ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΝΟΠΟΙΗΣΗ ΕΡΓΑΛΕΙΩΝ ΕΝΤΟΠΙΣΜΟΥ ΟΣΜΩΝ ΚΩΔΙΚΑ

Διπλωματική Εργασία

του

Ιχτσή Απόστολου

Θεσσαλονίκη, Φεβρουάριος 2022

ΕΝΟΠΟΙΗΣΗ ΕΡΓΑΛΕΙΩΝ ΕΝΤΟΠΙΣΜΟΥ ΟΣΜΩΝ ΚΩΔΙΚΑ

Ιχτσής Απόστολος

Πτυχίο Εφαρμοσμένης Πληροφορικής, ΠΑΜΑΚ, 2018

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Χατζηγεωργίου Αλέξανδρος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22/02/2022

Χατζηγεωργίου Αλέξανδρος	Σακελλαρίου Ηλίας	Αμπατζόγλου Απόστολος
.....	.....	.....

Ιχτσής Απόστολος

.....

Περίληψη

Ο Fowler το 1999 εισήγαγε την έννοια των οσμών κώδικα. Οι οσμές αυτές, αποτελούν ουσιαστικά ενδείξεις κακής ποιότητας λογισμικού. Συνεπώς, η συσσώρευση αυτών σε ένα προϊόν λογισμικού, μπορεί να αποτελέσει σημαντικό παράγοντα επιβάρυνσης κατά την συντήρηση του προϊόντος, καθώς καθιστά δυσκολότερη την επιδιόρθωση λαθών και την προσθήκη νέων λειτουργιών. Για το λόγο αυτό, κρίνεται ιδιαίτερα σημαντική η αναγνώριση των υπαρχόντων οσμών και στη συνέχεια η αφαίρεση αυτών με τη χρήση αναδομήσεων.

Παρότι ένας έμπειρος προγραμματιστής έχει συνήθως τη δυνατότητα αναγνώρισης οσμών στον πηγαίο κώδικα λογισμικού, η διαδικασία αυτή μπορεί να γίνει ιδιαίτερα χρονοβόρα καθώς ένα σύστημα μπορεί να αποτελείται από εκατοντάδες κλάσεις και χιλιάδες γραμμές κώδικα ή και ακόμη περισσότερο. Λύση στο πρόβλημα αυτό λοιπόν, αποτελούν εργαλεία εντοπισμού οσμών στον κώδικα, τα οποία αναπτύσσονται από διάφορους ερευνητές και όχι μόνο. Τα εργαλεία αυτά, αναλαμβάνουν την εύρεση των υπαρχόντων οσμών σε ένα σύστημα και την παρουσίαση τους με έναν ευανάγνωστο τρόπο στον χρήστη. Αποτέλεσμα αυτού, είναι να εξοικονομείται πολύτιμος χρόνος από τους προγραμματιστές και να αποκτούν εύκολα μια γενική εικόνα της κατάστασης τους συστήματος.

Η χρήση όμως των εργαλείων αυτών, απαιτεί προσοχή από τους χρήστες. Για την εύρεση οσμών, τα τελευταία υλοποιούν διαφορετικές μεθόδους ή ακόμη και την ίδια αλλά με διαφορετικό τρόπο μεταξύ τους. Για το λόγο αυτό κρίνεται απαραίτητη η αξιολόγηση των επιμέρους εργαλείων, καθώς και η ύπαρξη συγκρίσεων μεταξύ τους. Μέσα στο πλαίσιο αυτό, έχουν πραγματοποιηθεί διάφορες μελέτες στη βιβλιογραφία για σύγκριση και αξιολόγηση εργαλείων, οι οποίες όμως περιλαμβάνουν συνήθως περιορισμένο αριθμό εργαλείων στα πειράματα τους ή δεν πραγματοποιούν καθόλου πειράματα και βασίζονται σε αναφορές των δημιουργών των εργαλείων.

Στην παρούσα εργασία λοιπόν, έγινε μελέτη των υπαρχόντων εργαλείων και ενσωμάτωση έξι (6) εξ' αυτών κάτω από ένα Eclipse plug-in με όνομα SmellDetectorMerger. Σκοπός αυτού, είναι η εύκολη και αυτοματοποιημένη χρήση και πρόσβαση στα επιμέρους εργαλεία, η εφαρμογή τους σε προϊόντα λογισμικού και η συγκεντρωτική εξαγωγή αποτελεσμάτων για τις υπάρχουσες οσμές. Με τον τρόπο αυτό,

μπορεί να πραγματοποιηθεί εύκολα και γρήγορα μια αξιολόγηση των αποτελεσμάτων και της αξιοπιστίας των εργαλείων αυτών.

Στα πλαίσια αυτής της διπλωματικής, χρησιμοποιήθηκε το plug-in που αναπτύχθηκε, στο Apache HTTP Components, το οποίο αποτελεί ένα έργο ανοιχτού κώδικα της Apache που παρέχει εργαλεία χαμηλού επιπέδου σχετικά με το HTTP και άλλα παρόμοια πρωτόκολλα. Αφού εντοπίστηκαν οι οσμές στο τελευταίο, εξήχθησαν σε ένα csv αρχείο το οποίο χρησιμοποιήθηκε για την εξαγωγή πληροφοριών. Με βάση αυτά, αποδεικνύεται ότι στην πλειοψηφία των περιπτώσεων όπου δύο (2) ή περισσότερα εργαλεία εντόπιζαν τον ίδιο τύπο οσμής, η συμφωνία των εργαλείων ως προς τις οσμές αυτού του τύπου που βρέθηκαν στο project ήταν πολύ μικρή. Μάλιστα, μόνο σε μία εξ' αυτών ήταν τόσο μεγάλος ο βαθμός συμφωνίας ώστε να θεωρηθεί σημαντικός.

Με βάση τα προηγούμενα, καταλήγουμε στο συμπέρασμα ότι οι οσμές κώδικα δεν είναι αυστηρά ορισμένες, με αποτέλεσμα οι διάφορες τεχνικές αλλά και οι υλοποιήσεις αυτών να αναφέρουν διαφορετικές περιπτώσεις οσμών. Για τον λόγο αυτό, ένας προγραμματιστής θα πρέπει να είναι ιδιαίτερα προσεκτικός πριν την επιλογή και χρήση ενός εργαλείου εντοπισμού οσμών.

Λέξεις Κλειδιά: Οσμές κώδικα, Εργαλεία εντοπισμού οσμών, Αναδομήσεις

Abstract

Fowler in 1999 introduced the term of code smells. These smells are actually indications of bad software quality. Therefore, their accumulation in a software product can play a very important role in degrading the maintenance phase, since it makes it harder to fix spotted issues and add new functionality. For this purpose, the identification of existing code smells and then their removal by using refactorings becomes vital.

Although an experienced developer is usually able to detect smells in the source code of a software product, this process can be very time-consuming since it can consist by hundreds of classes with thousands lines of code each or even more. A solution to this problem are code smell detection tools, which are developed by researchers as well as other individuals. These tools are responsible for the detection of existing smells in software projects and the representation of the latter in a user-friendly way. This can have a great impact since the developers save a valuable amount of time and they can also have an overview of the projects status.

Developers though need to be extra careful when using these tools. There are significant differences between the latter on the detection techniques they are utilizing or even when they use the same technique but implement it in a different way. Therefore, the evaluation of existing tools as well as comparisons between them become a necessity. In these terms, many literature reviews have been conducted in order to compare and evaluate code smell detection tools, but most of them use a limited amount of tools in their experiments or they don't do any experiments at all and their extracted conclusions are based on the reported results of the tools' authors.

In this thesis, a study is conducted on existing tools and six (6) of them have been integrated in an Eclipse plug-in named SmellDetectorMerger. The goal of implementing the latter, is the simple and automated use and access of the included tools, their application on software products and the extraction of aggregate results for the detected smells. A notable advantage is that it offers a fast and easy way to perform an evaluation of these results and the reliability of the tools.

An experiment using the plug-in was conducted in terms of this study on Apache HTTP Components, which is an open source project from Apache that provides low level tools related to the HTTP and other similar protocols. After the existing smells were detected, they were exported to a csv file which was used to extract data. The latter

showed that in the majority of the cases in which two (2) or more of the tools detected the same smell type, the agreement between the tools regarding the smells of a given type that were detected in the project was very low. Additionally, it’s worth mentioning that in only one of them the agreement degree was high enough to consider it important.

Based on the previous experiment, we conclude that code smells aren't strictly defined, leading to different reported cases between the existing techniques and their implementations. For this reason, a developer needs to be very careful before the selection and usage of a code smell detection tool.

Keywords: Code Smells, Code Smell Detection Tools, Refactorings

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τους ανθρώπους οι οποίοι με βοήθησαν και με στήριξαν σε όλη αυτή την προσπάθεια συγγραφής της παρούσας εργασίας. Αρχικά, θα ήθελα να ευχαριστήσω τον κύριο Χατζηγεωργίου Αλέξανδρο, επιβλέποντα καθηγητή της εργασίας, για την εμπιστοσύνη που μου έδειξε για το θέμα αυτό, καθώς και την πολύτιμη βοήθεια και συνεισφορά του καθ' όλη τη διάρκεια. Επίσης, θα ήθελα να ευχαριστήσω τον κύριο Αμπατζόγλου Απόστολο και τον κύριο Μήττα Νικόλαο για τη βοήθεια και τη συνεισφορά τους. Ενα μεγάλο ευχαριστώ θα ήθελα να πω επίσης στην οικογένεια μου καθώς και στη φίλη μου, Ζέρβα Ευτυχία, για την υπομονή και την στήριξη που μου έδειξαν καθ' όλη τη διάρκεια.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Οι οσμές κώδικα (code smells) αναφέρθηκαν πρώτη φορά από τον Fowler (Fowler, 1999) και αποτελούν ουσιαστικά κακές πρακτικές συγγραφής κώδικα. Η εύρεση αυτών και η αφαίρεση τους καθίσταται απαραίτητη σε προϊόντα λογισμικού, διότι η συσσώρευση τους μπορεί να αποτελέσει σημαντικό παράγοντα επιβάρυνσης στη συντήρηση του προϊόντος. Για το σκοπό αυτό, έχουν αναπτυχθεί διάφορα εργαλεία, που σε μεγάλο βαθμό αποτελούν προϊόν έρευνας, τα οποία αυτοματοποιούν τη διαδικασία εντοπισμού και παρουσίασης των οσμών που υπάρχουν σε ένα έργο λογισμικού. Τα εργαλεία αυτά, εντοπίζουν συνήθως διαφορετικά σύνολα οσμών, ενώ κάποιες εξ αυτών είναι περισσότερο δημοφιλής σε σύγκριση με άλλες και εντοπίζονται σε μεγαλύτερο αριθμό εργαλείων, όπως για παράδειγμα το Long Method και το God Class.

Για την εύρεση των οσμών, τα υπάρχοντα εργαλεία χρησιμοποιούν διάφορες τεχνικές, οι οποίες κάποιες φορές είναι κοινές μεταξύ τους, ώστε να εντοπίσουν τα σημεία στα οποία βρίσκονται οι οσμές αυτές. Ανάλογα με τον τρόπο με τον οποίο είναι υλοποιημένη η κάθε τεχνική στο εκάστοτε εργαλείο, επιτυγχάνονται διαφορετικά ποσοστά επιτυχίας εντοπισμού, τα οποία συνήθως μετρούνται με τους όρους της ακρίβειας (precision) και της ανάκλησης (recall). Πιο συγκεκριμένα, το precision δείχνει τι ποσοστό από το σύνολο των οσμών ενός τύπου που εντοπίστηκαν αντιστοιχεί σε πραγματικά υπαρκτές οσμές αυτού του τύπου. Από την άλλη, το recall δείχνει τι ποσοστό από το σύνολο των οσμών ενός τύπου που αποτελούν πραγματικά υπαρκτές οσμές ανακτήθηκαν τελικά από το εργαλείο.

Για την εύκολη σύγκριση λοιπόν της αποτελεσματικότητας των εργαλείων, κρίνεται ιδιαίτερα ενδιαφέρουσα και χρήσιμη η ύπαρξη ενός τρόπου για εύκολη και όσο το δυνατόν αυτόματη αξιολόγηση των αποτελεσμάτων που επιστρέφει το εκάστοτε εργαλείο καθώς και τις ομοιότητες και διαφορές τους.

1.2 Σκοπός – Στόχοι

Σκοπός της παρούσας μελέτης είναι η δημιουργία λογισμικού, το οποίο θα ενσωματώνει ένα πλήθος υπαρχόντων εργαλείων εντοπισμού οσμών, με αποτέλεσμα την εύκολη και ταυτόχρονη χρήση τους για ανάλυση έργων λογισμικού και επιστροφή των συγκεντρωτικών αποτελεσμάτων που εντοπίζουν. Για το σκοπό αυτό λοιπόν,

πραγματοποιείται μια βιβλιογραφική ανασκόπηση σε πηγές οι οποίες είτε παρουσιάζουν υπάρχοντα εργαλεία είτε συγκρίσεις μεταξύ τους, με αποτέλεσμα να γίνει μια προσπάθεια ενσωμάτωσης όσο το δυνατόν περισσότερων εξ' αυτών.

1.3 Βασική Ορολογία

Java ARchive (JAR): Είναι ένα αρχείο το οποίο συνήθως έχει επέκταση *.jar* και περιλαμβάνεις αρχεία Java με επέκταση *.class* τα οποία μπορούν να εκτελεστούν από την εικονική μηχανή της Java (Java Virtual Machine – JVM), καθώς και άλλα αρχεία που σχετίζονται με αυτά, όπως εικόνες κτλ (JAR (file format), 2022)

Batch file: Πρόκειται για ένα αρχείο το οποίο συνήθως έχει επέκταση *.bat* και περιλαμβάνει εντολές οι οποίες εκτελούνται από τη γραμμή εντολών. Το αρχείο αυτό είναι ουσιαστικά ένα αρχείο απλού κειμένου, το οποίο περιέχει εντολές που ο χρήστης θέλει να εκτελεστούν μαζί (Batch file, 2022)

Comma-Separated Values (CSV): Ένα comma-separated values (CSV), είναι ένα αρχειο με επέκταση *.csv* και περιλαμβάνει, όπως δηλώνεται και από το όνομα του, δεδομένα τα οποία χωρίζονται με κόμμα. Τα δεδομένα που αποθηκεύονται στο αρχείο αυτό έχουν συγκεκριμένη δομή και κάθε γραμμή αποτελεί και μια εγγραφή. Για το λόγο αυτό, όλες οι γραμμές έχουν τον ίδιο αριθμό στοιχείων χωρισμένων με κόμμα (Comma-separated values, 2022)

eXtensible Markup Language (XML): Πρόκειται για μια γλώσσα που χρησιμοποιείται για την αποθήκευση πληροφορίας με δομημένο τρόπο, ο οποίος είναι κατανοητός και από ανθρώπους και από υπολογιστές. Επίσης, περιλαμβάνει κανόνες οι οποίοι ορίζουν τον τρόπο αναπαράστασης (XML, 2022)

Unified Modeling Language (UML): Είναι μια γλώσσα μοντελοποίησης, η οποία χρησιμοποιείται για την αναπαράσταση της σχεδίασης ενός συστήματος (Unified Modeling Language, 2022)

1.4 Διάρθρωση της μελέτης

Το υπόλοιπο της παρούσας εργασίας οργανώνεται ως εξής. Στην Ενότητα 2, γίνεται μια παράθεση κάποιων θεωρητικών και τεχνικών γνώσεων που απαιτούνται. Στην Ενότητα 3 υπάρχει μια παρουσίαση σχετικών μελετών από τη βιβλιογραφία. Στην Ενότητα 4, παρουσιάζεται η μεθοδολογία που ακολουθήθηκε, το εργαλείο που αναπτύχθηκε καθώς και τα αποτελέσματα τα οποία προέκυψαν από ένα πείραμα που

διεξάχθηκε. Τέλος, στην Ενότητα 5, γίνεται μια συζήτηση γύρω από τα συμπεράσματα που εξήχθησαν καθώς και προτάσεις για μελλοντική μελέτη.

2 Θεωρητικό Υπόβαθρο

Στην ενότητα αυτή, παρουσιάζονται γνώσεις θεωρητικές και τεχνικές, τις οποίες πρέπει να έχει κάποιος προκειμένου να κατανοήσει όσα παρουσιάζονται στο κυρίως κομμάτι της εργασίας που αφορά τη μεθοδολογία που ακολουθήθηκε, καθώς και τα αποτελέσματα τα οποία εξήχθησαν. Για το σκοπό αυτό, η ενότητα έχει χωριστεί σε δύο επιμέρους υποενότητες. Στην πρώτη, παρουσιάζονται κάποια θεωρητικά στοιχεία σχετικά με έννοιες που χρησιμοποιήθηκαν, ενώ στη συνέχεια ακολουθούν κάποια τεχνικά στοιχεία, ώστε να παρουσιαστούν διάφορα εργαλεία και τεχνολογίες που αξιοποιήθηκαν.

2.1 Θεωρητικά στοιχεία

2.1.1 Αναδομήσεις

Καθ' όλη τη διάρκεια ανάπτυξης λογισμικού, κώδικας προστίθεται συνεχώς αλλά και τροποποιείται ο υπάρχων, με αποτέλεσμα πολλές φορές να υποβαθμίζεται η ποιότητα του, καθιστώντας με αυτόν τον τρόπο δυσκολότερη τη μελλοντική συντήρηση του και την ανάπτυξη νέων λειτουργιών σε αυτό. Για το λόγο αυτό, κρίνεται απαραίτητη η επένδυση χρόνου από τους προγραμματιστές με σκοπό την πραγματοποίηση αναδομήσεων στον υπάρχοντα κώδικα ενός προϊόντος. Οι αναδομήσεις (refactorings) (Fowler, 1999) στον κώδικα είναι αλλαγές στην δομή ενός προγράμματος, οι οποίες, ενώ δεν επηρεάζουν τον τρόπο λειτουργίας του, καθιστούν ευκολότερη τη συντήρηση και κατανόηση του.

Στο βιβλίο του, ο Fowler, έχει συγκεντρώσει ένα σύνολο 72 αναδομήσεων, τις οποίες ο ίδιος έχει χρησιμοποιήσει κατά τη διάρκεια των ετών και τις έχει διαχωρίσει σε 7 κατηγορίες, ανάλογα με το είδος των αλλαγών που επιφέρουν στον κώδικα. Για κάθε μια εξ' αυτών, έχει προσθέσει ένα όνομα, μια περιγραφή του τρόπου λειτουργίας της, κίνητρο για τη χρήση της, βήματα για την εφαρμογή της σε ένα σύστημα, καθώς και απλουστευμένα παραδείγματα για να γίνει ευκολότερα κατανοητή. Παρακάτω παρουσιάζονται τρεις από αυτές, οι οποίες είναι ιδιαίτερα σημαντικές.

- **Εξαγωγή μεθόδου (Extract method):** Πρόκειται για μια αναδόμηση η οποία εφαρμόζεται σε μακροσκελείς μεθόδους, με σκοπό να εξαχθεί μία μέθοδος ή περισσότερες, η οποία θα ενσωματώνει κομμάτια της αρχικής που έχουν κοινό σκοπό. Με τον τρόπο αυτό, η μεγαλύτερη μέθοδος

απλουστεύεται, καθώς πλέον αποτελείται από λιγότερες γραμμές και γίνεται ευκολότερη η κατανόηση της. Επίσης, οι μικρότερες και πιο στοχευμένες μέθοδοι που περιλαμβάνουν κώδικα για μια μόνο συγκεκριμένη λειτουργία, είναι πιθανότερο να επαναχρησιμοποιηθούν.

- **Μετακίνηση μεθόδου (Move method):** Είναι μια ακόμη αναδόμηση η οποία σχετίζεται με μεθόδους, όπως φαίνεται και από το όνομα της. Η συγκεκριμένη εφαρμόζεται σε μεθόδους οι οποίες χρησιμοποιούν ή χρησιμοποιούνται περισσότερο από στοιχεία άλλης κλάσης, συγκριτικά με την κλάση στην οποία βρίσκονται. Στόχος εδώ, είναι η μεταφορά της μεθόδου αυτής στην κλάση με την οποία σχετίζεται περισσότερο. Αυτό που επιτυγχάνεται με αυτόν τον τρόπο, είναι η ελάττωση της σύζευξης μεταξύ των κλάσεων ενός συστήματος, δηλαδή καθίστανται περισσότερο ανεξάρτητες μεταξύ τους, με αποτέλεσμα αλλαγές που τυχόν γίνουν στη μια, να είναι λιγότερο πιθανό να επιφέρουν αλλαγές και στην άλλη.
- **Extract Class:** Όπως υποδηλώνεται και από το όνομα της, η αναδόμηση αυτή σχετίζεται με κλάσεις. Εφαρμόζεται σε αυτές οι οποίες είναι πολύ μεγάλες και περιέχουν λειτουργικότητα η οποία θα μπορούσε να μοιραστεί σε δύο ή περισσότερες κλάσεις. Σκοπός αυτής της αναδόμησης, είναι ο διαχωρισμός των μερών (μεταβλητών και μεθόδων) μιας κλάσης που εκτελούν μια συγκεκριμένη λειτουργία, και η δημιουργία μιας ξεχωριστής κλάσης που θα τα περιέχει. Με τον τρόπο αυτό, η αρχική κλάση μπορεί να γίνει ευκολότερα κατανοητή αλλά και να συντηρηθεί καθώς πλέον έχει μία μόνο λειτουργία.

2.1.2 Οσμές κώδικα

Ο κώδικας ενός προϊόντος λογισμικού μπορεί να αλλάζει ανά τακτά χρονικά διαστήματα, διορθώνοντας σφάλματα ή προσθέτοντας καινούρια λειτουργικότητα. Κατά την πραγματοποίηση αυτών των αλλαγών, οι προγραμματιστές δεν χρησιμοποιούν πάντα βέλτιστες πρακτικές, με αποτέλεσμα ο νέος κώδικας που γράφεται ενδεχομένως να μην είναι τόσο καλής ποιότητας. Οι οσμές κώδικα είναι τα τμήματα αυτά του κώδικα, στα οποία δεν εφαρμόζονται οι βέλτιστες πρακτικές υλοποίησης (Fowler, 1999). Η ύπαρξη οσμών δεν συνεπάγεται άμεσα την ύπαρξη προβλήματος στη λειτουργία του προϊόντος, αλλά η συσσώρευση τους οδηγεί σε δυσκολίες στη συντήρηση και επέκταση του. Για το

λόγο αυτό, η απομάκρυνση τους καθίσταται απαραίτητη και επιτυγχάνεται μέσω των αναδομήσεων που συζητήθηκαν στην προηγούμενη ενότητα.

Συνολικά στο βιβλίο του Fowler παρουσιάζονται 22 οσμές. Από αυτές, κάποιες αφορούν προβλήματα υλοποίησης που σχετίζονται με κλάσεις ενώ οι υπόλοιπες σχετίζονται με προβλήματα σε μεθόδους. Παρακάτω, στον Πίνακα 1, παρουσιάζονται ονομαστικά όλες οι οσμές, καθώς και μια μικρή επεξήγηση σχετικά με τα χαρακτηριστικά της κάθε μιας.

Πίνακας 1: Οσμές και τα χαρακτηριστικά τους

Οσμή	Χαρακτηριστικά
Duplicate Code	Αφορά τη χρήση του ίδιου κώδικα σε περισσότερα από ένα σημεία μέσα σε ένα πρόγραμμα
Long Method	Πρόκειται για μεθόδους οι οποίες είναι ιδιαίτερα μακροσκελής, με αποτέλεσμα να καθίσταται δυσκολότερη η κατανόηση της λειτουργίας τους
Large Class	Συναντάται σε κλάσεις με μεγάλο μέγεθος, οι οποίες συνήθως αποτελούνται από πληθώρα παραμέτρων και πολλές ή/και μεγάλες μεθόδους
Long Parameter List	Αφορά μεθόδους οι οποίες έχουν πολύ μεγάλο αριθμό παραμέτρων στη δήλωση τους
Divergent Change	Είναι προϊόν μη σωστής δόμησης του κώδικα, με αποτέλεσμα μια κλάση να απαιτεί πολλές αλλαγές κατά τη συντήρηση του προγράμματος
Shotgun Surgery	Επίσης προϊόν μη σωστής δόμησης, όπου μια αλλαγή κατά τη συντήρηση του προγράμματος επιφέρει μικρές αλλαγές σε πολλά επιμέρους στοιχεία του
Feature Envy	Πρόκειται για μια μέθοδο ή μέρος αυτής, που χρησιμοποιεί περισσότερα στοιχεία άλλης κλάσης συγκριτικά με αυτή στην οποία είναι δηλωμένη
Data Clumps	Είναι μεταβλητές ενός προγράμματος που συνυπάρχουν σε διαφορετικά σημεία μέσα στον κώδικα αντί να βρίσκονται συγκεντρωμένα σε μια ξεχωριστή κλάση
Primitive Obsession	Αφορά την εκτεταμένη χρήση των βασικών τύπων δεδομένων που προσφέρονται από μια γλώσσα, αντί για την κατασκευή κλάσεων που θα περιέχουν τα τελευταία ομαδοποιημένα

Switch Statements	Συναντάται σε περιπτώσεις όπου ένα switch statement υπάρχει πολλές φορές μέσα σε ένα πρόγραμμα και επομένως μια αλλαγή (π.χ. προσθήκη ενός ελέγχου) συνεπάγεται την ανάγκη για τροποποίηση όλων των σημείων που το switch εντοπίζεται
Parallel Inheritance Hierarchies	Πρόκειται για μια περίπτωση Shotgun Surgery, στην οποία η δημιουργία υποκλάσης για μια κλάση, επιβάλει και τη δημιουργία υποκλάσης για μια άλλη
Lazy Class	Είναι κλάσεις, οι οποίες δεν προσφέρουν ή και καθόλου αρκετά στο πρόγραμμα
Speculative Generality	Αφορά κλάσεις ή μεθόδους που έχουν κατασκευαστεί για κάποιο σκοπό αλλά τελικά δεν χρησιμοποιήθηκαν ποτέ
Temporary Field	Είναι μεταβλητές οι οποίες χρησιμοποιούνται μόνο σε συγκεκριμένες περιπτώσεις κατά τη λειτουργία ενός αντικειμένου
Message Chains	Συναντάται σε περιπτώσεις που υπάρχει μεγάλη ακολουθία κλήσεων σε μεθόδους αντικειμένων για να αποκτήσει η αρχική κλάση πρόσβαση σε μια μεταβλητή ή σε κάποια λειτουργία
Middle Man	Πρόκειται για μονάδες ενός προγράμματος που είναι υλοποιημένες με τέτοιο τρόπο ώστε οι περισσότερες μέθοδοί του πραγματοποιούν τη ζητούμενη λειτουργία καλώντας μεθόδους άλλων μονάδων
Inappropriate Intimacy	Αφορά κλάσεις οι οποίες έχουν απευθείας πρόσβαση στις μεταβλητές άλλης κλάσης
Alternative Classes with Different Interfaces	Είναι ουσιαστικά κλάσεις οι οποίες πραγματοποιούν την ίδια λειτουργεία αλλά με διαφορετικά ονόματα στις μεθόδους τους
Incomplete Library Class	Αναφέρεται σε βιβλιοθήκες οι οποίες δεν μπορούν να τροποποιηθούν προκειμένου να ικανοποιούν κάποιες περαιτέρω ανάγκες των χρηστών
Data Class	Κλάσεις, των οποίων η μόνη ιδιότητα είναι να διαθέτουν μεταβλητές για αποθήκευση δεδομένων, καθώς και μεθόδους που προσφέρουν πρόσβαση σε αυτές
Refused Bequest	Πρόκειται για κλάσεις οι οποίες δεν υλοποιούν όλες τις

	μεθόδους που κληρονομούν από μια υπερκλάση
Comments	Αφορά κλάσεις που έχουν πολλά σχόλια αντί να δοθεί βάση στη βελτίωση της κατανοησιμότητας του κώδικα

2.1.3 Τεχνικές εντοπισμού οσμών

Όπως έχει ήδη αναφερθεί, έχει ιδιαίτερη σημασία ο εντοπισμός οσμών σε ένα σύστημα και η απομάκρυνση τους. Για το λόγο αυτό, έχουν γίνει προσπάθειες για την ανάπτυξη τεχνικών οι οποίες στοχεύουν στον εντοπισμό οσμών σε ένα σύστημα. Οι (Menshawy, Yousef, & Salem, 2021) έχουν μελετήσει και παρουσίασει ένα σύνολο τεχνικών εντοπισμού οσμών, οι οποίες έχουν προταθεί από ερευνητές στη βιβλιογραφία κατά τη διάρκεια των ετών. Συνολικά, οι τεχνικές αυτές έχουν χωριστεί σε επτά (7) κατηγορίες, οι οποίες παρουσιάζονται συνοπτικά παρακάτω:

- **Χειροκίνητη προσέγγιση (manual approach):** Στην περίπτωση αυτή, ο ίδιος ο προγραμματιστής αναλμβάνει να εντοπίσει τις οσμές με βάση τον ορισμό τους κοιτώντας τον κώδικα. Βέβαια η διαδικασία αυτή μπορεί να είναι ιδιαίτερα χρονοβόρα, ειδικά σε μεγάλα συστήματα.
- **Χρήση μετρικών (metrics-based):** Εδώ, τα διάφορα μέρη του προγράμματος ελέγχονται με σκοπό τον υπολογισμό συγκεκριμένων μετρικών, π.χ. το πλήθος των γραμμών κώδικα, με στόχο την λήψη αποφάσεων συγκρίνοντας τις τιμές που υπολογίστηκαν με κάποιες τιμές-όρια που έχουν τεθεί. Στην περίπτωση αυτή, μεγάλη προσοχή απαιτείται κατά των ορισμό των ορίων, καθώς οι όχι και τόσο σωστά ορισμένες τιμές μπορεί να οδηγήσουν σε εφαλμένα αποτελέσματα.
- **Χρήση αναζήτησης (search-based):** Περιλαμβάνει τη χρήση αλγορίθμων και κανόνων. Στην κατηγορία αυτή ανήκουν τόσο ευρετικές μέθοδοι, όπως για παράδειγμα οι ευρετικοί αλγόριθμοι, όσο και τεχνητή νουμοσύνη.
- **Παρακολούθηση ιστορικού αλλαγών (history-based):** Στην τεχνική αυτή, αναλύονται οι διάφορες αλλαγές που πραγματοποιούνται κατά τη διάρκεια ανάπτυξης και συντήρησης με σκοπό να βρεθούν οσμές που εισάγονται στην πορεία. Συνεπώς, η τεχνική αυτή μπορεί να χρησιμοποιηθεί εφόσον πρώτα έχει αναπτυχθεί μέρος του προϊόντος.

- **Χρήση κάποιας στρατηγικής (strategy-based):** Αφορά τη χρήση ενός σετ κανόνων, λογικών εκφράσεων και μετρικών. Ουσιαστικά οι κανόνες αυτοί ορίζονται με σκοπό τον εντοπισμό συγκεκριμένων οσμών και αφορούν τη χρήση κάποιων μετρικών. Και εδώ απαιτείται ιδιαίτερη προσοχή κατά την επιλογή κανόνων και μετρικών ώστε να είναι όσο πιο αποτελεσματική γίνεται η τεχνική.
- **Χρήση απεικόνισης (visualization-based):** Είναι μια μεικτή τεχνική στην οποία ένα εργαλείο αναλαμβάνει να εντοπίσει τις οσμές στον κώδικα και να τις παρουσιάσει στο χρήση απεικονίζοντας τες για παράδειγμα στο IDE το οποίο χρησιμοποιεί. Στη συνέχεια ο χρήστης αναλαμβάνει να τις μελετήσει.
- **Συνεργατική προσέγγιση (cooperative-based):** Στην περίπτωση αυτή, εφαρμόζονται αλγόριθμοι οι οποίοι τρέχουν παράλληλα προκειμένου να βελτιωθεί η απόδοση της αναζήτησης.

2.2 Τεχνικά στοιχεία

2.2.1 Java

Η Java (Arnold, Gosling, & Holmes, 2005), είναι μία γλώσσα αντικειμενοστρεφούς προγραμματισμού, η οποία κυκλοφόρησε το 1996 από την εταιρία Sun. Ένα πολύ σημαντικό πλεονέκτημα που προσφέρει, είναι ότι τα προγράμματα που γράφονται σε αυτή μπορούν να τρέξουν σε οποιαδήποτε συσκευή στην οποία υπάρχει εγκατεστημένη η Εικονική Μηχανή της Java (Java Virtual Machine – JVM). Λόγω της πληθώρας λειτουργικών συστημάτων και συσκευών που υπάρχουν, έχουν αναπτυχθεί διάφορες υλοποιήσεις του JVM, με αποτέλεσμα πλέον να υπάρχει κάποια έκδοση συμβατή με τις περισσότερες συσκευές.

Όπως προαναφέρθηκε, πρόκειται για μια αντικειμενοστρεφή γλώσσα, οπότε θεμελιώδη στοιχεία της αποτελούν οι κλάσεις (classes) και τα αντικείμενα (objects). Μια κλάση είναι ουσιαστικά ένα σχεδιάγραμμα, το οποίο ορίζει τον τρόπο παραγωγής αντικειμένων από αυτή. Συστατικά στοιχεία μιας κλάσης και κατ' επέκταση των αντικειμένων είναι οι μεταβλητές (variables) και οι μέθοδοι (methods). Οι μεταβλητές αποτελούν την κατάσταση (state) ενός αντικειμένου που παράγεται από μια κλάση, ενώ

οι μέθοδοι αποτελούν την συμπεριφορά (behavior) αυτού. Σκοπός των αντικειμένων είναι η μεταφορά μηνυμάτων μεταξύ τους μέσω των μεθόδων που υπάρχουν σε αυτά.

```
1  public class Main {
2
3      public static void main(String[] args) {
4          int x = 5;
5          int y = 10;
6
7          int sum = calculateSum(x, y);
8          System.out.print("The sum is: " + sum);
9      }
10
11     private static int calculateSum(int x, int y) {
12         return x + y;
13     }
14
15 }
```

Εικόνα 1: Παράδειγμα προγράμματος υπολογισμού αθροίσματος σε Java

Στην Εικόνα 1 παραπάνω, φαίνεται ένα πολύ μικρό παράδειγμα χρήσης της Java για τον υπολογισμό του αθροίσματος δύο αριθμών. Αρχικά, στη γραμμή 1 φαίνεται η δήλωση της κλάσης Main, ενώ στη συνέχεια ακολουθεί η δήλωση της μεθόδου main, η οποία αποτελεί τη βασική μέθοδο που πρέπει να υπάρχει τουλάχιστον μια φορά σε ένα πρόγραμμα, καθώς από αυτή θα ξεκινήσει η εκτέλεση του κώδικα κατά την έναρξη του προγράμματος. Έπειτα, στις γραμμές 4 και 5 γίνεται δήλωση δύο μεταβλητών, συγκεκριμένα των x και y με τιμές 5 και 10 αντίστοιχα, οι οποίες είναι τύπου int, δηλαδή ακέραιοι αριθμοί μεγέθους μέχρι 32bit. Στη συνέχεια, ακολουθεί μια ακόμη δήλωση μεταβλητής, μόνο που στη συγκεκριμένη περίπτωση, αυτό που αποθηκεύεται στη μεταβλητή sum είναι ο αριθμός που επιστρέφεται από τη μέθοδο *calculateSum(int, int)*, της οποία τη δήλωση και το σώμα βλέπουμε στις γραμμές 11 με 13. Η προηγούμενη μέθοδος δέχεται δύο αριθμούς τύπου int ως παραμέτρους και στις οποίες εισάγονται οι μεταβλητές x και y. Τέλος, στη γραμμή 8 έχουμε μια εντολή εκτύπωσης, η οποία εκτυπώνει στην κονσόλα το άθροισμα των δύο αυτών μεταβλητών, δηλαδή 15.

2.2.2 IDE

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment – IDE), είναι λογισμικό, το οποίο παρέχει στους προγραμματιστές ένα ολοκληρωμένο σύνολο εργαλείων τα οποία βοηθούν στην ανάπτυξη προγραμμάτων με σκοπό την αύξηση της παραγωγικότητας τους. Το σύνολο αυτό, περιλαμβάνει εργαλεία όπως η αυτόματη δημιουργία διαγραμμάτων UML από τον κώδικα ενός προγράμματος,

η αυτόματη συμπλήρωση κώδικα, η προβολή και επεξεργασία κειμένου πέραν της γλώσσας προγραμματισμού, πχ XML, η ύπαρξη browser μέσα στο IDE για προβολή αλλαγών απευθείας και άλλα πολλά (des Rivie` res & Wiegand, 2004).

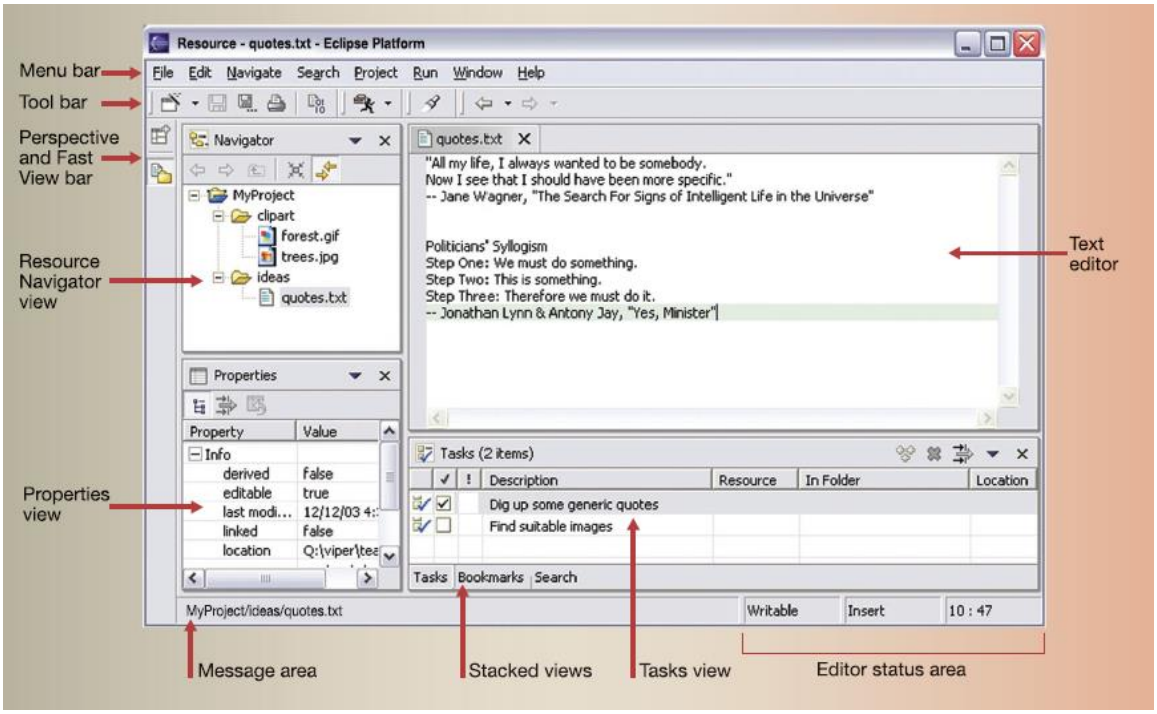
Ένα πολύ σημαντικό πλεονέκτημα που παρέχεται από ένα IDE είναι η επεκτασιμότητα του ως προς την προσθήκη νέων εργαλείων. Είτε πρόκειται για ελεύθερα είτε για εμπορικά συστήματα, προσφέρονται συνήθως τρόποι ώστε μεμονωμένοι προγραμματιστές ή ομάδες αυτών να μπορούν να αναπτύσσουν νέα εργαλεία τα οποία οι χρήστες μπορούν να αποκτήσουν και να ενσωματώσουν στο δική τους εγκατάσταση του IDE. Το γεγονός που καθιστά τη δυνατότητα αυτή ιδιαίτερα σημαντική, είναι ότι η ομάδα που αναπτύσσει ένα IDE δεν είναι δυνατό να αναπτύξει και τα εργαλεία που θα καλύψουν ολόκληρος το εύρος χρηστών που το χρησιμοποιούν. Αυτό πετυχαίνεται με την παροχή στους προγραμματιστές ενός Application Programming Interface (API) το οποίο είναι ουσιαστικά μια βιβλιοθήκη η οποία δίνει πρόσβαση στα διάφορα στοιχεία του IDE και επιτρέπει την εκμετάλλευση τους για τη δημιουργία τελικά ενός νέου εργαλείου (des Rivie` res & Wiegand, 2004).

2.2.3 Eclipse

Το Eclipse είναι ένα IDE ανοιχτού κώδικα το οποίο αναπτύχθηκε το 2001 από την εταιρεία IBM και μπορεί να χρησιμοποιηθεί τόσο στο λειτουργικό σύστημα Windows, όσο και στο Linux. Ενώ η γλώσσα προγραμματισμού η οποία χρησιμοποιήθηκε για την ανάπτυξη του είναι η Java, το εργαλείο υποστηρίζει τη συγγραφή προγραμμάτων σε πληθώρα άλλων γλωσσών. Όπως και στα υπόλοιπα IDEs, έτσι και στο Eclipse, προσφέρεται ένα API με στόχο την παροχή δυνατότητας ανάπτυξης εργαλείων (τα οποία στο Eclipse ονομάζονται plug-ins (βλ. ενότητα 2.2.4)) από τρίτους (des Rivie` res & Wiegand, 2004). Μέσω των plug-in, υποστηρίζεται η συγγραφή προγραμμάτων σε άλλες γλώσσες, η εκτέλεση και η απασφαλμάτωση τους, καθώς και πληθώρα άλλων λειτουργιών.

Παρακάτω, στην Εικόνα 2, απεικονίζεται ένα παράδειγμα της διεπαφής χρήστη που προσφέρει το IDE. Πάνω αριστερά φαίνεται το *Resource Navigator view*, το οποίο περιλαμβάνει όλα τα αρχεία που περιλαμβάνονται σε ένα έργο σε δεντρική μορφή. Ακριβώς κάτω από αυτό υπάρχει το *Properties view*, το οποίο περιέχει διάφορες πληροφορίες του επιλεγμένου στοιχείου από το *Resource Navigator view*. Επίσης, πάνω δεξιά απεικονίζεται ο *Text Editor*, όπου προβάλλονται τα περιεχόμενα του επιλεγμένου

αρχείου από το *Resource Navigator view*. Μέσω αυτού μπορεί ο χρήστης επίσης να πραγματοποιήσει αλλαγές στα αρχεία αυτά (des Rivie` res & Wiegand, 2004).



Εικόνα 2: Παράδειγμα διεπαφής του Eclipse (des Rivie` res & Wiegand, 2004)

2.2.4 Eclipse plug-in

Τα plug-ins του Eclipse είναι μικρές μονάδες οι οποίες προσφέρουν λειτουργικότητα σε ολόκληρο το σύστημα. Στην ουσία, πέρα από έναν μικρό πυρήνα του IDE, όλη η υπόλοιπη λειτουργικότητα προσφέρεται μέσω plug-ins. Κάθε plug-in είναι γραμμένο σε γλώσσα Java και αποτελείται από ένα Java Archive (JAR) αρχείο, το οποίο περιλαμβάνει τον κώδικα και όλα τα αρχεία όπως π.χ. εικόνες, τα οποία απαιτούνται για την εκτέλεση του (des Rivie` res & Wiegand, 2004). Εφόσον, όπως προαναφέρθηκε, οι πλειοψηφία των λειτουργιών στο Eclipse προσφέρεται μέσω των plug-ins, η δημιουργία ενός νέου απαιτεί συνήθως τη χρήση λειτουργικότητας κάποιου υπάρχοντος. Για το λόγο αυτό, κατά την ανάπτυξη ενός plug-in είναι απαραίτητος ο ορισμός των εξωτερικών plug-in τα οποία χρειάζονται για τη λειτουργία του τρέχοντος. Επίσης, κάποια plug-ins μπορεί, αντί να χρησιμοποιούν κάποιο εξωτερικό plug-in αυτό καθ' αυτό, να θέλουν απλώς να επεκτείνουν τη λειτουργία του. Και σε αυτή την περίπτωση απαιτείται αντίστοιχη μέριμνα συμπερίληψης του στο τρέχον. Συνδετικός κρίκος για όλα τα προηγούμενα είναι ένα αρχείο XML με όνομα plugin.xml, μέσα στο οποίο γίνονται οι απαιτούμενες δηλώσεις για τις ανάγκες του καινούριου plug-in από υπάρχοντα.

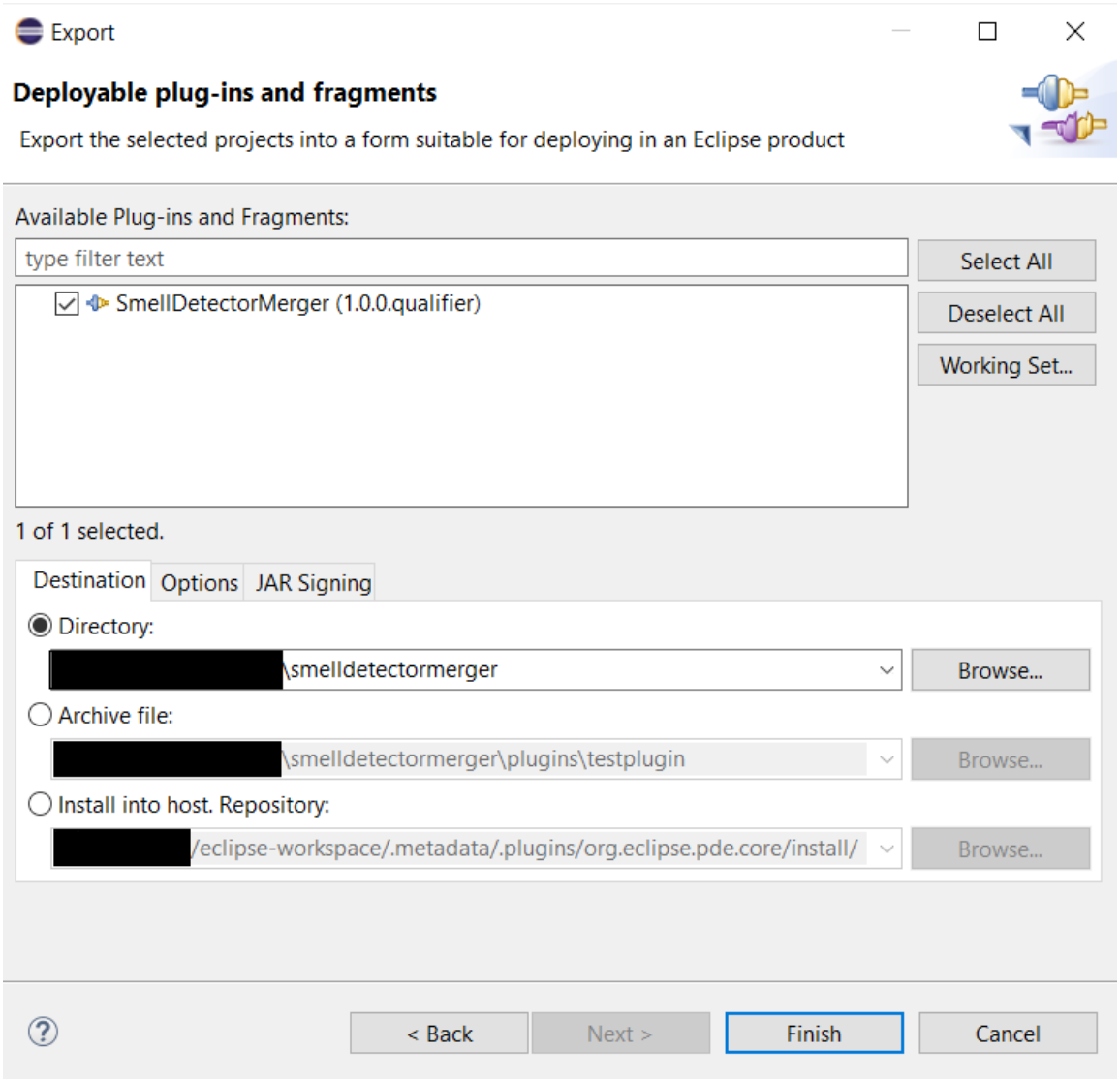
Το Eclipse υπάρχει διαθέσιμο σε διάφορες διανομές, ανάλογα με τις ανάγκες των χρηστών. Αυτό συνεπάγεται αντίστοιχα και το πλήθος των plug-in με τα οποία έρχεται προ εγκατεστημένα η συγκεκριμένη διανομή. Για παράδειγμα, ο χρήστης μπορεί να κατεβάσει την βασική έκδοση ή εναλλακτικά μπορεί να κατεβάσει την Java Enterprise Edition (JEE) έκδοση, η οποία περιλαμβάνει παραπάνω plug-ins που μπορούν να αξιοποιηθούν για τη συγγραφή εφαρμογών διαδικτύου (Web-based). Βέβαια, δεν είναι απαραίτητο να υπάρχουν προ εγκατεστημένα όλα τα plug-in που θα χρειαστεί ο χρήστης. Εφόσον προκύψει κάποια ανάγκη στην πορεία, ο χρήστης μπορεί να εγκαταστήσει κάποιο plug-in με έναν από τους ακόλουθους δύο τρόπους:

- **Eclipse Marketplace:** Πρόκειται για το επίσημο κατάστημα του Eclipse IDE. Ο χρήστης έχει πρόσβαση σε αυτό μέσω του ίδιου του IDE ακολουθώντας το μονοπάτι “Help” -> “Eclipse Marketplace…”. Επιλέγοντας τα προηγούμενα, θα ανοίξει ένα νέο παράθυρο το οποίο εμφανίζει κάποιες προτάσεις, αλλά ο χρήστης μπορεί να αναζητήσει κάποιο plug-in είτε με το όνομα του εφόσον το γνωρίζει, είτε πληκτρολογώντας κάποιες λέξεις σχετικές με αυτό. Για παράδειγμα αν πληκτρολογήσει *JavaFx*[1], που είναι μια βιβλιοθήκη γραφικών της Java, το Marketplace θα εμφανίσει μια σειρά plug-in σχετικά με αυτή. Από εκεί, ο χρήστης μπορεί να πατήσει το κουμπί *Install* και το plug-in θα εγκατασταθεί.
- **Μέσω jars:** Στην περίπτωση αυτή αντιστοιχούν plug-in τα οποία δεν υπάρχουν διαθέσιμα μέσω του Marketplace και ενδέχεται να έχουν αναπτυχθεί από τρίτους ή από τον ίδιο τον προγραμματιστή. Ειδικά στην περίπτωση που πρόκειται για plug-in που έχει αναπτυχθεί από τρίτους, απαιτείται ιδιαίτερη προσοχή ως προς την αξιοπιστία του. Εφόσον ο χρήστης διαθέτει ήδη το *jar* αρχείο του plug-in, μπορεί να το εγκαταστήσει είτε μέσω του Eclipse ακολουθώντας τις επιλογές “*Help*” -> “*Install New Software…”* και να ορίσει τη διαδρομή στο δίσκο όπου βρίσκεται το *jar*, είτε κάνοντας το επικόλληση στον φάκελο *dropins* που βρίσκεται στον κατάλογο που είναι εγκατεστημένο το Eclipse. Εναλλακτικά, εφόσον ο χρήστης διαθέτει τον κώδικα του plug-in, πρέπει

[1] https://openjfx.io/

πρώτα να το εισάγει στο Eclipse επιλέγοντας "*File*" -> "*Import*" -> "*Existing Projects into Workspace*" και στη συνέχεια να εξάγει το *jar* από αυτό. Για να εξάγει το τελευταίο, πρέπει να κάνει δεξί κλικ στο project που εισήγαγε μόλις και να επιλέξει "*Export…*" -> "*Deployable plug-ins and fragments*". Εφόσον εκτελέσει τα προηγούμενα, το Eclipse επιτρέπει στον χρήστη να εγκαταστήσει απευθείας το plug-in ή να εξάγει το *jar* αρχείο. Παράδειγμα των επιλογών αυτών φαίνεται στην Εικόνα 3.

Για τα plug-ins τα οποία έρχονται προ εγκατεστημένα με τη διανομή του Eclipse, καθώς και εκείνα τα οποία εγκαθίστανται στην πορεία μέσω του Marketplace, το Eclipse ελέγχει συχνά για τυχόν νέες εκδόσεις τους. Εφόσον βρεθεί κάποια, το IDE εμφανίζει σχετικό μήνυμα στον χρήστη, ο οποίος μπορεί να επιλέξει ποια θέλει να ενημερώσει και το Eclipse στη συνέχεια αναλαμβάνει τα υπόλοιπα.



Εικόνα 3: Παράδειγμα εξαγωγής jar του SmellDetectorMerger

2.2.5 Version Control Systems (VCS)

Ένα Version Control System (VCS), είναι ένα σύστημα το οποίο διαχειρίζεται τον κώδικα ενός έργου λογισμικού και τις αλλαγές που πραγματοποιούνται σε αυτό. Βασικό στοιχείο ενός VCS είναι η ύπαρξη ενός αποθετηρίου (repository), στο οποίο αποθηκεύεται ο κώδικας καθώς και όλες οι αλλαγές που πραγματοποιούνται στα αρχεία αυτού. Πλέον, η χρήση ενός VCS κρίνεται απαραίτητη καθώς τα οφέλη τα οποία προσφέρει είναι πολλά και χωρίς ένα VCS είναι από εξαιρετικά δύσκολη έως και αδύνατη η διαχείριση ενός έργου λογισμικού. Ένα πολύ σημαντικό πλεονέκτημα είναι ότι ο κάθε προγραμματιστής μπορεί να προβάλει εύκολα αλλαγές που έχουν κάνει άλλοι προγραμματιστές, ώστε να δουλεύει πάντα σε όσο το δυνατόν πιο ενημερωμένη έκδοση του κώδικα. Επίσης, όλες οι αλλαγές που γίνονται στα αρχεία από την αρχή χρήσης του VCS είναι αποθηκευμένες και επομένως καθίσταται ιδιαίτερα εύκολη η αναζήτηση σε αυτές για εξαγωγή πληροφορίων όπως τα ονόματα των προγραμματιστών που έχουν δουλέψει σε κάποιο αρχείο, τι είδους αλλαγές έχουν κάνει και άλλες. Ακόμη ένα πλεονέκτημα, είναι η δυνατότητα δημιουργία ενός νέου κλαδιού (branch), το οποίο μπορεί να χρησιμοποιηθεί για την ανάπτυξη μιας συγκεκριμένης λειτουργίας χωρίς να επηρεαστεί το βασικό κλαδί που χρησιμοποιείται από τους χρήστες. Με τον τρόπο αυτό, η νέα λειτουργία μπορεί να αναπτυχθεί και εν τέλει να ελεγχθεί στο νέο κλάδο και αφού τελικά επιβεβαιωθεί η σωστή λειτουργία της, να ενωθεί ο νέος κλάδος με τον βασικό ώστε να γίνει διαθέσιμη η νέα αυτή λειτουργιά στους χρήστες (Spinellis, 2005).

Παρακάτω παρουσιάζονται κάποιες από τις βασικές λειτουργίες, οι οποίες είναι κοινές μεταξύ των διαφόρων υλοποιήσεων των VCS:

- **checkout:** Μπορεί να εκτελεστεί κατά την πρώτη επαφή με το έργο λογισμικού που βρίσκεται στο VCS και κατεβάζει την τελευταία έκδοση του λογισμικού σε ένα τοπικό κατάλογο στον υπολογιστή του προγραμματιστή.
- **update:** Κάθε φορά που εκτελείται, κατεβάζει από το repository του VCS όλες τις αλλαγές που έχουν πραγματοποιηθεί από τους υπόλοιπους προγραμματιστές από την τελευταία φορά που έγινε update. Σωστή πρακτική αποτελεί η εκτέλεση του κάθε μέρα, την πρώτη φορά που έρχεται σε επαφή με τον κώδικα ο προγραμματιστής, αλλά και πριν ανεβάσει ο ίδιος κάποια αλλαγή, για να βεβαιωθεί ότι δεν έχουν γίνει αλλαγές από άλλους προγραμματιστές στα ίδια αρχεία στο ενδιάμεσο.

- **commit:** Ανεβάζει στο repository του VCS όλες τις αλλαγές που πραγματοποιήθηκαν από τον προγραμματιστή στα αρχεία που έχει επιλέξει ο ίδιος να εκτελεστεί η εντολή αυτή.
- **tag:** Χρησιμοποιείται για να δώσει μία ονομασία κατά τη δημιουργία μιας καινούριας έκδοσης του κώδικα.

Ένα από τα πιο γνωστά VCS το οποίο χρησιμοποιείται στις μέρες μας είναι το Git (Git, 2022). Το Git έχει τα χαρακτηριστικά που αναφέρθηκαν προηγουμένως καθώς και πληθώρα άλλων, και μπορεί να εκτελεστεί μέσω της γραμμής εντολών σε λειτουργικό σύστημα Windows και Linux. Ένας πολύ γνωστός πάροχος ο οποίος προσφέρει τη δυνατότητα χρήσης του Git, είναι το GitHub. Το τελευταίο, δίνει τη δυνατότητα αξιοποίησης των λειτουργίων του Git με σκοπό τη δημιουργία απομακρυσμένων αποθετηρίων και την διαχείριση τους. Εκτός από τη χρήση του Git, το GitHub προσφέρει και πολλές άλλες λειτουργίες, όπως η διαχείριση project, η παρακολούθηση σφαλμάτων (bug tracking) και άλλα. Επίσης, αξιζεί να σημειωθεί ότι είναι ευρέως διαδεδομένο στην κοινότητα των προγραμματιστών, καθώς μετράει 73 εκατομύρια χρήστες και 200 εκατομύρια αποθετήρια (GitHub, 2022).

3 Βιβλιογραφική επισκόπηση

Από την εισαγωγή της έννοιας των οσμών κώδικα, έγιναν πολλές προσπάθειες από διάφορους ερευνητές και όχι μόνο για την δημιουργία εργαλείων τα οποία υλοποιούν διαφορετικές τεχνικές εντοπισμού των οσμών αυτών και στη συνέχεια της προβολής τους με ποικίλους τρόπους στους τελικούς χρήστες. Παρακάτω, παρουσιάζονται ορισμένες από τις προσπάθειες αυτές, οι οποίες μάλιστα αποτελούν και μέρος του plug-in που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας. Τα έργα αυτά, έχουν αναπτυχθεί στη γλώσσα προγραμματισμού Java και στην πλειοψηφία τους υποστηρίζουν τον εντοπισμό οσμών σε project τα όποια έχουν γραφτεί στην προαναφερθείσα γλώσσα.

3.1 CheckStyle

Πρόκειται για ένα εργαλείο (CheckStyle, 2022) ανεπτυγμένο σε Java, μέσω του οποίου μπορούν να οριστούν συγκεκριμένα πρότυπα συγγραφής κώδικα, με σκοπό την βοήθεια των προγραμματιστών, ώστε να συμμορφώνονται με τα πρότυπα αυτά κατά την ανάπτυξη λογισμικού. Το εργαλείο αυτό, μπορεί να εκτελεστεί μέσω της γραμμής εντολών, προσφέροντας πληθώρα παραμέτρων. Επίσης, έχουν γίνει διάφορες ανεξάρτητες υλοποιήσεις από τρίτους, οι οποίοι έχουν δημιουργήσει plug-ins που ενσωματώνουν το CheckStyle και μπορούν να εγκατασταθούν σε διάφορα IDEs. Μια συγκεντρωτική αναφορά στα plug-ins αυτά μπορεί κάποιος να βρει στη σελίδα του εργαλείου[2]. Επίσης, μια σημαντική λειτουργία που προσφέρει το CheckStyle, είναι ότι δίνει τη δυνατότητα στον χρήστη να ορίσει ο ίδιος τους δικούς κανόνες – πρότυπα, ώστε να καθοδηγείται ο προγραμματιστής με βάση αυτούς. Όλα αυτά τελικά τα περιλαμβάνει ο χρήστης σε ένα XML αρχείο, το οποίο παρέχει ως παράμετρο κατά την εκτέλεση του εργαλείου.

Μέσα στους διάφορους κανόνες που προσφέρονται, περιλαμβάνονται και κάποιοι, οι οποίοι ελέγχουν την ύπαρξη συγκεκριμένων οσμών κώδικα. Αναλυτικότερα, μπορεί κανείς να προσθέσει κανόνες για τον εντοπισμό των τριών οσμών που φαίνονται στον Πίνακα 2. Βέβαια, αξίζει να σημειωθεί πως οι έλεγχοι αυτοί γίνονται με πολύ αυστηρό τρόπο, καθώς απαιτείται ο ορισμός συγκεκριμένης τιμής, η υπέρβαση της οποίας οδηγεί σε παράβαση του κανόνα. Για παράδειγμα, στην περίπτωση του Long

[2] https://checkstyle.org/index.html#Related_Tools

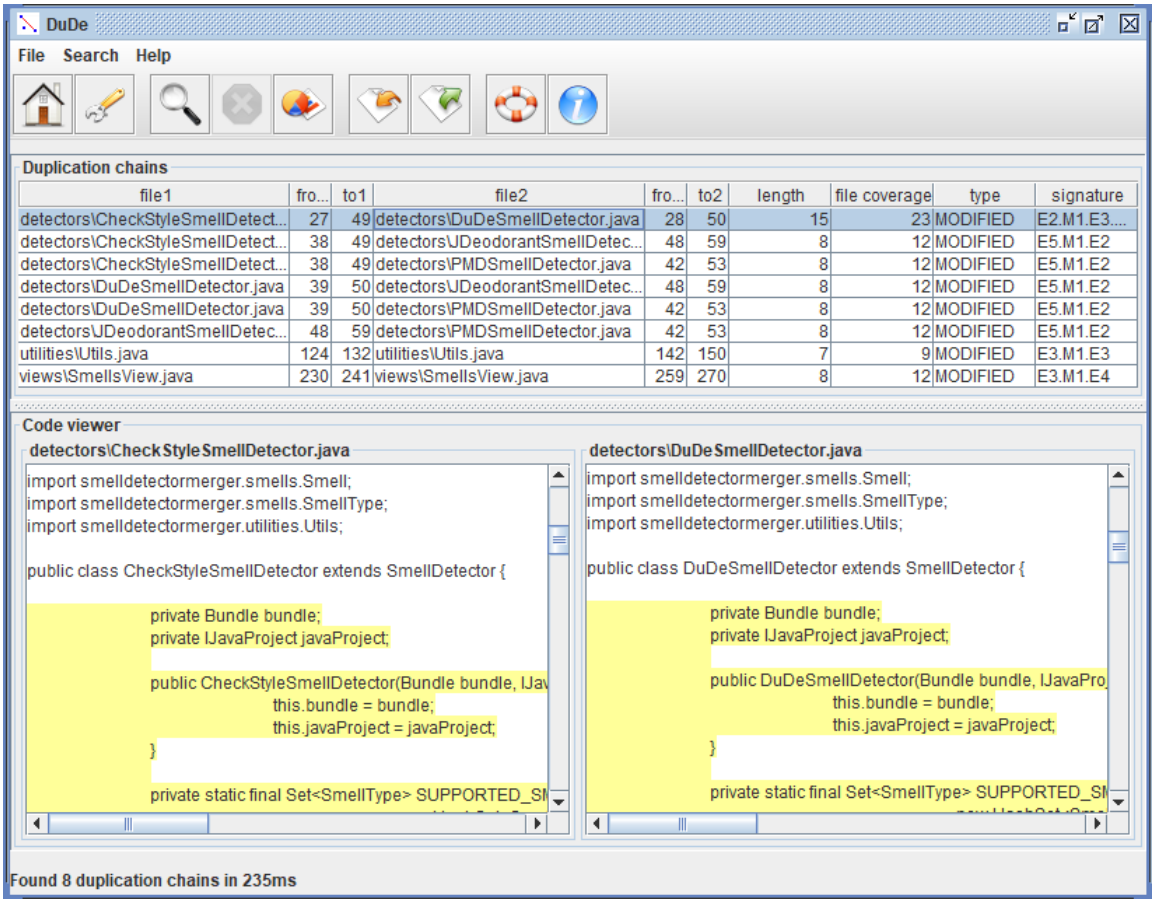
Method, η προεπιλεγμένη τιμή είναι 150, που σημαίνει ότι μέθοδοι με μεγαλύτερο πλήθος γραμμών από αυτό θεωρούνται ως προβληματικές. Σχετικά με τις άλλες δύο οσμές, οι προεπιλεγμένες τιμές τους είναι 2.000 και 7 αντίστοιχα. Όλες αυτές οι τιμές είναι παραμετροποιήσιμες και από τους ίδιους τους χρήστες με τιμές που τους ενδιαφέρουν.

3.2 DuDe

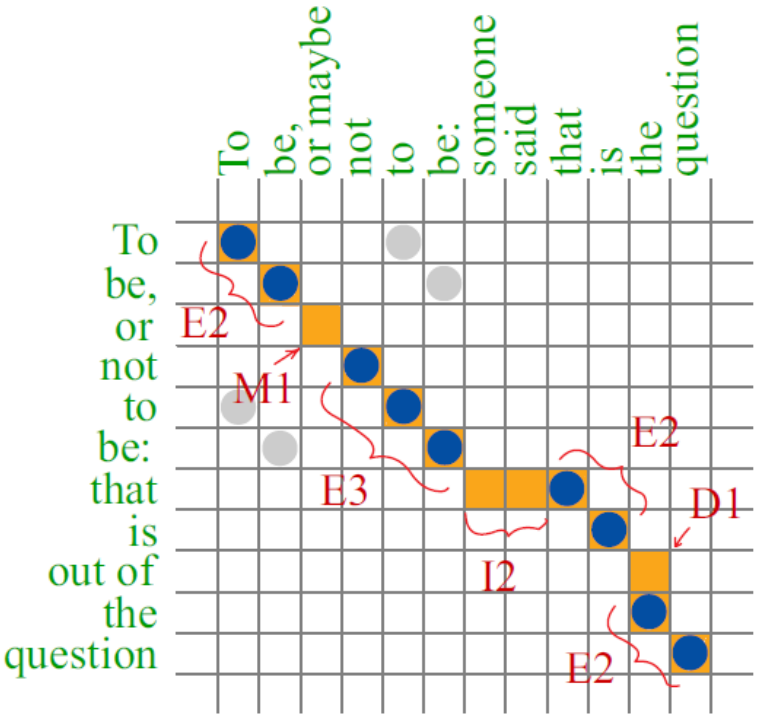
Ένα εργαλείο (Wettel & Marinescu, 2005), το οποίο έχει αναπτυχθεί για τον εντοπισμό διπλότυπων (duplicates) στον κώδικα. Πρόκειται για ένα προϊόν έρευνας που έχει αναπτυχθεί σε Java, το οποίο υλοποιεί μια τεχνική εντοπισμού διπλοτύπων που προτάθηκε από τους ίδιους του ερευνητές που το δημιούργησαν και φέρει σημαντικά πλεονεκτήματα. Το εργαλείο αυτό είναι διαθέσιμο προς εκτέλεση μέσω ενός *jar* αρχείου. Οι ενδιαφερόμενοι χρήστες, μπορούν είτε να αξιοποιήσουν την γραφική διεπαφή (GUI) που προσφέρει, είτε να το εκτελέσουν μέσω της γραμμής εντολών, ώστε να το εισάγουν επίσης, εφόσον το επιθυμούν, σε δικές τους υλοποιήσεις. Παρακάτω, στην Εικόνα 4, φαίνεται ένα παράδειγμα χρήσης του GUI στον κώδικα του SmellDetectorMerger.

Όπως προαναφέρθηκε, το σημαντικότερο χαρακτηριστικό του εργαλείου αυτού είναι ότι υλοποιεί μια διαφορετική τεχνική εντοπισμού διπλοτύπων. Οι ερευνητές εδώ, πρότειναν τη χρήση των αλυσίδων διπλοτύπων (duplication chains), οι οποίες συνιστώνται από αλληλουχίες κρίκων που αντιστοιχούν σε ακριβείς αντιγραφές και οι οποίες ενδέχεται να διαχωρίζονται από κρίκους που αντιστοιχούν σε μεταξύ τους τροποποιημένα κομμάτια. Έτσι, η τεχνική αυτή μπορεί να εντοπίσει διπλότυπα κώδικα, τα οποία όμως έχουν τροποποιηθεί μερικώς σε ορισμένα σημεία τους.

Προκειμένου να γίνει ευκολότερα κατανοητή η λειτουργία της προτεινόμενης τεχνικής, παρουσιάζεται ένα παράδειγμα στην Εικόνα 5. Εδώ, τα τετράγωνα τα οποία έχουν χρωματιστεί με κίτρινο σχηματίζουν μια αλυσίδα, η οποία αντιστοιχεί σε ένα διπλότυπο που περιέχει μικρές τροποποιήσεις. Πιο συγκεκριμένα, τα τετράγωνα στα οποία περιέχονται μπλε κύκλοι, αντιστοιχούν σε κρίκους ακριβούς (exact – E) αντιγραφής, ενώ τα υπόλοιπα ανάμεσα τους αντιστοιχούν σε κρίκους που αντιπροσωπεύουν τροποποιημένα (inserted/deleted/modified – I/D/M) κομμάτια. Ο αριθμός δίπλα από το εκάστοτε γράμμα στην Εικόνα 5, αντιστοιχεί στο πλήθος των συνεχόμενων κρίκων αυτού του τύπου.



Εικόνα 4: Παράδειγμα χρήσης του DuDe στο SmellDetectorMerger



Εικόνα 5: Παράδειγμα duplication chain του DuDe (Wettel & Marinescu, 2005)

3.3 PMD

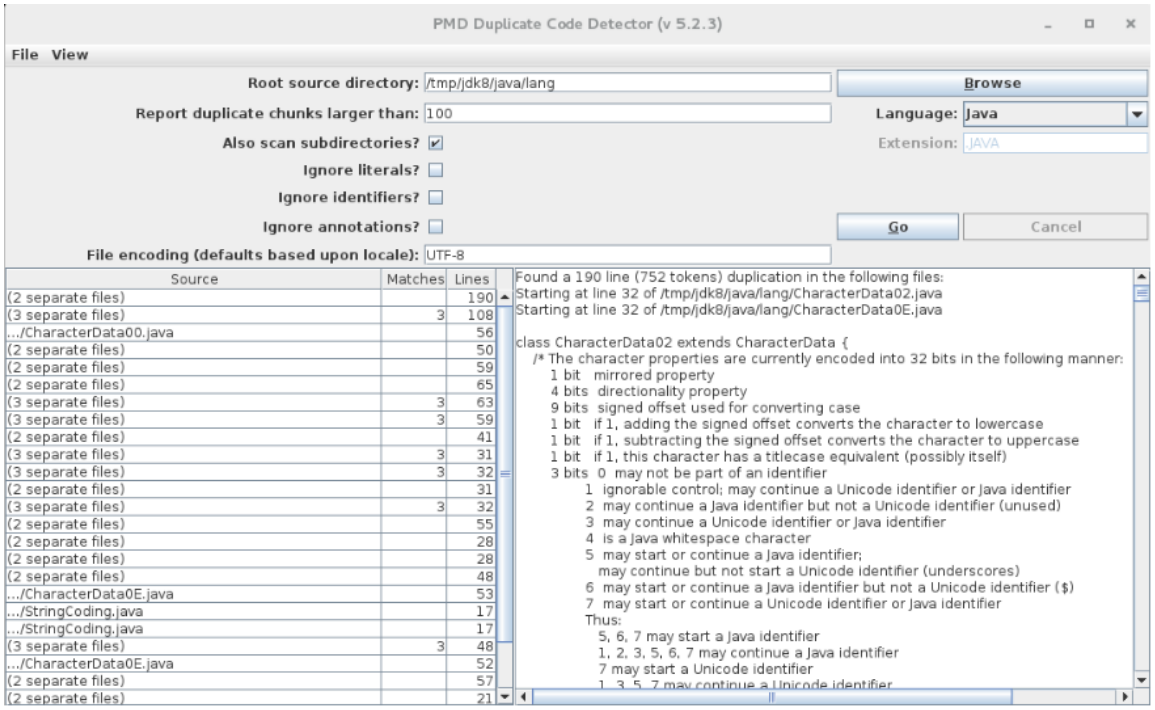
Είναι ένα ακόμη εργαλείο (PMD, 2022), το οποίο αναλύει τον κώδικα προϊόντων λογισμικού, προκειμένου να εντοπίσει λανθασμένες πρακτικές. Η γλώσσα στην οποία έχει υλοποιηθεί είναι η Java, αλλά υποστηρίζει τον εντοπισμό των κακών αυτών πρακτικών τόσο σε αυτή όσο και σε πληθώρα άλλων γλωσσών προγραμματισμού. Επίσης, μπορεί να εκτελεστεί με ποικίλους τρόπους, ανάλογα με τις ανάγκες του χρήστη, όπως για παράδειγμα μέσω του command line ή ως plugin στα περισσότερα δημοφιλή IDE.

Ο εντοπισμός των λανθασμένων πρακτικών, γίνεται μέσω προγεγραμμένων κανόνων ανάλογα με τη γλώσσα προγραμματισμού που ενδιαφέρει τον χρήστη. Οι κανόνες αυτοί, διατίθενται συγκεντρωτικά στη σελίδα του εργαλείου[3]. Ο χρήστης πρέπει να συμπεριλάβει όσους από αυτούς επιθυμεί σε ένα XML αρχείο και στη συνέχεια περνάει το τελευταίο ως παράμετρο κατά την εκτέλεση του PMD. Βέβαια, αξίζει να σημειωθεί ότι προσφέρεται η δυνατότητα δημιουργίας κανόνων και από τους ίδιους τους χρήστες, ώστε να θεσπίσουν οι ίδιοι παραβιάσεις που επιθυμούν να εντοπίσει το εργαλείο.

Από το πλήθος των κανόνων που προσφέρονται, μόνο τρεις αντιστοιχούν σε οσμές, όπως αυτές έχουν οριστεί από τον Fowler, ονομαστικά God Class, Long Method και Long Parameter List. Για τις δύο τελευταίες, χρησιμοποιούνται σταθερές τιμές εξ ορισμού, οι οποίες είναι 100 γραμμές κώδικα και 10 παράμετροι αντίστοιχα, ενώ ο χρήστης μπορεί, εφόσον το επιθυμεί, να θέσει δικές του τιμές. Αντιθέτως, για την πρώτη οσμή, χρησιμοποιεί τεχνική εντοπισμού μέσω μετρικών καθώς και συγκεκριμένες τιμές αυτών, όπως ορίζονται στο βιβλίο των (Lanza & Marinescu, 2007) για το God Class. Πιο συγκεκριμένα, υπολογίζει για κάθε κλάση το άθροισμα τις πολυπλοκότητας όλων των μεθόδων μιας κλάσης (Weighted Method Count – WMC), τον αριθμό των άμεσα συνδεδεμένων μεθόδων μιας κλάσης μέσω αναφορών σε μεταβλητές τις κλάσεις (Tight Class Cohesion – TCC) και τον αριθμό άλλων κλάσεων που έχουν μεταβλητές στις οποίες αποκτά πρόσβαση η προς έλεγχο κλάση (Access To Foreign Data – ATFD). Οι προεπιλεγμένες τιμές που χρησιμοποιούνται για κάθε μία από τις τρεις αυτές μετρικές είναι 47, 5 και 1/3 αντίστοιχα.

[3] https://pmd.github.io/pmd-6.41.0/pmd_rules_java.html

Τέλος, μέσω του PMD υποστηρίζεται και ένα ακόμη εργαλείο, το CPD (Finding duplicated code with CPD, 2022), το οποίο εντοπίζει διπλότυπα (Duplicate Code) σε πληθώρα γλωσσών προγραμματισμού. Το εργαλείο αυτό, μπορεί να εκτελεστεί με διάφορους τρόπους, ενώ κάποιοι εξ' αυτών είναι μέσω του command line ή μέσω GUI (ένα παράδειγμα αυτού φαίνεται στην Εικόνα 6). Επίσης, η μόνη τεχνική εντοπισμού διπλοτύπων που υποστηρίζει το εργαλείο είναι η ακριβής αντιγραφή, δηλαδή τα κομμάτια κώδικα που εντοπίζονται σε μια κλάση θα πρέπει να υπάρχουν αυτούσια και σε μια άλλη για να θεωρηθούν ως διπλότυπα.



Εικόνα 6: Παράδειγμα χρήσης του GUI του CPD (Finding duplicated code with CPD, 2022)

3.4 JDeodorant

Πρόκειται για ένα εργαλείο (JDeodorant, 2022), το οποίο προσεγγίζει τον εντοπισμό οσμών με λίγο διαφορετικό τρόπο συγκριτικά με τα υπόλοιπα εργαλεία της παρούσας μελέτης. Αρχικά, αξίζει να σημειωθεί πως το εν λόγω εργαλείο είναι το μοναδικό μεταξύ των υπολοίπων, το οποίο προσφέρει στους χρήστες τη δυνατότητα εφαρμογής αναδομήσεων στον κώδικα για τα διάφορα προβλήματα τα οποία εντοπίζει. Στη λογική αυτή, ενώ το εργαλείο υποστηρίζει τον εντοπισμό οσμών, παρουσιάζει τα αποτελέσματα υπό τη μορφή δυνατών αναδομήσεων για τις μεθόδους ή τις κλάσεις στις οποίες εντοπίζονται οι οσμές αυτές. Μάλιστα, οι αναδομήσεις που προτείνονται για τις

οντότητες αυτές έχουν κάποια ταξινόμηση, η λογική της οποίας διαφέρει ανάλογα με τον τύπο της οσμής.

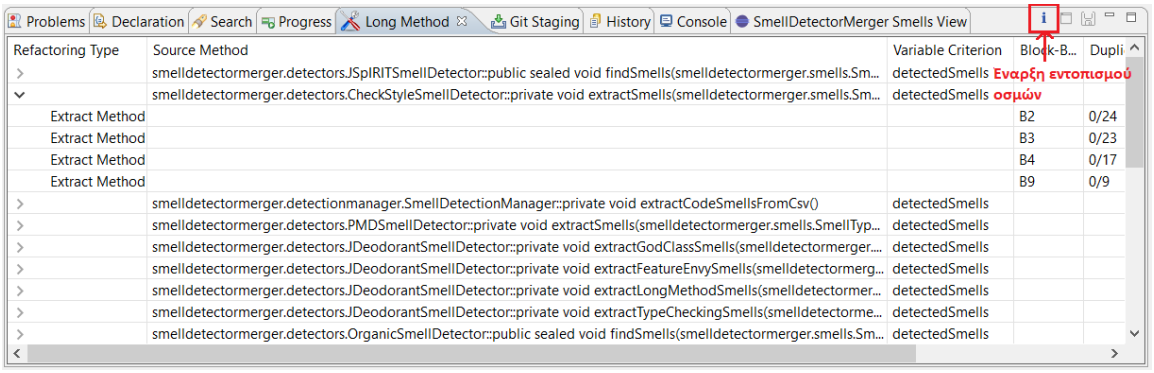
Στο σύνολο τους, υποστηρίζονται πέντε (5) οσμές, ονομαστικά God Class, Long Method, Type Checking, Feature Envy και Duplicate Code. Για τις πρώτες τέσσερις μεθόδους, οι δημιουργοί του εργαλείου ανέπτυξαν αλγορίθμους και τεχνικές εντοπισμού ενώ για την τελευταία οσμή έχουν ενσωματωθεί στο JDeodorant άλλα, επιμέρους εργαλεία, τα οποία εντοπίζουν διπλότυπα στον κώδικα και μπορεί κανείς να τα χρησιμοποιήσει μέσω του τελευταίου. Λεπτομέρειες για τους αλγορίθμους και τις τεχνικές που εφαρμόζονται στις τέσσερις πρώτες οσμές, μπορούν να βρεθούν στις παρακάτω πηγές:

- God Class: (Fokaefs, Tsantalis, Stroulia, & Chatzigeorgiou, 2011)
- Long Method: (Tsantalis & Chatzigeorgiou, 2009)
- Featue Envy: (Fokaefs, Tsantalis, & Chatzigeorgiou, 2007)
- Type Checking: (Tsantalis, Chaikalis, & Chatzigeorgiou, 2008)

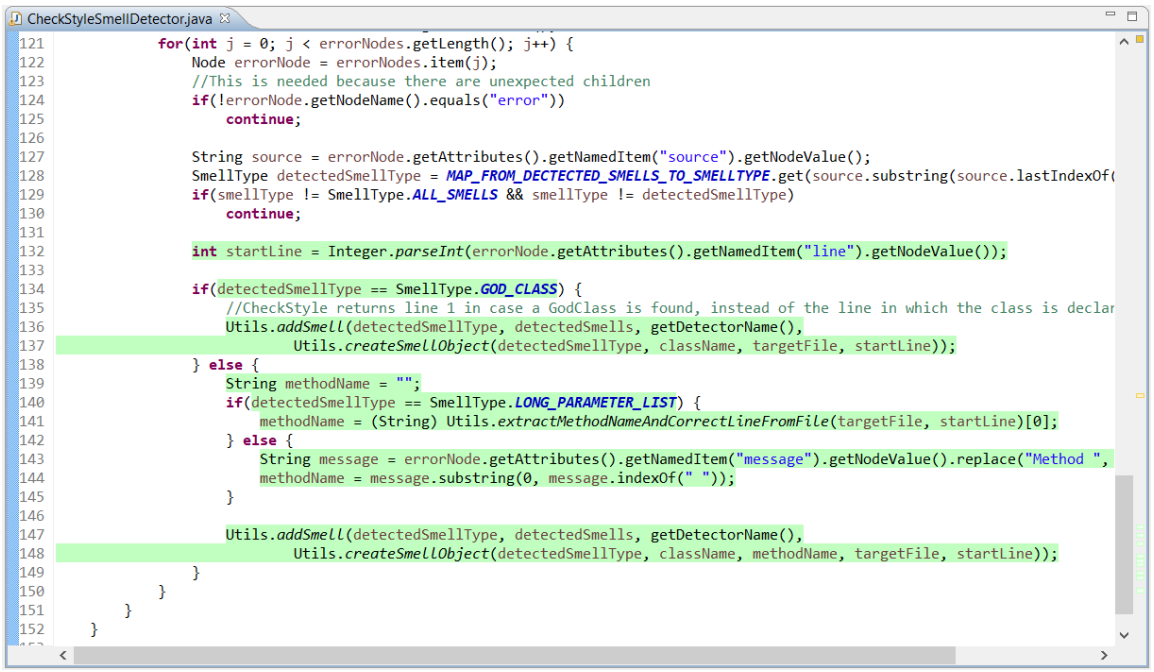
Το εργαλείο αυτό έχει αναπτυχθεί ως Eclipse plug-in, και για να το χρησιμοποιήσει ο χρήστης μπορεί, είτε να το εγκαταστήσει μέσω του Eclipse Marketplace, καθώς διατίθεται επίσημα εκεί, ακολουθώντας το μονοπάτι "*Help*" -> "*Eclipse Marketplace…*" και πληκτρολογώντας το όνομα του εργαλείου, είτε να κατεβάσει το *jar* αρχείο από το τελευταίο release και να το τοποθετήσει στον φάκελο *dropins* στον κατάλογο όπου βρίσκεται εγκατεστημένο το Eclipse. Αφού ολοκληρωθεί η διαδικασία αυτή, εμφανίζει μια καινούρια επιλογή στο μενού του Eclipse με όνομα "*Bad Smells*", μέσω της οποίας ο χρήστης μπορεί να επιλέξει την οσμή που θέλει να εντοπίσει σε κάποιο project.

Παρακάτω, παρουσιάζεται ένα παράδειγμα χρήσης του JDeodorant για την εύρεση Long Method οσμών στο project του SmellDetectorMerger, προκειμένου να γίνει ευκολότερα κατανοητός ο τρόπος λειτουργίας του. Στην Εικόνα 7 λοιπόν, φαίνεται το view με όνομα "*Long Method*" που δημιουργείται. Αρχικά είναι άδειο και ο χρήστης πρέπει να επιλέξει το project που επιθυμεί και να πατήσει στο κουμπί που έχει μαρκαριστεί με κόκκινο χρώμα στην Εικόνα 7. Έπειτα, ξεκινάει ο εντοπισμός πιθανών αναδομήσεων για οσμές τύπου Long Method που εντοπίζονται στο project και τελικά παρουσιάζονται όπως φαίνεται στην Εικόνα 7. Για κάθε ομάδα αναδομήσεων που αφορά μια συγκεκριμένη μέθοδο, φαίνεται στη στήλη "*Source Method*" η κλάση μαζί με το πακέτο στο οποίο αυτή βρίσκεται, καθώς και η δήλωση της μεθόδου μέσα σε αυτή. Εδώ

στο παράδειγμα, οι δηλώσεις δεν φαίνονται ολόκληρες για λόγους απλότητας. Ακόμη, επεκτείνοντας την κάθε ομάδα, βλέπουμε όλες τις αναδομήσεις που έχουν προταθεί. Κάνοντας κλικ σε κάποια από αυτές, ανοίγει στο Eclipse το αντίστοιχο αρχείο που περιέχει τη συγκεκριμένη μέθοδο. Μέσα στη μέθοδο αυτή, υπάρχουν μαρκαρισμένες με πράσινο χρώμα οι γραμμές, στις οποίες θα εφαρμοστεί αναδόμηση τύπου "*Extract Method*" (στη συγκεκριμένη περίπτωση). Ένα παράδειγμα του τελευταίου φαίνεται στην Εικόνα 8. Τέλος, εάν ο χρήστης επιθυμεί να εφαρμόσει τη συγκεκριμένη αναδόμηση, πρέπει να πατήσει στο κουμπί που υπάρχει δίπλα σε αυτό που εκκινεί τον εντοπισμό οσμών (βλ. Εικόνα 7) και ο κώδικας τροποποιείται αναλόγως, εφαρμόζοντας τις αλλαγές που επιλέχθηκαν. Παράδειγμα αυτού φαίνεται στην Εικόνα 9, όπου έχει εφαρμοστεί η τελευταία "*Extract Method*" αναδόμηση που φαίνεται στην Εικόνα 7 για τη δεύτερη μέθοδο, δηλαδή *CheckStyleSmellDetector::extractSmells*.



Εικόνα 7: Παράδειγμα χρήσης JDeodorant για τον εντοπισμό οσμών Long Method



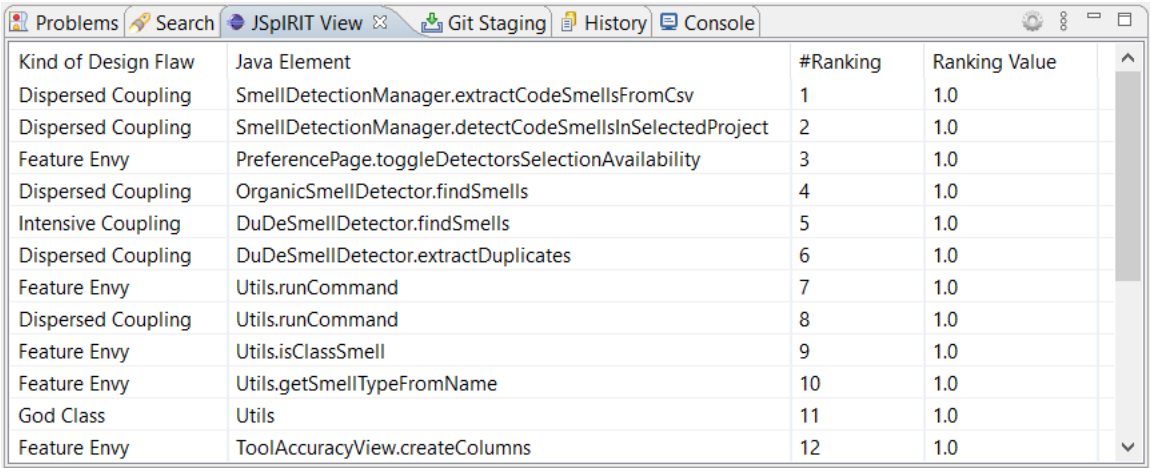

```
121             for(int j = 0; j < errorNodes.getLength(); j++) {
122                 Node errorNode = errorNodes.item(j);
123                 //This is needed because there are unexpected children
124                 if(!errorNode.getNodeName().equals("error"))
125                     continue;
126
127                 String source = errorNode.getAttributes().getNamedItem("source").getNodeValue();
128                 SmellType detectedSmellType = MAP_FROM_DECTECTED_SMELLS_TO_SMELLTYPE.get(source.substring(source.lastIndexOf(
129                 if(smellType != SmellType.ALL_SMELLS && smellType != detectedSmellType)
130                     continue;
131
132                 int startLine = Integer.parseInt(errorNode.getAttributes().getNamedItem("line").getNodeValue());
133
134                 if(detectedSmellType == SmellType.GOD_CLASS) {
135                     //CheckStyle returns line 1 in case a GodClass is found, instead of the line in which the class is declar
136                     Utils.addSmell(detectedSmellType, detectedSmells, getDetectorName(),
137                             Utils.createSmellObject(detectedSmellType, className, targetFile, startLine));
138                 } else {
139                     String methodName = "";
140                     if(detectedSmellType == SmellType.LONG_PARAMETER_LIST) {
141                         methodName = (String) Utils.extractMethodNameAndCorrectLineFromFile(targetFile, startLine)[0];
142                     } else {
143                         String message = errorNode.getAttributes().getNamedItem("message").getNodeValue().replace("Method ",
144                         methodName = message.substring(0, message.indexOf(" "));
145                     }
146
147                     Utils.addSmell(detectedSmellType, detectedSmells, getDetectorName(),
148                             Utils.createSmellObject(detectedSmellType, className, methodName, targetFile, startLine));
149                 }
150             }
151         }
152     }
```

Εικόνα 8: Προβολή προτεινόμενης αναδόμησης για Long Method

Εικόνα 9: Εφαρμογή επιλεγμένης αναδόμησης στην περίπτωση του Long Method

3.5 JSpIRIT

Το εργαλείο αυτό (Vidal, και συν., 2015) έχει αναπτυχθεί ως ένα Eclipse plug-in. Ο χρήστης μπορεί να το χρησιμοποιήσει σε Java projects τα οποία έχει κάνει import στο Eclipse, πατώντας δεξί κλικ στο project που επιθυμεί και στη συνέχεια "*JSpIRIT*" -> "*Find Code Smells*". Μετά τα προηγούμενα βήματα, το εργαλείο θα ελέγξει τον κώδικα του project και στη συνέχεια θα ανοίξει ένα καινούριο view με όνομα "*JSpIRIT View*", όπου θα εμφανίσει, σε μορφή λίστας, όλες τις οσμές που εντοπίστηκαν. Ένα παράδειγμα χρήσης του εργαλείου στο project του SmellDetectorMerger, φαίνεται παρακάτω στην Εικόνα 10, όπου παρουσιάζεται μέρος τον οσμών που εντοπίστηκαν.



Kind of Design Flaw	Java Element	#Ranking	Ranking Value
Dispersed Coupling	SmellDetectionManager.extractCodeSmellsFromCsv	1	1.0
Dispersed Coupling	SmellDetectionManager.detectCodeSmellsInSelectedProject	2	1.0
Feature Envy	PreferencePage.toggleDetectorsSelectionAvailability	3	1.0
Dispersed Coupling	OrganicSmellDetector.findSmells	4	1.0
Intensive Coupling	DuDeSmellDetector.findSmells	5	1.0
Dispersed Coupling	DuDeSmellDetector.extractDuplicates	6	1.0
Feature Envy	Utils.runCommand	7	1.0
Dispersed Coupling	Utils.runCommand	8	1.0
Feature Envy	Utils.isClassSmell	9	1.0
Feature Envy	Utils.getSmellTypeFromName	10	1.0
God Class	Utils	11	1.0
Feature Envy	ToolAccuracyView.createColumns	12	1.0

Εικόνα 10: Παράδειγμα χρήσης του JSpIRIT στο SmellDetectorMerger

Το JSpIRIT υποστηρίζει εξ ορισμού τον εντοπισμό δέκα (10) οσμών, οι οποίες παρουσιάζονται στον Πίνακα 2, χρησιμοποιώντας την τεχνική των μετρικών όπως αυτή έχει προταθεί από τους (Lanza & Marinescu, 2007). Βέβαια, δίνεται και η δυνατότητα στους χρήστες να τροποποιήσουν, εφόσον το επιθυμούν, τις τιμές των αρχικών μετρικών ούτως ώστε να παραμετροποιήσουν σε κάποιο βαθμό τη διαδικασία. Επίσης, εκτός αυτού, το εργαλείο επιτρέπει στους χρήστες να ορίσουν μέσω του κώδικα τις δικές τους οσμές και τεχνικές εντοπισμού των τελευταίων, με αποτέλεσμα να μπορεί να επεκταθεί η λειτουργία ακόμη περισσότερο ανάλογα με τις υπάρχουσες ανάγκες.

Μια ακόμη δυνατότητα που προσφέρεται εδώ, είναι η ταξινόμηση των οσμών που εντοπίστηκαν με βάση το πόσο σημαντικές θεωρούνται και πόσο άμεση ενδεχομένως κρίνεται η επιδιόρθωση τους. Όπως φαίνεται στην Εικόνα 10 στην τελευταία στήλη, εξ ορισμού η τιμή που υπάρχει παντού είναι 1.0, εφόσον δεν έχει γίνει κάποια αλλαγή από τον χρήστη. Μετά την παρουσίαση των αποτελεσμάτων, ο χρήστης μπορεί μέσω του view αυτού να επιλέξει με ποιο/α κριτήριο/α επιθυμεί να ταξινομηθούν οι οσμές και να εισάγει τις απαιτούμενες με βάση τις ανάγκες του τιμές σε αυτά ώστε τελικά ο πίνακας να ταξινομηθεί με βάση τις νέες τιμές που προκύπτουν. Επειδή οι επιλογές ως προς τα κριτήρια και τις τιμές αυτών είναι πολλές, προτείνεται η χρήση του σχετικού άρθρου ως αναφορά, καθώς και ο πειραματισμός με το ίδιο το εργαλείο.

Τέλος, αξίζει να σημειωθεί πως η ύπαρξη ταξινόμησης στις οσμές ως προς τη σπουδαιότητα τους είναι ιδιαίτερα σημαντική, διότι με τον τρόπο αυτό μπορεί να δοθεί προτεραιότητα στον εντοπισμό και στην επιδιόρθωση των οσμών που επιβαρύνουν σε μεγαλύτερο βαθμό το σύστημα. Αποτέλεσμα αυτού είναι να καθίσταται ευκολότερη η συντήρηση των κλάσεων που τις περιέχουν, καθώς και η προσθήκη νέων λειτουργιών σε αυτές.

3.6 Organic

Είναι ένα ακόμη εργαλείο (Organic, 2022), το οποίο έχει υλοποιηθεί ως plug-in για το Eclipse. Η διαφορά του συγκεκριμένου με τα προηγούμενα εργαλεία, είναι ότι εκτελείται μέσω του command line και δεν απαιτεί αλληλεπίδραση με τον χρήστη. Για την εκτέλεση του λοιπόν, πρέπει ο χρήστης να εκτελέσει το jar αρχείο του plug-in περνώντας κάποιες απαραίτητες παραμέτρους, όπως φαίνεται στην Εικόνα 11, και αφού ολοκληρωθεί ο έλεγχος του ζητούμενου project, τα αποτελέσματα εκτυπώνονται στο αρχείο που έχει οριστεί σε μορφή JSON.

```
ECLIPSE_PATH="/path/to/eclipse/installation"
EQUINOX="${ECLIPSE_PATH}/plugins/org.eclipse.equinox.launcher_<version>.jar org.eclipse.core.launcher.Main"
SMELL_DETECTOR="smell-detector-plugin.SmellDetector"
java -jar -XX:MaxPermSize=2560m -Xms40m -Xmx2500m ${EQUINOX} -application ${SMELL_DETECTOR} -sf "smells.json"
-src "/path/to/source/folder/"
```

Εικόνα 11: Παράδειγμα παραμέτρων εκτέλεσης του Organic plug-in (Organic, 2022)

Κατά τη δημιουργία του, το plug-in υποστήριζε τον εντοπισμό έντεκα (11) οσμών συνολικά, ενώ με βάση τον τρέχοντα κώδικα του, υποστηρίζονται συνολικά δεκαεπτά (17) οσμές. Όλες οι υποστηριζόμενες οσμές του εργαλείου βρίσκονται συγκεντρωτικά στον Πίνακα 2. Μάλιστα, αξίζει να σημειωθεί πως πρόκειται για το εργαλείο που ενσωματώνει τον έλεγχο των περισσότερων οσμών συγκριτικά με όσα εργαλεία μελετήθηκαν στα πλαίσια της παρούσας εργασίας. Για τον εντοπισμό των οσμών, το Organic χρησιμοποιεί τεχνικές βασισμένες σε μετρικές, οι οποίες έχουν προταθεί από τους (Bavota, De Lucia, Di Penta, Oliveto, & Palomba, 2015). Οι τελευταίοι μάλιστα στα πειράματα τους εξέτασαν τρία open source projects, ονομαστικά Apache Ant, ArgoUML και Xerces-J, για την εύρεση έντεκα οσμών, τα οποία αποτέλεσαν και τις πρώτες έντεκα οσμές που υποστήριζε αρχικά το Organic.

Πίνακας 2: Υποστηριζόμενες οσμές ανά εργαλείο

Tools	Code Smells
CheckStyle	God Class, Long Method, Long Parameter List
DuDe	Duplicate Code
PMD	God Class, Long Method, Long Parameter List, Duplicate Code
JDeodorant	God Class, Long Method, Feature Envy, Type Checking
JSpIRIT	God Class, Long Method, Feature Envy, Brain Class, Brain Method, Data Class, Dispersed Coupling, Intensive Coupling, Refused Parent Bequest, Shotgun Surgery, Tradition Breaker
Organic	God Class, Long Method, Long Parameter List, Feature Envy, Brain Class, Brain Method, Data Class, Dispersed Coupling, Intensive Coupling, Refused Parent Bequest, Shotgun Surgery, Class Data Should Be Private, Complex Class, Lazy Class, Message Chain, Speculative Generality, Spaghetti Code

4 Μεθοδολογία

4.1 Δομή μελέτης

4.1.1 Εύρεση εργαλείων εντοπισμού οσμών

Πρωταρχικό και απαραίτητο βήμα για την δημιουργία του σχετικού plug-in, ήταν η αναζήτηση των διαθέσιμων πηγών στη βιβλιογραφία, προκειμένου να εντοπιστούν αναφορές σε υπάρχοντα εργαλεία και να μελετηθούν. Κύριο σημείο για την εύρεση πηγών αποτέλεσε το Google Scholar. Για την επιστροφή σχετικών αποτελεσμάτων, χρησιμοποιήθηκαν λέξεις-κλειδιά όπως "*code smell detectors*", "*code smell detection*" και "*code smell detection tools*". Τα αποτελέσματα τα οποία επιστράφηκαν με τη χρήση των προηγούμενων λέξεων-κλειδιών, περιλάμβαναν άρθρα τόσο από συγγραφείς οι οποίοι πρότειναν κάποιο συγκεκριμένο εργαλείο και πραγματοποιούσαν μια μελέτη περί αυτού, όσο και μελέτες οι οποίες παρουσίαζαν αποτελέσματα μιας βιβλιογραφικής ανασκόπησης σχετικά με εργαλεία εντοπισμού οσμών, καθώς και αποτελέσματα, τάσεις, ομοιότητες και πειράματα με χρήση μιας επιλογής ορισμένων εξ αυτών πάνω σε έργα λογισμικού. Τα αποτελέσματα αυτά έδειξαν ότι από την εισαγωγή της έννοιας των οσμών, ερευνητές και όχι μόνο έδειξαν πολύ μεγάλο ενδιαφέρον γι' αυτές και την σημασία τους στα προϊόντα λογισμικού. Για το λόγο αυτό, οι προτάσεις οι οποίες υπήρξαν ήταν πολλές. Οι (Fernandes, Oliveira, Vale, Paiva, & Figueiredo, 2016), πραγματοποίησαν μια βιβλιογραφική επισκόπηση, η οποία οδήγησε στην εύρεση 84 εργαλείων εντοπισμού κώδικα, η συντριπτική πλειοψηφία των οποίων χρονολογείται από το 1990 και μετά, που αποτελεί και την χρονολογία που εμφανίστηκε για πρώτη φορά η έννοια των οσμών, καθώς και στην εξαγωγή κάποιων σημαντικών στοιχείων. Από το σύνολο των εργαλείων, μόνο 29 ήταν διαθέσιμα για λήψη και εγκατάσταση ενώ τα υπόλοιπα όχι. Επίσης, τα αποτελέσματα έδειξαν μία τάση των προγραμματιστών για ανάπτυξη των εργαλείων σε γλώσσα Java, καθώς η πλειοψηφία αυτών, συγκεκριμένα 56 εργαλεία, έχουν γραφεί με χρήση της τελευταίας, ενώ τα υπόλοιπα αξιοποίησαν πληθώρα άλλων γλωσσών, με ένα μάλιστα να κάνει χρήση του MATLAB[4]. Αντίστοιχα με τη συγγραφή, ιδιαίτερα δημοφιλής ήταν η Java και για την ανάλυση προγραμμάτων, καθώς από το σύνολο των εργαλείων, τα 40 είναι ικανά να αναλύσουν αυτά τα οποία είναι γραμμένα σε Java. Μια ακόμη τάση παρατηρείται και στις οσμές τις οποίες

[4] https://www.mathworks.com/products/matlab.html

εντοπίζουν τα εργαλεία αυτά. Σύμφωνα με τη μελέτη, ιδιαίτερα δημοφιλής είναι οι οσμές *Duplicate Code* και *Large Class (God Class)*, οι οποίες εντοπίζονται σε ποσοστό περίπου 44% και 40% αντίστοιχα, ενώ ακολουθούν το *Long Method* και *Feature Envy* με ποσοστό περίπου 23% και 15% αντίστοιχα.

Στην παρούσα εργασία, σκοπός ήταν η εύρεση, μελέτη και αξιοποίηση εργαλείων τα οποία έχουν γραφτεί σε Java και μπορούν να αναλύσουν προγράμματα σε Java, επομένως τα εργαλεία τα οποία είναι γραμμένα σε άλλη γλώσσα ή υποστηρίζουν την ανάλυση άλλης γλώσσας δεν λήφθηκαν υπόψιν. Τα εργαλεία τα οποία πληρούν τα προηγούμενα κριτήρια, αποτελούν τόσο αποτέλεσμα ερευνητών όσο και ανεξάρτητων προγραμματιστών ή ομάδων. Αξίζει να σημειωθεί επίσης ότι η πλειοψηφία αυτών των εργαλείων είναι plug-ins τα οποία έχουν αναπτυχθεί για το Eclipse.

Στα πλαίσια της μελέτης αυτής λοιπόν, εντοπίστηκαν συνολικά δεκαεννιά (19) εργαλεία εντοπισμού οσμών. Από το σύνολο αυτό, μόνο τα έξι (6) μπόρεσαν να αξιοποιηθούν τελικά και να προστεθούν στη λειτουργία του plug-in το οποίο δημιουργήθηκε με σκοπό να ενσωματώσει υπάρχοντα εργαλεία κάτω από μια κοινή ομπρέλα. Αιτία αυτού υπήρξε κατά κύριο λόγο η έλλειψη διαθεσιμότητας των εργαλείων αυτών. Τα περισσότερα δεν ήταν άμεσα διαθέσιμα για λήψη, ούτε με τη μορφή κάποιου εκτελέσιμου αρχείου, ούτε με τη μορφή πηγαίου κώδικα. Επίσης, παρά τις προσπάθειες επικοινωνίας με τους δημιουργούς του εκάστοτε εργαλείου με σκοπό την απόκτηση του κώδικα μέσω αυτών, τα αποτελέσματα δεν ήταν θετικά. Στο μεγαλύτερο ποσοστό αυτών, ο(ι) ερευνητής(ές) δεν είχε(αν) πλέον στην κατοχή του(ς) τον κώδικα, ήταν παρωχημένος ή δεν υπήρξε καμία απόκριση από αυτόν(ους). Υπήρξε επίσης μια περίπτωση (Nongpong, 2012), στην οποία το εργαλείο χρησιμοποιούσε κάποια εμπορική βιβλιοθήκη σε ένα τμήμα του, επομένως δεν μπορούσε να χρησιμοποιηθεί διότι το plug-in που παράχθηκε είναι ανοιχτού κώδικα αποτελούμενο από ελεύθερα εργαλεία. Ακόμη, σε κάποιες περιπτώσεις (Murphy-Hill & P. Black, 2010), ενώ το εργαλείο ήταν διαθέσιμο, δεν μπόρεσε να αξιοποιηθεί επειδή το κύριο στοιχείο του είναι η οπτική αναπαράσταση των οσμών που εντοπίζονται, ενώ στο εργαλείο που αναπτύχθηκε, οι οσμές που εντοπίζονται παρατίθενται ονομαστικά με μορφή λίστας.

Παρακάτω, στον Πίνακα 3, φαίνονται όλα τα εργαλεία τα οποία εντοπίστηκαν και είναι χωρισμένα σε δύο κατηγορίες, ανάλογα με το αν μπόρεσαν να ενσωματωθούν στο εργαλείο που αναπτύχθηκε ή όχι.

Πίνακας 3: Εργαλεία που εντοπίστηκαν κατά τη διάρκεια της μελέτης

Εργαλεία που ενσωματώθηκαν	Εργαλεία που δεν ενσωματώθηκαν
CheckStyle (CheckStyle, 2022) **DuDe** (Wettel & Marinescu, 2005) **PMD** (PMD, 2022) **JDeodorant** (JDeodorant, 2022) **JSpIRIT** (Vidal, και συν., 2015) **Organic** (Organic, 2022)	**InsRefactor, HIST (Historical Information for Smell detection)** (Palomba, και συν., 2013) **TACO (Textual Analysis for Code smell detection)** (Palomba, Panichella, De Lucia, Oliveto, & Zaidman, 2016) **ConcernReCS** (Alves, Santana, & Figueiredo, 2012) **TrueRefactor** (Griffith, Wahl, & Izurieta, 2011) **CBSDetector** (Hall, Zhang, Bowes, & Sun, 2014) **Stench Blossom** (Murphy-Hill & P. Black, 2010) **JSD (Java Smell Detector)** (Mathur, 2011) **JCodeCanine** (Nongpong, 2012) **SISSy** (SISSy, 2022) **No-name tool** (Kirk, Roper, & Wood, 2007) **BSDT (Bad Smell Detecting Tool)** (Danphitsanuphan & Suwantada, 2012) **JSmell** (Roperia, 2009)

4.1.2 Ανάλυση εργαλείων εντοπισμού οσμών

Μετά τον εντοπισμό και την συγκέντρωση εργαλείων εντοπισμού οσμών τα οποία μπορούν να αξιοποιηθούν, το επόμενο βήμα είναι η ανάλυση τους προκειμένου να γίνει κατανοητός ο τρόπος λειτουργίας τους ώστε να ενσωματωθούν μετά στο ενιαίο εργαλείο, το SmellDetectorMerger, που αποτελεί προϊόν της παρούσας έρευνας. Τα εργαλεία που εντοπίστηκαν, είναι χωρισμένα σε δύο κατηγορίες. Τα μισά εξ αυτών διαθέτουν εκτελέσιμα αρχεία, τα οποία καλούνται μέσω της γραμμής εντολών και είτε εκτυπώνουν τα αποτελέσματα στην τελευταία, είτε δημιουργούν ένα καινούριο αρχείο και τα αποθηκεύουν σε αυτό. Τα υπόλοιπα, είναι plug-ins, τα οποία έχουν αναπτυχθεί με σκοπό τη χρήσης τους στο Eclipse IDE. Για τα λόγο αυτό, εφόσον ο χρήστης εκτελέσει το plug-in, τα αποτελέσματα του εντοπισμού εμφανίζονται μέσα στο ίδιο το Eclipse, σε ένα καινούριο παράθυρο το οποίο δημιουργείται από το IDE.

Όπως προαναφέρθηκε, τα εργαλεία τα οποία εκτελούνται μέσω της γραμμής εντολών είναι το CheckStyle, το DuDe και το PMD. Το πρώτο εξ αυτών, απαιτεί τον ορισμό ενός XML αρχείου, το οποίο περιλαμβάνει τους κανόνες με βάση τους οποίους θα ελεγχθούν τα project για να εντοπιστούν συγκεκριμένες οσμές, όπως αναφέρθηκε στην Ενότητα 3.1. Στην Εικόνα 12, φαίνεται το αρχείο που χρησιμοποιήθηκε στο SmellDetectorMerger για το CheckStyle. Όπως παρουσιάζεται στην Εικόνα 12, έχουν οριστεί τρεις κανόνες για τον εντοπισμό τριών οσμών, ονομαστικά των God Class, Long Method και Long Parameter List. Πιο συγκεκριμένα, οι τρεις αυτές οσμές ορίζονται στο αρχείο στις γραμμές 13, 19 και 23 αντίστοιχα. Το αρχείο αυτό περνάει ως όρισμα κατά την εκτέλεση του εργαλείου μέσω της γραμμής εντολών και αποτελεί απαραίτητη παράμετρο.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE module PUBLIC
3            "-//Checkstyle//DTD Checkstyle Configuration 1.3//EN"
4            "https://checkstyle.org/dtds/configuration_1_3.dtd">
5
6  <module name="Checker">
7
8      <!-- Set to only check for java files in a project -->
9      <property name="fileExtensions" value="java"/>
10
11     <!-- The check for large classes -->
12     <!-- https://checkstyle.sourceforge.io/apidocs/com/puppycrawl/tools/checkstyle/checks/sizes/FileLengthCheck.html -->
13     <module name="FileLength"/>
14
15     <module name="TreeWalker">
16
17         <!-- The check for long method -->
18         <!-- https://checkstyle.sourceforge.io/apidocs/com/puppycrawl/tools/checkstyle/checks/sizes/MethodLengthCheck.html -->
19         <module name="MethodLength"/>
20
21         <!-- The check for long parameter list -->
22         <!-- https://checkstyle.sourceforge.io/apidocs/com/puppycrawl/tools/checkstyle/checks/sizes/ParameterNumberCheck.html -->
23         <module name="ParameterNumber"/>
24
25     </module>
26
27 </module>
```

Εικόνα 12: Αρχείο ορισμού κανόνων του CheckStyle

Για το επόμενο εργαλείο, το DuDe, απαιτείται ως παράμετρος ένα απλό αρχείο κειμένου, το οποίο πρέπει να περιέχει το μονοπάτι για το project το οποίο πρόκειται να αναλυθεί. Επίσης, συμβουλή του αρθρογράφου είναι η προσθήκη παραμέτρων για την Java στην περίπτωση που το project που θα αναλυθεί είναι μεγάλο, ώστε να δοθούν περισσότεροι πόροι για να επιτευχθεί ο έλεγχος. Συγκεκριμένα, προτείνεται η χρήση των εξής δύο παραμέτρων: -Xms256m -Xmx1024m. Αξίζει να σημειωθεί επίσης ότι, όπως αναφέρεται και στην Ενότητα 3.2, το εργαλείο υποστηρίζει και γραφικό περιβάλλον για τη χρήση του, αλλά στη συγκεκριμένη περίπτωση η εκτέλεση του μέσω της γραμμής εντολών ήταν ιδανική για την ενσωμάτωση του.

Το τελευταίο εργαλείο το οποίο εκτελείται από τη γραμμή εντολών, το PMD, απαιτεί και αυτό την προσθήκη ενός XML αρχείου ως παράμερο. Όπως έχει ήδη

αναφερθεί στην Ενότητα 3.3, το αρχείο αυτό περιλαμβάνει τον ορισμό των κανόνων που θα χρησιμοποιηθούν για τον εντοπισμό οσμών. Στην Εικόνα 13, φαίνεται το αρχείο το οποίο χρησιμοποιείται από το SmellDetectorMerger. Οι οσμές που εντοπίζονται εδώ είναι οι God Class, Long Method και Long Parameter List και οι γραμμές στις οποίες ορίζονται οι κανόνες είναι οι 9, 10 και 11 αντίστοιχα. Μια ακόμη παράμετρος η οποία μπορεί να προστεθεί στο PMD, είναι το μονοπάτι για ένα απλό αρχείο κειμένου, το οποίο χρησιμοποιείται ως κρυφή μνήμη (cache). Όταν εκτελείται το εργαλείο σε κάποιο project, αποθηκεύονται στο αρχείο αυτό τόσο κάποιες λεπτομέρειες σχετικά με το project το οποίο αναλύθηκε, όσο και τα αποτελέσματα της αναζήτησης. Στόχος αυτού, είναι να μην πραγματοποιείται εκ νέου η αναζήτηση εφόσον το project έχει παραμείνει ίδιο και να επιστρέφονται αυτόματα τα αποτελέσματα της προηγούμενης αναζήτησης. Έτσι, τα αποτελέσματα που είχαν βρεθεί την πρώτη φορά επιστρέφονται αμέσως και εξοικονομούνται πόροι και χρόνος καθώς η διαδικασία εντοπισμού δεν χρειάζεται να εκτελεστεί από την αρχή.

```
1  <?xml version="1.0"?>
2  <ruleset name="MyRuleSet"
3      xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0 https://pmd.sourceforge.io/ruleset_2_0_0.xsd">
6
7      <description>A custom rule set, created to find 3 well known bad smells. GodClass, LongMethod, LongParameterList</description>
8
9      <rule ref="category/java/design.xml/GodClass" />
10     <rule ref="category/java/design.xml/ExcessiveMethodLength" />
11     <rule ref="category/java/design.xml/ExcessiveParameterList" />
12
13 </ruleset>
```

Εικόνα 13: Αρχείο ορισμού κανόνων του PMD

Τα υπόλοιπα τρία εργαλεία τα οποία συμπεριλήφθηκαν στο SmellDetectorMerger, είναι, όπως έχει αναφερθεί, plug-ins του Eclipse. Προκειμένου να μπορέσουν να ενσωματωθούν τα εργαλεία αυτά στο ενιαίο plug-in, χρειάστηκε η εισαγωγή του πηγαίου τους κώδικα στο Eclipse, και στη συνέχεια η εξαγωγή τους, μέσω του προηγούμενου με τη μορφή *jar* αρχείου (βλ. Ενότητα 2.2.4), ώστε να εισαχθούν έπειτα στο SmellDetectorMerger ως εξωτερικές βιβλιοθήκες. Στο πρώτο εργαλείο, το JDeodorant, χρειάστηκε να γίνουν κάποιες πολύ μικρές αλλαγές στον κώδικα του πριν την εξαγωγή του σε *jar*, προκειμένου να μπορέσει να ενσωματωθεί χωρίς προβλήματα. Καμία από τις αλλαγές που έγιναν δεν τροποποίησε τη λογική του εργαλείου ως προς τον εντοπισμό οσμών, παρά μόνο έδωσε πρόσβαση σε αναγκαία σημεία του κώδικα και έθεσε σταθερές σε κάποια σημεία αντί κάποιων μεταβλητών που υπήρχαν, οι οποίες αρχικά λάμβαναν την τιμή τους μέσω μιας σελίδας προτιμήσεων που διαθέτει το εργαλείο στο μονοπάτι “*Windows*” -> “*Preferences*”. Οι σταθερές τιμές που

χρησιμοποιήθηκαν, ήταν αυτές οι οποίες υπήρχαν εξ ορισμού στη σελίδα προτιμήσεων του εργαλείου με την εγκατάσταση του. Στα άλλα δύο εργαλεία, ονομαστικά τα JSpIRIT και Organic, δεν χρειάστηκε να γίνει κάποια αλλαγή στον κώδικα τους, διότι τα απαραίτητα τμήματα του κώδικα ήταν ήδη ορατά και επίσης δεν διαθέτουν κάποια σελίδα προτιμήσεων που να προσφέρει συγκεκριμένη παραμετροποίηση.

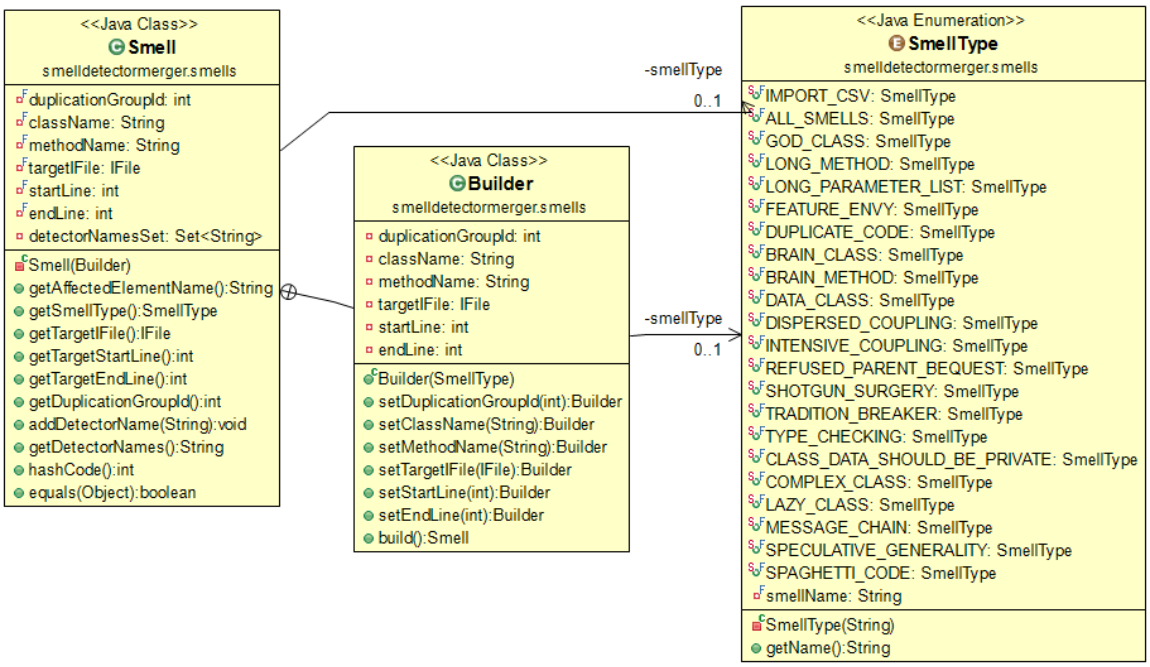
4.2 Σχεδίαση και υλοποίηση του SmellDetectorMerger

Μετά από την εύρεση και την ανάλυση των εργαλείων εντοπισμού οσμών, σειρά έχει η σχεδίαση και υλοποίηση του SmellDetectorMerger, ενός plug-in που αποτελεί ομπρέλα για υπάρχοντα εργαλεία με σκοπό την εύκολη πρόσβαση σε αυτά και την αυτοματοποιημένη χρήση τους για την εξαγωγή συγκεντρωτικών αποτελεσμάτων. Η ανάπτυξη του plug-in πραγματοποιήθηκε σε Java 8, ενώ δοκιμάστηκε στο Eclipse και συγκεκριμένα στην έκδοση 2021-06 (4.20). Στις υποενότητες που ακολουθούν, παρουσιάζονται διαφορετικά τμήματα της σχεδίασης που αντιστοιχούν και σε διαφορετικά κομμάτια του plug-in. Για την ευκολότερη κατανόηση των διάφορων κλάσεων που έχουν δημιουργηθεί, των σχέσεων που υπάρχουν μεταξύ τους, καθώς και των μεθόδων που υλοποιούν οι τελευταίες, έχουν προστεθεί αντίστοιχα διαγράμματα UML, στα οποία αναπαριστώνται τα τμήματα κώδικα που συζητώνται στην αντίστοιχη ενότητα.

4.2.1 Βασικός κορμός

Μία από τις βασικές απαιτήσεις υπήρξε ο ορισμός, με κάποιο τρόπο, των οσμών. Στην Εικόνα 14, απεικονίζονται τα στοιχεία τα οποία σχετίζονται με τον ορισμό των οσμών. Λόγω του ότι οι οσμές γενικά έχουν διάφορα κοινά μεταξύ τους χαρακτηριστικά, όπως για παράδειγμα το όνομα της κλάσης ή της μεθόδου (αν η οσμή αφορά μεθόδους), η γραμμή στην οποία συναντάται η οσμή κ.τ.λ., αποφασίστηκε να χρησιμοποιηθεί μια κλάση με όνομα *Smell*, η οποία θα περιλαμβάνει όλες τις πληροφορίες που μπορούν να συναντηθούν σε μία οσμή. Στην ίδια κλάση, υπάρχουν και διάφορες μέθοδοι (getters) οι οποίες παρέχουν πρόσβαση στις διάφορες αυτές πληροφορίες. Για τον διαχωρισμό μεταξύ των διαφόρων τύπων οσμών, δημιουργήθηκε ένα enum. Σε αυτό, υπάρχουν δηλωμένοι όλοι οι τύποι οσμών οι οποίοι υποστηρίζονται, κατά τη συγγραφή αυτής της εργασίας από το SmellDetectorMerger. Πέραν αυτών, υπάρχουν δηλωμένοι και άλλοι δύο τύποι, ονομαστικά *IMPORT_CSV* και *ALL_SMELLS*, για λόγους ευκολίας, όπως θα εξηγηθεί σε λίγο. Μέσα στο enum, έχει δηλωθεί επίσης και ένας constructor, ώστε κάθε

type μέσα στο enum να χρησιμοποιεί και ένα αλφαριθμητικό το οποίο θα προσδιορίζει το όνομα του. Αυτό, χρησιμοποιείται για την αναγνώριση του τύπου των οσμών, είτε όταν αυτές επιλέγονται από τον χρήστη κατά την εκκίνηση του plug-in, είτε για την εκτύπωση της εκάστοτε οσμής με βάση το enum type το οποίο της αντιστοιχεί. Σχετικά με τα δύο enum types που δεν αποτελούν ονόματα οσμών, έχουν προστεθεί ώστε να αναγνωρίζεται πότε ο χρήστης επιλέγει στο plug-in να εντοπισθούν όλοι οι τύποι οσμών ή να εισαχθούν οσμές που έχουν ήδη εντοπισθεί μέσω ενός αρχείου *csv*. Ο εκάστοτε τύπος οσμής που έχει επιλεγεί από τον χρήστη, αναγνωρίζεται από τη *SmellDetectionManager* (βλ. ενότητα 4.2.2) και ανάλογα εκτελείται η αντίστοιχη λογική στον κώδικα.

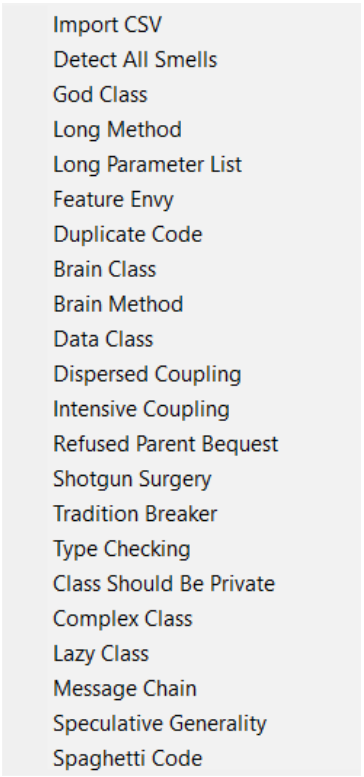


Εικόνα 14: Διάγραμμα UML για τις κλάσεις δημιουργίας οσμών

Για την κατασκευή της εκάστοτε οσμής μέσω της κλάσης *Smell*, αποφασίστηκε η χρήση του Builder pattern (Gamma, Helm, Johnson, & Vlissides, 1995) ώστε να υπάρχει ένας εύκολος τρόπος κατασκευής των διαφόρων τύπων οσμών. Όπως φαίνεται και στην Εικόνα 14, υπάρχει η κλάση *Builder* η οποία είναι εσωτερική στην κλάση *Smell*. Για τη δημιουργία λοιπόν αντικειμένων της κλάσης, δημιουργείται αρχικά ένα καινούριο αντικείμενο τύπου *Builder* που το μόνο υποχρεωτικό όρισμα που έχει είναι ο τύπος της οσμής. Στη συνέχεια, μέσω των διαφόρων μεθόδων που υπάρχουν στην κλάση αυτή, μπορούν να οριστούν οι διάφορες πληροφορίες που απαιτούνται για την οσμή και αφού αυτό το βήμα ολοκληρωθεί, απαιτείται η κλήση της μεθόδου *build()*, ώστε να

δημιουργηθεί και να επιστραφεί ένα αντικείμενο τύπου *Smell* με τα δεδομένα που προστέθηκαν.

Το επόμενο κομμάτι, αφορά τον βασικό κορμό του plug-in. Στην Εικόνα 15, φαίνονται οι κλάσεις που απαρτίζουν τον βασικό αυτό κορμό. Αρχικά, η κλάση *Activator* είναι υπεύθυνη για την εκκίνηση του plug-in και ορίζει ένα μοναδικό ID για αυτό. Κατά τη δημιουργία ενός νέου plug-in project στο Eclipse, η κλάση αυτή δημιουργείται αυτόματα. Στην περίπτωση του SmellDetectorMerger, δεν έγινε κάποια αλλαγή στην εν λόγω κλάση. Στη συνέχεια, ένα βασικό στοιχείο του εργαλείου είναι η κλάση *SmellDetectionHandler*. Στο plug-in, ορίζονται διάφορες ενέργειες τις οποίες χρησιμοποιεί ο χρήστης προκειμένου να αλληλεπιδράσει με αυτό. Οι ενέργειες αυτές αφορούν κλικ στο όνομα μιας οσμής ώστε να ξεκινήσει η αναζήτηση γι' αυτή. Όπως αναφέρθηκε, εκτός από τις οσμές, υπάρχει η δυνατότητα επιλογής της αναζήτησης όλων των οσμών (*ALL_SMELLS*) αλλά και η εισαγωγή οσμών από αρχείο csv (*IMPORT_CSV*) όπως φαίνεται και στην Εικόνα 15. Όταν ο χρήστης επιλέξει το *SmellType* που επιθυμεί, αναλαμβάνει η *SmellDetectionHandler* να εξάγει πληροφορίες για το επιλεγμένο προς ανάλυση project και στη συνέχεια να περάσει τόσο αυτές όσο και το επιλεγμένο *SmellType* στην επόμενη κλάση που ελέγχει όλη τη διαδικασία εντοπισμού, την *SmellDetectionManager*.



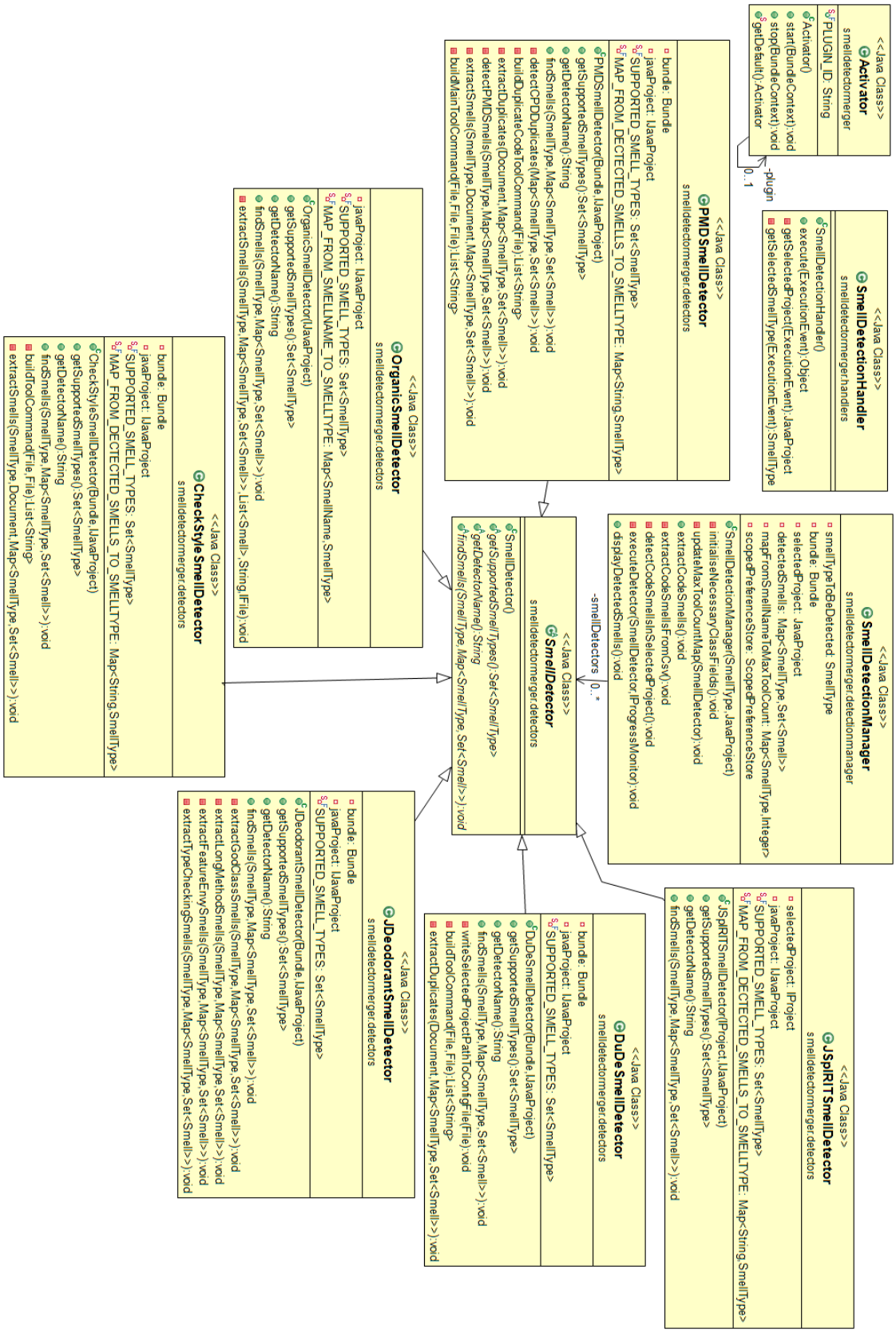
Εικόνα 15: Διαθέσιμες επιλογές κατά τη χρήση του SmellDetectorMerger

Αρμοδιότητες της *SmellDetectionManager* είναι η ανάλυση του επιλεγμένου project και η προβολή των εντοπισμένων οσμών. Αρχικά, πραγματοποιεί έναν έλεγχο ώστε να βρει τα ενεργοποιημένα ενσωματωμένα εργαλεία και στη συνέχεια για κάθε ένα από αυτά, εφόσον περιλαμβάνει την επιλεγμένη από τον χρήστη οσμή (ή αν ο χρήστης έχει επιλέξει τον εντοπισμό όλων των οσμών), εκτελείται ο εντοπισμός οσμών μέσω αυτού και γεμίζει μια δομή με τις οσμές που εντοπίστηκαν. Η προηγούμενη διαδικασία επαναλαμβάνεται μέχρις ότου εκτελεστούν όλα τα επιλεγμένα εργαλεία. Αφού ολοκληρωθεί η διαδικασία αυτή, εκτυπώνονται τα αποτελέσματα που έχουν εντοπιστεί στο *SmellDetectorMerger Smells View* (βλ. ενότητα 4.2.2).

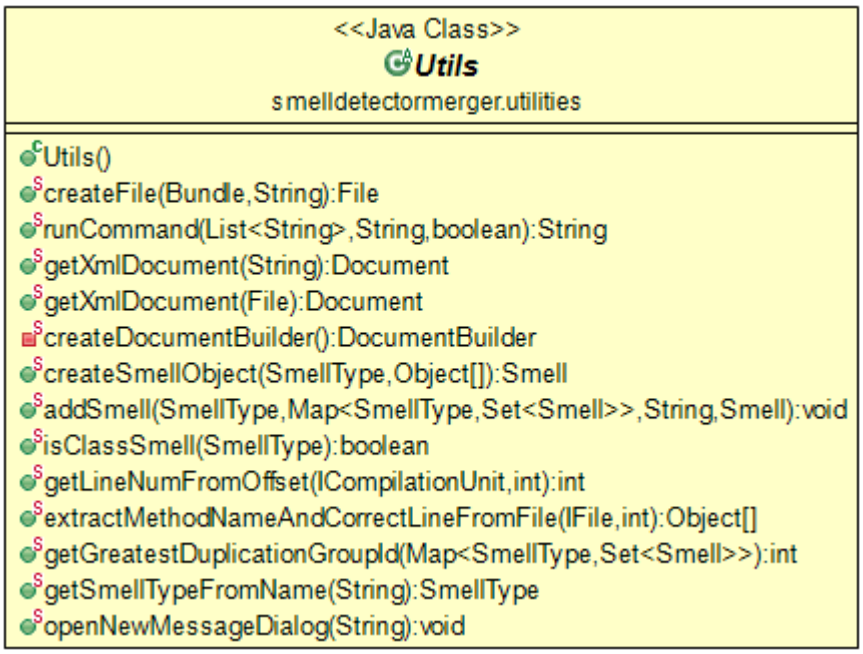
Τέλος, το επόμενο βασικό κομμάτι του plug-in είναι ο ορισμός των ενσωματωμένων εργαλείων, με σκοπό η εκτέλεσή τους να πραγματοποιείται με τέτοιο τρόπο, ώστε να επιστρέφονται ομοιόμορφα τα αποτελέσματα και να μπορεί η *SmellDetectionManager*, που αναλύθηκε προηγουμένως, να χειριστεί το ίδιο όλα τα εργαλεία. Για το λόγο αυτό, δημιουργήθηκε η κλάση *SmellDetector*, η οποία είναι abstract και αναπαριστά ένα εργαλείο εντοπισμού οσμών. Η κλάση αυτή έχει τρεις μεθόδους, οι οποίες αφορούν την επιστροφή του ονόματος του εργαλείου και τις οσμές που υποστηρίζει, καθώς και την εκτέλεση του εντοπισμού των οσμών και επιστροφή των αποτελεσμάτων. Κάθε μία από τις κλάσεις που κληρονομεί την κλάση *SmellDetector*, υλοποιεί τις μεθόδους αυτές και ορίζει τα στοιχεία και τη λειτουργία του εργαλείου που αντιπροσωπεύει. Την τρέχουσα χρονική στιγμή, υπάρχουν έξι (6) κλάσεις οι οποίες κληρονομούν την κλάση *SmellDetector*, όπως φαίνεται και στην Εικόνα 16, κάθε μια εκ των οποίων αντιστοιχεί και σε κάποιο από τα έξι (6) εργαλεία που έχουν ενσωματωθεί. Ανεξάρτητα από τον τρόπο εκτέλεσης του κάθε εργαλείου, του οποίου η λογική εκτελείται μέσα στη μέθοδο *findSmells()*, τα ευρήματα επιστρέφονται από αυτή σε μια συγκεκριμένη δομή. Επίσης, ο ορισμός των υποστηριζόμενων οσμών στη μέθοδο *getSupportedSmellTypes()*, εξυπηρετεί στην προσπέραση των εργαλείων που δεν περιέχουν τον επιλεγμένο από τον χρήστη τύπο οσμής.

Οι κλάσεις οι οποίες συζητήθηκαν προηγουμένως, καθώς και κάποιες από αυτές που θα συζητηθούν στις επόμενες ενότητες, χρησιμοποιούν λογική που είναι η ίδια μεταξύ τους. Για το λόγο αυτό, δημιουργήθηκε μια κλάση που περιέχει όλη αυτή τη λογική ώστε να παρέχεται σε όσες τη χρειάζονται από ένα κοινό σημείο. Ένα UML διάγραμμα για την κλάση αυτή, φαίνεται στην Εικόνα 17. Η Εικόνα αυτή, περιλαμβάνει

μόνο τη συγκεκριμένη κλάση καθώς σχετίζεται με τις περισσότερες από τις υπόλοιπες κλάσεις και ένα διάγραμμα το οποίο τις περιλαμβάνει όλες θα ήταν ιδιαίτερα πολύπλοκο.



Εικόνα 16: Διάγραμμα UML για τις βασικές κλάσεις



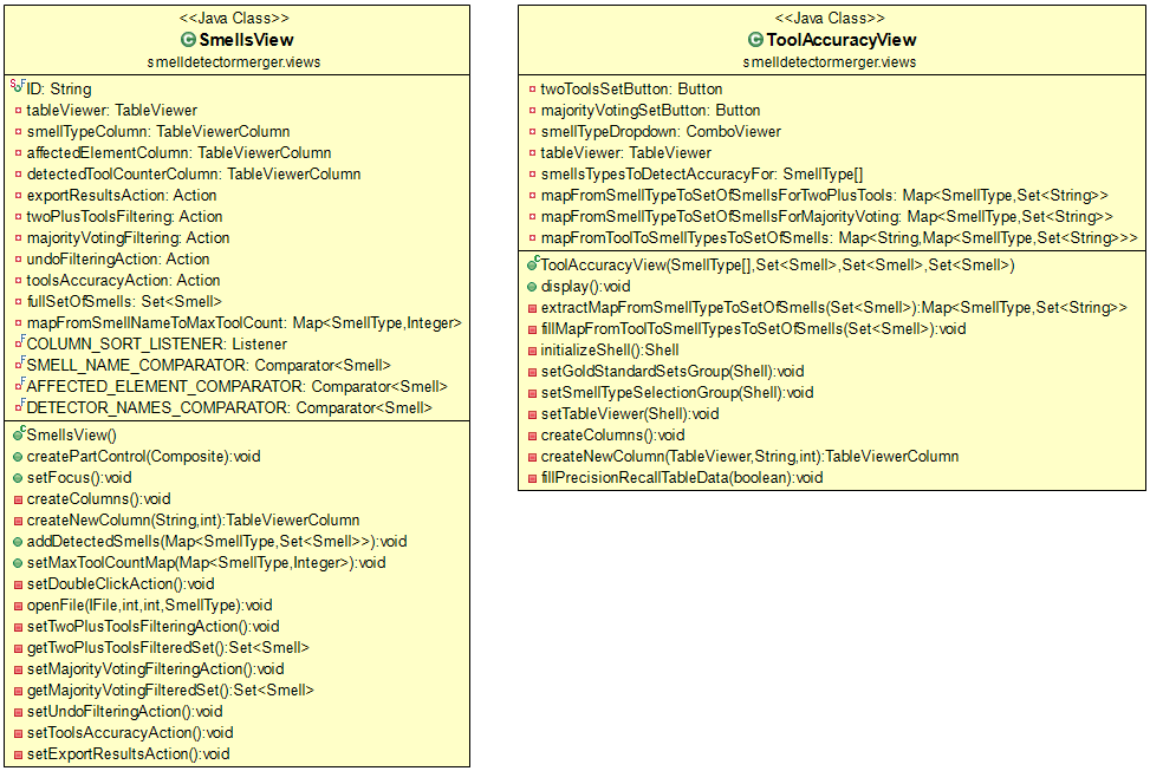
Εικόνα 17: Διάγραμμα UML για την κλάση Utils

Η κλάση *Utils*, είναι δηλωμένη ως static, έτσι ώστε να μην απαιτείται η δημιουργία αντικειμένων της για την κλήση των μεθόδων που διαθέτει. Μια από τις μεθόδους που χρησιμοποιείται από όλες τις κλάσεις που κληρονομούν την κλάση *SmellDetector* είναι η *addSmell()* που είτε προσθέτει μια νέα οσμή στο υπάρχον σύνολο οσμών, είτε προσθέτει το όνομα του επιπλέον εργαλείου που εντόπισε την οσμή, εφόσον η τελευταία υπάρχει ήδη μέσα στο σύνολο. Επίσης, μια μέθοδος που χρησιμοποιείται από όλα τα εργαλεία που εκτελούνται μέσω της γραμμής εργαλείων είναι η *runCommand()*, η οποία μεταξύ άλλων παίρνει ως όρισμα μια λίστα με τα τμήματα της εντολής που θα εκτελεστεί, την εκτελεί και επιστρέφει τα αποτελέσματα. Ακόμη μια συχνά χρησιμοποιούμενη μέθοδος είναι η *isClassSmell()*, η οποία ελέγχει αν μια δοσμένη οσμή αντιστοιχεί σε οσμή κλάσης ή όχι. Τέλος, η μέθοδος *openNewMessageDialog()* χρησιμοποιείται επίσης από διάφορες κλάσεις και αρμοδιότητά της είναι να ανοίγει ένα νέο παράθυρο, κυρίως όταν προκύπτουν λάθη κατά την εκτέλεση του προγράμματος, ενημερώνοντας τον χρήστη με ένα κατάλληλο μήνυμα λάθους, το οποίο δέχεται ως όρισμα.

4.2.2 Διαθέσιμα Views

Τα Views, αποτελούν ουσιαστικά λειτουργικότητα η οποία προσφέρεται από το εργαλείο είτε για προβολή δεδομένων στον χρήστη, είτε για αλληλεπίδραση με αυτόν. Στο SmellDetectorMerger, περιλαμβάνονται την τρέχουσα στιγμή δύο μόνο Views. Το

ένα αφορά την προβολή των οσμών που εντοπίστηκαν από τα διάφορα εργαλεία εντοπισμού που έχουν επιλεγεί και την παροχή διάφορων επιλογών όπως θα συζητηθεί σε λίγο. Το δεύτερο, σχετίζεται με την προβολή πληροφορίων σχετικά με την ακρίβεια των εργαλείων που χρησιμοποιήθηκαν και η οποία υπολογίζεται για την εκάστοτε οσμή με βάση τα αποτελέσματα που εξήχθησαν. Στην Εικόνα 18, φαίνεται το διάγραμμα UML, το οποίο περιέχει τις κλάσεις που σχετίζονται με το συγκεκριμένο τμήμα του plug-in. Όπως στην περίπτωση της κλάσης *Utils* έτσι και εδώ προβάλλονται στο διάγραμμα μόνο αυτές οι κλάσεις, χωρίς να υπάρχουν οι υπόλοιπες με τις οποίες αλληλεπιδρούν για λόγους απλότητας.



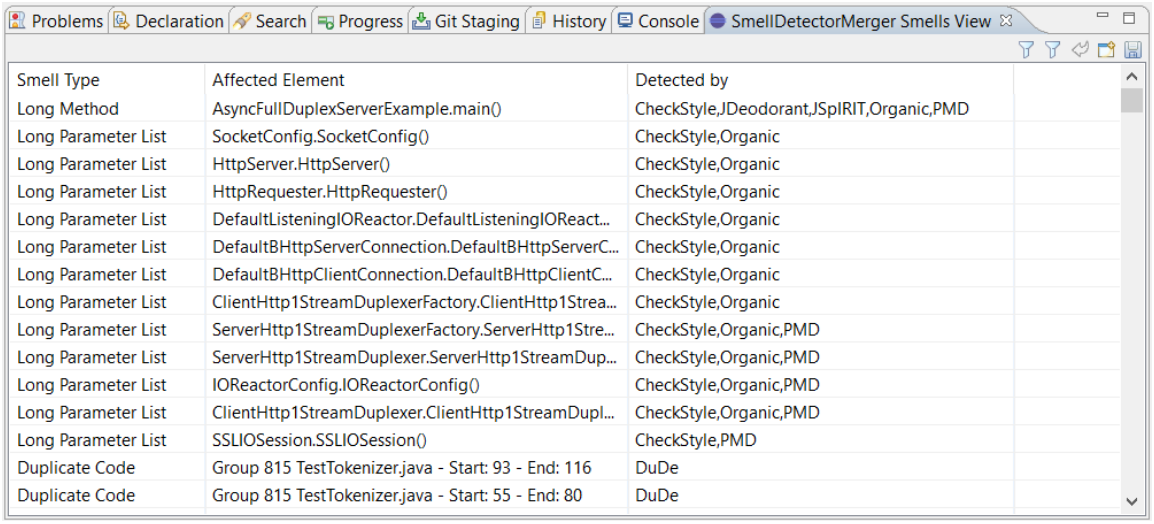
Εικόνα 18: Διάγραμμα UML για τις κλάσεις που αντιστοιχούν στα Views

Η κλάση *SmellsView* αφορά την προβολή των οσμών που εντοπίστηκαν από το plug-in. Αφού ολοκληρωθεί ο εντοπισμός μέσω της κλάσης *SmellDetectionManager* που συζητήθηκε στην ενότητα 4.2.1, τα αποτελέσματα μεταβιβάζονται στην κλάση *SmellsView*, η οποία αναλαμβάνει την προβολή και επεξεργασία τους. Ένα παράδειγμα του συγκεκριμένου View στο οποίο παρουσιάζονται αποτελέσματα εντοπισμένων οσμών ενός open-source project (βλ ενότητα 5), φαίνεται στην Εικόνα 19. Κάθε γραμμή στη λίστα αντιστοιχεί και σε μια διαφορετική οσμή, ενώ για κάθε μία εξ αυτών υπάρχουν τρεις στήλες στις οποίες περιλαμβάνονται ο τύπος της οσμής, το στοιχείο (μέθοδος ή κλάση) στο οποίο εντοπίστηκε η συγκεκριμένη οσμή καθώς και το πλήθος των

εργαλείων, χωρισμένο με κόμμα, που εντόπισαν τη συγκεκριμένη οσμή. Επίσης, το plug-in επιτρέπει στους χρήστες να κάνουν διπλό κλικ σε μια οσμή που τους ενδιαφέρει και να μεταφερθούν σε αυτή, ανοίγοντας το συγκεκριμένο αρχείο κώδικα στο Eclipse και μεταβαίνοντας στη γραμμή του κώδικα όπου αυτή εντοπίστηκε. Ακόμη, υπάρχει η δυνατότητα ταξινόμησης των οσμών ως προς κάθε μια από τις υπάρχουσες στήλες. Κάνοντας κλικ στον τίτλο μιας στήλης, τα αποτελέσματα ταξινομούνται αλφαβητικά σε αύξουσα ή φθίνουσα σειρά, ανάλογα με το πόσα κλικ έχει κάνει ο χρήστης (με το πρώτο κλικ ταξινομούνται σε αύξουσα σειρά ενώ με το δεύτερο σε φθίνουσα).

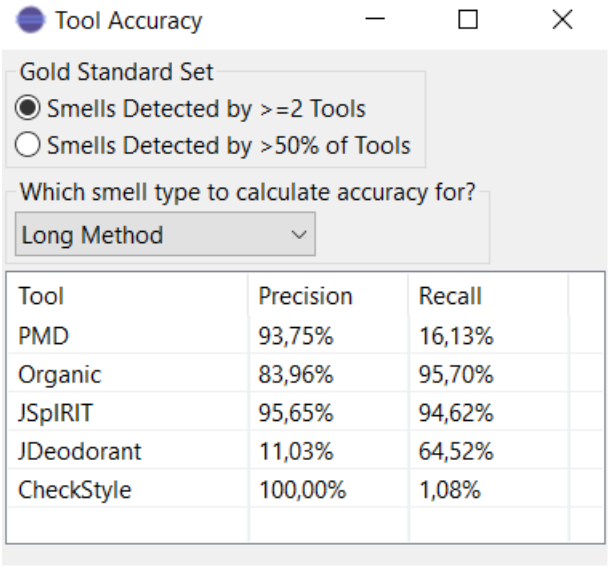
Περαιτέρω επιλογές για φιλτράρισμα και αξιοποίηση των αποτελεσμάτων παρέχονται μέσω των κουμπιών που περιέχονται στο View και εντοπίζονται πάνω δεξιά. Αρχικά, τα πρώτα δύο κουμπιά, τα οποία έχουν το εικονίδιο του φίλτρου, επιτρέπουν το φιλτράρισμα των δεδομένων. Το πρώτο κουμπί, φιλτράρει τα δεδομένα έτσι ώστε να διατηρηθούν μόνο αυτά που εντοπίστηκαν από αριθμό εργαλείων >=2, ενώ το δεύτερο τα φιλτράρει με τέτοιο τρόπο ώστε να διατηρηθούν αυτά που εντοπίστηκαν από ποσοστό >50% των εργαλείων. Σκοπός των συγκεκριμένων λειτουργιών είναι ο διαχωρισμός των οσμών που εντοπίστηκαν από πολλά εργαλεία, καθώς η συμφωνία αρκετών εργαλείων για την ύπαρξη οσμής σε κάποιο συγκεκριμένο στοιχείο αποτελεί ένδειξη ότι πραγματικά υπάρχει κάποιο πρόβλημα στο στοιχείο αυτό. Τα παραπάνω φιλτραρίσματα μπορούν εύκολα να αναιρεθούν με τη χρήση του τρίτου κουμπιού, το οποίο απεικονίζεται ως ένα βέλος που δείχνει προς τα αριστερά. Πατώντας το πλήκτρο αυτό, οποιοδήποτε φιλτράρισμα έχει εφαρμοσθεί αναιρείται και τα αποτελέσματα επιστρέφουν στην αρχική τους κατάσταση.

Το επόμενο κουμπί, το τέταρτο, σχετίζεται με την κλάση *ToolAccuracyView* και αφορά το άλλο View που προσφέρεται από το plug-in, το οποίο έχει να κάνει με τον υπολογισμό της ακρίβειας. Η ακρίβεια που αναφέρεται εδώ, μεταφράζεται στη χρήση των δεικτών της ακρίβειας (precision) και της ανάκλησης (recall), ο ορισμός των οποίων δόθηκε στην ενότητα 1.1. Οι δύο δείκτες αυτοί αποτελούν έναν σημαντικό παράγοντα που μετρά το πόσο αποτελεσματικές είναι οι τεχνικές εντοπισμού που υλοποιεί το κάθε εργαλείο σχετικά με τον εντοπισμό οσμών σε ένα έργο λογισμικού.



Smell Type	Affected Element	Detected by
Long Method	AsyncFullDuplexServerExample.main()	CheckStyle,JDeodorant,JSpIRIT,Organic,PMD
Long Parameter List	SocketConfig.SocketConfig()	CheckStyle,Organic
Long Parameter List	HttpServer.HttpServer()	CheckStyle,Organic
Long Parameter List	HttpRequester.HttpRequester()	CheckStyle,Organic
Long Parameter List	DefaultListeningIOReactor.DefaultListeningIOReact...	CheckStyle,Organic
Long Parameter List	DefaultBHttpServerConnection.DefaultBHttpServerC...	CheckStyle,Organic
Long Parameter List	DefaultBHttpClientConnection.DefaultBHttpClientC...	CheckStyle,Organic
Long Parameter List	ClientHttp1StreamDuplexerFactory.ClientHttp1Strea...	CheckStyle,Organic
Long Parameter List	ServerHttp1StreamDuplexerFactory.ServerHttp1Stre...	CheckStyle,Organic,PMD
Long Parameter List	ServerHttp1StreamDuplexer.ServerHttp1StreamDup...	CheckStyle,Organic,PMD
Long Parameter List	IOReactorConfig.IOReactorConfig()	CheckStyle,Organic,PMD
Long Parameter List	ClientHttp1StreamDuplexer.ClientHttp1StreamDupl...	CheckStyle,Organic,PMD
Long Parameter List	SSLIOSession.SSLIOSession()	CheckStyle,PMD
Duplicate Code	Group 815 TestTokenizer.java - Start: 93 - End: 116	DuDe
Duplicate Code	Group 815 TestTokenizer.java - Start: 55 - End: 80	DuDe

Εικόνα 19: Παράδειγμα προβολής εντοπισμένων οσμών



Tool	Precision	Recall
PMD	93,75%	16,13%
Organic	83,96%	95,70%
JSpIRIT	95,65%	94,62%
JDeodorant	11,03%	64,52%
CheckStyle	100,00%	1,08%

Εικόνα 20: Παράδειγμα υπολογισμού Precision και Recall των εργαλείων

Κάνοντας κλικ, λοιπόν, στο τέταρτο αυτό κουμπί, εμφανίζεται στην οθόνη ένα καινούριο παράθυρο σαν αυτό που απεικονίζεται στην Εικόνα 20. Όπως προαναφέρθηκε, για τον υπολογισμό του ποσοστού των precision και recall, είναι απαραίτητη η ύπαρξη ενός συνόλου το οποίο θα περιέχει οσμές που θεωρούνται πραγματικά υπαρκτές μέσα στο έργο λογισμικού. Το σύνολο αυτό ονομάζεται και gold standard set. Τη θέση αυτού του σετ στο plug-in παίρνουν τα δύο σετ οσμών που εντοπίζονται από αριθμό εργαλείων >=2 και >50%. Ο χρήστης μπορεί κάθε φορά να επιλέξει ποιο από τα δύο θέλει να χρησιμοποιήσει και θα δει αντίστοιχα τις τιμές των ποσοστών στις δύο αυτές μετρικές να αλλάζει. Επίσης, εκτός από την επιλογή μεταξύ των δύο gold standard set, ο χρήστης πρέπει να επιλέξει από το αντίστοιχο μενού και την

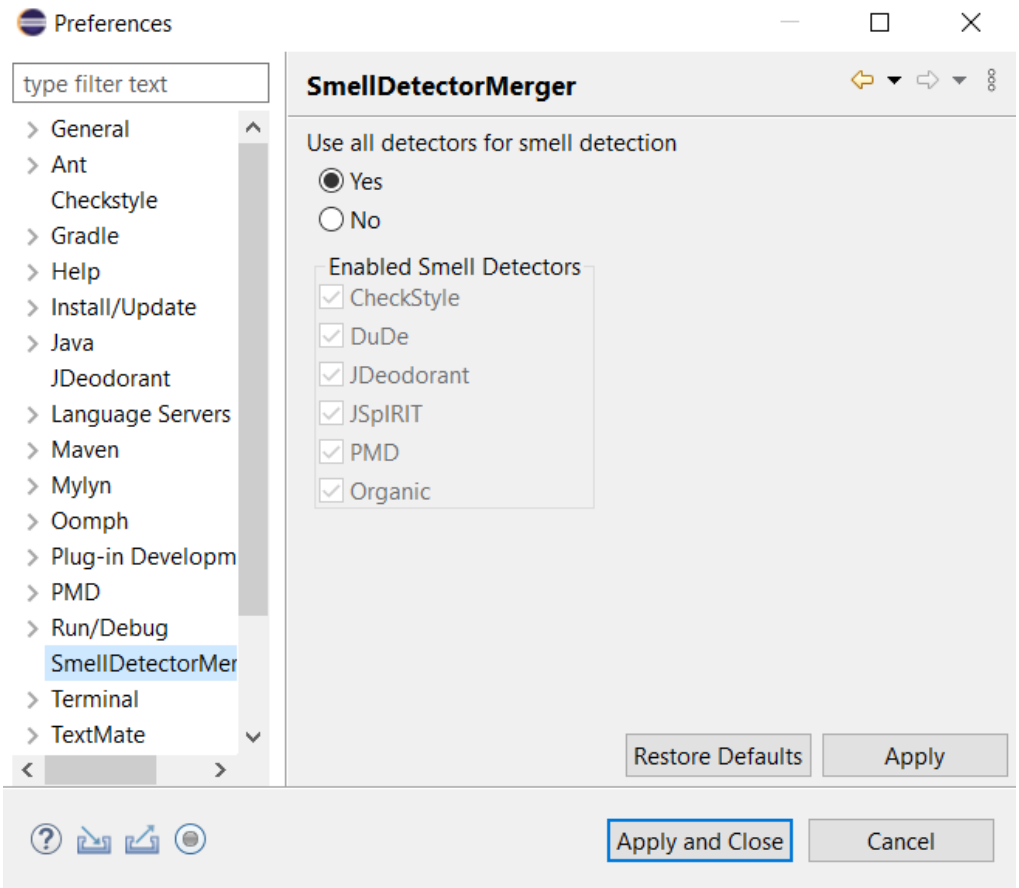
οσμή για την οποία θέλει να προβάλει τα αποτελέσματα. Τα ποσοστά που μπορεί να πάρει ο χρήστης ακολουθώντας την προηγούμενη διαδικασία έχουν αξία, διότι μπορούν να αποτελέσουν μια πρώτη ένδειξη της αποτελεσματικότητας του εκάστοτε εργαλείου. Ειδικά στις περιπτώσεις όπου οσμές μπορούν να εντοπιστούν από την πλειοψηφία των εργαλείων, μπορούν να ξεχωρίσουν αυτά τα οποία είναι πιο αποδοτικά, καθώς τα gold standard set θα έχουν πιο αξιόπιστα δεδομένα.

Το τελευταίο κουμπί το οποίο παρέχεται στο View των αποτελεσμάτων εντοπισμού και έχει το εικονίδιο δισκέτας αποθήκευσης, αφορά την αποθήκευση των αποτελεσμάτων σε ένα αρχείο *csv*. Όταν το πατήσει ο χρήστης, το plug-in αναλαμβάνει να αποθηκεύσει όλα τα δεδομένα τα οποία έχουν εξαχθεί με έναν δομημένο τρόπο μέσα σε ένα καινούριο *csv* αρχείο. Σημαντικό είναι, επίσης, ότι ο χρήστης μπορεί να αποθηκεύσει όλα τα δεδομένα τα οποία υπάρχουν στο View την τρέχουσα στιγμή που πατηθεί το κουμπί αυτό. Το γεγονός αυτό συνεπάγεται ότι αν ο χρήστης είχε προηγουμένως φιλτράρει για παράδειγμα τα αποτελέσματα ώστε να προβάλει μόνο αυτά τα οποία έχουν εντοπιστεί από >=2 εργαλεία, τότε μόνο τα αποτελέσματα αυτά θα αποθηκευτούν στο νέο αρχείο. Η λειτουργία αυτή μπορεί να διευκολύνει τους χρήστες σε μεγάλο βαθμό, διότι μετά το πέρας του εντοπισμού, μπορούν να αποθηκεύσουν τα δεδομένα και να τα επεξεργαστούν όποτε το επιθυμούν, χωρίς να χρειάζεται να εκτελέσουν από την αρχή το plug-in. Επίσης, μέσω του *csv* αρχείου, μπορούν τα δεδομένα να εισαχθούν σε ένα πρόγραμμα επεξεργασίας και να εξαχθούν μετρήσεις και διαγράμματα που προκύπτουν από αυτά. Τέλος, όπως συζητήθηκε και στην ενότητα 4.2.1, το SmellDetectorMerger προσφέρει τη δυνατότητα εισόδου δεδομένων από ένα *csv* αρχείο. Με τον τρόπο αυτό, ο χρήστης μπορεί να επιλέξει ένα αρχείο που έχει προηγουμένως εξαχθεί από το ίδιο το εργαλείο και να φορτώσει όλα τα δεδομένα που είχαν αποθηκευτεί. Με τον τρόπο αυτό, καθίστανται άμεσα διαθέσιμες όλες οι επιλογές ταξινόμησης, φιλτραρίσματος και υπολογισμού της ακρίβειας των επιμέρους εργαλείων, χωρίς την ανάγκη επανέναρξης της διαδικασίας εντοπισμού.

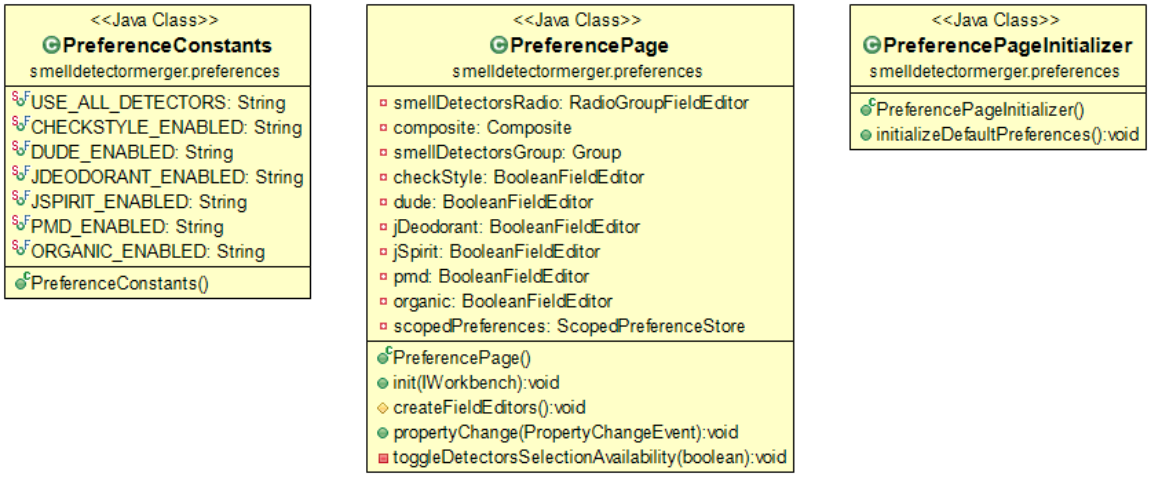
4.2.3 Διαθέσιμες επιλογές ως Preferences

Το Eclipse, δίνει τη δυνατότητα στα plug-ins να προσφέρουν στους χρήστες τους την επιλογή να θέσουν προτιμήσεις που χρησιμοποιούνται στη συνέχεια από τα ίδια και καθορίζουν σε κάποιο βαθμό τον τρόπο λειτουργίας τους. Όπως έχει ήδη αναφερθεί, οι επιλογές αυτές βρίσκονται στο μονοπάτι “*Window*” -> “*Preferences*” στο Eclipse και

από εκεί πρέπει να αναζητήσουν το όνομα του plug-in, εφόσον βέβαια αυτό προσφέρει τη δυνατότητα ορισμού προτιμήσεων. Στο SmellDetectorMerger λοιπόν, υποστηρίζονται τα Preferences που απεικονίζονται στην Εικόνα 21. Όπως είναι προφανές, στην περίπτωση αυτή αφορούν μόνο τη ρύθμιση των επιμέρους εργαλείων, τα οποία θα χρησιμοποιηθούν για τον εντοπισμό οσμών. Εξ ορισμού, η πρώτη επιλογή έχει την τιμή "*Yes*", με αποτέλεσμα όλα τα εργαλεία να είναι ενεργοποιημένα και να χρησιμοποιούνται. Εφόσον όμως ο χρήστης το επιθυμεί, μπορεί να αλλάξει αυτή την επιλογή, κάνοντας κλικ στο "*No*" , ενεργοποιώντας με αυτόν τον τρόπο τις επιλογές στην ομάδα "*Enabled Smell Detectors*", ώστε να επιλέξει ο ίδιος συγκεκριμένα ποια εργαλεία επιθυμεί να χρησιμοποιηθούν. Αφού πραγματοποιήσει τις επιλογές που επιθυμεί και πατήσει "*Apply and Close*", οι επιλογές του αποθηκεύονται και χρησιμοποιούνται στην επόμενη αναζήτηση. Τέλος, για να γίνουν περισσότερο κατανοητά τα κομμάτια που σχετίζονται με τα Preferences στο SmellDetectorMerger, έχει προστεθεί το διάγραμμα UML για τα στοιχεία αυτά στην Εικόνα 22.



Εικόνα 21: Preferences του SmellDetectorMerger



Εικόνα 22: Διάγραμμα UML των κλάσεων για τα Preferences

4.3 Μελέτη περίπτωσης

Στην ενότητα αυτή, παρουσιάζεται μια μελέτη περίπτωσης η οποία πραγματοποιήθηκε, με σκοπό τη χρήση του SmellDetectorMerger σε ένα έργο λογισμικού ανοιχτού κώδικα, ώστε να αξιολογηθούν τα επιμέρους εργαλεία καθώς και ο βαθμός συμφωνίας μεταξύ τους στις οσμές που εντοπίζονται στο έργο αυτό. Για τους σκοπούς της μελέτης αυτής λοιπόν, επιλέχθηκε τμήμα του Apache HttpComponents, το οποίο αποτελεί ουσιαστικά ένα σύνολο εργαλείων χαμηλού επιπέδου της Java σχετικά με το πρωτόκολλο HTTP καθώς και άλλων συναφών πρωτοκόλλων (The Apache Software Foundation, 2022). Πιο συγκεκριμένα, το τελευταίο χωρίζεται σε δύο επιμέρους έργα εκ των οποίων το ένα αποτελεί τον πυρήνα για τη λειτουργία των εργαλείων και χρησιμοποιήθηκε κατά τη μελέτη περίπτωσης, ενώ το άλλο είναι μια υλοποίηση η οποία χρησιμοποιεί τον προηγούμενο πυρήνα. Επίσης, το Apache HttpComponents Core[5] που αποτελεί των βασικό πυρήνα που αναφέρθηκε πριν, αποτελείται από τέσσερα επιμέρους project, καθένα εκ των οποίων υλοποιεί μέρος της λειτουργίας του ολοκληρωμένου έργου.

Για να αξιολογηθεί το έργο αυτό λοιπόν, χρειάστηκε αρχικά να εισαχθούν και τα τέσσερα επιμέρους project στο Eclipse και στη συνέχεια να αναλυθούν, το ένα μετά το άλλο, από το SmellDetectorMerger ώστε να εντοπιστούν οι οσμές σε κάθε ένα από αυτά. Μετά το πέρας της εκτέλεσης και της προβολής των αποτελεσμάτων, αυτά αποθηκεύονταν σε ένα *csv* αρχείο με τη χρήση της λειτουργίας "*Export Results*" που

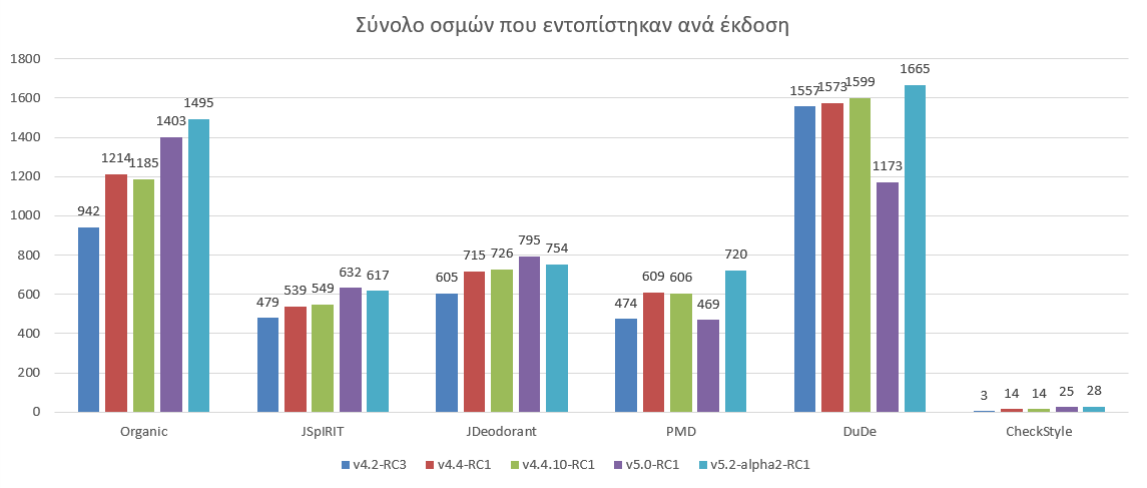
[5] https://github.com/apache/httpcomponents-core

συζητήθηκε στην προηγούμενη ενότητα. Στη συνέχεια, τα δεδομένα από κάθε επιμέρους *csv* αρχείο αποθηκεύτηκαν σε ένα κοινό αρχείο ώστε να γίνει επεξεργασία αυτών.

Ένα αρχικό εύρημα από τη χρήση του plug-in, είναι το πλήθος οσμών που εντοπίζονται από τα επιμέρους εργαλεία σε διάφορες εκδόσεις του HttpComponents Core project. Για τον σκοπό αυτό, χρησιμοποιήθηκε η σελίδα του GitHub από το HttpComponents Core που φαίνεται παραπάνω και πραγματοποιήθηκε μια αναζήτηση στις υπάρχουσες εκδόσεις του project. Από το σύνολο αυτών, επιλέχθηκαν κάποιες οι οποίες απέχουν χρονικά ένα σημαντικό διάστημα η μία από τη άλλη, έτσι ώστε να έχουν μεσολαβήσει διάφορες αλλαγές και να έχει τροποποιηθεί ο κώδικας του έργου σε σημαντικό βαθμό από την μία έκδοση στην επόμενη. Βέβαια, αξίζει να σημειωθεί ότι λόγο της ιδιαιτερότητας στη σειρά με την οποία κυκλοφορούν οι διάφορες εκδόσεις του, έχουν επιλεγεί εκδόσεις οι οποίες αποτελούν «υποψήφιες εκδόσεις» (Release Candidates – RC) και όχι τελικές εκδόσεις για τον συγκεκριμένο σκοπό.

Στην Εικόνα 23 παρακάτω λοιπόν, απεικονίζονται τα αποτελέσματα που εξήχθησαν από τις αναλύσεις των εκδόσεων αυτών. Στον κάθετο άξονα του διαγράμματος φαίνεται το πλήθος των οσμών, ενώ στον οριζόντιο υπάρχουν και τα έξι εργαλεία, για καθένα από τα οποία φαίνεται το πλήθος των οσμών που εντόπισε ανά έκδοση, ξεκινώντας από την πιο παλιά και καταλήγοντας στην πιο πρόσφατη. Στο κάτω μέρος του διαγράμματος, υπάρχει επίσης μια σημείωση σχετικά με το ποια έκδοση αντιστοιχεί σε ποιο χρώμα. Μια πρώτη παρατήρηση στο διάγραμμα, είναι ότι όσο προχωρούν οι εκδόσεις, το πλήθος των οσμών κατά κύριο λόγω αυξάνεται. Στην προηγούμενη παρατήρηση υπάρχουν βέβαια ορισμένες εξαιρέσεις, όπως για παράδειγμα η δεύτερη και η τρίτη έκδοση η οποία είτε παρέμεινε σταθερή είτε μεταβλήθηκε σε πάρα πολύ μικρό βαθμό σε όλα τα εργαλεία. Η συμφωνία αυτή όλων των εργαλείων ως προς αυτή τη μεταβολή, δείχνει ότι πιθανότατα δεν παρουσιάστηκαν σημαντικές αλλαγές στο έργο μεταξύ των δύο αυτών εκδόσεων, ή ότι αυτές που πραγματοποιήθηκαν προσέθεσαν πολύ μικρό αριθμό οσμών. Επίσης, ένα ακόμη σημείο στο οποίο συμφωνούν δύο εργαλεία είναι η πτώση των οσμών κατά τη μετάβαση από την τρίτη στην τέταρτη έκδοση. Βέβαια, ένα κοινό που έχουν τα δύο αυτά εργαλεία και ξεχωρίζει από τα υπόλοιπα, είναι ότι εντοπίζουν οσμές τύπου Duplicate Code. Ειδικά το εργαλείο DuDe, υποστηρίζει αποκλειστικά την εύρεση τέτοιων οσμών. Επομένως, οι Duplicate Code οσμές μειώθηκαν σημαντικά μεταξύ των δύο αυτών εκδόσεων και επομένως το πιθανότερο είναι και η πτώση στις οσμές που εντοπίζονται από το PMD να οφείλεται

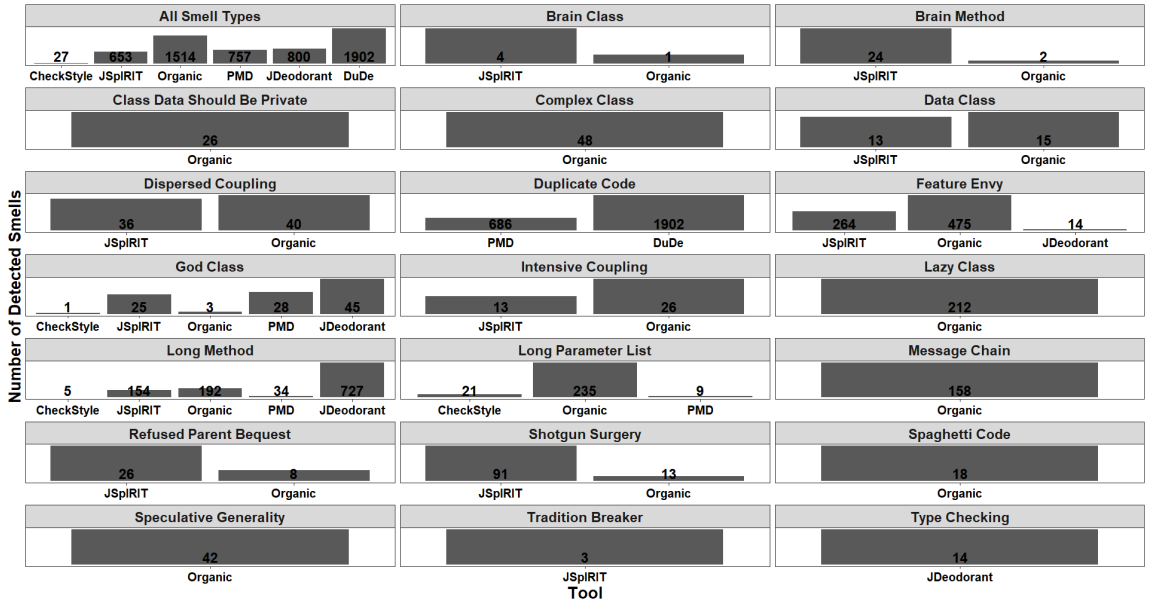
συγκεκριμένα σε αυτόν τον τύπο οσμής και όχι στις υπόλοιπες που υποστηρίζει, καθώς όλα τα υπόλοιπα εργαλεία εντόπισαν αύξηση στον συνολικό αριθμό των οσμών κατά τη μετάβαση μεταξύ των ίδιων εκδόσεων. Ακόμη, διαφορές παρατηρούνται και στο σύνολο των οσμών που εντοπίζονται μεταξύ των εργαλείων. Οι διαφορές αυτές, οφείλονται στις τεχνικές που χρησιμοποιεί το εκάστοτε εργαλείο, καθώς επίσης και στο πλήθος οσμών προς εντοπισμό που υποστηρίζεται από καθένα από αυτά. Για παράδειγμα, το Organic που έχει εμφανή διαφορά με το JSpIRIT και JDeodorant τα οποία αποτελούν επίσης plug-ins, εντοπίζει σημαντικά μεγαλύτερο πλήθος οσμών από τα τελευταία δύο και πιο συγκεκριμένα δεκαεπτά (17) έναντι έντεκα (11) και τέσσερα (4) αντίστοιχα. Αξιοσημείωτη διαφορά στις οσμές που εντοπίστηκαν παρουσιάζεται και στο CheckStyle, το οποίο εντόπισε εξαιρετικά μικρό αριθμό οσμών συγκριτικά με τα υπόλοιπα εργαλεία. Το γεγονός αυτό είναι πολύ πιθανό να οφείλεται στα πολύ αυστηρά κριτήρια που χρησιμοποιεί για την αναγνώριση των οσμών που υποστηρίζει και ενδέχεται να διαφοροποιούνταν σημαντικά αν είχαν χρησιμοποιηθεί άλλες τιμές πέραν αυτών που εφαρμόζει εξ ορισμού.



Εικόνα 23: Σύνολο οσμών που εντοπίστηκαν ανά έκδοση

Επόμενο βήμα στη μελέτη περίπτωσης ήταν η ανάλυση της τελευταίας σταθερής έκδοσης του Apache HttpComponents Core project και η εξαγωγή δεδομένων από αυτό. Στο συγκεκριμένο βήμα, μεγαλύτερη βάση δόθηκε στο πως συγκρίνονταν τα εργαλεία μεταξύ τους ως προς των τύπο οσμών που εντόπισαν, καθώς και τον βαθμό συμφωνίας που υπήρξε. Η έκδοση η οποία χρησιμοποιήθηκε εδώ ήταν η 5.1.2 και η διαδικασία που ακολουθήθηκε για την εξαγωγή και επεξεργασία των δεδομένων από το SmellDetectorMerger ήταν η ίδια με αυτή που περιγράφηκε προηγουμένως για την ανάλυση των διαφόρων εκδόσεων.

Όπως αναφέρθηκε προηγουμένως, ένα πρώτο εύρημα από την ανάλυση της τελευταίας έκδοσης του έργου, ήταν οι ομοιότητες και οι διαφορές ως προς τις οσμές που εντοπίστηκαν μεταξύ των επιμέρους εργαλείων. Στην Εικόνα 24 λοιπόν, παρουσιάζονται γραφήματα ανά τύπο οσμής τα οποία περιέχουν το(α) εργαλείο(α) που εντόπισε(αν) την συγκεκριμένη οσμή αλλά και το πλήθος των περιπτώσεων στις οποίες εντοπίστηκε η οσμή αυτή. Μοναδική εξαίρεση στο προηγούμενο, αποτελεί το διάγραμμα πάνω αριστερά, το οποίο απεικονίζει το σύνολο των οσμών που εντοπίστηκαν από κάθε εργαλείο ανεξάρτητα από τον τύπο της οσμής.



Εικόνα 24: Πλήθος οσμών που εντοπίστηκαν ανά τύπο οσμής και εργαλείο

Αρχικά, βλέποντας κανείς την Εικόνα, μπορεί με μια πρώτη να εντοπίσει ποια εργαλεία υποστηρίζουν ποιους τύπος οσμών, καθώς και ποιοι από τους τελευταίους είναι περισσότερο δημοφιλείς μεταξύ των εργαλείων και ποιοι όχι. Φαίνεται για παράδειγμα ότι από το σύνολο των οσμών, μόνο δύο υποστηρίζονται από το σύνολο των εργαλείων και βρίσκονται στη μέση αριστερά στην Εικόνα, ονομαστικά God Class και Long Method, με μοναδική εξαίρεση το DuDe, το οποίο όπως έχει αναφερθεί προορίζεται αποκλειστικά για τον εντοπισμό Duplicate Code. Παρατηρείται επίσης, ότι το πλήθος των οσμών οι οποίες εντοπίζονται από ένα μόνο εργαλείο είναι σημαντικό. Αυτό δείχνει ότι κάποιες οσμές δεν είναι ιδιαίτερα «δημοφιλείς» συγκριτικά με τις υπόλοιπες και εντοπίζονται από μικρότερο αριθμό εργαλείων. Πιο συγκεκριμένα, από τις 20 συνολικά οσμές που εντοπίζονται από το SmellDetectorMerger, οι 8 υποστηρίζονται από ένα μόνο εργαλείο, ενώ οι υπόλοιπες 12 υποστηρίζονται από δύο ή περισσότερα. Η σημαντικότερη όμως παρατήρηση στην Εικόνα 24, αφορά τις μεγάλες διαφορές που

υπάρχουν μεταξύ των εργαλείων ως προς το πλήθος οσμών που εντοπίζουν για έναν δεδομένο τύπο οσμής. Παίρνοντας για παράδειγμα την περίπτωση του Long Method, φαίνεται ότι τα εργαλεία έχουν πολύ μεγάλες διαφορές μεταξύ τους. Ξεκινώντας με το CheckStyle, φαίνεται ότι έχει εντοπίσει μόλις 5 σημεία στα οποία εμφανίζεται η οσμή αυτή ενώ τα JSpIRIT και Organic που βρίσκονται και σε σχετικά κοντινά επίπεδα μεταξύ τους, εντοπίζουν 154 και 192 περιπτώσεις αντίστοιχα, αριθμός που έχει σημαντική διαφορά από τον προηγούμενο. Στο JDeodorant βεβαίως, η διαφορά αυτή πολλαπλασιάζεται, καθώς εδώ εντοπίστηκαν συνολικά 727 περιπτώσεις της συγκεκριμένης οσμής. Το νούμερο αυτό απέχει σημαντικά από τα αντίστοιχα νούμερα των υπόλοιπων εργαλείων, ειδικά από το CheckStyle που ανέφερε τα λιγότερα ευρήματα. Αντίστοιχες περιπτώσεις παρατηρούνται και σε άλλους τύπους οσμής, όπως για παράδειγμα το Long Parameter List και το Feature Envy, στις οποίες το πλήθος οσμών που εντοπίζονται από το εργαλείο με τα λιγότερα ευρήματα συγκριτικά με αυτό με τα περισσότερα απέχει σε πολύ μεγάλο βαθμό. Βέβαια, υπάρχουν και κάποιες περιπτώσεις στις οποίες το πλήθος των ευρημάτων μεταξύ των εργαλείων είναι σε πολύ κοντινά επίπεδα, όπως για παράδειγμα στην περίπτωση του Data Class, στην οποία τα JSpIRIT και Organic εντόπισαν 13 και 15 οσμές αντίστοιχα, καθώς και σε αυτή του Dispersed Coupling, όπου τα ίδια εργαλεία εντόπισαν 36 και 40 οσμές αντίστοιχα.

Ενώ το πλήθος των οσμών που εντοπίζονται ανά εργαλείο είναι ένα σημαντικό εύρημα, ένα ακόμη σημαντικότερο είναι η εύρεση του βαθμού συμφωνίας μεταξύ των εργαλείων ως προς τις οσμές που εντόπισαν. Το τελευταίο, αποτέλεσε στόχο του δεύτερου βήματος της μελέτης που πραγματοποιήθηκε. Τα αποτελέσματα που εξήχθησαν και χρησιμοποιήθηκαν για τον υπολογισμό των τιμών στα ευρήματα που παρουσιάστηκαν προηγουμένως, αξιοποιήθηκαν επίσης και για να μετρηθεί ο βαθμός συμφωνίας μεταξύ των ευρημάτων αυτών. Από τους διαθέσιμους δείκτες για τον υπολογισμό του βαθμού συμφωνίας, επιλέχθηκε να χρησιμοποιηθεί αυτός του positive specific agreement. Ο συγκεκριμένος, ορίζεται ως “η υπό όρους πιθανότητα ότι ένας βαθμολογητής θα συμφωνήσει πως μια περίπτωση είναι θετική δεδομένου ότι ο άλλος την βαθμολόγησε θετική, όπου ο ρόλος των δύο βαθμολογητών επιλέγεται τυχαία. Προσεγγίζει ουσιαστικά το ποσοστό των θετικών υποθέσεων που συμφωνήθηκαν” (Hripcsak & Rothschild, 2005). Επιλέχθηκε ο δείκτης αυτός, διότι ο υπολογισμός του δίνει μεγαλύτερη βάση στις θετικές συμφωνίες που βρίσκονται μεταξύ των βαθμολογητών (raters), όπου στη δική μας περίπτωση, θετική συμφωνία σημαίνει ότι

δύο εργαλεία έχουν εντοπίσει τον ίδιο τύπο οσμής σε ένα συγκεκριμένο, κοινό, σημείο. Επίσης, ως βαθμολογητές θεωρούνται τα επιμέρους εργαλεία που εντοπίζουν τις οσμές. Στην Εικόνα 25 παρακάτω, απεικονίζεται ο τύπος με τον οποίο υπολογίζεται ο συγκεκριμένος δείκτης, καθώς και μια επεξήγηση των μερών του.

		Rater 2's judgment	
		Positive	**Negative**
Rater 1's judgment	Positive	a	b
	Negative	c	d

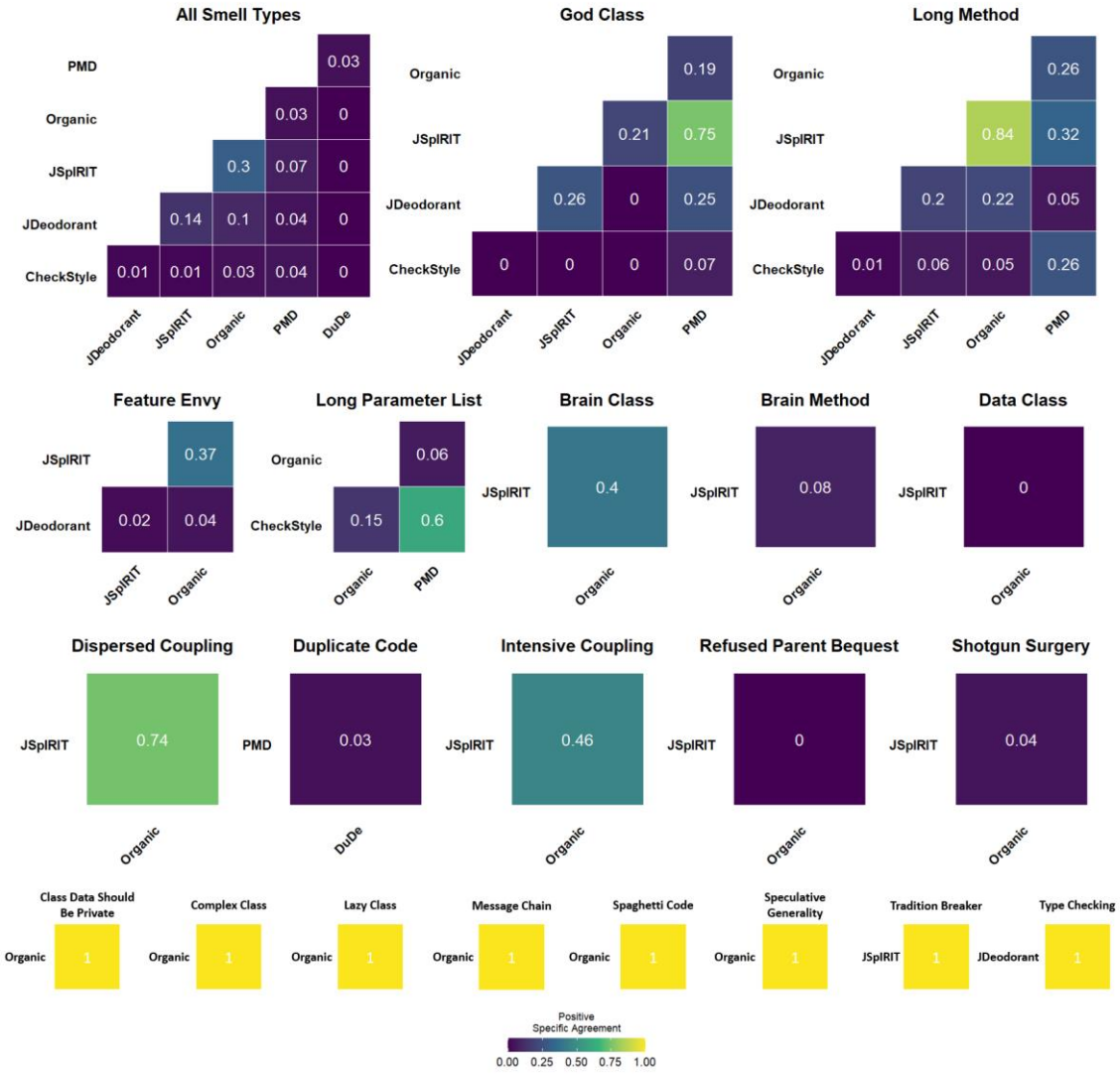
$$P_{pos} = 2a/(2a + b + c)$$

Εικόνα 25: Τύπος υπολογισμού του positive specific agreement

Στην Εικόνα 26 παρακάτω, έχουν αποτυπωθεί τα αποτελέσματα από τον υπολογισμό του positive specific agreement για όλους τους τύπους οσμής ανά εργαλείο. Εξαίρεση στο τελευταίο αποτελεί το διάγραμμα που φαίνεται πάνω αριστερά στην Εικόνα, το οποίο απεικονίζει τον βαθμό συμφωνίας μεταξύ των εργαλείων επί του συνόλου των οσμών, ανεξάρτητα από τον τύπο των οσμών που εντοπίζει το καθένα. Γενικότερα, σε όλες τις περιπτώσεις, το μοτίβο που ακολουθείται είναι ότι παρουσιάζονται τα αποτελέσματα ανά τύπο οσμής και η τομή δύο εργαλείων δείχνει την τιμή του positive specific agreement που προέκυψε μεταξύ των δύο. Επίσης, στο τέλος του διαγράμματος φαίνεται και η κλίμακα, χρωματική αλλά και σε τιμές, στην οποία αποτυπώθηκαν τα αποτελέσματα.

Παρατηρώντας τα διάφορα διαγράμματα, καθίσταται σαφές ότι στο σύνολο των περιπτώσεων ο βαθμός συμφωνίας βρίσκεται σε πολύ χαμηλά επίπεδα. Στην πλειοψηφία των περιπτώσεων, η τιμή του positive specific agreement κυμαίνεται σε επίπεδα κάτω του 0.2-0.3, ενώ σε πολύ μεγάλο ποσοστό, ο δείκτης αυτός έχει ακόμα και τιμές ίσες ή πάρα πολύ κοντά στο 0. Βέβαια, υπάρχουν και εξαιρέσεις στην κατάσταση αυτή, όπως για παράδειγμα το Long Method και συγκεκριμένα για την τομή των JSpIRIT και Organic, καθώς και το God Class για τα JSpIRIT και PMD αλλά και το Dispersed Coupling για τα JSpIRIT και Organic με τιμές 0.84, 0.75 και 0.74 αντίστοιχα. Επίσης, κοντά στα επίπεδα αυτά βρίσκεται και η περίπτωση του Long Parameter List για την τομή των Organic και PMD με τιμή 0.6. Αξίζει να σημειωθεί όμως, ότι ως

ικανοποιητικές για την ύπαρξη ασφαλούς συμφωνίας, θεωρούνται τιμές >=0.8, που παρατηρείται σε μία μόνο περίπτωση από το σύνολο, αυτή του Long Method που αναφέρθηκε προηγουμένως. Τέλος, αξίζει να σημειωθεί επίσης ότι έχουν προστεθεί στην τελευταία σειρά της εικόνας και περιπτώσεις στις οποίες ένας τύπος οσμής εντοπίζεται από ένα μόνο εργαλείο, όπου και για προφανείς λόγους η τιμή του positive specific agreement είναι 1.



Εικόνα 26: Positive Specific Agreement

5 Επίλογος

5.1 Σύνοψη και συμπεράσματα

Στην παρούσα εργασία, έγινε μια μικρή παρουσίαση των οσμών κώδικα οι οποίες συναντώνται, σε μικρότερο ή μεγαλύτερο βαθμό, σε προϊόντα λογισμικού, και η συσσώρευσή τους μπορεί να οδηγήσει σε δυσκολίες κατά τη συντήρηση του προϊόντος. Για το λόγο αυτό, έχουν αναπτυχθεί εργαλεία, είτε αυτόνομα, είτε plug-ins για κάποιο IDE, όπως αυτά που παρουσιάστηκαν στην Ενότητα 3, με σκοπό την παροχή ενός αυτόματου τρόπου εντοπισμού των οσμών, ώστε να διευκολύνονται οι προγραμματιστές και να επενδύουν περισσότερο χρόνο για την απομάκρυνση τους. Τα εργαλεία αυτά λοιπόν, υλοποιούν διαφορετικές τεχνικές εντοπισμού, όπως αυτές που παρουσιάστηκαν στην Ενότητα 2.1.3, προκειμένου να επιτύχουν τον εντοπισμό αυτό. Φυσικά, η απόδοση του κάθε εργαλείου στην επίτευξη εντοπισμού οσμών στα σωστά σημεία, είναι άμεσα συνδεδεμένη με την τεχνική που αξιοποιεί καθώς και με τον τρόπο που αυτή έχει υλοποιηθεί. Για τον λόγο αυτό, όπως έχει αναφερθεί, είναι ιδιαίτερα σημαντική η διενέργεια πειραμάτων με σκοπό την αξιολόγηση των υπαρχόντων εργαλείων και της μεταξύ τους σύγκρισης για να μετρηθεί η αποτελεσματικότητα τους. Στα πλαίσια της ανάγκης αυτής λοιπόν, αναπτύχθηκε και χρησιμοποιήθηκε στην παρούσα εργασία ένα Eclipse plug-in, το SmellDetectorMerger, το οποίο ενσωματώνει κάποια από τα πιο γνωστά εργαλεία εντοπισμού οσμών. Στόχος της δημιουργίας του συγκεκριμένου plug-in, είναι η εύκολη πρόσβαση και αξιοποίηση των ενσωματωμένων εργαλείων για την αυτοματοποίηση της εκτέλεσης τους, την εξαγωγή των αποτελεσμάτων τους και την άμεση σύγκριση τους. Ακόμη, δίνεται η δυνατότητα επιλογής των εργαλείων που θα χρησιμοποιηθούν, ώστε ο χρήστης να μπορεί να ενεργοποιήσει μόνο αυτά που τον ενδιαφέρουν. Έτσι, μπορούν να πραγματοποιηθούν μελέτες ευκολότερα, καθώς οι ερευνητές δεν χρειάζεται να αναζητούν το εκάστοτε εργαλείο, να το εγκαθιστούν στο σύστημα τους, να εξάγουν τα αποτελέσματα σε κάποιο αρχείο και στη συνέχεια να τα συγχωνεύουν ώστε να πραγματοποιούν τελικά τις μεταξύ τους συγκρίσεις.

Για τον πειραματισμό με το SmellDetectorMerger, πραγματοποιήθηκε μια μελέτη περίπτωσης, στην οποία χρησιμοποιήθηκε ένα έργο ανοικτού κώδικα, το Apache HttpComponents Core, για την εύρεση οσμών από τα επιμέρους εργαλεία και τη σύγκριση μεταξύ τους. Πιο συγκεκριμένα, για τον υπολογισμό της συμφωνίας μεταξύ των εργαλείων, χρησιμοποιήθηκε ο όρος του positive specific agreement. Αναλυτικά η

διενέργεια και τα αποτελέσματα της μελέτης αυτής παρουσιάζονται στην Ενότητα 4.2.3. Με βάση αυτά, το συμπέρασμα που προκύπτει είναι ότι τα εργαλεία εντοπίζουν διαφορετικό πλήθος οσμών, ακόμα και σε περιπτώσεις που αφορούν τον ίδιο τύπο οσμής, αλλά και η συμφωνία μεταξύ τους στα ευρήματα είναι πολύ χαμηλή στην πλειοψηφία των περιπτώσεων. Το γεγονός αυτό, δημιουργεί ερωτήματα σχετικά με τις τεχνικές που είναι υλοποιημένες στο εκάστοτε εργαλείο, αλλά και στον τρόπο με τον οποίο αυτές έχουν υλοποιηθεί, καθώς μεγάλες διαφορές στο πλήθος αλλά και μικρή συμφωνία παρατηρούνται και σε περιπτώσεις που οι τεχνικές που χρησιμοποιούνται είναι ίδιες. Αιτία αυτού μπορεί να είναι ότι ο ορισμός της εκάστοτε οσμής δεν είναι απόλυτος ή οι τεχνικές δεν είναι σωστά υλοποιημένες. Για παράδειγμα, στην περίπτωση της εύρεσης με τη χρήση μετρικών, απαιτείται πειραματισμός προκειμένου να βρεθούν οι σωστές τιμές των τελευταίων, οι οποίες θα επιστρέψουν τα πιο στοχευμένα αποτελέσματα.

5.2 Όρια και περιορισμοί της έρευνας

Κατά τη διάρκεια της παρούσας μελέτης, υπήρξαν κάποιοι περιορισμοί οι οποίοι ενδεχομένως επηρέασαν τα αποτελέσματα που εξήχθησαν. Αρχικά, ένας πολύ σημαντικός περιορισμός ήταν ότι κάποια από τα εργαλεία που ενσωματώθηκαν επιτρέπουν την παραμετροποίησή τους σε κάποιο βαθμό ως προς τις τεχνικές εντοπισμού. Η παραμετροποίηση αυτή δεν επιτρέπεται αυτή τη στιγμή μέσω του SmellDetectorMerger, επομένως δεν αξιοποιούνται στο έπακρο οι λειτουργίες που προσφέρονται από τα ενσωματωμένα εργαλεία. Ένας ακόμη περιορισμός, είναι η χρήση ενός μόνο έργου λογισμικού κατά τον πειραματισμό. Όπως έχει αναφερθεί, στη μελέτη περίπτωσης που διεξάχθηκε χρησιμοποιήθηκε ένα έργο λογισμικού, επομένως και τα αποτελέσματα που εξήχθησαν βασίζονταν στα ευρήματα των εργαλείων από αυτό εξ ολοκλήρου.

5.3 Μελλοντικές Επεκτάσεις

Ως μελλοντική επέκταση της παρούσας εργασίας, μπορεί να οριστεί η επίλυση των περιορισμών οι οποίοι υπήρξαν. Πιο συγκεκριμένα, θα είχε ιδιαίτερο ενδιαφέρον να επεκταθεί η λειτουργία του SmellDetectorMerger, με σκοπό να παρέχεται η δυνατότητα ορισμού των παραμέτρων σε όσα εργαλεία το επιτρέπουν. Έτσι, θα δίνεται η δυνατότητα να μην παραμένει η εκτέλεση του κάθε εργαλείου στις προεπιλεγμένες παραμέτρους, αλλά να τροποποιείται, ώστε να τη φέρει ο χρήστης στα μέτρα του, όπως ακριβώς θα

έκανε, εφόσον επιτρεπόταν, και κατά τη χρήση του ίδιου του εργαλείου. Επίσης, μεγάλη σημασία θα είχε και η εφαρμογή του SmellDetectorMerger σε περισσότερα έργα λογισμικού για την εξαγωγή αποτελεσμάτων, ώστε να υπάρχει σύγκριση μεταξύ τους σχετικά με τον βαθμό συμφωνίας των επιμέρους εργαλείων. Με αυτόν τον τρόπο, μπορούν να εξαχθούν πιο ασφαλή συμπεράσματα σχετικά με τα επίπεδα συμφωνίας. Τέλος, ένα ακόμη πολύ σημαντικό πείραμα θα αποτελούσε η εφαρμογή του εργαλείου σε κάποιο έργο λογισμικού, στο οποίο με κάποιον τρόπο έχει γίνει γνωστό νωρίτερα ποια στοιχεία του περιέχουν οσμές καθώς και ο τύπος των οσμών αυτών. Σε ένα τέτοιο έργο, εκτός από τις συγκρίσεις σχετικά με τη συμφωνία των εργαλείων, μπορούν να εξαχθούν συμπεράσματα και σχετικά με το πόσο αποδοτικό είναι το κάθε εργαλείο, καθώς θα είναι ήδη γνωστό αν μια συγκεκριμένη οσμή η οποία εντοπίστηκε από το SmellDetectorMerger αποτελεί πραγματικά οσμή ή όχι.

Βιβλιογραφία

Alves, P., Santana, D., & Figueiredo, E. (2012). ConcernReCS: finding code smells in software aspectization. *2012 34th International Conference on Software Engineering (ICSE)*, σσ. 1463-1464.

Arnold, K., Gosling, J., & Holmes, D. (2005). *The Java™ Programming Language, Fourth Edition.* Addison Wesley Professional.

Batch file. (2022). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Batch_file [Ανακτήθηκε στις: 31/01/2022]

Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., & Palomba, F. (2015). An experimental investigation on the innate relationship between quality and refactoring. *Journal of Systems and Software, 107*, σσ. 1-14.

CheckStyle. (2022). Ανάκτηση από CheckStyle: https://checkstyle.sourceforge.io/ [Ανακτήθηκε στις: 31/01/2022]

Comma-separated values. (2022). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Comma-separated_values [Ανακτήθηκε στις: 31/01/2022]

Danphitsanuphan, P., & Suwantada, T. (2012). Code Smell Detecting Tool and Code Smell-Structure Bug Relationship. *2012 Spring Congress on Engineering and Technology*, σσ. 1-5.

des Rivie` res, J., & Wiegand, J. (2004). Eclipse: A platform for integrating development tools. *IBM Systems Journal, 43*(2), σσ. 371-383.

Fernandes, E., Oliveira, J., Vale, G., Paiva, T., & Figueiredo, E. (2016). A Review-based Comparative Study of Bad Smell Detection Tools. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, σσ. 1-12.

Finding duplicated code with CPD. (2022). Ανάκτηση από CPD: https://pmd.github.io/pmd-6.41.0/pmd_userdocs_cpd.html [Ανακτήθηκε στις: 31/01/2022]

Fokaefs, M., Tsantalis, N., & Chatzigeorgiou, A. (2007). Jdeodorant: Identification and removal of feature envy bad smells. *ieee international conference on software maintenance*.

Fokaefs, M., Tsantalis, N., Stroulia, E., & Chatzigeorgiou, A. (2011). JDeodorant: identification and application of extract class refactorings. *33rd International Conference on Software Engineering (ICSE)*.

Fowler, M. (1999). *Refactoring: improving the design of existing code.* Addison-Wesley Professional.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Massachusetts: Addison-Wesley.

Git. (2022). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Git [Ανακτήθηκε στις: 31/01/2022]

GitHub. (2022). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/GitHub [Ανακτήθηκε στις: 31/01/2022]

Griffith, I., Wahl, S., & Izurieta, C. (2011). TrueRefactor: An automated refactoring tool to improve legacy system and application comprehensibility. *24th International Conference on Computer Applications in Industry and Engineering*.

Hall, T., Zhang, M., Bowes, D., & Sun, Y. (2014). Some code smells have a significant but small effect on faults. *ACM Transactions on Software Engineering and Methodology (TOSEM), 23*(4), σσ. 1-39.

Hripcsak, G., & Rothschild, A. (2005). Agreement, the F-Measure, and Reliability in Information Retrieval. *Journal of the American medical informatics association, 12*(3), pp. 296-298.

JAR (file format). (2022). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/JAR_(file_format) [Ανακτήθηκε στις: 31/01/2022]

JDeodorant. (2022). Ανάκτηση από GitHub: https://github.com/tsantalis/JDeodorant [Ανακτήθηκε στις: 31/01/2022]

Kirk, D., Roper, M., & Wood, M. (2007). A heuristic-based approach to code-smell detection. σσ. 54-56.

Lanza, M., & Marinescu, R. (2007). *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems.* Springer Science & Business Media.

Mathur, N. (2011). Java Smell Detector. *Java Smell Detector*.

Menshawy, R., Yousef, A., & Salem, A. (2021). Code Smells and Detection Techniques: A Survey. *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, σσ. 78-83.

Murphy-Hill, E., & P. Black, A. (2010). An interactive ambient visualization for code smells. *Proceedings of the 5th international symposium on Software visualization*, σσ. 5-14.

Nongpong, K. (2012). *Integrating "Code Smells" Detection with Refactoring Tool Support.* ProQuest Dissertations Publishing.

Organic. (2022). Ανάκτηση από Github: https://github.com/opus-research/organic [Ανακτήθηκε στις: 31/01/2022]

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., & Poshyvanyk, D. (2013). Detecting bad smells in source code using change history information. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, σσ. 268-278.

Palomba, F., Panichella, A., De Lucia, A., Oliveto, R., & Zaidman, A. (2016). A textual-based technique for smell detection. *2016 IEEE 24th international conference on program comprehension*, σσ. 1-10.

PMD. (2022). Ανάκτηση από PMD: https://pmd.github.io/ [Ανακτήθηκε στις: 31/01/2022]

Roperia, N. (2009). *JSmell: A Bad Smell detection tool for Java systems.* California.

SISSy. (2022). Ανάκτηση από Sourceforge: https://sourceforge.net/projects/sissy/ [Ανακτήθηκε στις: 31/01/2022]

Spinellis, D. (2005). Version control systems. *IEEE Software, 22*(5), σσ. 108-109.

The Apache Software Foundation. (2022). Retrieved from Apache HttpComponents: https://hc.apache.org/ [Ανακτήθηκε στις: 31/01/2022]

Tsantalis, N., & Chatzigeorgiou, A. (2009). Identification of Extract Method Refactoring Opportunities. *13th European Conference on Software Maintenance and Reengineering*.

Tsantalis, N., Chaikalis, T., & Chatzigeorgiou, A. (2008). JDeodorant: Identification and removal of type-checking bad smells. *12th European conference on software maintenance and reengineering.*

Unified Modeling Language. (2022). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Unified_Modeling_Language [Ανακτήθηκε στις: 31/01/2022]

Vidal, S., Vazquez, H., Dıaz-Pace, J., Marcos, C., Garcia, A., & Oizumi, W. (2015). JSpIRIT: A Flexible Tool for the Analysis of Code. *2015 34th International Conference of the Chilean Computer Science Society (SCCC)*, σσ. 1-6.

Wettel, R., & Marinescu, R. (2005). Archeology of code duplication: recovering duplication chains from small duplication fragments. *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*, σ. 8.

XML. (2022). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/XML [Ανακτήθηκε στις: 31/01/2022]