Π Α Ν Ε Π Ι Σ Τ Η Μ Ι Ο Π Α Τ Ρ Ω Ν
Πολυτεχνικη Σχολη
Τμημα Μηχανικων Ηλεκτρονικων Υπολογιστων & Πληροφορικης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Εργαλεία για την αξιολόγηση της ποιότητας λογισμικού»

ΚΟΡΔΑΣ ΑΘΑΝΑΣΙΟΣ

ΑΜ: 3949

ΥΠΕΥΘΥΝΟΣ ΚΑΘΗΓΗΤΗΣ:
Χρήστος Μπούρας, Καθηγητής

ΕΠΙΒΛΕΠΟΝΤΕΣ
Κωνσταντίνος Στάμος

ΠΑΤΡΑ, ΙΑΝΟΥΑΡΙΟΣ 2014

Περίληψη

Στα πλαίσια αυτής της διπλωματικής έχουν εξεταστεί και συγκριθεί εργαλεία για την αξιολόγηση της ποιότητας λογισμικού καθώς και οι θεωρητικές πτυχές της έννοιας «ποιότητα λογισμικού». Τα εργαλεία καλύπτουν τις γλώσσες προγραμματισμού C, C++ και JAVA. Για τις ανάγκες σύγκρισης και δοκιμής των θεωρητικών δυνατοτήτων αυτών των εργαλείων επιλέχθηκε μια πληθώρα δοκιμαστικών προγραμμάτων, τα οποία σαρώθηκαν από τα αντίστοιχα εργαλεία και εξάχθηκαν πειραματικά αποτελέσματα (συγκριτικοί πίνακες και γραφήματα) ανά εργαλείο.

Επίσης, αναπτύχθηκε σε γλώσσα C το εργαλείο “Static Analysis Tool” για σάρωση και αξιολόγηση κώδικα γραμμένου σε γλώσσα C. Το εργαλείο αυτό υπολογίζει μέσω της στατικής ανάλυσης κώδικα τις μετρικές:

- Συνολικές γραμμές κώδικα (physical lines of code)
- Γραμμές σχολίων (comment lines)
- Αναλογία γραμμών σχολίων/συνολικών γραμμών (comment ratio)
- Μετρικές Halstead (Halstead metrics)
- Μετρική ABC (ABC metric)
- Πλήθος κενών γραμμών (blank lines)
- Λογικές γραμμές κώδικα (logical lines of code)

Τα αποτελέσματα εξάγονται σε κοινή αναφορά (σε μορφή εγγράφου κειμένου). Το εργαλείο χρησιμοποιεί ελάχιστους πόρους συστήματος (μνήμη, υπολογιστική ισχύ) και εκτελείται ικανοποιητικά γρήγορα (μερικά λεπτά της ώρας) για μικρού (έως 100 χιλιάδες γραμμές κώδικα) και μεσαίου (έως 500 χιλιάδες γραμμές κώδικα) μεγέθους προγράμματα εισόδου.

Executive summary

In this Diploma thesis several tools have been examined and compared concerning software quality assessment, as well as theoretical aspects of the concept “software quality”. These tools cover the programming languages C, C++ and JAVA. In need of comparison and testing of the theoretical features of these tools, a variety of test programs have been chosen and scanned by corresponding tools and experimental results have been extracted (tables and graphs) per tool.

In addition, the tool “Static Analysis Tool” has been developed in the language C for scanning and assessing source code written in the language C. This tool calculates through static code analysis the following metrics:

- Physical lines of code
- Comment lines
- Comment to source code ratio
- Halstead metrics
- ABC metric
- Blank lines
- Logical lines of code

The results are extracted in a common report (in text document format). This tool uses very few system resources (memory, cpu power) and is executed acceptably fast (few minutes) for small (up to 100 thousand lines of code) and medium (up to 500 thousand lines of code) size input programs.

Πρόλογος

Ολοκληρώνοντας την παρούσα διπλωματική εργασία θα ήθελα να απευθύνω ευχαριστίες στα άτομα που με βοήθησαν στον ευρύτερο Πανεπιστημιακό χώρο, ο καθένας με τη δική του συμβολή, που χωρίς τη βοήθειά τους πιθανόν να μην ήταν δυνατή η εκπόνησή της.

Κατ' αρχάς, θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου κ. Χρήστο Μπούρα (Καθηγητής τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής του Πανεπιστημίου Πατρών) για την επίβλεψη της εργασίας, την εμπιστοσύνη που έδειξε στο πρόσωπό μου στην ανάθεσή της και για τη δυνατότητα που μου παρείχε ν' ασχοληθώ με ένα τόσο ενδιαφέρον αντικείμενο.

Ακόμα θα ήθελα να απευθύνω ιδιαίτερες ευχαριστίες στον Δρ. Κώστα Στάμο γιατί η βοήθειά του, η υπομονή του και η διάθεσή του να με υποστηρίξει υπήρξε καταλυτική στην εκπόνηση της διπλωματικής μου εργασίας.

Κλείνοντας, θα ήθελα να ευχαριστήσω την οικογένεια μου για τη ψυχική και υλική βοήθειά και την αμέριστη συμπαράσταση που μου προσέφερε κατά τη διάρκεια των προπτυχιακών σπουδών, επιτρέποντάς μου να τις ολοκληρώσω επιτυχώς.

Πάτρα, Ιανουάριος 2014

Αθανάσιος Ν. Κόρδας

Λίστα γραφημάτων

Λίστα Πινάκων

Ακρώνυμα

IDE	Integrated Development Environment
ISO	International Organization for Standardization
CISQ	Consortium for Internet Technologies software Quality
SEI	Software Engineering Institute
OMG	Object Management Group
API	Application Programmable Interface
WAI	Web Accessibility Consortium
W3C	World Wide Web Consortium
CPU	Central Processing Unit
GUI	Graphical User Interface
BLAST	Berkeley Lazy Abstraction Software verification Tool
BSD	Berkeley Software Distribution
CFA	Control Flow Automata
CIL	Common Intermediate Language
MISRA	Motor Industry Software Reliability Association
POSIX	Portable Operation System Interface
LAN	Local Area Network
ANSI	American National Standards Institute
STL	Standard Template Library

Περιεχόμενα

Κεφάλαιο 1) Βασικοί Όροι και Έννοιες

1.1) Ορισμός της ποιότητας (Quality)

Η ποιότητα είναι μια υποκειμενική και διαισθητική έννοια και ο ακριβής ορισμός της είναι δύσκολος. Ο καθένας αντιλαμβάνεται την ποιότητα με διαφορετικό τρόπο, διαφορετικά κριτήρια και προσδοκίες. Λόγω της υποκειμενικής της υπόστασης συχνά αναφερόμαστε σε εκφράσεις όπως «όταν την δω, θα την αναγνωρίσω», κάνοντας πολύ δύσκολο τον ορισμό μεθόδων για την εξασφάλισή της ή την παραγωγή προϊόντων που την επιτυγχάνουν. Όμως η ποιότητα είναι βασική πτυχή σε κάθε πλευρά της καθημερινότητάς μας, καθώς είναι ένα από τα επιθυμητά χαρακτηριστικά για οποιοδήποτε προϊόν παράγεται, για όλες τις υπηρεσίες που προσφέρονται και σε πληθώρα άλλων περιπτώσεων. Συνεπώς είναι πολύ σημαντικό να προσδιορίσουμε πως επιτυγχάνεται, πως μπορούμε να αντιληφθούμε την ύπαρξή της, όπως και τη βελτίωσή της όπου χρειάζεται.

Μερικοί ορισμοί της ποιότητας αναφέρονται στον βαθμό επίτευξης των στόχων του προϊόντος ή τον βαθμό τελειότητάς του, όμως κανένας από αυτούς δεν μπορεί να δώσει με σαφήνεια την έννοια της ποιότητας. Έτσι, σε πρακτικό επίπεδο συνήθως ως ποιότητα εκφράζεται ως ο βαθμός ικανοποίησης του πελάτη και ο βαθμός επίτευξης των αρχικών στόχων για το προϊόν. Με αυτόν τον τρόπο μπορεί η ποιότητα να «μετρηθεί» και να διαπιστωθεί κατά πόσο το τελικό αποτέλεσμα ικανοποιεί τις προσδοκίες των παραγωγών του και των πελατών. Μια άποψη για την ποιότητα ορίζεται αποκλειστικά από τον πελάτη και βασίζεται στην αξιολόγησή του ως προς τη συνολική εμπειρία του για τις προσφερόμενες υπηρεσίες ή προϊόντα. Στοιχεία που συνεισφέρουν και επηρεάζουν αυτή την αξιολόγηση είναι το πώς πωλήθηκε το προϊόν, πως παραδόθηκε, πόσο καλά λειτούργησε, πόσο καλή υποστήριξη έχει και άλλα.

Η ποιότητα στις επιχειρήσεις, στη Μηχανική και στη βιομηχανία έχει μια ρεαλιστική ερμηνεία ως μη κατωτερότητα ή ανωτερότητα κάποιου αντικειμένου. Εναλλακτικά ορίζεται ως «καταλληλότητα για σκοπό» (fitness for purpose). Για τους παραπάνω τομείς, η ποιότητα συνήθως συνίσταται σε 5 βασικά χαρακτηριστικά:

1. **Κατασκευή (Producing) :** Αφορά στην προμήθεια ενός συγκεκριμένου προϊόντος ή υπηρεσίας.
2. **Δοκιμή (Checking) :** Αφορά στην επιβεβαίωση της σωστής και σύμφωνης με τις προδιαγραφές κατασκευής του προϊόντος ή υπηρεσίας.
3. **Έλεγχος Ποιότητας (Quality Control) :** Αφορά τον έλεγχο και καθοδήγηση της διαδικασίας παραγωγής για τη διασφάλιση ότι τα αποτελέσματα είναι τα προβλεπόμενα.
4. **Διαχείριση Ποιότητας (Quality Management) :** Αφορά την καθοδήγηση ενός οργανισμού ώστε να βελτιστοποιήσει την απόδοσή του μέσω της ανάλυσης και της βελτίωσης.
5. **Διασφάλιση Ποιότητας (Quality Assurance) :** Αφορά στην επίτευξη της εμπιστοσύνης ότι το προϊόν ή η υπηρεσία θα είναι ικανοποιητικά (συνήθως από την πλευρά του πελάτη ή του αγοραστή).

1.2) Ορισμός του λογισμικού (Software)

Το λογισμικό αποτελείται από μια συλλογή προγραμμάτων υπολογιστή και σχετικά δεδομένα, τα οποία παρέχουν οδηγίες στον υπολογιστή για το τι ακριβώς θα κάνει καθώς και το πώς θα το κάνει. Το λογισμικό αναφέρεται σε ένα ή περισσότερα προγράμματα και δεδομένα που διατηρούνται αποθηκευμένα στον υπολογιστή για διάφορους λόγους. Συνεπώς, το λογισμικό ορίζεται ως ένα σύνολο προγραμμάτων, διαδικασιών, αλγορίθμων και των τεκμηριώσεών τους που απασχολούνται με τη λειτουργία ενός συστήματος επεξεργασίας δεδομένων. Τα προγράμματα λογισμικού εκτελούν τη λειτουργία του προγράμματος που εφαρμόζουν, είτε παρέχοντας άμεσες οδηγίες στο υλικό (hardware) του υπολογιστή, είτε λειτουργώντας ως είσοδος σε κάποιο άλλο κομμάτι λογισμικού. Ο όρος λογισμικό (software) εφευρέθηκε σε αντίθεση με τον όρο υλικό (hardware) και αντιστοιχούν στο άυλο και υλικό κομμάτι ενός συστήματος υπολογιστή.

Το υλικό του υπολογιστή περιλαμβάνει της φυσικές διασυνδέσεις και συσκευές που απαιτούνται ώστε να αποθηκεύεται (store) και να εκτελείται (execute) το λογισμικό. Στα χαμηλότερα επίπεδα ο εκτελέσιμος κώδικας αποτελείται από εντολές σε γλώσσα μηχανής, οι οποίες είναι ξεχωριστές για κάθε επεξεργαστή. Η γλώσσα μηχανής αποτελείται από ομάδες δυαδικών (binary) τιμών που υποδηλώνουν οδηγίες στον επεξεργαστή για τον τρόπο με τον οποίο θα αλλάξει την κατάσταση του συστήματος. Τα προγράμματα είναι τακτοποιημένες ακολουθίες εντολών που έχουν σκοπό να αλλάξουν την κατάσταση του συστήματος με συγκεκριμένο τρόπο. Συνήθως τα προγράμματα είναι σε γραμμένα σε γλώσσες προγραμματισμού υψηλού επιπέδου οι οποίες είναι ευκολότερες και πιο αποδοτικές για χρήση από τους ανθρώπους (είναι πιο κοντά στη φυσική γλώσσα). Οι υψηλού επιπέδου γλώσσες πρέπει να μεταγλωττίζονται (compiled) ή να μεταφράζονται (interpreted) σε πηγαίο κώδικα (source code) γλώσσας μηχανής. Εναλλακτικά, τα προγράμματα μπορούν να γράφονται σε γλώσσα συναρμολόγησης (assembly), η οποία ουσιαστικά αποτελεί μια μνημονική αναπαράσταση της γλώσσας μηχανής με φυσικό αλφάβητο. Και η γλώσσα συναρμολόγησης πρέπει να μεταγλωττιστεί σε γλώσσα μηχανής, κάτι το οποίο γίνεται από το συναρμολογητή (assembler).

Το λογισμικό περιλαμβάνει όλες τις διάφορες μορφές και ρόλους που μπορούν τα ψηφιακά αποθηκευμένα δεδομένα να έχουν και να παίξουν σε έναν υπολογιστή: κώδικας προς εκτέλεση στην Κεντρική Μονάδα Επεξεργασίας, μεταγλωττιστές, λειτουργικά συστήματα, εφαρμογές, δεδομένα, ιστοσελίδες και πολλά άλλα. Συνήθως οι εφαρμογές εκτελούνται πάνω σε ένα υποκρυπτόμενο λειτουργικό σύστημα όπως το Linux, τα Windows και άλλα. Τα πρακτικά συστήματα λογισμικού συνήθως διαχωρίζουν τα είδη του λογισμικού σε 3 βασικές κατηγορίες, οι οποίες όμως συχνά επικαλύπτονται ή τα όρια μεταξύ τους είναι δυσδιάκριτα:

- **Λογισμικού του Συστήματος (System Software) :** Το λογισμικό του συστήματος είναι σχεδιασμένο για να διαχειρίζεται το υλικό του υπολογιστή, να προσφέρει μια βασική λειτουργικότητα και να προσφέρει μια πλατφόρμα για την εκτέλεση του λογισμικού των εφαρμογών. Το λογισμικό του συστήματος περιλαμβάνει οδηγούς συσκευών (device drivers), λειτουργικά συστήματα (operating systems), εξυπηρετητές (servers), χρήσιμες εφαρμογές (utilities) και παραθυρικά συστήματα (window systems). Επίσης, είναι επιφορτισμένο με τη διαχείριση μιας ποικιλίας ανεξάρτητων εξαρτημάτων υλικού, έτσι ώστε να λειτουργούν μαζί αρμονικά. Ο σκοπός του είναι να απαλλάσσει τους προγραμματιστές λογισμικού εφαρμογών από τις συχνά περίπλοκες λεπτομέρειες ενός συγκεκριμένου εξαρτήματος που χρησιμοποιείται, όπως συσκευές επικοινωνίας (communication devices), εκτυπωτές (printers), αναγνώστες συσκευών (device readers), οθόνες (displays),

πληκτρολόγια (keyboards) καθώς και να κατανέμει τους πόρους του υπολογιστή όπως η μνήμη (memory) και ο χρόνος επεξεργαστή (processor time) σε ασφαλή και σταθερό τρόπο.

- **Λογισμικό προγραμματισμού (Programming Software) :** Το λογισμικό προγραμματισμού περιλαμβάνει εργαλεία με τη μορφή προγραμμάτων ή εφαρμογών τα οποία χρησιμοποιούν οι δημιουργοί λογισμικού για να δημιουργήσουν (create), αποσφαλματώσουν (debug), συντηρήσουν (maintain) και υποστηρίξουν (support) γενικότερα άλλα προγράμματα και εφαρμογές. Ο όρος συχνά αναφέρεται σε συγκριτικά απλά προγράμματα όπως μεταγλωττιστές (compilers), αποσφαλματωτές (debuggers), μεταφραστές (interpreters), συνδέτες (linkers) και επεξεργαστές κειμένου (text editors), τα οποία μπορούν να συνδυαστούν για την επίτευξη ενός στόχου, όπως αντίστοιχα μπορούν να χρησιμοποιηθούν πολλά εργαλεία χειρός για την επισκευή ενός φυσικού αντικειμένου. Ο σκοπός των εργαλείων προγραμματισμού είναι η υποστήριξη των προγραμματισμών στην συγγραφή προγραμμάτων υπολογιστή και πλέον συνδυάζονται σε ενσωματωμένα περιβάλλοντα ανάπτυξης (integrated development environment IDE) για την ευκολότερη διαχείριση όλων αυτών των λειτουργιών.

- **Λογισμικό Εφαρμογών (Applications Software):** Το λογισμικό εφαρμογών αναπτύσσεται για να υλοποιήσει οποιαδήποτε εργασία μπορεί να ωφεληθεί από υπολογισμούς. Είναι ένα σύνολο από προγράμματα που επιτρέπουν στον υπολογιστή να εκτελέσει εξειδικευμένες εργασίες επεξεργασίας δεδομένων για το χρήστη. Περιλαμβάνει μια ευρεία ποικιλία κατηγοριών λογισμικού, όπως προγράμματα περιήγησης στο διαδίκτυο (internet browsers), ηλεκτρονικά παιχνίδια (video games), λογιστικά προγράμματα (financial applications) και πολλά άλλα.

1.3) Ορισμός ποιότητας λογισμικού (Software Quality)

Όπως χρειαζόμαστε ποιότητα σε κάθε προϊόν ή υπηρεσία σε κάθε πτυχή της καθημερινής ζωής μας, έτσι και η ποιότητα στο λογισμικό είναι ζωτικής σημασίας. Είτε αναφερόμαστε στην διαδικασία ανάπτυξης του λογισμικού είτε στον έλεγχο της ορθότητας και σωστής λειτουργίας του, η ποιότητα συναντάται παντού. Πλέον τα προγράμματα και οι εφαρμογές αποκτούν τεράστιο μέγεθος με πολλές δεκάδες χιλιάδες (αν όχι εκατομμύρια) γραμμές κώδικα, με αποτέλεσμα να είναι αναγκαία πρότυπα για το διαρκή έλεγχο της ποιότητας των προγραμμάτων που αναπτύσσονται. Επίσης, λόγω του γεγονότος ότι η πολυπλοκότητα των προγραμμάτων αυξάνεται ραγδαία και τα επιμέρους κομμάτια ή λειτουργικές μονάδες αναπτύσσονται από διαφορετικά άτομα (λόγω του μεγέθους τους, του κόστους ανάπτυξης, των χρονικών περιορισμών και πολλών ακόμη αιτιών), αυξάνεται ραγδαία και το κόστος σε χρόνο και πόρους που χρειάζεται η αποσφαλμάτωση και έλεγχός τους. Συνεπώς, είναι αναγκαία η μέτρηση της ποιότητας του λογισμικού σε κάθε στάδιο ανάπτυξης και ελέγχου του λογισμικού.

Η ποιότητα λογισμικού αναφέρεται σε 2 σχετιζόμενες αλλά διαφορετικές ιδέες:

- **Λειτουργική Ποιότητα Λογισμικού (Software Functional Quality) :** Αναφέρεται στο πόσο καλά συμμορφώνεται ή εναρμονίζεται το λογισμικό με ένα δοθέν σχέδιο, το οποίο είναι βασισμένο στις λειτουργικές προδιαγραφές ή απαιτήσεις. Μπορεί να χαρακτηριστεί επίσης και ως καταλληλότητα για το σκοπό (Fitness for Purpose) για ένα κομμάτι λογισμικού ή το πώς συγκρίνεται με τους εμπορικούς ανταγωνιστές του ως ένα αξιόλογο λογισμικό. Συνήθως αξιολογείται μέσω της επιβολής και μέτρησης κατά τον δοκιμή του λογισμικού.

- **Δομική Ποιότητα Λογισμικού (Software Structural Quality) :** Αναφέρεται στο πως εκπληρώνονται οι μη λειτουργικές προδιαγραφές οι οποίες υποστηρίζουν την επίτευξη των λειτουργικών απαιτήσεων, όπως ευρωστία ή συντηρησιμότητα ή το βαθμό κατά τον οποίο το λογισμικό παράχθηκε σωστά. Συνήθως αξιολογείται μέσω της ανάλυσης των εσωτερικών χαρακτηριστικών της δομής του λογισμικού, του πηγαίου κώδικα, σε σχέση με το πόσο η αρχιτεκτονική τηρεί αξιόπιστες αρχές της αρχιτεκτονικής λογισμικού.

Βασισμένος στα πρότυπα ποιότητας *ISO 9126-3* και *ISO 25000:2005* ο οργανισμός *Consortium for IT Software Quality(CISQ)*, ένας ανεξάρτητος οργανισμός που έχει ιδρυθεί από το *Software Engineering Institute(SEI)* και το *Object Management Group(OMG)*, έχει ξεκάθαρα ορίσει τα ποιοτικά χαρακτηριστικά της δομικής ποιότητας λογισμικού. Αυτά τα 5 επιθυμητά χαρακτηριστικά ώστε ένα κομμάτι λογισμικού να έχει εμπορική αξία είναι:

1. **Αξιοπιστία (Reliability) :** Ένα γνώρισμα της προσαρμοστικότητας και της δομικής αντοχής. Η αξιοπιστία μετρά το επίπεδο κινδύνου και την πιθανότητα δυνητικών αποτυχιών των εφαρμογών. Επίσης μετρά τα εισαγόμενα ελαττώματα λόγω τροποποιήσεων που έγιναν στο λογισμικό, δηλαδή τη σταθερότητά του. Ο στόχος του ελέγχου και της παρακολούθησης της αξιοπιστίας είναι η μείωση ή πρόληψη των χρόνων που οι εφαρμογές διακόπτουν τη λειτουργία τους και λαθών που επηρεάζουν άμεσα τους χρήστες και επιδρούν στην επιχειρησιακή απόδοση μιας εταιρείας.

2. **Αποδοτικότητα (Efficiency) :** Τα γνωρίσματα του πηγαίου κώδικα και της αρχιτεκτονικής του λογισμικού είναι τα στοιχεία που διασφαλίζουν υψηλή απόδοση για τις εφαρμογές όταν αυτές βρίσκονται σε χρόνο εκτέλεσης. Η αποδοτικότητα είναι ζωτικής σημασίας για εφαρμογές που εκτελούνται σε περιβάλλοντα με υψηλή

ταχύτητα εκτέλεσης, όπως αυτά επεξεργασίας αλγορίθμων ή συναλλαγών, όπου η δυνατότητα επέκτασης μαζί με την αποδοτικότητα είναι ύψιστης σημασίας. Μια ανάλυση της αποδοτικότητας και της δυνατότητας επέκτασης του πηγαίου κώδικα προσφέρει μια ξεκάθαρη εικόνα για κρυφούς επιχειρησιακούς κινδύνους και της ζημιάς που μπορούν να προκαλέσουν στην ικανοποίηση των πελατών, λόγω της υποβάθμιση του χρόνου απόκρισης των συστημάτων.

3. **Ασφάλεια (Security) :** Ένα μέτρο για την πιθανότητα δυνητικών ρηγμάτων ασφαλείας, λόγω κακού προγραμματισμού και αρχιτεκτονικών τακτικών. Συνεπώς, ποσοτικοποιείται ο κίνδυνος για την αντιμετώπιση κρίσιμων αδυναμιών που προκαλούν ζημιές στην επιχείρηση.

4. **Συντηρησιμότητα (Maintainability) :** Η συντηρησιμότητα περιλαμβάνει τις έννοιες της προσαρμοστικότητας, της φορητότητας και της μεταφερσιμότητας (από τη μια ομάδα ανάπτυξης σε μια άλλη). Η μέτρηση και η παρακολούθηση της συντηρησιμότητας είναι επιβεβλημένη σε εφαρμογές κρίσιμου σκοπού, όπου οι αλλαγές καθοδηγούνται από σφιχτά προγράμματα παράδοσης στην αγορά και όπου είναι σημαντικό για τις Τεχνολογίες Πληροφορίας (Information Technologies) να παραμείνουν υπεύθυνες σε αλλαγές καθοδηγούμενες από εταιρείες. Επίσης, είναι βασικό μέλημα να διατηρούνται τα κόστη συντήρησης υπό έλεγχο.

5. **Επαρκές μέγεθος (Adequate Size) :** Παρόλο που δεν είναι ένα απόλυτο ποιοτικό γνώρισμα, οι διαστάσεις του πηγαίου κώδικα είναι προφανώς ένα χαρακτηριστικό του λογισμικού που επηρεάζει τη συντηρησιμότητα. Συνδυαζόμενο με τα προηγούμενα ποιοτικά χαρακτηριστικά μπορεί να χρησιμοποιηθεί για την εκτίμηση του ποσού δουλειάς που παράχθηκε, και αυτού που απομένει να παραχθεί, από τις ομάδες ανάπτυξης, καθώς και της παραγωγικότητάς τους σε σχέση με χρονοδιαγράμματα ή άλλες αποδεκτές μετρικές.

Η μέτρηση της ποιότητας λογισμικού αναφέρεται σε ποιο βαθμό αξιολογείται το σύστημα ή το λογισμικό σε αυτές τις 5 διαστάσεις. Μια συγκεντρωτική μέτρηση μπορεί να υπολογιστεί από ένα ποσοτικό ή ποιοτικό ή μεικτό σύστημα αξιολόγησης, που χρησιμοποιεί επιμέρους βάρη για να ορίσει τις προτεραιότητες. Αυτή η άποψη για το λογισμικό πρέπει να εμπλουτιστεί από την ανάλυση Κρίσιμων Προγραμματιστικών Λαθών (Critical Programming Errors), τα οποία κάτω από ορισμένες συνθήκες μπορούν να οδηγήσουν σε καταστροφικά αποτελέσματα ή σε υποβάθμιση της απόδοσης και μπορούν να οδηγήσουν σε ακαταλληλότητα του λογισμικού, ανεξάρτητα της αξιολόγησης στις συγκεντρωτικές μετρήσεις.

Η εξέλιξη της Μηχανικής Λογισμικού (Software Engineering) σε ώριμο στάδιο καθιστά όχι μόνο δυνατό, αλλά και επιτακτική ανάγκη τη μέτρηση της ποιότητας λογισμικού. Οι δυο κυριότεροι λόγοι ακολουθούν:

- **Διαχείριση Ρίσκου (Risk Management) :** Οι αποτυχίες του λογισμικού μπορούν να προκαλέσουν περισσότερα από απλή ενόχληση. Σφάλματα στο λογισμικό έχουν προκαλέσει ανθρώπινους θανάτους. Οι αιτίες ποικίλουν από κακώς σχεδιασμένες διεπαφές χρήστη μέχρι άμεσα προγραμματιστικά λάθη. Λόγω αυτών των ατυχημάτων έχουν προκύψει απαιτήσεις στην ανάπτυξη ορισμένων τύπων λογισμικού, ειδικότερα για συστήματα ενσωματωμένα σε ιατρικές εφαρμογές και σε συσκευές που ρυθμίζουν κρίσιμα συστήματα υποδομών.

- **Διαχείριση Κόστους (Cost Management) :** Όπως και σε όλα τα υπόλοιπα πεδία της Μηχανικής, μια εφαρμογή με καλή δομική ποιότητα λογισμικού κοστίζει λιγότερο

για τη συντήρησή της και είναι ευκολότερο να κατανοηθεί και να αλλάξει σε επείγουσες ανάγκες της επιχείρησης. Βιομηχανικά δεδομένα δείχνουν ότι κακή δομική ποιότητα στο λογισμικό των εφαρμογών σε βασικά συστήματα των επιχειρήσεων, οδηγεί σε υπερβάσεις στο κόστος, σε ανατροπή του σχεδιασμού και σε σπατάλες με τη μορφή διορθώσεων σε περίπου 45% του χρόνου ανάπτυξης σε μερικούς οργανισμούς. Επίσης, η κακή δομική ποιότητα λογισμικού σχετίζεται άμεσα με υψηλών επιπτώσεων διαταραχές στις επιχειρήσεις, λόγω αλλοιωμένων δεδομένων, διακοπών στη λειτουργία των εφαρμογών, κενών ασφαλείας και προβλημάτων απόδοσης.

1.4) Ορισμός Μηχανικής Λογισμικού (Software Engineering)

Ως Μηχανική Λογισμικού (Software Engineering) αναφέρεται η εφαρμογή μιας συστηματικής, πειθαρχημένης, μετρήσιμης προσέγγισης στην ανάπτυξη, λειτουργία και συντήρηση του λογισμικού, καθώς και η μελέτη αυτών των προσεγγίσεων. Όλα αυτά συνιστούν στην εφαρμογή της μηχανικής στο λογισμικό. Η μηχανική αναφέρεται επειδή οι προσεγγίσεις ενσωματώνουν σημαντικά στοιχεία από την επιστήμη των μαθηματικών, την επιστήμη των υπολογιστών και πρακτικές οι οποίες έχουν ρίζες στη μηχανική γενικότερα. Ως εναλλακτικοί ορισμοί αναφέρονται οι εξής:

- Η συστηματική προσέγγιση στην ανάλυση, στο σχεδιασμό, στην αξιολόγηση, στην ενσωμάτωση, στον έλεγχο, στη συντήρηση και στον ανασχεδιασμό του λογισμικού.

- Το ενδιαφέρον για την ανάπτυξη και συντήρηση συστημάτων λογισμικού τα οποία συμπεριφέρονται αξιόπιστα και αποτελεσματικά, υπάρχει η οικονομική δυνατότητα να αναπτυχθούν και να συντηρηθούν και ικανοποιούν όλες τις προδιαγραφές που έχουν τεθεί από τους πελάτες. Είναι πολύ σημαντική (η μηχανική λογισμικού) λόγω της επίδρασης μεγάλων, ακριβών συστημάτων λογισμικού και του ρόλου του λογισμικού σε εφαρμογές που είναι κρίσιμες ή έχουν να κάνουν με την ασφάλεια.

Η μηχανική λογισμικού συνήθως αποτελεί κομμάτι των σπουδών της επιστήμης των υπολογιστών και ειδικότερα της ανάπτυξης λογισμικού. Η καλή γνώση προγραμματισμού είναι απαραίτητη προϋπόθεση για την ενασχόληση με τον τομέα, πράγμα το οποίο οδηγεί σε άτομα με πτυχίο από την επιστήμη των υπολογιστών να κυριαρχούν στο χώρο. Οι κυριότερες κατηγορίες από τις οποίες αποτελείται η μηχανική λογισμικού είναι οι ακόλουθες:

- **Απαιτήσεις Λογισμικού (Software Requirements) :** Η εκμαίευση, η ανάλυση, ο προσδιορισμός και η επικύρωση των προδιαγραφών για το λογισμικό.

- **Σχεδιασμός Λογισμικού (Software Design) :** Η διαδικασία καθορισμού της αρχιτεκτονικής, των συστατικών μερών, των διεπαφών και άλλων χαρακτηριστικών ενός συστήματος ή συστατικού μέρους.

- **Κατασκευή Λογισμικού (Software Construction) :** Η λεπτομερής δημιουργία λειτουργικού, με νόημα λογισμικού, με τη χρησιμοποίηση ενός συνδυασμού προγραμματισμού, επικύρωσης, ελέγχου μονάδων, ενσωματωμένου ελέγχου και αποσφαλμάτωσης.

- **Δοκιμή Λογισμικού (Software Testing) :** Η δυναμική επικύρωση της συμπεριφοράς ενός προγράμματος για ένα πεπερασμένο πλήθος δοκιμαστικών σεναρίων, κατάλληλα επιλεγμένων από τα άπειρα δυνατά, αντίθετων με την αναμενόμενη συμπεριφορά.

- **Συντήρηση Λογισμικού (Software Maintenance) :** Η ολότητα των δραστηριοτήτων που απαιτούνται για να παρέχεται αποδοτική σε σχέση με το κόστος υποστήριξη στο λογισμικό.

- **Διαχείριση Δομής Λογισμικού (Software Configuration Management) :** Η ταυτοποίηση της δομής ενός συστήματος σε διαφορετικά σημεία στο χρόνο, για το σκοπό του συστηματικού ελέγχου των αλλαγών στη δομή και για τη διατήρηση της ακεραιότητας και της δυνατότητας ανίχνευσης της δομής καθ όλη τη διάρκεια του κύκλου ζωής του συστήματος.

- **Διαχείριση Μηχανικής Λογισμικού (Software Engineering Management) :** Η εφαρμογή δραστηριοτήτων διαχείρισης όπως ο σχεδιασμός, ο συντονισμός, η μέτρηση, η παρακολούθηση ο έλεγχος και η αναφορές, για τη διασφάλιση ότι η ανάπτυξη και η συντήρηση του λογισμικού παραμένουν συστηματικές, πειθαρχημένες και ποσοτικοποιημένες.

- **Διαδικασία Μηχανικής Λογισμικού (Software Engineering Process) :** Ο ορισμός, η ενσωμάτωση, η αξιολόγηση, η μέτρηση, η διαχείριση, η αλλαγή και η βελτίωση της ίδιας της διαδικασίας του κύκλου ζωής του προγράμματος.

- **Εργαλεία και Μέθοδοι Μηχανικής Λογισμικού (Software Engineering Tools and Methods) :** Τα εργαλεία βασισμένα σε υπολογιστή που έχουν σκοπό την υποστήριξη κατά τη διάρκεια του κύκλου ζωής του λογισμικού, όπως η υποβοηθούμενη από υπολογιστή Μηχανική Λογισμικού (Computer Aided Software Engineering), καθώς και οι μέθοδοι που επιβάλλουν συγκεκριμένη δομή στη μηχανική λογισμικού με σκοπό να κάνουν τη διαδικασία συστηματική και τελικά με μεγάλη πιθανότητα επιτυχημένη.

- **Ποιότητα Λογισμικού (Software Quality) :** Ο βαθμός κατά τον οποίο ένα σύνολο από έμφυτα χαρακτηριστικά ικανοποιούν τις προδιαγραφές.

1.5) Ορισμός εργαλείου αξιολόγησης και ανάλυσης κώδικα (Source Code Analyzers)

Ένας από τους πιο ραγδαία αναπτυσσόμενους τομείς στη βιομηχανία του λογισμικού είναι τα εργαλεία ανάλυσης πηγαίου κώδικα (source code analysis tools), γνωστά και ως εργαλεία στατικής ανάλυσης. Αυτά τα εργαλεία αναλύουν τον πηγαίο κώδικα γραμμή-γραμμή, με σκοπό να εντοπίσουν ευάλωτα σημεία και να προσφέρουν συμβουλές για την άμεση αντιμετώπιση, προτού ο κώδικας βγει στην παραγωγή. Ως εργαλεία αξιολόγησης και ανάλυσης του κώδικα εφαρμογών αναφέρονται ολοκληρωμένα περιβάλλοντα που υποστηρίζουν τους προγραμματιστές και τους διευκολύνουν στην ανάπτυξη λογισμικού. Ο ρόλος τους είναι βοηθητικός, καθώς δεν μπορούν να αντικαταστήσουν τον άνθρωπο όσον αφορά τον προγραμματισμό, αλλά μπορούν να βοηθήσουν και διευκολύνουν το έργο του όσον αφορά την αποσφαλμάτωση ή συγκεκριμένες αυτοματοποιημένες διαδικασίες. Τα περιβάλλοντα αυτά συνήθως προσφέρουν συγκεκριμένες δυνατότητες και διευκολύνσεις στους προγραμματιστές, ανάλογα με το είδος τους ή την έκδοσή τους.

Μέσω της κατανόησης του τρόπου που μπορεί να βελτιστοποιηθεί η ανάπτυξη λογισμικού με τη χρήση προχωρημένων εργαλείων (πέραν των απλών αποσφαλματωτών (Debuggers)) και τεχνικών, που περιλαμβάνουν προχωρημένες τεχνικές αποσφαλμάτωσης, εργαλεία για την απόδοση και τη λειτουργική έκταση του κώδικα, μπορεί να επιτευχθεί υψηλή ποιότητα στα συστήματα λογισμικού. Τα εργαλεία ανάλυσης λογισμικού και οι τεχνικές χρήσης τους, μπορούν να βοηθήσουν στην αντιμετώπιση των δαπανηρών και δύσκολα εντοπίσιμων σφαλμάτων, τα οποία προκαλούν αναστάτωση και καθυστερήσεις στην εξέλιξη, στη δοκιμή και στην ανάπτυξη. Επίσης, βοηθούν στην έγκαιρη ταυτοποίηση προβλημάτων, μειώνοντας ταυτόχρονα και το κόστος διόρθωσης (όσο πιο νωρίς εντοπίζεται ένα σφάλμα, τόσο πιο μικρό είναι το κόστος διόρθωσης) καθώς και συνεισφέροντας στην αποτροπή της εμπορικής κυκλοφορίας του λογισμικού με εκτεταμένα σφάλματα, πράγμα το οποίο προκαλεί ζημιές στα έσοδα και την αξιοπιστία της επιχείρησης.

Η κυριότερη διαφορά μεταξύ της παραδοσιακής αποσφαλμάτωσης και των αυτοματοποιημένων εργαλείων είναι ότι στα τελευταία δεν είναι απαραίτητη η διακοπή της εκτέλεσης της εφαρμογής για την εκτέλεση δοκιμών (όσον αφορά εργαλεία αξιολόγησης κατά το χρόνο εκτέλεσης(run-time)). Με τον παραδοσιακό τρόπο είναι απαραίτητη η επαναλαμβανόμενη διακοπή της εκτέλεσης για το διεξοδικό έλεγχο της ροής και λειτουργίας του κώδικα. Το ιδανικό λογισμικό ανάλυσης κώδικα προσκολλάται σε ένα εκτελούμενο σύστημα, προκαλώντας ελάχιστη παρενόχληση ή καθυστέρηση στο σύστημα.

Κεφάλαιο 2) Μέτρηση και Αξιολόγηση της Ποιότητας

2.1) Βασικά χαρακτηριστικά και τρόποι αξιολόγησης της ποιότητας λογισμικού

Η ποιότητα λογισμικού συνίσταται σε 6 υποκατηγορίες, κάθε μια από τις οποίες καθορίζει συγκεκριμένα χαρακτηριστικά του λογισμικού.

1. **Λειτουργικότητα (Functionality) :** Η λειτουργικότητα μπορεί να οριστεί ως ένα σύνολο χαρακτηριστικών, που έχουν σχέση με την ύπαρξη ενός συνόλου συναρτήσεων και των καθορισμένων ιδιοτήτων τους. Οι συναρτήσεις αυτές είναι που ικανοποιούν δηλωμένες ή υπονοούμενες ανάγκες και αποτελούν βασικό χαρακτηριστικό για κάθε λογισμικό. Σύμφωνα με το πρότυπο ISO 9126 (χρησιμοποιείται για την αξιολόγηση προϊόντων) η λειτουργικότητα μπορεί να χωριστεί σε 5 επιμέρους υποκατηγορίες:

    - **Καταλληλότητα (Suitability) :** Ορίζεται ως ένα χαρακτηριστικό το οποίο σχετίζεται με συναρτήσεις που καλύπτουν εξειδικευμένες ανάγκες και εξαρτάται από τους στόχους και τις ανάγκες σε κάθε περίπτωση.

    - **Ακρίβεια (Precision) :** Αφορά στην ακρίβεια των τιμών και των αποτελεσμάτων που υπολογίζονται.

    - **Διαλειτουργικότητα (Interoperability) :** Αφορά στην ιδιότητα του λογισμικού να επικοινωνεί με άλλα καθορισμένα συστήματα.

    - **Συμμόρφωση (Compliance) :** Αφορά στην προσκόλληση σε νόμους, καθορισμένα ή συμφωνημένα πρότυπα και κανονισμούς.

    - **Ασφάλεια (Security) :** Ορίζεται ως η ικανότητα να αποτρέπει μη εξουσιοδοτημένη πρόσβαση, είτε από λάθος είτε από πρόθεση, σε δεδομένα και προγράμματα.

2. **Αξιοπιστία (Reliability) :** Ορίζεται ως η ικανότητα το λογισμικό να διατηρεί την απόδοσή του κάτω από καθορισμένες συνθήκες και καθορισμένο χρονικό διάστημα. Σύμφωνα με το πρότυπο ISO 9126 χωρίζεται σε 3 υποκατηγορίες:

    - **Ωριμότητα (Maturity) :** Αναφέρεται στη συχνότητα αποτυχίας λόγω λαθών και σφαλμάτων στο λογισμικό. Συχνά θεωρείται ως περιγραφή για την αξιοπιστία από μόνη της. Είναι το πιο σημαντικό χαρακτηριστικό σε υψηλής κρισιμότητας συστήματα, όπου αποτυχία του λογισμού μπορεί να οδηγήσει σε τραυματισμό, θάνατο ή καταστροφή του περιβάλλοντος. Βέβαια ο κύριος παράγοντας αποτυχίας είναι ανθρώπινα σφάλματα κατά το σχεδιασμό του λογισμικού.

    - **Αντοχή σε σφάλματα (Fault Tolerance) :** Αναφέρεται ως η ιδιότητα του λογισμικού να διατηρεί ένα καθορισμένο επίπεδο απόδοσης, κυρίως σε περιπτώσεις σφαλμάτων ή παραβίασης της διεπαφής του.

    - **Ανανηψιμότητα (Recoverability) :** Αναφέρεται στην ιδιότητα του λογισμικού να ανακάμπτει μετά από σφάλματα και να αποκαθιστά την

λειτουργικότητα και τα δεδομένα που επηρεάστηκαν κατά τη δυσλειτουργία, σε καθορισμένα χρονικό διάστημα και με καθορισμένη προσπάθεια.

3. **Αποδοτικότητα (Efficiency) :** Η αποδοτικότητα αναφέρεται στη σχέση μεταξύ του επιπέδου απόδοσης που πετυχαίνει το λογισμικό και στους πόρους που χρησιμοποιεί, κάτω από ορισμένες συνθήκες. Σε περιπτώσεις όπου υπάρχουν φυσικοί περιορισμοί για την απόδοση (περιορισμοί χρόνου, χώρου για εφαρμογές πραγματικού χρόνου) καθορίζει αν το λογισμικό είναι κατάλληλο για χρήση. Σύμφωνα με το πρότυπο ISO 9126 χωρίζεται σε 2 υποκατηγορίες:

    - **Συμπεριφορά χρόνου (Time Behavior) :** Αναφέρεται στους χρόνους επεξεργασίας και ανταπόκρισης καθώς και στη ρυθμαπόδοση του συστήματος.

    - **Συμπεριφορά πόρων (Resource Behavior) :** Αναφέρεται στους πόρους που χρησιμοποιούνται και στο διάρκεια της χρησιμοποίησής τους.

4. **Συντηρησιμότητα (Maintainability) :** Η συντηρησιμότητα αφορά την προσπάθεια που χρειάζεται ώστε να γίνουν αλλαγές και τροποποιήσεις στο λογισμικό. Οι αλλαγές δεν αφορούν μόνο διορθώσεις σφαλμάτων του κώδικα, αλλά καλύπτουν και επεκτάσεις του, για την ενσωμάτωση νέων χαρακτηριστικών ή λειτουργιών. Συνεπώς, το λογισμικό πρέπει να έχει τέτοια σχεδίαση και δομή, ώστε να επιτρέπει την εύκολη και με χαμηλό κόστος συντήρηση ή επέκτασή του. Το χαρακτηριστικό αυτό είναι επιθυμητό τόσο σε λογισμικό ειδικής χρήσης, όπου αλλαγές μπορεί να χρειάζονται άμεσα, όσο και σε γενικής χρήσης λογισμικό όπου πρέπει να είναι εύκολη η συντήρηση και υποστήριξη του κώδικα. Επίσης, για να πραγματοποιηθούν αλλαγές είναι απαραίτητη η ανάλυση του κώδικα για τον εντοπισμό των σημείων που πρέπει να τροποποιηθούν, η αλλαγή του κώδικα στα καθορισμένα σημεία, διασφαλίζοντας όμως ότι οι αλλαγές αυτές δεν αλληλεπιδρούν με αρνητικό τρόπο στον υπόλοιπο κώδικα και τέλος η δοκιμή της νέας έκδοσης του λογισμικού. Σύμφωνα με το πρότυπο ISO 9126 η συντηρησιμότητα αποτελείται από τις εξής υποκατηγορίες:

    - **Δυνατότητα ανάλυσης (Analysis Capability) :** Αφορά τη δυσκολία στην ανάλυση και διάγνωση των δυσλειτουργιών ή των πηγών προβλημάτων ή την ταυτοποίηση των μονάδων που ευθύνονται για τα σφάλματα.

    - **Δυνατότητα τροποποίησης (Modification Capability) :** Αφορά στην προσπάθεια που χρειάζεται ώστε να τροποποιηθεί ο κώδικας, να διορθωθούν σφάλματα ή να γίνει αλλαγή περιβάλλοντος στο λογισμικό.

    - **Σταθερότητα (Stability) :** Αφορά στην πιθανότητα πρόκλησης ή εμφάνισης απροσδόκητων αποτελεσμάτων μετά τις τροποποιήσεις.

    - **Δυνατότητα ελέγχου (Controllability) :** Αφορά στην ευκολία να πιστοποιηθεί η σωστή και επιθυμητή λειτουργία του τροποποιημένου λογισμικού.

5. **Μεταφερσιμότητα (Transferability) :** Η μεταφερσιμότητα αναφέρεται στην προσπάθεια ή το κόστος που απαιτείται, ώστε να μεταφερθεί το λογισμικό από ένα περιβάλλον σε ένα νέο. Ο όρος περιβάλλον περικλείει τις έννοιες του λειτουργικού συστήματος, μηχανήματος και γενικά τα τεχνικά χαρακτηριστικά του υλικού που φιλοξενεί το σύστημα πάνω στο οποίο εκτελείται το λογισμικό. Συνήθως το

λογισμικό χρησιμοποιείται για περισσότερο σε σχέση με το περιβάλλον στο οποίο εκτελείται. Επίσης μπορεί να υπάρχει ανάγκη για δυνατότητα εκτέλεσης σε πολλαπλά συστήματα, κάνοντας τη μεταφερσιμότητα ένα πολύ σημαντικό χαρακτηριστικό. Σύμφωνα με το πρότυπο ISO 9126 η μεταφερσιμότητα αναλύεται στις εξής υποκατηγορίες:

- **Προσαρμοστικότητα (Adaptability) :** Αφορά στην δυνατότητα προσαρμογής και σωστής λειτουργίας σε κάθε περιβάλλον στο οποίο εκτελείται.

- **Ευκολία εγκατάστασης (Ease of Installation) :** Αφορά στην προσπάθεια ή το κόστος που απαιτείται, προκειμένου να εγκατασταθεί το λογισμικό στο σύστημα στο οποίο θα εκτελεστεί.

- **Δυνατότητα συμμόρφωσης (Compliance Capability) :** Αφορά στη δυνατότητα του λογισμικού να συμμορφώνεται με τις διάφορες συμβάσεις, προδιαγραφές και πρότυπα που χρησιμοποιούνται στα διάφορα συστήματα.

- **Δυνατότητα αντικατάστασης (Replacement Capability) :** Αφορά στην προσπάθεια ή το κόστος που απαιτείται για την αντικατάσταση του λογισμικού, όταν αυτό δεν θα χρειάζεται πια.

2.2) Χρήση διαδικασιών και μεθοδολογιών για την παραγωγή ποιοτικού λογισμικού

Η εξασφάλιση της ποιότητας είναι μια χρονοβόρα και δαπανηρή διαδικασία που ιδανικά διαρκεί από το στάδιο του σχεδιασμού του λογισμικού (των δυνατοτήτων, χαρακτηριστικών και στόχων του) μέχρι και το στάδιο όπου το λογισμικό μπορεί να χρησιμοποιηθεί εμπορικά ή με όποιον άλλο τρόπο έχει προβλεφθεί. Παρατηρούμε λοιπόν ότι η εξασφάλιση της ποιότητας στο λογισμικό είναι μια συνεχής διαδικασία, η οποία πιθανόν να συνεχίζεται και αφού το λογισμικό έχει ολοκληρωθεί και χρησιμοποιηθεί εμπορικά (νεώτερες εκδόσεις ή αναβαθμίσεις). Αυτή όμως η συνεχής διαδικασία απαιτεί πολλούς πόρους, τόσο ανθρώπινους όσο και υπολογιστικούς, για να επιτευχθεί καθώς και οργάνωση, κανόνες να τη διέπουν, σωστή διαχείριση και σαφείς στόχους προς επίτευξη.

Η μέτρηση της ποιότητας στο λογισμικό συνίσταται σε συγκεκριμένους παράγοντες, οι οποίοι μπορούν να παρακολουθηθούν, ώστε να υπάρχει κάποια μετρική για την αξιολόγηση της ποιότητας. Αυτοί οι παράγοντες αφορούν:

1. Αν το λογισμικό ανταποκρίνεται ή όχι στους στόχους που έχουν τεθεί.
2. Αν κατά την εκτέλεσή του λογισμικού προκαλούνται δυσλειτουργίες και προβλήματα συνέπειας.
3. Αν το λογισμικό είναι αξιόπιστο σε κάθε περίπτωση χρήσης του και έχουν ελαχιστοποιηθεί οι πιθανότητες δυσλειτουργίας του.
4. Αν υπάρχει δυνατότητα για μελλοντική επέκτασή του και το πόσο εύκολη/φθηνή είναι αυτή.

Η χρήση μετρικών αναφέρεται σε ποσοδείκτες που αντιστοιχούν σε μετρήσιμα μεγέθη σχετικά με το λογισμικό. Οι ποσοδείκτες αυτοί πρέπει να καθορίζονται κατά την φάση σχεδιασμού του λογισμικού και θα πρέπει να δίνουν πληροφορίες για το πόσο κοντά είναι το προϊόν στο στόχο και τις απαιτήσεις που έχουν τεθεί. Είναι δυνατός ο ορισμός νέων μετρικών, ανάλογα με την περίπτωση, διότι δεν μπορούν να χρησιμοποιηθούν καθολικές μετρικές για όλα τα προβλήματα. Σύμφωνα με τις μετρικές που έχουν οριστεί, πρέπει να υπάρχει συστηματική μέτρηση σε όλα τα στάδια ανάπτυξης. Τα δεδομένα που προκύπτουν από τις μετρήσεις και τους ελέγχους στα επιμέρους στάδια χρειάζονται ανάλυση και επεξεργασία, ώστε να παραχθούν χρήσιμα συμπεράσματα για την κατάσταση του προϊόντος. Όπως και οι μετρικές, οι διαδικασίες που θα χρησιμοποιηθούν πρέπει να καθοριστούν από την αρχή και να σχεδιαστούν ώστε να ανταποκρίνονται στις ανάγκες που υπάρχουν. Το αποτέλεσμα των παραπάνω στοιχείων δίνει ένα συστηματικό τρόπο για τη μέτρηση της ποιότητας του λογισμικού που παράγεται, τόσο στα επιμέρους στάδια όσο και στο τελικό αποτέλεσμα.

Η παραγωγή οποιουδήποτε λογισμικού συνίσταται σε 4 επιμέρους κομμάτια:

1. Ένα πλαίσιο για την ανάπτυξη, δηλαδή ένα μοντέλο ανάπτυξης του λογισμικού.
2. Ένα προκαθορισμένο αριθμό σταδίων και παραδοτέων σε κάθε στάδιο που συνιστούν την μεθοδολογία που ακολουθείται.
3. Ένα τρόπο επιβεβαίωσης και πιστοποίησης της ορθότητας των παραδοτέων και επικύρωσής τους.

4. Ένα τρόπο αναγνώρισης προϊόντων λογισμικού και ελέγχου της αλλαγής και τροποποίησής τους. Όσον αφορά τα διαθέσιμα μοντέλα ανάπτυξης λογισμικού, έχουν επικρατήσει τα μοντέλα καταρράκτη, σπιράλ και μοντέλο V.

Ακολουθεί ανάλυση των βασικών χαρακτηριστικών για τα επικρατέστερα μοντέλα ανάπτυξης λογισμικού:

Μοντέλο καταρράκτη

Σύμφωνα με το μοντέλο αυτό η ανάπτυξη λογισμικού αποτελείται από μια σειριακή διαδικασία, η οποία συνίσταται από τα εξής μέρη:

- Ανάλυση απαιτήσεων όπου σε συνεννόηση με τον πελάτη καθορίζονται οι απαιτήσεις του και καθορίζονται οι στόχοι του προϊόντος.
- Καθορισμός των λειτουργιών, όπως προκύπτουν από την ανάλυση απαιτήσεων, που θα υλοποιηθούν για την εφαρμογή.
- Σχεδιασμός και υλοποίηση του λογισμικού.
- Δοκιμή και έλεγχος του τελικού αποτελέσματος.

Το μεγαλύτερο μειονέκτημα του μοντέλου καταρράκτη είναι ότι ο έλεγχος του προϊόντος αφήνεται για το τελικό στάδιο ανάπτυξης, το οποίο μπορεί να δημιουργήσει προβλήματα. Παρ' όλα αυτά χρησιμοποιείται ευρέως στην βιομηχανία για την παραγωγή λογισμικού, ειδικά σε περιπτώσεις όπου δεν υπάρχει κάποιο επίσημο μοντέλο για την ανάπτυξη λογισμικού. Στην ευρεία χρήση του συμβάλλει η απλή δομή και η εύκολη εφαρμογή του.

Μοντέλο σπιράλ

Σύμφωνα με το συγκεκριμένο μοντέλο η ανάπτυξη του λογισμικού γίνεται σε διαδοχικούς κύκλους, όπου σε κάθε κύκλο δημιουργείται ένα πρωτότυπο. Το πρωτότυπο αυτό χρησιμοποιείται για να καθοριστεί ο βαθμός επίτευξης των στόχων του προϊόντος και σε περίπτωση αποτυχίας το πρωτότυπο ανασχεδιάζεται και εξελίσσεται. Στην τελική φάση ανάπτυξης, η οποία όμως δεν είναι ξεκάθαρη ούτε καθορισμένη εξ' αρχής λόγω της δομής του μοντέλου, η ανάπτυξη του λογισμικού μπορεί να ακολουθήσει το μοντέλο καταρράκτη, εφόσον έχουν καθοριστεί οι στόχοι και οι απαιτήσεις για το προϊόν. Η περίπλοκη δομή καθιστά το μοντέλο αυτό δύσχρηστο και απαιτεί σωστή διαχείριση των επιμέρους φάσεων, για την επίτευξη των στόχων. Βέβαια, η αυξανόμενη δημοτικότητα των τεχνικών που χρησιμοποιούν πρωτότυπα, αυξάνουν τη δημοτικότητα και χρήση του μοντέλου.

Μοντέλο V

Το μοντέλο αυτό αναφέρεται σε μια πιο θεωρητική εκδοχή του μοντέλου καταρράκτη. Στο μοντέλο V ο έλεγχος και οι δοκιμές για την ορθότητα του λογισμικού γίνονται και σε πρώιμα στάδια ανάπτυξης, σε αντίθεση με το μοντέλο καταρράκτη, βελτιώνοντας το τελικό αποτέλεσμα και διευκολύνοντας την ανίχνευση σφαλμάτων. Λόγω της δομής του συνήθως δεν υλοποιείται εξ' ολοκλήρου, αλλά συνήθως ακολουθείται η

γενική ιδέα της δοκιμής και επιβεβαίωσης κάθε σταδίου μετά την υλοποίησή του, αντί του συνολικού προϊόντος στο τέλος της ανάπτυξής του.

Η έννοια της μεθοδολογίας που ακολουθείται συνίσταται σε ένα σύνολο επιμέρους στοιχείων που αφορούν γραφικές ή γραπτές σημειώσεις, ορισμούς των επιμέρους φάσεων ανάπτυξης και το σκοπό τους, το σύνολο των αναμενόμενων εισόδων και εξόδων κάθε φάσης και τα παραδοτέα που απαιτούνται από κάθε φάση. Η εφαρμογή μεθοδολογιών απορροφά μέρος των πόρων από την ανάπτυξη του λογισμικού, αλλά συμβάλει στην καλύτερη οργάνωση των αποτελεσμάτων και τη διαχείριση των επιμέρους φάσεων της ανάπτυξης.

Η επιβεβαίωση της ορθότητας και η πιστοποίηση των παραδοτέων είτε ανά φάση είτε του τελικού αποτελέσματος είναι μια απαραίτητη διαδικασία. Με αυτό τον τρόπο εξασφαλίζεται στον πελάτη η ανταπόκριση στις απαιτήσεις του και η σωστή λειτουργία του λογισμικού.

Η διαχείριση των επιμέρους λειτουργικών μονάδων του λογισμικού και η δημιουργία του ολοκληρωμένου τελικού προϊόντος είναι μια δύσκολη διαδικασία. Για την σωστή λειτουργία όλων των υπομονάδων ανεξάρτητα και μεταξύ τους, είναι απαραίτητη η πειθαρχία και η ορθή οργάνωσή τους. Σημαντικό ρόλο σε αυτή τη διαδικασία έχει η αναγνωρισιμότητα με μοναδικό τρόπο κάθε συστατικού μέρους, ένας συνεπής τρόπος ελέγχου τυχόν αλλαγών στα επιμέρους μέρη και αλληλεπιδράσεων μεταξύ τους καθώς και η δυνατότητα να εντοπίσουμε την προέλευση αντικειμένων και να μπορούμε να τα τροποποιήσουμε σε περίπτωση αλλαγής προδιαγραφών.

2.3) Τεχνικές ανάλυσης και αξιολόγησης λογισμικού

Οι τεχνικές ανάλυσης λογισμικού βασίζονται σε επιλεγμένες μετρικές και στην επεξεργασία και ανάλυσή τους. Έχουν προταθεί εκατοντάδες διαφορετικές μετρικές τα τελευταία 20 χρόνια αλλά μόνο λίγες από αυτές χρησιμοποιούνται στην πραγματικότητα, λόγω της πρακτικότητας και του σχετικά μικρού κόστους υλοποίησης. Οι 3 κυριότερες τεχνικές που αξιοποιούν τις πιο συνηθισμένες μετρικές είναι η στατική ανάλυση, η ανάλυση χρόνου εκτέλεσης και η επιθεώρηση.

Στατική ανάλυση (Static Analysis)

Η στατική ανάλυση αφορά ένα τρόπο αξιολόγησης του λογισμικού, όπου δεν προσομοιώνεται η εκτέλεση του λογισμικού. Συνήθως αναφέρεται στην ανάλυση του πηγαίου κώδικα ή των στατικών χαρακτηριστικών του. Τα χαρακτηριστικά που εξετάζονται αφορούν κυρίως:

- **Ανάλυση ανωμαλιών (Anomalies Analysis) :** Η ανάλυση ανωμαλιών αναφέρεται σε τυπικές γλώσσες και ειδικότερα σε πηγαίο κώδικα και εξετάζει τα επιμέρους χαρακτηριστικά του πηγαίου κώδικα, τα οποία θα μπορούσαν να προκαλέσουν σφάλματα και μη μεταφερσιμότητα. Χρησιμοποιείται για την εκτίμηση της αξιοπιστίας και της μεταφερσιμότητας του πηγαίου κώδικα.

- **Λεκτική ανάλυση (Verbal Analysis) :** Η λεκτική ανάλυση χρησιμοποιείται ώστε να καθοριστεί το πλήθος και είδος των χαρακτήρων ή των λέξεων που περιέχονται στον πηγαίο κώδικα και καθορίζουν το πόσο εύκολα συντηρήσιμος και πόσο εύχρηστος είναι ο κώδικας.

- **Δομική ανάλυση (Structural Analysis) :** Η δομική ανάλυση αναφέρεται στην εξαγωγή διαγραμμάτων ροής ή γράφων καταστάσεων για τον κώδικα και εκτίμηση της πολυπλοκότητάς του και του κόστους συντήρησης που προκύπτει.

- **Ανάλυση με παραπομπές (Reference Analysis) :** Η ανάλυση με παραπομπές χρησιμοποιεί ειδικές προδιαγραφές και ελέγχους για να εκτιμήσει τη διαλειτουργικότητα του κώδικα και την επίτευξη τεχνικών χαρακτηριστικών που απαιτούνται.

Ανάλυση χρόνου εκτέλεσης (Run-Time Analysis)

Η ανάλυση χρόνου εκτέλεσης ή δυναμική ανάλυση απαιτεί την εκτέλεση του κώδικα είτε στην μηχανή για την οποία προορίζεται είτε σε μια μηχανή που προσομοιώνει τις συνθήκες λειτουργίας που θα τρέχει ο κώδικας. Υπάρχουν 2 διαφορετικές προσεγγίσεις που αφορούν την θεώρηση «μαύρου κουτιού» ή «κλειστού κουτιού», όπου μας ενδιαφέρει μόνο η σχέση των εισόδων του κώδικα με τις εξόδους που παράγονται και την θεώρηση «λευκού κουτιού» ή «ανοικτού κουτιού», όπου μας ενδιαφέρει και η λειτουργία των εσωτερικών υπομονάδων και λειτουργιών. Η δοκιμή του κώδικα γίνεται με βάση ειδικά επιλεγμένα δεδομένα που τροφοδοτούνται στην είσοδο με σκοπό τον έλεγχο της ορθής λειτουργίας του κώδικα και της επίτευξης των στόχων του. Προφανώς δεν είναι δυνατός ο εκτεταμένος ή ολοκληρωτικός έλεγχος του κώδικα, καθώς αυτό θα απαιτούσε σημαντικό ποσοστό των πόρων ανάπτυξης (εξαίρεση αποτελούν ειδικές ή πολύ μικρές εφαρμογές).

Η διαδικασία που ακολουθείται αφορά σε 3 βήματα:

1. Συλλογή δεδομένων κατά τη φάση ελέγχου.
2. Επανεκτέλεση των σεναρίων ελέγχου που έχουν προετοιμαστεί.
3. Εκτέλεση επιλεγμένων ελέγχων και δοκιμών κατά την φάση αξιολόγησης.

Τα αποτελέσματα που προκύπτουν από την επεξεργασία των δεδομένων που συλλέχθηκαν κατά τη φάση δοκιμής αξιολογούνται και γίνονται αλλαγές και διορθώσεις στον κώδικα, όπου αυτό απαιτείται.

Επιθεώρηση (Inspection)

Η ανάλυση μέσω επιθεώρησης αναφέρεται σε ομάδες ενεργειών που γίνονται από ανθρώπους για την αξιολόγηση του λογισμικού. Το μειονέκτημα της μεθόδου είναι η έλλειψη μετρικών που μπορούν να χρησιμοποιηθούν για την αξιολόγηση. Μια ευρείας αποδοχής λύση αποτελεί η χρήση λιστών ελέγχου, όπου κάθε ερώτηση αφορά σε συγκεκριμένα χαρακτηριστικά, με πεπερασμένο αριθμό απαντήσεων και συνδέεται με μια βαθμολογία. Οι επιμέρους βαθμολογίες μπορούν να έχουν βάρη, ανάλογα με την σπουδαιότητα του χαρακτηριστικού που αποτιμάται και συναθροίζονται για τον τελικό βαθμό της αξιολόγησης. Το πλεονέκτημα της μεθόδου είναι η απλότητα εφαρμογής και η ευκολία σύγκρισης λογισμικών. Όμως η προσέγγιση των λιστών ελέγχου δεν παράγει δεδομένα από την φάση ελέγχου και συνήθως απαιτεί πολύ χρόνο, λόγω του ότι γίνεται από ανθρώπους.

2.4) Βασικές αρχές και διαδικασίες για σωστές μετρήσεις

Όταν αναφερόμαστε σε μετρήσεις, εννοούμε τον καθορισμό μετρήσιμων μεγεθών για τη σύγκριση ορισμένων χαρακτηριστικών. Στην καθημερινή ζωή ή σε ορισμένους τομείς της επιστήμης, δεν είναι απαραίτητος ο ακριβής καθορισμός των μετρήσιμων μεγεθών ή των συνθηκών κάτω από τις οποίες γίνονται. Οι μετρήσεις στο λογισμικό πρέπει να έχουν σαφείς στόχους και τα μετρήσιμα χαρακτηριστικά ή μεγέθη σωστό σχεδιασμό. Αποτυχία στον ορθό καθορισμό των μετρήσιμων μεγεθών οδηγεί σε συλλογή και επεξεργασία δεδομένων για των κώδικα, τα οποία όμως δεν προσφέρουν πληροφορίες για την ποιότητα ή τα επιθυμητά χαρακτηριστικά του. Επίσης είναι απαραίτητο να διατυπώνονται πλήρως τα μεγέθη που επιλέγονται για μέτρηση, ώστε να μην υπάρχουν παρερμηνείες.

Η μεγαλύτερη δυσκολία στον καθορισμό των μεγεθών που θα μετρηθούν στο λογισμικό είναι η έκφραση με μαθηματικά μοντέλα, τύπους ή αυστηρά καθορισμένα σύνολα αόριστων ή επιθυμητών χαρακτηριστικών, για τα οποία δεν υπάρχει σημείο αναφοράς. Συνεπώς, οδηγούμαστε στη χρήση διαδικασιών και κανόνων από την καθημερινή ζωή ως σημείο εκκίνησης για τον καθορισμό τελικά των χαρακτηριστικών που θέλουμε. Τέτοιες διαδικασίες είναι:

- **Ο εμπειρικός κανόνας (rule of thumb) :** όπου χρησιμοποιούμε συγκρίσεις μεταξύ αντικειμένων ή χαρακτηριστικών, χωρίς να τα έχουμε ποσοτικοποιήσει πρώτα. Έτσι μπορούμε να απομονώσουμε τα μεγέθη που μας ενδιαφέρουν και να τα εκφράσουμε με πιο αυστηρό τρόπο.

- **Ο κανόνας της χαρτογράφησης (mapping rule) :** όπου καθορίζονται συγκεκριμένα σύνολα με απαντήσεις σε καθορισμένες ερωτήσεις ή εκφράσεις. Με αυτό τον τρόπο μπορούμε να ορίσουμε σύνολα απαντήσεων για να εκφράσουμε υποκειμενικές έννοιες που δεν μπορούν να προσδιοριστούν σαφώς και εύκολα μέσω μαθηματικών τύπων. Στη συνέχεια γίνεται μια αντιστοίχηση κάθε μέλους του συνόλου απαντήσεων με ένα ποσοδείκτη και οι ποσοδείκτες που ορίζονται χρησιμοποιούνται για τον καθορισμό των μετρήσιμων μεγεθών.

- **Ο κανόνας των λιστών επιλογής (choice list rule) :** όπου καθορίζεται μια λίστα από χαρακτηριστικά, μεγέθη και ιδιότητες, τα οποία είτε επιβεβαιώνονται (επιλέγονται) είτε απορρίπτονται (δεν επιλέγονται). Όπως και με τον κανόνα της χαρτογράφησης χρησιμοποιείται κυρίως για την εξαγωγή μετρήσιμων μεγεθών από υποκειμενικά, γενικά ή αόριστα χαρακτηριστικά.

2.5) Μετρήσιμα μεγέθη του λογισμικού

Αν και υπάρχει πληθώρα μεγεθών που μπορούν να χρησιμοποιηθούν για την αξιολόγηση του λογισμικού, συνήθως χρησιμοποιείται ένα πιο σαφώς ορισμένο υποσύνολό τους. Αυτό το υποσύνολο αποτελείται από 3 βασικές κατηγορίες μεγεθών: Εσωτερικά χαρακτηριστικά ως προς το μέγεθος, εσωτερικά χαρακτηριστικά ως προς τη δομή και εξωτερικά χαρακτηριστικά.

Εσωτερικά χαρακτηριστικά ως προς το μέγεθος (Internal Characteristics to size) :

1. **Μήκος κώδικα (code length) :** Η πιο απλή προσέγγιση είναι ο ορισμός του συνολικού αριθμού των γραμμών κώδικα ως μήκος του κώδικα. Όμως με αυτόν τον τρόπο προκύπτουν αρκετά προβλήματα. Βασικότερο αυτών είναι τι θεωρούμε γραμμή κώδικα, δηλαδή αν συνυπολογίζουμε τις γραμμές σχολίων, τις κενές γραμμές, τις δηλώσεις δεδομένων και τις γραμμές με πολλαπλές εντολές. Μια αποδεκτή λύση είναι να μετρώνται μόνο οι μη σχολιασμένες γραμμές κώδικα στο συνολικό αριθμό. Βέβαια και αυτή η λύση δημιουργεί με τη σειρά της προβλήματα, στην περίπτωση που το μέγεθος του κώδικα χρησιμοποιείται για τον καθορισμό του μεγέθους της μνήμης που απαιτείται για την αποθήκευση του κώδικα ή στην περίπτωση της εκτύπωσής του, όπου το πραγματικό μέγεθος είναι μεγαλύτερο από το μετρούμενο. Επίσης, δεν έχουν όλες οι γραμμές κώδικα την ίδια πολυπλοκότητα εκτέλεσης ή δυσκολία στην παραγωγή τους, δημιουργώντας άλλη μια παράμετρο που θα πρέπει να υπολογίζεται σε περιπτώσεις που η πολυπλοκότητα ή η απλότητα του κώδικα είναι κρίσιμη. Συνεπώς, ο σαφής ορισμός των γραμμών κώδικα και της μέτρησής τους αποτελεί σημαντικό εμπόδιο στη δημιουργία καθολικών και αποτελεσματικών μετρικών για τη σύγκριση μεταξύ λογισμικών.

2. **Επαναχρησιμοποίηση κώδικα (Code Reusability) :** Πλέον λόγω του τεράστιου αριθμού γραμμών και πολυπλοκότητας του κώδικα υπάρχει η ανάγκη ο νέος κώδικας που παράγεται να είναι επαναχρησιμοποιήσιμος. Αυτό επιτυγχάνεται είτε ενσωματώνοντας τις λειτουργικές μονάδες και τις διάφορες συναρτήσεις του σε βιβλιοθήκες συναρτήσεων είτε απλά χρησιμοποιώντας τις σε μελλοντικά και πιο σύνθετα κομμάτια λογισμικού. Όμως για να είναι επιτυχημένη η επαναχρησιμοποίηση του κώδικα είναι απαραίτητη η σωστή δόμηση, ο καλός σχολιασμός των επιμέρους μονάδων και κυρίως η σωστή λειτουργία του. Το αποτέλεσμα είναι μειωμένο κόστος παραγωγής λογισμικού, με τη χρησιμοποίηση υπομονάδων από άλλα λογισμικά, είτε ολόκληρα, είτε μέρος τους, είτε με αλλαγές και προσαρμογή.

3. **Πολυπλοκότητα κώδικα (Code Complexity) :** Η πολυπλοκότητα του κώδικα αποτελεί ένα πολύ σημαντικό χαρακτηριστικό, το οποίο σε συνδυασμό και με τα προηγούμενα μπορεί να επηρεάσει την ποιότητα του λογισμικού. Ιδανικά, θέλουμε η πολυπλοκότητα της υλοποίησής μας και του κώδικα να είναι ίδια με αυτήν του προβλήματος ή των στόχων που έχουμε. Όμως στην πράξη αυτό σπάνια συμβαίνει, λόγω περιορισμών της τεχνολογίας ή απρόβλεπτων παραγόντων που αυξάνουν την πολυπλοκότητα της υλοποίησης, είτε στο χρόνο (υπολογιστικό χρόνο), είτε στο χώρο (μνήμη του συστήματος) που απαιτείται για την επίτευξη των στόχων που έχουν τεθεί.

Εσωτερικά χαρακτηριστικά ως προς τη δομή (Internal Characteristics to structure) :

1. **Ροή ελέγχου (Control Flow) :** Μέσω της ροής ελέγχου εξετάζεται η ακολουθία εκτέλεσης εντολών, καθώς και πιθανών βρόχων ή αναδρομών, τα οποία δεν

συνυπολογίζονται στη μετρική μεγέθους για τον κώδικα. Αυτά τα στοιχεία μπορούν να αυξήσουν πάρα πολύ την πολυπλοκότητα και συνήθως χρησιμοποιούνται διαγράμματα ροής, μια μορφή κατευθυνόμενων γράφων, για τη γραφική αναπαράσταση και ευκολότερη ανάλυσή τους.

2. **Ροή δεδομένων (Data Flow) :** Μέσω της ροής δεδομένων η ανάλυση εστιάζεται στα δεδομένα που επεξεργάζεται ή δημιουργεί ο κώδικας, παρά στην ίδια την υλοποίηση του κώδικα. Δηλαδή, ανιχνεύεται η διαδρομή και η συμπεριφορά των δεδομένων από τη δημιουργία τους, καθώς και ο τρόπος που αυτά επεξεργάζονται και αλληλεπιδρούν με το πρόγραμμα. Λόγω της εστίασης στα δεδομένα, αυτό το είδος ανάλυσης χρησιμοποιείται στον οντοκεντρικό προγραμματισμό.

3. **Δομή δεδομένων (Data Structure) :** Η δομή δεδομένων εστιάζει στην οργάνωση των δεδομένων, ανεξάρτητα από το πρόγραμμα. Όταν τα δεδομένα είναι οργανωμένα σαν λίστες, ουρές, στοίβες ή άλλες καλά ορισμένες δομές δεδομένων, τότε και οι αλγόριθμοι για τη δημιουργία, τροποποίηση ή διαγραφή τους είναι πιθανώς καλά ορισμένοι. Συνεπώς, η δομή των δεδομένων προσφέρει πολλές πληροφορίες για την πρόβλεψη της πολυπλοκότητας δημιουργίας κώδικα για το χειρισμό τους καθώς και τον έλεγχο και δοκιμή της ορθής λειτουργίας τους.

Κεφάλαιο 3) Έλεγχος και Αποσφαλμάτωση Λογισμικού

3.1) Χρησιμότητα του ελέγχου και της αποσφαλμάτωσης

Ο έλεγχος του λογισμικού, κατά τη διάρκεια και μετά την παραγωγή του, είναι αναπόσπαστο κομμάτι της διαδικασίας ανάπτυξης λογισμικού. Είτε πρόκειται για εκτενείς δοκιμές με γενικού σκοπού μετροπρογράμματα, είτε πρόκειται για ειδικά σχεδιασμένες δοκιμές, είτε για εκτελέσεις του κώδικα κάτω από καθορισμένες συνθήκες, σε όλες τις περιπτώσεις ο έλεγχος είναι απαραίτητος. Η ανάγκη αυτή προκύπτει λόγω της ύπαρξης λαθών σε κάθε κομμάτι κώδικα. Σύμφωνα με στατιστικές μελέτες προκύπτει ότι ακόμα και προγράμματα για τα οποία έχουν ακολουθηθεί πιστά πρότυπα και προδιαγραφές εξασφάλισης της ποιότητας, περιέχουν 1-3 σφάλματα κάθε 100 γραμμές κώδικα. Βέβαια, ο σκοπός του ελέγχου είναι η πρόληψη των σφαλμάτων. Δυστυχώς όμως δεν είναι συνήθως εφικτό να επιτύχουμε αυτόν τον στόχο, με αποτέλεσμα να αρκούμαστε στην ανακάλυψη και διόρθωση όσο το δυνατό περισσότερων σφαλμάτων.

Η σχεδίαση των ελέγχων και η υλοποίησή τους είναι μια σύνθετη και δύσκολη διαδικασία. Σε πολλές περιπτώσεις ο έλεγχος πραγματοποιείται ανά τακτά διαστήματα κατά την ανάπτυξη του λογισμικού. Με αυτόν τον τρόπο είναι δυνατό να ελέγχονται τα επιμέρους κομμάτια κώδικα και λειτουργικές μονάδες με μικρότερο κόστος σε χρόνο και πόρους. Όμως, ακόμα και οι έλεγχοι που σχεδιάζονται για τη δοκιμή του λογισμικού πρέπει να παράγονται από αυστηρά καθορισμένες διαδικασίες, όμοιες με αυτές που εφαρμόζονται για την παραγωγή του λογισμικού. Σε διαφορετική περίπτωση ο έλεγχος αποτυγχάνει στον κύριο στόχο του, την ανίχνευση και κατά συνέπεια διόρθωση των σφαλμάτων, με αποτέλεσμα να σπαταλά πόρους άσκοπα.

3.2) Βασικές πτυχές και χαρακτηριστικά του ελέγχου

Ο έλεγχος λογισμικού είναι μια έρευνα που πραγματοποιείται, ώστε να παραχθούν ενδιαφέρουσες πληροφορίες σχετικά με την ποιότητα του προϊόντος ή της υπηρεσίας που ελέγχεται. Ο έλεγχος λογισμικού μπορεί επίσης να παράσχει μια αντικειμενική, ανεξάρτητη άποψη για το λογισμικό, επιτρέποντας τη σωστή εκτίμηση και κατανόηση των κινδύνων της εφαρμογής του λογισμικού. Οι τεχνικές ελέγχου περιλαμβάνουν τη διαδικασία της εκτέλεσης ενός προγράμματος ή μιας εφαρμογής, με σκοπό την ανεύρεση σφαλμάτων λογισμικού (λαθών ή άλλων ελαττωμάτων), αλλά δεν περιορίζονται σε αυτή. Ως έλεγχος λογισμικού μπορεί να οριστεί η διαδικασία επικύρωσης και επαλήθευσης των παρακάτω χαρακτηριστικών ενός προγράμματος, εφαρμογής ή άλλου προϊόντος υπολογιστή:

- Επιτυγχάνει τις προδιαγραφές που καθοδήγησαν το σχεδιασμό και την ανάπτυξή του.
- Λειτουργεί όπως αναμένεται.
- Μπορεί να εφαρμόζεται με τα ίδια χαρακτηριστικά.
- Ικανοποιεί τις ανάγκες των πελατών και των χρηστών.

Ο έλεγχος λογισμικού, ανάλογα με τη μέθοδο που χρησιμοποιείται, μπορεί να εφαρμοστεί σε οποιοδήποτε σημείο της διαδικασίας ανάπτυξης. Συνήθως, το μεγαλύτερο μέρος των διαδικασιών ελέγχου συμβαίνει μετά τον καθορισμό των προδιαγραφών και την ολοκλήρωση της συγγραφής του κώδικα. Διαφορετικά μοντέλα ανάπτυξης λογισμικού, εστιάζουν την προσπάθεια του ελέγχου σε διαφορετικά στάδια της διαδικασίας ανάπτυξης, ανάλογα με τα πρότυπα που ακολουθούν και τους στόχους που θέλουν να επιτύχουν. Πιο σύγχρονα μοντέλα ανάπτυξης, απαιτούν ανάπτυξη λογισμικού που καθορίζεται από τα αποτελέσματα του ελέγχου, αναθέτοντας το μεγαλύτερο μέρος του ελέγχου στον προγραμματιστή, προτού το προϊόν φτάσει στο στάδιο του «επίσημου» ελέγχου.

Ο έλεγχος λογισμικού δεν είναι ποτέ δυνατό να ταυτοποιήσει όλα τα ελαττώματα του λογισμικού. Αντιθέτως, παρέχει μια συσχέτιση ή σύγκριση η οποία συγκρίνει την κατάσταση και τη συμπεριφορά του προϊόντος ενάντια σε προφήτες (oracles), δηλαδή αρχές ή μηχανισμούς με τους οποίους κάποιος μπορεί να αναγνωρίσει ένα πρόβλημα. Οι προφήτες μπορεί να περιλαμβάνουν προδιαγραφές (specifications), συμφωνίες (contracts), συγκρίσιμα προϊόντα (comparable products), παλιές εκδόσεις του ίδιου προϊόντος (past versions), συμπεράσματα (inferences) σχετικά με αναμενόμενους ή μη σκοπούς, προσδοκίες των χρηστών (user) ή των πελατών (customer), σχετικά πρότυπα (relevant standards), εφαρμόσιμους νόμους (applicable laws) καθώς και πληθώρα άλλων κριτηρίων, ανάλογα με τη χρήση τους. Ο βασικός σκοπός του ελέγχου είναι η ανίχνευση και διόρθωση ελαττωμάτων στο λογισμικό. Ο έλεγχος δεν είναι δυνατό να τεκμηριώσει την ορθή λειτουργία του λογισμικού κάτω από οποιεσδήποτε συνθήκες, αντιθέτως μπορεί μόνο να τεκμηριώσει ότι το προϊόν δεν λειτουργεί ορθά κάτω από συγκεκριμένες συνθήκες.

3.3) Κατηγορίες ελέγχου

Στατικός και δυναμικός έλεγχος (static and dynamic testing)

Υπάρχουν αρκετές προσεγγίσεις σχετικά με τον έλεγχο λογισμικού. Οι αξιολογήσεις (reviews), περιδιαβάσεις (walkthroughs) και επιθεωρήσεις (inspections) αναφέρονται ως στατικός έλεγχος(static testing), ενώ η πραγματική εκτέλεση του κώδικα του προγράμματος, για ένα δοθέν σύνολο δοκιμαστικών περιπτώσεων, αναφέρεται ως δυναμικός έλεγχος (dynamic testing). Ο στατικός έλεγχος μπορεί να παραληφθεί, κάτι το οποίο συνήθως γίνεται στην πραγματικότητα. Ο δυναμικός έλεγχος λαμβάνει μέρος όταν το ίδιο το πρόγραμμα χρησιμοποιείται και μπορεί να ξεκινήσει ακόμα και αν το πρόγραμμα δεν είναι ολοκληρωμένο, ώστε να ελεγχθούν συγκεκριμένα τμήματα κώδικα και να εφαρμοστούν σε διακριτές συναρτήσεις ή λειτουργικές μονάδες. Οι συνηθισμένες τεχνικές είτε χρησιμοποιούν στελέχη/οδηγούς είτε εκτελούνται σε ένα περιβάλλον αποσφαλματωτή.

Η προσέγγιση του κουτιού (the box approach)

Οι μέθοδοι ελέγχου λογισμικού χωρίζονται παραδοσιακά σε ελέγχους άσπρου (white-box) και μαύρου (black-box) κουτιού. Αυτές οι δύο προσεγγίσεις χρησιμοποιούνται για να περιγράψουν την οπτική γωνία (point-of-view) την οποία ένας μηχανικός ελέγχου (test engineer) λαμβάνει υπόψη κατά το σχεδιασμό των σεναρίων ελέγχου.

Έλεγχος άσπρου κουτιού (White-box testing)

Αυτό το είδος ελέγχου είναι γνωστό και ως έλεγχος καθαρού κουτιού (clear-box testing), έλεγχος γυάλινου κουτιού (glass-box testing), έλεγχος διαφανούς κουτιού (transparent-box testing) και δομικός έλεγχος (structural testing) και ελέγχει τις εσωτερικές δομές ή λειτουργίες ενός προγράμματος, σε αντίθεση με τη λειτουργικότητα που παρουσιάζεται στον τελικό χρήστη. Στον έλεγχο άσπρου κουτιού χρησιμοποιείται μια εσωτερική προοπτική του συστήματος, καθώς και ικανότητες προγραμματισμού, για να σχεδιαστούν τα σενάρια ελέγχου. Ο ελεγκτής επιλέγει εισόδους για να δοκιμάσει μονοπάτια μέσα από τον κώδικα και καθορίζει τις κατάλληλες εξόδους. Ο παραπάνω τρόπος είναι σε αναλογία με τον έλεγχο κόμβων σε ένα κύκλωμα.

Ο έλεγχος άσπρου κουτιού μπορεί να εφαρμοστεί στα επίπεδα μονάδος (unit), ενσωμάτωσης (integration) και συστήματος (system) της διαδικασίας ελέγχου λογισμικού, ενώ συνήθως πραγματοποιείται στο επίπεδο μονάδος. Μπορεί να ελέγξει μονοπάτια μέσα σε μια μονάδα, μονοπάτια μεταξύ μονάδων κατά των ενσωμάτωση και μεταξύ υποσυστημάτων κατά τη διάρκεια ενός ελέγχου στο επίπεδο του συστήματος. Παρότι αυτή η μέθοδος σχεδιασμού ελέγχων μπορεί να ανακαλύψει αρκετά σφάλματα ή προβλήματα, είναι πιθανό να μην εντοπίσει μη εφαρμοσμένα τμήματα των προδιαγραφών ή προϋποθέσεις που λείπουν.

Οι τεχνικές που χρησιμοποιούνται στον έλεγχο άσπρου κουτιού περιλαμβάνουν:

- Έλεγχος προγραμματιστικής διεπαφής εφαρμογής (API application Programming Interface) : Έλεγχος της εφαρμογής χρησιμοποιώντας δημόσιες και ιδιωτικές προγραμματιστικές διεπαφές εφαρμογής (APIs).

- Κάλυψη του κώδικα (code coverage) : Δημιουργία ελέγχων για την ικανοποίηση ορισμένων κριτηρίων για την κάλυψη κώδικα, όπως η σχεδίαση ελέγχων έτσι ώστε όλες οι δηλώσεις (statements) στον κώδικα να εκτελεστούν τουλάχιστον μια φορά.

- Μέθοδοι έγχυσης σφαλμάτων (Fault injection methods) : Εισαγωγή σφαλμάτων σκόπιμα για τη μέτρηση της απόδοσης των στρατηγικών ελέγχου.

- Στατικές μέθοδοι ελέγχου (static testing methods).

- Μέθοδοι ελέγχου με μετάλλαξη (mutation testing methods).

Τα εργαλεία για την κάλυψη κώδικα μπορούν να αξιολογήσουν την πληρότητα ενός συνόλου ελέγχων, το οποίο δημιουργήθηκε με οποιαδήποτε μέθοδο, περιλαμβάνοντας και τον έλεγχο μαύρου κουτιού. Αυτό επιτρέπει στην ομάδα ανάπτυξης του λογισμικού να εξετάσει τμήματα ενός συστήματος τα οποία σπανίως δοκιμάζονται και διασφαλίζει ότι τα πιο σημαντικά λειτουργικά σημεία ελέγχονται. Η κάλυψη του κώδικα ως μετρική του λογισμικού αναφέρεται ως ένα ποσοστό (συνήθως επί τις εκατό) για τα ακόλουθα χαρακτηριστικά:

- Κάλυψη συναρτήσεων (Function coverage) : Το οποίο αναφέρεται στις συναρτήσεις που εκτελέστηκαν κατά τον έλεγχο.

- Κάλυψη δηλώσεων (Statement coverage) : Το οποίο αναφέρεται στο πλήθος των γραμμών κώδικα που εκτελέστηκαν για την ολοκλήρωση του ελέγχου.

Εάν υπάρχει 100% κάλυψη δηλώσεων, διασφαλίζεται ότι όλα τα μονοπάτια του κώδικα και οι διακλαδώσεις (όσον αφορά τον έλεγχο ροής) έχουν εκτελεστεί τουλάχιστον μια φορά. Αυτό είναι πολύ βοηθητικό για την εξασφάλιση της σωστής λειτουργικότητας, αλλά δεν είναι αρκετό διότι ο ίδιος κώδικας μπορεί να χειριστεί διαφορετικές εισόδους σωστά ή λανθασμένα.

Έλεγχος μαύρου κουτιού (Black-box testing)

Ο έλεγχος μαύρου κουτιού αντιμετωπίζει το λογισμικό ως «μαύρο κουτί», εξετάζοντας τη λειτουργικότητα χωρίς καμία γνώση της εσωτερικής υλοποίησης. Ο ελεγκτής γνωρίζει μόνο τι υποτίθεται ότι κάνει το λογισμικό, όχι πως το επιτυγχάνει. Οι μέθοδοι για τον έλεγχο μαύρου κουτιού περιλαμβάνουν ισοδύναμη διαμέριση (equivalent partitioning), ανάλυση συνοριακών τιμών (boundary value analysis), έλεγχο όλων των ζευγαριών (all-pair testing), πίνακες μετάβασης καταστάσεων (state transition tables), έλεγχο πινάκων απόφασης (decision table testing), έλεγχο βασισμένο στο μοντέλο (model-based testing), έλεγχο περιπτώσεων χρήσης (use case testing), διερευνητικό έλεγχο (exploratory testing) και έλεγχο βασισμένο στις προδιαγραφές (specification-based testing).

Ο έλεγχος βασισμένος στις προδιαγραφές στοχεύει στον έλεγχο της λειτουργικότητας του λογισμικού σύμφωνα με τις σχετικές προδιαγραφές. Αυτό το επίπεδο ελέγχου συνήθως απαιτεί την παροχή διεξοδικών σεναρίων ελέγχου στον ελεγκτή, ο οποίος ύστερα μπορεί απλά να επιβεβαιώσει ότι για κάποια δοθείσα είσοδο, η τιμή της εξόδου (η συμπεριφορά) είτε ‘είναι’ είτε ‘δεν είναι’ η ίδια με την αναμενόμενη τιμή που καθορίστηκε στο σενάριο ελέγχου. Τα σενάρια ελέγχου φτιάχνονται γύρω από προδιαγραφές και απαιτήσεις, όπως τι πρέπει να κάνει η εφαρμογή. Χρησιμοποιούνται εξωτερικές περιγραφές για το λογισμικό, περιλαμβάνοντας προδιαγραφές, απαιτήσεις και σχέδια για να παραχθούν τα σενάρια ελέγχου. Συνολικά, ο έλεγχος βασισμένος στις προδιαγραφές είναι απαραίτητος για τη διασφάλιση της σωστής λειτουργικότητας, αλλά δεν επαρκεί ως ασφάλεια ενάντια σε πολύπλοκες ή υψηλού κινδύνου καταστάσεις.

Ένα βασικό πλεονέκτημα του ελέγχου μαύρου κουτιού είναι ότι δεν απαιτούνται γνώσεις προγραμματισμού. Οποιεσδήποτε προκαταλήψεις και αν είχε ο προγραμματιστής, ο ελεγκτής πιθανώς έχει διαφορετικές και ίσως εστιάσει σε διαφορετικές περιοχές της

λειτουργικότητας. Στον αντίποδα, ο έλεγχος μαύρου κουτιού μπορεί να χαρακτηριστεί ως «περίπατος σε ένα σκοτεινό λαβύρινθο χωρίς φακό», διότι δεν εξετάζει τον πηγαίο κώδικα, με αποτέλεσμα να υπάρχουν περιπτώσεις όπου ο ελεγκτής δημιουργεί πολλαπλά σενάρια ελέγχου για τη δοκιμή ενός χαρακτηριστικού για το οποίο θα επαρκούσε μόνο ένα ή αφήνει μέρη του προγράμματος χωρίς έλεγχο.

Αυτή η μέθοδος ελέγχου μπορεί να εφαρμοστεί σε όλα τα επίπεδα ελέγχου: μονάδος (unit), ενσωμάτωσης (integration), συστήματος (system) και αποδοχής (acceptance). Συνήθως απαρτίζεται κυρίως, αν όχι εξ' ολοκλήρου, από ελέγχους σε υψηλά επίπεδα, αλλά μπορεί να κυριαρχεί και στον έλεγχο μονάδων.

Έλεγχος γκρίζου κουτιού (Grey-box testing)

Ο έλεγχος γκρίζου κουτιού περιλαμβάνει την ύπαρξη γνώσης για τις εσωτερικές δομές δεδομένων και τους αλγορίθμους για το σκοπό της σχεδίασης ελέγχων, ενώ η εκτέλεση αυτών των ελέγχων από το χρήστη γίνεται σε επίπεδο μαύρου κουτιού. Ο ελεγκτής δεν απαιτείται να έχει πλήρη πρόσβαση στον πηγαίο κώδικα του λογισμικού. Η μεταχείριση δεδομένων εισόδου και η μορφοποίηση της εξόδου δεν πληρούν τις προϋποθέσεις για γκρίζο κουτί, επειδή η είσοδος και η έξοδος είναι καθαρά εκτός ενός μαύρου κουτιού που καλούμε ως σύστημα υπό έλεγχο. Αυτή η διάκριση είναι ιδιαίτερα σημαντική όταν διεξάγουμε έλεγχο ενσωμάτωσης μεταξύ δυο τμημάτων κώδικα γραμμένων από διαφορετικούς προγραμματιστές, όπου μόνο οι διεπαφές είναι διαθέσιμες προς έλεγχο. Ωστόσο, η μορφοποίηση μιας αποθήκης δεδομένων (data repository) πληροί τις προϋποθέσεις για γκρίζο κουτί, επειδή ο χρήστης δεν θα μπορούσε υπό κανονικές συνθήκες να τροποποιήσει δεδομένα εκτός του συστήματος υπό έλεγχο. Ο έλεγχος μαύρου κουτιού μπορεί να περιλαμβάνει ακόμα αντίστροφη μηχανική (reverse engineering) για τον καθορισμό, για παράδειγμα, συνοριακών τιμών ή μηνυμάτων σφάλματος.

Με τη γνώση των βασικών εννοιών που διέπουν τη λειτουργία του λογισμικού, ο ελεγκτής μπορεί να κάνει καλύτερα ενημερωμένες επιλογές για τον έλεγχο, καθώς εκτελεί τον έλεγχο εξωτερικά του συστήματος. Συνήθως, επιτρέπεται σε ένα ελεγκτή γκρίζου κουτιού να εγκαθιδρύσει το περιβάλλον ελέγχου που θα χρησιμοποιήσει, όπως η δημιουργία μιας βάσης δεδομένων, και μέσα από το οποίο μπορεί να παρατηρεί το προϊόν να ελέγχεται, πραγματοποιώντας συγκεκριμένες ενέργειες σε αυτό. Ο έλεγχος γκρίζου κουτιού εφαρμόζει έξυπνα σενάρια ελέγχου, βασισμένος σε περιορισμένες πληροφορίες. Αυτό εφαρμόζεται ιδιαίτερα σε χειρισμό τύπων δεδομένων (data type handling) και χειρισμό εξαιρέσεων (exception handling).

Οπτικός έλεγχος (Visual testing)

Ο στόχος του οπτικού ελέγχου είναι να προσφέρει στους προγραμματιστές την ικανότητα να εξετάσουν τι συμβαίνει στο σημείο της αποτυχίας του λογισμικού, παρουσιάζοντας τα δεδομένα με τέτοιο τρόπο ώστε ο προγραμματιστής μπορεί να εντοπίσει εύκολα την πληροφορία που χρειάζεται και η οποία εκφράζεται καθαρά. Η βασική ιδέα του οπτικού ελέγχου είναι ότι η οπτική παρουσίαση ενός προβλήματος (μιας αποτυχίας) αυξάνει σημαντικά τη σαφήνεια (clarity) και την κατανόηση (understanding) του, σε σχέση με την απλή περιγραφή του προβλήματος. Ο οπτικός έλεγχος επομένως απαιτεί την καταγραφή ολόκληρης της διαδικασίας ελέγχου, συλλαμβάνοντας οτιδήποτε συμβαίνει στο σύστημα ελέγχου σε μορφή βίντεο. Τα βίντεο εξόδου συμπληρώνονται από πραγματικού χρόνου

είσοδο από τον ελεγκτή, μέσω εικόνας μέσα σε εικόνα (picture-in-a-picture) από βιντεοκάμερα και ηχητικού σχολιασμού από μικρόφωνο.

Ο οπτικός έλεγχος προσφέρει έναν αριθμό πλεονεκτημάτων. Η ποιότητα της επικοινωνίας αυξάνει δραματικά επειδή οι ελεγκτές μπορούν να δείξουν ένα πρόβλημα (καθώς και τα γεγονότα που οδηγούν σε αυτό) στους προγραμματιστές, αντί απλά να το περιγράφουν και η ανάγκη για αναπαραγωγή των αποτυχιών του ελέγχου παύει στις περισσότερες περιπτώσεις. Ο προγραμματιστής έχει συνεπώς όλα τα αποδεικτικά στοιχεία που του χρειάζονται από μια αποτυχία ελέγχου και μπορεί να εστιάσει στην αιτία του σφάλματος και της διόρθωσής του. Ο οπτικός έλεγχος είναι κατάλληλος για περιβάλλοντα που εφαρμόζουν ευέλικτες μεθόδους για την ανάπτυξη του λογισμικού, εφόσον οι ευέλικτες μέθοδοι απαιτούν μεγαλύτερη επικοινωνία μεταξύ των ελεγκτών και των προγραμματιστών και συνεργασία εντός μικρών ομάδων.

Οι εξειδικευμένοι έλεγχοι (ad hoc testing) και οι διερευνητικοί έλεγχοι (exploratory testing) είναι σημαντικές μεθοδολογίες για τον έλεγχο της συνοχής του λογισμικού, επειδή απαιτούν λιγότερο χρόνο προετοιμασίας για να ενσωματωθούν, ενώ τα σημαντικά σφάλματα μπορούν να ταυτοποιηθούν γρήγορα. Στον εξειδικευμένο έλεγχο, όπου ο έλεγχος εκτελείται με ένα τρόπο αυθόρμητο και με αυτοσχεδιασμό, η ικανότητα ενός εργαλείου ελέγχου να καταγράφει οπτικά οτιδήποτε συμβαίνει στο σύστημα γίνεται πολύ σημαντική.

Ο οπτικός έλεγχος συγκεντρώνει αναγνώριση στην αποδοχή των πελατών και στον έλεγχο χρηστικότητας, επειδή μπορεί να χρησιμοποιηθεί από πολλούς διαφορετικούς χρήστες που εμπλέκονται στη διαδικασία ανάπτυξης. Για τον πελάτη γίνεται εύκολο να παρέχει αναφορές σφαλμάτων και πληροφορίες ως ανάδραση και για τους χρήστες του προγράμματος, ο οπτικός έλεγχος μπορεί να καταγράφει τις ενέργειές τους στην οθόνη, καθώς και τη φωνή και την εικόνα τους, ώστε να παρέχουν ολοκληρωμένη εικόνα τη στιγμή της αποτυχίας του λογισμικού στους προγραμματιστές.

Επίπεδα ελέγχου (Testing levels)

Έλεγχος μονάδων (Unit testing)

Ο έλεγχος μονάδων (ή έλεγχος συστατικών μερών) αναφέρεται σε ελέγχους που επαληθεύουν τη λειτουργικότητα ενός συγκεκριμένου τμήματος κώδικα, συνήθως σε επίπεδο λειτουργίας. Σε οντοκεντρικά περιβάλλοντα αυτό αναφέρεται σε επίπεδο κλάσεων και οι ελάχιστες μονάδες ελέγχου περιλαμβάνουν τους κατασκευαστές (constructors) και τους καταστροφείς (destructors). Αυτοί οι τύποι ελέγχων συνήθως δημιουργούνται από τους προγραμματιστές καθώς εργάζονται πάνω στον κώδικα (με τρόπο άσπρου κουτιού) για να διασφαλίσουν ότι η συγκεκριμένη λειτουργία δουλεύει όπως αναμένεται. Μια λειτουργία μπορεί να περιλαμβάνει αρκετούς ελέγχους, για να πιάσει ακραίες περιπτώσεις ή διακλαδώσεις στον κώδικα. Από μόνος του ο έλεγχος μονάδων δεν μπορεί να επαληθεύσει τη λειτουργικότητα ενός τμήματος λογισμικού, αλλά αντίθετα χρησιμοποιείται για να διασφαλίσει ότι τα συστατικά τμήματα που χρησιμοποιεί το λογισμικό λειτουργούν ανεξάρτητα μεταξύ τους.

Έλεγχος ενσωμάτωσης (Integration testing)

Ο έλεγχος ενσωμάτωσης περιλαμβάνει οποιονδήποτε τύπο ελέγχου λογισμικού ο οποίος αναζητά την επαλήθευση των διεπαφών μεταξύ των συστατικών μερών ενάντια στο

σχεδιασμό του λογισμικού. Τα συστατικά μέρη του λογισμικού μπορούν να ενσωματωθούν με ένα επαναληπτικό τρόπο ή όλα μαζί. Συνήθως το προηγούμενο θεωρείται καλύτερη πρακτική, επειδή επιτρέπει τα προβλήματα των διεπαφών να εντοπίζονται πιο γρήγορα και να διορθώνονται. Ο έλεγχος ενσωμάτωσης έχει σκοπό την αποκάλυψη ελαττωμάτων στις διεπαφές και την αλληλεπίδραση μεταξύ ενσωματωμένων συστατικών μερών. Προοδευτικά μεγαλύτερες ομάδες από ελεγμένα συστατικά μέρη του λογισμικού, που αναλογούν σε στοιχεία της αρχιτεκτονικής σχεδίασης, ενσωματώνονται και ελέγχονται μέχρι το λογισμικό να λειτουργεί ως σύστημα.

Έλεγχος συστήματος (System testing)

Ο έλεγχος συστήματος πραγματοποιείται σε εξ' ολοκλήρου ενσωματωμένα συστήματα για να επαληθεύσει ότι πληρούν τις προδιαγραφές τους.

Έλεγχος αποδοχής (Acceptance testing)

Το πλήρως ελεγμένο σύστημα παραδίδεται στο χρήστη/πελάτη για έλεγχο αποδοχής.

Προσέγγιση ελέγχου (Testing approach)

Από κάτω προς τα πάνω (Bottom-up)

Ο έλεγχος από κάτω προς τα πάνω είναι μια προσέγγιση για ενσωματωμένο έλεγχο, όπου τα χαμηλότερα επίπεδα συστατικών μερών ελέγχονται πρώτα και μετά χρησιμοποιούνται για να διευκολύνουν τον έλεγχο υψηλότερων επιπέδων συστατικών μερών. Η διαδικασία επαναλαμβάνεται μέχρι να ελεγχθεί η κορυφή της ιεραρχίας. Όλα τα χαμηλού επιπέδου τμήματα, διαδικασίες ή συναρτήσεις ενσωματώνονται και ύστερα ελέγχονται. Μετά την ολοκλήρωση της ενσωμάτωσης ενός χαμηλότερου επιπέδου συστατικού, τα επόμενα στην ιεραρχία τμήματα σχηματίζονται και ενσωματώνονται για έλεγχο. Αυτή η προσέγγιση είναι χρήσιμη μόνο όταν τα περισσότερα ή όλα τα συστατικά μέρη του ίδιου επιπέδου είναι έτοιμα. Αυτή η μέθοδος βοηθάει επίσης στον καθορισμό των επιπέδων του λογισμικού που αναπτύσσονται και κάνει ευκολότερη την αναφορά της προόδου του ελέγχου με τη μορφή ποσοστού.

Από πάνω προς τα κάτω (Top-down)

Ο έλεγχος από πάνω προς τα κάτω είναι μια προσέγγιση για ενσωματωμένο έλεγχο, όπου τα υψηλότερου επιπέδου συστατικά μέρη ελέγχονται και μετά ελέγχονται οι διακλαδώσεις τους βήμα βήμα, μέχρι το τέλος των τμημάτων που αυτά σχετίζονται.

Στόχοι ελέγχου (Objectives of testing)

Έλεγχος εγκατάστασης (Installation testing)

Ο έλεγχος εγκατάστασης διασφαλίζει ότι το λογισμικό εγκαταστάθηκε σωστά και λειτουργεί στον εξοπλισμό(hardware) του πελάτη.

Έλεγχος συμβατότητας (Compatibility testing)

Μια κοινή αιτία για αποτυχίες λογισμικού είναι η έλλειψη συμβατότητας με λογισμικό άλλων εφαρμογών, λειτουργικά συστήματα (operating systems) (με παλιότερες ή νεώτερες εκδόσεις λειτουργικών συστημάτων) καθώς και περιβάλλοντα εκτέλεσης που διαφέρουν σημαντικά από το αρχικό, όπως η κονσόλα εντολών (terminal) ή η γραφική διεπαφή χρήστη (Graphical User Interface) να στοχεύουν σε εκτέλεση στην επιφάνεια εργασίας και να απαιτείται να μετατραπούν σε δικτυακή εφαρμογή, η οποία απεικονίζεται (render) από ένα περιηγητή ιστού (web browser). Επίσης, η προς τα πίσω συμβατότητα (backward compatibility) μπορεί να μην εξασφαλίζεται αν το λογισμικό αναπτύσσεται και ελέγχεται μόνο με βάση την τελευταία έκδοση το περιβάλλοντος στον οποίο στοχεύει, το οποίο δεν θα χρησιμοποιείται καθολικά από τους χρήστες. Το αποτέλεσμα είναι μη αναμενόμενες και απροσδόκητες επιπτώσεις σε νεώτερες εκδόσεις του περιβάλλοντος στόχου ή σε περιβάλλοντα με παλιό εξοπλισμό (hardware). Μερικές φορές αυτά τα ζητήματα αντιμετωπίζονται προληπτικά κάνοντας τη λειτουργικότητα των λειτουργικών συστημάτων αφαιρετική και τοποθετώντας την σε βιβλιοθήκες (library) ή ξεχωριστά τμήματα κώδικα.

Έλεγχος καπνού και λογικής (Smoke and sanity testing)

Ο έλεγχος λογικής καθορίζει εάν είναι λογικό να πραγματοποιηθούν περαιτέρω έλεγχοι.

Ο έλεγχος καπνού χρησιμοποιείται για να καθορίσει εάν υπάρχουν σοβαρά προβλήματα με ένα τμήμα λογισμικού.

Έλεγχος οπισθοδρόμησης (Regression testing)

Ο έλεγχος οπισθοδρόμησης εστιάζει στον εντοπισμό ελαττωμάτων ύστερα από μια μεγάλη αλλαγή στον κώδικα. Ειδικότερα, αναζητά να αποκαλύψει οπισθοδρομήσεις στο λογισμικό ή παλιά σφάλματα τα οποία επανήλθαν. Τέτοια σφάλματα συμβαίνουν όταν η λειτουργικότητα του λογισμικού, η οποία προηγουμένως δούλευε σωστά σταματά να λειτουργεί όπως αναμένεται. Τυπικά οι οπισθοδρομήσεις συμβαίνουν ως μη επιθυμητές επιπτώσεις των αλλαγών στον κώδικα και ειδικότερα όταν ένα νέο κομμάτι κώδικα συγκρούεται με ένα παλιότερο. Κοινές μέθοδοι για έλεγχο οπισθοδρόμησης περιλαμβάνουν επανεκτέλεση ελέγχων που έχουν εκτελεστεί στο παρελθόν και έλεγχο για την επανεμφάνιση παλιότερων σφαλμάτων. Το βάθος του ελέγχου εξαρτάται από το στάδιο ανάπτυξης και τον κίνδυνο των επιπρόσθετων λειτουργιών. Ο έλεγχος μπορεί να είναι από πλήρης, για αλλαγές που προστέθηκαν αργά κατά την ανάπτυξη ή κρίθηκαν ως επικίνδυνες, έως πολύ ρηχός, για αλλαγές που έγιναν νωρίς ή κρίθηκαν ακίνδυνες.

Έλεγχος αποδοχής (Acceptance testing)

Ο έλεγχος αποδοχής αναφέρεται σε δυο πράγματα:

1. Ένας έλεγχος καπνού χρησιμοποιείται ως έλεγχος αποδοχής προτού παρουσιαστεί μια νέα φάση στη βασική διαδικασία ελέγχου, όπως για παράδειγμα πριν την ενσωμάτωση ή την οπισθοδρόμηση.

2. Έλεγχος αποδοχής που πραγματοποιείται από τον πελάτη, συνήθως στο δοκιμαστικό περιβάλλον και με το δικό του εξοπλισμό. Ο έλεγχος αποδοχής μπορεί να εκτελείται και ως μέρος της διαδικασίας μεταφοράς μεταξύ των φάσεων ανάπτυξης.

Πρώτη δοκιμή (Alpha testing)

Η πρώτη δοκιμή εκτελείται είτε σε πραγματικό περιβάλλον ελέγχου από πιθανούς πελάτες/χρήστες είτε από ανεξάρτητους ελεγκτές στο περιβάλλον του προγραμματιστή. Η πρώτη δοκιμή συνήθως εφαρμόζεται σε έτοιμο λογισμικό, ως μορφή εσωτερικού ελέγχου αποδοχής και πριν περάσει στη δεύτερη δοκιμή.

Δοκιμαστικός έλεγχος (Beta testing)

Ο δοκιμαστικός έλεγχος ακολουθεί την πρώτη δοκιμή και μπορεί να θεωρηθεί ως μια μορφή εξωτερικού ελέγχου αποδοχής από τους χρήστες. Εκδόσεις του λογισμικού, γνωστές ως δοκιμαστικές εκδόσεις (beta versions), γίνονται διαθέσιμες σε περιορισμένο κοινό εκτός της προγραμματιστικής ομάδας. Το λογισμικό γίνεται διαθέσιμο σε άτομα εκτός της προγραμματιστικής ομάδας, ώστε να πραγματοποιηθούν περαιτέρω έλεγχοι και να διασφαλιστεί ότι το προϊόν έχει λίγα σφάλματα και ελαττώματα. Μερικές φορές οι δοκιμαστικές εκδόσεις είναι διαθέσιμες σε όλους (open beta testing) με σκοπό την αύξηση της ανταπόκρισης και των αποτελεσμάτων του ελέγχου από όσο το δυνατόν περισσότερους μελλοντικούς χρήστες.

Έλεγχος λειτουργικότητας (Functional and non-functional testing)

Ο έλεγχος λειτουργικότητας αναφέρεται σε δραστηριότητες που έχουν σκοπό την επικύρωση μιας συγκεκριμένης ενέργειας ή λειτουργίας στον κώδικα. Συνήθως συναντώνται στα έγγραφα που περιέχουν τις προδιαγραφές του κώδικα. Ο στόχος του ελέγχου λειτουργικότητας είναι η απάντηση σε ερωτήσεις με συγκεκριμένες λειτουργίες, προδιαγραφές και δυνατότητες που θα πρέπει να έχει το λογισμικό.

Ο έλεγχος μη λειτουργικότητας αναφέρεται σε πλευρές του λογισμικού που δεν αναφέρονται σε συγκεκριμένες συναρτήσεις ή ενέργειες του χρήστη, όπως η κλιμακοσιμότητα (scalability) ή την απόδοση, η συμπεριφορά κάτω από συγκεκριμένους περιορισμούς ή θέματα ασφάλειας. Ο έλεγχος θα καθορίσει τα σημεία στα οποία δεν υπάρχει συνοχή, δηλαδή τα σημεία στα οποία ακραίες επιλογές για την κλιμακοσιμότητα ή την απόδοση θα προκαλέσουν αστάθεια στην εκτέλεση του συστήματος. Οι μη λειτουργικές προδιαγραφές τείνουν να είναι αυτές που αντανακλούν την ποιότητα του προϊόντος, ειδικά στο ευρύτερο πλαίσιο της καταλληλότητας από την οπτική γωνία των χρηστών.

Καταστροφικός έλεγχος (Destruction testing)

Ο καταστροφικός έλεγχος επιχειρεί να προκαλέσει την αποτυχία του λογισμικού ή κάποιου υποσυστήματος. Επιβεβαιώνει ότι το λογισμικό λειτουργεί σωστά, ακόμα και όταν δέχεται άκυρες ή μη αναμενόμενες εισόδους, με συνέπεια να εγκαθιστά την ευρωστία των ρουτινών επικύρωσης εισόδου και διαχείρισης σφαλμάτων. Η εισαγωγή σφαλμάτων στο

λογισμικό είναι ένα παράδειγμα ελέγχου αποτυχιών. Διάφορα εμπορικά εργαλεία ελέγχου μη λειτουργικότητας συνδέονται από τη σε σελίδα εισαγωγής σφαλμάτων, ενώ υπάρχει πληθώρα διαθέσιμων δωρεάν και ανοιχτού κώδικα εργαλείων τα οποία πραγματοποιούν καταστροφικό έλεγχο.

Έλεγχος απόδοσης λογισμικού (Software performance testing)

Ο έλεγχος ποιότητας λογισμικού γενικά εκτελείται για να καθοριστεί πως ένα σύστημα ή υποσύστημα αποδίδει σε θέματα ανταπόκρισης και σταθερότητας κάτω από συγκεκριμένο φόρτο εργασίας. Μπορεί επίσης να χρησιμοποιηθεί για τη διερεύνηση, μέτρηση, επικύρωση ή επιβεβαίωση άλλων χαρακτηριστικών της ποιότητας του συστήματος, όπως η κλιμακοσιμότητα, αξιοπιστία και χρησιμοποίηση πόρων.

Ο έλεγχος φόρτου κυρίως ασχολείται με ον έλεγχο της δυνατότητας του συστήματος να λειτουργεί κάτω από συγκεκριμένο φόρτο εργασίας, είτε πρόκειται για μεγάλες ποσότητες δεδομένων είτε για μεγάλες ποσότητες χρηστών. Αυτό το χαρακτηριστικό συνήθως αναφέρεται ως κλιμακοσιμότητα του λογισμικού. Η σχετική δραστηριότητα ελέγχου του φόρτου όταν εκτελείται ως μη λειτουργική δραστηριότητα συχνά αναφέρεται ως έλεγχος ανθεκτικότητας (endurance testing). Ο έλεγχος όγκου (volume testing) είναι ένας τρόπος ελέγχου συναρτήσεων λογισμικού ακόμα και αν μερικά τμήματα (όπως ένα αρχείο ή μια βάση δεδομένων) αυξάνονται ριζικά σε μέγεθος. Ο έλεγχος πίεσης (stress testing) είναι ένας τρόπος ελέγχου της αξιοπιστίας κάτω από μη αναμενόμενες ή σπάνιες περιπτώσεις φόρτου εργασίας. Ο έλεγχος σταθερότητας (stability testing) ελέγχει τη δυνατότητα του λογισμικού για συνεχόμενη ορθή λειτουργία κατά τη διάρκεια ή πέραν μιας αποδεκτής χρονικής περιόδου.

Υπάρχει μικρός βαθμός σύγκλισης στους συγκεκριμένους στόχους του ελέγχου απόδοσης. Οι όροι έλεγχος φόρτου (load testing), έλεγχος απόδοσης (performance testing), έλεγχος αξιοπιστίας (reliability testing) και έλεγχος όγκου (volume testing) συχνά χρησιμοποιούνται εναλλακτικά μεταξύ τους.

Έλεγχος χρηστικότητας (Usability testing)

Ο έλεγχος χρηστικότητας χρειάζεται για να ελεγχθεί αν οι διεπαφές χρήστη είναι εύκολες στη χρήση και κατανόηση. Ασχολείται κυρίως με τη χρηστικότητα της εφαρμογής από την πλευρά των χρηστών.

Έλεγχος προσβασιμότητας (Accessibility testing)

Ο έλεγχος προσβασιμότητας περιλαμβάνει τη συμμόρφωση με πρότυπα προσβασιμότητας όπως:

- Americans with Disabilities Act of 1990
- Section 508 Amendment to the Rehabilitation Act of 1973
- Web Accessibility Initiative(WAI) of the World Wide Web Consortium(W3C)

Έλεγχος ασφάλειας (Security testing)

Ο έλεγχος ασφάλειας είναι ουσιώδης για λογισμικό που χειρίζεται προσωπικά δεδομένα, ώστε να αποτρέπει την εισβολή από ηλεκτρονικούς πειρατές (hackers).

Έλεγχος διεθνοποίησης και αποκέντρωσης (Internationalization and localization testing)

Η γενικότερη ικανότητα του λογισμικού να διεθνοποιείται (internationalized) και να αποκεντρώνεται (localized) μπορεί να ελεγχθεί άμεσα χωρίς να χρειάζεται να γίνει ουσιαστική μετάφραση, μέσω της ψευδό-αποκέντρωσης (pseudo localization). Μέσω της τελευταίας θα επικυρωθεί ότι η εφαρμογή λειτουργεί, ακόμα και αν μεταφραστεί σε κάποια άλλη γλώσσα ή προσαρμοστεί σε κάποια καινούργια κουλτούρα, όπως διαφορετικό νόμισμα (currency) ή ζώνη ώρας (time zone).

Η ουσιαστική μετάφραση σε άλλες ανθρώπινες γλώσσες θα πρέπει να ελεγχθεί επίσης. Πιθανές αιτίες αποτυχίας της αποκέντρωσης είναι οι παρακάτω:

- Το λογισμικό συνήθως αποκεντρώνεται μεταφράζοντας μια λίστα συμβολοσειρών χωρίς να λαμβάνονται υπόψη τα συμφραζόμενα (out of context) με αποτέλεσμα να είναι πιθανό ο μεταφραστής να επιλέξει λανθασμένη μετάφραση για μια διφορούμενη συμβολοσειρά προς μετάφραση.

- Η τεχνική ορολογία μπορεί να αποκτήσει αντιφάσεις, εάν η μετάφραση πραγματοποιηθεί από αρκετούς ανθρώπους, χωρίς σωστό συντονισμό τους ή ο μεταφραστής λειτουργεί αλόγιστα (imprudent).

- Κυριολεκτικές λέξη προς λέξη μεταφράσεις μπορεί να ακούγονται ακατάλληλες (inappropriate), τεχνητές (artificial) ή πολύ τεχνικές (technical) στην γλώσσα στόχο της μετάφρασης.

- Μηνύματα που δεν έχουν μεταφραστεί από αρχική γλώσσα είναι πιθανό να μην μπορούν να αλλαχθούν στον πηγαίο κώδικα.

- Ορισμένα μηνύματα μπορεί να δημιουργούνται αυτόματα κατά το χρόνο εκτέλεσης και οι συμβολοσειρές που προκύπτουν είναι πιθανό να είναι λάθος με βάση τη γραμματική (ungrammatical), λανθασμένα λειτουργικά (functionally incorrect), παραπλανητικά (misleading) ή παράξενα (confusing).

- Το λογισμικό μπορεί να χρησιμοποιεί μια συντόμευση του πληκτρολογίου η οποία δεν έχει καμία λειτουργία στην διάταξη πληκτρολογίου της αρχικής γλώσσας, αλλά χρησιμοποιείται για την πληκτρολόγηση χαρακτήρων στη διάταξη πληκτρολογίου της γλώσσας στόχου.

- Το λογισμικό μπορεί να μην παρέχει υποστήριξη για την κωδικοποίηση χαρακτήρων της γλώσσας στόχου.

- Η γραμματοσειρά και το μέγεθος γραμματοσειράς που είναι κατάλληλο για την αρχική γλώσσα, είναι πιθανό να είναι ακατάλληλο για την γλώσσα στόχο.

- Μια συμβολοσειρά στη γλώσσα στόχο μπορεί να είναι πιο μακριά από ότι μπορεί να χειριστεί το λογισμικό. Αυτό έχει ως συνέπεια η συμβολοσειρά να είναι μερικώς αόρατη προς το χρήστη ή μπορεί να προκαλέσει δυσλειτουργία ή ακόμα και κατάρρευση του λογισμικού.

- Το λογισμικό μπορεί να μην παρέχει σωστή υποστήριξη για την ανάγνωση ή εγγραφή αμφι-κατευθυνόμενου κειμένου.
- Το λογισμικό μπορεί να προβάλει εικόνες με κείμενο το οποίο δεν είχε αποκεντρωθεί/μεταφραστεί.
- Αποκεντρωμένα λειτουργικά συστήματα μπορεί να έχουν διαφορετικά ονομασμένα αρχεία ρυθμίσεων και περιβαλλοντικές μεταβλητές και διαφορετικές μορφοποιήσεις ημερομηνίες και νομίσματος.

Έλεγχος ανάπτυξης (Development testing)

Ο έλεγχος ανάπτυξης είναι μια διαδικασία ανάπτυξης λογισμικού η οποία περιλαμβάνει συγχρονισμένη εφαρμογή ενός ευρέως φάσματος πρόληψης ελαττωμάτων και στρατηγικών εντοπισμού, με σκοπό τη μείωση των κινδύνων, του χρόνου και του κόστους ανάπτυξης του λογισμικού. Εκτελείται από τον προγραμματιστή ή μηχανικό κατά τη φάση κατασκευής του κύκλου ζωής του λογισμικού. Αντί να αντικαθιστά τις παραδοσιακές εστίες ενδιαφέροντος της αξιολόγησης λογισμικού, τις επαυξάνει. Ο έλεγχος ανάπτυξης στοχεύει στην εξάλειψη των σφαλμάτων κατασκευής προτού το προϊόν φτάσει στη φάση αξιολόγησης, με σκοπό την αύξηση της ποιότητας του λογισμικού που προκύπτει όπως και της αποδοτικότητας και της συνολικής διαδικασίας ανάπτυξης και αξιολόγησης του λογισμικού. Ανάλογα με τις προσδοκίες για την ανάπτυξη του λογισμικού, ο έλεγχος ανάπτυξης μπορεί να περιλαμβάνει:

- Στατική ανάλυση κώδικα (static code analysis).
- Ανάλυση της ροής δεδομένων (data flow analysis).
- Ανάλυση μετρικών (metrics analysis).
- Ομότιμες κριτικές κώδικα (peer code reviews).
- Έλεγχο μονάδων (unit testing).
- Έλεγχο κάλυψης κώδικα (code coverage analysis).
- Δυνατότητα ανίχνευσης (traceability).

3.4) Βασικές πτυχές και χαρακτηριστικά της αποσφαλμάτωσης

Η αποσφαλμάτωση είναι η μεθοδική διαδικασία της εύρεσης και ελάττωσης του αριθμού των σφαλμάτων ή των ελαττωμάτων σε ένα πρόγραμμα υπολογιστή ή σε ένα κομμάτι ηλεκτρονικού υλικού με συνέπεια να το κάνει να λειτουργεί όπως αναμένεται. Η αποσφαλμάτωση τείνει να είναι πιο δύσκολη όταν τα διάφορα υποσυστήματα είναι στενά συνδυασμένα μεταξύ τους, επειδή οι αλλαγές στο ένα μπορεί να προκαλέσουν τη δημιουργία σφαλμάτων στο άλλο. Πολλές τεχνικές και στρατηγικές έχουν εφευρεθεί για να καλύψουν την πληθώρα απόψεων και πλευρών της αποσφαλμάτωσης, οι οποίες περιλαμβάνουν διαδραστική αποσφαλμάτωση (interactive debugging), έλεγχο ροής (control flow), έλεγχο ενσωμάτωσης (integration testing), αρχεία ημερολογίου (log files), παρακολούθηση εφαρμογής και συστήματος (monitoring application and system), σκουπιδότοπους μνήμης (memory dumps), καταγραφή χαρακτηριστικών (profiling), στατιστικός έλεγχος διαδικασίας (statistical process control) και ειδικές τακτικές σχεδίασης για την βελτίωση της ανίχνευσης ενώ απλοποιούνται οι αλλαγές.

Τεχνικές αποσφαλμάτωσης

- Αποσφαλμάτωση με ανίχνευση (trace debugging) είναι η ενέργεια της παρακολούθησης (ζωντανών ή μαγνητοσκοπημένων) δηλώσεων ανίχνευσης (trace statements) που υποδεικνύουν τη ροή εκτέλεσης μιας διαδικασίας. Μερικές φορές αναφέρεται και ως αποσφαλμάτωση εκτύπωσης (printf debugging) λόγω της χρήσης της δήλωσης printf στη γλώσσα C.

- Απομακρυσμένη αποσφαλμάτωση (Remote debugging) είναι η διαδικασία της αποσφαλμάτωσης ενός προγράμματος που εκτελείται σε σύστημα διαφορετικό από αυτό του αποσφαλματωτή. Για να ξεκινήσει η απομακρυσμένη διαδικασία, ο αποσφαλματωτής συνδέεται με ένα απομακρυσμένο σύστημα μέσω ενός δικτύου. Όταν έχει επιτευχθεί η σύνδεση, ο αποσφαλματωτής μπορεί να ελέγξει την εκτέλεση του προγράμματος και να ανακτήσει δεδομένα αναφορικά με την κατάσταση του συστήματος.

- Αποσφαλμάτωση μετά από τη νεκροψία (post mortem debugging) είναι η διαδικασία αποσφαλμάτωσης σε ένα πρόγραμμα αφού αυτού έχει καταρρεύσει. Σχετικές τεχνικές περιλαμβάνουν διάφορες τεχνικές ανίχνευσης και ανάλυσης της χωματερής της μνήμης (memory dump) της διαδικασίας που κατέρρευσε. Η χωματερή της διαδικασίας μπορεί να αποκτηθεί αυτόματα από το σύστημα, από μια εντολή εισαγμένη από τον προγραμματιστή ή χειρονακτικά από των χρήστη που αλληλεπιδρά.

- Αποσφαλμάτωση Δέλτα (delta debugging) είναι μια τεχνική για αυτοματοποίηση της απλοποίησης των περιπτώσεων ελέγχου.

Αποσφαλμάτωση στα ενσωματωμένα συστήματα (Debugging for embedded systems)

Σε αντίθεση με το γενικού σκοπού περιβάλλον σχεδίασης λογισμικού υπολογιστών, ένα κύριο χαρακτηριστικό των ενσωματωμένων περιβαλλόντων είναι ο καθαρός αριθμός των διαφορετικών πλατφορμών που είναι διαθέσιμες στους προγραμματιστές, όπως αρχιτεκτονικές ΚΜΕ (Κεντρική Μονάδα Επεξεργασίας) (CPU architectures), πωλητές (vendors) και λειτουργικά συστήματα και οι παραλλαγές τους (operating systems and their

variants). Τα ενσωματωμένα συστήματα έχουν από τον ορισμό τους μη γενικού σκοπού σχεδιασμό, είναι σχεδιασμένα για ένα μοναδικό σκοπό και η πλατφόρμα επιλέγεται συγκεκριμένα για να βελτιστοποιήσει την απόδοση της εφαρμογής. Βέβαια αυτό δυσκολεύει ιδιαίτερα τους προγραμματιστές των ενσωματωμένων περιβαλλόντων, καθώς και την αποσφαλμάτωση και τον έλεγχο αυτών των συστημάτων, επειδή διαφορετικά εργαλεία απαιτούνται για την αποσφαλμάτωση διαφορετικών πλατφορμών.

Αντί Αποσφαλμάτωση (Anti-debugging)

Η αντί αποσφαλμάτωση είναι η εφαρμογή ενός ή περισσοτέρων τεχνικών μέσα στον κώδικα του λογισμικού, η οποίες εμποδίζουν απόπειρες για αντίστροφη μηχανική (reverse engineering) ή για αποσφαλμάτωση μιας συγκεκριμένης διαδικασίας. Χρησιμοποιείται ενεργά σε νόμιμα σχήματα προστασίας αντιγραφής (copy protection) ενώ χρησιμοποιείται επίσης από κακόβουλα λογισμικά (malware) για να περιπλέξει την ανίχνευση και εξολόθρευσή τους. Οι τεχνικές που περιλαμβάνονται στην αντί αποσφαλμάτωση είναι:

- Τεχνική βασισμένη σε Προγραμματιστική Διεπαφή Εφαρμογής (API based) : Ελέγχει για την ύπαρξη ενός αποσφαλματωτή χρησιμοποιώντας τις πληροφορίες του συστήματος.
- Τεχνική βασισμένη σε εξαιρέσεις (Exception based) : Ελέγχει για τυχόν παρεμβάσεις στις εξαιρέσεις.
- Κομμάτια διαδικασιών και νημάτων (Process and thread blocks) : Ελέγχει εάν τα κομμάτια διαδικασιών και νημάτων έχουν υποστεί μεταχείριση (manipulated).
- Τροποποιημένος κώδικας (Modified code) : Ελέγχει για τροποποιήσεις του κώδικα από ένα αποσφαλματωτή που χειρίζεται τα όρια ευαισθησίας του λογισμικού (software breakpoints).
- Τεχνική βασισμένη στο υλικό και τους καταχωρητές (Hardware and register based) : Ελέγχει για τα όρια ευαισθησίας του υλικού και τους καταχωρητές της ΚΜΕ.
- Χρονισμός και λανθάνουσα περίοδος (Timing and latency) : Ελέγχει τον χρόνο που παρήλθε για την εκτέλεση των εντολών.
- Ανίχνευση και τιμωρία αποσφαλματωτών (Detection and penalizing debuggers).

3.5) Κατηγορίες σφαλμάτων (Bugs)

Ο όρος σφάλμα αναφέρεται σε όλα εκείνα τα ελαττώματα στο λογισμικό, δηλαδή όλα εκείνα τα τμήματα του λογισμικού ή εκείνες τις εντολές, τα οποία προκαλούν δυσλειτουργίες, κατάρρευση του συστήματος, διαφορετική συμπεριφορά από την αναμενόμενη και γενικά είναι επιβλαβή για την ορθή λειτουργία του συστήματος. Τα σφάλματα μπορούν να κατηγοριοποιηθούν ανάλογα με τη σημασία ή τις επιπτώσεις τους.

Κατηγοριοποίηση ανάλογα με τη σημασία (Grouping to Importance):

- **Συχνότητα (Frequency)** : Αφορά τη συχνότητα εμφάνισης για κάθε είδος σφάλματος που έχει εντοπιστεί.

- **Κόστος διόρθωσης (Correction Cost)** : Αφορά στο κόστος που απαιτείται για να διορθωθεί το συγκεκριμένο σφάλμα. Οι κυριότεροι παράγοντες που επηρεάζουν το κόστος διόρθωσης, είναι το κόστος για την ανακάλυψη και το κόστος για τη διόρθωση του σφάλματος. Αυτά τα επιμέρους κόστη αυξάνονται δραματικά όσο αυξάνεται το μέγεθος του κώδικα ή του συστήματος και όσο αργότερα ανακαλύπτεται το σφάλμα στη διαδικασία ανάπτυξης.

- **Κόστος εγκατάστασης (Installation Cost)** : Αφορά στο κόστος εγκατάστασης της διόρθωσης ενός σφάλματος. Το κόστος αυτό διαφέρει σημαντικά σε περιπτώσεις όπου η διόρθωση πρέπει να γίνει σε μεγάλο αριθμό προγραμμάτων και σε μεγάλες αποστάσεις (όπως για παράδειγμα σφάλμα στο λειτουργικό σύστημα) σε σχέση με διόρθωση η οποία πρέπει να γίνει τοπικά και σε ένα μόνο υπολογιστή.

- **Επιπτώσεις (Impacts)** : Αφορά στην αξιολόγηση των επιπτώσεων που έχει το σφάλμα στη συνολική λειτουργία του συστήματος. Ως ποσοτική μετρική μπορεί να χρησιμοποιηθεί η παρακάτω ποσότητα

Η σημασία σύμφωνα με την οποία αξιολογούνται τα σφάλματα ορίζεται ως:

σημασία= συχνότητα * (κόστος διόρθωσης + κόστος εγκατάστασης + κόστος επιπτώσεων)

Κατηγοριοποίηση ανάλογα με τις επιπτώσεις (Grouping to Impacts):

- **Ήπια (Mild)** : Οι επιπτώσεις αφορούν κυρίως αισθητικές δυσλειτουργίες που δεν έχουν σοβαρή επίδραση στο σύστημα (ορθογραφικά λάθη, μη ευθυγραμμισμένες γραμμές)

- **Μέτρια (Moderate) :** Οι έξοδοι είναι παραπλανητικοί ή περιττοί, επιβαρύνοντας την απόδοση του συστήματος.

- **Ενοχλητική (Annoying)** : Η συμπεριφορά του συστήματος, λόγω του σφάλματος είναι απάνθρωπη. Υπάρχουν σημαντικές δυσλειτουργίες και οι χειριστές συχνά πρέπει να προβούν σε σύνθετες και αφύσικες ενέργειες, για να πάρουν τα επιθυμητά αποτελέσματα.

- **Ανησυχητική (Disturbing)** : Το σύστημα αρνείται να διεκπεραιώσει νόμιμες συναλλαγές, παράγοντας σφάλματα και δυσλειτουργία σε νόμιμες εισόδους.

- **Σοβαρή (Severe)** : Το σύστημα χάνει δεδομένα και πληροφορίες κατά την αλληλεπίδραση με του χρήστες του, οδηγώντας σε προβλήματα συνέπειας.

- **Πολύ σοβαρή (Very Severe)** : Το σύστημα όχι μόνο χάνει δεδομένα και πληροφορίες, αλλά πραγματοποιεί και ενέργειες που είναι διαφορετικές από τις ζητούμενες, καθώς και λανθασμένες.

- **Ακραία (Extreme)** : Τα σφάλματα δεν περιορίζονται σε λίγους χρήστες ή σε συγκεκριμένα είδη ασυνήθιστων ενεργειών, αλλά είναι συχνά και αυθαίρετα.

- **Ανυπόφορη (Intolerable)** : Μακροπρόθεσμη, δύσκολα εντοπίσιμη και μη αντιστρέψιμη διαφθορά της βάσης δεδομένων του συστήματος οδηγεί στη πιθανότητα τερματισμού λειτουργίας του συστήματος.

- **Καταστροφική (Disastrous)** : Το σύστημα καταρρέει, αποτρέποντας ακόμα και προσπάθεια ελεγχόμενου τερματισμού του.

- **Μολυσματική (Contagious)** : Μπορεί να είναι χειρότερη από ένα σύστημα το οποίο καταρρέει. Το σύστημα μολύνει άλλα συστήματα με τα οποία επικοινωνεί, ακόμα και αν το ίδιο λειτουργεί σωστά. Οι επιπτώσεις αυτού του τύπου είναι ανυπολόγιστες (σύστημα ελέγχου στρατιωτικών εγκαταστάσεων, σύστημα ελέγχου πυρηνικών εγκαταστάσεων).

Κεφάλαιο 4) Εργαλεία Ελέγχου και Αξιολόγησης της Ποιότητας

4.1) Χρησιμότητα αυτοματοποιημένων εργαλείων

Στις μέρες μας η ανάγκη για χρήση εργαλείων, είτε αυτοματοποιημένων είτε όχι, είναι μεγαλύτερη από ποτέ. Ο λόγος είναι η ολοένα αυξανόμενη πολυπλοκότητα και βάθος των προγραμμάτων καθώς και το συνήθως πολύ μεγάλο μέγεθός τους. Συνέπεια της μεγάλης αυτής ζήτησης για εργαλεία για την αξιολόγηση και ανάλυση κώδικα, είναι η μεγάλη ανάπτυξη που γνωρίζουν αυτά τα τελευταία χρόνια. Όμως ανακύπτουν και προβλήματα από τη χρήση των εργαλείων αυτών, καθώς και ζητήματα ως προς την επιλογή του καταλληλότερου, ανάλογα με την περίπτωση και τους στόχους που θέλουμε να επιτύχουμε. Η μελέτη των διαθέσιμων επιλογών και η αξιολόγηση των πλεονεκτημάτων της χρήσης τους είναι επιτακτική για να αποφευχθεί η σπατάλη πόρων, κατά την παραγωγή του προϊόντος.

Κάθε ομάδα ανάπτυξης λογισμικού ελέγχει τα προϊόντα της, ωστόσο το λογισμικό που τελικά παραδίδεται πάντα περιέχει σφάλματα. Οι μηχανικοί ελέγχου προσπαθούν να τα εντοπίσουν πριν το προϊόν κυκλοφορήσει, αλλά τα σφάλματα πάντα εμφανίζονται και συχνά παλιά σφάλματα ξαναβγαίνουν στην επιφάνεια, παρότι το λογισμικό έχει ελεγχθεί εκτενώς. Ο αυτοματοποιημένος έλεγχος λογισμικού (automated software testing) είναι ο καλύτερος τρόπος για να αυξηθεί η αποτελεσματικότητα, η αποδοτικότητα και η κάλυψη του λογισμικού κατά τον έλεγχο.

Η χρήση των αυτοματοποιημένων εργαλείων μπορεί να απλουστεύσει δραματικά τον έλεγχο, να αυξήσει το ρυθμό ανίχνευσης σφαλμάτων και τελικά να επιτύχει σε υψηλότερη ποιότητα λογισμικού. Πέραν αυτών, μπορεί να οδηγήσει σε βελτιώσεις στην αξιοπιστία των λύσεων που προτείνονται, κάνοντάς τες πιο παραγωγικές και αποδοτικές από την οπτική γωνία των καταναλωτών. Υπάρχουν πολλές κατηγορίες εργαλείων αξιολόγησης λογισμικού και ασχολούνται με διάφορες πτυχές της διαδικασίας ανάπτυξης και ελέγχου, όπως η εφαρμογή σε διάφορους τύπους λογισμικού, εξειδίκευση σε διάφορες γλώσσες προγραμματισμού και ενασχόληση με συγκεκριμένους τύπους ελέγχων. Κάθε μια από αυτές τις κατηγορίες προσφέρει διαφορετικά πλεονεκτήματα και διαφορετικούς βαθμούς υποστήριξης κατά τη διαδικασία ανάπτυξης, ελέγχου και αξιολόγησης.

Πολλά από τα υπάρχοντα εργαλεία μπορούν να δοκιμαστούν χωρίς δεσμεύσεις και να αποκτηθούν χωρίς χρέωση (Free of charge). Υπάρχει βέβαια η επιλογή για αγορά μιας άδειας και αναβάθμισης σε πλήρη έκδοση γρήγορα και εύκολα, εάν η αξιολογήσεις για το εργαλείο είναι καλές. Ωστόσο, η αξία του εκάστοτε εργαλείου εξαρτάται από τις δυνατότητες του εργαλείου και τη βοήθεια που μπορεί να παρέχει στις διαδικασίες ανάπτυξης και ελέγχου του λογισμικού. Ουσιαστικά ένα εργαλείο θα πρέπει να επιλέγεται έτσι ώστε να ταιριάζει και να βελτιώνει ήδη υπάρχουσες τεχνικές και μεθόδους της ομάδας ανάπτυξης ή ελέγχου και όχι το αντίθετο. Η ευχρηστία και οι διευρυμένες τα τελευταία χρόνια δυνατότητες έχουν καταστήσεις τα αυτοματοποιημένα εργαλεία τόσο δημοφιλή και χρήσιμα. Καθώς τα τελευταία γίνονται ολοένα πιο εύκολα στη χρήση τους, ολοένα αυξάνεται το υποψήφιο κοινό των χρηστών τους. Τα κέρδη σε παραγωγικότητα και αποδοτικότητα από τη χρήση των εργαλείων αυξάνονται δραματικά όσο αυξάνονται τα μέλη της ομάδας ανάπτυξης ή ελέγχου που τα χρησιμοποιούν.

Τελικώς, μπορούμε να ισχυριστούμε ότι οι βελτιώσεις που παρέχουν τα εργαλεία αξιολόγησης και ελέγχου λογισμικού, τα οποία ενσωματώνουν τις τελευταίες μεθοδολογίες

και τεχνικές για έλεγχο λογισμικού, δεν αυξάνουν μόνο το κίνητρο της ομάδας ανάπτυξης αλλά και την ποιότητα του προϊόντος με αρκετούς τρόπους:

- Μείωση του κόστους ανάπτυξης και ελέγχου του λογισμικού.
- Μείωση του χρόνου ανάπτυξης νέων προϊόντων λογισμικού.
- Βελτίωση της απόδοσης, της συμμόρφωσης με τις προδιαγραφές και της διαλειτουργικότητας με άλλα λογισμικά/εξοπλισμό.
- Αύξηση του κινήτρου της ομάδας, με συνέπεια την αύξηση της κάλυψης των ελέγχων και την μείωση του χρόνου που απαιτείται για αυτούς.

4.2) Κατηγορίες και είδη εργαλείων

Υπάρχει μεγάλη ποικιλία εργαλείων αξιολόγησης και ανάλυσης κώδικα. Συγκεκριμένα υπάρχουν εργαλεία που προσφέρουν στατιστικά στοιχεία για τον κώδικα (γραμμές κώδικα, γραμμές σχολίων, μέγεθος πηγαίου κώδικα, χαρακτηριστικά μοτίβα προγραμματισμού), εργαλεία για την αναγνώριση «επιβλαβών» χαρακτηριστικών (συναρτήσεις, χειρισμός μεταβλητών και δεδομένων, εξαιρέσεις), εργαλεία αποσφαλμάτωσης, εργαλεία για τον έλεγχο κατά το χρόνο εκτέλεσης καθώς και εργαλεία βελτιστοποίησης. Κάθε μια από αυτές τις κατηγορίες έχει συγκεκριμένα πλεονεκτήματα που προσπαθεί να εκμεταλλευτεί ο προγραμματιστής, ώστε να εξασφαλίσει την παραγωγή ποιοτικού κώδικα, καθώς και τον έλεγχο της ορθότητας του κώδικά του. Προφανώς αυτά τα εργαλεία χρειάζονται σωστό χειρισμό και επίβλεψη καθ' όλη τη διάρκεια της λειτουργίας τους, ώστε να αποφευχθούν τυχόν σφάλματα και δυσλειτουργίες τους (και τα εργαλεία αποτελούν κομμάτια λογισμικού και είναι επιρρεπή στις ίδιες ατέλειες και κινδύνους όπως όλο το λογισμικό και πρέπει να ελέγχονται). Υπάρχουν γενικά χαρακτηριστικά που είναι επιθυμητά σε κάθε εργαλεία που χρησιμοποιείται και αυτά είναι:

- **Γενικευμένη διεπαφή χρήστη (Generalized User Interface)** : Τα καλύτερα εργαλεία είναι αυτά που προσφέρουν στον χρήστη τους ευελιξία στον καθορισμό τόσο των ελέγχων που θα πραγματοποιηθούν, όσο και των συνθηκών κάτω από της οποίες θα δοκιμαστεί το συγκεκριμένο πρόγραμμα που ελέγχεται.

- **Δυνατότητα διαμοίρασης (Distribute Capability)** : Η ευκολία στο διαμοιρασμό και τη χρήση ενός εργαλείου μέσα σε μια κοινότητα χρηστών είναι παραπάνω από επιθυμητή. Αυτό επιτυγχάνεται με τη δημιουργία ενός αρκετά γενικού εργαλείου στην διατύπωση και υλοποίησή του.

- **Δυνατότητα επαναχρησιμοποίησης (Reusability)** : Τα εργαλεία θα πρέπει να έχουν τέτοιο σχεδιασμό, ώστε να είναι δυνατή η χρησιμοποίησή τους σε διάφορες και πολλαπλές φάσεις του κύκλου ζωής των προγραμμάτων που αναπτύσσονται, εκτελούνται ή ελέγχονται.

Οι κυριότερες κατηγορίες εργαλείων που χρησιμοποιούνται είναι οι εξής:

- **<u>Στατικοί αναλυτές (Static Analyzers) :</u>** Οι στατικοί αναλυτές προσφέρουν περιορισμένες δυνατότητες ανάλυσης. Αυτό συμβαίνει διότι εστιάζουν σε προϋποθέσεις, σχεδιαστικές απαιτήσεις ή δομικά στοιχεία των προγραμμάτων. Στην πλειοψηφία τους αναλύουν το πρόγραμμα και εξάγουν αποτελέσματα χωρίς να το εκτελούν. Υπάρχει μεγάλη ποικιλία εργαλείων που κυμαίνεται από εργαλεία επιβολής προτύπων και περιορισμών μέχρι εργαλεία που επιτελούν σύνθετους και περίπλοκους ελέγχους και αναλύσεις. Συνήθεις χρησιμοποιούμενες κατηγορίες είναι:

    1. **Έλεγχος κώδικα (Code Check) :** Αυτά τα εργαλεία εξετάζουν τον πηγαίο κώδικα για να εντοπίσουν προγραμματιστικά χαρακτηριστικά καθορισμένα από πρότυπα ή κανόνες ή για να επιβάλουν τη χρήση συγκεκριμένων προγραμματιστικών τεχνικών.

    2. **Έλεγχος συνοχής (Consistency Check) :** Τα εργαλεία αυτά χρησιμοποιούνται για να ελέγξουν την εσωτερική συνοχή των λειτουργικών

μονάδων του κώδικα και την προσκόλληση στις προδιαγραφές που έχουν τεθεί.

3. **Έλεγχος με παραπομπές (References Check) :** Αυτά τα εργαλεία χρησιμοποιούν βιβλιοθήκες ή λεξικά για την υποστήριξη χαρακτηριστικών σε υψηλές γλώσσες προγραμματισμού. Συνήθως βρίσκονται σε μεταγλωττιστές αλλά συναντώνται και σε εργαλεία αποσφαλμάτωσης.

4. **Ανάλυση διεπαφής (Interface Analysis) :** Τα εργαλεία αυτά ελέγχουν τις διεπαφές μεταξύ των λειτουργικών μονάδων του προγράμματος. Οι έλεγχοι αφορούν τη συνοχή, την υλοποίηση των προδιαγραφών και την ορθή λειτουργία των διεπαφών.

5. **Ανάλυση εισόδου/εξόδου (Input/Output Analysis) :** Τα εργαλεία ανάλυσης εισόδου/εξόδου έχουν ως στόχο την παραγωγή νόμιμων δεδομένων εισόδου, τα οποία προκύπτουν από τις προδιαγραφές που έχουν τεθεί για την είσοδο/έξοδο του συστήματος.

6. **Ανάλυση ροής δεδομένων (Data Flow Analysis) :** Αυτά τα εργαλεία πραγματοποιούν γραφική ανάλυση ακολουθιών δεδομένων, αναφορών και μοτίβων για τον καθορισμό περιορισμών, οι οποίοι μπορούν να επιβληθούν στα δεδομένα σε κρίσιμα σημεία κατά την εκτέλεση. Αρχικά είχαν εμφανιστεί και χρησιμοποιηθεί στη διαδικασία βελτιστοποίησης στους μεταγλωττιστές.

7. **Έλεγχος λαθών (Fault Check) :** Αυτά τα εργαλεία προσδιορίζουν τυχόν αποκλίσεις, καθώς και τη σημασία και τις αιτίες τους.

8. **Ανάλυση τύπων (Type Analysis) :** Τα εργαλεία αυτά ασχολούνται με τον προσδιορισμό της σωστής χρήσης ονοματισμένων αντικειμένων δεδομένων και λειτουργιών. Συνήθως, χρησιμοποιούνται για να καθοριστεί εάν το πεδίο ορισμού των ορισμάτων των οντοτήτων είναι ορθά καθορισμένο.

9. **Ανάλυση μονάδων (Unit Analysis) :** Τα εργαλεία αυτά καθορίζουν εάν οι λειτουργικές μονάδες που προσδίδονται σε μια οντότητα είναι σωστή ορισμένες και χρησιμοποιούνται με συνέπεια.

- **Δυναμικοί αναλυτές (Dynamic Analyzers) :** Οι δυναμικοί αναλυτές προσφέρουν βοήθεια στον έλεγχο του προγράμματος, με την απ' ευθείας εκτέλεσή του και παραγωγή και αξιολόγηση στατιστικών, χαρακτηριστικών και πολλών άλλων ειδών αποτελέσματα. Η ποικιλία αυτού του είδους των εργαλεία είναι ευρεία, όπως και οι τεχνικές ανάλυσης και αξιολόγησης που χρησιμοποιούνται από αυτά. Οι κυριότερες κατηγορίες του είναι οι εξής:

  1. **Ανάλυση κάλυψης (Coverage Analysis)** : Καθορίζει και εκτιμά μεγέθη που αφορούν την επίκληση των δομικών στοιχείων του προγράμματος, ώστε να προσδιοριστεί η πληρότητα μιας δοκιμαστικής εκτέλεσης.

  2. **Ανίχνευση (Tracing)** : Ανίχνευση του ιστορικού εκτέλεσης ενός προγράμματος. Μπορεί να αναλυθεί περαιτέρω σε ανίχνευση μονοπατιών εκτέλεσης, έλεγχος σημείων καμπής, ανίχνευση ροής λογικής και ανίχνευση ροής δεδομένων.

3. **Εξομοίωση (Simulation)** : Αναπαράσταση συγκεκριμένων χαρακτηριστικών της συμπεριφοράς ενός πραγματικού ή αφαιρετικού συστήματος, μέσω ενεργειών που εκτελούνται από έναν υπολογιστή.

4. **Χρονισμό (Timing)** : Αναφορά πραγματικών χρόνων αξιοποίησης της Κεντρικής Μονάδας Επεξεργασίας από το πρόγραμμα ή από επιμέρους μέρη του.

5. **Χρησιμοποίηση πόρων (Resource Utilization)** : Ανάλυση των πόρων συστήματος, σε υλικό και λογισμικό, που χρησιμοποιούνται από το πρόγραμμα.

6. **Συμβολική εκτέλεση (Token Execution)** : Αναδόμηση της λογικής και των υπολογισμών κατά μήκος ενός μονοπατιού εκτέλεσης, μέσω της εκτέλεσης του μονοπατιού με συμβολικές τιμές στα δεδομένα, αντί των πραγματικών.

7. **Έλεγχος ισχυρισμών (Claims Check)** : Έλεγχος ισχυρισμών καθορισμένων από το χρήστη, οι οποίοι αφορούν σε σχέσεις μεταξύ στοιχείων του προγράμματος. Οι ισχυρισμοί αποτελούν λογικές εκφράσεις που καθορίζουν μια συνθήκη ή μια σχέση μεταξύ μεταβλητών του προγράμματος. Η ανάλυση μπορεί να γίνει με συμβολικά ή με πραγματικού χρόνου εκτέλεσης δεδομένα.

8. **Έλεγχος περιορισμών (Restraint Check)** : Παραγωγή ή/και λύση περιορισμών για τα μονοπάτια εισόδου/εξόδου για τον καθορισμό των δεδομένων ελέγχου, καθώς και για την επιβεβαίωση της ορθής λειτουργίας των προγραμμάτων.

4.3) Πλεονεκτήματα και μειονεκτήματα των αυτοματοποιημένων εργαλείων

Τα εργαλεία ανάλυσης και αξιολόγησης κώδικα είναι σχεδιασμένα για να αναλύουν πηγαίο κώδικα ή μεταφρασμένο κώδικα, ώστε να ανακαλύψουν κενά ασφαλείας. Ιδανικά, τα εργαλεία θα ανακάλυπταν αυτόματα τα κενά ασφαλείας, με υψηλό βαθμό βεβαιότητας ότι όντως αυτά που ανακαλύπτονται είναι κενά ασφαλείας. Ωστόσο, κάτι τέτοιο δεν είναι επιτεύξιμο με τα σημερινά μέσα και εργαλεία, για αρκετούς τύπους ελαττωμάτων και κενών ασφαλείας. Συνεπώς, τα εργαλεία συχνά χρησιμοποιούνται ως βοηθήματα για τους ανθρώπους αναλυτές, ώστε να διευκολύνουν και να κάνουν πιο αποδοτική τη δουλειά τους, παρά ως αυτοματοποιημένα και αυτόνομα εργαλεία. Μερικά εργαλεία πλέον βρίσκονται στα ενσωματωμένα περιβάλλοντα ανάπτυξης (IDE integrated development environments). Για τύπους προβλημάτων οι οποίοι μπορούν να ανιχνευτούν στη φάση ανάπτυξης λογισμικού, η χρησιμοποίηση εργαλείων μπορεί να προσφέρει άμεση ανταπόκριση στον προγραμματιστή για ζητήματα που μπορούν να προκληθούν κατά τη φάση ανάπτυξης.

<u>Κυριότερα Πλεονεκτήματα Αυτοματοποιημένων Εργαλείων</u>

- Μπορεί να γίνει σημαντική κλιμάκωση, δηλαδή να εκτελούνται πάνω σε πολλά και διαφορετικά είδη λογισμικού καθώς και να εκτελούνται επαναλαμβανόμενα σε διαδοχικές εκδόσεις.
- Για προβλήματα στα οποία τα εργαλεία μπορούν με μεγάλο βαθμό βεβαιότητας να εντοπίσουν σφάλματα και κενά ασφαλείας, όπως υπερχειλίσεις αποταμιευτών (buffer overflows), λάθη εισαγωγής στην SQL (SQL injection flaws) και άλλα, μπορούν να αποδώσουν τα μέγιστα.
- Μπορούν αν εξοικονομήσουν αρκετό χρόνο, παρέχοντας γρήγορα μια λίστα σημείων προς διερεύνηση ειδικά για μεγάλα μεγέθη λογισμικού.
- Συνήθως παρουσιάζουν καλά δομημένα αποτελέσματα, τα οποία αποτελούν ένα καλό σημείο εκκίνησης για τον πλήρη έλεγχο κατά τη διάρκεια ανάπτυξης ή μετά το τέλος της.
- Μπορούν να χρησιμοποιηθούν περιοδικά για τον εντοπισμό ελαττωμάτων και σφαλμάτων, χωρίς να υπάρχουν επιβαρύνσεις στο κόστος χρήσης τους.
- Πολλά εργαλεία είναι δωρεάν και υποστηρίζουν πολλαπλές πλατφόρμες και λειτουργικά συστήματα.
- Μπορούν να μειώσουν το κόστος υλοποίησης κατά την ανάπτυξη του λογισμικού, παρέχοντας την δυνατότητα για εντοπισμό ελαττωμάτων και κενών ασφαλείας, καθώς και τη φθηνότερη σε πόρους διόρθωσή τους κατά τη διάρκεια των πρώτων σταδίων ανάπτυξης.
- Βοηθούν τους προγραμματιστές να μάθουν ασφαλή προγραμματισμό, πράγμα το οποίο γίνεται συνεχώς δυσκολότερο λόγω της συνεχούς προσθήκης βιβλιοθηκών λογισμικού, μεθοδολογιών και τεχνικών. Μερικά από τα εργαλεία προσφέρουν πληροφορίες σχετικά με τη διόρθωση πιθανών σφαλμάτων, εκτός της ανίχνευσης των τελευταίων, κάνοντας πιο εύκολη τη ζωή των προγραμματιστών.
- Μπορούν να χρησιμοποιηθούν για τον εντοπισμό προβλημάτων και σφαλμάτων ρουτίνας ή μικρής δυσκολίας ανίχνευσης, αφήνοντας τους προγραμματιστές και τους μηχανικούς ελέγχου να ασχολούνται με την ανεύρεση πιο περίπλοκων σφαλμάτων.
- Μπορούν να βελτιστοποιήσουν των κώδικα, οδηγώντας σε μεγιστοποίηση της αποδοτικότητάς του και εξασφάλιση της ορθής λειτουργίας του λογισμικού.

Κυριότερα Μειονεκτήματα Αυτοματοποιημένων Εργαλείων

- Αρκετοί τύποι κενών ασφαλείας είναι πολύ δύσκολο να εντοπιστούν με αυτοματοποιημένο τρόπο, όπως προβλήματα πιστοποίησης (authentication), ελέγχου της πρόσβασης (access control), ανασφαλούς χρήσης της κρυπτογραφίας (insecure use of cryptography) και άλλα. Τα υπάρχοντα εργαλεία μπορούν να εντοπίζουν με αυτοματοποιημένο τρόπο μόνο ένα μικρό ποσοστό των κενών ασφαλείας των εφαρμογών. Ωστόσο, υπάρχει συνεχής πρόοδος και εξελίξεις σε αυτών τον τομέα.
- Υψηλός αριθμός από εσφαλμένες ανακαλύψεις. Τα εργαλεία έχουν την τάση να ανακαλύπτουν κενά ασφαλείας και σφάλματα ενώ δεν υπάρχουν. Αυτό αντιμετωπίζεται με χειροκίνητη διόρθωση των ανακαλύψεων αυτών από ανθρώπους αναλυτές, καταναλώνοντας πολύτιμους όμως πόρους.
- Συχνά δεν μπορούν να αντιμετωπιστούν ζητήματα δομής και ρυθμίσεων, διότι αυτά δεν απεικονίζονται μέσα στον κώδικα.
- Υπάρχει δυσκολία στην επιβεβαίωση ότι ένα κενό ασφαλείας ή ελάττωμα που ανακαλύφθηκε όντως υφίσταται και πρέπει να αντιμετωπιστεί.
- Είναι πιθανό να παραχθούν αρκετές εσφαλμένες προειδοποιήσεις και αποτελέσματα.
- Πολλά από τα εργαλεία έχουν δυσκολίες ή αδυνατούν να αναλύσουν πηγαίο κώδικα ο οποίος δεν μπορεί να μεταγλωττιστεί. Συχνά οι αναλυτές δεν μπορούν να μεταγλωττίσουν τον πηγαίο κώδικα, διότι δεν έχουν πρόσβαση ή δεν διαθέτουν τις κατάλληλες βιβλιοθήκες λογισμικού, τις οδηγίες μεταγλώττισης, ολόκληρο των κώδικα και άλλα παρόμοια ζητήματα.
- Τα δωρεάν εργαλεία πολλές φορές έχουν περιορισμένες δυνατότητες, υπάρχει έλλειψη υποστήριξης ή βοηθητικών εγγράφων και δυσκολία στον χειρισμό και παραμετροποίησή τους.
- Δεν υπάρχει δυνατότητα για εντοπισμό όλων των σφαλμάτων και κενών ασφαλείας από τα αυτοματοποιημένα εργαλεία. Επίσης ανεξάρτητα των επαναλήψεων εκτέλεσης ενός συγκεκριμένου τύπου ελέγχου σε ένα δεδομένο κομμάτι λογισμικού, τα αποτελέσματα θα είναι τα ίδια, εκτός αν τροποποιηθούν οι ρυθμίσεις του εργαλείου.

Κεφάλαιο 5) Παρουσίαση υπαρχόντων εργαλείων

5.1) BLAST

Περίληψη

Το BLAST (Berkeley Lazy Abstraction Software verification Tool) είναι ένα αυτοματοποιημένο εργαλείο αξιολόγησης και ελέγχου των χρονικών χαρακτηριστικών ασφάλειας για προγράμματα γραμμένα σε γλώσσα C. Δοθέντος ενός προγράμματος C και ενός χρονικού χαρακτηριστικού ασφαλείας, το BLAST είτε αποδεικνύει στατικά ότι το πρόγραμμα ικανοποιεί το χαρακτηριστικό που ελέγχεται είτε παρέχει ένα μονοπάτι εκτέλεσης που εκθέτει μια παράβαση του χαρακτηριστικού (είτε εφόσον μπορεί ο έλεγχος να μην καταλήγει σε συμπέρασμα, δεν ολοκληρώνει την εκτέλεσή του). Το BLAST κατασκευάζει, εξερευνά και βελτιώνει αφηρημένα σχέδια του χώρου καταστάσεων του προγράμματος, βάσει τεμπέλικων κατηγορημάτων (lazy predicate) και ανακάλυψης κατηγορημάτων μέσω παρεμβολής (interpolation-based predicate discovery). Το BLAST είναι σχετικά ανεξάρτητο από το μηχάνημα/υλικό και το λειτουργικό/λογισμικό πάνω στα οποίο τρέχει. Βέβαια, συνίσταται η χρήση του εργαλείου σε επεξεργαστές Intel Pentium και σε λειτουργικά Linux και Microsoft Windows πάνω σε Cygwin, καθώς άλλοι συνδυασμοί λειτουργικών συστημάτων και επεξεργαστών δεν έχουν ελεγχθεί εκτεταμένα (παρότι το εργαλείο μπορεί να δουλεύει εκεί). Τέλος, το BLAST είναι δωρεάν λογισμικό/εργαλείο, το οποίο έχει κυκλοφορήσει κάτω από την τροποποιημένη άδεια BSD (Modified BSD license).

Διαθέσιμα έγγραφα

Το εγχειρίδιο (manual) για το BLAST διανέμεται στις παρακάτω μορφές:

- HTML
- PDF
- Postscript files

Τα αρχεία αυτά περιέχονται στο πακέτο των αρχείων, καθώς και στον ιστότοπο του εργαλείου στη διεύθυνση:

http://mtc.epfl.ch/blast/

Στην ίδια διεύθυνση υπάρχει και είναι διαθέσιμο δωρεάν για κατέβασμα το πακέτο αρχείων του εργαλείου.

Δημιουργοί

Η τρέχουσα έκδοση του εργαλείου(v2.5, 2008-07-11) έχει δημιουργηθεί και συντηρείται από τους:

- Dirk Beyer (SFU)
- Thomas A. Henzinger (EPFL)
- Ranjit Jhala (UCSD)
- Rupak Majumdar (UCLA)

Επίσης, στην ανάπτυξη του εργαλείου έχουν συνεισφέρει και πολλοί άλλοι, συμπεριλαμβανομένων και των:

- Adam Chlipala
- Jeff Fischer
- Ken McMillan
- Shaz Qadeer
- Gregoire Sutre
- Gregory Theoduloz
- Damien Zufferey

Σχετικά με πνευματικά δικαιώματα και άδεια χρήσης

Το BLAST έχει κυκλοφορήσει κάτω από την άδεια χρήσης *Apache License, Version 2.0 (the "License")* και πρέπει να χρησιμοποιείται σύμφωνα με τους όρους της άδειας αυτής. Το περιεχόμενο της άδειας χρήσης μπορεί να βρεθεί στον ιστότοπο http://www.apache.org/licenses/LICENSE-2.0.

Βασικά χαρακτηριστικά και δυνατότητες του εργαλείου

Έλεγχος προσβασιμότητας (Reachability checking)

Ο πιο απλός τρόπος συγγραφής ενός προγράμματος το οποίο θα ελεγχθεί από το BLAST είναι να καθοριστεί μια ετικέτα “C”, σε μια τοποθεσία του προγράμματος στην οποία δεν πρέπει να υπάρχει πρόσβαση. Ο ελεγκτής μοντέλων του BLAST θα ελέγξει εάν υπάρχει πρόσβαση σε αυτή την ετικέτα “C”. Η βασική εντολή για την εκτέλεση του εργαλείου είναι η:
”pblast.opt prog.c -main start -L errlabel”,
η οποία εκτελεί τον ελεγκτή μοντέλων για το πρόγραμμα “prog.c” και ελέγχει την προσβασιμότητα στην ετικέτα “errlabel”, ξεκινώντας από την αρχική τοποθεσία της συνάρτησης “start”. Η εκτέλεση της εντολής ολοκληρώνεται με μια ανίχνευση του σφάλματος ή με ένα μήνυμα ότι το σφάλμα ήταν απροσπέλαστο(ή, εφόσον δεν μπορεί να ληφθεί απόφαση σχετικά με την προσβασιμότητα μπορεί να μην τερματίσει). Ως προκαθορισμένα για τα ορίσματα “start” και “errlabel” υπάρχουν η “main” και η ετικέτα “ERROR” αντίστοιχα. Συνεπώς, καλώντας το BLAST με την εντολή:
“*pblast.opt prog.c*”,
ελέγχεται εάν η ετικέτα “ERROR” είναι προσβάσιμη, όταν η εκτέλεση του προγράμματος ξεκινά από τη “main”.

Έλεγχος ισχυρισμών (Assertion checking)

Η πιο βολική μορφή για τις ιδιότητες ασφαλείας είναι οι ισχυρισμοί (assertions), οι οποίοι καθορίζουν ποια τμήματα του προγράμματος παραμένουν αναλλοίωτα (invariants). Η σύνταξη της εντολής είναι "assert(e)", όπου "e" είναι μια έκφραση σε "C". Εάν η έκφραση αυτή πάρει την τιμή "0" κατά τη διάρκεια εκτέλεσης, το πρόγραμμα ματαιώνει την εκτέλεσή του. Η λογική πίσω από αυτό είναι ότι ο προγραμματιστής έχει εμπεριστατωμένη άποψη σχετικά με το γεγονός ότι η έκφραση "e" αποτελεί ένα αναλλοίωτο κομμάτι του προγράμματος, στο σημείο όπου έχει εισαχθεί η εντολή "assert(e)". Τυπικά, οι ισχυρισμοί ελέγχονται κατά τη διάρκεια εκτέλεσης του προγράμματος. Με τη χρήση του BLAST μπορούν να ελεγχθούν όμως και στατικά, κατά τη διάρκεια της μεταγλώττισης και ακόμα δεν απαιτείται να δημιουργηθεί ένα συγκεκριμένο σενάριο ελέγχου.

Εκτέλεση του BLAST από τη γραμμή εντολών

Η βασική εντολή για την εκτέλεση του BLAST είναι :
pblast.opt filename -main mainfunction ErrorLabel,
όπου "filename" είναι το όνομα του αρχείου που θα ελεγχθεί, "mainfunction" το όνομα της συνάρτησης από την οποία θα αρχίσει ο έλεγχος και "ErrorLabel" η ετικέτα η οποία θα αναζητηθεί κατά τον έλεγχο. Υπάρχουν περιπτώσεις για τις οποίες δεν είναι απαραίτητα όλα τα ορίσματα, όπως για παράδειγμα όταν θέλουμε να κάνουμε έλεγχο προσβασιμότητας, ξεκινώντας από τη συνάρτηση "main", καθώς αυτά είναι τα προκαθορισμένα ορίσματα της εντολής. Στην τελευταία περίπτωση θα αρκούσε η εντολή:
pblast.opt filename
για να επιτευχθεί το επιθυμητό αποτέλεσμα.

Εκτέλεση του BLAST από τη γραφική διεπαφή χρήστη (Graphical User Interface, GUI)

Εκτελώντας από τη γραμμή εντολών την εντολή:

blastgui.opt

ξεκινάει η λειτουργία της γραφικής διεπαφής, η οποία παρέχει την ίδια λειτουργικότητα και είναι πιο εύκολη και φιλική στη χρήση.

Χρονικές προδιαγραφές ασφάλειας (temporal security specification)

Οι ισχυρισμοί (assertions) είναι ένας απλός και τοπικός τρόπος για τον καθορισμό της ορθότητας ενός προγράμματος. Γενικότερα, ενδιαφερόμαστε για τις χρονικές προδιαγραφές ασφαλείας, όπου θέλουμε να ελέγξουμε εάν το πρόγραμμά μας ικανοποιεί ιδιότητες πεπερασμένων καταστάσεων. Για παράδειγμα, ένα πρόγραμμα διαχείρισης κλειδωμάτων (locks) θα πρέπει να αποκτά και να απελευθερώνει τα κλειδώματα σε αυστηρή εναλλαγή (δύο διαδοχικές κλήσεις για κλείδωμα, χωρίς ενδιάμεση κλήση για ξεκλείδωμα, θα θεωρούνται σφάλμα και αντιστρόφως). Σε γενικές γραμμές μας απασχολούν τα χαρακτηριστικά ασφαλείας σχετικά με την αλληλουχία γεγονότων ενός προγράμματος και τη διασφάλιση της ορθότητας της τελευταίας.

Ανακάλυψη έξυπνων κατηγορημάτων (smart predicates)

Η προκαθορισμένη μηχανή ανακάλυψης κατηγορημάτων του BLAST μπορεί να μην επιτυγχάνει πάντοτε να βρίσκει «καλά» κατηγορήματα. Συνεπώς, το BLAST ενσωματώνει κάποιες επιπλέον τεχνικές ανάλυσης για να βρίσκει κατάλληλα κατηγορήματα. Για να χρησιμοποιηθούν αυτές πρέπει να χρησιμοποιηθούν τα παρακάτω ορίσματα από τη γραμμή εντολών:

-craig 1

-craig 1 -scope

-craig 2

Τα οποία υλοποιούν ανακάλυψη κατηγορημάτων βάση παρεμβολής. Η επιλογή "-craig 1 –scope" χρησιμοποιεί ένα επιπλέον ευρετικό απόδοσης σε σχέση με το "-craig 1", το οποίο αφαιρεί τα κατηγορήματα που δεν περιέχονται στο "scope". Η επιλογή "-craig 2" κάνει μια ακριβή ανάλυση, η οποία χρειάζεται περισσότερο χρόνο σε σχέση με την επιλογή "-craig 1", αλλά παράγει μια πολύ καλύτερη αφαίρεση (abstraction). Ακόμα και όταν η προκαθορισμένη μέθοδος ανακάλυψης κατηγορημάτων επιτυγχάνει, οι παραπάνω επιλογές "-craig" συχνά επιτυγχάνουν με λιγότερα κατηγορήματα. Επίσης, υπάρχουν υλοποιημένα και άλλα ευρετικά τα οποία μπορούν να χρησιμοποιηθούν μέσω της σημαίας "-predH". Συνίσταται η εκτέλεση του BLAST με χρήση της σημαίας "-predH 7", όπου 7 είναι το υψηλότερο επίπεδο.

Αποθηκευμένες αφηρημένες δομές (Abstractions)

Όταν το BLAST ολοκληρώσει την εκτέλεση του προγράμματος "filename.c",δημιουργεί ένα αρχείο με όνομα " filename.abs", το οποίο περιέχει την αφηρημένη (abstraction) δομή που χρησιμοποιήθηκε. Αυτή η δομή μπορεί να χρησιμοποιηθεί σε επόμενες εκτελέσεις για να εξοικονομήσει χρόνο στην ανακάλυψη κατηγορημάτων. Μετέπειτα εκτελέσεις μπορεί να απαιτούν ένα σενάριο ελέγχου με επιστροφές (regression), όπου κάποια τμήματα του προγράμματος έχουν αλλάξει και το BLAST εκτελείται για να επιβεβαιώσει την ορθή λειτουργία τους. Το αρχείο που περιέχει την προηγούμενη αφηρημένη δομή μπορεί να φορτωθεί μαζί με το αρχείο με χρήση της σημαίας "-loadabs", επιταχύνοντας το χρόνο της νέας εκτέλεσης.

Η γλώσσα προδιαγραφών

Χρήση του εργαλείου

Η γλώσσα προδιαγραφών επεξεργάζεται από ένα εργαλείο της γραμμής εντολών, το οποίο δέχεται ως είσοδο μια προδιαγραφή και μια λίστα από αρχεία πηγαίου κώδικα "C". Ένα νέο ενορχηστρωμένο (instrumented) αρχείο "C" δημιουργείται, το οποίο συνδυάζει τα πηγαία αρχεία της εισόδου και διασφαλίζει ότι ικανοποιούν τις ιδιότητες που περιγράφονται στην προδιαγραφή που δίνεται στην είσοδο. Επίσης, δημιουργείται ένα αρχείο "instrumented.pred" το οποίο περιέχει υποδείξεις για κατηγορήματα του BLAST. Η εντολή με την οποία εκτελείται το εργαλείο που επεξεργάζεται τη γλώσσα προδιαγραφών είναι η:

spec.opt myspec.spc myfile.c
και τα αρχεία που θα παραχθούν σε αυτή την περίπτωση είναι:

instrumented.c και instrumented.pred

όπου “myspec.spc” είναι η προδιαγραφή που δίνεται σαν είσοδος και το “myfile.c” είναι η λίστα των αρχείων που δίνονται σαν είσοδος.

Χρησιμοποιώντας το BLAST

- **Επιλογές χρήστη:** Χρήσιμες εντολές για την εκτέλεση του BLAST από την γραμμή εντολών (εισάγετε την εντολή “ pblast.opt -help ” για πλήρη λίστα των επιλογών).
- **Επιλογές ελέγχου μοντέλων:** Αυτές οι εντολές είναι διαθέσιμες για την εξατομίκευση της εκτέλεσης ελέγχου μοντέλων στις εκάστοτε ανάγκες.
- **Επιλογές βελτιστοποίησης προγράμματος:** Το BLAST ενσωματώνει ένα σύνολο από ρουτίνες ανάλυσης προγραμμάτων οι οποίες μπορούν να κάνουν την ανάλυση να τρέξει γρηγορότερα. Μπορούν να ενεργοποιηθούν ή να απενεργοποιηθούν χρησιμοποιώντας τις κατάλληλες εντολές.
- **Έλεγχος παράλληλου μοντέλου και ανταγωνισμός:** Το BLAST ενσωματώνει ένα τμηματικό αλγόριθμο για νήματα (threads) για τον έλεγχο του ανταγωνισμού σε πολυνηματικά προγράμματα “C”. Αυτές οι επιλογές σχετίζονται με τον αλγόριθμο για τον έλεγχο του ανταγωνισμού.
- **Αποθηκευμένες αφηρημένες δομές και Σύνοψη:** Αυτές οι επιλογές χρησιμοποιούνται για την αποθήκευση και φόρτωση αφηρημένων δομών κατά τη διάρκεια μιας εκτέλεσης του BLAST.
- **Επιλογές παραγωγής αποδείξεων:** Το BLAST ενσωματώνει ένα σύνολο επιλογών για την παραγωγή αποδείξεων της μορφής “PCC”. Οι αποδείξεις είναι η έξοδος σε μορφή κειμένου και σε σύνταξη “LF”. Συνεπώς μπορούν να διαβαστούν και να κωδικοποιηθούν από ένα κανονικοποιημένο “PCC” κωδικοποιητή αποδείξεων.
- **Παλιά ευρετικά τα οποία δεν υποστηρίζονται/χρησιμοποιούνται πλέον:** Αυτές οι επιλογές μπορούν να παραληφθούν, καθώς αναφέρονται σε παλιότερες εκδόσεις ευρετικών. Τα προκαθορισμένα ευρετικά του BLAST είναι αυτά που βρέθηκε ότι λειτουργούν βέλτιστα.
- **Γενικές επιλογές:** Αυτές οι επιλογές επιτρέπουν στο χρήστη να καθορίσει διάφορες διατάξεις εκτέλεσης, κυρίως για σκοπούς αποσφαλμάτωσης.

Γραφική διεπαφή χρήστη (GUI)

Το BLAST έρχεται με ένα στοιχειώδες GUI, το οποίο σκοπό έχει να κάνει πιο εύκολη την προβολή των ιχνών των αντιπαραδειγμάτων. Για να ξεκινήσει η εκτέλεση πρέπει να δοθεί η εντολή:
blastgui.opt

Οι πηγαίοι κώδικες και τα κατηγορήματα μπορούν να φορτωθούν χρησιμοποιώντας την επιλογή “File” από την κεντρική εργαλειοθήκη, όπως και εισάγοντας τα ονόματα στα κατάλληλα πεδία κειμένου και επιλέγοντας “Load”. Υπάρχουν 4 υπό-πλαίσια που εμφανίζονται και αναφέρονται στον καταγραφέα γεγονότων (event log), πηγαίο αρχείο (source file), αρχείο κατηγορημάτων (predicate file) και ίχνη αντιπαραδειγμάτων (counterexamples traces) αντίστοιχα. Για να εκτελεστεί το BLAST πρέπει αρχικά να επιλεγεί ένα αρχείο (προαιρετικά και ένα αρχείο κατηγορημάτων), στη συνέχεια να δοθούν οι εντολές στο πεδίο κειμένου με ετικέτα “options” και τέλος να επιλεγεί το πλήκτρο “Run”. Στην περίπτωση που το σύστημα δεν περιέχει σφάλματα, το BLAST (θεωρητικά) θα παράγει ένα παράθυρο που θα το δηλώνει, αλλιώς (θεωρητικά) θα αλλάξει στο υπό-πλαίσιο που περιέχει τα ίχνη για τα αντιπαραδείγματα, προβάλλοντας ένα αντιπαράδειγμα το οποίο παραβιάζει την προδιαγραφή. Στο προηγούμενο σενάριο, λέμε θεωρητικά επειδή είναι πιθανό, όπως είδαμε νωρίτερα, το BLAST να φτάσει σε ένα σημείο όπου δεν θα μπορεί να παράγει τα σωστά κατηγορήματα και να συνεχίσει την εκτέλεσή του. Σε αυτή την περίπτωση, το GUI αλλάζει στο υπό-πλαίσιο ιχνών αντιπαραδειγμάτων, όπου περιέχεται

πληροφορία σχετικά με το σημείο στο οποίο κόλλησε η εκτέλεση και υπάρχει η δυνατότητα στο χρήστη να παρέχει (μαντεύοντας) κατηγορήματα στο BLAST.

Ομαλοποίηση (Aliasing)

Η ομαλοποίηση δεικτών (pointer aliasing) είναι μια μεγάλη πηγή πολυπλοκότητας στην υλοποίηση του BLAST. Το BLAST έρχεται με έναν αλγόριθμο ανάλυσης Andersen ο οποίος είναι αναίσθητος σχετικά με τη ροή και το πεδίο, με σκοπό την απάντηση στις ερωτήσεις της ομαλοποίησης δεικτών εσωτερικά. Η υλοποίηση της ανάλυσης δεικτών χρησιμοποιεί τα BDD. Επιπλέον, το BLAST επιτρέπει στο χρήστη να εισάγει πληροφορίες σχετικά με την ομαλοποίηση (οι οποίες έχουν παραχθεί από κάποια άλλη ανάλυση ομαλοποίησης) μέσω ενός αρχείου. Η σύνταξη του αρχείου ομαλοποίησης (alias file) είναι μια λίστα ισοτήτων “C”, ανάμεσα σε εκφράσεις μνήμης “C” (μεταβλητές, αποαναφορές, προσβάσεις πεδίων) διαχωρισμένες με κόμματα.

Αρχιτεκτονική του BLAST

Το BLAST χρησιμοποιεί την υποδομή CIL ως κύριο τρόπο για να διαβάζει προγράμματα “C”. Τα προγράμματα εσωτερικά αναπαρίστανται ως αυτόματα ελέγχου ροής (control flow automata), βάση του μοντέλου CFA. Σύνολα καταστάσεων αναπαρίστανται με τη δομή δεδομένων Region, το οποίο αναπαριστά τα σύνολα καταστάσεων ως Boolean φόρμουλες πάνω σε ένα βασικό σύνολο κατηγορημάτων και επιτρέπει ενέργειες Boolean σε περιοχές (regions), ελέγχοντας για έλλειψη (emptiness) ή ένταξη (inclusion).

Ο αφηρημένος συναρτητής (abstraction functor) παίρνει το μοντέλο Region και το μοντέλο CFA, προσθέτοντας επιπλέον (δομημένες και αφηρημένες) ενέργειες για πριν και μετά, καθώς και μεθόδους για ανάλυση των αντιπαραδειγμάτων. Χρησιμοποιώντας το τελευταίο, ο συναρτητής LazyAbstraction υλοποιεί τον αλγόριθμο ελέγχου μοντέλων σε υψηλό επίπεδο αφαίρεσης. Το BLAST χρησιμοποιεί το Simplify Theorem Prover και το Vampyre Proof-Generating Theorem Prover ως διαδικασίες απόφασης. Για να χειριστεί τις Boolean φόρμουλες το BLAST χρησιμοποιεί το Colorado University Decision Diagram package.

5.2) PC-Lint

Σχετικά με Lint και PC-lint

Το PC-Lint είναι μια βελτιωμένη και εμπορική (επί πληρωμή) εκδοχή του δωρεάν εργαλείου “lint”, το οποίο πρωτοεμφανίστηκε στην έκδοση 7 (version 7) του Unix το 1979 και εκτελείται από τη γραμμή εντολών. Η αρχική αυτή έκδοση (lint) μπορούσε να εντοπίσει ορισμένες ύποπτες και μη μεταφέρσιμες (non portable) δομές, ως πιθανά σφάλματα σε προγράμματα γραμμένα σε γλώσσα “C”. Ο όρος “lint” έχει καθιερωθεί έκτοτε και αναφέρεται στη σημείωση και ανίχνευση ύποπτης συμπεριφοράς τμημάτων κώδικα, γραμμένων σε οποιαδήποτε γλώσσα προγραμματισμού. Συνήθως, η σάρωση από εργαλεία που συμπεριφέρονται όμοια με το “lint” (lint-like behavior) κάνουν στατική ανάλυση κώδικα (static code analysis). Στις ύποπτες δομές περιλαμβάνονται μεταβλητές οι οποίες χρησιμοποιούνται προτού οριστούν, συνθήκες οι οποίες είναι σταθερές (πάντα αληθείς/ψευδείς) και υπολογισμοί τα αποτελέσματα των οποίων είναι πιθανό να είναι εκτός των ορίων που μπορούν αν αναπαρασταθούν από τον τύπο δεδομένων που χρησιμοποιείται. Παρόμοιες μορφές ανάλυσης με τα “lint-like” εργαλεία πραγματοποιούνται από σύγχρονους μεταγλωττιστές βελτιστοποίησης, οι οποίοι προσδοκούν στη δημιουργία γρηγορότερου κώδικα, όπως και στην πραγματοποίηση ελέγχων για τις παραδοσιακές αυτές ύποπτες δομές ως μέρος της δουλειάς τους. Με την πάροδο των ετών έχουν βελτιωθεί και εμπλουτιστεί οι μέθοδοι που χρησιμοποιούνται για τον έλεγχο, καθώς και οι δομές που ελέγχονται ως ύποπτες. Οι κυριότερες μορφές ελέγχου που ενσωματώνουν τα σύγχρονα “lint-like” εργαλεία είναι έλεγχος συνοχής μεταξύ των διαφορετικών υπό-μονάδων (cross-module consistency checking), έλεγχος για τη μεταφερσιμότητα του κώδικα σε άλλους μεταγλωττιστές (code portatability) και υποστήριξη υποσημειώσεων οι οποίες καθορίζουν την αναμενόμενη συμπεριφορά ή τις ιδιότητες του κώδικα.

Θέματα Συμβατότητας (Compatibility issues)

Εκτός του PC-lint, το οποίο έχει φτιαχτεί και είναι συμβατό με τα Microsoft Windows και OS/2, υπάρχει και ένα ξεχωριστό εργαλείο το FlexeLint το οποίο είναι συμβατό για συστήματα Unix και άλλες πλατφόρμες, αλλά με μεγαλύτερο χρηματικό κόστος. Βέβαια, με τη χρήση ενός εξομοιωτή (emulator) των windows είναι δυνατή η χρήση και εκτέλεση του PC-lint σε περιβάλλον Unix. Ένας από αυτούς τους εξομοιωτές είναι ο “wine” ο οποίος μπορεί να εγκατασταθεί απλά και εύκολα και στη συνέχεια πάνω σε αυτόν να εγκατασταθεί το εργαλείο στα Unix.

Σχετικά με τη λειτουργία του PC-lint

Το εργαλείο PC-lint μπορεί να χρησιμοποιηθεί για την εξασφάλιση της ποιότητας (quality assurance) προγραμμάτων γραμμένων σε γλώσσα “C” και τον έλεγχο της συμμόρφωσης με τα πρότυπα προγραμματισμού “MISRA C” και “MISRA C++”. Στο εργαλείο ενσωματώνονται μέθοδοι για έλεγχο προβλημάτων σε παράλληλα προγράμματα γραμμένα με χρήση νημάτων POSIX (POSIX threads). Η έξοδος του εργαλείου μπορεί να χρησιμοποιηθεί από άλλα εργαλεία για τη παραγωγή αναφορών και για την παρουσίαση των προειδοποιήσεων σε μια προσιτή προς το χρήστη μορφή. Το μεγαλύτερο πλεονέκτημα του PC-lint απέναντι στους ανταγωνιστές του είναι η πολύ καλή δουλειά που κάνει στην ανάλυση κώδικα (code analysis) μαζί με το γεγονός ότι είναι ένα φθηνό, ελαφρύ (δεν καταναλώνει πολλούς υπολογιστικούς πόρους), απλό στη δομή και εκτέλεση εργαλείο. Δεν χρειάζεται ιδιαίτερες ή ακριβές υποδομές για να εκτελεστεί ή ένα εξουσιοδοτημένο

εξυπηρετητή (license server), αλλά εκτελείται όμοια με τους μεταγλωττιστές από τη διεπαφή της γραμμής εντολών και η έξοδος του κατευθύνεται στα “STDOUT” και “STDERR”.

Σχετικά με άδεια χρήσης

Η τιμή για την απόκτηση άδειας χρήσης του εργαλείου για ένα υπολογιστή είναι στα 390 δολάρια Αμερικής (USD). Η άδεια χρήσης για 10 υπολογιστές ενός τοπικού δικτύου (LAN) είναι στα 3500 δολάρια Αμερικής και κάθε επιπλέον άδεια χρήσης κοστίζει 300 δολάρια. Για της άδειες χρήσης υπάρχει εγγύηση επιστροφής χρημάτων για 30 ημέρες. Αυτό αφορά την έκδοση του PC-lint για DOS/Windows, ωστόσο υπάρχει και μια έκδοση πηγαίου κώδικα (περίπλοκου πηγαίου κώδικα) που ονομάζεται FlexeLint και η οποία μπορεί να χρησιμοποιηθεί σε οποιοδήποτε σύστημα υποστηρίζεται η χρήση ενός μεταγλωττιστή για γλώσσα “C”, όπως τα Linux/Unix. Ωστόσο, η άδειες για το FlexeLint κοστίζουν σχεδόν 3 φορές περισσότερο (998 δολάρια η μια άδεια) σε σχέση με το PC-lint (λόγω της δυνατότητας να χρησιμοποιηθεί σε πληθώρα συστημάτων) και είναι ασύμφορη για ιδιώτες ή μικρές επιχειρήσεις.

Δυνατότητες και χαρακτηριστικά PC-lint/FlexeLint

Η τρέχουσα έκδοση είναι η έκδοση 9 (version 9.00j) που κυκλοφόρησε στις 3/1/2013 και μπορεί να βρεθεί από εδώ http://www.gimpel.com/html/index.htm. Ο σκοπός του PC-lint/FLexeLint είναι η ανίχνευση σφαλμάτων, ασυνεπειών (inconsistency) και μη μεταφέρσιμων δομών για κώδικα γραμμένο σε γλώσσες “C” και “C++”, μέσω της στατικής ανάλυσης του πηγαίου κώδικα. Όπως και τα περισσότερα, αν όχι όλα, τα αυτόματα εργαλεία σάρωσης κώδικα, παράγει εσφαλμένα μηνύματα ανεύρεσης σφαλμάτων για τα οποία γίνεται συνεχώς προσπάθεια βελτίωσης και ελαχιστοποίησης. Υποστήριξη παρέχεται για τις γλώσσες:

- K&R C
- ANSI C
- ANSI/ISO C++

Επίσης, παρέχεται υποστήριξη για τους κυριότερους μεταγλωττιστές (Microsoft, GCC) και βιβλιοθήκες, για μεταγλωττιστές ενσωματωμένων συστημάτων (embedded systems) καθώς και για αρκετούς ανεξάρτητους μεταγλωττιστές. Οι βασικές κατηγορίες ελέγχου κατά τον έλεγχο είναι:

- Στατική ανίχνευση τιμών για αυτόματες και στατικές (ακόμα και παγκόσμιες) μεταβλητές, με σκοπό την ανίχνευση διακριτικών αρχικοποιήσεων και προβλήματα από τη λάθος χρήση των τιμών.
- Ανίχνευση τιμών μεταξύ συναρτήσεων (inter-function) μέσω της ισχυρής διαδικασίας ανίχνευσης μεταξύ δηλώσεων (inter-statement) η οποία ξεπερνά τα όρια μεταξύ των συναρτήσεων.
- Επαναλαμβανόμενα περάσματα (multi-pass) για την πλήρη εκμετάλλευση της ανίχνευσης τιμών μεταξύ συναρτήσεων και της στατικής ανίχνευσης τιμών, όπου το πλήθος των περασμάτων μπορεί να οριστεί από το χρήστη.
- Με την ανίχνευση τιμών ως τεχνολογία επίτρεψης, υποστηρίζεται ο έλεγχος σήμανσης (semantics) για σχεδόν 100 βιβλιοθήκες συναρτήσεων καθώς και συναρτήσεις χρηστών.
- Προαιρετικός ισχυρός έλεγχος τύπων (βασισμένος στο typedef) με πλούσιο σύνολο επιλογών για την ανίχνευση πλασματικών διαφορών μεταξύ των τύπων δεδομένων. Υπάρχει η δυνατότητα δημιουργίας μιας ολόκληρης ιεραρχίας ελεγμένων τύπων για βαθμωτούς τύπους χρησιμοποιώντας μόνο ένα “typedef”.

- Έλεγχος σήμανσης ορισμένης από το χρήστη για ορίσματα συναρτήσεων και τιμές που επιστρέφονται.
- Εύρεση αχρησιμοποίητων μακροεντολών (macros), “typedef”, κλάσεων (classes), μελών (members) και δηλώσεων (declarations) σε ολόκληρο τον κώδικα.
- Διαχείριση της ροής για τον ελέγχου πιθανώς μη αρχικοποιημένων μεταβλητών.
- Παροχή υποστήριξης για υποσύνολα των προτύπων “MISRA C” και “MISRA C++” (Motor Industry Software Reliability Association).
- Χρησιμοποίηση προ-μεταγλωττισμένων επικεφαλίδων (pre-compiled headers) για την επίτευξη ταχύτατης επεξεργασίας πολλαπλών υπομονάδων.
- Δυνατότητα σε έμμεσα αρχεία (εμφωλευμένα σε οποιοδήποτε βάθος) να περιέχουν ονόματα αρχείων (filenames), επιλογές (options) και περιβαλλοντικές μεταβλητές (environment variables) παρέχουν ευελιξία.
- Δυνατότητα μορφοποίησης και προσαρμογής των μηνυμάτων σφάλματος του ”lint” ώστε να υποστηρίζεται η ενσωμάτωση σε μια ευρεία ποικιλία συντακτών (editors) και ενσωματωμένων περιβαλλόντων ανάπτυξης (Integrated Development Environments, IDEs).
- Δυνατότητα ενσωμάτωσης όλων των επιλογών σε κώδικα χρήστη.
- Ύπαρξη γρήγορης σάρωσης ενός περάσματος (one pass operation), με επιλογή για πολλαπλά περάσματα για επίτευξη της ανίχνευσης μεταξύ συναρτήσεων.
- Δυνατότητα επέκτασης των πινάκων ανάλογα με τις ανάγκες για το χειρισμό μεγαλύτερο εφαρμογών και σαρώσεων.

Διαδραστική online δοκιμαστική έκδοση (Interactive Online Demo)

Στον ιστότοπο της εφαρμογής υπάρχει μια online διαδραστική δοκιμαστική έκδοση http://www.gimpel-online.com/OnlineTesting.html για την επίδειξη των χαρακτηριστικών και των επιδόσεων της εφαρμογής. Στην ιστοσελίδα αυτή υπάρχουν έτοιμα ορισμένα απλά προγράμματα γραμμένα σε γλώσσες “C” και “C++”, τα οποία επιδεικνύουν την συμπεριφορά του εργαλείου για διάφορα ζητήματα. Εκτός των έτοιμων παραδειγμάτων, υπάρχει δυνατότητα για περιορισμένη δοκιμή πραγματικού κώδικα (ο κώδικας δίνεται μέσω επικόλλησης από το χρήστη), ώστε να αξιολογηθεί η επίδοση του εργαλείου σε πραγματικές συνθήκες από πιθανούς χρήστες/αγοραστές της.

5.3) Cppcheck

Περίληψη

Το Cppcheck είναι ένα ανοιχτού κώδικα εργαλείο ανάλυσης για πηγαίο κώδικα γραμμένο σε γλώσσες C και C++. Αντίθετα με τους μεταγλωττιστές και αρκετά άλλα εργαλεία ανάλυσης, δεν ανιχνεύει συντακτικά λάθη στον κώδικα. Το Cppcheck ανιχνεύει μόνο τύπους σφαλμάτων τα οποία δεν ανιχνεύονται από τους μεταγλωττιστές. Ο στόχος της εκτέλεσης είναι να μην παράγονται εσφαλμένα μηνύματα για ανιχνευμένα σφάλματα. Συνίσταται η χρήση όσο το δυνατό περισσοτέρων προειδοποιήσεων (warnings) στον μεταγλωττιστή. Βασικές λειτουργίες του Cppcheck είναι:

- Έλεγχος μη-κανονικοποιημένου κώδικα ο οποίος περιέχει διάφορες προεκτάσεις μεταγλωττιστή (compiler extensions) και κώδικα συμβολικής γλώσσας στην ίδια γραμμή (inline assembly).
- Δυνατότητα μεταγλώττισης του εργαλείου Cppcheck από οποιονδήποτε μεταγλωττιστή C++, ο οποίος χειρίζεται το τελευταίο πρότυπο C++.
- Διαλειτουργικότητα σε συστήματα τα οποία πληρούν τις απαιτήσεις του εργαλείου σε υπολογιστική ισχύ (cpu) και μνήμη (memory).

Ακρίβεια (Accuracy)

Το Cppcheck κάθε άλλο παρά είναι ολοκληρωμένο. Αντιθέτως, συνεχώς βελτιώνεται και εμπλουτίζεται ώστε να είναι πιο ακριβές και έγκυρο. Σπάνια τα σφάλματα που αναφέρονται είναι εσφαλμένα, αλλά υπάρχουν αρκετοί τύποι σφαλμάτων οι οποίοι δεν ανιχνεύονται από το Cppcheck. Συνήθως είναι πιο πιθανό να βρεθούν αρκετά σφάλματα στον κώδικα μετά από προσεκτικό έλεγχο και δοκιμές, παρά με τη χρήση του Cppcheck. Βέβαια, ορισμένα σφάλματα είναι πολύ δύσκολο να εντοπιστούν μέσω δοκιμών και ελέγχων και εκεί είναι που το Cppcheck βοηθάει, καθώς εντοπίζει αρκετούς τέτοιους τύπους σφαλμάτων.

Εγχειρίδιο οδηγιών (manual)

Το εγχειρίδιο οδηγιών του Cppcheck είναι διαθέσιμο σε μορφή PDF από τον ιστότοπο του εργαλείου.

Τύποι ελέγχων

Μεταφερσιμότητα στα 64-bit (portability)

Ελέγχει κατά πόσο υπάρχουν θέματα μεταφερσιμότητας (portability) στα 64-bit:

- Ανάθεση διεύθυνσης σε/από ακέραιο (int) /μεγάλο ακέραιο (long)
- Μετατροπή διευθύνσεων από/σε ακέραιο όταν γίνεται επιστροφή από συνάρτηση

Αυτόματες μεταβλητές (auto variables)

Ένας δείκτης προς μια μεταβλητή είναι έγκυρος όσο η μεταβλητή είναι στο πεδίο εφαρμογής (in scope). Ελέγχονται τα:

- Επιστροφή ενός δείκτη σε μια αυτόματη ή προσωρινή μεταβλητή
- Ανάθεση της διεύθυνσης μια μεταβλητής σε μια αποδοτική παράμετρο μιας συνάρτησης
- Επιστροφή αναφοράς σε τοπική/προσωρινή μεταβλητή
- Επιστροφή της διεύθυνσης μιας παραμέτρου συνάρτησης

Χρήση Boost

Έλεγχος για τη σωστή χρήση του Boost: Τροποποίηση του δοχείου (container) κατά τη διάρκεια του BOOST_FOREACH.

Έλεγχος ορίων (Bounds checking)

Έλεγχος για περιπτώσεις όπου στοιχεία του προγράμματος βρίσκονται εκτός ορίων.

Έλεγχος κλάσεων (class checking)

Έλεγχος του κώδικα κάθε κλάσης για:

- Έλλειψη δημιουργών (constructors) και αντίγραφα (copies) δημιουργών
- Έλεγχος για την αρχικοποίηση όλων των μεταβλητών από τους δημιουργούς
- Έλεγχος της ανάθεσης όλων των μεταβλητών από τον τελεστή"="
- Δημιουργία μηνύματος προειδοποίησης αν χρησιμοποιούνται οι συναρτήσεις memset και memcopy στον κώδικα μιας κλάσης
- Έλεγχος αν ο καταστροφέας (destructor) είναι εικονικός (virtual), αν η κλάση είναι βασική
- Ο τελεστής "=" θα πρέπει να επιστρέφει αναφορά στον εαυτό του
- Ο τελεστής "=" θα πρέπει να ελέγχει για αναθέσεις στον εαυτό του
- Έλεγχος κόστους για τις συναρτήσεις μέλη (member functions)
- Έλεγχος της σειράς αρχικοποιήσεων (initializations)
- Δημιουργία προτεινόμενης χρήσης της λίστας αρχικοποιήσεων
- Έλεγχος ύποπτων αφαιρέσεων (substractions) από το "this"

Ασφάλεια εξαιρέσεων (Exception safety)

Έλεγχος της ασφάλειας εξαιρέσεων για:

- Παραγωγή (throwing) εξαιρέσεων στους καταστροφείς (destructors)
- Παραγωγή εξαιρέσεων κατά τη διάρκεια άκυρης κατάστασης (invalid state)
- Παραγωγή ενός αντιγράφου της εξαίρεσης που πιάστηκε (caught) αντί για την παραγωγή ξανά της αρχικής εξαίρεσης
- Η εξαίρεση πιάστηκε από την τιμή (value) και όχι από αναφορά (reference)

Είσοδος/έξοδος (Input/Output, I/O)

Έλεγχος των πράξεων εισόδου/εξόδου για:

- Κακή χρήση της συνάρτησης “sprintf”, με επικαλυπτόμενα δεδομένα (overlapping data)
- Έλλειψη ή λάθος πλάτος των προσδιοριστών (specifiers) στο αλφαριθμητικό μορφοποίησης κατά την κλήση της “scanf”
- Χρήση ενός αρχείου το οποίο έχει κλείσει (closed)
- Είσοδος/έξοδος αρχείου χωρίς τοποθέτηση (positioning) ,το οποίο οδηγεί σε απροσδιόριστη συμπεριφορά (undefined behavior)
- Ανάγνωση ενός αρχείου το οποίο έχει ανοιχτεί για εγγραφή (και αντίστροφα)
- Χρήση της “fflush()” σε μια ροή εισόδου (stream)
- Άκυρη χρήση μιας ροής εξόδου, όπως “std::cout<<std::cout”
- Λανθασμένο πλήθος ορισμάτων προς τις συναρτήσεις “printf” και “scanf”

Διαρροές αυτόματων μεταβλητών (auto variables leaks)

Έλεγχος για περιπτώσεις όπου μια αυτόματη μεταβλητή κατανέμεται (allocate), αλλά δεν από-κατανέμεται (deallocate).

Ταύτιση αναθέσεων και προϋποθέσεων (match assignments and conditions)

Ταύτιση αναθέσεων και προϋποθέσεων για:

- Αναντιστοιχία ανάθεσης και σύγκρισης, με αποτέλεσμα η σύγκριση να είναι πάντα αληθής/ψευδής
- Αναντιστοιχία αριστερού και δεξιού μέρους της πρότασης που συγκρίνεται, με αποτέλεσμα η σύγκριση να είναι πάντα αληθής/ψευδής
- Ανίχνευση των αντιστοιχιών μεταξύ των “if” και των “else” στις συνθήκες

Διαρροές μνήμης, η διεύθυνση δεν είναι δεσμευμένη (Memory leaks, address not taken)

Δεν γίνεται δέσμευση της μνήμης που κατανέμεται.

Διαρροές μνήμης, μεταβλητές κλάσεων (Memory leaks, class variables)

Αν ο δημιουργός (constructor) κατανέμει μνήμη και ο καταστροφέας (destructor) δεν την από-κατανέμει.

Διαρροές μνήμης, μεταβλητές συναρτήσεων (Memory leaks, function variables)

Ελέγχει αν υπάρχει κατανεμημένη μνήμη όταν μια συνάρτηση βγαίνει από το πεδίο εφαρμογής (out of scope).

Διαρροές μνήμης, μεταβλητές δομών (Memory leaks, struct variables)

Ελέγχει αν υπάρχουν μέλη δομών τα οποία δεν έχουν από-κατανεμηθεί.

Συναρτήσεις χωρίς επανείσοδο (non reentrant functions)

Παραγωγή προειδοποίησης αν χρησιμοποιηθεί κάποια από της παρακάτω συναρτήσεις χωρίς επανείσοδο:

- crypt
- ctermid
- ecvt
- fcvt
- fgetgrent
- fgetpwent
- fgetspent
- gcvt
- getgrent
- getgrgid
- getgrnam
- gethostbyaddr
- gethostbyname
- gethostbyname2
- gethostent
- getlogin
- getnetbyaddr
- getnetbyname
- getnetgrent
- getprotobyname
- getpwent
- getpwnam
- getpwuid
- getrpcbyname
- getrpcbynumber
- getrpcent
- getservbyname
- getservbyport
- getservent
- getspent
- getspnam
- gmtime
- localtime
- readdir
- strtok
- tempnam
- ttyname

Άκυροι δείκτες (Null pointers)

Ελέγχει για τη σωστή από-αναφορά (dereferencing) δεικτών οι οποίοι είναι πλέον άκυροι.

Παρωχημένες συναρτήσεις (Obsolete functions)

Παραγωγή προειδοποίησης αν χρησιμοποιηθεί κάποια από τις παρακάτω παρωχημένες συναρτήσεις:

- asctime_r
- bcmp
- bcopy
- bsd_signal
- bzero
- ctime_r
- ecvt
- fcvt
- ftime
- gcvt
- getcontext
- gethostbyaddr
- gethostbyname
- getwd
- index
- makecontext
- pthread_attr_getstackaddr
- pthread_attr_setstackaddr
- rand_r
- rindex
- scalbln
- swapcontext
- tmpnam
- tmpnam_r
- ualarm
- usleep
- utime
- vfork
- wcswcs

Διάφορα (other)

Διάφοροι άλλοι έλεγχοι:

- Ανάθεση τιμής bool σε δείκτη (μετατροπή τιμής bool σε διεύθυνση)
- Διαίρεση με το μηδέν
- Καταστροφή αντικειμένου στο πεδίο εφαρμογής (in scope) αμέσως μετά τη δημιουργία του
- Ανάθεση σε δήλωση ισχυρισμού (assert statement)
- Η συνάρτηση “sizeof” για πίνακα δίνεται ως όρισμα συνάρτησης
- Η συνάρτηση “sizeof” για αριθμό δίνεται ως όρισμα συνάρτησης
- Η συνάρτηση “sizeof(pointer)” χρησιμοποιείται αντί για το μέγεθος των δεδομένων τα οποία δεικτοδοτούνται
- Λανθασμένο μήκος ορισμάτων για τις συναρτήσεις “substr” και “strncmp”
- Χρησιμοποίηση των συναρτήσεων “free()” και “delete()” για άκυρη τοποθεσία μνήμης
- Χρησιμοποίηση διπλής “free()” ή “closedir()”
- Τέλεση πράξης μεταξύ bit (bitwise operation) με αρνητικό δεξί τελούμενο (operand)
- Πλεονάζουσα (redundant) αντιγραφή δεδομένων για μεταβλητή τύπου “const”

- Μεταγενέστερη ανάθεση ή αντιγραφή σε μια μεταβλητή ή αποταμιευτή (buffer)
- Εύρεση νεκρού κώδικα ο οποίος είναι απροσπέλαστος, λόγω ελέγχου αντί-προϋποθέσεων (counter-conditions) σε εμφωλευμένες δηλώσεις “if”
- Μετατροπή δείκτη σε αρχείο “Cpp” με τρόπο που χρησιμοποιείται στη “C”
- Μετατροπή μεταξύ μη συμβατών τύπων δεικτών
- Πλεονάζοντα “if”
- Λανθασμένη χρήση της συνάρτησης “strtol”
- Μη προσημασμένη (unsigned) διαίρεση
- Πέρασμα παραμέτρου μέσω τιμής (passing by value)
- Μη ολοκληρωμένες δηλώσεις (incomplete statements)
- Έλεγχος του τρόπου με τον οποίο χρησιμοποιούνται οι προσημασμένες μεταβλητές χαρακτήρων (signed char variables)
- Το πεδίο εφαρμογής των μεταβλητών μπορεί να είναι περιορισμένο
- Συνθήκη η οποία είναι πάντοτε αληθής/ψευδής
- Ασυνήθιστη αριθμητική δεικτών
- Πλεονάζουσα ανάθεση σε μια δήλωση “switch”
- Πλεονάζουσα πράξη πριν/μετά από μια δήλωση “switch”
- Πλεονάζουσα πράξη μεταξύ bit σε μια δήλωση “switch”
- Πλεονάζουσα χρήση της συνάρτησης “strcpy” σε μια δήλωση “switch”
- Αναζήτηση για “sizeof sizeof”
- Αναζήτηση για υπολογισμούς μέσα στη συνάρτηση “sizeof()”
- Ανάθεση μιας μεταβλητής στον εαυτό της
- Αμοιβαίος αποκλεισμός (mutual exclusion) για το “||” πάντα εκτιμάται σε αληθές
- Αποσαφήνιση υπολογισμών με χρήση παρενθέσεων
- Χρήση προσαύξησης (increment) σε Boolean
- Σύγκριση μιας Boolean με ένα μη μηδενικό ακέραιο
- Σύγκριση μιας έκφρασης Boolean με έναν ακέραιο ο εκτός του “0” ή “1”
- Σύγκριση μιας συνάρτησης που επιστρέφει Boolean με χρήση σχεσιακών τελεστών (relational operator)
- Σύγκριση μιας τιμής Boolean με μια άλλη τιμή Boolean με χρήση σχεσιακών τελεστών
- Έλεγχος ύποπτων συνθηκών (ανάθεση+σύγκριση)
- Έλεγχος ύποπτων συνθηκών (σύγκριση πραγματικού χρόνου μεταξύ κυριολεκτικών αλφαριθμητικών)
- Έλεγχος ύποπτων συνθηκών (αλφαριθμητικό κυριολεκτεί ως boolean)
- Έλεγχος ύποπτης σύγκρισης ενός κυριολεκτικού (literal) αλφαριθμητικού με μια μεταβλητή “char*”
- Διπλότυπη δήλωση “break”
- Απροσπέλαστος κώδικας
- Έλεγχος αν μη προσημασμένη μεταβλητή είναι αρνητική
- Έλεγχος αν μη προσημασμένη μεταβλητή είναι θετική
- Χρήση “bool” σε έκφραση μεταξύ bit
- Ύποπτη χρήση του “;” στο τέλος της δήλωσης των “if/for/while”
- Λανθασμένη χρήση των συναρτήσεων από τη βιβλιοθήκη “ctype”
- Σύγκριση αποτελεσμάτων υπολοίπου (modulo) τα οποία είναι πάντα αληθή/ψευδή
- Πίνακας ο οποίος δεν έχει γεμίσει εξ’ ολοκλήρου με τη χρήση των συναρτήσεων “memset/memcpy/memmove”

Χρήση STL

Έλεγχος για άκυρη χρήση του STL:

- Σφάλματα εκτός ορίων (out of bounds)

- Λανθασμένη χρήση επαναληπτών (iterators) κατά την επανάληψη σε ένα δοχείο (container)
- Αναντιστοιχία δοχείων στις κλήσεις
- Για διανύσματα η χρήση επαναληπτών/δεικτών μετά την χρήση της “push_back”
- Βελτιστοποίηση με τη χρήση της “empty()” αντί της “size()” για τη διασφάλιση γρήγορου κώδικα
- Ύποπτη συνθήκη κατά τη χρήση της “find”
- Πλεονάζουσες συνθήκες
- Κοινά σφάλματα κατά τη χρήση του “string::c_str”
- Χρησιμοποίηση αυτόματου δείκτη (auto_ptr)
- Άχρηστες κλήσεις αλφαριθμητικών και συναρτήσεων STL

Μη αρχικοποιημένες μεταβλητές (uninitialized variables)

Έλεγχος για τη χρήση μη αρχικοποιημένων μεταβλητών και δεδομένων.

Μη χρησιμοποιούμενες συναρτήσεις (unused functions)

Έλεγχος για συναρτήσεις οι οποίες δεν καλούνται ποτέ.

UnusedVar

Έλεγχοι UnusedVar:

- Μη χρησιμοποιούμενες μεταβλητές
- Κατανεμημένη αλλά όχι χρησιμοποιημένη μεταβλητή
- Μη αναγνωσμένη μεταβλητή
- Μεταβλητή στην οποία δεν έχει γίνει ανάθεση
- Μη χρησιμοποιούμενο μέλος δομής

Χρήση τελεστών postfix (postfix operators)

Παραγωγή προειδοποίησης για τη χρήση τελεστών postfix “++” και “--” αντί για τελεστές prefix.

Μελλοντικές εκδόσεις

Υπάρχει σχεδιασμός για την κυκλοφορία νέων εκδόσεων κάθε 1-2 μήνες.

Η επόμενη προς κυκλοφορία έκδοση είναι η 1.58 η οποία σχεδιάζεται να κυκλοφορήσει στις 12 Ιανουαρίου 2013.

Σφάλματα και αιτήσεις δυνατοτήτων (bugs and feature requests)

Συνίσταται η χρήση του Trac για την αναφορά προβλημάτων και σφαλμάτων στο σύνδεσμο http://apps.sourceforge.net/trac/cppcheck/.

Κυριότερες πηγές ελαττωμάτων για αναφορά αφορούν:

- Εσφαλμένες ανακαλύψεις σφαλμάτων (false positives)
- Το εργαλείο Cppcheck κολλάει(hangs)/καταρρέει(crashes) κατά την εκτέλεσή του
- Σφάλμα και αποτυχία κατά τη μεταγλώττιση

Κυριότερες αναφορές για προσθήκες αφορούν:

- Αποτυχία ανίχνευσης σφάλματος
- Αλλαγή της εξόδου του εργαλείου Cppcheck
- Προτάσεις για νέους τύπους ελέγχων

Άδεια χρήσης

Είναι δυνατό να αποκτήσετε δωρεάν τον πηγαίο κώδικα της τελευταίας έκδοσης του εργαλείου από εδώ https://github.com/danmar/cppcheck/.

Επίσης, λόγω του ότι το εργαλείο είναι ανοιχτού κώδικα, χρειάζεται τη συνεισφορά της κοινότητας των χρηστών του για την περαιτέρω βελτίωση και ανάπτυξή του.

5.4) PMD

Περίληψη:

Το PMD είναι ένα στατικό εργαλείο ανάλυσης πηγαίου κώδικα γραμμένου σε γλώσσα JAVA, το οποίο στηρίζεται στη χρήση συνόλου κανόνων και το οποίο αναγνωρίζει πιθανά προβλήματα όπως:

- Πιθανά σφάλματα: Κενά τμήματα κώδικα “try”, ”catch”, ”finally” και ”switch”
- Νεκρό κώδικα: Μη χρησιμοποιούμενες τοπικές μεταβλητές, παραμέτρους και ιδιωτικές μεθόδους
- Κενές δηλώσεις “if” και ”while”
- Υπέρ-πολύπλοκες εκφράσεις: Αχρείαστες δηλώσεις “if”, βρόχους (loops) τύπου “for” οι οποίοι μπορούν να γίνουν τύπου “while”
- Μη βελτιστοποιημένο κώδικα: Σπατάλη στη χρήση των “String” και “StringBuffer”
- Κλάσεις με υψηλές μετρήσεις κυκλωματικής πολυπλοκότητας (Cyclomatic Complexity)
- Διπλότυπα κώδικα: Κώδικας που έχει αντιγραφεί και επικολληθεί μπορεί να σημαίνει σφάλματα που έχουν αντιγραφεί και επικολληθεί

Στη γενική περίπτωση τα σφάλματα που εντοπίζει το PMD δεν είναι “πραγματικά” σφάλματα, αλλά περισσότερο μη αποδοτικός κώδικας (η εφαρμογή ίσως θα μπορούσε να εκτελείται σωστά ακόμα και χωρίς να διορθωθούν).

Αν και επισήμως το όνομα “PMD” δεν έχει κάποια σημασία, ανεπίσημα υπάρχουν αρκετά ονόματα και το ποιο κατάλληλο είναι το “Programming Mistake Detector” (Ανιχνευτής Προγραμματιστικών Σφαλμάτων).

Άδεια χρήσης:

Το PMD κυκλοφορεί με άδεια χρήσης τύπου “BSD”. Οι “BSD” άδειες αφορούν ελεύθερο λογισμικό και διέπουν τους τρόπους χρήσης και διανομής του. Συνήθως αρκούνται στην περίληψη ορισμένων προϋποθέσεων (ανάλογα με την έκδοση της “BSD” άδειας), οι οποίες καθορίζουν τις λεπτομέρειες για τον τρόπο χρήσης, διανομής, τροποποίησης ή εμπλουτισμού του λογισμικού.

Τρόπος λειτουργίας:

Το PMD ελέγχει πηγαίο κώδικα με βάση ένα σύνολο κανόνων και παράγει μια αναφορά. Η ροή και τα στάδια της εκτέλεσης έχουν ως εξής:

- Το PMD εκτελείται από τη γραμμή εντολών και περνιούνται ως ορίσματα ένα όνομα αρχείου και ένα σύνολο κανόνων
- Το PMD διαχειρίζεται μια ροή εισόδου (Input stream) από το αρχείο προς έναν αναλυτή (parser) δημιουργημένο σε JavaCC
- Το PMD δέχεται μια αναφορά σε ένα Αφηρημένο Δέντρο Σύνταξης (abstract syntax tree) από τον αναλυτή
- Το PMD παραδίδει το Αφηρημένο Δέντρο Σύνταξης στο επίπεδο του πίνακα συμβόλων, το οποίο δημιουργεί σκοπούς (scopes), βρίσκει δηλώσεις (declarations) και χρήσεις (usages)
- Αν κάποιος κανόνας χρειάζεται Ανάλυση Ροής Δεδομένων (Data Flow Analysis), το Αφηρημένο Δέντρο Σύνταξης παραδίδεται στο επίπεδο Ανάλυσης Ροής Δεδομένων

για την δημιουργία γράφων ελέγχου ροής (control flow graphs) και κόμβων ροής δεδομένων (data flow nodes)
- Κάθε κανόνας στο σύνολο κανόνων τελικώς διασχίζει το Αφηρημένο Δέντρο Σύνταξης και ελέγχει για προβλήματα
- Η αναφορά του PMD γεμίζει με παραβιάσεις των κανόνων και εκτυπώνεται σε μορφή XML,HTML ή άλλη

Πρόσθετα:

Υπάρχουν υλοποιημένα πρόσθετα για την εκτέλεση του PMD για τα παρακάτω:

- JDeveloper
- Eclipse
- jEdit
- JBuilder
- Omnicore's CodeGuide
- NetBeans/Sun Studio
- IntelliJ IDEA
- TextPad
- Maven
- Ant
- Gel
- JCreator
- Hudson
- Jenkins
- Sonar
- Emacs

Τρέχουσα έκδοση:

Η τρέχουσα έκδοση είναι η 5.0.1, η οποία είναι μικρής έκταση βελτίωση (σε σχέση με την 5.0) που κυκλοφόρησε στις 28-11-2012 και περιέχει αρκετές προσθήκες και διορθώσεις σχετικά με σφάλματα. Η επόμενη έκδοση που αναπτύσσεται είναι η 5.1, ενώ θα κυκλοφορήσουν εκδόσεις 5.0.x για διορθώσεις σφαλμάτων αν χρειαστεί.

Σχέδιο Δράσης (Roadmap):

Το σχέδιο δράσης περιλαμβάνει όλα τα διαφορετικά τμήματα του PMD πάνω στα οποία οι προγραμματιστές δουλεύουν τώρα, χωρίς να υπάρχει κάποια εγγύηση για το πότε θα ολοκληρωθούν (αν ποτέ ολοκληρωθούν) :

- Καλύτερη ανάλυση συμβόλων (better symbol analysis): Με τον σημερινό τρόπο λειτουργίας του το PMD εξετάζει μόνο ένα πηγαίο αρχείο τη φορά. Αντιθέτως, θα έπρεπε να μπορεί να επιλύει σύμβολα κατά πλάτος των κλάσεων. Αυτή η δυνατότητα θα επιτρέψει τη διόρθωση ορισμένων υπαρχόντων σφαλμάτων και θα επιτρέψει την συγγραφή πολλών επιπλέον κανόνων. Ωστόσο, θα χρειαστεί αρκετή δουλειά ακόμα προτού ολοκληρωθεί, επειδή απαιτείται ανάλυση αρχείων κλάσεων
- Ανάλυση της ροής δεδομένων (data flow analysis) : Ένα επίπεδο ανάλυσης ροής δεδομένων είναι υπό ανάπτυξη, το οποίο θα επιτρέπει πιο περίπλοκους κανόνες,

όπως εξάλειψη κοινών υπό-εκφράσεων (common subexpressions), κίνηση αμετάβλητου κώδικα βρόχων (loop invariant code motion), ανέλκυση προτάσεων κώδικα (code hoisting suggestions), συρρίκνωση καλυμμάτων (shrink wrapping) και εξάλειψη μερικού πλεονασμού (partial redundancy elimination).

- Εκκαθάριση κώδικα (code cleanups) : Ορισμένα τμήματα του κώδικα είναι λίγο ακατάστατα:
    - Το RuleSetFactory είναι σε άσχημη κατάσταση. Πρέπει να παραγοντοποιηθεί ξανά σε επίπεδα ή γενικά σε κάτι ποιο δομημένο
    - Εκκαθαρίσεις θα είναι ευπρόσδεκτες για τα ConstructorCallsOverridableMethod και DoubleCheckedLocking
    - Η γραφική διεπαφή χρήστη είναι σε άσχημη κατάσταση (τα κάτω πλαίσια δείχνουν περίεργα)
    - Η γραμματική έχει ορισμένα περίεργα κομμάτια:
        - Το BlockStatement έχει μια περίεργη τροποποίηση για τους ορισμούς κλάσεων εντός μεθόδων
        - Το enumLookahead() φαίνεται λίγο υπερβολικό και ίσως θα μπορούσε να χρησιμοποιήσει Modifiers.
        - Ολόκληρος ο "κόμβος ξεσκαρταρίσματος" (discardable node) φαίνεται σπάταλος
        - Το ExtendsList ίσως δεν χρειάζεται το ExtendsMoreThanOne
        - Η κλάση ClassOrInterfaceBodyDeclaration έχει μια τερατώδη πρόβλεψη για τον έλεγχο enums
        - Ορισμένες υποσημειώσεις δεν είναι σωστές

Εκτέλεση από τη γραμμή εντολών:

Για Linux και άλλα λειτουργικά συστήματα βασισμένα στο UNIX:

Για την εκτέλεση από τη γραμμή εντολών στα Linux δίνεται η εντολή:

"./run.sh pmd -d [filename|jar or zip file containing source code|directory] -f [report format] -R [ruleset file]"

Το PMD περιέχει αρκετές χρήσιμες λειτουργίες (utilities) από τη γραμμή εντολών. Σε παλιότερες εκδόσεις αυτές οι επιμέρους λειτουργίες είχαν ξεχωριστές δέσμες ενεργειών (script) για την εκκίνησή τους, ωστόσο η διαδικασία έχει απλοποιηθεί για τα συστήματα Unix στην έκδοση 5.0. Πλέον υπάρχει μόνο μια δέσμη ενεργειών, η οποία καλείται "run.sh" και περιέχεται μέσα στον φάκελο bin/ της διανομής του PMD.

Το πρώτο όρισμα κατά την εκτέλεση από τη γραμμή εντολών είναι το όνομα της λειτουργίας που θέλουμε να εκτελέσουμε (πχ "pmd","designer") ενώ τα υπόλοιπα ορίσματα είναι συγκεκριμένα με τη λειτουργία που χρησιμοποιείται.

Βασικός τρόπος για χρήση σε λειτουργικό σύστημα Windows:

Για την εκτέλεση από τη γραμμή εντολών στα Windows δίνεται η εντολή:

pmd -d [filename|jar or zip file containing source code|directory] -f [report format] -R [ruleset file]"

Ως είσοδο στο PMD μπορεί να δοθεί ένα όνομα αρχείου, ένα όνομα καταλόγου ή ένα αρχείο .jar ή .zip το οποίο περιέχει τον πηγαίο κώδικα ενός προγράμματος γραμμένου σε JAVA. Επίσης, η διανομή του PMD περιέχει αρχεία κανόνων μέσα στα αρχεία .jar .

5.5) FindBugs

Περίληψη:

Το FindBugs ψάχνει για σφάλματα σε προγράμματα γραμμένα σε JAVA. Βασίζεται στην ιδέα των μοτίβων σφαλμάτων (bug patterns). Τα μοτίβα σφαλμάτων αναφέρονται σε τμήματα κώδικα με συγκεκριμένη μορφή, τα οποία συχνά περιέχουν λάθη και σφάλματα. Η ύπαρξη των μοτίβων αυτών οφείλεται σε διάφορους λόγους:

- Δύσκολες στην χρήση δυνατότητες της γλώσσας προγραμματισμού
- Παρεξηγημένες μέθοδοι της Προγραμματιζόμενης Διεπαφής της Εφαρμογής (API)
- Παρεξηγημένα μη μεταβλητά στοιχεία του κώδικα, όταν ο τελευταίος τροποποιείται κατά τη διάρκεια συντήρησης
- Τυπογραφικά λάθη

Το FindBugs χρησιμοποιεί στατική ανάλυση κώδικα για την επιθεώρηση δυαδικού κώδικα JAVA για τυχόν εμφανίσεις μοτίβων σφαλμάτων. Το πλεονέκτημα της στατικής ανάλυσης που υλοποιεί το FindBugs σημαίνει ότι μπορεί να εντοπίσει τα μοτίβα απλά σαρώνοντας τον κώδικα, χωρίς να χρειάζεται εκτέλεσή του. Το τελευταίο αυτό χαρακτηριστικό συμβάλλει στην απλότητα του εργαλείου, καθιστώντας δυνατή την εκτέλεσή του μόλις μερικά λεπτά μετά το κατέβασμα και εγκατάστασή του. Επίσης, η σάρωση γίνεται και σε μεταγλωττισμένα προγράμματα JAVA, ώστε να μην χρειάζεται ο πηγαίος κώδικας. Ωστόσο, λόγω της απλότητας χρήσης και λειτουργίας του, το εργαλείο εμφανίζει λανθασμένες προειδοποιήσεις και μηνύματα για ανιχνευμένα σφάλματα, ποσοστό που στην πράξη είναι μικρότερο του 50%.

Το FindBugs υποστηρίζει μια αρχιτεκτονική προσθέτων (plugins), η οποία επιτρέπει στον οποιονδήποτε να προσθέσει νέους ανιχνευτές σφαλμάτων. Σύνδεσμοι και οδηγίες υπάρχουν στη σελίδα των δημοσιεύσεων του εργαλείου, ακόμα για αυτούς που δεν είναι εξοικειωμένοι με δυαδικό κώδικα JAVA. Το εργαλείο είναι γραμμένο σε JAVA και μπορεί να εκτελεστεί σε οποιοδήποτε περιβάλλον συμβατό με το *Java Development Kit 1.5 (JDK)* της *Sun*. Το FindBugs μπορεί να αναλύσει προγράμματα γραμμένα σε οποιαδήποτε έκδοση της JAVA. Αρχικά είχε αναπτυχθεί από τους Bill Pugh και David Hovemeyer, ενώ πλέον συντηρείται και επεκτείνεται από μια ομάδα εθελοντών υπό τον Bill Pugh.

Άδεια Χρήσης:

Το εργαλείο FindBugs είναι δωρεάν και κυκλοφορεί υπό την άδεια χρήσης *Lesser GNU Public License*.

Δοκιμαστική έκδοση (Demo):

Είναι δυνατή η δοκιμή του εργαλείου σε κώδικα του χρήστη με τη χρήση της τεχνολογίας Java Webstart. Για εκδόσεις κάτω τις Java 1.4 χρησιμοποιείται η παλιά γραφική διεπαφή χρήστη (GUI) ενώ για εκδόσεις από 1.5 και πάνω χρησιμοποιείται η νέα, η οποία περιέχει ορισμένες νέες δυνατότητες. Ωστόσο και οι δυο εκδοχές των διεπαφών χρησιμοποιούν την ίδια μηχανή ανάλυσης κώδικα.

Η εκτέλεση της δοκιμαστικής αυτής έκδοσης βασίζεται στην έκδοση FindBugs 1.2.0 και παρέχει αποτελέσματα από την ανάλυση πηγαίων αρχείων κώδικα. Τα αποτελέσματα

αποτελούνται από μια συγκεντρωτική αναφορά των σφαλμάτων που βρέθηκαν, μια λίστα των σφαλμάτων που έχει παραχθεί σε HTML καθώς και μια παρουσίαση μέσω της νέας γραφικής διεπαφής χρήστη (GUI) για το Java Webstart, η οποία εισήχθηκε στην έκδοση 1.1 του FindBugs, η οποία παρουσιάζει τις προειδοποιήσεις και τις σχετικές πηγές.

Σχετικά με την δοκιμαστική έκδοση: Τα αποτελέσματα που προκύπτουν από τα την εκτέλεση της έκδοσης 1.2.0 του FindBugs αφορούν την εξ' ορισμού προσπάθεια. Τα αποτελέσματα δεν περιέχουν προειδοποιήσεις χαμηλής προτεραιότητας ή προειδοποιήσεις για κακόβουλο κώδικα. Ενώ ορισμένες φορές χρειάζεται και γίνεται χειρωνακτικά διόρθωση των αποτελεσμάτων, δεν έχει γίνει το ίδιο με εσφαλμένα μηνύματα σφάλματος, ώστε να δίνεται μια σωστή εντύπωση για τον τρόπο λειτουργίας και τις δυνατότητες του εργαλείου.

Γνωστοποίηση Αδυναμιών: Ευτυχώς η *Java* δεν είναι *C* ή *C++* ,όπου η από-αναφορά ενός άκυρου δείκτη ή η πρόσβαση εκτός ορίων ενός πίνακα παράγει μια εξαίρεση χρόνου εκτέλεσης (runtime exception) και όχι μια εκμετάλλευση κελύφους (shell exploit). Παρότι καμία προειδοποίηση εδώ δεν θεωρείται ότι αποτελεί μια αδυναμία ασφαλείας, τα αποτελέσματα δεν έχουν τροποποιηθεί χειρωνακτικά για να επιβεβαιώνεται αυτό. Ωστόσο, ενώ το FindBugs είναι ανοιχτού κώδικα και στη διάθεση τόσο των προγραμματιστών όσο και των ατόμων με κακόβουλες προθέσεις, δεν θεωρείται ότι η γνωστοποίηση αυτών των αποτελεσμάτων και του τρόπου λειτουργίας είναι μια απερίσκεπτη γνωστοποίηση.

Προτάσεις: Συνίσταται αρχικά ο έλεγχος των προειδοποιήσεων για διορθώσεις που αναφέρονται από το εργαλείο, καθώς οι προγραμματιστές θα θέλουν να διορθώσουν τα σφάλματα με μεσαία και υψηλή προτεραιότητα προτού ελέγξουν τις υπόλοιπες κατηγορίες σφαλμάτων για επιπλέον διορθώσεις.

Κατηγορίες σφαλμάτων:

- Σφάλμα Ορθότητας (Correctness bug) : Πιθανό σφάλμα λόγω λανθασμένης λογικής κατά τη συγγραφή του κώδικα, το οποίο δεν είχε σκοπό να εισάγει ο προγραμματιστής. Είναι πιθανή η εμφάνιση αρκετών εσφαλμένων προειδοποιήσεων για σφάλματα.
- Κακή Πρακτική (Bad Practice) : Παραβιάσεις των συνιστώμενων και βασικών τεχνικών προγραμματισμού. Παραδείγματα είναι ο κατακερματισμένος κώδικας (hash code), προβλήματα ισότητας (equals problems), κλωνοποιημένο ιδίωμα (cloneable idiom), παρατημένες εξαιρέσεις (dropped exceptions), προβλήματα σειριοποίησης (serializable problems) και κακή χρήση του *finalize* (misuse of finalize).
- Επισφαλής κώδικας (Dodgy code) : Κώδικας ο οποίος προκαλεί σύγχυση, με ανομοιογένεια ή γραμμένος με τρόπο που οδηγεί εύκολα σε σφάλματα. Παραδείγματα είναι οι νεκρές τοπικές αποθήκες (dead local stores), αποτυχημένο *switch* (switch fall through), ανεπιβεβαίωτες μετατροπές (unconfirmed casts) και πλεονάζοντες έλεγχοι άκυρων τιμών που είναι γνωστό ότι είναι άκυρες (redundant null check of value known to be null).

Έκδοση FindBugs 2:

Η έκδοση FindBugs 2.0 έχει κυκλοφορήσει και προσφέρει καλύτερη ακρίβεια και απόδοση σε σχέση με τις προηγούμενες. Τα κυριότερα χαρακτηριστικά της νέας έκδοσης είναι:

- Κατάταξη σφαλμάτων (Bug Rank) : Τα σφάλματα αντιστοιχίζονται σε κατηγορίες (σε κλίμακα 1-20) και ομαδοποιούνται στις κατηγορίες:
    - 1-4 -> Πιο τρομακτικά (scariest)
    - 5-9 -> Τρομακτικά (scary)
    - 10-14 -> Ενοχλητικά (troubling)
    - 15-20 -> Ανησυχητικά (of concern)

  Επειδή πολλοί χρήστες συγχύζονταν από την προτεραιότητα που υπήρχε στις αναφορές σφαλμάτων και θεωρούσαν ότι τα υψηλής προτεραιότητας σφάλματα είναι σημαντικά, η προτεραιότητα (priority) μετονομάστηκε σε εμπιστοσύνη (confidence). Σφάλματα διαφορετικών μοτίβων θα πρέπει να συγκρίνονται με βάση την κατηγορία τους και όχι από την εμπιστοσύνη (προτεραιότητα) τους.

- Αποθήκευση στο νέφος (cloud storage) : Υποστήριξη ενός βολικού τρόπου διαμοίρασης πληροφοριών μεταξύ προγραμματιστών σχετικά με την πρώτη εμφάνιση προβλημάτων ή σφαλμάτων. Για μεγάλα σε μέγεθος προγράμματα, τέτοιου είδους επικοινωνία για τυχόν ζητήματα στον κώδικα είναι πολύ σημαντικά για την επιτυχή και την αποδοτική, σχετικά με το κόστος τους, ανάπτυξη.
- Έλεγχος ενημερώσεων (update checks) : Το FindBugs ελέγχει αυτόματα για τυχόν νέες εκδόσεις που κυκλοφόρησαν. Αυτή η δυνατότητα είναι δυνατό να απενεργοποιηθεί πολύ εύκολα. Επίσης, μέσω των ενημερώσεων γίνεται μια άτυπη καταγραφεί του αριθμού των ενεργών χρηστών του εργαλείου.
- Πρόσθετα (plugins) : Η έκδοση 2.0 του εργαλείου διευκολύνει τον καθορισμό προσθέτων που παρέχουν διάφορες δυνατότητες, καθώς και την ενσωμάτωση και εγκατάστασή τους στο FindBugs.
- Η εντολή *fb* (fb command) : Αντίθετα με τη χρήση μιας συλλογής ανοργάνωτων δεσμών ενεργειών γραμμής εντολών (command line script), οι οποίες έχουν αναπτυχθεί κατά τη διάρκεια πολλών χρόνων για διάφορες εντολές του εργαλείου, πλέον υποστηρίζεται η χρήση της καθολικής εντολής *fb*:
    - fb analyze – Ξεκινά την ανάλυση από το FindBugs
    - fb gui – Ξεκινά τη γραφική διεπαφή χρήστη (GUI) του FindBugs
    - fb list – Δημιουργεί μια λίστα ζητημάτων από το αρχείο ανάλυσης του FindBugs
    - fb help – Δημιουργεί μια λίστα από τις διαθέσιμες εντολές της γραμμής εντολών

  Τα πρόσθετα μπορούν να χρησιμοποιηθούν ώστε να επεκτείνουν τις εντολές που υποστηρίζει η εντολή *fb*.

- Νέα μοτίβα σφαλμάτων και ανιχνευτές, καθώς και βελτιωμένη ακρίβεια.
- Βελτιωμένη απόδοση (improved performance) : Στην έκδοση 2.0 του εργαλείου έχει επιτευχθεί βελτίωση της τάξης του 10% σε πολλά μετροπρογράμματα (benchmarks), ωστόσο ορισμένοι χρήστες έχουν παρατηρήσει μειώσεις στην απόδοση που ερευνώνται από τους δημιουργούς του εργαλείου.
- Υποστήριξη Guarva (Guarva support) : Σε συνεργασία με τον Kevin Bourrillion παρέχεται επιπλέον υποστήριξη στη βιβλιοθήκη Guarva, καθώς και αναγνώριση πολλών κοινών ζητημάτων λάθος χρήσης.

- Υποστήριξη JSR-305(JSR-305 support) : Παροχή βελτιωμένης ανίχνευσης προβλημάτων που ταυτοποιούνται από το JSR-305, καθώς και βελτιωμένη ακρίβεια και απόδοση της ανάλυσης.

Πρόσθετα (Plugins) :

Το FindBugs υποστηρίζει την δημιουργία και εγκατάσταση προσθέτων για την επέκταση των δυνατοτήτων του, μια διαδικασία η οποία έγινε ακόμα πιο εύκολη στην έκδοση 2.0. Υπάρχουν πολλά πρόσθετα ήδη και πολλά περιλαμβάνονται με τα αρχεία του εργαλείου (κάποια ενεργοποιημένα εξ' αρχής και κάποια όχι). Ορισμένες βασικές κατηγορίες που περιλαμβάνονται είναι οι παρακάτω:

- Πρόσθετα νέφους (cloud plugins) : Προσφέρουν εύκολους τρόπους για τη διατήρηση και διαμοίραση πληροφοριών σχετικά με θέματα που συναντώνται κατά την ανάλυση (πρώτη εμφάνιση ζητήματος, σοβαρότητα και επικινδυνότητα, σχετικά σχόλια και παρατηρήσεις)
    - bugCollectionCloud: Αποθηκεύει αξιολογήσεις ζητημάτων σε XML
    - findbugsCommunalCloud: Αποθηκεύει αξιολογήσεις ζητημάτων στο κοινό νέφος που φιλοξενείται στο *findbugs.appspot.com*
    - jdbcCloudClient: Ένα παλαιότερο και υποτιμημένο πλέον νέφος το οποίο αποθήκευε πληροφορίες σε μια βάση δεδομένων SQL
- noUpdateChecks: Απενεργοποιεί τις αυτόματες ενημερώσεις και μέτρηση ενεργών χρηστών.
- poweruser: Προσφέρει ορισμένες επιπλέον εντολές για την εντολή *fb*. Οι εντολές αυτές χρησιμοποιούνται κυρίως από άτομα εκτός της ομάδας ανάπτυξης του FindBugs.
- Πρόσθετα συμπλήρωσης αναφορών σφάλματος (Bug filing plugins) : Τα πρόσθετα αυτά βοηθούν στη συμπλήρωση αναφορών σφάλματος για ζητήματα που εντοπίζονται. Το πλαίσιο των αναφορών αυτών έχει σχεδιαστεί ώστε να είναι δυνατή η επέκτασή του σε άλλα συστήματα αναφοράς σφαλμάτων.

5.6) Cobertura

Περίληψη:

Το Cobertura είναι ένα δωρεάν εργαλείο γραμμένο σε Java, το οποίο υπολογίζει το ποσοστό του κώδικα ενός προγράμματος γραμμένου σε Java ο οποίος έχει αξιολογηθεί από ελέγχους, καθώς και ποια τμήματα του κώδικα χρειάζονται επιπλέον ελέγχους. Η λειτουργία του βασίζεται στο *jcoverage*. Τα βασικότερα χαρακτηριστικά του εργαλείου είναι:

- Δυνατότητα εκτέλεσης από τη γραμμή εντολών ή από το *ant*
- Ελέγχει και παρακολουθεί κώδικα γραμμένο σε Java, μετά τη μεταγλώττιση
- Παράγει αναφορές σε μορφή HTML ή XML
- Δείχνει το ποσοστό κάλυψης για γραμμές και διακλαδώσεις για κάθε κλάση, για κάθε πακέτο και συνολικά για τον κώδικα
- Δείχνει την κυκλωματική πολυπλοκότητα κώδικα (cyclomatic code complexity) McCabe για κάθε κλάση, τον μέσο όρο της κυκλωματικής πολυπλοκότητας για κάθε πακέτο και συνολικά για τον κώδικα
- Δυνατότητα ταξινόμησης (αύξουσα ή φθίνουσα ταξινόμηση) των αποτελεσμάτων σε HTML με βάση το όνομα κλάσης, το ποσοστό κάλυψης γραμμών κώδικα και το ποσοστό κάλυψης διακλαδώσεων

Το Cobertura αρχικά δημιουργήθηκε από τον Mark Doliner και πλέον συντηρείται και επεκτείνεται από τους:

- Joakim Erdfelt
- John Lewis
- Grzegorz Lukasik
- Jiří Mareš
- Jeremy Thomerson

Το βασικό πλεονέκτημα σε αντίθεση με άλλα εργαλεία αξιολόγησης κώδικα είναι η ευκολία ανάγνωσης, επεξεργασίας και αξιολόγησης των αποτελεσμάτων του, καθώς και η ευκολία εγκατάστασης και εκτέλεσης του εργαλείου. Για την εκτέλεση του Cobertura απαιτείται η έκδοση Java 5, καθώς και διάφορα άλλα αρχεία και πακέτα, τα οποία όμως περιέχονται στα αρχεία του Cobertura.

Άδεια Χρήσης:

Τα *ant tasks* του Cobertura κυκλοφορούν υπό την άδεια χρήσης *Apache Software License, Version 1.1*, ενώ το υπόλοιπο εργαλείο κυκλοφορεί υπό την άδεια χρήσης *GNU General Public License, Version 2.0*.

Εργαλεία κάλυψης (coverage tools) :

Ανεξάρτητα από την πιστή εφαρμογή δομημένων και σωστά οργανωμένων μεθοδολογιών κατά την ανάπτυξη κώδικα, δεν είναι δυνατόν να εξασφαλιστεί η διεξοδική δοκιμή του κώδικα των προγραμμάτων, συνήθως λόγω του τεράστιου όγκου και της πολύ μεγάλης πολυπλοκότητάς τους. Συνεπώς, η χρήση αυτοματοποιημένων εργαλείων δοκιμής, ελέγχου, αξιολόγησης καθώς και παρακολούθησης του ποσοστού κάλυψης των ελέγχων (εργαλεία κάλυψης) είναι απαραίτητη.

Πλεονεκτήματα από τη χρήση του Cobertura :

Το Cobertura λόγω της απλότητας στη χρήση, της μικρής πολυπλοκότητας και του γεγονότος ότι είναι δωρεάν συμβάλλει τα μέγιστα στη διαδικασία ανάπτυξης κώδικα. Εστιάζει στην παραγωγή και παρουσίαση πληροφοριών σχετικά με τα τμήματα του κώδικα που έχουν ελεγχθεί και δοκιμαστεί ήδη και κυρίως για αυτά τα οποία δεν έχουν ελεγχθεί ακόμα.

Κατ' αρχάς, το εργαλείο προσθέτει στην ενορχήστρωση στον δυαδικό κώδικα της Java επειδή αυτή η προσέγγιση προσφέρει μεγαλύτερη ταχύτητα κατά την ανάλυση και αποφεύγεται η διπλή μεταγλώττιση του κώδικα. Επιπλέον, το Cobertura είναι εύκολο να ενσωματωθεί με το *Apache Ant*. Υπάρχουν εξειδικευμένοι ορισμοί που μπορούν να χρησιμοποιηθούν με το *Ant*. Είναι δυνατή και εύκολη η ενορχήστρωση είτε μιας κλάσης είτε ενός ολόκληρου προγράμματος με το Cobertura, ανάλογα με τις ανάγκες σε κάθε περίπτωση. Τέλος, Το Cobertura είναι εντελώς δωρεάν και δεν υπάρχουν ζητήματα με περιόδους δοκιμής ή δοκιμαστικές εκδόσεις. Η απλότητα του εργαλείου και ειδικά η ευκολία εγκατάστασης και εκτέλεσής του το καθιστούν μια πολύ καλή επιλογή για την παρακολούθηση του ποσοστού κάλυψης των ελέγχων για τα επιμέρους τμήματα οποιουδήποτε προγράμματος Java.

Προσθήκη εργασιών (tasks) του Cobertura στο Ant:

Το Cobertura ενσωματώνεται ομαλά με το Ant και έχει τη δυνατότητα να προσθέτει εργασίες του χρήστη (custom tasks). Ωστόσο, προτού χρησιμοποιηθούν οι εργασίες χρήστη πρέπει να ορισθούν στο αρχείο κατασκευής (build file) του Ant, το οποίο εξ' ορισμού έχει όνομα *build.xml*. Ο ορισμός αυτός επιτυγχάνεται με την εκτέλεση της εντολής *taskdef* όπως φαίνεται στη συνέχεια:

```
<taskdef classpath="cobertura.jar" resource="tasks.properties"/>
```

Προσθήκη ενορχήστρωσης στις κλάσεις:

Το Cobertura δουλεύει με την ενσωμάτωση οδηγιών ενορχήστρωσης στις μεταγλωττισμένες κλάσεις της Java. Όταν η εικονική μηχανή της Java (Java Virtual Machine) συναντά τις οδηγίες αυτές, ο επιπλέον κώδικας που έχει εισαχθεί αυξάνει ορισμένους μετρητές καθιστώντας δυνατή την παρακολούθηση των διαφόρων τμημάτων κώδικα που έχουν συναντηθεί και αυτών που δεν έχουν μέχρι στιγμής. Η γενικότερη μορφή της εντολής εισαγωγής οδηγιών ενορχήστρωσης στο *Ant* είναι η εξής:

```
<cobertura-instrument todir="build/instrumented-classes">
        <fileset dir="build/classes">
                <include name="**/*.class"/>
        </fileset>
</cobertura-instrument>
```

Εκτέλεση εφαρμογών με ενορχήστρωση:

Αφότου έχει ολοκληρωθεί η ενορχήστρωση στις κλάσεις, είναι δυνατή η συνέχεια της δοκιμής των εφαρμογών μέσω της εργασίας (task) *JUnit*. Ωστόσο, είναι απαραίτητη η εισαγωγή μιας εγγραφής μονοπατιού κλάσης (classpath entry) για τις ενορχηστρωμένες κλάσεις πριν από τις αναφορές στις αντίστοιχες κλάσεις χωρίς ενορχήστρωση. Η γενική μορφή εισαγωγής είναι η εξής:

```
 <junit fork="yes">
        <classpath location="${build.instrumented.dir}"/>
        <classpath location="${build.classes.dir}"/>
   ...
</junit>
```

Αναφορές κάλυψης σε HTML (HTML coverage report):

Η βασική μορφή μιας αναφοράς κάλυψης σε HTML είναι η εξής:

```
<target name="coverage">
   <cobertura-report srcdir="${src.dir}" destdir="${build.coverage.dir}"/>
```

```
</target>
```

Αναφορές κάλυψης σε XML (XML coverage report):

Η βασική μορφή μιας αναφοράς κάλυψης σε XML είναι η εξής:

```
<target name="coverage">
   <cobertura-report format="xml" srcdir="${src.dir}" destdir="${build.coverage.dir}"/>
</target>
```

5.7) Checkstyle

Περίληψη:

Το Checkstyle είναι ένα δωρεάν εργαλείο ανάπτυξης κώδικα το οποίο βοηθά τους προγραμματιστές στην συγγραφή κώδικα Java και το οποίο τηρεί τα πρότυπα προγραμματισμού. Χρησιμοποιείται για την αυτοματοποίηση της διαδικασίας ελέγχου κώδικα γραμμένου σε Java, ώστε να μην χρειάζεται να γίνεται η τελευταία σημαντική αλλά και βαρετή διαδικασία χειροκίνητα από τους προγραμματιστές. Η χρήση του εργαλείου συνίσταται για τις περιπτώσεις όπου υπάρχει ανάγκη για την επιβολή και τήρηση ενός προγραμματιστικού προτύπου. Το Checkstyle τηρεί εξ' ορισμού τις συμβάσεις της *Sun*, ενώ έχει τη δυνατότητα να προσαρμόζεται και να ρυθμίζεται ανάλογα με τις εκάστοτε ανάγκες και το πρότυπο που πρέπει να ακολουθηθεί. Τέλος, μπορεί να ενσωματωθεί και να εκτελεστεί από τις εργασίες *Ant* και από τη γραμμή εντολών.

Έκδοση:

Η τρέχουσα έκδοση του εργαλείου είναι η 5.6 η οποία κυκλοφόρησε στις 18-9-2012.

Άδεια Χρήσης:

Το Checkstyle είναι δωρεάν εργαλείο και κυκλοφορεί υπό την άδεια χρήσης *GNU Library or Lesser General Public License version 2.0 (LGPLv2)*.

Εγκατάσταση και παραμετροποίηση

Μια εγκατάσταση του Checkstyle καθορίζει τα εξαρτήματα (modules) τα οποία θα συνδεθούν και εφαρμοστούν στον κώδικα Java. Τα εξαρτήματα είναι οργανωμένα σε μορφή δέντρου η ρίζα του οποίου είναι το εξάρτημα *Checker* και ακολουθούν σε επόμενο επίπεδο τα:

- *FileSetChecks*-> εξαρτήματα τα οποία δέχονται ένα σύνολο από αρχεία εισόδου και εξάγουν μηνύματα σφάλματος
- *Filters*-> εξαρτήματα τα οποία φιλτράρουν γεγονότα ελέγχου, όπως μηνύματα σφάλματος, για επαλήθευση
- *AuditListeners*-> εξαρτήματα τα οποία αναφέρουν αποδεκτά γεγονότα

Χρήση στις εργασίες Ant (Ant Tasks) :

Το Checkstyle έχει δοκιμαστεί στην έκδοση *Ant* 1.5 και πάνω. Ο πιο εύκολος τρόπος για την εγκατάσταση του εργαλείου είναι η συμπερίληψη του *checkstyle-5.6-all.jar* στο *classpath* του *Ant*. Στο αρχείο αυτό περιλαμβάνονται όλες οι κλάσεις που απαιτούνται για την εκτέλεση του Checkstyle. Για να επιτευχθεί αυτό πρέπει να εκτελεστεί η εξής εντολή *taskdef* για τη δήλωση:

```
<taskdef resource="checkstyletask.properties"
      classpath="/path/to/checkstyle-5.6-all.jar"/>
```

Χρήση από τη γραμμή εντολών (Command line) :

Ο πιο εύκολος τρόπος για την εγκατάσταση του εργαλείου είναι η συμπερίληψη του *checkstyle-5.6-all.jar* στο *classpath*. Η εντολή που πρέπει να εκτελεστεί ώστε να εκτελεστεί το εργαλείο από τη γραμμή εντολών είναι:

```
java -D<property>=<value>  \
    com.puppycrawl.tools.checkstyle.Main \
    -c <configurationFile> \
    [-f <format>] [-p <propertiesFile>] [-o <file>] \
    [-r <dir>] file...
```

Το Checkstyle θα επεξεργαστεί τα αρχεία που καθορίστηκαν και θα αναφέρει εξ' ορισμού τυχόν σφάλματα στην τυπική έξοδο (standard out) σε απλή μορφή. Το εργαλείο απαιτεί την ύπαρξη ενός αρχείου ρυθμίσεων (configuration file) σε XML το οποίο καθορίζει και ρυθμίζει τους ελέγχους που θα εκτελεστούν. Επιπλέον επιλογές και ορίσματα για την εκτέλεση από τη γραμμή εντολών είναι τα παρακάτω:

- -c configurationFile-> καθορίζει την τοποθεσία του αρχείου που ορίζει τα τμήματα των ρυθμίσεων. Η τοποθεσία μπορεί να είναι είτε μια τοποθεσία του συστήματος αρχείων (file system location) είτε ένα όνομα που περνιέται σαν όρισμα στη μέθοδο *ClassLoader.getResource()*.
- -f format-> καθορίζει τη μορφή της εξόδου. Οι επιλογές είναι "plain" για το DefaultLogger (προκαθορισμένη) και "xml" για το XMLLogger.
- -p propertiesFile-> καθορίζει ποιό αρχείο ιδιοτήτων θα χρησιμοποιηθεί.
- -o file->καθορίζει το αρχείο στο οποίο θα οδηγηθεί η έξοδος.
- -r dir-> καθορίζει τον κατάλογο ο οποίος θα διασχισθεί για τα πηγαία αρχεία Java.

Πρότυποι έλεγχοι (Standard checks) :

Οι πρότυποι έλεγχοι του Checkstyle αφορούν γενικά μοτίβα προγραμματισμού για τη Java και δεν απαιτούν επιπλέον εξωτερικές βιβλιοθήκες. Οι πρότυποι αυτοί έλεγχοι περιέχονται στη βασική διανομή του εργαλείου και είναι οι παρακάτω:

- **AnnotationUseStyle:** Έλεγχος του στιλ προγραμματισμού μέσω της χρήσης υποσημειώσεων
- **EmptyBlock:** Έλεγχος για κενά μπλοκ
- **VisibilityModifier:** Έλεγχος ορατότητας για μέλη των κλάσεων. Μόνο τελικά στατικά μέλη μπορούν να είναι δημόσια (public) ενώ όλα τα υπόλοιπα πρέπει να είναι ιδιωτικά (private), εκτός και αν έχει οριστεί το *protectedAllowed* ή το *packageAllowed*
- **ArrayTrailingComma:** Έλεγχος για την ύπαρξη υποδιαστολής (κόμμα) στο τέλος της αρχικοποίησης πινάκων
- **DuplicateCode:** Ανίχνευση διπλότυπου κώδικα ο οποίος έχει προκύψει από αντιγραφή/επικόλληση (Copy/Paste). Συνήθως μέσω της αντιγραφής/επικόλλησης κώδικα τα σφάλματα μεταφέρονται σε πολλαπλά σημεία και πρέπει να διορθωθούν πολλαπλές φορές, οδηγώντας σε μεγάλο κόστος συντήρησης. Συνήθως τα χαρακτηριστικά ενός τέτοιου ελέγχου (και του προγράμματος που τον πραγματοποιεί) είναι:
    - Ταχύτητα
    - Χαμηλή κατανάλωση μνήμης
    - Αποφυγή λανθασμένων αποτελεσμάτων

    - Υποστήριξη πολλαπλών γλωσσών
    - Υποστήριξη ασαφών ταιριασμάτων (σχόλια, κενά διαστήματα, μετονομασίες μεταβλητών, αλλαγή γραμμής)
- **StrictDuplicateCode:** Εκτέλεση σύγκρισης γραμμή-γραμμή (line-by-line) όλων των γραμμών κώδικα και αναφορά διπλότυπου κώδικα. Όλες οι δηλώσεις *import* στον κώδικα Java αγνοούνται, ενώ οποιαδήποτε άλλη γραμμή (συμπεριλαμβανομένων των κενών χαρακτήρων μεταξύ μεθόδων και του javadoc) λαμβάνονται υπόψιν, για αυτό και καλείται αυστηρός (strict). Η μέθοδος αυτή είναι αρκετή ώστε να μπορεί να διευκολύνει τον έλεγχο μεγάλων τμημάτων κώδικα σε αποδεκτό χρόνο (συνήθως λεπτά). Καταναλώνει πολύ λίγη μνήμη, ενώ λανθασμένα μηνύματα σφάλματος είναι αδύνατα. Ωστόσο, παρότι υποστηρίζει αρκετές γλώσσες δεν επιτρέπει ασαφή ταιριάσματα (λόγω του αυστηρού τρόπου ελέγχου)
- **Header:** Έλεγχος πηγαίων αρχείων για τον εντοπισμό καθορισμένων επικεφαλίδων με τις οποίες πρέπει να ξεκινούν. Η ιδιότητα *headerFile* καθορίζει ένα αρχείο το οποίο περιέχει την απαιτούμενη επικεφαλίδα. Εναλλακτικά, ο καθορισμός της επικεφαλίδας μπορεί να γίνει απ' ευθείας στην ιδιότητα *headerFile*, χωρίς τη χρήση εξωτερικού αρχείου
- **AvoidStarImport:** Έλεγχος για τη διασφάλιση της μη χρησιμοποίησης της σημείωσης * στις δηλώσεις *import*
- **JavadocPackage:** Ελέγχει αν κάθε πακέτο Java έχει ένα αρχείο Javadoc το οποίο χρησιμοποιείται για σχόλια. Εξ' ορισμού επιτρέπεται μόνο ένα αρχείο *package-info.java*, αλλά είναι δυνατή η ρύθμιση ώστε να επιτρέπεται και αρχείο *package.html*. Σε περίπτωση ύπαρξης και των δύο αρχείων θα αναφερθεί ένα σφάλμα, καθώς αυτό δεν επιτρέπεται από το εργαλείο Javadoc
- **BooleanExpressionComplexity:** Περιορίζει το πλήθος των τελεστών `&&`, `||`, `&`, `|` και `^` σε μια έκφραση. Πολλαπλές συνθήκες και τελεστές οδηγούν σε δυσανάγνωστο κώδικα ο οποίος είναι δύσκολο να ελεγχθεί και συντηρηθεί
- **NewLineAtEndOfFile:** Έλεγχος για την ύπαρξη χαρακτήρα νέας γραμμής (new line) στο τέλος των αρχείων, καθώς για τα πηγαία αρχεία και για τα αρχεία κειμένου συνίσταται
- **ModifierOrder:** Έλεγχος για την τήρηση της σειράς των τροποποιητών (modifiers), σύμφωνα με τις προτάσεις των προτύπων. Η σωστή σειρά είναι:
    - Public
    - Protected
    - Private
    - Abstract
    - Static
    - Final
    - Transient
    - Volatile
    - Synchronized
    - Native
    - Strictfp
- **RegexpSingleline:** Έλεγχος για την ανίχνευση μονών γραμμών οι οποίες ταιριάζουν με μια δοσμένη κανονική έκφραση (regular expression). Είναι συμβατό και δουλεύει για οποιονδήποτε τύπο αρχείου
- **ExecutableStatementCount:** Περιορίζει το πλήθος των εκτελέσιμων δηλώσεων σε ένα καθορισμένο όριο
- **GenericWhitespace:** Έλεγχος για την ορθότητα των κενών χαρακτήρων γύρω από τους τελεστές < και > σύμφωνα με την τυπική σύμβαση:

List<Integer> x = new ArrayList<Integer>();

```
List<List<Integer>> y = new ArrayList<List<Integer>>();
```

5.8) YASCA

Περίληψη:

Το Yasca είναι ένα δωρεάν ανοικτού κώδικα εργαλείο που δημιουργήθηκε για να βοηθά τους προγραμματιστές να διασφαλίζουν ότι οι εφαρμογές τους ακολουθούν πρότυπα υψηλής ποιότητας. Το εργαλείο σχετίζεται με την αξιολόγησης ποιότητας (quality assessment) και με τις σαρώσεις αδυναμιών (vulnerability scanning) αλλά δεν υποκαθιστά κανένα από τα δύο. Το Yasca διανέμεται τόσο με δικούς του σαρωτές όσο και με ενσωματωμένους σαρωτές τρίτων (Jlint,antic,Lint4j,FindBugs και PMD). Συνολικά, το Yasca λειτουργεί ως αθροιστικό εργαλείο "και κάτι ακόμα", χρησιμοποιώντας άλλα δωρεάν ανοιχτού κώδικα εργαλεία για τον εντοπισμό σφαλμάτων και προσθέτοντας δικά του πρόσθετα για να καλύψει τα όποια κενά υπάρχουν. Συνεπώς το εργαλείο μπορεί να χρησιμοποιηθεί ως βάση για τη λειτουργία και συγκέντρωση αναφορών και δυνατοτήτων πολλών άλλων δωρεάν εργαλείων αξιολόγησης και σάρωσης κώδικα, κάνοντας τη χρήση όλων αυτών πολύ εύκολη.

Άδεια χρήσης:

Το Yasca κυκλοφορεί και διανέμεται δωρεάν κάτω από την άδεια χρήσης *BSD license*.

Τρέχουσα έκδοση:

Η τελευταία σταθερή έκδοση του εργαλείου είναι η 2.2 η οποία κυκλοφόρησε στις 4/6/2010. Σε δοκιμαστικό στάδιο βρίσκεται η έκδοση 3.04 η οποία είναι διαθέσιμη από τις 22/12/2012.

Δημιουργός:

Το Yasca έχει δημιουργήσει και συνεχίζει να αναπτύσσει και να συντηρεί ο Michael Scovetta. Το εργαλείο είναι γραμμένο στις γλώσσες PHP και Java και υποστηρίζει όλες τις δυνατές πλατφόρμες (cross-platform)

Βασικές Δυνατότητες:

- Σάρωση πηγαίου κώδικα για αδυναμίες (vulnerability scanning) για τις γλώσσες:
  - Java
  - C/C++
  - HTML
  - JavaScript
  - ASP
  - ColdFusion
  - PHP
  - COBOL
  - .NET(VB.NET, C#, ASP.NET)
  - CSS

    - Perl
    - Python
    - Visual Basic
    - Raw HTTP Traffic
- Δυνατότητα ενσωμάτωσης αρκετών άλλων εργαλείων:
    - FindBugs
    - PMD
    - Jlint
    - JavaScript Lint
    - PHPLint
    - Cppcheck
    - ClamAV
    - RATS
    - Pixy
- Δυνατότητα λειτουργίας τόσο σε περιβάλλον Windows όσο και σε Linux
- Ευελιξία στο σχεδιασμό που επιτρέπει εύκολη επέκταση με έτοιμα πρόσθετα ή πρόσθετα του χρήστη

Πρόσθετα:

Το Yasca χρησιμοποιεί ξεχωριστά πρόσθετα για να επιτύχει την σάρωση των αρχείων. Αυτός ο σχεδιασμός επιτρέπει ευελιξία και εύκολη επέκταση των δυνατοτήτων του εργαλείου, ανάλογα με τις ανάγκες. Είναι δυνατή η δημιουργία και διανομή πακέτων προσθέτων (plugin packs) για σάρωση και ανίχνευση συγκεκριμένων τύπων αρχείων ή ζητημάτων αδυναμιών, απόδοσης ή πολυπλοκότητας. Τα εργαλείο περιέχει δύο βασικές κατηγορίες προσθέτων, τα εσωτερικά (internal) και τα εξωτερικά (external) παρότι δεν υπάρχει διαχωρισμός αναφορικά με τη διαχείρισή τους. Ως εσωτερικά αναφέρονται αυτά που είναι αυτόνομα και επεκτείνουν την κλάση προσθέτων, ενώ ως εξωτερικά αναφέρονται αυτά που απαιτούν και επιπλέον λογισμικό για την εκτέλεσή τους. Τα εξωτερικά πρόσθετα που περιλαμβάνονται στη διανομή του εργαλείου είναι :

- Grep-> Χρησιμοποιεί εξωτερικά αρχεία GREP για τη σάρωση αρχείων για απλά μοτίβα σφαλμάτων και αδυναμιών
- PMD-> Χρησιμοποιεί το PMD για την ανάλυση και σάρωση πηγαίου κώδικα Java (και JSP) για διάφορα ζητήματα
- JLint-> Χρησιμοποιεί το JLint για τη σάρωση Java αρχείων *.class* για ζητήματα
- antic-> Χρησιμοποιεί το antiC για τη σάρωση πηγαίων αρχείων Java, C και C++ για ζητήματα
- FindBugs-> Χρησιμοποιεί το FindBugs για τη σάρωση Java αρχείων *class* και *Jar* για ζητήματα
- Lint4J-> Χρησιμοποιεί το Lint4J για τη σάρωση Java αρχείων *.class* για ζητήματα

Πρόσθετο GREP:

Το πρόσθετο GREP χρησιμοποιεί εξωτερικά αρχεία (*.grep) που βρίσκονται στο φάκελο προσθέτων για τη σάρωση των αρχείων για μοτίβα σφαλμάτων και ζητημάτων.

Πρόσθετο PMD:

Το πρόσθετο PMD χρησιμοποιεί το ανοιχτού κώδικα εργαλείο PMD για την ανάλυση και μερική μεταγλώττιση πηγαίων αρχείων Java και JSP και στη συνέχεια σαρώνει το αφηρημένο δέντρο σύνταξης που προκύπτει για συγκεκριμένα μοτίβα. Το πρόσθετο έχει πολλές δυνατότητες αλλά αναφέρεται και λειτουργεί μόνο για πηγαίο κώδικα Java (και JSP).

Πρόσθετο JLint:

Το πρόσθετο JLint χρησιμοποιεί το ανοιχτού κώδικα εργαλείο JLint 3.0 για τη σάρωση μεταγλωττισμένων αρχείων *.class* για την ανίχνευση σφαλμάτων, ασυνεπειών και προβλημάτων συγχρονισμού.

Πρόσθετο antiC:

Το πρόσθετο antiC χρησιμοποιεί το ανοιχτού κώδικα εργαλείο antiC 1.11.1 για τη σάρωση πηγαίου κώδικα Java, C και C++ για σφάλματα, ασυνέπειες και προβλήματα συγχρονισμού.

5.9) Sonar

Περίληψη:

Το Sonar είναι μια δωρεάν ανοιχτού κώδικα πλατφόρμα γραμμένη σε γλώσσα Java για την αξιολόγηση λογισμικού. Το Sonar χρησιμοποιεί διάφορα εργαλεία στατικής ανάλυσης κώδικα όπως το Checkstyle, το PMD και το FindBugs για την εξαγωγή μετρικών του λογισμικού, οι οποίες στη συνέχεια χρησιμοποιούνται για τη βελτίωση της ποιότητας του λογισμικού.

Βασικά χαρακτηριστικά:

- Παροχή αναφορών για:
  - Διπλότυπο κώδικα (duplicated code)
  - Πρότυπα προγραμματισμού (coding standards)
  - Έλεγχο μονάδων (unit tests)
  - Κάλυψη κώδικα (code coverage)
  - Πολυπλοκότητα κώδικα (complex code)
  - Πιθανά σφάλματα (potential bugs)
  - Σχόλια (comments)
  - Σχεδιασμό και αρχιτεκτονική (design and architecture)
- Κύρια γλώσσα που υποστηρίζεται είναι η Java. Οι υπόλοιπες γλώσσες προγραμματισμού υποστηρίζονται μέσω προσθέτων. Υπάρχουν αρκετές ανοιχτού κώδικα επεκτάσεις που υποστηρίζουν τις γλώσσες:
  - C
  - C#
  - PHP
  - Flex
  - Groovy
  - JavaScript
  - Python
  - PL/SQL
  - Cobol
  - Visual Basic 6
- Ενσωματώνεται εύκολα με το Maven, το Ant καθώς και με εργαλεία συνεχούς ενσωμάτωσης (Atlassian Bamboo, Jenkins, Hudson)
- Δυνατότητα επέκτασης των δυνατοτήτων του εργαλείου μέσω προσθέτων
- Μετρικές σχεδιασμού και αρχιτεκτονικής των προγραμμάτων
- Υλοποίηση της μεθοδολογίας SQALE

Άδεια χρήσης:

Το Sonar κυκλοφορεί υπό την άδεια χρήσης *GNU Lesser GPL License,* έκδοση 3 υπό την ελβετική νομοθεσία.

Τρέχουσα έκδοση:

Η τρέχουσα έκδοση του εργαλείου είναι η 3.5.1 η οποία κυκλοφόρησε στις 3/4/2013.

Σχέδιο δράσης (roadmap) :

Τα επόμενα χαρακτηριστικά που σχεδιάζονται και υλοποιούνται για να ενσωματωθούν στο εργαλείο είναι τα εξής:

- Οι δύο αντιλήψεις για την παραβίαση (violation) και επισκόπηση (review) θα ενοποιηθούν σε μια αντίληψη ζητήματος. Επιπλέον, θα γίνει δυνατή η απ' ευθείας δημιουργία φίλτρων για ζητήματα και ομαδικές αλλαγές στη λίστα ζητημάτων
- Ο κύκλος ζωής (lifecycle) / πρόοδος εργασίας (workflow) των ζητημάτων θα γίνουν πλήρως προσαρμόσιμα στις εκάστοτε ανάγκες
- Υποστήριξη πλοήγησης δια μέσω πηγών
- Υπολογισμός μετρικών *Robert C. Martin O.O.* για το συνολικό πρόγραμμα, λαμβάνοντας υπόψιν τις εξαρτήσεις μεταξύ βιβλιοθηκών και εφαρμογών κατά την ανάλυση
- Υποστήριξη μετρικών Code Churn για την ανίχνευση ενημερωμένου πηγαίου κώδικα
- Πλήρης υποστήριξη πολυγλωσσικών προγραμμάτων

Προσθήκες στο Sonar το 2013:

Η λίστα με τα ζητήματα και χαρακτηριστικά που θα απασχολήσουν την ομάδα ανάπτυξης το 2013 ακολουθεί:

- **Χαρτογραφία (cartography) :** Το πιο φιλόδοξο έργο για το εργαλείο. Ο όρος ομαδοποιεί αρκετά χαρακτηριστικά βασισμένα σε εξαρτήσεις μεταξύ μεθόδων, ιδιοτήτων, κλάσεων, αρχείων, τμημάτων κώδικα, έργων, ομάδων και παραρτημάτων. Οι αρχικές περιπτώσεις που θα καλυφθούν σε αυτή την κατεύθυνση είναι:
    - Πλοήγηση δια μέσω πηγαίων κωδίκων (cross-sources navigation) : Δυνατότητα επιλογής μιας κλήσης μεθόδου της διεπαφής χρήστη (user interface) για την προβολή της δήλωσής της, επιλογή ενός αναγνωριστικού για την προβολή της δήλωσής του, την επιλογή μιας ντιρεκτίβας (directive) προεπεξεργασίας COBOL COPY για την προβολή του περιεχομένου της και την επιλογή της δήλωσης μιας συνάρτησης C για την προβολή πληροφοριών σχετικά με το χρόνο χρήσης της
    - Δυνατότητα ανίχνευσης των αρχείων τα οποία περιέχουν αρχεία βιβλιοθηκών C ή COBOL *Copybook*

- **Γραμμές που καλύπτονται από έλεγχο μιας μονάδας:** Δυνατότητα παροχής πληροφοριών σχετικά με το πλήθος καθώς και σχετικά με τους ελέγχους μονάδων που καλύπτουν συγκεκριμένες γραμμές και κομμάτια κώδικα. Επίσης, παροχή πληροφοριών σχετικά με τις γραμμές και τμήματα κώδικα που καλύπτονται από συγκεκριμένους ελέγχους μονάδων. Οι πληροφορίες αυτές θα καταστήσουν δυνατή τη δημιουργία νέων μετρικών για τους ελέγχους μονάδων, οι οποίες θα αναφέρονται στη συνοχή των μονάδων αυτών

- **Ζητήματα (issues) :** Στο Sonar μέχρι στιγμής υπάρχουν δύο κυρίαρχες αντιλήψεις, οι παραβιάσεις (violations) και οι επισκοπήσεις (reviews), οι οποίες θα ενοποιηθούν και θα δημιουργήσουν τα ζητήματα (issues). Επιπλέον, θα προστεθεί η δυνατότητα ομαδικής αλλαγής ζητημάτων. Παλιά και κλειστά πλέον ζητήματα δεν θα εκκαθαριστούν ώστε να ανιχνεύεται παλαιότερη δραστηριότητα
- **Ενσωμάτωση των προσθέτων των C και C++:** Τα δύο πρόσθετα για τις γλώσσες C και C++ είναι ήδη αρκετά ανεπτυγμένα και θα ενσωματωθούν σε ένα. Ο στόχος του εργαλείου είναι να προσφέρει ενιαία υποστήριξη στις δύο γλώσσες και για αυτό τα πρόσθετα έχουν δημιουργηθεί από το μηδέν και δεν στηρίζονται σε εργαλεία τρίτων
- **Java:** Η υποστήριξη της Java στηρίζεται ακόμα σε μεγάλο βαθμό στα εργαλεία PMD και Checkstyle, ωστόσο γίνεται προσπάθεια μεταφοράς και υιοθέτησης μεγάλου αριθμού κανόνων στην τεχνική στοίβα του Sonar (βλέπε SSLR).
- **Ικανότητα ανίχνευσης και επισκόπησης ανανεωμένου κώδικα:** Με το Sonar είναι ήδη δυνατό να ανιχνεύεται η κάλυψη κώδικα για νέο κώδικα και η ανίχνευση νέων εισερχόμενων παραβιάσεων. Συνεπώς, είναι απαραίτητη η δυνατότητα ανίχνευσης νέου κώδικα για τη χειρωνακτική επισκόπησή του από τον προγραμματιστή, ώστε το Sonar να γίνει ένα πρότυπο εργαλείο επισκόπησης κώδικα
- **Sonar Eclipse, υποστήριξη οριακής (incremental) τοπικής ανάλυσης:** Καθώς έχει ήδη υλοποιηθεί (μετά την ανακατασκευή του προσθέτου Eclipse) και υποστηρίζεται πλήρως η τοπική ανάλυση, ο στόχος είναι να υπάρχει δυνατότητα χρήσης της για μεγάλα προγράμματα μέσω της χρήσης οριακής (incremental) και διαφορικής (differential) ανάλυσης
- **Έτος ωρίμανσης για το SSLR:** Η τεχνολογία ανάλυσης κώδικα που χρησιμοποιείται στο Sonar έγινε ανοιχτού κώδικα για να υπάρχει δυνατότητα επαναχρησιμοποίησης στα περισσότερα πρόσθετα γλωσσών του Sonar, είτε δωρεάν είτε εμπορικά: *Javascript, Python, Java, Erlang, Cobol, C, C++, Flex, C#, PL/I, PL/SQL*. Ο στόχος για το SSLR είναι να μπορεί να παρέχει έτοιμες για χρήση προγραμματιζόμενες διεπαφές εφαρμογών (APIs), ώστε να είναι δυνατή η δημιουργία καλά διατυπωμένων AST για την υποστήριξη οριακής ανάλυσης (incremental parsing) και ο εμπλουτισμός του AST με σημασιολογικές πληροφορίες (semantic information)

Βασικές περιπτώσεις χρήσης του Sonar:

Οι βασικές περιπτώσεις όπου συνίσταται η χρήση του Sonar είναι οι παρακάτω:

- Έλεγχοι μονάδων (Unit Tests)
- Έλεγχοι ενσωμάτωσης (Integration Tests)
- Διπλότυπα (Duplication)
- Αξιολόγηση κώδικα (Code Review)
- Διαφορικές υπηρεσίες (Differential Services)
- Eclipse
- Μοντέλο ποιότητας (Quality Model)
- Αρχική σελίδα (Home Page)
- Πολλαπλές γλώσσες (Multi-languages)
- Κάλυψη κώδικα (Code Coverage)
- Διαρκής επιθεώρηση (Continuous Inspection)
- Σχεδιασμός κλάσεων (Class Design)
- Σχεδιασμός πακέτων (Package Design)
- Τεχνικό χρέος (Technical debt)
- Πιστοποίηση και εξουσιοδότηση (Authentication and Authorization)
- Sonar Time Machine
- Κύκλοι πακέτων (Package cycles)

- Ζευγάρωμα και συνεκτικότητα (Coupling and Cohesion)
- Διαχείριση Portofolio (Portfolio management)
- Sonar Light
- Συνεχής ενσωμάτωση (Continuous Integration)

5.10) Static Analysis Tool

Περίληψη:

Το εργαλείο “static analysis tool” είναι ένα εργαλείο στατικής ανάλυσης κώδικα το οποίο αναπτύχθηκε στο πλαίσιο της παρούσης διπλωματικής εργασίας. Το εργαλείο είναι γραμμένο σε γλώσσα C και εξάγει αποτελέσματα για των πηγαίο κώδικα γραμμένο σε γλώσσα C που σαρώθηκε αναφορικά με τις παρακάτω μετρικές:

- Συνολικές γραμμές κώδικα (physical lines of code)
- Γραμμές σχολίων (comment lines)
- Αναλογία γραμμών σχολίων/συνολικών γραμμών (comment ratio)
- Μετρικές Halstead (Halstead metrics)
- Μετρική ABC (ABC metric)
- Πλήθος κενών γραμμών (blank lines)
- Λογικές γραμμές κώδικα (logical lines of code)

Μετά την σάρωση τα αποτελέσματα εξάγονται σε κοινή αναφορά (στο αρχείο “report.txt”) και δημιουργούνται τα αρχεία καταγραφής (log files) “log_file.txt” και “log_file2.txt” τα οποία καταγράφουν τη ροή εκτέλεσης για το κύριο πρόγραμμα και τις επιμέρους συναρτήσεις για τις μετρικές αντίστοιχα.

Βασικές δυνατότητες:

Συνολικές γραμμές κώδικα:

Σε αυτήν τη μετρική περιέχονται όλες οι γραμμές που περιέχονται στα πηγαία αρχεία εισόδου (όχι οι κενές γραμμές). Δεν γίνεται διάκριση μεταξύ γραμμών κώδικα, σχολίων ή συνδυασμούς αυτών.

Γραμμές σχολίων:

Σε αυτήν τη μετρική περιέχονται όλες οι γραμμές σχολίων που αντιστοιχούν στις αποδεκτές μορφές σχολίων για τη γλώσσα C. Οι μορφές αυτές είναι:

- Οποιαδήποτε γραμμή περιέχει τους χαρακτήρες “//” είτε στην αρχή είτε σε ενδιάμεσο σημείο, καταγράφεται ως γραμμή σχολίων. Για τη συγκεκριμένη μορφή σχολίων, οτιδήποτε βρίσκεται μετά τους ειδικούς χαρακτήρες και μέχρι το τέλος της γραμμής θεωρείται ως σχόλιο.
- Οποιεσδήποτε γραμμές περιέχονται ανάμεσα στους χαρακτήρες “/*” και “*/” καταγράφονται ως γραμμές σχολίων. Για τη συγκεκριμένη μορφή σχολίων, οι χαρακτήρες έναρξης και τέλους των σχολίων μπορούν να βρίσκονται στην ίδια ή σε διαφορετικές γραμμές. Όσες γραμμές περιέχονται ενδιάμεσα μετράνε ως σχόλια.

Παρατηρήσεις:

Στην περίπτωση που εμφανιστεί η πρώτη περίπτωση και στη συνέχεια η δεύτερη (στην ίδια γραμμή) η γραμμή μετράει μόνο μια φορά στο σύνολο σχολίων.

Στην περίπτωση που εμφανιστεί η δεύτερη περίπτωση και περιέχεται στις γραμμές που περικλείουν οι ειδικοί χαρακτήρες της, οι χαρακτήρες τις πρώτης περίπτωσης, η γραμμή μετράει μόνο μια φορά.

Στην περίπτωση που υπάρχει κώδικας σε μια γραμμή και στη συνέχεια εμφανίζεται η πρώτη, η δεύτερη ή και οι δύο μορφές σχολίων, η γραμμή μετράει ως σχόλια (μια φορά).

Αναλογία γραμμών σχολίων/συνολικών γραμμών:

Σε αυτήν τη μετρική εξάγεται το ποσοστό που αντιστοιχεί στις γραμμές σχολίων επί του συνολικού αριθμού γραμμών των αρχείων εισόδου.

Μετρικές Halstead:

Οι μετρικές Halstead εξάγονται από ένα σύνολο 4 βασικών χαρακτηριστικών του κώδικα:

- Πλήθος διακριτών τελεστών (n1)
- Πλήθος διακριτών τελούμενων (n2)
- Συνολικό πλήθος τελεστών (N1)
- Συνολικό πλήθος τελούμενων (N2)

Από αυτές τις βασικές μεταβλητές υπολογίζονται οι μετρικές που ακολουθούν σύμφωνα με συγκεκριμένους τύπους:

- **Λεξιλόγιο προγράμματος (Program vocabulary) :** n=n1+n2
- **Μήκος προγράμματος (Program length) :** N=N1+N2
- **Υπολογισμένο μήκος προγράμματος (Calculated program length) :** N'=n1*log2 (n1) + n2*log2 (n2)
- **Όγκος (Volume) :** V=N*log2 (n)
- **Δυσκολία (Difficulty) :** D=(n1/2) * (N2/n2), αλλιώς (1/L)
- **Επίπεδο προγράμματος (Program level) :** L= (2/n1) * (n2/N2), αλλιώς (1/D)
- **Προσπάθεια (Effort) :** E=D*V
- **Χρόνος που χρειάστηκε (Time required to program) :** T=E/18 (δευτερόλεπτα)
- **Πλήθος σφαλμάτων που παραδόθηκαν (Number of delivered bugs):** B= (E^ (2/3)) / (3000)
- **Έξυπνο περιεχόμενο (Intelligent content) :** I=V/D
- **Διανοητική προσπάθεια (Interllectual effort) :** E'= (N') *log2 (n/L)

Μετρική ABC:

Η μετρική αυτή εξάγεται από 3 βασικά χαρακτηριστικά του κώδικα:

- A (assignments) : Πλήθος αναθέσεων που υπάρχουν στον κώδικα

- B (branches) : Πλήθος διακλαδώσεων που υπάρχουν στον κώδικα
- C (conditions) : Πλήθος συνθηκών που εξετάζονται στον κώδικα

Η τελική μορφή παρουσιάζεται ως τριάδα αριθμών και ως βαθμωτός όπως παρακάτω:

ABC=<A,B,C>

$|ABC| = \sqrt{(A * A) + (B * B) + (C * C)}$

Πλήθος κενών γραμμών:

Σε αυτή την μετρική περιέχεται το πλήθος των κενών γραμμών που υπάρχουν στον κώδικα.

Λογικές γραμμές κώδικα:

Σε αυτήν την μετρική περιέχεται το πλήθος των εκτελούμενων γραμμών κώδικα και των εντολών του προεπεξεργαστή (εντολών που ξεκινούν με τον ειδικό χαρακτήρα "#"), χωρίς να περιλαμβάνονται οι γραμμές σχολίων και οι κενές γραμμές

Γενικές παρατηρήσεις:

- Σε κάθε μετρική αποκλείονται οι δεσμευμένες λέξεις της γλώσσας C, οι οποίες διαγράφονται προτού επεξεργαστεί οποιαδήποτε γραμμή.
- Για τον υπολογισμό πολλών μετρικών καλούνται πρώτα συναρτήσεις για την διαγραφή σχολίων, δεσμευμένων λέξεων και μια συνάρτηση που ελέγχει αν η γραμμή είναι κενή (οι κενές γραμμές δεν χρειάζονται σάρωση ή επεξεργασία).
- Το εργαλείο χρειάζεται για τη σωστή εκτέλεσή του ένα αρχείο (με όνομα "sources_list.txt") το οποίο θα περιέχει τα ονόματα των αρχείων που θα σαρωθούν. Υπάρχει το αρχείο "filelist.bat" (για τα windows) το οποίο γεμίζει το αρχείο αυτόματα με τα ονόματα των αρχείων ".c" και ".h" που βρίσκονται στον ίδιο φάκελο ή σε υποφακέλους του.
- Το εργαλείο χρειάζεται δικαιώματα για δημιουργία, γράψιμο και διάβασμα αρχείων για τη σωστή λειτουργία και επιτυχή σάρωση των αρχείων εισόδου.
- Το εργαλείο δεν κάνει έλεγχο στο συντακτικό ή στη γραμματική της γλώσσας C, μόνο ψάχνει για τις συγκεκριμένες δομές, μορφές και στοιχεία του κώδικα που χρειάζονται για τον υπολογισμό της κάθε μετρικής.
- Παράγονται δύο αρχεία καταγραφής (log files) τα οποία περιέχουν πληροφορίες σχετικά με τη ροή εκτέλεσης και την πορεία των ελέγχων κατά τη διάρκεια της σάρωσης.
- Το εργαλείο δεν απαιτεί συγκεκριμένη εγκατάσταση ή ρυθμίσεις, μόνο την ύπαρξη του εκτελέσιμου αρχείου και του αρχείο sources_list.txt στον ίδιο φάκελο με τα αρχεία εισόδου (τα αρχεία εισόδου μπορούν να βρίσκονται στον ίδιο φάκελο ή σε υποφάκελο, αρκεί η διαδρομή προς τα αρχεία που θα σαρωθούν να περιέχεται στο αρχείο sources_list.txt).
- Στο λειτουργικό σύστημα linux, το script που υπάρχει και γεμίζει το αρχείο «sources_list.txt" (filelist.sh)δουλεύει σωστά μόνο όταν τόσο τα αρχεία εισόδου όσο

και το πρόγραμμα βρίσκονται στον ίδιο φάκελο (και το αρχείο sources_list.txt) και δεν εντοπίζει αρχεία εισόδου σε υποφακέλους.

- Τα script για τα windows και τα linux ΔΕΝ είναι απαραίτητα για το γέμισμα του αρχείου “sources_list.txt” και τη σωστή λειτουργία του εργαλείου, αλλά υπάρχουν για τη διευκόλυνση της διαδικασίας αυτής.

Κεφάλαιο 6) Αποτελέσματα συγκρίσεων με τα εργαλεία

Για τη διενέργεια δοκιμαστικών σαρώσεων και συγκρίσεων με τα διάφορα εργαλεία χρησιμοποιήθηκε ο πηγαίος κώδικας των παρακάτω προγραμμάτων:

- **0AD (C++) :** Ανοιχτού κώδικα τρισδιάστατο παιχνίδι γραμμένο σε γλώσσα C++ το οποίο δημιουργήθηκε και εξελίσσεται από τη δική του κοινότητα.
- **Appache standard cpp libraries (C++)** : Βιβλιοθήκες λογισμικού για τη γλώσσα C++ από την Apache.
- **Autobahn (JAVA) :** Εργαλείο σάρωσης κώδικα γραμμένο σε JAVA.
- **Vassal game engine (JAVA)** : Δωρεάν μηχανή δημιουργίας παιχνιδιών γραμμένη σε γλώσσα JAVA.
- **Akelpad (C++)** : Δωρεάν, ανοιχτού κώδικα και μικρού μεγέθους πρόγραμμα (αριθμομηχανή) γραμμένο σε γλώσσα C++.
- **Chromium (C++)** : Μεγάλου μεγέθους και ανοιχτού κώδικα πρόγραμμα πίσω από τον περιηγητή Chrome γραμμένο σε γλώσσα C++.
- **Blender (C++, Python) :** Δωρεάν και ανοιχτού κώδικα πρόγραμμα για τη δημιουργία τρισδιάστατων μοντέλων γραμμένο στις γλώσσες C++ και Python.
- **Linux kernel (C) :** Ο πυρήνας του λειτουργικού συστήματος Linux γραμμένος σε γλώσσα C.
- **Openoffice (C++)** : Δωρεάν, ανοιχτού κώδικα και μεγάλου μεγέθους εμπορικό πρόγραμμα (πακέτο προγραμμάτων openoffice) γραμμένο σε γλώσσα C++.

Τα εργαλεία αυτά εκπροσωπούν διάφορες κατηγορίες (εμπορικά προγράμματα, προγράμματα ανοιχτού κώδικα, δωρεάν προγράμματα) και γλώσσες προγραμματισμού (C, C++, JAVA, Python) και επιλέχθηκαν ως δείγματα από αυτές τις κατηγορίες και για τις συγκεκριμένες γλώσσες στις οποίες είναι γραμμένα.

6.1) Sonar

	0AD	Akelpad	Autobahn	Blender (C++)	Blender (Python)
Γραμμές κώδικα	129,407	47,064	153,561	707,743	215,112
Συνολικές γραμμές	271,052	82,904	274,047	1,310,974	293,066
Δηλώσεις	80,140	33,589	62,211	456,135	134,796
Αρχεία	2,026	27	1,777	5,176	909
Πλήθος κλάσεων	1,544	310	2,487	4,766	2,247
Πλήθος μεθόδων	40,436	807	25,324	34,454	8,972
Γραμμές σχολίων	51,278	12,323	34,300	194,509	31,461
Κενές γραμμές	15,184	6,116	-	54,682	-
Ποσοστό σχολίων	28.4%	20.8%	18.3%	21.6%	12.8%
Γραμμές διπλού κώδικα	65,430	18,249	36,932	87,117	28,552
Ποσοστό διπλού κώδικα	24.1%	22.0%	13.5%	6.4%	9.7%
Περιπλοκότητα (Συνολική)	27.182	11.733	33.487	129.158	45.828
Περιπλοκότητα (ανά μέθοδο)	2.6	14.5	1.3	3.7	4.4
Περιπλοκότητα (ανά κλάση)	17.1	37.8	13.5	26.3	17.1
Περιπλοκότητα (ανά αρχείο)	13.4	434.6	18.8	25.0	43.8
Ζητήματα (Σύνολο)	1,350	24	12,833	2,772	45,828
Απαγορευτικά	0	0	0	0	0
Κρίσιμα	0	0	27	0	0
Σημαντικά	1,350	24	6,296	2,772	2,264
Μικρής σημασίας	0	0	4,915	0	43,561
Πληροφορίες	0	0	1,595	0	0
Συμμόρφωση με τους κανόνες	96.9%	99.8%	84.4%	98.8%	76.6%

Πίνακας 1: Αποτελέσματα Sonar μέρος 1

	Apache cpp libraries	Chromium part 1	Chromium part 2	Openoffice part 1	Openoffice part 2
Γραμμές κώδικα	45,335	1,049,479	1,891,258	890,131	1,862,773
Γραμμές	110,266	1,912,897	4,138,825	4,495,333	2,945,166
Δηλώσεις	28,451	563,313	1,314,524	445,169	1,029,727
Αρχεία	980	13,201	12,823	8,258	11,622
Πλήθος κλάσεων	434	7,023	13,419	6,145	9,770
Πλήθος μεθόδων	2,091	89,690	111,081	51,324	97,415
Γραμμές σχολίων	21,301	293,477	738,830	210,544	370,816
Κενές γραμμές	8,236	19,376	173,928	56,651	140,933
Ποσοστό σχολίων	32.0%	21.9%	28.1%	19.1%	16.6%
Γραμμές διπλού κώδικα	18,466	112,961	850,519	113,618	159,571
Ποσοστό διπλού κώδικα	16.7%	5.9%	20.5%	7.6%	5.4%
Περιπλοκότητα (Συνολική)	8.085	215.285	412.062	161.446	374.005
Περιπλοκότητα (ανά μέθοδο)	3.9	2.4	3.7	3.1	3.8
Περιπλοκότητα (ανά κλάση)	18.0	29.5	30.1	24.3	36.1
Περιπλοκότητα (ανά αρχείο)	8.3	16.3	22.6	19.6	32.2
Ζητήματα (Σύνολο)	541	9,640	9,340	6,084	8,734
Απαγορευτικά	0	0	0	0	0
Κρίσιμα	0	0	0	0	0
Σημαντικά	541	9,640	9,340	6,084	8,734
Μικρής σημασίας	0	0	0	0	0
Πληροφορίες	0	0	0	0	0
Συμμόρφωση με τους κανόνες	96.4%	97.2%	98.5%	97.9%	98.6%

Πίνακας 2: : Αποτελέσματα Sonar μέρος 2

6.2) CppCheck

	Σφάλματα	Προειδοποιήσεις	Προειδοποιήσεις για το στυλ	Προειδοποιήσεις μεταφερσιμότητας	Ζητήματα απόδοσης	Πληροφορίες	Σύνολο
0AD	23	260	743	2	55	149	962
Akelpad	21	13	514	3	3	0	554
Blender	111	930	5,381	26	789	147	7,384
Chromium	243	1,662	8,456	30	612	1,326	12,329
Linux kernel	450	1,217	27,724	48	1,569	1,351	32,359
Openoffice	104	854	5,363	13	1,966	401	8,701
Apache Cpp libraries	32	210	296	0	77	190	805

Πίνακας 3: Αποτελέσματα CppCheck

6.3) YASCA

	Κρίσιμα	Υψηλής σοβαρότητας	Μεσσαίας σοβαρότητας	Χαμηλής σοβαρότητας	Πληροφορίες	Σύνολο
0AD	1	3	6	0	6	16
Akelpad	0	0	0	0	0	2
Blender	8	3	1	12	13	27
Chromium	267	76	55	0	32	430
Linux kernel	4	51	23	0	12	90
Openoffice	51	8	612	80	4,501	5,252
Apache Cpp libraries	0	1	0	0	2	3
Autobahn	35	3	95	20	263	416
Vassal engine	16	6	129	18	154	323

Πίνακας 4: Αποτελέσματα YASCA

6.4) PMD

	Autobahn	Vassal engine
Απαραίτητη διόρθωση	51	176
Σοβαρά συνιστώμενη διόρθωση	49	355
Συνιστώμενη διόρθωση	7,023	6,262
Προαιρετική διόρθωση	160	420
Απόλυτα προαιρετική διόρθωση	4,780	2,928
Σύνολο ζητημάτων	12,063	10,141

Πίνακας 5: Αποτελέσματα PMD για το ruleset group 1

	Autobahn	Vassal engine
Απαραίτητη διόρθωση	1,103	228
Σοβαρά συνιστώμενη διόρθωση	1	2
Συνιστώμενη διόρθωση	10,316	8,035
Προαιρετική διόρθωση	125	85
Απόλυτα προαιρετική διόρθωση	0	0
Σύνολο ζητημάτων	11,545	8,350

Πίνακας 6: Αποτελέσματα PMD για το ruleset group 2

	Autobahn	Vassal engine
Απαραίτητη διόρθωση	0	0
Σοβαρά συνιστώμενη διόρθωση	0	0
Συνιστώμενη διόρθωση	13,822	12,728
Προαιρετική διόρθωση	120	257
Απόλυτα προαιρετική διόρθωση	0	0
Σύνολο ζητημάτων	13,942	13,035

Πίνακας 7: Αποτελέσματα PMD για το ruleset group 3

Σχόλια σχετικά με τα πειράματα και τα αποτελέσματα:

- Το ruleset group 1 περιλαμβάνει τους ελέγχους unused code, basic, design και controversial.
- Το ruleset group 2 περιλαμβάνει τους ελέγχους code size, empty code και naming.
- Το ruleset group 3 περιλαμβάνει τους ελέγχους optimization, security code guidelines και unneccessary.

6.5) Checkstyle

	Autobahn	Vassal engine	Openoffice
Σφάλματα	217,484	61,950	854,756
Προειδοποιήσεις	0	0	0
Σύνολο ζητημάτων	217,484	61,950	854,756

Πίνακας 8: : Αποτελέσματα Checkstyle για το αρχείο sun_checks.xml

	Autobahn	Vassal engine	Openoffice
Σφάλματα	240,129	169,866	908,220
Προειδοποιήσεις	13	526	21
Σύνολο ζητημάτων	240,139	170,392	908,241

Πίνακας 9: Αποτελέσματα Checkstyle για το αρχείο Checkstyle_checks.xml

6.6) Static Analysis Tool

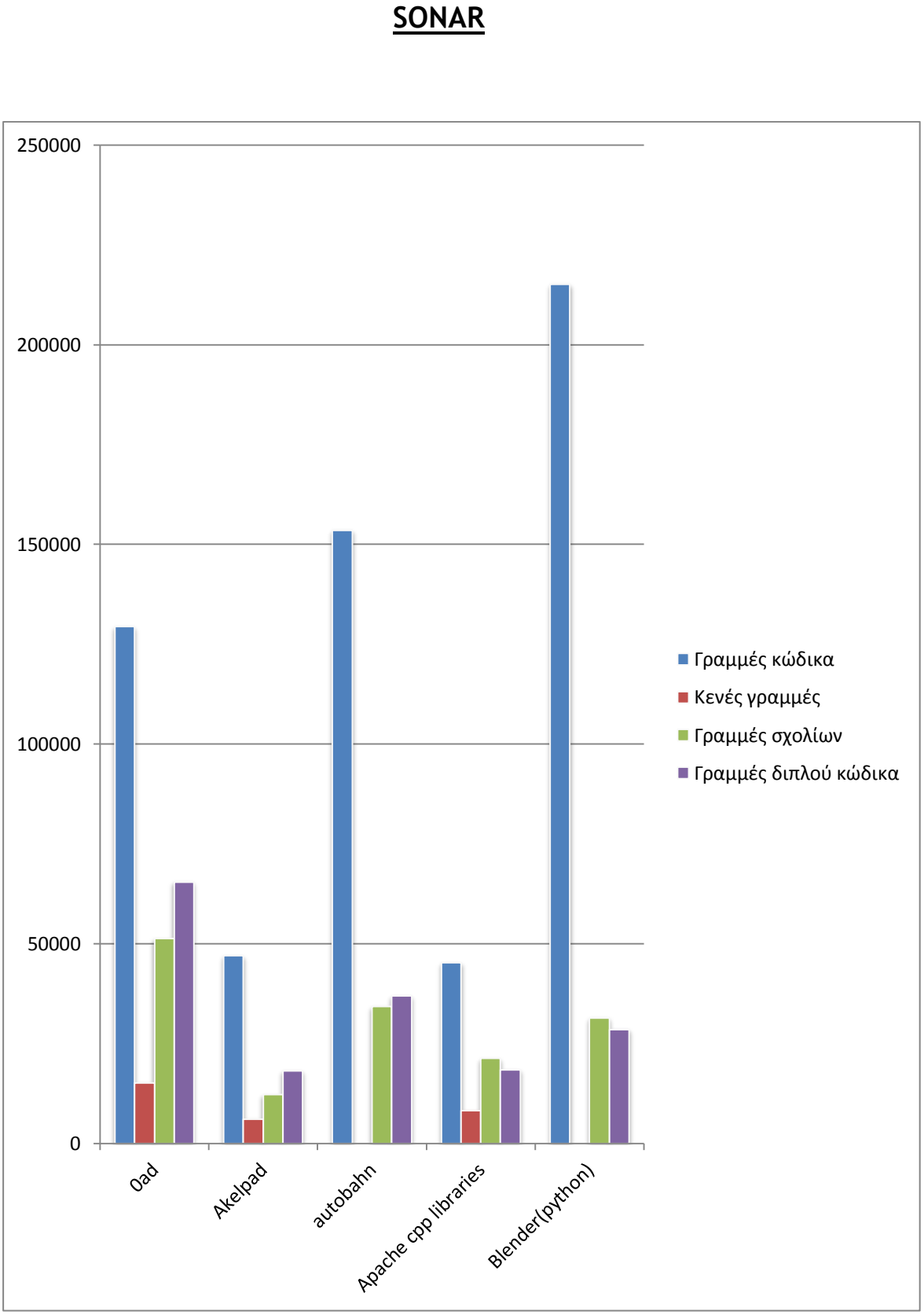
	Linux kernel	Akelpad	0AD	Apache Cpp libraries	Static Analysis Tool
Γραμμές κώδικα	21,123,252	95,405	31,932	39,357	3,048
Πηγαία αρχεία	33,004	28	1,204	220	5
Γραμμές σχολίων	16,704,816	61,708	28,734	35,869	1,184
Κενές γραμμές	2,963,199	13,816	-	-	288
Ποσοστό σχολίων	79.08%	64.68%	64.66%	91.14%	38.85%
Λογικές γραμμές κώδικα	8,090,169	35,264	-	-	1,317

Πίνακας 10: Αποτελέσματα Static Analysis Tool μέρος 1

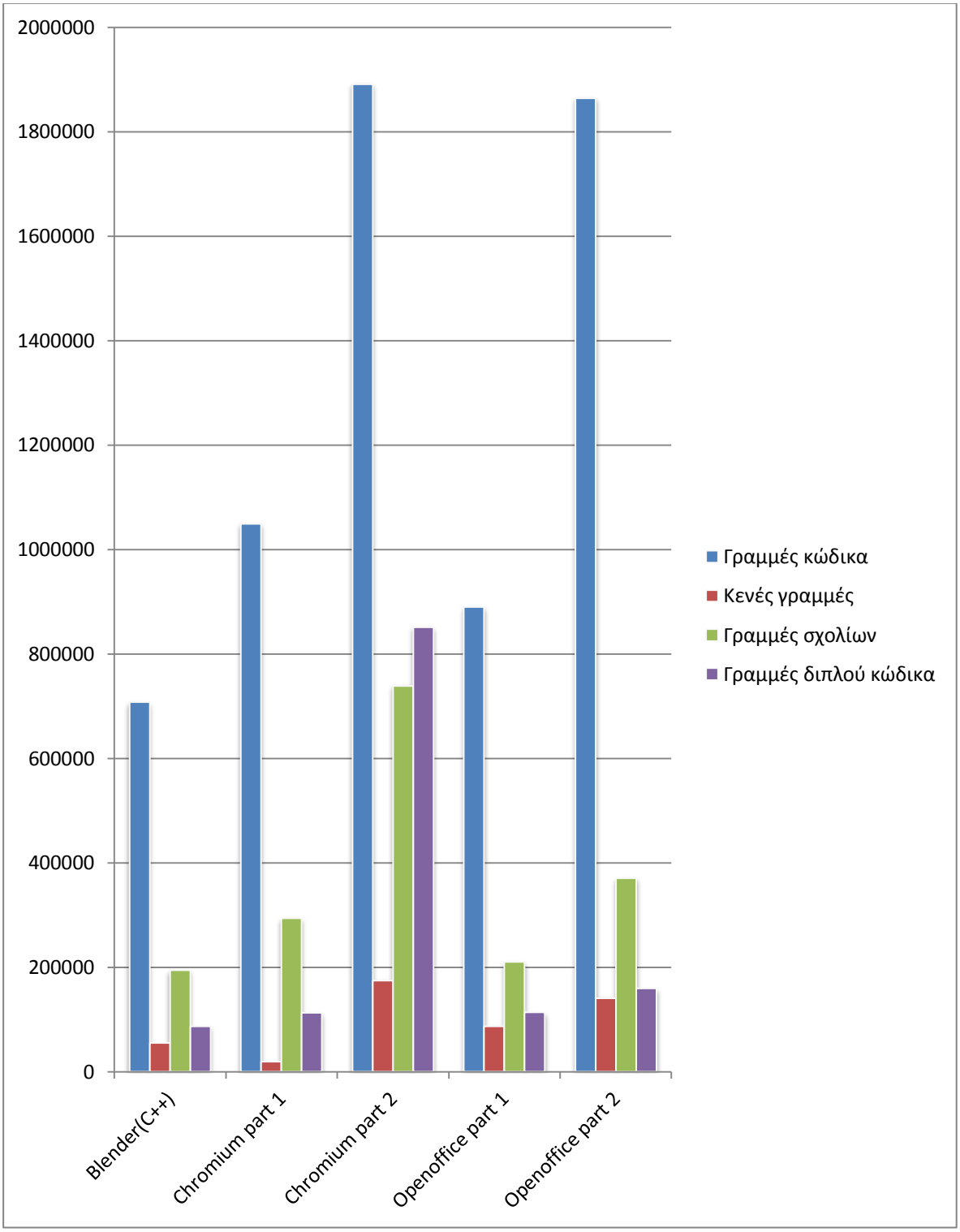
	Linux kernel	Akelpad	Static Analysis Tool
Λεξιλόγιο	14,344,176	11,796	406
Μήκος προγράμματος	97,711,518	377,043	16,305
Υπολ. Μήκος προγράμματος	341,017,004.08	159,158.86	3,349.54
Όγκος	2,322,989,892.37	5,099,887.43	141,288.30
Δυσκολία	70.068	304.31	252.44
Επίπεδο προγράμματος	0.014	0.0032	0.0034
Προσπάθεια	162,766,780,316.59	1,551,957,351.05	35,666,940.53
Χρόνος που χρειάστηκε (seconds)	9,042,598,906.48	86,219,852.84	1,981,496.70
Σφάλματα που παραδόθηκαν	744,329.96	169,996.25	47,061
Έξυπνο περιεχόμενο	33,153,460.61	16,758.74	559.69
Διανοητική προσπάθεια	10,197,991,203.43	3,465,750.71	55,753.57
ABC	<4.855.776,4.758.807,4.585.302>	<22.768,13.624,19.791>	<1.115,562,151>
|ABC|	88,267.0	33,101.0	1,257.0

Πίνακας 11: : Αποτελέσματα Static Analysis Tool μέρος 2

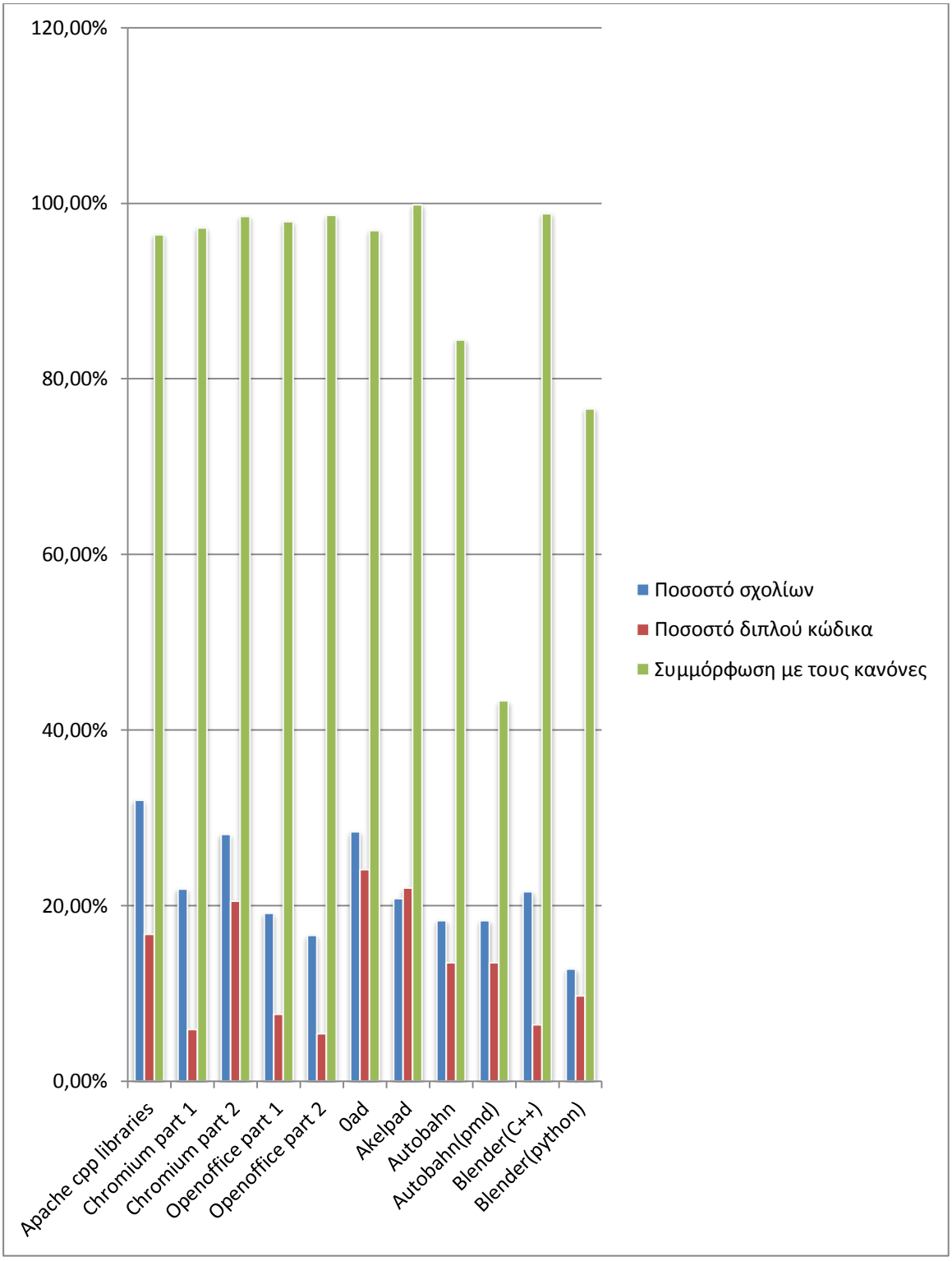
6.7) Γραφήματα



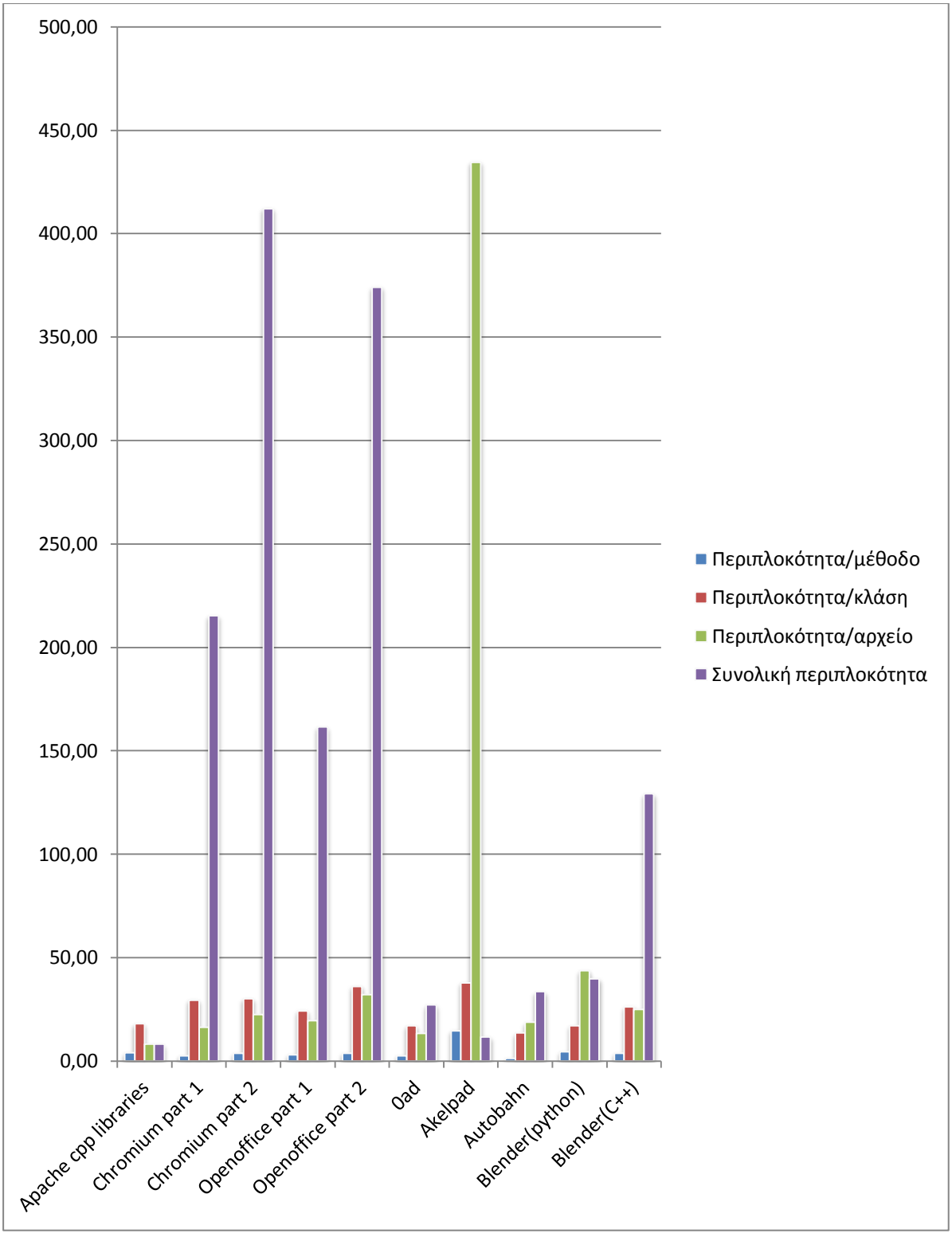
Γράφημα 1: Αποτελέσματα Sonar μέρος 1



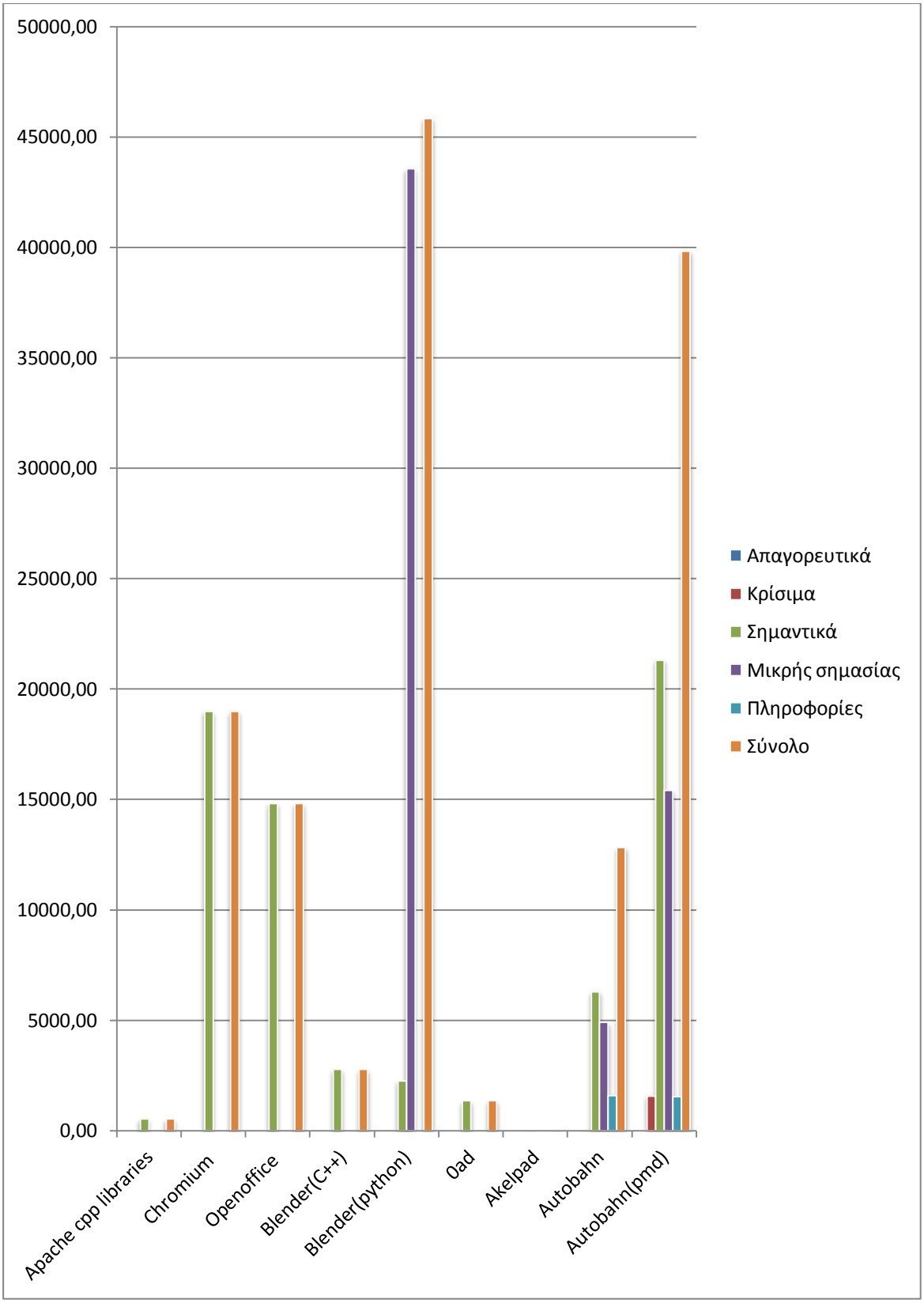
Γράφημα 2: Αποτελέσματα Sonar μέρος 2



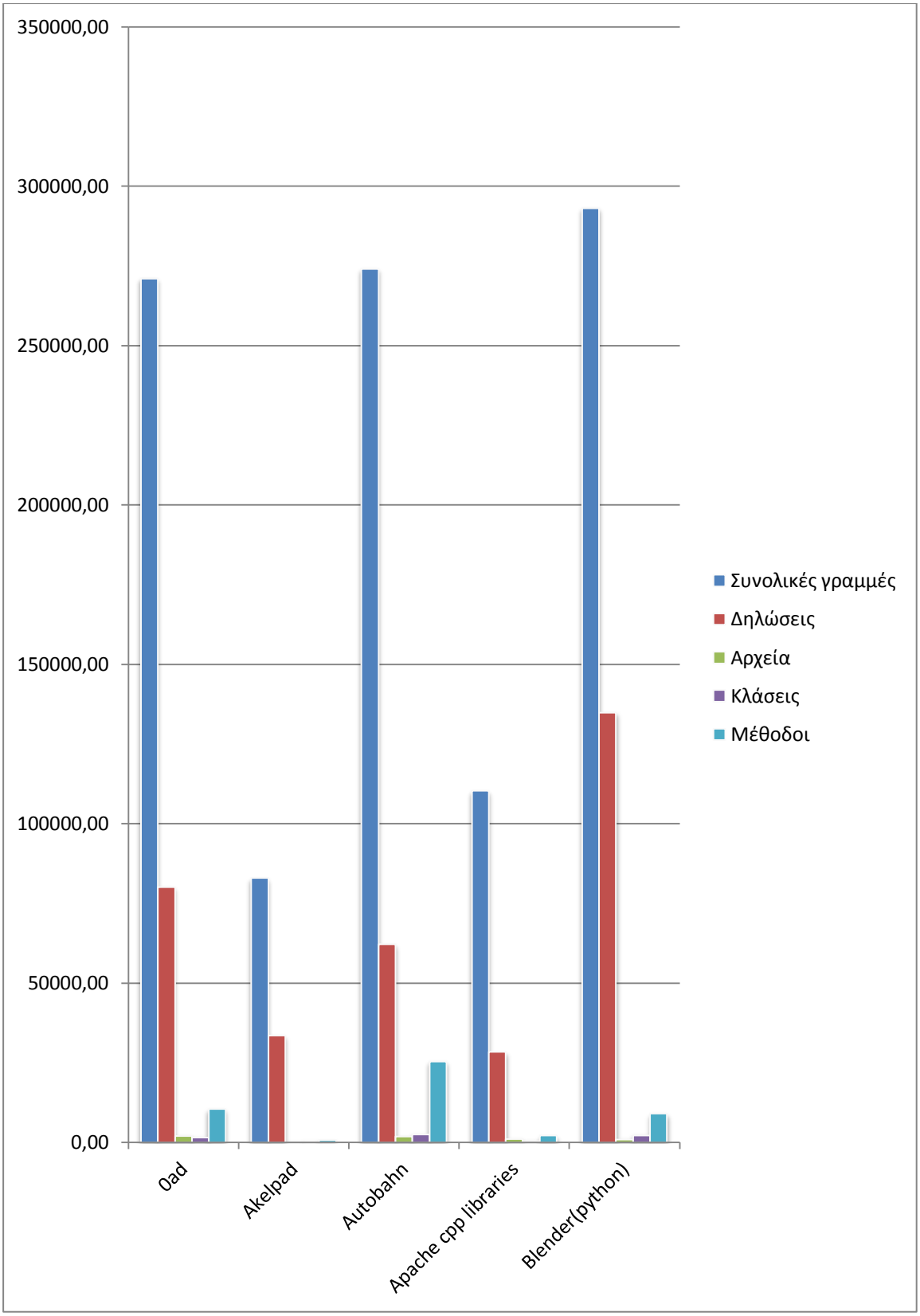
Γράφημα 3: Αποτελέσματα Sonar μέρος 3



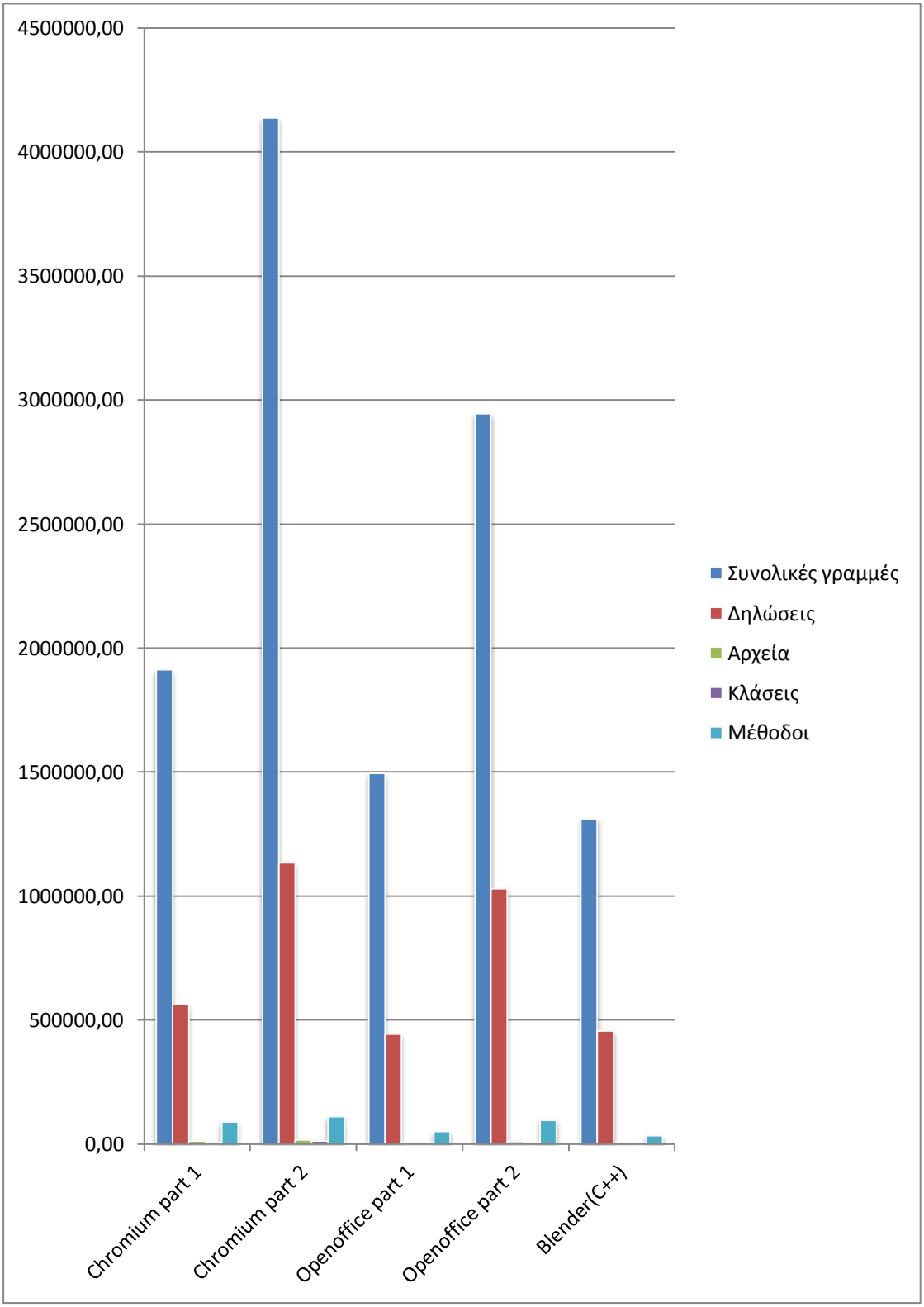
Γράφημα 4: Αποτελέσματα Sonar μέρος 4



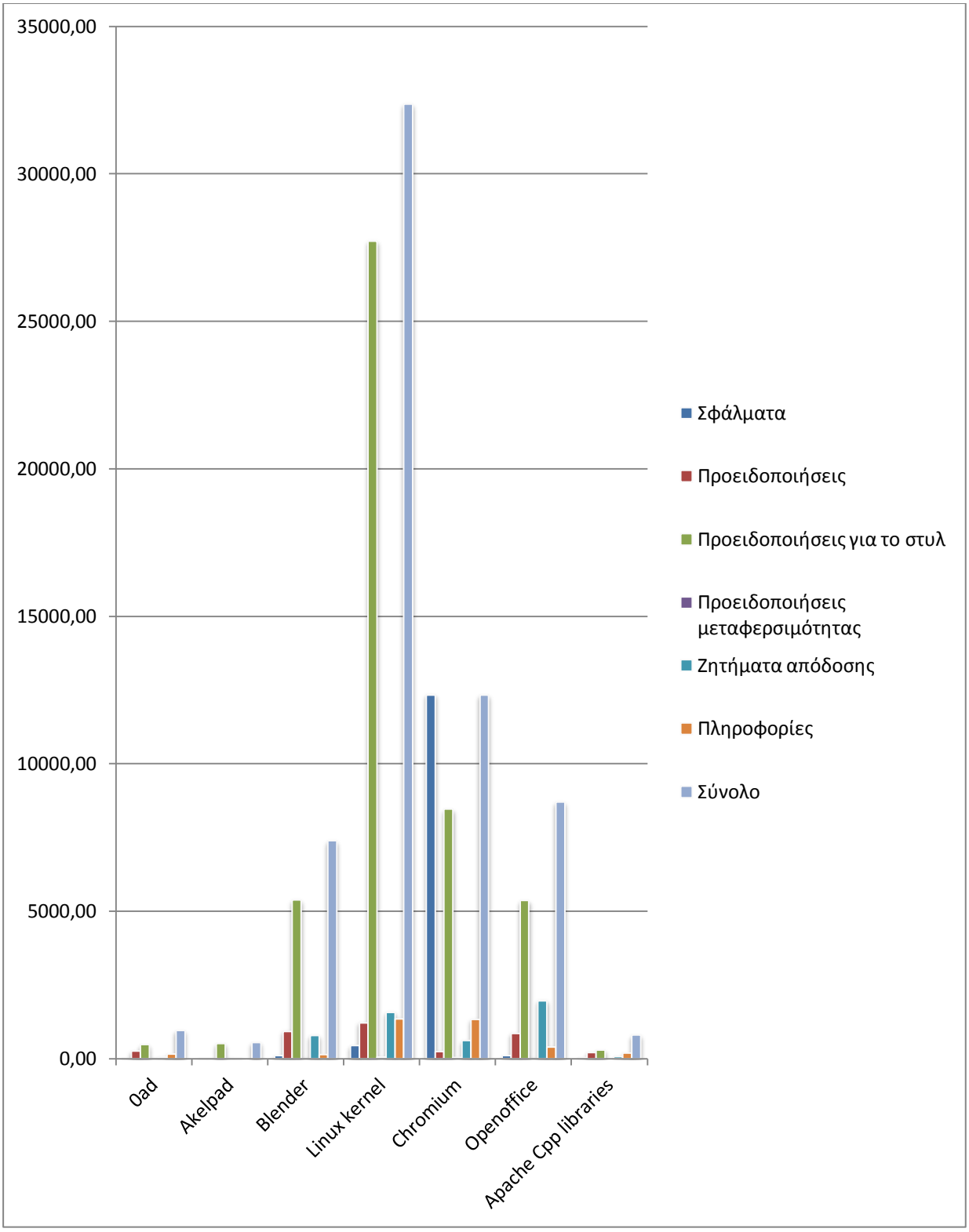
Γράφημα 5: Αποτελέσματα Sonar μέρος 5



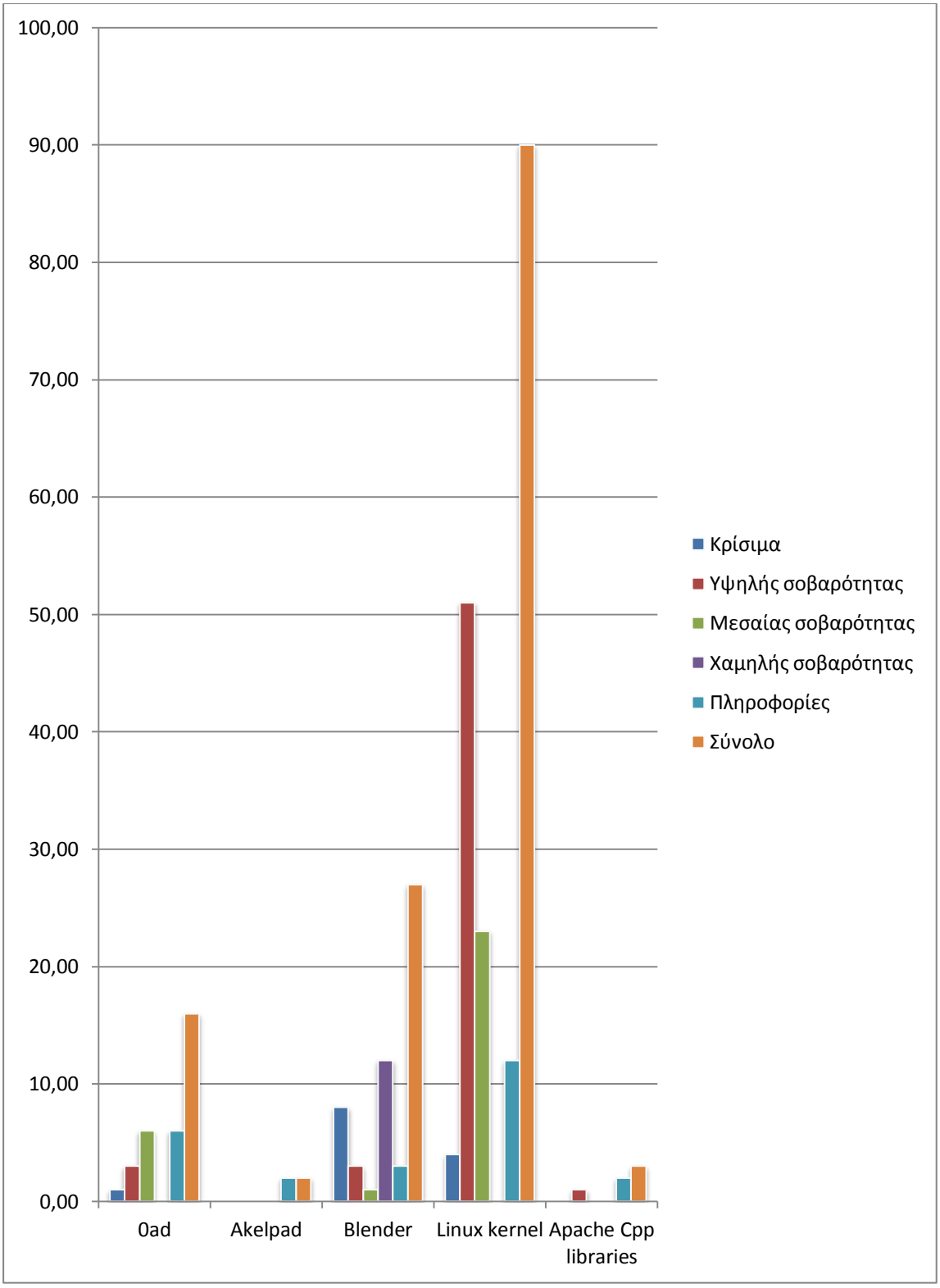
Γράφημα 6: Αποτελέσματα Sonar μέρος 6



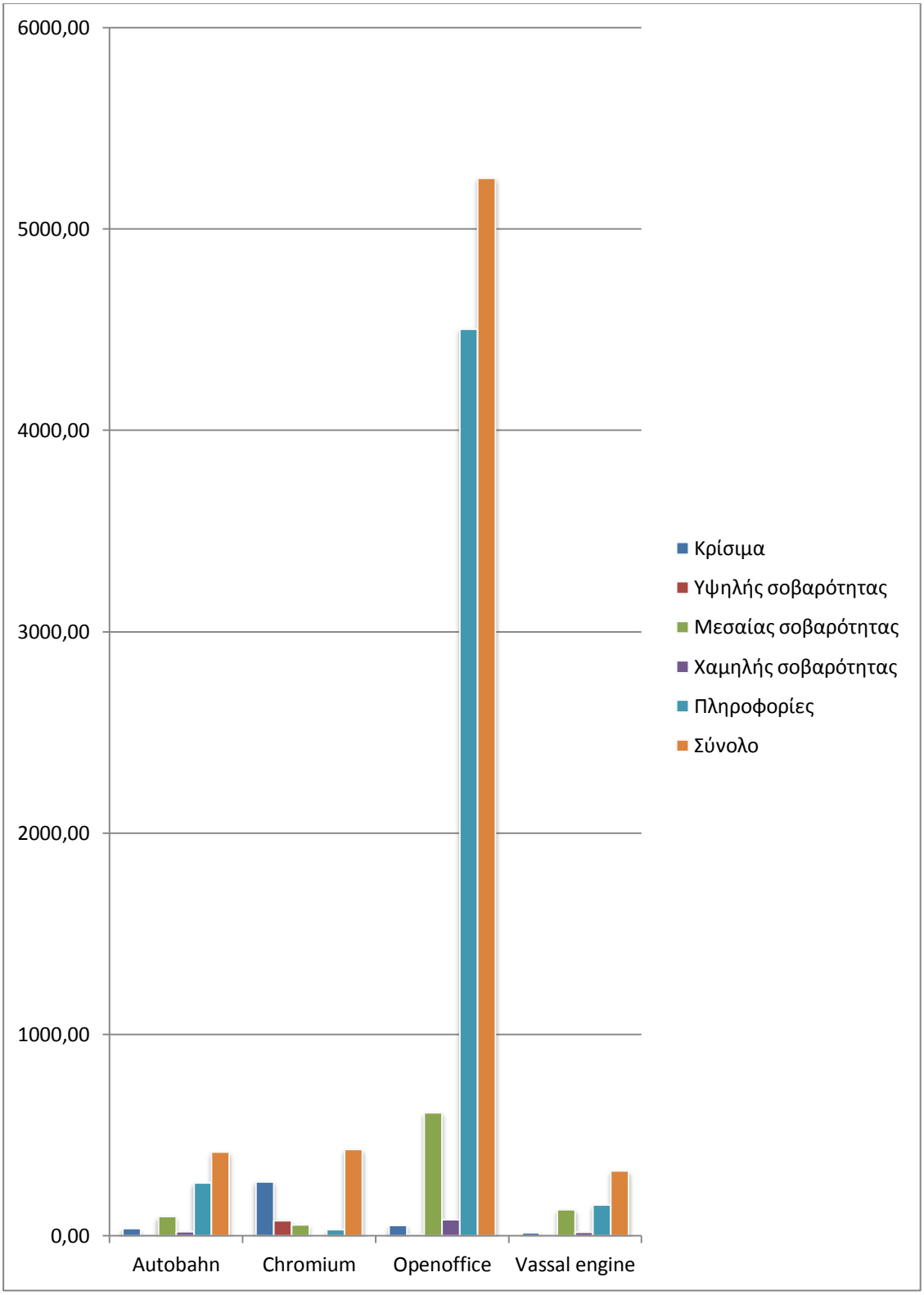
Γράφημα 7: Αποτελέσματα Sonar μέρος 7



Γράφημα 8: Αποτελέσματα CppCheck

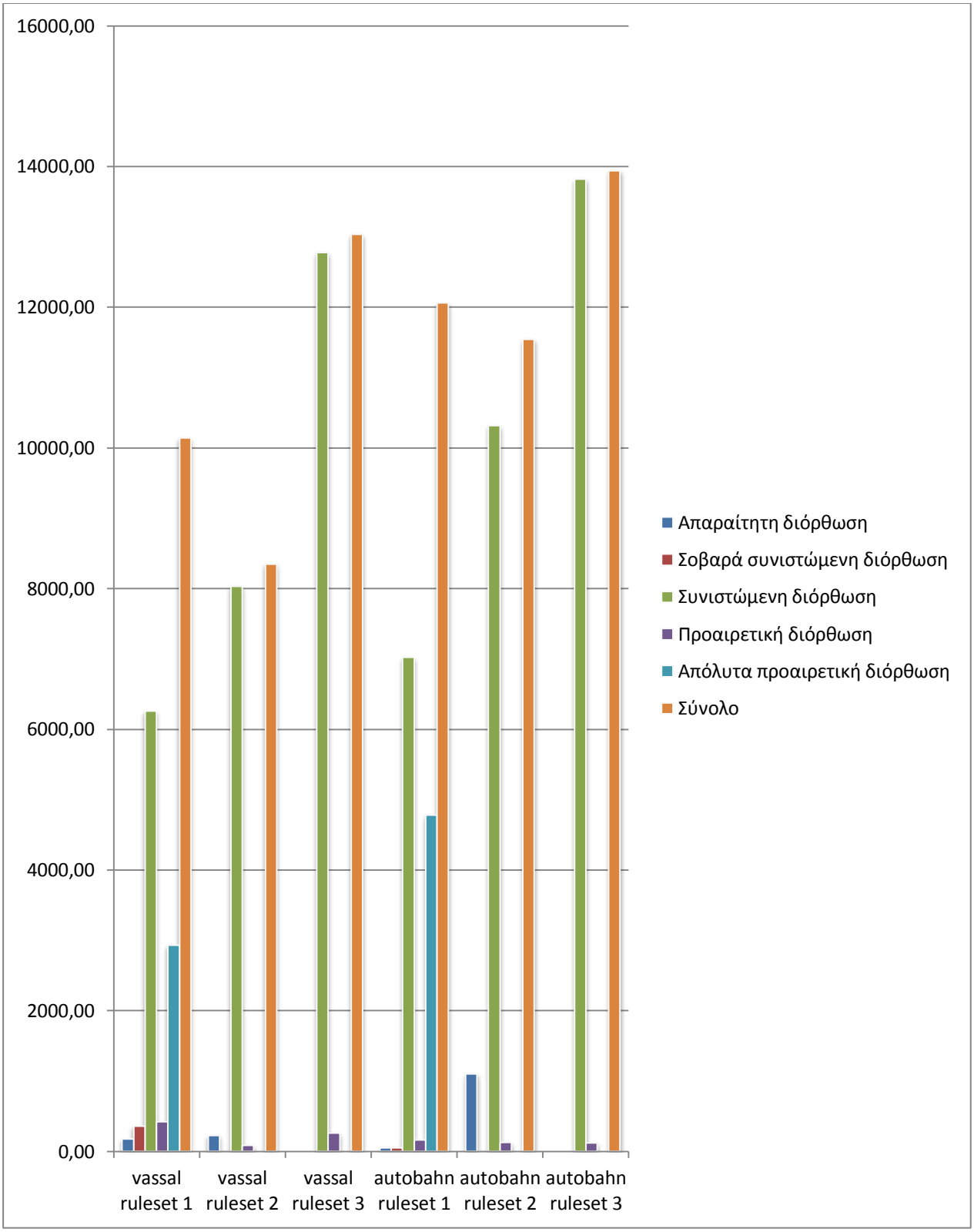


Γράφημα 9: Αποτελέσματα YASCA μέρος 1



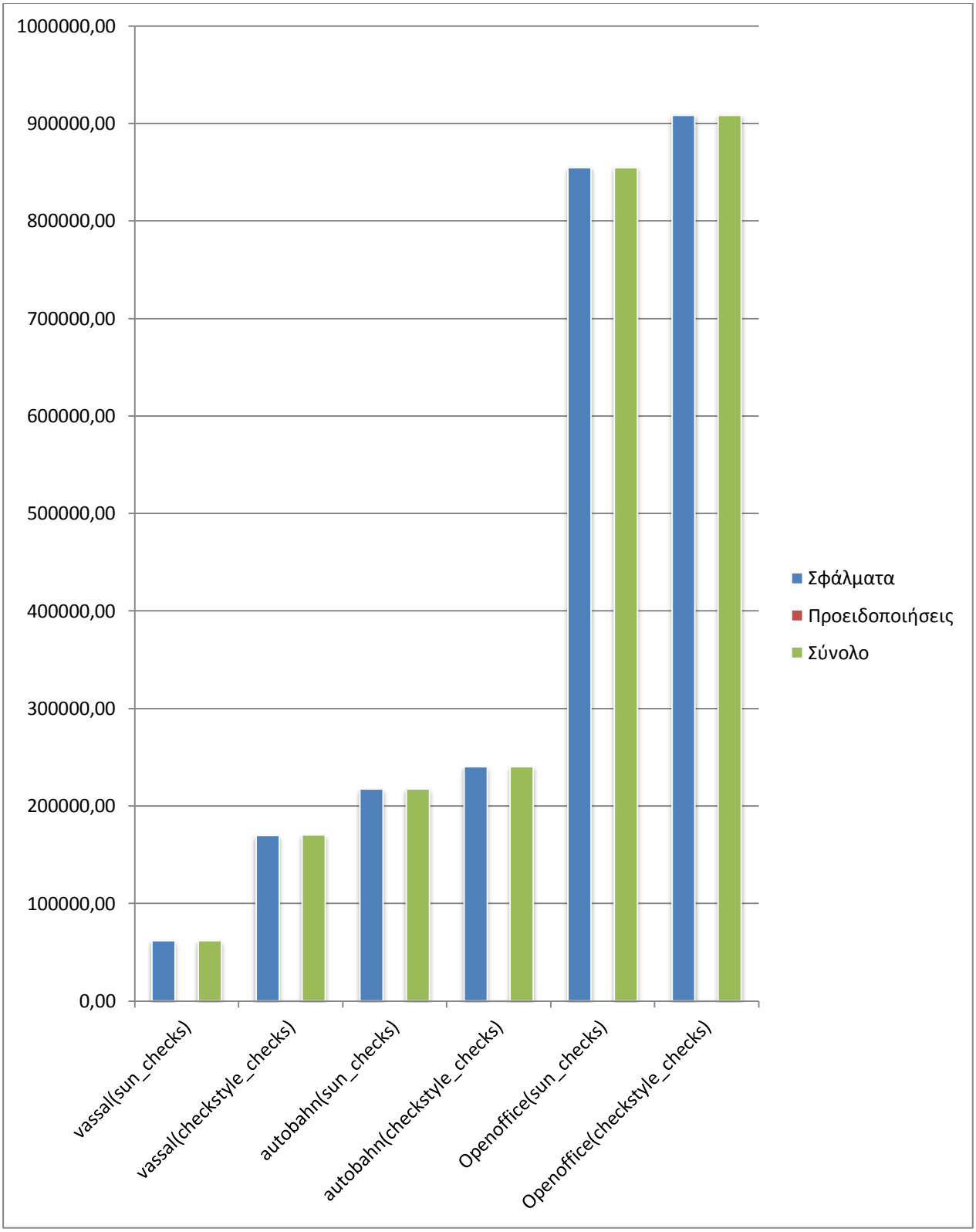
Γράφημα 10: Αποτελέσματα YASCA μέρος 2

PMD

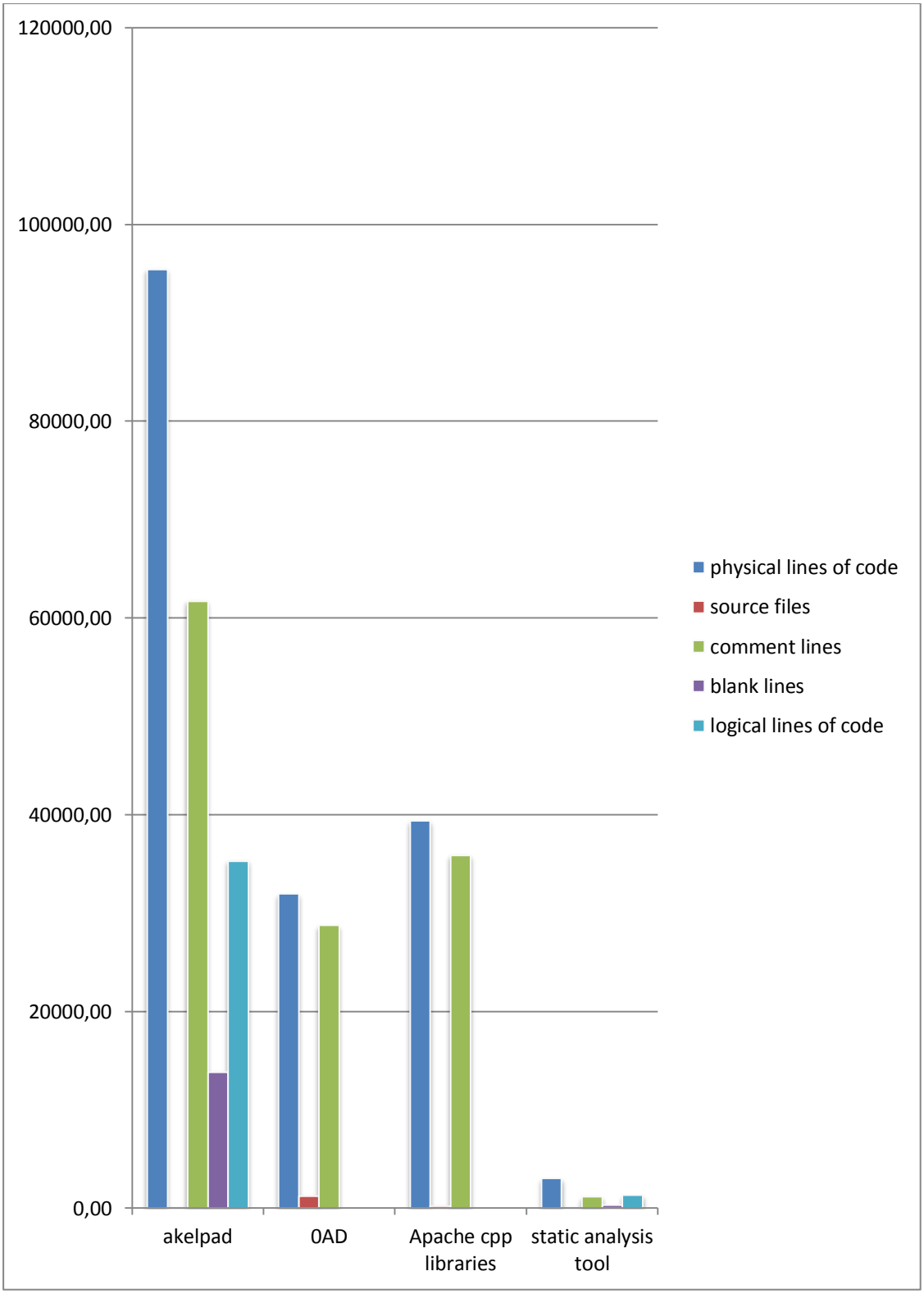


Γράφημα 11: Αποτελέσματα PMD

CHECKSTYLE



Γράφημα 12: Αποτελέσματα Checkstyle



Γράφημα 13: Αποτελέσματα Static Analysis Tool

Βιβλιογραφία

[1]. ‘Software Testing and Evaluation’
DeMillo-McCracken-Martin-Passafiume
Εκδότης: Addison-Wesley (January 1987)

[2]. ‘Software Testing Techniques’
Boris Beizer-Second Edition
Εκδότης: Intl Thomson Computer Pr (T) (June 1990)

[3]. ‘Software Metrics: A rigorous and Practical Aprroach’
Norman E.Fenton-Shari Lawrence Pfleeger Second Edition
Εκδότης: Course Technology(February 24, 1998)

[4]. ‘Software Metrics for Product Assessment’
Richard Bache-Gualtiero Bazzana
Εκδότης: Mcgraw Hill Book Co Ltd (December 1993)

[5]. ‘Managing Software Quality’
Brian Hambling
Εκδότης: Mcgraw Hill Book Co Ltd (June 1996)

[6]. Source Code Analysis Tools
https://www.owasp.org/index.php/Source_Code_Analysis_Tools

[7]. Cobertura
http://cobertura.github.io/cobertura/

[8]. PMD
http://pmd.sourceforge.net/

[9]. Apache Ant
http://ant.apache.org/

[10]. Software QA and Testing Resource Center - Table of Contents
http://www.softwareqatest.com/

[11]. Padre
http://padre.perlide.org

[12]. Pylint
https://pypi.python.org/pypi/pylint

[13]. FindBugs
http://findbugs.sourceforge.net/

[14]. Checkstyle
http://checkstyle.sourceforge.net/

[15]. Gimpel Software
http://www.gimpel.com/html/index.htm

[16]. BLAST
http://mtc.epfl.ch/software-tools/blast/index-epfl.php

[17]. CppCheck
http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main_Page

[18]. Parasoft
http://www.parasoft.com/

[19]. YASCA
http://www.scovetta.com/yasca.html

[20]. Modified BSD Lisence
http://oss-watch.ac.uk/resources/modbsd

[21]. Apache Software Foundation
http://www.apache.org/licenses/LICENSE-2.0

[22]. The Chromium Projects

http://www.chromium.org/Home
[23]. Linux Kernel Archives
https://www.kernel.org/
[24]. Blender buildbot
http://builder.blender.org/
[25]. Apache C++ Standard Library (STDCXX)
http://stdcxx.apache.org/
[26]. 0AD
http://play0ad.com/
[27]. SonarQube
http://www.sonarqube.org/
[28]. Wikipedia
http://www.wikipedia.org/
[29]. Google
http://www.google.com