ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εντοπισμός διαδεδομένων προτύπων ανακατασκευής (Refactoring Patterns) σε έργα λογισμικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΜΠΑΛΤΖΗΣ ΔΗΜΗΤΡΙΟΣ

Επιβλέπων: Μπίμπη Σταματία

Αναπληρώτρια Καθηγήτρια

ΚΟΖΑΝΗ/ΙΟΥΛΙΟΣ/2024

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ

HELLENIC DEMOCRACY
UNIVERSITY OF WESTERN MACEDONIA
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL
& COMPUTER ENGINEERING

Retrieving Software Refactoring Patterns

THESIS

BALTZIS DIMITRIOS

SUPERVISOR: Bibi Stamatia

Associate Professor

KOZANI/JULY/2024

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "**Εντοπισμός διαδεδομένων προτύπων ανακατασκευής (Refactoring Patterns) σε έργα λογισμικού**" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κα. **Μπίμπη Σταματία** αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ

Περίληψη

Οι τεχνικές του Refactoring αποτελούν απαραίτητες μέθοδοι που εφαρμόζονται για την αντιμετώπιση πιθανών προβλημάτων ή αδυναμιών στον σχεδιασμό ή/και στη δομή του κώδικα. Τα προβλήματα αυτά, συχνά περιγράφονται από την έννοια του «Code Smells». Η διαδικασία του Refactoring είναι ιδιαίτερα σημαντική αφού έχει ώς σκοπό να επιτύχει πολλαπλά οφέλη, όπως είναι η συντηρησιμότητα, η αύξηση της απόδοσης, η μείωση της πολυπλοκότητας του κώδικα αλλά και η ευκολότερη ενσωμάτωση νέων λειτουργιών, σε υπάρχοντα λογισμικά, χωρίς να επηρεάζεται η λειτουργικότητά του. Αντικείμενο της συγκεκριμένης διπλωματικής αποτελεί ο εντοπισμός διαδεδομένων προτύπων ανακατασκευής (Refactoring Patterns) σε έργα λογισμικού γραμμένα σε γλώσσα Python. Στα πλαίσια την εργασίας αναπτύχθηκε λογισμικό, το οποίο μετά την ανάλυση του κώδικα δύο εκδόσεων οποιουδήποτε έργου Python, εντοπίζει διάφορους τύπους Refactor. Η εύρεση των Refactors πραγματοποιείται μέσω αλγορίθμων που δημιουργήθηκαν για τον σκοπό αυτό. Ως αποτέλεσμα, η παρούσα έκδοση του λογισμικού, είναι σε θέση να εντοπίζει 24 διαφορετικούς τύπους Refactor. Στη συνέχεια, πραγματοποιήθηκε η εκτέλεση του λογισμικού σε 209 διαφορετικά έργα μεταξύ 5368 εκδόσεων, όπου εντοπίστηκαν συνολικά 70422 διαφορετικά Refactors. Τέλος, χρησιμοποιώντας τα Refactors που εντοπίστηκαν, δημιουργήθηκαν στατιστικά που αφορούν την συχνότητα εμφάνισης εφαρμογής των τύπων και κατηγοριών Refactors κατά την εξέλιξη των διαφόρων έργων.

Λέξεις Κλειδιά

Refactoring, Code Smells, Πρότυπα ανακατασκευής, Συντηρησιμότητα, Μείωση πολυπλοκότητας, Python

Abstract

Refactoring techniques are essential methods used to deal with potential issues or weaknesses in the design or/and structure of code, which are usually indicated by certain patterns in a codebase, called “code smells”. Refactoring code is a significant process that aims to enhance the structure and design of existing code without altering its overall behavior and at the same time to achieve valuable benefits such as maintainability, increased performance, reduced complexity and easier integration of new functions into existing software. The purpose of this thesis is to identify widespread Refactoring Patterns in software projects written in Python. Within the scope of the study, a software tool was developed which identifies several Refactor Patterns, after analyzing the code of two different versions of any Python project. The finding of Refactors is performed through algorithms created for this purpose. As a result, the current version of the software is able to detect 24 different types of Refactor. Furthermore, the software was executed on 209 different projects among 5368 versions, in which a total of 70422 different Refactors were identified. Eventually, using the identified Refactors, specific statistics were generated; regarding the frequency of application of Refactors types and categories during the progress of various projects.

Keywords

Refactoring patterns, Code Smells, Sustainability, Complexity reduction, Python

Ευχαριστίες

Πρωτίστως, θα ήθελα να ευχαριστήσω ιδιαίτερα την επιβλέπουσα αναπληρώτρια καθηγήτρια, κα Μπίμπη Σταματία, για την καθοδήγηση και το χρόνο που μου διέθεσε, παρέχοντάς μου την αμέριστη συμπαράστασή της για την ολοκλήρωση της διπλωματικής μου εργασίας.

Ακόμη, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στο σύνολο των καθηγητών για τις γνώσεις και τις συμβουλές που μου πρόσφεραν στα χρόνια σπουδών μου.

Τέλος, οφείλω να ευχαριστήσω την οικογένειά μου για τη στήριξή της και ιδιαίτερα την αδερφή μου που με εμψύχωνε διαρκώς και με βοήθησε με την εμπειρία και τις παρατηρήσεις της στην ολοκλήρωση της διπλωματικής μου εργασίας.

ΚΟΖΑΝΗ/ ΙΟΥΛΙΟΣ/2024

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1: Εισαγωγή

Στον κόσμο της ανάπτυξης λογισμικού, η ύπαρξη κακού κώδικα αποτελεί συχνό φαινόμενο και προκύπτει από διάφορους παράγοντες. Η έλλειψη καλού σχεδιασμού και προγραμματισμού, όπως και η βιασύνη για γρήγορη παράδοση, οδηγούν συχνά σε πρόχειρες και μη βέλτιστες λύσεις. Ακόμη συνθήκες όπως είναι η ανεπάρκεια γνώσεων και εμπειρίας των προγραμματιστών, σε συνδυασμό με περιορισμένους πόρους και αυστηρά χρονοδιαγράμματα είναι υπεύθυνες για την συχνότητα εμφάνισης του κακού κώδικα αυτού. Επίσης, η απουσία κατάλληλων ελέγχων ποιότητας, όπως οι δοκιμές και οι αναθεωρήσεις κώδικα, οι αλλαγές στο προσωπικό, η κακή επικοινωνία, καθώς και η έλλειψη ή η παρωχημένη τεκμηρίωση, δυσκολεύουν στη συντήρηση του κώδικα δυσχεραίνοντας περαιτέρω την όλη κατάσταση.

Ένα μέρος των προβλημάτων του κακού κώδικα εντάσσονται στην έννοια των «Code Smells». Η έννοια αυτή[9] αναφέρεται σε τμήματα κώδικα που υποδηλώνουν πιθανά προβλήματα, χωρίς όμως αυτά να αποτελούν άμεσα σφάλματα. Στην έννοια του «Code Smells» περιέχονται περιπτώσεις όπως: υπερβολικά μεγάλα ή πολύπλοκα αρχεία και μέθοδοι, επαναλαμβανόμενα μοτίβα κώδικα και δύσχρηστες ή μη αποδοτικές δομές δεδομένων. Λύση των παραπάνω προβλημάτων αποτελεί η συγγραφή καθαρού κώδικα. Η αξία του καθαρού κώδικα θεωρείται δεδομένη καθώς η έλλειψη αυτού επέχει πολλές αρνητικές επιπτώσεις. Κάθε τροποποίηση, επέκταση ή διόρθωση ενός υπάρχοντος κακογραμμένου κώδικα, μπορεί να αποβεί ιδιαίτερα απαιτητική, προβληματική και χρονοβόρα διαδικασία. Ένας προβληματικός κώδικας μπορεί να οδηγήσει σε λειτουργικά σφάλματα και ανεπιθύμητα αποτελέσματα, αποθαρρύνοντας τους προγραμματιστές από την κατανόηση και την επέκτασή του. Κάτι τέτοιο είναι ικανό να προκαλέσει σημαντικές καθυστερήσεις στην ανάπτυξη-λόγω υψηλών χρόνων αποσφαλμάτωσης και επιδιόρθωσης και επιπλέον αυξάνει το κόστος του ίδιου έργου. Γίνεται σαφές λοιπόν, ότι ένας καλά τεκμηριωμένος, σαφής και ποιοτικός κώδικας, είναι απαραίτητος καθώς διευκολύνει άμεσα την μετατροπή, συντήρηση και επέκτασή του. Επίσης έχει ως άμεσα αποτελέσματα την αύξηση την παραγωγικότητας και αποτελεσματικότητας των προγραμματιστών, τη μείωση του χρόνου εκτέλεσης των όποιων μετατροπών της δομής του κώδικα, τη μείωση του κόστους συντήρησης, και τη μείωση των πιθανοτήτων σφάλματος ή δυσλειτουργιών.

Ωστόσο, η συγγραφή καθαρού κώδικα , όπως αναφέρει και ο Robert C. Martin στο βιβλίο «Clean Code - A Handbook of Agile Software Craftsmanship» [6], μπορεί να χαρακτηριστεί και σαν μια μορφή Τέχνης. Όπως και ο ίδιος έγραψε «οι περισσότεροι μπορούν να αναγνωρίσουν αν ένας πίνακας είναι καλός και αποτελεί καλή Τέχνη αυτό όμως δεν τους κάνει ικανούς και να ζωγραφίσουν πίνακες που να είναι όντως Τέχνη». Το ίδιο ακριβώς συμβαίνει και με την συγγραφή καθαρού κώδικα. Η συγγραφή καθαρού κώδικα μπορεί να επιτευχθεί με την εφαρμογή των δοκιμασμένων τεχνικών και πρακτικών του Refactoring. Ο όρος Refactoring πρωτοαναφέρθηκε από τον Martin Fowler στο βιβλίο που εκδόθηκε το 1999, με τίτλο "Refactoring: Improving the Design of Existing Code."[1] (Βελτίωση της σχεδίασης του υπάρχοντος κώδικα). Σε αυτό το βιβλίο, ο Fowler παρέχει πρακτικά παραδείγματα και τεχνικές για τη βελτίωση του σχεδιασμού του κώδικα. Το βιβλίο είναι αποτέλεσμα συλλογικού κόπου και έρευνας από μια ομάδα ανθρώπων, στους

οποίους μάλιστα γίνεται μνεία από τον ίδιο τον Fowler στο βιβλίο του. Ενδεικτικά αναφέρονται: Kent Beck, Ward Cunningham, Ralph Johnson κ.α. Πιο συγκεκριμένα, το βιβλίο περιέχει αναλυτικά τις αρχές και τις τεχνικές του Refactoring και παρέχει πολλά παραδείγματα και οδηγίες για την εφαρμογή αυτών. Σημαντικό είναι να τονιστεί ότι τα παραδείγματα που περιέχει είναι γραμμένα σε γλώσσα Java. Στόχος του βιβλίου, αποτελεί η επιτυχημένη και ορθή εφαρμογή της διαδικασίας των Refactoring, ώστε ο κώδικας κάθε φορά να βελτιώνεται μεθοδικά, χωρίς να εισάγονται τυχόν σφάλματα στη δομή του κώδικα.

Τα οφέλη της εφαρμογής των τεχνικών αυτών έχουν απασχολήσει ιδιαίτερα την επιστημονική κοινότητα. Μία μελέτη[7] κάνει αναφορά στην βελτίωση της συντηρησιμότητας, της αναγνωσιμότητας, της απόδοσης και της μείωσης του μεγέθους του κώδικα. Ακόμη κάνει λόγο για προτερήματα όπως είναι η μεγαλύτερη δυνατότητα ελέγχου και η ευκολότερη προσθήκη νέων λειτουργιών. Τα παραπάνω οφέλη έχουν ως αποτέλεσμα την μείωση του κόστους συντήρησης αλλά και ανάπτυξης του λογισμικού. Η μελέτη αυτή πραγματοποίησε ημιδομημένες συνεντεύξεις σε επαγγελματίες μηχανικούς λογισμικού καθώς και ανάλυση στο ιστορικό των εκδόσεων των Windows 7 διαπιστώνοντας τα οφέλη που προαναφέρθηκαν. Μια άλλη μελέτη[8] έδειξε ότι μετά την εφαρμογή των τεχνικών του Refactoring, ο αριθμός των διορθωμένων σφαλμάτων αυξάνεται ενώ ο χρόνος που απαιτείται για τη διόρθωση αυτών μειώνεται. Στην έρευνα αυτή πραγματοποιήθηκε μελέτη των Refactorings στο επίπεδο του API και στις διορθώσεις σφαλμάτων σε 3 ανοιχτού τύπου λογισμικά σε μεγάλο πλήθος αναθεωρημένων εκδόσεων.

Τα παραπάνω οφέλη έχουν ως αποτέλεσμα όλο και περισσότεροι προγραμματιστές να μπαίνουν στην διαδικασία των Refactorings ώστε επωφεληθούν από αυτά. Ωστόσο, η λανθασμένη εφαρμογή αυτών επιφέρει αρκετούς κινδύνους. Η λανθασμένη εφαρμογή των Refactors επιφέρει προβλήματα όπως η απώλεια λειτουργικότητας, η αύξηση του χρόνου ανάπτυξης, τους κινδύνους ασφάλειας και το αυξημένο κόστος. Έτσι η ορθή εφαρμογή των τεχνικών του Refactoring είναι ιδιαίτερα σημαντική. Λύση για την σωστή εφαρμογή αποτελεί η βαθιά κατανόηση και η πιστή εφαρμογή των τεχνικών αυτών. Επιπλέον η αναγνώριση των εφαρμοσμένων Refactors και η επικύρωση της σωστής λειτουργίας αυτών, αποτελούν την κύρια διέξοδο αποφυγής των κινδύνων που αναφέρθηκαν προηγουμένως.

Για τον λόγο αυτό, η ανάγκη ύπαρξης εργαλείων εντοπισμού Refactoring έχει προκαλέσει το ενδιαφέρον πολλών ερευνητών.Έτσι αρκετοί ερευνητές ασχοληθήκαν με τη δημιουργία τέτοιων εργαλείων. Τα περισσότερα από τα εργαλεία αυτά, απευθύνονται σε λογισμικά γραμμένα σε γλώσσες όπως η Java. Αυτό συμβαίνει διότι η Java χαρακτηρίζεται από αυστηρό συντακτικό και υποχρεωτική δήλωση μεταβλητών, γεγονός που την καθιστά αυστηρά δομημένη. Τα χαρακτηριστικά της έχουν σαν αποτέλεσμα η ανάλυση τμημάτων του κώδικά της να είναι αρκετά ξεκάθαρη, διευκολύνοντας την διαδικασία εύρεσης των διαφόρων εφαρμοσμένων Refactors. Αντιθέτως, τα υπάρχοντα εργαλεία που αφορούν την γλώσσα Python είναι αρκετά περιορισμένα. Η γλώσσα Python είναι μια υψηλού επιπέδου γλώσσα προγραμματισμού που διακρίνεται για τη δυναμική της φύση, την ευελιξία και την έλλειψη αυστηρών κανόνων και δομής. Τα χαρακτηριστικά της δυσχεραίνουν την ανάλυση του κώδικα καθιστώντας τον εντοπισμό εφαρμοσμένων

Refactors ιδιαίτερα δύσκολο. Ωστόσο η γλώσσα Python είναι μια από τις πιο δημοφιλείς γλώσσες της εποχής αφού βρίσκει εφαρμογές σε πολλούς τομείς όπως είναι τεχνητή νοημοσύνη, η μηχανική μάθηση, η ανάλυση δεδομένων κ.α.. Η απουσία πλήθους εργαλείων εντοπισμού εφαρμοσμένων Refactorings στη γλώσσα αυτή, η δημοφιλία αυτής αλλά και η πρόκληση που προκύπτει από την δυσκολία που παρουσιάζει η ανάλυση του κώδικά τους, καθιστά την ενασχόληση με τη δημιουργία ενός τέτοιου εργαλείου ιδιαίτερα ενδιαφέρουσα. Οι παραπάνω λόγοι αποτέλεσαν την επιλογή της ενασχόλησης του συγκεκριμένου τομέα για την μελέτη αυτή.

1.1 Αντικείμενο της διπλωματικής

Η διπλωματική εργασία αφορά τον εντοπισμό διαδεδομένων προτύπων ανακατασκευής (Refactoring Patterns) σε έργα λογισμικού. Πιο συγκεκριμένα η μελέτη εστιάζει σε λογισμικά που είναι γραμμένα σε γλώσσα Python μιας και για την συγκεκριμένη γλώσσα, τα υπάρχοντα εργαλεία εύρεσης Refactor είναι αρκετά περιορισμένα.

Στόχοι της παρούσας εργασίας αποτελούν:

- Η δημιουργία λογισμικού το οποίο θα επιτρέπει τον αυτοματοποιημένο εντοπισμό διαφόρων τύπων Refactor μεταξύ δύο εκδόσεων οποιουδήποτε έργου Python.
- Η εξαγωγή στοιχείων και στατιστικών που αφορούν την συχνότητα εμφάνισης εφαρμογής των τύπων και κατηγοριών Refactors κατά την εξέλιξη των διαφόρων έργων.

Στο πλαίσιο της εργασίας, το λογισμικό που δημιουργήθηκε, είναι ικανό να εντοπίζει 24 διαφορετικούς τύπους Refactors. Η εκτέλεση του λογισμικού απέφερε 70422 διαφορετικά Refactors έχοντας σαν είσοδο 209 ανοιχτού τύπου λογισμικά μεταξύ 5368 εκδόσεων που ελήφθησαν από το αποθετήριο του Github[1].

1.2 Οργάνωση του τόμου

Η διπλωματική εργασία ακολουθεί την παρακάτω δομή:

- Η Ενότητα 2 περιέχει το θεωρητικό υπόβαθρο που αφορά τα Refactors τους τύπους αυτών αλλά και τα οφέλη που παρουσιάζουν.
- Η Ενότητα 3 περιγράφει τον σκοπό και τον τρόπο λειτουργίας του λογισμικού που αναπτύχθηκε αλλά και τους τρόπους-βήματα που ακολουθούνται για την εύρεση των διαφόρων τύπων Refactor.
- Η Ενότητα 4 αφορά την επιλογή των έργων και τον τρόπο εκτέλεσης του λογισμικού.
- Η Ενότητα 5 παρουσιάζει τα αποτελέσματα του λογισμικού που προέκυψαν μετά την εκτέλεσή του στα έργα που επιλέχθηκαν όπως και τα συμπεράσματα που απορρέουν από αυτά .

[1] https://github.com/

- Η Ενότητα 6 αναφέρει τους περιορισμούς και παραδοχές που έγιναν, τις μελλοντικές επεκτάσεις της μελέτης καθώς και τη σημασία του έργου για την επιστημονική κοινότητα.
- Τέλος, στα παραρτήματα περιέχεται η λειτουργία των βοηθητικών script που δημιουργήθηκαν πέρα του βασικού λογισμικού.

Κεφάλαιο 2: Θεωρητικό υπόβαθρο

Ως Refactoring ορίζεται η τεχνική επανεγγραφής τμημάτων κώδικα με σκοπό την βελτίωση της δομής και της ποιότητάς του χωρίς όμως αυτό να επηρεάζει την λειτουργία και την αποτελεσματικότητά του. Μελέτες[7][8] έχουν δείξει πως η εφαρμογή των τεχνικών του Refactoring επιφέρει οφέλη όπως είναι:

- Η βελτίωση της ποιότητα και η συντηρησιμότητας του κώδικα
- Η μείωση του τεχνικού χρέους
- Η ενίσχυσης της προσαρμοστικότητας σε αλλαγές
- Η βελτίωση της απόδοσης
- Η ενίσχυση των συνεργασιών
- Η μακροπρόθεσμη εξοικονόμηση κόστους
- Η επιδιόρθωση πιθανών σφαλμάτων

Οι τεχνικές του Refactoring κατατάσσονται στις παρακάτω βασικές κατηγορίες[10].

Για κάθε κατηγορία που ακολουθεί, γίνεται αναφορά στα οφέλη που αποφέρει, στους τύπους των Refactor που την αποτελούν αλλά και δίνονται ενδεικτικά παραδείγματα εφαρμογής των Refactors αυτών.

Οι κατηγορίες αυτές είναι:

- **Σύνθεση μεθόδων (Composing Methods):** Ο σκοπός της κατηγορίας αυτής είναι η αφαίρεση επαναλήψεων του κώδικα και η προετοιμασία του για μελλοντικές βελτιώσεις.
  Στην κατηγορία αυτή ανήκουν οι παρακάτω τύποι Refactors:
  - Extract Variable
  - Extract Method
  - Inline Method
  - Inline Temp
  - Replace Temp with Query
  - Split Temporary Variable
  - Remove Assignments to Parameters
  - Replace Method with Method Object
  - Substitute Algorithm

Before

```
return order.quantity * order.itemPrice - max(0,
order.quantity - 500) * order.item_price * 0.05 +
min(base_price * 0.1, 100)
```

After

```
base_price = order.quantity *
order.item_price
quantity_discount = max(0,
order.quantity - 500) *
order.item_price * 0.05
shipping = min(base_price * 0.1, 100)
return base_price - quantity_discount
+ shipping
```

Εικόνα 1: ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR EXTRACT VARIABLE

Before

```
def printOwing(self):
    self.printBanner()

    # print details
    print("name:", self.name)
    print("amount:", self.getOutstanding())
```

After

```
def printOwing(self):
    self.printBanner()
    self.printDetails(self.getOutstanding())

def printDetails(self, outstanding):
    print("name:", self.name)
    print("amount:", outstanding)
```

Εικόνα 2 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR EXTRACT METHOD

- **Μετακίνηση "στοιχείων" μεταξύ αντικειμένων (Moving Features between Objects):** Η εφαρμογή των Refactors της κατηγορίας αυτής, επιτυγχάνει τη βελτίωση της δομής, της συντηρησιμότητας και της επεκτασιμότητας του κώδικα. Η κατηγορία αυτή περιέχει τους εξής τύπους Refactor:
    - Move Method
    - Move Field
    - Extract Class
    - Inline Class
    - Hide Delegate
    - Remove Middle Man
    - Introduce Foreign Method
    - Introduce Local Extension

Before

```
class A:
    def method_A1(self):
        pass
    def method_A2(self):
        pass
class B:
    def method_B1(self):
        pass
    def method_B2(self):
        pass
```

After

```
class A:
    def method_A1(self):
        pass
class B:
    def method_B1(self):
        pass
    def method_B2(self):
        pass
    def method_A2(self):
        pass
```

Εικόνα 3 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR MOVE METHOD

Before

```
class A:
    def __init__(self,a,b):
        self.a=a
        self.b=b
class B:
    def __init__(self,c,d):
        self.c=c
        self.d=d
```

After

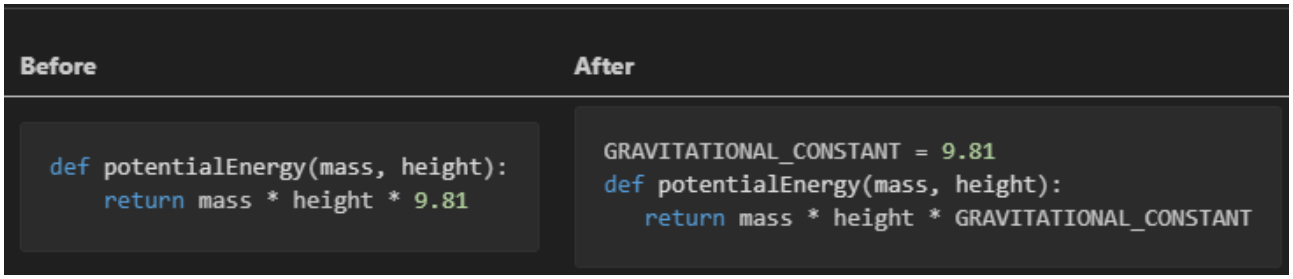
```
class A:
    def __init__(self,a):
        self.a=a

class B:
    def __init__(self,b,c,d):
        self.b=b
        self.c=c
        self.d=d
```

Εικόνα 4 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR MOVE FIELD

- **Οργάνωση δεδομένων (Organizing Data) :** Η κατηγορία αυτή έχει ως στόχο τη βελτίωση της διαχείρισης των δεδομένων με σκοπό την επίτευξη φορητότητας και την επαναχρησιμοποίηση των κλάσεων.
  Σε αυτήν την κατηγορία εμπίπτουν οι παρακάτω τύποι Refactors:
  - Change Value to Reference
  - Change Reference to Value
  - Duplicate Observed Data
  - Self Encapsulate Field
  - Replace Data Value with Object
  - Replace Array with Object
  - Change Unidirectional Association to Bidirectional

- Change Bidirectional Association to Unidirectional
- Encapsulate Field
- Encapsulate Collection
- Replace Magic Number with Symbolic Constant
- Replace Type Code with Class
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Subclass with Fields

Εικόνα 5 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR REPLACE ARRAY WITH OBJECT



Εικόνα 6 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR REPLACE MAGIC NUMBER WITH SYMBOLIC CONSTANT

- **Απλοποίηση συνθηκών εκφράσεων (Simplifying Conditional Expressions) :** Η συγκεκριμένη κατηγορία επιφέρει οφέλη όπως είναι η συντηρησιμότητα, η κατανοητότητα και η αποτελεσματικότητα του κώδικα.
  Στην κατηγορία αυτή ανήκουν οι παρακάτω τύποι Refactors:
  - Consolidate Conditional Expression
  - Consolidate Duplicate Conditional Fragments
  - Decompose Conditional
  - Replace Conditional with Polymorphism
  - Remove Control Flag
  - Replace Nested Conditional with Guard Clauses

- Introduce Null Object
- Introduce Assertion

Before

```
if date.before(SUMMER_START) or date.after(SUMMER_END):
    charge = quantity * winterRate + winterServiceCharge
else:
    charge = quantity * summerRate
```

After

```
if isSummer(date):
    charge = summerCharge(quantity)
else:
    charge = winterCharge(quantity)
```

Εικόνα 7 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR DECOMPOSE CONDITIONAL

Before

```
def getPayAmount(self):
    if self.isDead:
        result = deadAmount()
    else:
        if self.isSeparated:
            result = separatedAmount()
        else:
            if self.isRetired:
                result = retiredAmount()
            else:
                result = normalPayAmount()
    return result
```

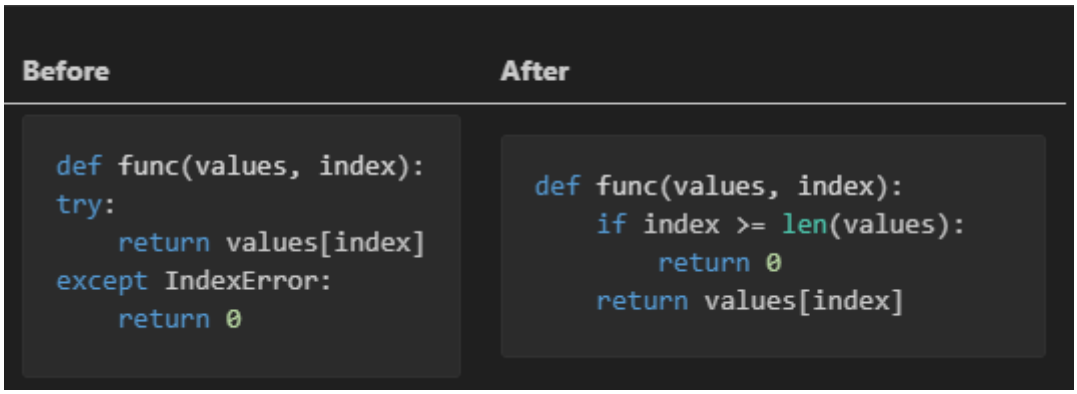
After

```
def getPayAmount(self):
    if self.isDead:
        return deadAmount()
    if self.isSeparated:
        return separatedAmount()
    if self.isRetired:
        return retiredAmount()
    return normalPayAmount()
```

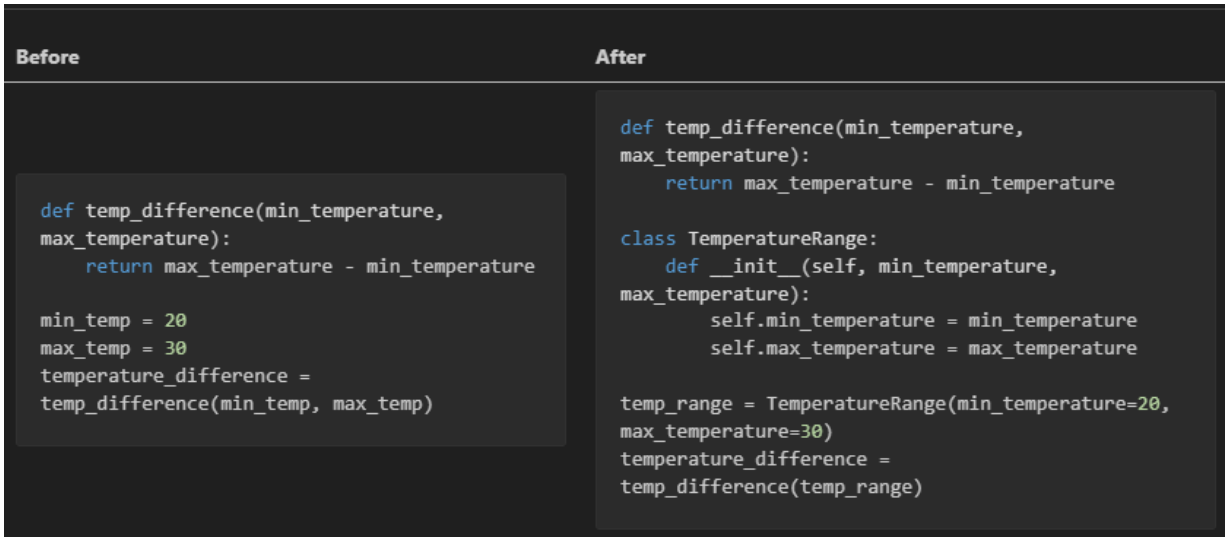
Εικόνα 8 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR REPLACE NESTED CONDITIONAL WITH GUARD CLAUSES

- **Απλοποίηση κλήσης μεθόδων (Simplifying Method Calls) :** Η εφαρμογή των Refactors της κατηγορίας αυτής επιτυγχάνει τη βελτίωση της αναγνωσιμότητας, της συντηρησιμότητας και της επαναχρησιμοποίησης του κώδικα, ενισχύοντας την απλότητα και την καθαρότητά του μειώνοντας τον κίνδυνο εμφάνισης λαθών.
  Η κατηγορία περιέχει τους εξής τύπους Refactor:
  - Add Parameter
  - Remove Parameter
  - Rename Method
  - Separate Query from Modifier
  - Parameterize Method

- Introduce Parameter Object
- Preserve Whole Object
- Remove Setting Method
- Replace Parameter with Explicit Methods
- Replace Parameter with Method Call
- Hide Method
- Replace Constructor with Factory Method
- Replace Error Code with Exception
- Replace Exception with Test



Εικόνα 9 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR REPLACE EXCEPTION WITH TEST



Εικόνα 10 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR PRESERVE WHOLE OBJECT

- **Αντιμετώπιση γενικεύσεων (Dealing with Generalizations):** Η κατηγορία αυτή στοχεύει στην βελτίωση της κατανοητότητας, της συντηρησιμότητας και στην ευκολία επέκτασης του κώδικα.
  Στην κατηγορία αυτή ανήκουν οι παρακάτω τύποι Refactors:
  - Pull Up Field

- Pull Up Method
- Pull Up Constructor Body
- Push Down Field
- Push Down Method
- Extract Subclass
- Extract Superclass
- Extract Interface
- Collapse Hierarchy
- Form Template Method
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance

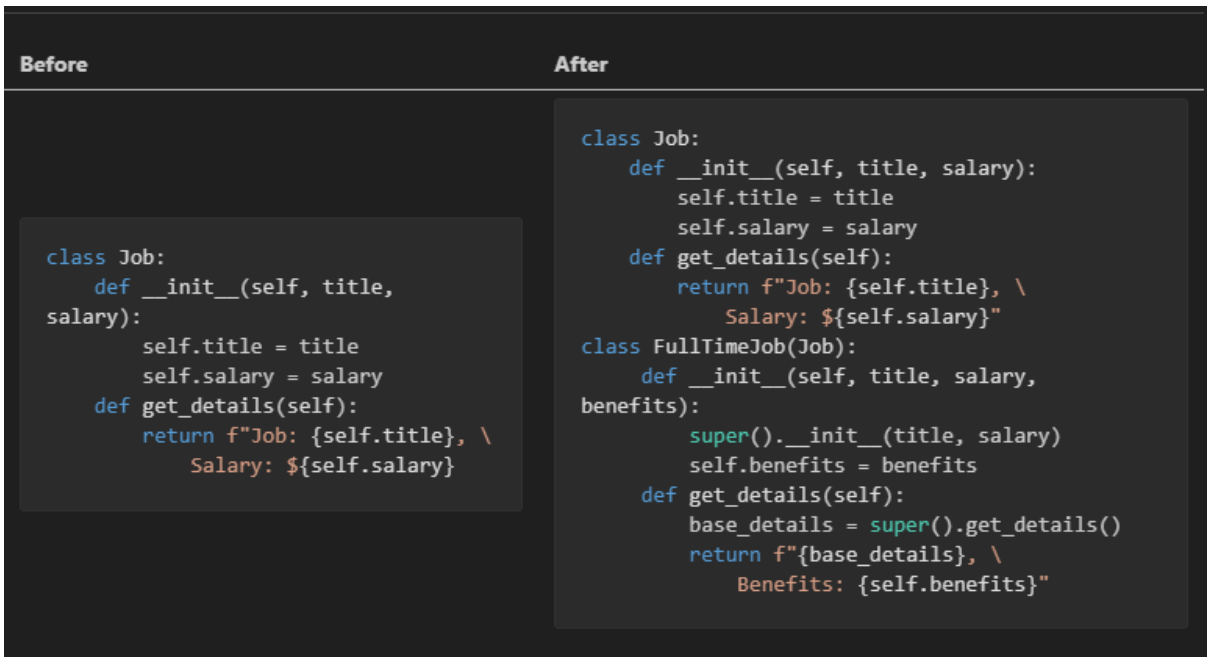
Before

```
class A:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def a_method(self):
        return self.x+self.y
class B(A):
    def __init__(self,x,y,z):
        super().__init__(x,y)
        self.z=z
    def a_method(self):
        base = super().a_method()
        return base + self.z
```

After

```
class A:
    def __init__(self,x,y,z=None):
        self.x=x
        self.y=y
        self.z=z
    def a_method(self):
        return self.x+self.y +\
            self.z if self.z is not None else ""
```

Εικόνα 11 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR EXTRACT SUBCLASS



Εικόνα 12 : ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ REFACTOR COLLAPSE HIERARCHY

2.1 Η δυσκολία εύρεσης Refactoring σε έργα Python

Όπως παρατηρήθηκε, τα περισσότερα εργαλεία εύρεσης Refactoring που έχουν αναπτυχθεί αφορούν έργα γραμμένα σε γλώσσα java. Αυτό συμβαίνει επειδή η γλώσσα java ακολουθεί αυστηρή και στατική δομή αλλά και αυστηρούς κανόνες. Από την άλλη, η γλώσσα Python χαρακτηρίζεται από την δυναμική της φύση, την ευελιξία αλλά και την έλλειψη κανόνων και δομής. Αν και αυτά τα χαρακτηριστικά την καθιστούν μια από τις ισχυρότερες γλώσσες υψηλού επιπέδου, παράλληλα όμως καθιστούν την ανάλυση του κώδικά της ιδιαίτερα δύσκολη. Ένα παράδειγμα της δυσκολίας αυτής είναι η δυναμική δημιουργία και επεξεργασία δεδομένων. Ο χαρακτηρισμός των δεδομένων αυτών είναι ιδιαίτερα δύσκολος αφού η γλώσσα Python δεν απαιτεί τον ορισμό τύπων(strings ,int, list κ.α.). Μια άλλη δυσκολία είναι η ανάλυση των πολλαπλών τρόπων επίλυσης θεμάτων. Για παράδειγμα, το θέμα εκτύπωσης μια λίστας δεδομένων μπορεί να επιτευχθεί με τουλάχιστον 5 τρόπους, με τη χρήση της απλής επανάληψης, της συνάρτησης range(), enumerate(), zip() κ.α.). Το ίδιο ακριβώς συμβαίνει με τις περισσότερες λειτουργίες που την αποτελούν. Αυτό έχει ως αντίκτυπο την πρόβλεψη πολλαπλών περιπτώσεων καθιστώντας την ανάλυση ιδιαίτερα πολύπλοκη.

2.2 Υπάρχουσες υλοποιήσεις

Οι τεχνικές εύρεσης του Refactoring είναι ένα θέμα που έχει απασχολήσει ιδιαίτερα την ερευνητική κοινότητα. Πολλοί ερευνητές μπήκαν στη διαδικασία δημιουργίας εργαλείων που να πραγματοποιούν τον εντοπισμό αυτών. Ένα από τα σημαντικότερα έργα είναι το Refactor miner[2]. Το συγκεκριμένο έργο αφορά ένα εργαλείο το οποίο εντοπίζει τύπους Refactors σε έργα γραμμένα σε java. Ο εντοπισμός των Refactorings γίνεται στις καταχωρήσεις (commits) στο Github του εκάστοτε έργου. Η λειτουργία του περιγράφεται από την ανάλυση των δενδρικών δομών δεδομένων Abstract Syntax Tree(AST) και

χρησιμοποιώντας ευρετικές μεθόδους και αλγόριθμους μηχανικής μάθησης, επιτρέπει τον αυτόματο εντοπισμό και παρακολούθηση των Refactoring. Πάνω στο Refactor miner έχουν αναπτυχθεί εργαλεία όπως είναι το JavaFyPy[3] και το Pyref[4].Το έργο JavaFyPy μετατρέπει το εκάστοτε Python έργο σε java και με την βοήθεια του Refactor miner εντοπίζει Refactors. Το Pyref με την ίδια λογική με το Refactor miner χρησιμοποιεί AST δεδομένα και ευρετικούς (Heuristic) αλγορίθμους. Σημαντικό είναι να αναφερθεί ότι η υπάρχουσα έκδοση του έργου εντοπίζει σχετικά λίγους τύπους Refactors αλλά με σχετικά μεγάλη ακρίβεια. Ένα ακόμα σημαντικό εργαλείο που έχει αναπτυχθεί είναι το Ref-Finder[5] το οποίο με την χρήση του έργου lsdiff[6] εντοπίζει Refactoring σε ιστορικά δεδομένα δεσμεύσεων λογισμικού γραμμένα σε Java. Συνοψίζοντας, τα περισσότερα εργαλεία που υπάρχουν αφορούν αυστηρά δομημένες γλώσσες, όπως η Java, ενώ αυτά που αφορούν την γλώσσα Python βρίσκονται ακόμη σε πρώιμο στάδιο, μιας και τα περισσότερα εντοπίζουν ελάχιστους τύπους Refactoring, κάτι το οποίο δημιουργεί χώρο για περαιτέρω μελέτη και επέκταση αυτών.

Κεφάλαιο 3: Ανάλυση και Σχεδίαση Λογισμικού

Το κεφάλαιο αυτό περιέχει τον σκοπό του λογισμικού που αναπτύχθηκε και την περιγραφή αυτού. Περιγράφει ακόμη την μεθοδολογία που ακολουθείται για την εύρεση των διαφόρων τύπων Refactor.

3.1 Σκοπός του λογισμικού

Όπως προαναφέρθηκε, ο αρχικός στόχος της διπλωματικής αυτής εργασίας είναι, ο αυτοματοποιημένος εντοπισμός Refactors σε έργα λογισμικού που είναι γραμμένα σε γλώσσα Python. Ο απώτερος στόχος είναι η συγκέντρωση στατιστικών και στοιχείων που αφορούν διάφορους τύπους Refactors σε μεγάλο πλήθος εκδόσεων διαφόρων έργων.

Για την επίτευξη των στόχων αυτών αναπτύχθηκε λογισμικό το οποίο εκτελεί τις παρακάτω ενέργειες :

- Εντοπίζει αλλαγές μεταξύ εκδόσεων
- Αναλύει τα αρχεία κώδικα Python που παρατηρήθηκαν αλλαγές
- Με την χρήση αλγορίθμων και κανόνων εντοπίζει διαφόρους τύπους Refactors
- Δημιουργεί αυτοματοποιημένα τα αποτελέσματα και τα στατιστικά για τα Refactors που βρέθηκαν

3.2 Λογισμικό

Ακολουθεί η περιγραφή των εισόδων και εξόδων του λογισμικού αλλά και η βασική περιγραφή της λειτουργίας του.

3.2.1 Είσοδος

Το πρόγραμμα μπορεί να έχει ως είσοδο μια από τις παρακάτω μορφές:

- Δύο φάκελοι οι οποίοι είναι δύο εκδόσεις του ίδιου λογισμικού

  Παράδειγμα : **python src/main.py release_1_folder release_2_folder**

- Ένας φάκελος με το όνομα του project ο οποίος περιέχει υποφακέλους με πολλαπλές εκδόσεις του λογισμικού

  Παράδειγμα : **python src/main.py project_name_folder**

3.2.2 Έξοδος

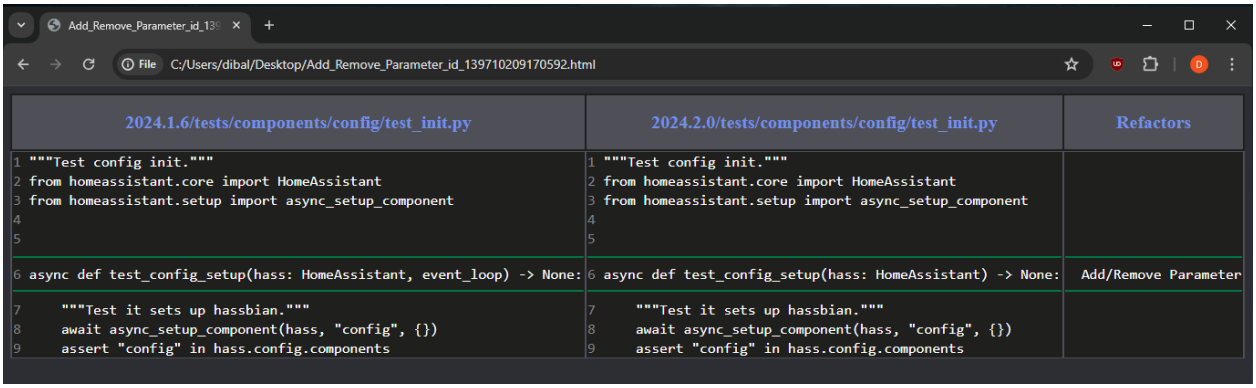
Η έξοδος του προγράμματος βρίσκεται σε έναν φάκελο με το όνομα του project (αυτό που δόθηκε στην είσοδο ή με όνομα των εκδόσεων μέσα σε αγκύλες αντίστοιχα πχ [release_1_folder][release_2_folder]).Ο φάκελος αυτός βρίσκεται στο φάκελο Results ο οποίος είναι στον ίδιο κατάλογο (directory) με τον φάκελο src.

Ο φάκελος εξόδου περιέχει τα παρακάτω :

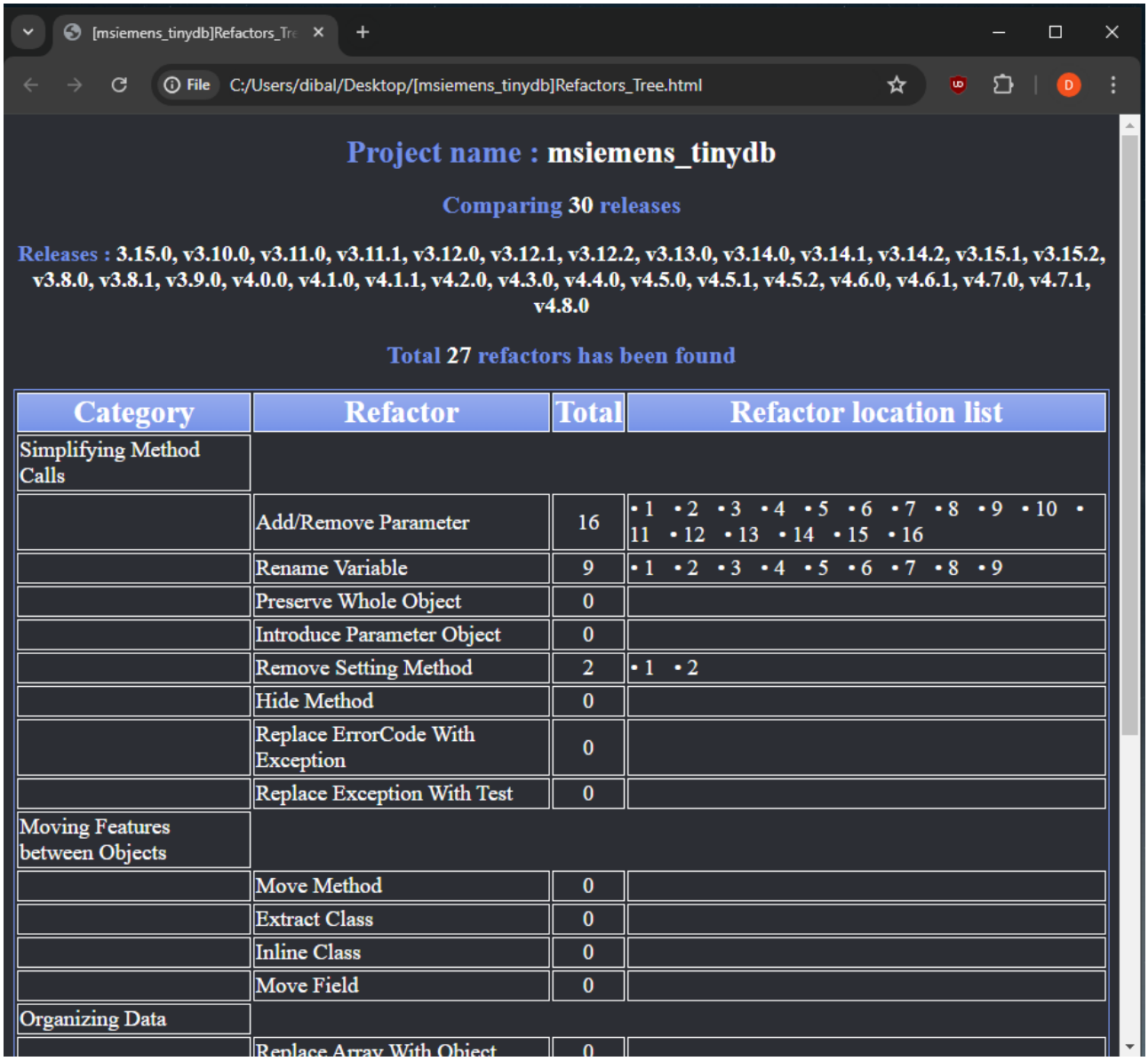
- Έναν φάκελο με όνομα **html files** ο οποίος περιέχει αρχεία html. Το κάθε html αρχείο είναι και ένα ξεχωριστό Refactor που εντοπίστηκε. Το περιεχόμενο του αρχείου είναι τρεις στήλες, όπου οι δύο πρώτες είναι ο κώδικας των δύο διαφορετικών εκδόσεων Python αρχείων και η τρίτη ο τύπος του Refactor που εντοπίστηκε.

- Ένα αρχείο με όνομα **Refactors_tree.html** το οποίο είναι ένας πίνακας με όλα τα Refactors που εντοπίστηκαν. Ο πίνακας επίσης περιέχει υπερσυνδέσμους που οδηγούν στα αρχεία του φακέλου html files.

- Ένα αρχείο **Execution.txt** που περιέχει πληροφορίες για την εκτέλεση του προγράμματος και τα αποτελέσματά του.
- Ένα αρχείο **Refactors.xlsx** που περιέχει όλα τα Refactors που εντοπίστηκαν. Κάθε σειρά αυτού excel αρχείου είναι ένα ξεχωριστό Refactor με πληροφορίες για αυτό. Οι στήλες του αρχείου είναι:
    - Το όνομα του Refactor
    - Την κατηγορία του Refactor
    - Το όνομα του project που βρέθηκε το Refactor
    - Οι δύο εκδόσεις του project
    - Οι δύο διαδρομές των αρχείων Python στων οποίων βρέθηκε το Refactor
    - Ο αριθμός της αρχικής και τελικής γραμμής κώδικα των παραπάνω δύο αρχείων



Εικόνα 13 : ΠΑΡΑΔΕΙΓΜΑ ΕΞΟΔΟΥ HTML ΑΡΧΕΙΟΥ ΜΕΤΑ ΤΗΝ ΕΥΡΕΣΗ ΤΟΥ REFACTOR ADD/REMOVE PARAMETER



[msiemens_tinydb]Refactors_Tre

File C:/Users/dibal/Desktop/[msiemens_tinydb]Refactors_Tree.html

Project name : msiemens_tinydb

Comparing 30 releases

Releases : 3.15.0, v3.10.0, v3.11.0, v3.11.1, v3.12.0, v3.12.1, v3.12.2, v3.13.0, v3.14.0, v3.14.1, v3.14.2, v3.15.1, v3.15.2, v3.8.0, v3.8.1, v3.9.0, v4.0.0, v4.1.0, v4.1.1, v4.2.0, v4.3.0, v4.4.0, v4.5.0, v4.5.1, v4.5.2, v4.6.0, v4.6.1, v4.7.0, v4.7.1, v4.8.0

Total 27 refactors has been found

Category	Refactor	Total	Refactor location list
Simplifying Method Calls			
	Add/Remove Parameter	16	• 1 • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9 • 10 • 11 • 12 • 13 • 14 • 15 • 16
	Rename Variable	9	• 1 • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9
	Preserve Whole Object	0	
	Introduce Parameter Object	0	
	Remove Setting Method	2	• 1 • 2
	Hide Method	0	
	Replace ErrorCode With Exception	0	
	Replace Exception With Test	0	
Moving Features between Objects			
	Move Method	0	
	Extract Class	0	
	Inline Class	0	
	Move Field	0	
Organizing Data			
	Replace Array With Object	0	

Εικόνα 14 : ΠΑΡΑΔΕΙΓΜΑ ΕΞΟΔΟΥ ΑΡΧΕΙΟΥ Refactors_tree.html ΜΕΤΑ ΤΗΝ ΕΠΙΤΥΧΗΜΕΝΗ ΕΚΤΕΛΕΣΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ

3.2.3 Βασική περιγραφή λειτουργίας

Για την εκτέλεση του προγράμματος απαιτείται η ύπαρξη ενός config.py αρχείου (εντός του src φακέλου) το οποίο θα πρέπει να περιέχεται το πεδίο “MINIFY_HTML_BOOL = True”. Το πεδίο αυτό χρησιμοποιείται για την επιλογή χρήσης ή μη της βιβλιοθήκης **minify_html**[2] η οποία ελαχιστοποιεί το μέγεθος των html αρχείων που δημιουργούνται. Τα περαιτέρω πεδία του αρχείου αυτού αναφέρονται σε παρακάτω κεφάλαια. Κατά την εκτέλεση του προγράμματος ελέγχεται αν η είσοδός του, είναι μία από τις δύο διαφορετικές εκδοχές (δύο φάκελοι με δύο διαφορετικές εκδόσεις ή ένας φάκελος ο

[2] https://pypi.org/project/minify-html/

οποίος περιέχει υποφακέλους με πολλαπλές εκδόσεις). Στην περίπτωση της δεύτερης εκδοχής, δημιουργείται μια λίστα με ζευγάρια από τους υποφακέλους, όπου κάθε ζευγάρι είναι η διαδρομή των φακέλων δύο διαδοχικών εκδόσεων.Έπειτα δημιουργεί ένα νήμα (thread) για κάθε ζευγάρι της παραπάνω λίστας. Κάθε νήμα εκτελεί την ίδια κοινή διαδικασία που θα εκτελούσε το πρόγραμμα, αν η αρχική είσοδος του προγράμματος ήταν η πρώτη εκδοχή (δύο φάκελοι με δύο διαφορετικές εκδόσεις).

Στη συνέχεια ακολουθούνται τα παρακάτω βήματα

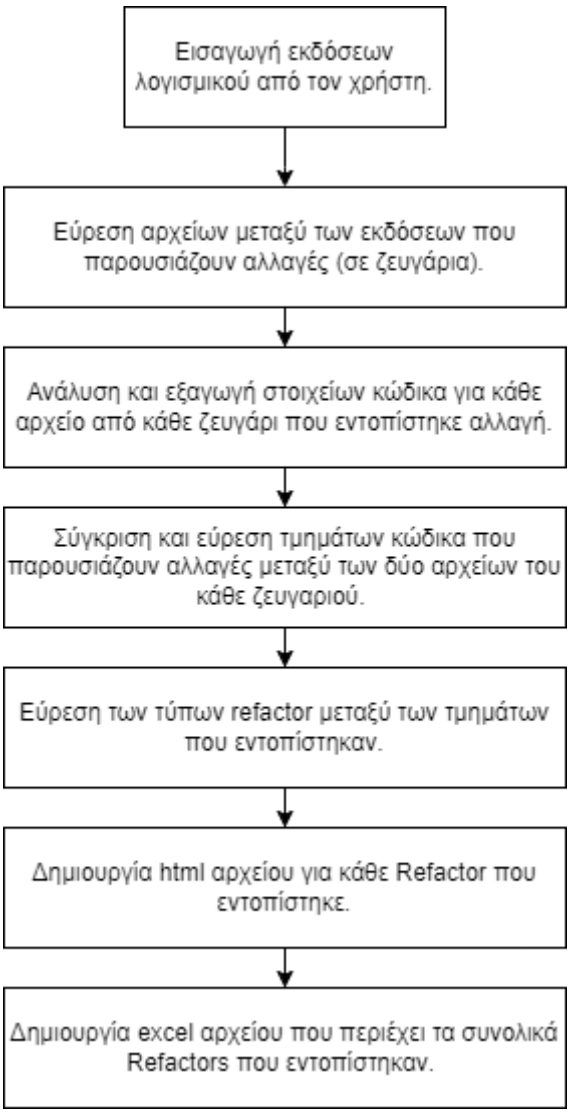
1. Μέσω τη βιβλιοθήκης **filecmp**[3] συγκρίνονται όλα τα Python αρχεία κάθε φακέλου των δύο εκδόσεων και δημιουργείται μια λίστα με ζευγάρια (με όνομα της λίστας comparable files) τα οποία έχουν κοινή διαδρομή και όνομα αλλά το περιεχόμενό τους έχει αλλάξει μεταξύ των δύο εκδόσεων αυτών. Ακολουθεί η εύρεση νεοεισαχθέντων και διαγραμμένων κλάσεων και ανάθεση αυτών σε αντίστοιχες λίστες.
2. Για κάθε ζευγάρι αρχείων της παραπάνω λίστας, διαβάζεται το κάθε αρχείο ξεχωριστά και μέσω της συνάρτησης **Line_type_separation** γίνεται ο διαχωρισμός της κάθε γραμμής του κώδικα αυτού. Ο διαχωρισμός που γίνεται καθορίζει αν η κάθε γραμμή είναι σχόλιο, κλάση, επανάληψη, ορισμός μεταβλητής, ορισμός συνάρτησης κ.α. .

   Μόλις ολοκληρωθεί η παραπάνω διαδικασία , μέσω της βιβλιοθήκης **difflib**[4] βρίσκονται τα κομμάτια κώδικα που έχουν τροποποιηθεί μεταξύ των εκδόσεων και τοποθετούνται σε μια λίστα (με όνομα της λίστας Diffblocks). Η λίστα αυτή αποτελείται από αντικείμενα **Diffblock** τα οποία έχουν ως χαρακτηριστικά (attributes) δύο λίστες με τον κώδικα που αντιστοιχεί σε κάθε αρχείο ξεχωριστά.

3. Πλέον, για κάθε αντικείμενο της λίστας Diffblocks γίνεται ο έλεγχος για την ύπαρξη Refactors μέσω της συνάρτησης **CheckForRefactors**. Η λειτουργία της συνάρτησης αυτής περιγράφεται σε επόμενο κεφάλαιο.
4. Όταν ολοκληρωθεί η εύρεση των Refactors για το κάθε στοιχείο της λίστας Diffblocks , δημιουργούνται ξεχωριστά html αρχεία με κάθε διαφορετικό Refactor στον φάκελο εξόδου **html files**.

Η διαδικασία των βημάτων 2, 3 και 4 εκτελείται για κάθε ζευγάρι της λίστας comparable files. Μετά το πέρας αυτής, γίνεται η δημιουργία των συγκεντρωτικών αρχείων **Refactors_tree.html**, **Refactors.xlsx** και **Execution.txt.**

[3] https://docs.python.org/3/library/filecmp.html
[4] https://docs.python.org/3/library/difflib.html



Σχήμα 1 : Διαγραμματική απεικόνιση του λογισμικού Refactoring Patterns Recognition for Python

3.2.4 Διαδικασία εύρεσης Refactors

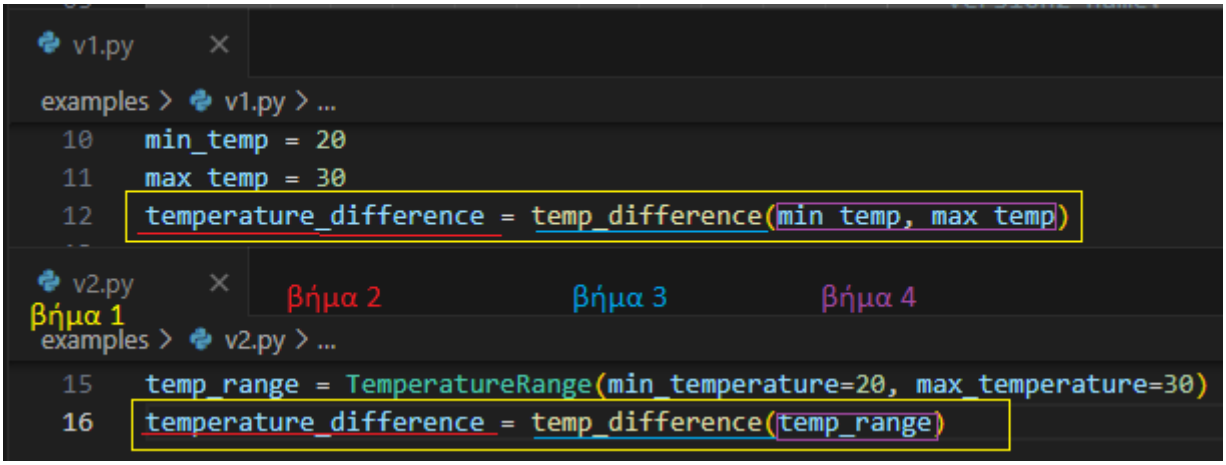
Η εύρεση των Refactors όπως προαναφέρθηκε, γίνεται μέσω της συνάρτησης **CheckForRefactors**. Η λειτουργία της συνάρτησης αυτής είναι να καλεί διαδοχικά και με τα σωστά ορίσματα τις συναρτήσεις που εντοπίζουν τους διάφορους τύπους Refactors. Κάθε μια συνάρτηση από αυτές αντιστοιχεί σε διαφορετικό τύπο Refactor.

Το κοινό χαρακτηριστικό των συναρτήσεων αυτών είναι ότι η διαδικασία εύρεσης πραγματοποιείται ανάμεσα σε κάθε Diffblock αντικείμενο, το οποίο αποτελείται από δύο λίστες με κομμάτια κώδικα δύο αρχείων διαφορετικών εκδόσεων. Ανάλογα τον τύπο του Refactor ακολουθούνται διαφορετικά κριτήρια με σκοπό την ταυτοποίηση της ύπαρξης αυτών. Η διαδικασία αυτή αναλύεται παρακάτω ξεχωριστά για κάθε τύπο Refactor. Μετά την ολοκλήρωση της διαδικασίας αυτής, δημιουργούνται αντικείμενα Refactor, τα οποία προστίθενται στην συγκεντρωτική λίστα με τα υπόλοιπα Refactors.

3.3 Κριτήρια/βήματα εύρεσης ανά τύπο Refactor

Τα βήματα που ακολουθούνται για την εύρεση των διαφόρων τύπων περιγράφονται στην συνέχεια. Οι τύποι των Refactors χωρίζονται στις εξής κατηγορίες :

- Απλοποίηση κλήσεων μεθόδων (Simplifying Method Calls)
  - **Προσθήκη/Αφαίρεση παραμέτρου (Add/Remove Parameter)**
    1. Εύρεση συναρτήσεων σε κάθε ζεύγος γραμμών ανάμεσα στις γραμμές κώδικα των δύο λιστών του αντικειμένου Diffblock.
    2. Έλεγχος κοινού ονόματος των δύο συναρτήσεων.
    3. Έλεγχος των παραμέτρων των δύο συναρτήσεων αν είναι διαφορετικές.
  - **Μετονομασία μεταβλητής (Rename variable)**
    1. Εύρεση ορισμού μεταβλητής (variable declaration) σε κάθε ζεύγος γραμμών ανάμεσα στις γραμμές κώδικα των δύο λιστών του αντικειμένου Diffblock.
    2. Έλεγχος του ονόματος των δύο μεταβλητών αν είναι διαφορετικό.
    3. Έλεγχος του περιεχομένου της ανάθεσης των δύο μεταβλητών εάν είναι κοινό.
  - **Διατήρηση ολόκληρου αντικειμένου (Preserve Whole Object)**
    1. Εύρεση ορισμού μεταβλητής (variable declaration) σε κάθε ζεύγος γραμμών ανάμεσα στις γραμμές κώδικα των δύο λιστών του αντικειμένου Diffblock.
    2. Έλεγχος αν τα ονόματα των δύο μεταβλητών των δυο εκδόσεων είναι κοινά.
    3. Έλεγχος του περιεχομένου της ανάθεσης της μεταβλητής αν είναι κλήση συνάρτησης.
    4. Σύγκριση των παραμέτρων των συναρτήσεων των δύο εκδόσεων.
    5. Έλεγχος αν η παράμετρος της συνάρτησης της επόμενης έκδοσης είναι αντικείμενο που ανήκει στη λίστα με τις νεοεισαχθείσες κλάσεις.



Εικόνα 15 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR PRESERVE WHOLE OBJECT

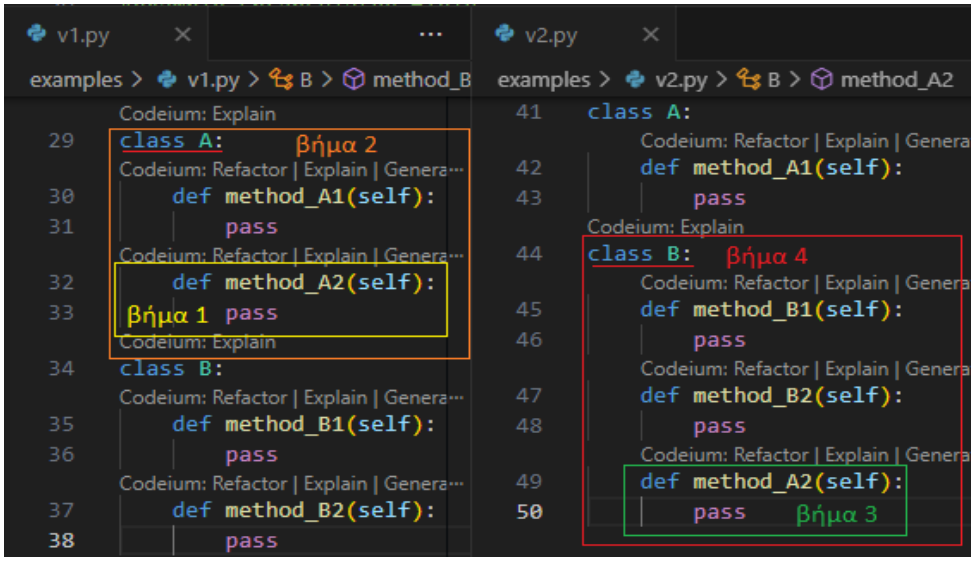
- **Εισαγωγή αντικειμένου παραμέτρων (Introduce Parameter Object)**
  1. Εύρεση συναρτήσεων σε κάθε ζεύγος γραμμών ανάμεσα στις γραμμές κώδικα των δύο λιστών του αντικειμένου Diffblock.

2. Έλεγχος αν τα ονόματα των δύο συναρτήσεων των δυο εκδόσεων είναι κοινά.
3. Σύγκριση των ορισμάτων των δυο συναρτήσεων.
4. Έλεγχος αν το όρισμα της συνάρτησης της δεύτερης έκδοσης είναι αντικείμενο που ανήκει στη λίστα με τις νεοεισαχθείσες κλάσεις.

- **Αφαίρεση μεθόδου ορισμού (Remove Setting Method)**
  1. Εύρεση συναρτήσεων στην πρώτη έκδοση οι οποίες είναι μέθοδοι κλάσεων και το όνομά τους ξεκινά με “Set” ή “set”.
  2. Έλεγχος αν η συνάρτηση της πρώτης έκδοσης έχει διαγραφεί από την επόμενη.
- **Απόκρυψη μεθόδου (Hide Method)**
  1. Εύρεση συναρτήσεων στις γραμμές κώδικα της πρώτης έκδοσης του αρχείου.
  2. Έλεγχος της συνάρτησης αν είναι μέθοδος μιας κλάσης (ως Func1).
  3. Εύρεση των συναρτήσεων στις γραμμές της δεύτερης έκδοσης (ως Func2).
  4. Έλεγχος της συνάρτησης Func1 εάν έχει το ίδιο όνομα με την συνάρτηση Func2 με την προϋπόθεση να έχουν προστεθεί στη αρχή του ονόματός της οι χαρακτήρες “__”.
- **Αντικατάσταση κωδικού σφάλματος με εξαίρεση (Replace Error Code With Exception)**
  1. Εύρεση γραμμών κώδικα τύπου επιστροφής (return) του αρχείου της πρώτης έκδοσης .
  2. Έλεγχος αν το περιεχόμενο της επιστροφής είναι αριθμητικό. (πχ return -1).
  3. Έλεγχος της αντίστοιχης γραμμής κώδικα του αρχείου της δεύτερης έκδοσης αν έχει αντικατασταθεί με το “raise” ακολουθούμενα από κάποιον τύπο εξαίρεσης.
- **Αντικατάσταση Εξαίρεσης με Δοκιμή (Replace Exception With Test)**
  1. Εύρεση γραμμών κώδικα τύπου εξαίρεσης (Exception) του αρχείου της πρώτης έκδοση .
  2. Εύρεση γραμμών κώδικα τύπου (Conditional) του αρχείου της δεύτερης έκδοσης.
  3. Έλεγχος του περιεχομένου του except αν είναι ίδιο με το περιεχόμενο του Conditional.

Εικόνα 16 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR REPLACE EXCEPTION WITH TEST

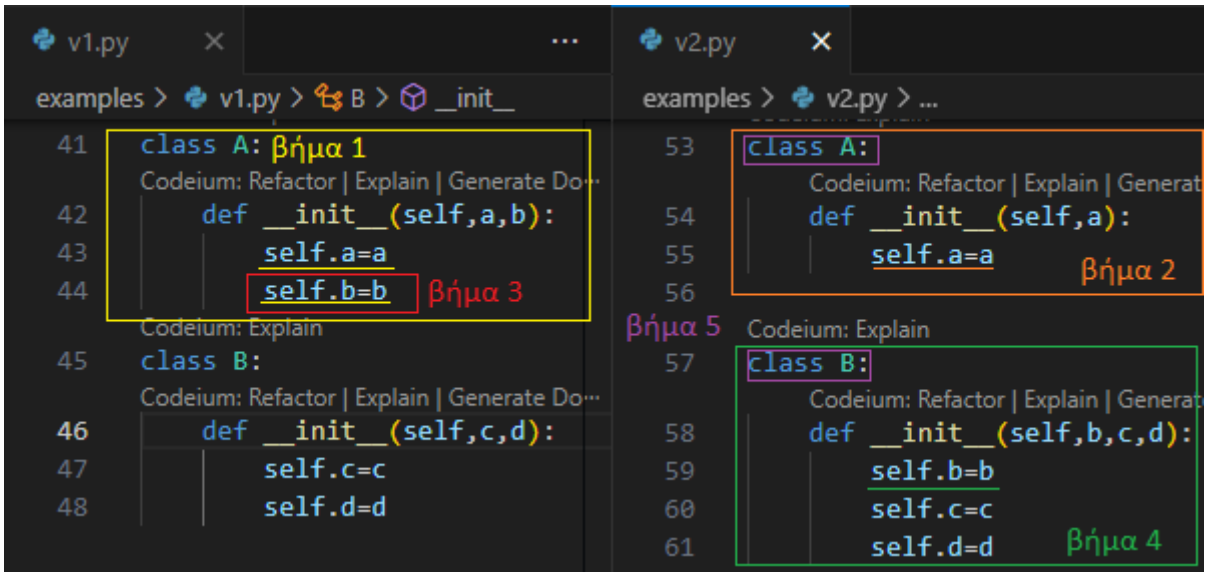
- Μετακίνηση χαρακτηριστικών μεταξύ αντικειμένων (Moving Features Between Objects)
  - **Μετακίνηση μεθόδου (Move method)**
    1. Εύρεση συναρτήσεων στις γραμμές κώδικα της πρώτης έκδοσης του αρχείου.
    2. Έλεγχος της συνάρτησης αν είναι μέθοδος κλάσης.
    3. Έλεγχος στο αρχείο της δεύτερης έκδοσης για την εύρεση της ίδιας συνάρτησης με αυτής της πρώτης.
    4. Έλεγχος της συνάρτησης που βρέθηκε στη δεύτερη έκδοση αν είναι μέθοδος κλάσης και αν η κλάση που ανήκει δεν έχει κάποια σχέση (υπό ή υπέρ κλάση) με την κλάση της πρώτης.
    5. Έλεγχος αν η συνάρτηση που βρέθηκε στην δεύτερη έκδοση δεν υπάρχει εντός της ίδιας κλάσης με αυτής της πρώτης.



Εικόνα 17 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR MOVE METHOD

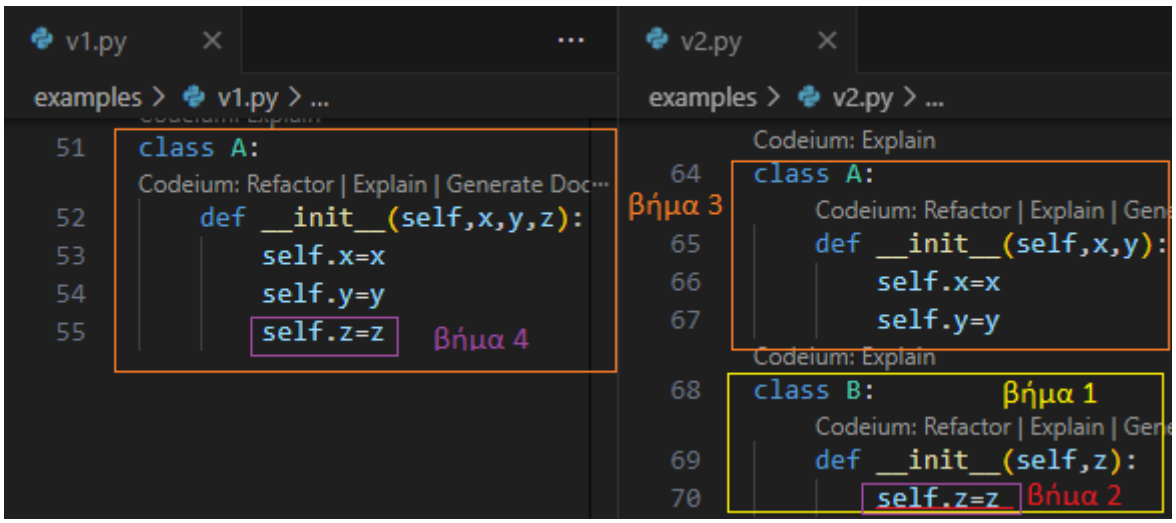
  - **Μετακίνηση πεδίου (Move field)**
    1. Εύρεση κλάσεων στις γραμμές κώδικα της πρώτης έκδοσης του αρχείου και των πεδίων αυτών(ως class A από v1).
    2. Εύρεση της ίδιας κλάσης (της πρώτης έκδοσης) στην δεύτερη έκδοση του αρχείου και των πεδίων αυτής(ως class A από v2).
    3. Σύγκριση των πεδίων των δύο κλάσεων.
    4. Εύρεση κάθε διαφορετικού πεδίου εντός κάποιας άλλης κλάσης στο αρχείο της δεύτερης έκδοσης (ως class B από v2).
    5. Έλεγχος της συσχέτισης της κλάσης που βρέθηκε να ανήκει το διαφορετικό πεδίο της πρώτης κλάσης(class A και class B v2). Η συσχέτιση αφορά το αν είναι υπό ή υπέρ κλάση η μια της άλλης.

6. Έλεγχος του πεδίου της κλάσης class B v2 αν υπάρχει στην ίδια κλάση της πρώτης έκδοσης class B v1.



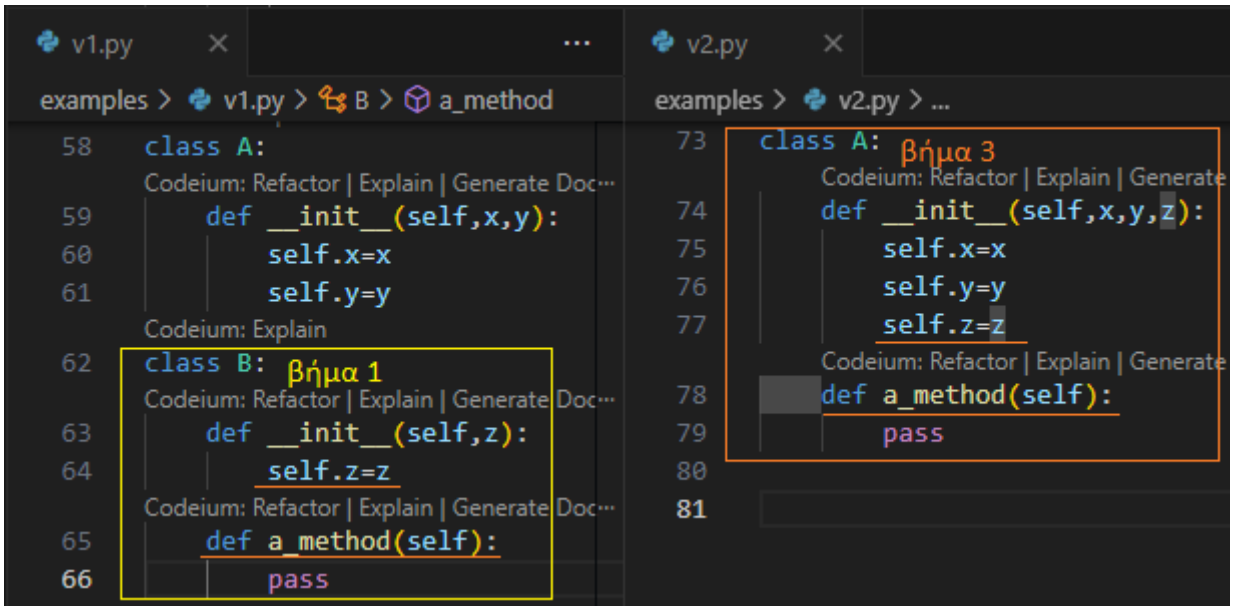
Εικόνα 18 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR MOVE FIELD

- **Εξαγωγή κλάσης (Extract class)**
  1. Εύρεση κλάσεων στις γραμμές κώδικα της δεύτερης έκδοσης του αρχείου (ως class B v2) που ανήκουν στη λίστα με τις νεοεισαχθείσες κλάσεις.
  2. Εύρεση των πεδίων στον κονστράκτορα της κλάσης αυτής (class B v2).
  3. Εύρεση κλάσεων στο αρχείο της πρώτης έκδοσης που να αντιστοιχούν με αυτές της δεύτερης (ως class A v1 και class A v2 για την δεύτερη έκδοση).
  4. Εύρεση των πεδίων στον κονστράκτορα της κλάσης αυτής (class A v1) και σύγκριση των πεδίων αυτών με τα πεδία της class B v2. Ο σκοπός της σύγκρισης είναι η εύρεση κοινών πεδίων.
  5. Έλεγχος για την ύπαρξη του κοινού πεδίου στη class A v2



Εικόνα 19 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR EXTRACT CLASS

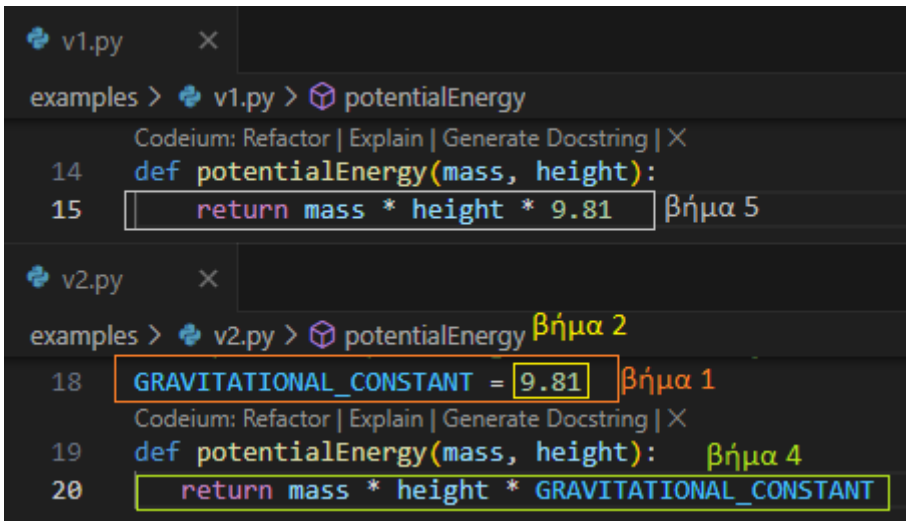
- **Ενσωμάτωση κλάσης (Inline class)**
  1. Εύρεση κλάσεων στις γραμμές κώδικα της πρώτης έκδοσης του αρχείου (ως class B v1) που ανήκουν στη λίστα με τις διαγραμμένες κλάσεις.
  2. Έλεγχος για το αν η κλάση class B υπάρχει στην δεύτερη έκδοση και απλώς δεν έχει μετονομαστεί.
  3. Εύρεση κλάσης στην δεύτερη έκδοση η οποία έχει κοινά πεδία και μεθόδους με την class B από v1 (ως class A v2).
  4. Έλεγχος για το αν υπάρχουν τα κοινά πεδία και οι μέθοδοι στην ίδια κλάση class A στην πρώτη έκδοση.



Εικόνα 20 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR INLINE CLASS

- Οργάνωση δεδομένων (Organizing data)
  - **Αντικατάσταση πίνακα με Αντικείμενο (Replace array with object)**
    1. Εύρεση ορισμού μεταβλητής (variable declaration) σε κάθε ζεύγος γραμμών ανάμεσα στις γραμμές κώδικα των δύο λιστών του αντικειμένου Diffblock.
    2. Έλεγχος του περιεχομένου της μεταβλητής της πρώτης έκδοσης αν είναι πίνακας.
    3. Έλεγχος του περιεχομένου της μεταβλητής της επόμενης έκδοσης είναι αντικείμενο και ανήκει στη λίστα με τις νεοεισαχθείσες κλάσεις.

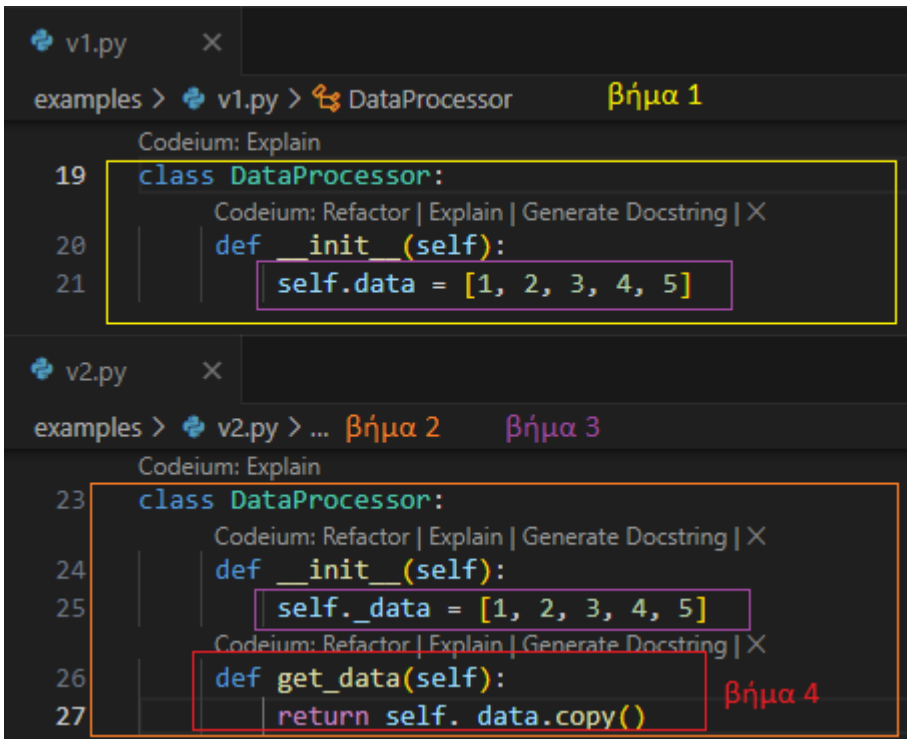
  - **Αντικατάσταση "μαγικού αριθμού" με σταθερή μεταβλητή(Replace magic number with symbolic constant)**
    1. Εύρεση ορισμών μεταβλητών (variable declaration) στις γραμμές κώδικα της δεύτερης έκδοσης του αρχείου.
    2. Έλεγχος του περιεχομένου της μεταβλητής της, εάν είναι αριθμητικό.
    3. Έλεγχος στις γραμμές του κώδικα της πρώτης έκδοσης αν δεν υπάρχει ο ορισμός μεταβλητής που βρέθηκε στην δεύτερη έκδοση.
    4. Έλεγχος σε όλες τις γραμμές κώδικα της δεύτερης έκδοσης εφόσον εάν αυτή η μεταβλητή έχει χρησιμοποιηθεί.
    5. Σύγκριση της γραμμής που βρέθηκε στην έκδοση δύο αν είναι κοινή με αυτή της πρώτης έκδοσης με τη διαφορά του ότι έχει αντικατασταθεί η αριθμητική τιμή με την μεταβλητή.



Εικόνα 21 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR REPLACE MAGIC NUMBER WITH SYMBOLIC CONSTANT

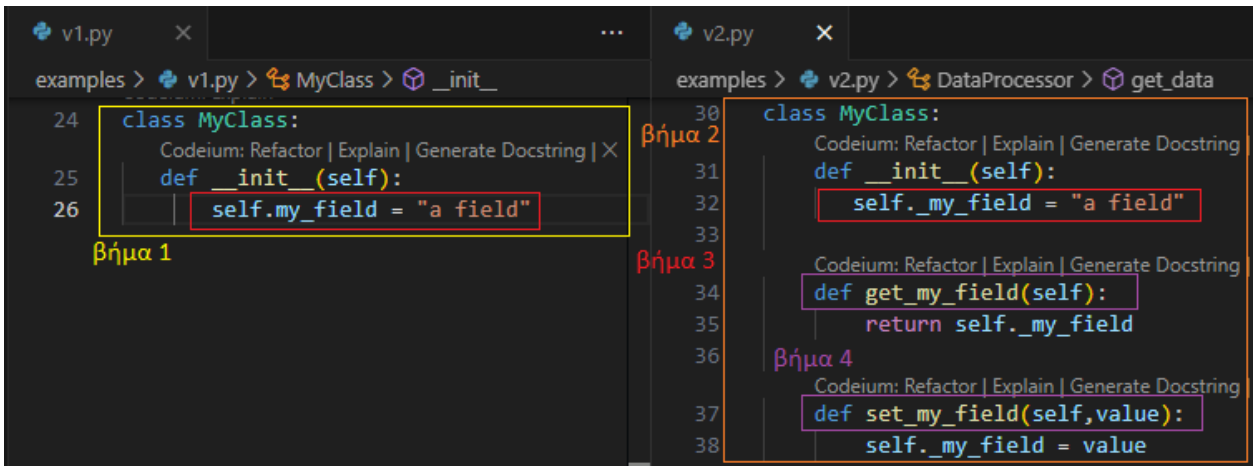
  - **Ενθυλακωμένη συλλογή (Encapsulated Collection)**
    1. Εύρεση κλάσεων στις γραμμές κώδικα της πρώτης έκδοσης του αρχείου.

2. Εύρεση της ίδιας κλάσης στις γραμμές κώδικα της δεύτερης έκδοσης του αρχείου.
3. Σύγκριση των πεδίων του κονστράκτορα των δύο κλάσεων. Η σύγκριση αφορά αν το πεδίο είναι ορισμός πίνακα, λίστας ή συλλογής και το όνομα του πεδίου της δεύτερης έκδοσης έχει προστεθεί στη αρχή του ο χαρακτήρας "_".
4. Έλεγχος αν έχει προστεθεί κάποια μέθοδος στην κλάση της δεύτερης έκδοσης και περιέχει τις λέξεις "Get" ή " get".



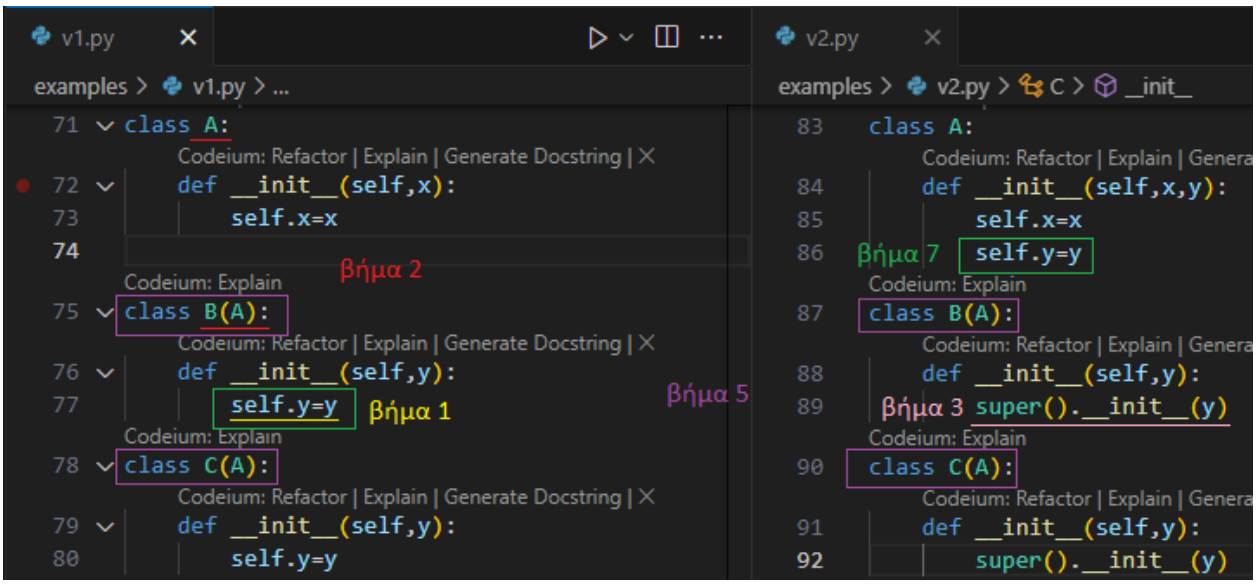
Εικόνα 22 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR ENCAPSULATED COLLECTION

- **Ενθυλακωμένο πεδίο (Encapsulated field)**
  1. Εύρεση κλάσεων στις γραμμές κώδικα της πρώτης έκδοσης του αρχείου.
  2. Εύρεση της ίδιας κλάσης στις γραμμές κώδικα της δεύτερης έκδοσης του αρχείου.
  3. Σύγκριση των πεδίων του κονστράκτορα των δύο κλάσεων. Η σύγκριση αφορά αν το πεδίο είναι κοινό και στις δύο εκδόσεις με τη διαφορά ότι το όνομα του πεδίου της δεύτερης έκδοσης έχει προστεθεί στη αρχή του ο χαρακτήρας "_".
  4. Έλεγχος αν έχουν προστεθεί μέθοδοι στην κλάση της δεύτερης έκδοσης και περιέχουν τις λέξεις "Get" ή " get" και "Set" ή "set".



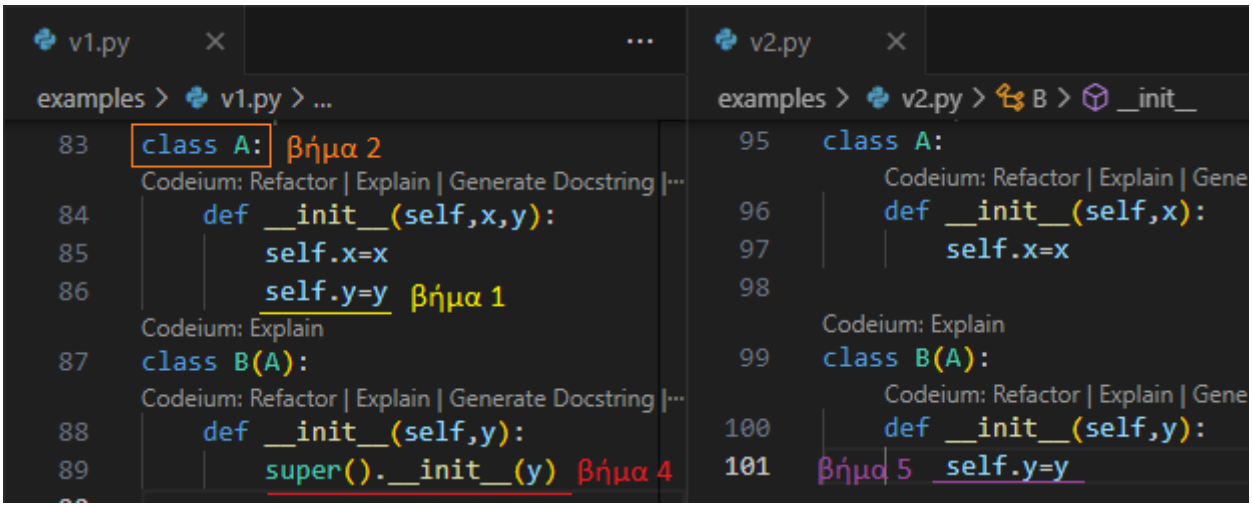
Εικόνα 23 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR ENCAPSULATED FIELD

- Αντιμετώπιση γενικεύσεων (Dealing with Generalizations)
  - **Pull Up Field**
    1. Εύρεση ορισμών μεταβλητών (variable declaration) οι οποίοι είναι πεδία κλάσης (ως class B από v1 και το πεδίο var_y).
    2. Έλεγχος για το αν η κλάση που ανήκει το πεδίο δεν είναι υπέρ κλάση. Η class B έχει υπέρ κλάση την class A.
    3. Έλεγχος στην ίδια μεταβλητή στην δεύτερη έκδοση αν ξεκινάει με “super().__init__(“ (class B από v2).
    4. Έλεγχος αν η υπέρ κλάση της class B από v1 δεν περιέχει το πεδίο.
    5. Έλεγχος αν υπάρχουν παραπάνω από μια υπό κλάση με το ίδιο πεδίο και σύγκριση με τις αντίστοιχες κλάσεις της δεύτερης έκδοσης για την ύπαρξη αυτού (class B και class C).
    6. Έλεγχος στην έκδοση 2 του αρχείου αν δεν υπάρχει το ίδιο πεδίο(var_y) στις υπό κλάσεις(class B και class C από v2) με την ίδια υπέρ κλάση τους (class A από v2).
    7. Έλεγχος για την ύπαρξη του αρχικού πεδίου (var_y από class B της v1) στην υπέρ κλάση της δεύτερης έκδοσης(class A από v2).



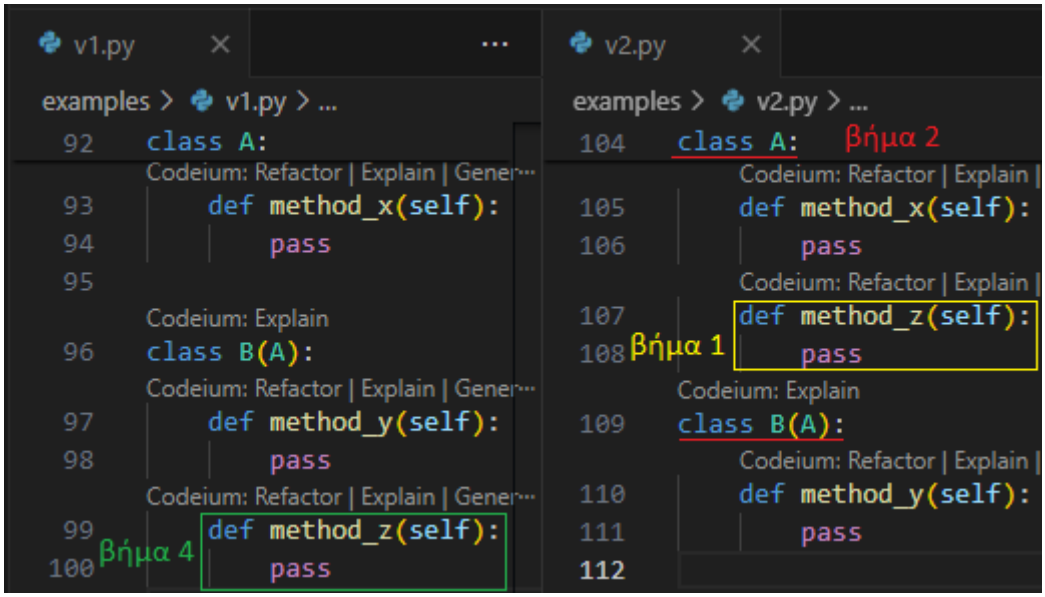
Εικόνα 24 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR PULL UP FIELD

- **Push down Field**
  1. Εύρεση ορισμών μεταβλητών (variable declaration) οι οποίοι είναι πεδία κλάσης (ως class A από v1 και το πεδίο var_y).
  2. Έλεγχος για το αν η κλάση που ανήκει το πεδίο είναι υπέρ κλάση.
  3. Έλεγχος της αντίστοιχης γραμμής κώδικα της δεύτερης έκδοσης αν δεν υπάρχει το πεδίο(var_y). Πραγματοποιείται δηλαδή έλεγχος αν το πεδίο δεν υπάρχει στη δεύτερη έκδοση του αρχείου.
  4. Έλεγχος στις υπό κλάσεις της class A από v1 αν περιέχεται το πεδίο με την μορφή "super().__init__(" .
  5. Έλεγχος στις υπό κλάσεις της class A από v2 αν περιέχεται το πεδίο, όχι με την μορφή "super().__init__(" αλλά με την κανονική μορφή του.



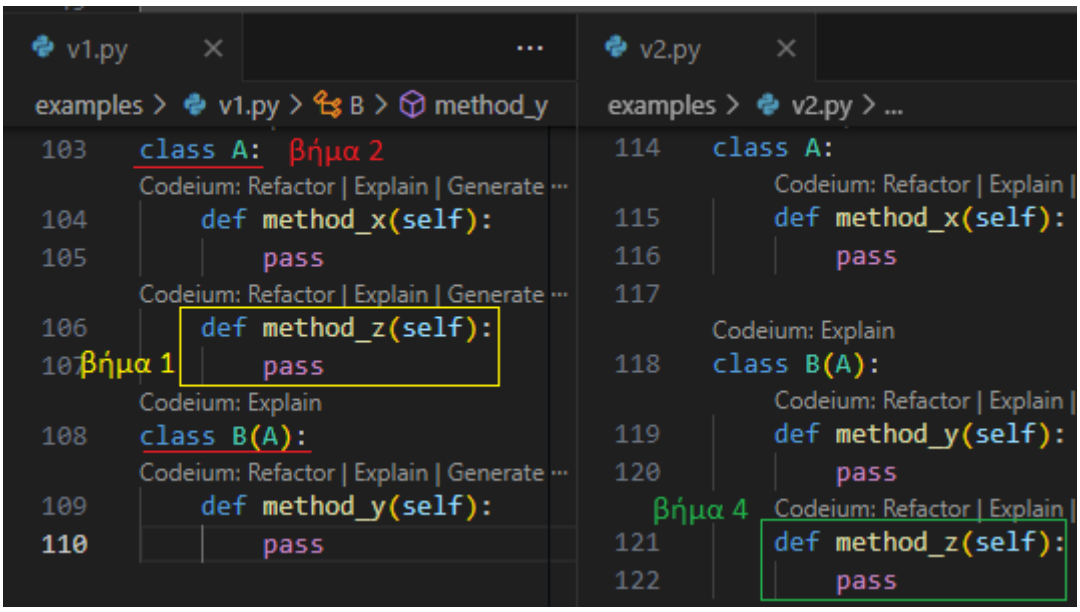
Εικόνα 25 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR PUSH DOWN FIELD

- **Pull up Method**
    1. Εύρεση συναρτήσεων οι οποίες είναι μέθοδοι κλάσεων στην δεύτερη έκδοση του αρχείου(ως method_z της class A από v2).
    2. Έλεγχος αν η κλάση class A από v2 είναι υπέρ κλάση και οι υπό κλάσεις της δεν περιέχουν την ίδια μέθοδο (method_z).
    3. Έλεγχος αν η μέθοδος(method_z) δεν υπάρχει στην κλάση της πρώτης έκδοσης (class A από v1).
    4. Έλεγχος αν η μέθοδος method_z της class A από v2 υπάρχει σε κάποια από τις υπό κλάσεις της class A από v1.



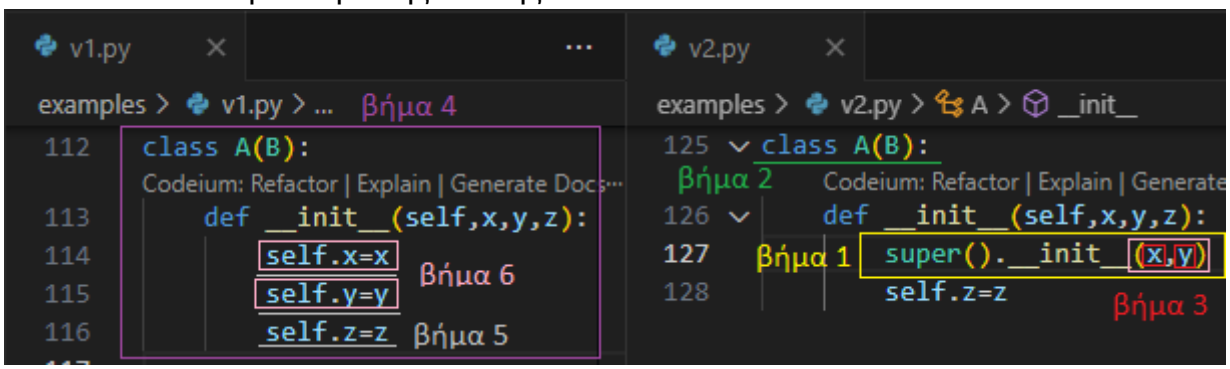
Εικόνα 26 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR PULL UP METHOD

- **Push down Method**
    1. Εύρεση συναρτήσεων οι οποίες είναι μέθοδοι κλάσεων στην πρώτη έκδοση του αρχείου(ως method_z της class A από v1).
    2. Έλεγχος αν η κλάση class A από v1 είναι υπέρ κλάση και οι υπό κλάσεις της δεν περιέχουν την ίδια μέθοδο (method_z).
    3. Έλεγχος αν η μέθοδος(method_z) δεν υπάρχει στην κλάση της δεύτερης έκδοσης (class A από v2).
    4. Έλεγχος αν η μέθοδος method_z της class A από v1 υπάρχει σε κάποια από τις υπό κλάσεις της class A από v2.



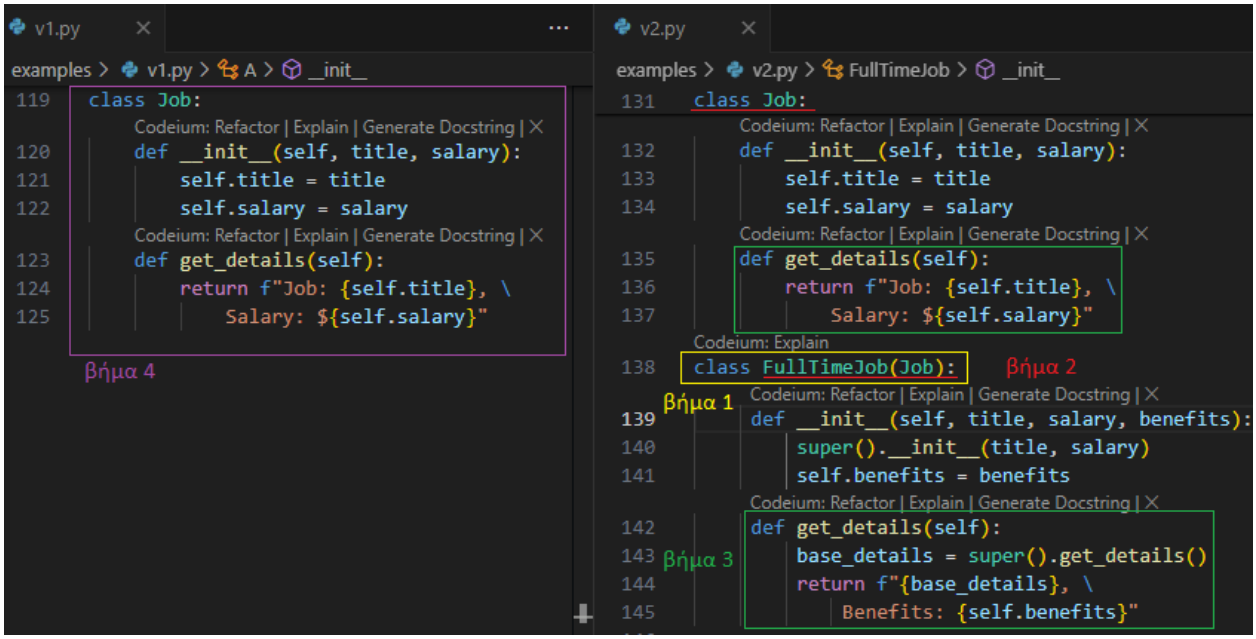
Εικόνα 27 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR PUSH DOWN METHOD

- **Pull up constructor Body**
  1. Εύρεση γραμμών εντός του κώδικα της δεύτερης έκδοσης του αρχείου που ξεκινούν με "super().__init__(".
  2. Έλεγχος αν η γραμμή αυτή ανήκει σε κάποια κλάση(ως class A από v2) και αν αυτή η κλάση είναι υπό κλάση κάποιας άλλης.
  3. Εξαγωγή των παραμέτρων εντός των παρενθέσεων της γραμμής που εντοπίστηκε.
  4. Εύρεση της ίδιας κλάσης class A v2 στην πρώτη έκδοση(ως class A από v1).
  5. Εύρεση των παραμέτρων του κονστράκτορα της κλάσης class A από v1.
  6. Σύγκριση και έλεγχος ύπαρξης των παραμέτρων της γραμμής με αυτές του κονστράκτορα της κλάσης class A από v1.



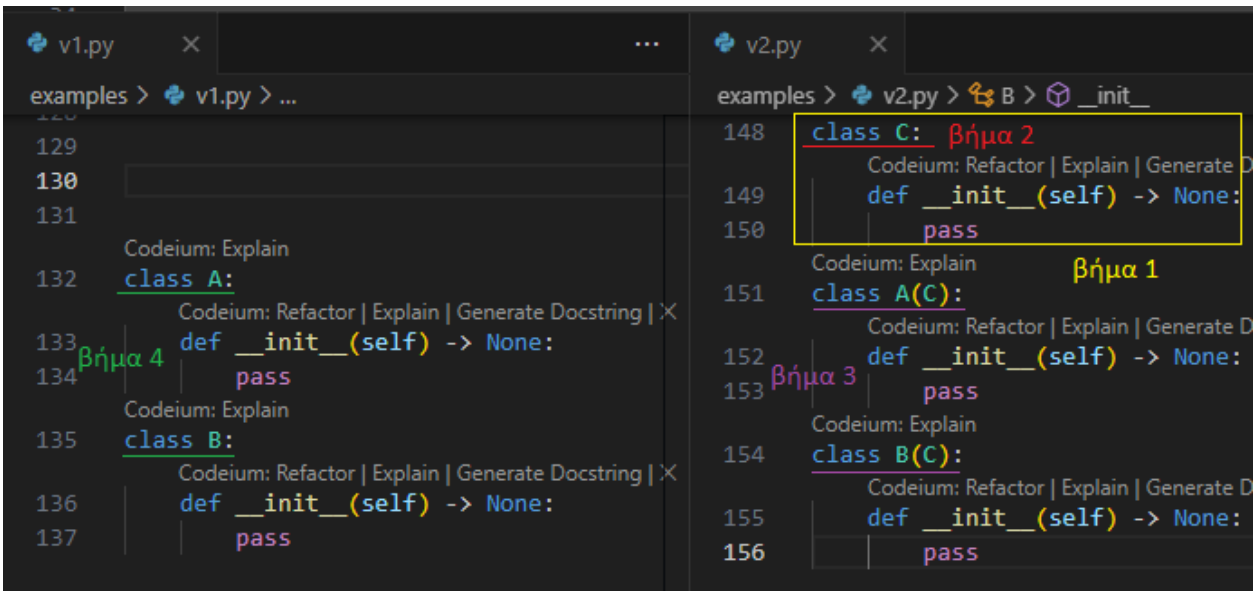
Εικόνα 28 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR PULL UP CONSTRUCTOR BODY

- **Εξαγωγή υπό κλάσης (Extract subclass)**
    1. Εύρεση κλάσεων που ανήκουν στην λίστα με τις νεοεισαχθείσες κλάσεις στην δεύτερη έκδοση του αρχείου(ως class B από v2).
    2. Έλεγχος αν η κλάση class B από v2 έχει υπέρ κλάση (ως class A από v2).
    3. Εύρεση και σύγκριση κοινών μεθόδων των δύο κλάσεων class B και class A από v2.
    4. Έλεγχος για την ύπαρξη τηςυπέρ κλάσης class A από v2 στην πρώτη έκδοση.



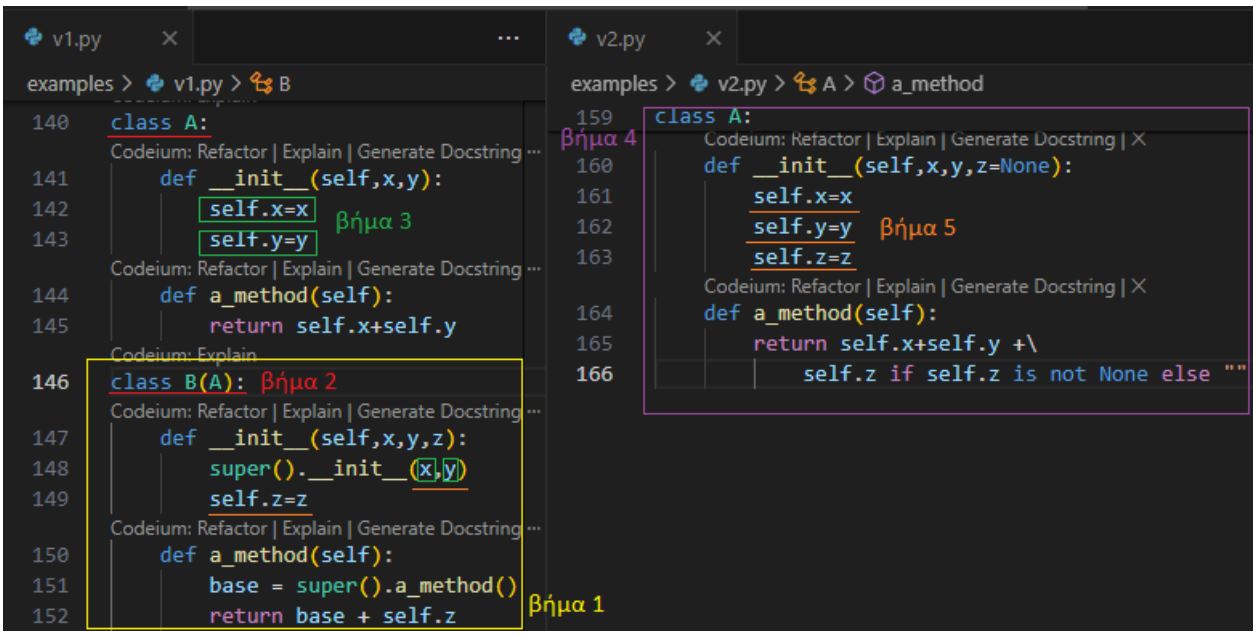
Εικόνα 29 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR EXTRACT SUBCLASS

- **Εξαγωγή υπέρ κλάσης (Extract superclass)**
    1. Εύρεση κλάσεων που ανήκουν στην λίστα με τις νεοεισαχθείσες κλάσεις στην δεύτερη έκδοση του αρχείου(ως class A από v2).
    2. Έλεγχος αν η κλάση class A από v2 είναι υπέρ κλάση.
    3. Εύρεση υπό κλάσεων που έχουν υπέρ κλάση την class A από v2(ως class B από v2).
    4. Έλεγχος αν η υπό κλάση class B από v2 είναι υπέρ κλάση στην πρώτη έκδοση.



Εικόνα 30 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR EXTRACT SUPERCLASS

- **Κατάρρευση ιεραρχίας (Collapse Hierarchy)**
    1. Εύρεση κλάσεων οι οποίες ανήκουν στη λίστα με τις διαγραμμένες κλάσεις στην πρώτη έκδοση του αρχείου (ως class B από v1).
    2. Έλεγχος αν η κλάση class B από v1 έχει υπέρ κλάση(ως class A από v1).
    3. Εύρεση και σύγκριση των πεδίων του κονστράκτορα της υπέρ κλάσης class A από v1 και της υπό κλάσης class B από v1.
    4. Έλεγχος ύπαρξης της αντίστοιχης υπέρ κλάσης στην δεύτερη έκδοση (ως class A από v2).
    5. Σύγκριση των πεδίων του κονστράκτορα της υπέρ κλάσης class A από v2 και της υπό κλάσης class B από v1.



Εικόνα 31 : ΒΗΜΑΤΑ ΕΥΡΕΣΗΣ ΤΟΥ REFACTOR COLLAPSE HIERARCHY

Κεφάλαιο 4: Υλοποίηση θέματος

Έχοντας αναλύσει τον τρόπο λειτουργίας του λογισμικού, ακολουθούν η εύρεση και η επιλογή πλήθος έργων γραμμένων σε Python που αφορούν την είσοδο του λογισμικού αλλά και τον τρόπο εκτέλεσής του.

4.1 Λεπτομέρειες υλοποίησης

Ως είσοδος του λογισμικού επιλέχθηκαν 209 έργα με 5368 συνολικό αριθμό εκδόσεων τα οποία είναι γραμμένα σε Python. Το βασικό κριτήριο για την επιλογή των έργων αφορά το πλήθος εκδόσεων που έχει το εκάστοτε έργο στο αποθετήριο του Github. Επιλέχθηκαν έργα τα οποία έχουν παραπάνω από 10 εκδόσεις ενώ χρησιμοποιήθηκαν, σαν μέγιστο, οι τελευταίες 30 εκδόσεις του κάθε έργου, όπου αυτό ήταν δυνατό. Κάποια έργα που επιλέχθηκαν είναι μεγάλου βεληνεκούς, όπως είναι το Openpilot[5], το Supervision[6] , το Home Assistant[7] κ.α.
Για την αυτοματοποίηση της διαδικασίας γράφτηκαν βοηθητικά script, τα οποία επιτρέπουν την αυτοματοποιημένη λήψη των εκδόσεων για το πλήθος των έργων από το αποθετήριο του Github, την επαναλαμβανόμενη εκτέλεση του λογισμικού για κάθε έργο και την δημιουργία των συγκεντρωτικών αποτελεσμάτων. Η λειτουργία των script που αναφέρθηκαν περιγράφεται αναλυτικά στα παραρτήματα.

4.2 Εκτέλεση του έργου σε cloud provider

Για την εκτέλεση του όλου έργου επιλέχθηκε να πραγματοποιηθεί απομακρυσμένα μέσω ενός παρόχου υπηρεσιών νέφους(cloud provider). Η επιλογή αυτή έγινε με σκοπό την εκμετάλλευση της δυνατότητας παράλληλης επεξεργασίας του προγράμματος (συστήματα με περισσότερους και γρηγορότερους πυρήνες) αλλά και την μεγαλύτερη ταχύτητα σύνδεσης την οποία παρέχουν. Αυτό είχε ως αποτέλεσμα, την γρηγορότερη λήψη των έργων από το αποθετήριο του Github, την ταχύτερη εκτέλεση αλλά και την εξαγωγή αποτελεσμάτων. Για την εγκατάσταση των προ απαιτούμενων του έργου χρησιμοποιήθηκε το bash script που δημιουργήθηκε. Το περιεχόμενο του script αυτού περιγράφεται στο αντίστοιχο κεφάλαιο

4.2.1 Σύστημα

Επιλέχθηκε ένα σύστημα που αποτελείται από 8 αποκλειστικούς πυρήνες, 16 Gb μνήμης ram και ταχύτητα σύνδεσης ίση με 40 Gbps κατεβάσματος και 6 Gbps ανεβάσματος. Ως λειτουργικό του συστήματος επιλέχθηκε το λειτουργικό Linux ονόματος Debian έκδοσης 11. Ο χειρισμός του συστήματος πραγματοποιήθηκε μέσω τερματικού ssh.

[5] https://github.com/commaai/openpilot
[6] https://github.com/roboflow/supervision
[7] https://github.com/home-assistant/core

4.2.2 Διαδικασία εκτέλεσης

Ως πρώτο βήμα της διαδικασίας εκτέλεσης αποτέλεσε η εκτέλεση του bash script **initiate_refactors_project.sh.** Μετά την επιτυχή εκτέλεση αυτού, στον φάκελο projects που δημιουργήθηκε από ίδιο, περιέχονται φάκελοι με όλα τα projects της λίστας εισαγωγής μαζί με τις κυκλοφορίες/εκδόσεις αυτών. Στη συνέχεια ακολούθησε η εκτέλεση του βοηθητικού script **run_for_all_the_projects.py,** το οποίο εκτελεί το βασικό πρόγραμμα με σκοπό την εύρεση Refactors μεταξύ των κυκλοφοριών/εκδόσεων όλων των έργων εντός του φακέλου projects. Έπειτα εκτελέστηκε το βοηθητικό script **merge_excels.py** ώστε να δημιουργηθεί το συγκεντρωτικό excel αρχείο με τα αποτελέσματα. Ακολούθησε η συμπίεση των αποτελεσμάτων του φακέλου **Results.** Τέλος, έγινε λήψη του συμπιεσμένου zip αρχείου και του συγκεντρωτικού excel αρχείου.

Κεφάλαιο 5: Αποτελέσματα και Συμπεράσματα

Στο κεφάλαιο αυτό γίνεται αναφορά στα ευρήματα που προέκυψαν μετά την εκτέλεση του λογισμικού αλλά και στα συμπεράσματα που προκύπτουν από αυτά. Η παρουσίαση των αποτελεσμάτων γίνεται με την χρήση πινάκων και γραφημάτων.

5.1 Αποτελέσματα

Πίνακας 1 : Συνοπτικός πίνακας ευρημάτων

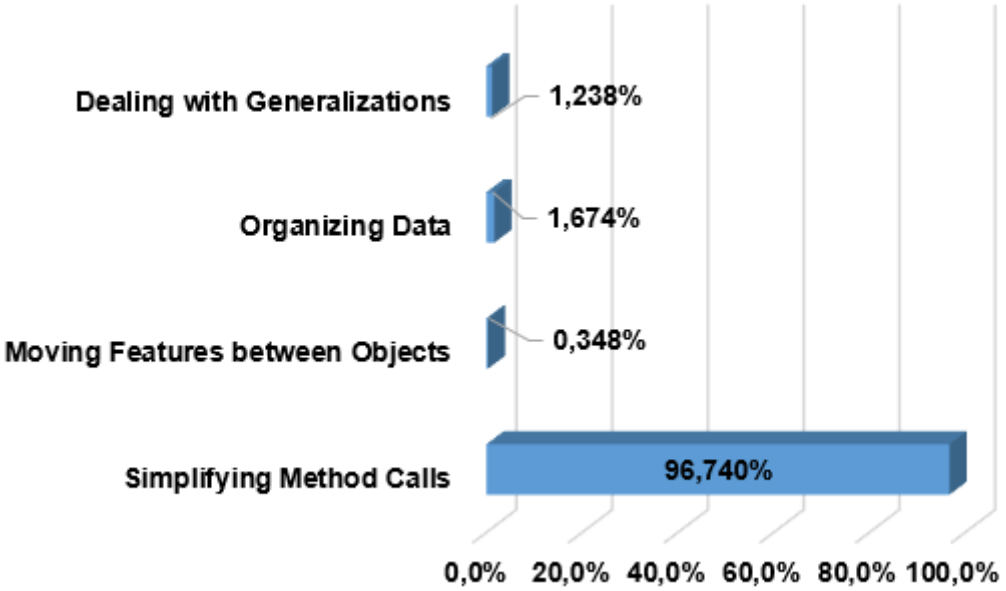
	Πλήθος
Projects	209
Releases	5368
Comparable files	199611
Refactors	70422

Ο Πίνακας 1 κάνει αναφορά στα ποσοτικά μεγέθη που προέκυψαν μετά την εκτέλεση του λογισμικού για τα 209 έργα που επιλέχθηκαν. Τα στοιχεία δείχνουν ότι υπήρξαν 199611 διαφορετικά αρχεία τα οποία φαίνεται να παρουσίασαν αλλαγές μεταξύ των εκδόσεων. Μεταξύ των συγκρίσιμων αρχείων εντοπίστηκαν 70422 διαφορετικά Refactors. Ο Πίνακας 2 παρουσιάζει το αριθμό των Refactors ανά κατηγορία και τύπο.

Πίνακας 2 : Ποσοτικός πίνακας των Refactors που ευρέθησαν

Κατηγορία	**Τύπος**	**Πλήθος**	**Πλήθος ανά κατηγορία**
Simplifying Method Calls			68126
	Add/Remove Parameter	47608	
	Rename Variable	19398	
	Preserve Whole Object	0	
	Introduce Parameter Object	0	
	Remove Setting Method	1075	
	Hide Method	9	
	Replace ErrorCode With Exception	1	
	Replace Exception With Test	35	
Moving Features between Objects			245
	Move Method	94	
	Extract Class	27	
	Inline Class	55	
	Move Field	69	
Organizing Data			1179

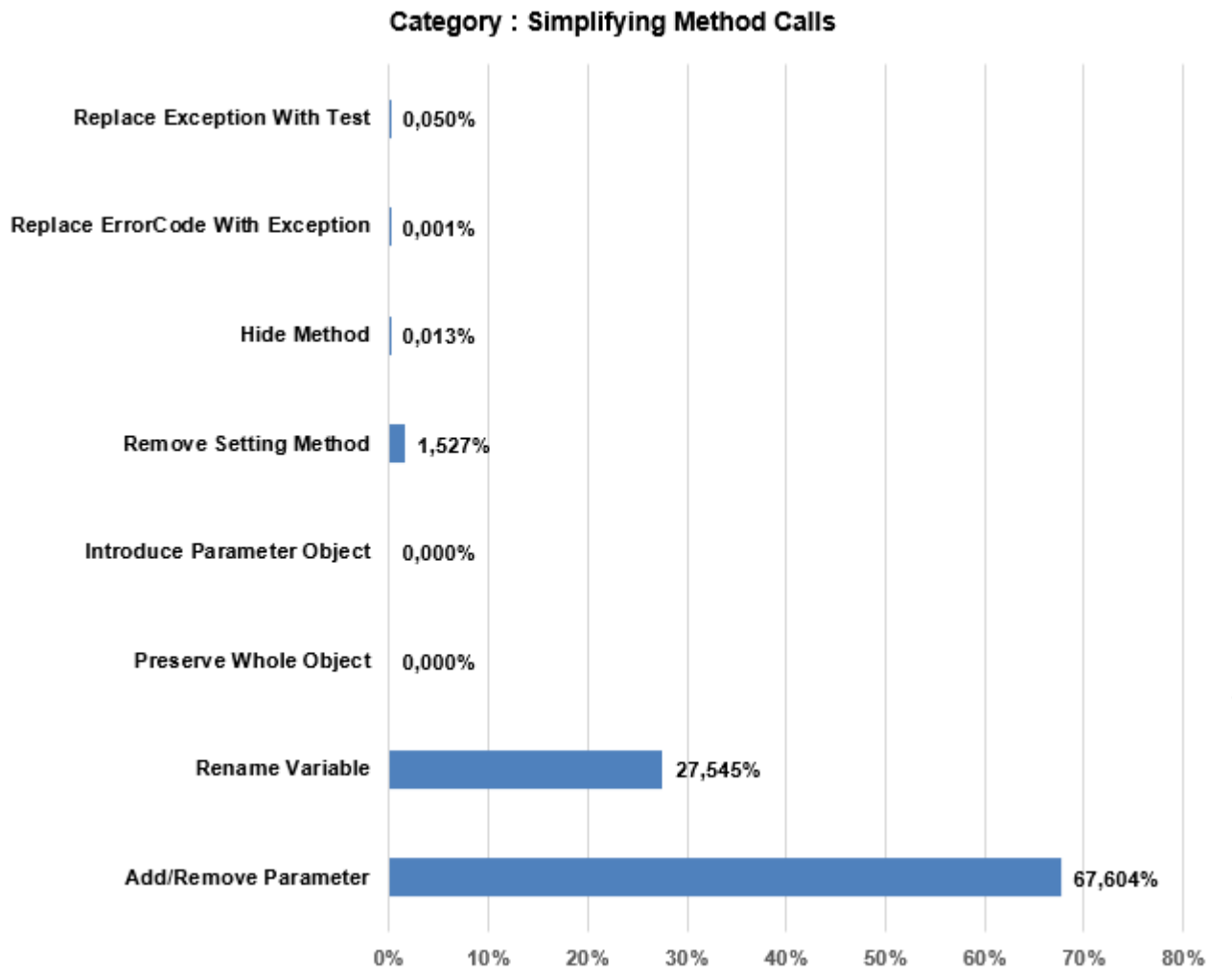
	Replace Array With Object	43	
	Replace Magic Number With Symbolic Constant	1007	
	Encapsulated Collection	0	
	Encapsulated Field	129	
Dealing with Generalizations			872
	Pull Up Field	1	
	Push Down Field	7	
	Pull Up Method	38	
	Push Down Method	8	
	Pull Up Constructor Body	25	
	Extract Subclass	739	
	Extract Superclass	29	
	Collapse Hierarchy	25	



Σχήμα 2 : Ποσοστό του πλήθους των Refactors που ευρέθησαν ανά κατηγορία

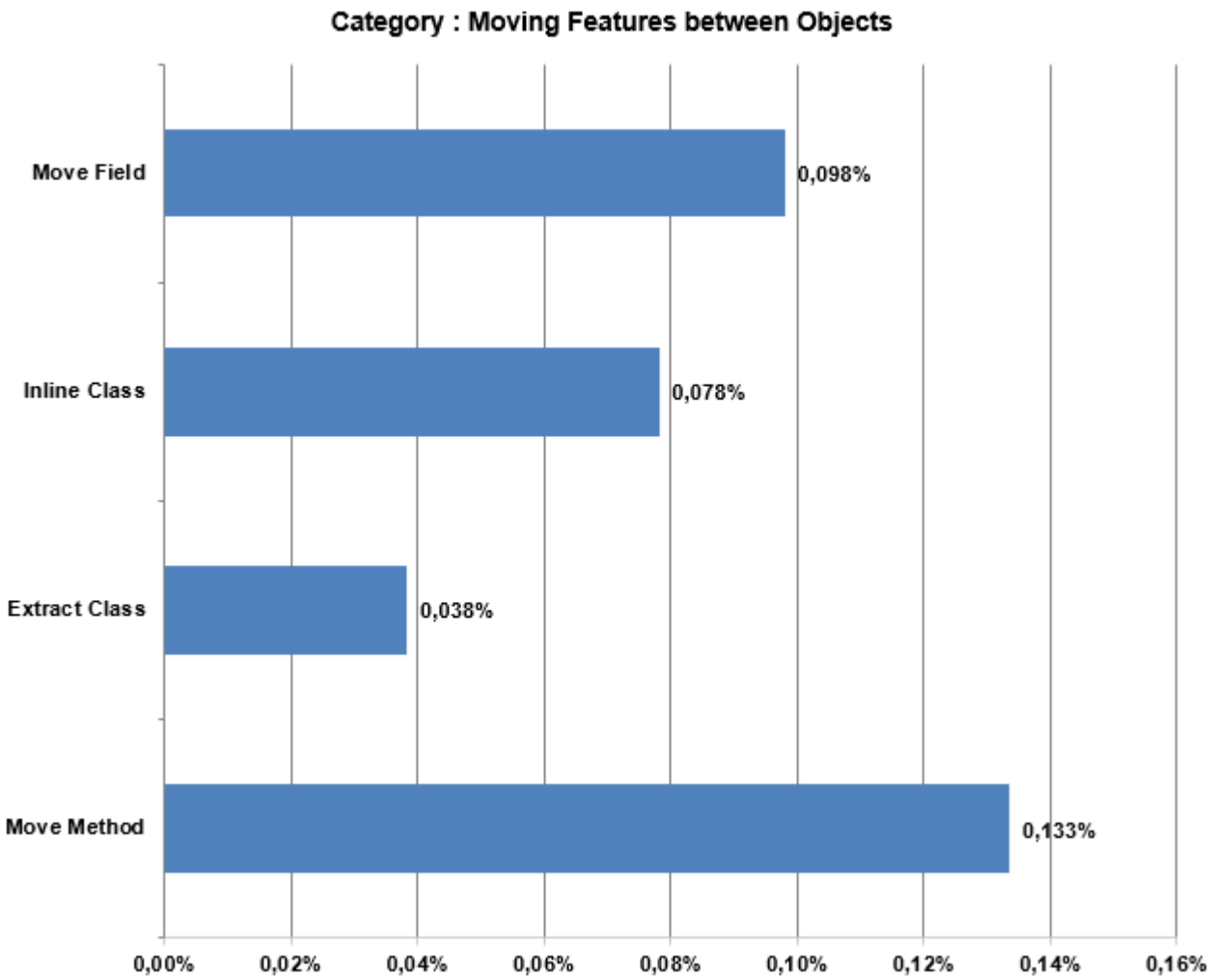
Το παραπάνω γράφημα κάνει ορατή την υπεροχή της κατηγορίας Simplifying Method Calls μιας και από τα 70422 συνολικά Refactors που εντοπίστηκαν τα 68126 ανάγονται σε αυτήν, φτάνοντας το ποσοστό συμμετοχής στο 96,74%. Οι υπόλοιπες κατηγορίες φαίνονται να έρχονται σε δεύτερη μοίρα αφού τα ποσοστά συμμετοχής τους είναι χαμηλά.

Τα Σχήματα 3,4,5,6 περιέχουν τα ποσοστά συμμετοχής των τύπων των Refactors της κάθε κατηγορίας ξεχωριστά, σε σχέση με το πλήθος των συνολικών Refactors που ευρέθησαν.



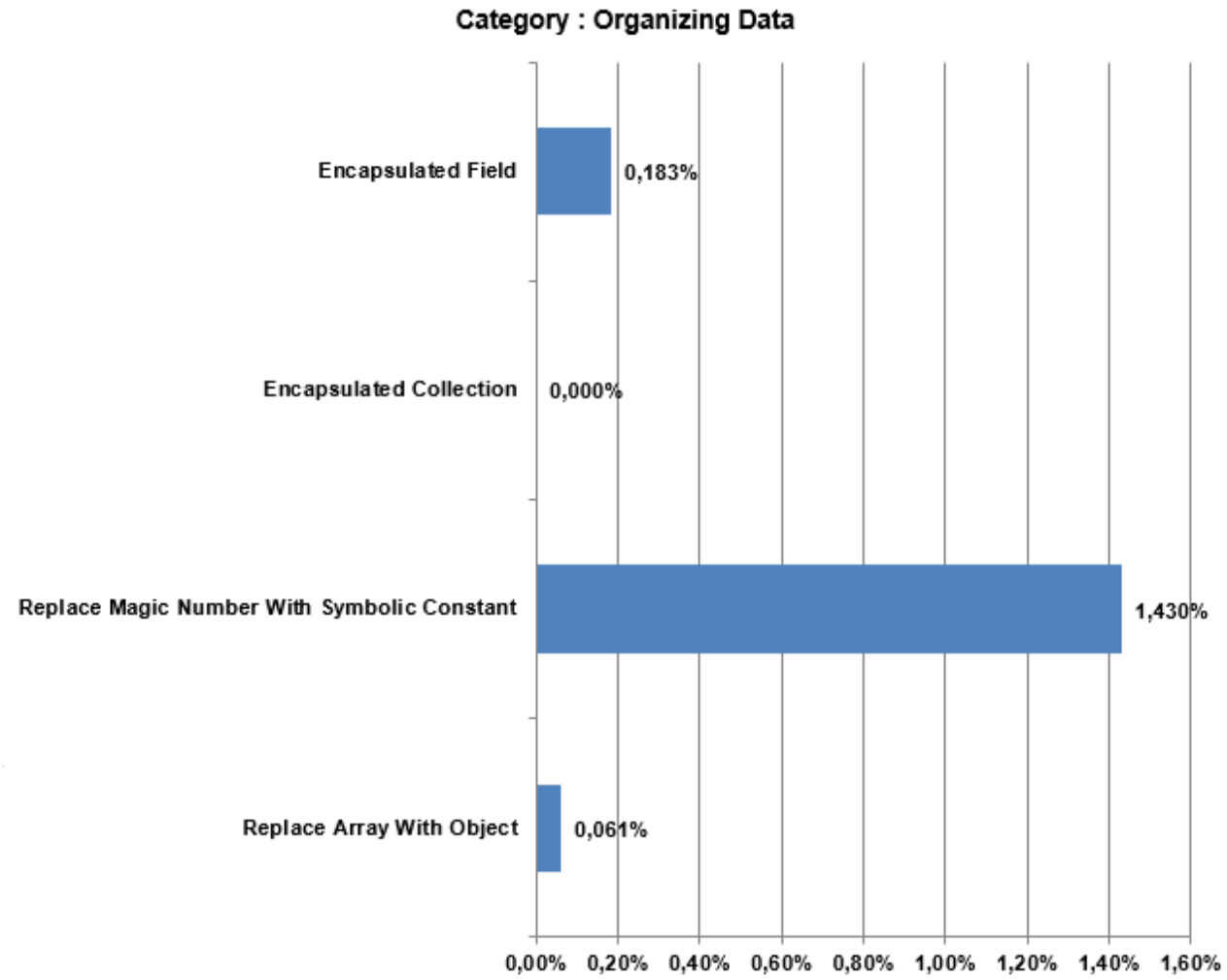
Σχήμα 3 : Ποσοστό του πλήθους των Refactors για την κατηγορία Simplifying Method Calls

Στο Σχήμα 3 είναι εμφανές ότι το Refactor Add/Remove Parameter είναι το επικρατέστερο, όχι μόνο για αυτήν την κατηγορία αλλά και γενικά, έχοντας ποσοστό ίσο με 67,604%. Βρέθηκαν συνολικά 47608 για αυτόν τον τύπο. Ακολουθεί το Refactor Rename Variable με ποσοστό 27,545%. Παρατηρείται ότι για τα Refactors Introduce Parameter Object και Preserve Whole Object δεν βρέθηκε κανένα Refactor που να τα αφορά, ενώ για τα υπόλοιπα της κατηγορίας εντοπίστηκαν ελάχιστα.



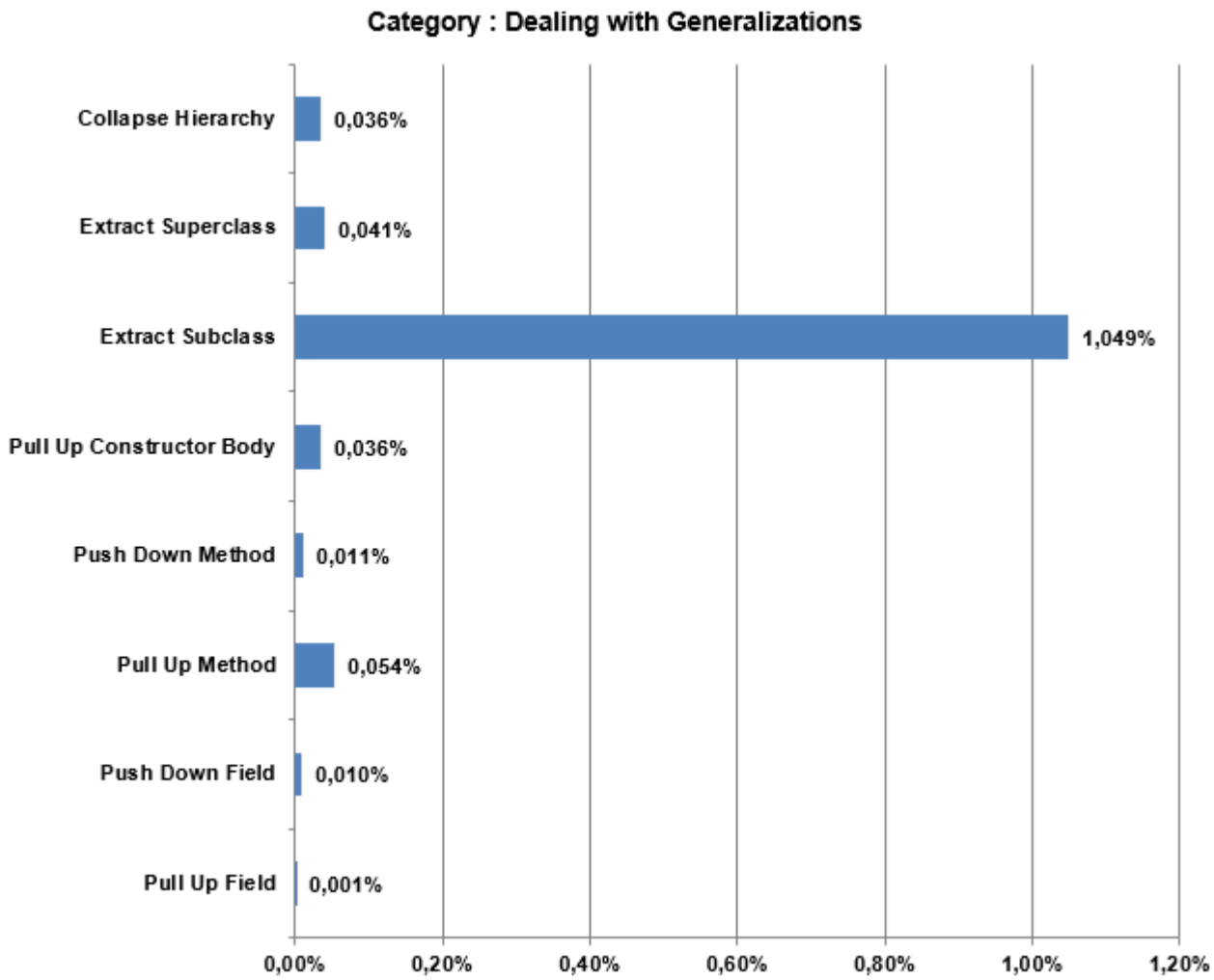
Σχήμα 4: Ποσοστό του πλήθους των Refactors για την κατηγορία Moving Features Between Objects

Η κατηγορία που αναφέρεται στο Σχήμα 4, φαίνεται να είναι και η κατηγορία με τα λιγότερα Refactors που εντοπίστηκαν. Το πλήθος των Refactors που την αφορούν, δεν ξεπέρασαν τα 245.Έτσι τα ποσοστά που συγκέντρωσαν οι τύποι της, είναι αρκετά μικρά.



Σχήμα 5 : Ποσοστό του πλήθους των Refactors για την κατηγορία Organizing Data

Τα δεδομένα του Σχήματος 5 παρουσιάζουν ότι δεν βρέθηκε κανένα Refactor του τύπου Encapsulated Collection. Το Refactor Replace Magic Number with Symbolic Constant παρατηρείται ότι προέκυψε 1007 φορές συμπληρώνοντας το ποσοστό 1,430%. Τα Refactor Replace Arry with Object και Encapsulated Field συγκέντρωσαν από 0,061% και 0,183% αντιστοίχως.



Σχήμα 6 : Ποσοστό του πλήθους των Refactors για την κατηγορία Dealing with Generalizations

Το Σχήμα 6 δείχνει ότι το επικρατέστερο Refactor της κατηγορίας αυτής είναι το Extract Subclass με σύνολο 1,049%, ενώ το λιγότερο επικρατέστερο να είναι το Refactor Pull Up field με 0.001% έχοντας εντοπιστεί μόλις μια φορά. Οι υπόλοιποι τύποι Refactor φτάνουν ποσοστά από 0,010% μέχρι 0,036%.

Ο Πίνακας 3 περιέχει τα πρώτα 30 έργα με τα περισσότερα Refactors που εντοπίστηκαν. Στον πίνακα αυτόν παρατηρείται, όπως και προηγουμένως, η κυριαρχία του Add/Remove Parameter σαν τύπος Refactor με τη μεγαλύτερη συχνότητα εμφάνισης στην πλειοψηφία των έργων.

Πίνακας 3 : Τα 30 πρώτα έργα με τα περισσότερα Refactors που εντοπίστηκαν

Όνομα του δημιουργού και του έργου στο Github	Πλήθος Refactors	Τύπος Refactor με τη μεγαλύτερη συχνότητα εμφάνισης
getsentry_sentry	5657	Add/Remove Parameter
sympy_sympy	5452	Add/Remove Parameter
pandas-dev_pandas	5422	Add/Remove Parameter
localstack_localstack	3658	Add/Remove Parameter
ibis-project_ibis	2645	Add/Remove Parameter
zulip_zulip	1770	Add/Remove Parameter
commaai_openpilot	1629	Add/Remove Parameter
pola-rs_polars	1591	Add/Remove Parameter
tensorflow_models	1432	Add/Remove Parameter
dagster-io_dagster	1359	Add/Remove Parameter
mongodb_mongo-python-driver	1268	Add/Remove Parameter
sqlalchemy_sqlalchemy	1213	Add/Remove Parameter
matplotlib_matplotlib	1188	Add/Remove Parameter
quantopian_zipline	1071	Add/Remove Parameter
hi-primus_optimus	1028	Add/Remove Parameter
dynaconf_dynaconf	995	Add/Remove Parameter
great-expectations_great_expectations	991	Add/Remove Parameter
elastic_elasticsearch-py	900	Add/Remove Parameter
kedro-org_kedro	900	Add/Remove Parameter
pydata_xarray	864	Add/Remove Parameter
zenml-io_zenml	850	Add/Remove Parameter
modin-project_modin	748	Add/Remove Parameter
redis_redis-py	748	Add/Remove Parameter
keras-team_keras	679	Rename Variable
pytorch_vision	664	Add/Remove Parameter
aws_aws-sdk-pandas	618	Add/Remove Parameter
neo4j_neo4j-python-driver	617	Add/Remove Parameter
Legrandin_pycryptodome	615	Add/Remove Parameter
databricks_koalas	578	Rename Variable
openai_gym	524	Add/Remove Parameter

5.2 Συμπεράσματα

Συνοψίζοντας, τα δεδομένα δείχνουν ότι η κυρίαρχη κατηγορία Refactor που εφαρμόστηκε είναι η Simplifying Method Calls με συνολικό ποσοστό 96.7%. Ο δημοφιλέστερος τύπος Refactor που συναντήθηκε είναι ο Add/Remove Parameter με ποσοστό 67.6% ενώ στη συνέχεια ακολουθεί το Rename Variable με ποσοστό 27.5%. Επίσης παρατηρήθηκε ότι για τους τύπους Refactor Preserve Whole Object, Introduce Parameter Object και Encapsulate Collection δεν εντοπίστηκε κανένα ενώ για τους τύπους Replace ErrorCode With Exception και Pull Up Field εντοπίστηκε μόνο από ένα.

Η κυριαρχία του Refactor Add/Remove Parameter και γενικά της κατηγορίας Simplifying Method Calls μπορεί να ερμηνευτεί από τις εξής παρατηρήσεις. Η συγκεκριμένη κατηγορία χαρακτηρίζεται από την απλότητα της εφαρμογής των τεχνικών σε σχέση με τις υπόλοιπες. Έτσι προκύπτουν περισσότερες περιπτώσεις εφαρμογής της από τους προγραμματιστές. Επίσης η απλότητα της κατηγορίας αυτής καθιστά τον τρόπο εντοπισμού αυτών των τύπων Refactor πιο ξεκάθαρο βοηθώντας στην εύρεση περισσοτέρων Refactors. Από την άλλη, η απώλεια εύρεσης των τύπων Refactor που δεν συναντήθηκαν καμία ή μια φορά μπορεί να δικαιολογηθεί λόγω των μειονεκτημάτων που προκύπτουν από την εφαρμογή τους. Η εφαρμογή του Refactor Preserve Whole Object για παράδειγμα έχει σαν μειονέκτημα ότι οι μέθοδοι του κώδικα που επαναναγράφονται, να γίνονται λιγότερο ευέλικτες. Μειονεκτήματα σαν και αυτό καθιστούν την συνάντηση αυτών των τύπων Refactor αρκετά σπάνια.

Τα παραπάνω στοιχεία που συγκεντρώθηκαν κάνουν εμφανή την εφαρμογή των διαφόρων τεχνικών Refactoring από τους δημιουργούς των έργων. Η πλειοψηφία των προγραμματιστών φαίνεται να μπαίνει στην διαδικασία του Refactoring στον κώδικά της με σκοπό να επωφεληθεί από αυτά. Τα δεδομένα αυτά δείχνουν την σημασία και τη σημαντικότητα των τεχνικών του Refactoring επιβεβαιώνοντας την προσοχή που δείχνει η επιστημονική κοινότητα σε αυτά. Σημαντικό ακόμη είναι να τονιστεί το γεγονός ότι το ποσοτικό πλήθος των αποτελεσμάτων που προέκυψαν έδωσε την ευκαιρία της παρατήρησης των δυνατότητων και ικανοτήτων του λογισμικού που δημιουργήθηκε.

Κεφάλαιο 6: Επίλογος

Στο κεφάλαιο αυτό περιγράφονται οι περιορισμοί και οι παραδοχές σχετικά με τα αποτελέσματα που εξήχθησαν. Ακόμη παρουσιάζονται οι μελλοντικές επεκτάσεις αλλά και η σημασία του έργου για την εύρεση Refactor σε έργα Python.

6.1 Περιορισμοί και Παραδοχές

Όπως προαναφέρθηκε η γλώσσα προγραμματισμού Python χαρακτηρίζεται από την ευελιξία αλλά και την δυνατότητα υλοποίησης λειτουργιών μέσω πολλαπλών τρόπων επίλυσης, καθιστώντας την ανάλυση του κώδικα αυτής αρκετά δύσκολη. Αν και κατά την ανάπτυξη του λογισμικού πραγματοποιούνταν ενδελεχείς έλεγχοι του κώδικα, κάποια Refactors που εντοπίζονται είναι ψευδοθετικά. Ο εντοπισμός των ψευδοθετικών Refactors είναι ιδιαίτερα δύσκολος αφού μπορεί να πραγματοποιηθεί μόνο χειροκίνητα. Λόγω του μεγάλου πλήθους των Refactors που εντοπίστηκαν, η επικύρωση όλων των αποτελεσμάτων είναι σχεδόν αδύνατη. Αυτό είχε ως αποτέλεσμα, να παρθεί η απόφαση να μην ληφθεί υπόψη η ύπαρξη των ψευδοθετικών Refactors στα αποτελέσματα που εξήχθησαν. Αν και η παραπάνω παραδοχή επηρεάζει αισθητά την εγκυρότητα των αποτελεσμάτων, δεν παύει να μειώνει το ενδιαφέρον των ευρημάτων που εντοπίστηκαν.

6.2 Μελλοντικές επεκτάσεις

Μια από τις σημαντικότερες μελλοντικές επεκτάσεις της παρούσας μελέτης αποτελεί η δημιουργία μιας διαδικασίας επικύρωσης των Refactors που εντοπίστηκαν. Ένας από τους τρόπους επικύρωσης θα μπορούσε να είναι η δημιουργία ενός script που θα έχει σαν είσοδο όλα τα html αρχεία που δημιουργήθηκαν για κάθε Refactor και θα τα εμφανίζει στον χρήστη διαδοχικά με σκοπό την εύρεση της εγκυρότητάς του. Μόλις ο χρήστης χαρακτηρίσει το αντίστοιχο Refactor σαν έγκυρο ή ψευδοθετικό θα το εισάγει σε μια βάση δεδομένων και θα προχωρά στο επόμενο μέχρι να ολοκληρωθεί η διαδικασία. Κάτι τέτοιο θα επιτρέψει την βελτίωση του υπάρχοντος λογισμικού αλλά και την εξαγωγή στατιστικών που θα αφορούν το ποσοστό των εγκύρων Refactors σε σχέση με αυτών των ψευδοθετικών. Σε συνέχεια της προηγούμενης πρότασης το φυσικό επακόλουθο της παρούσης μελέτης είναι η διεύρυνση του εύρους των τύπων Refactors που εντοπίζονται από το λογισμικό.

6.3 Σημασία του έργου

Μετά το πέρας της διπλωματικής εργασίας, το λογισμικό που δημιουργήθηκε, μπορεί να προστεθεί στα λιγοστά εργαλεία εύρεσης Refactoring για έργα γραμμένα σε γλώσσα Python, μιας και η παρούσα έκδοση του λογισμικού είναι σε θέση να εντοπίζει έναν ικανοποιητικό αριθμό τύπων Refactoring. Επίσης, ο βασικός κορμός του έργου είναι ικανός να τεθεί ως βάση για την επέκταση του λογισμικού, διευρύνοντας το πλήθος των Refactors που εντοπίζονται.

Παραρτήματα

Το παρόν κεφάλαιο περιέχει τον τρόπο λειτουργίας των βοηθητικών scripts που δημιουργήθηκαν. Ο σκοπός των scripts είναι η αυτοματοποίηση διαφόρων διαδικασιών του βασικού λογισμικού. Το script **download_releases_from_github.py** χρησιμοποιείται για τη λήψη πολλαπλών κυκλοφοριών/εκδόσεων πολλαπλών έργων από το αποθετήριο του Github. Το script **run_for_all_the_projects.py** χρησιμοποιείται για την εκτέλεση του βασικού λογισμικού έχοντας σαν είσοδο κάθε έργο που ελήφθη από το προηγούμενο script. Το script **merge_excels.py** πραγματοποιεί συνένωση των excel αρχείων που δημιουργήθηκαν για κάθε project μετά την επιτυχή εκτέλεση του προηγούμενου script. Η λειτουργία του κάθε script περιγράφεται παρακάτω. Επίσης, όπως αναφέρθηκε, η εκτέλεση του όλου έργου πραγματοποιήθηκε σε έναν cloud provider. Για την διευκόλυνση της εγκατάστασης των προ απαιτούμενων στο σύστημα που επιλέχθηκε, δημιουργήθηκε ένα bash script με όνομα **initiate_refactors_project.sh**. Το περιεχόμενο του εκάστοτε script περιγράφεται στη συνέχεια.

A. Script download_releases_from_github.py

Όπως προαναφέρθηκε, ο σκοπός αυτού του script είναι η λήψη πολλαπλών κυκλοφοριών/εκδόσεων πολλαπλών έργων από το αποθετήριο του Github. Η θέση του αρχείου κατά την εκτέλεσή του πρέπει να είναι εντός του φακέλου που είναι επιθυμητός για την λήψη των πολλαπλών έργων. Η λειτουργία του script προϋποθέτει την ύπαρξη ενός αρχείου **config.py** (εντός του ίδιου φακέλου) που να περιέχει τα εξής στοιχεία:

- **GITHUB_TOKEN = 'TOKEN' :** Η διεπαφή προγραμματισμού εφαρμογών (API) του Github έχει περιορισμό στις μη αυθεντικοποιημένες αιτήσεις όσον αφορά το πλήθος των λήψεων, 60 αιτήσεις/λήψης ανά ώρα. Για τον λόγο αυτό η δημιουργία ενός επικυρωμένου token είναι απαραίτητη. Η δημιουργία του authentication token γίνεται από τον σύνδεσμο https://github.com/settings/tokens επιλέγοντας ως δικαιώματα(permissions) το 'Read access of projects' και το 'Download packages from GitHub Package Registry'.
- **MAX_RELEASES_TO_DOWNLOAD = 30 :** Είναι ο μέγιστος αριθμός κυκλοφοριών που μπορούν να ληφθούν ανά έργο (project).

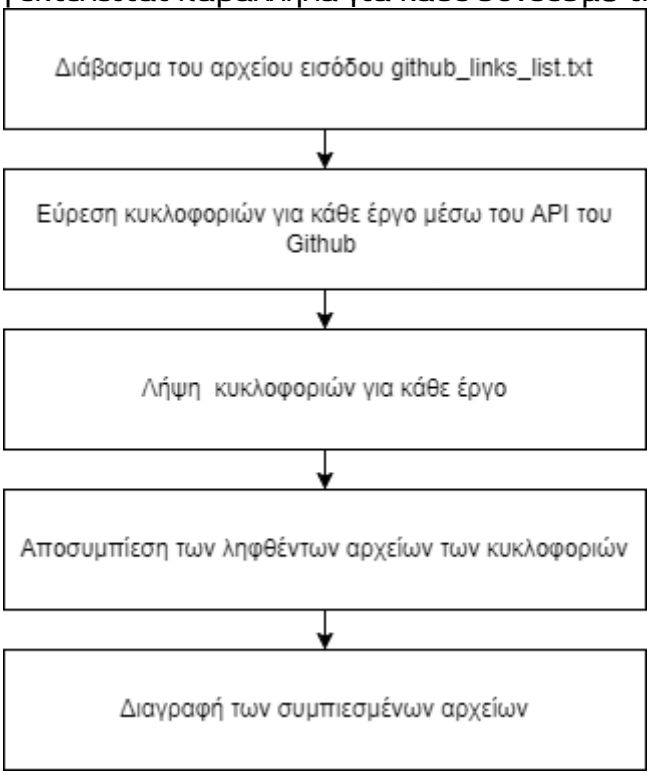
Το script έχει ως είσοδο ένα αρχείο κειμένου txt με όνομα **github_links_list.txt.** Αυτό το αρχείο περιέχει συνδέσμους με διάφορα Python projects όπου σε κάθε γραμμή του, αντιστοιχεί σε έναν σύνδεσμο ενός project.

Κατά την εκτέλεση του script διαβάζεται το αρχείο txt και τοποθετείται σε μια λίστα.

Για κάθε στοιχείο της λίστας δημιουργείται ένα διαφορετικό νήμα το οποίο εκτελεί τις παρακάτω ενέργειες:

- Κλήση στο API του Github με σκοπό την εύρεση ύπαρξης κυκλοφοριών του έργου.
- Λήψη κάθε κυκλοφορίας που ευρέθη, με περιορισμό τον ορισμό που έχει οριστεί στο αρχείο config.py.
- Αποσυμπίεση των ληφθέντων αρχείων και διαγραφή των συμπιεσμένων.

Η διαδικασία αυτή εκτελείται παράλληλα για κάθε σύνδεσμο της λίστας εισόδου.



Σχήμα 7 : Διαγραμματική απεικόνιση του script download_releases_from_github.py

B. Script run_for_all_the_projects.py

Ο σκοπός του script αυτού είναι η επαναλαμβανόμενη εκτέλεση του βασικού προγράμματος για κάθε έργο που έχει ληφθεί από το παραπάνω script. Η εκτέλεσή του, προϋποθέτει την ύπαρξη ενός config.py αρχείου που περιέχει το πεδίο PROJECT_FOLDER_PATH= r"path_to_projects_folder". Το πεδίο αφορά την διαδρομή του φακέλου με τα πολλαπλά έργα που έχουν ληφθεί.
Κατά την εκτέλεση του script διαβάζονται όλοι οι υποφάκελοι της διαδρομής που ορίστηκε στο config αρχείο και τοποθετούνται σε μια λίστα. Για κάθε υποφάκελο δημιουργείται και μια νέα διεργασία η οποία ανατίθεται σε κάποιον διαθέσιμο πυρήνα του συστήματος. Η διεργασία που δημιουργείται καλεί το βασικό πρόγραμμα με είσοδο την διαδρομή του κάθε υποφακέλου της παραπάνω λίστας.

C. Script merge_excels.py

Ο σκοπός του script αυτού είναι η δημιουργία ενός συγκεντρωτικού αρχείου excel που θα περιέχει όλα τα Refactors εντός των excel αρχείων που εξήγαγε η εκτέλεση του βασικού προγράμματος. Το συγκεντρωτικό αρχείο excel θα περιέχει επίσης στατιστικά, τα οποία δημιουργούνται αυτοματοποιημένα, για το σύνολο των έργων αλλά και για κάθε έργο ξεχωριστά. Η θέση του αρχείου κατά την εκτέλεσή του πρέπει να είναι στον ίδιο κατάλογο με τον φάκελο Results. Η λειτουργία του script αποτελείται από τις παρακάτω ενέργειες:

- Διάβασμα των υποφακέλων του φακέλου Results και τοποθέτηση αυτών σε λίστα.

- Για κάθε υποφάκελο ελέγχεται η ύπαρξη του excel αρχείου εξόδου και του αρχείου Execution.txt.
- Εξαγωγή των Refactors από το excel αρχείο και τοποθέτηση αυτών, στη σελίδα(sheet) με όνομα Refactors του συγκεντρωτικού excel.
- Εξαγωγή στοιχείων από το αρχείο Execution.txt. Τα στοιχεία που εξάγονται είναι :
    1. Το πλήθος των εκδόσεων/κυκλοφοριών του κάθε έργου.
    2. Το πλήθος των συγκρίσιμων αρχείων (η λίστα comparable_files του βασικού προγράμματος) του κάθε έργου.
    3. Τη λίστα ονομάτων των συγκρίσιμων αρχείων του κάθε έργου.
- Δημιουργία συγκεντρωτικών στατιστικών σε μορφή πινάκων, που αφορούν το σύνολο των έργων. Οι πίνακες αυτοί τοποθετούνται στη σελίδα(sheet) με όνομα "General stats".

  Ο πρώτος πίνακας περιέχει το πλήθος των Refactors ανά κατηγορία και τύπο όπως και το ποσοστό συμπλήρωσης ανά Refactor κατά το πλήθος των συνολικών Refactors.

  Ο δεύτερος πίνακας αφορά το πλήθος των συνολικών έργων, εκδόσεων, συγκρίσιμων αρχείων και Refactors.

  Ο τρίτος πίνακας περιέχει τα ονόματα των έργων, το πλήθος των Refactors και τον τύπο του Refactor που εμφανίστηκε περισσότερο στο κάθε έργο. Ο πίνακας είναι ταξινομημένος κατά φθίνουσα σειρά με βάση το πλήθος των Refactors.

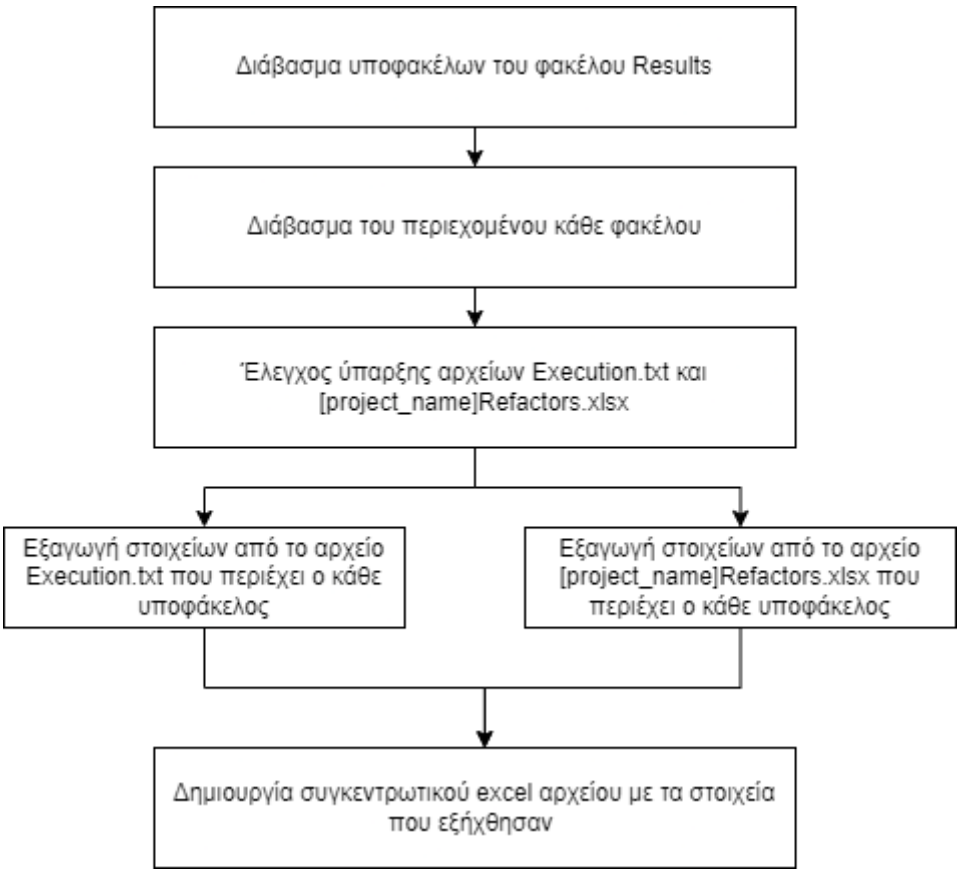
  Τέλος στην ίδια σελίδα δημιουργούνται γραφήματα σχετικά με τα παραπάνω στοιχεία.
- Δημιουργία στατιστικών, σε μορφή πίνακα για κάθε έργο ξεχωριστά.

  Ο πίνακας αυτός τοποθετείται στη σελίδα(sheet) με όνομα "Projects".

  Τα στατιστικά αφορούν το ποσοστό συμπλήρωσης ανά Refactor κατά το πλήθος των συνολικών Refactors για το κάθε διαφορετικό έργο.

  Ακόμα δημιουργείται ένας πίνακας που περιέχει τα στοιχεία :
    1. Το όνομα του έργου.
    2. Τον σύνδεσμο που οδηγεί στη σελίδα του έργου στο Github.
    3. Τα ονόματα και το πλήθος των εκδόσεων/κυκλοφοριών του συγκεκριμένου έργου.
    4. Το πλήθος των συγκρίσιμων αρχείων για το συγκεκριμένο έργο.
    5. Το ζευγάρι των εκδόσεων/κυκλοφοριών των οποίων βρέθηκαν τα περισσότερα Refactors.
    6. Το πλήθος των Refactors για το συγκεκριμένο έργο.

  Επίσης δεξιά των πινάκων αυτών, δημιουργούνται γραφήματα που αφορούν τα παραπάνω στοιχεία. Οι πίνακες και τα γραφήματα των διαφορετικών έργων τοποθετούνται διαδοχικά ο ένας κάτω από τον άλλον.



Σχήμα 8 : Διαγραμματική απεικόνιση του script merge_excels.py

D. Λειτουργία του bash script

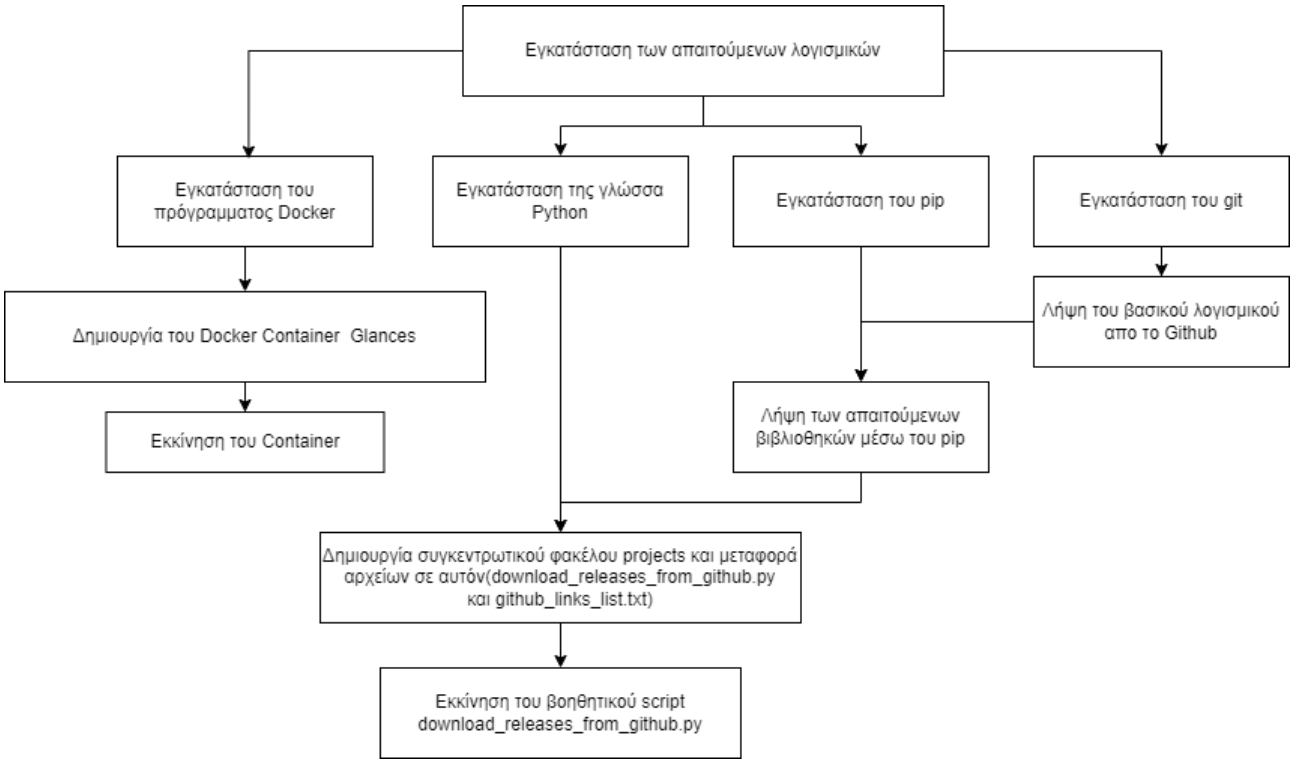
Η λειτουργία του script **initiate_refactors_project.sh** αφορά την εγκατάσταση προ απαιτούμενων στο σύστημα του cloud provider που δημιουργήθηκε. Κατά την εκτέλεση του script , η διαδικασία που ακολουθείται είναι η παρακάτω:

1. Έλεγχος αν το σύστημα έχει εγκατεστημένη κάποια έκδοση Docker, αν όχι, προχωράει σε εγκατάσταση.
2. Εκκίνηση/δημιουργία του Docker container Glances[8].
3. Έλεγχος αν το σύστημα έχει εγκατεστημένη κάποια έκδοση της Python 3, αν όχι, προχωράει σε εγκατάσταση.
4. Έλεγχος αν το σύστημα έχει εγκατεστημένη κάποια έκδοση του pip, αν όχι, προχωράει σε εγκατάσταση. Το pip είναι ο διαχειριστής πακέτων Python. Χρησιμοποιείται για την λήψη βιβλιοθηκών για την γλώσσα Python.
5. Έλεγχος αν το σύστημα έχει εγκατεστημένη κάποια έκδοση του git, αν όχι, προχωράει σε εγκατάσταση.
6. Δημιουργία του αρχείου config.py εντός του src φακέλου.
7. Μέσω του git προχωρά στην λήψη του όλου project από το Github.
8. Εγκατάσταση των προ απαιτούμενων βιβλιοθηκών του project μέσω του pip.

[8] https://github.com/nicolargo/glances

9. Δημιουργία του συγκεντρωτικού φακέλου projects, στον οποίο θα γίνει λήψη όλων των project.
10. Αντιγραφή του βοηθητικού αρχείου **download_releases_from_github.py** και του αρχείου κειμένου **github_links_list.txt.**
11. Δημιουργία του αρχείου config.py με τα αντίστοιχα πεδία εντός του φακέλου projects.
12. Εκκίνηση του script **download_releases_from_github.py** με σκοπό την λήψη των πολλαπλών κυκλοφοριών/εκδόσεων (releases) πολλαπλών έργων (project) από το αποθετήριο του Github.

Τα βήματα 1 και 2 είναι προαιρετικά μιας και βοηθούν στην παρακολούθηση και επιτήρηση του συστήματος. Μόλις ολοκληρωθεί η εκτέλεση του βοηθητικού script **download_releases_from_github.py** το σύστημα είναι έτοιμο για την εκτέλεση του βασικού προγράμματος.



Σχήμα 9 : Διαγραμματική απεικόνιση του script initiate_refactors_project.sh

Βιβλιογραφία

[1] M. Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, November 2018 ,ISBN: 9780134757681

[2] N. Tsantalis, A. Ketkar and D. Dig, "RefactoringMiner 2.0," in IEEE Transactions on Software Engineering, vol. 48, no. 3, pp. 930-950, 1 March 2022
doi: 10.1109/TSE.2020.3007722

[3] JavaFyPy M. Dilhara, A. Ketkar, N. Sannidhi and D. Dig, "Discovering Repetitive Code Changes in Python ML Systems," 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, 2022, doi:10.1145/3510003.3510225.

[2] H. Atwi et al., "PYREF: Refactoring Detection in Python Projects," 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM), Luxembourg, 2021, pp. 136-141 doi: 10.1109/SCAM52516.2021.00025.

[4] Ref-Finder: a refactoring reconstruction tool based on logic query templates. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (FSE '10). Association for Computing Machinery, New York, NY, USA doi: 10.1145/1882291.1882353

[5] LSdiff: a program differencing tool to identify systematic structural differences Alex Loh and Miryung Kim. 2010.. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10). Association for Computing Machinery, New York, NY, USA, 263–266. doi: 10.1145/1810295.1810348

[6] Martin, Robert. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. ISBN:978-0132350884, pp 37

[7] Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan. 2012. A field study of refactoring challenges and benefits. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12). Association for Computing Machinery, New York, NY, USA, Article 50, 1–11. doi:10.1145/2393596.2393655

[8] M. Kim, D. Cai and S. Kim, "An empirical investigation into the role of API-level refactorings during software evolution," *2011 33rd International Conference on Software Engineering (ICSE)*, Honolulu, HI, USA, 2011, doi:10.1145/1985793.1985815.

[9] Kokol, Peter Prof; Kokol, Marko; and Zagoranski, Sašo, "Code smells: A Synthetic Narrative Review" (2020). Library Philosophy and Practice (e-journal). 3622. https://digitalcommons.unl.edu/libphilprac/3622

[10] Alexander Shvets. Dive Into Refactoring Offline Edition (Java) v2019-1.3

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλπ	βλέπε
κ.λπ.	και τα λοιπά
κ.α	και άλλα
π.χ	παραδείγματος χάριν
html	Hypertext Markup Language
pip	pip installs packages
AST	Abstract Syntax Tree
API	Application Programming Interface

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
Πρότυπα ανακατασκευής	Refactoring patterns
Αναγραφή	Refactor
Έκδοση	Release
Εκτέλεση	Execution
Έργο	Project
Νήμα	Thread
Συγκρίσιμα αρχεία	Comparable files
Χαρακτηριστικά	Attributes
Δήλωση μεταβλητής	Variable declaration
Κλάση	Class
Συνάρτηση	Function
Κονστράκτορας	Constructor
Συλλογή	Collection
Πεδίο	Field
Αποθετήριο	Repository
Σενάριο	Script
Πάροχος υπολογιστικού νέφους	Cloud provider
Πυρήνας	Core
Λειτουργικό σύστημα	Operating system
Εκκίνηση	Initiate
Συγχώνευση	Merge

Απόδοση	Ξενόγλωσσος όρος
Αποτέλεσμα	Result
Ψευδό-θετικό	False-positive
Διαμόρφωση	Configuration
Σύμβολο πιστοποίησης ταυτότητας	Authentication token
Δικαιώματα	Permissions
Μητρώο	Registry
Σύνδεσμος	Link
Μονοπάτι	Path
Σελίδα	Sheet