ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μεθοδολογία ποσοτικοποίησης σύζευξης και συνοχής σε επίπεδο πακέτου με στόχο τη βελτίωση της σχεδιαστικής ποιότητας αντικειμενοστρεφών συστημάτων

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΑΓΓΕΛΙΚΗ - ΑΓΑΘΗ ΤΣΙΝΤΖΗΡΑ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΡΙΑ : ΣΤΑΜΑΤΙΑ ΜΠΙΜΠΗ

ΚΟΖΑΝΗ 2018

DIPLOMA THESIS

Quantification methodology for coupling and cohesion at package level to improve the design quality of object-oriented systems

---

UNIVERSITY OF WESTERN MACEDONIA
FACULTY OF ENGINEERING
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS ENGINEERING

ANGELIKI-AGATHI TSINTZIRA

SUPERVISOR: STAMATIA BIBI

KOZANI 2018

Περίληψη

Ο σκοπός της συγκεκριμένης διπλωματικής εργασίας ήταν η ανάπτυξη μεθοδολογίας ποσοτικοποίησης συνοχής και σύζευξης σε επίπεδο πακέτου προκειμένου να προταθούν λύσεις για τη βελτίωση της σχεδιαστικής ποιότητας αντικειμενοστρεφών συστημάτων.

Η συνοχή των κλάσεων σε ένα πακέτο είναι επιθυμητή από τη στιγμή που προωθεί την ενθυλάκωση. Αποδίδει πόσο ισχυρές είναι οι λειτουργίες σε κάθε πακέτο του προγράμματος. Καλά δομημένα πακέτα, οδηγούν σε ιδιαίτερα συνεκτικά προγράμματα. Η ασθενής ή η απουσία συνοχής υποδηλώνει ότι το πακέτο πιθανώς να πρέπει να διασπαστεί σε δύο ή περισσότερα πακέτα ή ότι κάποιες κλάσεις πρέπει να μετακινηθούν σε κάποιο άλλο πακέτο στο οποίο οι εξαρτήσεις είναι πιο ισχυρές.

Η σύζευξη μετρά πόσο εξαρτάται η κάθε ενότητα από τις άλλες ενότητες του προγράμματος. Αλληλεπιδράσεις μεταξύ κλάσεων συμβαίνουν επειδή υπάρχει σύζευξη. Τα χαλαρά συνδεδεμένα προγράμματα έχουν υψηλή ευελιξία και δυνατότητα επέκτασης. Αυξημένα επίπεδα σύζευξης είναι ανεπιθύμητα σε συστήματα αποτελούμενα από υπό-μονάδες και αποτελούν τροχοπέδη στην επαναχρησιμοποίηση.

Η έλλειψη συνοχής και σύζευξης αυξάνει την πολυπλοκότητα και την πιθανότητα εμφάνισης λαθών κατά τα στάδια ανάπτυξης, συντήρησης και επέκτασης ενός λογισμικού. Η χαλαρή σύζευξη και η ισχυρή συνοχή παρέχουν το καλύτερο λογισμικό.

Η μεθοδολογία που αναπτύχθηκε, υλοποιήθηκε σε γλώσσα προγραμματισμού Java. Δέχεται ως είσοδο ένα πρόγραμμα λογισμικού, ακολουθεί συντακτική ανάλυση των σχέσεων μεταξύ κλάσεων και πακέτων και υπολογίζει μετρικές συνοχής και σύζευξης. Στη συνέχεια, δημιουργεί ομάδες κλάσεων n πλήθους (n = 1,2,3,..) με σκοπό τη μεταφοράς τους σε κάποιο άλλο πακέτο όπου οι παραπάνω μετρικές βελτιώνονται. Ο στόχος είναι να εξαχθούν συμπεράσματα σχετικά με το πλήθος των ομάδων κλάσεων το οποίο επιφέρει το βέλτιστο αποτέλεσμα σε επίπεδο συνοχής και σύζευξης.

Η εγκυρότητα της μεθοδολογίας ελέγχθηκε σε προγράμματα ελεύθερου λογισμικού γραμμένα σε java και υλοποιημένα με πρότυπα αντικειμενοστρεφούς προγραμματισμού.

Λέξεις κλειδιά : Τεχνολογία λογισμικού, Μετακίνηση κλάσεων, Σύζευξη μεταξύ πακέτων, Συνοχή μεταξύ πακέτων, Ποιότητα λογισμικού, Αντικειμενοστρεφής σχεδίαση, Τμηματοποίηση

Abstract

The purpose of this diploma thesis was to develop a methodology for quantifying cohesion and coupling at package level in order to propose solutions to improve the design quality of object-oriented systems.

The consistency of classes in a package is desirable once it promotes encapsulation. It shows how powerful the functions are in each program package. Well-structured packages lead to highly coherent programs. The weakness or lack of cohesion suggest that the package may have to be split into two or more packages or that some classes have to be moved to another package where the dependencies are more powerful.

Coupling measures how much each module depends on the other sections of the program. Interactions between classes occur because there is coupling. Loosely linked programs have high flexibility and extendability. Increased coupling levels are undesirable in systems composed of sub-units and are a brake on reuse.

Lack of cohesion and coupling increases the complexity and likelihood of errors occurring during the development, maintenance, and expansion stages of software. Loose coupling and strong cohesion provide the best software.

The methodology developed was implemented in Java. It accepts as input a software program, follows a structural analysis of relations between classes and packages and calculates cohesion and coupling metrics. Then, it creates groups of n classes (n = 1, 2, 3, ...) for the purpose of transferring them to another package where the above mentioned metrics are improved. The goal is to draw conclusions about the number of classes that bring the optimal result at cohesion and coupling level.

The validity of the methodology was tested in java-based open source programs implemented with object-oriented programming.

Keywords: Software Engineering, Move Classes, Package Coupling, Package Cohesion, Software Quality, Object Oriented Design, Modularity

Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Μεθοδολογία ποσοτικοποίησης σύζευξης και συνοχής σε επίπεδο πακέτου με στόχο τη βελτίωση της σχεδιαστικής ποιότητας αντικειμενοστρεφών συστημάτων" 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. Μπίμπη Σταματία, 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.

Ευχαριστίες

Αρχικά, θέλω να ευχαριστήσω τον Επισκέπτη Ερευνητή του Πανεπιστημίου του Groningen, κ. Αμπατζόγλου Απόστολο, για την καθοδήγηση, την εμπιστοσύνη και την πολύτιμη βοήθεια που μου παρέχει κατά το διάστημα εκπόνησης της διπλωματικής εργασίας. Επίσης, θέλω να τον ευχαριστήσω για την ευκαιρία που μου έδωσε να μελετήσω τον τομέα της μηχανικής λογισμικού και να πάρω μία πρώτη γνώση του ερευνητικού τομέα.

Στη συνέχεια, θέλω να ευχαριστήσω την επιβλέπουσα καθηγήτρια της διπλωματικής εργασίας, την Λέκτορα κ. Μπίμπη Σταματία, η οποία μου εμπιστεύτηκε την ανάθεση της εργασίας αυτής. Ακολούθως, ένα μεγάλο ευχαριστώ αξίζει σε όλους τους ανθρώπους τους οποίους γνώρισα το διάστημα εκπόνησης της διπλωματικής μου εργασίας όπου η βοήθειά τους και η θετική τους διάθεση έπαιξε σημαντικό ρόλο, στον καθηγητή του Τμήματος Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας κ. Χατζηγεωργίου Αλέξανδρο, την υποψήφια διδάκτορα Αρβανίτου Ελβίρα-Μαρία και τον υποψήφιο διδάκτορα Δίγκα Γεώργιο.

Κατόπιν, θέλω να ευχαριστήσω τους γονείς μου Σπύρο και Βίκυ για όλα όσα μου προσφέρουν εδώ και τόσα χρόνια. Την αγαπημένη μου φίλη, Μίνα, η οποία ήταν δίπλα μου με υπομονή, υποστήριξη, συμπαράσταση καθ' όλη τη διάρκεια των προπτυχιακών μου σπουδών. Όλους όσους γνώρισα στα πέντε χρόνια της φοιτητικής μου ζωής στην Κοζάνη για τις στιγμές χαράς, γέλιου, συνεργασίας, ανταλλαγής απόψεων και υποστήριξης.

Τέλος, θέλω να ευχαριστήσω όλους τους δύσκολους ανθρώπους που είχα την τύχη να γνωρίσω σε τόση μικρή ηλικία. Αποτελέσατε κίνητρο για να βελτιώνομαι και να εξελίσσομαι κάθε μέρα σε όλους τους τομείς της ζωής μου.

"It had long since come to my attention that people of accomplishment rarely sat back and let things happen to them. They went out and happened to things."
— Leonardo da Vinci

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Εικόνων

1. Εισαγωγή

"Software is the magic thing whose importance only goes up over time."
— Bill Gates

Η εξάρτηση των περισσότερων χωρών από σύνθετα συστήματα που βασίζονται σε υπολογιστές εν έτη 2018 είναι πιο έντονη από ποτέ. Οι εθνικές υποδομές και υπηρεσίες στηρίζονται σε υπολογιστικά συστήματα ενώ, τα περισσότερα ηλεκτρικά προϊόντα περιλαμβάνουν έναν υπολογιστή και κάποιο λογισμικό ελέγχου. Η βιομηχανική παραγωγή και διανομή έχουν αυτοματοποιηθεί, όπως άλλωστε και το οικονομικό σύστημα. Κατά συνέπεια, η οικονομικά αποτελεσματική ανάπτυξη και συντήρηση του λογισμικού είναι ουσιώδης για τη λειτουργία των εθνικών οικονομιών και της διεθνούς οικονομίας.

Η καθιέρωση των σύγχρονων Τεχνολογιών Πληροφορικής και Επικοινωνιών (ΤΠΕ) στον χώρο των επιχειρήσεων τις τελευταίες δεκαετίες, παρά τις μεγάλες αμφισβητήσεις και αντιδράσεις τις οποίες συνάντησε, έχει δημιουργήσει σημαντικότατες ανακατατάξεις σε όλα τα επίπεδα οργάνωσης και λειτουργίας τους, με τελικό αποτέλεσμα την αύξηση της αποδοτικότητας και παραγωγικότητας των επιχειρήσεων. Το περιβάλλον στο οποίο κινείται και εργάζεται ο σύγχρονος άνθρωπος έχει πλέον ως βάση τις εφαρμογές της Πληροφορικής. Η ανάπτυξη του συγκεκριμένου κλάδου, σε συνδυασμό με την αξιοποίηση των νέων τεχνολογιών της ηλεκτρονικής, της επικοινωνίας και του διαδικτύου είναι κυριολεκτικά αλματώδης. Δεν είναι λίγοι εκείνοι που πιστεύουν ότι οι περισσότερο επιτυχημένες επιχειρήσεις σήμερα είναι εκείνες που χρησιμοποιούν την πληροφορία πιο αποτελεσματικά από τις υπόλοιπες στον αντίστοιχο κλάδο.

Τα εθνικά συστήματα όπως το σύστημα υγείας και το σύστημα καταγραφής πολιτών περιέχουν μεγάλο όγκο πληροφοριών. Τα συστήματα αυτά πρέπει να είναι σε θέση να επεξεργαστούν όλο αυτό το φόρτο πληροφοριών καθώς επίσης και να δεχτούν τροποποιήσεις και νέες λειτουργίες οποιαδήποτε στιγμή. Στο σημείο αυτό, φαίνεται η αναγκαιότητα της τεχνολογίας λογισμικού. Τεχνολογία λογισμικού ορίζεται η περιοχή εκείνη της επιστήμης της πληροφορικής η οποία ασχολείται με την εύρεση και θεμελίωση μεθόδων για την περιγραφή, την κατασκευή και τη συντήρηση του λογισμικό. Επιθυμητά χαρακτηριστικά του λογισμικού και της διαδικασίας κατασκευής του είναι η ποιότητα, η μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα και το ελάχιστο δυνατό κόστος παραγωγής και

συντήρησης. Το λογισμικό είναι αφηρημένο και άυλο. Δεν περιορίζεται από υλικά, ούτε διέπεται από φυσικούς νόμους ή διαδικασίες. Το γεγονός αυτό κατά κάποιον τρόπο απλουστεύει την τεχνολογία λογισμικού, αφού δεν υφίστανται φυσικοί περιορισμοί για τις δυνατότητες του λογισμικού. Όμως, αυτή η έλλειψη φυσικών περιορισμών σημαίνει ότι το λογισμικό μπορεί εύκολα να γίνει εξαιρετικά πολύπλοκο και επομένως δυσνόητο.

Συνήθεις αιτίες για τροποποιήσεις στο λογισμικό είναι η διόρθωση σφαλμάτων, η βελτιστοποίηση της απόδοσης, η αυτοματοποίηση της εκτέλεσης νέων εργασιών και η ενσωμάτωση μεταβολών που οφείλονται σε αλλαγές που συμβαίνουν στον πραγματικό κόσμο. Η πραγματοποίηση μεταβολών/διορθώσεων στις εφαρμογές λογισμικού αναφέρεται με τον όρο «συντήρηση λογισμικού» (software maintenance). Όλες οι φάσεις από τις οποίες διέρχεται το λογισμικό αναφέρονται ως «κύκλος ζωής λογισμικού» (software life cycle). Γίνεται σαφές ότι, η Τεχνολογία Λογισμικού δεν ασχολείται μόνο με την κατασκευή, αλλά με ολόκληρο τον κύκλο ζωής του λογισμικού. Χρονικά, πρόκειται για το διάστημα από τη σύλληψη της ιδέας της κατασκευής μιας εφαρμογής λογισμικού μέχρι την απόσυρση αυτής από τη χρήση.

Επομένως, γίνεται αντιληπτή η σημασία της ποιότητας λογισμικού για την εύρυθμη λειτουργία και συντήρησή του. Αντικείμενο αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μεθοδολογίας ποσοτικοποίησης συνοχής και σύζευξης σε επίπεδο πακέτου προκειμένου να προταθούν λύσεις για τη βελτίωση της σχεδιαστικής ποιότητας αντικειμενοστρεφών συστημάτων. Η χαλαρή σύζευξη και η ισχυρή συνοχή παρέχουν το καλύτερο λογισμικό. Πιο συγκεκριμένα, αναπτύχθηκε λογισμικό το οποίο υπολογίζει και προτείνει πιθανές μετακινήσεις κλάσεων μεταξύ πακέτων με σκοπό τη βελτίωση μετρικών συνοχής και σύζευξης οι οποίες έχουν ως αποτέλεσμα ένα εύκολα επεκτάσιμο, επαναχρησιμοποιήσιμο και ποιοτικό λογισμικό.

1.1 Κίνητρο Διεξαγωγής Εργασίας

Το λογισμικό είναι μέρος της καθημερινότητάς μας και αποτελεί αναπόσπαστο κομμάτι σε κάθε τομέα της ζωής μας. Λόγω της έκτασης και της χρησιμότητάς του είναι πολύ σημαντικό η ποιότητά του συνεχώς να αυξάνεται και να βελτιώνεται. Το κύριο κίνητρο που με ώθησε να ασχοληθώ με αυτό τον τομέα ήταν να προσθέσω και εγώ ένα μικρό λιθαράκι με σκοπό τη βελτίωση του λογισμικού. Σαν μηχανικός, είναι στο αίμα μου να

ψάχνω λύσεις για τα πάντα οι οποίες αρχικά να λύνουν το πρόβλημα και στη συνέχεια να το λύνουν με τον καλύτερο δυνατό τρόπο.

Ο τομέας της τεχνολογίας λογισμικού είναι ένας τομέας που δεν είχα την ευκαιρία να διδαχτώ τόσο εκτενές στο πρόγραμμα σπουδών μου οπότε, το θεώρησα ως μία πρόκληση να ανταπεξέλθω σε έναν τομέα όχι τόσο γνωστό προς εμένα. Ασφαλώς, χωρίς την εξαιρετική βοήθεια που δέχτηκα από όλους τους ανθρώπους με τους οποίους συνεργάστηκα, τίποτα από όλα όσα αναφέρονται δεν θα ήταν εφικτά σε τόσο μικρό χρονικό διάστημα.

1.2 Συνεισφορά στην Επιστημονική Κοινότητα

Ο τομέας της τεχνολογίας ή μηχανικής λογισμικού έκανε την εμφάνισή του το 1960. Από τότε πολλές τεχνικές και μέθοδοι έχουν αναπτυχθεί οι οποίες οδηγούν στη βελτίωση της εσωτερικής δομής του λογισμικού. Η παρούσα διπλωματική εργασία προτείνει μια μεθοδολογία για τη βελτίωση της τμηματικότητας του λογισμικού. Μέχρι στιγμής, η τμηματικότητα έχει μελετηθεί σε επίπεδο μεθόδου και κλάσης, δηλαδή έχει γίνει εκτενής έρευνα για τη μετακίνηση ή εξαγωγή μεθόδων καθώς και για την εξαγωγή κλάσεων. Σε επίπεδο πακέτου λογισμικού, η μέχρι τώρα σχετικά μικρή βιβλιογραφία, αφορά μεθοδολογίες που πρότειναν σχετικές αναδομήσεις οι οποίες μετακινούσαν μια κλάση από ένα πακέτο λογισμικού σε ένα άλλο. Το καινοτόμο στοιχείο αυτής της διπλωματικής, το οποίο προστίθεται σε μία επιστήμη αρκετά ώριμη, είναι η μετακίνηση περισσότερων από μία κλάσεων από το ένα πακέτο στο άλλο. Η προτεινόμενη μεθοδολογία στοχεύει στην ομαδοποίηση κλάσεων με υψηλή σύζευξη με σκοπό την μετακίνηση τους σε κάποιο πιο συνεκτικό πακέτο. Ο σκοπός ήταν να εξαχθούν αποτελέσματα και συμπεράσματα σχετικά με το αν η ομαδοποίηση κλάσεων για μετακίνηση οδηγεί σε καλύτερα αποτελέσματα από την μετακίνηση μίας κλάσης καθώς και να εξαχθεί, αν είναι δυνατόν, κάποιος καθολικός κανόνας για το πλήθος των κλάσεων προς μετακίνηση που επιφέρει τα καλύτερα αποτελέσματα.

1.3 Σύνοψη Διπλωματικής Εργασίας

Το υπόλοιπο κείμενο αποτελείται από εννιά κεφάλια τα οποία έχουν σκοπό να παρουσιάσουν στον αναγνώστη το αντικείμενο, τη μεθοδολογία, την υλοποίηση, τη διαδικασία έρευνας, τα αποτελέσματα και τα συμπεράσματα που εξήχθησαν. Πιο συγκεκριμένα στο κεφάλαιο δύο έγινε ανάλυση των βασικών αρχών της σχεδίασης

λογισμικού, των βασικών αρχών σχεδίασης σε επίπεδο πακέτου, το οποίο είναι και το αντικείμενο που ασχολείται η εργασία αυτή, και της αρχιτεκτονικής των λογισμικών τα οποία αναλύθηκαν.

Στο κεφάλαιο τρία παρουσιάζεται το θεωρητικό υπόβαθρο, δηλαδή οι μετρικές ποιότητας λογισμικού, οι μετρικές σύζευξης και συνοχής, οι έννοιες της αναδόμησης και των κακών οσμών, οι μετρικές που χρησιμοποιούνται από τη μεθοδολογία που αναπτύχθηκε και τέλος, γίνεται αναφορά σε παρόμοιες εργασίες που έχουν δημοσιευθεί.

Το κεφάλαιο τέσσερα είναι αφιερωμένο στην παρουσίαση, ανάλυση και εξήγηση της μεθοδολογίας και του αλγορίθμου που αναπτύχθηκαν για την επίλυση του προβλήματος, μέσω επεξηγήσεων και παραδειγμάτων.

Το κεφάλαιο πέντε προστέθηκε ώστε να δώσει μία καλύτερη οπτική και ανάλυση του τρόπου υλοποίησης και εκτέλεσης του λογισμικού. Στο κεφάλαιο αυτό, αναφέρονται πιο τεχνικά ζητήματα και παρουσιάζεται ο τρόπος εκτέλεσης του λογισμικού, ο ψευδοκώδικας, το διάγραμμα κλάσεων και τα σημαντικά σημεία κώδικα. Επίσης παρουσιάζονται τα αποτελέσματα και δεδομένα που προκύπτουν από την εκτέλεση του λογισμικού.

Το επόμενο κεφάλαιο, έκτο, αναλύει τους στόχους της εργασίας, τις περιπτώσεις μελέτης, τις ερευνητικές ερωτήσεις και το δείγμα που επιλέχθηκε να ελεγχθεί. Επίσης περιλαμβάνει επεξήγηση του στατιστικού ελέγχου που πραγματοποιήθηκε για τη διεξαγωγή των αποτελεσμάτων.

Στο κεφάλαιο επτά παρουσιάζονται τα αποτελέσματα που προέκυψαν από την στατιστική ανάλυση με δείγμα τα δεδομένα εξόδου του λογισμικού που αναπτύχτηκε, πάνω στα τρία έργα λογισμικού που επιλέχθηκαν για ανάλυση. Γίνεται μία μικρή επεξήγηση αυτών. Επίσης το κεφάλαιο αυτό περιλαμβάνει πέρα από πίνακες αποτελεσμάτων, διαγράμματα θερμότητας, κυκλικά διαγράμματα και διάγραμμα γραμμών.

Τέλος, στο κεφάλαιο οκτώ γίνεται ανάλυση και επεξήγηση των αποτελεσμάτων. Ακολουθεί συζήτηση σχετικά με το πόσο επιτεύχθηκαν οι στόχοι, τι συμπεράσματα προέκυψαν και προτάσεις για μελλοντικές βελτιώσεις.

2. Αρχιτεκτονική Λογισμικού

"If you think good architecture is expensive, try bad architecture."
— Brian Foote

Κατά τη διάρκεια σχεδίασης οποιουδήποτε τεχνικού έργου, επιδιώκεται η αποσύνθεση του συστήματος σε τμήματα, η ανάθεση αρμοδιοτήτων σε κάθε τμήμα και η επικύρωση ότι όλα τα τμήματα μαζί επιτυγχάνουν τους σκοπούς του συστήματος. Στα πλαίσια του λογισμικού, η σχεδίαση είναι μια διαδικασία επίλυσης, της οποίας στόχος είναι η περιγραφή του τρόπου υλοποίησης των λειτουργικών απαιτήσεων, υπό τους περιορισμούς που θέτουν οι μη λειτουργικές απαιτήσεις (συμπεριλαμβανομένων των περιορισμών κόστους και χρόνου) και η οποία συμμορφώνεται με συγκεκριμένες αρχές καλής ποιότητας.

Η διαδικασία αυτή περιλαμβάνει συνήθως ένα πλήθος σχεδιαστικών προβλημάτων, όπου κάθε ένα από αυτά οποία μπορεί να επιλυθεί με πολλές και διαφορετικές λύσεις. Προτιμάτε να γίνεται χρήση τρόπων γραφικής απεικόνισης των λύσεων (διαγράμματα UML) διότι βοηθούν στην κατανόηση του προβλήματος, της προτεινόμενης λύσης και αποτελούν ένα δυνατό εργαλείο για την εξέλιξη του λογισμικού. Ένα λογισμικό μπορεί να εξελιχθεί από πολλούς και διαφορετικούς προγραμματιστές οπότε η σωστή σχεδίαση και η καταγραφή αυτής με διαγράμματα αποτελεί χρήσιμη λύση η οποία αποδίδει χρονικά και οικονομικά. Ένα αντικειμενοστρεφές σύστημα λογισμικού, μπορεί προφανώς να δομηθεί με πάρα πολλούς τρόπους, αν αναλογιστεί κανείς τον αριθμό των δυνατών κλάσεων, των λειτουργιών που μπορούν να περιλαμβάνουν καθώς και των δυνατών συσχετίσεων μεταξύ τους.

Το κόστος της σχεδίασης στον κύκλο ζωής ενός προϊόντος λογισμικού είναι σημαντικό και εξαρτάται κυρίως από το μέγεθος του συστήματος και κατ' επέκταση από το πλήθος και είδος των απαιτήσεων. Ωστόσο, αυτό που καθιστά τη σχεδίαση ιδιαιτέρως σημαντική από πλευράς κόστους, είναι το ότι καθορίζει σε εξαιρετικά μεγάλο βαθμό, την ευκολία με την οποία μπορούν να πραγματοποιηθούν αλλαγές και επεκτάσεις στο σύστημα. Εφόσον το κόστος συντήρησης ενός μεγάλου έργου λογισμικού είναι συνήθως πολλαπλάσιο του κόστους ανάπτυξής του, γίνεται εύκολα κατανοητό, ότι όσο μεγαλύτερη πρόνοια ληφθεί κατά τη διάρκεια της σχεδίασης ώστε το σύστημα να καθίσταται εύκολα επεκτάσιμο και τροποποιήσιμο, τόσο οικονομικότερη γίνεται η εξέλιξη του λογισμικού. Παρόλο που είναι σχετικά δύσκολο να οριστεί τι συνιστά η "καλή" αντικειμενοστρεφή σχεδίαση

(χρησιμοποιώντας πρότυπα όπως το ISO 9001 και ISO 9126), είναι αρκετά ευκολότερο να διατυπωθούν τα συμπτώματα μιας σχεδίασης "κακής" ποιότητας. Τα συμπτώματα που μπορεί να παρουσιαστούν είτε κατά την αρχική ανάπτυξης του λογισμικού, είτε πολύ περισσότερο κατά την εξέλιξη που υφίσταται στη διάρκεια ζωής του είναι τα εξής [3]:

1) *Δυσκαμψία (Rigidity):* Το σύστημα είναι δύσκολο να τροποποιηθεί διότι κάθε αλλαγή σε κάποια μονάδα λογισμικού οδηγεί σε πληθώρα αλλαγών σε άλλες μονάδες του συστήματος. Είναι η τάση του λογισμικού να είναι δύσκολο να αλλάξει ακόμα και με απλούς τρόπους. Με το πρόβλημα αυτό έρχονται πολύ συχνά αντιμέτωποι οι προγραμματιστές όταν, για μια φαινομενικά απλή αλλαγή αδυνατούν να εκτιμήσουν ορθά την απαιτούμενη προσπάθεια. Συνήθως, πρόκειται για αλλαγές που πρέπει να γίνουν σε αλληλοεξαρτώμενες μονάδες λογισμικού, οι οποίες για να λειτουργήσουν ορθά, πρέπει να τροποποιηθούν καταλλήλως.
2) *Ευθραυστότητα (Fragility):* Οι αλλαγές που πραγματοποιούνται στο λογισμικό προκαλούν σφάλματα σε διάφορα σημεία. Συχνά, τα νέα προβλήματα ανακύπτουν σε περιοχές φαινομενικά άσχετες προς αυτήν στην οποία πραγματοποιήθηκε η αλλαγή. Η διόρθωση των νέων προβλημάτων δημιουργεί με τη σειρά της νέα προβλήματα και αποτυχίες του λογισμικού. Όσο το λογισμικό γίνεται πιο εύθραυστο, τόσο η πιθανότητα εμφάνισης νέων προβλημάτων πλησιάζει τη βεβαιότητα, με άλλα λόγια η λίστα των bugs δεν αδειάζει ποτέ.
3) *Ακινησία (Immobility):* Υπάρχει δυσκολία διαχωρισμού του συστήματος σε συστατικά τα οποία μπορούν να επαναχρησιμοποιηθούν σε άλλες εφαρμογές. Μια σχεδίαση χαρακτηρίζεται από ακινησία, αν περιλαμβάνει τμήματα που θα μπορούσαν να χρησιμοποιηθούν σε άλλες περιπτώσεις, αλλά η προσπάθεια και το ρίσκο που εμπλέκονται στο διαχωρισμό αυτών των τμημάτων είναι πολύ μεγάλα με αποτέλεσμα να μην γίνεται ο διαχωρισμός αυτός.
4) *Έλλειψη ρευστότητας (Viscosity):* Η πραγματοποίηση τροποποιήσεων με λάθος τρόπο είναι ευκολότερη από την πραγματοποίησή τους με τον ορθό τρόπο. Όταν απαιτούνται αλλαγές, οι προγραμματιστές συνήθως βρίσκουν περισσότερους από έναν τρόπους για να τις πραγματοποιήσουν, μερικές από αυτές αλλοιώνουν την καλή σχεδίαση ενώ άλλες τη διατηρούν. Αν οι πρώτες αλλαγές πραγματοποιούνται δυσκολότερα από ότι οι δεύτερες, το λογισμικό παρουσιάζει μεγάλη έλλειψη ρευστότητας.

5) *Περιττή Πολυπλοκότητα (Needless Complexity):* Το λογισμικό περιλαμβάνει στοιχεία που δεν προσφέρουν (ούτε πρόκειται να προσφέρουν) κάποιο όφελος. Η ύπαρξη περίπλοκων δομών κώδικα, είτε λόγω αδυναμίας εύρεσης αποδοτικότερων δομών, είτε λόγω ενσωμάτωσης πιθανών μελλοντικών απαιτήσεων χρήστη, οδηγεί συνήθως σε εκφυλισμό της αρχικής σχεδίασης και σε δυσκολία κατανόησης του συστήματος.
6) *Περιττή Επανάληψη (Needless Repetition):* Η σχεδίαση περιλαμβάνει επαναλαμβανόμενες δομές που θα μπορούσαν να ενοποιηθούν υπό μία κοινή αφαίρεση. Οι λειτουργίες αντιγραφής και επικόλλησης (copy-paste) μπορεί να είναι χρήσιμες σε επίπεδο επεξεργασίας κειμένου ή κώδικα, αλλά έχουν συχνά καταστροφικές συνέπειες για το σύστημα αν χρησιμοποιούνται χωρίς σύνεση. Όταν το ίδιο τμήμα κώδικα εμφανίζεται σε διάφορα σημεία με ελαφρά διαφοροποιημένες εκδόσεις, η ομάδα ανάπτυξης λογισμικού παραβλέπει συχνά την υπερκείμενη αφαίρεση και τις ενδεχόμενες αλλαγές που πρέπει να πραγματοποιούνται σε κάθε σημείο ξεχωριστά. Η εύρεση όλων των επαναλαμβανόμενων τμημάτων και η εξάλειψή τους χρησιμοποιώντας μια κατάλληλη αφαίρεση, βελτιώνει αισθητά την ποιότητα του σχεδίου, του χρόνου επέκτασης ή τροποποίησης του λογισμικού και φυσικά κάνει πιο εύκολη τη δουλειά των προγραμματιστών.
7) *Αδιαφάνεια (Opacity):* Δυσκολία κατανόησης μιας μονάδας (σε επίπεδο σχεδίου ή κώδικα). Ο κώδικας μπορεί να γραφεί με ξεκάθαρο και εκφραστικό τρόπο, είτε μπορεί να γραφεί με περίπλοκο και αδιαφανή τρόπο. Η εξέλιξη του κώδικα με την πάροδο του χρόνου, συνήθως γίνεται ολοένα και πιο αδιαφανής. Για την αποφυγή αυτού του χαρακτηριστικού απαιτείται συστηματική προσπάθεια συγγραφής κώδικα, έτσι ώστε αυτός να γίνεται εύκολα κατανοητός από τρίτους και ταυτοχρόνως συχνή επισκόπηση από άλλους προγραμματιστές. Όπως χιουμοριστικά αναφέρει ο Robert C. Martin [17], "*Η σωστή χρήση των σχολίων είναι για να αντισταθμίσουμε την αποτυχία μας να εκφράσουμε τον εαυτό μας στον κώδικα. Τα σχόλια είναι πάντα αποτυχίες. Χρειάζονται, γιατί δεν μπορούμε πάντα να καταλάβουμε πώς να εκφραζόμαστε χωρίς αυτά, αλλά η χρήση τους δεν είναι λόγος γιορτής. Έτσι όταν βρίσκεστε σε μια θέση όπου πρέπει να γράψετε ένα σχόλιο, σκεφτείτε το και δείτε αν υπάρχει κάποιος τρόπος να εκφράσετε τον εαυτό σας στον κώδικα.*"

Συνοψίζοντας, είναι υψίστης σημασίας η σχεδίαση του λογισμικού να διατηρείτε όσο καλύτερη γίνεται καθ' όλη τη διάρκεια του κύκλου ζωής του. Μια βασική αρχή που

υποστηρίζουν οι καλοί προγραμματιστές είναι ότι δεν αφήνουν ποτέ τα προβλήματα για την επόμενη μέρα. Μόλις τα εντοπίσουν τα διορθώνουν, έτσι προλαμβάνεται το πρόβλημα του να ξεχαστούν και να μαζευτούν προβλήματα στο μέλλον. Ο σχεδιασμός και ο κώδικας πρέπει να παραμένουν "καθαροί" και η διαδικασία για να επιτευχθεί αυτό απαιτεί δέσμευση και συνεχής προσπάθεια.

Μανιφέστο για την ευέλικτη ανάπτυξη λογισμικού [3] :

- Η ύψιστη προτεραιότητά της ομάδας ανάπτυξης λογισμικού είναι η ικανοποίηση του πελάτη μέσω της έγκαιρης και συνεχούς παράδοσης πολύτιμου-χρήσιμου λογισμικού.
- Οι αλλαγές των απαιτήσεων είναι καλοδεχούμενες, ακόμη και σε αργά στάδια της ανάπτυξης. Ο καλός και ευέλικτος σχεδιασμός λογισμικού δίνει ένα δυνατό ανταγωνιστικό πλεονέκτημα στον πελάτη.
- Συχνή παράδοση λειτουργικού λογισμικού, από μερικές εβδομάδες μέχρι μερικούς μήνες, με προτίμηση το συντομότερο χρονικό διάστημα.
- Οι επιχειρηματίες και οι προγραμματιστές πρέπει να συνεργάζονται καθημερινά καθ' όλη τη διάρκεια του έργου.
- Η ομάδα ανάπτυξης λογισμικού πρέπει να αποτελείται από άτομα εμπνευσμένα, με όρεξη να παρέχουν τις καλύτερες υπηρεσίες. Οι υπεύθυνοι και οι διαχειριστές έχουν ως κύρια αρμοδιότητα τη δημιουργία περιβάλλοντος, παροχής υποστήριξης και εμπιστοσύνης προς αυτούς για την ολοκλήρωση της δουλειάς.
- Η πιο αποδοτική και αποτελεσματική μέθοδος μεταφοράς πληροφοριών σε μια ομάδα ανάπτυξης και εντός αυτής είναι η επικοινωνία πρόσωπο με πρόσωπο.
- Το λειτουργικό λογισμικό είναι το κύριο μέτρο της προόδου.
- Οι ευέλικτες διαδικασίες προάγουν την αειφόρο ανάπτυξη. Οι χορηγοί, οι προγραμματιστές και οι χρήστες θα πρέπει να μπορούν να διατηρούν σταθερό ρυθμό επ' αόριστον.
- Η συνεχής προσοχή στην τεχνική αριστεία και τον καλό σχεδιασμό ενισχύει την ευελιξία.
- Η απλότητα - η τέχνη της μεγιστοποίησης της ποσότητας της εργασίας που δεν έχει γίνει - είναι απαραίτητη.
- Οι καλύτερες αρχιτεκτονικές, απαιτήσεις και σχέδια προκύπτουν από αυτο-οργανωτικές ομάδες.

- Σε τακτά χρονικά διαστήματα, η ομάδα αντικατοπτρίζει τον τρόπο με τον οποίο θα γίνει πιο αποτελεσματική, και στη συνέχεια συντονίζει και προσαρμόζει ανάλογα τη συμπεριφορά της.

2.1 Βασικές Αρχές Σχεδίασης Λογισμικού

"… It is simplicity that is difficult to make."
— Bertholdt Brecht

Ο σχεδιασμός του λογισμικού αποδίδει τρία επίπεδα αποτελεσμάτων, τον αρχιτεκτονικό σχεδιασμό, τη σχεδίαση υψηλού επιπέδου και τη λεπτομερής σχεδίαση. Ο αρχιτεκτονικός σχεδιασμός είναι μία αφηρημένη έκδοση του συστήματος και προσδιορίζει το λογισμικό ως ένα σύστημα με πολλά στοιχεία που αλληλεπιδρούν μεταξύ τους. Η σχεδίαση υψηλού επιπέδου σπάει την αντίληψη «αρχιτεκτονικού σχεδιασμού μιας ενιαίας οντότητας πολλαπλών συστατικών» σε μια λιγότερο περιορισμένη άποψη των υποσυστημάτων και των ενοτήτων και απεικονίζει την αλληλεπίδρασή μεταξύ τους. Η λεπτομερής σχεδίαση είναι λεπτομερέστερη απέναντι στις ενότητες και τις εφαρμογές τους. Καθορίζει τη λογική δομή κάθε μονάδας και των διεπαφών της για επικοινωνία με άλλες μονάδες. Στη συγκεκριμένη εργασίας μας ενδιαφέρει η λεπτομερής σχεδίαση.

Η τμηματικότητα (modularity) είναι μια τεχνική για τη διάσπαση ενός συστήματος λογισμικού σε πολλαπλές διακριτές και ανεξάρτητες μονάδες, οι οποίες αναμένεται να είναι ικανές να εκτελούν καθήκοντα ανεξάρτητα. Αυτές οι ενότητες μπορούν να λειτουργήσουν ως βασικές δομές για όλο το λογισμικό. Οι σχεδιαστές τείνουν να σχεδιάζουν ενότητες έτσι ώστε να μπορούν να εκτελούνται και / ή να καταρτίζονται ξεχωριστά και ανεξάρτητα. Πλεονεκτήματα της τμηματικότητας είναι ότι τα μικρότερα συστατικά είναι ευκολότερα προς τη συντήρηση και την υποστήριξη, το πρόγραμμα μπορεί να χωριστεί με βάση τις λειτουργικές πτυχές του, το πρόγραμμα μπορεί να φτάσει το επιθυμητό επίπεδο αφαίρεσης, τα τμήματα με υψηλή συνοχή μπορούν να ξαναχρησιμοποιηθούν, καθίσταται δυνατή η ταυτόχρονη εκτέλεση και είναι επιθυμητή από άποψη ασφάλειας.

Όταν ένα πρόγραμμα λογισμικού είναι τμηματοποιημένο, τα καθήκοντά του χωρίζονται σε διάφορες ενότητες βασισμένες σε ορισμένα χαρακτηριστικά. Οι ενότητες είναι ένα σύνολο οδηγιών που συνθέτουν για να επιτύχουν κάποιες εργασίες. Ωστόσο, θεωρούνται

ως ενιαία οντότητα, αλλά μπορούν να επικοινωνούν μεταξύ τους για να συνεργαστούν. Υπάρχουν μέτρα με τα οποία μπορεί να μετρηθεί η ποιότητα ενός σχεδιασμού μονάδων και η αλληλεπίδρασή μεταξύ τους. Τα μέτρα αυτά ονομάζονται σύζευξη και συνοχή. Η συνοχή είναι ένα μέτρο που καθορίζει το βαθμό ενδο-αξιοπιστίας μέσα στα στοιχεία μιας ενότητας. Όσο μεγαλύτερη είναι η συνοχή, τόσο καλύτερος είναι ο σχεδιασμός του προγράμματος. Η σύζευξη είναι ένα μέτρο που καθορίζει το επίπεδο αλληλεξάρτησης μεταξύ των ενοτήτων ενός προγράμματος. Δείχνει σε ποιο επίπεδο οι ενότητες παρεμβαίνουν και αλληλεπιδρούν μεταξύ τους. Όσο χαμηλότερη είναι η σύζευξη, τόσο καλύτερα σχεδιασμένο είναι το πρόγραμμα.

2.2 Αρχές που Διέπουν την Αντικειμενοστρεφή Σχεδίαση

Στην ενότητα αυτή θα αναλυθούν οι αρχές (principles) που πρέπει να διέπουν την αντικειμενοστρεφή σχεδίαση ενός συστήματος λογισμικού. Η παραβίαση μιας ή περισσοτέρων από αυτές τις αρχές οδηγεί σχεδόν με βεβαιότητα σε ένα ή περισσότερα από τα συμπτώματα που περιγράφονται στην προηγούμενη ενότητα ως τεχνικές κακής σχεδίασης.

Αρχή της Μοναδικής Αρμοδιότητας - (SRP: The Single Responsibility Principle) :

Η αρχή αυτή περιγράφηκε στη δουλειά του Tom DeMarco (1979) και του Meilir Page-Jones την οποία ονόμασαν συνοχή. Όρισαν τη συνοχή ως τη λειτουργική συγγένεια των στοιχείων μιας ενότητας. Η Αρχή της Μοναδικής Αρμοδιότητας ορίζει ότι μια κλάση πρέπει να έχει μόνο ένα λόγο να αλλάξει. Μια κλάση δεν πρέπει να έχει περισσότερες από μία αρμοδιότητες. Κάθε αρμοδιότητα συνιστά έναν εν δυνάμει άξονα αλλαγών. Όταν αλλάξουν οι απαιτήσεις του συστήματος, οι αλλαγές εμφανίζονται μέσα στο λογισμικό ως αλλαγή στις αρμοδιότητες μιας (ή περισσοτέρων) κλάσεων. Αν μια κλάση αναλαμβάνει περισσότερες από μία αρμοδιότητες, θα υπάρχουν περισσότεροι από ένας λόγοι αλλαγής της. Η ίδια αρχή ισχύει και για το επίπεδο πακέτου. Κάθε πακέτο πρέπει να χαρακτηρίζεται από μία αρμοδιότητα την οποία κρίνονται να υλοποιήσουν οι κλάσεις που περιλαμβάνει.

Επιπλέον, αν μια κλάση έχει πολλαπλές αρμοδιότητες, τότε μεταξύ των αρμοδιοτήτων εμφανίζεται αναπόφευκτα σύζευξη. Οι αλλαγές στη μία αρμοδιότητα είναι δυνατό να εμποδίσουν αλλαγές για την ικανοποίηση μιας άλλης απαίτησης ή ακόμη χειρότερα, αν ο προγραμματιστής δεν επιδείξει την κατάλληλη προσοχή, είναι δυνατό να

δημιουργηθούν σφάλματα στις άλλες αρμοδιότητες. Είναι προφανές ότι, στην περίπτωση αυτή το λογισμικό γίνεται δύσκαμπτο και εύθραυστο.

Στην περίπτωση που υπάρχουν περισσότερες από μία αρμοδιότητες, καλό είναι να ελέγχεται η πράξη του διαχωρισμού. Δύο τμήματα κώδικα τα οποία αλλάζουν για διαφορετικούς λόγους αλλά περιέχονται μέσα στην ίδια ενότητα πρέπει να χωριστούν σε δύο ενότητες όπου η κάθε ενότητα θα είναι υπεύθυνη για μία λειτουργία. Γίνεται χρήση της λέξης ενότητα διότι η αρχή αυτή λειτουργεί και σε επίπεδο κλάσης και σε επίπεδο πακέτου όπως αναφέρθηκε παραπάνω.

Η έννοια της συνεκτικότητας είναι εύκολα κατανοητή αλλά η Αρχή της Μοναδικής Αρμοδιότητας είναι μια από τις δυσκολότερες αρχές στην εφαρμογή τους. Ο συγκερασμός αρμοδιοτήτων είναι κάτι που πραγματοποιείται από τους προγραμματιστές κατά φυσικό τρόπο. Ο εντοπισμός και ο διαχωρισμός μη συνεκτικών λειτουργιών αποτελεί κατά πολλούς ένα ουσιαστικό κομμάτι της σχεδίασης και ανάπτυξης λογισμικού.

Αρχή της Ανοικτής-Κλειστής Σχεδίασης - (OCP: The Open-Closed Principle) :

Πολύ συχνά, μία μοναδική αλλαγή σε ένα πρόγραμμα καταλήγει σε αλυσιδωτές αλλαγές (ripple effects) σε διάφορες εξαρτώμενες μονάδες, με αποτέλεσμα η σχεδίαση να χαρακτηρίζεται από ακαμψία και ευθραυστότητα. Το ζητούμενο είναι αν το σύστημα μπορεί να υποστεί αναδόμηση (refactoring), έτσι ώστε περαιτέρω αλλαγές του ιδίου τύπου να μην προκαλέσουν περαιτέρω τροποποιήσεις. Ο Bertrand Meyer το 1988 διατύπωσε προς αυτή την κατεύθυνση τη διάσημη Αρχή της Ανοικτής-Κλειστής Σχεδίασης (Open-Closed Principle): Οι οντότητες λογισμικού (κλάσεις, μονάδες, συναρτήσεις κλπ), θα πρέπει να είναι ανοιχτές για επέκταση, αλλά κλειστές για τροποποίηση. "Ανοικτές για επέκταση", η ιδιότητα αυτή σημαίνει ότι η συμπεριφορά της μονάδας μπορεί να επεκταθεί, προσθέτοντας νέες λειτουργίες που ικανοποιούν τις καινούργιες απαιτήσεις. "Κλειστές για τροποποίηση", η επέκταση της συμπεριφοράς δεν οδηγεί σε αλλαγές του πηγαίου ή αντικειμένου κώδικα της μονάδας. Με άλλα λόγια, προστίθεται νέα λειτουργικότητα χωρίς να τροποποιηθεί η εκτελέσιμη εκδοχή της μονάδας, (βιβλιοθήκη, DLL, ή .jar αρχείο).

Μια από τις θεμελιώδεις έννοιες σε όλες τις αντικειμενοστρεφείς γλώσσες προγραμματισμού, είναι η έννοια της αφαίρεσης (abstraction). Αφαίρεση ονομάζεται η σκόπιμη σύμπτυξη ή απόκρυψη πληροφορίας που αφορά μια διαδικασία ή μία ενότητα, με σκοπό την καλύτερη κατανόηση άλλων απόψεων, λεπτομερειών ή δομής. Η εφαρμογή

αφαίρεσης σε ένα σύστημα συμβάλλει στην απόκρυψη πληροφορίας (information hiding), μια τεχνική σύμφωνα με την οποία ορίζει σε κάποιο επίπεδο (π.χ. σε μια βασική κλάση) τις σημαντικές μόνο πλευρές μιας λειτουργίας ή μιας οντότητας και αγνοεί τις υπόλοιπες λεπτομέρειες οι οποίες ορίζονται σε κάποιο χαμηλότερο επίπεδο (π.χ. σε μια παράγωγη κλάση). Η έννοια της αφαίρεσης μπορεί να θεωρηθεί ως η εισαγωγή δομής σε ένα σύστημα και μπορεί να υποδιαιρεθεί σε δύο υποκατηγορίες. Η πλέον συνήθης τεχνική αφαίρεσης είναι ο διαμερισμός ενός συνόλου στα τμήματα που το αποτελούν. Η τεχνική αυτή δεν διαφέρει από την κλασική στρατηγική του "διαίρει και βασίλευε" (divide and conquer). Η άλλη τεχνική αφαίρεσης βασίζεται στην εξειδίκευση του γενικού, προσέγγιση που υλοποιείται στις αντικειμενοστρεφείς γλώσσες με τις αρχές της κληρονομικότητας. Μια μονάδα λογισμικού είναι δυνατό να "χειρίζεται" μια αφαίρεση. Μια τέτοια μονάδα μπορεί να είναι κλειστή για τροποποιήσεις καθώς εξαρτάται από την αφαίρεση που είναι σταθερή. Ωστόσο, η συμπεριφορά της μονάδας μπορεί να επεκταθεί δημιουργώντας νέες παράγωγες κλάσεις της αφαίρεσης.

Ο αντικειμενοστρεφής προγραμματισμός προσφέρει σημαντικά πλεονεκτήματα στην περίπτωση ανάπτυξης συστημάτων που πρόκειται να εξελιχθούν λόγω αλλαγών στις απαιτήσεις. Αν υποθέσουμε ότι ένα σύστημα πρόκειται να κατασκευαστεί και να παραμείνει ως έχει για ολόκληρο τον κύκλο ζωής του, τότε ο αντικειμενοστρεφής προγραμματισμός όχι μόνο δεν υπερέχει έναντι άλλων μοντέλων, αλλά ίσως να προσθέτει περιττή πολυπλοκότητα. Αν όμως, όπως συμβαίνει στην πραγματικότητα, οι αλλαγές στις απαιτήσεις επιβάλλουν την τροποποίηση του λογισμικού, τότε η Αρχή της Ανοικτής-Κλειστής Σχεδίασης εξασφαλίζει ότι οι αλλαγές θα πραγματοποιηθούν με τη μικρότερη δυνατή προσπάθεια, χρόνο και κόστος.

Αρχή της Υποκατάστασης της Liskov - (LSP: The Liskov Substitution Principle) :

Η Barbara Liskov από το MIT (Liskov, 1988) θεώρησε τον ίδιο τον αντικειμενοστρεφή προγραμματισμό ως μια τεχνική αφαίρεσης και διατύπωσε την εξής αρχή σχετικά με μια ιεραρχία τύπων: Αυτό που είναι επιθυμητό εδώ είναι κάτι σαν την ακόλουθη ιδιότητα υποκατάστασης: Αν για κάθε αντικείμενο object1 του τύπου S υπάρχει ένα αντικείμενο object2 του τύπου T τέτοιο ώστε για όλα τα προγράμματα P που ορίζονται υπό όρους του T η συμπεριφορά του P παραμένει αναλλοίωτη όταν το object1 υποκαταστήσει το object2 τότε, ο S είναι παράγωγος τύπος (υποκατηγορία) του T. Η αντίστοιχη αρχή

αντικειμενοστρεφούς σχεδίασης μπορεί να διατυπωθεί ως εξής: Οι παράγωγοι τύποι πρέπει να μπορούν να υποκαθιστούν τους βασικούς τους τύπους.

Η σημασία αυτής της αρχής καθίσταται φανερή αν αναλογιστεί κανείς τις συνέπειες της παραβίασής της. Έστω ότι υπάρχει μια συνάρτηση function1() η οποία λαμβάνει ως παράμετρο έναν δείκτη ή μία αναφορά προς μια βασική κλάση A (π.χ. μια λίστα). Υποθέστε επίσης ότι υπάρχει μια παράγωγος κλάση B της A (π.χ. ένα σύνολο), η οποία όταν περνά στη συνάρτηση function1() ως παράμετρος, αναγκάζει την function1() να συμπεριφερθεί με λάθος τρόπο. Τότε η κλάση B παραβιάζει την Αρχή Υποκατάστασης της Liskov (Liskov Substitution Principle).

Αρχή Αναστροφής Εξαρτήσεων - (DIP : Dependency Inversion Principle) :

Στόχος της Αρχής της Αντιστροφής των Εξαρτήσεων (Dependency-Inversion Principle) είναι να αντιστρέφει πλήρως τη δομή ενός συστήματος λογισμικού, έτσι ώστε οι μονάδες υψηλού επιπέδου να μην εξαρτώνται αλλά να καθορίζουν τους κανόνες για τις μονάδες χαμηλού επιπέδου. Ένας έμπειρος σχεδιαστής, πρέπει να μπορεί να διαπιστώσει εύκολα αν η δομή ενός αντικειμενοστρεφούς συστήματος είναι πράγματι "αντεστραμμένη".

Η αρχή διατυπώνεται ως εξής: (α) Οι μονάδες υψηλού επιπέδου δεν θα πρέπει να εξαρτώνται από μονάδες χαμηλού επιπέδου. (β) Οι αφαιρέσεις δεν θα πρέπει να εξαρτώνται από λεπτομέρειες. Οι λεπτομέρειες θα πρέπει να εξαρτώνται από αφαιρέσεις. Σε μια αντεστραμμένη διαστρωμάτωση μια κλάση που αποτελεί τμήμα ενός υψηλού επιπέδου δηλώνει μια αφηρημένη διασύνδεση για τις λειτουργίες που χρειάζεται από κλάσεις χαμηλότερων επιπέδων. Τονίζεται ότι η κλάση υψηλού επιπέδου δεν διατηρεί απευθείας αναφορές (δείκτες) προς τις κλάσεις χαμηλού επιπέδου, αλλά χειρίζεται μόνο τη διασύνδεση, δυνατότητα που όπως είναι γνωστό, είναι εφικτή στις αντικειμενοστρεφείς γλώσσες. Κατά συνέπεια, δεν εξαρτάται από τις κλάσεις των χαμηλότερων στρωμάτων. Μια κλάση χαμηλού επιπέδου "υποχρεούται" να υλοποιήσει τις μεθόδους που καθορίζονται από τη διασύνδεση, ώστε να παράσχει τις αντίστοιχες λειτουργίες, και υπό αυτή την έννοια "εξαρτάται" από την αφηρημένη διασύνδεση που δηλώνεται σε ένα ανώτερο στρώμα του λογισμικού. Βεβαίως, κατά τη διάρκεια της εκτέλεσης, στη θέση της αναφοράς που διατηρεί η κλάση υψηλού επιπέδου προς τη διασύνδεση, "περνάει" μια αναφορά προς μια κλάση χαμηλού επιπέδου, ενέργεια που όμως δεν επηρεάζει τον κώδικα της κλάσης.

Ο παραδοσιακός διαδικασιακός προγραμματισμός δημιουργεί δομές όπου η υψηλού επιπέδου πολιτική ενός προγράμματος εξαρτάται από τις λεπτομέρειες υλοποίησης. Η αρχιτεκτονική αυτή είναι ατυχής καθώς η πολιτική καθίσταται ευάλωτη σε αλλαγές της υλοποίησης και δεν μπορεί να επαναχρησιμοποιηθεί σε νέα περιβάλλοντα. Η αντικειμενοστρεφής σχεδίαση πρέπει να έχει ως στόχο την αντιστροφή αυτής της εξάρτησης, έτσι ώστε οι λεπτομέρειες και η πολιτική να εξαρτώνται μόνο από αφαιρέσεις. Πολλοί θεωρούν τη διαφορά αυτή ως ειδοποιό: Αν οι εξαρτήσεις είναι αντεστραμμένες πρόκειται για αντικειμενοστρεφή σχεδίαση. Αν όχι, πρόκειται για διαδικασιακό σχεδιασμό.

Αρχή του Διαχωρισμού Διασυνδέσεων - (ISP : The Interface-Segregation Principle) :

Παρόλο που η σχεδίαση με διασυνδέσεις είναι καλή τακτική, θα πρέπει να ελέγχεται και να υλοποιείται προσεκτικά η ίδια η σχεδίαση των διασυνδέσεων. Ενώ μια διασύνδεση είναι ο τρόπος με τον οποίο εξασφαλίζεται ότι τα αντικείμενα των κλάσεων που υλοποιούν τη διασύνδεση θα παρουσιάζουν συγκεκριμένη συμπεριφορά, δημιουργούνται προβλήματα όταν πολλαπλά είδη αντικειμένων υλοποιούν την ίδια διασύνδεση. Η Αρχή του Διαχωρισμού των Διασυνδέσεων (Interface-Segregation Pricinple) διαπραγματεύεται τα μειονεκτήματα των ογκωδών διασυνδέσεων ("fat" interfaces). Οι κλάσεις που έχουν μεγάλο αριθμό δημόσιων μεθόδων είναι κλάσεις των οποίων η διασύνδεση δεν είναι συνεκτική και οι μέθοδοι κατά πάσα πιθανότητα μπορούν να διαχωριστούν σε ένα σύνολο διασυνδέσεων. Κάθε σύνολο εξυπηρετεί μια διαφορετική ομάδα από πελάτες ή αρμοδιότητες και παρουσιάζει υψηλότερη συνεκτικότητα από την αρχική διασύνδεση. Η αρχή διατυπώνεται ως εξής: Πολλές εξειδικευμένες διασυνδέσεις είναι προτιμότερες από μια γενική διασύνδεση.

Μια γενική διασύνδεση, προφανώς περιλαμβάνει μεθόδους που εξυπηρετούν πολλούς διαφορετικούς πελάτες ή αρμοδιότητες. Μια διασύνδεση θεωρείται "ιδιοκτησία" των κλάσεων πελατών που τη χρησιμοποιούν και όχι αυτών που την υλοποιούν και ότι κάτι τέτοιο πρέπει να αποτυπώνεται και στο όνομα της διασύνδεσης. Συνήθως, θεωρεί κανείς ότι οι αλλαγές στις διασυνδέσεις είναι αυτές που επιβάλλουν αλλαγές στις κλάσεις - πελάτες. Ωστόσο, τις περισσότερες φορές οι αλλαγές αυτές ουσιαστικά ξεκινούν από τις ίδιες τις κλάσεις - πελάτες. Η λειτουργικότητα των κλάσεων - πελατών είναι αυτή που τροποποιήθηκε και επέβαλε την αλλαγή στη διασύνδεση.

Με βάση την παραπάνω παρατήρηση, προκύπτει ότι όλες οι κλάσεις - πελάτες μιας διασύνδεσης ασκούν επιρροή στη διασύνδεση που διαθέτουν και κατά καιρούς επιβάλλουν αλλαγές σε αυτήν. Αν επομένως, κλάσεις με τελείως διαφορετικές ανάγκες, χρησιμοποιούν την ίδια διασύνδεση, υπόκεινται στις αλλαγές των υπολοίπων και δημιουργείται μια αναίτια σχέση εξάρτησης μεταξύ τους. Για το λόγο αυτό η Αρχή Διαχωρισμού των Διασυνδέσεων διατυπώνεται με βάση το λόγο ύπαρξης της, ως εξής: Οι πελάτες δεν θα πρέπει να εξαναγκάζονται σε εξάρτηση από μεθόδους που δεν χρησιμοποιούν. Η εξήγηση είναι απλή: Αν οι πελάτες εξαρτώνται από μεθόδους που δεν χρησιμοποιούν, υπόκεινται στις αλλαγές που συμβαίνουν σε αυτές τις μεθόδους (και οι οποίες προκαλούνται από άλλους, άσχετους πελάτες). Στην περίπτωση αυτή, υπάρχει ανεπιθύμητη σύζευξη μεταξύ των πελατών. Στόχος είναι η μείωση της σύζευξης και επιτυγχάνεται με το διαχωρισμό των διασυνδέσεων.

2.3 Αρχές Σχεδιασμού σε Επίπεδο Πακέτου

“ … with proper design, the features come cheaply. This approach is arduous, but continues to succeed.”

— Dennis Ritchie

Καθώς ένα λογισμικό μεγαλώνει σε μέγεθος και πολυπλοκότητα, απαιτείται υψηλού επιπέδου οργάνωση. Οι κλάσεις αποτελούν μία πολύ βολική μονάδα οργάνωσης μικρών εφαρμογών αλλά δεν ισχύει το ίδιο σε περιπτώσεις μεγάλων εφαρμογών. Χρειάζεται κάτι μεγαλύτερο από την κλάση για να οργανώσει μεγάλα συστήματα και αυτό είναι το πακέτο. Η σχεδίαση του πακέτου διαθέτει τις δικές της αρχές καλής σχεδίασης. Οι αρχές σχεδιασμού πακέτου είναι έξι. Οι τρεις αφορούν τη συνοχή του πακέτου και οι άλλες τρεις τη σύζευξη.

Αρχικά, πακέτο ορίζεται μία μονάδα που περιέχει μία ομάδα κλάσεων. Η ομαδοποίηση των κλάσεων οδηγεί σε υψηλό επίπεδο αφαιρετικότητας. Αλλά οι κλάσεις έχουν εξαρτήσεις με άλλες κλάσεις, οι οποίες εξαρτήσεις συνήθως φτάνουν το επίπεδο πακέτου. Στόχος είναι η ομαδοποίηση των κλάσεων να γίνει με συγκεκριμένα κριτήρια τα οποία βελτιστοποιούν τη σύζευξη και τη συνοχή του πακέτου.

Λεπτομερειακότητα ή Αρχές Συνοχής Πακέτου ονομάζεται η πρώτη κατηγορία η οποία περιλαμβάνει τρεις αρχές που σχετίζονται με τη συνοχή του πακέτου.

- ***Αρχή Ισοδυναμίας Επαναχρησιμοποίησης / Κυκλοφορίας Έκδοσης - (Reuse-release Equivalence Principle (REP)) :***

  "*Το στοιχείο επαναχρησιμοποίησης είναι το στοιχείο κυκλοφορίας έκδοσης.*"

  Η REP ουσιαστικά σημαίνει ότι το πακέτο πρέπει να δημιουργηθεί με επαναχρησιμοποιούμενες κλάσεις - "Είτε όλες οι κλάσεις μέσα στο πακέτο είναι επαναχρησιμοποιήσιμες, είτε καμία από αυτές δεν είναι". Οι κλάσεις πρέπει επίσης να είναι της ίδιας οικογένειας. Όποιες από αυτές δεν σχετίζονται με το σκοπό του πακέτου δεν πρέπει να περιλαμβάνονται. Ένα πακέτο κατασκευασμένο ως οικογένεια επαναχρησιμοποιήσιμων κλάσεων τείνει να είναι πολύ χρήσιμο και ασφαλώς επαναχρησιμοποιούμενο. Ένα στοιχείο που μπορεί να επαναχρησιμοποιηθεί, είτε πρόκειται για ένα τμήμα, για μια κλάση είτε για ένα σύμπλεγμα κλάσεων, δεν μπορεί να επαναχρησιμοποιηθεί, εκτός εάν διαχειρίζεται από ένα σύστημα κυκλοφορίας εκδόσεων κάποιου είδους (μια ομάδα ανάπτυξης υπεύθυνη για τη βελτίωση και την αναβάθμιση του λογισμικού). Οι χρήστες δεν είναι πρόθυμοι να χρησιμοποιούν ένα στοιχείο λογισμικού εάν αναγκάζονται να κάνουν αναβάθμιση κάθε φορά που η ομάδα ανάπτυξης το αλλάζει. Έτσι, παρόλο που η ομάδα ανάπτυξης κυκλοφορεί μια νέα έκδοση της επαναχρησιμοποιούμενης μονάδας, η μονάδα λογισμικού πρέπει να είναι σε θέση να υποστηρίξει και να διατηρήσει παλαιότερες εκδόσεις για τους πελάτες που προτιμούν να παραμείνουν σε αυτές. Συνεπώς οι πελάτες αρνούνται να επαναχρησιμοποιήσουν μια μονάδα λογισμικού εκτός αν η ομάδα ανάπτυξης υπόσχεται να παρακολουθεί τους αριθμούς έκδοσης και να διατηρεί παλιές εκδόσεις, έστω για ένα χρονικό διάστημα. Κατά συνέπεια, ένα κριτήριο για την ομαδοποίηση των κλάσεων σε πακέτα είναι η επαναχρησιμοποίηση. Δεδομένου ότι τα πακέτα είναι η μονάδα απελευθέρωσης - δηλαδή η κυκλοφορία εκδόσεων λογισμικού αφορά αυτά, είναι επίσης η μονάδα επαναχρησιμοποίησης.

- ***Αρχή Κοινής Επαναχρησιμοποίησης - (The Common Reuse Principle (CRP)):***

  "*Οι κλάσεις που χρησιμοποιούνται μαζί βρίσκονται στο ίδιο πακέτο. Αν επαναχρησιμοποιηθεί μία κλάση, τότε πρέπει να επαναχρησιμοποιηθούν όλες.*"

  Η CRP δηλώνει ότι οι κλάσεις που τείνουν να επαναχρησιμοποιηθούν μαζί ανήκουν στο ίδιο πακέτο. Είναι ένας τρόπος που διευκολύνει την απόφαση σχετικά με το ποιες κλάσεις ανήκουν σε ποιο πακέτο. Εντός ενός πακέτου, οι κλάσεις πρέπει να είναι αδιαχώριστες και αλληλοεξαρτώμενες. Η αρχή αυτή επίσης δηλώνει ποιες κλάσεις

δεν πρέπει να βρίσκονται στο ίδιο πακέτο. Συνεπώς, κλάσεις οι οποίες δεν έχουν ισχυρές σχέσεις μεταξύ τους δεν πρέπει να βρίσκονται εντός του ίδιου πακέτου.

- ***Αρχή Κοινού Κλεισίματος - (The Common Closure Principle (CCP)) :***

  "*Οι κλάσεις που αλλάζουν μαζί, βρίσκονται στο ίδιο πακέτο.*"

  Η CCP δηλώνει ότι το πακέτο δεν πρέπει να έχει περισσότερους από έναν λόγους για να αλλάξει. Εάν οι αλλαγές επήλθαν σε μια εφαρμογή που εξαρτάται από μια σειρά πακέτων, στην ιδανική περίπτωση οι αλλαγές αυτές θα εμφανιστούν σε ένα πακέτο αντί σε έναν αριθμό από πακέτα. Αυτό βοηθάει στον εντοπισμό κλάσεων που είναι πιθανό να αλλάξουν, τις οποίες ομαδοποιούμε στο ίδιο πακέτο για τους ίδιους λόγους. Αν οι κλάσεις είναι στενά συζευγμένες, ανήκουν στο ίδιο πακέτο. Στην περίπτωση ενός μεγάλου αναπτυξιακού έργου, το λογισμικό υποδιαιρείται σε ένα μεγάλο δίκτυο αλληλένδετων πακέτων. Η εργασία για τη διαχείριση, δοκιμή και απελευθέρωση - κυκλοφορία νέων εκδόσεων αυτών των πακέτων δεν είναι καθόλου τετριμμένη. Όσο περισσότερα πακέτα αλλάζουν σε οποιαδήποτε δεδομένη έκδοση, τόσο μεγαλύτερη είναι η εργασία για την αναδημιουργία, τη δοκιμή και την ανάπτυξη της απελευθέρωσης - κυκλοφορίας εκδόσεων. Ως εκ τούτου, είναι επιθυμητό να ελαχιστοποιηθεί ο αριθμός των πακέτων που αλλάζουν σε οποιονδήποτε δεδομένο κύκλο έκδοσης του προϊόντος. Αυτό επιτυγχάνεται με την συγκέντρωση κλάσεων που αλλάζουν μαζί και απαιτεί μια ορισμένη ακρίβεια στο σχεδιασμό αφού πρέπει να προβλεφθούν οι πιθανές αλλαγές. Ακόμα, όταν ομαδοποιούνται κλάσεις που αλλάζουν μαζί στα ίδια πακέτα, τότε η επίδραση του πακέτου από την απελευθέρωση στην απελευθέρωση (κυκλοφορία πολλαπλών εκδόσεων λογισμικού) θα ελαχιστοποιηθεί.

Αυτές οι τρεις αρχές είναι αμοιβαία αποκλειόμενες. Δεν μπορούν ταυτόχρονα να ικανοποιηθούν. Αυτό οφείλεται στο γεγονός ότι κάθε αρχή ωφελεί μια διαφορετική ομάδα ανθρώπων. Η REP και η CRP καθιστούν τη ζωή εύκολη για τους ανθρώπους που ασχολούνται με την επαναχρησιμοποίηση του κώδικα, ενώ η CCP διευκολύνει τη ζωή των συντηρητών. Η CCP προσπαθεί να κάνει τα πακέτα όσο το δυνατόν μεγαλύτερα ενώ η CRP προσπαθεί να κάνει τα πακέτα πολύ μικρά. Ευτυχώς, τα πακέτα δεν έχουν ρόλο σταθερών και μπορούν να αλλάζουν δυναμικά με το χρόνο. Πράγματι, είναι στη φύση των πακέτων ο σχεδιασμός που είναι ιδανικός σήμερα να μην είναι ιδανικός στο μέλλον. Αρχικά, σε ένα έργο, η ομάδα ανάπτυξης πρέπει να δημιουργήσει τη δομή των πακέτων έτσι ώστε να

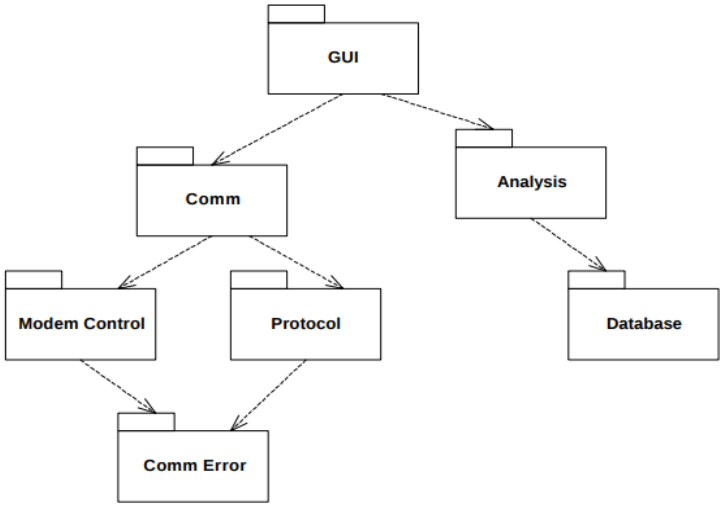
κυριαρχεί η CCP και να ενισχύεται η ανάπτυξη και η συντήρηση. Αργότερα, καθώς η αρχιτεκτονική σταθεροποιείται, η ομάδα ανάπτυξης μπορεί να επαναπροσδιορίσει τη δομή του πακέτου για να μεγιστοποιηθεί η REP και η CRP για τους εξωτερικούς χρήστες που επαναχρησιμοποιούν το λογισμικό.

Σταθερότητα ή Αρχές Σύζευξης Πακέτου ονομάζεται η δεύτερη κατηγορία αρχών στην οποία ανήκουν οι τρεις αρχές που αναφέρονται στην σύζευξη του πακέτου. Οι εφαρμογές τείνουν να είναι μεγάλα δίκτυα συνδυασμένων πακέτων. Οι κανόνες που διέπουν αυτή την αλληλεξάρτηση είναι μερικοί από τους σημαντικότερους κανόνες στην αντικειμενοστρεφή αρχιτεκτονική.

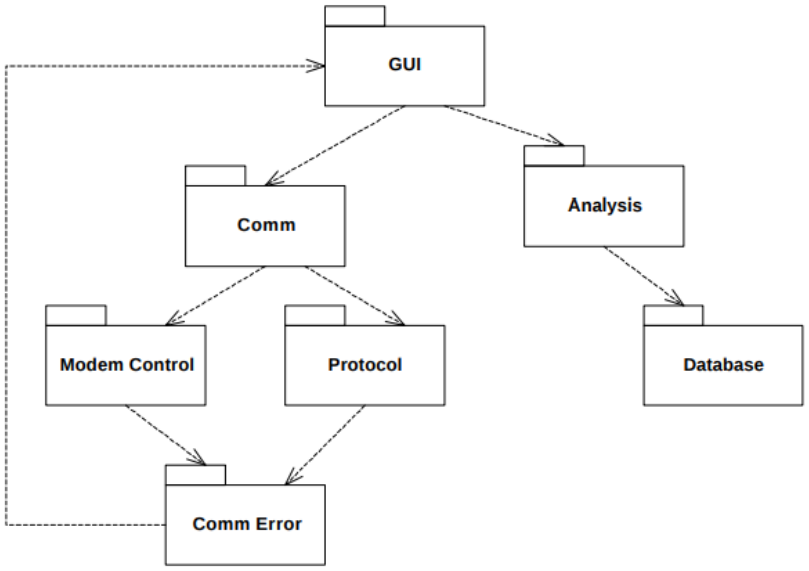
- ***Αρχή Ακυκλικών Εξαρτήσεων - (The Acyclic Dependencies Principle (ADP)) :***
  "*Το γράφημα εξάρτησης των πακέτων δεν πρέπει να έχει κύκλους.*"
  Σε έναν αναπτυξιακό κύκλο με πολλαπλούς προγραμματιστές, η συνεργασία και η ολοκλήρωση ενοτήτων κώδικα πρέπει να συμβεί σε μικρές αυξητικές εκδόσεις. Η ADP δηλώνει ότι δεν μπορεί να υπάρξουν κύκλοι στη δομή εξάρτησης και ότι όταν ολοκληρωθεί μία νέα έκδοση και δοθεί στους πελάτες, οι άλλοι προγραμματιστές μπορούν να την υιοθετήσουν και να την αξιοποιήσουν χωρίς να δημιουργηθούν προβλήματα εξάρτησης. Στην Εικόνα 1 φαίνεται ένα παράδειγμα γραφήματος εξαρτήσεων χωρίς κύκλους.

  Έστω ότι ένα μηχανικός λογισμικού ο οποίος εργάζεται στο πακέτο Comm Error αποφασίσει να εμφανίσει ένα μήνυμα στην οθόνη. Το πακέτο GUI είναι υπεύθυνο για την εμφάνιση μηνυμάτων στον χρήστη οπότε δημιουργείται ο κύκλος που φαίνεται στην Εικόνα 2.

  Στην περίπτωση που ο μηχανικός που δουλεύει στο Protocol θέλει να δημοσιεύει μία νέα έκδοση του λογισμικού, ο έλεγχος που θα πρέπει να υλοποιήσει αφορά τα πακέτα Comm Error, Modem Control, Protocol, Comm, GUI, Analysis και Database. Αυτό είναι σαφώς καταστροφικό και ο φόρτος εργασίας των μηχανικών αυξάνεται κατά ένα συνειδητό ποσό, εξαιτίας μιας μόνο μικρής εξάρτησης που ξεφεύγει από τον έλεγχο. Έχουν αναπτυχθεί διάφορες τεχνικές για να σπάνε τους κύκλους ωστόσο το θέμα αυτό δεν αποτελεί στόχο αυτής της διπλωματικής και δεν θα αναλυθεί επιπλέον.



Εικόνα 1 : Διάγραμμα Άκυκλων Εξαρτήσεων



Εικόνα 2 : Διάγραμμα Κυκλικών Εξαρτήσεων

- ***Αρχή Σταθερών Εξαρτήσεων - (Stable-Dependencies Principle (SDP)) :***

  "*Εξάρτηση από τη σταθερότητα.*"

  Τα σχέδια, είναι στη φύση τους να χρησιμοποιούνται και σε βάθος χρόνου να αλλάζουν σύμφωνα με το περιβάλλον. Έτσι, ο σχεδιασμός ενός πακέτου πρέπει να υποστηρίζει τη δυνατότητα αλλαγών. Η SDP δηλώνει ότι κάθε πακέτο που θέλει να είναι ασταθές δεν πρέπει να εξαρτάται από ένα πακέτο που είναι δύσκολο να αλλάξει. Η σταθερότητα σχετίζεται με το ποσό εργασίας που απαιτείται να γίνει μια αλλαγή σε

ένα πακέτο (πολυπλοκότητα, μέγεθος, σαφήνεια). Ένας σίγουρος τρόπος για να γίνει ένα πακέτο λογισμικού δύσκολο να αλλάξει, είναι να εξαρτάται από πολλά άλλα πακέτα λογισμικού. Ένα πακέτο με πολλές εισερχόμενες εξαρτήσεις είναι πολύ σταθερό, επειδή απαιτεί μεγάλη εργασία για να συμβιβάσει τις αλλαγές με όλα τα εξαρτώμενα πακέτα. Για το λόγο αυτό έχουν αναπτυχθεί διάφορες μετρικές που μετράνε τη σταθερότητα ενός πακέτου, οι οποίες αναλύονται στο κεφάλαιο 3 ως μετρικές σύζευξης σε επίπεδο πακέτου.

- ***Αρχή Σταθερών Αφαιρέσεων - (Stable-Abstractions Principle (SAP)) :***

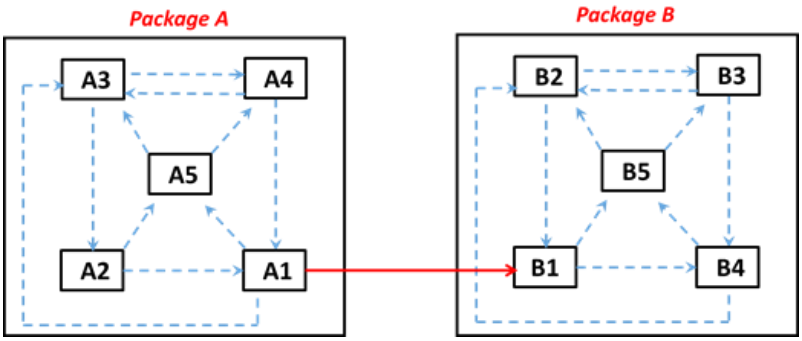
  "*Η αφηρητικότητα αυξάνεται με τη σταθερότητα.*"

  Η SAP αναφέρει ότι ένα σταθερό πακέτο θα πρέπει επίσης να είναι αφηρημένο έτσι ώστε, η σταθερότητά του να μην εμποδίζει την επέκτασή του. Αναφέρει επίσης ότι, ένα ασταθές πακέτο πρέπει να είναι συγκεκριμένο, καθώς η αστάθεια του, του επιτρέπει τον εύκολο μετασχηματισμό του συγκεκριμένου κώδικα εντός αυτού. Μπορούμε να οραματιστούμε τη δομή των πακέτων μιας εφαρμογής ως ένα σύνολο διασυνδεδεμένων πακέτων με ασταθή πακέτα στην κορυφή και σταθερά πακέτα στο κάτω μέρος. Από αυτή την άποψη, όλες οι εξαρτήσεις δείχνουν προς τα κάτω. Τα πακέτα στην κορυφή είναι ασταθή και ευέλικτα. Αλλά εκείνα που βρίσκονται στο κάτω μέρος είναι πολύ δύσκολο να αλλάξουν. Και αυτό οδηγεί στο εξής δίλημμα: Είναι θεμιτό τα πακέτα στο σχεδιασμό να είναι δύσκολο να αλλάξουν; Είναι σαφές ότι, όσο περισσότερα πακέτα είναι δύσκολο να αλλάξουν, τόσο λιγότερο ευέλικτος θα είναι ο γενικός σχεδιασμός. Ωστόσο, υπάρχει ένα κενό που μπορεί να ανιχνευτεί. Τα ιδιαίτερα σταθερά πακέτα στο κάτω μέρος του γράφου εξάρτησης μπορεί να είναι πολύ δύσκολο να αλλάξουν, αλλά σύμφωνα με την OCP δεν χρειάζεται να είναι δύσκολο να επεκταθούν. Εάν τα σταθερά πακέτα στο κάτω μέρος είναι επίσης εξαιρετικά αφηρημένα, τότε μπορούν να επεκταθούν εύκολα. Αυτό σημαίνει ότι είναι δυνατό να γίνει η σύνθεση μιας εφαρμογής από ασταθές πακέτα που είναι εύκολο να αλλάξουν και σταθερά πακέτα που είναι εύκολο να επεκταθούν. Αυτό είναι ένα καλό χαρακτηριστικό. Έτσι, η SAP είναι απλώς μια αναδιατύπωση της DIP (Αρχή Αναστροφής Εξαρτήσεων). Δηλώνει ότι τα πακέτα που είναι τα περισσότερο εξαρτώμενα (δηλαδή σταθερά) πρέπει επίσης να είναι τα πιο αφηρημένα.

2.4 Αρχιτεκτονική Συστήματος

Η τμηματικότητα (modularity) είναι μία από τις βασικές αρχές σχεδιασμού ενός λογισμικού. Προκειμένου ένα σύστημα λογισμικού να είναι τμηματοποιημένο (modular), θα πρέπει να είναι οργανωμένο σε ενότητες που είναι ιδιαίτερα συνεκτικές εσωτερικά, ενώ παράλληλα οι ενότητες αυτές να είναι όσο το δυνατόν ανεξάρτητες από άλλες μονάδες. Πιο συγκεκριμένα, η τμηματικότητα (modularity) είναι η αρχή του διαχωρισμού των καθηκόντων έτσι ώστε διαφορετικά μέρη ενός προγράμματος (ενότητες) να εκτελούν αυτά τα ξεχωριστά καθήκοντα. Στην παρούσα εργασία διερευνώνται οι μετρικές σύζευξης και συνοχής σε επίπεδο πακέτου, το οποίο θεωρείται ένα από τα πιο βασικά επίπεδα λειτουργικής αποσύνθεσης του λογισμικού σε αντικειμενοστρεφείς συστήματα, με σκοπό τη διερεύνηση του επιπέδου αποσύνθεσης που θεωρείται το καλύτερο.

Σύμφωνα με τον Van Vliet, ο σχεδιασμός λογισμικού υψηλού επιπέδου θα πρέπει να καθοδηγείται από τέσσερις βασικές αρχές, οι οποίες είναι οι εξής: παροχή αφαίρεσης, επιβολή τμηματικότητας (modularity), επιβολή κρυπτογράφησης πληροφοριών και μείωση της πολυπλοκότητας. Μεταξύ αυτών, η παρούσα εργασία επικεντρώνεται στη λειτουργικότητα του λογισμικού. Σύμφωνα με το πρότυπο ISO / IEC 25010, η τμηματικότητα (modularity) είναι ένα από τα δευτερεύοντα χαρακτηριστικά της συντηρησιμότητας. Η ακεραιότητα ορίζεται ως ο "βαθμός στον οποίο ένα σύστημα ή ένα πρόγραμμα υπολογιστή αποτελείται από διακριτά στοιχεία, έτσι ώστε η αλλαγή σε ένα στοιχείο να έχει ελάχιστη επίδραση σε άλλα συστατικά". Για να εκτιμηθεί η τμηματικότητα, πρέπει να ποσοτικοποιηθούν δύο ποιοτικές ιδιότητες, οι οποίες είναι η σύζευξη και η συνοχή. Η βελτιωμένη τμηματικότητα (modularity) επιτυγχάνεται με την προώθηση της χαμηλής σύζευξης και της υψηλής συνοχής. Η σύζευξη αντιπροσωπεύει τη δύναμη - ένταση της σύνδεσης μεταξύ των μονάδων, ενώ η συνοχή το συστατικό το οποίο συγκρατεί μια ενότητα μαζί. Σε ένα τυπικό αντικειμενοστρεφές σύστημα, οι κλάσεις (δηλαδή το πιο κεντρικό στοιχείο στον αντικειμενοστρεφή προγραμματισμό) ομαδοποιούνται σε πακέτα, με βάση τη λειτουργική τους ομοιότητα. Προκειμένου ένα σύστημα αντικειμενοστρεφούς σχεδίασης να είναι τμηματοποιημένο (modular), αναμένεται ότι οι κλάσεις του ίδιου πακέτου αλληλεπιδρούν μεταξύ τους (υψηλή συνοχή, όπως οι κλάσεις των πακέτων A και B στο Σχήμα 3), ενώ οι εξαρτήσεις μεταξύ των κλάσεων που ανήκουν σε διαφορετικά πακέτα είναι περιορισμένες (χαμηλή σύζευξη, όπως η απλή εξάρτηση του A1 προς την B1 στην Εικόνα 3).



Εικόνα 3 : Αρχιτεκτονική Συστήματος - Εξαρτήσεις Πακέτων

3. Θεωρητικό Υπόβαθρο

"What gets measured gets improved."
— Peter Drucker

3.1 Ποιότητα Λογισμικού και Μετρικές

3.1.1 Η Σημασία των Μετρικών

Μέτρηση ονομάζεται η διαδικασία κατά την οποία οι αριθμοί και τα σύμβολα συνδέονται με ιδιότητες οντοτήτων του πραγματικού κόσμου έτσι ώστε να περιγραφούν σύμφωνα με αυστηρά καθορισμένους κανόνες. Για την παρακολούθηση, διαχείριση, ποιοτική μελέτη και βελτίωση οποιουδήποτε τεχνικού έργου είναι απαραίτητη η έννοια της μέτρησης. Η εξαγωγή μέτρων είναι υποκειμενική (π.χ. πολυπλοκότητα λογισμικού). Πιο συγκεκριμένα ως μέτρο ορίζεται η ποσοτική ένδειξη αριθμού, διαστάσεων, χωρητικότητας, όγκου ενός προϊόντος ή διαδικασίας (π.χ. αριθμός λαθών σε ένα πρόγραμμα). Μέτρηση ορίζεται η διαδικασία υπολογισμού του μέτρου (π.χ. συλλογή και καταμέτρηση λαθών). Ως μετρική ορίζεται η ποσοτική εκτίμηση του βαθμού κατά τον οποίο ένα σύστημα κατέχει ένα χαρακτηριστικό (συσχετισμός λαθών με κάποιο χαρακτηριστικό, π.χ. πάνω από 100 λάθη σημαίνει κακή ποιότητα λογισμικού ενώ κάτω από 10 λάθη ορίζεται η καλή ποιότητα λογισμικού).

Μία διαδικασία μέτρησης προκειμένου να καταλήξει σε αξιόπιστα αποτελέσματα πρέπει να περιλαμβάνει τις ακόλουθες δραστηριότητες:

- Διατύπωση (Καθορισμός Μετρικών)
- Συλλογή (Συγκέντρωση Δεδομένων)
- Ανάλυση (Υπολογισμός Μετρικών)
- Ερμηνεία (Αξιολόγηση Μετρικών)
- Ανάδραση (Συστάσεις για Βελτίωση)

Οι βασικές αρχές διατύπωσης των μετρικών παραθέτονται ως εξής:

- Οι στόχοι των μετρήσεων πρέπει να καθοριστούν πριν από τη συλλογή δεδομένων (μέγεθος κώδικα, πλήθος δεδομένων, πλήθος τελεστών).
- Σαφής ορισμός των μετρικών.
- Χρήση μετρικών προσαρμοσμένων στα προϊόντα και τις διαδικασίες (αριθμός εγγραφών σε μία βάση δεδομένων, πολυπλοκότητα).

Αρχές συλλογής και ανάλυσης μετρικών:

- Όπου είναι δυνατόν, η συλλογή και ανάλυση θα πρέπει να αυτοματοποιείται.
- Χρήση αξιόπιστων στατιστικών τεχνικών για διερεύνηση εσωτερικών και εξωτερικών χαρακτηριστικών (π.χ. συσχετισμός πολυπλοκότητας και αριθμού λαθών)
- Για κάθε μετρική θα πρέπει να επιδιώκεται ο καθορισμός συγκεκριμένων κανόνων ερμηνείας.

Συνοπτικά, μια ιδανική μετρική χαρακτηρίζεται από την απλότητα και την υπολογισιμότητα της, από το πόσο εμπειρικά και διαισθητικά πειστική είναι, από τη συνέπεια και την αντικειμενικότητα, από τη συνέπεια προς τη χρήση μονάδων, την ανεξαρτησία της από τη γλώσσα προγραμματισμού και τέλος, πόσο ουσιαστικός είναι ο μηχανισμός ανάδρασης.

3.1.2 Μετρικές Ποιότητας Λογισμικού σε Επίπεδο Σχεδίασης

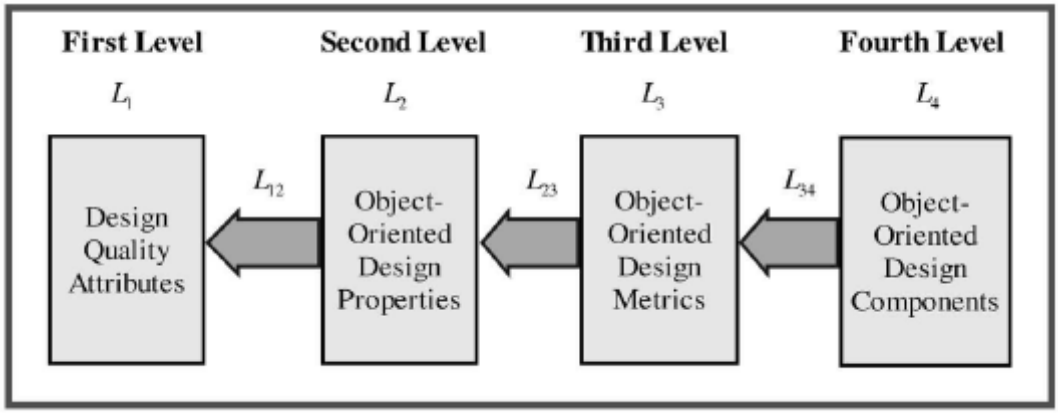
"Measure what is measurable and make measurable what is not so."
— Galileo Galilei

Σε αυτή την ενότητα γίνεται η περιγραφή ενός βελτιωμένου ιεραρχικού μοντέλου για την αξιολόγηση των χαρακτηριστικών ποιότητας υψηλού επιπέδου στην αντικειμενοστρεφή σχεδίαση. Σ' αυτό το μοντέλο, οι ιδιότητες σχεδιασμού των κλάσεων, των αντικειμένων και οι συσχετίσεις τους αξιολογούνται με τη χρήση μετρικών αντικειμενοστρεφούς σχεδίασης. Αυτό το μοντέλο συσχετίζει ιδιότητες σχεδιασμού όπως είναι η ενθυλάκωση (encapsulation), η σύζευξη (coupling) και η συνοχή (cohesion) με χαρακτηριστικά ποιότητας υψηλού επιπέδου όπως η επαναχρησιμοποίηση (reusability), η ευελιξία (flexibility), και η πολυπλοκότητα (complexity), χρησιμοποιώντας πληροφορίες από άλλες εμπειρικές μελέτες.

3.1.2.1 Ανάπτυξη Μοντέλων Ποιότητας

Τα πιο πρόσφατα μοντέλα ποιότητας αναπτύχθηκαν από τον Dromey. Ο Dromey εξετάζει μερικά προβλήματα των προηγούμενων μοντέλων ποιότητας όπως του McCall και του ISO 9126. Ως πλαίσιο εργασίας στη συγκεκριμένη περίπτωση ορίζεται μια μεθοδολογία για την ανάπτυξη των μοντέλων ποιότητας «από κάτω προς τα πάνω», παρέχοντας μια προσέγγιση που εξασφαλίζει ότι οι λεπτομέρειες χαμηλού επιπέδου είναι καλά ορισμένες και

υπολογίσιμες. Το πλαίσιο εργασίας του μοντέλου που προτείνει ο Dromey, όπως και τα προηγούμενα μοντέλα, βασίζεται στη διάσπαση των χαρακτηριστικών ποιότητας σε αντιληπτές ιδιότητες ποιότητας των τμημάτων ενός προϊόντος (απαιτήσεις, σχεδιασμός και υλοποίηση). Υπάρχουν τρία στοιχεία στο γενικό μοντέλο ποιότητας του Dromey: οι ιδιότητες προϊόντος που επηρεάζουν την ποιότητα, κάποια χαρακτηριστικά ποιότητας υψηλού επιπέδου και ένα μέσο διασύνδεσης τους. Αυτή η μεθοδολογία χρησιμοποιείται για την ανάπτυξη του Ιεραρχικού Μοντέλου για την Αξιολόγηση Ποιότητας Αντικειμενοστρεφούς Σχεδίασης (Quality Model for Object-Oriented Design, QMOOD) που επεκτείνει το γενικό μοντέλο ποιότητας του Dromey και περιλαμβάνει τα βήματα που φαίνονται στην Εικόνα 4. Παρακάτω ακολουθεί ανάλυση των βημάτων.



Εικόνα 4 : Ιεραρχικό Μοντέλο Αξιολόγησης Ποιότητας (GMOOD)

Στην Εικόνα 4 φαίνονται τα τέσσερα επίπεδα (L_1 μέχρι L_4) και τις τρεις συσχετίσεις (L_{12}, L_{23}, L_{34}) που χρησιμοποιούνται για να συνδέσουν τα τέσσερα επίπεδα στο QMOOD. Μετά τον καθορισμό των επιπέδων πρέπει να προσδιοριστούν τα ποιοτικά χαρακτηριστικά όπως είναι οι αντικειμενοστρεφείς σχεδιαστικές μετρικές, τα αντικειμενοστρεφή σχεδιαστικά συστατικά και ο καθορισμός των συσχετίσεων, που απαιτεί τη σύνδεση του χαμηλότερου επιπέδου με το αμέσως επόμενο υψηλότερο επίπεδο.

Προσδιορισμός Χαρακτηριστικών Ποιότητας Σχεδιασμού (L_1)

Τα χαρακτηριστικά ποιότητας που χρησιμοποιούνται στο Πρότυπο ISO 9126, λειτουργικότητα (functionality), αξιοπιστία (reliability), αποδοτικότητα (efficiency), χρηστικότητα (usability), συντηρησιμότητα (maintainability) και μεταφερσιμότητα (portability), επιλέχθηκαν ως το αρχικό σύνολο χαρακτηριστικών ποιότητας στο μοντέλο

QMOOD. Η επιλογή αυτή είχε ως σκοπό να οδηγήσει στην απόδειξη αν συμβάλλουν στον καθορισμό της ποιότητας και αν το σύνολο τους είναι αρκετά ευρύ ώστε να περιλαμβάνει όλες τις πτυχές της ποιότητας σχεδιασμού. Τα χαρακτηριστικά αξιοπιστία (reliability) και χρηστικότητα (usability) εξαιρέθηκαν από το σύνολο γιατί βασίζονται κυρίως στην υλοποίηση και όχι στον σχεδιασμό. Ο όρος μεταφερσιμότητα (portability) είναι πιο κατάλληλος για το πλαίσιο της εφαρμογής της ποιότητας του λογισμικού και αντικαταστάθηκε από τον όρο επεκτασιμότητα (extendibility), που αντικατοπτρίζει καλύτερα τα χαρακτηριστικά σχεδίασης. Επίσης, ο όρος συντηρησιμότητα (maintainability) προϋποθέτει την ύπαρξη ενός προϊόντος λογισμικού και αντικαταστάθηκε από τον όρο «κατανόηση» (understandability), διότι ο νέος όρος επικεντρώνεται καλύτερα στα χαρακτηριστικά της σχεδίασης.

Ένας σημαντικός λόγος υιοθέτησης της αντικειμενοστρεφούς σχεδίασης είναι η ικανότητα σχεδιασμού συστημάτων λογισμικού που χαρακτηρίζονται από αξιοπιστία, προσαρμοστικότητα και ευελιξία, εντός μικρού χρονικού διαστήματος. Ένας τρόπος για την επίτευξη αυτού του στόχου ήταν η επαναχρησιμοποίηση σε όλα τα επίπεδα της σχεδίασης. Αυτό δικαιολογεί την ένταξη της «επαναχρησιμοποίησης» ως ένα πολύ σημαντικό χαρακτηριστικό ποιότητας στον αντικειμενοστρεφή σχεδιασμό.

Η «ευελιξία» (flexibility) των συστημάτων λογισμικού αποτελεί επίσης ένα σημαντικό χαρακτηριστικό του επιπέδου ανάπτυξης και του επιπέδου τελικού-χρήστη (προϊόν που παραδίδεται στον πελάτη). Για να μπορέσει το χαρακτηριστικό αυτό να αποτελέσει μέρος των χαρακτηριστικών ποιότητας πρέπει να καθοριστεί στη φάση σχεδιασμού με την ενσωμάτωση των αρχιτεκτονικών λογισμικού. Αποφασίστηκε λοιπόν να συμπεριληφθεί το χαρακτηριστικό «ευελιξία» ως ένα από τα χαρακτηριστικά ποιότητας στο μοντέλο του σχεδιασμού. Μ' αυτόν τον τρόπο το αρχικό σύνολο των χαρακτηριστικών ποιότητας στο QMOOD διαμορφώνεται ως εξής : «λειτουργικότητα», «επεκτασιμότητα», «ικανότητα κατανόησης», «αποτελεσματικότητα», «επαναχρησιμοποίηση» και «ευελιξία».

Προκύπτει ότι το σύνολο αυτών των χαρακτηριστικών είναι αρκετά ευρύ για να επιτρέπει τον προσδιορισμό των επιθυμητών χαρακτηριστικών αντικειμενοστρεφών συστημάτων. Το σύνολο των χαρακτηριστικών δεν είναι απόλυτο, μπορεί εύκολα να αλλάξει προκειμένου να αντιμετωπίσει διαφορετικούς στόχους και σκοπούς. Τα χαρακτηριστικά ποιότητας είναι αφηρημένες έννοιες και δεν είναι άμεσα παρατηρήσιμες. Συμπληρωματικά,

δεν υπάρχουν καθολικά αποδεκτοί ορισμοί για τα χαρακτηριστικά ποιότητας του QMOOD. Στον Πίνακα 1, παρατίθενται οι ορισμοί για κάθε ένα από τα χαρακτηριστικά ποιότητας.

Πίνακας 1 : Ορισμοί Χαρακτηριστικών Ποιότητας

Χαρακτηριστικά Ποιότητας	Ορισμός
Reusability (Επαναχρησιμοποίηση)	Αντανακλά την παρουσία αντικειμενοστρεφών χαρακτηριστικών σχεδιασμού τα οποία επιτρέπουν σε ένα σχεδιασμό να επαναλαμβάνεται σε ένα νέο πρόβλημα χωρίς σημαντική προσπάθεια.
Flexibility (Ευελιξία)	Χαρακτηριστικά που επιτρέπουν την ενσωμάτωση αλλαγών σε ένα σχεδιασμό. Η ικανότητα ενός σχεδιασμού που πρέπει να προσαρμόζεται για να παρέχει λειτουργικά σχετικές ικανότητες.
Understandability (Ικανότητα Κατανόησης)	Οι ιδιότητες του σχεδιασμού που επιτρέπουν εύκολα τη μάθηση και την κατανόηση. Αυτό σχετίζεται άμεσα με την πολυπλοκότητα της δομής του σχεδιασμού.
Functionality (Λειτουργικότητα)	Οι ευθύνες αποδίδονται στις κλάσεις ενός σχεδιασμού, οι οποίες διατίθενται από τις κλάσεις μέσω των δημοσίων διασυνδέσεων τους.
Extendibility (Επεκτασιμότητα)	Αναφέρεται στην παρουσία και τη χρήση των ιδιοτήτων σε έναν υπάρχοντα σχεδιασμό, που επιτρέπουν την ενσωμάτωση νέων απαιτήσεων στο σχεδιασμό.
Effectiveness (Αποτελεσματικότητα)	Αναφέρεται στην ικανότητα ενός σχεδιασμού να πετύχει την επιθυμητή λειτουργικότητα και τη συμπεριφορά χρησιμοποιώντας αντικειμενοστρεφή έννοιες και τεχνικές σχεδιασμού.

Προσδιορισμός των Ιδιοτήτων Αντικειμενοστρεφούς Σχεδιασμού (L_2)

Οι ιδιότητες σχεδιασμού είναι αντιληπτές και μπορούν άμεσα να αξιολογηθούν εξετάζοντας την εσωτερική και εξωτερική δομή, τις σχέσεις, τη λειτουργικότητα των συστατικών σχεδιασμού, των χαρακτηριστικών, των μεθόδων και των κλάσεων. Η εκτίμηση του ορισμού κλάσης για τις εξωτερικές σχέσεις (τύπος κληρονομικότητας) με τις άλλες κλάσεις και η εξέταση των εσωτερικών συστατικών, χαρακτηριστικών και μεθόδων του, αποκαλύπτει σημαντικές πληροφορίες οι οποίες περιγράφουν τα δομικά και λειτουργικά χαρακτηριστικά της κλάσης και των αντικειμένων της.

Οι ιδιότητες σχεδιασμού της αφαίρεσης (abstraction), ενσωμάτωσης (encapsulation), σύζευξης (coupling), συνοχής (cohesion), πολυπλοκότητας (complexity), και του μεγέθους σχεδίου (design size) χρησιμοποιούνται ως ιδιότητες που αντιπροσωπεύουν τα χαρακτηριστικά ποιότητας σχεδιασμού σε δομημένο και αντικειμενοστρεφές περιβάλλον. Η μετάδοση των μηνυμάτων (messaging), η σύνθεση (aggregation), η κληρονομικότητα (inheritance), ο πολυμορφισμός (polymorphism) καθώς και οι ιεραρχίες των κλάσεων (hierarchies) αντιπροσωπεύουν νέες σχεδιαστικές έννοιες που παρουσιάζονται στο αντικειμενοστρεφές μοντέλο και είναι ζωτικής σημασίας για την ποιότητα και τον αντικειμενοστρεφή σχεδιασμό. Η αρχική έκδοση του QMOOD περιλαμβάνει και τα δύο σύνολα των ιδιοτήτων όπως ορίζονται στον Πίνακα 2.

Πίνακας 2 : Ορισμοί Ιδιοτήτων Σχεδιασμού

Ιδιότητες Σχεδιασμού	Ορισμός
Design Size (Μέγεθος Σχεδιασμού)	Ένα μέτρο για τον αριθμό των κλάσεων που συμμετέχουν στο σχεδιασμό.
Hierarchies (Ιεραρχίες)	Οι ιεραρχίες χρησιμοποιούνται για να αντιπροσωπεύουν διαφορετικές γενικευμένες - εξειδικευμένες έννοιες σε ένα σχεδιασμό. Είναι μία μέτρηση του αριθμού των μη-κληρονομούντων κλάσεων που έχουν παιδιά σε ένα σχεδιασμό.

Abstraction (Αφαίρεση)	Ένα μέτρο για την γενικευμένη - εξειδικευμένη πτυχή του σχεδιασμού. Οι κλάσεις του σχεδιασμού που έχουν έναν ή περισσότερους απογόνους εμφανίζουν αυτή την ιδιότητα της αφαίρεσης.
Encapsulation (Ενσωμάτωση)	Ορίζεται ως η ενσωμάτωση δεδομένων και συμπεριφοράς σε ένα ενιαίο κατασκεύασμα. Στον αντικειμενοστρεφή σχεδιασμό η ιδιότητα αναφέρεται συγκεκριμένα στο σχεδιασμό των κλάσεων που εμποδίζει την πρόσβαση στις δηλώσεις των χαρακτηριστικών ορίζοντας τα ως ιδιωτικά (private), προστατεύοντας έτσι την εσωτερική αναπαράσταση των αντικειμένων.
Coupling (Σύζευξη)	Καθορίζει την αλληλεξάρτηση ενός αντικειμένου με άλλα αντικείμενα σε ένα σχεδιασμό. Είναι ένα μέτρο για τον αριθμό των άλλων αντικειμένων που θα πρέπει να είναι προσβάσιμα από ένα άλλο αντικείμενο προκειμένου το εν λόγω αντικείμενο να λειτουργήσει σωστά.
Cohesion (Συνοχή)	Αξιολογεί τη συνάφεια των μεθόδων και των ιδιοτήτων σε μια κλάση ή μεταξύ μιας ομάδας κλάσεων. Ισχυρή επικάλυψη των παραμέτρων της μεθόδου και των τύπων του χαρακτηριστικού είναι μια ένδειξη ισχυρής συνοχής.
Composition (Σύνθεση)	Μετρά τις «μέρος-του", "έχει", "αποτελείται από", ή "μέρος-όλου» σχέσεις, οι οποίες είναι οι σχέσεις συνάθροισης σε έναν αντικειμενοστρεφή σχεδιασμό.

Inheritance (Κληρονομικότητα)	Ένα μέτρο της "IS-A" σχέσης μεταξύ των κλάσεων. Αυτή η σχέση σχετίζεται με το επίπεδο ένθεσης των κλάσεων σε μια ιεραρχία κληρονομικότητας.
Polymorphism (Πολυμορφισμός)	Η ικανότητα αντικατάστασης των αντικειμένων των οποίων οι διασυνδέσεις τους ταιριάζουν σε ένα άλλο, κατά το χρόνο εκτέλεσης. Είναι ένα μέτρο των υπηρεσιών που καθορίζεται δυναμικά κατά το χρόνο εκτέλεσης σε ένα αντικείμενο.
Messaging (Μετάδοση Μηνυμάτων)	Μία μέτρηση του αριθμού των δημόσιων μεθόδων που είναι διαθέσιμες ως υπηρεσίες σε άλλες κλάσεις. Είναι ένα μέτρο των υπηρεσιών που παρέχει μία κλάση.
Complexity (Πολυπλοκότητα)	Ένα μέτρο του βαθμού δυσκολίας στην κατανόηση της εσωτερικής και εξωτερικής δομής των κλάσεων και των σχέσεων.

Προσδιορισμός των Μετρικών Αντικειμενοστρεφούς Σχεδιασμού (L_3)

Όλες οι ιδιότητες που προσδιορίζονται στο μοντέλο QMOOD αντιπροσωπεύουν μια ιδιότητα ή ένα χαρακτηριστικό σχεδιασμού που ορίζεται χρησιμοποιώντας καλά ορισμένες μετρικές κατά τη διάρκεια της φάσης σχεδίασης.

Για τον αντικειμενοστρεφή σχεδιασμό, αυτή η πληροφορία θα πρέπει να περιλαμβάνει τον ορισμό των κλάσεων, τις ιεραρχίες των κλάσεων, και τις δηλώσεις των μελών των κλάσεων. Οι έρευνες για τις ήδη υπάρχουσες μετρικές αποκαλύπτουν ότι, υπάρχουν πολλές μετρικές που μπορούν να διαμορφωθούν και να χρησιμοποιηθούν στην αξιολόγηση μερικών ιδιοτήτων σχεδιασμού, όπως η αφαίρεση, η κληρονομικότητα και η ανταλλαγή μηνυμάτων. Ωστόσο, υπάρχουν πολλές άλλες ιδιότητες σχεδιασμού, όπως είναι η ενσωμάτωση και η σύνθεση, για τις οποίες δεν υπάρχουν αντικειμενοστρεφείς μετρικές. Ακόμη, ενώ οι μετρικές για την αξιολόγηση της πολυπλοκότητας, της συνοχής και της σύζευξης έχουν ήδη οριστεί, αυτές οι μετρικές απαιτούν σχεδόν ολοκληρωμένη υλοποίηση κλάσεων πριν μπορέσουν να υπολογιστούν, και γι’ αυτό δεν μπορούν να χρησιμοποιηθούν

στο QMOOD. Αυτό οδηγεί στον προσδιορισμό πέντε νέων μετρικών, τη μετρική Πρόσβασης Δεδομένων (Data Access Metric - DAM), τη μετρική Άμεσης Σύζευξης (Direct Class Coupling - DCC), τη μετρική Συνοχής μεταξύ Μεθόδων μιας Κλάσης (Cohesion Among Methods of Class - CAM), τη μετρική Μέτρηση της Συσσώρευσης (Measure of Aggregation - MOA), και τη μετρική Μέτρηση της Λειτουργικής Αφαίρεσης (Measure of Functional Abstraction - MFA). Οι πέντε αυτές μετρικές μπορούν να υπολογιστούν μόνο από τις πληροφορίες σχεδιασμού. Το σύνολο αυτών των μετρικών που χρησιμοποιείται στο QMOOD περιγράφεται στον Πίνακα 3.

Πίνακας 3 : Περιγραφή Μετρικών Σχεδιασμού

Μετρική	Όνομα	Περιγραφή
DSC	Design Size in Classes	Είναι μια μετρική που μετράει το συνολικό αριθμό κλάσεων στο σχεδιασμό.
NOH	Number of Hierarchies	Είναι μια μετρική που μετράει τον αριθμό των ιεραρχιών μιας κλάσης στο σχεδιασμό.
ANA	Average Number of Ancestors	Η μετρική αυτή τιμή, υποδηλώνει το μέσο αριθμό των κλάσεων από τις οποίες μια κλάση κληρονομεί πληροφορίες. Υπολογίζεται με τον προσδιορισμό του αριθμού των κλάσεων κατά μήκος όλων των διαδρομών από την κλάση(εις) ρίζα σε όλες τις κλάσεις, σε μια δομή κληρονομικότητας.
DAM	Data Access Metric	Αυτή η μετρική είναι ο λόγος του αριθμού των ιδιωτικών (προστατευμένων) χαρακτηριστικών προς τον συνολικό αριθμό των χαρακτηριστικών που δηλώθηκαν στην κλάση. Μια υψηλή τιμή για τη DAM είναι επιθυμητή. (Κλίμακα 0 - 1)

DCC	Direct Class Coupling	Είναι μια μετρική που μετράει το πλήθος των διαφορετικών κλάσεων που είναι άμεσα συνδεδεμένες με μια άλλη κλάση. Περιλαμβάνει κλάσεις που σχετίζονται απευθείας μέσω των δηλώσεων χαρακτηριστικών και των παραμέτρων στις μεθόδους.
CAM	Cohesion Among Methods of Class	Η μετρική αυτή υπολογίζει την σχετικότητα μεταξύ των μεθόδων μιας κλάσης η οποία βασίζεται στον κατάλογο παραμέτρων των μεθόδων. Πιο συγκεκριμένα, υπολογίζεται χρησιμοποιώντας το άθροισμα της τομής των παραμέτρων μιας μεθόδου με το μέγιστο ανεξάρτητο σύνολο όλων των τύπων παραμέτρων στην κλάση. Η τιμή της προτιμάται κοντά στο 1. (Κλίμακα 0 - 1)
MOA	Measure of Aggregation	Αυτή η μετρική μέτρα την έκταση της «μέρος-όλου» σχέσης, πραγματοποιώντας την με τη χρήση των ιδιοτήτων. Υπολογίζει τον αριθμό των δηλωμένων δεδομένων των οποίων οι τύποι τους ορίζονται από το χρήστη στις κλάσεις.
MFA	Measure of Functional Abstraction	Αυτή η μετρική είναι ο λόγος του αριθμού των μεθόδων που κληρονομούνται από μια κλάση προς τον συνολικό αριθμό των μεθόδων που είναι προσβάσιμες από μεθόδους μέλη της κλάσης. (Κλίμακα 0 - 1)
NOP	Number of Polymorphic Methods	Είναι μια μετρική που μετράει τον αριθμό των μεθόδων που μπορούν να παρουσιάζουν πολυμορφική συμπεριφορά. Τέτοιες μέθοδοι στην C++ λέγονται virtual (εικονικοί μέθοδοι).

CIS	Class Interface Size	Αυτή η μετρική μετράει τον αριθμό των δημόσιων μεθόδων σε μια κλάση.
NOM	Number of Methods	Αυτή η μετρική μετράει το πλήθος των μεθόδων που υπάρχουν σε μια κλάση.

Προσδιορισμός Συστατικών Αντικειμενοστρεφούς Σχεδιασμού (L_4)

Τα συστατικά σχεδιασμού που είναι αναγνωρίσιμα και προσδιορίζουν την αρχιτεκτονική της αντικειμενοστρεφούς σχεδίασης είναι τα αντικείμενα, οι κλάσεις καθώς και οι σχέσεις που έχουν μεταξύ τους. Τα αντικείμενα ενσωματώνουν τις δομές δεδομένων που αντιπροσωπεύουν τις ιδιότητες της κλάσης. Ένα σύνολο από λειτουργίες (μέθοδοι) που ορίζονται στις κλάσεις μπορούν να διαχειριστούν τα δεδομένα που είναι ενσωματωμένα στο αντικείμενο. Η ποιότητα του αντικειμένου καθορίζεται από τα συστατικά του, δηλαδή ιδιότητες, μεθόδους, και άλλα αντικείμενα (σύνθεση). Άλλο συστατικό που μπορεί να προσδιοριστεί είναι οι γενικές - ειδικές δομές ή οι ιεραρχίες των κλάσεων που οργανώνουν τις κλάσεις που σχετίζονται. Επιπλέον, ένα σύνολο συστατικών που μπορεί να βοηθήσει στην ανάλυση, υλοποίηση και αναπαράσταση ενός αντικειμενοστρεφούς σχεδιασμού πρέπει να περιλαμβάνει ιδιότητες, μεθόδους, αντικείμενα (κλάσεων), σχέσεις και ιεραρχίες κλάσεων.

Γενικά, όλες οι γλώσσες αντικειμενοστραφούς προγραμματισμού παρέχουν συντακτική δομή για τα θεμελιώδη συστατικά σχεδίασης. Το γεγονός ότι οι αντικειμενοστρεφείς γλώσσες προγραμματισμού (C++, Java, C#, Python) χρησιμοποιούνται για τη σχεδιαστική αναπαράσταση, έχει ως ακόλουθο η ποιότητα σχεδιασμού να μπορεί εύκολα να πραγματοποιηθεί με την αξιολόγηση αυτών των αυτόματα ανιχνεύσιμων συστατικών.

Αναγνώριση Ιδιοτήτων Ποιότητας για κάθε Συστατικό

Όταν έχει οριστεί το σύνολο των ιδιοτήτων ποιότητας για κάθε συστατικό σχεδίασης τότε αυτό θεωρείται ότι είναι μια εμπειρική διαδικασία. Η διαδικασία καθοδηγείται από ένα σύνολο ερωτήσεων, όπως: «Τι ρόλο παίζει το συστατικό στη σχεδίαση;», «Ποια είναι η

σημασία των συστατικών στη σχεδίαση;», «Ποιες οι διαφορετικές μορφές που έχει το συστατικό;».

Οι πληροφορίες αυτές είναι απαραίτητες για την αξιολόγηση των ιδιοτήτων και πρέπει να είναι διαθέσιμες κατά τη διάρκεια της φάσης σχεδιασμού. Επίσης πρέπει να υπάρχουν πολλές πληροφορίες στην αναπαράσταση των συστατικών για την αναγνώριση και την αξιολόγηση των ιδιοτήτων ποιότητας, με σαφή και ακριβή τρόπο.

Τα χαρακτηριστικά είναι τα θεμελιώδη στοιχεία στον προσδιορισμό ενός αντικειμένου και η δήλωσή τους υποστηρίζεται άμεσα στις αντικειμενοστρεφείς γλώσσες προγραμματισμού. Το σύνολο των ιδιοτήτων επηρεάζουν άμεσα την ποιότητα ενός αντικειμένου, και επιπλέον τη συνολική ποιότητα σχεδίασης και πρέπει να περιλαμβάνουν: όνομα, ενσωμάτωση, μέγεθος, τύπους, σχέσεις και δυνατότητα απαρίθμησης.

Οι δηλώσεις μεθόδων επίσης, υποστηρίζονται άμεσα στις γλώσσες προγραμματισμού. Οι ιδιότητες των μεθόδων επηρεάζουν την ποιότητα μιας κλάσης είτε άμεσα είτε έμμεσα, γι' αυτό και οι δηλώσεις πρέπει να περιλαμβάνουν: όνομα, ενσωμάτωση, τύπους παραμέτρων, μηχανισμούς περάσματος παραμέτρων, αριθμό παραμέτρων και ανάλυση.

Οι κλάσεις στις αντικειμενοστρεφείς γλώσσες περιγράφονται με τη συντακτική δομή για να είναι εύκολα αναγνωρίσιμες. Το σύνολο ιδιοτήτων που μπορεί να επηρεάσουν τη συνολική ποιότητα σχεδιασμού περιλαμβάνουν: όνομα, τύπους κληρονομικότητας και ενσωμάτωσης, πλήθος των γονέων, πλήθος των παιδιών, βάθος κληρονομικότητας, μέγεθος κλάσης, πλήθος μεθόδων και ιδιοτήτων, πλήθος εσωτερικών λειτουργιών, σύζευξη και συνοχή.

Σύνδεση των Ιδιοτήτων Ποιότητας με τις Ιδιότητες Σχεδιασμού (L_{34})

Οι ιδιότητες ποιότητας αντικειμένων ταξινομούνται με βάση τις ιδιότητες σχεδιασμού που επηρεάζουν. Έτσι, παρόλο που το σύνολο των ιδιοτήτων ποιότητας των θεμελιωδών συστατικών (ιδιότητες, μέθοδοι και κλάσεις) είναι μεγάλο, είναι και ιδιαίτερα επικαλυπτόμενο. Για παράδειγμα, οι ιδιότητες, οι μέθοδοι και τα συστατικά της κλάσης έχουν ένα όνομα και αναφέρονται στις ιδιότητες ποιότητας. Όταν το όνομα αυτό είναι περιγραφικό βοηθά στην καλύτερη κατανόηση και κατά συνέπεια επηρεάζει και την πολυπλοκότητα του σχεδιασμού. Οι ιδιότητες ποιότητας, παραδείγματος χάριν η ενθυλάκωση, αναγνωρίζεται για τις ιδιότητες, μεθόδους, και κλάσεις, και είναι ίδια με τη

γενική ιδιότητα ενθυλάκωσης. Παρόμοια, οι υπόλοιπες ιδιότητες ποιότητας που έχουν μείνει και προσδιορίζονται για τις ιδιότητες, μεθόδους και κλάσεις μπορούν να ομαδοποιηθούν σε ένα μικρότερο σύνολο έντεκα θεμελιωδών ιδιοτήτων που περιγράφονται στον Πίνακα 2.

Εφαρμογή Μετρικών Σχεδιασμού στις Ιδιότητες Σχεδιασμού (L_{23})

Οι μετρικές Σχεδιαστικό Μέγεθος Κλάσεων (Design Size in Classes - DSC), και Πλήθος Ιεραρχιών (Number of Hierarchies - NOH), χρησιμοποιούνται για την αξιολόγηση των δύο ιδιοτήτων σχεδιασμού, Design Size και Hierarchies στο QMOOD. Η αφαίρεση αναφέρεται στη γενική - ειδική δομή του σχεδιασμού και αξιολογείται από τη μετρική Μέσο Πλήθος Προγόνων (Average Number of Ancestors - ANA). Ο ορισμός της ιδιότητας ενθυλάκωσης στον Πίνακα 2 αναφέρει την πρόσβαση ελέγχου του χαρακτηριστικού των δηλώσεων στην κλάση η οποία αντικατοπτρίζεται στην περιγραφή της μετρικής Πρόσβαση Δεδομένων (Data Access Metric - DAM). Η μετρική Απευθείας Σύζευξη Κλάσης (Direct Class Coupling - DCC) είναι μια μέτρηση των κλάσεων που σχετίζονται απευθείας με την κλάση και επιπλέον χρησιμοποιείται για την αξιολόγηση της σύζευξης μιας ιδιότητας σχεδιασμού. Οι μετρικές Συνοχή Μεταξύ των Μεθόδων μιας κλάσης (Cohesion Among Method of Class - CAM), Μέτρο της Συνάθροισης (Measure of Aggregation - MOA), και Μέγεθος Διεπαφής Κλάσης (Class Interface Size - CIS) χρησιμοποιούνται για τη μέτρηση των ιδιοτήτων συνοχή (Cohesion), σύνθεση (Aggregation) και ανταλλαγή μηνυμάτων (Messaging).

Η κληρονομικότητα αναφέρεται στο βαθμό της επαναχρησιμοποίησης της λειτουργικότητας (μπορεί να μετρηθεί με τη μετρική Μέτρηση Λειτουργικής Αφαιρετικότητας - Measure of Functional Abstraction, MFA) και μπορεί να επιτευχθεί με τη δημιουργία των υποκλάσεων της υπάρχουσας κλάσης. Κατά συνέπεια η μετρική MFA χρησιμοποιήθηκε για να μετρήσει την ιδιότητα της κληρονομικότητας στο QMOOD. Για τον αντικειμενοστρεφή σχεδιασμό που παρουσιάζεται στο συντακτικό της γλώσσας C++, η ιδιότητα σχεδιασμού Πολυμορφισμός είναι η μέτρηση των Εικονικών Μεθόδων μιας κλάσης και αξιολογείται από την μετρική Πλήθος Πολυμορφικών Μεθόδων (Number of Polymorphic Methods - NOP). Η μετρική Πλήθος Μεθόδων (Number of Methods - NOM) χρησιμοποιήθηκε για να μετρήσει την πολυπλοκότητα της κλάσης από τους Chidamber και Kemerer στη μετρική Σταθμισμένες Μέθοδοι ανά Κλάση (Weighted Methods Per Class - WMC). Όταν όλες οι μέθοδοι είναι εξίσου σταθμισμένες, η μετρική WMC έχει ίδια μέτρηση

με την μετρική Πλήθος Μεθόδων (Number of Methods - NOM) στην κλάση. Ο Πίνακας 4 συνοψίζει τις μετρικές σχεδιασμού που χρησιμοποιούνται για να αξιολογήσουν τις έντεκα ιδιότητες σχεδιασμού του Πίνακα 3.

Πίνακας 4 : Μετρικές Σχεδιασμού για τις Ιδιότητες Σχεδιασμού

Ιδιότητες Σχεδιασμού	Μετρικές Σχεδιασμού
Design Size	Design Size in Classes (DSC)
Hierarchies	Number of Hierarchies (NOH)
Abstraction	Average Number of Ancestors (ANA)
Encapsulation	Data Access Metric (DAM)
Coupling	Direct Class Coupling (DCC)
Cohesion	Cohesion Among Methods of Class (CAM)
Composition	Measure of Aggregation (MOA)
Inheritance	Measure of Functional Abstraction (MFA)
Polymorphism	Number of Polymorphic Methods (NOP)
Messaging	Class Interface Size (CIS)
Complexity	Number of Methods (NOM)

Σύνδεση Ιδιοτήτων Σχεδιασμού με τα Χαρακτηριστικά Ποιότητας (L_{12})

Υπάρχουν διάφορες απόψεις για το πως οι ιδιότητες σχεδιασμού μπορούν να επηρεάσουν την ποιότητα σχεδίασης, γι' αυτό το λόγο πραγματοποιήθηκε μια εκτενή ανασκόπηση δημοσιεύσεων και βιβλίων αντικειμενοστρεφούς ανάπτυξης.

Οι πληροφορίες από αυτή την ανασκόπηση δείχνουν ότι η ιδιότητα αφαίρεσης έχει σημαντική επιρροή στα παρακάτω χαρακτηριστικά ποιότητας σχεδιασμού: ευελιξία (flexibility), αποτελεσματικότητα (effectiveness), λειτουργικότητα (functionality), και επεκτασιμότητα (extendibility).

Η ενθυλάκωση (encapsulation) θεωρήθηκε κατάλληλη για την ευελιξία (flexibility), την επαναχρησιμοποίηση (reusability) και την ικανότητα κατανόησης (understandability). Ενώ η χαμηλή σύζευξη (coupling) θεωρήθηκε κατάλληλη για την επεκτασιμότητα (extendibility), την ικανότητα κατανόησης (understandability) και την επαναχρησιμοποίηση (reusability). Οι υψηλότερες τιμές της σύζευξης επηρεάζουν αρνητικά αυτά τα ποιοτικά χαρακτηριστικά.

Η συνοχή (Cohesion) αποδείχτηκε ότι έχει σημαντική επιρροή στην επαναχρησιμοποίηση και στην ικανότητα κατανόησης του σχεδιασμού. Τα αντικείμενα επικοινωνούν μέσω της ανταλλαγής μηνυμάτων και επιπλέον η απευθείας μετάδοση μηνυμάτων επηρεάζει τη λειτουργικότητα, την αποτελεσματικότητα και βοηθά στην προώθηση της επαναχρησιμοποίησης.

Η χρήση της κληρονομικότητας προωθεί την εσωτερική επαναχρησιμοποίηση, τη λειτουργικότητα, την επεκτασιμότητα και την αποτελεσματικότητα. Έχει τη δυνατότητα να επηρεάσει αρνητικά την ευελιξία και την ικανότητα κατανόησης.

Ομοίως, ενώ η προσεκτική χρήση της σύνθεσης αντικειμένων μπορεί σημαντικά να αυξήσει την εσωτερική επαναχρησιμοποίηση, τη λειτουργικότητα και την ευελιξία, η υπερβολική και λανθασμένη χρήση της μπορεί να κάνει το σχεδιασμό δύσκολο στην κατανόηση. Η χρήση της σύνθεσης μπορεί επίσης να επηρεάσει την αποτελεσματικότητα και την επεκτασιμότητα. Η χρήση του πολυμορφισμού μπορεί να αυξήσει τις ιδιότητες σχεδιασμού όπως ευελιξία, επεκτασιμότητα, αποτελεσματικότητα, λειτουργικότητα, καθώς επίσης να καταστήσει το σχεδιασμό δύσκολο στην κατανόηση. Η πολυπλοκότητα (complexity) είναι ένας δείκτης της κατανόησης του σχεδιασμού. Ως γενικός κανόνας, όσο πιο πολύπλοκος είναι ο σχεδιασμός, τόσο πιο δύσκολη είναι η κατανόησή του. Ο Πίνακας 5 δείχνει την επιρροή της κάθε ιδιότητας σχεδιασμού στα χαρακτηριστικά της ποιότητας. Το

βέλος προς τα πάνω (↑) δείχνει ότι η ιδιότητα σχεδιασμού έχει θετική επιρροή στα χαρακτηριστικά ποιότητας.

Πίνακας 5 : Σχέσεις Ιδιοτήτων Σχεδιασμού

	Reusability	Flexibility	Understandability	Functionality	Extendibility	Effectiveness
Design Size	↑			↑		
Hierarchies				↑		
Abstraction					↑	↑
Encapsulation		↑	↑			↑
Coupling						
Cohesion	↑		↑	↑		
Composition		↑				↑
Inheritance					↑	↑
Polymorphism		↑		↑	↑	↑
Messaging	↑			↑		
Complexity						

Απόδοση Βαρύτητας στη Συσχέτιση Σχεδιαστικών Ιδιοτήτων και Ποιότητας

Η σημαντικότητα της συσχέτισης για τις μεμονωμένες ιδιότητες σχεδιασμού που επηρεάζουν τα χαρακτηριστικά ποιότητας (Πίνακας 5), σταθμίζεται αναλογικά έτσι ώστε οι

υπολογίσιμες τιμές όλων των χαρακτηριστικών ποιότητας να έχουν το ίδιο εύρος. Το εύρος από το 0 μέχρι το ±1 επιλέχθηκε για να υπολογιστούν οι τιμές των χαρακτηριστικών ποιότητας. Η επιλογή αυτή έγινε διότι είναι απλή και εύκολη στην εφαρμογή. Οι αρχικές τιμές ανάθεσης βαρών ήταν +1 ή +0.5 και χρησιμοποιήθηκαν για τις θετικές επιρροές, ενώ η τιμές -1 ή -0.5 για τις αρνητικές επιρροές. Όμως, οι τιμές αυτές άλλαξαν για να εξασφαλιστεί ότι στο άθροισμα των νέων τιμών όλων των ιδιοτήτων σχεδιασμού που επηρεάζουν τα χαρακτηριστικά ποιότητας προστέθηκε το ±1. Τα αποτελέσματα φαίνονται στον Πίνακα 6.

Πίνακας 6 : Ορισμοί Χαρακτηριστικών Ποιότητας

Χαρακτηριστικά Ποιότητας	Υπολογισμός Εξίσωσης
Reusability	-0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 Design Size
Flexibility	0.25 * Encapsulation – 0.25 * Coupling + 0.5 * Composition + 0.5 * Polymorphism
Understandability	-0.33 * Abstraction + 0.33 * Encapsulation – 0.33 * Coupling + 0.33 * Cohesion – 0.33 * Polymorphism – 0.33 * Complexity – 0.33 * Design Size
Functionality	0.12 * Cohesion + 0.22 * Polymorphism + 0.22 * Messaging + 0.22 * Design Size + 0.22 * Hierarchies
Extendability	0.5 * Abstraction – 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Effectiveness	0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism

Βελτίωση και Προσαρμογή του Μοντέλου

Το μοντέλο ποιότητας QMOOD επιτρέπει να γίνονται εύκολα αλλαγές έτσι ώστε να προσαρμόζεται σε διαφορετικές αναθέσεις βαρών, καινούριους σκοπούς και στόχους. Στο χαμηλό επίπεδο, οι μετρικές που χρησιμοποιούνται για την αξιολόγηση των ιδιοτήτων

σχεδιασμού μπορεί να αλλάξουν, ή ακόμη ένα σύνολο ιδιοτήτων σχεδιασμού μπορεί να χρησιμοποιηθεί για την αξιολόγηση χαρακτηριστικών ποιότητας. Επίσης, και το σύνολο των χαρακτηριστικών ποιότητας μπορεί να υποστεί αλλαγές. Οι σχέσεις επιρροής που υπάρχουν στις ιδιότητες σχεδιασμού και στα χαρακτηριστικά ποιότητας, καθώς και οι αναθέσεις βαρών ενδέχεται να αλλάξουν προκειμένου να ανταποκρίνονται στους στόχους και στην πολιτική του κάθε οργανισμού.

3.1.3 Μετρικές Ποιότητας Λογισμικού σε Επίπεδο Κώδικα

"Not everything that counts can be counted and not everything that can be counted counts."
— William Bruce Cameron

Οι μετρικές λογισμικού αποτελούν αναπόσπαστο κομμάτι της τεχνολογίας λογισμικού. Όλο και περισσότερες εταιρείες προσδιορίζουν τις μετρικές ποιότητας ως μέρος των απαιτήσεών τους για την παραγωγή λογισμικού. Τα βιομηχανικά πρότυπα όπως το ISO 9000 και τα βιομηχανικά μοντέλα όπως το Software Engineering Institute's (SEI) και το Capability Maturity Model Integrated (CMMI) συμπεριλαμβάνουν αυτές τις μετρήσεις. Οι εταιρείες χρησιμοποιούν τις μετρικές για την καλύτερη κατανόηση, παρακολούθηση και έλεγχο των έργων λογισμικού.

Η διασφάλιση ποιότητας συνδέεται με την έννοια των μετρήσεων. Οι μετρικές εφαρμόζονται σε ένα λογισμικό προκειμένου να μετρηθούν όλα εκείνα τα χαρακτηριστικά που συμβάλλουν στην ποιότητα του.

Η ποιότητα του λογισμικού αποτελεί ζήτημα από τις πρώτες μέρες εμφάνισης του προγραμματισμού των υπολογιστών. Συνεπώς, έχουν προκύψει πολλοί ορισμοί από διαφορετικούς συγγραφείς για το τι είναι η ποιότητα λογισμικού. Οι περισσότεροι ορισμοί είναι ανακριβείς και αφηρημένοι. Σύμφωνα με το πρότυπο του ISO 8402, ποιότητα είναι το σύνολο των χαρακτηριστικών που έχει μια οντότητα που της αποδίδουν την ικανότητα να ικανοποιεί εκφρασμένες και συνεπαγόμενες ανάγκες. Το διεθνές πρότυπο του ISO 9126 ορίζει κάποια ποιοτικά χαρακτηριστικά όπως είναι η λειτουργικότητα, η αξιοπιστία, η χρηστικότητα, η αποτελεσματικότητα, η συντηρησιμότητα και η φορητότητα.

Οι μετρικές λογισμικού μπορούν να ταξινομηθούν σε τρεις κατηγορίες: μετρικές προϊόντος, μετρικές διαδικασίας και μετρικές έργου. Οι μετρικές προϊόντος σχετίζονται με το

προϊόν, για παράδειγμα τον πηγαίο κώδικα ή τις δηλώσεις ελέγχου και περιγράφουν κάποια σχεδιαστικά χαρακτηριστικά όπως το μέγεθος, την πολυπλοκότητα, την απόδοση και το επίπεδο ποιότητας. Χωρίζονται σε δύο επιπλέον κατηγορίες, τις εσωτερικές μετρικές λογισμικού και τις εξωτερικές μετρικές λογισμικού:

1. ***Οι εσωτερικές μετρικές λογισμικού*** χρησιμοποιούνται για τη μέτρηση χαρακτηριστικών του λογισμικού για τα οποία υπάρχει απτή αντίληψη για τη φυσική τους σημασία και δυνατότητα άμεσης μέτρησης. Η χρήση εσωτερικών μετρικών λογισμικού ενθαρρύνεται από όλα τα διεθνή πρότυπα. Η μέτρηση τους μπορεί να γίνει πολύ εύκολα, γρήγορα και αυτόματα. Οι εσωτερικές μετρικές χρησιμοποιούνται για τη μέτρηση χαρακτηριστικών λογισμικού όπως ο αριθμός γραμμών κώδικα, το ποσοστό των σχολίων, τα λάθη του κώδικα και ο χρόνος εκτέλεσης. Οι μετρικές αυτές μπορούν εύκολα να υπολογιστούν αλλά δεν παρέχουν πληροφόρηση σχετικά με την ποιότητα του λογισμικού καθώς δεν στηρίζονται στη λειτουργία του.
2. ***Οι εξωτερικές μετρικές λογισμικού*** είναι υψηλού επιπέδου και εφαρμόζονται στο λογισμικό όταν είναι σε λειτουργία. Αναφέρονται ως μετρικές ποιότητας και είναι, η λειτουργικότητα (functionality), η ποιότητα (quality), η πολυπλοκότητα (complexity), η αποτελεσματικότητα (efficiency), η αξιοπιστία (reliability) και η συντηρησιμότητα (maintainability). Τα αποτελέσματα της συγκεκριμένης κατηγορίας είναι εμπειρικά και βασίζονται κυρίως σε κάποιες έρευνες είτε με μορφή ερωτηματολογίων ή συνεντεύξεων για τη γνώμη που έχουν οι πελάτες, είτε με τη μορφή παρατήρησης και μελέτης της συμπεριφοράς του χρήστη. Παρέχουν δεδομένα άμεσα αξιοποιήσιμα και ερμηνεύσιμα, αφού μετρούν απευθείας τα ζητούμενα εξωτερικά ποιοτικά χαρακτηριστικά του λογισμικού. Τα αρνητικά στοιχεία των εξωτερικών μετρικών είναι ότι, πρώτον βασίζονται σε υποκειμενικά δεδομένα και επομένως τα αποτελέσματά τους είναι πιθανόν να υπόκεινται σε αμφισβητήσεις, και δεύτερων υπάρχει αδυναμία αυτοματοποίησης της διαδικασίας των εξωτερικών μετρήσεων.

Οι μετρικές διαδικασίας μπορούν να χρησιμοποιηθούν για να βελτιώσουν την ανάπτυξη και τη συντήρηση του λογισμικού. Παραδείγματα περιλαμβάνουν την αφαίρεση και διόρθωση σφαλμάτων κατά τη διάρκεια της ανάπτυξης και το χρόνο απόκρισης κατά την εκτέλεση.

Οι μετρικές έργου χρησιμοποιούνται για τον καθορισμό στρατηγικής, σχετίζονται με την τακτική εκτέλεσης του έργου και καθορίζουν τη ροή του και τις τεχνικές που θα

ακολουθηθούν. Πιο συγκεκριμένα, περιγράφουν τα χαρακτηριστικά του έργου και της εκτέλεσης. Τέτοια παραδείγματα είναι το πλήθος των προγραμματιστών, το κόστος, η παραγωγικότητα και το χρονοδιάγραμμα. Κάποιες μετρικές μπορεί να ανήκουν σε πολλές κατηγορίες. Οι μετρικές ποιότητας λογισμικού είναι υποσύνολο των μετρικών λογισμικού που εστιάζουν σε θέματα ποιότητας του προϊόντος, της διαδικασίας και του έργου. Γενικά, οι μετρικές ποιότητας σχετίζονται περισσότερο με τις μετρικές διαδικασίας και προϊόντος παρά με τις μετρικές έργου. Παρόλα αυτά, οι παράμετροι του έργου όπως είναι ο αριθμός των προγραμματιστών και οι ικανότητές τους, το χρονοδιάγραμμα, το μέγεθος του έργου, και η δομή οργάνωσης του επηρεάζουν εξίσου την ποιότητα του προϊόντος.

Η αντικειμενοστρεφής προσέγγιση για την ανάπτυξη λογισμικού έχει δημιουργήσει ένα αυξημένο ενδιαφέρον για τις μετρικές ανάπτυξης και τις μετρικές αξιολόγησης. Το αντικειμενοστρεφές λογισμικό διαφέρει από το λογισμικό που αναπτύσσεται με συμβατικές μεθόδους. Αυτό έχει ως αποτέλεσμα, οι μετρικές που αναπτύσσονται για το αντικειμενοστρεφές λογισμικό να εστιάζουν στα ειδικά χαρακτηριστικά του όπως είναι η τοπικότητα, η ενσωμάτωση, η απόκρυψη πληροφορίας και η κληρονομικότητα. Παρακάτω παρουσιάζονται βασικές αντικειμενοστρεφείς μετρικές και μετρικές για δομημένο προγραμματισμό.

Coupling between Object Classes (CBO)

Η μετρική Σύζευξη μεταξύ Κλάσεων Αντικειμένων (CBO) υπολογίζει τη σύζευξη μεταξύ των κλάσεων αντικειμένων ή διαφορετικά το πλήθος των κλάσεων από τις οποίες εξαρτάται η τρέχουσα κλάση. Μια κλάση μπορεί να έχει σύζευξη με μια άλλη στην περίπτωση που τα αντικείμενα των δύο κλάσεων αλληλεπιδρούν. Κάθε τέτοια εξάρτηση μπορεί να είναι αμφίδρομη ή μονόδρομη (οποιασδήποτε κατεύθυνσης). Θεωρείται ότι δύο κλάσεις είναι σε σύζευξη όταν οι μέθοδοι που έχουν οριστεί σε μια κλάση χρησιμοποιούν άλλες μεθόδους ή στιγμιότυπα μεταβλητών τα οποία είναι ορισμένα στη δεύτερη κλάση. Η μετρική αυτή σχετίζεται με τη δυνατότητα επαναχρησιμοποίησης μιας κλάσης. Η αυξημένη σύζευξη είναι ανεπιθύμητη, διότι η υπερβολική εξάρτηση των κλάσεων αποτρέπει την επαναχρησιμοποίηση. Συνεπώς, οι κλάσεις πρέπει να είναι όσο το δυνατόν πιο ανεξάρτητες μεταξύ τους προκειμένου να καθίσταται δυνατή η χρήση τους και σε άλλες εφαρμογές. Με αυτόν τον τρόπο γίνεται πιο εύκολη η συντήρηση, η υλοποίηση και ο έλεγχος του λογισμικού. Επίσης, οι αυξημένες τιμές της μετρικής CBO αυξάνουν την πιθανότητα

σφαλμάτων στον κώδικα. Η τιμή που πρέπει να έχει η CBO για τις διαδικασίες - μεθόδους δεν πρέπει να είναι μεγαλύτερη από το 10. Προκειμένου να μειωθεί η τιμή της σε κάποια μέθοδο, πρέπει να γίνει η διάσπασή της σε ξεχωριστές μεθόδους. Αντίστοιχα, η τιμή της CBO για τις κλάσεις δεν πρέπει να υπερβαίνει το 30. Σε περίπτωση που μια κλάση έχει πολύ μεγάλη τιμή σύζευξης τότε η κλάση αυτή θα πρέπει να διασπαστεί σε περισσότερες κλάσεις στη μελλοντική σχεδίαση του συστήματος.

Cyclomatic Complexity (CC)

Η μετρική Κυκλωματική Πολυπλοκότητα (CC) αρχικά προτάθηκε από τον McCabe με σκοπό τη μέτρηση της πολυπλοκότητας του προγράμματος λογισμικού εξετάζοντας το γράφο ροής του. Ορίζει τον αριθμό των ανεξάρτητων διαδρομών του γράφου που πρέπει να ελεγχθούν μέσω ενός αντίστοιχου αριθμού περιπτώσεων ελέγχου, που εξασφαλίζει ότι όλες οι εντολές του προγράμματος έχουν εκτελεστεί τουλάχιστον μια φορά. Η μετρική αυτή βασίζεται στο γεγονός ότι οι συνθήκες καθώς και οι δομές ελέγχου (if, for, while) αυξάνουν την πολυπλοκότητα του προγράμματος. Μπορεί να υπολογιστεί από τον παρακάτω τύπο, όπου οι κόμβοι παριστάνουν τις εντολές ενώ οι ακμές τη μεταφορά ελέγχου μεταξύ των κόμβων:

$$CC = E - N + 2P$$

Όπου E είναι το πλήθος των ακμών, N είναι το πλήθος των κόμβων και P είναι το πλήθος των συνδεδεμένων τμημάτων (υπορουτίνες, διαδικασίες).

Η Κυκλωματική Πολυπλοκότητα προσδιορίζει την πολυπλοκότητα ενός τμήματος, δηλαδή μιας μεθόδου, αντικειμενοστρεφούς κώδικα. Η μείωση της πολυπλοκότητας συνεπάγει την αύξηση των δυνατοτήτων ανάγνωσης επειδή ο κώδικας γίνεται πιο απλός, πιο κατανοητός και ευκολότερα ελέγξιμος χωρίς να απαιτείται μεγάλο πλήθος σχολίων. Επίσης υπάρχει ένας συσχετισμός της πολυπλοκότητας με τη συνοχή. Ένα τμήμα με υψηλή πολυπλοκότητα έχει χαμηλή συνοχή σε σχέση με ένα τμήμα που έχει χαμηλότερη πολυπλοκότητα. Τέλος, η μετρική αυτή σχετίζεται με τον αριθμό των γραμμών του κώδικα καθώς και με τον αριθμό των λαθών που μπορούν να εντοπιστούν σ' ένα πρόγραμμα.

Coupling Factor (CF)

Η μετρική Παράγοντας Σύζευξης (CF) είναι μια μετρική αντικειμενοστρεφούς σχεδίασης (Metrics for Object-Oriented Design M.O.O.D). Η μετρική αυτή μετρά τη σύζευξη μεταξύ των κλάσεων εξαιρώντας τις σχέσεις κληρονομικότητας. Είναι ο λόγος του μέγιστου αριθμού των συζεύξεων στην κλάση προς τον πιθανό αριθμό των μη κληρονομούντων συζεύξεων σε ένα σύστημα. Ο στόχος είναι όσο το δυνατόν χαμηλότερη τιμή της.

Comments Ratio (CR)

Η μετρική αυτή αποτελεί ένα μέτρο για το ποσοστό των σχολίων που εμφανίζεται σε όλες τις γραμμές του κώδικα. Δείχνει κατά πόσο ο προγραμματιστής σχολιάζει τη μεθοδολογία που ακολουθεί για τη συγγραφή του κώδικα. Με τη βοήθεια των σχολίων ο κώδικας διαβάζεται εύκολα και γίνεται πιο κατανοητός. Προτείνεται το ποσοστό των σχολίων να φτάνει τουλάχιστον το 5% σε σχέση με το συνολικό ποσό των γραμμών του κώδικα. Εντούτοις, το μεγάλο ποσοστό σχολίων αποτελεί έναν παράγοντα κακής σχεδίασης.

Lines of Code (LOC)

Ο πιο εύκολος τρόπος για να μετρηθεί το μέγεθος του προγράμματος είναι να μετρηθούν οι γραμμές του πηγαίου κώδικα (Source Lines Of Code - SLOC) ή οι χιλιάδες γραμμές του πηγαίου κώδικα (KLOC). Είναι η πιο παλιά και η πιο διαδεδομένη μετρική. Υπάρχουν διάφοροι τρόποι για τη μέτρηση των γραμμών του κώδικα. Το μέγεθος ενός συστήματος μπορεί να μετρηθεί είτε με βάση τον αριθμό των γραμμών των προγραμμάτων που το απαρτίζουν, είτε με βάση τον αριθμό των λειτουργιών που έχουν υλοποιηθεί σε κώδικα. Ο διαχωρισμός των γραμμών στα αρχεία κώδικα γίνεται ανάλογα με τον τύπο και τη λειτουργικότητά τους, σε κενές γραμμές, γραμμές σχολίων, μη-εκτελέσιμες γραμμές, δηλώσεις εντολών και γραμμές που δημιουργούνται από κάποιο εργαλείο. Η πιο κοινή χρήση της είναι η μέτρηση των γραμμών που δεν είναι κενές ή των γραμμών που δεν έχουν σχόλια.

Η μετρική αυτή συνήθως χρησιμοποιείται για να προβλέψει τη συνολική προσπάθεια που απαιτείται για την ανάπτυξη του προγράμματος καθώς και την απόδοση των προγραμματιστών (παραγωγικότητα). Οι γραμμές του κώδικα δεν μπορούν να μετρήσουν την πολυπλοκότητα του λογισμικού, ωστόσο σχετίζονται με αυτή. Όσο περισσότερες είναι οι γραμμές του κώδικα τόσο αυξάνεται και η πολυπλοκότητα του λογισμικού.

Υπάρχουν δύο βασικοί τύποι της μετρικής: το φυσικό και το λογικό LOC. Παρόλο που οι ορισμοί αυτών των δύο τύπων ποικίλουν, ο πιο κοινός ορισμός του φυσικού LOC είναι η καταμέτρηση των γραμμών του πηγαίου κώδικα συμπεριλαμβάνοντας και τις γραμμές σχολίων. Περιλαμβάνει επίσης και τις κενές γραμμές του κώδικα.

Το λογικό LOC επιχειρεί να μετρήσει τον αριθμό των εκτελέσιμων "δηλώσεων", αλλά οι συγκεκριμένοι ορισμοί τους συνδέονται με συγκεκριμένες γλώσσες υπολογιστών (ένα απλό λογικό μέτρο LOC για τις γλώσσες προγραμματισμού που μοιάζουν με τη C είναι ο αριθμός των ερωτηματικών που τερματίζουν μία έκφραση). Είναι πολύ πιο εύκολο να δημιουργηθούν εργαλεία που μετρούν το φυσικό LOC και οι ορισμοί των φυσικών LOC είναι πιο εύκολο να εξηγηθούν. Ωστόσο, τα φυσικά μέτρα LOC είναι ευαίσθητα σε λογικά άσχετες μορφοποιήσεις και συμβάσεις στυλ, ενώ το λογικό LOC είναι λιγότερο ευαίσθητο στις μορφοποιήσεις και τις συμβάσεις στυλ. Εντούτοις, τα μέτρα των φυσικών LOC αναφέρονται συχνά χωρίς να δίνουν τον ορισμό τους και το λογικό LOC μπορεί συχνά να διαφέρει σημαντικά από το φυσικό LOC.

Κατηγορίες της LOC είναι οι:

- *LOC (Lines of Code):* Μετρά τον συνολικό αριθμό των γραμμών που υπάρχουν στον πηγαίο κώδικα.
- *NCLOC (Non-Commented Lines of Code):* Μετρά τον συνολικό αριθμό γραμμών του πηγαίου κώδικα που δεν περιέχει σχόλια.
- *CLOC (Commented Lines of Code):* Μετρά τον συνολικό αριθμό των γραμμών του πηγαίου κώδικα που περιέχει σχόλια.
- *CD (Comment Density):* Είναι ο λόγος της ποσοτικοποίησης του όγκου των σχολίων προς το μέγεθος του πηγαίου κώδικα.
- *ES (Executable Statements):* Μετρά τον αριθμό των εκτελέσιμων εντολών που υπάρχουν στον πηγαίο κώδικα.
- *DSI (Delivered Source Instructions):* Μετρά τον αριθμό των εντολών που θα παραδοθούν στον πελάτη δηλαδή δεν περιλαμβάνει τα πρωτότυπα ή τον κώδικα ελέγχου.

LCOM - Lack Of Cohesion Of Methods

Η μετρική Έλλειψη Συνοχής Μεθόδων (LCOM) έχει προταθεί από τους S.R Chidamber και C.F Kemerer, και μετρά την έλλειψη συνοχής. Βασίζεται στο γεγονός ότι

κάθε μέθοδος που υπάρχει σε μια κλάση προσπελαύνει ένα ή και περισσότερα κοινά μέλη δεδομένων. Δύο μέθοδοι είναι συνεκτικές αν το σύνολο των δεδομένων που χρησιμοποιούν έχουν στοιχεία τα οποία είναι κοινά. Όσο περισσότερες είναι οι συνεκτικές μέθοδοι, τόσο μεγαλύτερη συνεκτικότητα υπάρχει και η τιμή της LCOM είναι χαμηλότερη. Αυτό οδηγεί στην αύξηση της πολυπλοκότητας. Σε αντίθετη περίπτωση, αν η τιμή της LCOM είναι υψηλή τότε η συνεκτικότητα είναι χαμηλή. Μ' αυτόν τον τρόπο ο σχεδιασμός της κλάσης μπορεί επιτευχθεί με βέλτιστο τρόπο μέσο της διάσπασή της σε δύο ή περισσότερες ανεξάρτητες κλάσεις.

Η συνοχή μιας μονάδας αναφέρεται στη σχέση των συστατικών των μονάδων, δείχνει πόσο καλά τα συστατικά μέρη μιας μονάδας συνδέονται μεταξύ τους. Επιθυμητό είναι να υπάρχει ισχυρή συνοχή μεταξύ των μερών. Μια μονάδα με υψηλή συνοχή εκτελεί μια βασική λειτουργία και δεν μπορεί να διασπαστεί σε δυο ξεχωριστές ενότητες εύκολα. Η υψηλή συνοχή στις μονάδες είναι περισσότερο κατανοητή, τροποποιήσιμη και διατηρήσιμη. Η έλλειψη συνοχής σε μια μονάδα μειώνει την ενθυλάκωση, αυξάνει την πολυπλοκότητα και την πιθανότητα εμφάνισης σφαλμάτων και αποδίδει επιπλέον δυσκολία στη συντήρηση και στην επαναχρησιμοποίηση του λογισμικού.

Weighted Methods per Class (WMC)

Η μετρική Σταθμισμένες Μέθοδοι ανά Κλάση (WMC) προτάθηκε πρώτη φορά από τους S.R Chidamber και C.F Kemerer και σχετίζεται με την έννοια της πολυπλοκότητας μιας κλάσης. Ορίζεται ως το άθροισμα της πολυπλοκότητας όλων των μεθόδων μιας κλάσης. Είναι ο δείκτης που δείχνει πόσος χρόνος και πόση προσπάθεια απαιτείται για την ανάπτυξη και συντήρηση μιας συγκεκριμένης κλάσης. Οι υψηλές τιμές αυτής της μετρικής δείχνουν ότι απαιτείται μεγαλύτερη προσπάθεια για την ανάπτυξη και τη συντήρηση της κλάσης. Επίσης, αν ο αριθμός των μεθόδων σε μια κλάση είναι αρκετά μεγάλος πιθανό είναι να δημιουργηθούν προβλήματα επαναχρησιμοποίησης της κλάσης.

Υπάρχουν 2 υποκατηγορίες αυτής της μετρικής :

- *Weighted Methods Per Class 1 (WMPC1),* που αντιπροσωπεύει το άθροισμα της πολυπλοκότητας όλων των μεθόδων μιας κλάσης, όπου η κάθε κλάση σταθμίζεται από την κυκλωματική πολυπλοκότητά της.
- *Weighted Methods Per Class 2 (WMPC2)*, που μετράει την πολυπλοκότητα της κλάσης, υποθέτοντας ότι μια κλάση που έχει περισσότερες μεθόδους σε σχέση με

κάποια άλλη που έχει λιγότερες μεθόδους, είναι περισσότερο πολύπλοκη. Το ίδιο ισχύει και για την περίπτωση όπου μία μέθοδος έχει περισσότερες παραμέτρους σε σχέση με κάποια άλλη μέθοδο. Αυτή με τις περισσότερες παραμέτρους θεωρείται πιο πολύπλοκη.

Number of Children (NOC)

Η μετρική Πλήθος Παιδιών (NOC) αντιπροσωπεύει το πλήθος των κλάσεων που κληρονομούν άμεσα από την τρέχουσα. Υψηλό NOC σημαίνει υψηλή επαναχρησιμοποίηση κώδικα αλλά πιθανόν να υποδεικνύει εννοιολογικά εσφαλμένη χρήση της κληρονομικότητας. Δεν υπάρχει κάποιο γενικώς αποδεκτό βέλτιστο NOC, αφού αυτό εξαρτάται από την εκάστοτε κλάση, αλλά εν γένει οι κλάσεις οι ευρισκόμενες υψηλότερα στην ιεραρχία κληρονομικότητας είναι θεμιτό να έχουν υψηλότερο NOC από όσες τοποθετούνται χαμηλότερα αφού είναι λιγότερο εξειδικευμένες.

Fan-out (FO), Fan-in (FI)

Οι πιο απλές μετρικές σύζευξης είναι η fan-in και η fan-out. Για μια μονάδα λογισμικού, το fan-in είναι ο αριθμός των μονάδων που χρησιμοποιούν τη συγκεκριμένη μονάδα. Ενώ το fan-out είναι το αντίστροφο δηλαδή, ο αριθμός των μονάδων λογισμικού που χρησιμοποιεί η συγκεκριμένη μονάδα. Ο στόχος είναι η ελαχιστοποίηση του αριθμού μονάδων με υψηλό fan-out. Μια μονάδα όταν ελέγχει μεγάλο αριθμό άλλων μονάδων, έχει ως αποτέλεσμα να εκτελεί πολλές εργασίες επομένως έχει κακή συνεκτικότητα. Αυτό οδηγεί στη δυσκολία εντοπισμού σφαλμάτων, στην επαναχρησιμοποίηση και στην τροποποίηση. Επίσης, πρέπει να γίνει μεγιστοποίηση του αριθμού των μονάδων που έχουν υψηλό fan-in διότι είναι προτιμότερο μια λειτουργία που εκτελείται συχνά να βρίσκεται σε μία μονάδα παρά να υλοποιείται σε πολλές άλλες.

3.2 Αναδόμηση (Refactoring) και Κακές Οσμές (Bad smell)

"When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous."
— Martin Fowler

3.2.1 Συντήρηση λογισμικού

Η διαδικασία ανάπτυξης λογισμικού μεγάλης κλίμακας είναι μία εξαιρετικά πολύπλοκη διαδικασία στην οποία συμμετέχουν πολλά άτομα με ετερογενές υπόβαθρο. Οι συνεχείς αλλαγές απαιτήσεων από την πλευρά των πελατών καθώς και τα πιεστικά χρονικά όρια οδηγούν τους προγραμματιστές -πολλές φορές- σε προγραμματιστικές λύσεις που δεν υπακούν στις βασικές αρχές της αντικειμενοστρεφούς σχεδίασης λογισμικού. Το ίδιο αποτέλεσμα επιφέρει και η αύξηση του μεγέθους της εφαρμογής και συνεπώς και των γραμμών κώδικα. Οι πρόχειρες αυτές προσθήκες πολύ συχνά οδηγούν σε αλυσιδωτές αλλαγές και σε άλλα σημεία του λογισμικού. Όλα αυτά έχουν ως επακόλουθο η ποιότητα και η δομή του γραφομένου κώδικα να μειώνεται συνεχώς.

Η μείωση της ποιότητας σχεδίασης του κώδικα έχει ως συνέπεια την αυξανόμενη δυσκολία ενσωμάτωσης μελλοντικών αλλαγών στη λειτουργία του συστήματος οι οποίες προέρχονται από τις συνεχείς αλλαγές στις απαιτήσεις του πελάτη. Η δυσκολία αυτή προέρχεται από τη καταστρατήγηση τόσο των ευρετικών κανόνων (Heuristics) όσο και των προτύπων σχεδίασης (Design Patterns) τα οποία όταν εφαρμόζονται εγγυώνται έναν καθορισμένο αλγοριθμικό και εύκολο τρόπο ενσωμάτωσης των αλλαγών στη λειτουργία του λογισμικού. Αυτός ο τρόπος ενσωμάτωσης αφήνει αναλλοίωτη την έως τότε σχεδίαση του συστήματος.

Όσο χειρότερη είναι η σχεδίαση του κώδικα, τόσο περισσότερος κόπος, χρόνος και συνεπώς πολύτιμες ανθρωποώρες απαιτούνται για να ενσωματωθεί μία νέα απαίτηση. Επειδή ο χρόνος είναι ιδιαίτερα πολύτιμος και πολλές φορές μεταφράζεται σε χρήμα, οι νέες αλλαγές και προσθήκες συνήθως γίνονται με πρόχειρο τρόπο και η μείωση της ποιότητας της σχεδίασης συνεχίζεται. Επομένως, τίθενται αναγκαίο να οριστεί κάποιος τρόπος ο οποίος να καθορίζει όσο το δυνατόν αυστηρότερα τη μεθοδολογία και τη διαδικασία σχεδίασης του συστήματος προκειμένου από την αρχή του κύκλου ζωής, το λογισμικό, να είναι σε θέση να υποστηρίξει τροποποιήσεις ή νέες προσθήκες δίχως να επηρεάζονται οι ήδη υπάρχουσες λειτουργίες. Όπως γίνεται αντιληπτό, οι δομικές αλλαγές σε υπάρχοντα συστήματα είναι πολύπλοκες και απαιτείται ένας αυστηρά καθορισμένος τρόπος εντοπισμού των προβλημάτων και της εφαρμογής των αλλαγών.

Στον κύκλο ζωής ενός προϊόντος λογισμικού, το κόστος της σχεδίασης είναι σημαντικό και εξαρτάται κυρίως από το μέγεθος του συστήματος και κατ' επέκταση από το πλήθος και το είδος των απαιτήσεων. Ωστόσο, αυτό που καθιστά τη σχεδίαση ιδιαιτέρως

σημαντική από πλευράς κόστους, είναι το ότι καθορίζει σε εξαιρετικά μεγάλο βαθμό, την ευκολία με την οποία μπορούν να πραγματοποιηθούν αλλαγές στο σύστημα. Λόγω του υψηλού κόστους της συντήρησης ενός μεγάλου έργου λογισμικού, συνήθως πολλαπλάσιο του κόστους ανάπτυξής του, κρίνεται αναγκαίο να ληφθεί πρόνοια κατά τη διάρκεια της σχεδίασης ώστε το σύστημα να επεκτείνεται και να τροποποιείται με μικρή προσπάθεια. Όσο περισσότερη πρόνοια ληφθεί τόσο οικονομικότερη και ταχύτατη καθίσταται η εξέλιξη του λογισμικού. Ως επακόλουθο, ένα καλά σχεδιασμένο λογισμικό αντέχει περισσότερο στο χρόνο και συνεπώς ο κύκλος ζωής του έχει μεγαλύτερη διάρκεια.

Ένα καλά σχεδιασμένο σύστημα θα πρέπει να είναι ευέλικτο στις επερχόμενες αλλαγές. Να μπορεί να τροποποιηθεί ανάλογα, με την ελάχιστη δυνατή προσπάθεια και με τις μικρότερες δυνατές συνέπειες για τις υπόλοιπες λειτουργίες του. Η σχεδίαση λοιπόν του συστήματος, είναι αυτή που εξασφαλίζει την ποιότητα αναφορικά με την εύκολη συντήρηση.

Ωστόσο, η σχεδίαση των αντικειμενοστρεφών συστημάτων λογισμικού είναι δυνατόν να παρουσιάσει συμπτώματα κακής ποιότητας σχεδίασης (poorly/bad design). Στα προβλήματα αυτά, οι ερευνητές αναφέρονται με διάφορα ονόματα όπως κακές οσμές (bad smells) [1], ελαττώματα σχεδίασης (design flaws) [2], έλλειψη συμμόρφωσης με αρχές σχεδίασης (design principles) [3], παραβίαση 8 ευρετικών κανόνων (design heuristics) [4], απουσία προτύπων σχεδίασης (design patterns) [5], καθώς επίσης και εφαρμογή αντί-προτύπων σχεδίασης (anti patterns) [6]. Ο εντοπισμός και η εξάλειψη των προβλημάτων αποτελεί μείζονος σημασία εφόσον οδηγεί σε πηγαίο κώδικα του συστήματος που είναι πιο εύκολος στην κατανόηση, την τροποποίηση, την επαναχρησιμοποίηση και τον έλεγχο.

Οι αρχές, τα πρότυπα αλλά και οι ευρετικοί κανόνες σχεδίασης, έχουν ως στόχο την αξιοποίηση των πλεονεκτημάτων που προσφέρει το αντικειμενοστρεφές μοντέλο, στην περίπτωση ανάπτυξης συστημάτων που πρόκειται να εξελιχθούν λόγω αλλαγών στις απαιτήσεις. Συνεπώς, η παραβίαση αυτών, οδηγεί με βεβαιότητα στα συμπτώματα κακής ποιότητας της σχεδίασης του συστήματος λογισμικού. Ωστόσο, η τήρηση αυτών δεν αποτελεί πανάκεια και δεν θα πρέπει να εφαρμόζονται διαρκώς και χωρίς λόγο παρά μόνο όταν παρατηρηθούν κάποια από τα συμπτώματα.

3.2.2 Κακές Οσμές (Bad smell)

“Code never lies, comments sometimes do.”
— Ron Jeffries

Οι «κακές οσμές» (bad smells) αποτελούν συμπτώματα κακής σχεδίασης τμήματος κώδικα και οφείλουν την ύπαρξη τους κυρίως σε παραβιάσεις αρχών σχεδίασης (design principles), προτύπων σχεδίασης (design patterns) ή ακόμα και στην εφαρμογή αντί-προτύπων (antipatterns). Τα bad smells δεν είναι bugs. Παρόλο που η λειτουργία του συστήματος είναι η επιθυμητή, η εμφάνιση κακών οσμών στον κώδικα επιφέρει χαμηλές τιμές σε αρκετές κρίσιμες για την ποιότητα του λογισμικού μετρικές, αυξάνει την πολυπλοκότητα του συστήματος και καθιστά δυσκολότερη τη διαδικασία συντήρησης του. Οι κακές οσμές ανιχνεύονται σχετικά εύκολα από τους έμπειρους σχεδιαστές και ο εντοπισμός μπορεί να υποβοηθάτε από εργαλεία. Η αντιμετώπιση αυτού του είδους των προβλημάτων γίνεται στη συντριπτική πλειοψηφία των περιπτώσεων με την εφαρμογή «αναδομήσεων» (refactoring). Στην ουσία, ο υπάρχων κώδικας που εμφανίζει κακή οσμή αντικαθίσταται με νέο κώδικα συμβατό με τις αρχές και τα πρότυπα σχεδίασης χωρίς να επηρεάζεται η εξωτερική συμπεριφορά του συστήματος. Στη συνέχεια, θα ακολουθήσει μία σύντομη αναφορά στα code smells ταξινομημένα σε 5 ομάδες.

- **Bloaters :**

  Τα Bloaters είναι κώδικας, μέθοδοι και κλάσεις που έχουν αυξηθεί σε τέτοιο μέγεθος που είναι δύσκολα στη διαχείριση. Συνήθως δεν παρουσιάζονται αμέσως, αλλά συσσωρεύονται κατά τη διάρκεια της ανάπτυξης του έργου του λογισμικού μαζί με την εξέλιξη του προγράμματος.

Πίνακας 7 : Bloaters

Τεχνική	Ορισμός
Long Method	Μία μέθοδος περιέχει πολλές γραμμές κώδικα.
Long Class	Μία κλάση συμπεριλαμβάνει πολλά πεδία, μεθόδους, ή γραμμές κώδικα.
Long Parameter List	Χρήση περισσότερων από 3 ή 4 παραμέτρους σε μία μέθοδο.
Data Clumps	Μερικές φορές, διαφορετικά μέρη του κώδικα περιέχουν πανομοιότυπες ομάδες μεταβλητών. Αυτές οι ομάδες θα πρέπει να μετατρέπονται σε δικές τους κλάσεις.
Primitive Obsession	Θεωρείται η κατάχρηση των βασικών τύπων δεδομένων αντί για μικρά αντικείμενα σε απλές διεργασίες.

- **Object-Orientation Abusers :**

  Είναι η ελλιπής ή εσφαλμένη εφαρμογή των αρχών του αντικειμενοστρεφή προγραμματισμού.

Πίνακας 8 : Object-Orientation Abusers

Τεχνική	Ορισμός
Switch Statements	Σύνθετη χρήση της εντολή switch ή της έκφρασης if.
Temporary Field	Προσωρινά πεδία παίρνουν τις τιμές τους μόνο υπό ορισμένες προϋποθέσεις. Σε οποιαδήποτε άλλη περίπτωση είναι άδεια.
Refused Request	Αν μια κλάση χρησιμοποιεί μόνο μερικές από τις μεθόδους και τις ιδιότητες που κληρονόμησε από τις κλάσεις-γονείς της, η ιεραρχία δε συμβαδίζει. Οι περιττές μέθοδοι μπορούν να μην χρησιμοποιηθούν ή να επαναπροσδιοριστούν και να παρουσιάσουν σφάλματα.
Alternative Classes with Different Interfaces	Δύο κλάσεις εκτελούν πανομοιότυπες λειτουργίες, αλλά έχουν διαφορετικά ονόματα μεθόδων.

- **Change Preventers :**

  Αυτά τα code smells σημαίνουν ότι αν χρειαστεί να αλλάξει κάτι σε ένα σημείο του κώδικα, θα πρέπει να γίνουν πολλές αλλαγές και σε άλλα σημεία του. Ως αποτέλεσμα, η ανάπτυξη του λογισμικού γίνεται πολύ πιο πολύπλοκη και δαπανηρή.

Πίνακας 9 : Change Preventers

Τεχνική	Ορισμός
Divergent Change	Οι αλλαγές σε μία κλάση αποφέρουν πολλές αλλαγές σε ασυσχέτιστες μεθόδους.
Shotgun Surgery	Οποιαδήποτε τροποποίηση για να γίνει χρειάζονται να γίνουν πολλές μικρές αλλαγές σε πολλές διαφορετικές κλάσεις. Είναι το αντίθετο από την προηγούμενη (Divergent Change είναι όταν πολλές αλλαγές γίνονται σε μία κλάση, ενώ στην Shotgun Surgery όταν μία αλλαγή γίνεται σε πολλές κλάσεις ταυτόχρονα).
Parallel Inheritance Hierarchies	Όποτε δημιουργείται μια υποκλάση για μία κλάση, δημιουργείται η ανάγκη για δημιουργία μιας υποκλάσης για μία άλλη κλάση.

- **Dispensables :**

  Dispensables ονομάζονται τα 'απορρίμματα', είναι δηλαδή κάτι το ανούσιο και αχρείαστο στον κώδικα και η αφαίρεση του θα κάνει τον κώδικα πιο καθαρό, πιο αποδοτικό και πιο κατανοητό.

Πίνακας 10 : Dispensables

Τεχνική	Ορισμός
Comments	Μία μέθοδος είναι γεμάτη με επεξηγηματικά σχόλια.
Data Class	Μία κλάση περιέχει μόνο πεδία και μεθόδους για την πρόσβαση τους (get και set). Δεν προσφέρει λειτουργικότητα αλλά λειτουργεί ως μέσο αποθήκευσης δεδομένων για άλλες κλάσεις.
Duplicate Code	Δύο κομμάτια κώδικα είναι πανομοιότυπα.
Dead Code	Μία μεταβλητή, παράμετρος, πεδίο, μέθοδος ή κλάση δεν χρησιμοποιείται πλέον (κυρίως γιατί έχει απαρχαιωθεί).
Lazy Class	Μία κλάση έχει περιορισμένη χρηστικότητα. Προτείνεται η διαγραφή της.
Speculative Generality	Υπάρχει αχρησιμοποίητη κλάση, μέθοδος, πεδίο ή παράμετρος (συνήθως δημιουργούνται για μελλοντική χρήση αλλά ποτέ δεν χρησιμοποιούνται).

- **Couplers :**

  Όλα τα code smells σε αυτή την κατηγορία συμβάλλουν στην υπερβολική σύζευξη μεταξύ κλάσεων ή δείχνουν τι γίνεται όταν η σύζευξη αντικατασταθεί από υπερβολική ανάθεση.

Πίνακας 11 : Couplers

Τεχνική	Ορισμός
Feature Envy	Μία μέθοδος έχει συχνότερη πρόσβαση σε δεδομένα ενός άλλου αντικειμένου από ότι στα δικά της.
Inappropriate Intimacy	Μία κλάση χρησιμοποιεί τα εσωτερικά πεδία και μεθόδους μιας άλλης κλάσης.
Message Chains	Στον κώδικα υπάρχουν πολλές αλυσίδες κλήσεων. ($a->b()->c()->d())
Middle Man	Μία κλάση εκτελεί μόνο μία ενέργεια, μεταβιβάζοντας την εργασία σε άλλη τάξη.

Incomplete Library Class	Μία βιβλιοθήκη αργά ή γρήγορα σταματά να πληροί τις ανάγκες του χρήστη.

3.2.3 Αναδόμηση (Refactoring)

"I've found that refactoring helps me write fast software. It slows the software in the short term while I'm refactoring, but it makes the software easier to tune during optimization. I end up well ahead."
— Martin Fowler

Αναδόμηση είναι η διαδικασία που αλλάζει τον κώδικα ενός λογισμικού με τέτοιο τρόπο ώστε να βελτιώνεται η εσωτερική σχεδίαση χωρίς να αλλάζει η εξωτερική συμπεριφορά του προγράμματος [Fowler 1999]. Είναι ο μοναδικός τρόπος απαλοιφής των κακών οσμών, ο οποίος εγγυάται ότι η λειτουργικότητα του συστήματος, μετά την αλλαγή, θα είναι ακριβώς η ίδια με πριν (behavior preserving changes).

Θα πρέπει να σημειωθεί ότι, όλες οι αλλαγές που επιφέρει η διαδικασία της αναδόμησης του πηγαίου κώδικα ενός συστήματος λογισμικού, έχουν στόχο αρχικά, την απλοποίηση του ώστε να καθίσταται πιο κατανοητός, στοιχείο ιδιαιτέρως σημαντικό όταν το λογισμικό δεν πρόκειται να αναπτύσσεται πάντοτε από τα ίδια άτομα. Δεύτερων, ο πηγαίος κώδικας να είναι εύκολα και αποδοτικά συντηρήσιμος, δηλαδή, με μικρή προσπάθεια και κόστος, χωρίς να υποβαθμίζεται η ποιότητά του, και τέλος, εύκολα ελεγχόμενος είτε κατά τη διάρκεια της υλοποίησής του είτε μετά το πέρας αυτής.

Οι αναδομήσεις εφαρμόζονται σε τρία επίπεδα: α) αναδομήσεις υψηλού επιπέδου (high level refactorings) που αφορούν τις αλλαγές των υπογραφών μιας κλάσης, μιας μεθόδου ή μιας ιδιότητας και περιλαμβάνουν τις αναδομήσεις Rename Class, Move Static Field, και Add Parameter, β) αναδομήσεις μεσαίου επιπέδου (medium level refactorings), αλλαγές που συμπεριλαμβάνουν τις αλλαγές του προηγούμενου επιπέδου αλλά και αλλαγές που αφορούν σημαντικές τροποποιήσεις τμημάτων του πηγαίου κώδικα. Σε αυτό το επίπεδο ανήκουν οι αναδομήσεις Extract Method, Inline Constant και Convert Anonymous Type to Nested Type, και γ) αναδομήσεις χαμηλού επιπέδου (low level refactorings) όπου οι αλλαγές αφορούν μονάχα τροποποιήσεις τμημάτων του πηγαίου κώδικα και περιλαμβάνουν τις αναδομήσεις Extract Local Variable, Rename Local Variable και Add Assertion.

Οι αναδομήσεις, έγιναν ευρέως αποδεκτές από την κοινότητα της Τεχνολογίας Λογισμικού αμέσως μετά την πρώτη συστηματική τους καταγραφή [1][1] εφόσον η εφαρμογή τους είναι βέβαιο ότι οδηγεί σε βελτίωση της ποιότητας της σχεδίασης ενός συστήματος λογισμικού, γεγονός που επιβεβαιώνεται τόσο μέσα από θεωρητικές μελέτες [8] όσο και από εμπειρικές [9], [10], [11] που δείχνουν ότι οι αναδομήσεις επιδρούν θετικά πάνω σε μετρικές ποιότητας αντικειμενοστρεφούς λογισμικού.

Η διαδικασία της αναδόμησης περιλαμβάνει έναν αριθμό ξεχωριστών ενεργειών:

1) Αναγνώριση των τμημάτων του κώδικα που χρειάζονται αναδόμηση.
2) Προσδιορισμός ποιας τεχνικής πρέπει να εφαρμοστεί στα συγκεκριμένα τμήματα.
3) Βεβαιότητα ότι η εφαρμογή της τεχνικής διατηρεί την ίδια συμπεριφορά.
4) Εφαρμογή της τεχνικής.
5) Εκτίμηση της επίπτωσης της αναδόμησης στα χαρακτηριστικά του λογισμικού (π.χ. complexity, understandability, maintainability)
6) Διατήρηση της συνοχής μεταξύ του τροποποιημένου λογισμικού και των προδιαγραφών του λογισμικού.

Τεχνικές Αναδόμησης

Οι τεχνικές αναδόμησης έχουν χωριστεί σε έξι κατηγορίες. Παρακάτω παρουσιάζεται η περιγραφή αυτών των κατηγοριών καθώς επίσης κάποιες βασικές τεχνικές που τις απαρτίζουν

1) *Συγκρότηση Μεθόδων :*

   Απαρτίζεται από τεχνικές που βελτιώνουν την εσωτερική σύσταση των μεθόδων. Η χρήση αυτών απλοποιεί τον κώδικα και συνήθως μειώνει το μέγεθός του. Μεγάλο μέρος της διαδικασίας αφιερώνεται στη σωστή σύνταξη των μεθόδων. Στις περισσότερες περιπτώσεις, οι υπερβολικά μεγάλες μέθοδοι είναι η πηγή του προβλήματος. Οι ιδιομορφίες του κώδικα σε αυτές τις μεθόδους περιέχουν τη λογική εκτέλεσης κάνοντας έτσι τη μέθοδο δυσνόητη και ακόμα δυσκολεύουν οποιαδήποτε αλλαγή. Οι τεχνικές σε αυτή την κατηγορία βελτιστοποιούν τις μεθόδους, αφαιρούν τα αντίγραφα κώδικα και διευκολύνουν μελλοντικές αναβαθμίσεις.

[1] Σημειώνεται επίσης ότι, η πρώτη συστηματική λίστα καταγραφής των αναδομήσεων, σύντομα επεκτάθηκε [7].

- *Extract Method* : Η τεχνική αυτή παίρνει ένα κομμάτι κώδικα από μία μέθοδο και το εξάγει σε μία νέα, αντικαθιστώντας έπειτα αυτό το κομμάτι με κλήση προς τη νέα μέθοδο.
- *Inline Method* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας όλες τις κλήσεις μια συγκεκριμένης μεθόδου με ολόκληρο τον κώδικα αυτής. Μετά την αντικατάσταση η μέθοδος αυτή αφαιρείται από το σύστημα.
- *Inline Temp* : Η τεχνική αυτή αντικαθιστά όλες τις εμφανίσεις μιας προσωρινής μεταβλητής με τον κώδικα υπολογισμού της.
- *Replace Temp with Query* : Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κώδικα υπολογισμού μιας προσωρινής μεταβλητής σε μία νέα μέθοδο και αντικαθιστώντας όλες τις εμφανίσεις της με κλήση προς τη μέθοδο αυτή.
- *Introduce Explaining Variable* : Η τεχνική αυτή εφαρμόζεται εισάγοντας προσωρινές μεταβλητές στον κώδικα για την επεξήγηση πολύπλοκων εκφράσεων.
- *Split Temporary Variable* : Η τεχνική αυτή εφαρμόζεται διασπώντας προσωρινές μεταβλητές στον κώδικα που έχουν παραπάνω από μία αρμοδιότητες.
- *Remove Assignments to Parameters* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας τις αναθέσεις που γίνονται στις παραμέτρους μιας μεθόδου με αναθέσεις σε μια προσωρινή μεταβλητή της παραμέτρου.
- *Replace Method with Method Object* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια ολόκληρη μέθοδο με ένα νέο αντικείμενο βασισμένο σε αυτή.
- *Substitute Algorithm* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας πλήρως κάποιον αλγόριθμο του συστήματος με κάποιον άλλο.

2) *Μεταφορά Χαρακτηριστικών Μεταξύ Αντικειμένων :*

Περιέχει τεχνικές που σκοπό έχουν τη σωστή ανάθεση καθηκόντων στις διάφορες δομές του κώδικα. Η χρήση τους έχει ως αποτέλεσμα ο κώδικας να περιέχει κλάσεις και μεθόδους με ξεκάθαρη λειτουργία και σκοπό στο πρόγραμμα.

- *Move Method* : Η τεχνική αυτή εφαρμόζεται μετακινώντας μία μέθοδο από την κλάση που αυτή ανήκει σε μία άλλη.

- *Move Field* : Η τεχνική αυτή εφαρμόζεται μετακινώντας ένα πεδίο από την κλάση που αυτό ανήκει σε μία άλλη.
- *Extract Class* : Η τεχνική αυτή εφαρμόζεται μετακινώντας πεδία και μεθόδους μίας κλάσης σε μία νέα.
- *Move Class* : Η τεχνική αυτή εφαρμόζεται μετακινώντας μία κλάση από το αρχικό της πακέτο σε κάποιο άλλο πιο σχετικό από άποψη λειτουργικότητας, συνοχής και σύζευξης.
- *Inline Class* : Η τεχνική αυτή εφαρμόζεται μετακινώντας όλα τα πεδία και τις μεθόδους μιας κλάσης σε μία άλλη και διαγράφοντας την πρώτη μετά την μεταφορά.
- *Hide Delegate* : Η τεχνική αυτή εφαρμόζεται δημιουργώντας μια μέθοδο στην κλάση εξυπηρετητή για κάθε μέθοδο που η κλάση πελάτη καλεί από την κλάση αντιπρόσωπο.
- *Remove Middle Man* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας τις κλήσεις των μεθόδων στην κλάση εξυπηρετητή από την κλάση πελάτη με απευθείας κλήσεις προς τις μεθόδους της κλάσης αντιπροσώπου. Στη συνέχεια οι αντίστοιχες μέθοδοι στην κλάση εξυπηρετητή αφαιρούνται.
- *Introduce Foreign Method* : Η τεχνική αυτή εφαρμόζεται προσθέτοντας μία επιπλέον λειτουργία στην κλάση εξυπηρετητή δημιουργώντας όμως την μέθοδο που την υλοποιεί εντός της κλάσης πελάτη.
- *Introduce Local Extension* : Η τεχνική αυτή εφαρμόζεται προσθέτοντας επιπλέον λειτουργίες στην κλάση εξυπηρετητή υλοποιώντας αυτές όμως ως μία νέα υποκλάση της.

3) *Οργάνωση Δεδομένων :*

Αποτελείται από τεχνικές που προσπαθούν να κάνουν τη διαχείριση των δεδομένων στον κώδικα πιο εύκολη. Η χρήση τους μετατρέπει και αντικαθιστά διάφορους τύπους δεδομένων στον κώδικα βελτιώνοντας την εσωτερική του οργάνωση.

- *Replace Data Value with Object* : Η τεχνική αυτή εφαρμόζεται μετατρέποντας μία μεταβλητή σε αντικείμενο μιας νέας κλάσης.
- *Change Value to Reference* : Η τεχνική αυτή εφαρμόζεται μετατρέποντας ένα αντικείμενο τιμής σε ένα αντικείμενο αναφοράς.

- *Change Reference to Value* : Η τεχνική αυτή εφαρμόζεται μετατρέποντας ένα αντικείμενο αναφοράς σε ένα αντικείμενο τιμής.
- *Replace Array with Object* : Η τεχνική αυτή εφαρμόζεται μετατρέποντας μία μεταβλητή τύπου πίνακα σε αντικείμενο μιας κλάσης.
- *Duplicate Observed Data* : Η τεχνική αυτή εφαρμόζεται αντιγράφοντας ένα σύνολο δεδομένων και παρέχοντας μία κλάση η οποία θα συγχρονίζει τις αλλαγές που θα γίνονται σε αυτά.
- *Replace Magic Number with Symbolic Constant* : Η τεχνική αυτή εφαρμόζεται ορίζοντας μία αριθμητική σταθερά για κάποιον αριθμό με συγκεκριμένη σημασία στον κώδικα και αντικαθιστώντας όλες τις εμφανίσεις του με αυτό.
- *Encapsulate Field* : Η τεχνική αυτή εφαρμόζεται αλλάζοντας τον τρόπο πρόσβασης ενός πεδίου και μετατρέποντάς το από δημόσιο σε ιδιωτικό παρέχοντας τις κατάλληλες μεθόδους set και get για αυτό.

4) *Απλοποίηση Κατηγορηματικών Εκφράσεων :*

Προσφέρει τεχνικές που απλουστεύουν την πολυπλοκότητα της συνδυαστικής λογικής. Η χρήση τους βελτιώνει την αναγνωσιμότητα των δομών στον κώδικα που περιέχουν κατηγορηματικούς όρους.

- *Decompose Conditional* : Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κώδικα που περιέχεται στη συνθήκη του if, στο then και στο else σε νέες μεθόδους.
- *Consolidate Conditional Expression* : Η τεχνική αυτή εφαρμόζεται συνδυάζοντας διάφορες συνθήκες σε μία.
- *Consolidate Duplicate Conditional Fragments* : Η τεχνική αυτή εφαρμόζεται μετακινώντας ένα κομμάτι κώδικα το οποίο είναι ίδιο σε όλες τις διακλαδώσεις της κατηγορηματικής έκφρασης, έξω από αυτήν.
- *Remove Control Flag* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια μεταβλητή σημαίας σε μια συνθήκη με εντολές break ή return.
- *Replace Conditional with Guard Clauses* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας τις εμφωλευμένες εντολές συνθήκης με συνθήκες φρουρούς.
- *Replace Conditional with Polymorphism* : Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κώδικα σε κάθε κλαδί του κατηγορήματος σε μια μέθοδο της υποκλάσης.

5) *Βελτίωση Κλήσεων Μεθόδων :*

Παρέχει τεχνικές που επηρεάζουν τις διάφορες μεθόδους του κώδικα ώστε η χρήση αυτών να είναι απλή και κατανοητή. Η χρήση τους συνεισφέρει σε απλοποιημένα πρότυπα μεθόδων.

- *Rename Method* : Η τεχνική αυτή εφαρμόζεται αλλάζοντας το όνομα μιας μεθόδου έτσι ώστε να αντικατοπτρίζει τον σκοπό της.
- *Add Parameter* : Η τεχνική αυτή εφαρμόζεται προσθέτοντας μία επιπλέον παράμετρο σε μία μέθοδο.
- *Remove Parameter* : Η τεχνική αυτή εφαρμόζεται αφαιρώντας μία από τις παραμέτρους μιας μεθόδου.
- *Separate Query from Modifier* : Η τεχνική αυτή εφαρμόζεται διασπώντας μία μέθοδο η οποία επιστρέφει μία τιμή και μεταβάλει την κατάσταση ενός αντικειμένου σε δύο νέες ξεχωριστές μεθόδους.
- *Parameterize Method* : Η τεχνική αυτή εφαρμόζεται ενοποιώντας παρόμοιες μεθόδους σε μία της οποίας οι παράμετροι καθορίζουν τις διαφορετικές τιμές μεταξύ των αρχικών μεθόδων.
- *Hide Method* : Η τεχνική αυτή εφαρμόζεται μεταβάλλοντας την ορατότητα μιας μεθόδου από δημόσια σε ιδιωτική.
- *Encapsulate Downcast* : Η τεχνική αυτή εφαρμόζεται κάνοντας downcast την τιμή που επιστρέφει μια μέθοδος μέσα στο σώμα της και αλλάζοντας έπειτα τον τύπο επιστροφής της μεθόδου.
- *Replace Error Code with Exception* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας την τιμή επιστροφής μιας μεθόδου που αυτή χρησιμοποιεί για να υποδείξει κάποιο πιθανό σφάλμα με μία εξαίρεση.

6) *Βελτίωση Γενίκευσης :*

Αποτελείται από τεχνικές που βελτιώνουν την ιεραρχία των κλάσεων του συστήματος. Η χρήση τους έχει ως αποτέλεσμα την μετακίνηση στοιχείων μεταξύ της ιεραρχίας των κλάσεων με σκοπό τη βελτίωση της κληρονομικότητας.

- *Pull Up Field* : Η τεχνική αυτή εφαρμόζεται μετακινώντας τα κοινά πεδία μεταξύ δύο ή περισσότερων υποκλάσεων στην υπερκλάση τους.
- *Pull Up Method* : Η τεχνική αυτή εφαρμόζεται μετακινώντας τις κοινές μεθόδους μεταξύ δύο ή περισσότερων υποκλάσεων στην υπερκλάση τους.

- *Push Down Method* : Η τεχνική αυτή εφαρμόζεται μετακινώντας μια μέθοδο της υπερκλάσης σε μια συγκεκριμένη υποκλάση.
- *Push Down Field* : Η τεχνική αυτή εφαρμόζεται μετακινώντας ένα πεδίο της υπερκλάσης σε μια συγκεκριμένη υποκλάση.
- *Extract Superclass* : Η τεχνική αυτή εφαρμόζεται εξάγοντας τα κοινά χαρακτηριστικά μεταξύ δύο ή περισσότερων κλάσεων σε μια νέα υπερκλάση και μετατρέποντάς τες σε υποκλάσεις αυτής.
- *Collapse Hierarchy* : Η τεχνική αυτή εφαρμόζεται συγχωνεύοντας μια υπερκλάση με μία υποκλάση της.
- *Replace Inheritance with Delegation* : Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια σχέση κληρονομικότητας μεταξύ δύο κλάσεων σε απλή συσχέτιση.

3.2.4 Μετρικές Αναδόμησης στη Μεθοδολογία

"Stick to only a few metrics. Don't get analysis paralysis"

3.2.4.1 Μέτρο Επίδρασης Αλλαγών - Ripple Effect Measure (REM)

Το REM είναι ένα μέτρο το οποίο αξιολογεί την πιθανότητα αλλαγής μιας κλάσης λόγω αλλαγών σε μια άλλη κλάση του συστήματος. Αποτελεί μέτρηση δομικής σύζευξης.

Η ανάλυση εξαρτήσεων αποτελεί τον πυρήνα αλγορίθμων που ερευνούν το φαινόμενο κυματισμών (ripple effect) το οποίο προκαλείται από την αλλαγή σε μία κλάση, υπό την έννοια ότι οι αλλαγές μεταδίδονται, σε περισσότερες κλάσεις του συστήματος, μέσω των εξαρτήσεων τους. Αυτές οι μεταβολές αλλαγών (δηλαδή, το πιο κοινό φαινόμενο κυμάτωσης), είναι το αποτέλεσμα ορισμένων τύπων αλλαγών σε μία κλάση (π.χ. αλλαγή της υπογραφής μεθόδου - δηλαδή, όνομα μεθόδου, τύποι παραμέτρων και τύπος επιστροφής - που γίνεται επίκληση μέσα σε μια άλλη μέθοδο) και εκπέμπουν δυνητικά τις αλλαγές σε άλλες κλάσεις. Αυτοί οι τύποι αλλαγών ποικίλουν ανάλογα με τους διαφορετικούς τύπους εξαρτήσεων. Σύμφωνα με τον Van Vliet (Van Vliet, 2008), υπάρχουν τρεις τύποι εξαρτήσεων κλάσεων, η γενίκευση, η συγκράτηση και η σύνδεση.

- **Η γενίκευση** χρησιμοποιείται για να αντιπροσωπεύει τις σχέσεις "IS-A". Σε μια γενίκευση, υπάρχουν τρεις πιθανοί λόγοι μεταβολής αλλαγής: (1) επίκληση υπερ -

μεθόδου (χρήση της λέξης κλειδί super), (2) πρόσβαση των προστατευόμενων πεδίων, και (3) παράκαμψη ή εφαρμογή αφηρημένων μεθόδων από την υποκατηγορία.

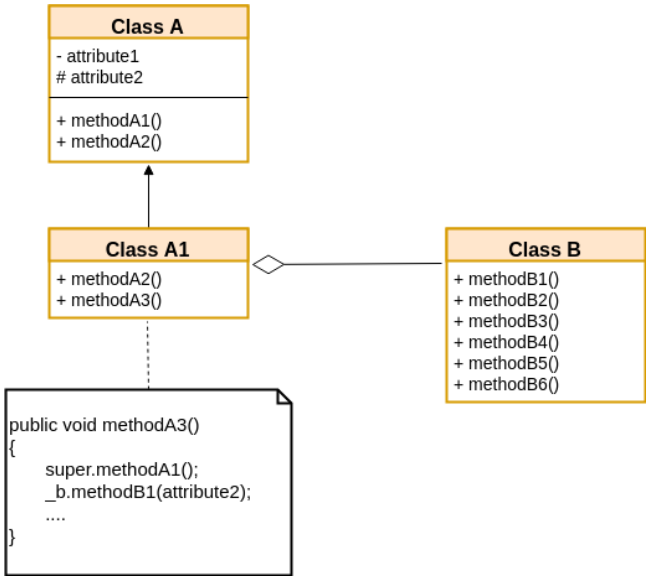
- **Η συγκράτηση** χρησιμοποιείται για να αντιπροσωπεύει τις σχέσεις "HAS-A" ή "PART-WHOLE". Σε μια σχέση περιεκτικότητας, οι αλλαγές μπορούν να διαδοθούν μέσω των κλήσεων μεθόδου της κλάσης Container στη δημόσια διεπαφή της κατηγορίας των Containee κλάσεων. Εκφράζει μια σχέση όπου ένα αντικείμενο κλάσης Container έχει την ευθύνη να διατηρήσει και να διαχειριστεί αντικείμενα τύπου κλάσης Containee τα οποία έχουν δημιουργηθεί εκτός της κλάσης Container.
- **Η ένωση** χρησιμοποιείται για να αντιπροσωπεύει τις σχέσεις που οφείλονται στη δήλωση τοπικής μεταβλητής μέσα σε μια μέθοδο ή λόγω της χρήση ενός αντικειμένου κλάσης ως τύπου παραμέτρου / επιστροφής σε μία μέθοδο μιας άλλης κλάσης. Σε μια σχέση ένωσης, οι αλλαγές μπορούν να μεταδοθούν μέσω κλήσεων μεθόδου μιας κλάσης προς τη δημόσια διασύνδεση μιας άλλης κλάσης.

Σημειώνεται ότι ο προαναφερθείς τρόπος με τον οποίο διαδίδονται οι αλλαγές μέσω εξαρτήσεων κλάσης αφορά σχεδίαση λογισμικού που ακολουθεί βασικές αρχές αντικειμενοστρεφούς σχεδίασης, δηλαδή ενθυλάκωση (οι κλάσεις δεν διαθέτουν δημόσια χαρακτηριστικά). Σε περιπτώσεις όπου οι κλάσεις διατηρούν δημόσιες ιδιότητες, αυτά τα δημόσια χαρακτηριστικά θεωρούνται ως αιτία αλλαγής της διάδοσης, υπό την έννοια ότι αυτές ανήκουν στην κλάση δημόσιας διασύνδεσης.

Ακολουθεί παράδειγμα το οποίο εξετάζει την περίπτωση που φαίνεται στην Εικόνα 5. Στην Εικόνα 5 υπάρχουν δύο σχέσεις (η κλάση A1 εξαρτάται από την κλάση A, και η κλάση A1 εξαρτάται από την κλάση B). Τίθενται δύο ερωτήματα για τις προαναφερθείσες σχέσεις. Είναι οι σχέσεις αυτές εξίσου ισχυρές; Είναι εξίσου πιθανό για μια αλλαγή που συμβαίνει στην κλάση A και μια αλλαγή που συμβαίνει στην κλάση B να οδηγήσουν σε αλλαγές στην κλάση A1; Προκειμένου να δοθεί απάντηση σε τέτοιου είδους ερωτήσεις, είναι απαραίτητη μια μέτρηση που ποσοτικοποιεί την πιθανότητα μιας κυμαινόμενης επίδρασης (ripple effect) μεταξύ των κλάσεων. Για το σκοπό αυτό, ορίζεται το Ripple Effect Measure (REM), μία μετρική που μετρά την πιθανότητα μιας τυχαίας αλλαγής που συμβαίνει στη δημόσια διεπαφή μιας κλάσης πηγής (A ή B) να μεταδοθεί σε μια εξαρτώμενη κλάση (A1) που τη χρησιμοποιεί, υποθέτοντας ότι όλα τα στοιχεία των κλάσεων προέλευσης (δηλαδή τα

χαρακτηριστικά και οι μέθοδοι) έχουν την ίδια πιθανότητα αλλαγής. Για τον υπολογισμό του REM, πρέπει να ληφθεί υπόψη το πλήθος των παρακάτω:

- όλα τα μέλη των κλάσεων προέλευσης (A ή B) που έχουν πρόσβαση από την εξαρτώμενη κλάση (A1), τα οποία (αν αλλάξουν) θα εκπέμψουν μία ή περισσότερες αλλαγές στην εξαρτώμενη κλάση (A1)
- όλα τα μέλη της δημόσιας διασύνδεσης των κλάσεων προέλευσης (A ή B)



Εικόνα 5 : Παράδειγμα Εξαρτήσεων σε Επίπεδο Κλάσεων

Η αναλογία των δύο προαναφερθέντων αριθμών είναι μια εκτίμηση της πιθανότητας μιας τυχαίας αλλαγής στη δημόσια διεπαφή μιας κλάσης πηγής, η οποία θα εμφανιστεί σε ένα μέλος το οποίο στη συνέχεια θα μεταβιβάσει την αλλαγή στην κλάση που εξαρτάται. Με άλλα λόγια, όπως ο αριθμός των μέλη της κλάσης πηγής που εκπέμπουν αλλαγές σε μια άλλη κλάση η οποία εξαρτάται από τον συνολικό αριθμό των μελών, μπορεί να αλλάξει στην κλάση πηγής, έτσι καθίσταται πιο πιθανό οι αλλαγές να μεταδίδονται από την κλάση προέλευσης στις εξαρτώμενες κλάσεις. Συνεπώς, λαμβάνοντας υπόψη τους τρεις τύπους εξαρτήσεων και του τρόπου με τον οποίο μεταβάλλονται οι αλλαγές σε αυτές, η μετρική REM για μια εξάρτηση (δηλαδή ο παράγοντας διάδοσης) μεταξύ των εξαρτημένων κλάσεων και της κλάσης πηγής, μπορεί να υπολογιστεί ως εξής:

$$\text{REM dependency} = \frac{NDMC + NOP + NPrA}{NOM + NA} \quad (1)$$

NDMC: Αριθμός διακριτών κλήσεων μεθόδων από τις εξαρτώμενες κλάσεις στην κλάση προέλευσης (+ .super() για την περίπτωση γενίκευσης)

NOP: Αριθμός πολυμορφικών μεθόδων στην κλάση προέλευσης (ισχύει μόνο για γενίκευση)

NPrA: Αριθμός προστατευμένων χαρακτηριστικών στην κλάση προέλευσης (ισχύει μόνο για γενίκευση ή κλάσεις φίλων)

NOM: Αριθμός μεθόδων στην κλάση προέλευσης

NA: Αριθμός χαρακτηριστικών στην κλάση προέλευσης

Με βάση τα παραπάνω, στο παράδειγμα της Εικόνας 5 υπάρχουν τρεις αλλαγές οι οποίες προέρχονται από την υπερκλάση A στην υποκλάση A1:

- Αλλαγή στην υπογραφή της μεθόδου methodA1. Αλλαγή της υπογραφής της μεθόδου methodA1, θα οδηγούσε σε σφάλμα μεταγλώττισης (compile error) στην αντίστοιχη κλήση (σώμα της μεθόδου methodA3).
- Αλλαγή στην υπογραφή της μεθόδου methodA2. Αλλαγή της υπογραφής της μεθόδου methodA2 θα οδηγούσε σε σφάλμα μεταγλώττισης (compile error) στη θέση όπου η methodA2 υπερισχύει, αφού η methodA2 δηλώνεται ως αφηρημένη στην υπερκλάση.
- Αλλαγή στο όνομα του attribute2. Αλλαγή του ονόματος του χαρακτηριστικού attribute2, θα οδηγήσει σε σφάλμα μεταγλώττισης (compile error) στο σώμα της μεθόδου methodA3.

Επομένως η μετρικής REM για την εξάρτηση της κλάσης A1 προς την κλάση A υπολογίζεται ως εξής :

$$REM_{A1(A)}{}^{2} = \frac{NDMC+NOP+NPrA}{NOM+NA} = \frac{1+1+0}{2+0} = \frac{1+1+1}{2+2} = 0.75$$

Ομοίως, υπάρχει μόνο μία περίπτωση για διάδοση αλλαγής από την κλάση containee (B) στην κλάση container (A1), δηλαδή η αλλαγή της υπογραφής της μεθόδου methodB1. Ξ αλλαγή της υπογραφής της methodB1, θα οδηγήσει σε σφάλμα μεταγλώττισης (compile error) στην αντίστοιχη κλήση. Όπως αναφέρεται πιο πάνω, αν αλλάξει οποιαδήποτε άλλη μέθοδος της κλάσης B, η αλλαγή που θα γίνει δεν διαδίδεται στην κλάση A1, εφόσον η μέθοδος δεν καλείται στο εφαρμογή του A1 (στο παράδειγμα δεν υπάρχει άλλη μέθοδος της κλάσης B που να καλείται από το A1, εκτός από την methodB1). Έτσι, το REM, για την εξάρτηση της κλάσης A1 στην κλάση B, μπορεί να υπολογιστεί ως εξής:

[2] This refers to the REM of class A1, due to its dependency to A.

$$REM_{A1(B)}^{2} = \frac{NDMC+NOP+NPrA}{NOM+NA} = \frac{1+1+0}{2+0} = \frac{1+0+0}{5+0} = 0.20$$

Επομένως, με βάση το REM, για το παράδειγμα του σχήματος στην Εικόνα 5, συμπεραίνεται ότι μια αλλαγή που συμβαίνει στην κλάση Α είναι πιο πιθανό να μεταδοθεί στην κλάση Α1, από μια αλλαγή που εμφανίζεται στην κλάση Β.

Προς το παρόν, ο υπολογισμός του REM πραγματοποιείται στο επίπεδο εξαρτήσεων κλάσης, δηλαδή από μία κλάση προς κάποια άλλη. Προκειμένου να συσσωματωθεί η μετρική REM σε υψηλότερο επίπεδο αρχιτεκτονικής, δηλαδή σε επίπεδο κλάσης, πρέπει να ληφθούν υπόψη οι εξαρτήσεις μίας κλάσης προς όλες τις άλλες.

Στο παράδειγμα της Εικόνας 5, η κλάση Α1 θα πρέπει να αλλάξει εάν μια αλλαγή μεταδίδεται από οποιαδήποτε κλάση Β ή Α ή και από τις δύο. Επομένως, για μια κλάση που έχει πολλές εξαρτήσεις, προτείνεται η χρήση της κοινής πιθανότητας όλων των γεγονότων (δηλαδή αλλαγή σε οποιαδήποτε εξάρτηση), για τη συγκέντρωση της βαθμολογίας του REM από τις εξαρτήσεις σε επίπεδο κλάσης. Για παράδειγμα, στην περίπτωση 1 του Σχήματος της Εικόνας 5, η πιθανότητα υπολογίζεται ως εξής:

P(A∪B) = P(A) + P(B) – P(A) P(B) = 0.75 + 0.20 – 075*0.20 = 0.8

P(A∪B): $REM_{A1(A, B)}$

P(A): $REM_{A1(A)}$

P(B): $REM_{A1(B)}$

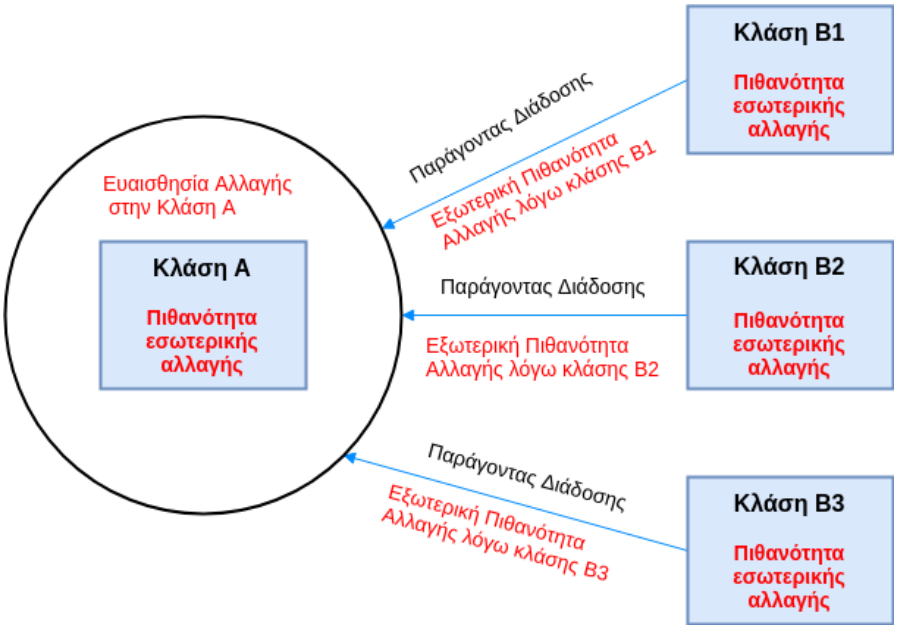
3.2.4.2 Πιθανότητα Μεταβλητότητας Κλάσης - Module Change Proneness Measure (MCPM)

Η μεταβλητότητα της ευκρίνειας είναι ένα ποιοτικό χαρακτηριστικό των αντικειμένων λογισμικού που αντιπροσωπεύει την πιθανότητά τους να αλλάξουν στο μέλλον εξαιτίας: (α) εξελισσόμενων απαιτήσεων, (β) διόρθωσης σφαλμάτων, (γ) αλλαγές λόγω εξαρτήσεων (ripple effects). Η μετρική αυτή ελέγχει την πιθανότητα μεταβολής σε επίπεδο αρχιτεκτονικής. Στη βιβλιογραφία, η μεταβλητότητα έχει συνδεθεί με πολλές αρνητικές συνέπειες κατά την εξέλιξη του λογισμικού. Για παράδειγμα, αντικείμενα που είναι επιρρεπή σε αλλαγές τείνουν να παράγουν περισσότερα ελαττώματα και να συσσωρεύουν περισσότερο τεχνικό χρέος. Συνεπώς, η αναγνώριση και η παρακολούθηση ενοτήτων του συστήματος που είναι επιρρεπείς στις αλλαγές είναι υψίστης σημασίας.

Η αξιολόγηση της μεταβλητότητας απαιτεί πληροφορίες από δύο πηγές: (α) το ιστορικό αλλαγών στο λογισμικό ως υποκατάστατο του πόσο συχνά μεταβάλλεται το ίδιο το λογισμικό και (β) τη δομή του πηγαίου κώδικα που επηρεάζει την πιθανότητα να μεταδοθεί μια αλλαγή μεταξύ των στοιχείων που αποτελούν ένα αντικειμενοστρεφές λογισμικό όπως αντικείμενα, κλάσεις, μέθοδοι.

Η αξιολόγηση του αν μια δεδομένη ενότητα λογισμικού θα αλλάξει στο μέλλον είναι ένας φιλόδοξος στόχος, διότι οποιαδήποτε απόφαση για πραγματοποίηση αλλαγών σε μία κλάση, επιφέρει αλλαγές σε άλλες κλάσεις ή ενότητες λογισμικού. Γνωρίζοντας και αξιολογώντας εκ των προτέρων ποιες ενότητες κώδικα συνδέονται, οδηγεί σε καλύτερη σχεδίαση και συντήρηση του λογισμικού. Η πιθανότητα να αλλάξει κάποια κλάση στο μέλλον επηρεάζεται όχι μόνο από την πιθανότητα τροποποίησης της ίδιας της κλάσης, αλλά και από τις πιθανές αλλαγές σε άλλες κλάσεις οι οποίες συνδέονται μαζί της, συνήθως μέσω κλήσεων μεθόδων. Αυτά, τα λεγόμενα φαινόμενα κυμάτωσης (ripple effects) [12] που προκαλούν μεταβολή αλλαγής, είναι το αποτέλεσμα των εξαρτήσεων [13] μεταξύ των κλάσεων μέσω των οποίων μια αλλαγή σε μια κλάση (όπως η αλλαγή σε μια υπογραφή μεθόδου - δηλαδή, το όνομα της μεθόδου, ο τύπος) μπορεί να επηρεάσει άλλες κλάσεις που τους επιβάλλουν να τροποποιηθούν. Η μέθοδος που χρησιμοποιείται στη μελέτη αυτή [13] αναλύει τις εξαρτήσεις στις οποίες συμμετέχει κάθε κλάση και υπολογίζει την αλλαγή ευκρίνειας της κλάσης. Ο υπολογισμός της μεταβλητότητας βασίζεται σε δύο κύριους παράγοντες: την εσωτερική πιθανότητα αλλαγής (η πιθανότητα μιας κλάσης να αλλάξει λόγω αλλαγών στις απαιτήσεις, διόρθωση σφαλμάτων, κλπ.) και την εξωτερική πιθανότητα αλλαγής, η οποία αντιστοιχεί στην πιθανότητα μιας κλάσης να αλλάξει λόγω της κυμάτωσης (δηλαδή, αλλαγές που μεταδίδονται από άλλες κλάσεις). Κάθε εξάρτηση φέρει διαφορετική πιθανότητα διάδοσης αλλαγών (συντελεστή διάδοσης), η οποία πιθανότητα χρησιμοποιείται στον υπολογισμό της αντίστοιχης εξωτερικής πιθανότητας αλλαγής: εάν η κλάση A έχει εξάρτηση σε μια άλλη κλάση B, η εξωτερική πιθανότητα της κλάσης A λόγω της μεταβολής της κλάσης B λαμβάνεται ως εξής:

$P(A:external_B) = P(A\|B) \bullet P(B)$
$P(A\|B)$ είναι ο συντελεστής διάδοσης μεταξύ των κλάσεων B και A (δηλαδή, η πιθανότητα μιας αλλαγής που έγινε στην κλάση B να εκπέμπεται στην κλάση A). $P(B)$ αναφέρεται στην εσωτερική πιθανότητα αλλαγής της κλάσης B.



Εικόνα 6 : Σχέσεις Εσωτερικών και Εξωτερικών Πιθανοτήτων μεταξύ Κλάσεων

Παρακάτω ακολουθεί παράδειγμα το οποίο εξετάζει την περίπτωση του Σχήματος της Εικόνας 6.

Ο υπολογισμός της τάσης μεταβολής για την κλάση A (Εικόνα 6) πρέπει να λαμβάνει υπόψη:

- Την εσωτερική πιθανότητα αλλαγής της κλάσης A, το P (A).
- Την εξωτερική πιθανότητα αλλαγής λόγω των επιπτώσεων κυματισμού (ripple effects) από την κλάση B1, $P(A:external_{B1})$. Αυτή η τιμή αντιπροσωπεύει την πιθανότητα αλλαγής της κλάσης A λόγω της εξάρτησής της από την κλάση B1. Επομένως, εξαρτάται τόσο από την εσωτερική πιθανότητα να αλλάξει η B1 (ως έναυσμα για την επίδραση κυματισμού) όσο και από την πιθανότητα οι αλλαγές αυτές να μεταδοθούν κατά μήκος της εξάρτησης B1 → A (ως υποκατάστατο της πιθανότητας να μεταδοθεί η αλλαγή από την κλάση B1 στην κλάση A).
- Την εξωτερική πιθανότητα αλλαγής λόγω των επιπτώσεων κυματισμού (ripple effects) από την κλάση B2, $P(A:external_{B2})$.
- Την εξωτερική πιθανότητα αλλαγής λόγω των επιπτώσεων κυματισμού (ripple effects) από την κλάση B3, P(A:externalB3).

Επειδή μια κλάση μπορεί να εμπλέκεται σε διάφορες εξαρτήσεις (π.χ., η κλάση A στην Εικόνα 6) και επειδή ακόμη και μία αλλαγή των εξαρτημένων κλάσεων (είτε στη B1

είτε στη B2 ή στη B3 για το παράδειγμα του Σχήματος της Εικόνας 6) είναι ένας λόγος για να αλλάξει αυτή η κλάση, η αλλαγή της ευκρίνειας (CPM) υπολογίζεται ως η κοινή πιθανότητα όλων των γεγονότων που μπορούν να προκαλέσουν αλλαγή σε μια κλάση. Έτσι, στο προαναφερθέντα παράδειγμα (Εικόνα 6), η κλάση A μπορεί να αλλάξει λόγω των παρακάτω γεγονότων (των οποίων οι πιθανότητες να συμβούν υπολογίζεται με την πράξη της ένωσης): (α) αλλαγή στην ίδια την κλάση A, (β) ένα φαινόμενο κυμάτωσης (ripple effect) από την κλάση B1, (γ) ένα φαινόμενο κυμάτωσης (ripple effect) από την κλάση B2, ή (δ) ένα φαινόμενο κυμάτωσης (ripple effect) από την κλάση B3:

CPM(A) = *Joint Probability* {P(A), $P(A:external_{B1})$, $P(A:external_{B2})$, $P(A:external_{B3})$}

Η ακρίβεια στην εκτίμηση του CPM εξαρτάται από την ακρίβεια των εκτιμήσεων της εσωτερικής πιθανότητας αλλαγής για κάθε κλάση και του συντελεστή διάδοσης για κάθε εξάρτηση.

Όσον αφορά την εσωτερική πιθανότητα αλλαγής, χρησιμοποιείται το ποσοστό των εξαρτήσεων στις οποίες άλλαξε μια κλάση. Συγκεκριμένα, μελετούνται όλες οι εξαρτήσεις μεταξύ δύο διαδοχικών εκδόσεων ενός συστήματος και μετράτε σε πόσες από αυτές, κάθε κλάση έχει αλλάξει. Αυτό το ποσοστό υπολογίζεται για όλα τα προηγούμενα ζεύγη εκδόσεων και ο μέσος όρος που λαμβάνεται χρησιμοποιείται ως η εσωτερική πιθανότητα αλλαγής. Σημειώνεται ότι, προτιμήθηκε να χρησιμοποιηθεί ένας μέσος όρος της συχνότητας αλλαγής μεταξύ όλων των ζευγών εκδόσεων, αντί της συχνότητας αλλαγής σε ολόκληρο τον κύκλο ζωής. Το όφελος αυτής της απόφασης είναι ότι η εσωτερική πιθανότητα αλλαγής υπολογίζεται μέσω ενός αριθμού δεσμεύσεων που βρίσκονται στο ίδιο επίπεδο μεγέθους με την προβλεπόμενη μεταβλητή (δηλαδή την πιθανότητα αλλαγής από μία έκδοση σε μια άλλη).

Όσον αφορά τον παράγοντα διάδοσης των μεταβολών μεταξύ των εξαρτημένων κλάσεων, χρησιμοποιείται το Ripple Effect Measure (REM), το οποίο ποσοτικοποιεί την πιθανότητα μιας αλλαγής που συμβαίνει στην κλάση B να μεταδοθεί σε μία εξαρτώμενη κλάση A. Το REM ουσιαστικά ποσοτικοποιεί το ποσοστό της δημόσιας διασύνδεσης μιας κλάσης στην οποία γίνεται πρόσβαση από μια εξαρτημένη κλάση. Ο υπολογισμός του REM βασίζεται στην ανάλυση εξάρτησης και αναλύθηκε στην προηγούμενη ενότητα. Το REM έχει αξιολογηθεί εμπειρικά και θεωρητικά ως έγκυρος αξιολογητής της ύπαρξης του

κυμαινόμενου αποτελέσματος (ripple effect) μέσω μιας μελέτης περιπτώσεων για έργα ανοιχτού κώδικα [15].

3.2.4.3 Μετρικές Σύζευξης σε Επίπεδο Πακέτου

Οι μετρικές σύζευξης, της ελαφριάς (efferent - Ce) και σχετικής (afferent - Ca) σύζευξης βοηθούν στην κατανόηση του πόσο πιθανό είναι ένα ελάττωμα, bug ή αλλαγή σε μια κλάση να μεταδοθεί σε ένα σύστημα. Ένα σύστημα λογισμικού υπολογιστών που έχει χαμηλή σύζευξη τείνει να είναι πιο σταθερό από κάποιο που έχει υψηλή σύζευξη. Αυτές οι μετρικές, που ορίζονται από τον Robert C. Martin στο βιβλίο του [3], αποτελούν το κλειδί για τη μέτρηση της συνολικής σύζευξης σε ένα σύστημα λογισμικού.

Η ελαφριά σύζευξη μετρά τον αριθμό των κλάσεων με τις οποίες εξαρτάται μια δεδομένη κλάση. Ένας καλός τρόπος να οριστεί το efferent (έναντι του afferent) είναι ότι οι κλάσεις με υψηλή τιμή της ελαφριάς σύζευξης θα λάβουν τα αποτελέσματα αλλαγών ή ελαττωμάτων από άλλες κλάσεις. Είναι μία κλασική περίπτωση όπου θα παρουσιαστεί ένα σφάλμα σε ένα τμήμα λογισμικού, το οποίο θα επιφέρει σφάλματα και σε άλλα τμήματα λογισμικού. Μερικές φορές, η φύση ενός στοιχείου θα οδηγήσει σε υψηλότερο επίπεδο ελαφριάς σύζευξης. Οι κλάσεις που εκτελούν δραστηριότητες όπως η ενορχήστρωση τείνουν να καταλήγουν σε αυτή την κατηγορία, επειδή μέρος της λειτουργίας τους είναι ότι πρέπει να μπορούν να συνεργαστούν με πολλές άλλες κλάσεις για την υλοποίηση λειτουργιών. Η κατανομή των αρμοδιοτήτων μπορεί να συμβάλλει στη μείωση της σύζευξης, αλλά στην πραγματικότητα υπάρχουν ορισμένα στοιχεία των οποίων η φύση θα απαιτήσει τουλάχιστον ένα μέτριο επίπεδο απαγωγής.

Η συγκεκριμένη διπλωματική μελετά την αρχιτεκτονική σε επίπεδο πακέτου, επομένως η μετρική “έντασης ή ελαφριάς σύζευξης” (Efferent coupling) προσδιορίζει τον αριθμό των κλάσεων εντός ενός πακέτου οι οποίες έχουν εξαρτήσεις με κλάσεις κάποιου άλλου πακέτου. Η σχέση αυτή αφορά ζευγάρια. Δηλαδή πακέτο Α → Β, πακέτο Α → Γ, πακέτο Β → Γ και ούτε καθεξής. Μια μεγάλη τιμή της μετρικής αυτής υποδηλώνει ότι ένα πακέτο δεν ακολουθεί την αρχή της μοναδικής αρμοδιότητας και μπορεί επίσης να υποδηλώνει ότι είναι ασταθές το πακέτο αυτό, καθώς εξαρτάται από τη σταθερότητα όλων πακέτων με τα οποία είναι συνδεδεμένο. Το RefactorIT (ένα NetBeans module) συνιστά ένα ανώτατο όριο, την τιμή 20. Το Ce μπορεί να μειωθεί με την μετακίνηση κλάσεων από το

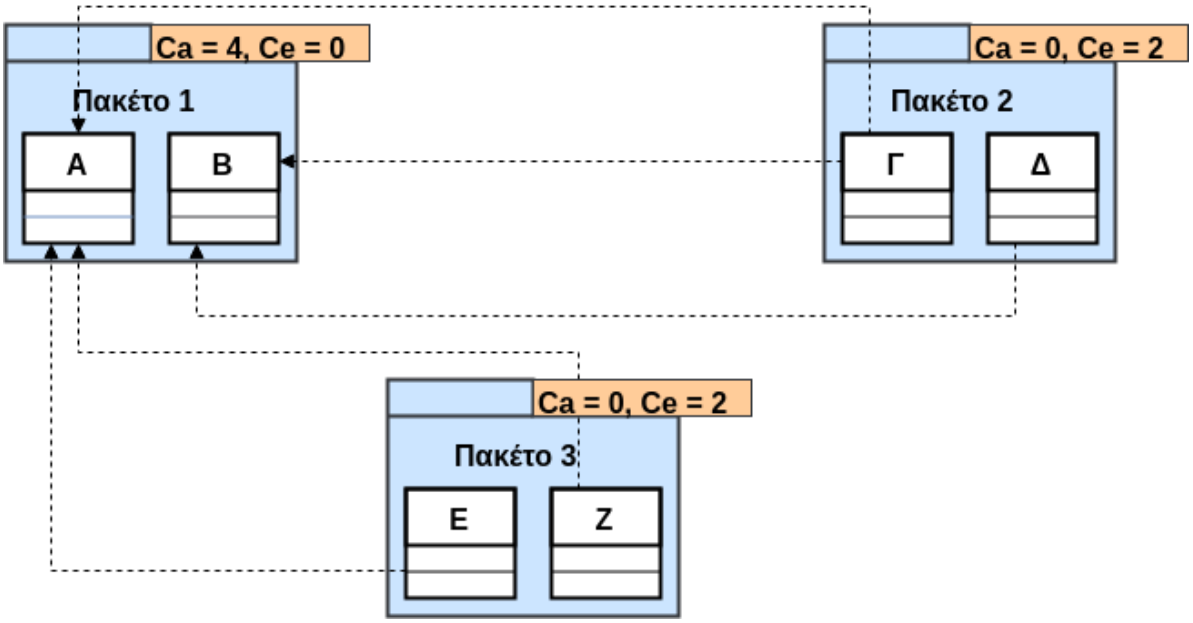
αρχικό πακέτο προς ένα άλλο πιο σχετικό, έτσι ώστε το πακέτο να γίνει πιο συνεκτικό και να μειωθούν οι εξωτερικές συζεύξεις.

Ο δείκτης αστάθειας μετρά την ελαφριά σύζευξη σε σχέση με την ολική σύζευξη. Εάν υπάρχει υψηλή σχετική σύζευξη αλλά δεν υπάρχει ελαφριά σύζευξη, εξακολουθεί να είναι λογικό να υποθέσουμε ότι η συγκεκριμένη κλάση ή το συγκεκριμένο πακέτο είναι σταθερό. Μια κλάση ή πακέτο που, από την άλλη πλευρά, έχει υψηλές τιμές ελαφριάς και σχετικής σύζευξης τείνει να είναι το μέσο, μέσω του οποίου τα σφάλματα διαδίδονται σε άλλα τμήματα του λογισμικού. Ο δείκτης αστάθειας υπολογίζεται χρησιμοποιώντας τον παρακάτω τύπο:

$$\frac{Ce}{Ce + Ca}$$

Όπου Ce είναι η ελαφριά σύζευξη και Ca είναι η σχετική σύζευξη. Η τιμή 1 σημαίνει ότι είναι εξαιρετικά ασταθής ενώ η τιμή 0 σημαίνει ότι είναι σταθερή η κλάση ή το πακέτο.

Στην Εικόνα 7 φαίνεται ένα παράδειγμα ενός τμήματος λογισμικού και ο υπολογισμός των μετρικών ελαφριάς και σχετικής σύζευξης. Υπάρχουν τρία πακέτα, το πακέτο 1, το πακέτο 2 και το πακέτο 3 τα οποία αποτελούνται από δύο κλάσεις το καθένα. Το Πακέτο 1 δεν επικοινωνεί με κάποιο άλλο πακέτο ωστόσο επικοινωνούν μαζί του τα Πακέτα 2 και 3. Οι κλάσεις από τα Πακέτα 2 και 3 που έχουν εξαρτήσεις με τις κλάσεις του πακέτου 1 είναι οι Γ, Δ, Ε και Ζ επομένως το Ca είναι τέσσερα (λόγω των σχέσεων Γ → Β, Δ → Β, Ε → Α και Ζ → Α). Το Ce είναι μηδέν διότι το Πακέτο 1 δεν έχει εξαρτήσεις προς άλλα πακέτα. Το Πακέτο 2 επικοινωνεί με τις κλάσεις του πακέτου 1 αλλά δεν υπάρχουν εξαρτήσεις από άλλα πακέτα προς αυτό ή από αυτό. Οπότε το Ce είναι δύο (λόγω των εξαρτήσεων Γ → Β, Γ → Α και Δ → Α, είναι δύο επειδή αναφέρετε σε πλήθος κλάσεων και όχι συνδέσεων) και το Ca είναι μηδέν. Τέλος, το Πακέτο 3 έχει δύο εξαρτήσεις προς το Πακέτο 1 και προς αυτό δεν έχει εξαρτήσεις κανένα άλλο πακέτο, έτσι το Ca είναι μηδέν και το Ce είναι δύο λόγω των σχέσεων Ε → Α και Ζ→ Α.



Εικόνα 7 : Εξαρτήσεις μεταξύ Κλάσεων Πακέτων

Μία ακόμα σημαντική μετρική η οποία λαμβάνεται υπόψη από αυτή τη διπλωματική είναι η μετρική Συνολική Σύζευξη Μεταξύ Πακέτων (TCIP – Total Coupling Intensity between Packages). Η μετρική αυτή αντιπροσωπεύει τον αριθμό των εξαρτήσεων κλάσης μεταξύ διαφορετικών πακέτων. Αυτή η μέτρηση εμπνέεται από την παραδοσιακή μέτρηση σύζευξης μεταξύ αντικειμένων, η οποία υπολογίζεται σε επίπεδο κλάσης [16]. Η ιδέα της χρησιμοποίησης δύο μετρήσεων ζεύξης είναι ότι η (Ce) καταγράφει το πλήθος των εξαρτήσεων στο επίπεδο αρχιτεκτονικής, ενώ η (TCIP) την ένταση της εξάρτησης.

3.2.4.4 Μετρικές Συνοχής σε Επίπεδο Πακέτου

Για τον υπολογισμό της συνοχής σε επίπεδο πακέτου χρησιμοποιήθηκε η μετρική Συνάφεια Μεταξύ των Κλάσεων των Πακέτων (CaPC - Cohesion among Package Classes). Αυτή η μετρική αξιολογεί πόσο στενά οι κλάσεις που ανήκουν στο ίδιο πακέτο συνεργάζονται μεταξύ τους. Η μέτρηση εμπνέεται από την αντιστροφή του υπολογισμού της έλλειψης συνοχής μεθόδων [16]. Για τον υπολογισμό αυτών των μετρικών υπολογίζεται ο συνολικός αριθμός των ζευγών κλάσεων που ανήκουν σε ένα πακέτο και στη συνέχεια διερευνάται το ποσοστό αυτών των ζευγών που είναι συνεπείς (δηλαδή που συνδέονται μεταξύ τους).

3.3 Σχετική Βιβλιογραφία

Κατά την αναζήτηση σχετικής βιβλιογραφίας παρατηρήθηκε, όπως αναφέρθηκε και στην εισαγωγή, ότι δεν υπήρχε πολύ υλικό για το αντικείμενο "μετακίνηση κλάσης" (move class). Πιο συγκεκριμένα, μετά από εκτενή έρευνα σε βιβλιοθήκες ερευνητικών εργασιών βρέθηκαν δύο εργασίες οι οποίες ασχολήθηκαν με τη μετακίνηση κλάσης. Η εξαγωγή κλάσης (extract class) είναι πιο διαδεδομένη διαδικασία για την οποία υπάρχει αρκετή βιβλιογραφία, ωστόσο αποτελεί κάτι διαφορετικό από τη μετακίνηση κλάσης και αναφέρεται για να μην προκληθούν θέματα παρανόησης. Παρακάτω ακολουθεί μία συνοπτική περιγραφή των δύο εργασιών.

Στο [18] προτείνετε ένα μοντέλο σύντηξης δεδομένων για το συνδυασμό πληροφοριών από διαφορετικές πηγές προκειμένου να εντοπιστούν ευκαιρίες αναδόμησης (refactoring) η οποίες αφορούν τη μετακίνηση μίας κλάσης από το ένα πακέτο σε κάποιο άλλο πιο σχετικό. Η μεθοδολογία που αναπτύχθηκε πραγματοποιεί σημασιολογική ανάλυση στις κλάσεις και τα πακέτα του λογισμικού και χρησιμοποιεί δομικές εξαρτήσεις για να προτείνει κλάσεις προς μετακίνηση σε κάποιο πιο κατάλληλο πακέτο. Η σημασιολογική ανάλυση γίνεται υπολογίζοντας την απόσταση συνημίτονου μεταξύ κάθε κλάσης και πακέτου. Ο υπολογισμός τον δομικών εξαρτήσεων θυμίζει τις μετρικές σύζευξης που αναλύθηκαν στο ίδιο κεφάλαιο (Κεφάλαιο 3) πιο πάνω. Μετρούνται οι σχέσεις από και προς κάθε κλάση. Τα δεδομένα αυτά αποθηκεύονται σε δύο πίνακες και στο τελευταίο στάδιο οι πίνακες αυτοί συνδυάζονται μεταξύ τους. Ελέγχεται η σημασιολογική και η δομική σχέση κάθε κλάσης προς κάθε πακέτο με τη χρήση της απόστασης του συνημίτονου και αυτό που δίνει καλύτερη τιμή (κοντά στο 1) θεωρείται ως η καλύτερη επιλογή. Αν η τιμή που προκύψει είναι μεγαλύτερη από ένα συγκεκριμένο όριο τότε το πρόγραμμα προτείνει τη μετακίνηση της κλάσης προς κάποιο άλλο πακέτο. Αν η τιμή είναι μεγαλύτερη από ένα δεύτερο όριο θεωρείται θόρυβος και αγνοείται. Τέλος, αν η απόσταση συνημίτονου είναι ίδια με το αρχικό πακέτο τότε το πρόγραμμα δεν προτείνει την μετακίνηση της κλάσης. Οι συγγραφείς καταλήγουν ότι η μεθοδολογία τους βελτιώνει την τμηματικότητα του κώδικα κατά 29% και οι προγραμματιστές που χρησιμοποιήθηκαν για την αξιολόγηση, εκτίμησαν τις προτάσεις κατά 70% χρήσιμες.

Στο [19] παρουσιάζεται ένας αλγόριθμος για τη βελτίωση του σχεδιασμού των προγραμμάτων αφαιρώντας τις κυκλικές εξαρτήσεις μεταξύ των πακέτων χωρίς να καταρρεύσει εντελώς η δομή τους. Αυτό επιτυγχάνεται με τη μετακίνηση κλάσεων μεταξύ

των πακέτων. Ο αλγόριθμος βασίζεται σε μια λειτουργία βαθμολόγησης που χρησιμοποιείται για την επιλογή των κλάσεων που πρόκειται να μετακινηθούν. Ως ασθενής κυκλική εξάρτηση ορίζεται όταν οι κλάσεις ενός πακέτου επικοινωνούν με τις κλάσεις ενός δεύτερου πακέτου και αντίστροφα χωρίς ωστόσο η σχέση επικοινωνίας να είναι αμφίδρομη. Ισχυρή εξάρτηση υπάρχει όταν οι σχέσεις είναι αμφίδρομες. Αρχικά, ο αλγόριθμος βρίσκει τις ισχυρές εξαρτήσεις από τα bytecodes και ορίζει ένα βάρος για την κάθε μία, το βάρος είναι ανάλογο του πλήθους των εξαρτήσεων. Ακολουθεί ανάλυση των σχέσεων με υψηλή βαθμολογία. Πιο συγκεκριμένα, ο αλγόριθμος υπολογίζει το νέο βάρος αν η κλάση προέλευσης μετακινηθεί στο πακέτο προορισμού και αν η κλάση προορισμού μετακινηθεί στο πακέτο προέλευσης. Επιλέγεται η περίπτωση που μειώνει τον αριθμό των κυκλικών εξαρτήσεων. Αν και η δύο προκαλούν μείωση, επιλέγεται η μεγαλύτερη μείωση. Αν δεν υπάρχει μείωση εξαρτήσεων, η κλάση δεν προτείνετε προς μετακίνηση. Αν μία μετακίνηση ικανοποιεί το κριτήριο να μειώσει τις εξαρτήσεις, τίθεται στο γράφο εξαρτήσεων και η διαδικασία επαναλαμβάνεται μέχρι να μην υπάρχουν άλλες ισχυρές εξαρτήσεις ή μέχρι έναν ορισμένο αριθμό επαναλήψεων. Η επικύρωση του αλγορίθμου έγινε σε διάφορες μελέτες περιπτώσεων ανοιχτού κώδικα. Τα αποτελέσματα δείχνουν ότι ο αλγόριθμος βελτιώνει τη δομή του προγράμματος και αφαιρεί τους κύκλους μεταξύ των πακέτων.

4. Μεθοδολογία

"If you can't explain it simply, you don't understand it well enough."
— Albert Einstein

Το πρόγραμμα που σχεδιάστηκε και υλοποιήθηκε αποτελείται από δύο βασικά τμήματα. Το πρώτο τμήμα είναι η συντακτική ανάλυση του λογισμικού εισόδου και ο υπολογισμός μετρικών σύζευξης και συνοχής. Το δεύτερο τμήμα χρησιμοποιεί αυτά τα δεδομένα με σκοπό να προταθούν βελτιώσεις που θα οδηγήσουν σε ένα επαναχρησιμοποιήσιμο, επεκτάσιμο, ευέλικτο και χωρίς λάθη λογισμικό.

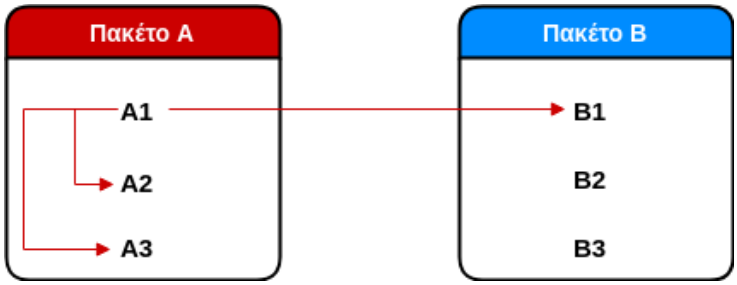
Στο πρώτο τμήμα, δίνεται ως είσοδος ένα πρόγραμμα λογισμικού προς ανάλυση. Ο αλγόριθμος αναγνωρίζει τα αρχεία .class και αποθηκεύει τις σχετικές πληροφορίες που περιέχονται στα bytecodes αυτών. Πιο συγκεκριμένα, αποθηκεύεται το όνομα του πακέτου στο οποίο βρίσκεται η κάθε κλάση, το όνομά της, τις μεθόδους και τα χαρακτηριστικά της. Σε λίστες αποθηκεύονται οι σχέσεις μεταξύ κλάσεων εντός του ίδιου πακέτου αλλά και οι σχέσεις μεταξύ των κλάσεων διαφορετικών πακέτων. Έχοντας συλλέξει όλη αυτή την πληροφορία, είναι δυνατό πλέον να υπολογιστούν οι μετρικές συνοχής και σύζευξης. Έστω δύο κλάσεις, A και B. Η κλάση A έχει εξάρτηση με την κλάση B (A → B). Ειδικότερα, σε επίπεδο κλάσης υπολογίζονται οι μετρικές REM, πλήθος μεθόδων που έχουν εξαρτήσεις από τη μία κλάση προς κάποια άλλη, πλήθος πολυμορφικών κλάσεων στη γονική κλάση, πλήθος προστατευμένων χαρακτηριστικών της υπερκλάσης που χρησιμοποιείται από την υποκλάση, ολικό πλήθος μεθόδων της κλάσης προορισμού (έστω B1 Εικόνα 8) και πλήθος χαρακτηριστικών της κλάσης προέλευσης (έστω A1 Εικόνα 8). Όλα τα παραπάνω δεδομένα εξάγονται από τα bytecodes εκτός από τη μετρική REM η οποία υπολογίζεται από το πρόγραμμα με βάση αυτά τα δεδομένα. Σε επίπεδο πακέτου, υπολογίζεται ο παράγοντας διάδοσης, δηλαδή η μετρική MCPM. Οι μετρικές REM και MCPM παρουσιάστηκαν και εξηγήθηκαν στο Υποκεφάλαιο 3.2.4.

Στο δεύτερο τμήμα, έχοντας συλλέξει και υπολογίσει όλες τις παραπάνω πληροφορίες, το πρόγραμμα είναι σε θέση να μελετήσει τους πιθανούς συνδυασμούς κλάσεων για τη δημιουργία ομάδων. Για λόγους απλούστευσης και κατανόησης, ο αλγόριθμος θα αναλυθεί με τη βοήθεια παραδείγματος και στη συνέχεια θα ακολουθήσει ένα επεξηγηματικό παράδειγμα.

Μετακίνηση μίας κλάσης από πακέτο Α προς πακέτο Β:

Ο αλγόριθμος διατρέχει τη λίστα αντικειμένων και ελέγχει μία μία τις συνδέσεις μεταξύ πακέτων. Υπενθυμίζεται ότι, στο πρώτο μέρος παράχθηκε λίστα σύνδεσης πακέτων η οποία περιέχει λίστα σύνδεσης κλάσεων. Για κάθε σύνδεση μεταξύ δύο πακέτων, γίνεται ανάλυση της λίστας με τις σχέσεις μεταξύ κλάσεων, δηλαδή τις κλάσεις που επικοινωνούν από το πακέτο Α προς το πακέτο Β. Έστω η κλάση Α1 ανήκει στο πακέτο Α και η κλάση Β1 ανήκει στο πακέτο Β. Για κάθε μία σχέση δημιουργείται ένα νέο αντικείμενο κλάσης τύπου Refactoring, το οποίο περιλαμβάνει όλες τις απαραίτητες πληροφορίες και μετρικές για τη συγκεκριμένη μετακίνηση (η κλάση Α1 να μετακινηθεί στο πακέτο Β). Οι μετρικές αυτές είναι οι προαναφερθέντες και προστίθεται η αρχική τιμή της μετρικής έντασης ή ελαφριάς σύζευξης (Ce), δηλαδή η τιμή πριν την μετακίνηση της κλάσης. Η κλάση Refactoring είναι μία νέα κλάση η οποία δημιουργήθηκε για την καταγραφή δεδομένων κλάσεων προς μετακίνηση. Στη συνέχεια, υπολογίζεται η νέα ένταση ή ελαφριά σύζευξη μεταξύ των πακέτων Α → Β, δηλαδή η σύζευξη αφού η κλάση Α1 μετακινηθεί στο πακέτο Β. Η μετρική αυτή υπολογίζεται ως εξής: Από την αρχική ένταση αφαιρείται το πλήθος εξαρτήσεων που επιλύθηκαν και προστίθεται το άθροισμα του πλήθους εξαρτήσεων που δημιουργείται από τη μετακίνηση με το πλήθος εξαρτήσεων που δεν επιλύθηκαν. Οι νέες εξαρτήσεις που δημιουργούνται στη σχέση Α → Β είναι οι κλάσεις από το πακέτο Α που επικοινωνούν με την κλάση Α1. Οι κλάσεις με τις οποίες επικοινωνεί η Α1 προστίθενται στη σχέση πακέτων Β → Α. Γίνονται οι αντίστοιχοι υπολογισμοί και για αυτή τη σχέση πακέτων, Β → Α. Ακολουθεί έλεγχος της αντίστροφης μετακίνησης, δηλαδή η κλάση Β1 να μετακινηθεί στο πακέτο Α και ξανά υπολογίζονται οι μετρικές που αναφέρθηκαν για πακέτα τα Α → Β και τα πακέτα Β → Α.

Ο αμφίδρομος αυτός έλεγχος προστέθηκε ώστε να καλυφθούν όλες οι περιπτώσεις και να ελεγχθεί η επίδραση και στα δύο μέρη. Ακολουθεί ένα μικρό παράδειγμα που εξηγεί γιατί αυτός ο έλεγχος είναι σημαντικός. Το παράδειγμα φαίνεται πιο κάτω, στην Εικόνα 8. Η κλάση Α1 του πακέτου Α καλεί την κλάση Β1 του πακέτου Β. Ωστόσο η κλάση Β1 δεν επικοινωνεί με καμία άλλη από το πακέτο της οπότε η μεταφορά της κλάσης Β1 προς το πακέτο Α αντί της μεταφοράς της κλάσης Α1 στο πακέτο Β είναι πιο κερδοφόρα καθώς λύνει την εξάρτηση μεταξύ των δύο πακέτων χωρίς να δημιουργεί καινούργιες.



Εικόνα 8 : Παράδειγμα Σημαντικότητας Αμφίδρομου Ελέγχου

Συνεχίζοντας, για κάθε μετακίνηση υπολογίζεται ο νέος παράγοντας διάδοσης (MCPM) μεταξύ των πακέτων. Η διαδικασία αυτή επαναλαμβάνεται για όλες τις σχέσεις μεταξύ όλων των ζευγών πακέτων. Μόλις συλλεχθούν και υπολογιστούν όλα τα παραπάνω δεδομένα, η λίστα αντικειμένων τύπου Refactoring είναι έτοιμη και οι προτάσεις μετακίνησης μίας κλάσης έχουν ολοκληρωθεί και αποθηκευτεί σε αρχείο.

Μετακίνηση ομάδας κλάσεων από πακέτο Α προς πακέτο Β:

Για τον υπολογισμό ομάδων κλάσεων προς μετακίνηση υλοποιήθηκε μία νέα συνάρτηση η οποία καλείται αναδρομικά τόσες φορές όσο το πλήθος των ομάδων που επιθυμεί ο χρήστης, έστω n. Ο υπολογισμός ξεκινάει για i = 2 και τερματίζει όταν δεν υπάρχουν άλλες κλάσεις οι οποίες συνδέονται και μπορούν να δημιουργήσουν ομάδα ή όταν το i πάρει την τιμή που επέλεξε ο χρήστης, δηλαδή n. Η συνάρτηση αυτή εκτελείται μετά την εκτέλεση της πιο πάνω συνάρτησης, δηλαδή του υπολογισμού μετακίνησης μία κλάσης και δέχεται ως όρισμα τη λίστα αντικειμένων Refactoring η οποία παράχθηκε επίσης στην προηγούμενη συνάρτηση.

Η επιλογή της ομάδας κλάσεων προς μετακίνηση γίνεται ως εξής: Αρχικά ο αλγόριθμος ξεκινάει με την περίπτωση i = 2. Η συνάρτηση δέχεται ως παράμετρο εισόδου τη λίστα αντικειμένων η οποία παράχθηκε νωρίτερα και τη λίστα με όλες τις σχέσεις και εξαρτήσεις μεταξύ πακέτων και κλάσεων. Η πρώτη λίστα περιέχει τις πιθανές μετακινήσεις μίας κλάσης και τις πληροφορίες που τη διέπουν όπως οι μετρικές σύζευξης και συνοχής, οι κλάσεις τις οποίες καλεί και οι οποίες την καλούν από το πακέτο στο οποίο βρίσκεται. Επομένως, η επιλογή κλάσεων για τη δημιουργία ομάδας γίνεται από τις κλάσεις τις οποίες καλεί η κλάση προς εξέταση και από τις κλάσεις οι οποίες την καλούν. Παράγονται όλα τα πιθανά ζευγάρια τα οποία μπορούν να προκύψουν με τις παραπάνω κλάσεις. Στη συνέχεια,

αναζητούνται οι εξαρτήσεις οι οποίες επιλύονται με τη μεταφορά κάθε ζεύγους κλάσεων και ακολούθως υπολογίζονται οι νέες εξαρτήσεις οι οποίες δημιουργούνται. Έχοντας αυτές τις πληροφορίες το λογισμικό είναι σε θέση να υπολογίσει τις νέες μετρικές όπως το νέο MCPM, τη νέα ένταση - ελαφριά σύζευξη και να δημιουργήσει τα αντικείμενα με τις νέες προτάσεις για ομάδες των δύο κλάσεων. Για κάθε μία ομάδα κλάσεων υλοποιείται και ο αμφίδρομος έλεγχος, δηλαδή, έστω οι κλάσεις που χρησιμοποιήθηκαν προηγουμένως, η κλάση A1 μαζί με την κλάση A2 να μετακινηθεί στο πακέτο B και η κλάση B1 να μετακινηθεί στο πακέτο A. Υπολογίζονται εκ νέου οι μετρικές των σχέσεων πακέτων A → B και B → A.

Μόλις ολοκληρωθεί αυτή η διαδικασία, τα δεδομένα αποθηκεύονται σε αρχείο και η συνάρτηση καλεί τον εαυτό της αναδρομικά με τη νέα λίστα αντικειμένων (αυτή που μόλις παράχθηκε με τις ομάδες δύο κλάσεων) προκειμένου να δημιουργηθούν ομάδες κλάσεων i=3,...n μέχρι δηλαδή, να ικανοποιηθεί κάποιο από τα κριτήρια τερματισμού.

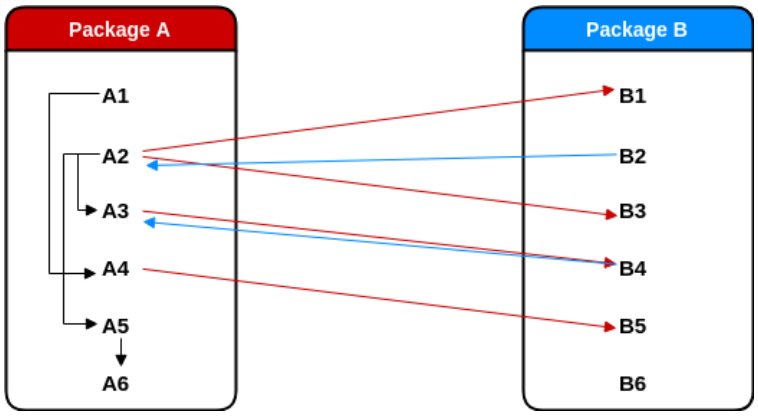
Μόλις ολοκληρωθεί η όλη διαδικασία, η συνάρτηση επιστρέφει στην αρχική συνάρτηση, αποθηκεύει όλα τα δεδομένα σε αρχεία και το πρόγραμμα τερματίζει. Για κάθε πλήθος ομάδων δημιουργείται νέο αρχείο ώστε τα δεδομένα να είναι σε πιο ευανάγνωστη και επεξεργάσιμη μορφή.

Πολύ σημαντικό να αναφερθεί ότι η τεχνική ωμής βίας (brute force) που χρησιμοποιείται για την παραγωγή ομάδων κλάσεων έχει ως αποτέλεσμα να δημιουργούνται πολλά διπλότυπα. Το γεγονός αυτό έχει προβλεφθεί οπότε μόλις ολοκληρωθεί η λίστα αντικειμένων, τα διπλότυπα διαγράφονται και ως είσοδος στην αναδρομική συνάρτηση δίνεται η επεξεργασμένη λίστα, δηλαδή αυτή που περιέχει μοναδικές εγγραφές. Τέλος, στα αρχεία αποθηκεύονται μόνο οι μετακινήσεις που έχουν ως αποτέλεσμα θετική ένταση ή ελαφριά σύζευξη (Ce).

Παράδειγμα μεθοδολογίας :

Λεπτομερής εξήγηση βήμα προς βήμα της διαδικασίας επιλογής και δημιουργίας ομάδας κλάσεων θα παρουσιαστεί μέσω ενός επεξηγηματικού παραδείγματος, ως εξής: Θεωρείται ότι εφαρμόζεται η προτεινόμενη μεθοδολογία σε ένα λογισμικό το οποίο αποτελείται από δύο πακέτα (Πακέτο A και Πακέτο B) όπως παρουσιάζεται στην Εικόνα 9. Στην Εικόνα 9, το Πακέτο A αποτελείται από έξι κλάσεις τις A1, A2, A3, A4, A5 και A6 ενώ το Πακέτο B αποτελείται επίσης από έξι κλάσεις με τα εξής ονόματα B1, B2, B3, B4, B5 και

Β6. Οι σχέσεις μεταξύ των κλάσεων των δύο πακέτων φαίνονται επίσης στην Εικόνα 9. Με κόκκινο παρουσιάζονται οι σχέσεις από τις κλάσεις του Πακέτου Α ενώ με μπλε παρουσιάζονται οι σχέσεις από τις κλάσεις του Πακέτου Β.



Εικόνα 9 : Παράδειγμα Μεθοδολογίας

Σημειώνεται ότι οι σχέσεις που προκύπτουν από εσωτερικές κλάσεις έχουν ενσωματωθεί ως δεδομένα των βασικών κλάσεων. Θεωρείται ότι η κλάση Α2 έχει δύο εσωτερικές κλάσεις, τις Α21 και Α22. Έστω η Α21 καλεί την κλάση Β1 και όχι η Α2 όπως φαίνεται στο σχήμα παραπάνω, η σχέση αυτή ενσωματώθηκε στην κλάση Α2. Προφανώς επειδή είναι εσωτερική κλάση δεν μπορεί να πραγματοποιηθεί καμία αλλαγή που δεν προέρχεται από τη βασική κλάση Α2 καθώς θα προκληθούν πολλά νέα προβλήματα τα οποία τελικά θα χειροτερέψουν την κατάσταση από πλευράς σύζευξης και συνοχής. Η τεχνική που χρησιμοποιείται εδώ είναι μετακίνηση κλάσης και όχι εξαγωγή οπότε, οτιδήποτε προκαλείται από τις εσωτερικές κλάσεις προστίθενται στις βασικές και δεν γίνεται διαχωρισμός. Στο συγκεκριμένο παράδειγμα, για λόγους απλότητας θα μετρηθεί μόνο η μετρική έντασης (Ce) μεταξύ των πακέτων Α και Β.

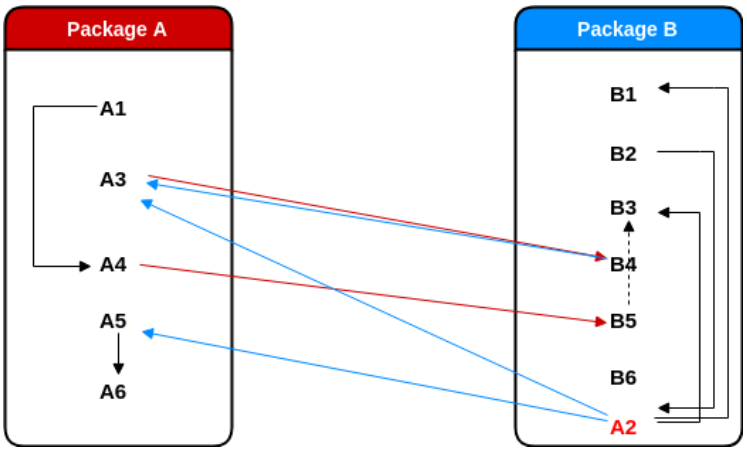
Η αρχική ένταση από το πακέτο Α προς το Β είναι 4 (σχέσεις Α2 → Β1, Α2 → Β3, Α3 → Β4 και Α4 → Β5) ενώ η αρχική ένταση από το πακέτο Β προς το πακέτο Α είναι 2 (σχέσεις Β2 → Α2 και Β4 → Α3). Θα γίνει ανάλυση των σχέσεων Α2 → Β1 και Α2 → Β3. Στη συνέχεια, για να διευκολυνθεί η κατανόηση των βημάτων του αλγορίθμου, απεικονίζονται λεπτομερώς σε μορφή πινάκων και σχεδιαγραμμάτων. Το μέγιστο μέγεθος ομάδας στο συγκεκριμένο παράδειγμα είναι 4 όπως θα αποδειχθεί στη συνέχεια.

Αρχικά, υπολογίζεται η επίδραση μεταφοράς μίας κλάσης από το πακέτο Α προς το πακέτο Β. Η κλάση Α2 του πακέτου Α καλεί την κλάση Β1 και Β3 του πακέτου Β. Για την πρώτη σχέση (Α2 → Β1), αν η κλάση Α2 μεταφερθεί στο πακέτο Β θα επιλυθούν δύο

εξαρτήσεις (προς B1 και B3) και θα δημιουργηθούν δύο νέες εξαρτήσεις προς το πακέτο A, οι A2 → A3 και A2 → A5. Οι νέες εξαρτήσεις προς την κλάση A2 είναι μηδέν. Επομένως, η νέα ένταση θα είναι 4 - 2 = 2. Οι δύο νέες εξαρτήσεις προκύπτουν στη σχέση πακέτου B προς πακέτο A και όχι το αντίθετο. Στο συγκεκριμένο παράδειγμα αναλύεται μόνο η σχέση πακέτου A προς το πακέτο B. Ωστόσο, η μεθοδολογία που αναπτύχθηκε υπολογίζει και τις επιπτώσεις στη σχέση πακέτο B προς πακέτο A στην περίπτωση όπου μεταφέρεται μία κλάση. Προφανώς, τα ίδια ισχύουν και για την σχέση A2 → B3. Ο Πίνακας 12 περιέχει τις πληροφορίες που υπολογίζονται από το πρόγραμμα και όπως αναφέρεται πιο πάνω αποθηκεύονται σε αντικείμενο της κλάσης Refactoring. Ασφαλώς, στο παράδειγμα δεν απεικονίζονται όλες οι μετρικές αλλά μόνο αυτές που χρειάζονται για την κατανόησή του. Η Εικόνα 10 απεικονίζει τη νέα αρχιτεκτονική των πακέτων A και B μετά την μετακίνηση. Επειδή η μετακίνηση είναι μόνο της κλάσης A2 αλλά προς δύο εξαρτήσεις, η αρχιτεκτονική είναι ίδια και για την εξάρτηση A2 → B1 και για την εξάρτηση A2 → B3.

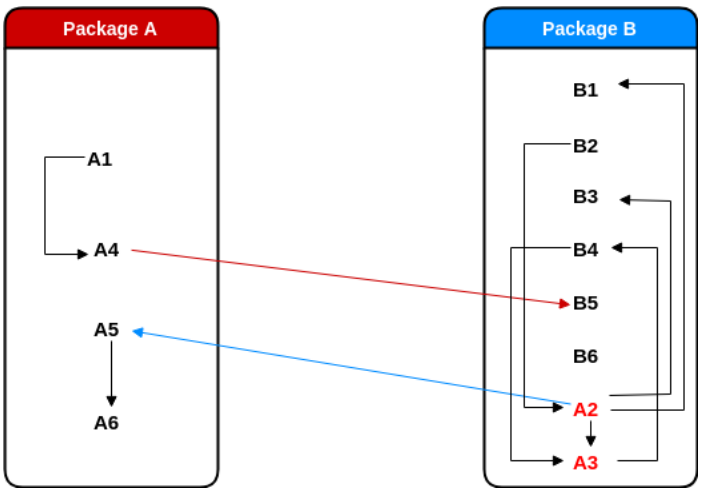
Πίνακας 12 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Μίας Κλάσης

Κλάση Από	Κλάση Προς	Πακέτο Από	Πακέτο Προς	Ν.Ε.	Πλήθος Ε.Ε.	Πλήθος Ν.Ε.	Σύζευξη Πριν	Σύζευξη Μετά
Ομάδα 1 κλάσης								
A2	B1	A	B	A3, A5	2	2	4	2
A2	B3	A	B	A3, A5	2	2	4	2



Εικόνα 10 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Μίας Κλάσης

Στη συνέχεια, δημιουργούνται ζευγάρια μεταξύ των κλάσεων από το πακέτο προέλευσης (A) και των εξαρτήσεων τους, προς τις κλάσεις του πακέτου Α όπως φαίνεται στον Πίνακα 13. Ως πρώτο ζευγάρι επιλέγεται η Α2 με την Α3 προς την εξάρτηση Β1. Η μετακίνηση αυτών των δύο κλάσεων μαζί επιλύει τρεις εξαρτήσεις προς το Πακέτο Β (Α2 → Β1, Α2 → Β3, Α3 → Β4) και δημιουργεί μία νέα από το πακέτο Β προς το πακέτο Α, την (Α2 → Α5). Οι εξαρτήσεις προς τις κλάσεις Α2 και Α3 είναι μηδέν. Οπότε η νέα ένταση είναι 4 - 3 = 1. Η παραπάνω διαδικασία εκτελείται επίσης για την εξάρτηση προς την κλάση Β3. Η αρχιτεκτονική των πακέτων Α και Β μετά από αυτή τη μετακίνηση απεικονίζεται στην Εικόνα 11. Επειδή οι κλάσεις προέλευσης (Α2, Α3) είναι ίδιες και στις δύο εξαρτήσεις (Β1, Β3), η αρχιτεκτονική είναι επίσης η ίδια.

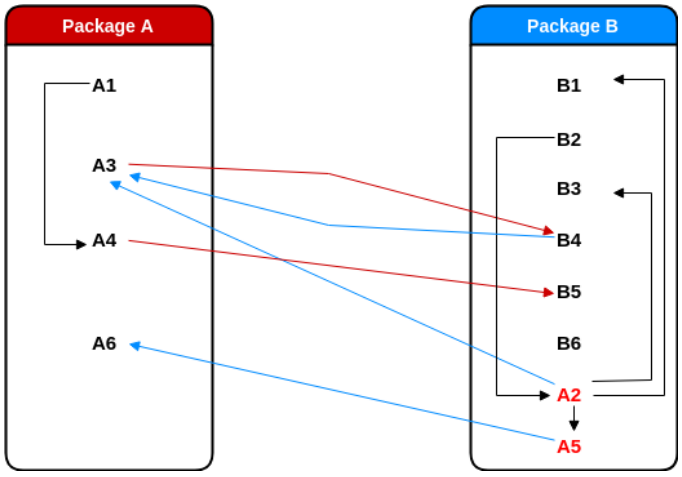


Εικόνα 11 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Δύο Κλάσεων

Ακολουθεί το ζευγάρι Α2, Α5 προς την κλάση Β1. Η κλάση Α2 καλεί τις κλάσεις Α3 και Α5, επειδή όμως με την Α5 θα μετακινηθούν μαζί αυτή η εξάρτηση λύνεται και μένει μόνο με την Α3. Η κλάση Α5 καλεί την κλάση Α6 οπότε, προστίθεται και αυτή η εξάρτηση στο σύνολο των εξαρτήσεων. Άρα δημιουργούνται δύο νέες εξαρτήσεις με τη μεταφορά των κλάσεων από το πακέτο Α προς το πακέτο Β. Οι εξαρτήσεις που επιλύονται είναι 2 και προέρχονται από την κλάση Α2 (Α2 → Β1, Α2 → Β3). Άρα η νέα ένταση Α → Β είναι 4 - 2 = 2. Όμοια για την εξάρτηση των κλάσεων Α2, Α5 προς την κλάση Β3. Η νέα αρχιτεκτονική των πακέτων Α και Β φαίνεται στην Εικόνα 12. Επειδή οι κλάσεις προέλευσης (Α2, Α5) είναι ίδιες και στις δύο εξαρτήσεις (Β1, Β3), η αρχιτεκτονική είναι επίσης η ίδια.

Πίνακας 13 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Δύο Κλάσεων

Κλάση Από	Κλάση Προς	Πακέτο Από	Πακέτο Προς	Ν.Ε.	Πλήθος Ε.Ε.	Πλήθος Ν.Ε.	Σύζευξη Πριν	Σύζευξη Μετά
Ομάδες 2 κλάσεων								
A2, A3	B1	A	B	A5	3	1	4	1
A2, A3	B3	A	B	A5	3	1	4	1
A2, A5	B1	A	B	A3, A6	2	2	4	2
A2, A5	B3	A	B	A3, A6	2	2	4	2

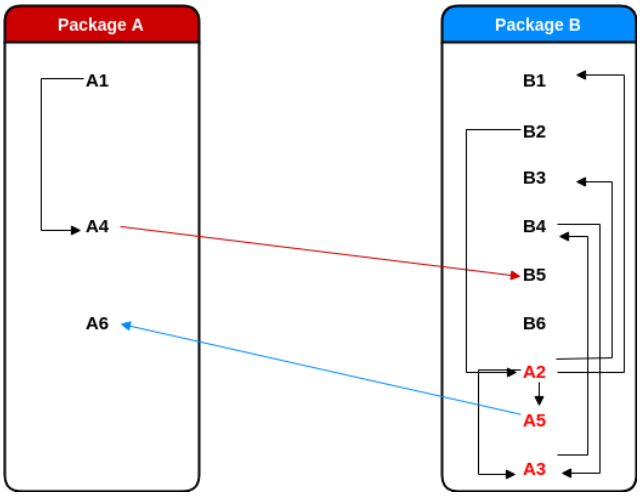


Εικόνα 12 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Δύο Κλάσεων

Στη συνέχεια, δημιουργούνται ομάδες τριών κλάσεων με βάση το κριτήριο που αναφέρθηκε παραπάνω, δηλαδή μεταξύ των κλάσεων από το πακέτο προέλευσης (A) και των εξαρτήσεων τους, προς τις κλάσεις του πακέτου A. Όλες οι ομάδες και οι υπολογισμοί των μετρικών φαίνεται στον Πίνακα 14. Ως πρώτη ομάδα επιλέγεται η A2, η A3 και η A5 προς την εξάρτηση B1. Η μετακίνηση αυτών των τριών κλάσεων μαζί επιλύει τρεις εξαρτήσεις προς το πακέτο B (A2 → B1, A2 → B3, A3 → B4) και δημιουργεί μία νέα από το πακέτο B προς το πακέτο A, την (A5 → A6). Οι εξαρτήσεις προς τις κλάσεις A2, A3 και A5 είναι μηδέν. Οπότε η νέα ένταση είναι 4 - 3 = 1. Η παραπάνω διαδικασία εκτελείται επίσης για την εξάρτηση προς την κλάση B3. Η αρχιτεκτονική των πακέτων A και B μετά από αυτή τη μετακίνηση απεικονίζεται στην Εικόνα 13. Επειδή οι κλάσεις προέλευσης (A2, A3, A5) είναι ίδιες και στις δύο εξαρτήσεις (B1, B3) η αρχιτεκτονική είναι επίσης η ίδια.

Πίνακας 14 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Τριών Κλάσεων

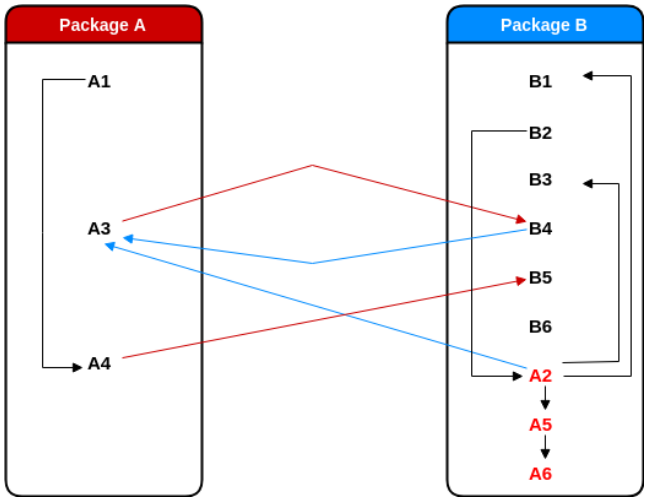
Κλάση Από	Κλάση Προς	Πακέτο Από	Πακέτο Προς	Ν.Ε.	Πλήθος Ε.Ε.	Πλήθος Ν.Ε.	Σύζευξη Πριν	Σύζευξη Μετά
Ομάδες 3 κλάσεων								
A2, A3, A5	B1	A	B	A6	3	1	4	1
A2, A3, A5	B3	A	B	A6	3	1	4	1
A2, A5, A3	B1	A	B	A6	3	1	4	1
A2, A5, A6	B1	A	B	A3	2	1	4	2
A2, A5, A3	B3	A	B	A6	3	1	4	1
A2, A5, A6	B3	A	B	A3	2	1	4	2



Εικόνα 13 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Τριών Κλάσεων

Η επόμενη ομάδα που επιλέγεται περιέχει της κλάσεις A2, A5 και A6 προς την εξάρτηση B1. Η μετακίνηση αυτών των τριών κλάσεων μαζί επιλύει δύο εξαρτήσεις προς το πακέτο B (A2 → B1, A2 → B3) και δημιουργεί μία νέα εξάρτηση από το πακέτο B προς το πακέτο A, την A2 → A3 . Οι εξαρτήσεις προς τις κλάσεις A2, A5 και A6 είναι μηδέν. Οπότε η νέα ένταση είναι 4 - 2 = 2. Η παραπάνω διαδικασία εκτελείται επίσης για την εξάρτηση προς την κλάση B3. Η αρχιτεκτονική των πακέτων A και B μετά από αυτή τη μετακίνηση

απεικονίζεται στην Εικόνα 14. Επειδή οι κλάσεις προέλευσης (A2, A5, A6) είναι ίδιες και στις δύο εξαρτήσεις (B1, B3) η αρχιτεκτονική είναι επίσης η ίδια.



Εικόνα 14 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Τριών Κλάσεων

Τέλος, η ομάδα τεσσάρων κλάσεων που είναι δυνατόν να δημιουργηθεί είναι μόνο μία και αποτελείται από τις κλάσεις A2, A3, A5 και A6 προς την εξάρτηση B1. Ο Πίνακας 15 απεικονίζει όλους τους πιθανούς συνδυασμούς και υπολογισμούς. Η μετακίνηση αυτών των τεσσάρων κλάσεων μαζί επιλύει τρεις εξαρτήσεις προς το πακέτο B (A2 → B1, A2 → B3, A3 → B4) και δεν δημιουργεί καμία νέα εξάρτηση από το πακέτο B προς το πακέτο A. Οι εξαρτήσεις προς τις κλάσεις A2, A3, A5 και A6 είναι μηδέν. Οπότε η νέα ένταση είναι 4 - 3 = 1. Η παραπάνω διαδικασία εκτελείται επίσης για την εξάρτηση προς την κλάση B3. Η αρχιτεκτονική των πακέτων A και B μετά από αυτή τη μετακίνηση απεικονίζεται στην Εικόνα 15. Επειδή οι κλάσεις προέλευσης (A2, A3, A5, A6) είναι ίδιες και στις δύο εξαρτήσεις (B1, B3) η αρχιτεκτονική είναι επίσης η ίδια.

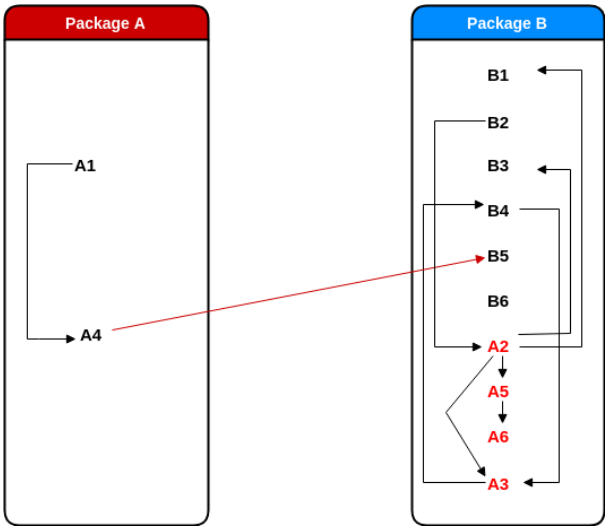
Η διαδικασία αυτή επαναλαμβάνεται μέχρι να μην μπορούν να δημιουργηθούν μεγαλύτερες ομάδες κλάσεων ή μέχρι ένα πλήθος που έχει επιλέξει ο χρήστης. Στο συγκεκριμένο παράδειγμα ο αλγόριθμος τερματίζει καθώς, το πλήθος των νέων εξαρτήσεων είναι μηδέν. Παρατηρείται ότι, μετά το πέρας της μετακίνησης τεσσάρων κλάσεων όλες οι εξαρτήσεις της σχέσης πακέτων B → A επιλύονται (δηλαδή από δύο γίνονται μηδέν) ενώ οι εξαρτήσεις της σχέσης πακέτων A → B από τέσσερεις γίνονται μία. Επομένως ο σκοπός της μεθοδολογίας επιτεύχθηκε, η τμηματικότητα σε επίπεδο πακέτου βελτιώθηκε.

Είναι σημαντικό να αναφερθεί, ότι παράγονται πολλά διπλότυπα καθώς ο αλγόριθμος ελέγχει όλες τις πιθανές ομάδες κλάσεων. Η λύση που χρησιμοποιήθηκε για να μειώσει την

πολυπλοκότητα και το χρόνο εκτέλεσης χωρίς ωστόσο να χαθούν περιπτώσεις είναι, πριν την αναδρομική κλήση και την εμφάνιση των αποτελεσμάτων, όλα τα διπλότυπα να διαγράφονται. Στους πίνακες αυτού του κεφαλαίου τα διπλότυπα δεν αφαιρέθηκαν για να γίνει πιο κατανοητός ο τρόπος δημιουργίας ομάδων κλάσεων. Σε μελλοντική εργασία προβλέπετε βελτιστοποίηση του τρόπου επιλογής ομάδων κλάσεων και σε αλγοριθμικό επίπεδο και σε επίπεδο χρόνου εκτέλεσης αλγορίθμου.

Πίνακας 15 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Τεσσάρων Κλάσεων

Κλάση Από	Κλάση Προς	Πακέτο Από	Πακέτο Προς	Ν.Ε.	Πλήθος Ε.Ε.	Πλήθος Ν.Ε.	Σύζευξη Πριν	Σύζευξη Μετά
Ομάδες 4 κλάσεων								
A2, A3, A5, A6	B1	A	B	-	3	0	4	1
A2, A3, A5, A6	B3	A	B	-	3	0	4	1
A2, A5, A3, A6	B1	A	B	-	3	0	4	1
A2, A5, A6, A3	B1	A	B	-	3	0	4	1
A2, A5, A3, A6	B3	A	B	-	3	0	4	1
A2, A5, A6, A3	B3	A	B	-	3	0	4	1



Εικόνα 15 : Παράδειγμα Μεθοδολογίας - Μετακίνηση Πέντε Κλάσεων

Η βελτίωση της έντασης ή ελαφριάς σύζευξης (Ce) προκύπτει από το αν η νέα ένταση- σύζευξη είναι μικρότερη από την αρχική. Στο συγκεκριμένο παράδειγμα, σε όλες τις περιπτώσεις η κατάσταση βελτιώνεται. Παρατηρείται ότι, όσο αυξάνεται το πλήθος της ομάδας κλάσεων προς μετακίνηση, τόσο βελτιώνεται η ένταση. Το συμπέρασμα αυτό είναι αναμενόμενο εφόσον με μεγαλύτερες ομάδες μειώνονται οι εξαρτήσεις προς και εντός του πακέτου από το οποίο προέρχονται (στο παράδειγμα Πακέτο Α). Εντούτοις, αν το πλήθος αυξηθεί πολύ, οδηγούμαστε σε μία κατάσταση “θεϊκού” πακέτου το οποίο περιέχει μεγάλο πλήθος κλάσεων με αποτέλεσμα να χάνεται η συνοχή. Δεν υπάρχει κάποιος καθολικός κανόνας για το πλήθος των ομάδων κλάσεων, κάθε λογισμικό είναι διαφορετικό και η τελική επιλογή ή η βέλτιστη επιλογή αφήνεται στην ευχέρεια του προγραμματιστή ή του ερευνητή.

Όπως θα αποδειχθεί στο Κεφάλαιο 7, το κεφάλαιο των αποτελεσμάτων, βελτίωση των μετρικών παρατηρείται στις ακραίες τιμές. Δηλαδή στις περιπτώσεις μετακίνησης μίας ή πέντε κλάσεων. Ο αριθμός πέντε επιλέχθηκε τυχαία, σε μεγάλα προγράμματα είναι ένα αποδεκτό πλήθος μετακίνησης αλλά σε μικρά προγράμματα με λίγες κλάσεις έχει ως αποτέλεσμα τα πακέτα να αδειάζουν.

5. Υλοποίηση

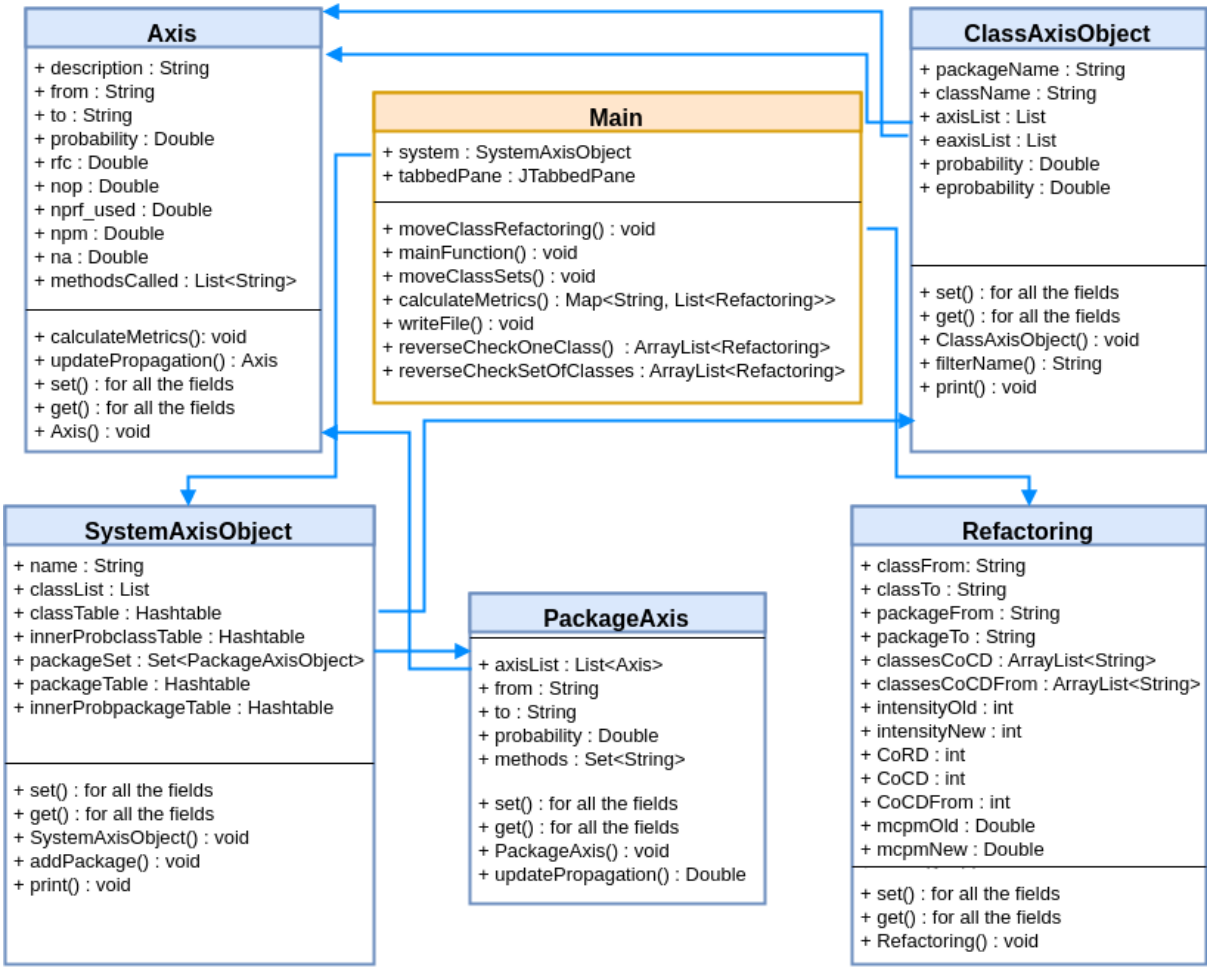
"I would love to change the world, but they won't give me the source code."

Ο αλγόριθμος που αναλύθηκε στο Κεφάλαιο 4 υλοποιήθηκε σε γλώσσα προγραμματισμού Java με τη χρήση του προγράμματος Eclipse το οποίο, είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) λογισμικού. Το σύστημα στο οποίο το πρόγραμμα αναπτύχθηκε και εκτελέστηκε ήταν περιβάλλον Gnu Linux και συγκεκριμένα ο υπολογιστής ανάπτυξης έτρεχε λειτουργικό σύστημα Ubuntu ενώ ο διακομιστής που δοκιμάστηκε το εργαλείο έτρεχε λειτουργικό σύστημα Debian. Οι απαιτήσεις μνήμης (ram) του υλικού ήταν πολύ υψηλές και αυτό είχε ως αποτέλεσμα να χρειαστούν πολλές βελτιστοποιήσεις και αλλαγές διότι, ο χρόνος εκτέλεσης ήταν υψηλός. Η διαδικασία αυτή αποτέλεσε τροχοπέδη για την εύρυθμη διεξαγωγή των αποτελεσμάτων ωστόσο, έδειξε πολλά σημεία που χρειαζόντουσαν βελτιστοποίηση και οδήγησε σε καλύτερη κατανόηση και επεξεργασία των δεδομένων. Κάποιες ενέργειες βελτιστοποίησης υλοποιήθηκαν, όπως η διαγραφή των διπλότυπων (ο αλγόριθμος δουλεύει με ωμή βία - brute force, δημιουργεί όλους τους πιθανούς συνδυασμούς ομάδων κλάσεων), και ο έλεγχος του πλήθους εγγραφών σε αρχείο απορρίπτοντας δεδομένα που δεν προσφέρουν χρήσιμη πληροφορία. Στο κοντινό μέλλον, στα πλαίσια μιας ερευνητικής εργασίας, όλα τα σημεία που χρήζουν βελτιστοποίηση θα βελτιστοποιηθούν για την ομαλή εκτέλεση του αλγορίθμου και την ομαλή διεξαγωγή της έρευνας και των αποτελεσμάτων.

5.1 Διάγραμμα Κλάσεων

Το λογισμικό που αναπτύχθηκε αποτελείται από τρία βασικά πακέτα. Η αρμοδιότητα του πρώτου πακέτου είναι να εξάγει και να μετατρέπει τις πληροφορίες από τα bytecodes σε χρήσιμη πληροφορία. Το δεύτερο πακέτο έχει ως αρμοδιότητα το γραφικό περιβάλλον και τη βασική συνάρτηση main, ενώ το τρίτο πακέτο, αυτό που θα παρουσιαστεί, είναι υπεύθυνο για την υλοποίηση της μεθοδολογίας. Στην Εικόνα 16, απεικονίζεται το διάγραμμα κλάσεων του τρίτου πακέτου και συγκεκριμένα οι κλάσεις οι οποίες έχουν βασικό ρόλο στην υλοποίηση της μεθοδολογίας. Παρακάτω, ακολουθεί ανάλυση του τρόπου ανάπτυξης του αλγορίθμου και του τρόπου εκτέλεσης της μεθοδολογίας. Τα άλλα δύο πακέτα δεν θα

αναλυθούν καθώς, οι υλοποιήσεις τους έχουν πιο καθολικό χαρακτήρα και βασίζονται σε βιβλιοθήκες συστήματος.



Εικόνα 16 : Διάγραμμα Κλάσεων

Το σύστημα ακολουθεί ιεραρχική δομή δεδομένων, οπότε η επεξήγηση θα ξεκινήσει από το χαμηλότερο επίπεδο προς το υψηλότερο (ή αλλιώς από μέσα προς τα έξω). Αρχικά, η κλάση Axis περιέχει τις εξαρτήσεις μεταξύ κλάσεων. Πιο συγκεκριμένα, διαθέτει ως χαρακτηριστικά την κλάση προέλευσης και προορισμού, την εσωτερική πιθανότητα ή πιθανότητα φαινομένου κυματισμού (rem), το πλήθος των μεθόδων που καλούνται μεταξύ των κλάσεων (rfc), το πλήθος πολυμορφικών κλάσεων στην υπερκλάση (nop), το πλήθος προστατευμένων πεδίων στην υπερκλάση που χρησιμοποιείται από την κλάση προέλευσης (nprf_used), το πλήθος μεθόδων στην κλάση προορισμού (npm), το πλήθος χαρακτηριστικών (na) και τέλος μία λίστα με όλες τις μεθόδους. Οι τιμές των πεδίων τύπου Double υπολογίζονται από τη συνάρτηση calculateMetrics. Είναι σημαντικό να αναφερθεί ότι όλες οι

παραπάνω μεταβλητές απαιτούνται για τον υπολογισμό της εσωτερικής πιθανότητας (rem). Επίσης, η κλάση περιλαμβάνει τη συνάρτηση updatePropagation μέσω της οποίας οποιαδήποτε αλλαγή ανανεώνει τις τιμές των μετρικών. Επιπλέον, περιλαμβάνει συναρτήσεις προσθήκης, διαγραφής και τροποποίησης των σχέσεων μεταξύ κλάσεων. Τέλος, κάθε χαρακτηριστικό διαθέτει τις συναρτήσεις ανάθεσης (set) και τις συναρτήσεις εξαγωγής (get).

Η κλάση PackageAxis έχει ως χαρακτηριστικά το όνομα πακέτου προορισμού και προέλευσης, τις μεθόδους που καλούνται, τον παράγοντα διάδοσης μεταξύ των δύο πακέτων (probability, MCPM) και τις εξαρτήσεις μεταξύ κλάσεων που διέπουν τα πακέτα ανά δύο (axisList). Οι ιδιότητες της κλάσης αυτής είναι να διατηρεί, να τροποποιεί, να προσθέτει ή να διαγράφει τις σχέσεις κλάσεων μεταξύ πακέτων (axisList), να υπολογίζει τον παράγοντα διάδοσης μέσω της συνάρτησης updatePropagation και να επιστρέφει αυτά τα δεδομένα στο χρήστη. Η κλάση αυτή λειτουργεί, ως επί το πλείστον, σαν δομή δεδομένων, καθώς η λειτουργικότητά της είναι περιορισμένη, αλλά είναι ιδιαίτερα χρήσιμη και σημαντική καθώς σε αυτή γίνεται η αναζήτηση για την εύρεση κλάσεων προς μετακίνηση.

Η κλάση ClassAxisObject έχει παρόμοια λειτουργία με την κλάση PackageAxis με τη διαφορά ότι η κλάση αυτή περιλαμβάνει και τις εσωτερικές εξαρτήσεις κάθε πακέτου ενώ η PackageAxis περιλαμβάνει τις εξαρτήσεις μεταξύ διαφορετικών πακέτων. Πιο συγκεκριμένα, η αποθήκευση εδώ γίνεται με βάση την κλάση και όχι το πακέτο. Για κάθε κλάση (className), αποθηκεύεται το πακέτο στο οποίο βρίσκεται, οι εξαρτήσεις μεταξύ κλάσεων που έχει με τις κλάσεις εντός του πακέτου της αλλά και οι εξαρτήσεις που έχει με τις κλάσεις άλλων πακέτων. Ειδικά, η λίστα axisList περιλαμβάνει τις εξαρτήσεις με τις κλάσεις εντός του πακέτου της ενώ η eaxisList περιλαμβάνει τις εξαρτήσεις με τις κλάσεις άλλων πακέτων. Αντίστοιχα, το χαρακτηριστικό probability είναι ο παράγοντας διάδοσης του πακέτου που ανήκει (εσωτερική πιθανότητα) ενώ το χαρακτηριστικό eprobability είναι ο παράγοντας διάδοσης προς όλα τα άλλα πακέτα (εξωτερική πιθανότητα). Η κλάση αυτή λειτουργεί ως κλάση αποθήκευσης δεδομένων με βασικές συναρτήσεις, τον κατασκευαστή (constructor), τις συναρτήσεις ανάθεσης (set) και τις συναρτήσεις εξαγωγής (get) κάθε χαρακτηριστικού, τη συνάρτηση εκτύπωσης και τη συνάρτηση επιστροφής «καθαρού» ονόματος. «Καθαρό όνομα» σημαίνει ότι επιστρέφει μόνο το όνομα της κλάσης και όχι όλο το μονοπάτι, δηλαδή τη διαδρομή πακέτων στην οποία ανήκει (πχ org.apache.maven.model.BuildAProject είναι ολόκληρο το όνομα και BuildAProject είναι το καθαρό όνομα).

Η βασική κλάση σε αυτή την ιεραρχική δομή, η οποία περιλαμβάνει όλες τις προηγούμενες δομές είναι η SystemAxisObject. Ένα αντικείμενο αυτής της κλάσης δίνει πρόσβαση σε όλες τις κλάσεις που αναλύθηκαν προηγουμένως. Πιο συγκεκριμένα, περιλαμβάνει χαρακτηριστικό τύπου ClassAxisObject σε μορφή Hashtable – πίνακα κατακερματισμού (δηλαδή ως δείκτης θεωρείται το όνομα της κλάσης και ως δεδομένο το αντικείμενο ClassAxisObject). Αντίστοιχα, περιλαμβάνει πίνακα κατακερματισμού όπου ο δείκτης είναι το όνομα του πακέτου και δεδομένο είναι το αντικείμενο τύπου PackageAxis. Χαρακτηριστικό του πίνακα κατακερματισμού είναι ότι μπορεί να εκτελέσει σε σταθερό χρόνο, δηλαδή με πολυπλοκότητα O(1), τις λειτουργίες εισαγωγής, αναζήτησης και διαγραφής στοιχείων. Η κλάση αυτή, επίσης λειτουργεί ως κλάση αποθήκευσης δεδομένων με βασικές συναρτήσεις, τον κατασκευαστή (constructor), τις συναρτήσεις ανάθεσης (set) και τις συναρτήσεις εξαγωγής (get) κάθε χαρακτηριστικού, τη συνάρτηση εκτύπωσης και τη συνάρτηση επιστροφής «καθαρού» ονόματος. Η πρόσβαση σε όλες τις προηγούμενες δομές αποκτάτε από αυτή την κλάση.

Μία ακόμα σημαντική κλάση για την αναδόμηση είναι η Refactoring. Με βάση τα αντικείμενα αυτής της κλάσης δημιουργούνται οι ομάδες κλάσεων προς αναδόμηση και υπολογίζονται οι νέες μετρικές. Η κλάση αυτή έχει ως χαρακτηριστικά το πακέτο προορισμού και προέλευσης, την κλάση προορισμού και προέλευσης, τις εξαρτήσεις που έχει η κλάση προέλευσης (από και προς αυτήν), την τιμή της ελαφριάς σύζευξης (Ce) πριν και μετά την αναδόμηση, το πλήθος εξαρτήσεων από την κλάση (ή κλάσεις) προέλευσης αλλά και το πλήθος προς την κλάση (ή κλάσεις) προέλευσης. Επίσης, ως χαρακτηριστικά διαθέτει την πιθανότητα ή παράγοντα διάδοσης (mcpm) πριν και μετά την αναδόμηση και τέλος την πιθανότητα κυματισμού μεταξύ των κλάσεων που υπάρχει η εξάρτηση (rem). Η κλάση αυτή δεν κάνει υπολογισμούς, απλά αποθηκεύει δεδομένα που υπολογίζονται στη βασική κλάση, η οποία εξηγείται στη συνέχεια.

Τέλος, η βασική κλάση Main, είναι αυτή που περιλαμβάνει την λειτουργικότητα της μεθοδολογίας. Το πραγματικό της όνομα είναι InternalProbabilityFrame ωστόσο για λόγους απλότητας εδώ αναφέρετε ως Main. Η κλάση Main έχει δύο χαρακτηριστικά, το ένα αφορά το γραφικό κομμάτι και το άλλο είναι τύπου SystemAxisObject. Η βασική συνάρτηση είναι η mainFuction όπου είναι υπεύθυνη για την εμφάνιση του γραφικού πλαισίου κατά την αρχή εκτέλεσης του λογισμικού, την συντακτική ανάλυση του λογισμικού εισόδου καλώντας τις κατάλληλες μεθόδους κλάσεων και τέλος τον υπολογισμό των αρχικών τιμών των μετρικών.

Στη συνέχεια, καλείται η συνάρτηση moveClassRefactoring η οποία προετοιμάζει κατάλληλα τις δομές δεδομένων και καλεί τη συνάρτηση calculateMetrics για την αναδόμηση μεγέθους ένα, δηλαδή τη μεταφορά μίας κλάσης. Μόλις ολοκληρωθεί η εκτέλεση της παραπάνω συνάρτησης επιστρέφει στην moveClassRefactoring, αποθηκεύονται τα δεδομένα σε αρχεία και καλείται η συνάρτηση moveClassSets η οποία είναι υπεύθυνη για τον υπολογισμό ομάδων κλάσεων. Μόλις ολοκληρωθεί η εκτέλεση των παραπάνω συναρτήσεων το πρόγραμμα αποθηκεύει όλα τα δεδομένα σε αρχεία και τερματίζει. Η συνάρτηση calculateMetrics, για κάθε σχέση μεταξύ πακέτων αναζητεί τις κλάσεις που δημιουργούν εξαρτήσεις και εξετάζει τη μεταφορά από το ένα πακέτο στο άλλο και αντίστροφα (αμφίδρομος έλεγχος). Για κάθε μία μετακίνηση υπολογίζονται οι μετρικές που αναφέρθηκαν νωρίτερα στη θεωρία αλλά και σε αυτό το κεφάλαιο, δηλαδή οι μετρικές στις κλάσεις Axis και PackageAxis. Επίσης, όλες οι μεταφορές αποθηκεύονται σε ξεχωριστό αντικείμενο στην κλάση Refactoring. Σημαντικό είναι να αναφερθεί ότι, υπολογίζεται η επίδραση κάθε μετακίνησης και από το πακέτο προέλευσης στο πακέτο προορισμού και από το πακέτο προορισμού στο πακέτο προέλευσης. Η συνάρτηση που είναι υπεύθυνη για τους αμφίδρομους ελέγχους είναι η reverseCheckOneClass. Μόλις ολοκληρωθούν οι υπολογισμοί για τις αναδομήσεις μεγέθους ένα, αποθηκεύονται όλα τα δεδομένα σε αρχεία και εκτελείται η συνάρτηση moveClassSets. Η συνάρτηση αυτή, η οποία είναι και η πιο σημαντική στη μεθοδολογία καθώς αποτελεί το καινοτόμο στοιχείο, είναι υπεύθυνη για τη δημιουργία ομάδων κλάσεων. Δέχεται ως παραμέτρους τη λίστα αντικειμένων τύπου Refactoring, το αρχικό PackageAxis και το μέγιστο μέγεθος ομάδων που επιθυμεί ο χρήστης. Ο λεπτομερής τρόπος λειτουργίας της θα περιγραφεί στο επόμενο υποκεφάλαιο με τίτλο Ψευδοκώδικας. Η λογική είναι η ίδια με την calculateMetrics, καλείται αναδρομικά, κάθε φόρα με είσοδο τη νέα λίστα αντικειμένων Refactoring. Μόλις ολοκληρωθούν όλοι οι υπολογισμοί, τα δεδομένα αποθηκεύονται σε αρχείο και το πρόγραμμα τερματίζει. Η συνάρτηση reverseCheckSetOfClasses έχει ως αρμοδιότητα τον αμφίδρομο έλεγχο για τη μετακίνηση ομάδων κλάσεων. Τέλος, η συνάρτηση writeFile είναι υπεύθυνη για την εγγραφή των δεδομένων σε αρχεία. Ένα αρχείο για κάθε μέγεθος αναδόμησης και ένα για κάθε αμφίδρομο έλεγχο. Η επιλογή αυτή έγινε για να είναι πιο κατανοητή η πληροφορία στο χρήστη αλλά και να μειωθεί ο χρόνος αναζήτησης από αυτόν.

5.2 Ψευδοκώδικας

"The most challenging part of programming is conceptualizing the problem, and many errors in programming are conceptual errors."
— Steve McConnell

Στο υποκεφάλαιο αυτό παρουσιάζεται ο ψευδοκώδικας της συνάρτησης moveClassSets που αναφέρθηκε στο προηγούμενο κεφάλαιο. Η συνάρτηση αυτή είναι υπεύθυνη για τον υπολογισμό ομάδων κλάσεων. Στην Εικόνα 17 απεικονίζεται ο αλγόριθμος.

Algorithm 1 Function that calculates the sets of classes to be moved

```
1:  procedure MOVECLASSSETS(packageAxis, Refactoring)
2:      for Each particle in packageAxis do
3:          while CoCD.isnotEmpty() and numOfClass <= numOfSets do
4:              c1 ← pop class from CoCD
5:              classesToBeMoved ← c1
6:              Remove dependencies from packageAxis
7:              if fromClass.packageAxis = c1 then
8:                  CoCD ← toClass.packageAxis
9:                  Add dependency from package B to package A
10:             if toClass.packageAxis = c1 then
11:                 CoCDFrom ← fromClass.packageAxis
12:                 Add dependency from package A to package B
13:             Update packageAxis A to B and B to A
14:             Reverse check
15:             Update packageAxis A to B and B to A
16:             Calculate metrics
17:             Refactoring ← push new metrics and refactoring data
            ▷ Recursion with parameters the new Refactoring and packageAxis
18:     goto 1.
```

Εικόνα 17 : Ψευδοκώδικας Υπολογισμού Ομάδων Κλάσεων

Όπως ειπώθηκε νωρίτερα, η κλάση PackageAxis περιέχει τις σχέσεις κλάσεων μεταξύ των πακέτων. Για λόγους απλότητας στον ψευδοκώδικα υπάρχει μόνο η λίστα με τα PackageAxis. Στη συνέχεια, θα αναλυθεί η υλοποίηση της μεθοδολογίας σε java και εκεί θα γίνει κατανοητή η χρήση των κλάσεων ClassAxisObject και PackageAxis. Ο αλγόριθμος δέχεται δύο λίστες ως παραμέτρους εισόδου, τη λίστα με όλες τις σχέσεις μεταξύ πακέτων και τις προτάσεις αναδόμησης του προηγούμενου μεγέθους, δηλαδή, έστω τώρα ο αλγόριθμος θα εκτελέσει το επίπεδο αναδόμησης n, θα δεχτεί ως παράμετρο εισόδου τις μετακινήσεις κλάσεων του μεγέθους n-1. Ακολούθως, για κάθε εξάρτηση κλάσεων μεταξύ των δύο πακέτων ο αλγόριθμος ελέγχει αν υπάρχουν κλάσεις που έχουν εξαρτήσεις με τις

κλάσεις από το πακέτο προέλευσης. Εφόσον υπάρχουν κλάσεις ή το μέγεθος ομάδας δεν έχει ξεπεράσει την τιμή numOfSets, η οποία δίνεται από το χρήστη, σε κάθε επανάληψη προστίθεται μία νέα κλάση στην ομάδα κλάσεων προς μετακίνηση. Η παραπάνω διαδικασία απεικονίζεται στα βήματα τρία, τέσσερα και πέντε. Στη συνέχεια, για την κλάση επιλογής προς μετακίνηση, διαγράφονται οι εξαρτήσεις προς το πακέτο προορισμού από το PackageAxis (γραμμή έξι) και γίνεται αναζήτηση των νέων εξαρτήσεων που δημιουργούνται λόγω της μετακίνησης. Οι εξαρτήσεις που έχουν ως κλάση προέλευσης την κλάση c1 προστίθενται στη σχέση πακέτων προορισμού προς προέλευσης (γραμμές επτά - εννιά) ενώ οι εξαρτήσεις που έχουν την κλάση c1 ως κλάση προορισμού προστίθενται στη σχέση πακέτων προέλευσης προς προορισμού (γραμμές δέκα – δώδεκα). Ακολουθεί η ανανέωση των σχέσεων μεταξύ πακέτων και ο υπολογισμός των νέων παραγόντων διάδοσης. Η παραπάνω διαδικασία αφορά τις σχέσεις: πακέτο προέλευσης προς πακέτο προορισμού και πακέτο προορισμού προς πακέτο προέλευσης. Ο αντίστροφος έλεγχος ή αλλιώς αμφίδρομος που αναφέρεται στη γραμμή δεκατέσσερα σημαίνει να μετακινηθεί η κλάση προορισμού στο πακέτο προέλευσης. Ο λόγος που ελέγχεται η περίπτωση αυτή αναλύθηκε με παράδειγμα στο Κεφάλαιο 4, Εικόνα 8. Στη γραμμή δεκαπέντε γίνεται ο υπολογισμός των παραγόντων διάδοσης και για τον αμφίδρομο έλεγχο. Ο υπολογισμός μετρικών στη γραμμή δεκαέξι αφορά την ελαφριά σύζευξη ή ένταση (Ce) μεταξύ των πακέτων. Τέλος, όταν όλες οι μετρικές της μετακίνησης έχουν υπολογιστεί, τα δεδομένα αποθηκεύονται σε αντικείμενο τύπου Refactoring το οποίο προστίθεται σε λίστα αντικειμένων. Η διαδικασία αυτή επαναλαμβάνεται για κάθε εξάρτηση κλάσεων κάθε σχέσης μεταξύ πακέτων. Μόλις ελεγχθούν όλες, οι προτάσεις μετακίνησης κλάσεων για το συγκεκριμένο επίπεδο έχουν ολοκληρωθεί, τα δεδομένα αποθηκεύονται σε αρχείο τύπου .csv και ο αλγόριθμος συνεχίζει να εκτελείται καλώντας αναδρομικά τη συνάρτηση moveClassSets για τους υπολογισμούς του επόμενου μεγέθους αναδόμησης. Η αναδρομική κλήση συνεχίζεται μέχρις ότου είτε να μην υπάρχουν κλάσεις για να δημιουργηθούν άλλες ομάδες ή το μέγεθος αναδόμησης να φτάσει την μέγιστη τιμή που έδωσε ο χρήστης.

5.3 Υλοποίηση σε Java

"A programmer had a problem. He decided to use Java.
He now has a ProblemFactory."

Στο υποκεφάλαιο αυτό θα παρουσιαστούν βασικά σημεία του κώδικα τα οποία υλοποιούν ότι έχει αναφερθεί ως τώρα στο Κεφάλαιο 4 και στο Κεφάλαιο 5. Στην Εικόνα 18 απεικονίζεται η υλοποίηση της συνάρτησης moveClassRefactoring, της κλάσης Main. Το κύριο στοιχείο υλοποίησης που φαίνεται είναι το πώς ξεδιπλώνεται η ιεραρχική δομή δεδομένων. Από το αντικείμενο system παρέχεται πρόσβαση στην κλάση PackageAxisObject, η κλάση αυτή δεν αναλύθηκε πιο πάνω καθώς είναι μία κλάση που διατηρεί αντικείμενα κλάσης τύπου PackageAxis. Στη συνέχεια, με τη χρήση Iterator (δομή επανάληψης), διατρέχεται η λίστα με τα PackageAxisObject με σκοπό την απόκτηση πρόσβασης στη λίστα PackageAxis. Τα αντικείμενα αυτά αποθηκεύονται σε μία νέα λίστα με PackageAxis ώστε να μην χρειάζεται να διατρέχετε όλη η δομή κάθε φορά που αυτή περνάτε ως παράμετρος στις συναρτήσεις calculateMetrics και moveClassSets.

Είναι σημαντικό να αναφερθεί ότι, η Java είναι γλώσσα προγραμματισμού που χειρίζεται τα αντικείμενα της με pass-by-reference και αυτό προσθέτει αρκετούς φραγμούς στο χειρισμό τους. Σε πολλά σημεία δημιουργήθηκαν νέα αντικείμενα ή πραγματοποιήθηκε επαναφορά των αλλαγών που έγιναν νωρίτερα προκειμένου να μην χαθεί η αρχική πληροφορία. Παρόλο που η Java παρέχει συναρτήσεις κλωνοποίησης αντικειμένων για τέτοιες περιστάσεις, οι συναρτήσεις αυτές δεν είναι «σταθερές» με αποτέλεσμα στα περισσότερα έργα λογισμικού να μην έχουν τον επιθυμητό τρόπο λειτουργίας.

Συνεχίζοντας, όπως φαίνεται και στον κώδικα, μόλις η συνάρτηση τερματίσει, τα δεδομένα αποθηκεύονται σε αρχεία και καλείται η συνάρτηση υπολογισμού ομάδων κλάσεων προς αναδόμηση. Η πρώτη παράμετρος με την τιμή 5 δηλώνει το μέγιστο μέγεθος αναδόμησης.

```
// Iterate System Axis Object
Set<PackageAxisObject> packageAxisObjectSet = system.getPackageSet();
Iterator<PackageAxisObject> packageAxisIterator = packageAxisObjectSet.iterator();

ArrayList <PackageAxis> packageAxisObjects =  new ArrayList <PackageAxis>();
while(packageAxisIterator.hasNext())
{

    PackageAxisObject packageAxisObject = packageAxisIterator.next();
    Hashtable<String, PackageAxis> classAxisObjectSet = packageAxisObject.getPackageAxisSet();

    for(String key : classAxisObjectSet.keySet())
    {
        PackageAxis packageAxis = classAxisObjectSet.get(key);
        packageAxisObjects.add(packageAxis);
    }
}

//%%%%%%%%%%%%%%%%%%%%%%%% CALCULATE METRICS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Map<String, List<Refactoring>> map = new HashMap<String, List<Refactoring>>();
map = calculateMetrics(packageAxisObjects, system);

refactoringObjects.addAll((List<Refactoring>) map.get("mainList"));
refactoringObjectsReverse.addAll((List<Refactoring>) map.get("reverseList"));

//%%%%%%%%%%%%%%%%%%%%%%%% WRITE IN FILE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
writeFile("refactoring_level1.csv",refactoringObjects);
writeFile("reverseRefactoringLevel1.csv",refactoringObjectsReverse);

//%%%%%%%%%%%%%%%%%%%%%%%% MOVE CLASSES SETS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
moveClassSets(5, refactoringObjects, packageAxisObjects, system, refactoringObjects);
```

Εικόνα 18 : Κλάση Αρχικοποίησης – moveClassRefactoring

Στην Εικόνα 19 απεικονίζονται κάποια τμήμα υλοποίησης της συνάρτησης moveClassSets.

```
while (refObject.get(j).getFromClass().equals(fromClass))
{
    System.out.println("<---> From class: " + refObject .get(j).getFromClass()
            + " <---> To class: " + refObject.get(j).getToClass());

    if (refObject.get(j).getclassesCoCD() != null)
    {
        System.out.println("<---> Refactoring opportunities <--->");
        ArrayList<String> dependentClasses = new ArrayList<String>();
        ArrayList<String> dependentClassesToFrom = new ArrayList<String>();

        Iterator y = refObject.get(j).getclassesCoCD();
        while (y.hasNext())
        {
            // Step 1 : Pop class from CoCD
            String c1 = (String) y.next();
            System.out.println("<---> C1 class is : " + c1);
            dependentClasses.addAll(refObject.get(j).getListCoCDClasses());
            dependentClasses.remove(c1);
            dependentClassesToFrom.addAll(refObject.get(j).getListCoCDClassesFrom());
            dependentClassesToFrom.remove(c1);
            ArrayList<PackageAxis> tempPackageAxis = new ArrayList<PackageAxis> (packAxisObject);

            // Step 2 : Add c1 as class to be removed
            classesToBeMoved.add(c1);
            if (f == 1)
                classesToBeMoved.add(fromClass);

            // Step 3 : Remove dependency from Package Axis (packageFrom -> PackageTo)
            ArrayList<Axis> axisToBeDeleted = new ArrayList<Axis>();
            for (int in = 0; in < tempPackageAxis.size(); in++)
            {
                if (tempPackageAxis.get(in).getFromPackage().equals(packageFrom) &&
                        tempPackageAxis.get(in).getToPackage().equals(packageTo))
```

Εικόνα 19 : Υλοποίηση συνάρτησης moveClassSets – Μέρος 1

Πιο συγκεκριμένα, διατρέχεται η λίστα αντικειμένων Refactoring η οποία περιέχει τις προτάσεις μετακίνησης για μία κλάση. Για κάθε μία πρόταση, αν υπάρχουν κλάσεις με εξαρτήσεις προς ή από την κλάση προέλευσης, επιλέγονται μία μία από τη λίστα μέχρις ότου είτε η λίστα να αδειάσει είτε να συμπληρωθεί το μέγεθος ομάδας αυτής της εκτέλεσης. Έστω η πρώτη κλάση από τις εξαρτήσεις που επιλέγεται η c1, εξάγεται από τη λίστα και αποθηκεύεται στον πίνακα με τις κλάσεις προς μετακίνηση. Ως επόμενο βήμα, είναι η αφαίρεση των εξαρτήσεων από τις κλάσεις που θα μετακινηθούν προς το πακέτο προορισμού. Δεδομένου ότι οι κλάσεις θα μετακινηθούν στο πακέτο προορισμού, αυτές οι εξαρτήσεις από το πακέτο προέλευσης προς το πακέτο προορισμού επιλύονται. Μία λεπτομέρεια που πρέπει να ειπωθεί προτού συνεχιστεί η ανάλυση του αλγορίθμου. Στην περίπτωση μετακίνησης από τριών κλάσεων και πάνω, ως κλάσεις προέλευσης, οι κλάσεις αποθηκεύονται σε ένα χαρακτηριστικό κλάσης (classFrom) και διαχωρίζονται με «#». Στο σημείο υλοποίησης του βήματος δύο, όπως φαίνεται στην Εικόνα 19, ελέγχεται αν το τρέχων μέγεθος αναδόμησης είναι 2 (η αρίθμηση ξεκινάει από 0, οπότε το 1 θεωρείται το μέγεθος 2). Αν είναι δύο (1 στον κώδικα), αποθηκεύεται με αυτό τον τρόπο η αρχική κλάση της εξάρτησης (όχι η c1, η c1 μεταφέρετε επειδή έχει εξάρτηση με την fromClass, ως αρχική κλάση θεωρείται η fromClass). Αν το μέγεθος αναδόμησης είναι μεγαλύτερο από δύο, η αποθήκευση των κλάσεων προέλευσης υλοποιείται όπως φαίνεται στην Εικόνα 20.

```
if (fromClass.contains("#"))
{
    classesToBeMoved.clear();
    String array[] = fromClass.split("#");
    for (String x : array)
    {
        System.out.print(x);
        classesToBeMoved.add(x);
    }
}
```

Εικόνα 20 : Υλοποίηση Συνάρτησης moveClassSets - Μέρος 2

Στην Εικόνα 21 απεικονίζεται το επόμενο βήμα του αλγορίθμου, ο υπολογισμός των νέων εξαρτήσεων που δημιουργούνται λόγο της μετακίνησης. Γίνεται χρήση του αντικειμένου ClassAxisObject και ελέγχεται η λίστα των εσωτερικών εξαρτήσεων εντός του πακέτου. Επίσης, στην Εικόνα 21 φαίνεται ο έλεγχος όπου η κλάση προέλευσης είναι η c1. Σε επόμενες γραμμές ελέγχονται και οι υπόλοιπες κλάσεις που υπάρχουν στη λίστα με τις κλάσεις προς μετακίνηση, classesToBeMoved. Η διαφορά είναι στη συνθήκη ελέγχου (if). Ο έλεγχος που φαίνεται στην εικόνα σχετίζεται με τις κλάσεις τις οποίες καλεί η c1, υπάρχει αντίστοιχος έλεγχος για τις κλάσεις οι οποίες καλούν την c1. Με την ολοκλήρωση του

βήματος αυτού, ο αλγόριθμος είναι σε θέση να ανανεώσει τις μετρικές και να συνεχίσει με την επόμενη ομάδα προς μετακίνηση.

```
ListIterator it = system.getClassListIterator();
int flagToBreak = 0;
while(it.hasNext())
{
    ClassAxisObject ca = (ClassAxisObject)it.next();
    ListIterator axisListIt = ca.getAxisListIterator(); //internal

    if (ca.getPackageName().equals(packageFrom))
    {
        flagToBreak = 1;
    }
    if (flagToBreak == 1 && !ca.getPackageName().equals(packageFrom))
    {
        break;
    }

    if (ca.getPackageName().equals(packageFrom) && ca.getName().equals(c1))
    {
        while(axisListIt.hasNext())
        {
            Axis axisObject = (Axis)axisListIt.next();

            String innerClassTo = axisObject.getToClass();
```

Εικόνα 21 : Υλοποίηση Συνάρτησης moveClassSets - Μέρος 3

Επισημαίνεται ότι, οι εξαρτήσεις που επιλύθηκαν είναι το πλήθος αυτών που υπολογίστηκαν από τη γραμμή 6 του ψευδοκώδικα ή την Εικόνα 19 του κώδικα ενώ οι νέες εξαρτήσεις είναι το άθροισμα των εξαρτήσεων που δεν επιλύθηκαν και αυτών που δημιουργήθηκαν λόγω της μετακίνησης. Η διαδικασία αυτή φαίνεται στον ψευδοκώδικα στις γραμμές 7 – 12 ενώ στον κώδικα στην Εικόνα 21. Στην Εικόνα 22 φαίνεται η ανανέωση των τιμών των μετρικών και οι αποθήκευσή τους στο αντικείμενο τύπου Refactoring.

```
Refactoring newRefObject = new Refactoring();
newRefObject.setFromclass(fromClass + "#" + c1);
newRefObject.setToclass(refObject.get(j).getToClass());
newRefObject.setPackageNameFrom(packageFrom);
newRefObject.setPackageNameTo(packageTo);
for (int k = 0; k < dependentClasses.size(); k++)
    newRefObject.setClassesCoCD(dependentClasses.get(k));
for (int k = 0; k < dependentClassesToFrom.size(); k++)
    newRefObject.setClassesCoCDFrom(dependentClassesToFrom.get(k));
newRefObject.setIntensityOld(refObject.get(j).getIntensityOld());
newRefObject.setIntensityNew(refObject.get(j).getIntensityOld() - resolvedCounter + dependentClassesToFrom.size());
newRefObject.setCoCD(dependentClasses.size());
newRefObject.setCoCDFrom(dependentClassesToFrom.size());
newRefObject.setCoRD(resolvedCounter);
newRefObject.setMcpmOld(refObject.get(j).getMpcmOld());
newRefObject.setRem(refObject.get(j).getRem());


for (int in = 0; in < tempPackageAxis.size(); in++)
{
    if (tempPackageAxis.get(in).getFromPackage().equals(packageFrom) &&
            tempPackageAxis.get(in).getToPackage().equals(packageTo))
    {
        tempPackageAxis.get(in).updatePropagation();
        newRefObject.setMcpmNew(tempPackageAxis.get(in).getProbability());
        break;
    }
}
```

Εικόνα 22 : Υλοποίηση Συνάρτησης moveClassSets - Μέρος 4

Όπως απεικονίζεται, οι κλάσεις προέλευσης αποθηκεύονται όλες μαζί σε μία μεταβλητή με το στοιχείο «#» ως στοιχείο διαχωρισμού. Στη συνέχεια, οι κλάσεις που δημιουργούν εξαρτήσεις προς και από τις κλάσεις που μεταφέρονται αποθηκεύονται σε πίνακες, ο νέος υπολογισμός της έντασης ή ελαφριάς σύζευξης πραγματοποιείται αφαιρώντας από την αρχική τιμή το πλήθος εξαρτήσεων που επιλύθηκαν και προσθέτοντας το άθροισμα αυτών που δεν επιλύθηκαν και αυτών που δημιουργήθηκαν. Τέλος, ο νέος παράγοντας διάδοσης, υπολογίζεται εκ νέου με τα νέα δεδομένα και αποθηκεύεται μέσω της συνάρτησης setMcpmNew στο αντικείμενο newRefObject το οποίο είναι τύπου Refactoring. Όλα τα αντικείμενα τύπου Refactoring αποθηκεύονται σε μία λίστα αντικειμένων με όνομα newRefactoring. Στην Εικόνα 23 φαίνεται η αναδρομική κλήση της συνάρτησης moveClassSets και η λίστα αυτή περνάτε ως η δεύτερη παράμετρος. Εφόσον δεν έχει υπολογιστεί το μέγεθος αναδόμησης που δόθηκε από το χρήστη, η συνάρτηση ξανά καλεί τον εαυτό της με τα νέα στοιχεία αναδόμησης ως παραμέτρους.

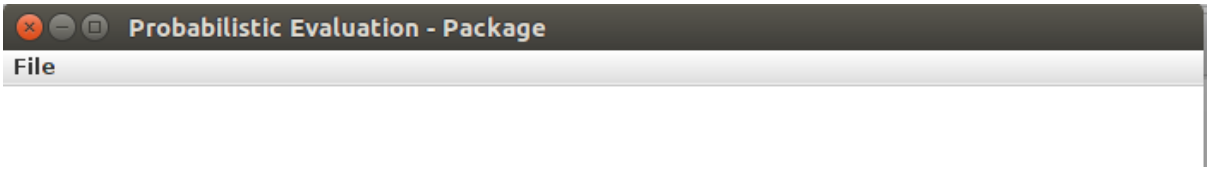
```
f++;
if ( f < numOfSets)
{
    System.out.println("end");
    moveClassSets(numOfSets, newRefactoring, packAxisObject, system, initiallyRefObject);
    System.out.println("end again");
}
```

Εικόνα 23 : Υλοποίηση Συνάρτησης moveClassSets - Μέρος 5 Αναδρομή

5.4 Εκτέλεση Λογισμικού

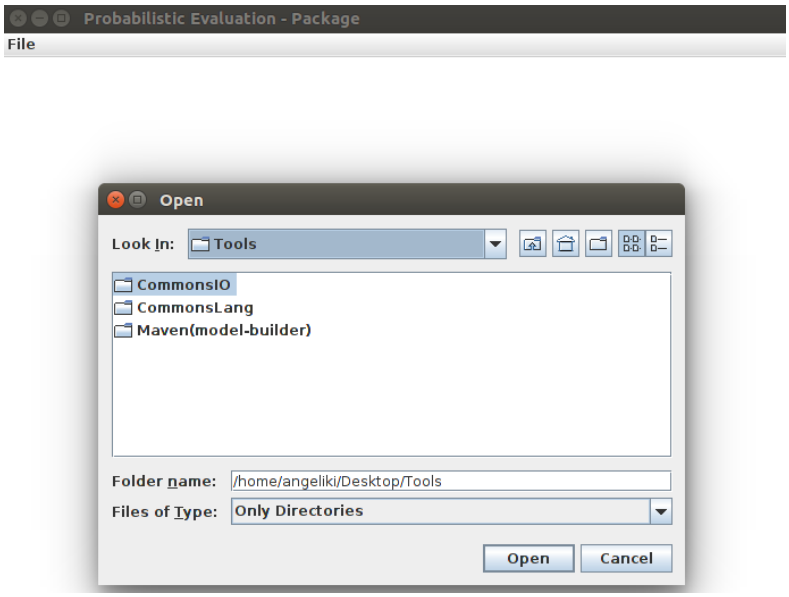
“Anyone who has never made a mistake has never tried anything new.”
— Albert Einstein

Το λογισμικό εκτελείται από το IDE Eclipse, Oxygen. Επιλέγοντας εκτέλεση, ανοίγει το πλαίσιο που φαίνεται στην Εικόνα 24. Στη συνέχεια, στο πλαίσιο που εμφανίζεται, πάνω αριστερά υπάρχει η επιλογή File. Το λογισμικό είναι σε θέση να διαβάσει αρχεία .java, .class και .xml. Τα έργα λογισμικού που χρησιμοποιήθηκαν ως είσοδος ήταν τύπου .class οπότε, η πρώτη επιλογή, άνοιγμα φακέλου από το File, δίνει τη δυνατότητα στο χρήστη να επιλέξει ως είσοδο για ανάλυση, το λογισμικό που επιθυμεί όπως φαίνεται στην Εικόνα 25.



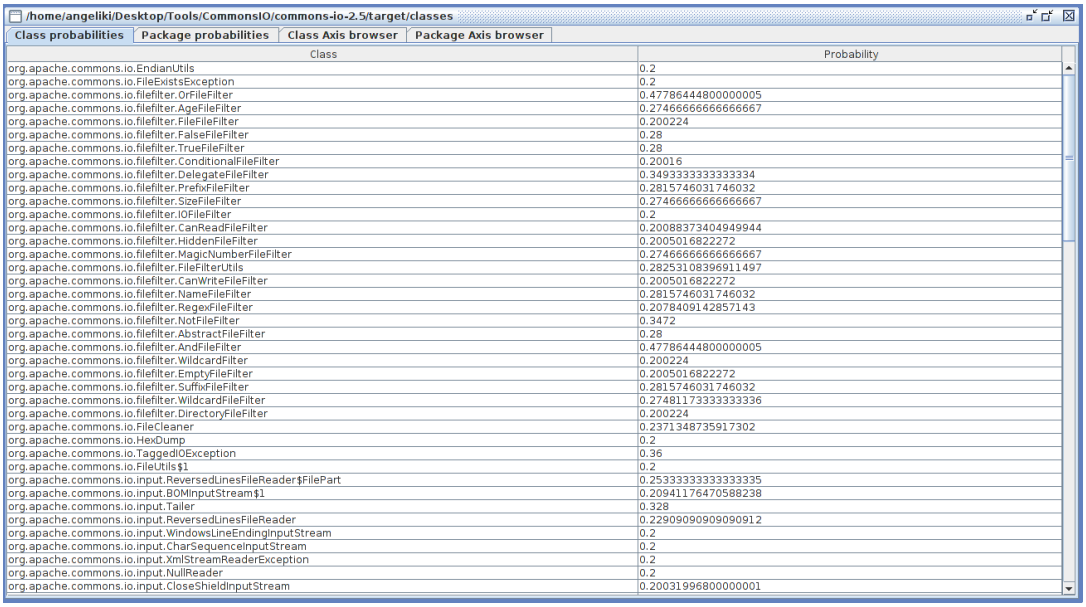
Εικόνα 24 : Εκτέλεση Λογισμικού - Πλαίσιο Γραφικών

Στα πλαίσια αυτής της εκτέλεσης, επιλέγεται ένα από τα λογισμικά που χρησιμοποιήθηκαν για την εξαγωγή των αποτελεσμάτων, κάτι που εξηγείται στα παρακάτω κεφάλαια. Το λογισμικό εισόδου είναι το Commons IO, του οργανισμού Apache, αποτελεί ελεύθερο λογισμικό και στο Κεφάλαιο 6 γίνεται λεπτομερής ανάλυσή του.



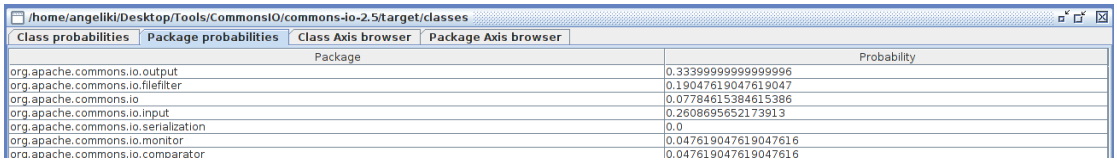
Εικόνα 25 : Εκτέλεση Λογισμικού - Επιλογή Εισόδου

Μόλις επιλεχτεί το λογισμικό εισόδου, το πρόγραμμα εκτελεί συντακτική ανάλυση και αποθηκεύει τις πληροφορίες στις δομές δεδομένων. Στη συνέχεια, εκτελείται η μεθοδολογία αναδόμησης μεγέθους πέντε. Για τον υπολογισμό ομάδων κλάσεων μεγέθους πέντε έχουν υπολογιστεί και όλες οι μικρότερες ομάδες μεγέθους τέσσερα, τρία, δύο και ένα. Κατά την ολοκλήρωση της εκτέλεσης εμφανίζεται το πλαίσιο που φαίνεται στις Εικόνες 26, 27, 28 και 29. Κάθε μία από αυτές εμφανίζει ένα διαφορετικό πίνακα (tab). Υπάρχουν τέσσερις καρτέλες, η πρώτη από αριστερά εμφανίζει την εσωτερική πιθανότητα κάθε κλάσης και φαίνεται στην Εικόνα 26.



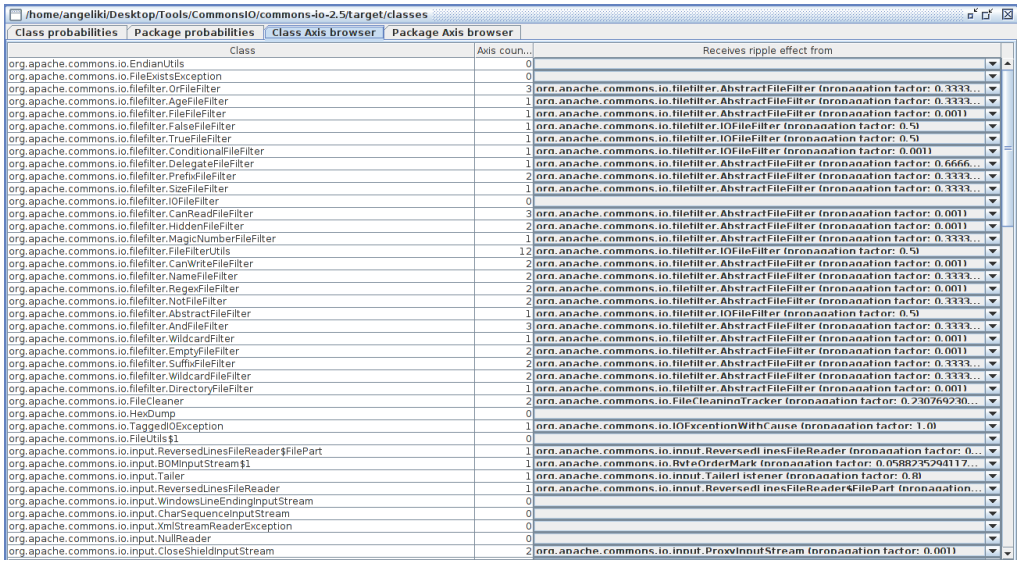
Εικόνα 26 : Εκτέλεση Λογισμικού - Αποτελέσματα Μέρος 1

Η δεύτερη περιέχει την εσωτερική πιθανότητα κάθε πακέτου όπως απεικονίζεται στην Εικόνα 27.

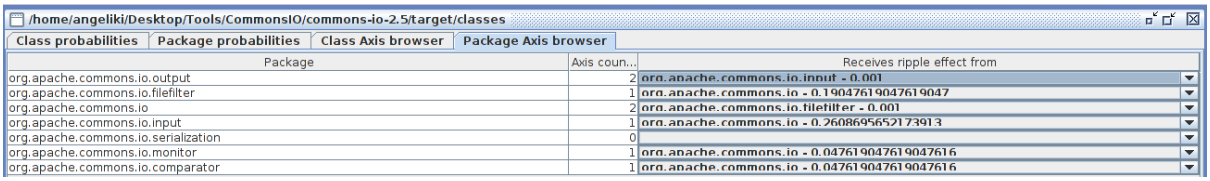


Εικόνα 27 : Εκτέλεση Λογισμικού - Αποτελέσματα Μέρος 2

Η τρίτη, η οποία φαίνεται στην Εικόνα 28, απεικονίζει τις εξαρτήσεις κάθε κλάσης. Πιο συγκεκριμένα, η αριστερή στήλη περιέχει τις κλάσεις προέλευσης, η μεσαία στήλη το πλήθος εξαρτήσεων και η δεξιά στήλη τις κλάσεις προορισμού και την πιθανότητα κυματισμού της εξάρτησης (REM). Τέλος, στην Εικόνα 29 απεικονίζονται οι εξαρτήσεις μεταξύ των πακέτων και ο παράγοντας διάδοσης αυτών (MCPM). Όλες οι σχέσεις που απεικονίζονται στα τέσσερα πλαίσια εξήχθησαν από τα bytecodes και σύμφωνα με αυτά υπολογίστηκαν οι πιθανότητες (rem, mcpm). Στην επόμενη υποενότητα παρουσιάζονται τα αποτελέσματα που παράχθηκαν από τη μεθοδολογία.



Εικόνα 28 : Εκτέλεση Λογισμικού - Αποτελέσματα Μέρος 3



Εικόνα 29 : Εκτέλεση Λογισμικού - Αποτελέσματα Μέρος 4

5.5 Έξοδος – Αποτελέσματα Λογισμικού

"Success represents the 1% of your work which results from the 99% that is called failure."
— Soichiro Honda

Στην υποενότητα αυτή παρουσιάζονται τα αποτελέσματα του λογισμικού που αφορούν τις προτάσεις αναδόμησης. Στις Εικόνες 30, 31 και 32 φαίνονται τα αποτελέσματα αναδόμησης που παράγονται για μέγεθος αναδόμησης ένα. Στην Εικόνα 30, η κλάση FileUtils του πακέτου org.apache.commons.io έχει εξάρτηση με την κλάση IOFileFilter του πακέτου org.apache.commons.io.filefilter όπως φαίνεται στην πρώτη γραμμή. Προτείνετε από το πρόγραμμα να μετακινηθεί η κλάση FileUtils στο πακέτο org.apache.commons.io.filefilter.

From class	To class	Package Name From	Package Name To
org.apache.commons.io.FileUtils	org.apache.commons.io.filefilter.IOFileFilter	org.apache.commons.io	org.apache.commons.io.filefilter
org.apache.commons.io.FileUtils	org.apache.commons.io.filefilter.SuffixFileFilter	org.apache.commons.io	org.apache.commons.io.filefilter
org.apache.commons.io.IOUtils	org.apache.commons.io.output.StringBuilderWriter	org.apache.commons.io	org.apache.commons.io.output
org.apache.commons.io.IOUtils	org.apache.commons.io.output.ByteArrayOutputStream	org.apache.commons.io	org.apache.commons.io.output
org.apache.commons.io.output.TaggedOutputStream	org.apache.commons.io.TaggedIOException	org.apache.commons.io.output	org.apache.commons.io
org.apache.commons.io.filefilter.FileFilterUtils	org.apache.commons.io.IOCase	org.apache.commons.io.filefilter	org.apache.commons.io
org.apache.commons.io.filefilter.RegexFileFilter	org.apache.commons.io.IOCase	org.apache.commons.io.filefilter	org.apache.commons.io
org.apache.commons.io.filefilter.WildcardFileFilter	org.apache.commons.io.IOCase	org.apache.commons.io.filefilter	org.apache.commons.io
org.apache.commons.io.DirectoryWalker	org.apache.commons.io.filefilter.IOFileFilter	org.apache.commons.io	org.apache.commons.io.filefilter
org.apache.commons.io.FileUtils	org.apache.commons.io.output.NullOutputStream	org.apache.commons.io	org.apache.commons.io.output
org.apache.commons.io.input.XmlStreamReader	org.apache.commons.io.ByteOrderMark	org.apache.commons.io.input	org.apache.commons.io
org.apache.commons.io.input.TaggedInputStream	org.apache.commons.io.TaggedIOException	org.apache.commons.io.input	org.apache.commons.io
org.apache.commons.io.comparator.PathFileComparator	org.apache.commons.io.IOCase	org.apache.commons.io.compara	org.apache.commons.io
org.apache.commons.io.comparator.ExtensionFileComparator	org.apache.commons.io.IOCase	org.apache.commons.io.compara	org.apache.commons.io
org.apache.commons.io.comparator.NameFileComparator	org.apache.commons.io.IOCase	org.apache.commons.io.compara	org.apache.commons.io
org.apache.commons.io.output.ByteArrayOutputStream	org.apache.commons.io.input.ClosedInputStream	org.apache.commons.io.output	org.apache.commons.io.input
org.apache.commons.io.filefilter.SuffixFileFilter	org.apache.commons.io.IOCase	org.apache.commons.io.filefilter	org.apache.commons.io
org.apache.commons.io.filefilter.PrefixFileFilter	org.apache.commons.io.IOCase	org.apache.commons.io.filefilter	org.apache.commons.io
org.apache.commons.io.filefilter.NameFileFilter	org.apache.commons.io.IOCase	org.apache.commons.io.filefilter	org.apache.commons.io
org.apache.commons.io.input.BOMInputStream	org.apache.commons.io.ByteOrderMark	org.apache.commons.io.input	org.apache.commons.io

Εικόνα 30 : Εκτέλεση Λογισμικού - Αποτελέσματα Αναδόμησης #1

Η επίδραση αυτής της μετακίνησης φαίνεται στην Εικόνα 31, στην πρώτη γραμμή. Εκεί απεικονίζεται η ένταση ή αλλιώς ελαφριά σύζευξη (Ce). Για το συγκεκριμένο παράδειγμα το όφελος είναι δύο. Η αναδόμηση αυτή κατάφερε και μείωσε την ελαφριά σύζευξη κατά δύο. Από τρία που ήταν την μετέτρεψε σε ένα. Η εξήγηση αυτού φαίνεται στην Εικόνα 32.

Intensity Old	Intensity New	Intensity Benefit
3	1	2
3	1	2
3	1	2
3	1	2
1	0	1
6	5	1
6	5	1
6	5	1
3	2	1
3	2	1
3	2	1
3	2	1
3	2	1
3	2	1
3	2	1
1	1	0
6	6	0
6	6	0
6	6	0
3	3	0

Εικόνα 31 : Εκτέλεση Λογισμικού - Αποτελέσματα Αναδόμησης - Σύζευξη

Στην πρώτη γραμμή της Εικόνας 32, στην τρίτη και τέταρτη στήλη, απεικονίζονται το πλήθος των εξαρτήσεων που επιλύθηκαν και το άθροισμα του πλήθος των νέων εξαρτήσεων και αυτών που δεν επιλύθηκαν. Στην τρίτη στήλη η τιμή είναι 2 ενώ στην τέταρτη 0. Αυτό σημαίνει ότι επιλύθηκαν 2 εξαρτήσεις και δεν δημιουργήθηκε κάποια καινούργια.

Created Dependencies Classes from class A	Created Dependencies Classes to class A	Count of Removed D	Count of Created Depende	Count of Created Dependencies to class A	MCMP Old
org.apache.commons.io.FileExistsException #		2	0	0	0.001000000000000
org.apache.commons.io.FileExistsException #		2	0	0	0.001000000000000
org.apache.commons.io.LineIterator #		2	0	0	0.076923076923076
org.apache.commons.io.LineIterator #		2	0	0	0.076923076923076
org.apache.commons.io.output.ProxyOutputStream #		1	0	0	0.333333333333333
org.apache.commons.io.filefilter.IOFileFilter #	org.apache.commons.io.filefilter.DelegateFileFI	1	0	0	0.190476190476190
org.apache.commons.io.filefilter.AbstractFileFilter #		1	0	0	0.190476190476190
org.apache.commons.io.filefilter.AbstractFileFilter #		1	0	0	0.190476190476190
		1	0	0	0.001000000000000
org.apache.commons.io.FileExistsException #		1	0	0	0.076923076923076
org.apache.commons.io.input.BOMInputStream #	org.apache.commons.io.input.XmlStreamR	1	0	0	0.260869565217391
org.apache.commons.io.input.ProxyInputStream #		1	0	0	0.260869565217391
org.apache.commons.io.comparator.AbstractFileComparator #	org.apache.commons.io.comp	1	0	0	0.047619047619047
org.apache.commons.io.comparator.AbstractFileComparator #	org.apache.commons.io.comp	1	0	0	0.047619047619047
org.apache.commons.io.comparator.AbstractFileComparator #	org.apache.commons.io.comp	1	0	0	0.047619047619047
	org.apache.commons.io.output.DeferredFil	1	1	1	0.001000000000000
org.apache.commons.io.filefilter.AbstractFileFilter	org.apache.commons.io.filefilter.FileFilterU	1	1	1	0.190476190476190
org.apache.commons.io.filefilter.AbstractFileFilter	org.apache.commons.io.filefilter.FileFilterU	1	1	1	0.190476190476190
org.apache.commons.io.filefilter.AbstractFileFilter	org.apache.commons.io.filefilter.FileFilterU	1	1	1	0.190476190476190
org.apache.commons.io.input.ProxyInputStream	org.apache.commons.io.input.XmlStreamR	1	1	1	0.260869565217391

Εικόνα 32 : Εκτέλεση Λογισμικού - Αποτελέσματα Αναδόμησης - Εξαρτήσεις

Η μία εξάρτηση που δεν επιλύθηκε προέρχεται από άλλη κλάση από το πακέτο προέλευσης προς το πακέτο προορισμού. Στην δεύτερη γραμμή της Εικόνας 30 απεικονίζεται αυτή η σχέση.

Ενδεικτικά, στην Εικόνα 33, φαίνεται ένα παράδειγμα αρχείου εξόδου μεγέθους αναδόμησης τρία. Η λογική και η δομή είναι ακριβώς η ίδια με το αρχείο αναδόμησης μεγέθους ένα. Στην Εικόνα 33 γίνεται εμφανές πως αποθηκεύονται οι κλάσεις της ομάδας στο ίδιο αντικείμενο χρησιμοποιώντας το στοιχείο διαχωρισμού «#»

From class	To class
org.apache.commons.io.output.ByteArrayOutputStream#org.apache.commons.io.output.DeferredFileOutputStream#org.apache.commons.io.output.ThresholdI	org.apache.commons.io.input.ClosedInputStream
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.AgeFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.SuffixFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.AndFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.OrFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.MagicNumberFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.NameFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.PrefixFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.DelegateFileFilter#org.apache.commons.io.filefilter.SizeFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AgeFileFilter#org.apache.commons.io.filefilter.SuffixFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AgeFileFilter#org.apache.commons.io.filefilter.AndFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AgeFileFilter#org.apache.commons.io.filefilter.OrFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AgeFileFilter#org.apache.commons.io.filefilter.MagicNumberFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AgeFileFilter#org.apache.commons.io.filefilter.NameFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AgeFileFilter#org.apache.commons.io.filefilter.PrefixFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AgeFileFilter#org.apache.commons.io.filefilter.SizeFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.SuffixFileFilter#org.apache.commons.io.filefilter.AndFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.SuffixFileFilter#org.apache.commons.io.filefilter.OrFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.SuffixFileFilter#org.apache.commons.io.filefilter.MagicNumberFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.SuffixFileFilter#org.apache.commons.io.filefilter.NameFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.SuffixFileFilter#org.apache.commons.io.filefilter.PrefixFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.SuffixFileFilter#org.apache.commons.io.filefilter.SizeFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AndFileFilter#org.apache.commons.io.filefilter.OrFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AndFileFilter#org.apache.commons.io.filefilter.MagicNumberFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AndFileFilter#org.apache.commons.io.filefilter.NameFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AndFileFilter#org.apache.commons.io.filefilter.PrefixFileFilter	org.apache.commons.io.IOCase
org.apache.commons.io.filefilter.FileFilterUtils#org.apache.commons.io.filefilter.AndFileFilter#org.apache.commons.io.filefilter.SizeFileFilter	org.apache.commons.io.IOCase

Εικόνα 33 : Εκτέλεση Λογισμικού - Αποτελέσματα Αναδόμησης #3

Προτεραιότητα σε αυτή την εργασία δόθηκε στη μεθοδολογία και τη λειτουργικότητα και όχι στο γραφικό περιβάλλον. Ωστόσο η εργασία αυτή δεν ολοκληρώνεται σε αυτή εδώ τη διπλωματική αλλά θα συνεχιστεί σε πλαίσια ερευνητικής δουλειάς. Εκεί θα δημιουργηθεί ένα πιο εύχρηστο και όμορφο γραφικό περιβάλλον προς το χρήστη, όπου οι καλύτερες προτάσεις μετακίνησης θα αποτυπώνονται στο γραφικό περιβάλλον με αποτέλεσμα ο χρήστης να μην χρειάζεται να ανοίγει και να επεξεργάζεται αρχεία .csv. Επίσης, ο αλγόριθμος και οι δομές δεδομένων θα βελτιστοποιηθούν και ενδέχεται να δοθεί ως Plugin στην πλατφόρμα ελεύθερου λογισμικού του Eclipse, εφόσον καλύπτονται όλα τα κριτήρια καλής σχεδίασης.

6. Επικύρωση Μεθοδολογίας

"To raise new questions, new possibilities, to regard old problems from a new angle, require creative imagination and marks real advance in science."
— Albert Einstein

Σε αυτή την ενότητα παρουσιάζεται ο σχεδιασμός της μελέτης περιπτώσεων, σύμφωνα με τις οδηγίες του Runeson. Συγκεκριμένα, παρουσιάζεται:

- ο σκοπός της μελέτης περιπτώσεων και τα ερευνητικά ερωτήματα που προκύπτουν
- η περιγραφή των περιπτώσεων και των μονάδων ανάλυσης
- η συλλογή δεδομένων
- η διαδικασία της ανάλυσης δεδομένων

6.1 Στόχοι και Ερευνητικές Ερωτήσεις

"If we knew what it was we were doing, it would not be called research, would it?"
— Albert Einstein

Σκοπός αυτής της μελέτης περιπτώσεων είναι η διερεύνηση των αποτελεσμάτων της προτεινόμενης μεθόδου από τις ακόλουθες προοπτικές:

- (στόχος #1) αξιολόγηση του οφέλους από την εφαρμογή των προτεινόμενων λύσεων αναδόμησης σε όρους αρχιτεκτονικού τεχνικού χρέους (Architectural Technical Debt - ATD),
- (στόχος #2) έρευνα εάν το πλήθος των κλάσεων που πρόκειται να μετακινηθούν επηρεάζει το όφελος που επιτυγχάνεται με την εφαρμογή των λαμβανομένων αναδομήσεων,
- (στόχος #3) παροχή αρχικής έρευνας σχετικά με το αν η εννοιολογική ομοιότητα των εντοπισθέντων αντικειμένων (των κλάσεων προς μετακίνηση) σχετίζεται με την επείγουσα ανάγκη αναδόμησης, όπως προβλέπεται από τη μέθοδο.

Μία σημαντική έννοια που είναι άμεσα συνδεδεμένη με την ποιότητα λογισμικού, ωστόσο δεν αποτελεί τον κύριο άξονα σε αυτή τη διπλωματική εργασία είναι ο όρος αρχιτεκτονικό τεχνικό χρέος ή απλώς τεχνικό χρέος. Το τεχνικό χρέος είναι μια έννοια στην ανάπτυξη λογισμικού που αντικατοπτρίζει το κόστος της επιπρόσθετης επεξεργασίας που

προκαλείται από την επιλογή μιας εύκολης αλλά όχι ποιοτικότερης λύσης, αντί της χρήσης μιας καλύτερης μεθοδολογίας που θα χρειαζόταν περισσότερο χρόνο κατά την ανάπτυξη.

Με βάση τον προαναφερθέντα στόχο, έχουν τεθεί τρία ερευνητικά ερωτήματα που θα χρησιμοποιηθούν για το σχεδιασμό της μελέτης και θα καθοδηγήσουν την αναφορά των αποτελεσμάτων που προέκυψαν:

- [**RQ1**] *Ποια είναι η αποτελεσματικότητα της προτεινόμενης μεθόδου όσον αφορά τη μείωση του αρχιτεκτονικού τεχνικού χρέους;*

  Αυτή η ερευνητική ερώτηση χρησιμεύει ως επικύρωση της προτεινόμενης μεθοδολογίας αναδόμησης. Ως δείκτες επιτυχίας για την αξιολόγηση της έκβασης της μεθόδου, χρησιμοποιήθηκαν τρεις μετρικές, η τμηματικότητα, η συνοχή και η σύζευξη σε επίπεδο πακέτου. Το αντικείμενο της συγκεκριμένης εργασίας δεν είναι το τεχνικό χρέος ωστόσο τα αποτελέσματα που προκύπτουν είναι άμεσα συσχετιζόμενα με αυτό.
- [**RQ2**] *Το πλήθος των κλάσεων που πρέπει να μετακινηθούν όπως προτείνεται από τη μέθοδο επηρεάζει την αποτελεσματικότητα της αναδόμησης;*

  Όπως αναφέρθηκε στο Κεφάλαιο 1.2, ένα από τα βασικά πλεονεκτήματα της προτεινόμενης μεθόδου σε σύγκριση με την υπάρχουσα βιβλιογραφία είναι το γεγονός ότι αυτή η μέθοδος επιτρέπει την μετακίνηση πολλαπλών κλάσεων από ένα πακέτο προέλευσης σε ένα πακέτο προορισμού. Στο RQ2, διερευνάται ο βαθμός στον οποίο το θεωρητικό όφελος κεφαλαιοποιείται όσον αφορά το ATD, δηλαδή τον καθορισμένο δείκτη επιτυχίας.
- [**RQ3**] *Η εννοιολογική ομοιότητα της μετακινούμενης κλάσης με το πακέτο προορισμού επηρεάζει την αποτελεσματικότητα της αναδόμησης;*

  Η προτεινόμενη μέθοδος βασίζεται αποκλειστικά στη δομή (structure) του εξεταζόμενου λογισμικού. Ωστόσο, στο Κεφάλαιο 3 γίνεται σαφές ότι η παραβίαση της αρχής μοναδικής αρμοδιότητας και των προτεινόμενων αναδομήσεων, δεν σχετίζονται μόνο με το μέγεθος της μονάδας λογισμικού, αλλά και με την εννοιολογική της συνοχή. Υπό αυτή την έννοια, σε αυτή την ερευνητική ερώτηση διερευνάται αν ο δείκτης αναδόμησης, όπως υπολογίζεται με την προτεινόμενη μέθοδο, είναι σε θέση να κάνει διακρίσεις μεταξύ εννοιολογικά συσχετισμένων και εννοιολογικά ασυσχέτιστων προτάσεων αναδόμησης.

6.2 Επιλογή Περιπτώσεων και Μονάδες Ανάλυσης

"Why do we never have time to do it right, but always have time to do it over?"

Η μελέτη περίπτωσης αυτής της διπλωματικής εργασίας αποτελεί μία ενσωματωμένη μελέτη περίπτωσης, στην οποία το πλαίσιο είναι ο τομέας Λογισμικών Ανοιχτού Κώδικα (Open Source Software - OSS), τα θέματα είναι τα έργα Λογισμικού Ανοιχτού Κώδικα (Open Source Software - OSS) και οι μονάδες ανάλυσης είναι οι ευκαιρίες αναδόμησης που προτείνονται από τη μέθοδο που αναλύθηκε στο Κεφάλαιο 4. Προκειμένου τα δεδομένα να ανακτηθούν μόνο από προγράμματα υψηλής ποιότητας που εξελίσσονται σε μια περίοδο, επιλέχθηκε η εστίαση μόνο σε έργα από το οικοσύστημα λογισμικού Apache. Τα έργα του οργανισμού Apache είναι μεταξύ αυτών που κατά μήκος της εξέλιξης μειώνουν την ποσότητα κανονικοποιημένου τεχνικού χρέους που έχει συσσωρευτεί. Ως εκ τούτου, αποτελούν υποψήφια θέματα καλής ποιότητας, τα οποία είναι ταυτόχρονα εδραιωμένα στην κοινότητα της μηχανικής λογισμικού ή τεχνολογίας λογισμικού. Τα έργα που χρησιμοποιήθηκαν ως θέματα παρουσιάζονται στον Πίνακα 16. Το πρώτο project ονομάζεται Commons IO, αποτελεί μια βιβλιοθήκη του οργανισμού Apache και περιέχει βοηθητικές κλάσεις, υλοποιήσεις ροών, φίλτρα αρχείων, συγκριτές αρχείων, μετασχηματισμούς endian κλάσεων και πολλά άλλα. Οι τυποποιημένες βιβλιοθήκες της Java αποτυγχάνουν να παρέχουν αρκετές μεθόδους για το χειρισμό των κλάσεων. Το δεύτερο project, Commons Lang παρέχει αυτές τις επιπλέον μεθόδους. Το Lang παρέχει πλήθος βοηθητικών κλάσεων για το java.lang API, κυρίως μεθόδους χειρισμού αλφαριθμητικών, βασικές αριθμητικές μεθόδους, καλύτερη διαχείριση αντικειμένων, ταυτόχρονη εκτέλεση, δημιουργία και σειριοποίηση και ιδιότητες του συστήματος. Επιπλέον, περιέχει βασικές βελτιώσεις στο java.util.Date και μια σειρά βοηθητικών προγραμμάτων που προορίζονται να βοηθήσουν με μεθόδους οικοδόμησης, όπως hashCode, toString και equals. Το τρίτο και τελευταίο project, Maven, αποτελεί ένα εργαλείο διαχείρισης έργων λογισμικού. Με βάση την ιδέα ενός έργου μοντέλου αντικειμένου (Project Object Model - POM), το Maven μπορεί να διαχειριστεί την κατασκευή, την υποβολή εκθέσεων (release, κυκλοφορία νέων εκδόσεων λογισμικού) και την τεκμηρίωση ενός έργου από μια κεντρική πληροφορία. Δεδομένου ότι το τρίτο έργο λογισμικού ήταν ιδιαίτερα μεγάλο ως προς το πλήθος πακέτων και κλάσεων επιλέχθηκε μία υποενότητα αυτού, το model-builder.

Πίνακας 16 : Αντικείμενα (Λογισμικά) Μελέτης Περιπτώσεων

Όνομα Project	Περιγραφή	Πλήθος Πακέτων	Πλήθος Κλάσεων
Commons IO	Επιλέχθηκε όλο το project.	10	140
Commons Lang	Επιλέχθηκε όλο το project.	16	298
Maven	Επιλέχθηκε ένα τμήμα του project.	20	136

Στα πλαίσια αυτής της διπλωματικής εργασίας επιλέχθηκαν μεσαία προς μικρά project, και από ένα μεγάλο project επιλέχθηκε ένα τμήμα του. Ο λόγος που έγινε αυτή η επιλογή είναι η πολυπλοκότητα και ο χρόνος εκτέλεσης του αλγορίθμου. Η ανάλυση μεγάλων λογισμικών απαιτεί πολύ χρόνο και ένα καλό υπολογιστικό σύστημα για να εκτελεστεί η μεθοδολογία όπως, ένας διακομιστής. Συμπληρωματικά, η συγκεκριμένη εργασία αποτελεί ένα τμήμα μιας μεγαλύτερης και πιο εκτεταμένης ερευνητικής δουλειάς οπότε οποιαδήποτε συμπεράσματα ή παραλείψεις θα οδηγήσουν σε καλύτερα μελλοντικά αποτελέσματα και συμπεράσματα.

6.3 Συλλογή Δεδομένων

“Data! Data! Data! I can’t make bricks without clay!”
— Sir Arthur Conan Doyle (Sherlock Holmes)

Η συλλογή δεδομένων αυτής της περιπτωσιολογικής μελέτης είναι μια διαδικασία τριών φάσεων. Η πρώτη φάση περιλαμβάνει μετρήσεις στο στάδιο προ-αναδόμησης του συστήματος. Η δεύτερη φάση περιλαμβάνει τις μετρήσεις που επιτυγχάνονται με την εφαρμογή της μεθόδου, ενώ στην τρίτη φάση λαμβάνουν χώρα οι μετρήσεις μετά την αναδόμηση. Για κάθε μονάδα ανάλυσης, δηλαδή για κάθε ζεύγος πακέτων που εμπλέκονται στη διαδικασία αναδόμησης (refactoring), καταγράφονται οι ακόλουθες μεταβλητές :

Στάδιο προ-επεξεργασίας:

- [V1] Όνομα: Όνομα έργου
- [V2] Από: Πακέτο από το οποίο αφαιρείται η κλάση
- [V3] Προς: Πακέτο στο οποίο μετακινείται η κλάση

- [V4] TCIP_Pre: Συνολική ένταση σύζευξης μεταξύ των πακέτων πριν από την αναδόμηση
- [V5] CaPC_Pre: Συνοχή μεταξύ των κλάσεων των πακέτων πριν από την αναδόμηση
- [V6] Modularity_Pre: Τμηματικότητα του πακέτου πριν από την αναδόμηση

Στάδιο αναδόμησης:

- [V7] Όνομα Κλάσης (ή κλάσεων): Όνομα της κλάσης που πρόκειται να μετακινηθεί
- [V8] Μέγεθος Αναδόμησης: Το πλήθος των κλάσεων που μετακινούνται
- [V9] Ένταση Αναδόμησης: Η τιμή της μετρικής Ce, το πλήθος εξαρτήσεων μετά την αναδόμηση
- [V10] Εννοιολογική ομοιότητα: Μια δυαδική μεταβλητή (1 = ναι, 0 = όχι) που υποδηλώνει αν η μετακινούμενη κλάση σχετίζεται εννοιολογικά με την κλάση ή το πακέτο προορισμού.

Στάδιο μετά-επεξεργασίας:

- [V11] TCIP_Post: Συνολική ένταση σύζευξης μεταξύ των πακέτων μετά την αναδόμηση.
- [V12] CaPC_Post: Συνοχή μεταξύ των κλάσεων των πακέτων μετά από την αναδόμηση.
- [V13] Modularity_Post: Τμηματικότητα του πακέτου μετά από την αναδόμηση.

6.4 Ανάλυση Δεδομένων

"It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."
— Sir Arthur Conan Doyle (Sherlock Holmes)

Προκειμένου να δοθεί απάντηση στο RQ1 πραγματοποιήθηκε στατιστική ανάλυση των τιμών των μεταβλητών [V4] - [V6] και [V11] - [V13] συγκρίνοντας τις διαφορές των μέσων τιμών με τη βοήθεια ελέγχου υποθέσεων. Στο επόμενο κεφάλαιο, Κεφάλαιο 7, παρέχεται απεικόνιση μέσω γραφημάτων θερμότητας (heatMap), κυκλικών διαγραμμάτων (pie chart) και διαγράμματος γραμμών. Όσον αφορά το RQ2, ακολουθεί μια παρόμοια διαδικασία λαμβάνοντας υπόψη τη μεταβλητή [V8]. Τέλος, για την απάντηση του RQ3, έγινε συσχέτιση δύο φάσεων μεταξύ των μεταβλητών [V9] και [V10], καθώς η μεταβλητή [V10] είναι δυαδική.

Κατά τη διεξαγωγής μιας ποσοτικής έρευνας, οι ερευνητές προσπαθούν να απαντήσουν σε μια ερευνητική ερώτηση ή υπόθεση που έχουν θέσει. Μια μέθοδος αξιολόγησης αυτού του ερευνητικού ερωτήματος είναι μέσω μιας διαδικασίας που ονομάζεται έλεγχος υποθέσεων, η οποία μερικές φορές επίσης αναφέρεται ως έλεγχος σημαντικότητας. Κατά τη μέτρηση αντικειμένων σε μια μελέτη που σκοπό έχει την κατανόηση των διαφορών (ή οποιοδήποτε άλλο είδος επίδρασης), μπορεί απλά να γίνει μια σύνοψη των δεδομένα που έχουν συλλεχθεί. Η δομή του ελέγχου υποθέσεων ακολουθεί κάποια βήματα τα οποία είναι :

1. Καθορισμός της ερευνητική υπόθεσης για τη μελέτη.
2. Εξήγηση των βημάτων που πρόκειται να ακολουθηθούν, το αντικείμενο μελέτης και τον καθορισμό μεταβλητών που πρέπει να μελετηθούν.
3. Καθορισμός της μηδενικής και της εναλλακτικής υπόθεσης (ή περισσότερες από μία υποθέσεις).
4. Ορισμός επιπέδου σημαντικότητας (ορίου).
5. Πρόβλεψη κάποιων αποτελεσμάτων (one - or two-tailed prediction.).
6. Προσδιορισμός εάν η κατανομή που μελετάτε είναι κανονική (αυτό έχει συνέπειες για τους τύπους στατιστικών δοκιμών που μπορούν να εκτελεστούν στα δεδομένα).
7. Επιλογή ενός κατάλληλου στατιστικού ελέγχου βάσει των μεταβλητών που έχουν οριστεί και εάν η κατανομή είναι κανονική ή όχι.
8. Εκτέλεση των στατιστικών ελέγχων στα δεδομένα και ερμηνεία της εξόδου.
9. Απόρριψη ή αποτυχία απόρριψης της μηδενικής υπόθεσης.

Υπάρχουν τρεις βασικοί λόγοι για τους οποίους θα πρέπει να είμαστε σαφείς σχετικά με τον τρόπο διεξαγωγής της μέτρησης του αντικειμένου μελέτης. Πρώτον, πρέπει να υπάρχει σαφήνεια, ώστε οι άνθρωποι που διαβάζουν την εργασία να μην αμφιβάλλουν για το τι μελετάτε. Με αυτόν τον τρόπο διευκολύνεται η επανάληψη της μελέτης στο μέλλον για να διαπιστωθεί εάν άλλοι ερευνητές έχουν τα ίδια (ή παρόμοια) αποτελέσματα, κάτι που ονομάζεται εσωτερική εγκυρότητα. Δεύτερον, ένα από τα κριτήρια βάσει των οποίων αξιολογείται η ποσοτική έρευνα, είναι ο τρόπος με τον οποίο καθορίζεται αυτό που μετράτε (στην περίπτωση αυτή, την απόδοση) και πώς επιλέγεται να γίνει η μέτρηση. Τρίτον, θα καθορίσει ποιος στατιστικός έλεγχος πρέπει να χρησιμοποιηθεί επειδή η επιλογή του στατιστικού ελέγχου βασίζεται σε μεγάλο βαθμό στο πώς μετρήθηκαν οι μεταβλητές. Αξίζει

να σημειωθεί ότι, αυτές οι επιλογές είναι μερικές φορές προσωπικές επιλογές (δηλαδή είναι υποκειμενικές) ενώ άλλες φορές καθοδηγούνται από κάποιες άλλες εξωτερικές πληροφορίες.

Το επόμενο βήμα είναι ο ορισμός των μεταβλητών που χρησιμοποιούνται στη μελέτη. Μια μεταβλητή δεν είναι μόνο κάτι που μετράτε, αλλά και κάτι που μπορεί ο ερευνητής να χειριστεί και να ελέγξει. Μια ανεξάρτητη μεταβλητή, που μερικές φορές ονομάζεται πειραματική ή προγνωστική μεταβλητή, είναι μια μεταβλητή που με βάση αυτή σε ένα πείραμα, γίνεται η παρατήρηση του αποτελέσματος σε μια εξαρτημένη μεταβλητή, που μερικές φορές ονομάζεται μεταβλητή έκβασης. Στην πειραματική έρευνα, ο στόχος είναι ο χειρισμός μιας ανεξάρτητης μεταβλητής (ή μεταβλητών) και στη συνέχεια η εξέταση του αποτελέσματος αυτού του χειρισμού (της αλλαγής) σε μια εξαρτημένη μεταβλητή (ή μεταβλητές). Δεδομένου ότι είναι δυνατόν να χειριστεί κανείς την ανεξάρτητη μεταβλητή, η πειραματική έρευνα έχει το πλεονέκτημα ότι επιτρέπει σε έναν ερευνητή να εντοπίσει μια αιτία και το αποτέλεσμα μεταξύ μεταβλητών.

Προκειμένου να διεξαχθεί ένας έλεγχος υποθέσεων, πρέπει να εκφραστεί η ερευνητική υπόθεση ως μηδενική και εναλλακτική υπόθεση. Η μηδενική υπόθεση και η εναλλακτική υπόθεση είναι δηλώσεις σχετικά με τις διαφορές ή τα αποτελέσματα που συμβαίνουν στον πληθυσμό. Γίνεται χρήση του δείγματος ώστε να ελεγχθεί ποια δήλωση (δηλαδή η μηδενική υπόθεση ή η εναλλακτική υπόθεση) είναι πιθανότερη (αν και τεχνικά, δοκιμάζονται τα στοιχεία ενάντια στην μηδενική υπόθεση). Η μηδενική υπόθεση αποτελεί ουσιαστικά, κάτι που εκφράζει επιχειρήματα της αντίθετης άποψης από αυτής του ερευνητή, δηλαδή επιχειρήματα της άλλης πλευράς που συνήθως είναι και αρνητική. Η εναλλακτική υπόθεση δηλώνει το αντίθετο και είναι συνήθως η υπόθεση που προσπαθεί να αποδειχθεί. Ανάλογα με τον τρόπο που θεωρείται ο επιθυμητός για τη σύνοψη των αποδόσεων των μετρικών ορίζεται, ο καθορισμός μιας πιο συγκεκριμένης μηδενικής και εναλλακτικής υπόθεσης. Αυτό μπορεί να γίνει με πολλές επιλογές που περιλαμβάνουν τη σύγκριση των μέσων, την τυπική απόκλιση και άλλες.

Το επίπεδο στατιστικής σημασίας εκφράζεται συχνά ως η αποκαλούμενη τιμή ρ. Ανάλογα με τον στατιστικό έλεγχο που έχει επιλεχθεί, υπολογίζεται η πιθανότητα (δηλαδή η τιμή p) να παρατηρηθούν τα δείγματα των αποτελεσμάτων δεδομένου ότι, η μηδενική υπόθεση είναι αληθής. Ένας άλλος τρόπος διατύπωσης αυτού είναι να εξεταστεί η πιθανότητα να προκύψει μια διαφορά σε μία μέση τιμή (ή άλλο στατιστικό στοιχείο) με βάση την υπόθεση ότι δεν υπάρχει πραγματικά καμία διαφορά. Το εύρος τιμών της μεταβλητής p,

μιας που είναι πιθανότητα είναι [0,1]. Έτσι, μπορεί να προκύψει μια τιμή p, όπως 0.03 (δηλαδή, p = 0.03). Αυτό σημαίνει ότι υπάρχει πιθανότητα 3% να βρεθεί μια διαφορά τόσο μεγάλη όσο (ή μεγαλύτερη από) εκείνη της μελέτης δεδομένου ότι η μηδενική υπόθεση είναι αληθής. Ωστόσο, αυτό που είναι επιθυμητό να προκύψει ως συμπέρασμα είναι εάν αυτό είναι "στατιστικά σημαντικό". Το πιο συνηθισμένο όριο στατιστικά σημαντικότητας είναι το 5%. Αυτό το όριο χρησιμοποιήθηκε και σε αυτή τη διπλωματική εργασία. Ενώ υπάρχει σχετικά μικρή αιτιολόγηση για το λόγο που χρησιμοποιείται ένα όριο σημαντικότητας 0.05 αντί για 0.01 ή 0.10, χρησιμοποιείται ευρέως στην ακαδημαϊκή έρευνα. Ωστόσο, εάν είναι επιθυμητό τα αποτελέσματά να είναι ιδιαίτερα ακριβές, πρέπει να οριστεί ένα πιο αυστηρό επίπεδο, το 0.01 (πιθανότητα 1% ή λιγότερη, πιθανότητα 1 στις 100 ή λιγότερες). Κατά την εξέταση ή απόρριψη της μηδενικής υπόθεσης και αποδοχή της εναλλακτικής υπόθεσης, πρέπει να εξετάζεται η κατεύθυνση της δήλωσης εναλλακτικής υπόθεσης. Η εναλλακτική υπόθεση μας λέει δύο πράγματα. Πρώτον, ποιες προβλέψεις έγιναν σχετικά με την επίδραση της ανεξάρτητης μεταβλητής στην εξαρτημένη μεταβλητή. Δεύτερον, ποια ήταν η προβλεπόμενη κατεύθυνση αυτού του αποτελέσματος. Εάν μια εναλλακτική υπόθεση έχει μια κατεύθυνση, η υπόθεση είναι μονόπλευρη. Δηλαδή, προβλέπει την κατεύθυνση του αποτελέσματος. Εάν η εναλλακτική υπόθεση έχει δηλώσει ότι η επίδραση αναμενόταν να είναι αρνητική, αυτή είναι και η μόνη υπόθεση. Εναλλακτικά, μια πρόβλεψη δύο ουρών (two-tailed prediction) σημαίνει ότι δεν επιλέγεται η κατεύθυνση που ακολουθεί το αποτέλεσμα του πειράματος. Αντίθετα, σημαίνει απλώς ότι το αποτέλεσμα θα μπορούσε να είναι αρνητικό ή θετικό. Με άλλα λόγια, η λέξη "θετική", υποδηλώνει την κατεύθυνση του αποτελέσματος. Ωστόσο, αυτή είναι μόνο η γνώμη του ερευνητή και σίγουρα δεν σημαίνει ότι θα έχει το αποτέλεσμα που αναμένει. Σε γενικές γραμμές, η πρόβλεψη μιας ουράς κλονίζεται, καθώς αντανακλά συνήθως την ελπίδα ενός ερευνητή παρά την βεβαιότητα ότι θα συμβεί. Σημαντικές εξαιρέσεις σε αυτόν τον κανόνα είναι όταν υπάρχει μόνο ένας πιθανός τρόπος με τον οποίο θα μπορούσε να συμβεί μια αλλαγή. Αυτό μπορεί να συμβεί, για παράδειγμα, όταν μετρηθεί η βιολογική δραστηριότητα / παρουσία. Δηλαδή, μια πρωτεΐνη μπορεί να είναι "αδρανής" και το ερέθισμα που χρησιμοποιείτε μπορεί μόνο να "ξυπνήσει" (δηλαδή, δεν μπορεί να μειώσει τη δραστικότητα της "αδρανούς" πρωτεΐνης). Επιπλέον, για ορισμένους στατιστικούς ελέγχους, δεν είναι δυνατές οι δοκιμές με ένα πλήθος. Τελικά, στο ερώτημα αν απορρίπτεται ή αποτυγχάνει η απόρριψη της μηδενικής υπόθεσης ισχύει το εξής: Εάν η στατιστική ανάλυση δείξει ότι το επίπεδο σημαντικότητας είναι κάτω από την τιμή αποκοπής που έχει οριστεί

(π.χ. 0.05), απορρίπτεται η μηδενική υπόθεση και γίνεται αποδεκτή η εναλλακτική υπόθεση. Εναλλακτικά, αν το επίπεδο σημαντικότητας είναι πάνω από την τιμή αποκοπής, αποτυγχάνει η απόρριψη της μηδενικής υπόθεσης και δεν είναι δυνατή η αποδοχή της εναλλακτικής υπόθεσης. Θα πρέπει να σημειωθεί ότι, δεν γίνεται να γίνει αποδεκτή η μηδενική υπόθεση, παρά μόνο αν βρεθούν ενδείξεις εναντίον της.

Ο στατιστικός έλεγχος που υλοποιήθηκε ονομάζεται Paired Samples T-test. Ο έλεγχος Paired Samples T-test συγκρίνει δύο μέσες τιμές που προέρχονται από το ίδιο στοιχείο, αντικείμενο ή δείγμα. Οι δύο μέσες τιμές τυπικά αντιπροσωπεύουν δύο διαφορετικές χρονικές στιγμές (π.χ., προ-δοκιμή και μετά-δοκιμή με παρέμβαση μεταξύ των δύο χρονικών σημείων). Ο σκοπός του ελέγχου είναι να προσδιοριστεί εάν υπάρχουν στατιστικές αποδείξεις ότι η μέση διαφορά μεταξύ των δύο μονάδων παρατηρήσεως σε ένα συγκεκριμένο αποτέλεσμα είναι σημαντικά διαφορετική από το μηδέν.

7. Αποτελέσματα

"Failure is simply the opportunity to begin again, this time more intelligently."
— Henry Ford

Σε αυτή την ενότητα παρουσιάζονται τα αποτελέσματα της μελέτης περιπτώσεων. Ακολουθεί παρουσίαση των αποτελεσμάτων και μια σύντομη επεξήγηση. Στο επόμενο κεφάλαιο, Κεφάλαιο 8, θα ακολουθήσει σχολιασμός και συμπεράσματα των αποτελεσμάτων που παρουσιάζονται στο κεφάλαιο αυτό.

7.1 Αποτελεσματικότητα στη Μείωση του Αρχιτεκτονικού Τεχνικού Χρέους (RQ1)

Σε αυτή την ενότητα παρουσιάζονται τα αποτελέσματα που αφορούν την απάντηση στο RQ1. Στον Πίνακα 17 παρουσιάζονται τα περιγραφικά στατιστικά για τα χαρακτηριστικά ποιότητας όλου του δείγματος. Πιο συγκεκριμένα, σε κάθε σειρά παρουσιάζεται ένα από αυτά τα χαρακτηριστικά, τα οποία είναι η τμηματικότητα, η συνοχή και η σύζευξη. Οι στήλες χωρίζονται σε δύο κατηγορίες, στις στατιστικές τιμές πριν την αναδόμηση και στις στατιστικές τιμές μετά την αναδόμηση. Οι τιμές αυτές είναι, η ελάχιστη τιμή που έλαβε το κάθε χαρακτηριστικό από το συνολικό δείγμα, η μέγιστη τιμή που έλαβε το κάθε χαρακτηριστικό από το συνολικό δείγμα, η μέση τιμή και η τυπική απόκλιση. Ως μέση τιμή ορίζεται το άθροισμα των δειγμάτων διά του πλήθους των δειγμάτων ενώ η τυπική απόκλιση ορίζεται ως ένα μέτρο που χρησιμοποιείται για να υπολογιστεί το ποσό της μεταβολής ή της διασποράς ενός συνόλου τιμών δεδομένων. Σκοπός της εργασίας αυτής ήταν η βελτιστοποίηση των μεταβλητών αυτών μετά την αναδόμηση. Πιο συγκεκριμένα, τα χαρακτηριστικά τμηματικότητα και συνοχή είναι επιθυμητό να αυξηθούν μετά την αναδόμηση, ενώ το χαρακτηριστικό σύζευξη να μειωθεί. Αυτό ισχύει για όλες τις ερευνητικές ερωτήσεις αυτής της εργασίας. Σε κάθε γραμμή του πίνακα με γαλάζια σκίαση κελιού αποτυπώνετε εάν η βέλτιστη τιμή αναγνωρίζεται πριν ή μετά την αναδόμηση. Συνοπτικά, κοιτώντας τη στήλη *μέση τιμή* παρατηρείται ότι, στο χαρακτηριστικό τμηματικότητα δεν υπήρξε βελτίωση καθώς από 28.408 που ήταν πριν την αναδόμηση μειώθηκε σε 18.517. Για το χαρακτηριστικό συνοχή, η μέση τιμή από 0.638 έγινε 0.370,

οπότε ούτε εδώ παρατηρείται βελτίωση. Τέλος, η μέση τιμή της σύζευξης από 0.497 έγινε 0.233, συνεπώς η σύζευξη βελτιώθηκε λίγο παραπάνω από 50%.

Πίνακας 17 : Περιγραφικά Στατιστικά Δείγματος

Χαρακτηριστικό Ποιότητας	Πριν				Μετά			
	Ελάχιστο	Μέγιστο	Μέση τιμή	Τυπική Απόκλιση	Ελάχιστο	Μέγιστο	Μέση τιμή	Τυπική Απόκλιση
Τμηματικότητα	0.21884	500	28.408	85.995245	0	400	18.517	59.811423
Συνοχή	0.18182	1	0.6382	0.2342272	0	1	0.3705	0.2556441
Σύζευξη	0.001	3.86652	0.4976	1.0974411	0	3.63982	0.2333	0.5670822

Στον Πίνακα 18 παρουσιάζονται τα χαρακτηριστικά ποιότητας σε σχέση με τον έλεγχο υποθέσεων. Ειδικά, η κάθε γραμμή περιλαμβάνει ένα από τα χαρακτηριστικά ποιότητας που ερευνήθηκαν. Η κάθε στήλη περιλαμβάνει τη διαφορά μεταξύ των αρχικών και των τελικών τιμών, δηλαδή των τιμών πριν την αναδόμηση και μετά την αναδόμηση. Το συνολικό δείγμα αποτελούνταν από 137 στοιχεία. Η στήλη *όφελος* προσδιορίζει πόσα από αυτά βελτιώθηκαν, η στήλη *ουδέτερο* πόσα από αυτά παραμένουν ίδια, η στήλη *απώλεια* πόσα δείγματα χειροτέρεψαν μετά την αναδόμηση και τέλος η *στατιστική σημαντικότητα*. Η στατιστική σημαντικότητα είναι θεμελιώδης για τον έλεγχο στατιστικών υποθέσεων. Όπως εξηγήθηκε στο Κεφάλαιο 6.4, σε κάθε πείραμα ή παρατήρηση που περιλαμβάνει τη σύνταξη ενός δείγματος από έναν πληθυσμό, υπάρχει πάντα η πιθανότητα ότι ένα παρατηρούμενο αποτέλεσμα θα συνέβαινε λόγω σφάλμα δειγματοληψίας. Αλλά αν η p-τιμή είναι μικρότερη από το επίπεδο σημαντικότητας ($p < 0.05$ στη συγκεκριμένη περίπτωση), τότε ο ερευνητής μπορεί να συμπεράνει ότι η παρατηρούμενη επίδραση αντανακλά στα πραγματικά χαρακτηριστικά του πληθυσμού και όχι μόνο στο δειγματοληπτικό σφάλμα. Συνοπτικά από τον Πίνακα 18 φαίνεται ότι η τμηματικότητα έχει στατιστική σημαντικότητα 0.146 με όφελος 57, η συνοχή έχει στατιστική σημαντικότητα 0 με όφελος 11 ενώ η σύζευξη έχει επίσης στατιστική σημαντικότητα 0 με όφελος 99. Στην τμηματικότητα η στατιστική σημαντικότητα είναι μεγαλύτερη του ορίου οπότε υπάρχει πιθανότητα σφάλματος, ωστόσο στα άλλα δύο χαρακτηριστικά η στατιστική σημαντικότητα είναι μηδέν γεγονός που δηλώνει ότι η πιθανότητα σφάλματος είναι μηδενική.

Πίνακας 18 : Έλεγχος Υποθέσεων Δείγματος

Χαρακτηριστικό Ποιότητας	Όφελος	Ουδέτερο	Απώλεια	Στατιστική Σημαντικότητα
Τμηματικότητα	57	4	76	0.146
Συνοχή	11	5	121	0
Σύζευξη	99	17	21	0

7.2 Αποτελεσματικότητα Αναδόμησης σε σχέση με το Πλήθος Κλάσεων προς Μετακίνηση (RQ2)

Στον Πίνακα 19 παρουσιάζονται τα περιγραφικά χαρακτηριστικά ποιότητας ανά πλήθος αναδόμησης. Πιο συγκεκριμένα, σε κάθε γραμμή, για κάθε ένα πλήθος αναδόμησης (πλήθος κλάσεων ανά ομάδα n=1,2,3,4,5), παρουσιάζεται ένα από αυτά τα χαρακτηριστικά, τα οποία είναι η τμηματικότητα, η συνοχή και η σύζευξη. Οι στήλες χωρίζονται σε δύο κατηγορίες, στις στατιστικές τιμές πριν την αναδόμηση και στις στατιστικές τιμές μετά την αναδόμηση. Τα πεδία στις στήλες είναι ίδια με αυτά του Πίνακα 17 δηλαδή, η ελάχιστη τιμή που έλαβε το κάθε χαρακτηριστικό, η μέγιστη τιμή που έλαβε το κάθε χαρακτηριστικό, η μέση τιμή και η τυπική απόκλιση. Σε κάθε γραμμή του πίνακα με γαλάζια σκίαση κελιού αποτυπώνετε εάν η βέλτιστη τιμή αναγνωρίζεται πριν ή μετά την αναδόμηση. Από τα αποτελέσματα του πίνακα παρατηρείτε ότι, η τμηματικότητα του πακέτου βελτιώνεται από την αναδόμηση στη περίπτωση μετακίνησης μίας ή πέντε κλάσεων (μεγέθους αναδόμησης ένα ή πέντε). Η σύζευξη βελτιώνεται σε όλες τις περιπτώσεις εκτός της μετακίνησης τριών κλάσεων, ενώ η συνοχή χειροτερεύει σε όλες τις περιπτώσεις, ανεξαρτήτως του πλήθους των κλάσεων που μετακινούνται (μεγέθους αναδόμησης). Το γεγονός ότι η τμηματικότητα βελτιώνεται παρά τη μείωση της συνοχής των πακέτων δεν υποδεικνύει ότι η συνοχή έχει λιγότερη συμμετοχή στον υπολογισμό της τμηματικότητας, αλλά πιθανότατα οφείλεται στη μικρότερη ποσοστιαία μεταβολή της.

Πίνακας 19 : Περιγραφικά Στατιστικά ανά Μέγεθος Αναδόμησης

N	Χαρακτηριστικό Ποιότητας	Πριν				Μετά			
		Ελάχιστο	Μέγιστο	Μέση τιμή	Τυπική Απόκλιση	Ελάχιστο	Μέγιστο	Μέση τιμή	Τυπική Απόκλιση
1	Τμηματικότητα	0.21884	500	19.9002	90.773482	0	333.3333	34.2989	84.128507
	Συνοχή	0.18182	1	0.52480	0.1909832	0	1	0.46936	0.2313765
	Σύζευξη	0.001	3.86652	0.78959	1.4017352	0	1.3663	0.28804	0.4200545
2	Τμηματικότητα	0.21884	500	54.4067	125.64705	0	400	38.3715	93.660627
	Συνοχή	0.18182	0.84615	0.56323	0.1838881	0	0.86111	0.35362	0.2513019
	Σύζευξη	0.001	3.86652	0.75982	1.4458529	0	1.1663	0.17991	0.2937982
3	Τμηματικότητα	0.54382	285.714	36.8333	84.998202	0	17.53846	4.55802	4.8599389
	Συνοχή	0.18182	1	0.67083	0.2687795	0	0.75	0.33059	0.2397386
	Σύζευξη	0.001	0.33433	0.13395	0.0832087	0	0.875	0.14744	0.2134541
4	Τμηματικότητα	2.73	285.714	19.8126	56.266991	0	13.26781	4.08284	4.8765998
	Συνοχή	0.28571	1	0.73378	0.2368903	0	0.81818	0.25585	0.2037066
	Σύζευξη	0.001	0.19048	0.14041	0.0632952	0	0.16286	0.06609	0.0575945
5	Τμηματικότητα	0.21884	34.0909	5.08162	6.6889823	0	32.5	6.79520	7.6213314
	Συνοχή	0.52	1	0.7342	0.2125291	0	1	0.43986	0.3101448
	Σύζευξη	0.02	3.86652	0.64879	1.2747417	0	3.63982	0.52330	1.2017697

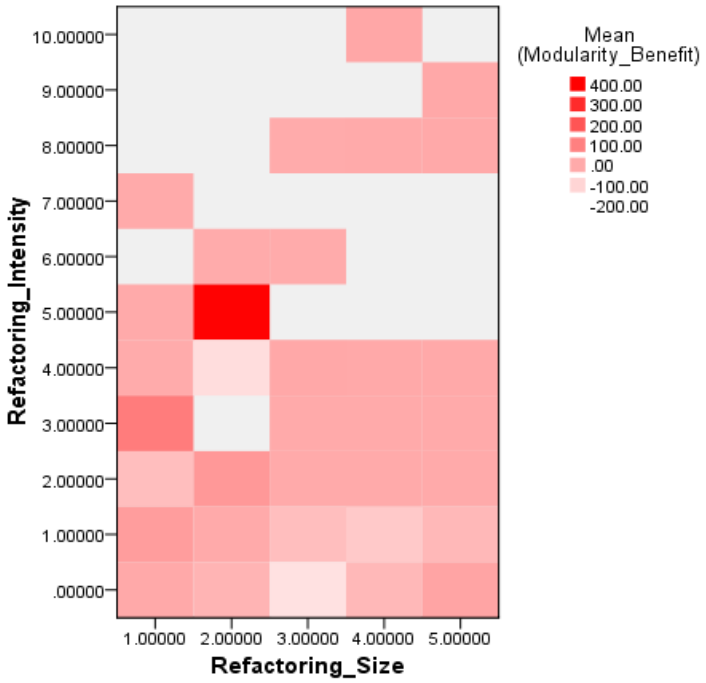
Στον Πίνακα 20 παρουσιάζονται τα χαρακτηριστικά ποιότητας σε σχέση με τον έλεγχο υποθέσεων ανά πλήθος ομάδων αναδόμησης. Ειδικά, σε κάθε γραμμή, για κάθε ένα πλήθος αναδόμησης (πλήθος κλάσεων ανά ομάδα n=1,2,3,4,5), περιλαμβάνει ένα από τα χαρακτηριστικά ποιότητας που ερευνήθηκαν. Η κάθε στήλη περιλαμβάνει τη διαφορά μεταξύ των αρχικών και των τελικών τιμών, δηλαδή των τιμών πριν την αναδόμηση και μετά την αναδόμηση. Ο Πίνακας 20 ακολουθεί την ίδια δομή με τον Πίνακα 18. Οι γραμμές του

πίνακα οι οποίες έχουν χρωματιστεί γαλάζιο είναι αυτές στις οποίες το κάθε χαρακτηριστικό έχει τη μεγαλύτερη βελτίωση ως προς τη στατιστική σημαντικότητα. Ως όριο στη στατιστική σημαντικότητα χρησιμοποιήθηκε το 0.05 (επεξήγηση στο Κεφάλαιο 6.4). Σκοπός είναι η στατιστική σημαντικότητα να πλησιάζει ή να είναι μηδέν. Όπως φαίνεται στον Πίνακα 20 το χαρακτηριστικό τμηματικότητα πετυχαίνει καλύτερη στατιστική σημαντικότητα σε μέγεθος αναδόμησης ένα, η συνοχή σε μέγεθος αναδόμησης δύο, τρία, τέσσερα και πέντε, ενώ η σύζευξη σε μέγεθος αναδόμησης ένα, τέσσερα και πέντε. Συμπερασματικά, φαίνεται ότι οι ακραίες τιμές έχουν τα καλύτερα αποτελέσματα.

Πίνακας 20 : Έλεγχος Υποθέσεων ανά Μέγεθος Αναδόμησης

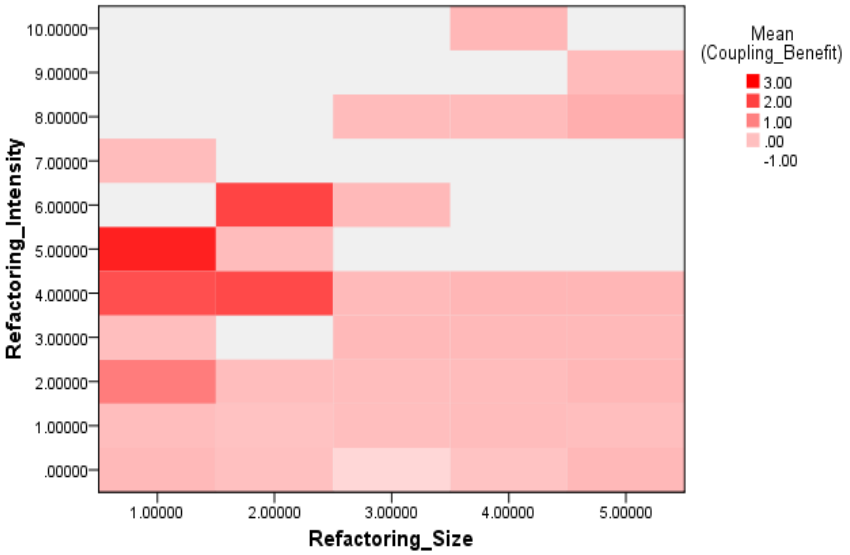
N	Χαρακτηριστικό Ποιότητας	Όφελος	Ουδέτερο	Απώλεια	Στατιστική Σημαντικότητα
1	Τμηματικότητα	22	1	7	0.002
	Συνοχή	6	1	23	0.012
	Σύζευξη	22	6	2	0
2	Τμηματικότητα	8	0	21	0.05
	Συνοχή	3	0	26	0
	Σύζευξη	16	5	8	0.055
3	Τμηματικότητα	6	0	24	0.012
	Συνοχή	0	0	30	0
	Σύζευξη	20	3	7	0.016
4	Τμηματικότητα	7	0	18	0.115
	Συνοχή	1	0	24	0
	Σύζευξη	21	0	4	0
5	Τμηματικότητα	14	3	6	0.047
	Συνοχή	1	4	18	0
	Σύζευξη	20	3	0	0

Ως συνέχεια των αποτελεσμάτων του ερευνητικού ερωτήματος 2 (RQ2) παρουσιάζονται τρία γραφήματα θερμότητας, ένα για κάθε χαρακτηριστικό. Μέσω των γραφημάτων γίνεται ευκολότερα κατανοητό και εμφανές σε ποιο μέγεθος αναδόμησης επιτυγχάνεται μεγαλύτερη βελτίωση των χαρακτηριστικών ποιότητας ως προς τη μέση τιμή. Στην Εικόνα 34 φαίνεται το διάγραμμα θερμότητας για το χαρακτηριστικό ποιότητας τμηματικότητα. Παρατηρείται ότι, για μέγεθος αναδόμησης δύο επιτυγχάνεται η μεγαλύτερη μέση τιμή. Το συμπέρασμα αυτό επιβεβαιώνεται και από τον Πίνακα 19.



Εικόνα 34 : Διάγραμμα Θερμότητας (HeatMap) Τμηματικότητας

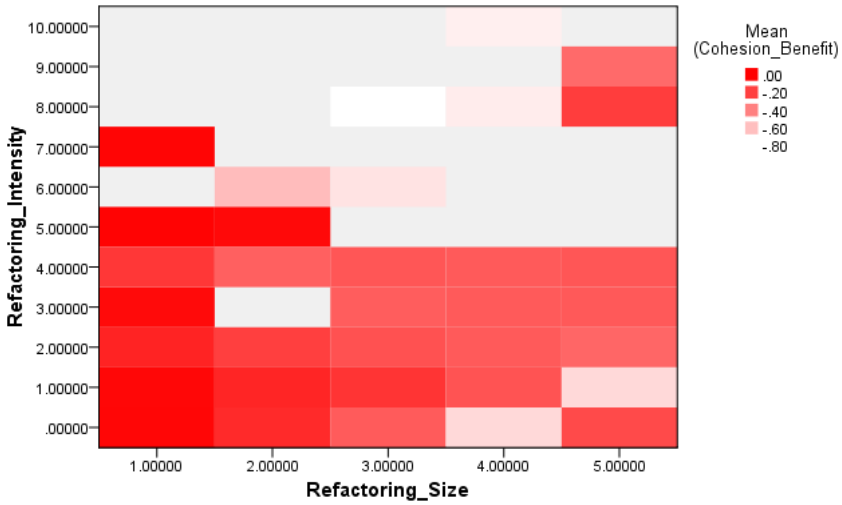
Στην Εικόνα 35 φαίνεται το διάγραμμα θερμότητας για το χαρακτηριστικό ποιότητας σύζευξη. Παρατηρείται ότι για μέγεθος αναδόμησης ένα, η τιμή της μετρικής σύζευξης παίρνει τη μεγαλύτερη τιμή. Το συμπέρασμα αυτό επιβεβαιώνεται και από τον Πίνακα 19.



Εικόνα 35 : Διάγραμμα Θερμότητας (HeatMap) Σύζευξης

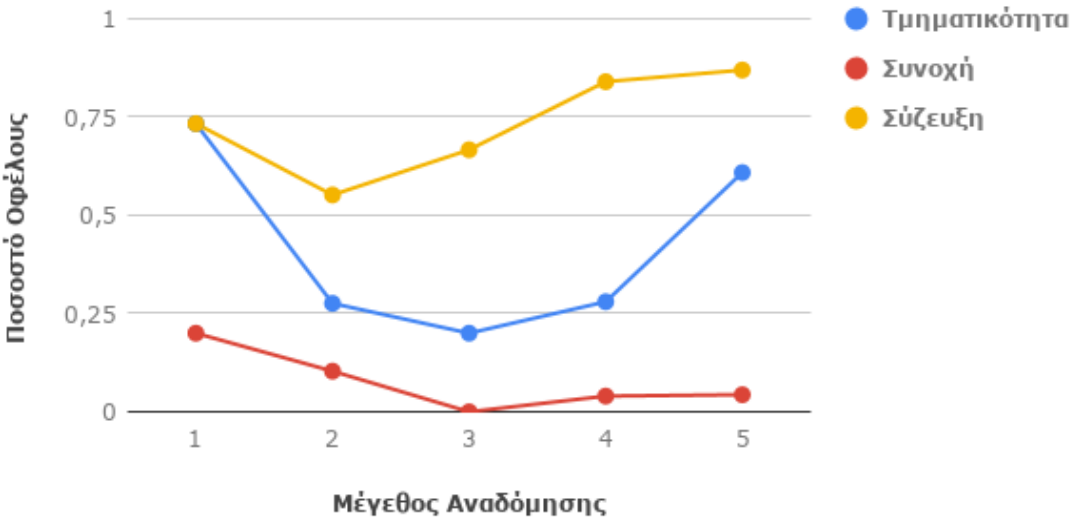
Τέλος, στην Εικόνα 36 απεικονίζεται το διάγραμμα θερμότητας του χαρακτηριστικού ποιότητας συνοχή. Γίνεται εμφανές ότι η συνοχή σε μέγεθος αναδόμησης ένα, έχει τη μεγαλύτερη τιμή. Το συμπέρασμα αυτό επιβεβαιώνεται και από τον Πίνακα 19.

Συμπερασματικά, τα χαρακτηριστικά ποιότητας αποδεικνύεται ότι έχουν μεγαλύτερες τιμές στις ακραίες τιμές του εύρους αναδόμησης από ότι στις μεσαίες. Συμπέρασμα αναμενόμενο διότι, συνήθως η εξάρτηση είναι είτε μεταξύ δύο κλάσεων οπότε με μία μετακίνηση βελτιώνονται οι τιμές είτε υπάρχει υψηλή εξάρτηση μεταξύ μιας ομάδας κλάσεων και με την μετακίνηση ολόκληρης της ομάδας τα χαρακτηριστικά ποιότητας βελτιώνονται.



Εικόνα 36 : Διάγραμμα Θερμότητας (HeatMap) Συνοχής

Στην Εικόνα 37 απεικονίζεται το διάγραμμα οφέλους όλων των χαρακτηριστικών προς το μέγεθος αναδόμησης. Παρατηρείται ότι, για το χαρακτηριστικό συνοχή καθώς αυξάνεται το μέγεθος αναδόμησης, η πιθανότητα η αναδόμηση να επιφέρει όφελος, μειώνεται. Στο μέγεθος αναδόμησης τρία μηδενίζεται, στη συνέχεια αποκτά αργή ανοδική πορεία και προς το τέλος σταθεροποιείται. Αντίθετα, η πιθανότητα βελτίωσης του χαρακτηριστικού σύζευξης συνεχώς αυξάνεται με μια μικρή μείωση στο μέγεθος δύο. Τέλος, το χαρακτηριστικό τμηματικότητα ακολουθεί παρόμοια πορεία με αυτή της σύζευξης. Οι δύο ακραίες τιμές του μεγέθους αναδόμησης δίνουν το μεγαλύτερο όφελος ενώ οι μεσαίες τιμές έχουν μικρότερες πιθανότητες επιτυχίας.



Εικόνα 37 : Συγκριτικό Διάγραμμα Οφέλους Ποιότητας Χαρακτηριστικών

7.3 Συσχέτιση Μετακινούμενων Κλάσεων με την Εννοιολογική Ομοιότητα του Πακέτου Στόχου (RQ3)

"I haven't failed. I've just found 10,000 ways that won't work."
— Thomas Edison

Στον Πίνακα 21 παρουσιάζονται περιγραφικά τα χαρακτηριστικά ποιότητας σε σχέση με την εννοιολογική ομοιότητα μεταξύ των κλάσεων που μεταφέρονται και του πακέτου προορισμού. Πιο συγκεκριμένα, σε κάθε γραμμή, για κάθε μία δυνατή επιλογή (Ε.Ο Εννοιολογική Ομοιότητα), δηλαδή αν είναι όμοια (✓) ή δεν είναι όμοια (✗) παρουσιάζονται

τα αποτελέσματα των χαρακτηριστικών ποιότητας, τα οποία είναι η τμηματικότητα, η συνοχή και η σύζευξη. Οι στήλες χωρίζονται σε δύο κατηγορίες, στις στατιστικές τιμές πριν την αναδόμηση και στις στατιστικές τιμές μετά την αναδόμηση. Τα πεδία στις στήλες είναι ίδια με αυτά του Πίνακα 17 δηλαδή, η ελάχιστη τιμή που έλαβε το κάθε χαρακτηριστικό, η μέγιστη τιμή που έλαβε το κάθε χαρακτηριστικό, η μέση τιμή και η τυπική απόκλιση. Σε κάθε γραμμή του πίνακα με γαλάζια σκίαση κελιού αποτυπώνετε εάν η βέλτιστη τιμή αναγνωρίζεται πριν ή μετά την αναδόμηση. Από τα αποτελέσματα του πίνακα παρατηρείται ότι, τα χαρακτηριστικά ποιότητας τμηματικότητα και συνοχή είναι βέλτιστα πριν την αναδόμηση, στην περίπτωση που υπάρχει εννοιολογική ομοιότητα και στην περίπτωση που δεν υπάρχει ενώ, το χαρακτηριστικό σύζευξη και στις δύο περιπτώσεις είναι καλύτερο μετά την αναδόμηση.

Πίνακας 21 : Περιγραφικά Στατιστικά σε σχέση με την Εννοιολογική Ομοιότητα

Ε. Ο.	Χαρακτηριστικό Ποιότητας	Πριν				Μετά			
		Ελάχιστο	Μέγιστο	Μέση τιμή	Τυπική Απόκλιση	Ελάχιστο	Μέγιστο	Μέση τιμή	Τυπική Απόκλιση
✗	Τμηματικότητα	0.218841	500	29.6459	92.057832	0	400	18.8793	61.515808
	Συνοχή	0.181818	1	0.64851	0.2428224	0	1	0.36641	0.2640317
	Σύζευξη	0.001	3.86652	0.55813	1.1655376	0	3.639820	0.25489	0.6044105
✓	Τμηματικότητα	1.236559	34.0909	20.2299	15.997257	0	208.3333	16.1224	48.430340
	Συνοχή	0.322581	0.73333	0.57012	0.1548424	0	0.763636	0.39799	0.1950025
	Σύζευξη	0.02	0.26087	0.09757	0.1024644	0	0.333333	0.09103	0.1107766

Στον Πίνακα 22 απεικονίζονται τα χαρακτηριστικά ποιότητας σε σχέση με τον έλεγχο υποθέσεων ανά περίπτωση εννοιολογικής ομοιότητας. Ειδικά, σε κάθε γραμμή, για κάθε μία δυνατή επιλογή, δηλαδή αν είναι όμοια (✓) ή δεν είναι όμοια (✗) παρουσιάζονται, τα χαρακτηριστικά ποιότητας που ερευνήθηκαν. Η κάθε στήλη περιλαμβάνει τη διαφορά μεταξύ των αρχικών και των τελικών τιμών, δηλαδή των τιμών πριν την αναδόμηση και μετά την αναδόμηση. Ο Πίνακας 22 ακολουθεί την ίδια δομή με τον Πίνακα 18. Οι γραμμές του

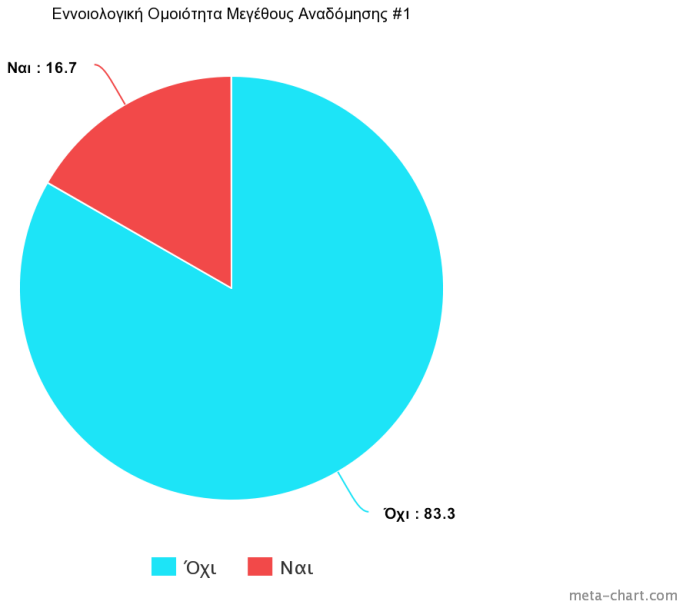
πίνακα οι οποίες έχουν χρωματιστεί γαλάζιο είναι αυτές στις οποίες το κάθε χαρακτηριστικό έχει τη μεγαλύτερη βελτίωση ως προς τη στατιστική σημαντικότητα. Είναι εμφανές ότι, η στατιστική σημαντικότητα των χαρακτηριστικών συνοχή και σύζευξη είναι μηδενική όταν δεν υπάρχει εννοιολογική ομοιότητα μεταξύ των μετακινούμενων κλάσεων και των πακέτων προορισμού ενώ το χαρακτηριστικό τμηματικότητα είναι κάτω από το επιθυμητό όριο όταν υπάρχει εννοιολογική ομοιότητα.

Πίνακας 22 : Έλεγχος Υποθέσεων σε σχέση με την Εννοιολογική Ομοιότητα

Εννοιολογική Ομοιότητα	Χαρακτηριστικό Ποιότητας	Όφελος	Ουδέτερο	Απώλεια	Στατιστική Σημαντικότητα
✗	Τμηματικότητα	55	4	60	0.885
	Συνοχή	9	5	105	0
	Σύζευξη	92	14	13	0
✓	Τμηματικότητα	2	0	16	0.007
	Συνοχή	2	0	16	0.001
	Σύζευξη	7	3	8	0.69

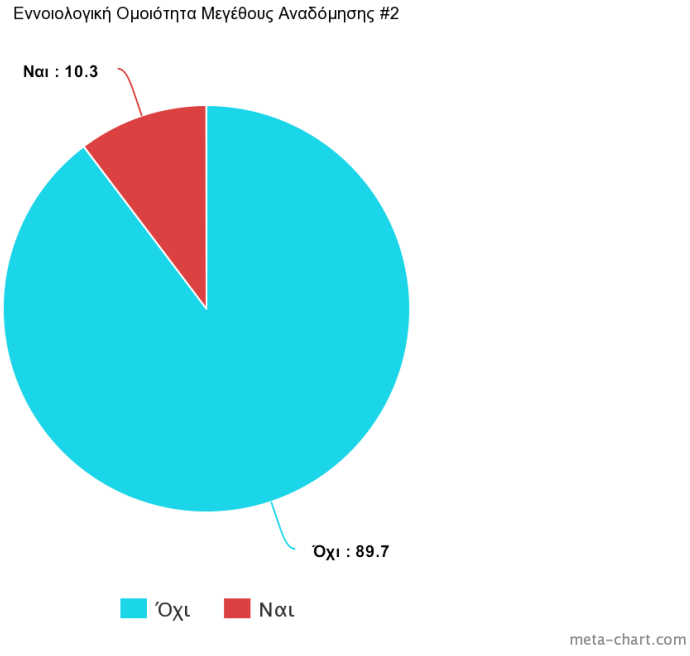
Στη συνέχεια παρουσιάζονται έξι κυκλικά διαγράμματα (pie chart) για καλύτερη απεικόνιση των ποσοστών εννοιολογικής ομοιότητας και μη εννοιολογικής ομοιότητας σε κάθε περίπτωση, δηλαδή ένα για κάθε μέγεθος αναδόμησης και ένα συνολικό όλου του δείγματος.

Στην Εικόνα 38 απεικονίζεται το ποσοστό δειγμάτων το οποίο είχε εννοιολογική ομοιότητα 16.7% ενώ το ποσοστό που δεν είχε εννοιολογική ομοιότητα ήταν 83.3%. Το πλήθος δειγμάτων ήταν 30. Η απεικόνιση αφορά το μέγεθος αναδόμησης μίας κλάσης.



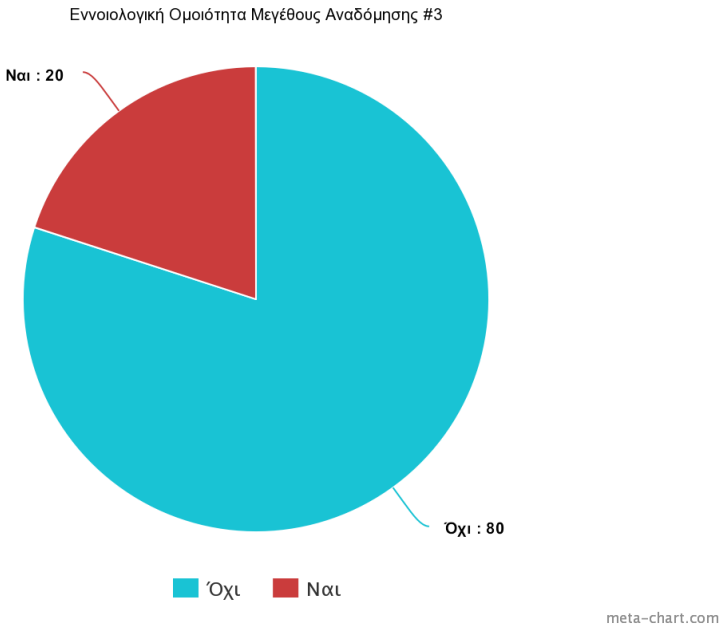
Εικόνα 38 : Διάγραμμα Εννοιολογικής Ομοιότητας Μεγέθους Αναδόμησης #1

Στην Εικόνα 39 απεικονίζεται το ποσοστό δειγμάτων το οποίο είχε εννοιολογική ομοιότητα 10.3% ενώ το ποσοστό που δεν είχε εννοιολογική ομοιότητα ήταν 89.7%. Το πλήθος δειγμάτων ήταν 29. Η απεικόνιση αφορά το μέγεθος αναδόμησης δύο κλάσεων.



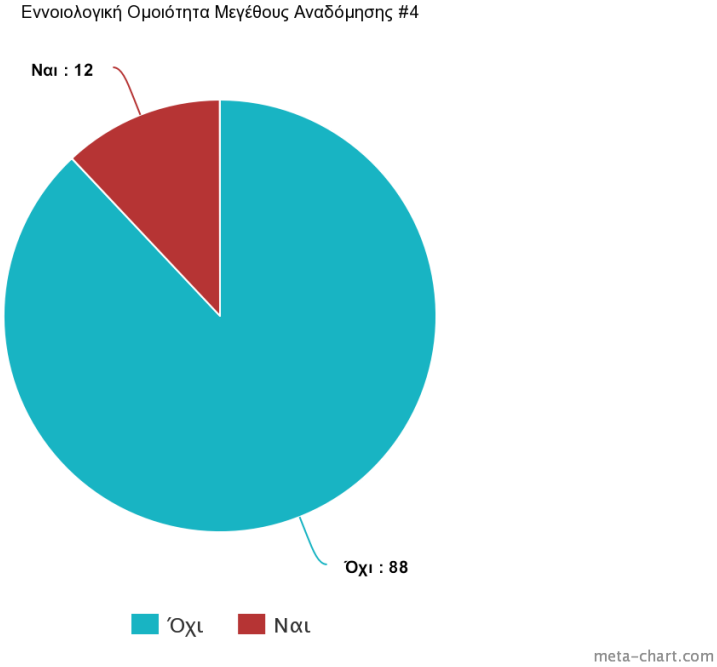
Εικόνα 39 : Διάγραμμα Εννοιολογικής Ομοιότητας Μεγέθους Αναδόμησης #2

Στην Εικόνα 40 απεικονίζεται το ποσοστό δειγμάτων το οποίο είχε εννοιολογική ομοιότητα 20% ενώ το ποσοστό που δεν είχε εννοιολογική ομοιότητα ήταν 80%. Το πλήθος δειγμάτων ήταν 30. Η απεικόνιση αφορά το μέγεθος αναδόμησης τριών κλάσεων.



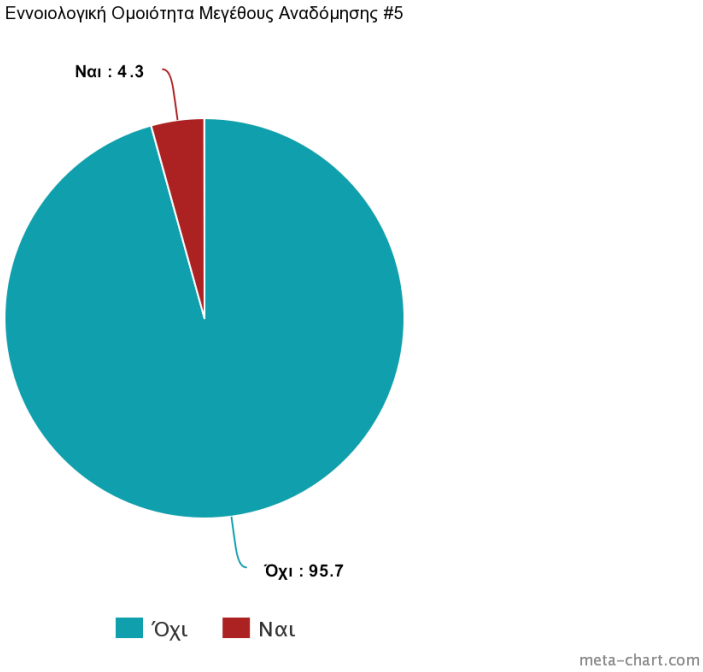
Εικόνα 40 : Διάγραμμα Εννοιολογικής Ομοιότητας Μεγέθους Αναδόμησης #3

Στην Εικόνα 41 απεικονίζεται το ποσοστό δειγμάτων το οποίο είχε εννοιολογική ομοιότητα 12% ενώ το ποσοστό που δεν είχε εννοιολογική ομοιότητα ήταν 88%. Το πλήθος δειγμάτων ήταν 25. Η απεικόνιση αφορά το μέγεθος αναδόμησης τεσσάρων κλάσεων.



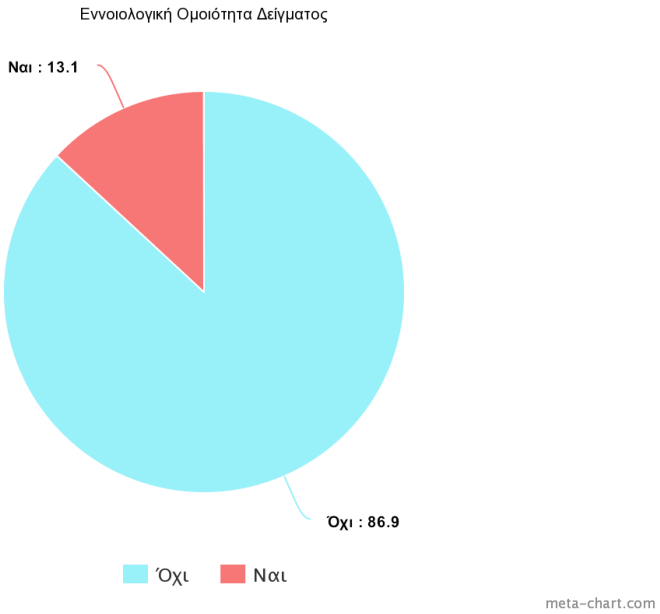
Εικόνα 41 : Διάγραμμα Εννοιολογικής Ομοιότητας Μεγέθους Αναδόμησης #4

Στην Εικόνα 42 απεικονίζεται το ποσοστό δειγμάτων το οποίο είχε εννοιολογική ομοιότητα 4.3% ενώ το ποσοστό που δεν είχε εννοιολογική ομοιότητα ήταν 95.7%. Το πλήθος δειγμάτων ήταν 23. Η απεικόνιση αφορά το μέγεθος αναδόμησης πέντε κλάσεων.



Εικόνα 42 : Διάγραμμα Εννοιολογικής Ομοιότητας Μεγέθους Αναδόμησης #5

Τέλος, στην Εικόνα 43 απεικονίζεται το ποσοστό δειγμάτων το οποίο είχε εννοιολογική ομοιότητα 13.1% ενώ το ποσοστό που δεν είχε εννοιολογική ομοιότητα ήταν 86.9%. Το πλήθος δειγμάτων ήταν 137. Η απεικόνιση αφορά ολόκληρο το δείγμα, όλων των μεγεθών αναδόμησης.



Εικόνα 43 : Διάγραμμα Εννοιολογικής Ομοιότητας Συνολικού Δείγματος

8. Συζήτηση – Συμπεράσματα

"No matter how good you get you can always get better, and that's the exciting part."
— Tiger Woods

Η παρούσα διπλωματική εργασία είχε ως στόχο την πρόταση και αξιολόγηση μιας μεθοδολογίας πρότασης αναδομήσεων σε επίπεδο πακέτου λογισμικού με στόχο τη μείωση του τεχνικού χρέους. Η μεθοδολογία βασίστηκε στη βελτίωση της τμηματικότητας του παραγόμενου λογισμικού, κάνοντας χρήση μετρικών σύζευξης και συνοχής. Ο αλγόριθμος που αναπτύχθηκε, υλοποιήθηκε σε ένα εργαλείο με γλώσσα προγραμματισμού Java με στόχο την αυτοματοποίηση της διαδικασίας η οποία θα ήταν ανέφικτη χωρίς υπολογιστικά μέσα.

Κατά την αξιολόγηση εφαρμόστηκε η μεθοδολογία σε 3 έργα ανοιχτού λογισμικού και μέσω ελέγχου υποθέσεων εξετάστηκε το εάν η εφαρμογή των προτεινόμενων αναδομήσεων βελτιώνει την τμηματικότητα. Η εμπειρική διερεύνηση μέσω μελέτης περιπτώσεων που διενεργήθηκε οδήγησε στα παρακάτω αποτελέσματα:

- ***Το μέγεθος της αναδόμησης είναι σημαντικός παράγοντας της επιτυχίας της μεθόδου***. Τα εμπειρικά αποτελέσματα έδειξαν ότι η τμηματικότητα του λογισμικού βελτιώνεται σημαντικά κατά τη μετακίνηση μιας ή πέντε κλάσεων. Η βελτίωση κατά τη μετακίνηση μιας κλάσης υποστηρίζεται και από τη σχετική βιβλιογραφία, ενώ η βελτίωση της τμηματικότητας όσο το πλήθος το κλάσεων που μετακινούνται αυξάνει είναι ένα νέο εύρημα για την ερευνητική περιοχή.
- ***Σύγκριση συνοχής και σύζευξης.*** Τα αποτελέσματα της μελέτης δείχνουν ότι, η σύζευξη επηρεάζεται θετικά από τη μεθοδολογίας, ενώ η συνοχή αρνητικά. Παρόμοια αποτελέσματα μπορούν να υποστηριχθούν από τη βιβλιογραφία καθώς τα δυο χαρακτηριστικά ποιότητας έχουν συνήθως αντίστροφη σχέση (η αύξηση της συνοχής οδηγεί σε μείωση της σύζευξης και το αντίστροφο). Επιπλέον, το γεγονός ότι η μέθοδος βελτιώνει (δηλαδή μειώνει) τα επίπεδο σύζευξης είναι αναμενόμενο, καθώς η «λογική» της μεθοδολογίας είναι οδηγούμενη από την εξάλειψη εξαρτήσεων μεταξύ πακέτων (inter-package relations), ενώ η μείωση των εσωτερικών εξαρτήσεων (intra-package relations) είναι δευτερεύουσα παράμετρος.

Τα παραπάνω αποτελέσματα οδηγούν σε κάποιες χρήσιμες κατευθύνσεις για μελλοντική έρευνα. Αρχικά, χρειάζεται η βελτίωση του αλγορίθμου ώστε να βελτιστοποιεί ταυτόχρονα τόσο τη σύζευξη όσο και τη συνοχή. Στη συνέχεια, απαιτείται η βελτίωση του γραφικού περιβάλλοντος και της εμφάνισης των αποτελεσμάτων στο χρήστη. Αυτό σημαίνει ένα πιο εύχρηστο και ευχάριστο περιβάλλον και η απεικόνιση των αναδομήσεων που έχουν περισσότερες πιθανότητες επιτυχίας δηλαδή, η πληροφορία αυτή η οποία είναι χρήσιμη για τον τελικό χρήστη.

“I think that’s the single best piece of advice: constantly think about how you could be doing things better and questioning yourself.”
— Elon Musk

Βιβλιογραφία

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

[2] R. Marinescu, "Measurement and Quality in Object-Oriented Design," Phd dissertation, “Politehinca” University of Timişoara, Romanian, 2002.

[3] R. C. Martin, Agile Software Development: Principles, Patterns, and Practices, Prentice Hall, 2003.

[4] J. A. Riel, Object-Oriented Design Heuristics, Addison-Wesley, 1996.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[6] W. H. Brown, R. C. Malveau, H. W. McCormick, and T. J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, John Wiley and Sons, 1998.

[7] J. Kerievsky, Refactoring to Patterns, Addison-Wesley, 2004.

[8] B. Du Bois, S. Demeyer, J. Verelst, "Refactoring-Improving Coupling and Cohesion of Existing Code," 11th Working Conference on Reverse Engineering (WCRE '04), Delft University of Technology, the Netherlands, November 2004, pp. 144-151.

[9] F. Bourqun, and R. K. Keller, "High-impact Refactoring Based on Architecture Violations," 11th European Conference of Software Maintenance and Reengineering (CSMR '07), Amsterdam, Netherlands, March 2007, pp. 149-158.

[10] Y. Kataoka, T. Imai, H. Andou, T. Fukaya, "A Quantitative Evaluation of Maintainability Enhancement by Refactoring," 18th International Conference on Software Maintenance (ICSM '02), Montréal, Canada, October 2002, pp. 576-585.

[11] L. Tahvildari, and K. Kontogiannis, "A Metric-Based Approach to Enhance Design Quality through Meta-pattern Transformations," 7th European Conference on Software Maintenance and Reengineering (CSMR '03), Benevento, Italy, March 2003, pp. 183-192.

[12] F. M. Haney, “Module connection analysis: A tool for scheduling of software debugging activities,” Proceedings of the AFIPS Fall Joint Computer Conference, pp. 173-179, 5-7 December 1972, USA.

[13] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Predicting the Probability of Change in Object-Oriented Systems", Transactions on Software Engineering, IEEE Computer Society, 31 (7), pp. 601-614, July 2005.

[14] H. van Vliet, “Software Engineering: Principles and Practice”, John Wiley & Sons, 2008.

[15] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, “Introducing a ripple effect measure: a theoretical and empirical validation”, 9 th International Symposium on Empirical Software Engineering and Measurement (ESEM 2015), Beijing, China, IEEE Computer Society, 22–23 October 2015.

[16] S. R. Chidamber and C. F. Kemerer, “A Metrics Suite for Object Oriented Design”, Transactions on Software Engineering, IEEE Computer Society, 20 (6), pp. 476 - 493, June 1994.

[17] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series), 2008.

[18] R. Mahouachi, K. Ghedira , “Data fusion for software remodularization”, 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2017), Vienna, Austria, IEEE Computer Society, 30 August - 1 September 2017

[19] S. M. A. Shah, J. Dietrich, C. McCartin, "Making Smart Moves to Untangle Programs", 16th European Conference on Software Maintenance and Reengineering (CSMR 2012), Szeged, Hungary, IEEE Computer Society, 72–30 March 2012.

[20] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "A method for Asserting Class Change Proneness", 21st International Conference on Evaluation and Assessment in Software Engineering (EASE 2017), Karlskrova, Sweden, IEEE Computer Society, 15–16 June 2017

[21] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, A. Gkortzis and P. Avgeriou, "Identifying Extract Method Refactoring Opportunities based on Functional Relevance", Transactions on Software Engineering, IEEE Computer Society, 43 (10), pp. 954-974, December 2016.

[22] A. Ampatzoglou, A. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou, "A Financial Approach for Managing Interest in Technical Debt", LNBIP, Springer, vol. 257, pp. 117-133, 2016.

[23] 1061-1998: IEEE Standard for a Software Quality Metrics Methodology, IEEE Standards, IEEE Computer Society, and 31 December 1998 (re-affirmed 9 December 2009).

[24] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review", Information and Software Technology, Volume 64, August 2015, pp 52-73.

[25] S. Charalampidou, A.Ampatzoglou, and P.Avgeriou., "Size and cohesion metrics as indicators of the long method bad smell: An empirical study", 11th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '15). ACM, New York, NY, USA, Article 8, 10 pages, 2015.

[26] P. Runeson, M. Host, A. Rainer and B. Regnell, "Case Study Research in Software Engineering: Guidelines and Examples", John Wiley & Sons, 2012. [23] R. Schwanke, L.

Xiao, and Y. Cai, “Measuring architecture quality by structure plus history analysis”, 35th International Conference on Software Engineering (ICSE 2013), San Francisco, USA, ACM/IEEE Computer Society, pp. 891-900, 18-26 May 2013.

[27] Areti. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun and K. Systa, "A perception of Technical Debt in the Embedded Systems Domain : An Industrial Case Study", 8th International Workshop on Managing Technical Debt (MTD 2016), Raleigh, NC, USA, 8 October 2016

[28] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster, and P. Avgeriou, “A Mapping Study on Design-Time Quality Attributes and Metrics”, Journal of Systems and Software, Elsevier, accepted for publication.

[29] N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, C. Seaman, “Identification and management of technical debt: A systematic mapping study”, Information and Software Technology, Volume 70, pp.100-121, February 2016

[30] A. Chatzigeorgiou and A. Manakos, “Investigating the evolution of code smells in object-oriented systems”, Innov. Syst. Softw. Eng. 10(1), pp. 3-18. March 2014.

[31] N. Yoshida, M. Kinoshita, H. Iida, “A cohesion metric approach to dividing source code into functional segments to improve maintainability”, 16th European Conference on Software Maintenance and Reengineering (CSMR), pp.365-370. 27-30 March 2012.

[32] A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou and P. Avgeriou, “The Effect of Gof Design Patterns on Stability: A Case Study”, Transactions on Software Engineering, IEEE Computer Society, 41 (8), pp. 781-802, March 2015

[33] E. M. Arvanitou, A. Ampatzoglou, K. Tzouvalidis, A. Chatzigeorgiou, P. Avgeriou, I. Deligiannis, “Assesing Change Proneness at the Architecture Level: An Empirical Validation”, 24th Asis-Pacific Software Engineering Conference Workshops (APSECW 2017), Nanjing, China, 4-8 December 2017

[34] N. Tsantalis, A. Chatzigeorgiou, "Identification of Move Method Refactoring Opportunities", IEEE Transaction Software Engineering, 35 (3), pp. 347-367, May 2009

[35] P. Kruchen, R. L. Nord, I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice", IEEE Software, 29(6), pp. 18-21, October 2012

[36] Refactoring, Bad code smells, sourcemaking.com/refactoring (Επίσκεψη 24/06/2018)