ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Διπλωματική Εργασία

Ανάλυση της Εξέλιξης του Τεχνικού Χρέους
Evolution Analysis of Technical Debt

Χρήστος Σαρηκυριακίδης

1ος Επιβλέπων Καθηγητής: **Δρ. Σταματία Μπίμπη**

2ος Επιβλέπων Καθηγητής: **Δρ. Απόστολος Αμπατζόγλου**

Κοζάνη | Ιούλιος 2017

Περιεχόμενα

Κατάλογος Εικόνων 3
Κατάλογος Πινάκων 4
Κατάλογος Εξισώσεων 5
Κατάλογος Διαγραμμάτων 6
Πρόλογος 7
Δήλωση πνευματικών δικαιωμάτων 8
Ευχαριστίες 9
Περίληψη 10
Abstract 11
Κεφάλαιο 1ο – Εισαγωγή 12
1.1 Αντικείμενο Διπλωματικής Εργασίας 12
1.2 Κίνητρο Διεξαγωγής Εργασίας 13
1.3 Συνεισφορά στην επιστημονική κοινότητα και παρόμοιες εργασίες 14
1.4 Σύνοψη διπλωματικής εργασίας 14
Κεφάλαιο 2ο - Θεωρητικό Υπόβαθρο 15
2.1 Γλώσσες Προγραμματισμού 15
2.1.1 Java 15
2.1.2 Perl 19
2.2 Εργαλεία και Βιβλιοθήκες 21
2.2.1 SonarQube 21
2.2.2 MySQL 26
2.2.3 Βιβλιοθήκες JDBC και DBI 28
2.2.4 Git && Github 29
2.3 Πλατφόρμες Ανάπτυξης Λογισμικού 30
2.3.1 Netbeans 30
2.3.2 Notepad++ 31
2.3.3 MySQL Workbench 32
2.3.4 Sublime Text editor 33
2.4 Ανάλυση μεθοδολογίας 34
2.4.1 Υπολογισμός του κεφαλαίου 34
2.4.2 Υπολογισμός του τόκου 35

2.4.3 Συνάρτηση αποτελεσματικότητας 38
Κεφάλαιο 3ο - Ανάπτυξη λογισμικού του συστήματος 39
3.1 Ανάλυση ενός έργου με το εργαλείο SonarQube 39
3.2 Ανάλυση μετρικών από το εργαλείο metrics_calculator 41
3.3 Υπολογισμός του σημείου ισορροπίας 43
3.3.1 Υπολογισμός του σημείου ισορροπίας σε επίπεδο κλάσης 43
3.3.2 Υπολογισμός του σημείου ισορροπίας σε επίπεδο πακέτου 45
3.3.3 Υπολογισμός του σημείου ισορροπίας σε επίπεδο έργου 46
3.4 Παρουσίαση του εργαλείου και παραδείγματα εκτέλεσης 46
3.5 Μετρικές Προγράμματος 52
Κεφάλαιο 4ο – Συμπεράσματα και Μελλοντικές Επεκτάσεις 53
4.1 Αποτελέσματα 54
4.2 Παρατηρήσεις και σχόλια 61
4.3 Μελλοντικές Επεκτάσεις 62
Βιβλιογραφία – Αναφορές 64

Κατάλογος Εικόνων

Εικόνα 1: Αρχιτεκτονική της πλατφόρμας Sonar 24
Εικόνα 2: Εμφάνιση μετρικών 25
Εικόνα 3: Ανίχνευση σφαλμάτων Sonar 25
Εικόνα 4: Γενική δομή μιας σχεσιακής βάσης δεδομένων. 27
Εικόνα 5: Σύνδεση στη βάση δεδομένων με τη βιβλιοθήκη JDBC 28
Εικόνα 6: Σύνδεση στη βάση δεδομένων με τη βιβλιοθήκη DBI 29
Εικόνα 7: Περιβάλλον λογισμικού Netbeans 31
Εικόνα 8: Γραφικό περιβάλλον του λογισμικού Notepad++ 32
Εικόνα 9: Περιβάλλον εργασίας MySQL Workbench. 33
Εικόνα 10: Περιβάλλον εργασίας του Sublime Text Editor 33
Εικόνα 11: Μορφή αρχείου Gitlog 40
Εικόνα 12: Μορφή αρχείο διαμόρφωσης για το πρόγραμμα Sonar 40
Εικόνα 13: Γραφικό περιβάλλον για το script σε Perl 41
Εικόνα 14: Γραφικό περιβάλλον για την ανάλυση του δεύτερου εργαλείου 41
Εικόνα 15: Γραφικό περιβάλλον για τον υπολογισμό του bp 43
Εικόνα 16: Αρχική φόρμα εργαλείου 47
Εικόνα 17: Μήνυμα όταν το git repository είναι κενό ή δεν περιέχει κατάληξη .git 48
Εικόνα 18: Γραφικό περιβάλλον για τον υπολογισμό του bp 49
Εικόνα 19: Παράδειγμα εκτέλεσης του εργαλείου σε επίπεδο κλάσης 50
Εικόνα 20: Παράδειγμα εκτέλεσης του εργαλείου σε επίπεδο πακέτου 51
Εικόνα 21: Παράδειγμα εκτέλεσης του εργαλείου σε επίπεδο πακέτου παλαιότερης έκδοσης 51

Κατάλογος Πινάκων

Πίνακας 1: Ημερομηνία εκδόσεων των πακέτων της Java. 16
Πίνακας 2: Λειτουργίες σε C++ και Java 18
Πίνακας 3: Ενδεικτικός πίνακας αποτελεσμάτων metrics_calculator 42
Πίνακας 4: Πίνακας με τις πιο όμοιες κλάσεις 44
Πίνακας 5: Πίνακας με τα πιο όμοια πακέτα 45
Πίνακας 6: Δεδομένα ανάλυσης για το πακέτο 'src/com/redomar/game' 55
Πίνακας 7: Δεδομένα ανάλυσης για το πακέτο 'src/com/redomare/game/lib' 56
Πίνακας 8: Δεδομένα ανάλυσης για τη κλάση AnimatedTile.java 58
Πίνακας 9: Δεδομένα ανάλυσης για τη κλάση InputHandler.java 58
Πίνακας 10: Δεδομένα ανάλυσης για τη κλάση Game.java 61
Πίνακας 11: Δεδομένα ανάλυσης σε επίπεδο έργου 61

Κατάλογος Εξισώσεων

Εξίσωση 1: Προσπάθεια συντήρησης 36
Εξίσωση 2: Αναλογία βέλτιστου με πραγματικού σχεδιασμού 36
Εξίσωση 3: Εξίσωση για τον συσσωρευμένο τόκο 37
Εξίσωση 4: Εξίσωση για τον υπολογισμό του σημείου ισορροπίας 37
Εξίσωση 5: Υπολογισμός ομοιότητας 43
Εξίσωση 6: Υπολογισμός συνάρτησης αποτελεσματικότητας 44

Κατάλογος Διαγραμμάτων

Διάγραμμα 1: Προσπάθεια αποπληρωμής και προσπάθεια συντήρησης. 35
Διάγραμμα 2: Συσσωρευμένος τόκος και προσπάθεια αποπληρωμής. 53
Διάγραμμα 3: Εξέλιξη τεχνικού χρέους για το πακέτο ‘src/com/redomar/game’ 56
Διάγραμμα 4: Εξέλιξη τεχνικού χρέους για το πακέτο ‘src/com/redomar/game’ 57
Διάγραμμα 5: Εξέλιξη τεχνικού χρέους για τη κλάση ‘Animated.java’ 57
Διάγραμμα 6: Εξέλιξη κεφαλαίου για τη κλάση ‘Mob.java’ 59
Διάγραμμα 7: Εξέλιξη τόκου για τη κλάση ‘Mob.java’ 59
Διάγραμμα 8: Εξέλιξη του τεχνικού χρέους για τη κλάση ‘Mob.java’ 60

Πρόλογος

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι η εμπειρική διερεύνηση της εξέλιξης του σημείου ισορροπίας (breaking point) του τεχνικού χρέους με τη βοήθεια ενός εργαλείου που κατασκευάστηκε για τον υπολογισμό του. Το breaking point είναι το χρονικό σημείο κατά το οποίο το όφελος που αποκτήθηκε από την απελευθέρωση του λογισμικού με τεχνικούς συμβιβασμούς, έχει πληρωθεί από την εταιρεία λόγω της πρόσθετης προσπάθειας για την συντήρηση του λογισμικού.

Δήλωση πνευματικών δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1968 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Ανάλυση της εξέλιξης του τεχνικού χρέους" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Σταματία Μπίμπη αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή /και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ' ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται στο συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Copyright (C) Χρήστος Σαρηκυριακίδης - Σταματία Μπίμπη - Απόστολος Αμπατζόγλου, 2017, Κοζάνη

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους επιβλέποντες καθηγητές της διπλωματικής μου εργασίας, τη κα. Σταματία Μπίμπη και τον κ. Απόστολο Αμπατζόγλου για τη καθοδήγηση και την ουσιαστική βοήθεια που μου παρείχαν κατά τη διάρκεια εκπόνησης αυτής της διπλωματικής εργασίας. Επίσης, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στην οικογένεια μου, για την ηθική και οικονομική υποστήριξη που μου προσέφεραν όλα αυτά τα χρόνια.

Περίληψη

Στην οικονομία, όταν κάποιος δανείζεται ένα κεφάλαιο αυτόματα επιβαρύνεται με ένα χρέος, ο τόκος του οποίου συνήθως είναι μικρότερος από το αρχικό κεφάλαιο που δανείστηκε. Ωστόσο, στο πλαίσιο του τεχνικού χρέους, το συσσωρευμένο χρέος με τη μορφή τόκου σε μερικές περιπτώσεις μπορεί πολύ εύκολα να συναθροιστεί σε μια ποσότητα, η οποία κάποια στιγμή γίνεται μεγαλύτερη από την προσπάθεια που καταβάλλεται για την αποπληρωμή του αρχικού χρέους. Στη παρούσα διπλωματική θα παρουσιαστεί ένας κατά προσέγγιση υπολογισμός του σημείου ισορροπίας του τεχνικού χρέους. Στο σημείο αυτό αξίζει να σημειωθεί ότι η πρόβλεψη για το πόσο αργά αναμένεται να συμβεί το σημείο ισορροπίας, μπορεί να στηρίξει την λήψη αποφάσεων όσον αφορά τις επενδύσεις για τη βελτίωση της ποιότητας. Η προσέγγιση που πραγματοποιήθηκε βασίζεται σε ένα ερευνητικό εργαλείο βελτιστοποίησης που είναι ικανό να προσδιορίσει την απόσταση μιας πραγματικής αντικειμενοστραφούς σχεδίασης στο αντίστοιχο βέλτιστό της.

Για τη κατασκευή του εργαλείου υπολογισμού του breaking point χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java. Το εργαλείο αυτό μπορεί να υπολογίσει το σημείο ισορροπίας σε τρία επίπεδα:

- Class
- Package
- Project

Λέξεις Κλειδιά: Τεχνικό χρέος, Σημείο ισορροπίας, Τεχνολογία λογισμικού, Ποιότητα Λογισμικού, Τόκος

Abstract

In classic economic, when borrowing an amount of money that causes a debt to debtor, it is not usual to have interest which can become larger than the principal. In the context of technical debt however, accumulated debt in the form of interest can in some cases quickly sum up to an amount that at some point, becomes larger than the effort required to repay the initial amount of technical debt. In this dissertation, we propose an approach for estimating this breaking point. Anticipating how late the breaking point is expected to come can support decision making with respect to investments on improving quality. The approach is bases on a search-based optimization tool that is capable of identifying the distance of an actual object-oriented design to the corresponding optimum one.

We used the programming language "Java" to construct the tool for estimation of breaking point. This tool can estimate the breaking point on 3levels: class, package or project.

Keywords: Technical debt, breaking point, Software engineering, Software quality, Interest

Κεφάλαιο 1ο – Εισαγωγή

1.1 Αντικείμενο Διπλωματικής Εργασίας

Η συντήρηση ενός λογισμικού είναι μία από τις εντατικότερες δραστηριότητες στον τομέα της τεχνολογίας λογισμικού. Εκτιμάται ότι οι δραστηριότητες συντήρησης καταναλώνουν το 50-75% της συνολικής προσπάθειας για την ολοκλήρωση ενός τυπικού προγράμματος λογισμικού [4]. Από όλες τις δραστηριότητες συντήρησης είναι εύκολο να κατανοηθεί ότι οι αιτήσεις για αλλαγές σχετικά με την «Προσθήκη λειτουργικότητας», «Προσαρμογή σε νέα περιβάλλοντα», «Βελτίωση του χρόνου εκτέλεσης» και η διόρθωση σφαλμάτων είναι δύσκολο να παραμεληθούν ή να αναβληθούν. Αντιθέτως, αλλαγές που δεν σχετίζονται άμεσα με την εξωτερική συμπεριφορά του αλλά σχετίζονται με τις ιδιότητες του χρόνου σχεδιασμού (design-time)[6] συχνά αναβάλλονται προκειμένου να μειωθεί ο χρόνος απελευθέρωσης του λογισμικού και να μειωθεί το βραχυπρόθεσμο κόστος. Ωστόσο, τα συστήματα λογισμικού είναι εξαιρετικά εξελισσόμενα προϊόντα των οποίων η ποιότητα χρόνου σχεδιασμού σταδιακά εξασθενεί [5]. Για αυτό το λόγο η αναβολή των δραστηριοτήτων συντήρησης (π.χ. αναδομήσεις (refactorings), επίλυση κακών «μυρωδιών» (resolution of bad smells), αντίστροφη μηχανική (reverse engineering)), μπορεί να έχει σημαντικό αντίκτυπο σε αρκετές ιδιότητες της ποιότητας όπως η συντήρηση, η κατανόηση, η επαναχρησιμοποίηση και άλλες πολύ σημαντικές ιδιότητες. Η στρατηγική αυτή οδηγεί στη δημιουργία οικονομικού χρέους λόγω της υποβαθμισμένης ποιότητας, η οποία ονομάστηκε τεχνικό χρέος από τον Ward Cunningham το 1992 [7].

Το τεχνικό χρέος (TD) είναι μια μεταφορική έννοια που χρησιμοποιείται για να αναδείξει μια αναλογία μεταξύ του οικονομικού χρέους, όπως ορίζεται στα οικονομικά, και της κατάστασης όταν ένας οργανισμός αποφασίζει να παράγει μη ολοκληρωμένα προϊόντα λογισμικού προκειμένου να παραδοθεί το προϊόν στην αγορά βραχυπρόθεσμα [7]. Το τεχνικό χρέος συσσωρεύεται κατά τη διάρκεια όλων των φάσεων ανάπτυξης (ανάλυση απαιτήσεων, σχεδιασμός αρχιτεκτονικής, εφαρμογή) και για αυτό το λόγο πρέπει να παρακολουθείται και να χειρίζεται κατά τη διάρκεια του κύκλου ζωής του λογισμικού [8]. Στη πράξη, ενώ το τεχνικό χρέος είναι μερικές φορές επιθυμητό (π.χ. σε περιπτώσεις που οι εταιρίες επιλέγουν να επενδύσουν σε ένα διαφορετικό προϊόν αντί να παράγουν ένα νέο βελτιωμένο προϊόν) η τελική αποπληρωμή του θεωρείται μη ρεαλιστική [9].

Ο όρος τεχνικό χρέος είναι ένας ενεργός τομέας καθώς το μεγαλύτερο ποσοστό των ερευνών είναι μετά το 2010. Ως εκ τούτου, πραγματοποιούνται πολλές μελέτες για το συγκεκριμένο θέμα ακόμα και τώρα. Επίσης, χρησιμοποιείται τόσο σε ακαδημαϊκά όσο και σε βιομηχανικά πλαίσια. Ένα από τα πιο σημαντικά χαρακτηριστικά του τεχνικού χρέους είναι η διεπιστημονική φύση του καθώς συνδυάζει στοιχεία από τη θεωρία των οικονομικών αλλά και από τη θεωρία της τεχνολογίας λογισμικού. Για να πραγματοποιηθεί λοιπόν μια έρευνα για το τεχνικό χρέος, δανείζονται κάποιες έννοιες από τα οικονομικά και τα χαρτογραφούν σε τεχνολογία λογισμικού. Οι δυο βασικότεροι οικονομικοί όροι που έχουν δανειστεί για την μελέτη του τεχνικού χρέους είναι ο τόκος (interest) και το κεφάλαιο (principal) [3,11]. Το κεφάλαιο είναι μια καθορισμένη και αποδεκτή έννοια και χαρακτηρίζεται ως η προσπάθεια που απαιτείται για την αντιμετώπιση της διαφοράς μεταξύ του τρέχοντος και του βέλτιστου επιπέδου ποιότητας λογισμικού σε ένα πλήρες σύστημα [3]. Από την άλλη η έννοια interest

είναι μια έννοια η οποία έχει αναλυθεί από πολλούς ερευνητές και έχει οδηγήσει πολλές φορές σε αμφιλεγόμενα αποτελέσματα. Επιπρόσθετα, το interest είναι μια έννοια η οποία δεν μπορεί να προσδιοριστεί εύκολα καθώς περιλαμβάνει την εκτίμηση μελλοντικών δραστηριοτήτων συντήρησης. Παρόλο που έχει αναλυθεί αρκετά η μελέτη για τον υπολογισμό του και έχουν γίνει αρκετές προσεγγίσεις επί του θέματος, δεν έχουν καταφέρει οι ερευνητές να το προσδιορίσουν με ακρίβεια και έτσι βρίσκονται ακόμα σε θεωρητικό στάδιο Στην εργασία αυτή εστιάζουμε στην θεωρητική δομή FITTED (Framework for managing Interest in Technical Debt) η οποία έχει αναλυθεί από Αμπατζόγλου [12]. Ουσιαστικά, πρόκειται για μια μελέτη η οποία παρουσίασε μια θεωρία για τον τόκο του τεχνικού χρέους, η οποία βασίστηκε πάνω σε οικονομικές θεωρίες σχετικά με τον τόκο. Η οικονομική θεωρία στην οποία βασίστηκε η παραπάνω μελέτη ήταν Liquidity Preference Theory [13], η οποία υποστηρίζει ότι ο τόκος (interest) καθορίζεται από το σημείο ισορροπίας της αγοράς. Όπως έχει αναφερθεί η συνάρτηση αποτελεσματικότητας, το συσσωρευμένο interest μπορεί ενδεχομένως να ανέρχεται σε ένα ποσό που γίνεται μεγαλύτερο από την προσπάθεια που απαιτείται για την αποπληρωμή του τεχνικού χρέους που αποκτήθηκε σε προγενέστερα στάδια της ανάπτυξης (κεφάλαιο). Υπό αυτές τις συνθήκες, οι διαχειριστές έργων θα πρέπει να είναι σε θέση να υπολογίσουν το χρονικό σημείο που ο συσσωρευμένος τόκος (interest) θα φτάσει το αντίστοιχο πόσο του κεφαλαίου (principal). Το συγκεκριμένο λοιπόν χρονικό σημείο στο οποίο το interest θα ισούται με το principal ονομάζεται σημείο ισορροπίας (breaking point). Στο breaking point οποιαδήποτε κέρδη έχουν μαζευτεί από την απόφαση να μην αποπληρωθεί το τεχνικό χρέος θα εξαφανιστούν λόγω της αυξημένης προσπάθειας για συντήρηση κατά την εξέλιξη του προϊόντος [12]. Το σημείο ισορροπίας ερμηνεύεται ως ο λόγος του κεφαλαίου (principal) προς τη μέση τιμή του τόκου (interest) ανά έκδοση. Ουσιαστικά σε κάθε ολοκληρωμένο σύστημα γίνονται κάποιες αλλαγές έτσι ώστε να αυξηθεί η ποιότητα του κώδικα και να βελτιστοποιηθεί το σύστημα. Η απόσταση μεταξύ του πραγματικού και του βέλτιστου συστήματος χρησιμοποιείται σαν δείκτης για το κεφάλαιο του τεχνικού χρέους. Αντίστοιχα, ο λόγος της ποιότητας του βέλτιστου προς την πραγματική σχεδίαση χρησιμοποιείται για τον υπολογισμό του interest [1].

Επομένως η συγκεκριμένη διπλωματική έχει ως στόχο τη δημιουργία του εργαλείου που θα είναι ικανό να υπολογίζει το breaking point. Ο υπολογισμός του breaking point θα μπορεί να είναι είτε σε επίπεδο κλάσης είτε σε επίπεδο πακέτου αλλά και σε επίπεδο project. Επίσης με τη βοήθεια του εργαλείου θα παρατηρήσουμε τη μεταβολή του breaking point, δηλαδή αν έρχεται πιο κοντά ή αν απομακρύνεται κατά την προσθήκη μιας αλλαγής στην ποιότητα του συστήματος.

1.2 Κίνητρο Διεξαγωγής Εργασίας

Ο όρος τεχνικό χρέος όπως έχει αναφερθεί ήδη είναι ένας πολύ ενεργός τομέας, με το μεγαλύτερο ποσοστό των ερευνών για αυτό το θέμα να έχουν αναπτυχθεί μετά το 2010. Το τεχνικό χρέος ενώνει δύο επιστήμες, την τεχνολογία λογισμικού και τα οικονομικά, και η ανάλυση του μπορεί να βοηθήσει σε μελλοντικές αποφάσεις σχετικά με το προϊόν αλλά και σε τυχών επενδυτικές αποφάσεις.

Η ανάληψη του παρόντος θέματος ήταν η αφορμή για να κατανοηθεί πως δυο σημαντικές επιστήμες μπορούν να ενωθούν και να προκαλέσουν ένα πολύ σημαντικό

αποτέλεσμα. Επίσης ήμουν αρκετά περίεργος για τις έρευνες που έχουν γίνει πάνω στους δυο τομείς και που μπορεί να βοηθήσει η εξέλιξη του, για αυτό και ήθελα να ασχοληθώ με αυτό. Τέλος, το βασικότερο κίνητρο ήταν η εμπλούτιση των γνώσεων μου για το τομέα της τεχνολογίας λογισμικού που μου τραβάει το ενδιαφέρον σε όλη τη διάρκεια του πανεπιστημίου και σίγουρα μετά από αυτό.

1.3 Συνεισφορά στην επιστημονική κοινότητα και παρόμοιες εργασίες

Παρόλο που το τεχνικό χρέος έχει προκαλέσει το ενδιαφέρον αρκετών μελετών έρευνας τα τελευταία χρόνια, η ανάλυση του είναι ακόμα σε θεωρητικά πλαίσια. Σύμφωνα με A. Chatzigeorgiou παρουσιάζεται μια προσεγγιστική μελέτη για την ανάλυση της εξέλιξης του τεχνικού χρέους σε πρακτικά επίπεδα [1]. Με την ολοκλήρωση λοιπόν της εργασίας η οποία αποτελεί την επέκταση του προαναφερθέντος άρθρου θα είμαστε σε θέση να εξάγουμε στοιχεία τα οποία θα μας βοηθήσουν για την εξαγωγή συμπερασμάτων τα οποία θα έχουν προηγηθεί από αναλύσεις και όχι από θεωρίες. Η ολοκλήρωση λοιπόν της εργασίας μπορεί να συνεισφέρει τόσο σε ερευνητικούς όσο και σε πρακτικούς φορείς. Όσον αφορά στο επίπεδο της έρευνας, μπορεί να προκαλέσει ενδιαφέρον η ανάλυση της εξέλιξης του τεχνικού χρέους και να ωθήσει ερευνητές στο να ασχοληθούν με αυτό. Από την άλλη , σε πρακτικούς τομείς μπορεί να βοηθήσει τις εταιρίες στη λήψη κάποιων αποφάσεων (μελλοντικές επενδύσεις), αλλά και να έχουν μια βάση για τη κατάσταση του λογισμικού τους.

1.4 Σύνοψη διπλωματικής εργασίας

Το κείμενο αποτελείται από τέσσερα κεφάλαια και το κάθε ένα από αυτά πραγματεύεται ένα διαφορετικό ζήτημα που ερευνήθηκε κατά την υλοποίηση του συγκεκριμένου εργαλείου. Στα επιμέρους κεφάλαια γίνεται αναλυτική παρουσίαση των εργαλείων που χρησιμοποιήθηκαν και γενικότερα η αναλυτική περιγραφή του συστήματος. Η δομή των υπόλοιπων κεφαλαίων έχει ως εξής: Στο δεύτερο κεφάλαιο της διπλωματικής εργασίας παρουσιάζεται το θεωρητικό υπόβαθρο, στο οποίο γίνεται η ανάλυση όλων των εργαλείων που χρησιμοποιήθηκαν, ώστε να βοηθήσουν στην πλήρη κατανόηση της έρευνας, αλλά και του προαναφερθέντος εργαλείου. Το τρίτο κεφάλαιο αναφέρεται στην προσέγγιση και στην ανάπτυξη του θέματος και θα πραγματοποιηθεί μια πλήρης περιγραφή όλων των φάσεων που χρειάστηκαν μέχρι την ολοκλήρωση της εργασίας. Στο συγκεκριμένο κεφάλαιο θα γίνει αναλυτική περιγραφή της μεθοδολογίας που τηρήθηκε αλλά και η παρουσίαση του εργαλείου με όλες τις δυνατότητες του. Το τελικό κεφάλαιο της παρούσας εργασίας ονομάζεται Συμπεράσματα και Μελλοντικές Επεκτάσεις, και περιγράφει τα αποτελέσματα, μαζί με τα κατάλληλα συμπεράσματα καθώς και πιθανές επεκτάσεις στο μέλλον.

Κεφάλαιο 2ο - Θεωρητικό Υπόβαθρο

Στο ακόλουθο κεφάλαιο παρουσιάζονται οι έννοιες και τα μέσα με τη βοήθεια των οποίων υλοποιήθηκε η παρούσα εργασία. Εκτός από τα εργαλεία λογισμικού τα οποία έχουν χρησιμοποιηθεί, παρουσιάζεται και η μεθοδολογία που χρησιμοποιήθηκε για την ολοκλήρωση του συστήματος.

2.1 Γλώσσες Προγραμματισμού

2.1.1 Java

Η πιο σημαντική συνεισφορά της επανάστασης των μικροεπεξεργαστών είναι ότι έκαναν δυνατή την ανάπτυξη των προσωπικών υπολογιστών, που σήμερα είναι εκατοντάδες εκατομμύρια σε όλο τον κόσμο [14]. Οι προσωπικοί υπολογιστές είχαν έντονη επίδραση στις ζωές των ανθρώπων και στους τρόπους, με τους οποίους οι οργανισμοί διεξάγουν και διαχειρίζονται τις δραστηριότητες τους. Οι μικροεπεξεργαστές επίσης έχουν βαθιά επίδραση στις έξυπνες ηλεκτρονικές συσκευές. Αναγνωρίζοντας το αυτό, η Sun Microsystems χρηματοδότησε ένα εσωτερικό ερευνητικό έργο το οποίο κατέληξε στην ανάπτυξη μιας γλώσσας βασισμένης στην C++, η οποία ονομάστηκε Oak. Στη συνέχεια ανακαλύφθηκε ότι υπήρχε ήδη μια γλώσσα προγραμματισμού Oak, για αυτό μετονομάστηκε σε Java και παρέμεινε ως σήμερα.. Το έργο αυτό συνάντησε κάποιες δυσκολίες, καθώς η αγορά για τέτοιες συσκευές δεν αναπτυσσόταν τόσο γρήγορα στις αρχές της δεκαετίας του ’90, όσο είχε εκτιμήσει η Sun. Από καθαρά καλή τύχη, το 1993 εκτοξεύθηκε η δημοτικότητα του World Wide Web και η Sun είδε την δυνατότητα να χρησιμοποιήσει τη Java για τη προσθήκη δυναμικών περιεχομένων, όπως διαδραστικότητας και κινήσεων παραστάσεων, σε ιστοσελίδες. Αυτή η κίνηση έδωσε νέα πνοή στο έργο. Η Sun λοιπόν ανακοίνωσε τη Java σε ένα συνέδριο το Μάιο του 1995. Η Java καθήλωσε το ενδιαφέρον της επιχειρηματικής κοινότητας λόγω του μεγάλου ενδιαφέροντος για το Web και συνεχίζει να αναπτύσσεται μέχρι σήμερα.

Η τελευταία λοιπόν έκδοση της Java Standard Edition είναι η Java SE 8. Με τη πρόοδο της Java και την ευρεία δημοτικότητα της, δημιουργήθηκαν πολλαπλές διαμορφώσεις για να ταιριάζουν σε διάφορους τύπους πλατφόρμων. Για παράδειγμα η J2EE αναπτύχθηκε για Επιχειρηματικές Εφαρμογές, ενώ η J2ME αναπτύχθηκε για κινητές εφαρμογές. Οι νέες εκδόσεις αυτές μετονομάστηκαν σε Java SE, Java EE και Java ME αντίστοιχα. Σκοπός της Java είναι να επιτρέψει στους προγραμματιστές εφαρμογών να εφαρμόσουν τον όρο ”Write once, run anywhere”, που σημαίνει ότι ένας κώδικας ο οποίος έχει μεταγλωττιστεί μπορεί να εκτελεστεί σε οποιοδήποτε πλατφόρμα η οποία υποστηρίζει τη Java, χωρίς να χρειαστεί να γίνει ξανά η μεταγλώττιση. Οι εφαρμογές της Java συνήθως μεταγλωττίζονται σε bytecodes, τα οποία μπορούν να εκτελεστούν από οποιαδήποτε εικονική μηχανή Java (JVM), ανεξαρτήτως από την αρχιτεκτονική του υπολογιστή. Από το 2016, η Java είναι μια από τις πιο δημοφιλείς γλώσσες και τη χρησιμοποιούν πάνω από 9 εκατομμύρια προγραμματιστές. Η Java χρησιμοποιείται σήμερα για τη δημιουργία ιστοσελίδων με δυναμικά περιεχόμενα, για την ανάπτυξη εφαρμογών μεγάλης κλίμακας, για την βελτίωση της λειτουργικότητας web διακομιστών, για τη δημιουργία εφαρμογών για καταναλωτικές συσκευές (π.χ. κινητά

τηλέφωνα, pagers, και προσωπικούς ηλεκτρονικούς βοηθούς) και για πολλούς άλλους σκοπούς.

Στον Πίνακα 1 που ακολουθεί μπορείτε να δείτε τις εκδόσεις της γλώσσας μαζί με τις ημερομηνίες που απελευθερώθηκαν στην αγορά καθώς και το όνομα της συγκεκριμένης έκδοσης (codename).

Release	**Codename**	**Release-Date**
JDK1.0	Oak	23/1/1996
JDK1.1	None	19/2/1997
J2SE1.2	Playground	8/12/1998
J2SE1.3	Kestrel	8/5/2000
J2SE1.4	Merlin	6/2/2002
J2SE5.0	Tiger	30/9/2004
Java SE 6	Mustang	11/12/2006
Java SE 7	Dolphin	7/7/2011
Java SE 8	Spider	18/3/2014

Πίνακας 1: Ημερομηνία εκδόσεων των πακέτων της Java.

Η Java λοιπόν όπως έχει αναφερθεί ήδη είναι μια πολύ δημοφιλής γλώσσα προγραμματισμού ή οποία συνεχώς εξελίσσεται. Συνοψίζοντας λοιπόν, πρόκειται για μια γλώσσα υψηλού επιπέδου, με τα κύρια χαρακτηριστικά:

- **Αντικειμενοστραφής**: Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού και όλα τα δεδομένα αντιμετωπίζονται ως αντικείμενα.

- **Ανεξαρτησία**: Η Java σε αντίθεση με άλλες γλώσσες προγραμματισμού όπως η C και η C++, όταν μεταγλωττιστεί δεν μετατρέπεται σε γλώσσα μηχανής η οποία εξαρτάται από τη πλατφόρμα, αλλά μετατρέπεται σε bytecodes τα οποία δεν εξαρτώνται από τη πλατφόρμα. Αυτά τα bytecodes διανέμεται στο διαδίκτυο και ερμηνεύεται από την Εικονική μηχανή σε οποιαδήποτε πλατφόρμα εκτελείται. Το χαρακτηριστικό αυτό αφορά λοιπόν τον όρο που εξηγήσαμε πριν “Write once, run anywhere”.

- **Απλή**: Η Java είναι αρκετά απλή για να κατανοηθεί. Εάν έχεις κατανοήσει τον όρο της αντικειμενοστρέφιας, τότε είναι πολύ εύκολο να διαχειριστείς τη συγκεκριμένη γλώσσα.

- **Ασφαλής**: Η Java με την ασφαλή λειτουργία της, επιτρέπει στην ανάπτυξη συστημάτων χωρίς ιούς και χωρίς παραβιάσεις. Οι τεχνικές ασφαλείας που χρησιμοποιεί βασίζονται σε κρυπτογράφηση του δημοσίου κλειδιού.

- **Ουδέτερη Αρχιτεκτονική:** Ο μεταγλωττιστής της Java, δημιουργεί μια ουδέτερη αρχιτεκτονική δομή, η οποία καθιστά τους μεταγλωττισμένους κώδικες ως εκτελέσιμους σε πολλούς επεξεργαστές, με τη παρουσία του συστήματος Java Runtime.

- **Φορητή**: Με την έννοια «φορητότητα», εννοούμε πόσο εύκολα μπορούμε να πάρουμε ένα πρόγραμμα από μια πλατφόρμα και να το τρέξουμε σε μια άλλη. Η Java λοιπόν θεωρείται μια από της πιο φορητές γλώσσες, λόγω της ουδέτερης αρχιτεκτονικής της και των μη εξαρτήσεων από πτυχές συγκεκριμένης πλατφόρμας.

- **Πολυνημάτωση**: Με τη λειτουργία πολυνημάτωσης που παρέχει η Java , καθίσταται δυνατή η συγγραφή προγραμμάτων που μπορούν να εκτελέσουν πολλές εργασίες ταυτόχρονα. Αυτό το σχεδιαστικό χαρακτηριστικό επιτρέπει στους προγραμματιστές να κατασκευάσουν διαδραστικές εφαρμογές που μπορούν να λειτουργούν ομαλά.

- **Interpreted**: Τα bytecode της Java μεταφράζονται σε μη αυτόματες εντολές μηχανής και δεν αποθηκεύονται οπουδήποτε. Η διαδικασία ανάπτυξης είναι ταχύτερη και αναλυτικότερη, καθώς η σύνδεση είναι μια διαδικασία ελαφρού βάρους.

- **Υψηλή Απόδοση**: Με τη χρήση των μεταγλωττιστών Just-In-Time, η Java καθιστά δυνατή την υψηλή απόδοση.

- **Δυναμική**: Η Java θεωρείται πιο δυναμική από τη C ή τη C ++, αφού έχει σχεδιαστεί για να προσαρμόζεται σε ένα εξελισσόμενο περιβάλλον. Τα προγράμματα Java μπορούν να μεταφέρουν εκτεταμένες πληροφορίες χρόνου εκτέλεσης που μπορούν να χρησιμοποιηθούν για την επαλήθευση και την επίλυση προσπελάσεων σε αντικείμενα κατά την εκτέλεση.

Σε όλα τα προηγούμενα κομμάτια έχει αναφερθεί πως η Java είναι μια γλώσσα προγραμματισμού η οποία έχει βασιστεί πάνω στη C++, χωρίς να σημαίνει πως οι δυο γλώσσες είναι ίδιες. Αρκετές λειτουργίες της C++ έχουν διατηρηθεί και στη δομή της Java, αλλά αρκετές λειτουργίες έχουν αφαιρεθεί από αυτή. Στον Πίνακα 2 μπορείτε να δείτε κάποιες βασικές λειτουργίες από την C++, και αν έχουν διατηρηθεί και στη Java. Για παράδειγμα διακρίνουμε ότι οι λειτουργίες των πινάκων έχει διατηρηθεί και στη Java, αλλά η λειτουργία του goto έχει αφαιρεθεί από αυτή και δεν υποστηρίζονται άλλο. Οι παραπάνω λειτουργίες ήταν κάποιες ενδεικτικές από αυτές που θα παρουσιαστούν στον παρακάτω πίνακα, οπότε μπορείτε να δείτε για περισσότερες λειτουργίες της C++ αν διατηρήθηκαν και στη Java.

Features	**C++**	**Java**
Control Statements	Supported	Supported
Functions	Supported	Supported
Arrays	Supported	Supported
Data types	Supported	Supported
Pointer Concept	Supported	Not Supported
Goto Statement	Supported	Not Supported
Unsigned Data Types	Supported	Not Supported
Operator Overloading	Supported	Not Supported
Preprocessor Directives (#define)	Supported	Not Supported
Multiple Inheritance	Supported	Not Supported
Garbage values for unassigned data types	Supported	Not Supported
Memory Management	Explicit	Implicitly
Structures Concept	Supported	Not Supported

Πίνακας 2: Λειτουργίες σε C++ και Java

Οι πιο διάσημες δωρεάν πλατφόρμες για τη συγγραφή προγραμμάτων σε Java είναι:

- Eclipse
- Netbeans
- Notepad++
- Editplus
- JCreator
- BlueJ
- jEdit
- DrJava
-

Αντίστοιχα υπάρχουν πλατφόρμες οι οποίες είναι εξίσου δημοφιλείς με τις προηγούμενες αλλά απαιτούν πληρωμή. Ενδεικτικά κάποιες από αυτές είναι:

- Jbuilder
- Intellij IDEA
- JDeveloper

2.1.2 Perl

Η Perl [15,16] αρχικά γράφτηκε από τον Larry Wall, καθώς δούλευε στα εργαστήρια της NASA. Ο Larry είναι ένας θρύλος του διαδικτύου, καθώς όχι μόνο δημιούργησε τη Perl, αλλά επίσης ήταν ο δημιουργός των βοηθητικών προγραμμάτων για Unix rn και patch. Το rn (read news) είναι ουσιαστικά μια εφαρμογή με την οποία μπορείς να διαβάσεις άρθρα από το Usenet. Αντίστοιχα, το patch είναι ένα αρχείο κειμένου που περιέχει μια λίστα διαφορών και προκύπτει από την εκτέλεση του προγράμματος diff με παραμέτρους το αρχικό και το ενημερωμένο αρχείο.

Ο αρχικός σκοπός της Perl ήταν να λειτουργεί σαν μια 'glue' γλώσσα προγραμματισμού για αυτόν και τους συναδέλφους του. Με τον όρο 'glue' γλώσσα αναφερόμαστε σε μια γλώσσα προγραμματισμού που έχει σχεδιαστεί ειδικά για την εγγραφή προγραμμάτων τα οποία συνδέουν διαφορετικά στοιχεία λογισμικού μεταξύ τους. Η Perl ενσωμάτωσε τις καλές λειτουργίες από πολλές γλώσσες προγραμματισμού. Για παράδειγμα ενσωμάτωσε τις λειτουργίες για κανονικές εκφράσεις από sed (stream editor σε Unix), τη λειτουργία σάρωσης προτύπων από τη γλώσσα awk και άλλες λειτουργίες από άλλες γλώσσες προγραμματισμού. Στη σύνταξη της γλώσσας συνέβαλαν η C, Pascal, Basic, shell γλώσσες από Unix και σίγουρα η αγγλική γλώσσα.

Η πρώτη έκδοση (Perl 1), εμφανίστηκε στις 18 Δεκεμβρίου 1987, και σταδιακά εξελίσσεται μέχρι και σήμερα με τη συνεισφορά αμέτρητων ανθρώπων. Η δεύτερη έκδοση (Perl 2), επέκτεινε τη λειτουργία των κανονικών εκφράσεων, ενώ η τρίτη έκδοση (Perl 3), πρόσθεσε τη δυνατότητα στη γλώσσα να μπορεί να διαχειρίζεται δεδομένα σε δυαδική μορφή. Μέχρι εκείνη τη στιγμή δεν είχε δημιουργηθεί κάποιο επίσημο έγγραφο για τη χρήση της γλώσσας, παρά μόνο μια απλή σελίδα βοήθειας (man page). Η τέταρτη λοιπόν έκδοση (Perl 4), απελευθερώθηκε μαζί με το βιβλίο το οποίο είχαν γράψει για τη διαχείριση της γλώσσας, το οποίο το ονόμασαν "Camel book", χωρίς να προσθέσουν κάποια σημαντική λειτουργία στην καινούργια έκδοση. Η μεγάλη αλλαγή στη γλώσσα, έγινε με την απελευθέρωση της πέμπτης έκδοσης (Perl 5), η οποία είχε πολλές αλλαγές στη σύνταξη σε σχέση με τις προηγούμενες εκδόσεις, αλλά και φανταστικές επεκτάσεις στις λειτουργίες της. Μέχρι και σήμερα λοιπόν γίνονται αναβαθμίσεις πάνω στη συγκεκριμένη γλώσσα και η τελευταία έκδοση που υπάρχει είναι η Perl 5.26 που απελευθερώθηκε στις 30-5-2017. Αξίζει να σημειώσουμε βέβαια πως παράλληλα με την Perl 5, δημιουργήθηκε η Perl 6. Η Perl 6 ξεκίνησε για να επεκτείνει την Perl 5, ωστόσο κατέληξε σε μια διαφορετική γλώσσα. Πλέον υπάρχουν διαφορετικές ομάδες που ασχολούνται με την Perl 5 και με την Perl 6, ανταλλάσσοντας ωστόσο ιδέες μεταξύ τους.

Το όνομα της γλώσσας Perl, δεν αποτελεί κάποιο επίσημο ακρωνύμιο, ωστόσο υπάρχει μια τάση από τους ανθρώπους να δημιουργούν μόνοι τους ακρωνύμια για κάποια λέξη. Κάτι το οποίο έκανε και ο δημιουργός της γλώσσας, ο Larry Wall, έχοντας χρησιμοποιήσει δυο ακρωνύμια για τη Perl. Το πρώτο είναι "Practical Extraction and Reporting Language", και το δεύτερο " Pathologically Eclectic Rubbish Lister". Η γλώσσα αυτή είναι εξαιρετικά πολυεργαλείο για αυτό το λόγο το σλόγκαν της είναι " There's More Than One Way To Do It ", που σημαίνει ότι υπάρχει πάντα ένας ακόμη τρόπος για να το κάνεις. Φυσικά με το παραπάνω γνωμικό θέλουν να τονίσουν την ελευθερία και την ευκαμψία της γλώσσας. Αξίζει να σημειωθεί ότι δε χρειάζεται να ξέρεις όλες τις λεπτομέρειες για τη γλώσσα προκειμένου να επιτύχεις αυτό που χρειάζεσαι.

Τα κυριότερα λοιπόν χαρακτηριστικά της γλώσσας που προσφέρουν πλεονέκτημα έναντι άλλων γλωσσών είναι:

- Είναι πολύ εύκολη για να τη μάθεις, και η κατανόηση της μπορεί να σε οδηγήσει σε σπουδαία πράγματα στο μέλλον.

- Η Perl γράφτηκε για να είναι εύκολη στο να τη γράψουν οι άνθρωποι και όχι για να είναι πιο εύκολο από τους υπολογιστές να τη καταλάβουν. Η σύνταξη της γλώσσας είναι σχεδόν όπως η ομιλία των ανθρώπων, και όχι σαν άλλες γλώσσες με παράξενη και αυστηρή σύνταξη.

- Η Perl είναι αρκετά φορητή (portable). Ο όρος φορητή όπως είναι κατανοητό υποδηλώνει τη δυνατότητα μεταφοράς ενός προγράμματος γραμμένο σε Perl από έναν υπολογιστή σε έναν άλλο, και να εκτελείται με απόλυτη επιτυχία. Η Perl είναι διαθέσιμη σε ποικίλα λειτουργικά συστήματα, έχοντας ως αποτέλεσμα ο κώδικας να εκτελείται παντού χωρίς καμία αλλαγή σε αυτόν.

- Η Perl «μιλάει στο κείμενο». Αυτό που σημαίνει η προηγούμενη πρόταση είναι ότι μπορεί να σκέφτεται λέξεις ή προτάσεις, ενώ άλλες γλώσσες διαχειρίζονται έναν χαρακτήρα τη φορά. Επίσης διαθέτει τρομερές κανονικές εκφράσεις, όπου μπορούν να ανιχνεύσουν ένα πρότυπο σε ένα κείμενο με πάρα πολλούς τρόπους, πολύ εύκολα και πολύ γρήγορα.

- Η Perl είναι μια γλώσσα υψηλού επιπέδου, γενικού σκοπού και δυναμική.

Η Perl χρησιμοποιείται σε πάρα πολλούς τομείς. Η πιο δημοφιλής χρήση της είναι στο CGI (Common Gateway Interface) προγραμματισμό. Η Perl είναι η κύρια δύναμη πίσω από πολύ γνωστές ιστοσελίδες όπως του amazon, Deja, αλλά και του Slashdot. Φυσικά η Perl χρησιμοποιείται από πολλούς για τον αρχικό σκοπό που δημιουργήθηκε, δηλαδή να εξάγει δεδομένα από μια πηγή και να τα μεταφράζει σε μια άλλη μορφή. Επίσης η Perl για τους διαχειριστές συστημάτων, θεωρείται «ελβετικός σουγιάς». Υπάρχουν πάρα πολλές εργασίες όπου χρησιμοποιούν τη γλώσσα αυτή, όπως για παράδειγμα να στέλνουν αυτοματοποιημένα email, για την αναφορά πληροφοριών για τη χρήση του δίσκου, ανίχνευση παραβιάσεων ασφαλείας, για τη παραγωγή αρχείων καταγραφής (αρχεία log) και σε πολλές άλλες εργασίες που χρειάζονται για να στήσουν το σύστημα τους. Στο σημείο αυτό πρέπει να τονίσουμε ότι η χρήση της γλώσσας δεν περιορίζεται μόνο στα παραπάνω. Τείνει η γλώσσα αυτή να γίνει η καθιερωμένη γλώσσα προγραμματισμού για το Internet, καθώς χρησιμοποιείται για το φιλτράρισμα των email, αυτόματη ανάρτηση άρθρων, για τη δημιουργία καθρεπτών μιας ιστοσελίδας, αυτοματοποιημένες λήψεις και μεταφορτώσεις αρχείων και πολλές άλλες καθημερινές εργασίες.

2.2 Εργαλεία και Βιβλιοθήκες

2.2.1 SonarQube

Σε κάθε προσπάθεια δημιουργίας ενός καινούριου κώδικα από τους προγραμματιστές υπάρχει έντονος προβληματισμός σχετικά με το αν αυτή η προσπάθεια τους είναι επιτυχής. Μέχρι πρόσφατα δεν υπήρχαν έγκυρα στοιχεία που να διασφαλίζουν την επιτυχία του εγχειρήματος. Οι πιο συνηθισμένες απαντήσεις στο ερώτημα που είχε τεθεί ήταν: «Ναι, δουλεύει» ή «Ναι, φαίνεται να δουλεύει μια χαρά». Για κάποιο χρονικό διάστημα, οι χρήστες δεν διαμαρτυρόντουσαν και, έτσι, οι παραπάνω απαντήσεις ήταν αρκετές. Όπως ήταν αναμενόμενο έφτασε η στιγμή όπου οι χρήστες άρχισαν να διαμαρτύρονται και οι προγραμματιστές κατανόησαν ότι δεν έγινε σωστά η συγγραφή του κώδικα. Για να επιλύσουν τη κατάσταση αυτή και να ελέγξουν το κώδικα αρκετοί προγραμματιστές ξεκίνησαν να γράφουν αυτόματα test scripts τα οποία θα ελέγχουν μια συγκεκριμένη λειτουργία του συστήματος. Ωστόσο, αποδείχθηκε ότι ούτε αυτό ήταν η λύση γιατί δεν ήταν αντιληπτό πόσα test έπρεπε να δημιουργηθούν, ενώ υπήρχαν λειτουργίες που ούτε τα scripts μπορούσαν να τα εντοπίσουν.

Καθώς οι προγραμματιστές αδυνατούσαν να διασφαλίσουν τον σωστό προγραμματισμό, δημιουργούνταν έντονες διαμαρτυρίες. Η διαδικασία αξιολόγησης ενός προγραμματιστή δεν είναι εύκολη, όπως αντίστοιχα στην περίπτωση ενός πωλητή(που εξαρτάται από τα προϊόντα που πουλάει), ή ενός δικηγόρου (ανάλογα με τις υποθέσεις που κερδίζει ή χάνει αντίστοιχα), η ακόμη ενός εργάτη εργοστασίου (ανάλογα με τα προϊόντα που έχει παράγει). Μέχρι πρότινος αρκετοί αξιολογούσαν τους προγραμματιστές όπως έναν εργάτη εργοστασίου, δηλαδή ανάλογα με τις γραμμές κώδικα που έχει συγγράψει. Όπως γίνεται αντιληπτό αυτό δεν ήταν αξιολόγηση, για αυτό το λόγο δημιουργήθηκε το SonarQube[17]. Το SonarQube είναι μια πλατφόρμα ανοιχτού λογισμικού για τον συνεχή έλεγχο της ποιότητας του κώδικα. Η πλατφόρμα αυτή είναι πολύ σημαντική για την ποιότητα του κώδικα του οποίου έχεις γράψει. Φυσικά είναι μια πολύ καλή μέθοδος για να απαντήσεις στον εαυτό σου εάν έχεις γράψει σωστά τον κώδικα σου. Το SonarQube παλαιότερα ονομαζόταν ως Sonar, για αυτό το λόγο θα δείτε γραμμένα και τα δυο ονόματα που ουσιαστικά είναι το ίδιο.

Το SonarQube συλλέγει και αναλύει πληροφορίες για τον πηγαίο κώδικα, μετρώντας την ποιότητα του αλλά και εξάγοντας πληροφορίες για αυτόν[18]. Συνδυάζει εργαλεία στατικής, αλλά και δυναμικής ανάλυσης και επιτρέπει τη συνεχή μέτρηση της ποιότητας με τη πάροδο του χρόνου. Όλα τα στοιχεία τα οποία επηρεάζουν αρνητικά τον κώδικα μας, από μικρές στιλιστικές λεπτομέρειες, έως κρίσιμα σφάλματα στη σχεδίαση, ελέγχονται και αξιολογούνται από το SonarQube. Κατά συνέπεια, οι προγραμματιστές μπορούν να παρακολουθούν την ανάλυση του κώδικα και να έχουν πρόσβαση σε σφάλματα τα οποία καταγράφηκαν από την πλατφόρμα. Η πλατφόρμα Sonar αναλύει τον πηγαίο κώδικα από διαφορετικές απόψεις και ελέγχει εξονυχιστικά κάθε επίπεδο του κώδικα (επίπεδο project, επίπεδο πακέτου, επίπεδο κλάσης). Σε κάθε επίπεδο παράγονται μετρικές αλλά και στατιστικά, αποκαλύπτοντας προβληματικές περιοχές στον κώδικα που χρειάζονται βελτίωση.

Το SonarQube παρέχει μια στιγμιαία εικόνα για τον κώδικα σου, εντοπίζει κομμάτια στα οποία ο κώδικας σου υστερεί αλλά και κομμάτια τα οποία θα προσφέρουν προβλήματα στο μέλλον. Ένας δείκτης ο οποίος εντοπίζει μελλοντικά προβλήματα είναι η κάλυψη

(coverage). Η μετρική αυτή χρησιμοποιείται για να μετρήσουμε το ποσοστό των επιτυχημένων test script που εφαρμόστηκαν στον κώδικα. Έστω ότι μετριέται από το Sonar ο δείκτης coverage σε ένα ποσοστό 50%, το οποίο με μια πρώτη ματιά δεν μας φαίνεται εξαιρετικά ικανοποιητικό. Ωστόσο αυτό που πρέπει να ελέγξουμε είναι τη τιμή της μετρικής στην προηγούμενη έκδοση. Εάν η τιμή της ήταν 35%, τότε έγινε καλή δουλειά και η ποιότητα του κώδικα έχει αναβαθμιστεί, αντίθετα εάν το νούμερο ήταν 70% τότε έχεις αλλοιώσει την ποιότητα του κώδικα και πρέπει να καταβάλλεις προσπάθεια για να την αναβαθμίσεις.

Η συγκεκριμένη πλατφόρμα δεν υποδεικνύει μόνο το τι πάει λάθος στον κώδικα. Προσφέρει επίσης εργαλεία για τη διαχείριση της ποιότητας τα οποία σε υποδεικνύουν πώς να διορθώσεις τα σφάλματα. Τα εργαλεία αυτά είναι :

- IDE integration
- Integration for Jenskins
- Continuous Integration server
- Code-review tools

Αξίζει να αναφερθεί στο συγκεκριμένο σημείο, ότι το SonarQube δεν είναι το μόνο λογισμικό το οποίο εξάγει πληροφορίες για την ποιότητα του κώδικα. Ωστόσο, ενώ τα άλλα λογισμικά έχουν εστιάσει στην εύρεση σφαλμάτων, το Sonar έχει εστιάσει σε αυτό που έχουν ονομάσει οι δημιουργοί του “Seven Axes of Quality”, δηλαδή οι επτά άξονες της ποιότητας. Οι άξονες αυτοί αντιπροσωπεύουν την ανίχνευση σφαλμάτων, κανόνες συγγραφής, ποσοστό κάλυψης test script, API documentation, περιπλοκότητα (complexity), και αρχιτεκτονική. Κάποια από τα λογισμικά που κάνουν παρόμοια χρήση με το Sonar εκτός από την ανίχνευση σφαλμάτων, εστιάζουν και στο ποσοστό κάλυψης, αλλά και στο API documentation. Βέβαια, το sonar είναι το μόνο το οποίο εστιάζει και στους επτά άξονες για αυτό το λόγο χρησιμοποιήθηκε και για την παρούσα εργασία.

Το SonarQube μπορεί να βοηθήσει όχι μόνο τους προγραμματιστές, αλλά και τους δοκιμαστές, διαχειριστές έργων και επιχειρηματίες. Όσον αφορά τους δοκιμαστές η συγκεκριμένη πλατφόρμα μπορεί να υποδείξει τα σημεία στα οποία τα test που έγιναν ήταν πολύ «χαλαρά» ή δεν υπήρχαν καθόλου. Φυσικά όσον αφορά τους προγραμματιστές βοηθάει για να ξέρουν τη έχουν κάνει λάθος στον κώδικα τους. Όσον αφορά τους επιχειρηματίες και τους διαχειριστές έργων τους βοηθάει η εξαγωγή στατιστικών στοιχείων και μετρικών, καθώς πλέον μπορούν να βλέπουν σχεδιαγράμματα για τον κώδικα και να αποφασίζουν για τυχών αλλαγές ή επενδύσεις.

Το SonarQube είναι γραμμένο στη Java και ξεκίνησε να αναλύει προγράμματα τα οποία ήταν γραμμένα σε Java, ωστόσο λόγω της μεγάλης του ανάπτυξης τα τελευταία χρόνια μπορεί να αναλύσει πάρα πολλές γλώσσες προγραμματισμού[18] οι οποίες είναι:

- C
- C++
- Javascript
- C#
- Java
- COBOL
- PL/SQL
- PL/I

- PHP
- ABAP
- VB.NET
- V86
- Python
- RPG
- Flex
- Objective-C
- Swift
- Web
- XML

Όπως αναφέρθηκε πριν ένα πολύ σημαντικό χαρακτηριστικό της πλατφόρμας αυτής είναι η εξαγωγή μετρικών και γραφημάτων σχετικά με τον κώδικα. Οι μετρικές που παράγονται είναι πάνω από 140 και αποθηκεύονται σε κατάλληλη βάση δεδομένων την οποία έχουμε διαμορφώσει κατάλληλα. Οι πιο σημαντικές μετρικές που παράγονται από το SonarQube και χρειάζονται για τη διαδικασία υπολογισμού του breaking point είναι:

Number of Function: Αναφέρεται στον αριθμό των συναρτήσεων που υπάρχουν στον κώδικα

Number of Statements: Αναφέρεται στον αριθμό των εντολών που υπάρχουν στον κώδικα.

Complexity: Η πολυπλοκότητα που υπολογίζεται βάσει του αριθμού των διαδρομών μέσω του κώδικα. Κάθε φορά που η ροή ελέγχου μιας συνάρτησης διασπάται, ο μετρητής πολυπλοκότητας αυξάνεται κατά ένα. Η ελάχιστη πολυπλοκότητα ισούται με 1. Αυτός ο υπολογισμός ποικίλλει ελαφρώς ανάλογα με τη γλώσσα προγραμματισμού, επειδή υπάρχουν διαφορετικές λέξεις-κλειδιά και λειτουργίες.

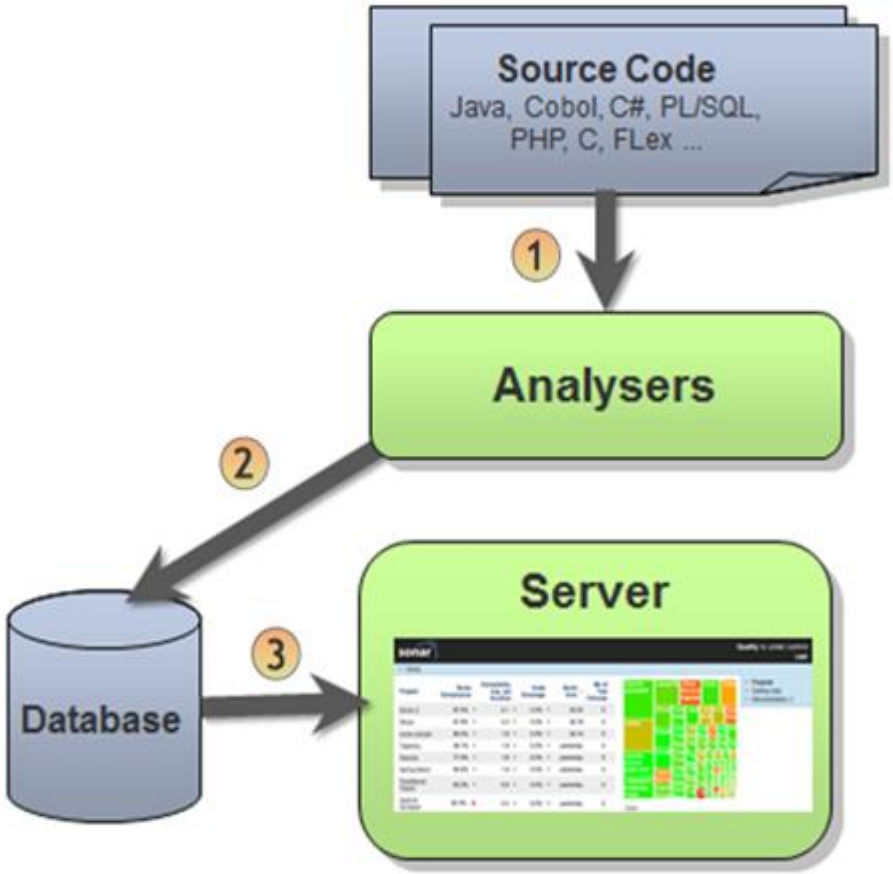
Sqale Index: Πρόκειται για μια μετρική η οποία συνδέεται άμεσα με τον υπολογισμό του bp. Με αυτή τη μετρική υπολογίζουμε την προσπάθεια που πρέπει να καταβληθεί ώστε να διορθωθούν όλα τα προβλήματα στην ποιότητα λογισμικού.

LCOM: Αναφέρεται στον αριθμό των γραμμών χωρίς τα σχόλια που υπάρχουν στον κώδικα.

Number of Classes: Αναφέρεται στον αριθμό των κλάσεων (συμπεριλαμβανομένων των nested κλάσεων, interfaces, enums και annotations) που υπάρχουν στον κώδικα.

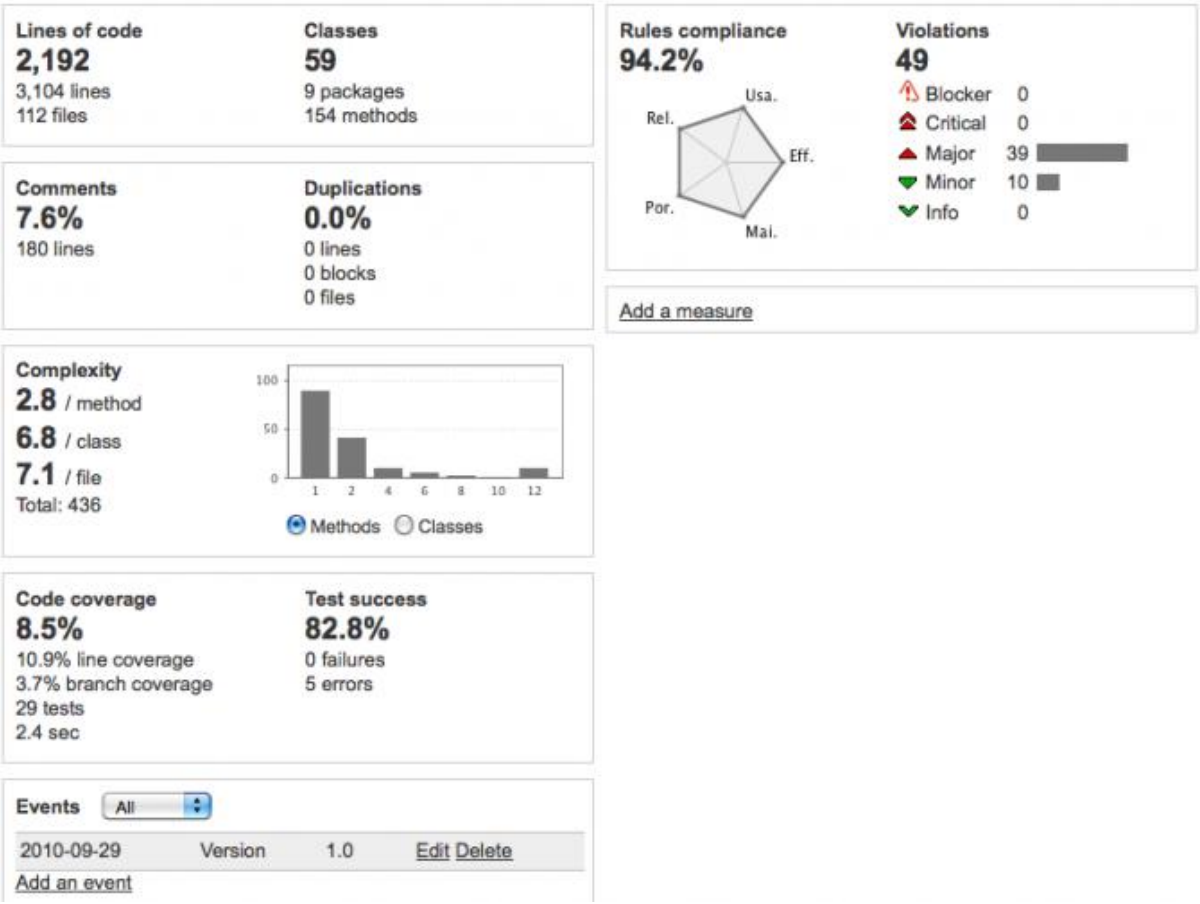
Φυσικά να τονίσουμε ότι επειδή το πρόγραμμα αναλύεται σε τρία επίπεδα (project, πακέτου, κλάσεων), οι μετρικές παρέχουν πληροφορίες για κάθε ένα στάδιο ξεχωριστά.

Στη παρακάτω εικόνα μπορούμε να δούμε τη δομή του SonarQube. Ουσιαστικά στο πρώτο επίπεδο βρίσκεται ο πηγαίος κώδικας ο οποίος θέλουμε να αναλύσουμε, με τη βοήθεια ενός κατάλληλου λογισμικού (sonar-scanner ή sonar-runner). Στη συνέχεια τα αποτελέσματα της ανάλυσης αποθηκεύονται στη βάση δεδομένων, και μπορούμε να τα δούμε γραφικά στον server με τη βοήθεια του browser.



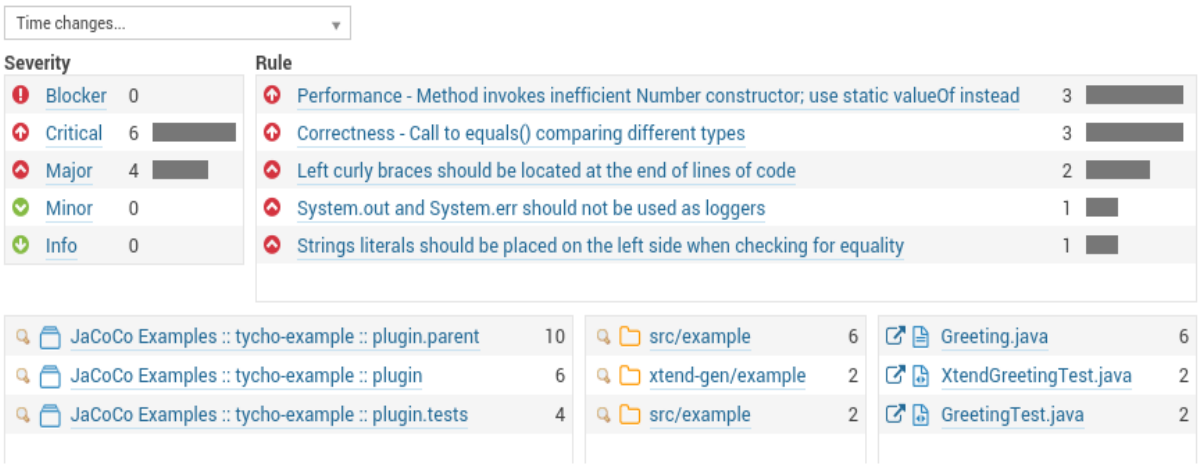
Εικόνα 1: Αρχιτεκτονική της πλατφόρμας Sonar

Όπως βλέπεται στην Εικόνα 1 στο τελευταίο στάδιο μπορούμε να δούμε τις πληροφορίες για τις μετρικές στον browser πληκτρολογώντας http://localhost:port_number. Συνήθως χρησιμοποιείται από προεπιλογή η 9000 αλλά μπορεί κάλιστα να χρησιμοποιηθεί οποιαδήποτε άλλη. Στην Εικόνα 2 μπορούμε να δούμε αποτελέσματα από την ανάλυση ενός project. Βλέπουμε ξεκάθαρα τις τιμές κάποιων μετρικών, π.χ τον αριθμό των γραμμών, τον αριθμό των αρχείων, τον αριθμό των κλάσσεων, το ποσοστό των σχόλιων την περιπλοκότητα, την κάλυψη και πολλές άλλες μετρικές.



Εικόνα 2: Εμφάνιση μετρικών

Όπως έχουμε αναφέρει παραπάνω η συγκεκριμένη πλατφόρμα διαθέτει λειτουργία ανίχνευσης σφαλμάτων. Στην Εικόνα 3 μπορούμε να διακρίνουμε τη λειτουργία αυτή. Αναλυτικότερα διακρίνουμε να εμφανίζονται όλα τα θέματα που έχουν προκύψει, καθώς και από πού προέκυψαν. Για παράδειγμα ανιχνεύθηκαν 3 θέματα τα οποία ανήκουν στην κατηγορία Performance και προκλήθηκαν από “Method invokes inefficient Number constructor; use static valueOF instead”, σύμφωνα με τον πίνακα Rule της Εικόνας 3. Τα θέματα αυτά χαρακτηρίζονται ως ‘Critical’ σύμφωνα με τον πίνακα Serevity της Εικόνας 3.



Εικόνα 3: Ανίχνευση σφαλμάτων Sonar

Επιπρόσθετα μπορούμε να διακρίνουμε στην Εικόνα 3 σε ποιο συγκεκριμένο πακέτο ή κλάση ανιχνεύθηκαν οι παραβιάσεις αυτές.

Συνοψίζοντας το πιο σημαντικό στοιχείο του Sonar δεν είναι ότι παρέχει μετρικές και στατιστικά στοιχεία για τον κώδικα που έχει αναλύσει, αλλά ότι μετατρέπει τις μετρικές αυτές σε πραγματικές επιχειρηματικές αξίες όπως είναι το τεχνικό χρέος και ο κίνδυνος (risk) [19].

2.2.2 MySQL

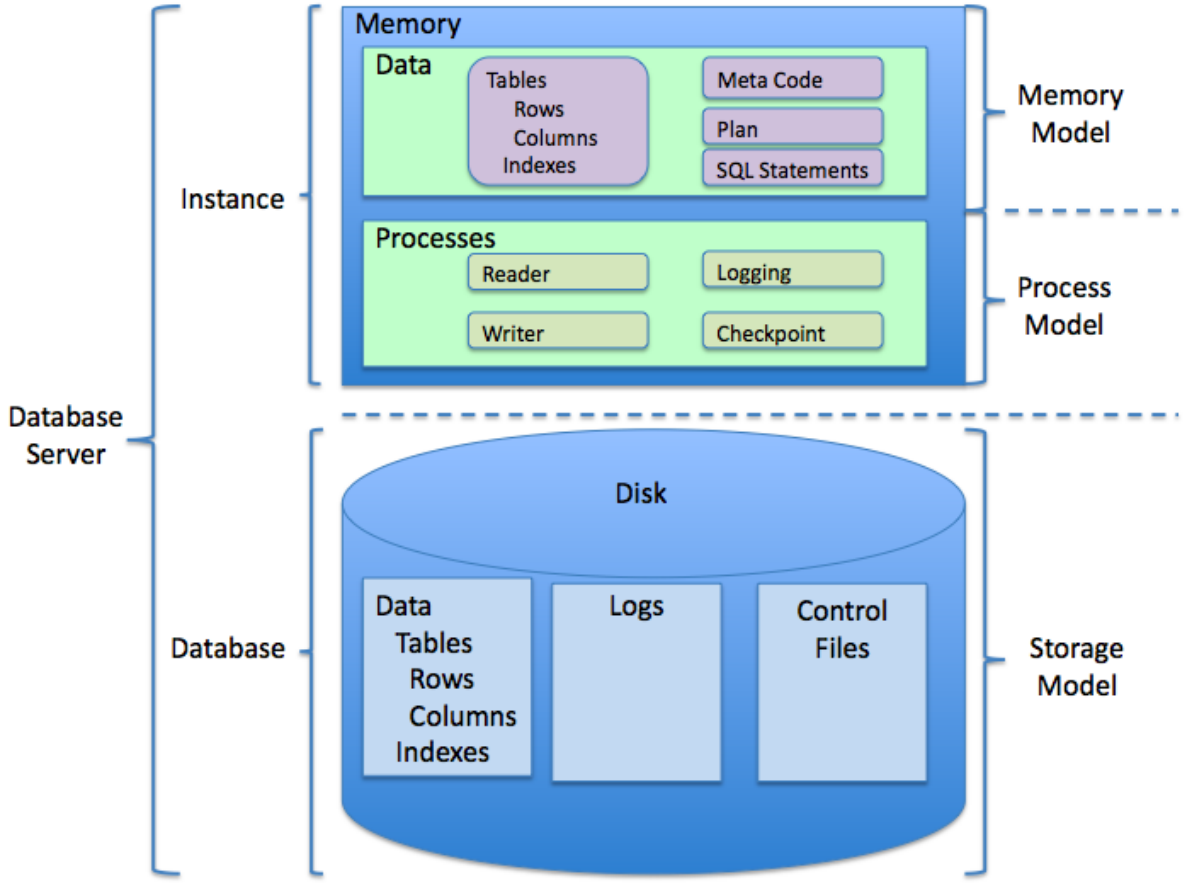
Η εποχή μας είναι μάρτυρας μιας έκρηξης μεγάλου όγκου πληροφορίας που είναι διαθέσιμη στο μέσο άνθρωπο, και έχει τεράστια σημασία για τη μέση επιχείρηση. Λόγω του μεγάλου όγκου πληροφοριών, δημιουργήθηκαν οι βάσεις δεδομένων. Βάση δεδομένων είναι οποιαδήποτε συλλογή δεδομένων που σχετίζεται με τις δραστηριότητες ενός (ή περισσότερων, αλληλοσυσχετιζομένων) οργανισμού(-ων). Πλέον με τη δημιουργία των βάσεων δεδομένων οι χρήστες μπορούν να αποθηκεύουν τα δεδομένα σε σύνθετα, μεγάλα δομικά συγκροτήματα [20]. Μέσα στις βάσεις δεδομένων εκτός από τις οντότητες ενός οργανισμού, μπορούν να αποθηκευτούν και οι συσχετίσεις τους. Στη συνέχεια δημιουργήθηκε η ανάγκη για τη διαχείριση των βάσεων δεδομένων, για αυτό και δημιουργήθηκαν κατάλληλα λογισμικά τα οποία θα μπορούσαν να διαχειρίζονται τις βάσεις αυτές. Τα προτερήματα των λογισμικών αυτών είναι:

- Ανεξαρτησία των δεδομένων
- Ταχεία πρόσβαση στα δεδομένα
- Ακεραιότητα των δεδομένων και ασφάλεια
- Διαχείριση των δεδομένων
- Ταυτόχρονη πρόσβαση και επαναφορά από βλάβη
- Μειωμένος χρόνος ανάπτυξης των εφαρμογών

Τα λογισμικά τα οποία είναι ειδικά σχεδιασμένα για τη διαχείριση και τη συντήρηση του μεγάλου όγκου πληροφορίας ονομάζονται Συστήματα Διαχείρισης Βάσεων Δεδομένων (**D**ata**B**ase **M**anagement **S**ystems) ή Συστήματα Σχεσιακής Διαχείριση Βάσεων Δεδομένων (**R**elational **D**ata**B**ase **M**anagement **S**ystems). Η διαφορά των δύο λογισμικών διαχείρισης είναι ότι το RDBMS βασίζεται στο σχεσιακό μοντέλο και η οργάνωση των δεδομένων ακολουθείται από αυτό. Στην Εικόνα 4 διακρίνεται η γενική δομή ενός συστήματος διαχείρισης σχεσιακών βάσεων δεδομένων.

Η MySQL λοιπόν είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων. Το όνομα της είναι ένας συνδυασμός από το όνομα του ιδρυτή της, (Michael Widenius) και της συντομογραφίας Δομημένης γλώσσας ερωτημάτων (SQL). Η πρώτη έκδοση της κυκλοφόρησε στις 23 Μαΐου του 1995. Η εγκατάσταση και η εκτέλεση της γίνεται από έναν εξυπηρετητή (server) και δίνει τη δυνατότητα ταυτόχρονης «εξυπηρέτησης» διάφορων χρηστών σε διαφορετικές βάσεις δεδομένων.

.



Εικόνα 4: Γενική δομή μιας σχεσιακής βάσης δεδομένων.

Τα κύρια χαρακτηριστικά της MySQL είναι:

- Σύστημα ανοιχτού λογισμικού, οπότε η χρήση του είναι ελεύθερη σε όλους.
- Ισχυρό πρόγραμμα που μπορεί να διαχειρίζεται τεράστιους όγκους δεδομένων χωρίς να επηρεάζεται αρνητικά η λειτουργικότητά του.
- Χρησιμοποιεί μια μορφή σύνταξης παρόμοια με την SQL.
- Μπορεί να χρησιμοποιηθεί σε πάρα πολλά λειτουργικά συστήματα (AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, macOS, Microsoft Windows κ.α.).
- Μπορεί να συνδυαστεί με πολλές γλώσσες προγραμματισμού (PHP, Perl, Java, C, C++ κ.α.).
- Υποστηρίζει πάρα πολύ μεγάλους όγκους δεδομένων (μέχρι και 50 εκατομμύρια γραμμές σε έναν πίνακα). Το προεπιλεγμένο μέγεθος για έναν πίνακα είναι 4GB, ωστόσο εάν το λογισμικό έχει δυνατότητες μπορεί να διαχειριστεί έως 8 εκατομμύρια terabytes (TB).
- Επειδή είναι ένα πρόγραμμα ανοιχτού λογισμικού, μπορεί όποιος επιθυμεί να επέμβει στο λογισμικό και να το προσαρμόσει στις δικές του ανάγκες.

Οι βασικότερες εντολές [20] που χρησιμοποιούνται είναι:

1. **SELECT**: Εξάγει δεδομένα από μια βάση δεδομένων.
2. **UPDATE**: Ενημερώνει δεδομένα πινάκων.
3. **DELETE**: Διαγράφει δεδομένα.
4. **INSERT INTO**: Εισάγει δεδομένα σε έναν πίνακα μιας βάσης δεδομένων.
5. **CREATE DATABASE**: Δημιουργεί μια βάση δεδομένων.
6. **ALTER DATABASE**: Τροποποιεί μια βάση δεδομένων.
7. **CREATE TABLE**: Δημιουργεί έναν νέο πίνακα.
8. **ALTER TABLE**: Τροποποιεί έναν πίνακα.
9. **DROP TABLE**: Διαγράφει έναν πίνακα.
10. **CREATE INDEX**: Δημιουργεί έναν δείκτη (κλειδί αναζήτησης).
11. **DROP INDEX**: Διαγράφει έναν δείκτη.
12. **JOIN**: Συνδυάζει σειρές από δύο ή περισσότερους πίνακες.

2.2.3 Βιβλιοθήκες JDBC και DBI

Όπως παρουσιάστηκε στο προηγούμενο υποκεφάλαιο, η MySQL μπορεί να συνδυαστεί με πάρα πολλές γλώσσες. Το εργαλείο το οποίο κατασκευάστηκε διαθέτει κομμάτια προγραμματισμού σε Perl και σε Java. Επομένως έπρεπε να βρούμε μια βιβλιοθήκη η οποία μπορεί να συνδυάσει τη Java και τη Perl με τη MySQL. Η πιο γνωστή βιβλιοθήκη για να «ενώσει» τη Java με τη MySQL είναι η JDBC, ενώ η PERL συνδυάζεται με τη MySQL με τη βοήθεια της βιβλιοθήκης DBI. Για να γίνει η σύνδεση απαιτείται το URL για τη βάση δεδομένων, η οποία πρόκειται να χρησιμοποιηθεί και στη συνέχεια ο χρήστης και ο κωδικός πρόσβασης του. Στην Εικόνα 5 απεικονίζονται όλες οι εντολές οι οποίες χρειάζονται για τη σύνδεση στη βάση δεδομένων από τη Java, ενώ στην Εικόνα 6 απεικονίζεται η σύνδεση της βάσης δεδομένων με τη Perl.

```
//URL For Database
String URL = "jdbc:mysql://localhost:3306/databaseName?useSSL=false";
//User of database
String USER = "Sarikiriakidis";
//Password for user Sarikiriakidis
String PASS = "Christos623";
//Connection on db
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

Εικόνα 5: Σύνδεση στη βάση δεδομένων με τη βιβλιοθήκη JDBC

Η βιβλιοθήκη jdbc πρέπει να τοποθετηθεί στο φάκελο με τα jar αρχεία, ενώ η dbi πρέπει να οριστεί στην αρχή του προγράμματος χρησιμοποιώντας την εντολή :

- use DBI;

Όνομα Βάσης Δεδοδομένων

```
my $dbh = DBI->connect('DBI:mysql:database=databaseName;host=localhost',
                       'Sarikiriakidis',
                       'Christos623!',{ RaiseError => 1, AutoCommit => 1 },);
```

Κωδικός Πρόσβασης

Όνομα Χρήστη

Εικόνα 6: Σύνδεση στη βάση δεδομένων με τη βιβλιοθήκη DBI

2.2.4 Git && Github

Για τη κατανόηση του εργαλείου Github είναι απαραίτητη η κατανόηση του όρου git. Το git είναι ένα σύστημα ελέγχου εκδόσεων (λέγεται και σύστημα ελέγχου αναθεωρήσεων ή σύστημα ελέγχου πηγαίου κώδικα) ή στα αγγλικά Version Control System, με έμφαση στην ταχύτητα, στην ακεραιότητα των δεδομένων και στην υποστήριξη για κατανεμημένες μη γραμμικές ροές εργασίας[21]. Το git σχεδιάστηκε και αναπτύχθηκε αρχικά από τον Λίνους Τόρβαλντς για την ανάπτυξη του πυρήνα Linux το 2005 και έχει γίνει από τότε το πιο διαδεδομένο σύστημα ελέγχου εκδόσεων για ανάπτυξη λογισμικού.

Όπως τα περισσότερα άλλα κατανεμημένα συστήματα ελέγχου εκδόσεων/αναθεωρήσεων και αντίθετα με τα περισσότερα συστήματα πελάτη-διακομιστή, κάθε κατάλογος εργασίας του Git είναι ένα ολοκληρωμένο αποθετήριο λογισμικού με πλήρες ιστορικό και δυνατότητες πλήρους παρακολούθησης της έκδοσης, ανεξάρτητα από την πρόσβαση δικτύου ενός ή ενός κεντρικού διακομιστή. Όπως ο πυρήνας Linux, το Git είναι και αυτό ελεύθερο λογισμικό. Το Git παρακολουθεί συχνά τις αλλαγές σε ένα σύνολο αρχείων κρατώντας στιγμιότυπα τους. Το σύνολο των στιγμιότυπων ονομάζεται αποθετήριο ή repository. Ο στόχος ουσιαστικά του git είναι να υποστηρίξει τη διανεμημένη παραγωγικότητα και την ταχύτητα. Το Git βοηθά τον χρήστη να διατηρήσει την υπευθυνότητα και την πατρότητα για κάθε δράστη του, να διατηρεί την ακεραιότητα του έργου και να βελτιώνει τη συνολική επικοινωνία μεταξύ των συνεργατών. Η σχεδιαστική φιλοσοφία αντικατοπτρίζεται στην κοινή φιλοσοφία χρήσης:

- Commit νωρίς και συχνά
- Κάθε Commit αντιπροσωπεύει μια ιδέα ή μια ολόκληρη αλλαγή (και έτσι είναι ευκολότερη η ανάγνωση και η επαναφορά σε προηγούμενη έκδοση)
- Κάθε branch αντιπροσωπεύει ένα χαρακτηριστικό (ή θέμα) (και έτσι είναι ευκολότερη η ανάγνωση και η συγχώνευση) για τον τοπικό κατάλογο εργασίας και ο δείκτης των τοπικών συμφωνιών επαναφοράς είναι το μηδέν.

Επομένως οι χρήσεις του git είναι:

- Συμμετοχή σε κοινά project, και έργα ανοιχτού λογισμικού
- Επιστροφή σε παλαιότερη έκδοση αν γίνει κάποιο λάθος
- Δοκιμή παραλλαγών χωρίς να επηρεάζεται η τρέχουσα έκδοση

Υπάρχουν πάρα πολλές εντολές, όμως οι βασικότερες εντολές του Git είναι:

- git clone git_repository: Χρησιμοποιείται για την αντιγραφή ενός project στο τοπικό δίσκο.
- git add filename: Προσθέτει το περιεχόμενο του αρχείου και το ετοιμάζει για να γίνει το επόμενο commit.
- git commit -m “Message”: Αποθηκεύει ένα μήνυμα, το οποίο εξηγεί τι έχει αλλάξει στο συγκεκριμένο commit.
- git push: Με αυτή την εντολή ανεβαίνει οριστικά στο Github το αρχείο το οποίο έχει γίνει commit και add.
- git-diff: Εμφανίζει διαφορές στα commit που έχουν γίνει και στο Gihtub.
- git log: Εμφανίζει πληροφορίες για τα commit που έχουν γίνει.

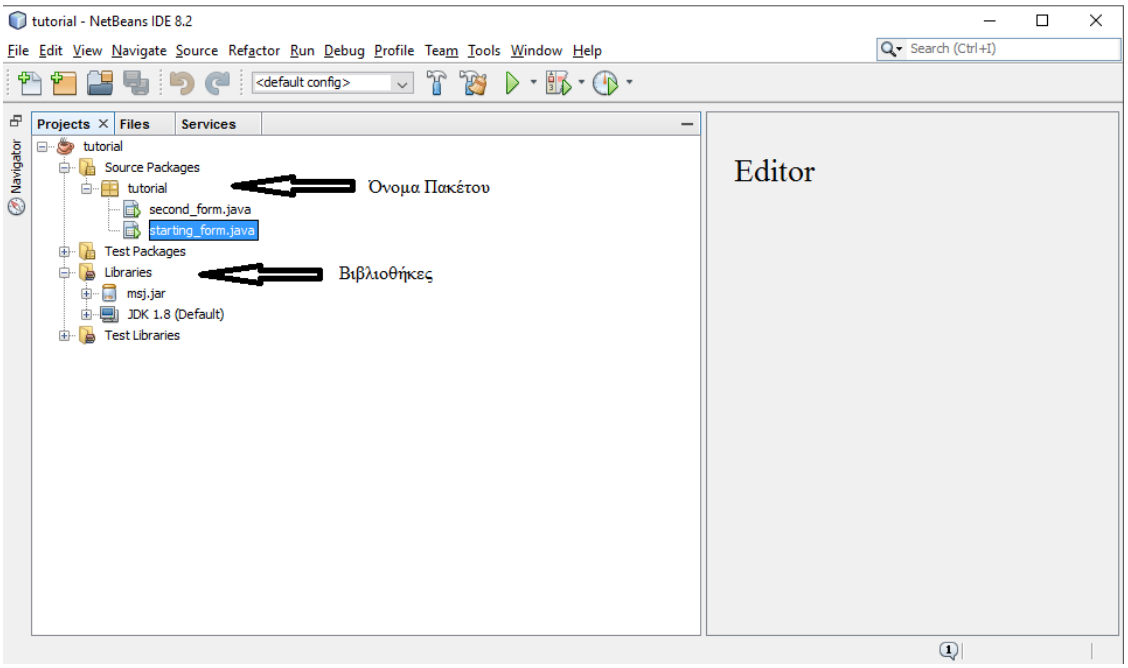
Το Github [22] λοιπόν πρόκειται για έναν εξυπηρετητή ο οποίος φιλοξενεί αποθετήρια (repositories) και παρέχει επίσης μια εύχρηστη διεπαφή ιστού. Είναι εξαιρετικά δημοφιλές καθώς το χρησιμοποιούν πάνω από εννιά εκατομμύρια χρήστες. Αυτό που κάνει το Github πετυχημένο ήταν ότι στήθηκε έτσι ώστε να ευνοεί τη συνεργασία μεταξύ προγραμματιστών (το μεγαλύτερο ποσοστό των χρηστών ανήκει στο κλάδο του προγραμματισμού). Τα έργα τα οποία είναι ανεβασμένα στο Github δεν περιορίζονται σε κάποια συγκεκριμένη κατηγορία, καθώς υπάρχουν project τα οποία αφορούν σχέδια βελτίωσης σπιτιού ή ακόμη και έργα που αφορούν τη μουσική. Υπάρχει μία μεγάλη ποικιλία έργων, από πολλούς και διαφορετικούς προγραμματιστές (ή ομάδες προγραμματιστών) με πάρα πολλούς σκοπούς. Θα αξιοποιηθεί, λοιπόν, το Github στη παρούσα εργασία, προκειμένου να υπάρχει μεγάλο δείγμα στην έρευνα και τα αποτελέσματα να είναι πιο κατατοπιστικά για την εύρεση έργων. Σκοπός στη παρούσα εργασία είναι η λήψη έργων από τον συγκεκριμένο εξυπηρετητή, επομένως οι λειτουργίες git που θα χρησιμοποιηθούν είναι:

- git clone
- git log

2.3 Πλατφόρμες Ανάπτυξης Λογισμικού

2.3.1 Netbeans

Το Netbeans [23] είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης προγραμμάτων και παρέχει στους προγραμματιστές όλα τα εργαλεία που χρειάζονται για μια πιο άμεση, γρήγορη και αξιόπιστη αποσφαλμάτωση. Το περιβάλλον εργασίας του λογισμικού φαίνεται στην Εικόνα 7. Το Netbeans έχει σχεδιαστεί κυρίως για Java, ωστόσο μπορεί να λειτουργήσει άψογα με πολλές ακόμα γλώσσες προγραμματισμού, όπως Html5, Javascript, PHP, C/C++, Ruby και άλλες πολλές γλώσσες προγραμματισμού. Το συγκεκριμένο εργαλείο χρησιμοποιήθηκε για τη κατασκευή του εργαλείου εύρεσης του bp.



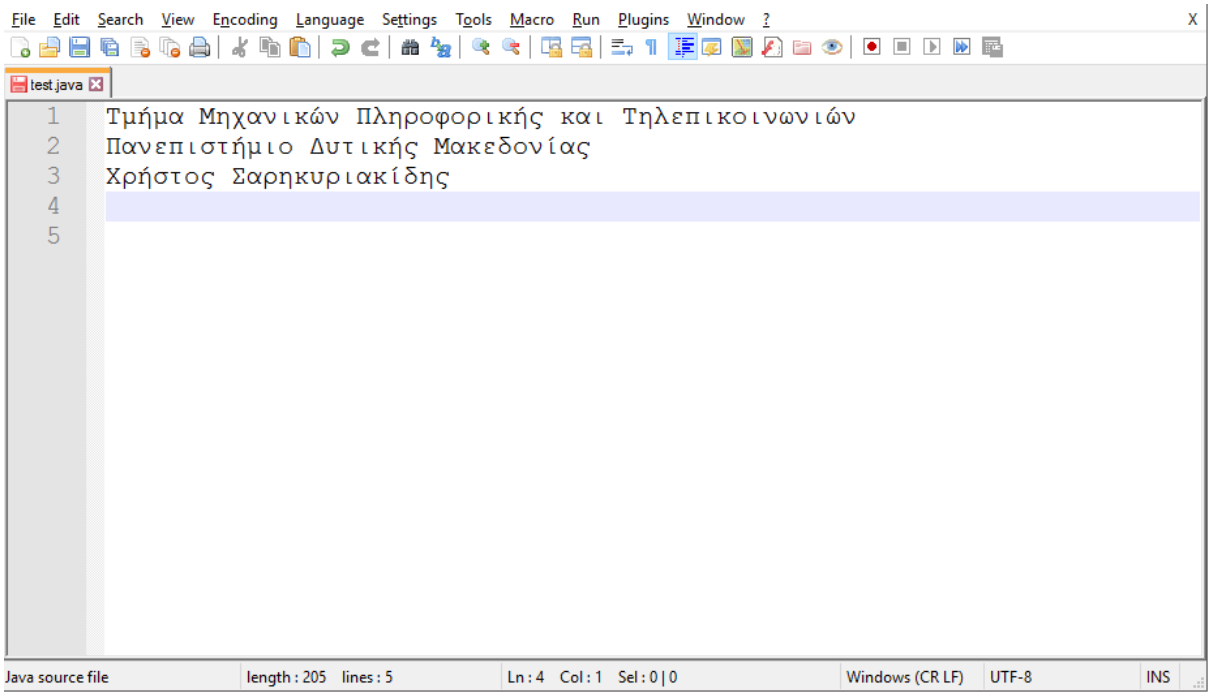
Εικόνα 7: Περιβάλλον λογισμικού Netbeans

Οι λόγοι που προτιμάται το συγκεκριμένο εργαλείο είναι αρκετοί. Αρχικά, το περιβάλλον εργασίας του είναι απλό και εύχρηστο, ακόμη και για νέους προγραμματιστές χωρίς ιδιαίτερη εμπειρία. Επίσης, με το Netbeans ο χρήστης έχει τη δυνατότητα να εκτελέσει τη μεταγλώττιση ενός προγράμματος, συνδέοντας απλά τις βιβλιοθήκες στο λογισμικό, χωρίς να απαιτούνται μακροσκελείς εντολές όπως σε ένα περιβάλλον τερματικού. Επιπρόσθετα, το Netbeans παρέχει μια γρήγορη και αξιόπιστη δυνατότητα για το σχεδιασμό ενός Graphical User Interface (GUI). Μπορεί το εργαλείο που κατασκευάστηκε να είναι αξιόπιστο και πολύ χρήσιμο, αλλά πρέπει ταυτόχρονα να είναι εύχρηστο και φιλικό προς το χρήστη. Για αυτό το λόγο, στη παρούσα εργασία, δόθηκε ιδιαίτερη σημασία στο σχεδιασμό του εργαλείου και στο σκοπό αυτό βοήθησε αρκετά το λογισμικό Netbeans με τις ποικίλες δυνατότητες που παρέχει.

Όσον αφορά την εγκατάσταση του, είναι αρκετά εύκολη και παρέχει εκδόσεις που υποστηρίζονται από όλα τα λειτουργικά συστήματα (Windows, Linux, Mac) με αποτέλεσμα ο χρήστης να μπορεί να το επεξεργαστεί εύκολα από πολλά λειτουργικά συστήματα.

2.3.2 Notepad++

Το Notepad++ [24] αποτελεί έναν από τους πιο δημοφιλείς επεξεργαστές κειμένου ανοιχτού κώδικα, με πάνω από είκοσι οχτώ εκατομμύρια χρήστες. Υποστηρίζει πάρα πολλές γλώσσες προγραμματισμού και παρέχει δυνατότητες που διευκολύνουν το προγραμματιστή, όπως η επισήμανση λαθών, επεκτάσεων κ.α. Στη παρούσα διπλωματική εργασία το Notepad++ συνέβαλε στη δημιουργία script σε Perl και MySQL. Το περιβάλλον της εφαρμογής απεικονίζεται στην Εικόνα 8.



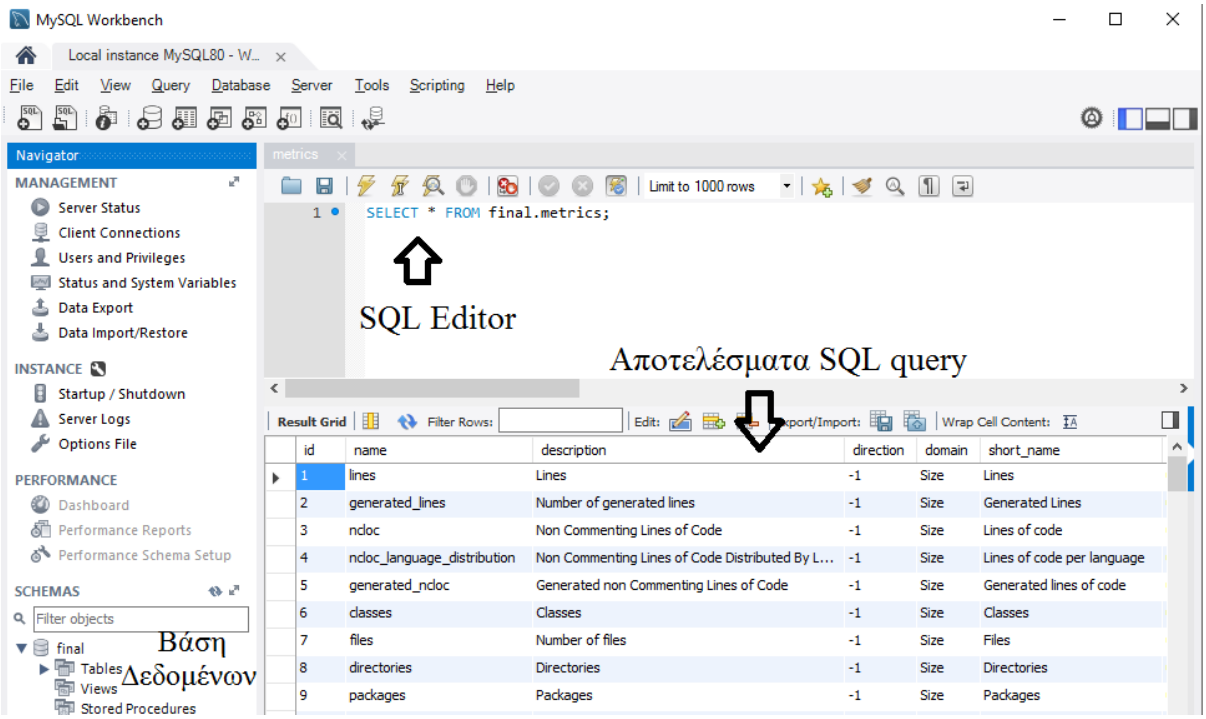
Εικόνα 8: Γραφικό περιβάλλον του λογισμικού Notepad++

2.3.3 MySQL Workbench

Το MySQL Workbench [25] αποτελεί ένα εργαλείο οπτικού σχεδιασμού μιας βάσης δεδομένων που ενσωματώνει σε ένα ενιαίο περιβάλλον τις παρακάτω λειτουργίες:

- Ανάπτυξη της SQL
- Διαχείριση μιας βάσης δεδομένων
- Σχεδιασμό μιας βάσης δεδομένων
- Δημιουργία και συντήρηση μιας βάσης δεδομένων

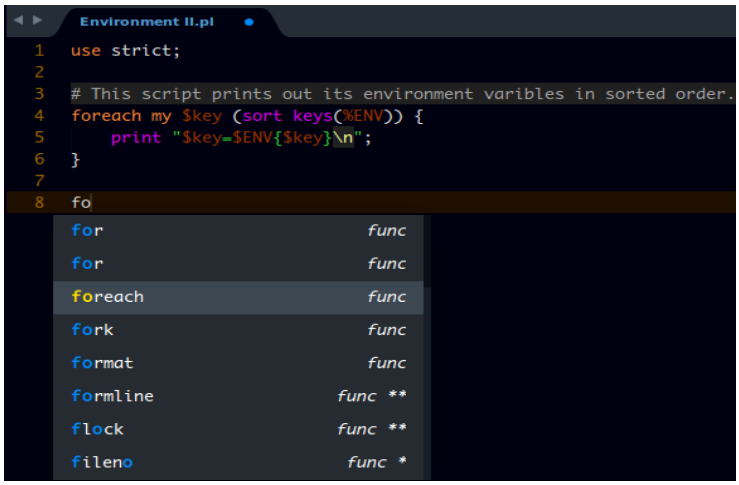
Στην Εικόνα 9 διακρίνεται το περιβάλλον του λογισμικού MySQL Workbench. Στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε το MySQL Workbench για το σχεδιασμό της βάσης δεδομένων που χρησιμοποιούμε, αλλά και για τη διαχείριση της.



Εικόνα 9: Περιβάλλον εργασίας MySQL Workbench.

2.3.4 Sublime Text editor

Η πλατφόρμα Sublime Text editor [26], είναι ένας ακόμη επεξεργαστής κειμένου ανοιχτού κώδικα. Το εργαλείο αυτό παρέχει όλα τα πλεονεκτήματα του Notepad++ που προαναφέρθηκε σε προηγούμενη υποενότητα. Χρησιμοποιήθηκε στη παρούσα διπλωματική διότι το Notepad++, δεν παρέχει έκδοση για τα λειτουργικά συστήματα Unix (μπορεί να χρησιμοποιηθεί στα λειτουργικά συστήματα Unix, με τη βοήθεια του εργαλείου wine). Το περιβάλλον του εργαλείου απεικονίζεται στην Εικόνα 10.



Εικόνα 10: Περιβάλλον εργασίας του Sublime Text Editor

2.4 Ανάλυση μεθοδολογίας

Ο υπολογισμός του σημείου ισορροπίας εφαρμόζεται σύμφωνα με τη διαίρεση του κεφαλαίου (principal) προς τον τόκο (interest) του συγκεκριμένου έργου [1]. Το κεφάλαιο είναι μια καθορισμένη και αποδεκτή έννοια από όλους. Από την άλλη, η έννοια interest είναι μια έννοια η οποία έχει αναλυθεί από πολλούς ερευνητές και έχει οδηγήσει σε αμφιλεγόμενα αποτελέσματα. Η υλοποίηση της εργασίας εστίασε στη συνάρτηση αποτελεσματικότητας (FITTED), για τον υπολογισμό του interest, όπως παρουσιάζεται από τον Α. Χατζηγεωργίου στο 'Estimating the breaking point of TD' από Ampatzoglou.

2.4.1 Υπολογισμός του κεφαλαίου

Υποθέτοντας ότι κάθε αντικειμενοστραφής λογισμικό, έχει μια πραγματική ποιότητα στο σχεδιασμό του λογισμικού του, μπορεί να εκτιμηθεί η ποιότητα αυτή με μια συνάρτηση αποτελεσματικότητας. Στη παρούσα εργασία η συνάρτηση αποτελεσματικότητας η οποία χρησιμοποιείται είναι η μέτρηση τόσο της σύζευξης όσο και της συνοχής. Η συγκεκριμένη συνάρτηση εκτιμά πόσο καλά οι οντότητες σε ένα σύστημα (μέθοδοι και χαρακτηριστικά) έχουν κατανεμηθεί στις κλάσεις [28]. Στη συνέχεια εφαρμόζονται τοπικά αλγόριθμοι αναζήτησης βελτιστοποίησης προκειμένου να εξάγουν τη βέλτιστη σχεδίαση, δηλαδή το σχεδιασμό ο οποίος αποτελείται από τα ίδια συστατικά, τα οποία βελτιστοποιούν τη συνάρτηση αποτελεσματικότητας. Η απόσταση μεταξύ της πραγματικής και της βέλτιστης σχεδίασης ενός λογισμικού, όσον αφορά τη διαφορά μεταξύ τους από τη τιμή της συνάρτησης αποτελεσματικότητας, μπορεί να θεωρηθεί ως το κεφάλαιο το οποίο χρειάζεται προκειμένου να μετατραπεί η πραγματική σχεδίαση λογισμικού στην αντίστοιχη βέλτιστη. Η προσπάθεια η οποία χρειάζεται για να καλυφθεί η «απόσταση» αυτή, ανάμεσα στον πραγματικό και στο βέλτιστο σχεδιασμό, είναι η προσπάθεια η οποία αφιερώνεται για την αποπληρωμή του τεχνικού χρέους και συνεπώς χαρακτηρίζεται ως $effort_r$ στην Εικόνα 11.

Σύμφωνα με τον B. Curtis [27], το κεφάλαιο για το τεχνικό χρέος, υπολογίζεται σαν μια συνάρτηση τριών μεταβλητών – τον αριθμό των προβλημάτων που πρέπει να διορθωθούν, το χρόνο που χρειάζεται για την αλλαγή των προβλημάτων και τέλος το κόστος που χρειάζεται για τη διόρθωση. Η συγκεκριμένη εργασία προσεγγιστικά υπολογίζει το κεφάλαιο με τη βοήθεια της δεύτερης μεταβλητής, δηλαδή το χρόνο που χρειάζεται για την αλλαγή των προβλημάτων αυτών. Στο σημείο αυτό, πρέπει να τονιστεί ότι υπήρχε παρόμοια έρευνα η οποία υπολόγιζε το κεφάλαιο σε χρήματα εστιάζοντας στη πρώτη μεταβλητή, δηλαδή στον αριθμό των προβλημάτων που πρέπει να αλλαχτούν [Estimating the Breaking Point for technical debt]. Στη παρούσα εργασία λοιπόν, έπρεπε να μετατραπεί προσεγγιστικά ο χρόνος που χρειάζεται για τη διόρθωση των σφαλμάτων στο κόστος για τα σφάλματα αυτά.

Το εργαλείο SonarQube το οποίο αναλύθηκε σε προηγούμενη ενότητα, βοήθησε πάρα πολύ στη διαδικασία αυτή. Το Sonar σύμφωνα με κανόνες που παραβιάζονται μπορεί να ανιχνεύσει λάθη τα οποία παραβιάζουν τους κανόνες αυτούς. Με βάση τις παραβιάσεις αυτές μπορεί να υπολογίσει το χρόνο που χρειάζεται για την αποσφαλμάτωση του συστήματος. Η πλατφόρμα αυτή διαθέτει πάνω από 409 κανόνες που οδηγούν σε προβλήματα στο λογισμικό. Οι κανόνες αυτοί χωρίζονται σε τρεις κατηγορίες με βάση το περιεχόμενο λάθος τους. Οι κατηγορίες αυτές είναι:

- Bugs
- Code Smells
- Vulnerabilities

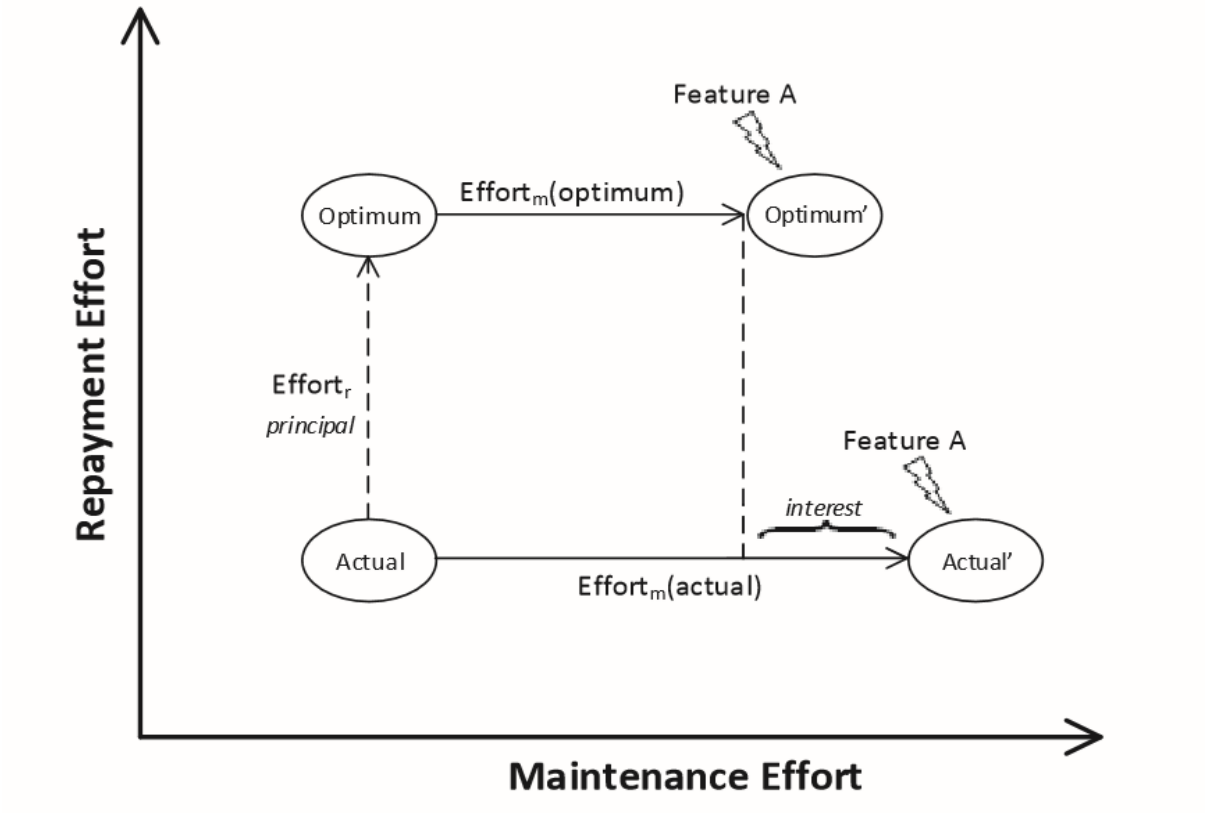
Το Sonar εξάγει το χρόνο αυτό, σε λεπτά της ώρας. Σύμφωνα με την αναφορά του U.S. Bureau of Labor Statistics για το 2014, το μέσο ωρομίσθιο για έναν προγραμματιστή είναι 45.81$ [National Occupational Employment and Wage Estimates]. Δεδομένου λοιπόν του χρόνου αποπληρωμής, αλλά και του ωρομίσθιου ενός προγραμματιστή μπορεί να υπολογιστεί το κόστος του κεφαλαίου. Οι υπολογισμοί που χρειάζεται να γίνουν είναι:

- Μετατροπή της μετρικής που εξάγει το Sonar από λεπτά σε ώρες
- Πολλαπλασιασμός των ωρών * μέσο ωρομίσθιο ενός προγραμματιστή (45.81$)

Οι παραπάνω ενέργειες θα οδηγήσουν σε μια προσέγγιση υπολογισμού του κόστους που απαιτείται για τη μετατροπή ενός συστήματος σε βέλτιστο.

2.4.2 Υπολογισμός του τόκου

Αντίστοιχα η προσπάθεια η οποία σχετίζεται με την ενίσχυση της λειτουργικότητας ή της διόρθωσης σφαλμάτων, αφορά τη συντήρηση του λογισμικού και χαρακτηρίζεται ως $effort_m$. Στην Εικόνα 11 απεικονίζεται ένα διάγραμμα μεταξύ των δυο προσπαθειών και επεξηγεί τι θεωρείται κεφάλαιο και τι τόκος.



Διάγραμμα 1: Προσπάθεια αποπληρωμής και προσπάθεια συντήρησης.

Πιο αναλυτικά, στον άξονα του x (οριζόντιος άξονας) απεικονίζεται η προσπάθεια που χρειάζεται για τη συντήρηση του λογισμικού, ενώ στον άξονα του y (κάθετος άξονας) η προσπάθεια για την αποπληρωμή του τεχνικού χρέους. Υπό κανονικές συνθήκες, η προσπάθεια για τη συντήρηση (π.χ. η προσθήκη μιας λειτουργίας "Feature A") ενός λογισμικού το οποίο δεν είναι στο βέλτιστο σχεδιασμό θα είναι μεγαλύτερη από τη προσπάθεια που θα χρειαστεί να προσθέσεις τη λειτουργία αυτή στο αντίστοιχο σύστημα με βέλτιστη σχεδίαση. Όπως διακρίνεται λοιπόν και στην Εικόνα 11 η διαφορά μεταξύ των δυο προσπαθειών, αντιστοιχεί στον τόκο, ο οποίος συσσωρεύεται κατά τη διάρκεια μετάβασης από την έκδοση t (πριν τη προσθήκη της λειτουργίας "Feature A") στην t+1. Όπως είναι φυσικό, λόγω της φύσης της εξέλιξης του λογισμικού, δεν είναι δυνατό να προβλεφθεί το είδος των αλλαγών που θα πραγματοποιηθούν σε μελλοντικές εκδόσεις λογισμικού. Για να αντιμετωπιστεί αυτός ο περιορισμός και να διατηρηθεί το μοντέλο το οποίο χρησιμοποιείται, εξετάζονται προηγούμενες αλλαγές στο ιστορικό του συστήματος. Μπορεί να χρησιμοποιηθεί, οποιοδήποτε μέτρο για το ποσοτικό προσδιορισμό της προηγούμενης προσπάθειας. Στη παρούσα διπλωματική εργασία, χρησιμοποιήθηκε σαν μέτρο ο μέσος αριθμός γραμμών που προστίθεται σε κάθε έκδοση. Οι προστιθέμενες γραμμές αντικατοπτρίζουν τη προσπάθεια για την εισαγωγή νέων λειτουργιών και σε κάποιο βαθμό τη προσπάθεια για τη τροποποίηση των υφισταμένων οντοτήτων.

Είναι εύκολο να κατανοηθεί ότι η προσθήκη k λειτουργιών σε ένα σύστημα με υψηλή ποιότητα λογισμικού (που ορίζεται ανάλογα με τα αποτελέσματα της συνάρτησης αποτελεσματικότητας), θα είναι ευκολότερο από τη προσθήκη των ίδιων λειτουργιών σε ένα σύστημα με χαμηλή ποιότητα λογισμικού. Για γενικούς λόγους, υποτίθεται ότι η προσπάθεια συντήρησης είναι ανάλογη της ποιότητας σχεδίασης:

$$Effort_m = c * FitnessValue$$

Εξίσωση 1: Προσπάθεια συντήρησης

όπου c ορίζεται μια αυθαίρετη σταθερά, στη περίπτωση μας ο μέσος αριθμός γραμμών που προστίθεται ανά έκδοση.

Σύμφωνα με τη προτεινόμενη προσέγγιση η αξία της συνάρτησης αποτελεσματικότητας είναι γνωστή για το πραγματικό αλλά και για το βέλτιστο σχεδιασμό λογισμικού. Επομένως, η αναλογία της θεωρητικής προσπάθειας (εάν η συντήρηση εφαρμοστεί στο βέλτιστο σχεδιασμό), με την πραγματική προσπάθεια μπορεί να υπολογιστεί ως:

$$\frac{Effort_m(\beta\acute{\varepsilon}\lambda\tau\iota\sigma\tau o\upsilon)}{effort_r\,(\pi\rho\alpha\gamma\mu\alpha\tau\iota\kappa o\acute{\upsilon})} = \frac{c * FitnessValue(\beta\acute{\varepsilon}\lambda\tau\iota\sigma\tau o\upsilon)}{c * FitnessValue(\pi\rho\alpha\gamma\mu\alpha\tau\iota\kappa o\acute{\upsilon})} \Rightarrow$$

$$Effort_m(\beta\acute{\varepsilon}\lambda\tau\iota\sigma\tau o\upsilon) = \frac{FitnessValue(\beta\acute{\varepsilon}\lambda\tau\iota\sigma\tau o\upsilon)}{FitnessValue(\pi\rho\alpha\gamma\mu\alpha\tau\iota\kappa o\acute{\upsilon})} * Effort_r(\pi\rho\alpha\gamma\mu\alpha\tau\iota\kappa o\acute{\upsilon})$$

Εξίσωση 2: Αναλογία βέλτιστου με πραγματικού σχεδιασμού

Επομένως, εάν η πραγματική προσπάθεια για συντήρηση λαμβάνεται ως ο μέσος όρος της προηγούμενης προσπάθειας για συντήρηση (όπως ο αριθμός των προστιθέμενων γραμμών k), τότε η βέλτιστη προσπάθεια μπορεί άμεσα να συνταχθεί. Ως αποτέλεσμα, ο τόκος που συσσωρεύεται μεταξύ δυο διαδοχικών εκδόσεων, μπορεί να οριστεί ως η διαφορά μεταξύ της πραγματικής και της βέλτιστης προσπάθειας που είναι ίσο με:

$$Επιτόκιο\ (Interest) = \Delta Effort = k * \left(1 - \frac{FitnessValue(βέλτιστου)}{FitnessValue(πραγματικού)}\right)$$

Εξίσωση 3: Εξίσωση για τον συσσωρευμένο τόκο

Επομένως γνωρίζοντας τον αριθμό των γραμμών που προστίθενται κατά μέσο όρο, τον αριθμό των γραμμών που γράφει ο προγραμματιστής ανά ώρα, αλλά και το ωρομίσθιο του προγραμματιστή, καθίσταται δυνατός ο υπολογισμός του επιτοκίου σε μορφή χρήματος. Οι ενέργειες που πρέπει να γίνουν για τη μετατροπή του επιτοκίου σε χρήματα είναι:

- Μέση τιμή από τη συνάρτηση αποτελεσματικότητας (Επιπλέον προσπάθεια (τόκος) για κάθε γραμμή κώδικα) .
- Πολλαπλασιασμός με το μέσο αριθμό προστιθέμενων γραμμών (Επιπλέον προσπάθεια (τόκος) για όλες τις γραμμές κώδικα) .
- Διαίρεση του τόκου για όλες τις γραμμές κώδικα με τις γραμμές που γράφει ο προγραμματιστής ανά ώρα. (Ώρες που απαιτούνται για την ολοκλήρωση των σφαλμάτων)
- Πολλαπλασιασμός με το ωρομίσθιο του προγραμματιστή (Τόκος σε $).

Σύμφωνα με το US Bureau of labour statistics ένας προγραμματιστής γράφει 25 γραμμές κώδικα την ώρα[30], και το ωρομίσθιο του όπως αναφέρθηκε στη προηγούμενη ενότητα είναι $45.81 [29]. Ξέροντας λοιπόν το κεφάλαιο που απαιτείται για τη βελτίωση του συστήματος και του επιτοκίου που συσσωρεύεται σε κάθε έκδοση, είναι εύκολος ο υπολογισμός της έκδοσης λογισμικού κατά την οποία θα πραγματοποιηθεί το σημείο ισορροπίας, με την εξίσωση:

$$versions = \frac{Prinipal(\$)}{Interest(\$)}$$

Εξίσωση 4: Εξίσωση για τον υπολογισμό του σημείου ισορροπίας

Θα πρέπει να σημειωθεί ότι πέρα από τη γραμμική φύση της αύξησης του επιτοκίου σε αυτό το απλοποιημένο μοντέλο, δεν λαμβάνονται υπόψη άλλες οικονομικές παράμετροι, όπως ο πληθωρισμός, αλλά μπορούν εύκολα να ενσωματωθούν στη προτεινόμενη ανάλυση. Επιπλέον, μια ντετερμινιστική πρόβλεψη των μελλοντικών προσπαθειών είναι ένα φιλόδοξο έργο διότι πολλοί παράγοντες (επιχειρηματικές απαιτήσεις και τεχνικές αλλαγές) έχουν σημαντικό ρόλο. Θα ήταν ωστόσο ενδιαφέρον σε μελλοντική έρευνα να εξετασθεί μια στοχαστική προσέγγιση ώστε αυτοί οι παράγοντες να λαμβάνονται ως πιθανότητες.

2.4.3 Συνάρτηση αποτελεσματικότητας

Έχει αναφερθεί πολλές φορές σε αυτή την ενότητα η συνάρτηση αποτελεσματικότητας, με βάση την οποία ορίζουμε τη διαφορά μεταξύ της πραγματικής και της βέλτιστης σχεδίασης. Η συνάρτηση αυτή είναι ένας ιδιαίτερος τύπος αντικειμενικής συνάρτησης που χρησιμοποιείται για να συνοψίσει πόσο κοντά είναι ο πραγματικός σχεδιασμός στην επίτευξη των καθορισμένων στόχων. Στη παρούσα εργασία η συνάρτηση αποτελεσματικότητας χρησιμοποίησε ως ορίσματα μετρικές του λογισμικού και η τελική τιμή της ήταν ο μέσος όρος όλων των τιμών. Οι μετρικές που χρησιμοποιήθηκαν για τη συνάρτηση ήταν:

1. **Depth of Inheritance Tree (DIT):** Το επίπεδο της κληρονομικότητας, η κλάση root έχει το επίπεδο 0.
2. **Number of Children Classes (NOCC)**: Ο αριθμός των υποκλάσεων που έχει η κλάση.
3. **Message Passing Coupling (MPC):** Ο αριθμός των σταλμένων μηνυμάτων.
4. **Response For a Class (RFC):** Ο αριθμός των τοπικών συναρτήσεων συν τον αριθμό των μεθόδων που καλούνται από τις τοπικές συναρτήσεις μέσα στη κλάση.
5. **Lack of Cohesion Of Methods (LCOM):** Ο αριθμός των διαχωρισμένων μεθόδων (μέθοδοι που δεν αλληλεπιδρούν μεταξύ τους) μέσα στη κλάση.
6. **Data Abstraction Coupling (DAC):** Ο αριθμός των αφηρημένων τύπων που έχουν οριστεί μέσα στη κλάση.
7. **Weighted Method per Class (WMPC):** Συνολική πολυπλοκότητα για όλες τις μεθόδους μέσα στη κλάση.
8. **Number of Methods (NOM):** Αριθμός των μεθόδων μέσα στη κλάση.
9. **Line of Code (Size1):** Ο αριθμός των ερωτηματικών (;) μέσα στη κλάση.
10. **Number of Properties (Size2):** Ο αριθμός των χαρακτηριστικών και των μεθόδων μέσα στη κλάση.

Στη συνέχεια εφαρμόζεται ο αλγόριθμος για να βρει το σύστημα με τη βέλτιστη τιμή. Ωστόσο δεν είναι βέλτιστη πάντα η μεγαλύτερη τιμή, ούτε πάντα η μικρότερη. Πιο συγκεκριμένα για τις παραπάνω μετρικές η βέλτιστη τιμή είναι:

1. **Depth of Inheritance Tree (DIT):** Μεγαλύτερη τιμή
2. **Number of Children Classes (NOCC)**: Μεγαλύτερη τιμή.
3. **Message Passing Coupling (MPC):** Μικρότερη τιμή.
4. **Response For a Class (RFC):** Μικρότερη τιμή.
5. **Lack of Cohesion Of Methods (LCOM):** Μικρότερη τιμή.
6. **Data Abstraction Coupling (DAC):** Μικρότερη τιμή.
7. **Weighted Method per Class (WMPC):** Μικρότερη τιμή.
8. **Number of Methods (NOM):** Μικρότερη τιμή.
9. **Line of Code (Size1):** Μικρότερη τιμή.
10. **Number of Properties (Size2):** Μικρότερη τιμή.

Έχοντας τις βέλτιστες τιμές, είναι πλέον εύκολο να υπολογιστεί η απόσταση της πραγματικής από τη βέλτιστη, και κατ' επέκταση ο υπολογισμός του επιτοκίου με τη συνάρτηση (2).

Κεφάλαιο 3ο - Ανάπτυξη λογισμικού του συστήματος

Στο παρόν κεφάλαιο αναλύεται διεξοδικά ο τρόπος λειτουργίας του προγράμματος καθώς και οι αλγόριθμοι που χρησιμοποιήθηκαν σε κάθε τμήμα ξεχωριστά. Στόχος του προγράμματος είναι η εύρεση του σημείου ισορροπίας στο τεχνικό χρέος. Οι τιμές από το κεφάλαιο και τον τόκο που χρησιμοποιούνται στην εξαγωγή του αποτελέσματος αναλύονται από ειδικά εργαλεία και αλγορίθμους οι οποίοι αναλύθηκαν στο προηγούμενο κεφάλαιο.

3.1 Ανάλυση ενός έργου με το εργαλείο SonarQube

Όπως έχει προαναφερθεί το εργαλείο Sonar είναι ένα εργαλείο το οποίο εξάγει μετρικές και πληροφορίες για ένα έργο. Για την εκτέλεση του Sonar απαιτείται η εγκατάσταση ενός λογισμικού (sonar-runner ή sonar-scanner), το οποίο θα μπορέσει να αναλύσει το project. Στο λογισμικό αυτό, υπάρχει ένα αρχείο διαμόρφωσης στο οποίο επιλέγονται οι βασικές επιλογές για τον κώδικα (όνομα έργου, όνομα προγραμματιστή, νούμερο έκδοσης κ.α.). Ωστόσο, για καλύτερα αποτελέσματα στην έρευνα έπρεπε να αναλυθούν αρκετά έργα από διάφορες κατηγορίες. Σε αυτή τη προσπάθεια ήταν χρήσιμος ο εξυπηρετητής Github, καθώς ο ίδιος διαθέτει πάρα πολλά αποθετήρια από έργα και η πρόσβαση σε αυτά είναι εύκολη. Επομένως, η βασική πηγή έργων ήταν το Github.

Ένας ακόμη στόχος της έρευνας, εκτός από τον υπολογισμό του σημείου ισορροπίας ήταν η μετατόπισή του (για παράδειγμα εάν το σημείο ισορροπίας απομακρύνεται ή έρχεται πιο κοντά ύστερα από την απελευθέρωση μιας καινούργιας έκδοσης). Το Github παρέχει τη δυνατότητα πρόσβασης σε παλαιότερα commits, τα οποία έπρεπε επίσης να αναλυθούν. Σε αυτό το σημείο χρησιμοποιήθηκε το script στη Perl,με τη βοήθεια του οποίου, γινόταν αυτόματα η ανάλυση όλων των έργων. Πιο συγκεκριμένα το script αυτό δέχεται το git repository από ένα έργο, και με την εντολή git log λαμβάνει πληροφορίες σχετικά με όλα τα commits που έχουν πραγματοποιηθεί σε αυτό. Οι πληροφορίες αυτές αποθηκεύονται σε ένα αρχείο με όνομα gitLog και στην Εικόνα 11 απεικονίζεται η γενική μορφή του. Για κάθε commit αποθηκεύεται η ημερομηνία που έγινε το commit, ο προγραμματιστής που το έκανε και ο αριθμός του commit. Όπως απεικονίζεται στην Εικόνα 11, η πρώτη γραμμή αναφέρεται στον αριθμό του commit, η δεύτερη στο όνομα του συγγραφέα (ενδεχομένως να μην είναι προγραμματιστής), η τρίτη στην ημερομηνία που πραγματοποιήθηκε το commit, και στη συνέχεια εμφανίζεται το μήνυμα το οποίο συνόδευε το commit. Στην Εικόνα 11, για παράδειγμα, καταγράφονται πληροφορίες για τρία commits.

```
Open ▾
commit c4ff0a2d6cc12e0b7f39e814c9a1dee2db5b5dca
Author: bobbylight <robert.e.futrell@gmail.com>
Date:   2016-09-18

    Bumping version to 2.6.1-SNAPSHOT          <= Commit's Message

commit 412ad0d8ef8dc6a29cbf73b5b6f0461f966ef7f0
Author: bobbylight <robert.e.futrell@gmail.com>
Date:   2016-09-17

    Preparing for 2.6.0 release                <= Commit's Message

commit 98f8c890e6d9e52a3ab8d0518f302eae7a6cb574
Author: bobbylight <robert.e.futrell@gmail.com>
Date:   2016-08-10

    Fix #31: Java: Annotated method parameters incorrectly identified as errors
```

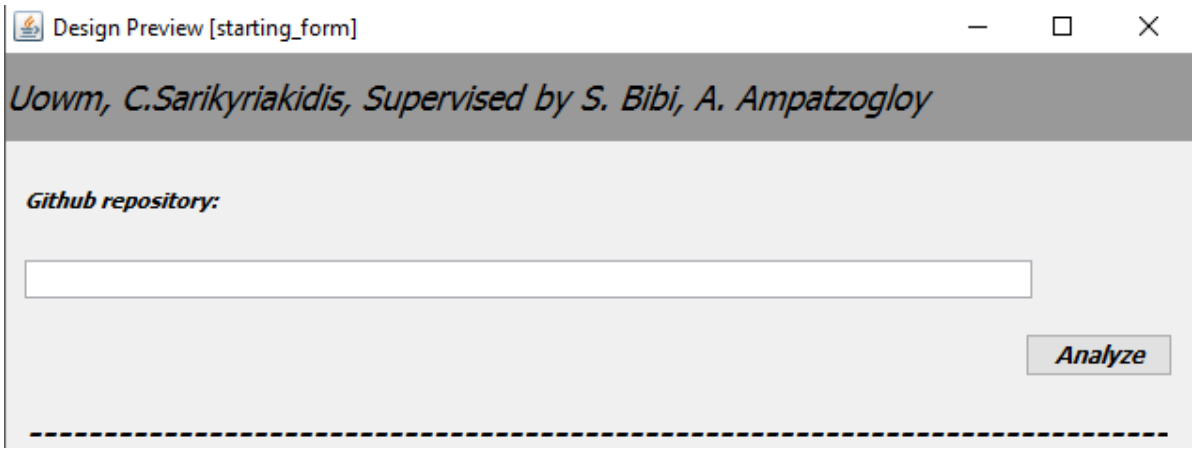
Εικόνα 11: Μορφή αρχείου Gitlog

Στη συνέχεια για κάθε ένα commit γίνεται η ανάλυση από το Sonarqube. Για την επιτυχή ανάλυση του Sonar μορφοποιείται κατάλληλα το αρχείο διαμόρφωσης για κάθε commit. Αρκούν οι πληροφορίες που έχουν εξαχθεί από το git log, δηλαδή το όνομα του έργου, ο αριθμός του commit, η ημερομηνία και ο προγραμματιστής που έκανε το commit, για να γίνει σωστή ανάλυση του έργου. Στην Εικόνα 12 απεικονίζεται η γενική μορφή του αρχείου διαμόρφωσης που περιέχει τις απαραίτητες πληροφορίες που απαιτούνται για την ανάλυση του έργου. Στο σημείο αυτό αξίζει να σημειωθεί ότι το αρχείο διαμόρφωσης περιέχει διαφορετικές πληροφορίες για κάθε commit, καθώς όπως είναι αυτονόητο υπάρχουν διαφορετικά στοιχεία για κάθε commit (π.χ. διαφορετική ημερομηνία του κάθε commit ή διαφορετικός αριθμός commit). Ύστερα από την ολοκλήρωση εκτέλεσης του script, η ανάλυση του έργου έχει ολοκληρωθεί για όλα τα commits, και οι μετρικές για κάθε ένα commit είναι αποθηκευμένα στη βάση.

```
Open ▾
# Required metadata
sonar.projectKey=b598620db48fd7fead916251630cadbcfe81067d
sonar.projectName=RSTALanguageSupport
sonar.projectVersion=123
sonar.projectDate=2010-04-19
sonar.projectAuthor=bobbylight <robert@fifesoft.com>

# Comma-separated paths to directories with sources (required)
sonar.sources=.

# Language
#sonar.language=java

# Encoding of the source files
sonar.sourceEncoding=UTF-8
```

Εικόνα 12: Μορφή αρχείο διαμόρφωσης για το πρόγραμμα Sonar

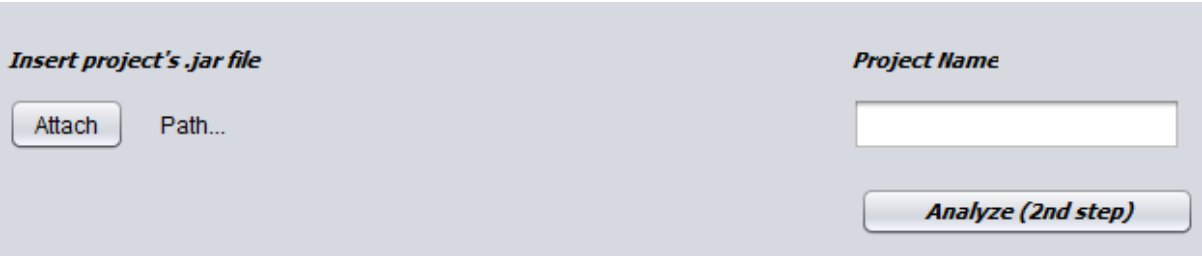
Έχει αναφερθεί ότι το εργαλείο που κατασκευάστηκε, εκτός από λειτουργικό πρέπει να είναι εύχρηστο και φιλικό προς το χρήστη, για αυτό το λόγο κατασκευάστηκε ένα γραφικό περιβάλλον για το χρήστη (GUI) προκειμένου η εκτέλεση του script να μην απαιτεί μακροσκελείς εντολές. Στο περιβάλλον το οποίο έχει κατασκευαστεί (Εικόνα 13), αρκεί η συμπλήρωση του git repository και με το πάτημα του κουμπιού analyze, πραγματοποιείται η ανάλυση του έργου, χωρίς να απαιτείται περεταίρω ενέργειες.



Εικόνα 13: Γραφικό περιβάλλον για το script σε Perl

3.2 Ανάλυση μετρικών από το εργαλείο metrics_calculator

Ορισμένες μετρικές που χρειάζονται για τη συνάρτηση αποτελεσματικότητας, δεν εξάγονται από το εργαλείο Sonar, επομένως πρέπει να χρησιμοποιηθεί το δεύτερο εργαλείο ανάλυσης το οποίο ονομάζεται metrics_calculator. Το εργαλείο δέχεται σαν είσοδο το αρχείο jar του έργου και όχι το git repository όπως το Sonar. Οπότε για την επιτυχή εκτέλεση του εργαλείου, απαιτείται το αρχείο jar του συγκεκριμένου έργου. Ύστερα από την εκτέλεση του, εξάγονται οι τιμές των μετρικών που χρειάζονται για τον υπολογισμό του σημείου ισορροπίας. Είναι γνωστό ότι το αρχείο jar μπορεί να έχει οποιοδήποτε όνομα (π.χ. test-project-name_v32.jar) επομένως σε πρώτη φάση δεν μπορεί να γίνει αυτόματη αντιστοίχιση μετρικών με έργο. Η αντιστοίχιση πραγματοποιείται χειροκίνητα από το χρήστη, ο οποίος εκτός από το αρχείο jar θα πρέπει να συμπληρώσει το όνομα του έργου στο οποίο θα αντιστοιχούν οι τιμές αυτές. Το GUI που αναφέρεται στη συγκεκριμένη χρήση του εργαλείου απεικονίζεται στην Εικόνα 14.



Εικόνα 14: Γραφικό περιβάλλον για την ανάλυση του δεύτερου εργαλείου

Πιο συγκεκριμένα, υπάρχει ένα κουμπί τύπου επιλογής αρχείου, για να συμπληρώνεται το μονοπάτι του αρχείου, και μια φόρμα στην οποία θα πρέπει να εισάγεται το όνομα του έργου. Έπειτα από την ανάλυση και των δύο εργαλείων, το στάδιο ανάλυσης έχει τελειώσει

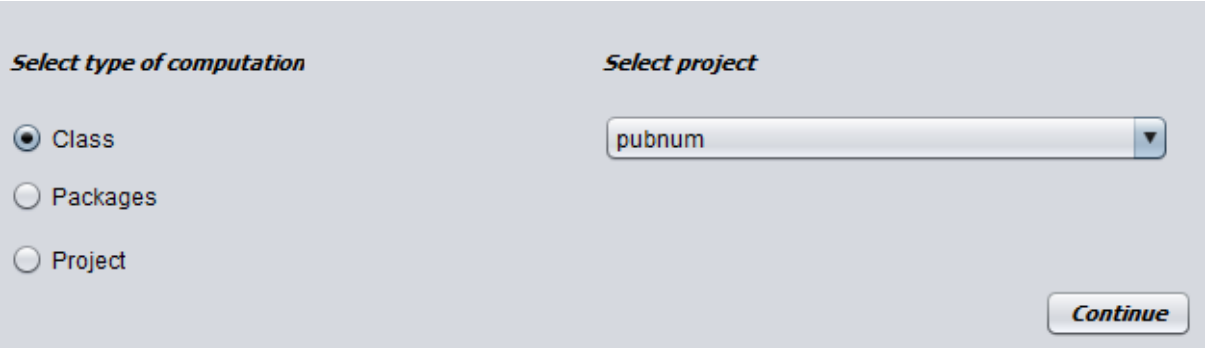
και όλες οι πληροφορίες είναι αποθηκευμένες στη βάση, και έτοιμες για χρήση. Το εργαλείο αυτό διαφέρει από το προηγούμενο (Sonar) ως προς την εξαγωγή αποτελεσμάτων. Το προηγούμενο εργαλείο υπολόγιζε τις τιμές των αποτελεσμάτων σε τρία επίπεδα, σε επίπεδο έργου, σε επίπεδο πακέτου αλλά και σε πακέτο κλάσεων. Το συγκεκριμένο εργαλείο εξάγει πληροφορίες μόνο σε επίπεδο κλάσεων, επομένως πρέπει να υπολογιστούν από το πρόγραμμα οι τιμές των μετρικών σε επίπεδο πακέτου (μέσος όρος τιμών των κλάσεων που αντιστοιχούν στο συγκεκριμένο πακέτο) αλλά και σε επίπεδο έργου (μέσος όρος τιμών των κλάσεων όλου του έργου). Στον Πίνακα 3 απεικονίζεται ένας ενδεικτικός πίνακας (δεν εμφανίζονται όλες οι μετρικές) με τις τιμές των μετρικών ύστερα από την εκτέλεση του εργαλείου. Δεν απαιτείται η σειριακή εκτέλεση των δυο εργαλείων, δηλαδή δεν είναι απαραίτητη η εκτέλεση του εργαλείου metrics_calculator ακριβώς μετά την εκτέλεση του Sonar. Μπορεί να γίνει ανάλυση με το πρώτο εργαλείο για όλα τα έργα τα οποία πρόκειται να εξετασθούν και στη συνέχεια η εκτέλεση του δεύτερου εργαλείου για όλα τα έργα. Σε κάθε περίπτωση βέβαια, η σειρά εκτέλεσης μπορεί να είναι τυχαία, αλλά για τον υπολογισμό του σημείου ισορροπίας απαιτείται οπωσδήποτε η εκτέλεση και των δυο εργαλείων.

Name	WMC	DIT	NOCC	CBO	RFC
org.fife.rsta.ac.sh.ShellLanguageSupport	6	3	0	2	24
org.fife.rsta.ac.sh.ShellFunctionCompletion	4	1	0	2	46
org.fife.rsta.ac.sh.ShellCompletionProvider	7	2	0	3	17
org.fife.rsta.ac.css.CommonFontCompletionGenerator	1	2	0	2	7
org.fife.rsta.ac.css.CompletionGenerator	1	1	5	6	2
org.fife.rsta.ac.css.BasicCssCompletion	1	1	0	2	6
org.fife.rsta.ac.css.TimeCompletionGenerator	2	2	0	2	15
org.fife.rsta.ac.css.PercentageOrLengthCompletionGenerator	2	2	0	2	15
org.fife.rsta.ac.css.CssCellRenderer	4	1	0	1	18
org.fife.rsta.ac.css.CssCompletionProvider	2	1	1	4	12
org.fife.rsta.ac.css.PropertyCompletion	1	1	0	2	9
org.fife.rsta.ac.css.ColorCompletionGenerator	3	2	0	2	19
org.fife.rsta.ac.css.CssLanguageSupport	5	3	1	4	22
org.fife.rsta.ac.css.BorderStyleCompletionGenerator	1	2	0	2	7
org.fife.rsta.ac.css.PropertyValueCompletionProvider	17	1	1	10	124
org.fife.rsta.ac.css.IconFactory	3	1	0	2	18
org.fife.rsta.ac.less.LessLanguageSupport	1	4	0	3	6
org.fife.rsta.ac.less.LessCompletionProvider	1	2	0	3	5
org.fife.rsta.ac.less.LessCodeCompletionProvider	5	2	0	2	30
org.fife.rsta.ac.jsp.JspLanguageSupport	6	4	0	4	30

Πίνακας 3: Ενδεικτικός πίνακας αποτελεσμάτων metrics_calculator

3.3 Υπολογισμός του σημείου ισορροπίας

Καθώς έχει ολοκληρωθεί η ανάλυση των έργων μπορεί πλέον να πραγματοποιηθεί ο υπολογισμός του σημείου ισορροπίας για το project αυτό. Όπως έχει αναφερθεί, το εργαλείο μπορεί να υπολογίσει το σημείο ισορροπίας σε τρία επίπεδα. Τα επίπεδα αυτά είναι η κλάση, το πακέτο και όλο το έργο. Επομένως, το γραφικό περιβάλλον (Εικόνα 15) το οποίο δημιουργήθηκε για τον υπολογισμό του σημείου αυτού, διαθέτει τρία radio buttons και το κάθε ένα αντιστοιχεί σε ένα επίπεδο (από προεπιλογή είναι επιλεγμένο το επίπεδο κλάση). Επίσης στο γραφικό περιβάλλον έχει τοποθετηθεί ένα dropdown μενού το οποίο σαν τιμές έχει όλα τα project τα οποία έχουν αναλυθεί.



Εικόνα 15: Γραφικό περιβάλλον για τον υπολογισμό του bp

3.3.1 Υπολογισμός του σημείου ισορροπίας σε επίπεδο κλάσης

Όσον αφορά το επίπεδο κλάσης, ο αλγόριθμος για τον υπολογισμό του breaking point ακολουθεί τα εξής βήματα:

1. Υπολογίζεται η ομοιότητα της κλάσης που εξετάζεται με όλες τις υπόλοιπες κλάσεις του έργου.

2. Χρησιμοποιείται αλγόριθμος ταξινόμησης, ώστε να βρεθούν οι καλύτερες (πιο όμοιες) κλάσεις με αυτήν που εξετάζεται.

3. Εφαρμογή της συνάρτησης αποτελεσματικότητας στις 5 καλύτερες κλάσεις.

4. Εύρεση επιτοκίου με βάση τα αποτελέσματα της συνάρτησης αποτελεσματικότητας.

5. Εύρεση του breaking point

Πιο αναλυτικά, το πρώτο βήμα του αλγόριθμου είναι ο υπολογισμός της ομοιότητας μεταξύ των κλάσεων του συστήματος. Η ομοιότητα εκφράζεται ποσοστιαία με τη βοήθεια της εξίσωσης (5).

$$Similarity(\%) = \frac{abs(Metricvalue_{class1} - Metricvalue_{class2})}{\max(Metricvalue_{class1}, Metricvalue_{class2})} * 100$$

Εξίσωση 5: Υπολογισμός ομοιότητας

Η εξίσωση (5) χρησιμοποιείται για τις μετρικές:

- Number of statement,
- Number of function
- Non-comment lines of code
- Complexity
- Number of class
- Sqale Index

Αφού υπολογισθεί η ομοιότητα για κάθε μια μετρική, υπολογίζεται ο μέσος όρος όλων αυτών και το ποσοστό αυτό είναι η ομοιότητα μιας κλάσης με μιας άλλης.

Με βάση το ποσοστό ομοιότητας που έχει υπολογισθεί στο βήμα 1, εφαρμόζεται ένας αλγόριθμος ταξινόμησης, ώστε να βρεθούν οι πέντε πιο όμοιες κλάσεις. Αφού ολοκληρωθεί η ανάλυση, οι κλάσεις αυτές εμφανίζονται στον Πίνακα 4 ο οποίος αναφέρει το όνομα της κλάσης, το ποσοστό ομοιότητας με την εξεταζόμενη κλάση για κάθε μια μετρική και το συνολικό ποσοστό ομοιότητας.

Name of Investigated Class

Metrics/Class_Name	Class1	Class2	Class3	Class4	Class5
Similarity	62.13	62.13	61.43	61.14	61.14
Sqale Index	41.18	41.18	23.53	16.47	16.47
Ncloc	90.79	90.79	76.32	78.95	78.95
Classes	100.00	100.00	100.00	100.00	100.00
Statements	75.00	75.00	100.00	100.00	100.00
Function	50.00	50.00	50.00	50.00	50.00
Complexity	15.79	15.79	18.75	21.43	21.43

Πίνακας 4: Πίνακας με τις πιο όμοιες κλάσεις

Αφού έχουν βρεθεί οι πιο όμοιες κλάσεις, μπορεί να εφαρμοστεί η συνάρτηση αποτελεσματικότητας. Η συνάρτηση αυτή, εφαρμόζεται σε όλες τις κλάσεις μαζί και όχι σε κάθε μια ξεχωριστά. Πιο συγκεκριμένα, η συνάρτηση αυτή ανιχνεύει από τις καλύτερες πέντε κλάσεις τη βέλτιστη τιμή και τη συγκρίνει με τη τιμή της μετρικής για τη κλάση που εξετάζεται. Δεν είναι απαραίτητο οι βέλτιστες τιμών των μετρικών να ανήκουν στην ίδια κλάση, για παράδειγμα για τη μετρική size1 μπορεί βέλτιστη τιμή να έχει η class1, ενώ για τη μετρική size2 μπορεί βέλτιστη τιμή να έχει η class3. Η συνάρτηση αποτελεσματικότητας υπολογίζεται από την εξίσωση (6).

$$fittedfunction = \frac{abs(metricvalue_{invclass} - metricvalue_{optclass})}{metricvalue_optclass}$$

Εξίσωση 6: Υπολογισμός συνάρτησης αποτελεσματικότητας

Όπου $metricvalue_{invclass}$ είναι η τιμή της μετρικής για τη κλάση η οποία εξετάζεται,

$Metricvalue_{optclass}$ είναι η τιμή της μετρικής για τη κλάση με τη βέλτιστη τιμή.

Όπως είχε τονιστεί η βέλτιστη τιμή μιας μετρικής δεν είναι πάντα η μεγαλύτερη τιμή, αλλά ούτε πάντα η μικρότερη, για αυτό το λόγο χρησιμοποιείται η απόλυτη τιμή, ώστε να μην αλλοιωθεί το αποτέλεσμα της συνάρτησης. Η συνάρτηση εφαρμόζεται σε δέκα μετρικές

επομένως εξάγει δέκα αποτελέσματα ένα για κάθε μετρική. Οι τιμές αυτές συναθροίζονται και υπολογίζεται ο μέσος όρος όλων αυτών.

Αφού έχει ολοκληρωθεί και η εφαρμογή της συνάρτησης αποτελεσματικότητας, το πρόγραμμα συνεχίζει στην εξαγωγή του επιτοκίου. Ο μέσος όρος που υπολογίστηκε από τα αποτελέσματα της συνάρτησης αποτελεί τον τόκο, ως επιπλέον προσπάθεια ανά commit. Η προσπάθεια αυτή όμως, καθώς ο τόκος συσσωρεύεται με την απελευθέρωση ενός καινούργιου commit, πρέπει να πολλαπλασιαστεί με μια σταθερά k (γραμμές κώδικα ανά commit) για να προκύψει η συνολική προσπάθεια και στη συνέχεια να υπολογιστεί ο συνολικός τόκος σε δολάρια.

Τέλος, εφαρμόζεται η Εξίσωση 4 και υπολογίζεται το σημείο ισορροπίας.

$$Σημείο\ Θραύσης = \frac{Κεφάλαιο(\$)}{Συσσωρευμένο\ Επιτόκιο(\$)}$$

3.3.2 Υπολογισμός του σημείου ισορροπίας σε επίπεδο πακέτου

Ο υπολογισμός του σημείου ισορροπίας ενός έργου σε επίπεδο πακέτου, είναι αρκετά όμοιος με το επίπεδο κλάσης. Τα βήματα που απαιτούνται για τον υπολογισμό του σημείου αυτού είναι:

1. Υπολογισμός της ομοιότητα του πακέτου που εξετάζεται με όλα τα υπόλοιπα πακέτα

2. Χρήση του αλγόριθμου ταξινόμησης, ώστε να βρεθούν τα καλύτερα (πιο όμοια) πακέτα με αυτό που εξετάζεται.

3. Εφαρμογή της συνάρτησης αποτελεσματικότητας στα 5 καλύτερα πακέτα.

4. Εύρεση επιτοκίου με βάση τα αποτελέσματα της συνάρτησης αποτελεσματικότητας.

5. Εύρεση του breaking point

Όπως έχει γίνει ήδη αντιληπτό, σε επίπεδο πακέτου, απαιτούνται ακριβώς οι ίδιες ενέργειες με το επίπεδο κλάσης. Η βασική διαφορά είναι ότι πλέον δεν εξετάζονται ομοιότητες ανάμεσα σε κλάσεις του συστήματος, αλλά σε πακέτα. Χρησιμοποιούνται ακριβώς οι ίδιες συναρτήσεις, χωρίς να αλλάξει απολύτως τίποτα. Στον Πίνακα 5 απεικονίζεται ο αντίστοιχος πίνακας ομοιότητας πακέτων.

Name of Investigated Package

Metrics/Class_Name	Package 1	Package 2	Package 3	Package 4	Package 5
Similarity	70.05	68.77	68.41	64.41	63.47
Sqale Index	50.64	36.86	81.68	23.40	31.41
Ncloc	71.51	68.92	72.94	75.19	69.15
Classes	71.43	100.00	71.43	83.33	71.43
Statements	64.94	61.95	48.29	56.64	45.56
Function	96.00	54.00	74.00	70.00	98.00
Complexity	65.81	90.91	62.10	77.92	65.25

Πίνακας 5: Πίνακας με τα πιο όμοια πακέτα

3.3.3 Υπολογισμός του σημείου ισορροπίας σε επίπεδο έργου

Το τρίτο επίπεδο το οποίο μπορεί να επιλεχθεί για να γίνει ο υπολογισμός του bp, είναι ολόκληρο το έργο. Τα βήματα τα οποία απαιτούνται για τον υπολογισμό είναι:

1. Υπολογισμός της ομοιότητας του έργου με άλλα έργα.

2. Χρήση του αλγόριθμου ταξινόμησης, ώστε να βρεθούν τα καλυτέρα (πιο όμοια) έργα με αυτό που εξετάζεται.

3. Εφαρμογή της συνάρτησης αποτελεσματικότητας στα 5 καλύτερα έργα.

4. Εύρεση επιτοκίου με βάση τα αποτελέσματα της συνάρτησης αποτελεσματικότητας

5. Εύρεση του breaking point

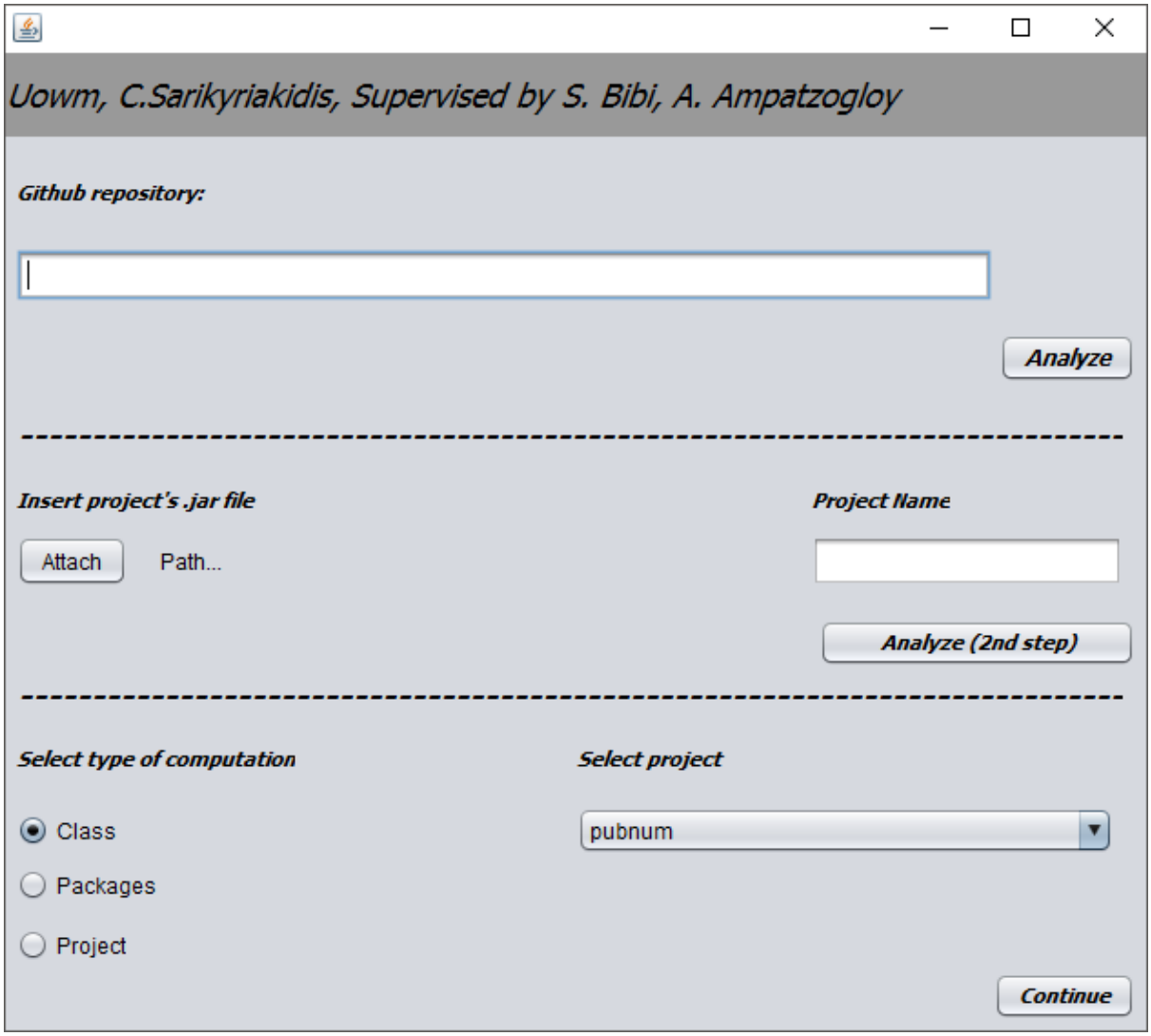
Τα βήματα για τον υπολογισμό του σημείου ισορροπίας σε επίπεδο έργου δε φαίνεται να διαφέρουν από τα βήματα των άλλων επιπέδων**.** Ωστόσο, υπάρχει μια ουσιαστική διαφορά στο συγκεκριμένο επίπεδο. Στα προηγούμενα επίπεδα εξετάζονταν είτε κλάσεις είτε πακέτα που ανήκαν στο ίδιο έργο και στο ίδιο commit, ενώ δεν ήταν δυνατή η εξέταση κλάσεων οι οποίες ανήκαν στο ίδιο έργο αλλά σε διαφορετικό commit ή απλώς σε άλλο έργο. Σε επίπεδο έργου, πλέον εξετάζεται το τελευταίο commit (το πιο πρόσφατο) του έργου με τα πιο πρόσφατα commits των άλλων έργων. Με αυτή τη μικρή αλλά ουσιαστική τροποποίηση διαφοροποιείται το συγκεκριμένο επίπεδο από τα άλλα δυο. Τα υπόλοιπα βήματα συνεχίζονται ακριβώς με τον ίδιο τρόπο όπως στα προηγούμενα επίπεδα και υπολογίζεται το σημείο ισορροπίας.

3.4 Παρουσίαση του εργαλείου και παραδείγματα εκτέλεσης

Στο συγκεκριμένο κεφάλαιο αναλύονται όλες οι απαιτήσεις και όλα τα βήματα τα οποία χρειάζονται για τον υπολογισμό του τελικού αποτελέσματος. Σε ορισμένα σημεία παρουσιάζονται κομμάτια από το GUI το οποίο κατασκευάστηκε. Συνοψίζοντας λοιπόν, το εργαλείο παρέχει μια αρχική φόρμα, στην οποία ο χρήστης έχει τη δυνατότητα να εκτελέσει τρεις λειτουργίες:

1. Ανάλυση ενός έργου με το εργαλείο Sonarqube
2. Ανάλυσης ενός έργου με το εργαλείο metrics_calculator
3. Υπολογισμός του σημείου ισορροπίας για ένα έργο

Η τελική μορφή της αρχικής φόρμας του εργαλείου απεικονίζεται στην Εικόνα 16. Πλέον ο χρήστης μπορεί να εκτελέσει οποιαδήποτε λειτουργία απλά με ένα κλικ, χωρίς να χρειάζονται διαφορετικά GUI για κάθε μια λειτουργία.



Εικόνα 16: Αρχική φόρμα εργαλείου

Οι πρώτες δυο λειτουργίες, ύστερα από την ολοκλήρωση τους, εμφανίζουν ένα dialog box το οποίο ενημερώνει το χρήστη ότι η λειτουργία ολοκληρώθηκε με επιτυχία ή για κάποιο ενδεχόμενο σφάλμα το οποίο προέκυψε. Φυσικά, **για** να μην υπάρχουν σφάλματα κατά την εκτέλεση μιας λειτουργίας, υπάρχουν κάποιοι περιορισμοί. Αυτοί οι περιορισμοί είναι:

- Να μη μπορεί να εκτελεστεί το script της Perl (δηλαδή το κουμπί 'Analyze') εάν η φόρμα είναι κενή ή το περιεχόμενο της δεν καταλήγει σε .git .
- Να μη μπορεί να εκτελεστεί το εργαλείο metrics_calculator (δηλαδή το κουμπί 'Analyze2nd step') εάν δεν έχει συμπληρωθεί το μονοπάτι για το jar αρχείο ή το όνομα του έργου.
- Όταν επιλέγεται ο υπολογισμός του σημείου ισορροπίας σε επίπεδο πακέτου ή κλάσης πρέπει να έχει αναλυθεί το συγκεκριμένο έργο και από τα δυο εργαλεία.
- Όταν επιλέγεται ο υπολογισμός του σημείου ισορροπίας σε επίπεδο έργου πρέπει να έχουν αναλυθεί όλα τα έργα και από τα δυο εργαλεία.

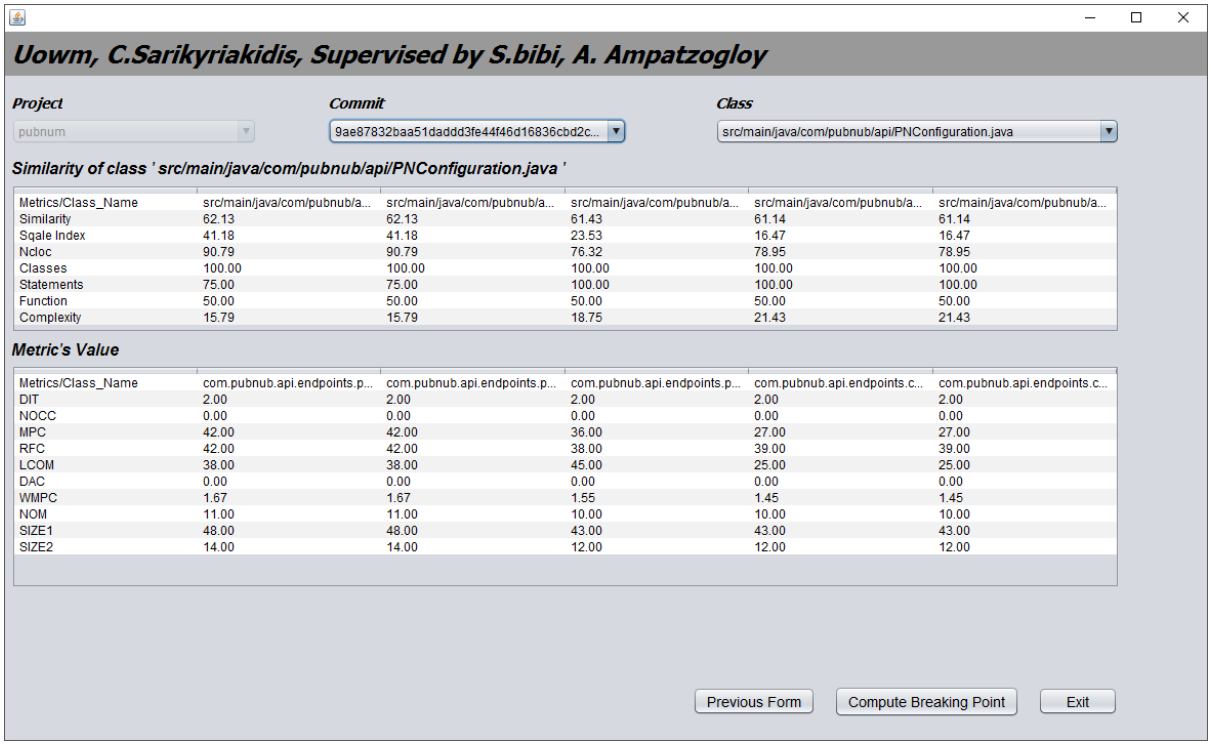
Η παραβίαση των περιορισμών εμποδίζει φυσικά την εκτέλεση της λειτουργίας και ενημερώνει το χρήστη με κατάλληλο μήνυμα (Εικόνα 17).

Εικόνα 17: Μήνυμα όταν το git repository είναι κενό ή δεν περιέχει κατάληξη .git

Η τρίτη λειτουργία είναι ο υπολογισμός του σημείου ισορροπίας. Φυσικά δε θα ήταν ωραίο αλλά ούτε εύκολο να ελεγχθεί το αποτέλεσμα αν αναφερόταν απλά στα πόσα commits εμφανίζεται το σημείο ισορροπίας. Για αυτό το λόγο σχεδιάστηκε και μια δεύτερη φόρμα η οποία απεικονίζεται στην Εικόνα 18. Διαθέτει αρχικά τρία dropdown μενού· το πρώτο είναι κλειδωμένο και έχει μέσα όλα τα ονόματα έργων τα οποία είναι στη βάση, το δεύτερο μενού αναφέρεται στα commits τα οποία έχουν γίνει για το συγκεκριμένο έργο και τέλος το τρίτο dropdown μενού αναφέρεται είτε στις κλάσεις είτε στο πακέτο είναι σε όλο το έργο. Στην περίπτωση που έχει επιλεχθεί το επίπεδο έργου κλειδώνει και το δεύτερο μενού καθώς όπως έχει προαναφερθεί χρησιμοποιείται το τελευταίο commit. Τα μενού αυτά προσφέρουν στο χρήστη τη δυνατότητα να επιλέξει για όποιο commit επιθυμεί μια κλάση ή ένα πακέτο και να υπολογίσει το σημείο ισορροπίας πάρα πολύ εύκολα. Στη συνέχεια κάτω από αυτά τα μενού αναφέρεται το όνομα της κλάσης (ή του πακέτου ή του έργου) και από κάτω εμφανίζεται ο πίνακας με τις ομοιότητες των άλλων κλάσεων, όπως ακριβώς στους Πίνακες 4 και 5.

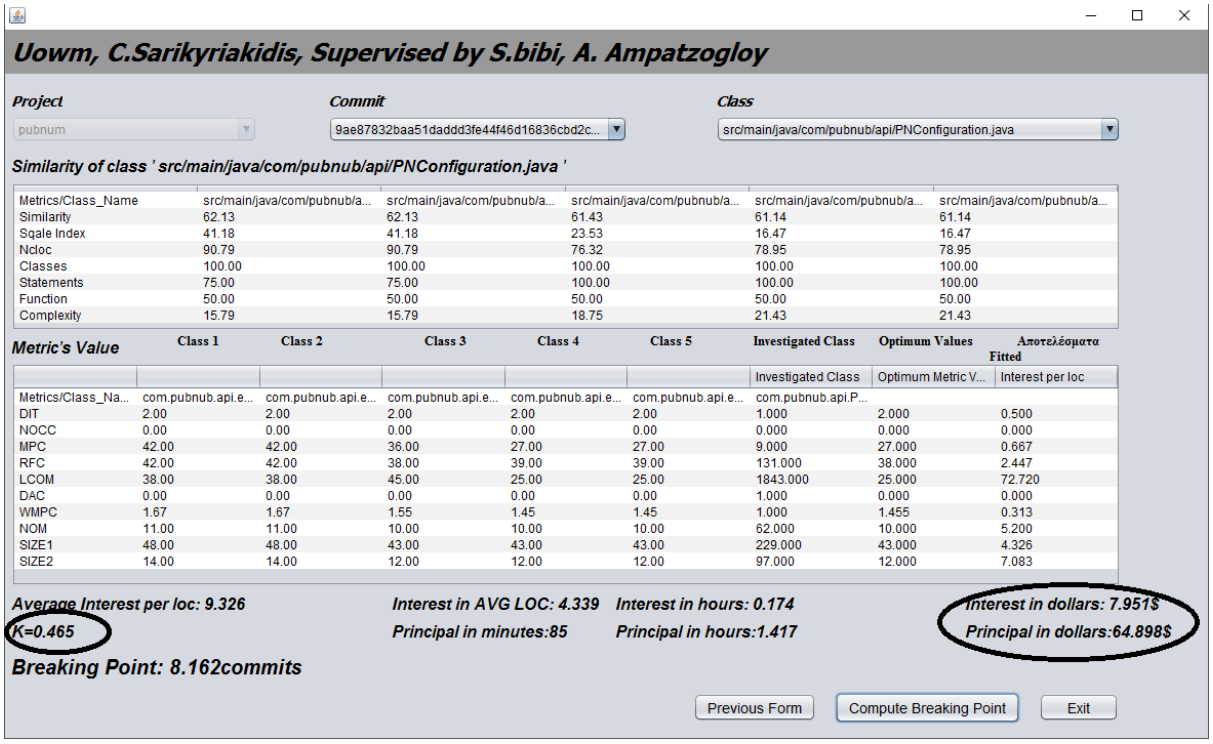
Στη συνέχεια εμφανίζεται ένας δεύτερος πίνακας ο οποίος περιέχει τις τιμές από τις μετρικές των πέντε καλύτερων κλάσεων. Οι τιμές αυτές θα χρησιμοποιηθούν από την συνάρτηση αποτελεσματικότητας όπως έχει αναλυθεί παραπάνω. Τέλος υπάρχουν τρία κουμπιά τα οποία ονομάζονται:

- Previous Form
- Compute Breaking Point
- Exit



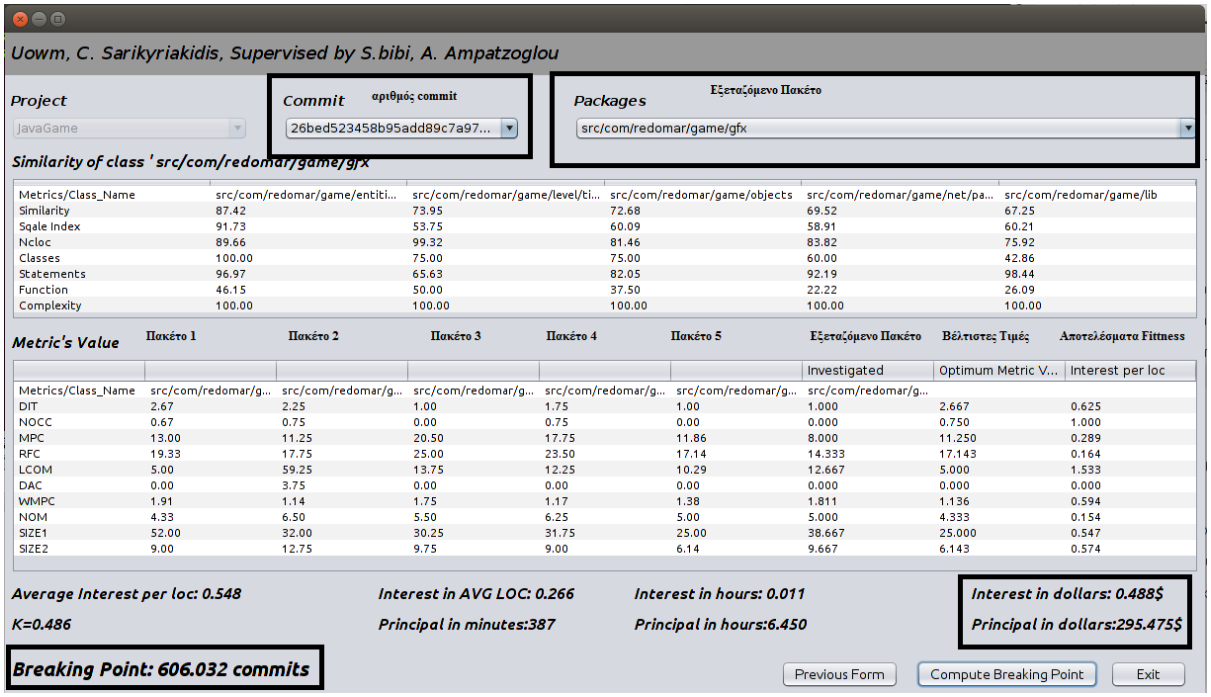
Εικόνα 18: Γραφικό περιβάλλον για τον υπολογισμό του bp

Το κουμπί Previous Form μεταφέρει το χρήστη στην αρχική φόρμα για να αλλάξει ενδεχομένως έργο ή για να αλλάξει επίπεδο. Το κουμπί Exit τερματίζει το πρόγραμμα και τέλος έμεινε το κουμπί Compute Breaking Point. Αφού πατηθεί το συγκεκριμένο κουμπί, εμφανίζονται τρεις επιπλέον στήλες στον δεύτερο πίνακα (Εικόνα 19) οι οποίες αναφέρονται στις τιμές μετρικών από την εξεταζόμενη κλάση, στις τιμές των βέλτιστων μετρικών και στο αποτέλεσμα της συνάρτησης αποτελεσματικότητας για κάθε μια μετρική. Επίσης, εμφανίζονται πληροφορίες για τον τόκο (σε μορφή χρόνου και πως μετατρέπεται σε δολάρια), για τη σταθερά K (γραμμές που προστίθενται ανά commit) και για το κεφάλαιο (σε μορφή χρόνου και πως μετατρέπεται σε δολάρια) και φυσικά το σημείο ισορροπίας σε commits. Στην Εικόνα 19 απεικονίζεται ένα παράδειγμα εκτέλεσης για το έργο pubnum, το commit με αριθμό commit 9ae87832baa51daddd3fe44f46d16836cbd2cb4b και στη κλάση PNConfiguration.java, αφού έχει πατηθεί το κουμπί για τον υπολογισμό του breaking point εισάγονται στο δεύτερο πίνακα με τις μετρικές οι τρεις στήλες οι οποίες βοηθάνε το χρήστη για την επαλήθευση του αποτελέσματος και τέλος εμφανίζεται το breaking point, το οποίο για τη συγκεκριμένη κλάση είναι 8.162 commits. Αυτό προέκυψε από τη διαίρεση του κεφαλαίου προς τον συσσωρευμένο τόκο, δηλαδή 64.898 δολάρια προς 7.951 δολάρια, το οποίο φυσικά ισούται με το 8.162 που είναι και το σημείο ισορροπίας. Παρατηρείται, επίσης, στην Εικόνα 19 πως ο μέσος όρος από τα αποτελέσματα της συνάρτησης αποτελεσματικότητας (Average Interest per Loc =9.326) πολλαπλασιάζεται με το k και αυτό αποτελεί τον συσσωρευμένο τόκο 4.339, στη συνέχεια μετατρέπεται σε ώρες και τέλος σε δολάρια. Αντίστοιχα, μετατρέπεται και το κεφάλαιο που αναφέρεται ως Principal στο εργαλείο.



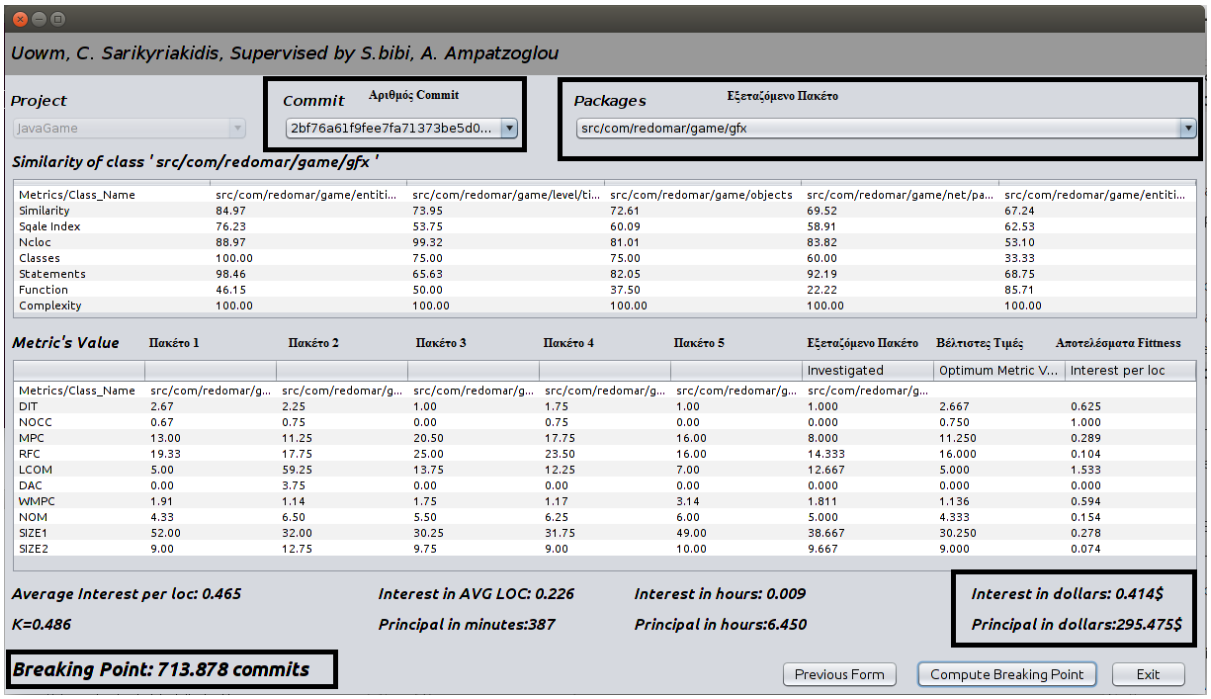
Εικόνα 19: Παράδειγμα εκτέλεσης του εργαλείου σε επίπεδο κλάσης

Στη συνέχεια, θα αναλυθεί ένα παράδειγμα εκτέλεσης για τον υπολογισμό του σημείου ισορροπίας σε επίπεδο πακέτου. Το προηγούμενο παράδειγμα αναφερόταν σε ένα έργο ανοιχτού κώδικα, με την ονομασία pubnum. Το έργο το οποίο θα αναλυθεί ονομάζεται JavaGame και είναι επίσης ένα έργο ανοιχτού κώδικα. Στην Εικόνα 20 απεικονίζεται ο υπολογισμός του σημείου ισορροπίας στο πακέτο src/com/redomar/game/gfx, από το έργο JavaGame και αριθμό commit 26bed523458b95add89c7a975fad42b401912118. Όπως γίνεται αντιληπτό από την Εικόνα 20, το σημείο ισορροπίας εμφανίζεται στα 606.032 commits. Το αποτέλεσμα αυτό προέρχεται από τη διαίρεση του κεφαλαίου που είναι 295.475 δολάρια προς τον συσσωρευμένο τόκο ο οποίος είναι 0.488 δολάρια. Στόχος όμως της παρούσας εργασία εκτός από τον υπολογισμό του bp, είναι και η εξέλιξη του στο χρόνο, δηλαδή εάν απομακρύνεται ή εάν έρχεται πιο κοντά με την απελευθέρωση ενός καινούργιου commit. Το συγκεκριμένο commit το οποίο αναλύθηκε στην Εικόνα 20 είχε κατοχυρωθεί στο Github στις 29 Αυγούστου 2016 και ήταν το τελευταίο commit τη στιγμή που έγινε η ανάλυση του έργου. Για να γίνει απολύτως αντιληπτή η διαδικασία διερεύνησης της εξέλιξης του τεχνικού χρέους (με βάση το σημείο ισορροπίας), θα αναλυθεί το συγκεκριμένο πακέτο από το έργο JavaGame σε κάποιες παλαιότερες εκδόσεις. Πιο συγκεκριμένα, θα αναλύσουμε το πακέτο src/com/redomar/game/gfx στην ημερομηνία 6 Αυγούστου 2016. Ο αριθμός commit είναι 2bf76a61f9fee7fa71373be5d008d6c5341998ce και ανάμεσα σε αυτό το commit και στο τελευταίο commit το οποίο αναλύθηκε προηγουμένως υπάρχουν 32 ακόμη commits.



Εικόνα 20: Παράδειγμα εκτέλεσης του εργαλείου σε επίπεδο πακέτου

Υπολογίζοντας λοιπόν το bp για το παλαιότερο commit παρατηρείται ότι το σημείο ισορροπίας υπολογίζεται στα 713.878 commits (Εικόνα 21). Συγκρίνοντας λοιπόν το αποτέλεσμα θα μπορούσε να θεωρηθεί ότι το σημείο ισορροπίας έρχεται πιο κοντά με την απελευθέρωση ενός καινούργιου commit, καθώς στις 6 Αυγούστου 2016 το bp ήταν 713.878 commits, ενώ στις 29 Αυγούστου του ίδιου χρόνου ήταν 606.032 commits. Ωστόσο, δεν θα ήταν σωστό να ερμηνευθεί η εξέλιξη του σημείου ισορροπίας από ένα μόνο έργο και για ένα μόνο συγκεκριμένο πακέτο. Η εξέλιξη θα αναλυθεί στο επόμενο κεφάλαιο με τη χρήση κατάλληλων διαγραμμάτων.



Εικόνα 21: Παράδειγμα εκτέλεσης του εργαλείου σε επίπεδο πακέτου παλαιότερης έκδοσης

Το εργαλείο θα είναι διαθέσιμο για δυο λειτουργικά συστήματα είτε για Windows είτε για Linux. Η λειτουργικότητα θα είναι εξίσου αξιόπιστη και στα δυο συστήματα. Το παράδειγμα εκτέλεσης στην Εικόνα 19 είναι εκτελεσμένο σε Windows, ενώ το παράδειγμα στις Εικόνες 20 και 21, είναι εκτελεσμένο σε Linux και συγκεκριμένα στο πακέτο Ubuntu 16.04.

3.5 Μετρικές Προγράμματος

Αρχική Φόρμα (Starting_form.java)

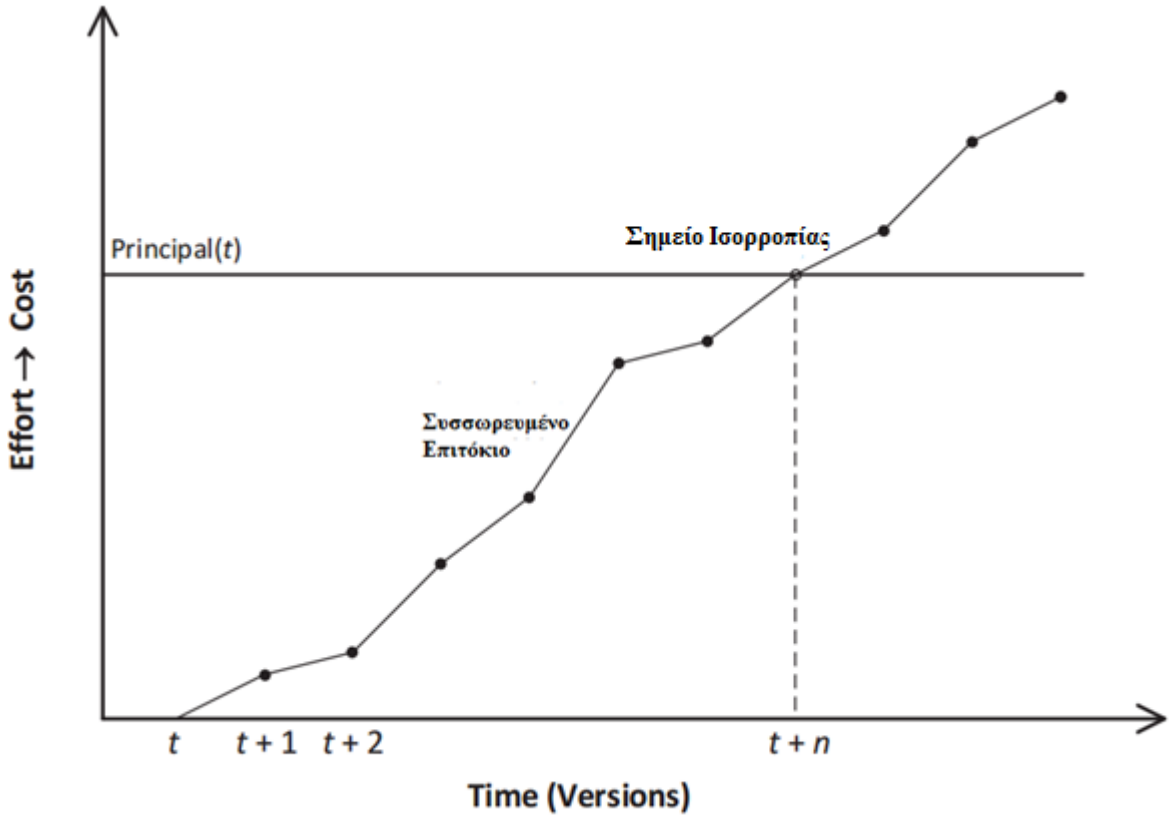
1. Depth of Inheritance Tree (DIT): 1
2. Number of Children Classes (NOCC): 0
3. Message Passing Coupling (MPC): 495
4. Response For a Class (RFC): 174
5. Lack of Cohesion Of Methods (LCOM): 226
6. Data Abstraction Coupling (DAC): 0
7. Weighted Method per Class (WMPC): 2.25
8. Number of Methods (NOM): 23
9. Line of Code (Size1): 216
10. Number of Properties (Size2): 52

Δεύτερη Φόρμα (Second_form.java)

1. Depth of Inheritance Tree (DIT): 1
2. Number of Children Classes (NOCC): 0
3. Message Passing Coupling (MPC): 1267
4. Response For a Class (RFC): 162
5. Lack of Cohesion Of Methods (LCOM): 101
6. Data Abstraction Coupling (DAC): 0
7. Weighted Method per Class (WMPC): 10.21
8. Number of Methods (NOM): 18
9. Line of Code (Size1): 310
10. Number of Properties (Size2): 85

Κεφάλαιο 4ο – Συμπεράσματα και Μελλοντικές Επεκτάσεις

Ένας από τους στόχους της συγκεκριμένης έρευνας είναι να βοηθήσει τους διαχειριστές έργων στη λήψη των αποφάσεων, παρέχοντας μια εκτίμηση του χρονικού σημείου κατά το οποίο ο συσσωρευμένος τόκος από το τεχνικό χρέος θα έχει υπερβεί την αρχική εξοικονόμηση που προήλθε από την απόφαση να μην πληρωθεί το κεφάλαιο. Το διάγραμμα 2 απεικονίζει όλες τις παραπάνω πληροφορίες, με σκοπό να γίνουν περισσότερο κατανοητές. Πιο συγκεκριμένα για το διάγραμμα, υποθέτουμε ότι το Principal(t) είναι το ποσό το οποίο εξοικονομείται από το να μην πληρωθεί το υφιστάμενο τεχνικό χρέος, το κόστος, δηλαδή, το οποίο δεν έχει δαπανηθεί για την ανακατασκευή του κώδικα, έτσι ώστε να μετατραπεί στη βέλτιστη σχεδίαση [3]. Δεδομένου ότι η απόφαση σχετικά με το εάν θα πρέπει να πραγματοποιηθεί η πληρωμή του τεχνικού χρέους πρέπει να ληφθεί σε συγκεκριμένη ώρα, συμβολίζεται το κεφάλαιο ως σταθερή τιμή (οριζόντια γραμμή). Αν και το κεφάλαιο πιθανότατα θα αυξηθεί με την πάροδο του χρόνου, δεδομένου ότι η ποιότητα του λογισμικού συνήθως φθίνει, στο χρονικό σημείο t μπορεί να θεωρηθεί μόνο ως μια σταθερή τιμή. Εφόσον, λοιπόν, δεν έχει αποπληρωθεί το τεχνικό χρέος στη χρονική στιγμή t, μελλοντικές εργασίες συντήρησης απαιτούν επιπρόσθετη προσπάθεια. Αυτή η προσπάθεια, όπως έχει αναφερθεί, είναι ο τόκος και για τα εξελισσόμενα συστήματα (πολλαπλές εκδόσεις) θα συσσωρεύεται όπως απεικονίζεται στο Διάγραμμα 2.



Διάγραμμα 2: Συσσωρευμένος τόκος και προσπάθεια αποπληρωμής.

Ουσιαστικά λοιπόν η προσέγγιση η οποία εξετάζεται θα επιτρέπει στους διαχειριστές έργων λογισμικού να εκτιμούν σε πόσο χρόνο η απόφαση να μην πληρώσουν το τεχνικό χρέος (σε μια συγκεκριμένη χρονική στιγμή t), θα επιφέρει συσσωρευμένο τόκο ο οποίος θα είναι μεγαλύτερος από το κεφάλαιο. Σε περίπτωση που αυτό το χρονικό σημείο είναι πολύ απομακρυσμένο στο μέλλον (n εκδόσεις από τη τωρινή χρονική στιγμή t) τότε πιθανότατα δεν έχει νόημα να επιλυθεί το τεχνικό χρέος στο τωρινό χρονικό σημείο t, δηλαδή εάν για παράδειγμα βρισκόμαστε στην τρίτη έκδοση ενός λογισμικού, και το σημείο ισορροπίας υπολογίζεται στην 81η έκδοση, τότε δεν έχει νόημα να αποπληρωθεί το κεφάλαιο κατά την 3η έκδοση, καθώς αργεί πάρα πολύ το σημείο ισορροπίας. Αντιθέτως, εάν το σημείο ισορροπίας αναμένεται να έρθει στο εγγύς μέλλον και ιδιαίτερα όταν βρίσκεται μέσα στην εξέλιξη του προϊόντος, ο διαχειριστής θα πρέπει να αντιμετωπίσει το τεχνικό χρέος και να αποπληρώσει το κεφάλαιο που προέκυψε από αυτό.

4.1 Αποτελέσματα

Όλες οι εκτελέσεις εφαρμόστηκαν σε προγράμματα ανοιχτού λογισμικού τα οποία προερχόντουσαν από το Github. Τα έργα τα οποία χρησιμοποιήθηκαν είναι:

- JavaGame (315 commits)
- RSTALanguageSupport (359 commits)
- ashot (178 commits)
- reflections (114 commits)
- automon (306 commits)

Έγιναν πάρα πολλές εκτελέσεις για πάρα πολλές κλάσεις και πακέτα σε πάρα πολλά έργα και για διαφορετικά commits. Δεδομένου του μεγάλου όγκου πληροφορίας από τις αναλύσεις δε γίνεται να παρουσιαστούν όλες. Για αυτό το λόγο θα παρουσιαστούν ενδεικτικά οι αναλύσεις από το έργο JavaGame σε επίπεδο κλάσης και σε επίπεδο πακέτου.

Τα ονόματα των κλάσεων είναι:

- AnimatedTile.java
- InputHandler.java
- Game.java
- Mob.java

Ενώ το όνομα από τα πακέτα:

- src/com/redomar/game
- src/com/redomar/game/lib

Αρχικά, θα εξεταστεί η ανάλυση του έργου JavaGame σε επίπεδο πακέτου. Όπως προαναφέρθηκε, τα πακέτα τα οποία πρόκειται να εξεταστούν είναι δύο. Το πρώτο είναι το πακέτο 'src/com/redomar/game'. Ο κύκλος ζωής για το συγκεκριμένο πακέτο ξεκινάει στις 17 Μαΐου 2013 και η ημερομηνία του τελευταίου commit είναι στις 28 Αυγούστου 2016. Στο διάγραμμα ταδε απεικονίζεται η πορεία εξέλιξης του τεχνικού χρέους για το συγκεκριμένο

πακέτο. Όπως μπορεί να γίνει αντιληπτό υπάρχουν οι μπλε κουκίδες πάνω στο διάγραμμα και σημαίνει ότι το commit το οποίο απελευθερώθηκε τη συγκεκριμένη ημερομηνία άλλαξε το σημείο ισορροπίας. Η αλλαγή αυτή προέρχεται είτε από την αλλαγή του interest είτε από την αλλαγή του principal. Τα δεδομένα τα οποία βοήθησαν στη δημιουργία του γραφήματος, απεικονίζονται στον Πίνακα 6.

Date	Breaking Point	Principal ($)	Interest ($)
5/17/2013	26.33	304.637	11.5688
5/18/2013	30.95	358.082	11.5688
5/18/2013	28.31	327.542	11.5688
5/29/2013	44.81	518.417	11.5688
5/31/2013	45.47	526.052	11.5688
6/1/2013	46.13	533.687	11.5688
6/18/2013	51.27	595.53	11.615
6/22/2013	59.16	687.15	11.615
8/18/2013	73.88	858.174	11.615
9/13/2013	71.54	874.208	12.219
10/8/2013	78.4	958.193	12.219
10/30/2013	75.78	911.619	12.0283
2/18/2014	78.07	939.105	12.0283
4/8/2014	83.86	989.496	11.7991
8/13/2014	92.8	1116.24	12.0283
6/13/2016	104.98	1262.83	12.0283
8/28/2016	88.87	1049.81	11.8129
8/28/2016	77.17	911.619	11.8129

Πίνακας 6: Δεδομένα ανάλυσης για το πακέτο ‘src/com/redomar/game’

Όπως φαίνεται από τον παραπάνω πίνακα, ο τόκος από τις 17/5/2013 έως τις 1/6/2013 είναι σταθερός και ίσος με 11.5688, ωστόσο το κεφάλαιο έχει κάποιες αλλαγές και αυτές είναι που προκαλούν τις αλλαγές στο breaking point. Εφόσον δηλαδή ο παρονομαστής (τόκος) είναι σταθερός, το σημείο ισορροπίας θα εξαρτάται πλήρως από τον αριθμητή ο οποίος στη περίπτωση της παρούσας διπλωματικής είναι το κεφάλαιο. Όπως διακρίνεται στο διάγραμμα 3 υπάρχουν 18 μπλε κουκίδες, το οποίο σημαίνει ότι υπάρχουν 18 αλλαγές στο σημείο ισορροπίας για το συγκεκριμένο πακέτο. Οπότε προσεγγιστικά, εφόσον το έργο έχει 315 commits, μπορούμε να υποθέσουμε ότι το bp για το συγκεκριμένο πακέτο μεταβάλετε ανά 315/18= 17,5 commits.

Διάγραμμα 3: Εξέλιξη τεχνικού χρέους για το πακέτο ‘src/com/redomar/game’

Το δεύτερο πακέτο το οποίο θα παρουσιαστεί είναι το ‘src/com/redomar/game/lib’. Το πακέτο αυτό είναι εσωτερικό, δηλαδή βρίσκεται μέσα στο πακέτο οποίο αναλύθηκε προηγουμένως. Αυτό σημαίνει ότι στο προηγούμενο πακέτο υπήρχαν πολύ περισσότερες κλάσεις από το συγκεκριμένο πακέτο και αυτό θα οδηγήσει σε λιγότερες αλλαγές στο breaking point. Πράγματι ενώ στο προηγούμενο υπήρχαν 18 αλλαγές, στο τωρινό υπάρχουν 9. Τα δεδομένα ανάλυσης του πακέτου παρουσιάζονται στον Πίνακα 7.

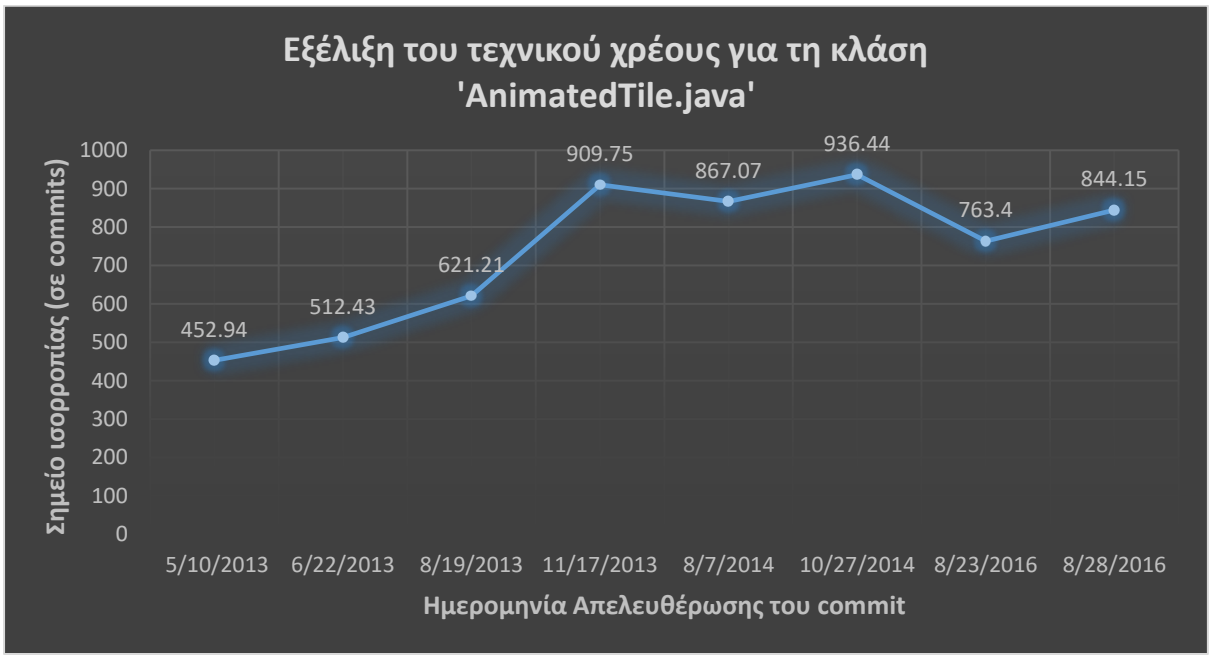
Date	Breaking Point	Principal ($)	Interest ($)
5/10/2013	57.34	39.702	0.692349
5/31/2013	96.33	64.134	0.665739
6/22/2013	292.44	194.693	0.665739
8/19/2013	290.68	198.51	0.682909
10/13/2013	532.13	354.264	0.665739
2/18/2014	507.71	338.994	0.667691
9/7/2014	572.16	359.608	0.628505
8/22/2016	511.42	321.434	0.628505
8/28/2016	198.69	177.895	0.895342

Πίνακας 7: Δεδομένα ανάλυσης για το πακέτο ‘src/com/redomare/game/lib’

Για τη καλύτερη κατανόηση της εξέλιξης του τεχνικού χρέους στο πακέτο αυτό, μπορείτε να δείτε το διάγραμμα 4.

Διάγραμμα 4: Εξέλιξη τεχνικού χρέους για το πακέτο ‘src/com/redomar/game’

Εκτός βέβαια της ανάλυσης των πακέτων, θα παρουσιαστεί και η ανάλυση ενός έργου σε επίπεδο πακέτου. Η πρώτη κλάση η οποία θα αναλυθεί είναι η AnimatedTile.java. Τα δεδομένα τα οποία προκύψαν από τη συγκεκριμένη κλάση ήταν ιδιαίτερα ενδιαφέροντα, καθώς το κεφάλαιο παρέμεινε σταθερό σε όλη τη πορεία του λογισμικού, και το μόνο που μεταβαλλόταν ήταν το interest. Το σημείο ισορροπίας γενικά είχε αύξουσα πορεία, το οποίο σημαίνει ότι με τη πάροδο του χρόνου το bp απομακρύνεται. Στο παρακάτω διάγραμμα απεικονίζεται η εξέλιξη του τεχνικού χρέους για τη κλάση AnimatedTile.java



Διάγραμμα 5: Εξέλιξη τεχνικού χρέους για τη κλάση ‘Animated.java’

Τα δεδομένα ανάλυσης για τη συγκεκριμένη κλάση, φαίνονται στον Πίνακα 8.

Date	Breaking Point	Principal ($)	Interest($)
5/10/2013	452.94	5.3445	0.0117994
6/22/2013	512.43	5.3445	0.0104297
8/19/2013	621.21	5.3445	0.00860336
11/17/2013	909.75	5.3445	0.00587464
8/7/2014	867.07	5.3445	0.0061638
10/27/2014	936.44	5.3445	0.00570723
8/23/2016	763.4	5.3445	0.00700086
8/28/2016	844.15	5.3445	0.00633122

Πίνακας 8: Δεδομένα ανάλυσης για τη κλάση AnimatedTile.java

Η δεύτερη κλάση ονομάζεται InputHandler.java. Τα δεδομένα ανάλυσης για τη συγκεκριμένη κλάση δεν είναι και πολύ ξεκάθαρα καθώς, υπάρχουν αλλαγές τόσο στο κεφάλαιο όσο και στον τόκο. Τα δεδομένα ανάλυσης για τη κλάση αυτή απεικονίζονται στον Πίνακα 9. Συνολικά υπάρχουν 12 αλλαγές στο σημείο ισορροπίας, το οποίο προσεγγιστικά σημαίνει ότι το bp αλλάζει ανά 315/12=26.25 commits.

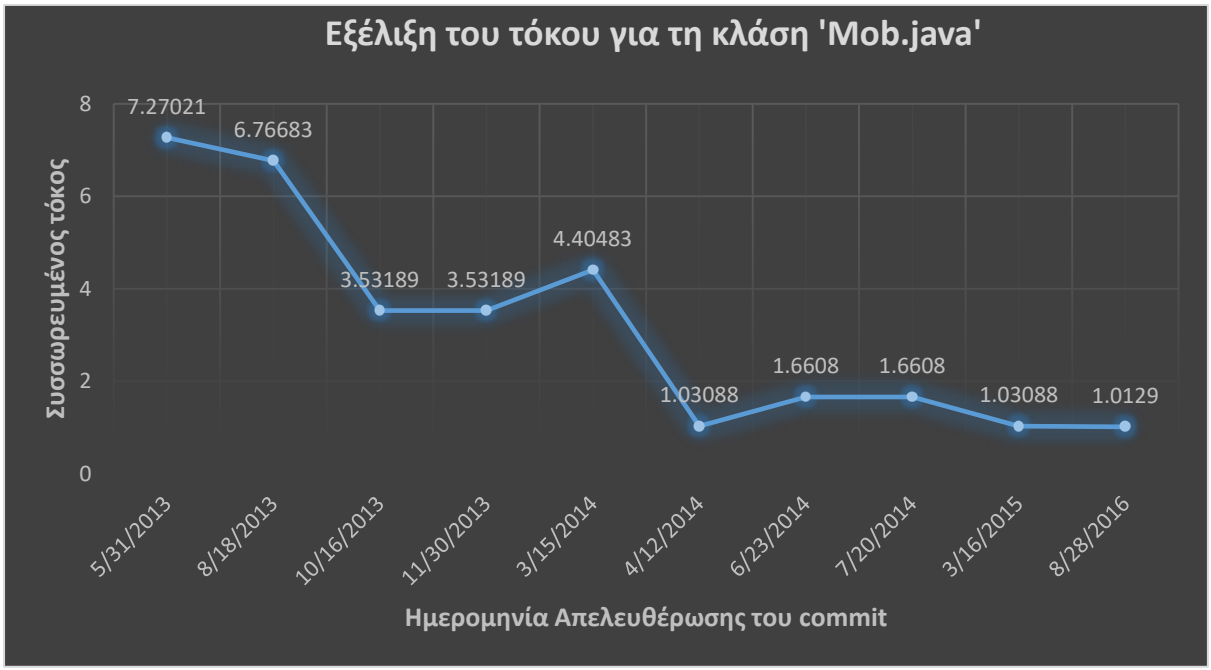
Date	Breaking Point	Principal ($)	Interest($)
6/30/2013	2.963	66.4245	22.4163
9/12/2013	15.12	98.4915	6.50995
9/20/2013	19.27	144.301	7.48495
10/13/2013	20.29	151.936	7.48495
10/18/2013	22.81	174.078	7.62872
12/8/2013	18.77	174.078	9.27065
3/16/2014	51.29	202.327	3.94411
4/21/2014	52.66	213.78	4.05912
8/13/2014	53.79	218.361	4.05912
9/7/2014	144.4	313.798	2.17257
8/28/2016	78.8	158.044	2.00552
8/28/2016	33.09	142.775	4.31353

Πίνακας 9: Δεδομένα ανάλυσης για τη κλάση InputHandler.java

Η τρίτη κλάση η οποία θα αναλυθεί είναι η Mob.java. Πρόκειται για μια εσωτερική κλάση, δηλαδή βρίσκεται σε εσωτερικό πακέτο και συγκεκριμένα στο ‘src/com/redomar/game/entities’. Συνολικά υπάρχουν 10 αλλαγές στο σημείο ισορροπίας της

συγκεκριμένης κλάσης. Για τη κλάση αυτή θα παρουσιαστούν το κεφάλαιο και ο τόκος σε διαγράμματα. Δηλαδή στο διάγραμμα 6 αναφέρεται η εξέλιξη του κεφαλαίου ενώ στο διάγραμμα 7 αναφέρεται η εξέλιξη του επιτοκίου. Τέλος, παρουσιάζεται γραφικά όπως στα προηγούμενα παραδείγματα η ανάλυση εξέλιξης του τεχνικού χρέους.

Διάγραμμα 6: Εξέλιξη κεφαλαίου για τη κλάση ‘Mob.java’



Διάγραμμα 7: Εξέλιξη τόκου για τη κλάση ‘Mob.java’

Με τη βοήθεια των διαγραμμάτων μπορούμε να καταλάβουμε και το σημείο ισορροπίας, δηλαδή, παρατηρούμε ότι ανάμεσα στο πρώτο και στο δεύτερο σημείο το κεφάλαιο είναι σταθερό, ενώ το interest μικραίνει, άρα αναμένεται το σημείο ισορροπίας να είναι πιο μακριά (να μεγαλώσει ο αριθμός). Επίσης, αντίστοιχα στο δεύτερο με το τρίτο

σημείο, στο οποίο υπάρχει απότομη κλίση προς τα κάτω (μικραίνει ο τόκος) και αντίστοιχα υπάρχει αύξηση του κεφαλαίου, αναμένουμε το σημείο ισορροπίας να είναι μεγαλύτερο. Πράγματι, οι ισχυρισμοί αυτοί είναι σωστοί και μας το επιβεβαιώνει το διάγραμμα 8.

Διάγραμμα 8: Εξέλιξη του τεχνικού χρέους για τη κλάση 'Mob.java'

Τέλος, η τελευταία κλάση είναι η πιο κεντρική κλάση του προγράμματος, δηλαδή το όνομα της είναι Game.java και είναι η κλάση στην οποία βασίζεται το έργο JavaGame. Το μονοπάτι για τη κλάση Game.java είναι 'src/com/redomar/game', το πιο εξωτερικό πακέτο του έργου όπως αναφέρθηκε σε προηγούμενη παράγραφο. Στη συγκεκριμένη κλάση υπάρχουν 26 αλλαγές στο σημείο ισορροπίας, το οποίο σημαίνει ότι το bp αλλάζει ανά 12.11 (315/26) commits. Επειδή υπάρχουν πάρα πολλές αλλαγές με πολύ μικρά νούμερα δεν θα παρουσιαστεί το διάγραμμα για την εξέλιξη του τεχνικού χρέους παρά μόνο τα δεδομένα ανάλυσης, τα οποία απεικονίζονται στον Πίνακα 10.

Date	Breaking Point	Principal ($)	Interest($)
5/28/2013	7.3446	260.353	35.4483
5/29/2013	25.8759	458.863	17.7333
6/30/2013	35.8215	635.232	17.7333
8/19/2013	38.4048	681.042	17.7333
9/13/2013	42.3658	751.284	17.7333
9/16/2013	43.1497	747.466	17.3226
10/8/2013	31.7766	781.824	24.6037
10/17/2013	32.6599	797.094	24.4059
10/18/2013	34.2241	835.269	24.4059
12/7/2013	29.8444	728.379	24.4059
2/18/2014	30.3137	739.831	24.4059

2/20/2014	31.0957	758.919	24.4059
3/16/2014	31.2522	762.737	24.4059
3/20/2014	29.2187	713.109	24.4059
3/22/2014	52.4437	713.109	13.5976
4/8/2014	56.0934	762.737	13.5976
6/23/2014	58.6201	797.094	13.5976
7/30/2014	60.9223	828.398	13.5976
8/13/2014	62.1575	845.195	13.5976
9/1/2014	64.2912	874.208	13.5976
9/1/2014	64.2351	873.444	13.5976
9/7/2014	64.572	878.025	13.5976
8/8/2016	64.0105	870.39	13.5976
8/28/2016	59.5185	809.31	13.5976
8/28/2016	49.9169	678.752	13.5976
8/28/2016	50.4784	686.386	13.5976

Πίνακας 10: Δεδομένα ανάλυσης για τη κλάση Game.java

Μετέπειτα, θα παρουσιαστεί η ανάλυση έργου σε επίπεδο έργου. Σε αυτό το επίπεδο δεν υπάρχει η δυνατότητα να ανατρέξουμε για προηγούμενη έκδοση παρά μόνο για τη τελευταία. Εκτελέστηκε το εργαλείο λοιπόν για όλα τα έργα τα οποία υπήρχαν αποθηκευμένα στη βάση και τα αποτελέσματα διακρίνονται στον Πίνακα 11.

Project Name	Date of last commit	Breakin Point	Principal($)	Interest ($)
ashot	8/28/2016	148.676	1061.27	7.13812
automon	8/28/2016	1461.02	4807	3.29017
JavaGame	8/28/2016	95.0899	6662.3	70.0632
reflections	8/28/2016	177.052	5263.57	29.7289
RSTALanguageSupport	8/28/2016	387.814	34291.8	88.4234

Πίνακας 11: Δεδομένα ανάλυσης σε επίπεδο έργου

4.2 Παρατηρήσεις και σχόλια

Στο συγκεκριμένο κεφάλαιο θα παρουσιάσουμε κάποια συμπεράσματα και παρατηρήσεις οι οποίες προήλθαν από την στατική ανάλυση των παραπάνω στοιχείων. Από τα δεδομένα ανάλυσης των παραπάνω έργων παρατηρούμε πως υπήρχαν συνολικά 83 αλλαγές στο σημείο ισορροπίας. Παρατηρούμε ότι από τις συνολικές αλλαγές που έγιναν:

- 61 ήταν θετικές
- 22 ήταν αρνητικές

Ας εστιάσουμε αρχικά στις αλλαγές οι οποίες κατάφεραν να απομακρύνουν το bp. Έχουμε αναφέρει ότι το σημείο ισορροπίας εξαρτάται από το κεφάλαιο και από τον τόκο, οπότε θα δούμε τις αλλαγές οι οποίες έγιναν σε αυτές τις δυο μεταβλητές όταν είχαμε θετική αλλαγή. Από τις 61 λοιπόν αλλαγές, οι 25 είχαν μείωση στον συσσωρευμένο τόκο, 30 είχαν ίδιο τόκο, ενώ οι υπόλοιπες 6 είχαν αύξηση. Στη περίπτωση που έχουμε μείωση στον τόκο και το κεφάλαιο να έχει μείνει ίδιο δεν μας επηρεάζει καθώς θα αυξηθεί το σημείο ισορροπίας (μειώνουμε το παρονομαστή). Αντιθέτως, στη περίπτωση που ο τόκος παραμένει σταθερός ή έχει αυξηθεί πρέπει οπωσδήποτε να έχει αυξηθεί και το κεφάλαιο . Οπότε αναμένουμε να υπάρχουν τουλάχιστον 36 αλλαγές (30+6) στο κεφάλαιο, με θετική φορά φυσικά. Πράγματι εάν παρατηρήσουμε τους πίνακες θα δούμε πως υπάρχουν αυξήσεις στο κεφάλαιο σε 51 περιπτώσεις, ενώ σε 8 περιπτώσεις παρέμεινε σταθερό και μόλις σε 2 περιπτώσεις είχε μειωθεί.

Ας επισημάνουμε επίσης τις περιπτώσεις στις οποίες το σημείο ισορροπίας δεν απομακρύνθηκε αλλά αντίθετα ήρθε πιο κοντά. Από τις 22 αλλαγές οι οποίες οδήγησαν σε αρνητικό αποτέλεσμα, οι 10 είχαν αύξηση στον τόκο, οι 9 είχανε παραμείνει σταθερές, ενώ οι 3 είχανε μείωση. Από τη μεριά του κεφαλαίου παρατηρήθηκε ότι σε 15 περιπτώσεις το κεφάλαιο είχε μειωθεί, σε 4 είχε παραμείνει ίδιο και σε μόλις 3 είχε αυξηθεί.

Οπότε συνοπτικά μπορούμε να εκφράσουμε κάποιες συχνότητες με βάση τα αποτελέσματα τα οποία παρατηρήσαμε παραπάνω:

- Η αλλαγή που παρατηρήθηκε να είναι θετική (να απομακρύνει το bp): 61/83 ή 73%.
- Η αλλαγή που παρατηρήθηκε να είναι αρνητική (να φέρνει πιο κοντά το bp): 22/83 ή 26%.
- Όταν η αλλαγή είναι θετική να υπάρχει μείωση του επιτοκίου: 25/61 ή 40.9%.
- Όταν η αλλαγή είναι θετική να υπάρχει μείωση του κεφαλαίου: 2/61 ή 3.2 %.
- Όταν η αλλαγή είναι θετική ο τόκος να παραμένει ίδιο: 30/61 ή 49,18%.
- Όταν η αλλαγή είναι θετική το κεφάλαιο να παραμένει ίδιο: 8/61 ή 13,11%.
- Όταν η αλλαγή είναι θετική ο τόκος να έχει αυξηθεί: 6/61 ή 9,8%.
- Όταν η αλλαγή είναι θετική το κεφάλαιο να έχει αυξηθεί: 51/61 ή 83,61%.
- Όταν η αλλαγή είναι αρνητική να υπάρχει μείωση του επιτοκίου: 3/22 ή 13,6%.
- Όταν η αλλαγή είναι αρνητική να υπάρχει μείωση του κεφαλαίου: 15/22 ή 68,18 %.
- Όταν η αλλαγή είναι αρνητική ο τόκος να παραμένει ίδιο: 9/22 ή 40,9%.
- Όταν η αλλαγή είναι αρνητική το κεφάλαιο να παραμένει ίδιο: 4/22 ή 18,18%.
- Όταν η αλλαγή είναι αρνητική ο τόκος να έχει αυξηθεί: 10/22 ή 45,45%.
- Όταν η αλλαγή είναι αρνητική το κεφάλαιο να έχει αυξηθεί: 3/22 ή 13,6%.

4.3 Μελλοντικές Επεκτάσεις

Στη συγκεκριμένη ενότητα παρουσιάζονται κάποιες επεκτάσεις που θα βελτίωναν το εργαλείο διευκολύνοντας την ευχρηστία και την αποδοτικότητα του.

Αρχικά έχουμε αναφέρει ότι το εργαλείο αντί για απελευθερωμένες εκδόσεις, δέχεται commits από το Github. Μια ενδιαφέρουσα μελλοντική επέκταση θα ήταν να προστεθεί μια λειτουργία στο εργαλείο, η οποία θα ομαδοποιεί τα commits ενός έργου σε απελευθερωμένες εκδόσεις, δηλαδή εάν για παράδειγμα ένα έργο έχει 100 commits στο Github και έχει 6

απελευθερωμένες εκδόσεις, η πρώτη έκδοση ενδεχομένως να είναι τα πρώτα 30 commits, η δεύτερη τα επόμενα 15 και ου το καθ' εξής. Το πρόγραμμα θα μπορεί να επεξεργάζεται τα στοιχεία από τα commits και πλέον να εξάγει το αποτέλεσμα όχι σε αριθμό commits αλλά σε αριθμό εκδόσεων.

Επίσης μια πολύ ενδιαφέρουσα επέκταση για το εργαλείο θα ήταν η αυτόματη δημιουργία του jar αρχείου. Πιο συγκεκριμένα θα μπορούσε να δημιουργηθεί ένα script, το οποίο θα δημιουργεί αυτόματα το jar αρχείο του έργου απευθείας από το git repository του. Φυσικά η λειτουργία αυτή θα είναι πολύ σημαντική καθώς ο χρήστης δεν θα πρέπει να κατεβάζει αυτό το αρχείο jar και να συμπληρώνει το όνομα του jar το οποίο πρόκειται να εκτελεστεί. Πλέον όλα θα γίνονται αυτόματα και από τρεις λειτουργίες που υπάρχουν στην αρχική φόρμα θα μειωθούν σε δυο. Θα υπάρχει πλέον ένα κουμπί για την ανάλυση των έργων και όχι δυο διαφορετικά κουμπιά. Με αυτό τον τρόπο η ευχρηστία του εργαλείου θα αυξηθεί και απλά με το git repository του κάθε έργου θα επιτυγχάνεται η ανάλυση του.

Επιπρόσθετα, μια μελλοντική επέκταση η οποία θα είχε ενδιαφέρον, είναι η ανάλυση έργων από βιομηχανίες (δηλαδή τα έργα τα οποία δεν είναι ανοιχτού κώδικα). Κάνοντας την ανάλυση των έργων αυτών θα μπορούσαμε να συγκρίνουμε τα δυο είδη έργων ως προς το τεχνικό χρέος του κάθε ενός και να εξάγουμε τα κατάλληλα συμπεράσματα.

Τελευταία λειτουργία η οποία θα μπορούσε να ενσωματωθεί στο ίδιο υπάρχον εργαλείο, είναι η «Παρακολούθηση του σημείου ισορροπίας». Με τη συγκεκριμένη λειτουργία, ο χρήστης θα έχει τη δυνατότητα να εμφανίζει για κάποια κλάση ή πακέτο το σημείο ισορροπίας για όλες τις απελευθερωμένες εκδόσεις. Πιο συγκεκριμένα θα εμφανίζεται ένα διάγραμμα (παρόμοιο με τα διαγράμματα στην ενότητα 4.2) το οποίο θα εμφανίζει το σημείο ισορροπίας για όλες τις εκδόσεις του έργου. Με αυτό τον τρόπο ο διαχειριστής θα έχει μια καλύτερη εικόνα για το έργο του βασισμένη φυσικά πάνω σε κάποια στοιχεία.

Βιβλιογραφία – Αναφορές

[1] Alexander Chatzigeorgiou, Ap. Ampatzoglou, A. Ampatzoglou, T. Amanatidis: Estimating the Breaking Point for Technical Debt, IEEE, Bremen, Germany, 2015

[2] A. Ampatzoflou, Ap. Ampatzoglou, P. Avgeriou, A. Chatzigeorgiou: A Financial Approach for Managing Interest in Technical Debt

[3] Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P.: The financial aspect of managing technical debt: a systematic literature review. Inf. Softw. Technol. 64, 52 –73 (2015). Elsevier

[4] H. van Vliet, Software Engineering: Principal and Practice, John Wiley & Sons, 2008

[5] D.L Parnas, Software Aging, 6th International Conference on Software Engineering (ICSE'94), 16-21 May 1994, IEEE Computer Sociey, Sorrento, Italy, 1994, pp. 279-287.

[6] L. Bass, P.Clements, R. Kazman, Software Architecture in Practice, second ed., Addison-Wesley Professional,2003

[7] W. Cunningham, The WyCash portfolio management system, in: 7th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '92), Vancouver, Canada, 5–10 October 1992, pp. 29–30.

[8] P. Kruchten, R.L. Nord, I. Ozkaya, Technical debt: from metaphor to theory and practice, Software, IEEE Computer Society 29 (6) (2012) 18–21.

[9] H. Erdogmus, Comparative evaluation of software development strategies based on Net Present Value, in: 1st Workshop on Economics Driven Software Engineering Research (EDSER' 99), Los Angeles, USA, 17 May 1999.

[10] N. Zazworka, M. Shaw, F. Shull, C. Seaman, Investigating the impact of design debt on software quality, in: 2nd Workshop on Managing Technical Debt (MTD '11), ACM, Hawaii, USA, 21–28 May 2011, pp. 17–23.

[11] Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220 (2015). Elsevier

[12] Ampatzoglou, A., Ampatzoglou, A., Avgeriou, P., Chatzigeorgiou, A.: Establishing a framework for managing interest in technical debt. In: 5th International Symposium on Business Modeling and Software Design, BMSD 2015, Milan, Italy (2015)

[13] Mishkin, F., Eakins, S.: Financial Markets and Institutions, 7th edn. Prentice Hall, Upper Saddle River (2012)

[14] P. Deitel, H. Deitel : Java Προγραμματισμός, 8η έκδοση

[15] L. James: Beginning Perl

[16] Modern Perl, Fourth edition, by chromatic

[17] G. Ann Campbell, P. Papapetrou: SonarQube in Action

[18] https://www.sonarqube.org/ (Official site of sonar)

[19] https://blog.sonarsource.com/ (Official forum of sonar)

[20] Ramakrishnan, Gehrke: Συστήματα Διαχείρισης Βάσεων Δεδομένων, 3η έκδοση

[21] https://en.wikipedia.org/wiki/Git

[22] https://en.wikipedia.org/wiki/GitHub

[23] https://netbeans.org/features/

[24] https://en.wikipedia.org/wiki/Notepad%2B%2B

[25] https://en.wikipedia.org/wiki/MySQL_Workbench

[26] https://en.wikipedia.org/wiki/Sublime_Text

[27] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the size, cost, and types of Technical Debt," 3rd International Workshop on Managing Technical Debt (MTD '12), Zurich, Switzerland, 2012, pp. 49 – 53

[28] N. Tsantalis and A. Chatzigeorgiou, "Identification of Move Method Refactoring Opportunities," IEEE Trans. Softw. Eng., vol. 35, no. 3, pp. 347–367, May 2009

[29] U.S. Bureau of Labor Statistics, "National Occupational Employment and Wage Estimates," United States, Mar. 2015.

[30] L. Prechelt, "An Empirical Comparison of Seven Programming Languages," Computer, vol. 33, no. 10, pp. 23–29, Oct. 2000.

[31] Nord, R., Ozkaya, I., Kruchten, P., Gonzalez-Rojas, M.: In Search of a metric for managing architectural technical debt. In: 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA). IEEE Computer Society (2012)

[32] Siebra, C.S., Tonin, G.S., Silva, F.Q., Oliveira, R.G., Junior, A.L., Miranda, R.C., Santos, A. L.: Managing technical debt in practice: an industrial report. In: 6th International Symposium on Empirical Software Engineering and Measurement (ESEM). ACM (2012)

[33] Tom, E., Aurum, A., Vidgen, R.: An exploration of technical debt. J. Syst. Softw. 86(6), 1498–1516 (2013). Elsevier