Διερευνηση Ιστορικων Δεδομενων Αναφορικα με Προβληματα Σχεδιασης σε Αντικειμενοστρεφη Συστηματα Λογισμικου

Αναστασιος Μ. Μανακος

mai 32/09

Θεσσαλονικη, Ιουνιος 2010

Τμημα Εφαρμοσμενης Πληροφορικης
Πανεπιστημιο Μακεδονιας
Department of Applied Informatics
University of Macedonia

ΔΙΕΡΕΥΝΗΣΗ ΙΣΤΟΡΙΚΩΝ ΔΕΔΟΜΕΝΩΝ ΑΝΑΦΟΡΙΚΑ ΜΕ ΠΡΟΒΛΗΜΑΤΑ ΣΧΕΔΙΑΣΗΣ ΣΕ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

υποβάλλεται στην
ορισθείσα από τη Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Εφαρμοσμένης Πληροφορικής Εξεταστική Επιτροπή

από τον
ΑΝΑΣΤΑΣΙΟ ΜΑΝΑΚΟ

ως μέρος των υποχρεώσεων για τη λήψη του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ
ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ
ΣΤΑ ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΤΩΝ

ΙΟΥΝΙΟΣ 2010

© Αναστάσιος Μ. Μανάκος, 2010. *Με επιφύλαξη κάθε νόμιμου δικαιώματος.*

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του τμήματος Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας.

ΑΦΙΕΡΩΣΗ

Στη μνήμη της γιαγιάς μου,
"Μαρία Μ. Μπαμίχα-Μανάκου"

ΕΥΧΑΡΙΣΤΙΕΣ

Η ανάπτυξη και η συγγραφή μιας διπλωματικής εργασίας είναι εξαιρετικά επίπονη και χρονοβόρα διαδικασία, πόσο μάλλον σε περίπτωση που πραγματοποιείται κάτω από πίεση χρόνου. Η λίστα των ανθρώπων που ακολουθεί, περιλαμβάνει όσους συνέφεραν με το δικό τους τρόπο για την περάτωση της παρούσας διπλωματικής εργασίας.

Θα ήθελα λοιπόν αρχικά, να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή της διπλωματικής μου, κ.Αλέξανδρο Ν. Χατζηγεωργίου, επίκουρο καθηγητή, για την εμπιστοσύνη που έδειξε στο πρόσωπό μου, την πολύτιμη καθοδήγησή του, το άψογο πνεύμα συνεργασίας που εκδήλωσε, τις σπουδαίες γνώσεις και εμπειρίες που αποκόμισα κατά τη διάρκεια της συνεργασίας μαζί του καθώς επίσης και το πάθος και τον ενθουσιασμό του που μου μετέδωσε για τη συγκεκριμένη διπλωματική εργασία και γενικότερα για την έρευνα.

Επιπλέον, ευχαριστώ την συνάδελφο και καλή μου φίλη Θεοδώρα Γ. Αναστασίου για όλους τους γνωστούς λόγους (ηθική υποστήριξη, παρηγοριά στις δύσκολες στιγμές και τις καλοπροαίρετα κάθε φορά παρατηρήσεις).

Τέλος, θα ήθελα να ευχαριστήσω την αδερφή μου, Μαρία Μ. Μανάκου για τις ώρες μελέτης που μου πρόσφερε ώστε να αποκτήσω το απαραίτητο υπόβαθρο για να πετύχω στις πανελλαδικές εξετάσεις αλλά και τους γονείς μου, Μιχάλη Α. Μανάκο και Ολυμπία Χ. Μπάμπη-Μανάκου για την αμέριστη συμπαράστασή τους καθ' όλη τη διάρκεια των σπουδών μου και για όλα όσα μου έχουν προσφέρει αυτά τα χρόνια.

Πινακας Περιεχομενων

Κατάλογος Σχημάτων **viii**

Κατάλογος Πινάκων **ix**

Περίληψη **x**

Abstract **xii**

1 Εισαγωγή **1**

1.1 Καθορισμός Προβλήματος 2

1.2 Στόχος Διπλωματικής 3

1.3 Διάρθρωση Διπλωματικής 4

2 Βοηθητικό Υπόβαθρο **6**

2.1 Προβλήματα Σχεδίασης 7

2.2 Αναδομήσεις 8

2.3 Αυτόματος Εντοπισμός Προβλημάτων Σχεδίασης 9

2.4 Αποθετήρια Λογισμικού 11

3 Βιβλιογραφική Ανασκόπηση **13**

3.1 Μεθοδολογίες Εντοπισμού Αναδομήσεων 14

3.2 Εμπειρικές Μελέτες Βάση Ιστορικών Δεδομένων 15

3.3 Εργαλεία Εντοπισμού Προβλημάτων Σχεδίασης 18

4 Προβλήματα Σχεδίασης & Έργα Λογισμικού **20**

4.1 Προβλήματα Σχεδίασης 21

4.2 Έργα Λογισμικού 32

5 Εμπειρική Μελέτη **35**

5.1 Συνολικός Αριθμός Προβλημάτων Σχεδίασης 36

5.2 Παραμονή Προβλημάτων Σχεδίασης 37

5.3 Εξέλιξη Προβλημάτων Σχεδίασης 40

5.4 Ενεργά Προβλήματα Σχεδίασης 49

6 Συμπεράσματα & Μελλοντική Επέκταση **55**

6.1 Συμπεράσματα Αποτελεσμάτων 56

6.2 Απειλές Εγκυρότητας Αποτελεσμάτων 57

6.3 Μελλοντική Επέκταση Εργασίας 57

Αναφορές **58**

Κατάλογος Σχημάτων

2.1 Παράδειγμα Ακρίβειας & Ανάκλησης 11

4.1 Παράδειγμα Προβλήματος Σχεδίασης "Long Method" 23
4.2 Παράδειγμα Εφαρμογής Αναδόμησης *Extract Method* 23
4.3 Παράδειγμα Προβλήματος Σχεδίασης "Feature Envy" 25
4.4 Παράδειγμα Εφαρμογής Αναδόμησης *Move Method* 26
4.5 Παράδειγμα Προβλήματος Σχεδίασης "State Checking" 28
4.6 Παράδειγμα Εφαρμογής Αναδόμησης *Replace Type Code with State/Strategy* 29
4.7 Παράδειγμα Προβλήματος Σχεδίασης "God Class" 31
4.8 Παράδειγμα Εφαρμογής Αναδόμησης *Extract Class* 32

5.1 Συνολικός Αριθμός Προβλημάτων Σχεδίασης για το Έργο JFlex 36
5.2 Συνολικός Αριθμός Προβλημάτων Σχεδίασης για το έργο JFreeChart . . . 37
5.3 Παραμονή Προβλημάτων Σχεδίασης "Long Method" για το έργο JFlex . . 38
5.4 Εξέλιξη Προβλημάτων Σχεδίασης "Long Method" για το έργο JFlex . . . 44
5.5 Εξέλιξη Προβλημάτων Σχεδίασης "Long Method" για το έργο JFreeChart . 46
5.6 Ενεργά Προβλημάτων Σχεδίασης "State Checking" για το έργο JFreeChart . 53

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

4.1 Χαρακτηριστικά Μεγέθους Εξεταζόμενου Πακέτου για το έργο JFlex . . . 33
4.2 Χαρακτηριστικά Μεγέθους Εξεταζόμενου Πακέτου για το έργο JFreeChart 34

5.1 Εντοπιζόμενες Περιπτώσεις Προβλημάτων Σχεδίασης 47
5.2 Μέσος Χρόνος Παραμονής Προβλημάτων Σχεδίασης 48
5.3 Σκόπιμες Ενέργειες Επίλυσης Εντοπιζόμενων Προβλημάτων Σχεδίασης . . 49
5.4 Ενεργά Προβλήματα Σχεδίασης 54

ΔΙΕΡΕΥΝΗΣΗ ΙΣΤΟΡΙΚΩΝ ΔΕΔΟΜΕΝΩΝ ΑΝΑΦΟΡΙΚΑ ΜΕ ΠΡΟΒΛΗΜΑΤΑ ΣΧΕΔΙΑΣΗΣ ΣΕ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

Αναστάσιος Μ. Μανάκος, MSc

Τμήμα Εφαρμοσμένης Πληροφορικής

Πανεπιστήμιο Μακεδονίας

Ιούνιος, 2010

Αλέξανδρος Ν. Χατζηγεωργίου, Επιβλέπων

Περίληψη

Τα προβλήματα σχεδίασης, σε αντίθεση με τα πρότυπα, αποτελούν συμπτώματα κακής ποιότητας της σχεδίασης ενός συστήματος λογισμικού και στην Τεχνολογία Λογισμικού εμφανίζονται υπό διαφορετικά ονόματα αλλά και οπτική γωνία θεώρησης. Ο εντοπισμός και εξάλειψη τους, δηλαδή η εφαρμογή της ευρέως γνωστής διαδικασίας της αναδόμησης, είναι μείζονος σημασίας εφόσον οδηγεί σε πηγαίο κώδικα του συστήματος που είναι πιο εύκολος στην κατανόηση, την τροποποίηση, την επαναχρησιμοποίηση και τον έλεγχο.

Η παρούσα διπλωματική εργασία διερευνά την εξέλιξη των προβλημάτων σχεδίασης, αξιοποιώντας εργαλεία-CASE tools που υποστηρίζουν τον αυτόματο εντοπισμό τους, σε πολλαπλές διαδοχικές γενιές επιλεγμένων έργων λογισμικού ανοιχτού κώδικα, υλοποιημένα στη γλώσσα προγραμματισμού Java. Η αξιολόγηση των ευρημάτων, αναμένεται να παράσχει σημαντικές πληροφορίες και να απαντήσει σε αρκετά ερωτήματα όπως για παράδειγμα: α) εάν ο αριθμός των προβλημάτων σχεδίασης αυξάνεται με τη πάροδο των γενεών ενός έργου β) εάν τα προβλήματα σχεδίασης εξαλείφονται λόγω κατάλληλων στοχευόμενων ανθρώπινων παρεμβάσεων

κάθε φορά γ) εάν τα προβλήματα σχεδίασης εμφανίζονται κατά την εξέλιξη των σχετικών τμημάτων του πηγαίου κώδικα ή ενυπάρχουν ήδη από τη στιγμή της εισαγωγής τους στο σύστημα, και δ) κατά πόσο συχνά εκτελούνται οι κατάλληλες ενέργειες αναδόμησης από τους σχεδιαστές των υπό εξέταση συστημάτων λογισμικού ανοιχτού κώδικα για την εξάλειψη των εντοπιζόμενων προβλημάτων σχεδίασης.

Σε αντίθεση με προηγούμενες ερευνητικές προσεγγίσεις που έχουν διερευνήσει την εφαρμογή των αναδομήσεων κατά τη διάρκεια εξέλιξης ενός έργου λογισμικού, στα πλαίσια της παρούσας διπλωματικής εργασίας, θα επιχειρηθεί η εξέταση του θέματος από την οπτική γωνία των ίδιων των προβλημάτων σχεδίασης διαχωρίζοντας τις σκόπιμες ενέργειες αναδόμησης από τις ακούσιες ενέργειες αφαίρεσης των προβλημάτων που προκύπτουν λόγω της διορθωτικής ή της προσαρμοστικής συντήρησης του συστήματος λογισμικού.

Η εμπειρική μελέτη πραγματοποιείται εξετάζοντας δυο έργα λογισμικού ανοιχτού κώδικα αναζητώντας τέσσερα προβλήματα σχεδίασης που θεωρούνται τα περισσότερο σημαντικά μεταξύ των προβλημάτων που εμφανίζονται σε έργα λογισμικού μεγάλης κλίμακας μιας και σχετίζονται με τη διανομή της λειτουργικότητας μεταξύ των κλάσεων και των μεθόδων του συστήματος. Επιπλέον, η εξάλειψή αυτών επιτυγχάνεται κατόπιν στοχευόμενων ενεργειών από πλευράς των σχεδιαστών.

Λέξεις Κλειδιά: Προβλήματα Σχεδίασης, Αναδομήσεις, Αποθετήρια Λογισμικού, Εξέλιξη Λογισμικού.

INVESTIGATING THE HISTORICAL DATA OF CODE SMELLS IN OBJECT-ORIENTED SYSTEMS

Anastasios M. Manakos, MSc

Department of Applied Informatics

University of Macedonia

June, 2010

Alexander N. Chatzigeorgiou, Advisor

Abstract

In contrast with patterns, design problems are poor implementation choices and indicate symptoms of bad design of software systems. In Software Engineering they are known and perceived under many different terms. Detecting and removing design problems, in other words, the performance of well known refactoring activity is of vital importance since it allows to simplify the source code and increase understandability, maintainability, extensibility and verifiability.

This thesis, taking advantage of recent state-of-the-art Computer-Aided Software Engineering (CASE) tools that offer support for the identification of design problems, focuses on the presence and evolution of code smells, by analyzing past versions of software projects written in Java. Several interesting questions can be investigated such as, a) whether the number of design problems increases with the passage of software versions, b) whether design problems vanish only by targeted human intervention c) whether code smells exist in a software module right from its initial construction or do they appear during its evolution and d) whether how

frequent are refactoring activities that target identified code smells which performs the designers of the examined software systems.

In contrast to previous studies that investigate the application of refactorings in the history of a software project, in this thesis, it will be attempted to study the subject from the point of view of the problems themselves. In this way, it will be possible to distinguish targeted refactoring activities from the removal of design problems as a side effect of corrective or adaptive maintenance.

This case study investigates four code smells, by analyzing the historical data over several successive versions of two large scale open-source systems. The main reason for choosing this code smells is that they are among the most serious ones for the design of object-oriented systems since they are related to the distribution of functionality among the classes and methods of a system. Additionally, removal of these smells requires systematic and elaborate refactoring actions.

Keywords: Design Problems, Refactorings, Software Repositories, Software Evolution.

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Περιεχόμενα

1.1 Καθορισμός Προβλήματος 2

1.2 Στόχος Διπλωματικής 3

1.3 Διάρθρωση Διπλωματικής 4

"empirical, adj:
based on, concerned with, or verifiable by observation or experience
rather than theory or pure logic!"

[Chief editor: Cowie
Oxford Advanced Learner's Dictionary, 1989]

1.1 Καθορισμός Προβλήματος

Σε αντίθεση με ένα λογοτεχνικό έργο που γράφεται μια φορά και παραμένει αναλλοίωτο, το λογισμικό επεκτείνεται, τροποποιείται, μεταφέρεται σε άλλη υπολογιστική πλατφόρμα και ούτω καθεξής καθ' όλη τη διάρκεια ζωής του.

Το κόστος της σχεδίασης ενός συστήματος λογισμικού είναι σημαντικό και εξαρτάται κυρίως από το μέγεθός του 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. Συνεπώς, το πλέον εμφανές σύμπτωμα που πιθανόν να παρουσιάσει η σχεδίαση στο κύκλο ζωής ενός προϊόντος λογισμικού είναι η δυσκολία της συντήρησης, δηλαδή της ικανοποίησης των νέων απαιτήσεων των πελατών.

Για την αποφυγή λοιπόν του εκφυλισμού της ποιότητας της σχεδίασης, απαιτείται συστηματική σχεδίαση και ανάλυση σε όλες τις φάσεις ανάπτυξης του συστήματος λογισμικού [37]. Ο εντοπισμός και εξάλειψη των προβλημάτων σχεδίασης αποτελεί μείζονος σημασία και ένδειξη αυτού αποτελεί το γεγονός ότι στην Τεχνολογία Λογισμικού τα προβλήματα σχεδίασης εμφανίζονται υπό διαφορετικά ονόματα αλλά και οπτική γωνία θεώρησης. Για παράδειγμα, στα προβλήματα αυτά, οι ερευνητές αναφέρονται ως κακές οσμές-bad smells [13], ελαττώματα σχεδίασης-design flaws [30], έλλειψη συμμόρφωσης με αρχές σχεδίασης-design principles [31], παραβίαση ευρετικών κανόνων-design heuristics [40], απουσία προτύπων σχεδίασης-design patterns [14] καθώς επίσης και εφαρμογή αντι-προτύπων σχεδίασης-anti patterns [5].

Για την εξάλειψη των προβλημάτων και συνάμα τη βελτίωση της ποιότητας της σχεδίασης ενός συστήματος λογισμικού, διατηρώντας αναλλοίωτη την εξωτερική του συμπεριφορά, εφαρμόζεται η διαδικασία, της κατάλληλης κάθε φορά, αναδόμησης-refactoring [36]. Οι αναδομήσεις έγιναν ευρέως αποδεκτές κυρίως γιατί η εφαρμογής τους είναι βέβαιο ότι οδηγεί σε βελτίωση της ποιότητας της σχεδίασης, γεγονός που επιβεβαιώνεται τόσο μέσα από θεωρητικές [10] όσο και εμπειρικές μελέτες [4], [23], [42]. Επίσης η εφαρμογή τους καθορίζεται από συγκεκριμένα βήματα εκτέλεσης επιτρέποντας την αυτοματοποίηση της διαδικασίας.

Η αξιολόγηση της ποιότητας της σχεδίασης ενός έργου λογισμικού συνήθως πραγματοποιείται στη τρέχουσα διαθέσιμη έκδοσή του. Ωστόσο, η ύπαρξη οργανωμένων αποθετηρίων λογισμικού και κυρίως πηγαίου κώδικα αποτελεί μια επιπρόσθετη πηγή πληροφοριών για την άντληση στοιχείων που αφορούν την εξέλιξη έργων λογισμικού καθώς παρέχουν πρόσβαση σε παλαιότερες γενιές του πηγαίου κώδικα [22].

Αναγνωρίζοντας τη σημασία των ιστορικών δεδομένων κατά τη διαδικασία της συντήρησης, αναπτύχθηκαν αρκετές αξιόπιστες μεθοδολογίες εστιάζοντας στον εντοπισμό αναδομήσεων που πραγματοποιήθηκαν σε παλαιότερες γενιές έργων λογισμικού καθώς επίσης και εμπειρικές μελέτες με απώτερο στόχο τη μελέτη των συνηθειών που υιοθετούνται από συντηρητές και προγραμματιστές λογισμικού αναφορικά με τη πρακτική της αναδόμησης.

Στο πλαίσιο αυτό κρίνεται εξαιρετικά ενδιαφέρουσα η διερεύνηση ιστορικών δεδομένων πηγαίου κώδικα από αποθετήρια λογισμικού, αναφορικά με τα προβλήματα σχεδίασης που εμφανίστηκαν σε παρελθούσες γενιές ενός συστήματος λογισμικού και η συσχέτισή τους με την εξέλιξη άλλων χαρακτηριστικών της σχεδίασης.

1.2 Στόχος Διπλωματικής

Η παρούσα διπλωματική εργασία διερευνά την εξέλιξη των προβλημάτων σχεδίασης, αξιοποιώντας εργαλεία-CASE tools, που υποστηρίζουν τον αυτόματο εντοπισμό τους (και αποτελούν φυσικά ευκαιρίες για αναδόμηση), σε πολλαπλές διαδοχικές γενιές επιλεγμένων έργων λογισμικού ανοιχτού κώδικα, υλοποιημένα σε γλώσσα προγραμματισμού Java.

Σε αντίθεση με προηγούμενες ερευνητικές προσεγγίσεις που έχουν διερεύνηση την εφαρμογή αναδομήσεων κατά τη διάρκεια εξέλιξης ενός έργου λογισμικού, στα πλαίσια της παρούσας διπλωματικής εργασίας, θα επιχειρηθεί η εξέταση του θέματος από την οπτική γωνία των ίδιων των προβλημάτων σχεδίασης. Η αξιολόγηση των ευρημάτων αναμένεται να παράσχει σημαντικές πληροφορίες και να απαντήσει στα ερωτήματα που έπονται:

- *Ο αριθμός των προβλημάτων σχεδίασης αυξάνεται με τη πάροδο των γενεών του έργου;*

- *Η πλειονότητα των προβλημάτων σχεδίασης παραμένει στο λογισμικό καθ' όλη τη διάρκεια ζωής του;*

- *Ποιος τύπος προβλημάτων έχει το μεγαλύτερο ποσοστό εμφάνισης;*

- *Τα προβλήματα σχεδίασης εμφανίζονται κατά την εξέλιξη των σχετικών τμημάτων του πηγαίου κώδικα ή ενυπάρχουν ήδη από τη στιγμή της εισαγωγής των αντίστοιχων τμημάτων στο σύστημα;*

- *Τα προβλήματα σχεδίασης εξαλείφονται λόγω κατάλληλων στοχευόμενων ανθρώπινων παρεμβάσεων κάθε φορά ή είναι παράπλευρη συνέπεια ενεργειών συντήρησης;*

- *Πόσο συχνά εκτελούνται οι κατάλληλες ενέργειες αναδόμησης για την εξάλειψη των εντοπιζόμενων προβλημάτων σχεδίασης από τους σχεδιαστές των υπό εξέταση συστημάτων λογισμικού ανοιχτού κώδικα;*

- *Πόσο επιτακτική κρίνεται η ανάγκη αναδόμησης των εντοπιζόμενων προβλημάτων σχεδίασης;*

Σημειώνεται ότι τα ευρήματα της μελέτης θα αφορούν σε πρώτο επίπεδο τα υπό εξέταση έργα λογισμικού εφόσον θα χαρακτηρίζουν το είδος της ποιότητας σχεδίασής τους. Επίσης, θα παρέχουν σημαντικές ενδείξεις που θα σχετίζονται με τη πρακτική της αναδόμησης (εντοπισμό και επίλυση των προβλημάτων σχεδίασης) που υιοθετούν οι ομάδες ανάπτυξης των αντίστοιχων έργων σύμφωνα με τα ιστορικά δεδομένα του πηγαίου τους κώδικα. Στο πλαίσιο αυτό, τα ευρήματα θα εξάγουν πληροφορίες σχετιζόμενες με τις γνώσεις, τις επιδεξιότητες αλλά και τις στάσεις των ομάδων ανάπτυξης προς τη διαδικασία της αναδόμησης. Ωστόσο, απαιτείται περισσότερη διερεύνηση για την έγκυρη γενίκευση των ευρημάτων που θα προκύψουν.

1.3 Διάρθρωση Διπλωματικής

Η διπλωματική εργασία αποτελείται από έξι κεφάλαια τα οποία δομούνται ως εξής:

Το Κεφάλαιο 1 (παρόν κεφάλαιο) αποτελεί την εισαγωγή της διπλωματικής εργασίας.

Στο Κεφάλαιο 2 παρατίθεται το απαραίτητο βοηθητικό υπόβαθρο στον αναγνώστη για την κατανόηση μερικών βασικών εννοιών λόγω της θεμελιώδους σημασίας τους.

Στο Κεφάλαιο 3 πραγματοποιείται ενδεικτική επισκόπηση στις μεθοδολογίες που επιτρέπουν τον εντοπισμό αναδομήσεων σε διαδοχικές γενιές έργων λογισμικού καθώς επίσης και μερικές εμπειρικές μελέτες που σχετίζονται με τη πρακτική της αναδόμησης. Το κεφάλαιο ολοκληρώνεται με τη παράθεση μερικών εμπειρικών μελετών που διερευνούν την εξέλιξη των προβλημάτων σχεδίασης και την επίπτωσή τους στη συμπεριφορά των συστημάτων και τη παρουσίαση εργαλείων που επιτρέπουν τον αυτόματο εντοπισμό των προβλημάτων σχεδίασης.

Στο Κεφάλαιο 4 περιγράφονται τα προβλήματα σχεδίασης και τα έργα λογισμικού ανοιχτού κώδικα που επιλέχθηκαν προς αναζήτηση των πιθανών εξεταζόμενων προβλημάτων σε αρκετές παρελθούσες τους γενιές. Επίσης, αναφέρονται και οι λόγοι που επιλέχθηκαν τα συγκεκριμένα προβλήματα σχεδίασης και τα έργα λογισμικού.

Στο Κεφάλαιο 5 παρουσιάζονται τα αποτελέσματα της εμπειρική μελέτης σχετικά με την εξέλιξη των προβλημάτων σχεδίασης.

Τέλος, στο Κεφάλαιο 6 αναφέρονται τα συμπεράσματα που προέκυψαν με βάση τα αποτελέσματα της εμπειρικής μελέτης καθώς επίσης και οι απειλές εγκυρότητας των αποτελεσμάτων που πιθανό να δεχθεί η παρούσα διπλωματική εργασία. Το κεφάλαιο ολοκληρώνεται με τις προτεινόμενες προτάσεις για περαιτέρω διερεύνηση του συγκεκριμένου θέματος.

ΚΕΦΑΛΑΙΟ 2

ΒΟΗΘΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Περιεχόμενα

2.1 Προβλήματα Σχεδίασης 7

2.2 Αναδομήσεις 8

2.3 Αυτόματος Εντοπισμός Προβλημάτων Σχεδίασης 9

2.4 Αποθετήρια Πηγαίου Κώδικα 11

"A word of warning and encouragement:
Don't worry if you don't understand this book completely on the first reading.
We didn't understand it all on the first writing!"

[Gamma, Helm, Johnson & Vlissides:
Design Patterns: Elements of Reusable Object-Oriented Software, 1995]

2.1 Προβλήματα Σχεδίασης

Το λογισμικό, σε αντίθεση με ένα λογοτεχνικό έργο που γράφεται μια φορά και παραμένει αναλλοίωτο, επεκτείνεται, τροποποιείται, μεταφέρεται σε άλλη υπολογιστική πλατφόρμα και ούτω καθεξής καθ' όλη τη διάρκεια ζωής του. Η διαδικασία που υφίσταται το λογισμικό πραγματοποιείται από διαφορετικούς ανθρώπους και σε χρονικό διάστημα ετών, συχνά και δεκαετιών. Σχεδόν όλοι οι προγραμματιστές γνωρίζουν ότι η κατανόηση του πηγαίου κώδικα που έχει γραφεί από άλλους ή ακόμα και από τους ίδιους πριν από κάποιο χρονικό διάστημα είναι μια από τις δυσκολότερες εργασίες.

Συνεπώς, στο 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, γεγονός που επιβεβαιώνεται μέσα από σχετικές έρευνες [17], [11], [12], 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 θα γίνει και η εξέλιξη του λογισμικού.

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 τη ποιότητα αναφορικά με την εύκολη συντήρηση.

Ωστόσο, η σχεδίαση των αντικειμενοστρεφών συστημάτων λογισμικού είναι δυνατόν να παρουσιάσει συμπτώματα κακής ποιότητας της σχεδίασης-poorly/bad design που στη Τεχνολογία Λογισμικού εμφανίζονται υπό διαφορετικά ονόματα αλλά και οπτική γωνία θεώρησης. Για παράδειγμα, στα προβλήματα αυτά, οι ερευνητές αναφέρονται ως κακές οσμές-bad smells [13], ελαττώματα σχεδίασης-design flaws [30], έλλειψη συμμόρφωσης με αρχές σχεδίασης-design principles [31], παραβίαση

ευρετικών κανόνων-design heuristics [40], απουσία προτύπων σχεδίασης-design patterns [14], καθώς επίσης και εφαρμογή αντι-προτύπων σχεδίασης-anti patterns [5]. Ο εντοπισμός και η εξάλειψη των προβλημάτων αποτελεί μείζονος σημασία εφόσον οδηγεί σε πηγαίο κώδικα του συστήματος που είναι πιο εύκολος στη κατανόηση, τη 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.

2.2 Αναδομήσεις

Για την εξάλειψη των προβλημάτων σχεδίασης και συνάμα τη βελτίωση της ποιότητας της σχεδίασης ενός έργου λογισμικού, εφαρμόζεται η διαδικασία, της κατάλληλης κάθε φορά, αναδόμησης-refactoring.

Αναδόμηση είναι η διαδικασία της τροποποίησης του πηγαίου κώδικα ενός συστήματος λογισμικού με τέτοιο τρόπο ώστε να εξασφαλίζεται η εξωτερική του συμπεριφορά και ταυτόχρονα να βελτιώνεται η εσωτερική του σχεδίαση. Δηλαδή, η διαδικασία εγγυάται, μετά την εφαρμογή της, τη διατήρηση της λειτουργικότητας του συστήματος-behavior preserving changes [36].

Θα πρέπει να σημειωθεί ότι, όλες οι αλλαγές που επιφέρει η διαδικασία της αναδόμησης του πηγαίου κώδικα ενός συστήματος λογισμικού, έχουν στόχο α) την απλοποίηση του ώστε να είναι εύκολα κατανοητός, στοιχείο ιδιαιτέρως σημαντικό όταν το λογισμικό δεν πρόκειται να αναπτύσσεται πάντοτε από τα ίδια άτομα, β) εύκολα και αποδοτικά συντηρήσιμος, δηλαδή με μικρή προσπάθεια και κόστος χωρίς να υποβαθμίζεται η ποιότητά του και γ) εύκολα ελεγχόμενος, είτε κατά τη διάρκεια της υλοποίησής του είτε μετά το πέρας αυτής [36].

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: αλλαγές που αφορούν μονάχα τροποποιήσεις τμημάτων του πηγαίου κώδικα και περιλαμβάνουν τις αναδομήσεις *Extract Local Variable*, *Rename Local Variable* και *Add Assertion* [34].

Οι αναδομήσεις, έγιναν ευρέως αποδεκτές από την κοινότητα της Τεχνολογίας Λογισμικού αμέσως μετά τη πρώτη συστηματική τους καταγραφή [13][1] εφόσον η εφαρμογής τους είναι βέβαιο ότι οδηγεί σε βελτίωση της ποιότητας της σχεδίασης ενός συστήματος λογισμικού γεγονός που επιβεβαιώνεται τόσο μέσα από θεωρητικές μελέτες [10] όσο και εμπειρικές [4], [23], [42] που δείχνουν ότι οι αναδομήσεις επιδρούν θετικά πάνω σε μετρικές ποιότητας αντικειμενοστρεφούς λογισμικού.

2.3 Αυτόματος Εντοπισμός Προβλημάτων Σχεδίασης

Στο χώρο της Αντικειμενοστρεφούς Τεχνολογίας Λογισμικού, τα τελευταία χρόνια έχουν αναπτυχθεί εργαλεία, είτε ως αυτόνομα προγράμματα είτε ως ενσωματωμένες λειτουργίες σε ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού, για την ευκολότερη κατανόηση και συντήρηση συστημάτων λογισμικού. Συγκεκριμένα, τα εργαλεία αυτά παρέχουν σημαντική υποστήριξη στους σχεδιαστές και κυρίως σε εκείνους που πραγματοποιούν συντήρηση λογισμικού καθώς επιτρέπουν τον αυτόματο εντοπισμό των προβλημάτων σχεδίασης και την πραγματοποίηση συστάσεων για την εφαρμογή της κατάλληλης ενδεδειγμένης τεχνικής επίλυσής τους.

[1] Σημειώνεται επίσης ότι, η πρώτη συστηματική λίστα καταγραφής των αναδομήσεων, σύντομα επεκτάθηκε [24].

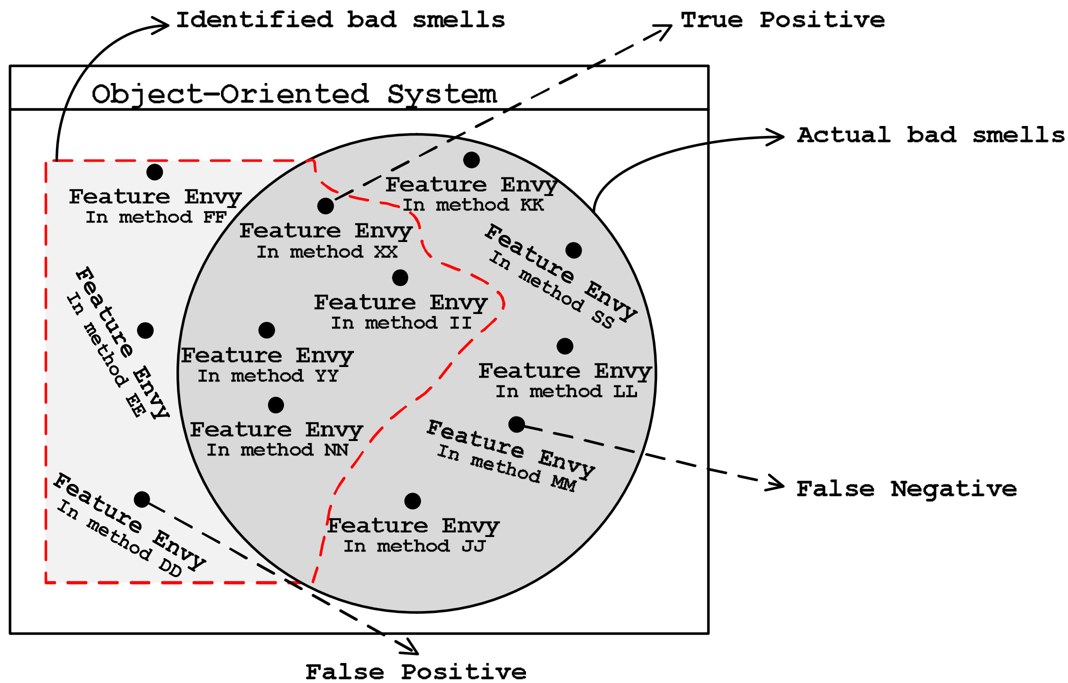
Ο εντοπισμός των προβλημάτων σχεδίασης σε έργα λογισμικού μεγάλης κλίμακας με το χέρι εκτός του ότι είναι αρκετά δύσκολη και χρονοβόρα διαδικασία [28], οδηγεί πολλές φορές σε αντιπαραθέσεις μεταξύ των σχεδιαστών οι οποίες και σχετίζονται με την επιλογή των αναδομήσεων που θα εφαρμοστούν εφόσον κάθε σχεδιαστής βλέπει διαφορετικά τα προβλήματα σχεδίασης ανάλογα με την εμπειρία που έχει ως σχεδιαστής αλλά και με τη διάρκεια εργασίας του σε ένα συγκεκριμένο έργο λογισμικού [27]. Συνεπώς, η ύπαρξη εργαλείων αυτόματου εντοπισμού των προβλημάτων σχεδίασης κρίνεται απαραίτητη καθώς μειώνουν το χρόνο αναζήτησης των προβλημάτων και κατά συνέπεια αυξάνουν σε σημαντικό βαθμό τη παραγωγικότητα των σχεδιαστών.

Ωστόσο, απαιτείται ακόμα μεγάλη προσπάθεια ώστε οι μεθοδολογίες που βασίζονται τα εργαλεία για τον εντοπισμό των προβλημάτων σχεδίασης να γίνουν ευρέως αποδεκτές από την κοινότητα της Αντικειμενοστρεφούς Τεχνολογίας Λογισμικού από τη στιγμή που η εκτίμηση των αποτελεσμάτων τους είναι ανεπαρκή. Μερικές από τις μεθοδολογίες έχουν έλλειψη αξιολόγησης στο σύνολό τους, άλλες στηρίζονται στην απόφαση κάποιου ειδικού-expert, ενώ άλλες έχουν υποκειμενική αξιολόγηση.

Δυο από τις περισσότερο ευρέως χρησιμοποιούμενες τεχνικές εκτίμησης της απόδοσης μιας μεθοδολογίας, δηλαδή της εξαγωγής συμπερασμάτων από την αξιολόγηση των αποτελεσμάτων της, είναι η έννοια της Ακρίβειας-Precision (P) και η έννοια της Ανάκλησης-Recall (R) [26].

Ο αυτόματος εντοπισμός των προβλημάτων σχεδίασης, σχετίζεται με τις δυο αναφερόμενες τεχνικές από την στιγμή που το εργαλείο, που υποστηρίζει την όλη διαδικασία του αυτόματου εντοπισμού των προβλημάτων, είναι πιθανό να εντοπίσει υποψήφια προβλήματα σχεδίασης, μερικά από τα οποία είναι πραγματικά προβλήματα και για τα οποία θα συμφωνούσε και κάποιος ειδικός σχεδίασης συστημάτων λογισμικού-true positive ενώ άλλα τα οποία εντοπίστηκαν αλλά δεν είναι πραγματικά προβλήματα σχεδίασης και για τα οποία ούτε και ο ειδικός θα συμφωνούσε και θα απέρριπτε τη πρόταση του εργαλείου-false positive. Φυσικά, είναι πολύ πιθανό, να υπάρχουν πραγματικά προβλήματα σχεδίασης τα οποία δεν θα είναι εφικτό να εντοπιστούν από το εργαλείο λόγω διαφορετικής προσέγγισης της μεθοδολογίας εντοπισμού-false negative αλλά και περιπτώσεις που το εργαλείο δεν

θα εντοπίσει κανένα πρόβλημα σχεδίασης μιας και δεν θα υπάρχουν πραγματικά πρόβλημα-true negative. Η ανωτέρω περιγραφή αναπαρίσταται στο Σχήμα 2.1 για προβλήματα σχεδίασης τύπου "Feature Envy" σε ένα υποθετικό σύστημα λογισμικού.



Σχήμα 2.1: Παράδειγμα Ακρίβειας και Ανάκλησης στα πλαίσια εντοπισμού προβλημάτων σχεδίασης

Συνεπώς, με βάση τα όσα αναφέρθηκαν, η εκτίμηση των αποτελεσμάτων μιας μεθοδολογίας εντοπισμού προβλημάτων σχεδίασης, είναι εφικτό να ποσοτικοποιηθεί με βάση τις δυο τεχνικές ως εξής:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad \text{και} \quad Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

2.4 Αποθετήρια Πηγαίου Κώδικα

Η ποιότητα σχεδίασης ενός συστήματος λογισμικού συνήθως αξιολογείται βάση της τρέχουσας διαθέσιμης γενιάς του έργου ωστόσο οργανωμένες συλλογές αποθετηρίων λογισμικού και κυρίως πηγαίου κώδικα-source code repositories, αποτελούν πολύτιμη πηγή πληροφοριών για την άντληση στοιχείων που αφορούν την εξέλιξη έργων λογισμικού καθώς παρέχουν πρόσβαση σε παλαιότερες γενιές του πηγαίου κώδικα. Σχεδόν όλα τα μοντέρνα έργα λογισμικού διατηρούν όλες τις

αλλαγές που υπέστη ο πηγαίος τους κώδικας, ξεχωριστά για κάθε γενιά του έργου, σε αυτά τα ειδικά αποθετήρια. Αν ένα κομμάτι του κώδικα έχει αναδομηθεί, ένδειξη αυτού, θα είναι αποθηκευμένη στο αποθετήριο. Επεξήγηση του πηγαίου κώδικα που περιγράφει πώς να χρησιμοποιείται το λογισμικό πριν και μετά την διαδικασία της αναδόμησης, επίσης είναι αποθηκευμένη στο αποθετήριο. Όλες οι αλλαγές που πραγματοποιούνται για τη διόρθωση των σφαλμάτων που παρουσιάζει το λογισμικό καταγράφονται και αυτές στο αποθετήριο. Συνεπώς, οι οργανωμένες συλλογές αποθετηρίων λογισμικού παρέχουν μια επιπρόσθετη πηγή πληροφοριών αναφορικά με την ποιότητα του πηγαίου κώδικα καθώς καταγράφουν την εξέλιξη της σχεδίασης και κατά συνέπεια μπορούν να είναι χρήσιμα στην αξιολόγηση της συντηρησημότητας του λογισμικού.

Η "Εξόρυξη Γνώσης από Αποθετήρια Λογισμικού-Mining Software Repositories" είναι ένα ολοκληρωμένο πεδίο έρευνας που εστιάζει στην αξιοποίηση ιστορικών δεδομένων έργων λογισμικού για την υποστήριξη της συντήρησης, της βελτίωσης της σχεδίασης και της επαναχρησιμοποίησης των συστημάτων λογισμικού αλλά και της εμπειρικής επιβεβαίωσης καινοτόμων ιδεών και τεχνικών [22].

Σημειώνεται επίσης ότι, αναγνωρίζοντας τη σημασία των ιστορικών δεδομένων κατά τη διαδικασία της συντήρησης, διεξήχθησαν αρκετές εμπειρικές μελέτες με απώτερο στόχο τη μελέτη των συνηθειών που υιοθετούνται από συντηρητές και προγραμματιστές λογισμικού αναφορικά με τη πρακτική της αναδόμησης.

ΚΕΦΑΛΑΙΟ 3

ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΑΝΑΣΚΟΠΗΣΗ

Περιεχόμενα

3.1 Μεθοδολογίες Εντοπισμού Αναδομήσεων 14

3.2 Εμπειρικές Μελέτες Βάση Ιστορικών Δεδομένων 15

3.3 Εργαλεία Εντοπισμού Προβλημάτων Σχεδίασης 18

"No amount of experimentation can ever prove me right,
A single experiment, can prove me
wrong!"

[Albert Einstein:
Quotes]

3.1 Μεθοδολογίες Εντοπισμού Αναδομήσεων

Αρκετές μελέτες έχουν εστιάσει στον εντοπισμό αναδομήσεων που έχουν εφαρμοστεί σε παλαιότερες γενιές έργων λογισμικού αναγνωρίζοντας τη σημασία των ιστορικών δεδομένων κατά τη διαδικασία της συντήρησης. Στη συνέχεια παρατίθενται ενδεικτικά μερικές από τις σημαντικές εργασίες που διεκπεραιώθηκαν:

Μια μεθοδολογία βασίζεται στον εντοπισμό αναδομήσεων κάνοντας χρήση μετρικών ποιότητας αντικειμενοστρεφούς λογισμικού. Οι τιμές των μετρικών σχετίζονται με το μέγεθος των μεθόδους, των κλάσεων και το μέγεθος της κληρονομικότητας ενός συστήματος λογισμικού. Σύμφωνα με την εμπειρία των ερευνητών που ανέπτυξαν τη συγκεκριμένη μεθοδολογία, οι τρεις αναφερόμενοι τύποι αναδομήσεων υλοποιούνται περισσότερο κατά την εξέλιξη των αντικειμενοστρεφών συστημάτων. Για τον εντοπισμό των αναδομήσεων που εφαρμόστηκαν ανάμεσα σε δυο διαδοχικές γενιές ενός συστήματος λογισμικού, καθορίστηκαν τέσσερεις ευρετικοί κανόνες σε συνδυασμό με τις αλλαγές των τιμών των μετρικών. Η εκτίμηση των αποτελεσμάτων της μεθοδολογίας πραγματοποιήθηκε σε τρία έργα λογισμικού που περιέχουν διαθέσιμα ελεύθερες αρκετές γενιές στα αποθετήρια πηγαίου τους κώδικα και ένα από τα βασικά συμπεράσματα συγκαταλέγεται στο γεγονός ότι οι ενέργειες αναδόμησης υλοποιούνται σε συγκεκριμένα τμήματα του πηγαίου τους κώδικα τα οποία και υφίστανται εμπλουτισμό της λειτουργικότητάς τους [7].

Μια άλλη τεχνική εντοπισμού, χρησιμοποιεί έναν αλγόριθμος ο οποίος συνδυάζει μια γρήγορη συντακτική ανάλυση-syntactic analysis για τον εντοπισμό των υποψήφιων αναδομήσεων καθώς επίσης και μια περισσότερο επακριβή σημασιολογική ανάλυση-semantic analysis για τη βελτίωση των αποτελεσμάτων που προέκυψαν από τη πρώτη συντακτική ανάλυση. Συγκεκριμένα, η πρώτη ανάλυση του αλγορίθμου, βασίζεται στη κωδικοποίηση Shingles και χρησιμοποιείται για τον εντοπισμό όμοιων ζευγαριών οντοτήτων (μεθόδων, κλάσεων, και πακέτων), σε δυο γενιές ενός συστατικού που είναι υποψήφιες για αναδόμηση. Συνεπώς, το αποτέλεσμα της πρώτης ανάλυσης είναι ένα σύνολο ζευγαριών από οντότητες με κωδικοποίηση Shingles για δυο γενιές ενός συστατικού. Κάθε ζευγάρι του συνόλου αποτελείται από δυο οντότητες, μια για τη πρώτη γενιά του συστατικού και μια για τη δεύτερη. Σημειώνεται, ότι υπάρχουν ξεχωριστά ζευγάρια οντοτήτων για μεθόδους,

κλάσεις και πακέτα. Στη συνέχεια, η σημασιολογική ανάλυση εντοπίζει από τα υποψήφια ζευγάρια εκείνα που η δεύτερη οντότητα πιθανόν να προέκυψε από την αναδόμηση της πρώτης. Ο αλγόριθμος εντοπίζει επτά τύπους αναδομήσεων μεταξύ δυο διαδοχικών γενεών ενός συστατικού και η εκτίμηση των αποτελεσμάτων που επιφέρει πραγματοποιήθηκε σε τρεις πραγματικές εφαρμογές δείχνοντας ότι ο αλγόριθμος έχει ακρίβεια πάνω από 85% [8].

Διαφορετική προσέγγιση στον εντοπισμό των αναδομήσεων αποτελεί η μεθοδολογία επιπέδου σχεδίασης η οποία και χρησιμοποιεί τον αλγόριθμο UMLDiff. Ο αλγόριθμος εντοπίζει αρκετά βασικά είδη δομικών αλλαγών ενός συστήματος λογισμικού, όπως για παράδειγμα τις προσθήκες, τις διαγραφές, τις μετονομασίες διάφορων συστατικών του (πακέτων, κλάσεων, μεθόδων, ιδιοτήτων), κλπ. Η εκτίμηση των αποτελεσμάτων της μεθοδολογίας πραγματοποιήθηκε σε πολλές γενιές δυο αντικειμενοστρεφών συστημάτων λογισμικού και με βάση τα ευρήματα, η μεθοδολογία εντόπισε όλες τις καταγεγραμμένες περιπτώσεις προβλημάτων των σχεδιαστών καθώς επίσης και μερικές επιπλέον μη καταγεγραμμένες περιπτώσεις [48].

3.2 Εμπειρικές Μελέτες Βάση Ιστορικών Δεδομένων

Η ύπαρξη μεθοδολογιών και εργαλείων αυτόματου εντοπισμού των αναδομήσεων που πραγματοποιήθηκαν σε παλαιότερες γενιές έργων λογισμικού, επιτρέπουν τους ερευνητές να διεξάγουν εμπειρικές μελέτες με απώτερο στόχο τη μελέτη των συνηθειών που υιοθετούνται από συντηρητές και προγραμματιστές λογισμικού αναφορικά με τη πρακτική της αναδόμησης. Στη συνέχεια παρουσιάζονται ενδεικτικά μερικές από τις σχετικές έρευνες:

Μια μελέτη περίπτωση που πραγματοποιήθηκε αφορά την εξέλιξη της αρχιτεκτονικής δομής της πλατφόρμας Eclipse με σκοπό τη διερεύνηση του ποσοστού των τροποποιήσεων του πηγαίου κώδικα που αποτελούν αναδομήσεις καθώς επίσης και ποιοι τύποι αναδομήσεων εφαρμόζονται συχνότερα. Με βάση τα ευρήματα της μελέτης, το 70% των δομικών αλλαγών πιθανόν να οφείλεται σε αναδομήσεις και το υψηλό ποσοστό μπορεί να οφείλεται στην ήδη εξελιγμένη ποιότητα σχεδίασης του συστήματος λογισμικού αλλά παραμένει άγνωστο αν οι

εφαρμοσμένες αναδομήσεις στόχευαν στην εξάλειψη συγκεκριμένων προβλημάτων σχεδίασης [49].

Μια λεπτομερής μελέτη εφαρμογής των αναδομήσεων βασίζεται σε τέσσερις συλλογές δεδομένων, συμπεριλαμβάνοντας δεδομένα από χρήστες της πλατφόρμας Eclipse IDE που εθελοντικά επέτρεπαν την αυτόματη υποβολή όλων των εντολών αναδόμησης στη βάση δεδομένων των Eclipse Foundation και δεδομένα πηγαίου κώδικα από αποθετήρια του Eclipse και του JUnit. Ένα από τα σημαντικότερα ευρήματα αυτής της μελέτης είναι η παρατήρηση ότι οι αναδομήσεις πραγματοποιούνται συχνά και κυρίως ότι οι προγραμματιστές εφαρμόζουν αναδομήσεις παράλληλα με την ανάπτυξη λογισμικού-floss refactorings. Αναφέρεται επίσης ότι, σύμφωνα με τα αποτελέσματα της μελέτης, οι αναδομήσεις μεσαίου επιπέδου-medium level refactorings, όπως για παράδειγμα η αναδόμηση *Extract Method* εφαρμόζονται συχνότερα, ωστόσο και σε αυτή τη μελέτη, δεν διερευνάται το ερώτημα, εάν οι προγραμματιστές χρησιμοποιούν τις δυνατότητες της αυτόματης εφαρμογής των αναδομήσεων που παρέχονται από τα διάφορα εργαλεία που υποστηρίζουν τη διαδικασία ανάπτυξης-CASE tools για την εξάλειψη συγκεκριμένων προβλημάτων και κυρίως των σύνθετων προβλημάτων που πιθανόν να παρουσιάζει το λογισμικό [34].

Μια ακόμα επιπλέον μελέτη που πραγματοποιήθηκε, αφορά τον εντοπισμό αναδομήσεων σε γενιές πέντε διαφορετικών έργων λογισμικού ανοιχτού κώδικα με σκοπό τη διερεύνηση της σχέσης μεταξύ των αναδομήσεων και των πιθανών μελλοντικών ελαττωμάτων που παρουσίαζαν τα σχετικά έργα λογισμικού σε χρονικό διάστημα έξι μηνών. Τα ευρήματα της μελέτης δείχνουν ότι όσο αυξάνονται οι ενέργειες αναδόμησης στις προηγούμενες γενιές των λογισμικών, τόσο μειώνονται και τα πιθανά μελλοντικά του μειονεκτήματά [39]. Αναφέρεται επίσης ότι, ο εντοπισμός των αναδομήσεων της συγκεκριμένης μελέτης, βασίζεται στην ανάλυση μηνυμάτων κειμένου και συγκεκριμένα στην αναζήτηση 13 λέξεων που αποτελούν κλειδιά στη διαδικασία της αναδόμησης, όπως για παράδειγμα μετακίνηση, τροποποίηση ονόματος και αποκλείοντας τη λέξη ανάγκη αναδόμησης. Η στρατηγική της αναζήτησης των λέξεων έχει ακρίβεια 95.5 % σύμφωνα με την εκτίμηση των αποτελεσμάτων της [38].

Πρόσφατα, αρκετές μελέτες διερευνούν την εξέλιξη των προβλημάτων σχεδίασης ενός συστήματος λογισμικού και την επίπτωση που φέρουν στη συμπεριφορά του συστήματος. Στη συνέχεια παρατίθενται ενδεικτικά μερικές από τις σημαντικές εργασίες που πραγματοποιήθηκαν:

Μια εμπειρική μελέτη περιλαμβάνει την ανάλυση ιστορικών δεδομένων δυο μεγάλων έργων λογισμικού ανοιχτού κώδικα εστιάζοντας στα προβλήματα σχεδίασης "God Class" και "Shotgun Surgery". Ο εντοπισμός των προβλημάτων πραγματοποιείται αυτόματα χρησιμοποιώντας μια στρατηγική που περιλαμβάνει: α) ένα σύνολο μετρικών ποιότητας λογισμικού, β) ένα σύνολο κανόνων για κάθε τιμή μετρικής του συνόλου των μετρικών και γ) μια λογική σύνθεση των αποτελεσμάτων. Ένα σημαντικό συμπέρασμα της συγκεκριμένης ανάλυσης είναι ότι η εξέλιξη του συστήματος πραγματοποιείται σε διαφορετικές φάσεις και ο αριθμός των προβλημάτων σχεδίασης αυξάνεται ή μειώνεται ανάλογα. Επίσης, από το σύνολο των κλάσεων των υπό εξέταση συστημάτων, οι κλάσεις που περιέχουν τα προβλήματα σχεδίασης υφίστανται τις περισσότερες αλλαγές [35].

Παρόμοια μελέτη που πραγματοποιήθηκε και περιλαμβάνει τη στατιστική ανάλυση 29 προβλημάτων σχεδίασης σε αρκετές γενιές δυο έργων λογισμικού ανοιχτού κώδικα, κατέληξε σε παρόμοια αποτελέσματα με τη μελέτη που αναφέρθηκε. Συγκεκριμένα, αναφέρεται ότι, οι κλάσεις που παρουσιάζουν προβλήματα σχεδίασης, είναι περισσότερο πιθανό να αποτελέσουν αντικείμενο αλλαγών καθώς επίσης και ότι τα προβλήματα σχεδίασης γίνονται περισσότερο κατανοητά από τους σχεδιαστές απ' ότι οι τιμές των μετρικών ποιότητας αντικειμενοστρεφούς λογισμικού [25].

Μια ακόμα εμπειρική μελέτη πραγματοποιήθηκε σε τρία δικτυακά λογισμικά ανοιχτού κώδικα στοχεύοντας στην ανάλυση της εξέλιξης των κομματιών πηγαίου κώδικα που παρουσιάζουν σοβαρά προβλήματα-vulnerabilities, όπως για παράδειγμα οι επιθέσεις ασφάλειας, και ο εντοπισμός τους πραγματοποιήθηκε με τη βοήθεια εργαλείων στατικής ανάλυσης. Μερικά ερωτήματα που διερευνά η μελέτη συγκαταλέγονται στο χρόνο παραμονής των προβλημάτων στο σύστημα και το λόγο εξάλειψής τους. Ωστόσο, σύμφωνα με τα στατιστικά αποτελέσματα της μελέτης, τα σοβαρά προβλήματα που παρουσιάζουν τα δικτυακά λογισμικά, σε αντίθεση με τα

προβλήματα σχεδίασης, τείνουν να διαγράφονται από τα δικτυακά λογισμικά υπονοώντας τη διαφορετική μεταχείριση σε θέματα ασφάλειας [9].

3.3 Εργαλεία Εντοπισμού Προβλημάτων Σχεδίασης

Εκτός των εργαλείων-CASE tools που υποστηρίζουν την αυτόματη υλοποίηση της διαδικασίας των αναδομήσεων απαλλάσσοντας φυσικά τους σχεδιαστές από τις λεπτομέρειες της εφαρμογής τους, πρόσφατες ερευνητικές προσεγγίσεις στοχεύουν στην ανάπτυξη εργαλείων για τον αυτόματο εντοπισμό των προβλημάτων σχεδίασης που αποτελούν ευκαιρίες για υλοποίηση των κατάλληλων κάθε φορά αναδομήσεων με σκοπό την εξάλειψή τους.

Δίχως να έχει πραγματοποιηθεί λεπτομερής έρευνα του συγκεκριμένου πεδίου, ενδεικτικά αναφέρονται ορισμένα εργαλεία που υποστηρίζουν την αυτόματη διαδικασία του εντοπισμού συγκεκριμένων προβλημάτων σχεδίασης.

Το ProDeOOS χρησιμοποιεί ένα σύνολο μετρικών ποιότητας αντικειμενοστρεφούς λογισμικού για τον εντοπισμό δυο βασικών τύπων προβλημάτων σχεδίασης που πιθανόν να εμφανίσουν οι κλάσεις ενός συστήματος λογισμικού. Οι θεϊκές κλάσεις με μορφή συμπεριφοράς- Behavioral "God Class" και οι κλάσεις με το πρόβλημα "Data Class", δηλαδή κλάσεις δίχως συμπεριφορά, αποτελούν τους δυο αναφερόμενους τύπους προβλημάτων σχεδίασης [28].

Το jCOSMO [47] και ο ανάδοχός του CodeNose [41] υποστηρίζουν τον αυτόματο εντοπισμό των προβλημάτων σχεδίασης "Switch Statement", "Long Method", "Long Parameter List", "Message Chain", "Empty Catch Clause", "Refused Bequest", "Feature Envy" καθώς επίσης και μερικά επιπρόσθετα προβλήματα σχεδίασης που σχετίζονται με το μέγεθος των κλάσεων. Ο εντοπισμός των προβλημάτων πραγματοποιείται χρησιμοποιώντας τη μέθοδο της στατική ανάλυσης.

Το iPlasma με τη βοήθεια ενός συνόλου μετρικών ποιότητας αντικειμενοστρεφούς λογισμικού και σε συνδυασμό με τους τελεστές AND/OR παράγει ευρετικούς κανόνες σχεδίασης με σκοπό των εντοπισμό μερικών προβλημάτων σχεδίασης [43].

Το εργαλείο DETectionEXpert-DETEX, επιτρέπει στους σχεδιαστές λογισμικού και κυρίως σε εκείνους που πραγματοποιούν συντήρηση λογισμικού να εντοπίσουν αυτόματα τα εξής ευρέως γνωστά προβλήματα σχεδίασης: αντι-πρότυπα Blobs, Functional Decomposition, Spaghetti Code, και Swiss Army Knife [33].

Το εργαλείο Borland Together επίσης βασίζεται σε συνδυασμό ενός σύνολο μετρικών ποιότητας αντικειμενοστρεφούς λογισμικού και προκαθορισμένων τιμών [3].

ΚΕΦΑΛΑΙΟ 4

ΠΡΟΒΛΗΜΑΤΑ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΕΡΓΑ ΛΟΓΙΣΜΙΚΟΥ

Περιεχόμενα

4.1 Προβλήματα Σχεδίασης 21

4.2 Έργα Λογισμικού 32

"Any fool can write code that a computer can understand
but
good programmers write code that humans can understand!"

[Fowler with contributions:
Refactoring: Improving the design of Existing Code, 1999]

4.1 Προβλήματα Σχεδίασης

Οι κακές οσμές-bad smells, όπως είναι ευρέως γνωστές [13], αποτελούν συμπτώματα κακής ποιότητας σχεδίασης σε αντικειμενοστρεφή συστήματα λογισμικού και προκαλούν σοβαρά προβλήματα συντηρησημότητας. Με τη βοήθεια των εργαλείων jDeodorant και inCode, θα εξετάσουμε έργα λογισμικού ανοιχτού κώδικα για τον εντοπισμό των προβλημάτων σχεδίασης που έπονται.

Το εργαλείο jDeodorant που αξιοποιείται πλήρως στη παρούσα μελέτη, εντοπίζει τρεις σύνθετους τύπους προβλημάτων σχεδίασης και υποστηρίζει αυτόματα την υλοποίηση της κατάλληλης τεχνικής επίλυσής τους απαλλάσσοντας τους συντηρητές έργων λογισμικού από την επίπονη και χρονοβόρα διαδικασία [19]. Ενώ το εργαλείο inCode υποστηρίζει τον αυτόματο εντοπισμό τεσσάρων τύπων προβλημάτων σχεδίασης και πραγματοποιεί συστάσεις στους συντηρητές έργων λογισμικού για την εφαρμογή της κατάλληλης αναδόμησης για την εξάλειψή τους [18]. Ωστόσο, στα πλαίσια της παρούσας μελέτης, επιλέγεται προς διερεύνηση μονάχα ένας τύπος του σύνολο των προβλημάτων σχεδίασης που εντοπίζει το εργαλείο inCode.

Θα πρέπει να σημειωθεί ότι, τα προβλήματα σχεδίασης που έπονται, επιλέχθηκαν προς αναζήτηση κυρίως γιατί θεωρούνται τα περισσότερο σημαντικά μεταξύ των προβλημάτων που εμφανίζονται σε έργα λογισμικού μεγάλης κλίμακας, υλοποιημένα σε αντικειμενοστρεφή γλώσσα προγραμματισμού, μιας και σχετίζονται με τη διανομή της λειτουργικότητας μεταξύ των κλάσεων και των μεθόδων του συστήματος. Παράλληλα, η εξάλειψη των συγκεκριμένων προβλημάτων σχεδίασης επιτυγχάνεται κατόπιν στοχευόμενων ενεργειών από πλευράς των σχεδιαστών πράγμα που διαχωρίζει σαφώς τις σκόπιμες ενέργειες αναδόμησης από τις ακούσιες ενέργειες αφαίρεσης των προβλημάτων που προκύπτουν λόγω διορθωτικής ή προσαρμοστικής συντήρησης του συστήματος. Τέλος σημειώνεται για ακόμα μια φορά, ότι ο εντοπισμός τους παρέχεται αυτόματα από τα αναφερόμενα εργαλεία.

"Long Method" Bad Smell

Οι μέθοδοι, γνωστές και ως μέθοδοι οριζόμενες από το χρήστη, επιτρέπουν στο προγραμματιστή να διαρθρώσει ένα πρόγραμμα χωρίζοντας τις εργασίες του σε

αυτόνομες μονάδες. Οι πραγματικές προτάσεις που υλοποιούν τις μεθόδους γράφονται μια φορά και αποκρύπτονται από εκείνες των άλλων μεθόδων.

Υπάρχουν διάφορα κίνητρα για την διάρθρωση ενός προγράμματος μέσω μεθόδων. Ένα κίνητρο είναι η προσέγγιση του "διαίρει και βασίλευε" που διευκολύνει σημαντικά την διαχείριση της ανάπτυξης προγραμμάτων, δομώντας προγράμματα από μικρά, απλά κομμάτια. Ένα άλλο κίνητρο είναι η "επαναληπτική χρήση λογισμικού". Δηλαδή, η χρήση υπαρκτών μεθόδων ως δομικών μπλοκ για τη δημιουργία νέων προγραμμάτων. Συχνά, μπορείτε να δημιουργήσετε προγράμματα από τυποποιημένες μεθόδους, αντί να δομείτε προσαρμοσμένο κώδικα. Ένα τρίτο κίνητρο είναι η αποφυγή της δημιουργίας των πολλαπλών αντιγράφων πηγαίου κώδικα, ευρέως γνωστά ως "κλώνοι" [2], μέσα στο πρόγραμμα. Η διαίρεση ενός προγράμματος σε σημαντικές μεθόδους καθιστά ευκολότερη την ανίχνευση των λαθών καθώς και τη συντήρησή του [6].

Το πρόβλημα της περίπλοκης μεθόδου-"Long Method" bad smell, αναφέρεται σε σύνθετες μεθόδους που αποτελούνται από πολλές και διαφορετικές μεταξύ τους λειτουργίες. Δηλαδή μεθόδους με μεγάλο μέγεθος, υψηλή πολυπλοκότητα και χαμηλή συνεκτικότητα[2]. Κάθε μέθοδος θα πρέπει να περιορίζεται στην εκτέλεση μιας σαφώς ορισμένης εργασίας και το όνομα της μεθόδου θα πρέπει να εκφράζει περιγραφικά αυτή την εργασία. Τέτοιες μέθοδοι καθιστούν ευκολότερη τη γραφή, την ανάγνωση, τη διόρθωση, τη συντήρηση αλλά και την τροποποίηση των προγραμμάτων [13].

Το ακόλουθο σχήμα απεικονίζει το πρόβλημα "Long Method" που παρουσιάζει η μέθοδος `printOwing()`. Η μέθοδος, σε κάποιο σημείο του σώματός της, περιέχει σχόλια για την επεξήγηση του μπολ κώδικα που έπεται. Παρατηρώντας το συγκεκριμένο μπλοκ κώδικα, καταλαβαίνουμε ότι η λειτουργία που εκτελεί δεν σχετίζεται άμεσα με τη λειτουργία που υποδηλώνει το όνομα της μεθόδου και συνεπώς, θα έπρεπε να βρίσκεται σε ξεχωριστή μέθοδο.

[2] Σημειώνεται ότι, σε επίπεδο μεθόδων, με τον όρο συνεκτικότητα αναφερόμαστε στο κατά πόσο τα εσωτερικά συστατικά της συσχετίζονται μεταξύ τους και είναι όλα απαραίτητα για την επίτευξη της λειτουργίας που αναλαμβάνει να εκτελέσει η αναφερόμενη μέθοδος. Η συνεκτικότητα των μεθόδων μπορεί για παράδειγμα να εκφραστεί με χρήση μετρικών συνεκτικότητας που βασίζονται στην ιδέα του slice [32].

"Long Method" Bad Smell

```
void printOwing() {

   Enumeration e = _orders.elements ();
   double outstanding = 0.0;

   printBanner();

   // calculate outstanding
   while(e.hasMoreElements()) {
      Order each = (Order) e.nextElement();
      outstanding += each.getAmount();
   }

   // print details
   System.out.println("name:"   + _name);
   System.out.println("amount:" + outstanding);
}
```

Σχήμα 4.1: Παράδειγμα προβλήματος σχεδίασης "Long Method"

Η αντιμετώπιση του προβλήματος των σύνθετων μεθόδων επιτυγχάνεται με τις αναδομήσεις *Extract Method*, *Replace Temp with Query*, *Replace Method with Method Object* και *Decompose Conditional* ανάλογα φυσικά με τη περίπτωση κάθε φορά [13]. Για την εξάλειψη του προβλήματος που παρουσιάζει η μέθοδος `printOwing()` εφαρμόζεται η ενδεδειγμένη τεχνική επίλυσης *Extract Method* για την εξαγωγή του μπλοκ κώδικα που εκτελεί λειτουργικότητα που δεν υποδηλώνεται άμεσα από το όνομά της.

Implementation of *Extract Method* Refactoring

```
void printOwing() {

   double outstanding = getOutstanding();

   printBanner();

   // print details
   System.out.println("name:"   + _name);
   System.out.println("amount:" + outstanding);
}


double getOutstanding() {

   Enumeration e = _orders.elements ();
   double outstanding = 0.0;

   while(e.hasMoreElements()) {
      Order each = (Order) e.nextElement();
      outstanding += each.getAmount();
   }

  return(outstanding);
}
```

Σχήμα 4.2: Παράδειγμα εφαρμογής αναδόμησης *Extract Method*

Στο Σχήμα 4.2 απεικονίζεται η νέα-τροποποιημένη έκδοση της μεθόδου `printOwing()` καθώς επίσης και η νέα μέθοδος `getOutstanding()`, που δημιουργείται με την εφαρμογής της αναδόμησης και με όνομα που να υποδηλώνει τη λειτουργία του πηγαίου της κώδικα.

Το jDeodorant εντοπίζει προβλήματα σχεδίασης "Long Method" και προτείνει για την εξάλειψη του αναφερόμενου προβλήματος, την αναδόμηση *Extract Method*. Η μεθοδολογία αυτόματου εντοπισμού του συγκεκριμένου προβλήματος βασίζεται στην ιδέα του slice. Δηλαδή, στην εύρεση όλων των συνόλων των statements του πηγαίου κώδικα με κάποια εξάρτηση είτε λόγω των δεδομένων-data dependency είτε λόγω κάποιας συνθήκης ελέγχου ή δομής επανάληψης-control dependency τα οποία επηρεάζουν την τελική τιμή μιας μεταβλητής [44].

Θα πρέπει επίσης να αναφερθεί ότι, πρόσφατες εξελίξεις της μεθοδολογίας του εργαλείου, επιτρέπουν τον εντοπισμό και την εξαγωγή όλων των συνόλων των statements του πηγαίου κώδικα τα οποία επηρεάζουν τη κατάσταση ενός αντικειμένου.

"Feature Envy" Bad Smell

Μια από τις σημαντικότερες αποφάσεις στον αντικειμενοστρεφή σχεδιασμό, αν όχι η σημαντικότερη, είναι η απόφαση της τοποθέτησης των αρμοδιοτήτων του συστήματος. Όλη η σημασία των αντικειμένων είναι η τεχνική που επιτρέπει την ομαδοποίηση των ομάδων δεδομένων που σχετίζονται μεταξύ τους και εκτελούν συγκεκριμένες λειτουργίες. Ένα κλασικό πρόβλημα σχεδίασης το οποίο παραβιάζει την παραπάνω αρχής της ομαδοποίησης, είναι το πρόβλημα της μεθόδου που ενδιαφέρεται περισσότερο για τα χαρακτηριστικά μιας άλλης κλάσης παρά για τα χαρακτηριστικά της κλάσης στην οποία ανήκει-"Feature Envy" bad smell [13].

Στο ακόλουθο σχήμα αναπαρίσταται το πρόβλημα "Feature Envy". Η μέθοδος `amountFor()` της κλάσης `Customer` για την εκτέλεση της λειτουργίας της, χρησιμοποιεί-ζηλεύει μονάχα τα χαρακτηριστικά της κλάσης `Rental`, που δέχεται ως παράμετρο, και κανένα χαρακτηριστικό της κλάσης `Customer` στην οποία και πραγματικά ανήκει.

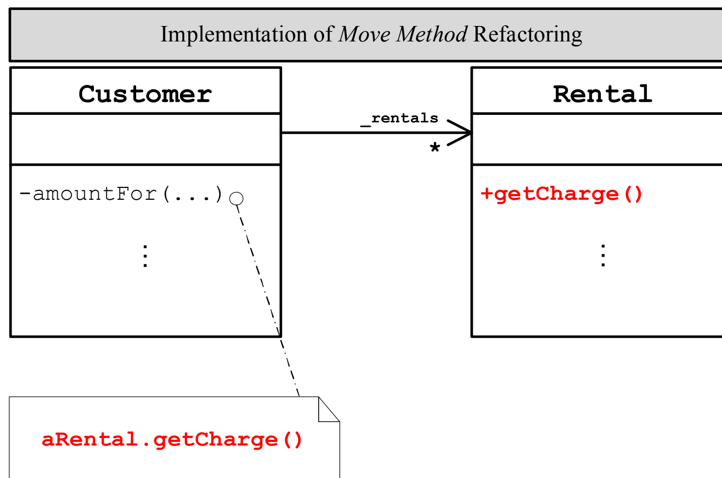
"Feature Envy" Bad Smell

```
class Customer...

  private double amountFor(Rental aRental) {

     double result = 0;

     switch(aRental.getMovie().getPriceCode()) {
        case Movie.REGULAR:
           result += 2;
           if(aRental.getDaysRented() > 2)
              result += (aRental.getDaysRented()
                                        - 2) * 1.5;
          break;
        case Movie.NEW_RELEASE:
           result += aRental.getDaysRented() * 3;
          break;
        case Movie.CHILDRENS:
           result += 1.5;
           if(aRental.getDaysRented() > 3)
              result += (aRental.getDaysRented()
                                        - 3) * 1.5;
          break;
     }
    return(result);
  }
```

Σχήμα 4.3: Παράδειγμα προβλήματος σχεδίασης "Feature Envy"

Για την εξάλειψη του προβλήματος "Feature Envy" υπάρχουν πολλές υποψήφιες αναδομήσεις που πιθανόν να χρειαστεί να εφαρμοστούν ανάλογα φυσικά με τη κάθε περίπτωση, όπως: α) *Move Method*, μεταφορά της προβληματικής μεθόδου στη κλάση της οποίας τις ιδιότητες προσπελαύνει, β) *Extract + Move Method*, εξαγωγή τμήματος κώδικα της μεθόδου σε ξεχωριστή μέθοδο και μεταφορά της νεοδημιουργηθείσας προβληματικής μεθόδου στη κλάση την οποία ζηλεύει και γ) *Move Field*, μεταφορά της ιδιότητας της κλάσης, στη κλάση την οποία και ζηλεύει [13]. Για την αντιμετώπιση του προβλήματος σχεδίασης που παρουσιάζει η μέθοδος `amountFor()` της κλάσης `Customer` θα εφαρμοστεί η ενδεδειγμένη τεχνική επίλυσης *Move Method* για τη μεταφορά της μεθόδου στη κλάση `Rental`, της οποίας τις ιδιότητες και θα προσπελαύνει άμεσα.

Το διάγραμμα κλάσεων του Σχήματος 4.4, απεικονίζει τις αλλαγές που επιφέρει η εφαρμογή της αναδόμησης και στις δυο κλάσεις. Η μόνη λειτουργικότητα της συνάρτησης `amountFor()` της κλάσης `Customer` είναι η απλή μεταβίβαση μηνύματος-simple delegation message στη νέα μέθοδο `getCharge()` της κλάσης `Rental`.



Σχήμα 4.4: Παράδειγμα εφαρμογής αναδόμησης *Move Method*

Το jDeodorant εντοπίζει προβλήματα σχεδίασης τύπου "Feature Envy" και προτείνει τη τεχνική επίλυσης *Move Method* για την εξάλειψη του συγκεκριμένου προβλήματος. Η μεθοδολογία για τον αυτόματο εντοπισμό του αναφερόμενου προβλήματος βασίζεται στην ιδέα της απόστασης ανάμεσα στη μέθοδο/ιδιότητα και τη κλάση της οποίας τα χαρακτηριστικά χρησιμοποιεί-ζηλεύει. Σε περίπτωση που η απόσταση μεταξύ της μεθόδου/ιδιότητας και της κλάσης είναι μικρότερη από την απόσταση με τη κλάση στην οποία ανήκει, τότε το πρόβλημα προτείνεται από το εργαλείο [45].

"State Checking" Bad Smell

Η έννοια του πολυμορφισμού-polymorphism αποτελεί μια από τις κύριες πλευρές του αντικειμενοστρεφούς προγραμματισμού. Με τον όρο πολυμορφισμό, αναφερόμαστε στη διεργασία μέσω της οποίας οι διαφορετικές υλοποιήσεις μιας μεθόδου είναι δυνατό να προσπελαστούν με το ίδιο όνομα. Γι' αυτό το λόγο, ο πολυμορφισμός, περιγράφεται με μια φράση "μια διασύνδεση, πολλές μέθοδοι". Αυτό σημαίνει ότι κάθε μέλος μια γενικής τάξης λειτουργιών μπορεί να προσπελαστεί με τον ίδιο τρόπο, παρά το γεγονός ότι οι συγκεκριμένες ενέργειες που σχετίζονται με κάθε λειτουργία μπορεί να διαφέρουν. Συνεπώς, ο πολυμορφισμός αποτελεί το βασικό μηχανισμό που δίνει τη δυνατότητα να προγραμματίζουμε γενικώς αντί να προγραμματίζουμε ειδικώς [6].

Το λογισμικό που εμπλέκει πολυμορφική συμπεριφορά, είναι ανεξάρτητο του τύπου αντικειμένου στο οποίο αποστέλλονται τα μηνύματα. Νέοι τύποι δεδομένων

που μπορούν να αποκριθούν σε κλήσεις υπαρκτών μεθόδων είναι εφικτό να ενσωματωθούν στο σύστημα δίχως να απαιτείται η τροποποίησή του. Το μοναδικό σημείο που πρέπει να τροποποιηθεί είναι ο κώδικας του πελάτη-client που δημιουργεί νέα αντικείμενα για να επιτύχει τους νέους τύπους [50].

Παρά τη θεμελιώδη σημασία της έννοιας του πολυμορφισμού στην αντικειμενοστρεφή σχεδίαση, συχνά παρατηρείται, σε πολλαπλά σημεία του πηγαίου κώδικα ενός συστήματος λογισμικού, συνθήκες ελέγχου να καθορίζουν τη ροή της εκτέλεσης βάση είτε της τιμής μιας μεταβλητής που αντιπροσωπεύει τη τρέχουσα κατάσταση του αντικειμένου και συγκρίνοντάς την με ένα σύνολο σταθερών που υποδηλώνουν μια πιθανή εναλλακτική κατάσταση, είτε, ανακτώντας το τύπο του ενεργού αντικειμένου με χρήση του μηχανισμού της αναγνώρισης τύπου κατά το χρόνο εκτέλεσης-run time type identification (RTTI). Η χρήση της ανωτέρω επιλογής, υποδηλώνει κακό σχεδιασμό-sloppy/bad design του συστήματος και αυξάνει σημαντικά το μέγεθος του πηγαίου κώδικα εξαιτίας των αλυσιδωτών εντολών `if/else` ή εντολών `switch` που υπάρχουν σε διάφορα σημεία του πηγαίου κώδικα. Κατά συνέπεια, η αναγνωσιμότητα του συστήματος μειώνεται και προκαλούνται σοβαρά προβλήματα στη συντηρησημοτητά του. Η προσθήκη επιπρόσθετων λειτουργιών απαιτεί τον εντοπισμό, τη κατανόηση και τη τροποποίηση όλων των σημείων του κώδικα που πραγματοποιείται ο έλεγχος ροής-"State Checking" bad smell, ευρέως γνωστό ως "Switch Statements" bad smell, διαδικασία εξαιρετικά επίπονη και δαπανηρή από πλευράς χρόνου και ενεργειών [13].

Η ύπαρξη του συγκεκριμένου προβλήματος σχεδίασης ουσιαστικά υποδηλώνει τη παραβίαση της αρχής της Ανοιχτής και Κλειστής Σχεδίασης-Open and Closed Principle και συνεπώς, το σύστημα είναι εύθραυστο-fragility καθώς όλες οι απαιτούμενες αλλαγές είναι πολύ πιθανό να εισάγουν πολλαπλά σφάλματα [31].

Το Σχήμα 4.5, απεικονίζει το πρόβλημα "State Checking" που παρουσιάζει η μέθοδος `payAmount()` της κλάσης `Employee`. Η μέθοδος, ελέγχει τη τιμή της μεταβλητής μέλους με όνομα `_type` και με βάση αυτή, καθορίζει τη ροή της εκτέλεσης συγκρίνοντάς την με ένα σύνολο επώνυμων σταθερών, ENGINEER, SALESMAN, κλπ, που υποδηλώνουν μια πιθανή εναλλακτική κατάσταση.

"State Checking" Bad Smell

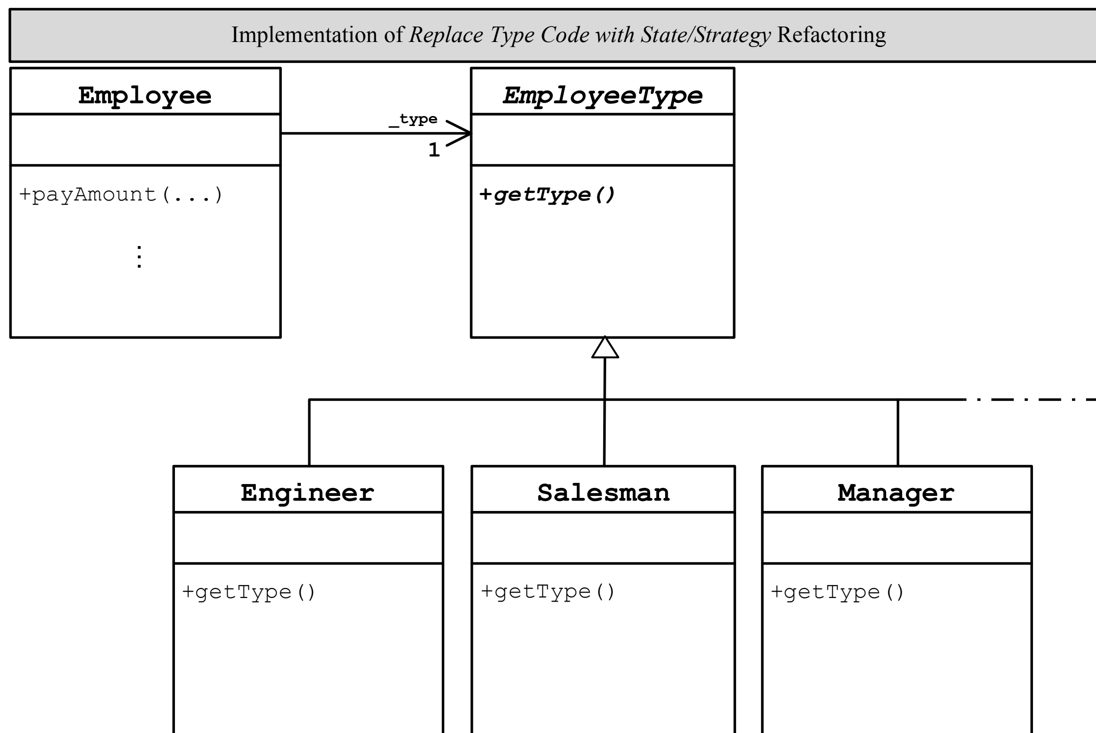
```
class Employee...

   int payAmount() {

      switch (_type) {

         case ENGINEER:
            // some code

         case SALESMAN:
            // some code

         case MANAGER:
            // some code

         default:
            // some code
      }
   }
```

Σχήμα 4.5: Παράδειγμα προβλήματος σχεδίασης "State Checking"

Η αντιμετώπιση του προβλήματος "State Checking" επιτυγχάνεται με τις αναδομήσεις *Replace Conditional with Polymorphism*, *Replace Type Code with Subclasses*, *Replace Type Code with State/Strategy*, *Replace Parameter with Explicit Methods* και *Introduce Null Object* ανάλογα με τη περίπτωση κάθε φορά [13]. Για την αντιμετώπιση του προβλήματος που παρουσιάζει η μέθοδος `payAmount()` της κλάσης `Employee` θα εφαρμοστεί η αναδόμηση *Replace Type Code with State/Strategy*. Δηλαδή, η εφαρμογή είτε του προτύπου σχεδίασης "Κατάσταση-State" είτε του προτύπου "Στρατηγική-Strategy" επιτρέποντας την αντιστροφή της εξάρτησης του γενικού αλγορίθμου από τη συγκεκριμένη υλοποίηση [14].

Στο Σχήμα 4.6, απεικονίζεται το διάγραμμα κλάσεων που προκύπτει με την εφαρμογή της αναδόμησης που αναφέρθηκε για την αντιμετώπιση του προβλήματος "State Checking". Η αφηρημένη-abstract βασική κλάση *`EmployeeType`*, παραγοντοποιεί την κοινή συμπεριφορά όλων των τύπων των υπαλλήλων. Αντικείμενα της κάθε κλάσης που υλοποιεί ένα συγκεκριμένο τύπο υπαλλήλου, "περνάνε" στο αντικείμενο της κλάσης του γενικού τύπου προς εκτέλεση των λειτουργιών τους. Διάφοροι δυνατοί τύποι υπαλλήλων, ενσωματώνονται σε διαφορετικές κλάσεις, για παράδειγμα, `Engineer`, `Salesman`, κλπ, και εναλλάσσονται ανεξάρτητα από τη κλάση `Employee`. Με αυτό τον τρόπο, αποφεύγεται η χρήση εντολών ελέγχου για την επιλογή μιας συγκεκριμένης

υλοποίησης. Ωστόσο, όλοι οι τύποι υπαλλήλων θα πρέπει να υλοποιούν την ίδια διασύνδεση, όπως αυτή ορίζεται από την αφηρημένη βασική υπερκλάση.



Σχήμα 4.6: Παράδειγμα εφαρμογής αναδόμησης *Replace Type with State/Strategy*

Το jDeodorant εντοπίζει προβλήματα σχεδίασης "State Checking" αναζητώντας κομμάτια πηγαίου κώδικα που είναι εφικτό να αντικατασταθούν με το μηχανισμό που μας παρέχει ο πολυμορφισμός. Συγκεκριμένα, εντοπίζει σημεία του κώδικα που καθορίζουν τη ροή της εκτέλεσης είτε μέσω συνθηκών ελέγχου είτε μέσω της χρήσης του μηχανισμού της αναγνώρισης τύπου κατά το χρόνο εκτέλεσης και προτείνει, ανάλογα φυσικά με τη περίπτωση κάθε φορά, τις αναδομήσεις *Replace Conditional with Polymorphism* και *Replace Type Code with State/Strategy* [46].

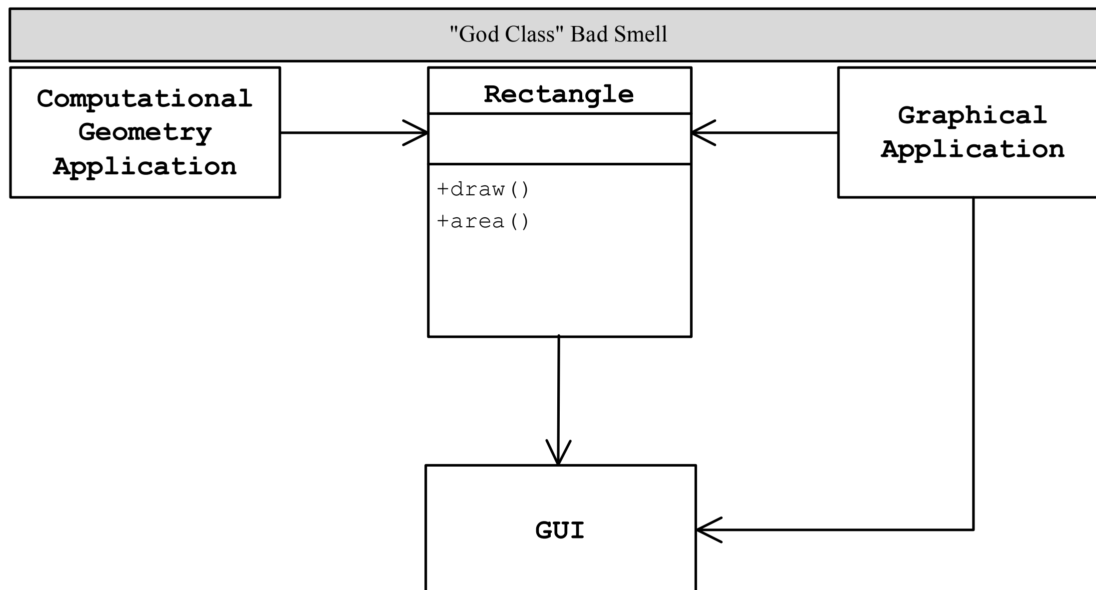
"God Class" Bad Smell

Με την εξέλιξη ενός συστήματος λογισμικού, είναι λογικό, να προστίθενται στις κλάσεις του συστήματος νέες λειτουργικότητες αλλά και ιδιότητες μιας και δεν αξίζει το κόπο να δημιουργηθούν νέες κλάσεις που να τις υποστηρίζουν. Καθώς όμως οι λειτουργικότητες αυτές αυξάνουν, οι αναφερόμενες κλάσεις γίνονται ολοένα και περισσότερο περίπλοκες με αποτέλεσμα να αναλαμβάνουν πολλές και διαφορετικές μεταξύ τους αρμοδιότητες.

Το πρόβλημα των θεϊκών κλάσεων-"God Class" bad smell, ευρέως γνωστό ως "Large Class" bad smell [13], αποτελεί παραβίαση ενός από τους βασικότερους αντικειμενοστρεφείς ευρετικούς κανόνες σχεδίασης-object oriented design heuristics και αναφέρεται συνήθως σε υπερβολικά αναπτυγμένες κλάσεις που ενσωματώνουν μεγάλος μέρος της λογικής του συστήματος. Η καταστρατήγηση του συγκεκριμένου ευρετικού κανόνα πραγματοποιείται με δυο τρόπους: α) είτε δημιουργώντας κλάσεις που συγκεντρώνουν πολλά δεδομένα του συστήματος-"Data God Classes", β) είτε δημιουργώντας κλάσεις που εμφανίζουν υπερβολική συμπεριφορά-"Behavioral God Classes" η οποία και δεν σχετίζεται φυσικά με την επικοινωνία μεταξύ άλλων αντικειμένων. Δηλαδή, οι μέθοδοι της κλάσης δρουν σε ένα συγκεκριμένο υποσύνολο των μελών των δεδομένων της [40].

Μια κλάση με πολλές και διαφορετικές αρμοδιότητες εμφανίζει αναπόφευκτα εξαιρετικά χαμηλή σύζευξη μεταξύ των αρμοδιοτήτων της πράγμα που σημαίνει ότι αλλαγές στη μια αρμοδιότητα είναι δυνατό να εμποδίσουν αλλαγές για την ικανοποίηση μιας άλλης απαίτησης ή ακόμα χειρότερα, αν ο προγραμματιστής δεν επιδείξει την κατάλληλη προσοχή, είναι πολύ πιθανό να δημιουργηθούν σφάλματα στις άλλες αρμοδιότητες. Είναι λοιπόν προφανές, ότι σε αυτή τη περίπτωση οδηγούμαστε σε δύσκαμπτο-rigidity και εύθραυστο-fragility σχεδιασμό της κλάσης εξαιτίας της δυσκολίας που παρουσιάζει η τροποποίησή της σε επερχόμενες αλλαγές [31].

Το ακόλουθο σχήμα, απεικονίζει το πρόβλημα "God Class" που παρουσιάζει η κλάση `Rectangle` καθώς αναλαμβάνει το ρόλο του κεντρικού συντονιστή λαμβάνοντας και αποστέλλοντας μηνύματα στις υπόλοιπες εφαρμογές. Από το διάγραμμα κλάσεων γίνεται αντιληπτό ότι η συγκεκριμένη κλάση περιλαμβάνει λειτουργικότητα που αφορά τη γραφική διασύνδεση χρήστη-μέθοδος `draw()`, και λειτουργικότητα που αφορά τον υπολογισμό εμβαδού του τετραγώνου-μέθοδος `area()`, για μια γεωμετρική εφαρμογή η οποία δεν πρόκειται ποτέ να σχεδιάσει τετράγωνα. Η συγκέντρωση αρμοδιοτήτων στη κλάση `Rectangle` που αφορούν δυο διαφορετικές εφαρμογές, τη καθιστά αναμφίβολα ως θεϊκή κλάση καθώς ενδέχεται να οδηγήσει σε προβλήματα κατά τη διαδικασία τροποποίησης της μιας λειτουργικότητας και επηρεασμού της άλλης.

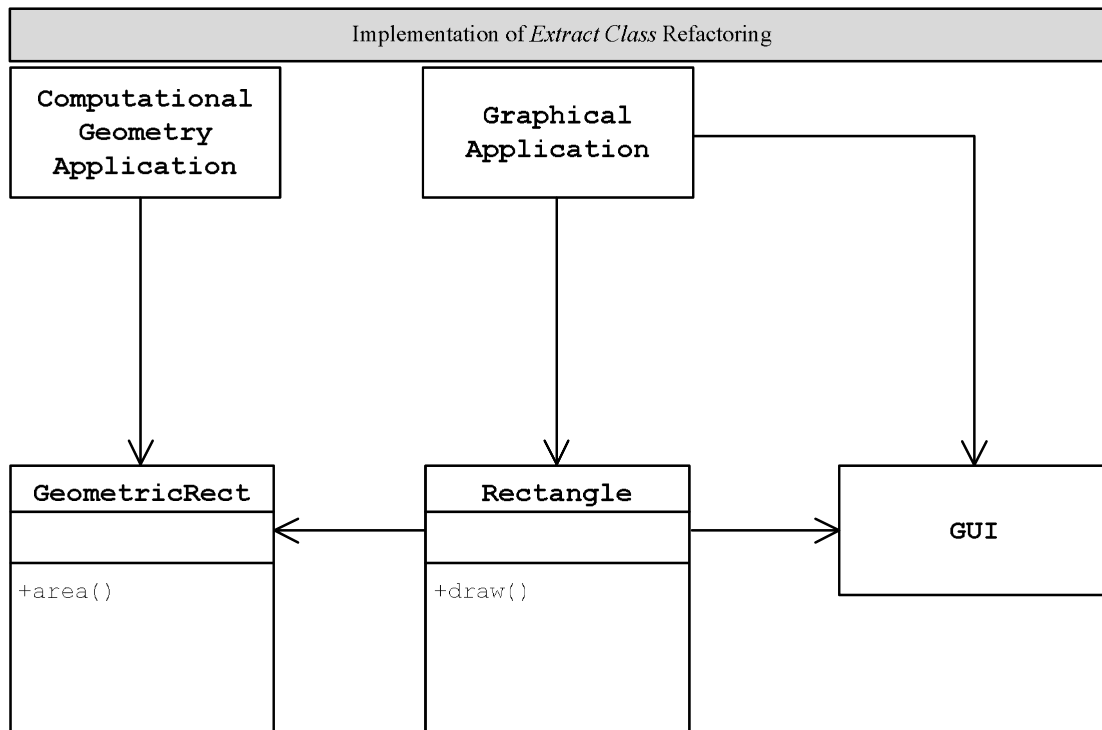


Σχήμα 4.7: Παράδειγμα προβλήματος σχεδίασης "God Class"

Όπως συμβαίνει με όλα τα προβλήματα σχεδίασης, έτσι και το πρόβλημα "God Class", είναι εφικτό να αντιμετωπιστεί με την εφαρμογή της κατάλληλης ενδεδειγμένης τεχνικής επίλυσης κάθε φορά. Οι αναδομήσεις *Extract Class*, *Extract Subclass*, *Extract Interface* και *Replace Data Value with Object* δίνουν λύση στο αναφερόμενο πρόβλημα [13]. Για την αντιμετώπιση του προβλήματος που παρουσιάζει η κλάση `Rectangle` θα εφαρμοστεί η αναδόμηση *Extract Class* για το διαχωρισμό των αρμοδιοτήτων σε δυο τελείως διαφορετικές κλάσεις.

Η εξάλειψη του αναφερόμενου προβλήματος στην ουσία υποδηλώνει ρητά τη συμμόρφωση της κλάσης με την Αρχή της Μοναδικής Αρμοδιότητας-Single Responsibility Principle (SRP) η οποία και εξασφαλίζει ότι κάθε κλάση του συστήματος πρέπει να έχει ένα και μοναδικό λόγο να αλλάξει [31].

Στο Σχήμα 4.8, απεικονίζεται το διάγραμμα κλάσεων που προκύπτει με την εφαρμογή της αναδόμησης που αναφέρθηκε για την αντιμετώπιση του προβλήματος "God Class" που παρουσιάζει η κλάση `Rectangle`. Η λειτουργικότητα που αφορά τη γεωμετρική εφαρμογή μετακινείται στη νέα κλάση `GeometricRect`. Πλέον, οι δυο κλάσεις περιλαμβάνουν πλήρως συσχετισμένα συστατικά και είναι όλα απαραίτητα για την εκτέλεση της ίδιας και μοναδικής λειτουργίας που αναλαμβάνει η κάθε κλάση.



Σχήμα 4.8: Παράδειγμα εφαρμογής αναδόμησης *Extract Class*

Το inCode εντοπίζει προβλήματα σχεδίασης "God Class" αναζητώντας μη συνεκτικές κλάσεις με μεγάλο μέγεθος οι οποίες παρουσιάζουν υψηλή σύζευξη μεταξύ των υπολοίπων κλάσεων του συστήματος. Συγκεκριμένα, για τον εντοπισμό των θεϊκών κλάσεων χρησιμοποιεί ένα σύνολο μετρικών ποιότητας αντικειμενοστρεφούς λογισμικού που περιλαμβάνουν τις μετρικές *Access To Foreign Data-ATFD*, *Weighted Method Count-WMC*, και *Tight Class Cohesion-TCC* [43], [29].

4.2 Έργα Λογισμικού

Για τον εντοπισμό των προβλημάτων σχεδίασης που παρουσιάστηκαν στη προηγούμενη ενότητα και την εξαγωγή των συμπερασμάτων της παρούσας εμπειρικής μελέτης, επιλέχτηκαν προς ανάλυση δυο έργα λογισμικού ανοιχτού κώδικα, το JFlex και το JFreeChart.

Θα πρέπει να σημειωθεί ότι η επιλογή των συγκεκριμένων έργων λογισμικού πραγματοποιήθηκε κυρίως γιατί:

- Υλοποιήθηκαν σε γλώσσα προγραμματισμού Java επιτρέποντας τα εργαλεία jDeodorant και inCode να αναλύσουν το πηγαίο τους κώδικα.

- Υπάρχουν ελεύθερα διαθέσιμες αρκετές γενιές και για τα δυο έργα λογισμικού στα αποθετήρια πηγαίου τους κώδικα.

- Πρόκειται για έργα λογισμικού με διάρκεια ζωής περισσότερο των 9 χρόνων πράγμα που αυξάνει τις πιθανότητες εντοπισμού προβλημάτων σχεδίασης. Σημείων δηλαδή του πηγαίου κώδικα που αποτελούν ευκαιρίες για αναδόμηση.

"JFlex" Open-Source Project

Το JFlex πρόκειται για έναν γρήγορο δημιουργό λεξικολογικού αναλυτή για Java, υλοποιημένο στη γλώσσα προγραμματισμού Java. Έχει επίσης αναπτυχθεί και ως αυτόνομο εργαλείο, το JLex, το οποίο αναπτύχθηκε από τον Elliot Berk στο πανεπιστήμιο Princeton και το οποίο βασίζεται στο μοντέλο του Lex-δημιουργό λεξικολογικού αναλυτή για C σε λειτουργικά συστήματα UNIX [20].

Η ανάλυση για πιθανά προβλήματα σχεδίασης στο αναφερόμενο έργο εκτελέστηκε για το πακέτο `JFlex`. Συνολικά, για την παρούσα μελέτη περίπτωσης, αναλύθηκαν 10 γενιές του έργου, από τη γενιά 1.3 έως και τη 1.4.3.

Στον ακόλουθο πίνακα, παρουσιάζονται για το εξεταζόμενο πακέτο του έργου, τα χαρακτηριστικά μεγέθους για κάθε γενιά.

Πίνακας 4.1
Χαρακτηριστικά μεγέθους εξεταζόμενου πακέτου του έργου JFlex

Measures	***Versions***									
	1.3	***1.3.1***	***1.3.2***	***1.3.3***	***1.3.4***	***1.3.5***	***1.4***	***1.4.1***	***1.4.2***	***1.4.3***
#Classes	34	34	34	35	35	35	40	40	40	40
Source lines of code*	7.463	7.615	7.687	8.211	8.259	8.299	8.591	8.519	9.075	9.084

** Για τη μέτρηση των γραμμών πηγαίου κώδικα-SLOC χρησιμοποιήθηκε το εργαλείο SLOCCount.*

Τέλος αναφέρεται ότι, για την επιτυχή εκτέλεση του έργου, απαιτείται η εγκατάσταση των αρχείων `ant-1.4.1` και `junit-3.7` σε ορισμένες γενιές του έργου.

"JFreeChart" Open-Source Project

Το JFreeChart πρόκειται για μια βιβλιοθήκη της Java που επιτρέπει την δημιουργία ενός συνόλου διαγραμμάτων υψηλής ποιότητας. Το έργο ξεκίνησε από τον κ.David Gilbert πριν από δέκα χρόνια περίπου, το Φεβρουάριο του 2000, ενώ

σήμερα, ασχολούνται με το συγκεκριμένο έργο, περίπου 40 με 50 χιλιάδες σχεδιαστές. Ωστόσο, το έργο διευθύνεται ακόμα από τον κ.Gilbert με τη συνεισφορά διάφορων ομάδων σχεδιαστών [21].

Η ανάλυση για τον εντοπισμό πιθανών προβλημάτων σχεδίασης εκτελέστηκε για το πακέτο `com.jrefinery.chart`. Συνολικά, για την παρούσα μελέτη περίπτωσης, αναλύθηκαν 14 γενιές του έργου, από τη γενιά 0.5.6 έως και τη 0.9.4a.

Στον πίνακα που ακολουθεί, παρατίθενται τα χαρακτηριστικά μεγέθους του εξεταζόμενου πακέτου για κάθε γενιά του έργου.

Πίνακας 4.2
Χαρακτηριστικά μεγέθους εξεταζόμενου πακέτου του έργου JFreeChart

Measures	***Versions***													
	0.5.6	***0.6.0***	***0.7.0***	***0.7.1***	***0.7.2***	***0.7.3***	***0.7.4***	***0.8.0***	***0.8.1***	***0.9.0***	***0.9.1***	***0.9.2***	***0.9.3***	***0.9.4a***
#Classes	47	59	75	66	68	68	70	72	77	99	99	103	106	110
Source lines of code*	3.912	5.801	7.154	6.994	7.456	7.544	7.894	7.961	9.192	12.947	13.021	13.582	15.931	17.620

** Για τη μέτρηση των γραμμών πηγαίου κώδικα-SLOC χρησιμοποιήθηκε το εργαλείο SLOCCount.*

Τέλος αναφέρεται ότι, για την επιτυχή εκτέλεση του λογισμικού, απαιτείται η εγκατάσταση των αρχείων `batik-1.1.1.jar` και `javax.servlet-1.0.0` σε ορισμένες γενιές του έργου.

ΚΕΦΑΛΑΙΟ 5

ΕΜΠΕΙΡΙΚΗ ΜΕΛΕΤΗ

Περιεχόμενα

5.1 Συνολικός Αριθμός Προβλημάτων Σχεδίασης 36

5.2 Παραμονή Προβλημάτων Σχεδίασης 37

5.3 Εξέλιξη Προβλημάτων Σχεδίασης 40

5.4 Ενεργά Προβλήματα Σχεδίασης 49

*"The true worth of experimenter consists in his pursuing
not only what he seeks in his experiment,
but also what he did not seek!"*

[Bernard:
An Introduction to the Study of Experimental Medicine, 1865]

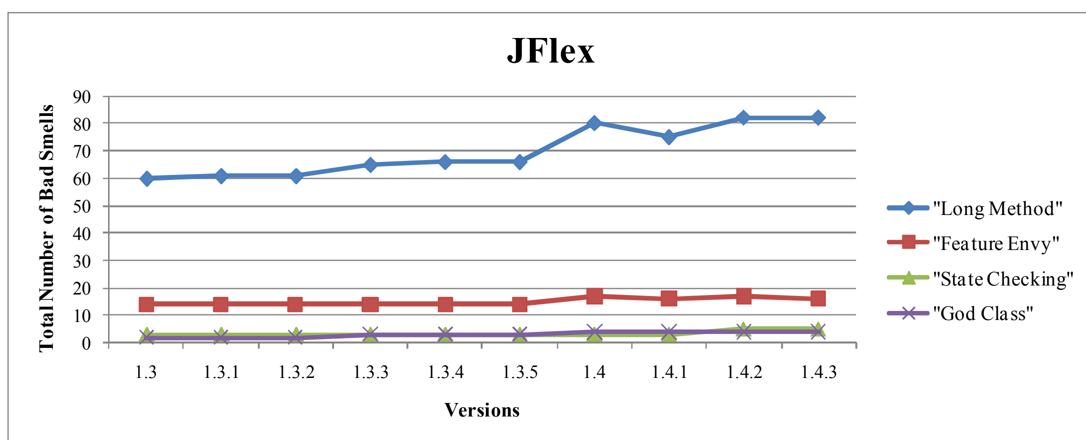
5.1 Συνολικός Αριθμός Προβλημάτων Σχεδίασης

Είναι απόλυτα φυσιολογικό, από τη στιγμή που σε κάθε νέα γενιά ενός έργου λογισμικού προστίθεται νέα λειτουργικότητα και από τη στιγμή που τα έργα λογισμικού ανοιχτού κώδικα δεν υφίστανται συστηματική συντήρηση, ο συνολικός αριθμός των προβλημάτων σχεδίασης να αυξάνεται με την εξέλιξη των συστημάτων λογισμικού.

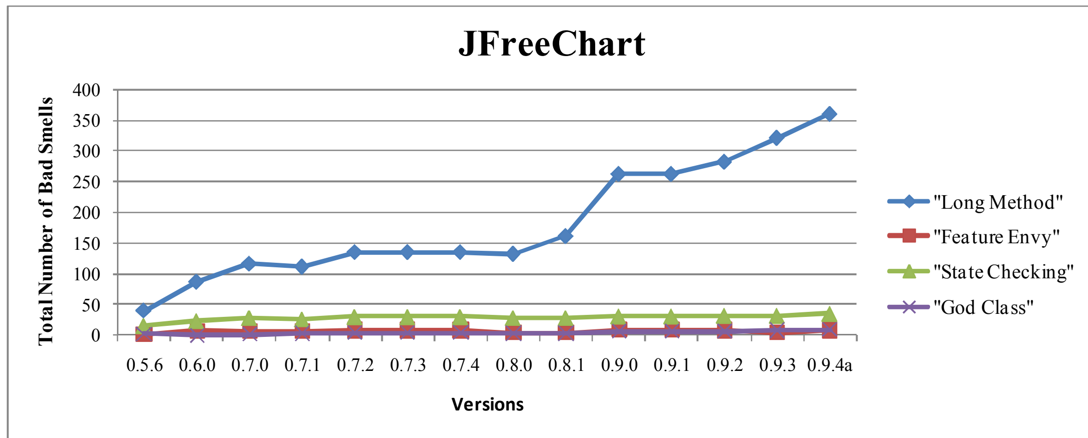
Στα δυο σχήματα που ακολουθούν, παρουσιάζεται ο συνολικός αριθμός των προβλημάτων σχεδίασης "Long Method", "Feature Envy", "State Checking" και "God Class" που εντοπίστηκαν από τα εργαλεία JDeodorant και inCode στα έργα λογισμικού ανοιχτού κώδικα που επιλέχτηκαν προς ανάλυση στην παρούσα εμπειρική μελέτη.

Διακρίνοντας τα σχήματα, γίνεται εύκολα αντιληπτό ότι, ο συνολικός αριθμός των προβλημάτων σχεδίασης "Long Method" και για τα δυο έργα λογισμικού είναι σημαντικά μεγαλύτερος από το συνολικό αριθμό των υπολοίπων προβλημάτων. Συνεπώς, μπορούμε να συμπεράνουμε ότι οι μέθοδοι με μεγάλο μέγεθος, υψηλή πολυπλοκότητα και χαμηλή συνεκτικότητα είναι το πιο κοινό πρόβλημα των σχεδιαστών που αναπτύσσουν ή συντηρούν λογισμικό μεγάλης κλίμακας.

Επίσης, όλα τα προβλήματα σχεδίασης που εξετάστηκαν, αυξάνονται με την εξέλιξη και των δυο συστημάτων λογισμικού πράγμα που επιβεβαιώνει ότι τα έργα λογισμικού ανοιχτού κώδικα δεν υφίστανται συστηματική συντήρηση.



Σχήμα 5.1: Συνολικός αριθμός προβλημάτων σχεδίασης για το έργο JFlex



Σχήμα 5.2: Συνολικός αριθμός προβλημάτων σχεδίασης για το έργο JFreeChart

5.2 Παραμονή Προβλημάτων Σχεδίασης

Για να αποκτήσουμε μια πρώτη εικόνα του τρόπου με τον οποίο τα προβλήματα σχεδίασης αναπτύσσονται κατά τη διάρκεια εξέλιξης ενός συστήματος λογισμικού, το ακόλουθο σχήμα απεικονίζει τα προβλήματα "Long Method" που εμφανίζονται, διατηρούνται ή εξαλείφονται σε κάθε διαδοχική γενιά του έργου JFlex.

Κάθε οριζόντια μπάρα, με χρώμα γεμίσματος γκρι, αντιστοιχεί σε ένα πρόβλημα σχεδίασης το οποίο εντοπίστηκε από το εργαλείο jDeodorant. Παρατηρώντας τη κάθε γκρι μπάρα του σχήματος εύκολα διακρίνεται η γενιά του έργου στην οποία το πρόβλημα εμφανίστηκε ή εξαλείφθηκε αντίστοιχα. Η δεξιά κατακόρυφη διακεκομμένη γραμμή του σχήματος αντιστοιχεί σε μια υποθετική επερχόμενη γενιά του έργου που θα ακολουθεί τη τελευταία εξεταζόμενη. Συνεπώς, κάθε γενιά του έργου αναπαρίσταται ως το διάστημα που μεσολαβεί για τη δημιουργία της επόμενης γενιάς.

Από το συγκεκριμένο σχήμα είναι εμφανές ότι η πλειονότητα των προβλημάτων σχεδίασης (89.8%) από τη στιγμή που εμφανίζονται σε κάποια γενιά του έργου, παραμένουν μέχρι και τη τελευταία εξεταζόμενη. Γεγονός που πιθανόν να υπονοεί ότι τα προβλήματα σχεδίασης δεν εξαλείφονται με την εξέλιξη των έργων λογισμικού παρά μόνο με την εκτέλεση κατάλληλων αναδομήσεων. Δηλαδή, κατόπιν στοχευόμενης ανθρώπινης παρέμβασης. Επίσης, διακρίνεται ότι μεγάλο μέρος των προβλημάτων (57.7%), παραμένουν στο σύστημα λογισμικού σε όλες τις εξεταζόμενες γενιές του.

Αναφέρεται επίσης ότι, ο τρόπος με τον οποίο τα υπόλοιπα προβλήματα σχεδίασης αναπτύσσονται κατά τη διάρκεια εξέλιξης των δυο εξεταζόμενων συστημάτων λογισμικού συνοψίζονται στο Πίνακα 5.1 ο οποίος παρατίθενται στην επόμενη ενότητα.

Σχήμα 5.3: Παραμονή προβλημάτων σχεδίασης "Long Method" για το έργο JFlex

Τέλος σημειώνεται ότι, λόγω των αρκετών περιπτώσεων εξάλειψης ενός προβλήματος σχεδίασης κατά τη μετάβαση από τη μια γενιά ενός έργου λογισμικού στην αμέσως επόμενη, κατόπιν λεπτομερής εξέτασης του πηγαίου κώδικα των

εξεταζόμενων έργων, παρουσιάζονται κατά προσέγγιση οι ακόλουθοι λόγοι που προκαλούν την εξάλειψη των προβλημάτων:

- *Τροποποίηση Μπλοκ Πηγαίου Κώδικα*

  Σε αυτή τη κατηγορία, το μπλοκ του πηγαίου κώδικα το οποίο παρουσίαζε πρόβλημα σχεδίασης στη προηγούμενη γενιά του έργου έχει υποστεί αλλαγές οι οποίες δεν περιλαμβάνουν καμιά ένδειξη που να υποδηλώνει κάποια στοχευόμενη ενέργεια αναδόμησης. Για παράδειγμα, οι περισσότερες περιπτώσεις των έργων λογισμικού που εξετάστηκαν περιλαμβάνουν σύνθετες εκφράσεις συνθηκών ελέγχου που ένα μέρος αυτών επηρεάζει τη τιμή μιας μεταβλητής. Επομένως, διαγράφοντας το σχετικό κομμάτι των συνθηκών (για λόγους που πιθανόν να σχετίζονται με τη συμπεριφορά) εξαλείφεται το αναφερόμενο πρόβλημα "Long Method". Συνεπώς, ενέργειες που κατατάσσονται σε αυτή τη κατηγορία θεωρούνται τυχαίες διαγραφές των προβλημάτων.

- *Διαγραφή Μπλοκ Πηγαίου Κώδικα*

  Η κατηγορία αυτή περιλαμβάνει όλες τις περιπτώσεις όπου το προβληματικό κομμάτι του πηγαίου κώδικα που εμφανιζόταν στη προηγούμενη γενιά του έργου, έχει αφαιρεθεί. Είναι προφανές ότι τέτοιες τροποποιήσεις του πηγαίου κώδικα δεν μπορούν να θεωρηθούν ως σκόπιμες ενέργειες συντήρησης με σκοπό την εξάλειψη του συγκεκριμένου προβλήματος από τη στιγμή που η εξάλειψή του προκλήθηκε λόγω της τροποποίησης της μέχρι τώρα λειτουργικότητας της μεθόδου.

- *Διαγραφή Κλάσης/Μεθόδου*

  Παρόμοιο με την προηγούμενη περίπτωση με τη διαφορά ότι σε αυτή τη κατηγορία ολόκληρη η κλάση ή η μέθοδος που εμφάνιζαν το σχετικό πρόβλημα σχεδίασης στην προηγούμενη γενιά του έργου έχουν διαγραφεί. Επίσης φανερό ότι οι συγκεκριμένες ενέργειες δεν μπορούν να θεωρηθούν ως σκόπιμες ενέργειες συντήρησης με σκοπό την εξάλειψη του συγκεκριμένου προβλήματος.

- *Σκόπιμες Ενέργειες Αναδόμησης*

  Σε αυτή τη κατηγορία κατατάσσονται όλες οι περιπτώσεις που ο πηγαίος κώδικας του συστήματος λογισμικού υφίσταται σκόπιμες ενέργειες επίλυσης των σχετικών προβλημάτων σχεδίασης που εμφάνιζε η αμέσως προηγούμενη γενιά του έργου.
  Με τον όρο σκόπιμες ενέργειες αναδόμησης εννοούμε όλες τις *by-the-book* ενέργειες επίλυσης ενός προβλήματος σχεδίασης. Για παράδειγμα, για την εξάλειψη του προβλήματος "Long Method", μια αναμφίβολη ενέργεια αναδόμησης περιλαμβάνει την εξαγωγή του μπλοκ κώδικα μιας

προβληματικής μεθόδου το οποίο επηρεάζει τη τελική τιμή μιας μεταβλητής ή τη κατάσταση ενός αντικειμένου σε μια νέα ξεχωριστή μέθοδο η οποία και θα καλείται από την αρχική. Για την εξάλειψη του προβλήματος "Feature Envy", μια ένδειξη ενέργειας αναδόμησης αποτελεί η μετακίνηση της μεθόδου μιας κλάσης, στη κλάση της οποίας τα χαρακτηριστικά ζηλεύει ενώ για την εξάλειψη του προβλήματος "State Checking" ένας ξεκάθαρος ειδοποιός είναι η αντικατάσταση όλων των αλυσιδωτών εντολών `if/else` ή εντολών `switch` που υπάρχουν σε διάφορα σημεία του πηγαίου κώδικα και καθορίζουν τη ροή της εκτέλεσης με το μηχανισμό που μας παρέχει ο πολυμορφισμός. Ολοκληρώνοντας, για την εξάλειψη του προβλήματος "God Class" ως σκόπιμη ενέργεια αναδόμησης, μπορεί να θεωρηθεί η μεταφορά των μεθόδων μιας κλάσης που δρουν σε ένα συγκεκριμένο υποσύνολο των μελών των δεδομένων της σε μια νέα κλάση [13].

5.3 Εξέλιξη Προβλημάτων Σχεδίασης

Για τη πλήρη κατανόηση του μηχανισμού με τον οποίο τα προβλήματα σχεδίασης αναπτύσσονται κατά τη διάρκεια εξέλιξης ενός συστήματος λογισμικού ή τους λόγους που προκαλούν την εξάλειψη αυτών, τα Σχήματα 5.4 και 5.5 απεικονίζουν με περισσότερη λεπτομέρεια τον τρόπο με τον οποίο τα προβλήματα εμφανίζονται, διατηρούνται ή εξαλείφονται σε κάθε διαδοχική γενιά των εξεταζόμενων έργων λογισμικού ανοιχτού κώδικα.

Τα επιλεγμένα προς αναζήτηση προβλήματα σχεδίασης της παρούσας εμπειρικής μελέτης ομαδοποιούνται στις ακόλουθες κατηγορίες ο καθορισμός των οποίων γίνεται ξεκάθαρος μέσω των σχετικών σχημάτων που έπονται.

Κατηγορία "A"

Σε αυτή τη κατηγορία κατατάσσονται όλα τα προβλήματα σχεδίασης που εντοπίζονται στη πρώτη εξεταζόμενη γενιά του έργου και ενυπάρχουν μέχρι και τη τελευταία εξεταζόμενη.

Κατηγορία "B"

Προβλήματα τα οποία εμφανίστηκαν σε κάποια εξεταζόμενη γενιά του έργου και παραμένουν μέχρι και τη τελευταία εξεταζόμενη ανήκουν σε αυτή τη κατηγορία.

Σημειώνεται ότι τα συγκεκριμένα προβλήματα σχεδίασης δεν υπήρχαν στη πρώτη εξεταζόμενη γενιά του έργου.

Η κατηγορία "B" είναι δυνατό να υποστεί περαιτέρω αποσύνθεση βάσει των ακόλουθων περιπτώσεων που σχετίζονται με τα αίτια εμφάνισης των αντίστοιχων προβλημάτων:

- Υποκατηγορία "B_1"

  Περιπτώσεις προβλημάτων που εμφανίστηκαν κατά τη διάρκεια εμπλουτισμού της λειτουργικότητας του συστήματος λογισμικού και δεν υπήρχαν όταν η κλάση/μέθοδος στην οποία και εντοπίζονται από τα εργαλεία εισήχθη στο σύστημα.

- Υποκατηγορία "B_2"

  Περιπτώσεις προβλημάτων που ενυπάρχουν ήδη από τη στιγμή της εισαγωγής των αντίστοιχων τμημάτων πηγαίου κώδικα (κλάση/μέθοδο) στο σύστημα.

Κατηγορία "C"

Προβλήματα σχεδίασης τα οποία εντοπίζονται στη πρώτη εξεταζόμενη γενιά του έργου και εξαλείφθηκαν σε κάποια επερχόμενη γενιά του, η οποία δεν είναι και η τελευταία εξεταζόμενη, κατατάσσονται σε αυτή τη κατηγορία.

Βάσει των ακόλουθων περιπτώσεων που σχετίζονται με τα αίτια διαγραφής των αντίστοιχων προβλημάτων, η αναφερόμενη κατηγορία, είναι εφικτό να υποστεί περαιτέρω αποσύνθεση.

- Υποκατηγορία "C_1"

  Περιπτώσεις προβλημάτων που εξαλείφθηκαν από τα αντίστοιχα τμήματα του πηγαίου κώδικα (κλάση/μέθοδος) τα οποία όμως και παραμένουν στο σύστημα λογισμικού. Είναι φανερό ότι η συγκεκριμένη κατηγορία περιλαμβάνει περιπτώσεις προβλημάτων που η εξάλειψή τους, βελτιώνει τη ποιότητα της σχεδίασης του συστήματος.

- Υποκατηγορία "C_2"

  Περιπτώσεις προβλημάτων που ενυπήρχαν ήδη από τη πρώτη εξεταζόμενη γενιά του έργου και εξαλείφθηκαν από το σύστημα λογισμικού λόγω διαγραφής των σχετικών τμημάτων πηγαίου κώδικα (κλάση/μέθοδο) στα οποία και εντοπίζονται από τα εργαλεία.

Προφανώς, τέτοιες περιπτώσεις εξάλειψης των προβλημάτων, δεν μπορούν να θεωρηθούν ως σκόπιμες ενέργειες αναδόμησης (by-the-book) από τη στιγμή που η κλάση/μέθοδος που παρουσιάζει το σχετικό πρόβλημα, διαγράφεται εντελώς από το σύστημα.

Κατηγορία "D"

Προβλήματα σχεδίασης τα οποία εμφανίζονται και εξαλείφονται κατά τη διάρκεια εξέλιξης των ενδιάμεσων γενεών ενός συστήματος λογισμικού οι οποίες δεν αφορούν τη πρώτη και τη τελευταία εξεταζόμενη γενιά του έργου περιλαμβάνονται σε αυτή τη κατηγορία.

Η κατηγορία "D", περιλαμβάνει τις ακόλουθες τέσσερις υποκατηγορίες που σχετίζονται με τα τμήματα του πηγαίου κώδικα (κλάση/μέθοδος) τα οποία και περιλαμβάνουν τα αντίστοιχα εντοπιζόμενα προβλήματα.

- Υποκατηγορία "D_1"

  Περιπτώσεις προβλημάτων που ενυπήρχαν ήδη με την εισαγωγή των σχετικών τμημάτων πηγαίου κώδικα (κλάση/μέθοδο) και εξαλείφθηκαν από το σύστημα λογισμικού λόγω διαγραφής των σχετικών τμημάτων που τα περιέχουν.

- Υποκατηγορία "D_2"

  Περιπτώσεις προβλημάτων που εμφανίστηκαν κατά τη διάρκεια εμπλουτισμού της λειτουργικότητας των σχετικών τμημάτων πηγαίου κώδικα, για παράδειγμα λόγω διορθωτικής ή προσαρμοστικής συντήρηση[3] του έργου, και εξαλείφθηκαν όταν η σχετική κλάση/μέθοδο διαγράφτηκε εξ' ολοκλήρου από το σύστημα λογισμικού.

- Υποκατηγορία "D_3"

  Περιπτώσεις προβλημάτων που ενυπήρχαν ήδη με την εισαγωγή των σχετικών τμημάτων πηγαίου κώδικα (κλάση/μέθοδο) και εξαλείφθηκαν από το σύστημα λογισμικού σε κάποια επερχόμενη γενιά του. Σημειώνεται ότι τα σχετικά τμήματα του πηγαίου κώδικα παραμένουν στο σύστημα παρά την εξάλειψη των προβλημάτων και συνεπώς, η εξάλειψη αυτών βελτιώνουν τη ποιότητα της σχεδίασης του έργου.

[3] Με τον όρο διορθωτική συντήρηση-corrective maintenance, αναφερόμαστε στις ενέργειες με στόχο τη διόρθωση σφαλμάτων που αποκαλύπτονται λόγω της εγκατάστασης και χρήσης του συστήματος λογισμικού που παρήχθη ενώ με τον όρο προσαρμοστική συντήρηση-adaptive maintenance, αναφερόμαστε στις ενέργειες με στόχο τη προσθήκη επιπρόσθετων λειτουργιών του λογισμικού βάσει των νέων απαιτήσεων των πελατών [50].

- Υποκατηγορία "D_4"

  Περιπτώσεις προβλημάτων που εμφανίζονται και εξαλείφονται κατά τη διάρκεια εξέλιξης των σχετικών τμημάτων πηγαίου κώδικα (κλάση/μέθοδο). Είναι λοιπόν λογικό τα σχετικά τμήματα να υπάρχουν στο σύστημα λογισμικού, πριν την εμφάνιση των προβλημάτων και κατόπιν εξαλείψεώς αυτών. Συνεπώς, και αυτές οι περιπτώσεις προβλημάτων αποτέλεσαν αντικείμενο ανθρώπινων παρεμβάσεων.

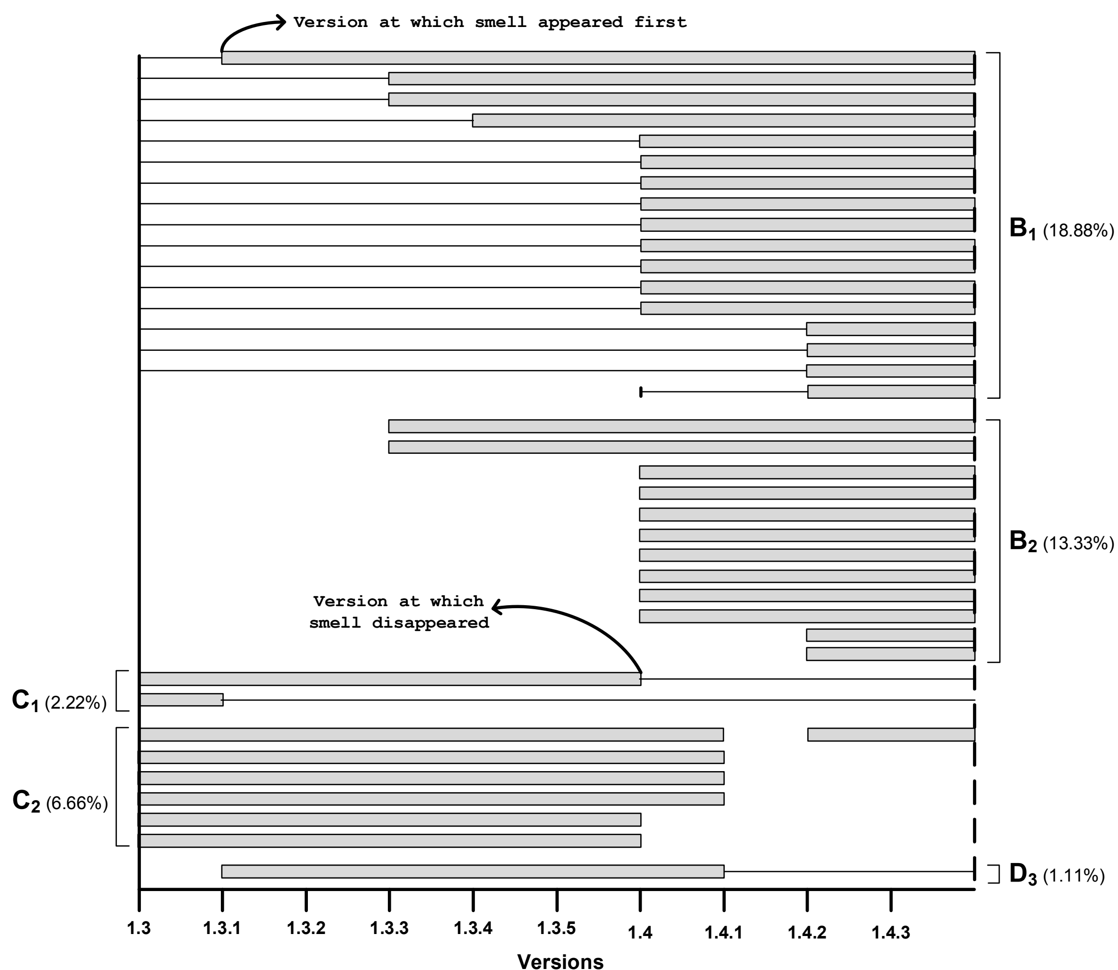
Σημειώνεται ότι, για τη κατανόηση των κατηγοριών που μόλις παρουσιάστηκαν, απεικονίζονται στα Σχήματα 5.4 και 5.5 τα εντοπιζόμενα από το εργαλείο jDeodorant προβλήματα σχεδίασης "Long Method" και για τα δυο έργα λογισμικού ανοιχτού κώδικα που εξετάζονται στην παρούσα μελέτη. Τα υπόλοιπα προβλήματα σχεδίασης συνοψίζονται σε πίνακες.

Το ακόλουθο σχήμα απεικονίζει τα προβλήματα "Long Method" που εμφανίζονται, διατηρούνται ή εξαλείφονται σε κάθε διαδοχική γενιά εξέλιξης του έργου JFlex.

Κάθε εντοπιζόμενο πρόβλημα αναπαρίσταται και πάλι ως οριζόντια μπάρα, με χρώμα γεμίσματος γκρι ώστε να διακρίνεται εύκολα η γενιά του έργου στην οποία το πρόβλημα εμφανίστηκε ή εξαλείφθηκε αντίστοιχα. Η συνεχή γραμμή μπροστά ή πίσω από κάθε μπάρα υποδηλώνει ότι το σχετικό τμήμα πηγαίου κώδικα που περιέχει το πρόβλημα ενυπήρχε στο σύστημα λογισμικού πριν την εμφάνισή του ή παρέμεινε σε αυτό μετά την εξάλειψή του. Στο συγκεκριμένο σχήμα, τα προβλήματα σχεδίασης που ανήκουν στη κατηγορία "A", δηλαδή προβλήματα που παραμένουν σε όλες τις εξεταζόμενες γενιές του έργου λογισμικού, δεν απεικονίζονται. Όλες οι υπόλοιπες κατηγορίες προβλημάτων σχεδίασης που εντοπίστηκαν στο έργο JFlex παρατίθενται στο σχήμα μαζί με τη συχνότητα επανάληψής τους.

Παρατηρώντας το σχήμα, όπως άλλωστε έχει ήδη αναφερθεί είναι φανερό ότι η πλειονότητα των προβλημάτων σχεδίασης, από τη στιγμή που θα εμφανιστούν στο σύστημα λογισμικού, παραμένουν μέχρι και τη τελευταία εξεταζόμενη γενιά του. Τα προβλήματα αυτά αντιστοιχούν στις κατηγορίες "A" (η οποία και δεν αναπαρίσταται στο Σχήμα 5.4) και "B", και αποτελούν περίπου το 90% όλων των περιπτώσεων. Συνεπώς, τα σύνθετα προβλήματα σχεδίασης δεν εξαλείφονται με την εξέλιξη των έργων λογισμικού λόγω παράπλευρης συνέπειας της διορθωτικής ή της

προσαρμοστικής συντήρησης του έργου. Γίνεται επίσης αντιληπτό ότι αρκετές περιπτώσεις προβλημάτων που ανήκουν στις κατηγορίες "C" και "D" και αποτελούν το 10% του συνόλου των περιπτώσεων, εξαλείφονται κατά τη μετάβαση από τη μια γενιά του έργου στην αμέσως επόμενη. Ωστόσο, όπως έχει αναφερθεί, μονάχα οι περιπτώσεις που αντιστοιχούν στις υποκατηγορίες "C_1" και "D_3" και αποτελούν το 3.33% των περιπτώσεων θεωρούνται ως ενέργειες επίλυσης των σχετικών προβλημάτων σχεδίασης από τι στιγμή που οι περιπτώσεις των προβλημάτων που ανήκουν στην υποκατηγορία "C_2" εξαλείφονται μονάχα όταν τα αντίστοιχα τμήματα του πηγαίου κώδικα διαγράφονται εξ' ολοκλήρου από το σύστημα.



Σχήμα 5.4: Εξέλιξη προβλημάτων σχεδίασης "Long Method" για το έργο JFlex

Ολοκληρώνοντας τη παρουσίαση των προβλημάτων "Long Method" που εντοπίστηκαν στο έργο λογισμικού JFlex, αναφέρεται ότι κατόπιν λεπτομερής εξέτασης του πηγαίου κώδικα του έργου, η πλειονότητα των περιπτώσεων "C_1" και "D_3" δεν αποτελούν σκόπιμες ενέργειες επίλυσης των σχετικών προβλημάτων σχεδίασης. Για παράδειγμα, οι σχεδιαστές του έργου, για ένα εντοπιζόμενο πρόβλημα

αυτού του τύπου, δεν εξάγουν το σχετικό μπλοκ πηγαίου κώδικα της μεθόδου που παρουσιάζει το πρόβλημα "Long Method" σε νέα ξεχωριστή μέθοδο που αποτελεί μια από τις by-the-book ενέργειες επίλυσης του συγκεκριμένου προβλήματος σχεδίασης [13].

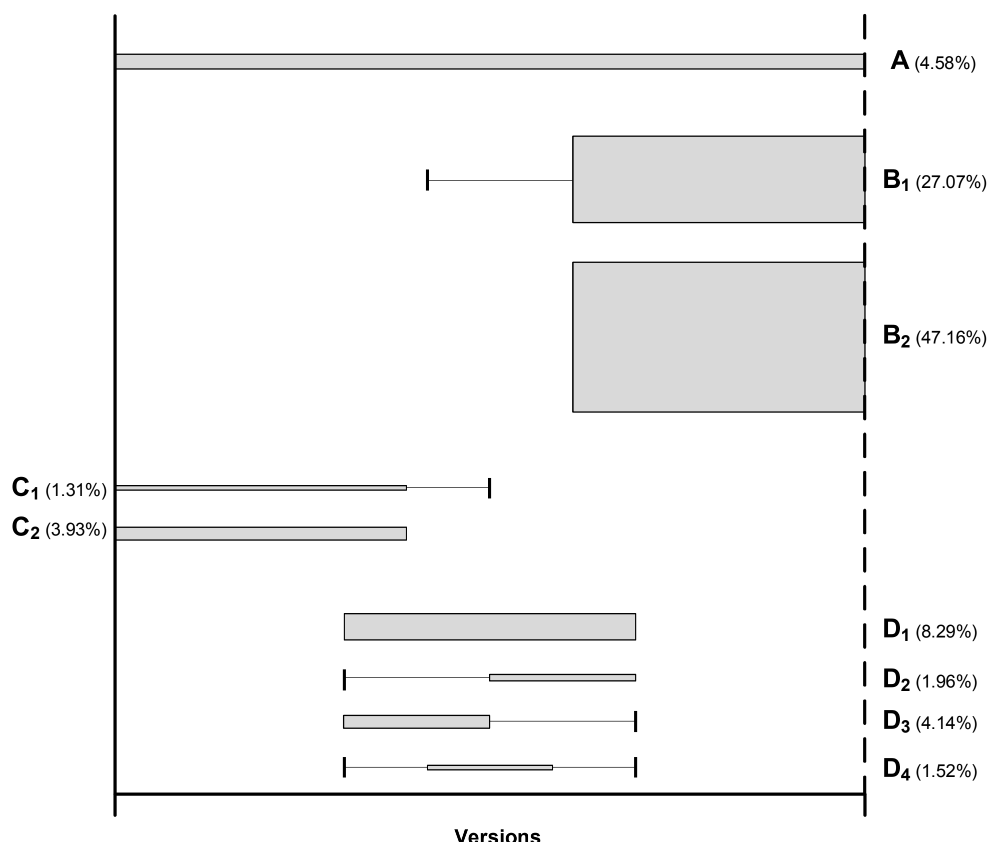
Διευκρινίζεται επίσης ότι, η πρώτη μπάρα της υποκατηγορίας "C_2" εμφανίζεται να διακόπτεται σε μια γενιά του έργου και στη συνέχεια να εμφανίζεται και να διαρκεί μέχρι το τέλος. Ο λόγος οφείλεται στο γεγονός ότι η μέθοδος που εμφανίζει το αντίστοιχο πρόβλημα διαγράφεται από το σύστημα σε εκείνη τη γενιά και εισάγεται εκ νέου, με διαφορετικό όνομα και περισσότερη λειτουργικότητα, στην επόμενη γενιά. Συνεπώς, η συγκεκριμένη περίπτωση είναι δυνατό να περιλαμβάνεται και στην υποκατηγορία "B_2" ωστόσο η έμφαση εδώ δίνεται στη μη εκτέλεση των κατάλληλων ενδεδειγμένων τεχνικών επίλυσης του συγκεκριμένου προβλήματος σχεδίασης.

Για το δεύτερο έργο λογισμικού ανοιχτού κώδικα που εξετάστηκε στα πλαίσια της παρούσας μελέτης, τα εντοπιζόμενα προβλήματα "Long Method" απεικονίζονται στο Σχήμα 5.5. Λόγω του υπερβολικά μεγάλου αριθμού των εντοπιζόμενων προβλημάτων είναι αδύνατο να απεικονιστεί ξεχωριστά κάθε πρόβλημα ως διακριτή οριζόντια μπάρα. Συνεπώς, στο σχήμα απεικονίζονται οι αντίστοιχες κατηγορίες των προβλημάτων σχεδίασης καθώς επίσης και η συχνότητα επανάληψής τους. Το πάχος κάθε μπάρας αντιπροσωπεύει, τη συχνότητα επανάληψης της κάθε κατηγορίας.

Με βάση τα εντοπιζόμενα προβλήματα του δεύτερου εξεταζόμενου έργου, ισχυροποιείται η προηγούμενη παρατήρηση ότι η πλειονότητα των προβλημάτων σχεδίασης από τη στιγμή που εμφανίζονται σε κάποια γενιά του έργου, παραμένουν μέχρι και τη τελευταία εξεταζόμενη. Συνεπώς, προβλήματα που ανήκουν στις κατηγορίες "A" και "B" και αποτελούν το 78.8% του συνόλου των περιπτώσεων για το έργο JFreeChart δηλώνουν ότι ο αριθμός των προβλημάτων σχεδίασης "Long Method" αυξάνεται με τη πάροδο των γενεών ενός συστήματος λογισμικού.

Διακρίνεται επίσης ότι, μεγάλο μέρος των προβλημάτων "Long Method" που ανήκουν στις υποκατηγορίες "B_2", "D_1" και "D_3" και αποτελούν το 59.59% του συνόλου των περιπτώσεων, αντιστοιχούν σε προβλήματα σχεδίασης τα οποία ενυπάρχουν κατά την αρχική εισαγωγή των σχετικών τμημάτων πηγαίου κώδικα στο

σύστημα[4]. Σημειώνεται επίσης ότι και για το έργο JFlex το συγκεκριμένο ποσοστό είναι σημαντικό και περιλαμβάνει το 14.44% των περιπτώσεων. Συνεπώς, εάν η συγκεκριμένη παρατήρηση επιβεβαιωθεί και από άλλες σχετικές μελέτες, σημαίνει ότι τα προβλήματα σχεδίασης δεν είναι μόνο αποτέλεσμα της γήρανσης ενός έργου λογισμικού [37] αλλά και συνέπεια της μη επαρκούς αρχικής ανάλυσης και σχεδίασης του συστήματος.



Σχήμα 5.5: Εξέλιξη προβλημάτων σχεδίασης "Long Method" για το έργο JFreeChart

Παράλληλα, οι περιπτώσεις προβλημάτων σχεδίασης που εξαλείφθηκαν από τα αντίστοιχα τμήματα του πηγαίου κώδικα (κλάση/μέθοδος) τα οποία όμως και παραμένουν στο σύστημα λογισμικού JFreeChart, είναι επίσης περιορισμένα. Συγκεκριμένα, κατόπιν λεπτομερής εξέτασης του πηγαίου κώδικα του έργου, μονάχα τρεις περιπτώσεις προβλημάτων "Long Method" θεωρούνται ως αναμφίβολα σκόπιμες ενέργειες αναδόμησης *Extract Method* για την εξαγωγή του μπλοκ κώδικα που εκτελεί διαφορετική λειτουργικότητα σε νέα ξεχωριστή μέθοδο. Τα προβλήματα αυτά ανήκουν στις υποκατηγορίες "C_1", "D_3" και "D_4". Στις υπόλοιπες υποκατηγορίες

[4] Προβλήματα που ανήκουν στις κατηγορίες "A" και "C" δεν λαμβάνονται υπόψη εφόσον δεν υπάρχουν επακριβή στοιχεία που να δηλώνουν ότι η πρώτη εξεταζόμενη γενιά των έργων λογισμικού, αντιστοιχεί και στη πρώτη γενιά ανάπτυξής τους.

"C_2", "D_1" και "D_2", επίσης περιορισμένες, τα προβλήματα σχεδίασης εξαλείφονται με την εξ' ολοκλήρου διαγραφή των αντίστοιχων τμημάτων πηγαίου κώδικα από το σύστημα.

Ο ακόλουθος πίνακας συνοψίζει όλες τις κατηγορίες των εντοπιζόμενων προβλημάτων σχεδίασης από τα εργαλεία jDeodorant και inCode για τα δυο έργα λογισμικού ανοιχτού κώδικα.

Πίνακας 5.1
Εντοπιζόμενες περιπτώσεις προβλημάτων σχεδίασης

Κατηγορίες Προβλημάτων		***JFlex***				***JFreeChart***			
		Long Method	***Feature Envy***	***State Checking***	***God Class***	***Long Method***	***Feature Envy***	***State Checking***	***God Class***
A		52	8	3	2	21	-	7	-
		57.77%	36.36%	60.00%	50.00%	4.58%	-	14.00%	-
B	**B_1**	17	3	1	2	124	5	5	5
		18.88%	13.63%	20.00%	50.00%	27.07%	21.73%	10.00%	41.66%
	B_2	12	5	1	-	216	3	23	2
		13.33%	22.72%	20.00%	-	47.16%	13.04%	46.00%	16.66%
C	**C_1**	2	5	-	-	6	-	3	2
		2.22%	22.72%	-	-	1.31%	-	6.00%	16.66%
	C_2	6	1	-	-	18	1	5	-
		6.66%	4.54%	-	-	3.93%	4.34%	10.00%	-
D	**D_1**	-	-	-	-	38	1	6	1
		-	-	-	-	8.29%	4.34%	12.00%	8.33%
	D_2	-	-	-	-	9	2	-	2
		-	-	-	-	1.96%	8.69%	-	16.66%
	D_3	1	-	-	-	19	11	1	-
		1.11%	-	-	-	4.14%	47.82%	2.00%	-
	D_4	-	-	-	-	7	-	-	-
		-	-	-	-	1.52%	-	-	-
Συνολικός Αριθμός Προβλημάτων		90	22	5	4	458	23	50	12

Τα δεδομένα, παρέχονται σε ποσοστό επί τοις εκατό καθώς επίσης και ως επακριβής αριθμός. Άσχετα από τη συχνότητα εμφάνισης των προβλημάτων σχεδίασης, είναι προφανές ότι τα περισσότερα προβλήματα από τη στιγμή που εμφανίζονται σε μια γενιά του έργου, παραμένουν στο σύστημα έως και τη τελευταία εξεταζόμενη γενιά. Σε αντίθεση, οι περιπτώσεις σκόπιμων ή όχι ενεργειών εξάλειψης των προβλημάτων σχεδίασης από το σύστημα, δίχως να διαγράφονται εξ' ολοκλήρου από αυτό τα αντίστοιχα τμήματα του πηγαίου κώδικα, δηλαδή τα προβλήματα των υποκατηγοριών "C_1", "D_3" και "D_4" είναι σημαντικά μικρού ποσοστού και τονίζεται ότι περιλαμβάνουν μονάχα το 13.19% του συνόλου των περιπτώσεων. Επίσης, αναφέρεται ότι οι υποκατηγορίες των προβλημάτων "B_2" "D_1" και "D_3", προβλήματα δηλαδή που ενυπάρχουν κατά την αρχική εισαγωγή των σχετικών τμημάτων πηγαίου κώδικα στο σύστημα, αποτελούν το 33.38% του συνόλου των περιπτώσεων.

Είναι φανερό ότι ο μέσος χρόνος παραμονής ενός προβλήματος σχεδίασης στο σύστημα, για παράδειγμα, ένα συγκεκριμένο πρόβλημα, σε πόσες εξεταζόμενες γενιές του έργου εμφανίζεται, είναι εφικτό να υπολογιστεί καθώς καθορίζεται από το λόγο των γενεών του συστήματος λογισμικού που εμφανίζεται προς το συνολικό αριθμό των εξεταζόμενων γενεών του.

Ο ακόλουθος πίνακας παρουσιάζει το μέσο χρόνο παραμονής των προβλημάτων "Long Method", "Feature Envy", "State Checking", και "God Class" στα δυο έργα λογισμικού που εξετάστηκαν. Σημειώνεται ότι, το ποσοστό 100% υποδηλώνει ότι ο μέσος χρόνος παραμονής του συγκεκριμένου τύπου προβλήματος εμφανίζεται σε όλες τις εξεταζόμενες γενιές του συστήματος-ιδανική περίπτωση από τη σκοπιά των ίδιων των προβλημάτων σχεδίασης.

Πίνακας 5.2
Μέσος χρόνος παραμονής προβλημάτων σχεδίασης

JFlex				*JFreeChart*			
Long Method	*Feature Envy*	*State Checking*	*God Class*	*Long Method*	*Feature Envy*	*State Checking*	*God Class*
77%	68%	68%	77%	40%	28%	57%	35%

Το γεγονός ότι μερικά προβλήματα σχεδίασης, όπως για παράδειγμα τα προβλήματα "Long Method", είναι τα περισσότερο κοινά πρόβλημα των σχεδιαστών που αναπτύσσουν ή συντηρούν λογισμικό μεγάλης κλίμακας, όπως έχει ήδη αναφερθεί, σε συνδυασμό με το υψηλό ποσοστό εμφάνισής τους, μπορούν να χρησιμοποιηθούν ως ειδοποιός από τις ομάδες σχεδιαστών που αναπτύσσουν λογισμικό ώστε να είναι περισσότερο προσεκτικοί.

Ο Πίνακας 5.3 περιλαμβάνει όλες τις εντοπιζόμενες ενέργειες αναδόμησης που στοχεύουν στην εξάλειψη των προβλημάτων σχεδίασης συνολικά για όλους τους τύπους των προβλημάτων και για τα δυο έργα λογισμικού, JFlex και JFreeChart, που εξετάστηκαν. Σύμφωνα με τα δεδομένα που συλλέχθηκαν, οι σχεδιαστές των εξεταζόμενων έργων δεν εκτελούν ενέργειες αναδόμησης με σκοπό την εξάλειψη των σχετικών προβλημάτων σχεδίασης. Από τις 664 συνολικά περιπτώσεις προβλημάτων σχεδίασης που εντοπίστηκαν, μονάχα σε 5 από αυτές εκτελέστηκαν στοχευόμενες ενέργειες για την εξάλειψη των αντίστοιχων προβλημάτων.

Παρόλο που άλλες μελέτες [34] καταλήγουν ότι οι αναδομήσεις πραγματοποιούνται συχνά στο κύκλο ζωής ενός προϊόντος λογισμικού, τα ευρήματα

της παρούσας μελέτης, πιθανόν να υπονοούν ότι οι σχεδιαστές, εκτελούν τις κατάλληλες τεχνικές επίλυσης των προβλημάτων σχεδίασης βασιζόμενοι στη δική τους υποκειμενική αντίληψη επιλογής των περιοχών του πηγαίου κώδικα που πρέπει να εφαρμοστούν. Πράγμα που μπορεί επίσης να σχετίζεται και με την έλλειψη εργαλείων που υποστηρίζουν τον εντοπισμό σύνθετων προβλημάτων σχεδίασης.

Πίνακας 5.3
Σκόπιμες ενέργειες επίλυσης εντοπιζόμενων προβλημάτων σχεδίασης

JFlex				*JFreeChart*			
Long Method	*Feature Envy*	*State Checking*	*God Class*	*Long Method*	*Feature Envy*	*State Checking*	*God Class*
0	1	0	0	3	1	0	0
0.00%	4.52%	0.00%	0.00%	0.65%	4.34%	0.00%	0.00%

Ολοκληρώνοντας, αναφέρεται ότι τα αποτελέσματα που σχετίζονται με όλους τους τύπος των προβλήματα σχεδίασης που επιλέχθηκαν προς αναζήτηση για όλες τις γενιές και των δυο έργων λογισμικού ανοιχτού κώδικα που εξετάστηκαν στα πλαίσια της παρούσας εμπειρικής μελέτης είναι ελεύθερα διαθέσιμα [1].

5.4 Ενεργά Προβλήματα Σχεδίασης

Γενικά, μια λογική ανησυχία που σχετίζεται με την ανάπτυξης οποιασδήποτε μεθοδολογίας που στοχεύει στον εντοπισμό ενός συγκεκριμένου τύπου προβλημάτων σχεδίασης είναι εάν τα εντοπιζόμενα προβλήματα δεν θεωρούνται αρκετά σημαντικά για τους σχεδιαστές των αντίστοιχων έργων λογισμικού που εφαρμόζεται. Συνεπώς, δεν αποτελεί ένδειξη αιφνιδιασμού η περίπτωση της μη εκτέλεσης των κατάλληλων ενδεδειγμένων τεχνικών επίλυσης των σχετικών προβλημάτων σχεδίασης που προτείνει η μεθοδολογία. Για παράδειγμα, γιατί να βελτιωθεί η ποιότητα της σχεδίασης μιας μεθόδου που εμφανίζει το πρόβλημα "Long Method" από τη στιγμή που δεν αποτέλεσε ποτέ αντικείμενο συντήρησης σε όλη τη διάρκεια ζωής ενός προϊόντος λογισμικού; Το πρόβλημα ασφαλώς και υπάρχει, ωστόσο, από το σύνολο των ευκαιριών για αναδόμηση, η πρόταση που αφορά μπλοκ πηγαίου κώδικα το οποίο και δεν τροποποιήθηκε ποτέ στο παρελθόν, πιθανόν να βαθμολογείται χαμηλά στην ιεραρχία των αναδομήσεων με τη λογική ότι δεν επείγει η επίλυσή του συγκεκριμένου προβλήματος.

Μια από τις εναλλακτικές προσεγγίσεις για την εξαγωγή συμπερασμάτων που σχετίζονται με την εύρεση των προβλημάτων που βρίσκονται στην κορυφή της ιεραρχίας των αναδομήσεων και η επίλυσή τους κρίνεται επιτακτική είναι η εξέταση των ιστορικών γενεών του πηγαίου κώδικα ενός έργου λογισμικού. Η φιλοσοφία βασίζεται στην υπόθεση, η οποία φυσικά και δεν ισχύει πάντοτε, ότι το μπλοκ του πηγαίου κώδικα το οποίο αποτέλεσε αντικείμενο εργασιών συντήρησης στο παρελθόν, είναι πολύ πιθανό να παρουσιάσει αλλαγές στην ερχόμενη γενιά του έργου και συνεπώς η βελτίωση της ποιότητας σχεδίασης του αντίστοιχου μπλοκ κώδικα θα πρέπει να είναι υψηλής προτεραιότητας των σχεδιαστών του έργου. Αντίθετα, σε περίπτωση που ένα μπλοκ πηγαίου κώδικα που παρουσιάζει ένα πρόβλημα σχεδίασης, παραμένει δίχως τροποποίηση σε αρκετές γενιές του έργου, τότε η εφαρμογή της κατάλληλης αναδόμησης για την εξάλειψή του, δεν αποτελεί υψηλή προτεραιότητα των σχεδιαστών.

Για την διερεύνηση του αναφερόμενου θέματος, χρησιμοποιείται ο όρος *ενεργό πρόβλημα* για την αναφορά σε προβλήματα σχεδίασης των οποίων τα αντίστοιχα τμήματα του πηγαίου κώδικα που τα περιέχουν, αποτέλεσαν αντικείμενο συντήρησης, τουλάχιστον μια φορά στο παρελθόν. Ο καθορισμός του όρου, προέρχεται από το χώρο της ηφαιστειολογίας όπου σύμφωνα με ορισμένους ερευνητές, ένα ηφαίστειο θεωρείται ενεργό σε περίπτωση που έχει καταγραφεί κάποια έκρηξή του κατά τη διάρκεια των ιστορικών του χρόνων. Σημειώνεται επίσης ότι, στη περίπτωση που ο στόχος της μελέτης ήταν η ταξινόμηση των προβλημάτων σχεδίασης στην ιεραρχία των αναδομήσεων, θα μπορούσε να χρησιμοποιηθεί μια περισσότερο εξελιγμένη προσέγγιση, αξιολογώντας για παράδειγμα τα ευρήματα των αλλαγών του πρόσφατου παρελθόντος με τη τρέχουσα γενιά του συστήματος λογισμικού [15].

Όσον αφορά τα προβλήματα σχεδίασης "Long Method", η ύπαρξη τους υπονοεί ότι απαιτείται περισσότερη προσπάθεια και χρόνος των σχεδιαστών να εκτελέσουν εργασίες συντήρησης στις μεθόδους που παρουσιάζουν το συγκεκριμένο πρόβλημα. Αξίζει λοιπόν το κόπο να εφαρμόσουμε τις κατάλληλες ενδεδειγμένες τεχνικές επίλυσης για την εξάλειψη του αναφερόμενου προβλήματος που παρουσιάζει μια μέθοδος στη περίπτωση που αναμένουμε να αποτελέσει αντικείμενο συντήρησης στις επερχόμενες γενιές του συστήματος λογισμικού. Συνεπώς, θα πρέπει να εξεταστούν οι προηγούμενες γενιές του έργου ώστε να εντοπιστεί εάν έχουν εφαρμοστεί αλλαγές

στην αντίστοιχη μέθοδο. Η εισαγωγή ενός νέου statement, η τροποποίηση ή η διαγραφή ενός υπάρχοντος statement θεωρούνται ως αλλαγές της μεθόδου μεταξύ δυο διαδοχικών γενεών του έργου. Σε περίπτωση που εντοπιστεί έστω και μια από τις τρεις περιπτώσεις των αλλαγών που αναφέρθηκαν, καταγράφεται η ύπαρξη της αλλαγής και οποιοδήποτε "Long Method" πρόβλημα που σχετίζεται με την αναφερόμενη μέθοδο, χαρακτηρίζεται ως ενεργό.

Τα προβλήματα σχεδίασης "Feature Envy", στα πλαίσια της παρούσας μελέτης, σχετίζονται με τη πρόσβαση ιδιοτήτων και μεθόδων που βρίσκονται σε διαφορετική κλάση. Περιπτώσεις που ο συνολικός αριθμός της πρόσβασης των χαρακτηριστικών της διαφορετικής κλάσης δεν μεταβάλλεται κατά τη διάρκεια εξέλιξης των γενεών ενός συστήματος λογισμικού, η επίλυση του συγκεκριμένου προβλήματος δεν κρίνεται τόσο επιτακτική όσο των περιπτώσεων που ο αριθμός της πρόσβασης αυξάνεται. Συνεπώς, ένα πρόβλημα "Feature Envy" χαρακτηρίζεται ως ενεργό εάν για μια μέθοδο, ο συνολικός αριθμός πρόσβασης προς τα χαρακτηριστικά μιας άλλης κλάσης μεταβάλλεται τουλάχιστον σε μια γενιά του έργου.

Τα προβλήματα "State Checking" δηλώνουν την απουσία του μηχανισμού που μας παρέχει ο πολυμορφισμός. Ωστόσο, ο πολυμορφισμός έχει νόημα εφαρμογής μονάχα στις περιπτώσεις που τα αντίστοιχα τμήματα του πηγαίου κώδικα πρόκειται να εξελιχθούν λόγω αλλαγών στις απαιτήσεις, διαφορετικά προσθέτει στο σύστημα περιττή πολυπλοκότητα [31]. Συνεπώς, ένα πρόβλημα σχεδίασης "State Checking" χαρακτηρίζεται ως ενεργό σε περίπτωση που καταγραφεί έστω και μια ύπαρξη των αλλαγών που έπονται σε μια τουλάχιστον εξεταζόμενη γενιά του συστήματος λογισμικού: α) προσθήκη επιπρόσθετου ελέγχου στο σύνολο των εντολών συνθήκης `if/else` ή εντολών `switch` που συνθέτουν ένα εντοπιζόμενο πρόβλημα "State Checking". Περιπτώσεις αλλαγών αυτού του είδους, αντιστοιχούν στη τροποποίηση των αρμοδιοτήτων (axis of change) του συστήματος και υποδηλώνουν την προσθήκη νέας υποκλάσης στην ιεραρχία της κληρονομικότητας σε περίπτωση εφαρμογής της αναδόμησης του συγκεκριμένου προβλήματος, β) τροποποίηση του αριθμού των φορών που λαμβάνει χώρα στο σύστημα λογισμικού ο έλεγχος ροής που συνθέτει ένα εντοπιζόμενο πρόβλημα-"State Checking" occurrences. Δηλαδή, πόσες φορές θα αξιοποιηθεί ο μηχανισμός του πολυμορφισμού σε όλο το σύστημα σε περίπτωση επίλυσης του σχετικού προβλήματος, και γ) τροποποίηση του αριθμού των statements που περιέχουν τα τμήματα εκτέλεσης του πηγαίου κώδικα των εντολών συνθήκης

`if/else` ή εντολών `switch` που απαρτίζουν ένα εντοπιζόμενο πρόβλημα "State Checking". Περιπτώσεις αλλαγών αυτού του είδους, σχετίζονται με τα κομμάτια του πηγαίου κώδικα που θα μεταφερθούν στις αντίστοιχες υποκλάσεις της ιεραρχίας του δέντρου της κληρονομικότητας που θα δημιουργηθεί με την εξάλειψη του συγκεκριμένου προβλήματος από το σύστημα.

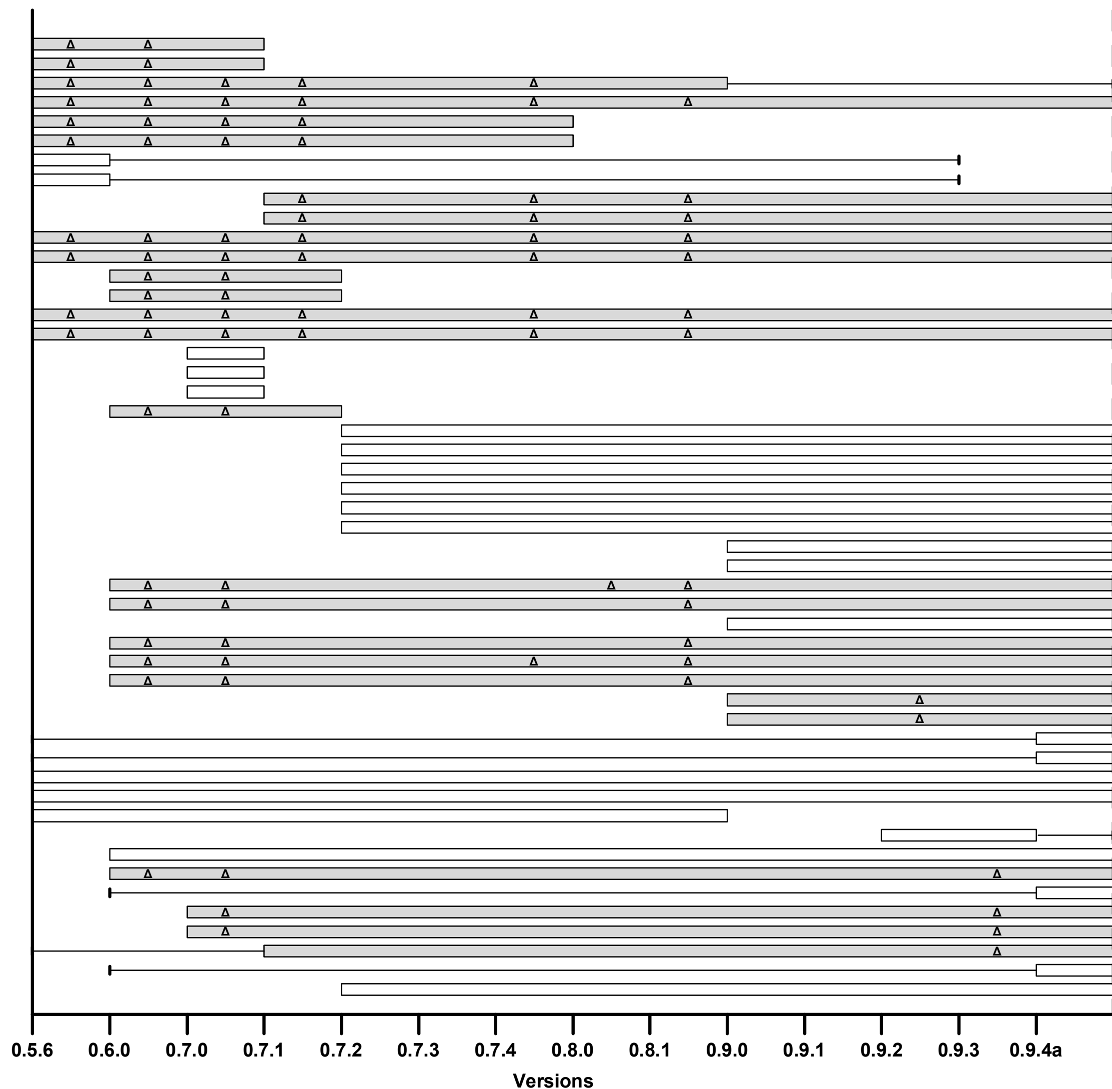
Τα προβλήματα σχεδίασης "God Class" υποδηλώνουν συνήθως υπερβολικά αναπτυγμένες κλάσεις που ενσωματώνουν μεγάλο μέρος της λογικής του συστήματος. Η δημιουργία των θεϊκών κλάσεων πραγματοποιείται δημιουργώντας κλάσεις προσθέτοντας σε αυτές υπερβολική συμπεριφορά ή αρκετά δεδομένα του συστήματος αφήνοντας ένα μικρό μέρος λεπτομερειακών εργασιών στις υπόλοιπες στοιχειώδες κλάσεις [40]. Συνεπώς, περιπτώσεις που μεταβάλλεται ο συνολικός αριθμός είτε των μεθόδων είτε των ιδιοτήτων μιας θεϊκής κλάσης σε μια τουλάχιστον εξεταζόμενη γενιά του συστήματος λογισμικού, τότε το αντίστοιχο πρόβλημα "God Class" χαρακτηρίζεται ως ενεργό.

Το Σχήμα 5.6 απεικονίζει τα εντοπιζόμενα προβλήματα "State Checking" του έργου JFreeChart και η ύπαρξη των ενεργών προβλημάτων σχεδίασης του συνόλου δηλώνεται με κάθε γκρι οριζόντια μπάρα. Επιπλέον, η γενιά στην οποία τα αντίστοιχα τμήματα του πηγαίου κώδικα αποτέλεσαν αντικείμενο συντήρησης δηλώνεται με το Ελληνικό Δέλτα (Δ)[5]. Το σύμβολο τοποθετείται ανάμεσα σε δυο διαδοχικές γενιές του έργου από τη στιγμή που η αλλαγή προκλήθηκε κατά τη μετάβαση από τη μια γενιά του έργου στην αμέσως επόμενη.

Όπως είναι φανερό, αρκετά προβλήματα "State Checking" χαρακτηρίζονται ως *ενεργά προβλήματα* πράγμα που σημαίνει ότι κατά τη διάρκεια εξέλιξης των γενεών του συστήματος λογισμικού JFreeChart, ένα ή περισσότερα κομμάτια του πηγαίου κώδικα που σχετίζονται με την απουσία του μηχανισμού που μας παρέχει ο πολυμορφισμός αποτέλεσαν αντικείμενο συντήρησης. Τα ιστορικά δεδομένα δηλώνουν ξεκάθαρα ότι αρκετές ευκαιρίες αναδόμησης είναι σημαντικές και σε περίπτωση υλοποίησης της κατάλληλης τεχνικής επίλυσης των σχετικών προβλημάτων καμιά από τις καταγεγραμμένες περιπτώσεις αλλαγών δεν θα επηρέαζαν τον υπάρχοντα πηγαίο κώδικα. Ο χρόνος του εντοπισμού και της

[5] Σύμφωνα με επίσημες προσεγγίσεις, το Ελληνικό Δέλτα (Δ) αντιστοιχεί στην ιδέα ότι πραγματοποιήθηκε αλλαγή [16].

κατανόησης όλων των σχετικών σημείων του κώδικα που πραγματοποιούνταν ο έλεγχος ροής για την εφαρμογή των απαιτούμενων αλλαγών θα μειωνόταν και θα περιοριζόταν σημαντικά η πιθανότητα εισαγωγής πολλαπλών σφαλμάτων. Επίσης τονίζεται και πάλι ότι τα προβλήματα που δεν χαρακτηρίζονται ως ενεργά, σύμφωνα με τη μεθοδολογία εντοπισμού τους, παραμένουν προβλήματα σχεδίασης ωστόσο η εξάλειψή τους δεν θεωρείται επιτακτική με βάση τις ιστορικές αλλαγές.



Σχήμα 5.6: Ενεργά προβλήματα σχεδίασης "State Checking" για το έργο JFreeChart

Ο ακόλουθος πίνακας παρουσιάζει τον ακριβή αριθμό καθώς επίσης και το ποσοστό επί τοις εκατό (%) των ενεργών προβλημάτων σχεδίασης όλων των τύπων προβλημάτων που επιλέχθηκαν προς αναζήτηση και για τα δυο έργα λογισμικού ανοιχτού κώδικα που εξετάστηκαν στην παρούσα μελέτη. Με βάση τα αποτελέσματα και υιοθετώντας τη σημαντικότητα των αλλαγών του παρελθόντος που υπέστη ο πηγαίος κώδικας του έργου, ένα μικρό ποσοστό των εντοπιζόμενων προβλημάτων

σχεδίασης είναι αρκετά ανησυχητικό. Τα προβλήματα "Long Method" για άλλη μια φορά καταλαμβάνουν το υψηλό ποσοστό. Σε συνδυασμό λοιπόν με τον υψηλό συνολικό αριθμό που παρουσιάζει ο συγκεκριμένος τύπος προβλημάτων αλλά και τη μεγαλύτερη παραμονή τους στο σύστημα λογισμικού σε σχέση με τα υπόλοιπα προβλήματα σχεδίασης που μελετούνται, συμπεραίνεται ότι οι σχεδιαστές έργων λογισμικού μεγάλης κλίμακας θα πρέπει να δίνουν προτεραιότητα συντήρησης σε αυτή τη κατηγορία προβλημάτων.

Πίνακας 5.4
Ενεργά προβλήματα σχεδίασης

JFlex				***JFreeChart***			
Long Method	***Feature Envy***	***State Checking***	***God Class***	***Long Method***	***Feature Envy***	***State Checking***	***God Class***
53	6	1	3	285	0	26	7
58.89%	27.27%	20.00%	75.00%	62.23%	0.00%	52.00%	58.33%

ΚΕΦΑΛΑΙΟ 6

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΠΕΚΤΑΣΗ

Περιεχόμενα

6.1 Συμπεράσματα Αποτελεσμάτων 56
6.2 Απειλές Εγκυρότητας Αποτελεσμάτων 57
6.3 Μελλοντική Επέκταση Εργασίας 57

*"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:
A Scandal in Bohemia, 1891]

6.1 Συμπεράσματα Αποτελεσμάτων

Στη παρούσα διπλωματική εργασία, παρουσιάστηκαν τα αποτελέσματα της εμπειρικής μελέτης που σχετίζονται με την εξέλιξη των προβλημάτων σχεδίασης "Long Method", "Feature Envy", "State Checking" και "God Class" σε δυο έργα λογισμικού ανοιχτού κώδικα υλοποιημένα σε Java, το JFlex και το JFreeChart.

Τα προβλήματα σχεδίασης που επιλέχθηκαν προς αναζήτηση στις παρελθούσες γενιές των δυο έργων λογισμικού, θεωρούνται τα περισσότερο σημαντικά μεταξύ των προβλημάτων που εμφανίζονται σε έργα λογισμικού μεγάλης κλίμακας εφόσον σχετίζονται με τη διανομή των μεθόδων και των κλάσεων του συστήματος λογισμικού. Παράλληλα, η εξάλειψη των αναφερόμενων προβλημάτων σχεδίασης επιτυγχάνεται κατόπιν στοχευόμενων ενεργειών από πλευράς των σχεδιαστών πράγμα που διαχωρίζει σαφώς τις σκόπιμες ενέργειες αναδόμησης από τις ακούσιες ενέργειες αφαίρεσης των προβλημάτων που προκύπτουν λόγω διορθωτικής ή προσαρμοστικής συντήρησης του συστήματος. Τέλος σημειώνεται ότι ο εντοπισμός τους παρέχεται αυτόματα από τα εργαλεία jDeodorant και inCode που αξιοποιήθηκαν στη παρούσα μελέτη.

Τα αποτελέσματα που προέκυψαν από τη μελέτη που πραγματοποιήθηκε δηλώνουν ξεκάθαρα ότι:

- *Η πλειονότητα των προβλημάτων σχεδίασης από τη στιγμή που εμφανίζονται σε κάποια γενιά του έργου, παραμένουν μέχρι και τη τελευταία εξεταζόμενη και συνεπώς τα προβλήματα συσσωρεύονται καθώς το έργο λογισμικού ωριμάζει.*

- *Τα προβλήματα "Long Method" είναι το πιο συχνό πρόβλημα των σχεδιαστών που αναπτύσσουν ή συντηρούν λογισμικό μεγάλης κλίμακας.*

- *Ένα σημαντικό ποσοστό των προβλημάτων σχεδίασης ενυπάρχουν κατά την αρχική εισαγωγή των σχετικών τμημάτων πηγαίου κώδικα στο σύστημα λογισμικού, και*

- *Η εξάλειψη μερικών προβλημάτων σχεδίασης κατά τη μετάβαση από τη μια γενιά του έργου στην αμέσως επόμενη δεν οφείλεται σε στοχευόμενες ενέργειες επίλυσης των αντίστοιχων προβλημάτων αλλά είναι αποτέλεσμα παράπλευρης συνέπειας ενεργειών προσαρμοστικής συντήρησης του έργου.*

6.2 Απειλές Εγκυρότητας Αποτελεσμάτων

Τα αποτελέσματα της παρούσας εμπειρικής μελέτης προέκυψαν από την ανάλυση δυο εξεταζόμενων έργων λογισμικού ανοιχτού κώδικα, το JFlex και το JFreeChart, και ο εντοπισμός των προβλημάτων σχεδίασης στα αναφερθέντα έργα λογισμικού επικεντρώθηκε σε τέσσερις σύνθετους τύπους προβλημάτων σχεδίασης. Συνεπώς, όπως είναι φανερό, τα αποτελέσματα της παρούσας μελέτης δεν μπορούν να γενικευτούν.

Επιπλέον, δυο ακόμα απειλές που μπορεί να δεχθεί η παρούσα μελέτη για την εγκυρότητα των αποτελεσμάτων που προέκυψαν σχετίζονται με τις μεθοδολογίες αυτόματου εντοπισμού των προβλημάτων σχεδίασης που βασίζονται τα εργαλεία jDeodorant και inCode που αξιοποιήθηκαν. Είναι πολύ πιθανό, τα εργαλεία να εντόπισαν προβλήματα σχεδίασης για τα οποία ένας ειδικός δεν θα συμφωνούσε και θα απέρριπτε τη πρόταση του εργαλείου. Για παράδειγμα, προβλήματα που δεν θεωρούνται ως πραγματικά προβλήματα σχεδίασης-false positive. Επίσης, υπάρχει και η περίπτωση, να υπήρχαν πραγματικά προβλήματα σχεδίασης τα οποία και δεν εντοπίστηκαν από τα αντίστοιχα εργαλεία λόγω διαφορετικής προσέγγισης της μεθοδολογίας εντοπισμού-false negative.

6.3 Μελλοντική Επέκταση Εργασίας

Για την περαιτέρω διερεύνηση της παρούσας μελέτης, μεγάλο ενδιαφέρον θα παρουσίαζε η σύγκριση των εντοπιζόμενων προβλημάτων σχεδίασης που προέκυψαν με τα αποτελέσματα των εργαλείων που υποστηρίζουν τον αυτόματο εντοπισμό αναδομήσεων που έχουν εφαρμοστεί στο πηγαίο κώδικα σε διάφορες γενιές ενός συστήματος λογισμικού. Η σύγκριση αναμένεται να αναδείξει με τον πλέον κατηγορηματικό τρόπο, εάν υπάρχει ή όχι συσχέτιση μεταξύ των αναδομήσεων που εκτελούσαν οι σχεδιαστές των δυο έργων λογισμικού που εξετάστηκαν και των εντοπιζόμενων προβλημάτων σχεδίασης.

ΑΝΑΦΟΡΕΣ

[1] Code Smells Evolution Results, http://java.uom.gr/~amanakos/CodeSmellsResults.rar.

[2] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "*Comparison and Evolution of Clone Detection Tools*," IEEE Transactions on Software Engineering, vol. 33, no. 9, pp. 577-591, September 2007.

[3] Borland Together, http://www.borland.com/together/, 2010.

[4] 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.

[5] 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.

[6] H. M. Deitel, and P. J. Deitel, *Java How to Programm (6th Edition)*, Prentice Hall, 1997.

[7] S. Demeyer, S. Ducasse, and O. Nierstrasz, "*Finding Refactorings via Change Metrics*," 15th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '2000), Minneapolis, USA, October 2000, pp. 166-177.

[8] D. Dig, C. Comertoglu, D. Marinok, and R. Johnson, "*Automated Detection of Refactorings in Evolving Components*," 20th European Conference on Object-Oriented Programming (ECOOP '06), Nantes, France, July 2006, pp. 404-428.

[9] M. Di Penta, L. Cerulo, and L. Aversano, "*The Life and Death of Statically Detected Vulnerabilities: An Empirical Study*," Information and Software Technology, vol. 51, issue 10, pp. 1469-1484, October 2009.

[10] 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.

[11] A. Eastwood, "*Firm Fires Shots at Legacy Systems*," Computing Canada, vol. 19, no. 2, pp. 17, 1993.

[12] A. Erlikh, "*Leveraging Legacy System Dollars for E-Business*," IEEE IT Professional, vol. 2, no. 3, pp. 17-23, May/June 2000, doi:10.1109/6294.846201.

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

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

[15] T. Gîrba, S. Ducasse, and M. Lanza, "*Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes*," 20th International Conference on Software Maintenance (ICSM '04), Illinois, Chicago, September 2004, pp. 40-49.

[16] D. Heath, D. Allum, and L. Dunckley, *Introductory Logic and Formal Methods*, Alfred Waller Limited, Henley-on-Thames, 1994.

[17] S. Huff, "*Information Systems Maintenance*," The Business Quarterly, vol. 55, pp. 30-32, 1990.

[18] inCode Eclipse plug-in, http://www.intooitus.com/incode, 2010.

[19] JDeodorant Eclipse plug-in, http://jdeodorant.com/, 2010.

[20] JFlex lexical analyzer generator for Java, http://jflex.de/index.html, 2010.

[21] JFreeChart Java chart library, http://www.jfree.org/jfreechart/, 2010.

[22] H. Kagdi, M. L. Collard, and J. I. Maletic, "*A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution*," Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, issue 2, pp. 77-131, March 2007.

[23] 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.

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

[25] F. Khomh, M. Di Penta, and Y.-G. Guéhéneuc, "*An Exploratory Study of the Impact of Code Smells on Software Change-Proneness*," 16th Working Conference on Reverse Engineering (WCRE '09), Lille, France, October 2009, pp. 75-84.

[26] J. Makhoul, F. Kubala, R. Schwartz, R. Weischedel, "*Performance Measures for Information Extraction*," Proc. Defense Advanced Research Projects Agency (DARPA '99), Broadcast News Workshop, February 1999, pp. 249-252.

[27] M. V. Mäntylä, J. Vanhanenand, C. Lassenius, "*Bad Smells-Humans as Code Critics*," 20th International Conference on Software Maintenance (ICSM '04), Illinois, Chicago, September 2004, pp. 399-408.

[28] R. Marinescu, "*Detecting Design Flaws via Metrics in Object-Oriented Systems*," 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS '01), Santa Barbara, California, July/August 2001, pp. 173-182.

[29] R. Marinescu, "*Detection Strategies: Metrics-Based Rules for Detecting Design Flaws*," 20th International Conference on Software Maintenance (ICSM '04), Illinois, Chicago, September 2004, pp. 350-359.

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

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

[32] T. M. Meyers, and D. Binkley, "*An Empirical Study of Slice-Based Cohesion and Coupling Metrics*," ACM Transactions on Software Engineering and Methodology, vol. 17, no. 1, pp. 1-27, December 2007.

[33] N. Moha, Y. G. Guéhéneuc, L. Duchien, and A. F. Le Meur, "*DECOR: A Method for the Specification and Detection of Code and Design Smells*," IEEE Transactions on Software Engineering, vol. 36, no. 1, pp. 20-36, January/February 2010.

[34] E. Murphy-Hill, C. Parnin, and A. P. Black, "*How we Refactor, and How we Know It*," 31th International Conference of Software Engineering (ICSE '09), Vancouver, Canada, May 2009, pp. 287-297.

[35] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "*The Evolution and Impact of Code Smells: A Case Study of Two Open Source Systems*," 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09), Florida, USA, October 2009, pp. 390-400.

[36] W. F. Opdyke, "*Refactoring Object-Oriented Frameworks*," Phd dissertation, University of Illinois at Urbana-Champaign, 1992.

[37] D. L. Parnas, "*Software Aging*," 16th International Conference of Software Engineering (ICSE '94), Sorrento, Italy, May 1994, pp. 279-287.

[38] J. Ratzinger, "*sPACE: Software Project Assessment in the Course of Evolution*," Phd dissertation, Vienna University of Technology, Austria, 2007.

[39] J. Ratzinger, T. Sigmund, and H. C. Gall, "*On the Relation of Refactoring and Software Defects*," International Working Conference on Mining Software Repositories (MSR '08), Laipzig, Germany, May 2008, pp. 35-38.

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

[41] S. Slinger, "*Code Smell Detection in Eclipse*," Master Thesis, Delft University of Technology, German, 2005.

[42] 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.

[43] A. Trifu, and R. Marinescu, "*Diagnosing Design Problems in Object Oriented Systems*," 12th Working Conference on Reverse Engineering (WCRE '05), Pittsburgh, Pennsylvania, November 2005, pp. 155-164.

[44] N. Tsantalis, and A. Chatzigeorgiou, "*Identification of Extract Method Refactoring Opportunities*," 13th European Conference of Software Maintenance and Reengineering (CSMR '09), Kaiserslautern, Germany, March 2009, pp. 199-128.

[45] N. Tsantalis, and A. Chatzigeorgiou, "*Identification of Move Method Refactoring Opportunities*," IEEE Transactions on Software Engineering, vol. 35, no. 3, pp. 347-367, May/June 2009.

[46] N. Tsantalis, and A. Chatzigeorgiou, "*Identification of Refactoring Opportunities Introducing Polymorphism*," Journal of Systems and Software, 2009, doi:10.1016/j.jss2009.09.017.

[47] E. Van Emden, and L. Moonen, "*Java Quality Assurance by Detecting Code Smells*," 9th Working Conference on Reverse Engineering (WCRE '02), Richmond, Virginia, October 2002, pp. 0097.

[48] Z. Xing, and E. Stroulia, "*Refactoring Detection based on UMLDiff Change-Facts Queris*," 13th Working Conference on Reverse Engineering (WCRE '06), Benevento, Italy, October 2006, pp. 263-274.

[49] Z. Xing, and E. Stroulia, "*Refactoring Practice: How it is and How it Should be Supported-An Eclipse Case Study*," 22th International Conference on Software Maintenance (ICSM '06), Philadelphia, Pennsylvania, September 2006, pp. 458-468.

[50] A. N. Χατζηγεωργίου, *Αντικειμενοστρεφής Σχεδίαση: UML, Αρχές, Πρότυπα και Ευρετικοί Κανόνες*, Κλειδάριθμος, 2005.

ΔΗΜΟΣΙΕΥΣΕΙΣ

Alexander Chatzigeorgiou and Anastasios Manakos, "*Investigating the Evolution of Bad Smells in Object-Oriented Code*," 7th International Conference on the Quality of Information and Communications Technology (QUATIC '2010), Oporto, Portugal, September/October 2010, (accepted).

Συντομο Βιογραφικο Σημειωμα

Ο Αναστάσιος γεννήθηκε στην Κατερίνη-Πιερίας τον Ιανουάριο του 1983. Απεφοίτησε το 2005 από το τμήμα Τεχνολογίας Πληροφορικής και Τηλεπικοινωνιών, πρώην Τηλεπληροφορικής και Διοίκησης, του Ανώτατου Τεχνολογικού Εκπαιδευτικού Ιδρύματος Ηπείρου και το 2008 έγινε δεκτός στο Πρόγραμμα Μεταπτυχιακών Σπουδών του τμήματος Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας.

Τα ερευνητικά του ενδιαφέροντα αυτή τη περίοδο επικεντρώνονται στα προβλήματα σχεδίασης που συχνά προκύπτουν σε συστήματα λογισμικού και εκφυλίζουν την ποιότητα της σχεδίασής τους, τις αναδομήσεις και τις μετρικές ποιότητας λογισμικού με έμφαση σε αντικειμενοστρεφή συστήματα.