ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Αναδόμηση Κώδικα με Πρότυπα Σχεδίασης»

Χριστοπούλου Αικατερίνη
Μ 3090003

ΑΘΗΝΑ, 2 2011

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Αναδόμηση Κώδικα με Πρότυπα Σχεδίασης»

Χριστοπούλου Αικατερίνη
Μ 3090003

Επιβλέπων Καθηγητής: Γιακουμάκης Εμμανουήλ
Εξωτερικός Κριτής: Μαλεύρης Νικόλαος

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΘΗΝΑ, 2 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους ανθρώπους, οι οποίοι με τη συνεισφορά τους συνέβαλαν στην ολοκλήρωση αυτής της εργασίας.

Κατ' αρχάς, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον επιβλέποντα καθηγητή μου κ. Εμμανουήλ Γιακουμάκη για την καθοδήγηση που μου παρείχε. Οι υποδείξεις του αποδείχθηκαν πολύτιμες και η ενθάρρυνσή του με βοήθησε καθ' όλη τη διάρκεια εκπόνησης της διπλωματικής μου εργασίας.

Επιπλέον, θα ήθελα να ευχαριστήσω θερμά τον διδάκτωρ Πληροφορικής κ. Νικόλαο Διαμαντίδη, ο οποίος με την πολύτιμη βοήθεια του, τις χρήσιμες συμβουλές του και την επιμονή και υπομονή του, στάθηκε αρωγός στην ολοκλήρωση της ερευνητικής μου μελέτης και της συγγραφής της διπλωματικής μου.

Θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον καθηγητή κ. Αλέξανδρο Χατζηγεωργίου και τον διδάκτωρ Πληροφορικής κ. Νικόλαο Τσάνταλη που μου παραχώρησαν το ερευνητικό τους έργο και με εμπιστεύτηκαν στη συνέχιση του.

Επιπλέον, θα ήθελα να ευχαριστήσω τον διδάκτωρ Πληροφορικής κ. Βασίλη Ζαφείρη, ο οποίος μου εμπιστεύτηκε τον πηγαίο κώδικα της διδακτορικής του διατριβής για την διεξαγωγή των απαραίτητων ελέγχων, καθώς και για την συμβολή του στην ανεξάρτητη αξιολόγηση της προτεινόμενης τεχνικής, και τις χρήσιμες συμβουλές του για τη βελτίωση της μεθοδολογίας μου.

Αφιερώνω αυτή τη διπλωματική στους γονείς μου, που πάντα στέκονται στο πλευρό μου, με στηρίζουν και πιστεύουν σ' εμένα, στις αδερφές μου και τους φίλους μου, που αποτελούν για εμένα πηγή έμπνευσης και βοήθησαν να γίνει πιο εύκολο και πιο ευχάριστο αυτό το δύσκολο έργο.

Περίληψη

Η σταδιακή υποβάθμιση της ποιότητας του πηγαίου κώδικα αποτελεί ένα από τα πιο συχνά προβλήματα που αντιμετωπίζουν οι μηχανικοί λογισμικού κατά τη φάση της συντήρησης του κύκλου ζωής του λογισμικού, θέτοντας σε κίνδυνο την μακροβιότητα και την εξέλιξη του λογισμικού, καθιστώντας αναγκαία την άμεση αντιμετώπιση, σήμερα πιο έντονα από ποτέ άλλοτε.

Η αναδόμηση με πρότυπα σχεδίασης, μια από τις κύριες πρακτικές της εξελικτικής σχεδίασης, αντιμετωπίζει επιτυχώς το πρόβλημα υποβάθμισης της ποιότητας του λογισμικού, παρέχοντας ένα ισχυρό και πολύτιμο εργαλείο ενάντια σε αυτό. Η αναδόμηση είναι η διαδικασία βελτίωσης της ποιότητας σχεδίου ενός υπάρχοντος συστήματος λογισμικού, εφαρμόζοντας μετασχηματισμούς χαμηλού επιπέδου στην εσωτερική του δομή, χωρίς να μεταβάλλεται η εξωτερικά παρατηρήσιμη συμπεριφορά του. Από την άλλη, τα πρότυπα σχεδίασης συνιστούν υψηλού επιπέδου ευρέως αποδεκτές επαναχρησιμοποιήσιμες λύσεις σε συχνά εμφανιζόμενα σχεδιαστικά προβλήματα. Τα πλεονεκτήματα που προκύπτουν από το συνδυασμό αυτών των δύο χρήσιμων αντικειμενοστραφών τεχνικών, μπορούν να αξιοποιηθούν αποδοτικά μέσω της αναδόμησης με πρότυπα σχεδίασης.

Παρά τα αδιαμφισβήτητα πλεονεκτήματα της, η αναδόμηση συχνά υποχρησιμοποιείται. Οι αυξημένες απαιτήσεις κόστους της αναφορικά με τον χρόνο και την προσπάθεια που δαπανάται, οι οποίες πηγάζουν από τη διαρκή αύξηση του μεγέθους των σύγχρονων συστημάτων λογισμικού, συνοδευόμενη από την ευπάθεια της διαδικασίας σε σφάλματα, είναι ορισμένοι από τους κυριότερους λόγους γι’ αυτό. Το πρόβλημα οξύνεται περαιτέρω εξαιτίας της έλλειψης εργαλείων που υποστηρίζουν την αυτόματη εκτέλεση αναδόμησης με ή χωρίς πρότυπα σχεδίασης.

Τα εργαλεία αναδόμησης παρέχουν ταχύτητα και ασφάλεια απαλείφοντας την πιθανότητα εισαγωγής σφαλμάτων, διασφαλίζοντας παράλληλα τη διατήρηση της αρχικής συμπεριφοράς του συστήματος λογισμικού. Παρά την ύπαρξη μιας πληθώρας εργαλείων που υποστηρίζουν απλές χαμηλού επιπέδου αναδομήσεις, η πολυπλοκότητα των απαιτούμενων μετασχηματισμών για την ανίχνευση και την

εφαρμογή της αναδόμησης με πρότυπα σχεδίασης έχει ως αποτέλεσμα την απουσία αντίστοιχων αυτόματων εργαλείων.

Η παρούσα διπλωματική εργασία αναφέρεται στο πρόβλημα της αναδόμησης με πρότυπα σχεδίασης. Συγκεκριμένα, εξετάζεται η αυτοματοποίηση της αναδόμησης προς τη στρατηγική, όπου η επιλογή της υπό εκτέλεσης στρατηγικής επαφίεται πλήρως στην αρμοδιότητα της κλάσης πελάτη, τόσο στα πλαίσια της αυτοματοποιημένης ανίχνευση των πιθανών προτάσεων αναδόμησης σε υπάρχοντα πηγαίο κώδικα, καθώς επίσης και αναφορικά με την εφαρμογή της συγκεκριμένης αναδόμησης με αυτόματο τρόπο. Τη βάση της συγκεκριμένης διπλωματικής αποτέλεσε η μεθοδολογία ανίχνευσης υποψήφιων περιπτώσεων υποθετικής λογικής για την αναδόμηση με χρήση πολυμορφισμού που έχει υλοποιηθεί στο JDeodorant (1). Η παρούσα διπλωματική αποπειράται πρωτίστως να διαφοροποιηθεί από τις υπάρχουσες ερευνητικές εργασίες στον τομέα της αναδόμησης με πρότυπα σχεδίασης, ορίζοντας ένα πιο κατάλληλο σύνολο κριτηρίων ανίχνευσης, ειδικά επιλεγμένων για την περίπτωση της αναδόμησης προς τη στρατηγική, προκειμένου να τη διαχωρίσει από άλλες υποψήφιες δυνατότητες αναδόμησης πολύπλοκής υποθετικής λογικής. Ο πλήρης διαχωρισμός ανάμεσα στην αυτόματη αναδόμηση προς τη στρατηγική και την αναδόμηση προς την κατάσταση, το πρότυπο σχεδίασης εκείνο το οποίο διαθέτει την ίδια δομή με το πρότυπο στρατηγική, ωστόσο εξυπηρετεί διαφορετικό σκοπό, εξετάζεται εις βάθος.

Ένα επιπλέον ερευνητικό ερώτημα που εξετάζεται περαιτέρω αφορά τον τρόπο με τον οποίο η υποθετική λογική μπορεί να μετασχηματιστεί κατά την εφαρμογή της αναδόμησης. Οι υπάρχουσες μεθοδολογίες αναδόμησης απλοποιούν τις υποθετικές δηλώσεις, ωστόσο αποτυγχάνουν να τις απομακρύνουν πλήρως από τον πηγαίο κώδικα. Στα πλαίσια βελτιστοποίησης της ποιότητας του κώδικα, ένας επιπλέον στόχος που αποπειράται να επιτύχει η παρούσα ακαδημαϊκή ερευνητική εργασία είναι η πλήρης απαλοιφή των υποθετικών δηλώσεων, στις περιπτώσεις όπου αυτό καθίσταται εφικτό, με αυτοματοποιημένο τρόπο.

Αναφορικά με την αυτόματη ανίχνευση υποψήφιων περιπτώσεων αναδόμησης προς τη στρατηγική η προτεινόμενη μεθοδολογία περιλαμβάνει τη συντακτική ανάλυση του υπό εξέταση κώδικα και την κατασκευή, τόσο της αντίστοιχης αναπαράστασης του αντίστοιχου αφηρημένου συντακτικού δέντρου (AST), όσο και των γράφων ροής ελέγχου και εξαρτήσεων προγράμματος (CFG, PDG αντίστοιχα), όπου οι τελευταίοι χρησιμοποιούνται σε ένα μετέπειτα στάδιο της διαδικασίας ανίχνευσης αποκλειστικά για ένα συγκεκριμένο υποσύνολο των κριτηρίων ανίχνευσης. Το AST δέντρο αναλύεται περαιτέρω και τα πολύπλοκα τμήματα υποθετικού κώδικα εντοπίζονται εφαρμόζοντας ένα σύνολο κριτηρίων ανίχνευσης που κατατάσσονται σε τρείς ξεχωριστές κατηγορίες ανάλογα με την οντότητα η οποία αναλύεται:

a. **Κριτήρια σε επίπεδο μεθόδου**: κριτήρια που αφορούν την υπογραφή των μεθόδων οι οποίες εμπεριέχουν την υποθετική λογική.
b. **Κριτήρια σε επίπεδο συνθήκης**: κριτήρια αναφορικά με τις υποθετικές εκφράσεις των υποθετικών δηλώσεων.
c. **Κριτήρια σε επίπεδο διακλάδωσης**: κριτήρια που αναφέρονται στα τμήματα του κώδικα στο εσωτερικό των διακλαδώσεων των υποθετικών δηλώσεων.

Τα κριτήρια ανίχνευσης μπορούν επίσης να κατηγοριοποιηθούν ανάλογα με την εμβέλεια του ελέγχου ως εξής:

a. **Κριτήρια που ελέγχονται στα πλαίσια μεθόδων** (intraprocedural analysis): η πλειοψηφία των κριτηρίων απλά απαιτούν τον έλεγχο της υποθετικής λογικής.
b. **Κριτήρια που ελέγχονται στα πλαίσια κλάσης** (intraclass - interprocedural analysis): κάποια από τα κριτήρια αναφέρονται σε πρόσθετους ελέγχους των κλήσεων μεθόδων που δηλώνονται στο εσωτερικό της ίδιας κλάσης.
c. **Κριτήρια που ελέγχονται στο σύστημα** (interclass – interprocedural analysis): ένα συγκεκριμένο υποσύνολο κριτηρίων που περιλαμβάνει την ανάλυση κλήσεων μεθόδων ολόκληρου του συνόλου των κλάσεων συστήματος.

Οι προτάσεις αναδόμησης που ανιχνεύονται ως αποτέλεσμα της εφαρμογής των κριτηρίων ανίχνευσης στον πηγαίο κώδικα του συστήματος λογισμικού, βαθμονομούνται μέσω της ανάθεσης ενός ξεχωριστού βάρους, το οποίο αντανακλά την προσωπική εκτίμηση της συγγραφέως για την εγκυρότητα των προτάσεων αναδόμησης, βάσει στατιστικών δεδομένων που συλλέχθηκαν μέσω προσωπικής παρατήρησης κατά την ανάπτυξη και τον έλεγχο της προτεινόμενης μεθοδολογίας. Το επιλεγμένο σύνολο βαρών που χρησιμοποιήθηκε για τις διάφορες προτάσεις αναδόμησης παρέχει μια μοναδική ένα-προς-ένα συσχέτιση των δομικών χαρακτηριστικών των υποθετικών δηλώσεων, ενώ η υψηλότερη τιμή ανατίθεται σε μια ειδική περίπτωση πρόταση αναδόμησης, στην οποία καθίσταται εφικτή η πλήρης απαλοιφή της υποθετικής λογικής.

Τελικά τα αποτελέσματα της ανίχνευσης παρουσιάζονται στο χρήστη υπό τη μορφή μιας ταξινομημένης λίστας διατεταγμένης πρωταρχικά ανάλογα με το βάρος τους, ενώ η σειρά κατάταξής τους σε δεύτερο επίπεδο βασίζεται στον υπολογισμό του μέσου αριθμού δηλώσεων ανά διακλάδωση για κάθε τμήμα υποθετικού κώδικα που προτείνεται προς αναδόμηση. Ακολούθως, ο χρήστης έχει τη δυνατότητα να επιθεωρήσει τις προτάσεις αναδόμησης μια-προς-μια και να επιλέξει εκείνες που επιθυμεί να εκτελέσει.

Κατά την εφαρμογή του προτεινόμενου μετασχηματισμού αναδόμησης έπειτα από την έγκριση του χρήστη, το αφηρημένο συντακτικό δέντρο AST ανασυντάσσεται και εν συνεχεία, οι μετασχηματισμοί του χρησιμοποιούνται για να τροποποιηθεί ο πρωτότυπος πηγαίος κώδικας χωρίς την παρέμβαση του χρήστη. Σε αυτό το στάδιο, η πλήρης απαλοιφή της υποθετικής λογικής από τον πηγαίο κώδικα στις συγκεκριμένες περιπτώσεις, κατά τις οποίες η καθορισμένη υποθετική διακλάδωση προς εκτέλεση είναι γνωστή κατά το χρόνο μεταγλώττισης, συνιστά μια από τις κυριότερες συνεισφορές της συγκεκριμένης ακαδημαϊκής ερευνητικής εργασίας. Αυτό επιτυγχάνεται για ένα συγκεκριμένο υποσύνολο των δηλώσεων στις οποίες ο υποθετικός έλεγχος βασίζεται σε παραμέτρους της μεθόδου πρωταρχικού ή απαριθμητού τύπου, μέσω της στατικής ανάλυσης των κλήσεων της μεθόδου που περιλαμβάνει την υποθετική λογική, προκειμένου να διαπιστωθεί η γνώση των τιμών των ορισμάτων κατά το χρόνο μεταγλώττισης. Στο στάδιο εφαρμογής των μετασχηματισμών των περιπτώσεων αυτών, το εργαλείο αναδόμησης επεξεργάζεται κάθε κλήση μεθόδου και την τροποποιεί ούτως ώστε να προσδιορίζει το αντίστοιχο στιγμιότυπο της συγκεκριμένης υποκλάσης στρατηγικής. Η ανίχνευση της κατάλληλης στρατηγικής και της προς εκτέλεση διακλάδωσης ανάλογα με τις τιμές των ορισμάτων της εκάστοτε κλήσης μεθόδου, απαιτεί την μεμονωμένη εκτέλεση των υποθετικών εκφράσεων για το συγκεκριμένο σύνολο ορισμάτων, χωρίς ωστόσο να εκτελείται ολόκληρο το υπό εξέταση σύστημα λογισμικού.

Η προτεινόμενη μεθοδολογία υλοποιήθηκε με χρήση της γλώσσας προγραμματισμού Java και των βιβλιοθηκών JDT και LTK του Eclipse, και ενσωματώθηκε σαν τμήμα του ήδη υπάρχοντος εργαλείου αναδόμησης του Eclipse IDE JDeodorant. Το JDeodorant, το οποίο αποτέλεσε τη βάση αυτής της υλοποίησης, αποτελεί ένα plug-in του Eclipse που αφορά τα σχεδιαστικά προβλήματα «Feature Envy», «Type Checking», «Long Class» και «Long Method» και παρέχει τη δυνατότητα αναδόμησης όταν συναντώνται στον κώδικα. Η αναδόμηση πλήρους απαλοιφής της υποθετικής λογικής επιτυγχάνεται μέσω της δυναμικής προσθήκης μεθόδων με χρήση της βιβλιοθήκης javaassist. Επιπλέον, η δυνατότητα reflection της java αξιοποιείται για την εκτέλεση των μεθόδων που προστίθενται.

Μετά την ολοκλήρωση της υλοποίησης του σχετικού υποστηρικτικού εργαλείου, η προτεινόμενη μεθοδολογία δοκιμάστηκε σε διάφορα προγράμματα ανοικτού κώδικα, προκειμένου να μετρηθεί η ακρίβεια των αποτελεσμάτων της, τόσο ως προς την ακρίβεια, όσο και αναφορικά με την πληρότητα. Το σύνολο των υπό εξέταση συστημάτων λογισμικού περιλαμβάνει διάφορα συστήματα από διαφορετικούς τομείς εφαρμογών, τόσο γνωστής όσο και άγνωστης λειτουργικότητας στον αξιολογητή που διενέργησε την αξιολόγηση. Η αξιολόγηση των αποτελεσμάτων αναφορικά με την εγκυρότητα τους ως πραγματικές περιπτώσεις αναδόμησης προς τη στρατηγική εκτελέστηκε με τη βοήθεια ενός ανεξάρτητου αξιολογητή σε τρία διαφορετικά δοκιμαστικά συστήματα λογισμικού (IceHockeyManager, JFScheduler, MOEM), στα οποία η λίστα των προτάσεων αναδόμησης κατατάσσεται σε true/false positives και true/false negatives. Σχετικά με την κλιμακωσιμότητα της προσέγγισης, στο προηγούμενο σύνολο συστημάτων λογισμικού, προστέθηκαν τρία ακόμα (Nutch, Violet, Pamvotis). Καθένα από τα επιλεγμένα συστήματα ανοιχτού λογισμικού έχει διαφορετικό μέγεθος και διαφορετικές απαιτήσεις σε μνήμη και χρησιμοποιήθηκε για τον υπολογισμό του απαιτούμενου χρόνου εκτέλεσης της διαδικασίας ανίχνευσης. Τα αποτελέσματα των δύο πειραμάτων παρουσιάζονται και χρησιμοποιούνται προκειμένου να εξαχθούν ορισμένα χρήσιμα συμπεράσματα.

Εν κατακλείδι, η μελλοντική έρευνα περιλαμβάνει περαιτέρω έλεγχο της ακρίβειας της μεθοδολογίας και της πληρότητας σε ένα ευρύτερο σύνολο συστημάτων λογισμικού, ενώ επιπλέον ενδιαφέρον παρουσιάζει η δυνατότητα προσαρμογής των κριτηρίων ανίχνευσης του εργαλείου στα συγκεκριμένα χαρακτηριστικά του εκάστοτε εξεταζόμενου συστήματος λογισμικού.

Abstract

Gradual deterioration of source code quality is one of the most common problems software engineers face during the stage of maintenance of the software life cycle, putting the software's longevity and evolution at risk, urging for immediate treatment, nowadays more than ever before.

Refactoring to patterns, one of the key principles of evolutionary design, successfully addresses the problem of software quality decay, providing a powerful and valuable tool against it. Refactoring is the process of improving the design quality of an existing software system, by applying low level transformations to its internal structure, without modifying its externally observable behaviour. On the other hand, design patterns constitute to high-level reusable proven solutions to commonly occurring software design problems. The benefits resulting from the combination of these two useful object-oriented techniques can be effectively utilised through refactoring to patterns.

Despite its indisputable advantages, refactoring is often underutilized. Its great cost demands regarding time and effort spent, originating from the constant growth in size of modern software systems, accompanied by the procedure's error proneness, are some of the main reasons for that. The problem is further exacerbated due to the lack of tool support for the automatic application of refactoring to patterns, the use of which would overcome all above mentioned problems.

Refactoring tools quickly and safely automate code transformations while guaranteeing the preservation of the software system's initial behaviour. Despite the existence of a majority of tools supporting primitive refactorings, the complexity of the transformations required for the identification and the application of refactoring to patterns, has leaded to an absence of respective automatic tools.

This thesis wishes to address the problem of refactoring to patterns. Specifically, the automation of refactoring to strategy is examined, where the selection of the strategy to be executed lies in the client's entire responsibility, both in terms of automatic identification of possible refactoring suggestions in existing source code, and also in

regards to the application of the corresponding refactoring in an automatic way. The basis for this thesis was the methodology regarding identification of conditional logic refactoring opportunities introducing polymorphism implemented in JDeodorant (1). This thesis primarily attempts to differentiate from related work in the field of refactoring to patterns, by selecting a more appropriate set of identification criteria, particularly chosen for the case of refactoring to strategy, in order to distinguish it from other possible candidate refactorings of complex conditional logic. The complete separation between the automatic refactoring to strategy and refactoring to state, the design pattern which has the same structure with strategy pattern, yet serves another purpose, is examined in depth.

Another research question further investigated is the way conditional logic can be transformed during the application of refactoring. Existing refactoring methodologies simplify conditional statements, however fail to completely remove them from source code. In the scope of code quality optimization, a further goal to be accomplished by this scientific research is the full elimination of conditional statements, in cases where this is possible, in an automatic manner.

Regarding the automatic identification of refactoring to strategy candidates the methodology proposed includes syntactic analysis of the examined source code and the construction of both its corresponding abstract syntax tree (AST) representation and its control flow and program dependence graph (CFG, PDG respectively), the latter used on a later stage of the identification process only for a specific subset of the identification criteria. The AST tree is further analyzed and the complex conditional fragments of code are being detected by applying a set of identification criteria, classified to three distinct categories according to the entity analyzed:

a. **method level criteria**: criteria regarding the signature of the method whose body contains the conditional logic.
b. **condition level criteria**: criteria regarding the conditional expressions of the conditional statements.
c. **branch level criteria**: criteria addressing the fragments of code inside the branches of the conditional statement.

The identification criteria can also be categorised regarding their check scope as to

a. **criteria checked in methods** (intraprocedural analysis): the majority of the criteria only demand the checking of the conditional logic.
b. **criteria checked in class** (intraclass - interprocedural analysis): some of the criteria infer the additional check of method calls inside the boundaries of the same class.
c. **criteria checked in system** (interclass – interprocedural analysis): a particular subset of the criteria includes the analysis of method calls within the complete set of the system's classes.

The detected refactoring suggestions, resulting from the application of the identification criteria on the software system's source code, are calibrated through the assignment of discrete correspondent weights, reflecting the writer's personal estimation of the validity of the refactoring suggestions, based on statistical data gathered through personal observations during the development and testing of the proposed methodology. The chosen set of weights used to evaluate the various refactoring suggestions provides a unique one-to-one mapping to structural characteristics of the conditional statements, while the highest weight value is assigned to a special case of refactoring suggestion, in which the full elimination of conditional logic is applicable.

Finally, the results of identification are presented to the user in the form of a sorted list primarily according to their weights, whereas the basis of the second level sort order relies on the computation of the average number of statements per branch for each conditional code fragment proposed for refactoring. Subsequently, the user is able to preview the refactoring suggestions one-by-one and select the ones who wishes to apply.

During the application of the proposed refactoring transformation after the user's approval, the AST Tree is being restructured and its modifications are used to transform the original source code without the user's contribution. At this stage, the complete elimination of conditional logic from source code in certain cases, when the specific conditional branch to be executed is known at compilation time, constitutes one of the major contributions of this scientific-academic research work. This is achieved for the specific subset of statements in which conditional check is based on primitive or enumerated method parameters, through static analysis of the invocations of the method containing the conditional logic, in order to assure the knowledge of their values during compilation time. At the phase of the transformations application, each method call should be processed and modified as to specify an instance of the corresponding strategy subclass. The identification of the appropriate strategy and the branch code to be executed in regard to each method invocations arguments consequently, requires the isolated execution of conditional expressions for the specified arguments set, without actually executing the examined software system.

The proposed methodology has been implemented with use of Java programming language and also of the JDT and LTK Eclipse libraries and has been integrated in the already existing refactoring tool for Eclipse IDE, JDeodorant, as part of it. JDeodorant, which has been the basis of this implementation, is a plug-in for eclipse that targets “Feature Envy”, “Type Checking”, “Long Class” and “Long Method” bad smells and provides refactoring solutions to these problems. The full elimination of conditional logic refactoring is accomplished through the dynamic addition of methods with use of library javaassist. Furthermore, java’s reflection support is taken into advantage for the execution of the added methods.

After constructing the corresponding tool support, the proposed methodology has been tested in various open-source projects, in order to measure the accuracy of its results, both in terms of precision and recall. The set of examined projects includes various systems from different application domains, both of familiar and unfamiliar functionality to the expert conducting the evaluation. The evaluation of the calculated results regarding their validity as real opportunities of refactoring to strategy is performed with use of an independent expert on three different testing projects (IceHockeyManager, JFScheduler, MOEM), in which the list of refactoring suggestions is being classified as to true/false positives and true/false negatives. In regards to the scalability of this approach, three more systems have been added (Nutch, Violet, Pamvotis) to the previous set of examined software systems. Each one of the chosen open-source project has different size and memory requirements and has been used for the calculation of the necessary amount of computation/execution time required for the identification process. The results of both experiments are being presented and are used in order to infer some useful conclusions.

To conclude, future work involves further examination of the methodology's precision and recall in an even larger set of software systems, while another interesting aspect would be the adaptation of the implemented tool's identification criteria each time to the specific characteristics of the examined software system.

Περιεχόμενα

Πίνακας Πινάκων

Πίνακας Εικόνων

Κεφάλαιο 1

Εισαγωγή

Η ραγδαία ανάπτυξη που έχει συντελεστεί τα τελευταία χρόνια στην *τεχνολογία λογισμικού* συνδέεται άμεσα με την αύξηση της σημασίας που διαδραματίζει η φάση της *συντήρησης (maintainance)*. Η εκθετική αύξηση της σημασίας αυτής στην ανάπτυξη λογισμικού διαφαίνεται από διάφορες έρευνες, που υπολογίζουν ότι στη δεκαετία του '70 η φάση αυτή καταλάμβανε το 35% με 40% του προϋπολογισμού κόστους της ανάπτυξης ενός πληροφοριακού συστήματος, ποσοστό που στη δεκαετία του '80 αυξήθηκε σε 60%, ενώ από τη δεκαετία του '90 μέχρι σήμερα, σε αρκετές περιπτώσεις υπερβαίνει ακόμα και το 80 με 90% του κόστους ανάπτυξης ενός συστήματος λογισμικού (2) (3) (4) (5). Η πρωταρχική σημασία της συντήρησης αναδεικνύεται επιπλέον από το γεγονός ότι, οι οικονομικές επενδύσεις σ' αυτή καθώς και στην *εξέλιξη (evolution)* ενός συστήματος λογισμικού, υπερβαίνουν το αρχικό κόστος για την *σχεδίαση* και *ανάπτυξη* του, ενώ το 80% της συνολικής προσπάθειας που επενδύεται στη φάση αυτή, αφορά μη-διορθωτικές ενέργειες (6).

Οι ενέργειες αυτές σχετίζονται με τη δυναμική φύση των συστημάτων λογισμικού, των οποίων η λειτουργικότητα υφίσταται διαρκείς ενημερώσεις προκειμένου να παραμείνουν βιώσιμα και ανταγωνιστικά. Επιπλέον, οι συνεχείς αλλαγές στις απαιτήσεις καθώς και η εξέλιξη των χρησιμοποιούμενων τεχνολογιών, καθιστά αναγκαίο τον μετασχηματισμό των συστημάτων λογισμικού, ούτως ώστε να προσαρμοστούν σ' αυτές και παράλληλα να εμπλουτιστούν με την προσθήκη νέων χαρακτηριστικών. Από τα παραπάνω καθίσταται εμφανές ότι η αλλαγή αποτελεί αναπόσπαστο κομμάτι της *εξελικτικής ανάπτυξης (evolutionary development).* Ωστόσο, οι διαρκείς τροποποιήσεις στις οποίες υπόκειται ένα σύστημα λογισμικού έχουν ως αποτέλεσμα την βαθμιαία υποβάθμιση της ποιότητας του.

Για τη βελτίωση της ποιότητας ενός συστήματος λογισμικού με ταυτόχρονη μείωση του κόστους και της απαιτούμενης προσπάθειας για την διενέργεια της συντήρησης, έχουν προταθεί στη βιβλιογραφία διάφορες αντικειμενοστραφείς μεθοδολογίες. Ευρέως διαδεδομένη μεθοδολογία για την επίτευξη του σκοπού αυτού αποτελεί η χρήση ***προτύπων σχεδίασης*** *(design patterns)*, τα οποία ορίζονται ως προτιμώμενες λύσεις σε κοινά σχεδιαστικά προβλήματα. Τα πρότυπα σχεδίασης αφορούν τον συσχετισμό κλάσεων, μεθόδων και αντικειμένων μεταξύ τους, κατά τρόπο ώστε η δομή του παραγόμενου σχεδίου να επιτρέπει την ευκολότερη τροποποίηση και επέκταση του. Για το σκοπό αυτό αξιοποιούν χρήσιμες έννοιες της αντικειμενοστραφούς σχεδίασης, όπως η έννοια της κληρονομικότητας. Η κληρονομικότητα συμβάλει στην επαναχρησιμοποίηση και τη συντήρηση, επιτρέποντας τη δημιουργία υποκλάσεων που εξειδικεύουν τη λειτουργικότητα μιας ήδη υπάρχουσας κλάσης, χωρίς να επεμβαίνουν καθόλου και να τροποποιούν τον πρωτότυπο κώδικα της.

Εκτός από τα πρότυπα σχεδίασης στη βελτίωση της ποιότητας του σχεδίου του κώδικα αποσκοπεί και η *αναδόμηση (refactoring)*. Η αναδόμηση συνιστά την κυριότερη ενέργεια μετασχηματισμού του πηγαίου κώδικα, με στόχο τη βελτίωση των ποιοτικών χαρακτηριστικών του, χωρίς παράλληλα να τροποποιείται η εκτελούμενη και εξωτερικά παρατηρήσιμη συμπεριφορά του. Στα πλαίσια της παρούσας διπλωματικής εργασίας μελετάται ο συνδυασμός της αναδόμησης με τα πρότυπα σχεδίασης, ο οποίος συνδυάζει τα οφέλη των δύο αυτών τεχνικών, προκειμένου να αυξήσει την συντηρησιμότητα, την κατανοησιμότητα και την επεκτασιμότητα του πηγαίου κώδικα.

1.1. Σκοπός και Συμβολή της Διπλωματικής Εργασίας

Η παρούσα διπλωματική αποσκοπεί να συμβάλει στον τομέα της αναδόμησης με πρότυπα σχεδίασης. Συγκεκριμένα μελετάται το πρόβλημα της αυτοματοποιημένης αναδόμησης προς τη στρατηγική. Η αυτοματοποίηση απασχολεί την ερευνητική κοινότητα από τότε που πρωτοεμφανίστηκε η έννοια της αναδόμησης, καθώς μειώνει το κόστος της συντήρησης, την απαιτούμενη προσπάθεια και το χρόνο για την εκτέλεση της, την πιθανότητα εισαγωγής σφαλμάτων και την ανάγκη για *αποσφαλμάτωση (debugging)*.

Ωστόσο, η σχετική βιβλιογραφία χαρακτηρίζεται από σχετική έλλειψη αυτόματων τεχνικών αναδόμησης αναφορικά με τα πρότυπα σχεδίασης. Τα υπάρχοντα εργαλεία εστιάζουν στην αυτόματη υποστήριξη ενός υποσυνόλου μόνο, από τις πιο συχνές αναδομήσεις χαμηλού επιπέδου, και στερούνται της δυνατότητας εκτέλεσης πιο πολύπλοκων αναδομήσεων. Η έλλειψη αυτή οφείλεται κυρίως στην αυξημένη δυσκολία που παρουσιάζει ο αυτόματος εντοπισμός των σημείων στον κώδικα, όπου θα μπορούσαν να αξιοποιηθούν τα πρότυπα σχεδίασης, λόγω της εννοιολογικής τους φύσης, η οποία απαιτεί, πέρα από την εξέταση δομικών κριτηρίων, την εξέταση διάφορων ποιοτικών και εννοιολογικών κριτηρίων. Αξιόλογη προσπάθεια για την αντιμετώπιση αυτού του προβλήματος, αποτελεί το ερευνητικό έργο των συγγραφέων Τσάνταλη και Χατζηγεωργίου, αναφορικά με την αναδόμηση με χρήση των προτύπων κατάσταση ή στρατηγική (1).

Τα πρότυπα αυτά αξιοποιούν τις έννοιες της κληρονομικότητας και του πολυμορφισμού προκειμένου να αντικαταστήσουν πολύπλοκες υποθετικές δομές, που δυσχεραίνουν τη συντηρησιμότητα του κώδικα και είναι ευπαθείς στην εισαγωγή σφαλμάτων. Η αναδόμηση με τα πρότυπα αυτά, αντιμετωπίζεται από την προσέγγιση της συγκεκριμένης συγγραφικής ομάδας, από κοινού, βάσει της ομοιότητας στη δομή τους. Ωστόσο, υπό το πρίσμα της παρούσας διπλωματικής, τα δύο πρότυπα παρά την ομοιότητα τους, παρουσιάζουν και σημαντικές διαφορές. Συνεπώς, η εργασία αυτή διατείνεται ότι η κατάσταση και η στρατηγική χρήζουν διαφορετικής αντιμετώπισης κατά την αναδόμηση και αποπειράται να διαφοροποιηθεί από την ανωτέρω προσέγγιση, η οποία αποτέλεσε τη βάση για την υλοποίηση της.

Η προσέγγιση που αναπτύχθηκε στα πλαίσια αυτά αποσκοπεί να αποδώσει τη διαφορά των δύο προτύπων, τροποποιώντας την τεχνική ανίχνευσης και εκτέλεσης της αναδόμησης προς τη στρατηγική, από αυτή που χρησιμοποιείται για το πρότυπο κατάσταση. Απώτερο σκοπό και ταυτόχρονα βασική συμβολή αυτής της διαφοροποίησης, αποτελεί η κατά το δυνατόν βελτίωση των παραγόμενων προτάσεων αναδόμησης, ούτως ώστε να περιλαμβάνονται και περιπτώσεις, που στο παρελθόν δεν ανιχνεύονταν, εφόσον τα κριτήρια που είχαν οριστεί για το σκοπό αυτό ήταν κοινά, και δεν εξειδικεύονταν ανάλογά με το σκοπό που επιτελεί καθένα από τα δύο αυτά πρότυπα ξεχωριστά. Τέλος, έμφαση δίνεται στην επίτευξη πλήρους απαλοιφής της υποθετικής δομής από τον κώδικα, η οποία δεν επιτυγχάνεται στην μεθοδολογία αναδόμησης προς την κατάσταση ή τη στρατηγική. Η επίτευξη αυτή ακόμα και για ελάχιστες προτάσεις αναδόμησης θεωρείται μια από τις κυριότερες συμβολές της παρούσας ερευνητικής εργασίας καθώς γεννά νέες προοπτικές περαιτέρω μελέτης και βελτίωσης των υπαρχουσών μεθοδολογιών.

1.2. Δομή της Διπλωματικής Εργασίας

Ακολούθως παρουσιάζεται η κατανομή των ερευνητικών ζητημάτων στα κεφάλαια που αποτελούν αυτή τη διπλωματική εργασία.

Στο ***Κεφάλαιο 2*** παρουσιάζονται οι θεμελιώδεις έννοιες της τεχνολογίας λογισμικού αναφορικά με την *αναδόμηση (refactoring)*, τα *πρότυπα σχεδίασης (design patterns)* και την *αναδόμηση με πρότυπα σχεδίασης (refactoring to patterns).* Η παρουσίαση αυτή καθίσταται αναγκαία για την πληρέστερη κατανόηση της σημασίας των εννοιών που χρησιμοποιούνται σε όλη την έκταση της παρούσας διπλωματικής εργασίας.

Ακολούθως, στο ***Κεφάλαιο 3*** πραγματοποιείται μια επισκόπηση της βιβλιογραφίας περί της αυτοματοποιημένης αναδόμησης με χρήση προτύπων σχεδίασης, τόσο αναφορικά με την ανίχνευση των προβληματικών σημείων στον πηγαίο κώδικα, όσο και στα πλαίσια της εκτέλεσης του μετασχηματισμού της αναδόμησης, προκειμένου να εφαρμοστεί το πρότυπο σχεδίασης που απουσιάζει.

Εν συνεχεία, στο ***Κεφάλαιο 4*** αναλύεται περαιτέρω η αυτόματη αναδόμηση προς τη στρατηγική. Παρουσιάζεται ενδελεχώς η προσέγγιση που παρέχεται από το εργαλείο *JDeodorant* (εφεξής αναφέρεται και ως υπάρχουσα*)*, η οποία αποτέλεσε τη βάση για την ανάπτυξη και υλοποίηση της προσέγγισης της παρούσας διπλωματικής εργασίας (εφεξής και ως προτεινόμενης).

Το ***Κεφάλαιο 5*** παρουσιάζει αναλυτικά τη μεθοδολογία που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας και παρέχει περαιτέρω πληροφορίες αναφορικά με την υλοποίηση της. Για την πληρέστερη παρουσίαση της προτεινόμενης προσέγγισης περιλαμβάνεται η επίδειξη του τροποποιημένου κώδικα που παράγεται από την εκτέλεση της αναδόμησης μερικής αντικατάστασης της υποθετικής λογικής σε ένα απλοποιημένο παράδειγμα που αναπτύχθηκε αποκλειστικά για το σκοπό αυτό, αλλά και ο μετασχηματισμένος κώδικας ενός αποσπάσματος ενός πραγματικού συστήματος λογισμικού, στο οποίο η υποθετική λογική απαλείφεται πλήρως. Επιπλέον αναλύονται οι περιορισμοί στους οποίους υπόκειται η προτεινόμενη προσέγγιση αναδόμησης, ενώ στην τελευταία ενότητα του κεφαλαίου συνοψίζονται οι διαφορές των δύο μεθοδολογιών μέσω της συγκεντρωτικής αντιπαραβολής τους.

Για την πληρέστερη ανάπτυξη της προτεινόμενης μεθοδολογίας εκτελέστηκε πειραματική εφαρμογή της σε διάφορα συστήματα *ανοικτού λογισμικού (open-source)* με στόχο την αποτίμηση της κλιμακωσιμότητας της μεθόδου, καθώς και της ποιότητας των προτεινόμενων αναδομήσεων, από πλευράς *ακρίβειας (precision)* και πληρότητας *(recall),* τα αποτελέσματα της οποίας παρατίθενται στο ***Κεφάλαιο 6***, σε συνδυασμό με τις όλες τις απαραίτητες πληροφορίες αναφορικά με την πειραματική διαδικασία (συστήματα λογισμικού που χρησιμοποιήθηκαν, μετρικές αξιολόγησης, κ.ο.κ.).

Ολοκληρώνοντας, στο ***Κεφάλαιο 7*** συνοψίζεται η συνεισφορά της προτεινόμενης αυτοματοποιημένης τεχνικής αναδόμησης προς τη στρατηγική και παρουσιάζονται διάφορες ενδιαφέρουσες ερευνητικές κατευθύνσεις σχετικά με τις δυνατότητες για την περαιτέρω εξέλιξη στον τομέα αυτό.

Κεφάλαιο 2

Αναδόμηση Κώδικα και Πρότυπα Σχεδίασης

2.1 Αναδόμηση Κώδικα

Με την πάροδο του χρόνου τα *αντικειμενοστραφή συστήματα λογισμικού* (object-oriented systems) υφίστανται συχνά αλλαγές, είτε εξαιτίας τροποποιήσεων στις απαιτήσεις σχεδίασης, είτε λόγω της ανάγκης προσθήκης νέας λειτουργικότητας, είτε εξαιτίας της ανάγκης προσαρμογής στις τεχνολογικές εξελίξεις. Οι αλλαγές αυτές συχνά εκτελούνται βεβιασμένα, υπό την πίεση αυστηρών προθεσμιών παράδοσης, με αποτέλεσμα να πλήττεται η ποιότητα του λογισμικού και συγκεκριμένα τα χαρακτηριστικά της *συντηρησιμότητας (maintainability),* της *κατανοησιμότητας (understandability),* και της *τροποποιησιμότητας/επεκτασιμότητας (extensibility)* του πηγαίου κώδικα (source code). Τα χαρακτηριστικά αυτά τίθενται σε κίνδυνο επιπλέον, όταν οι απαιτούμενες τροποποιήσεις πραγματοποιούνται από μέλη της ομάδας ανάπτυξης λογισμικού που στερούνται πλήρης γνώσης και κατανόησης του σχεδίου του συστήματος και της λειτουργικότητας του. Ζωτικής σημασίας για την εξέλιξη των αντικειμενοστραφών συστημάτων, την αποφυγή της υποβάθμισης της ποιότητας και τη διασφάλιση της μακροβιότητας τους, αποτελεί η *αναδόμηση (refactoring)*.

Η έννοια της *αναδόμησης* εμφανίστηκε για πρώτη φορά στη διδακτορική διατριβή του W. Opdyke (7). Ορίζεται ως η διαδικασία τροποποίησης της εσωτερικής δομής του κώδικα, κατά τρόπο ώστε να μη μεταβάλλεται η εξωτερικά παρατηρήσιμη συμπεριφορά του. Στο βιβλίο του Fowler (8) παρέχονται δύο διαφορετικοί ορισμοί σύμφωνα με τους οποίους η λέξη *«αναδόμηση»* αναφέρεται τόσο ως διαδικασία (ουσιαστικό), όσο και ως ενέργεια (ρήμα):

> ***Αναδόμηση** (Ουσιαστικό):* μια αλλαγή στην εσωτερική δομή ενός λογισμικού ούτως ώστε να είναι ευκολότερη η κατανόηση του και οικονομικότερη η τροποποίηση του, χωρίς να μεταβάλλεται η παρατηρήσιμη συμπεριφορά του.

> ***Αναδομώ*** *(Ρήμα):* αναδιαρθρώνω το λογισμικό εφαρμόζοντας μια ακολουθία από αναδομήσεις χωρίς να τροποποιείται η εξωτερικά παρατηρήσιμη συμπεριφορά του.

Κατ' αντιστοιχία, ο K. Beck ορίζει την αναδόμηση ως την «*τροποποίηση του συστήματος που αφήνει αμετάβλητη τη συμπεριφορά του, αλλά βελτιώνει ορισμένα μη-λειτουργικά ποιοτικά χαρακτηριστικά του, την απλότητα, την ευελιξία, την κατανοησιμότητα και την απόδοση*» (9) και ο W. Wake αναφέρει την αναδόμηση ως την «*τεχνική βελτίωσης του σχεδίου υπάρχοντα κώδικα. Η αναδόμηση μας παρέχει τρόπους αναγνώρισης του προβληματικού κώδικα και μας δίνει οδηγίες για να τον βελτίωσουμε*» (10).

Η έννοια της ***αναδόμησης*** *(refactoring)* συχνά συγχέεται με τους συγγενείς όρους της περιοχής της συντήρησης λογισμικού, του ***ανασχεδιασμού*** *(reengineering)* και της ***αναδιάρθρωσης*** *(restructuring).* Προκειμένου να αποσαφηνιστεί η έννοια της αναδόμησης καθίσταται αναγκαίο να παρουσιαστούν οι ορισμοί των συγγενών αυτών όρων, καθώς και οι μεταξύ τους συσχετίσεις.

Η πιο γενική και υψηλού επιπέδου έννοια εκ των τριών, είναι η έννοια του ***ανασχεδιασμού*** *(reengineering)*, η οποία εμπεριέχει τις έννοιες της αναδόμησης και της *αναδιάρθρωσης* (11). Ορίζεται από την συγγραφική ομάδα των Chikofsky και Cross (12) ως «*η εξέταση και μετατροπή του υποκείμενου συστήματος για την ανασύσταση του σε μια νέα μορφή και η μεταγενέστερη υλοποίηση βάσει της νέας μορφής*». Ο ανασχεδιασμός αφορά στη δημιουργία ενός καλώς δομημένου συστήματος από υποβαθμισμένο ή legacy[1] κώδικα (13) ή ακόμα και στη μετατροπή του κώδικα σε μια άλλη γλώσσα προγραμματισμού (migration) (14). Ο ανασχεδιασμός περιλαμβάνει κάποιας μορφής αντίστροφο σχεδιασμό (reverse engineering), προκειμένου να επιτευχθεί μια πιο αφηρημένη αναπαράσταση, ακολουθούμενη από κάποιας μορφής ευθείας σχεδίασης (forward engineering) ή αναδιάρθρωσης (restructuring). Παράλληλα ενδέχεται να περιλαμβάνει και τροποποιήσεις αναφορικά με καινούργιες απαιτήσεις που δεν υποστηρίζονται στο πρωτότυπο σύστημα.

Η έννοια της ***αναδιάρθωσης*** *(restructuring)* ορίζεται σύμφωνα με την ταξονομία των Chikofsky και Cross (12) ως

[1] Legacy κώδικας: πηγαίος κώδικας ο οποίος βρίσκεται υπό λειτουργία για μεγάλο χρονικό διάστημα και δε βρίσκεται πλέον υπό ανάπτυξη.

> «ο μετασχηματισμός από μια μορφή αναπαράστασης σε μια άλλη, στο ίδιο σχετικό επίπεδο αφαίρεσης[2], διατηρώντας παράλληλα την εξωτερική συμπεριφορά του υποκείμενου συστήματος (λειτουργικότητα και σημασιολογία). Ο μετασχηματισμός αναδιάρθρωσης, όπως παραδείγματος χάριν η αλλαγή του κώδικα προκειμένου να βελτιωθεί η δομή του, αφορά συνήθως την εμφάνιση του κώδικα, υπό την παραδοσιακή έννοια της δομημένης σχεδίασης. Ενώ η αναδιάρθρωση δημιουργεί νέες εκδόσεις που υλοποιούν ή συνιστούν αλλαγή του υποκείμενου συστήματος, κανονικά δεν περιλαμβάνει τροποποιήσεις χάριν νέων απαιτήσεων».

Η αναδιάρθρωση δεν προϋποθέτει την κατανόηση της σημασιολογίας του κώδικα και χαρακτηρίζεται καλύτερα από το παράδειγμα μετασχηματισμού ενός αδόμητου τμήματος κώδικα (παραδείγματος χάριν “spaghetti code”) , σε δομημένο κώδικα (“goto-less” μορφή). Ωστόσο ο όρος έχει ευρύτερη έννοια και μπορεί να χρησιμοποιηθεί πρόσθετα για το μετασχηματισμό του μοντέλου δεδομένων (π.χ. κανονικοποίηση σχήματος), του μοντέλου σχεδίασης και των απαιτήσεων.

Τέλος, η ***αναδόμηση*** (refactoring) αποτελεί το είδος της αναδιάρθρωσης (restructuring), η οποία εφαρμόζεται σε επίπεδο κώδικα *(code refactoring)* στα πλαίσια της αντικειμενοστραφούς σχεδίασης υιοθετώντας τις αρχές της κληρονομικότητας, της ενθυλάκωσης, της ανάθεσης *(delegation)*, κ.ο.κ., για την ανακατανομή των κλάσεων, των μεθόδων και των μεταβλητών στην ιεραρχία των κλάσεων, ούτως ώστε να διευκολύνονται μελλοντικές προσθήκες και τροποποιήσεις της λειτουργικότητας του συστήματος, ενώ παραλλαγή της αναδόμησης μπορεί να εφαρμοστεί και σε επίπεδο μοντέλου *(model refactoring)*, μετασχηματίζοντας κατάλληλα τα διαγράμματα UML ενός υπάρχοντος συστήματος λογισμικού. Ο M. Fowler τονίζει τη διαφοροποίηση της αναδιάρθρωσης ως ένα σύνολο ευρύτερων μετασχηματισμών κατά τη διάρκεια των οποίων η δυνατότητα εκτέλεσης του συστήματος μπορεί να διακόπτεται, εν αντιθέσει με την αναδόμηση, όπου το εύρος των εκτελούμενων μετασχηματισμών είναι τόσο περιορισμένο, ώστε οποιοδήποτε σφάλμα να μπορεί να εντοπιστεί και να διορθωθεί άμεσα, ούτως ώστε το πρόγραμμα να παραμένει εκτελέσιμο διαρκώς.

[2] Στην ταξινομία των Chikofsky και Cross ορίζονται τα εξής επίπεδα αφαίρεσης: α) απαιτήσεων (ο προσδιορισμός του προς επίλυση προβλήματος ο οποίος περιλαμβάνει τους στόχους, τους περιορισμούς και τους επιχειρησιακούς κανόνες, β) σχεδίου (ο προσδιορισμός της λύσης), γ) υλοποίησης (η κωδικοποίηση, ο έλεγχος και η παράδοση του λειτουργικού συστήματος).

2.1.1. Πλεονεκτήματα και Μειονεκτήματα Αναδόμησης

Τα οφέλη από την εφαρμογή της αναδόμησης ποικίλουν και είναι άμεσα συνυφασμένα με τα χαρακτηριστικά του ποιοτικού λογισμικού. Συγκεκριμένα, η αναδόμηση βελτιώνει την κατανοησιμότητα του κώδικα και συνεπαγωγικά την επεκτασιμότητα του. Ως εκ τούτου, διευκολύνεται η συντήρηση του συστήματος λογισμικού, ενώ παράλληλα μειώνεται αισθητά ο χρόνος που απαιτείται για τη διεξαγωγή της διαδικασίας αυτής καθώς και της διαδικασίας ανάπτυξης. Κατ' αντιστοιχία, μειώνεται και ο απαιτούμενος χρόνος διόρθωσης σφαλμάτων που υπάρχουν στον κώδικα, καθώς διευκολύνεται ο εντοπισμός τους. Η βελτίωση του σχεδίου του λογισμικού συμβάλει παράλληλα στην καλύτερη αξιοποίηση των προγραμματιστικών πόρων και στην αποφυγή εισαγωγής σφαλμάτων κατά την υλοποίηση ενδεχόμενης απαιτούμενης νέας λειτουργικότητας.

Εντούτοις, η εφαρμογή της αναδόμησης, παρά τη θετική συνδρομή της στην ποιότητα του κώδικα, ενέχει ταυτόχρονα και ορισμένους κινδύνους. Ουσιώδους σημασίας είναι ο κίνδυνος εισαγωγής σφαλμάτων καθώς και ο κίνδυνος τροποποίησης της λειτουργικότητας του συστήματος, σε περίπτωση που δεν ληφθεί η απαιτούμενη μέριμνα κατά την εφαρμογή των βημάτων της αναδόμησης. Το γεγονός αυτό σε συνδυασμό με το αυξημένο κόστος της διαδικασίας, λόγω της απαιτούμενης προσπάθειας σε ανθρωποώρες για την ολοκλήρωση της, το οποίο δεν αντισταθμίζεται από άμεσα αντιληπτά οικονομικά οφέλη, συντελεί ούτως ώστε η διοίκηση ενός έργου να διατηρεί συχνά απρόθυμη στάση έναντι της αναδόμησης. Διστακτικότητα αναφορικά με την αλλαγή της δομής του κώδικα, συναντάται συχνά και από την πλευρά των μηχανικών λογισμικού (software engineers), κυρίως λόγω της εγγενούς αντίστασης του ανθρώπου στην αλλαγή και του φόβου με τον οποίο συνοδεύεται, και επιπλέον λόγω της απαιτούμενης προσπάθειας που θα πρέπει να καταβάλουν, για την ολοκλήρωση αυτής της επίπονης και χρονοβόρας διαδικασίας.

Ωστόσο οι κίνδυνοι αυτοί δύναται να παρακαμφθούν, εφόσον η αναδόμηση εφαρμόζεται ως μια τυπική δομημένη διαδικασία, αποτελούμενη από συντονισμένες διακριτές ενέργειες, και όχι ως τυχαία ακολουθία ad hoc ενεργειών. Συγκεκριμένα, οι ενέργειες που συνιστούν την αναδόμηση είναι οι εξής (15) :

1. Ανίχνευση του υποψήφιου τμήματος του λογισμικού που χρήζει αναδόμησης.

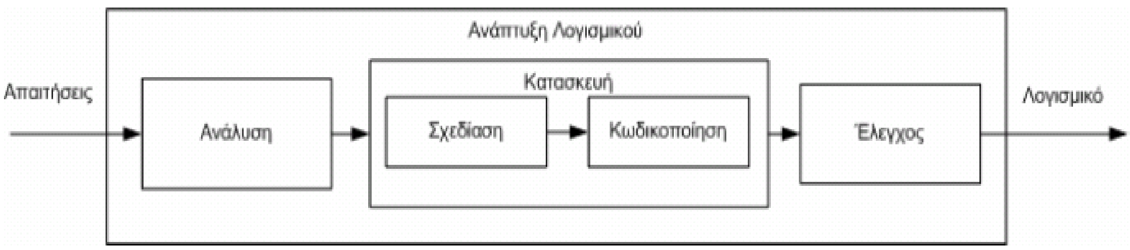
2. Προσδιορισμός της κατάλληλης αναδόμησης που ενδείκνυται στην εκάστοτε περίπτωση.
3. Παροχή εχεγγύων για τη διασφάλιση διατήρησης της συμπεριφοράς του συστήματος έπειτα από την εφαρμοσμένη αναδόμηση.
4. Υλοποίηση της αναδόμησης.
5. Αποτίμηση της αξίας της αναδόμησης μέσω της αξιολόγησης της επιρροής της στα ποιοτικά χαρακτηριστικά του λογισμικού (π.χ. πολυπλοκότητα, κατανοησιμότητα, συντηρησιμότητα) ή των χαρακτηριστικών της ίδιας διαδικασίας (π.χ. παραγωγικότητα, κόστος, απαιτούμενη προσπάθεια).
6. Διατήρηση της συνοχής του αναδομημένου κώδικα και ενδεχόμενων συμπληρωματικών – συνοδευτικών εγγράφων (software artefacts) που σχετίζονται με αυτόν (έγγραφα τεκμηρίωσης, έγγραφα περιγραφής απαιτήσεων – προδιαγραφών, διαγράμματα UML, τεστ, παρεμβαλλόμενα σχόλια, κλπ.).

Ιδιαίτερη δυσκολία κατά την διαδικασία αναδόμησης αντιμετωπίζεται στα αρχικά της στάδια, που αφορούν την ανίχνευση του προβληματικού κώδικα και την επιλογή του κατάλληλου μετασχηματισμού προκειμένου να βελτιωθεί η ποιότητα του σχεδίου. Βοήθεια σχετικά με την ανίχνευση παρέχουν οι λεγόμενες «**κακές οσμές**» **(bad smells)**, όπως αναφέρθηκαν για πρώτη φορά από τον K. Beck τα σημάδια του κώδικα που υποδεικνύουν την ύπαρξη κάποιου προβλήματος και την ανάγκη κάποιου είδους βελτίωσης μέσω της διαδικασίας της αναδόμησης (8). Τα bad ή code smells, όπως εναλλακτικά αναφέρονται στη σχετική βιβλιογραφία, αποτελούν συγκεκριμένης δομής περιγραφές, που καθιστούν αναγκαία την εφαρμογή της αναδόμησης και παρέχουν στους μηχανικούς λογισμικού ένα κατάλληλο μέσο να επικοινωνήσουν μεταξύ τους το πρόβλημα σχεδίασης, προκειμένου να εντοπίσουν ακολούθως την κατάλληλη λύση. Τα βιβλία των M. Fowler (8) και J.Kerievsky (16) παρέχουν ιδιαίτερη βοήθεια και καθοδήγηση στην ανίχνευση των σχεδιαστικών προβλημάτων καθώς αποδίδουν στα bad smells χαρακτηριστικά ονόματα, αντιπροσωπευτικά του προβλήματος στο οποίο απευθύνονται, τα περιγράφουν αναλυτικά και υποδεικνύουν σε κάθε περίπτωση bad smell ποιές αναδομήσεις ή ποιοι συνδυασμοί αναδομήσεων ενδείκνυνται για την βέλτιστη αντιμετώπιση του προβλήματος. Ωστόσο ο εντοπισμός των bad smells στον κώδικα απευθύνεται κυρίως στην ανθρώπινη λογική, συνεπώς δεν είναι δυνατό να επιτευχθεί από κάποιο αυτόματο εργαλείο. Επιπλέον, η ανθρώπινη νόηση απαιτείται και κατά την επιλογή του κατάλληλου μετασχηματισμού, εφόσον κάθε bad smell σχετίζεται με πολλούς εναλλακτικούς τρόπους αναδόμησης και εκεί έγκειται η δυσκολία του δεύτερου βήματος της διαδικασίας αναδόμησης.

Η διαδικασία της αναδόμησης καθίσταται αναγκαίο να ενσωματωθεί στις πρακτικές ανάπτυξης λογισμικού και να εκτελείται διαρκώς. Η σημασία της διαρκούς εκτέλεσης τα αναδόμησης αποτελεί βασική αρχή της ***εξελικτικής σχεδίασης*** (*evolutionary design*) και αναπόσπαστο κομμάτι των σύγχρονων μεθοδολογιών ανάπτυξης, *της **agile ανάπτυξης λογισμικού** (agile software development)* και του ***extreme προγραμματισμού*** *(extreme programming XP)* (9). Η agile ανάπτυξη λογισμικού με κυρίαρχη πρακτική της τη μεθοδολογία ανάπτυξης του extreme προγραμματισμού, συστήνει μια διαδικασία ανάπτυξης η οποία αποτελείται από πολλές *επαναλήψεις (iterations)* όλων των φάσεων του κύκλου ζωής του λογισμικού (Εικόνα 1), καθεμία από τις οποίες επεκτείνει τη λειτουργικότητα του συστήματος λίγο περισσότερο. Αυτή η μεθοδολογία ανάπτυξης έρχεται σε αντίθεση με παραδοσιακά μοντέλα ανάπτυξης όπως το μοντέλο του καταρράκτη όπου η διαδοχή των διαφόρων φάσεων εκτελείται σειριακά, έπειτα από την επιτυχή πλήρη ολοκλήρωση της προηγούμενης φάσης. Κατ' αντιστοιχία, η φιλοσοφία της ***εξελικτικής σχεδίασης*** *(evolutionary design)*, σύμφωνα με την οποία η υλοποίηση συγκεκριμένων χαρακτηριστικών στον κώδικα θα πρέπει να αναβάλλεται μέχρις ότου καταστούν πραγματικά αναγκαία, οπότε η σχεδίαση θα μεταβληθεί κατάλληλα με χρήση της αναδόμησης για να ενσωματώσει τις αλλαγές, έρχεται σε αντίθεση με την έννοια της **εκ των προτέρων σχεδίασης** (upfront design), βάση της οποίας αποτελεί η πρόβλεψη των μελλοντικών αναγκών του συστήματος, προκειμένου να ληφθούν υπόψη κατά τη σχεδίαση.

Από τα παραπάνω γίνεται σαφές ότι σύμφωνα με τις σύγχρονες μεθοδολογίες ανάπτυξης λογισμικού, η αναδόμηση αποτελεί αναπόσπαστο κομμάτι της ανάπτυξης, το οποίο εκτελείται διαρκώς. Συγκεκριμένα κατά την προσθήκη νέας λειτουργικότητας η αναδόμηση διευκολύνει στον εντοπισμό και την κατανόηση του κώδικα προς τροποποίηση και στην υλοποίηση της αλλαγής. Κατά τη διόρθωση ενός σφάλματος η αναδόμηση συμβάλει στον εντοπισμό και την αποφυγή του, ενώ κατά την επανεξέταση του πηγαίου κώδικα συνεισφέρει στη μετάδοση της γνώσης στην ομάδα ανάπτυξης και στην ευρύτερη κατανόηση περισσότερων πτυχών του συστήματος λογισμικού καθώς και του μελλοντικού ορίζοντα του έργου.

Επιπλέον, ανάλογα με τη συχνότητα με την οποία επιτελείται καθώς και με τον τρόπο με τον οποίο συνδυάζεται με τα υπόλοιπα προγραμματιστικά καθήκοντα, η αναδόμηση διακρίνεται σύμφωνα με τους Murphy-Hill και Black (17) σε *αναδόμηση* ***tactic floss***, που επιλέγεται για τη διατήρηση της ορθότητας του πηγαίου κώδικα, και *αναδόμηση* ***root canal***, που επιλέγεται για τη διόρθωση κώδικα που εμπεριέχει σφάλματα. Η αναδόμηση *tactic floss* εκτελείται συχνά και εναλλάσσεται με τις υπόλοιπες τροποποιήσεις που εκτελούνται στον πηγαίο κώδικα, ενώ η αναδόμηση *root canal* εφαρμόζεται σπάνια, σε παρατεταμένες περιόδους κατά τη διάρκεια των οποίων πραγματοποιούνται ελάχιστες έως καθόλου προγραμματιστικές αλλαγές.



Εικόνα 1: Ο κύκλος ζωής του λογισμικού

2.1.2. Είδη Αναδόμησης

Στη βιβλιογραφία αναφέρονται διάφορα είδη αναδόμησης τα οποία ανάλογα με την πολυπλοκότητα τους, τον τρόπο εφαρμογής τους και την συμβολή του ελέγχου κατά την ανάπτυξη τους διακρίνονται σε

- **απλή/χαμηλού επιπέδου αναδόμηση** (atomic/simple/primitive/low level refactoring): στοιχειώδης μετασχηματισμός που εκτελείται σε επίπεδο κλάσης, μεθόδου ή μεταβλητής και δεν τροποποιεί τη συμπεριφορά του υπό εξέταση λογισμικού.
- **σύνθετη αναδόμηση** (compound/composite refactoring): υψηλού επιπέδου αναδόμηση που αποτελείται από ένα σύνολο χαμηλού επιπέδου αναδομήσεων. Ανάμεσα στην εφαρμογή των διαφόρων στοιχειωδών αναδομήσεων εκτελούνται έλεγχοι για την επιβεβαίωση της διατήρησης της συμπεριφοράς του πηγαίου κώδικα.

- **test-driven αναδόμηση** (test-driven refactoring): η αναδόμηση κατά την οποία ο μετασχηματισμένος κώδικας παράγεται αξιοποιώντας την καθοδηγούμενη από έλεγχο ανάπτυξη (test-driven development) και κατόπιν αντικαθιστά τον παλιό κώδικα.

Ο Fowler παρέχει έναν εκτενή κατάλογό με 72 αναδομήσεις, τόσο απλές όσο και πιο σύνθετες (8). Ο κατάλογος αυτός αποτελεί σημαντική συμβολή καθώς πρόκειται για την πρώτη προσπάθεια συλλογής της εμπειρικής γνώσης που σχετίζεται με την αναδόμηση, οργάνωσης και παρουσίασης της με τρόπο σαφή, δομημένο, συστηματικό και καλώς προσδιορισμένο, ούτως ώστε ο αναγνώστης να διαθέτει ένα πλήρες εγχειρίδιο με οδηγίες σχετικά με τη βελτίωση του κώδικα που παράγει. Για το σκοπό αυτό ο Fowler για κάθε αναδόμηση που περιγράφει, προσδιορίζει

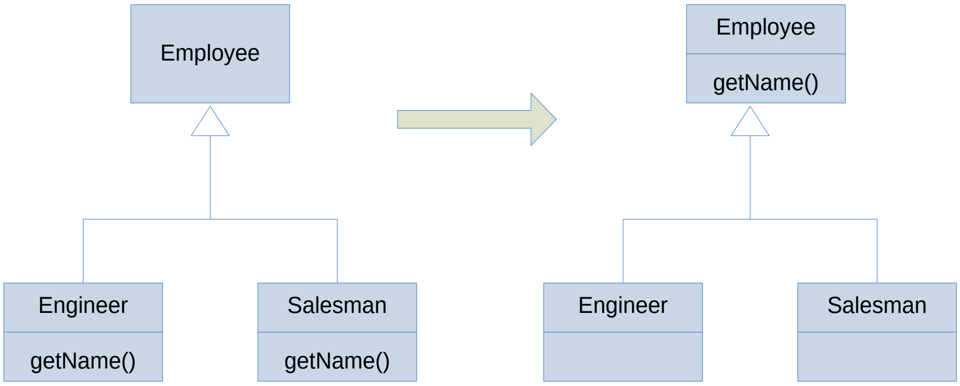
- ένα μοναδικό ***όνομα***, συναφές με το σκοπό της αναδόμησης,
- μια σύντομη ***περιγραφή*** της περίπτωσης στην οποία συνίσταται η αναδόμηση καθώς και της λειτουργικότητας της αναδόμησης,
- τους ***λόγους*** - τα ***κίνητρα*** *(motivation)* για την εφαρμογή της αναδόμησης καθώς και παραδείγματα, στα οποία αντενδείκνυται,
- τον ***αλγόριθμο*** *(mechanics)* που περιλαμβάνει βήμα προς βήμα τις οδηγίες για την εφαρμογή της αναδόμησης και τέλος
- απλά ***παραδείγματα*** εφαρμογής της αναδόμησης, που επιδεικνύουν τον τρόπο με τον οποίο λειτουργεί.

Επιπροσθέτως, ο Fowler ομαδοποιεί τις διαφορετικές αναδομήσεις με βάση τη λειτουργικότητα τους στις ακόλουθες κατηγορίες : σύνθεσης μεθόδων (composing methods) προκειμένου να γίνεται καλύτερη κατανομή του κώδικα, μεταφοράς γνωρισμάτων μεταξύ αντικειμένων (moving features between objects), οργάνωσης δεδομένων (organizing data) που διευκολύνουν την ενασχόληση με δεδομένα, απλοποίησης υποθετικών εκφράσεων (simplifying conditional expressions), απλοποίησης κλήσεων μεθόδων (making method calls simpler), σχετικές με τη γενίκευση (dealing with generalization), αναδομήσεις μεγάλης κλίμακας (big refactorings) (Πίνακας 1).

Κατηγορία	Αναδομήσεις
Composing methods	Extract Method Inline Method Replace Temp with Query
Moving features between objects	Move Method Move Field Extract Class Inline Class
Organizing data	Encapsulate Field Replace Type Code with Subclasses Replace Type Code with State/Strategy
Simplifying conditional expressions	Decompose Conditional Replace Conditional With Polymorphism Introduce Null Object
Making method calls simpler	Rename Method Add Parameter Remove Parameter Introduce Parameter Object Replace Constructor with Factory Method
Dealing with generalization	Pull up Field Pull up Method Push Down Field Push Down Method Extract Subclass Extract superclass Extract Interface Form Template Method Replace Inheritance with Delegation Replace Delegation with Inheritance
Big Refactorings	Tease Apart Inheritance Extract Hierarchy Separate Domain From Presentation

Πίνακας 1 : Κατηγοριοποίηση αναδομήσεων του Fowler

Ένα παράδειγμα των αναδομήσεων του Fowler απεικονίζεται στην Εικόνα 2, όπου προκειμένου να επιτευχθεί μεγαλύτερη γενίκευση και επαναχρησιμοποίηση κώδικα, οι κοινές μέθοδοι των υποκλάσεων, μεταφέρονται έπειτα από την εφαρμογή της αναδόμησης στην αμέσως επόμενη υπερκλάση (αναδόμηση pull up method).



Εικόνα 2 : Παράδειγμα πριν και μετά την αναδόμηση "Pull up method"

2.1.3. Τρόποι και Εργαλεία Αναδόμησης

Η αναδόμηση αποτελεί ιδιαίτερα απαιτητική διαδικασία, τόσο αναφορικά με την απαιτούμενη προσπάθεια όσο και σχετικά με το χρόνο που πρέπει να δαπανηθεί για την ολοκλήρωση της. Οι απαιτήσεις αυτές σε συνδυασμό με το διαρκώς αυξανόμενο μέγεθος των σύγχρονων συστημάτων λογισμικού καθιστούν δύσκολη την *αυτόνομη χειρονακτική (manual)* διενέργεια της από τους μηχανικούς λογισμικού, οι οποίοι συνήθως αξιοποιούν εμπειρικούς κανόνες καθώς και κανόνες που αναφέρονται στη σχετική βιβλιογραφία για την εφαρμογή της. Προκειμένου να διευκολυνθεί αυτό το δύσκολο έργο έχει αναπτυχθεί μια πληθώρα από εργαλεία, τα οποία αφενός συμβάλουν στην αύξηση της ταχύτητας με την οποία εκτελούνται οι απαιτούμενοι μετασχηματισμοί, αφετέρου παρέχουν ασφάλεια, μειώνοντας τον κίνδυνο εισαγωγής σφαλμάτων κατά τη διενέργεια τους, διασφαλίζοντας παράλληλα την διατήρηση της παρεχόμενης λειτουργικότητας. Τα εργαλεία που έχουν αναπτυχθεί γι’ αυτό το σκοπό, υποστηρίζουν τη διαδικασία αναδόμησης τόσο σε επίπεδο αυτόματης ανίχνευσης των υποψήφιων προς αναδόμηση τμημάτων κώδικα όσο και σε επίπεδο αυτόματης εκτέλεσης του καθεαυτού μετασχηματισμού. Ανάλογα με τις δυνατότητες που παρέχουν διακρίνονται σε

- ***ημιαυτόματα*** *(semi-automatic),* όταν απαιτείται η συμβολή των μηχανικών λογισμικού για την υπόδειξη των συγκεκριμένων βασικών μετασχηματισμών που θα εκτελεστούν καθώς και των τμημάτων κώδικα στα οποία θα εφαρμοστούν και
- ***πλήρως αυτοματοποιημένα*** *(full-automatic),* όταν τόσο η ανίχνευση των υποψήφιων προς αναδόμηση σημείων, όσο και η εφαρμογή της αναδόμησης υλοποιείται εξ' ολοκλήρου από το εκάστοτε εργαλείο.

Στο σημείο αυτό κρίνεται αναγκαίο να τονιστεί πως η διαδικασία χειρονακτικής αναδόμησης διαφέρει από την αυτοματοποιημένη αναδόμηση. Διαφορές εντοπίζονται σε καθένα από τα σταδία που αναφέρθηκαν στην ενότητα 2.1.1, ξεκινώντας από το στάδιο της ανίχνευσης, όπου τα εργαλεία αναδόμησης δεν είναι δυνατό να ανιχνεύσουν τα bad smells και χρησιμοποιούν τεχνικές βάσει μετρικών, λογικών κανόνων, ή άλλου είδους τεχνικές για τον εντοπισμό των σημείων εφαρμογής της αναδόμησης.

Ακολούθως στο στάδιο επιλογής της κατάλληλης αναδόμησης συνήθως απαιτείται η συμμετοχή του χρήστη. Παρά τη διάκριση των εργαλείων υποστήριξης της αναδόμησης σε ημιαυτόματα και πλήρως αυτοματοποιημένα, η συμμετοχή των μηχανικών λογισμικού είναι αδύνατο να εξαλειφθεί πλήρως. Η αδυναμία αυτή οφείλεται σε διάφορους παράγοντες, ο κυριότερος από τους οποίους σχετίζεται με τη σημασιολογική φύση πολλών σχεδιαστικών προβλημάτων και την ύπαρξη πολλών εναλλακτικών τρόπων βελτίωσης της ποιότητας του σχεδίου, η οποία καθιστά αναγκαία την επισκόπηση και επιλογή των καταλληλότερων προτάσεων αναδόμησης με χρήση της ανθρώπινης λογικής βάσει της υπό εξέταση εφαρμογής. Επιπλέον, η ανθρώπινη παρέμβαση καθίσταται απαραίτητη για τον έλεγχο των εργαλείων, καθώς και για τον καθορισμό των τιμών των παραμέτρων που απαιτούνται συνήθως κατά τη διαδικασία της αναδόμησης. Παράδειγμα περίπτωσης καθορισμού παραμέτρων κατά την αναδόμηση αποτελεί ο προσδιορισμός του επιθυμητού ονόματος μιας κλάσης κατά τη μετονομασία της.

Αναφορικά με τον έλεγχο και την παροχή εχεγγύων σχετικά με τη διατήρηση της συμπεριφοράς, τα εργαλεία αυτόματης αναδόμησης διαφυλάσσουν τον κώδικα από τροποποιήσεις και πιθανή εισαγωγή σφαλμάτων, ελέγχοντας πριν από την εφαρμογή της αναδόμησης, ένα σύνολο από προϋποθέσεις που θα πρέπει να ικανοποιούνται προκειμένου να συνεχιστεί η διαδικασία του μετασχηματισμού. Αντιθέτως, ο έλεγχος διασφάλισης διατήρησης της συμπεριφοράς[3] στην χειρονακτική εφαρμογή της αναδόμησης εισάγει

[3] Βλέπε βήμα 3 της διαδικασίας αναδόμησης, σελ. - 20 -

πρόσθετη πολυπλοκότητα στη διαδικασία καθώς απαιτεί την πρόσθετη εκτέλεση αυτόματων ελέγχων *(unit tests)* μετά από κάθε μετασχηματισμό, ούτως ώστε να διαπιστωθεί η απουσία σφαλμάτων.

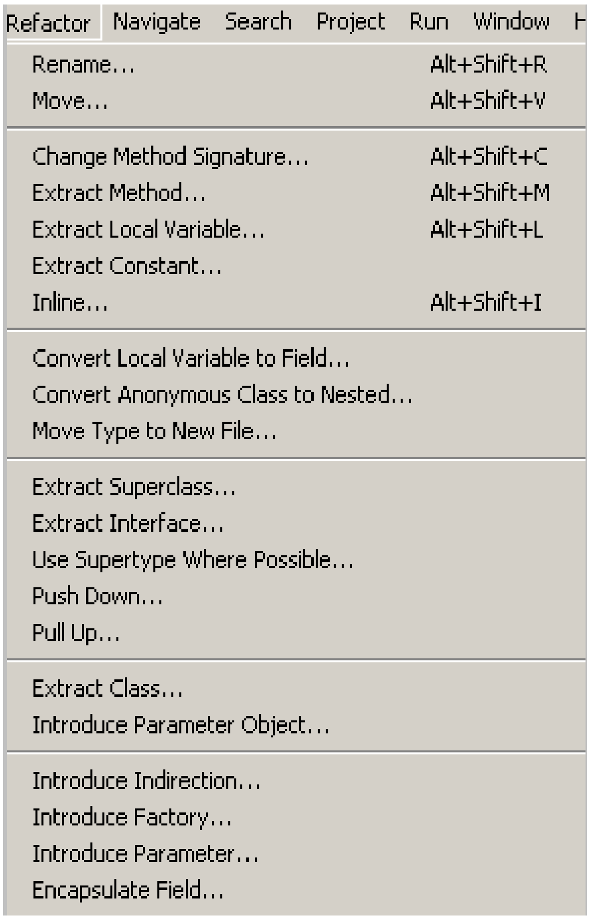
Διαφορές εντοπίζονται και στην αυτόματη εφαρμογή των μετασχηματισμών, οι οποίοι δεν εφαρμόζονται με την ίδια σειρά με την οποία ορίζονται τα βήματα των αναδομήσεων στα βιβλία των Fowler και Kerievsky. Άλλωστε ορισμένες από τις αλλαγές που περιγράφονται στα βήματα αυτά δεν είναι καν εφικτές με αυτοματοποιημένο τρόπο, καθώς δεν είναι εφικτός ο έλεγχος των προϋποθέσεων που θα πρέπει να πληρούνται προκειμένου να διενεργηθούν. Τέλος, τα εργαλεία αυτόματης αναδόμησης αξιολογούν τις προτάσεις αναδόμησης συνήθως βάσει μετρικών ή διαφόρων κριτηρίων που έχουν οριστεί κατά την υλοποίηση τους ενώ προς το παρόν, αδυνατούν να παρέχουν αυτόματη ενημέρωση των συμπληρωματικών εγγράφων του κώδικα.

Παρά τις αδυναμίες που παρουσιάζουν τα αυτοματοποιημένα εργαλεία αναδόμησης, η χρησιμότητα και αναγκαιότητα τους έγιναν αντιληπτές από πολύ νωρίς, κυρίως λόγω της ευχρηστίας τους, της αξιοπιστίας και της ασφάλειας που παρέχουν, της ταχύτητας που προσφέρουν, της κλιμακωσιμότητας τους σε μεγάλα συστήματα λογισμικού, της δυνατότητας επανάληψης μιας εφαρμοσμένης αναδόμησης, ή της ευκολίας στην αναίρεση μιας ήδη εφαρμοσμένης αναδόμησης που τελικά δεν κρίθηκε συνετή.

Συγκεκριμένα, η ανάγκη ανάπτυξης ενός αυτόματου εργαλείου που εκτελεί τις διάφορες αναδομήσεις με ασφαλή τρόπο καθώς και οι βασικές αρχές που σχετίζονται με τη δημιουργία του, αναφέρθηκαν για πρώτη φορά το 1992 στη διδακτορική διατριβή του W. Opdyke (7). Έκτοτε έχει σημειωθεί σημαντική πρόοδος στον τομέα αυτό με τη δημιουργία πληθώρας εργαλείων αναδόμησης για διάφορες γλώσσες προγραμματισμού : διαδικαστικές (procedural languages), συναρτησιακές (functional languages), λογικές (logic languages), αντικειμενοστρεφείς (object-oriented languages), aspect-oriented (aspect-oriented languages).

Δημοφιλέστερο εργαλείο αναδόμησης είναι το Smalltalk Refactoring Browser, το οποίο αποτελεί το πρώτο εργαλείο αναδόμησης που αναπτύχθηκε (18) (19). Την δημιουργία του Smalltalk Refactoring Browser ακολούθησαν στη συνέχεια και άλλα εργαλεία αναδόμησης, τα οποία συνήθως βρίσκονται ενσωματωμένα *(embedded)* στα ίδια τα περιβάλλοντα ανάπτυξης λογισμικού (*integrated development environment - IDE*) (20), ως μια από τις παρεχόμενες λειτουργίες τους, και υποστηρίζουν διάφορες γλώσσες προγραμματισμού.

Στον Πίνακας 2 παρουσιάζονται τα δημοφιλέστερα εργαλεία αναδόμησης για τις αντικειμενοστραφείς γλώσσες προγραμματισμού Java, .NET και C++. Τα εργαλεία που παρουσιάζονται στον Πίνακας 2 υποστηρίζουν αυτόματα αρκετές από τις απλές αναδομήσεις που περιγράφει στο βιβλίο του ο Fowler. Ωστόσο έλλειψη συναντάται αναφορικά με την υποστήριξη αυτοματοποιημένης εκτέλεσης σύνθετων αναδομήσεων όπως είναι οι αναδομήσεις με πρότυπα σχεδίασης, ακόμα και σε μεγάλα περιβάλλοντα ανάπτυξης λογισμικού, όπως το Eclipse (Εικόνα 3), όπου η μοναδική σύνθετη αναδόμηση που υποστηρίζεται αυτόματα είναι η *εισαγωγή εργοστασίου (introduce factory)*. Την έλλειψη σχετικά με την αυτόματη υποστήριξη αναδομήσεων με πρότυπα σχεδίασης αποπειράται να μειώσει η παρούσα εργασία μελετώντας την αυτοματοποίηση της αναδόμησης με το πρότυπο στρατηγική.



Εικόνα 3 : Επιλογές αναδόμησης στο Eclipse

Γλώσσα Προγραμματισμού	Εργαλεία υποστήριξης αναδόμησης
Java	Eclipse (21), Netbeans (22), IntelliJ (23), JDeodorant (24), JRefactory (25), Xrefactory (26), RefactorIt (27)
.NET	ReSharper (28), Refactor! Pro (29), JustCode! (30)
C++	Xrefactory (26), Visual Assist X (31), Refactor! (32)

Πίνακας 2 : OO γλώσσες προγραμματισμού και εργαλεία αναδόμησης

2.2 Πρότυπα Σχεδίασης

Τα *πρότυπα σχεδίασης (design patterns)* πρωτοεμφανίστηκαν ως μια ρηξικέλευθη ιδέα στον τομέα της *ανάπτυξης λογισμικού (Software Engineering)* το 1987, στο συνέδριο OOPSLA, όταν ο Kent Beck και ο Ward Cunnigham άρχισαν να πειραματίζονται με την εφαρμογή τους στον προγραμματισμό. Έπειτα από την έκδοση του βιβλίου *Design Patterns: Elements of Reusable Object-Oriented Software* από την αυτοαποκαλούμενη ομάδα «Gang of Four», των συγγραφέων Erich Gamma, Richard Helm, Ralph Johnson και John Vlissides (33), κατόρθωσαν να κεντρίσουν το ενδιαφέρον της επιστημονικής κοινότητας, πρωταγωνιστώντας σε αρκετές ερευνητικές δημοσιεύσεις, και να γίνουν ιδιαίτερα δημοφιλή.
Τα πρότυπα σχεδίασης παρέχουν ένα κοινό λεξιλόγιο για την επικοινωνία μεταξύ των μηχανικών λογισμικού. Αποτελούν έτοιμες ή σχεδόν έτοιμες, υψηλού επιπέδου σχεδιαστικές λύσεις σε κοινά και επαναλαμβανόμενα προβλήματα σχεδίασης. Ως εκ τούτου η αξιοποίηση τους διευκολύνει το έργο των μηχανικών λογισμικού οδηγώντας σε υψηλής ποιότητας λογισμικό.

Στο βιβλίο της συγγραφικής ομάδας GoF καταγράφονται 23 πρότυπα σχεδίασης, ταξινομημένα είτε βάσει του *σκοπού (purpose)* για τον οποίο χρησιμοποιούνται είτε βάσει της *εμβέλειας (scope)* τους, που αντανακλά τη χρήση του εκάστοτε προτύπου σε κλάσεις ή αντικείμενα (Πίνακας 3).

Η κατηγοριοποίηση που προκύπτει βάσει του σκοπού κατατάσσει τα πρότυπα σχεδίασης σε *κατασκευαστικά (creational),* τα οποία σχετίζονται με διαδικασίες δημιουργίας αντικειμένων, *δομικά (structural),* που ασχολούνται με τη σύνθεση κλάσεων ή αντικειμένων, *και* *συμπεριφορικά (behavioural,)* που προσδιορίζουν τον τρόπο με τον οποίο τα αντικείμενα αλληλεπιδρούν μεταξύ τους και οι αρμοδιότητες διανέμονται σε κλάσεις (34).

Σύμφωνα με το κριτήριο της εμβέλειας τα πρότυπα κατηγοριοποιούνται σε *πρότυπα κλάσης (class patterns),* τα οποία αφορούν τις συσχετίσεις κληρονομικότητας που καθορίζονται στατικά μεταξύ κλάσεων και των υποκλάσεων τους κατά το *χρόνο μεταγλώττισης (compilation-time),* και *πρότυπα αντικειμένου (object patterns),* που αφορούν τις συσχετίσεις αντικειμένων που καθορίζονται δυναμικά κατά το *χρόνο εκτέλεσης (run-time).*

Τα οφέλη των προτύπων σχεδίασης αξιοποιούνται τόσο κατά τη φάση της ανάπτυξης όσο και κατά τη φάση της συντήρησης, μέσω των κατάλληλων αναδομήσεων του πηγαίου κώδικα, προκειμένου να βελτιώσουν την ποιότητα του παραγόμενου σχεδίου. Στην παρούσα εργασία μελετώνται το πρότυπο *Στρατηγική* συγκριτικά με το πρότυπο *Κατάσταση* και οι σχετικές τους αναδομήσεις προκειμένου να διαφοροποιηθούν.

Τα πρότυπα *Κατάσταση* και *Στρατηγική* παρότι παρουσιάζουν πολλές ομοιότητες στη δομή και στον τρόπο χρήσης τους, διαφέρουν. Χαρακτηριστικά στο βιβλίο του Eric Freeman *Head First Design Patterns* (35) τα δύο πρότυπα αναφέρονται ως «δίδυμα που χωρίστηκαν στη γέννηση τους». Καθίσταται λοιπόν αναγκαίο να παρουσιαστούν ενδελεχώς και να εντοπιστούν τα στοιχεία εκείνα στα οποία τα δύο πρότυπα σχεδίασης διαφοροποιούνται.

		Σκοπός		
		Κατασκευαστικά	Δομικά	Συμπεριφορικά
Εμβέλεια	Κλάσης	**Factory Method** (Μέθοδος Εργοστάσιο)	**Adapter** (Προσαρμογέας)	**Interpreter** (Διερμηνέας) **Template Method** (Μέθοδος Υπόδειγμα)
	Αντικειμένου	**Abstract Factory** (Αφηρημένο Εργοστάσιο) **Builder** (Κτίστης) **Prototype** (Πρωτότυπο) **Singleton** (Μοναδιαίο)	**Adapter** (Προσαρμογέας) **Bridge** (Γέφυρα) **Composite** (Σύνθετο) **Decorator** (Διακοσμητής) **Façade** (Πρόσοψη) **Proxy** (Αντιπρόσωπος)	**Chain of Responsibility** (Αλληλουχία Αρμοδιοτήτων) **Command** (Εντολή) **Iterator** (Επαναλήπτης) **Mediator** (Μεσολαβητής) **Memento** (Υπόμνηση) **Flyweight** **Observer** (Παρατηρητής) **State** (Κατάσταση) **Strategy** (Στρατηγική) **Visitor** (Επισκέπτης)

Πίνακας 3 : Κατηγοριοποίηση των Προτύπων Σχεδίασης

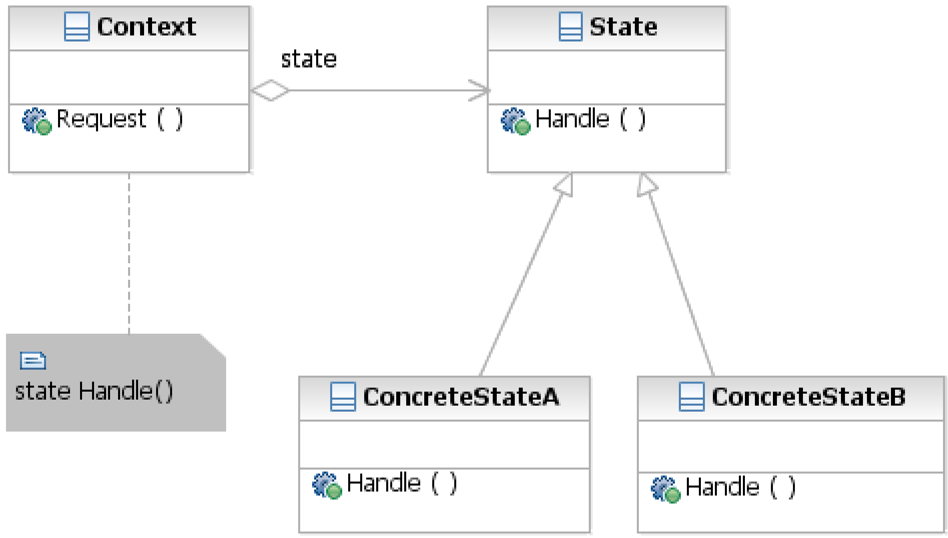
2.2.1. Πρότυπο Κατάσταση

Το πρότυπο *κατάσταση* (State) ανήκει βάσει της κατηγοριοποίησης των GoF (33) στα συμπεριφορικά (behavioral) πρότυπα. Σκοπός του προτύπου είναι να καταστήσει δυνατή την αλλαγή συμπεριφοράς ενός αντικειμένου, όταν αλλάζει και η εσωτερική του κατάσταση, κατά τρόπο ώστε να φαίνεται ότι αλλάζει η κλάση του.

Η χρήση του προτύπου αυτού ενδείκνυται όταν η συμπεριφορά ενός αντικειμένου καθορίζεται συναρτήσει της κατάστασης του και τροποποιείται κατά το χρόνο εκτέλεσης, σύμφωνα με τις μεταβάσεις των καταστάσεων του. Επιπλέον, η εφαρμογή του προτύπου συνίσταται σε περιπτώσεις όπου συναντώνται στον κώδικα εκτενείς, πολύτμηματικές υποθετικές δηλώσεις που εξαρτώνται από την κατάσταση του αντικειμένου. Στην περίπτωση αυτή, το πρότυπο κατάσταση, αντικαθιστά τις υποθετικές δηλώσεις που δυσχεραίνουν τη συντηρησιμότητα του κώδικα, αξιοποιώντας τις έννοιες της *κληρονομικότητας (inheritance)* και του *πολυμορφισμού (polymorphism)*, συμβάλλοντας κατ' αυτό τον τρόπο στην μείωση της πολυπλοκότητας του συστήματος.

Άλλα πλεονεκτήματα που πηγάζουν από τη χρήση του προτύπου είναι η αύξηση της κατανοησιμότητας και της ευαναγνωσιμότητας του κώδικα, μέσω της βελτίωσης της δομής του. Μέσω του προτύπου κατάσταση κάθε ξεχωριστή συμπεριφορά, δεν βρίσκεται διασκορπισμένη σε διάφορα σημεία, αλλά διαχωρίζεται από τις υπόλοιπες και εντοπίζεται στην αντίστοιχη υποκλάση της εκάστοτε κατάστασης. Κατ' αυτό τον τρόπο αντανακλάται σαφέστερα ο σκοπός του κώδικα, ενώ παράλληλα διευκολύνεται η επέκταση της τρέχουσας λειτουργικότητας με την προσθήκη νέων καταστάσεων και συμπεριφορών, η οποία επιτυγχάνεται ταχύτατα μέσω της προσθήκης νέων κλάσεων. Ένα ακόμα πλεονέκτημα της χρήσης του προτύπου αυτού, είναι ότι καθιστά σαφέστερη τη μετάβαση καταστάσεων, καθώς κάθε διαφορετική κατάσταση σχετίζεται με ένα ξεχωριστό αντικείμενο. Επιπλέον, δεδομένου ότι κάθε μετάβαση καταστάσεων που προκύπτει είναι ατομική, η κλάση Context προστατεύεται από ασυνεπείς εσωτερικές καταστάσεις.

Ακολούθως απεικονίζονται τα δομικά στοιχεία από τα οποία απαρτίζεται το πρότυπο (Εικόνα 4). Η κλάση *Context* αποτελεί τη διεπαφή για τους πελάτες που χαρακτηρίζεται από μια κατάσταση. Η κλάση *State (κατάσταση)* παρέχει τη διεπαφή για την ενθυλάκωση της συμπεριφοράς που σχετίζεται με μια συγκεκριμένη κατάσταση της κλάσης Context. Τέλος, οι υποκλάσεις *ConcreteStateA, ConcreteStateB* παρέχουν την υλοποίηση μιας συμπεριφοράς που συσχετίζεται με μια κατάσταση της κλάσης Context.



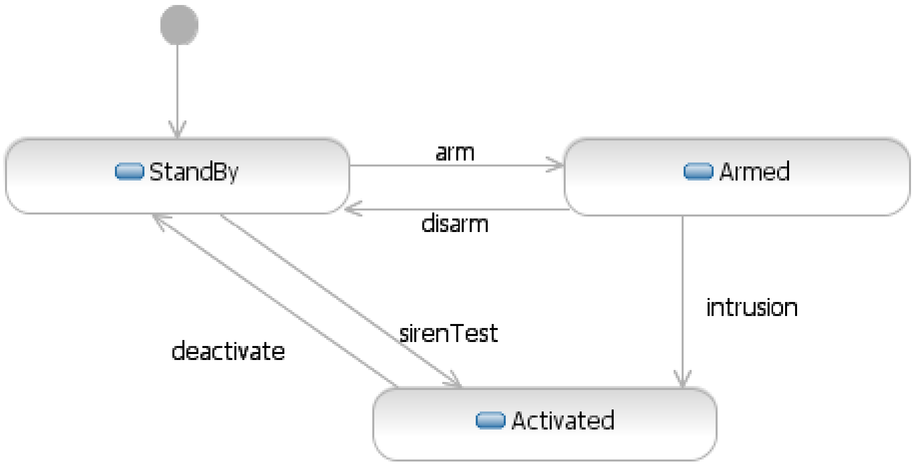
Εικόνα 4 : Η δομή του προτύπου Κατάσταση

Παρόλο που οι GoF (33) στο βιβλίο τους σχετικά με τα πρότυπα σχεδίασης ορίζουν τη δομή του προτύπου κατάσταση, ορισμένα ζητήματα υλοποίησης δεν είναι προκαθορισμένα αλλά επαφίενται στην κρίση και την επιλογή του εκάστοτε προγραμματιστή, ανάλογα με τις ειδικές απαιτήσεις της εφαρμογής που αναπτύσσει. Το γεγονός αυτό οφείλεται άλλωστε και στην ίδια τη φύση των προτύπων σχεδίασης, ως γενικές - αφαιρετικές σχεδιαστικές λύσεις. Τέτοια ζητήματα αφορούν στην υπεύθυνη κλάση για τον προσδιορισμό των μεταβάσεων καταστάσεων, που μπορεί να επιτευχθεί είτε από την κλάση Context, είτε εναλλακτικά από τις διαφορετικές υποκλάσεις κατάστασης, στη δημιουργία των αντικειμένων κατάστασης εκ των προτέρων, ή κατά τη στιγμή που απαιτούνται, και την άμεση καταστροφή τους έπειτα από τη χρήση και στη υλοποίηση της κατάστασης είτε ως διεπαφή, είτε ως αφηρημένη υποκλάση.

Ακολούθως παρατίθεται ένα παράδειγμα μιας περίπτωσης στην οποία βρίσκει εφαρμογή το πρότυπο κατάσταση (34). Στο συγκεκριμένο παράδειγμα μοντελοποιείται ένα σύστημα συναγερμού το οποίο χαρακτηρίζεται από τρείς διακριτές καταστάσεις:

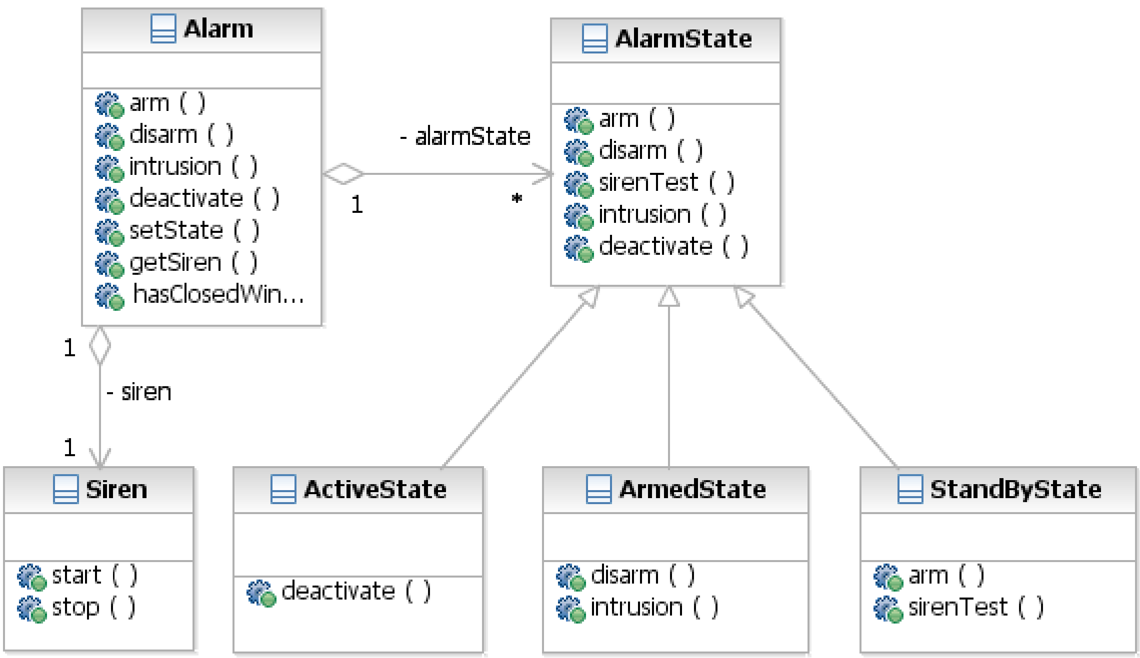
- κατάσταση αναμονής (standby)
- κατάσταση οπλισμένο (armed) και
- κατάσταση ενεργοποιημένο (activated).

Οι καταστάσεις αυτές, καθώς και οι αντίστοιχες μεταβάσεις μεταξύ τους, απεικονίζονται στο ακόλουθο διάγραμμα μηχανής καταστάσεων (Εικόνα 5).



Εικόνα 5 : Διάγραμμα Μηχανής Καταστάσεων της κλάσης Alarm

Με χρήση του προτύπου κατάσταση, καθεμία από τις διακριτές αυτές καταστάσεις αναπαριστάται από μια ξεχωριστή υποκλάση της κλάσης AlarmState, που αποτελεί τη διεπαφή για την ενθυλάκωση της συμπεριφοράς, ενώ οι διαφορετικές συμπεριφορές μοντελοποιούνται από διαφορετικές μεθόδους στο σώμα των υποκλάσεων και της κλάσης AlarmState (Εικόνα 6). Το σύστημα συναγερμός (κλάση Alarm) αντιστοιχεί με βάση τη δομή του προτύπου στην κλάση Context, ενώ η κλάση Siren δεν αντιστοιχεί σε κάποιο δομικό στοιχείο του προτύπου, αλλά η ύπαρξη της συνεισφέρει στην πληρότητα του παραδείγματος. Τέλος, στις Εικόνα 60 έως Εικόνα 65 του Παράρτημα Α παρατίθεται ο πηγαίος κώδικας για καθεμία από τις κλάσεις του παραδείγματος.



Εικόνα 6 : Διάγραμμα Κλάσεων του παραδείγματος συστήματος συναγερμού

2.2.2. Πρότυπο Στρατηγική

Το πρότυπο *στρατηγική* (Strategy) ανήκει επίσης στα συμπεριφορικά (behavioral) πρότυπα. Σκοπός του προτύπου είναι ο καθορισμός μιας οικογένειας αλγορίθμων και η ενθυλάκωση τους ώστε να είναι εναλλάξιμοι.

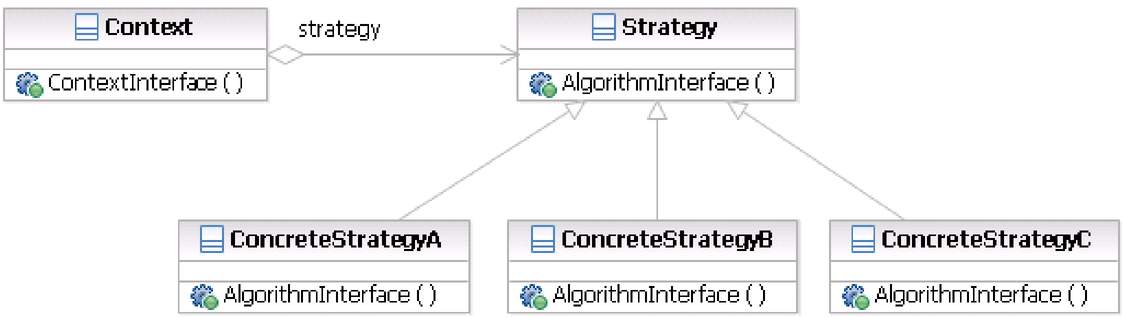
Το πρότυπο στρατηγική χρησιμοποιείται στην περίπτωση όπου πολλές συναφείς κλάσεις διαφέρουν μόνο στη συμπεριφορά ή εναλλακτικά, όταν απαιτούνται πολλές διαφορετικές παραλλαγές ενός αλγορίθμου. Επιπλέον, η εφαρμογή του προτύπου ενδείκνυται σε περιπτώσεις όπου καθίσταται αναγκαία η απόκρυψη δεδομένων από τους πελάτες, για την αποφυγή της έκθεσης πολύπλοκων αλγοριθμικών δομών δεδομένων. Μια ακόμη πιθανή χρήση της στρατηγικής, είναι η απλοποίηση υποθετικών δηλώσεων με παρόμοια συμπεριφορά. Το πρότυπο στρατηγική υποκαθιστά την υποθετική λογική, μεταφέροντας τις δηλώσεις που περιέχονται στις διακλαδώσεις της υποθετικής δομής, σε ξεχωριστές μεθόδους διαφορετικών κλάσεων.

Η χρήση του προτύπου στρατηγική κατά την ανάπτυξη λογισμικού παρέχει αρκετά σημαντικά οφέλη, έχει όμως και ορισμένα μειονεκτήματα. Αφενός μέσω της αξιοποίησης της κληρονομικότητας, προσδιορίζεται η κοινή λειτουργικότητα των αλγορίθμων και ορίζονται διαφορετικές οικογένειες αλγορίθμων ως ιεραρχίες κλάσεων στρατηγικής. Κατ' αυτό τον τρόπο παρέχεται στους πελάτες η δυνατότητα επιλογής μεταξύ των διαφορετικών αλγορίθμων. Παράλληλα η χρήση της κληρονομικότητας αυξάνει την ευελιξία του συστήματος, καθώς οι αλγόριθμοι αποσυσχετίζονται από την κλάση context. Συνεπώς, διευκολύνεται η τροποποίηση και επαύξηση τους, ενώ παράλληλα βελτιώνεται και η κατανοησιμότητα και η συντηρησιμότητα του κώδικα. Επιπροσθέτως, με την ενθυλάκωση της συμπεριφοράς σε διαφορετικές κλάσεις στρατηγικής, το πρότυπο συμβάλει στη μείωση των υποθετικών δηλώσεων στον κώδικα, που συχνά αποτελούν αιτία πρόσθετης πολυπλοκότητας του υπό ανάπτυξη συστήματος.

Αφετέρου όμως, με την υλοποίηση των παραλλαγών των αλγορίθμων ως ξεχωριστές κλάσεις, αυξάνεται ο αριθμός των αντικειμένων στο σύστημα, ενώ ενδέχεται να προκύψει και αυξημένη επιβάρυνση από την επικοινωνία μεταξύ της κλάσης Context και της στρατηγικής. Η επιβάρυνση αυτή προκύπτει από το ενδεχόμενο, η κλάση context να δημιουργεί και να αρχικοποιεί παραμέτρους που απαιτούνται λόγω της κοινής διεπαφής των διαφορετικών στρατηγικών, δεν χρησιμοποιούνται όμως κατ' ανάγκη από την εκάστοτε στρατηγική. Ταυτόχρονα, πιθανό μειονέκτημα ενδέχεται να αποτελέσει το γεγονός ότι οι πελάτες θα πρέπει να γνωρίζουν τις εναλλακτικές στρατηγικές καθώς και τις διαφορές τους, προκειμένου να είναι σε θέση να επιλέξουν την κατάλληλη.

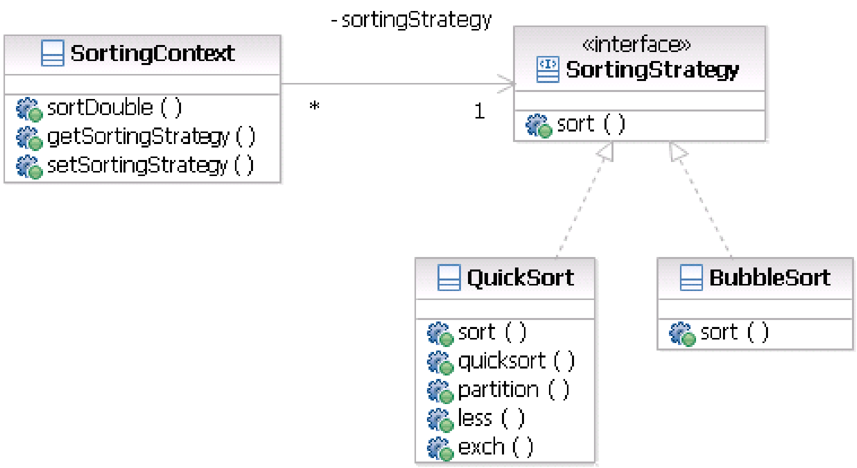
Εν συνεχεία στην Εικόνα 7 παρουσιάζεται η δομή του προτύπου στρατηγική, η οποία ουσιαστικά είναι όμοια με τη δομή του προτύπου κατάσταση. Η κλάση *Strategy* (στρατηγική) παρέχει την κοινή διεπαφή για κάθε εναλλακτικό αλγόριθμο που υποστηρίζεται, ενώ οι κλάσεις *ConcreteStrategyA*, *ConcreteStrategyB*, *ConcreteStrategyC*, κ.ο.κ. αποτελούν τις διαφορετικές υλοποιήσεις του αλγορίθμου. Τέλος, η κλάση *Context* προσδιορίζει τον επιθυμητό αλγόριθμο, χρησιμοποιώντας τη διεπαφή Strategy.

Όπως και στην περίπτωση του προτύπου κατάσταση, ορισμένα ζητήματα υλοποίησης δεν προσδιορίζονται σαφώς, αλλά παραμένουν ανοικτά ούτως ώστε να προσαρμόζονται ανάλογα με τις απαιτήσεις της εκάστοτε εφαρμογής. Παράδειγμα τέτοιου ζητήματος υλοποίησης αποτελεί ο προσδιορισμός της διεπαφής της στρατηγικής και της κλάσης context, κατά τρόπο ώστε η στρατηγική είτε να δέχεται όλα τα απαραίτητα για την εκτέλεση του αλγορίθμου δεδομένα, μέσω ξεχωριστών παραμέτρων (μειωμένη σύζευξη (coupling) μεταξύ των κλάσεων), είτε εναλλακτικά να δέχεται ως παράμετρο την ίδια την κλάση context, η διεπαφή της οποίας θα παρέχει τη δυνατότητα πρόσβασης σε όλα τα απαραίτητα δεδομένα (αυξημένη σύζευξη μεταξύ των κλάσεων). Ένα ακόμα ζήτημα υλοποίησης αποτελεί ο προσδιορισμός μιας προεπιλεγμένης συμπεριφοράς για την κλάση context, ούτως ώστε η στρατηγική να χρησιμοποιείται προαιρετικά, μόνο στην περίπτωση που η προεπιλεγμένη συμπεριφορά δεν είναι η επιθυμητή.



Εικόνα 7 : Η δομή του προτύπου Στρατηγική

Ακολούθως παρατίθεται ένα παράδειγμα μιας περίπτωσης στην οποία βρίσκει εφαρμογή το πρότυπο στρατηγική (36). Στο συγκεκριμένο παράδειγμα το πρότυπο στρατηγική αξιοποιείται προκειμένου να υλοποιηθούν εναλλακτικές παραλλαγές ενός αλγορίθμου ταξινόμησης, τις οποίες χρησιμοποιεί η κλάση SortingContext, κατ' αντιστοιχία με την κλάση Context του προτύπου. Οι εναλλακτικές παραλλαγές του αλγορίθμου ταξινόμησης, στην προκειμένη περίπτωση ο αλγόριθμος ταξινόμησης φυσαλίδας (bubble-sort) και ο αλγόριθμος ταξινόμησης με συγχώνευση (merge-sort), μοντελοποιούνται από τις διαφορετικές κλάσεις στρατηγικής, που υλοποιούν τη διεπαφή SortingStrategy (Εικόνα 8). Η διεπαφή SortingStrategy ορίζει τη στρατηγική για τη λειτουργία της ταξινόμησης (sort), ενώ η επιλογή της κατάλληλης στρατηγικής επιτυγχάνεται πολυμορφικά. Στις Εικόνα 66 έως Εικόνα 69 του Παράρτημα Β παρατίθεται ο πηγαίος κώδικας για καθεμία από τις κλάσεις του παραδείγματος.



Εικόνα 8 : Διάγραμμα κλάσεων για το παράδειγμα χρήσης της στρατηγικής

2.2.3. Σύγκριση Προτύπων Κατάσταση και Στρατηγική

Τα πρότυπα σχεδίασης *Κατάσταση* και *Στρατηγική* ανήκουν και τα δύο στην κατηγορία των *συμπεριφορικών προτύπων* (*behavioural patterns)* και είναι πανομοιότητα αναφορικά με τη δομή τους, αξιοποιώντας τις έννοιες της *σύνθεσης (composition)* και του *πολυμορφισμού (polymorphism),* προκειμένου να αποφευχθεί η χρήση πολύπλοκης υποθετικής λογικής. Το γεγονός αυτό συχνά προκαλεί σύγχυση των δύο πρότυπων και καθιστά αναγκαία την αποσαφήνιση τους.

Ουσιαστική διαφορά των δύο προτύπων συντελεί η διαφορά στο σκοπό τους. Η χρήση του προτύπου *Κατάσταση* ενδείκνυται όταν ένα αντικείμενο που χαρακτηρίζεται από μια κατάσταση, αλλάζει συμπεριφορά ανάλογα με τις μεταβάσεις των καταστάσεων του. Αντίστοιχα, το πρότυπο *Στρατηγική* στοχεύει στην ενθυλάκωση εναλλακτικών τρόπων υλοποίησης ενός αλγορίθμου, προκειμένου να μπορεί να επιλεγεί εύκολα ο καταλληλότερος. Στη *Στρατηγική* όμως η επιλογή του αλγορίθμου γίνεται εκ των προτέρων, κατά τη φάση της παραμετροποίησης και συνήθως έπειτα δε μεταβάλλεται κατά το χρόνο εκτέλεσης, ούτε τροποποιείται για κάθε αντικείμενο Context, ενώ στην *Κατάσταση,* οι μεταβάσεις των καταστάσεων που ακολουθούνται και από μεταβολές στη συμπεριφορά, συμβαίνουν σύγχρονα, κατά το χρόνο εκτέλεσης (37) (35). Συνεπώς παρότι και τα δύο πρότυπα είναι εναλλάξιμα σύμφωνα με τη δομή τους, η εναλλαγή στην περίπτωση της *Στρατηγικής* σπάνια εφαρμόζεται στην πράξη.

Μια επιπλέον διαφοροποίηση των δύο προτύπων έγκειται στο γεγονός ότι στη *Στρατηγική* η επιλογή του εκάστοτε αλγορίθμου επαφίεται στον πελάτη, ο οποίος γνωρίζει την ύπαρξη των διαφορετικών στρατηγικών (33). Αντίθετα, στην *Κατάσταση* ο πελάτης αγνοεί τις διαφορετικές καταστάσεις και δεν δύναται να καθορίσει την εκάστοτε κατάσταση, καθώς αλληλλεπιδρά μόνο με την κλάση Context. Εφόσον, οι διάφορες καταστάσεις εναλλάσσονται μεταξύ τους καθορίζοντας την κατάσταση στην οποία μεταβαίνει η κλάση Context, υποννοείται έμμεσα και η γνώση των διαφορετικών καταστάσεων μεταξύ τους (33), γεγονός που δεν ισχύει στη *Στρατηγική*, όπου οι διάφορες στρατηγικές είναι πλήρως ανεξάρτητες και αγνοούν την ύπαρξη των υπολοίπων.

Και στα δύο πρότυπα, προκειμένου η κλάση Context να μπορέσει να αλληλεπιδράσει με τις καταστάσεις / στρατηγικές θα πρέπει, είτε να περνάει τον εαυτό της ως παράμετρο, είτε να περνάει όλες τις απαιτούμενες παραμέτρους για την εκάστοτε διάδραση. Παρόλο που στη *Στρατηγική* κριτήριο για την επιλογή ανάμεσα στις δύο εναλλακτικές αποτελεί ο αριθμός τον απαιτούμενων παραμέτρων, στην *Κατάσταση* φαίνεται πώς η προτιμώμενη λύση, ως επί το πλείστον, είναι να οριστεί ως παράμετρος ολόκληρο το αντικείμενο Context. Αυτό οφείλεται στη φύση του προτύπου, όπου οι καταστάσεις είναι υπεύθυνες για τη διαδοχή των καταστάσεων και τη μεταβολή της συμπεριφοράς της κλάσης Context, και συνεπώς απαιτείται να έχουν πρόσβαση στην κλάση Context (33), γεγονός που δεν είναι απαραίτητο στη Στρατηγική, όπου η ροή ελέγχου συνήθως είναι μονόδρομη και πηγάζει από την κλάση *Context*.

Επιπλέον διαφοροποίηση αποτελεί το γεγονός ότι στην *Κατάσταση*, οι κλάσεις καταστάσεις γνωρίζουν πλήρως τη συμπεριφορά του αντικειμένου Context και διαθέτουν μεθόδους χειρισμού κάθε πιθανής συμπεριφοράς του, εφόσον το αντικείμενο Context αναθέτει την ευθύνη (*delegation*) για την εκτέλεση τους στις καταστάσεις. Αντιθέτως, η *Στρατηγική* παρέχει στο αντικείμενο Context μια μόνο καθορισμένη πρόσθετη συμπεριφορά, ενώ όλες οι υπόλοιπες λειτουργίες παραμένουν στη δικαιοδοσία και την υπευθυνότητα του ίδιου του αντικειμένου Context, χωρίς η *Στρατηγική* να παρεμβαίνει, ούτε να διαθέτει γνώση γι' αυτές. Επιπλέον, ενώ όλες οι κλάσεις στρατηγικής υλοποιούν την ίδια συμπεριφορά με διαφορετικό τρόπο, οι κλάσεις κατάστασης αναπαριστούν διαφορετικές περιπτώσεις υλοποιώντας διαφορετικές κάθε φορά ενέργειες.

Επιπροσθέτως, σύμφωνα με το R. Johnson (38) (39), τα αντικείμενα καταστάσεις συνήθως δε διαθέτουν μεταβλητές μέλους (instance variables), ενώ διαθέτουν αρκετές ασυσχέτιστες μεταξύ τους μεθόδους με χαμηλή *σύζευξη (coupling)* ή *συνεκτικότητα (cohesion)*. Αντιθέτως, τα αντικείμενα στρατηγικές συνήθως διαθέτουν instance variables ενώ αποτελούνται από ορισμένες δημόσιες μεθόδους που καλούν ένα πλήθος από ιδιωτικές μεθόδους με υψηλή συνεκτικότητα μεταξύ τους.

Στον Πίνακας 4 συνοψίζονται οι διαφορές των προτύπων Κατάσταση και Στρατηγική.

Κατάσταση	Στρατηγική
Το πρότυπο *Κατάσταση* παρέχει ένα τρόπο υλοποίησης ενός αντικειμένου που η συμπεριφορά του βασίζεται στην κατάσταση του.	Το πρότυπο *Στρατηγική* παρέχει έναν τρόπο για την ενθυλάκωση διαφορετικών αλγορίθμων και τον εξαρχής προσδιορισμό του καταλληλότερου.
Ο πελάτης δε γνωρίζει τις διαφορετικές καταστάσεις. Η αλλαγή κατάστασης συμβαίνει αδιαφανώς για τον πελάτη. Η κλάση Context αλληλεπιδρά με τις κλάσεις καταστάσεις και τους αναθέτει αρμοδιότητες.	Ο πελάτης γνωρίζει τις διαφορετικές στρατηγικές και αλληλεπιδρά με την κλάση Context για να καθορίσει την στρατηγική με την οποία συσχετίζεται.
Οι καταστάσεις γνωρίζουν την ύπαρξη των υπόλοιπων καταστάσεων και αλληλεπιδρούν για να καθορίσουν οι ίδιες τη μετάβαση καταστάσεων εκ μέρους της κλάσης Context. Η κλάση Context αρχικοποιείται με μια αρχική κατάσταση, η οποία αλλάζει στο χρόνο εκτέλεσης καθώς η *Κατάσταση* βασίζεται στην εναλλαγή διακριτών καταστάσεων.	Οι διαφορετικές στρατηγικές δεν γνωρίζουν τις υπόλοιπες στρατηγικές και δεν αλληλεπιδρούν καθώς συνήθως η στρατηγική της κλάσης Context προσδιορίζεται κατά τη φάση παραμετροποίησης και δε μεταβάλλεται κατά το χρόνο εκτέλεσης, ούτε τροποποιείται ανάμεσα στα διαφορετικά αντικείμενα Context.
Η κλάση Context συνήθως περνάει τον εαυτό της ως παράμετρο στην *Κατάσταση.*	Η κλάση Context περνάει είτε τον εαυτό της, είτε τις εκάστοτε παραμέτρους στη *Στρατηγική*.
Οι καταστάσεις γνωρίζουν και διαθέτουν μεθόδους χειρισμού ολόκληρης της συμπεριφοράς της κλάσης Context. Οι καταστάσεις αναπαριστούν διαφορετικές περιπτώσεις με διαφορετική συμπεριφορά.	Οι στρατηγικές παρέχουν υλοποίηση μιας συγκεκριμένης πρόσθετης συμπεριφοράς της κλάσης Context, ενώ οι υπόλοιπες συμπεριφορές παραμένουν στη δικαιοδοσία της. Οι στρατηγικές αποτελούν διαφορετική υλοποίηση της ίδιας συμπεριφοράς.
Τα αντικείμενα καταστάσεις συνήθως δεν διαθέτουν πεδία (instance variables) και το σύνολο των μεθόδων δε σχετίζονται ιδιαίτερα μεταξύ τους (έχουν πολύ χαμηλή σύζευξη).	Τα αντικείμενα στρατηγικές συνήθως διαθέτουν πεδία (instance variables) και συχνά διαθέτουν πολλές δημόσιες μεθόδους που καλούν διάφορες ιδιωτικές μεθόδους. Υπάρχει υψηλή συνεκτικότητα μεταξύ των μεθόδων.

Πίνακας 4: Διαφορές Κατάστασης και Στρατηγικής

2.1 Αναδόμηση με Πρότυπα Σχεδίασης

Στη σύγχρονή ανάπτυξη λογισμικού, ολοένα και μεγαλύτερη έμφαση αποδίδεται στην ποιότητα του σχεδίου των συστημάτων λογισμικού και ιδιαίτερα στα αντικειμενοστραφή. Ευρέως αποδεκτή είναι η σημαντικότητα της κατασκευής ενός συστήματος λογισμικού με έμφαση στις έννοιες της συντηρησιμότητας και της επαναχρησιμοποίησης, έναντι της απλής υλοποίησης όλων των απαιτήσεων, χωρίς την λήψη μέριμνας για την εξασφάλιση των ποιοτικών χαρακτηριστικών του κώδικα. Στην κατεύθυνση αυτή για τη διασφάλιση της ποιότητας λογισμικού συμβάλλουν τα πρότυπα σχεδίασης, παρέχοντας μεγαλύτερη ευελιξία, επεκτασιμότητα και επαναχρησιμοποίηση των συστημάτων λογισμικού.

Ωστόσο, συχνά η πρόωρη υιοθέτηση των προτύπων σχεδίασης κατά τη φάση της σχεδίασης και της ανάπτυξης της πρώτης έκδοσης του συστήματος, έχει ως αποτέλεσμα την παραγωγή ενός συστήματος με πρόσθετη ευελιξία και πιο εξεζητημένη δομή απ' ότι απαιτείται *(over-engineering).* Από την άλλη, συνηθέστερό πρόβλημα αποτελεί η ελλιπής σχεδίαση *(under-engineering)* και η απουσία χρήσης των προτύπων, που συντελούν στην παραγωγή συστημάτων χαμηλής ποιότητας. Λύση στα προβλήματα αυτά καλείται να προσφέρει η αναδόμηση με πρότυπα σχεδίασης, ο *«γάμος της αναδόμησης - της διαδικασίας βελτίωσης του σχεδίου του υπάρχοντος κώδικα - με τα πρότυπα σχεδίασης – τις ευρέως αποδεκτές υψηλού-επιπέδου λύσεις σε κοινά επαναλαμβανόμενα προβλήματα σχεδίασης»,* όπως αναφέρεται στο βιβλίο του Kerievsky (16).

Στο βιβλίο των GoF τα πρότυπα σχεδίασης αναφέρονται ως στόχοι των αναδομήσεων (33), ενώ ο Fowler (8).αναφέρει πώς ανάμεσα στα πρότυπα σχεδίασης και την αναδόμηση υπάρχει μια φυσική σχέση:

> «Τα πρότυπα αποτελούν ένα επιθυμητό σημείο, οι αναδομήσεις είναι τρόποι να προσεγγιστεί από κάπου αλλού»

Η αναδόμηση με πρότυπα σχεδίασης, οδηγεί στην εφαρμογή των κατάλληλων προτύπων σχεδίασης σε μια μεταγενέστερη φάση του κύκλου ζωής του λογισμικού, και ιδιαίτερα κατά τη φάση της συντήρησης, όπου, τόσο οι υπάρχουσες όσο και οι μελλοντικές ανάγκες του συστήματος βραχυπρόθεσμά, μπορούν να προσδιοριστούν με μεγαλύτερη ακρίβεια. Συνεπώς η χρήση των προτύπων σχεδίασης για τη βελτίωση ενός υπάρχοντος σχεδίου θεωρείται καλύτερη από την εκ των προτέρων υιοθέτηση των προτύπων σε ένα καινούργιο σχέδιο. Για την επίτευξη της βελτίωσης της ποιότητας ενός υπάρχοντος σχεδίου, εφαρμόζονται ακολουθίες χαμηλού-επιπέδου μετασχηματισμών, γνωστές ως αναδομήσεις.

Η αναδόμηση με πρότυπα σχεδίασης συνδυάζει το όφελος από την εφαρμογή της αναδόμησης, με τα πλεονεκτήματα των προτύπων σχεδίασης. Η αναδόμηση αποτελεί μια σημαντική τεχνική για τη βελτίωση του υπάρχοντος σχεδίου, ενώ τα πρότυπα σχεδίασης μειώνουν τις εξαρτήσεις των επιμέρους συστατικών ενός προγράμματος, αυξάνοντας την ευελιξία του, την επεκτασιμότητα του, την συντηρησιμότητα και την επαναχρησιμοποιησιμότητα, μειώνοντας παράλληλα το χρόνο και το κόστος της ανάπτυξης και της συντήρησης.

Η αναδόμηση με πρότυπα σχεδίασης εμφανίστηκε για πρώτη φορά στο βιβλίο του Fowler, όπου μεταξύ των υπόλοιπων αναδομήσεων προτείνονται δύο τύποι αναδόμησης που αξιοποιούν τα πρότυπα σχεδίασης *κατάσταση/στρατηγική* και *μέθοδος εργοστάσιο*. Η αναδόμηση με χρήση της κατάστασης ή της στρατηγικής ονομάζεται από τον Fowler ως ***«Αντικατάσταση Κωδικού που υποδηλώνει τύπο με Κατάσταση/Στρατηγική» (Replace Type Code With State/Strategy)*** και χρησιμοποιείται όταν η συμπεριφορά μιας κλάσης καθορίζεται από την τιμή ενός πεδίου τύπου (type). Η εφαρμογή της αναδόμησης έχει ως αποτέλεσμα την αντικατάσταση του πεδίου τύπου (type) με ένα αντικείμενο κατάστασης/στρατηγικής και την απομάκρυνση της υποθετικής λογικής από τον κώδικα, η λειτουργικότητα της οποίας επιτυγχάνεται μέσω της κληρονομικότητας. Η αναδόμηση με χρήση του προτύπου «Μέθοδος Εργοστάσιο» αναφέρεται ως *«Αντικατάσταση Κατασκευαστή με τη Μέθοδο Εργοστάσιο»*(***Replace Constructor with Factory Method)*** και χρησιμοποιείται όταν απαιτείται επιπλέον λειτουργικότητα από την απλή δημιουργία ενός αντικειμένου, αντικαθιστώντας τον κατασκευαστή με μια μέθοδο εργοστάσιο. Η συγκεκριμένη αναδόμηση λόγω της συχνής χρήσης της έχει αυτοματοποιηθεί και παρέχεται από διάφορα εργαλεία αναδόμησης (Eclipse, IntelliJ, ReSharper).

Συνέχεια της δουλειάς του Fowler και κυριότερη συμβολή στη βιβλιογραφία αναφορικά με την αναδόμηση με πρότυπα σχεδίασης αποτελεί το βιβλίο του Kerievsky. Ο Kerievsky στο βιβλίο του *Refactoring to Patterns* ορίζει 27 διαφορετικές *pattern-directed* αναδομήσεις, οι οποίες αξιοποιούν 19 από τα 23 πρότυπα σχεδίασης. Καθεμία από τις αναδομήσεις αυτές κατατάσσεται **ανάλογα με τον βαθμό υλοποίησης** του προτύπου σχεδίασης στο οποίο αναφέρεται (Πίνακας 5).

Ορισμένες αναδομήσεις απαιτείται να υλοποιήσουν πλήρως ένα πρότυπο, ούτως ώστε να επιφέρουν αποτέλεσμα, παραδείγματος χάριν η αναδόμηση *«Διαμόρφωση Μεθόδου Υποδείγματος» (Form Template Method)*) (**κατηγορία to**). Άλλες αναδομήσεις παρέχουν αποδεκτές βελτιώσεις ακόμα και αν η διαδικασία υιοθέτησης ενός προτύπου υλοποιηθεί μερικώς, παραδείγματος χάριν η αναδόμηση *«Ενοποιήση Διεπαφών με Προσαρμογέα» (Unify Interfaces with Adapter)* (**κατηγορία towards**). Από την άλλη, ορισμένες φορές κρίνεται αναγκαίο να απομακρύνουμε ένα πρότυπο από τον κώδικα (περίπτωση αναδόμησης *Inline Singleton*) ή να το αντικαταστήσουμε με κάποιο άλλο πρότυπο (περίπτωση αναδόμησης *Ενθυλάκωση Σύνθετου με Κτίστη - Encapsulate Composite with Builder*) (**κατηγορία away**).

Επιπλέον ο Kerievsky κατατάσσει τις αναδομήσεις με πρότυπα σχεδίασης σε 6 κατηγορίες **ανάλογα με το σκοπό** που επιτελούν: δημιουργίας αντικειμένων *(creation)*, απλοποίησης δομής του κώδικα *(simplification),* γενίκευσης - μετατροπής του ειδικού σε γενικού σκοπού κώδικα *(generalization),* προστασίας *(protection)*, συσσώρευσης *(accumulation)* και λειτουργιών - αναδομήσεων χαμηλού επιπέδου *(utilites).* Ωστόσο, για την επιλογή της κατάλληλης αναδόμησης και τον προσδιορισμό του κατάλληλου προτύπου, σημαντικό ρόλο διαδραματίζει η εξέταση των συγκεκριμένων σχεδιαστικών προβλημάτων (bad smells) τα οποία επιλύει το πρότυπο και όχι του σκοπού στον οποίο αναφέρεται. Συνηθέστερα σχεδιαστικά προβλήματα που καλούνται να αντιμετωπίσουν οι αναδομήσεις με πρότυπα σχεδίασης αποτελούν η επανάληψη κώδικα και η αυξημένη πολυπλοκότητα του κώδικα. Στον Πίνακας 6 απεικονίζονται διάφορα σχεδιαστικά προβλήματα, μια σύντομη περιγραφή τους, καθώς και οι αντίστοιχες αναδομήσεις με πρότυπα σχεδίασης που τα επιλύουν.

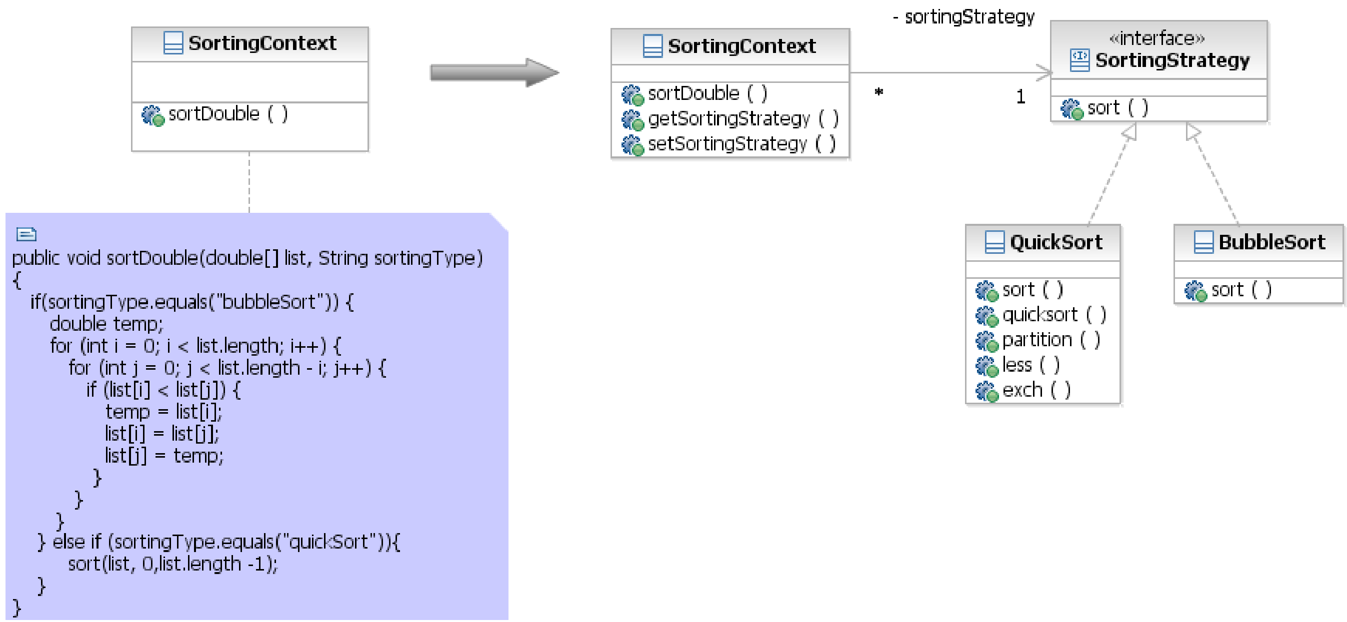
Pattern	To	Towards	Away
Adapter	Extract Adapter, Unify Interfaces with Adapter	Unify Interfaces with Adapter	
Builder	Encapsulate Composite with Builder		
Collecting Parameter	Move Accumulation to Collecting Parameter		
Command	Replace Conditional Dispatcher with Command	Replace Conditional Dispatcher with Command	
Composed Method	Compose Method		
Composite	Replace One/Many Distinctions with Composite, Extract Composite, Replace Implicit Tree with Composite		Encapsulate Composite with Builder
Creation Method	Replace Constructors with Creation Method		
Decorator	Move Embellishment to Decorator	Move Embellishment to Decorator	
Factory	Move Creation Knowledge to Factory, Encapsulate Classes with Factory		
Factory Method	Introduce polymorphic creation with factory method		
Interpreter	Replace Implicit Language with Interpreter		
Iterator			Move Accumulation To visitor
Null Object	Introduce Null Object		
Observer	Replace hard-coded notifications with observer	Replace hard-coded notifications with observer	
Singleton	Limit Instantiation with singleton		Inline singleton
State	Replace State-altering conditionals with state	Replace State-altering conditionals with state	
Strategy	Replace conditional logic with strategy	Replace conditional logic with strategy	
Template Method	Form template method		
Visitor	Move accumulation to visitor	Move accumulation to visitor	

Πίνακας 5 : Κατάλογος Αναδομήσεων με Πρότυπα Σχεδίασης

Για καθεμία από τις αναδομήσεις με πρότυπα σχεδίασης, ακολουθώντας το παράδειγμα του Fowler, ο Kerievsky προσδιορίζει

- ένα μοναδικό ***όνομα***,
- μια σύντομη ***περιγραφή*** τόσο λεκτική όσο και διαγραμματική, για την απεικόνιση του σχεδιαστικού μετασχηματισμού της αναδόμησης
- τους ***λόγους - κίνητρα*** *(motivation)* για την εφαρμογή της αναδόμησης καθώς και τα πλεονεκτήματα και τα μειονεκτήματα της
- τον ***αλγόριθμο*** *(mechanics)* που περιλαμβάνει βήμα προς βήμα οδηγίες για την εφαρμογή της αναδόμησης και τέλος,
- απλά ***παραδείγματα*** εφαρμογής της αναδόμησης, που επιδεικνύουν τον τρόπο με τον οποίο λειτουργεί.

Ακολούθως στην Εικόνα 9 παρουσιάζεται η διαγραμματική περιγραφή της αναδόμησης προς τη στρατηγική σύμφωνα με τον Kerievsky. Πρόκειται για το ίδιο παράδειγμα της κλάσης SortingContext που χρησιμοποιήθηκε και κατά την παρουσίαση του Πρότυπο Στρατηγική (36), ωστόσο στην κάτωθι απεικόνιση συμπληρωματικά παρουσιάζεται και ο εναλλακτικός τρόπος δόμησης της κλάσης context πριν από την αξιοποίηση του προτύπου, που αντενδείκνυται για πολύπλοκες δηλώσεις if και switch. Εν συνεχεία, στο επόμενο κεφάλαιο μελετώνται διεξοδικά ορισμένες από τις προσπάθειες που αναφέρονται στη βιβλιογραφία και αποσκοπούν στην αυτοματοποίηση ορισμένων από τις αναδομήσεις με πρότυπα σχεδίασης, οι οποίες περιγράφονται στο βιβλίο του Kerievsky.



Εικόνα 9 : Παράδειγμα πρίν και μετά την αναδόμηση προς τη στρατηγική

Σχεδιαστικό Πρόβλημα	Περιγραφή	Αναδόμηση Με Πρότυπα Σχεδίασης
Διπλότυπος Κώδικας (Duplicated Code)	Επανάληψη του ίδιου τμήματος κώδικα σε διαφορετικά σημεία του συστήματος, που συνήθως συμβαίνει λόγω της χρήσης “copy - paste”.	Form Template Method Introduce Polymorphic Creation with Factory Method Chain Constructors Replace One/Many Distinctions with Composite Extract Composite Unify Interfaces with Adapter Introduce Null Object
Εκτενής Μέθοδος (Long Method)	Εκτενής μέθοδος που προκύπτει λόγω της επανάληψης κώδικα, της ύπαρξής εκτενών υποθετικών δηλώσεων για το χειρισμό διαφορετικών αιτήσεων, της ύπαρξης διαφορετικών εκδόσεων ενός αλγορίθμου, κ.ο.κ. Ανάλογα με την αιτία του προβλήματος ενδείκνυται διαφορετικός τύπος αναδόμησης.	Compose Method Move Accumulation to Collecting Parameter Replace Conditional Dispatcher with Command Move Accumulation to Visitor Replace Conditional Logic with Strategy
Εμμονή με πρωταρχικούς τύπους (Primitive Obsession)	Υπερβολική χρήση δεδομένων πρωτογενών τύπων (συμβολοσειρών, ακεραίων, δεκαδικών, πινάκων, κλπ.) αντί για τη γενίκευση και απλοποίηση του κώδικα μέσω της ομαδοποίησης εννοιολογικά συσχετισμένων δεδομένων σε κλάσεις.	Replace Type Code with Class Replace State-Altering Conditionals with State Replace Conditional Logic with Strategy Replace Implicit Tree with Composite Replace Implicit Language with Interpreter Move Embellishment to Decorator Encapsulate Composite with Builder
Υποθετική Πολυπλοκότητα (Conditional Complexity)	Υπερβολική χρήση πολύπλοκης υποθετικής λογικής με πολλές και εκτενείς διακλαδώσεις αντί για την αξιοποίηση χρήσιμων αρχών της αντικειμενοστρέφειας όπως η κληρονομικότητα και ο πολυμορφισμός.	Replace Conditional Logic with Strategy Move Embellishment to Decorator Replace State-Altering Conditionals with State Introduce Null Object

Εκτενής Κλάση (Large/God Class)	Κατ' αντιστοιχία με το large method, πρόκειται για κλάση που αποτελείται από υπερβολικά πολλές γραμμές κώδικα λόγω του ότι αναλαμβάνει την υλοποίηση πολλών διαφορετικών αρμοδιοτήτων.	Replace Conditional Dispatcher with Command Replace State-Altering Conditionals with State Replace Implicit Language with Interpreter
Δηλώσεις Switch (Switch Statements)	Χρήση πολλών υποθετικών δηλώσεων αντί του πολυμορφισμού, που επαναλαμβάνονται σε διαφορετικά σημεία του κώδικα και περιπλέκουν τη σχεδίαση.	Replace Conditional Dispatcher with Command Move Accumulation to Visitor

Πίνακας 6 : Σχεδιαστικά προβλήματα και κατάλληλες αναδομήσεις με πρότυπα σχεδίασης

Κεφάλαιο 3

Αυτοματοποίηση Αναδόμησης σε Πρότυπα Σχεδίασης

Η αυτοματοποίηση της αναδόμησης απασχολεί την επιστημονική κοινότητα ήδη από τις αρχές της δεκαετίας του ’90, όταν ο W. Opdyke στη διπλωματική του αναφέρθηκε για πρώτη φορά στην ιδέα αξιοποίησης αυτόματων εργαλείων για την επιτυχή εκτέλεση της αναδόμησης των ΟΟ συστημάτων (7). Έκτοτε έχουν δημοσιευθεί πολλά ερευνητικά έργα, τα οποία πραγματεύονται το πρόβλημα της αυτοματοποίησης της αναδόμησης, αλλά και ορισμένα εργαλεία που επιτελούν το σκοπό αυτό (ενότητα 2.1.3).

Η σημασία της αυτοματοποίησης της αναδόμησης γίνεται αντιληπτή αναλογικά με την αύξηση του μεγέθους των σύγχρονων συστημάτων λογισμικού και την διαρκώς αυξανόμενη πολυπλοκότητα τους. Η ραγδαία αύξηση του μεγέθους των συστημάτων λογισμικού, προκειμένου να ανταποκριθούν στις διαρκώς αυξανόμενες απαιτήσεις, καθιστά την χειροκίνητη εξέταση τους απαγορευτική, τόσο από πλευράς χρόνου, όσο και κόστους και απαιτούμενης προσπάθειας που πρέπει να δαπανηθεί. Επιπροσθέτως, η εκτέλεση μιας τόσο απαιτητικής διαδικασίας αποκλειστικά από τον ανθρώπινο παράγοντα, χωρίς την υποστήριξη από κάποιο εργαλείο, κρίνεται ευπαθής σε σφάλματα, η ανίχνευση των οποίων, απαιτεί την διεξαγωγή ελέγχων έπειτα από κάθε τροποποίηση. Ωστόσο, η εκτέλεση ελέγχων για τη διασφάλιση της διατήρησης της συμπεριφοράς του λογισμικού, αυξάνει ακόμα περισσότερο τις απαιτήσεις της διαδικασίας αναδόμησης σε χρόνο και προσπάθεια.

Ένα ακόμα μειονέκτημα της μη-αυτοματοποιημένης αναδόμησης έγκειται στο γεγονός ότι ο χαρακτηρισμός ενός σημείου ως προβληματικό, αποκλειστικά και μόνο βάσει της ανθρώπινης διαίσθησης, χαρακτηρίζεται από έντονη υποκειμενικότητα, καθώς εξαρτάται από την προσωπική κρίση του μηχανικού λογισμικού αλλά και από την γνώση και κατανόηση των εννοιών της τεχνολογίας λογισμικού, θέτοντας υπό αμφισβήτηση την αξιοπιστία της διενεργούμενης διαδικασίας, παρακωλύοντας παράλληλα την γενίκευση του συμπεράσματος, σε ενδεχόμενες προσιδιάζουσες περιπτώσεις

Η αυτοματοποίηση της αναδόμησης καθίσταται αναγκαία για ένα ακόμα λόγο. Συχνά διατηρούνται πολλές διαφορετικές εκδόσεις του ίδιου έργου, για την εκπλήρωση διαφορετικών σκοπών: το σύστημα παραγωγής (production version), το σύστημα ανάπτυξης (development version), το σύστημα ελέγχου (testing version), κ.ο.κ. Προκειμένου η εφαρμογή της αναδόμησης να μπορεί να επαναληφθεί εύκολα και με ασφάλεια σε όλες τις διαφορετικές εκδόσεις του συστήματος, κρίνεται απαραίτητο η ανίχνευση και η απάλειψη των προβληματικών σημείων να γίνεται με αυτοματοποιημένο τρόπο.

Η αυτοματοποίηση της αναδόμησης, από το σύνολο των διακριτών ενεργειών που αποτελούν την αναδόμηση ως διαδικασία, όπως παρουσιάστηκαν στο προηγούμενο κεφάλαιο (σελ.33), αναφέρεται στις ενέργειες της ανίχνευσης των προβληματικών σημείων στον κώδικα, του προσδιορισμού της κατάλληλης αναδόμησης προς εφαρμογή, καθώς και της εκτέλεσης της. Ωστόσο, η επίτευξη πλήρους αυτοματοποίησης των τριών αυτών ενεργειών αποτελεί πρόκληση. Ιδιαίτερα κρίσιμη θεωρείται η αυτοματοποίηση της ανίχνευσης των υποψήφιων τμημάτων προς αναδόμηση, καθώς παρουσιάζει και το μεγαλύτερο βαθμό δυσκολίας.

Η δυσκολία αυτόματης ανίχνευσης σημείων αναδόμησης με πρότυπα σχεδίασης με ένα αξιόπιστο τρόπο, έγκειται στην εννοιολογική φύση των προβλημάτων στα οποία προσπαθούν να δώσουν λύση τα πρότυπα σχεδίασης. Η εννοιολογική φύση των προβλημάτων αυτών είναι δύσκολο να αποδοθεί με κάποιο τρόπο στον πηγαίο κώδικα. Για το λόγο αυτό κρίνεται απαραίτητη η ανθρώπινη συμμετοχή και η ανθρώπινη νοημοσύνη, προκειμένου να ανιχνευθούν σημεία, στα οποία θα μπορούσε να γίνει χρήση των προτύπων σχεδίασης. Επιπροσθέτως, τα πρότυπα σχεδίασης συνίστανται από αρκετά πολύπλοκές δομές και παράλληλα το πρόβλημα προσδιορισμού μιας αυτόματης μεθοδολογίας για την ανίχνευση της απουσίας τους σε ένα συγκεκριμένο τμήμα του κώδικα, δεν ορίζεται μονοσήμαντα, εφόσον υπάρχουν διάφορες εναλλακτικές στρατηγικές αναδόμησης που θα μπορούσαν να χρησιμοποιηθούν.

Στη συνέχεια, δυσκολία συναντάται και κατά τον προσδιορισμού του κατάλληλου προτύπου προς εφαρμογή, που εν μέρει σχετίζεται με την ανίχνευση των προβληματικών σημείων. Ο προσδιορισμός του κατάλληλου προτύπου σε ένα μεγάλο βαθμό ανάγεται στο πρόβλημα της *σημασιολογικής ανάλυσης (semantic analysis)* του πηγαίου κώδικα, ένα πρόβλημα ιδιαίτερα δυσεπίλυτο με χρήση αυτόματων τεχνικών.

Αναφορικά με την αυτοματοποίηση της εκτέλεσης της αναδόμησης με πρότυπα σχεδίασης, η δυσκολία που συναντάται σε σύγκριση με την αυτοματοποίηση των απλών αναδομήσεων που περιγράφει ο Fowler στο βιβλίο του (8), οφείλεται στην ίδια τη σύνθετη φύση των αναδομήσεων αυτών. Συγκεκριμένα, η αναδόμηση με πρότυπα σχεδίασης συνίσταται από μετασχηματισμούς του *εκτελέσιμου τμήματος (executable part)* ενός συστήματος, δηλαδή των εντολών οι οποίες εκτελούνται και περιλαμβάνονται στο σώμα των μεθόδων, σε αντίθεση με τις απλές αναδομήσεις που αφορούν μετασχηματισμούς στο *δηλωτικό τμήμα (declarative part)* μιας εφαρμογής (στη δήλωση μιας μεθόδου δηλαδή ουσιαστικά στην υπογραφή της, στη δήλωση πεδίων και κλάσεων καθώς και των συσχετίσεων τους). Οι τεχνικές που χρησιμοποιούνται για την επίτευξη των μετασχηματισμών αυτών σχετίζονται με το πρόβλημα της ανάλυσης του *ελέγχου ροής (flow control analysis),* της *ροής δεδομένων (data flow analysis)* και της *συμβολικής εκτέλεσης (symbolic execution /analysis),* προβλήματα τα οποία, ανάλογα με το υπό εξέταση σύστημα, ενδέχεται να εμφανίζουν αυξημένο βαθμό πολυπλοκότητας.

Εξαιτίας όλων των ανωτέρω λόγων, η έρευνα στην περιοχή της αναδόμησης με πρότυπα σχεδίασης, βρίσκεται σε ένα σχετικά πρώιμο στάδιο. Η σχετική βιβλιογραφία χαρακτηρίζεται από έλλειψη ώριμων τεχνικών αυτοματοποίησης της αναδόμησης με πρότυπα σχεδίασης, στις οποίες δεν απαιτείται η ανθρώπινη παρέμβαση. Ο κυριότερος από τους ανωτέρω λόγους είναι η αυξημένη δυσκολία στην αυτόματη ανίχνευση σημείων αναδόμησης. Ακολούθως εξετάζονται οι εναλλακτικές προσεγγίσεις σχετικά με την αυτοματοποίηση της ανίχνευσης των προβληματικών σημείων στον κώδικα που χρήζουν αναδόμησης γενικά, και εν συνεχεία παρουσιάζονται αναλυτικότερα οι προσεγγίσεις αυτόματης ανίχνευσης ευκαιριών αναδόμησης με πρότυπα σχεδίασης ειδικά.

3.1 Προσεγγίσεις Αυτόματης Ανίχνευσης

Ο Kerievksy στο βιβλίο του περί της αναδόμησης με πρότυπα σχεδίασης (16) συσχετίζει την ανάγκη χρήσης κάθε προτύπου σχεδίασης με τα αντίστοιχα προβλήματα στον κώδικα που επιλύει (*bad smells)* (Πίνακας 6). Ωστόσο, οι προσεγγίσεις αυτοματοποιημένης ανίχνευσης προτάσεων αναδόμησης με πρότυπα σχεδίασης που συναντώνται στη βιβλιογραφία ακολουθούν διαφορετική λογική, καθώς η λογική των *bad smells* δεν μπορεί να αναπαρασταθεί στον πηγαίο κώδικα, και αξιοποιείται μόνο από την ανθρώπινη λογική και τη διαίσθηση. Κατά συνέπεια οι προσεγγίσεις αυτοματοποιημένης αναδόμησης στηρίζονται σε τεχνικές ανίχνευσης, οι οποίες βάσει της ταξινόμησης του Τσάνταλη (40) διακρίνονται σε τεχνικές με χρήση

- **μετρικών** (metrics)
- **λογικού/δηλωτικού προγραμματισμού** (logic/declarative προγραμματισμού)
- **ταιριάσματος προτύπων** (pattern matching)
- **αναζήτησης** (search based)
- **πιθανοτικών μοντέλων** (probabilistic approaches) και
- **τεχνικών οπτικοποίησης** (visualization techniques).

Πολλές από τις προσεγγίσεις στη βιβλιογραφία βασίζονται σε **μετρικές** για την εξαγωγή συμπερασμάτων, καθώς οι μετρικές αποτελούν αξιόπιστες ποσοτικές μεθόδους που αντανακλούν άμεσα τη βελτίωση της ποιότητας του παραγόμενου σχεδίου *(metric-based approaches).*

Την επίδραση των μετασχηματισμών (μετακίνηση, ενθυλάκωση, γενίκευση, κλπ.) στις τιμές των μετρικών εκτιμά η προσέγγιση που προδιαγράφεται στο (41), η οποία προτείνει ένα framework στο οποίο η αυτόματη ανίχνευση των σημείων στα οποία μπορούν να εφαρμοστούν συγκεκριμένοι μετασχηματισμοί επιτυγχάνεται με τη χρήση ενός καταλόγου αποτελούμενου από επτά object-oriented μετρικές της σύζευξης, της συνεκτικότητας και της πολυπλοκότητας . Στα πλαίσια του αλγορίθμου ανίχνευσης που ορίζεται από τη μεθοδολογία αυτή κάθε επιθυμητός μετασχηματισμός συσχετίζεται με ένα διαφορετικό συνδυασμό μετρικών. Παρόμοια με την προσέγγιση των Tahvildari και Kontogiannis, της μετρικές της σύζευξης, της πολυπλοκότητας και της συνεκτικότητας χρησιμοποιεί και η προσέγγιση των Salehie et al. (42) για την ανίχνευση πέντε διαφορετικών σχεδιαστικών προβλημάτων (god class, shotgun surgery, refused bequest, god method).

Κατ’ αντιστοιχία, στο (43) ο Marinescu παρουσίασε μια προσέγγιση για την ανίχνευση 10 διαφορετικών σχεδιαστικών προβλημάτων βάσει στρατηγικών ανίχνευσης που εντοπίζουν τις αποκλίσεις από πρακτικές καλής σχεδίασης και ευρυστικές του συνδυασμό διαφορετικών μετρικών που προκύπτει από τη χρήση των λογικών τελεστών σύζευξης και διάζευξης (And/Or/ButNot). Οι τιμές που υπολογίζονται από τις μετρικές εν συνεχεία συγκρίνονται με τις τιμές απόλυτων και σχετικών ορίων (thresholds). Η προτεινόμενη προσέγγιση επεκτείνεται περαιτέρω στο (44) όπου το σύνολο των χρησιμοποιούμενων μετρικών εμπλουτίστηκε και προσδιορίστηκαν συγκεκριμένες τιμές για τα χρησιμοποιούμενα όρια βάσει στατιστικών δεδομένων που προέκυψαν από την ανάλυση ενός μεγάλου αριθμού συστημάτων λογισμικού σε java.

Μια ακόμα προσέγγιση που αξιοποιεί τις μετρικές στην ανίχνευση σημείων εφαρμογής των προτύπων σχεδίασης στον κώδικα, είναι η προσέγγιση που προτείνει η συγγραφική ομάδα των T. Muraki και M. Saeki (45). Σύμφωνα με την προσέγγιση αυτή, η οποία περιλαμβάνει αρκετά από τα 23 πρότυπα σχεδίασης και από τις 3 κατηγορίες (Abstract Factory, Factory Method, Prototype, Template Method, Strategy, State, Chain of Responsibility, Command, Visitor, Decorator), οι υπάρχουσες μετρικές ποιότητας του πηγαίου κώδικα αδυνατούν να προσεγγίσουν κατάλληλα το πρόβλημα. Για το σκοπό μελέτησαν διάφορα συστήματα λογισμικού χαμηλής ποιότητας και ανέπτυξαν ένα σύνολο 20 μετρικών ικανών να εκφράσουν την πολυπλοκότητα της υποθετικής λογικής και την ισχύ των εξαρτήσεων μεταξύ των υποκλάσεων στα δέντρα ιεραρχίας/κληρονομικότητας. Ο ορισμός των μετρικών αυτών βασίστηκε σε ένα μεταμοντέλο των αντικειμενοστραφών συστημάτων το οποίο ορίστηκε για τους σκοπούς της συγκεκριμένης προσέγγισης.

Επιπλέον, η συγγραφική ομάδα των A. Trifu και U. Reupke προτείνει μια μεθοδολογία για την αυτόματη ανίχνευση code smells στον πηγαίο κώδικα βάσει 3 διαφορετικών παραγόντων (indicators): αθροιστικούς παράγοντες (aggregating indicators) δηλαδή μετρικές ή τον συνδυασμό των μετρικών μέσω λογικών εκφράσεων, δομικούς (structural) δηλαδή πρότυπα στη δομή του κώδικα, και σημασιολογικούς παράγοντες (semantic) όπως για παράδειγμα το όνομα μιας κλάσης (46). Ορισμένες από τις μετρικές της μεθοδολογίας αυτής είναι η κυκλωματική πολυπλοκότητα (cyclomatic complexity), η σύζευξη κλάσεων (class cohesion), ο αριθμός μεθόδων μιας κλάσης (number of methos NOM), κ.ο.κ. Η προτεινόμενη μεθοδολογία αναπτύχθηκε σαν plugin του Eclipse, ονόματι CodeClinic και συνίσταται από τον εντοπισμό και υπολογισμό των ανωτέρω παραγόντων και την επαλήθευση του σημασιολογικού πλαισίου των προτεινόμενων προβληματικών σημείων βάσει προκαθορισμένων ερωτήσεων που τίθενται στο σχεδιαστή του συστήματος.

Μια πιο πρόσφατη προσέγγιση που κάνει χρήση μετρικών προτάθηκε από τους N. Moha et. Al. για την ανίχνευση 4 design smells: blob, functional decomposition, spaghetti code και swiss army knife (47). Το εργαλείο DÉCOR που αναπτύχθηκε βασίζεται στη χρήση μιας τεχνικής ανίχνευσης των προβληματικών σημείων, η οποία ονομάστηκε DETEX και υλοποιήθηκε στα πλαίσια της προσέγγισης. Η σύνθετη αυτή τεχνική, που αποτελείται από 4 βήματα (ανάλυση πεδίου, απαιτήσεων, παραγωγή αλγορίθμου, ανίχνευση) κάνει μεταξύ άλλων, και χρήση ενός συνόλου από μετρικές ανάμεσα στις οποίες είναι το βάθος της κληρονομικότητας (depth of inheritance DIT), ο αριθμός γραμμών σε μια κλάση (LOC Class), ο αριθμός γραμμών κώδικα σε μια μέθοδο (LOC Method), η έλλειψη σύζευξης στις μεθόδους (lack of cohesion in methods LCOM), κ.ο.κ. Κατ' αντιστοιχία με την προσέγγιση των Trifu και Reupke (46). για την ανίχνευση των smells, οι οποίες χωρίζονται σε smells εντός κλάσης (intraclass) και διαταξικές (interclass), πέρα από τα μετρήσιμα χαρακτηριστικά του κώδικα, που εξετάζονται με χρήση των μετρικών, συνδυάζεται επιπλέον η ανάλυση λεξικογραφικών (π.χ. ονομάτων) και δομικών πληροφοριών (π.χ. ύπαρξη συσχέτισης μεταξύ κλάσεων).

Τέλος, για τους σκοπούς της αναγνώρισης περιπτώσεων όπου μπορεί να εφαρμοστεί η αναδόμηση μετακίνηση μεθόδου (move method), η συγγραφική ομάδα του ερευνητικού έργου (48) υιοθετεί την αρχή της χαμηλής σύζευξης (coupling) και υψηλής συνεκτικότητας (cohesion) και ορίζει μια πρωτότυπη μετρική, ονόματι Entity Placement. Η μετρική αποδίδει με ένα ποσοτικό τρόπο πόσο καλά έχουν τοποθετηθεί οι οντότητες (μέθοδοι και ιδιότητες) στις κλάσεις του συστήματος. Για το σκοπό αυτό ο αλγόριθμος που έχει αναπτυχθεί στα πλαίσια της προσέγγισης αυτής αφενός εξετάζει την τιμή της μετρικής ως προς κάθε κλάση ξεχωριστά στοχεύοντας σε όσο το δυνατόν πλησιέστερες του μηδενός τιμές και αφετέρου τη σταθμισμένη τιμή της μετρικής ολόκληρου του συστήματος, στοχεύοντας στη χαμηλότερη δυνατή τιμή, ούτως ώστε να αποφανθεί ότι οι οντότητες έχουν κατανεμηθεί ορθά στις κλάσεις του συστήματος. Η μετρική Entity Placement μιας κλάσης υπολογίζεται από το λόγο της μέσης απόστασης Jaccard των οντοτήτων που είναι δηλωμένες στα πλαίσια της κλάσης ως προς τη μέση απόσταση των οντοτήτων που δεν ανήκουν στην κλάση. Αντίστοιχα, η σταθμισμένη μετρική συστήματος υπολογίζεται από το άθροισμα των γινομένων της τιμής της Entity Placement κάθε κλάσης επί του λόγου των οντοτήτων της δια το συνολικό πλήθος οντοτήτων του συστήματος.

Εξίσου δημοφιλής με τις μεθοδολογίες βάσει μετρικών είναι και οι προσεγγίσεις οι οποίες αξιοποιώντας τις δυνατότητες του **λογικού προγραμματισμού**, κάνουν χρήση γεγονότων *(facts)* και κανόνων *(rules)* προκειμένου να αποφανθούν για την ανάγκη χρήσης των προτύπων σχεδίασης στον πηγαίο κώδικα *(logic-based approaches)*. Στις προσεγγίσεις αυτές τόσο το υπό εξέταση σύστημα όσο και τα προς αναζήτηση *bad smells* εκφράζονται με τον ίδιο αφηρημένο συμβολισμό *(abstract notation)*, και στη συνέχεια για την εξαγωγή των υποψηφίων αναδομήσεων, εφαρμόζονται στην αφηρημένη αναπαράσταση του συστήματος οι *λογικοί κανόνες*.

Μια ημιαυτόματη προσέγγιση **λογικού μεταπρογραμματισμού** χρησιμοποιεί η συγγραφική ομάδα των Tourwe και Mens για την αναζήτηση ευκαιριών αναδόμησης (49). Η προσέγγιση περιλαμβάνει τον προσδιορισμό ενός framework που χρησιμοποιεί τη μεταπληροφορία των συστημάτων λογισμικού προκειμένου να προτείνει την εφαρμογή αναδόμησης σε σημεία που εντοπίζονται τα bad smells *obsolete parameter* και *inappropriate interface*, με χρήση δηλώσεων λογικού προγραμματισμού σε SOUL.

Τη χρήση δηλωτικού προγραμματισμού, μέσω της γλώσσας Prolog όμως, αξιοποιεί και η ομάδα των συγγραφέων J. Rajesh και D. Janakiram στο εργαλείο που ανέπτυξε, ονόματι JIAD προκειμένου να επιτύχει την αυτόματη ανίχνευση intent aspects, που βοηθούν με τη σειρά τους στην εφαρμογή του κατάλληλου πρότυπου σχεδίασης κατά την αναδόμηση (50). Τα πρότυπα στα οποία απευθύνονται τα intent aspects είναι το αφηρημένο εργοστάσιο, το μοναδιαίο, το σύνθετο, ο διακοσμητής και η μέθοδος υπόδειγμα. Για κάθε πρότυπο σχεδίασης παράγεται ένα σύνολο κανόνων prolog (prolog rule), οι οποίοι χρησιμοποιούνται για την ανίχνευση intent aspects που χρειάζεται να αναδομηθούν με το αντίστοιχο πρότυπο. Ο πηγαίος κώδικας του υπό εξέταση συστήματος λογισμικού αναλύεται με χρήση του εργαλείου AspectJ προκειμένου να κατασκευαστεί η AST αναπαράσταση του, η οποία λειτουργεί ως βάση για την παραγωγή των prolog γεγονότων (facts) του προγράμματος. Τέλος ελέγχεται η ικανοποιησιμότητα των κανόνων έναντι των γεγονότων του προγράμματος

Κατ’ αντιστοιχία με τη χρήση λογικού προγραμματισμού διάφορες προσεγγίσεις προτείνουν τεχνικές **ταιριάσματος προτύπων** *(pattern-matching)* προκειμένου να εντοπιστούν οι προτάσεις αναδόμησης. Στις τεχνικές αυτές προσδιορίζεται η πιθανή δομή των προβληματικών σημείων του κώδικα ως αναζητούμενο πρότυπο, και το σύνολο από τις προτάσεις αναδόμησης απαρτίζεται από τις δομές του κώδικα που ικανοποιούν το πρότυπο αυτό.

Για την ανίχνευση διπλότυπου κώδικα (duplicated code) σε ένα σύστημα λογισμικού ανεξάρτητα από τη γλώσσα προγραμματισμού που έχει χρησιμοποιηθεί, οι συγγραφείς Ducasse et al. ανέπτυξαν ένα αλγόριθμο που δεν στηρίζεται στη στατική ανάλυση του κώδικα, αλλά χρησιμοποιεί τη λεκτική του αναπαράσταση για να εκτελέσει απλό line-based

ταίριασμα συμβολοσειρών (string-matching) και να εντοπίσει πιθανή επανάληψη κώδικα (51). Η προσέγγιση αυτή περιλαμβάνει γραφική αναπαράσταση του διπλότυπου κώδικα καθώς και λεπτομερών συνοδευτικών αναφορών.

Την τεχνική ταιριάσματος προτύπων χρησιμοποιεί και το εργαλείο Daikon, το οποίο υλοποίησε η συγγραφική ομάδα των Kataoka et al. (52) και ανιχνεύει με χρήση scripts σε perl τα **πρότυπα** (patterns) των invariants[4] του προγράμματος, τα οποία συσχετίζει με την δυνατότητα εφαρμογής διαφόρων προτάσεων αναδόμησης (Remove Parameter, Eliminate Useless Return Value, Separate Query from Modifier, Encapsulate Downcast, Replace Temp with Query). Αξίζει να τονιστεί πώς η συγκεκριμένη προσέγγιση σε αντίθεση με την πλειοψηφία των προτεινόμενων μεθοδολογιών αυτόματης αναδόμησης, κάνει χρήση δυναμικής ανάλυσης και χρησιμοποιείται συμπληρωματικά σε πολλές προσεγγίσεις που στηρίζονται στη στατική ανάλυση.

Αντίθετα από την προηγούμενη προσέγγιση, στην προσέγγιση αναφορικά με τον εντοπισμό περιπτώσεων αναδόμησης εξαγωγής μεθόδου (extract method) που παρουσιάζεται στο (53) από τους συγγραφείς Τσάνταλη και Χατζηγεωργίου δεν αναζητούνται στον κώδικα συγκεκριμένα πρότυπα αλλά καθορισμένες δομές, και συγκεκριμένα τα τμήματα που περιλαμβάνονται σε πολύπλοκες και εκτενείς μεθόδους και αφορούν τον υπολογισμό της τιμής μιας μεταβλητής. Για την επίτευξη του σκοπού αυτού αξιοποιείται η τεχνική τεμαχισμού βάσει δομικών μπλοκ (block-based slicing) με χρήση της οποίας υπολογίζονται όλα τα δυνατά διαφορετικά τεμάχια για τα διάφορα κριτήρια τεμαχισμού και τα διάφορα boundary blocks. Η προτεινόμενη προσέγγιση επίσης εντοπίζει περιπτώσεις στις οποίες η εξαγωγή μεθόδου θα παραβίαζε την αρχή διατήρησης της συμπεριφοράς καθώς στο εσωτερικό των blocks περιλαμβάνονται εντολές διαφυγής (break, return, continue).

Επιπλέον, το πρόβλημα της βελτίωσης του OO σχεδίου μέσω της αναδόμησης, διατυπώνεται σε ορισμένες προσεγγίσεις ως ένα **πρόβλημα αναζήτησης στο χώρο των πιθανών εναλλακτικών σχεδίων** *(search-based approaches)*. Στις προσεγγίσεις αυτές το σύνολο των μετασχηματισμών των αναδομήσεων μοντελοποιείται από τις μεταβάσεις στο χώρο των εναλλακτικών σχεδίων, τα οποία ταξινομούνται βάσει της ποιότητας τους, μέσω μιας *συνάρτησης καταλληλότητας (fitness function) (quality evaluation function).* Προκειμένου να επιλεγεί το βέλτιστο ή το σχεδόν βέλτιστο σχέδιο *(optimal/near-optimal)* του OO

[4] Invariant : Ιδιότητα του λογισμικού που δεν μεταβάλλεται, παραδείγματος χάριν όταν η τιμή της παραμέτρου μιας μεθόδου είναι πάντα σταθερή.

συστήματος που δόθηκε ως είσοδος, από το οποίο κάθε μετάβαση προς οποιοδήποτε εναλλακτικό σχέδιο χειροτερεύει την τιμή της συνάρτησης καταλληλότητας, επιλέγεται μια τεχνική αναζήτησης. Ως αποτέλεσμα προκύπτει ένα σύνολο από μετασχηματισμούς που πρέπει να εφαρμοστούν ακολουθιακά, προκειμένου να επιτευχθεί το βέλτιστο σχέδιο. Κυριότερες search-based προσεγγίσεις στη βιβλιογραφία είναι η προσέγγιση που παρουσιάζεται από τους O' Keeffe και Ó Cinnéide στα (54) (55) (56) που αφορούν ορισμένες από τις απλές αναδομήσεις του Fowler (pull up method/field, push down method/field, κλπ), η προσέγγιση των Seng et al. (57) η οποία βασίζεται στην χρήση ενός γενετικού αλγορίθμου και της οποίας η συνάρτηση καταλληλότητας είναι πρωτότυπη αλλά βασίζεται στο συνδυασμό γνωστών μετρικών και τέλος η προσέγγιση των Harman και Tratt (58) για την αναδόμηση μετακίνησης μεθόδου(extract method) που διαφοροποιείται από τις προηγούμενες καθώς χρησιμοποιεί την έννοια της βελτιστότητας Pareto για την εύρεση του βέλτιστου σχεδίου. Κυριότερο μειονέκτημα αυτών των προσεγγίσεων είναι ότι οι προτάσεις που παρέχουν είναι μη ντετερμινιστικές, πρόβλημα από το οποίο πάσχουν όλες οι αντίστοιχες ευρυστικές προσεγγίσεις, ενώ ταυτόχρονα χαρακτηρίζονται από τυχαιότητα και επηρεάζονται σε μεγάλο βαθμό από την αρχική παραμετροποίηση.

Υπό το πρίσμα άλλων δημοσιεύσεων λύση στο πρόβλημα της ανίχνευσης των υποψήφιων τμημάτων στον κώδικα αποπειρώνται να δώσουν τα **πιθανοτικά μοντέλα** ταξινομώντας τις κλάσεις ως αναπαραστάσεις σχεδιαστικών προβλημάτων ή μη *(probabilistic approaches).*

Μια από τις προσεγγίσεις που χρησιμοποιούν πιθανοτικά μοντέλα για την ανίχνευση περιπτώσεων αναδόμησης των bad smells shotgun surgery και divergent change είναι η προσέγγιση που προδιαγράφεται στο (59). Σύμφωνα με την προσέγγιση αυτή η ανίχνευση των bad smells στον κώδικα επιτυγχάνεται μέσω της μήτρας DCPP (design change propagation probability) η οποία κατασκευάζεται για τις κλάσεις του συστήματος και απεικονίζει την πιθανότητα μια κλάση να χρειάζεται τροποποίηση έπειτα από την μεταβολή μιας άλλης κλάσης. Οι κλάσεις του συστήματος αναπαρίστανται σε ένα γράφο URA βάσει του οποίου υπολογίζεται η ισχύς των εξαρτήσεων μεταξύ τους. Η προσέγγιση ορίζει ένα σύνολο υποθέσεων – κριτήριων βάσει των τιμών στις γραμμές και τις στήλες της μήτρας η ταυτόχρονη ικανοποίηση τριών εξ' αυτών αυξάνει την πιθανότητα μια κλάση να αποτελεί bad smell καθώς ενδέχεται να υφίσταται συχνές αλλαγές.

Στη χρήση πιθανοτικών μοντέλων βασίζεται και η προσέγγιση των Khomh et. al (60), η οποία υποστηρίζει πώς οι υπάρχουσες προσεγγίσεις για την ανίχνευση των bad smells δεν μπορούν να αντιμετωπίσουν την εγγενή αβεβαιότητα της διαδικασίας ανίχνευσης. Για το σκοπό αυτό προτείνουν μια προσέγγιση για την ανίχνευση του antipattern Blob βασισμένη στη χρήση δικτύων πεποίθησης Bayes (Bayesian velief networks, BBN), όπου η πιθανότητα μια κλάση να αποτελεί bad smell προκύπτει ως αποτέλεσμα του BBN. Για το σκοπό αυτό προσδιορίζεται για το συγκεκριμένο σχεδιαστικό πρόβλημα ένα rule-card, το οποίο, κατ' αντιστοιχία με την προσέγγιση της συγγραφικής ομάδας Moha et al, αποτελείται από τριών ειδών ιδιότητες: τιμές μετρικών, δομικές και λεκτικές ιδιότητες. Επιπλέον η συγκεκριμένη προσέγγιση απέδειξε ότι τα BBNs μπορούν να βαθμονομηθούν με τη χρήση ιστορικών δεδομένων του ίδιου ή διαφορετικού συστήματος.

Παρόμοια με την προηγούμενη προσέγγιση, την πιθανότητα μια κλάση να αποτελεί περίπτωση κλάσης Blob (ή αλλιώς God class) υπολογίζει και η προσέγγιση που αναλύεται στο (61). Στην προσέγγιση αυτή που συγκρίνεται τόσο με την προηγούμενη μεθοδολογία όσο και με την προσέγγιση της συγγραφικής ομάδας Moha et al,δημιουργούνται οι καμπύλες B-Splines που μοντελοποιούν την υπογραφή μιας κλάσης βάσει των τιμών ορισμένων μετρικών της π.χ. σύζευξη. Στη συνέχεια η πιθανότητα η κλάση να αποτελεί υποψήφια περίπτωση αναδόμησης εκτιμάται βάσει της απόστασης της καμπύλης της κλάσης είτε από την αντίστοιχη καμπύλη της ιδανικής μορφής περίπτωσης Blob (περίπτωση one-tailed) ή από τις αντίστοιχες καμπύλες ενός συνόλου δεδομένων εκπαίδευσης (περίπτωση two-tailed).

Τέλος, συμβολή στην ανίχνευση των προβλημάτων του σχεδίου παρέχουν οι **τεχνικές οπτικοποίησης** *(visualization techniques)* των προβληματικών σημείων *(design flaws)* στον κώδικα.

Μια μεθοδολογία που συνδυάζει τη χρήση μετρικών με τις τεχνικές οπτικοποίησης για την ανίχνευση ευκαιριών εφαρμογής απλών αναδομήσεων (Move Method, Move Attribute, Extract Class και Inline Class), είναι η προσέγγιση που προτείνεται από τη συγγραφική ομάδα F. Simon, F. Steinbrückner και C. Lewerentz στην ανάπτυξη του εργαλείου Crocodile (62). Οι συγγραφείς της προσέγγισης αυτής απεικονίζουν γραφικά το σύνολο των μεθόδων και των ιδιοτήτων των κλάσεων σε ένα μοντέλο VRML-world, και ο προγραμματιστής χρησιμοποιεί την αναπαράσταση αυτή για να αποφασίσει ποιες μεθόδους και ιδιότητες θα μετακινηθούν, βάσει του υπολογισμού της σύνθεσης (cohesion) των ιδιοτήτων και των μεθόδων. Για την εύρεση της σύνθεσης επιλέγεται μια συνάρτηση της απόστασης Jaccard των μεθόδων και των ιδιοτήτων (distance-based cohesion) και τα αποτελέσματα του υπολογισμού της μετατρέπονται σε ένα τρισδιάστατο καρτεσιανό σύστημα συντεταγμένων.

Παρόμοια είναι και η προσέγγιση που προτάθηκε από τους Van Emden και Moonen (63) για την ανίχνευση code smells που σχετίζονται με τον έλεγχο ή την εκχώρηση τύπου κατά το χρόνο εκτέλεσης (instanceof και typecast), σε συστήματα λογισμικού σε γλώσσα java με ένα μηχανισμό οπτικοποίησης με χρήση γράφων. Ο μηχανισμός αυτός υλοποιήθηκε στον browser jCosmo και απεικονίζει τα τμήματα του συστήματος που επηρεάζονται περισσότερο καθώς και τα σημεία στα οποία συναντάται μεγαλύτερη συγκέντρωση από bad smells .

Οι συγγραφείς Parnin et al. (64) ανέπτυξαν ως plugin του Visual Studio το εργαλείο Noseprints, το οποίο απεικονίζει πιθανά προβλήματα που ανιχνεύονται από αυτόματα εργαλεία επιθεώρησης κώδικα (automated inspection tools) και τα οποία ταξινομήθηκαν σε 4 διαφορετικές κατηγορίες : smells δήλωσης (statement), κλάσης, μεθόδου και collaboration code. Οι αναπαραστάσεις (visualizations) σχετικά με τα smells, δεν απεικονίζονται απευθείας στον πηγαίο κώδικα, αλλά σε ξεχωριστή πλήρη οθόνη με χρήση μιας δισδιάστατης γραφικής απεικόνισης.

Αντιθέτως, το εργαλείο Stench Blossom που ανέπτυξαν οι συγγραφείς E. Murphy-Hill et. al. (65) και το οποίο παρέχεται ως plug-in του Eclipse, προσφέρει τη δυνατότητα οπτικοποίησης των design smells του υπό εξέταση συστήματος λογισμικού, όχι σαν ξεχωριστές αναπαραστάσεις, αλλά απευθείας στην προβληματική δομή του κώδικα (direct visualization of code structure) στον επεξεργαστή του Eclipse (workbench editor). Το εργαλείο παρέχει τρείς διαφορετικές όψεις (ambient view, active view και explanation view) που παρέχουν διαφορετικού είδους πληροφορίες αναφορικά με τα ανιχνευμένα smells.

Επιλογικά, όλες οι ανωτέρω προσεγγίσεις, καθεμία με τα δικά της πλεονεκτήματα και μειονεκτήματα, θεωρούνται ιδιαίτερα αξιόλογες αναλογικά και με τη δυσκολία του προβλήματος στο οποίο καλούνται να δώσουν λύση. Ακολούθως, παρουσιάζονται επιγραμματικά οι κυριότερες δημοσιεύσεις στη βιβλιογραφία αναφορικά με την αυτοματοποιημένη αναδόμηση με πρότυπα σχεδίασης, ορισμένες από τις οποίες υιοθετούν τις τεχνικές ανίχνευσης που αναφέρθηκαν στην ενότητα αυτή.

3.1.1. Επισκόπηση Βιβλιογραφίας

Παρόλο που το βιβλίο του Kerievsky αναφορικά με την αναδόμηση με πρότυπα σχεδίασης δημοσιεύθηκε το 2004, η ιδέα της αυτοματοποίησης της χρήσης προτύπων σχεδίασης για τη βελτίωση της ποιότητας του υπάρχοντα κώδικα είχε ήδη προταθεί από τους Tokuda και Batory το ’95 (Batory 1995). Ωστόσο χρειάστηκε να περάσουν ορισμένα χρόνια ακόμη μέχρι την εμφάνιση της πρώτης μεθοδολογίας για την αυτόματη υποστήριξη της εισαγωγής προτύπων σχεδίασης σε ένα υπάρχον ΟΟ σύστημα με χρήση μετασχηματισμών από τον Ó Cinneide (66) (Cinnéide 2000) (67).

Για το σκοπό αυτό η μεθοδολογία του Ó Cinneide ορίζει το *σημείο εκκίνησης* του μετασχηματισμού *(precursor)*, τα πρότυπα σχεδίασης ως σύνθεση *μίνι-προτύπων (minipatterns)* και την ανάπτυξη των κατάλληλων *μίνι-μετασχηματισμών (minitransformations)* για την εισαγωγή των *μίνι-προτύπων* στον πηγαίο κώδικα. Επιπλέον τονίζει τη σημασία της διατήρησης της συμπεριφοράς *(behaviour preservation)* του συστήματος και συσχετίζει κάθε *μίνι-μετασχηματισμό* με διάφορες *προϋποθέσεις (preconditions)* και *μετασυνθήκες (postconditions)*, ο έλεγχος της ικανοποιησιμότητας των οποίων, μέσω της εκτέλεσης ενός αλγορίθμου, διασφαλίζει τη διατήρηση της συμπεριφοράς. Ο μετασχηματισμός ενός προτύπου σχεδίασης ορίζεται ως η σύνθεση των διάφορων *μίνι-μετασχηματισμών*. Για την υλοποίηση της προτεινόμενης μεθοδολογίας και την εφαρμογή σε προγράμματα Java κατασκευάστηκε ένα εργαλείο, το DPT, το οποίο διαθέτει μια *στρωματοποιημένη αρχιτεκτονική (layered architecture).* Η χρήση της αρχιτεκτονικής αυτής παρέχει ανεξαρτησία από τη γλώσσα προγραμματισμού *(language independency)* καθώς οι συγκεκριμένες λεπτομέρειες της γλώσσας προγραμματισμού μπορούν να απομονωθούν στα χαμηλότερα επίπεδα. Η προτεινόμενη μεθοδολογία εφαρμόστηκε σε αρκετά πρότυπα σχεδίασης όπως το *αφηρημένο εργοστάσιο (abstract factory)*, η *μέθοδος εργοστάσιο (factory method),* ο *κτίστης (builder),* το *πρωτότυπο (prototype),* το *μοναδιαίο (singleton),* κ.α.

Ακολουθώντας το παράδειγμα του Ó Cinneide η συγγραφική ομάδα των Jeon, Lee και Bae παρουσίασε μια προσέγγιση για την ανίχνευση τμημάτων κώδικα, όπου θα μπορούσε να είχε χρησιμοποιηθεί το πρότυπο σχεδίασης αφηρημένο εργοστάσιο *(abstract factory)* σε συστήματα λογισμικού Java (68). Η προσέγγιση αυτή κάνει χρήση του λογικού προγραμματισμού, δημιουργώντας ένα *μεταμοντέλο (metamodel)* σε Prolog, ενώ βασίζει τις υποθέσεις της για την πιθανότητα χρήσης του αφηρημένου εργοστασίου στην ανάλυση δύο ή περισσότερων εκδόσεων του ίδιου συστήματος. Η προσέγγιση βασίζεται στη μοντελοποίηση ενός object-oriented προγράμματος ως ένα σύνολο από *λογικά κατηγορήματα (logical predicates)* που αναπαριστούν τις δομικές και συμπεριφορικές ιδιότητες του, τα οποία στη συνέχεια μετατρέπονται σε ένα σύνολο από *γεγονότα (facts) Prolog,* ενώ αντίστοιχα οι κανόνες συμπερασμού μετατρέπονται σε Prolog rules. Τέλος, για την ανίχνευση των υποψήφιων σημείων εφαρμόζεται στον υπό εξέταση κώδικα μια Prolog επερώτηση *(query)*. Ακολούθως, η στρατηγική αναδόμησης που ορίζεται από τη μεθοδολογία εφαρμόζεται στα αποτελέσματα που προκύπτουν από την επερώτηση για να δημιουργηθεί το ζητούμενο πρότυπο σχεδίασης. Η ομάδα σκόπευε να υλοποιήσει την προτεινόμενη μεθοδολογία με χρήση της java και συγκεκριμένα του εργαλείου JavaCC για την συντακτική ανάλυση του πηγαίου κώδικα και τη χρήση της Prolog. Ωστόσο δεν ακολούθησε κάποια νεώτερη δημοσίευση.

Με την ανίχνευση περιπτώσεων απουσίας του πρότυπου σχεδίασης *αφηρημένο εργοστάσιο (abstract factory)* σε αντικειμενοστραφή συστήματα λογισμικού σε Java ασχολήθηκε και ο C. Jebelean (69) (70). Και αυτή η μεθοδολογία του αξιοποιεί το λογικό προγραμματισμό για τον εντοπισμό στον κώδικα υποψήφιων περιπτώσεων χρήσης του αφηρημένου εργοστασίου. Ωστόσο, η προσέγγιση αυτή οριοθέτησε πιο στενά το πρόβλημα προσδιορίζοντας υπό μορφή antipattern, τη δομή της πιο συχνής περίπτωσης στην οποία το πρότυπο εσφαλμένα δεν χρησιμοποιήθηκε. Η αναζήτηση που προτείνεται πραγματοποιείται σε τρία διαφορετικά επίπεδα: σε επίπεδο μεθόδου, ανεξάρτητα από τη θέση των δηλώσεων δημιουργίας αντικειμένων, σε επίπεδο μεθόδου στο ίδιο μονοπάτι ελέγχου *(control path)* και τέλος στο ίδιο μονοπάτι ελέγχου σε επίπεδο κλάσης. Συγκεκριμένα, για την ανίχνευση των εντολών δημιουργίας αντικειμένων σε ΟΟ κώδικα παράγεται στην Prolog η *βάση γνώσης (knowledge base)*, η οποία περιλαμβάνει τους κανόνες που πρέπει να ισχύουν για να θεωρηθεί ένα τμήμα κώδικα υποψήφιο για την εφαρμογή του *αφηρημένου εργοστασίου*. Η στρατηγική ανίχνευσης υλοποιείται μέσω της διατύπωσης Prolog *επερωτήσεων (queries)* που ελέγχουν την ικανοποιησιμότητα των κανόνων. Η συγκεκριμένη προσέγγιση υπερτερεί της προσέγγισης του Ó Cinneide (66) (71) (67) ,καθώς παρέχεται και υλοποίηση, βασισμένη σε ένα framework της Java, το RECORDER, για την ανάλυση *(parsing)* του κώδικα του συστήματος – στόχου, και στη χρήση της διεπαφής JPL, για την διασύνδεση της Java με την Prolog και την εκτέλεση των επερωτήσεων ανίχνευσης.

Αναφορικά με τα συμπεριφορικά πρότυπα, στη διδακτορική του διατριβή ο N. Juillerat περιγράφει έναν αλγόριθμο σχετικά με την εφαρμογή της αναδόμησης με χρήση του προτύπου *μέθοδος υπόδειγμα (form template method),* σε προγράμματα γραμμένα στην OO γλώσσα προγραμματισμού java (72) (73). Για το σκοπό αυτό συσχετίζει τη διαδικασία σχηματισμού του προτύπου *μέθοδος υπόδειγμα (template method)*, με τη διαδικασία ανίχνευσης κλώνων *(clone detection)* (74) και απομάκρυνσης τους από τον κώδικα, με αξιοποίηση της αναδόμησης *εξαγωγής μεθόδου (extract method)*. Ο αλγόριθμος εκτελεί ένα μετασχηματισμό λαμβάνοντας ως είσοδο δύο μεθόδους που προσομοιάζουν, αλλά δεν είναι πανομοιότυπες, και δημιουργεί τη μέθοδο υπόδειγμα, εξάγοντας τις όμοιες δηλώσεις των δύο μεθόδων. Οι διαφορές εξάγονται σε νέες μεθόδους που αρχικά τοποθετούνται στις δύο κλάσεις, ενώ αργότερα μεταφέρονται στην αμέσως επόμενη υπερκλάση στην ιεραρχία. Απαραίτητη προϋπόθεση για την εφαρμογή της αναδόμησης αποτελεί η εκτέλεση ενός ακόμα πρόσθετου βήματος, πέραν την ανίχνευσης. Κατά την εκτέλεση του βήματος αυτού ουσιαστικά εξετάζεται ο *έλεγχος ροής* και η ανάλυση της *ροής δεδομένων* των μεθόδων *(control flow and data flow analysis)*, και ελέγχεται ότι ικανοποιούνται όλες οι απαραίτητες προϋποθέσεις, ούτως ώστε η εξαγωγή της μεθόδου να μπορεί να πραγματοποιηθεί με ασφάλεια. Η συγκεκριμένη προσέγγιση βασίζεται στη χρήση, τόσο τη δημιουργία όσο και τη διάσχιση, αφηρημένων συντακτικών δέντρων *(Abstract Syntax Trees AST)*. Για τη σύγκριση και την ανίχνευση των όμοιων και των διαφορετικών δηλώσεων χρησιμοποιείται ενός αλγόριθμος σύγκρισης *(differentiation algorithm),* παρόμοιος με τον αλγόριθμο που χρησιμοποιεί η εντολή diff στο λειτουργικό σύστημα Linux, ο οποίες εφαρμόζεται σε λίστες από *tokens* που δημιουργούνται με χρήση *μεταδιατεταγμένης διάσχισης (post-order traversal)* των AST δέντρων, ενώ παράλληλα παρέχεται και υλοποίηση βάσει της βιβλιοθήκης JDT *(Java Development Tools)* του Eclipse, υπό τη μορφή plug-in.

Τη δημιουργία και τη διάσχιση αφηρημένων συντακτικών δέντρων αξιοποιεί και η προσέγγιση της συγγραφικής ομάδας των Τσάνταλη και Χατζηγεωργίου που αποσκοπεί στην αυτοματοποιημένη αναδόμηση τμημάτων κώδικα με χρήση των συμπεριφορικών προτύπων σχεδίασης *κατάσταση (state)* και *στρατηγική (strategy)* (40) (75) (1). Η προσέγγιση αυτή παρουσιάζεται επί του παρόντος πολύ επιγραμματικά καθώς αναλυτική παρουσίαση της ακολουθεί στο επόμενο κεφάλαιο, εφόσον η παρούσα διπλωματική βασίστηκε σ' αυτή. Αναφορικά με την ανίχνευση των πιθανών σημείων χρήσης των δύο συμπεριφορικών αυτών προτύπων, η προσέγγιση ανιχνεύει μέσω *συντακτικής ανάλυσης (syntactic analysis)* του κώδικα τις υποθετικές δηλώσεις και εξετάζει ως πιθανές προς αναδόμηση όλες τις υποθετικές δηλώσεις, στις συνθήκες των οποίων γίνεται σύγκριση μιας *τοπικής μεταβλητής (local variable)*, ενός *πεδίου (field)* ή μιας *παραμέτρου (parameter)* με μια σταθερά *(public static final)* μέσω του *τελεστή ισότητας (equality operator)*. Για το σκοπό αυτό κατασκευάστηκε με χρήση της βιβλιοθήκης JDT του Eclipse, ένα plug-in ονόματι JDeodorant (24). Το JDeodorant υλοποιεί την προτεινόμενη μεθοδολογία με βάση τη χρήση αφηρημένων συντακτικών δέντρων *(Abstract Syntax Trees AST)* και της βιβλιοθήκης LTK *(Language Toolkit)* του Eclipse για την ανασύνταξη τους. Μέσω του γραφικού περιβάλλοντος παρέχεται στο χρήστη η δυνατότητα προεπισκόπησης των προτεινόμενων αναδομήσεων, οι οποίες παρουσιάζονται σε μια ταξινομημένη λίστα βάσει τριών κριτηρίων *(αριθμός εμφανίσεων σε επίπεδο συστήματος, αριθμός εμφανίσεων σε επίπεδο κλάσης και μέσος αριθμός εντολών ανά υποθετική δήλωση)*.

Αναφορικά με την αναδόμηση των δομικών πρότυπων, ο C. Jebelean, κατ’ αντιστοιχία με τη μεθοδολογία του περί της ανίχνευσης απουσίας του προτύπου *αφηρημένο εργοστάσιο*, παρουσιάζει σε συνεργασία με τους C.-B. Chirilă, και V. Creţu μια προσέγγιση για την ανίχνευση περιπτώσεων απουσίας του πρότυπου σχεδίασης *σύνθετο (composite)* σε αντικειμενοστραφή κώδικα (76). Ωστόσο η προσέγγιση αυτή περιορίζεται μόνο στην ανίχνευση και δεν επεκτείνεται και στην αυτόματη εκτέλεση της αναδόμησης. Για την ανίχνευση, ορίζεται η έννοια των *συμπτωμάτων (symptoms)*, ως εσφαλμένες υλοποιήσεις της δομής των προτύπων σχεδίασης που χρήζουν αναδόμησης. Συγκεκριμένα στην περίπτωση του σύνθετου ανιχνεύονται οι περιπτώσεις όπου απουσιάζει στον κώδικα η σχέση *γενίκευσης (generalisation, extends)* ανάμεσα στο κλάση *σύνθετο* και την κλάση *context*. Προκειμένου να εντοπιστούν οι περιπτώσεις αυτές γίνεται χρήση λογικού μέτα-προγραμματισμού *(logic metaprogramming)* μέσω της Prolog καθώς της κατάλληλης στρατηγικής εντοπισμού. Για την υλοποίηση της προσέγγισης αξιοποιείται το plug-in του Eclipse JTransformer, το οποίο λειτουργεί ως μηχανή εκτέλεσης *επερωτήσεων (queries)* και μετασχηματισμού του πηγαίου κώδικα σε Java. Με χρήση του JTransformer δημιουργείται μια αναπαράσταση *αφηρημένων συντακτικών δέντρων (AST)* ως βάση δεδομένων με Program Element Facts (PEFs) της Prolog. Αφού δημιουργηθεί η βάση δεδομένων χρησιμοποιείται η *μηχανή συμπερασμού (inference machine)* της Prolog για την εξαγωγή συμπερασμάτων αναφορικά με τα σημεία που χρήζουν αναδόμησης του υπό εξέταση συστήματος. Η συγκεκριμένη προσέγγιση αναφέρεται να ορίζει επίσης *συμπτώματα (symptoms)* για την αυτόματη ανίχνευση και αναδόμηση των προτύπων *κατάσταση (state), στρατηγική (strategy)* και *επισκέπτης (visitor)*, χωρίς όμως να αναφέρονται περαιτέρω λεπτομέρειες αναφορικά με τη λογική εντοπισμού της απουσίας των προτύπων αυτών.

3.1.2. Συμπεράσματα από την Επισκόπηση Βιβλιογραφίας

Η σύντομη επισκόπηση της βιβλιογραφίας αναφορικά με την αυτοματοποιημένη αναδόμηση των προτύπων σχεδίασης, καθιστά εμφανή την έλλειψη πλήρους κάλυψης του καταλόγου των αναδομήσεων με πρότυπα σχεδίασης που αναφέρει στο βιβλίο του ο Kerievsky. Συγκεκριμένα από τα 19 πρότυπα σχεδίασης, τα οποία πραγματεύονται οι αναδομήσεις που παρουσιάζει ο Kerievsky, μόνο για τα 5 από αυτά έχουν αναπτυχθεί και υλοποιηθεί μεθοδολογίες για τον αυτόματο εντοπισμό περιπτώσεων στις οποίες η χρήση τους θα μπορούσε να συμβάλει στην βελτίωση της ποιότητας του κώδικα. Το γεγονός αυτό οφείλεται στην εννοιολογική φύση των προτύπων σχεδίασης, που καθιστά δύσκολο τον εντοπισμό περιπτώσεων ελλιπούς χρήσης τους, καθώς και στην διαφορετική φύση κάθε προτύπου, η οποία καθιστά αδύνατη την υλοποίηση μιας γενικής προσέγγισης για τη συμπερίληψη και κάλυψη όλων των πιθανών αναδομήσεων με χρήση προτύπων.

Επιπλέον, ορισμένες από τις υπάρχουσες μεθοδολογίες δεν είναι γενικεύσιμες αναφορικά με την αναδόμηση του εκάστοτε προτύπου στο οποίο αναφέρονται, αλλά περιορίζονται συχνά σε συγκεκριμένες περιπτώσεις απουσίας που έχουν εντοπιστεί. Το γεγονός αυτό οφείλεται στην απειρία του χώρου των δυνατών διαφορετικών σχεδίων του κώδικα, η οποία καθιστά δύσκολο τον σαφή προσδιορισμό και την κάλυψη όλων των συγκεκριμένων ελαττωματικών δομών στα οποία αναφέρεται κάθε πρότυπο.

Η συγκεκριμένη διπλωματική καλείται να προσπαθήσει να συμβάλει στον τομέα της αυτόματης αναδόμησης, επανεξετάζοντας την προσέγγιση της συγγραφικής ομάδας των Τσάνταλη και Χατζηγεωργίου, περί της αναδόμησης με χρήση του προτύπου σχεδίασης *στρατηγική (strategy)*. Για το σκοπό αυτό στο επόμενο κεφάλαιο παρουσιάζεται διεξοδικά η συγκεκριμένη προσέγγιση καθώς και γενικότερα η αυτοματοποίηση της αναδόμησης προς τη στρατηγική ως ερευνητικό ζήτημα προς εξέταση.

3.2 Μέθοδοι Ανάλυσης Κώδικα στην Αυτοματοποίηση Αναδόμησης

Η μεθοδολογία που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας αξιοποιεί για την επίτευξη της αυτοματοποιημένης ανίχνευσης και αναδόμησης υποθετικών δηλώσεων στις οποίες ενδείκνυται το πρότυπο στρατηγική, την τεχνική της στατικής ανάλυσης. Οι στατικές τεχνικές που αφορούν κυρίως την εξερεύνηση των αφηρημένων συντακτικών δέντρων του κώδικα χρησιμοποιούνται αρχικά προκειμένου να εντοπιστούν οι υποψήφιες προτάσεις αναδόμησης, ορισμένες από τις οποίες αναθεωρούνται σε επόμενο στάδιο προκειμένου να διαπιστωθεί η δυνατότητα ή μη, πλήρους απαλοιφής της υποθετικής δομής κατά την εκτέλεση της αναδόμησης. Το υποσύνολο εκείνο στο οποίο η πλήρης αντικατάσταση είναι εφικτή εκτελείται προκειμένου να εντοπιστεί το κατάλληλο αντικείμενο στρατηγική ανάλογα με τις παραμέτρους κλήσης της μεθόδου.

Η δυναμική ανάλυση (dynamic analysis) ουσιαστικά αναλύει τις ιδιότητες ενός υπό εκτέλεση συστήματος λογισμικού και παρέχει πληροφορίες σχετικά με τη συμπεριφορά του (77). Η ανάλυση αυτή πραγματοποιείται με χρήση κάποιου πραγματικού ή εικονικού επεξεργαστή (real/virtual processor) και εξετάζει τις ιδιότητες του λογισμικού για συγκεκριμένες εισόδους (π.χ. ακολουθία κλήσεων μεθόδων, δεδομένα εξόδου, κλπ), συνεπώς οι πληροφορίες που προκύπτουν ισχύουν για μία ή ένα σύνολο συγκεκριμένων εκτελέσεων, αλλά δεν είναι δυνατό να γενικευτούν για κάθε πιθανή εκτέλεση του προγράμματος.

Αντίθετα από τη δυναμική ανάλυση, η στατική ανάλυση (static analysis) εξετάζει τον πηγαίο κώδικα στο χρόνο μεταγλώττισης χωρίς δεδομένα εισόδου και χωρίς να τον εκτελεί, παρέχοντας πληροφορίες αναφορικά με τη δομή του (τις κλάσεις, τις μεθόδους, τις μεταβλητές, κ.ο.κ.) και τον έλεγχο ροής, οι οποίες είναι γνωστές κατά το χρόνο αυτό. Χαρακτηριστικό των πληροφοριών που αντλούνται με αυτό τον τρόπο ανάλυσης, το οποίο τις διαφοροποιεί από τις δυναμικές πληροφορίες, είναι ότι αποτελούν συγκεκριμένες ιδιότητες του προγράμματος, οι οποίες ισχύουν για κάθε πιθανή εκτέλεση του και δεν εξαρτώνται από τα δεδομένα εισόδου του. Κυριότερες μορφές αναπαράστασης που παράγονται από την ανάλυση αυτή είναι οι εξής:

- Τα αφηρημένα συντακτικά δέντρα (abstract syntax tree - AST)

- Ο γράφος ροής ελέγχου (control flow graph - CFG)
- Ο γράφος εξαρτήσεων προγράμματος (program dependence graph - PDG)
- Το διάγραμμα ροής δεδομένων (data flow diagram - DFD).

Οι ανωτέρω μορφές αναπαράστασης ενός προγράμματος διαφέρουν ως προς τις πληροφορίες που παρέχουν καθώς επίσης και ως προς το βαθμό λεπτομέρειας των χαρακτηριστικών του πηγαίου κώδικα που απεικονίζουν, γι' αυτό και αξιοποιούνται στη λύση διαφορετικών προβλημάτων. Σύμφωνα με τον Kalleberg (78)

> «η αποτελεσματικότητα και εφαρμοσιμότητα ενός συστήματος λογισμικού που εκτελεί μετασχηματισμούς εξαρτάται σε μεγάλο βαθμό από τον τρόπο με τον οποίο έχει διαμορφωθεί το υποκείμενο μοντέλο του προγράμματος. Το μοντέλο καθορίζει ποιοι μετασχηματισμοί θα είναι εύκολοι, δύσκολοι ή αδύνατοι. Συγκεκριμένα η «αφαιρετικότητα» της αναπαράστασης καθορίζει ποιες αναλύσεις και ποιοι μετασχηματισμοί είναι εφικτοί – αν το μοντέλο είναι πολύ αφηρημένο, η αναδόμηση μπορεί να μην είναι εφικτή, και αν το μοντέλο είναι υπερβολικά λεπτομερές, πολλές αναλύσεις γίνονται πολύ ακριβές.»

Για τον λόγο αυτό, προκειμένου να επιτευχθεί ο σκοπός της παρούσας εργασίας η μεθοδολογία αναδόμησης που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας αξιοποίησε ως κύρια μορφή αναπαράστασης του πηγαίου κώδικα χρησιμοποιήθηκαν τα αφηρημένα συντακτικά δέντρα, τα οποία παρουσιάζονται αναλυτικά ακολούθως, καθώς άλλωστε χρησιμοποιούνται ευρέως από αντίστοιχα αυτοματοποιημένα εργαλεία αναδόμησης, ενώ σε περιπτώσεις όπου η ζητούμενη πληροφορία δεν μπορούσε να εξαχθεί από αυτά, χρησιμοποιήθηκαν συμπληρωματικά ο γράφος ροής ελέγχου και ο γράφος εξαρτήσεων προγράμματος, που παρουσιάζονται συνοπτικά στη συνέχεια. Η επιλογή αναπαραστάσεων παραγόμενων με αποκλειστική χρήση στατικής ανάλυσης έναντι της δυναμικής ενδείκνυται για τις ανάγκες της επιλεγμένης αναδόμησης, ενώ ταυτόχρονα συμβάλει στη βελτιστοποίηση της απόδοσης της εφαρμογής.

3.2.1. Αφηρημένα Συντακτικά Δέντρα (AST)

Τα αφηρημένα συντακτικά δέντρα (AST) αποτελούν μια από τις πλέον δημοφιλής αναπαραστάσεις πηγαίου κώδικα που χρησιμοποιούνται κατά την ανάλυση κώδικα από τους *μεταγλωττιστές (compilers).* Αποτελούν την πλήρη αναπαράσταση του πηγαίου κώδικα, με εξαίρεση τα κενά, τις κενές γραμμές και τα σχόλια, ως μια δενδρική απεικόνιση που αποτελείται από κόμβους και ακμές. Συγκεκριμένα, κάθε αφηρημένο συντακτικό δέντρο αποτελείται από μια ρίζα, ένα σύνολο από κόμβους φύλλα, οι οποίοι αναπαριστούν *τερματικά σύμβολα (tokens)*, όπως παραδείγματος χάριν οι αριθμητικές τιμές, οι συμβολοσειρές, οι τελεστές και οι χαρακτήρες, και ένα σύνολο *εσωτερικών κόμβων,* που αντιστοιχούν σε μη τερματικά σύμβολα όπως παραδείγματος χάριν οι δηλώσεις μεθόδων, οι δηλώσεις μεταβλητών, οι υποθετικές δηλώσεις, κ.ο.κ.

Τα αφηρημένα συντακτικά δέντρα ως μορφή αναπαράστασης του πηγαίου κώδικα, πλεονεκτούν έναντι άλλων αναπαραστάσεων του, καθώς παρέχουν μια ένα-προς-ένα αντιστοίχιση του συνόλου των συστατικών του, εν αντιθέσει με άλλες αναπαραστάσεις στις οποίες απεικονίζονται μόνο συγκεκριμένα από τα χαρακτηριστικά του κώδικα, καθιστώντας δύσκολη την απευθείας αντιστοίχιση. Παραδείγματα τέτοιων αναπαραστάσεων αποτελούν: ο *γράφος κλάσεων (class graph),* ο οποίος μοντελοποιεί μόνο το δηλωτικό κομμάτι του κώδικα δηλαδή τις δηλώσεις των κλάσεων, των ιδιοτήτων και των μεθόδων ενός συστήματος καθώς και τις μεταξύ τους συσχετίσεις, ο *γράφος ροής ελέγχου (control flow graph),* που μοντελοποιεί τα *μονοπάτια εκτέλεσης* (execution paths) κατά το *χρόνο εκτέλεσης (runtime),* δηλαδή απεικονίζει τα δυναμικά χαρακτηριστικά του εκτελέσιμου κώδικα, και ο οποίος παρουσιάζεται ακολούθως, και ο *γράφος ροής δεδομένων (data flow graph),* ο οποίος μοντελοποιεί μόνο τη ροή των δεδομένων. Ωστόσο, μειονεκτήματα των AST έναντι των υπολοίπων μοντέλων αναπαράστασης αποτελούν οι αυξημένες απαιτήσεις σε μνήμη για την αποθήκευση τους λόγω του όγκου της πληροφορίας που αποθηκεύουν, ο αυξημένος χρόνος για τη δημιουργία τους και η δυσκολία στη διάσχιση τους για την εξαγωγή πληροφοριών απ' αυτά, που σε άλλες αναπαραστάσεις διεξάγεται με ένα τρόπο άμεσο. Συγκεκριμένα για τη διάσχιση των AST δέντρων αξιοποιείται το πρότυπο *επισκέπτης (visitor)* (33).

Τα αφηρημένα συντακτικά δέντρα συνδέονται στενά με την BNF *(Backus-Naur Form)* γραμματική της εκάστοτε γλώσσας προγραμματισμού. Συνεπώς, οι κανόνες που ισχύουν στις BNF γραμματικές επεκτείνονται και στα αφηρημένα συντακτικά δέντρα. Ως εκ τούτου για τα AST ισχύουν οι εξής κανόνες (72):

- Υπάρχει πεπερασμένος αριθμός τύπων κόμβων κατ' αντιστοιχία με τον πεπερασμένο αριθμό μη-τερματικών (non-terminal) συμβόλων και των εκφράσεων που ορίζονται στη BNF γραμματική.

- Κάθε κόμβος σχετίζεται με ένα προκαθορισμένο σύνολο ιδιοτήτων-κόμβων, από το σύνολο όλων των διαθέσιμων κόμβων, σύμφωνα με το συντακτικό που ορίζει η γραμματική.
- Οι διάφοροι τύποι κόμβων υλοποιούνται συνήθως με τη μορφή κλάσεων, που ανήκουν σε μια ιεραρχία. Η ιεραρχία αυτή καθορίζεται με βάση την δομή των κανόνων που ορίζει η γραμματική της γλώσσας προγραμματισμού.
- Κάθε τύπος κόμβου διαθέτει ένα προκαθορισμένο σύνολο ιδιοτήτων, το οποίο ορίζεται πάλι σύμφωνα με το συντακτικό της γραμματικής.

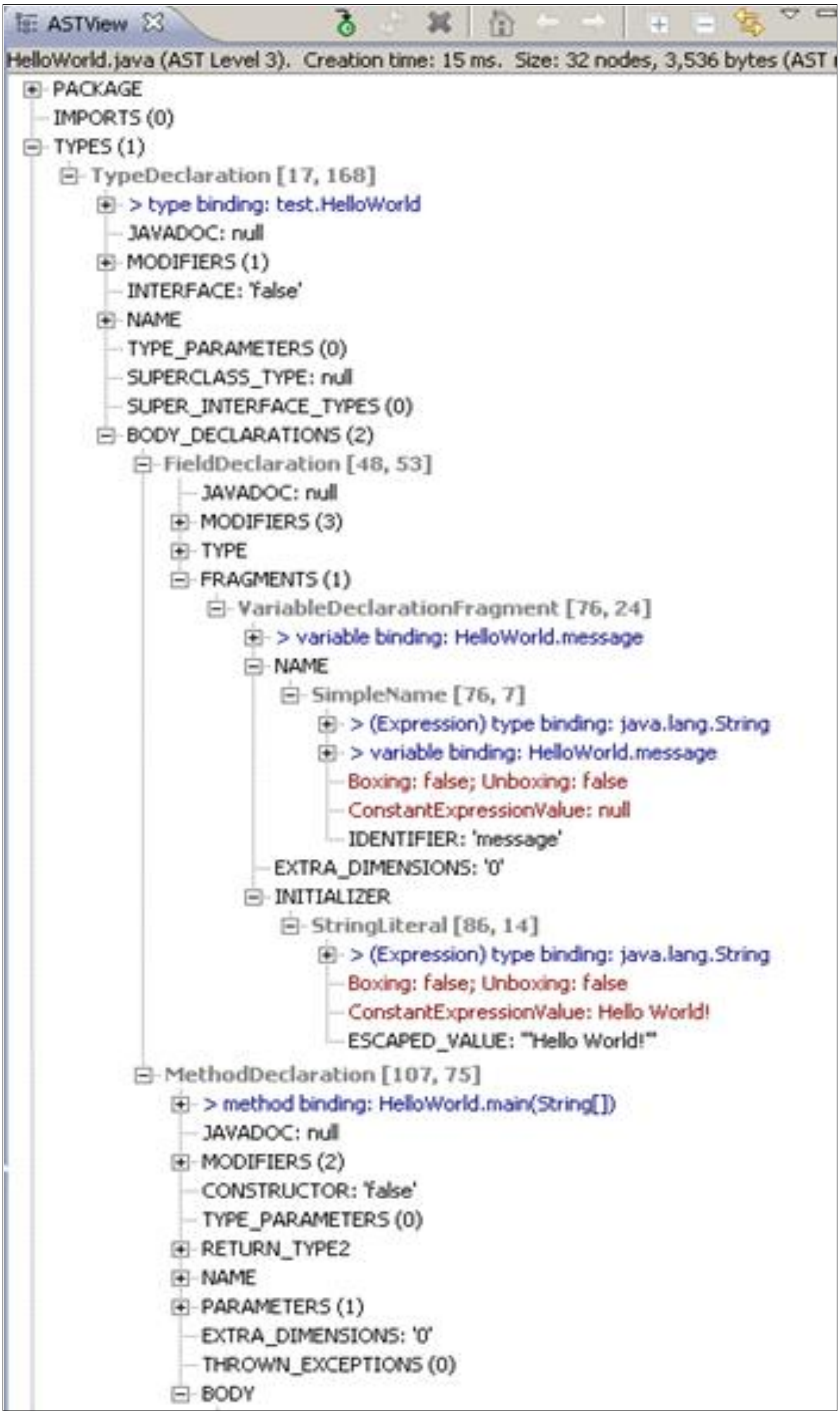
```
public class HelloWorld {

    private static final String message = "Hello World!";

    public static void main(String args[]){
        System.out.println(message);
    }
}
```

Εικόνα 10 : Παράδειγμα κλάσης HelloWorld.java

Στην Εικόνα 10 απεικονίζεται μια απλή κλάση HelloWorld.java και στην Εικόνα 11

Εικόνα 10 : Παράδειγμα κλάσης HelloWorld.java

ππαρουσιάζεται απόσπασμα του αντίστοιχου AST για την κλάση αυτή όπως παράγεται από το ASTViewer plug-in του Eclipse (79). Θεωρείται σημαντικό να τονιστεί ότι δεν υπάρχει μια επίσημη ευρέως αποδεκτή μορφή για τον τρόπο αναπαράστασης των δυαδικών δέντρων, συνεπώς στη βιβλιογραφία συναντώνται διάφορες άτυπες μορφές αναπαράστασης. Εφεξής τα αφηρημένα συντακτικά δέντρα θα αναφέρονται και ως AST.



Εικόνα 11 : Δημιουργία AST δέντρου της κλάσης HelloWorld από το plug-in ASTViewer

3.2.2. Γράφος Ροής Ελέγχου (CFG)

Σε αντίθεση με τα αφηρημένα συντακτικά δέντρα, τα οποία ως μορφή αναπαράστασης πλησιάζουν πολύ τον πηγαίο κώδικα, παρέχοντας μια ένα-προς-ένα αντιστοιχία προς τις οντότητες που τον απαρτίζουν, και αναπαριστώντας τον στο σύνολο του, ο γράφος ροής ελέγχου μοντελοποιεί μόνο συγκεκριμένα χαρακτηριστικά του κώδικα, ορισμένα από τα οποία δεν είναι άμεσα διαθέσιμα από τον πηγαίο κώδικα, ούτε μπορούν να εξαχθούν από το AST. Ο γράφος ροής ελέγχου εστιάζει στην απεικόνιση των πιθανών μονοπατιών εκτέλεσης κατά το χρόνο εκτέλεσης και στην γραφική παρουσίαση της ροής του ελέγχου στα πλαίσια ενός προγράμματος ή μιας διεργασίας.

Για το σκοπό αυτό χρησιμοποιεί κόμβους (nodes, vertices), οι οποίοι μοντελοποιούν τα βασικά συστατικά ενός προγράμματος (δηλώσεις, εντολές) και ακμές (edges), οι οποίες χρησιμοποιούνται για να αναπαραστήσουν την πιθανή ροή ελέγχου μεταξύ των κόμβων του γράφου, ενώνοντας τους (80). Οι ακμές αυτές είναι κατευθυνόμενες και τα άκρα τους συνδέουν τον κόμβο προκάτοχο (predecessor), από τον οποίο ξεκινάει η ακμή, με τον κόμβο διάδοχο (successor), στον οποίο καταλήγει η ακμή.

Ο γράφος ροής ελέγχου διευκολύνει την ανάλυση των διακλαδώσεων εκτέλεσης (execution branches) που προκύπτουν από τις υποθετικές δηλώσεις καθώς και τους επαναληπτικούς βρόχους (loops) που προκύπτουν από τις δηλώσεις επανάληψης (do, while, for, κλπ).Λόγω της δυνατότητας αναπαράστασης των μονοπατιών εκτέλεσης ο γράφος ροής ελέγχου αξιοποιείται στη βελτιστοποίηση μεταγλωττιστών και τα εργαλεία ανάλυσης, ενώ συχνή χρήση του αποτελεί ο έλεγχος προσπελασιμότητας (reachability check). Κατά την εκτέλεση κάποιου μετασχηματισμού αναδόμησης δεν χρησιμοποιείται άμεσα. Ωστόσο μια πιθανή χρήση του στα πλαίσια της αναδόμησης αποτελεί ο έλεγχος των προϋποθέσεων της, όπως παραδείγματος χάριν στην περίπτωση της αναδόμησης εξαγωγής μεθόδου για τον έλεγχο ότι το μονοπάτι εκτέλεσης δεν περιλαμβάνει εντολή διαφυγής (return ή break).

3.2.3. Γράφος Εξαρτήσεων Προγράμματος (PDG)

Κατ' αντιστοιχία με το γράφο ροής ελέγχου, ο γράφος εξαρτήσεων προγράμματος (program dependence graph PDG) μοντελοποιεί τις εξαρτήσεις ροής ελέγχου και επιπλέον και τις εξαρτήσεις ροής δεδομένων, με ένα σύνολο από κόμβους και ακμές.

Οι κόμβοι που απαρτίζουν ένα γράφο εξαρτήσεων προγράμματος απεικονίζουν τις εντολές του προγράμματος, ενώ οι ακμές μοντελοποιούν τις μεταξύ τους εξαρτήσεις. Υπάρχουν δύο ειδών ακμές:

- **Ακμή εξάρτησης ελέγχου** (control dependence edge): δείχνει ότι ο κόμβος πηγή καθορίζει την εκτέλεση ή μη του κόμβου προορισμού, ή τον αριθμό των φορών που θα εκτελεστεί.
- **Ακμή εξάρτησης δεδομένων** (data dependence edge): δείχνει ότι οι δύο υπολογισμοί των κόμβων πηγής και προορισμού χρησιμοποιούν τα ίδια δεδομένα (μεταβλητές) κατά τρόπο ώστε να μεταβληθεί η σημασιολογία του προγράμματος, αν αλλάξει η σειρά τους (81).

Πλεονέκτημα του γράφου εξαρτήσεων προγράμματος αποτελεί το γεγονός ότι οι ακμές εξαρτώνται από την πραγματική ροή δεδομένων και από τις εμφωλευμένες δηλώσεις και όχι από τη σειρά με την οποία εμφανίζονται οι εντολές στον πηγαίο κώδικα. Με χρήση του γράφου εξαρτήσεων δεδομένων είναι εύκολος ο εντοπισμός εντολών, οι οποίες δεν έχουν καμία εξάρτηση δεδομένων ή ελέγχου και κατ' επακόλουθο είναι εφικτό να αλλάξει η σειρά τους χωρίς να επηρεάζεται η σημασιολογία του κώδικα. Γι' αυτό το λόγο ο γράφος εξαρτήσεων προγράμματος βρίσκει εφαρμογή στη βελτιστοποίηση μεταγλωττιστών καθώς και στην τεχνική slicing. Η τελευταία αξιοποιείται στην παρούσα εργασία κατά τη διενέργεια του ελέγχου ικανοποιησιμότητας των κριτηρίων, και περιγράφεται συνοπτικά ακολούθως.

3.2.4. Slicing

Μια από τις μεθόδους που βρίσκει εφαρμογή στην αναδόμηση προγραμμάτων και κυρίως στην εξαγωγή μεθόδου (extract method) είναι ο *τεμαχισμός προγράμματος (program slicing).* Η μέθοδος αυτή χρησιμοποιείται για τη διάσπαση του τμήματος κώδικα ενός προγράμματος σε ένα σύνολο μικρότερου μεγέθους αυτόνομα τμήματα, τα λεγόμενα *τεμάχια (slices)*, τα οποία ωστόσο εξακολουθούν να έχουν την ίδια λειτουργικότητα/συμπεριφορά (82) (83). Βάσει του ορισμού του M. Weiser ως τεμάχιο (slice) ορίζεται το σύνολο των δηλώσεων ενός προγράμματος, οι οποίες ενδέχεται να μεταβάλουν την τιμή μιας μεταβλητής x σε ένα συγκεκριμένο σημείο ενδιαφέροντος p.

Κατ' αντιστοιχία με τις μεθόδους ανάλυσης προγραμμάτων ο τεμαχισμός διακρίνεται σε **στατικό** (static slicing), όπου ο υπολογισμός των τεμαχίων γίνεται με χρήση των στατικά διαθέσιμων πληροφοριών, και **δυναμικό** (dynamic slicing) όπου κατά το χρόνο εκτέλεσης του προγράμματος χρησιμοποιούνται οι συγκεκριμένες τιμές των μεταβλητών προκειμένου τα τεμάχια να υπολογιστούν με μεγαλύτερη ακρίβεια (84). Επιπλέον, σύμφωνα με τον διαχωρισμό που αναφέρει ο Τσάνταλης (85), ο τεμαχισμός ανάλογα με την κατεύθυνση στην οποία εφαρμόζεται διακρίνεται σε

- **ευθύς** (forward) και
- **ανάστροφος** (backward)

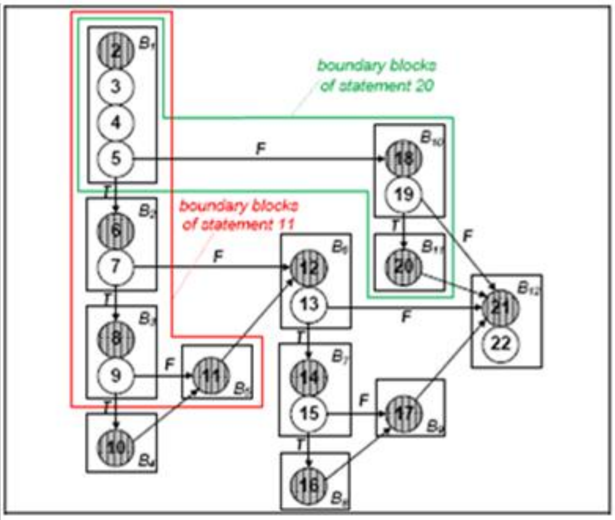
και ανάλογα με την εμβέλεια του σε

- **intraprocedural**, που περιορίζεται στα πλαίσια μιας διεργασίας και
- **interprocedural**, που επεκτείνεται εκτός των ορίων μιας διεργασίας.

Μια δημοφιλής τεχνική intraprocedural στατικού τεμαχισμού η οποία αξιοποιείται τόσο από τον Τσάνταλη κατά την αυτοματοποίηση της αναδόμησης εξαγωγής μεθόδου, όσο και από την προτεινόμενη προσέγγιση αναδόμησης προς τη στρατηγική της παρούσας εργασίας, είναι η τεχνική τεμαχισμού βάσει μπλοκ (**block-based slicing**) που πρότεινε ο Maruyama (86). Σύμφωνα με την τεχνική αυτή, ο υπολογισμός των τεμαχίων επιτυγχάνεται με την διάσπαση του κώδικα της μεθόδου σε block-based περιοχές (block-based regions). Με τη χρήση της προτεινόμενης τεχνικής προσδιορίζονται οι περιοχές εκείνες της μεθόδου, που αποτελούνται από την ένωση των τεμαχίων που ξεκινούν από δηλώσεις διαφορετικών μπλοκ και εξάγονται από τη μέθοδο.

Προκειμένου να επιτευχθεί αυτό ορισμένα από τα βήματα που ακολουθούνται περιλαμβάνουν τη δημιουργία του CFG γράφου της μεθόδου προκειμένου να διασπαστεί σε βασικά μπλοκ (basic block) και τη δημιουργία του PDG γράφου της μεθόδου προκειμένου να υπολογισθούν τα boundary blocks μιας δήλωσης n. Ως βασικό μπλοκ (basic block) ορίζεται μια ακολουθία διαδοχικών δηλώσεων στις οποίες η ροή του ελέγχου ξεκινάει από την αρχή και σταματάει στο τέλος χωρίς να σταματάει ενδιάμεσα ή να διακλαδίζεται, και αποτελείται υποχρεωτικά από ένα leader κόμβο καθώς και από όλους τους επόμενους κόμβους που μεσολαβούν μέχρι τον επόμενο leader κόμβο. Ως leader κόμβος θεωρείται είτε ο πρώτος κόμβος της μεθόδου, είτε ένας κόμβος συγχώνευσης (join node), είτε ένας κόμβος που βρίσκεται ακριβώς μετά από έναν κόμβο διακλάδωσης (branch node). Για τον υπολογισμό των boundary blocks μιας δήλωσης n που ανήκει στο βασικό block B, υπολογίζεται αρχικά το σύνολο των επόμενων μπλοκ με τα οποία συνδέεται το μπλοκ B (forward reachable blocks). Επίσης υπολογίζεται το σύνολο των dominated blocks του κόμβου r, δηλαδή του κόμβου ο οποίος βρίσκεται στο αμέσως προηγούμενο ιεραρχικά μπλοκ του B και συνδέεται άμεσα με τον leader κόμβο του μπλοκ B. Προκειμένου ένα μπλοκ να θεωρείται dominated από τον κόμβο r, θα πρέπει να υπάρχει μια εξάρτηση ελέγχου (control dependence) που μεταβαίνει από τον κόμβο r στο block αυτό. Υπολογίζοντας την τομή των επόμενων ιεραρχικά block του B (forward reachable blocks) και των dominated blocks του r, προκύπτει το σύνολο των boundary blocks της δήλωσης n. Τέλος ως block-based περιοχή ενός boundary block B_n θεωρείται το σύνολο των κόμβων που βρίσκονται σε επόμενα ιεραρχικά blocks (forward reachable blocks) από το B_n. Ακολούθως η Εικόνα 12 (85) απεικονίζει τις εντολές που απαρτίζουν τη μέθοδο printDocument() στο αριστερό τμήμα και στο δεξί τμήμα τα βασικά μπλοκ της μεθόδου, τους γραμμοσκιασμένους leader κόμβους τους καθώς και τα boundary blocks για δύο από τις εντολές της.

```
1  public void printDocument(Packet document) {
2    String author = "Unknown";
3    String title = "Untitled";
4    int startPos = 0, endPos = 0;
5    if (document.message_.startsWith("!PS")) {
6      startPos = document.message_.indexOf("author:");
7      if (startPos >= 0) {
8        endPos = document.message_.indexOf(
         ".", startPos + 7);
9        if (endPos < 0)
10         endPos = document.message_.length();
11       author = document.message_.substring(
         startPos + 7, endPos);
       }
12     startPos = document.message_.indexOf("title:");
13     if (startPos >= 0) {
14       endPos = document.message_.indexOf(
         ".", startPos + 6);
15       if (endPos < 0)
16         endPos = document.message_.length();
17       title = document.message_.substring(
         startPos + 6, endPos);
       }
     } else {
18     title = "ASCII DOCUMENT";
19     if (document.message_.length() >= 16)
20       author = document.message_.substring(8, 16);
     }
21   System.out.println(author);
22   System.out.println(title);
   }
```



Εικόνα 12 : Παράδειγμα μεθόδου, των βασικών μπλόκ της και των boundary blocks ορισμένων από τις εντολές της

Ο τρόπος που τα boundary blocks αξιοποιούνται στην προτεινόμενη προσέγγιση αυτοματοποίησης της αναδόμησης προς τη στρατηγική παρουσιάζεται αναλυτικά στην ενότητα 0.0.5.3.1, ενώ στο κεφάλαιο που ακολουθεί παρουσιάζεται αναλυτικά η προσέγγιση του Τσάνταλη αναφορικά με την αυτοματοποίηση της αναδόμησης.

Κεφάλαιο 4

Αυτόματη Αναδόμηση προς τη Στρατηγική στο JDeodorant

Η αναδόμηση του πηγαίου κώδικα προς τη στρατηγική αποτελεί ένα ενδιαφέρον ερευνητικό ζήτημα στον τομέα της αναδόμησης καθώς η στρατηγική αποτελεί ένα από τα πλέον συχνά χρησιμοποιούμενα πρότυπα σχεδίασης (87).

Η στρατηγική, αξιοποιώντας την κληρονομικότητα και τον πολυμορφισμό, χρησιμοποιείται σε περιπτώσεις όπου απαιτείται η επιλογή της καταλληλότερης υλοποίησης ενός αλγορίθμου μεταξύ των διαφόρων εναλλακτικών παραλλαγών που παρέχονται από το σύστημα. Σε περιπτώσεις όπου, λόγω άγνοιας, εσφαλμένης αντίληψης για τη χρήση του προτύπου, βεβιασμένης υλοποίησης, έλλειψης πρόληψης για τις μελλοντικές απαιτήσεις του συστήματος, κ.ο.κ., η χρήση της στρατηγικής, παρόλο που ενδείκνυται απουσιάζει, η επιλογή της υλοποίησης που θα εκτελεστεί, συντελείται με χρήση υποθετικής λογικής. Ωστόσο, η υποθετική λογική συχνά χαρακτηρίζεται ως προβληματική καθώς εισάγει πρόσθετη πολυπλοκότητα σε ένα σύστημα καθιστώντας δύσκολη τη συντηρησιμότητα και την κατανοησιμότητα του. Επιπλέον, πλήττεται και η ελεγξιμότητα του πηγαίου κώδικα, καθώς η ύπαρξη πολύπλοκων υποθετικών δηλώσεων στο σύστημα καθιστά αναγκαία την αύξηση των διαφορών δοκιμασιών ελέγχου (test cases) που πρέπει να εκτελεστούν για την κάλυψη όλων των διαφορετικών διακλαδώσεων.

Η πολυπλοκότητα που εισάγεται από την εσφαλμένη χρήση υποθετικών δηλώσεων είναι ιδιαίτερα σημαντική όπως καταδεικνύεται και από το γεγονός ότι η *κυκλωματική πολυπλοκότητα (cyclomatic complexity)* (88), η μετρική η οποία συναρτά την πολυπλοκότητα του λογισμικού με τον αριθμό διακλαδώσεων υποθετικής λογικής και η οποία χρησιμοποιείται ευρέως, προκειμένου να διατηρηθεί η αξιοπιστία, η ελεγξιμότητα και η διαχειρισιμότητα του λογισμικού, αποτελεί τη δημοφιλέστερη μετρική πολυπλοκότητας λογισμικού.

Λύση στο πρόβλημα της αυξημένης πολυπλοκότητας των υποθετικών δηλώσεων που υποδεικνύεται από την κυκλωματική πολυπλοκότητα καλείται να προσφέρει η αναδόμηση με τη χρήση στρατηγικής. Ως εκ τούτου, η ανίχνευση προβληματικών σημείων στον πηγαίο κώδικα για χρήση της στρατηγικής, ανάγεται στο πρόβλημα εντοπισμού πολύπλοκης υποθετικής λογικής στον κώδικα και απλοποίησης της. Συνεπώς καθίσταται αναγκαίο να εξεταστεί περαιτέρω το πρόβλημα της χρήσης πολύπλοκης υποθετικής λογικής.

4.1. Η Χρήση Πολύπλοκης Υποθετικής ως Πρόβλημα

Οι υποθετικές δηλώσεις συνιστούν μια από τις συχνότερες δομές που χρησιμοποιούνται κατά τη συγγραφή ενός συστήματος λογισμικού καθώς αποτελούν μια εκ των τριών δομών, η αποκλειστική χρήση των οποίων μπορεί να συντελέσει στην παραγωγή/συγγραφή οποιουδήποτε προγράμματος ανεξαρτήτως της λειτουργικότητας του (89). Ωστόσο, συχνά η χρήση τους σχετίζεται με αυξημένη πολυπλοκότητα, η οποία, βάσει της αρνητικής επίδρασης της στα χαρακτηριστικά της ποιότητας του σχεδίου θα πρέπει να ελαχιστοποιείται.

Η προβληματική χρήση της υποθετικής λογικής πηγάζει συνήθως από τις συχνές τροποποιήσεις που υφίσταται ένα σύστημα κατά την προσπάθεια υλοποίησης νέων χαρακτηριστικών καθώς και αλλαγών στις απαιτήσεις, οι οποίες συχνά εκτελούνται βεβιασμένα υπό την πίεση αυστηρών προθεσμιών παράδοσης χωρίς να συνοδεύονται από επανασχεδίαση του υπάρχοντα κώδικα, ούτως ώστε οι νέες υλοποιήσεις να εισαχθούν ομαλά στο σύστημα. Επιπλέον αιτία της πολυπλοκότητας αποτελεί το γεγονός ότι αρκετοί προγραμματιστές μη έχοντας αφομοιώσει τις έννοιες της αντικειμενοστραφούς σχεδίασης χρησιμοποιούν υποθετικές δηλώσεις σε περιπτώσεις όπου θα έπρεπε να αξιοποιηθεί ο πολυμορφισμός, για την προσομοίωση του dynamic dispatch[5] και του late binding[6] (1) .

Άμεσο επακόλουθο των ανωτέρω αιτιών αποτελεί η αυξημένη χρήση πολύπλοκης υποθετικής λογικής (conditional logic) στον κώδικα, η οποία αποτελεί μια από τις πλέον ευρέως διαδεδομένες *«κακές οσμές» (bad smells)* κυρίως λόγω της πρόσθετης πολυπλοκότητας που επιφέρει αλλά και της αυξημένης συχνότητας με την οποία συναντάται

[5] Dynamic Dispatch: ο μηχανισμός εκτέλεσης του dynamic binding ορίζεται ως dynamic dispatch. Αποτελεί ουσιαστικά τη διαδικασία αντιστοίχησης μιας κλήσης μεθόδου σε μια συγκεκριμένη ακολουθία κώδικα δυναμικά κατά τη διάρκεια του χρόνου εκτέλεσης (runtime) και όχι κατά το χρόνο μεταγλώττισης (compile time). (118)

[6] Late binding: ο όρος χρησιμοποιείται για να εκφράσει τις περιπτώσεις εκείνες όπου η διαδικασία name binding, δηλαδή η συσχέτιση των αντικειμένων με αναγνωριστικά (identifiers), εκτελείται κατά τη διάρκεια του χρόνου εκτέλεσης. (118)

στον κώδικα. Συγκεκριμένα, στο (90) αναφέρεται πώς την πιο δημοφιλή αλλαγή στις κλάσεις των υπό εξέταση συστημάτων αποτελεί η αλλαγή στη λογική ελέγχου και συγκεκριμένα στις δηλώσεις if, ενώ στο βιβλίο της συγγραφικής ομάδας GoF (33),οι εκτενείς υποθετικές δηλώσεις χαρακτηρίζονται ως ανεπιθύμητες, μονολιθικές δομές που καθιστούν δύσκολη την κατανόηση, την επέκταση και την τροποποίηση του κώδικα.

Επιπλέον η χρήση υποθετικής λογικής συμβάλει στην *επανάληψη κώδικα (code duplication)*, καθώς τα τμήματα των υποθετικών δηλώσεων βρίσκονται διασκορπισμένα σε διαφορετικά σημεία του συστήματος για διαφορετικούς σκοπούς, δυσχεραίνοντας κατ' αυτό τον τρόπο, τόσο την συντήρηση όσο και την προσθήκη νέας λειτουργικότητας, που ενδέχεται να προκύψει από μελλοντικές αλλαγές των απαιτήσεων. Η αυξημένη πολυπλοκότητα και η επανάληψη κώδικα αυξάνουν με τη σειρά τους την πιθανότητα εισαγωγής σφαλμάτων κατά τη συντήρηση ή την προσθήκη νέας λειτουργικότητας, που συνεπάγεται την άμεση αύξηση των εκτελούμενων δοκιμασιών ελέγχου *(test suites)* για την πληρέστερη κάλυψη των διαφορετικών μονοπατιών εκτέλεσης, καθιστώντας αναγκαία την αναδόμηση κώδικα προκειμένου να ελαχιστοποιηθούν οι υποθετικές δηλώσεις.

Για το λόγο αυτό έχουν προταθεί διάφορες μεθοδολογίες προκειμένου να απαλειφθεί η πολύπλοκή υποθετική λογική. Ωστόσο, ανάλογα με τη δομή με την οποία εμφανίζεται στον κώδικα, η χρήση υποθετικών δηλώσεων επιτελεί διαφορετικό σκοπό και χρήζει διαφορετικής αντιμετώπισης προκειμένου να τροποποιηθεί, ούτως ώστε να μειωθεί η πολυπλοκότητα του σχεδίου.

Ο Fowler στο βιβλίο του αφιερώνει ένα ολόκληρο κεφάλαιο στην απλοποίηση υποθετικών εκφράσεων (8), ενώ στο κεφάλαιο που περιγράφει τις *«κακές οσμές» (bad smells)* ως ενδείξεις προβληματικών σημείων στον κώδικα που χρήζουν αναδόμησης, αναφέρεται στη χρήση δηλώσεων switch, ως μία από αυτές. Συγκεκριμένα η χρήση δηλώσεων switch παρουσιάζεται ως ένδειξη για τη χρήση πολυμορφισμού, ο οποίος αποκρύπτει τις λεπτομέρειες της συμπεριφοράς από τον καλούντα και συνεπώς καθιστά ευκολότερη την επέκταση της λειτουργικότητας. Ως τέτοια συνδέεται με διάφορες απλές και σύνθετες αναδομήσεις όπως η *αντικατάσταση κωδικού που υποδηλώνει τύπο με κατάσταση/στρατηγική* (*Replace Type Code with State/Strategy), αντικατάσταση υποθετικής δομής με πολυμορφισμό (Replace Conditional with Polymorphism), εισαγωγή null αντικειμένου (Introduce Null Object)*, κ.ο.κ.

Κατ' αντιστοιχία ο Kerievsky στο βιβλίο του συσχετίζει την απλοποίηση πολύπλοκής υποθετικής λογικής με τέσσερις διαφορετικές αναδομήσεις με πρότυπα σχεδίασης (16):

- Αναδόμηση προς τη *στρατηγική* (Replace Conditional Logic with Strategy):
  η χρήση της συστήνεται σε περιπτώσεις όπου η υποθετική λογική ελέγχει την εκτέλεση μίας από τις διάφορες παραλλαγές ενός αλγορίθμου.
- Αναδόμηση προς την *κατάσταση* (Replace State-Altering Conditionals with State):
  η χρήση της προτείνεται για τον έλεγχο των μεταβάσεων μεταξύ των καταστάσεων ενός αντικειμένου.
- Αναδόμηση προς τον *διακοσμητή* (Move Embellishment to Decorator):
  η χρήση της προτείνεται για την επιλογή των διαφόρων τμημάτων εξειδικευμένης συμπεριφοράς, τα οποία θα πρέπει να εκτελεστούν συμπληρωματικά ως προς την εκτέλεση της κύριας συμπεριφοράς της κλάσης.
- Εισαγωγή null αντικειμένου (Introduce Null Object):
  η χρήση του αντικειμένου null αντικαθιστά την υποθετική λογική, σε περιπτώσεις όπου, ο έλεγχος για το κατά πόσο μια μεταβλητή έχει την τιμή null, επαναλαμβάνεται συχνά σε διάφορα σημεία του συστήματος.

Ειδικά για τις πολύπλοκες δηλώσεις *switch* ο Kerievsky ορίζει δύο ακόμα αναδομήσεις:

- Αναδόμηση προς την *εντολή* (Replace Conditional Dispatcher with Command):
  η χρήση της ενδείκνυται για την αντικατάσταση μιας εκτενούς δήλωσης switch από μια συλλογή αντικειμένων *εντολών*, των οποίων η αναζήτηση και κλήση μπορεί να επιτευχθεί χωρίς να ελέγχεται κατ' ανάγκη από την υποθετική λογική.
- Αναδόμηση προς τον επισκέπτη (Move Accumulation to Visitor):
  η χρήση του προτείνεται σε περιπτώσεις όπου οι δηλώσεις switch χρησιμοποιούνται για να αντλήσουν δεδομένα από στιγμιότυπα κλάσεων με διαφορετικές διεπαφές.

Στη συσχέτιση της απλοποίησης υποθετικής λογικής με διαφορετικές αναδομήσεις με πρότυπα σχεδίασης, ανάλογα με το σκοπό που επιτελεί, αναφέρεται και η ερευνητική εργασία των A. Mukhtar και F. Lodhi (91). Ο

Πρότυπο Σχεδίασης	**Είδος Υποθετικής Λογικής**

Μέθοδος Εργοστάσιο (Factory Method)	Υποθετική λογική που περιλαμβάνει δημιουργία αντικειμένων
Αφηρημένο Εργοστάσιο (Abstract Factory)	Υποθετική λογική που αναλαμβάνει τη δημιουργία οικογενειών αντικειμένων διαφορετικών τύπων
Σύνθετο (Composite)	Υποθετική λογική της οποίας η δομή δημιουργεί ένα δέντρο
Διακοσμητή (Decorator)	Υποθετική λογική που παρέχει την εκτέλεση πρόσθετης λειτουργικότητας σε σχέση με την κύρια λειτουργικότητα μιας κλάσης
Εντολή (Command)	Υποθετική λογική που χρησιμοποιείται για τον χειρισμό αιτήσεων και την εκτέλεση ενεργειών
Κατάσταση (State)	Εκτενής υποθετικές δηλώσεις με πολλές διακλαδώσεις, που εξαρτώνται από την κατάσταση του αντικειμένου. Η κατάσταση αναπαρίσταται συνήθως από μια ή περισσότερες σταθερές – απαριθμητές (enumerations). Συχνά οι ίδιες υποθετικές δηλώσεις επαναλαμβάνονται σε διάφορες μεθόδους.
Στρατηγική (Strategy)	Υποθετικές δηλώσεις στις διακλαδώσεις των οποίων περιλαμβάνονται πολλές εναλλακτικές συμπεριφορές.
Αλληλουχία Αρμοδιοτήτων (Chain of Responsibility)	Υποθετικές δηλώσεις με τη χρήση του τελεστή instanceof στις συνθήκες τους, οι οποίες αναλαμβάνουν τον εντοπισμό της κατάλληλης κλάσης *(handler)* για το χειρισμό μιας συγκεκριμένης αίτησης.

Πίνακας 7 απεικονίζει τη συσχέτιση αυτή ανάμεσα στα πρότυπα σχεδίασης και στο είδος της υποθετικής λογικής, το οποίο δύνανται να αντικαταστήσουν.

Εν κατακλείδι, από τις ανωτέρω προσεγγίσεις συμπεραίνεται ότι ανάλογα με τον σκοπό που επιτελεί η υποθετική λογική, διακρίνεται από διαφορετικά χαρακτηριστικά και χρήζει διαφορετικής αντιμετώπισης κατά την αναδόμηση. Η προσέγγιση των συγγραφέων Τσάνταλη και Χατζηγεωργίου στην οποία βασίστηκε η παρούσα διπλωματική, εστιάζει στην αναγνώριση προβληματικών σημείων στον κώδικα, με χρήση υποθετικής δηλώσεων που βασίζονται στον *έλεγχο τύπου (type-checking)* και στην αντικατάσταση τους, με την αξιοποίηση της αναδόμησης με χρήση των προτύπων *κατάσταση* και *στρατηγική*. Στην ενότητα που ακολουθεί παρουσιάζεται αναλυτικά η συγκεκριμένη προσέγγιση.

Πρότυπο Σχεδίασης	Είδος Υποθετικής Λογικής
Μέθοδος Εργοστάσιο (Factory Method)	Υποθετική λογική που περιλαμβάνει δημιουργία αντικειμένων
Αφηρημένο Εργοστάσιο (Abstract Factory)	Υποθετική λογική που αναλαμβάνει τη δημιουργία οικογενειών αντικειμένων διαφορετικών τύπων
Σύνθετο (Composite)	Υποθετική λογική της οποίας η δομή δημιουργεί ένα δέντρο
Διακοσμητή (Decorator)	Υποθετική λογική που παρέχει την εκτέλεση πρόσθετης λειτουργικότητας σε σχέση με την κύρια λειτουργικότητα μιας κλάσης
Εντολή (Command)	Υποθετική λογική που χρησιμοποιείται για τον χειρισμό αιτήσεων και την εκτέλεση ενεργειών
Κατάσταση (State)	Εκτενής υποθετικές δηλώσεις με πολλές διακλαδώσεις, που εξαρτώνται από την κατάσταση του αντικειμένου. Η κατάσταση αναπαρίσταται συνήθως από μια ή περισσότερες σταθερές – απαριθμητές (enumerations). Συχνά οι ίδιες υποθετικές δηλώσεις επαναλαμβάνονται σε διάφορες μεθόδους.
Στρατηγική (Strategy)	Υποθετικές δηλώσεις στις διακλαδώσεις των οποίων περιλαμβάνονται πολλές εναλλακτικές συμπεριφορές.
Αλληλουχία Αρμοδιοτήτων (Chain of Responsibility)	Υποθετικές δηλώσεις με τη χρήση του τελεστή instanceof στις συνθήκες τους, οι οποίες αναλαμβάνουν τον εντοπισμό της κατάλληλης κλάσης *(handler)* για το χειρισμό μιας συγκεκριμένης αίτησης.

Πίνακας 7 : Αντιστοίχιση είδους υποθετικής λογικής στο κατάλληλο προς αναδόμηση πρότυπο σχεδίασης

4.2. Αυτόματη Αναδόμηση Υποθετικής Λογικής στο JDeodorant

Το ερευνητικό έργο των συγγραφέων Τσάνταλη και Χατζηγεωργίου ξεχωρίζει στον τομέα της αναδόμησης καθώς αποτελείται από την αυτοματοποιημένη υλοποίηση της ανίχνευσης υποψήφιων τμημάτων κώδικα προς αναδόμηση με πρότυπα σχεδίασης, καθώς και από την υποστήριξη πλήρους αυτοματοποιημένης εφαρμογής των προτεινόμενων αναδομήσεων (40).

Η προσέγγιση αυτή αναφορικά με την αναδόμηση υποθετικής λογικής εστιάζει σε δύο από τις περιπτώσεις αναδόμησης που παρουσιάζει ο Fowler στο βιβλίο του, καθεμία από τις οποίες βασίζεται σε διαφορετικά κριτήρια ανίχνευσης και τυγχάνει διαφορετικού χειρισμού, κατά τη διαδικασία εκτέλεσης της αναδόμησης. Οι περιπτώσεις που υλοποιεί το JDeodorant είναι οι εξής (75) (1):

- **Αντικατάσταση κωδικού που υποδηλώνει τύπο με κατάσταση/στρατηγική** (Replace type code with state / strategy)
- **Αντικατάσταση υποθετικής λογικής με πολυμορφισμό** (Replace conditional logic with polymorphism)

Στην πρώτη περίπτωση ανιχνεύονται οι υποθετικές εκφράσεις που ελέγχουν την τιμή μιας μεταβλητής τύπου *(type-checking conditionals),* και αντικαθίστανται από την χρήση των προτύπων κατάσταση και στρατηγική. Αντίστοιχα, στη δεύτερη περίπτωση ανιχνεύονται οι υποθετικές εκφράσεις στις οποίες ελέγχεται ο τύπος μιας μεταβλητής κατά το *χρόνο εκτέλεσης (RunTime Type Identification, RTTI)* με χρήση του τελεστή instanceof και αντικαθίστανται από την χρήση πολυμορφισμού.

Η προσέγγιση των συγγραφέων Τσάνταλη και Χατζηγεωργίου ουσιαστικά επιτυγχάνεται μέσω της συντακτικής ανάλυσης του κώδικα, της παραγωγής του αντίστοιχου αφηρημένου συντακτικού δέντρου του υπό εξέταση συστήματος και της διάσχισής του, προκειμένου να εντοπιστούν υποθετικές δομές που ικανοποιούν τα κριτήρια που ορίζονται από τη μεθοδολογία. Για την δημιουργία του AST αξιοποιούνται οι δυνατότητες που παρέχει η βιβλιοθήκη του

Eclipse JDT (92) (93) (94).Στο στάδιο της εκτέλεσης αναδόμησης, οι δυνατότητες της βιβλιοθήκης JDT συνδυάζονται με την υποστήριξη της αναδόμησης της βιβλιοθήκης LTK του Eclipse (95), προκειμένου να ανασυνταχθεί το AST δέντρο και εν συνεχεία να τροποποιηθεί ο πηγαίος κώδικας, βάσει του τροποποιημένου AST δέντρου.

4.2.1. Ανίχνευση Type-Checking Υποθετικών Δομών

Η ανίχνευση των προτάσεων αναδόμησης επιτυγχάνεται μέσω του εντοπισμού στον κώδικα δηλώσεων if και switch που παρουσιάζουν συγκεκριμένα χαρακτηριστικά ως προς τη δομή και χρήζουν αναδόμησης. Η ύπαρξη των χαρακτηριστικών αυτών ελέγχεται από ένα σύνολο καθορισμένων κριτήριων, τα οποία προσδιορίζονται από τη συγκεκριμένη προσέγγιση και εστιάζουν αποκλειστικά και μόνο στο πλήθος και τη δομή των υποθετικών εκφράσεων/συνθηκών *(conditions),* χωρίς να εξετάζουν τον κώδικα των διακλαδώσεων.

Συγκεκριμένα, το κριτήριο αναφορικά με το πλήθος των εναλλακτικών συνθηκών από τις οποίες θα πρέπει να απαρτίζονται οι δηλώσεις switch και if/else-if, ορίζεται σε 2 διακλαδώσεις κατ' ελάχιστον. Αναφορικά με τη δομή των υποθετικών εκφράσεων (conditions), το κριτήριο παραλλάσσεται ανάμεσα στις δηλώσεις if και switch.

Προκειμένου να θεωρηθεί υποψήφια μια δήλωση switch θα πρέπει η έκφραση της να περιλαμβάνει *τοπική μεταβλητή (local variable), παράμετρο (parameter)* ή μια μεταβλητή μέλος (instance variable) *πρωταρχικού (primitive)* ή *απαριθμητού τύπου (enum),* ή εναλλακτικά την κλήση της αντίστοιχης *getter* μεθόδου μιας μεταβλητής μέλους ενός από τους ανωτέρω τύπους. Οι αντίστοιχες δηλώσεις *case* θα πρέπει επίσης να περιλαμβάνουν πεδία *πρωταρχικού* ή *απαριθμητού τύπου,* τα όποια να έχουν *δημόσια προσπέλαση, στατική* εμβέλεια και να είναι *τελικά (public static final)* (Εικόνα 13 α). Συνεπώς, περιπτώσεις όπου στις δηλώσεις *case* περιλαμβάνονται απλές αριθμητικές τιμές *(number literal)* ή χαρακτήρες *(char literal)* δε λαμβάνονται υπόψη.

Αντίστοιχα, προκειμένου να θεωρηθούν υποψήφιες οι υποθετικές δηλώσεις if/else…if (Εικόνα 13 β), θα πρέπει να ικανοποιούνται τα εξής κριτήρια:

- Ο έλεγχος θα πρέπει να εκτελείται βάσει της ίδιας *τοπικής μεταβλητής, παραμέτρου* ή *μεταβλητής μέλους* σε όλες τις υποθετικές εκφράσεις που συνιστούν την υποθετική δήλωση.
- Ο μοναδικός τύπος ελέγχου που θεωρείται υποψήφιος προς εξέταση στις συνθήκες είναι ο έλεγχος ισότητας, ο οποίος ορίζεται με μια infix έκφραση που περιλαμβάνει τον *τελεστή ισότητας (equality operator)* ανάμεσα στους δύο διαφορετικούς τελεστές *(operands)*. Επιπλέον το κριτήριο εξετάζει και τον τύπο των συγκρινόμενων τελεστών ούτως ώστε ο ένας εκ των δύο να συνιστά είτε την κλήση μιας *getter* μεθόδου, είτε κάποια *τοπική μεταβλητή, παράμετρο* ή *μεταβλητή* μέλος, και ο έτερος να αποτελεί *τελική μεταβλητή μέλος* με *δημόσια προσπέλαση* και *στατική* εμβέλεια *(public static final), πρωταρχικού* ή *απαριθμητού* τύπου.

a: *switch* statement

```
class Context {
  private int type;
  public void m() {
    switch(type) {
      case VALUE_0:
      //code for case 0
      case VALUE_1:
      //code for case 1
      case …
    }
  }
}
```

b: *if/else if* statement

```
class Context {
  private int type;
  public void m() {
    if(type == VALUE_0) {
      //code for case 0
    }
    else if(type == VALUE_1) {
      //code for case 1
    }
    else if(type ==…) {…}
  }
}
```

Εικόνα 13 : Παραδείγματα type-checking κώδικα όπου το γνώρισμα αναπαριστά κατάσταση

4.2.2. Ανίχνευση RunTime Type Checking Υποθετικών Δομών

Η προσέγγιση που ακολουθείται για την ανίχνευση υποθετικών δομών που εκτελούν έλεγχο του τύπου μιας μεταβλητής κατά το χρόνο εκτέλεσης *(RTTI)*, είναι αντίστοιχη με την προσέγγιση που ορίζεται από τη μεθοδολογία για την ανίχνευση περιπτώσεων ελέγχου τύπου (*type-checking)*. Ωστόσο, η περίπτωση αυτή αναφέρεται αποκλειστικά μόνο σε δομές if/else…if, καθώς στην έκφραση του switch οι μόνοι τύποι που επιτρέπονται από το συντακτικό της java είναι οι *ακέραιοι (int)* και οι *χαρακτήρες (char)*, ενώ επιπλέον ο έλεγχος ενός τύπου κατά το χρόνο εκτέλεσης επιτυγχάνεται κυρίως με χρήση του τελεστή instanceof, η ύπαρξη του οποίου στην έκφραση μιας δήλωσης switch παραβιάζει το συντακτικό της.

Το υπό εξέταση σύνολο υποθετικών δηλώσεων if/else…if αποτελείται μόνο από το υποσύνολο εκείνο των δηλώσεων του προγράμματος, οι οποίες διαθέτουν τουλάχιστον δύο διακλαδώσεις. Από αυτές στον τελικό σύνολο των υποψήφιων θεωρούνται ότι ανήκουν μόνο όσες ικανοποιούν τις ακόλουθες προϋποθέσεις:

- Ο έλεγχος θα πρέπει να εκτελείται βάσει της ίδιας *τοπικής μεταβλητής, παραμέτρου* ή *μεταβλητής μέλους* σε όλες τις υποθετικές εκφράσεις που συνιστούν την υποθετική δήλωση.
- Οι συνθήκες θα πρέπει να περιλαμβάνουν είτε τη σύγκριση της μεταβλητής μέσω του τελεστή instanceof με τον τύπο μιας κλάσης (Εικόνα 14a), είτε μια σύγκριση ανάμεσα σε δύο τελεστές μέσω του *τελεστή ισότητας (equality operator),* όπου ο ένας θα πρέπει να αποτελεί τον επιστρεφόμενο τύπο της μεταβλητής όπως προκύπτει μέσω της κλήσης της μεθόδου getClass(), με *καλούντα (invoker)* την ίδια τη μεταβλητή *(v.getClass()),* και ο άλλος τελεστής θα πρέπει να είναι ένας τύπος *(type literal) (Type.class)* (Εικόνα 14b).Ένας επιπλέον έλεγχος που θα πρέπει να ικανοποιείται και στις δύο περιπτώσεις, είναι ο έλεγχος ότι ο τύπος κλάσης αποτελεί υποτύπο του τύπου της εξεταζόμενης μεταβλητής. Στην Εικόνα 14c απεικονίζεται ο εναλλακτικός τρόπος έκφρασης των περιπτώσεων a και b, όπως επιτυγχάνεται με χρήση του πολυμορφισμού.

a: *instanceof*

```
class Client {
  public void m(SuperType type) {
    if(type instanceof Subclass0) {
      Subclass0 s = (Subclass0)type;
      //code for case 0
    } else
    if(type instanceof Subclass1) {
      Subclass1 s = (Subclass1)type;
      //code for case 1
    }
  }
}
```

b: *getClass*

```
class Client {
  public void m(SuperType type) {
    if(type.getClass() == Subclass0.class) {
      Subclass0 s = (Subclass0)type;
      //code for case 0
    } else
    if(type.getClass() == Subclass1.class) {
      Subclass1 s = (Subclass1)type;
      //code for case 1
    }
  }
}
```

c: The subclass type is polymorphically obtained

```
class Client {
  public void m(SuperType type) {
    if(type.getType() == STATIC_VALUE_0) {
      Subclass0 sub = (Subclass0)type;
      //code for case 0
    }
    else if(type.getType() == STATIC_VALUE_1) {
      Subclass1 sub = (Subclass1)type;
      //code for case 1
    }
  }
}
```

Εικόνα 14 : Παραδείγματα type-checking κώδικα που εκτελεί RTTI

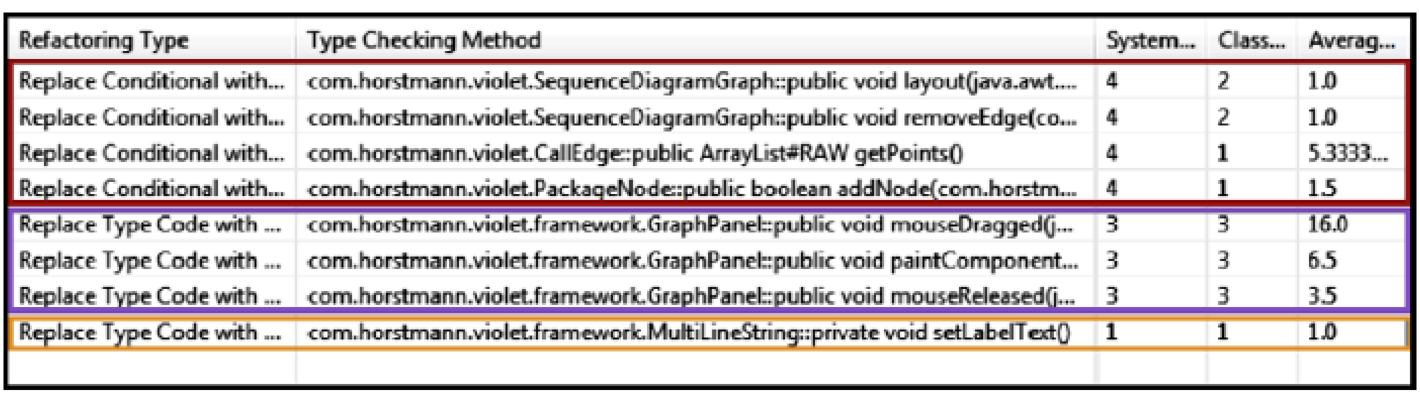
4.2.3. Ταξινόμηση Προτεινόμενων Αναδομήσεων

Η μεθοδολογία που υλοποιείται στο JDeodorant σχετικά με την ανίχνευση των προτεινόμενων αναδομήσεων, εφαρμόζει μια είδους ταξινόμηση των αποτελεσμάτων που προκύπτουν προτού εμφανιστούν στο χρήστη. Συγκεκριμένα, οι προτάσεις αναδόμησης ομαδοποιούνται βάσει της μεταβλητής τύπου για την οποία διενεργείται ο έλεγχος (περίπτωση type-checking) ή εναλλακτικά βάσει της κοινής ιεραρχίας κλάσεων (περίπτωση RTTI), αξιολογούνται και ταξινομούνται με χρήση των εξής κριτηρίων:

1. Συνολικός αριθμός εμφανίσεων υποθετικής λογικής με κοινό έλεγχο τύπου ή κοινή ιεραρχία κλάσεων σε επίπεδο συστήματος.
2. Συνολικός αριθμός εμφανίσεων υποθετικής λογικής με κοινό έλεγχο τύπου *(type check)* ή κοινή ιεραρχία κλάσεων σε επίπεδο κλάσης.
3. Μέσος αριθμός δηλώσεων σε κάθε διακλάδωση της υποθετικής δήλωσης.

Ως αποτέλεσμα στο χρήστη εμφανίζεται μια ταξινομημένη λίστα με τις προτεινόμενες αναδομήσεις (Εικόνα 15), για καθεμία από τις οποίες αναφέρεται:

- το **είδος** της **αναδόμησης**, *αναδόμηση κωδικού που υποδηλώνει τύπο με κατάσταση/στρατηγική* (*Replace Type Code with State/Strategy)* ή *αναδόμηση με πολυμορφισμό (Replace Conditional With Polymorphism)*
- η **κλάση** και η **μέθοδος** στην οποία περιέχεται
- οι **τιμές** των **κριτηρίων ταξινόμησης** με τη σειρά που αναφέρονται παραπάνω.



Refactoring Type	Type Checking Method	System...	Class...	Averag...
Replace Conditional with...	com.horstmann.violet.SequenceDiagramGraph::public void layout(java.awt....	4	2	1.0
Replace Conditional with...	com.horstmann.violet.SequenceDiagramGraph::public void removeEdge(co...	4	2	1.0
Replace Conditional with...	com.horstmann.violet.CallEdge::public ArrayList#RAW getPoints()	4	1	5.3333...
Replace Conditional with...	com.horstmann.violet.PackageNode::public boolean addNode(com.horstm...	4	1	1.5
Replace Type Code with ...	com.horstmann.violet.framework.GraphPanel::public void mouseDragged(j...	3	3	16.0
Replace Type Code with ...	com.horstmann.violet.framework.GraphPanel::public void paintComponent...	3	3	6.5
Replace Type Code with ...	com.horstmann.violet.framework.GraphPanel::public void mouseReleased(j...	3	3	3.5
Replace Type Code with ...	com.horstmann.violet.framework.MultiLineString::private void setLabelText()	1	1	1.0

Εικόνα 15 : Εμφάνιση προτεινόμενων αναδομήσεων, ομαδοποίηση και ταξινόμηση

4.2.4. Εφαρμογή Αναδόμησης

Η εφαρμογή της αναδόμησης στο JDeodorant αποτελείται από μετασχηματισμούς του παραγόμενου AST δέντρου και όχι από μετασχηματισμούς που εφαρμόζονται απευθείας στον πηγαίο κώδικα. Κατ' αυτό τον τρόπο αφενός επιτυγχάνεται η παροχή δυνατότητας προεπισκόπησης της προτεινόμενης αναδόμησης πριν την εφαρμογή της, αφετέρου δε καθίσταται εφικτή η αναίρεση μιας εφαρμοσμένης αναδόμησης.

Η εφαρμογή της αναδόμησης διαφοροποιείται ανάμεσα στους δύο τύπους υποθετικής λογικής που ανιχνεύονται από το JDeodorant. Ουσιαστική διαφορά των δύο αναδομήσεων αποτελεί το γεγονός ότι στην περίπτωση της αναδόμησης με χρήση πολυμορφισμού, η ιεραρχία των κλάσεων υπάρχει ήδη στο σύστημα, συνεπώς δε χρειάζεται να δημιουργηθεί εξ' αρχής, όπως απαιτείται στην αναδόμηση με κατάσταση/στρατηγική. Ακολούθως παρουσιάζονται ξεχωριστά οι λεπτομέρειες για την κάθε περίπτωση, με μεγαλύτερη έμφαση στην αναδόμηση με κατάσταση/στρατηγική.

4.2.5. Εφαρμογή Αναδόμησης με Κατάσταση/Στρατηγική

Κατά την αναδόμηση με χρήση των προτύπων κατάσταση/στρατηγική (*Replace Type Code with State/Strategy*) οι μετασχηματισμοί που εφαρμόζονται στο AST δέντρο στοχεύουν στην υλοποίηση των διάφορων δομικών στοιχείων των προτύπων στο αναδομούμενο σύστημα. Τα δομικά στοιχεία από τα οποία αποτελούνται τα πρότυπα κατάσταση και στρατηγική, όπως παρουσιάστηκαν στις ενότητες 2.3.1 και 2.3.2, είναι η κλάση context, η αφηρημένη κλάση κατάσταση/στρατηγική και οι συγκεκριμένες υποκλάσεις τους.

Κατά συνέπεια, οι εκτελούμενοι μετασχηματισμοί της αναδόμησης συνοψίζονται στην αξιοποίηση της μεταβλητής μέλους *τύπος (type field)* για τη δημιουργία της αφηρημένης κλάσης κατάστασης/στρατηγικής, ενώ για την απομάκρυνση των διακλαδώσεων της υποθετικής δομής από τον κώδικα εφαρμόζονται οι απλές αναδομήσεις *εξαγωγή μεθόδου (extract method)* και *μετακίνηση μεθόδου (move method)* στην κατάλληλη υποκλάση στρατηγική, η εύρεση και δημιουργία της οποίας, στηρίζεται στη σταθερά που περιλαμβάνεται στην αντίστοιχη υποθετική έκφραση. Ακολούθως περιγράφεται αναλυτικά η διαδικασία της συγκεκριμένης αναδόμησης, ενώ η Εικόνα 16 απεικονίζει την εφαρμογή της σε ένα συγκεκριμένο παράδειγμα (1).

Κατά την εφαρμογή της συγκεκριμένης αναδόμησης το ρόλο της κλάσης context διαδραματίζει η κλάση που φιλοξενεί την υποθετική λογική καθώς και τη μεταβλητή μέλος *τύπο (type field)*. Αρχικά δημιουργείται η ιεραρχία της αφηρημένης κλάσης κατάσταση/στρατηγική και των συγκεκριμένων υποκλάσεων της, το πλήθος των οποίων είναι ανάλογο του πλήθους των διαφορετικών διακλαδώσεων της υποθετικής δήλωσης, ενώ ως προεπιλεγμένα ονόματα για την ονοματοθεσία τους χρησιμοποιούνται τα ονόματα της μεταβλητής τύπου και των σταθερών με τις οποίες συγκρίνετε αντίστοιχα.

Ακολούθως, προκειμένου να επιτευχθεί η ενθυλάκωση της διαφορετικής συμπεριφοράς δημιουργείται στην αφηρημένη κλάση κατάσταση μια αφηρημένη μέθοδος, η οποία υπερφορτώνεται από τις συγκεκριμένες υποκλάσεις της, και της οποίας η υπογραφή διαφέρει από την πρωτότυπη μέθοδο υποθετικής λογικής. Συγκεκριμένα, παρόλο που το προεπιλεγμένο όνομα της νέας μεθόδου καθορίζεται ίδιο με το όνομα της πρωτότυπης, ο τύπος επιστροφής και η λίστα των παραμέτρων ενδέχεται να διαφέρουν, ανάλογα με τον κώδικα που περιλαμβάνεται στις διακλαδώσεις. Αναφορικά με τον τύπο επιστροφής, αν οι διακλαδώσεις δεν περιλαμβάνουν ούτε κάποια return εντολή ούτε κάποια εντολή εκχώρησης (assignment) σε μεταβλητή/παράμετρο της μεθόδου, ο τύπος επιστροφής ορίζεται ως void. Στην περίπτωση ύπαρξης εντολών return, ο τύπος της αφηρημένης μεθόδου, είναι ίδιος με τον τύπο επιστροφής της μεθόδου της κλάσης context, ενώ στην περίπτωση εντολών εκχώρησης καθορίζεται από τον τύπο της εκχωρούμενης μεταβλητής. Οι παράμετροι της αφηρημένης μεθόδου αποτελούνται από το σύνολο τοπικών μεταβλητών και παραμέτρων που χρησιμοποιούνται στο τμήμα υποθετικής λογικής, ενώ υπό συνθήκη προστίθεται σε αυτές και η κλάση Context, εφόσον στη δήλωση if/switch περιλαμβάνονται εντολές εκχώρησης ή προσπέλασης των πεδίων ή των μεθόδων της κλάσης Context ή των υπερκλάσεων της. Τέλος, στη νεοδημιουργηθείσα κλάση κατάσταση προστίθεται μια αφηρημένη getter μέθοδος που επιστρέφει τη μεταβλητή μέλος *τύπο*.

Εν συνεχεία, υπερφορτώνεται στις υποκλάσεις της αφηρημένης κλάσης κατάστασης/στρατηγικής τόσο η αφηρημένη μέθοδος ενθυλάκωσης της διαφορετικής συμπεριφοράς, όσο και η μέθοδος getter, ώστε να επιστρέφει την σταθερά που αντιστοιχεί στην εκάστοτε υποκλάση. Ο κώδικας που προστίθεται στην υπερφόρτωση της μεθόδου, αντιγράφεται από την αντίστοιχη διακλάδωση, και τροποποιείται κατάλληλα ώστε ενδεχόμενες εκχωρήσεις ή προσπελάσεις πεδίων της κλάσης context ή των υπερκλάσεων της, να αντικατασταθούν από τις κλήσεις των αντίστοιχων getter/setter μεθόδων. Αντίστοιχα, κατάλληλες τροποποιήσεις υφίσταται και η κλάση context προκειμένου να αποφευχθούν πιθανά σφάλματα μεταγλώττισης. Συγκεκριμένα, τροποποιείται σε δημόσια η εμβέλεια πιθανών ιδιωτικών μεθόδων που καλούνται από τον κώδικα του υποθετικού τμήματος και δημιουργούνται οι κατάλληλες getter και setter μέθοδοι των πεδίων, οι οποίες απουσιάζουν.

```java
public class GraphPanel extends JPanel {
   private Graph graph;
   private ToolBar toolBar;
   private double zoom;
   private Point2D mouseDownPoint;
   private int dragMode;

   private static final int DRAG_NONE = 0;
   private static final int DRAG_MOVE = 1;
   private static final int DRAG_RUBBERBAND = 2;
   private static final int DRAG_LASSO = 3;
   private static final int CONNECT_THRESHOLD = 8;

   public void mouseReleased(MouseEvent event) {
      Point2D mousePoint = new Point2D.Double(
         event.getX()/zoom, event.getY()/zoom);
      Object tool = toolBar.getSelectedTool();

      if(dragMode == DRAG_RUBBERBAND) {
         Edge prototype = (Edge)tool;
         Edge newEdge = (Edge)prototype.clone();
         if(mousePoint.distance(mouseDownPoint) >
               CONNECT_THRESHOLD
               && graph.connect(newEdge,
               mouseDownPoint, mousePoint)) {
            setModified(true);
            setSelectedItem(newEdge);
         }
      }
      else if(dragMode == DRAG_MOVE) {
         graph.layout();
         setModified(true);
      }

      dragMode = DRAG_NONE;
      revalidate();
      repaint();
   }
}
```

(a) original code

```java
public class GraphPanel extends JPanel {
   private Graph graph;
   private ToolBar toolBar;
   private double zoom;
   private Point2D mouseDownPoint;
   private DragMode dragMode = new DragNone();

   public static final int DRAG_NONE = 0;
   public static final int DRAG_MOVE = 1;
   public static final int DRAG_RUBBERBAND = 2;
   public static final int DRAG_LASSO = 3;
   private static final int CONNECT_THRESHOLD = 8;

   public void mouseReleased(MouseEvent event) {
      Point2D mousePoint = new Point2D.Double(
         event.getX()/zoom, event.getY()/zoom);
      Object tool = toolBar.getSelectedTool();

      dragMode.mouseReleased(tool,mousePoint,this);

      setDragMode(DRAG_NONE);
      revalidate();
      repaint();
   }
}
```

Original class playing the role of Context

```java
public abstract class DragMode {
   public abstract int getDragMode();

   public abstract void mouseReleased(Object tool,
      Point2D mousePoint, GraphPanel graphPanel);
}
```

Abstract class playing the role of State

```java
public class DragMove extends DragMode {
   public int getDragMode() {
      return GraphPanel.DRAG_MOVE;
   }

   public void mouseReleased(Object tool,
      Point2D mousePoint, GraphPanel graphPanel) {

      graphPanel.getGraph().layout();
      graphPanel.setModified(true);
   }
}
```

Concrete State subclass representing *DRAG_MOVE*

(b) refactored code

Εικόνα 16 : Εφαρμογή της αναδόμησης "Replace Type code with State / Strategy"

Τέλος, μετασχηματισμούς υφίσταται και η πρωτότυπη κλάση context, προκειμένου να διατηρηθεί τόσο η αρχική της συμπεριφορά όσο και η δημόσια διεπαφή της. Συγκεκριμένα, το τμήμα κώδικα της υποθετικής λογικής αντικαθίσταται από την πολυμορφική κλήση της μεθόδου της κλάσης κατάσταση/στρατηγική, ενώ ο τύπος της μεταβλητής μέλους *τύπου (type field)* μεταβάλλεται ώστε να αποτελεί αναφορά της αφημένης κλάσης κατάσταση/στρατηγική, και ενημερώνονται κατάλληλα οι αντίστοιχες setter και getter μέθοδοι. Η μέθοδος setter τροποποιείται ώστε να εκχωρεί το κατάλληλο στιγμιότυπο της υποκλάσης κατάσταση/στρατηγική, ενώ ο κώδικας στο εσωτερικό της getter μεθόδου αντικαθίσταται από την πολυμορφική κλήση της getter μεθόδου της κλάσης κατάσταση/στρατηγική. Επιπλέον, τροποποίηση υφίστανται και οι ενδεχόμενες αναφορές στο πεδίο *τύπο* στο εσωτερικό της κλάσης context, οι οποίες αντικαθίστανται από κλήσεις της αντίστοιχης setter και getter μεθόδου.

4.2.6. Εφαρμογή Αναδόμησης με Χρήση Πολυμορφισμού

Στην περίπτωση που το τμήμα της υποθετικής λογικής εκτελεί RTTI έλεγχο, το JDeodorant παρέχει τη δυνατότητα εφαρμογής της αναδόμησης με χρήση πολυμορφισμού (*Replace Conditional with Polymorphism*). Κατά την εφαρμογή της αναδόμησης αυτής, αρχικά δημιουργείται στην υπερτάξη της ιεραρχίας μια μέθοδος, η οποία έχει την ίδια υπογραφή με τη μέθοδο που περιέχει την υποθετική λογική στην πρωτότυπη κλάση και κάθε διακλάδωση του τμήματος της υποθετικής λογικής εξάγεται σε μια νέα μέθοδο. Οι νέες μέθοδοι τοποθετούνται στις κλάσεις που ορίζονται στις δήλωσεις καλουπώματος (*downcasting)* του αντικειμένου που περιλαμβάνονται στην context. Τέλος, η υποθετική λογική αντικαθίσταται από την πολυμορφική κλήση της αφηρημένης μεθόδου μέσω της αναφοράς της (Εικόνα 17).

4.3. Παρατηρήσεις Σχετικά με το JDeodorant

Η προσέγγιση του JDeodorant περί της αναδόμησης περιέχει μια γενίκευση ως προς την εξέταση των προτύπων σχεδίασης κατάσταση και στρατηγική, σύμφωνα με την οποία τα δύο αυτά συμπεριφορικά πρότυπα, αντιμετωπίζονται από κοινού σαν να πρόκειται για το ίδιο πρότυπο. Την προσέγγιση αυτή ακολουθεί και ο Fowler στο βιβλίο του περί της αναδόμησης, βασιζόμενος στο γεγονός ότι τα δύο πρότυπα είναι πανομοιότυπα στη δομή.

Ωστόσο, σύμφωνα με τη σύγκριση των προτύπων όπως παρουσιάστηκε στην ενότητα 2.2.3, τα δύο πρότυπα, παρόλο που διαθέτουν όμοια δομή παρουσιάζουν σημαντικές διαφορές, οι οποίες δε θα πρέπει να αγνοηθούν. Άλλωστε σύμφωνα με το βιβλίο του Kerievsky, η αναδόμηση με το πρότυπο κατάσταση διαχωρίζεται από την αναδόμηση προς τη στρατηγική, κυρίως λόγω του διαφορετικού σκοπού των δύο προτύπων. Η διαφορά του σκοπού των δύο προτύπων, συνεπάγεται διαφοροποίηση κυρίως στα κριτήρια ανίχνευσης των σημείων απουσίας

```java
public class CallEdge extends SegmentedLineEdge {
   public ArrayList getPoints() {
      ArrayList a = new ArrayList();
      Node n = getEnd();
      Rectangle2D start = getStart().getBounds();
      Rectangle2D end = n.getBounds();

      if (n instanceof CallNode &&
            ((CallNode)n).getImplicitParameter() ==
            ((CallNode)getStart()).
               getImplicitParameter()) {

         Point2D p = new Point2D.Double(
            start.getMaxX(),
            end.getY() - CallNode.CALL_YGAP / 2);
         Point2D q = new Point2D.Double(
            end.getMaxX(), end.getY());
         Point2D s = new Point2D.Double(
            q.getX() + end.getWidth(), q.getY());
         Point2D r = new Point2D.Double(
            s.getX(), p.getY());
         a.add(p);
         a.add(r);
         a.add(s);
         a.add(q);
      }
      else if (n instanceof PointNode) {
         a.add(new Point2D.Double(
            start.getMaxX(), start.getY()));
         a.add(new Point2D.Double(
            end.getX(), start.getY()));
      }
      else {
         Direction d = new Direction(
            start.getX() - end.getX(), 0);
         Point2D endPoint =
            getEnd().getConnectionPoint(d);
         if (start.getCenterX() < endPoint.getX())
            a.add(new Point2D.Double(
               start.getMaxX(), endPoint.getY()));
         else
            a.add(new Point2D.Double(
               start.getX(), endPoint.getY()));
         a.add(endPoint);
      }
      return a;
   }
}
```

(a) original code

```java
public class CallEdge extends SegmentedLineEdge {
   public ArrayList getPoints() {
      ArrayList a = new ArrayList();
      Node n = getEnd();
      Rectangle2D start = getStart().getBounds();
      Rectangle2D end = n.getBounds();

      n.getPoints(a, start, end, this);
      return a;
   }
}
```

Original class after the replacement of conditional code with polymorphic method invocation

```java
public abstract class AbstractNode implements Node {
   public void getPoints(ArrayList a,
      Rectangle2D start, Rectangle2D end,
      CallEdge callEdge) {
      Direction d = new Direction(
         start.getX() - end.getX(), 0);
      Point2D endPoint =
         callEdge.getEnd().getConnectionPoint(d);
      if (start.getCenterX() < endPoint.getX())
         a.add(new Point2D.Double(
            start.getMaxX(), endPoint.getY()));
      else
         a.add(new Point2D.Double(
            start.getX(), endPoint.getY()));
      a.add(endPoint);
   }
}
```

Existing class **AbstractNode** providing the default implementation

```java
public class CallNode extends RectangularNode {
   public void getPoints(ArrayList a,
      Rectangle2D start, Rectangle2D end,
      CallEdge callEdge) {
      if (getImplicitParameter() ==
         ((CallNode)callEdge.getStart())
            .getImplicitParameter()) {
         Point2D p = new Point2D.Double(
            start.getMaxX(),
            end.getY() - CallNode.CALL_YGAP / 2);
         Point2D q = new Point2D.Double(
            end.getMaxX(), end.getY());
         Point2D s = new Point2D.Double(
            q.getX() + end.getWidth(), q.getY());
         Point2D r = new Point2D.Double(
            s.getX(), p.getY());
         a.add(p);
         a.add(r);
         a.add(s);
         a.add(q);
      }
   }
}
```

Existing class **CallNode** providing the functionality of the first branch

(b) refactored code

Εικόνα 17 : Εφαρμογή της αναδόμησης "Replace Conditional with Polymorphism"

τους και σε πολύ μικρότερο βαθμό διαφοροποίηση στην εφαρμογή της αναδόμησης, εφόσον η δομή τους είναι όμοια.

Κυριότερη διαφοροποίηση των δύο προτύπων αποτελεί το γεγονός ότι στο πρότυπο κατάσταση, η κλάση context διαθέτει μια μεταβλητή μέλος *κατάσταση*, η οποία λαμβάνει τιμές από ένα σύνολο *σταθερών πεδίων (public static final)* που χρησιμοποιούνται για να μοντελοποιήσουν την εκάστοτε κατάσταση της κλάσης context. Για την εφαρμογή της αναδόμησης προς τη στρατηγική ωστόσο η ύπαρξη *σταθερών πεδίων* παρότι ενδέχεται να υφίσταται, δεν αποτελεί απαραίτητη προϋπόθεση. Η προσέγγιση του JDeodorant αναφορικά με την ανίχνευση των προτεινόμενων αναδομήσεων στηρίζεται πρωτίστως στην ανίχνευση σταθερών πεδίων στις υποθετικές εκφράσεις, έχοντας ως αποτέλεσμα την απώλεια περιπτώσεων όπου δε γίνεται χρήση σταθερών, αλλά ενδείκνυται η χρήση της στρατηγικής. Παρομοίως, απαραίτητη προϋπόθεση για την ανίχνευση των προτεινόμενων αναδομήσεων αποτελεί η ύπαρξη του τελεστή ισότητας στις συνθήκες των υποθετικών δηλώσεων, προϋπόθεση που, ενώ είναι υποχρεωτικό να ισχύει για την αναδόμηση προς την κατάσταση, δεν κρίνεται απαραίτητη στην περίπτωση της αναδόμησης προς τη στρατηγική.

Σημαντική ακόμα παρατήρηση για την ανίχνευση της ανάγκης χρήσης του προτύπου στρατηγική αποτελεί το γεγονός ότι, ο καθορισμός της στρατηγικής της κλάσης context πραγματοποιείται από μια διαφορετική κλάση πελάτη, η οποία, προκειμένου να μπορεί να επιλέξει την κατάλληλη, γνωρίζει την ύπαρξη των διαφορετικών στρατηγικών. Αντίθετα, η αλλαγή της κατάστασης συμβαίνει αδιαφανώς για τον πελάτη, ο οποίος έχει πλήρη άγνοια των διαφορετικών καταστάσεων ανάμεσα στις οποίες μπορεί να μεταβεί η κλάση context. Το γεγονός αυτό αντανακλάται στα κριτήρια ανίχνευσης της απουσίας των προτύπων, ελέγχοντας την εμβέλεια των μεταβλητών που αναμένονται στις υποθετικές εκφράσεις. Παρόλο που η υπάρχουσα μεθοδολογία, εφόσον δε αντιμετωπίζει ξεχωριστά τα δύο πρότυπα και την ανάγκη αναδόμησης προς αυτά, θεωρεί εξίσου πιθανή την ανίχνευση τοπικής μεταβλητής, παραμέτρου και πεδίου στην υποθετική έκφραση, η ανάγκη καθορισμού της στρατηγικής από τον πελάτη, αποκλείει την πιθανότητα εμφάνισης τοπικής μεταβλητής στις υποθετικές εκφράσεις. Η εμφάνιση παραμέτρων θεωρείται ως η πιο πιθανή, καθώς οι τιμές τους καθορίζονται κατά την κλήση της μεθόδου. Εξίσου πιθανή θεωρείται και η εμφάνιση πεδίων στις υποθετικές εκφράσεις δεδομένου ότι οι τιμές τους μπορούν να προσδιοριστούν εκτός της context, είτε μέσω του κατασκευαστή, είτε μέσω των αντίστοιχων setter μεθόδων.

Τέλος, η προσέγγιση του JDeodorant βασίζεται αποκλειστικά και μόνο στην εξέταση των υποθετικών εκφράσεων προκειμένου να αποφανθεί για την ανάγκη χρήσης των προτύπων κατάσταση και στρατηγική. Ωστόσο, σύμφωνα με την ανάλυση που προηγήθηκε στην ενότητα 4.1, σημασία για τον χαρακτηρισμό ενός τμήματος υποθετικής λογικής ως υποψήφιο για αναδόμηση με κάποιο πρότυπο, έχει και η εξέταση του πηγαίου κώδικα στο σώμα των διακλαδώσεων, για την κατά το μεγαλύτερο δυνατό βαθμό εξαγωγή συμπερασμάτων αναφορικά με το σκοπό που επιτελούν, και την αντιστοίχιση στο κατάλληλο πρότυπο σχεδίασης. Παρόλο που η ανάλυση αυτή καθίσταται ιδιαίτερα απαιτητική και ενδεχομένως να μην είναι το ίδιο αποδοτική σε όλες τις περιπτώσεις, ορισμένες φορές μπορεί να συνεισφέρει σημαντικά στον αποκλεισμό ή τον χαρακτηρισμό ενός τμήματος κώδικα ως προβληματικό.

Στο κεφάλαιο που ακολουθεί παρουσιάζεται η μεθοδολογία που αναπτύχθηκε στα πλαίσια της εργασίας αυτής σχετικά με την αναδόμηση προς τη στρατηγική, η οποία αποσκοπεί στην επίτευξη της διαφοροποίησης της από την αναδόμηση με την κατάσταση, με απώτερο στόχο την ποιοτική βελτίωση των αποτελεσμάτων της ανίχνευσης.

Κεφάλαιο 5

Προτεινόμενη Μεθοδολογία

Στόχος της μεθοδολογίας που αναπτύχθηκε είναι η αυτόματη ανίχνευση υποψήφιων περιπτώσεων αναδόμησης προς τη στρατηγική, εστιάζοντας στις περιπτώσεις όπου η επιλογή του προς εκτέλεση αλγορίθμου συντελείται εκτός των ορίων της κλάσης που περιλαμβάνει την υποθετική λογική (context), στα πλαίσια της κλάσης πελάτη (client). Επιπλέον, η προτεινόμενη προσέγγιση αφόρα και την αυτόματη ανασύνταξη του πηγαίου κώδικα, ώστε να ακολουθεί τη δομή του προτύπου.

Για την επίτευξη των ανωτέρω στόχων η ανάπτυξη της προτεινόμενης μεθοδολογίας βασίσθηκε ως ένα βαθμό στην μεθοδολογία που έχει υλοποιηθεί στο JDeodorant, η οποία παρουσιάστηκε αναλυτικά στο προηγούμενο κεφάλαιο. Ωστόσο, παρά τις ομοιότητες που πηγάζουν άμεσα από το γεγονός αυτό, η τεχνική που αναπτύχθηκε παρουσιάζει αρκετές σημαντικές διαφορές.

Η ανίχνευση των περιπτώσεων απουσίας στρατηγικής εκτελείται με τη χρήση ***στατικής ανάλυσης (static analysis)*** του πηγαίου κώδικα προκειμένου να διαπιστωθεί η ομοιότητα με συγκεκριμένες δομές του κώδικα και η ικανοποιησιμότητα ορισμένων προϋποθέσεων που θα πρέπει να πληρούνται προκειμένου μια υποθετική δήλωση να προταθεί ως υποψήφια για αναδόμηση με στρατηγική. Οι δομές αυτές και οι προϋποθέσεις ορίζονται ως ένα σύνολο κριτήριων ανίχνευσης. Η κυριότερη διαφοροποίηση της προτεινόμενης μεθοδολογίας ως προς την ανίχνευση έγκειται στα κριτήρια που επιλέχθηκαν για τον εντοπισμό των προτεινόμενων αναδομήσεων. Επιπλέον διαφοροποίηση της τεχνικής που προτείνεται αποτελεί η χρήση των γράφων CFG και PDG ως μορφές στατικής αναπαράστασης του πηγαίου κώδικα συμπληρωματικά του δέντρου AST.

Αναφορικά με την εφαρμογή της αναδόμησης, σημαντική διαφοροποίηση της προτεινόμενης προσέγγισης και ταυτόχρονα μια από τις κυριότερες συνεισφορές της αποτελεί η επίτευξη πλήρους αντικατάστασης υποθετικής λογικής. Για τον σκοπό αυτό κατά την εφαρμογή της αναδόμησης, σε ορισμένες από τις προτάσεις στις οποίες η δομή των δηλώσεων if και switch το επιτρέπει, οι πληροφορίες που προκύπτουν από την εκτέλεση στατικής ανάλυσης συνδυάζονται με την **μεμονωμένη εκτέλεση** των συνθηκών των υποθετικών δηλώσεων, εντούτοις χωρίς να εκτελείται ολόκληρο το υπό εξέταση πρόγραμμα λογισμικού. Η εκτέλεση αυτή διενεργείται προκειμένου να εντοπιστούν τα συγκεκριμένα στιγμιότυπα κλάσεων στρατηγικής που θα πρέπει να δημιουργούνται στις κλάσεις πελάτες μόνο εφόσον στις περιπτώσεις αυτές καθίσταται εφικτή η πλήρης υποκατάσταση της υποθετικής λογικής. Επιπλέον περαιτέρω τροποποιήσεις μικρότερου βαθμού υπέστη ο τρόπος εφαρμογής της αναδόμησης μερικής αντικατάστασης των υποθετικών δηλώσεων, προκειμένου να προσαρμοστεί και να μπορεί να εκτελεστεί με βάση τα νέα δεδομένα.

Εν συνεχεία παρουσιάζονται αναλυτικά οι διάφοροι τύποι κριτηρίων ανίχνευσης και επεξηγούνται συνοπτικά οι λόγοι, βάσει των οποίων επιλέχθηκαν τα συγκεκριμένα κριτήρια. Κατόπιν παρουσιάζεται με τη μορφή ψευδοκώδικα ο αλγόριθμος με τον οποίο τα κριτήρια αυτά εφαρμόζονται στον κώδικα, προκειμένου να εντοπιστούν οι προτεινόμενες αναδομήσεις, ενώ ακολουθούν η περιγραφή του τρόπου αυτοματοποιημένης εφαρμογής της αναδόμησης καθώς και μια συνοπτική παρουσίαση της υλοποίησης της μεθοδολογίας και των περιορισμών της. Το κεφάλαιο ολοκληρώνεται με τη συγκεντρωτική αντιπαραβολή των δύο μεθοδολογιών.

5.1. Επισκόπηση Κριτηρίων Ανίχνευσης

Η ανίχνευση της απουσίας της στρατηγικής και στις δύο μεθοδολογίες ανάγεται στο πρόβλημα ανίχνευσης υποθετικών δομών στον πηγαίο κώδικα, για τις οποίες πληρείται ένα σύνολο καθορισμένων κριτηρίων.

Απαραίτητη προϋπόθεση για την ορθή εφαρμογή του προτύπου στρατηγική, η οποία ελέγχεται από το σύνολο κριτηρίων, αποτελεί ο έλεγχος του εκτελούμενου αλγορίθμου από τον πελάτη. Συνεπώς, απαιτείται η ανίχνευση όλων των περιπτώσεων υποθετικής λογικής στις οποίες η επιλογή της εκτελούμενης διακλάδωσης επιτελείται εκτός της κλάσης context. Το γεγονός αυτό επιτυγχάνεται στην περίπτωση υποθετικής λογικής που βασίζεται σε παραμέτρους δηλωμένες στα πλαίσια δημόσιων ή προστατευόμενων μεθόδων, ή αντίστοιχα σε μεταβλητές μέλους κλάσης. Επιπλέον, εφόσον η εκτελούμενη στρατηγική απευθύνεται σε ξεχωριστά στιγμιότυπα της κλάσης context στα πλαίσια της οποίας ορίζεται, οι υποθετικές δομές που ορίζονται στα πλαίσια στατικών ή υπερφορτωμένων μεθόδων απορρίπτονται από τα κριτήρια ανίχνευσης. Τέλος, τα κριτήρια ανίχνευσης ορίζουν το ελάχιστο επιτρεπτό μέγεθος υποψήφιων υποθετικών δομών προς αναδόμηση, ούτως ώστε η εφαρμογή της να αποτελεί πραγματική βελτίωση του παραγόμενου σχεδίου κώδικα.

Εκτός των βασικών κριτηρίων, που αποσκοπούν αφενός στον εντοπισμό υποθετικών δηλώσεων για τις οποίες ενδείκνυται η αναδόμηση προς τη στρατηγική και αφετέρου στην απόρριψη περιπτώσεων όπου καταλληλότερος θα ήταν κάποιος άλλος τύπος αναδόμησης με πρότυπα σχεδίασης, η προτεινόμενη μεθοδολογία ορίζει ένα δεύτερο σύνολο κριτηρίων, τα οποία ελέγχονται σε επόμενο στάδιο, αφότου διαπιστωθεί η ένταξη της υποθετικής δήλωσης στο σύνολο των υποψηφίων. Τα κριτήρια δεύτερου επιπέδου δρουν συμπληρωματικά για την βαθμονόμηση των υποψήφιων προτάσεων αναδόμησης με βάρη, προκειμένου κατά την εμφάνιση τους στο χρήστη να ταξινομηθούν ανάλογα με την εκτιμώμενη εγκυρότητα τους ως πραγματικές ευκαιρίες αναδόμησης.

Τα ανωτέρω κριτήρια κατατάσσονται σε διαφορετικές κατηγορίες ανάλογα με το πεδίο εφαρμογής του απαιτούμενου ελέγχου καθώς και τα δομικά στοιχεία του κώδικα τα οποία εξετάζονται. Ως προς τις οντότητες των οποίων η δομή ελέγχεται, η παρούσα μεθοδολογία χρησιμοποιεί τους εξής διαφορετικούς τύπους κριτηρίων:

- κριτήρια αναφορικά με τις μεθόδους που φιλοξενούν τις υποθετικές δομές (εφεξής αναφέρονται ως **κριτήρια σε επίπεδο μεθόδου**)
- κριτήρια αναφορικά με τις συνθήκες των υποθετικών δομών (εφεξής αναφέρονται ως **κριτήρια σε επίπεδο συνθήκης**)
- κριτήρια αναφορικά με το τμήμα του κώδικα στο σώμα των διακλαδώσεων των υποθετικών δομών (εφεξής αναφέρονται ως **κριτήρια σε επίπεδο διακλάδωσης**).

Εναλλακτικά τα κριτήρια ανίχνευσης κατηγοριοποιούνται ανάλογα με την οντότητα του κώδικα στα πλαίσια της οποίας εκτελείται ο απαραίτητος έλεγχος. Συγκεκριμένα με βάση το πεδίο εφαρμογής του ελέγχου ορίζονται οι εξής κατηγορίες κριτηρίων:

- **κριτήρια στα πλαίσια μεθόδου**: τα κριτήρια για τον έλεγχο της ικανοποιησιμότητας των οποίων απαιτείται ανάλυση μόνο του πηγαίου κώδικα της μεθόδου που φιλοξενεί την υποθετική λογική (intraprocedural analysis).
- **κριτήρια στα πλαίσια κλάσης**: τα κριτήρια, ο έλεγχος της ικανοποιησιμότητας των οποίων, απαιτεί η ανάλυση να επεκταθεί και σε άλλες μεθόδους της ίδια κλάσης εκτός της μεθόδου με την υποθετική λογική, ακολουθώντας ανάστροφα την ροή του ελέγχου (backward flow analysis) μέσω της ακολουθίας των κλήσεων μεθόδων (method calls) (intraclass - interprocedural analysis).
- **κριτήρια στα πλαίσια συστήματος**: τα κριτήρια, ο έλεγχος της ικανοποιησιμότητας των οποίων, απαιτεί η ανάλυση να επεκταθεί και σε άλλες μεθόδους ξεχωριστών κλάσεων, ακολουθώντας ανάστροφα την ροή του ελέγχου (backward flow analysis) μέσω της ακολουθίας των κλήσεων της μεθόδου με την υποθετική λογική εκτός της κλάσης context (interclass – interprocedural analysis).

Στον Πίνακας 8 παρουσιάζεται η κατάταξη των κριτηρίων ανίχνευσης στις ανωτέρω κατηγορίες, τα οποία επεξηγούνται αναλυτικά στην επόμενη ενότητα.

Εξεταζόμενη Δομή / Πεδίο εφαρμογής ελέγχου	Μέθοδος	Συνθήκη	Διακλάδωση
Μέθοδος	Απόρριψη υποθετικών δομών στατικών μεθόδων Απόρριψη υποθετικών δομών υπερφορτωμένων μεθόδων	Απόρριψη συνθηκών ελέγχου τύπου κατά το χρόνο εκτέλεσης (τελεστής instanceof) Απόρριψη συνθηκών με τοπικές μεταβλητές Απόρριψη συνθηκών με παραμέτρους που τροποποιούνται εντός μεθόδου	Ανίχνευση υποθετικών δομών με εκχώρηση σε πεδίο Ανίχνευση υποθετικών δομών με εντολή επιστροφής πεδίου ή κλήσης μεθόδου Ανίχνευση υποθετικών δομών με όμοιες υπογραφές μεθόδων έπειτα από εξαγωγή διακλαδώσεων σε ξεχωριστές μεθόδους
Κλάση	Απόρριψη υποθετικών δομών ιδιωτικών μεθόδων με προσδιορισμό παραμέτρων και πεδίων εντός κλάσης	Απόρριψη συνθηκών με μεταβλητές μέλη που λαμβάνουν τιμή εντός context	Απόρριψη διακλαδώσεων με μέσο αριθμό δηλώσεων ίσο με ένα, εκτός κλήσεων μεθόδων κλάσης Απόρριψη δηλώσεων if/switch με δύο διακλαδώσεις, η μία εκ των οποίων διαθέτει μόνο μια εντολή (εκτός κλήσης μεθόδου κλάσης)
Σύστημα	Ανίχνευση κλήσεων μεθόδων με υποθετικές δηλώσεις, οι παράμετροι των οποίων έχουν γνωστές τιμές κατά το χρόνο μεταγλώττισης		

Πίνακας 8: Κατάταξη κριτηρίων ανίχνευσης σε κατηγορίες

5.2. Ανάλυση Κριτηρίων Ανίχνευσης

Για την ανίχνευση της απουσίας της στρατηγικής εξετάζονται όλες οι υποθετικές δομές που διαθέτουν κατ’ ελάχιστό δύο διακλαδώσεις. Ωστόσο προτού αναλυθεί η δομή των δηλώσεων if και switch, ελέγχονται τα χαρακτηριστικά ορισμένων εκ των οντοτήτων που συνθέτουν την υπογραφή των μεθόδων στις οποίες εμπεριέχονται, και συγκεκριμένα η ορατότητα, η εμβέλεια και το ενδεχόμενο υπερφόρτωσης μεθόδου υπερκλάσης.

Απόρριψη υποθετικών δομών ιδιωτικών μεθόδων

Αναφορικά με την ορατότητα, παρόλο που οι υποθετικές δομές που εμπεριέχονται στο σώμα *δημόσιων (public)* ή *προστατευόμενων (protected)* μεθόδων εξετάζονται ως υποψήφιες χωρίς περαιτέρω επεξεργασία του τρόπου κλήσης τους, οι υποθετικές δομές που δηλώνονται στα πλαίσια *ιδιωτικών μεθόδων (private)* εν γένει απορρίπτονται. Εξαίρεση αποτελούν οι περιπτώσεις ιδιωτικών μεθόδων στις οποίες, παρά τον περιορισμό της προσβασιμότητας τους αποκλειστικά εντός των ορίων της κλάσης στην οποία δηλώνονται, ο έλεγχος εξακολουθεί να βασίζεται σε παραμέτρους που λαμβάνουν είσοδο εξωτερικά της κλάσης context. Η προϋπόθεση αυτή πηγάζει από την ανάγκη εντοπισμού των περιπτώσεων εκείνων όπου η επιλογή της στρατηγικής που πρόκειται να εκτελεστεί δεν βρίσκεται στη δικαιοδοσία της κλάσης context, αλλά καθορίζεται εξωτερικά από αυτή, σε μια ξεχωριστή κλάση *πελάτη (client).* Η ιδιωτική ορατότητα των μεθόδων που τις καθιστά προσβάσιμες μόνο στα πλαίσια της κλάσης context αντίκειται με την προϋπόθεση αυτή και συνεπώς υπό το πρίσμα αυτό, οι υποθετικές τους δηλώσεις θα έπρεπε να απορρίπτονται. Εντούτοις μια συνήθης πρακτική των μηχανικών λογισμικού κατά την ανάπτυξη, είναι η απομόνωση των εναλλακτικών αλγορίθμων σε ξεχωριστές μεθόδους ιδιωτικής προσπέλασης της ίδιας κλάσης, εφόσον εννοιολογικά σχετίζονται με αυτή και καλούνται εσωτερικά, προκειμένου να τις απομονώσουν από την υπόλοιπη λογική της μεθόδου και να την απλοποιήσουν. Κατά συνέπεια, η υποθετική λογική που εντάσσεται στα πλαίσια μιας ιδιωτικής μεθόδου εξακολουθεί να αποτελεί υποψήφια πρόταση αναδόμησης προς τη στρατηγική, εφόσον ισχύει μια από τις παρακάτω προϋποθέσεις:

A. **Υποθετική έκφραση με έλεγχο βάσει πεδίων:** η κλάση context δεν περιλαμβάνει εντολές εκχώρησης τιμής στα πεδία (Εικόνα 18γ). Κατ' εξαίρεση επιτρέπονται εντολές εκχώρησης στις μεταβλητές μέλους μόνο στον κατασκευαστή της κλάσης ή εναλλακτικά στα πλαίσια των setter μεθόδων τους.

B. **Υποθετική έκφραση με έλεγχο βάσει παραμέτρων**: στην περίπτωση αυτή οι παράμετροι θα πρέπει να λαμβάνουν τιμή είτε από κάποιο πεδίο που ικανοποιεί τη συνθήκη που περιγράφεται στην περίπτωση Α, είτε από παράμετρο δημόσιας μεθόδου (Εικόνα 18β). Συνεπώς, εντοπίζονται οι κλήσεις της μεθόδου και ο έλεγχος μεταφέρεται σε αυτές. Συγκεκριμένα ελέγχεται η περίπτωση που κατά την κλήση εκχωρούνται στις παραμέτρους πεδία για τα οποία ικανοποιείται η συνθήκη Α. Στην περίπτωση που οι μεταβλητές που εκχωρούνται στην κλήση αποτελούν παραμέτρους της καλούσας μεθόδου, κρίνεται απαραίτητος ο έλεγχος της εμβέλειας της, προκειμένου να ληφθεί απόφαση σχετικά με την υποψήφια πρόταση αναδόμησης. Συγκεκριμένα η υποθετική λογική εξακολουθεί να αποτελεί υποψήφια πρόταση αναδόμησης εφόσον η καλούσα μέθοδος έχει δημόσια ορατότητα. Αν ωστόσο η εμβέλεια της είναι ιδιωτική, αναζητούνται εκ νέου οι δικές της κλήσεις και εκτελείται αναδρομικά ο ίδιος έλεγχος των προϋποθέσεων A και B (Εικόνα 18 α).

Στο σημείο αυτό κρίνεται αναγκαίο να τονιστεί πώς ο ανάστροφος έλεγχος των κλήσεων μιας ιδιωτικής μεθόδου, δεν εξακολουθεί να εκτελείται αναδρομικά έως ότου εντοπιστεί μια δημόσια μέθοδος, αφενός γιατί ενδέχεται να μη βρεθεί κάποια δημόσια μέθοδος στο μονοπάτι των κλήσεων, αφετέρου γιατί ένας τόσο εξαντλητικός έλεγχος της ακολουθίας των κλήσεων των μεθόδων θα αύξανε πολύ τις απαιτήσεις σε μνήμη και τον απαιτούμενο χρόνο εκτέλεσης της διαδικασίας ανίχνευσης. Συνεπώς, κρίθηκε αναγκαίο να καθοριστεί ένα όριο αναφορικά με τον αριθμό των επιτρεπτών προς τα πίσω βημάτων στην ακολουθία των κλήσεων μεθόδου, προτού η υποθετική δήλωση μιας ιδιωτικής μεθόδου απορριφθεί. Το άνω όριο της αναζήτησης καθορίστηκε σε 2 προς τα πίσω βήματα, αριθμός που θεωρήθηκε λογικός αναφορικά με την πιθανότητα αλυσιδωτής εμφώλευσης κλήσεων μεθόδων, ενώ παράλληλα κρίθηκε αρκούντος μεγάλος ώστε να μην απορρίπτονται λανθασμένα αρκετές περιπτώσεις υποθετικής λογικής, και ταυτόχρονα αρκούντος μικρός, ώστε να μην αυξάνονται επικίνδυνα οι απαιτήσεις του υπό ανάπτυξη αυτόματου εργαλείου αναδόμησης.

Απόρριψη υποθετικών δομών στατικών μεθόδων

Κατ' αντιστοιχία με τον έλεγχο και την απόρριψη υποθετικών δηλώσεων μεθόδων ιδιωτικής εμβέλειας, απορρίπτονται από το υπό εξέταση σύνολο οι υποθετικές δηλώσεις που βρίσκονται εμφωλευμένες σε μεθόδους με *στατική εμβέλεια (static).* Η στατική εμβέλεια των μεθόδων συνεπάγεται ότι οι μεταβλητές και οι μέθοδοι που προσπελαύνονται από αυτές, πρέπει επίσης να έχουν στατική εμβέλεια. Το γεγονός αυτό θα σήμαινε ότι κατά τη φάση εκτέλεσης της αναδόμησης, προκειμένου να μην ανακύψουν σφάλματα μεταγλώττισης, η μεταβλητή μέλος *στρατηγική* θα έπρεπε να δηλωθεί με στατική εμβέλεια. Η στατική εμβέλεια μιας μεταβλητής σχετίζεται με το χρόνο δημιουργίας της, κατά τη φάση μεταγλώττισης και όχι κατά το χρόνο εκτέλεσης. Ωστόσο, η συγκεκριμένη στρατηγική με

public void showSomething(int allignment) {
.....
printMessage(allignment);
}
private void printMessage(int allignment){
......
allign(allignment);
}
private void allign(int allignment)
{
if(allignment == VERTICAL)
printVertically();
} else {
printHorizontically();
}
}
	public void printMessage(int allignment){
	......
	allign(allignment);
	......
	}
	private void allign(int allignment)
	{
	if(allignment == VERTICAL{
	printVertically();
	} else {
	printHorizontically();
	}
	}
		public class printer() {
		private int allignment;
		public static final int VERTICAL = 1;
		private void allign() {
		if(allignment == VERTICAL) {
		printVertically();
		} else {
		printHorizontically();
		}
		}
		public void setAllign(int allignment) {
		this.allignment = alignment;
		}
		//no other methods
		}

Εικόνα 18: Παραδείγματα έγκυρων περιπτώσεων ιδιωτικών μεθόδων α) υποθετική έκφραση με παράμετρο, η οποία λαμβάνει τιμή από παράμετρο δημόσιας μεθόδου σε βάθος 2 κλήσεων πίσω, β) υποθετική έκφραση με παράμετρο που λαμβάνει τιμή βάσει παραμέτρου δημόσιας μεθόδου σε βάθος 1 κλήσης πίσω, γ) υποθετική έκφραση βάσει πεδίου χωρίς εκχώρηση στην context

αυξημένη πιθανότητα ενδέχεται να μην είναι γνωστή στο χρόνο μεταγλώττισης. Επιπλέον, η δήλωση μιας μεταβλητής με στατική εμβέλεια αποσυσχετίζει την μεταβλητή από τα συγκεκριμένα στιγμιότυπα-αντικείμενα της κλάσης μέσα στην οποία ορίζεται και την συνδέει με την ίδια την κλάση, κατά τρόπο ώστε να μπορεί να κληθεί και να χρησιμοποιηθεί χωρίς την ανάγκη ύπαρξης και χρήσης κάποιου αντικειμένου. Ωστόσο, δεν είναι εφικτή η εξαγωγή συμπεράσματος για το κατά πόσο είναι επιθυμητό να οριστεί με στατική εμβέλεια η μεταβλητή μέλος στρατηγική εφόσον ενδέχεται να απαιτείται να καθοριστεί διαφορετική στρατηγική για κάθε αντικείμενο της κλάσης context και όχι ένα κοινό για όλα τα στιγμιότυπα της. Η χρήση στατικής εμβέλειας στο πεδίο της στρατηγικής θα καθιστούσε αδύνατη την εναλλαγή μεταξύ των παραλλαγών των αλγορίθμων στα διαφορετικά στιγμιότυπα της κλάσης context. Συνεπώς, μια τέτοια ενέργεια αφενός περιορίζει τη χρησιμότητα και τις δυνατότητες του προτύπου και αφετέρου ενδεχομένως αντίκειται στην *αρχή διατήρησης της συμπεριφοράς (behaviour preservation)* με την αυστηρή έννοια του όρου, η οποία επιβάλλεται να διασφαλίζεται κατά την αναδόμηση, και γι' αυτό το λόγο δεν θεωρείται θεμιτή, ούτε επιτρεπτή. Η υπόθεση αυτή ενισχύεται σύμφωνα με τις προϋποθέσεις που περιγράφονται στη διδακτορική διατριβή του O Cinnéide (71). Οι προϋποθέσεις αυτές θα πρέπει να τηρούνται προκειμένου να είναι εφικτή η εφαρμογή της αναδόμησης και ορίζουν ότι καμία μέθοδος κλάσης στρατηγικής δε θα πρέπει να έχει στατική εμβέλεια.

Απόρριψη υποθετικών δομών υπερφορτωμένων μεθόδων

Επιπροσθέτως, στο σύνολο των δηλώσεων υποθετικής λογικής που δε λαμβάνονται υπόψη κατά την ανίχνευση προτεινόμενων αναδομήσεων προστίθενται οι δηλώσεις εκείνες που περιλαμβάνονται στο σώμα μεθόδων, οι οποίες υπερφορτώνουν μεθόδους κλάσεων που δηλώνονται εκτός των ορίων του συστήματος, είτε στα πλαίσια κάποιας βιβλιοθήκης που σχετίζεται με το υπό εξέταση σύστημα, είτε στα πλαίσια του ίδιου του api (application programming interface) της γλώσσας προγραμματισμού Java (Εικόνα 19). Η λογική απόρριψης αυτών των μεθόδων σχετίζεται με το γεγονός ότι η στρατηγική παρέχει εναλλακτικούς τρόπους εκτέλεσης αλγορίθμων που αφορούν τη συμπεριφορά της κλάσης context και σχετίζονται ουσιαστικά με το υπό εξέταση σύστημα. Ωστόσο κατά την υπερφόρτωση μιας μεθόδου, ουσιαστικά παρέχεται μια εναλλακτική υλοποίηση της συμπεριφοράς της κληρονομημένης κλάσης ή διεπαφής, συνεπώς η στρατηγική θα έπρεπε να συσχετιστεί με αυτή την κλάση και όχι με την κληρονομούσα. Κατά συνέπεια, η εφαρμογή του προτύπου στην περίπτωση υπερφόρτωσης λειτουργικότητας άλλων συστημάτων, ξεφεύγει από τα όρια του υπό εξέταση συστήματος, δεν είναι εφικτή και θεωρείται λανθασμένη, γι' αυτό και οι υπερφορτωμένες μέθοδοι όταν ανιχνευθούν παύουν να αποτελούν αντικείμενα περαιτέρω εξέτασης.

```java
public String toString() {

    StringBuffer s =  new StringBuffer("Radio Interface:\n");

    if (description == LONG_DESC) {
        for (Flow f : servedFlows) {
            s.append("\t\t ");
            s.append(f.toString());
            s.append("\n");
        }
    } else {
        s.append("\t ");
        for (Flow f : servedFlows) {
            s.append(f.toString());
            s.append(" | ");
        }
        s.append("\n");
    }
    return s.toString();
}
```

Εικόνα 19 : Απόσπασμα υπερφορτωμένης μεθόδου του project JFScheduler με δήλωση if που δεν αποτελεί υποψήφια πρόταση αναδόμησης

Απόρριψη υποθετικών εκφράσεων με RTTI

Εν συνεχεία, αναφορικά με τα κριτήρια σε επίπεδο συνθήκης κρίνεται αναγκαίο να απορριφθούν όλες οι περιπτώσεις υποθετικής δομής if/else…if στις οποίες εκτελείται έλεγχος του τύπου της μεταβλητής κατά το χρόνο εκτέλεσης *(RTTI)*, καθώς σύμφωνα με την ανάλυση της ενότητας 4.1, οι περιπτώσεις αυτές είναι υποψήφιες για την εφαρμογή της αναδόμησης με πολυμορφισμό *(Replace Conditional Logic with Polymorphism).*

Απόρριψη υποθετικών εκφράσεων με έλεγχο εντός context

Επιπλέον, από το σύνολο των υποψηφίων απορρίπτονται όλες οι υποθετικές δομές στις συνθήκες των οποίων περιλαμβάνονται *τοπικές μεταβλητές (local variables)* ή *μεταβλητές μέλη (fields),* εφόσον σ' αυτές εκχωρούνται τιμές στο εσωτερικό της κλάσης context και όχι εξωτερικά από κάποια κλάση πελάτη. Υπόψη λαμβάνονται μόνο οι δομές, στις συνθήκες των οποίων, περιλαμβάνονται παράμετροι, σταθερές *(public static final πεδία),* ή πεδία, των οποίων οι τιμές καθορίζονται εκτός της κλάσης context. Ο περιορισμός του συνόλου των επιτρεπτών μεταβλητών σε παραμέτρους, σταθερές και πεδία χωρίς εκχώρηση εντός context, πηγάζει από την ανάγκη εντοπισμού αποκλειστικά των περιπτώσεων εκείνων όπου η στρατηγική καθορίζεται εξωτερικά της κλάσης *context,* στα πλαίσια της κλάσης *πελάτη (client).* Εξαιτίας αυτού του περιορισμού άλλωστε απορρίπτονται και περιπτώσεις όπου οι παράμετροι της υποθετικής έκφρασης τροποποιούνται στα πλαίσια της μεθόδου, πριν την εκτέλεση του ελέγχου (Εικόνα 20).

```
public void printMessage(int alignment,
boolean changeAllignment) {
    …
    if(changeAllignment) {
        allignment = HORIZONTAL;

        if(allignment == VERTICAL)
            printVertically();
        } else {
            printHorizontically();
        }
    }
}
```

Εικόνα 20: Παράδειγμα μη έγκυρης πρότασης αναδόμησης με έλεγχο βάσει παραμέτρου

Απόρριψη υποθετικών εκφράσεων με μικρό αριθμό εντολών

Τέλος, αναφορικά με τα κριτήρια σε επίπεδο διακλάδωσης, ένας ακόμα περιορισμός που επιβάλλεται από τη συγκεκριμένη μεθοδολογία αναφορικά με την εξέταση των υποψήφιων προς αναδόμηση υποθετικών δηλώσεων, αφορά το μέγεθος τους, εκτιμώμενο από τον υπολογισμό του *μέσου αριθμού δηλώσεων ανά διακλάδωση (average statements/branch)*. Συγκεκριμένα, ως ελάχιστος επιτρεπτός μέσος αριθμός δηλώσεων ανά διακλάδωση θεωρείται ο αριθμός 2. Η επιβολή αυτού του ορίου δεν έγινε τυχαία, καθώς σύμφωνα με τον Πίνακας 9, περίπου το 50% των υποθετικών δηλώσεων με τουλάχιστον δύο διακλαδώσεις που περιλαμβάνονται σε ένα σύστημα λογισμικού, περιέχουν στις διακλαδώσεις τους μόνο μία εντολή, ενώ το αντίστοιχο ποσοστό δηλώσεων με μέσο αριθμό εντολών μικρότερο ή ίσο του 2 ανέρχεται σε 72%, κατά μέσο όρο.

Παρά το γεγονός ότι η πλειοψηφία των υποθετικών δηλώσεων ενός συστήματος είναι αμελητέου μεγέθους, το σύνολο των δηλώσεων αυτών δε κρίνεται υποψήφιο προς αναδόμηση καθώς θεωρείται πώς η δημιουργία μιας ολόκληρης ιεραρχίας κλάσεων για την δημιουργία μεθόδων με μια ή δύο μόνο εντολές, έχει μεγαλύτερη αρνητική επίδραση στην πολυπλοκότητα του συστήματος από ότι η χρήση υποθετικής λογικής, για την επίτευξη της ίδιας λειτουργικότητας, χωρίς η αξία της αναδόμησης αυτής να μπορεί να αντισταθμίσει το αρνητικό αυτό κόστος και δη, να το υπερκαλύψει, ώστε να είναι προτιμητέα ανάμεσα στις δύο εναλλακτικές. Επιπλέον θεωρείται σχετικά απίθανο οι παραλλαγές των αλγορίθμων να αποτελούνται μόνο από μια-δύο εντολές, εκτός και αν πρόκειται για κάποιο απλό υπολογισμό, για τον οποίο δεν κρίνεται απαραίτητη η χρήση του προτύπου καθώς θα είχε μικρή επίδραση στη βελτίωση της πολυπλοκότητας του συστήματος.

Εξαίρεση στις υποθετικές δηλώσεις με μέσο αριθμό εντολών ανά διακλάδωση μικρότερο του δύο που απορρίπτονται, αποτελούν εκείνες οι οποίες περιλαμβάνουν τουλάχιστον στις μισές από τις διακλαδώσεις τους, κλήση μεθόδου και δη, μεθόδου δηλωμένης στο εσωτερικό της κλάσης context, εκτός των setter μεθόδων. Ο διαχωρισμός αυτός γίνεται προκειμένου να καλυφθεί η περίπτωση στην οποία οι μέθοδοι που καλούνται ουσιαστικά περιέχουν εναλλακτικές υλοποιήσεις του ίδιου αλγορίθμου, οι οποίες έχουν διαχωριστεί από τη λογική της υπόλοιπης μεθόδου και έχουν απομονωθεί στα πλαίσια ξεχωριστών μεθόδων, που συνήθως έχουν ιδιωτική

PROJECTS	JFScheduler	Nutch	Pamvotis	Violet	IceHockey Manager	MOEM
SLOC[7]	23452	16810	5285	19965	27113	10671
#Classes	213	206	33	224	310	37
#Methods	1640	1050	299	1430	2320	513
Συνολικός #if/switch με >=2 διακλαδώσεις	268	200	74	107	280	250
#Υποθετικών Δηλώσεων με μέσο αρ. δηλώσεων/διακλάδωση = 1	146	75	43	54	163	129
#Υποθετικών Δηλώσεων με μέσο αρ. δηλώσεων/διακλάδωση <= 2	202	126	56	76	211	168
Ποσοστό #μέσου αρ. = 1	54%	37.5%	58%	59%	58%	51.6%
Ποσοστό #μέσου αρ. <= 2	75%	63%	75%	71%	75%	67.2%

Πίνακας 9 : Εκτίμηση αριθμού υποθετικών δηλώσεων με μέσο αρ. δηλώσεων/διακλάδωση <=2 σε διάφορα συστήματα λογισμικού

προσπέλαση, και δηλώνονται στο εσωτερικό της κλάσης με την οποία σχετίζονται εννοιολογικά, όπως παραδείγματος χάριν στην περίπτωση που απεικονίζει η Εικόνα 21 α. Ωστόσο, οι υποθετικές δηλώσεις απορρίπτονται αν στις μισές τουλάχιστον διακλαδώσεις καλείται η ίδια μέθοδος με διαφορετικά ορίσματα, αν οι μέθοδοι που καλούνται αποτελούν απλά setter μέθοδο ή εναλλακτικά εάν οι μέθοδοι που περιλαμβάνονται στις διακλαδώσεις τους, αποτελούν μεθόδους δημιουργίας αντικειμένων (Εικόνα 21 β). Η περίπτωση αυτή απορρίπτεται καθώς θεωρείται υποψήφια για αναδόμηση με το πρότυπο μέθοδος εργοστάσιο (factory method) σύμφωνα με την ανάλυση της ενότητας 4.1. Για τον εντοπισμό της θα πρέπει να ισχύουν οι εξής προϋποθέσεις:

[7] SLOC (Source Lines Of Code) : Ορίζεται ως ο αριθμός των εκτελέσιμων γραμμών πηγαίου κώδικα σε ένα σύστημα λογισμικού, όπου οι γραμμές που αντιστοιχούν σε σχόλια δεν προσμετρούνται.

- Ύπαρξη κλήσεων μεθόδων της κλάσης context στις μισές τουλάχιστον διακλαδώσεις της.
- Οι καλούμενοι μέθοδοι περιλαμβάνουν μια εντολή δημιουργίας αντικειμένου, το οποίο επιστρέφουν.
- Ο τύπος του δημιουργούμενου αντικειμένου είναι όμοιος με τον τύπο της μεθόδου που περιλαμβάνει την υποθετική λογική ή υποτύπος του.

Αναφορικά με το ελάχιστο επιτρεπτό μέγεθος ανά διακλάδωση, αναπτύχθηκε ένα ακόμα κριτήριο βάσει του οποίου απορρίπτονται οι υποθετικές δηλώσεις με μέσο αριθμό εντολών ανά διακλάδωση μεγαλύτερο του 2. Οι δηλώσεις οι οποίες απορρίπτονται είναι εκείνες που αποτελούνται από ακριβώς 2 διακλαδώσεις, η μια εκ των οποίων περιλαμβάνει μια μόνο ή καμία εκτελέσιμη εντολή, που δεν αποτελεί κλήση μεθόδου της κλάσης context, με εξαίρεση τις setter μεθόδους και τις μεθόδους δημιουργίας, ενώ η άλλη διακλάδωση διαθέτει επαρκή αριθμό δηλώσεων. Η περίπτωση αυτή απορρίπτεται, θεωρώντας σχετικά απίθανο η μια εκ των δύο παραλλαγών του αλγορίθμου να αποτελείται από μια μόνο εντολή (Εικόνα 21 γ).

```
public void openFile
(boolean isWindows)
{
    …..
    …..
    if (isWindows) {
      openNotepad();
    } else {
      openJEdit();
    }

}
```

```
public Shape createShape (Dimesions
d, String type){
     Shape shape = null;
     if(type.equals("CIRCLE")){
         shape = createCircle(d);
     }
     else if(type.equals("SQUARE")){
         shape = createSquare(d);
     }
     return shape;
}
private Shape createCircle(Dimensions
d){
     return new Circle(d);
}
```

```
switch(parameter) {
   case 1: {
     System.out.println("…..");
     calculate(…..);
     doSomething();
   }
   default: {
     //exit
     break;
   }
}
```

Εικόνα 21: Παράδειγμα υποθετικών δήλωσεων if με avg.stmts/block = 1 α) έγκυρη πρόταση αναδόμησης του project JFScheduler με κλήσεις μεθόδων της κλάσης context, β) μη έγκυρη πρόταση αναδόμησης με κλήσεις μεθόδων δημιουργίας, γ) μη έγκυρη πρόταση αναδόμησης με avg.

Επιπροσθέτως, η προτεινόμενη μεθοδολογία περιλαμβάνει ορισμένα κριτήρια τα οποία χαρακτηρίζονται ως *«χαλαρά»*, υπό την έννοια ότι ο έλεγχος τους δεν αποσκοπεί στην απόρριψη υποθετικών δηλώσεων από το σύνολο των προτεινόμενων προς αναδόμηση. Αντιθέτως, τα κριτήρια αυτά χρησιμοποιούνται συμπληρωματικά στην εξαγωγή συμπερασμάτων ως προς την ενδεχόμενη ομοιότητα της δομής του κώδικα στο εσωτερικό των διακλαδώσεων με τα χαρακτηριστικά δομής που περιγράφουν, προκειμένου να αποδοθούν τα κατάλληλα βάρη στις προτάσεις αναδόμησης για την ταξινομημένη εμφάνιση τους στο χρήστη.

Ένας από τους ελέγχους που διεξάγονται για το σκοπό αυτό εξετάζει το σύνολο των παραμέτρων που προκύπτουν στις υπογραφές των μεθόδων που θα δημιουργηθούν, αν ο κώδικας κάθε διακλάδωσης εξαχθεί σε ξεχωριστή μέθοδο. Το κριτήριο αυτό αποσκοπεί στην απόδοση μεγαλύτερου βάρους σε εκείνες τις δηλώσεις, η εξαγωγή των διακλαδώσεων των οποίων σε ξεχωριστές μεθόδους έχει ως αποτέλεσμα την παραγωγή μεθόδων με κοινή υπογραφή, με βάση την υπόθεση ότι, οι παραλλαγές ενός αλγορίθμου ενδέχεται να χρειάζονται για την ορθή εκτέλεση τους το ίδιο σύνολο παραμέτρων.

Επιπλέον κριτήριο που ελέγχεται για τον ίδιο σκοπό αποτελεί η ύπαρξη εντολών επιστροφής (return) στις διακλαδώσεις της δήλωσης if/switch, όπου η επιστρεφόμενη έκφραση αποτελεί είτε κάποια μεταβλητή, είτε κλήση μεθόδου, είτε κάποια infix έκφραση και όχι απλά κάποια αληθοτιμή, αριθμητική ή λεκτική τιμή. Αντίστοιχο είναι και το κριτήριο σύμφωνα με το οποίο αποδίδεται πρόσθετο βάρος στις δηλώσεις εκείνες των οποίων κάθε διακλάδωση περιλαμβάνει εκχώρηση στην ίδια μεταβλητή μέλος της κλάσης context, είτε με χρήση του τελεστή εκχώρησης (τελεστής "="), είτε μέσω της κλήσης της setter μεθόδου της. Τα κριτήρια αυτά αναπτύχθηκαν βασισμένα στην υπόθεση ότι οι παραλλαγές ενός αλγορίθμου ενδέχεται είτε να υπολογίζουν την τιμή κάποιας μεταβλητής μέλους, είτε να αφορούν κάποιον άλλο υπολογισμό, το αποτέλεσμα του οποίου πρέπει να επιστραφεί.

Συμπερασματικά, η σημασία των ανωτέρω κριτηρίων αποδεικνύεται καλύτερα από την μελέτη διάφορων στατιστικών στοιχείων αναφορικά με τον αριθμό των υποθετικών δομών, για τις οποίες τα κριτήρια αυτά ικανοποιούνται, σε διάφορα υπό μελέτη συστήματα. Στον Πίνακας 10 παρουσιάζονται τα στατιστικά στοιχεία αναφορικά με τα κριτήρια σε επίπεδο μεθόδου.

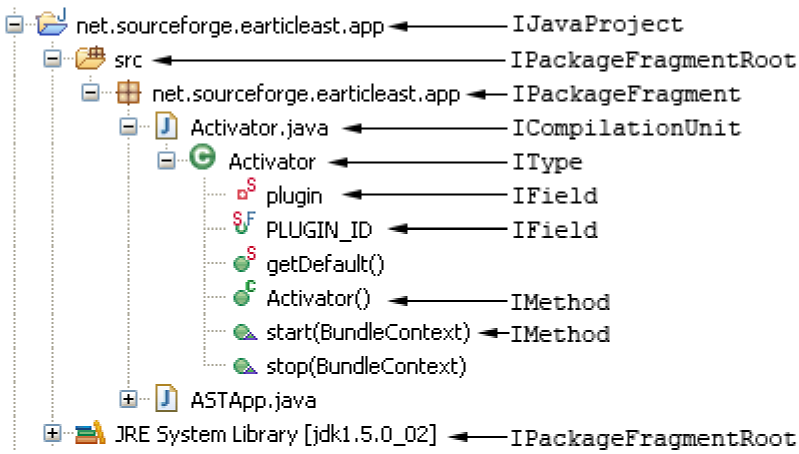
	PROJECTS	JFScheduler	IceHockeyManager	MOEM
	SLOC	23452	27113	10671
	#Κλάσεων	213	310	37
	#Μεθόδων	1640	2320	513
Κριτήρια Απόρριψης	Συνολικός #if/switch με >=2 διακλαδώσεις	268	280	250
ΚΑ1	#Υποθετικών Δηλώσεων σε Ιδιωτικές Μεθόδους	78	61	81
ΚΑ2	#Υποθετικών Δηλώσεων σε Στατικές Μεθόδους	9	35	0
ΚΑ3	#Υποθετικών Δηλώσεων σε Υπερφορτωμένες Μεθόδους	20	40	58
	(ΚΑ1 U ΚΑ2) – (ΚΑ1 ∩ ΚΑ2)	87	87	81
	(ΚΑ1 U ΚΑ2 U ΚΑ3) - (ΚΑ1 ∩ ΚΑ2 ∩ ΚΑ3)	107	127	139
	Συνολικός αρ. υποθετικών δηλώσεων υπό εξέταση	**161**	**153**	**11**

Πίνακας 10 : Στατιστικά στοιχεία εμφάνισης κριτηρίων σε επίπεδο μεθόδου σε διάφορα συστήματα λογισμικού

5.3. Αλγόριθμος Ανίχνευσης

Ο αλγόριθμος που σχεδιάστηκε για την εκτέλεση της ανίχνευσης των υποψήφιων τμημάτων κώδικα προς αναδόμηση, αναπτύχθηκε με παρόμοια λογική σύμφωνα με την υπάρχουσα υλοποίηση του JDeodorant.

Ο αλγόριθμος ανίχνευσης υιοθετεί τη λογική θεώρησης ενός συστήματος λογισμικού ως μια δενδρική απεικόνιση, με ρίζα ολόκληρο το σύστημα, ενδιάμεσους κόμβους τις κλάσεις του συστήματος και φύλλα τις μεθόδους. Η αναπαράσταση ενός συστήματος λογισμικού java σε δενδρική μορφή αναφέρεται και στο μοντέλο αναπαράστασης Java Model της βιβλιοθήκης του eclipse JDT (Εικόνα 22) (96) (94). Για την ανίχνευση των προτεινόμενων αναδομήσεων πραγματοποιείται κατά βάθος διάσχιση του δέντρου από το επίπεδο συστήματος, σε επίπεδο κλάσης και τέλος σε επίπεδο μεθόδου. Στην κλάση που μοντελοποιεί το επίπεδο μεθόδου πραγματοποιείται μόνο η συλλογή όλων των υποθετικών δηλώσεων, ενώ στην κλάση που αντιπροσωπεύει το επίπεδο κλάσης εκτελούνται όλοι οι απαραίτητοι έλεγχοι των κριτηρίων καθώς και η αποτίμηση των αποτελεσμάτων της ανίχνευσης και η ανάθεση βαρών. Τέλος, η κλάση που μοντελοποιεί το επίπεδο συστήματος αναλαμβάνει ρόλο συντονισμού της διαδικασίας ανίχνευσης και της συλλογής των αποτελεσμάτων που προκύπτουν από την ξεχωριστή εξέταση καθεμίας από τις κλάσεις του συστήματος και την εφαρμογή των κριτηρίων. Η κλάση αυτή είναι και η μοναδική που επικοινωνεί άμεσα με τις κλάσεις που μοντελοποιούν τη γραφική διεπαφή προκειμένου να αποστείλει τις προτάσεις αναδόμησης προς παρουσίαση.



Εικόνα 22 : Παράδειγμα δενδρικής απεικόνισης της δομής ενός java project από το Java Model του Eclipse JDT

Ο αλγόριθμος ανίχνευσης απεικονίζεται με τη μορφή ψευδοκώδικα στην Εικόνα 23, ενώ στην Εικόνα 24 απεικονίζεται το τμήμα του αλγορίθμου που αφορά τον έλεγχο του κριτηρίου επιλογής υποθετικών δηλώσεων που περιλαμβάνονται σε ιδιωτικές μεθόδους. Τα υπόλοιπα αποσπάσματα του αλγορίθμου που παραλήφθηκαν στο σημείο αυτό χάριν απλοποίησης (σημεία @see στην Εικόνα 23) παρουσιάζονται στις Εικόνα 70 και Εικόνα 71 του Παράρτημα Γ : Αλγόριθμος

Ακολούθως, στις επόμενες υποενότητες αναλύονται περαιτέρω ορισμένα τμήματα του αλγορίθμου ανίχνευσης που κρίθηκε ότι χρήζουν περαιτέρω αποσαφήνισης. Συγκεκριμένα παρουσιάζεται η μεθοδολογία που χρησιμοποιήθηκε για την εύρεση των περιπτώσεων υποθετικής λογικής με παραμέτρους, οι οποίες απορρίπτονται καθώς η τιμή μιας τουλάχιστον από τις παραμέτρους τροποποιείται πριν την εκτέλεση του ελέγχου. Επιπλέον παρουσιάζεται η μεθοδολογία ανίχνευσης προτάσεων αναδόμησης στις οποίες είναι εφικτή, η πλήρης αντικατάσταση της υποθετικής λογικής.

```
For every class in system
        For every method in class
                If method body not empty
                Then Do
                        Find all conditional statements in method
                        For every conditional statement in method
                           If conditional statement has at least two branches
                           Then Do
                                @see analyze conditional expressions
                                analyze conditional expressions
                                If conditional expressions valid
                                Then Do
                                      @see analyze conditional branches
                                      analyze conditional branches
                                EndIF
        /* check method level criteria based on information retrieved by previous analysis*/
                                @see check private method criterion
                                If method not invalid private* AND method not static AND method
                                not overriding method outside system boundary
                                Then Do
                                      @see check valid sized conditional statement
    /* check condition & branch level criteria based on data retrieved by previous analysis */
                                      If conditional statement not RTTI AND conditional statement
                                      has valid size*
                                      Then Do
                                      /* further check of condition level criteria criterion A*/
                                            If conditional statement has parameters in conditions
                                            Then Do
                                                Check method doesn't contain assignments in
                                                parameters before conditional statement
                                             EndIF
                                            /* criterion B*/
                                            If conditional statement has fields in conditions
                                            Then Do
                                                 Check context not contains assignments on fields
                                            EndIF
                                            If (criterion A AND criterion B) OR (criteria A is inactive
                                            AND criterion B is inactive )
                                            Then Do
                                            /* check if conditional logic can be fully removed */
                                            If all conditional parameters/fields are primitives or enum
                                            Then do
                                                 Check if values can be found at compilation time
                                            EndIF
                                                Calculate refactoring' s scoring
                                                Add conditional statement to refactoring candidates
                                            EndIF
                                      EndIF
                                 EndIF
                           EndIF
                        EndFor
                EndIF
        EndFor
        Return refactoring candidates
EndFor
```

Εικόνα 23: Αλγόριθμος ανίχνευσης υποψήφιων υποθετικών δηλώσεων προς αναδόμηση με ψευδοκώδικα

```
   check private method criterion
1  /* check criterion for conditionals in private methods approval */
2  If method with conditional logic is private
3  Then do
4          If conditional has parameters
5          Then do
6                  Find method's invocations
7                  For every method invocation
8                  Do
9                    If method containing invocation is public
10                   Then Do
11                           For every argument passed
12                           Do
13                               If argument not parameter nor field
14                               Then Do
15                                   return false
16                               EndIf
17                               Else if argument is field
18                                     Execute lines 43-48
19                               End Else
20                           EndFor
21                   EndIF
22                   If method containing invocation is private
23                   Then Do
24                           For every argument passed
25                           Do
26                               If argument not parameter nor field
27                               Then Do
28                                   return false
29                               EndIf
30                               Else if argument is field
31                                     Execute lines 43-48
32                               Else if argument is parameter and backward steps ==1
33                                    repeat execution from line 6 for the caller method
34                               End Else
35                           EndFor
36                   EndIF
37                 EndFor
38         EndIF
39         If conditional has fields
40         Then do
41                 For every field in conditional
42                 Do
43                     Find field assignments in context except
44                     from the constructor and their setter method
45                     If assignments exist
46                     Then Do
47                             return false
48                     EndIf
49         EndIF
50 EndIF
51 return true
```

Εικόνα 24 : Αλγόριθμος ελέγχου εγκυρότητας private μεθόδων

5.3.1. Πλήρης Έλεγχος Πελάτη στην Υποθετική Λογική

Προκειμένου να εντοπιστούν οι περιπτώσεις όπου ο έλεγχος στη συνθήκη της υποθετικής έκφρασης πραγματοποιείται βάσει παραμέτρων, ωστόσο ουσιαστικά η επιλογή της εκτελούμενης παραλλαγής του αλγορίθμου συντελείται στα πλαίσια της κλάσης context, δεδομένου του ότι οι εντολές της κλάσης παρεμβαίνουν και τροποποιούν τις τιμές των παραμέτρων, εκχωρώντας τους κάποια άλλη τιμή πριν την εκτέλεση της υποθετικής δήλωσης, καθίσταται αναγκαία η χρήση τόσο στατικών πληροφοριών, σχετικά με τις εντολές του πηγαίου κώδικα, όσο και πληροφοριών που σχετίζονται με τη ροή του ελέγχου στο πρόγραμμα.

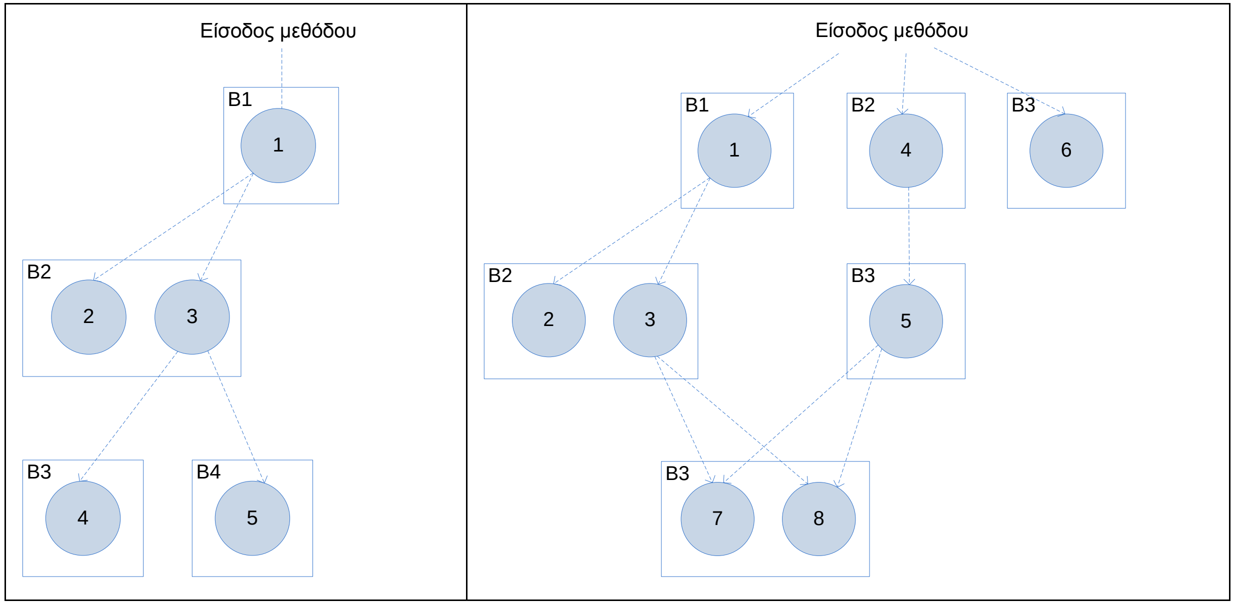
Η ανάγκη εξέτασης της ροής ελέγχου του προγράμματος πηγάζει από το γεγονός ότι προκειμένου η εκχώρηση στην τιμή της παραμέτρου, εφόσον υπάρχει, να επηρεάζει την εκτέλεση ή μη του ελέγχου και επομένως να πρέπει να απορριφθεί, θα πρέπει να βρίσκεται στο ίδιο μονοπάτι εκτέλεσης ή τουλάχιστον σε ένα από τα μονοπάτια εκτέλεσης που καταλήγουν στην εντολή του ελέγχου. Η Εικόνα 25 παρουσιάζει δύο παραδείγματα μεθόδων με εκχωρήσεις στις τιμές των παραμέτρων, ωστόσο μόνο στην περίπτωση του κώδικα της Εικόνα 25 α, η εκχώρηση βρίσκεται στο ίδιο μονοπάτι εκτέλεσης με την υποθετική λογική.

Συνεπώς, προκειμένου να αναλυθεί η ροή ελέγχου του προγράμματος, πέραν από τη χρήση του AST που περιλαμβάνει πληροφορίες σχετικά με τη δομή του πηγαίου κώδικα, χρησιμοποιούνται και δύο άλλες μορφές αναπαράστασης που απεικονίζουν τόσο τη ροή του ελέγχου, όσο και των δεδομένων. Οι αναπαραστάσεις αυτές είναι ο γράφος ροής ελέγχου (control flow graph - CFG) και ο γράφος εξαρτήσεων προγράμματος (program dependence graph - PDG), ο οποίος περιλαμβάνει τόσο τις εξαρτήσεις ελέγχου όσο και τις εξαρτήσεις δεδομένων (Εικόνα 26). Η δημιουργία των γράφων έγινε χρησιμοποιώντας τη σχετική υλοποίηση που είχε αναπτύχθει στα πλαίσια της ανίχνευσης περιπτώσεων αναδόμησης με εξαγωγή μεθόδου στο (85). Ουσιαστικά ο γράφος CFG της κλάσης έχει βοηθητική λειτουργία και χρησιμοποιείται ως ενδιάμεσο στάδιο για την κατασκευή του αντίστοιχου γράφου PDG της μεθόδου. Ακολούθως, ο γράφος PDG αξιοποιείται για την εύρεση του συνόλου των boundary blocks της υποθετικής δήλωσης και την εύρεση εντολών εκχώρησης στις παραμέτρους της υποθετικής έκφρασης, οι οποίες πιθανώς περιλαμβάνονται στο σύνολο των κόμβων των boundary blocks. Με βάση την

```
public void printMessage(int alignment,
boolean changeAllignment) {

1     if(changeAllignment) {
2            allignment = HORIZONTAL;
3            if(allignment == VERTICAL)
4                 printVertically();
              } else {
5                 printHorizontically();
              }
      }
}
                    □ boundary blocks of statement 3
```

```
public void printSales (int alignment,
boolean header) {

1     if (isHeader) {
2            allignment = HORIZONTAL;
3            printHeader(allignment)
      } else {
4            if(allignment == VERTICAL){
5                 printMsgVertically();
             } else {
6                 printMsgHorizontically();
             }
      }
7     allignment = HORIZONTAL;
8     printFooter(allignment)
}
                    □ boundary blocks of statement 4
```

Εικόνα 25 : Παραδείγματα μεθόδων με εκχώρηση στις παραμέτρους της υποθετικής λογικής α) μη έγκυρη περίπτωση με εκχώρηση στο ίδιο μονοπάτι εκτέλεσης, β) έγκυρη περίπτωση με εκχώρηση σε διαφορετικό μονοπάτι εκτέλεσης



Εικόνα 26 : Γράφος εξαρτήσεων ελέγχου των μεθόδων της Εικόνας 25 α) και β). Στην περίπτωση της μεθόδου α το σύνολο των boundary blocks για τη δήλωση 3 είναι {B1, B2} ενώ για τη μέθοδο β) το σύνολο των boundary blocks είναι το {B2}.

πληροφορία που αντλείται από τους γράφους, η μεθοδολογία ανίχνευσης που αναπτύχθηκε περιλαμβάνει τα εξής βήματα:

1. Έλεγχος ότι η υποθετική έκφραση περιλαμβάνει παραμέτρους.
2. Κατασκευή του γράφου ροής ελέγχου (CFG) της μεθόδου με την υποθετική λογική.
3. Κατασκευή του γράφου εξαρτήσεων του προγράμματος PDG βάσει του CFG που δημιουργήθηκε στο προηγούμενο βήμα για τη μέθοδο με την υποθετική λογική.
4. Εύρεση του κόμβου υποθετικής λογικής στο PDG διάγραμμα.
5. Έλεγχος ταύτισης με τον πρώτο κόμβο του γραφήματος για διαφυγή.
6. Υπολογισμός του συνόλου των boundary blocks του κόμβου υποθετικής λογικής.
7. Εύρεση όλων των εντολών εκχώρησης που περιλαμβάνονται στο σύνολο των κόμβων των boundary blocks εξαιρουμένου του κόμβου υποθετικής λογικής.
8. Έλεγχος του αριθμού των εντολών εκχώρησης που εντοπίστηκαν για διαφυγή στην περίπτωση μηδενικού αριθμού.
9. Αναζήτηση σχετικής εκχώρησης καθεμιάς από τις παραμέτρους της υποθετικής λογικής στο σύνολο των εκχωρήσεων που ανιχνεύθηκαν.

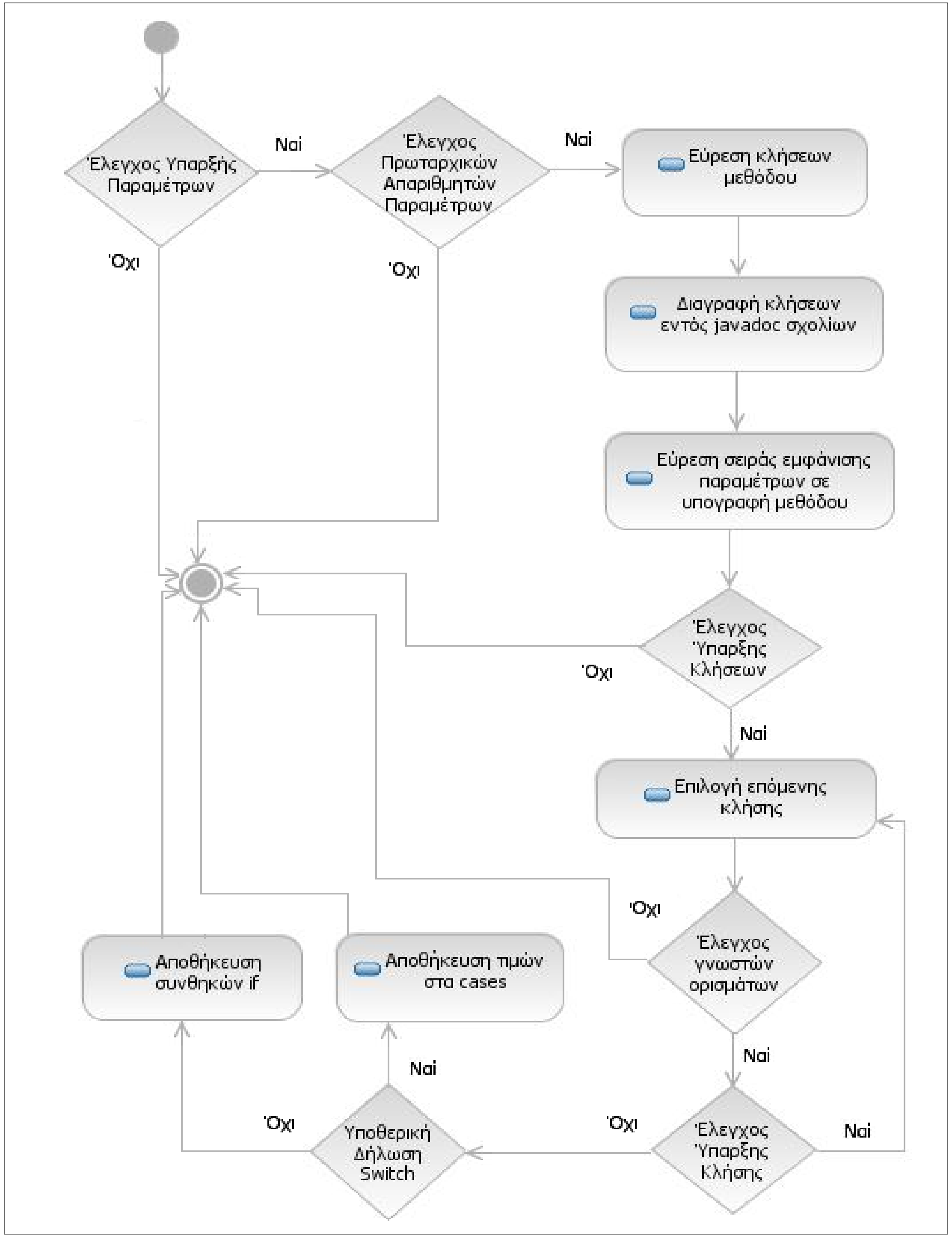
5.3.2. Ανίχνευση Πλήρους Αντικατάστασης Υποθετικής Λογικής

Η ανίχνευση υποψήφιων προτάσεων αναδόμησης όπου η υποθετική λογική μπορεί να αντικατασταθεί πλήρως απαιτεί την επέκταση του ελέγχου εκτός των ορίων της κλάσης context, και την ανάλυση των κλήσεων της μεθόδου στο εσωτερικό των κλάσεων πελάτη. Απαραίτητη προϋπόθεση προκειμένου οι κλάσεις πελάτες μετά την αναδόμηση να προσδιορίζουν την επιθυμητή στρατηγική, περνώντας την ως παράμετρο στην κλάση context αποτελεί η γνώση των τιμών των παραμέτρων σε χρόνο μεταγλώττισης (compilation time) και όχι σε χρόνο εκτέλεσης (runtime). Συνεπώς, το πρόβλημα ανίχνευσης υποψήφιων προτάσεων αναδόμησης με πλήρη υποκατάσταση της υποθετικής λογικής ανάγεται στο πρόβλημα εντοπισμού μεθόδων, τα ορίσματα των οποίων στις κλήσεις τους έχουν καθορισμένες τιμές κατά το χρόνο μεταγλώττισης.

Η πλήρης επίλυση του προβλήματος αυτού απαιτεί να ληφθεί μέριμνα τόσο για την περίπτωση όπου οι παράμετροι της υποθετικής δήλωσης έχουν πρωταρχικό ή απαριθμητό τύπο (primitive και enum), καθώς και για την περίπτωση που αποτελούν αντικείμενα (objects). Στην περίπτωση των αντικειμένων η πολυπλοκότητα του ελέγχου αυξάνεται εκθετικά καθώς η ανίχνευση δυνατότητας πλήρους αντικατάστασης της υποθετικής λογικής προϋποθέτει να μπορεί να προσδιοριστεί με ακρίβεια το πεδίο/πεδία του αντικειμένου στα οποία βασίζεται ο έλεγχος της υποθετικής έκφρασης και να ανιχνευθούν οι συγκεκριμένες εκχωρήσεις που τους αναθέτουν μια συγκεκριμένη τιμή (είτε με χρήση του τελεστή =, είτε με χρήση setter μεθόδου, είτε μέσω του κατασκευαστή), καθώς και όλες οι κλήσεις μεθόδων που ενδέχεται να επηρεάζουν την τιμή αυτή. Σε περίπτωση δε που το πεδίο/πεδία αυτά αποτελούν πάλι αντικείμενα, ο έλεγχος θα πρέπει να εκτελείται αναδρομικά για καθένα από αυτά μέχρις ότου αποκλεισθεί το ενδεχόμενο, η τιμή τους να είναι γνωστή στο χρόνο μεταγλώττισης, κάθε φορά που χρησιμοποιούνται ως παράμετροι σε κλήσεις της μεθόδου με την υποθετική λογική.

Κατά συνέπεια στα πλαίσια της παρούσας εργασίας ο έλεγχος περιορίζεται σε υποθετικές εκφράσεις με παραμέτρους πρωταρχικού και απαριθμητού τύπου και η αναζήτηση αποπειράται να εντοπίσει κλήσεις μεθόδων όπου στις παραμέτρους που συμμετέχουν στη λογική ελέγχου καθορίζονται ως ορίσματα συγκεκριμένες τιμές.

Ο αλγόριθμος που αναπτύχθηκε για το σκοπό αυτό απεικονίζεται διαγραμματικά στην Εικόνα 27. Ο αλγόριθμός αυτός εκτελείται έπειτα από την εκτέλεση όλων των ελέγχων αναφορικά με τα κριτήρια που αναφέρθηκαν στην ενότητα 5.2. Εν συνεχεία, εφόσον η υπό εξέταση υποθετική δήλωση περιλαμβάνει παραμέτρους, ελέγχεται κατά πόσο καθεμία από τις παραμέτρους αυτές έχει πρωταρχικό ή απαριθμητό τύπου. Έπειτα εντοπίζονται όλες οι κλήσεις της μεθόδου με την υποθετική λογική, ενώ απορρίπτονται ενδεχόμενες μη εκτελέσιμες κλήσεις που περιλαμβάνονται στο σώμα javadoc σχολίων. Ακολούθως, βρίσκεται και αποθηκεύεται η σειρά εμφάνισης καθεμίας από τις παραμέτρους στην λίστα ορισμάτων της μεθόδου, ούτως ώστε να μπορεί να αντιστοιχιστεί με τις τιμές των ορισμάτων που ορίζονται στην κλήση της μεθόδου. Με το πέρας του βήματος αυτού, εφόσον έχουν εντοπιστεί κλήσεις της μεθόδου, καθεμία από αυτές τυγχάνει ξεχωριστής επεξεργασίας και ελέγχου προκειμένου να διαπιστωθεί αν περιλαμβάνει γνωστή τιμή στις ζητούμενες παραμέτρους κατά το χρόνο εκτέλεσης, δηλαδή κάποια συγκεκριμένη αριθμητική τιμή, κάποια συμβολοσειρά, κάποια σταθερά, κ.ο.κ. Με την επιτυχή ολοκλήρωση του ελέγχου αυτού για όλες τις κλήσεις μεθόδων, το μόνο που απομένει είναι η αντίστοιχη επεξεργασία και συγκέντρωση πληροφοριών αναφορικά με τις υποθετικές δηλώσεις if και switch, ώστε κατά την εφαρμογή της αναδόμησης να μπορεί να εντοπιστεί η συνθήκη που αληθεύει και να μπορεί να αντιστοιχιστεί σε κάθε κλήση, η κατάλληλη κλάση στρατηγική, στην οποία έχει μεταφερθεί ο αλγόριθμος της αληθούς συνθήκης.



Εικόνα 27: Διαγραμματική απεικόνιση του αλγόριθμου εύρεσης υποψήφιων προτάσεων πλήρους αντικατάστασης της υποθετικής λογικής

5.4. Αποτίμηση Ανίχνευσης

Οι προτάσεις αναδόμησης με χρήση του προτύπου στρατηγική παρουσιάζονται στον χρήστη υπό μορφή ταξινομημένης λίστας. Κατ' αντιστοιχία με την υπάρχουσα μεθοδολογία του JDeodorant, οι προτεινόμενες αναδομήσεις ταξινομούνται, προκειμένου να διευκολυνθεί η επιλογή του προγραμματιστή ως προς την εφαρμογή της αναδόμησης. Τα κριτήρια με βάση τα οποία πραγματοποιείται η ταξινόμηση των προτεινόμενων αναδομήσεων είναι τα εξής:

- ένα ανεξάρτητο βάρους το οποίο αποδίδεται από το σύστημα με βάση τα συμπληρωματικά κριτήρια σε επίπεδο διακλάδωσης που πληρούνται στην εκάστοτε περίπτωση
- ο μέσος αριθμός δηλώσεων ανά διακλάδωση της υποθετικής δήλωσης (average statements / block).

Αναφορικά με τα ανεξάρτητα βάρη που αποδίδονται στις προτεινόμενες αναδομήσεις, κρίνεται αναγκαίο να αποσαφηνιστεί η λογική απόδοσης τους. Επιλέχθηκαν δύο τύποι βαρών:

- τα βασικά που αποδίδουν τιμή σε καθεμία από τις προτάσεις αναδόμησης ανάλογα με τη δομή της, και
- τα συμπληρωματικά, τα οποία προσαυξάνουν την τιμή του αρχικού βάρους εφόσον πέραν των χαρακτηριστικών της δομής, πληρούνται ορισμένες επιπλέον προϋποθέσεις.

Οι αριθμητικές τιμές των ανεξάρτητων βαρών επιλέχθηκαν τυχαία, κατά τρόπο ώστε να είναι διακριτές και να είναι εμφανής η διαφοροποίηση τους. Το εύρος τιμών των βασικών βαρών [1,5], επιλέγει έχοντας ως παράδειγμα την πενταβάθμια κλίμακα Likert, ενώ το βήμα 2 μεταξύ των διαφορετικών βαρών ανά κριτήριο επιλέγει, ούτως ώστε οι τιμές που προκύπτουν μετά την προσαύξηση του βάρους από την εφαρμογή των συμπληρωματικών βαρών, να είναι και πάλι διακριτές και να υπάρχει ένα-προς-ένα αντιστοιχία μεταξύ των χαρακτηριστικών της δομής των προτάσεων αναδόμησης και του βάρους που τους αποδόθηκε. Ο ίδιος λόγος αποτέλεσε κριτήριο και κατά την επιλογή των τιμών των συμπληρωματικών βαρών, κατά τρόπο ώστε να μην συντελούν στην παραγωγή όμοιων τιμών για διαφορετικού είδους προτάσεις αναδόμησης δηλαδή διαφορετικές εισόδους. Η επιλογή των βαρών έχει ως στόχο δηλαδή, να καταστήσει εφικτή την οπτική ερμηνεία τους κατά τρόπο μοναδικό, αναφορικά με την δομή που αναμένεται να παρουσιάζει σε κάθε περίπτωση ο πηγαίος κώδικας.

Η διαβάθμιση των βαρών μεταξύ των διαφόρων περιπτώσεων βασίστηκε στην οπτική παρατήρηση της συσχέτισης της συχνότητας εμφάνισης των συγκεκριμένων περιπτώσεων σε συνδυασμό με την πιθανότητα θεώρησης τους ως υποψήφιες περιπτώσεις αναδόμησης προς τη στρατηγική, στα συστήματα λογισμικού που χρησιμοποιήθηκαν κατά την υλοποίηση και τον έλεγχο της προτεινόμενης μεθοδολογίας.

Στον Πίνακας 11, εμφανίζεται η αντιστοίχιση τιμών των ανεξάρτητων βαρών με καθένα από τα κριτήρια/προϋποθέσεις που ικανοποιεί ο πηγαίος κώδικας της υποθετικής λογικής.

	Κριτήριο	Βάρος
1	Υποθετική λογική με εκχώρηση σε πεδίο (field) σε κάθε διακλάδωση	5
2	Υποθετική λογική με δήλωση επιστροφής (return), σε κάθε διακλάδωση	3
3	Υποθετική λογική με παραμέτρους στις υποθετικές εκφράσεις χωρίς εκχώρηση ούτε δήλωση επιστροφής	1
4	Υποθετική λογική όπου αν κάθε μια από τις διακλαδώσεις εξαχθεί σε ξεχωριστή μέθοδο, τότε οι μέθοδοι που προκύπτουν έχουν την ίδια υπογραφή	+ 1
5	Υποθετική λογική η οποία δύναται να απαλειφθεί πλήρως κατά την εκτέλεση της αναδόμησης	+6

Πίνακας 11 : Αντιστοίχηση βαρών με κριτήρια απόδοσης τους

5.5. Εφαρμογή Αναδόμησης

Η εφαρμογή της αναδόμησης προς τη στρατηγική διαφοροποιείται από την αντίστοιχη εκτέλεση στο JDeodorant, ωστόσο παρουσιάζει και αρκετές ομοιότητες, καθώς τα δύο πρότυπα εμφανίζουν πανομοιότυπη δομή, συνεπώς για την εφαρμογή τους στον πηγαίο κώδικα θα πρέπει να εκτελεστούν οι ίδιοι μετασχηματισμοί. Η κυριότερη διαφοροποίηση της μεθοδολογίας έγκειται ουσιαστικά στην διαχείριση των νέων δεδομένων που προκύπτουν από τις αλλαγές στην ανίχνευση και στην ξεχωριστή αντιμετώπιση της περίπτωσης πλήρους απαλοιφής της υποθετικής λογικής. Ακολούθως, παρουσιάζεται η γενική περίπτωση αναδόμησης μερικής αντικατάστασης της υποθετικής λογικής προσαρμοσμένη στις ιδιαιτερότητες των ανιχνευμένων προτάσεων σε σύγκριση με την υπάρχουσα υλοποίηση του JDeodorant προκειμένου να καταστεί εμφανής η διαφοροποίηση των δύο υλοποιήσεων, και εν συνεχεία περιγράφεται η περίπτωση πλήρους αντικατάστασης υποθετικής λογικής.

5.5.1. Μερική Αντικατάσταση Υποθετικής Λογικής

Στην πλειονότητα προτάσεων αναδόμησης προς τη στρατηγική δεν είναι εφικτό να απαλειφτούν πλήρως οι πολύπλοκες υποθετικές δηλώσεις. Στην περίπτωση αυτή η τροποποίηση του πηγαίου κώδικα ουσιαστικά περιορίζεται στην απλοποίηση της υποθετικής δήλωσης μεταφέροντας τον κώδικα των διακλαδώσεων σε ξεχωριστές μεθόδους των συγκεκριμένων υποκλάσεων στρατηγικής.

Για το σκοπό αυτό αρχικά, η προτεινόμενη εφαρμογή της αναδόμησης προς τη στρατηγική ακολουθεί διαφορετική λογική ονοματοθεσίας των κλάσεων που δημιουργούνται. Στην υπάρχουσα υλοποίηση του JDeodorant ως προεπιλεγμένο όνομα της αφηρημένης κλάσης κατάσταση/στρατηγική οριζόταν το όνομα του πεδίου βάσει του οποίου εκτελούνταν ο έλεγχος τύπου, στο τμήμα της υποθετικής δήλωσης, ενώ τα ονόματα των υποκλάσεων καθορίζονταν από τα αντίστοιχα ονόματα των σταθερών. Ωστόσο, έπειτα από την τροποποίηση των κριτηρίων ανίχνευσης, η εφαρμογή της ονοματοθεσίας κατ' αυτό τον τρόπο δεν είναι πλέον εφικτή, καθώς δεν είναι υποχρεωτική η ύπαρξη σταθερών σε όλες τις ανιχνεύσιμες προτάσεις αναδόμησης. Κατά συνέπεια, το προεπιλεγμένο όνομα της αφηρημένης κλάσης στρατηγικής παράγεται από το όνομα της μεθόδου που φιλοξενεί την υποθετική λογική σε συνδυασμό με το επίθεμα Strategy. Κατ' αντιστοιχία ορίζονται και τα προεπιλεγμένα ονόματα των συγκεκριμένων υποκλάσεων της στρατηγικής, ως αποτέλεσμα του συνδυασμού του ονόματος της μεθόδου ακολουθούμενου από το επίθεμα "StrategyImpl_" και τέλος από έναν αύξων αριθμό. Ο αύξων αριθμός που προσδίδεται στις υποκλάσεις στρατηγικής παράγεται ξεκινώντας την αρίθμηση από την τιμή 1.

Επιπροσθέτως, στην υπάρχουσα υλοποίηση η ιεραρχία των κλάσεων ορισμένες φορές προϋπήρχε, οπότε δεν απαιτούνταν να δημιουργηθεί εξαρχής και η αναδόμηση απλά πρόσθετε στις υπάρχουσες κλάσεις της ιεραρχίας τους ορισμούς των νέων μεθόδων. Η περίπτωση αυτή αποκλείεται στη νέα υλοποίηση, οπότε η ιεραρχία των κλάσεων δημιουργείται εξαρχής κάθε φορά που εφαρμόζεται η αναδόμηση.

Κατ' αντιστοιχία, επιπλέον διαφοροποίηση της εκτέλεσης έγκειται στο γεγονός ότι στην αναδόμηση προς την κατάσταση/στρατηγική, η μεταβλητή μέλος *τύπος (type code)* στην πλειονότητα των ανιχνευμένων περιπτώσεων προϋπήρχε. Κατά συνέπεια στην περίπτωση αυτή, ο μετασχηματισμός αφορούσε μόνο την τροποποίηση του πρωταρχικού τύπου της μεταβλητής μέλους στον τύπο της αφηρημένης κλάσης κατάσταση/στρατηγική. Στην προτεινόμενη αναδόμηση προς στρατηγική, εφόσον η ανίχνευση έχει αποδεσμευθεί από την προϋπόθεση ύπαρξης κωδικού που υποδηλώνει τύπο *(type code)* στην κλάση context, η μεταβλητή μέλος στρατηγική δημιουργείται εξαρχής με προεπιλεγμένο όνομα, το όνομα της αφηρημένης κλάσης στρατηγικής και τύπο, τον τύπο της νεοδημιουργηθείσας κλάσης, και τοποθετείται αμέσως μετά τη δήλωση της κλάσης. Παράλληλα δημιουργούνται και οι αντίστοιχες getter και setter μέθοδοι, για την μεταβλητή μέλος στρατηγική όπως και στην υπάρχουσα υλοποίηση. Ωστόσο, η αναδόμηση προς κατάσταση/στρατηγική όριζε την μέθοδο setter κατά τρόπο ώστε να δημιουργεί το αντίστοιχο στιγμιότυπο της κατάλληλης υποκλάσης κατάσταση/στρατηγική,δηλώνοντας στο εσωτερικό της μια εντολή switch που βασιζόταν στον έλεγχο της τιμής του ορίσματος της μεθόδου για την επιλογή του προς δημιουργία αντικειμένου. Αντιθέτως, στη νέα υλοποίηση μοναδική λειτουργία της setter μεθόδου αποτελεί η εκχώρηση της τιμής της παραμέτρου στην μεταβλητή μέλος στρατηγική.

Εν συνεχεία, ένα άλλο κομμάτι της εφαρμογής της αναδόμησης που διαφοροποιείται, είναι το κομμάτι που αφορά την τροποποίηση της μεθόδου της κλάσης context, η οποία εμπεριέχει την υποθετική λογική. Στην υπάρχουσα υλοποίηση, η αναδόμηση είχε ως αποτέλεσμα την δημιουργία μεθόδων στις διάφορες υποκλάσεις, αντιγράφοντας το κομμάτι του κώδικα κάθε διακλάδωσης, και τη δημιουργία της αντίστοιχης αφηρημένης μεθόδου στην υπερκλάση. Επιπλέον, η πολυμορφική κλήση της μεθόδου της κλάσης κατάστασης/στρατηγικής μέσω της αντίστοιχης μεταβλητής μέλους, η οποία καθοριζόταν από την αντίστοιχη setter μέθοδο, αντικαθιστούσε την υποθετική δήλωση. Ωστόσο, στη νέα προσέγγιση για την εφαρμογή της αναδόμησης προς τη στρατηγική η υποθετική λογική στο εσωτερικό της μεθόδου παραμένει και μετά την εφαρμογή του μετασχηματισμού. Η τροποποίηση που εκτελείται συνίσταται στη δημιουργία μεθόδων αντιγράφοντας στο εσωτερικό τους τον κώδικα των διακλαδώσεων της υποθετικής δήλωσης και στην αντικατάσταση του κώδικα αυτού στην πρωτότυπη μέθοδο, από πηγαίο κώδικα ο οποίος πραγματοποιεί τη δημιουργία του κατάλληλου αντικειμένου στρατηγικής, την εκχώρηση του αντικειμένου στη μεταβλητή μέλος στρατηγική, και τέλος την πολυμορφική κλήση της μεθόδου μέσω του πεδίου στρατηγική.

Πέραν από τις ανωτέρω διαφορές που παρουσιάστηκαν σε υψηλό επίπεδο, οι οποίες αφορούν τη γενική περίπτωση αναδόμησης, η προτεινόμενη μεθοδολογία διαφοροποιείται κυρίως στην περίπτωση όπου είναι εφικτή η πλήρης απομάκρυνση της υποθετικής δήλωσης, η οποία δεν υποστηρίζεται από την υπάρχουσα υλοποίηση του JDeodorant, εφόσον η υποθετική δήλωση απλά μεταφέρεται στη setter μέθοδο της κατάστασης. Στην επόμενη ενότητα αναλύεται η μεθοδολογία εύρεσης της κατάλληλης στρατηγικής κατά την εκτέλεση της αναδόμησης και παρουσιάζεται συνοπτικά ο εκτελούμενος μετασχηματισμός.

5.5.2. Πλήρης Αντικατάσταση Υποθετικής Λογικής

Κατά την εκτέλεση της αναδόμησης με το πρότυπο στρατηγική εφόσον η υπό αναδόμηση υποθετική λογική εντάσσεται στο σύνολο των προτάσεων εκείνων στις οποίες είναι εφικτή η πλήρης αντικατάσταση των δηλώσεων if/switch (Ενότητα 0.0.5.3.2), καθίσταται αναγκαία η τροποποίηση της υπογραφής της μεθόδου ούτως ώστε να δέχεται σαν παράμετρο το συγκεκριμένο αντικείμενο της κλάσης στρατηγική. Η παράμετρος αυτή χρησιμοποιείται για την ολική αντικατάσταση της υποθετικής δήλωσης από την εντολή εκχώρησής της στο αντίστοιχο πεδίο της κλάσης καθώς και για την πολυμορφική κλήση της μεθόδου με την κατάλληλη παραλλαγή του αλγορίθμου.

Πέραν των τροποποιήσεων που υφίσταται η κλάση context, απαιτείται η τροποποίηση των κλήσεων της μεθόδου στις κλάσεις πελάτη (client) προκειμένου να προσδιορίζουν ως όρισμα κατά την κλήση την επιθυμητή στρατηγική που θα πρέπει να εκτελεστεί. Για το σκοπό αυτό καθίσταται αναγκαίος ο προσδιορισμός της συνθήκης η οποία ενεργοποιείται από τα ορίσματα που περνιούνται κατά την κλήση της μεθόδου, αποδίδοντας της την αληθοτιμή «αληθές». Ωστόσο η πληροφορία αυτή, δεν είναι εφικτό να αντληθεί από κάποια από τις στατικές μορφές αναπαράστασης του κώδικα (AST, CFG, PDG), καθώς παράγεται δυναμικά κατά την εκτέλεση του κώδικα. Συνεπώς καθίσταται αναγκαία η εκτέλεση του πηγαίου κώδικα.

Ωστόσο, η εκτέλεση συντελείται εκτός του προγράμματος στο οποίο δηλώνεται, στα πλαίσια του εργαλείου αναδόμησης καθώς δεν αφορά τον πηγαίο κώδικα στο σύνολο του, παρά μόνο το κομμάτι των κλήσεων μεθόδων και το κομμάτι της υποθετικής λογικής, ούτως ώστε να βρεθούν οι συνθήκες που αληθεύουν για κάθε σύνολο τιμών των παραμέτρων που ορίζονται στα πλαίσια των κλήσεων. Για το λόγο αυτό δε χρησιμοποιήθηκε η μέθοδος δυναμικής ανάλυση του πηγαίου κώδικα, ούτε κάποιο από τα διαθέσιμα εργαλεία που την υποστηρίζουν, αλλά υλοποιήθηκε ένας πρωτότυπος τρόπος εκτέλεσης των τμημάτων του πηγαίου κώδικα που συνιστούν αντικείμενο ενδιαφέροντος, ο οποίος παρουσιάζεται και επεξηγείται περαιτέρω στην ενότητα 1.1.5.6.2.

Ακολούθως παρουσιάζονται τα βήματα της μεθοδολογίας που ακολουθήθηκαν κατά την εφαρμογή της αναδόμησης για τον προσδιορισμό των κλάσεων στρατηγικής που θα πρέπει να αντικαταστήσουν τα υπάρχοντα ορίσματα στο σύνολο των κλήσεων της μεθόδου:

1. Εύρεση της κλάσης στην οποία περιλαμβάνεται η υπό εξέταση κλήση μεθόδου και δημιουργία του αντίστοιχου AST.
2. Μεμονωμένη επεξεργασία των συνθηκών που απαρτίζουν τη δήλωση if/switch και αντικατάσταση των σχετικών παραμέτρων από τις συγκεκριμένες τιμές που προσδιορίζονται στην κλήση της μεθόδου
3. Αποτίμηση της συνθήκης μέσω της δυναμικής δημιουργίας στο χρόνο εκτέλεσης μιας νέας μεθόδου, η οποία ελέγχει και επιστρέφει την αληθοτιμή της συνθήκης για το συγκεκριμένο σύνολο ορισμάτων.

4. Έλεγχος ισότητας της υπολογισμένης αληθοτιμής με την τιμή «αληθές» και εύρεση του ονόματος της στρατηγικής στην οποία μεταφέρθηκε ο πηγαίος κώδικας της αντίστοιχης διακλάδωσης της αληθούς συνθήκης, εφόσον ο έλεγχος ικανοποιείται.

Περισσότερες λεπτομέρειες σχετικά με την υλοποίηση των ανωτέρω βημάτων αναδόμησης ειδικά και της αναδόμησης προς τη στρατηγική γενικά, παρουσιάζονται στην ενότητα που ακολουθεί όπου δίδονται περαιτέρω πληροφορίες αναφορικά με τις κλάσεις που δημιουργήθηκαν, τις κλάσεις που υπέστησαν τροποποίηση, κ.ο.κ, και επιπλέον παρατίθενται συγκεκριμένα παραδείγματα από την εκτέλεση της αναδόμησης.

5.6. Υλοποίηση

Η προτεινόμενη μεθοδολογία αυτόματης ανίχνευσης και αναδόμησης προς τη στρατηγική που παρουσιάστηκε στις προηγούμενες ενότητες του κεφαλαίου βασίστηκε στην ήδη υπάρχουσα υλοποίηση του JDeodorant αξιοποιώντας στοιχεία της δομής του, αλλά και της ίδιας της λειτουργικότητας του. Η υλοποίηση που αναπτύχθηκε ενσωματώθηκε ως πρόσθετο κομμάτι του JDeodorant, χωρίς την τροποποίηση της ήδη υπάρχουσας λειτουργικότητας αναφορικά με την αναδόμηση προς την κατάσταση ή τη στρατηγική, ενώ παράλληλα από πλευράς λειτουργικότητας προστέθηκε η δυνατότητα παραμετροποίησης των προτεινόμενων κριτηρίων ανίχνευσης.

Η λειτουργικότητα που προστέθηκε αναπτύχθηκε σχεδόν εξ' ολοκλήρου σε καινούργιες ξεχωριστές κλάσεις, ενώ σε ορισμένες περιπτώσεις απαιτήθηκε η επέκταση υπαρχουσών κλάσεων μέσω της προσθήκης νέας λειτουργικότητας. Επιπλέον, σε ορισμένες περιπτώσεις, για την αποφυγή επανάληψης κοινού κώδικα μεταξύ των δύο προσεγγίσεων, εφαρμόστηκε αναδόμηση του υπάρχοντα κώδικα με αξιοποίηση της κληρονομικότητας και του πολυμορφισμού και εφαρμόστηκαν περιορισμένης κλίμακας τροποποιήσεις προκειμένου οι απαραίτητες κλάσεις να υιοθετήσουν ομαλά τη νέα δομή του μετασχηματισμένου κώδικα. Η αναδόμηση αυτή εκτελέστηκε προσεκτικά, ούτως ώστε να διατηρηθεί η υπάρχουσα συμπεριφορά, και μόνο σε περιπτώσεις όπου απαιτήθηκε, για την προσθήκη της νέας λειτουργικότητας χωρίς την υποβάθμιση του υπάρχοντος σχεδίου. Επιπλέον, η εφαρμογή της αναδόμησης στις επιλεγμένες κλάσεις έχει ως αποτέλεσμα την βελτίωση της επεκτασιμότητας του JDeodorant, ούτως ώστε να διευκολυνθεί η προσθήκη νέων προσεγγίσεων αναδόμησης με πρότυπα σχεδίασης, εφόσον αυτό κριθεί αναγκαίο μελλοντικά.

Για την υλοποίηση της προτεινόμενης μεθοδολογίας χρησιμοποιήθηκαν για την συντακτική ανάλυση του κώδικα και τη δημιουργία του AST δέντρου, καθώς και για τον εντοπισμό των προβληματικών σημείων στον κώδικα και την εφαρμογή της αναδόμησης, οι βιβλιοθήκες JDT (92) (93) (97) και LTK (95) του Eclipse. Επιπλέον για την ανίχνευση των προτάσεων πλήρους αντικατάστασης υποθετικής λογικής δημιουργήθηκαν οι απαιτούμενοι CFG και PDG γράφοι με αξιοποίηση της σχετικής υλοποίησης που είχε αναπτυχθεί από τον συγγραφέα της ερευνητικής εργασίας (85) για τις απαιτήσεις της αναδόμησης εξαγωγής μεθόδου. Βοηθητική ήταν επίσης ή χρήση του plug-in ASTViewer (79) για την επισκόπηση του AST δέντρου των υπο εξέταση συστημάτων λογισμικού που χρησιμοποιήθηκαν για τους σκοπούς του ελέγχου κατά τη διαδικασία ανάπτυξης.

Ακολούθως, παρατίθενται περαιτέρω λεπτομέρειες σχετικά με ορισμένες λειτουργίες της υλοποίησης που κρίθηκε ότι απαιτούν καλύτερη αποσαφήνιση όπως παραδείγματος χάριν η υλοποίηση της πλήρους αντικατάστασης υποθετικής λογικής ή της δυνατότητας παραμετροποίησης των κριτηρίων ανίχνευσης. Επιπλέον, παρουσιάζεται το γραφικό περιβάλλον του εργαλείου που υλοποιήθηκε και επιδεικνύεται η εφαρμογή της αναδόμησης σε ένα απλοποιημένο παράδειγμα. Τέλος, στο Παράρτημα Δ : Υλοποίηση Προτεινόμενης Μεθοδολογίαςπαρουσιάζονται συνοπτικά πληροφορίες σχετικά με τις τροποποιήσεις που απαιτήθηκαν στον υπάρχοντα κώδικα, και τις καινούργιες κλάσεις που δημιουργήθηκαν.

5.6.1. Εύρεση Κλήσεων Μεθόδου

Προκειμένου να ανιχνευθούν οι περιπτώσεις πλήρους αντικατάστασης υποθετικής λογικής καθώς και οι περιπτώσεις υποθετικών εκφράσεων στα πλαίσια ιδιωτικών μεθόδων, οι οποίες ωστόσο λαμβάνουν είσοδο εξωτερικά της κλάσης context, καθίσταται αναγκαία η ανίχνευση των κλήσεων των μεθόδων που της εμπεριέχουν και η περαιτέρω ανάλυσή τους. Για την ανίχνευση των κλήσεων μιας μεθόδου αξιοποιήθηκε η δυνατότητα αναζήτησης που παρέχεται από το JDT Core του Eclipse.

Η δυνατότητα αυτή καθιστά εύκολο τον γρήγορο εντοπισμό των οντοτήτων που συνθέτουν ένα σύστημα λογισμικού επιτρέποντας παραδείγματος χάριν την αναζήτηση της υλοποίησης μιας μεθόδου, ή μιας κλάσης, της αναφοράς ενός τύπου, μιας μεθόδου ή ενός πεδίου, ή αντίστοιχα την εύρεση της δήλωσής τους. Επιπλέον παρέχεται η δυνατότητα στο χρήστη να περιορίσει το πεδίο αναζήτησης στα πλαίσια μιας μεθόδου, κλάσης, κλπ. ή να το διευρύνει περιλαμβάνοντας ακόμα τις κλάσεις εκτός του συστήματος λογισμικού που περιλαμβάνονται στα πλαίσια βιβλιοθηκών που χρησιμοποιούνται από αυτό. Στην Εικόνα 28 απεικονίζεται ο κώδικας με τον οποίο επιτυγχάνεται η ανίχνευση των κλήσεων μιας μεθόδου.

Συγκεκριμένα, η προς αναζήτηση μέθοδος ορίζεται ως παράμετρος της μεθόδου. Για την ανίχνευση των κλήσεων της, ως αντικείμενο της αναζήτησης ορίζονται όλες οι αναφορές στη μέθοδο αυτή μέσω της εντολής:

SearchPattern.createPattern(iMethod,IJavaSearchConstants.REFERENCES);

ενώ ως πεδίο της αναζήτησης ορίζονται όλα τα πακέτα του υπό εξέταση συστήματος:

SearchEngine.createJavaSearchScope(sourceICompilationUnit.getJavaProject().getAllPackageFragmentRoots());

Η αναζήτηση εκτελείται με κλήση της μεθόδου *search()* της κλάσης SearchEngine του JDT και για την συλλογή των αποτελεσμάτων της αναζήτησης υπερφορτώνεται η μέθοδος *acceptSearchMatch()* της κλάσης SearchRequestor.

Εντούτοις, εμπόδιο στη διενέργεια της στατικής ανάλυσης αποτελεί το γεγονός ότι στα αποτελέσματα που προκύπτουν από την ανωτέρω αναζήτηση δεν περιλαμβάνεται η αναπαράσταση της κλήσης της μεθόδου ως κόμβο του AST δέντρου, παρά μόνο η ονομασία της καλούσας μεθόδου ως συμβολοσειρά (String). Η αναπαράσταση της κλήσης της μεθόδου ως κόμβο του δέντρου ωστόσο, καθίσταται απαραίτητη για την συνέχιση της στατικής ανάλυσης και την εξαγωγή συμπερασμάτων σχετικά με την ικανοποιησιμότητα των σχετικών κριτηρίων.

```
private List<SearchResult> findClients(MethodDeclaration methodDeclaration) {
	final List<SearchResult> tempSearchResults = new ArrayList<SearchResult>();
	IMethodBinding methodBinding = methodDeclaration.resolveBinding();
	IMethod iMethod = (IMethod) methodBinding.getJavaElement();
	try {
		// Create search pattern
		SearchPattern pattern = SearchPattern.createPattern(iMethod,IJavaSearchConstants.REFERENCES);
		ICompilationUnit                                    sourceICompilationUnit                                    =
		     (ICompilationUnit)((CompilationUnit)typeDeclaration.getRoot()).getJavaElement();
		IJavaSearchScope scope =SearchEngine.createJavaSearchScope(
		      sourceICompilationUnit.getJavaProject().getAllPackageFragmentRoots());

		SearchRequestor requestor = new SearchRequestor() {

			@Override
			public void acceptSearchMatch(SearchMatch match) throws CoreException {
				IJavaElement result = (IJavaElement) match.getElement();
				if (result instanceof IMethod) {
					IMethod method = (IMethod) result;
					ICompilationUnit cu = method.getCompilationUnit();
					Set<ClassObject> classObjects = ASTReader.getSystemObject().getClassObjects(cu);
					Object[] classArray = classObjects.toArray();
					ClassObject classObject = new ClassObject();
					if(classArray.length == 1) {
						classObject = (ClassObject) classArray[0];
					} else {
						String containerClass = result.getParent().getElementName();
						for(Object tempClassObject : classArray){
							if(((ClassObject)tempClassObject).getName().endsWith(containerClass)){
								classObject = (ClassObject)tempClassObject;
								break;
							}
						}
					}
					if(classObject != null) {
						tempSearchResults.add(new        SearchResult(match.getOffset(),match.getLength(),
						classObject));
```

```
                    }
                }
            }
        };
        /* Search Engine Declaration */
        SearchEngine searchEngine = new SearchEngine();
        try {
            searchEngine.search(pattern, new SearchParticipant[] { SearchEngine
            .getDefaultSearchParticipant() }, scope, requestor, null);
        } catch (CoreException e) {
            e.printStackTrace();
        }
    } catch (Exception e1) {
        e1.printStackTrace();
    } return tempSearchResults;
}
```

Εικόνα 28 : Εύρεση κλήσεων μεθόδου

Η πληροφορία αυτή εξάγεται έμμεσα από τα αποτελέσματα της αναζήτησης αποθηκεύοντας προσωρινά την κλάση στην οποία περιλαμβάνεται η μέθοδος, καθώς και τις πληροφορίες σχετικά με την θέση στην οποία εμφανίζεται η κλήση της μεθόδου στη συγκεκριμένη κλάση (offset και length) σε ένα αντικείμενο της κλάσης SearchResult, η οποία υλοποιήθηκε για τη μοντελοποίηση των αποτελεσμάτων της αναζήτησης. Ακολούθως τα αποτελέσματα αυτά συνδυάζονται με χρήση της κλάσης NodeFinder προκειμένου να ανιχνευθεί ο κόμβος της κλήσης μεθόδου στο AST δέντρο και να συνεχιστεί η στατική ανάλυση των κλήσεων (Εικόνα 29).

```
for (SearchResult result : currentSearchResults) {
        NodeFinder finder = new NodeFinder(result.getClassObject().getTypeDeclaration(),
                result.getOffset(), result.getLength());
        ASTNode parent = finder.getCoveredNode();
        /* find method invocation*/
```

```
        MethodInvocation methodInvocation = null;
        if(parent instanceof MethodInvocation){
                methodInvocation = (MethodInvocation) parent;
        } else if (parent.getParent() instanceof MethodInvocation) {
                methodInvocation = (MethodInvocation) parent.getParent();
        }
}
```

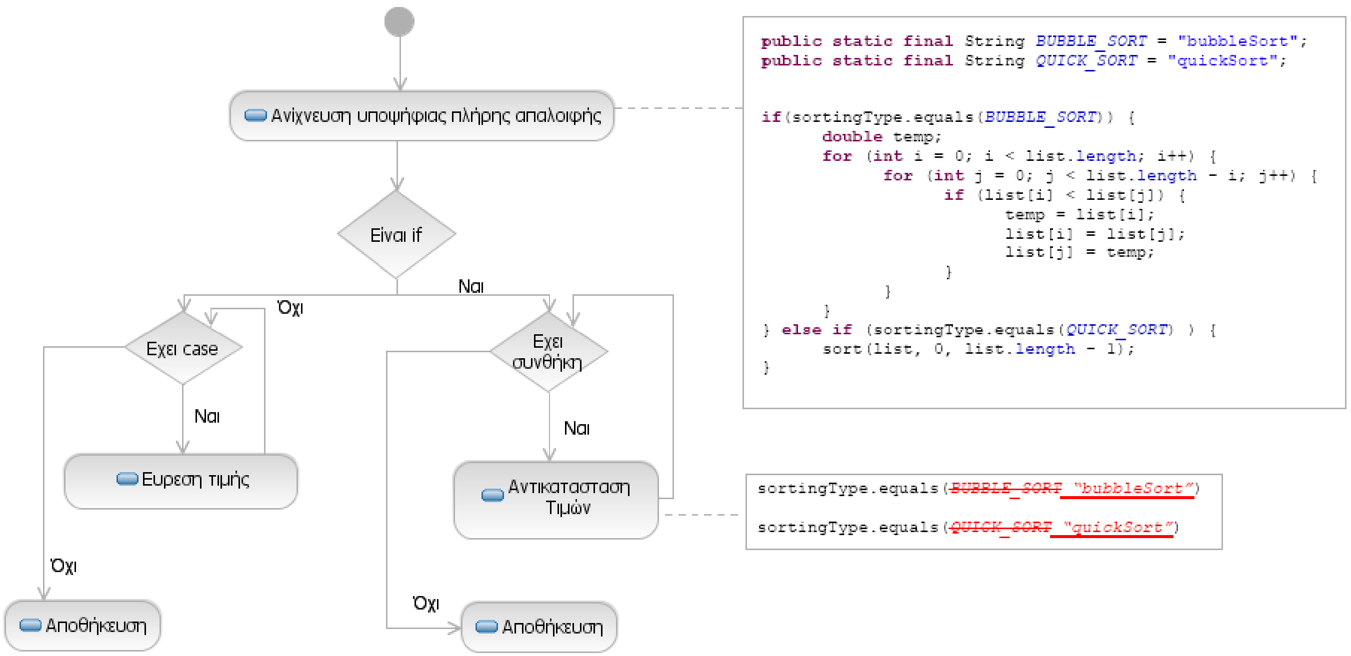
Εικόνα 29 : Ανάκτηση κόμβου συγκεκριμένης κλήσης μεθόδου από το AST δέντρο

5.6.2. Υλοποίηση Πλήρους Αντικατάστασης Υποθετικής Λογικής

Προκειμένου να επιτευχθεί η πλήρης αντικατάσταση της υποθετικής λογικής κατά την εφαρμογή της αναδόμησης στις περιπτώσεις όπου αυτή είναι εφικτή απαιτείται η εκτέλεση των υποθετικών δηλώσεων if και switch για τις συγκεκριμένες τιμές που ορίζουν οι εντολές κλήσεις της μεθόδου. Για το σκοπό αυτό στη φάση της ανίχνευσης παράλληλα με τον εντοπισμό των περιπτώσεων στις οποίες μπορεί να εφαρμοστεί η πλήρης αντικατάσταση συλλέγονται από το σύστημα και αποθηκεύονται αποκλειστικά για τις συγκεκριμένες υποψήφιες περιπτώσεις αναδόμησης, οι εξής πρόσθετες πληροφορίες:

- η σειρά εμφάνισης των παραμέτρων που χρησιμοποιούνται στην υποθετική έκφραση στη λίστα παραμέτρων της μεθόδου,
- οι υποθετικές εκφράσεις με τη μορφή συμβολοσειράς στην περίπτωση δήλωσης if, αφού πρώτα αντικατασταθούν σε αυτές οι τιμές σταθερών ή enum που πιθανώς χρησιμοποιούνται (Εικόνα 30),
- οι εκφράσεις case στην περίπτωση δήλωσης switch αφού αντικατασταθούν σε αυτές οι τιμές σταθερών ή enum που πιθανώς χρησιμοποιούνται,
- οι τιμές των συγκεκριμένων ορισμάτων που αφορούν τις παραμέτρους κάθε κλήσης της μεθόδου.

Η ανίχνευση των περιπτώσεων αυτών και η συλλογή των απαραίτητων πληροφοριών έχει απομονωθεί από την υπόλοιπη ανίχνευση των κριτηρίων σε μια ξεχωριστή κλάση (*CompleteRemovalCandidatesDetector.java*) και εκτελείται μόνο αφότου ολοκληρωθεί προηγουμένως ο έλεγχος των κριτηρίων από την κλάση, της οποίας αποτελεί και τη μοναδική αρμοδιότητα.

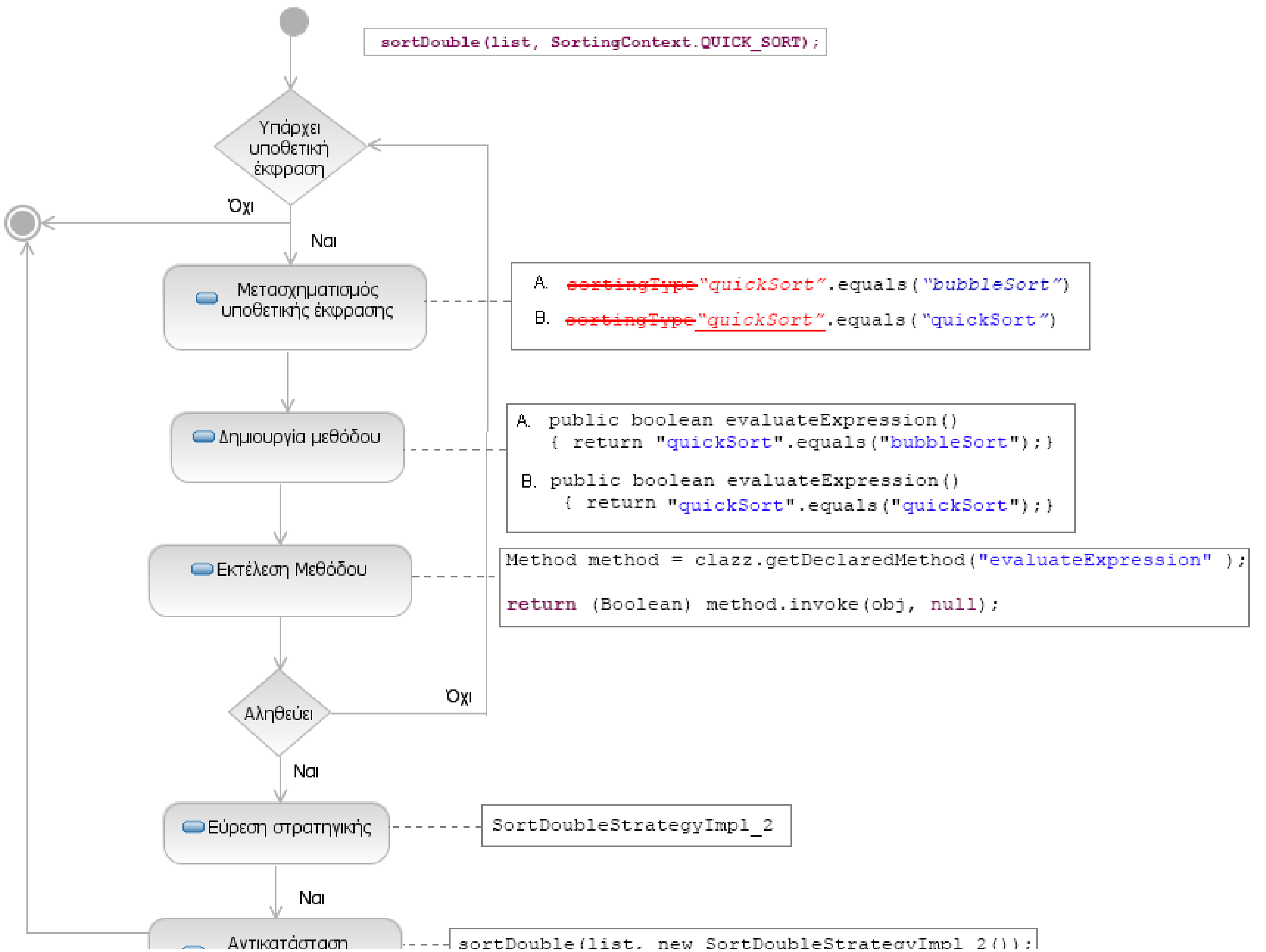


Εικόνα 30 : Μετασχηματισμός και αποθήκευση υποθετικών εκφράσεων για πλήρη αντικατάσταση

Προκειμένου να επιτευχθεί η εκτέλεση των υποθετικών εκφράσεων των υποψήφιων προς πλήρη αντικατάσταση δηλώσεων if και switch, βάσει της προσέγγισης που περιγράφηκε στην ενότητα 5.5.5.5.2, αξιοποιήθηκε η βιβλιοθήκη javaassist (98), η οποία καθιστά εφικτή την διαχείριση και τροποποίηση του bytecode της Java. Κατ' αυτό τον τρόπο επιτρέπει το δυναμικό ορισμό ή την τροποποίηση κλάσεων και μεθόδων σε ένα σύστημα λογισμικού κατά το χρόνο εκτέλεσης του συστήματος, όταν η java virtual machine (JVM) φορτώνει την συγκεκριμένη κλάση. Οι δυνατότητες που παρέχει η συγκεκριμένη βιβλιοθήκη επεξεργασίας καθίστανται ιδιαίτερα εύχρηστες και προσφιλείς για τους προγραμματιστές καθώς εκτός από το API που διατίθεται σε επίπεδο bytecode, παρέχεται και API, το οποίο επιτρέπει την τροποποίηση ενός εκτελέσιμου αρχείου *.class* σε επίπεδο κώδικα χωρίς να είναι απαραίτητη η γνώση των χαρακτηριστικών του bytecode της java.

Σε συνδυασμό με την δυνατότητα reflection που παρέχει η java (99), (100) η λειτουργικότητα της βιβλιοθήκης javaassist μπορεί να χρησιμοποιηθεί προκειμένου να ενσωματωθούν στο εργαλείο αναδόμησης προς τη στρατηγική οι υποθετικές δηλώσεις και να εκτελεστούν ούτως ώστε να βρεθεί η αληθοτιμή η οποία προκύπτει για κάθε σύνολο ορισμάτων των κλήσεων μεθόδων. Κατ' αυτό τον τρόπο εντοπίζεται η υποθετική έκφραση που αληθεύει για κάθε κλήση της μεθόδου και κατ' επακόλουθο ο τύπος της στρατηγικής στον οποίο θα μεταφερθεί ο κώδικας της συγκεκριμένης διακλάδωσης κατά την αναδόμηση. Συνεπώς, καθίσταται εφικτή η τροποποίηση των κλάσεων πελάτη κατά τρόπο ώστε να περνούν ως όρισμα τη συγκεκριμένη κλάση στρατηγική κατά την κλήση της μεθόδου, η οποία τροποποιείται στην κλάση context ούτως ώστε να εκτελεί πολυμορφικά την μέθοδο της στρατηγικής που λαμβάνει ως παράμετρο.

Όπως απεικονίζεται στο παράδειγμα της Εικόνα 31 για κάθε κλήση της προς αναδόμησης μεθόδου που εντοπίζεται στους πελάτες, κατά την εκτέλεση της αναδόμησης εξετάζονται μια-προς-μια οι υποθετικές εκφράσεις. Αρχικά οι τιμές των ορισμάτων της κλήσης μεθόδου χρησιμοποιούνται για το μετασχηματισμό της υποθετικής έκφρασης. Για την αντιστοίχιση των παραμέτρων στις τιμές τους από τη λίστα των ορισμάτων χρησιμοποιήθηκε η σειρά εμφάνισης των μεταβλητών της υποθετικής έκφρασης στη λίστα παραμέτρων της μεθόδου, πληροφορία που συλλέχθηκε κατά την ανίχνευση, καθώς η αντιστοίχηση αυτή δεν είναι δυνατό να επιτευχθεί με χρήση των πληροφοριών του AST δέντρου, εφόσον πρόκειται για εντελώς ξεχωριστές μεταβλητές με διαφορετικά αναγνωριστικά. Εν συνεχεία δημιουργείται μια εντολή επιστροφής της μετασχηματισμένης υποθετικής έκφρασης, η οποία τοποθετείται στο σώμα της μεθόδου που δημιουργείται για τον υπολογισμό της αληθοτιμής της. Ακολούθως η μέθοδος αυτή εκτελείται και στη συνέχεια ανάλογα με το αν αληθεύει ή όχι είτε εντοπίζεται η κλάση στρατηγικής που αντιστοιχεί στη συγκεκριμένη υποθετική έκφραση και αντικαθιστά τα κατάλληλα ορίσματα στην κλήση της μεθόδου, είτε εκτελείται η ίδια διαδικασία για την επόμενη υποθετική έκφραση της δήλωσης, μέχρις ότου βρεθεί η έκφραση που αληθεύει.



Εικόνα 31 : Μετασχηματισμός κλήσεων μεθόδου κατά την πλήρη αντικατάσταση υποθετικής δομής

5.6.3. Παραμετροποίηση Κριτηρίων Ανίχνευσης

Κατά την υλοποίηση της προτεινόμενης μεθοδολογίας προστέθηκε στο εργαλείο αναδόμησης η δυνατότητα ενεργοποίησης και απενεργοποίησης των προτεινόμενων κριτηρίων ανίχνευσης μέσω ενός αρχείου παραμετροποίησης. (configuration file). Η λειτουργικότητα αυτή υλοποιήθηκε αρχικά για τη διευκόλυνση των ελέγχων που διενεργούνταν σχετικά με το εκάστοτε επιλεγμένο σύνολο κριτηρίων καθ' όλη τη διάρκεια της ανάπτυξης, καθώς και για τη διευκόλυνση της αξιολόγησης της προτεινόμενης μεθοδολογίας ως προς την πληρότητα, εφόσον η απενεργοποίηση όλων των κριτηρίων επιτρέπει το γρήγορο εντοπισμό όλων των υποθετικών δηλώσεων με δύο διακλαδώσεις κατ' ελάχιστο, που υπάρχουν στον κώδικα του συστήματος και κατά συνέπεια, και των περιπτώσεων false negative. Τελικά, η λειτουργικότητα παρέμεινε στο εργαλείο ανίχνευσης προκειμένου να διευκολύνει μελλοντικούς ελέγχους κατά την πιθανή αξιολόγηση του εργαλείου ανίχνευσης μέσω της εκτέλεσης σε περισσότερα συστήματα λογισμικού, καθώς και για τη διευκόλυνση της πιθανής επέκτασης της μεθοδολογίας μέσω της προσθήκης νέων κριτηρίων ανίχνευσης.

Το αρχείο παραμετροποίησης που δημιουργήθηκε (criteria.properties) περιλαμβάνει τα μοναδικά αναγνωριστικά των κριτηρίων και την προεπιλεγμένη τους τιμή (true). Για την απενεργοποίηση του κριτηρίου συνίσταται η αλλαγή της τιμής του σε ψευδή (false). Η υλοποίηση της λειτουργικότητας ανάγνωσης του αρχείου για την παραμετροποίηση των κριτηρίων τοποθετήθηκε εξ' ολοκλήρου σε μια καινούργια κλάση που δημιουργήθηκε για το σκοπό αυτό (*CriteriaConfiguration.java*), η οποία δημιουργήθηκε με χρήση του προτύπου σχεδίασης μοναδιαίο. Η κλάση αυτή ουσιαστικά αναλαμβάνει την ανάγνωση του αρχείου παραμετροποίησης και την αποθήκευση των τιμών των κριτηρίων σε μεταβλητές του συστήματος, οι οποίες στη συνέχεια χρησιμοποιούνται για τον έλεγχο της κατάστασης τους. Η επιλογή αυτή καθώς και η επιλογή χρήσης του προτύπου μοναδιαίου στην προκειμένη περίπτωση έγινε για να διασφαλιστεί η ανάγνωση του αρχείου μόνο μια φορά κατά την εκτέλεση της ανίχνευσης καθώς η διαρκής ανάγνωση των τιμών των κριτηρίων από αυτό, θα είχε ως αποτέλεσμα της δραματική αύξηση του απαιτούμενου χρόνου ανίχνευσης.

Τα κριτήρια ανίχνευσης αρχικοποιούνται με lazy initialization[8] όχι κατά την εκκίνηση του Eclipse, αλλά κατά την πρώτη ανίχνευση προτάσεων αναδόμησης στο σύστημα λογισμικού που θα εκτελεστεί το εργαλείο. Επιπλέον, οι αντίστοιχες μεταβλητές του συστήματος αρχικοποιούνται εξ' ορισμού με την τιμή αληθές ούτως ώστε αν το αρχείο παραμετροποίησης δε βρεθεί, αν το αναγνωριστικό ή η τιμή ενός κριτηρίου δε βρεθεί στο αρχείο, ή προκύψει οποιοδήποτε σφάλμα κατά τη φάση της παραμετροποίησης, η διαδικασία ανίχνευσης να μην τερματιστεί αλλά να πραγματοποιηθεί με το προεπιλεγμένο σύνολο κριτηρίων. Η επιλογή αυτή αιτιολογείται λόγω του ότι, όπως προαναφέρθηκε, η λειτουργικότητα δεν αναπτύχθηκε με σκοπό την παροχή δυνατότητας παραμετροποίησης της ανίχνευσης αλλά προκειμένου να διευκολυνθεί η διαδικασία της ανάπτυξης της μεθοδολογίας και του ελέγχου.

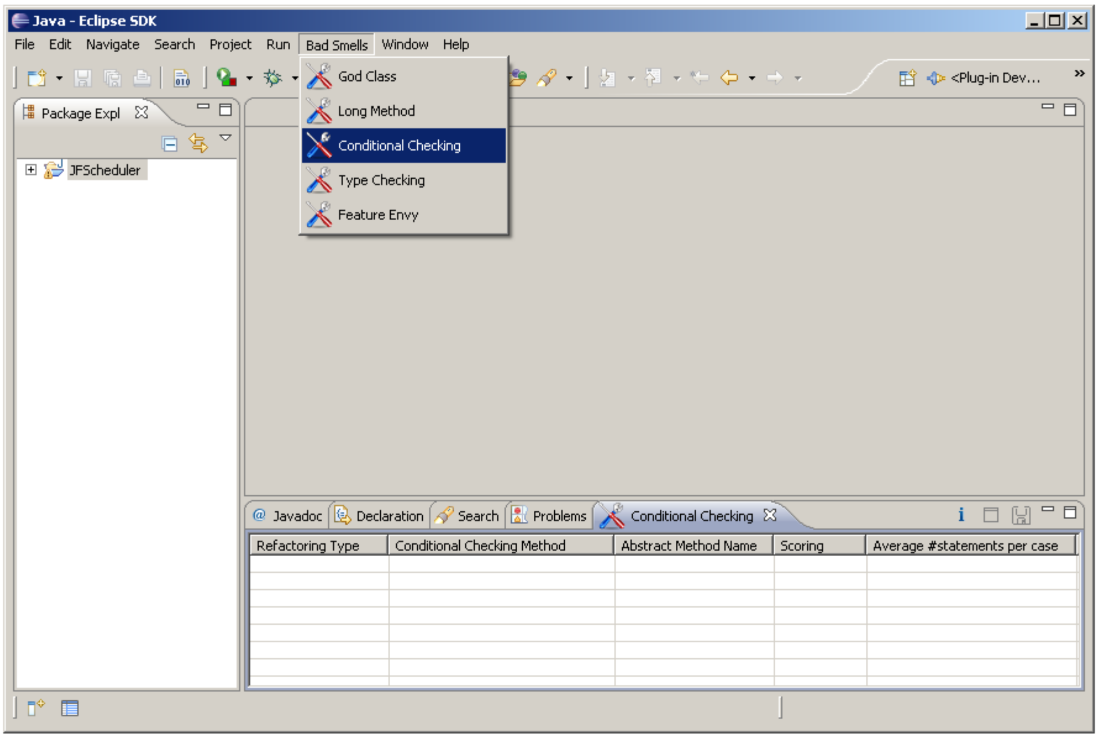
5.6.4. Παρουσίαση Εργαλείου Αναδόμησης

Η προτεινόμενη μεθοδολογία ενσωματώθηκε στο plug-in JDeodorant του Eclipse. Οι επιλογές αναδόμησης που παρέχει το JDeodorant τίθενται στη διάθεση του χρήστη μέσω του μενού «Bad Smells» που εμφανίζεται γραμμή επιλογών του συγκεκριμένου περιβάλλοντος ανάπτυξης, μετά την εγκατάσταση του σχετικού plug-in (οδηγίες σχετικά με την εγκατάσταση παρέχονται στο

[8] Lazy initialization : Η αναβολή της αρχικοποίησης μιας μεταβλητής έως ότου να είναι απολύτως απαραίτητη. (119) (120).

Παράρτημα Ε: Οδηγίες Εγκατάστασης του Plug-in).

Το μενού «Bad Smells» περιέχει ως ξεχωριστές επιλογές τα σχεδιαστικά προβλήματα, τα οποία επιλύονται από τις αναδομήσεις που υποστηρίζει αυτόματα το JDeodorant. Η επιλογή που δημιουργήθηκε για την αναδόμηση προς τη στρατηγική ονομάστηκε *υποθετικός έλεγχος (Conditional Checking)* (Εικόνα 32). Μέσω της επιλογής *Conditional Checking* ενεργοποιείται και εμφανίζεται στο παράθυρο *(workbench)* του Eclipse, η αντίστοιχη όψη *(view/perspective)*, μέσω της οποίας παρέχονται οι επιλογές ανίχνευσης των προτάσεων αναδόμησης προς τη στρατηγική καθώς και αυτόματης εκτέλεσης των προτεινόμενων αναδομήσεων (Εικόνα 32). Επιπροσθέτως, μέσω της όψης *(perspective) Conditional Checking* επιτυγχάνεται και η παρουσίαση των προτάσεων αναδόμησης προς τη στρατηγική στο χρήστη.



Εικόνα 32 : Perspective Conditional Checking με επιλογές ανίχνευσης και εκτέλεσης των προτάσεων αναδόμησης προς τη στρατηγική

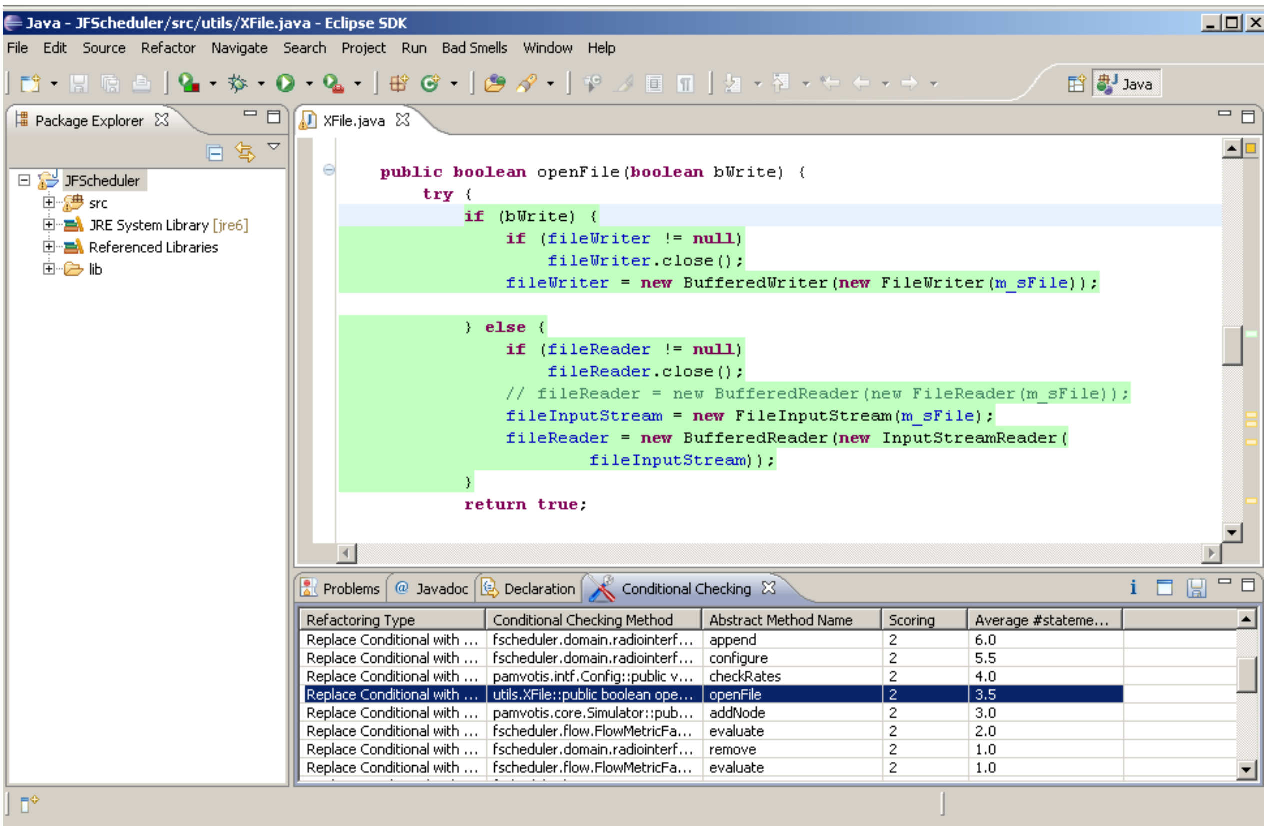
Οι προτάσεις που προκύπτουν ως αποτέλεσμα της ανίχνευσης, εμφανίζονται στο χρήστη υπό τη μορφή πίνακα, ταξινομημένες αρχικά ως προς το βάρος που τους αποδόθηκε (στήλη scoring) και εν συνεχεία ανάλογα με τον μέσο αριθμό δηλώσεων που εμφανίζονται στο σώμα της υποθετικής δήλωσης (στήλη average #stmts/case). Επιπλέον, εμφανίζεται για κάθε προτεινόμενη αναδόμηση το πλήρες όνομα της κλάσης στην οποία περιέχεται συνοδευόμενο από την υπογραφή της μεθόδου (στήλη *Conditional Checking Method*) και το όνομα της μεθόδου με την υποθετική λογική (στήλη *Abstract Method Name*). Η στήλη *Refactoring Type* που εμφανίζει τον τύπο της μεθόδου, προστέθηκε για λόγους συμβατότητας με τις υπόλοιπες όψεις του JDeodorant και έχει πάντα την τιμή «Αναδόμηση προς τη Στρατηγική» (Refactoring to Strategy).

Η όψη *Conditional Checking* παρέχει στο χρήστη τη δυνατότητα επισκόπησης του προτεινόμενου προς αναδόμηση σημείου κάνοντας διπλό κλικ στην γραμμή του πίνακα που περιέχει την πρόταση αναδόμησης που τον ενδιαφέρει. Στην περίπτωση αυτή εμφανίζεται στο χρήστη στον επεξεργαστή του Eclipse η σχετική κλάση, όπου η υποθετική δήλωση προς αναδόμηση επισημαίνεται με διαφορετικό από τον υπόλοιπο κώδικα χρώμα, για τον ευκολότερο εντοπισμό της (Εικόνα 33).

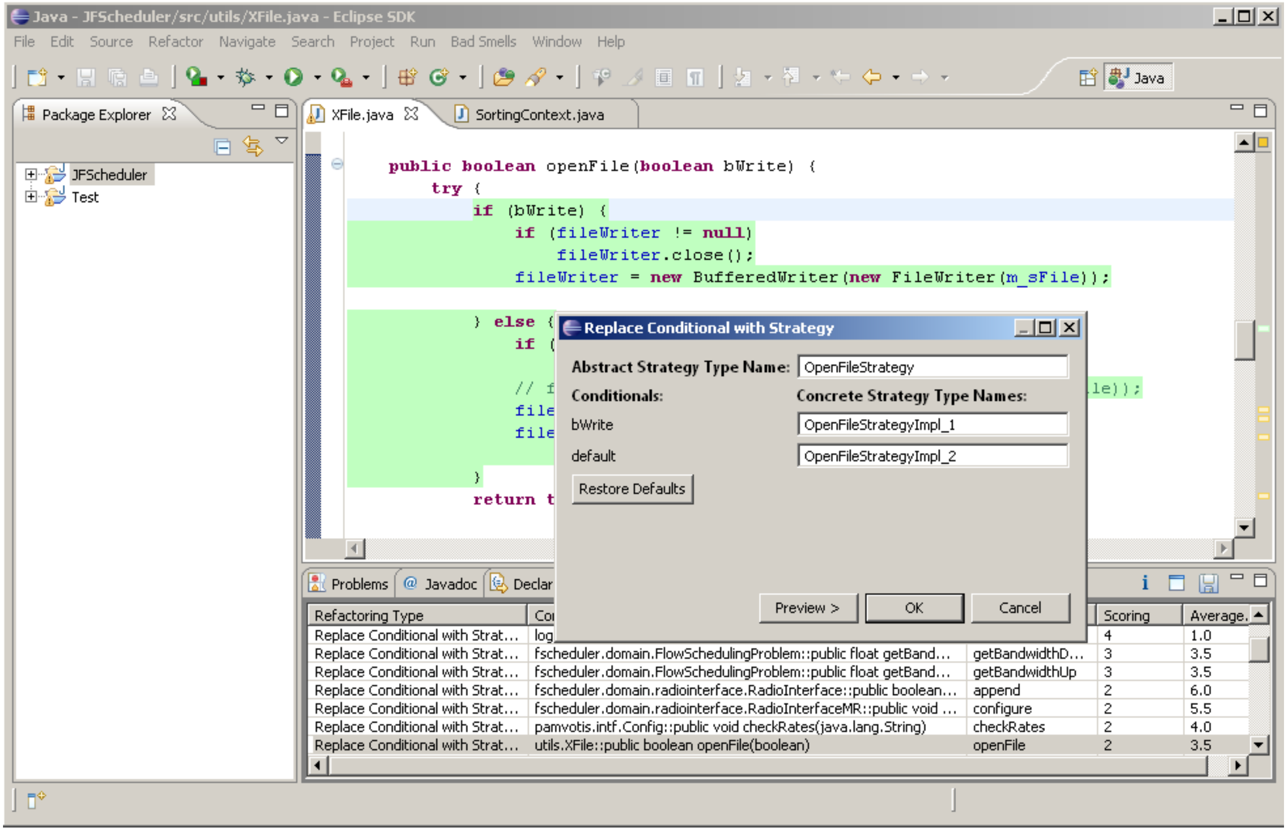
Η εκτέλεση της αναδόμησης προς τη στρατηγική επιτυγχάνεται μέσω της αλληλεπίδρασης με το χρήστη. Εφόσον ο χρήστης θεωρήσει χρήσιμη την προτεινόμενη αναδόμηση, βάσει εννοιολογικών όσο και βάσει ποιοτικών κριτηρίων, μπορεί να προχωρήσει στην αυτόματη εκτέλεση της αναδόμησης προς τη στρατηγική. Η εκτέλεση επιτυγχάνεται επιλέγοντας την επιθυμητή πρόταση και κάνοντας κλικ στο κουμπί εφαρμογής που ενεργοποιείται στην όψη Conditional Checking, έπειτα από την επιλογή κάποιου από τα διαθέσιμα αποτελέσματα. Η ενέργεια αυτή έχει ως αποτέλεσμα την εμφάνιση ενός οδηγού *(wizard)* που σχετίζεται με την εκτέλεση της αναδόμησης. Η λειτουργικότητα του συγκεκριμένου οδηγού περιλαμβάνει την παροχή στο χρήστη της δυνατότητας

- επιθεώρησης των προεπιλεγμένων ονομάτων που θα χρησιμοποιηθούν κατά τη δημιουργία των κλάσεων στρατηγικής και τροποποίηση τους (Εικόνα 34)
- προεπισκόπησης του μετασχηματισμού που θα διενεργηθεί από την εκτέλεση της αναδόμησης και αποδοχής ή απόρριψης του (Εικόνα 35).

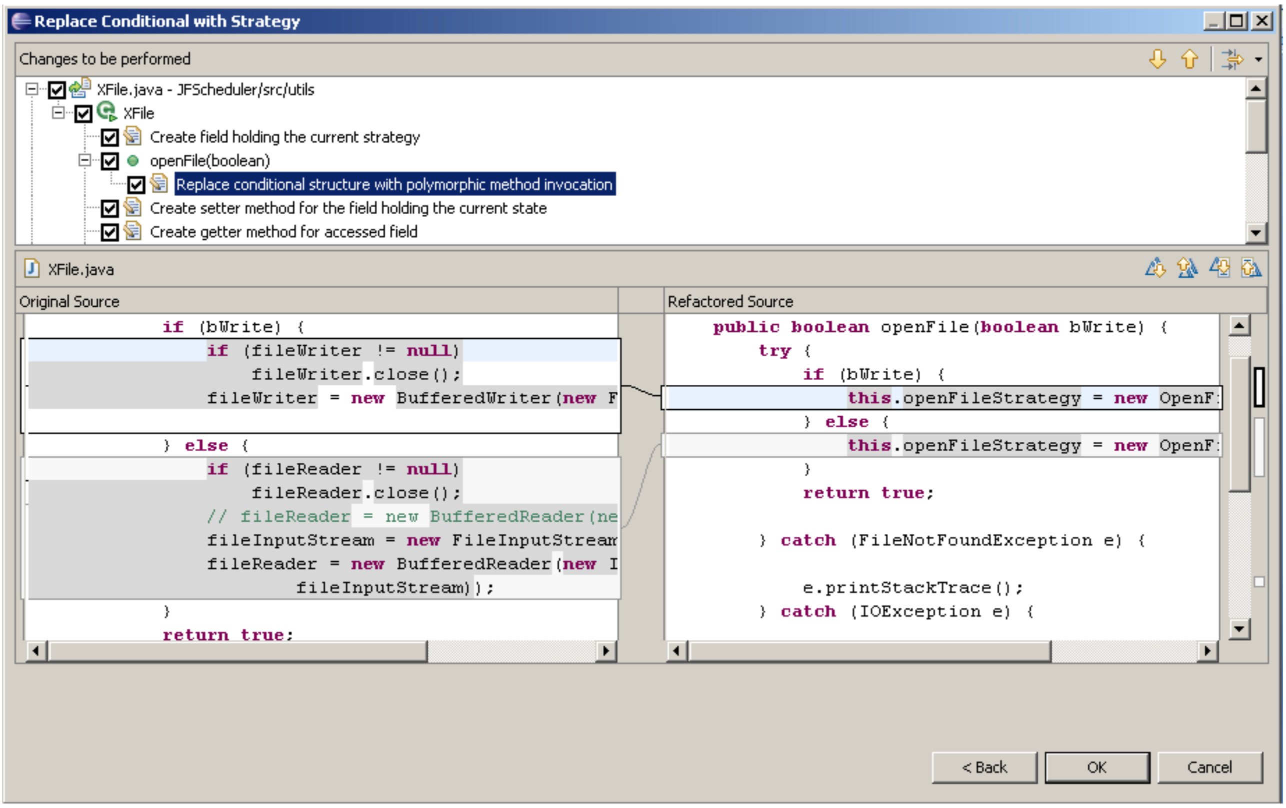
Στην επόμενη ενότητα παρουσιάζονται αναλυτικά οι μετασχηματισμοί που επιτελούνται με την εκτέλεση της αναδόμησης προς τη στρατηγική.



Εικόνα 33 : Αποτελέσματα προτάσεων αναδόμησης προς τη στρατηγική και επισκόπηση



Εικόνα 34 : Wizard για την προεπισκόπηση και τροποποίηση από τον χρήστη των προεπιλεγμένων ονομάτων των κλάσεων που θα δημιουργηθούν



Εικόνα 35 : Wizard για την προεπισκόπηση και αποδοχή του προτεινόμενου μετασχηματισμού

5.6.5. Παρουσίαση Εκτελούμενης Αναδόμησης

Στην ενότητα αυτή επιδεικνύεται η εφαρμογή της αναδόμησης μερικής και πλήρους αντικατάστασης της υποθετικής λογικής. Η επίδειξη της μερικής αντικατάστασης υποθετικής λογικής πραγματοποιείται μέσω ενός αντιπροσωπευτικού παραδείγματος πηγαίου κώδικα που κατασκευάστηκε για το σκοπό αυτό, ενώ για την επίδειξη πλήρους αντικατάστασης επιλέχθηκε ένα απόσπασμα κώδικα από το σύστημα λογισμικού JFScheduler (101) προκειμένου να καταστεί εμφανές η χρησιμότητα της περίπτωσης αυτής σε πραγματικά συστήματα λογισμικού και να αναδειχθεί η πρακτική αξία της αναδόμησης στην πράξη.

5.6.5.1. Παρουσίαση Μερικής Αντικατάστασης Υποθετικής Λογικής

Αναφορικά με το παράδειγμα μερικής αντικατάστασης υποθετικής λογικής, η οποία βρίσκει εφαρμογή συχνότερα στην πράξη απ’ ότι η περίπτωση ολικής αντικατάστασης, το απόσπασμα του κώδικα που παρατίθεται είναι ιδιαίτερα απλοποιημένο καθώς δεν αποτελεί τμήμα κάποιου πραγματικού συστήματος. Η επιλογή ενός πραγματικού συστήματος για την επίδειξη του προτεινόμενου μετασχηματισμού θα είχε το πλεονέκτημα πιθανής αντιμετώπισης δυσκολιών, οι οποίες θα αποδείκνυαν και έμπρακτα τις δυνατότητες και τη χρησιμότητα του εργαλείου που αναπτύχθηκε. Ωστόσο, η παράθεση αποσπάσματος κώδικα ενός πραγματικού συστήματος λογισμικού ενδέχεται να καταστήσει δυσκολότερη την κατανόηση της λειτουργικότητας του. Συνεπώς θεωρήθηκε βέλτιστη η παράθεση ενός απλοποιημένου παραδείγματος προκειμένου να γίνει κατανοητός ο τρόπος με τον οποίο εκτελείται η αναδόμηση προς τη στρατηγική και να αποφευχθεί παράλληλα η άσκοπη εισαγωγή πρόσθετης δυσκολίας.

Στο συγκεκριμένο παράδειγμα, η υποψήφια προς αναδόμηση κλάση SortingContext (Εικόνα 36α) διαθέτει μεταξύ άλλων, τη μέθοδο sortDouble, η οποία χρησιμοποιείται για την ταξινόμηση των στοιχείων ενός πίνακα δεκαδικών αριθμών, τον οποίο δέχεται ως παράμετρο. Η ταξινόμηση των στοιχείων του πίνακα μπορεί να επιτευχθεί είτε με χρήση του αλγόριθμου *ταξινόμησης φυσαλίδας (bubblesort),* είτε με τον αλγόριθμο *γρήγορης ταξινόμησης (quicksort).* Η επιλογή του αλγορίθμου που θα χρησιμοποιηθεί για την ταξινόμηση πραγματοποιείται από μια υποθετική δήλωση στο σώμα της μεθόδου με βάσει την τιμή της παραμέτρου sortingType της μεθόδου, παρόλο που η ίδια λειτουργικότητα θα μπορούσε εναλλακτικά να είχε επιτευχθεί με τη χρήση του προτύπου στρατηγική.

Η εκτέλεση της αναδόμησης προς τη στρατηγική έχει ως αποτέλεσμα τη δημιουργία μιας συσχέτισης ανάμεσα στην κλάση SortingContext και στην αφηρημένη κλάση SortDoubleStrategy, μέσω της δημιουργίας ενός πεδίου στρατηγικής στην κλάση context. Επιπλέον, μέσω της αναδόμησης δημιουργείται για το πεδίο *στρατηγικής* και η αντίστοιχη setter μέθοδος.

Άμεση συνέπεια της αναδόμησης αποτελεί η χρήση διαφορετικών κλάσεων στρατηγικής για την ενθυλάκωση των εναλλακτικών παραλλαγών ταξινόμησης. Για το σκοπό αυτό, ο κώδικας που περιλαμβάνεται στο εσωτερικό των διακλαδώσεων αντικαθίσταται από εντολές δημιουργίας των κατάλληλων αντικειμένων στρατηγικής, τα οποία εκχωρούνται στο σχετικό πεδίο της κλάσης context. Επιπλέον, προκειμένου να επιτευχθεί η εκτέλεση του επιθυμητού αλγορίθμου ταξινόμησης προστίθεται στον τροποποιημένο κώδικα η πολυμορφική κλήση της μεθόδου που δημιουργήθηκε από την αντιγραφή του κώδικα της διακλάδωσης στην αντίστοιχη υποκλάση. Στην Εικόνα 36β παρουσιάζεται η τροποποιημένη κλάση Context μετά την εφαρμογή της αναδόμησης.

<table>
<tr>
<td>

```
public class SortingContext {


  public void sortDouble(double[] list, String sortingType) {
    if(sortingType.equals("bubbleSort")) {
        double temp;
        for (int i = 0; i < list.length; i++) {
            for (int j = 0; j < list.length - i; j++) {
                if (list[i] < list[j]) {
                    temp = list[i];
                    list[i] = list[j];
                    list[j] = temp;
                }
            }
        }
    } else if (sortingType.equals("quickSort")){
        sort(list, 0,list.length -1);
    }
}

private void sort(double[] a, int left, int right) {
    if (right <= left)
        return;
    int i = part(a, left, right);
    sort(a, left, i - 1);
    sort(a, i + 1, right);
}

private int part(double[] a, int left, int right) {
    int i = left;
    int j = right;
    while (true) {
        while (a[i] < a[right])
            i++;
        while (smaller(a[right], a[--j]))
            if (j == left)
                break;
            if (i >= j)
                break;
        swap(a, i, j);
    }
    swap(a, i, right);
    return i;
}
```

</td>
<td>

```
public class SortingContext {

  private SortDoubleStrategy sortDoubleStrategy;

  public void sortDouble(double[] list, String sortingType) {
    if(sortingType.equals("bubbleSort")) {
        this.sortDoubleStrategy = new SortDoubleStrategyImpl_1();
    } else if (sortingType.equals("quickSort"))
    {
        this.sortDoubleStrategy = new SortDoubleStrategyImpl_2();
    }
    sortDoubleStrategy.sortDouble(list, this);
}

private void sort(double[] a, int left, int right) {
    if (right <= left)
        return;
    int i = part(a, left, right);
    sort(a, left, i - 1);
    sort(a, i + 1, right);
}

private int part(double[] a, int left, int right)
{
    int i = left;
    int j = right;
    while (true) {
        while (a[i] < a[right])
            i++;
        while (smaller(a[right], a[--j]))
            if (j == left)
                break;
            if (i >= j)
                break;
        swap(a, i, j);
    }
    swap(a, i, right);
    return i;
}

private boolean smaller(double x, double y)
{
    return (x < y);
```

</td>
</tr>
</table>

private boolean smaller(double x, double y) {
    return (x < y);
}
 	
private void swap(double[] a, int i, int j) {
    double swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
}
	}
	
	private void swap(double[] a, int i, int j) {
	    double swap = a[i];
	    a[i] = a[j];
	    a[j] = swap;
	}
	
	public void setSortDoubleStrategy(SortDoubleStrategy sortDoubleStrategy) {
	    this.sortDoubleStrategy = sortDoubleStrategy;
	}
	}

Α. Η κλάση SortingContext πρίν την αναδόμηση	**Β. Η κλάση SortingContext μετά την αναδόμηση**

Εικόνα 36 : Η υποψήφια προς αναδόμηση κλάση SortingContext πρίν και μετά την αναδόμηση

Εν συνεχεία δημιουργείται η αφηρημένη κλάση στρατηγική, η οποία χρησιμοποιείται για την ενθυλάκωση της συμπεριφοράς των παραλλαγών του αλγορίθμου ταξινόμησης. Όπως φαίνεται στην Εικόνα 37 στο εσωτερικό της αφηρημένης κλάσης δημιουργείται μόνο η αφηρημένη μέθοδος sortDouble, με βάση το όνομα της μεθόδου στην οποία εκτελείται η αναδόμηση. Η μέθοδος αυτή λαμβάνει σαν παράμετρο την παράμετρο list, εφόσον αυτή χρησιμοποιείται στο εσωτερικό της υποθετικής λογικής, καθώς και την κλάση context, εφόσον οι εντολές της υποθετικής λογικής περιλαμβάνουν κλήση στις μεθόδους της.

SortDoubleStrategy.java

```
package strategyrefactoringforrefactoringafter;


public abstract class SortDoubleStrategy {
    public abstract void sortDouble(double[] list, SortingContext sortingContext);
}
```

Εικόνα 37 : Η αφηρημένη κλάση στρατηγική που δημιουργείται από την αναδόμηση

Εφόσον η υποθετική δομή αποτελείται από 2 διακλαδώσεις η αναδόμηση έχει ως αποτέλεσμα τη δημιουργία δύο διαφορετικών υποκλάσεων στρατηγικής, Εικόνα 38 και Εικόνα 39. Στις κλάσεις αυτές υπερφορτώνεται η μέθοδος που δημιουργήθηκε στην

SortDoubleStrategyImpl_2.java

```java
package strategyrefactoringforrefactoringafter;


public class SortDoubleStrategyImpl_2 extends SortDoubleStrategy {
    public void sortDouble(double[] list, SortingContext sortingContext) {
        double temp;
        for (int i = 0; i < list.length; i++) {
            for (int j = 0; j < list.length - i; j++) {
                if (list[i] < list[j]) {
                    temp = list[i];
                    list[i] = list[j];
                    list[j] = temp;
                }
            }
        }
    }
}
```

Εικόνα 38 : Η συγκεκριμένη υποκλάση που δημιουργείται με βάση τον κώδικα της πρώτης διακλάδωσης

αφηρημένη κλάση SortDoubleStrategy, και αντιγράφεται ο κώδικας από τις διαφορετικές διακλαδώσεις, ούτως ώστε καθεμία να υλοποιεί και μια διαφορετική παραλλαγή του αλγορίθμου ταξινόμησης. Κατά την εκτέλεση της αναδόμησης ο χρήστης έχει τη δυνατότητα, εφόσον το επιθυμεί να τροποποιήσει τα προεπιλεγμένα ονόματα των κλάσεων που παράγονται σύμφωνα με τη λογική που περιγράφηκε στην ενότητα 5.5.5.5.1. Ακολούθως παρουσιάζεται η εφαρμογή της αναδόμησης για την περίπτωση πλήρους αντικατάστασης της υποθετικής λογικής.

SortDoubleStrategyImpl_1.java

```java
package strategyrefactoringforrefactoringafter;


public class SortDoubleStrategyImpl_1 extends SortDoubleStrategy {
    public void sortDouble(double[] list, SortingContext sortingContext) {
        sortingContext.sort(list, 0, list.length - 1);
    }
}
```

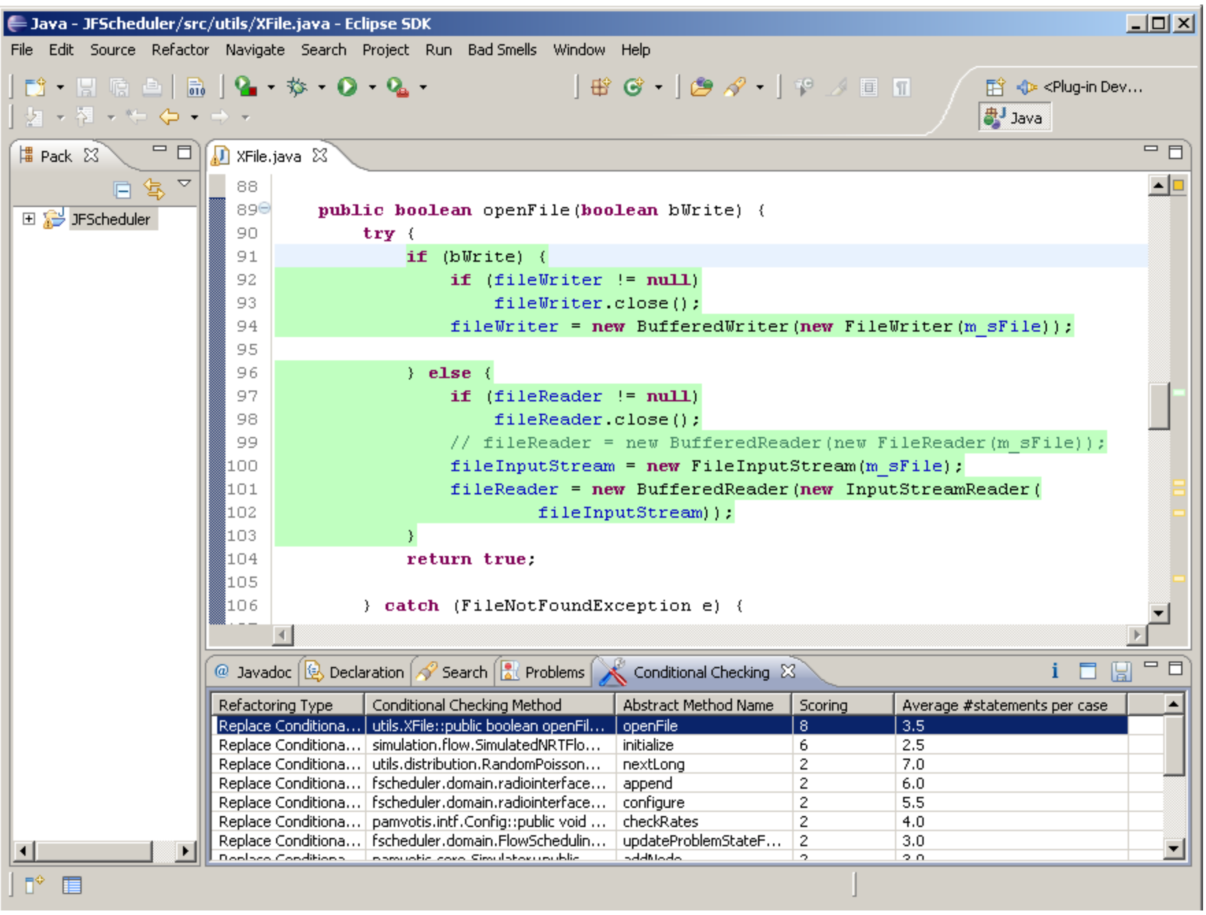
Εικόνα 39 : Η συγκεκριμένη υποκλάση που δημιουργείται με βάση τον κώδικα της δεύτερης διακλάδωσης

5.6.5.2. Παρουσίαση Πλήρους Αντικατάστασης Υποθετικής Λογικής

Για τους σκοπούς της παρουσίασης της περίπτωσης πλήρους αντικατάστασης της υποθετικής λογικής επιλέχθηκε, όπως αναφέρθηκε και νωρίτερα, ένα απλό απόσπασμα κώδικα από ένα πραγματικό σύστημα λογισμικού, το JFScheduler (101). Η παράθεση του παραδείγματος αυτού αποσκοπεί να καταστήσει εμφανή τη διαφοροποίηση του παραγόμενου κώδικα και των μετασχηματισμών που εφαρμόζονται στην ολική αναδόμηση προς τη στρατηγική καθώς και τη δυνατότητα εφαρμογής της στην πράξη σε πραγματικά συστήματα λογισμικού.

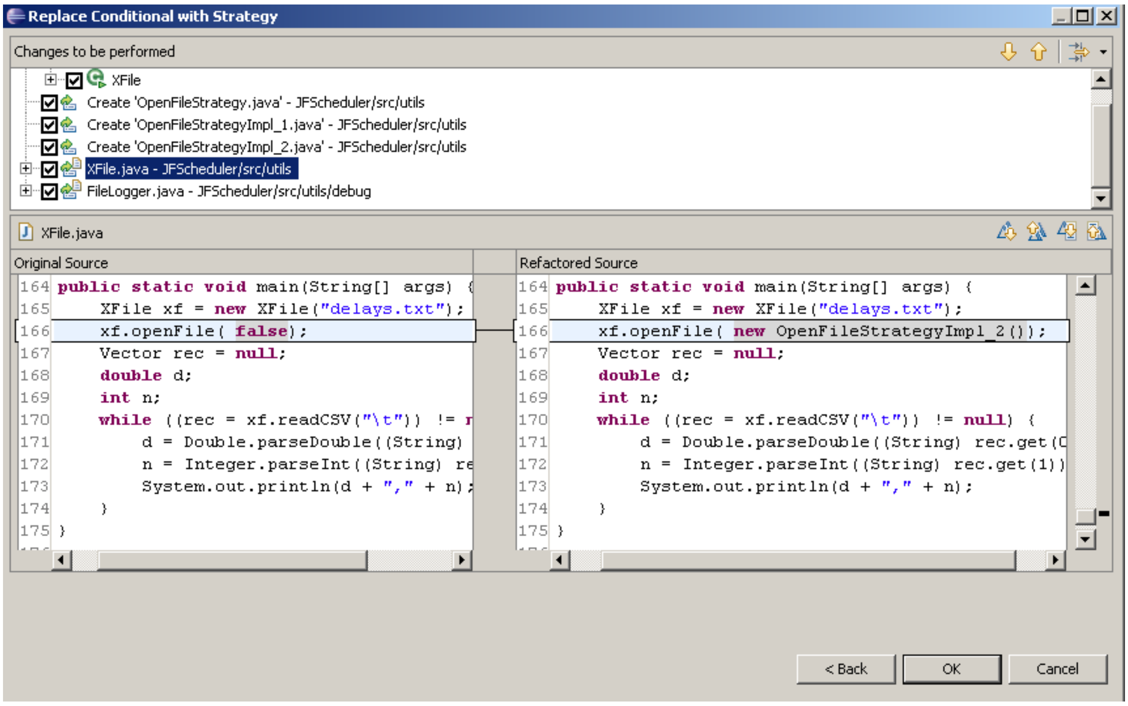
Στην προς αναδόμηση μέθοδο που παρουσιάζεται στην Εικόνα 40 ανάλογα με την τιμή της παραμέτρου επιλέγεται ο τρόπος ανοίγματος ενός αρχείου, για εγγραφή ή για ανάγνωση. Η επιλογή αυτή θα μπορούσε εναλλακτικά να πραγματοποιηθεί επιλέγοντας την εκτέλεση της κατάλληλης στρατηγικής ανοίγματος ενός αρχείου.

Στο σημείο αυτό αξίζει να τονιστεί η σημασία απόδοσης των βαρών στις προτάσεις αναδόμησης. Στην Εικόνα 40 απεικονίζεται και το βάρος που έχει αποδοθεί από το σύστημα στην πρόταση αναδόμησης της μεθόδου (openFile). Βάσει του Πίνακας 11, ο οποίος περιλαμβάνει τις αντιστοιχίες βαρών και κριτηρίων εξάγεται το συμπέρασμα ότι στη συγκεκριμένη περίπτωση είναι εφικτή η εφαρμογή πλήρους αντικατάστασης της υποθετικής λογικής, γεγονός που μπορεί εύκολα να διαπιστωθεί μέσω της προεπισκόπησης της προτεινόμενης αναδόμησης.

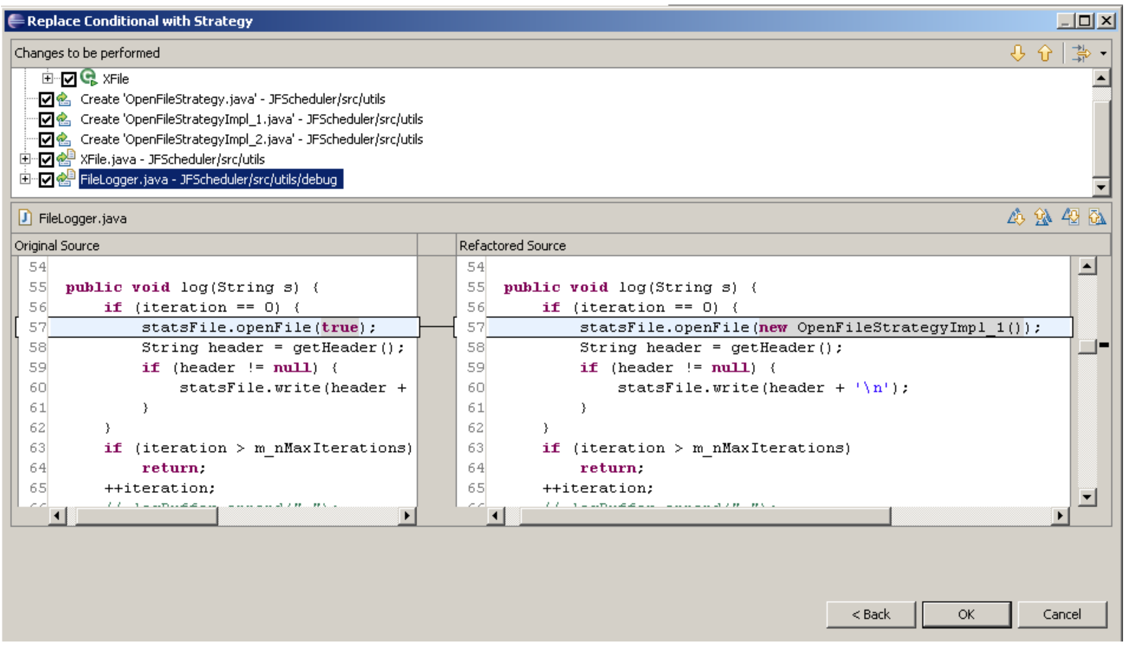


Εικόνα 40 : Υποψήφια αναδόμηση πλήρους αντικατάστασης υποθετικής λογικής

Πράγματι όπως απεικονίζεται στην Εικόνα 41 και στην Εικόνα 42, στις κλήσεις των μεθόδων που περιλαμβάνονται στις κλάσεις πελάτη XFile.java και FileLogger.java, η τιμή του ορίσματος που περνιέται είναι γνωστή κατά το χρόνο μεταγλώττισης καθώς αποτελεί συγκεκριμένη αληθοτιμή (boolean literal). Στην περίπτωση της κλήσης που περιλαμβάνεται στην κλάση XFile.java (Εικόνα 41) η τιμή της παραμέτρου έχει ως αποτέλεσμα την εκτέλεση του κώδικα που περιλαμβάνεται στη δεύτερη διακλάδωση της υποθετικής δήλωσης if. Γι' αυτό το λόγο ο τύπος του αντικειμένου στρατηγικής που δημιουργείται και περνιέται ως παράμετρος κατά την κλήση της μεθόδου έπειτα από την αναδόμηση έχει το προεπιλεγμένο όνομα OpenFileStrategyImpl_2 (Εικόνα 44). Κατ' αντιστοιχία, προσδιορίζεται και ο τύπος εκχωρούμενης στρατηγικής στην κλήση της κλάσης FileLogger.java (Εικόνα 42), όπου επειδή η τιμή του ορίσματος (true) έχει ως αποτέλεσμα την εκτέλεση της πρώτης διακλάδωσης, ο προεπιλεγμένος τύπος προσδιορίζεται σε OpenFileStrategyImpl_1.



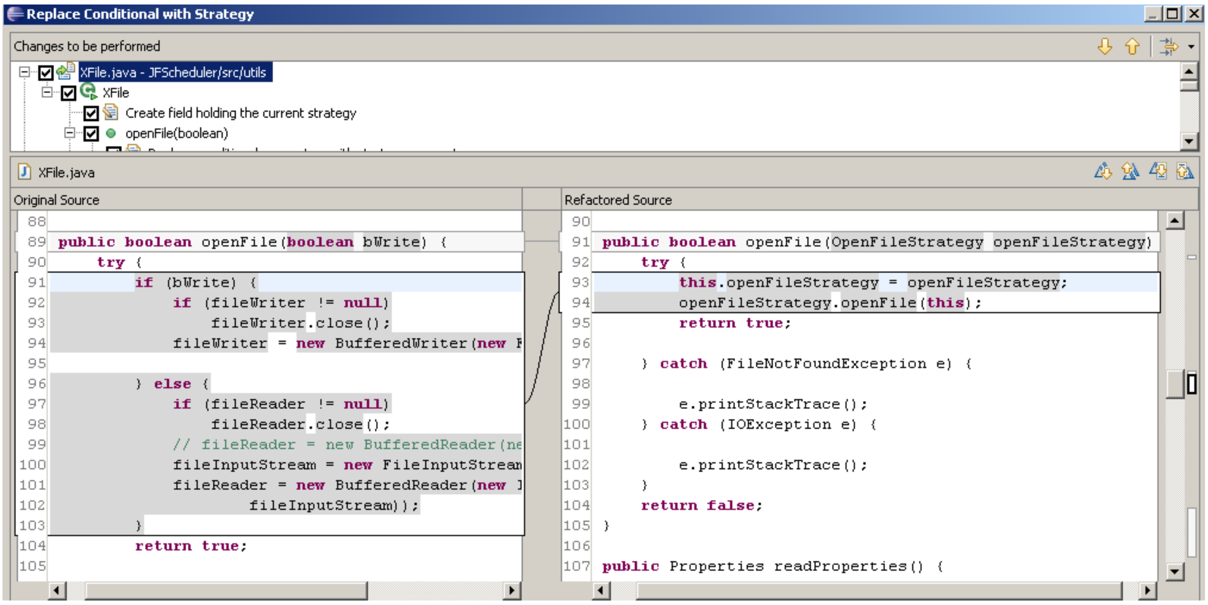
Εικόνα 41 : Πέρασμα συγκεκριμένου αντικειμένου στρατηγικής κατά την αναδόμηση πλήρους αντικατάστασης



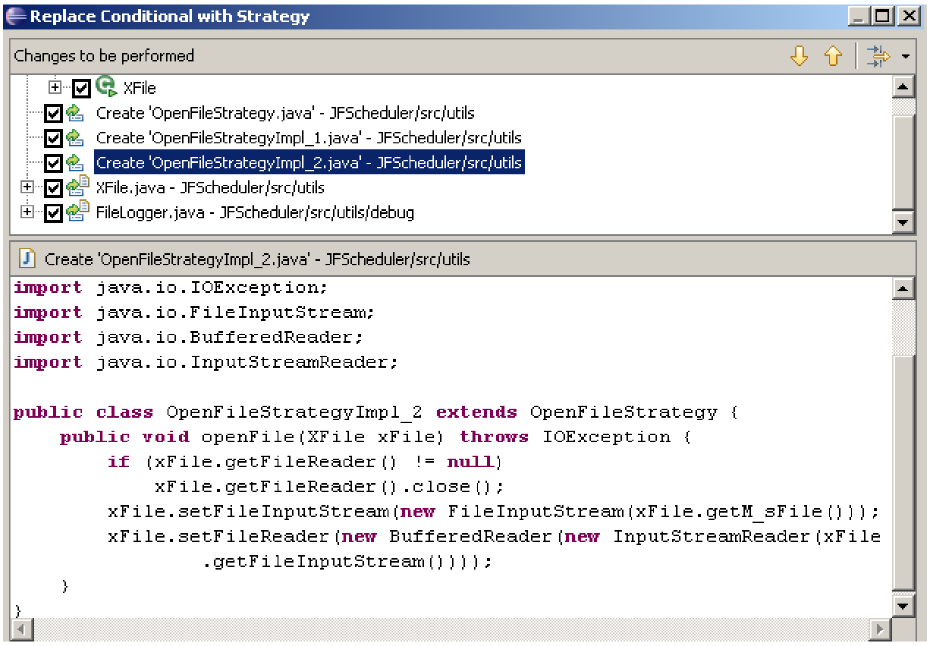
Εικόνα 42 : Πέρασμα διαφορετικού αντικειμένου στρατηγικής κατά την αναδόμηση πλήρους αντικατάστασης

Εν συνεχεία, τροποποιείται η λίστα παραμέτρων της μεθόδου με την υποθετική λογική, ούτως ώστε να δέχεται ως παράμετρο αντικείμενα τύπου στρατηγικής. Συνεπώς, ως τύπος της παραμέτρου προσδιορίζεται ο τύπος της αφηρημένης κλάσης στρατηγικής. Επιπλέον, ως αποτέλεσμα εφαρμογής των μετασχηματισμών της αναδόμησης απαλείφεται πλήρως η υποθετική λογική (Εικόνα 43). Στη θέση της υποθετικής λογικής τοποθετούνται η εντολή εκχώρησης της παραμέτρου στρατηγικής στο αντίστοιχο πεδίο της κλάσης context καθώς και η πολυμορφική κλήση της μεθόδου που υλοποιεί τον αλγόριθμο ανοίγματος αρχείου. Στο σημείο αυτό αξίζει να αναφερθεί πώς η δημιουργία τόσο της αφηρημένης όσο και των συγκεκριμένων υποκλάσεων στρατηγικής δεν παρουσιάζει καμία διαφορά από τη γενική περίπτωση αναδόμησης (Εικόνα 44).

Με την ολοκλήρωση επίδειξης της υλοποίησης της προτεινόμενης μεθοδολογίας, κρίνεται αναγκαίο να παρουσιαστούν οι περιορισμοί της, οι οποίοι περιγράφονται αναλυτικά στην ενότητα που ακολουθεί.



Εικόνα 43: Παράδειγμα μετασχηματισμού της κλάσης context στην περίπτωση πλήρους αντικατάστασης υποθετικής λογικής



Εικόνα 44 : Παράδειγμα συγκεκριμένης υποκλάσης στρατηγικής που δημιουργείται κατά την αναδόμηση πλήρους αντικατάστασης υποθετικής λογικής

5.7. Συμπεράσματα και Περιορισμοί

Συνοψίζοντας την παρουσίαση της προτεινόμενης μεθοδολογίας καθίσταται αναγκαίο αφένος να γίνει μια συγκεντρωτική αναφορά στις διαφορές μεταξύ των δύο μεθοδολογιών ώστε να καταστούν σαφείς, και αφετέρου να επισημανθούν οι περιορισμοί.στους οποίους υπάγεται η προσέγγιση αυτή, βάσει των οποίων εξάγονται χρήσιμα συμπεράσματα.

Ως προς τη διαφοροποίηση των δύο μεθοδολογιών αναφορικά με την ανίχνευση των προτάσεων αναδόμησης προς τη στρατηγική, θα πρέπει να αποσαφηνιστεί ότι η υπάρχουσα μεθοδολογία του JDeodorant βασίζεται σε κριτήρια που αφορούν αποκλειστικά τις συνθήκες των υποθετικών εκφράσεων. Αντιθέτως, η μεθοδολογία που προτείνεται, επεκτείνεται και σε κριτήρια που αφορούν τη μέθοδο που φιλοξενεί την υποθετική λογική, καθώς και τις διακλαδώσεις των υποθετικών δηλώσεων. Επιπλέον οι έλεγχοι που εκτελούνται στην ανίχνευση type checking για την περίπτωση αναδόμησης με την κατάσταση ή τη στρατηγική, περιορίζονται στα πλαίσια της μεθόδου με την υποθετική δήλωση (intraprocedural έλεγχος). Αντιθέτως, η προτεινόμενη μεθοδολογία δεν περιορίζει τους εκτελούμενους ελέγχους μόνο μέσα στην ίδια τη μέθοδο, αλλά ακολουθεί τη ροή ελέγχου (control flow) του προγράμματος προς τα πίσω (ανάστροφη ανάλυση ροής– backward flow analysis), εντοπίζοντας τις κλήσεις των μεθόδων (method calls), και εκτελώντας ελέγχους τόσο στα πλαίσια άλλων μεθόδων της ίδια κλάσης (intraclass checks), όσο και στα πλαίσια μεθόδων εντελώς ξεχωριστών κλάσεων (interclass ckecks), (interprocedural έλεγχοι).

Η διαφοροποίηση επεκτείνεται και στη δομή των υποθετικών δηλώσεων που δύνανται να ανιχνεύσουν οι δύο μεθοδολογίες. Συγκεκριμένα, ο έλεγχος βάσει τύπου εντοπίζει δηλώσεις if και switch που περιλαμβάνουν είτε πεδία, είτε παραμέτρους, είτε τοπικές μεταβλητές, οι οποίες συγκρίνονται με σταθερές (public static final). Αντιθέτως, ο υποθετικός έλεγχος (conditional checking) λαμβάνει υπόψη τις δηλώσεις if και switch εφόσον οι συνθήκες τους περιλαμβάνουν παραμέτρους και πεδία, ασχέτως της ύπαρξης σταθεράς ή μη.

Βασική διαφοροποίηση της προτεινόμενης μεθοδολογίας είναι η παροχή δυνατότητας παραμετροποίησης των κριτηρίων που χρησιμοποιούνται στην ανίχνευση, ενεργοποιώντας ή απενεργοποιώντας καθένα από αυτά. Η δυνατότητα αυτή δεν υποστηρίζεται στην ανίχνευση περιπτώσεων ελέγχου τύπου (type checking).

Μια ακόμα διαφορά της ανίχνευσης περιπτώσεων αναδόμησης προς τη στρατηγική αποτελεί η ανίχνευση ορισμένων περιπτώσεων υποθετικής λογικής στις οποίες ενδείκνυνται δύο άλλοι τύποι αναδόμησης με πρότυπα σχεδίασης και η απομάκρυνση τους από το σύνολο των υποψήφιων προτάσεων. Συγκεκριμένα εντοπίζονται υποθετικές δηλώσεις που φιλοξενούν δομές για τις οποίες συνίσταται η αναδόμηση με το πρότυπο *κατάσταση*, καθώς και περιπτώσεις στις οποίες ενδείκνυται η αναδόμηση βάσει του προτύπου *μέθοδος εργοστάσιο* (factory method).

Τέλος, κυριότερη διαφοροποίηση αναφορικά με την ανίχνευση αποτελεί το γεγονός ότι η υπάρχουσα υλοποίηση του JDeodorant αξιοποιεί αποκλειστικά ελέγχους που βασίζονται στη στατική ανάλυση του πηγαίου κώδικα (static analysis) με χρήση του δέντρου AST για την αναπαράσταση του πηγαίου κώδικα. Η προτεινόμενη μεθοδολογία ωστόσο εκτός από το δέντρο AST χρησιμοποιεί και τους γράφους CFG και PDG προκειμένου να αξιοποιήσει και τις πληροφορίες ροής ελέγχου που προσφέρουν οι συγκεκριμένες μορφές αναπαράστασης, ενώ προκειμένου να επιτύχει την ανίχνευση περιπτώσεων στις οποίες η υποθετική λογική μπορεί να απαλειφθεί πλήρως, περιλαμβάνει επιπλέον την επιλεκτική εκτέλεση ορισμένων τμημάτων κώδικα έξω από τα πλαίσια του προγράμματος στο οποίο δηλώνονται, στα πλαίσια του προγράμματος ανίχνευσης.

Αναφορικά με την ταξινόμηση των προτάσεων αναδόμησης η ανάθεση των βαρών που προτείνεται διαφοροποιείται από την υπάρχουσα, η οποία σχετίζεται με την συχνότητα εμφάνισης όμοιων ελέγχων και το μέσο μέγεθος κάθε διακλάδωσης. Η προτεινόμενη ανάθεση βαρών σε πρώτο επίπεδο συσχετίζει τα βάρη με ορισμένες από τις δομές (patterns) εναλλακτικών αλγορίθμων στρατηγικής που απαντώνται συχνότερα στον κώδικα και εν συνεχεία τις κατατάσσει βάσει του μέσου μεγέθους ανά διακλάδωση.

Ως προς την εκτέλεση της αναδόμησης, κυριότερη διαφοροποίηση της προτεινόμενης μεθοδολογίας αποτελεί η υπό συνθήκη πλήρης υποκατάσταση της υποθετικής λογικής μέσω του προσδιορισμού της κατάλληλης κλάσης στρατηγικής και της κλήσης της μεθόδου της. Άλλες μικρότερες διαφορές αποτελούν, ο τρόπος ονοματοθεσίας των κλάσεων, η παραγωγή των ονομάτων των οποίων δε δύναται πλέον να βασιστεί στα ονόματα των σταθερών, αφού οι υποθετικές εκφράσεις που ανιχνεύονται δεν περιορίζονται σε αυτές που περιλαμβάνουν υποχρεωτικά μία σταθερά (constant) κατ' ελάχιστο, καθώς και ο τρόπος που ανασυντάσσεται ο κώδικας μετά την αναδόμηση. Στην υπάρχουσα μεθοδολογία η εκτέλεση της αναδόμησης αντικαθιστά πλήρως την υποθετική λογική στην πρωτότυπη μέθοδο με την πολυμορφική κλήση της μεθόδου. Ωστόσο αντίγραφο της υποθετικής λογικής δημιουργείται στη setter μέθοδο του πεδίου στρατηγικής, προκειμένου να δημιουργείται το κατάλληλο αντικείμενο. Αντιθέτως, στην προτεινόμενη μεθοδολογία, η υποθετική λογική παραμένει στην πρωτότυπη μέθοδο και οι εντολές των διακλαδώσεων αντικαθίστανται από εντολές δημιουργίας του αντικειμένου στρατηγικής καθώς και την πολυμορφική κλήση της μεθόδου.

Οι ανωτέρω διαφοροποιήσεις υιοθετήθηκαν με σκοπό την κατά το δυνατό βαθμό βελτίωση των αποτελεσμάτων και την ταυτόχρονη αύξηση του εύρους των δυνατοτήτων ανίχνευσης υποψήφιων περιπτώσεων αναδόμησης προς τη στρατηγική. Έμφαση επιπλέον δόθηκε στο διαχωρισμό των περιπτώσεων αναδόμησης μεταξύ των προτύπων κατάσταση και στρατηγική, τα οποία παρα την ομοιότητα στη δομή διαφέρουν καθώς απευθύνονται σε διαφορετικής φύσης προβλήματα, συνεπώς θα πρέπει να εξετάζονται ξεχωριστά κατά την αναδόμηση. Ο διαχωρισμός αυτός σε συνδυασμό με την επίτευξη πλήρους αντικατάστασης υποθετικής λογικής αποτελούν την κυριότερη συμβολή της προσέγγισης αυτής.

Εντούτοις, παρά τη συμβολή της, η προτεινόμενη μεθοδολογία όπως και κάθε αναπτυσσόμενη προσέγγιση αυτοματοποιημένης αναδόμησης παρουσιάζει και ορισμένες αδυναμίες. Ορισμένες από τις αδυναμίες αυτές είναι σύμφυτες με το υπό εξέταση πρόβλημα και την έλλειψη υποστήριξης νοητικών ικανοτήτων από την πλευρά του τεχνικού μέσου που αναλαμβάνει την εκτέλεση της μεθοδολογίας, ενώ άλλες σχετίζονται άμεσα με την προτεινόμενη προσέγγιση και το σύνολο των κριτηρίων που επιλέχθηκε για τους σκοπούς της ανίχνευσης.

Συγκεκριμένα, το πρότυπο στρατηγική και η κυριότερη διαφοροποίηση του από το πρότυπο κατάσταση, η οποία σχετίζεται με το σκοπό των δύο προτύπων, έχει έντονη σημασιολογική-εννοιολογική υφή. Συνεπώς, η ανίχνευση περιπτώσεων ελλιπούς σχεδίασης που χρήζουν αναδόμησης για την εφαρμογή του προτύπου στρατηγικής ανάγεται στο πρόβλημα σημασιολογικής ανάλυσης του κώδικα. Η ανίχνευση διαφορετικών παραλλαγών ενός αλγορίθμου, που περιλαμβάνονται στις διαφορετικές διακλαδώσεις μιας υποθετικής δήλωσης, σχετίζεται με την ερμηνεία που αποδίδεται στις λέξεις «αλγόριθμος» και «παραλλαγή», καθώς και τη σημασιολογία του πηγαίου κώδικα, η οποία συχνά καθίσταται δύσκολη ακόμα και για έμπειρους μηχανικούς λογισμικού, που εκτός από την κατανόηση του πηγαίου κώδικα μπορούν να αξιοποιήσουν και την προγραμματιστική εμπειρία τους, προκειμένου να ανιχνεύσουν την έλλειψη του συγκεκριμένου προτύπου. Το πρόβλημα αυτό καθίσταται δυσκολότερο για τους ηλεκτρονικούς υπολογιστές, οι οποίοι αδυνατούν να συμπεράνουν με κάποιο τρόπο τη σημασιολογία τμημάτων κώδικα.

Ένας ακόμα περιορισμός της μεθόδου είναι η έλλειψη υποστήριξης πλήρους αυτοματοποίησης, δηλαδή η απουσία υποστήριξής αυτόματης λήψης αποφάσεων σχετικά με την εφαρμογή ή μη, των προτεινόμενων αναδομήσεων, υπό την έννοια ότι το προτεινόμενο εργαλείο δρα συμβουλευτικά και υποβοηθητικά, ενώ η λήψη της τελικής απόφασης επαφίεται στον προγραμματιστή και χρήστη της εφαρμογής. Η εξέταση του αποτελέσματος της ανίχνευσης απαιτεί την ανθρώπινη συμμετοχή και την ενεργοποίηση των προσωπικών ικανοτήτων του χρήστη όπως η δυνατότητα κατανόησης του κώδικα, η γνώση του συγκεκριμένου προτύπου και η εμπειρία από τη χρήση του, η προσωπική του κρίση και δυνατότητα εξαγωγής συμπερασμάτων σχετικά με τη βελτίωση της ποιότητας του κώδικα έπειτα από την εφαρμογή. Συνεπώς, η αποτελεσματικότητα της προτεινόμενης μεθοδολογίας σχετικά με τη βελτίωση της ποιότητας του κώδικα επηρεάζεται από εξωγενείς, αστάθμητους παράγοντες, που δεν είναι δυνατό να ελεγχθούν.

Σημαντικό επιπλέον περιορισμό του εργαλείου που αναπτύχθηκε αποτελεί η δυνατότητα εξέτασης και εφαρμογής αυτοματοποίησης αναδόμησης μόνο στο υποσύνολο των προγραμμάτων λογισμικού που έχουν αναπτυχθεί στη γλώσσα προγραμματισμού java. Η αδυναμία αυτή, πέραν των χρησιμοποιούμενων τεχνολογιών κατά την ανάπτυξη του εργαλείου, οφείλεται στην ίδια την μεθοδολογία ανίχνευσης που αναπτύχθηκε, η οποία αναφέρεται ειδικά στη δομή και τις ιδιαιτερότητες της γλώσσας προγραμματισμού Java. Η γενίκευση της προτεινόμενης μεθόδου, προκειμένου η χρήση της να μπορεί να επεκταθεί και σε άλλες γλώσσες προγραμματισμού, καθίσταται ιδιαίτερα δύσκολη, ειδικά σε σύγκριση με άλλες παρόμοιες μεθόδους, καθώς βασίζεται στην τεχνική ταιριάσματος προτύπων (pattern matching), και μια τέτοια ενέργεια απαιτεί τη μελέτη των χαρακτηριστικών κάθε γλώσσας προγραμματισμού ξεχωριστά και την ανάπτυξη ξεχωριστών προτύπων που αφορούν καθεμία από αυτές.

Επιπλέον, εφόσον ανάλογα με τη δομή και τη σημασιολογία της υποθετικής λογικής ενδέχεται να ενδείκνυται και διαφορετικός τύπος αναδόμησης με πρότυπα σχεδίασης, η προτεινόμενη μεθοδολογία για να θεωρηθεί πλήρης, θα έπρεπε να αποκλείει το ενδεχόμενο, κάποια από τις προτάσεις αναδόμησης που προκύπτουν ως αποτέλεσμα της ανίχνευσης, να αφορά ουσιαστικά πρόταση αναδόμησης για κάποιο από τα πρότυπα που παρουσιάστηκαν στην ενότητα 4.1. Ωστόσο αυτό υπάγεται στη μελέτη ενός ευρύτερου και συνθετότερου προβλήματος, το πρόβλημα αναδόμησης πολύπλοκής υποθετικής λογικής με το κατάλληλο πρότυπο σχεδίασης. Όπως άλλωστε αναφέρεται και από τον Τσάνταλη στο ερευνητικό έργο του περί αναγνώρισης δυνατοτήτων αναδόμησης με χρήση του πολυμορφισμού (1), κάθε πρότυπο σχεδίασης, εφόσον απευθύνεται σε διαφορετικό σχεδιαστικό πρόβλημα και έχει διαφορετική δομή, απαιτεί μια εντελώς διαφορετική προσέγγιση για την αναγνώριση των περιπτώσεων στις οποίες θα μπορούσε να εισαχθεί, γεγονός που καθιστά αδύνατη την ανάπτυξη μιας κοινής τεχνικής που θα κάλυπτε όλες τις ευκαιρίες αναδόμησης έναντι στην πολύπλοκη υποθετική λογική. Μια τέτοια μελέτη θα απαιτούσε την ανάπτυξη ξεχωριστών ελέγχων, βάσει των συνηθέστερων κακών πρακτικών (anti-patterns) για καθένα από τα πρότυπα αυτά, προσαρμοσμένων στις ξεχωριστές τους ανάγκες και ιδιαιτερότητες, η οποία δεν ήταν δυνατό να διεξαχθεί στο διαθέσιμο χρονικό διάστημα που καθορίστηκε για την ανάπτυξη της παρούσας εργασίας.

Επιπροσθέτως κατ' αντιστοιχία με το ερευνητικό έργο του Τσάνταλη (1), η παρούσα εργασία μελετά μόνο τις υποθετικές δηλώσεις if και switch, ενώ εξαιρεί την εξέταση του τριαδικού τελεστή «?:», ο οποίος χρησιμοποιείται συνήθως ως δεξί μέλος σε εκφράσεις εκχώρησης, για τον υπολογισμό κάποιας τιμής και όχι για πολύπλοκα τμήματα εκτελέσιμου κώδικα, ενώ η χρήση του σε γενικές γραμμές σε πραγματικά συστήματα λογισμικού δεν είναι το ίδιο συχνή με τις υπόλοιπες δύο υποθετικές δηλώσεις. Επιπλέον ο τριαδικός τελεστής αδυνατεί να αναπαραστήσει πολύτμηματικές υποθετικές δηλώσεις, εφόσον περιορίζεται σε δύο διαφορετικές περιπτώσεις, συνεπώς θεωρείται πώς η αναδόμηση του θα είχε μικρή θετική επίδραση στην ποιότητα του σχεδίου, ενώ παράλληλα κρίνεται σχετικά απίθανο, ο εκτελέσιμος κώδικας του, κυρίως λόγω του περιορισμένου μεγέθους του, να ταιριάζει στην περίπτωση χρήσης του προτύπου στρατηγικής.

Αναφορικά με τον αυτοματοποιημένο εκτελούμενο μετασχηματισμό, θα πρέπει να τονιστεί ότι η προτεινόμενη μεθοδολογία δεν απαλείφει την υποθετική λογική σε κάθε περίπτωση παρά μόνο σε κάποιο υποσύνολο των περιπτώσεων, όπου μπορεί να βρεθεί σε χρόνο μεταγλώττισης, η συνθήκη που ικανοποιείται από τις τιμές των ορισμάτων που προσδιορίζουν οι κλάσεις πελάτες κατά την κλήση τους.

Ακολούθως παρουσιάζονται ορισμένες αδυναμίες της μεθοδολογίας που σχετίζονται με το σύνολο των κριτηρίων που αναπτύχθηκαν για την επίτευξη της ανίχνευσης.

Συγκεκριμένα, εφόσον η μεθοδολογία που αναπτύχθηκε αναζητά στο σύνολο των υποθετικών δηλώσεων εκείνων, στις οποίες ο έλεγχος διεξάγεται με βάση την τιμή των παραμέτρων και των πεδίων της κλάσης, ενδέχεται να μην ανιχνεύονται έγκυρες περιπτώσεις αναδόμησης όπου στη συνθήκη υπάρχει κλήση μεθόδου της ίδιας κλάσης, η οποία καλείται χωρίς κάποιο όρισμα και χωρίς κάποιον καλούντα, όπως απεικονίζεται στο παράδειγμα της Εικόνα 45 που αποτελεί απόσπασμα από το σύστημα JFScheduler.

```
public void openUrl(String url) throws InterruptedException, IOException
{
        String cmd = null;
        if (isWindows())
        {
            cmd = ("rundll32 url.dll,FileProtocolHandler " +
            maybeFixupURLForWindows(url));
            Runtime.getRuntime().exec(cmd);
        }
        else {
            cmd = "firefox " + url;
             Process p = Runtime.getRuntime().exec(cmd);
             int exitcode = p.waitFor();
             if (exitcode != 0) {
                    cmd = "netscape " + url;
                    Runtime.getRuntime().exec(cmd);
             }
        }
}
```

Εικόνα 45: Περίπτωση μεθόδου του JFScheduler όπου η ανίχνευση υποθετικών δομών αποτυγχάνει λόγω της κλήσης μεθόδου στην υποθετική έκφραση

Κατ’ αντιστοιχία αδυναμία συναντάται από την προτεινόμενη μεθοδολογία, σχετικά με τον εντοπισμό περιπτώσεων υποθετικής λογικής που σχετίζονται με configuration παραμέτρους και παραμέτρους του συστήματος, όπως συμβαίνει στο παράδειγμα της Εικόνα 46 που απεικονίζει ένα απόσπασμα του κώδικα του συστήματος λογισμικού JFScheduler (101).

```
public void openTxt(String txt) throws IOException, InterruptedException{
    String cmd = null;
    if (System.getProperty("os.name").indexOf("Windows") != -1)
    {
        cmd = "notepad " + txt;
        Runtime.getRuntime().exec(cmd);
    }
    else {
        cmd = "gedit " + txt;
        Process p = Runtime.getRuntime().exec(cmd);
        int exitcode = p.waitFor();
        if (exitcode != 0) openUrl(txt);
    }
}
```

Εικόνα 46 : Αδυναμία ανίχνευσης υποθετικών δομών με έλεγχο μεταβλητών συστήματος

Επιπλέον, η προτεινόμενη μεθοδολογία απορρίπτει από το σύνολο των εξεταζόμενων υποθετικών δηλώσεων τις δηλώσεις εκείνες οι οποίες βασίζονται στον έλεγχο τιμής μιας τουλάχιστον τοπικής μεταβλητής. Ωστόσο η τιμή της τοπικής μεταβλητής ενδέχεται να καθορίζεται εξωτερικά της κλάσης context και να λαμβάνεται είτε μέσω κάποιας παραμέτρου της μεθόδου που συμμετέχει στην εκχώρηση, είτε κάποιου πεδίου. Ο έλεγχος του πλήρους υπολογισμού μιας τοπικής μεταβλητής έχει μελετηθεί στα πλαίσια του ερευνητικού έργου (85), ωστόσο λόγω του χρονικού περιορισμού δεν κατέστει εφικτή η ενσωμάτωση της στην προτεινόμενη μεθοδολογία.

Μια ακόμα περίπτωση υποθετικής δήλωσης που απορρίπτεται από το σύνολο των υποψήφιων προτάσεων αναδόμησης είναι η περίπτωση όπου στο ίδιο μονοπάτι ή σε κάποιο από τα μονοπάτια εκτέλεσης της υποθετικής λογικής, περιλαμβάνεται εντολή εκχώρησης που αλλάζει την τιμή μιας τουλάχιστον από τις παραμέτρους της συνθήκης, ωστόσο η τιμή που της εκχωρείται, λαμβάνεται είτε από κάποια άλλη παράμετρο, ή από το συνδυασμό διαφορετικών παραμέτρων, είτε από κάποιο πεδίο της κλάσης, για το οποίο δεν υπάρχουν εντολές εκχώρησης στα πλαίσια της κλάσης context, εκτός από τον κατασκευαστή και τη setter μέθοδό του.

Αντίστοιχη με την προηγούμενη περίπτωση είναι η περίπτωση όπου στην υποθετική έκφραση υπάρχουν πεδία της κλάσης context τα όποια ωστόσο περιλαμβάνονται σε εκφράσεις εκχωρήσεων, όπου η ανατιθέμενη τιμή δεν προσδιορίζεται από την ίδια την κλάση context αλλά λαμβάνεται μέσω της κλήσης μεθόδου από κάποια εξωτερική πηγή, όπως παραδείγματος χάριν κάποιο αρχείο παραμετροποίησης (configuration file) (Εικόνα 47). Η περίπτωση αυτή είναι δύσκολο να εντοπιστεί καθώς εντάσσεται στα πλαίσια σημασιολογικής ανάλυσης του πηγαίου κώδικα, συνεπώς δεν υπάρχει τρόπος σε χρόνο μεταγλώττιsης να διαπιστωθεί. Κατ’ αντιστοιχία, από το υπό εξέταση σύνολο εκχωρήσεων των πεδίων της υποθετικής έκφρασης θα έπρεπε να εξαιρούνται εκείνες οι οποίες χρησιμοποιούνται για την αρχικοποίηση των πεδίων και περιλαμβάνονται στα πλαίσια μεθόδων clone(), initialize(), configure(), κ.ο.κ. Οι μέθοδοι αυτοί θεωρείται ότι θα έπρεπε ουσιαστικά να εξαιρούνται καθώς συνήθως, χρησιμοποιούνται για την ανάθεση μιας αρχικής τιμής στα πεδία, η οποία στη συνέχεια τροποποιείται σε άλλα σημεία του προγράμματος κατά τη διάρκεια εκτέλεσης του. Ωστόσο, ο διαχωρισμός εκχωρήσεων που στοχεύουν απλά στην αρχικοποίηση ενός πεδίου, και δεν βρίσκονται ούτε στην αρχική δήλωση του πεδίου, ούτε στον κατασκευαστή, είναι δύσκολη καθώς απαιτεί την εννοιολογική ερμηνεία του πλαισίου μέσα στο οποίο δηλώνονται.

```
public void configureParams(){
try {

        DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document doc = db.parse("config" + File.separator + configFile);
        phyLayer = doc.getElementsByTagName("phyLayer").item(0).getTextContent().charAt(0);
        ….
        }
}

private void collision()
{
      ….
      //802.11a or 802.11g
```

```
        if((phyLayer=='a')||(phyLayer=='g'))
        {
                //handle collision
                …
        } //802.11b
        else {
                //handle collision, different handling
                ……
        }
}
```

Εικόνα 47: Απόσπασμα του project JFScheduler έγκυρης πρότασης αναδόμησης υποθετικής λογικής βάσει πεδίου που δεν ανιχνεύεται λόγω εκχώρησης στο πεδίο από configuration.file

Τέλος, μια ακόμη περίπτωση υποψήφιας προς αναδόμηση δήλωσης if, η οποία ωστόσο δεν ανιχνεύεται, είναι η περίπτωση της Εικόνα 48, στην οποία οι υποθετικές εκφράσεις που απαρτίζουν τη δήλωση if δε διαθέτουν το ίδιο σύνολο παραμέτρων/πεδίων. Η περίπτωση αυτή απορρίπτεται κατά την αναζήτηση καθώς σύμφωνα με έναν από τους περιορισμούς που θέτουν τα κριτήρια ανίχνευσης προκειμένου να θεωρηθούν υποψήφιες αναδόμησης οι δηλώσεις if, θα πρέπει ο έλεγχος στις συνθήκες τους να στηρίζεται στις ίδιες παραμέτρους. Ωστόσο παρόλο που το συγκεκριμένο παράδειγμα έρχεται σε αντίθεση με τον περιορισμό αυτό, ορισμένες φορές συνάδει με την χρήση του προτύπου στρατηγικής.

```
public void closeFile(boolean bWrite) {
        try {
                if (bWrite) {
                        if (fileWriter != null)
                                fileWriter.close();
                } else if (fileReader != null) {
                        fileReader.close();
                }
        } catch (IOException e) {
                        e.printStackTrace();
        }
}
```

Εικόνα 48 : Αδυναμία ανίχνευσης υποθετικών εκφράσεων με διαφορετικές μεταβλητές

Συμπερασματικά, αρκετές από τις αδυναμίες της προτεινόμενης μεθοδολογίας οφείλονται στις περιορισμένες δυνατότητες εξαγωγής συμπερασμάτων αποκλειστικά με χρήση αυτοματοποιημένων μεθόδων ανάλυσης κώδικα, χωρίς την αξιοποίηση της ανθρώπινης νόησης, εφόσον το υπο εξέταση πρόβλημα έχει έντονη εννοιολογική φύση. Επιπροσθέτως, οι ανωτέρω αδυναμίες καθιστούν εμφανή τη δυσκολία εύρεσης κριτηρίων περιγραφής της δομής των υποψήφιων προς αναδόμηση υποθετικών δηλώσεων, τα οποία ικανοποιούνται σε κάθε περίπτωση. Το σύνολο των κριτηρίων ανίχνευσης θα πρέπει να είναι γενικού τύπου ώστε να μην απορρίπτονται εσφαλμένα έγκυρες προτάσεις αναδόμησης, ενώ ταυτόχρονα θα πρέπει να είναι και αρκετά ειδικά ώστε να μην προκύπτουν πολλές ψευδείς/λανθασμένες προτάσεις αναδόμησης. Η εύρεση ενός τέτοιου συνόλου κριτηρίων είναι αρκετά δύσκολη καθώς, εφόσον η συγγραφή κώδικα συντελείται αυτόνομα χωρίς την αυστηρή προσκόλληση σε συγκεκριμένα πρότυπα και προγραμματιστικές δομές, είναι σχεδόν αδύνατο να μην υπάρχουν εξαιρέσεις οι οποίες αντίκειται στους ορισμούς των κριτηρίων, όποια κι αν είναι αυτά. Γι’ αυτό το λόγο τα κριτήρια που ορίζει η συγκεκριμένη μεθοδολογία δεν αποτελούν πανάκεια, αλλά επιλέχθηκαν επειδή θεωρήθηκε ότι παρουσιάζουν τις ελάχιστες δυνατές εξαιρέσεις, ενώ η δομή που περιγράφουν κατά κανόνα αρμόζει στις υποψήφιες προτάσεις αναδόμησης υποθετικής λογικής, ούτως ώστε κατά τη διεξαγωγή της ανίχνευσης να προκύπτουν τα βέλτιστα δυνατά αποτελέσματα.

Τα αποτελέσματα από την πειραματική εκτέλεση της μεθοδολογίας σε ορισμένα πραγματικά συστήματα λογισμικού (real world projects) αναλύονται στο κεφάλαιο που ακολουθεί.

Κεφάλαιο 6

Πειραματική αξιολόγηση

Για την αξιολόγηση της αποτελεσματικότητας της προτεινόμενης μεθοδολογίας τόσο ως προς την ποιότητα των αποτελεσμάτων όσο και ως προς την κλιμακωσιμότητα, καθίσταται αναγκαία η πειραματική εφαρμογή της.

Η αξιολόγηση της ποιότητας των αποτελεσμάτων με χρήση διαφόρων μετρικών ορθότητας, επιτυγχάνεται μέσω της εφαρμογής της προτεινόμενης μεθοδολογίας σε διάφορα συστήματα λογισμικού ανοικτού κώδικα *(open-source)*. Εν συνεχεία, τα αποτελέσματα που προκύπτουν, συγκρίνονται με την προσωπική εκτίμηση ενός ανεξάρτητου αξιολογητή. Αυτή η μέθοδος αξιολόγησης βασίζεται στο γεγονός ότι, παρόλο που οι προτάσεις αναδόμησης ανιχνεύονται με αυτόματο τρόπο, ωστόσο η τελική απόφαση για την εφαρμογή τους, βάσει εννοιολογικών και ποιοτικών κριτηρίων, επαφίεται στον τελικό χρήστη. Συνεπώς, η προτεινόμενη προσέγγιση μπορεί να χαρακτηριστεί ως ημιαυτόματη και η αξιολόγηση της μέσω της επιθεώρησης των αποτελεσμάτων από κάποιον ανεξάρτητο κριτή, για την εξέταση της σημασιολογίας τους, της χρησιμότητας τους και της θετική τους επίδρασης στην ποιότητα σχεδίου, καθίσταται αναγκαία.

Επιπλέον, αναγκαία κρίνεται και η αξιολόγηση της προτεινόμενης μεθοδολογίας συναρτήσει της κλιμακωσιμότητας της. Συγκεκριμένα, βάσει της ανάλυσης που προηγήθηκε στο κεφάλαιο 5, η προτεινόμενη τεχνική ανίχνευσης υποψήφιων προς αναδόμηση προτάσεων στηρίζεται στη στατική ανάλυση και στην εξέταση του συνδυασμού διάφορων εναλλακτικών μορφών αναπαράστασης του πηγαίου κώδικα, συνεπώς η κλιμακωσιμότητα της εξαρτάται άμεσα από την κλιμακωσιμότητα των χρησιμοποιούμενων μεθοδολογιών. Ωστόσο, η δημιουργία του AST δέντρου και η διάσχιση του για τον εντοπισμό των προτάσεων αναδόμησης αποτελεί μια διαδικασία με μεγάλες απαιτήσεις μνήμης, για την αποθήκευση των τελικών και ενδιάμεσων αποτελεσμάτων, η διάρκεια της οποίας αυξάνεται κατ' αναλογία με την αύξηση του μεγέθους των υπό εξέταση συστημάτων. Επιπλέον ιδιαίτερα απαιτητική είναι και η δημιουργία των αναπαραστάσεων των γράφων CFG και PDG, η κατασκευή των οποίων αυξάνει σημαντικά το χρόνο ανίχνευσης των προτάσεων αναδόμησης, ιδιαίτερα όταν το υπό εξέταση λογισμικό αποτελείται από εκτενείς πολύπλοκές μεθόδους με μεγάλο αριθμό πιθανών μονοπατιών εκτέλεσης.

Συνεπώς κρίνεται αναγκαίο να εξεταστεί η δυνατότητα απόκρισης της μεθόδου σε ένα εύλογο χρονικό διάστημα, ανεξαρτήτως του μεγέθους του υποψήφιου προς αναδόμηση συστήματος λογισμικού, καθώς η αξία της τίθεται υπό αμφισβήτηση, αν το χρονικό διάστημα που απαιτείται για την εξέταση του, καθιστά την χρήση της απαγορευτική. Για τη διενέργεια αυτού του ελέγχου επιλέχθηκαν διάφορα συστήματα ανοικτού λογισμικού, με διαφορετικά χαρακτηριστικά μεγέθους και μετρήθηκε ο *χρόνος υπολογισμού (computation time)* που καταλαμβάνει η εφαρμογή της μεθόδου στα συστήματα αυτά.

6.1. Συστήματα Λογισμικού ως Πειραματικά Δεδομένα

Τα συστήματα λογισμικού που χρησιμοποιήθηκαν ως δοκιμαστικά δεδομένα για την αξιολόγηση της προτεινόμενης μεθοδολογίας υπάγονται στην κατηγορία των συστημάτων ανοικτού κώδικα *(open source).* Όπως αναφέρει και ο Τσάνταλης κατά την αιτιολόγηση των χρησιμοποιούμενων συστημάτων στην εκτέλεση των δικών του πειραμάτων (1), η επιλογή αυτή καθίσταται αναγκαία για την εκτέλεση του πειράματος, προκειμένου ο πηγαίος κώδικας των υπό εξέταση συστημάτων να είναι δημόσια διαθέσιμος, ούτως ώστε να μπορεί να χρησιμοποιηθεί ως είσοδος του εργαλείου που υλοποιεί την προτεινόμενη τεχνική, καθώς επίσης και για την ενδεχόμενη επαλήθευση των αποτελεσμάτων, μέσω της επανάληψης των πειραμάτων από τρίτους.

Ωστόσο η διαθεσιμότητα του πηγαίου κώδικα δεν αποτέλεσε το μοναδικό κριτήριο για την επιλογή των συστημάτων λογισμικού. Βασικό περιορισμό ως προς τη γλώσσα προγραμματισμού η οποία έχει χρησιμοποιηθεί για την υλοποίηση των συστημάτων και ταυτόχρονα κριτήριο για την επιλογή τους, εισάγει η ίδια η υλοποίηση του εργαλείου που αναπτύχθηκε, καθώς οι δυνατότητες ανάλυσης της εστιάζουν σε συστήματα λογισμικού γραμμένα με χρήση της γλώσσας προγραμματισμού Java. Συνεπώς για την εκτέλεση του πειράματος επιλέχθηκαν συστήματα λογισμικού αποκλειστικά υλοποιημένα στη γλώσσα προγραμματισμού Java, ούτως ώστε να είναι εφικτή η ανάλυση του πηγαίου κώδικα από το εργαλείο που αναπτύχθηκε. Επιπλέον, σημαντικό ρόλο διαδραμάτισε κατά τη διαδικασία επιλογής η χρήση συστημάτων λογισμικού μεγάλου χρονικού ορίζοντα, τα οποία υφίστανται διαρκών τροποποιήσεων και βρίσκονται ακόμα υπό εξέλιξη, ούτως ώστε να αυξηθεί η πιθανότητα εντοπισμού πιθανών προτάσεων αναδόμησης προς τη στρατηγική.

Προκειμένου να καταστεί εμφανής η διαφοροποίηση των αποτελεσμάτων της προτεινόμενης προσέγγισης από την υπάρχουσα προσέγγιση του JDeodorant, τα συστήματα λογισμικού που χρησιμοποιήθηκαν ως επί το πλείστον, αποτελούν νεώτερες εκδόσεις των συστημάτων που επιλέχθηκαν για την αξιολόγηση της αναδόμησης με χρήση πολυμορφισμού στη σχετική ερευνητική δημοσίευση του Τσάνταλη (1). Τα συγκεκριμένα συστήματα ενδείκνυνται ως δοκιμαστικά δεδομένα και επιλέχθηκαν αρχικά καθώς εξυπηρετούν διαφορετικούς σκοπούς, πηγάζοντας από διαφορετικούς τομείς εφαρμογής, καθιστώντας εφικτή, ως ένα βαθμό, τη γενίκευση των συμπερασμάτων.

Από τα συστήματα λογισμικού που επιλέχθηκαν, κρίθηκε χρήσιμο ορισμένα να είναι ήδη γνωστά στον ανεξάρτητο αξιολογητή. Η επιλογή αυτή έγινε αφενός μεν για την διευκόλυνση της διαδικασίας του πειράματος, αφετέρου δε για να διασφαλιστεί η ορθότητα της κρίσης του και η αξιοπιστία της διενεργούμενης διαδικασίας, εφόσον πρόκειται για πηγαίο κώδικα του οποίου η λειτουργικότητα είναι γνωστή και πλήρως κατανοητή στον αξιολογητή. Για λόγους πληρότητας και για να εξεταστεί η ποιότητα των αποτελεσμάτων της μεθόδου γενικά σε διάφορα συστήματα, εφόσον μια μεθοδολογία που παράγει καλά αποτελέσματα σε μια μόνο μεμονωμένη περίπτωση δεν έχει πρακτική αξία, θεωρήθηκε επιπλέον αναγκαία η χρήση άλλων συστημάτων λογισμικού, δημόσια διαθέσιμων, άγνωστων στον αξιολογητή, σχετικά περιορισμένου μεγέθους ωστόσο, ώστε να είναι εφικτή η εξέταση τους και η επαρκής κατανόηση της λειτουργικότητας τους. Για τον υπολογισμό των χαρακτηριστικών μεγέθους και των μετρικών των υπό εξέταση συστημάτων χρησιμοποιήθηκε η έκδοση 1.3.6 του plug-in Metrics του eclipse (102), τα αποτελέσματα της οποίας παρουσιάζονται στον Πίνακας 12**Error! Reference source not found.**.

Από το σύνολο των υπό εξέταση συστημάτων λογισμικού, ορισμένα χρησιμοποιήθηκαν κατά την αξιολόγηση της ποιότητας των αποτελεσμάτων από τον ανεξάρτητο αξιολογητή και ορισμένα κατά την αξιολόγηση της κλιμακωσιμότητας της προτεινόμενης μεθοδολογίας. Κατά την εξέταση από τον ανεξάρτητο αξιολογητή χρησιμοποιήθηκαν τα συστήματα λογισμικού

- JFScheduler : το σύστημα περιλαμβάνει: α) υλοποίηση ενός αλγορίθμου βασισμένου σε τοπική αναζήτηση για την παραγωγή προσεγγιστικών λύσεων του προβλήματος κατανομής ροών κίνησης (traffic flow assignment problem - TFAP)[9] όπως προδιαγράφεται στο (101), β) υποδομή για την αξιολόγηση του αλγορίθμου μέσω παραγωγής τυχαίων προβλημάτων και των πραγματικών τους λύσεων, γ) προσομοιωτή διακριτών γεγονότων για την αξιολόγηση της επίδοσης του αλγορίθμου σε ένα προσομοιωμένο περιβάλλον ασύρματης δικτυακής πρόσβασης.
- IceHockeyManager 0.4 : πρόκειται για ένα ανοικτού κώδικα παιχνίδι, το οποίο αφορά τη διαχείριση μιας ομάδας χόκεϋ *(hockey)* και λειτουργεί σε περιβάλλον Linux, MacOS και Windows (103).
- MOEM : σύστημα που υλοποιεί μια πλατφόρμα για την μοντελοποίηση πολυδιάστατων ημι-δομημένων δεδομένων σύμφωνα με το μοντέλο Multidimensional Object Exchange Model (MOEM). Η λειτουργικότητα του συστήματος περιγράφεται στο (104) ενώ περισσότερες πληροφορίες για το MOEM μοντέλο περιλαμβάνονται στο (105).

Κατά την εξέταση της κλιμακωσιμότητας χρησιμοποιήθηκαν τα εξής συστήματα λογισμικού

- Apache Nutch 1.1 : πρόκειται για ένα ανοικτού κώδικα λογισμικό διαδικτυακής αναζήτησης *(web-search software)* (106). Το σύστημα βασίζεται στο Lucene και το Solr για την προσθήκη διαδικτυακών χαρακτηριστικών όπως crawlers, parsers για HTML και άλλα μορφότυπα αρχείων, κ.ο.κ.
- Violet : πρόκειται για ένα ανοικτού κώδικα εύχρηστο επεξεργαστή UML *(UML editor),* ο οποίος επιτρέπει τη δημιουργία διαφόρων τύπων διαγραμμάτων της UML, παραδείγματος χάριν διάγραμμα κλάσεων, διάγραμμα δραστηριοτήτων, διάγραμμα μηχανής καταστάσεων, κλπ. (107) (108).
- Pamvotis 1.1: πρόκειται για ένα προσομοιωτή ασύρματου δικτύου (WLAN simulator), ο οποίος υποστηρίζει όλες τις εκδόσεις του πρωτοκόλλου φυσικού επιπέδου IEEE 802.11 a, b, g και e (109).

[9] Κατανομή της κίνησης ενός ασύρματου σταθμού με πολλαπλές δικτυακές διεπαφές, σε κατάλληλες δικτυακές διεπαφές και υπηρεσίες μεταφοράς δεδομένων, με στόχο την από κοινού βελτιστοποίηση του οικονομικού κόστους και της κατανάλωσης ενέργειας κατά την λειτουργία της συσκευής.

PROJECTS	JFScheduler	Nutch 1.1	Pamvotis 1.1	Violet 0.21.2	IceHockeyManager 0.4	MOEM
SLOC	23452	16810	5285	19965	27113	10671
#Classes	213	206	33	224	310	37
#Methods	1640	1050	299	1430	2320	513
Συνολικός #if/switch με >=2 διακλαδώσεις	268	200	74	107	280	250

Πίνακας 12 : Χαρακτηριστικά μεγέθους των υπό εξέταση συστημάτων ανοικτού λογισμικού

Επιπλέον τα project που χρησιμοποιήθηκαν για την αξιολόγηση της ποιότητας των αποτελεσμάτων, εξετάστηκαν με χρήση της έκδοσης 4.5 του Design Pattern Detection Tool (110) (111), προκειμένου να διαπιστωθεί κατά πόσο υπάρχει σχετική έλλειψη χρήσης του προτύπου στρατηγική ούτως ώστε να μπορούν να χρησιμοποιηθούν ως πειραματικά δεδομένα. Τα αποτελέσματα από την εκτέλεση του εργαλείου, αξιολογήθηκαν ακολούθως οπτικά προκειμένου να διαπιστωθεί, σε ποιο από τα δύο πρότυπα κατάσταση ή στρατηγική, αναφέρεται καθεμία από τις ανιχνευμένες περιπτώσεις, εφόσον το εργαλείο αναγνωρίζει την κοινή τους δομή, αλλά αδυνατεί να αναγνωρίσει τη διαφορά στη σημασιολογία τους και να πετύχει διαχωρισμό τους. Στον Πίνακας 13 απεικονίζονται συγκεντρωτικά τα αποτελέσματα της εκτέλεσης του εργαλείου για την ανίχνευση των προτύπων κατάσταση/στρατηγική.

Projects	IceHockeyManager	JFScheduler	MOEM
#Classes	310	213	37
#State/Strategy που ανιχνεύθηκαν	8	19[10]	0
#Περιπτώσεων που αφορούν Strategy	4	3	0

Πίνακας 13: Αποτελέσματα Design Pattern Detection Tool στα project που χρησιμοποιήθηκαν για την αξιολόγηση της ποιότητας

[10] Οι 12 από τις 19 περιπτώσεις αφορούν την ίδια κλάση (IRandomGenerator), η οποία χρησιμοποιείται μέσα σε 12 διαφορετικές context κλάσεις, συνεπώς ουσιαστικά πρόκειται για 8 διαφορετικές εμφανίσεις της δομής του προτύπου.

6.2. Μετρικές Αξιολόγησης

Αναφορικά με την αξιολόγηση των αποτελεσμάτων καθίσταται αναγκαίο να οριστούν οι μετρικές που χρησιμοποιήθηκαν καθώς και οι έννοιες με βάση τις οποίες πραγματοποιήθηκε η κατάταξη των αποτελεσμάτων. Ο χαρακτηρισμός των αποτελεσμάτων, ορίζεται ως:

- **True Positive**: Η περίπτωση υποθετικής λογικής που θεωρήθηκε από τον ανεξάρτητο αξιολογητή ως υποψήφια για την εφαρμογή αναδόμησης με το πρότυπο στρατηγική, και ανιχνεύθηκε και από την υλοποίηση της προτεινόμενης τεχνικής.
- **False Positive**: Η περίπτωση υποθετικής λογικής που ανιχνεύθηκε από την υλοποίηση της προτεινόμενης τεχνικής ως υποψήφια για την εφαρμογή αναδόμησης με το πρότυπο στρατηγική, αλλά δεν ανιχνεύθηκε από τον ανεξάρτητο αξιολογητή.
- **True Negative**: Η περίπτωση υποθετικής λογικής που δεν ανιχνεύθηκε ως υποψήφια για την εφαρμογή αναδόμησης με το πρότυπο στρατηγική, ούτε από τον ανεξάρτητο αξιολογητή, ούτε από την υλοποίηση της προτεινόμενης τεχνικής.
- **False Negative**: Η περίπτωση υποθετικής λογικής που δεν ανιχνεύθηκε ως υποψήφια για την εφαρμογή αναδόμησης με το πρότυπο στρατηγική, από την υλοποίηση της προτεινόμενης τεχνικής, αλλά θεωρήθηκε ως υποψήφια από τον ανεξάρτητο αξιολογητή.

Οι ανωτέρω χαρακτηρισμοί αξιοποιούνται από διάφορες μετρικές, προκειμένου να ποσοτικοποιηθεί η ποιότητα των αποτελεσμάτων, ούτως ώστε να επιτραπεί και η σύγκριση διάφορων μεθοδολογιών. Οι πλέον ευρέως χρησιμοποιούμενες μετρικές για το σκοπό αυτό είναι οι μετρικές της ακρίβειας (precision), της πληρότητας (recall) και της ορθότητας (accuracy).

Η ακρίβεια ορίζεται ως εξής:

> Ακρίβεια (precision) : Ο αριθμός των έγκυρων / ορθών προτάσεων αναδόμησης με στρατηγική που προέκυψαν ως αποτέλεσμα από την εφαρμογή της μεθοδολογίας δια το συνολικό πλήθος προτεινόμενων αναδομήσεων. Η μαθηματική έκφραση υπολογισμού της ακρίβειας είναι η εξής:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Κατ' αντιστοιχία, η πληρότητα ορίζεται ως εξής:

Πληρότητα (recall) : Ο λόγος του αριθμού των έγκυρων / ορθών προτάσεων αναδόμησης με στρατηγική που προέκυψαν ως αποτέλεσμα από την εφαρμογή της μεθοδολογίας προς το συνολικό πλήθος προτάσεων αναδομήσεων με στρατηγική που υπάρχουν στο σύστημα. Ο αντίστοιχος μαθηματικός τύπος είναι ο ακόλουθος:

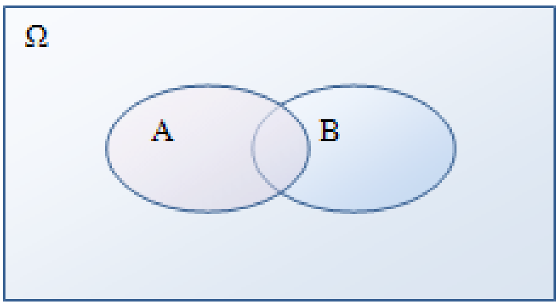
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Τέλος, η ορθότητα ορίζεται ως

Ορθότητα (accuracy) : Ο συνολικός αριθμός προτεινόμενων αναδομήσεων προς τη στρατηγική που είναι ορθός. Ο υπολογισμός της ορθότητας επιτυγχάνεται βάσει του τύπου:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

Για την πληρέστερη κατανόηση των αποτελεσμάτων παρατίθεται ακολούθως το διάγραμμα Venn που απεικονίζει τη σχέση των διαφόρων συνόλων, καθώς και η *μήτρα σύγχυσης (confusion matrix)* (**Error! Reference source not found.**) .



Εικόνα 49 : Διάγραμμα Venn

Όπου

Ω : Το σύνολο υποθετικών δηλώσεων που περιλαμβάνονται στο σύστημα λογισμικού.

Α: Το σύνολο των προτάσεων αναδομήσεων με στρατηγική που υπάρχουν στο σύστημα.

Β : Το σύνολο των έγκυρων / ορθών προτάσεων αναδόμησης που προέκυψαν ως αποτέλεσμα από την εφαρμογή της μεθοδολογίας.

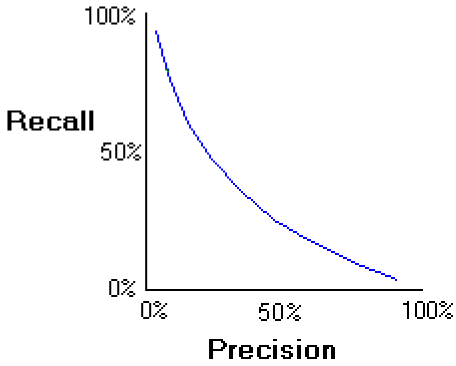
	Πραγματικές Αναδομήσεις	**Μη Πραγματικές Αναδομήσεις**
Προτεινόμενες Αναδομήσεις	True Positive	False Positive
Μη προτεινόμενες Αναδομήσεις	False Negative	True Negative

Πίνακας 14 : Μήτρα σύγχυσης διάφορων αποτελεσμάτων (confusion matrix)

6.3. Παρουσίαση Αποτελεσμάτων Πειραματικής Εκτέλεσης

Στην ενότητα αυτή παρουσιάζονται τα αποτελέσματα που προέκυψαν από την πειραματική εφαρμογή της προτεινόμενης τεχνικής στο σύνολο των δοκιμαστικών δεδομένων ούτως ώστε να εκτιμηθεί η απόδοση της μεθόδου από πλευράς πληρότητας και ακρίβειας, και να εξεταστεί η κλιμακωσιμότητα της αναφορικά με τον απαιτούμενο χρόνο ανίχνευσης και τις απαιτήσεις σε μνήμη.

Ως προς την απόδοση, η συγκεκριμένη μεθοδολογία αποσκοπεί στην επίτευξη της βέλτιστης δυνατής ποιότητας των αποτελεσμάτων, τόσο από πλευράς ορθότητας όσο και από πλευράς πληρότητας, προκειμένου ει δυνατόν η αξία της να είναι αναγνωρίσιμη όχι μόνο στα πλαίσια μιας ερευνητικής επιστημονικής μεθοδολογίας, αλλά και σε πρακτικό επίπεδο, παρέχοντας ένα χρήσιμο εργαλείο για τη διευκόλυνση της εφαρμογής της αναδόμησης. Ο στόχος αυτός μεταφράζεται στην επίτευξη υψηλού ποσοστού *ακρίβειας (precision)* και ταυτόχρονα υψηλού ποσοστού *πληρότητας (recall)*. Ωστόσο, η δυσκολία στην επίτευξη αυτού του στόχου έγκειται τόσο στις εγγενείς δυσκολίες της εφαρμογής της αναδόμησης, που σχετίζονται με τη δυσκολία στην ανίχνευση των πιθανών προβληματικών σημείων στον κώδικα, όσο και στην ίδια την αντιστρόφως ανάλογη σχέση μεταξύ των δύο μετρικών (Εικόνα 50) (112) (113), βάσει της οποίας αύξηση της ακρίβειας συνεπάγεται μείωση της πληρότητας, και αντιστρόφως.



Εικόνα 50 : Καμπύλη ακρίβειας και πληρότητας

Η αντιστρόφως ανάλογη αυτή σχέση αντανακλάται άμεσα στην επιλογή των κριτηρίων ανίχνευσης όπου

- η επιλογή λιγότερων και πιο γενικών («χαλαρών») κριτηρίων έχει ως αποτέλεσμα την αύξηση των προτεινόμενων αναδομήσεων, με αποτέλεσμα την ενδεχόμενη αύξηση της πληρότητας, αλλά και αντίστροφα την μείωση της ακρίβειας, εφόσον αυξάνεται ταυτόχρονα και ο αριθμός των προτάσεων που δεν αντιστοιχούν σε πραγματικές ευκαιρίες αναδόμησης.
- η επιλογή περισσότερων και πιο συγκεκριμένων («αυστηρών») κριτηρίων έχει ως αποτέλεσμα την μείωση των προτεινόμενων αναδομήσεων, με αποτέλεσμα την μείωση της πληρότητας, εφόσον ορισμένες πραγματικές αναδομήσεις μπορεί να μην ικανοποιούν τα συγκεκριμένα κριτήρια και συνεπώς να μην ανιχνευθούν. Αντίστροφα η ακρίβεια τείνει να αυξηθεί με την αντίστοιχη μείωση των λανθασμένων προτάσεων αναδόμησης που θα απορριφθούν από τα κριτήρια.

Από την ανωτέρω ανάλυση καθίσταται σαφές πώς η σχέση μεταξύ ακρίβειας και πληρότητας είναι αντίρροπη, συνεπώς καθίσταται δύσκολη η ταυτόχρονη επίτευξη υψηλών ποσοστών και στις δύο μετρικές. Κατά συνέπεια, στόχος της προτεινόμενης μεθοδολογίας είναι η επίτευξη των βέλτιστου εφικτού συνδυασμού πληρότητας και ακρίβειας, μέσω της επιλογής του βέλτιστου συνδυασμού κριτηρίων ανίχνευσης. Επιπλέον, βαρύτητα και στόχευση αποδίδεται στην επίτευξη του ελάχιστου δυνατού αριθμού υπαρχόντων δυνατοτήτων εφαρμογής της αναδόμησης προς τη στρατηγική στον κώδικα που δεν ανιχνεύονται. Η έμφαση που αποδίδεται στην επίτευξη αυτή, αιτιολογείται από το γεγονός ότι ο εντοπισμός υπαρχουσών προτάσεων αναδόμησης που δεν ανιχνεύθηκαν από την εκτέλεση του εργαλείου, με μη-αυτοματοποιημένο τρόπο παρουσιάζει μεγάλο βαθμό δυσκολίας, ενώ σε ορισμένες περιπτώσεις καθίσταται πρακτικά αδύνατος, κυρίως λόγω του αυξημένου μεγέθους του συστήματος και του κόστους που συνεπάγεται μια τέτοια ευρείας κλίμακας επιθεώρηση του πηγαίου κώδικα. Αντιθέτως, ο εντοπισμός ψευδών προτάσεων αναδόμησης καθίσταται εφικτός μέσω της δυνατότητας προεπισκόπησης των προτεινόμενων αναδομήσεων που παρέχει το εργαλείο στους χρήστες, οι οποίοι δύνανται κατ’ αυτό τον τρόπο να επιθεωρήσουν και να απορρίψουν ενδεχόμενες ψευδείς προτάσεις αναδόμησης που προκύπτουν στα αποτελέσματα βάσει της προσωπικής τους κρίσης και της γνώσης της λειτουργικότητας και των απαιτήσεων της εφαρμογής.

Ωστόσο, κατά την πρακτική εφαρμογή των κριτηρίων συναντήθηκε ιδιαίτερη δυσκολία στην ικανοποίηση της απαίτησης αυτής, καθώς είναι ιδιαίτερα δύσκολο να βρεθεί ένα σύνολο κριτηρίων το οποίο κατά την εφαρμογή του σε διαφορετικά συστήματα λογισμικού να παρουσιάζει ελάχιστες εξαιρέσεις, οι οποίες δεν ανιχνεύονται. Η δυσκολία της αντίστροφης σχέσης μεταξύ ακρίβειας και πληρότητας, συμπληρώνεται και από τη δυναμικότητα που παρουσιάζει η ανάπτυξη πηγαίου κώδικα, κατά την οποία δεν αποτελεί απαραίτητη προϋπόθεση η συμμόρφωση και ικανοποίηση συγκεκριμένων προγραμματιστικών προτύπων και πρακτικών. Κάθε προγραμματιστής είναι ελεύθερος να προγραμματίζει κατά βούληση, σύμφωνα με τη δική του κρίση, την προσωπική του εμπειρία και γνώση των σχετικών τεχνολογιών και το δικό του προγραμματιστικό στυλ. Το γεγονός αυτό συνεπάγεται ότι τα διάφορα συστήματα λογισμικού χαρακτηρίζονται από μεγάλη ετερογένεια αναφορικά με τη δομή του κώδικα. Συνεπώς είναι δύσκολο να ανιχνευθούν και να μοντελοποιηθούν ευρέως ακολουθούμενες συνήθεις κακές πρακτικές ανάπτυξης κώδικα που εν γένει πληρούνται στην πλειοψηφία των συστημάτων λογισμικού και χρήζουν αναδόμησης προς τη στρατηγική, ούτως ώστε να διασφαλιστεί η βέλτιστη τιμή της πληρότητας.

Ως προς την κλιμακωσιμότητα, στόχος του εργαλείου που αναπτύχθηκε είναι η αυτοματοποιημένη υποστήριξη αναδόμησης σε πραγματικά συστήματα λογισμικού από πλευράς μεγέθους και συνθετότητας. Το μέγεθος των συστημάτων λογισμικού γενικά διαφέρει, ωστόσο στα σύγχρονα συστήματα έχει διαπιστωθεί μια ανοδική τάση. Η δυνατότητα παραγωγής αποτελέσματος σε σύντομο χρόνο ακόμη και σε συστήματα που αποτελούνται από 30.000 γραμμές πηγαίου κώδικα θεωρήθηκε ένας εύλογος στόχος ως προς τις μη λειτουργικές απαιτήσεις του εργαλείου που αναπτύχθηκε.

6.3.1. Αποτελέσματα Αξιολόγησης Μεθοδολογίας

Ακολούθως παρουσιάζονται τα αποτελέσματα που προέκυψαν από την αξιολόγηση των τριών συστημάτων λογισμικού JFScheduler, IceHockeyManager και MOEM βάσει της κρίσης του ανεξάρτητου αξιολογητή της μεθοδολογίας, ο οποίος ταυτόχρονα, για τον χαρακτηρισμό των περιπτώσεων, επιστράτευσε τη γνώση του από την εφαρμογή των προτύπων σχεδίασης και την προγραμματιστική του εμπειρία. Βάσει των παραπάνω, ως υποψήφιες προς αναδόμηση θεωρήθηκαν έγκυρες οι περιπτώσεις στις οποίες η επιλογή της στρατηγικής συντελείται από μια εξωτερική κλάση πελάτη ακόμα και αν πρόκειται για περιορισμένου εύρους αλλαγές, οι οποίες δυνητικά σε ένα άλλο σύστημα λογισμικού θα μπορούσαν να επεκταθούν και να λάβουν ευρύτερη κλίμακα. Στον Πίνακας 15 παρουσιάζονται τα αποτελέσματα που προέκυψαν από την πειραματική εκτέλεση στα υπο εξέταση συστήματα ως προς την ακρίβεια και την πληρότητα, καθώς και οι απόλυτοι αριθμοί των περιπτώσεων true positive, false positive, true negative και false negative βάσει των οποίων υπολογίσθηκαν, ενώ στο Παράρτημα ΣΤ: Παραδείγματα Προτάσεων Αναδόμησης Πειραματικής Αξιολόγησης παρουσιάζονται ορισμένα παραδείγματα περιπτώσεων true positive και true negative.

Project	Precision	Recall	TP	FP	TN	FN	Σύνολο
JFScheduler	$\frac{6}{14}$ = **42%**	$\frac{6}{15}$ =**40%**	6	8	245	9	268
IceHockeyManager	$\frac{18}{30}$ = **60%**	$\frac{18}{26}$ = **69%**	18	12	242	8	280
MOEM	$\frac{9}{16}$ = **56.25%**	$\frac{9}{24}$ = **37.5%**	9	7	219	15	250

Πίνακας 15 : Αποτελέσματα αξιολόγησης της μεθοδολογίας στα υπο εξέταση συστήματα λογισμικού

Αναφορικά με τις υπάρχουσες προτάσεις αναδόμησης, τις οποίες η προτεινόμενη μεθοδολογία απέτυχε να ανιχνεύσει, πέραν από τον αριθμό τους, αξία από ερευνητικής άποψης, για λόγους κατανόησης και επαλήθευσης παράλληλα, της ορθότητας της υλοποίησης, θεωρείται ότι έχει και η εξέταση των αιτιών βάσει των οποίων απορρίφθηκαν από το σύνολο των υποψηφίων. Οι λόγοι σύμφωνα με τους οποίους προέκυψαν οι περιπτώσεις false negative, παρατίθενται συγκεντρωτικά και για τα τρία project στην ενότητα 6.3.4, ενώ μια περισσότερο ποιοτική εξέταση των αποτελεσμάτων περιλαμβάνεται στην ενότητα 6.3.3 όπου η χρησιμότητα της μεθοδολογίας που αναπτύχθηκε κρίνεται συγκριτικά με την μεθοδολογία αναδόμησης προς την κατάσταση ή τη στρατηγική του JDeodorant.

Συμπερασματικά, αξίζει να αναφερθεί ότι βάσει της διακύμανσης που εμφανίζουν οι τιμές της ακρίβειας και της πληρότητας στα υπό εξέταση project, ενδιαφέρον από ερευνητικής άποψης θα είχε η αξιολόγηση της προτεινόμενης μεθοδολογίας σε ένα ακόμα μεγαλύτερο δείγμα, το οποίο ωστόσο δεν ήταν εφικτό να πραγματοποιηθεί στα πλαίσια της εργασίας αυτής καθώς για την ασφαλή εξαγωγή συμπερασμάτων θα έπρεπε να μελετηθεί παράλληλα η ποιότητα αφενός και η επάρκεια του μεγέθους αφετέρου, του προς μελέτη στατιστικού δείγματος.

6.3.2. Αποτελέσματα Κλιμακωσιμότητας Μεθοδολογίας

Εξίσου σημαντικό ρόλο με την πληρότητα και την ακρίβεια της μεθοδολογίας διαδραματίζει η δυνατότητα απόκρισης σε εύλογο χρονικό διάστημα ως προς τις δυνατότητες αναδόμησης προς τη στρατηγική που περιλαμβάνονται σε συστήματα λογισμικού μεγάλου μεγέθους. Στην κατεύθυνση αυτή διενεργήθηκαν μετρήσεις του απαιτούμενου χρόνου ανίχνευσης σε συστήματα λογισμικού που διαφέρουν ως προς τα χαρακτηριστικά μεγέθους τους (αριθμός γραμμών εκτελέσιμου κώδικα SLOC, αριθμός κλάσεων, αριθμός μεθόδων, κλπ). Ακολούθως παρουσιάζονται τα αποτελέσματα της πειραματικής εκτέλεσης ως προς την κλιμακωσιμότητα. Τα αποτελέσματα αυτά μεταβάλλονται συναρτήσει των χαρακτηριστικών μεγέθους των υπό εξέταση συστημάτων λογισμικού, ενώ παράλληλα μεταβάλλονται και ανάλογα με τα χαρακτηριστικά του υπολογιστή, στον οποίο εκτελείται το εργαλείο αναδόμησης. Προκειμένου να αυξηθεί ακόμα περισσότερο ο βαθμός δυσκολίας του ελέγχου κλιμακωσιμότητας της μεθοδολογίας επιλέχθηκε ένας υπολογιστής, μικρών δυνατοτήτων, συγκριτικά με τις δυνατότητες που διαθέτουν οι σύγχρονοι υπολογιστές, τα χαρακτηριστικά του οποίου απεικονίζονται στον Πίνακας 16.

Επεξεργαστής	**Intel Core Duo T2500 2.00GHz**
Μνήμη	2.00GB DDR2 RAM

Πίνακας 16 : Χαρακτηριστικά υπολογιστή που χρησιμοποιήθηκε για τη διενέργεια της αξιολόγησης κλιμακωσιμότητας

Ως προς το χρόνο υπολογισμού (computation time) που απαιτείται για την ανίχνευση των προτεινόμενων αναδομήσεων κρίνεται αναγκαίο να εξεταστεί ξεχωριστά το χρονικό διάστημα που απαιτείται για την συντακτική ανάλυση του πηγαίου κώδικα και τη δημιουργία των απαραίτητων στατικών μορφών αναπαράστασης του πηγαίου κώδικα (AST δέντρο, γράφος CFG, γράφος PDG), από το χρονικό διάστημα που καταλαμβάνει η εξέταση των υποθετικών δομών που ανιχνεύθηκαν, βάσει των επιλεγμένων κριτηρίων. Η ανάγκη αυτή οφείλεται στο γεγονός ότι το χρονικό διάστημα που απαιτείται για τη στατική ανάλυση του πηγαίου κώδικα υπερβαίνει σαφώς του δεύτερου, δηλαδή του απαιτούμενου χρόνου για τον έλεγχο ικανοποίησης των κριτηρίων, καθώς σχετίζεται με το συνολικό αριθμό κλάσεων και εκτελέσιμων γραμμών πηγαίου κώδικα στο υπό εξέταση σύστημα. Αντιθέτως, στη δεύτερη περίπτωση εξετάζονται μόνο οι ανιχνευμένες υποθετικές δομές.

Αναφορικά με τη δημιουργία των μορφών αναπαράστασης καθίσταται αναγκαίο να τονιστεί πώς η δημιουργία του AST δέντρου διενεργείται σε οποιοδήποτε εξεταζόμενο σύστημα, μια φορά για ολόκληρο το λογισμικό, ενώ η εκτέλεση της είναι σαφώς διαχωρισμένη από την εξέταση των κριτηρίων και συγκεκριμένα προηγείται αυτής. Αντιθέτως, η δημιουργία των γράφων CFG και PDG εκτελείται μια φορά για καθεμία από τις μεθόδους του συστήματος, στις οποίες απαιτείται να εκτελεστεί σύμφωνα με τα κριτήρια της ανίχνευσης που ικανοποιούνται, ενώ η εκτέλεση της παρεμβάλλεται της εξέτασης των κριτηρίων ανίχνευσης. Συνεπώς, ο υπολογισμός του χρόνου για τη δημιουργία των CFG και PDG γράφων με απόλυτη ακρίβεια καθίσταται πιο δύσκολος.

Για τον υπολογισμό των χρόνων διεξαγωγής της στατικής ανάλυσης και του ελέγχου των κριτηρίων δεν χρησιμοποιήθηκε κάποιο framework ανάλυσης της ποιότητας του πηγαίου κώδικα, αλλά οι ίδιες οι εντολές της java με τις οποίες μπορεί να επιτευχθεί ο υπολογισμός του χρονικού διαστήματος υπολογισμού (παραδείγματος χάριν η εντολή System.*currentTimeMillis*()). Η μέθοδος υπολογισμού του χρόνου εκτέλεσης τοποθετήθηκε προσεκτικά στον κώδικα, ακριβώς πριν και έπειτα από τα τμήματα κώδικα, των οποίων η εκτέλεση έπρεπε να χρονομετρηθεί. Στον Πίνακας 17 παρουσιάζονται οι αντίστοιχοι χρόνοι για καθεμία από τις απαιτούμενες ενέργειες.

Συγκεκριμένα οι χρόνοι για τη δημιουργία των στατικών αναπαραστάσεων απεικονίζονται στις γραμμές *«Χρόνος Δημιουργίας AST δέντρου»* και *«Χρόνος Δημιουργίας CFG/PDG»*. Αξίζει να σημειωθεί ότι το AST κατασκευάζεται για ολόκληρο το σύστημα, ενώ οι γράφοι μόνο για περιπτώσεις υποθετικής δήλωσης με παραμέτρους που ανήκουν στο σύνολο των υποψηφίων. Στην γραμμή *«Χρόνος Ανίχνευσης»* εμφανίζεται συνολικά το υπόλοιπο χρονικό διάστημα εκτός της δημιουργίας του AST δέντρου, ενώ στη γραμμή *«Χρόνος ανίχνευσης κλήσεων μεθόδων»* παρατίθεται ο χρόνος που απαιτείται αποκλειστικά και μόνο για την ανάστροφη ανάλυση των κλήσεων μεθόδων για την ανίχνευση δυνατότητας πλήρους αντικατάστασης υποθετικής λογικής. Ο χρόνος αυτός εμφανίζεται και ξεχωριστά καθώς αποτελεί έναν από τους πλέον απαιτητικούς ελέγχους κριτήριων ανίχνευσης, εφόσον η ανάλυση του ξεπερνάει τα όρια μας κλάσης και επεκτείνεται σε επίπεδο συστήματος. Τέλος, στη γραμμή *«Συνολικός χρόνος»* του κάτωθι πίνακα εμφανίζεται το συνολικό χρονικό διάστημα που μεσολαβεί από τη χρονική στιγμή που ο χρήστης θα επιλέξει την ανίχνευση προτάσεων αναδόμησης ενός συστήματος λογισμικού μέχρι τη στιγμή που θα εμφανιστούν τα αποτελέσματα.

PROJECTS	Pamvotis 1.1	MOEM	Nutch 1.1	Violet	JFScheduler	IceHockey Manager 0.4
SLOC	5285	10671	16810	19965	23452	27113
#Classes	33	37	206	224	213	310
Συνολικός #if/switch με >=2 διακλαδώσεις	74	250	200	107	268	280
Χρόνος Δημιουργίας AST δέντρου	1782ms	2641ms	9938ms	10671 ms	11844ms	18188ms
Χρόνος Δημιουργίας CFG, PDG	0	1782 ms	0	0	831 ms	2467 ms
Χρόνος Ανίχνευσης κλήσεων μεθόδων	172ms	890 ms	297 ms	218 ms	2188 ms	2080 ms
Χρόνος Ανίχνευσης	1233 ms	4296 ms	4843 ms	4140 ms	6513 ms	10467 ms
Συνολικός Χρόνος	3015 ms	8719 ms	14781ms	15453ms	18828 ms	31141 ms

Πίνακας 17 : Χρόνος υπολογισμού για την ανίχνευση προτάσεων αναδόμησης προς τη στρατηγική σε ms

6.3.3. Σύγκριση Αποτελεσμάτων JDeodorant και Προτεινόμενης Μεθοδολογίας

Η άμεση σύγκριση των αποτελεσμάτων της υπάρχουσας μεθοδολογίας του JDeodorant και της προτεινόμενης μεθοδολογίας ως προς τις μετρικές της ακρίβειας και της πληρότητας δεν ενδείκνυται καθώς οι δύο μεθοδολογίες δεν εξετάζουν ακριβώς το ίδιο πρόβλημα, αλλά η δεύτερη αποτελεί εξειδίκευση της πρώτης αναφορικά με την αναδόμηση προς τη στρατηγική. Άλλωστε, οι δύο μεθοδολογίες δεν λειτουργούν ανταγωνιστικά η μια προς την άλλη, αλλά συμπληρωματικά για την από κοινού αντιμετώπιση του προβλήματος αναδόμησης πολύπλοκης υποθετικής λογικής.

Ωστόσο προκειμένου να καταστεί εμφανής η διαφοροποίηση των δύο μεθοδολογιών ως προς το σύνολο των αποτελεσμάτων που προκύπτουν και να αξιολογηθούν καλύτερα, θεωρήθηκε αναγκαίο να εντοπιστούν και να εξεταστούν συγκριτικά οι κοινές υποψήφιες περιπτώσεις υποθετικής λογικής προς αναδόμηση, με απώτερο στόχο την αποτίμηση του βαθμού επικάλυψης των δύο μεθοδολογιών και την ανάδειξη της συμβολής της προτεινόμενης μεθοδολογίας. Επιπλέον, εκτός από τον εντοπισμό των κοινών, θα πρέπει να εξεταστούν και οι μη-κοινές υποψήφιες περιπτώσεις υποθετικής λογικής, προκειμένου να διαπιστωθεί το ποσοστό των πραγματικών προτάσεων αναδόμησης προς τη στρατηγική που περιλαμβάνεται στις πρόσθετες προτάσεις που προέκυψαν με τη χρήση της προτεινόμενης μεθοδολογίας, καθώς και το ποσοστό εκείνο των υποψήφιων περιπτώσεων αναδόμησης που δεν ανιχνεύθηκαν από την προτεινόμενη προσέγγιση, ωστόσο εντοπίστηκαν από την υπάρχουσα υλοποίηση του JDeodorant. Εφεξής, στα πλαίσια της συγκεκριμένης ενότητας, για τη διευκόλυνση της κατανόησης από τον αναγνώστη, η υλοποίηση του JDeodorant αναφέρεται ως προσέγγιση/μεθοδολογία Α, ενώ η προσέγγιση που αναπτύχθηκε στα πλαίσια της εργασίας αυτής ως προσέγγιση/μεθοδολογία Β.

Σύμφωνα με τα παραπάνω θεωρήθηκε αναγκαίο να εντοπιστούν οι εξής περιπτώσεις ορθών προτάσεων αναδόμησης, βάσει των οποίων συντάχθηκε ο Πίνακας 19Πίνακας 18:

- Περίπτωση υποψήφιας υποθετικής λογικής για αναδόμηση με το πρότυπο στρατηγική που δεν ανιχνεύθηκε από την μεθοδολογία Β, ενώ ανιχνεύεται από την μεθοδολογία Α.
- Περίπτωση υποθετικής λογικής που προτάθηκε ως υποψήφια μόνο από την μεθοδολογία Β.

- Περίπτωση υποθετικής λογικής που προτάθηκε ως υποψήφια και από τις δύο μεθοδολογίες (A και B).

Για την εύρεση των κοινών προτάσεων αναδόμησης και οι δύο μεθοδολογίες εκτελέστηκαν στα υπο εξέταση project και εν συνεχεία για τον χαρακτηρισμό των προτάσεων αναδόμησης ως ορθών ή μη, χρησιμοποιήθηκαν οι χαρακτηρισμοί που είχαν αποδοθεί από τον αξιολογητή της μεθοδολογίας B κατά την εκτέλεση την αξιολόγησης που περιγράφεται στην ενότητα 6.3.1.

	MOEM	JFScheduler	IceHockeyManager
Αρ. ορθών προτάσεων αναδόμησης JDeodorant	0	0	0
Αρ. ορθών νέων προτάσεων αναδόμησης	9	4	15
Αρ. κοινών ορθών προτάσεων αναδόμησης	0	2	3

Πίνακας 18 : Αριθμός κοινών και μη κοινών προτάσεων αναδόμησης των δύο μεθοδολογιών

Σύμφωνα με τον Πίνακας 18,στο σύστημα MOEM δεν υπάρχει καμία επικάλυψη ανάμεσα στα αποτελέσματα των δύο μεθοδολογιών, ενώ η ανίχνευση των false positive περιπτώσεων της τεχνικής ανίχνευσης της μεθοδολογίας B δεν επιτυγχάνεται ούτε από την τεχνική της μεθοδολογίας A. Στο σύστημα MOEM οι δύο μεθοδολογίες συγκλίνουν ως προς τις false positive περιπτώσεις, ωστόσο η μεθοδολογία B επιτυγχάνει την ανίχνευση ορισμένων περιπτώσεων αναδόμησης προς τη στρατηγική που δεν ανιχνεύονταν προηγουμένως.

Σε αντίθεση με το MOEM που δεν υπήρχε καμία επικάλυψη μεταξύ των ορθών προτάσεων αναδόμησης των δύο μεθοδολογιών, στο JFScheduler προέκυψαν δύο κοινές προτάσεις. Και σ' αυτό το σύστημα, η εφαρμογή της προσέγγισης B είχε ως αποτέλεσμα τον εμπλουτισμό των προτάσεων αναδόμησης με 4 καινούργιες περιπτώσεις, ενώ παράλληλα η εφαρμογή των νέων κριτηρίων δεν προκάλεσε την απώλεια προτάσεων αναδόμησης που ανιχνεύονταν ήδη στη μεθοδολογία A.

Ολοκληρώνοντας, στο σύστημα IceHockeyManager παρατηρείται η μεγαλύτερη επικάλυψη μεταξύ των δύο μεθοδολογιών ως προς τις ορθές κοινές προτάσεις αναδόμησης σε σύγκριση και με τα υπόλοιπα συστήματα. Ωστόσο η προτεινόμενη τεχνική επιτυγχάνει επιπλέον των κοινών περιπτώσεων αρκετές καινούριες προτάσεις αναδόμησης, ο αριθμός των οποίων υπερέχει αισθητά του αριθμού των κοινών. Ως προς τις περιπτώσεις που αδυνατεί να ανιχνεύσει η προσέγγιση Β, η ίδια αδυναμία ανίχνευσης παρατηρείται και στην περίπτωση της προσέγγισης Α.

Αναφορικά με τις κοινές προτάσεις αναδόμησης που προκύπτουν από την εκτέλεση των δύο μεθοδολογιών ένα παράδειγμα υποθετικού κώδικα του συστήματος JFScheduler απεικονίζεται στην Εικόνα 51. Συγκεκριμένα στο παρόν τμήμα κώδικα λαμβάνεται η απόφαση για τον αλγόριθμο επεξεργασίας που θα εκτελεστεί ανάλογα με την τιμή του πεδίου *type* του γεγονότος *(AbstractEvent e)*. Η χρήση του λεκτικού type θα μπορούσε να επιφέρει σύγχυση σχετικά με την καταλληλότητα του προτύπου στρατηγική, έναντι του προτύπου κατάσταση (περίπτωση type checking). Ωστόσο, από τη μελέτη του κώδικα της κλάσης αλλά και εννοιολογικά, προκύπτει ότι ο τύπος ενός γεγονότος (πεδίο type) καθορίζεται μόνο μια φορά κατά την δημιουργία του και δεν τροποποιείται καθ’ όλη τη διάρκεια ζωής του. Απαραίτητη προϋπόθεση σύμφωνα με τον ορισμό του προτύπου κατάσταση αποτελεί η μετάβαση καταστάσεων της κλάσης context, σύμφωνα με την οποία η τιμή του πεδίου type θα έπρεπε να εναλλάσεται για να επιτρέπεται να εφαρμοστεί το πρότυπο αυτό. Αντιθέτως, το πρότυπο στρατηγική ενδείκνυται για τη συγκεκριμένη περίπτωση όπου ο αλγόριθμος επεξεργασίας είναι κοινός, ωστόσο παραλλάσσεται για κάθε διαφορετική τιμή του πεδίου type.

Η ανίχνευση της περίπτωσης που απεικονίζεται στην Εικόνα 51, σύμφωνα με τα κριτήρια που έχουν οριστεί στην μεθοδολογία Β, θα είχε απορριφθεί αν υπήρχαν εκχωρήσεις στο πεδίο type, καθιστώντας καταλληλότερο το πρότυπο κατάσταση, όπως συμβαίνει για το παράδειγμα της Εικόνα 52 που αποτελεί απόσπασμα από τον κώδικα του συστήματος IceHockeyManager. Στο παράδειγμα αυτό ανάλογα με την κατάσταση ενός αγώνα, κατά την κανονική διάρκεια του παιχνιδιού ή κατά την διάρκεια της παράτασης, εκτελείται η αντίστοιχη συμπεριφορά. Η κλάση Match περιλαμβάνει εκχωρήσεις στη μεταβλητή *currentState* προκειμένου να αλλάξει την κατάσταση της κλάσης όταν ο χρόνος κανονικής διάρκειας εκπνεύσει, γι’ αυτό και απορρίπτεται από τα κριτήρια ανίχνευσης της αναδόμησης προς τη στρατηγική (μεθοδολογία Β), ως κατάλληλη περίπτωση αναδόμησης προς την κατάσταση, ενώ ανιχνεύεται από τη μεθοδολογία Α. Συνεπώς, από τα παραπάνω παραδείγματα καθίσταται εμφανής η καταλληλότητα των κριτηρίων ανίχνευσης στον επιτυχή διαχωρισμό των προτάσεων αναδόμησης καθενός από τα πρότυπα κατάσταση και στρατηγική.

```
switch (e.getType()) {
        case FLOW_START:
                bPerformOptimization = processFlowStart((FlowEvent) e);
                break;
        case FLOW_TERM:
                bPerformOptimization = processFlowTerm((FlowEvent) e);
                break;
        case NETWORK_LOST:
                bPerformOptimization = processNetworkLost((NetworkEvent)e);
                break;
        case NETWORK_FOUND:
                bPerformOptimization =processNetworkFound((NetworkEvent)e);
                break;
        case NETWORK_STATUS_DETERIORATED:
                bPerformOptimization=
                processNetworkChanged((NetworkEvent)e, false);
                break;
        case NETWORK_STATUS_CHANGED:
                bPerformOptimization =
                processNetworkChanged((NetworkEvent)   e, true);
                break;
        case LOG_PERFORMANCE:
                bPerformOptimization = false;
                break;
        default:
                break;
}
```

Εικόνα 51 : Κοινή ανιχνεύσιμη πρόταση αναδόμησης στις δύο μεθοδολογίες, κατάλληλη για την εφαρμογή στρατηγικής

```
if (currentState == MATCH_STATE_REGULAR) {
    currentPeriod++;
    currentPeriodStarted = false;
    currentNextBreak += rules.secondsPerPeriod();
    initField();
} else if (currentState == MATCH_STATE_OVERTIME) {
    currentOvertimePeriod++;
    currentPeriodStarted = false;
    currentNextBreak += rules.secondsPerOvertime();
}
```

Εικόνα 52 : Παράδειγμα αναδόμησης προς τη στρατηγική που απορρίπτεται από τη μεθοδολογία Β

Αναφορικά με τις μη κοινές περιπτώσεις που ανιχνεύονται μόνο από τη μεθοδολογία Β ένα παράδειγμα φαίνεται στην Εικόνα 53. Στο συγκεκριμένο απόσπασμα κώδικα του συστήματος JFScheduler παράγεται μια τυχαία τιμή ανάλογα με την τιμή του ορίσματος κατανομής *(distr)*, που χρησιμοποιείται για την επιλογή του κατάλληλου αλγορίθμου μεταξύ της εκθετικής κατανομής (exponential), της λογαριθμικής κανονικής κατανομής (lognormal), κ.ο.κ. Η περίπτωση αυτή, η οποία αποτελεί έγκυρη περίπτωση αναδόμησης προς τη στρατηγική, απορρίπτεται από την μεθοδολογία Α, καθώς οι εκφράσεις case δεν περιλαμβάνουν σταθερές, ωστόσο ανιχνεύεται επιτυχώς από τη μεθοδολογία Β, καθώς τα κριτήρια της δεν ορίζουν ως αυστηρά υποχρεωτική την παρουσία σταθερών στις υποθετικές εκφράσεις, αλλά εστιάζουν στη χρήση παραμέτρων και πεδίων ώστε η στρατηγική να καθορίζεται εξωτερικά της κλάσης context από την κλάση πελάτη. Αντίστοιχη είναι και η περίπτωση που απεικονίζεται ακολούθως στην Εικόνα 54 σχετικά με την επιλογή μιας τυχαίας μεταβλητής ανάμεσα σε διαφορετικές παραλλαγές της κατανομής Poisson, η οποία επίσης λόγω της δομής των συνθηκών ανιχνεύεται μόνο από τον αλγόριθμο Β. Περισσότερα παραδείγματα κοινών και μη κοινών περιπτώσεων που ανιχνεύονται μόνο από την μεθοδολογία Β έχουν τοποθετηθεί στο Παράρτημα ΣΤ: Παραδείγματα Προτάσεων Αναδόμησης Πειραματικής Αξιολόγησης.

```
/** Generates a random variable obeying a specific distribution.
*
* @param distr
*   The distribution which the random variable will obey: 'c' for
*   constant, 'e' for exponential, 'p' for Pareto, 'l' for lognormal
*/
private float generateRandomVariable(char distr, float mean, float stDev,
                        float max, float min) {
        float rnd = 0;
        switch (distr) {
                case 'c': {
                        rnd = (int) mean; break;
                }
                case 'u': {
                        do {
                                rnd = mean * generator.nextFloat();
                        } while ((rnd > max) || (rnd < min));
                        break;
                }
                case 'e': // exponential
                {
                        do {
                                rnd = -(float) (mean *
                                Math.log(generator.nextDouble()));
                        } while ((rnd > max) || (rnd < min));
                        break;
                }
                case 'l': // lognormal
                {
                        double R1 = 0, R2 = 0, rNormal = 0, rLogNormal = 0, ml = 0, sl = 0;
                        // Generate uniform
                        do {
                                ml = 2 * java.lang.Math.log((double) mean)-
                                java.lang.Math.log(java.lang.Math.pow((double) stDev, 2) +
                                java.lang.Math.pow((double) mean, 2)) / 2;
                                sl = java.lang.Math.sqrt(-2
                                * java.lang.Math.log((double) mean)
                                + java.lang.Math.log(java.lang.Math.pow((double) stDev,2)
                                + java.lang.Math.pow((double) mean, 2)));
                                R1 = generator.nextDouble();
                                R2 = generator.nextDouble();
```

```
                    rNormal = ml + sl * java.lang.Math.cos(2 * 3.14 * R1)
                            * java.lang.Math.sqrt(-java.lang.Math.log(R2));
                    rLogNormal = java.lang.Math.exp(rNormal);
                } while ((rLogNormal >= max) || (rLogNormal <= min)); // truncate
                rnd = (float) rLogNormal;
                break;
            }
            case 'p': // Pareto
            {
                double alfa = 1.2d; // alfa parameter of pareto
                double b = 20 * 8;// (double)mean*(alfa-1d)/alfa;
                double d = 0;
                d = generator.nextDouble();
                // generate a pareto distributed number
                d = b / Math.pow(d, 1d / alfa);
                rnd = (float) (d + b);
                break;
            }
            default: {
                System.out.println("Distribution type configuration error.");
                System.exit(0);
            }
        } return rnd;
}
```

Εικόνα 53: Πρόταση αναδόμησης προς τη στρατηγική που ανιχνεύεται μόνο στην μεθοδολογία Β

```
/**  * Return a random number with Poisson distribution. */
public long nextLong() {
        // even though a Poisson distribution with mean 10 can be well approximated by a normal
        //distribution, use of the  standard algorithm for means up to 100
        if (mean < 100) {
            // See Knuth, TAOCP, vol. 2, second print
            // section 3.4.1, algorithm Q on page 117 Q1. [Calculate exponential]
            double p = Math.exp(-(double) mean);
            long N = 0;
            double q = 1.0;

            while (true) {
                // Q2. [Get uniform variable]
                double U = rand.nextDouble();
                // Q3. [Multiply]
                q = q * U;
                // Q4. [Test]
                if (q >= p)
                    N = N + 1;
                else
                    return N;
                }
            }
            // for larger mean values we approximate the Poisson distribution
            // using a normal distribution
            else {
                double z = rand.nextGaussian();
                long value = (long) (mean + z * Math.sqrt(mean) + 0.5);
                if (value >= 0)
                    return value;
                else
                    return 0;
            }
        }
```

```
}
```

Εικόνα 54 : Πρόταση αναδόμησης προς τη στρατηγική που ανιχνεύεται μόνο στην μεθοδολογία Β

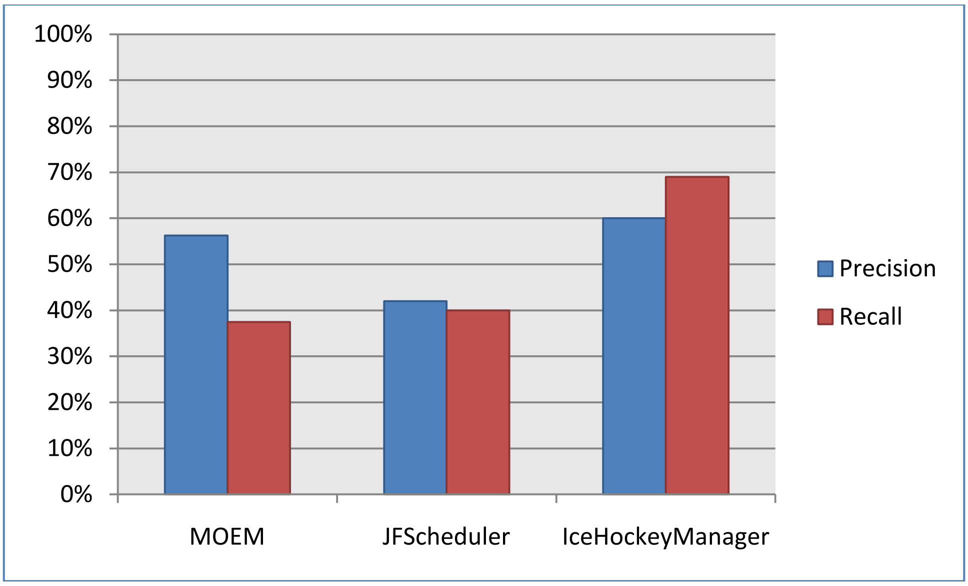
Συμπερασματικά, παρατηρείται πως η προσέγγιση Β (προτεινόμενη προσέγγιση) δρα συμπληρωματικά της μεθοδολογίας Α (υπάρχουσας μεθοδολογίας του JDeodorant) καθώς εμπλουτίζει το σύνολο των υπαρχουσών περιπτώσεων αναδόμησης προς τη στρατηγική που ανιχνεύονται, ενώ επιτυγχάνει το διαχωρισμό μεταξύ στρατηγικής και κατάστασης ως προς την αναδόμηση. Επιπλέον, η ανωτέρω ανάλυση αποδεικνύει πως η μεθοδολογία Β δεν υστερεί της μεθοδολογίας Α καθώς και στα τρία εξεταζόμενα project, ο αριθμός ορθών περιπτώσεων που ανιχνεύθηκαν μόνο από τα κριτήρια του JDeodorant είναι μηδενικός. Η ισχύς του γεγονότος αυτού παρότι υπονοείται έμμεσα, εφόσον η ανάπτυξη της μεθοδολογίας Β βασίστηκε στην μεθοδολογία Α, θα μπορούσε να είχε παραβιαστεί, αν το νέο σύνολο κριτηρίων είχε οδηγήσει στην απόρριψη περιπτώσεων που ανιχνεύονταν από τα κριτήρια που ορίζονται στο JDeodorant.

6.3.4. Συμπεράσματα Βάσει των Αποτελεσμάτων

Τα αποτελέσματα που προέκυψαν από την αξιολόγηση της μεθοδολογίας τόσο από πλευράς ακρίβειας και πληρότητας, όσο και από πλευράς κλιμακωσιμότητας της εφαρμοζόμενης τεχνικής ανίχνευσης χρήζουν περαιτέρω ανάλυσης.

Συγκεκριμένα, τα αποτελέσματα που προέκυψαν στα τρία εξεταζόμενα project (ραβδόγραμμα Εικόνα 55) παρουσιάζουν διακύμανση τόσο στην τιμή της ακρίβειας όσο και της πληρότητας. Η διακύμανση αυτή οφείλεται στο γεγονός ότι τα διάφορα συστήματα λογισμικού εμφανίζουν ετερογένεια όσον αφορά τη δομή του κώδικα, συνεπώς τα αποτελέσματα από την εφαρμογή του ίδιου συνόλου κριτηρίων ανίχνευσης είναι λογικό να διαφέρουν, ανάλογα με τον τρόπο δόμησης του εκάστοτε σχεδίου.

Επιπλέον, αναφορικά με τις περιπτώσεις υποθετικών δηλώσεων που χρήζουν αναδόμησης προς τη στρατηγική αλλά ωστόσο δεν ανιχνεύθηκαν, σε ορισμένες ο έλεγχος της υποθετικής έκφρασης στηριζόταν σε χρήση τοπικής μεταβλητής ή μεθόδου της ίδιας κλάσης χωρίς καμία παράμετρο, σε άλλες ο έλεγχος βασιζόταν σε πεδία που λάμβαναν τιμή εντός της κλάσης context, ενώ αρκετές ήταν και οι περιπτώσεις όπου οι διάφορες υποθετικές εκφράσεις μιας δήλωσης if αποτελούνταν από διαφορετικά σύνολα παραμέτρων και πεδίων.

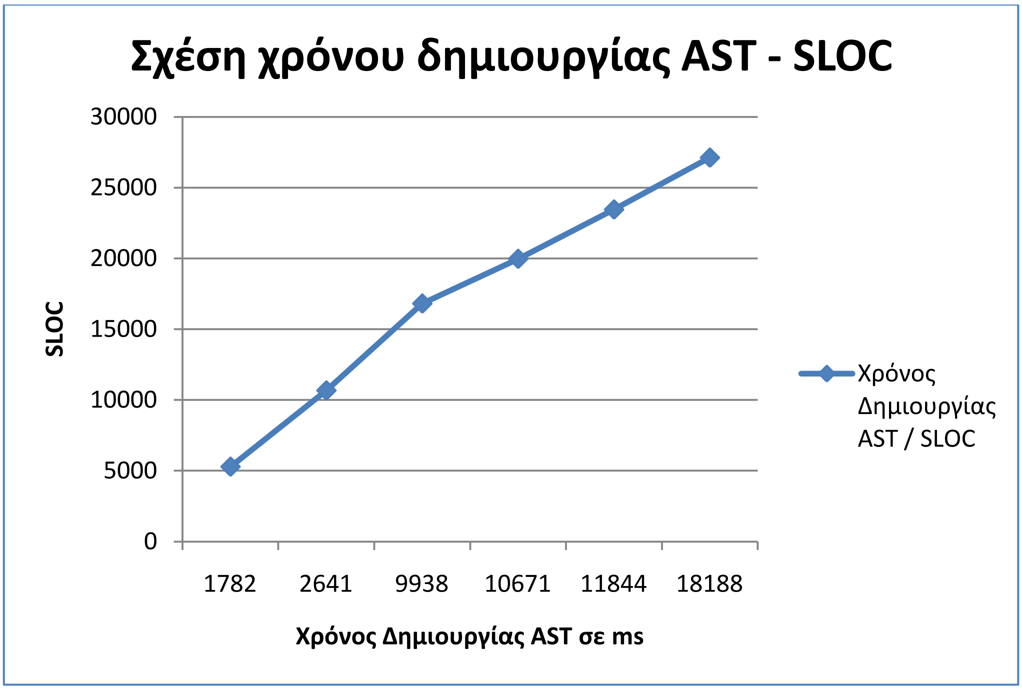


Εικόνα 55 : Ραβδόγραμμα ακρίβειας και πληρότητας των υπό εξέταση project

Κατά την διαδικασία επιλογής των κριτηρίων ανίχνευσης, δοκιμάστηκαν διάφορα σύνολα κριτηρίων. Ωστόσο, όπως μπορεί εύκολα να γίνει αντιληπτό η απενεργοποίηση ή αντίστοιχα ενεργοποίηση ενός επιπλέον κριτηρίου επιφέρει μεγάλες αλλαγές στα αποτελέσματα, οι οποίες λόγω της διαφορετικότητας των συστημάτων λογισμικού δεν ήταν κατ’ ανάγκη βελτιωτικές για όλα. Επιπλέον, λόγω της αντίρροπης σχέσης της ακρίβειας και της πληρότητας, ορισμένα σύνολα κριτηρίων που βελτίωναν αρκετά την τιμή της πληρότητας, είχαν αντίθετα ως αποτέλεσμα σημαντική μείωση στην τιμή της ακρίβειας.

Το συγκεκριμένο σύνολο κριτηρίων επιλέχθηκε καθώς προέκυπταν τα καλύτερα αποτελέσματα τόσο από πλευράς διακύμανσης των τιμών των μετρικών στα εξεταζόμενα συστήματα λογισμικού, όσο και από πλευράς ισορροπίας μεταξύ των τιμών της ακρίβειας και της πληρότητας.

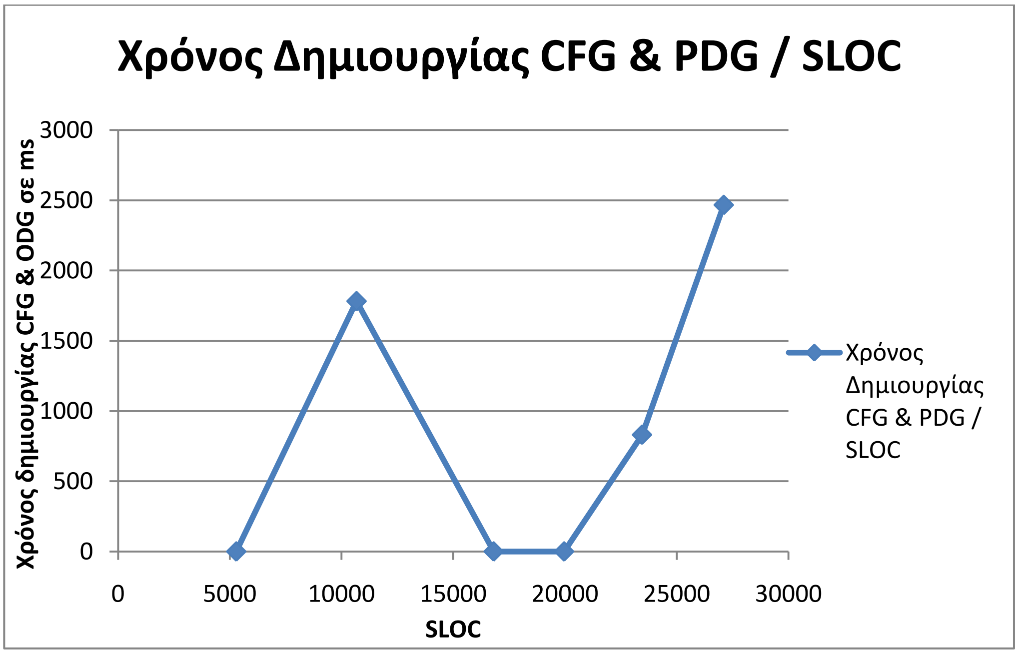
Αναφορικά με το χρόνο εκτέλεσης της ανίχνευσης βάσει του Πίνακας 18, αρχικά παρατηρείται ότι ο χρόνος δημιουργίας του AST δέντρου στην πλειοψηφία των περιπτώσεων υπερβαίνει σαφώς το συνολικό χρόνο που απαιτείται για τον έλεγχο των κριτηρίων στην ανίχνευση (γραμμή «Συνολικός χρόνος»). Συνεπώς, εφόσον ο χρόνος δημιουργίας του AST δέντρου είναι μεγαλύτερος, ασκεί και τη μεγαλύτερη επιρροή στη διαμόρφωση του συνολικού χρόνου που απαιτείται για την ανίχνευση. Συγκεκριμένα παρατηρείται πώς η αύξηση του μεγέθους των συστημάτων λογισμικού έχει ως αποτέλεσμα τη γραμμική αύξηση του χρόνου εκτέλεσης. Το γεγονός αυτό οφείλεται στην γραμμική σχέση που υπάρχει μεταξύ του χρόνου δημιουργίας AST δέντρου έναντι του συνολικού αριθμού γραμμών εκτελέσιμου κώδικα (SLOC) από τις οποίες αποτελείται ένα σύστημα λογισμικού (Εικόνα 56).



Εικόνα 56 : Γραφική παράσταση συσχέτισης χρόνου δημιουργίας AST με SLOC προγράμματος

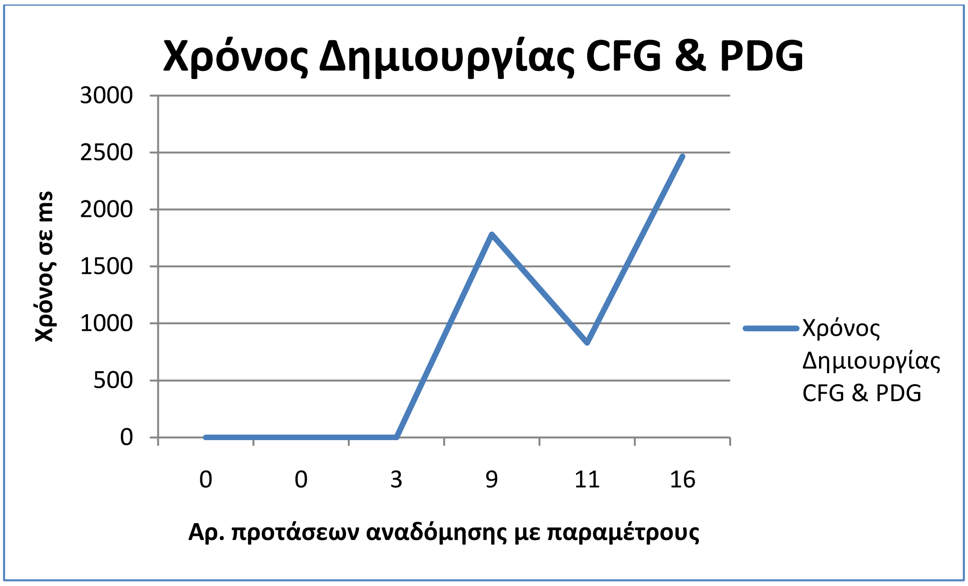
Η γραμμικότητα αυτή ωστόσο δεν ισχύει ούτε για το χρόνο δημιουργίας των γράφων CFG και PDG (Εικόνα 57), ούτε για τον χρόνο ελέγχου των υπόλοιπων κριτήριων (Εικόνα 58) καθώς οι χρόνοι αυτοί επηρεάζονται από το συνολικό αριθμό των υποθετικών δηλώσεων που συναντώνται στον κώδικα και απαιτούν τον έλεγχο των αντίστοιχων κριτηρίων, ο οποίος μπορεί να αυξομειώνεται άσχετα από το μέγεθος του. Συγκεκριμένα ο χρόνος δημιουργίας των γράφων CFG και PDG, επηρεάζεται αφενός από τον αριθμό των προτάσεων αναδόμησης (Εικόνα 59), οι οποίες περιλαμβάνουν παραμέτρους, αφετέρου όμως και από την πολυπλοκότητα τους, καθώς όσο αυξάνονται τα πιθανά μονοπάτια εκτέλεσης αυξάνεται και ο απαιτούμενος χρόνος ανάλυσης και δημιουργίας του γράφου PDG. Κατά συνέπεια, εφόσον ο χρόνος δημιουργίας του AST επηρεάζεται από παράγοντες εξωγενείς της μεθοδολογίας, οποιαδήποτε βελτίωση της απόδοσης του εργαλείου ανίχνευσης θα πρέπει να εστιάζει στη βελτιστοποίηση του αλγορίθμου ελέγχου κριτηρίων.

Ωστόσο, εφόσον η εκτέλεση των πειραμάτων απέδειξε ότι η απόκριση της μεθόδου σχετικά με την ανίχνευση παράγεται σε εύλογο χρόνο και μάλιστα ιδιαίτερα μικρό, της τάξης των 30sec στη χειρότερη περίπτωση όπου εκτελέστηκε σε σύστημα λογισμικού περίπου 30.000 γραμμών εκτελέσιμου κώδικα σε έκταση, η εφαρμοσιμότητα της μεθόδου σε ένα μεγάλο εύρος συστημάτων λογισμικού θεωρείται ότι μπορεί να επεκταθεί και σε ακόμα μεγαλύτερα συστήματα λογισμικού αποφέροντας αποτελέσματα πάλι σε εύλογο χρονικό διάστημα, ακόμα και χωρίς την περαιτέρω βελτιστοποίηση του αλγορίθμου. Πάραυτα, ειδικά για τη λήψη αποφάσεων βελτιστοποίησης τα ακριβή όρια του εργαλείου ανίχνευσης θα πρέπει να διερευνηθούν εκτενώς.



Εικόνα 57 : Γραφική παράσταση σχέσης χρόνου δημιουργίας γράφων CFG & PDG και SLOC

Εικόνα 58 : Γραφική παράσταση σχέσης χρόνου ελέγχου κριτηρίων και SLOC στο σύνολο των εξεταζόμενων συστημάτων λογισμικού



Εικόνα 59 : Γραφική παράσταση συσχέτισης χρόνου δημιουργίας γράφων CFG & PDG με τον αρ. προτάσεων αναδόμησης με παραμέτρους

Κεφάλαιο 7

Συμπεράσματα και Μελλοντική Έρευνα

Μια από τις κυριότερες εξελίξεις της προηγούμενης δεκαετίας στην τεχνολογία λογισμικού αποτελεί η ανάπτυξη και εξάπλωση των agile μεθοδολογιών ανάπτυξης και της εξελικτικής σχεδίασης, βασικό συστατικό των οποίων αποτελεί η αναδόμηση. Σύμφωνα με αυτές η αξιοποίηση της αναδόμησης δεν αναβάλλεται μέχρι τη φάση της συντήρησης όπως στις παραδοσιακές μεθοδολογίες ανάπτυξης, αλλά χρησιμοποιείται διαρκώς καθώς αποτελεί αναπόσπαστο κομμάτι της διαδικασίας ανάπτυξης που πραγματοποιείται σε επαναλήψεις για μικρά κάθε φορά τμήματα του συστήματος λογισμικού. Κατ' αυτό τον τρόπο μέσω των μεθοδολογιών αυτών, τα οφέλη της αναδόμησης βρίσκουν άμεσα εφαρμογή στη βελτίωση της ποιότητας του σχεδίου, το οποίο ως εκ τούτου υιοθετεί τις βέλτιστες προγραμματιστικές πρακτικές όπως είναι τα πρότυπα σχεδίασης, επικοινωνεί άμεσα το σκοπό του και πληροί τα χαρακτηριστικά ποιοτικού λογισμικού. Από την άλλη, στις παραδοσιακές μεθοδολογίες ανάπτυξης η αναδόμηση αποτελεί ένα πολύτιμο εργαλείο κατά τη φάση της συντήρησης για την αντιμετώπιση του προβλήματος της υποβάθμισης της ποιότητας του σχεδίου που έχει συντελεστεί έπειτα από τις διαρκείς προσθήκες και τροποποιήσεις που πραγματοποιούνται κατά την ανάπτυξη. Ωστόσο, και στις δύο προσεγγίσεις μεθοδολογιών ανάπτυξης, η αναδόμηση δεν είναι εφικτό να εφαρμοστεί χειροκίνητα, λόγω του αυξημένου μεγέθους του συντηρούμενου συστήματος και του αυξημένου κινδύνου εισαγωγής σφαλμάτων ή τροποποίησης της αρχικής συμπεριφοράς του,.

Κατά συνέπεια, καθίσταται αναγκαία η παροχή αυτοματοποιημένων εργαλείων που υποστηρίζουν διάφορους τρόπους αναδόμησης, και δη τους σύνθετους όπως οι αναδομήσεις με τα πρότυπα σχεδίασης, που ανταποκρίνονται καλύτερα στις απαιτήσεις των σύγχρονων συστημάτων λογισμικού. Η σημασία παροχής αυτοματοποιημένων εργαλείων αναδόμησης, τα οποία χαρακτηρίζονται από σχετική έλλειψη, αναδεικνύεται τόσο λόγω της αύξησης της ταχύτητας και της απλοποίησης των εκτελούμενων μετασχηματισμών, μειώνοντας άμεσα το κόστος της συντήρησης, όσο και επειδή διασφαλίζεται η ορθότητα τους, ενώ παράλληλα επιτυγχάνεται η διατήρηση της συμπεριφοράς του αρχικού συστήματος.

Στην παρούσα ερευνητική εργασία μελετήθηκε το πρόβλημα αυτοματοποίησης της αναδόμησης προς τη στρατηγική, όπου η επιλογή της υπό εκτέλεσης στρατηγικής, συντελείται από την κλάση πελάτη. Βάση για την υλοποίηση της παρούσας εργασίας αποτέλεσε η προσέγγιση ανίχνευσης περιπτώσεων αναδόμησης που εισάγουν πολυμορφισμό γενικά, και τα πρότυπα σχεδίασης κατάσταση και στρατηγική ειδικά, που προδιαγράφεται στο (1). Ωστόσο το εφαλτήριο της ανάπτυξης της προσέγγισης αυτής δόθηκε από τις διαφορές που προκύπτουν από την προσεκτική σύγκριση των δύο προτύπων, από τις οποίες πηγάζει και η ανάγκη ξεχωριστής αντιμετώπισης τους κατά την αναδόμηση. Στόχος της παρούσας διπλωματικής αποτελεί η αξιοποίηση των διαφορών των δύο προτύπων, ούτως ώστε να επιτευχθεί ο διαχωρισμός τους αναφορικά με την αναδόμηση. Για το σκοπό αυτό η προτεινόμενη προσέγγιση εστιάζει στον καθορισμό ενός πιο κατάλληλου συνόλου κριτηρίων ανίχνευσης, ειδικά επιλεγμένων για την περίπτωση της αναδόμησης προς τη στρατηγική, με αποτέλεσμα την ανίχνευση περιπτώσεων που οι υπάρχουσες προσεγγίσεις αδυνατούσαν να εντοπίσουν.

Στα πλαίσια αυτά αναπτύχθηκε ένα σύνολο κριτηρίων σε επίπεδο μεθόδου, συνθήκης και διακλάδωσης, ο έλεγχος της ικανοποιησιμότητας των οποίων απαιτεί την εξέταση όχι μόνο των υποθετικών δηλώσεων αυτών καθαυτών αλλά και των μεθόδων στις οποίες εντάσσονται, καθώς και των κλήσεων τους. Η μελέτη των κλήσεων των μεθόδων αποτελεί απαραίτητη προϋπόθεση για την επίτευξη της κυριότερης συμβολής της παρούσας διπλωματικής εργασίας, της ανίχνευσης δυνατότητας ολικής απαλοιφής της υποθετικής δήλωσης κατά την αναδόμηση. Η ολική απαλοιφή καθίσταται εφικτή σε περιπτώσεις όπου οι τιμές των ορισμάτων των κλήσεων μεθόδου είναι γνωστές κατά το χρόνο μεταγλώττισης, περιπτώσεις οι οποίες παρότι δεν εμφανίζουν μεγάλη συχνότητα εμφάνισης, εντούτοις έχουν μεγάλη αξία καθώς μειώνεται άμεσα η πολυπλοκότητα του συστήματος.

Για την διεξαγωγή του ελέγχου του συνόλου των προτεινόμενων κριτηρίων, το εργαλείο που υλοποιήθηκε και ενσωματώθηκε ως τμήμα του JDeodorant, βασίστηκε πρωταρχικά στη χρήση AST δέντρων και συμπληρωματικά στο συνδυασμό CFG γράφων και PDG γράφων των μεθόδων με τις υποθετικές δηλώσεις. Με τον τρόπο αυτό θεωρείται ότι επιτυγχάνονται πληρέστερα και ακριβέστερα αποτελέσματα σχετικά με τις προτάσεις αναδόμησης προς τη στρατηγική, καθώς επιτρέπεται ο εντοπισμός περιπτώσεων για την ανίχνευση των οποίων οι πληροφορίες που προκύπτουν από την διάσχιση του AST δέντρου δεν ήταν εφικτή.

Η προτεινόμενη μεθοδολογία τέθηκε σε εφαρμογή σε ένα σύνολο δοκιμαστικών συστημάτων ανοικτού λογισμικού και εξετάστηκε τόσο από πλευράς κλιμακωσιμότητας όσο και ορθότητας των αποτελεσμάτων της ανίχνευσης. Η εκτέλεση των πειραμάτων απέδειξε την εφαρμοσιμότητα της μεθόδου σε ένα μεγάλο εύρος συστημάτων λογισμικού, καθώς η απόκριση της σχετικά με την ανίχνευση παράγεται σε εύλογο χρόνο της τάξης των 30sec σε διάφορα συστήματα λογισμικού κλιμακούμενου μεγέθους αποτελούμενα το μέγιστο από 30.000 γραμμές εκτελέσιμου κώδικα, ο οποίος, παρότι τα ακριβή όρια του εργαλείου ανίχνευσης πρόκειται να διερευνηθούν μελλοντικά για την περαιτέρω βελτιστοποίηση του αλγορίθμου, αναμένεται να παραμείνει εύλογος ακόμα και σε συστήματα λογισμικού μεγαλύτερου μεγέθους, καθώς το χρονικό διάστημα που προέκυψε από τους συγκεκριμένους υπολογισμούς είναι ιδιαίτερα μικρό.

Η πειραματική αξιολόγηση της μεθοδολογίας ως προς την ορθότητα των προτάσεων αναδόμησης, εφαρμόστηκε σε ένα σύνολο τριών συστημάτων ανοικτού λογισμικού (IceHockeyManager, JFScheduler, MOEM), γνωστής και άγνωστης λειτουργικότητας στον ανεξάρτητο αξιολογητή που ανέλαβε την διεκπεραίωση της διαδικασίας. Τα αποτελέσματα που προέκυψαν αποδεικνύουν τη χρησιμότητα της μεθοδολογίας ως συμπληρωματική στην ανίχνευση προτάσεων αναδόμησης οι οποίες δεν ανιχνεύονταν από την υπάρχουσα μεθοδολογία που περιγράφεται στο (1), καθώς και την καταλληλότητα των κριτηρίων ανίχνευσης για την διαφοροποίηση των προτάσεων αναδόμησης προς τη στρατηγική από την κατάσταση. Η αξιοπιστία της μεθοδολογίας αναλογικά με τη γενικευσιμότητα της σε ένα μεγάλο εύρος συστημάτων λογισμικού, αποτελεί ένα ενδιαφέρον ερευνητικό ερώτημα προς μελλοντική διερεύνηση, το οποίο σχετίζεται με άλλα ερευνητικά ζητήματα όπως παραδείγματος χάριν ο τρόπος με τον οποίο διασφαλίζεται η ποιότητα αφενός και η επάρκεια του μεγέθους αφετέρου, του προς μελέτη στατιστικού δείγματος για την ασφαλή εξαγωγή συμπερασμάτων.

Επιπλέον, με την ολοκλήρωση της υλοποίησης της προτεινόμενης μεθοδολογίας σήμανε παράλληλα και η γέννηση αρκετών ενδιαφέροντων ερευνητικών ζητημάτων που χρήζουν περαιτέρω διερεύνησης στην ίδια κατεύθυνση. Πολλά από τα ζητήματα αυτά πηγάζουν άμεσα από τους περιορισμούς και τις ελλείψεις της προτεινόμενης μεθοδολογίας που παρουσιάστηκαν ενδελεχώς σε προηγούμενη ενότητα και η επανάληψη τους κρίνεται πλεονάζουσα. Επί του παρόντος, προτείνονται ορισμένα ενδιαφέροντα ερευνητικά ζητήματα που προέκυψαν κατά την ανάπτυξη της μεθοδολογίας αυτής και η διερεύνηση τους δε μπόρεσε να ενταχθεί στα πλαίσια της παρούσας εργασίας.

Συγκεκριμένα ενδιαφέρον από ερευνητικής άποψης παρουσιάζει η μελέτη και εξεύρεση περισσότερων περιπτώσεων αναδόμησης προς τη στρατηγική στις οποίες η υποθετική λογική δύναται να εξαλειφθεί πλήρως. Η προτεινόμενη μεθοδολογία εξετάζει τη γνώση των ορισμάτων κλήσης πρωταρχικού ή απαριθμητού τύπου απευθείας στην κλήση της μεθόδου. Η επέκταση του ελέγχου στα πλαίσια του συνόλου των boundary blocks της κλήσης της μεθόδου θεωρείται ότι θα μπορούσε να αξιοποιηθεί για την εξεύρεση ακόμα περισσότερων περιπτώσεων με δυνατότητα ολικής απαλοιφής.

Επιπλέον, η ανάπτυξη της προτεινόμενης μεθοδολογίας περιορίστηκε στη μελέτη ενός σχετικά μικρού αριθμού συστημάτων λογισμικού για την εξαγωγή συμπερασμάτων σχετικά με την πιθανή δομή των δηλώσεων if και switch στις οποίες απουσιάζει η χρήση του προτύπου στρατηγικής. Συνεπώς, προκειμένου να εμπλουτιστεί η προτεινόμενη μεθοδολογία και να διαπιστωθεί η γενίκευση των προτεινόμενων δομών στην πλειοψηφία των συστημάτων λογισμικού που αναπτύσσονται, καθίσταται αναγκαίο να διεξαχθεί μια μελέτη σε ευρύτερο σύνολο συστημάτων από διαφορετικούς τομείς εφαρμογών, ούτως ώστε να εντοπιστούν οι συνηθέστερες δομές των υποθετικών δηλώσεων για τις οποίες ενδείκνυται το πρότυπο στρατηγική. Η στατιστική εκτίμηση της εμφάνισης τους μπορεί στη συνέχεια να χρησιμοποιηθεί και ως βάση κατά την βαθμονόμηση των προτάσεων αναδόμησης μέσω της απόδοσης καταλληλότερων βαρών.

Μια ακόμη ενδιαφέρουσα ερευνητική μελέτη στην ίδια κατεύθυνση θα περιλάμβανε τη συγκέντρωση στατιστικών στοιχείων μέσω αρχείων καταγραφής (log) αναφορικά με τις προτάσεις αναδόμησης που προέκυψαν από το εργαλείο που αναπτύχθηκε και οι οποίες τελικά επιλέχθηκαν προς εφαρμογή από τους χρήστες του συστήματος. Τα στοιχεία αυτά θα μπορούσαν στη συνέχεια να αναλυθούν περαιτέρω και να χρησιμοποιηθούν για την εξαγωγή χρήσιμων συμπερασμάτων σχετικά με τις συνηθέστερες δομές που ακολουθούν οι προτάσεις αναδόμησης, ούτως ώστε να βελτιωθεί αφενός η προτεινόμενη μεθοδολογία ανίχνευσης μέσω του επαναπροσδιορισμού του πλέον καταλληλότερου συνόλου κριτηρίων, και αφετέρου να προσαρμοστούν ανάλογα τα βάρη που αποδίδονται από τη μεθοδολογία.

Η προσαρμογή των βαρών ώστε να συνάδουν με τα χαρακτηριστικά κάθε συστήματος λογισμικού και κάθε διαφορετικής ομάδας ανάπτυξης θεωρείται πώς θα βελτίωνε επιπλέον την αποδοτικότητα της μεθόδου, καθώς δεν υπάρχουν κάποιες ευρέως χρησιμοποιούμενες «κακές πρακτικές» κατ' αντιστοιχία με τις γνωστές καλές πρακτικές ανάπτυξης λογισμικού. Οι κακές πρακτικές που ακολουθεί κάθε προγραμματιστής εξαρτώνται από ενδεχόμενες προσωπικές λανθασμένες αντιλήψεις ή την πιθανή έλλειψη γνώσεων σχετικά με τα πρότυπα σχεδίασης ειδικά, καθώς και την έλλειψη εξοικείωσης αναφορικά με άλλες γνωστές καλές προγραμματιστικές τεχνικές.

Συνεπώς, μια επιπλέον πρόταση προς βελτίωση αποτελεί η εξειδικευμένη προσαρμογή των βαρών, η οποία θα μπορούσε να εκτελείται ξεχωριστά σε κάθε project μετά από την εκτέλεση ενός συγκεκριμένου αριθμού αναδομήσεων που θα λειτουργούσαν ως δεδομένα εκπαίδευσης (training set). Κατ' αυτό τον τρόπο η ταξινόμηση των προτάσεων αναδόμησης θα προσαρμοζόταν στη φύση του κάθε προγράμματος και στο προγραμματιστικό στυλ κάθε διαφορετικής ομάδας ανάπτυξης, επιτυγχάνοντας καλύτερα και πιο αντιπροσωπευτικά αποτελέσματα.

Εν κατακλείδι, η παρούσα διπλωματική εργασία συμβάλει στην αυτοματοποίηση της αναδόμησης μελετώντας εις βάθος τις ιδιαιτερότητες της αναδόμησης προς τη στρατηγική. Η επιλογή αυτή έναντι της επιλογής ανάπτυξης μιας γενικής μεθοδολογίας που θα απευθυνόταν σε αναδομήσεις περισσότερων προτύπων σχεδίασης, δεν συντελέστηκε τυχαία. Την παροχή αυτοματοποιημένης υποστήριξης πολλών διαφορετικών αναδομήσεων με πρότυπα σχεδίασης παρακωλύουν αρκετοί παράγοντες και ιδιαίτερα, οι δομικές ιδιαιτερότητες κάθε σχεδιαστικού προβλήματος, αλλά και η διαφοροποίηση του σκοπού των προτύπων σχεδίασης, που ουσιαστικά υπονοεί την ανάγκη επίλυσης διαφορετικών προβλημάτων και η οποία αποτέλεσε την αφορμή για την ανάπτυξη της παρούσας διπλωματικής. Επιπλέον καθίσταται αναγκαίο να αποσαφηνιστεί ότι εξαιτίας της εννοιολογικής φύσης των προβλημάτων στα οποία καλούνται να παράσχουν λύση τα πρότυπα σχεδίασης, η συμμετοχή του ανθρώπινου παράγοντα στη διαδικασία της αναδόμησης, με τις υπάρχουσες δυνατότητες της τεχνολογίας θεωρείται αδύνατο να απαλειφθεί πλήρως. Συνεπώς, στόχος των αυτοματοποιημένων εργαλείων αναδόμησης αποτελεί η λήψη υποστηρικτικού ρόλου και η αυτοματοποίηση τόσο της ανίχνευσης όσο και της εφαρμογής των απαραίτητων μετασχηματισμών, κατά τρόπο ώστε να ελαχιστοποιηθεί ο βαθμός αλληλεπίδρασης με τον ανθρώπινο παράγοντα. Στην κατεύθυνση αυτή έμφαση δίνεται στην επίτευξη του πληρέστερου και ακριβέστερου δυνατού συνόλου προτάσεων αναδόμησης, στην οποία συμβάλει και η παρούσα διπλωματική, ενώ για την επίτευξη της κατά το δυνατόν μεγαλύτερης κάλυψης του συνόλου των διαφορετικών τρόπων αναδόμησης, προτείνεται ο κατάλληλος συνδυασμός διαφορετικών προσεγγίσεων και μεθοδολογιών.

ΠΑΡΑΡΤΗΜΑ

Παράρτημα Α : Παράδειγμα Χρήσης του Προτύπου Σχεδίασης Κατάσταση

```
public class Alarm {

    private AlarmState state = new StandByState();
    private Siren siren = new Siren();

    public void arm() {
        state.arm(this);
    }

    public void disarm() {
        state.disarm(this);
    }

    public void sirenTest() {
        state.sirenTest(this);
    }

    public void intrusion() {
        state.intrusion(this);
    }

    public void deactivate() {
        state.deactivate(this);
    }

    void setState(AlarmState state) {
        this.state = state;
    }

    Siren getSiren() {
        return siren;
    }

    boolean hasClosedWindows() {
        return true;
    }
}
```

Εικόνα 60 : Η κλάση Alarm

```
public class Siren {
    public void start() {}

    public void stop() {}
}
```

Εικόνα 61 : Η κλάση Siren

```java
public class AlarmState {

    public void arm(Alarm alarm) {}

    public void disarm(Alarm alarm) {}

    public void sirenTest(Alarm alarm) {}

    public void intrusion(Alarm alarm) {}

    public void deactivate(Alarm alarm) {}
}
```

Εικόνα 62 : Η κλάση AlarmState

```java
public class ArmedState extends AlarmState {
    @Override
    public void intrusion(Alarm alarm) {
        alarm.getSiren().start();
        alarm.setState(new ActiveState());
    }

    @Override
    public void disarm(Alarm alarm) {
        alarm.setState(new StandByState());
    }

}
```

Εικόνα 63 : Η κλάση ArmedState

```java
public class ActiveState extends AlarmState {
    @Override
    public void deactivate(Alarm alarm) {
        alarm.getSiren().stop();
        alarm.setState(new StandByState());
    }
}
```

Εικόνα 64 : Η κλάση ActiveState

```java
public class StandByState extends AlarmState {
    @Override
    public void intrusion(Alarm alarm) {
        alarm.getSiren().start();
        alarm.setState(new ActiveState());
    }

    @Override
    public void disarm(Alarm alarm) {
        alarm.setState(new StandByState());
    }
}
```

Εικόνα 65 : Η κλάση StandByState

Παράρτημα Β : Παράδειγμα Χρήσης του Προτύπου Σχεδίασης Στρατηγική

```java
public class SortingContext {
    private SortingStrategy sortingStrategy = null;

    public void sortDouble(double[] list) {
        sortingStrategy.sort(list);
    }

    public SortingStrategy getSortingStrategy() {
     return sortingStrategy;
    }

    public void setSortingStrategy(SortingStrategy sortingStrategy) {
     this.sortingStrategy = sortingStrategy;
    }
}
```

Εικόνα 66 : Η κλάση SotringContext

```java
public interface SortingStrategy {
    public void sort(double[] list);
}
```

Εικόνα 67: Η κλάση SortingStrategy

```java
public class BubbleSort implements SortingStrategy {
    public void sort(double[] list) {
        double temp;
        for (int i = 0; i < list.length; i++) {
            for (int j = 0; j < list.length - i; j++) {
                if (list[i] < list[j]) {
                    temp = list[i];
                    list[i] = list[j];
                    list[j] = temp;
                }
            }
        }
    }
}
```

Εικόνα 68: Η κλάση BubbleSort

```
public class QuickSort implements SortingStrategy {
    public void sort(double[] u) {
        sort(u, 0, u.length - 1);
    }

    private void sort(double[] a, int left, int right) {
        if (right <= left)
            return;
        int i = part(a, left, right);
        sort(a, left, i - 1);
        sort(a, i + 1, right);
    }

    private int part(double[] a, int left, int right) {
        int i = left;
        int j = right;
        while (true) {
            while (a[i] < a[right])
                i++;
            while (smaller(a[right], a[--j]))
                if (j == left)
                    break;
            if (i >= j)
                break;
            swap(a, i, j);
        }
        swap(a, i, right);
        return i;
    }

    private boolean smaller(double x, double y) {
        return (x < y);
    }

    private void swap(double[] a, int i, int j) {
        double swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }
}
```

Εικόνα 69 : Η κλάση QuickSort

Παράρτημα Γ : Αλγόριθμος

```
/*analysis of conditional logic based on condition level criteria */
If conditional statement is switch
Then do
        Check if condition contains only parameters and fields set outside context
EndIF
If conditional statement is if
Then do
    For every condition in statement
        Check if condition contains instanceof and set RTTI flag
        Check if all condition contains parameters and/or fields
        Check if params/fields of current condition are the same with the ones in the first condition
  EndFor
EndIF
 /* check criterion for conditionals in private methods approval */
@see check private method criterion
check private method criterion and set invalid
```

```
/* gather branch information that will be used for scoring */
If method not void
Then do
        /* check scoring criterion of valid return statements in all branches */
        For every branch of conditional
        Do
                If branch contains return statement
                Then
                    Check returned expression is method invocation, variable or infix expression
                EndIF
        EndFor
EndIF
/* check scoring criterion regarding one field assignment in every branch */
Find all field assignments with equality operator and all setter method invocations contained in the first branch
For every branch of the conditional starting from the second branch
Do
     Find all field assignments with equality operator and all setter method
     Keep only the intersection of fields between the current and the first branch
EndFor
Check if only one field finally exists in the intersection
```

```
/* check scoring criterion of same method signatures resulting from method extraction on every branch */
For every branch
Do
    Find and save all local variables used in branch
    Find and save all parameters used in branch
EndFor
For every branch
Do
    Check variables are the same with the variables of next branch or else save their union removing duplicates
    Check parameters are the same with those of next branch or else save their union removing duplicates
EndFor
```

Εικόνα 70 : α) Αλγόριθμος ανάλυσης υποθετικών εκφράσεων, β) Αλγόριθμος ανάλυσης διακλαδώσεων

```
check valid sized conditional statement
If size criterion is active
Then Do
     return
     (average statements in branch > 2 AND statement does not have
     two branches, the one of which has only 1 instruction which is
     not a method invocation of the same class)
     OR
     (average statements in branch <= 2 AND (NOT branches contain
     different method invocations of the same class in at least the half
     of the branches OR branches contain creation methods))

EndIF
```

Εικόνα 71 : Έλεγχος κριτηρίου μεγέθους των υποθετικών δηλώσεων

Παράρτημα Δ : Υλοποίηση Προτεινόμενης Μεθοδολογίας

Η υλοποίηση της προτεινόμενης μεθοδολογίας στηρίχθηκε όπως προαναφέρθηκε στην ήδη υπάρχουσα υλοποίηση του JDeodorant. Πολλές από τις κλάσεις του JDeodorant επαναχρησιμοποιήθηκαν ενώ ορίστηκαν και νέες κλάσεις για την υλοποίηση της προτεινόμενης μεθοδολογίας αναδόμησης προς τη στρατηγική. Συγκεκριμένα οι κλάσεις που χρησιμοποιήθηκαν ή δημιουργήθηκαν εκ νέου αποτελούν τμήμα των πακέτων που απεικονίζονται στο σχετικό διάγραμμα (Εικόνα 75) το οποίο δημιουργήθηκε με χρήση του εργαλείου IBM Software Architect (IBM Rational Software Architect).

Στο πακέτο ***jdeodorant.refactoring.views*** περιλαμβάνονται όλες οι κλάσεις που αφορούν το γραφικό περιβάλλον του JDeodorant (Εικόνα 78) με εξαίρεση την κλάση που μοντελοποιεί το αρχικό μενού “Bad Smells”, η οποία βρίσκεται στο πακέτο ***jdeodorant.refactoring.actions***. Στο πακέτο ***jdeodorant.refactoring.manipulators*** εμπεριέχονται όλες οι κλάσεις που σχετίζονται με την εφαρμογή μιας επιλεγμένης αναδόμησης. Αναφορικά με την ανίχνευση των προτάσεων αναδόμησης οι σχετικές κλάσεις βρίσκονται στα πακέτα ***uom.java.ast, aueb.jdeodorant.ast.util*** ενώ στο πακέτο ***uom.java.ast.util*** έχουν δηλωθεί κλάσεις που μοντελοποιούν δηλώσεις(statements) και εκφράσεις (expressions) της java, καθώς και οι βοηθητικές κλάσεις που εξυπηρετούν τον γρήγορο εντοπισμό τους στο AST δέντρο. Από τα υπόλοιπα χρησιμοποιούμενα πακέτα το ***jdeodorant.preferences*** περιλαμβάνει κλάσεις που αφορούν την παραμετροποίηση του plug- in και στο πακέτο ***jdeodorant.decomposition.cfg*** περιλαμβάνονται οι κλάσεις για τη δημιουργία των γράφων CFG και PDG. Θα πρέπει να τονιστεί ότι η κατανομή των κλάσεων σε πακέτα με εξαίρεση το πακέτο ***aueb.jdeodorant.ast.util*** είχε ήδη διενεργηθεί για τους σκοπούς του (40).

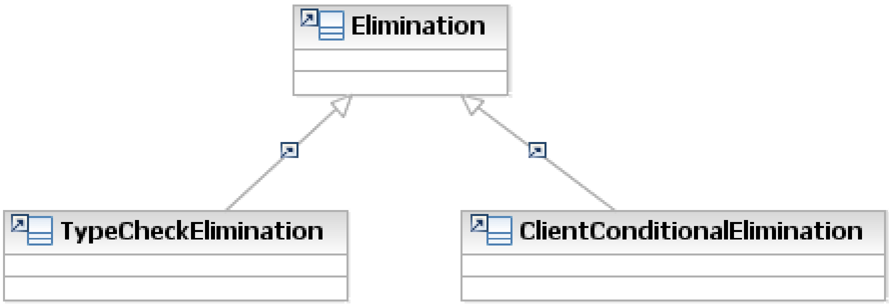
Ακολούθως στον Πίνακας 19 παρουσιάζονται επιγραμματικά οι κυριότερες κλάσεις του συστήματος που ανήκουν στα ανωτέρω πακέτα και χρησιμοποιούνται για τους σκοπούς της ανίχνευσης των προτάσεων αναδόμησης, οι περισσότερες από τις οποίες απεικονίζονται και στην Εικόνα 76 που παρουσιάζει το αντίστοιχο διάγραμμα κλάσεων. Προκειμένου να διαχωριστούν οι κλάσεις που προστέθηκαν από τις υπάρχουσες που απλά χρησιμοποιήθηκαν ή από αυτές που υπέστησαν μικρού βαθμού τροποποιήσεις έχουν χρησιμοποιηθεί διαφορετικά χρώματα που επεξηγούνται στο σχετικό υπόμνημα. Αντίστοιχα, η Εικόνα 77 απεικονίζει τις κλάσεις που χρησιμοποιούνται κατά την εφαρμογή των προτάσεων αναδόμησης. Συνοπτικές πληροφορίες σχετικά με τη λειτουργικότητα των κλάσεων αυτών και την κατανομή τους στα πακέτα σχεδίασης παρέχονται στον σχετικό πίνακα (Πίνακας 20). Τέλος, οι κλάσεις που απαρτίζουν το γραφικό περιβάλλον του υπό ανάπτυξη plug-in έχουν απομονωθεί από τις υπόλοιπες στο διάγραμμα κλάσεων της Εικόνα 78.

Αξίζει να σημειωθεί πώς τόσο στο διάγραμμα κλάσεων της Εικόνα 76 όσο και της Εικόνα 77 δεν έχουν συμπεριληφθεί όλες οι συμμετέχουσες κλάσεις του JDeodorant, ούτε του JDT,ούτε όλες οι σχέσεις εξάρτησης (dependency) μεταξύ των διαφόρων κλάσεων παρά μόνο οι βασικότερες για λόγους αναγνωσιμότητας του διαγράμματος.

Εν συντομία, σχετικά με το διάγραμμα κλάσεων της Εικόνα 76 για την ανίχνευση των προτεινόμενων αναδομήσεων αρχικά ο χρήστης επιλέγει το μενού Conditional Checking που δημιουργείται από την κλάση ***BadSmellsMenu*** και στη συνέχεια επιλέγοντας την ανίχνευση αλληλεπιδρά με την αντίστοιχη όψη που μοντελοποιείται από την κλάση ConditionalChecking. Η κλάση ***ConditionalChecking*** αρχικοποιεί ένα αντικείμενο της κλάσης ***ASTReader*** το οποίο αναλαμβάνει τη δημιουργία του AST δέντρου για το υπό εξέταση λογισμικό με τη βοήθεια της κλάσης ASTParser του JDT. Επιπλέον, αρχικοποιεί την κλάση ***SystemObject*** η οποία αναλαμβάνει τον περαιτέρω συντονισμό της διαδικασίας ανίχνευσης και την συλλογή των αποτελεσμάτων ούτως ώστε να τα επιστρέψει στη γραφική διεπαφή προς παρουσίαση. Η κλάση αυτή είναι η μόνη που μεσολαβεί μεταξύ των κλάσεων του γραφικού περιβάλλοντος (επιπέδου παρουσίασης) της εφαρμογής και των κλάσεων που εμπεριέχουν τη λογική ανίχνευσης (επίπεδο επιχειρησιακής λογικής) και αναλαμβάνει την μεταξύ τους επικοινωνία.

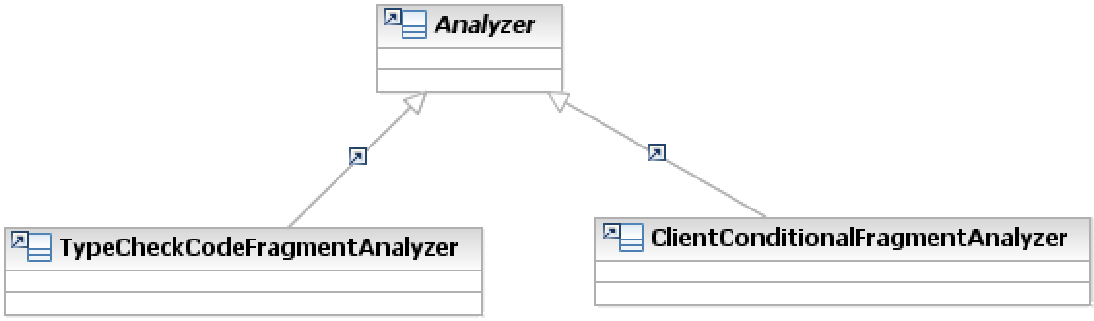
Οι προτάσεις αναδόμησης προς τη στρατηγική μοντελοποιήθηκαν στο σύστημα μέσω της κλάσης ***ClientCheckElimination***, στην οποία αποθηκεύονται όλες οι απαραίτητες πληροφορίες για τον εντοπισμό και τη διενέργεια της αναδόμησης, και η οποία κληρονομεί την υπερκλάση ***Elimination***, στην οποία μεταφέρθηκαν όλες οι κοινές μέθοδοι και πεδία μεταξύ των διαφορετικών προτάσεων αναδόμησης της στρατηγικής και της κατάστασης. Τα αντικείμενα αυτής της κλάσης δημιουργούνται κατά την ανίχνευση των προτεινόμενων αναδομήσεων σε επίπεδο μεθόδου (κλάση ***MethodBodyObject***) με βάση τις υποθετικές δηλώσεις που ανιχνεύονται. Ακολούθως σε επίπεδο κλάσης με τη βοήθεια των κλάσεων ***ClassObject*** και ***StrategyCriteriaConfiguration*** εκτελείται ο έλεγχος των κριτηρίων που ορίστηκαν στην ενότητα 5.2. Η συλλογή των απαιτούμενων πληροφοριών για τον έλεγχο των κριτηρίων πραγματοποιείται από την κλάση ***CompleteRemovalCandidatesDetector*** και την κλάση ***ClientConditionalFragmentAnalyzer***, που κληρονομεί την κλάση ***Analyzer***, στην οποία μεταφέρθηκαν κοινές μέθοδοι με την ανίχνευση προτάσεων αναδόμησης προς την κατάσταση ή με χρήση πολυμορφισμού. Το τελικό σύνολο των έγκυρων προτάσεων αναδόμησης τυγχάνει περαιτέρω επεξεργασίας στην κλάση ***ClassObject***, προκειμένου να αποδοθεί σε καθεμία από αυτές ένα ανεξάρτητο βάρος, βάσει το οποίου θα ταξινομηθούν για την παρουσίαση τους στο χρήστη. Οι προτάσεις αναδόμησης τελικά επιστρέφονται σε επίπεδο συστήματος δηλαδή στην κλάση ***SystemObject***. Οι υπόλοιπες κλάσεις του διαγράμματος είναι οι βασικότερες από τις υποβοηθητικές κλάσεις που συμμετέχουν στη διαδικασία ανίχνευσης.

Αναφορικά με το AST δέντρο θα πρέπει να αναφερθεί ότι η επανειλημμένη διάσχιση του για την εξέταση των κριτηρίων ανίχνευσης που ορίστηκαν από τη μεθοδολογία καθίσταται απαγορευτική τόσο για λόγους απόδοσης, όσο και εξαιτίας της δυναμικής φύσης του, καθώς τροποποιείται κάθε φορά που ο πηγαίος κώδικας υφίσταται αλλαγής. Η δυναμική αυτή φύση του AST δέντρου καθιστά αδύνατη τη μόνιμη αποθήκευση του στη μνήμη, η οποία αντενδείκνυται και λόγω της αυξημένης χωρητικότητας που θα απαιτούσε μια τέτοια ενέργεια. Ωστόσο, για τον εντοπισμό, την εμφάνιση των προτάσεων αναδόμησης και την εκτέλεση της σε επόμενο στάδιο απαιτείται η συλλογή και αποθήκευση πληροφοριών. Η διαδικασία αυτή επιτυγχάνεται μέσω της ενδιάμεσης αναπαράστασης των απαραίτητων στοιχείων της Java (συστήματος, κλάσεων, μεθόδων, κλπ.) που έχει κατασκευαστεί στα πλαίσια του ερευνητικού έργου (40) και ενός μηχανισμού που επιτρέπει την ανάκτηση των κόμβων του AST δέντρου χωρίς αυτό να αποθηκεύεται μόνιμα στη μνήμη *(*κλάση ***ASTInformation)***. Οι κλάσεις που αντιστοιχούν στην ενδιάμεση αυτή αναπαράσταση εμπεριέχονται στο πακέτο uom.ast και χρησιμοποιήθηκαν τόσο κατά την ανίχνευση όσο και κατά την εκτέλεση των προτεινόμενων αναδομήσεων.



Εικόνα 72 : Ιεραρχία κλάσεων που μοντελοποιούν τις προτάσεις αναδόμησης που δημιουργήθηκε

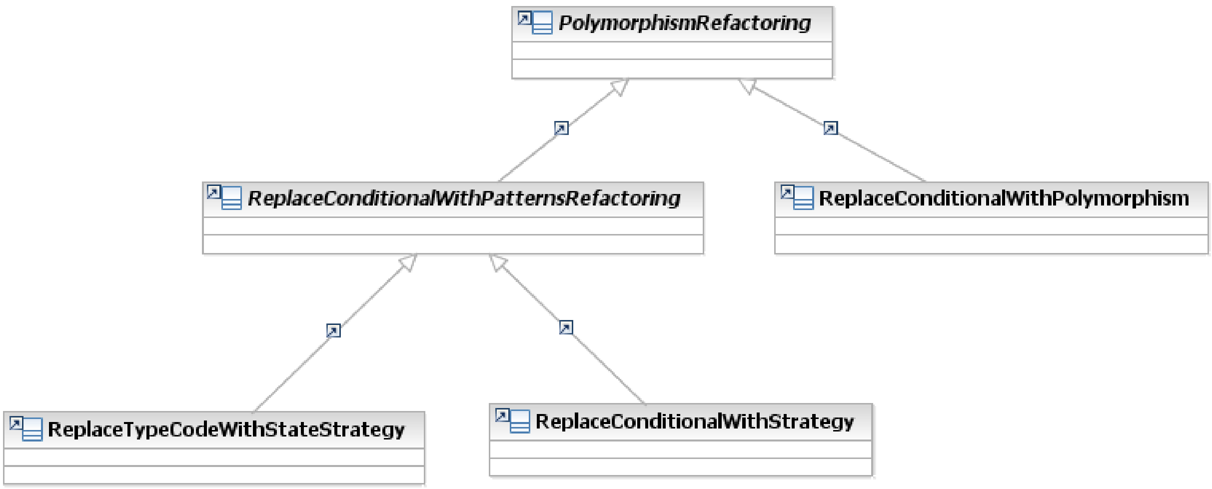
Στα πλαίσια της παραπάνω υλοποίησης, όπως αναφέρθηκε και παραπάνω, οι καινούργιες κλάσεις που δημιουργήθηκαν παρουσίασαν ορισμένα κομμάτια κοινής λειτουργικότητας, με κλάσεις που συμμετέχουν στην ανίχνευση αναδόμησης με την κατάσταση/στρατηγική. Προκειμένου να αποφευχθεί η επανάληψη κώδικα, η κοινή λειτουργικότητα μεταφέρθηκε με χρήση των απλών αναδομήσεων εξαγωγή μεθόδου (extract method), μετακίνηση μεταβλητής μέλους και μεθόδου σε υπερκλάση (pull up field, method), στην υπερκλάση που δημιουργήθηκε μέσω της απλής αναδόμησης εξαγωγή υπερκλάσης (extract superclass). Η ιεραρχία κλάσεων που προέκυψε από την εφαρμογή της αναδόμησης απεικονίζεται ακολούθως στις Εικόνα 72 και Εικόνα 73. Οι υποκλάσεις ***TypeCheckElimination*** και ***ClientConditionalElimination*** της κλάσης ***Elimination*** μοντελοποιούν τις προτάσεις αναδόμησης της υπάρχουσας προσέγγισης προς την κατάσταση/στρατηγική και της προτεινόμενης προσέγγισης προς τη στρατηγική αντίστοιχα, ενώ οι υποκλάσεις ***TypeCheckConditionalAnalyzer*** και ***ClientConditionalAnalyzer*** της υπερκλάσης ***Analyzer*** που δημιουργήθηκε αναλαμβάνουν να αναλύσουν περαιτέρω τις υποθετικές δηλώσεις και να συλλέξουν τις απαιτούμενες πληροφορίες για τη διεξαγωγή των ελέγχων που ορίζουν τα κριτήρια στις προτεινόμενες μεθοδολογίες.



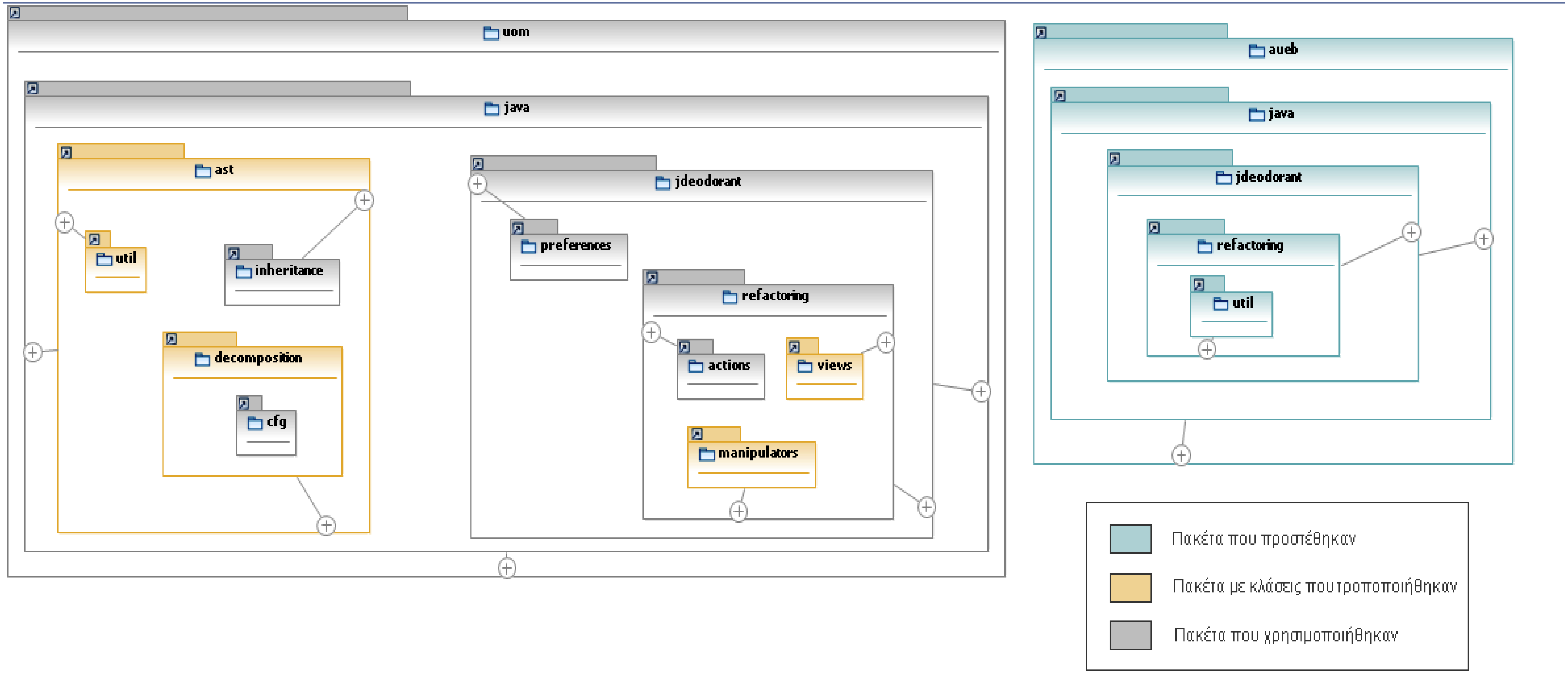
Εικόνα 73 : Ιεραρχία κλάσεων που αφορούν τον έλεγχο των κριτηρίων ανίχνευσης

Ως προς την εκτέλεση των μετασχηματισμών της αναδόμησης, οι μετασχηματισμοί αυτοί δημιουργούνται κατά τρόπο ώστε να μην τροποποιείται άμεσα το AST δέντρο. Η δυνατότητα αυτή απαιτείται προκειμένου να είναι εφικτή η προεπισκόπηση των προτεινόμενων αλλαγών και η μετέπειτα εφαρμογή τους στον κώδικα, μόνο κατόπιν της έγκρισης του χρήστη. Η λειτουργικότητα αυτή πραγματοποιήθηκε μέσω της δημιουργίας των κατάλληλων αντικειμένων που μοντελοποιούν τις αλλαγές του κώδικα (***CompilationUnitChange***), χωρίς την άμεση εφαρμογή τους στον κώδικα. Οι αλλαγές που σχετίζονται με την αναδόμηση προς τη στρατηγική ορίζονται στα πλαίσια της κλάσης ***ReplaceConditionalWithStrategy***, για την ομαλή ένταξη της οποίας στο σύστημα, διενεργήθηκε πάλι αναδόμηση του υπάρχοντα κώδικα με αποτέλεσμα την ιεραρχία κλάσεων που απεικονίζεται στην Εικόνα 74. Συγκεκριμένα, στην υπάρχουσα ιεραρχία προστέθηκε η ενδιάμεση κλάση ***ReplaceConditionalWithPatternsRefactoring*** στην οποία τοποθετήθηκαν όλες οι κοινές μέθοδοι και μεταβλητές μέλη μεταξύ της υπάρχουσας και της καινούργιας υλοποίησης.

Σύμφωνα με το διάγραμμα κλάσεων της Εικόνα 77, η διαδικασία εκτέλεσης μιας επιλεγμένης αναδόμησης (στιγμιότυπο της κλάσης ***ClientConditionalElimination***) εκκινείται πάλι από την κλάση ***ConditionalChecking*** η οποία περνάει τον έλεγχο της διαδικασίας σε ένα άλλο στοιχείο του γραφικού περιβάλλοντος που αρχικοποιεί, την κλάση ***MyRefactoringWizard,*** η οποία μοντελοποιεί τον βηματικό οδηγό για την προεπισκόπηση και αποδοχή του προτεινόμενου μετασχηματισμού. Η τελευταία αφενός δημιουργεί ένα στιγμιότυπο της κλάσης ***ReplaceConditionalWithStrategyInputPage*** μέσω της οποίας ο χρήστης μπορεί να αλλάξει τα προεπιλεγμένα ονόματα των κλάσεων της ιεραρχίας στρατηγικής και αφετέρου διαθέτει μια συσχέτιση με την υπερκλάση ***Refactoring.*** Στην προκειμένη περίπτωση ο τύπος του στιγμιότυπου Refactoring, ορίζεται από την κλάση ConditionalChecking σε ***ReplaceConditionalWithStrategy.*** Η κλάση αυτή αναλαμβάνει τη διενέργεια της πλειοψηφίας των μετασχηματισμών του AST δέντρου κληρονομώντας τις απαραίτητες γι' αυτό το σκοπό μεθόδους από την ιεραρχία των κλάσεων ***ReplaceConditionalWithPatternsRefactoring*** και ***PolymorphismRefactoring*** αλλά και ορίζοντας καινουργίες δικές της αποκλειστικά μεθόδους. Οι προτεινόμενες αλλαγές του AST δέντρου δημιουργούνται ως αντικείμενα της κλάσης ***CompilationUnitChange,*** ενώ στην περίπτωση ολικής αντικατάστασης υποθετικής λογικής απαιτείται αλληλεπίδραση της κλάσης ***ReplaceConditionalWithStrategy*** με την κλάση ***ConditionalExpressionEvaluator,*** για την ανίχνευση των αντικειμένων στρατηγικής που θα πρέπει να αντικαταστήσουν τα ορίσματα κάθε κλήσης της μεθόδου στις κλάσεις πελάτη (client). Τέλος, η κλάση ***ReplaceConditionalWithStrategy*** δημιουργεί επιπλέον και ένα στιγμιότυπο της κλάσης ***ReplaceConditionalLogicWithStrategyDescriptor*** το οποίο αναλαμβάνει να επαναλάβει τους συγκεκριμένους μετασχηματισμούς στην περίπτωση που τελικά ο χρήστης ακυρώσει την εφαρμογή των αλλαγών ή την αναιρέσει και επιλέξει αργότερα να την επαναλάβει.



Εικόνα 74 : Ιεραρχία κλάσεων που αφορούν την εκτέλεση αναδόμησης



Εικόνα 75 : Διάγραμμα πακέτων στα οποία περιλαμβάνονται οι κλάσεις που χρησιμοποιήθηκαν για την αναδόμηση προς τη στρατηγική

Πακέτο	Κλάση	Περιγραφή
ast	ASTReader	Δημιουργεί το AST δέντρο.
	ASTInformation	Συσχετίζει τις κλάσεις της ενδιάμεσης αναπαράστασης που παρέχονται στην υλοποίηση του JDeodorant με τις αντίστοιχες κλάσεις στο JDT Core, αποθηκεύοντας τις απαραίτητες πληροφορίες για την ανάκτηση ενός AST κόμβου.
	ClassObject	Μοντελοποιεί τις κλάσεις του συστήματος αποθηκεύοντας πληροφορίες όπως τα γνωρίσματα τους, τις μεθόδους, τους κατασκευαστές, κλπ. Επιπλέον αναλαμβάνει την ανίχνευση όλων των υποθετικών δηλώσεων με 2 διακλαδώσεις που βρίσκονται στις μεθόδους της και χρήζουν αναδόμησης. Περιλαμβάνει τον έλεγχο των κριτηρίων και την βαθμονόμηση των προτάσεων αναδόμησης.
	CompilationUnitCache	Αποθηκεύει προσωρινά τις κλάσεις που υφίστανται τροποποιήσεις προκειμένου, σε επόμενη εφαρμογή της ανίχνευσης να κατασκευαστεί το AST δέντρο μόνο για τις τροποποιημένες κλάσεις.
	SystemObject	Μοντελοποιεί το σύστημα αποθηκεύοντας πληροφορίες όπως οι κλάσεις του συστήματος, τα ονόματα τους, οι getter και οι setter μέθοδοι, κλπ. Αναλαμβάνει την επικοινωνία μεταξύ του view και controller της ανίχνευσης. Ελέγχει την ανίχνευση και συλλέγει τις προτάσεις αναδόμησης προκειμένου να τις επιστρέψει στο UI.
	ClientConditionalFragmentAnalyzer	Αναλύει τις υποθετικές εκφράσεις και τις υποθετικές διακλαδώσεις (Εικόνα 70) και συγκεντρώνει τις απαραίτητες πληροφορίες (παραμέτρους, γνωρίσματα, ύπαρξη δομής για την απόδοση βάρους, κλπ.) προκειμένου να εκτελεστεί ο έλεγχος όλων των κριτήριων ανίχνευσης προς τη στρατηγική και σε επόμενο στάδιο να εκτελεστεί η αναδόμηση.

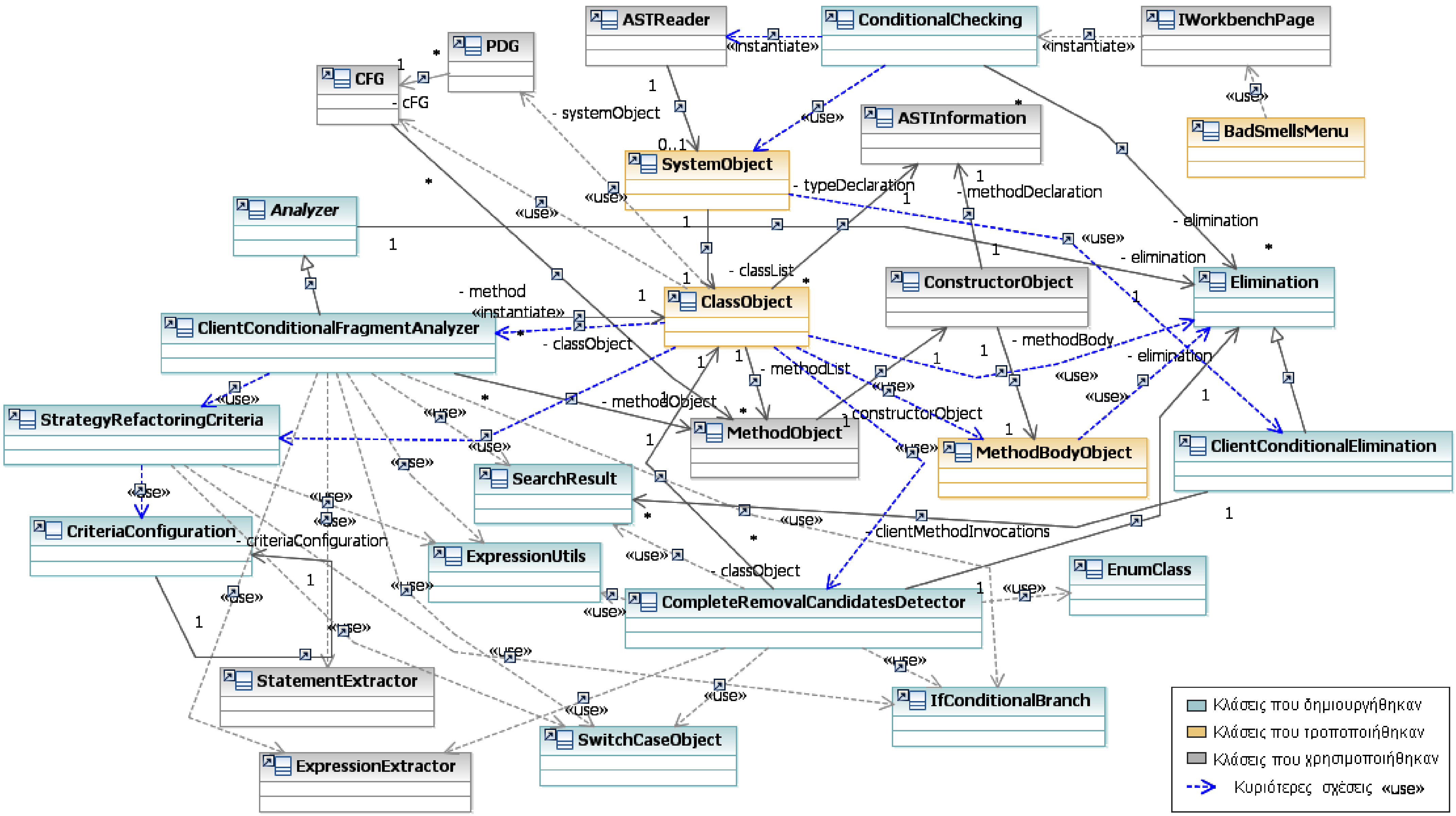
	Analyzer	Υπερκλάση των κλάσεων TypeCheckCodeFragmentAnalyzer και ClientConditionalFragmentAnalyzer που δημιουργήθηκε μέσω αναδόμησης για την παροχή κοινής λειτουργικότητας μεταξύ των δύο προσεγγίσεων. Αναλαμβάνει την επεξεργασία των διακλαδώσεων και την αποθήκευση κλήσεων μεθόδων και πεδίων της κλάσης context ή της υπερκλάσης της, παραμέτρων και τοπικών μεταβλητών, πληροφορίες οι οποίες θα πρέπει να τύχουν διαφορετικού χειρισμού κατά την εφαρμογή της αναδόμησης.
	SearchResult	Μοντελοποιεί τα αποτελέσματα αναζήτησης κλήσεων μεθόδων όπως περιγράφηκαν στην ενότητα 5.6.1
	CompleteRemovalCandidatesDetector	Αναλαμβάνει την ανίχνευση των υποψήφιων περιπτώσεων πλήρους αντικατάστασης υποθετικής λογικής εκτελώντας τους απαραίτητους ελέγχους και αποθηκεύοντας τις απαραίτητες πληροφορίες (ενότητες 0.0.5.3.2, 5.6.2).
Ast. decomposition	MethodBodyObject	Μοντελοποιεί το σώμα των μεθόδων. Στην κλάση αυτή απλώς ανιχνεύονται όλες οι υποθετικές δηλώσεις if/switch που περιλαμβάνονται στα πλαίσια κάθε μεθόδου.
	IfStatementExpressionAnalyzer	Αναλύει τις υποθετικές εκφράσεις μιας δήλωσης if και αποθηκεύει χρήσιμα δεδομένα όπως οι σταθερές που αντιστοιχούν σε κάθε μεταβλητή τύπο, οι αντίστοιχες υποκλάσεις, κλπ. Χρησιμοποιείται και για την ανάλυση σύνθετων υποθετικών εκφράσεων με χρήση των λογικών τελεστών AND και OR.
Ast.decomposition.cfg	CFG	Κατασκευάζει και μοντελοποιεί τον γράφο CFG για μια μέθοδο.
	PDG	Κατασκευάζει και μοντελοποιεί ένα γράφο PDG Βάσει του αντίστοιχου CFG γράφου.
Ast.	CompleteInheritanceDetection	Ανιχνεύει και αποθηκεύει τις ιεραρχίες κλάσεων που

inheritance		υπάρχουν στο σύστημα.
	InheritanceTree	Μοντελοποιεί μια ιεραρχική δενδρική δομή.
Ast.util	ExpressionExtractor	Παρέχει μεθόδους για τη διάσχιση του AST δέντρου και τη συλλογή όλων των εκφράσεων ενός συγκεκριμένου τύπου.
	StatementExtractor	Παρέχει μεθόδους για τη διάσχιση του AST δέντρου και τη συλλογή όλων των δηλώσεων ενός συγκεκριμένου τύπου.
	ExpressionUtils	Βοηθητική κλάση που δημιουργήθηκε με μεθόδους που υλοποιούν χρήσιμες λειτουργίες αναφορικά με τις εκφράσεις ενός προγράμματος (π.χ. έλεγχος αν είναι μεταβλητή, σύγκριση δύο διαφορετικών εκφράσεων, κλπ.) για την εύκολη επαναχρησιμοποίηση τους από διάφορα σημεία του κώδικα, με αποφυγή της επανάληψης κώδικα.
	IfConditionalBranch	Συνδυάζει μια υποθετική έκφραση και τη διακλάδωση της σε μια κλάση, πολλά διαφορετικά στιγμιότυπα της οποίας συνδυάζονται για να σχηματίσουν τη δήλωση if. Αυτή η μοντελοποίηση έχει ως στόχο την αποθήκευση μόνο των απαραίτητων πληροφοριών για την αναδόμηση προς τη στρατηγική και τη διευκόλυνση της εκτέλεσης των ελέγχων.
	SwitchCaseObject	Κατ' αντιστοιχία με την κλάση IfConditionalBranch, μοντελοποιεί τις περιπτώσεις case οι οποίες συνθέτουν μια δήλωση switch συνδυάζοντας την έκφραση case και τις εντολές που περιλαμβάνονται στην αντίστοιχη διακλάδωση σε ένα νέο ξεχωριστό αντικείμενο.
preferences	PreferenceConstants	Ορίζει τις σταθερές για την παραμετροποίηση του plug-in
Refactoring	Activator	Ελέγχει τον κύκλο ζωής του plug-in.
Refactoring. actions	BadSmellsMenu	Παρέχει το μενού με τις επιλογές των bad smells (Εικόνα 32).

Refactoring. Views	ConditionalChecking	Μοντελοποιεί το view "Conditional Checking" στο workbench του eclipse για την αναδόμηση προς τη στρατηγική. Παρέχει τα κουμπιά και τον χειρισμό της ανίχνευσης, εμφάνισης και εφαρμογής των προτεινόμενων αναδομήσεων (Εικόνα 32).
	NameSorter	Ταξινομεί τις προτάσεις αναδόμησης βάσει του βάρους τους σε πρώτο επίπεδο και βάσει του μέσου αριθμού δηλώσεων ανά διακλάδωση σε επόμενο επίπεδο
	ViewLabelProvider	Ορίζει τα ονόματα των στηλών του πίνακα στον οποίο παρουσιάζονται οι προτάσεις αναδόμησης
	ViewContentProvider	Παρέχει το περιεχόμενο δηλαδή τη λίστα με τις προτάσεις αναδόμησης στον πίνακα του γραφικού περιβάλλοντος
	ElementChangedListener	Ανιχνεύει τις αλλαγές που συμβαίνουν στο ενεργό project (προσθήκη, τροποποίηση, διαγραφή κλάσης) και ενημερώνει κατάλληλα την μνήμη cache. Σε επόμενη επιλογή της ανίχνευσης των προτεινόμενων αναδομήσεων το AST δέντρο δε δημιουργείται εξ' αρχής αλλά ενημερώνεται μόνο για τις τροποποιημένες κλάσεις.
Refactoring. Manipulators	ClientConditionalElimination	Μοντελοποιεί τις προτεινόμενες περιπτώσεις αναδόμησης προς τη στρατηγική .
	Elimination	Υπερκλάση των κλάσεων ClientConditionalElimination και TypeCheckElimination για τη παροχή των κοινών πεδίων και λειτουργιών των δύο τύπων προτάσεων αναδόμησης που ορίζονται από τις δύο προσεγγίσεις.
Aueb.jdeodorant.refactoring.util	CriteriaConfiguration	Αναλαμβάνει την ανάγνωση του αρχείου παραμετροποίησης και τον προσδιορισμό της κατάστασης των κριτηρίων ανίχνευσης (ενότητα 5.6.3).
	StrategyRefactoringCriteria	Κλάση που περιλαμβάνει μεθόδους που ελέγχουν και επιστρέφουν την κατάσταση ενός κριτηρίου και

	μεθόδους που ελέγχοντας την κατάσταση του κριτηρίου εκτελούν ή μη τους απαραίτητους ελέγχους των κριτηρίων και επιστρέφουν το αποτέλεσμα που προκύπτει από κάθε έλεγχο (ενότητα 5.6.3)
EnumClass	Μοντελοποιεί μια κλάση enumeration αποθηκεύοντας το πλήρες όνομα της καθώς και την αντιστοιχία μεταξύ ονομάτων των enum constants και των τιμών της προκειμένου να χρησιμοποιηθεί στην περίπτωση πλήρους απαλοιφής υποθετικής λογικής.

Πίνακας 19 : Κατανομή κλάσεων ανίχνευσης προτάσεων αναδόμησης σε πακέτα σχεδίασης

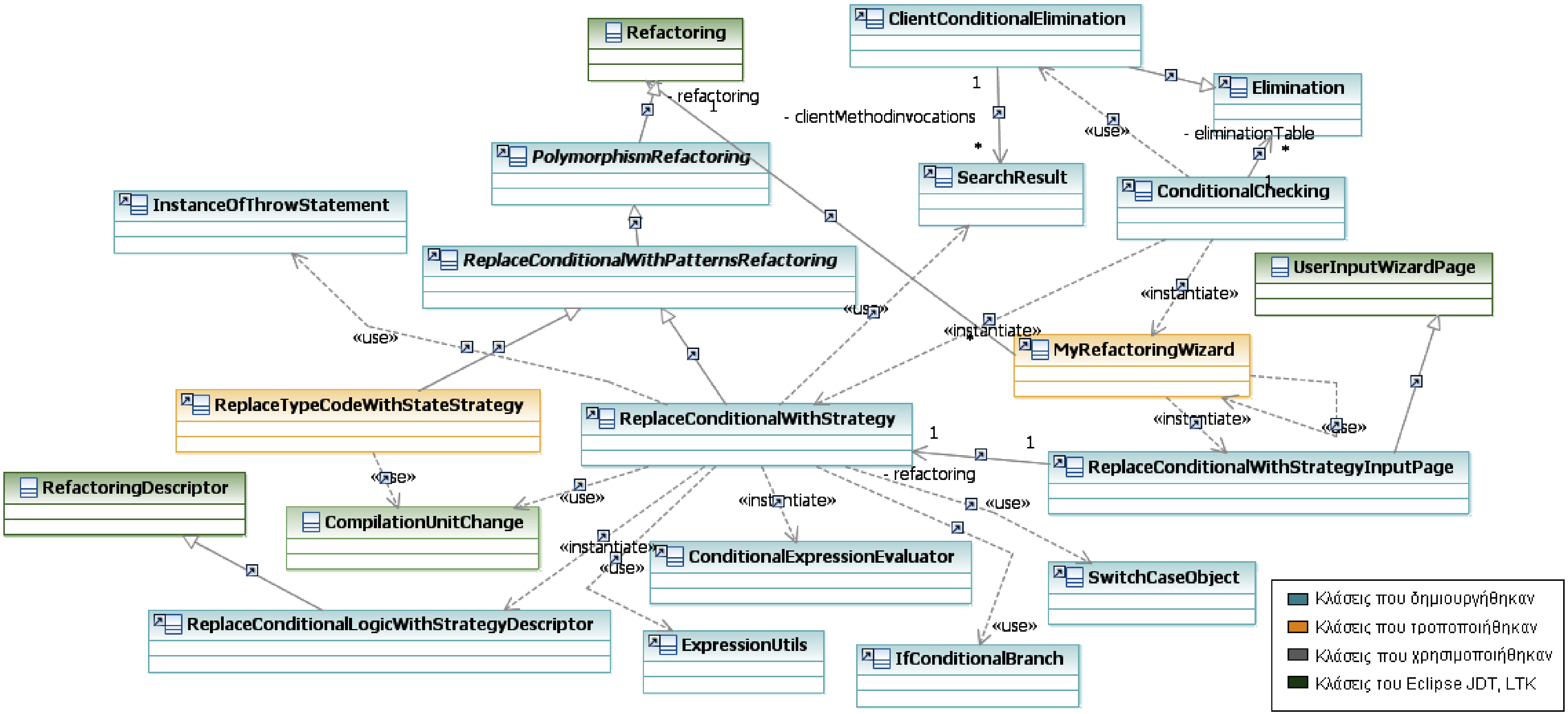


Εικόνα 76 : Οι βασικότερες κλάσεις που χρησιμοποιούνται στην ανίχνευση προτάσεων αναδόμησης προς τη στρατηγική

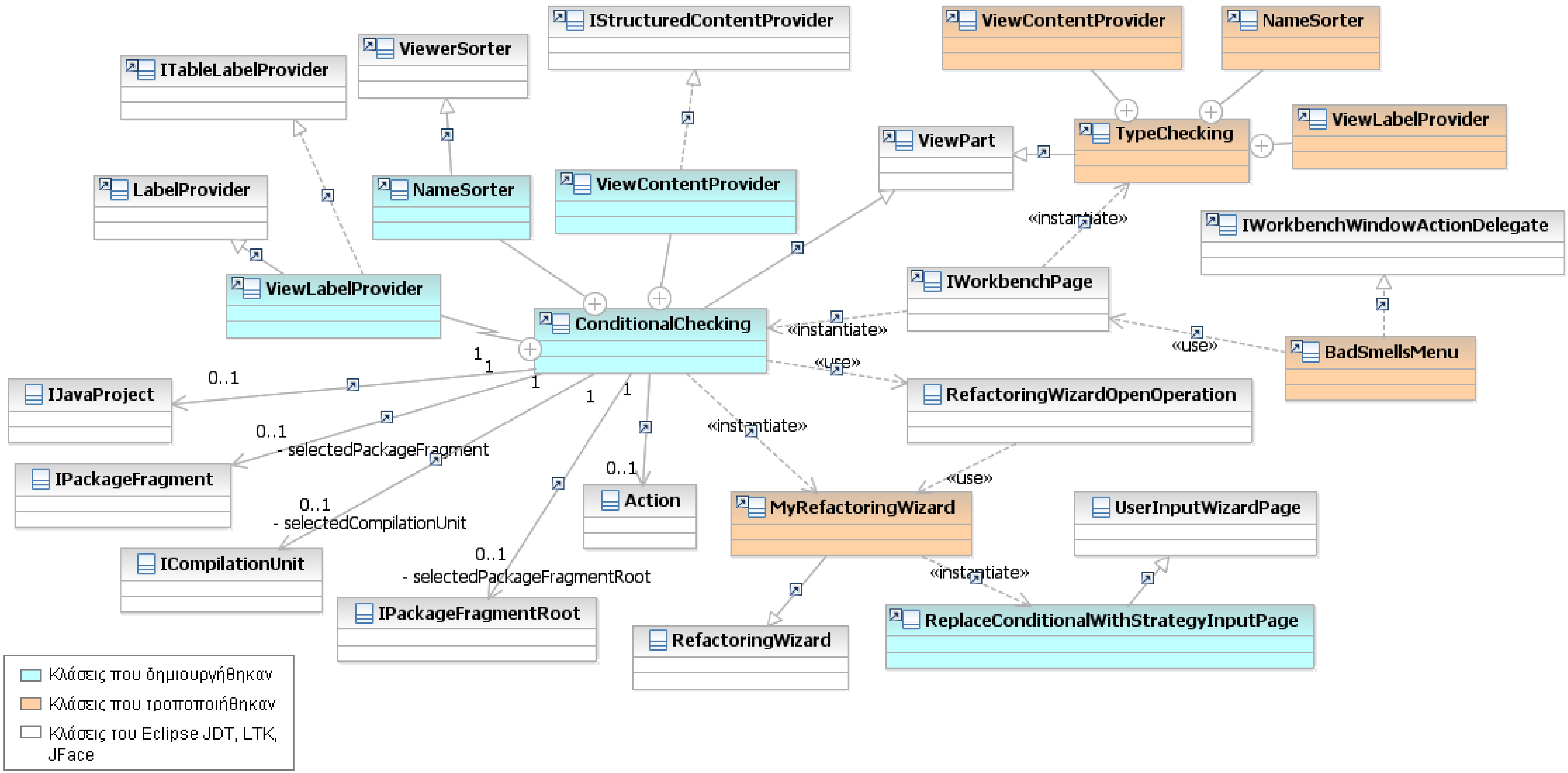
Πακέτο	Κλάση	Περιγραφή
Ast.util	InstanceOfThrowStatement	Μοντελοποιεί τις δηλώσεις throw προκειμένου να τύχουν κατάλληλου χειρισμού κατά την αναδόμηση (προσθήκη δήλωσης throws στις υπογραφές των νέων μεθόδων).
Refactoring. Views	ConditionalChecking	Μοντελοποιεί την όψη “Conditional Checking” στο workbench του eclipse για την αναδόμηση προς τη στρατηγική. Παρέχει τα κουμπιά και τον χειρισμό της ανίχνευσης, εμφάνισης και εφαρμογής των προτεινόμενων αναδομήσεων (Εικόνα 32).
	MyRefactoringWizard	Υλοποιεί τη γραφική διεπαφή για την προεπισκόπηση της αναδόμησης. Παρέχει τη δυνατότητα στο χρήστη να τροποποιήσει τις προεπιλεγμένες τιμές των παραμέτρων της αναδόμησης.
	ReplaceTypeCodeWithStrategy InputPage	Μοντελοποιεί την σελίδα του οδηγού (wizard) στην οποία εμφανίζονται τα προεπιλεγμένα ονόματα των κλάσεων που θα δημιουργηθούν και τις εκφράσεις που αφορούν στο χρήστη. Μέσω αυτής της σελίδας ο χρήστης μπορεί να αντικαταστήσει τα προεπιλεγμένα ονόματα των κλάσεων στρατηγικής με αυτά της επιλογής του. Επίσης παρέχει τη δυνατότητα προεπισκόπησης της αλλαγής (Εικόνα 34Εικόνα 35).
	ElementChangedListener	Ανιχνεύει τις αλλαγές που συμβαίνουν στο ενεργό project (προσθήκη, τροποποίηση, διαγραφή κλάσης) και ενημερώνει κατάλληλα την μνήμη cache. Σε επόμενη επιλογή της ανίχνευσης των προτεινόμενων αναδομήσεων θα ενημερωθεί το AST δέντρο μόνο για τις τροποποιημένες κλάσεις.
Refactoring. Manipulators	PolymorphismRefactoring	Υπερκλάση που παρέχει τις κοινές μεθόδους για τις αναδομήσεις βάσει πολυμορφισμού και βάσει προτύπου κατάστασης.
	ReplaceConditionalWithPatternsRefactoring	Υπερκλάση που δημιουργήθηκε για την παροχή των κοινών μεθόδων εκτέλεσης της αναδόμησης μεταξύ των αναδομήσεων με χρήση προτύπων των δύο

		μεθοδολογιών.
	ReplaceConditionalWithStrategy	Παρέχει τη λειτουργικότητα για την εφαρμογή της αναδόμησης προς τη στρατηγική. Εκτελεί τις αλλαγές που περιγράφηκαν στην ενότητα 5.5.
	ReplaceConditionalLogicWithStrategyDescriptor	Υπερφόρτωση της κλάσης RefactoringDescriptor του JDT για την αναδόμηση βάσει προτύπου στρατηγικής ούτως ώστε να είναι δυνατό να αναπαραχθεί η ίδια αναδόμηση ξανά (π.χ. μετά από χρήση των εντολών αναίρεση - επανεκτέλεση).
	ClientConditionalElimination	Μοντελοποιεί τις προτεινόμενες προτάσεις αναδόμησης προς τη στρατηγική
Aueb.jdeodorant.refactoring.util	ConditionalExpressionEvaluator	Αναλαμβάνει τον υπολογισμό των υποθετικών εκφράσεων που αληθεύουν στην περίπτωση αναδόμησης πλήρους αντικατάστασης υποθετικής λογικής για την ανίχνευση της κλάσης στρατηγικής που θα πρέπει να καθοριστεί στην αναδόμηση της κλήσης μεθόδου στις κλάσεις πελάτη (clients)βάσει της λογικής που περιγράφηκε στην ενότητα 5.6.2.

Πίνακας 20 : Κατανομή κλάσεων εκτέλεσης της αναδόμησης σε πακέτα σχεδίασης



Εικόνα 77 : Οι βασικότερες κλάσεις που συμμετέχουν στην εκτέλεση της αναδόμησης προς τη στρατηγική



Εικόνα 78 : Κλάσεις του UI του εργαλείου που υλοποιήθηκε που περιλαμβάνονται στο πακέτο jdeodorant.refactoring.views

Παράρτημα Ε: Οδηγίες Εγκατάστασης του Plug-in

Για την εγκατάσταση του plugin απαιτείται απλά η αντιγραφή του αρχείου gr.uom.java.jdeodorant_4.0.2.jar στον φάκελο $ECLIPSE_HOME/plugins του Eclipse, όπου $ECLIPSE_HOME ο φάκελος στον οποίο έχει εγκατασταθεί το Eclipse (παραδείγματος χάριν C:\Ecipse\plugins) και η επανεκκίνηση του Eclipse.

Πριν την επανεκκίνηση του Eclipse συστήνεται η τροποποίηση των ρυθμίσεις μνήμης της jvm είτε από το αρχείο eclipse.ini που επίσης βρίσκεται στο φάκελο $ECLIPSE_HOME

```
-vmargs
-Xms128m
-Xmx1024m
-XX:PermSize=128m
```

είτε απευθείας από την κονσόλα command line (cmd) των Windows με χρήση της εντολής

```
eclipse -vmargs -Xms128m -Xmx1024m -XX:PermSize=128m
```

Για την τροποποίηση των κριτηρίων ανίχνευσης των υποψήφιων προτάσεων αναδόμησης προς τη στρατηγική, θα πρέπει να αποσυμπιεστεί το gr.uom.java.jdeodorant_4.0.2.jar στον φάκελο $ECLIPSE_HOME/plugins/gr.uom.java.jdeodorant_4.0.2 και να τροποποιηθεί το αρχείο criteria.properties που βρίσκεται στον φάκελο προέλευσης (source folder) του jdeodorant. Οι προεπιλεγμένες τιμές των κριτηρίων έχουν οριστεί σε true. Για την απενεργοποίηση κάποιου από τα κριτήρια θα πρέπει να μεταβληθεί η τιμή του σε false. Αν η τιμή κάποιου κριτήριου δε βρεθεί στο αρχείο, τότε θεωρείται εξ' ορισμού ως αληθής. Μετά από κάθε τροποποίηση των τιμών των κριτηρίων το eclipse θα πρέπει να επανεκκινείται προκειμένου να ενεργοποιούνται οι αλλαγές.

reject.invalid.private.methods.criterion	Κριτήριο απόρριψης υποθετικών δηλώσεων ιδιωτικών μεθόδων με είσοδο εντός context
reject.static.methods.criterion	Κριτήριο απόρριψης υποθετικών δηλώσεων στο εσωτερικό στατικών μεθόδων
reject.overriden.methods.criterion	Κριτήριο απόρριψης υποθετικών δηλώσεων στο εσωτερικό υπερφορτωμένων μεθόδων
reject.instanceof.conditional	Κριτήριο απόρριψης υποθετικών δηλώσεων με χρήση του τελεστή instanceof
allow.in.conditional.not.modified.parameters	Κριτήριο ένταξης υποθετικών δηλώσεων με παραμέτρους που δεν τροποποιούνται εντός της μεθόδου στις προτάσεις αναδόμησης
allow.in.conditional.fields.not.set.inside.context	Κριτήριο ένταξης υποθετικών δηλώσεων με πεδίο που λαμβάνει τιμή εκτός context στις προτάσεις αναδόμησης
reject.conditionals.with.non.allowable.size	Κριτήριο απόρριψης υποθετικών δηλώσεων με μέσο αρ. δηλώσεων >= 2 ή 2 διακλαδώσεις εκ των οποίων η μια έχει μια εντολή

Πίνακας 21 : Αντιστοίχιση παραμέτρων criteria.properties με τα κριτήρια ανίχνευσης προτάσεων αναδόμησης

Παράρτημα ΣΤ: Παραδείγματα Προτάσεων Αναδόμησης Πειραματικής Αξιολόγησης

Στο παρόν παράρτημα αρχικά παρουσιάζονται ορισμένα παραδείγματα των περιπτώσεων true positive και true negative που προέκυψαν από την εκτέλεση του πειράματος που περιγράφεται αναλυτικά στην ενότητα 6.3.1. Ακολούθως, παρατίθενται ορισμένα ακόμα παραδείγματα προτάσεων αναδόμησης που προέκυψαν από τις μεθοδολογίες ανίχνευσης Α και Β όπως ορίζονται στην ενότητα 6.3.3 και παρέχονται για την πληρέστερη κατανόηση της χρησιμότητα της μεθοδολογίας Β ως συμπληρωματική στην ανίχνευση προτάσεων αναδόμησης οι οποίες δεν ανιχνεύονταν από την μεθοδολογία Α που παρουσιάζεται αναλυτικά στο (1), καθώς και την καταλληλότητα των κριτηρίων ανίχνευσης για την διαφοροποίηση των προτάσεων αναδόμησης προς τη στρατηγική από τις προτάσεις αναδόμησης προς την κατάσταση.

Παραδείγματα περιπτώσεων true positive και true negative

1. Έγκυρη πρότασης αναδόμησης με πεδίο στην υποθετική έκφραση που προκύπτει από την εκτέλεση της μεθοδολογίας στο σύστημα MOEM. Στο συγκεκριμένο απόσπασμα κώδικα επιλέγεται η στρατηγική σχεδιασμού των ακμών του γράφου σε ευθείες γραμμές ή σε καμπύλες. Παρά το μικρό της μέγεθος η πρόταση αυτή θεωρείται ότι έχει αξία γι’ αυτό και προκύπτει ως αποτέλεσμα της ανίχνευσης, καθώς σε επόμενο στάδιο περαιτέρω βελτίωση της ποιότητας του παραγόμενου σχεδίου είναι εφικτή με συνδυασμό της αναδόμησης προς τη στρατηγική και της αναδόμησης μετακίνηση μεθόδου (move method) στις υποκλάσεις της στρατηγικής που θα δημιουργηθούν.

```
if (m_nEdgeNo == 0)
        drawLines(g);
else
        drawCurves(g);
```

Εικόνα 79 : Περίπτωση true positive με πεδίο στην υποθετική έκφραση

2. Έγκυρη υποψήφια αναδόμηση προς τη στρατηγική του συστήματος JFScheduler όπου επιλέγεται ο τρόπος ανοίγματος ενός αρχείου για εγγραφή ή για ανάγνωση μόνο. Ο έλεγχος βασίζεται στην τιμή της παραμέτρου bWrite, η οποία δεν τροποποιείται στα πλαίσια της μεθόδου openFile γι’ αυτό και συμπεριλαμβάνεται στα αποτελέσματα της ανίχνευσης.

Επιπλέον η συγκεκριμένη περίπτωση είναι υποψήφια για ολική αντικατάσταση της δήλωσης if, όπως παρουσιάζεται στην ενότητα 5.6.5.6.5.2.

```
public boolean openFile(boolean bWrite) {
try {
        if (bWrite) {
                if (fileWriter != null)
                        fileWriter.close();
                        fileWriter = new BufferedWriter(new FileWriter(m_sFile));

        } else {
                if (fileReader != null)
                        fileReader.close();
                fileInputStream = new FileInputStream(m_sFile);
                fileReader = new BufferedReader(new InputStreamReader(
                fileInputStream));
        }
        return true;
} catch (FileNotFoundException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
}
return false;
}
```

Εικόνα 80 : Περίπτωση true positive με παράμετρο στην υποθετική έκφραση

3. Έγκυρη περίπτωση που εξαιρείται από τις προτάσεις αναδόμησης που προκύπτουν από την εκτέλεση της προτεινόμενης μεθοδολογίας καθώς οι διακλαδώσεις της υποθετικής λογικής περιλαμβάνουν κλήσεις σε μεθόδους δημιουργίας (creation method) και συνεπώς αποτελεί υποψήφια αναδόμησης με τη μέθοδο εργοστάσιο (factory method). Επιπλέον η συγκεκριμένη περίπτωση απορρίπτεται και λόγω του κριτηρίου των ιδιωτικών μεθόδων. Παρόμοια είναι και η περίπτωση της Εικόνα 82 που περιλαμβάνει τη δημιουργία αντικειμένων τα οποία τελικά επιστρέφονται από τη μέθοδο.

```
private TrafficPattern getTrafficPattern(Application type, int id) {
        TrafficPatternFactory tpf =TrafficPatternFactory.getInstance();
        TrafficPattern tp = null;
        String sId = type.toString() + "_" + id;
        switch (type) {
                case HTTP:
                        tp = tpf.createDefaultHTTPOnOffTrafficPattern(sId);
                        break;
                case FTP_UP:
                        tp = tpf.createDefaultFTPOnOffTrafficPattern(Flow.UPLINK, sId);
                        break;
                case FTP_DOWN:
                        tp = tpf.createDefaultFTPOnOffTrafficPattern(
                        Flow.DOWNLINK, sId);
                        break;
                case VOIP:
                        tp = tpf.createVoIPTrafficPattern(sId);
```

```
                    break;
            case VIDEOCONFERENCE:
                    tp = tpf.createVideoConfTrafficPattern(sId);
                    break;
            case EMAIL:
                    break;
            }
        return tp;
}
```

Εικόνα 81 : Περίπτωση true negative με μεθόδους δημιουργίας στις διακλαδώσεις της υποθετικής έκφρασης

```
public IScheduler getInstance(FlowSchedulingProblem fsp) {
        IScheduler instance = null;
        switch (type) {
            case HEURISTIC_EX:
                    instance = new HeuristicSchedulerEx(fsp, constrType);
                    break;
            case UTILITY:
                    instance = new UtilityScheduler(fsp);
                    break;
            case HEURISTIC:
                    instance = new HeuristicScheduler(fsp, constrType);
                    break;
            case LINGO:
                    instance = LingoScheduler.createBiObjScheduler(fsp);
                    break;
        }
        return instance;
}
```

Εικόνα 82 : Περίπτωση true negative με δημιουργία αντικειμένων στις διακλαδώσεις

4. Περίπτωση υποθετικής λογικής του συστήματος MOEM που απορρίπτεται από το σύνολο των υποψηφίων λόγω του περιορισμένου μεγέθους της.

```
if (fromNode.getType() == Node.MULTIDIM)
        return true;
else
        return false;
```

Εικόνα 83 : Περίπτωση true negative περιορισμένης έκτασης

5. Περίπτωση υποθετικής λογικής του συστήματος IceHockeyManager που εξαιρείται από το σύνολο των υποψηφίων καθώς ορίζεται στο εσωτερικό της μεθόδου compareTo που υπερφορτώνει τη γνωστή μέθοδο του api της java για τη σύγκριση μεταξύ δύο αντικειμένων και δεν αποτελεί έγκυρη πρόταση αναδόμησης.

```
@Override
 public int compareTo(Object o) {
   .....
  //Then check difference between goalsFor & goalsAgainst
   int otherDiff = other.getGoalsDiff();
   int ourDiff = getGoalsDiff();

   if (otherDiff != ourDiff) {
```

```
    if (otherDiff > ourDiff) {
      return smaller;
    } else {
      return bigger;
    }
  }
}
```

Εικόνα 84 : Περίπτωση true negative σε υπερφόρτωση μεθόδου

6. Περίπτωση υποθετικής λογικής που ορίζεται στα πλαίσια στατικής μεθόδου και ορθώς εξαιρείται από το σύνολο των προτάσεων αναδόμησης καθώς η εφαρμογή του προτύπου στρατηγικής στο παρόν απόσπασμα κώδικα δε θα είχε νόημα.

```
public static void main(String[] args) throws EntityManagerNotInitializedException {
        …
        if (app.hasParameter("small")) {
                exp = Experiment.getInstance(expId, simulationId, "small scale - "
                        + args[0] + " - " + duration, type, 1, 1, 1, 1, 0, 0, 5, 4);
        } else {
                exp = Experiment.getInstance(expId, simulationId, expDesc + " - "
                        + duration, type, 5, 5, 5, 5, 0, 0, 5, 4);
        }
        …
}
```

Εικόνα 85: Περίπτωση true negative σε στατική μέθοδο

Παραδείγματα κοινών και μη κοινών περιπτώσεων των μεθοδολογιών Α και Β

1. Απόσπασμα υποθετικής λογικής του συστήματος IceHockeyManager που ανιχνεύεται και από τις δύο μεθοδολογίες και επιλέγει τη στρατηγική σχεδίασης ενός στοιχείου του γραφικού περιβάλλοντος του συστήματος, ανάλογα με τον τύπο του. Στην περίπτωση αυτή ενδείκνυται το πρότυπο στρατηγική έναντι του προτύπου κατάσταση καθώς, ο τύπος του στοιχείου του γραφικού περιβάλλοντος καθορίζεται μια φορά κατά τη δημιουργία του και εκκίνηση του προγράμματος και έπειτα δε μεταβάλλεται. Συνεπώς εφόσον δεν ανιχνεύονται εκχωρήσεις στο πεδίο block ανιχνεύεται και από τη μεθοδολογία Β, αλλά και από τη μεθοδολογία Α που ελέγχει μόνο την ύπαρξη τελεστή ισότητας και τη σύγκριση με σταθερά.

```
if (block.getBlockType() == Block.BLOCK_NORMAL || block.getBlockType() ==
Block.BLOCK_POWERPLAY) {
    this.lblCenter.setEnabled(true);
    this.lblCenter.setVisible(true);
    this.lblCenter.setText(PlayerTools.getShortName(block.getCenter()));
    this.lblCenter.setTransferHandler(new LineUpTransferHelper());
        this.lblCenter.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        block.setCenter(getSelectedFieldPlayer());
        lblCenter.setText(PlayerTools.getShortName(block.getCenter()));
```

```
        }
      });
} else {
      this.lblCenter.setEnabled(false);
      this.lblCenter.setVisible(false);
      this.lblCenter.setTransferHandler(null);
}
```

Εικόνα 86 : Κοινή πρόταση αναδόμησης των δύο μεθοδολογιών στο IceHockeyManager

2. Απόσπασμα υποθετικής λογικής του συστήματος MOEM που καθορίζει τον τρόπο με τον οποίο σχεδιάζεται ένας κόμβος ανάλογα με τον τύπο του. Η περίπτωση δεν ανιχνεύεται στη μεθοδολογία Α λόγω του τελεστή ανισότητας «!=» της δεύτερης υποθετικής έκφρασης, ωστόσο ανιχνεύεται στην μεθοδολογία Β που δε θέτει τέτοιου είδους περιορισμό.

```
if (m_nType == MULTIDIM)
{
        g.setColor( m_bPicked ? ( m_bHanging ?
        selectColorHung : selectMColor ) : ( m_bHanging ? hangingColor : cMNodeColor ) );
        g.fillRect((int)x - w/2,(int) y - h / 2, w, h);
        g.setColor(Color.black);
        if (m_bRoot)
                g.drawRect((int)x - w/2 + 2,(int) y - h / 2 + 2, w-5, h-5);
        g.drawRect((int)x - w/2,(int) y - h / 2, w-1, h-1);
        g.drawString( m_sLabel, (int)x - (w-12)/2, ((int)y - (h-4)/2)
                + fm.getAscent());
}
else if (m_nType != NON_VISIBLE)
{
        g.setColor( m_bPicked ? ( m_bHanging ? selectColorHung
                : selectCColor ) :( m_bHanging ? hangingColor : cCNodeColor) );
        g.fill(new Ellipse2D.Double(x - (double)w/2,
                y - (double)w / 2, w, w));
        g.setColor(Color.black);
        if (m_bRoot)
                g.draw(new Ellipse2D.Double(x -(double) w/2 + 3, y -(double)w / 2 + 3,
                        w - 6 ,w - 6));
        g.draw(new Ellipse2D.Double(x -(double) w/2, y -(double)w / 2, w, w));
        g.drawString(m_sLabel, (int)x- (w - 12) / 2,(int)y + w/8);
        // Display atomic value directly under the node.
        if (m_nType == ATOMIC)
                g.drawString(m_sValue, (int)x - 6*(m_sValue.length() / 2), (int)y+w);
}
else
{
        g.setColor(IEdge.arcColor3);
        g.fill(new Ellipse2D.Double(x - w/2, y - w/2, w, w));
        g.setColor(Color.black);
}
```

Εικόνα 87 : Περίπτωση που ανιχνεύεται μόνο στη μεθοδολογία Β λόγω χρήσης του τελεστή ανισότητας

3. Απόσπασμα υποθετικής λογικής του συστήματος IceHockeyManager που δεν ανιχνεύεται από τη μεθοδολογία Α καθώς η υποθετική έκφραση δεν περιλαμβάνει σύγκριση με σταθερά, αλλά τον έλεγχο ενός πεδίου boolean.Το παράδειγμα αυτό αποτελεί έγκυρη περίπτωση

αναδόμησης προς τη στρατηγική, καθώς εκτελείται διαφορετικός αλγόριθμος στην κανονική εκτέλεση του συστήματος και διαφορετικός αν το σύστημα εκτελείται για αποσφαλμάτωση, και ανιχνεύεται από τη μεθοδολογία Β.

```
if (!debug) {
    // Scenario
        LeagueOwner[] leagueOwners =
        GameController.getInstance().getScenario().getLeagueOwners();

    for (int i = 0; i < leagueOwners.length; i++) {
      League[] leagues = leagueOwners[i].getLeagues();
      for (int n = 0; n < leagues.length; n++) {
        eventOwners.add(leagues[n]);
        cbEventOwners.addItem(leagues[n].getName());
      }
    }
} else {
        eventOwners.add(GameController.getInstance().getScenario().getScheduler());
        cbEventOwners.addItem(ClientController.getInstance().getTranslation("calendar.debugEvents"));
}
```

Εικόνα 88: Πρόταση αναδόμησης μεθοδολογίας Β με έλεγχο πεδίου Boolean χωρίς εκχωρήσεις στην κλάση context

4. Αντίστοιχως της προηγούμενης, στην ακόλουθη περίπτωση επιλέγεται η στρατηγική σχεδιασμού ενός στοιχείου του γραφικού περιβάλλοντος του συστήματος IceHockeyManager, με περιθώριο (border) ή χωρίς, η οποία ανιχνεύεται μόνο από την μεθοδολογία Β λόγω του ελέγχου βάσει μεταβλητής boolean. Στην περίπτωση αυτή παράλληλα καθίσταται εφικτή η ολική αντικατάσταση της υποθετικής λογικής κατά την αναδόμηση.

```
if (border) {
    this.content = panel;
    panelContentCenter.setLayout(panelContentCenterLayout);
    this.panelContentCenter.add(content, BorderLayout.CENTER);
} else {
    JlhmPanel host = new JlhmPanel();
    host.setLayout(flowLayout);
    host.add(panel);
    this.panelContentCenter.add(host);
    this.content = host;
}
```

Εικόνα 89 : Πρόταση αναδόμησης μεθοδολογίας Β με εφικτή την ολική αντικατάσταση της δήλωσης if

Συντομογραφίες

API : Application Programming Interface

AST : Abstract Syntax Tree

CFG : Control Flow Graph

JDT : Java Development Tools

LTK : Language ToolKit

OO : Object Oriented

PDG: Program Dependency Graph

RTTI : RunTime Type Identification

SLOC : Source Lines of Code

Βιβλιογραφία

1. N. Tsantalis, A. Chatzigeorgiou, Identification of Refactoring Opportunities Introducing Polymorphism, Journal of Systems and Software, vol. 83, no. 3, pp. 391-404, 2010.

2. B. P. Lientz, E. B. Swanson, Modernizing Legacy Systems Technologies Engineering, Addison-Wesley, 1980.

3. L. Tokuda and D. S Batory, Automated software evolution via design pattern transformations, In Proceedings of the International Symposium Applied, 1995.

4. T.M Pigoski, Practical Software Maintenance: Best Practices for Managing Your Software Investment (1st edition), Wiley, 1996.

5. R. C. Seacord, D. Plakosh, G. A. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices,* Addison-Wesley Professional, 2003.

6. Tom Mens, Serge Demeyer, *Software Evolution,* Springer, 2008.

7. W. F.Opdyke, Refactoring Object-Oriented Frameworks., Ph.D. Dissertation, University of Illinois at Urbana-Champaign, 1992.

8. M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the Design of Existing Code, Addison Wesley, Boston, MA, 1999.

9. Kent Beck, Extreme Programming Explained : Embrace Change, Addison Wesley, 1999.

10. William C Wake, Refactoring Workbook, Addison Wesley, 2003

11. S. Demeyer, S. Ducasse, O. Nierstrasz, Object-Oriented Reengineering Patterns, Square Bracket Associates, 2008.

12. E. J. Chikofsky, J. H. Cross, Reverse engineering and design recovery: A taxonomy,Reverse engineering and design recovery: A taxonomy, IEEE Software, vol. 7, no. 1, pp. 13–17, 1990.

13. R. Fanta, V. Rajlich, Reengineering object-oriented code, In Proceedings of the International Conference in Software Maintenance, IEEE Computer Society, pp. 238–246,Washington DC, USA, 1998.

14. R. Fanta, V. Rajlich, Restructuring legacy C code into C++, In Proceedings of the International Conference in Software Maintenance, IEEE Computer Society, pp. 77–85,Washington DC, USA , 1999.

15. T. Mens, T. Tourwé, A Survey of Software Refactoring, IEEE Transac-tions on Software Engineering, vol. 30, no. 2, pp. 126-139, 2004.

16. J.Kerievsky, Refactoring to Patterns, Addison Wesley, 2004.

17. E. Murphy-Hill, A. P. Black, Refactoring tools: Fitness for purpose, IEEE Software, vol. 25, no. 5, pp. 38–44, 2008.

18. Don Roberts, John Brant, and Ralph Johnson, A Refactoring Tool for Smalltalk, Theory and Practice of Object Systems, vol. 3, no. 4,pp. 253–263, 1997.

19. Don. Roberts, Practical Analysis for Refactoring, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1999.

20. Stephan.Erb, A survey of Software Refactoring Tools, Research Paper in Applied Computer Science, University of Cooperative State at Baden-Württemberg, Karlsruhe, 2010.

21. Eclipse. [Online] http://www.eclipse.org/.

22. Netbeans. [Online] http://netbeans.org/.

23. IntelliJ. [Online] http://www.jetbrains.com/idea/.

24. JDeodorant. [Online] http://www.jdeodorant.com/.

25. JRefactory. [Online] http://jrefactory.sourceforge.net/.

26. XRefactory. [Online] http://www.xref.sk/about.html.

27. RefactorIt. [Online] http://sourceforge.net/projects/refactorit/.

28. ReSharper. [Online] http://www.jetbrains.com/resharper/.

29. Refactor! Pro. [Online] http://www.devexpress.com/Products/Visual_Studio_Add-in/Refactoring/.

30. JustCode. [Online] http://www.telerik.com/products/justcode.aspx.

31. Visual Assist. [Online] http://www.wholetomato.com/products/featureRefactoring.asp.

32. Refactor! for C++. [Online] http://devexpress.com/Products/Visual_Studio_Add-in/RefactorCPP/.

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

34. Εμμ. Α. Γιακουμάκης, Ν. Διαμαντίδης, Τεχνολογία Λογισμικού, Εκδόσεις Α. Σταμούλης, Αθήνα , 2009.

35. E. Freeman, Er. Freeman, B. Bates, K. Sierra,Head First Design Patterns, O'Reilly Media, 2004.

36. B. Christiansson (Ed.), M. Forss, I. Hagen, K. Hansson, J. Jonasson, M. Jonasson, F. Lott, S. Olsson, T. Rosevall, GoF Design Patterns - with examples using Java and UML2, Creative Commons, 2008.

37. J. Coplien, Multi-Paradigm for C++,Addison Wesley, 1999.

38. Strategy Pattern. [Online] [Cited:] http://c2.com/cgi/wiki?StrategyPattern.

39. State vs. Strategy. [Online] [Cited:] http://www.mail-archive.com/gang-of-4-patterns@cs.uiuc.edu/msg00223.html.

40. Nikolaos Tsantalis, Evaluation and Improvement of Software Architecture: Identification of Design Problems in Object-Oriented Systems and Resolution through Refactorings, PhD Thesis, University of Macedonia,Thesaloniki, 2010.

41. L. Tahvildari and K Kontogiannis, A metric-based approach to enhance design quality through meta-pattern transformations, In Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 183–192, IEEE Computer Society, 2003.

42. M. Salehie, S. Li, and L. Tahvildari, A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws, In Proceedings of the 14th IEEE International Conference on Program Comprehension, pp. 159-168, 2006.

43. Radu Marinescu, Detection strategies: Metrics-based rules for detecting design flaws, In Proceedings of the 20th International Conference on Software Maintenance,pp. 350–359, IEEE Computer Society Press, 2004.

44. A. Trifu, and R. Marinescu, Diagnosing Design Problems in Object Oriented Systems, In Proceedings of the 12th Working Conference on Reverse Engineering, pp.155-164, 2005.

45. T. Muraki and M. Saeki, Metrics for Applying GOF Design Patterns in Refactoring, In Proceedings of the International WorkShop on Principles of Software Evolution, 2001.

46. A. Trifu and U. Reupke,Towards Automated Restructuring of Object Oriented Systems,In Proceedings of the 11th European Conference on Software Maintenance and Reengineering, pp.39-48, 2007.

47. 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, 2010.

48. N. Tsantalis and A. Chatzigeorgiou, Identification of Move Method Refactoring Opportunities, IEEE Transactions on Software Engineering, vol. 35, no. 3, pp. 347–367, 2009.

49. T. Tourwé and T.Mens, Identifying refactoring opportunities using logic meta programming.,In Proceedings of the 7th International Conference on Software Maintenance and Re-engineering, IEEE Computer Society, pp. 91–100, 2003.

50. J. Rajesh and D. Janakiram, JIAD: A Tool to Inter Design Patterns in Refactoring, In Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, ACM Press, pp. 227−237 , 2004.

51. S. Ducasse, M. Rieger, and S. Demeyer, A language independent approach for detecting duplicated code, In Proceedings of the. International Conference on Software Maintenance , IEEE Computer Society, pp. 109–118, 1999.

52. Y. Kataoka, M. D. Ernst, W. G. Griswold, and D. Notkin, Automated Support for Program Refactoring using Invariants, In Proceedings of the 17th IEEE International Conference on Software Maintenance, IEEE Computer Society, pp. 736-743, 2001.

53. N. Tsantalis, and A. Chatzigeorgiou, Identification of Extract Method Refactoring Opportunities, In Proceedings of the 13th European Conference on Software Maintenance and Reengineering Kaiserslautern, Germany, pp. 119-128, 2009.

54. M. O'Keeffe, and M. Ó Cinnéide, Search-based software maintenance, In Proceedings of the 10th European Conference on Software Maintenance and Reengineering, pp. 249-260, 2006.

55. M. O'Keeffe, and M. Ó Cinnéide, Search-based software maintenance, The Journal of Systems and Software, vol. 81, no. 4, pp. 502-516, 2008.

56. M. O'Keeffe, and M. Ó Cinnéide, Getting the Most from Search-Based Refactoring, In Proceedings of the 9th Annual Genetic and Evolutionary Computation Conference, pp. 1114-1120, 2007.

57. O. Seng, J. Stammel, and D. Burkhart, Search-based determination of refactorings for improving the class structure of object-oriented systems,In Proceedings of 8th annual conference on Genetic and evolutionary computation, pp. 1909–1916, ACM Press, New York, USA, 2006.

58. M.Harman and L.Tratt, Pareto Optimal Search Based Refactoring at the Design Level, In Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation, pp. 1106–1113, 2007.

59. A. Ananda Rao, and K. Narendar Reddy, Detecting Bad Smells in Object Oriented Design Using Design Change Propagation Probability Matrix, In Proceeding of the International MultiConference of Engineers and Computer Scientists 2008, vol. 1, pp. 1001-1007, 2008.

60. F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui,A Bayesian Approach for the Detection of Code and Design Smells, In Proceedings of the 9th International Conference on Quality Software (QSIC'09), pp. 305-314, 2009.

61. R. Oliveto, F. Khomh, G. Antoniol, and Yann-Gaël Guéhéneuc, Numerical Signatures of Antipatterns: An Approach based on B-Splines, In Proceedings of the 14th Conference on Software Maintenance and Reengineering, IEEE Computer Society Press, 2010.

62. F. Simon, F. Steinbrückner, and C. Lewerentz, Metrics based refactoring, In Proceedings of the European Conference in Software Maintenance and Reengineering, IEEE Computer Society, pp. 30–38, 2001.

63. E. van Emden and L.Moonen, Java quality assurance by detecting code smells, In Proceedings of the 9th Working Conference on Reverse Engineering, IEEE Computer Society, pp. 97-108, 2002.

64. C. Parnin, C. Görg, and O. Nnadi,A Catalogue of Lightweight Visualizations to Support Code Smell Inspection, In Proceedings of the 4th ACM symposium on Software Visualization, pp. 77-86, 2008.

65. Emerson Murphy-Hill, Andrew P. Black, An interactive ambient visualization for code smells,In Proceedings of the 5th International Symposium on Software Visualization, 2010.

66. M. Ó Cinnéide, P. Nixon, A Methodology for the Automated Introduction of Design Patterns, In Proceedings of the IEEE International Conference on Software Maintenance, 1999.

67. M. Ó Cinnéide, Automated refactoring to introduce design patterns, In Proceedings of the 2000 International Conference on Software Engineering, pp. 722-724, 2000.

68. Sang-Uk Jeon, Joon-Sang Lee, and Doo-Hwan Bae, An automated refactoring approach to design pattern-based program transformations in Java programs, In Proceedings of the 9th Asia-Pacific Software Engineering Conference, 2002.

69. Călin Jebelean, Automatic detection of missing abstract-factory design pattern in object-oriented code, In Proceedings of the International Conference on Technical Informatics, Politehnica University in Timişoara, 2004.

70. Călin Jebelean, A Study on Detecting Refactoring Opportunities to Introduce the Abstract Factory Design Pattern in Object-Oriented Code, 2005.

71. M. Ó Cinnéide, Automated Application of Design Patterns: A Refactoring Approach, Ph.D. dissertation,University of Dublin, Trinity College, 2000.

72. Nicolas Juillerat, Models and Algorithms for Refactoring Statements, PhD. Dissertation. University of Fribourg,Switzerland, 2009.

73. Nicolas Juillerat, Béat Hirsbrunner, Toward an Implementation of the "Form Template Method, In Proceedings of the 7th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM 2007),Paris, France, 2007.

74. Nicolas Juillerat, Béat Hirsbrunner, Detecting and Removing Clones in an Automated Way, In Proceedings of the 3rd Workshop on Software Evolution through Transformations: Embracing the Change (SeTra 2006), pp. 63 - 74, Rio Grande do Norte, Brazil , 2006.

75. N. Tsantalis, T. Chaikalis, A. Chatzigeorgiou, JDeodorant: identification and removal of type-checking bad smells, In Proceedings of the 12th European Conference on Software Maintenance and Reengineering, pp. 329-331, 2008.

76. Călin Jebelean, Ciprian-Bogdan Chirilă, Vladimir Creţu**,** A logic based approach to locate composite refactoring opportunities in object-oriented code, In Proceedings of the IEEE International Conference on Automation Quality and Testing Robotics, Politehnica University in Timişoara, pp. 1-6, 2010.

77. T.Ball, The concept of dynamic analysis, In Proceedings of the 7th European Software Engineering Conference and the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, vol. 24, no. 26, pp. 216–234. Springer Verlag, 1999.

78. K.Kalleberg, Abstractions for Language-Independent Program Transformations, PhD thesis, University of Bergen, 2007.

79. ASTViewer. [Online] http://www.eclipse.org/jdt/ui/astview/index.php.

80. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman**,** Compilers: Principles, Techniques, and Tools (2nd Edition), Addison-Wesley, 2006.

81. J. Ferrante, K. Ottenstein, and J. Warren**,** The Program Dependence Graph and Its Use in Optimization, ACM Transactions on Programming Languages and Systems, vol. 9, no. 3, pp. 319–349, 1987.

82. Mark Weiser, Program Slicing, In Proceedings of the 5th International Conference on Software Engineering, 1981.

83. Mark Weiser, Program Slicing,. IEEE Transactions on Software Engineering, vol. SE-10, no. 4, pp. 352-357, 1984

84. B. Korel, and J. Laski**,** Dynamic program slicing, Information Processing Letters, vol. 29, no. 3, pp. 155-163, 1988.

85. N. Tsantalis, A. Chatzigeorgiou**,** Identification of Extract Method Refactoring Opportunities, In Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), pp. 119-128, Kaiserslautern, Germany , 2009.

86. K.Maruyama, Automated Method-Extraction Refactoring by Using Block-Based Slicing, In Proceedings of the Symposium on Software Reusability, pp.31-40, 2001.

87. Alecsandar Stoianov, Ioana Sora**,** Detecting Patterns and Antipatterns in Software using Prolog Rules, In Proceedings of the 2010 International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI), pp.253 - 258, Politehnica University of Timisoara, Romania , 2010 .

88. Thomas J. McCabe**,** A Complexity Measure,IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308-320, 1976.

89. Giuseppe Jacopini and Corrado Böhm**,** Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules, Magazine Communications of the ACM, vol. 9, no. 5, pp. 366-371, 1966.

90. S. Counsell, Y. Hassoun, R. Johnson, K. Mannock, E. Mendes**,** Trends in Java code changes: the key to identification of refactorings? In Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java, Computer Science Press, Inc, New York, NY, USA, pp.45-48, 2003.

91. Amina Mukhtar, Fakhar Lodhi**,** Conditional Logic Detection for Identifying the opportunity of Design Pattern Application, In Proceedings of the Graduate Colloquium on Computer Sciences (GCCS), National University of Computer and Emerging Sciences NUCES-FAST, Lahore, Pakistan, 2010.

92. Eclipse Java Development Tools . [Online] http://www.vogella.de/articles/EclipseJDT/article.html#jdt_ast.

93. Eclipse Java development tools (JDT) Overview. [Online] http://www.eclipse.org/jdt/overview.php#JDT_APT .

94. Eclipse JDT - Abstract Syntax Tree (AST) and the Java Model - Tutorial. [Online] http://www.vogella.de/articles/EclipseJDT/article.html.

95. The Language Toolkit: An API for Automated Refactorings in Eclipse-based IDEs. [Online] http://www.eclipse.org/articles/Article-LTK/ltk.html.

96. Abstract Syntax Tree. [Online] http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html.

97. Unleashing the Power of Refactoring. [Online] http://www.eclipse.org/articles/article.php?file=Article-Unleashing-the-Power-of-Refactoring/index.html.

98. Javaassist. [Online] http://www.csg.is.titech.ac.jp/~chiba/javassist/.

99. Using Java Reflection. [Online] http://java.sun.com/developer/technicalArticles/ALT/Reflection/.

100. Java Tutorial : The reflection API. [Online] http://download.oracle.com/javase/tutorial/reflect/index.html.

101. V.E. Zafeiris, E.A.Giakoumakis, *Optimized traffic flow assignment in multi-homed, multi-radio mobile hosts,* Computer Networks, In Press, Corrected Proof, 2010.

102. Metrics 1.3.6. [Online] http://metrics.sourceforge.net/ .

103. Ice Hockey Manager. [Online] http://sourceforge.net/projects/ihm/.

104. Vassilis Zafeiris, Christos Doulkeridis, Yannis Stavrakas and Manolis Gergatsoulis**,** An infrastructure for manipulating Multidimensional Semistructured Data, In Proceedings of the Hellenic Data Management Symposium (HDMS'02), 2002.

105. Yannis Stavrakas, Manolis Gergatsoulis, Christos Doulkeridis and Vassilis Zafeiris, Representing and querying histories of semistructured databases using multidimensional OEM, In Elsevier Information Systems, vol.24, no. 6, pp. 461-482, 2004.

106. Nutch. [Online] http://nutch.apache.org/.

107. Violet . [Online] http://www.horstmann.com/violet/.

108. Violet Editor. [Online] http://sourceforge.net/projects/violet/.

109. Pamvotis WLAN Simulator. *Pamvotis - IEEE 802.11 WLAN Simulator.* [Online] http://pamvotis.org/.

110. Design Pattern Detection Tool. [Online] http://java.uom.gr/~nikos/pattern-detection.html.

111. N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, S. T. Halkidis, Design Pattern Detection Using Similarity Scoring, IEEE Transactions on Software Engineering, vol. 32, no. 11, pp. 896-909, 2006.

112. C.W. Cleverdon, On the inverse relationship of recall and precision, Journal of Documentation, vol. 28. no 3, pp. 195-201, 1972.

113. M. Buckland, F. Grey, The relationship between Recall and Precision, Journal of the American Society for Information Science, vol. 45,no. 1, pp. 12-19, 1994.

114. Dynamic dispatch - Definition. [Online] http://www.wordiq.com/definition/Dynamic_dispatch.

115. Name Binding. [Online] http://en.wikipedia.org/wiki/Name_binding#cite_note-tkac08-0.

116. Using early binding and late binding in Automation. [Online] http://support.microsoft.com/kb/245115.

117. IBM Rational Software Architect. [Online] http://www.ibm.com/developerworks/rational/products/rsa/.

118. Heinz W Schmidt and S. Milton, Dynamic Dispatch in Object-Oriented Languages, Technical Report TR-CS-94-02, The Australian National University, Canberra, 1994.

119. Lazy Initialization. [Online] http://www.javapractices.com/topic/TopicAction.do?Id=34.

120. Lazy Load. [Online] http://martinfowler.com/eaaCatalog/lazyLoad.html.