ΤΑΞΙΝΟΜΗΣΗ ΠΡΟΤΕΙΝΟΜΕΝΩΝ ΑΝΑΔΟΜΗΣΕΩΝ ΣΕ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗ ΣΥΣΤΗΜΑΤΑ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΙΣΤΟΡΙΚΑ ΔΕΔΟΜΕΝΑ

Λιάκος Αριστείδης

Διπλωματική Εργασία

Επιβλέπων Καθηγητής : Χατζηγεωργίου Αλέξανδρος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Θεσσαλονίκη, Ιούνιος 2010

ΠΕΡΙΛΗΨΗ

Η συντήρηση του λογισμικού είναι μια διαδικασία εξόχως σημαντική και η αναγκαιότητα της γίνεται ιδιαίτερα αντιληπτή καθώς το λογισμικό «παλιώνει». Για την αποφυγή επιβάρυνσης της εργασίας του συντηρητή είναι επιθυμητό το λογισμικό να είναι «καλά» σχεδιασμένο πράγμα που σημαίνει πως πρέπει να συμβαδίζει με *Αρχές Σχεδίασης* και *Πρότυπα Σχεδίασης*. Ωστόσο, αν και για την πλειοψηφία των συστημάτων κυριαρχεί αρχικά η αίσθηση ότι είναι καλά σχεδιασμένα, η ποιότητα τους φθίνει με την πάροδο του χρόνου. Οι τροποποιήσεις που λαμβάνουν χώρα από έκδοση σε έκδοση έχουν ως αποτέλεσμα τη συχνή εμφάνιση τμημάτων κώδικα που αγνοούν τις αρχές και τα πρότυπα σχεδίασης. Για τα τμήματα αυτά έχει επικρατήσει στη βιβλιογραφία της Τεχνολογίας Λογισμικού ο όρος «κακές οσμές» (bad smells). Η εξάλειψη τους στη συντριπτική πλειοψηφία των περιπτώσεων επιτυγχάνεται με την εφαρμογή αναδόμησης του κώδικα.

Τα τελευταία χρόνια έχουν αναπτυχθεί αρκετά ημιαυτόματα εργαλεία που εντοπίζουν κακές οσμές στον κώδικα και επιπλέον προτείνουν για κάθε περίπτωση κατάλληλη αναδόμηση. Στα μεσαία και μεγάλα συστήματα όμως, παραμένει το πρόβλημα της πληθώρας των κακών οσμών που καθιστά αδύνατη την εφαρμογή αναδόμησης για όλες αυτές. Μια λύση για το πρόβλημα είναι η ανάθεση προτεραιότητας σε κάθε αναδόμηση ώστε να παρέχονται στο συντηρητή ταξινομημένες με αποτέλεσμα η επιλογή αυτών που θα εφαρμοστούν να μην αποτελεί χρονοβόρα διαδικασία.

Έτσι, ο σκοπός της εργασίας αυτής είναι η πρόταση μιας νέας μεθοδολογίας ταξινόμησης των προτεινόμενων αναδομήσεων χρησιμοποιώντας τις προηγούμενες εκδόσεις του ίδιου λογισμικού. Η μεθοδολογία υλοποιήθηκε επίσης και προγραμματιστικά ως επέκταση των δυνατοτήτων του *JDeodorant* που είναι εργαλείο εντοπισμού κακών οσμών και αναπτύχθηκε από το *εργαστήριο Τεχνολογίας Λογισμικού* του *Τμήματος Εφαρμοσμένης Πληροφορικής* του Πανεπιστημίου Μακεδονίας. Οι τρεις μη τετριμμένες κακές οσμές που εντοπίζει το εργαλείο και επιπρόσθετα προτείνει αναδόμηση για κάθε περίπτωση είναι οι Long Method, State Checking, Feature Envy οι οποίες αποτελούν αντικείμενο και της παρούσης εργασίας.

Η μεθοδολογία για την ταξινόμηση των αναδομήσεων βασίζεται σε έναν παράγοντα που υπολογίζεται μέσω της εξέτασης του κώδικα των προηγούμενων εκδόσεων και τον εντοπισμό αλλαγών σε συγκεκριμένα τμήματα αυτού, στα οποία εμφανίζονται οι κακές οσμές. Για τον υπολογισμό του παράγοντα λαμβάνονται υπόψη η ύπαρξη και το εύρος της αλλαγής τα οποία ορίζονται διαφορετικά για κάθε κακή οσμή καθώς και η εγγύτητα της έκδοσης που η κάθε μια εντοπίζεται με την τρέχουσα έκδοση του λογισμικού. Τέλος έχει γίνει μια πρώτη προσπάθεια αξιολόγησης των αποτελεσμάτων που εξήχθησαν από πραγματικά συστήματα ανοικτού κώδικα από έναν ανεξάρτητο ειδικό.

ABSTRACT

Software maintenance is a procedure of the utmost importance to the software lifecycle and its necessity becomes quite apparent as software systems get older. To relieve the software designer from increased effort during maintenance, software should be compatible with *Design Principles* and *Design Patterns*. Nevertheless, despite the fact that the initial implementation of the majority of systems denotes that they are adequately designed, their quality gradually declines with time. A side-effect of modifications occurring in code from version to version is the appearance of code pieces that disregard *Design Principles* or *Design Patterns.* The term that has prevailed in the literature of Software Engineering for these pieces of code is "Bad Smells". Their elimination, for the vast majority of the circumstances in which they manifest themselves, is accomplished by the application of an appropriate refactoring.

Last years, a considerable number of semiautomatic tools that detect bad smell occurrences and also make a suggestion for their removal have been developed. However, the problem is persistent for middle and large size systems, due to the copiousness of bad smell occurrences that make the application of refactoring for all of them impossible. A possible treatment is the assignment of priority to every candidate refactoring in order all of them to be granted to the designer in ranking order. This way the selection of which should be applied becomes a less time-consuming process.

Hence, the purpose of this work is to propose a novel methodology for ranking the refactoring suggestions using the antecedent versions of the same software. The methodology has also been implemented in a programming language (Java) as an expansion of *JDeodorant* capacities. JDeodorant is a semiautomatic tool for the detection of bad smell occurrences in Java projects developed in the *Software Engineering Laboratory* in the *Applied Informatics Department* of *University of Macedonia Greece*. Three none trivial bad smells which the tool is able to identify are *Long Method, State Checking and Feature Envy*. It also renders a concrete solution for each occasion by an appropriate refactoring suggestion.

The proposed methodology is based on a factor being computed by the examination of the antecedent versions' source code and by the identification of "changes" have taken place on particular pieces of code affected by the existence of bad smells. The existence of change, the extent of it and also the proximity of the version in which it has been detected are taken into consideration for the computation of this factor. Finally, an initial attempt has been made for evaluating the results obtained from open source code projects by comparing them to the opinion of an independent expert.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τη μεγάλη μου εκτίμηση και τις θερμές μου ευχαριστίες προς τον επιβλέποντα της διπλωματικής μου εργασίας Επίκουρο Καθηγητή του τμήματος Εφαρμοσμένης Πληροφορικής κ. Χατζηγεωργίου Αλέξανδρο. Η άψογη συνεργασία που είχαμε και ο καθοδηγητικός του ρόλος ήταν δυο παράγοντες καίριας σημασίας για την εκπόνηση της εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα του τμήματος Εφαρμοσμένης Πληροφορικής κ. Τσάνταλη Νικόλαο , εμπνευστή, δημιουργό και σχεδιαστή του λογισμικού JDeodorant για την πολύτιμη βοήθεια του σε προγραμματιστικά θέματα που προέκυψαν καθ' όλη τη διάρκεια εκπόνησης της διπλωματικής εργασίας.

Περιεχόμενα

1. **Εισαγωγή**……..8
   - 1.1. **Κακές οσμές και αναδομήσεις**…….....8
   - 1.2. **Εργαλεία εντοπισμού κακών οσμών**……...8
   - 1.3. **Σκοπός της εργασίας**……9
   - 1.4. **Γενικότερη φιλοσοφία της εργασίας**…….....10
   - 1.5. **Διάρθρωση της εργασίας**……...11
2. **Επισκόπηση σχετικών εργασιών**……...13
3. **Μεθοδολογία**……..16
4. **Long Method**…….21
   - 4.1. **Σύντομη περιγραφή της κακής οσμής**……..21
   - 4.2. **Χειρισμός των ιστορικών δεδομένων**……...22
   - 4.3. **Παρατηρήσεις και σχόλια**……..25
   - 4.4. **Αποτελέσματα**…….26
5. **State Checking**……..29
   - 5.1. **Σύντομη περιγραφή της κακής οσμής**…….29
   - 5.2. **Χειρισμός των ιστορικών δεδομένων**……...31
   - 5.3. **Αποτελέσματα**…….34
6. **Feature Envy**…….36
   - 6.1. **Σύντομη περιγραφή της κακής οσμής**…….36
   - 6.2. **Χειρισμός των ιστορικών δεδομένων**……...37
   - 6.3. **Αποτελέσματα**…….39
7. **Προγραμματιστική υλοποίηση**……42
   - 7.1. **Εισαγωγικά**…….42
   - 7.2. **Κοινή τεχνική στις τρεις περιπτώσεις**…….……43
   - 7.3. **Long Method**……..44
   - 7.4. **State Checking**……45
   - 7.5. **Feature Envy**……..47
8. **Αξιολόγηση των αποτελεσμάτων**……...49
   - 8.1. **Γενικά**……..49
   - 8.2. **Μέθοδοι αξιολόγησης**……50
   - 8.3. **Αξιολόγηση αποτελεσμάτων για Long Method**……...51
   - 8.4. **Αξιολόγηση αποτελεσμάτων για State Checking**...…….53
   - 8.5. **Αξιολόγηση αποτελεσμάτων για Feature Envy**…….…..54

9. Συμπεράσματα και μελλοντική εργασία……...56
9.1. Συμπεράσματα……56
9.2. Μελλοντική εργασία……..57

1. ΕΙΣΑΓΩΓΗ

1.1. «Κακές οσμές» και αναδομήσεις

Οι «κακές οσμές» (bad smells) αποτελούν συμπτώματα κακής σχεδίασης τμήματος κώδικα και οφείλουν την ύπαρξη τους κυρίως σε παραβιάσεις αρχών σχεδίασης (design principles) [14] και προτύπων σχεδίασης (design patterns) [6] η ακόμα και την εφαρμογή αντί-προτύπων (antipatterns) [2]. Παρόλο που η λειτουργία του συστήματος είναι η επιθυμητή, η εμφάνιση κακών οσμών στον κώδικα επιφέρει χαμηλές τιμές σε αρκετές κρίσιμες για την ποιότητα του λογισμικού μετρικές, αυξάνει την πολυπλοκότητα του συστήματος και καθιστά δυσκολότερη τη διαδικασία συντήρησης του. Η αντιμετώπιση αυτού του είδους των προβλημάτων γίνεται στη συντριπτική πλειοψηφία των περιπτώσεων με την εφαρμογή «αναδομήσεων» (refactoring). Στην ουσία ο υπάρχων κώδικας που εμφανίζει κακή οσμή αντικαθίσταται με νέο κώδικα συμβατό με τις αρχές και τα πρότυπα σχεδίασης χωρίς να επηρεάζεται η εξωτερική συμπεριφορά του συστήματος.

1.2. Εργαλεία εντοπισμού κακών οσμών

Στα μεσαία και μεγάλα συστήματα λογισμικού ο εντοπισμός του συνόλου των κακών οσμών είναι σχεδόν αδύνατον να διεκπεραιωθεί με επισκόπηση ή ανάγνωση του κώδικα λόγου του τεράστιου πλήθους των γραμμών κώδικα που περιλαμβάνουν. Για το λόγο αυτό τα τελευταία χρόνια έχουν αναπτυχθεί «ημιαυτόματα» εργαλεία που εντοπίζουν τέτοιου είδους προβλήματα σχεδίασης. Τα περισσότερα από αυτά παρέχουν στο χρήστη τη δυνατότητα όχι μόνο να εντοπίσει τις κακές οσμές σε ουσιαστικά μηδαμινό χρόνο αλλά προτείνουν και το βέλτιστο τρόπο αναδόμησης για την πλήρη αντιμετώπιση του προβλήματος.

Ένα εργαλείο της παραπάνω κατηγορίας έχει αναπτυχθεί και στο εργαστήριο υπολογιστικών συστημάτων του τμήματος «Εφαρμοσμένης Πληροφορικής» του Πανεπιστημίου Μακεδονίας. Ονομάζεται *Jdeodorant* [9], είναι υλοποιημένο σε γλώσσα προγραμματισμού Java και ενσωματώνεται υπό μορφή plug-in στο Eclipse. Είναι ικανό να εντοπίσει μη τετριμμένες περιπτώσεις κακών οσμών τριών ειδών σε συστήματα λογισμικού υλοποιημένα σε Java προτείνοντας ταυτόχρονα αναδόμηση

για την εξάλειψη της κάθε περίπτωσης. Οι κακές οσμές που εντοπίζονται και απομακρύνονται από το *Jdeodorant* είναι οι *Long Method, State Checking* και *Feature Envy.*

1.3. Σκοπός της εργασίας

Στα μεσαία και μεγάλα συστήματα λογισμικού, οι περιπτώσεις εμφάνισης μιας κακής οσμής είναι συνήθως τόσες πολλές που καθιστούν τη διαδικασία εξάλειψης του συνόλου αυτών εξαιρετικά χρονοβόρα. Βέβαια, η «διαγραφή» κάθε τέτοιας μεμονωμένης περίπτωσης κακής οσμής δεν επιφέρει ισοδύναμη βελτίωση στην ποιότητα του συστήματος και στην αύξηση της ευκολίας συντηρησιμότητας του. Κάποιες περιπτώσεις επηρεάζουν ελάχιστα η καθόλου τα παραπάνω χαρακτηριστικά ενώ άλλες παίζουν καθοριστικό ρόλο για την τελική διαμόρφωση τους. Συνεπώς ο σχεδιαστής – συντηρητής θα πρέπει να θέσει κάποιου είδους προτεραιότητα που να καθορίζει για ποιες περιπτώσεις είναι επιτακτικότερη ανάγκη η εξάλειψη της κακής οσμής και για ποιες η εξάλειψη τους είναι πιθανό να επιφέρει αμελητέα η καθόλου επιθυμητά αποτελέσματα.

Το πρώτο στάδιο αυτής της εργασίας αφορά στην ανάπτυξη και παρουσίαση μιας μεθοδολογίας για ταξινόμηση των «υποψήφιων» αναδομήσεων που προτείνονται για τη διαγραφή των εντοπισμένων κακών οσμών από το JDeodorant. Οι περισσότερες μεθοδολογίες που έχουν αναπτυχθεί έως σήμερα βασίζονται στην παρατήρηση μεταβολής τιμών μετρικών που επιφέρει κάθε αναδόμηση και ανάλογα με αυτή καθορίζεται η θέση της σε μια ταξινόμηση προτεραιότητας. Η διαφορά αυτής της εργασίας με τις παραπάνω προσεγγίσεις είναι η παροχή βοήθειας στο σχεδιαστή λογισμικού για τη ανάπτυξη αυτού με καλύτερες προδιαγραφές όσον αφορά μελλοντική συντήρηση του. Με άλλα λόγια η συντηρησιμότητα του λογισμικού εκλαμβάνεται ως κρίσιμός παράγοντας και στα πλαίσια αυτού γίνεται προσπάθεια για την ταξινόμηση των προτάσεων αναδομήσεων.

Η πρωτοτυπία της προτεινόμενης μεθοδολογίας πηγάζει από τη χρήση ιστορικών στοιχείων για την εξαγωγή συμπερασμάτων ως προς το για ποια τμήματα κώδικα η αναδόμηση εμφανίζεται σημαντικότερη και πιο επείγουσα. Τα ιστορικά στοιχεία που αφορούν «πακέτα» λογισμικού προέρχονται από επισκόπηση προηγούμενων

«γενιών» του ίδιου λογισμικού ή πιο πρακτικά παλαιότερων εκδόσεων. Για το σκοπό αυτό καταλυτικός είναι ο ρόλος των Software Repositories τα οποία παρέχουν σε κάθε προγραμματιστή – σχεδιαστή την πρόσβαση σε όλες τις εκδόσεις των έργων λογισμικού ανοικτού κώδικα. Οι εκδόσεις των πακέτων λογισμικού που έχουν χρησιμοποιηθεί για την εξαγωγή αποτελεσμάτων και την επαλήθευση συμπερασμάτων στα πλαίσια αυτής της εργασίας ανακτήθηκαν με τη βοήθεια τους.

Ο τελικός σκοπός της εργασίας ήταν η προγραμματιστική υλοποίηση της προτεινόμενης μεθοδολογίας για την ταξινόμηση των προτεινόμενων αναδομήσεων που απομακρύνουν τις κακές οσμές που εντοπίζονται από το JDeodorant δηλαδή τις *Long Method*, *State Checking* και *Feature Envy*. Με άλλα λόγια πρόκειται για μια επέκταση των δυνατοτήτων του JDeodorant ώστε να παρέχει και μια προτεινόμενη ταξινόμηση των αποτελεσμάτων (αναδομήσεων) για το ποιά από αυτά είναι επιτακτικότερη ανάγκη να αντιμετωπιστούν.

Πρακτικά αυτό επιτεύχθηκες με την προσθήκη εξτρά κλάσεων στο υπάρχον λογισμικό και τη μερική τροποποίηση των ήδη υλοποιημένων κλάσεων. Με την προσθήκη αυτών τα τελικά αποτελέσματα (περιπτώσεις κακών οσμών και αντίστοιχες αναδομήσεις τους) εμφανίζονται ταξινομημένα με σειρά από αυτά που εμφανίζουν μεγαλύτερη αναγκαιότητα εφαρμογής αναδόμησης προς αυτά με τη μικρότερη αντίστοιχη αναγκαιότητα. Η ταξινόμηση γίνεται με βάση έναν *παράγοντα* που υπολογίζεται (αυτόματα) για κάθε ξεχωριστή περίπτωση κακής οσμής με βάση δεδομένα που αντλούνται από παλαιότερες εκδόσεις του λογισμικού τις οποίες ο συντηρητής οφείλει να εισάγει στο workspace του Eclipse υπό τη μορφή Java Project. Ο παράγοντας αυτός υπολογίζεται λαμβάνοντας υπόψη τις παρελθοντικές αλλαγές στα τμήματα κώδικα που πάσχουν από κακή οσμή, την έκταση της αλλαγής, καθώς και την εγγύτητα της γενιάς στην οποία ανιχνεύεται σε σχέση με την τρέχουσα γενιά.

1.4. Γενικότερη φιλοσοφία της εργασίας

Η επισκόπηση των παλαιότερων εκδόσεων ενός έργου λογισμικού καθιστά δυνατή μια κάποια είδους καταγραφή της εξελικτικής πορείας ενός τμήματος κώδικα κρατώντας στοιχεία σχετικά με το πλήθος και το εύρος των αλλαγών στις οποίες έχει υποβληθεί. Οι αλλαγές τις οποίες έχει υποστεί κάθε τμήμα κώδικα είναι ο καίριας

σημασίας για τον καθορισμό της σπουδαιότητας του τμήματος αυτού για την όλη συντήρηση του λογισμικού.

Η παραπάνω παράγραφος αντικατοπτρίζει τη γενικότερη φιλοσοφία που οδήγησε στη «σύλληψη» της ιδέας της εργασίας και της μεθοδολογίας που εφαρμόστηκε. Πιο συγκεκριμένα η φιλοσοφία αυτή υποδηλώνει ότι αν για παράδειγμα μια συγκεκριμένη μέθοδος έχει υποστεί τροποποιήσεις στον κώδικα της σε αρκετές από τις εκδόσεις του λογισμικού που ανήκει, τότε αποτελεί μείζον αντικείμενο συντήρησης. Μια αναδόμηση που προτείνεται για αυτή τη μέθοδο πρέπει να καταταγεί με υψηλή προτεραιότητα ώστε να απαλλαγεί από την κακή οσμή που φέρει και να απλουστευθεί η διαδικασία συντήρησης τους. Αντίθετα, για μια μέθοδο με την οποία από την αρχή της ύπαρξης της έως και σήμερα δεν «ασχοληθεί» κανείς, η αναδόμηση του κώδικα της πιθανότατα δεν θα επιφέρει κανένα όφελος στο συντηρητή και έτσι κατατάσσεται με χαμηλή προτεραιότητα.

Η βελτίωση της ακρίβειας των αποτελεσμάτων ενισχύεται από τον συνυπολογισμό της εγγύτητας της γενιάς που εντοπίζεται μια «αλλαγή» ενός τμήματος κώδικα με την τρέχουσα γενιά του λογισμικού. Αυτό σημαίνει ότι κάθε αλλαγή δεν συμμετέχει στον τελικό υπολογισμό του παράγοντα κατάταξης με ισοδύναμη βαρύτητα. Είναι λογικό μια από αυτές η οποία ανιχνεύεται στις πιο πρόσφατες εκδόσεις του λογισμικού να είναι ισχυρότερη ένδειξη για το πόσο πιθανόν είναι να απασχολήσει το δοθέν τμήμα κώδικα το συντηρητή από μια άλλη ίσου εύρους που ανιχνεύεται σε κάποια από τις αρχικές εκδόσεις. Με άλλα λόγια οι αλλαγές που ανιχνεύονται προσμετρούνται με κάποιου είδους στάθμιση για τον τελικό υπολογισμό του ζητούμενου παράγοντα που αντικατοπτρίζει την εγγύτητα της γενιάς εντοπισμού της αλλαγής με την τρέχουσα.

1.5. Διάρθρωση της εργασίας

Η εργασία είναι χωρισμένη σε 9 κεφάλαια εκ των οποίων το πρώτο είναι εισαγωγικό και περιγράφει τη γένεση της ιδέας της εργασίας, τον τελικό σκοπό και γενικότερα το πρίσμα υπό το οποίο διεκπεραιώθηκε. Το δεύτερο κεφάλαιο περιέχει μια επισκόπηση παρόμοιων εργασιών – δημοσιεύσεων σε συνέδρια ή επιστημονικά περιοδικά για να γίνει το γνωστό το επίπεδο και η πρόοδος της έρευνας μέχρι στιγμής

στο συγκεκριμένο τομέα. Επίσης βοηθά στο να γίνει η αντιπαραβολή της εργασίας με παρόμοιες στο είδος της για να αναδειχθούν ευκρινέστερα τα σημεία στα οποία διαφοροποιείται καθώς και το τι πρωτότυπο εισάγει. Η γενικότερη μεθοδολογία που εφαρμόζεται ανεξαρτήτως κακής οσμής παρατίθεται στο κεφάλαιο 3. Τα κεφάλαια 4, 5, 6 συγκεκριμενοποιούν και αναλύουν σε μεγαλύτερο βάθος τη μεθοδολογία για τις κακές οσμές Long Method, State Checking και Feature Envy αντίστοιχα. Στο 7^{ο} κεφάλαιο γίνεται μια συνοπτική ανάλυση και περιγραφή της τεχνικής που ακολουθήθηκε για τη σύνταξη του κώδικα που υλοποιεί προγραμματιστικά τη μεθοδολογία. Στο κεφάλαιο 8 γίνεται μια αρχική προσπάθεια αξιολόγησης των αποτελεσμάτων από κάποιον ανεξάρτητο «ειδικό» και παρατίθενται τα ευρήματα αυτής. Τέλος τα συμπεράσματα της όλης προσέγγισης και οι μελλοντικές βλέψεις αποτελούν το αντικείμενο του κεφαλαίου 9.

2.ΕΠΙΣΚΟΠΙΣΗ ΣΧΕΤΙΚΩΝ ΕΡΓΑΣΙΩΝ

Το ερευνητικό πεδίο *Mining Software Repositories* (MSR) [12] έχει ως κύριο χαρακτηριστικό την εξέταση ιστορικών στοιχείων δηλαδή προηγούμενων εκδόσεων του λογισμικού για την πρόβλεψη πιθανών αλλαγών (change proneness) σε επικείμενες εκδόσεις του λογισμικού. Η σχετική βιβλιογραφία έχει να επιδείξει αρκετές εργασίες μια και είναι ένα πεδίο που απασχολήσει πολλούς ερευνητές. Οι περισσότερες από αυτές παρουσιάζουν μια μεθοδολογία για την υπόδειξη του τρόπου ή τον υπολογισμό της πιθανότητας με την οποία μπορεί να προβλεφθεί μια αλλαγή ή ποιό τμήμα κώδικα είναι πιθανότερο να τροποποιηθεί σε μια νεότερη έκδοση του λογισμικού.. Παρόλα αυτά, καμία από αυτές δεν έχει προταθεί υπό το πρίσμα της συντήρησης, καμία από αυτές δεν προσπαθεί να διευκολύνει και να κατατοπίσει το συντηρητή και πολύ περισσότερο καμία από αυτές δεν εμπλέκει αναδομήσεις κώδικα. Αυτά είναι τα πρωτότυπα στοιχεία που εισάγει η παρούσα εργασία και την διαφοροποιούν σε σχέση με τις υπόλοιπες.

Ο Zimmerman et al.[21], έχει αναπτύξει μια προσέγγιση που χρησιμοποιεί εξόρυξη κανόνων συσχέτισης (association rules mining) για να υποδείξει αρχικά τμήματα κώδικα που συσχετίζονται με κάποιο άλλο συγκεκριμένο τμήμα κώδικα. Οι κανόνες αυτοί μπορούν να περιγράψουν τη σχέση αλλαγών μεταξύ αρχείων ή μεθόδων. Εξετάζουν πλέγματα του λογισμικού που υπάρχουν σε συγκεκριμένα ερωτήματα (queries) ενώ υπολογίζουν την τελική «προβλεψιμότητα» (predictability) μέσω της «ακρίβειας» (precision) και της «ανάκλησης» (recall). Τέλος παρουσιάζουν κάποιες συσχετίσεις αλλαγών που προκύπτουν από τη μεθοδολογία τους τεκμηριώνοντας παράλληλα την ορθότητα και το υψηλό ενδιαφέρον τους.

Παρόμοια με την παραπάνω εργασία είναι και αυτή των Ying *et al.* [20] η οποία εισάγει στον τομέα μια προσέγγιση με διπλή συνεισφορά. Αυτή είναι από τη μία η περιγραφή της χρησιμότητα της εξόρυξης πλεγμάτων αλλαγών (change pattern mining) επικυρώνοντας τα ευρήματα της αντίστοιχης εργασίας του Zimmerman καθώς και μια εισαγωγή – παρουσίαση ενός συνόλου κριτηρίων – ενδιαφέροντος (interestingness criteria) για την εκτίμηση της χρησιμότητας των προτεινόμενων πλεγμάτων αλλαγών. Η μεθοδολογία χωρίζεται σε 3 στάδια. Στο πρώτο εξάγονται τα δεδομένα με τη βοήθεια Configuration Management Systems (CMS) και γίνεται

σχετική προεργασία. Το δεύτερο περιλαμβάνει την εξόρυξη των κανόνων με τη βοήθεια ενός αλγορίθμου εξόρυξης κανόνων. Στο τελευταίο προτείνονται σχετικά αρχεία κώδικα πάνω σε μια τροποποίηση εκφράζοντας ερωτήματα πάνω σε εξορυγμένα πλέγματα αλλαγών. Η προβλεψιμότητα υπολογίζεται και πάλι με τη βοήθεια ακρίβειας και ανάκλησης. Η πρακτική εφαρμογή της όλης μεθοδολογίας γίνεται στα συστήματα Mozilla (C++) και Eclipse (Java). Προφανώς ο σκοπός και των δυο παραπάνω προσεγγίσεων διαφέρει αρκετά από το σκοπό αυτής της εργασίας.

Μεγάλο βαθμό συνάφειας με την παρούσα εργασία έχει η δουλειά των Girba *et al.* [8] οι οποίοι χρησιμοποιούν επίσης προηγούμενες εκδόσεις του ίδιου λογισμικού για να προτείνουν ένα καλό σημείο έναρξης για *reverse engineering*. Η γενική φιλοσοφία της προσέγγισης τους βασίζεται στην παραδοχή ότι τα τμήματα κώδικα που έχουν αλλάξει περισσότερες φορές στο παρελθόν είναι οι βασικότεροι υποψήφιοι και για μελλοντικές αλλαγές. Η ύπαρξη αλλαγών εντοπίζεται αποκλειστικά σε επίπεδο κλάσης θεωρώντας ως αλλαγή την απλή πρόσθεση ή αφαίρεση μιας μεθόδου. Τελικά χρησιμοποιούν τη μεθοδολογία που ονομάζουν «*Yesterday's weather*» για να εκφράσουν μέσω αναδρομικής παρατήρησης ως ποσοστό το φαινομένου ότι τουλάχιστον μια κλάση που έχει υποστεί «ευρείες» αλλαγές στο παρελθόν ανήκει στο σύνολο των κλάσεων που είναι οι ισχυρότεροι υποψήφιοι για μελλοντικές αλλαγές. Η εργασία περιλαμβάνει και πρακτική εφαρμογή της προσέγγισης σε δυο πραγματικά συστήματα λογισμικού. Η σημαντική διαφοροποίηση της παρούσας προσέγγισης με την παραπάνω είναι ότι εξετάζει αλλαγές και σε επίπεδο «χαμηλότερο» της κλάσης και ότι δεν προβλέπει μελλοντικές αλλαγές αλλά υποδεικνύει τα τμήματα κώδικα που αποτελούν συχνότερα αντικείμενα συντήρησης.

Μια μικρή διαφοροποίηση σε σχέση με τις προηγούμενες εργασίες παρατηρείται στο σκοπό της εργασίας των Graves *et al.* [16] και προέρχεται από το γεγονός ότι η προσέγγιση αποσκοπεί στην πρόβλεψη σφαλμάτων σε μελλοντικές εκδόσεις του λογισμικού. Χρησιμοποιούν ως βάση των υπολογισμών δεδομένα από προηγούμενες εκδόσεις και όχι της μετρικές του τρέχοντος συστήματος. Επίσης γίνεται μια στάθμιση των μετρήσεων που εξάγονται από κάθε «γενιά» ανάλογα με το χρονικό «βάθος» που εντοπίζονται έτσι ώστε να προσμετρούν τις πιο πρόσφατες μετρήσεις με μεγαλύτερη βαρύτητα στο τελικό αποτέλεσμα. Θα αναλυθεί διεξοδικά σε επόμενο κεφάλαιο ότι και η δικιά μας εργασία διέπεται από μια ανάλογη φιλοσοφία.

Οι Kagdi και Maletic [13] κάνουν μια γενικότερη επισκόπηση των δυο κλάδων που αναπτύσσουν μεθοδολογίες σχετικά την πρόβλεψη αλλαγών σε ένα σύστημα δηλαδή την *Ανάλυση Συνεπειών* (Impact Analysis) και των *Mining Software Repositories*. Συνηγορούν υπέρ της συνδυαστικής χρήσης τους για την εξαγωγή αποτελεσμάτων μεγαλύτερης ακρίβειας που αφορούν την πιθανότητα αλλαγής. Η δουλεία τους δεν περιέχει πρακτική εφαρμογή σε πραγματικά συστήματα ούτε αριθμητικά αποτελέσματα. Παρόλα αποτελεί αποφέρει χρήσιμα συμπεράσματα και συντελεί στη καθοδήγηση σχετικών ερευνών που ανήκουν και στους δυο επιμέρους κλάδους.

3.ΜΕΘΟΔΟΛΟΓΙΑ

Η προσέγγιση που προτείνεται στην παρούσα εργασία βασίζεται, όπως έχει αναφερθεί, στον υπολογισμό για κάθε περίπτωση κακής οσμής ενός παράγοντα που προκύπτει από στοιχεία αντλούμενα από παλαιότερες εκδόσεις του λογισμικού. Ο παράγοντας αυτός ονομάζεται *Παράγοντας Ιστορικής Μεταβολής* (Historical Variation Factor) και συμβολίζεται από τα αρχικά της αγγλικής του ονομασίας με *HVF*. Αφού γίνει ο εντοπισμός των αλλαγών που έχει υποστεί ένα συγκεκριμένο τμήμα κώδικα (συνήθως πρόκειται για μέθοδο) στο οποίο προφανώς εντοπίζεται η κακή οσμή, για τον υπολογισμό του *HVF* λαμβάνονται υπόψη:

1) Το πλήθος των αλλαγών που εντοπίστηκαν
2) Το εύρος κάθε αλλαγής
3) Η εγγύτητα της έκδοσης λογισμικού που εντοπίζεται κάθε αλλαγή με την αντίστοιχη τρέχουσα έκδοση.

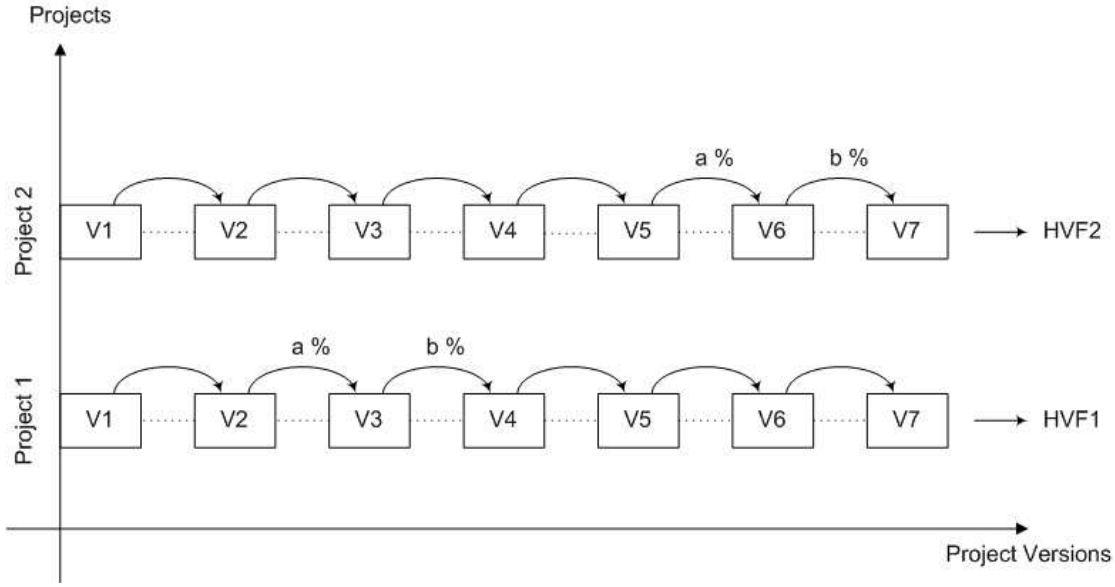
Στο κεφάλαιο αυτό αναπτύσσεται ο τρόπος με τον οποίο συμμετέχουν και αξιοποιούνται στον υπολογισμό του *HVF* οι παράγοντες 1 και 3 από τους παραπάνω. Το εύρος της αλλαγής είναι κάτι που εξαρτάται από τη φύση της κακής οσμής και επομένως ορίζεται διαφορετικά για κάθε μια από τις τρεις ξεχωριστές περιπτώσεις κακών οσμών που εξετάζονται στην εργασία. Για το λόγο αυτό αποτελεί αντικείμενο επόμενων κεφαλαίων που αποσκοπούν στην εξέταση κάθε κατηγορίας κακής οσμής.

Κάθε αλλαγή σε ένα τμήμα κώδικα που εντοπίζεται σε μια έκδοση λογισμικού συσχετίζεται καλύτερα με τη *μετάβαση* από τη μια έκδοση στην άλλη παρά στην ίδια την έκδοση αφού για να οριστεί απαιτείται η συμμετοχή δυο διαδοχικών εκδόσεων. Από εδώ και στο εξής κάθε αλλαγή θα αναφέρεται στη μετάβαση που εντοπίζεται.

Κυρίαρχο χαρακτηριστικό της μεθοδολογίας είναι ότι δεν προσμετρά ισοδύναμα τις αλλαγές που εντοπίζονται σε κάθε μετάβαση αλλά προσδίδει μια βαρύτητα ώστε οι αλλαγές που εντοπίζονται σε πιο σύγχρονες μεταβάσεις να συμμετέχουν με μεγαλύτερο μερίδιο στον υπολογισμό του *HVF*. Αυτό συμβαίνει διότι είναι λογικό οι πιο πρόσφατες εκδόσεις λογισμικού να περιέχουν ισχυρότερα και τεκμήρια για το βαθμό που έχει υποστεί συντήρηση κάθε κομμάτι του λογισμικού. Με άλλα λόγια μια αλλαγή που εντοπίζεται σε κάποια από τις τελευταίες στην χρονική ακολουθία

μεταβάσεις συμμετέχει περισσότερο στη διαμόρφωση της τιμής του *HVF* συγκριτικά με μια αλλαγή ίσου εύρους που εντοπίζεται όμως σε κάποια από τις αρχικές μεταβάσεις.

Η ιδιαιτερότητα αυτή της μεθοδολογίας που αναλύεται παραπάνω αναπαρίσταται γραφικά στο σχήμα 1. Σε αυτό παραθέτονται οι ακολουθίες των εκδόσεων για 2 διαφορετικά συστήματα λογισμικού. Εντελώς τυχαία και χωρίς να επηρεάζει το συλλογισμό έχει επιλέγει ο αριθμός 7 για το πλήθος των εκδόσεων. Σε αμφότερα τα συστήματα έχουν εντοπιστεί 2 αλλαγές ίσου εύρους (a% και b%) αλλά σε διαφορετικές μεταβάσεις. Λόγω του ότι η βαρύτητα των πιο πρόσφατων μεταβάσεων είναι μεγαλύτερη τελικά προκύπτει $HVF_2 > HVF_1$.



Σχήμα 1.

Η αποτύπωση αυτής της ιδέας στους μαθηματικούς υπολογισμούς γίνεται με τον πολλαπλασιασμό κάθε αλλαγής με ένα συντελεστή που εκφράζει την εγγύτητα της έκδοσης (μετάβασης) όπου παρατηρήθηκε ως προς την τρέχουσα έκδοση του λογισμικού. Δηλαδή σε κάθε μετάβαση για την οποία εντοπίζονται αλλαγές αντιστοιχεί ένας τέτοιος συντελεστής. Οι τιμή του συντελεστή αυτού δεν μπορεί να είναι μοναδική και προκαθορισμένη για κάθε πρόβλημα αλλά εξαρτάται από το πλήθος των εκδόσεων που είναι διαθέσιμες προς εξέταση.

Στην παρούσα εργασία χρησιμοποιείται η τεχνική *weighted moving average* [19] η οποία είναι μια υποκατηγορία της γενικότερης τεχνικής *moving average.* Η τελευταία χρησιμοποιείται κυρίως για μελλοντική πρόβλεψη τιμών μεγεθών λαμβάνοντας υπόψη ως κρίσιμο παράγοντα τη μέση τιμή ενός καθορισμένου πλήθος παλαιότερων δεδομένων. Η

weighted moving average υπολογίζει επίσης μια μορφή μέσου όρου με την ιδιαιτερότητα ότι πολλαπλασιάζει τα παλαιότερα δεδομένα με συντελεστές βαρύτητας έτσι ώστε τα πιο πρόσφατα δεδομένα να συμμετέχουν περισσότερο στους υπολογισμούς σε σύγκριση με δεδομένα χρονικά απομακρυσμένα κάτι που ταυτίζεται απόλυτα με το σκοπό της μεθοδολογίας της παρούσης εργασίας. Η συνήθης ονομασία τέτοιου είδους συντελεστών στην πλειοψηφία των περιπτώσεων που χρησιμοποιούνται ονομάζονται «βάρη».

Για τη διευκόλυνση της μαθηματικής μοντελοποίησης της μεθοδολογίας υποθέτουμε ότι οι ξεχωριστές διαδοχικές εκδόσεις λογισμικού που εμπλέκονται είναι *n+1*. Όπως έχει ήδη αναλυθεί, απαραίτητος και πιο χρήσιμος πρακτικά είναι ο αριθμός των *μεταβάσεων*. Για *n+1* εκδόσεις λογισμικού ο αντίστοιχος αριθμός των μεταβάσεων είναι κατά 1 μικρότερος δηλαδή *n* κάτι που αιτιολογείται από την απουσία έκδοσης «προγόνου» για την πρώτη (αρχική) έκδοση. Κάθε μετάβαση απαιτεί την ύπαρξη 2 εκδόσεων (διαδοχικών) και έτσι τα ζεύγη διαδοχικών εκδόσεων για *n+1* εκδόσεις είναι *n*.

Ο υπολογισμός των συντελεστών που προκύπτουν με βάση την τεχνική *weighted moving average* γίνεται ως εξής: Αν *i = 1,2,3…n* είναι ένας δείκτης εκφράζει τον αριθμό της μετάβασης μεταξύ δυο διαδοχικών εκδόσεων *i-1* και *i* (με τον ίδιο δείκτη συμβολίζονται και οι εκδόσεις με την προϋπόθεση ότι η αρίθμηση τους ξεκινά από την έκδοση 0) τότε ο συντελεστής για κάθε μετάβαση υπολογίζεται από τη σχέση:

$$w_i = \frac{i}{\sum_{i=1}^{n} i} = \frac{i}{n(n+1)/2} \quad (1)$$

Ουσιαστικά οι τιμές των συντελεστών όπως διαμορφώνονται από τη σχέση (1) ακολουθούν αριθμητική πρόοδο καθώς κάθε μια από αυτές προκύπτει από την προηγούμενη με την πρόσθεση ενός σταθερού παράγοντα. Επίσης κυμαίνονται στο διάστημα (0,1] δηλαδή $w_i \sim (0,1]$. Ο w_i γίνεται ίσος με τη μονάδα όταν υπάρχει μόνο μια μετάβαση ή αλλιώς *i=1* κάτι που πρακτικά δεν συμβαίνει καθώς η ύπαρξη μόνο μιας μετάβασης ή 2 εκδόσεων λογισμικού είναι αδύνατο να δώσει αξιόπιστα αποτελέσματα. Το άθροισμα όλων των συντελεστών είναι επίσης 1 δηλαδή $\sum_{i=1}^{n} w_i = 1$. Οι δυο τελευταίες συνθήκες αποτελούν μια μορφή κανονικοποίησης, κάτι επιθυμητό για τη διασφάλιση της ακεραιότητας και της μη αλλοίωσης των δεδομένων.

Αριθμητικό Παράδειγμα 1:

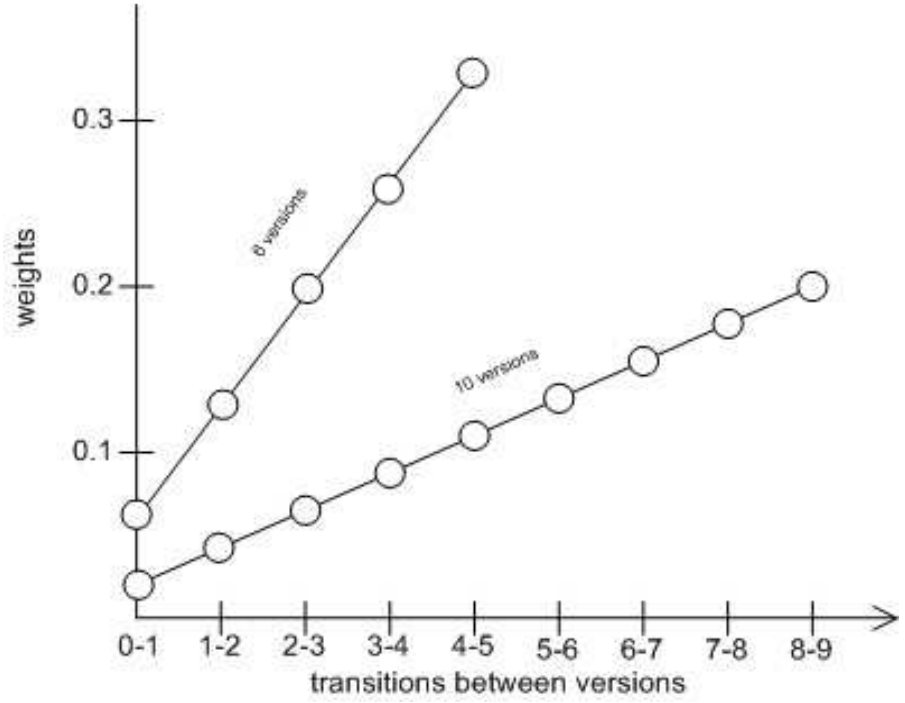
Αν υποθέσουμε ότι είναι διαθέσιμες για εισαγωγή στο workspace του Eclipse 7 εκδόσεις ενός συγκεκριμένου λογισμικού τότε οι αντίστοιχες μεταβάσεις όπως έχει εξηγηθεί είναι κατά 1 λιγότερες δηλαδή 6. Τα «βάρη» που θα υπολογιστούν αυτόματα από το εργαλείο είναι :

$w_1 = \frac{1}{\sum_{i=1}^{6} i} = \frac{1}{21} = 0.0476,$

$w_2 = \frac{2}{\sum_{i=1}^{6} i} = \frac{2}{21} = 0.0952,$

και ομοίως $w_3 = 0.1429$, $w_4 = 0.1905$, $w_5 = 0.2381$, $w_5 = 0.2857$.

Με τρόπο ανάλογο του παραδείγματος 1 μπορούν να υπολογιστούν οι ζητούμενοι συντελεστές για οποιοδήποτε αριθμό διαθέσιμων εκδόσεων. Δεν υπάρχει κάποιο συγκεκριμένο πλήθος εκδόσεων που να θεωρείται ενδεδειγμένο για την εξαγωγή «καλύτερων» αποτελεσμάτων αλλά προφανώς όσο περισσότερες εκδόσεις εισαχθούν, τόσο περισσότερη πληροφορία γίνεται εκμεταλλεύσιμη και επομένως τα αποτελέσματα γίνονται πιο αξιόπιστα.



Σχήμα 3. Κατανομή των βαρών για 6 και 10 εκδόσεις λογισμικού αντίστοιχα.

Το σχήμα 2 απεικονίζει την κατανομή των βαρών για 6 και 10 εκδόσεις αντίστοιχα. Μέσω αυτού γίνεται αντιληπτό ότι όσο μεγαλύτερο το πλήθος των

εκδόσεων τόσο περισσότερο οι τιμές των συντελεστών τείνουν να συγκλίνουν. Αντίθετα για μικρό πλήθος εκδόσεων οι διαφοροποιήσεις στις τιμές των συντελεστών είναι έντονες. Δηλαδή μια αλλαγή που ανιχνεύεται στην πρώτη μετάβαση θα πολλαπλασιαστεί με σημαντικά χαμηλότερο συντελεστή σε σχέση με μια που θα εντοπισθεί στην τελευταία.

Οι συντελεστές που υπολογίζονται με τον τρόπο που αναλύεται παραπάνω, απλά εκφράζουν την εγγύτητα μιας εντοπισμένης αλλαγής με την τρέχουσα έκδοση του λογισμικού. Για να αποτυπωθεί πλήρως η πληροφορία σχετικά με το βαθμό που έχει αποτελέσει αντικείμενο συντήρησης ένα τμήμα κώδικα είναι απαραίτητο να ορισθεί και το *εύρος της αλλαγής* (extent of change - *ec*) για κάθε μετάβαση. Όπως έχει ήδη εξηγηθεί, ο τύπος και το εύρος της αλλαγής εξαρτάται από τη φύση κάθε κακής οσμής και πιο συγκεκριμένα από το επίπεδο κώδικα που επηρεάζεται από αυτήν. Για το λόγο αυτό ο ακριβής ορισμός του εύρους της αλλαγής παραθέτεται στις παρακάτω ενότητες όπου γίνεται λεπτομερής ανάλυση για κάθε κακή οσμή ξεχωριστά. Ο τελικός παράγοντας για κάθε προτεινόμενη αναδόμηση υπολογίζεται από τη σχέση:

$$HVF = \sum_{i=1}^{n} w_i \times ec_i \qquad (2)$$

Το *i* είναι ένας δείκτης που αναπαριστά όπως και στην περίπτωση των βαρών την κάθε μετάβαση.

4. LONG METHOD

4.1. Σύντομη περιγραφή της κακής οσμής

Η κακή οσμή “Long Method” εμφανίζεται σε μεθόδους που η υλοποίηση τους συγκρούεται με τη γενικότερη φιλοσοφία του αντικειμενοστραφούς προγραμματισμού που επιτάσσει την κατασκευή όσο το δυνατόν μικρότερων μεθόδων. Παρόλα αυτά η ύπαρξη της δεν αφορά μεθόδους των οποίων ο κώδικάς που περιλαμβάνουν απλώς ξεπερνά ένα συγκεκριμένο αριθμό γραμμών. Σε αναλογία με την αρχή της μοναδικής αρμοδιότητας που ορίζεται για κλάσεις, έτσι και η κάθε μέθοδος πρέπει να έχει έναν μοναδικό και αποκλειστικό σκοπό. Σε περίπτωση που κάτι τέτοιο δεν ισχύει για κάποια συγκεκριμένη μέθοδο, τότε «πάσχει» από την παραπάνω κακή οσμή.

Οι μέθοδοι που εμφανίζουν το πρόβλημα σχεδίασης Long Method είναι συνήθως «μεγάλα» τμήματα κώδικα με υψηλή πολυπλοκότητα [7] και χαμηλή συνοχή [15], με αποτέλεσμα να απαιτούν σημαντική προσπάθεια και χρόνο για την κατανόηση, την απασφαλμάτωση, τον έλεγχο και τη συντήρηση τους. Η συντριπτική πλειοψηφία αυτών των περιπτώσεων αντιμετωπίζεται με τη εφαρμογή της αναδόμησης Extract Method [5] η οποία απλοποιεί τον κώδικα διασπώντας την αρχική «μεγάλη» μέθοδο σε «μικρότερες» που υποκαθιστούν πλήρως τη λειτουργία της.

Η κακή οσμή Long Method ανήκει σε αυτές που εντοπίζονται από το εργαλείο JDeodorant και παράλληλα για κάθε περίπτωση προτείνεται αναδόμησης της μορφής Extract Method. Η αναδόμηση αυτή χωρίζεται σε δυο βασικές κατηγορίες:

1) Εξαγωγή statements που τροποποιούν τις τιμή συγκεκριμένης μεταβλητής και κατασκευή νέας μεθόδου από αυτές [16].
2) Εξαγωγή statements που επηρεάζουν την κατάσταση αναφορών σε αντικείμενα και υφίστανται είτε ως τοπικές μεταβλητές είτε ως στοιχεία της κλάσης στη οποία ανήκει η μέθοδος (και κατασκευή νέας μεθόδου από αυτές).

Σε αμφότερες τις παραπάνω περιπτώσεις η νέα μέθοδος που δημιουργείται καλείται σε κατάλληλο σημείο του σώματος της αρχικής. Απώτερος σκοπός είναι η διασφάλιση της ικανοποιητικής λειτουργικότητας του εξαγόμενου κώδικα. Ένα αντιπροσωπευτικό παράδειγμα για κάθε μια από τις παραπάνω κατηγορίες

αναπαριστάται στο Σχήμα 2. Το τμήμα της μεθόδου που μπορεί να εξαχθεί και να σχηματίσει νέα μέθοδο περικλείεται και στις δυο περιπτώσεις από το πλαίσιο.

(α)

```
public void translate(double dx, double dy)
   {
      if (implicitParameter != null)
      {
         dx =
implicitParameter.getBounds().getCenterX() +
xoffset - getBounds().getCenterX();
      }

      Node parent = getParent();
      if (parent instanceof CallNode)
      {
         double y = getBounds().getY() + dy;
         y = Math.max(y,
parent.getBounds().getMinY());
         y = Math.min(y,
parent.getBounds().getMaxY());
         dy = y - getBounds().getY();
      }
      else if (parent == null)
      {
         double y = getBounds().getY() + dy;
         y = Math.max(y,
implicitParameter.getTopRectangle().getHeight()
               +
ImplicitParameterNode.TOP_GAPY);
         y = Math.min(y,
implicitParameter.getBounds().getMaxY());
         dy = y - getBounds().getY();
      }

      super.translate(dx, dy);
   }
```

(β)

```
public void removeSelected()
   {
      Iterator iter =
selectedItems.iterator();
      while (iter.hasNext())
      {
         Object selected =
iter.next();
         if (selected instanceof Node)
         {
            graph.removeNode((Node)
selected);
         }
         else if (selected instanceof
Edge)
         {
            graph.removeEdge((Edge)
selected);
         }
      }

      if (selectedItems.size() > 0)
setModified(true);
      repaint();
   }
```

Σχήμα 3: (α) Εξαγωγή statements για υπολογισμό μεταβλητής, (β) Εξαγωγή statements που αλλάζουν την κατάσταση αντικειμένου.

4.2. Χειρισμός των ιστορικών δεδομένων

Η ύπαρξη της κακής οσμής Long Method σε μια μέθοδο δυσχεράνει τις προσπάθειες και τις ενέργειες σχετικά με τη συντήρηση της. Για το λόγο αυτό καλό θα ήταν να αποδοθεί υψηλή προτεραιότητα στην εφαρμογή αναδόμησης σε μια μέθοδο που έχει συντηρηθεί συχνά στο παρελθόν ή με άλλα λόγια ο κώδικας της έχει τροποποιηθεί αρκετά στις προηγούμενες εκδόσεις του λογισμικού.

Η εξέταση του κώδικα **ολόκληρης της μεθόδου** και η καταγραφή του εύρους των αλλαγών από έκδοση σε έκδοση είναι ο καθοριστικός παράγοντας που οδηγεί στην εξαγωγή της επιθυμητής πληροφορίας. Αυτό σημαίνει ότι το πρώτο βήμα είναι η αναζήτηση και η εύρεση της ίδιας μεθόδου, που πάσχει από Long Method στην τελευταία έκδοση, σε όλες τις εκδόσεις του λογισμικού που έχουν εισαχθεί στο workspace του Eclipse άσχετα αν σε αυτές πάσχει ή όχι από το αντίστοιχο πρόβλημα.. Για τον καθορισμό της *ομοιότητας* των μεθόδου (δηλαδή ότι πρόκειται πράγματι για την ίδια) εξετάζονται και επαληθεύονται :

1) Η ύπαρξη κοινού ονόματος.
2) Η ύπαρξη κοινού επιστρεφόμενου τύπου.
3) Η ύπαρξη κοινών παραμέτρων (arguments)
4) Το αν ανήκει η μέθοδος και στις δυο περιπτώσεις στην ίδια κλάση.

Αφότου έχει καθοριστεί ότι η εξέταση των αλλαγών θα αφορά το πλήρες σώμα της μεθόδου, επόμενο ζητούμενο είναι ο ορισμός του εύρους της κάθε αλλαγής. Δε θα ήταν και πολύ συνετό να σημειώνεται απλά και μόνο το αν συνέβη αλλαγή ή όχι δηλαδή το αποτέλεσμα για κάθε μετάβαση να λάβει δυαδική μορφή (0 ή 1), καθώς διαφορετικά πρέπει να συνυπολογιστεί μια αλλαγή που αφορά ένα μόνο statement της μεθόδου και διαφορετικά μια αλλαγή που συνεπάγεται την πλήρη τροποποίηση του σώματος της μεθόδου.

Το πρώτο βήμα πριν οριστεί το εύρος της αλλαγής είναι η εξαγωγή από μια μέθοδο όλων των statements που την αποτελούν. Αυτό σημαίνει πως και οι βρόχοι *for, while, do while,* τα blocks if, if-else καθώς και οι έλεγχοι switch αναλύονται στα statements που περιέχουν και δεν θεωρούνται ένα απλό statement.

Εφόσον έχουν ανακτηθεί όλα τα statements για την υπό εξέταση μέθοδο σε όλες τις διαθέσιμες εκδόσεις του λογισμικού τότε **για κάθε μέθοδο *i* αυτά συγκρίνονται ένα προς ένα με τα αντίστοιχα statements της ίδιας μεθόδου στην αμέσως προηγούμενη έκδοση *i*-1**. Κρατείται ο αριθμός των statements που αποδείχθηκαν κοινά στις δυο διαδοχικές μεθόδους δηλαδή αυτά που δεν τροποποιήθηκαν καθόλου. Ως τροποποίηση ενός statement θεωρείται η παραμικρή αλλαγή σε αυτό (μετονομασία μεταβλητής, αλλαγή κάποιου τελεστή, επέκταση ή αποκοπή τμήματος του κτλ.).

Το εύρος της αλλαγής ορίζεται σε δύο άξονες με τη μορφή «ποσοστού» :

1) Ποσοστό αλλαγής που οφείλεται στην προσθήκη νέων statements στην μέθοδο από την έκδοση *i*-1 στην έκδοση *i*. Συμβολίζεται με *pca* (percentage of change owed to addition) και υπολογίζεται ως ο λόγος των νέων statements μιας μεθόδου ως προς τα συνολικά της statements. Η μαθηματική σχέση που το περιγράφει είναι :

$$pca_i = \frac{\#(new\ statements\ of\ m_i)}{\#(statements\ of\ m_i)} \quad (3)$$

2) Ποσοστό αλλαγής που οφείλεται στην απομάκρυνση statements από τη μέθοδο της έκδοσης *i*-1 στη μέθοδο της έκδοσης *i*. Στην περίπτωση αυτή το ποσοστό υπολογίζεται με βάση τα statements της «παλιάς» μεθόδου δηλαδή της *i*-1. Συμβολίζεται με *pcr* (percentage of change owed to removal) και ορίζεται ως το πηλίκο των statements που έχουν αφαιρεθεί από την μέθοδο της έκδοσης *i*-1 ως προς τα συνολικά statements αυτής της μεθόδου. Δίνεται από τη μαθηματική σχέση:

$$pcr_i = \frac{\#(removed\ statements\ of\ m_{i-1})}{\#(statements\ of\ m_{i-1})} \quad (4)$$

Έτσι, για κάθε μέθοδο, σε κάθε μετάβαση αντιστοιχούν δυο ποσοστά που το καθένα εκφράζει αλλαγή από λίγο διαφορετική σκοπιά όπως εξηγείται παραπάνω. Το συνολικό ποσοστό αλλαγής προκύπτει από τη μέγιστη τιμή των δυο προαναφερθέντων ποσοστών ή αλλιώς :

$$pc_i = \max(pca_i, pcr_i) \quad (5)$$

Προφανώς η τιμή του κάθε ένα από τα δυο ποσοστά είναι κυμαίνεται στο διάστημα [0,1], κάτι επιθυμητό αφού η ύπαρξη άνω και κάτω ορίων στην διάστημα εξασφαλίζει τη συγκρισιμότητα των αλλαγών.

Αριθμητικό παράδειγμα 2:

Έστω ότι στην μέθοδο m_{i-1} υπάρχουν 10 statements και στη μέθοδο m_i να αυξήθηκαν σε 15 με τα 8 από αυτά να υπάρχουν και στην m_{i-1} ενώ τα υπόλοιπα 7 να

είναι νέες προσθήκες. Τότε τα επιμέρους ποσοστά υπολογίζονται χρησιμοποιώντας τις σχέσεις (3) και (4) ως εξής:

$pca_i = \frac{7}{15}$, και $pcr_i = \frac{10-8}{10} = \frac{2}{10}$.

Από τη σχέση (5) λαμβάνεται το συνολικό ποσοστό ως $pc_i = \max\left(\frac{7}{15}, \frac{2}{10}\right) = \frac{7}{15}$.

Το τελικό στάδιο για τον υπολογισμό του HVF είναι να πολλαπλασιαστεί το ποσοστό της αλλαγής που αφορά κάθε μετάβαση με τον αντίστοιχο συντελεστή ιστορικής βαρύτητας. Το άθροισμα των επιμέρους γινομένων δίνει την τελική τιμή του HVF κατά αναλογία με τον τύπο (2) αντικαθιστώντας το *ec* με *pc*:

$$HVF = \sum_{i=1}^{n} w_i \times pc_i \quad (6)$$

4.3. Παρατηρήσεις και σχόλια

Δεν είναι μηδαμινή η πιθανότητα της περίπτωσης για κάποια μέθοδο που εμφανίζει την κακή οσμή Long Method να μην υπάρχει σε όλες τις εκδόσεις του λογισμικού. Αυτό σημαίνει ότι δημιουργήθηκε σε κάποια ενδιάμεση έκδοση και όχι από την αρχική. Για την περίπτωση αυτή η εξέταση της ύπαρξης και του εύρους των αλλαγών έχει νόημα μόνο για τις εκδόσεις στις οποίες η μέθοδος είναι υπαρκτή. Με άλλα λόγια ο αριθμός των μεθόδων που λαμβάνονται υπόψη είναι μικρότερος από το συνολικό αριθμό των διαθέσιμων εκδόσεων. Η γενικότερη μεθοδολογία δεν διαφοροποιείται με εξαίρεση την ιδιαιτερότητα ότι σταματά στην έκδοση που για τελευταία φορά εμφανίζεται η μέθοδος εφόσον η αναζήτηση της στις εκδόσεις του λογισμικού γίνεται προς την τελική προς την αρχική. Τα ποσοστά των αλλαγών υπολογίζονται κανονικά και πολλαπλασιάζονται με τους αντίστοιχους ιστορικούς συντελεστές που αφορούν τις μεταβάσεις που αυτές σημειώθηκαν. Ο παράγοντας *HVF* υπολογίζεται από τη σχέση (6).

Αξίζει επίσης να σημειωθεί ότι στη βιβλιογραφία της τεχνολογίας λογισμικού υπάρχουν προσεγγίσεις που να ορίζουν και να υπολογίζουν την ομοιότητα μεταξύ δυο διαφορετικών τμημάτων κώδικα. Παρόλα αυτά δεν κρίθηκε ως ικανοποιητική λύση η χρήση κάποιου από αυτούς τους ορισμούς ομοιότητας για τη σύγκριση της ίδιας μεθόδου σε δυο διαδοχικές εκδόσεις. Ο λόγος είναι ότι η παρούσα εργασία είναι

υλοποιημένη υπό το πρίσμα της συντήρησης λογισμικού και η τροποποίηση, η προσθήκη ή η αφαίρεση statements από μια μέθοδο αποτελούν κυρίαρχα ζητήματα συντήρησης. Για το λόγο αυτό η χρήση του ορισμού που εισάγεται στην ενότητα 4.3 είναι πρακτικά ενδεδειγμένη.

4.4. Αποτελέσματα

Έχει ήδη αναφερθεί ότι η παραπάνω μεθοδολογία έχει υλοποιηθεί προγραμματιστικά σε γλώσσα Java και έχει ενσωματωθεί ως plug-in στο Eclipse. Πιο συγκεκριμένα, έχει προστεθεί ως extra λειτουργία στο ήδη υπάρχον εργαλείο JDeodorant. Έτσι, υπάρχει η δυνατότητα εφαρμογής του σε λογισμικό ανοικτού κώδικα για την εξαγωγή αποτελεσμάτων. Τα αποτελέσματα είναι μια λίστα κακών οσμών (long method στην περίπτωση αυτή) που εντοπίζονται στο λογισμικό, με τις προτεινόμενες αναδομήσεις τους, ταξινομημένες με σειρά προτεραιότητας ως προς το πόσο επείγουσα είναι η εφαρμογή της αναδόμησης λαμβάνοντας υπόψη τις προηγούμενες διαθέσιμες εκδόσεις. Πιο συγκεκριμένα το εργαλείο εντοπίζει όλες τις περιπτώσεις Long Method στην πιο πρόσφατη έκδοση του λογισμικού και στη συνέχεια προβαίνει στους απαραίτητους ελέγχους των προηγούμενων εκδόσεων υπολογίζοντας σε κάθε μετάβαση, για κάθε μέθοδο που υποφέρει από το πρόβλημα, το ποσοστό αλλαγής σύμφωνα με τη μεθοδολογία της ενότητας 4.2 και ασφαλώς την τελική τιμή του HVF.

Στη ενότητα αυτή παρουσιάζονται ενδεικτικά αποτελέσματα που έχουν εξαχθεί από το λογισμικό *JFreeChart* [10].. Είναι σύστημα ανοικτού κώδικα υλοποιημένο σε Java που παρέχει στο χρήστη τη δυνατότητα να δημιουργήσει ιστογράμματα (charts) επαγγελματικής ποιότητας

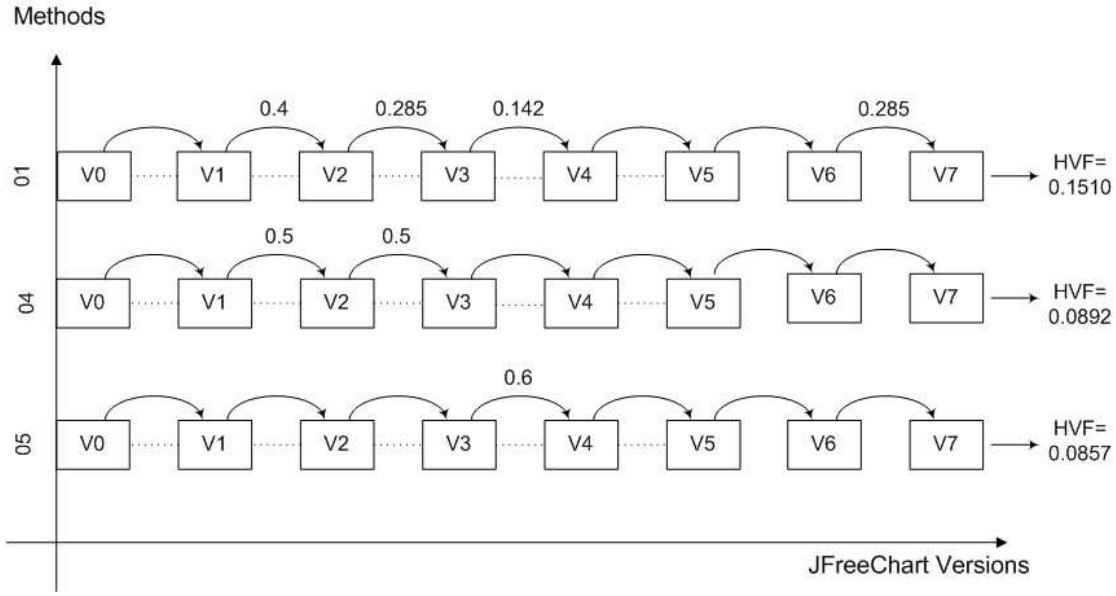
Χρησιμοποιήθηκαν 8 εκδόσεις του λογισμικού και πιο συγκεκριμένα όλες οι εκδόσεις διαδοχικά από την jfreechart-0.5.6 έως την jfreechart-0.8.0. Για να περιοριστεί ο όγκος των αποτελεσμάτων η εξέταση της ύπαρξης περιορίστηκε στο πακέτο *com.jrefinery.chart.* Συνολικά εντοπίστηκε το πρόβλημα long method σε 38 μεθόδους εκ των οποίων αλλαγές σε σχέση με τις προηγούμενες εκδόσεις του λογισμικού έχουν υποστεί οι 21. Στον πίνακα 1 περιέχονται ενδεικτικά οι 10 με το μεγαλύτερο συντελεστή *HVF* των οποίων η αναδόμηση είναι περισσότερο επιτακτική σύμφωνα με τη μεθοδολογία.

Πίνακας 1:*HVF* για κάθε μέθοδο με Long Method στο JFreeChart 0.8.0

	Method Name	**HVF**
01.	createStackedVerticalBarChart3D	0.1510
02.	createVerticalXYBarChart	0.1369
03.	drawHorizontal	0.1260
04.	createHighLowChart	0.0892
05.	drawHorizontal	0.0857
06.	paintComponent	0.0855
07.	createScatterPlot	0.0674
08.	draw	0.0670
09.	reserveHeight	0.0267
10.	setVerticalAxis	0.0238

Το όνομα μεθόδου “drawHorizontal” εμφανίζεται δυο φορές αλλά πρόκειται για δυο ξεχωριστές μεθόδους που ανήκουν σε διαφορετικές κλάσεις. Παρατηρεί κανείς ότι οι συντελεστές *HVF* λαμβάνουν σχετικά μικρές τιμές που δεν προσεγγίζουν το 1. Αυτό είναι κάτι αναμενόμενο καθώς για να εμφανιστεί αυξημένη τιμή του *HVF* που να τείνει στο 1 πρέπει μια μέθοδος να έχει υποστεί ουσιαστικά καθολικές αλλαγές σε όλες σχεδόν τις εκδόσεις του λογισμικού, κάτι που δεν είναι σύνηθες.

Εκτός από την τελική τιμή του *HVF*, το εργαλείο έχει τη δυνατότητα να καταγράφει για κάθε μέθοδο τα ενδιάμεσα αποτελέσματα δηλαδή το ποσοστό αλλαγής της συγκεκριμένης μεθόδου που αντιστοιχεί σε κάθε μετάβαση. Στο σχήμα 4 γίνεται μια γραφική αναπαράσταση της εξέλιξης τριών συγκεκριμένων μεθόδων του πίνακα 1 με ταυτόχρονη καταγραφή του ενδιάμεσου ποσοστού αλλαγής για κάθε μετάβαση. Οι μέθοδοι αυτές επελέγησαν ως πιο αντιπροσωπευτικές σχετικά με τα ενδιάμεσα ποσοστά αλλαγής. Στον κατακόρυφο άξονα σημειώνονται οι αριθμοί που αντιστοιχούν σε κάθε μέθοδο βάση του πίνακα 1 ενώ στον οριζόντιο υπάρχουν οι 8 εκδόσεις του λογισμικού που χρησιμοποιήθηκαν. Να σημειωθεί τέλος ότι για τις μεταβάσεις των οποίων το ποσοστό αλλαγής είναι μηδενικό, το τελευταίο παραλείπεται από το σχήμα.



Σχήμα 4. Ποσοστά αλλαγής τριών μεθόδων σε κάθε μετάβαση.

5. STATE CHECKING

5.1. Σύντομη περιγραφή της κακής οσμής.

Η κακή οσμή “State Checking” που είναι γνωστή και με το όνομα “switch statements” [5] εμφανίζεται όταν για τον καθορισμό μιας γραμμής εκτέλεσης χρησιμοποιούνται statements που ελέγχουν την κατάσταση ενός αντικειμένου. Ο έλεγχος αυτός γίνεται με δυο τρόπους. Ο πρώτος είναι συγκρίνοντας την τιμή μιας μεταβλητής με ένα σετ στατικών σταθερών και ο δεύτερος όταν η υποκλάση του τύπου μια αναφοράς ανακτάται κατά τη διάρκεια της εκτέλεσης του προγράμματος μέσω μηχανισμών RTTI (Run Time Type Identification).

Πρακτικά το State Checking απαντάται σε τμήματα κώδικα με statements της μορφής if-elseif ή switch που είναι διάσπαρτα σε διάφορα σημεία του κώδικα. Στην ουσία, η κακή οσμή αποτελεί μια χαμένη ευκαιρία για την υλοποίηση της λειτουργικότητας με χρήση πολυμορφισμού ή διαφορετικά συνεπάγεται την απουσία του προτύπου σχεδίασης «Στρατηγική» (State/Strategy design pattern). Επιπροσθέτως, επιφέρει την παραβίαση της αρχής ανοικτής-κλειστής σχεδίασης καθώς η επέκταση της λειτουργικότητας συγκεκριμένου τμήματος κώδικα με την προσθήκη νέας κατάστασης απαιτεί την τροποποίηση του υπάρχοντος κώδικα, σε αντίθεση με την προαναφερθείσα αρχή που συνιστά σε ανάλογη περίπτωση αποκλειστικά την προσθήκη νέου κώδικα. Αρνητικές συνέπειες όλων των παραπάνω είναι η επιβάρυνση της προσπάθεια συντήρησης του λογισμικού και η αύξηση της πιθανότητας εισαγωγής λαθών.

Το JDeodoarnt εντοπίζει σε λογισμικό υλοποιημένο σε Java την ύπαρξη περιπτώσεων State Checking και προτείνει αναδομήσεις που αντικαθιστούν το τμήμα κώδικα που εμφανίζει την κακή οσμή με εισαγωγή πολυμορφισμού [18]. Ο εντοπισμός γίνεται με την αναζήτηση statements που είτε επιλέγουν γραμμή εκτέλεσης ελέγχοντας την τρέχουσα κατάσταση ενός αντικειμένου από ένα set σταθερών, είτε ανακτούν την πραγματική υποκλάση ενός αντικειμένου μέσω μηχανισμών RTTI (Run Time Type Identification).

Στο σχήμα 5 γίνεται οπτική αναπαράσταση τμήματος κώδικα που εμφανίζει State Checking σε συνδυασμό με την αναδόμηση που προτείνεται από το JDeodorant. Ο

έλεγχος του σετ των στατικών μεταβλητών (α) αντικαταστάθηκε από το πρότυπο State/Strategy (β). Στο (β) εκτός από την τροποποίηση της αρχικής κλάσης φαίνεται

(α) αρχικός κώδικας

```
public class GraphPanel extends JPanel{

private int dragMode;
private static final int DRAG_NONE = 0;
private static final int DRAG_MOVE = 1;
private static final int DRAG_RUBBERBAND =
2;
private static final int DRAG_LASSO = 3;
private static final int GRID = 10;

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

 if (dragMode == DRAG_RUBBERBAND)
      {
         Edge prototype = (Edge) tool;
         Edge newEdge = (Edge)
prototype.clone();

if(mousePoint.distance(mouseDownPoint) >
CONNECT_THRESHOLD
&& graph.connect(newEdge, mouseDownPoint,
mousePoint))
         {
            setModified(true);
            setSelectedItem(newEdge);
         }
      }

      else if (dragMode == DRAG_MOVE)
      {
         graph.layout();
         setModified(true);
      }

      dragMode = DRAG_NONE;

      revalidate();
      repaint();
   }
}
```

(β) αναδομημένος κώδικας

```
public class GraphPanel extends JPanel{

private int dragMode;
private static final int DRAG_NONE = 0;
private static final int DRAG_MOVE = 1;
private static final int DRAG_RUBBERBAND =
2;
private static final int DRAG_LASSO = 3;
private static final int GRID = 10;
private DragMode dragMode = new DragNone();

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

      setDragMode(DRAG_NONE);

      revalidate();
      repaint();
   }
}
```

Αρχική κλάση σε ρόλο «πλαισίου»

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

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

Αφηρημένη κλάση σε ρόλο State

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

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

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

Υποκλάση State που αντιπροσωπεύει το *DRAG_MOVE*

Σχήμα 5. Παράδειγμα κώδικα με State Checking

και η αφηρημένη κλάση που δημιουργείται και αναπαριστά την «κατάσταση ελέγχου» και μια από τις υποκλάσεις που δημιουργούνται εκφράζοντας κάθε μια από τις στατικές μεταβλητές.

5.2. Χειρισμός των ιστορικών δεδομένων.

Η αντιμετώπιση του προβλήματος σχεδίασης State Checking δεν ξεφεύγει από το γενικότερο πλαίσιο φιλοσοφίας της όλης εργασίας. Συνεπώς η ταξινόμηση των υποψήφιων αναδομήσεων γίνεται με τέτοιο τρόπο ώστε να αποδοθεί υψηλότερη προτεραιότητα σε αυτές που αφορούν τμήματα κώδικα που έχουν τροποποιηθεί περισσότερο σε προηγούμενες εκδόσεις του λογισμικού. Με άλλα λόγια, για τα τμήματα κώδικα των οποίων η συχνότητα συντήρησης τους είναι αυξημένη με βάση πάντα ιστορικά στοιχεία η ανάγκη αναδόμησης τους φαντάζει περισσότερο επιτακτική, έτσι ώστε να αναδομημένος κώδικας να συμβαδίζει με τις αρχές και τα πρότυπα σχεδίασης, απλουστεύοντας τη διαδικασία συντήρησης τους.

Από τα παραπάνω γίνεται προφανές ότι το επίπεδο εξέτασης κώδικα στην περίπτωση του State Checking δεν είναι ολόκληρη μέθοδος αλλά μόνο το τμήμα κώδικα που περιλαμβάνει το σύνολο των statements που είναι υπεύθυνα για τους ελέγχους δηλαδή οι εκφράσεις που περιέχονται στα if-elseif ή switch. Οι τροποποιήσεις που πιθανόν να έχουν γίνει στο υπόλοιπο σώμα μιας μεθόδου (εκτός αυτού που εμφανίζει State Checking) θεωρούμε ότι δεν επηρεάζονται από της ύπαρξη της κακής οσμής.

Ο ορισμός καθώς και το εύρος των αλλαγών ορίζονται σε δυο άξονες:

1. Μεταβολή (προσθήκη, αντικατάσταση, μείωση) των ελεγχόμενων καταστάσεων του αντικειμένου για την επιλογή της γραμμής εκτέλεσης στο συγκεκριμένο τμήμα κώδικα που εμφανίζει State Checking κατά τη μετάβαση από μια έκδοση έως την επόμενη.

Πιο συγκεκριμένα αν ο έλεγχος στην έκδοση *i*-1 γινόταν για *n* σταθερές και στην έκδοση *i* έγινε *n*+1 αυτό συνεπάγεται αύξηση των ελεγχόμενων καταστάσεων και καταγράφεται ως αλλαγή. Το είδος αυτό της αλλαγής ονομάζεται "*Addition of new branch*" και συμβολίζεται με *anb*. Στην περίπτωση αυτή δεν έχει κάποιο νόημα να

αποτυπωθεί η αλλαγή ως ποσοστό διότι η καταβολή προσπάθειας σε μη «ορθά» πλαίσια είναι η ίδια είτε έχει προστεθεί ένα είτε περισσότερα expressions ελέγχου. Άρα ο τύπος της μεταβλητής *anb* είναι bollean και λαμβάνει τιμή 1 όταν έχει εντοπιστεί αλλαγή και 0 όταν δεν έχει εντοπισθεί. Η αλλαγή καταγράφεται και στην περίπτωση αντικατάστασης ενός expression από ένα άλλο καθώς και στην περίπτωση απλής εξάλειψης του (μείωση των ελεγχόμενων καταστάσεων) αν και ειδικά η τελευταία περίπτωση είναι εξαιρετικά σπάνια. Συνοπτικά η σχέση που διαμορφώνει την τιμή της *anb* για τη μετάβαση ή αντίστοιχα την έκδοση *i* είναι :

$$anb_i = \begin{cases} 1, \alpha\nu\ \acute{\epsilon}\chi\epsilon\iota\ \alpha\lambda\lambda\acute{\alpha}\zeta\epsilon\iota\ \tau o\ \pi\lambda\acute{\eta}\theta o\varsigma \\ \tau\omega\nu\ \epsilon\lambda\epsilon\gamma\chi\acute{o}\mu\epsilon\nu\omega\nu\ \kappa\alpha\tau\alpha\sigma\tau\acute{\alpha}\sigma\epsilon\omega\nu. \\ 0, \alpha\lambda\lambda\iota\acute{\omega}\varsigma. \end{cases} \quad (7)$$

2. Αύξηση του βαθμού χρήσης του ελέγχου κατάστασης του ίδιου αντικειμένου στην πλήρη έκταση του κώδικα του συστήματος.

Η ύπαρξη και η αιτιολόγηση αυτού του άξονα αλλαγής γίνονται αντιληπτές αν σκεφτεί κανείς ότι είναι δυνατόν κατά την εξέλιξη του λογισμικού να χρησιμοποιείται ο ίδιος έλεγχος καταστάσεων για την υλοποίηση περισσότερων του ενός λειτουργιών. Για το λόγο αυτό καταγράφεται το πλήθος των διαφορετικών σημείων που εντοπίζεται ο ίδιος έλεγχος καταστάσεων στην έκδοση *i* και συγκρίνεται με το αντίστοιχο πλήθος στην έκδοση *i*-1. Ο τύπος της αλλαγής αυτής ονομάζεται "*Increase of system occurrences*" και συμβολίζεται με *iso*. Και στην περίπτωση αυτή ο τύπος της μεταβλητής *iso* είναι bollean, λαμβάνει δηλαδή τιμές 0 ή 1. Η τιμή 1 αντιστοιχεί στην ύπαρξη αλλαγής ενώ η τιμή 0 στην περίπτωση που δεν εντοπίστηκε αύξηση του «βαθμού χρήσης» του ίδιου ελέγχου καταστάσεων. Η παραπάνω υπολογιστική τεχνική περιγράφεται συνοπτικά από τη μαθηματική σχέση:

$$iso_i = \begin{cases} 1, \gamma\iota\alpha\ \alpha\acute{\upsilon}\xi\eta\sigma\eta\ \tau o\upsilon\ \beta\alpha\theta\mu o\acute{\upsilon}\ \chi\rho\acute{\eta}\sigma\eta\varsigma. \\ 0, \alpha\lambda\lambda\iota\acute{\omega}\varsigma. \end{cases} \quad (8)$$

Αν και η πιθανότητα μείωσης του βαθμού χρήσης σε κάποια έκδοση κρίνεται απειροελάχιστη, η μεθοδολογία δεν την αγνοεί. Αυτό γίνεται με τη σύγκριση των ξεχωριστών περιπτώσεων ελέγχου που καταγράφονται σε κάθε έκδοση με τον

αντίστοιχο μέγιστο αριθμό που έχει εντοπιστεί έως την εξεταζόμενη έκδοση. Μόνο αν ξεπερνά αυτό το μέγιστο αριθμό σημειώνεται αλλαγή, ειδάλλως η *iso* λαμβάνει την τιμή 0.

Το αποτέλεσμα της παραπάνω μεθοδολογίας είναι να αντιστοιχίζονται ζεύγη δυαδικών μεταβλητών για κάθε μετάβαση που υποδεικνύουν την ύπαρξη ή όχι αλλαγής σε κάθε ένα από τους δυο άξονες ξεχωριστά. Εφόσον και οι δυο άξονες θεωρούνται ότι συμμετέχουν με την ίδια βαρύτητα στις εργασίες συντήρησης και γενικότερα αλλαγών στον κώδικα, το συνολικό εύρος της αλλαγής extent of change - *ec* που αφορά την κακή οσμή State Checking επιβάλλεται να συμπεριλαμβάνει και τις δυο. Αυτό γίνεται θεωρώντας ως συνολικό εύρος αλλαγής το αλγεβρικό άθροισμα των μεταβλητών *anb* και *iso*:

$$ec_i = anb_i + iso_i \tag{9}$$

Σύμφωνα με τη σχέση (9) αν για μια περίπτωση αναγνωρισμένου προβλήματος State Checking σε κάποια μετάβαση έχει εντοπισθεί αλλαγή σε έναν από τους δυο άξονες τότε το συνολικό εύρος αλλαγής θα είναι 1. Αν για την ίδια περίπτωση εντοπισθούν αλλαγές και στους δυο άξονες τότε το εύρος αλλαγής θα είναι 2, κάτι φυσιολογικό διότι με αυτόν τον τρόπο καταδεικνύεται ότι η αλλαγή που αφορά τη δεύτερη κατάσταση είναι «μεγαλύτερη» από την πρώτη. Θα ήταν ανυπόστατος χειρισμός αν και στη δεύτερη κατάσταση το εύρος αλλαγής ισοδυναμούσε με 1 διότι έτσι θα συνέβαινε απώλεια πληροφορίας.

Το τελευταίο στάδιο υπολογισμών περιλαμβάνει τον υπολογισμό του *HVF* για κάθε ξεχωριστή περίπτωση State Checking που εντοπίζεται στον κώδικα του λογισμικού και γίνεται προφανώς με τη χρήση της σχέσης (2). Επειδή όμως είναι επιθυμητό τα αποτελέσματα για κάθε πρόβλημα να κυμαίνονται στο διάστημα [0,1] για χάρη διευκόλυνσης της συγκρισιμότητας τους, το τελικό εύρος αλλαγής για κάθε μετάβαση πρέπει να κανονικοποιηθεί. Έτσι από τη σχέση (9) διαιρώντας το εύρος της αλλαγής με το 2 - την πιθανή μέγιστη τιμή του, λαμβάνεται τελικά το κανονικοποιημένο εύρος αλλαγής *nec* (normalized extent of change) :

$$nec_i = \frac{1}{2} \cdot ec_i \tag{10}$$

Το κανονικοποιημένο εύρος της αλλαγής είναι αυτό που εισέρχεται τελικά στη σχέση (2) για τον υπολογισμό του *HVF*.

5.3. Αποτελέσματα.

Η τρόπος εξαγωγής αποτελεσμάτων δεν διαφέρει από την περίπτωση της κακής οσμής Long Method. Βασίζεται στη χρήση του τροποποιημένου ημιαυτόματου εργαλείου JDeodorant με την εφαρμογή του στην πιο πρόσφατη έκδοση του λογισμικού για τον αρχικό εντοπισμό των διακριτών περιπτώσεων State Checking σε αυτή. Εφόσον στο workspace του Eclipse έχουν εισαχθεί και παλαιότερες εκδόσεις του λογισμικού, το εργαλείο αυτόματα εξετάζει αυτές για τον εντοπισμό αλλαγών σε κάθε μετάβαση σύμφωνα με τη μεθοδολογία της παραγράφου 5.2 και φυσικά του τελικού παράγοντα *HVF* για κάθε ξεχωριστή περίπτωση. Λαμβάνεται τελικά μια λίστα από ξεχωριστές περιπτώσεις State Checking και τις αντίστοιχες προτάσεις αναδόμησης τους ταξινομημένη ως προς την υψηλότερη προτεραιότητα ανάγκης αναδόμησης με κριτήριο πάντα τα ιστορικά δεδομένα (υψηλότερη τιμή του *HVF*).

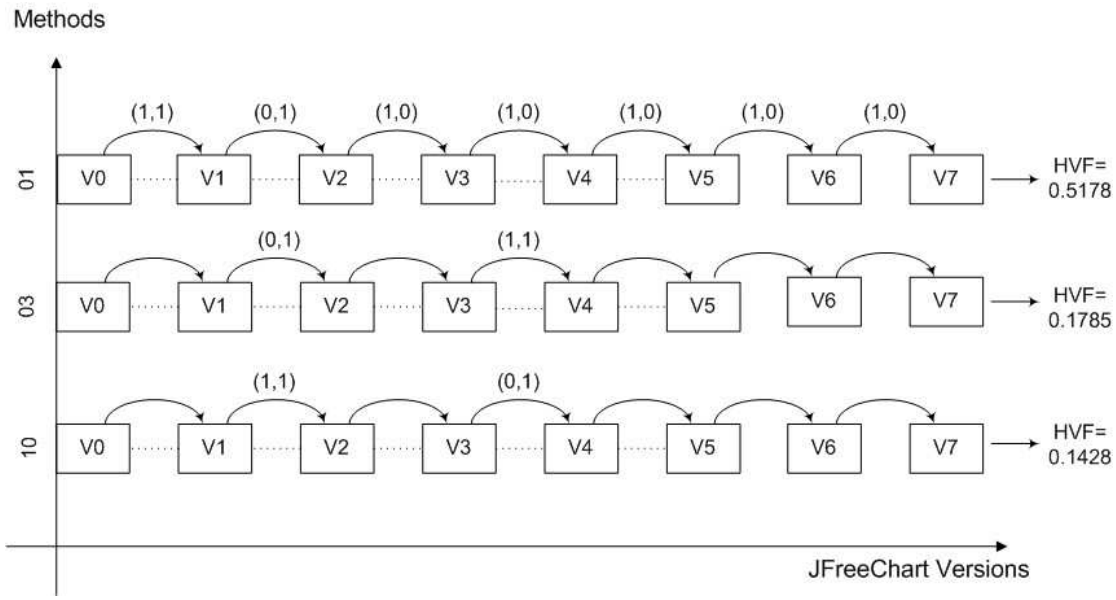
Χρησιμοποιήθηκαν και πάλι οι 8 αρχικές εκδόσεις (0.5.6-0.8.0) του JFreeChart [10] όπως και στην περίπτωση του Long Method. Στην έκδοση JFreeChart 0.8.0 εντοπίστηκαν 21 ξεχωριστές περιπτώσεις εμφάνισης του State Checking και οι 10 με το μεγαλύτερο *HVF* περιέχονται στον πίνακα 2. Η έλλειψη ποικιλομορφίας στα αποτελέσματα, καθώς μόνο τρεις διακριτές τιμές εμφανίζονται, δεν προκαλεί εντύπωση. Από τη στιγμή που το εύρος της αλλαγής μπορεί να λάβει μόνο τρείς τιμές (0,0.5,1) σε αντίθεση με την περίπτωση Long Method που λαμβάνει άπειρες δυνατές μεταξύ στο [0,1], αν αυτό υπολογίζεται ισοδύναμο σε ταυτόσημες μεταβάσεις για διαφορετικές μεθόδους, τότε η τελική τιμή του *HVF* για αυτές θα ταυτίζεται.

Πίνακας 2 : *HVF* που υπολογίστηκε για κάθε μέθοδο με state checking στο JFreeChart 0.8.0

	Method Name	**HVF**
01	isCompatibleHorizontalAxis	0.5178
02	getAdjustedHeight	0.1785
03	getAdjustedWidth	0.1785
04	getBottomSpace	0.1785
05	getTopSpace	0.1785
06	getRightSpace	0.1785

07	getLeftSpace	0.1785
08	layoutLegendItems	0.1428
09	draw	0.1428
10	getPlotType	0.1428

Στο σχήμα 6 απεικονίζεται και τα ενδιάμεσα αποτελέσματα, δηλαδή το εύρος αλλαγής σε κάθε μετάβαση για 3 διαφορετικές μεθόδους του πίνακα 2 που συμβολίζονται με νούμερα ίδια με αυτά του πίνακα. Σε κάθε μετάβαση αντιστοιχεί ένα ζεύγος τιμών εκ των οποίων η πρώτη εκφράζει το *anb* ενώ η δεύτερη το *iso*. Ομοίως με το αντίστοιχο σχήμα που αφορά την κακή οσμή Long Method τα αποτελέσματα που έχουν τιμή 0 για κάποιες μεταβάσεις παραλείπονται (εδώ όταν και οι δυο τιμές του ζεύγους είναι 0).



Σχήμα 6. Εύρος αλλαγής τριών μεθόδων σε κάθε μετάβαση.

6. FEATURE ENVY

6.1. Σύντομη περιγραφή της κακής οσμής

Μια μέθοδος εμφανίζει την κακή οσμή “Feature Envy” σε περίπτωση που είναι «περισσότερο ενδιαφερόμενη» στα χαρακτηριστικά μιας άλλης κλάσης από αυτά της κλάσης στην οποία ανήκει [5]. Αυτό σημαίνει ότι χρησιμοποιεί περισσότερες ιδιότητες και καλεί περισσότερες μεθόδους άλλων κλάσεων παρά της κλάσης που είναι τοποθετημένη. Το γεγονός αυτό επηρεάζει αρνητικά τη σύζευξη (coupling) και τη συνεκτικότητα (cohesion) του συστήματος και για το λόγο αυτό συνιστά ένα πρόβλημα σχεδίασης.

Ενδεικτικό παράδειγμα μεθόδου που εμφανίζει Feature Envy παρουσιάζεται στο σχήμα 7. Η μέθοδος *resetManeuverPanel* μέσω της αναφοράς *maneuverPanel* καλεί έξι μεθόδους από την κλάση *ManeuverPanel* και μέσω της αναφοράς map μια μέθοδο της *MapPanel*. Επίσης χρησιμοποιεί και τη στατική ιδιότητα *MOVE_TO_REINFORCE_STATE* της κλάσης *ManeuverPanel*.

Η εξάλειψη της κακής οσμής επιτυγχάνεται είτε μετακινώντας τη μέθοδο (Move Method Refactoring) στην κλάση που έπρεπε πραγματικά να ανήκει βάση της πληθώρας των χαρακτηριστικών που χρησιμοποιεί από αυτή, είτε μετακινώντας κάποια ιδιότητα (Move Field Refactoring) από την προαναφερθείσα κλάση. Το ημιαυτόματο εργαλείο JDeodorant εντοπίζει τις περιπτώσεις της κακής οσμής και προτείνει την εφαρμογή αναδόμησης με βάση την πρώτη τεχνική (Move Method Refactoring) [17]. Ο εντοπισμός γίνεται χρησιμοποιώντας την έννοια της «απόστασης» κλάσης - μεθόδου ή κλάσης - ιδιότητας. Σε περίπτωση που υπολογίζεται απόσταση της μεθόδου από την κλάση που «ζηλεύει» μικρότερη της απόστασης από την κλάση που ανήκει, τότε η *επισήμανση* του προβλήματος είναι γεγονός.

```
public class MainGame implements Serializable{

private ManeuverPanel maneuverPanel;

private MapPanel map;

private void resetManeuverPanel(){
JTextField maneuverToTF = maneuverPanel.getManeuverToTF();
TextField[] maneuverFromTF = maneuverPanel.getManeuverFromTF();
JTextField[] maneuverFromForcesTF =
maneuverPanel.getManeuverFromForcesTF();
JSlider[] forcesSL = maneuverPanel.getForcesSL();
JRadioButton[] maneuverFromRB = maneuverPanel.getManeuverFromRB();

            maneuverToTF.setText("");
            for (int i =0; i < maneuverFromTF.length ; i++){
                  maneuverFromTF[i].setText("");
                  maneuverFromTF[i].setVisible(false);
                  maneuverFromRB[i].setVisible(false);
                  maneuverFromForcesTF[i].setVisible(false);
                  forcesSL[i].setVisible(false);

                  maneuverFromRB[i].setEnabled(true);
                  maneuverFromTF[i].setEnabled(false);
                  maneuverFromForcesTF[i].setEnabled(false);
                  forcesSL[i].setEnabled(false);
            }

            maneuverFromRB[0].setSelected(true);
            maneuverFromTF[0].setEnabled(true);
            maneuverFromForcesTF[0].setEnabled(true);
            forcesSL[0].setEnabled(true);
            maneuverPanel.setUndoButtonEnabled(false);


            map.setLockHighLight(true);

maneuverPanel.setButtonsState(ManeuverPanel.MOVE_TO_REINFORCE_STATE);
      }
}
```

Σχήμα 7. Παράδειγμα μεθόδου με Feature Envy.

6.2. Χειρισμός των ιστορικών δεδομένων

Έχει αναφερθεί επανειλημμένα πως η γενικότερη φιλοσοφία της εργασίας και της μεθοδολογίας που προτείνεται είναι στα πλαίσια της διευκόλυνσης της συντήρησης του λογισμικού και της παροχή στον σχεδιαστή – συντηρητή αξιόπιστων προτάσεων σχετικά με τις πιθανές αναδομήσεις του κώδικα. Έτσι και στην περίπτωση του Feature Envy ο καταλυτικός παράγοντας για την κατάταξη των προτεινόμενων αναδομήσεων είναι παρατήρηση και η καταγραφή του κατά ποσό το υπάρχον πρόβλημα έχει επηρεάσει τις εργασίες συντήρησης κατά την εξέλιξη του συστήματος.

Από τη στιγμή που το εργαλείο JDeodorant εντοπίσει μια μέθοδο που εμφανίζει Feature Envy, με κατάλληλο υπολογισμό αποστάσεων, όπως έχει αναλυθεί στην ενότητα 6.1, καθορίζει σε ποια κλάση πρέπει να μεταφερθεί η μέθοδος. Αυτό γίνεται διότι είναι αρκετά συχνό το φαινόμενο μια μέθοδος να χρησιμοποιεί μεθόδους ή ιδιότητες από πολλές διαφορετικές κλάσεις. Η κλάση που καθορίζεται ως η πιο κατάλληλη για τη μετακίνηση της μεθόδου ονομάζεται *targetClass*.

Εφόσον η προτεινόμενη αναδόμηση είναι της μορφής Move Method, δηλαδή η εξάλειψη του προβλήματος συνεπάγεται την μετακίνηση της μεθόδου σε άλλη κλάση, το επίπεδο παρατήρησης και εξέτασης του κώδικα στην περίπτωση αυτή δεν μπορεί να είναι άλλο από αυτό της μεθόδου. Για την επικέντρωση της εξέτασης του κώδικα στα πλαίσια του προβλήματος Feature Envy ο καταλληλότερος τρόπος καταγραφής αλλαγής προέρχεται από την ανίχνευση τροποποιήσεων στο σώμα της μεθόδου που έχουν να κάνουν με τα στοιχεία που χρησιμοποιεί η μέθοδος από άλλες κλάσεις. Πιο συγκεκριμένα, ορίζεται και καταγράφεται ως «αλλαγή» σε μια μέθοδο που εμφανίζει Feature Envy αν έχει μεταβληθεί το συνολικό πλήθος μεθόδων και ιδιοτήτων που χρησιμοποιεί η μέθοδος από την *targetClass* κατά τη μετάβαση από μια έκδοση *i-1* στην αμέσως επόμενη *i*.

Ο παραπάνω ορισμός βασίζεται στην παραδοχή ότι αν ο βαθμός χρήσης των στοιχείων άλλης κλάσης παραμένει σταθερός τότε η ανάγκη μετακίνησης της μεθόδου δεν φαντάζει επιτακτική. Αντίθετα αν ο βαθμός χρήσης τους παρουσιάζει σταδιακή αύξηση στις πιο πρόσφατες εκδόσεις του λογισμικού σε σχέση με την αρχική, τότε στην αναδόμηση που επιτάσσει μετακίνηση της μεθόδου πρέπει να προσδοθεί υψηλή προτεραιότητα. Με άλλα λόγια, αν μια μέθοδος στην αρχική έκδοση του λογισμικού χρησιμοποιεί *κ* στοιχεία (ιδιότητες και μεθόδους) από την *targetClass*, τότε η μεταβολή (αύξηση) του αριθμού αυτού στις εκδόσεις που έπονται της αρχικής θα καθορίζει την προτεραιότητα εφαρμογής της αναδόμησης.

Ο υπολογισμός του «ποσοστού» αλλαγής για κάθε μετάβαση στην περίπτωση του Feature Envy βασίζεται στην ποσοστιαία μεταβολή του διακριτού αριθμού ιδιοτήτων και μεθόδων που χρησιμοποιεί η μέθοδος από την *tagetClass* από την έκδοση *i-1* στην έκδοση *i*. Ο συνολικός αριθμός αυτών των ιδιοτήτων και μεθόδων της *targetClass* συμβολίζεται με *nee* (number of envied elements). Έτσι η μαθηματική σχέση που εκφράζει το ποσοστό αλλαγής είναι:

$$pc_i = \frac{nee_i - nee_{i-1}}{nee_i} \quad (11)$$

Η φιλοσοφία που διέπει τον ορισμό του «ποσοστού» αλλαγής ο οποίος περιγράφεται στην παραπάνω παράγραφο και από τη σχέση (11) είναι να έχει νόημα μόνο όταν έχει εντοπισθεί αύξηση του βαθμού χρήσης του συνολικού αριθμού μεθόδων και ιδιοτήτων της *targetClass* από μια έκδοση λογισμικού στην αμέσως επόμενη της. Αντίθετα, είναι ανούσιο να χρησιμοποιηθεί όταν εντοπίζεται αντίστοιχη μείωση συνυπολογίζοντας το γεγονός ότι έτσι το πρόβλημα σχεδίασης Feature Envy αμβλύνεται παρά οξύνεται. Αν και η τελευταία περίπτωση είναι εξαιρετικά σπάνια σε πραγματικά συστήματα, η μεθοδολογία δεν μπορεί να την αγνοήσει για να θεωρηθεί πλήρης και ταυτόχρονα να προσδώσει στο εργαλείο τη δυνατότητα εξαγωγής αξιόπιστων αποτελεσμάτων ακόμα και σε ακραίες περιπτώσεις. Έτσι όταν υπάρχει μείωση του βαθμού χρήσης, δηλαδή $nee_i < nee_{i-1}$ ή στην περίπτωση που ο συνολικός αριθμός ιδιοτήτων και μεθόδων της *targetClass* που χρησιμοποιεί η μέθοδος είναι μηδενικός, δηλαδή $nee_i = 0$ τότε το ποσοστό αλλαγής θεωρείται 0.

Το τελευταίο στάδιο υπολογισμών περιλαμβάνει τον υπολογισμό του *HVF* για κάθε ξεχωριστή περίπτωση Feature Envy που εντοπίζεται στον κώδικα του λογισμικού. Ο υπολογισμός αυτός γίνεται απλά με τη χρήση της σχέσης (2) εφόσον το εύρος των αλλαγών *ec* αντικατασταθεί για την περίπτωση αυτή με το ποσοστό αλλαγής pc που έχει υπολογιστεί από τη σχέση (11).

6.3. Αποτελέσματα

Η εξαγωγή αποτελεσμάτων με την εφαρμογή της προτεινόμενης μεθοδολογίας για την περίπτωση της κακής οσμής Feature Envy βασίστηκες στην εφαρμογή του ημιαυτόματου εργαλείου στο λογισμικό JMol [18]. Το JMol είναι λογισμικό ανοικτού κώδικα υλοποιημένο σε Java για την τρισδιάστατη απεικόνιση χημικών δομών. Χρησιμοποιήθηκαν 6 εκδόσεις του λογισμικού και συγκεκριμένα οι jmol-10.00, jmol-10.2.0, jmol-11.0, jmol-11.0.1, jmol-11.0.2, jmol-11.0.3. Οι ξεχωριστές περιπτώσεις της κακής οσμής εντοπίσθηκαν από το εργαλείο στην σχετικά πιο πρόσφατη έκδοση

δηλαδή την jmol-11.0.3 ενώ οι υπόλοιπες χρησιμοποιήθηκαν ως πηγές για εξέταση των ιστορικών δεδομένων.

Η ανάλυση περιορίστηκε στο πακέτο *org.jmol.viewer* λόγω του ικανοποιητικού αριθμού διακριτών περιπτώσεων της κακής οσμής που εμφανίζει. Συνολικά εντοπίστηκαν 255 διαφορετικές μέθοδοι που υποφέρουν από Feature Envy και για κάθε μια από αυτές το εργαλείο υπολόγισε αυτόματα τον παράγοντα ιστορικής μεταβολής *HVF* που αποτελεί τον κρίσιμο παράγοντα για την ταξινόμηση των προτεινόμενων αναδομήσεων. 169 ήταν οι μέθοδοι που παρουσίασαν αλλαγή κατά την εξέλιξη των εξεταζόμενων εκδόσεων σύμφωνα με τον ορισμό της αλλαγής που περιλαμβάνεται στην προηγούμενη ενότητα. Οι 10 με το μεγαλύτερο συντελεστή *HVF* περιέχονται στον πίνακα 3.

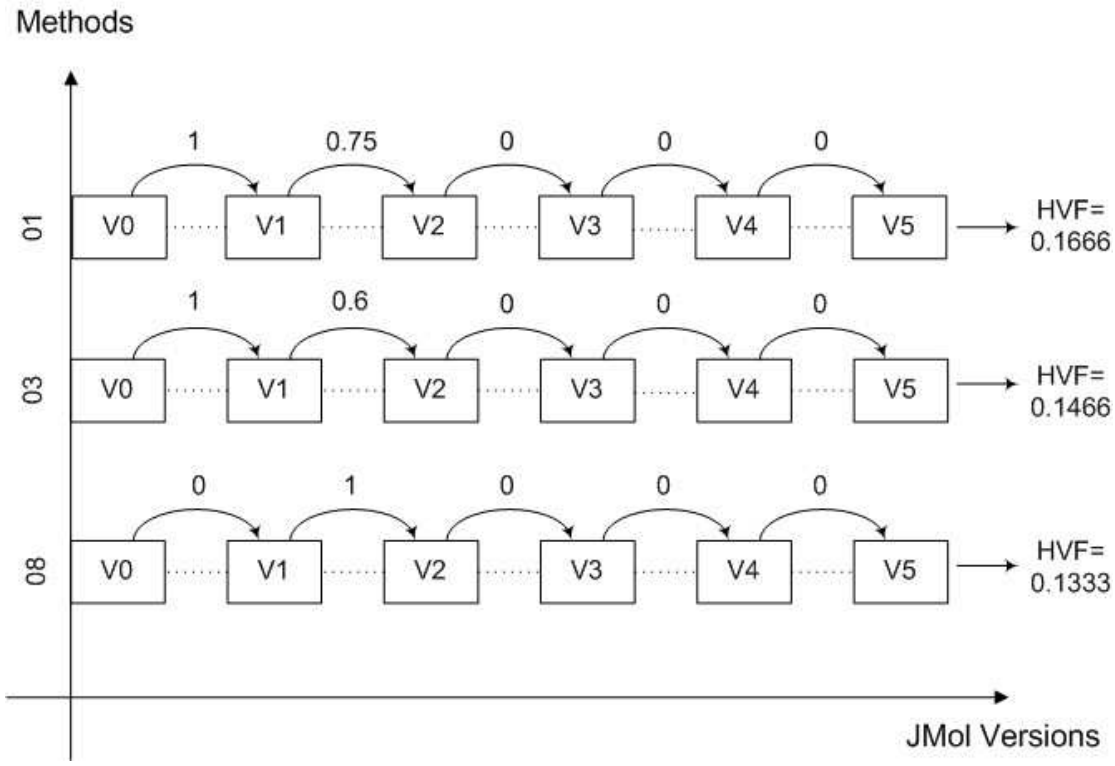
Πίνακας 3 : *HVF* που υπολογίστηκε για κάθε μέθοδο με Feature Envy στο jmol 11.0.3

	Method Name	**HVF**
01	renderPoints	0.1666
02	getMaxMappedValue	0.1466
03	getMinMappedValue	0.1466
04	renderLabel	0.1333
05	jvxlGetDefinitionLine	0.1333
06	floatValue()	0.1333
07	render1	0.1333
08	getDrawCommand	0.1333
09	renderHandles	0.1333
10	renderHandles	0.1333

Παρατηρείται από τον πίνακα 3 αλλά και από το σύνολο των αποτελεσμάτων ότι η ποικιλομορφία στα αποτελέσματα απουσιάζει και πολλές μέθοδοι εμφανίζουν την ίδια τιμή *HVF*. Το γεγονός αυτό δεν είναι περίεργο καθώς αν στην ιδία μετάβαση ο αριθμός των στοιχείων που χρησιμοποιεί η κλάση αυξηθεί από 0 σε k όπου $k \in N$ τότε το «ποσοστό αλλαγής» είναι 1 (100%) και πολλαπλασιασμένο με τον ίδιο συντελεστή ιστορικής βαρύτητας αποδίδει στο *HVF* την ίδια ακριβώς τιμή, εφόσον δεν έχει ύπαρξη αλλαγή σε άλλη μετάβαση. Για το λογισμικό JMol το φαινόμενο αυτό είναι εξαιρετικά συχνό. Η ταξινόμηση των μεθόδων που εμφανίζουν το ίδιο

HVF έγινε με βάση το μέγεθος EntityPlacement [17]. Όσο μικρότερη η τιμή του, τόσο υψηλότερα στην τελική λίστα τοποθετείται η μέθοδος . Αυτό είναι το κριτήριο ταξινόμησης που χρησιμοποιούσε εξ αρχής το JDeodorant χωρίς τη χρήση ιστορικών δεδομένων.

Σε αναλογία με τις άλλες δυο κακές οσμές, έχουν καταγραφεί και τα ενδιάμεσα αποτελέσματα δηλαδή για την κάθε μέθοδο τα ποσοστά αλλαγής που αντιστοιχούν σε κάθε μετάβαση. Επελέγησαν ενδεικτικά τρείς μέθοδοι που θεωρήθηκαν ως κατάλληλες και αντιπροσωπευτικές για την επίδειξη ενδιάμεσων αποτελεσμάτων όπως φαίνονται στο σχήμα 8. Ο οριζόντιος άξονας περιέχει τις 6 εκδόσεις του λογισμικού JMol που εξετάστηκαν ενώ ο κατακόρυφος τρεις διακριτές μεθόδους του πίνακα 3 που συμβολίζονται με τα νούμερα που τους αντιστοιχούν από τον ίδιο πίνακα.



Σχήμα 8. Ποσοστά αλλαγής τριών μεθόδων σε κάθε μετάβαση.

7. ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΥΛΟΠΟΙΗΣΗ

7.1 Εισαγωγικά

Η μεθοδολογία για τον υπολογισμό του *HVF* έτσι όπως περιγράφεται στα αντίστοιχα κεφάλαια έχει υλοποιηθεί ως plug-in στο προγραμματιστικό περιβάλλον του Eclipse. Για την ακρίβεια, έχει ήδη επισημανθεί ότι πρόκειται για μια επέκταση των δυνατοτήτων του JDeodorant που εντοπίζει κακές οσμές στον κώδικα ενός συστήματος και επιπλέον προτείνει τρόπο αναδόμησης τού για την εξάλειψη του προβλήματος. Στις παρακάτω ενότητες περιγράφεται ο τρόπος κατά τον οποίο έχει υλοποιηθεί προγραμματιστικά αυτή η επέκταση. Γίνεται μια προσπάθεια παράθεσης της αλληλουχίας των βημάτων που ακολουθήθηκε καθώς και της βοήθειας που λήφθηκε από τη χρήση ήδη υλοποιημένων μεθόδων του JDeodorant. Αποφεύχθηκε η περιγραφή σε ψευδοκώδικα διότι κρίθηκε ότι θα ήταν αρκετά δυσνόητος για τον αναγνώστη.

Η φύση και η λειτουργικότητα του κώδικα που προστέθηκε για τον υπολογισμό του *HVF* είναι κατεξοχήν αλγοριθμική και έτσι απουσιάζουν οι αρχές και τα πρότυπα σχεδίασης. Παρότι είναι επεκτάσιμος και επιδέχεται τροποποιήσεις για μελλοντική αναπροσαρμογή ή εμπλουτισμό της μεθοδολογίας, ο έλεγχος της συμβατότητας με αρχές και πρότυπα σχεδίασης δεν είναι εφικτός.

Να σημειωθεί τέλος, ότι ο χρήστης του εργαλείου οφείλει να εισάγει στο workspace του Eclipse όσες εκδόσεις ενός συγκεκριμένου λογισμικού επιθυμεί να συμπεριλάβει για άντληση ιστορικών δεδομένων που θα χρησιμοποιηθούν στους υπολογισμούς. Πρέπει να φροντίσει για την κατάλληλη ονομασία των projects έτσι ώστε να είναι εισαγμένα στο workspace με ιστορική σειρά δηλαδή από το αρχαιότερο προς το πιο πρόσφατο.

Η ενότητα 7.2 περιγράφει τα αρχικά γενικά βήματα της τεχνικής που ακολουθήθηκε και είναι κοινά και στις τρεις περιπτώσεις κακών οσμών, ενώ οι ενότητες 7.3, 7.4, 7.5 αναλύουν τη δομή του κώδικα για κάθε μια από τις τρεις περιπτώσεις.

7.2 Κοινή τεχνική στις τρεις περιπτώσεις

Η λειτουργία του εργαλείου Jdeodorant προϋποθέτει την «ανάγνωση» του κώδικα ενός υπό εξέταση project. Η πλήρης υλοποίηση του έχει γίνει πρακτικά με τη βοήθεια των δυνατοτήτων που παρέχει το Eclipse και πιο συγκεκριμένα του *AST* (Abstract Syntax Tree) [19] . Το *AST* στην ουσία αποτελεί τον τρόπο με τον οποίο το Eclipse «διαβάζει» τον κώδικα κάποιου project δηλαδή με την αναπαράσταση κάθε Java αρχείου με τη μορφή ενός *ASTNode*. Η αναπαράσταση αυτού του είδους συνεχίζεται και για χαμηλότερα επίπεδα κώδικα π.χ. για κλάση, μέθοδο κτλ.

Ο εντοπισμός των αλλαγών στις προηγούμενες εκδόσεις ενός project όπως αυτές ορίζονται στις αντίστοιχες θεωρητικές ενότητες απαιτεί την εξέταση του κώδικα τους και επακόλουθα την επενέργεια του *ASTReader* σε αυτές. Ασφαλώς κάτι τέτοιο γίνεται αυτόματα, δηλαδή με μια χρήση του εργαλείου πάνω στην τελευταία έκδοση του λογισμικού και όχι με διαδοχικές χρήσεις πάνω σε κάθε μια από τις διαφορετικές εκδόσεις. Συνέπεια του γεγονότος αυτού είναι και στις τρείς διαφορετικές περιπτώσεις των κακών οσμών να επιβάλλεται η εκτέλεση των εξής αρχικών βημάτων της προγραμματιστικής διαδικασίας:

1. Κατασκευή μιας νέας κλάσης που περιέχει όλο τον κώδικα που υπολογίζει το HVF για κάθε μέθοδο που εντοπίζεται ότι εμφανίζει τη συγκεκριμένη κακή οσμή.
2. Στη βασική μέθοδο της παραπάνω κλάσης γίνεται η «σάρωση» των projects που ήδη έχουν εισαχθεί στο workspace. Κάθε ένα από αυτά μπορεί να χειριστεί με τη μορφή *ASTNode* αφού δημιουργηθεί το σύστημα (*SystemObject*). Τα παραπάνω υλοποιούνται μέσω των εντολών:

```
root = ResourcesPlugin.getWorkspace().getRoot();
IProject[] projects = root.getProjects();

for(IProject project : projects){

IJavaProject javaProject = JavaCore.create(project);

astReader = new ASTReader(javaProject, null);
final SystemObject systemObject = ASTReader.getSystemObject();

      …

}
```

Από τη δημιουργία του *systemObject* και μετά, ακολουθεί κώδικας διαφοροποιείται ανάλογα με τις απαιτήσεις και τις ιδιαιτερότητες της κάθε κακής οσμής και αναλύεται στις παρακάτω ενότητες.

7.3 Long Method

Τα βήματα της υλοποίησης του κώδικα που υπολογίζει το *HVF* για κάθε μέθοδο που εντοπίζεται αρχικά από το JDeodorant ότι εμφανίζει Long Method είναι :

1) Βήμα 1- ενότητα 7.2 : Κατασκευή μιας νέας κλάσης, συγκεκριμένα *LongMethodChanges* στο πακέτο *gr.uom.java.jdeodorant.refactoring.manipulators*.
2) Κατασκευή ενός αντικειμένου *LongMethodChanges* στην κλάση *LongMethod* που δέχεται ως παράμετρο στον κατασκευαστή τη δομή δεδομένων (πίνακα) που περιέχει όλες τις ξεχωριστές περιπτώσεις slice (τμήματα μεθόδων) που έχουν ανιχνευθεί από το εργαλείο ότι πρέπει να εξαχθούν από κάποια μέθοδο για την εξάλειψη του προβλήματος. Με τη βοήθεια των μεθόδων του JDeodorant , λαμβάνονται δομές δεδομένων που αποθηκεύουν επιθυμητά στοιχεία για κάθε εντοπισμένη μέθοδο (π.χ. όνομα, κλάση, αντικείμενο μεθόδου - MethodObject).
3) Βήμα 2- ενότητα 7.2
4) Ανίχνευση για κάθε εντοπισμένη μέθοδο της όμοιας της σε κάθε project και αποθήκευση όλων αυτών σε μια δυσδιάστατη δομή (πίνακας) που ως μια διάσταση έχει τον αριθμό των projects και ως δεύτερη το πλήθος των διακριτών μεθόδων που έχουν εντοπισθεί στο τρέχον project, έστω *allMethodObjects[][]*. Ο εντοπισμός γίνεται με αναζήτηση στο κάθε project μεθόδου με ίδιο όνομα, ίδια κλάση και ίδιες παραμέτρους για κάθε μια που υπάρχει στην αρχική δομή, δηλαδή αυτές που έχουν εντοπισθεί στο τρέχον project.
5) Κατασκευή μιας μεθόδου που υπολογίζει τα ποσοστά αλλαγής σύμφωνα με τον ορισμό που υπάρχει στην ενότητα 4.2. Αυτά αποθηκεύονται επίσης σε μια δισδιάστατη δομή (τύπου double) με τη μια διάσταση να εκφράζει τις μεταβάσεις (*πλήθος projects -1*) και τη δεύτερη το πλήθος των διακριτών μεθόδων όπως στην *allMethodObjects.*

Ο υπολογισμός του ποσοστού αλλαγής μιας μεθόδου σε κάθε μετάβαση γίνεται με τη σάρωση της δομής *allMethodObjects* κατά τη διάσταση του αριθμού των projects και ανάλυση του σώματος της κάθε μεθόδου σε statements όπως έχει εξηγηθεί και στην ενότητα 4.2. Αυτό γίνεται με την κατασκευή και προσθήκη μιας νέας μεθόδου στην κλάση *MethodBodyObject,*την *getAllStatements(),* που αναλύει την κάθε μέθοδο σε «τελικά» statements δηλ αναλύοντας ακόμα και τους βρόγχους for, while, do-while και τα block ελέγχου switch και if-elseif. Η υλοποίηση της *getAllStatements()* είναι πλήρως υλοποιημένη στο πρότυπο σχεδίασης Composite Pattern.

6) Κατασκευή μιας μεθόδου που υπολογίζει τους συντελεστές ιστορικής βαρύτητας λαμβάνοντας ως παράμετρο το πλήθος των μεταβάσεων.
7) Υπολογισμός για κάθε μέθοδο του συντελεστή *HVF* με πολλαπλασιασμό των ποσοστών με τα αντίστοιχα βάρη και συσχέτιση του για κάθε μέθοδο στο αντίστοιχο αντικείμενο της *MethodObject* μέσω μιας συνάρτησης *set()*. Έχει προηγηθεί η προσθήκη της ιδιότητας *HVF* στην κλάση MethodObject. Τέλος στην κλάση LongMethod, για κάθε μέθοδο που έχει εντοπισθεί εξάγεται ως αποτέλεσμα στην οθόνη ο *HVF* που τη συνοδεύει μέσω μιας συνάρτησης *get()*.

7.4 State Checking

Παρατήρηση: Η ονοματολογία που έχει ακολουθηθεί κατά την υλοποίηση του JDeodorant αναφέρει το State Checking ως Type Checking. Για το λόγο αυτό, ο επιπλέον κώδικας που έχει προστεθεί για την αξιοποίηση των ιστορικών δεδομένων έχει ακολουθήσει τον ίδιο όρο και ανάλογα έχουν ονοματιστεί οι σχετικές δομές, μέθοδοι κτλ.

Τα βήματα της υλοποίησης του κώδικα που υπολογίζει το HVF για κάθε μέθοδο που εντοπίζεται αρχικά από το JDeodorant ότι εμφανίζει State Checking είναι :

1) Βήμα 1- ενότητα 7.2 : Κατασκευή μιας νέας κλάσης, συγκεκριμένα *TypeCheckingChanges* στο πακέτο *gr.uom.java.jdeodorant.refactoring.manipulators*.

2) Κατασκευή ενός αντικειμένου *TypeCheckingChanges* στην κλάση *TypeChecking* που δέχεται ως παράμετρο στον κατασκευαστή τη δομή δεδομένων (πίνακα) που περιέχει όλες τις ξεχωριστές περιπτώσεις block κώδικα που συνιστούν περίπτωση type checking και έχουν ανιχνευθεί από το εργαλείο στην πιο πρόσφατη έκδοση του λογισμικού. Αυτές αποθηκεύονται σε μεταβλητές τύπου *TypeCheckElimation.* Με τη βοήθεια των μεθόδων του JDeodorant , λαμβάνονται δομές δεδομένων που αποθηκεύουν επιθυμητά στοιχεία για κάθε εντοπισμένη περίπτωση (π.χ. όνομα μεθόδου που περιέχεται, κλάση που ανήκει).
3) Βήμα 2- ενότητα 7.2
4) Από τη στιγμή που έχει δημιουργηθεί το *SystemObject* είναι δυνατόν να ληφθούν όλες οι διακριτές περιπτώσεις Type Checking του project και να αποθηκευτούν σε μια μεταβλητή τύπου *TypeCheckEliminationResults*. Αυτό γίνεται με την βοήθεια της μεθόδου *generateTypeCheckEliminations()* και πιο συγκεκριμένα με τις παρακάτω γραμμές κώδικα:

```
typeCheckEliminationResults =
systemObject.generateTypeCheckEliminations(systemObject.getClassObjects
( ),null);
List<TypeCheckElimination> allEliminations =
typeCheckEliminationResults.getTypeCheckEliminations();
```

Σαρώνοντας την παραπάνω λίστα: α) αποθηκεύεται σε μια δυσδιάστατη δομή η όμοια περίπτωση Type Checking με κάθε μια από αυτές που έχουν εντοπισθεί., β) εντοπίζεται το πλήθος των περιπτώσεων Type Checking στο ίδιο project που βασίζονται στον έλεγχο ίδιας μεταβλητής, Αυτή η δομή έχει ως μια διάσταση το πλήθος των project και ως δεύτερη το πλήθος των διακριτών περιπτώσεων Type Checking.

5) Η δομή που συμπληρώνεται από το σκέλος (α) της παραπάνω παραγράφου περνάει ως όρισμα στη μέθοδο που δημιουργείται για τον υπολογισμού του εύρους αλλαγής του πρώτου άξονα σύμφωνα με το ορισμό της ενότητας 5.2. Το πλήθος που υπολογίζεται από το σκέλος (β) (αποθηκεύεται επίσης σε δυσδιάστατη δομή) περνά ως όρισμα στη μέθοδο που δημιουργείται για τον υπολογισμό του εύρους αλλαγής κατά το δεύτερο άξονα σύμφωνα με τον

ορισμό της ενότητας 5.2. Οι δυο διαφορετικές δυαδικές τιμές που προκύπτουν για κάθε ένα από τους δυο άξονες αλλαγής χρησιμοποιούνται για τον υπολογισμό του συνολικού εύρους σύμφωνα και πάλι με τον ορισμό, το οποίο αποθηκεύεται επίσης σε μια δισδιάστατη δομή.

6) Κατασκευή μιας μεθόδου για τον υπολογισμό των συντελεστών ιστορικής βαρύτητας λαμβάνοντας ως παράμετρο το πλήθος των μεταβάσεων (*πλήθος projects -1*).
7) Υπολογισμός για κάθε μέθοδο του *HVF* με πολλαπλασιασμό του εύρους της αλλαγής για κάθε μετάβαση και του αντίστοιχου συντελεστή ιστορικής βαρύτητας και συσχέτιση με την αντίστοιχη περίπτωση *typeCheckElimination* μέσω μεθόδου *set()*. Το τελικό αποτέλεσμα λαμβάνεται από την κλάση *TypeChecking* κατά τη διαδικασία εξαγωγής αποτελεσμάτων με σάρωση του πίνακα που περιέχει όλα τα *typeCheckElimination* μέσω συνάρτησης *get()*.

7.5 Feature Envy

Τα πρώτα 4 βήματα της προγραμματιστικής διαδικασίας και υλοποίησης του κώδικα που είναι υπεύθυνος για τον υπολογισμό του *HVF* στην περίπτωση της κακής οσμής Feature Envy είναι πανομοιότυπα με αυτά της κακής οσμής Long Method επομένως δεν υπάρχει λόγος να επαναληφθούν. Η μόνη διαφορά είναι ότι προφανώς η νέα κλάση που κατασκευάζεται ονομάζεται *FeatureEnvyChanges* και δημιουργείται ένα αντικείμενο της στην κλάση *FeatureEnvy* το οποίο δέχεται ως παράμετρο στον κατασκευαστή τον πίνακα τύπου *CandidateRefactoring* που περιέχει τις μεθόδους υποψήφιες για αναδόμηση. Τα υπόλοιπα βήματα είναι τα εξής:

5) Κατασκευή μιας μεθόδου που υπολογίζει τα ποσοστά αλλαγής σύμφωνα με τον ορισμό που υπάρχει στην ενότητα 6.2. Αυτά αποθηκεύονται σε μια δισδιάστατη δομή της οποίας η μια διάσταση εκφράζει τον αριθμό των μεταβάσεων (*πλήθος projects -1*) ενώ η δεύτερη το πλήθος των διακριτών μεθόδων που έχουν εντοπιστεί ότι πάσχουν από Feature Envy.

   Για τον υπολογισμό του ποσοστού αλλαγής σαρώνεται το σύνολο των μεθόδων και λαμβάνονται για κάθε μια από αυτές το πλήθος των μεθόδων που καλούν από άλλες κλάσεις μέσω της μεθόδου *getMethodInvocations()* καθώς

και των ιδιοτήτων από άλλες κλάσεις μέσω της μεθόδου *getFieldInstructions()*. Για κάθε κλάση μεθόδου από άλλη κλάση και για κάθε χρήση ιδιότητας ελέγχεται αν η κλάση που ανήκει αυτή ταυτίζεται με την *targetClass* (βλ ενότητα 6.2).Το συνολικό πλήθος αυτών που ανήκουν πράγματι στην *targetClass* κρατείται και αποθηκεύεται σε μια δυσδιάστατη ιδίων διαστάσεων με την *allMethodObjects* (βλ. ενότητα 7.3 – Long Method). Το ποσοστό αλλαγής υπολογίζεται ως το ποσοστό αύξησης των «στοιχείων» της *targetClass* που χρησιμοποιεί η μέθοδος σε κάθε έκδοση του λογισμικού, σε πλήρη ακολουθία με τον ορισμό της παραγράφου 6.2.

6) Κατασκευή μιας μεθόδου για τον υπολογισμό των συντελεστών ιστορικής βαρύτητας λαμβάνοντας ως παράμετρο το πλήθος των μεταβάσεων (*πλήθος projects -1*).
7) Υπολογισμός για κάθε μέθοδο του *HVF* με πολλαπλασιασμό του ποσοστού της αλλαγής για κάθε μετάβαση και του αντίστοιχου συντελεστή ιστορικής βαρύτητας και συσχέτιση με την αντίστοιχη περίπτωση *MoveMethodCandidateRefactoring* (υποκλάση της *CandidateRefactoring*) μέσω μεθόδου *set()*. Το τελικό αποτέλεσμα λαμβάνεται από την κλάση *Feature Envy* κατά τη διαδικασία εξαγωγής αποτελεσμάτων με σάρωση του πίνακα που περιέχει όλα τα *CandidateRefactoring* μέσω συνάρτησης *get()*.

8. ΑΞΙΟΛΟΓΗΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

8.1. Γενικά

Ο σκοπός αυτής της εργασίας είναι η θεμελίωση μιας μεθοδολογίας για την ταξινόμηση των προτεινόμενων αναδομήσεων που προτείνονται από το εργαλείο JDeodorant. Από εκεί και πέρα είναι στην ευχέρεια του συντηρητή πόσο πιστά θα εμπιστευτεί τα αποτελέσματα που αυτή του παρέχει. Σε ανάλογες περιπτώσεις όπου μια νέα μεθοδολογία προτείνεται, συνηθίζεται να συνοδεύεται από «αξιολόγηση» των αποτελεσμάτων που προκύπτουν από αυτήν έτσι να γίνει σχηματιστεί μια πρώτη εντύπωση σχετικά με την ορθότητα και την αξιοπιστία των αποτελεσμάτων.

Για να επιτευχθεί αυτό στα πλαίσια αυτής της εργασίας, ανατέθηκε σε ένα ειδικό σε θέματα συντήρησης να σχηματίσει μια δική του σειρά προτεραιότητας των αναδομήσεων που προτείνει το JDeodorant από την εφαρμογή του σε συγκεκριμένα έργα λογισμικού. Για την ταξινόμηση αυτή, ο ειδικός έλαβε εξίσου υπόψη του δυο πράγματα: α) συγκεκριμένα κριτήρια ταξινόμησης των αναδομήσεων που έχουν βαρύνουσα σημασία για κάθε κακή οσμή, β) την τιμή του *HVF* που προκύπτει για κάθε ξεχωριστή περίπτωση κακής οσμής. Επίσης είχε τη δυνατότητα να εξετάσει τον κώδικα που εμπλέκεται σε κάθε περίπτωση για τη συλλογή περισσότερων πληροφοριών και το σχηματισμό σφαιρικότερης άποψης.

Τα κριτήρια ταξινόμησης που έλαβε υπόψη του ο ειδικός δεν έχουν σχέση με ιστορικά δεδομένα, συνεπώς η διαδικασία της αξιολόγησης δεν μπορεί να θεωρηθεί πλήρως ασφαλής, αξιόπιστη και αιτιολογημένη θεωρητικά. Παρόλα τα σημαντικά θέματα αυθαιρεσίας που εισάγει, κρίθηκε ωφέλιμο να γίνει κατά αυτό τον τρόπο ως μια πρώτη προσπάθεια παροχής ενδείξεων ορθότητας η μη της μεθοδολογίας. Εφόσον πρόκειται για κάτι καινοτόμο στα πλαίσια της συντήρησης λογισμικού, είναι αρκετά δύσκολο να κατευθύνει κάποιος ειδικός τη σκέψη του με τρόπο παραπλήσιο της μεθοδολογίας για να γίνει άρτια αξιολόγηση των αποτελεσμάτων.

8.2. Μέθοδοι αξιολόγησης

Το αποτέλεσμα της αξιολόγησης εξάγεται από τη σύγκριση των δυο ταξινομήσεων (αυτής που γίνεται με βάση της τιμής του *HVF* και αυτής που γίνεται με βάση τη γνώμη του ειδικού). Για αυτή τη σύγκριση χρησιμοποιήθηκαν δυο τρόποι:

1. Το μέτρο Spearman's footrule [1][3] το οποίο μπορεί να υπολογιστεί για δυο λίστες ταξινόμησης ίδιου μεγέθους, έστω $|S|$. Σύμφωνα με αυτό, αν με σ_1 και σ_2 συμβολίζονται τα στοιχεία της πρώτης και της δεύτερης λίστας αντίστοιχα και λαμβάνουν τιμές $1..S$ τότε το μέτρο Spearman' s footrule υπολογίζεται από τη σχέση:

$$Fr^{|S|}(\sigma_1, \sigma_2) = \sum_{i=1}^{|S|} |\sigma_1(i) - \sigma_2(i)| \,. \tag{12}$$

Η ελάχιστη τιμή του F_r όπως προκύπτει από τη σχέση (12) είναι 0 και αυτό συμβαίνει όταν οι δυο λίστες είναι πανομοιότυπες. Η μέγιστη τιμή είναι $\frac{1}{2}S^2$ όταν ο S είναι άρτιος και $\frac{1}{2}(|S|+1)(|S|-1)$ όταν ο S είναι περιττός και προκύπτει όταν η σειρά της δεύτερης λίστας είναι αντίστροφη σε σχέση με αυτή της πρώτης. Αν διαιρέσουμε το F_r με τη μέγιστη τιμή του, τότε λαμβάνουμε το κανονικοποιημένο μέτρο Spearman's footrule NF_r. Η μαθηματική σχέση με την οποία υπολογίζεται είναι :

$$NF_r = \frac{{F_r}^{|S|}}{max{F_r}^{|S|}} \,. \tag{14}$$

Η τιμή του κανονικοποιημένου μέτρου κυμαίνεται στο διάστημα [0..1] και έτσι οι τιμές που προκύπτουν από τη χρήση του κάνουν τα αποτελέσματα της αξιολόγησης ιδιαίτερα ευκρινή, μια και δίνουν καλύτερη αίσθηση για το σε ποιο βαθμό οι δυο λίστες είναι όμοιες. Για μεγαλύτερη ευκολία χρησιμοποιήθηκε και για τις τρεις κακές οσμές το συμπληρωματικό ως προς τη μονάδα του κανονικοποιημένου μέτρου που η τιμή του δίνεται από τη σχέση:

$$F = 1 - NF_r \tag{15}$$

Το αποτέλεσμα της τη σύγκρισης των ταξινομήσεων με βάση την τιμή του *F* υποδηλώνει ότι όσο πλησιέστερα είναι η τιμή του *F* στο 1, τόσο περισσότερο η ταξινόμηση που προτείνει ο ειδικός προσεγγίζει αυτήν που προκύπτει από τη μεθοδολογία και επακόλουθα τα αποτελέσματα της δεύτερης μπορούν να θεωρηθούν αξιόπιστα. Αντίστροφα, χαμηλές τιμές για το *F* , ιδιαίτερα κοντά στο 0 μαρτυρούν σημαντική απόκλιση στις δυο ταξινομήσεις.

3. Το μέτρο σχετικής σειράς στοιχείων ζευγών *mpco* (measure of values' pair comparable order) από τις δυο ταξινομήσεις. Η τιμή αυτού του μέτρου προκύπτει από το ποσοστό όλων των ζευγών των τιμών που τα στοιχεία τους βρίσκονται στην ίδια σχετική σειρά και στις δυο λίστες. Αν το μέγεθος αμφότερων των ταξινομήσεων είναι *n* (επιβάλλεται να είναι κοινό και για τις δυο) τότε η τιμή του *mpco* δίνεται από τη σχέση:

$$mpco = \frac{\#pairs\ of values\ with\ the\ same\ comparable\ order}{n(n-1)/2} \quad (16)$$

Ο παρανομαστής του κλάσματος της σχέσης (16) είναι το πλήθος όλων των δυνατών ζευγών σε μια λίστα μεγέθους *n*. Η διαίρεση με αυτόν εξυπηρετεί την κανονικοποίηση και αυτού του παράγοντα ώστε η τιμή του να κυμαίνεται επίσης στο [0..1]. Και πάλι η βέλτιστη τιμή που δείχνει απόλυτη ταύτιση των δυο ταξινομήσεων είναι το 1, ενώ η τιμή 0 δείχνει την πλήρη απόκλιση μεταξύ τους. Ασφαλώς, στα πλαίσια της αξιολόγησης επιθυμητές τιμές είναι αυτές που προσεγγίζουν το 1.

8.3 Αξιολόγηση αποτελεσμάτων για Long Method

Τα αποτελέσματα που χρησιμοποιήθηκαν για αξιολόγηση στην περίπτωση της κακής οσμής Long Method προέρχονται από την εφαρμογή του εργαλείου στο λογισμικό JFreeChart 0.8.0. Στην ουσία είναι τα ίδια με αυτά που παρατίθενται στην ενότητα 4.4. Για τον υπολογισμό του *HVF* χρησιμοποιήθηκαν επίσης οι 7 ίδιες προηγούμενες εκδόσεις του λογισμικού όπως αναφέρονται στη σχετική ενότητα που αφορά την κακή οσμή Long Method.

Στο σημείο αυτό, για την αποφυγή σύγχυσης με τα αποτελέσματα της αξιολόγησης, πρέπει να γίνει η εξής διευκρίνηση : Τα αυθεντικά αποτελέσματα του JDeodorant είναι προτεινόμενες αναδομήσεις της μορφής "extract method" για μεθόδους που εμφανίζουν Long Method. Στη κάθε μέθοδο αντιστοιχούν

περισσότερες από μια αναδομήσεις μια και από κάθε αρχική μέθοδο μπορεί να εξαχθεί διαφορετικό τμήμα της με βάση διαφορετική μεταβλητή [16]. Η ταξινόμηση που έγινε από τον ειδικό, επιτελέστηκε με διαχωρισμό των περιπτώσεις εξαγωγής διαφορετικής μεταβλητής, θεωρώντας τες ξεχωριστές αναδομήσεις, σύμφωνα με τη λειτουργία του JDeodorant, άσχετα αν επειδή προέρχονται από την ίδια αρχική μέθοδο εμφανίζουν την ίδια τιμή *HVF*. Έτσι τα αποτελέσματα παρατίθενται στον πίνακα 4 με βάση διακριτές εξαγόμενες μεταβλητές συνοδευόμενες από τη μέθοδο που προέρχονται και τον αντίστοιχο συντελεστή *HVF*.

Πίνακας 4 : Ταξινόμηση μεθόδων με Long Method στο JFreeChart 0.8.0 με δύο τρόπους.

***HVF* Rank**	**Expert rank**	**Method Name**	**Extracted Variable**	**HVF**
01	12	createStackedVerticalBarChart3D	plot	0.1510
02	01	createVerticalBarChart3D	plot	0.1510
03	02	createVerticalXYBarChart	plot	0.1369
04	03	drawHorizontal	startX	0.1260
05	10	drawHorizontal	rightSpace	0.1260
06	11	drawHorizontal	leftSpace	0.1260
07	04	drawHorizontal	bottomSpace	0.1260
08	09	drawHorizontal	topSpace	0.1260
09	05	drawHorizontal	startY	0.1260
10	06	createHighLowChart	plot	0.0892
11	07	drawHorizontal	titleX	0.0857
12	08	drawHorizontal	titleY	0.0857
13	13	drawHorizontal	rightSpace	0.0857
14	14	drawHorizontal	leftSpace	0.0857

Η μέθοδος *drawHorizontal* που υπάρχει στις θέσεις 04-09 της *HVF* Rank δεν είναι η ίδια με την *drawHorizontal* των θέσεων 11-14 και αυτό διότι πρόκειται για μεθόδους με ίδιο όνομα που ανήκουν σε διαφορετικές κλάσεις. Παρατηρώντας την ταξινόμηση με βάση το *HVF* και αυτή που προτείνεται από τον ειδικό, συμπεραίνουμε πως η ταύτιση δεν είναι ικανοποιητική καθώς υπάρχουν σημαντικές διαφοροποιήσεις. Αυτό άλλωστε φανερώνουν και οι τιμές των μέτρων που έχουν επιλεγεί για τη σύγκριση των δυο ταξινομήσεων. Συγκεκριμένα για το μέτρο Spearman's footrule λαμβάνεται $F = 0.551$, ενώ το μέτρο σχετικής σειράς μεταξύ ζευγών εμφανίζεται ελαφρώς αυξημένο δίνοντας *mpco* = 0.703.

Εντούτοις, οι παραπάνω τιμές καθώς και η μη απόλυτη ταύτιση των δυο ταξινομήσεων δεν κρίνονται ως κάτι περίεργο και μη αναμενόμενο. Έχει άλλωστε αναφερθεί πως τα κριτήρια που λαμβάνει υπόψη του ο ειδικός δεν έχουν να κάνουν με ιστορικά δεδομένα και έτσι η αξιολόγηση αυτή ήταν ούτος η άλλος πειραματική. Στην περίπτωση του Long Method το επιπλέον κριτήριο είναι για κάθε μεταβλητή «ο μέσος όρος του λόγου των διπλά εμφανιζόμενων statements στην αρχική και εξαγόμενη μέθοδο προς τα συνολικά εξαγόμενα statements». Ο μέσος όρος υπολογίζεται επειδή και για κάθε μεταβλητή υπάρχει περισσότεροι του ενός τρόποι για να εξαχθεί block από statements.

8.4 Αξιολόγηση αποτελεσμάτων για State Checking

Τα αποτελέσματα που χρησιμοποιήθηκαν για αξιολόγηση στην περίπτωση της κακής οσμής State Checking είναι εκείνα ακριβώς που παραθέτονται στην ενότητα 5.3 και προκύπτουν από το λογισμικό JFreeChart 0.8.0. Για τον υπολογισμό του *HVF* χρησιμοποιήθηκαν επίσης οι 7 ίδιες προηγούμενες εκδόσεις του λογισμικού όπως αναφέρονται στη σχετική ενότητα που αφορά την κακή οσμή. Λήφθηκαν υπόψη οι 10 πρώτες μέθοδοι με το μεγαλύτερο *HVF* που χρειάζονται αναδόμηση για να απαλλαγούν από την κακή οσμή. Ζητήθηκε από τον ειδικό να ταξινομήσει εκ νέου τις ίδιες μεθόδους συνυπολογίζοντας εκτός του *HVF* και κάποια κριτήρια που ο ίδιος επέλεξε ως κρίσιμα:

α) Για περίπτωση της κατηγορίας αντικατάστασης του State Checking με το πρότυπο σχεδίασης State/Strategy το επιπλέον κριτήριο είναι ο αριθμός που οι ίδιες εντολές ελέγχου (ίδιο State Checking) είναι διάσπαρτες σε όλο το σύστημα.

β) Για περίπτωση της κατηγορίας State Checking RTTI που αντικαθίσταται με πολυμορφισμό το επιπλέον κριτήριο είναι ο αριθμός των υποκλάσεων που ανακτώνται για το εμπλεκόμενο αντικείμενο ή πιο απλοϊκά η έκταση του κώδικα που εμπεριέχει το State Checking

Στον πίνακα 5 παρατίθενται οι μέθοδοι που εμφανίζουν State Checking ταξινομημένες κατά την αύξουσα τιμή του *HVF* καθώς και κατά τη γνώμη του ειδικού.

Με βάση τα δεδομένα του πίνακα 5 παρατηρείται μια πλήρης ταύτιση της εκτίμησης του ειδικού με την ταξινόμηση που προκύπτει από τη μεθοδολογία. Προφανώς, οι τιμές των μέτρων σύγκρισης είναι βέλτιστες δηλαδή για το

κανονικοποιημένο μέτρο Spearman's footrule είναι $F = 1$ και το μέτρο σχετικής θέσης μεταξύ ζευγών είναι *mpco* =1 επίσης.

Πίνακας 5 : Ταξινόμηση μεθόδων με State Checking στο JFreeChart 0.8.0 με δύο τρόπους.

***HVF* Rank**	**Expert rank**	**Method Name**	***HVF***
01	01	isCompatibleHorizontalAxis	0.5178
02	02	isCompatibleHorizontalAxis	0.5178
03	03	isCompatibleHorizontalAxis	0.5178
04	04	isCompatibleHorizontalAxis	0.5178
05	05	getAdjustedHeigh	0.1785
06	06	getAdjustedHeigh	0.1785
07	07	getBottomSpace	0.1785
08	08	getLeftSpace	0.1785
09	09	getRightSpace	0.1785
10	10	getTopSpace	0.1785

8.5. Αξιολόγηση αποτελεσμάτων για Feature Envy

Τα αποτελέσματα που χρησιμοποιήθηκαν για αξιολόγηση στην περίπτωση της κακής οσμής Feature Envy είναι εκείνα ακριβώς που παραθέτονται στην ενότητα 6.3 και προκύπτουν από το πακέτο *org.jmol.viewer* του λογισμικού jmol-11.0.3. Για τον υπολογισμό του *HVF* χρησιμοποιήθηκαν 6 εκδόσεις του λογισμικού και συγκεκριμένα από την έκδοση jmol-10.00 έως την έκδοση jmol-11.0.3.Λήφθηκαν υπόψη οι 10 πρώτες μέθοδοι με το μεγαλύτερο *HVF* που χρειάζονται αναδόμηση για να απαλλαγούν από την κακή οσμή. Ζητήθηκε από τον ειδικό να ταξινομήσει εκ νέου τις ίδιες μεθόδους συνυπολογίζοντας εκτός του *HVF* και κάποια κριτήρια που ό ίδιος θεωρεί σημαντικά. Στην περίπτωση του Feature Envy το επιπλέον κριτήριο ήταν μια μετρική που υπολογίζεται για κάθε προτεινόμενη αναδόμηση και εκφράζει τη βελτίωση (ή τον υποβιβασμό) της ποιότητας που επιφέρει αυτή στο σύστημα και ονομάζεται EntityPlacement [17]. Ο πίνακας 6 περιέχει την ταξινόμηση των αποτελεσμάτων αρχικά με βάση το *HVF* (ομοίως με την παράγραφο 6.3) καθώς και αυτή που προτείνει ο ειδικός.

Πίνακας 6 : Ταξινόμηση μεθόδων με Feature Envy στο jmol 11.0.3 με δυο τρόπους

***HVF* Rank**	**Expert rank**	**Method Name**	***HVF***
01	04	renderPoints	0.1666
02	05	getMaxMappedValue	0.1466
03	06	getMinMappedValue	0.1466
04	07	renderLabel	0.1333
05	08	jvxlGetDefinitionLine	0.1333
06	09	floatValue()	0.1333
07	10	render1	0.1333
08	01	getDrawCommand	0.1333
09	02	renderHandles	0.1333
10	03	renderHandles	0.1333

Η σύγκριση των δυο διαφορετικών ταξινομήσεων αποφέρει ένα συμπέρασμα μετρίως ικανοποιητικό καθώς υπάρχουν σημαντικές αποκλίσεις. Μάλιστα η τιμή του *F* προκύπτει μόλις 0.16 που είναι αρκετά μικρή. Κάπως καλύτερη είναι η τιμή του μέτρου σχετικής θέσης μεταξύ στοιχείων ζευγών που είναι *mpco* = 0.533 και οφείλεται στο ότι αρκετά ζεύγη τιμών από τις δυο ταξινομήσεις έχουν τα στοιχεία τους στην ίδια σχετική σειρά. Ο λόγος του μικρού «ποσοστού» ταύτισης των δυο ταξινομήσεων είναι ο ίδιος με αυτόν που περιγράφεται στην εισαγωγή του κεφαλαίου καθώς και στην περίπτωση της κακής οσμής Long Method, ότι δηλαδή τα κριτήρια του ειδικού δεν σχετίζονται με ιστορικά δεδομένα.

9. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

9.1. Συμπεράσματα

Η μεθοδολογία που αναλύεται σε αυτή την εργασία έχει σκοπό την ταξινόμηση των αναδομήσεων που αφορούν κακές οσμές του κώδικα ενός συστήματος και προκύπτουν από συγκεκριμένο ημιαυτόματο εργαλείο που ονομάζεται JDedorant. Η ταξινόμηση αυτή γίνεται με βάση την εξέταση προηγούμενων εκδόσεων του ίδιου λογισμικού και τον υπολογισμό του «παράγοντα ιστορικής μεταβολής τιμή αυτού του παράγοντα εκφράζει το κατά πόσο έχει αποτελέσει ένα τμήμα κώδικα που εμφανίζει κακή οσμή αντικείμενο συντήρησης στο παρελθόν. Όσο μεγαλύτερος προκύπτει αυτός ο παράγοντας τόσο υψηλότερη είναι η προτεραιότητα που ανατίθεται στη συγκεκριμένη αναδόμηση διότι συντηρείται συχνότερα. Η απαλλαγή του από την κακή οσμή διευκολύνει τη διαδικασία συντήρησης.

Ο παράγοντας ιστορικής μεταβολής υπολογίζεται δίνοντας περισσότερη βαρύτητα σε δεδομένα που ανακύπτουν από την εξέταση των πιο πρόσφατων εκδόσεων του λογισμικού μια και θεωρείται, λόγω της εγγύτητας τους στην τρέχουσα έκδοση, ότι ασκούν μεγαλύτερη επιρροή στην επικείμενη εξέλιξη του. Αυτή είναι μια κοινή παραδοχή που ισχύει και εφαρμόζεται και για τις τρεις κακές οσμές που εξετάζει η μεθοδολογία. Αυτό διαφοροποιείται σε κάθε μια είναι ο ορισμός της αλλαγής του εμπλεκόμενου τμήματος κώδικα κατά τη μετάβαση από μια έκδοση στην επόμενη καθώς και το εύρος αυτής αλλαγής. Είναι πράγματα που εξαρτώνται από τη φύση της κακής οσμής και προσδιορίζονται διαφορετικά για κάθε μια από αυτές. Επίσης διαφορετικά προσδιορίζεται και η «σφαίρα επιρροής» της κάθε κακής οσμής δηλαδή το τμήμα κώδικα στο οποίο εξετάζεται η ύπαρξη ή όχι αλλαγών σε πλαίσια συντήρησης.

Η μεθοδολογία έχει υλοποιηθεί και προγραμματιστικά, προστέθηκε στις δυνατότητες της λειτουργίας του JDeodorant το οποίο ενσωματώνεται στο Eclipse ως plug-in. Χρησιμοποιήθηκε επάνω σε συστήματα ανοικτού κώδικα για την εξαγωγή αποτελεσμάτων τα οποία κρίνονται αναμενόμενα. Το μόνο αρνητικό σημείο των αποτελεσμάτων είναι η έλλειψη ποικιλομορφίας για αυτά που αφορούν την κακή

οσμή State Checking. Αυτό οφείλεται εν πολλοίς στον ορισμό της αλλαγής που στα πλαίσια της συγκεκριμένης κακής οσμής καθώς και στο ότι σε κάποια συστήματα λογισμικού τα τμήματα κώδικα που επιτελούν ελέγχους σταθερών για την επιλογή γραμμής εκτέλεσης δεν υπάρχει λόγος να αλλάξουν κατά την εξέλιξη του συστήματος.

Στα αποτελέσματα που ελήφθησαν έγινε μια πειραματικής μορφής αξιολόγηση από κάποιο ανεξάρτητο ειδικό ο οποίος επιφορτίστηκε με την κατασκευή νέας ταξινόμησης για τις προτεινόμενες αναδομήσεις λαμβάνοντας υπόψη τόσο το συντελεστή ιστορικής μεταβολής, όσο και κριτήρια που ο ίδιος θεώρησε σημαντικά σχετικά με το είδος της κακής οσμής και της αναδόμησης. Η διαδικασία της αξιολόγησης ήταν η σύγκριση των δυο ταξινομήσεων για κάθε κακή οσμή με τη χρήση κατάλληλων μέτρων. Για την περίπτωση του Feature Envy ο βαθμός ταύτισης των δυο ταξινομήσεων δεν ήταν ικανοποιητικός. Αυτό δε συνεπάγεται απαραίτητα τη χαμηλή αξιοπιστία της μεθοδολογίας καθώς τα κριτήρια με τα οποία εξήγαγε την ταξινόμηση ο ειδικός δε συσχετίζονται με ιστορικά δεδομένα.

9.2. Μελλοντική εργασία

Μελλοντικός στόχος στα πλαίσια έρευνας του ζητήματος της ταξινόμησης αναδομήσεων με βάση ιστορικά δεδομένα είναι η βελτίωση και η εξέλιξη των ορισμών της αλλαγής του τμήματος κώδικα που εμπλέκεται σε κάθε κακή οσμή. Σε αυτό θα βοηθούσε ο συνυπολογισμός περισσότερων παραμέτρων και στοιχείων για τον κάθε ορισμό. Ωφέλιμο θα ήταν να βρεθεί και μια τεχνική συνδυασμού των ιστορικών δεδομένων με ποσότητες που ήδη χρησιμοποιούνται από το JDeodorant έτσι ώστε το ημιαυτόματο εργαλείο να παρέχει μια «συνδυασμένη» ταξινόμηση που προκύπτει από σφαιρικότερη επισκόπηση του λογισμικού και των προηγούμενων εκδόσεων του. Τέλος αρκετό ενδιαφέρον θα είχε και μια προσπάθεια μετατροπής της φιλοσοφίας της μεθοδολογίας σε «πιθανοκρατικού» χαρακτήρα με την εισαγωγή σε αυτήν ασαφούς λογικής (fuzzy logic).

Α ν α φ ο ρ έ ς:

[1] J. Bar-IIan, M. Mat-Hassan and M. Levene, “Methods for comparing rankings of search engine results”, Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 50, issue 10, pp. 1448 – 1463, July 2006.

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

[3] P. Diaconis, R. L. Graham, “Spearman’s footrule as a measure of disarray”,

Journal of the Royal Statistical Society, Series B (Methodological), pp. 262 - 268, 39(1977).

[4]http://www.eclipse.org/articles/article.php?file=ArticleJavaCodeManipulation_AST/index.html 21-05-2010

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

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

[7] G. K. Gill, and C. F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," IEEE Transactions on Software Engineering, vol. 17, no. 12, pp. 1284-1288, December 1991.

[8] T. Girba, S. Ducasse, and M. Lanza, “Yesterdays Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes,” in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, 2004, pp. 284–293.

[9] JDeodorant - Bad Smell Identification for Software Refactoring, http://www.jdeodorant.org/.

[10] http://www.jfree.org/jfreechart/ 07-05-2010.

[11] http://jmol.sourceforge.net/ 18-05-2010.

[12] H. Kagdi, M. L. Collard and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution",

Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, issue 2, pp. 77-131, March 2007.

[13] H. Kagdi and J. I. Maletic, “Software-change prediction: Estimated+actual,” in *Second International IEEE Workshop on Software Evolvability (SE)*, 2006, pp. 38–43.

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

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

[16] N. Tsantalis and A. Chatzigeorgiou, "Identification of Extract Method Refactoring Opportunities", 13th European Conference on Software Maintenance and Reengineering (CSMR'09), Kaiserslautern, Germany, pp. 119-128, March 2009.

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

[18] N. Tsantalis and A. Chatzigeorgiou, "Identification of Refactoring Opportunities Introducing Polymorphism", Journal of Systems and Software, vol. 83, issue 3, pp. 391-404, March 2010.

[19] http://en.wikipedia.org/wiki/Moving_average, 01-05-2010.

[20] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll, “PredictingSource Code Changes by Mining Change History,” *IEEE Trans. on Soft. Eng.*, vol. 30, no. 9, pp. 574–586, 2004.

[21] T. Zimmermann, A. Zeller, P. Weigerber, and S. Diehl, “Mining Version Histories to Guide Software Changes,” *IEEE Trans. on Soft. Eng.*, vol. 31, no. 6, pp. 429–445, 2005.