθέσαμε σωρεία τακτικών και μετρικών ελέγχου και ποιότητας που χρησιμοποιούνται ανά καιρούς σε επιμέρους τμήματα λογισμικού a priori του τελικού release στο ευρύ κοινό.

6 Βιβλιογραφία

https://www.atlassian.com/git/tutorials/what-is-version-control
https://www.debian.org/security/2008/dsa-1571
https://arxiv.org/pdf/2009.00959.pdf
https://arxiv.org/pdf/2009.00959.pdf
https://opensource.org
https://github.com/ossu/computer-science
https://stackoverflow.com
https://www.archivematica.org
https://softeng.gr
https://www.freecodecamp.org/news/how-to-contribute-to-open-source-projects-beginners-guide https://ubuntu.com
https://www.kernel.org
Tamer Abdou, Peter Grogono, and Pankaj Kamthan. A conceptual framework for open source so-

ftware test process. pages 458 –463, 07 2012.

ΣΑΣ ΕΥΧΑΡΙΣΤΟΥΜΕ ΠΟΛΥ!