REFACTORING TRIP ADVISOR: ΥΠΟΒΟΗΘΗΣΗ ΣΥΓΚΡΟΤΗΜΕΝΗΣ ΑΝΑΚΑΤΑΣΚΕΥΗΣ ΚΩΔΙΚΑ

Η

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Μηχανικών Η/Υ και Πληροφορικής
Εξεταστική Επιτροπή

από τον

Θεοφάνη Βαρτζιώτη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Απρίλιος 2016

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τους γονείς μου για την υποστήριξη τους κατά την διάρκεια των προπτυχιακών αλλά και των μεταπτυχιακών σπουδών μου όλα αυτά τα χρόνια. Επίσης θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου, Αποστόλη Ζάρρα, για όλη την βοήθεια και καθοδήγηση που παρείχε στα πλαίσια αυτής της διατριβής. Τέλος θα ήθελα να ευχαριστήσω τους Χρήστο Γιαννάκο, Στέφανο Συμεωνίδη και Βιργινία Τσίντζου για τις συμβουλές και τις υποδείξεις τους στη φάση της αξιολόγησης.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Θεοφάνης Βαρτζιώτης του Λεωνίδα και της Ελένης. MSc, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Απρίλιος, 2016. Refactoring Trip Advisor: Υποβοήθηση Συγκροτημένης Ανακατασκευής Κώδικα. Επιβλέποντας: Απόστολος Ζάρρας.

Η εργασία αυτή εντάσσεται στα πλαίσια της τεχνολογίας λογισμικού και συγκεκριμένα στην αξιολόγηση της ποιότητας αυτού. Η αξιολόγηση του λογισμικού είναι αρκετά αναγκαία καθώς αυτή συμβάλει σημαντικά στην ποιοτική του βελτίωση. Η εργασία επικεντρώνεται σε κάποια συγκεκριμένη κατηγορία βελτιώσεων του λογισμικού. Συγκεκριμένα, στις βελτιώσεις που μεταβάλλουν την εσωτερική δομή του πηγαίου κώδικα χωρίς όμως να επηρεάζουν την λειτουργικότητα του, γνωστό και ως ανακατασκευή κώδικα. Περιγράφονται οι 68 τεχνικές ανακατασκευής που περιέχονται στον κατάλογο του Martin Fowler και ομαδοποιούνται σε 6 κατηγορίες. Η πληθώρα αυτή των τεχνικών σε συνδυασμό με το γεγονός ότι οι προγραμματιστές αποφεύγουν την χρήση των αυτοματοποιημένων εργαλείων οδηγεί στην μη αποτελεσματική διαδικασία ανακατασκευής κώδικα. Στηριζόμενοι σε αυτό, προτείνουμε μια νέα μέθοδο συγκροτημένης ανακατασκευής κώδικα η οποία βασίζεται στον Χάρτη Ανακατασκευών. Ο Χάρτης Ανακατασκευών είναι ένα γράφημα στο οποίο κάθε κόμβος αποτελεί μια από τις 68 τεχνικές ανακατασκευής και κάθε ακμή ένα είδος συσχέτισης μεταξύ αυτών. Τα τρία είδη συσχέτισης που προτείνουμε, προέρχονται από την μελέτη του καταλόγου του Fowler και είναι τα εξής: Σχέση Διαδοχής, Σχέση “Μέρος από” και Σχέση “Αντί για”. Η μελέτη του Χάρτη έδειξε πως οι τεχνικές Extract Method και Move Method αποτελούν τις βασικότερες στην διαδικασία ανακατασκευής ενώ οι τεχνικές Extract Superclass και Extract Subclass είναι οι πιο πολύπλοκές. Για την οπτικοποιήση του Χάρτη Ανακατασκευών και για την αξιοποίηση του από τον προγραμματιστή, αναπτύσσεται μια επέκταση του Eclipse με το όνομα Refactoring Trip Advisor. Το εργαλείο αυτό επίσης παρέχει χρήσιμες πληροφορίες για κάθε τεχνική ανακατασκευής και για

μερικές από αυτές προσφέρει την δυνατότητα του αυτόματου εντοπισμού ευκαιριών ανακατασκευής στον κώδικα του χρήστη. Για την αξιολόγηση του Refactoring Trip Advisor δώσαμε τον κώδικα δύο μεθόδων σε έναν αριθμό από προγραμματιστές και τους ζητήσαμε να ανακατασκευάσουν τον έναν χειροκίνητα και τον άλλον με την βοήθεια του εργαλείου. Τα αποτελέσματα έδειξαν ότι η πλειοψηφία των χρηστών, τουλάχιστον διπλασίασε το πλήθος των ανακατασκευών που εφάρμοσε χρησιμοποιώντας τον Refactoring Trip Advisor. Επίσης παρατηρήθηκε ότι ο χρόνος εκτέλεσης της διαδικασίας ανακατασκευής με τον Refactoring Trip Advisor, για μερικούς χρήστες μειώθηκε και για κάποιους άλλους αυξήθηκε. Τέλος οι χρήστες ανέφεραν γενικά θετικά σχόλια από την εμπειρία τους με τη χρήση του εργαλείου.

EXTENDED ABSTRACT IN ENGLISH

Theofanis Vartziotis, MSc, Computer Science and Engineering Department, University of Ioannina, Greece. April, 2016. Refactoring Trip Advisor: An Aid to Structured Code Refactoring. Thesis Supervisor: Apostolos Zarras.

This work's general theme is part of software engineering and more specifically of the evaluation of its quality. Software evaluation is quite necessary as it contributes significantly to its quality improvement. The work focuses on a particular category of software improvements. Specifically, the improvements which modify the internal structure of the source code without affecting its functionality, better known as code refactoring. We describe the 68 refactoring techniques contained in Martin Fowler's catalog of refactorings and we group them into 6 distinct categories: Method Composition, Method Call Improvement, Conditional Expression Simplification, Generalization Improvement, Data Organization and Feature Movement Between Objects. The large number of those techniques in combination with the fact that programmers avoid the use of automated tools leads to an inefficient code refactoring process. In regards to this, we propose a new method of structured code refactoring based on the Map of refactorings. The Map of refactorings is a graph in which each node is represented by one of the 68 refactoring techniques and each edge represents a relationship between them. The three types of relationships that we propose derive from the study of Fowler's catalog of refactorings and are as follows: Succession Relationship, "Part of" Relationship and "Instead of" Relationship. The study of the Map of refactorings showed that the Extract Method and Move Method techniques are the most basic ones in the refactoring process while the Extract Superclass and Extract Subclass are the most complex. For the visualization of the Map of Refactorings and so as to make it available to the developer, we developed an Eclipse plug-in named Refactoring Trip Advisor. In addition this tool provides useful information for every refactoring technique as well as the feature of automatic identification of refactoring opportunities for some of them. For the evaluation of Refactoring Trip Advisor we

provided the source code of two methods in a number of developers and asked them to refactor one of them manually and the other one with the help of our tool. The results showed that the majority of users, at least doubled the number of the refactorings they implemented while using Refactoring Trip Advisor. We also observed that the refactoring process' duration while using Refactoring Trip Advisor, increased for some users and decreased for some others. Finally, the users generally reported positive feedback from their experience using our tool.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

1.1 Ορισμός Ποιότητας Λογισμικού

Το βασικό θέμα που διαπραγματεύεται η παρούσα εργασία ανήκει στο γενικότερο πλαίσιο της ποιότητας λογισμικού. Στα πλαίσια της τεχνολογίας λογισμικού, η ποιότητα αναφέρεται σε δύο ξεχωριστές αλλά σχετικές έννοιες που υπάρχουν, όπου η ποιότητα ορίζεται στο πλαίσιο της επιχείρησης:

- Λειτουργική ποιότητα λογισμικού: Αντικατοπτρίζει το πόσο καλά το λογισμικό προσαρμόζεται σε ένα συγκεκριμένο σχέδιο, με βάση τις λειτουργικές απαιτήσεις ή προδιαγραφές. Αυτό το χαρακτηριστικό υποδεικνύει επίσης το πώς συγκρίνεται το λογισμικό με τους ανταγωνιστές του στην αγορά ως ένα αξιόλογο προϊόν.
- Μη λειτουργική ποιότητα λογισμικού: Αναφέρεται στο κατά πόσο το λογισμικό πληροί τις μη λειτουργικές απαιτήσεις που υποστηρίζουν την υλοποίηση των λειτουργικών απαιτήσεων όπως η αξιοπιστία και η συντηρησιμότητα.

Σε μια ενδιαφέρουσα εργασία [1] στην οποία παρουσιάζονται διάφορες όψεις της ποιότητας, ο David Garvin ερευνά πως μπορεί η ποιότητα να γίνει αντιληπτή από διάφορους χώρους όπως η φιλοσοφία, η οικονομία, το marketing και η διοίκηση επιχειρήσεων. Έτσι κατέληξε πως η ποιότητα είναι μια πολύπλευρη έννοια που

μπορεί να περιγραφεί από πέντε διαφορετικές σκοπιές. Στηριγμένοι πάνω στα αποτελέσματα αυτής της έρευνας, οι B. Kitchenham και S. L. Pfleeger στο [2] παρουσιάζουν τις πέντε αυτές σκοπιές με γνώμονα την ποιότητα λογισμικού:

- Υπερβατική άποψη: Σύμφωνα με αυτή την άποψη η ποιότητα είναι κάτι που αναγνωρίζουμε, αλλά δεν μπορούμε να ορίσουμε. Δηλαδή όπως ακριβώς κάθε πραγματικό τραπέζι είναι μια προσέγγιση του ιδανικού τραπεζιού, μπορούμε να φανταστούμε την ποιότητα λογισμικού ως ένα ιδανικό το οποίο θα θέλαμε πάρα πολύ να προσεγγίσουμε αλλά ποτέ δεν θα είμαστε σε θέση να το υλοποιήσουμε πλήρως.
- Άποψη του χρήστη: Σε αυτή την άποψη υποστηρίζεται ότι η ποιότητα είναι η προσαρμογή σε ένα σκοπό. Σε αντίθεση με την υπερβατική άποψη, η άποψη του χρήστη είναι πιο χειροπιαστή και μπορεί να χρησιμοποιηθεί για την μέτρηση των χαρακτηριστικών ενός προϊόντος, όπως η αξιοπιστία και η συχνότητα ελαττωμάτων, με στόχο να κατανοήσουμε την συνολική ποιότητα του προϊόντος.
- Κατασκευαστική άποψη: Πρόκειται για την οπτική γωνία που υποστηρίζει ότι η ποιότητα είναι η συμμόρφωση στις προδιαγραφές. Εξετάζει την ποιότητα κατά την διάρκεια της παραγωγής και μετά την παράδοση του λογισμικού σε συνδυασμό με τις απαιτήσεις που έχουν καθοριστεί. Εξετάζεται αν το προϊόν κατασκευάστηκε σωστά από την αρχή, ώστε να περιοριστεί το κόστος με αποφυγή ελαττωμάτων μετά την ολοκλήρωση της παραγωγής.
- Άποψη του προϊόντος: Σύμφωνα με την άποψη του προϊόντος η ποιότητα είναι συνδεδεμένη με έμφυτα χαρακτηριστικά του προϊόντος. Αυτή η άποψη εκφράζει συνήθως τους περισσότερους κατασκευαστές στην αξιολόγηση του λογισμικού που πιστεύουν ότι αν ένα λογισμικό έχει ενδείξεις καλής εσωτερικής ποιότητας, κατά πάσα πιθανότητα θα έχει και καλές εξωτερικές ενδείξεις, όπως η αξιοπιστία και η ευκολία στην χρήση.
- Άποψη με βάση την αξία: Η ποιότητα εξαρτάται από το ποσό των χρημάτων που είναι διατεθειμένος να καταβάλει για αυτή ο πελάτης. Η άποψη της αξίας συνδυάζει την άποψη του προϊόντος και την κατασκευαστική άποψη. Το αδύναμο σημείο αυτής της άποψης είναι η δυσκολία καθορισμού ξεκάθαρων ορίων καθώς και η εφαρμογή της στην πράξη. Εάν καταφέρουμε να

εξισώσουμε την ποιότητα λογισμικού, με το ποσό που είναι διατεθειμένος να δώσει ο πελάτης για την αγορά του, έχουμε μια πολύ σημαντική σχέση που συνδέει το κόστος με την ποιότητα και αν παρουσιαστεί κάποιο πρόβλημα μπορούμε να την αξιοποιήσουμε.

1.2 Αναγκαιότητα Αξιολόγησης Ποιότητας Λογισμικού

Στην προηγούμενη ενότητα έγινε μια προσπάθεια περιγραφής της έννοιας της ποιότητας λογισμικού. Για να μπορέσει λοιπόν να μιλήσει κανείς για την ποιότητα ενός λογισμικού θα πρέπει πρώτα να υπάρχει κάποιος τρόπος να την μετρήσει ώστε να καταλήξει σε κάποια συμπεράσματα για αυτή. Πριν όμως παρουσιάσουμε μερικούς τρόπους με τους οποίους μπορεί να επιτευχθεί αυτό αξίζει να σημειωθεί για ποιο λόγο είναι σημαντική η αξιολόγηση του λογισμικού. Υπάρχουν δύο βασικά κίνητρα που ωθούν τους κατασκευαστές λογισμικού στην συνεχή αξιολόγηση του:

- Διαχείριση κινδύνων: Τα σφάλματα στο λογισμικό μπορούν να εγκυμονούν μεγάλους κινδύνους. Σφάλματα λογισμικού έχουν φτάσει στο σημείο να προκαλούν ανθρώπινες απώλειες, τα αίτια των οποίων κυμαίνονται από κακοσχεδιασμένες διεπαφές χρήστη, μέχρι και άμεσα προγραμματιστικά λάθη. Αυτό οδήγησε σε συγκεκριμένες απαιτήσεις για την ανάπτυξη εξειδικευμένων τύπων λογισμικού, ιδιαίτερα για λογισμικό ενσωματωμένο σε ιατρικές και άλλες συσκευές που ρυθμίζουν κρίσιμες υποδομές.
- Διαχείριση κόστους: Όπως και σε κάθε άλλο τομέα της μηχανικής, μια εφαρμογή με ποιοτικά δομημένο λογισμικό είναι περισσότερο ευανάγνωστη και έχει μικρότερο κόστος συντήρησης. Διάφορες έρευνες [16] καταδεικνύουν ότι φτωχά δομημένο λογισμικό, κυρίως σε επιχειρησιακές εφαρμογές, κοστίζει χρήμα και χρόνο λόγο της επιπλέον εργασίας πάνω σε αυτό (έως 33% του χρόνου ανάπτυξης σε ορισμένους οργανισμούς ανάπτυξης λογισμικού). Επιπλέον, η κακή ποιότητα της δομής του λογισμικού συσχετίζεται ισχυρά με τα προβλήματα των επιχειρήσεων λόγω αλλοιωμένων στοιχείων, παραβιάσεις της ασφάλειας, και μειωμένης απόδοσης.

1.3 Στόχος της Διατριβής

Με βάση την προηγούμενη ενότητα μπορεί να συμπεράνει κανείς πως είναι αναγκαία η άμεση και συνεχής αξιολόγηση αλλά και βελτίωση της ποιότητας του λογισμικού από τους κατασκευαστές του. Χρησιμοποιώντας καθιερωμένα πρότυπα και υπολογίζοντας γνώστες μετρικές, οι σχεδιαστές αποκτούν μια καλύτερη άποψη της ποιότητας του λογισμικού που αναπτύσσουν. Εάν κρίνουν ότι το λογισμικό τους είναι χαμηλής ποιότητας, οι σχεδιαστές παρεμβαίνουν στον σχεδιασμό του και εφαρμόζουν τις κατάλληλες αλλαγές προς τη βελτίωση αυτού.

Η παρούσα διατριβή εστιάζεται στην ανακατασκευή κώδικα, η οποία είναι μια εξειδικευμένη κατηγορία αυτών των βελτιώσεων. Συγκεκριμένα, η ανακατασκευή κώδικα αφορά τις βελτιώσεις που μεταβάλλουν την εσωτερική δομή του πηγαίου κώδικα χωρίς όμως να επηρεάζουν την λειτουργικότητα του. Υπάρχει μια πληθώρα τεχνικών ανακατασκευής καθώς και μια σειρά από εργαλεία που αυτοματοποιούν την εφαρμογή τους διευκολύνοντας έτσι τον προγραμματιστή. Παρόλα αυτά, στην πράξη οι προγραμματιστές αποφεύγουν την χρήση αυτών των εργαλείων και προτιμούν να κάνουν όλη τη διαδικασία χειροκίνητα [18,19]. Αυτό έχει ως αποτέλεσμα η διαδικασία ανακατασκευής που εφαρμόζουν στον κώδικα τους να μην είναι η βέλτιστη. Αιτίες αυτού του προβλήματος είναι η πιθανή έλλειψη γνώσης από τον προγραμματιστή του πλήρους εύρους των διαθέσιμων τεχνικών ανακατασκευής ή η αδυναμία εντοπισμού σημείων στον κώδικα που χρήζουν την εφαρμογή κάποιας συγκεκριμένης τεχνικής ανακατασκευής.

Ο βασικός στόχος αυτής της διατριβής είναι η παρουσίαση μιας νέας μεθόδου συγκροτημένης ανακατασκευής κώδικα. Σκοπός της μεθόδου που προτείνεται είναι να δώσει στον προγραμματιστή μια πιο ολοκληρωμένη άποψη των διαφόρων τεχνικών ανακατασκευής κώδικα βασισμένη στις συσχετίσεις μεταξύ αυτών. Οι συσχετίσεις αυτές προκύπτουν από τη συστηματική μελέτη των διαθέσιμων τεχνικών ανακατασκευής και δημιουργούν έναν γράφημα που ονομάζεται Χάρτης Ανακατασκευών τον οποίο ο προγραμματιστής συμβουλεύεται κατά την διαδικασία ανακατασκευής. Αξιοποιώντας αυτή τη νέα γνώση, ο προγραμματιστής συμβουλεύεται τον Χάρτη Ανακατασκευών και εφαρμόζει μια σειρά από διαδοχικές

τεχνικές ανακατασκευής στον κώδικα του. Επίσης στα πλαίσια αυτής της διατριβής παρουσιάζεται και αναπτύσσεται μια επέκταση του Eclipse για την οπτικοποίηση του Χάρτη Ανακατασκευών μέσω κατάλληλων γράφων. Η ονομασία του εργαλείου, Refactoring Trip Advisor, προκύπτει από το γεγονός ότι ο χρήστης συμβουλευόμενος τον Χάρτη, πλοηγείται προς τις προτεινόμενες διαδρομές μεταξύ των τεχνικών ανακατασκευής. Ο Refactoring Trip Advisor είναι εφοδιασμένος επίσης με την δυνατότητα αυτόματου εντοπισμού ευκαιριών εφαρμογής μερικών τεχνικών ανακατασκευής. Η αρχική ιδέα του Χάρτη Ανακατασκευών εμφανίστηκε στο [17] και επεκτείνεται στην παρούσα διατριβή με την υλοποίηση της σε ένα ρεαλιστικό προγραμματιστικό περιβάλλον.

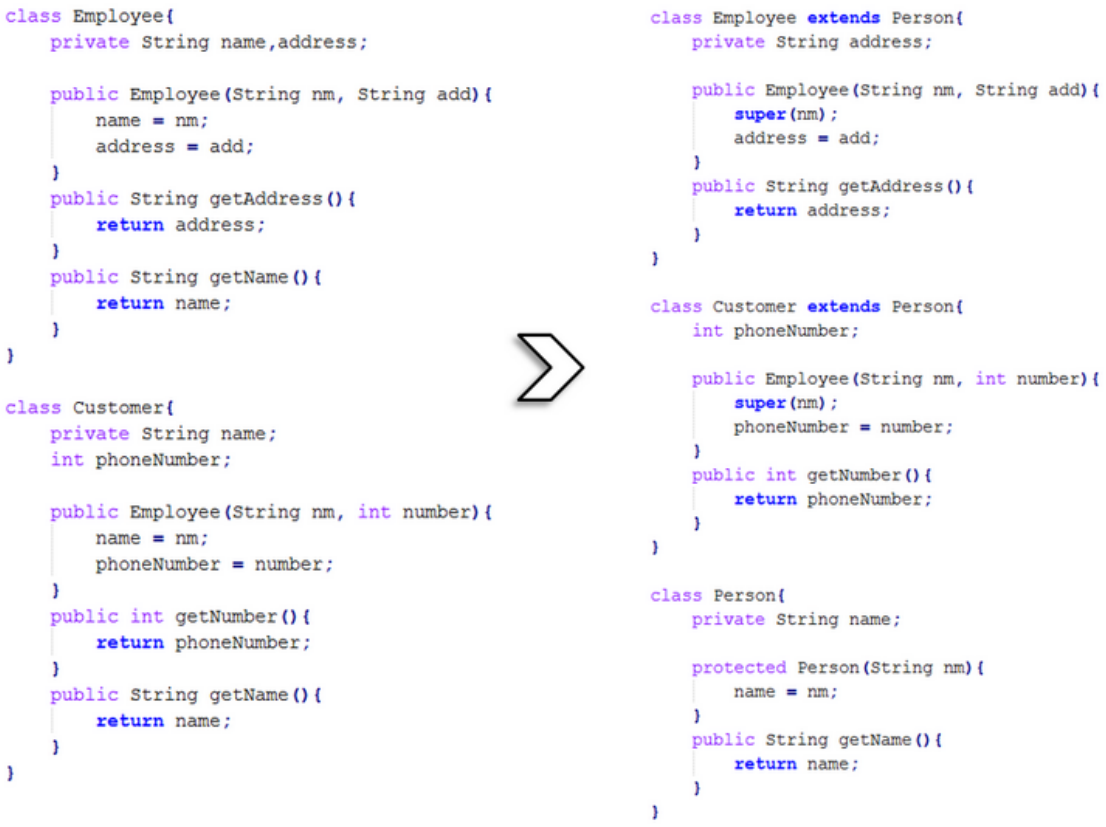
1.4 Δομή της Διατριβής

Η διατριβή περιέχει τα εξής κεφάλαια: Αρχικά στο Κεφάλαιο 2 περιγράφεται το θεωρητικό υπόβαθρο της ανακατασκευής κώδικα πάνω στο οποίο στηρίχθηκε η εκπόνηση αυτής της εργασίας και επίσης αναφέρονται σχετικές δουλειές πάνω στην υποβοήθηση ανακατασκευής κώδικα. Στο Κεφάλαιο 3 περιγράφονται οι συσχετίσεις μεταξύ των τεχνικών ανακατασκευής που προτείνονται και παρουσιάζεται ο Χάρτης Ανακατασκευών. Στο Κεφάλαιο 4 περιλαμβάνεται ο σχεδιασμός του Refactoring Trip Advisor και παρουσιάζονται κάποια σενάρια χρήσης αυτού. Το Κεφάλαιο 5 παρουσιάζει την αξιολόγηση της εφαρμογής από διάφορους χρήστες μαζί με τα αποτελέσματα αυτής και τέλος στο Κεφάλαιο 6 παρουσιάζονται τα συμπεράσματα από αυτή την εργασία και κάποια πιθανά μελλοντικά σχέδια.

ΚΕΦΑΛΑΙΟ 2. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ ΚΑΙ ΣΧΕΤΙΚΗ ΔΟΥΛΕΙΑ

2.1 Ορισμός Ανακατασκευής Κώδικα

Η ανακατασκευή κώδικα είναι μια διαδικασία τροποποίησης ενός συστήματος λογισμικού βελτιώνοντας την εσωτερική δομή του κώδικα, αφήνοντας όμως την εξωτερική συμπεριφορά του αμετάβλητη [7]. Η ανακατασκευή κώδικα είναι ένας πειθαρχημένος τρόπος εκκαθάρισης του κώδικα που ελαχιστοποιεί τις πιθανότητες εισαγωγής σφαλμάτων στο σύστημα. Ουσιαστικά η ανακατασκευή τροποποιεί τα μη λειτουργικά χαρακτηριστικά του συστήματος βελτιώνοντας τελικά τον σχεδιασμό του. Στην πράξη, ανακατασκευή κάνουμε εφαρμόζοντας μια σειρά από τυποποιημένες βασικές μικρό-ανακατασκευές η κάθε μια από τις οποίες επιφέρει μια μικρή αλλαγή στο σχεδιασμό του κώδικα χωρίς να επηρεάζεται η εξωτερική του συμπεριφορά. Στο Σχήμα 2.1 παρουσιάζεται ένα απλό παράδειγμα μιας συγκεκριμένης τεχνικής ανακατασκευής κώδικα στην πράξη. Ο αρχικός κώδικα που βρίσκεται στα αριστερά του Σχήματος περιέχει δύο κλάσεις (Employee και Customer) τις οποίες αν παρατηρήσουμε θα δούμε πως το πεδίο name και η μέθοδος getName() περιέχεται και στις δύο. Σε αυτές μπορούμε να εφαρμόσουμε την τεχνική Extract Superclass και να εξάγουμε την κοινή αυτή συμπεριφορά σε μία υπερκλάση (Person), μετατρέποντας τες έπειτα σε υποκλάσεις αυτής όπως φαίνεται στα δεξιά του Σχήματος.



```
class Employee{
    private String name,address;

    public Employee(String nm, String add){
        name = nm;
        address = add;
    }
    public String getAddress(){
        return address;
    }
    public String getName(){
        return name;
    }
}

class Customer{
    private String name;
    int phoneNumber;

    public Employee(String nm, int number){
        name = nm;
        phoneNumber = number;
    }
    public int getNumber(){
        return phoneNumber;
    }
    public String getName(){
        return name;
    }
}
```

```
class Employee extends Person{
    private String address;

    public Employee(String nm, String add){
        super(nm);
        address = add;
    }
    public String getAddress(){
        return address;
    }
}

class Customer extends Person{
    int phoneNumber;

    public Employee(String nm, int number){
        super(nm);
        phoneNumber = number;
    }
    public int getNumber(){
        return phoneNumber;
    }
}

class Person{
    private String name;

    protected Person(String nm){
        name = nm;
    }
    public String getName(){
        return name;
    }
}
```

Σχήμα 2.1 Κώδικας πριν (Αριστερά) και μετά (Δεξιά) την Εφαρμογή Extract Superclass

2.2 Τα Οφέλη της Ανακατασκευής Κώδικα

Αφού καταγράψαμε τον ορισμό της ανακατασκευής κώδικα, αξίζει σε αυτό το σημείο να δει κανείς τι μπορεί να κερδίσει ανακατασκευάζοντας τον κώδικα του. Υπάρχουν δύο βασικά χαρακτηριστικά του λογισμικού που επωφελούνται από την ανακατασκευή:

- Συντηρησιμότητα: Η διόρθωση σφαλμάτων καθώς και η κατανόηση της πρόθεσης του κατασκευαστή του λογισμικού γίνεται ευκολότερη όταν ο πηγαίος κώδικας είναι ευανάγνωστος [6]. Αυτό μπορεί να επιτευχθεί με τη διάσπαση μεγάλων μονολιθικών ρουτίνων σε μια σειρά από ξεχωριστές συνοπτικές μεθόδους που η κάθε μια εξυπηρετεί μόνο ένα σκοπό. Μπορει επίσης να επιτευχθεί με τη μετακίνηση μιας μεθόδου σε μια καταλληλότερη κλάση, ή και με την απομάκρυνση παραπλανητικών σχολίων.

- Επεκτασιμότητα: Οι δυνατότητες μιας εφαρμογής μπορούν να επεκταθούν πιο εύκολα εάν χρησιμοποιεί αναγνωρίσιμα μοτίβα σχεδιασμού. Η βελτίωση της επεκτασιμότητας συνεισφέρει στην αύξηση της ευελιξίας του κώδικα παρέχοντας δυνατότητες που δεν υπήρχαν σε προηγούμενες φάσεις αυτού.

Ο σχεδιασμός του συστήματος επηρεάζει άμεσα την ποιότητα του παραγόμενου λογισμικού. Εφόσον σημειώθηκε στο προηγούμενο κεφάλαιο ότι η ανάπτυξη ποιοτικού λογισμικού είναι αναγκαία, η χρήση τεχνικών ανακατασκευής κώδικα για την βελτίωση αυτών των ιδιοτήτων του κώδικα κρίνεται απαραίτητη.

2.3 Διαδικασία Ανακατασκευής Κώδικα

Η χρήση ενεργειών ανακατασκευής συχνά ξεκινά μετά από την παρατήρηση ύποπτων σημείων στον κώδικα, γνωστά και ως code smells [7]. Για παράδειγμα μπορεί μια μέθοδος του συστήματος να είναι αρκετά μεγάλη ή σχεδόν ίδια με κάποια άλλη. Όταν κάτι τέτοιο εντοπιστεί, τότε εφαρμόζεται η κατάλληλη τεχνική ανακατασκευής. Πριν από την εφαρμογή μιας τεχνικής ανακατασκευής σε κάποιο τμήμα του πηγαίου κώδικα, είναι πρώτα απαραίτητο να οριστούν κατάλληλες αυτοματοποιημένες μονάδες ελέγχου. Οι δοκιμές αυτές χρησιμοποιούνται για να αποδείξουν ότι η συμπεριφορά του κώδικα είναι σωστή πριν από την εκτέλεση της ανακατασκευής. Αν βρεθεί ότι υπάρχει λάθος, τότε είναι γενικά καλύτερο να διορθωθεί αυτό πρώτα γιατί αλλιώς είναι δύσκολο να γίνει διάκριση μεταξύ των αποτυχιών που εισήγαγε η ανακατασκευή και τις αποτυχίες που υπήρχαν ήδη εκεί. Μετά την ανακατασκευή, οι δοκιμές επαναλαμβάνονται για να ελεγχθεί ότι αυτή δεν επηρέασε την λειτουργικότητα του κώδικα. Η διαδικασία αυτή είναι επαναληπτική, ελέγχοντας για σφάλματα μετά από κάθε μικρό-ανακατασκευή έτσι ώστε εάν όλα λειτουργούν σωστά, να εφαρμοστεί η επόμενη. Αυτό επαναλαμβάνεται μέχρι οι σχεδιαστικές βελτιώσεις που επιφέρουν το σύνολο των μικρό-ανακατασκευών να είναι ικανοποιητικές. Τέλος η διαδικασία αυτή είναι αρκετά αποτελεσματική καθώς πολλές φορές γλιτώνει σημαντικό χρόνο στην ανάπτυξη και καθιστά την ανακατασκευή κώδικα ασφαλέστερη.

2.4 Κατηγοριοποίηση των Τεχνικών Ανακατασκευής

Ιστορικά η πρώτη αναφορά σχετικά με την ανακατασκευή κώδικα έγινε από τον William Griswold [8] το 1991 και ακολουθήθηκε ένα χρόνο αργότερα από τον William Opdyke [9] ο οποίος επικεντρώθηκε περισσότερο στην ανακατασκευή αντικειμενοστρεφούς κώδικα. Πλέον ως μια κοινώς αποδεκτή αναφορά πάνω στην ανακατασκευή κώδικα χρησιμοποιείται ο κατάλογος των τεχνικών ανακατασκευής που καταγράφει στο βιβλίο του ο Martin Fowler [7]. Σε αυτό γίνεται μια προσπάθεια για μια οργανωμένη καταγραφή των διαφόρων τεχνικών που υπάρχουν, προσφέροντας επίσης χρήσιμες πληροφορίες όπως το πότε και με ποιο τρόπο θα πρέπει κάποιος να τις εφαρμόσει στον κώδικα του. Παράλληλα με την καταγραφή των τεχνικών γίνεται και μια ομαδοποίηση αυτών σε διαφορετικές κατηγορίες ανάλογα με το ποια στοιχεία του κώδικα βελτιώνουν. Στη συνέχεια αναφέρονται οι έξι κατηγορίες που παρουσιάζονται στο [7] και δίνεται μια συνοπτική περιγραφή για τον τύπο των τεχνικών ανακατασκευής που αυτές εμπεριέχουν.

- Συγκρότηση μεθόδων: Απαρτίζεται από τεχνικές που βελτιώνουν την εσωτερική σύσταση των μεθόδων. Η χρήση αυτών απλοποιεί των κώδικα και συνήθως μειώνει το μέγεθός του.
- Μετακίνηση στοιχείων μεταξύ αντικειμένων: Περιέχει τεχνικές που σκοπό έχουν την σωστή ανάθεση καθηκόντων στις διάφορες δομές του κώδικα. Η χρήση τους έχει ως αποτέλεσμα ο κώδικας να περιέχει κλάσεις και μεθόδους με ξεκάθαρη λειτουργία και σκοπό στο πρόγραμμα.
- Οργάνωση δεδομένων: Αποτελείται από τεχνικές που προσπαθούν να κάνουν την διαχείριση των δεδομένων στον κώδικα πιο εύκολη. Η χρήση τους μετατρέπει και αντικαθιστά διαφόρους τύπους δεδομένων στον κώδικα βελτιώνοντας την εσωτερική του οργάνωση.
- Απλοποίηση κατηγορηματικών εκφράσεων: Προσφέρει τεχνικές που απλουστεύουν την πολυπλοκότητα της συνδυαστικής λογικής. Η χρήση τους βελτιώνει την αναγνωσιμότητα των δομών στον κώδικα που περιέχουν κατηγορηματικούς όρους.
- Βελτίωση κλήσεων μεθόδων: Παρέχει τεχνικές που επηρεάζουν τις διάφορες μεθόδους του κώδικα ώστε η χρήση αυτών να είναι απλή και κατανοητή. Η χρήση τους συνεισφέρει σε απλοποιημένα πρότυπα μεθόδων.

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

2.5 Περιγραφή των Τεχνικών Ανακατασκευής

Ύστερα από την παρουσίαση των διαφόρων κατηγοριών των τεχνικών ανακατασκευής σειρά έχει η ατομική περιγραφή αυτών. Η περιγραφή σε αυτή την ενότητα θα γίνει ανά κατηγορία και καθώς το πλήθος των τεχνικών είναι σχετικά μεγάλο θα δοθούν συνοπτικές πληροφορίες σχετικά με το τι κάνει, τι βελτιώνει και πότε πρέπει να εφαρμοστεί κάθε μια από αυτές.

2.5.1 Συγκρότηση Μεθόδων

Extract Method

Η εξαγωγή μεθόδου αποτελεί μια από τις πιο απλές και ίσως η πιο δημοφιλής τεχνική ανακατασκευής κώδικα. Η τεχνική αυτή παίρνει ένα κομμάτι κώδικα από μια μέθοδο και το εξάγει σε μια νέα, αντικαθιστώντας έπειτα αυτό το κομμάτι με κλήση προς τη νέα μέθοδο. Η τεχνική αυτή εφαρμόζεται συνήθως σε μεγάλου μήκους και πολύπλοκες μεθόδους με σκοπό την μείωση του μήκους τους και την απλοποίηση-επεξήγηση συγκεκριμένων κομματιών κώδικα.

Inline Method

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας όλες τις κλήσεις μια συγκεκριμένης μεθόδου με ολόκληρο τον κώδικα αυτής. Μετά την αντικατάσταση η μέθοδος αυτή δεν έχει κάποιο λόγο ύπαρξης και αφαιρείται από το σύστημα. Η τεχνική αυτή χρησιμοποιείται συνήθως σε μεθόδους μικρού μήκους, των οποίων ο κώδικας είναι απλός και ξεκάθαρος, μειώνοντας έτσι το πλήθος των ασήμαντων κλήσεων στο σύστημα.

Inline Temp

Η τεχνική αυτή αντικαθιστά όλες τις εμφανίσεις μιας προσωρινής μεταβλητής με τον κώδικα υπολογισμού της. Συνήθως ο υπολογισμός αυτής της μεταβλητής είναι το αποτέλεσμα μιας απλής κλήσης μεθόδου η οποία μπορεί να γίνει απευθείας χωρίς την χρήση της προσωρινής μεταβλητής. Η αντικατάσταση αυτή των προσωρινών μεταβλητών συνήθως γίνεται για να διευκολύνει την εφαρμογή άλλων τεχνικών.

Replace Temp with Query

Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κώδικα υπολογισμού μιας προσωρινής μεταβλητής σε μια νέα μέθοδο και αντικαθιστώντας όλες τις εμφανίσεις της με κλήση προς τη μέθοδο αυτή. Εφαρμόζοντας την τεχνική αυτή, ο κώδικας υπολογισμού της μεταβλητής γίνεται πλέον προσβάσιμος και έξω από την μέθοδο που αυτή ορίζεται. Επίσης η εξάλειψη των προσωρινών μεταβλητών των μεθόδων διευκολύνει την εφαρμογή άλλων τεχνικών όπως Extract Method.

Introduce Explaining Variable

Η τεχνική αυτή εφαρμόζεται εισάγοντας προσωρινές μεταβλητές στον κώδικα για την επεξήγηση πολύπλοκων εκφράσεων. Ουσιαστικά οι μεταβλητές χρησιμοποιούνται για να κρατήσουν ενός μέρους μιας σύνθετης έκφρασης και ονομάζονται κατάλληλα έτσι ώστε να μπορεί κανείς καταλάβει εύκολα την συνολική έκφραση.

Split Temporary Variable

Η τεχνική αυτή εφαρμόζεται διασπώντας προσωρινές μεταβλητές στον κώδικα που έχουν παραπάνω από μια αρμοδιότητες. Με εξαίρεση τις συναθροιστικές μεταβλητές (π.χ., μετρητές βρόγχων) σε κάθε προσωρινή μεταβλητή τίθεται μια συγκεκριμένη τιμή μια μόνο φορά. Στις περιπτώσεις που γίνονται παραπάνω αναθέσεις μπορεί να υπάρξει σύγχυση για τον ρόλο της μεταβλητής στον κώδικα και θα πρέπει αυτή να διασπαστεί.

Remove Assignments to Parameters

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας τις αναθέσεις που γίνονται στις παραμέτρους μιας μεθόδου με αναθέσεις σε μια προσωρινή μεταβλητή της παραμέτρου. Η χρήση της τεχνικής γίνεται διότι στην Java γίνεται πέρασμα τιμής αντί για πέρασμα αναφοράς και οποιαδήποτε αλλαγή στις παραμέτρους δεν αντανακλάται στην καλούσα μέθοδο.

Replace Method with Method Object

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια ολόκληρη μέθοδο με ένα νέο αντικείμενο βασισμένο σε αυτή. Ουσιαστικά αυτή η τεχνική μετατρέπει μια μέθοδο σε μια νέα κλάση η οποία έχει ως πεδία τις παραμέτρους και τις προσωρινές μεταβλητές της μεθόδου και μια μέθοδο με τον κώδικα της αρχικής. Η τεχνική αυτή συνήθως χρησιμοποιείται όταν μεγάλος αριθμός των προσωρινών μεταβλητών σε μια μέθοδο καθιστούν δύσκολη την εφαρμογή Extract Method στον κώδικα της.

Substitute Algorithm

Η τεχνική αυτή εφαρμόζεται μετατρέποντας η αντικαθιστώντας πλήρως κάποιον αλγόριθμο του συστήματος με έναν άλλον. Η αλλαγή αυτή συνήθως γίνεται με κάποιον αλγόριθμο ο οποίος είναι λιγότερο πολύπλοκος και πιο αποτελεσματικός.

2.5.2 Μετακίνηση Στοιχείων Μεταξύ Αντικειμένων

Move Method

Η τεχνική αυτή εφαρμόζεται μετακινώντας μια μέθοδο από την κλάση που αυτή ανήκει σε μια άλλη. Η τεχνική χρησιμοποιείται όταν παρουσιάζεται κάποιου είδους σύζευξης μεταξύ των κλάσεων όπως για παράδειγμα όταν μια μέθοδος χρησιμοποιεί μεθόδους και πεδία κάποιας κλάσης περισσότερο από αυτήν στην οποία ανήκει. Η εφαρμογή αυτής της τεχνικής συμβάλει στην δημιουργία απλών κλάσεων με ξεκάθαρες αρμοδιότητες.

Move Field

Η τεχνική αυτή εφαρμόζεται μετακινώντας ένα πεδίο από την κλάση που αυτό ανήκει σε μια άλλη. Η τεχνική αυτή χρησιμοποιείται όταν το πεδίο μιας κλάσης χρησιμοποιείται από άλλες περισσότερο από την ίδια.

Extract Class

Η τεχνική αυτή εφαρμόζεται μετακινώντας πεδία και μεθόδους μιας κλάσης σε μια νέα. Η τεχνική αυτή χρησιμοποιείται συνήθως όταν το μέγεθος των κλάσεων μεγαλώνει και αυτές καταλήγουν να έχουν πολλές αρμοδιότητες στο πρόγραμμα. Σημάδια στον κώδικα όπως η παράλληλη αλλαγή ενός υποσυνόλου των δεδομένων μιας κλάσης ή η εξάρτηση μεταξύ συγκεκριμένων μεθόδων και πεδίων υποδεικνύουν την εφαρμογή αυτής της τεχνικής.

Inline Class

Η τεχνική αυτή εφαρμόζεται μετακινώντας όλα τα πεδία και τις μεθόδους μιας κλάσης σε μια άλλη και διαγράφοντας την πρώτη μετά την μεταφορά. Η τεχνική αυτή συνήθως χρησιμοποιείται σε μικρές κλάσεις που δεν έχουν λόγο ύπαρξης από μόνες τους στο πρόγραμμα.

Hide Delegate

Η τεχνική αυτή εφαρμόζεται δημιουργώντας μια μέθοδο στην κλάση εξυπηρετητή για κάθε μέθοδο που η κλάση πελάτη καλεί από την κλάση αντιπρόσωπο. Μετά την δημιουργία η κλάση πελάτη καλεί τις νέες μεθόδους της κλάσης εξυπηρετητή οι οποίες κάνουν μια απλή κλήση στις μεθόδους της κλάσης αντιπροσώπου. Η τεχνική αυτή χρησιμοποιείται για την απόκρυψη λεπτομερειών από των κλάσεων πελάτη.

Remove Middle Man

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας τις κλήσεις των μεθόδων στην κλάση εξυπηρετητή από την κλάση πελάτη με απευθείας κλήσεις προς τις μεθόδους της κλάσης αντιπροσώπου. Μετά την αντικατάσταση οι αντίστοιχες μέθοδοι στην κλάση

εξυπηρετητή αφαιρούνται. Η τεχνική αυτή χρησιμοποιείται όταν το πλήθος των μεθόδων αντιπρόσωπων στην κλάση εξυπηρετητή γίνεται μεγάλο.

Introduce Foreign Method

Η τεχνική αυτή εφαρμόζεται προσθέτοντας μια επιπλέον λειτουργία στην κλάση εξυπηρετητή δημιουργώντας όμως την μέθοδο που την υλοποιεί στην κλάση πελάτη. Η τεχνική αυτή χρησιμοποιείται όταν δεν υπάρχουν τα κατάλληλα δικαιώματα ώστε να γίνουν αλλαγές στην κλάση εξυπηρετητή για την προσθήκη μιας νέας λειτουργίας.

Introduce Local Extension

Η τεχνική αυτή εφαρμόζεται προσθέτοντας επιπλέον λειτουργίες στην κλάση εξυπηρετητή υλοποιώντας αυτές όμως ως μια νέα υποκλάση της. Η τεχνική αυτή χρησιμοποιείται όταν δεν υπάρχουν τα κατάλληλα δικαιώματα ώστε να γίνουν αλλαγές στην κλάση εξυπηρετητή για την προσθήκη πάνω από μίας λειτουργίας.

2.5.3 Οργάνωση Δεδομένων

Self Encapsulate Field

Η τεχνική αυτή εφαρμόζεται δημιουργώντας τις κατάλληλες get και set μεθόδους για την πρόσβαση σε ένα πεδίο της κλάσης. Η μετατροπή αυτή σε έμμεση πρόσβαση των πεδίων είναι ιδιαίτερα χρήσιμή καθώς επιτρέπει τις υποκλάσεις να ορίσουν τον τρόπο διαχείρισης των δεδομένων όπως αυτές επιθυμούν.

Replace Data Value with Object

Η τεχνική αυτή εφαρμόζεται μετατρέποντας μια μεταβλητή σε αντικείμενο μιας νέας κλάσης. Η τεχνική αυτή χρησιμοποιείται όταν τα δεδομένα τα οποία αντιπροσωπεύουν κάποιες μεταβλητές δεν είναι επαρκή και θέλουμε να προσθέσουμε επιπλέον πληροφορίες για κάθε μια από αυτές.

Change Value to Reference

Η τεχνική αυτή εφαρμόζεται μετατρέποντας ένα αντικείμενο τιμής σε ένα αντικείμενο αναφοράς. Η τεχνική αυτή χρησιμοποιείται όταν θέλουμε τα αντικείμενα του συστήματος να αναπαριστούν μια ξεχωριστή οντότητα το κάθε ένα και οποιαδήποτε αλλαγή να αντικατοπτρίζεται σε όποιον έχει αναφορά σε αυτά.

Change Reference to Value

Η τεχνική αυτή εφαρμόζεται μετατρέποντας ένα αντικείμενο αναφοράς σε ένα αντικείμενο τιμής. Η τεχνική αυτή χρησιμοποιείται όταν η διαχείριση των αντικειμένων αναφοράς γίνεται πολύπλοκη και θέλουμε τα δεδομένα που αναπαριστούμε με τα αντικείμενα να παραμένουν αμετάβλητα.

Replace Array with Object

Η τεχνική αυτή εφαρμόζεται μετατρέποντας μια μεταβλητή τύπου πίνακα σε αντικείμενο μιας νέας κλάσης. Η τεχνική αυτή χρησιμοποιείται όταν στον πίνακα αποθηκεύονται τιμές διαφορετικού τύπου μεταξύ τους (π.χ. String στην πρώτη θέση, int στην δεύτερη θέση κ.α.) και δυσκολεύεται η ανάγνωση του.

Duplicate Observed Data

Η τεχνική αυτή εφαρμόζεται αντιγράφοντας ένα σύνολο δεδομένων και παρέχοντας μια κλάση η οποία θα συγχρονίζει τις αλλαγές που θα γίνονται σε αυτά. Η τεχνική αυτή χρησιμοποιείται συνήθως σε συστήματα που έχουν υλοποιημένη κάποια γραφική διεπαφή και υπάρχει διαχωρισμός μεταξύ του κώδικα που τη διαχειρίζεται και του κώδικα που διαχειρίζεται τους υπόλοιπους υπολογισμούς.

Change Unidirectional Association to Bidirectional

Η τεχνική αυτή εφαρμόζεται σε ζευγάρια κλάσεων που έχουν μονόδρομη συσχέτιση μεταξύ τους προσθέτοντας έναν δείκτη προς τα πίσω μετατρέποντας την έτσι σε αμφίδρομη. Η τεχνική αυτή χρησιμοποιείται όταν υπάρχει ανάγκη πρόσβασης της

κλάσης, προς την οποία υπάρχει η μονόδρομη συσχέτιση, σε στοιχεία της κλάσης που αναφέρεται σε αυτήν.

Change Bidirectional Association to Unidirectional

Η τεχνική αυτή εφαρμόζεται σε ζευγάρια κλάσεων που έχουν αμφίδρομη συσχέτιση αφαιρώντας μια από τις αναφορές μετατρέποντας την έτσι σε μονόδρομη. Η τεχνική αυτή χρησιμοποιείται όταν δεν αξιοποιείται η αμφίδρομη συσχέτιση πλήρως και μια από τις δύο αναφορές είναι άχρηστη. Εφαρμόζοντας αυτή την τεχνική μειώνεται η σύζευξη και η πολυπλοκότητα στο σύστημα.

Replace Magic Number with Symbolic Constant

Η τεχνική αυτή εφαρμόζεται ορίζοντας μια νέα αριθμητική σταθερά για κάποιον αριθμό με συγκεκριμένη σημασία στον κώδικα (π.χ., επιτάχυνση της βαρύτητας: 9.8) και αντικαθιστώντας όλες τις εμφανίσεις του με αυτή. Η τεχνική αυτή χρησιμοποιείται όταν ο συγκεκριμένος αριθμός χρησιμοποιείται συχνά στον κώδικα. Εφαρμόζοντας αυτή την τεχνική βελτιώνεται η αναγνωσιμότητα του κώδικα και οποιεσδήποτε μελλοντικές αλλαγές στον αριθμό καθίστανται ευκολότερες.

Encapsulate Field

Η τεχνική αυτή εφαρμόζεται αλλάζοντας τον τρόπο πρόσβασης ενός πεδίου μετατρέποντας το από δημόσιο σε ιδιωτικό και παρέχοντας της κατάλληλες get και set μεθόδους για αυτό. Η εφαρμογή αυτής της τεχνικής συνάδει με την φιλοσοφία ανάπτυξης λογισμικού που προτείνει τα δεδομένα να μην είναι δημόσια έτσι ώστε να υπάρχει διαχώριση στο σύστημα μεταξύ δεδομένων και συμπεριφοράς.

Encapsulate Collection

Η τεχνική αυτή εφαρμόζεται αλλάζοντας τις get και set μεθόδους για πεδία που είναι συλλογές αντικειμένων έτσι ώστε η get να επιστρέφει μια μη-τροποποιήσιμη συλλογή και η set να μετατραπεί σε add και remove για ένα συγκεκριμένο στοιχείο στη

συλλογή. Η τεχνική αυτή χρησιμοποιείται έτσι ώστε να μην μπορεί ο χρήστης να τροποποιήσει στοιχεία στην συλλογή χωρίς η κλάση να το γνωρίζει.

Replace Record with Data Class

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια εγγραφή με μια νέα κλάση. Η τεχνική αυτή χρησιμοποιείται συνήθως όταν γίνεται μεταφορά κώδικα από μη αντικειμενοστρεφής γλώσσες προγραμματισμού και υπάρχει ανάγκη μετατροπής των μη συμβατών δομών δεδομένων.

Replace Type Code with Class

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας ένα type code με μια νέα κλάση. Η τεχνική αυτή χρησιμοποιείται όταν η κλάση περιέχει ένα αριθμητικό type code ο οποίος δεν χρησιμοποιείται σε εντολή συνθήκης και δεν επηρεάζει την συμπεριφορά του προγράμματος. Η τεχνική αυτή βελτιώνει την αναγνωσιμότητα του κώδικα.

Replace Type Code with Subclasses

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας ένα type code με νέες υποκλάσεις. Η τεχνική αυτή χρησιμοποιείται όταν η κλάση περιέχει ένα type code το οποίο επηρεάζει την συμπεριφορά του κώδικα και συνήθως εμφανίζεται σε κάποια εντολή συνθήκης (switch, if-then-else).

Replace Type Code with State/Strategy

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας ένα type code με μια νέα state κλάση. Η τεχνική αυτή χρησιμοποιείται όταν η κλάση περιέχει ένα type code το οποίο δεν μπορεί να αντικατασταθεί από υποκλάσεις.

Replace Subclass with Fields

Η τεχνική αυτή εφαρμόζεται μετατρέποντας τις μεθόδους των υποκλάσεων σε πεδία της υπερκλάσης και έπειτα εξαλείφοντας τις υποκλάσεις. Η τεχνική αυτή

χρησιμοποιείται συνήθως όταν οι υποκλάσεις αποτελούνται μόνο από μεθόδους που επιστρέφουν σταθερές.

2.5.4 Απλοποίηση Κατηγορηματικών Εκφράσεων

Decompose Conditional

Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κώδικα που περιέχεται στην συνθήκη του if, στο then και στο else σε νέες μεθόδους. Η τεχνική αυτή χρησιμοποιείται όταν οι εντολές συνθήκης if στο πρόγραμμα γίνονται μεγάλες σε μέγεθος και πολύπλοκες για ανάγνωση.

Consolidate Conditional Expression

Η τεχνική αυτή εφαρμόζεται συνδυάζοντας διάφορες συνθήκες σε μία. Η τεχνική αυτή χρησιμοποιείται όταν στον κώδικα υπάρχουν περισσότερες από μία συνθήκες οι οποίες αν ισχύουν εκτελούν τον ίδιο ακριβώς κώδικα προσθέτοντας έτσι περιττούς ελέγχους στο πρόγραμμα.

Consolidate Duplicate Conditional Fragments

Η τεχνική αυτή εφαρμόζεται μετακινώντας ένα κομμάτι κώδικα το οποίο είναι ίδιο σε όλες τις διακλαδώσεις της κατηγορηματικής έκφρασης έξω από αυτήν. Χρησιμοποιώντας αυτή την τεχνική απλοποιείται ο κώδικας των κατηγορηματικών εκφράσεων και μειώνεται η επανάληψη κώδικα στο σύστημα.

Remove Control Flag

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια μεταβλητή σημαίας σε μια συνθήκη με εντολές break ή return. Η χρήση τέτοιου τύπου μεταβλητών δεν συνιστάται καθώς καθιστούν την ανάγνωση των κατηγορηματικών εκφράσεων πολύπλοκη.

Replace Conditional with Guard Clauses

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας τις εμφωλευμένες εντολές συνθήκης με συνθήκες φρουρούς. Η τεχνική αυτή χρησιμοποιείται όταν υπάρχει ανάγκη να δοθεί έμφαση στις συνθήκες ελέγχου του κατηγορήματος καθώς πρόκειται για σπάνιες περιπτώσεις με πιθανόν ασυνήθιστη συμπεριφορά στο πρόγραμμα.

Replace Conditional with Polymorphism

Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κώδικα σε κάθε κλαδί του κατηγορήματος σε μια μέθοδο μιας υποκλάσης. Όλες αυτές οι υποκλάσεις κληρονομούν από την ίδια κλάση και κάνουν override την ίδια abstract μέθοδο. Η τεχνική αυτή χρησιμοποιείται όταν στις συνθήκες του κατηγορήματος ελέγχεται ο τύπος ενός αντικειμένου και εκτελείται διαφορετικός κώδικας ανάλογα τον τύπο αυτό.

Introduce Null Object

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας τους ελέγχους με την τιμή null στις συνθήκες με ένα αντικείμενο null. Η τεχνική αυτή χρησιμοποιείται όταν στις συνθήκες του κατηγορήματος ελέγχεται πολλές φορές κάποια μεταβλητή με την τιμή null και κάνοντας έτσι την αντικατάσταση αυτή μπορεί να μειωθεί το υπολογιστικό κόστος.

Introduce Assertion

Η τεχνική αυτή εφαρμόζεται προσθέτοντας μια εντολή assertion σε ένα κομμάτι του κώδικα που υποθέτει κάτι για την κατάσταση του συστήματος. Η τεχνική αυτή χρησιμοποιείται όταν στο σύστημα υπάρχουν σημεία στα οποία υποτίθεται πως ισχύουν κάποια πράγματα για τον κώδικα χωρίς όμως να υπάρχει ο κατάλληλος κώδικας να το επιβεβαιώσει. Η χρήση των εντολών assertion μπορεί να εντοπίσει σφάλματα που μπορεί να προκύψουν από αυτές τις υποθέσεις.

2.5.5 Βελτίωση Κλήσεων Μεθόδων

Rename Method

Η τεχνική αυτή εφαρμόζεται αλλάζοντας το όνομα μιας μεθόδου έτσι ώστε να αντικατοπτρίζει τον σκοπό της. Η τεχνική αυτή χρησιμοποιείται όταν σε μια κλάση υπάρχουν πολλές μέθοδοι και τα σχόλια ή η ονομασία τους δεν δίνουν αρκετά ξεκάθαρη πληροφορία για αυτές.

Add Parameter

Η τεχνική αυτή εφαρμόζεται προσθέτοντας μια επιπλέον παράμετρο σε μια μέθοδο. Η τεχνική αυτή χρησιμοποιείται όταν η μέθοδος απαιτεί περισσότερη πληροφορία από το σύστημα για την εκτέλεση του κώδικα της.

Remove Parameter

Η τεχνική αυτή εφαρμόζεται αφαιρώντας μια από τις παραμέτρους μιας μεθόδου. Η τεχνική αυτή χρησιμοποιείται όταν κάποια από τις παραμέτρους μιας μεθόδου δεν χρησιμοποιείται στον κώδικα αυτής.

Separate Query from Modifier

Η τεχνική αυτή εφαρμόζεται διασπώντας μια μέθοδο η οποία επιστρέφει μια τιμή και μεταβάλει την κατάσταση ενός αντικειμένου σε δύο νέες ξεχωριστές μεθόδους, μια για κάθε ενέργεια. Η διαφοροποίηση αυτών των δύο ενεργειών έχει ως αποτέλεσμα τον διαχωρισμό των μεθόδων στο σύστημα σε αυτές που έχουν ορατή επιρροή στην κατάσταση του συστήματος και αυτές που απλά εκτελούν έναν υπολογισμό χωρίς παρενέργειες.

Parameterize Method

Η τεχνική αυτή εφαρμόζεται ενοποιώντας παρόμοιες μεθόδους σε μια της οποίας οι παράμετροι καθορίζουν τις διαφορετικές τιμές μεταξύ των αρχικών μεθόδων. Η

τεχνική αυτή χρησιμοποιείται όταν στο σύστημα υπάρχουν μέθοδοι που κάνουν το ίδιο πράγμα για διαφορετικές τιμές και έτσι υπάρχει η δυνατότητα συγχώνευσης αυτών.

Replace Parameter with Explicit Methods

Η τεχνική αυτή εφαρμόζεται διασπώντας μια μέθοδο σε νέες μεθόδους, μια για κάθε διαφορετική τιμή μιας συγκεκριμένης παραμέτρου της. Η τεχνική αυτή χρησιμοποιείται συνήθως όταν ο κώδικας μιας μεθόδου περιέχει μόνο μια εντολή συνθήκης η οποία εκτελεί διαφορετικό κώδικα για τις πιθανές τιμές μιας παραμέτρου.

Preserve Whole Object

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μερικές παραμέτρους μιας μεθόδου που περνάνε διαφορετικές τιμές ενός αντικειμένου με ολόκληρο το αντικείμενο. Η τεχνική αυτή χρησιμοποιείται όταν η πλειοψηφία των παραμέτρων μιας μεθόδου προέρχονται από τον υπολογισμό τιμών πάνω στο ίδιο αντικείμενο και υπάρχει η πιθανότητα να χρειαστούν και άλλες τιμές από αυτό στο μέλλον.

Replace Parameter with Method

Η τεχνική αυτή εφαρμόζεται αφαιρώντας μια παράμετρο μιας μεθόδου η οποία προκύπτει από τον υπολογισμό της κλήσης μια άλλης και αφήνοντας η κλήση της τελευταίας να γίνει μέσα στο σώμα της αρχικής. Η τεχνική αυτή χρησιμοποιείται όταν η κλήση κάποιας μεθόδου μπορεί να γίνει τοπικά στον κώδικα κάποιας άλλης και έτσι μπορεί να μειωθεί το μέγεθος της λίστας των παραμέτρων που αντιπροσωπεύουν απλά αποτελέσματα κλήσεων.

Introduce Parameter Object

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας ένα σύνολο παραμέτρων μιας μεθόδου με ένα νέο αντικείμενο. Η τεχνική αυτή χρησιμοποιείται όταν στο σύστημα υπάρχουν αρκετές μέθοδοι που χρησιμοποιούν το ίδιο συγκεκριμένο σύνολο παραμέτρων και

έτσι είναι δυνατή η αντικατάσταση τους με ένα αντικείμενο που τις περιέχει όλες μαζί.

Remove Setting Method

Η τεχνική αυτή εφαρμόζεται αφαιρώντας όλες τις set μεθόδους που αφορούν ένα πεδίο μιας κλάσης. Η τεχνική αυτή χρησιμοποιείται όταν για κάποιο πεδίο που έχει set μέθοδο στην κλάση, αποφασιστεί ότι δεν πρέπει να μεταβληθεί η τιμή του μετά την δημιουργία του αντικειμένου.

Hide Method

Η τεχνική αυτή εφαρμόζεται μεταβάλλοντας την ορατότητα μιας μεθόδου από δημόσια σε ιδιωτική. Η τεχνική αυτή χρησιμοποιείται όταν στο σύστημα παρατηρηθεί ότι μια μέθοδος χρησιμοποιείται μόνο από την κλάση της ή όταν είναι θέλουμε να απαγορεύσουμε τη χρήση της από εξωτερικές κλάσεις.

Replace Constructor with Factory Method

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια constructor μέθοδο μιας κλάσης με μια factory μέθοδο. Η τεχνική αυτή χρησιμοποιείται σε περιπτώσεις που οι δυνατότητες μιας απλής constructor μέθοδος είναι περιορισμένες (συνήθως σε περιπτώσεις που το αντικείμενο δημιουργείται με ένα type code).

Encapsulate Downcast

Η τεχνική αυτή εφαρμόζεται κάνοντας downcast την τιμή που επιστρέφει μια μέθοδος μέσα στο σώμα της και αλλάζοντας έπειτα τον τύπο επιστροφής της μεθόδου. Η τεχνική αυτή χρησιμοποιείται όταν έχουμε πληροφορία για τι τύπο δεδομένων ακριβώς επιστρέφει μια μέθοδος και δεν θέλουμε αυτός που θα την χρησιμοποιήσει να πρέπει να ψάξει για να τον βρει.

Replace Error Code with Exception

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας την τιμή επιστροφής μιας μεθόδου που αυτή χρησιμοποιεί για να υποδείξει κάποιο πιθανό σφάλμα με μια εξαίρεση. Η τεχνική αυτή χρησιμοποιείται βελτιώνοντας τον τρόπο διαχείρισης σφαλμάτων στο σύστημα μέσω των εξαιρέσεων.

Replace Exception with Test

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια εντολή εξαίρεσης με μια εντολή συνθήκης που ελέγχει την ίδια περίπτωση με την εξαίρεση. Η τεχνική αυτή χρησιμοποιείται όταν ο έλεγχος της περίπτωσης σφάλματος είναι αρκετά απλός και μπορεί να γίνει με μια απλή εντολή if χωρίς την χρήση περιττών εξαιρέσεων.

2.5.6 Βελτίωση Γενίκευσης

Pull Up Field

Η τεχνική αυτή εφαρμόζεται μετακινώντας τα κοινά πεδία μεταξύ δύο η περισσότερων υποκλάσεων στην υπερκλάση τους. Η τεχνική αυτή χρησιμοποιείται όταν παρατηρούνται πεδία που χρησιμοποιούνται με τον ίδιο τρόπο μέσα στις υποκλάσεις και έτσι μπορούν να ανέβουν ένα επίπεδο στην ιεραρχία.

Pull Up Method

Η τεχνική αυτή εφαρμόζεται μετακινώντας τις κοινές μεθόδους μεταξύ δύο η περισσότερων υποκλάσεων στην υπερκλάση τους. Η τεχνική αυτή χρησιμοποιείται όταν παρατηρούνται μέθοδοι που είτε περιέχουν πανομοιότυπο κώδικα είτε δίνουν τα ίδια αποτελέσματα για όλες τις πιθανές εισόδους. Επίσης η τεχνική αυτή μπορεί να εφαρμοστεί σε μεθόδους που κάνουν override μια μέθοδο της υπερκλάσης αλλά στην πράξη κάνουν το ίδιο πράγμα.

Pull Up Constructor Body

Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κοινό κώδικα μεταξύ των constructor υποκλάσεων σε έναν νέο constructor στην υπερκλάση παρέχοντας τις ανάλογες κλήσεις στα σημεία που έγινε η εξαγωγή. Η τεχνική αυτή χρησιμοποιείται όταν παρατηρούνται ίδια κομμάτια κώδικα μεταξύ των constructors των υποκλάσεων. Επειδή δεν μπορεί να γίνει ένα απλό Pull Up Method για τους constructors, θα πρέπει να δημιουργηθεί νέος constructor στην υπερκλάση και να γίνουν οι κατάλληλες κλήσεις προς αυτόν.

Push Down Method

Η τεχνική αυτή εφαρμόζεται μετακινώντας μια μέθοδο της υπερκλάσης σε μια συγκεκριμένη υποκλάση. Η τεχνική αυτή χρησιμοποιείται όταν παρατηρείται ότι η συμπεριφορά μιας μεθόδου της υπερκλάσης είναι σχετική μόνο για μια από τις υποκλάσεις της.

Push Down Field

Η τεχνική αυτή εφαρμόζεται μετακινώντας ένα πεδίο της υπερκλάσης σε μια συγκεκριμένη υποκλάση. Η τεχνική αυτή χρησιμοποιείται όταν παρατηρείται ότι ένα πεδίο της υπερκλάσης χρησιμοποιείται μόνο από μια συγκεκριμένη υποκλάση και όχι από τις υπόλοιπες.

Extract Subclass

Η τεχνική αυτή εφαρμόζεται εξάγοντας μερικά από τα χαρακτηριστικά μιας κλάσης σε μια νέα υποκλάση αυτής. Η τεχνική αυτή εφαρμόζεται όταν παρατηρείται όταν ένα υποσύνολο των χαρακτηριστικών μιας κλάσης χρησιμοποιείται μόνο για μερικά στιγμιότυπα αυτής.

Extract Superclass

Η τεχνική αυτή εφαρμόζεται εξάγοντας τα κοινά χαρακτηριστικά μεταξύ δύο η περισσότερων κλάσεων σε μια νέα υπερκλάση και μετατρέποντας τες σε υποκλάσεις

αυτής. Η τεχνική αυτή χρησιμοποιείται όταν παρατηρείται παρόμοια συμπεριφορά μεταξύ κλάσεων η οποία συνήθως υποδεικνύεται από την ομοιότητα των χαρακτηριστικών τους.

Extract Interface

Η τεχνική αυτή εφαρμόζεται εξάγοντας ένα υποσύνολο των χαρακτηριστικών μιας κλάσης σε μια νέα κλάση διεπαφής. Η τεχνική αυτή εφαρμόζεται όταν παρατηρείται ότι μερικά από τα χαρακτηριστικά μιας κλάσης χρησιμοποιούνται συχνά από πολλούς πελάτες στο σύστημα.

Collapse Hierarchy

Η τεχνική αυτή εφαρμόζεται συγχωνεύοντας μια υπερκλάση με μια υποκλάση αυτής. Η τεχνική αυτή χρησιμοποιείται όταν έπειτα από διάφορες ανακατασκευές στο σύστημα οι δύο αυτές κλάσεις καταλήγουν να μοιάζουν αρκετά έτσι ώστε η υπάρχουσα σχέση ιεραρχίας μεταξύ τους να μην έχει νόημα.

Form Template Method

Η τεχνική αυτή εφαρμόζεται εξάγοντας τον κοινό κώδικα μεταξύ παρόμοιων μεθόδων των υποκλάσεων σε μία μέθοδο ανά βήμα του κώδικα. Έπειτα εφαρμόζεται Pull Up Method στις αρχικές μεθόδους από τις οποίες εξάχθηκαν οι νέες. Η τεχνική αυτή χρησιμοποιείται όταν υπάρχει παρόμοιος κώδικας μεταξύ μεθόδων των υποκλάσεων ο οποίος όμως διαφέρει σε κάθε βήμα του και έτσι δεν μπορεί να γίνει Pull Up Method κατευθείαν στις μεθόδους αυτές.

Replace Inheritance with Delegation

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια σχέση κληρονομικότητας μεταξύ δύο κλάσεων σε απλή συσχέτιση. Η τεχνική αυτή χρησιμοποιείται όταν μια υποκλάση του συστήματος αξιοποιεί μόνο ένα μέρος των λειτουργιών που προσφέρονται στην υπερκλάση και έτσι δεν δικαιολογείται πλήρως η σχέση κληρονομικότητας μεταξύ αυτών.

Replace Delegation with Inheritance

Η τεχνική αυτή εφαρμόζεται αντικαθιστώντας μια σχέση συσχέτισης μεταξύ δύο κλάσεων σε σχέση κληρονομικότητας. Η τεχνική αυτή χρησιμοποιείται όταν μια από τις κλάσεις χρησιμοποιεί όλες τις μεθόδους της άλλης και θα ήταν ευκολότερη η πρόσβαση εάν γινόταν υποκλάση της.

2.6 Ανακατασκευή Κώδικα στην Πράξη

Στο προηγούμενες ενότητες δόθηκε ο ορισμός της ανακατασκευής κώδικα και παρουσιάστηκαν διάφορες τεχνικές ανακατασκευής που βελτιώνουν την ποιότητα του συστήματος. Αξίζει όμως σε αυτό το σημείο να δει κανείς τι πραγματικά γίνεται στη πράξη. Σύμφωνα με μια έρευνα [10] πάνω στα οφέλη της ανακατασκευής κώδικα, διαπιστώθηκε πως ο ορισμός της ανακατασκευής στη πράξη διαφέρει ανάμεσα στους κατασκευαστές λογισμικού. Οι κατασκευαστές πιστεύουν πως οι τεχνικές ανακατασκευής κώδικα που περιγράφονται στο [7] είναι βασικού επιπέδου και πως στην πράξη οι ανακατασκευές που κάνουν στο λογισμικό τους, περιλαμβάνουν την εφαρμογή μιας σειράς από αυτών των τεχνικών. Παίρνοντας υπόψη τα παραπάνω, σε αυτή τη διατριβή προτείνεται μια νέα μέθοδος ανακατασκευής κώδικα η οποία στηρίζεται πάνω στις συσχετίσεις μεταξύ των τεχνικών που περιγράφηκαν στην ενότητα 2.5. Οι συσχετίσεις αναλύονται εκτενέστερα στο Κεφάλαιο 3 όπου παράλληλα παρουσιάζεται το γράφημα αυτών. Μία παρόμοια εργασία στην οποία προτείνεται μια σειρά εφαρμογής των τεχνικών ανακατασκευής σύμφωνα με τις συσχετίσεις μεταξύ τους παρουσιάζεται στο [21]. Συγκριτικά με το [21], σε αυτή τη διατριβή παρουσιάζονται οι συσχετίσεις μεταξύ όλων των τεχνικών ανακατασκευής τους καταλόγου του Fowler και η σειρά εφαρμογής αυτών που προτείνεται είναι ανεξάρτητη του κώδικα του κάθε χρήστη. Επιπλέον, όπως αναλύεται παρακάτω, υλοποιείται και μια εφαρμογή για την υλοποίηση αυτής της ιδέας στην πράξη.

Η ανακατασκευή μπορεί να ωφελήσει σημαντικά την ποιότητα του κώδικα βελτιώνοντας τα χαρακτηριστικά του. Παρ’ όλα αυτά αυτή διαδικασία μπορεί να είναι πολύπλοκη, χρονοβόρα και επιρρεπής σε σφάλματα. Για την επίλυση αυτών των προβλημάτων έχουν αναπτυχθεί αρκετά εργαλεία τα οποία βοηθούν αυτοματοποιώντας είτε μέρος, είτε ολόκληρη την διαδικασία ανακατασκευής. Μια

από τις σημαντικότερες φάσεις της διαδικασίας αυτής αποτελεί ο εντοπισμός σημείων στον κώδικα τα οποία χρήζουν κάποιου είδους ανακατασκευής. Σε μια συστηματική έρευνα [11] που έγινε πάνω σε εργαλεία που αυτοματοποιούν αυτήν την φάση, βρέθηκε ότι για το 72% των τεχνικών που προτείνονται από τον Fowler [7] δεν υπήρχε υποστήριξη εντοπισμού στο σύνολο των εργαλείων. Ενώ αντίθετα για τις τεχνικές του Move Method, Extract Class και Extract Method υπάρχει υποστήριξη εντοπισμού από την πλειοψηφία των εργαλείων υποβοήθησης. Στα πλαίσια αυτής της διατριβής αναπτύσσεται ένα εργαλείο για την οπτικοποίηση του Χάρτη Ανακατασκευών μεταξύ των τεχνικών ανακατασκευής που αναφέρθηκε στην προηγούμενη ενότητα. Παράλληλα γίνεται μια προσπάθεια υλοποίησης αυτοματοποιημένων εντοπισμών για την κάλυψη σημαντικού εύρους των τεχνικών ανακατασκευής. Η παρουσίαση του εργαλείου υποβοήθησης Refactoring Trip Advisor και οι υλοποιημένες δυνατότητες αυτόματου εντοπισμού αναλύονται στο Κεφάλαιο 4.

ΚΕΦΑΛΑΙΟ 3. ΣΥΣΧΕΤΙΣΕΙΣ ΜΕΤΑΞΥ ΤΕΧΝΙΚΩΝ ΑΝΑΚΑΤΑΣΚΕΥΗΣ

3.1 Περιγραφή Συσχετίσεων

Όπως προαναφέρθηκε στην ενότητα 2.6, βασική συνεισφορά αυτής της διατριβής είναι η πρόταση μιας νέας μεθόδου στη διαδικασία ανακατασκευής λογισμικού. Η μέθοδος αυτή προτείνει η ανακατασκευή κώδικα να γίνεται με γνώμονα έναν Χάρτη που περιγράφει συσχετίσεις μεταξύ των τεχνικών ανακατασκευής. Πριν γίνει η παρουσίαση του Χάρτη αυτού θα πρέπει πρώτα να γίνει μια ανάλυση των τύπων των συσχετίσεων που προτείνονται. Οι τρεις συσχετίσεις προκύπτουν από την μελέτη του καταλόγου των τεχνικών ανακατασκευής που προτείνονται από τον Fowler [7]. Συγκεκριμένα για την εξαγωγή του είδους των συσχετίσεων μελετήθηκαν οι ενότητες Motivation, Mechanics και Example για κάθε τεχνική συσχέτισης. Στη συνέχεια, στις υποενότητες που ακολουθούν, δίνεται ο ορισμός για κάθε μια από τις τρεις συσχετίσεις μαζί με ένα παράδειγμα για το πώς εξάγαμε αυτή τη σχέση από το βιβλίο του Fowler και πως την οπτικοποιούμε τελικά στο γράφημα.

3.1.1 Σχέση Διαδοχής

Σύμφωνα με την ονομασία της η σχέση αυτή εκφράζει μια διαδοχή στην εκτέλεση των τεχνικών ανακατασκευής. Οι σχέσεις διαδοχής ουσιαστικά προτείνουν στον κατασκευαστή μια σειρά με την οποία θα πρέπει να εφαρμόσει τις τεχνικές ανακατασκευής στον κώδικα του. Η σχέση διαδοχής σημειώνεται στον Χάρτη Ανακατασκευών με μια συμπαγή κατευθυνόμενη ακμή μεταξύ των συσχετιζόμενων

κόμβων. Στο βιβλίο του Fowler υπάρχουν σημεία που υποδεικνύουν την εκτέλεση συγκεκριμένων τεχνικών ανακατασκευής **πριν** από άλλες:

"Before you can begin with Replace Conditional with Polymorphism you need to have the necessary inheritance structure. You may already have this structure from previous refactorings. If you don't have the structure, you need to create it. To create the inheritance structure you have two options: Replace Type Code with Subclasses and Replace Type Code with State/Strategy."

Πίνακας 3.1 Απόσπασμα από Mechanics της Τεχνικής Replace Conditional with Polymorphism [7]

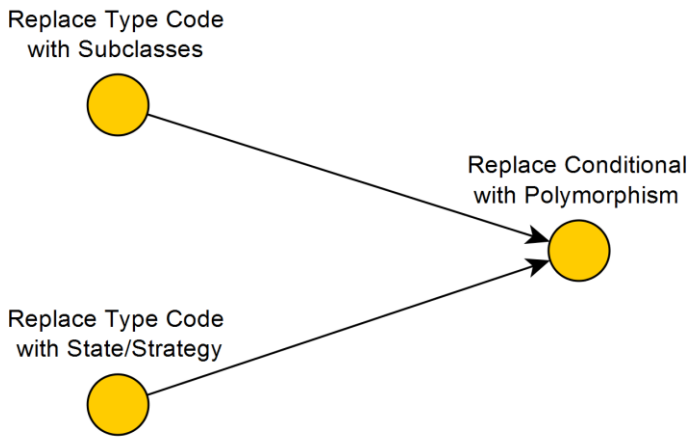
Επίσης υπάρχουν σημεία που προτείνουν την εκτέλεση συγκεκριμένων τεχνικών ανακατασκευής **μετά** από άλλες:

"Consolidating the conditional code is important for two reasons. First, it makes the check clearer by showing that you are really making a single check that's oring the other checks together. The sequence has the same effect, but it communicates carrying out a sequence of separate checks that just happen to be done together. The second reason for this refactoring is that it often sets you up for Extract Method. Extracting a condition is one of the most useful things you can do to clarify your code. It replaces a statement of what you are doing with why you are doing it."

Πίνακας 3.2 Απόσπασμα από Motivation της Τεχνικής Consolidate Conditional Expression [7]

Το πρώτο απόσπασμα (Πίνακας 3.1) προτείνει την εφαρμογή των τεχνικών Replace Type Code with Subclasses ή Replace Type Code with State/Strategy σε περίπτωση που δεν υπάρχει η κατάλληλη ιεραρχική δομή στον κώδικα για την εφαρμογή της Replace Conditional with Polymorphism. Το δεύτερο απόσπασμα (Πίνακας 3.2) προτείνει την εξαγωγή της σύνθετης συνθήκης που προκύπτει από την εφαρμογή της Consolidate Conditional Expression σε μια νέα μέθοδο καθώς αυτό βελτιώνει την

αναγνωσιμότητα του κατηγορήματος. Η οπτικοποίηση αυτών τον δύο αποσπασμάτων φαίνεται στο Σχήμα 3.1 για το πρώτο και στο Σχήμα 3.2 για το δεύτερο.



Σχήμα 3.1 Οπτικοποίηση Σχέσης Διαδοχής για Replace Conditional with Polymorphsm

Σχήμα 3.2 Οπτικοποίηση Σχέσης Διαδοχής για Consolidate Conditional Expression

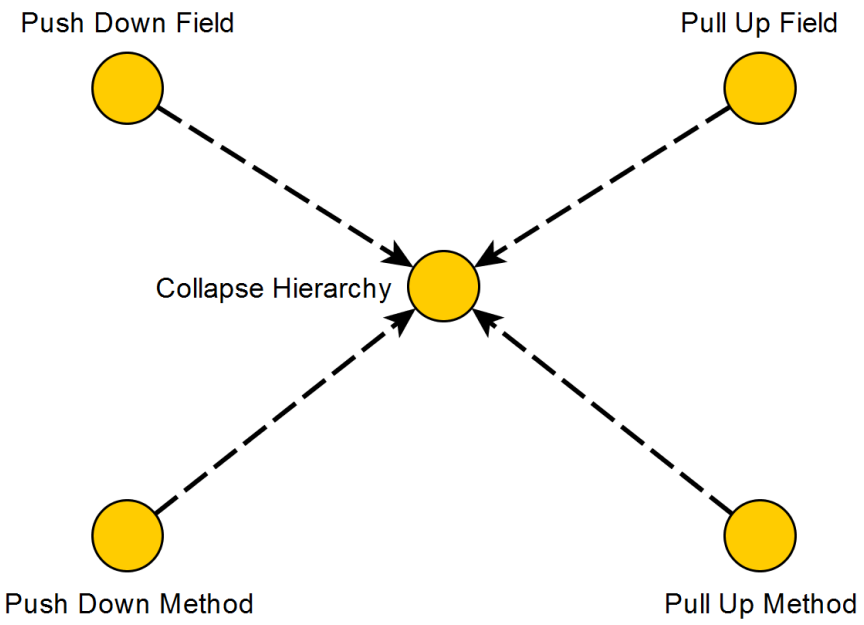
3.1.2 Σχέση "Μέρος από"

Μια σχέση "Μέρος από" μεταξύ δύο τεχνικών ανακατασκευής εκφράζει ότι η εφαρμογή της μιας ενδέχεται να περιλαμβάνει την εφαρμογή της άλλης. Αυτό συμβαίνει καθώς μερικές πολύπλοκες τεχνικές ανακατασκευής αποτελούνται από την συνάθροιση απλούστερων τεχνικών. Για την αναπαράσταση αυτών των συσχετίσεων χρησιμοποιείται η σχέση "Μέρος από". Η σχέση "Μέρος από" σημειώνεται στον Χάρτη Ανακατασκευών με μια διακεκομμένη κατευθυνόμενη ακμή μεταξύ των συσχετιζόμενων κόμβων. Ένα παράδειγμα εξαγωγής τέτοιας σχέσης από συγκεκριμένο απόσπασμα του βιβλίου του Fowler φαίνεται στον Πίνακα 3.3.

- *Choose which class is going to be removed: the superclass or the subclasses.*
- *Use Pull Up Field and Pull Up Method or Push Down Method and Push Down Field*
- *to move all the behavior and data of the removed class to the class with which it is being merged.*
- *Compile and test with each move.*
- *Adjust references to the class that will be removed to use the merged class. This will affect variable declarations, parameter types, and constructors.*
- *Remove the empty class.*
- *Compile and test.*

Πίνακας 3.3 Απόσπασμα από Motivation της Τεχνικής Collapse Hierarchy [7]

Το απόσπασμα αυτό αποτελεί τις οδηγίες βήμα προς βήμα για την εφαρμογή της τεχνικής ανακατασκευής Collapse Hierarchy. Σύμφωνα με αυτό, ανάλογα με το ποια κλάση θα επιλέξει να αφαιρέσει έτσι ώστε να συμπτυχθεί η ιεραρχία, ο προγραμματιστής θα πρέπει να εφαρμόσει μια σειρά από Pull Up Field και Pull Up Method (εάν αφαιρεθεί υποκλάση) ή Push Down Field και Push Down Method (εάν αφαιρεθεί υπερκλάση) σε πεδία και μεθόδους του κώδικα. Η συγκεκριμένη σχέση αναπαρίσταται γραφικά από το γράφημα του Σχήματος 3.3.



Σχήμα 3.3 Οπτικοποίηση Σχέσης "Μέρος από" για Collapse Hierarchy

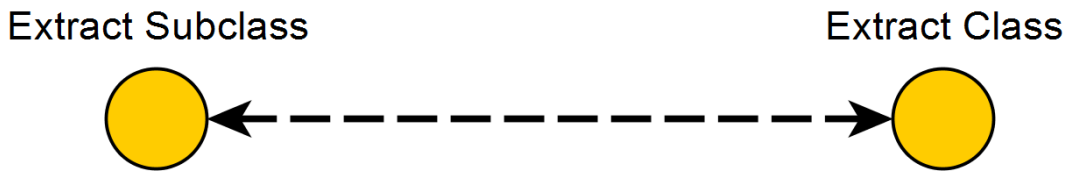
3.1.3 Σχέση "Αντί για"

Μια σχέση "Αντί για" μεταξύ δύο τεχνικών ανακατασκευής εκφράζει μια εναλλακτική επιλογή μεταξύ αυτών. Υπάρχουν περιπτώσεις στην διαδικασία ανακατασκευής όπου συγκεκριμένες ιδιομορφίες στον κώδικα καθιστούν την εφαρμογή μιας τεχνικής δύσκολη ή αδύνατη. Επίσης υπάρχουν περιπτώσεις που η εφαρμογή μιας διαφορετικής τεχνικής μπορεί να βοηθήσει τον χρήστη με μελλοντικές αλλαγές ή επεκτάσεις στον κώδικα. Η σχέση "Αντί για" προτείνει στον χρήστη την εφαρμογή μιας εναλλακτικής τεχνικής που ενδείκνυται για κάποια συγκεκριμένη περίπτωση και σκοπεύει στην παρόμοια βελτίωση του κώδικα με την αρχική. Η σχέση "Αντί για" σημειώνεται στον Χάρτη Ανακατασκευών με μια διακεκομμένη διπλά κατευθυνόμενη ακμή μεταξύ των συσχετιζόμενων κόμβων. Στο απόσπασμα του Πίνακα 3.4 φαίνεται ένα παράδειγμα που εμφανίζεται αυτή η σχέση:

"The main alternative to Extract Subclass is Extract Class. This is a choice between delegation and inheritance. Extract Subclass is usually simpler to do, but it has limitations. You can't change the class-based behavior of an object once the object is created. You can change the class-based behavior with Extract Class simply by plugging in different components."

Πίνακας 3.4 Απόσπασμα από Motivation της Τεχνικής Extract Subclass [7]

Το συγκεκριμένο απόσπασμα προτείνει δύο επιλογές στον κατασκευαστή στην περίπτωση που αυτός θέλει να εξάγει μέρος του κώδικα μιας κλάσης σε μια νέα κλάση. Συγκεκριμένα, σε περίπτωση που επιθυμεί η συμπεριφορά μιας κλάσης να μπορεί να αλλάξει όταν το αντικείμενο δημιουργηθεί θα πρέπει να εφαρμόσει Extract Class. Διαφορετικά του προτείνεται η Extract Subclass καθώς συνήθως αυτή είναι ευκολότερο να εφαρμοστεί. Η σχέση αυτή παρουσιάζεται σχηματικά στο γράφημα του Σχήματος 3.4.



Σχήμα 3.4 Οπτικοποίηση Σχέσης "Αντί για" για Extract Subclass

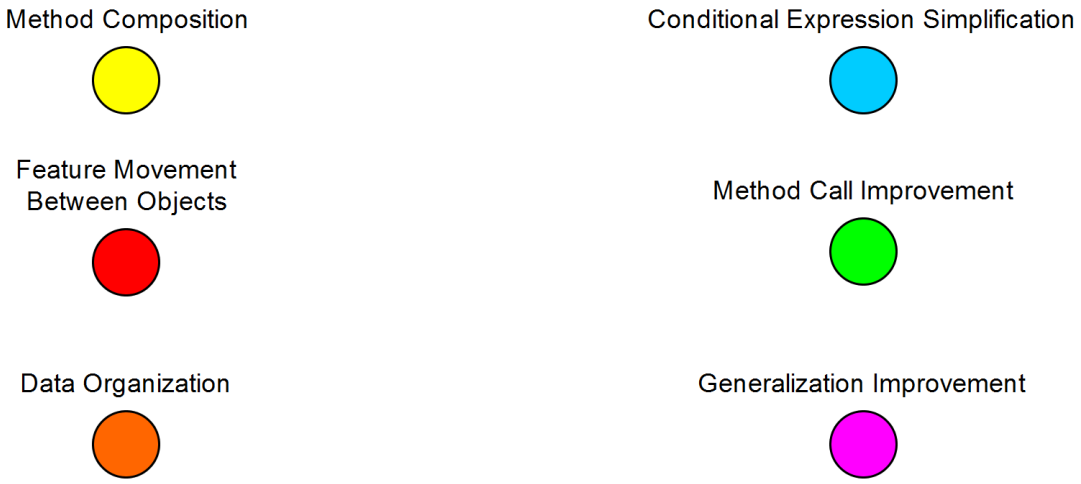
3.2 Χάρτης Ανακατασκευών

Ακολουθώντας την αναλυτική περιγραφή των τριών σχέσεων στην προηγούμενη ενότητα, σε αυτό το σημείο θα γίνει η παρουσίαση του Χάρτη των συσχετίσεων. Ο Χάρτης Ανακατασκευών είναι ουσιαστικά ένα γράφημα στο οποίο κάθε κόμβος αναπαριστά μια τεχνική ανακατασκευής και τα τρία διαφορετικά είδη ακμών μεταξύ των κόμβων αναπαριστούν τις τρεις σχέσεις που αναφέρθηκαν προηγουμένως. Όπως και για τον ορισμό των τριών σχέσεων, έτσι και για την δημιουργία του Χάρτη κύρια πηγή αποτελεί ο κατάλογος του Fowler [7]. Στην υποενότητα 3.2.1 γίνεται μια

τμηματική αναπαράσταση του Χάρτη με βάση τις διάφορες κατηγορίες τεχνικών ανακατασκευής ενώ στην 3.2.2 παρουσιάζεται ο συνολικός Χάρτης.

3.2.1 Χάρτης Ανακατασκευών ανά Κατηγορία Τεχνικών Ανακατασκευής

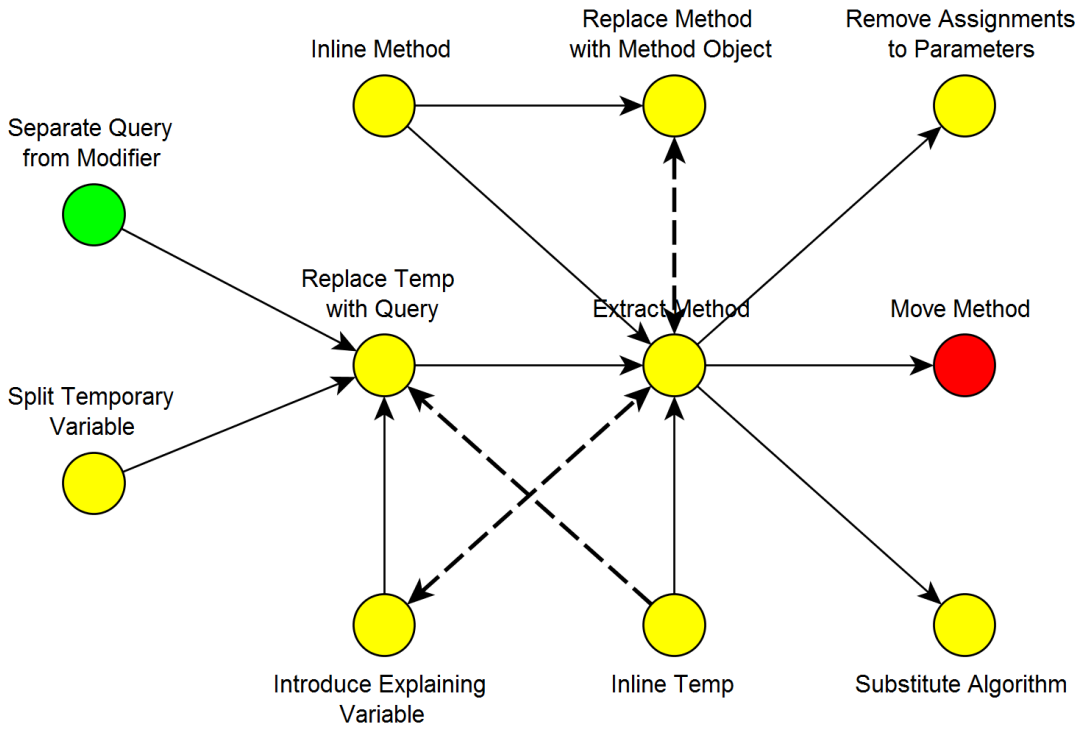
Σκοπός του Χάρτη Ανακατασκευών είναι η μοντελοποίηση των συσχετίσεων μεταξύ των τεχνικών ανακατασκευής που προτάθηκαν από τον Fowler [7]. Ακολουθώντας τον τρόπο με τον οποίο αυτές περιγράφηκαν στην ενότητα 2.4, θα γίνει η παρουσίαση του Χάρτη Ανακατασκευών για κάθε μια από τις έξι κατηγορίες τεχνικών ανακατασκευής. Για κάθε κατηγορία, στον Χάρτη της παρουσιάζονται οι συσχετίσεις μεταξύ των εσωτερικών τεχνικών (τεχνικές που ανήκουν σε αυτή την κατηγορία) της, αλλά και συσχετίσεις μεταξύ των εσωτερικών και εξωτερικών (τεχνικές που δεν ανήκουν σε αυτή την κατηγορία) τεχνικών. Για την ξεκάθαρη οπτικοποίηση μεταξύ εσωτερικών και εξωτερικών τεχνικών, για κάθε κατηγορία έχει ανατεθεί ένα διαφορετικό χρώμα για τις εσωτερικές τεχνικές της. Τα χρώματα για κάθε κατηγορία φαίνονται στο Σχήμα 3.4.



Σχήμα 3.4 Χρώμα εσωτερικών τεχνικών για κάθε κατηγορία

Στη συνέχεια παρουσιάζονται ο Χάρτης κάθε κατηγορίας ακολουθούμενος από ένα σύντομο σχολιασμό για τις συσχετίσεις μεταξύ των τεχνικών ανακατασκευής.

Method Composition

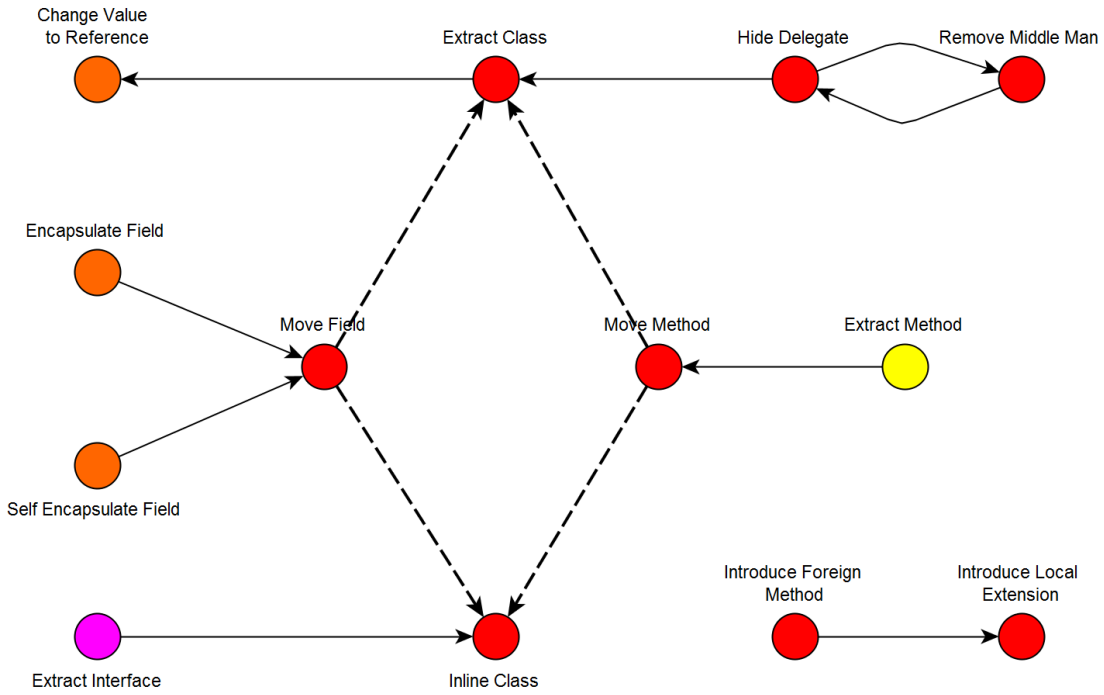


Σχήμα 3.5 Χάρτης Ανακατασκευών για “Method Composition”

Σκοπός των τεχνικών ανακατασκευής που παρουσιάζονται σε αυτή την κατηγορία είναι η μείωση του μεγέθους και γενικότερα η απλοποίηση του κώδικα των μεθόδων. Από τις σημαντικότερες τεχνικές ανακατασκευής που το πετυχαίνουν αυτό αποτελεί η Extract Method. Το γεγονός αυτό επιβεβαιώνεται αν παρατηρήσει κανείς τον Χάρτη Ανακατασκευών καθώς η Extract Method έχει τις περισσότερες συσχετίσεις στο γράφημα. Λόγω της σημαντικότητας αυτής της τεχνικής, αυτή αποτελεί ένα καλό σημείο εκκίνησης της πλοήγησης στον Χάρτη αυτής της κατηγορίας κατά την διαδικασία της ανακατασκευής κώδικα. Παρατηρώντας κανείς τον Χάρτη μπορεί να δει πως πριν την χρήση της Extract Method προτείνεται η εφαρμογή τεχνικών ανακατασκευής όπως οι Inline Method, Replace Temp with Query και Inline Temp για την διαχείριση των προσωρινών μεταβλητών και των μεθόδων που ο κώδικας είναι πολύ ασήμαντος για να δικαιολογεί την ύπαρξή τους. Η εξάλειψη των διάφορων προσωρινών μεταβλητών και μεθόδων έχει σκοπό την διευκόλυνση της εξαγωγής κώδικα σε νέες μεθόδους κατά την εφαρμογή της Extract Method. Εάν η αντιμετώπιση των προσωρινών μεταβλητών είναι πολύπλοκη και η εξαγωγή κώδικα

φαίνεται δύσκολη τότε η Replace Method with Method Object και Introduce Explaining Variable προτείνονται ως εναλλακτικές επιλογές. Έπειτα από την εφαρμογή της Extract Method, καθώς στο σύστημα προστίθενται νέες μέθοδοι, ο Χάρτης προτείνει την ανίχνευση τυχόν ευκαιριών εφαρμογής Remove Assignments to Parameters και Substitute Algorithm στον κώδικα τον μεθόδων ή την πιθανή μετακίνηση αυτών σε άλλες κλάσεις.

Feature Movement Between Objects

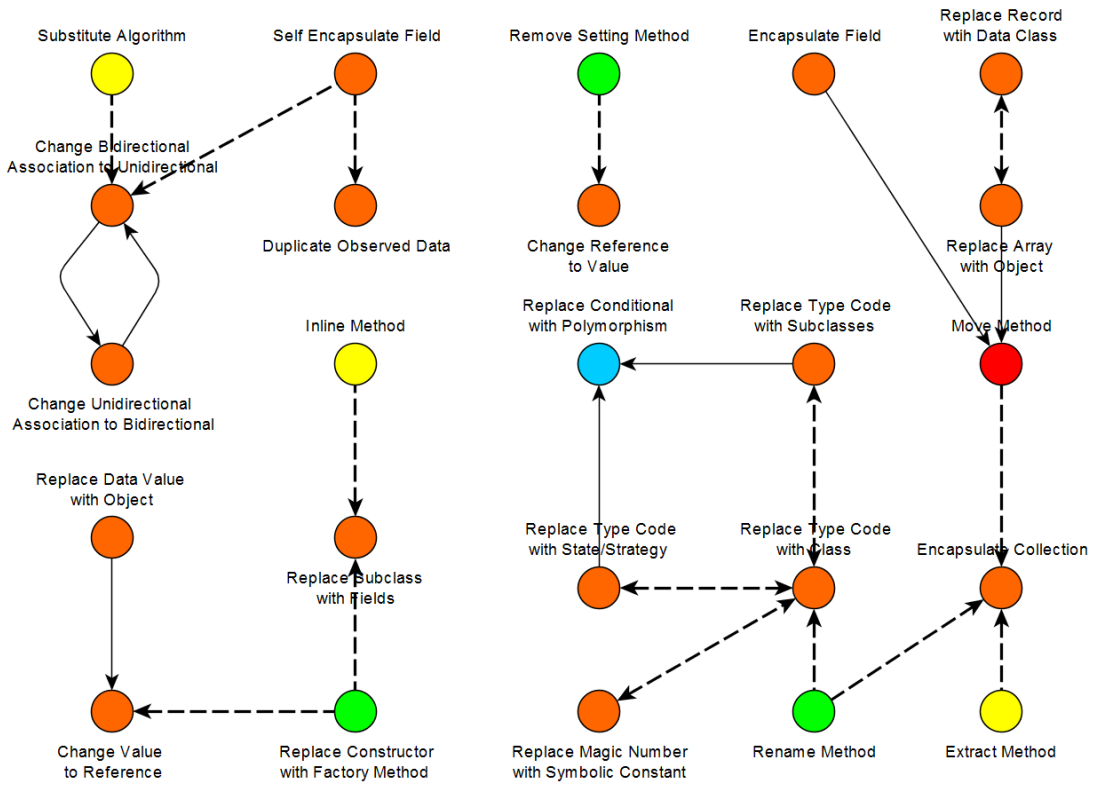


Σχήμα 3.6 Χάρτης Ανακατασκευών για "Feature Movement Between Objects"

Στον Χάρτη αυτό παρουσιάζονται οι τεχνικές που μετακινούν διάφορα στοιχεία μεταξύ των κλάσεων με σκοπό την ανάθεση ξεκάθαρων αρμοδιοτήτων σε κάθε δομή του κώδικα. Τα δύο βασικά στοιχεία των κλάσεων στο σύστημα αποτελούν τα πεδία και οι μέθοδοι αυτών και για την μετακίνηση τους σε άλλες κλάσεις χρησιμοποιούνται οι Move Field και Move Method αντίστοιχα. Πριν την μετακίνηση των πεδίων προτείνεται η ενθυλάκωση αυτών μέσω των Encapsulate Field και Self Encapsulate Field έτσι ώστε να υπάρχει ο κατάλληλος τρόπος πρόσβασης σε αυτά. Οι Extract Class και Inline Class οι οποίες εξάγουν νέες κλάσεις και ευθυγραμμίζουν κλάσεις χωρίς ιδιαίτερο λόγο ύπαρξης, αποτελούν τις βασικές τεχνικές

ανακατασκευής για την διάρθρωση απλών και σωστά δομημένων κλάσεων στο σύστημα. Για την εφαρμογή των δύο αυτών τεχνικών ανακατασκευής απαιτείται η μετακίνηση πεδίων και μεθόδων μεταξύ κλάσεων, πράγμα το οποίο αναπαρίσταται στον Χάρτη μέσω της σχέσεως "Μέρος από" με τις τεχνικές Move Field και Move Method. Για την απόκρυψη ή μη της πληροφορίας που περιλαμβάνει η εσωτερική αναπαράσταση μιας κλάσης χρησιμοποιούνται οι Hide Delegate και Remove Middle Man και σύμφωνα με τον Χάρτη γίνεται εναλλασσόμενη εφαρμογή αυτών ανάλογα τις απαιτήσεις του μοντέλου κάθε φορά.

Data Organization

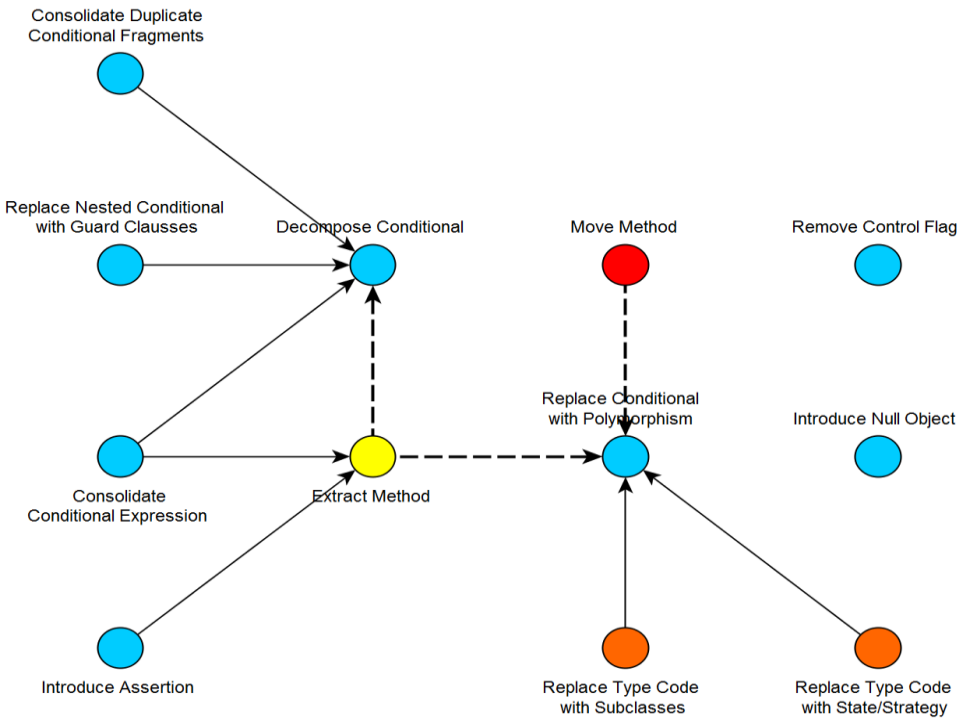


Σχήμα 3.7 Χάρτης Ανακατασκευών για "Data Organization"

Στο σχήμα 3.7 φαίνονται οι συσχετίσεις μεταξύ των τεχνικών ανακατασκευής που βελτιώνουν την διαχείριση των δεδομένων στον κώδικα. Ενδιαφέρον παρουσιάζει το γεγονός ότι στο Χάρτη υπάρχουν αρκετές συσχετίσεις με εξωτερικές τεχνικές ανακατασκευής που συνδέονται μέσω της σχέσης "Μέρος από". Αυτό συμβαίνει διότι η βελτίωση της αναπαράστασης των δεδομένων επιτυγχάνεται μέσω της εφαρμογής

βασικών ανακατασκευών σε επίπεδο μεθόδου όπως Extract Method, Rename Method, Move Method και Inline Method. Πέρα από τις εξωτερικές συσχετίσεις εξετάζοντας τον Χάρτη παρατηρούμε ότι για την διαχείριση των τύπων κώδικα στο σύστημα υπάρχουν διάφορες εναλλακτικές επιλογές όπως Replace Type Code with Class, Replace Type Code with State/Strategy και Replace Type Code with Subclasses. Το ίδιο ισχύει και για τις ανακατασκευές Replace Record with Data Class και Replace Array with Object οι οποίες αντικαθιστούν συγκεκριμένες δομές δεδομένων με κλάσεις. Ο Χάρτης επίσης προτείνει την εναλλασσόμενη μετατροπή της συσχέτισης μεταξύ δύο κλάσεων από μονόδρομη σε αμφίδρομη ανάλογα με την περίσταση, το οποίο φαίνεται από την σχέση ανάμεσα στις Change Unidirectional Association to Bidirectional και Change Bidirectional Association to Bidirectional.

Conditional Expression Simplification

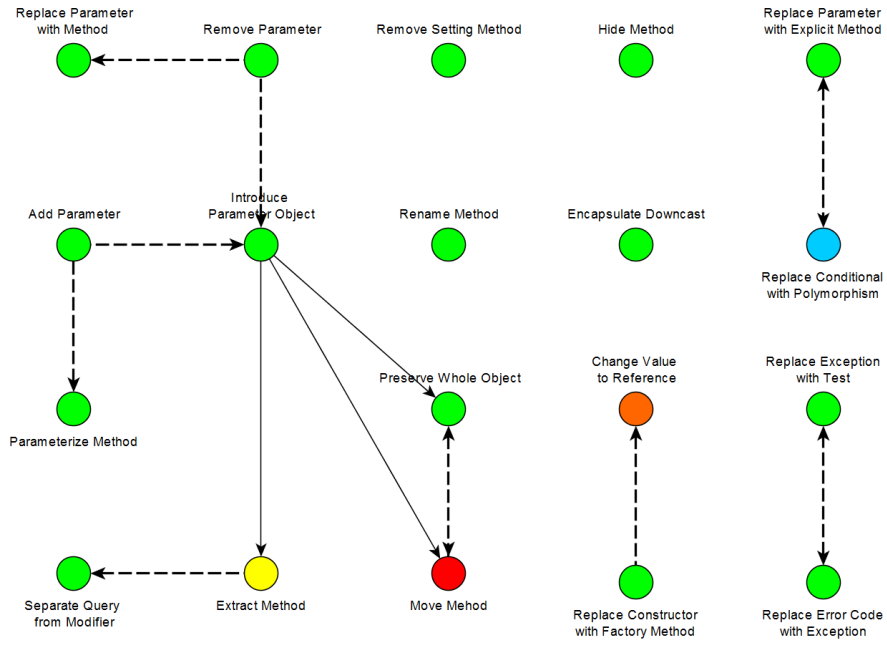


Σχήμα 3.8 Χάρτης Ανακατασκευών για “Conditional Expression Simplification”

Ο Χάρτης του Σχήματος 3.8 προκύπτει από τη συσχέτιση των τεχνικών που απλοποιούν τις κατηγορηματικές εκφράσεις στον κώδικα. Η πιο απλή τεχνική ανακατασκευής που το πετυχαίνει αυτό είναι η Decompose Conditional. Η τεχνική

αυτή εξάγει τον κώδικα της κατηγορηματικής έκφρασης σε νέες μεθόδους το οποίο φαίνεται και γραφικά από τη σχέση "Μέρος από" με την Extract Method. Καθώς η εφαρμογή της συγκεκριμένης τεχνικής είναι αρκετά γενική, πριν από αυτή προτείνεται η χρήση άλλων για ειδικότερες περιπτώσεις κατηγορημάτων όπως Consolidate Conditional Expression, Consolidate Duplicate Conditional Fragments και Replace Nested Conditional with Guard Clauses. Μια άλλη βασική τεχνική ανακατασκευής για την απλοποίηση του κατηγορήματος είναι η Replace Conditional with Polymorphism η οποία αξιοποιεί τον πολυμορφισμό του συστήματος μετακινώντας τον κώδικα σε υποκλάσεις για την εξάλειψη πολύπλοκων κατηγορηματικών εκφράσεων. Ο Χάρτης επίσης συμβουλεύει για την πιθανή εφαρμογή Replace Type Code with Subclasses ή Replace Type Code with State/Strategy πριν από τη Replace Conditional with Polymorphism καθώς μπορεί να μην υπάρχει ήδη η κατάλληλη υποδομή κληρονομικότητας στο σύστημα.

Method Call Improvement

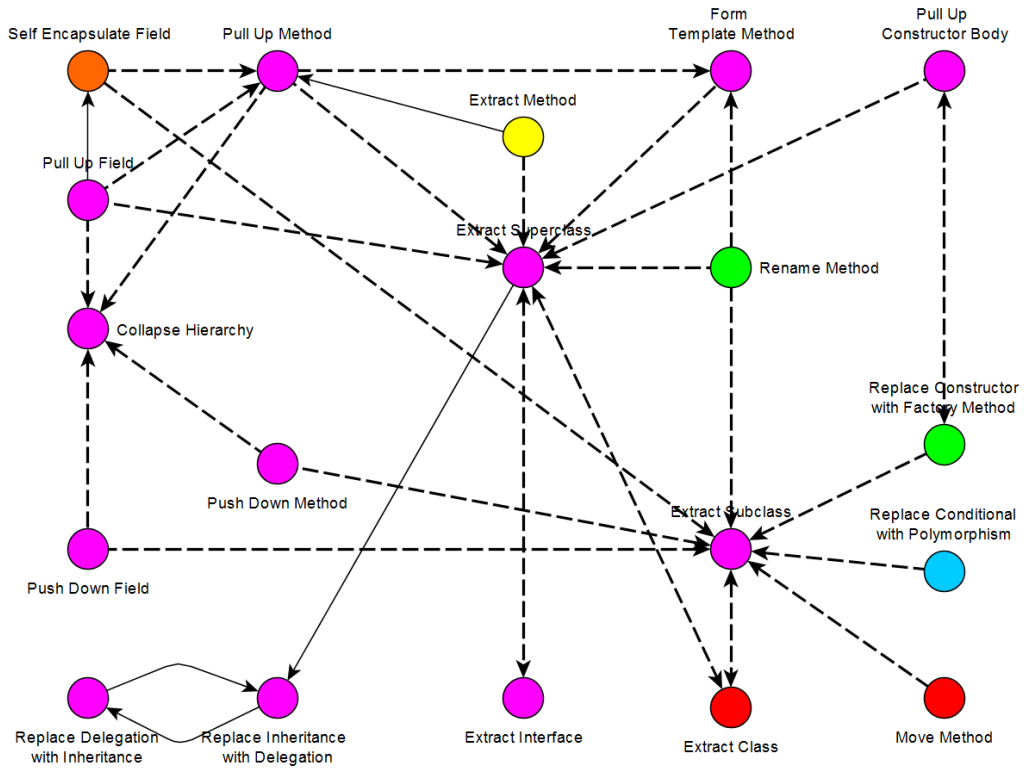


Σχήμα 3.9 Χάρτης Ανακατασκευών για "Method Call Improvement"

Οι τεχνικές ανακατασκευής που παρουσιάζονται στο Σχήμα 3.9 στοχεύουν στη βελτίωση των πρωτότυπων των μεθόδων. Σημαντικό ρόλο στη βελτίωση αυτή έχει η

σωστή παραμετροποίηση των μεθόδων η οποία πραγματοποιείται προσθέτοντας και αφαιρώντας παραμέτρους ανάλογα με την περίσταση. Οι δύο τεχνικές που το εφαρμόζουν αυτό είναι η Add Parameter και Remove Parameter. Εξετάζοντας τον Χάρτη διαπιστώνουμε πως αυτές οι δύο τεχνικές είναι αρκετά βασικές στην βελτίωση των προτύπων των μεθόδων καθώς συσχετίζονται με άλλες ανακατασκευές μέσω της σχέσεως "Μέρος από". Η τεχνική ανακατασκευής με τις περισσότερες συσχετίσεις στον Χάρτη, Introduce Parameter Object, χρησιμοποιείται για την αντικατάσταση ενός συχνά εμφανιζόμενου συνόλου παραμέτρων στο σύστημα, από ένα αντικείμενο μιας νέας κλάσης δημιουργημένης από αυτές. Οι δύο τεχνικές ανακατασκευής για την βελτίωση της διαχείρισης σφαλμάτων στον κώδικα Replace Exception with Test και Replace Error Code with Exception χρησιμοποιούνται εναλλάξ ανάλογα με την περίσταση.

Generalization Improvement



Σχήμα 3.10 Χάρτης Ανακατασκευών για "Generalization Improvement"

Τέλος στον Χάρτη του Σχήματος 3.10 εμπεριέχονται οι συσχετίσεις μεταξύ των τεχνικών ανακατασκευής που βελτιώνουν την ιεραρχία των κλάσεων. Στον Χάρτη

φαίνεται πως οι τεχνικές ανακατασκευής Pull Up Method, Push Down Method, Pull Up Field και Push Down Field, οι οποίες μετακινούν πεδία και μεθόδους μεταξύ της ιεραρχίας των κλάσεων, αποτελούν τις βασικές διαδικασίες καθώς είναι κομμάτι αρκετών άλλων τεχνικών της ίδιας κατηγορίας. Αντίθετα, τεχνικές όπως οι Extract Superclass, Extract Subclass και Collapse Hierarchy παρουσιάζουν αυξημένη πολυπλοκότητα καθώς η υλοποίηση τους απαιτεί την εφαρμογή ενός συνόλου άλλων τεχνικών. Η Extract Class προτείνεται ως εναλλακτική επιλογή των Extract Superclass και Extract Subclass στην περίπτωση που ο προγραμματιστής δεν επιθυμεί η νέα κλάση να περιορίζεται από τους κανόνες του πολυμορφισμού. Ο Χάρτης επίσης προτείνει την εναλλασσόμενη εφαρμογή μεταξύ Replace Delegation with Inheritance και Replace Inheritance with Delegation ανάλογα με τον τύπο συσχέτισης μεταξύ τον κλάσεων που απαιτείται κάθε φορά στον κώδικα.

Παρατήρηση: Κάποιες τεχνικές ανακατασκευής που παρουσιάζονται σε κάποια κατηγορία ως εξωτερικές, συμμετέχουν σε συσχετίσεις οι οποίες δεν εμφανίζονται στον Χάρτη της κατηγορίας που αυτές ανήκουν (δηλαδή της κατηγορίας που αυτές είναι εσωτερικές). Αυτό συμβαίνει γιατί στον Χάρτη κάθε κατηγορίας παρουσιάζονται μόνο οι συσχετίσεις από τη σκοπιά των βελτιώσεων που αυτή η κατηγορία σκοπεύει να επιφέρει στον κώδικα. Για παράδειγμα οι συσχετίσεις που συμμετέχει ως εξωτερική τεχνική το Extract Method στον Χάρτη του Σχήματος 3.8 δεν υπάρχουν στον Χάρτη του Σχήματος 3.5 καθώς δεν συνάδουν νοηματικά με τις βελτιώσεις που παρέχει η κατηγορία του Method Composition. Ο πλήρης Χάρτης με όλες τις συσχετίσεις για όλες τις τεχνικές ανακατασκευής παρουσιάζεται στην επόμενη υποενότητα.

3.2.2 Συνολικός Χάρτης Ανακατασκευών

Ακολουθώντας την τμηματική παρουσίαση του Χάρτη Ανακατασκευών ανά κατηγορία τεχνικών ανακατασκευής, σε αυτή την υποενότητα θα γίνει παρουσίαση του συνολικού Χάρτη Ανακατασκευών. Ο Χάρτης αυτός αποτελεί την ένωση των έξι Χαρτών που περιγράφηκαν παραπάνω και αναπαριστά όλες τις συσχετίσεις μεταξύ των 68 τεχνικών ανακατασκευής. Ο χρωματισμός τον κόμβων ακολουθεί τη σύμβαση που έχει οριστεί στην προηγούμενη υποενότητα έτσι ώστε να είναι και εδώ ορατό σε

ποία κατηγορία ανήκει κάθε τεχνική ανακατασκευής. Ο συνολικός Χάρτης Ανακατασκευών παρουσιάζεται στο Σχήμα 3.11.

Αναφορικά με την διαδικασία ανακατασκευής λογισμικού, είναι προτιμότερο ο κατασκευαστής να συμβουλευτεί τους έξι τμηματικούς Χάρτες ανά κατηγορία καθώς αυτοί είναι λιγότερο πολύπλοκοι και επικεντρώνονται στην βελτίωση συγκεκριμένων χαρακτηριστικών ο καθένας. Παρ' όλα αυτά, μελετώντας κανείς τον συνολικό Χάρτη Ανακατασκευών μπορεί να εξάγει κάποιες πληροφορίες πού αφορούν συγκεκριμένες τεχνικές ανακατασκευής. Αρχικά κοιτώντας τον Χάρτη μπορεί κανείς να δει πως πέρα από 10 συγκεκριμένους κόμβους, υπάρχει μονοπάτι σύνδεσης από τον ένα κόμβο στον άλλο καθώς ανήκουν στην ίδια συνεκτική συνιστώσα. Αυτό αποδεικνύει το γεγονός ότι η ανακατασκευή λογισμικού αποτελεί μια σύνθετη διαδικασία εφαρμογής ενός συνόλου τεχνικών και όχι την εκτέλεση κάποιων από αυτών μεμονωμένα. Εστιάζοντας στο επίπεδο των κόμβων, κοιτώντας τον αριθμό τον εισερχόμενων και εξερχόμενων ακμών μπορεί να δει κανείς την σημαντικότητα κάποιων τεχνικών στην διαδικασία ανακατασκευής. Στον Πίνακα 3.5 παρουσιάζεται ο αριθμός των ακμών ανά είδος σχέσης για τις τέσσερις τεχνικές ανακατασκευής με τον μεγαλύτερο βαθμό στον Χάρτη. Για τις δύο πρώτες τεχνικές στον πίνακα παρατηρούμε έναν σημαντικό αριθμό από εξερχόμενες ακμές της σχέσεως "Μέρος από". Από αυτό συμπεραίνουμε πως αυτές οι τεχνικές αποτελούν τις βασικότερες στην διαδικασία ανακατασκευής καθώς η εφαρμογή αρκετών άλλων τεχνικών εμπεριέχει την εφαρμογή αυτών των δύο. Επίσης για αυτές τις δύο τεχνικές διακρίνουμε αυξημένο αριθμό εισερχομένων ακμών σχέσεων διαδοχής. Αυτό συμβαίνει διότι έπειτα από την εφαρμογή κάποιας τεχνικής ανακατασκευής, οι αλλαγές που αυτή επιφέρει στον κώδικα, συχνά ανοίγουν ευκαιρίες για την εφαρμογή των δύο αυτών βασικών τεχνικών. Συμπεραίνοντας λοιπόν μπορούμε να πούμε ότι είναι αρκετά πιθανό σε κάποια διαδικασία ανακατασκευής κώδικα να χρησιμοποιηθεί Extract Method ή Move Method. Τέλος εξετάζοντας τις δύο τελευταίες τεχνικές του πίνακα παρατηρούμε ότι η πλειοψηφία των ακμών τους ανήκουν στη σχέση "Μέρος από" και είναι εισερχόμενες σε αυτές. Από αυτό το γεγονός καταλήγουμε ότι οι Extract Superclass και Extract Subclass αποτελούν τις πιο πολύπλοκες στην διαδικασία ανακατασκευής καθώς η εφαρμογή τους επιβάλει την εφαρμογή ενός

συνόλου άλλων ανακατασκευών. Ο συνολικός πίνακας με το πλήθος των ακμών για όλες τις τεχνικές ανακατασκευής παρουσιάζεται στον Πίνακα 3.6 για πληρότητα.

Πίνακας 3.5 Πλήθος Ακμών Extract Method, Move Method, Extract Superclass και Extract Subclass

Τεχνική Ανακατασκευής	**Βαθμός**	**Σχέση "Αντί για"**	**Σχέση "Μέρος από"**		**Σχέση Διαδοχής**	
			Εισερχ.	Εξερχ.	Εισερχ.	Εξερχ.
Extract Method	18	2	0	5	7	4
Move Method	10	1	0	5	4	0
Extract Superclass	9	2	6	0	0	1
Extract Subclass	8	1	7	0	0	0

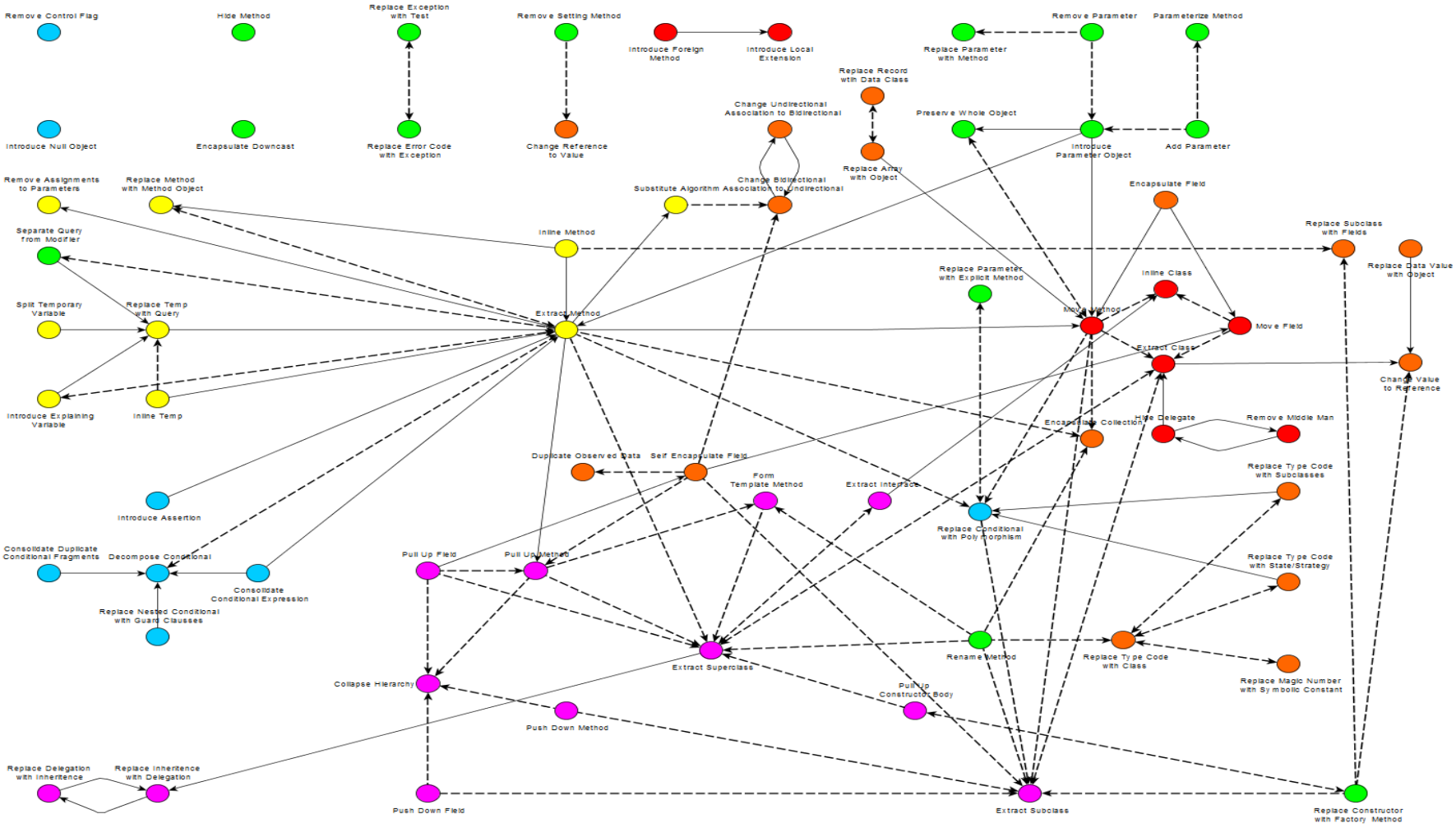
Πίνακας 3.6 Πλήθος Ακμών για κάθε Τεχνική Ανακατασκευής

Τεχνική Ανακατασκευής	**Βαθμός**	**Σχέση "Αντί για"**	**Σχέση "Μέρος από"**		**Σχέση Διαδοχής**	
			Εισερχ.	Εξερχ.	Εισερχ.	Εξερχ.
Method Composition						
Extract Method	18	2	0	5	7	4
Replace Temp with Query	5	0	1	0	3	1
Split Temporary Variable	1	0	0	0	0	1
Inline Temp	2	0	0	1	0	1
Replace Method with Method Object	2	1	0	0	1	0
Inline Method	3	0	0	1	0	2
Remove Assignments to Parameters	1	0	0	0	0	1
Introduce Explaining Variable	2	1	0	0	0	1
Substitute Algorithm	2	0	0	1	1	0
Feature Movement Between Objects						
Move Method	10	1	0	5	4	0
Move Field	4	0	0	2	2	0

Τεχνική Ανακατασκευής	Βαθμός	Σχέση "Αντί για"	Σχέση "Μέρος από"		Σχέση Διαδοχής	
			Εισερχ.	Εξερχ.	Εισερχ.	Εξερχ.
Extract Class	6	2	2	0	1	1
Inline Class	3	0	2	0	1	0
Hide Delegate	3	0	0	0	1	2
Remove Middle Man	2	0	0	0	1	1
Introduce Foreign Method	1	0	0	0	0	1
Introduce Local Extension	1	0	0	0	1	0
Conditional Expression Simplification						
Remove Control Flag	0	0	0	0	0	0
Introduce Null Object	0	0	0	0	0	0
Introduce Assertion	1	0	0	0	0	1
Decompose Conditional	4	0	1	0	3	0
Consolidate Conditional Expression	2	0	0	0	0	2
Replace Nested Conditional with Guard Clausses	1	0	0	0	0	1
Consolidate Duplicate Conditional Fragments	1	0	0	0	0	1
Replace Conditional with Polymorphism	6	1	2	1	2	0
Data Organization						
Change Reference to Value	1	0	1	0	0	0
Change Value to Reference	3	0	1	0	2	0
Replace Record with Data Class	1	1	0	0	0	0
Replace Array with Object	2	1	0	0	0	1
Change Unidirectional Association to Bidirectional	2	0	0	0	1	1

Τεχνική Ανακατασκευής	Βαθμός	Σχέση "Αντί για"	Σχέση "Μέρος από"		Σχέση Διαδοχής	
			Εισερχ.	Εξερχ.	Εισερχ.	Εξερχ.
Change Bidirectional Association to Unidirectional	4	0	2	0	1	1
Duplicate Observed Data	1	0	1	0	0	0
Self Encapsulate Field	6	0	0	4	1	1
Encapsulate Field	2	0	0	0	0	2
Encapsulate Collection	3	0	3	0	0	0
Replace Data Value With Object	1	0	0	0	0	1
Replace Subclass with Fields	2	0	2	0	0	0
Replace Type Code with Subclasses	2	1	0	0	0	1
Replace Type Code with State/Strategy	2	1	0	0	0	1
Replace Type Code with Class	4	3	1	0	0	0
Replace Magic Number with Symbolic Constant	1	1	0	0	0	0
Generalization Improvement						
Replace Delegation with Inheritance	2	0	0	0	1	1
Replace Inheritance with Delegation	3	0	0	0	2	1
Push Down Field	2	0	0	2	0	0
Push Down Method	2	0	0	2	0	0
Pull Up Field	4	0	0	3	0	1
Pull Up Method	6	0	2	3	1	0
Collapse Hierarchy	4	0	4	0	0	0
Extract Superclass	9	2	6	0	0	1
Extract Subclass	8	1	7	0	0	0
Pull Up Constructor Body	2	1	0	1	0	0

Τεχνική Ανακατασκευής	Βαθμός	Σχέση "Αντί για"	Σχέση "Μέρος από"		Σχέση Διαδοχής	
			Εισερχ.	Εξερχ.	Εισερχ.	Εξερχ.
Form Template Method	3	0	2	1	0	0
Extract Interface	2	1	0	0	0	1
Method Call Improvement						
Separate Query from Modifier	2	0	1	0	0	1
Hide Method	0	0	0	0	0	0
Encapsulate Downcast	0	0	0	0	0	0
Replace Exception with Test	1	1	0	0	0	0
Replace Error Code with Exception	1	1	0	0	0	0
Remove Setting Method	1	0	0	1	0	0
Rename Method	5	0	0	5	0	0
Replace Constructor with Factory Method	4	1	0	3	0	0
Replace Parameter with Explicit Method	1	1	0	0	0	0
Replace Parameter with Method	1	0	1	0	0	0
Remove Parameter	2	0	0	2	0	0
Add Parameter	2	0	0	2	0	0
Parameterize Method	1	0	1	0	0	0
Preserve Whole Object	2	1	0	0	1	0
Introduce Parameter Object	5	0	2	0	0	3



Σχήμα 3.11 Συνολικός Χάρτης Ανακατασκευών

ΚΕΦΑΛΑΙΟ 4. ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΠΑΡΟΥΣΙΑΣΗ ΤΟΥ REFACTORING TRIP ADVISOR

4.1 Μοντέλο Δεδομένων

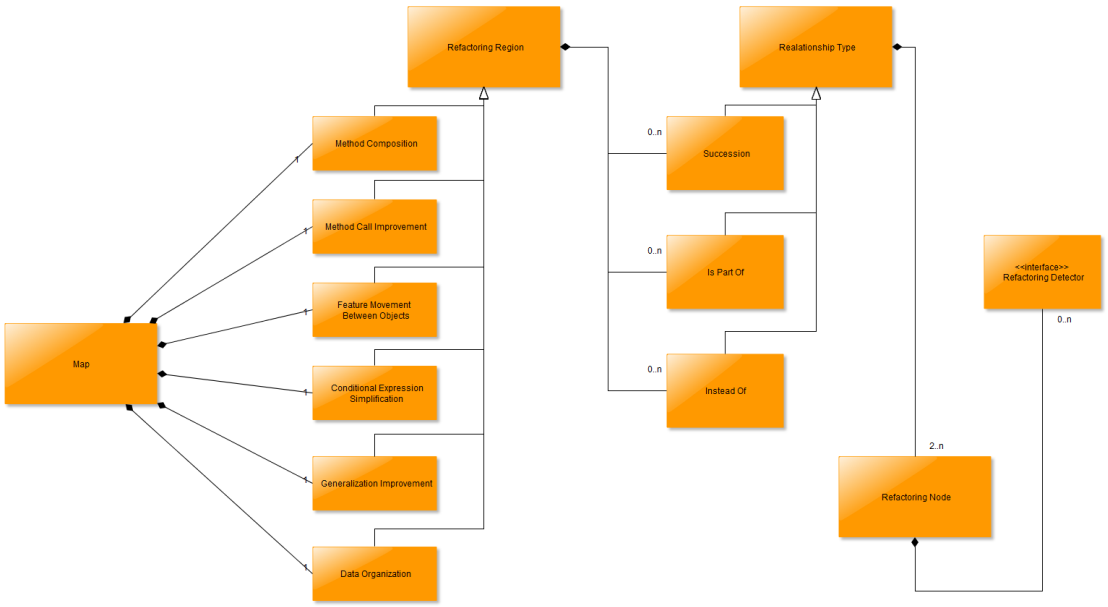
Στα προηγούμενα κεφάλαια έγινε λόγος για το θεωρητικό υπόβαθρο αναφορικά με την ανακατασκευή κώδικα και παρουσιάστηκε ο Χάρτης Ανακατασκευών χρησιμοποιώντας κατάλληλους γραφήματα για την απεικόνιση του. Βασική συνεισφορά της διατριβής αποτελεί ο Χάρτης Ανακατασκευών. Παράλληλα όμως, στα πλαίσια αυτής της διατριβής αναπτύσσεται ένα εργαλείο το οποίο έχει στόχο να βοηθήσει τον κατασκευαστή στη διαδικασία ανακατασκευής, αξιοποιώντας το Χάρτη Ανακατασκευών.

Η ανάπτυξη του εργαλείου υποβοήθησης γίνεται ως επέκταση της γνωστής πλατφόρμας ανάπτυξης λογισμικού Eclipse. Ως γλώσσα ανάπτυξης χρησιμοποιείται η Java και γίνεται αξιοποίηση των εργαλείων ανάπτυξης Java Development Tools (JDT) που παρέχει το Eclipse. Για την οπτικοποίηση των γράφων που αντιστοιχούν στους έξι Χαρτες συσχετίσεων που περιγράφηκαν στην υποενότητα 3.2.1 χρησιμοποιείται η βιβλιοθήκη οπτικοποίησης γράφων JGraphX [12].

Παράλληλα με τον Χάρτη Ανακατασκευών το εργαλείο υποβοήθησης προσφέρει τη δυνατότητα στο χρήστη να εντοπίσει σημεία στον κώδικα του που προσφέρουν ευκαιρίες για συγκεκριμένες τεχνικές ανακατασκευής. Αυτές οι δυνατότητες

αναλύονται περισσότερο στην επόμενη ενότητα. Το γεγονός ότι η υλοποίηση στην πράξη αυτής της δυνατότητας εντοπισμού μπορεί σε μερικές περιπτώσεις να είναι περίπλοκη, σε συνδυασμό με το πλήθος των διαφορετικών τεχνικών ανακατασκευής οδήγησε αυτή τη διατριβή στην υλοποίηση της μόνο για μερικές τεχνικές.

Με βάση το παραπάνω το σχέδιο ανάπτυξης της εφαρμογής θα πρέπει να γίνει με τέτοιο τρόπο ώστε να διευκολύνει την μελλοντική κάλυψη όλων των τεχνικών με τη δυνατότητα εντοπισμού. Επίσης στο σχεδιασμό του συστήματος θα πρέπει να ληφθεί υπόψη η δυνατότητα εφαρμογής πιθανών αλλαγών στον Χάρτη Ανακατασκευών. Αυτές οι δύο ιδιότητες υλοποιούνται στο σχέδιο του συστήματος το οποίο φαίνεται στο Σχήμα 4.1.



Σχήμα 4.1 Διάγραμμα Κλάσεων του Μοντέλου Δεδομένων

Όπως φαίνεται και από το διάγραμμα κλάσεων του Σχήματος 4.1 το εργαλείο υλοποιεί τους έξι Χάρτες συσχετίσεων ανά κατηγορία και όχι τον συνολικό Χάρτη καθώς η τμηματική αυτή αναπαράσταση είναι περισσότερο βοηθητική για τον κατασκευαστή. Κάθε Χάρτης υλοποιείται από ένα σύνολο κόμβων κάθε ένας εκ των οποίων αντιστοιχεί σε μια τεχνική ανακατασκευής και από ένα σύνολο τριών διαφορετικών ακμών ανάμεσα σε αυτούς που αντιστοιχούν στα τρία είδη

συσχετίσεων. Αλλαγές στους Χάρτες όπως διαγραφή η εισαγωγή τεχνικών και συσχετίσεων μπορεί εύκολα να γίνει επηρεάζοντας μόνο τις κλάσεις αναπαράστασης των συνόλων των κόμβων και των ακμών (Succession, Instead of, Part of και Refactoring Node στο διάγραμμα κλάσεων). Αναφορικά με τη δυνατότητα εντοπισμού ευκαιριών για κάθε τεχνική ανακατασκευής παρέχεται μια διεπαφή (<<interface>> Refactoring Detector στο διάγραμμα κλάσεων) η οποία υλοποιείται για ένα σύνολο από τις τεχνικές αυτές. Η διεπαφή αυτή επιτρέπει στον κατασκευαστή την μελλοντική κάλυψη περισσότερων τεχνικών με την δυνατότητα του εντοπισμού και δίνει επίσης την δυνατότητα της παροχής παραπάνω από μίας υλοποίησης του εντοπισμού για κάθε τεχνική.

4.2 Εντοπισμός Δυνατοτήτων Ανακατασκευής Κώδικα

Ένα σημαντικό κομμάτι στην διαδικασία ανακατασκευής του λογισμικού αποτελεί η εύρεση σημείων στον κώδικα τα οποία υποδεικνύουν ένα βαθύτερο πρόβλημα στο σύστημα, γνωστά και ως Code Smells [7]. Στα σημεία αυτά μπορεί κανείς να βρει κομμάτια κώδικα υποψήφια προς ανακατασκευή. Για την διευκόλυνση αυτής της διαδικασίας στο εργαλείο υποβοήθησης παρέχεται ένας αυτοματοποιημένος εντοπισμός για δυνατότητα εφαρμογής κάποιων από τις τεχνικές ανακατασκευής που περιγράφηκαν στην ενότητα 2.5. Για την υλοποίηση αυτής της δυνατότητας γίνεται συντακτική ανάλυση του κώδικα του χρήστη, χρησιμοποιώντας τον συντακτικό αναλυτή που παρέχει το Eclipse JDT. Η μέθοδος με την οποία γίνεται αυτό παρουσιάζεται στον ψευδοκώδικα του Σχήματος 4.2. Ο αλγόριθμος της μεθόδου *identificationForSpecificRefactoring*(Method *method)* διαφέρει για τον εντοπισμό κάθε διαφορετικής τεχνικής ανακατασκευής. Στη συνέχεια περιγράφεται πως υλοποιείται αυτή η μέθοδος για κάθε μία από τις τεχνικές που ο Refactoring Trip Advisor παρέχει τη δυνατότητα εντοπισμού.

identifyRefactoringOpportunities(EclipseProject *p*)

for each JavaFile *file* in *p*

t = getASTTree(*file*)

for each Class *class* in *t*

for each Method *method* in *class*

suggestedEntity = *identificationForSpecificRefactoring*(Method *method)*

return(*suggestedEntity*)

Σχήμα 4.2 Αλγόριθμος Εντοπισμού Δυνατοτήτων Ανακατασκευής Κώδικα

Extract Method

Η υλοποίηση της δυνατότητας αυτού του εντοπισμού βασίζεται στη [13], όπου προτεινόμενος κώδικας προς εξαγωγή σε νέα μέθοδο είναι τμήματα κώδικα (slices) που αφορούν τον υπολογισμό μιας μεταβλητής και τμήματα (slices) που επηρεάζουν την κατάσταση ενός συγκεκριμένου αντικειμένου. Ένα παράδειγμα αυτών των τμημάτων κώδικα προς εξαγωγή φαίνεται στο Σχήμα 4.3.

```
public void draw(Graphics2D g2){
    super.draw(g2);
    Color oldColor = g2.getColor();
    g2.setColor(color);

    Shape path = getShape();
    g2.fill(path);
    g2.setColor(oldColor);
    g2.draw(path);

    Rectangle2D bounds = getBounds();
    GeneralPath fold = new GeneralPath();
    fold.moveTo((float)(bounds.getMaxX()
        - FOLD_X), (float)bounds.getY());
    fold.lineTo((float)(bounds.getMaxX()
        - FOLD_X), (float)(bounds.getY() + FOLD_X));
    fold.lineTo((float)bounds.getMaxX(),
        (float)(bounds.getY() + FOLD_Y));
    fold.closePath();

    oldColor = g2.getColor();
    g2.setColor(g2.getBackground());
    g2.fill(fold);
    g2.setColor(oldColor);
    g2.draw(fold);
    text.draw(g2,getBounds());
}
```

A

```
public void translate (double dx,double dy)
{
    if(getParent() == null){
        dy = TOP_GAPY - getBounds().getY();
    }
    else{
        double y = getBounds.getY() + dy;
        y = Math.max(y,
        getParent().getBounds().getMinY()
        - topHeight / 2);
        y = Math.min(y,
        getParent().getBounds().getMaxY()
        - topHeight / 2);
        dy = y - getBounds().getY();
    }
    super.translate(dx,dy);
}
```

B

Σχήμα 4.3 (A) Κώδικας που επηρεάζει την κατάσταση του αντικειμένου fold και (B) Κώδικας για τον υπολογισμό της μεταβλητής dy.

Η διαδικασία εξαγωγής των προτεινόμενων τμημάτων που αφορούν τον υπολογισμό μιας μεταβλητής έχει ως εξής:

- Υπολογισμός του συνόλου V των τοπικών μεταβλητών που δηλώνονται μέσα σε μια μέθοδο *m*.
- Για κάθε μεταβλητή $v \in V$ γίνεται υπολογισμός του συνόλου C των statements που περιέχουν μια ανάθεση της μεταβλητής *v*. Αυτά τα statements μαζί με την μεταβλητή *v* δημιουργούν το σύνολο των slicing criteria (*c*,*v*) όπου $c \in C$.
- Για κάθε statement $c \in C$ γίνεται υπολογισμός του συνόλου των οριακών μπλοκ *Blocks(c).*
- Υπολογισμός των κοινών οριακών μπλοκ για κάθε statement του συνόλου *C*: $Blocks(C) = \bigcap_{c \in C} Blocks(c)$.
- Για κάθε slicing criterion (*c*,*v*) όπου $c \in C$ και οριακό μπλοκ $B_n \in Blocks(C)$, γίνεται υπολογισμός του block-based slice $S_B(c, v, B_n)$. Το block-based slice είναι το σύνολο των statement που επηρεάζουν τον υπολογισμό της μεταβλητής *v* στο statement *c*.
- Για κάθε $B_n \in Blocks(C)$ η ένωση των slices $US_B(C, v, B_n) = \bigcup_{c \in C} S_B(c, v, B_n)$, αποτελεί ένα slice το οποίο καλύπτει τον πλήρη υπολογισμό της μεταβλητής *v* στην περιοχή B_n.

Η διαδικασία εξαγωγής των προτεινόμενων τμημάτων που επηρεάζουν την κατάσταση ενός αντικειμένου έχει ως εξής:

- Υπολογισμός του συνόλου R των αναφορών σε αντικείμενα που υπάρχουν μέσα μια μέθοδο *m*. Οι αναφορές αυτές μπορεί αν είναι: τοπικές μεταβλητές της *m*, παράμετροι της *m* ή πεδία της κλάσης που η *m* ανήκει και είναι τύπου non-primitive.
- Για κάθε αναφορά αντικειμένου $r \in R$, γίνεται υπολογισμός του συνόλου των πεδίων F_r τα οποία τροποποιούνται από την αναφορά *r* μέσω κλήσεις μεθόδων (ή μέσω άμεσης τροποποίησης πεδίου).
- Για κάθε πεδίο $f \in F_r$ γίνεται υπολογισμός του συνόλου των statement C_f μέσα στην *m*, που περιέχουν το f στο σύνολο των ορισμένων μεταβλητών

τους. Αυτά τα statements μαζί με την μεταβλητή *f* δημιουργούν το σύνολο των slicing criteria (*c*,*f*) όπου $c \in C_f$.

- Για κάθε statement $c \in C_f$. γίνεται υπολογισμός του συνόλου των οριακών μπλοκ *Blocks(c).*
- Υπολογισμός των κοινών οριακών μπλοκ για κάθε statement του συνόλου C_f: $Blocks(C_f) = \bigcap_{c \in C_f} Blocks(c)$.
- Υπολογισμός των κοινών οριακών μπλοκ για όλα τα μπλοκ $Blocks(C_f), \forall f \in F_r : Blocks(r) = \bigcap_{f \in F_r} Blocks(C_f)$.
- Για κάθε slicing criterion (*c*,*f*) όπου $c \in C_f$, $f \in F_r$ και οριακό μπλοκ $B_n \in Blocks(r)$ γίνεται υπολογισμός του block-based slice $S_B(c, f, B_n)$.
- Για κάθε $B_n \in Blocks(r)$ η ένωση των slices για το πεδίο f είναι: $US_B(C_f, f, B_n) = \bigcup_{c \in C_f} S_B(c, f, B_n)$.
- Για κάθε $B_n \in Blocks(r)$ η ένωση των slices της αναφοράς r είναι: $US_B(r, B_n) = \bigcup_{f \in F_r} S_B(C_f, f, B_n)$. Αυτή αποτελεί ένα slice που περιέχει όλα τα statements στη μέθοδο *m*, που επηρεάζουν την κατάσταση της αναφοράς αντικειμένου *r*.
-

Move Method

Ο υπολογισμός των προτεινόμενων μεθόδων προς μετακίνηση βασίζεται στη μέθοδο που προτείνεται στη [14]. Σε αυτή ορίζεται ένα σύνολο οντοτήτων για κάθε πεδίο, μέθοδο και κλάση στο σύστημα ως εξής:

Το σύνολο οντοτήτων ενός πεδίου περιέχει τις εξής οντότητες:

- Τις μεθόδους που έχουν άμεση πρόσβαση σε αυτό το πεδίο και ανήκουν στην ίδια κλάση με αυτό.
- Τις μεθόδους που έχουν πρόσβαση σε αυτό και ανήκουν σε άλλες κλάσεις του συστήματος.

Το σύνολο οντοτήτων μιας μεθόδου περιέχει τις εξής οντότητες:

- Τα άμεσα προσβάσιμα πεδία που ανήκουν στην ίδια κλάση με την μέθοδο.
- Τα προσβάσιμα πεδία που ανήκουν σε άλλες κλάσεις του συστήματος.

- Τις άμεσα προσβάσιμες μεθόδους που ανήκουν στην ίδια κλάση με την μέθοδο αυτή.
- Τις προσβάσιμες μεθόδους που ανήκουν σε άλλες κλάσεις του συστήματος.

Το σύνολο οντοτήτων μιας κλάσης περιέχει τις εξής οντότητες:

- Όλα τα πεδία που ανήκουν στην κλάση αυτή.
- Όλες τις μεθόδους που ανήκουν στην κλάση αυτή.

Στη [14] γίνεται υπολογισμός της απόστασης Jaccard η οποία εκφράζει την διαφοροποίηση μεταξύ δύο συνόλων. Εάν *e* είναι μια οντότητα του συστήματος, *C* μια κλάση του συστήματος και S_x το σύνολο της οντότητας ή κλάσης, τότε η απόσταση μιας οντότητας *e* και μίας κλάσης *C* ορίζεται ως εξής:

1ος Ορισμός: Εάν η οντότητα *e* δεν ανήκει στην κλάση *C*, τότε η απόσταση είναι η απόσταση Jaccard των συνόλων οντοτήτων τους.

$$distance(e, C) = 1 - \frac{|S_e \cap S_C|}{|S_e \cup S_C|}, where\ S_C = \bigcup_{e_i \in C} \{e_i\}.$$

2ος Ορισμός: Εάν η οντότητα *e* ανήκει στην κλάση *C*, τότε η οντότητα *e* δεν περιέχεται στο σύνολο οντοτήτων της κλάσης *C*.

$$distance(e, C) = 1 - \frac{|S_e \cap S'_C|}{|S_e \cup S'_C|}, where\ S'_C = S_C \backslash \{e\}.$$

Αξιοποιώντας τους παραπάνω ορισμούς, ο υπολογισμός της υποψήφιας κλάσης στην οποία θα πρέπει να μετακινηθεί μια μέθοδος *m* γίνεται ως εξής:

- Αναζήτηση στο σύνολο οντοτήτων της *m* για τον προσδιορισμό του συνόλου των υποψήφιων κλάσεων *T*.
- Φθίνουσα ταξινόμηση σε πρώτο επίπεδο του συνόλου *T* με βάση τον αριθμό των οντοτήτων των υποψήφιων κλάσεων στις οποίες η μέθοδος *m* έχει πρόσβαση. Αύξουσα ταξινόμηση σε δεύτερο επίπεδο με βάση την απόσταση της μεθόδου *m* από κάθε υποψήφια κλάση.

- Έλεγχος εάν η μέθοδος *m* τροποποιεί δομές δεδομένων σε κάποια από τις υποψήφιες κλάσεις.
- Πρόταση μετακίνησης της μεθόδου *m* στην πρώτη υποψήφια κλάση ακολουθώντας την ταξινόμηση του συνόλου *T*.

Extract Class

Η υλοποίηση του εντοπισμού των υποψήφιων κλάσεων προς εξαγωγή στο σύστημα βασίζεται πάνω στη [15]. Για την εύρεση των υποψήφιων εξαγόμενων κλάσεων στο σύστημα εφαρμόζεται ο agglomerative αλγόριθμος της ιεραρχικής ομαδοποίησης στον οποίο οι βασικές οντότητες που χρησιμοποιούνται για την δημιουργία των ομάδων είναι οι μέθοδοι και τα πεδία. Για την ομαδοποίηση των δύο αυτών οντοτήτων ο αλγόριθμος χρησιμοποιεί τα σύνολα οντοτήτων των μεθόδων και των πεδίων καθώς και την απόσταση Jaccard μεταξύ αυτών με βάση τον 1^{ο} Ορισμό που δόθηκε προηγουμένως. Συγκεκριμένα η διαδικασία για την εύρεση του υποψήφιου συνόλου πεδίων-μεθόδων για εξαγωγή σε νέα κλάση έχει ως εξής:

- Εφαρμογή του ιεραρχικού αλγόριθμου ομαδοποίησης για την εύρεση των ομάδων. Οι ομάδες αυτές αποτελούνται από πεδία και μεθόδους.
- Επιλογή των ομάδων πριν το τελευταίο σημείο συγχώνευσης. Η επιλογή αυτή γίνεται γιατί αυτές οι ομάδες έχουν την μεγαλύτερη απόσταση συγχώνευσης μεταξύ τους στο δενδροδιάγραμμα. Αυτό σημαίνει ότι τα πεδία και οι μέθοδοι σε αυτές τις ομάδες έχουν πρόσβαση σε λιγότερα ή και κανένα κοινό στοιχείο μεταξύ τους. Αυτές οι ομάδες αποκαλούνται general concepts.
- Για κάθε general concept διερευνάται το αντίστοιχο υποδένδρο για πιθανές ομάδες που μπορεί να προταθούν επίσης προς εξαγωγή. Αυτές οι ομάδες ονομάζονται subconcepts και η διαδιασία επιλογής τους έχει ως εξής:
  - Εάν τουλάχιστον μια από τις δύο ομάδες παιδιά του υποδένδρου αποτελείται μόνο από ένα στοιχείο κλάσης (πεδίο ή μέθοδο), τότε απορρίπτονται και οι δύο ομάδες.
  - Εάν και οι δύο ομάδες αποτελούνται από δύο ή περισσότερα στοιχεία κλάσης (πεδία ή μεθόδους), τότε επιλέγονται και οι δύο ως subconcepts.

- Ως προτεινόμενα σύνολα πεδίων-μεθόδων προς εξαγωγή σε νέα κλάση δίνονται τα general concepts και τα subconcepts της παραπάνω διαδικασίας.

Inline Method

Σκοπός αυτής της τεχνικής ανακατασκευής όπως αναφέρθηκε στην ενότητα 2.5 είναι η ευθυγράμμιση μικρών σε μέγεθος μεθόδων που δεν έχουν κάποιο ιδιαίτερο λόγο ύπαρξης ο κώδικας τους είναι ξεκάθαρος όσο το όνομα τους. Η δυνατότητα εντοπισμού για αυτήν την τεχνική ανακατασκευής προτείνει υποψήφιες μεθόδους που πληρούν τις εξής συνθήκες:

- Ο κώδικας της μεθόδου θα πρέπει να αποτελείται από ένα statement μόνο.
- Η μέθοδος θα πρέπει να είναι δηλωμένη ως private.
- Το πρόθεμα του ονόματος της μεθόδου δεν θα πρέπει να είναι “get”, “set”, “add” ή “remove”.

Replace Temp with Query

Αυτή η τεχνική ανακατασκευής εφαρμόζεται όταν υπάρχουν προσωρινές μεταβλητές στον κώδικα στις οποίες αποθηκεύεται το αποτέλεσμα μιας έκφρασης. Η έκφραση αυτή εξάγεται σε μια νέα μέθοδο και γίνεται αντικατάσταση της προσωρινής μεταβλητής με κλήση της εξαγόμενης μεθόδου. Η δυνατότητα εντοπισμού για αυτήν την τεχνική ανακατασκευής προτείνει υποψήφιες μεταβλητές που πληρούν τις εξής συνθήκες:

- Η μεταβλητή συμμετέχει μόνο σε μια σχέση ανάθεσης.
- Η ανάθεση αυτή της μεταβλητής είναι μια σύνθετη έκφραση η οποία περιέχει τουλάχιστον έναν τελεστή.

Inline Temp

Η εφαρμογή αυτής της τεχνικής ανακατασκευής ευθυγραμμίζει προσωρινές μεταβλητές οι οποίες συνήθως χρησιμοποιούνται για τη αποθήκευση του αποτελέσματος της κλήσης μιας μεθόδου. Η δυνατότητα εντοπισμού για αυτήν την

τεχνική ανακατασκευής προτείνει υποψήφιες μεταβλητές που πληρούν τις εξής συνθήκες:

- Η μεταβλητή συμμετέχει μόνο σε μια σχέση ανάθεσης.
- Η ανάθεση αυτή της μεταβλητής είναι το αποτέλεσμα της κλήσης μιας μεθόδου.

Split Temporary Variable

Όπως προαναφέρθηκε στην ενότητα 2.5 τοπικές μεταβλητές που αναθέτονται πάνω από μια φορά σε μια μέθοδο είναι υποδεικνύει πως αυτή η μεταβλητή έχει τουλάχιστον δύο αρμοδιότητες στον κώδικα. Για την εξάλειψη αυτού του φαινομένου χρησιμοποιείται αυτή η τεχνική ανακατασκευής. Η δυνατότητα εντοπισμού για αυτήν την τεχνική ανακατασκευής προτείνει υποψήφιες μεταβλητές που πληρούν τις εξής συνθήκες:

- Η μεταβλητή συμμετέχει σε πολλαπλές αναθέσεις.

Παρατήρηση: Καθώς είναι αρκετά πολύπλοκο να διαπιστωθεί ο σκοπός κάθε μεταβλητής με παραπάνω από μια ανάθεση, σαν αποτέλεσμα υποψήφιων μεταβλητών για αυτή την τεχνική προτείνονται επίσης μεταβλητές που χρησιμοποιούνται για τον υπολογισμό αθροισμάτων σε βρόγχους, μετρητές που υπολογίζουν διαφορές τιμές και συναθροιστικές μεταβλητές γενικότερα.

Replace Method with Method Object

Σύμφωνα με τον Fowler [7] σκοπός αυτής της τεχνικής ανακατασκευής είναι η εφαρμογή της ως εναλλακτική επιλογή σε περίπτωση που ο πολύπλοκος κώδικας μιας μεθόδου καθιστά τη χρήση Extract Method δύσκολη. Με βάση αυτό αξιοποιείται η μέθοδος εντοπισμού που χρησιμοποιείται στο Extract Method και προτείνονται οι ίδιες μέθοδοι προς αντικατάσταση. Η επιλογή της κατάλληλης τεχνικής ανακατασκευής ανήκει στην ευχέρεια του προγραμματιστή καθώς αυτός μπορεί να κρίνει την πολυπλοκότητα του κώδικα του για την χρήση της μιας ή της άλλης.

Remove Assignments to Parameters

Σκοπός αυτής της τεχνικής ανακατασκευής είναι η αντικατάσταση οποιασδήποτε ανάθεσης γίνεται στις παραμέτρους των μεθόδων. Η δυνατότητα εντοπισμού για αυτήν την τεχνική ανακατασκευής διασχίζει όλες τις παραμέτρους κάθε μεθόδου, ελέγχει εάν για αυτές γίνεται κάποια ανάθεση μέσα στον κώδικα της μεθόδου τους και τις προτείνει ως υποψήφιες στον χρήστη εάν ισχύει αυτό.

Introduce Explaining Variable

Αυτή η τεχνική ανακατασκευής στοχεύει στην απλοποίηση περίπλοκων εκφράσεων στον κώδικα που δυσκολεύουν την ανάγνωση του. Η δυνατότητα εντοπισμού για αυτήν την τεχνική ανακατασκευής καταδεικνύει ως υποψήφιες τις εκφράσεις που περιέχουν πολλούς τελεστές υπολογισμού.

Introduce Parameter Object

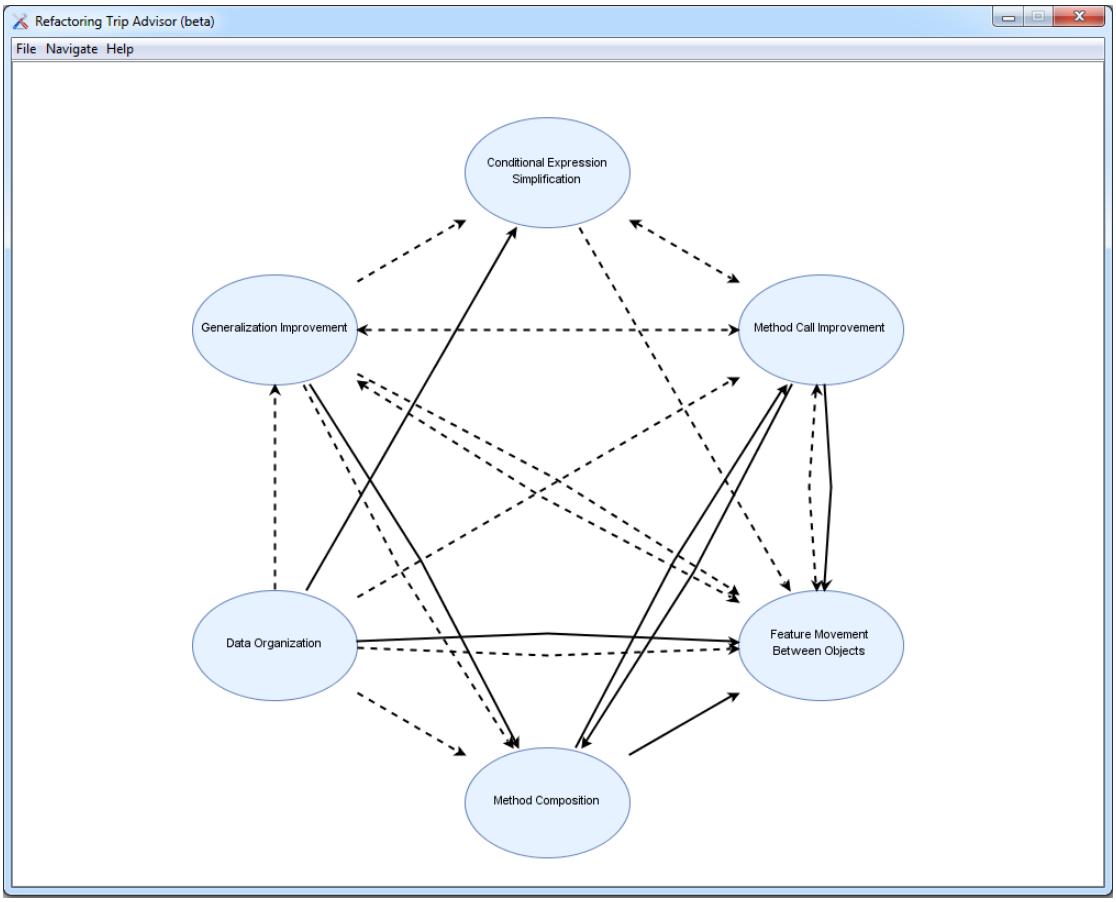
Όταν ένα συγκεκριμένο γκρουπ παραμέτρων εμφανίζεται σε αρκετές μεθόδους του συστήματος ή όταν η λίστα παραμέτρων μιας μεθόδου είναι αρκετά μεγάλη τότε ενδείκνυται η εφαρμογή αυτής της τεχνικής ανακατασκευής. Η δυνατότητα εντοπισμού για αυτήν την τεχνική ανακατασκευής προτείνει την εφαρμογή αυτής σε μεθόδους που έχουν πολλές παραμέτρους.

4.3 Πλοήγηση με τον Refactoring Trip Advisor

Στις προηγούμενες ενότητες του Κεφαλαίου παρουσιάστηκε ο σχεδιασμός του εργαλείου και περιγράφηκε ο τρόπος υλοποίησης των δυνατοτήτων εντοπισμού ευκαιριών εφαρμογής συγκεκριμένων τεχνικών ανακατασκευής. Σε αυτή την ενότητα θα γίνει μια παρουσίαση της διαδικασίας πλοήγησης και κάποιων σεναρίων χρήσης του εργαλείου που αναπτύχθηκε σε αυτή τη διατριβή.

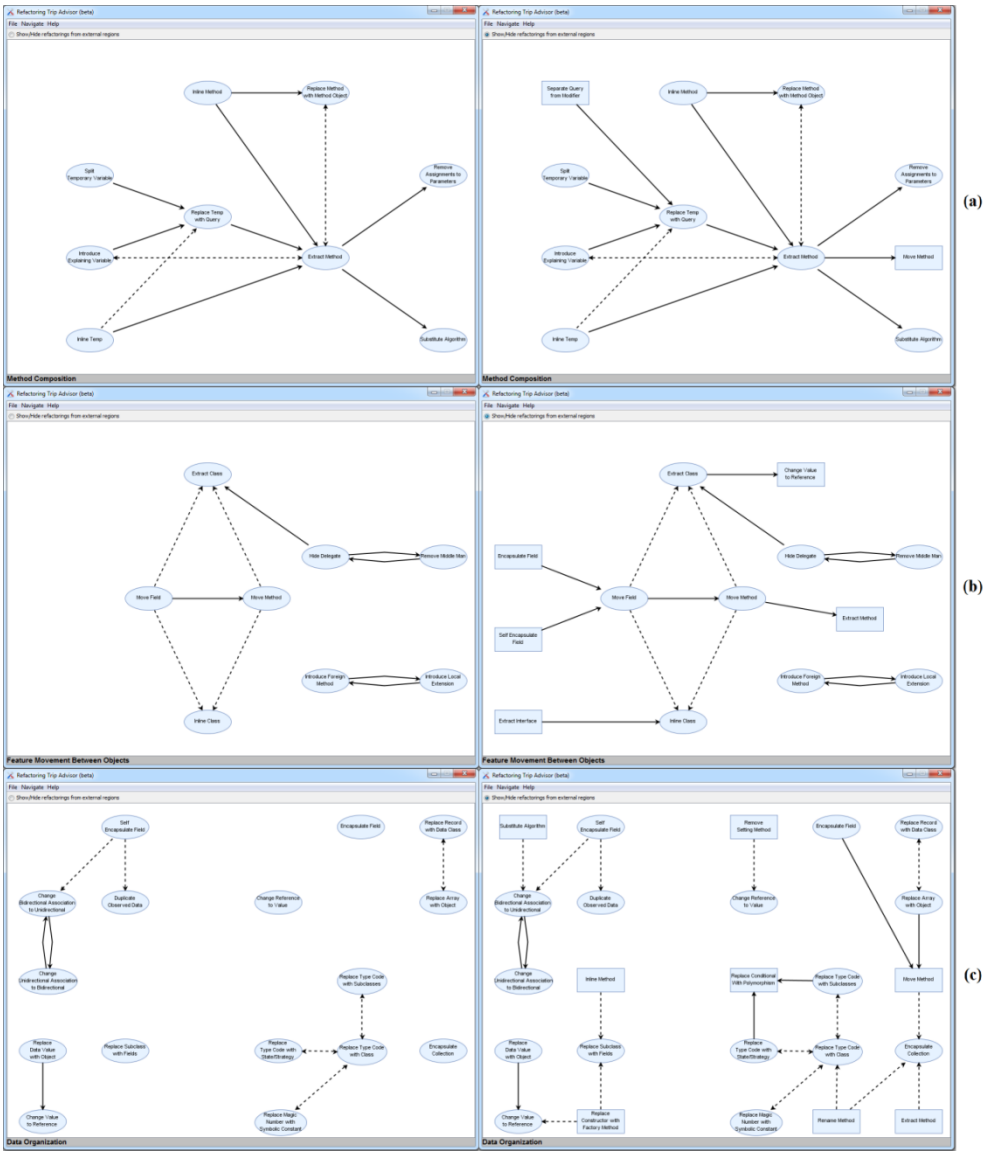
Για την εκκίνηση του εργαλείου έγινε μια προσθήκη επιλογής, με το όνομα του εργαλείου, στο ήδη υπάρχον “Refactor” μενού του Eclipse. Ένας χρήσιμος οδηγός για την δημιουργία μιας απλής επέκτασης στο Eclipse μπορεί να βρεθεί στο [20].

Ενεργοποιώντας τον Refactoring Trip Advisor παρουσιάζεται το αρχικό παράθυρο του εργαλείου όπως φαίνεται στο Σχήμα 4.3. Στο αρχικό παράθυρο παρουσιάζεται ένας γράφημα στο οποίο κάθε κόμβος αντιστοιχεί σε μια από τις έξι κατηγορίες ανακατασκευής που περιγράφηκαν στην υποενότητα 3.2.1. Οι συσχετίσεις μεταξύ αυτών αντιστοιχούν σε συσχετίσεις μεταξύ συγκεκριμένων τεχνικών ανακατασκευής από την μια κατηγορία στην άλλη. Η πληροφορία του γράφήματος αυτού μπορεί να μην είναι χρήσιμη όσο των έξι Χαρτών για τον χρήστη, παρ' όλα αυτά δίνει μια αρχική άποψη συσχέτισης μεταξύ των κατηγοριών ανακατασκευής. Ο Refactoring Trip Advisor περιέχει επίσης ένα βασικό μενού στην κορυφή του παραθύρου το οποίο παρέχει βασικές λειτουργίες στον χρήστη όπως το κλείσιμο του εργαλείου, η επιστροφή στο αρχικό παράθυρο και την πρόσβαση στα περιεχόμενα βοήθειας σύμφωνα με το Σχήμα 4.3.

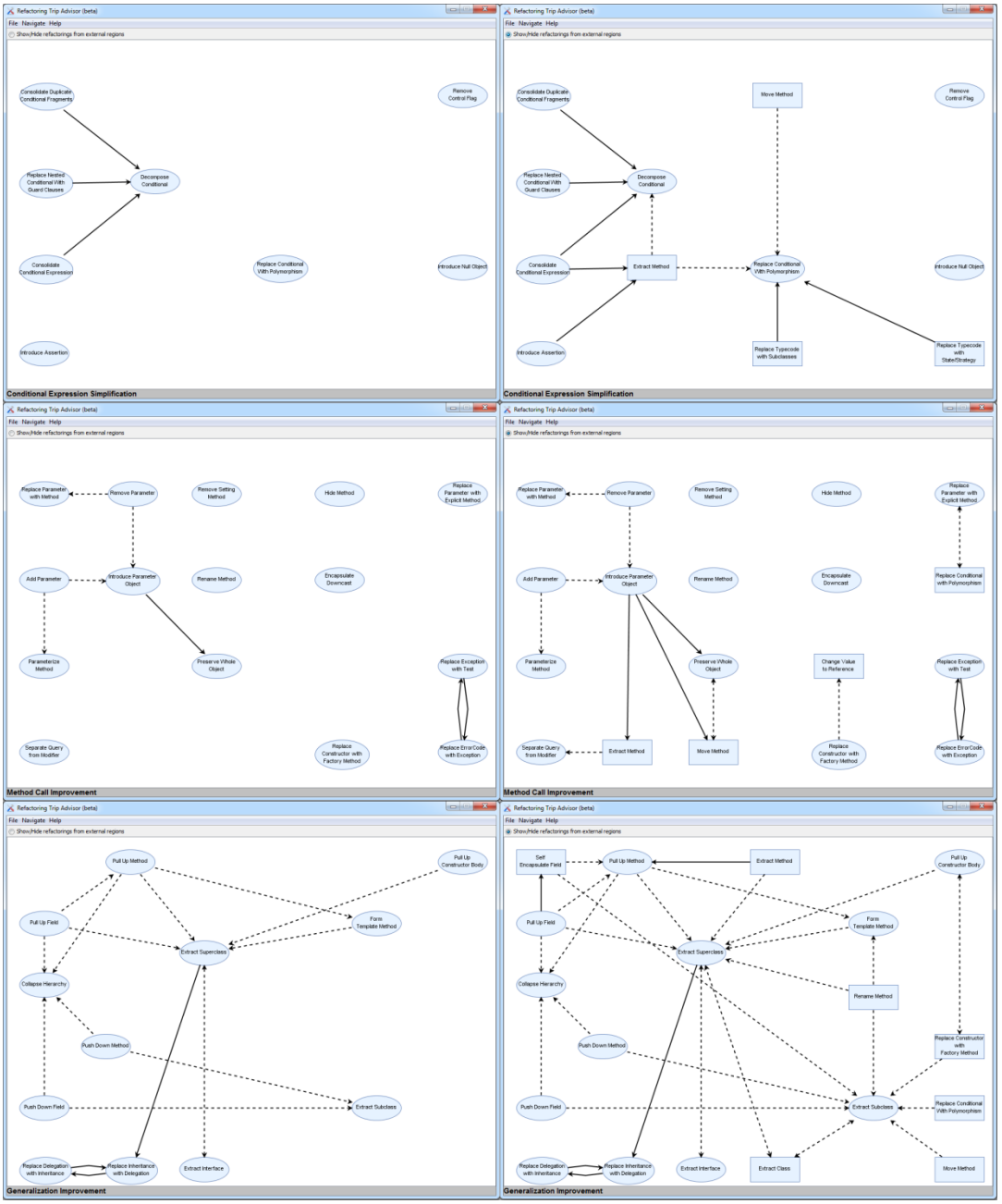


Σχήμα 4.4 Αρχικό Παράθυρο του Refactoring Trip Advisor

Για να επισκεφτεί ο χρήστης κάποιον από τους έξι διαθέσιμους Χάρτες συσχετίσεων αρκεί να κάνει δεξί κλικ στον κόμβο με το αντίστοιχο όνομα που βρίσκεται στην αρχική οθόνη. Κάνοντας το αυτό, το αρχικό γράφημα των κατηγοριών αντικαθιστάται από τον Χάρτη της κατηγορίας που επιλέχθηκε. Οι Χάρτες κάθε κατηγορίας συμβαδίζουν με την περιγραφή τους στην υποενότητα 3.2.1 και επιπλέον δίνουν την δυνατότητα στον χρήστη να εμφανίσει ή να κρύψει τις συσχετίσεις μεταξύ εσωτερικών και εξωτερικών τεχνικών ανακατασκευής μέσω ενός κουμπιού όπως φαίνεται στο Σχήμα 4.5 και 4.6.



Σχήμα 4.5 Χάρτης Ανακατασκευών για (a) Method Composition, (b) Feature Movement Between Objects και (c) Data Organization. (Αριστερά: Εσωτερικές Συσχετίσεις Μόνο. Δεξιά: Εσωτερικές και Εξωτερικές Συσχετίσεις)

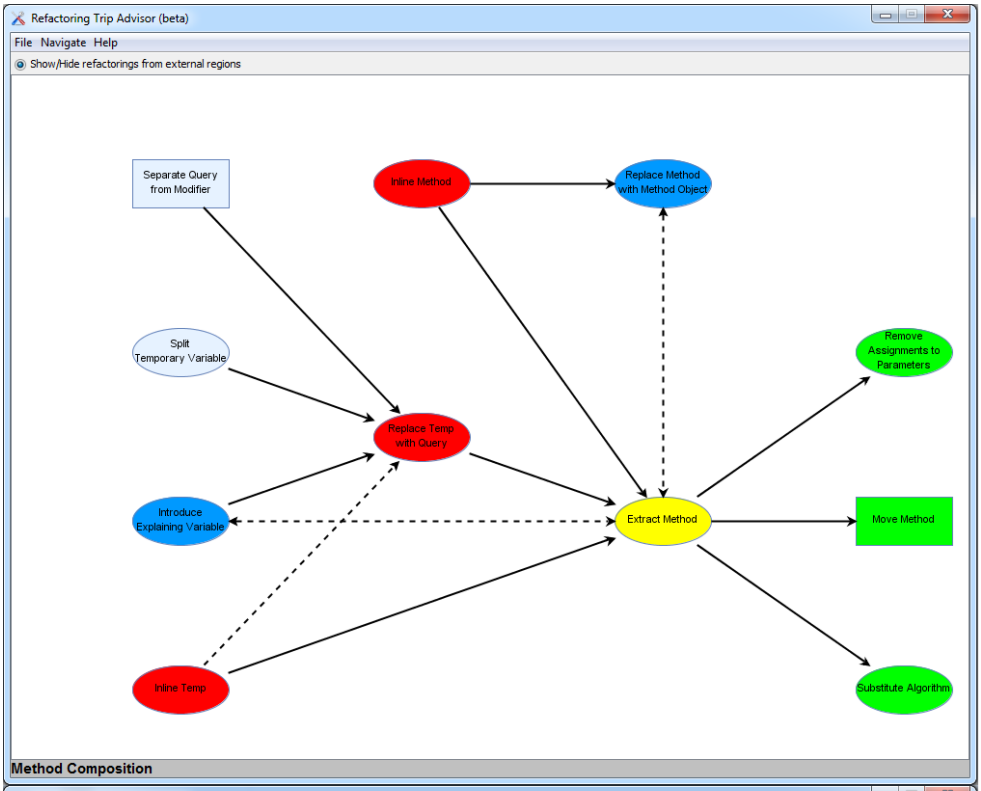


Σχήμα 4.6 Χάρτης Ανακατασκευών για (a) Conditional Expression Simplification, (b) Method Call Improvement και (c) Generalization Improvement. (Αριστερά: Εσωτερικές Συσχετίσεις Μόνο. Δεξιά: Εσωτερικές και Εξωτερικές Συσχετίσεις)

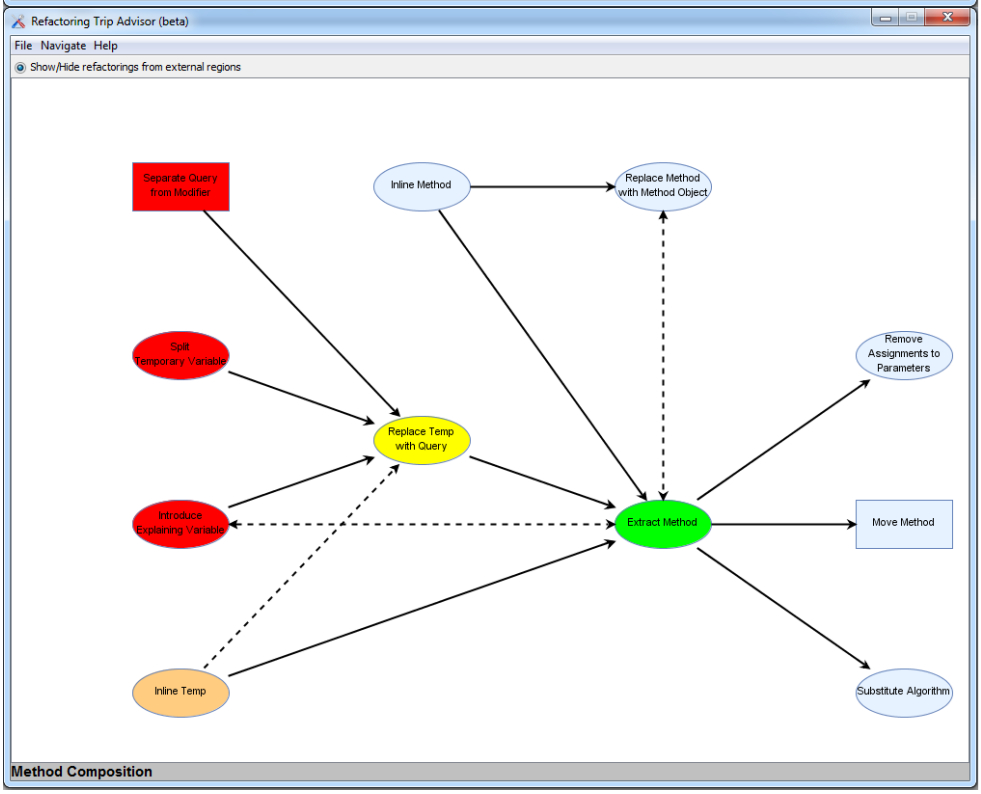
Πέρα από μια στατική αναπαράσταση ως ενός γραφήματος συσχετίσεων μεταξύ των τεχνικών ανακατασκευής, κάθε Χάρτης παρέχει επιπρόσθετες λειτουργίες για την υποβοήθηση της διαδικασίας ανακατασκευής. Ένα βασικό οπτικό υποβοήθημα που παρέχει ο Refactoring Trip Advisor είναι ο χρωματισμός του μονοπατιού ανακατασκευής. Επιλέγοντας ο χρήστης κάποια τεχνική ανακατασκευής, αυτόματα

οι γειτνιάζοντες κόμβοι χρωματίζονται κατάλληλα ώστε να δοθεί περισσότερη έμφαση στις συσχετιζόμενες τεχνικές και στην προτεινόμενη σειρά εφαρμογής αυτών. Ένα παράδειγμα χρωματισμού του μονοπατιού φαίνεται στο Σχήμα 4.7 και η σημασιολογία των διαφορετικών χρωμάτων εξηγείται παρακάτω:

- Κίτρινο: Με κίτρινο χρωματίζεται ο κόμβος τον οποίο επιλέγει ο χρήστης αρχικά. Πρόκειται για την τεχνική ανακατασκευής για την οποία ενδιαφέρεται να εφαρμόσει ο χρήστης.
- Κόκκινο: Με κόκκινο χρωματίζονται οι τεχνικές ανακατασκευής τις οποίες ο Refactoring Trip Advisor προτείνει να εφαρμοστούν πριν (σχέση διαδοχής) την επιλεγμένη τεχνική ανακατασκευής που ενδιαφέρει τον χρήστη. Ουσιαστικά, οι τεχνικές με κόκκινο χρώμα πρόκειται να διευκολύνουν την εφαρμογή της τεχνικής που ενδιαφέρει τον χρήστη.
- Πράσινο: Με πράσινο χρωματίζονται οι τεχνικές ανακατασκευής που ο Refactoring Trip Advisor προτείνει να εφαρμοστούν μετά (σχέση διαδοχής) την επιλεγμένη τεχνική ανακατασκευής που ενδιαφέρει τον χρήστη. Οι τεχνικές με πράσινο χρώμα προτείνονται γιατί ο ανακατασκευασμένος πλέον κώδικας δημιούργησε ευκαιρίες για την εφαρμογή αυτών.
- Μπλε: Με μπλε χρωματίζονται οι εναλλακτικές τεχνικές ανακατασκευής. Πρόκειται για εκείνες τις τεχνικές ανακατασκευής που ο Refactoring Trip Advisor προτείνει ως εναλλακτικές επιλογές (σχέση "Αντί για") σε περίπτωση που η εφαρμογή της τεχνικής που ενδιαφέρει τον χρήστη είναι δύσκολη ή πολύπλοκη.
- Πορτοκαλί: Με πορτοκαλί χρωματίζονται οι τεχνικές που είτε αποτελούν υπερσύνολο (σχέση "Μέρος από") της ενδιαφερόμενης τεχνικής, είτε η ενδιαφερόμενη τεχνική αποτελεί υπερσύνολο (σχέση "Μέρος από") αυτών. Έτσι ο χρήστης ενημερώνεται σε περίπτωση που η εφαρμογή της ενδιαφερόμενης τεχνικής θα έχει ως συνέπεια την εφαρμογή κάποιας άλλης.



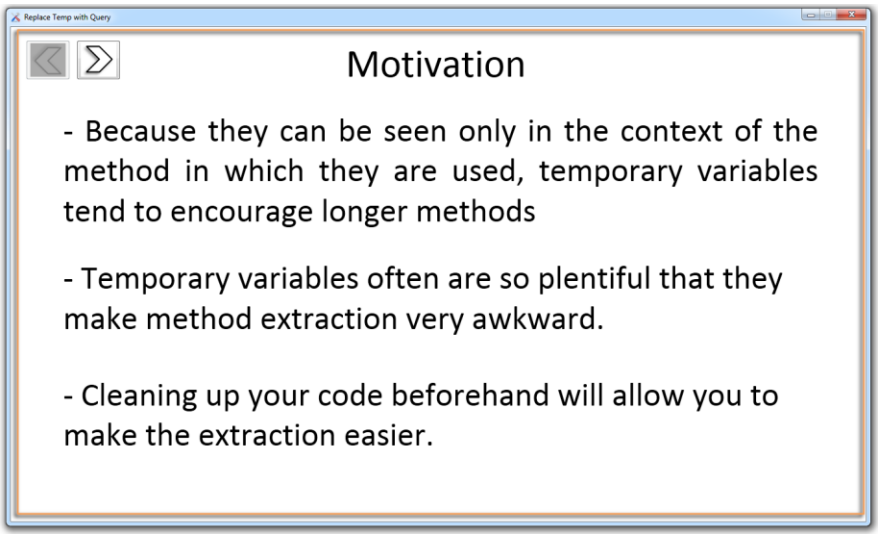
(a)



(b)

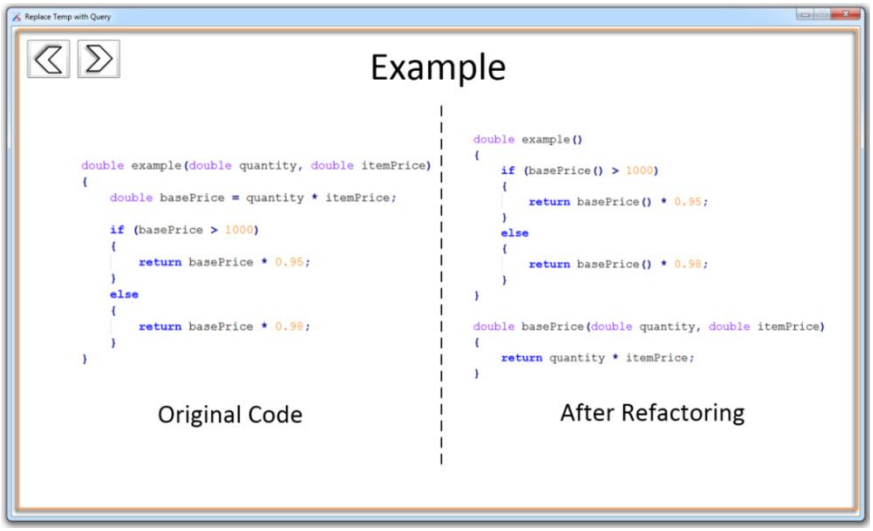
Σχήμα 4.7 Χρωματισμός μονοπατιού για (a) την Τεχνική Extract Superclass και (b) για την Τεχνική Replace Temp with Query

Εκτός από τον χρωματισμό του μονοπατιού που βοηθάει το χρήστη να εφαρμόσει μια σειρά από τεχνικές ανακατασκευής, ο Refactoring Trip Advisor παρέχει και χρήσιμες πληροφορίες για κάθε τεχνική ξεχωριστά. Κάθε τεχνική ανακατασκευής είναι εξοπλισμένη με ένα πάνελ διαφανειών στο οποίο ο χρήστης έχει πρόσβαση κάνοντας δεξί κλικ πάνω στον αντίστοιχο κόμβο της. Το κάθε πάνελ αποτελείται από τρεις διαφάνειες οι οποίες παρέχουν διαφορετικές πληροφορίες για την επιλεγμένη τεχνική ανακατασκευής. Ο χρήστης μπορεί να μεταβεί στις διάφορες διαφάνειες χρησιμοποιώντας τα κουμπιά βελάκια που βρίσκονται πάνω αριστερά στο πάνελ. Η πρώτη διαφάνεια περιέχει τα κίνητρα χρήσης της τεχνικής ανακατασκευής. Ουσιαστικά σε αυτή τη διαφάνεια παρουσιάζονται οι λόγοι για τους οποίους ο χρήστης θα πρέπει να χρησιμοποιήσει αυτή την τεχνική όπως το τι βελτιώσεις επιφέρει η εφαρμογή της στον κώδικα. Δίνονται επίσης πληροφορίες για το πότε θα ήταν καλό να εφαρμοστεί η τεχνική παίρνοντας υπ' όψη άλλες που συσχετίζονται άμεσα με αυτή. Ένα παράδειγμα αυτής της διαφάνειας φαίνεται στο Σχήμα 4.8.



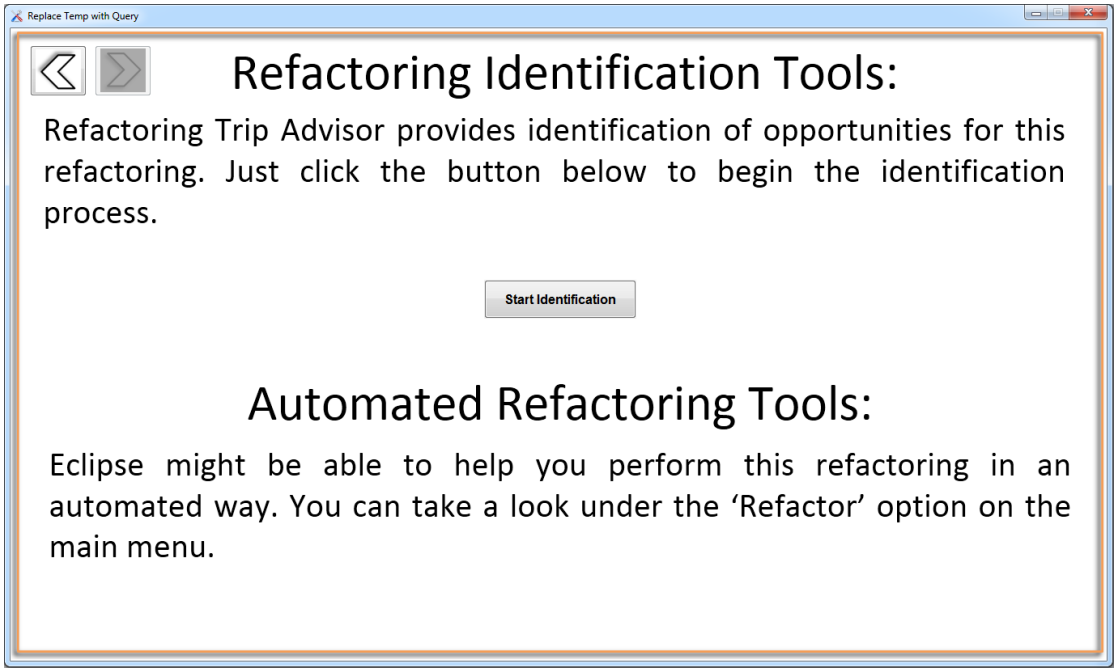
Σχήμα 4.8 Διαφάνεια με τα Κίνητρα Χρήσης της Τεχνικής Replace Temp with Query

Η δεύτερη διαφάνεια περιέχει ένα παράδειγμα χρήσης της επιλεγμένης ανακατασκευής στην πράξη. Στη διαφάνεια παρουσιάζεται ένα απλό κομμάτι κώδικα όπως αυτό είναι πριν την εφαρμογή της ανακατασκευής καθώς και πως θα είναι μετά. Το Σχήμα 4.9 περιγράφει ένα παράδειγμα αυτής της διαφάνειας για μια τεχνική ανακατασκευής.



Σχήμα 4.9 Διαφάνεια με το Παράδειγμα Χρήσης της Τεχνικής Replace Temp with Query

Η τρίτη και τελευταία διαφάνεια περιέχει πληροφορίες για τα διάφορα αυτοματοποιημένα εργαλεία που συσχετίζονται με την επιλεγμένη τεχνική ανακατασκευής. Συγκεκριμένα, η διαφάνεια περιέχει πληροφορίες σχετικά με διαθέσιμα εργαλεία που επικεντρώνονται στην εύρεση σημείων στον κώδικα που χρήζουν την εφαρμογή της τεχνικής και εργαλεία τα οποία αυτοματοποιούν την εφαρμογή αυτής. Για τις τεχνικές ανακατασκευής που ο Refactoring Trip Advisor έχει ήδη υλοποιημένη τη δυνατότητα του αυτόματου εντοπισμού, εμφανίζεται στην διαφάνεια κατάλληλο κουμπί με τίτλο “Start Identification”, το πάτημα του οποίου ξεκινά αυτή τη διαδικασία. Για τις υπόλοιπες τεχνικές παρέχεται ένας σύνδεσμος στην βιβλιογραφία που αφορά τον εντοπισμό των τεχνικών ανακατασκευής. Επίσης σε αυτή τη διαφάνεια αναφέρεται εάν παρέχεται η δυνατότητα της αυτοματοποιημένης εφαρμογής της τεχνικής από το Eclipse. Ένα παράδειγμα αυτής της διαφάνειας φαίνεται στο Σχήμα 4.10.



Σχήμα 4.10 Διαφάνεια με τα Εργαλεία Αυτοματοποιημένου Εντοπισμού και Εφαρμογής της Τεχνικής Replace Temp with Query

Οι δύο πρώτες διαφάνειες (Motivation και Example) για κάθε μία από τις 68 τεχνικές ανακατασκευής παρουσιάζονται για πληρότητα στο Παράρτημα αυτής της εργασίας (Σχήματα Π.1 έως Π.15).

ΚΕΦΑΛΑΙΟ 5. ΑΞΙΟΛΟΓΗΣΗ

5.1 Εισαγωγή

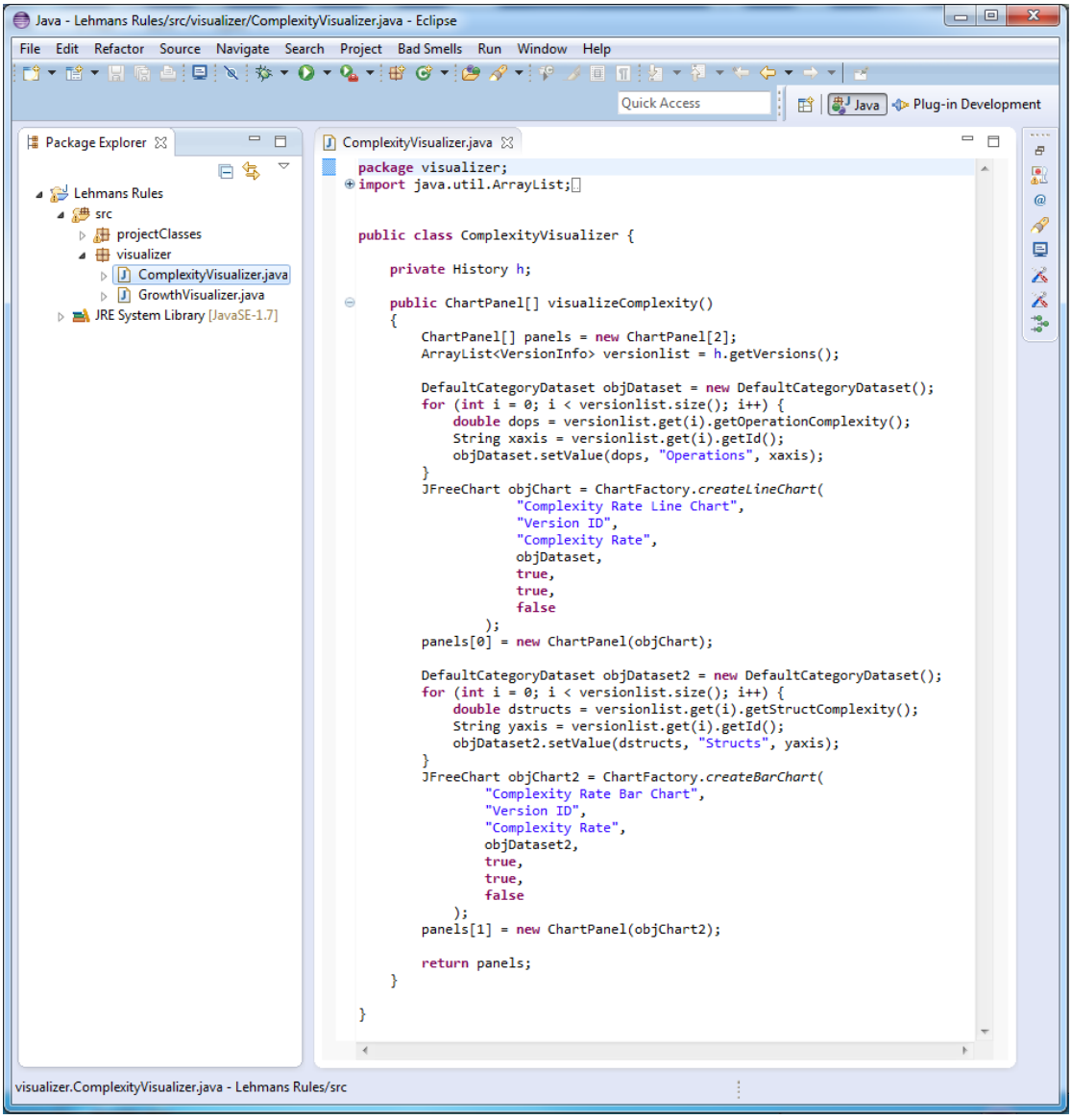
Στα προηγούμενα κεφάλαια περιγράφηκε η βασική συνεισφορά της εργασίας στην υποβοήθηση της διαδικασίας ανακατασκευής λογισμικού. Παρουσιάστηκε ο Χάρτης Ανακατασκευών και επεξηγήθηκαν οι διάφορες συσχετίσεις μεταξύ αυτών. Επίσης έγινε παρουσίαση του Refactoring Trip Advisor, της εφαρμογής που αναπτύχθηκε βασισμένη στον Χάρτη Ανακατασκευών ώστε να βοηθήσει το χρήστη σε αυτή τη διαδικασία. Θα πρέπει όμως να γίνει μια αξιολόγηση των συνεισφορών ώστε να εκτιμηθεί πόσο αυτές βοηθούν το χρήστη στην πράξη. Στις επόμενες ενότητες θα παρουσιαστεί η διαδικασία αξιολόγησης της εργασίας. Θα περιγραφούν οι εργασίες που δόθηκαν στους χρήστες να εφαρμόσουν και θα σχολιαστούν τα αποτελέσματα αυτών.

5.2 Διαδικασία Αξιολόγησης

Στην διαδικασία αξιολόγησης της εργασίας πήραν μέρος 16 χρήστες, με εμπειρία πάνω στην ανάπτυξη λογισμικού. Αυτοί κλήθηκαν να εκτελέσουν δύο εργασίες στο Eclipse οι οποίες περιγράφονται παρακάτω.

1η εργασία

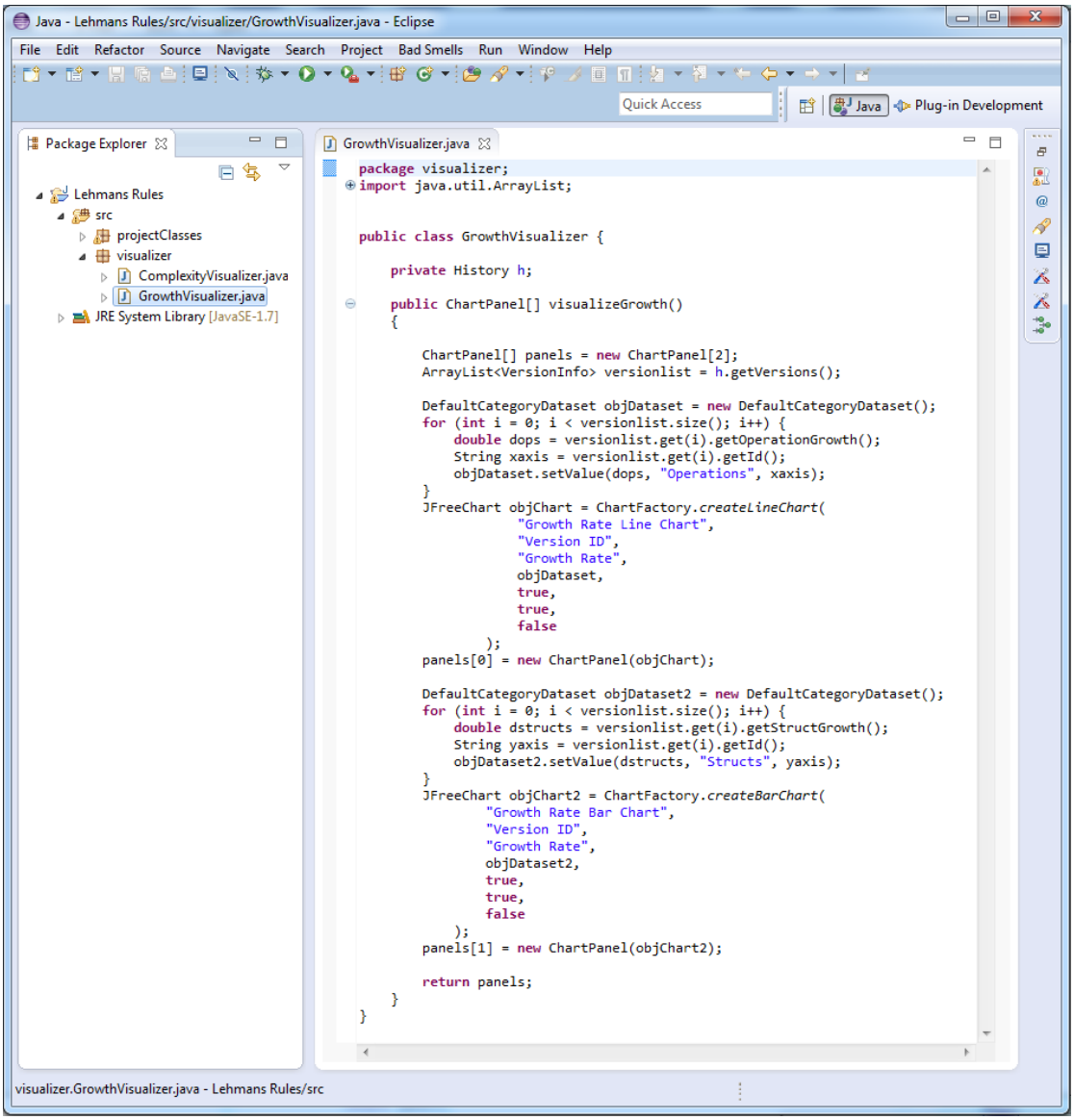
Στους χρήστες δόθηκε ο κώδικας μιας κλάσης (ComplexityVisualizer) η οποία περιέχει μόνο μια μέθοδο (public ChartPanel[] visualizeComplexity()). Ο κώδικας αυτός προέρχεται από παλαιό project φοιτητών του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστήμιου Ιωαννίνων και φαίνεται στο Σχήμα 5.1. Οι χρήστες έπειτα κλήθηκαν να απλοποιήσουν τον κώδικα της μεθόδου χρησιμοποιώντας τεχνικές ανακατασκευής κατά την κρίση τους.



Σχήμα 5.1 Κώδικας της Κλάσης ComplexityVisualizer

2η Εργασία

Στους χρήστες δόθηκε ο κώδικας μιας κλάσης (GrowthVisualizer) από το ίδιο project η οποία περιέχει μόνο μια μέθοδο (public ChartPanel[] visualizeGrowth()) όπως φαίνεται στο Σχήμα 5.2. Οι χρήστες έπειτα κλήθηκαν να απλοποιήσουν τον κώδικα της μεθόδου χρησιμοποιώντας τη βοήθεια του Refactoring Trip Advisor. Συγκεκριμένα καθώς πρόκειται για απλοποίηση του κώδικα της μεθόδου οι χρήστες συμβουλευτήκαν να χρησιμοποιήσουν τεχνικές ανακατασκευής της κατηγορίας Method Composition αξιοποιώντας την δυνατότητα του αυτόματου εντοπισμού που παρέχεται.



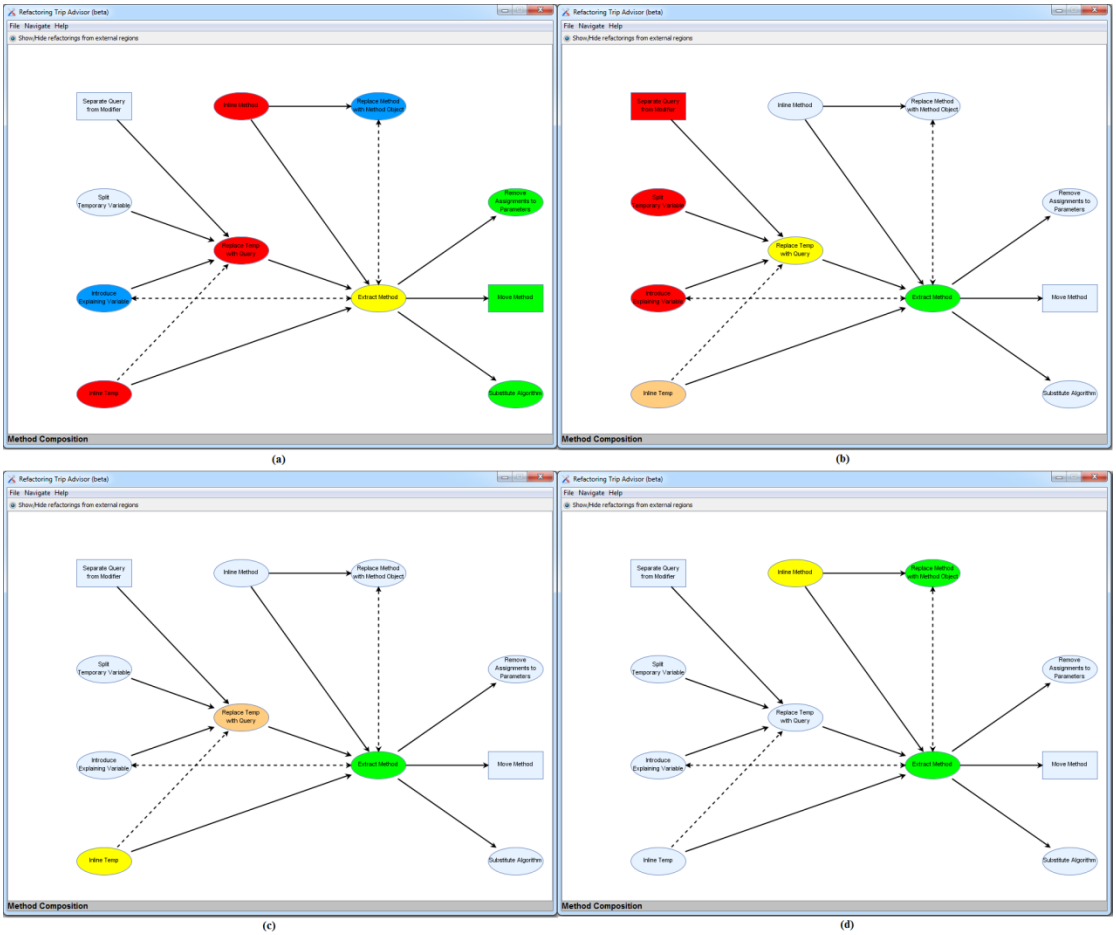
Σχήμα 5.2 Κώδικας της Κλάσης GrowthVisualizer

Οι χρήστες κλήθηκαν να εκτελέσουν πρώτα την πρώτη εργασία και έπειτα τη δεύτερη χωρίς να τους επιβληθεί κάποιος χρονικός περιορισμός για όλη τη διαδικασία. Σκοπός της πρώτης εργασίας είναι αξιολόγηση της εμπειρίας του χρήστη με την διαδικασία ανακατασκευής κώδικα. Σκοπός της δεύτερης εργασίας είναι η αξιολόγηση της εμπειρίας του χρήστη με την χρήση του Refactoring Trip Advisor. Βασικό κομμάτι της αξιολόγησης είναι η συσχέτιση των αποτελεσμάτων των δύο αυτών εργασιών και για αυτό το λόγο ο κώδικας των δύο μεθόδων είναι αρκετά παρόμοιος. Εξετάζουμε το πόσες και ποιες τεχνικές ανακατασκευής ο χρήστης εφήρμοσε με ($1^{η}$ εργασία) και χωρίς ($2^{η}$ εργασία) την χρήση του Refactoring Trip Advisor. Εξετάζουμε επίσης το χρόνο εκτέλεσης των δύο αυτών εργασιών. Μετά το πέρας των δύο εργασιών ο κάθε χρήστης εκλήθη να συμπληρώσει ένα ερωτηματολόγιο το οποίο περιλαμβάνει ερωτήσεις για την εμπειρία του με την διαδικασία ανακατασκευής κώδικα γενικά αλλά και για την εμπειρία του με την χρήση του Refactoring Trip Advisor. Καθώς δεν υπάρχει κάποιου είδους επίβλεψη των χρηστών κατά την διάρκεια εκτέλεσης των εργασιών, τους ζητήθηκε επίσης να συμπληρώσουν στο ερωτηματολόγιο ποιες τεχνικές ανακατασκευής εφάρμοσαν σε κάθε περίπτωση και το χρόνο κάθε διαδικασίας ξεχωριστά. Το ερωτηματολόγιο παραθέτεται στο Παράρτημα της εργασίας (Σχήμα Π.16 – Π.18) για λόγους πληρότητας.

5.3 Προτεινόμενη Ακολουθία Ανακατασκευών

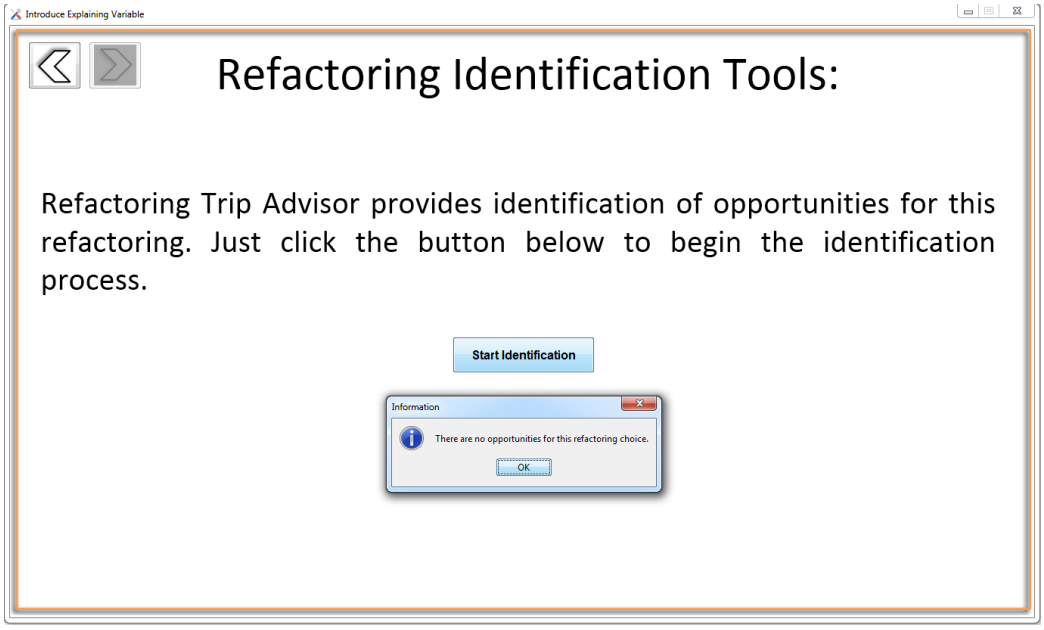
Όπως αναφέρθηκε στην προηγούμενη ενότητα κύριος σκοπός της εκτέλεσης των δύο εργασιών είναι η σύγκριση των αποτελεσμάτων μεταξύ αυτών. Καθώς ο κώδικας και στις δύο περιπτώσεις είναι παρόμοιος, βασικός γνώμονας για την συσχέτιση των αποτελεσμάτων αλλά και τον υπολογισμό διάφορων μετρικών αποτελεί η προτεινόμενη ακολουθία ανακατασκευών που δίνει ο Refactoring Trip Advisor. Προτεινόμενη ακολουθία ανακατασκευών είναι η σειρά των ανακατασκευών που προτείνεται από τον Refactoring Trip Advisor ακολουθώντας τις συσχετίσεις στον Χάρτη Ανακατασκευών κάποιας κατηγορίας. Στη συνέχεια περιγράφεται βήμα-βήμα το πως προκύπτει η προτεινόμενη ακολουθία ανακατασκευών για τον κώδικα της κλάσης GrowthVisualizer (Σχήμα 5.2) από τον Χάρτη Ανακατασκευών Method Composition αξιοποιώντας την δυνατότητα αυτόματου εντοπισμού του Refactoring Trip Advisor.

Ακολουθώντας τις οδηγίες για την εκτέλεση της δεύτερης εργασίας ξεκινούμε τον Refactoring Trip Advisor μέσω του Eclipse και επιλέγουμε την κατηγορία Method Composition από το αρχικό του μενού. Σαν κόμβο εκκίνησης επιλέγουμε τον Extract Method καθώς αποτελεί τη βασικότερη τεχνική για την απλοποίηση και μείωση του μεγέθους του κώδικα. Επιλέγοντας την, το μονοπάτι χρωματίζεται όπως φαίνεται στο Σχήμα 5.3(a). Συμβουλευόμενοι το χρωματισμένο γράφημα, ο Refactoring Trip Advisor προτείνει να ερευνηθούν πιθανότητες εφαρμογής των Replace Temp with Query, Inline Method και Inline Temp πριν το Extract Method έτσι ώστε να εξαλειφθούν οι πιθανές προσωρινές μεταβλητές από τον κώδικα. Επιλέγοντας τις τεχνικές Replace Temp with Query, Inline Temp και Inline Method ο χρωματισμός του γραφήματος αλλάζει όπως φαίνεται στο Σχήμα 5.3(b), 5.3(c) και 5.3(d) αντίστοιχα για κάθε μία.

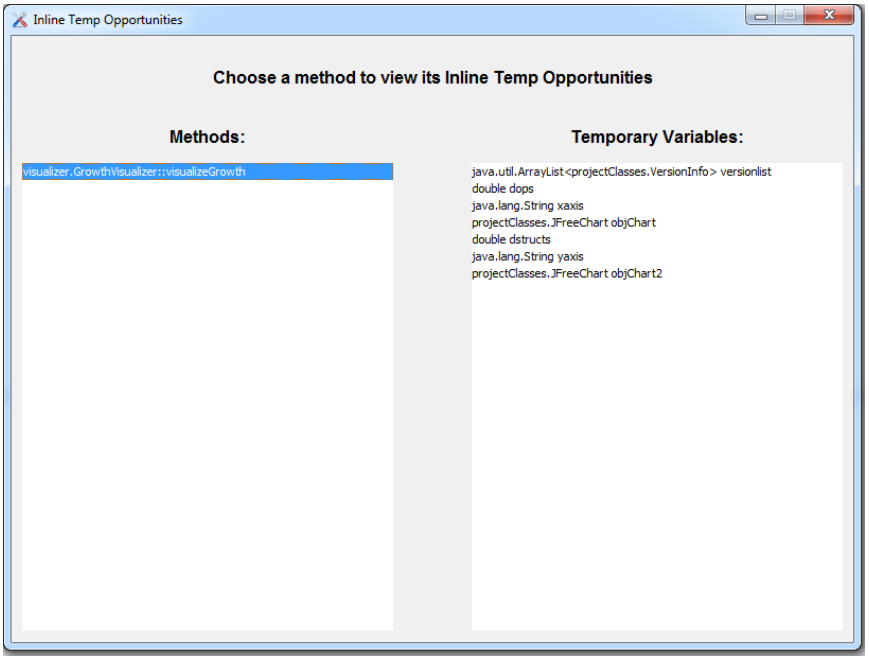


Σχήμα 5.3 Χρωματισμός Γραφήματος για την Τεχνική (a) Extract Method, (b) Replace Temp with Query, (c) Inline Temp και (d) Inline Method

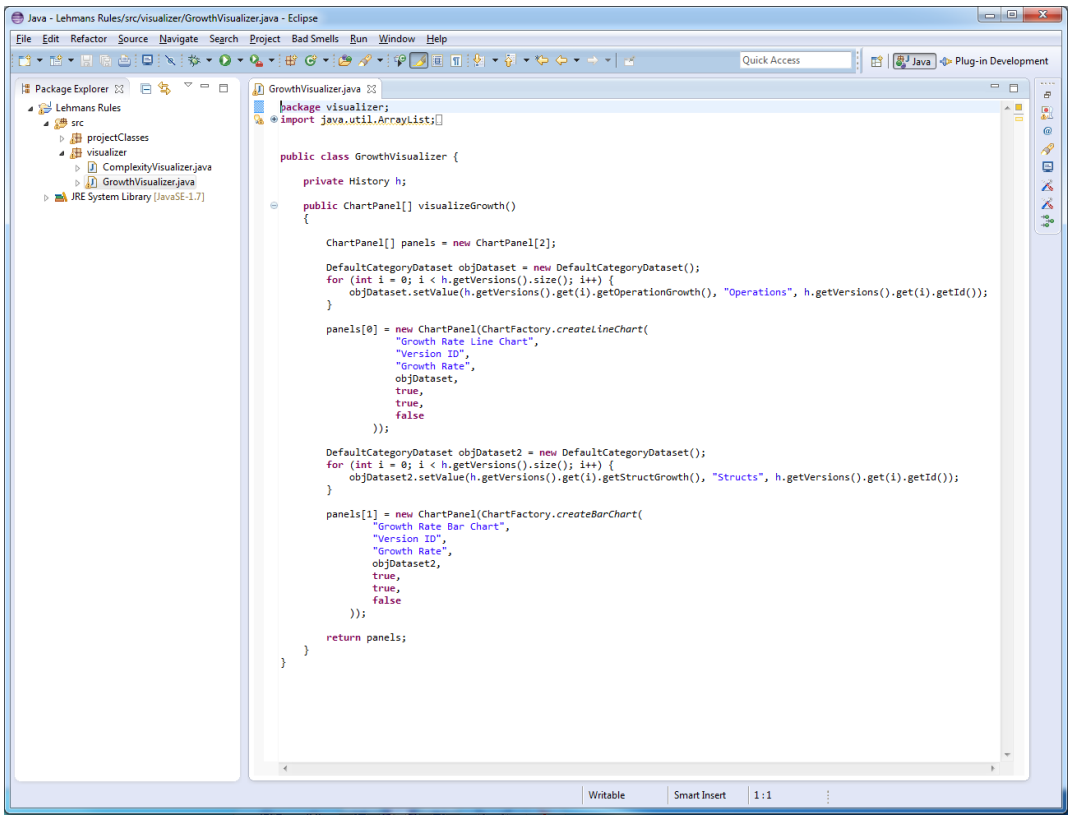
Παρατηρούμε ότι για την εφαρμογή του Replace Temp with Query ο Refactoring Trip Advisor προτείνει να ελεγχθούν πιθανές ευκαιρίες για Separate Query from Modifier, Split Temporary Variable και Introduce Explaining Variable. Για να το ελέγξουμε χρησιμοποιούμε την δυνατότητα του αυτόματου εντοπισμού ευκαιριών για αυτές τις τεχνικές που ο Refactoring Trip Advisor παρέχει. Κάνοντάς το αυτό ο Refactoring Trip Advisor απαντά πως δεν υπάρχουν ευκαιρίες εφαρμογής τέτοιου είδους ανακατασκευών στον κώδικα με ανάλογο μήνυμα όπως φαίνεται στο Σχήμα 5.4. Επιστρέφουμε λοιπόν να ψάξουμε για ευκαιρίες για Replace Temp with Query αλλά και πάλι παίρνουμε την ίδια απάντηση. Το ίδιο ακριβώς συμβαίνει και για το Inline Method. Συνεχίζουμε λοιπόν να ψάξουμε για το Inline Temp. Σε αυτή την περίπτωση ο Refactoring Trip Advisor ανιχνεύει ευκαιρίες εφαρμογής αυτής της ανακατασκευής και τις παρουσιάζει σε ξεχωριστό αναδυόμενο παράθυρο όπως φαίνεται στο Σχήμα 5.5. Παρατηρούμε ότι προτείνεται η εφαρμογή του Inline Temp στις εξής επτά προσωρινές μεταβλητές: versionlist, dops, xaxis, objChart, dstructs, yaxis και objChart2. Το επόμενο βήμα είναι να εφαρμόσουμε αυτές τις ανακατασκευές στον κώδικα πριν προχωρήσουμε να ελέγξουμε το Extract Method. Ο κώδικας έπειτα από την ευθυγράμμιση των επτά προσωρινών μεταβλητών φαίνεται στο Σχήμα 5.6.



Σχήμα 5.4 Μήνυμα σε Περίπτωση που δεν Βρεθούν Ευκαιρίες Εφαρμογής μιας Τεχνικής Ανακατασκευής

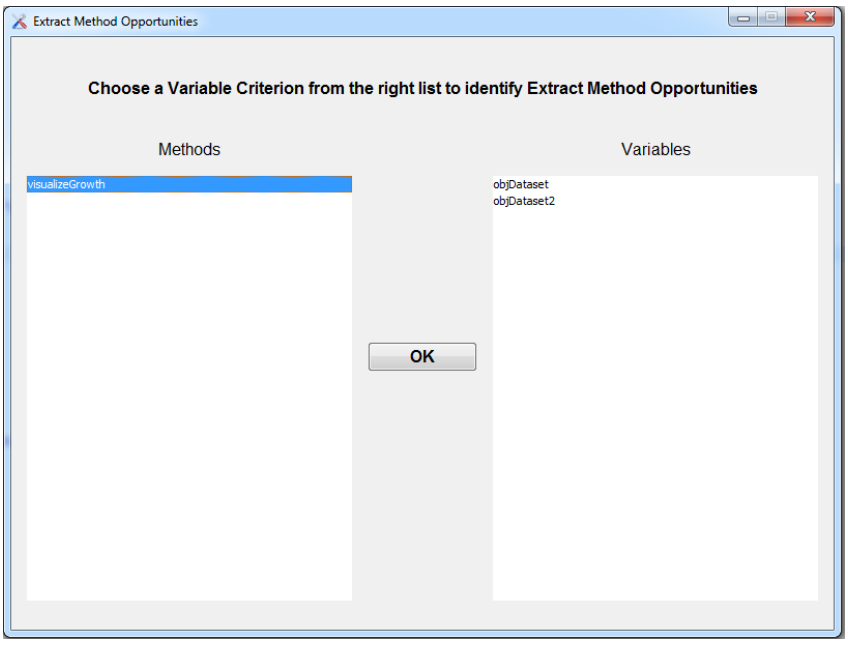


Σχήμα 5.5 Ευκαιρίες Εφαρμογής Inline Temp για την Μέθοδο visualizeGrowth

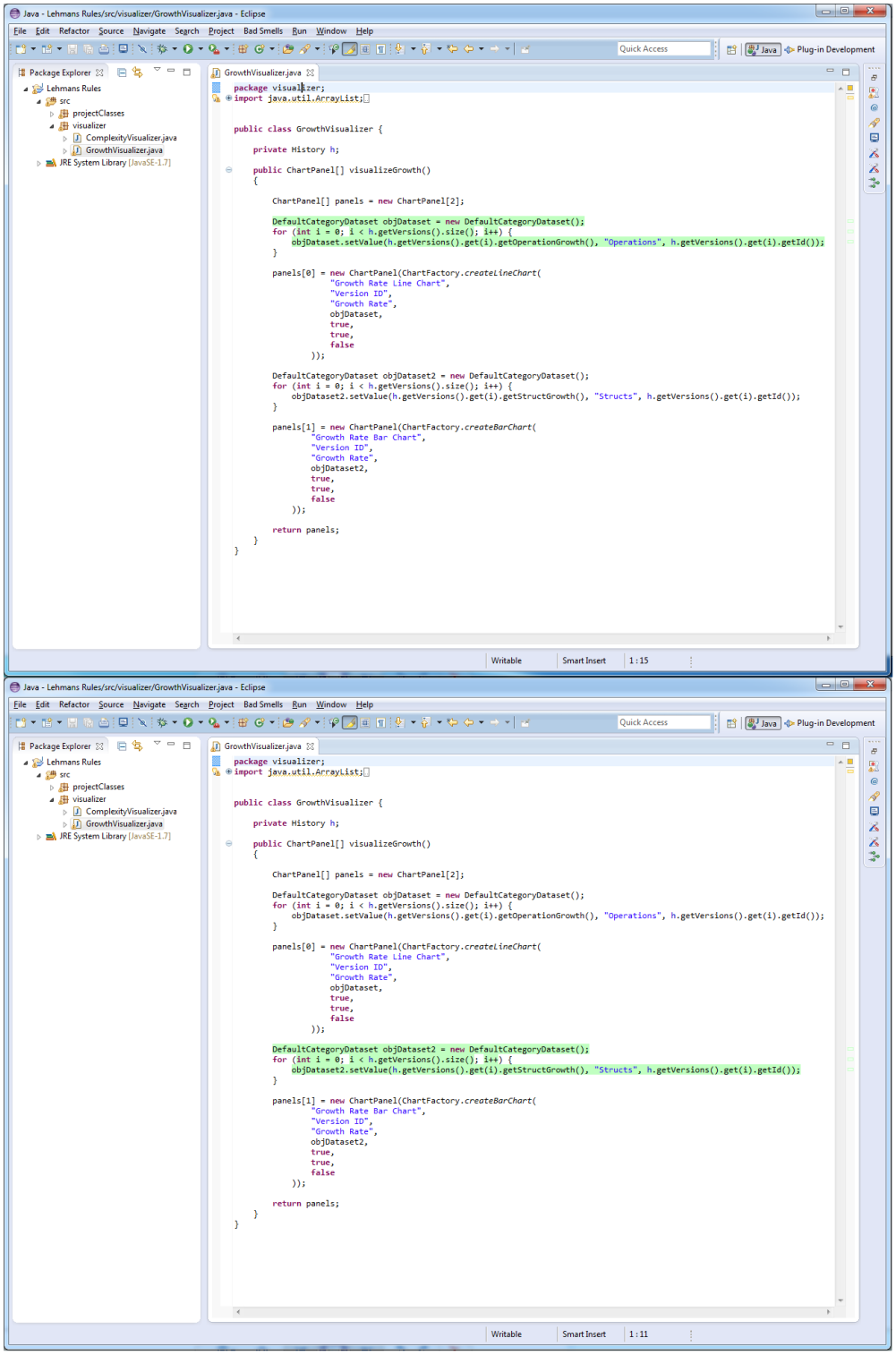


Σχήμα 5.6 Κώδικας της Μεθόδου visualizeGrowth μετά από την Εφαρμογή Inline Temp στις Προσωρινές της Μεταβλητές

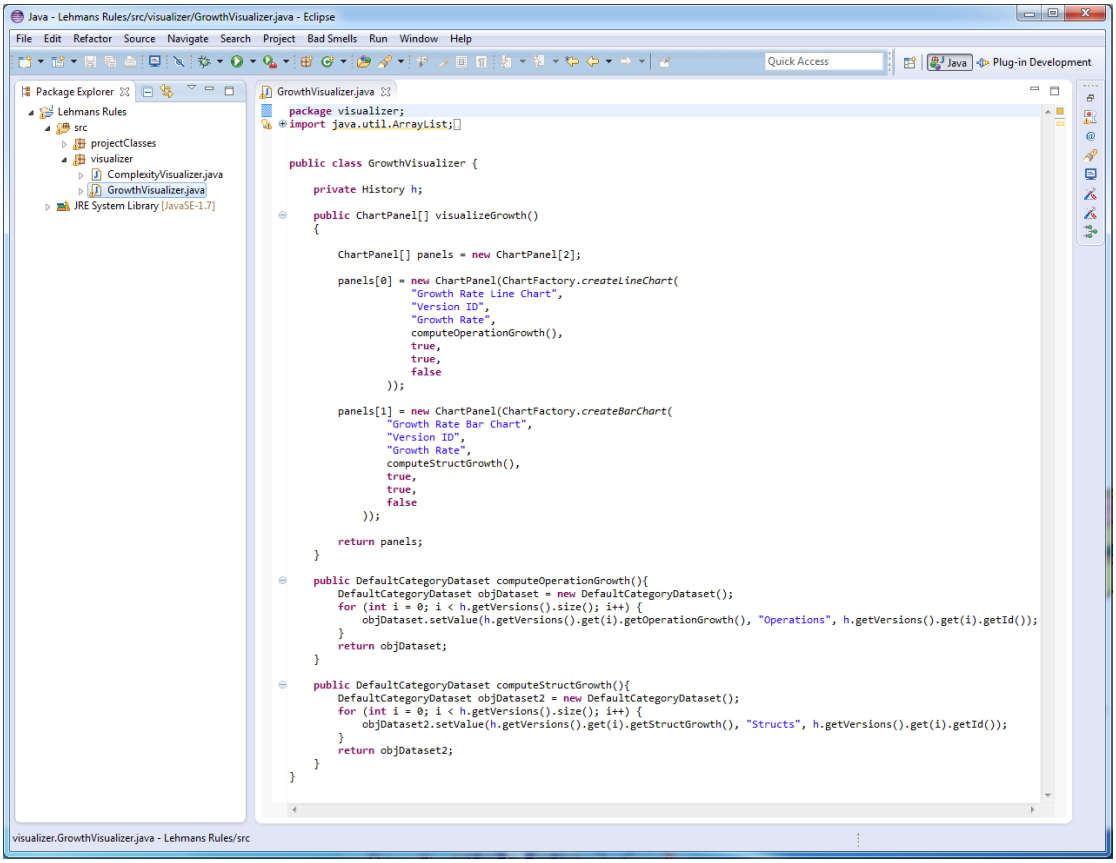
Αφού διαχειριστήκαμε το θέμα των προσωρινών μεταβλητών στον κώδικα, είναι ώρα να επικεντρωθούμε στο Extract Method. Χρησιμοποιώντας την δυνατότητα αυτομάτου εντοπισμού του Refactoring Trip Advisor στον κώδικα εντοπίζονται δύο ευκαιρίες για εξαγωγή σε νέα μέθοδο. Οι ευκαιρίες αυτές φαίνονται στο αναδυόμενο παράθυρο του Σχήματος 5.7. Ο υποψήφιος κώδικας προς εξαγωγή αφορά τον υπολογισμό των μεταβλητών objDataset και objDataset2. Επιλέγοντας μια από τις δυο μεταβλητές και πατώντας το κουμπί "OK" ο Refactoring Trip Advisor επισκιάζει τον υποψήφιο κώδικα στον Eclipse editor. Ο προτεινόμενος κώδικας και στις δύο περιπτώσεις φαίνεται στο Σχήμα 5.8. Εφαρμόζοντας λοιπόν Extract Method μπορούμε να εξάγουμε τα δύο διαφορετικά μπλοκ κώδικα σε δύο νέες μεθόδους. Το αποτέλεσμα αυτής της ανακατασκευής φαίνεται στον κώδικα του Σχήματος 5.9.



Σχήμα 5.7 Ευκαιρίες Εφαρμογής Extract Method για την Μέθοδο visualizeGrowth

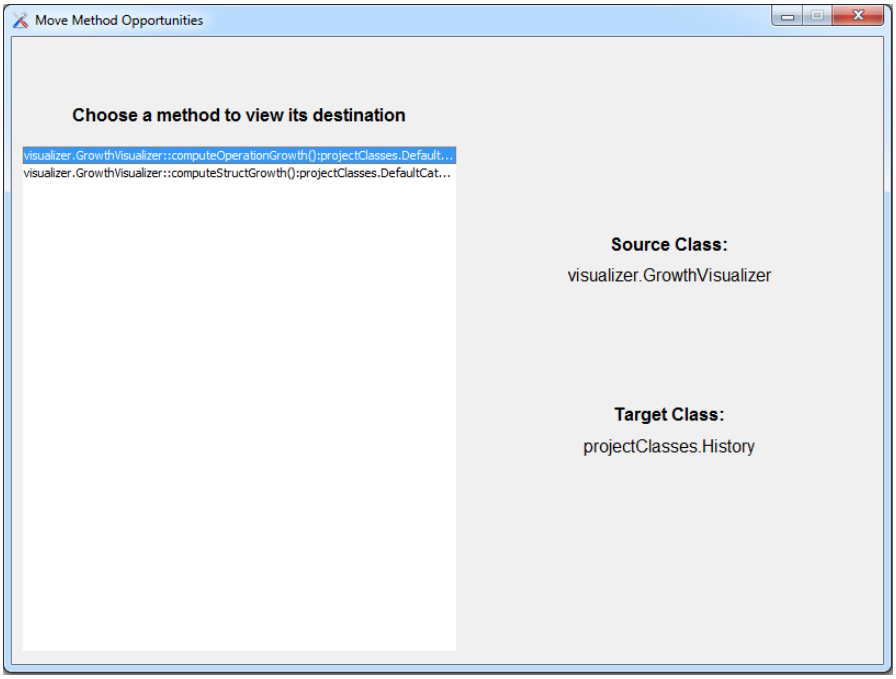


Σχήμα 5.8 Προτεινόμενος Κώδικας προς Εξαγωγή σε νέα Μέθοδο για την Μεταβλητή objDataset (πάνω) και objDataset2 (κάτω)

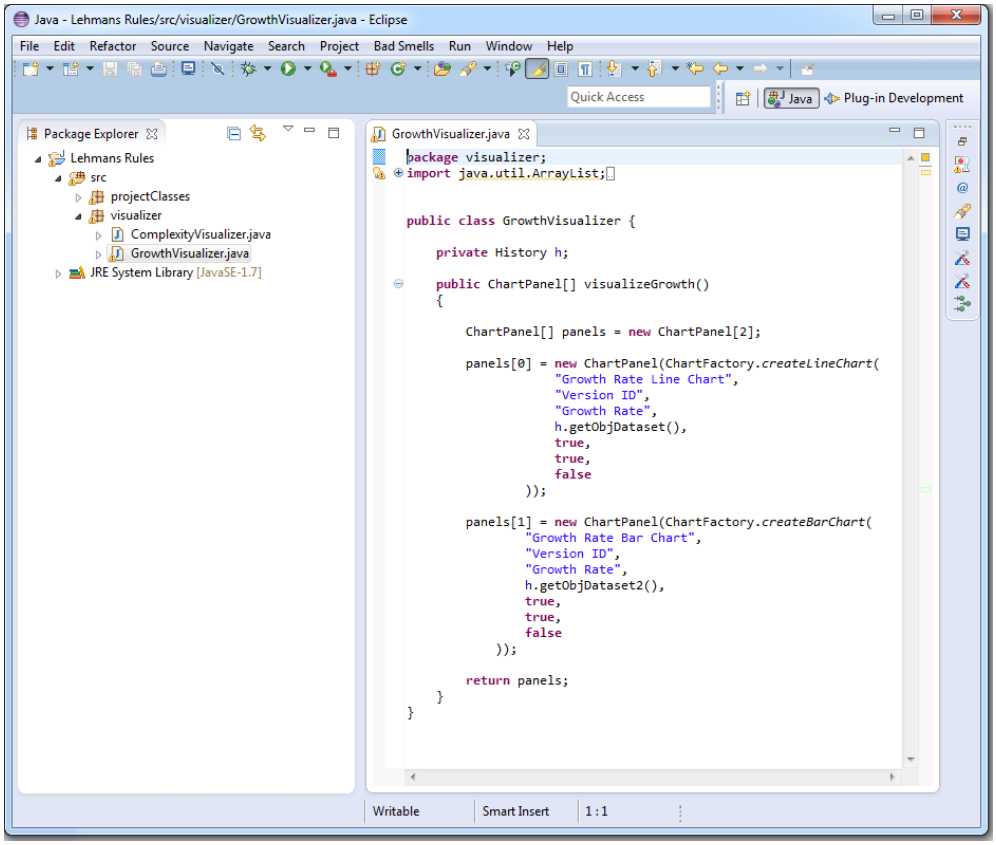


Σχήμα 5.9 Κώδικας της Μεθόδου visualizeGrowth μετά από την Εφαρμογή Extract Method του Προτεινομένου Κώδικα

Έπειτα από την εξαγωγή των δύο νέων μεθόδων ο κώδικας της visualizeGrowth μειώθηκε αρκετά σε μέγεθος και φαίνεται απλοποιημένος. Συμβουλευόμενοι όμως τον Χάρτη Ανακατασκευών παρατηρούμε ότι μετά το Extract Method ο Refactoring Trip Advisor προτείνει να ψάξουμε για πιθανές ευκαιρίες εφαρμογής Remove Assignments to Parameters, Substitute Algorithm και Move Method. Με τον ίδιο τρόπο όπως προηγουμένως αξιοποιώντας την δυνατότητα του αυτόματου εντοπισμού βλέπουμε ότι μόνο για το Move Method υπάρχουν ευκαιρίες στον κώδικα. Οι προτάσεις για μετακινήσεις μεθόδων φαίνονται στο αναδυόμενο παράθυρο του Σχήματος 5.10. Ο Refactoring Trip Advisor προτείνει την μετακίνηση των μεθόδων computeOperationGrowth και computeStructGrowth που εξάγαμε στο προηγούμενο βήμα στην κλάση History. Εφαρμόζοντας τέλος την μετακίνηση αυτών των δύο μεθόδων καταλήγουμε στον κώδικα που του Σχήματος 5.11 όπου παρατηρούμε ότι είναι αρκετά μικρότερος σε μέγεθος σε σχέση με αυτόν που είχαμε αρχικά.



Σχήμα 5.10 Ευκαιρίες Εφαρμογής Move Method για τις Μεθόδους computeOperationGrowth και computeStructGrowth



Σχήμα 5.11 Κώδικας της Μεθόδου visualizeGrowth μετά από την Εφαρμογή Move Method στις Μεθόδους getObjDataset και getObjDataset2

Συνοψίζοντας λοιπόν όλα τα παραπάνω καταλήγουμε στην εξής προτεινόμενη ακολουθία ανακατασκευών από τον Refactoring Trip Advisor για τον κώδικα της κλάσης GrowthVisualizer:

- Εφαρμογή Inline Temp των μεταβλητών versionlist, dops, xaxis, objChart, dstructs, yaxis και objChart2.
- Εφαρμογή Extract Method του μπλοκ υπολογισμού των μεταβλητών objDataset και objDataset2.
- Εφαρμογή Move Method των δύο νέων μεθόδων που δημιουργήθηκαν από το προηγούμενο βήμα στην κλάση History.

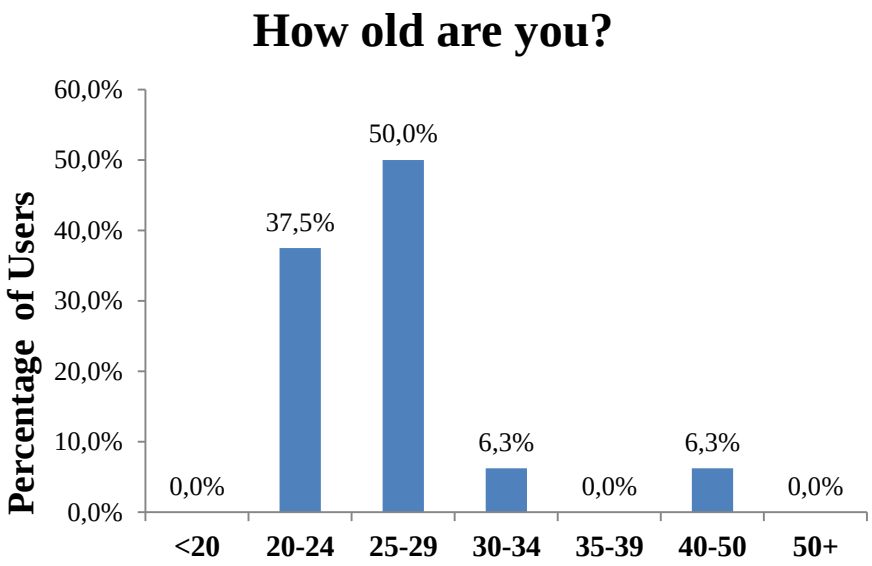
5.4 Αποτελέσματα Αξιολόγησης

Στις προηγούμενες ενότητες αναφέρθηκε η διαδικασία στην οποία υποβλήθηκαν οι χρήστες που πήραν μέρος στην αξιολόγηση της εργασίας και περιγράφηκε βήμα-βήμα η εξαγωγή της προτεινόμενης ακολουθίας ανακατασκευών χρησιμοποιώντας τον Refactoring Trip Advisor. Σε αυτή την ενότητα θα παρουσιαστούν τα αποτελέσματα της αξιολόγησης μαζί με ένα σχολιασμό πάνω σε αυτά.

Μετά την ολοκλήρωση των δύο εργασιών οι χρήστες κατέγραψαν σε ένα ερωτηματολόγιο, ποιες τεχνικές ανακατασκευής εφάρμοσαν και πόσο χρόνο κατανάλωσαν σε κάθε μια από αυτές. Εκτός από αυτό, το ερωτηματολόγιο περιέχει και ερωτήσεις πολλαπλής επιλογής για να γίνει αξιολόγηση τον γνώσεων του χρήστη αλλά και της εμπειρίας του με τη χρήση του Refactoring Trip Advisor. Κάθε μία υποενότητα που ακολουθεί, ασχολείται με διαφορετικό κομμάτι ερωτήσεων του ερωτηματολογίου

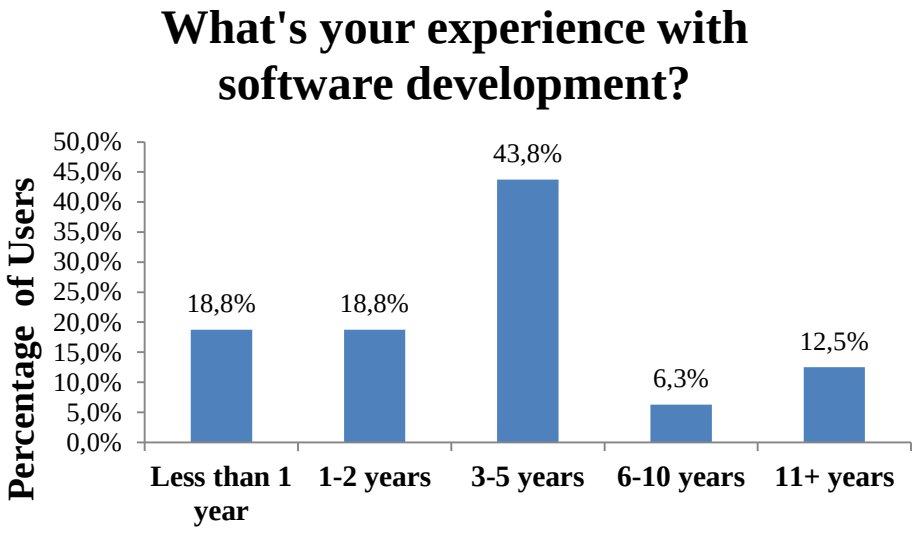
5.4.1 Αξιολόγηση Γνώσεων και Εμπειρίας Χρήστη

Σε αυτή την υποενότητα παρουσιάζονται οι απαντήσεις των ερωτήσεων που εξετάζουν τον γενικό υπόβαθρο του χρήστη και την εμπειρία του με την διαδικασία ανακατασκευής κώδικα στην πράξη. Παρουσιάζονται οι ερωτήσεις με το αντίστοιχο ποσοστό κάθε απάντησης ακολουθούμενα από ένα σύντομο σχολιασμό η κάθε μια.



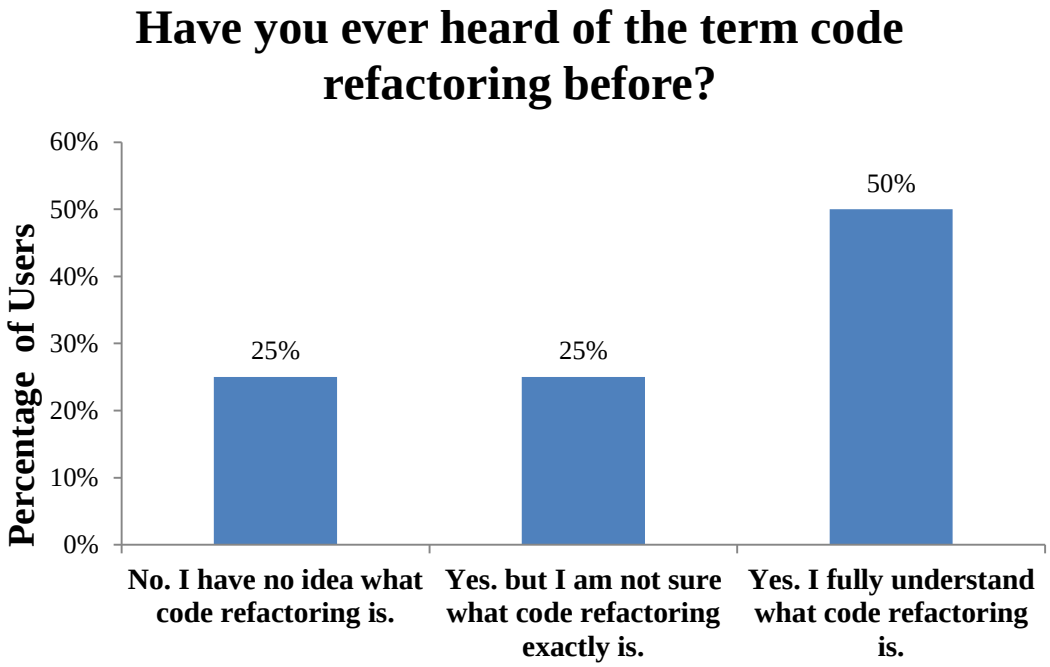
Σχήμα 5.12 Ηλικίες των Χρηστών

Η πλειοψηφία των χρηστών που επιλέχθηκαν να πάρουν μέρος στην αξιολόγηση είναι είτε μεταπτυχιακοί φοιτητές τμημάτων πληροφορικής, είτε προγραμματιστές που αποφοίτησαν από ανάλογα τμήματα. Για αυτό οι ηλικίες τον περισσότερων κυμαίνονται στα 25-29 (50%) και αρκετών στα 20-24 (37.5%) σύμφωνα με το Σχήμα 5.12.



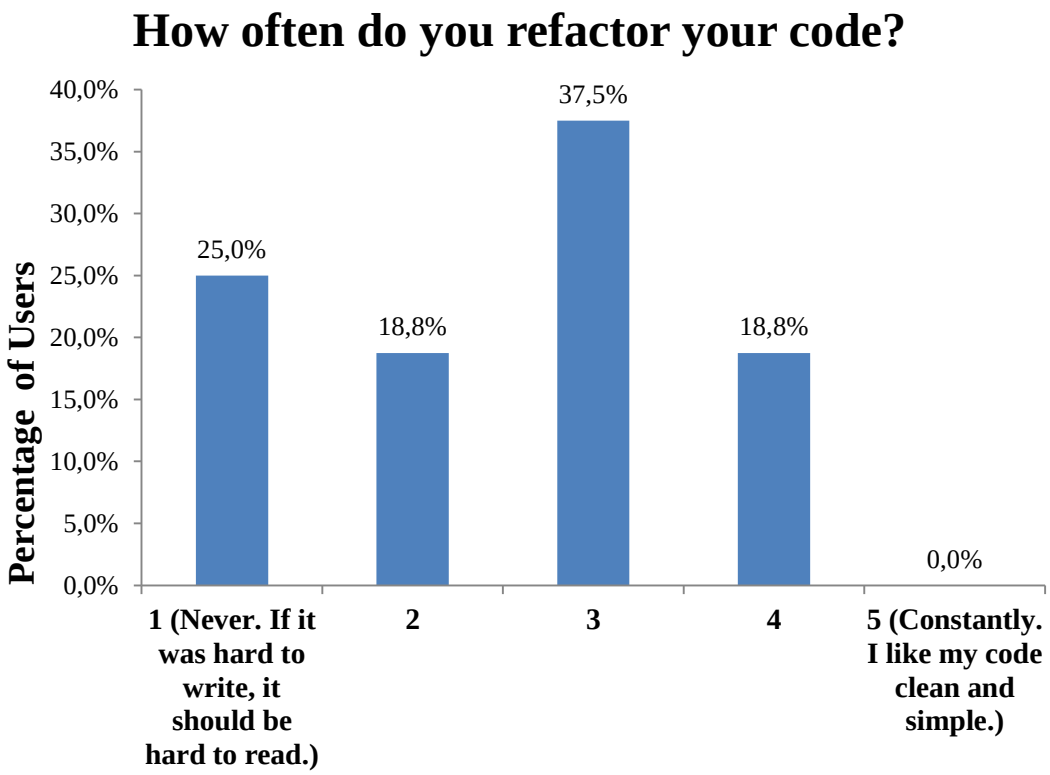
Σχήμα 5.13 Εμπειρία των Χρηστών με την Ανάπτυξη Λογισμικού

Παρατηρώντας το Σχήμα 5.13 μπορούμε να δούμε πως στην αξιολόγηση της εργασίας συμμετείχαν χρήστες με αρκετά διαφορετική εμπειρία πάνω στην ανάπτυξη λογισμικού. Κάποιοι από αυτούς με λιγότερο από 1 χρόνο εμπειρία (18.8%), κάποιοι με εμπειρία τουλάχιστον 11 χρόνων (12.5%), με την πλειοψηφία όμως να βρίσκεται κάπου στη μέση με 3 έως 5 χρόνια εμπειρίας (43.8%).



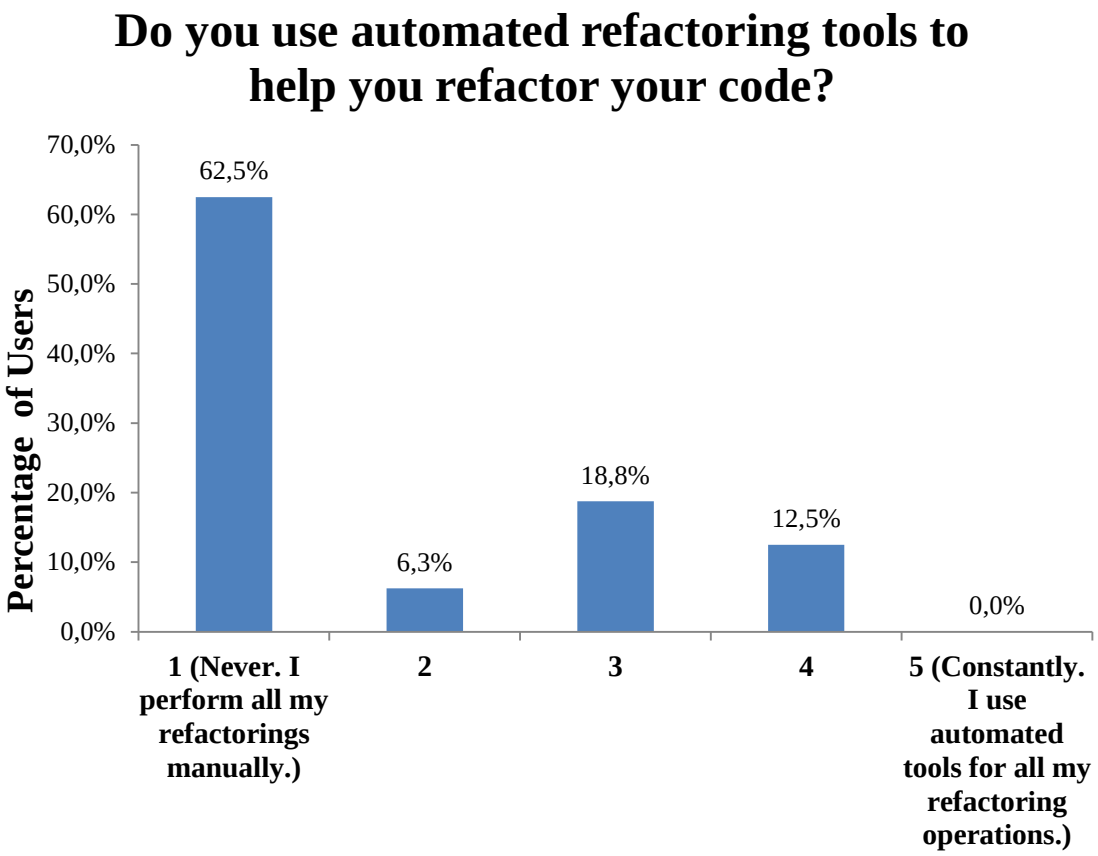
Σχήμα 5.14 Γνώση Χρηστών πάνω στην Ανακατασκευή Κώδικα

Οι χρήστες επίσης ρωτήθηκαν για το πόσο καλά γνωρίζουν τι σημαίνει ανακατασκευή κώδικα. Η πλειοψηφία αυτών (50%) έχει πλήρη κατανόηση του όρου, ένα κομμάτι αυτών (25%) έχει μερική κατανόηση αυτού και τέλος οι υπόλοιποι από αυτούς (25%) δεν ξέρουν τι είναι η ανακατασκευή κώδικα.



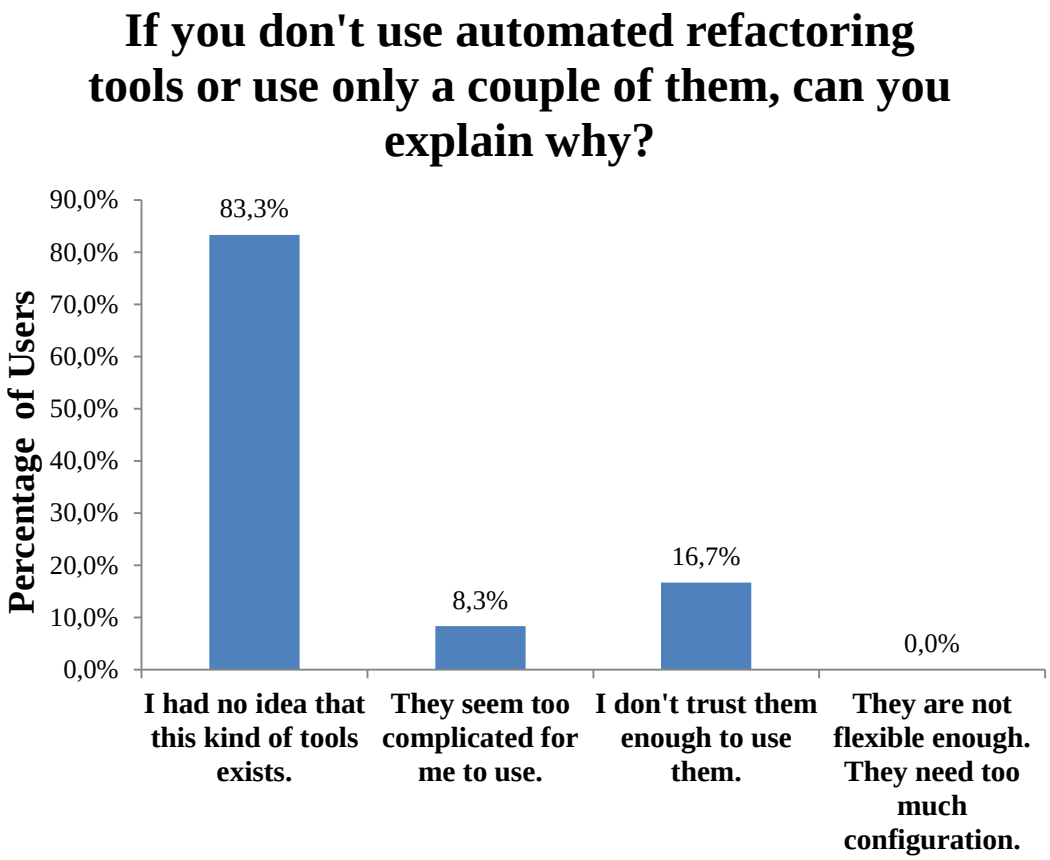
Σχήμα 5.15 Συχνότητα Εφαρμογής Ανακατασκευών από τους Χρήστες

Στους χρήστες ζητήθηκε επίσης να βαθμολογήσουν σε κλίμακα από το 1 έως το 5 για το πόσο συχνά εφαρμόζουν τεχνικές ανακατασκευής στον κώδικα τους όπως φαίνεται στο Σχήμα 5.15. Ένα σημαντικό ποσοστό (25%) απάντησε πως δεν εφαρμόζει ποτέ κάποια τεχνική ανακατασκευής στον κώδικα του ενώ η πλειοψηφία αυτών (37.5%) ανακατασκευάζει τον κώδικα τους με μέτρια συχνότητα.



Σχήμα 5.16 Συχνότητα Χρήσης Αυτοματοποιημένων Εργαλείων κατά την Διαδικασία Ανακατασκευής από τους Χρήστες

Στην ερώτηση του Σχήματος 5.16 φαίνονται τα ποσοστά των απαντήσεων για το πόσο συχνά οι χρηστές αξιοποιούν την βοήθεια κάποιων αυτοματοποιημένων εργαλείων όταν εφαρμόζουν ανακατασκευές στον κώδικα. Η πλειοψηφία των χρηστών (62.5%) δε χρησιμοποιεί κάποιο εργαλείο και προτιμάει να εφαρμόζει τη διαδικασία χειροκίνητα ενώ το υπόλοιπο ποσοστό αξιοποιεί την αυτοματοποιημένη βοήθεια με μέτρια συχνότητα.



Σχήμα 5.17 Λόγοι που οι Χρήστες Αποφεύγουν τη Χρήση Αυτοματοποιημένων Εργαλείων

Οι χρήστες που δε αξιοποιούν τη βοήθεια των αυτοματοποιημένων εργαλείων κατά την διαδικασία ανακατασκευής κλήθηκαν να επιλέξουν τον λόγο όπως φαίνεται στην ερώτηση του Σχήματος 5.17. Η πλειοψηφία αυτών (76.9%) δεν έχουν γνώση της ύπαρξης τέτοιου είδους εργαλείων. Επίσης ένα μικρό ποσοστό (7.7%) πιστεύουν ότι τα εργαλεία αυτά είναι αρκετά πολύπλοκα για να τα χρησιμοποιήσουν, ενώ άλλοι (15.4%) δεν τα εμπιστεύονται καθόλου.

Συνοψίζοντας τις παραπάνω ερωτήσεις μπορούμε να συμπεράνουμε μερικά πράγματα για το δείγμα των χρηστών που πήραν μέρος στην διαδικασία αξιολόγησης. Η πλειοψηφία των χρηστών είναι κάτοχοι διπλώματος πληροφορικής, ηλικίας 25 με 29 χρονών και έχουν 3 με 5 χρόνια εμπειρία πάνω στην ανάπτυξη λογισμικού. Γνωρίζουν τι είναι η ανακατασκευή λογισμικού και την εφαρμόζουν στον κώδικα τους. Ανακατασκευάζουν τον κώδικα τους χειροκίνητα καθώς δεν γνωρίζουν την

ύπαρξη των αυτοματοποιημένων εργαλείων για την υποβοήθηση της διαδικασίας. Αξίζει όμως να σημειωθεί πως πέραν της πλειοψηφίας υπήρχαν και σημαντικά ποσοστά από τις υπόλοιπες απαντήσεις, προσδίδοντας έτσι μια επιθυμητή ποικιλία στο εύρος των χρηστών που καλύπτει αυτή η αξιολόγηση.

5.4.2 Αξιολόγηση του Refactoring Trip Advisor μέσω των Αποτελεσμάτων των Εργασιών

Σε αυτή την υποενότητα παρουσιάζονται τα αποτελέσματα των δύο εργασιών που οι χρήστες εκτέλεσαν στα πλαίσια της αξιολόγησης. Συγκεκριμένα θα παρουσιαστούν κάποια ποσοτικά στοιχεία και θα υπολογιστούν κάποιες μετρικές πάνω στο σύνολο των τεχνικών ανακατασκευής που οι χρήστες εφάρμοσαν και στις δύο περιπτώσεις. Θα σχολιαστούν επίσης και οι χρόνοι εκτέλεσης των δύο εργασιών.

Οι δύο μετρικές που χρησιμοποιήθηκαν σε αυτή τη φάση της αξιολόγησης είναι οι Effectiveness και Efficiency. Ο ορισμός και το τι υπολογίζουν οι μετρικές σε κάθε περίπτωση περιγράφεται παρακάτω.

Effectiveness

Η μετρική Effectiveness χρησιμοποιείται για να υπολογίσει την αποτελεσματικότητα κάποιων οντοτήτων. Συγκεκριμένα, στην πρώτη εργασία μετράει την αποτελεσματικότητα του καταλόγου του Fowler κατά την διαδικασία ανακατασκευής στην πράξη. Δηλαδή μετράται το ποσοστό των τεχνικών που ανήκουν στον κατάλογο από το σύνολο των συνολικών τεχνικών που ο χρήστης εφάρμοσε. Ο τύπος υπολογισμού σε αυτή την περίπτωση είναι:

$$Fowler\ Effectiveness\ = \frac{|FCR \cap DER|}{|DER|}$$

- FCR (Fowler's Catalog Refactorings) = Το σύνολο των τεχνικών ανακατασκευής που περιγράφονται στον κατάλογο του Fowler.
- DER (Developer's Refactorings) = Το σύνολο των τεχνικών ανακατασκευής που ο χρήστης εφάρμοσε στην πράξη στον κώδικα του.

Μεγάλες τιμές αυτής της μετρικής υποδεικνύουν ότι οι ανακατασκευές του καταλόγου ήταν επαρκής για τον χρήστη, ενώ αντίθετα οι μικρές τιμές υποδηλώνουν ότι ο χρήστης εφάρμοσε κυρίως ανακατασκευές που δεν ανήκουν στον κατάλογο.

Στην δεύτερη εργασία, η μετρική Effectiveness χρησιμοποιείται για τον υπολογισμό της αποτελεσματικότητας του Χάρτη Ανακατασκευών που παρέχει ο Refactoring Trip Advisor. Δηλαδή υπολογίζεται το ποσοστό των τεχνικών που προτείνονται με βάση τον Χάρτη Ανακατασκευών από το σύνολο των συνολικών τεχνικών που ο χρήστης εφάρμοσε. Ο τύπος υπολογισμού σε αυτή την περίπτωση είναι:

$$Map\ Effectiveness\ = \frac{|MAR \cap DER|}{|DER|}$$

- MAR (Map's Refactorings) = Το σύνολο των τεχνικών ανακατασκευής που παρουσιάζονται στον Χάρτη Ανακατασκευών.
- DER (Developer's Refactorings) = Το σύνολο των τεχνικών ανακατασκευής που ο χρήστης εφάρμοσε στην πράξη στον κώδικα του.

Μεγάλες τιμές αυτής της μετρικής υποδεικνύουν ότι ο Χάρτης Ανακατασκευών ήταν επαρκής για τον χρήστη, ενώ αντίθετα οι μικρές τιμές υποδηλώνουν ότι ο χρήστης εφάρμοσε κυρίως ανακατασκευές που δεν ανήκουν στον Χάρτη αυτόν.

Efficiency

Η μετρική Efficiency χρησιμοποιείται για τον υπολογισμό της αποδοτικότητας του χρήστη στην διαδικασία ανακατασκευής. Αυτό γίνεται συγκρίνοντας την ακολουθία

ανακατασκευών που ο χρήστης εφάρμοσε στον κώδικα με την βέλτιστη ακολουθία ανακατασκευών για τον κώδικα αυτόν. Συγκεκριμένα, καθώς ο κώδικας των δύο εργασιών είναι σχεδόν ο ίδιος, ως βέλτιστη ακολουθία ανακατασκευών και στις δύο περιπτώσεις χρησιμοποιείται η προτεινόμενη ακολουθία ανακατασκευών που περιγράφηκε στην ενότητα 5.3. Ο τύπος υπολογισμού και για τις δύο περιπτώσεις είναι:

$$Refactoring\ Efficiency\ (Types) = \frac{|SST \cap DER_T|}{|SST|}$$

$$Refactoring\ Efficiency\ (Instances) = \frac{|SSI \cap DER_I|}{|SSI|}$$

- SST (Suggested Sequence of refactoring Types) = Το σύνολο των τύπων των τεχνικών της προτεινόμενης ακολουθίας ανακατασκευών.
- DER_T (Developer's refactoring Types) = Το σύνολο των τύπων των τεχνικών ανακατασκευής που ο χρήστης εφάρμοσε στην πράξη.
- SSI (Suggested Sequence of refactoring Instances) = Το σύνολο των στιγμιότυπων των τεχνικών ανακατασκευής της προτεινόμενης ακολουθίας ανακατασκευών.
- DER_I (Developer's refactoring Instances) = Το σύνολο των στιγμιότυπων των τεχνικών ανακατασκευής που χρήστης εφάρμοσε στην πράξη.

Τα σύνολα SST και SSI είναι στατικά καθώς η προτεινόμενη ακολουθία ανακατασκευών είναι συγκεκριμένη και ο υπολογισμός τους φαίνεται παρακάτω:

- SST = (Inline Temp, Extract Method, Move Method)
- |SST| = 3
- SSI = (Inline Temp[versionlist], Inline Temp[dops], Inline Temp[xaxis], Inline Temp[objChart], Inline Temp[dstructs], Inline Temp[yaxis], Inline Temp[objChart2], Extract Method[μπλοκ υπολογισμού της objDataset],

Extract Method[μπλοκ υπολογισμού της objDataset2], Move Method[την πρώτη εξαγόμενη μέθοδο], Move Method[την δεύτερη εξαγόμενη μέθοδο])

- |SSI| = 11

Αφού ορίστηκαν οι μετρικές που θα υπολογιστούν σε αυτό το κομμάτι της αξιολόγησης, στη συνέχεια θα παρουσιαστούν τα αποτελέσματα αυτής. Τα αποτελέσματα της πρώτης εργασίας που οι χρήστες εκτέλεσαν περιέχονται στον Πίνακα 5.1 ενώ τα αποτελέσματα της δεύτερης στον Πίνακα 5.2. Στους πίνακες παρουσιάζονται η διακύμανση των τιμών ανά τεταρτημόριο (quartile) και ο μέσος όρος για κάθε μετρική.

Πίνακας 5.1 Αποτελέσματα Μετρικών για την Πρώτη Εργασία της Αξιολόγησης

	Fowler Effectiveness	**Refactoring Efficiency (Types)**	**Refactoring Efficiency (Instances)**
Min	0.75	0.33	0.18
Q1	1	0.33	0.18
Mid	1	0.33	0.27
Q3	1	0.66	0.47
Max	1	1	0.9
Avg	0.98	0.49	0.36

Πίνακας 5.2 Αποτελέσματα Μετρικών για την Δεύτερη Εργασία της Αξιολόγησης

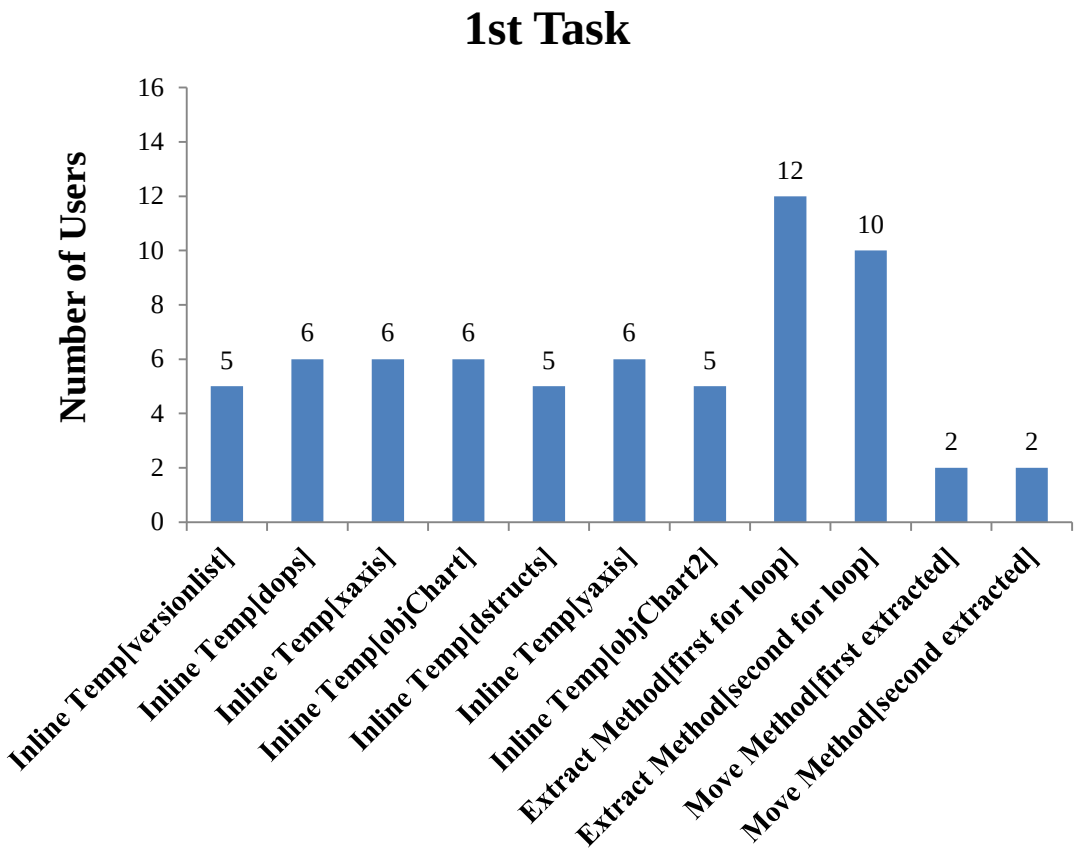
	Map Effectiveness	**Refactoring Efficiency (Types)**	**Refactoring Efficiency (Instances)**
Min	1	0.33	0.18
Q1	1	0.66	0.45
Mid	1	0.66	0.81
Q3	1	1	1
Max	1	1	1
Avg	1	0.76	0.71

Αρχικά εστιάζουμε την προσοχή μας στη μετρική Effectiveness η οποία μετρά την αποτελεσματικότητα του κατάλογου του Fowler στην πρώτη εργασία και του Χάρτη Ανακατασκευών στη δεύτερη. Η μέγιστη δυνατή τιμή της μετρικής βάση τύπου είναι 1. Αυτό συμβαίνει στην περίπτωση που όλο το σύνολο των ανακατασκευών που ο χρήστης εφάρμοσε στην πράξη ανήκει στον κατάλογο του Fowler (για την μετρική Fowler Effectiveness) ή στον Χάρτη Ανακατασκευών του Refactoring Trip Advisor (για την μετρική Map Effectiveness). Παρατηρώντας τους δύο πίνακες μπορεί κανείς

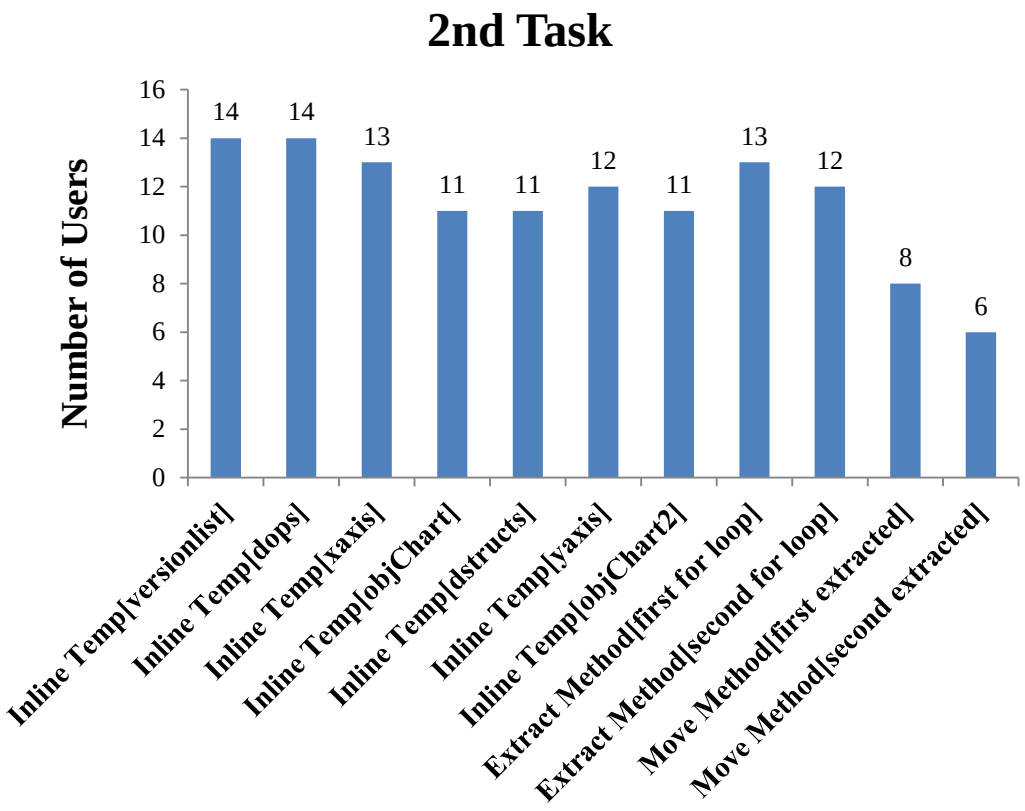
να δει ότι στην πρώτη εργασία υπάρχει ένα μικρό ποσοστό χρηστών που εφάρμοσαν κάποια τεχνική που δεν περιλαμβάνεται στον κατάλογο του Fowler, ενώ στη δεύτερη εργασία φαίνεται πως ο Χάρτης Ανακατασκευών ήταν πλήρης για την κάλυψη των αναγκών τους. Συγκεκριμένα, στην πρώτη εργασία, ένας από το σύνολο των χρηστών εφάρμοσε μια ανακατασκευή στον κώδικα που δεν ανήκει στον κατάλογο του Fowler (μετατροπή των μεταβλητών objDataset και objDataset2 σε πίνακα με όνομα objDatasets). Παρόλα αυτά δεν επανέλαβε αυτή την ανακατασκευή στη δεύτερη εργασία και εφάρμοσε μόνο ανακατασκευές που ο Refactoring Trip Advisor πρότεινε.

Έπειτα σειρά έχει η μετρική Efficiency η οποία χρησιμοποιείται για την μέτρηση της αποδοτικότητας του χρήστη στην διαδικασία ανακατασκευής. Για αυτή τη μετρική η μέγιστη δυνατή τιμή βάση τύπου είναι 1 επίσης. Αυτό συμβαίνει στην περίπτωση ο χρήστης εφαρμόσει όλο το σύνολο των τύπων (Types) και των στιγμιότυπων (Instances) που περιγράφονται στην προτεινόμενη ακολουθία ανακατασκευών. Η μετρική αυτή ουσιαστικά δίνει μια εικόνα για το πόσο κοντά στο βέλτιστο είναι αυτό που ο χρήστης έκανε στην πράξη. Από τα αποτελέσματα στους πίνακες έχουμε ότι η κατά μέσο όρο απόδοση των χρηστών στην πρώτη εργασία είναι 48% για τους τύπους και 36% για τα στιγμιότυπα ενώ για τη δεύτερη εργασία 77% και 74% αντίστοιχα. Παρατηρούμε λοιπόν πως χρησιμοποιώντας τον Refactoring Trip Advisor οι χρήστες εφάρμοσαν περισσότερες τεχνικές στον κώδικα τους και βελτίωσαν την απόδοση τους στην διαδικασία ανακατασκευής. Συγκεκριμένα, στα γραφήματα των Σχημάτων 5.18 - 5.19 και 5.20 – 5.21 παρουσιάζεται ο αριθμός των χρηστών που εκτέλεσαν το κάθε στιγμιότυπο και τον κάθε τύπο ανακατασκευής αντίστοιχα, της προτεινόμενης ακολουθίας ανακατασκευών. Συγκρίνοντας τα 5.18 και 5.19 μεταξύ τους παρατηρούμε ότι ο αριθμός των χρηστών που εφάρμοσαν το κάθε στιγμιότυπο τουλάχιστον διπλασιάστηκε στην δεύτερη εργασία για την πλειοψηφία από αυτά. Επίσης εξετάζοντας τα 5.20 και 5.21 παρατηρούμε πως μόνο ένας χρήστης εφάρμοσε Move Method στην πρώτη εργασία, ενώ στη δεύτερη επέλεξαν να το εφαρμόσουν τουλάχιστον οι μισοί από αυτούς. Γενικά συμπεραίνουμε ότι στην δεύτερη εργασία καλύφθηκε ένα σημαντικό ποσοστό τεχνικών ανακατασκευής της προτεινόμενης ακολουθίας, ενώ στην πρώτη εργασία η πλειοψηφία των χρηστών εφάρμοσε κυρίως Extract Method στον κώδικα των δύο for βρόγχων. Επίσης μπορούμε να δούμε ότι η

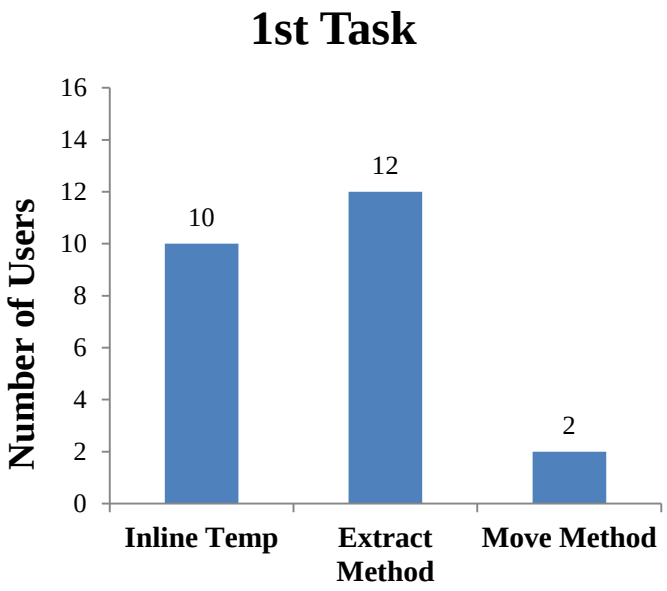
πλειοψηφία των χρηστών δεν κατάφερε να εντοπίσει την ευκαιρία μετακίνησης των δύο εξαγόμενων μεθόδων χωρίς την βοήθεια του Refactoring Trip Advisor.



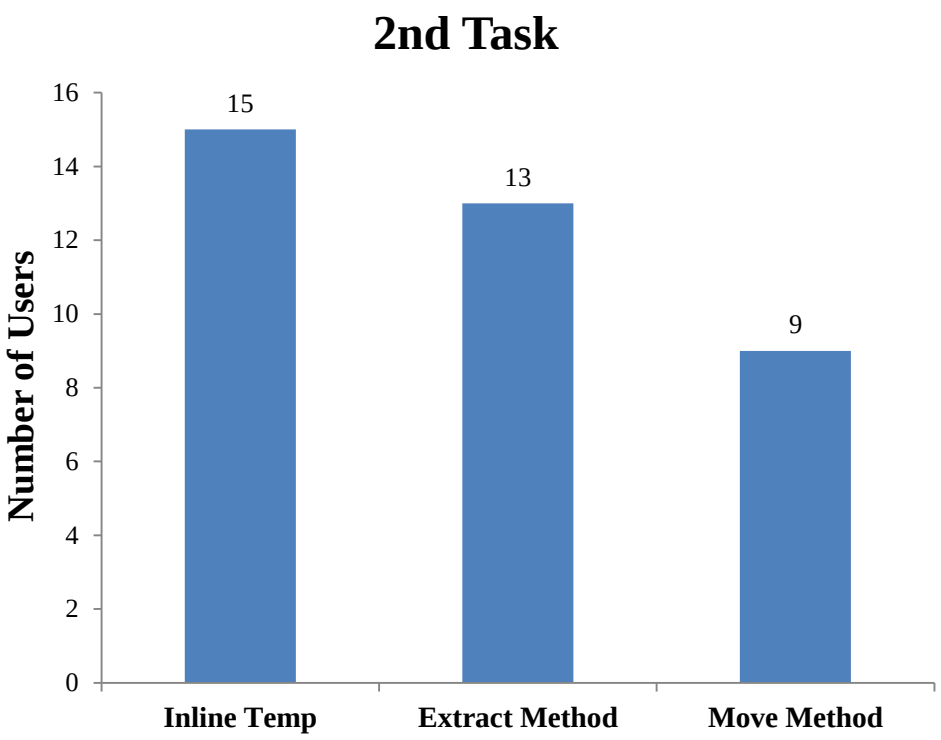
Σχήμα 5.18 Αριθμός Χρηστών για κάθε Στιγμιότυπο Ανακατασκευής της Πρώτης Εργασίας



Σχήμα 5.19 Αριθμός Χρηστών για κάθε Στιγμιότυπο Ανακατασκευής της Δεύτερης Εργασίας



Σχήμα 5.20 Αριθμός Χρηστών για κάθε Τύπο Ανακατασκευής της Πρώτης Εργασίας



Σχήμα 5.21 Αριθμός Χρηστών για κάθε Τύπο Ανακατασκευής της Δεύτερης Εργασίας

Μετά την παρουσίαση των αποτελεσμάτων των χρηστών σχετικά με το ποιες τεχνικές ανακατασκευής εφάρμοσαν σε κάθε εργασία, σε αυτό το σημείο θα σχολιαστούν οι χρόνοι εκτέλεσης αυτών. Στον Πίνακα 5.3 παρουσιάζεται η διακύμανση του χρόνου (σε λεπτά) ανά τεταρτημόριο (quartile) και ο μέσος χρόνος εκτέλεσης των δύο εργασιών.

Πίνακας 5.3 Αποτελέσματα Χρόνων Εκτέλεσης των Δύο Εργασιών

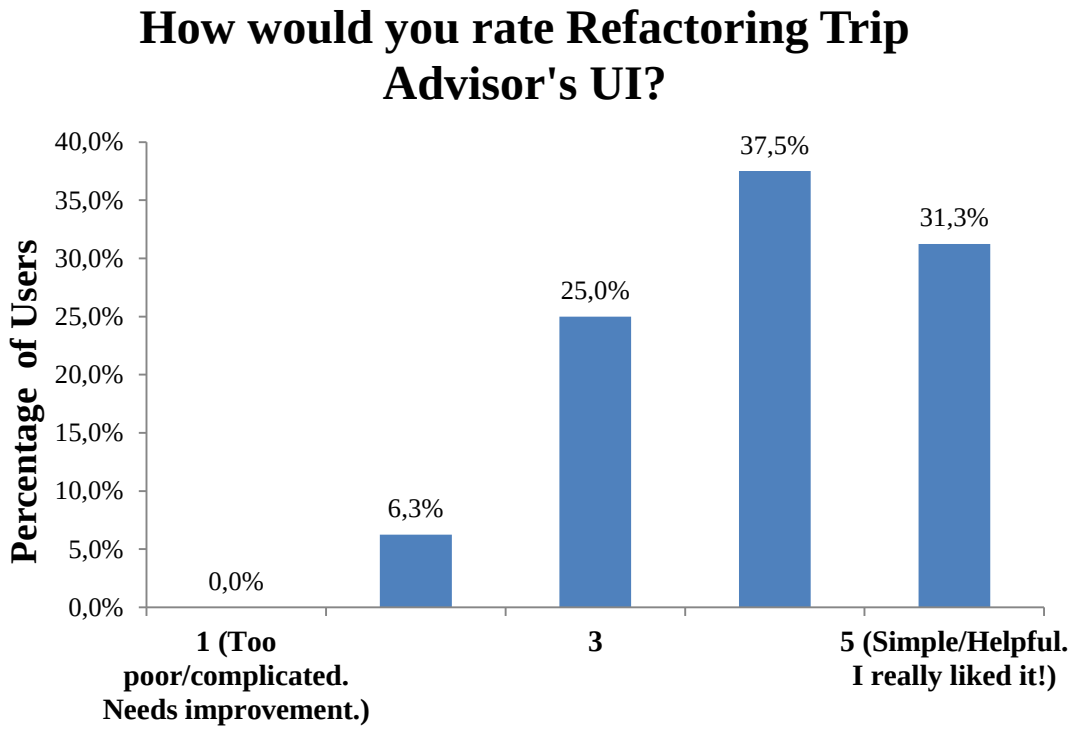
	1st Task Time (minutes)	**2nd Task Time (minutes)**
Min	2	3
Q1	10	7.25
Mid	10	12.5
Q3	16.25	21.25
Max	60	90
Avg	17.81	17.93

Στον πίνακα παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης και των δύο εργασιών είναι περίπου 18 λεπτά. Παρόλα αυτά οι τιμές όλων των τεταρτημορίων εκτός του πρώτου

(Q1), είναι μεγαλύτερες για την δεύτερη εργασία. Συγκεκριμένα, από το σύνολο των 16 χρηστών, 6 από αυτούς εκτέλεσαν την δεύτερη εργασία σε λιγότερο χρόνο από την πρώτη, 7 από αυτούς σε περισσότερο και 3 από αυτούς παρουσίασαν τον ίδιο χρόνο εκτέλεσης και στις δύο. Συνεπώς, η επιρροή του Refactoring Trip Advisor πάνω στον χρόνο εκτέλεσης της διαδικασίας ανακατασκευής ποικίλει από χρήστη σε χρήστη.

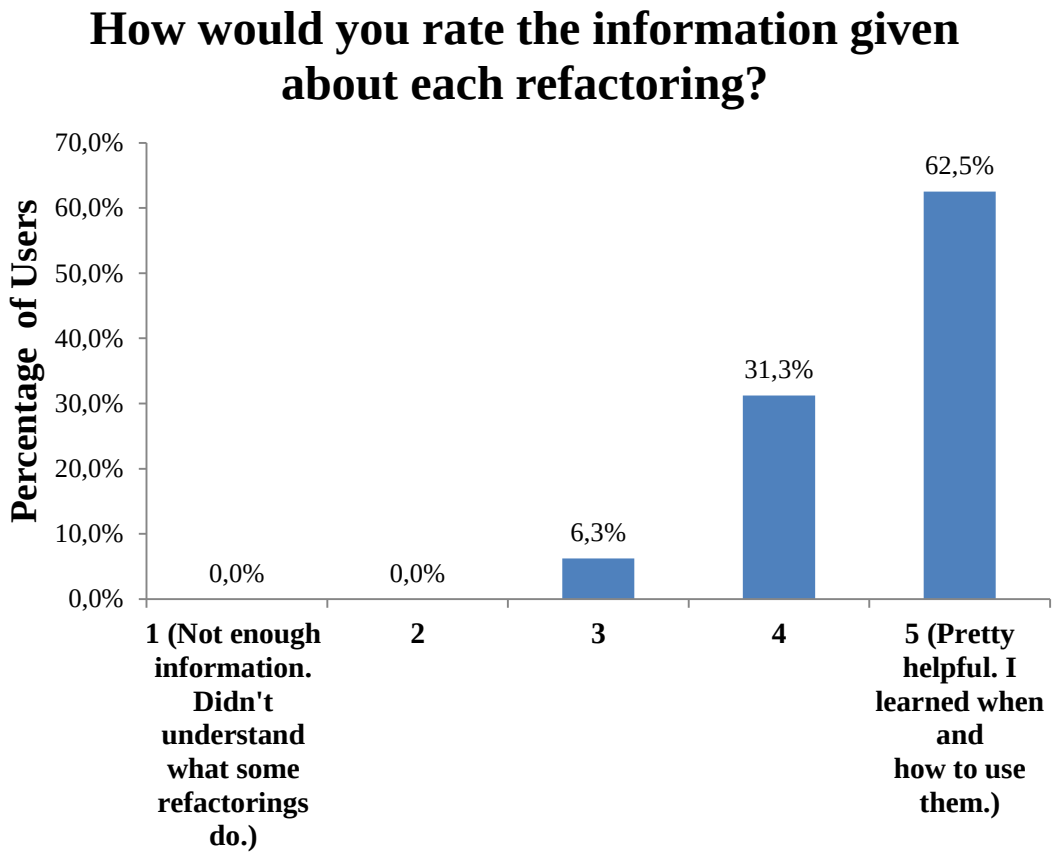
5.4.3 Αξιολόγηση του Refactoring Trip Advisor από τους Χρήστες

Σε αυτή την υποενότητα παρουσιάζονται οι απαντήσεις των χρηστών πάνω σε ερωτήσεις για την ποιότητα συγκεκριμένων χαρακτηριστικών του Refactoring Trip Advisor. Όλες οι ερωτήσεις απαιτούν από το χρήστη την βαθμολόγηση ενός χαρακτηριστικού σε κλίμακα από το 1 έως το 5. Στο ίδιο μοτίβο με την υποενότητα 5.4.1, παρουσιάζονται τα ποσοστά των απαντήσεων με το ανάλογο γράφημα πίτας, ακολουθούμενα από ένα σύντομο σχολιασμό.



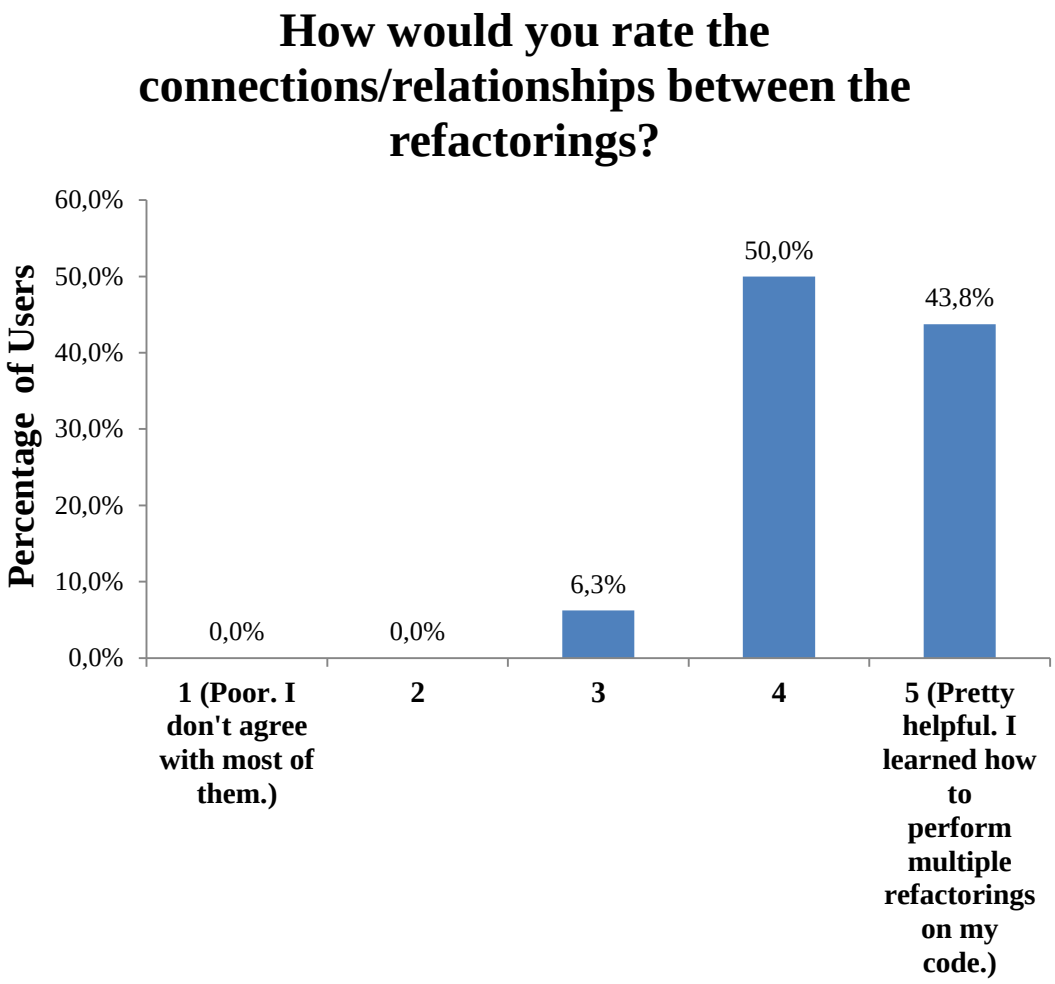
Σχήμα 5.22 Βαθμολόγηση της Διεπαφής Χρήστη του Refactoring Trip Advisor

Αρχικά οι χρήστες αξιολόγησαν τη διεπαφή χρήστη του Refactoring Trip Advisor. Η πλειοψηφία των απαντήσεων είναι θετική και συγκεντρώνεται στο 4 (37.5%) και 5 (31.3%), ενώ ένα σημαντικό ποσοστό (25%) πιστεύει πως η ποιότητα της διεπαφής χρήστη είναι μέτρια.



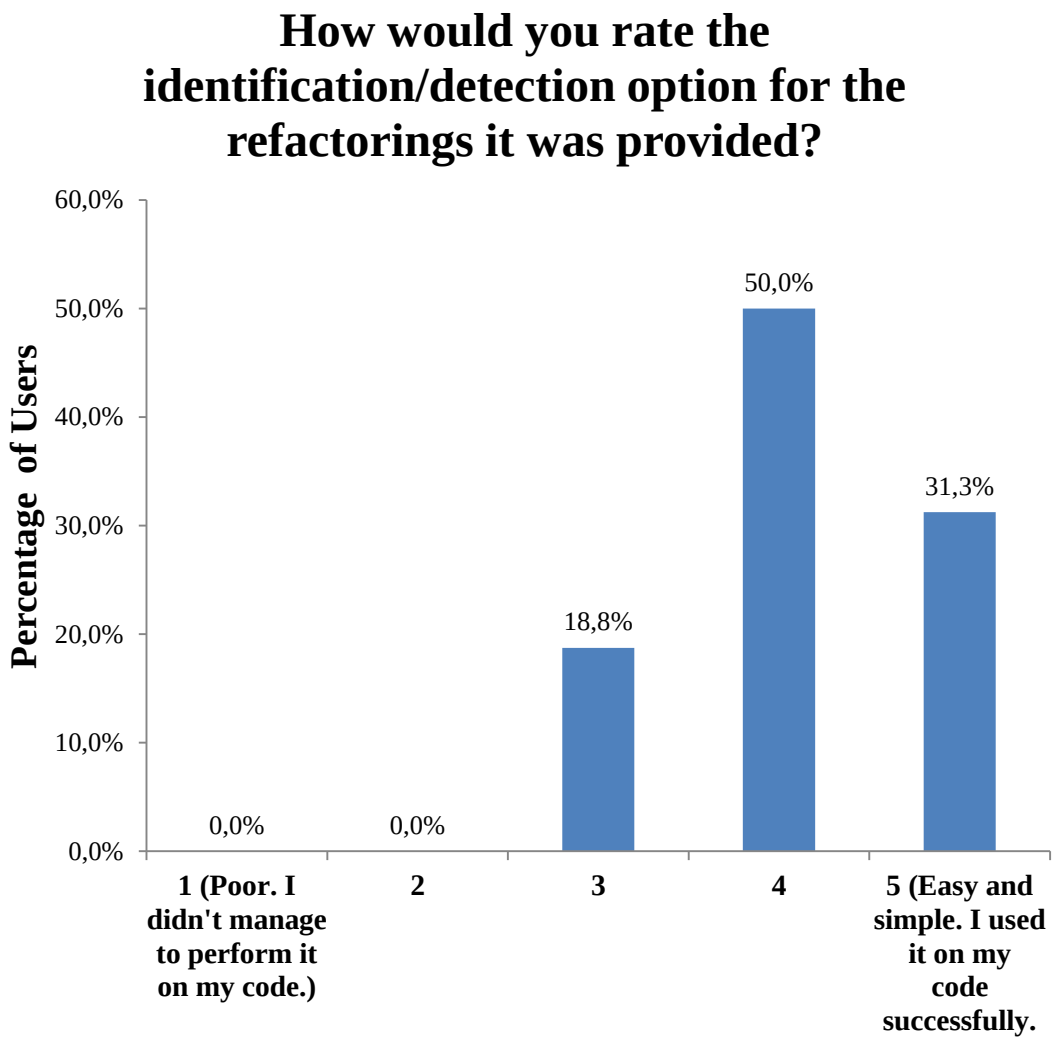
Σχήμα 5.23 Βαθμολόγηση των Πληροφοριών που Προσφέρονται για κάθε Τεχνική Ανακατασκευής

Στη συνέχεια οι χρήστες βαθμολόγησαν τις πληροφορίες που ο Refactoring Trip Advisor παρέχει για κάθε τεχνική ανακατασκευής. Συγκεκριμένα τις πληροφορίες που εμφανίζονται στο πάνελ των τριών διαφανειών (βλέπε Σχήμα 4.7, 4.8 και 4.9) για κάθε τεχνική ανακατασκευής. Το μεγαλύτερο ποσοστό (62.5%) των χρηστών πιστεύει πως οι πληροφορίες ήταν αρκετά βοηθητικές και ένα σημαντικό ποσοστό (31.3%) έχει θετική γνώμη επίσης.



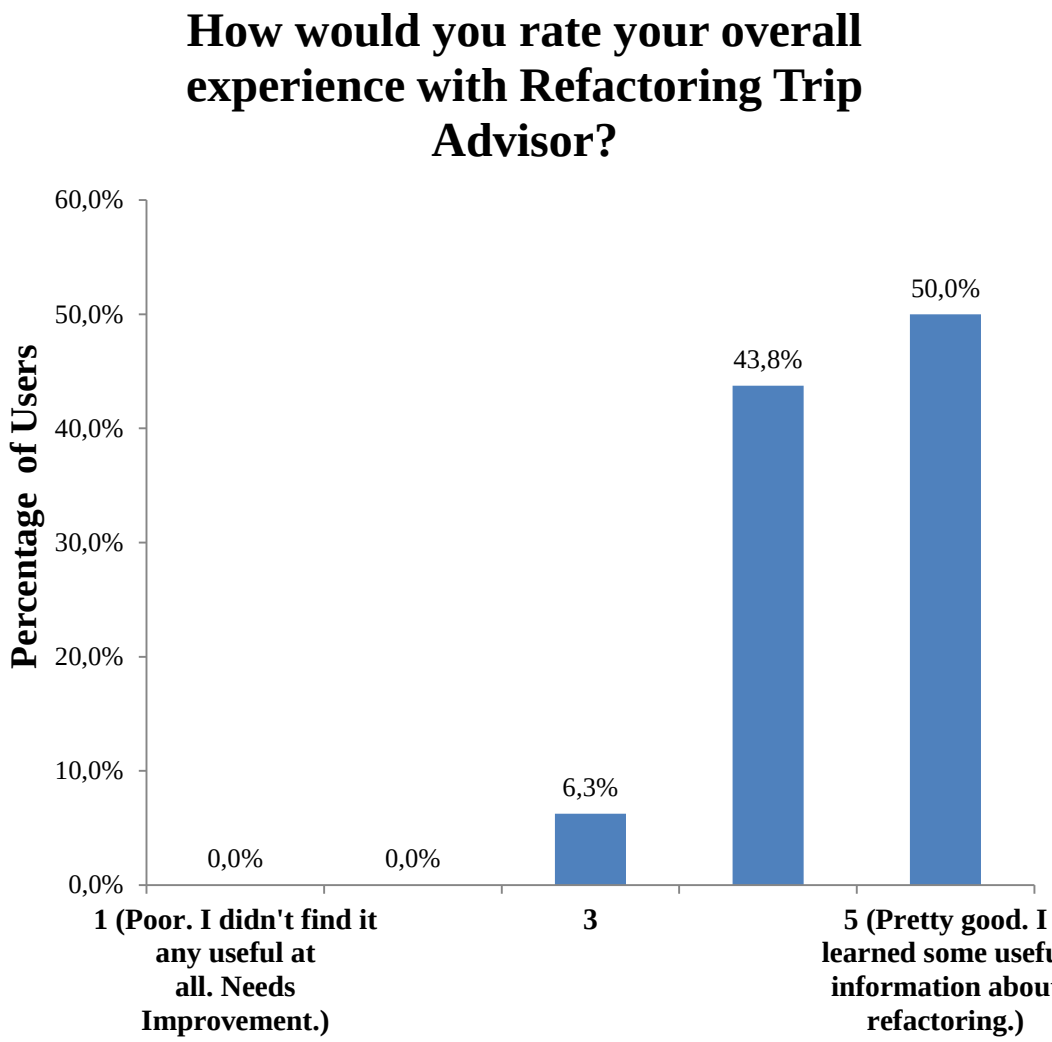
Σχήμα 5.24 Βαθμολόγηση των Συσχετίσεων μεταξύ των Τεχνικών Ανακατασκευής

Το επόμενο χαρακτηριστικό προς αξιολόγηση είναι οι συσχετίσεις ανάμεσα των διάφορων τεχνικών ανακατασκευής. Δηλαδή ουσιαστικά η αξιολόγηση των Χαρτών ανακατασκευής που παρουσιάζονται σε αυτή την εργασία. Και σε αυτή την ερώτηση οι απαντήσεις ήταν θετικές καθώς πολλοί χρήστες πιστεύουν ότι ο Χάρτης τους βοήθησε στην διαδικασία ανακατασκευής. Συγκεκριμένα οι απαντήσεις συγκεντρώνονται στο 4 (50%) και το 5 (43.8%).



Σχήμα 5.25 Βαθμολόγηση των Δυνατοτήτων Αυτόματου Εντοπισμού

Μετά την βαθμολόγηση του Χάρτη Ανακατασκευών σειρά έχει η δυνατότητα του αυτόματου εντοπισμού ευκαιριών εφαρμογής ανακατασκευών στον κώδικα του χρήστη. Η αξιολόγηση γενικά είναι θετική καθώς αρκετοί χρήστες αξιοποίησαν την βοήθεια του αυτόματου εντοπισμού στον κώδικα τους με επιτυχία. Συγκεκριμένα, οι απαντήσεις συγκεντρώνονται στο 3 (18.8%), 4 (50%) και το 5 (31.3%).



Σχήμα 5.26 Βαθμολόγηση της Εμπειρίας με τη Χρήση του Refactoring Trip Advisor Συνολικά

Τέλος οι χρήστες βαθμολόγησαν γενικά την εμπειρία που είχαν χρησιμοποιώντας τον Refactoring Trip Advisor στα πλαίσια αυτής της αξιολόγησης. Η πλειοψηφία των χρηστών (50%) είχε μια καλή εμπειρία χρησιμοποιώντας το εργαλείο και ένα αντίστοιχο ποσοστό (43.8%) απάντησε επίσης θετικά.

Συνοψίζοντας τα παραπάνω παρατηρούμε ότι όλες οι απαντήσεις των χρηστών είναι θετικές κατά γενική ομολογία. Πιστεύουν πως η διεπαφή χρήστη του εργαλείου καθώς και οι πληροφορίες που αυτό παρέχει για κάθε τεχνική ανακατασκευής είναι αρκετά βοηθητικές. Θεωρούν πως οι συσχετίσεις που παρουσιάζονται στους Χάρτες ανακατασκευής σε συνδυασμό με την δυνατότητα του αυτόματου εντοπισμού, τους

βοήθησαν αρκετά στην διαδικασία της ανακατασκευής κώδικα. Η όλη εμπειρία με την χρήση του Refactoring Trip Advisor κατά την διαδικασία αξιολόγησης ήταν γενικά θετική. Παρόλα αυτά υπήρχαν και σχόλια από μερικούς χρήστες οι οποίοι πρότειναν την επιλογή της αυξομείωσης του μεγέθους των παραθύρων, καθώς αυτό δεν είναι υλοποιημένο στην διεπαφή του Refactoring Trip Advisor.

5.5 Εγκυρότητα Αποτελεσμάτων

Όπως σε κάθε πειραματική έρευνα, έτσι και εδώ μπορεί να προκύψουν θέματα με την εγκυρότητα των αποτελεσμάτων. Το γεγονός ότι το δείγμα των χρηστών είναι μικρό, μπορεί να αποτελέσει κίνδυνο στην εξωτερική εγκυρότητα της αξιολόγησης. Αυτό προσπαθήσαμε να το βελτιώσουμε διαλέγοντας ένα μέρος του δείγματος των χρηστών τυχαία μέσω διαδικτυακών ερευνητικών φόρουμ (www.surveypolice.com, www.thestudentroom.co.uk και http://www.postgraduateforum.com). Σχετικά με την εσωτερική εγκυρότητα, για να αποφύγουμε την πιθανή επίδραση της συνεργασίας στην εκτέλεση των εργασιών, τονίσαμε την ατομική εκτέλεση αυτών στους χρήστες. Επίσης υπήρχε μια μικρή διαφοροποίηση μεταξύ του κώδικα των δύο εργασιών για να μειώσουμε την επιρροή της επαναληπτικής διαδικασίας στα αποτελέσματα.

ΚΕΦΑΛΑΙΟ 6. ΕΠΙΛΟΓΟΣ

6.1 Ανακεφαλαίωση και Συμπεράσματα

Η ενότητα αυτή του τελευταίου κεφαλαίου της εργασίας περιλαμβάνει μια σύντομη επανάληψη των προηγούμενων μαζί με κάποια τελευταία σχόλια πάνω σε αυτά. Το πρόβλημα που αυτή η εργασία προσπαθεί να επιλύσει αφορά τη διαδικασία με την οποία οι χρήστες ανακατασκευάζουν τον κώδικα τους. Στην πράξη το σύνολο των διαθέσιμων τεχνικών ανακατασκευής είναι αρκετά μεγάλο και έτσι δεν γίνεται πλήρης αξιοποίηση αυτού. Αυτό σε συνδυασμό με το γεγονός το ότι οι χρήστες επιλέγουν να ανακατασκευάσουν τον κώδικα τους χειροκίνητα, χωρίς την βοήθεια αυτοματοποιημένων εργαλείων καταδεικνύει πως υπάρχουν περιθώρια βελτίωσης της διαδικασίας ανακατασκευής. Στην εργασία αυτή παρουσιάστηκε μια νέα μέθοδος ανακατασκευής κώδικα βασισμένη στον Χάρτη Ανακατασκευών πάνω στον οποίο ο χρήστης πλοηγείται εφαρμόζοντας ανακατασκευές στον κώδικα του με συγκροτημένο τρόπο. Ο Χάρτης Ανακατασκευών απαρτίζεται από τις 68 τεχνικές ανακατασκευής που περιγράφονται στον κατάλογο του Fowler, οι οποίες συνδέονται μεταξύ τους μέσω τριών ειδών συσχετίσεων. Μελετώντας τις συσχετίσεις μεταξύ αυτών στον Χάρτη Ανακατασκευών διαπιστώθηκε πως οι Extract Method και Move Method είναι από τις πιο απλές τεχνικές με την μεγαλύτερη πιθανότητα εφαρμογής στην διαδικασία ανακατασκευής, ενώ οι τεχνικές Extract Superclass και Extract Subclass αποτελούν τις πιο πολύπλοκες του συνόλου. Στα πλαίσια αυτής της εργασία επίσης υλοποιήθηκε ο Refactoring Trip Advisor, ένα εργαλείο υποβοήθησης της διαδικασίας ανακατασκευής που παρέχει στο χρήστη μια οπτικοποίηση του Χάρτη Ανακατασκευών καθώς και τη δυνατότητα αυτομάτου εντοπισμού σημείων στον κώδικα που χρήζουν ανακατασκευής. Τέλος παρουσιάστηκε η αξιολόγηση του

εργαλείου και του Χάρτη Ανακατασκευών στην οποία πήραν μέρος 16 χρήστες. Τα αποτελέσματα επιβεβαίωσαν το πρόβλημα καθώς υπήρχαν χρήστες με μέτρια γνώση πάνω στην ανακατασκευή κώδικα αλλά και επίσης παρατηρήθηκε μειωμένη απόδοση στην χειροκίνητη εφαρμογή της διαδικασίας ανακατασκευής. Αξιοποιώντας τον Χάρτη Ανακατασκευών και τις δυνατότητες του εργαλείου, οι χρήστες παρουσίασαν σημαντική βελτίωση στην διαδικασία ανακατασκευής διπλασιάζοντας τον αριθμό των τεχνικών που εφάρμοσαν και ανέφεραν θετικά σχόλια από την χρήση αυτών.

6.2 Μελλοντική Δουλειά

Σε αυτή την ενότητα θα συζητηθούν πιθανές μελλοντικές επεκτάσεις πάνω στη βασική συνεισφορά αυτής της εργασίας, τον Refactoring Trip Advisor και τον Χάρτη Ανακατασκευών. Όπως προαναφέρθηκε στην ενότητα 4.2 ο Refactoring Trip Advisor είναι εξοπλισμένος με την δυνατότητα του αυτόματου εντοπισμού σημείων στον κώδικα που χρήζουν κάποιας συγκεκριμένης τεχνικής ανακατασκευής. Παρόλα αυτά επειδή η ανάπτυξη αυτής της δυνατότητας δεν αποτελεί βασικό αντικείμενο αυτής της εργασία και επειδή τα περιθώρια χρόνου είναι περιορισμένα, η δυνατότητα αυτή υλοποιήθηκε μόνο για μερικές τεχνικές ανακατασκευής. Όμως σύμφωνα με το 4.1 υπάρχει η κατάλληλη υποδομή κώδικα ώστε να διευκολύνει την μελλοντική υλοποίηση της δυνατότητας του εντοπισμού και για τις 68 τεχνικές ανακατασκευής ή και ακόμα για την προσθήκη επιπλέον σε αυτές που είναι ήδη υλοποιημένες. Επίσης στα ίδια πλαίσια θα μπορούσε να υλοποιηθεί μια επιπλέον λειτουργία στον Refactoring Trip Advisor η οποία θα εκτελούσε όλες τις διαθέσιμες δυνατότητες εντοπισμού για έναν από τους έξι Χάρτες ανακατασκευής και θα παρουσίαζε το συνολικό αποτέλεσμα στο χρήστη. Έτσι ο χρήστης θα έχει μια συνολική άποψη των Code Smells στον κώδικα του και συμβουλευόμενος τον Χάρτη Ανακατασκευών να διαλέξει πια τεχνική θα εφαρμόσει πρώτη. Επίσης μελλοντικές επεκτάσεις και πιθανές μετατροπές μπορούν να γίνουν και στο Χάρτη Ανακατασκευών. Πέρα από τον κατάλογο του Fowler θα μπορούσε να γίνει περαιτέρω μελέτη πάνω στην διαδικασία ανακατασκευής των χρηστών στην πράξη. Από εκεί είναι πιθανό να εξαχθεί καινούργια γνώση για την προσθήκη ή αφαίρεση μερικών συσχετίσεων στον Χάρτη Ανακατασκευών ή και ακόμη να δημιουργηθεί ένας νέος τύπος σχέσης μεταξύ των τεχνικών.

ΑΝΑΦΟΡΕΣ

[1]. David Garvin, “What Does 'Product Quality' Really Mean?”, MIT Sloan Management Review 26, No. 1, 1984.

[2]. B. Kitchenham and S. Pfleeger, “Software quality: the elusive target, Software”, IEEE, Vol. 13, No. 1, pp. 12–21, 1996.

[3]. ISO/IEC 9126-1:2001, Software Engineering – Product Quality – Part 1: Quality Model.

[4]. ISO/IEC 25010:2011, Systems and Software Engineering – Systems and software Quality Requirements and Evaluation (SquaRE) – System and Software Quality models.

[5]. CISQ, “Specifications for Automated Quality Characteristic Measures” (Last Version 2.1, 2012).

[6]. Martin Robert, “Clean Code”, Prentice Hall, 2009.

[7]. Fowler Martin, “Refactoring: Improving the design of existing code”, Addison Wesley, 1999.

[8]. William Griswold, “Program Restructuring as an Aid to Software Maintenance”, Ph.D. thesis, University of Washington, 1991.

[9]. William Opdyke, “Refactoring Object-Oriented Frameworks”, Ph.D. thesis, University of Illinois a Urbana-Champaign, 1992.

[10]. Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan, "An Empirical Study of Refactoring Challenges and Benefits at Microsoft", IEEE Transactions on Software Engineering, Vol. 40, No. 7, July 2014.

[11]. Jehad Al Dallal, "Identifying refactoring opportunities in object-oriented code: A systematic literature review", Information and Software Technology, Vol. 58, Pages 231 – 249, February 2015.

[12]. JGraphX on GitHub

[13]. Nikolaos Tsantalis and Alexander Chatzigeorgiou, "Identification of extract method refactoring opportunities for the decomposition of methods", Journal of Systems and Software, Vol. 84, Issue 10, Pages 1757–1782, October 2011.

[14]. Nikolaos Tsantalis and Alexander Chatzigeorgiou, "Identification of Move Method Refactoring Opportunities", IEEE Transactions on Software Engineering, Vol. 35, Issue 3, Pages 347-367 , May 2009.

[15]. Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia and Alexander Chatzigeorgiou, "Identification and application of Extract Class refactorings in object-oriented systems", Journal of Systems and Software, Vol. 85 Issue 10, Pages 2241-2260, October, 2012 .

[16]. Claude Y. Laporte, Nabil Berrhouma, Mikel Doucet and Edgardo Palza-Vargas, "Measuring the Cost of Software Quality of a Large Software Project at Bombardier Transportation: A Case Study", Software Quality Professional, Vol. 14, Issue 3, Pages 14-31, June 2012.

[17]. Apostolos V. Zarras, Theofanis Vartziotis and Panos Vassiliadis, “Navigating Through the Archipelago of Refactorings”, ESEC/FSE 2015 Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Pages 922-925, September 2015.

[18]. E. Murphy-Hill, C. Parnin and A. P. Black, “How we refactor, and how we know it”, IEEE Transactions on Software Engineering, Vol.38, Issue 1, Pages 5-18, December 2011.

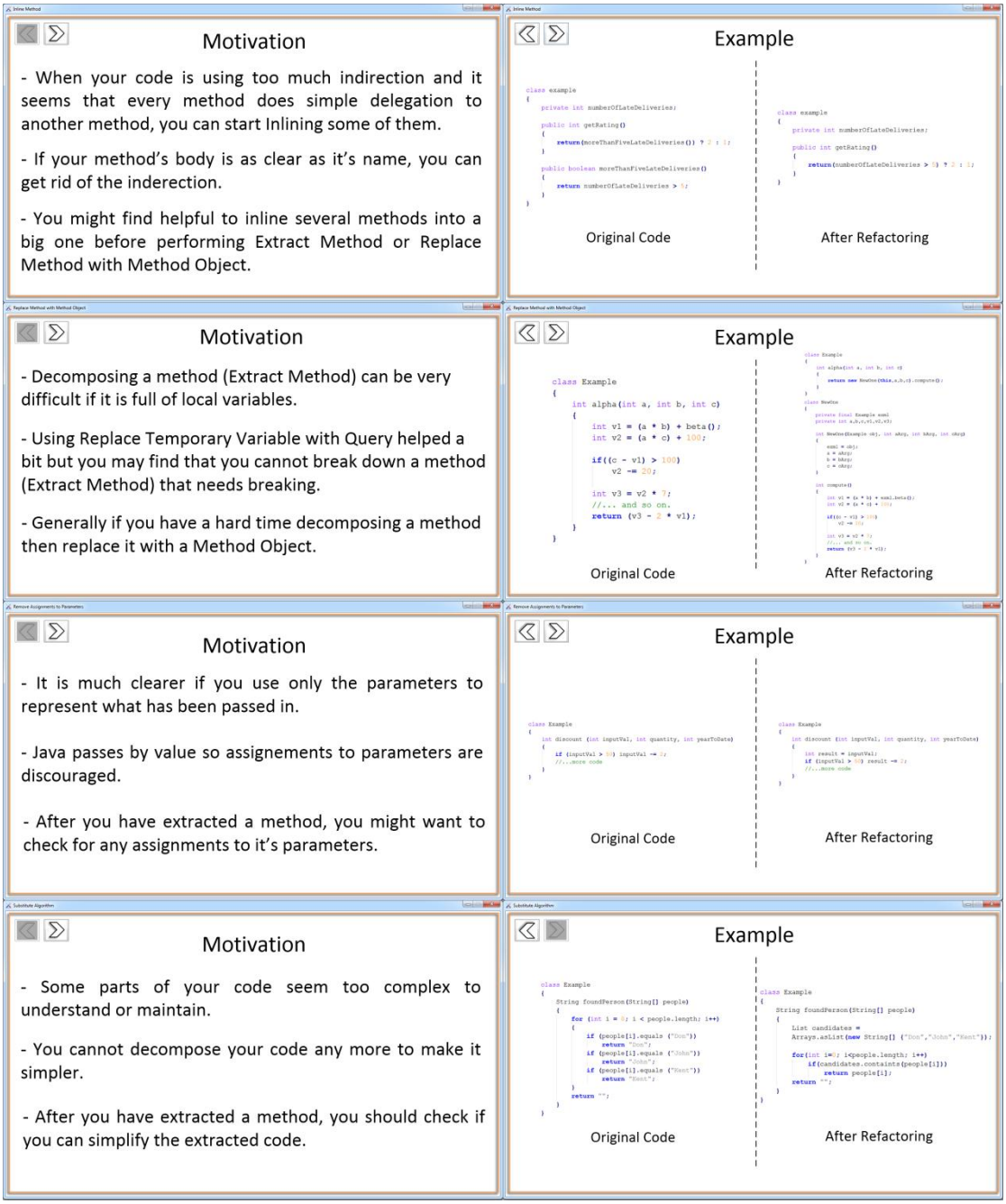
[19]. Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Brian P. Bailey and Ralph E. Johnson, “Use, Disuse, and Misuse of Automated Refactorings”, ICSE '12 Proceedings of the 34th International Conference on Software Engineering, Pages 233-243, 2012.

[20]. Your First Plug-in: Developing the Eclipse "Hello World" plug-in.

[21]. Tom Mens, Gabriele Taentzer and Olga Runge, “Analysing refactoring dependencies using graph transformation”, Software & Systems Modeling, Vol. 6, Issue 3, Pages 269-285, September 2007.

ΠΑΡΑΡΤΗΜΑ

Method Composition

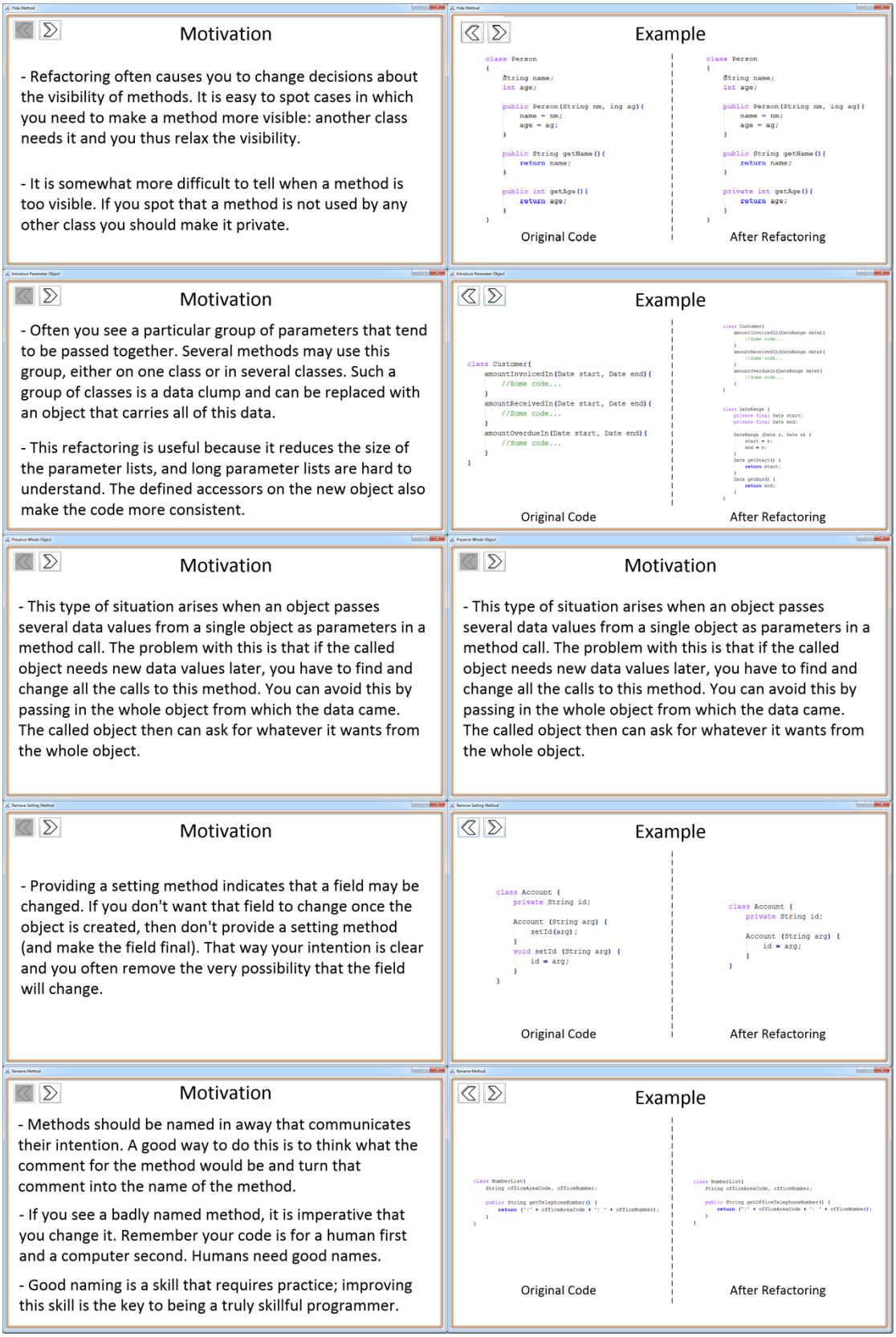


Σχήμα Π.1 Διαφάνειες για Inline Method, Replace Method with Method Object, Remove Assignments to Parameters και Substitute Algorithm

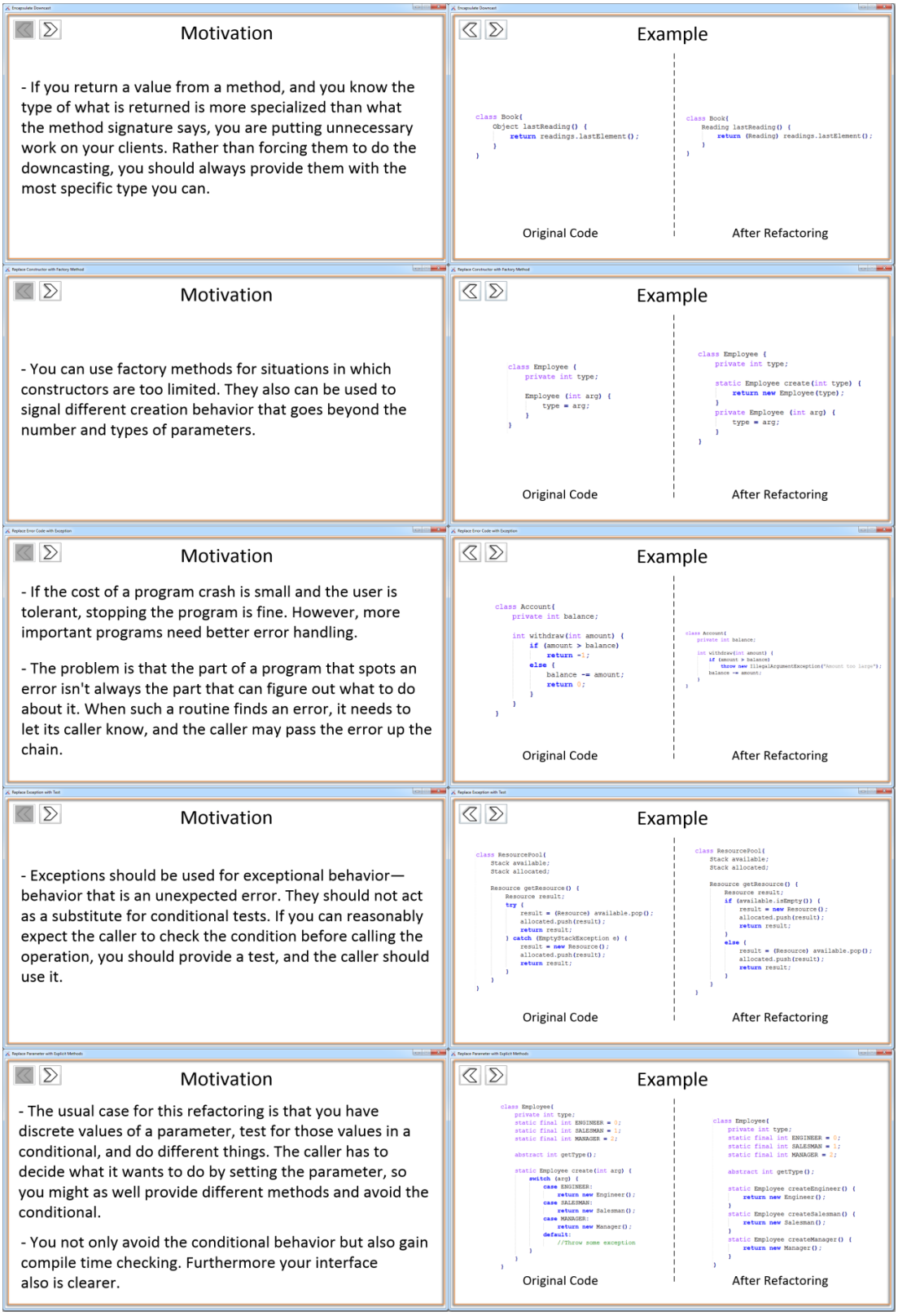
Σχήμα Π.2 Διαφάνειες για Split Temporary Variable, Introduce Explaining Variable, Inline Temp, Replace Temp with Query και Extract Method

Method Call Improvement

Σχήμα Π.3 Διαφάνειες για Add Parameter, Remove Parameter, Parameterize Method, Replace Parameter with Method και Separate Query from Modifier

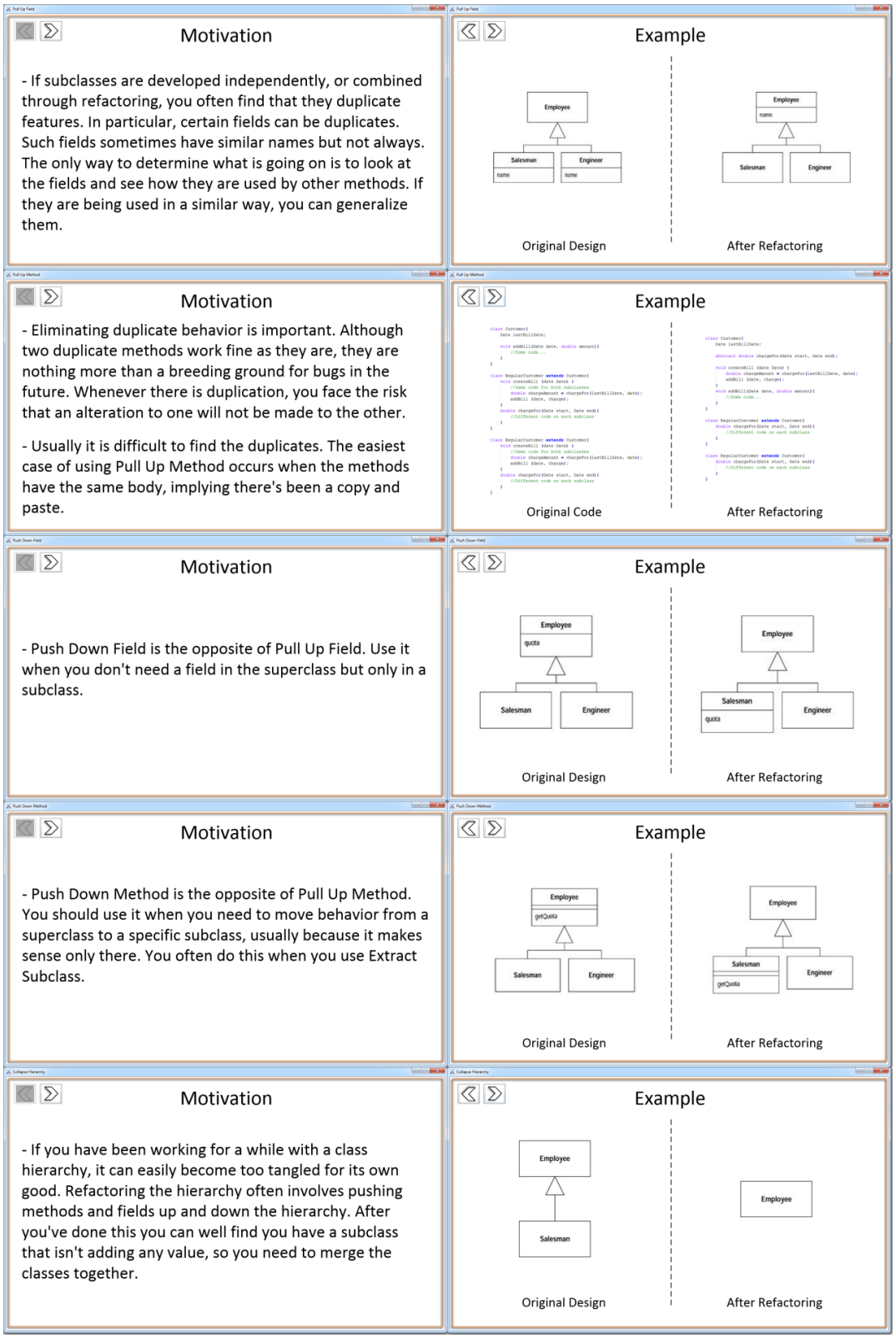


Σχήμα Π.4 Διαφάνειες για Hide Method, Introduce Parameter Object, Preserve Whole Object, Remove Setting Method και Rename Method

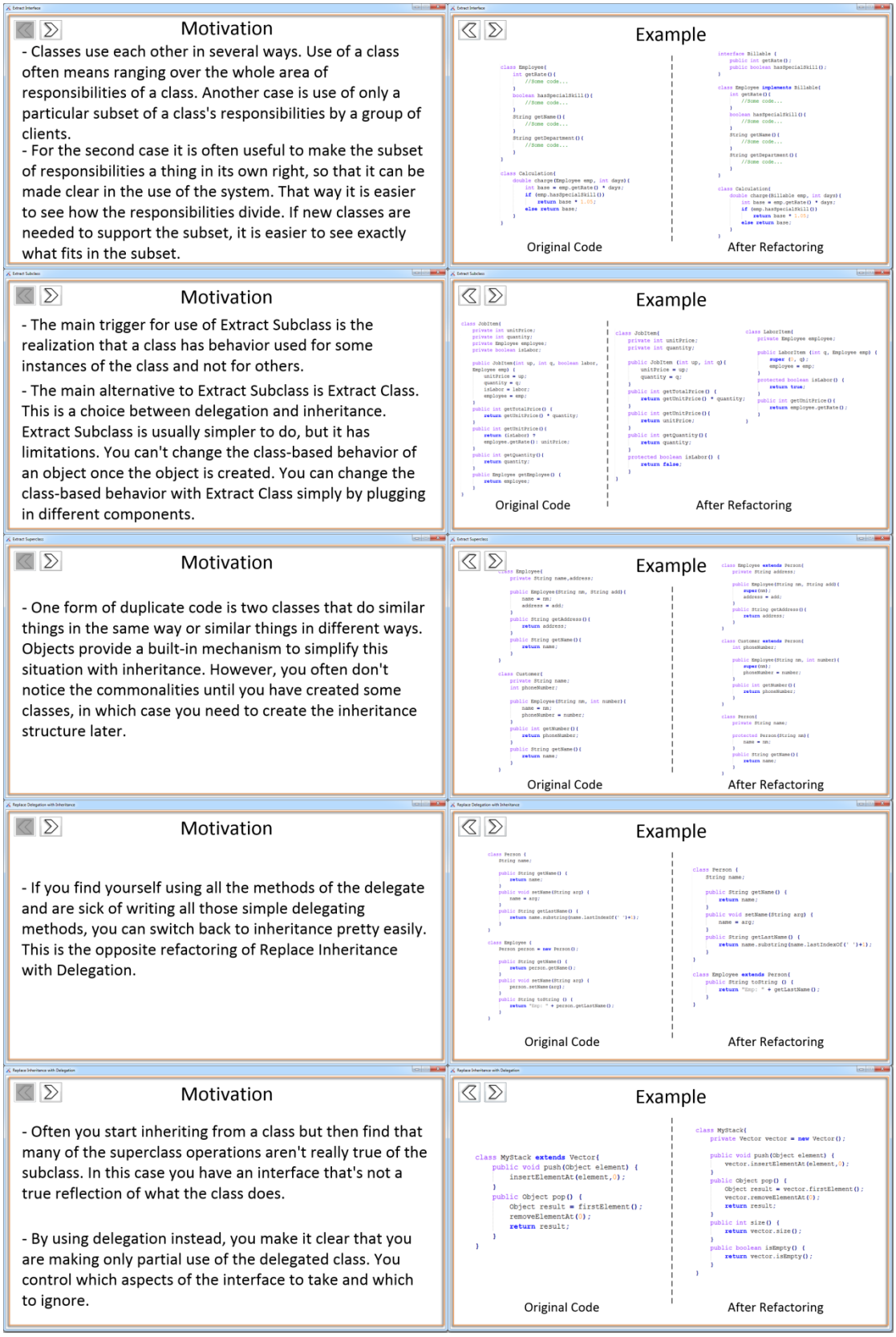


Σχήμα Π.5 Διαφάνειες για Encapsulate Downcast, Replace Constructor with Factory Method, Replace Error Code with Exception, Replace Exception with Test και Replace Parameter with Explicit Methods

Generalization Improvement

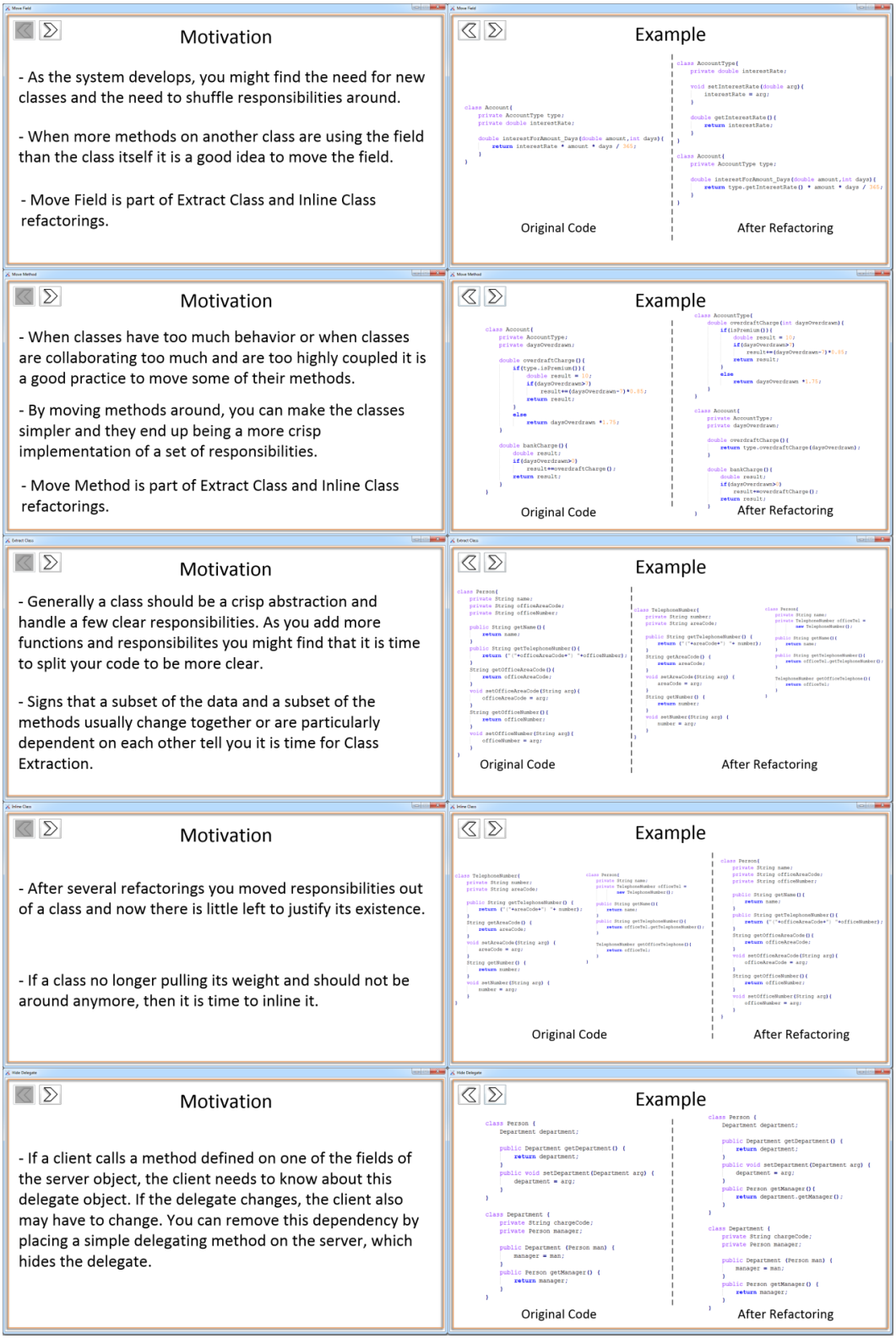


Σχήμα Π.6 Διαφάνειες για Pull Up Field, Pull Up Method, Push Down Field, Push Down Method και Collapse Hierarchy

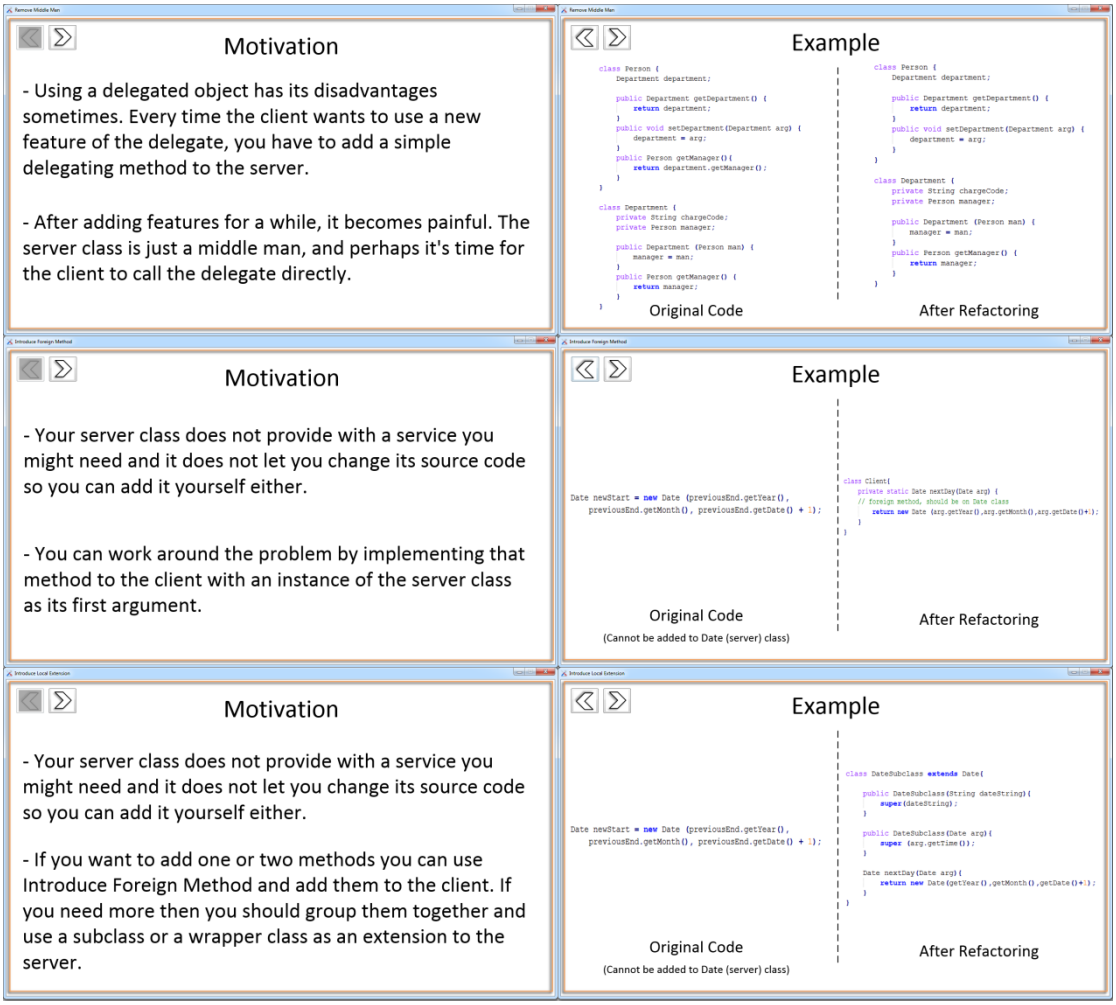


Σχήμα Π.7 Διαφάνειες για Extract Interface, Extract Subclass, Extract Superclass, Replace Delegation with Inheritance και Replace Inheritance with Delegation

Feature Movement Between Objects

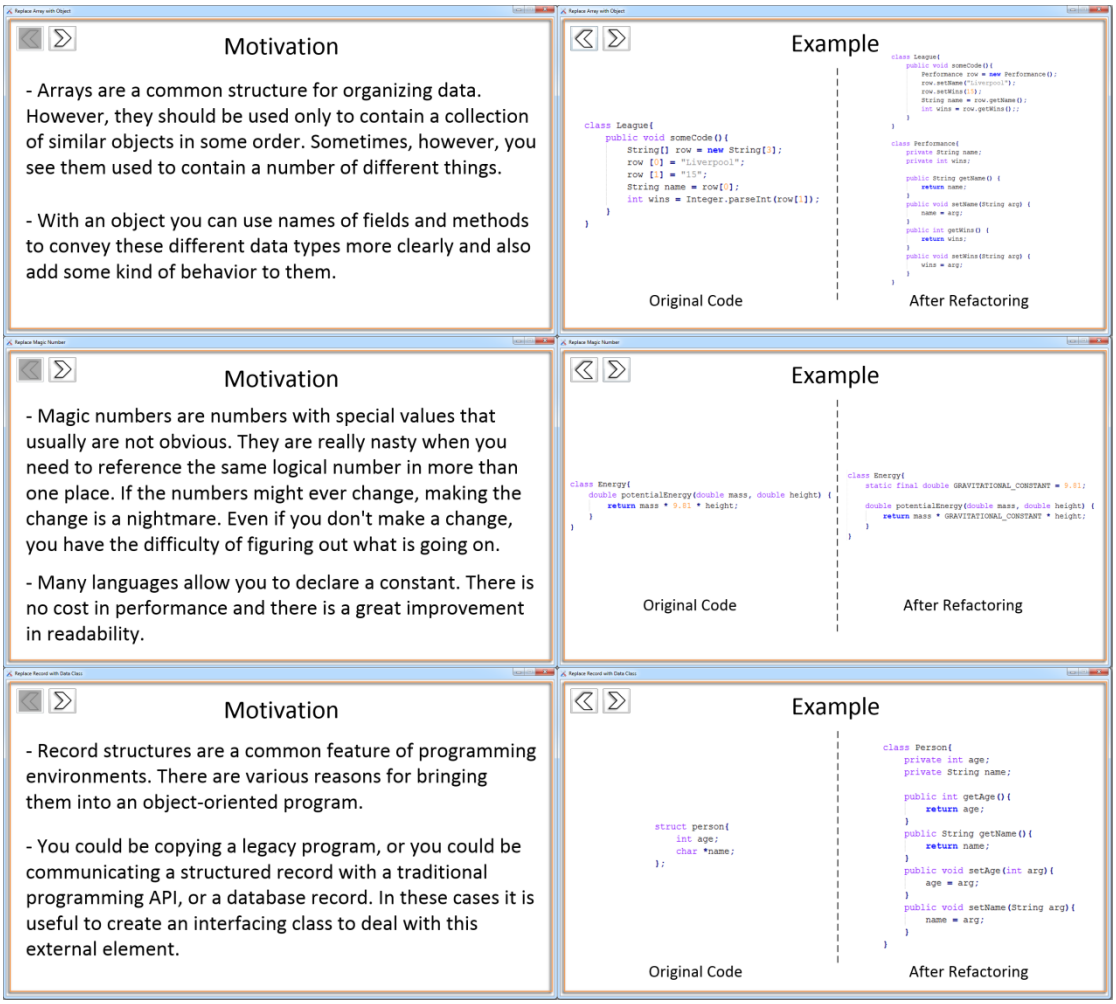


Σχήμα Π.8 Διαφάνειες για Move Field, Move Method, Extract Class, Inline Class και Hide Delegate

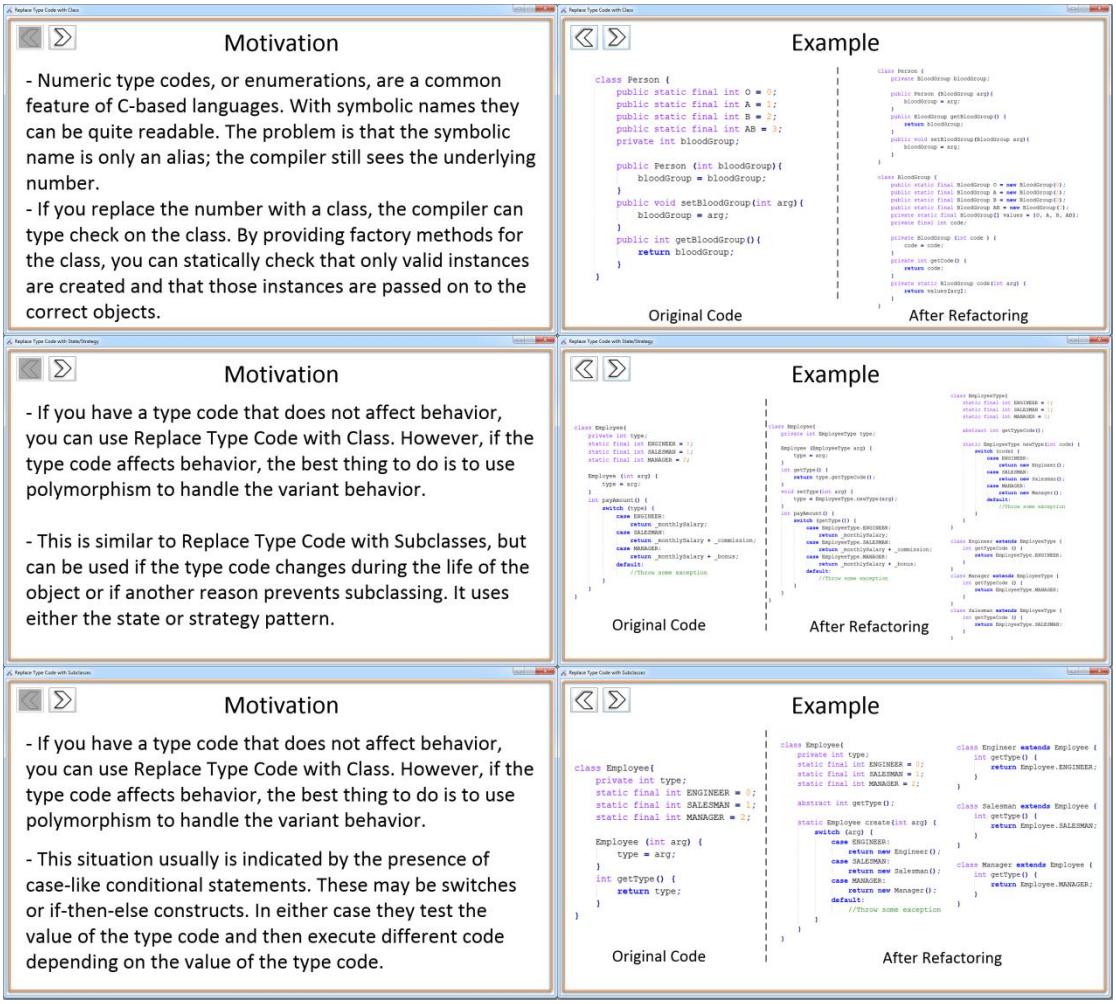


Σχήμα Π.9 Διαφάνειες για Remove Middle Man, Introduce Foreign Method και Introduce Local Exception

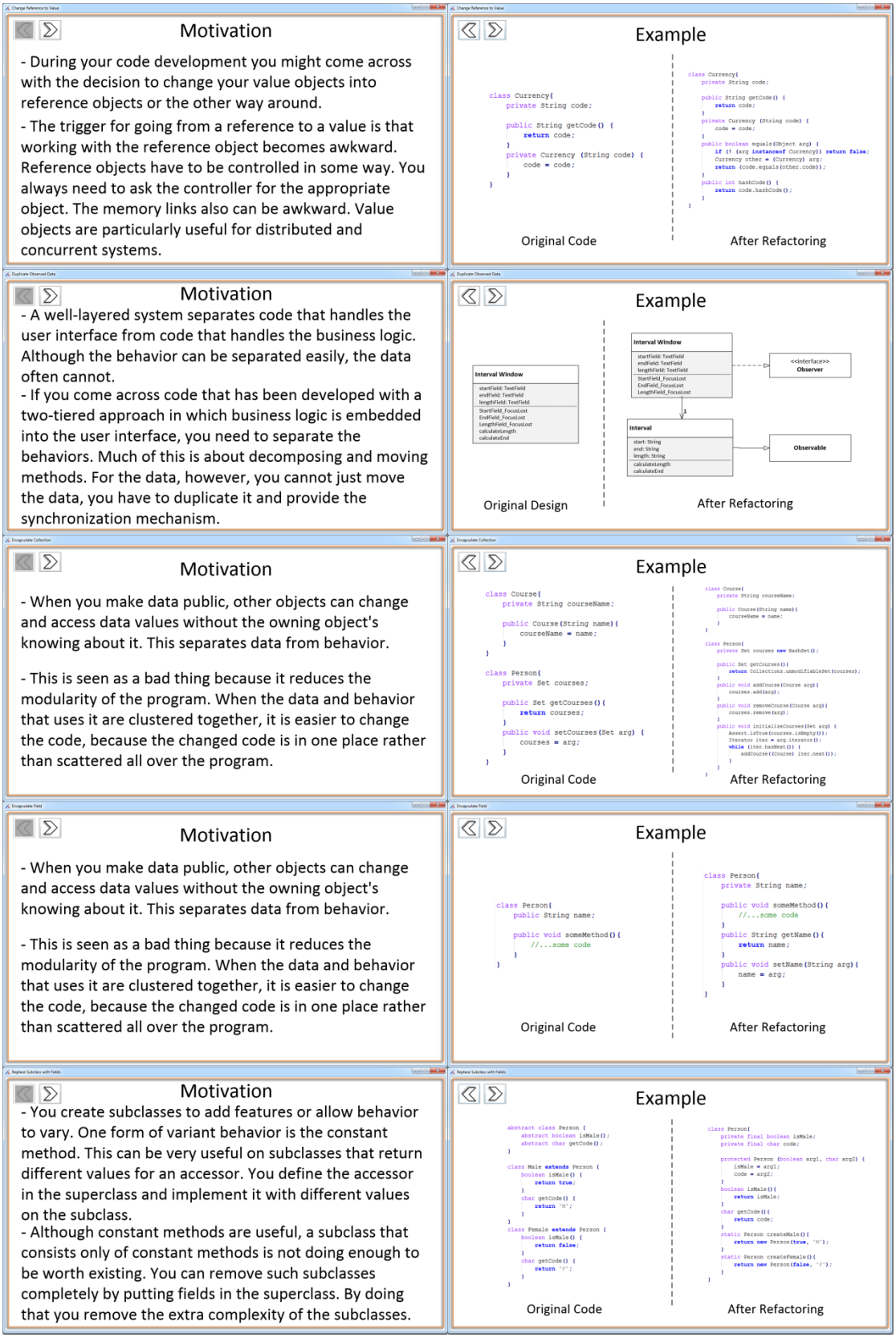
Data Organization



Σχήμα Π.10 Διαφάνειες για Replace Array with Object, Replace Magic Number και Replace Record with Data Class



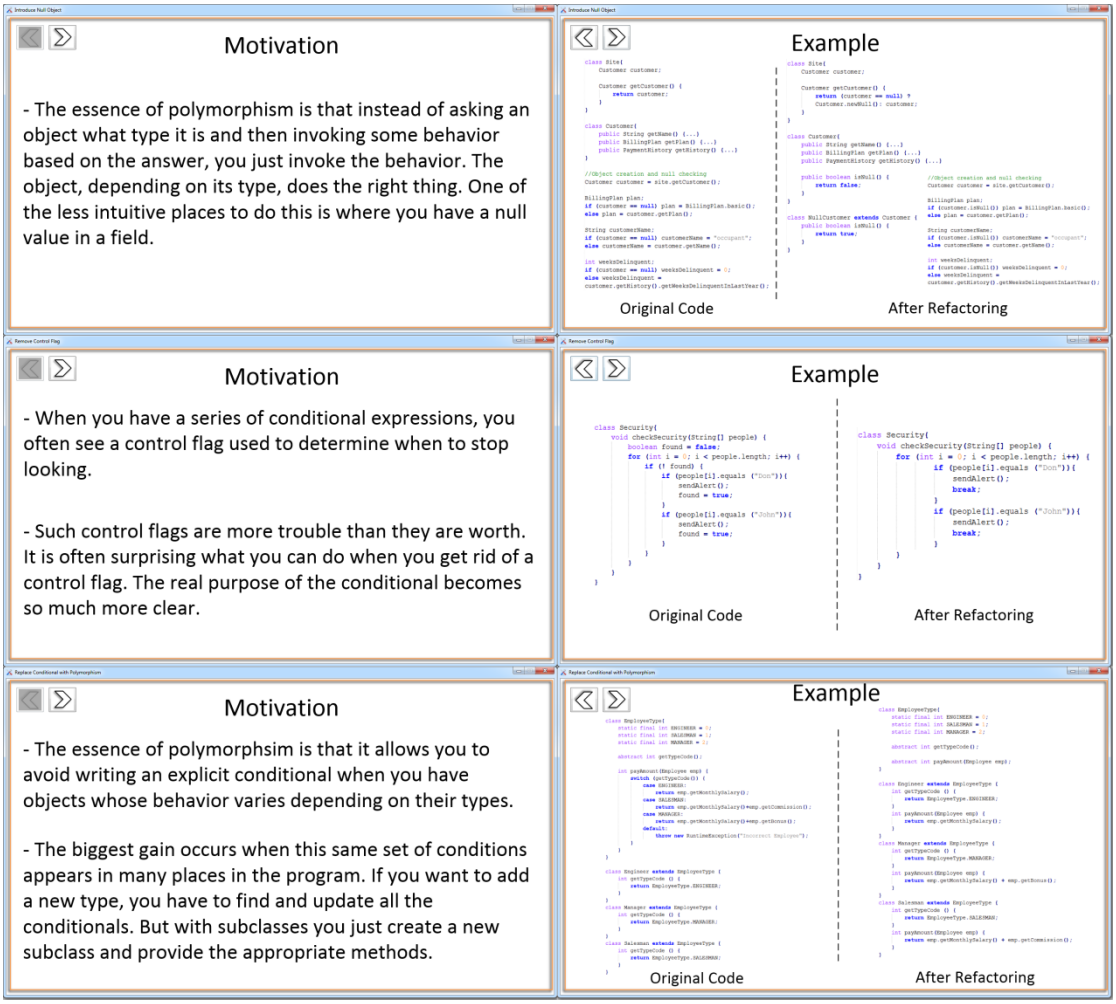
Σχήμα Π.11 Διαφάνειες για Replace Type Code with Class, Replace Type Code with State/Strategy και Replace Type Code with Subclasses



Σχήμα Π.12 Διαφάνειες για Change Reference to Value, Duplicate Observed Data, Encapsulate Collection, Encapsulate Field και Replace Subclass with Fields

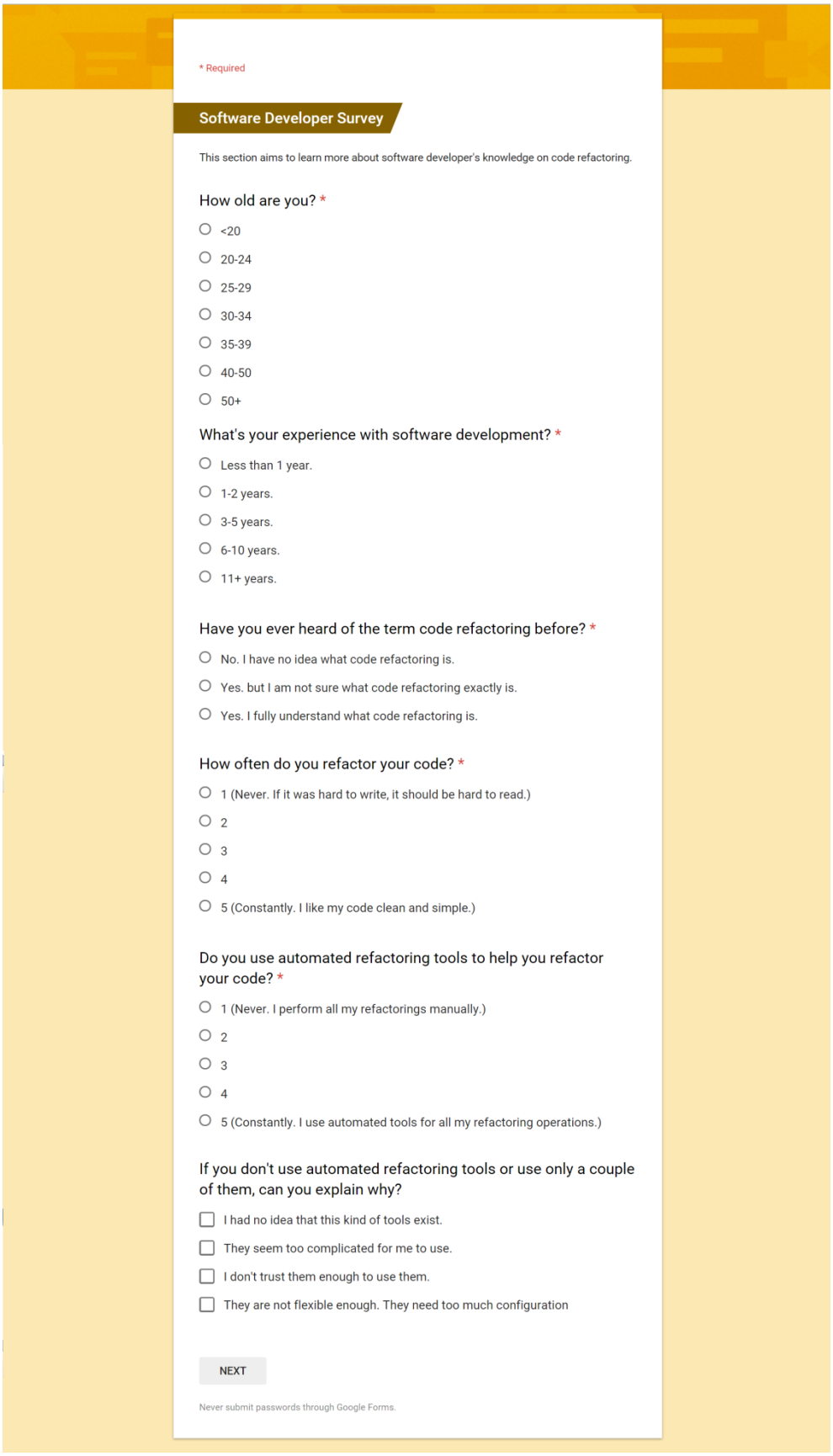
Σχήμα Π.13 Διαφάνειες για Change Bidirectional Association to Unidirectional, Change Unidirectional Association to Bidirectional, Change Value to Reference, Replace Data Value with Object και Self Encapsulate Field

Conditional Expression Simplification



Σχήμα Π.14 Διαφάνειες για Introduce Null Object, Remove Control Flag και Replace Conditional with Polymorphism

Σχήμα Π.15 Διαφάνειες για Consolidate Conditional Expression, Consolidate Duplicate Conditional Fragments, Decompose Conditional, Introduce Assertion και Replace Nested Conditional with Guard Clauses



* Required

Software Developer Survey

This section aims to learn more about software developer's knowledge on code refactoring.

How old are you? *

- <20
- 20-24
- 25-29
- 30-34
- 35-39
- 40-50
- 50+

What's your experience with software development? *

- Less than 1 year.
- 1-2 years.
- 3-5 years.
- 6-10 years.
- 11+ years.

Have you ever heard of the term code refactoring before? *

- No. I have no idea what code refactoring is.
- Yes. but I am not sure what code refactoring exactly is.
- Yes. I fully understand what code refactoring is.

How often do you refactor your code? *

- 1 (Never. If it was hard to write, it should be hard to read.)
- 2
- 3
- 4
- 5 (Constantly. I like my code clean and simple.)

Do you use automated refactoring tools to help you refactor your code? *

- 1 (Never. I perform all my refactorings manually.)
- 2
- 3
- 4
- 5 (Constantly. I use automated tools for all my refactoring operations.)

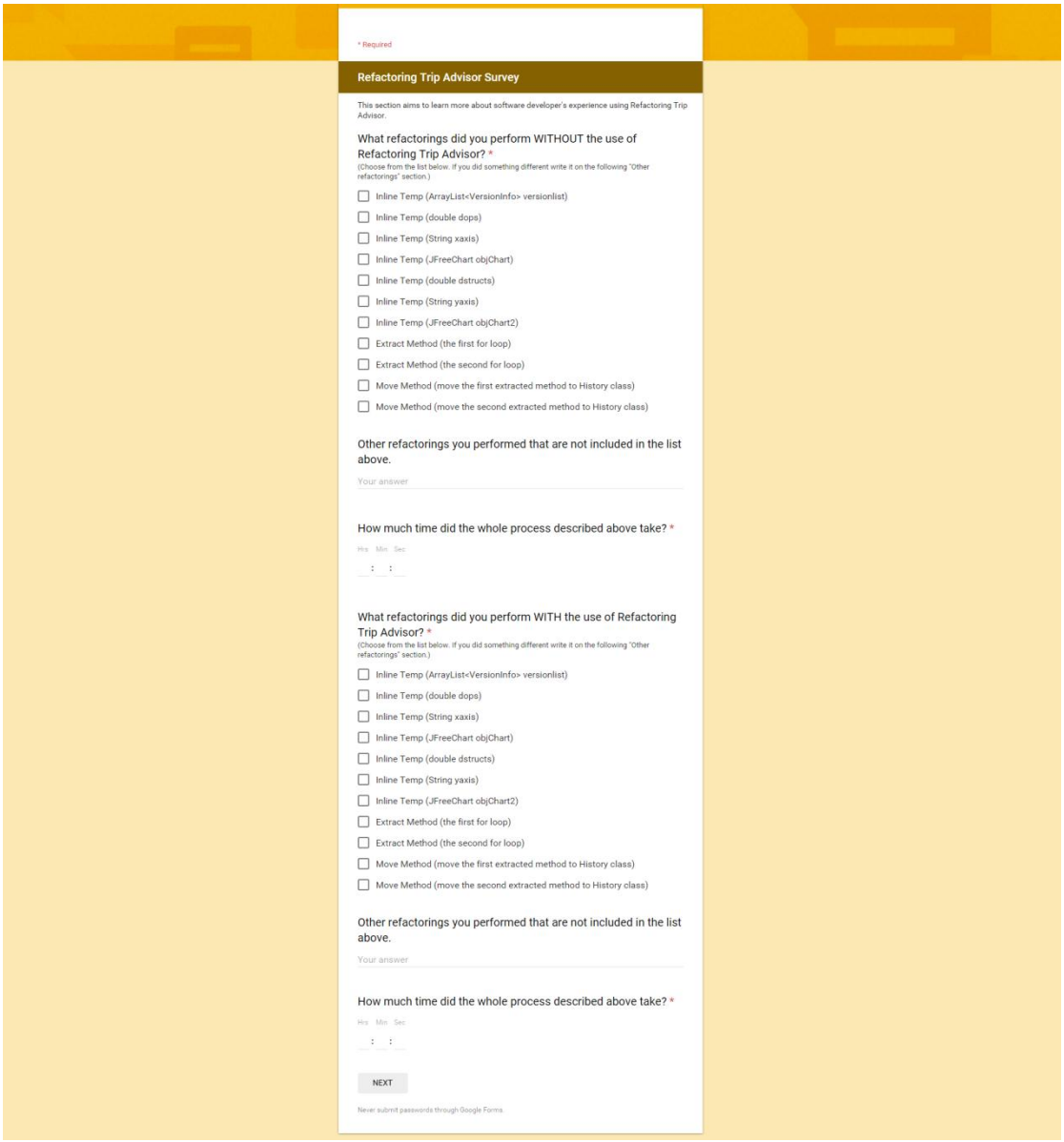
If you don't use automated refactoring tools or use only a couple of them, can you explain why?

- [ ] I had no idea that this kind of tools exist.
- [ ] They seem too complicated for me to use.
- [ ] I don't trust them enough to use them.
- [ ] They are not flexible enough. They need too much configuration

NEXT

Never submit passwords through Google Forms.

Σχήμα Π.16 Ερωτηματολόγιο Αξιολόγησης Εμπειρίας Χρηστών



* Required

Refactoring Trip Advisor Survey

This section aims to learn more about software developer's experience using Refactoring Trip Advisor.

What refactorings did you perform WITHOUT the use of Refactoring Trip Advisor? *
(Choose from the list below. If you did something different write it on the following "Other refactorings" section.)

- [ ] Inline Temp (ArrayList<VersionInfo> versionlist)
- [ ] Inline Temp (double dops)
- [ ] Inline Temp (String xaxis)
- [ ] Inline Temp (JFreeChart objChart)
- [ ] Inline Temp (double dstructs)
- [ ] Inline Temp (String yaxis)
- [ ] Inline Temp (JFreeChart objChart2)
- [ ] Extract Method (the first for loop)
- [ ] Extract Method (the second for loop)
- [ ] Move Method (move the first extracted method to History class)
- [ ] Move Method (move the second extracted method to History class)

Other refactorings you performed that are not included in the list above.

Your answer

How much time did the whole process described above take? *

Hrs Min Sec

: :

What refactorings did you perform WITH the use of Refactoring Trip Advisor? *
(Choose from the list below. If you did something different write it on the following "Other refactorings" section.)

- [ ] Inline Temp (ArrayList<VersionInfo> versionlist)
- [ ] Inline Temp (double dops)
- [ ] Inline Temp (String xaxis)
- [ ] Inline Temp (JFreeChart objChart)
- [ ] Inline Temp (double dstructs)
- [ ] Inline Temp (String yaxis)
- [ ] Inline Temp (JFreeChart objChart2)
- [ ] Extract Method (the first for loop)
- [ ] Extract Method (the second for loop)
- [ ] Move Method (move the first extracted method to History class)
- [ ] Move Method (move the second extracted method to History class)

Other refactorings you performed that are not included in the list above.

Your answer

How much time did the whole process described above take? *

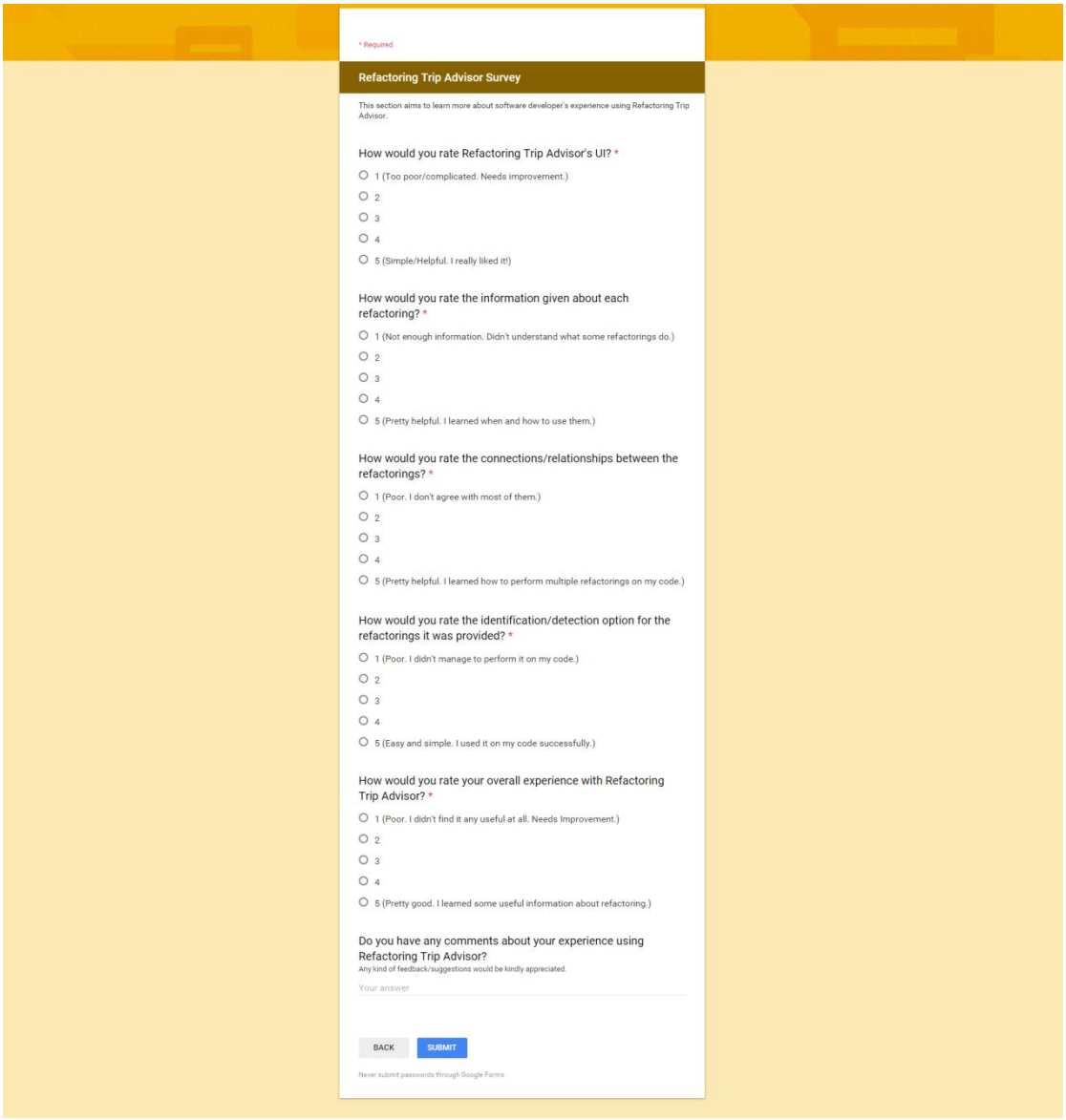
Hrs Min Sec

: :

NEXT

Never submit passwords through Google Forms.

Σχήμα Π.17 Ερωτηματολόγιο πάνω στις Δύο Εργασίες που οι Χρήστες εκτέλεσαν



Σχήμα Π.18 Ερωτηματολόγιο Αξιολόγησης του Refactoring Trip Advisor από τους Χρήστες