Ατομική Διπλωματική Εργασία

ΣΥΓΚΡΙΣΗ ΕΡΓΑΛΕΙΩΝ ΤΗΣ ΒΙΟΜΗΧΑΝΙΑΣ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ ΚΩΔΙΚΑ

Νικολέττα Πέτρου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2014

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σύγκριση εργαλείων της βιομηχανίας λογισμικού για στατική ανάλυση κώδικα

Νικολέττα Πέτρου

Επιβλέπων Καθηγήτρια

Γεωργία Καπιτσάκη

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ιούνιος 2014

Ευχαριστίες

Θα ήθελα να πω ένα μεγάλο ευχαριστώ στην επιβλέποντα καθηγήτρια μου κυρία Γεωργία Καπιτσάκη για την υποστήριξη και καθοδήγηση που μου πρόσφερε κατά τη διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας.

Στη συνέχεια θα ήθελα να ευχαριστήσω τους γονείς μου, που πάντοτε με στήριζαν με το δικό τους τρόπο και μου συμπαραστάθηκαν όχι μόνο για τη διπλωματικής μου εργασία, αλλά καθ' όλη τη πορεία μου στο Πανεπιστήμιο Κύπρου.

Περίληψη

Ο στόχος της παρούσας διπλωματικής εργασίας είναι η ποιοτική και ποσοτική σύγκριση διαφόρων εργαλείων στατικής ανάλυσης κώδικα που χρησιμοποιούνται στη βιομηχανία λογισμικού με σκοπό την εξασφάλιση ποιοτικού κώδικα. Στατική ανάλυση πραγματοποιείται σε πρόγραμμα το οποίο δε βρίσκεται υπό εκτέλεση και στις περισσότερες περιπτώσεις γίνεται σε επίπεδο πηγαίου κώδικα.

Σε γενικές γραμμές, θα αναφερθούμε στην ποιότητα και στον έλεγχο λογισμικού, στην στατική και στην δυναμική ανάλυση κώδικα και στην συνέχεια θα γίνει παρουσίαση της σύγκρισης των εργαλείων που επιλέχθηκαν. Η πραγματοποίηση της σύγκρισης έγινε χρησιμοποιώντας ένα σύνολο έργων λογισμικού με διαφορετικά χαρακτηριστικά. Στην συνέχεια, θα αναφερθούμε σε λιγότερη έκταση στην δυναμική ανάλυση και σε εργαλεία δυναμικής ανάλυσης και τελειώνοντας θα μιλήσουμε για τα συμπεράσματα μας, καθώς και για μελλοντική εργασία πάνω σε αυτού του είδους την μελέτη.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Με την εξέλιξη της τεχνολογίας, η ανάγκη για την εξασφάλιση της ποιότητας λογισμικού γίνεται όλο και μεγαλύτερη και αποτελεί βασικότατη επιδίωξη επιχειρήσεων, οργανισμών και προγραμματιστών. Η επίτευξη της διασφάλισης της ποιότητας του λογισμικού συντελεί στον περιορισμό του κόστου παραγωγής και ελέγχου του λογισμικού.

Οι εταιρίες λογισμικού συχνά χρησιμοποιούν – ή ακόμα και αναγκάζουν τους υπαλλήλους τους να χρησιμοποιούν - διάφορα εργαλεία στατικής ανάλυσης κώδικα με σκοπό την εξασφάλιση ποιοτικού κώδικα. Τα εργαλεία αυτά μπορούν να εντοπίσουν πιθανά λάθη του προγράμματος κατά τη φάση της συγγραφής του κώδικα. Εντοπίζοντας τα πιθανά λάθη, στην φάση της συγγραφής του κώδικα διευκολύνεται ο έλεγχος του λογισμικού που πραγματοποιείται σε μεταγενέστερο στάδιο καθώς αποφεύγονται πιθανά λάθη που θα κόστιζαν χρόνο για την διόρθωση τους.

1.2 Σκοπός

Στην παρούσα διπλωματική εργασία πραγματοποιείται μια μελέτη ενός αριθμού εργαλείων στατικής ανάλυσης κώδικα με σκοπό να δούμε εάν και τα διάφορα εργαλεία

στατικής ανάλυσης κώδικα βοηθούν στην εξασφάλιση της ποιότητας του κώδικα. Πιο συγκεκριμένα γίνεται μελέτη και σύγκριση των εργαλείων στατικής ανάλυσης με βάση ενός συνόλου έργων συστημάτων λογισμικού. Με την ολοκλήρωση της εργασίας αυτής θα είμαστε σε θέση να γνωρίζουμε τις δυνατότητες των εργαλείων στατικής ανάλυσης κώδικα και τα πλεονεκτήματα – μειονεκτήματα των εργαλείων αυτών.

1.3 Δομή

Στο κεφάλαιο 2 γίνεται μια αναφορά στην ποιότητα και στον έλεγχο λογισμικού, στην στατική και δυναμική ανάλυση κώδικα. Δίδεται ουσιαστικά ένα τεχνικό υπόβαθρο.

Στο κεφάλαιο 3 περιγράφεται η χρησιμότητα των εργαλείων στατικής ανάλυσης κώδικα και ο τρόπος επιλογής των εργαλείων για την πραγματοποίηση της σύγκρισης.

Στο κεφάλαιο 4 δίδεται το σχήμα της μεθοδολογίας που ακολουθήθηκε. Παρουσιάζονται τα κριτήρια που λήφθηκαν για την σύγκριση των εργαλείων και ακολούθως περιγράφεται η σύγκριση των εργαλείων με βάση τα κριτήρια αυτά.

Στο κεφάλαιο 5 γίνεται μια μικρή αναφορά στην δυναμική ανάλυση κώδικα και σε εργαλεία δυναμικής ανάλυσης κώδικα.

Στο κεφάλαιο 6 παρουσιάζονται τα συμπεράσματα από την όλη εργασία. Δίδονται, επίσης προτάσεις για μελλοντική εργασία.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Εισαγωγή

Για την σύγκριση των εργαλείων στατικής ανάλυσης κώδικας απαιτείται η κατανόηση των εννοιών ποιότητας λογισμικού, έλεγχος ποιότητας λογισμικού καθώς τα εργαλεία αυτά αποσκοπούν στην βελτίωση του πηγαίου κώδικα. Στο κεφάλαιο αυτό γίνεται αναφορά για τις έννοιες ποιότητα κώδικα λογισμικού και έλεγχος ποιότητας κώδικα λογισμικού. Περιγράφεται επίσης η στατική ανάλυση κώδικα και η δυναμική ανάλυση κώδικα.

2.2 Ποιότητα λογισμικού

Σύμφωνα με το πρότυπο ISO 8402 [17], '*ποιότητα είναι το σύνολο των χαρακτηριστικών μιας οντότητας που της αποδίδουν την ικανότητα να ικανοποιεί εκφρασμένες και συνεπαγόμενες ανάγκες*'. Πέρα από αυτό τον ορισμό, στη βιβλιογραφία υπάρχουν πάρα πολλοί ορισμοί. Ένας από τους πρώτους ορισμούς διατυπώθηκε από

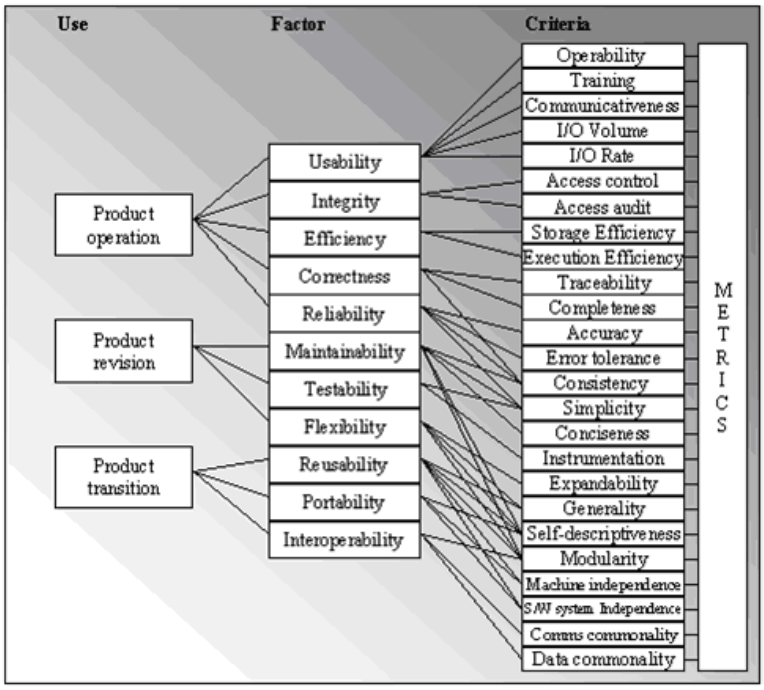
την Αμερικανική Κοινότητα Ποιοτικού Ελέγχου [2] το 1978 και λέει πως '*ποιότητα είναι η συλλογή των χαρακτηριστικών και ιδιοτήτων του προϊόντος που σχετίζονται με τη δυνατότητα του να εκπληρώνει τις ζητούμενες ανάγκες των πελατών*'. Ένας άλλος ορισμός δόθηκε από τον Philip Crosby το 1979 [8] και σύμφωνα με τον αυτόν '*η ποιότητα είναι η συμμόρφωση με τις απαιτήσεις των χρηστών'*. Αντίστοιχοι ορισμοί αναφέρουν πως '*η ποιότητα είναι η καταλληλότητα προς χρήση*' [19] ή '*ποιότητα είναι η συλλογή των χαρακτηριστικών σχεδιασμού, κατασκευής και συντήρησης, δια μέσου των οποίων το προϊόν κατά τη χρήση του θα εκπληρώσει τις προσδοκίες των πολιτών* ' [10]. Ο David Garvin το 1984 [12] ορίζει την ποιότητα ως μια πολύπλοκη έννοια που μπορεί να περιγραφεί με 5 διαφορετικές απόψεις:

1.Εμπειρική θεώρηση (transcendental view): η ποιότητα είναι κάτι που μπορεί να αναγνωριστεί εμπειρικά αλλά όχι να οριστεί ούτε να επιτευχθεί πλήρως.
2.Θεώρηση από την πλευρά του χρήστη (user view): η ποιότητα ορίζεται ως καταλληλότητα για χρήση.
3. Κατασκευαστική θεώρηση (manufacturing view) : η ποιότητα αντιμετωπίζεται ως η ικανοποίηση των απαιτήσεων και των ιδιαιτεροτήτων του χρήστη.
4. Θεώρηση από την πλευρά του προϊόντος (product view) : η ποιότητα ταυτίζεται με τα εσωτερικά χαρακτηριστικά του προϊόντος.
5. Θεώρηση βάσει της αξίας (value-based view) : η ποιότητα εξαρτάται από την αξία που κοστολογείται το προϊόν, δηλαδή από το ποσό που διατίθεται να πληρώσει ο χρήστης.

Οι διάφορες θεωρίες και ορισμοί γύρω από την ποιότητα του λογισμικού οδήγησαν στην δημιουργία των μοντέλων ποιότητας, στα οποία η ποιότητα διασπάται σε διάφορους παράγοντες που μπορούν να μετρηθούν με διάφορες μετρικές. Τα μοντέλα ποιότητας κατηγοριοποιούν την ποιότητα σε επιμέρους παράγοντες ως προς τον τελικό χρήστη, την ομάδα ανάπτυξης και την κοινότητα των χρηστών.

Το μοντέλο FCM του Jim McCall [31] περιλαμβάνει παράγοντες που σχετίζονται περισσότερο με την παραγωγή και το σχεδιασμό του λογισμικού, όπως η χρησιμότητα (usability), ακεραιότητα (integrity), αποδοτικότητα (efficiency), ορθότητα (correctness), αξιοπιστία (reliability), συντηρησιμότητα (maintainability), ελεγξιμότητα

(testability), ευελιξία (flexibility), επαναχρησιμοποιότητα (reusability), μεταφερσιμότητα (portability), διαλειτουργικότητα (interoperability). Οι παράγοντες αυτοί διασπώνται σε κριτήρια ποιότητας (quality criteria) τα οποία περιγράφουν εσωτερικά και εξωτερικά χαρακτηριστικά του προϊόντος και μετρώνται με διάφορες μετρικές. Τα εσωτερικά χαρακτηριστικά αφορούν μετρικές όπως οι γραμμές του κώδικα (LOC), ποσοστό σχολίων του κώδικα, ο χρόνος εκτέλεσης, λάθη του κώδικα (bugs) κλπ. Τα εξωτερικά χαρακτηριστικά αφορούν την λειτουργικότητα (functionality), την ποιότητα (quality), την πολυπλοκότητα (complexity), την αποτελεσματικότητα (efficiency), την αξιοπιστία (reliability) και την συντηρησιμότητα (maintainability).



Σχήμα 2.1: Μοντέλο FCM, McCall

Στην παρούσα διπλωματική εργασία δεν μελετούμε την ποιότητα του λογισμικού από την πλευρά του τελικού χρήστη που θα παραλάβει το λογισμικό αλλά από την πλευρά του προγραμματιστή που στην υλοποίηση του, οφείλει να εφαρμόσει καλές πρακτικές. Μελετούμε δηλαδή την ποιότητα του κώδικα ο οποίος πρέπει να έχει όσο το δυνατό πιο λίγα λάθη (bugs), να είναι ευανάγνωστος και καλογραμμένος και σωστά τεκμηριωμένος. Είναι γενικά αποδεκτό και σύμφωνα με το πρότυπο ISO 9126 [16] ότι το τελικό πρόγραμμα θα πρέπει να πληροί κάποια βασικά κριτήρια. Τα κριτήρια αυτά είναι τα εξής:

1. Αξιοπιστία (reliability): η δυνατότητα να αποφεύγονται προβλήματα που είναι αποτελέσματα λογικών λαθών, η δυνατότητα ανεκτικότητας σε λάθη (fault tolerance) και η ανακτησιμότητα των δεδομένων σε περίπτωση αποτυχίας.
2. Αποδοτικότητα (efficiency): η δυνατότητα να γίνεται όσο το δυνατόν πιο αποδοτική αξιοποίηση των πόρων του συστήματος και να παρέχεται καθορισμένος και αποδεκτός χρόνος απόκρισης του λογισμικού.
3. Αναγνωσιμότητα (readability): Ο παραγόμενος κώδικας πρέπει να είναι ευανάγνωστος και κατανοητός ούτως ώστε οποιοσδήποτε προγραμματιστής να μπορέσει να διαβάσει τον κώδικα και να τον ενημερώσει.
4. Μεταφερσιμότητα (portability): Ο παραγόμενος κώδικας να μπορεί να μεταφέρεται σε οποιοδήποτε περιβάλλον και να μπορεί να εκτελεστεί χωρίς κάποια επιπλέον προσπάθεια επαναπρογραμματισμού ή να χρησιμοποιηθεί στο τμήμα ενός άλλου λογισμικού.
5. Συντηρισιμότητα (maintainability): Η δυνατότητα πρόβλεψης ή διάγνωσης του βαθμού ανεπάρκειας λαθών στο λογισμικό, η ευκολία υλοποίησης αλλαγών και τροποποίησης του λογισμικού, η δυνατότητα ελαχιστοποίησης λαθών που οφείλονται στις τροποποιήσεις του λογισμικού.

Τα διάφορα εργαλεία στατικής ανάλυσης κώδικα εφαρμόζουν διάφορες μετρήσεις με σκοπό να παράξουν έναν πιο ποιοτικό κώδικα. Οι μετρήσεις διεξάγονται με τη χρησιμοποίηση των μετρικών. Σύμφωνα με τον Fenton [11] η μετρική είναι μια εμπειρική και αντικειμενική αντιστοίχηση ενός αριθμού ή συμβόλου σε μια οντότητα με σκοπό να χαρακτηρίσει ένα συγκεκριμένο χαρακτηριστικό της. Οι μετρικές χωρίζονται σε μετρικές προϊόντος (μετρικές ποιότητας κώδικα) που αφορούν τον προϊόν, δηλαδή τον πηγαίο κώδικα ή τις δηλώσεις ελέγχου, μετρικές έργου που σχετίζονται με την στρατηγική και την τακτική εκτέλεσης του έργου και καθορίζουν την ροή του έργου και τις τεχνικές που θα ακολουθηθούν και μετρικές διαδικασίας που αφορούν την διαδικασία κατασκευής του προϊόντος, όπως το σχεδιασμό, τη συγγραφή κώδικα και τους απαιτούμενους πόρους.

Η μέτρηση χαρακτηριστικών του λογισμικού είναι μια διαδικασία απαραίτητη για την εκτίμηση της κατάστασης του λογισμικού. Με τις μετρήσεις διασφαλίζεται πόσο καλά έχει συγγραφεί ο πηγαίος κώδικας, πόσο εφικτό είναι η επαναχρησιμοποίηση του

πηγαίου κώδικα σε άλλα έργα παραγωγής λογισμικού, πόσο συντηρήσιμος είναι ο πηγαίος κώδικας.

Οι πρώτες μετρικές προϊόντος παρουσιάστηκαν προς το τέλος της δεκαετίας του 70 από τον Halstead [14] και τον McCabe [24]. Τις ακολούθησαν ένας μεγάλος αριθμός μετρικών που προτάθηκαν τα επόμενα χρόνια και συνεχίζουν να προτείνονται ακόμα και σήμερα.
Στην πλειοψηφία τους, οι μετρικές προϊόντος περιορίζονται στη μέτρηση μεμονωμένων εσωτερικών χαρακτηριστικών του λογισμικού.
Πιο κάτω θα περιγράψουμε τις σημαντικότερες μετρικές προϊόντος που εφαρμόζονται σε αντικειμενοστρεφή η μη λογισμικό:

1. Lines of code (LOC): Μετρά τον συνολικό αριθμό των γραμμών του πηγαίου κώδικα ενός τμήματος ή του προγράμματος. Υπάρχουν 2 τύποι Γραμμών κώδικα. Οι φυσικές γραμμές κώδικα (LOC) και οι λογικές γραμμές κώδικα (LLOC). Οι φυσικές γραμμές κώδικα είναι η αρίθμηση των γραμμών στο κείμενο του πηγαίου κώδικα του προγράμματος συμπεριλαμβανομένων και των σχολίων και των κενών γραμμών. Οι λογικές γραμμές κώδικα είναι ο αριθμός των γραμμών του εκτελέσιμου κώδικα.
Η μετρική LOC είναι μια ένδειξη του βαθμού της πολυπλοκότητας και συνεπώς της κατανοησιμότητας, της αναγνωσιμότητας και της συντηρησιμότητας του κώδικα. Όσο αυξάνονται οι γραμμές του κώδικα τόσο αυξάνεται η πολυπλοκότητα του λογισμικού.
Παραλλαγές της μετρικής αυτής είναι:
Commented lines of code (CLOC): Ο συνολικός αριθμός των γραμμών που είναι σχόλια.
Non commented lines of code (NCLOC): Ο συνολικός αριθμός των γραμμών του πηγαίου κώδικα που δεν είναι σχόλια.
Comment Density (CD): Ο λόγος των σχολίων σε σχέση με το μέγεθος του αρχείου του πηγαίου κώδικα.
Method Lines of Code (MLOC): Ο αριθμός των γραμμών στις μεθόδους συμπεριλαμβανομένου και των κενών γραμμών και των σχολίων.
2. Number of statements: Είναι ο αριθμός των εντολών που υπάρχουν στο πρόγραμμα.
3. Number of public methods (NPM): Ο αριθμός των δημοσίων μεθόδων.

4. Nested Block Depth (NBD) ή Percent Branch statements: Το βάθος ή ποσοστό εντολών διακλάδωσης όπως για παράδειγμα το if, else, for, while, break, continue, goto, switch, case, default, return, catch. Πολλά φωλιασμένα μπλοκ (blocks) κάνουν τον κώδικα δυσανάγνωστο.

5. Number of classes: Ο αριθμός των κλάσεων.

6. Methods per classes ή Number of methods (NOM): Ο αριθμός των μεθόδων ανά κλάση. Θεμιτή τιμή για την μετρική NOM είναι μεταξύ του 3 και του 7.

7. Number of Static Methods (NSM): Ο αριθμός των στατικών μεθόδων. Οι στατικές κλάσεις είναι πιο γρήγορες από τις δυναμικές άρα οι στατικές μέθοδοι βοηθούν στην απόδοση. Η μεγάλη χρήση τους όμως δεν διευκολύνει την επαναχρησιμοποίηση.

8. Average statements per method: Ο αριθμός των εντολών δια τον αριθμό των μεθόδων.

9. Depth of inheritance tree (DIT): Είναι η μέγιστη απόσταση από τον κόμβο της κλάσης μέχρι τη ρίζα του δέντρου. Δέντρα με μεγάλο βάθος χαρακτηρίζονται από μεγαλύτερη πολυπλοκότητα, αφού κληρονομούνται πολλές μέθοδοι και κλάσεις. Θεμιτές τιμές θεωρούνται οι τιμές ανάμεσα στο 0 και στο 4.

10. Number of children (NOC): Αριθμός άμεσων απογόνων. Υψηλό NOC θα πει μεγάλη επαναχρησιμοποίηση κώδικα. Οι κλάσεις που βρίσκονται υψηλότερα στην ιεραρχία κληρονομικότητας είναι θεμιτό να έχουν υψηλότερο NOC από όσες βρίσκονται χαμηλότερα αφού είναι λιγότερο εξειδικευμένες. Μια θεμιτή τιμή είναι μεταξύ του 1 και του 4.

11. Coupling between Object classes (CBO): Σύζευξη μεταξύ κλάσεων. Το πλήθος των κλάσεων από τις οποίες εξαρτάται η τρέχουσα κλάση. Υψηλό CBO σημαίνει μεγάλη σύζευξη που αποτρέπει την επαναχρησιμοποίηση του κώδικα. Υψηλές τιμές CBO είναι οι τιμές που είναι μεγαλύτερες του 14 (περίπου). Θεμιτές τιμές θεωρούνται οι τιμές ανάμεσα στο 1 και στο 4.

12. Response for a class (RFC): Απόκριση για μια κλάση. Το πλήθος των μεθόδων (τοπικών ή μεθόδων που καλούνται άμεσα από την τρέχουσα κλάση) που εκτελούνται ως απάντηση στη λήψη ενός συμβάντος από την τρέχουσα κλάση.

13. Lack of cohesion in Methods (LCOM): Έλλειψη συνεκτικότητας στις μεθόδους. Το πλήθος των μη επικαλυπτόμενων τοπικών μεθόδων μιας κλάσης. Αν η τιμή LCOM > 0 τότε η κλάση είναι καλύτερα να διασπαστεί. Υπάρχουν 4 εκδοχές της μετρικής LCOM.

LCOM1:

LCOM1 = P – Q (αν P > Q), διαφορετικά LCOM1 = 0.

Παίρνουμε κάθε ζευγάρι μεθόδων σε μια κλάση. Αν μοιράζονται ασύνδετα σύνολα μεταβλητών αυξάνουμε το P κατά 1 (καμία κοινή μεταβλητή). Αν μοιράζονται τουλάχιστον μια μεταβλητή, τότε αυξάνουμε το Q κατά 1.

LCOM2:

LCOM2 = 1 - sum(mA)/(m*a), όπου m ο αρ. των μεθόδων της κλάσης, a ο αρ. των χαρακτ./μεταβλητών της κλάσης, mA ο αρ. των μεθόδων που έχουν πρόσβαση σε ένα χαρακτ./μεταβλητή, sum(mA) το άθροισμα όλων των mA στο σύνολο των χαρακτ./μεταβλητών της κλάσης.

Η LCOM2 ισούται με το ποσοστό των μεθόδων που δεν προσπελούν κάποιο χαρακτηριστικό της κλάσης σε σχέση με όλα τα χαρακτηριστικά της κλάσης (εξάγοντας το μέσο όρο όλων των χαρακτηριστικών στην κλάση).

LCOM3:

LCOM3 = (m - sum(mA)/a) / (m-1) όπου m ο αρ. των μεθόδων της κλάσης, a ο αρ. των χαρακτ./μεταβλητών της κλάσης, mA ο αρ. των μεθόδων που έχουν πρόσβαση σε ένα χαρακτ./μεταβλητή, sum(mA) το άθροισμα όλων των mA στο σύνολο των χαρακτ./μεταβλητών της κλάσης

Οι τιμές μεταξύ 0 και 2. Από 1-2 θεωρούνται προβληματικές. Όταν η LCOM3=0, τότε κάθε μέθοδος έχει πρόσβαση σε όλες τις μεταβλητές

LCOM4: Η LCOM4 είναι η τέταρτη εκδοχή της LCOM. Η εκδοχή αυτή μετρά των αριθμό των συνδεδεμένων στοιχείων (connected components) σε μία κλάση. Ένα συνδεδεμένο στοιχείο είναι ένα σύνολο από συσχετιζόμενες μεθόδους. Αν υπάρχουν 2 ή περισσότερα συνδεδεμένα στοιχεία τότε η κλάση θα πρέπει να διασπαστεί σε περισσότερες κλάσεις ούτως ώστε να μειωθεί η πολυπλοκότητα της κλάσης.

14. Cyclomatic Complexity ή Conditional Complexity (CC ή Vg): Η κυκλωματική πολυπλοκότητα έχει εισαχθεί από τον McCabe (1976) και είναι κριτήριο για την εκτίμηση της πολυπλοκότητας μιας μονάδας π.χ μίας μεθόδου. Συγκεκριμένα η τιμή της ορίζεται ως ο αριθμός των διαφορετικών μονοπατιών στη ροή μίας μονάδας και ο τύπος της είναι CC=P+1, όπου P ο αριθμός των Boolean αποφάσεων (if, for, while, case, &&, ||, ?) σε μονάδα. Περισσότερα μονοπάτια σημαίνει περισσότερη πολυπλοκότητα. Η μείωση της τιμής της συνεπάγεται περιορισμός της πολυπλοκότητας και αύξηση των δυνατοτήτων ανάγνωσης και κατανόησης του πηγαίου κώδικα. Αν η

τιμή της ξεπερνά το επιτρεπόμενο όριο τότε συνιστάται η τροποποίηση του κώδικα για να ελαττωθεί ο αριθμός των φωλιασμένων εντελών απόφασης ή η διάσπασης της μονάδας λογισμικού σε περισσότερες μονάδες.

15. Weighted Methods per Class (WMC): Σταθμισμένες μέθοδοι ανά κλάση, δηλαδή το άθροισμα της κυκλωματικής πολυπλοκότητας όλων των μεθόδων σε μία κλάση. Μεγάλο WMC οδηγεί σε μεγαλύτερη πιθανότητα σφαλμάτων και προβλημάτων συντήρησης. (WMC = $\sum_{i=1}^{n} cci$)**.**

16. Afferent Coupling (CA): Ο αριθμός των κλάσεων έξω από το τρέχον πακέτο που εξαρτώνται από τις κλάσεις μέσα στο τρέχον πακέτο.

17. Efferent Coupling (CE): Ο αριθμός των κλάσεων μέσα στο τρέχον πακέτο που εξαρτώνται από τις κλάσεις έξω από το τρέχον πακέτο.

18. Instability (I ή RMI): Αστάθεια. I = Ce / (Ca + Ce). Η μετρική I της αστάθειας κινείται στο [0-1], όπου το 0 δείχνει τη μέγιστη ευστάθεια και το 1 τη μέγιστη αστάθεια.

19. Number of Overridden Methods (NORM): Υποσκέλιση μεθόδων. Ο αριθμός των μεθόδων που υποσκελίζονται από την γονική κλάση. Υποσκέλιση μεθόδου γίνεται όταν μία θυγατρική κλάση και η γονική της έχουν μία μέθοδο ομώνυμη και με τα ίδια ορίσματα. Χάρη στη δυνατότητα του πολυμορφισμού ο μεταγλωττιστής γνωρίζει πότε να καλέσει ποια μέθοδο, βασισμένος στον τύπο του τρέχοντος αντικειμένου. Θεμιτή τιμή για την μετρική NORM θεωρείται η τιμή ανάμεσα στο 0 και στο 5.

20. Number of packages (NOP): Ο αριθμός των πακέτων.

21. Number of Static Attributes (NSF): Ο αριθμός των static χαρακτηριστικών.

22. Abstractness (RMA): Ο αριθμός των αφηρημένων (abstract) κλάσεων.

23. Number of parameters (PAR): Ο αριθμός των παραμέτρων.

24. Number of Interfaces (NOI): Ο αριθμός των διεπαφών. Μεγάλος αριθμός διεπαφών κάνουν τον κώδικα πιο ευέλικτο και επαναχρησιμοποιήσιμο.

25. Normalized Distance (RMD): | RMA + RMI - 1 |. Ο αριθμός αυτός πρέπει να είναι κοντά στο μηδέν για καλύτερη σχεδίαση των πακέτων.

26. Specialization Index (SIX): NORM * DIT / NOM.

27. Number of Attributes (NOF) ή Number of Variables per class: Ο αριθμός όλων των ιδιοτήτων (properties) μιας κλάσης. Μια κλάση με υψηλό NOF (>10) πιθανόν να σημαίνει φτωχό σχεδιασμό και να απαιτεί αποσύνθεση ούτως ώστε να μειωθεί η πολυπλοκότητα της κλάσης.

2.3 Έλεγχος λογισμικού

Ο έλεγχος λογισμικού είναι μια εμπειρική διαδικασία με την οποία εξασφαλίζονται πληροφορίες σχετικά με την ποιότητα του προϊόντος. Επίσης μπορεί να είναι μια ενδεικτική μελέτη που αποσκοπεί στην κατανόηση των ρίσκων που υπάρχουν με την υλοποίηση του προϊόντος. Ο έλεγχος λογισμικού αποσκοπεί στην επαλήθευση (verification) ότι το προϊόν έχει ολοκληρωθεί σωστά, λειτουργεί όπως αναμενόταν και στην επικύρωση (validation) ότι το προϊόν ικανοποιεί τις προδιαγραφές του και τον πελάτη.

Ο έλεγχος μπορεί να γίνει σε οποιαδήποτε φάση της διαδικασίας ανάπτυξης λογισμικού αφού όμως έχει ολοκληρωθεί η συγγραφή του κώδικα. Είναι σημαντικό να τονίσουμε ότι ο έλεγχος δεν εγγυάται ότι θα επιστρέψει όλα τα λάθη του προϊόντος αλλά εγγυάται ότι θα βρει πως το προϊόν δεν λειτουργεί σωστά σύμφωνα με τις προδιαγραφές που έχουν καθοριστεί.

Ο έλεγχος ενός λογισμικού όσον αφορά το τεχνικό επίπεδο έχει ως άμεσο στόχο την εύρεση προγραμματιστικών λαθών. Συχνά οι προγραμματιστές κάνουν λάθη που οδηγούν σε σφάλματα (faults,bugs) και έτσι όταν το πρόγραμμα εκτελεστεί πιθανόν να επιστρέψει λανθασμένα αποτελέσματα και συνεπώς θα αποτύχει. Λάθη μπορεί να εμφανιστούν στο κώδικα και να είναι είτε συντακτικής είτε λογικής φύσεως. Τα συντακτικά λάθη αντιμετωπίζονται με τη χρήση διερμηνέα ή μεταγλωττιστή. Τα λογικά λάθη από την άλλη είναι δύσκολο να εντοπισθούν. Μια μελέτη από την NIST το 2002 [27] αναφέρει ότι τα λογισμικά λάθη (software bugs) κοστίζουν στην αμερικανική οικονομία $59.5 δισεκατομμύρια τον χρόνο. Ωστόσο το ένα τρίτο του ποσού αυτού μπορεί να αποφευχθεί αν εφαρμόζεται σωστός έλεγχος λογισμικού. Όσο αργότερα στην διαδικασία ανάπτυξης αρχίσει να γίνεται ο έλεγχος τόσο πιο δύσκολο είναι να επιτευχθεί το επιθυμητό αποτέλεσμα. Ο έλεγχος λογισμικού περιλαμβάνει διάφορους μεθόδους ελέγχου. Περιλαμβάνει την στατική ανάλυση κώδικα που είναι και η μέθοδος που θα αναλύσουμε στην παρούσα εργασία, την δυναμική ανάλυση κώδικα που θα περιγράψουμε πιο κάτω και που μελετήθηκε σε λιγότερη έκταση. Υπάρχουν και άλλοι μέθοδοι ελέγχου λογισμικού όπως το black box testing, το white box testing, το grey box testing και το visual testing.

Ο έλεγχος black box ελέγχει τη λειτουργικότητα μιας εφαρμογής χωρίς να βασίζεται στις εσωτερικές δομές του προγράμματος δηλαδή τον κώδικα. Είναι σαν ένα μαύρο κουτί το οποίο δεν γνωρίζουμε τον τρόπο με τον οποίο λειτουργεί αλλά γνωρίζουμε τι κάνει. Η μέθοδος του black box, μπορεί να εφαρμοστεί σε όλα τα επίπεδα του ελέγχου λογισμικού, όπως unit testing, integration testing, system testing και acceptance testing.

Το unit testing είναι η μέθοδος που ελέγχει ξεχωριστά κομμάτια κώδικα για να κρίνει εάν είναι έτοιμα για να χρησιμοποιηθούν. Το unit testing είναι το μικρότερο κομμάτι που μπορεί να ελεγχθεί από μια συγκεκριμένη εφαρμογή. Ο στόχος του είναι να απομονώσει κάθε κομμάτι του προγράμματος και να αποφασίσει εάν τα κομμάτια αυτά είναι ορθά.

Το integration testing είναι η μέθοδος κατά την οποία ξεχωριστές ενότητες του λογισμικού ενώνονται και ελέγχονται σαν σύνολο. Γίνεται μετά από πολλαπλά unit tests και πριν από το system testing. Παίρνει όλα τα inputs που χρησιμοποιήθηκαν σε κάθε unit ξεχωριστά και τα συνενώνει με τρόπο τέτοιο ώστε να δημιουργηθούν inputs τέτοια ώστε να ανταποκρίνονται σε ένα πλάνο το οποίο θέλει να ελέγξει το κομμάτι του integration.

Το system testing είναι η μέθοδος που διεξάγεται σε ένα ολοκληρωμένο, σύστημα με σκοπό να αξιολογήσει εάν ακολουθεί τις προδιαγραφές του συστήματος που έχουν καθοριστεί σε προηγούμενη φάση. Ο έλεγχος αυτός γίνεται στα επίπεδα του black box testing και έτσι δεν είναι απαραίτητη η γνώση για την εσωτερική σχεδίαση του κώδικα. Σαν κανόνα το system testing, παίρνει ως δεδομένα εισόδου όλα τα δεδομένα που χρησιμοποιήθηκαν πιο πριν στα κομμάτια που πέρασαν τον έλεγχο του integration testing.

Το acceptance testing είναι η μέθοδος στην οποία ελέγχεται αν το προϊόν ικανοποιεί τις απαιτήσεις του χρήστη. Γίνεται συνήθως σε συνεργασία με τον χρήστη.

Τα σενάρια ελέγχου (test cases) που χρησιμοποιούνται στον έλεγχο black box, δημιουργούνται με βάση την εξωτερική περιγραφή του συστήματος λαμβάνοντας υπόψη τις προδιαγραφές, αλλά και την σχεδίαση του συστήματος. Ο ελεγκτής, επιλέγει

και δοκιμάζει έγκυρα αλλά και λανθασμένα δεδομένα εισόδου και αποφασίζει τα ορθά αποτελέσματα εξόδου.

Ο έλεγχος white box (διαφορετικά glass box) ελέγχει τις εσωτερικές δομές του λογισμικού δηλαδή των κώδικα. Ο ‘ελεγκτής’ δίνει στο λογισμικό δεδομένα εισόδου για να επεξεργαστεί όλες τις εκδοχές του λογισμικού. Η μμέθοδος του white Box, μπορεί να εφαρμοστεί στα επίπεδα Unit Testing, Integration Testing, και System Testing και μπορεί να ανακαλύψει πολλά προβλήματα όπως περιπτώσεις νεκρού κώδικα όμως υστερεί στην εύρεση λογικών λαθών, και είναι αδύνατον να εξεταστούν όλα τα μονοπάτια του κώδικα έτσι ώστε να βρεθούν όλα τα πιθανά λάθη.

Ο έλεγχος grey box είναι ο συνδυασμός του ελέγχου black box και white box. Ο κώδικας είναι γνωστός και ο βασικός σκοπός χρήσης του είναι η επαναχρησιμοποίηση των σεναρίων ελέγχων αφού γίνουν πιθανές αλλαγές, η προσθήκη στον κώδικα και η ανίχνευση λαθών που έχουν προκληθεί από κακό σχεδιασμό ή κακή υλοποίηση του λογισμικού.

Ο visual έλεγχος ελέγχει το γραφικό περιβάλλον του λογισμικού. Η τεχνική αυτή ηχογραφεί και καταγράφει σε μορφή video όλη την διαδικασία ελέγχου. Αυτή η τεχνική παρέχει στον χρήστη μια πλήρη εικόνα του λογισμικού αφού ηχογραφεί ήχο και εικόνες καθ’ όλη την διάρκεια του ελέγχου.

2.4 Στατική ανάλυση κώδικα

Ο στατικός έλεγχος είναι ένας έλεγχος κατά τον οποίο το λογισμικό δεν εκτελείται. Εξετάζει συνήθως τον πηγαίο κώδικα για τυχόν συντακτικά και λογικά λάθη με σκοπό την βελτίωση της ποιότητας του κώδικα. Οι μετρικές (software metrics) που αναφέραμε πιο πάνω καθώς και η αντίστροφη μηχανική (reverse engineering) είναι μορφές στατικής ανάλυσης. Για να είναι αποδοτική η στατική ανάλυση πρέπει να γίνει όσο το δυνατόν πιο γρήγορα γίνεται και φυσικά αφού έχει πραγματοποιηθεί η συγγραφή του κώδικα.

Υπάρχουν επίσης και άλλοι τρόποι χρήσης της στατικής ανάλυσης κώδικα. Για παράδειγμα, μπορεί να χρησιμοποιηθεί ως μέθοδος με σκοπό τον έλεγχο και την

διδασκαλία νέων προγραμματιστών σε έναν οργανισμό που δεν γνωρίζουν ακόμα τους προγραμματιστικούς κανόνες του οργανισμού.

Οι τεχνικές που χρησιμοποιεί η στατική ανάλυση κυρίως είναι η ανάλυση ροής δεδομένων και η syntactic pattern matching . Άλλες τεχνικές που χρησιμοποιεί είναι η αφηρημένη διερμηνεία και ο έλεγχος μοντέλων.

Ο στατικός έλεγχος μπορεί να γίνει από διάφορα αυτοματοποιημένα εργαλεία εμπορικά ή μη όπως θα δούμε και στην συνέχεια, τα οποία ελέγχουν τον πηγαίο (source code) ή τον ενδιάμεσο (byte code) κώδικα με τις διάφορες μετρικές καθώς και άλλους κανόνες (rules) με σκοπό την μέτρηση της πολυπλοκότητας του κώδικα και τον εντοπισμό προβλημάτων όπως η υπερχείλιση της ενδιάμεσης μνήμης (butter overflow), η αναφορά σε κενό δείκτη (null pointer dereference) κ.α καθώς και λάθη συντακτικά και στιλιστικά.

2.4.1 Πλεονεκτήματα στατικής ανάλυσης κώδικα

Το κύριο πλεονέκτημα της στατικής ανάλυσης κώδικα είναι το εξής:

1. Δίνει την δυνατότητα να ελαττώσει σημαντικά τα λάθη του προγράμματος. Όσο πιο γρήγορα εντοπίζεται το λάθος τόσο πιο λίγο θα κοστίσει η διόρθωση του. Σύμφωνα με τον McConnell και το βιβλίο του "Code Complete" η διόρθωση ενός λάθους στην φάση του ελέγχου κοστίζει 10 φορές περισσότερο από την φάση συγγραφής του κώδικα. Τα εργαλεία στατικής ανάλυσης μας επιτρέπουν να εντοπίσουμε τα λάθη γρήγορα κατά την φάση της συγγραφής του κώδικα. Τα περισσότερα τρέχουν αυτόματα στο παρασκήνιο μόλις το πρόγραμμα γίνει build ή ακόμα πριν καν εκτελεστεί και ενημερώνουν τον προγραμματιστή για τα πιθανά λάθη που υπάρχουν.

Άλλα πλεονεκτήματα της στατικής ανάλυσης κώδικα είναι τα εξής:

2. Πλήρης Κάλυψη Μονοπατιών (code coverage). Η στατική ανάλυση κώδικα καλύπτει όλα τα δυνατά μονοπάτια που υπάρχουν στον κώδικα βρίσκοντας λάθη όπως λανθασμένος χειρισμός των exceptions, αναφορά σε κενό δείκτη (null pointer dereferences), buffer overflow, buffer και array underflows, μη επιστροφή return τιμής, εγγραφή σε read-only memory, χρησιμοποίηση μνήμης που έχει ήδη αποδεσμευτεί κ.ά .
3. Εντοπισμός νεκρού κώδικα (dead code). Ο νεκρός κώδικας είναι ο κώδικας που δεν εκτελείται και προφανώς δεν χρειάζεται. Ο εντοπισμός του νεκρού κώδικα είναι

σημαντικός καθώς ο νεκρός κώδικας μειώνει την απόδοση του προγράμματος και δυσκολεύει την διαδικασία συντήρησης.

4. Εντοπισμός συντακτικών λαθών. Τα διάφορα εργαλεία προτείνουν οδηγίες που αφορούν την σύνταξη και το στυλ που είναι γραμμένος ο κώδικας καθιστώντας τον πιο ευανάγνωστο.

5. Εύκολος και γρήγορος έλεγχος σε μεγάλη τμήματα κώδικα. Τα αυτοματοποιημένα εργαλεία εκτελούν εύκολα και γρήγορα ελέγχους σε μεγάλα τμήματα κώδικα. Αν χρειαστεί επανέλεγχος του κώδικα, μπορεί να γίνει σχετικά εύκολα.

6. Εντοπισμός των λαθών στο σημείο που υπάρχει το λάθος. Η στατική ανάλυση παρέχει στον προγραμματιστή το σημείο που εντοπίζεται το πρόβλημα.

2.4.2 Μειονεκτήματα Στατικής ανάλυσης κώδικα

1. Το σημαντικότερο μειονέκτημα της στατικής ανάλυσης κώδικα είναι ότι η ανάλυση μπορεί να επιστρέψει false-positives. Με τον όρο false-positives εννοούμε τα λάθη που εντοπίζει η στατική ανάλυση στον κώδικα που στην πραγματικότητα δεν υπάρχουν. Δηλαδή, ενώ ο κώδικας είναι ορθός η στατική ανάλυση εντοπίζει λάθος.

Τα περισσότερα εργαλεία που δεν υπολογίζουν μόνο μετρικές επιστρέφουν κάποιο ποσοστό false-positive’s, αλλά υπάρχουν εργαλεία που παρέχουν σχετικές ρυθμίσεις ούτως ώστε να μην υπολογίζονται κανόνες που επιστρέφουν false positives.

2. Η διάγνωση προβλημάτων όπως memory leaks και προβλήματα συγχρονισμού (concurrency errors) είναι σχετικά ‘φτωχή’.

2.5 Δυναμική ανάλυση κώδικα

Η δυναμική ανάλυση κώδικα είναι η ανάλυση του κώδικα που γίνεται σε επίπεδο εκτέλεσης, δηλαδή όταν το πρόγραμμα εκτελείται. Για να γίνει η δυναμική ανάλυση το πρόγραμμα πρέπει να μεταγλωττιστεί και να εκτελεστεί. Σκοπός την δυναμικής ανάλυσης είναι η μείωση του χρόνου της μεταγλώττισης (debugging time) μέσω της αυτόματης ανίχνευσης και επεξήγησης πιθανών λαθών. Στην δυναμική ανάλυση δεν ελέγχονται οι γραμμές του κώδικα με την σειρά αλλά ελέγχονται μόνο οι γραμμές που ενεργοποιούνται στην διαδικασία εκτέλεσης. Η εκτέλεση του κώδικα γίνεται με βάση ένα σύνολο από σενάρια ελέγχου (test cases).

2.5.1 Πλεονεκτήματα Δυναμικής ανάλυσης κώδικα

Τα πλεονεκτήματα της δυναμικής ανάλυσης κώδικα είναι τα εξής:

1. Εντοπίζει run-time λάθη όπως memory leaks, security errors, threads and synchronization errors, exceptions errors, database errors.
2. Μπορεί να ανιχνεύσει εξαρτήσεις που η στατική ανάλυση δεν μπορεί να βρει.

2.5.2 Μειονεκτήματα Δυναμικής ανάλυσης κώδικα

Τα μειονεκτήματα της δυναμικής ανάλυσης κώδικα είναι τα εξής:

1. Η δυναμική ανάλυση είναι πολύπλοκη. Για την κατανόηση του λάθους που έχει επιστρέψει η δυναμική ανάλυση ο προγραμματιστής πρέπει να εντοπίσει από την αρχή την πηγή του προβλήματος και να διορθώσει το πρόβλημα και να κάνει αλλαγές που εξαρτώνται από την διόρθωση του προβλήματος. Αυτό είναι χρονοβόρο και συνεπώς είναι μια ‘ακριβή’ διαδικασία.
2. Δεν μπορεί να εγγυηθεί ότι έλεγξε όλα τα πιθανά μονοπάτια του κώδικα.

Κεφάλαιο 3

Εργαλεία στατικής ανάλυσης κώδικα

3.1 Εισαγωγή

Για τον έλεγχο της ποιότητας του παραγόμενου κώδικα έχουν αναπτυχθεί πολλά εργαλεία τα οποία κάνουν στατική ανάλυση κώδικα. Σε αυτό το κεφάλαιο θα αναλύσουμε την χρησιμότητα των εργαλείων στατικής ανάλυσης, τα 11 εργαλεία που επιλέχθηκαν για ανάλυση και στην συνέχεια θα γίνει αναφορά στην επιλογή και ανάλυση των εργαλείων με τα διάφορα έργα λογισμικού.

3.2 Χρησιμότητα εργαλείων στατικής ανάλυσης κώδικα

Η στατική ανάλυση κώδικα είναι μια περιοχή της τεχνολογίας λογισμικού που αναπτύσσεται ραγδαία και έτσι ολοένα και περισσότερα εργαλεία στατικής ανάλυσης κώδικα υπάρχουν στην αγορά και αναπτύσσονται συνεχώς.

Συχνά οι προγραμματιστές δεν εφαρμόζουν καλές πρακτικές κατά την συγγραφή του κώδικα με αποτέλεσμα ο κώδικας να μην έχει τη βέλτιστη ποιότητα και να περιέχει λάθη, κάποιες φορές σοβαρά και κάποιες άλλες λιγότερο σοβαρά. Συγκεκριμένα γίνεται συχνά κακή χρήση της πολυπλοκότητας και δε χρησιμοποιούνται έλεγχος μονάδων (unit tests) και προγραμματιστικές συμβάσεις (coding standards) στον κώδικα. Ακόμα μπορεί να γίνεται, λανθασμένη χρήση σχολίων και να επαναλαμβάνεται ο ίδιος κώδικας ή ακόμα να εντοπίζονται και πιο μικρά λάθη συντακτικά (έλλειψη παρενθέσεων κλπ). Όλα αυτά δημιουργούν στον κώδικα σημαντικά λάθη (bugs) που στοιχίζουν χρόνο για την αντιμετώπιση τους και συνεπώς χρήμα.

Είναι πολύ δύσκολο ο προγραμματιστής να ελέγξει χειροκίνητα μόνος του τον κώδικα στο σύνολο του. Εκτός του ότι είναι μια χρονοβόρα διαδικασία, πιθανόν να μην γνωρίζει τον λόγο που υπάρχει λάθος και κυρίως πού βρίσκεται το λάθος, (γιατί διαφορετικά θα απέφευγε από την αρχή το λάθος).

Για την πιστοποίηση της ποιότητας του παραγόμενου κώδικα έχουν αναπτυχθεί εργαλεία τα οποία κάνουν στατική ανάλυση κώδικα. Διαβάζουν και αναλύουν γραμμή γραμμή τον πηγαίο ή ενδιάμεσο κώδικα ελέγχοντας για τυχόν λάθη, παραλήψεις ή μη αποδοτικές τεχνικές προγραμματισμού. Τα εργαλεία εντοπίζουν και δείχνουν τις παραβιάσεις στο σημείο ακριβώς που βρίσκονται. Η ανάλυση είναι μία διαδικασία σχετικά γρήγορη και εύκολη και δεν καθυστερεί περεταίρω την ανάπτυξη του λογισμικού. Αντιθέτως, εντοπίζοντας τα πιθανά λάθη, στην φάση της συγγραφής του κώδικα διευκολύνεται ο έλεγχος του λογισμικού που πραγματοποιείται σε μεταγενέστερο στάδιο καθώς αποφεύγονται πιθανά λάθη που θα κόστιζαν χρόνο για την διόρθωση τους. Επίσης βοηθά τον προγραμματιστή να κατανοήσει τα λάθη του με αποτέλεσμα να τα αποφύγει στο μέλλον. Ακόμα, πολλά από τα εργαλεία παρέχουν την δυνατότητα συγγραφής κανόνων ελέγχων. Με αυτή την λειτουργία, μια εταιρεία ή ένας οργανισμός θα μπορούσε να καθορίσει ένα σύνολο από ελέγχους οι οποίοι θα καθορίζουν συγκεκριμένες οδηγίες που θα πρέπει να πληροί ο κώδικας. Με τα εργαλεία λοιπόν, μπορούμε να παρακολουθήσουμε και να ελέγξουμε την εξέλιξη του προγράμματος καθώς αναπτύσσεται.

Τα εργαλεία στατικής ανάλυσης κώδικα είναι σημαντικά για την ανάπτυξη ενός συστήματος λογισμικού. Σήμερα οργανισμοί και εταιρείες ελέγχουν τον κώδικα τους με τα εργαλεία αυτά, στην προσπάθεια τους να εξασφαλίσουν κώδικα ποιοτικό, χωρίς λάθη, ευανάγνωστο και συντηρήσιμο.

Βέβαια η ποιότητα του κώδικα δεν μπορεί να διασφαλιστεί μόνο κάνοντας χρήση τέτοιου είδους εργαλείων. Υπάρχουν περιπτώσεις κατά τις οποίες γίνεται σπατάλη πόρων ή υπάρχουν λογικά λάθη τα οποία δεν μπορούν να εντοπιστούν από κάποιο εργαλείο ή ακόμα και περιπτώσεις όπου τα εργαλεία επιστρέφουν πολλά λανθασμένα αποτελέσματα (false positives). Τα εργαλεία στατικής ανάλυσης διευκολύνουν σε μεγάλο βαθμό την φάση ελέγχου του λογισμικού αλλά δεν εγγυώνται την απουσία λαθών κατά την εκτέλεση του προγράμματος και σε καμία περίπτωση δεν αντικαταστούν την φάση ελέγχου. Εντούτοις, η χρήση τους δίνει μια καλή ένδειξη για την ποιότητα του παραγομένου κώδικα.

3.3 Επιλογή εργαλείων στατικής ανάλυσης κώδικα

Σήμερα υπάρχουν στην βιομηχανία πολλά εργαλεία στατικής ανάλυσης κώδικα, εμπορικά η μη. Για την παρούσα διπλωματική εργασία επιλέχθηκε ένας μικρός αριθμός εργαλείων και πραγματοποιήθηκε μια σύγκριση μεταξύ τους. Για την επιλογή των εργαλείων λήφθηκαν υπόψη τα παρακάτω κριτήρια:

- κάλυψη διαδεδομένων γλωσσών προγραμματισμού
- βάθος της ανάλυσης
- κανόνες ελέγχου
- μετρικές ποιότητας κώδικα.

Η επιλογή των περισσότερων εργαλείων βασίστηκε επίσης στο πόσο διαδεδομένα είναι τα εργαλεία, και ποια από αυτά χρησιμοποιούνται περισσότερο από χρήστες και οργανισμούς. Επίσης για να είναι αποδεκτό ένα εργαλείο θα πρέπει να μην επιστρέφει μεγάλο ποσοστό λανθασμένων αποτελεσμάτων (false positives).

Για κάθε εργαλείο είναι διαθέσιμες οι παρακάτω πληροφορίες: το όνομα, ο κατασκευαστής, η επίσημη του ιστοσελίδα, ο τύπος άδειας λογισμικού, οι γλώσσες προγραμματισμού που υποστηρίζει και το λειτουργικό σύστημα στο οποίο μπορεί να

εκτελεστεί. Πραγματοποιώντας μια αναζήτηση στο διαδίκτυο ανακαλύψαμε ότι τα περισσότερα εργαλεία αναλύουν κώδικα γραμμένο στις πιο διαδεδομένες γλώσσες Java, C και C++. Με βάση τα κριτήρια που αναφέρθηκαν πιο πάνω εντοπίσαμε αρχικά 14 εργαλεία. Τελικά καταλήξαμε σε 11 εργαλεία λόγω της περιορισμένης εμπορικής άδειας που είχαν κάποια εργαλεία (που δε μας επέτρεπε την ανάλυσή του σε βάθος), καθώς και λόγω της παλαιάς ημερομηνίας δημοσίευσης ορισμένων εργαλείων (για τα οποία δεν υπήρχε διαθέσιμη κάποια πρόσφατη έκδοση).

Τα 11 εργαλεία καλύπτουν συνολικά μερικές από τις πιο κάτω γλώσσες προγραμματισμού:

1. Java
2. C
3. C++
4. C#
5. PHP

Για την σύγκριση των εργαλείων επίσης, επιλέξαμε εργαλεία που υπολογίζουν κοινές μετρικές ούτως ώστε να εξετάσουμε αν τα εργαλεία επιστρέφουν την ίδια τιμή, για την ίδια μετρική, για την ίδια είσοδο.

Τα 10 από τα 11 εργαλεία παρέχονται δωρεάν. Όλα τα εργαλεία εγκαταστάθηκαν και αναλύθηκαν σε λειτουργικό περιβάλλον Windows 64-bit. Τα 4 εργαλεία εγκαταστάθηκαν ως .IDE plug-in, τα 3 μπορεί να τα εκτελέσει ο χρήστης από την γραμμή εντολών (command line) και τα υπόλοιπα είναι ανεξάρτητα εργαλεία με την δική τους διεπαφή χρήστη (stand alone).

Σε αυτό το σημείο θα πρέπει να αναφέρουμε ότι τα εργαλεία δεν συμπεριφέρονται όλα με τον ίδιο τρόπο. Κάποια υπολογίζουν μόνο μετρικές κώδικα ενώ κάποια άλλα παρέχουν περισσότερες επιπρόσθετες λειτουργίες και εξετάζουν τον κώδικα με διάφορους ελέγχους βασισμένους σε προγραμματιστικούς ή και στιλιστικούς κανόνες (rules). Έτσι τα εργαλεία στατικής ανάλυσης μπορούν να χωριστούν σε 3 κατηγορίες.

Τα εργαλεία που υπολογίζουν μόνο μετρικές προϊόντος. Τα εργαλεία που υπολογίζουν μόνο κανόνες ελέγχου και τα εργαλεία που υπολογίζουν και τα 2.

Ο παρακάτω πίνακας παρουσιάζει με λεπτομέρειες τα εργαλεία που επιλέχτηκαν βάσει των χαρακτηριστικών που αναφέρθηκαν πιο πάνω.

Εργαλεία											
	Metrics 1.3.6	**LocMetrics**	**SourceMonitor**	**PMD**	**Checkstyle**	**FindBugs**	**Ckjm**	**Sonar qube**	**Vil**	**CppCheck**	**NDepend**
Γλώσσα Προγ/μού	Java	Java C# C++ Sql	Java C# C/C++ VB.NET Delphi Visual- Basic(VB6) HTML	Java JavaScript XML XSL	Java	Java	Java	Java C# C/C++ JavaScript PHP Flex Groovy Python PL/SQL COBOL VB.NET Erlang Visual-Basic 6 XML WEB	.NET	C/C++	.NET
Plug-in	✓			✓	✓	✓					
Γραμμή εντολών							✓		✓		
Stand alone		✓	✓					✓		✓	✓
Ελεύθερο	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Μετρικές Προϊόντος	✓	✓	✓	✓	✓		✓	✓	✓		✓
Κανόνες ελέγχου				✓	✓	✓		✓			

Πίνακας 3.1 : Εργαλεία στατικής ανάλυσης κώδικα

3.3.1 Metrics 1.3.6

Το εργαλείο Metrics[1] με έκδοση 1.3.6 είναι ένα ανοικτού κώδικα λογισμικό το οποίο διατίθεται ως plugin για το περιβάλλον προγραμματισμού Eclipse. Αναπτύχθηκε από τον Frank Sauer [32] και είναι διαθέσιμο από τον Σεπτέμβριο του 2007 από το SourceForge.

Το Metrics 1.3.6 αναλύει στατικά πηγαίο κώδικα σε Java και μπορεί να υπολογίσει όλες τις διαδεδομένες μετρικές. Επίσης το εργαλείο υπολογίζει τον γράφο (dependency graph) με τις εξαρτώμενες συνδέσεις ανάμεσα στις κλάσεις και στα πακέτα του λογισμικού.

Το εργαλείο να σημειώσουμε ότι απαιτεί η έκδοση του Eclipse να είναι τουλάχιστον η 3.1.

Αφού ενεργοποιηθεί η ανάλυση, υπολογίζονται οι μετρικές για κάθε μέθοδο και κλάση και εμφανίζονται κάτω από την όψη Metrics (Metrics View). Οι μετρήσεις που υπολογίζονται είναι οι εξής:

1. Lines of code (LOC)
2. Number of Static Methods (NSM)
3. Afferent Coupling (CA)
4. Normalized Distance (RMD)
5. Number of Classes
6. Specialization Index (SIX)
7. Instability (RMI)
8. Number of Attributes (NOF)
9. Number of Packages (NOP)
10. Method Lines of Code (MLOC)
11. Weighted Methods per Class (WMC)
12. Number of Overridden Methods (NORM)
13. Number of Static Attributes (NSF)
14. Nested Block Depth (NBD)
15. Number of Methods (NOM)
16. Lack of Cohesion of Methods (LCOM)

[1] www.metrics.sourceforge.net

17. McCabe Cyclomatic Complexity (VG or CC)
18. Number of Parameters (PAR)
19. Abstractness (RMA)
20. Number of Interfaces (NOI)
21. Efferent Coupling (CE)
22. Number of children (NOC)
23. Depth of Inheritance Tree (DIT)

Όπως βλέπουμε στο σχήμα 3.1, για ένα παράδειγμα της εκτέλεσης της ανάλυσης μέσω του εργαλείου Metrics εάν η τιμή της μετρικής είναι έξω από τα όρια του επιθυμητού πεδίου τιμών (range) τότε εμφανίζεται με κόκκινο χρώμα, διαφορετικά εμφανίζεται με μπλε χρώμα. Για κάθε μετρική υπολογίζεται η τιμή για κάθε μέθοδο του προγράμματος (εάν είναι μετρήσιμη σε αυτό το επίπεδο), η τιμή για κάθε κλάση (εάν είναι μετρήσιμη σε αυτό το επίπεδο), η συνολική τιμή της (στο πρόγραμμα), η μικρότερη τιμή της, η μεγαλύτερη τιμή της και η τυπική απόκλιση της (standard deviation). Επίσης, όπως αναφέραμε και προηγουμένως, το εργαλείο δημιουργεί το δυναμικό γράφο με τις εξαρτήσεις των κλάσεων και των πακέτων του έργου.

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per	384	0,632	1,205	12	/BlackWindowSpider/src/org/archive/crawler/extract...	
Number of Attributes (avg/max per type)	1228	2,02	4,059	37	/BlackWindowSpider/src/org/archive/crawler/frame...	
Number of Children (avg/max per type)	324	0,533	2,177	26	/BlackWindowSpider/src/org/archive/util/TmpDirTes...	
Number of Classes (avg/max per packageFragr	608	12,408	15,047	66	/BlackWindowSpider/src/org/archive/util	
Method Lines of Code (avg/max per method)	44309	8,225	13,717	193	/BlackWindowSpider/src/org/archive/crawler/datam...	CrawlOrder
Number of Methods (avg/max per type)	4841	7,962	12,275	103	/BlackWindowSpider/src/org/archive/crawler/frame...	
Nested Block Depth (avg/max per method)		1,589	0,992	9	/BlackWindowSpider/src/org/apache/commons/poo...	borrowObject
src		1,589	0,992	9	/BlackWindowSpider/src/org/apache/commons/poo...	borrowObject
org.apache.commons.pool.impl		1,5	1,394	9	/BlackWindowSpider/src/org/apache/commons/poo...	borrowObject
org.archive.crawler.frontier		1,78	1,185	8	/BlackWindowSpider/src/org/archive/crawler/frontier...	importQueuesFromLog
RecoveryJournal.java		2,333	2,392	8	/BlackWindowSpider/src/org/archive/crawler/frontier...	importQueuesFromLog
RecoveryJournal		2,333	2,392	8	/BlackWindowSpider/src/org/archive/crawler/frontier...	importQueuesFromLog
importQueuesFromLog	8					
importCompletionInfoFroml	7					
readLine	3					
importRecoverLog	2					
RecoveryJournal	1					
added	1					
writeLongUriLine	1					
finishedSuccess	1					
emitted	1					
finishedDisregard	1					
finishedFailure	1					
rescheduled	1					
WorkQueueFrontier.java		1,811	1,467	7	/BlackWindowSpider/src/org/archive/crawler/frontier...	next
AbstractFrontier.java		1,7	0,988	6	/BlackWindowSpider/src/org/archive/crawler/frontier...	enforceBandwidthThrottle
BdbMultipleWorkQueues.java		2,217	1,214	6	/BlackWindowSpider/src/org/archive/crawler/frontier...	getFrom

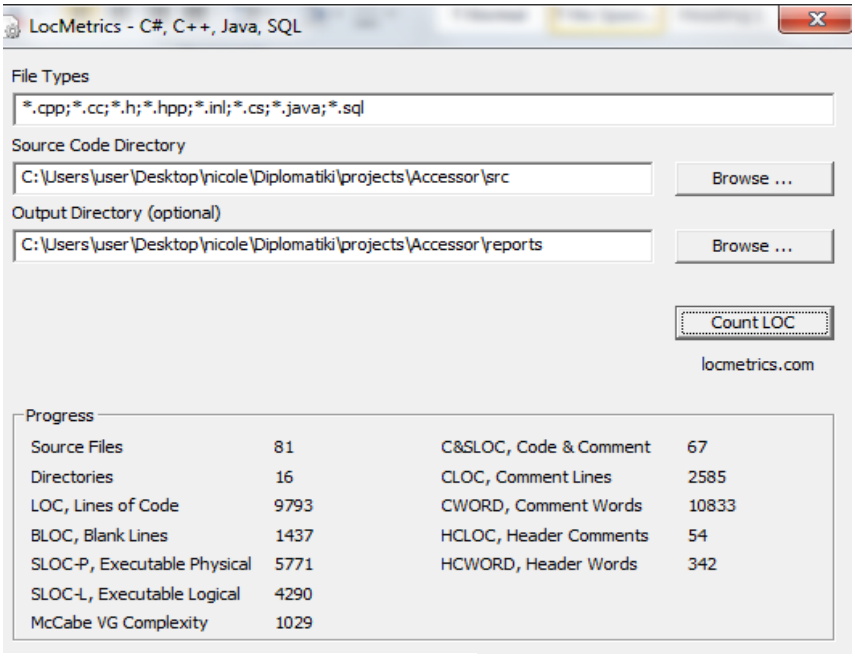
Σχήμα 3.1 : Εργαλείο Metrics 1.3.6

3.3.2 LocMetrics

Το εργαλείο LocMetrics[2] είναι ένα δωρεάν εργαλείο στατικής ανάλυσης κώδικα το οποίο μπορεί να τρέξει και από την γραμμή εντολών. Παρέχει μια εύκολη και απλή διεπαφή και υπολογίζει 10 μετρικές σχετικές με τις γραμμές του κώδικα σε πηγαίο κώδικα στις γλώσσες C#, Java, C++, Sql.
Οι μετρικές που υπολογίζονται είναι οι εξής:

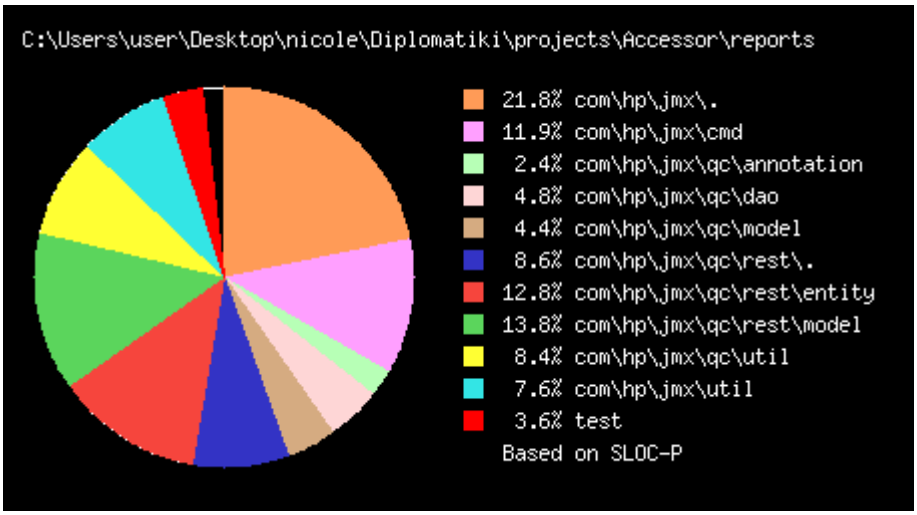
1. Total lines of code (LOC): Συνολικές γραμμές κώδικα
2. Blank lines of code (BLOC): Κενές γραμμές κώδικα
3. Comment lines of code (CLOC): Σχολιασμένες γραμμές κώδικα
4. Lines with both code and comments (C&SLOC): Γραμμές με κώδικα και σχόλια
5. Logical source lines of code (SLOC-L): Λογικές γραμμές κώδικα
6. McCabe VG complexity (CC): Κυκλωματική πολυπλοκότητα
7. Number of comment words (CWORDS): Αριθμός σχόλιων λέξεων
8. Physical executable source lines of code (SLOC-P): Εκτελέσιμες γραμμές κώδικα (οι συνολικές γραμμές κώδικα εκτός των κενών γραμμών και των σχολίων).
9. Header Comment Lines of Code (HCLOC): Σχολιασμένες γραμμές στην επικεφαλίδα
10. Header Commentary Words (HCWORD): Σχολιασμένες λέξεις στην επικεφαλίδα

[2] www.locmetrics.com



Σχήμα 3.2: Εργαλείο LocMetrics

Επίσης ο χρήστης μπορεί να εξάγει αναφορά σε μορφή html ή csv μορφή με τις λεπτομέρειες της ανάλυσης καθώς και ένα διάγραμμα βασισμένο στην μετρική SLOC-P.



Σχήμα 3.3:Εργαλείο LocMetrics, Pie Chart

3.3.3 SourceMonitor

Το εργαλείο SourceMonitor[3] είναι ένα δωρεάν λογισμικό που αναπτύχθηκε από την εταιρεία Campwood Software και συγκεκριμένα από τον Jim Wanner το 1997.
Το εργαλείο αναλύει στατικά λέξη-λέξη του κώδικα στις γλώσσες C++, C, C#, VB.NET, Java, Delphi, Visual Basic (VB6), HTML.
Οι μετρικές που υπολογίζονται είναι οι εξής:

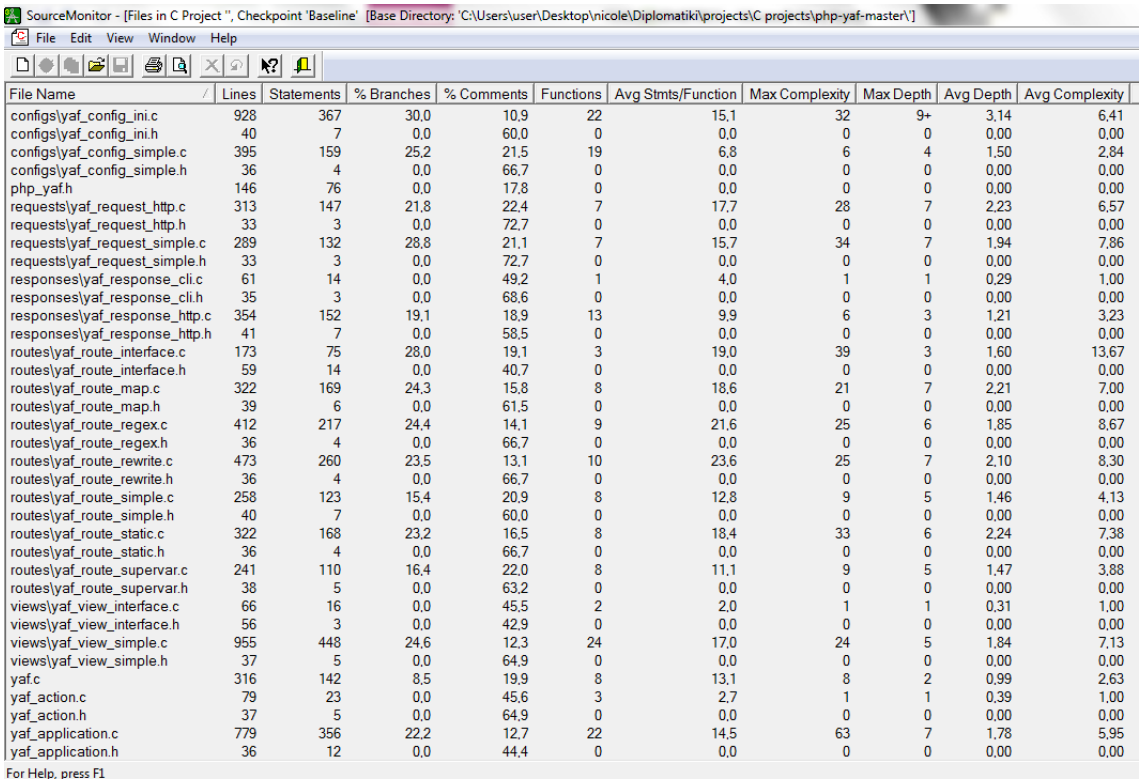
1. Number of Lines (LOC): ο αριθμός των φυσικών γραμμών σ' ένα αρχείο.
2. Percent comments: Οι γραμμές που περιλαμβάνουν σχόλια με βάση τον συνολικό αριθμό γραμμών του κώδικα.
3. Percent branches: Το ποσοστό εντολών διακλάδωσης.
4. Average methods per class: Ο συνολικός μέσος όρος όλων των μεθόδων των κλάσεων ενός αρχείου ή ενός checkpoint. Το πηλίκο του συνολικού αριθμού μεθόδων δια το συνολικό αριθμό για τις οποίες βρίσκονται οι εφαρμογές μεθόδου.
5. Average statements per method: Ο μέσος όρος εντολών ανά μέθοδο.
6. Number of functions: Ο αριθμός των συναρτήσεων.
7. Max Complexity: Η μέγιστη πολυπλοκότητα όλων των μεθόδων ή των συναρτήσεων μιας ρουτίνας.
8. Max depth: Το μέγιστο βάθος όλων των μεθόδων.
9. Average block depth: Το μέσο βάθος μίας ρουτίνας, δηλαδή το μέσο όρο των βαθών όλων των κλάσεων της ρουτίνας.
10. Average Complexity: Ο μέσος όρος της πολυπλοκότητας όλων των μεθόδων ή των συναρτήσεων μιας ρουτίνας.

Οι μετρικές για όλα τα αρχεία ενός checkpoint εμφανίζονται σε έναν πίνακα (σχήμα 3.4). Με τον όρο checkpoint αναφερόμαστε σε μια συλλογή από αρχεία με τις μετρικές για όλα τα αρχεία που έχουν επιλεγεί κατά την δημιουργία του checkpoint. Ένας δεύτερος πίνακας παρουσιάζει τις τιμές των μετρικών για κάθε αρχείο ενός checkpoint. Επίσης, υπάρχει ένας πίνακας που παρουσιάζει τις μετρικές για όλες τις μεθόδους ενός checkpoint ή ενός επιλεγμένου υποσυνόλου αρχείων ενός checkpoint.. Τέλος, ένας

[3] www.campwoodsw.com

λεπτομερής πίνακας περιέχει όλα τα δεδομένα που έχουν συλλεχθεί από ένα checkpoint ή ένα αρχείο. Ακόμη, ο χρήστης μπορεί να εξάγει αναφορά σε xml και csv μορφή, με τις λεπτομέρειες της ανάλυσης.

Επίσης υπάρχει η δυνατότητα αποθήκευσης του έργου σε αρχείο (project file).


SourceMonitor - [Files in C Project '', Checkpoint 'Baseline' [Base Directory: 'C:\Users\user\Desktop\nicole\Diplomatiki\projects\C projects\php-yaf-master\']

File Edit View Window Help

File Name	Lines	Statements	% Branches	% Comments	Functions	Avg Stmts/Function	Max Complexity	Max Depth	Avg Depth	Avg Complexity
configs\yaf_config_ini.c	928	367	30,0	10,9	22	15,1	32	9+	3,14	6,41
configs\yaf_config_ini.h	40	7	0,0	60,0	0	0,0	0	0	0,00	0,00
configs\yaf_config_simple.c	395	159	25,2	21,5	19	6,8	6	4	1,50	2,84
configs\yaf_config_simple.h	36	4	0,0	66,7	0	0,0	0	0	0,00	0,00
php_yaf.h	146	76	0,0	17,8	0	0,0	0	0	0,00	0,00
requests\yaf_request_http.c	313	147	21,8	22,4	7	17,7	28	7	2,23	6,57
requests\yaf_request_http.h	33	3	0,0	72,7	0	0,0	0	0	0,00	0,00
requests\yaf_request_simple.c	289	132	28,8	21,1	7	15,7	34	7	1,94	7,86
requests\yaf_request_simple.h	33	3	0,0	72,7	0	0,0	0	0	0,00	0,00
responses\yaf_response_cli.c	61	14	0,0	49,2	1	4,0	1	1	0,29	1,00
responses\yaf_response_cli.h	35	3	0,0	68,6	0	0,0	0	0	0,00	0,00
responses\yaf_response_http.c	354	152	19,1	18,9	13	9,9	6	3	1,21	3,23
responses\yaf_response_http.h	41	7	0,0	58,5	0	0,0	0	0	0,00	0,00
routes\yaf_route_interface.c	173	75	28,0	19,1	3	19,0	39	3	1,60	13,67
routes\yaf_route_interface.h	59	14	0,0	40,7	0	0,0	0	0	0,00	0,00
routes\yaf_route_map.c	322	169	24,3	15,8	8	18,6	21	7	2,21	7,00
routes\yaf_route_map.h	39	6	0,0	61,5	0	0,0	0	0	0,00	0,00
routes\yaf_route_regex.c	412	217	24,4	14,1	9	21,6	25	6	1,85	8,67
routes\yaf_route_regex.h	36	4	0,0	66,7	0	0,0	0	0	0,00	0,00
routes\yaf_route_rewrite.c	473	260	23,5	13,1	10	23,6	25	7	2,10	8,30
routes\yaf_route_rewrite.h	36	4	0,0	66,7	0	0,0	0	0	0,00	0,00
routes\yaf_route_simple.c	258	123	15,4	20,9	8	12,8	9	5	1,46	4,13
routes\yaf_route_simple.h	40	7	0,0	60,0	0	0,0	0	0	0,00	0,00
routes\yaf_route_static.c	322	168	23,2	16,5	8	18,4	33	6	2,24	7,38
routes\yaf_route_static.h	36	4	0,0	66,7	0	0,0	0	0	0,00	0,00
routes\yaf_route_supervar.c	241	110	16,4	22,0	8	11,1	9	5	1,47	3,88
routes\yaf_route_supervar.h	38	5	0,0	63,2	0	0,0	0	0	0,00	0,00
views\yaf_view_interface.c	66	16	0,0	45,5	2	2,0	1	1	0,31	1,00
views\yaf_view_interface.h	56	3	0,0	42,9	0	0,0	0	0	0,00	0,00
views\yaf_view_simple.c	955	448	24,6	12,3	24	17,0	24	5	1,84	7,13
views\yaf_view_simple.h	37	5	0,0	64,9	0	0,0	0	0	0,00	0,00
yaf.c	316	142	8,5	19,9	8	13,1	8	2	0,99	2,63
yaf_action.c	79	23	0,0	45,6	3	2,7	1	1	0,39	1,00
yaf_action.h	37	5	0,0	64,9	0	0,0	0	0	0,00	0,00
yaf_application.c	779	356	22,2	12,7	22	14,5	63	7	1,78	5,95
yaf_application.h	36	12	0,0	44,4	0	0,0	0	0	0,00	0,00

For Help, press F1

Σχήμα 3.4 : Εργαλείο SourceMonitor

3.3.4 PMD

Το PMD[4] είναι ένα λογισμικό ανοικτού κώδικα για κώδικα γραμμένο σε Java το οποίο διατίθεται ως plugin για τα περιβάλλοντα προγραμματισμού Eclipse, Maven, NetBeans, JBuilder, JDeveloper, IntelliJ IDEA. Επίσης μπορεί να τρέξει και από την γραμμή εντολών. Αναπτύχθηκε από τον Tom Copeland και είναι ένα δημοφιλής εργαλείο στην Java κοινότητα. Στην παρούσα διπλωματική χρησιμοποιήσαμε το plugin για το περιβάλλον προγραμματισμού Eclipse με έκδοση 5.0.5 ούτως ώστε να μπορούμε να κάνουμε μια πιο ολοκληρωμένη σύγκριση με τα υπόλοιπα εργαλεία που είναι plugin στο Eclipse.

Το PMD αναλύει στατικά πηγαίο κώδικα σε Java με βάση ενός συνόλου κανόνων με σκοπό να ανακαλυφτούν κοινά προγραμματιστικά λάθη (violations) που αναφέρουμε

[4] www.pmd.sourceforge.net

πιο κάτω. Αφού ενεργοποιηθεί η ανάλυση, υπολογίζονται αυτόματα οι ενεργοποιημένοι κανόνες και εμφανίζονται οι παραβιάσεις στην όψη (view) Violations Overview (σχήμα 3.5) και στην όψη Violations outline (σχήμα 3.6). Τα λάθη επίσης επισημαίνονται και στον κώδικα στο Eclipse. Επίσης το plugin διαθέτει και δικό του Perspective για την καλύτερη επισκόπηση των αποτελεσμάτων της ανάλυσης.

Το PMD αναλύει τον κώδικα με βάση 25 σύνολα (set) από κανόνες. Συνολικά οι κανόνες είναι 260 και εντοπίζουν σημαντικά προβλήματα όπως: προβλήματα απόδοσης (‘νεκρός’ κώδικας κλπ), υπερβολικά σύνθετες εκφράσεις, προβλήματα που αφορούν καλές πρακτικές (κανόνες σχεδιασμού, κανόνες ονομασίας κλπ), προβλήματα που αφορούν την ορθότητα του κώδικα όπως η κλωνοποίηση του κώδικα κλπ. Πιο κάτω παρουσιάζονται τα 25 σύνολα κανόνων:

1. Android: Κανόνες (3) που σχετίζονται με το εργαλείο Android SDK [3] .
2. Basic: Κανόνες (23) που αφορούν καλές πρακτικές συγγραφής κώδικα όπως αποφυγή κενών catch μπλοκς (blocks), κενών if εντολών, αχρείαστων return εντολών, κλπ.
3. Braces: Κανόνες (4) που αφορούν παραβιάσεις σχετικές με τη μη χρησιμοποίηση curly braces ({}) στις εντολές διακλάδωσης.
4. Clone Implementation: Κανόνες (3) για τον εντοπισμό λανθασμένων χρήσεων της μεθόδου clone() όπως για παράδειγμα όταν η μέθοδος δεν κάνει throw το exception CloneNotSupportedException ή δεν υλοποιεί (implement) την διεπαφή Cloneable.
5. Code Size: Κανόνες (11) που εντοπίζουν παραβιάσεις σχετικές με το μέγεθος του κώδικα ή την πολυπλοκότητα του. Το σύνολο αυτό εφαρμόζει μετρικές όπως η κυκλωματική πολυπλοκότητα (CC), ο αριθμός των δημόσιων (public) μεθόδων (NPM), ο αριθμός των μεθόδων (NOM), ο αριθμός των παραμέτρων (NOP), το μέγεθος της κλάσης, ο αριθμός γραμμών των κατασκευαστών (constructor) , ο αριθμός γραμμών των μεθόδων (MLOC).
6. Comments: Κανόνες (3) που εντοπίζουν παραβιάσεις σχετιζόμενες με τα σχόλια του κώδικα όπως ο αριθμός γραμμών των σχολίων (CLOC).
7. Controversial: Κανόνες (23) που εντοπίζουν αμφιλεγόμενες παραβιάσεις όπως κάθε κλάση πρέπει να δηλώνει τουλάχιστον έναν κατασκευαστή (constructor), κάθε μέθοδος που δεν είναι void πρέπει να επιστρέφει μόνο μια τιμή (return) κλπ.

8. Coupling: Κανόνες (5) σχετικοί με υψηλές συζεύξεις (coupling) μεταξύ αντικειμένων και πακέτων όπως η μετρική CBO.
9. Design: Κανόνες (54) σχετικοί με τον σχεδιασμό του κώδικα. Για παράδειγμα δήλωση αφηρημένης (abstract) κλάσης χωρίς να υπάρχει αφηρημένη (abstract) μέθοδος, κλάσεις με μόνο ιδιωτικούς (private) κατασκευαστές να δηλώνονται final, κλάσεις με τον όρο 'god' που είναι αρκετά μεγάλες και πολύπλοκες πρέπει να διασπαστούν κλπ.
10. Empty Code: Κανόνες (11) που αφορούν κενά μπλοκς (blocks), κενές εντολές διακλάδωσης, κενά try μπλοκς (blocks).
11. Finalizer: Κανόνες (6) που αφορούν την μέθοδο finalize().
12. Import Statements: Κανόνες (6) που αφορούν την εισαγωγή import εντολών. Για παράδειγμα πρέπει να αποφεύγεται η δήλωση 2 φορών του ίδιου import.
13. J2EE: Κανόνες (9) σχετικοί με την J2EE αρχιτεκτονική.
14. JavaBeans: Κανόνες (2) σχετικοί με το μοντέλο JavaBeans.
15. Junit: Κανόνες (12) για την βιβλιοθήκη Junit. Για παράδειγμα οι κλάσεις που περιέχουν σενάρια ελέγχου (test cases) πρέπει να τελειώνουν με Test.
16. Jakarta Commons Logging: Κανόνες (4) σχετικοί με το framework Jakarta Commons Logging.
17. Java Logging: Κανόνες (5) για το Java Logging framework. Για παράδειγμα οι logger μεταβλητές πρέπει να δηλώνονται στατικές (static) και final.
18. Migration: Κανόνες (14) σχετικοί με περιπτώσεις μετακίνησης από ένα JDK (Java Development Kid) σε άλλο.
19. Naming: Κανόνες (20) σχετικοί με το πως πρέπει να ονομάζονται οι κλάσεις, οι μέθοδοι, οι μεταβλητές, τα πακέτα σύμφωνα με το PMD.
20. Optimization: Κανόνες (12) για την βελτιστοποίηση του κώδικα βασισμένοι στις καλές πρακτικές. Για παράδειγμα μια μεταβλητή που της ανατίθεται τιμή μόνο μια φορά θα μπορούσε να είναι final.
21. Strict Exceptions: Κανόνες (12) σχετικοί με το throw και catch exceptions. Για παράδειγμα αποφυγή του catch του exception Throwable.
22. String and StringBuffer: Κανόνες (16) σχετικοί με τον τύπο String.
23. Security Code Guidelines: Κανόνες (2) σχετικοί με την σωστή αποθήκευση πινάκων.

24. Unnecessary: Κανόνες (7) για αχρείαστο κώδικα. Για παράδειγμα αχρείαστες παρενθέσεις θα πρέπει να αφαιρούνται..
25. Unused Code: Κανόνες (5) για κώδικα που δεν χρησιμοποιείται.

Οι παραβάσεις που εντοπίζονται κατηγοριοποιούνται σε 5 επίπεδα προτεραιότητας ανάλογα με την σοβαρότητα της παραβίασης:

Προτεραιότητα 1: Blocker (high error)

Προτεραιότητα 2: Critical (medium high error)

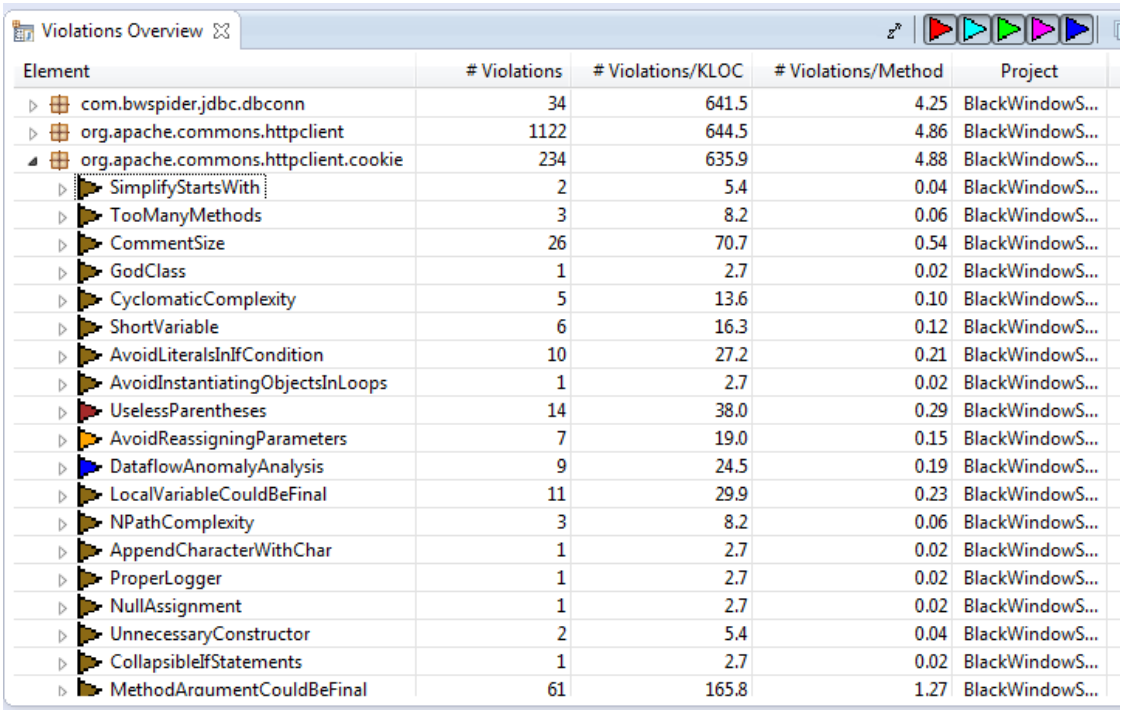
Προτεραιότητα 3: Urgent (medium error)

Προτεραιότητα 4: Important (medium low error)

Προτεραιότητα 5: Warning (low error)

Παράλληλα ο χρήστης έχει την δυνατότητα να καθορίσει ο ίδιος την προτεραιότητα για κάθε κανόνα.

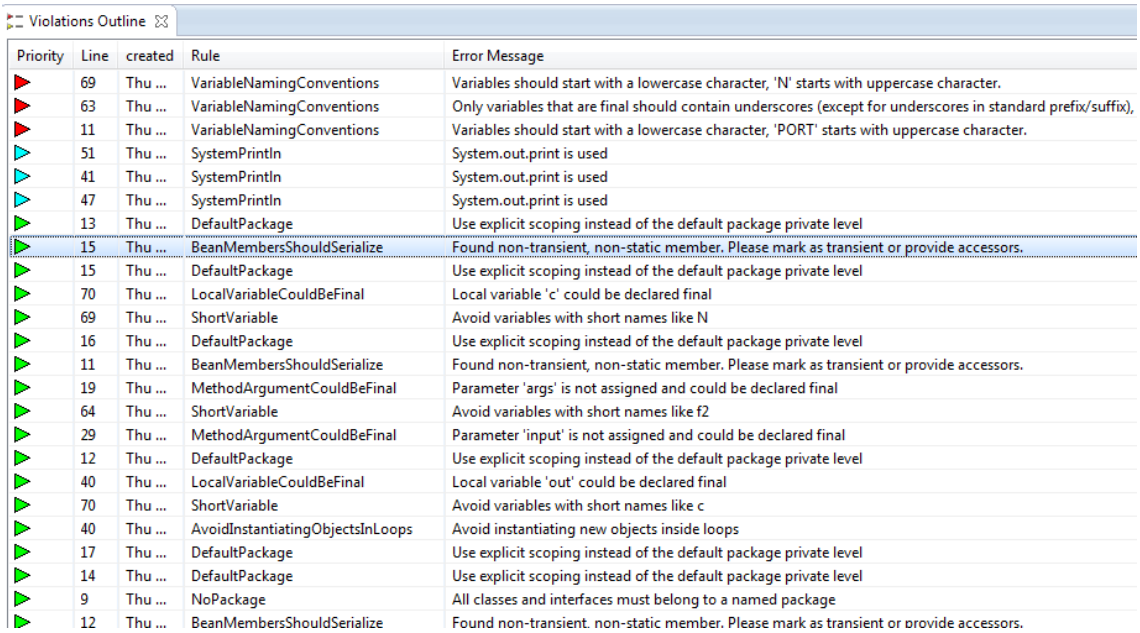
Για κάθε στοιχείο (element) στην όψη Violations Overview όπως φαίνεται και στο σχήμα 3.5 για ένα παράδειγμα της εκτέλεσης της ανάλυσης, υπολογίζεται το πλήθος των παραβιάσεων, ο αριθμός των παραβάσεων ανά τις γραμμές του κώδικα (LOC) και ο αριθμός των παραβάσεων ανά τις μεθόδους (NOM).



Element	# Violations	# Violations/KLOC	# Violations/Method	Project
com.bwspider.jdbc.dbconn	34	641.5	4.25	BlackWindowS...
org.apache.commons.httpclient	1122	644.5	4.86	BlackWindowS...
org.apache.commons.httpclient.cookie	234	635.9	4.88	BlackWindowS...
SimplifyStartsWith	2	5.4	0.04	BlackWindowS...
TooManyMethods	3	8.2	0.06	BlackWindowS...
CommentSize	26	70.7	0.54	BlackWindowS...
GodClass	1	2.7	0.02	BlackWindowS...
CyclomaticComplexity	5	13.6	0.10	BlackWindowS...
ShortVariable	6	16.3	0.12	BlackWindowS...
AvoidLiteralsInIfCondition	10	27.2	0.21	BlackWindowS...
AvoidInstantiatingObjectsInLoops	1	2.7	0.02	BlackWindowS...
UselessParentheses	14	38.0	0.29	BlackWindowS...
AvoidReassigningParameters	7	19.0	0.15	BlackWindowS...
DataflowAnomalyAnalysis	9	24.5	0.19	BlackWindowS...
LocalVariableCouldBeFinal	11	29.9	0.23	BlackWindowS...
NPathComplexity	3	8.2	0.06	BlackWindowS...
AppendCharacterWithChar	1	2.7	0.02	BlackWindowS...
ProperLogger	1	2.7	0.02	BlackWindowS...
NullAssignment	1	2.7	0.02	BlackWindowS...
UnnecessaryConstructor	2	5.4	0.04	BlackWindowS...
CollapsibleIfStatements	1	2.7	0.02	BlackWindowS...
MethodArgumentCouldBeFinal	61	165.8	1.27	BlackWindowS...

Σχήμα 3.5 : Εργαλείο PMD, Violations Overview

Η όψη Violations Overview όπως φαίνεται και στο σχήμα 3.6 δείχνει το βαθμό προτεραιότητας της παραβίασης, την γραμμή που εντοπίστηκε την ημερομηνία που έγινε η ανάλυση, το όνομα του κανόνα της παραβίασης και ένα μήνυμα (error message) για το λάθος που εντοπίστηκε.


Violations Outline

Priority	Line	created	Rule	Error Message
▶	69	Thu ...	VariableNamingConventions	Variables should start with a lowercase character, 'N' starts with uppercase character.
▶	63	Thu ...	VariableNamingConventions	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix),
▶	11	Thu ...	VariableNamingConventions	Variables should start with a lowercase character, 'PORT' starts with uppercase character.
▶	51	Thu ...	SystemPrintln	System.out.print is used
▶	41	Thu ...	SystemPrintln	System.out.print is used
▶	47	Thu ...	SystemPrintln	System.out.print is used
▶	13	Thu ...	DefaultPackage	Use explicit scoping instead of the default package private level
▶	15	Thu ...	BeanMembersShouldSerialize	Found non-transient, non-static member. Please mark as transient or provide accessors.
▶	15	Thu ...	DefaultPackage	Use explicit scoping instead of the default package private level
▶	70	Thu ...	LocalVariableCouldBeFinal	Local variable 'c' could be declared final
▶	69	Thu ...	ShortVariable	Avoid variables with short names like N
▶	16	Thu ...	DefaultPackage	Use explicit scoping instead of the default package private level
▶	11	Thu ...	BeanMembersShouldSerialize	Found non-transient, non-static member. Please mark as transient or provide accessors.
▶	19	Thu ...	MethodArgumentCouldBeFinal	Parameter 'args' is not assigned and could be declared final
▶	64	Thu ...	ShortVariable	Avoid variables with short names like f2
▶	29	Thu ...	MethodArgumentCouldBeFinal	Parameter 'input' is not assigned and could be declared final
▶	12	Thu ...	DefaultPackage	Use explicit scoping instead of the default package private level
▶	40	Thu ...	LocalVariableCouldBeFinal	Local variable 'out' could be declared final
▶	70	Thu ...	ShortVariable	Avoid variables with short names like c
▶	40	Thu ...	AvoidInstantiatingObjectsInLoops	Avoid instantiating new objects inside loops
▶	17	Thu ...	DefaultPackage	Use explicit scoping instead of the default package private level
▶	14	Thu ...	DefaultPackage	Use explicit scoping instead of the default package private level
▶	9	Thu ...	NoPackage	All classes and interfaces must belong to a named package
▶	12	Thu ...	BeanMembersShouldSerialize	Found non-transient, non-static member. Please mark as transient or provide accessors.

Σχήμα 3.6 : Εργαλείο PMD, Violations Outline

Επίσης το PMD δίνει τη δυνατότητα οι υψηλής προτεραιότητας παραβιάσεις να θεωρούνται σφάλματα του Eclipse, να απαιτείται δηλαδή η διόρθωση τους για να εκτελεστεί το πρόγραμμα., τον καθορισμό working set δηλαδή έργα που ακολουθούν τις ρυθμίσεις που καθορίζονται καθώς επίσης και την δυνατότητα ο χρήστης να γράψει τους δικούς του κανόνες χρησιμοποιώντας Xpath ή Java κλάσεις.

Επιπλέον, το PMD παρέχει τη δυνατότητα εντοπισμού κλώνων κώδικα με το Copy/Paste Detector που εφαρμόζει τον Karp-Rabin [20] αλγόριθμο, κάτι που βέβαια ξεφεύγει από την στατική ανάλυση κώδικα.

Ο χρήστης πέρα από την ανάλυση, στην οποία μπορεί να επιλέξει ποιες παραβιάσεις να ενεργοποιηθούν, μπορεί να εξάγει αναφορές (reports) με τις λεπτομέρειες της ανάλυσης στις μορφές html,xml, csv.

3.3.5 Checkstyle

Το εργαλείο Checkstyle[5] είναι ένα λογισμικό ανοικτού κώδικα το οποίο διατίθεται ως plugin για τα περιβάλλοντα προγραμματισμού Eclipse, Maven, NetBeans, IntelliJ IDEA κλπ. Αναπτύχθηκε από τον Oliver Burn [6] το 2001.

Στην παρούσα διπλωματική χρησιμοποιήσαμε το plugin για το eclipse περιβάλλον με έκδοση 5.7 ούτως ώστε να μπορούμε να κάνουμε μια πιο ολοκληρωμένη σύγκριση με τα υπόλοιπα εργαλεία που είναι plugin στο Eclipse.

Το Checkstyle αναλύει στατικά πηγαίο κώδικα σε Java με βάση ενός σύνολου κανόνων με σκοπό να ανακαλυφτούν κοινά προγραμματιστικά λάθη (violations) που αναφέρουμε πιο κάτω. Αφού ενεργοποιηθεί η ανάλυση οι παραβιάσεις εμφανίζονται στην όψη (view) Checkstyle Violations (σχήμα 3.7). Τα λάθη επίσης επισημαίνονται και στον κώδικα στο eclipse.

Το Checkstyle αναλύει τον κώδικα με βάση 126 ελέγχους που κατηγοριοποιούνται σε 14 κατηγορίες (modules). Οι προκαθορισμένες (default) ρυθμίσεις δεν έχουν ενεργοποιημένους όλους τους ελέγχους. Ο χρήστης μπορεί να επιλέξει τους ελέγχους που επιθυμεί δημιουργώντας το δικό του configuration. Πιο κάτω παρουσιάζονται οι 14 κατηγορίες ελέγχου:

1. Annotations: Έλεγχοι (5) σχετικοί με annotations.
2. Block Checks: Έλεγχοι (5) σχετικοί με κενά μπλοκς (blocks).
3. Class design: Έλεγχοι (8) για την ορατότητα των μελών της κλάσης. Για παράδειγμα μόνο τα μέλη (μέθοδοι, μεταβλητές) που είναι static final πρέπει να είναι δημόσια (public) ή μια κλάση που έχει μόνο ένα ιδιωτικό (private) κατασκευαστή πρέπει να δηλωθεί final κλπ.
4. Coding: Έλεγχοι (43) σχετικοί με το πως πρέπει να συγγράφεται ο κώδικας. Για παράδειγμα, κάθε δήλωση switch πρέπει να δηλώνει και default πεδίο πάντα στο τέλος (μετά από όλα τα cases), κάθε μεταβλητή που η τιμής της δεν έχει αλλάξει ποτέ στο πρόγραμμα πρέπει να δηλώνεται final, δεν πρέπει να δηλώνονται ‘magic numbers’, δηλαδή μεταβλητές που έχουν μια ακέραια σταθερή τιμή θα πρέπει να δηλώνονται ως σταθερές, κάθε κλάση πρέπει να έχει κατασκευαστή, σε κάθε γραμμή πρέπει να είναι δηλωμένη μόνο μια γραμμή κλπ.
5. Duplicate code: Έλεγχος γραμμή γραμμή του κώδικα για διπλότυπο κώδικα.

[5] www.checkstyle.sourceforge.net

6. Headers: Έλεγχοι (2) σχετικά με τα headers. Για παράδειγμα κάθε αρχείο ξεκινά με ένα συγκεκριμένο Header.
7. Imports: Έλεγχοι (7) σχετικά με την εισαγωγή imports. Για παράδειγμα καμιά import εντολή δεν χρησιμοποιεί το σύμβολο *.
8. Javadoc comments: Έλεγχοι (6) σχετικά με τα Javadoc σχόλια. Για παράδειγμα κάθε πακέτο πρέπει να περιέχει Javadoc σχόλια.
9. Metrics: Έλεγχος για τις εξής μετρικές:

   α) Boolean Expression Complexity: Περιορισμός των αριθμών των &&, | |, &, | και ^ σε μια έκφραση.

   β) Class Data Abstraction Coupling: Η πολυπλοκότητα της σύζευξης που προκαλείται από Αφηρημένους Τύπους Δεδομένων (Abstract Data Types,ADTs).

   γ) Class Fan Out Complexity: Ο αριθμός των υπόλοιπων κλάσεων που εξαρτώνται από την συγκεκριμένη κλάση.

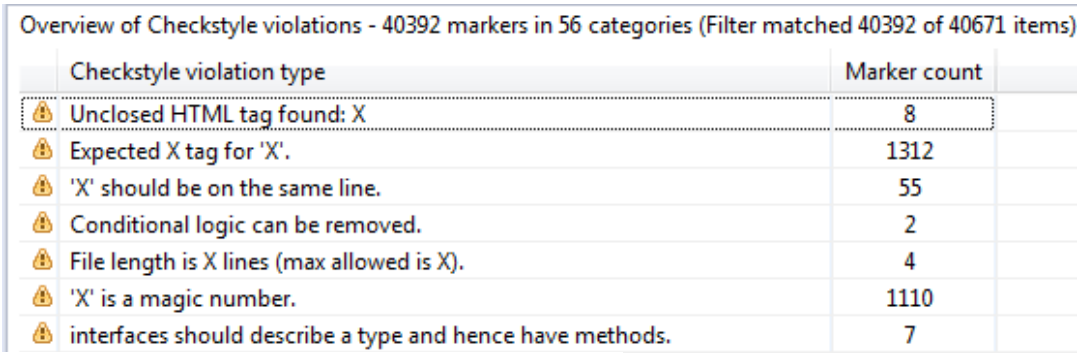
   δ) Cyclomatic Complexity (CC ή VG): Κυκλωματική Πολυπλοκότητα.

   ε) N path Complexity: Ο αριθμός των πιθανών μονοπατιών εκτέλεσης σε μια συνάρτηση (μέγιστη default τιμή=200).

   ζ) Non Commenting Source Statements (NCLOC): Ο αριθμός των μη σχολιασμένων γραμμών για κάθε μέθοδο (μέγιστη default τιμή=50), κλάση (μέγιστη default τιμή=1500) και αρχείο (μέγιστη default τιμή=2000).
10. Miscellaneous: Διάφοροι έλεγχοι (13) όπως αν το αρχείο τελειώνει με τον χαρακτήρα '\n' (καινούργια γραμμή), αν η δήλωση ενός πίνακα είναι με το στυλ όπως καθορίζει η Java, String[] args (true) ή το στυλ της C String args[] (false).
11. Modifiers: Έλεγχοι (2) για τους τύπους. Για παράδειγμα αν ακολουθούν την ιεραρχική δομή που καθορίζεται από το Java Language specification [18].
12. Naming Conventions: Έλεγχοι (12) για το αν το ονόματα των μεταβλητών, μεθόδων, παραμέτρων, κλάσεων, διεπαφών και πακέτων ακολουθούν μια συγκεκριμένη δομή. Για παράδειγμα τα ονόματα των μεταβλητών πρέπει να είναι στο όριο ^[a-z][a-zA-Z0-9]*$.
13. Regexp: Έλεγχοι (3) σχετικοί με τις κανονικές εκφράσεις. Για παράδειγμα έλεγχος για την ανίχνευση μιας γραμμής που ταιριάζει με μια συγκεκριμένη κανονική έκφραση (regular expression).

14. Size violations: Έλεγχοι (8) για μετρικές σχετικές με το μέγεθος. Για παράδειγμα το μέγεθος του αρχείου (μέγιστος αριθμός γραμμών, LOC default τιμή = 2000), το μέγεθος της γραμμής (μέγιστος αριθμός γραμμών, LOC default τιμή = 80), το μέγεθος της μεθόδου (μέγιστος αριθμός γραμμών, MLOC default τιμή = 150), ο αριθμός των παραμέτρων σε μια μέθοδο (μέγιστος αριθμός παραμέτρων, default τιμή = 7).
15. Whitespace: Έλεγχοι (12) για κενά διαστήματα. Για παράδειγμα δεν πρέπει να υπάρχει ο χαρακτήρας tab (‘\t’) στο αρχείο, πρέπει να υπάρχει κενό διάστημα μετά ή πριν από κάποιο σημείο (token).

Για κάθε παραβίαση (σχήμα 3.7) εμφανίζεται ο τύπος της παραβίασης και ο συνολικός αριθμός που εμφανίστηκε η παραβίαση. Επιλέγοντας την παραβίαση εμφανίζεται το αρχείο που βρέθηκε η παραβίαση, ο φάκελος, η γραμμή και το μήνυμα της παραβίασης.


Overview of Checkstyle violations - 40392 markers in 56 categories (Filter matched 40392 of 40671 items)

Checkstyle violation type	Marker count
Unclosed HTML tag found: X	8
Expected X tag for 'X'.	1312
'X' should be on the same line.	55
Conditional logic can be removed.	2
File length is X lines (max allowed is X).	4
'X' is a magic number.	1110
interfaces should describe a type and hence have methods.	7

Σχήμα 3.7 : Εργαλείο Checkstyle, Checkstyle

3.3.6 FindBugs

Το Findbugs[6] είναι ένα λογισμικό ανοικτού κώδικα το οποίο διατίθεται ως plugin για το περιβάλλον προγραμματισμού Eclipse και ως αυτόνομο εργαλείο με command line, ant και swing διεπαφές. Αναπτύχθηκε από William Pugh [28] το 2003 και τον David Hovermeyer [15] και φέρει την υπογραφή του πανεπιστημίου του Maryland. Έχει μεταφορτωθεί περισσότερες από 1 εκατομμύριο φορές και έχει κερδίσει ένα IBM Eclipse Innovation βραβείο [IBMEclipseInnovation]. Χορηγείται από τις εταιρείες SureLogic, Google και Sun Microsystems, ebay κλπ.

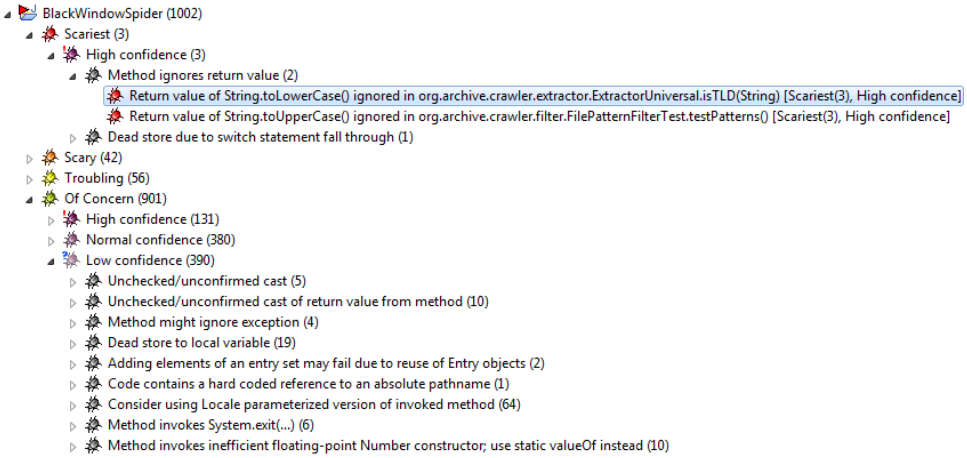
[6] www.findbugs.sourceforge.net

Στην παρούσα διπλωματική χρησιμοποιήσαμε το plugin για το περιβάλλον προγραμματισμού Eclipse με έκδοση 2.0.3 ούτως ώστε να μπορούμε να κάνουμε μια πιο ολοκληρωμένη σύγκριση με τα υπόλοιπα εργαλεία που είναι plugin στο Eclipse.
Το Findbugs αναλύει στατικά ενδιάμεσο (bytecode) κώδικα σε Java με βάση ενός συνόλου κανόνων με σκοπό να ανακαλυφτούν κοινά προγραμματιστικά λάθη (bugs). Αφού ενεργοποιηθεί η ανάλυση, υπολογίζονται οι ενεργοποιημένοι έλεγχοι (bug detectors) και εμφανίζονται τα λάθη στην όψη (view) Bug Explorer (σχήμα 3.8). Τα λάθη επισημαίνονται και στον κώδικα στο eclipse. Επίσης το plugin διαθέτει και δικό του Perspective για την καλύτερη επισκόπηση των αποτελεσμάτων της ανάλυσης.
Το FindBugs αναλύει τον κώδικα με βάση 128 μοτίβα (patterns types), δηλαδή σύνολα από κανόνες (bug detectors). Συνολικά οι κανόνες είναι 404 και κάθε κανόνας ανήκει σε μία από τις 9 κατηγορίες. Οι προκαθορισμένες (default) ρυθμίσεις δεν έχουν ενεργοποιημένους όλους τους κανόνες. Ο χρήστης μπορεί να επιλέξει τους κανόνες που επιθυμεί για ανάλυση. Πιο κάτω παρουσιάζονται οι 9 πιθανές κατηγορίες σφάλματος:

1. Bad practice: Έλεγχοι (83) για πρακτικές οι οποίες παραβιάζουν προτεινόμενες πρακτικές συγγραφής κώδικα. Για παράδειγμα, χρησιμοποίηση του συντελεστή '==' αντί της μεθόδου equals(), τα ονόματα των κλάσεων πρέπει να ξεκινούν με κεφαλαίο γράμμα κλπ.
2. Correctness: Έλεγχοι (144) για πιθανά λάθη κώδικα. Για παράδειγμα ατέρμονος κώδικας σε ένα loop, σύγκριση τοπικής μεταβλητής σε μία μέθοδο με τον εαυτό της κλπ.
3. Experimental: Έλεγχοι (3) σχετικοί με streams ή αντικείμενα βάσεων δεδομένων (database object). Για παράδειγμα, η μέθοδος μπορεί να αποτύχει να κλείσει (close) ένα stream, κλπ
4. Internationalization: Έλεγχοι (2) αν γίνεται χρήση non-local μεθόδων όπως για παράδειγμα χρήση της μεθόδου String.toUpperCase ή String.toLowerCase.
5. Malicious code vulnerability: Έλεγχοι (15) για περιπτώσεις που ο κώδικας χρησιμοποιείται λανθασμένα από άλλο κώδικα. Για παράδειγμα η μέθοδος finalize() πρέπει να είναι protected και όχι δημόσια (public), κλπ.
6. Multithreaded correctness: Έλεγχοι (45) σχετικά με τον συγχρονισμό νημάτων. Για παράδειγμα, χρησιμοποίηση της μεθόδου notify() αντί της μεθόδου notifyall() ή το κάλεσμα της μεθόδου wait() πρέπει να είναι μέσα σε loop, κλπ.

7. Performance: Έλεγχοι (28) για κώδικα που θα έπρεπε να συγγραφεί με διαφορετικό τρόπο ή να μην υπάρχει ούτως ώστε να βελτιωθεί η απόδοση του προγράμματος. Για παράδειγμα κώδικας που δεν χρησιμοποιείται ή δεν διαβάζεται ποτέ πρέπει να αφαιρείται, κλπ.
8. Security: Έλεγχοι (11) για κώδικα που μπορεί να προκαλέσει προβλήματα ασφαλείας. Για παράδειγμα σύνδεση σε βάση δεδομένων με κενό κωδικό, κλπ.
9. Dodgy code: Έλεγχοι (73) για ασαφή κώδικα που οδηγεί σε λάθη. Για παράδειγμα αναφορά σε κενή (null) τιμή, διπλή ανάθεση σε τοπική μεταβλητή, κλπ.

Για κάθε σφάλμα (σχήμα 3.10) εμφανίζεται ο συνολικός αριθμός που εμφανίστηκε το σφάλμα στο πρόγραμμα. Επιλέγοντας το σφάλμα εμφανίζονται γενικές πληροφορίες και επεξηγήσεις σχετικά με το σφάλμα στην όψη Bug Info.



Σχήμα 3.8 : Εργαλείο FindBugs, Bug Explorer

Πέρα από την ανάλυση στην οποία μπορεί να επιλέξει ποια σφάλματα να ενεργοποιηθούν, ο χρήστης μπορεί να καθορίσει αρχεία φίλτρων σε XML μορφή τα οποία μπορούν να χρησιμοποιηθούν για να συμπεριληφθούν ή να αποκλειστούν στην αναφορά σφάλματα που αφορούν συγκεκριμένες μεθόδους ή κλάσεις ή ακολουθούν συγκεκριμένα πρότυπα σφαλμάτων κλπ, μπορεί να επιλέξει τις κατηγορίες σφαλμάτων και την προτεραιότητα (confidence) των σφαλμάτων που θα εμφανίζονται και να ρυθμίσει το επίπεδο προσπάθειας της ανάλυσης (analysis effort) μεταξύ τριών τιμών (Default, Minimal Maximal). Μπορεί επίσης να εξάγει αναφορά με τις λεπτομέρειες της

ανάλυσης σε XML μορφή ή να κοινοποιήσει τα αποτελέσματα της ανάλυσης μέσω κάποιου cloud διακομιστή.

3.3.7 Ckjm

Το Ckjm[7] (Chidamber and Kemerer Java Metrics) είναι ένα δωρεάν εργαλείο το οποίο εκτελείται από τη γραμμή εντολών και διατίθεται επίσης ως plugin για το περιβάλλον προγραμματισμού Maven. Αναπτύχθηκε από τον Διομήδη Σπινέλλη [Σπινέλλη].
Στην παρούσα διπλωματική χρησιμοποιήσαμε το εργαλείο με έκδοση 1.9 από την γραμμή εντολών .
Το εργαλείο αναλύει στατικά ενδιάμεσο (bytecode) κώδικα σε Java και υπολογίζει τις αντικειμενοστρεφείς μετρικές Chidamber και Kemerer [7]. Οι μετρικές που υπολογίζονται είναι οι εξής:

Chidamber και Kemerer μετρικές:

1. Weighted methods per class (WMC)
2. Depth of Inheritance Tree (DIT)
3. Number of Children (NOC)
4. Coupling between object classes (CBO)
5. Response for a Class (RFC)
6. Lack of cohesion in methods (LCOM)

Υπολογίζει επίσης τις πιο κάτω μετρικές:

7. Afferent couplings (CA)
8. Number of public methods (NPM)

```
com.hp.jmx.BaseResultSetHandler 2 1 0 1 6 1 1 2
com.hp.jmx.AccessorFactory 6 1 0 4 28 15 1 5
com.hp.jmx.DBDump 2 1 0 0 2 1 4 2
com.hp.jmx.SQLServerAdminDBAccessor 11 0 0 2 31 55 1 10
com.hp.jmx.SQLServerDBAccessor 24 0 1 2 50 148 2 12
com.hp.jmx.DBDumpFactory 2 1 0 4 9 1 0 2
com.hp.jmx.OracleDBAccessor 29 0 0 3 58 274 5 16
com.hp.jmx.Configure 3 1 0 0 15 0 0 3
com.hp.jmx.BaseDBAccessor 31 1 2 3 62 463 2 12
com.hp.jmx.DPump 12 0 0 4 24 54 1 3
com.hp.jmx.BaseDBDump 14 1 2 4 46 85 2 3
com.hp.jmx.Test 2 1 0 3 8 1 0 2
com.hp.jmx.NormalDBDump 11 0 0 4 23 49 1 3
com.hp.jmx.DBAccessor 11 1 0 0 11 55 5 11
com.hp.jmx.ResultSetHandler 1 1 0 0 1 0 2 1
com.hp.jmx.BaseDBDump$StreamGobbler 2 2 0 0 12 0 1 1
```

Σχήμα 3.9 : Εργαλείο ckjm

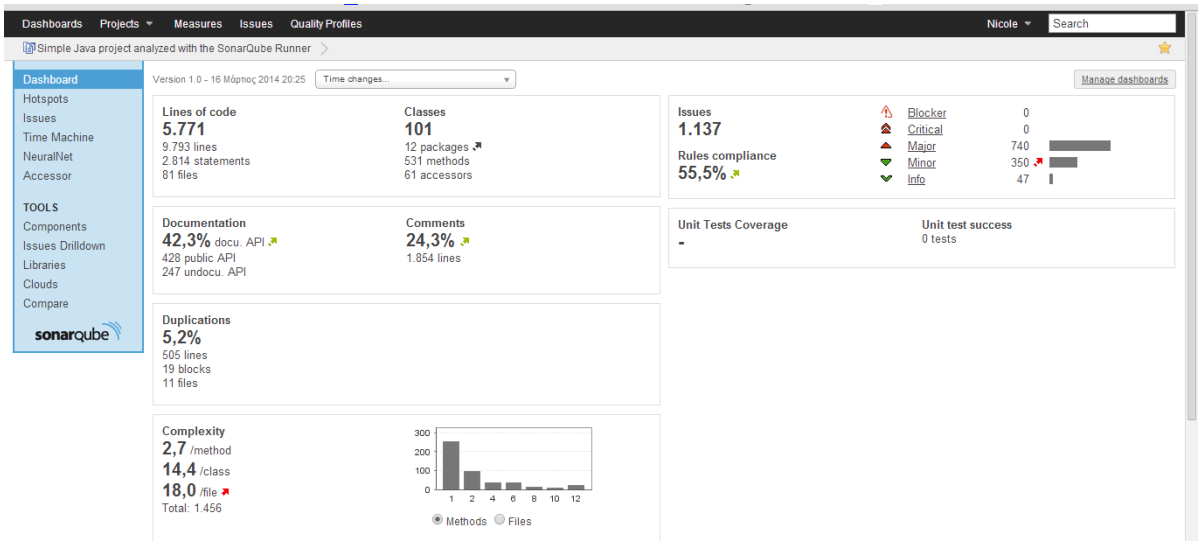
[7] www.spinellis.gr/sw/ckjm

3.3.8 Sonarqube

Το Sonarqube[8] (γνωστό και ως Sonar) είναι μια διαδικτυακή πλατφόρμα ανοικτού κώδικα και φέρει την υπογραφή της εταιρείας SonarSource [33]. Εκτελείται από την γραμμή εντολών ή μπορεί να ενσωματωθεί με τα περιβάλλοντα προγραμματισμού Eclipse, Maven, Ant, Gradle κ.α και επίσης μπορεί να ενσωματωθεί και με άλλα εργαλεία όπως τα JIRA, Mantis, LDAP, Fortify κλπ.

Στην παρούσα εργασία χρησιμοποιήσαμε το εργαλείο με έκδοση 3.7.4, από την γραμμή εντολών.

Το εργαλείο αναλύει στατικό ή ενδιάμεσο (bytecode) κώδικα σε περισσότερες από 25 γλώσσες (όχι όλες δωρεάν) μεταξύ των οποίων οι γλώσσες Java, C, C++, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL κλπ. Όσον αφορά την γλώσσα Java, το εργαλείο ενσωματώνει διάφορα άλλα εργαλεία (plugins) ανάμεσα στα οποία FindBugs, Pmd και Checkstyle σε ένα αυτόνομο εργαλείο. Η διαδικασία ανάλυσης είναι εύκολη αλλά σχετικά χρονοβόρα. Ο χρήστης ενεργοποιεί την βάση sonar (ή κάποια άλλη βάση όπως Oracle, MySql) και έπειτα εκτελεί το αρχείο sonar-runner.bat στο λογισμικό που επιθυμεί να αναλύσει. Τότε, ανατρέχει στο localhost:9000 όπου εμφανίζονται τα αποτελέσματα της ανάλυσης υπό μορφή dashboard (σχήμα 3.10). Το εργαλείο παρέχει αμέτρητες λειτουργίες.



Σχήμα 3.10 : Εργαλείο Sonarqube, Dashboard

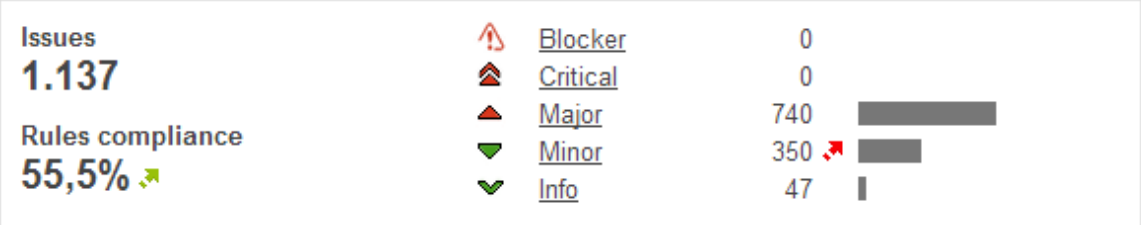
Το Sonarqube αναλύει τον κώδικα και αναζητεί για θέματα σχετικά με

[8] www.sonarqube.org

- διπλότυπο κώδικα (duplicate)
- πρότυπα κώδικα (coding standards)
- έλεγχο μονάδων (unit tests),
- κάλυψη μονοπατιών (code coverage)
- πολύπλοκο (complex) κώδικα
- πιθανά λάθη (potential bugs)
- σχόλια (comments)
- αρχιτεκτονική και σχεδιασμό (architecture and design)

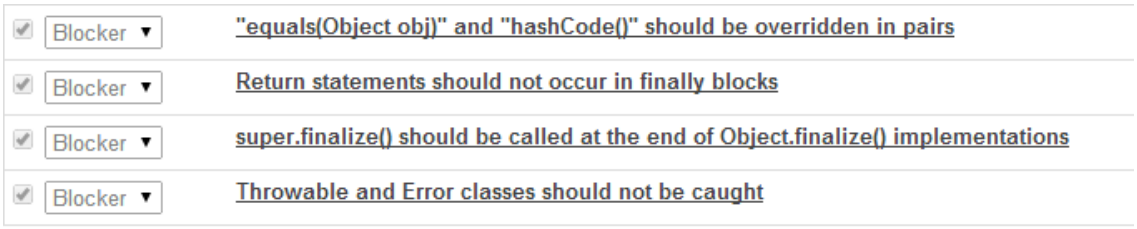
Potential bugs και Coding standards

Οι κατηγορίες αυτές είναι 2 ξεχωριστές κατηγορίες. Στην αναφορά όμως το Sonarqube τις εμφανίζει μαζί κάτω από το widget Issues (σχήμα 3.11). Έτσι στο widget Issues εμφανίζονται τα πιθανά λάθη της ανάλυσης (αναφορά σε κενό δείκτη κλπ) ή κανόνες ελέγχου βασισμένοι σε λάθη σύνταξης (οι εντολές διακλάδωσης πρέπει πάντα να δηλώνουν το σύμβολο '{}', κλπ) ή κανόνες ονομασίας.



Σχήμα 3.11: Εργαλείο Sonarqube, Issues widget

Οι κανόνες ελέγχου είναι περισσότεροι από 1000 (για την γλώσσα Java) και έτσι ο χρήστης έχει την επιλογή (αφού πρώτα συνδεθεί ως administrator) να ενεργοποιήσει ή να απενεργοποιήσει τους κανόνες ελέγχου που επιθυμεί. Επίσης κάποια αποτελέσματα (issues) είναι πιο κρίσιμα από άλλα και έτσι κατηγοριοποιούνται ανάλογα με την σοβαρότητα τους σε Blocker (σχήμα 3.12), Critical, Major , Minor, και Info. Επίσης ο χρήστης μπορεί να αλλάξει την κατηγορία κάθε κανόνα, να τον θέσει ως 'confirm' αν είναι σίγουρα λάθος ή ως 'favorite', ή μπορεί να τον θέσει ως 'false positive' (εάν θεωρεί πως είναι) και έτσι οι επόμενοι έλεγχοι να παραβλέψουν τον κανόνα αυτό και συνεπώς τα λανθασμένα αποτελέσματα (false positives).

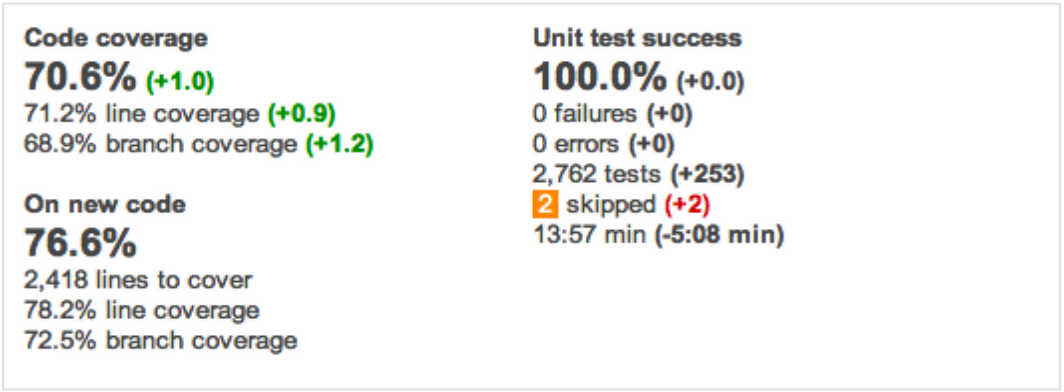


Σχήμα 3.12 : Εργαλείο Sonarqube, Παράδειγμα από Blocker issues

Unit tests

Πριν από κάθε κομμάτι κώδικα γράφεται πρώτα το αντίστοιχο κομμάτι ελέγχου μονάδας ή ενότητας (unit test). Ο κώδικας του ελέγχου μονάδας είναι ο ίδιος με τον παραγόμενο κώδικα, με την διαφορά της προσθήκης των εντολών ελέγχου. Οι έλεγχοι αυτού του τύπου είναι πολλαπλοί, μικρής εμβέλειας και συγκρίνουν τα αναμενόμενα αποτελέσματα με τα πραγματικά. Αποτελούν ένα ρητό μέρος του κώδικα, που διασφαλίζει και τεκμηριώνει τη σωστή λειτουργία του.

Το widget code coverage και unit tests (σχήμα 3.13) αναφέρεται το ποσοστό κάλυψης μονοπατιών και στο πόσο καλά ο κώδικας καλύπτεται (εάν καλύπτεται) από μονάδες ελέγχου (unit tests) και πόση επιτυχία έχουν οι έλεγχοι αυτοί.

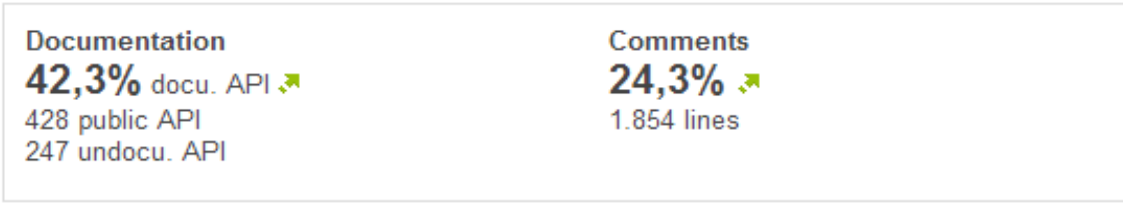


Σχήμα 3.13: Εργαλείο Sonarqube, Widget Code Coverage Unit Tests

Documentation and Comments

Το widget αυτό (σχήμα 3.14) αναπαριστά το ποσοστό των σχολίων στον κώδικα. Τα σχόλια σε ένα πρόγραμμα είναι απαραίτητα για την αναγνωσιμότητα και συντηρησιμότητα του κώδικα καθώς σχολιάζουν την λειτουργία του προγράμματος καθιστώντας την κατανοητή από οποιοδήποτε προγραμματιστή που θα αναλάβει το

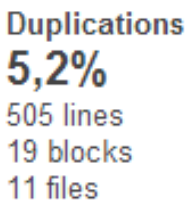
πρόγραμμα είτε για την περεταίρω ανάπτυξη του είτε για έλεγχο. Τα σχόλια χωρίζονται σε 2 κατηγορίες, εσωτερικά σχόλια (inline) ή εξωτερικά σχόλια (Api documentation). Τα εσωτερικά σχόλια είναι τα σχόλια που βρίσκονται μέσα σε μια οποιαδήποτε μέθοδο (public ή private) και αποσκοπούν στην επεξήγηση της λογικής του κώδικα. Τα εξωτερικά σχόλια είναι αυτά που βρίσκονται έξω από την μέθοδο και εξηγούν τι κάνει η μέθοδος (τις παραμέτρους, τις return τιμές κλπ).



Σχήμα 3.14: Εργαλείο Sonarqube, Widget Documentation and comments

Duplications

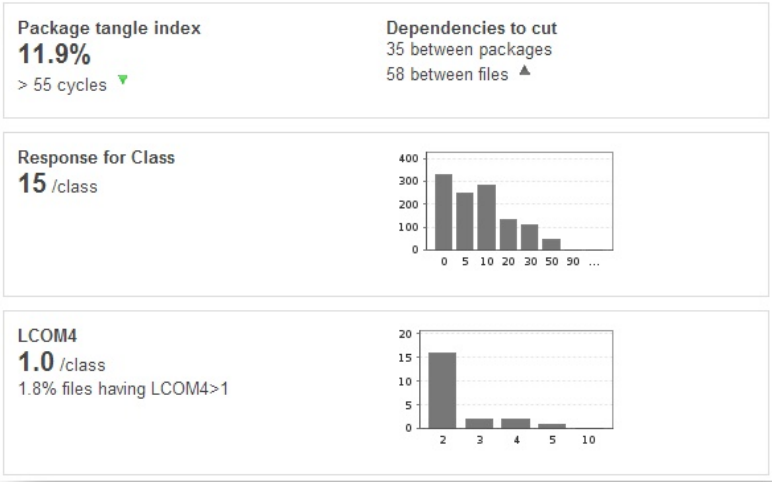
Ο όρος διπλότυπος κώδικας αναφέρεται στο ποσοστό του κώδικα που υπάρχει περισσότερος από μία φορά στον κώδικα. Το εργαλείο Sonarqube ‘μετρά’ το ποσοστό του διπλότυπου κώδικα καθώς μεγάλα ποσοστά αυξάνουν την πολυπλοκότητα του κώδικα.



Σχήμα 3.15 : Εργαλείο Sonarqube, Widget Duplications

Architecture and Design

Το Sonarqube κάνει έλεγχο για Spaghetti Design. Με τον όρο Spaghetti Design αναφερόμαστε στον κακό και πολύπλοκο σχεδιασμό στο αρχιτεκτονικό επίπεδο του έργου. Ελέγχει για εξαρτώμενους κύκλους (dependencies cycles) μεταξύ των πακέτων και των αρχείων και υπολογίζει επίσης την μετρική LCOM4 και Response for a class (RFC).

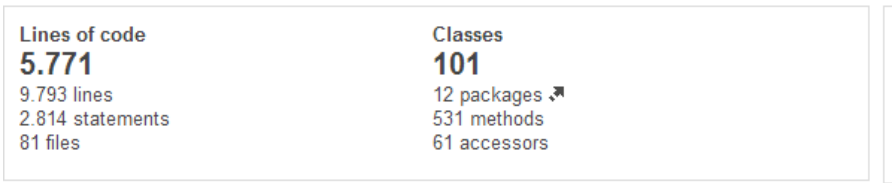


Σχήμα 3.16 : Εργαλείο Sonarqube, Widget Spaghetti Design

Υπολογίζονται επίσης διάφορες μετρικές μεγέθους (size).

Μετρική	**Περιγραφή**
Accessors	Ο αριθμός των getter και setter συναρτήσεων
Classes	Ο αριθμός των κλάσεων συμπεριλαμβανομένου των φωλιασμένων κλάσεων, διεπαφών, enums και annotations
Directories	Ο αριθμός των φακέλων
Files	Ο αριθμός των αρχείων
Lines	Ο αριθμός των γραμμών (LOC)
Lines of code	Ο αριθμός των γραμμών που περιέχουν τουλάχιστον ένα χαρακτήρα που είναι είτε κενός χαρακτήρας, tab ή μέρος σχόλιου (NLOC)
Methods	Ο αριθμός των μεθόδων (NOM)
Public Api	Public Api= #public κλάσεις + #public μέθοδοι + #public ιδιότητες
Statements	Ο αριθμός των δηλώσεων

Πίνακας 3.2: Εργαλείο Sonarqube, Size Metrics



Σχήμα 3.17 : Εργαλείο Sonarqube,Metrics Widget

Άλλες επιπρόσθετες λειτουργίες είναι οι εξής:

- Δημιουργία διάφορων φίλτρων εμφάνισης (διαγράμματα, widgets) και κοινοποίηση τους σε άλλους χρήστες.
- Τροποποίηση και κοινοποίηση dashboards, σε άλλους χρήστες.
- Ανάθεση ‘λαθών’ (issues) σε άλλους χρήστες.
- Αποθήκευση μετρικών και dashboards για ένα χρονικό διάστημα.
- Σύγκριση έργων.

3.3.9 Vil

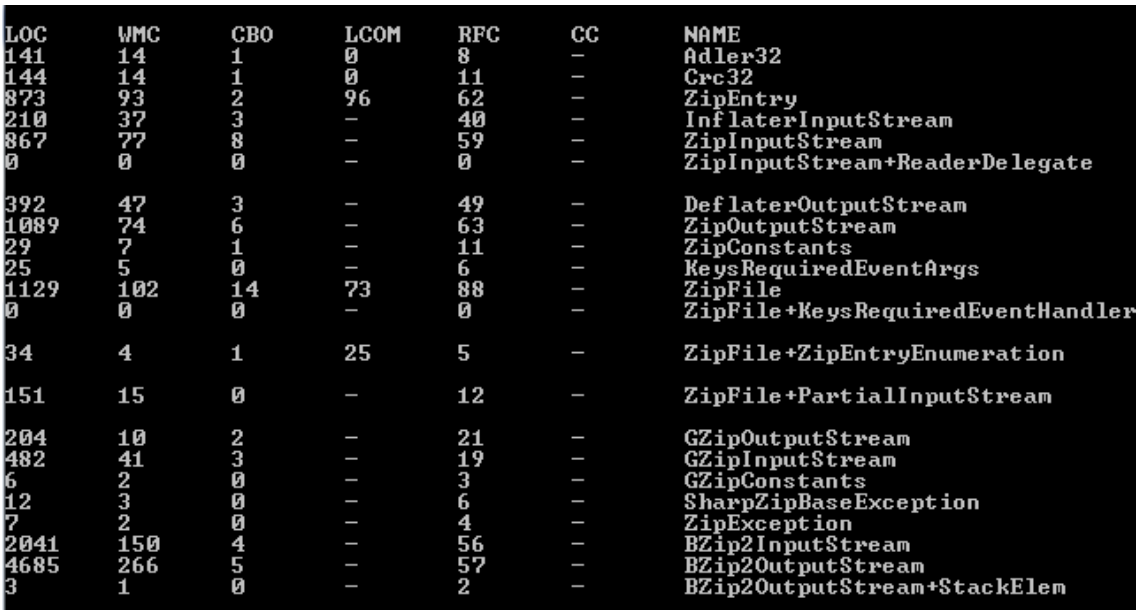
Το εργαλείο Vil[9] είναι ένα ελεύθερο λογισμικό το οποίο εκτελείται από τη γραμμή εντολών. Στην παρούσα διπλωματική χρησιμοποιήσαμε το εργαλείο με έκδοση 1.0.
Το Vil αναλύει ενδιάμεσο κώδικα σε .NET assemblies και υπολογίζει 33 μετρικές κλάσεων και μεθόδων. Οι μετρικές που υπολογίζει είναι οι εξής:

1. Number of Modules in an Assembly (Modules): Ο αριθμός των διαδικασιών (modules).
2. Number of Types that can have implementations (impTypes): Ο αριθμός των κλάσεων και δομών (structs) που υλοποιούν κάποια διεπαφή.
3. Number of defined interfaces and abstract classes (Abstracts): Ο αριθμός των αφηρημένων (abstract) κλάσεων ή των διεπαφών.
4. Efferent Coupling (CE): Ο αριθμός των διαδικασιών (modules) που εξαρτώνται από άλλες διαδικασίες.
5. Afferent Coupling (CA): Ο αριθμός των διαδικασιών (modules) έξω από το τρέχον assembly αρχείο που εξαρτώνται από τις διαδικασίες μέσα στο τρέχον αρχείο.
6. Abstractness (RMA): Το ποσοστό των τύπων (types) που είναι αφηρημένοι (abstract).
7. Instability (I ή RMI): Αστάθεια. I = Ce / (Ca + Ce).
8. Distance (RMD): RMA + RMI - 1)/2.
9. Number of classes: Ο αριθμός των κλάσεων.
10. Number of defined interfaces: Ο αριθμός των διεπαφών.

[9] www.1bot.com

11. Number of enumerations: Ο αριθμός των σταθερών απαρίθμησης (enumerations).
12. Number of structs: Ο αριθμός των δομών.
13. Number of Types: Ο αριθμός των κλάσεων, των σταθερών απαρίθμησης (enumerations), των διεπαφών και των δομών (structs).
14. Lines of Code (LOC): Οι γραμμές του ενδιάμεσου κώδικα (Intermediate Language instructions).
15. Dead Code: Ο αριθμός του ενδιάμεσου κώδικα που δεν εκτελείται ποτέ.
16. Number of events: Ο αριθμός των events.
17. Number of interfaces implemented: Ο αριθμός των διεπαφών που υλοποιούνται από κλάση ή δομή.
18. Cyclomatic Complexity (CC)
19. Weighted Methods per Class (WMC)
20. Depth of inheritance tree (DIT)
21. Coupling Between Objects (CBO)
22. Lack of Cohesion of Methods (LCOM)
23. Response for a Class (RFC)
24. Number of Children (NOC)
25. Number of properties: Ο αριθμός των ιδιοτήτων αντικειμένων (properties) μιας κλάσης ή δομής.
26. Number of constructors: Ο αριθμός των κατασκευαστών.
27. Number of methods (NOM)
28. Number of implementations: Ο αριθμός των μεθόδων και των κατασκευαστών.
29. Number of fields within a Type: Ο αριθμός των πεδίων (η ονομασία των μεταβλητών που ανήκουν στα αντικείμενα μιας κλάσης).
30. Number of parameters (PAR): Ο αριθμός των παραμέτρων μιας μεθόδου/κατασκευαστή.
31. Maximum size of stack: Το μέγιστο μέγεθος της στοίβας μιας μεθόδου/κατασκευαστή που αποκτά κατά την ώρα της εκτέλεσης.
32. Number of local variables: Ο αριθμός των τοπικών μεταβλητών σε μία μέθοδο/κατασκευαστή.

33. Number of "try" blocks: Ο αριθμός των ‘try’ μπλοκς (blocks) σε μία μέθοδο/κατασκευαστή.



LOC	WMC	CBO	LCOM	RFC	CC	NAME
141	14	1	0	8	-	Adler32
144	14	1	0	11	-	Crc32
873	93	2	96	62	-	ZipEntry
210	37	3	-	40	-	InflaterInputStream
867	77	8	-	59	-	ZipInputStream
0	0	0	-	0	-	ZipInputStream+ReaderDelegate
392	47	3	-	49	-	DeflaterOutputStream
1089	74	6	-	63	-	ZipOutputStream
29	7	1	-	11	-	ZipConstants
25	5	0	-	6	-	KeysRequiredEventArgs
1129	102	14	73	88	-	ZipFile
0	0	0	-	0	-	ZipFile+KeysRequiredEventHandler
34	4	1	25	5	-	ZipFile+ZipEntryEnumeration
151	15	0	-	12	-	ZipFile+PartialInputStream
204	10	2	-	21	-	GZipOutputStream
482	41	3	-	19	-	GZipInputStream
6	2	0	-	3	-	GZipConstants
12	3	0	-	6	-	SharpZipBaseException
7	2	0	-	4	-	ZipException
2041	150	4	-	56	-	BZip2InputStream
4685	266	5	-	57	-	BZip2OutputStream
3	1	0	-	2	-	BZip2OutputStream+StackElem

Σχήμα 3.18 : Εργαλείο Vil

Τέλος από την γραμμή εντολή ο χρήστης μπορεί να εξαγάγει μία xml ή html αναφορά με τις μετρικές που επίλεξε να υπολογιστούν.

3.3.10 CppCheck

Το CppCheck[10] είναι ένα λογισμικό ανοικτού κώδικα το οποίο διατίθεται ως plugin για τα περιβάλλοντα προγραμματισμού Eclipse, το Emacs, το Microsoft Visual Studio, το Jenkins, το Yasca, και άλλα. Επίσης διατίθεται και ως αυτόνομο εργαλείο ή μπορεί να τρέξει από την γραμμή εντολών. Αναπτύχθηκε από τον Daniel Marjamäki [23] και είναι ένα δημοφιλή εργαλείο στατικής ανάλυσης. Έχει εντοπίσει λάθη σε ανοικτού κώδικα έργα όπως το Vlc player, Linux Kernel, 7-zip κλπ. Στην παρούσα διπλωματική χρησιμοποιήσαμε το αυτόνομο εργαλείο με έκδοση 1.64.

Το εργαλείο αναλύει στατικά πηγαίο κώδικα στις γλώσσες προγραμματισμού C και C++. Η ανάλυση γίνεται με βάση ενός συνόλου ελέγχων που ομαδοποιούνται στις εξής κατηγορίες (severities):

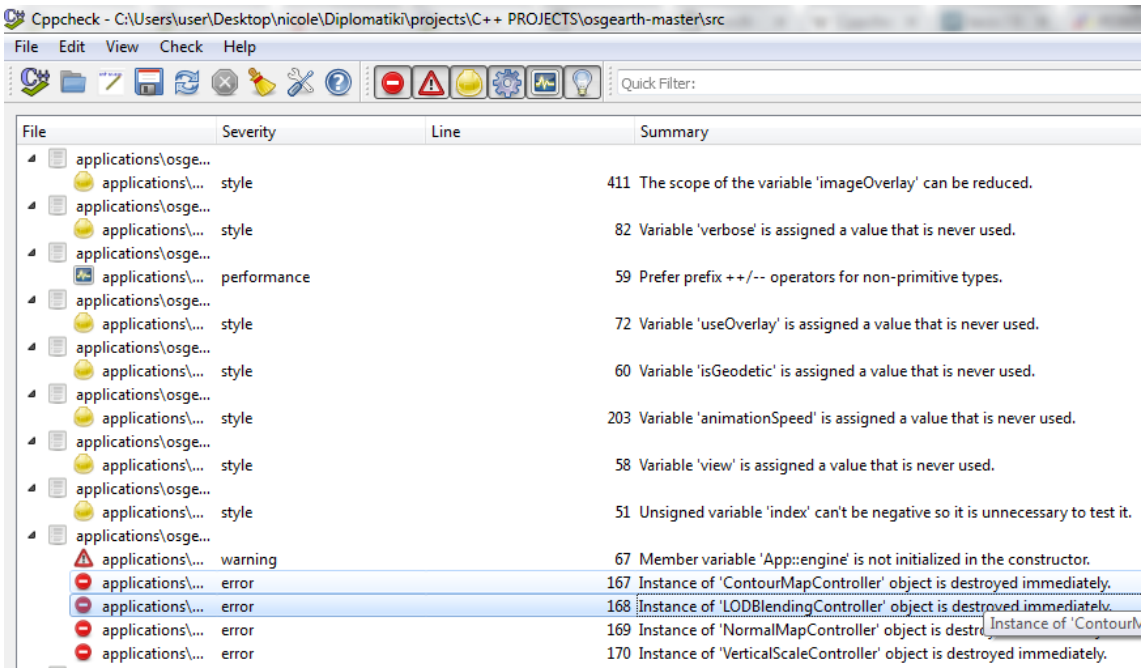
1. Error: Έλεγχος για λάθη (bugs). Για παράδειγμα, όταν γίνεται αναφορά σε θέση του πίνακα που είναι εκτός των ορίων του πίνακα (array index out of bounds),

[10] www.cppcheck.sourceforge.net

διαρροή μνήμης (memory leaks), αναφορά σε κενό δείκτη (possible null pointer dereferences).

2. Warning: Προειδοποιήσεις για την αποφυγή λαθών. Για παράδειγμα η μεταβλητή στον κατασκευαστή δεν έχει αρχικοποιηθεί (Member variable is not initialized in the constructor)
3. Style: Έλεγχος για το πως είναι γραμμένος ο κώδικας, στιλιστικοί έλεγχοι. Για παράδειγμα αχρησιμοποίητος κώδικας, περιττός κώδικας, μείωση της εμβέλειας μιας μεταβλητής (the scope of the variable can be reduced).
4. Performance: Εισηγήσεις για την βελτιστοποίηση της απόδοσης του κώδικα. Για παράδειγμα εισήγηση της χρήσης των χαρακτήρων ++/-- για τους μη πρωτογενής τύπους (non-primitive types).
5. Portability: Προειδοποιήσεις για πιθανά λάθη κατά την μετακίνηση του κώδικα. Ο κώδικας μπορεί να λειτουργεί διαφορετικά σε κάθε compiler.
6. Information: Πληροφορίες για τους ελέγχους.

Για κάθε αποτέλεσμα, εμφανίζεται το αρχείο και η γραμμή που εντοπίστηκε, η κατηγορία, εάν είναι ενεργοποιημένη (severity) στην οποία ανήκει και το σχετικό μήνυμα (summary) του λάθους.



Σχήμα 3.19 : Εργαλείο CppCheck

Επίσης ο χρήστης μπορεί να γράψει τους δικούς του κανόνες μέσω της xml για να ελέγξει τον κώδικα του. Τέλος, τα αποτελέσματα της ανάλυσης μπορούν να αποθηκευτούν σε ένα αρχείο τύπου xml, txt ή csv.

3.3.11 NDepend

Το Ndepend[11] είναι ένα εμπορικό λογισμικό στατικής ανάλυσης κώδικα και φέρει την υπογραφή της εταιρείας NDepend. Λειτουργεί ως αυτόνομο εργαλείο ή μπορεί να ενσωματωθεί με το Microsoft Visual Studio.

Στην παρούσα εργασία χρησιμοποιήσαμε το αυτόνομο εργαλείο με την δοκιμαστική (trial) έκδοση 5.2.1.8320.

Το NDepend αναλύει στατικά ενδιάμεσο κώδικα σε .NET. Το NDepend είναι ένα εργαλείο με αμέτρητες λειτουργίες. Τα κύρια χαρακτηριστικά του είναι ο έλεγχος του κώδικα με βάση περισσότερους από 200 κανόνες ελέγχου και ο υπολογισμός μετρικών προϊόντος. Οι κανόνες ελέγχου ανήκουν στις εξής κατηγορίες:

1. Code Quality: Έλεγχος μετρικών που καθορίζουν την ποιότητα του κώδικα. (Πολύ μεγάλες μέθοδοι (MLOC), πολύπλοκες μέθοδοι (CC), μέθοδοι με πολλές παραμέτρους (PAR), κλπ).
2. Code quality regression: Έλεγχος κώδικα που είναι ήδη ‘κακός’ και πολύπλοκος και γίνεται ακόμα πιο ‘κακός’ και πολύπλοκος. (Αποφυγή αύξηση της πολυπλοκότητας μιας μεθόδου που είναι ήδη πολύπλοκη, κλπ).
3. Object Oriented Design: Έλεγχος μετρικών σχετικά με τον αντικειμενοστρεφή σχεδιασμό κώδικα. (Οι μέθοδοι θα πρέπει να δηλώνονται ως στατικές (static) εάν είναι δυνατόν, οι κλάσεις πρέπει να έχουν μικρό DIT κλπ).
4. Design: Έλεγχος αν ο κώδικας πληροί συγκεκριμένες οδηγίες για τον σχεδιασμό του κώδικα. (Φωλιασμένοι τύποι (enums και structs) δεν πρέπει να είναι ορατοί, κλπ).
5. Architecture and Layering: Έλεγχοι σχετικοί με την αρχιτεκτονική του προγράμματος. (Assemblies με χαμηλή συνεκτικότητα, κλπ).
6. API Breaking Changes: Έλεγχοι σχετικοί με την διεπαφή API (Application Programming Interface) [4]. (Μια δημόσια (public) ορατή (visible) μέθοδος δεν είναι πλέον δημόσια ορατή ή έχει αφαιρεθεί, κλπ).

[11] www.ndepend.com

7. Test and Code Coverage: Έλεγχος αν ο κώδικας καλύπτεται από μονάδες ελέγχου (test units).
8. Dead code: Έλεγχος για νεκρό κώδικα.
9. Visibility: Έλεγχοι σχετικοί με την ορατότητα των κλάσεων, μεθόδων, κατασκευαστών, πεδίων. (Μέθοδοι που θα μπορούσαν να είχαν χαμηλότερη ορατότητα, κλπ, σταθερά πεδία (constant fields) δεν πρέπει να είναι δημόσια (public) ορατά, κλπ).
10. Purity-Immutability-Side-Effects: Ένα αντικείμενο λέγεται 'immutable' εάν η δήλωση του δεν αλλάζει από τότε που έχει δημιουργηθεί. Μια μέθοδος λέγεται 'pure' αν η εκτέλεση της δεν αλλάζει κανένα πεδίο. Τα αντικείμενα και οι μέθοδοι αυτοί μειώνουν τις παρενέργειες (side-effects, παρεμβολές από global μεταβλητές). (Εάν είναι δυνατό τα πεδία πρέπει να δηλώνονται ReadOnly, κλπ).
11. Naming Conventions: Έλεγχος αν ο κώδικας πληροί συγκεκριμένες οδηγίες για τα ονόματα των κλάσεων, (τα ονόματα διεπαφών θα πρέπει να ξεκινούν με 'I', τα ονόματα των μεθόδων θα πρέπει να ξεκινούν με μικρό γράμμα, κλπ).
12. Source Files Organization: Έλεγχοι για την οργάνωση των αρχείων.
13. .NET Framework Usage: Έλεγχοι σχετικοί με την .NET Framework πλατφόρμα (Αφαίρεση των κλήσεων στην μέθοδο GC.Collect(), κλπ).

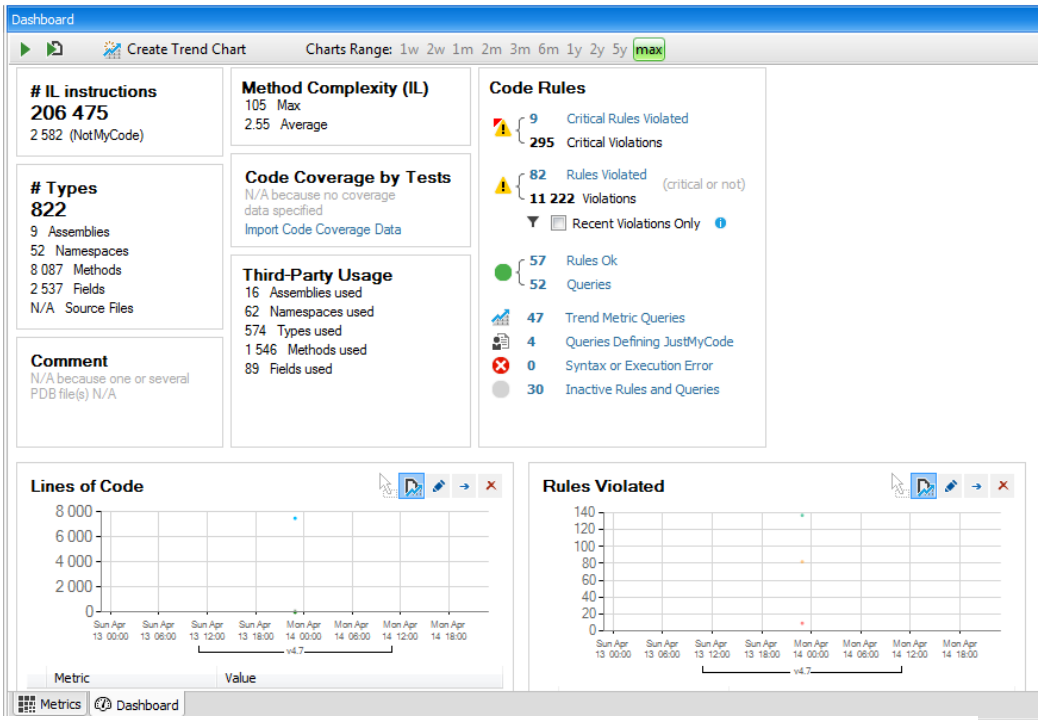
Ο πιο κάτω πίνακας (πίνακας 3.3) παρουσιάζει τις μετρικές που υπολογίζει το εργαλείο.

	Μετρική	Εφαρμογή	Assembly	Namespace	Τύπος	Μέθοδος	Πεδίο
1.	NbLinesOfCode (LOC)	✓	✓	✓	✓	✓	
2.	NbLinesOfComment (CLOC)	✓	✓	✓	✓	✓	
3.	PercentageComment	✓	✓	✓	✓	✓	
4.	NbILInstructions	✓	✓	✓	✓	✓	
5.	NbAssemblies	✓					
6.	NbNamespaces	✓	✓				
7.	Namespace level	✓	✓	✓			
8.	NbTypes	✓	✓	✓			
9.	NbMethods (NOM)	✓	✓	✓	✓		
10.	NbFields	✓	✓	✓	✓		
11.	PercentageCoverage	✓	✓	✓	✓	✓	
12.	NbLinesOfCodeCovered	✓	✓	✓	✓	✓	
13.	NbLinesOfCodeNotCovered	✓	✓	✓	✓	✓	
14.	Afferent coupling (Ca)		✓	✓	✓	✓	✓
15.	Efferent coupling (Ce)		✓	✓	✓	✓	
16.	Relational Cohesion(H)		✓				
17.	Instability (I)		✓				
18.	Abstractness (A/RMA)		✓				
19.	Distance from main sequence (D)		✓				
20.	Type level				✓		
21.	Type rank				✓		
22.	Lack of Cohesion Of Methods (LCOM)				✓		
23.	Lack of Cohesion Of Methods Henderson-Sellers (LCOM HS)				✓		
24.	Code Source Cyclomatic Complexity (CC)				✓	✓	
25.	IL Cyclomatic Complexity (ILCC)				✓	✓	
26.	Size of instance				✓		✓
27.	Association Between Class (ABC)				✓		
28.	Number of Children (NOC)				✓		
29.	Depth of Inheritance Tree (DIT)				✓		

30.	Method level					✓	
31.	Method rank					✓	
32.	IL Nesting Depth					✓	
33.	NbParameters (PAR)					✓	
34.	NbVariables					✓	
35.	NbOverloads					✓	
36.	PercentageBranchCoverage					✓	

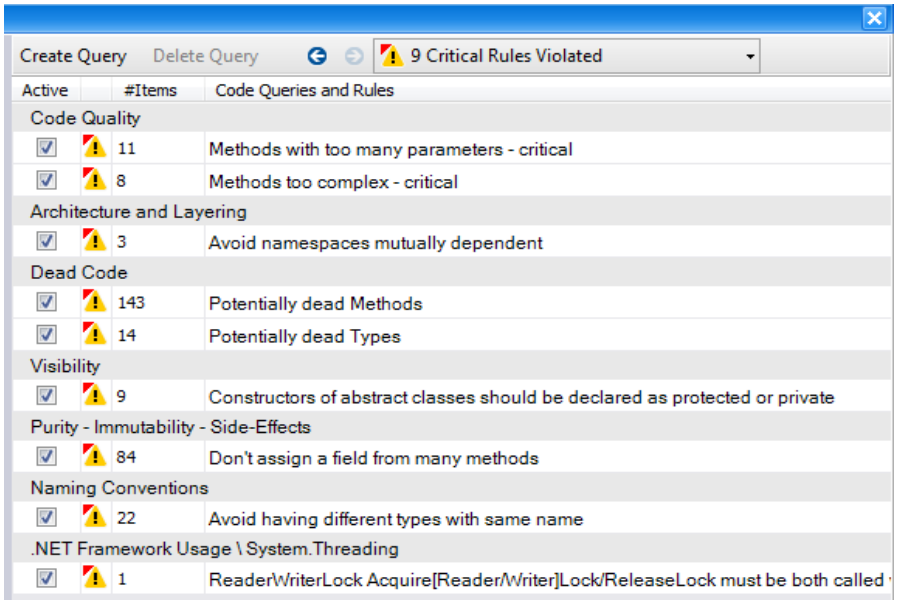
Πίνακας 3.3: Εργαλείο Ndepend, Μετρικές

Τα αποτελέσματα εμφανίζονται υπό τη μορφή Dashboard (σχήμα 3.19).



Σχήμα 3.19 Εργαλείο Ndepend, Dashboard

Οι υψηλής προτεραιότητας παραβιάσεις καλούνται ‘critical’ και δηλώνουν παραβιάσεις πολύ σοβαρές οι οποίες δεν πρέπει να υπάρχουν στο πρόγραμμα. Ο χρήστης μπορεί να επιλέξει την διακοπή της διαδικασίας build όταν εμφανίζεται μια critical παραβίαση κατά την ανάλυση.



Σχήμα 3.20 : Εργαλείο Ndepend ,Critical Rules Violated

Υπολογίζονται ακόμα, επιπλέον μετρικές (trend metrics) με τις οποίες ο χρήστης μπορεί να δημιουργήσει ανάλογα διαγράμματα (trend charts).

Trend Metrics			
Code Size	**Maximum and Average**	**Code Coverage**	**Third-party usage**
#Lines of Code	Max # Lines of Code for Methods (Just my code)	Percentage Code Coverage	#Third Party Assemblies Used
#Lines of Code (JustMyCode)	Average # Lines of Code for Methods	#Lines of Code Covered	#Third Party Namespaces Used
#Lines of Code (NotMyCode)	Average # Lines of Code for Methods with at least 3 Lined of Code	#Lines of Code Not Covered	#Third Party Types Used
#Lines of Code Added since the Baseline	Max # Lines of Code for Types (Just my code)	#Lines of Code in Types 100% Covered	#Third Party Methods Used
#IL Instructions	Average # Lines of Code for Types	#Lines of Code in Methods 100% Covered	#Third Party Fields Used
#IL Instructions (NotMyCode)	Max CC for Methods	Max Change Risk Analyzer and Predictor (C.R.A.P) Score	
#Lines of Comments	Average CC for Methods	Max Change Risk Analyzer and Predictor (C.R.A.P) Score	
#Assemblies	Max IL CC for Methods		

#Namespaces	Average IL CC for Methods		
#Types	Max IL Nesting Depth for Methods		
#Public Types	Average IL Nesting Depth for Methods		
#Classes	Max # of Methods for Types		
#Abstrac Classes	Average # of Methods for Types		
#Interfaces	Max # of Methods of Interfaces		
#Structures	Average # of Methods of Interfaces		
#Methods			
#Abstract Methods			
#Concrete Methods			
#Fields			

Πίνακας 3.4 : Εργαλείο Ndepend,Trend Metrics

Παρέχει επίσης στο χρήστη την δυνατότητα να γράψει τους δικούς του κανόνες ελέγχου με την τεχνική Code Query Language (CQLink).

Τέλος, μπορεί να γίνει σύγκριση μεταξύ 2 διαφορετικών build για το ίδιο αρχείο assembly καθώς επίσης και εισαγωγή αναφορών κάλυψης μονοπατιών (code coverage) από εργαλεία μονάδων ελέγχου (test units) με σκοπό την συλλογή μετρικών γύρω από την κάλυψη μονάδων ελέγχου (unit test coverage).

3.4 Έλεγχος εργαλείων με έργα λογισμικού

Αφού καταλήξαμε στα εργαλεία στατικής ανάλυσης κώδικα, έπρεπε τώρα να επιλέξουμε ένα σύνολο από ανοικτού κώδικα έργα για να συγκρίνουμε τα 11 εργαλεία, εφαρμόζοντας και μελετώντας την ανάλυση του κάθε εργαλείου. Η επιλογή των έργων ήταν τέτοια, έτσι ώστε να καλύπτονται συνολικά οι γλώσσες Java, C, C++, C#, VB.NET και PHP (τις οποίες υποστηρίζουν τα εργαλεία). Αναζητώντας στην διαδικτυακή υπηρεσία GitHub [13] και Sourgeforge επιλέξαμε άλλοτε τυχαία και άλλοτε μη, 100 έργα λογισμικού. Τα 50 από αυτά είναι γραμμένα στην γλώσσα Java και τα υπόλοιπα είναι στις γλώσσες C, C++, C#, .NET και PHP. Επιλέξαμε να αναλύσουμε έργα διαφορετικού μεγέθους, μικρού και μεγάλου ούτως ώστε να

καλύπτονται και οι απλοί χρήστες αλλά και μεγάλοι οργανισμοί ανάπτυξης λογισμικού που πιθανόν να ενδιαφέρονται για την χρήση των εργαλείων. Για τους σκοπούς της αναφοράς της διπλωματικής εργασίας παρουσιάζουμε τα αποτελέσματα 5 έργων λογισμικού στην γλώσσας java, 2 έργων λογισμικού στην γλώσσα C, 2 έργων λογισμικού στην γλώσσα C++, 1 έργου λογισμικού στην γλώσσα C# και 2 έργων λογισμικού στην γλώσσα PHP.

Έργο Λογισμικού	**Γλώσσα**	**Μέγεθος (#Αρχεία)**	**Μέγεθος (Πηγαίος κώδικας MB)**
1. BlackWindowSpider	Java	589	6,09
2. Accessor	Java	81	0,35
3. Esami	Java	49	0,55
4. BookMarker	Java	18	0,08
5. JavaGame	Java	28	0,06
6. Libgit2	C	238	2,36
7. Redis	C	88	2,01
8. DotNetOpenAuth	C#	26	0,83
9. K-Meleon	C++	425	90,1
10. Remote Control Center	C++	69	8,75
11. Mustache	PHP	31	0,17
12. PHP-Excel	PHP	189	3,61

Πίνακας 3.5 : Έργα λογισμικού

Κεφάλαιο 4

Σύγκριση εργαλείων στατικής ανάλυσης κώδικα

4.1 Εισαγωγή

Στο κεφάλαιο αυτό θα περιγράψουμε τις μετρικές αξιολόγησης των εργαλείων και ακολούθως θα αναλύσουμε την μεθοδολογία που ακολουθήθηκε για την σύγκριση των εργαλείων και τα αποτελέσματα της. Πιο συγκεκριμένα, η συγκριτική αξιολόγηση των

εργαλείων βασίστηκε σε 8 μετρικές. Βάση αυτών των μετρικών αξιολογήσαμε τα εργαλεία με τα έργα λογισμικού και θα παρουσιάσουμε τα αποτελέσματα της σύγκρισης.

4.2 Μετρικές αξιολόγησης εργαλείων

Αναλύοντας τα διάφορα έργα λογισμικού με τα εργαλεία στατικής ανάλυσης κώδικα παρατηρήσαμε ότι τα εργαλεία παρέχουν κοινές ή/και διαφορετικές λειτουργίες όσον αφορά συγκεκριμένους άξονες. Οι άξονες αυτοί κρίναμε ότι ήταν οι κατάλληλες μετρικές αξιολόγησης για την σύγκριση των εργαλείων και την ανάδειξη των 'σημαντικότερων' εργαλείων. Πιο κάτω παρουσιάζονται οι 8 μετρικές αξιολόγησης των εργαλείων.

Μετρική 1

Αριθμός υποστηριζόμενων γλωσσών προγραμματισμού

Κάποιες εταιρείες μπορεί να αναπτύσσουν λογισμικά σε μία συγκεκριμένη γλώσσα προγραμματισμού ενώ άλλες μπορεί να αναπτύσσουν λογισμικά σε περισσότερες από μία γλώσσες. Επομένως για την διευκόλυνση των χρηστών, όσες περισσότερες γλώσσες υποστηρίζει το εργαλείο, τόσο το καλύτερο.

Μετρική 2

Διεπαφή με το χρήστη-Έξοδος αποτελεσμάτων

Η διεπαφή με τον χρήστη είναι μια σημαντική μετρική για την αξιολόγηση των εργαλείων. Καταρχήν, η διαδικασία της ανάλυσης κάποιου έργου πρέπει να είναι εύκολη και γρήγορη. Αν η διαδικασία απαιτεί πολλά και πολύπλοκα βήματα ο χρήστης θα κουραστεί να συνεχίσει.

Το εργαλείο μπορεί εύκολα να υποτιμηθεί και οι λειτουργίες του να μείνουν αναξιοποίητες αν η διεπαφή με τον χρήστη δεν είναι σωστά σχεδιασμένη και υλοποιημένη. Ο χρήστης, θα προσαρμοστεί πιο εύκολα και θα αφοσιωθεί περισσότερο σε ένα κατανοητό, εύχρηστο και φιλικό εργαλείο.

Επίσης σε μεγάλα έργα λογισμικού τα εργαλεία επιστρέφουν ένα μεγάλο αριθμό αποτελεσμάτων. Αν η έξοδος των αποτελεσμάτων δεν ακολουθεί κάποια δομή τότε ο χρήστης θα κουραστεί να τα εξετάσει και πιθανόν να μη το κάνει.

Η διεπαφή με το χρήστη είναι μια σημαντική παράμετρος για την αξιολόγηση των εργαλείων όπως και κάθε λογισμικού.

Μετρική 3

Επεξήγηση αποτελεσμάτων

Η ύπαρξη επεξηγήσεων στα αποτελέσματα θεωρείται αναγκαία για τα εργαλεία στατικής ανάλυσης κώδικα αφού σκοπός τους είναι να εντοπίσουν λάθη που ο χρήστης δεν μπορεί να εντοπίσει. Τα λάθη πρέπει να συνοδεύονται από επεξηγηματικά σχόλια ούτως ώστε ο χρήστης να κατανοήσει την ύπαρξη του λάθους και να μπορέσει να το διορθώσει βελτιώνοντας έτσι τον κώδικα. Αν οι επεξηγήσεις δεν είναι σαφείς ο χρήστης πιθανόν να μην κατανοήσει την αιτιότητα του λάθους και είτε θα προσπελάσει το σφάλμα αυτό είτε θα προσπαθήσει να το διορθώσει χωρίς όμως να γνωρίζει γιατί.

Μετρική 4

Εξαγωγή αναφορών

Η εξαγωγή των αποτελεσμάτων σε αναφορές (reports) διαφόρων μορφών (xml, html, κλπ) είναι βοηθητικές για την κατανόηση των αποτελεσμάτων καθώς είναι πιο αναλυτικές. Επίσης διάφορα στατιστικά διαγράμματα είναι χρήσιμα για την εξαγωγή στατιστικών και την περεταίρω μελέτη των αποτελεσμάτων γύρω από τα λάθη των προγραμματιστών. Για παράδειγμα μπορούν να χρησιμοποιηθούν για να υποδείξουν τμήματα του έργου που ενδεχομένως χρίζουν διόρθωσης πιο έντονα.

Μετρική 5

Δυνατότητα προσαρμογής

Αν ο χρήστης έχει την δυνατότητα να καθορίσει ο ίδιος ρυθμίσεις και παραμέτρους που επιθυμεί τότε η έξοδος θα είναι η καλύτερη δυνατή για τον χρήστη, χωρίς κουραστικές λεπτομέρειες που πιθανόν να μην τον αφορούν. Είναι σημαντικό ο χρήστης να μπορεί να ενεργοποιεί τους κανόνες ελέγχους ή τις μετρικές που επιθυμεί καθώς και να μπορεί να απενεργοποιεί ελέγχους που δημιουργούν λανθασμένα αποτελέσματα (false positives). Ένα επιπλέον προτέρημα κάποιου εργαλείου είναι η δυνατότητα συγγραφής νέων κανόνων από τον χρήστη.

Μετρική 6

Κοινοποίηση αποτελεσμάτων

Στην βιομηχανία λογισμικού η ανάπτυξη λογισμικού είναι συνήθως μια ομαδική διαδικασία. Η κοινοποίηση των ρυθμίσεων και των αποτελεσμάτων της ανάλυσης στους υπόλοιπους προγραμματιστές της ομάδας προβάλει στους προγραμματιστές τα λάθη του κώδικα και μπορεί να βοηθήσει στην καλύτερη κατανόηση των αποτελεσμάτων και στην περεταίρω ενημέρωση και συντήρηση του κώδικα.

Μετρική 7

Μετρικές ποιότητας κώδικα

Οι μετρικές ποιότητας κώδικα είναι η βάση για τα περισσότερα εργαλεία στατικής ανάλυσης κώδικα. Οι μετρικές ορίζουν πόσο πολύπλοκος, ευανάγνωστος και συντηρήσιμος είναι ο κώδικας και είναι ένα καθοριστικό μέτρο για την ένδειξη της ποιότητας του κώδικα.

Για την αξιολόγηση των εργαλείων στην γλώσσα Java επιλέξαμε 7 μετρικές ποιότητας κώδικα για την εκτίμηση της πολυπλοκότητας και της ποιότητας μίας αντικειμενοστρεφούς σχεδίασης λογισμικού και ελέγξαμε αν τα εργαλεία ελέγχουν αυτές τις μετρικές και αν επιστρέφουν την ίδια τιμή για την ίδια είσοδο. Οι μετρικές είναι οι εξής:

1. LOC
2. NOM
3. CC
4. WMC
5. DIT
6. LCOM
7. NOC

Για την γλώσσα C επιλέξαμε 6 μετρικές. Οι μετρικές είναι οι εξής:

1. LOC
2. Number of functions
3. CC
4. WMC
5. LCOM
6. NBD

Για την γλώσσα C++ επιλέξαμε 5 μετρικές. Οι μετρικές είναι οι εξής:

1. LOC
2. Number of functions
3. CC
4. WMC
5. NBD

Για την γλώσσα C# επιλέξαμε 4 μετρικές. Οι μετρικές είναι οι εξής:

1. LOC
2. CC
3. WMC
4. DIT
5. LCOM
6. NOC

Για την γλώσσα PHP επιλέξαμε 4 μετρικές. Οι μετρικές είναι οι εξής:

1. LOC
2. Number of statements
3. NOM
4. CC
5. WMC

Σύμφωνα και με το πρότυπο ISO 9126 [16] οι μετρικές αυτές αξιολογούν κατά πόσο είναι εφικτή η συντηρησιμότητα (maintainability) του κώδικα. Η συντηρησιμότητα αποτελείται επιπλέον από τις εξής κατηγορίες:

1. Analyzability: Η ικανότητα του λογισμικού να μπορεί να διαγνωστεί για ‘ατέλειες’ ή για αιτίες που πιθανόν να προκαλέσουν την αποτυχία εκτέλεσης του λογισμικού.
2. Changeability: Η ικανότητα του λογισμικού να είναι τέτοιο υλοποιημένο που να επιτρέπει την τροποποίηση του.
3. Stability: Η ικανότητα του λογισμικού να αποφεύγει τις οποιοσδήποτε αρνητικές συνέπειες όταν το λογισμικό τροποποιηθεί.

4. Testability: Η ικανότητα του λογισμικού να μπορεί να επικυρωθεί (tested) μετά από κάποια τροποποίηση.

Maintainability							
Analyzability		**Changeability**		**Stability**		**Testability**	
High related Metrics	**Related Metrics**	**High related Metrics**	**Related Metrics**	**High related Metrics**	**Related Metrics**	**High related Metrics**	**Related Metrics**
LOC	NOC	LOC		LCOM	LOC	LOC	NOC
NOM		NOC			NOM	NOM	
CC		NOM			CC	CC	
WMC		CC			WMC	WMC	
DIT		WMC			DIT	DIT	
LCOM		DIT			NOC	LCOM	

Πίνακας 4.1: ISO 9126 μετρικές ποιότητας κώδικα

Υψηλές τιμές αυτών των μετρικών δυσκολεύουν την συντήρηση του κώδικα.

Μετρική 8

Αξιοπιστία αποτελεσμάτων

Η μετρική ‘αξιοπιστία αποτελεσμάτων’ είναι η σημαντικότερη μετρική καθώς ο κύριος στόχος των εργαλείων είναι ο εντοπισμός λαθών στον κώδικα πριν ο κώδικας εκτελεστεί. Ένα ‘καλό’ εργαλείο στατικής ανάλυσης κώδικα θεωρείται αυτό που εντοπίζει εύστοχα και σημαντικά ‘λάθη’ και ο αριθμός των λανθασμένων αποτελέσματων (false positives) που παράγει είναι πολύ μικρός. Συγκεκριμένα αξιολογούμε τα αποτελέσματα των έργων λογισμικού που επιστρέφει κάθε εργαλείο με βάση την ποιότητα, δηλαδή την σοβαρότητα του λάθους και την ευστοχία τους, δηλαδή το αποτέλεσμα που επιστρέφει να είναι όντως σωστό (true positive). Κάθε εργαλείο αναλύει τον κώδικα με βάση ενός σύνολου από κανόνες ελέγχου. Εξετάζουμε κυρίως τα αποτελέσματα που το εργαλείο επιστρέφει ως τα πιο σοβαρά.

4.3 Σύγκριση εργαλείων με βάση τις μετρικές αξιολόγησης εργαλείων

Για την υλοποίηση της διπλωματικής εργασίας επιλέξαμε ένα σύνολο έργων ανοικτού κώδικα (Open Source Software – OSS) για να αξιολογήσουμε τα 11 εργαλεία στατικής

ανάλυσης κώδικα. Καταγράψαμε τα αποτελέσματα της ανάλυσης και εξάγαμε τις ανάλογες αναφορές. Στην συνέχεια παρουσιάζουμε την αξιολόγηση των εργαλείων περιγράφοντας τα αποτελέσματα διάφορων έργου λογισμικού με βάση τις μετρικές αξιολόγησης που ειπώθηκαν στο υποκεφάλαιο 4.2.

4.3.1 Αριθμός υποστηριζόμενων γλωσσών προγραμματισμού

Στην μετρική αυτή συγκρίνουμε τα εργαλεία ως προς τον αριθμό αλλά και το είδος των γλωσσών προγραμματισμού που υποστηρίζουν. Αναζητούμε δηλαδή τα εργαλεία εκείνα που υποστηρίζουν τις πιο διαδεδομένες γλώσσες. Στο σημείο αυτό θα πρέπει να τονίσουμε ότι η ποιότητα των εργαλείων δεν θα κριθεί από τον αριθμό των υποστηριζόμενων γλωσσών προγραμματισμού όμως η υποστήριξη διαδεδομένων γλωσσών προγραμματισμού από ένα εργαλείο είναι ένα προτέρημα.

Το εργαλείο Sonarqube υποστηρίζει αρκετές διαδεδομένες γλώσσες προγραμματισμού όπως Java, C#, C, C++, Python, PHP ωστόσο πρέπει να αναφέρουμε ότι δεν διατίθονται όλες δωρεάν. Τα plugin για τις γλώσσες C, C++, COBOL, PL/I, PL/SQL και Visual Basic 6 είναι εμπορικά. Αυτό αποτελεί ένα μειονέκτημα για το εργαλείο καθώς αν όλες οι γλώσσες ήταν διαθέσιμες δωρεάν το Sonarqube θα ήταν ίσως το πιο ολοκληρωμένο δωρεάν εργαλείο στατικής ανάλυσης κώδικα όσον αφορά τις γλώσσες προγραμματισμού που υπάρχει αυτή την στιγμή στην βιομηχανία. Το εργαλείο SourceMonitor υποστηρίζει τις γλώσσες Java, C#, C/C++, VB.NET, Delphi, Visual Basic, Html, το LocMetrics υποστηρίζει τις γλώσσες Java, C#, C++, Sql, το CppCheck τις γλώσσες C και C++, το PMD υποστηρίζει τις γλώσσες Java, Javascript. Τα εργαλεία Metrics 1.3.6 Checkstyle, FindBugs, Ckjm είναι αποκλειστικά για την γλώσσα Java, και τα εργαλεία Vil και Ndepend υποστηρίζουν όλες τις γλώσσες .NET.

Ο πιο κάτω πίνακας (πίνακας 4.2) παρουσιάζει τις γλώσσες προγραμματισμού που υποστηρίζει κάθε εργαλείο.

Εργαλεία	Γλώσσα Προγραμματισμού																
Metrics 1.3.6	Java																
Checkstyle	Java																
FindBugs	Java																
Ckjm	Java																
CppCheck	C/C++																
Vil	.NET																
NDepend	.NET																
PMD	Java	JavaScript															
LocMetrics	Java	C#	C++	Sql													
SourceMonitor	Java	C#	C/C++	VB.NET	Delphi	Visual Basic	Html										
Sonarqube	Java	C#	C/C++	JavaScript	PHP	Python	Android	Abab	Flex/ActionSript	COBOL	Erlang	Groovy	PL/I	PL/SQL	VB.NET	Visual Basic 6	WEB

Σχήμα 4.2: Μετρική Αριθμός υποστηριζόμενων γλωσσών προγραμματισμού

4.3.2 Διεπαφή με το χρήστη-Έξοδος αποτελεσμάτων

Για την μετρική αυτή συγκρίναμε τα 11 εργαλεία ως προς την διαδικασία ανάλυσης και την διεπαφή με τον χρήστη ως προς τις λειτουργίες του εργαλείου και την έξοδο των αποτελεσμάτων. Σημειώνουμε πως η ποιότητα του εργαλείου δεν κρίνεται από την διεπαφή με τον χρήστη αλλά μια κακή διεπαφή πιθανόν να αποτρέψει τον χρήστη να χρησιμοποιήσει το εργαλείο.

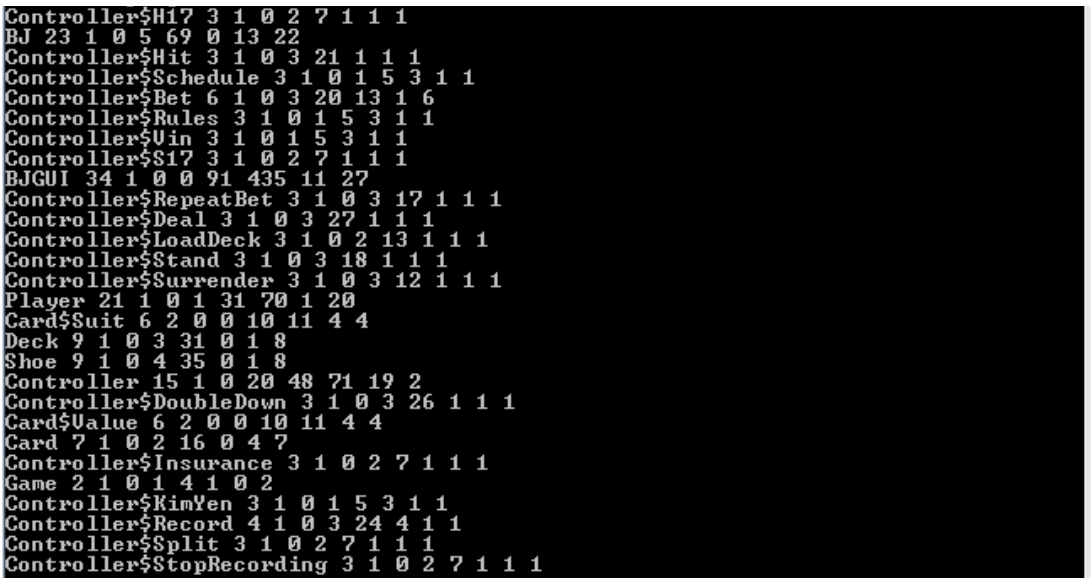
Ckjm

Το εργαλείο ckjm εκτελείται από την γραμμή εντολών και δεν υπάρχει διεπαφή με τον χρήστη. Η διαδικασία ανάλυσης είναι αρκετά έυκολη. Από την γραμμή εντολών ο χρήστης εκτελεί την εντολή

```
java -jar /.../ckjm-1.9.jar .../*.class
```

όπου /.../ckjm-1.9.jar το μονοπάτι που βρίσκεται το ckjm-1.9.jar και .../ckjm/*.class το μονοπάτι με τα αρχεία που θέλουμε να αναλύσουμε.

Η έξοδος των αποτελεσμάτων δεν είναι καθόλου ευανάγνωστη. Παρουσιάζονται μόνο το όνομα του αρχείου και δίπλα οι τιμές των μετρικών χωρίς το όνομα των μετρικών. Έτσι ο χρήστης πρέπει να ανατρέξει στην ιστοσελίδα του εργαλείου για να δει τις μετρικές που αντιστοιχούν στα αποτελέσματα. Σε μεγάλα project η ανάλυση των αποτελεσμάτων είναι αρκετά κουραστική.



Σχήμα 4.1: Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων, εργαλείο ckjm

Συμπεραίνουμε ότι, ενώ υπολογίζει σημαντικές μετρικές η παρουσίαση των αποτελεσμάτων είναι κουραστική όσον αφορά την ανάγνωση και πιθανόν να αποτρέπει την χρήση του.

Vil

Το εργαλείο vil εκτελείται από την γραμμή εντολών και δεν υπάρχει διεπαφή με τον χρήστη. Η διαδικασία ανάλυσης είναι αρκετά απλή. Από την γραμμή εντολών ο χρήστης εκτελεί την εντολή

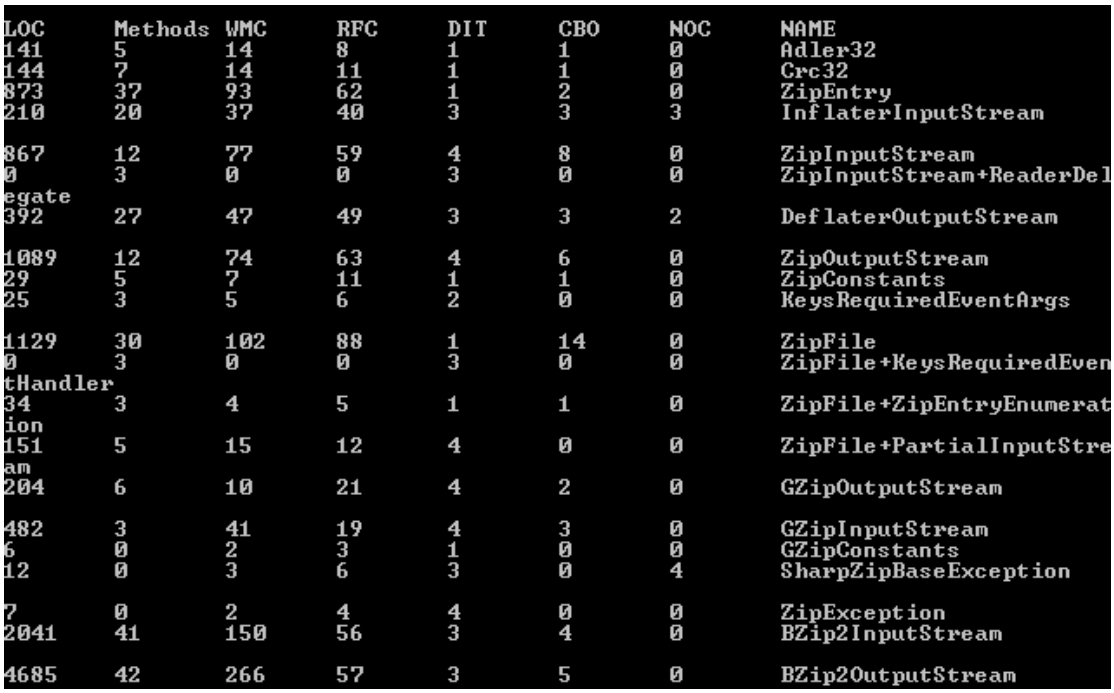
```
vil /a=C:\...\*.dll /m=loc,wmc,lcom,cc /sc=class
```

ή την εντολή

```
vil /a=C:\...\*.exe /m=loc,wmc,lcom,cc /sc=class vil
```

όπου /a=C:\...*.dll ή /a=C:\...*.exe το μονοπάτι που βρίσκονται τα αρχεία .dll ή .exe ανάλογα.

Η παρουσίαση των αποτελεσμάτων του Vil είναι σαφώς καλύτερη από την έξοδο του εργαλείου ckjm αφού το Vil ακολουθεί μια πιο προσεγμένη και εμφανίσιμη δομή (σχήμα 4.2)



LOC	Methods	WMC	RFC	DIT	CBO	NOC	NAME
141	5	14	8	1	1	0	Adler32
144	7	14	11	1	1	0	Crc32
873	37	93	62	1	2	0	ZipEntry
210	20	37	40	3	3	3	InflaterInputStream
867	12	77	59	4	8	0	ZipInputStream
0	3	0	0	3	0	0	ZipInputStream+ReaderDel
egate							
392	27	47	49	3	3	2	DeflaterOutputStream
1089	12	74	63	4	6	0	ZipOutputStream
29	5	7	11	1	1	0	ZipConstants
25	3	5	6	2	0	0	KeysRequiredEventArgs
1129	30	102	88	1	14	0	ZipFile
0	3	0	0	3	0	0	ZipFile+KeysRequiredEven
tHandler							
34	3	4	5	1	1	0	ZipFile+ZipEntryEnumerat
ion							
151	5	15	12	4	0	0	ZipFile+PartialInputStre
am							
204	6	10	21	4	2	0	GZipOutputStream
482	3	41	19	4	3	0	GZipInputStream
6	0	2	3	1	0	0	GZipConstants
12	0	3	6	3	0	4	SharpZipBaseException
7	0	2	4	4	0	0	ZipException
2041	41	150	56	3	4	0	BZip2InputStream
4685	42	266	57	3	5	0	BZip2OutputStream

Σχήμα 4.2: Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων, εργαλείο Vil

Επομένως, παρά το γεγονός ότι το εργαλείο δεν διαθέτει διεπαφή με τον χρήστη, η εύκολη διαδικασία ανάλυσης και η κατανοητή εμφάνιση των τιμών των μετρικών συνιστούν την χρήση του, για χρήστες που επιθυμούν να υπολογίσουν μετρικές σε γλώσσες .NET.

Sonarqube

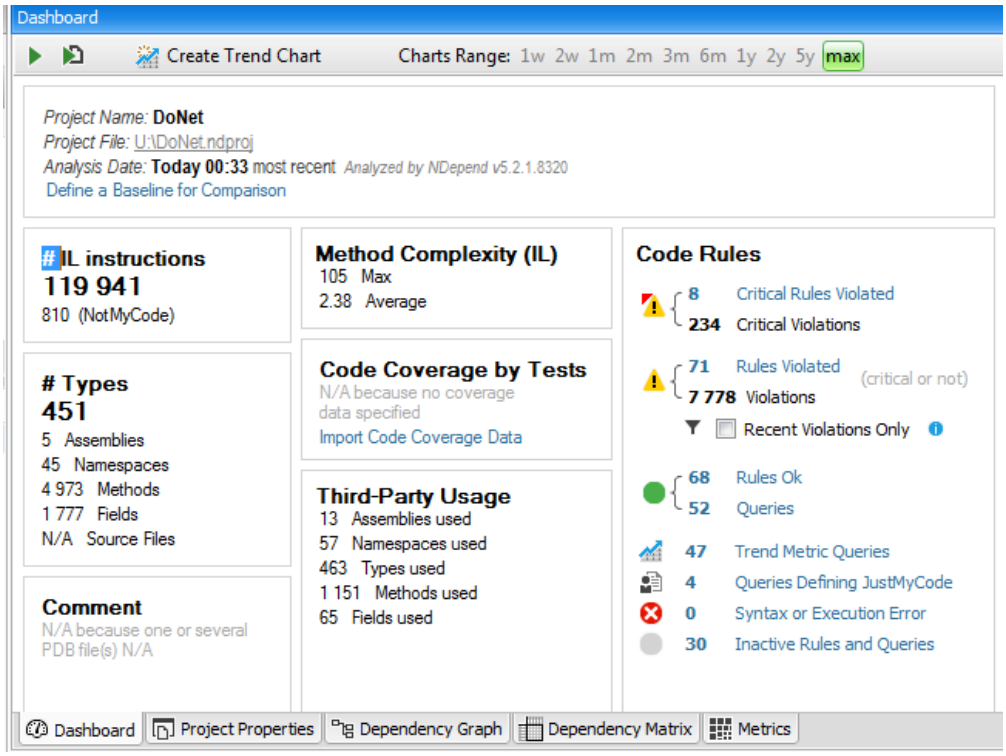
Η διαδικασία ανάλυσης του Sonarqube απαιτεί περισσότερα βήματα από τα υπόλοιπα εργαλεία και επομένως και περισσότερο χρόνο. Για την εκτέλεση του εργαλείου Sonarqube ο χρήστης πρέπει πρώτα να ξεκινήσει τον sonar server (StartSonar.bat). Στην συνέχεια από την γραμμή εντολών και τον φάκελο που βρίσκετε το έργο που επιθυμεί να αναλύσει (το οποίο πρέπει να περιέχει ένα αρχείο properties με σχετικές πληροφορίες) εκτελεί το αρχείο sonar-runner.bat. Τότε ανατρέχει στο localhost:9000 και βλέπει τα αποτελέσματα της ανάλυσης. Τα αποτελέσματα εμφανίζονται υπό την μορφή dashboard το οποίο περιλαμβάνει επιμέρους widget τα οποία αντιστοιχούν στα αποτελέσματα της ανάλυσης. Πίσω από κάθε widget υπάρχουν τα αντίστοιχα αποτελέσματα, τα οποία μπορείς να τα δεις στο σημείο που εντοπίστηκαν στον κώδικα.

Σχήμα 4.3: Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων, εργαλείο Sonarqube

Η διαδικασία ανάλυσης είναι σχετικά χρονοβόρα και ο χρόνος ανάλυσης συχνά είναι μεγάλος (σε μεγάλα έργα λογισμικού). Η διεπαφή με τον χρήστη είναι φιλική και λειτουργική με την έννοια ότι ο χρήστης μπορεί εύκολα να κατανοήσει και να αξιοποιήσει τις λειτουργίες του εργαλείου.

NDepend

Η διαδικασία ανάλυσης είναι αρκετά εύκολη. Ο χρήστης δημιουργεί ένα νέο NDepend project και επιλέγει τα assemblies αρχεία που επιθυμεί να αναλύσει. Η διεπαφή με τον χρήστη είναι πολύ παρόμοια με το εργαλείο Sonarqube καθώς τα αποτελέσματα εμφανίζονται σε διάφορα widgets υπό μορφή dashboard. Επίσης ο χρήστης μπορεί να δει τα αποτελέσματα στα σημεία του κώδικα που εντοπίστηκαν.



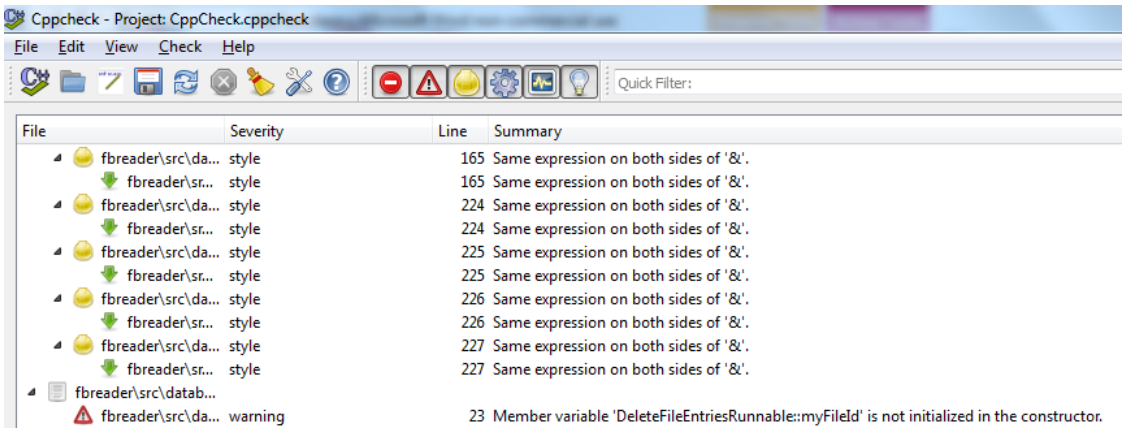
Σχήμα 4.4 : Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο Ndepend

Το Ndepend παρέχει μια φιλική και λειτουργική διεπαφή με τον χρήστη, πολύ κοινή με αυτήν του Sonarqube.

CppCheck

Η διαδικασία ανάλυσης είναι αρκετά εύκολη. Ο χρήστης δημιουργεί ένα νέο CppCheck project και επιλέγει τον φάκελο που βρίσκεται ο πηγαίος κώδικας που επιθυμεί να αναλύσει.

Για κάθε ‘λάθος’ εμφανίζεται το αρχείο στο οποίο ανήκει, η κατηγορία, η γραμμή που εντοπίστηκε και ένα σχετικό μήνυμα. Κάνοντας διπλό κλικ στο λάθος ο χρήστης μπορεί να δει τον κώδικα στο οποίο εντοπίστηκε το λάθος.

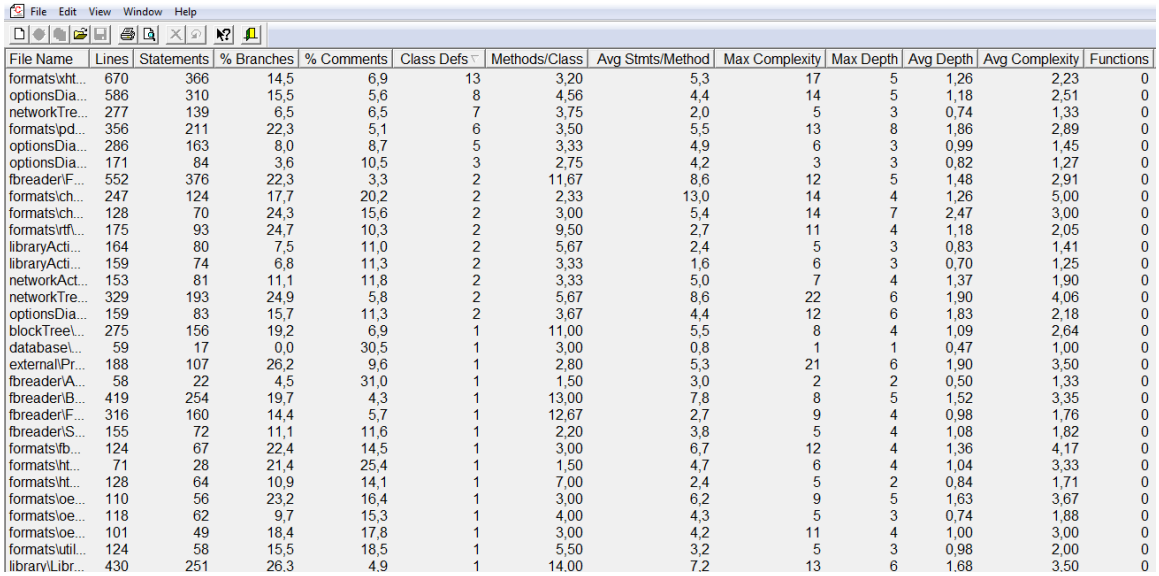


Σχήμα 4.5 : Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο CppCheck

Το CppCheck είναι πολύ εύκολο στην χρήση και η διεπαφή με τον χρήστη είναι απλή και κατανοητή.

SourceMonitor

Η διαδικασία ανάλυσης είναι απλή. Ο χρήστης δημιουργεί ένα νέο έργο (project), δηλώνει το όνομα του έργου, το φάκελο που θα το αποθηκεύσει , επιλέγει την γλώσσα του πηγαίου κώδικα (.java,.cpp κλπ), τη διαδρομή του, και το όνομα του checkpoint που θα δημιουργηθεί.

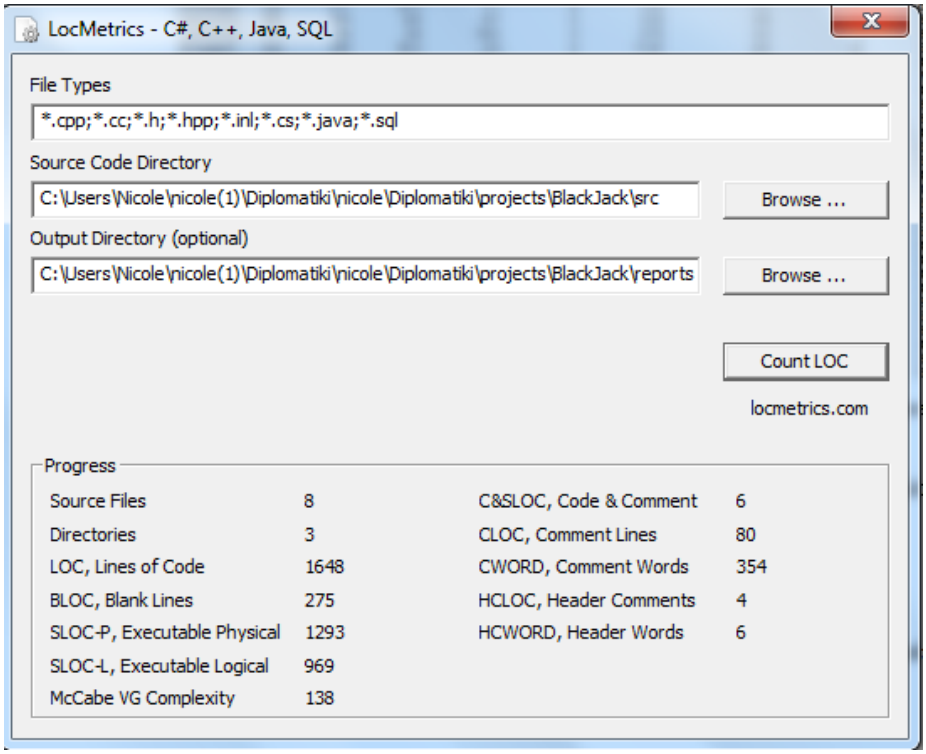


Σχήμα 4.6 : Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο SourceMonitor

Η διεπαφή με τον χρήστη είναι φιλική και η έξοδος των αποτελεσμάτων κατανοητή.

LocMetrics

Η διαδικασία ανάλυσης είναι πολύ απλή. Ο χρήστης επιλέγει τον φάκελο με τον πηγαίο κώδικα και ακολούθως επιλέγει την εντολή Count LOC και βλέπει στο ίδιο παράθυρο τις τιμές των μετρικών.

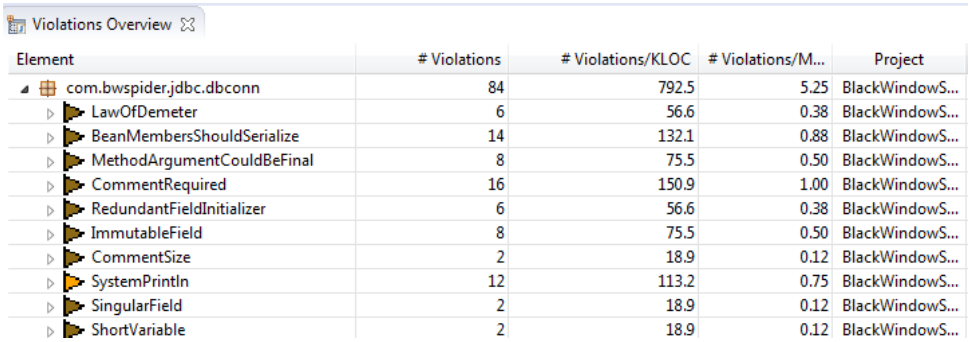


Σχήμα 4.7 : Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο LocMetrics

Οι λειτουργίες του εργαλείου είναι πολύ περιορισμένες και έτσι η χρήση του είναι πολύ εύκολη.

PMD

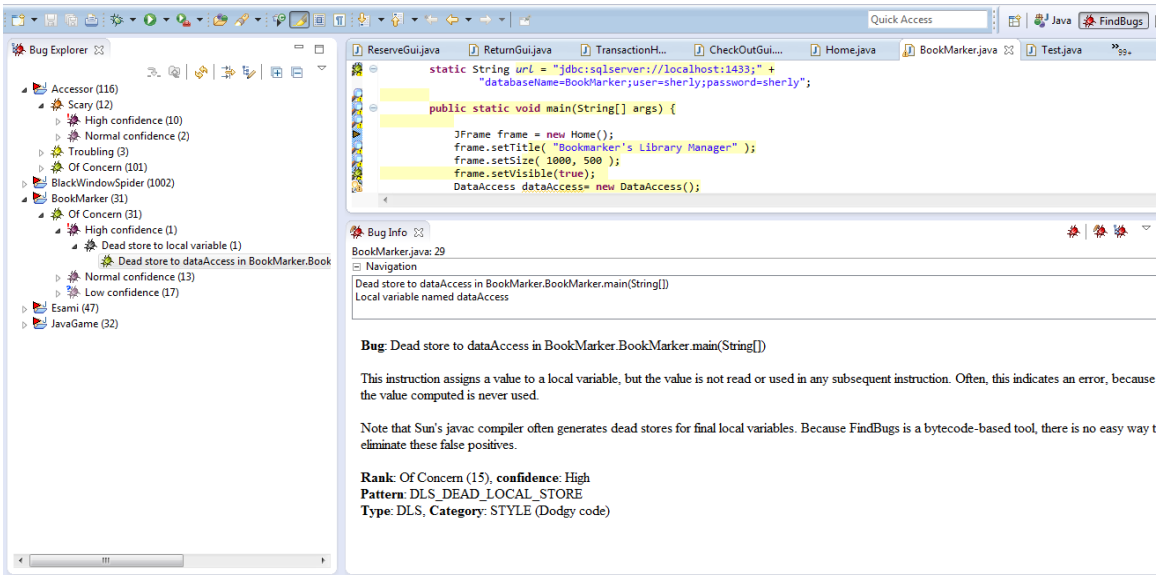
Η διαδικασία ανάλυσης είναι αρκετά απλή. Ο χρήστης επιλέγει το έργο που επιθυμεί να αναλύσει και από τα properties του έργου ενεργοποιεί την αντίστοιχη επιλογή (checkbox) που βρίσκεται στην καρτέλα (tab) PMD. Ως προς τη διεπαφή με τον χρήστη, για το εργαλείο υπάρχει το αντίστοιχο Perspective με τις όψεις Violations Overview και Violations Outline που επιτρέπει την επισκόπηση των αποτελεσμάτων και διευκολύνει την πλοήγηση σε αυτά. Τα αποτελέσματα επίσης επισημαίνονται και στον κώδικα στο eclipse με τα σχετικά μηνύματα.



Σχήμα 4.8 : Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο PMD

FindBugs

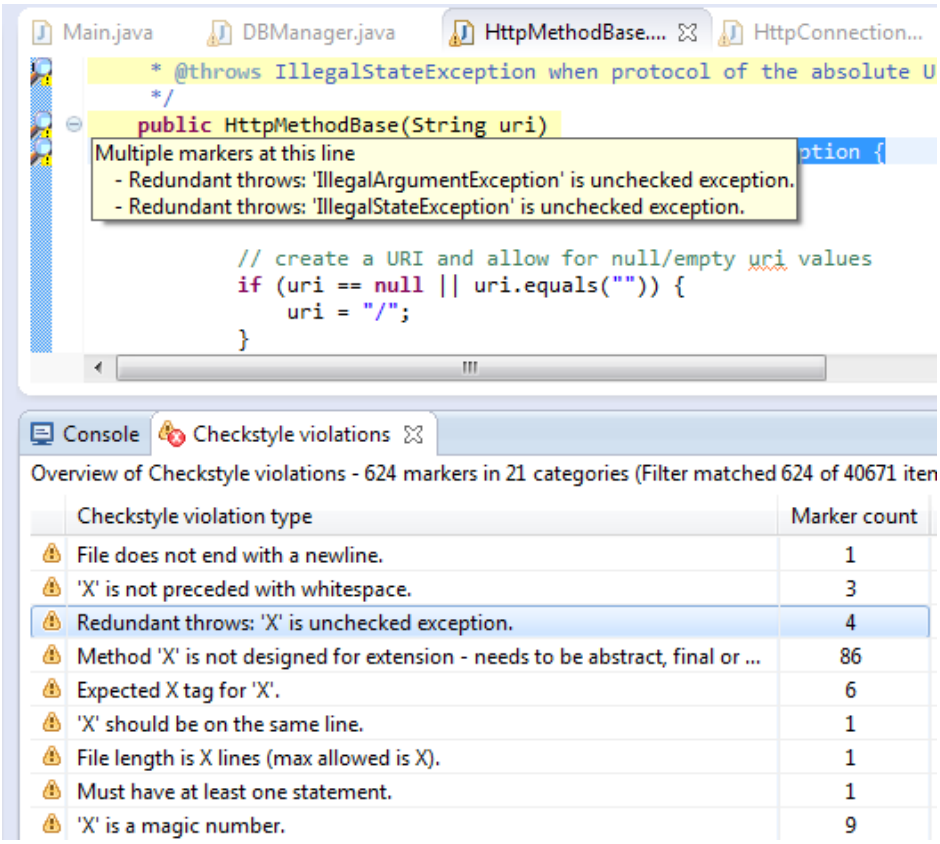
Η διαδικασία ανάλυσης είναι αρκετά απλή. Ο χρήστης επιλέγει το project που επιθυμεί να αναλύσει και από τα properties του έργου ενεργοποιεί την αντίστοιχη επιλογή (checkbox) που βρίσκεται στην καρτέλα (tab) FindBugs. Όπως και στο PMD για το εργαλείο υπάρχει το αντίστοιχο Perspective με τις όψεις Bug Reviews και Bug Info που επιτρέπει την επισκόπηση των αποτελεσμάτων και διευκολύνει την πλοήγηση σε αυτά. Τα αποτελέσματα επίσης επισημάνονται και στον κώδικα στο eclipse με τα σχετικά μηνύματα. Το FindBugs διαθέτει μια φιλική διεπαφή με τον χρήστη παρόμοια με αυτήν του PMD.



Σχήμα 4.9 : Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο FindBugs

Checkstyle

Η διαδικασία ανάλυσης είναι αρκετά απλή. Ο χρήστης επιλέγει το project που επιθυμεί να αναλύσει και από τα properties του έργου ενεργοποιεί την αντίστοιχη επιλογή (checkbox) που βρίσκεται στην καρτέλα (tab) Checkstyle. Τότε υπολογίζονται αυτόματα οι ενεργοποιημένοι κανόνες και εμφανίζονται οι παραβιάσεις στην όψη (view) Checkstyle Violations. Τα λάθη επίσης επισημαίνονται και στον κώδικα στο eclipse.

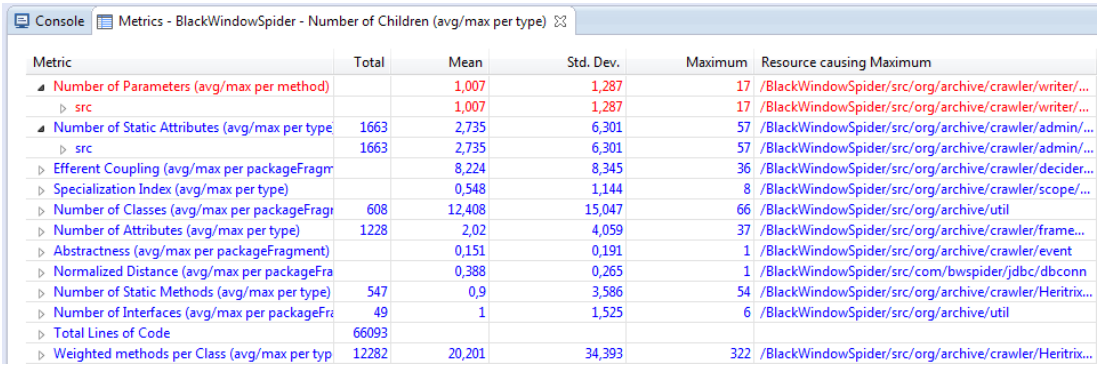


Σχήμα 4.10 : Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο CheckStyle

Το Checkstyle διαθέτει μια φιλική διεπαφή με τον χρήστη παρόμοια με αυτήν του PMD και του FindBugs.

Metrics 1.3.6

Η διαδικασία ανάλυσης είναι αρκετά εύκολη. Ο χρήστης επιλέγει το project που επιθυμεί να αναλύσει και από τα properties του έργου ενεργοποιεί την αντίστοιχη επιλογή (checkbox) που βρίσκεται στην καρτέλα (tab) Metrics. Τα αποτελέσματα εμφανίζονται κάτω από την όψη Metrics (Metrics View).



Σχήμα 4.11: Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων εργαλείο Metrics 1.3.6

Το Metrics 1.3.6 διαθέτει μια φιλική διεπαφή με τον χρήστη παρόμοια με αυτήν του PMD, του FindBugs και του Checkstyle.

Παρατηρούμε ότι πέραν από το εργαλείο ckjm, τα υπόλοιπα εργαλεία διαθέτουν μια φιλική διεπαφή με τον χρήστη.

Ο πιο κάτω πίνακας παρουσιάζει συνοπτικά την μετρική 'Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων'.

Εργαλεία	**Διαδικασία ανάλυσης**	**Διεπαφή με τον χρήστη**
Ckjm	Εύκολη	Κακή
Vil	Εύκολη	Καλή
Sonarqube	Χρονοβόρα	Καλή
NDepend	Εύκολη	Καλή
CppCheck	Εύκολη	Καλή
SourceMonitor	Εύκολη	Καλή
LocMetrics	Εύκολη	Καλή
PMD	Εύκολη	Καλή
FindBugs	Εύκολη	Καλή
Checkstyle	Εύκολη	Καλή
Metrics 1.3.6	Εύκολη	Καλή

Πίνακας 4.3: Μετρική Διεπαφή με τον χρήστη-Έξοδος αποτελεσμάτων

4.3.3 Επεξήγηση αποτελεσμάτων

Τα εργαλεία ckjm, Vil, LocMetrics, SourceMonitor και Metrics 1.3.6, δηλαδή τα εργαλεία που υπολογίζουν μόνο μετρικές ποιότητας κώδικα υπολογίζουν και εμφανίζουν τις τιμές των προϊόντων και δεν παρέχουν περαιτέρω επεξηγήσεις. Σε αυτές τις περιπτώσεις περαιτέρω επεξηγήσεις δεν θεωρούνται αναγκαίες καθώς ο χρήστης μπορεί να κρίνει από τις τιμές των προϊόντων αν ο κώδικας χρίζει διόρθωσης.

Πιο κάτω περιγράφουμε την μετρική 'Επεξήγηση αποτελεσμάτων' για τα υπόλοιπα εργαλεία.

Sonarqube

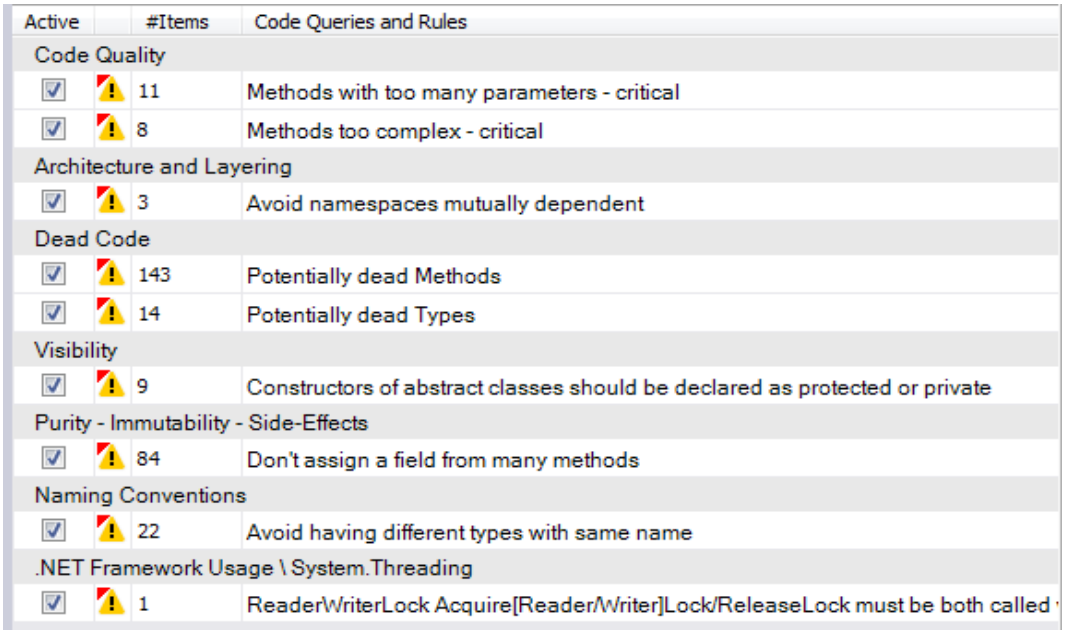
Το Sonarqube παρέχει περιγραφή και επεξηγηματικά σχόλια για κάθε αποτέλεσμα. Κάποιες φορές πέρα από την περιγραφή παρέχει σχετικό παράδειγμα που διευκολύνει την κατανόηση του αποτελέσματος.

Σχήμα 4.12: Επεξήγηση αποτελεσμάτων, Εργαλείο Sonarqube

Τα επεξηγηματικά σχόλια του Sonarqube είναι αναλυτικά και επαρκείς και βοηθούν τον χρήστη να κατανοήσει το αποτέλεσμα.

NDepend

Το NDpend παρέχει περιγραφή και επεξηγηματικά σχόλια για κάθε αποτέλεσμα.

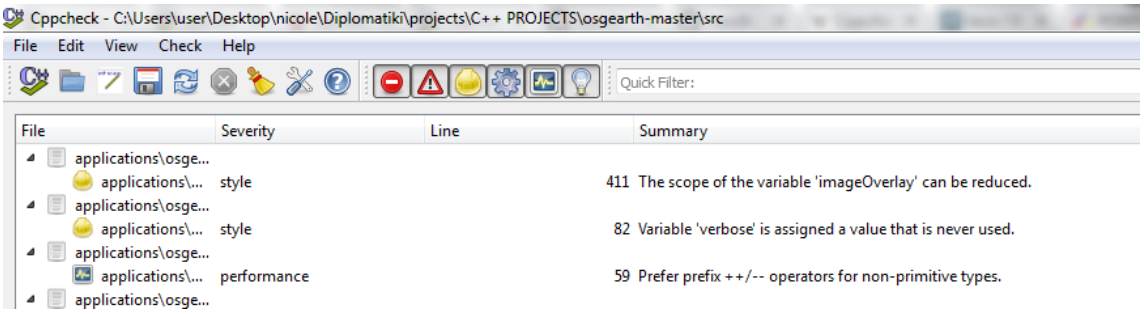


Active	#Items	Code Queries and Rules
Code Quality		
☑	11	Methods with too many parameters - critical
☑	8	Methods too complex - critical
Architecture and Layering		
☑	3	Avoid namespaces mutually dependent
Dead Code		
☑	143	Potentially dead Methods
☑	14	Potentially dead Types
Visibility		
☑	9	Constructors of abstract classes should be declared as protected or private
Purity - Immutability - Side-Effects		
☑	84	Don't assign a field from many methods
Naming Conventions		
☑	22	Avoid having different types with same name
.NET Framework Usage \ System.Threading		
☑	1	ReaderWriterLock Acquire[Reader/Writer]Lock/ReleaseLock must be both called

Σχήμα 4.13: Επεξήγηση αποτελεσμάτων, Εργαλείο NDepend

CppCheck

Το CppCheck πέρα από το μήνυμα (summary) του αποτελέσματος δεν παρέχει κάποια επιπλέον επεξήγηση και έτσι πολλές φορές ο χρήστης πρέπει από μόνος του να κατανοήσει το αποτέλεσμα.



Σχήμα 4.14: Επεξήγηση αποτελεσμάτων, Εργαλείο CppCheck

PMD

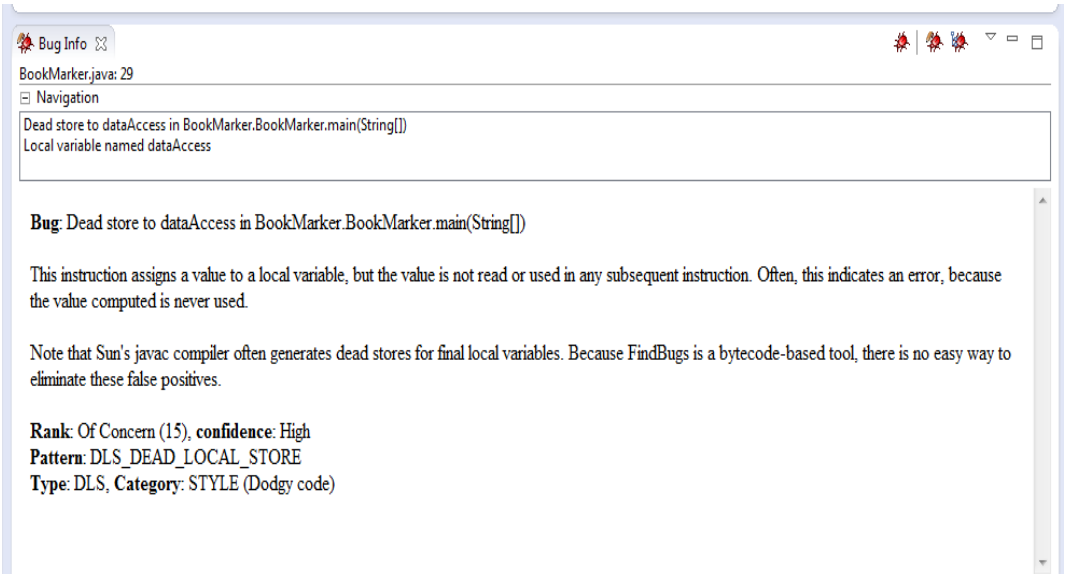
Το μήνυμα λάθους (error message) που παρέχει το εργαλείο PMD είναι επαρκές καθώς ο χρήστης είναι σε θέση να κατανοήσει το λάθος. Υπάρχουν και σχετικά παραδείγματα για την καλύτερη κατανόηση του λάθους.

Priority	Line	created	Rule	Error Message
▶	15...	Thu ...	E...	An empty method in an abstract class should be abstract instead
▶	14...	Thu ...	E...	An empty method in an abstract class should be abstract instead
▶	108	Thu ...	Im...	Private field 'requestHeaders' could be made final; it is only initialized in the declaration or constructor.
▶	521	Thu ...	La...	Potential violation of Law of Demeter (method chain calls)
▶	20...	Thu ...	M...	Parameter 'state' is not assigned and could be declared final

Σχήμα 4.18 Επεξήγηση αποτελεσμάτων, Εργαλείο PMD

FindBugs

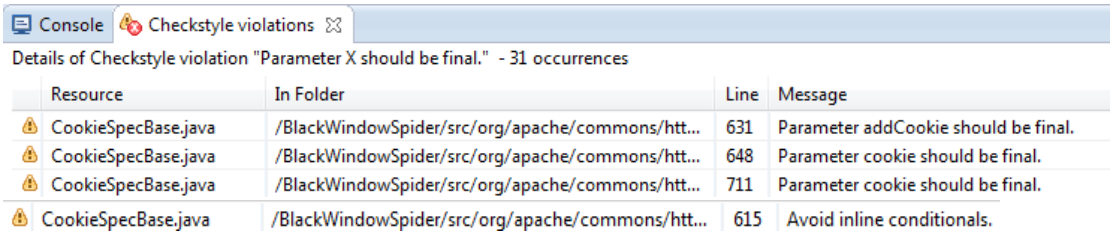
Στην όψη Bug Info ο χρήστης μπορεί να δει την γενική επεξήγηση του αποτελέσματος και να κατανοήσει το αποτέλεσμα.



Σχήμα 4.19: Επεξήγηση αποτελεσμάτων, Εργαλείο FindBugs

Checkstyle

Το μήνυμα (message) στην όψη Checkstyle violations δίνει την περιγραφή του λάθους αλλά δεν επεξηγεί γιατί είναι λάθος.



Σχήμα 4.20: Επεξήγηση αποτελεσμάτων, Εργαλείο Checkstyle

Συνοπτικά ο πιο κάτω πίνακας παρουσιάζει την Μετρική Επεξήγηση αποτελεσμάτων.

Εργαλεία	**Επεξήγηση Αποτελεσμάτων**
	Επαρκής
Ckjm	×
Vil	×
Sonarqube	✓
NDepend	✓
CppCheck	×
SourceMonitor	×
LocMetrics	×
PMD	✓
FindBugs	✓
Checkstyle	×
Metrics 1.3.6	×

Πίνακας 4.4: Μετρική Επεξήγηση αποτελεσμάτων

4.3.4 Εξαγωγή αναφορών

Για την μετρική αυτή εξετάζουμε αν τα εργαλεία εξάγουν αναφορές και στατιστικές για τα αποτελέσματα της ανάλυσης.

Πιο κάτω περιγράφουμε την μετρική ‘Εξαγωγή αναφορών’ για κάθε εργαλείο.

Ckjm

Το εργαλείο ckjm δεν παρέχει την δυνατότητα εξαγωγής αναφοράς των αποτελεσμάτων. Ο χρήστης όμως μπορεί με script να εξάγει μία έξοδο όπως επιθυμεί ο ίδιος (π.χ σε xml μορφή).

Vil

Με την εντολή /out=report ο χρήστης μπορεί να εξάγει τα αποτελέσματα σε ένα αρχείο με την μορφή που έχουν στην γραμμή εντολών.

LOC	WMC	LCOM	NAME
141	14	0	Adler32
144	14	0	Crc32
873	93	96	ZipEntry
210	37	-	InflaterInputStream
867	77	-	ZipInputStream
0	0	-	ZipInputStream+ReaderDelegate
392	47	-	DeflaterOutputStream
1089	74	-	ZipOutputStream

Σχήμα 4.21: Μετρική Εξαγωγή αναφορών, Εργαλείο Vil

Επίσης με την χρήση των αρχείων judge.vjf και vil.css ο χρήστης μπορεί να μορφοποιήσει τα αποτελέσματα σε html μορφή.

Sonarqube

Ο χρήστης αν επιθυμεί την εξαγωγή των αποτελεσμάτων της ανάλυσης μπορεί να κατεβάσει το plugin για την εξαγωγή των αποτελεσμάτων σε μορφή pdf, το οποίο περιγράφει τις λεπτομέρειες της ανάλυσης. Η ανάλυση περιλαμβάνει στατιστικά στοιχεία όπως τους κανόνες που παραβιάστηκαν τις περισσότερες φορές (most violated rules), τα αρχεία που παραβιάστηκαν τις περισσότερες φορές (most violated files), τις πιο πολύπλοκες κλάσεις (most complex classes) και τα αρχεία με κώδικα που επαναλαμβάνεται τις περισσότερες φορές (most duplicated files).

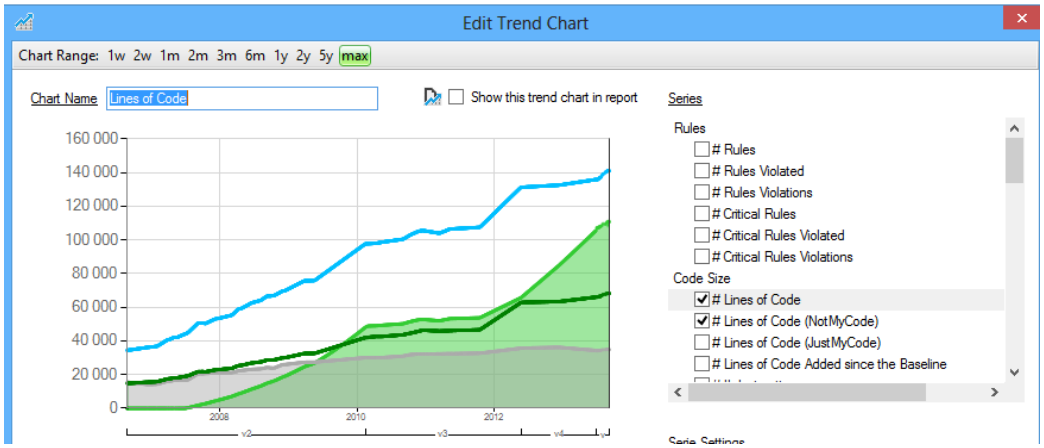
Most violated rules	
Magic Number	157
Malicious code vulnerability - May expose internal representation by returning reference to mutable object	39
Malicious code vulnerability - May expose internal representation by incorporating reference to mutable object	30
Performance - Could be refactored into a named static inner class	30
Empty Block	21

Most violated files	
CPD	24
MetricDef	14
Metric	13
CoreMetrics	13
TrendsChart	12

Σχήμα 4.22: Μετρική Εξαγωγή αναφορών, Εργαλείο Sonarqube

NDepend

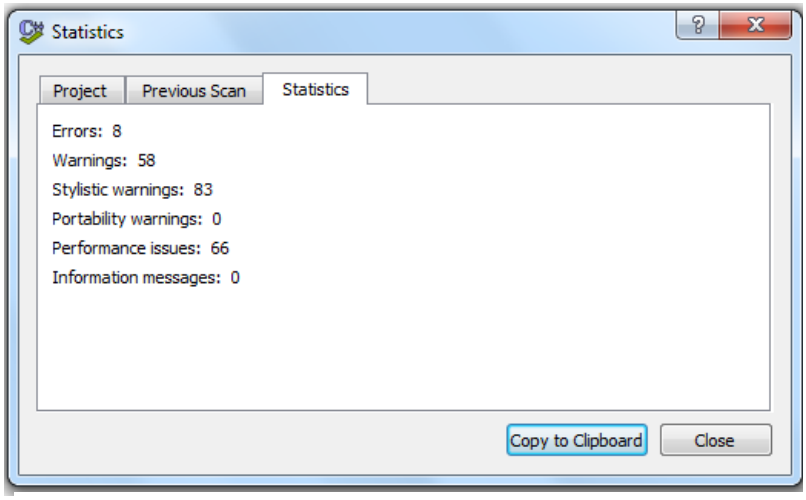
Ο χρήστης μπορεί να εξάγει μία αναφορά σε HTML με όλες τις λεπτομέρειες της ανάλυσης και τα διάφορα διαγράμματα όπως Dependency Graph, Dependency Matrix, Treemap Metric View, Abstractness vs Instability, Trend Charts τα οποία αφορούν εξαρτήσεις μεταξύ των δομών του προγράμματος ή τις μετρικές που υπολογίζονται.



Σχήμα 4.23 : Μετρική Εξαγωγή αναφορών, Εργαλείο Ndepend,Trend Charts

CppCheck

Το CppCheck παρέχει την δυνατότητα εξαγωγής αναφοράς σε μορφή xml,csv,txt. Ο χρήστης μπορεί να αντιγράψει (copy to clipboard) στατιστικά στοιχεία σχετικά με τον αριθμών των παραβάσεων σε κάθε κατηγορία.

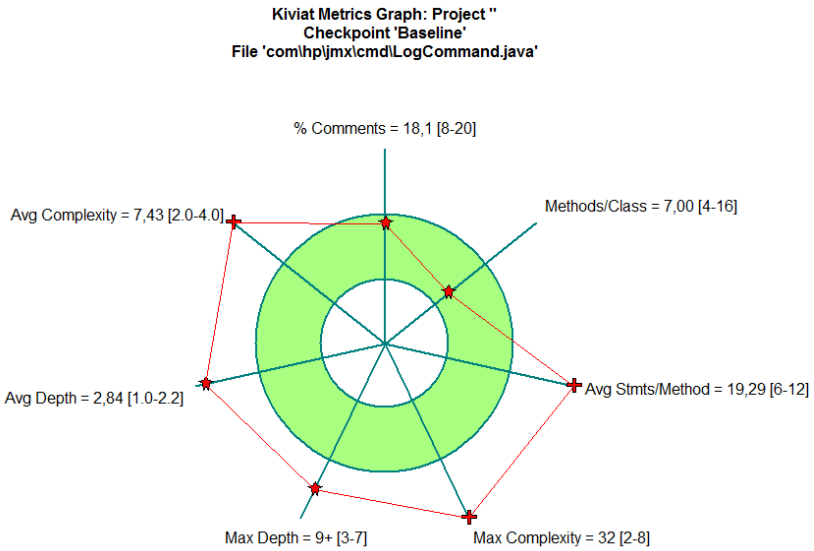


Σχήμα 4.24: Μετρική Εξαγωγή αναφορών, Εργαλείο CppCheck

SourceMonitor

Ο χρήστης μπορεί να εξάγει μια αναφορά σε xml μορφή με τις τιμές των μετρικών.

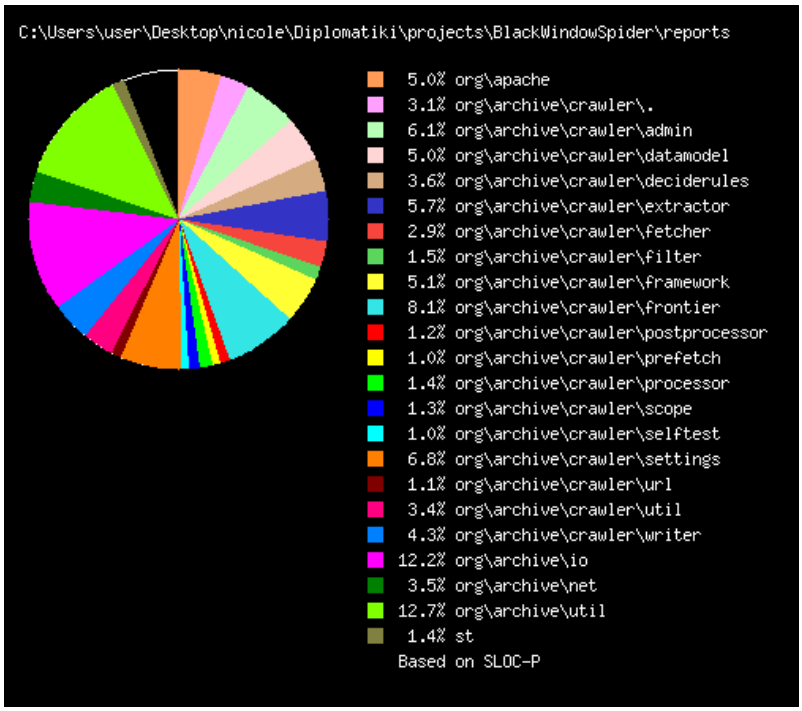
Το SourceMonitor παρουσιάζει κάποια αποτελέσματα με τη βοήθεια γραφημάτων. Στο διάγραμμα Metric Lines παρουσιάζονται οι μεταβολές της μετρικής αριθμός γραμμών (LOC) του έργου. Στο διάγραμμα συχνότητας (Kiviat Metrics graph) παρουσιάζονται οι τιμές των μετρικών για ένα αρχείου ενός checkpoint.



Σχήμα 4.25 : Μετρική εξαγωγή αναφορών, Εργαλείο SourceMonitor, Kiviat Metrics Chart

LocMetrics

Το LocMetrics δίνει την δυνατότητα εξαγωγής των αποτελεσμάτων σε html και csv μορφή. Η αναφορά περιλαμβάνει επίσης ένα διάγραμμα βασισμένο στην μετρική SLOC-P.



Σχήμα 4.26: Μετρική εξαγωγή αναφορών, Εργαλείο LocMetrics, Pie Chart

PMD

Ο χρήστης μπορεί να εξάγει αναφορές (reports) με τις λεπτομέρειες της ανάλυσης στις μορφές html,xml,csv. Στην html μορφή, για κάθε κανόνα που παραβιάζεται (problem) υπάρχει ο αντίστοιχος σύνδεσμος που παραπέμπει στην ιστοσελίδα του pmd με τον ορισμό του κανόνα και σχετικό παράδειγμα επεξήγησης.

PMD report

Problems found

File	Line	Problem
DBManager.java	12	Comment is too large: Too many lines
DBManager.java	27	Found non-transient, non-static member. Please mark as transient or provide accessors.
DBManager.java	27	Private field 'driverName' could be made final; it is only initialized in the declaration or constructor.
DBManager.java	27	fieldCommentRequirement Required
DBManager.java	28	Found non-transient, non-static member. Please mark as transient or provide accessors.
DBManager.java	28	Private field 'dbURL' could be made final; it is only initialized in the declaration or constructor.
DBManager.java	28	fieldCommentRequirement Required
DBManager.java	29	Found non-transient, non-static member. Please mark as transient or provide accessors.
DBManager.java	29	Private field 'userName' could be made final; it is only initialized in the declaration or constructor.

Σχήμα 4.27: Μετρική εξαγωγή αναφορών, Εργαλείο PMD

FindBugs

Ο χρήστης μπορεί να εξάγει αναφορά με τις λεπτομέρειες της ανάλυσης σε XML μορφή.

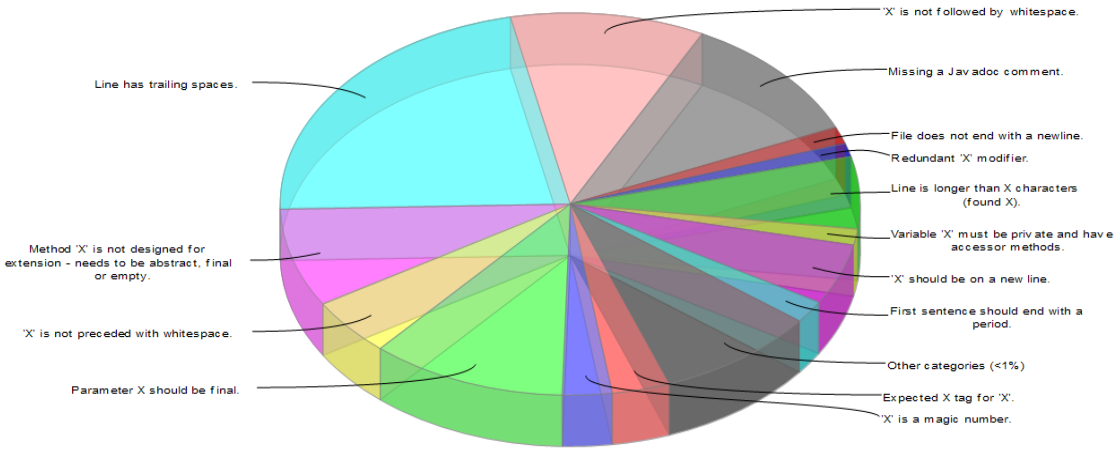
```
BugInstance type="NP_NULL_ON_SOME_PATH" priority="2" abbrev="NP" category="CORRECTNESS" first="1">
 <Class classname="com.bwspider.jdbc.dbconn.DBManager">
   <SourceLine classname="com.bwspider.jdbc.dbconn.DBManager" sourcefile="DBManager.java" sourcepath="com/bwspider/
 </Class>
 <Method classname="com.bwspider.jdbc.dbconn.DBManager" name="close" signature="()V" isStatic="false">
   <SourceLine classname="com.bwspider.jdbc.dbconn.DBManager" start="69" end="79" startBytecode="0" endBytecode="1
 </Method>
 <Field classname="com.bwspider.jdbc.dbconn.DBManager" name="conn" signature="Ljava/sql/Connection;" isStatic="fals
   <SourceLine classname="com.bwspider.jdbc.dbconn.DBManager" sourcefile="DBManager.java" sourcepath="com/bwspider/
 </Field>
 <SourceLine classname="com.bwspider.jdbc.dbconn.DBManager" start="72" end="72" startBytecode="20" endBytecode="20
 <SourceLine classname="com.bwspider.jdbc.dbconn.DBManager" start="69" end="69" startBytecode="4" endBytecode="4" s
/BugInstance>
BugInstance type="SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE" priority="3" abbrev="SQL" category="SECURITY" first="1"
 <Class classname="com.bwspider.jdbc.dbconn.DBManager">
   <SourceLine classname="com.bwspider.jdbc.dbconn.DBManager" sourcefile="DBManager.java" sourcepath="com/bwspider/
 </Class>
```

Σχήμα 4.28ι: Μετρική εξαγωγή αναφορών, Εργαλείο FindBugs

Checkstyle

Το Checkstyle δεν παρέχει την δυνατότητα εξαγωγής αναφοράς.

Διαθέτει γράφημα (pie chart) με τις παραβιάσεις που εντοπίστηκαν.



Σχήμα 4.29 : Μετρική εξαγωγή αναφορών, Εργαλείο Checkstyle, Checkstyle violations

Metrics 1.3.6

Ο χρήστης μπορεί να εξάγει μια αναφορά (report) στη γλώσσα σήμανσης xml με όλες τις λεπτομέρειες της ανάλυσης.

```
<Metric id = "NORM" description ="Number of Overridden Methods">
   <Values per = "type" total = "384" avg = "0,632" stddev = "1,205" max = "12">
      <Value name="ExtractorTagParser" source ="ExtractorSWF.java" package ="org.archive.crawler.extractor" value ="12"/>
      <Value name="RecoverableIOException" source ="RecoverableIOException.java" package ="org.archive.io" value ="10"/>
      <Value name="CachedBdbMap" source ="CachedBdbMap.java" package ="org.archive.util" value ="8"/>
      <Value name="RandomAccessInputStream" source ="RandomAccessInputStream.java" package ="org.archive.io" value ="7"/>
      <Value name="RecordingInputStream" source ="RecordingInputStream.java" package ="org.archive.io" value ="6"/>
      <Value name="GenericObjectPool" source ="GenericObjectPool.java" package ="org.apache.commons.pool.impl" value ="5"/>
```

Σχήμα 4.30 Μετρική εξαγωγή αναφορών, Εργαλείο Metrics 1.3.6

Ο πιο κάτω πίνακας παρουσιάζει συνοπτικά την μετρική Εξαγωγή αναφορών.

Εργαλεία	Εξαγωγή αναφορών	
	Αναφορά	Στατιστικά Διαγράμματα
Ckjm		
Vil	✓	
Sonarqube	✓	✓
NDepend	✓	✓
CppCheck	✓	✓
SourceMonitor	✓	✓
LocMetrics	✓	✓
PMD	✓	
FindBugs	✓	
Checkstyle		✓
Metrics 1.3.6	✓	

Πίνακας 4.5: Μετρική εξαγωγή αναφορών, Εργαλείο Metrics 1.3.6

4.3.5 Δυνατότητα προσαρμογής

Στην μετρική αυτή εξετάζουμε αν ο χρήστης μπορεί να προσαρμόσει τις ρυθμίσεις του εργαλείου. Πιο συγκεκριμένα, εξετάζουμε αν ο χρήστης μπορεί να

- προσαρμόσει την ανάλυση, δηλαδή να ενεργοποιήσει ή να απενεργοποιήσει κανόνες ελέγχους ή μετρικές
- προσαρμόσει την έξοδο των αποτελεσμάτων, δηλαδή να τα ομαδοποιήσει με διάφορα φίλτρα
- συγγράψει δικούς του κανόνες ελέγχου

Πιο κάτω περιγράφουμε τις δυνατότητες προσαρμογής του κάθε εργαλείου.

Τα εργαλεία ckjm, SourceMonitor, Locmetrics δεν παρέχουν στον χρήστη την δυνατότητα προσαρμογής.

Vil

Στο εργαλείο Vil ο χρήστης μπορεί να επιλέξει τις μετρικές ποιότητας κώδικα που επιθυμεί.

Sonarqube

Το Sonarqube δίνει την δυνατότητα στον χρήστη (αφού συνδεθεί ως administrator) να επιλέξει και να καθορίσει ο ίδιος τους κανόνες ελέγχους της ανάλυσης. Επίσης ο συνδεδεμένος χρήστης μπορεί να αλλάξει την κατηγορία κάθε κανόνα, να τον αναθέσει σε άλλον χρήστη, να τον θέσει ως 'confirm' αν είναι σίγουρα λάθος ή ως 'favorite', ή μπορεί να τον θέσει ως 'false positive' (εάν θεωρεί πως είναι) και έτσι οι επόμενοι έλεγχοι να παραβλέψουν τον κανόνα αυτό και συνεπώς τα λανθασμένα αποτελέσματα (false positives). Ο χρήστης μπορεί να γράψει νέους κανόνες στην γλώσσα java για τα μόνο για τις γλώσσες Java και COBOL. Παράλληλα με την γλώσσα XPath ο χρήστης μπορεί να γράψει νέους κανόνες στις γλώσσες C/C++, C#, COBOL, Flex, Java, JavaScript, PL/I, PL/SQL, Python και VB.NET.

Ένα σημαντικό προτέρημα του εργαλείου είναι ότι 'κρατά' το ιστορικό της ανάλυσης για κάποιο χρονικό διάστημα και ο χρήστης όταν ενημερώσει τον κώδικα του μπορεί να δει τις αλλαγές που έγιναν στην ανάλυση με την ενημέρωση του κώδικα.

Παρέχει επίσης και πολλές άλλες δυνατότητες προσαρμογής όπως, ο χρήστης μπορεί να καθορίσει την μορφή και την εμφάνιση του dashboard προσθέτοντας ή αφαιρώντας τα widget που επιθυμεί ή μπορεί να θέσει το dashboard ως 'favorite', κλπ.

Ndepend

Ο χρήστης μπορεί να επιλέξει τους κανόνες ελέγχου που επιθυμεί για την ανάλυση. Το Ndepend παρέχει επίσης στο χρήστη την δυνατότητα να γράψει τους δικούς του κανόνες ελέγχου με την τεχνική Code Query Language (CQLink).

Παρέχει ακόμα και άλλες δυνατότητες προσαρμογής όπως η σύγκριση του ίδιου έργου πριν και μετά από κάποια ενημέρωση του κώδικα, η μορφοποίηση του report, , κλπ.

CppCheck

Ο χρήστης μπορεί να επιλέξει ποιες κατηγορίες (severity) λάθους θα εμφανίζονται και να γράψει τους δικούς του κανόνες ελέγχου μέσω της γλώσσας xml.

PMD

Ο χρήστης μπορεί να ενεργοποιήσει τους κανόνες ελέγχου που επιθυμεί να υπολογιστούν και να τα ομαδοιποιήσει ανάλογα με το βαθμό προτεραιότητας τους.

Επίσης το PMD δίνει τη δυνατότητα οι υψηλής προτεραιότητας παραβιάσεις να θεωρούνται σφάλματα του Eclipse, να απαιτείται δηλαδή η διόρθωση τους για να εκτελεστεί το πρόγραμμα., τον καθορισμό working set δηλαδή έργα που ακολουθούν τις ρυθμίσεις που καθορίζονται καθώς επίσης και την δυνατότητα ο χρήστης να γράψει τους δικούς του κανόνες χρησιμοποιώντας Xpath ή Java κλάσεις.

FindBugs

Ο χρήστης μπορεί να ενεργοποιήσει τους κανόνες που επιθυμεί για ανάλυση, να επιλέξει τις κατηγορίες σφαλμάτων και την προτεραιότητα (confidence) των σφαλμάτων που θα εμφανίζονται και να ομαδοποιήσει τα αποτελέσματα της ανάλυσης με βάση:

1. Το bugRank, που αποτελείται από 4 βαθμούς (rank) κρισιμότητας που είναι αντίστοιχοι της κρισιμότητας του λάθους. Scariest, scary, troubling, of concern.
2. Το confidence (high, normal, low) του σφάλματος. Δηλαδή, πόσο αξιόπιστο είναι το λάθος που εντοπίζεται. Αν το confidence είναι high, αυτό δηλώνει ότι το λάθος που εντοπίστηκε είναι σωστό (true positive). Απενεργοποιώντας την εμφάνιση σφαλμάτων με low confidence τότε μειώνονται και οι πιθανότητες η ανάλυση να επιστρέψει λανθασμένα αποτελέσματα (false positives).
3. Την κατηγορία του σφάλματος.
4. Το μοτίβο που ανήκει το σφάλμα (pattern type).
5. Το σφάλμα.

Μπορεί επίσης να καθορίσει αρχεία φίλτρων σε XML μορφή τα οποία μπορούν να χρησιμοποιηθούν για να συμπεριληφθούν ή να αποκλειστούν στην αναφορά σφάλματα που αφορούν συγκεκριμένες μεθόδους ή κλάσεις ή ακολουθούν συγκεκριμένα πρότυπα σφαλμάτων κλπ, και να ρυθμίσει το επίπεδο προσπάθειας της ανάλυσης (analysis effort) μεταξύ τριών τιμών (Default, Minimal Maximal).

Checkstyle

Ο χρήστης μπορεί να επιλέξει και να ρυθμίσει τους ελέγχους που επιθυμεί δημιουργώντας το δικό του configuration. Μπορεί να επιλέξει την κατηγορία ενός κανόνα (error, warning, info), τα ελάχιστα και τα μέγιστα όρια του αν υπάρχουν και

άλλες ρυθμίσεις που αντιστοιχούν στους κανόνες. Επίσης μπορεί να ομαδοποιήσει τα αποτελέσματα ανάλογα με την κατηγορία που ανήκουν.

Metrics 1.3.6

Ο χρήστης μπορεί να καθορίσει τις ελάχιστες και μέγιστες τιμές (safe range) των μετρικών.

Ο πιο κάτω πίνακας παρουσιάζει συνοπτικά την μετρική Δυνατότητα Προσαρμογής.

Εργαλεία	**Δυνατότητα Προσαρμογής**		
	Ενεργοποίηση Κανόνων	**Ομαδοποίηση Αποτελεσμάτων**	**Συγραφή νέων κανόνων**
Ckjm			
Vil	✓		
Sonarqube	✓	✓	✓
NDepend	✓	✓	✓
CppCheck		✓	✓
SourceMonitor			
LocMetrics			
PMD	✓	✓	✓
FindBugs	✓	✓	✓
Checkstyle		✓	
Metrics 1.3.6			

Πίνακας 4.6: Μετρική δυνατότητα Προσαρμογής

4.3.6 Κοινοποίηση αποτελεσμάτων

Το εργαλείο Findbugs παρέχει την δυνατότητα κοινοποίησης των αποτελεσμάτων μέσω ενός cloud server είτε δημόσια σε όλους τους χρήστες που είναι συνδεδεμένοι στον cloud server, είτε ιδιωτικά (δημιουργούντας κάποια congifuration αρχεία). Ο χρήστης αφού συνδεθεί στο δίκτυο μπορεί να δει τα δημόσια κοινοποιημένα αποτελέσματα και να τα σχολιάσει, να τα αναθέσει στον εαυτό του για διόρθωση (I will fix) ή να το

κατηγοριοποιήσει ανάμεσα στις κατηγορίες ‘needs further study, not a bug, mostly harmless, should fix, must fix, bad analysis, unused’.

Το Sonarqube παρέχει επίσης την δυνατότητα κοινοποίησης των αποτελεσμάτων της ανάλυσης. Ο χρήστης αφού συνδεθεί ως διαχειριστής (administrator) μπορεί να επιτρέψει σε χρήστες να συνδεθούν στον Sonarqube λογαριασμό του (Allow users to sign up online) ή μπορεί να προσθέσει ο ίδιος νέους χρήστες (add new user). Μπορεί επίσης να καθορίσει ομάδες (add new group) από συγκεκριμένους χρήστες. Ο διαχειριστής μπορεί να δώσει στους χρήστες συγκεκριμένα δικαιώματα (permissions) και να τους αναθέσει αποτελέσματα για επίλυση.
Μερικά από τα δικαιώματα που μπορεί να δώσει ο διαχειριστής σε έναν χρήστη ή μια ομάδα είναι τα εξής:

- πρόσβαση στο dashboard της ανάλυσης, στις μετρήσεις και στα αποτελέσματα (issues)
- πρόσβαση στον πηγαίο κώδικα
- επισήμανση αποτελεσμάτων ως ‘false positive’ ή αλλαγή της κατηγορίας του (severity)
- πρόσβαση στις ρυθμίσεις του έργου
- καθορισμός συγκεκριμένου χρήστη για την διαχείριση του έργου

Ο χρήστης μπορεί επίσης να σχολιάσει (comment) ένα αποτέλεσμα.

4.3.7 Μετρικές ποιότητας κώδικα

Για τον έλεγχο των μετρικών προϊόντων των εργαλείων ομαδοποιήσαμε τα εργαλεία ανάλογα με την γλώσσα προγραμματισμού που υποστηρίζουν και πραγματοποιήσαμε επιμέρους συγκρίσεις για κάθε ‘ομάδα’ εργαλείων. Δηλαδή κατηγοριοποιήσαμε τα εργαλεία στις ομάδες Java, Php, C, C++ και C#. Πιο συγκεκριμένα ελέγξαμε και καταγράψαμε τις τιμές των μετρικών που υπολογίζει το κάθε εργαλείο για τα έργα λογισμικού που είναι υλοποιημένα στην γλώσσα προγραμματισμού που υποστηρίζει το εργαλείο. Στην συνέχεια ελέγξαμε αν οι τιμές των μετρικών που επιστρέφουν τα εργαλεία είναι κοινές.

4.3.7.1 Εργαλεία γλώσσας προγραμματισμού Java

Οι πιο κάτω πίνακες αναπαριστούν τα αποτελέσματα των μετρικών για τα έργα λογισμικού Accessor [1], BlackWindowSpider [5] που είναι υλοποιημένα στην γλώσσα προγραμματισμού Java. Για τα εργαλεία Sonarqube, PMD και Checkstyle ενεργοποιήσαμε τους κανόνες που υπολογίζουν τις μετρικές αυτές. Το εργαλείο FindBugs δεν υπολογίζει τις μετρικές.

Εργαλεία	**Μετρικές**						
	LOC/LLOC	**NOM**	**CC Avg CC/method**	**WMC**	**DIT Avg Dit/type**	**LCOM**	**NOC**
Ckjm				531	1,30	79644	13
Sonarqube	9793/5771	531	2,7	1465			
SourceMonitor	9792		2,025				
LocMetrics	9793			1029			
PMD			2				
FindBugs							
Checkstyle			2				
Metrics 1.3.6	/5771	402	2,025	1069	1,33	0,218	13

Πίνακας 4.7: Μετρικές Προϊόντος για το έργο λογισμικού Accessor

Εργαλεία	**Μετρικές**						
	LOC/LLOC	**NOM**	**CC Avg CC/method**	**WMC**	**DIT Avg Dit/type**	**LCOM**	**NOC**
Ckjm				5409	3,33	79931	324
Sonarqube	129392 /66093	5409	2,7	14364			
SourceMonitor	129391		2,28				
LocMetrics	129392			8295			
PMD			2				
FindBugs							
Checkstyle			2				
Metrics 1.3.6	/66093	4841	2,28	12282	3,35	0,296	324

Πίνακας 4.8: Μετρικές Προϊόντος για το έργο λογισμικού BlackWindowSpider

Όπως βλέπουμε και στους πίνακες 4.7 και 4.8 πέραν από το εργαλείο Metrics 1.3.6, τα υπόλοιπα δεν υπολογίζουν όλες τις μετρικές. Τα εργαλεία PMD και Checkstyle επιστρέφουν αποτελέσματα μόνο στις περιπτώσεις που η τιμή της μετρικής ξεπερνά το επιτρεπτό όριο. Επίσης υπάρχουν διαφορές στους υπολογισμούς των μετρικών μεταξύ των εργαλείων. Στην προσπάθεια μας να εξηγήσουμε τους λόγους για τους οποίους υπάρχουν διαφορές περιγράφουμε πιο κάτω τα αποτελέσματα των εργαλείων για μία συγκεκριμένη κλάση, την κλάση Handler του έργου BlackWindowSpider.

Για την μετρική LOC (Lines of code) το εργαλείο Metrics 1.3.6 υπολογίζει 110. Σε αυτή την περίπτωση το εργαλείο υπολογίζει τις εκτελέσιμες γραμμές του κώδικα (LLOC). Το Sonarqube υπολογίζει επίσης 110 για την μετρική LLOC και 156 για την μετρική LOC. Το LocMetrics υπολογίζει 156 και το SourceMonitor 155 για την LOC μετρική. Το SourceMonitor υπολογίζει τις γραμμές του κώδικα ξεκινώντας από το 0.

Για την μετρική NOM (Number of methods) το Metrics 1.3.6 υπολογίζει 1 και το Sonarqube υπολογίζει 2. Ο λόγος είναι ότι το Metrics 1.3.6 δεν συμπεριλαμβάνει την main (και τους κατασκευαστές). Το ckjm υπολογίζει μόνο τις δημόσιες (public) μεθόδους (1).

Για την μετρική CC (Cyclomatic complexity) το Metrics 1.3.6, PMD, Checkstyle και SourceMetric υπολογίζει 7 μέσο όρο (CC/μέδοθους) και το Sonarqube 10.5. Το Sonarqube σε αντίθεση με τα εργαλεία Metrics 1.3.6, PMD, Checkstyle αυξάνει την τιμή της κυκλωματικής πολυπλοκότητας όταν εντοπίζει στον κώδικα throw exceptions και εντολές return που δεν είναι στο τέλος κάθε μεθόδου.
Για την καλύτερη κατανόηση της μέτρησης της κυκλωματικής πολυπλοκότητας από τα εργαλεία στατικής ανάλυσης κώδικα παρουσιάζουμε πιο κάτω ενά τμήμα κώδικα της μεθόδου openConnection (URL u) της κλάσης Handler. Στην περίπτωση αυτή το Sonarqube θα υπολογίσει την τιμή της κυκλωματικής πολυπλοκότητας 7 και τα εργαλεία Metrics 1.3.6, PMD, Checkstyle και SourceMetric 6.

```
protected URLConnection openConnection(URL u)
   throws IOException {          //+1
      String accessKey = null;
      String secretAccessKey = null;
      String userInfo = u.getUserInfo();
      if (userInfo != null) {        //+1
         int index = userInfo.indexOf(':');
         if (index != -1) {          //+1
           accessKey = userInfo.substring(0, index);
           secretAccessKey = userInfo.substring(index + 1);
         } else {
           accessKey = userInfo;
         }
      }
      if (accessKey == null) {     //+1
       accessKey = System.getProperty("aws.access.key.id");
      }
      if (secretAccessKey == null) {          //+1
       secretAccessKey = System.getProperty("aws.access.key.secret");
      }
      if (accessKey == null && secretAccessKey == null) {       //+2
       throw new IllegalArgumentException("AWS " +
            "Access Key ID and Secret Access Key " +
            "must be specified as the username " +
            "or password (respectively) of a s3 URL, " +
            "or by setting the " +
            "aws.access.key.id or " +
            "aws.access.key.secret properties (respectively).");
      }
```

Σχήμα 4.31: Μετρική CC, εργαλείο Sonarqube,τμήμα κώδικα μεθόδου openConnection (URL u)

Όπως και στην μετρική CC, το εργαλείο Metrics 1.3.6 υπολογίζει την μετρική WMC (Weighted methods per class) 14, το Sonarqube 21. Το ckjm υπολογίζει 2 καθώς υπολογίζει μόνο το άθροισμα των μεθόδων (NOM). Για λόγους που δεν διευκρηνήσαμε το εργαλείο LocMetrics υπολογίζει 9.

Για την μετρική DIT (Depth of inheritance tree) το Metrics 1.3. και το ckjm 6 υπολογίζουν 2, καθώς κληρονομούν από την κλάση URLStreamHandler και από την κλάση Object.

Το εργαλείο Metrics 1.3.6 υπολογίζει την $3^{η}$ εκδοχή (LCOM3) και η τιμή της είναι 0, και το εργαλείο ckjm υπολογίζει την $1^{η}$ εκδοχή (LCOM1) και η τιμή της είναι 3.

Για την μετρική NOC (Number of children) τα εργαλεία Metrics 1.3.6 και ckjm υπολογίζουν 0.

4.3.7.2 Εργαλεία γλώσσας προγραμματισμού C

Οι πιο κάτω πίνακες αναπαριστούν τα αποτελέσματα των μετρικών για τα έργα λογισμικού Libgit2 [22] και Redis [30] που είναι υλοποιημένα στην γλώσσα προγραμματισμού C. Το CppCheck δεν υπολογίζει μετρικές.

Εργαλεία	**Μετρικές**			
	LOC/LLOC	**Functions**	**CC Avg CC/method**	**NBD**
CppCheck				
SourceMonitor	82814	2712	4,98	1,43

Πίνακας 4.9 Μετρικές Προϊόντος για το έργο λογισμικού Libgit2

Εργαλεία	**Μετρικές**			
	LOC/LLOC	**Functions**	**CC Avg CC/method**	**NBD**
CppCheck				
SourceMonitor	51956	1373	6.60	1.78

Πίνακας 4.10: Μετρικές Προϊόντος για το έργο λογισμικού Redis

3.7.3 Εργαλεία γλώσσας προγραμματισμού C++

Οι πιο κάτω πίνακες αναπαριστούν τα αποτελέσματα των μετρικών για τα έργα λογισμικού Remote Control Center [31] και K-Meleon [21] που είναι υλοποιημένα στην γλώσσα προγραμματισμού C++. Το CppCheck δεν υπολογίζει μετρικές.

Εργαλεία	**Μετρικές**				
	LOC/LLOC	**Functions**	**CC Avg CC/method**	**WMC**	**NBD**
CppCheck					
SourceMonitor	15684	21	4.05		1.49
LocMetrics	15749/10855			1356	

Πίνακας 4.11: Μετρικές Προϊόντος για το έργο λογισμικού Remote Control Center

Εργαλεία	Μετρικές				
	LOC/LLOC	**Functions**	**CC Avg CC/method**	**WMC**	**NBD**
CppCheck					
SourceMonitor	73264	753	4,87		2.03
LocMetrics	73440/51785			8415	

Πίνακας 4.12: Μετρικές Προϊόντος για το έργο λογισμικού K-Meleon

Για την μετρική LOC το SourceMonitor σε κάθε αρχείο ξεκινά την αρίθμηση από το 0.

4.3.7.3 Εργαλεία γλώσσας προγραμματισμού C#

Ο πιο κάτω πίνακας αναπαριστά τα αποτελέσματα των μετρικών για τα έργο λογισμικού DotNetOpenAuth [9].

Εργαλεία	Μετρικές			
	LOC/LLOC	**CC Avg CC/method**	**WMC**	**NBD**
Vil	4083	2	2775	2
SourceMonitor	4057	2.19		2.06
LocMetrics	4083		2775	
NDepend	4083	2,19	2775	2

Πίνακας 4.13: Μετρικές Προϊόντος για το έργο λογισμικού DotNetOpenAuth

4.3.7.5 Εργαλεία γλώσσας προγραμματισμού PHP

Οι πιο κάτω πίνακες αναπαριστούν τα αποτελέσματα των μετρικών για τα έργα λογισμικού PhpExcel [29] και Mustache [26] που είναι υλοποιημένα στην γλώσσα προγραμματισμού PHP.

Εργαλεία	Μετρικές				
	LOC/LLOC	**Statements**	**NOM**	**CC Avg CC/method**	**WMC**
Sonarqube	98199/50784	31486	2721	5,4	14816

Πίνακας 4.15: Μετρικές Προϊόντος για το έργο λογισμικού PhpExcel

Εργαλεία	Μετρικές				
	LOC/LLOC	**Statements**	**NOM**	**CC Avg CC/method**	**WMC**
Sonarqube	3874/1613	697	157	2,6	401

Πίνακας 4.16: Μετρικές Προϊόντος για το έργο λογισμικού Mustache

4.3.8 Αξιοπιστία αποτελεσμάτων

Για τον αξιολόγηση της μετρικής 'Αξιοπιστία αποτελεσμάτων' συγκρίναμε τα εργαλεία που εφαρμόζουν κανόνες ελέγχου κατά την ανάλυση και ακολουθήσαμε την ίδια μεθοδολογία που ακολουθήσαμε και για την αξιολόγηση των μετρικών προϊόντος. Δηλαδή συγκρίναμε τα εργαλεία με βάση την γλώσσα προγραμματισμού που υποστηρίζουν. Πραγματοποιήσαμε 5 επιμέρους συγκρίσεις, μια για κάθε 'ομάδα εργαλείων'. Για κάθε ομάδα αναλύσαμε τα αντίστοιχα έργα λογισμικού, καταγράψαμε τα αποτελέσματα της ανάλυσης και ακολούθως τα συγκρίναμε.

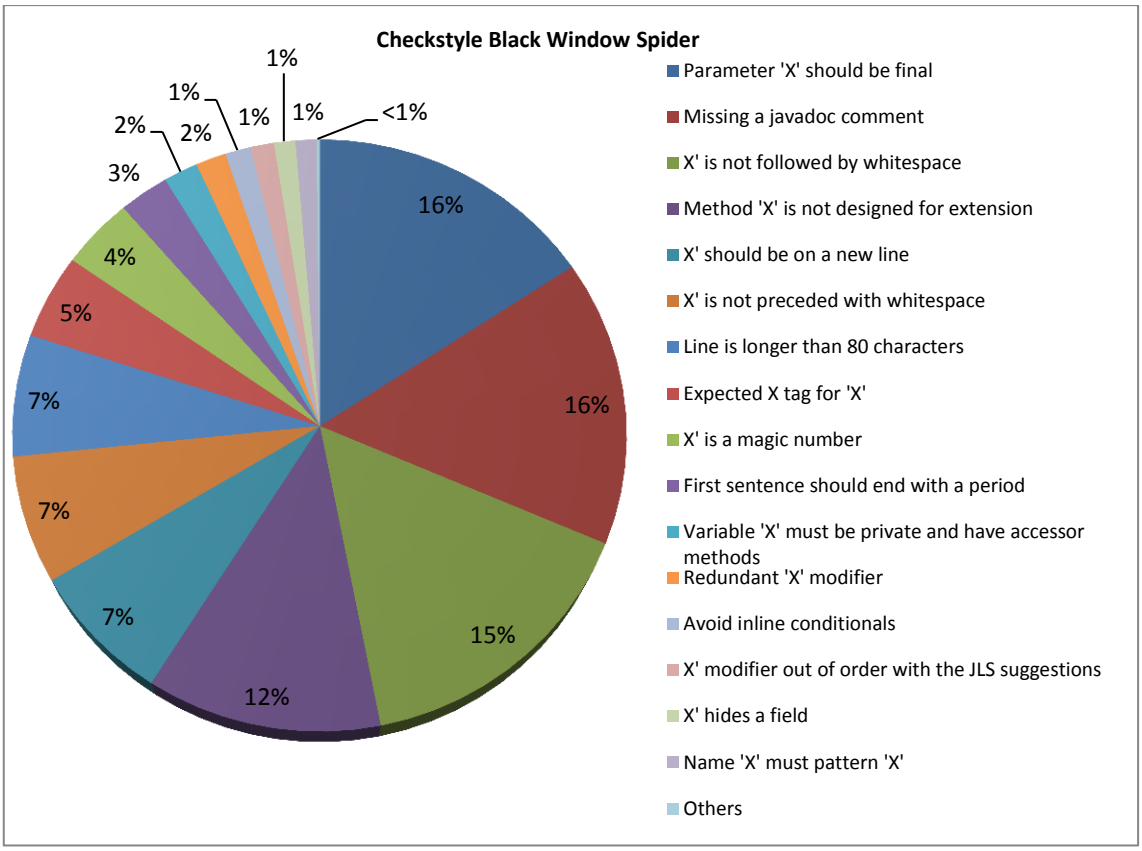
4.3.8.1 Εργαλεία γλώσσας προγραμματισμού Java

Στο σημείο αυτό θα συγκρίνουμε τα εργαλεία που ανήκουν στην ομάδα Java για τα έργα λογισμικού BlackWindowSpider και Accessor.

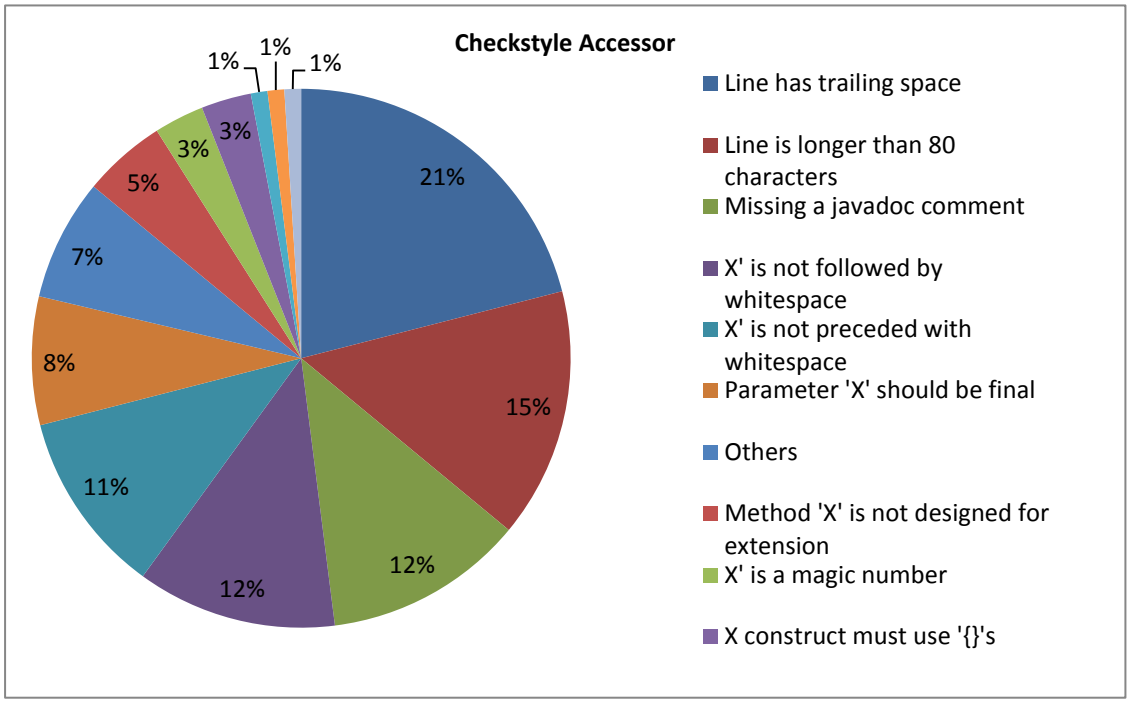
Checkstyle

Με όλους τους κανόνες ενεργοποιημένους το CheckStyle εντόπισε 29850 παραβιάσεις (violations) για το έργο λογισμικού BlackWindowSpider, μεταξύ των οποίων 16% αφορούν παραμέτρους σε μεθόδους και κατασκευαστές που θα έπρεπε να είναι final έτσι ώστε να μην υπάρχει περίπτωση να αλλάξει η τιμή της μέσα στην μέθοδο (Parameter 'X' should be final), 16% αφορούν την χρήση javadoc σχόλιων (Missing javadoc comment), 15% αφορούν την έλλειψη του κενού χαρακτήρα μετά από συγκεκριμένους τελεστές και εντολές ('X' is not followed by whitespace), 12% αφορούν μεθόδους που πρέπει να είναι abstract ή final ή κενές καθώς δεν είναι σχεδιασμένες για επέκταση (Method 'X' is not designed for extension) και οι υπόλοιπες παραβιάσεις είναι περισσότερο στιλιστικές παραβιάσεις.

Για το έργο Accessor εντόπισε 6043 παραβιάσεις (violations). Το 21% αφορούν γραμμές κώδικα οι οποίες μπορούν να γραφτούν ως κανονικές εκφράσεις (Line has trailing spaces), 15% αφορούν πολύ μεγάλες γραμμές (Line is longer than 80 characters), 12% αφορούν την έλλειψη του κενού χαρακτήρα μετά από συγκεκριμένους τελεστές και εντολές ('X' is not followed by whitespace), 12% αφορούν την χρήση javadoc σχόλιων (Missing javadoc comment) και οι υπόλοιπες παραβιάσεις είναι περισσότερο στιλιστικές παραβιάσεις.



Σχήμα 4.31: Αποτέλεσματα Checkstyle, έργο λογισμικού BlackWindowSpider



Σχήμα 4.32: Αποτέλεσματα Checkstyle, έργο λογισμικού Accessor

Αφού αναλύσαμε το έργο με όλους τους ελέγχους, στην συνέχεια επιλέξαμε 11 κανόνες ελέγχου (πίνακας 4.17) που ανήκουν στην κατηγορία ‘Coding problems’ για να εντοπίσουμε πιθανά λάθη του κώδικα.

Για το έργο BlackWindowSpider το Checkstyle εντόπισε 2133 παραβιάσεις. Το 52% είναι παραβιάσεις που αφορούν τον κανόνα ‘X is a magic number’. Το 18% αφορούν τον κανόνα ‘Avoid inline conditionals’, το 15% αφορούν τον κανόνα ‘X hides a field’ και το υπόλοιπο ποσοστό (15%) αφορούν τους υπόλοιπους κανόνες.

Για το έργο Accessor εντόπισε 280 παραβιάσεις. Το 56% είναι παραβιάσεις που αφορούν τον κανόνα ‘X is a magic number’. Το 24% αφορούν τον κανόνα ‘X hides a field’, το 9% αφορούν τον κανόνα ‘Nested If Depth’, το 8% αφορούν τον κανόνα ‘Avoid inline conditionals’ και το υπόλοιπο ποσοστό (3%) αφορούν τους υπόλοιπους κανόνες.

Παρατηρούμε ότι οι παραβιάσεις είναι εύστοχες (true positives) καθώς η επιδιόρθωση τους διευκολύνει την ανάγνωση και την συντήρηση του κώδικα ωστόσο είναι προειδοποιήσεις (warnings) που δεν θα οδηγήσουν σε πιθανά σοβαρά λάθη κατά την εκτέλεση.

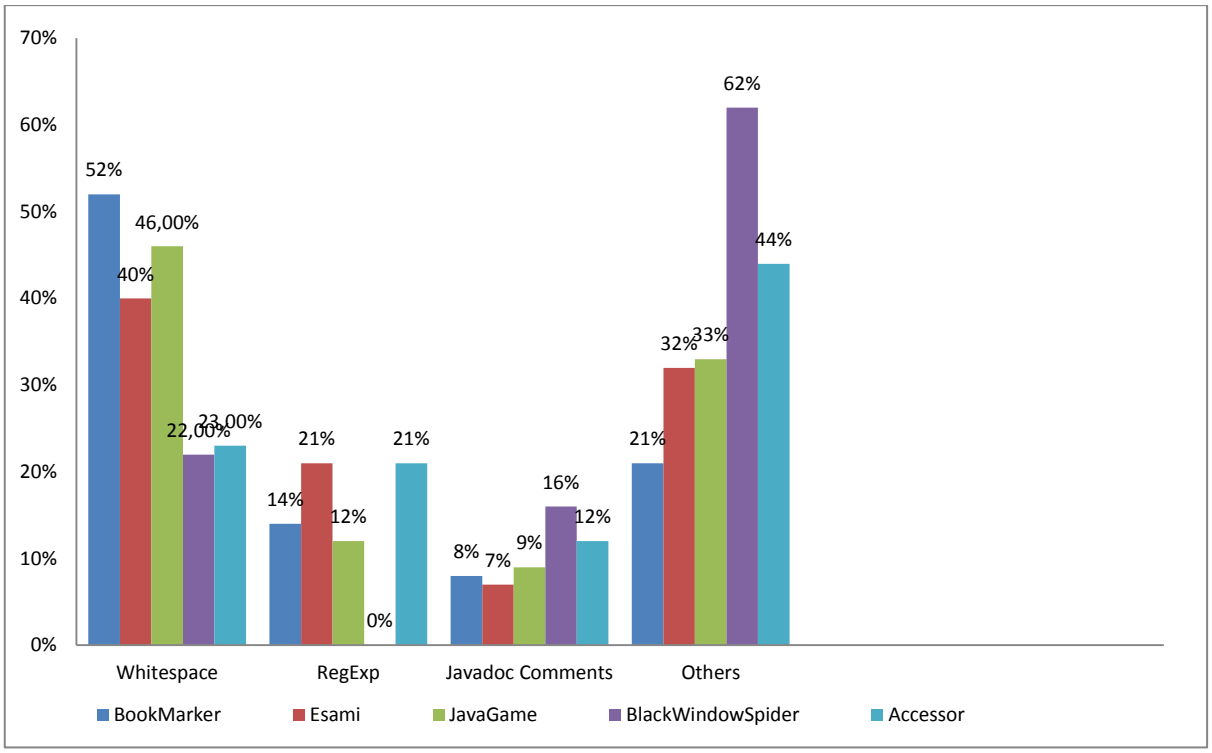
Κανόνας	Περιγραφή
1. Avoid inline conditionals	Οι inline συνθήκες δυσκολέουν την ανάγνωση του κώδικα Παράδειγμα inline συνθήκης String a = getParameter("a"); String b = (a==null || a.length<1) ? null : a.substring(1);
2. Avoid empty statements	Έλεγχος για κενές εντολές
3. Equals and HashCode	Έλεγχος ότι οι κλάσεις που υποσκελίζουν την μέθοδο equals() πρέπει να υποσκελίζουν και την μέθοδο hashCode() αφού ίσα αντικείμενα πρέπει να έχουν το ίδιο hashCode
4. Hidden field	Έλεγχος ότι μια τοπική μεταβλητή ή παράμετρος δεν επισκιάζει κάποιο πεδίο που ορίζεται στην ίδια κλάση
5. Inner assignment	Οι εσωτερικές αναθέσεις δυσκολεύουν την ανάγνωση του κώδικα Παράδειγμα inline ανάθεσης String s = Integer.toString(i = 2);
6. Magic Number	Αριθμοί που δεν δηλώνονται ως σταθερές δυσκολεύουν την ενημέρωση του κώδικα
7. Missing switch default	Έλεγχος ότι κάθε switch συνθήκη ορίζει και default περίπτωση.
8. Simplify Boolean Expression	Πολύπλοκες τύπου Boolean εκφράσεις δυσκολεύουν την ανάγνωση και την συντήρηση του κώδικα

9. Simplify Boolean Return	Έλεγχος για πολύπλοκες τύπου Boolean return δηλώσεις
	Ο κώδικας αυτός
	if (valid())
	  return false;
	else
	  return true;
	είναι προτιμότερο να γραφτεί
	  return !valid();

10. Nested For Depth	Το βάθος των φωλιασμένων for εντολών πρέπει να είναι μικρό.
11. Nested If Depth	Το βάθος των φωλιασμένων if εντολών πρέπει να είναι μικρό.

Πίνακας 4.17: Checkstyle, κανόνες Coding Problems

Για όλα τα έργα λογισμικού το Checkstyle εντόπισε προειδοποιήσεις (warnings) σχετικές με το στυλ και την σύνταξη του κώδικα.



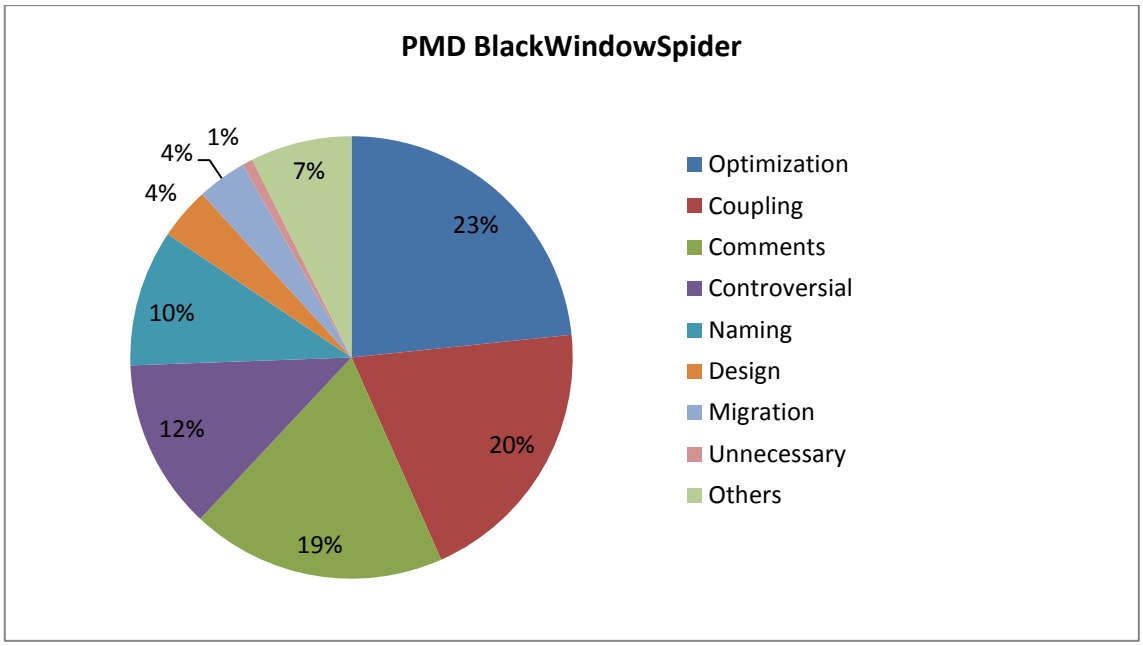
Σχήμα 4.33: Εργαλείο Checkstyle, έργα λογισμικού

Το Checkstyle απευθύνεται σε χρήστες που επιθυμούν ο κώδικας τους να ακολουθεί μια συγκεκριμένη δομή όσον αφορά το στυλ και την εμφάνιση.

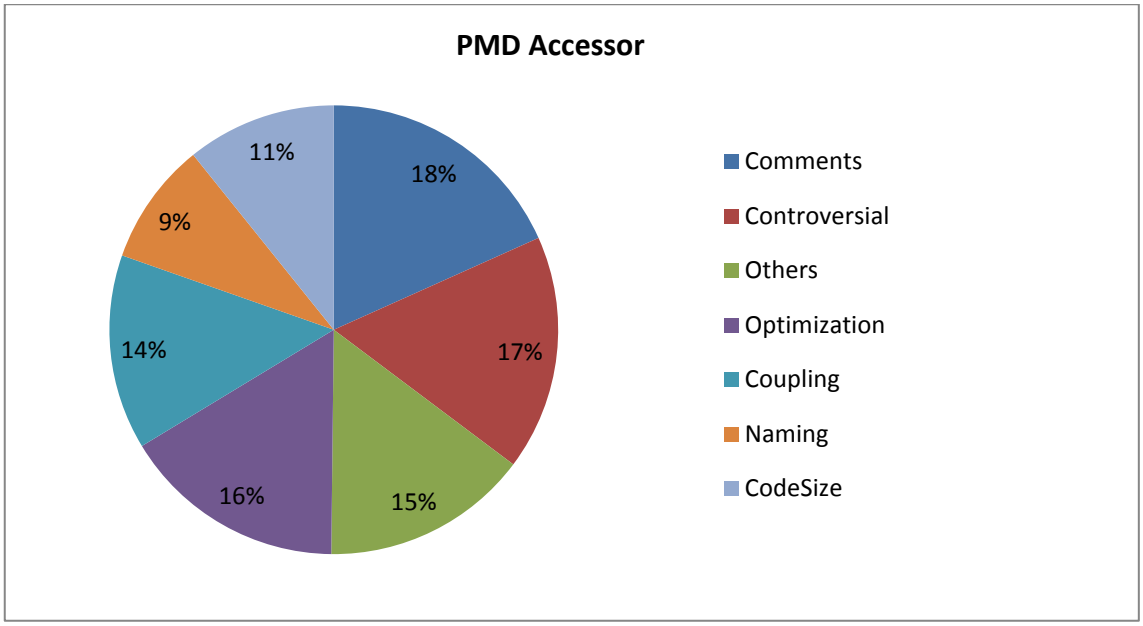
PMD

Με όλους τους κανόνες ενεργοποιημένους το PMD εντόπισε 33360 παραβιάσεις για το έργο BlackWindowSpider από τις οποίες οι περισσότερες ανήκουν στις κατηγορίες Optimization, Coupling και Comments, Controversial (σχήμα 4.33).

Για το έργο Accessor εντόπισε 3368 παραβιάσεις από τις οποίες οι περισσότερες ανήκουν επίσης στις κατηγορίες Comments, Controversial Optimization, και Coupling (σχήμα 4.34).

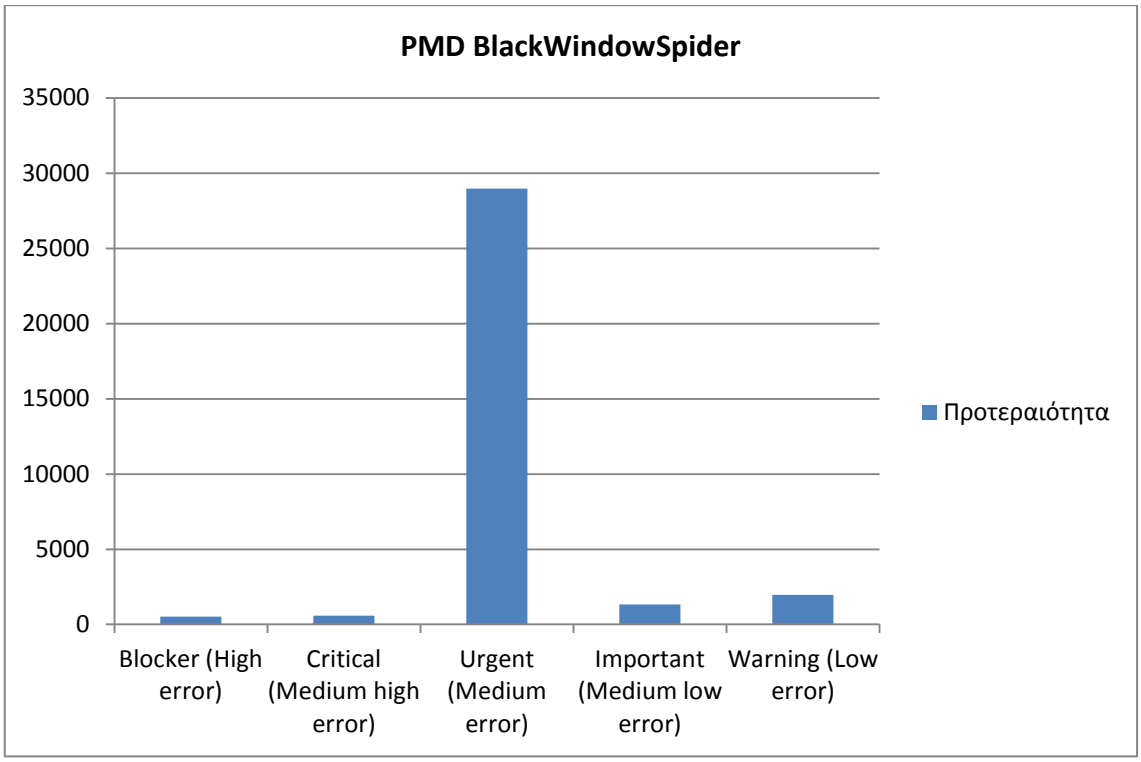


Σχήμα 4.33: Pmd, έργο λογισμικού BlackWindowSpider

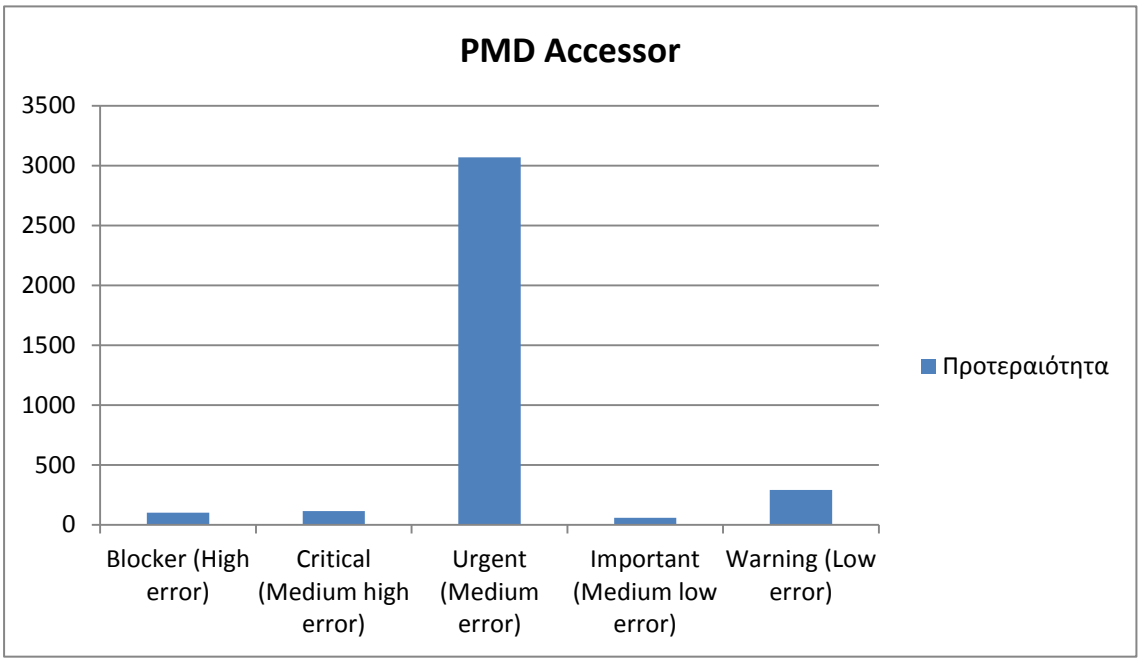


Σχήμα 4.34: Pmd, έργο λογισμικού Accessor

Ανά προτεραιότητα το PMD εντοπίζει κυρίως "medium" λάθη, δηλαδή λάθη που δεν είναι ούτε πολύ κρίσιμα αλλά ούτε και ασήμαντα (warnings).



Σχήμα 4.35: Pmd, έργο λογισμικού BlackWindowSpider, παραβιάσεις ανά προτεραιότητα



Σχήμα 4.36: Pmd, έργο λογισμικού Accessor, παραβιάσεις ανά προτεραιότητα

Ο πιο κάτω πίνακας αναπαριστά μερικές από τις παραβιάσεις που ανήκουν στην κατηγορία Blocker και Critical δηλαδή λάθη που το PMD τα επισημαίνει ως πολύ σοβαρά.

Παραβίαση	Κατηγορία	Κώδικας	Περιγραφή
1. VariableNaming Conventions	Blocker Naming	Esami – TestEserc1.java int[][] A5	Οι final μεταβλητές πρέπει να έχουν μόνο κεφαλαίους χαρακτήρες και οι μη final μεταβλητές δεν πρέπει να περιέχουν τον χαρακτήρα '_' και πρέπει να ξεκινούν με μικρούς χαρακτήρες.

2. AvoidThrowing RawExceptionTypes	Blocker
	Strict Exception Rules
		BlackWindowSpider – CrawIJob.java
		
		try {
		return this.bdbjeMBeanHelper.getAttribute(
		
		this.controller.getBdbEnvironment(), attribute_name);
		} catch (MBeanException e) {
		throw new RuntimeOperationsException(new RuntimeException(e));
		}
		}
			Αποφυγή των τύπων exception RuntimeException, Throwable, Exception,or Error. Αυτοί οι τύποι exception αποτελούν κακές πρακτικές.

3. EmptyMethodInAbstract ClassShouldBeAbstract	Blocker
	Design
		BlackWindowSpider - HttpMethodBase.java
		
		protected void processResponseBody(HttpState state, HttpConnection conn) {
		}
			Μία κενή μέθοδος σε μία abstract κλάση θα πρέπει να είναι abstract.

4. ConstructorCalls OverridableMethod	Blocker
	Design
		BlackWindowSpider – PDFParser.java
		
		public PDFParser(String doc) throws IOException {
		resetState();
		getInFromFile(doc);
		initialize();
		}
			Οι κατασκευαστές πρέπει να καλούν μόνο ιδιωτικές, static, ή final μεθόδους που αυτές δεν καλούν άλλες δημόσιες μεθόδους.

5. SystemPrintLn	Critical Java Logging	BookMarker – AdminGui.java System.out.println(cp);	Είναι καλύτερη η χρήση log.

6. AvoidThrowingNull PointerExceptionTypes	Blocker
	Strict Exception Rules
		BlackWindowSpider – Heritrix.java
		
		if (getJobHandler() == null) {
		throw new NullPointerException("Heritrix jobhandler is null.");
		}
			Αποφυγή του exception NullPointerException.
			Η χρήση του αποτελεί κακή πρακτική καθώς οι χρήστες θα θεωρήσουν πως είναι από την virtual machine.

7. TooFewBranches ForASwitchStatement	Blocker Design	BlackWindowSpider – LinkScoper.java switch (c) {case	Μια δήλωση switch που έχει πολύ λίγα cases είναι καλύτερα να αντικατασταθεί με

		Link.REFER_HOP: return (preferenceDepthHops >= 0 ? CandidateURI.HIGH : CandidateURI.MEDIUM); default: if (preferenceDepthHops == 0) return CandidateURI.HIGH;	if εντολή.

Πίνακας 4.18: Pmd, blocker παραβιάσεις

Ο πάρακατω πίνακας παρουσιάζει συνοπτικά τα αποτελέσματα της ανάλυσης για όλα τα έργα λογισμικού Java.

	BlackWindowSpider	Accessor	BookMarker	Esami	JavaGame
Optimization	7795	544	321	280	296
Coupling	6668	473	302	54	65
Comments	6213	616	193	175	180
Controversial	4153	571	141	124	140
Naming	3329	298	138	171	182
Design	1268	30	186	48	52
Migration	1206	1	0	0	0
Unnecessary	256	5	0	0	0
Code size	476	363	22	7	12
Others	1996	467	311	89	126
Λάθος Αποτελέσματα	✓	✓	✓	✓	✓

Πίνακας 4.19: Pmd, έργα λογισμικού

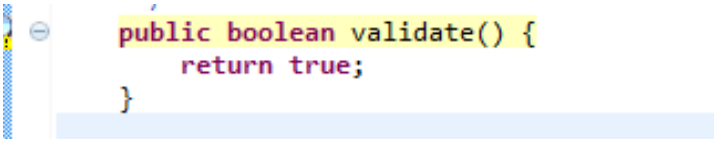
Το ποσοστό των λανθασμένων αποτελεσμάτων που επιστρέφει είναι μικρό και μπορεί να αποφευχθεί με την απενεργοποίηση των ανάλογων κανόνων ελέγχου.

Το πιο κάτω σχήμα παρουσιάζει ένα λάθος αποτέλεσμα που εντοπίζει το PMD στο έργο λογισμικού Accessor. Σύμφωνα με το εργαλείο στην μεταβλητή results ανατίθεται τιμή περισσότερες από μία φορά.
Η μεταβλητή results όμως είναι πίνακας και οι τιμές που ανατίθενται είναι για τις θέσεις 0,1 και 2 του πίνακα.

```
//update td.DATACONST set DC_VALUE='11.50' where DC_CONST_NA
public int[] prepareProjectsForDowngrade() {
    int[] results = new int[3];
    results[0] = deleteTable("FPE");
    results[1] = deleteTable("SAMPLE_FPE");
    String sql = getDowngradeProjectVersionSQL();
    results[2] = executeBatch(sql);
    return results;
```

Σχήμα 4.37: Pmd, έργο λογισμικού Accessor, Data flow Anomaly for variable results

Το πιο κάτω σχήμα παρουσιάζει ένα λάθος αποτέλεσμα που εντοπίζει το PMD στο έργο λογισμικού BlackWindowSpider.
Η μέθοδος validate() βρίσκεται στην abstract κλάση HttpMethodBase και σύμφωνα με το PMD πρέπει να είναι abstract καθώς είναι κενή.



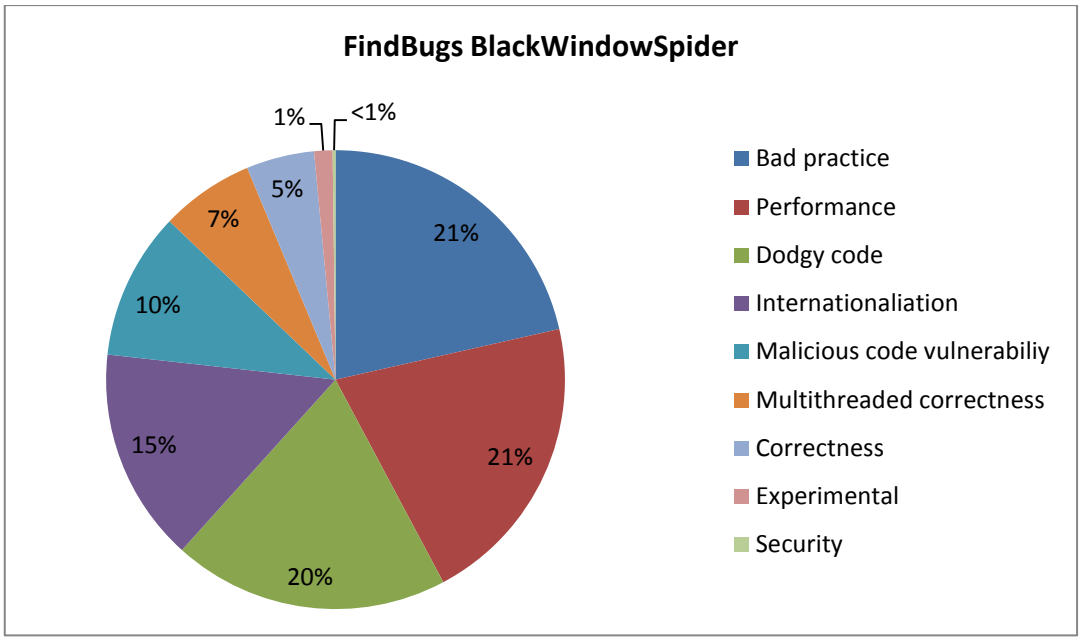
Σχήμα 4.38: Pmd, έργο λογισμικού BlackWindowSpider, Empty Method In Abstract Class Should Bee Abstract

Παρατηρούμε ότι το PMD εντοπίζει κυρίως παραβιάσεις που αποτελούν κακές πρακτικές συγγραφής κώδικα. Οι παραβιάσεις αφορούν κυρίως το σχεδιασμό και την εμφάνιση του κώδικα που με την διόρθωση τους διευκολύνεται η ανάγνωση του κώδικα.

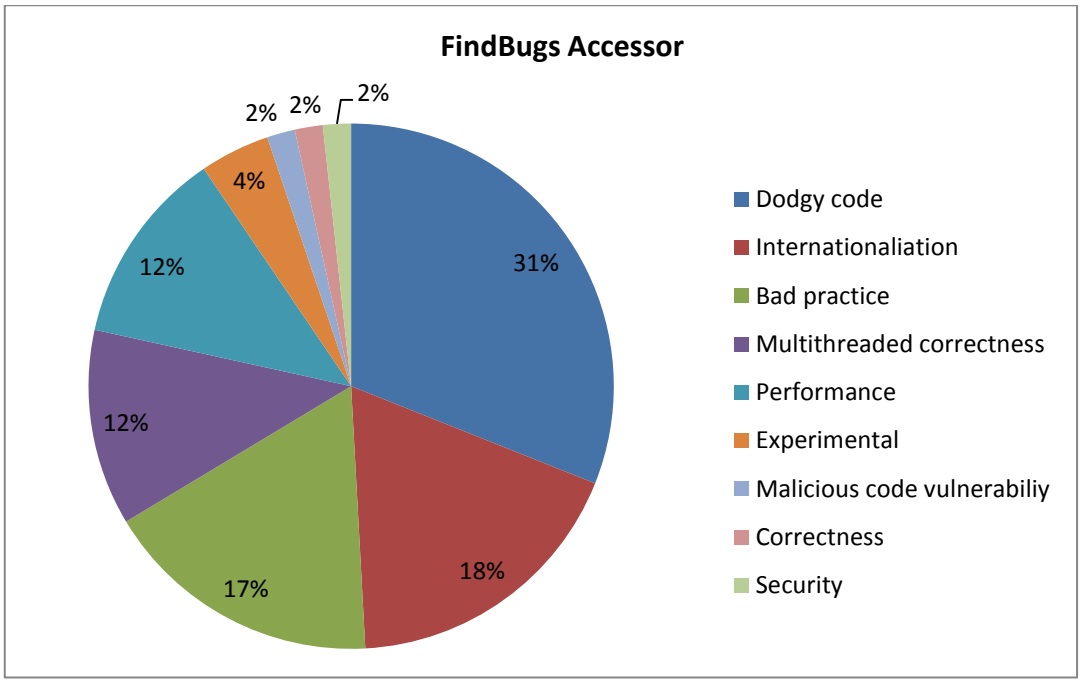
FindBugs

Με όλους τους κανόνες ελέγχου ενεργοποιημένους και με το βαθμό εμπιστοσύνης της αναφοράς (minimum confidence to report) καθορισμένο στην τιμή ‘low’ το FindBugs εντόπισε 1002 σφάλματα (bugs) για το έργο λογισμικού BlackWindowSpider. Το 42% ανήκουν στις κατηγορίες ‘Bad practice’ (21%) και ‘Performance’ (21%), το 20% ανήκουν στην κατηγορία ‘Dodgy code’, και το υπόλοιπο 38% ανήκουν στις υπόλοιπες κατηγορίες.

Για το έργο Accessor εντόπισε 116 σφάλματα (bugs). Το 31% ανήκουν στην κατηγορία ‘Dodgy code’, το 18% ανήκουν στην κατηγορία ‘Internationalization’, το 17% ανήκουν στην κατηγορία ‘Dodgy code’ και το υπόλοιπο 34% στις υπόλοιπες κατηγορίες.



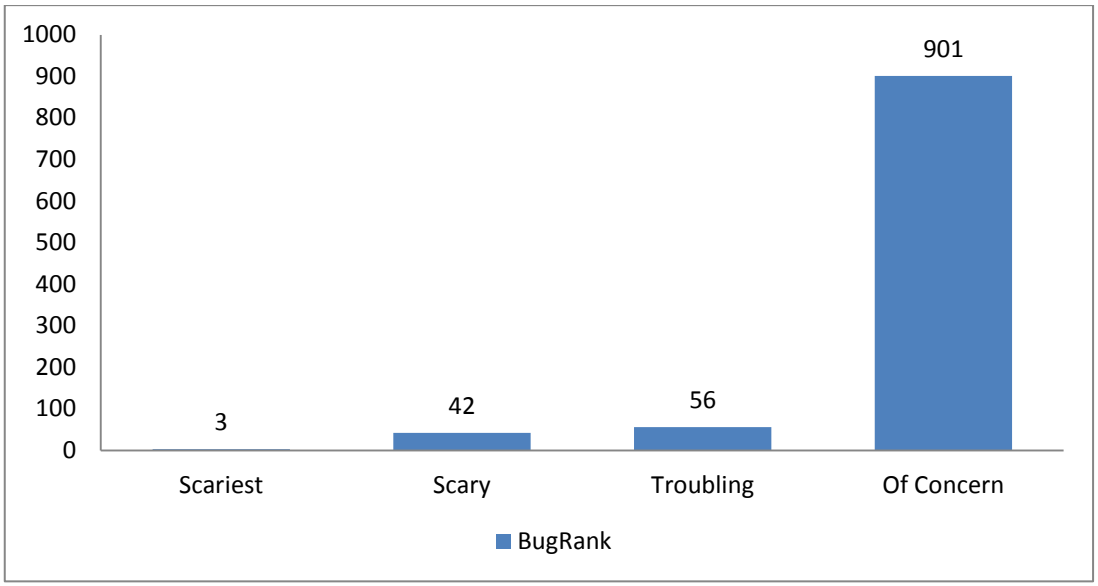
Σχήμα 4.39: Findbugs, έργο λογισμικού BlackWindowSpider



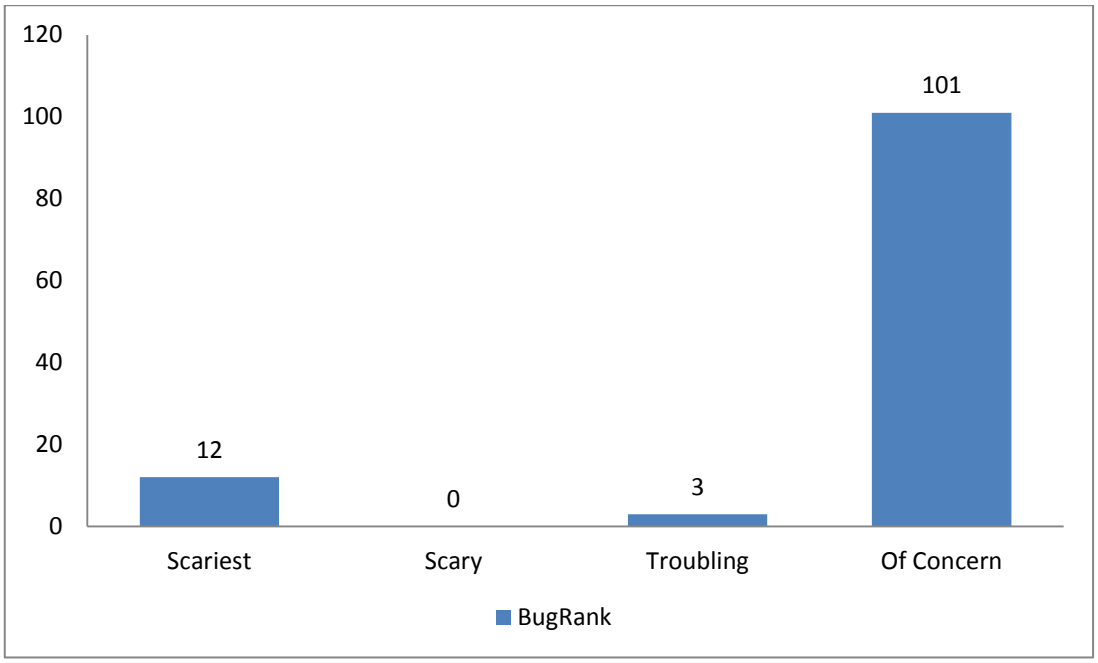
Σχήμα 4.40: Findbugs, έργο λογισμικού Accessor

Η πλειονότητα των σφαλμάτων που εντοπίζει το εργαλείο ανήκει στην κατηγορία (bugRank) 'of Concern' δηλαδή σφάλματα που δεν είναι πολύ σοβαρά.

Τα πιο κάτω σχήματα παρουσιάζουν τον αριθμό σφαλμάτων που εντοπίστηκαν στα 2 έργα λογισμικού ανά κατηγορία.



Σχήμα 4.41: Findbugs, έργο λογισμικού BlackWindowSpider, BugRank



Σχήμα 4.42: Findbugs, έργο λογισμικού BlackWindowSpider, BugRank

Ο πιο κάτω πίνακας αναπαριστά μερικά από τα σφάλματα που εντοπίστηκαν σε όλα τα έργα λογισμικού και ανήκουν στις κατηγορίες Scary και Scariest δηλαδή λάθη που το FindBugs τα επισημαίνει ως πολύ σοβαρά.

Σφάμα	Κατηγορία	Κώδικας	Περιγραφή

1. Value of 'shouldForget' from previous case is overwritten here due to switch statement fall through	Scariest/ Correctness	BlackWindowSpider – AdaptiveRevisitFrontier.java
		
		switch(curi.getFetchStatus()) {
		case S_OUT_OF_SCOPE:
		case S_TOO_MANY_EMBED_HOPS:
		case S_TOO_MANY_LINK_HOPS:
		shouldForget = true;
		default:
		shouldForget = false;
		}
			Η τιμή της μεταβλητής 'shouldForget' που έχει αποθηκευτεί από το προηγούμενο switch case γίνεται overwritten στο τώρα default case. Αυτό οφείλεται στο ότι στο προηγούμενο case δεν υπάρχει εντολή break ή return.

2. Possible null pointer dereference	Scary/ Correctness	BlackWindowSpider-WorkQueueFrontier.java
		
		for(int count = 0; iterator.hasNext() && (count < max); count++) {
		obj = iterator.next();
		if (obj == null) {
		continue;
		}
		q = (obj instanceof WorkQueue)?
		(WorkQueue)obj:
		
		(WorkQueue)this.allQueues.get((String)obj);
		if(q == null) {
		w.print("WARNING: No report for queue "+obj);
		}
		q.reportTo(w);
		}
			Πιθανή δημιουργία NullPointerException σε περίπτωση που ο συγκεκριμένος κώδικας εκτελεστεί.

3. Null passed for nonnull parameter	Scary/ Correctness	BlackWindowSpider - SimpleHttpServer.java
		
		public SimpleHttpServer(List webapps, int port, boolean expandWebapps)
		throws Exception {
		initialize(null, port);
			Η τιμή null παιρνάτε σε μή null παράμετρο

4. Comparison of String objects using == or !=	Scary/ Bad practices	Accessor - FileAccessor.java if (line != null && line.trim()!="")	Είναι καλύτερη πρακτική η χρήση της μεθόδου equals(Object).
5. Call to equals(null)	Scary/ Correctness	Esami – ProvaEserc4.java confronta(fg1.equals(null)+"", "false",5);	Η δήλωση αυτή πάντα θα επιστρέφει false

Σχήμα 4.20: Findbugs, Scariest, Scary παραβιάσεις

Ο παρακάτω πίνακας παρουσιάζει συνοπτικά τα αποτελέσματα της ανάλυσης για όλα τα έργα λογισμικού Java.

	BlackWindowSpider	Accessor	BookMarker	Esami	JavaGame
Bad practice	215	20	11	12	9
Performance	208	14	5	10	4
Dodgy code	195	36	4	12	11
Internationaliation	151	21	4	7	3
Malicious code vulnerabiliy	104	2	2	2	4
Multithreaded correctness	66	14	0	0	0
Correctness	48	2	0	4	1
Experimental	13	5	0	0	0
Security	2	2	5	0	0
Λάθος Αποτελέσματα	✓	✓			

Σχήμα 4.21: Findbugs, έργα λογισμικού

Το ποσοστό των λανθασμένων αποτελεσμάτων που επιστρέφει είναι πολύ μικρό και συνήθως ανήκουν στην κατηγορία 'low confidence'. Τα λανθασμένα αποτελέσματα μπορούν να αποφευχθούν με την απενεργοποίηση των ανάλογων κανόνων ελέγχου.

Το πιο κάτω σχήμα παρουσιάζει ένα λάθος αποτέλεσμα που επέστρεψε το FindBugs στο έργο Accessor. Σύμφωνα με το FindBugs η τιμή της μεταβλητής keyValue δεν πρόκειται να είναι ποτέ null και έτσι είναι αχρείαστη η συνθήκη. Όμως δεν γνωρίζουμε αν η μεταβλητή keyVaue δεν πρόκειται ποτέ να είναι null.

```
public synchronized void load (FileReader fr) throws IOException {
    BufferedReader br = new BufferedReader(fr);
    String s = null;
    while ((s=br.readLine())!= null) {
        s = s.trim();
        //skip empty line
        if (s.equals("")) continue;

        //skip line begin with #
        if (s.startsWith("#")) continue;

        //split by "="
        String[] keyValue = s.split(SPLITER);
        if (keyValue==null || keyValue.length<1) {
            System.out.println("Config file is not well formed. Must be one \"=\" a line!");
        }
        if (keyValue[1]==null || keyValue[1].equals("")) continue;

        configureTable.put(keyValue[0], keyValue[1]);
    }
```

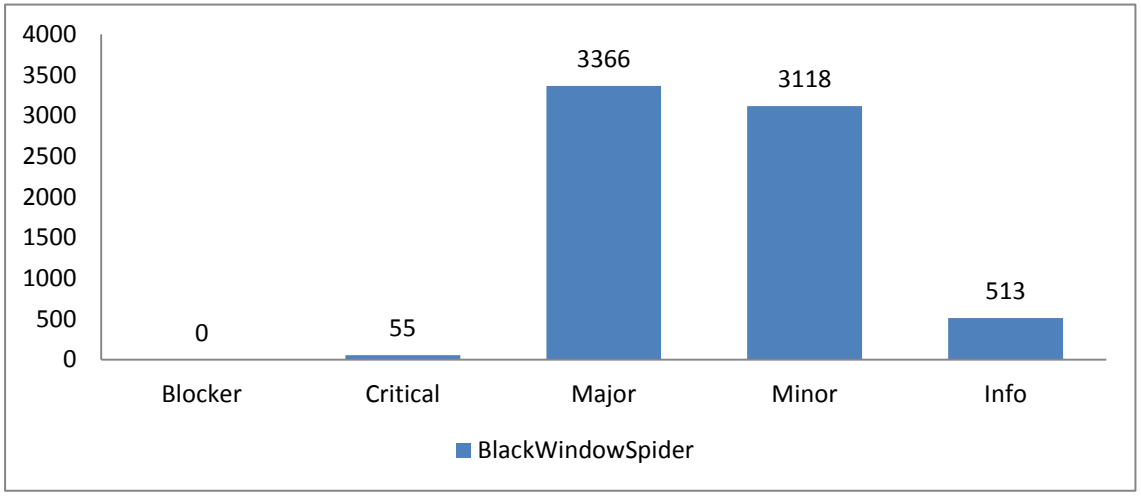
Σχήμα 4.43: Findbugs, Accessor, Redundant nullcheck of keyValue, which is known to be non-null

Το FindBugs εντοπίζει συχνά σημαντικά (potential bugs) και εύστοχα λάθη (ή πιθανά λάθη) και επισημαίνει κακές πρακτικές προγραμματισμού και σφάλματα που δυσκολεύουν την απόδοση.

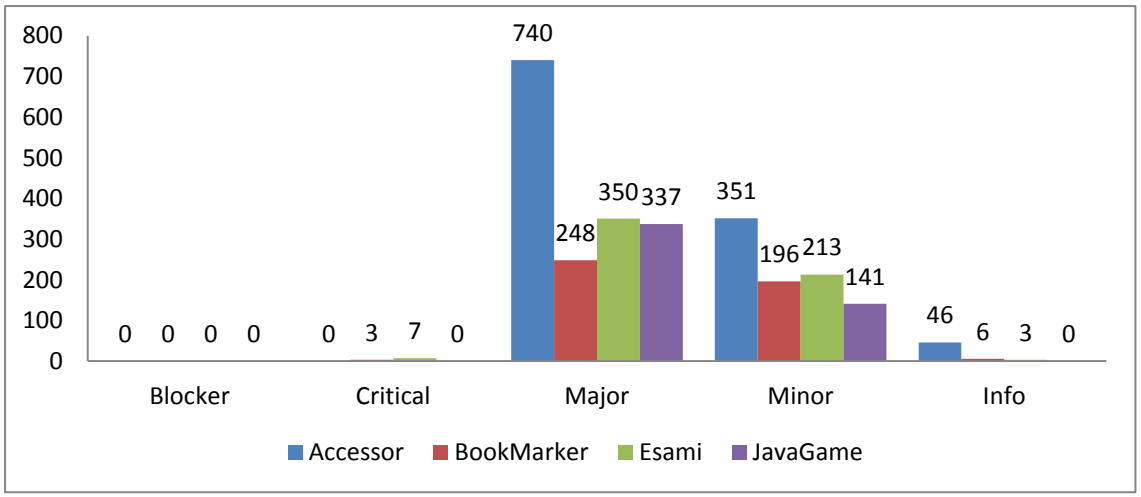
Sonarqube

Οι κανόνες ελέγχου που εφαρμόζει το Sonarqube είναι ένα σύνολο από κανόνες ελέγχου των εργαλείων Checkstyle, PMD και FindBugs και άλλων κανόνων.

Με όλους τους προκαθορισμένους κανόνες ελέγχου ενεργοποιημένους (profile: sonar way) το Sonarqube εντόπισε 7053 λάθη (issues) για το έργο BlackWindowSpider και 1037 λάθη (issues) για το έργο Accessor. Η πλειονότητα των λαθών ανήκουν στην κατηγορία 'major' δηλαδή λάθη όχι τόσο κρίσιμα αλλά ούτε και ασήμαντα.



Σχήμα 4.43: Sonarqube BlackWindowSpider issues



Ο πιο κάτω πίνακας αναπαριστά μερικές από τις παραβιάσεις που ανήκουν στην κατηγορία Critical και Major δηλαδή λάθη που το Sonarqube τα επισημαίνει ως σοβαρά και λιγότερο σοβαρά αντίστοιχα.

Παραβίαση	**Κατηγορία**	**Κώδικας**	**Περιγραφή**

1. Avoid using equals() to compare against null	Critical
	PMD
		BlackWindowSpider - SimpleTypeTest.java
		
		SimpleType t2 = new SimpleType("a1", "b1", "c1")
		SimpleType t3 = new SimpleType("a2", "b2", "c2")
		assertTrue(t1.equals(t2))
		assertFalse(t1.equals(t3)
		assertTrue(t1.equals(t1));
		assertFalse(t1.equals(null));
		}
			Αποφυγή της χρήσης της μεθόδου equals() με null τιμή.

2. Replace Vector With List	Major PMD	JacaGame-THBoard.java Vector<THTile> next_vector = new Vector<THTile>();	Συνιστάται η χρήση ArrayList

3. Parameter Assignment	Major
	Checkstyle
		BlackWindowSpider - HttpMethodBase.java
		
		try {
		// create a URI and allow for null/empty uri values
		if (uri == null || uri.equals("")) {
		uri = "/";
		}
			Αποφυγή ανάθεσης σε παραμέτρους.

4. SystemPrintLn	Major
	PMD
		Accessor – AccessorFactory.java
		
		} catch (Exception e) {
		System.out.println("The db config properties load failed.");
		throw e;
		}
			Είναι προτιμότερη η χρήση logger.

5. BookMarker.ChartPanel	Critical
	PMD
		BookMarker ChartPanel.java
		
		// cocnstructor
		public ChartPanel(double[] v, String[] n, String t){
		names = n;
		values = v;
		title = t;
		}
			Κατασκευαστές και μεθόδοι που έχουν παραμέτρους πίνακες θα πρέπει να κάνουν clone το αντικείμενο και να αποθηκεύουν το αντίγραφο της παραμέτρου.

6. Equals Hash Code	Critical
	PMD
		@Override
		public boolean equals(Object o) {
		if(o==null || !getClass().equals(o.getClass()))
		return false;
		FiguraGeometrica fg = (FiguraGeometrica)o;
			Μια κλάση που κάνει override την μέθοδο equals( ) πρέπει να κάνει και την μέθοδο hashCode().

Πίνακας 4.22: Sonarqube BlackWindowSpider issues

Το ποσοστό των λανθασμένων αποτελεσμάτων που επιστρέφει είναι μικρό. Ο χρήστης μπορεί να επισημάνει τα λανθασμένα αποτελέσματα ως ‘false positive’ και έτσι να μην επαναληφθούν στις επόμενες αναλύσεις.

Το πιο κάτω σχήμα αναπαριστά ένα λανθασμένο αποτέλεσμα στο έργο λογισμικού BlackWindowSpider. Το Sonarqube δεν αναγνωρίζει ότι και οι 2 συνθήκες αφορούν το ίδιο αντικείμενο και προτείνει την αντικατάσταση του τελεστή && με τον τελεστή ||.
Το λάθος σε αυτήν την περίπτωση είναι ότι η δεύτερη συνθήκη αφορά το αντικείμενο o2 και όχι το o1, ένα λάθος βέβαια που οφείλεται σε απροσεξία από τον προγραμματιστή.

```
74      * Comparator for treeset of uuris.
75      */
76     private static final Comparator<UURI> CMP = new Comparator<UURI> () {
77         public int compare(UURI o1, UURI o2) {
78             int result = -1;
79             if (o1 == null && o1 == null){
80                 result = 0;
81             } else if (o1 == null) {
82                 result = -1;
83             } else if (o2 == null) {
84                 result = 1;
```

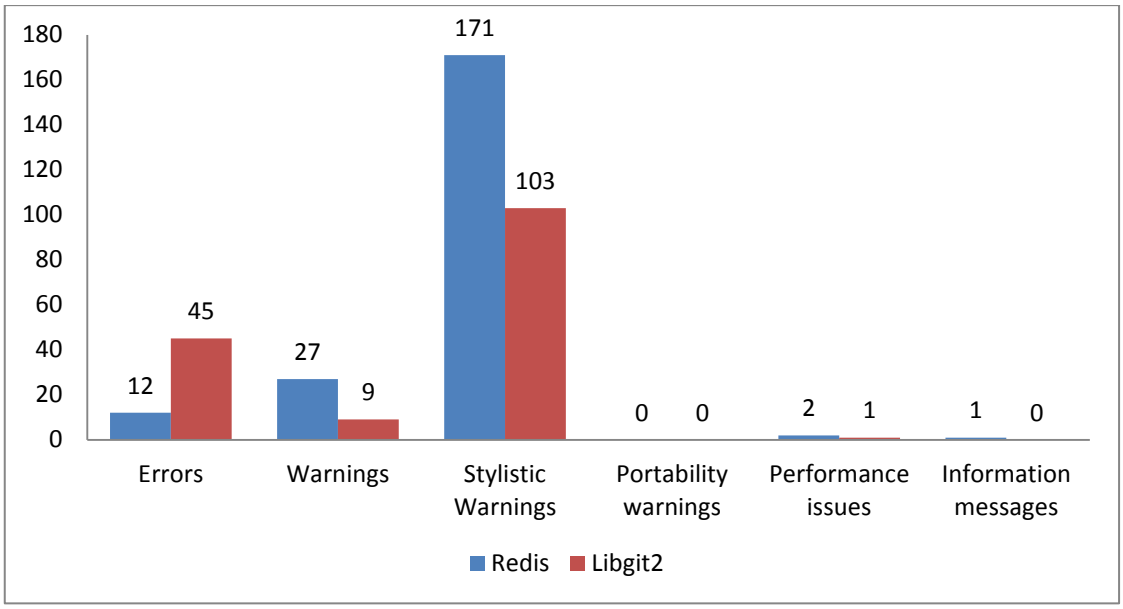
Σχήμα 4.44: Sonarqube BlackWindowSpider, SeeCachingScopeTest.java, λάθος αποτέλεσμα

Το Sonarqube συνδυάζει τους κανόνες ελέγχου των υπόλοιπων εργαλείων και αναλύει τον κώδικα υπολογίζοντας από μικρές στιλιστικές λεπτομέρειες, μέχρι σημαντικά σχεδιαστικά (design) λάθη.

4.3.8.2 Εργαλεία γλώσσας προγραμματισμού C

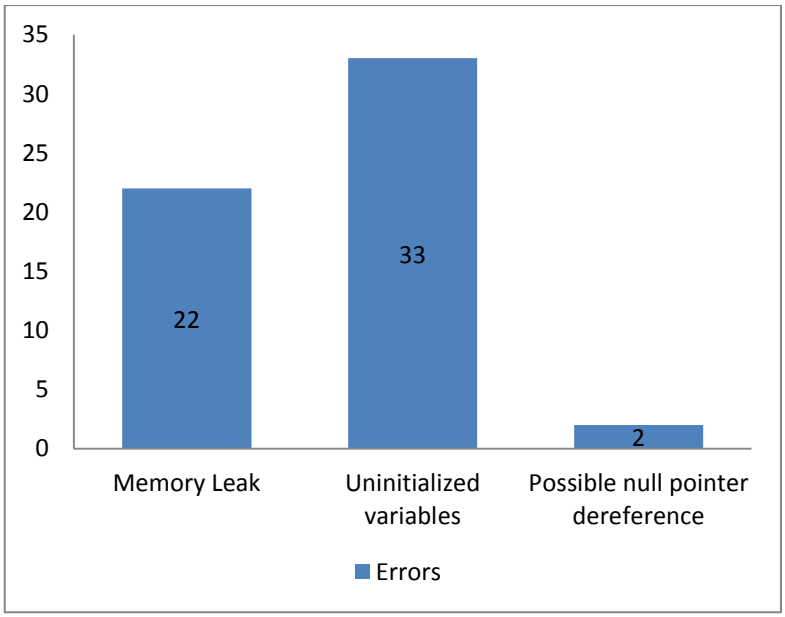
Στο σημείο αυτό θα αναλύσουμε τα έργα λογισμικού Libgit2 και Redis για να αξιολογήσουμε το εργαλείο CppCheck.

Το CppCheck και για τα 2 έργα λογισμικού εντόπισε περισσότερο σφάλματα στην κατηγορία ‘Stylistic warnings’. Ο πιο κάτω πίνακας παρουσιάζει τα αποτελέσματα της ανάλυσης και για τα 2 έργα λογισμικού.



Σχήμα 4.45: CppCheck, έργα λογισμικού, Redis, Libgit2

Τα λάθη που ανήκουν στην κατηγορία ‘error’ αφορούν κυρίως μη αρχικοποιημένες μεταβλητές (uninitialized variables) και memory leaks που στην γλώσσα C πιθανόν να οδηγήσουν σε λάθη κατά την εκτέλεση.



Σχήμα 4.46: CppCheck, έργα λογισμικού, Redis, Libgit2, errors

Ο πιο κάτω πίνακας παρουσιάζει μερικά από τα λάθη που εντοπίστηκαν κατά την ανάλυση των έργων λογισμικού Redis, Libgit2 και άλλων έργων λογισμικού.

Λάθος	**Κατηγορία**	**Κώδικας**	**Περιγραφή**

Uninitialized variable dst	Error	if (hi->encoding == REDIS_ENCODING_ZIPLIST) {
		unsigned char *vstr = NULL;
		unsigned int vlen = UINT_MAX;
		long long vll = LLONG_MAX;
		
		hashTypeCurrentFromZiplist(hi, what, &vstr, &vlen, &vll);
		if (vstr) {
		dst = createStringObject((char*)vstr, vlen);
		} else {
		dst = createStringObjectFromLongLong(vll);
		}
		} else if (hi->encoding == REDIS_ENCODING_HT) {
		
		hashTypeCurrentFromHashTable(hi, what, &dst);
		incrRefCount(dst);
		} else {
		redisPanic("Unknown hash encoding");
		}
		return dst;
		}
			Αν εκτελεστεί το τελευταίο else τότε η μεταβλητή dist δεν θα έχει τιμή.

Memory Leak: a	Error	int *a = new int[5];
		a[0] = 4;
		if(a[0] != n) { return; }
		delete []a;
			Η μνήμη που δεσμεύεται για το a δεν αποδεσμεύεται εκεί που πρέπει.

Array 'a[10]' index 20 out of bounds	Error	void f()
		{
		char a[10];
		a[20] = 0;
		}
			Ο πίνακας a έχει μόνο 10 θέσεις και δεν μπορεί να του ανατεθεί τιμή στην θέση 20

Πίνακας 4.23 CppCheck, errors

Το CppCheck επιστρέφει συχνά και λανθασμένα αποτελέσματα.

Το πιο κάτω σχήμα παρουσιάζει ένα λανθασμένο αποτέλεσμα που επιστρέφει η ανάλυση του έργου Redis. Το CppCheck επιστρέφει ‘Array 'a[10]' index 20 out of bounds’. Αν όμως ισχύει η συνθήκη (x+y==2) τότε δεν υπάρχει λάθος.

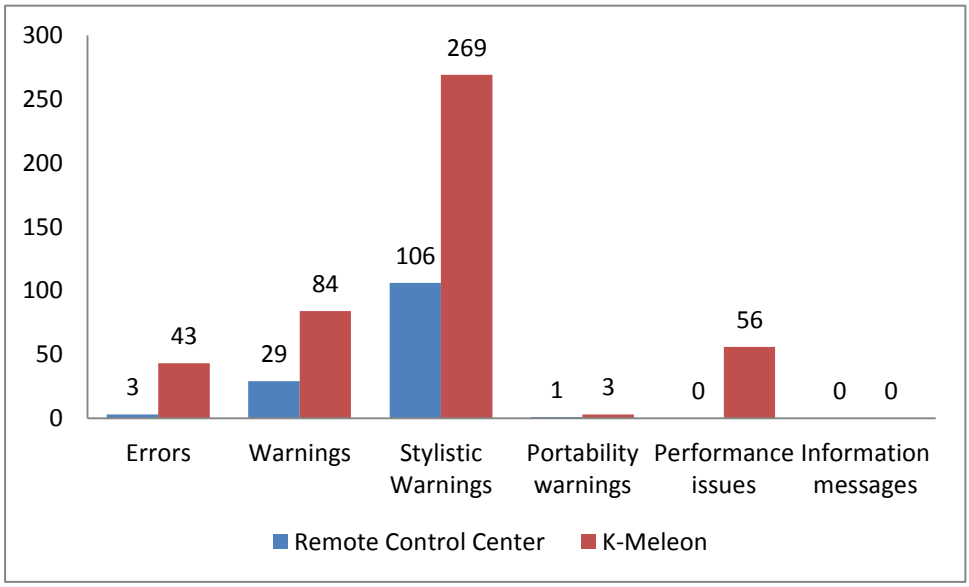
```
288
289  void f1(char *s)
290  {
291    s[20] = 0;
292  }
293  void f2()
294  {
295    char a[10];
296    if (x + y == 2) {
297    f1(a);
298    }
299  }
300
```

Σχήμα 4.47: CppCheck, Redis, false positive

4.3.8.3 Εργαλεία γλώσσας προγραμματισμού C++

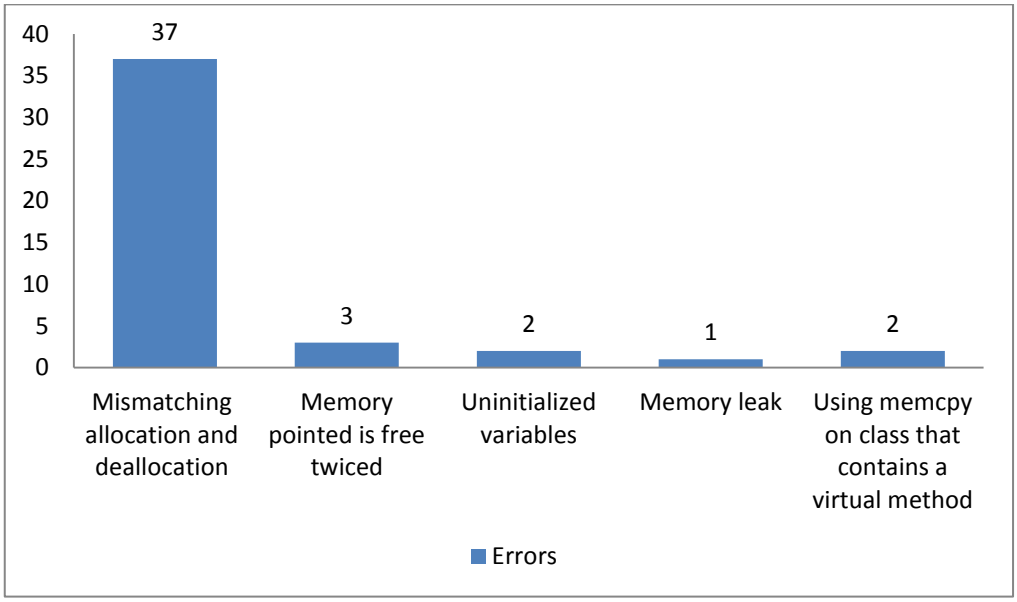
Στο σημείο αυτό θα αναλύσουμε τα έργα λογισμικού K-Meleon και Remote Control Center για να αξιολογήσουμε το εργαλείο CppCheck.

Το CppCheck και για τα 2 έργα λογισμικού εντόπισε περισσότερα σφάλματα στην κατηγορία 'Stylistic warnings'. Ο πιο κάτω πίνακας παρουσιάζει τα αποτελέσματα της ανάλυσης και για τα 2 έργα λογισμικού.



Σχήμα 4.48: CppCheck, errors

Τα λάθη που ανήκουν στην κατηγορία 'error' αφορούν κυρίως δεσμεύσεις και αποδεσμεύσεις στην μνήμη.



Σχήμα 4.49: CppCheck, έργα λογισμικού, K-Meleon, Remote Control Center, errors

Ο πιο κάτω πίνακας παρουσιάζει μερικά από τα λάθη που εντοπίστηκαν κατά την ανάλυση των έργων λογισμικού K-Meleon και Remote Control Center.

Λάθος	**Κατηγορία**	**Κώδικας**	**Περιγραφή**
1. Mismatching allocation and deallocation	Error	TCHAR *szFileName = new TCHAR[INTERNET_MAX_URL_LENGTH]; … delete szFileName;	Η αποδεσμεύση πρέπει να είναι delete[] szFileName;

2. Uninitialized variable n1, n2	Error	char ConvertToDecimal(char c1, char c2)
		{
		char n1, n2;
		if (c1>='0' && c1<='9')
		n1 = c1-'0';
		else if (c1>='A' && c1<='F')
		n1 = c1-'A'+10;
		else if (c1>='a' && c1<='f')
		n1 = c1-'a'+10;
		if (c2>='0' && c2<='9')
		...
		return n1*16+n2;
		}
			Αν καμιά if εντολή δεν εκτελεστεί οι 2 μεταβλητές δεν θα έχουν τιμή.

Πίνακας 4.23: CppCheck, errors

Το CppCheck επιστρέφει συχνά και λανθασμένα αποτελέσματα.

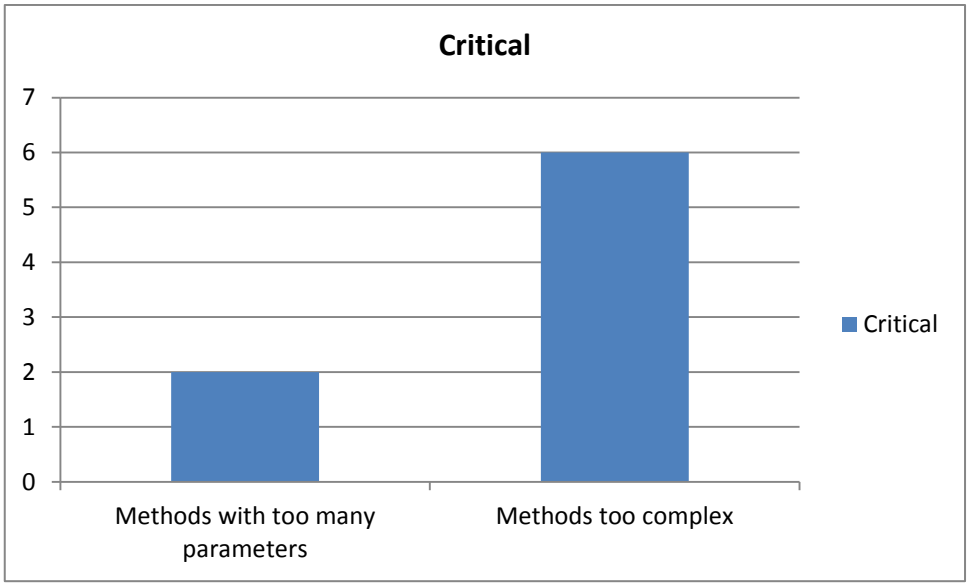
Στον πίνακα 4.23 στο 2ο λάθος υπάρχουν προηγούμενοι έλεγχοι που δεν επιτρέπουν στις μεταβλητές να μην πάρουν κάποια τιμή από αυτές στις συνθήκες. Επομένως λανθασμένα το CppCheck επιστρέφει ότι υπάρχουν μη αρχικοποιημένες μεταβλητές.

4.3.8.4 Εργαλεία γλώσσας προγραμματισμού C#

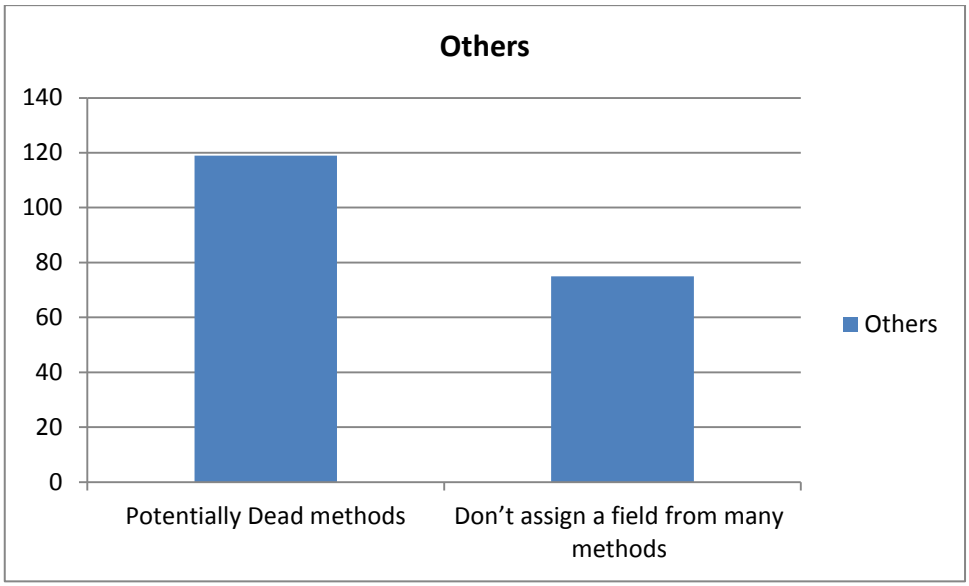
Στο σημείο αυτό θα αναλύσουμε το έργο λογισμικού DotNetOpenAuth με το εργαλείο NDepend.

Το Ndepend εντόπισε 8 λάθη με προτεραιότητα 'critical' , δηλαδή κρίσιμα λάθη, και 71 άλλα λάθη (critical or not).

Τα πιο κάτω σχήματα παρουσιάζουν μερικά από τα λάθη που εντοπίστηκαν κατά την ανάλυση του έργου.



Σχήμα 4.50: Ndepend, critical rules violated



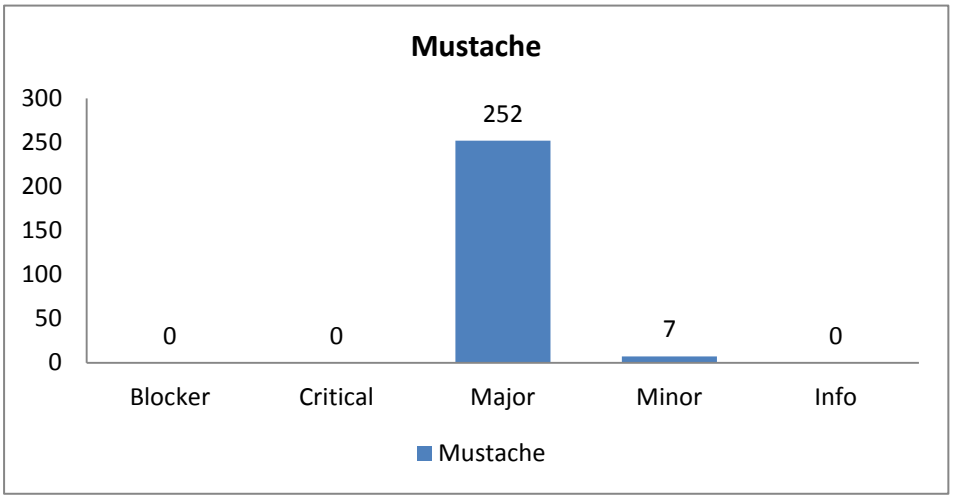
Σχήμα 4.51: Ndepend, others rules violated

Παρατηρούμε ότι το Ndepend επισημαίνει ως κρίσιμα λάθη κανόνες που αφορούν την πολυπλοκότητα του κώδικα.

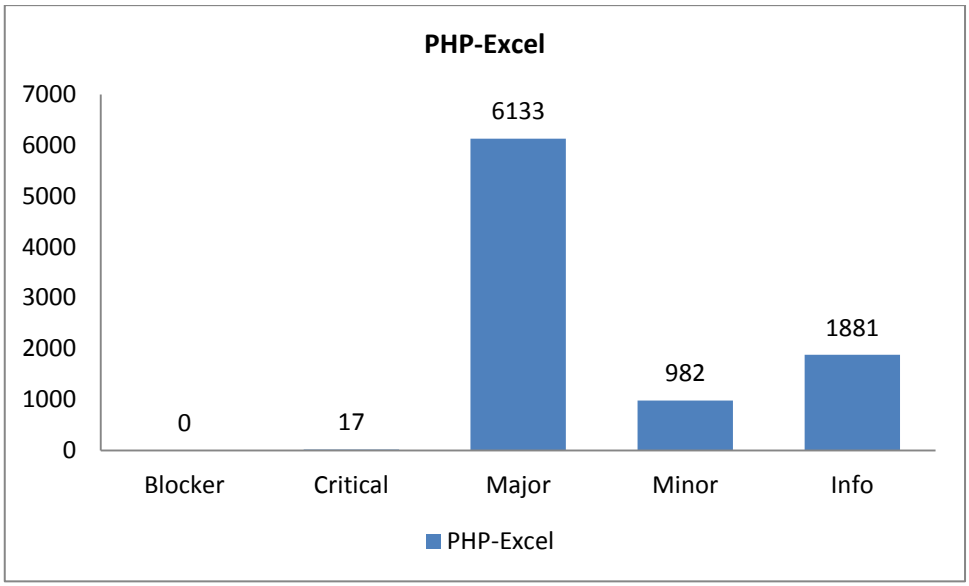
4.3.8.5 Εργαλεία γλώσσας προγραμματισμού PHP

Στο σημείο αυτό θα αναλύσουμε τα έργα λογισμικού PHP-Excel και Mustache με το εργαλείο Sonarqube.

Το Sonarqube και για τα 2 έργα λογισμικού εντόπισε περισσότερα σφάλματα (issues) στην κατηγορία ‘Major’. Οι πιο κάτω πίνακες παρουσιάζουν τα αποτελέσματα της ανάλυσης για τα 2 έργα λογισμικού.



Σχήμα 4.52: Sonarqube, Mustache issues



Σχήμα 4.53: Sonarqube, PHP-Excel issues

Ο πιο κάτω πίνακας παρουσιάζει μερικά από τα λάθη που εντοπίστηκαν κατά την ανάλυση των έργων λογισμικού PHP-Excel και Mustache.

Παραβίαση	Κατηγορία	Κώδικας	Περιγραφή
1. Switch cases should end with an unconditional break, return,continue,throw statement	Critical PMD	case 'area3DChart' : $dimensions = '3d'; case 'areaChart' : …	Κάθε case πρέπει να τελειώνει με μια από τις εντολές break, return, continue, throw.
2. Statements should be on separate lines	Major PMD	if ($stack->count() != 1) return $this->_raiseFormulaError("internal error");	Σε κάθε γραμμή πρέπει να υπάρχει μόνο μία εντολή

Πίνακας 4.24: Sonarqube, issues

Το ποσοστό των λανθασμένων αποτελεσμάτων που επιστρέφει είναι μικρό. Ο χρήστης μπορεί να επισημάνει τα λανθασμένα αποτελέσματα ως ‘false positive’ και έτσι να μην επαναληφθούν στις επόμενες αναλύσεις.

Στο πιο κάτω σχήμα παρουσιάζεται ένα λανθασμένο αποτέλεσμα που εντοπίζει το εργαλείο. Σύμφωνα με το εργαλείο κάθε case πρέπει να τελειώνει με μια από τις εντολές break, return, continue, throw. Σε αυτό το case υπάρχει η εντολή return ωστόσο το Sonarqube επιστρέφει λανθασμένα πως δεν υπάρχει.

```
switch (PHPExcel_Calculation_Functions::getReturnDateType()) {
        case PHPExcel_Calculation_Functions::RETURNDATE_EXCEL :
                        return (float) $excelDateValue;
        case PHPExcel_Calculation_Functions::RETURNDATE_PHP_NUMERIC :
                        return (integer) $phpDateValue = PHPExcel_Shared_D
        case PHPExcel_Calculation_Functions::RETURNDATE_PHP_OBJECT :
                        return new DateTime('1900-01-01 '.$PHPDateArray['h
}
```

Σχήμα 4.54: Sonarqube, PHPExcel, false positive

Για την γλώσσα προγραμματισμού PHP το Sonarqube εντοπίζει κυρίως ‘λάθη’ σχετικά με την σύνταξη και την εμφάνιση του κώδικα.

4.4 Επιλογή καλύτερων εργαλείων στατικής ανάλυσης κώδικα

Με βάση τα 11 εργαλεία που επιλέξαμε και την αξιολόγηση τους με τις μετρικές αξιολόγησης καταλήγουμε στα εξής συμπεράσματα:

1. Κρίνοντας από το πλήθος και το είδος των γλωσσών προγραμματισμού που υποστηρίζουν τα εργαλεία, το Sonarqube υπερτερεί από τα υπόλοιπα καθώς υποστηρίζει Java, .Net και WEB (html, php, ruby, κλπ) γλώσσες προγραμματισμού.
2. Τα περισσότερα εργαλεία έχουν μια φιλική διεπαφή με τον χρήστη (με εξαίρεση το εργαλεί ckjm). Τα εργαλεία που επιτρέπουν την πρόσβαση στον πηγαίο κώδικα, δηλαδή τα εργαλεία Checkstyle, Metrics 1.3.6 PMD, Findbugs και NDpened μπορούν να προτιμηθούν από τα υπόλοιπα καθώς εντοπίζουν και επισημαίνουν στον προγραμματιστή τα σημεία που πιθανόν να χρίζουν διόρθωσης. Με εξαίρεση το εργαλείο Sonarqube που απαιτεί περισσότερα βήματα για την διαδικασία ανάλυσης, η ανάλυση είναι αρκετά εύκολη και δεν αποτρέπει τον χρήστη από το να μην χρησιμοποιήσει το εργαλείο.
3. Τα περισσότερα εργαλεία παρέχουν ελλιπής επεξηγήσεις. Η μετρική επεξήγηση των αποτελεσμάτων δεν κρίνει από μόνη της την ποιότητα των εργαλεία ωστόσο οι επαρκής επεξηγήσεις βοηθούν τον χρήστη να κατανοήσει καλύτερα τα αποτελέσματα.
4. Για κάθε εργαλείο, (εξαιρώντας το ckjm), μπορεί να γίνει εξαγωγή αναφορών με τα αποτελέσματα ή/και δημιουργία στατιστικών διαγραμμάτων. Η ποιότητα

των εργαλείων δεν αξιολογείται από αυτήν την μετρική μόνο αλλά μία λεπτομερής και ευανάγνωστη αναφορά της ανάλυσης παρέχει στον χρήστη μια πιο ολοκληρωμένη εικόνα για τα αποτελέσματα της ανάλυσης.

5. Οι ρυθμίσεις των περισσότερων εργαλείων μπορούν να τροποποιηθούν από τον χρήστη προσαρμόζοντας έτσι την ανάλυση όπως ο ίδιος επιθυμεί. Η δυνατότητα προσαρμογής των ρυθμίσεων από τους χρήστες είναι πολύ σημαντική καθώς ο χρήστης πιθανόν να μην ενδιαφέρεται για αποτελέσματα σε συγκεκριμένες κατηγορίες.
6. Η κοινοποίηση των αποτελεσμάτων μεταξύ των χρηστών είναι μια σημαντική παράμετρος για ένα εργαλείο καθώς επιτρέπει την επικοινωνία μεταξύ των προγραμματιστών που αναπτύσσουν ομαδικά ένα έργο και συμβάλλει στην καλύτερη επαναχρησιμοποίηση και συντήρηση του κώδικα.
7. Υπάρχουν περιπτώσεις που τα εργαλεία δεν υπολογίζουν τις ίδιες τιμές για τις μετρικές ποιότητας κώδικα. Αυτό οφείλεται στην διαφορετική ερμηνεία με την οποία τα εργαλεία υπολογίζουν τις μετρικές, για παράδειγμα κάποια εργαλεία δεν θεωρούν τους κατασκευαστές ή την main ως μέθοδοι, ενώ άλλα τα θεωρούν, κάποια ξεκινούν την αρίθμηση από το 0 ενώ άλλα από το 1, κλπ.
8. Η δυνατότητα ορισμού συγκεκριμένων συνόλων κανόνων, δίνει το περιθώριο για εξιδεικευμένη ανάλυση.
9. Τα περισσότερα εργαλεία επιστρέφουν συνήθως ένα μικρό ποσοστό λανθασμένων αποτελεσμάτων. Αυτό οφείλεται κυρίως στο γεγονός ότι το εργαλείο δεν γνωρίζει ακριβώς το μονοπάτι εκτέλεσης του κώδικα καθώς ελέγχει τον πηγαίο κώδικα στατικά. Ωστόσο η δυνατότητα επιλογής των κανόνων ελέγχου ή της επισήμανσης των λανθασμένων αποτελεσμάτων απαλείφει κατά κάποιο βαθμό την επανεμφάνιση λανθασμένων αποτελεσμάτων.

Αφού για τα πλήστα εργαλεία οι μετρικές ‘διεπαφή με τον χρήστη’, ‘επεξήγηση αποτελεσμάτων’, ‘δυνατότητα προσαρμογής’ είναι σχεδόν ισοδύναμες, θεωρήθηκε ότι η επιλογή των καλύτερων εργαλείων έχει νόημα να γίνει συγκρίνοντας τα εργαλεία με βάση τις μετρικές ‘μετρικές ποιότητας κώδικα’ και ‘αξιοπιστία αποτελεσμάτων’.

Αξιολογώντας τα εργαλεία από την σκοπιά της μετρικής ‘Μετρικές προϊόντων’ μπορούμε να πούμε ότι για την ομάδα εργαλείων Java το εργαλείο Metrics 1.3.6 είναι

το κατάλληλο για τον χρήστη που επιθυμεί να ‘μετρήσει’ τον κώδικα του με μετρικές ποιότητας κώδικα καθώς υπολογίζει σημαντικές μετρικές. Παράλληλα αν ο χρήστης επιθυμεί να εντοπίσει συγκεκριμένα τμήματα του κώδικα που έχουν μεγάλη πολυπλοκότητα ή ξεπερνούν μια συγκεκριμένη τιμή πολυπλοκότητας που ο ίδιος θέλει να ορίσει μπορεί να το πράξει με τα εργαλεία Checkstyle, PMD και Sonarqube ενεργοποιώντας τους κανόνες που αφορούν τις αντίστοιχες μετρικές. Το εργαλείο LocMetrics περιορίζεται σε μετρικές που αφορούν κυρίως τις γραμμές του κώδικα (αριθμός γραμμών, αριθμός κενών γραμμών,αριθμός γραμμών που είναι σε σχόλια, κλπ). Το SourceMonitor υπολογίζει κυρίως μετρικές που αφορούν τον αριθμό ή το ποσοστό συγκεκριμένων στοιχείων στον κώδικα. (αριθμός γραμμών, αριθμός εντολών, ποσοστό brances, ποσοστό σχόλιων, αριθμός μεθόδων ανά κλάση κλπ). Τα εργαλείο Vil και .NET υπολογίζουν σημαντικές μετρικές για τον .NET κώδικα.

Η μετρική 8 ‘αξιολόγηση αποτελεσμάτων’ αντικατοπτρίζει σε μεγάλο βαθμό και την ‘αξία’ του εργαλείου καθώς είναι η μετρική που αξιολογά την ποιότητα και την ευστοχία των σφαλμάτων που το κάθε ένα εντοπίζει. Κρίνοντας τα εργαλεία από τα αποτελέσματα που επέστρεψαν καταλήγουμε στα εξής συμπεράσματα:

- Το εργαλείο Checkstyle επικεντρώνεται κυρίως σε προγραμματιστικές συμβάσεις και στην εμφάνιση του κώδικα με σκοπό ο κώδικας να μπορεί εύκολα να διαβαστεί και να επαναχρησιμοποιηθεί. Μπορούμε να πούμε ότι το Checkstyle απευθύνεται σε χρήστες που ο σκοπός τους είναι η βελτίωση της εμφάνισης και του στυλ του κώδικα.
- Το PMD επικεντρώνεται κυρίως σε κακές πρακτικές συγγραφής κώδικα. Το κύριο μειονέκτημα του εργαλείου είναι ότι η ενεργοποίηση των κανόνων ελέγχου που ο χρήστης επιθυμεί να ελέγξει τον κώδικα του είναι μία διαδικασία που απαιτεί αρκετό χρόνο καθώς το σύνολο κανόνων ελέγχου είναι πολύ μεγάλο. Με το προκαθορισμένο σύνολο ενεργοποιημένων κανόνων ελέγχου το PMD επιστρέφει αρκετά αποτελέσματα που το μεγαλύτερο πλήθος εξ’ αυτών αναφέρεται σε προειδοποιήσεις υψηλής προτεραιότητας και πληροφοριακές επισημάνσεις και δεν παρουσιάζουν κάποιο ιδιαίτερο ενδιαφέρον.
- Το Sonarqube συνδυάζει τους κανόνες ελέγχου των υπόλοιπων εργαλείων και επιστρέφει πληθώρα αποτελεσμάτων από μικρά στιλιστικά λάθη μέχρι κρίσιμα σχεδιαστικά λάθη. Το Sonarqube μπορεί να θεωρηθεί ως το κατάλληλο

εργαλείο για ομάδες χρηστών που αναπτύσσουν μαζί ένα έργο λογισμικού και επιθυμούν να ελέγξουν τον κώδικα στατικά. Επίσης απευθύνεται σε οργανισμούς, εταιρίες, χρήστες που αναπύσουν έργα λογισμικού σε διάφορες γλώσσες προγραμματισμού ή αναπτύσουν έργα λογισμικού που συνδιάζουν περισσότερες από μία γλώσσες. Οι επιπρόσθετες λειτουργίες του (κοινοποίηση αποτελεσμάτων, σύγκριση 2 αναλύσεων του ιδίου έργου, κλπ) το καθιστούν ένα δυνατό εργαλείο στατικής ανάλυσης κώδικα σε συνδυασμό με τον σημαντικό αριθμό γλωσσών προγραμματισμού που υποστηρίζει.

- Το FindBugs επικεντρώνεται στον εντοπισμό σοβαρών λαθών και θέματα απόδοσης που πιθανόν να δημιουργήσουν σημαντικά λάθη κατά την ώρα της εκτέλεσης. Η δυνατότητα κοινοποίησης των αποτελεσμάτων της ανάλυσης μπορεί να βοηθήσει τους χρήστες στην καλύτερη κατανόηση των αποτελεσμάτων και στην γρηγορότερη διόρθωση των λαθών. Επίσης τα αποτελέσματα μπορούν να κατηγοριοποιηθούν ανάλογα με τον βαθμό εμπιστοσύνης που έχουν, δηλαδή αν κάποιο αποτελέσμα ανήκει στην κατηγορία ψηλή προτεραιότητα (high confidence) τότε το εργαλείο εγγυάται την ορθότητα του. Γενικά το Findbugs αποδεικνύεται ένα χρήσιμο εργαλείο για την γλώσσα προγραμματισμού Java καθώς εντοπίζει σφάλματα εύστοχα και χρήσιμα και με περισσότερη ακρίβεια.
- Το CppCheck υπολογίζει σημαντικά λάθη (memory leaks, null pointer dereference, κλπ) για τις γλώσσες προγραμματισμού C και C++ ωστόσο επιστρέφει λανθασμένα αποτελέσματα και έτσι η χρήση του απαιτεί μεγάλη προσοχή από τον προγραμματιστή.
- Το Ndepend επικεντρώνεται κυρίως σε μετρικές ποιότητας κώδικα και στις εξαρτήσεις που υπάρχουν μεταξύ των assembly αρχείων. Είναι ένα χρήσιμο εργαλείο και απευθύνεται κυρίως για εταιρίες και οργανισμούς. Η εμπορική άδεια ίσως να περιορίζει τους απλούς προγραμματιστές.

Τέλος καταλήγουμε ότι η επιλογή των εργαλείων εξαρτάται συνήθως από τις ανάγκες του χρήστη. Αν ο χρήστης επιθυμεί να μετρήσει την πολυπλοκότητα και την ποιότητα του κώδικα τότε τα εργαλεία που υπολογίζουν μόνο μετρικές ποιότητας κώδικα μπορούν να καλύψουν τις ανάγκες του. Αν επιθυμεί να βελτιώσει την εμφάνιση και την δομή του κώδικα μπορεί να επιλέξει τα ανάλογα εργαλεία. Αν επιθυμεί να εντοπίσει

πιθανά λάθη στον κώδικα θα πρέπει να χρησιμοποιήσει τα ανάλογα εργαλεία που εφαρμόζουν κανόνες ελέγχου, κλπ.

Εργαλείο	Γλώσσες Προγραμματισμού	Διαδικασία Ανάλυσης/ Διεπαφή με τον χρήστη	Επεξήγηση αποτελεσμάτων	Εξαγωγή αναφορών ή/και στ.διαγ/τα	Δυνατότητα Προσαρμογής	Κοινοποίηση αποτελεσμάτων	Μετρικές Ποιότητας Κώδικα	Κανόνες ελέγχου	Λανθασμένα Αποτελέσματα
Metrics 1.3.6	Java	Εύκολη/ Καλή	Ελλιπής	✓			✓		
PMD	Java, JavaScript, XML, XSL	Εύκολη/ Καλή	Επαρκής	✓	✓		✓	✓	✓
Checkstyle	Java	Εύκολη/ Καλή	Ελλιπής	✓	✓		✓	✓	
FindBugs	Java	Εύκολη/ Καλή	Επαρκής	✓	✓	✓		✓	✓
Ckjm	java	Εύκολη/ Κακή	Ελλιπής				✓		
Sonarqube	Java, C/C++, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, VB.NET,Erlang, Visual Basic 6, XML,WEB	Χρονοβόρα/ Καλή	Επαρκής	✓	✓	✓	✓	✓	✓
CppCheck	C,C++	Εύκολη/ Καλή	Ελλιπής	✓	✓			✓	✓
SourceMonitor	-C++, C, C#, VB.NET, Java, Delphi, Visual Basic (VB6) HTML	Εύκολη/ Καλή	Ελλιπής	✓			✓		
LocMetrics	-C#, Java, C++, Sql.	Εύκολη/ Καλή	Ελλιπής	✓			✓		
NDepend	.NET	Εύκολη/ Καλή	Επαρκής	✓	✓		✓	✓	✓
Vil	.NET	Εύκολη/ Καλή	Ελλιπής	✓	✓		✓		

Πίνακας 4.25: Μετρικές Αξιολόγησης εργαλείων

Κεφάλαιο 5

Συμπεράσματα

5.1 Εισαγωγή

Στην παρούσα διπλωματική εργασία περιγράφηκε μία σύγκριση 11 εργαλείων στατικής ανάλυσης κώδικα χρησιμοποιώντας ένα σύνολο έργων λογισμικού. Παρουσιάστηκε η εφαρμογή και η χρήστη των εργαλείων και επεξηγήθηκε ένα σύνολο σφαλμάτων/παραβιάσεων που εντοπίστηκαν. Τα εργαλεία αξιολογήθηκαν με βάση 8 κριτήρια: υποστηριζόμενες γλώσσες, διεπαφή με τον χρήστη, επεξήγηση αποτελεσμάτων, εξαγωγή αναφορών, δυνατότητα προσαρμογής, δυνατότητα κοινοποίησης αποτελεσμάτων, μετρικές ποιότητας κώδικα, αξιοπιστία αποτελεσμάτων. Τέλος περιγράφηκε σε μικρότερη έκταση η δυναμική ανάλυση κώδικα.

5.2 Συμπεράσματα

Με την ολοκλήρωση της διπλωματικής εργασίας καταλήξαμε στα εξής γενικά συμπεράσματα.

1. Η χρήση των εργαλείων εξαρτάται από τις ανάγκες του χρήστη ή την ομάδα. Το εργαλείο Sonarqube συνδιάζει τα εργαλεία Findbugs, PMD , Checkstyle και άλλα εργαλεία και υποστηρίζει πληθώρα γλωσσών προγραμματισμού. Εντοπίζει από μικρά έως σημαντικά λάθη. Μια ενδιαφέρουσα πρόταση για χρήστες που δεν επιθυμούν την σύνδεση τους στον sonar διακομιστή θα μπορούσε να είναι η συνδυαστική χρήση των εργαλείων. Για παράδειγμα η συνδυαστική χρήση του FindBugs με το PMD καλύπτει και τον εντοπισμό πιθανών σοβαρών λαθών και κακών πρακτικών, ή η συνδιαστική χρήση του εργαλείου FindBugs με το Checkstyle καλύπτει και τον εντοπισμό σοβαρών λαθών που θα βελτιώσουν την

απόδοση του προγράμματος και την βελτίωση της εμφάνισης του κώδικα που θα κάνουν τον κώδικα πιο ευανάγνωστο.

2. Τα εργαλεία στατικής ανάλυσης κώδικα εντοπίζουν από μικρά έως σημαντικά λάθη στον κώδικα και διευκολύνουν την προσπάθεια εξασφάλισης ποιοτικού κώδικα. Παρέχουν μια ένδειξη της πολυπλοκότητας του κώδικα και συνεπώς η εφαρμογή τους διευκολύνουν την ανάγνωση, την επαναχρησιμοποίηση και την συντήρηση του κώδικα. Επίσης η χρήση τους, βοηθά τους προγραμματιστές να ‘μάθουν’ καλές πρακτικές συγγραφής κώδικα.
3. Υπάρχουν και περιπτώσεις που το μεγάλο ποσοστό λανθασμένων αποτελεσμάτων σε συνδυασμό με το μεγάλο πλήθος αποτελεσμάτων καθιστούν τη διαδικασία διερεύνησης αρκετά απαιτητική σε χρόνο και με λίγα ουσιαστικά αποτελέσματα.
4. Σε κάθε περίπτωση η στατική ανάλυση του κώδικα ενός έργου λογισμικού είναι μία χρήσιμη τακτική που μπορεί να οδηγήσει σε υψηλότερης ποιότητας συστήματα λογισμικού.
5. Η δυναμική ανάλυση κώδικα μπορεί να βοηθήσει στον εντοπισμό σημαντικών λαθών.

5.3 Μελλοντική εργασία

Ως μελλοντική εργασία σε σχέση με αυτή την διπλωματική εργασία θα μπορούσε να είναι η επέκταση του κεφαλαίου 5, δηλαδή η δυναμική ανάλυση κώδικα και η σύγκριση εργαλείων δυναμικής ανάλυσης κώδικα. Η δυναμική ανάλυση είναι μία άλλη τεχνική που εφαρμόζεται κατά την εκτέλεση του λογισμικού και συμβάλλει στην εξασφάλιση ποιοτικού κώδικα. Η δυναμική ανάλυση μπορεί να επιστρέψει run-time λάθη όπως διαρροή μνήμης (memory leaks), σφάλματα που αφορούν την ασφάλεια (security errors), σφάλματα που αφορούν νήματα και τον συγχρονισμό τους (threads και synchronization errors), exceptions errors, κλπ. Η μελέτη και η σύγκριση διαφόρων εργαλείων δυναμικής ανάλυσης που υπάρχουν στην βιομηχανία θα βοηθήσει στην καλύτερη κατανόηση της δυναμικής ανάλυσης και τα πλεονεκτήματα της στον εντοπισμό σημαντικών λαθών στον κώδικα.

Βιβλιογραφία

[1] Accessor, https://github.com/jiangmingxia/java

[2] American Society for Quality Control

[3] Android SDK http://developer.android.com/sdk/index.html

[4] Application Programming Interface

[5] BlackWindowSpider, https://github.com/zcqmanking/Java/tree/master/BlackWindowSpider

[6] Burn Oliver

[7] Chidamber, S.R.; Kemerer, C.F. IEEE Transactions on Software Engineering Volume 20, Issue 6, Jun 1994

[8] Crosby Philip B. http://en.wikipedia.org/wiki/Philip_B._Crosby

[9] DotNetOpenAuth, https://github.com/DotNetOpenAuth/DotNetOpenAuth

[10] Feigenbaum, A.V. 'Total quality control', 1983

[11] Fenton, N. (1997) 'Software Metrics - A Rigorous and Practical Approach'

[12] Garvin David, 1984 ‘What Does “Product Quality” Really Mean’

[13] GitHub, https://github.com/

[14] Halstead http://en.wikipedia.org/wiki/Halstead_complexity_measures

[15] Hovemeyer David, FindBugs 2003 University of Maryland,

[16] ISO 9126 http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749

[17] ISO 8402 http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=20115

[18] Java Language specification http://docs.oracle.com/javase/specs/jls/se8/jls8-diffs.pdf

[19] Joseph Juran, 1980, http://en.wikipedia.org/wiki/Joseph_M._Juran

[20] Karp-Rabin, algorithm http://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm

[21] K-Meleon, http://kmeleon.sourceforge.net/

[22] Libgit2, https://github.com/libgit2/libgit2

[23] Marjamäki Daniel, CppCheck developer

[24] McCabe, http://en.wikipedia.org/wiki/Cyclomatic_complexity

[25] McCall Jim, Paul Richards, Giene Walters. Factors in Software Quality. Springfield, 1976.

[26] Mustache, https://github.com/bobthecow/mustache.php

[27] NIST, 2002 'Risk Management Guide for Information Technology Systems'.

[28] Paugh, http://en.wikipedia.org/wiki/William_Pugh

[29] PHPExcel, https://github.com/PHPOffice/PHPExcel

[30] Redis, https://github.com/antirez/redis

[31] Remote Control Center, http://sourceforge.net/projects/remotectrlctr/files/Remote%20Control%20Center/4.0.003/

[32] Sauer Frank http://metrics.sourceforge.net/

[33] SonarSource http://www.sonarsource.com/