ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Ηλεκτρονικής και Υπολογιστών

Ομάδα Ευφυών Συστημάτων και Τεχνολογίας Λογισμικού

Εξαγωγή Προτύπων Αλλαγών Κώδικα από Αποθετήρια Ανοικτού Λογισμικού

Εκπόνηση:

Οδυσσέας Κυπαρίσσης

Α.Ε.Μ: 8955

Επίβλεψη:

Αν. Καθηγητής, **Ανδρέας Συμεωνίδης**

Υποψήφιος Διδάκτωρ, **Θωμάς Καρανικιώτης**

Θεσσαλονίκη, Σεπτέμβριος 2021

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης καθώς και όλους τους καθηγητές που διδάσκουν στο πρόγραμμα σπουδών, για την προσπάθεια που καταβάλλουν ώστε να μεταδώσουν τις γνώσεις τους και τα κατάλληλα εφόδια που χρειάζονται οι φοιτητές για να μεταβούν στη φάση της επαγγελματικής τους καριέρας.

Στη συνέχεια, οφείλω ένα μεγάλο ευχαριστώ στον αναπληρωτή καθηγητή και επιβλέποντα της διπλωματικής, Ανδρέα Λ. Συμεωνίδη για τις καθοριστικές υποδείξεις του και για την ευκαιρία που μου έδωσε, να ολοκληρώσω τις σπουδές μου υλοποιώντας τη διπλωματική εργασία στην Ομάδα Ευφυών Συστημάτων και Τεχνολογίας Λογισμικού.

Επιπρόσθετα θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτωρ Θωμά Καρανικιώτη, ο οποίος μου πρόσφερε συνεχή υποστήριξη, με βοήθησε επανειλημμένα να αντιμετωπίσω προβλήματα που προέκυψαν κατά τη διάρκεια της υλοποίησης του συγκεκριμένου έργου και μου χάρισε απλόχερα μερικές από τις ατελείωτες γνώσεις που έχει στο συγκεκριμένο αντικείμενο.

Παράλληλα, θα ήθελα να ευχαριστήσω θερμά την οικογένειά μου για την υποστήριξη που μου προσφέρουν τόσα χρόνια, διότι δίχως αυτούς θα ήταν αδύνατη η επίτευξη των στόχων και των ονείρων μου.

Τέλος, δε μπορώ να παραλείψω τους φίλους και τους συμφοιτητές μου, με τους οποίους περάσαμε μαζί τα πιο ξέγνοιαστα χρόνια της ζωής μου που θα θυμάμαι για πάντα.

Στη Β.

Περίληψη

Στη σημερινή εποχή παρατηρείται ραγδαία ανάπτυξη των συστημάτων ελέγχου εκδόσεων και των αποθετηρίων ανοικτού λογισμικού. Τεράστιος όγκος νέων έργων λογισμικού υλοποιείται, αναπτύσσεται και συντηρείται μέσω των συστημάτων αυτών. Κατ' αυτόν τον τρόπο, οι μηχανικοί λογισμικού μπορούν να συνεργάζονται άμεσα μεταξύ τους, να οργανώνουν αποτελεσματικά τα έργα που υλοποιούν και να διατηρούν ένα συγχρονισμένο ιστορικό της εξέλιξής τους. Επομένως, ο όγκος πληροφορίας που αποθηκεύεται είναι τεράστιος και η αξιοποίηση του μπορεί να οδηγήσει στη δημιουργία έξυπνων και αποτελεσματικών συστημάτων.

Τα αποθετήρια αυτά αποτελούν μια θαυμάσια πηγή πληροφοριών. Στη συγκεκριμένη εργασία υλοποιείται ένα σύστημα μηχανικής μάθησης το οποίο αποθηκεύει, επεξεργάζεται και ομαδοποιεί αλλαγές πηγαίου κώδικα που έχουν πραγματοποιηθεί κατά τη διάρκεια εξέλιξης των έργων λογισμικού με τελικό στόχο την εξόρυξη προτύπων αλλαγών πηγαίου κώδικα. Τα πρότυπα αυτά μπορούν να προσφέρουν προτάσεις σε νέα συστήματα που υλοποιούνται ώστε να πραγματοποιηθούν βελτιστοποιήσεις ή διορθώσεις πιθανών σφαλμάτων, οι οποίες έχουν γίνει επανειλημμένα σε μια πληθώρα διαφορετικών έργων λογισμικού στο παρελθόν.

Η μεθοδολογία εφαρμόστηκε στα αποθετήρια του συστήματος GitHub. Μέσω του GitHub, εξορύσσονται οι αλλαγές των αρχείων πηγαίου κώδικα που περιέχονται στο ιστορικό εξέλιξης των αποθετηρίων. Οι αλλαγές αυτές αναπαρίστανται ως Αφηρημένα Συντακτικά Δέντρα (Abstract Syntax Trees) για τον υπολογισμό της ομοιότητας που παρουσιάζει η αλγοριθμική δομή τους. Επίσης, υπολογίζεται η λεξιλογική ομοιότητά τους και, έτσι, είναι εφικτή η τελική τους ομαδοποίηση. Ομάδες οι οποίες πληρούν συγκεκριμένες προϋποθέσεις περιέχουν πρότυπα αλλαγών πηγαίου κώδικα, τα οποία χρησιμοποιούνται για την παροχή συστάσεων σε νέα έργα λογισμικού.

Οδυσσέας Κυπαρίσσης

odykypar@gmail.com

Θεσσαλονίκη, Σεπτέμβριος 2021

Abstract - Mining Source Code Change Patterns from Open-Source Repositories

Nowadays, there is a rapid growth of open-source version control systems and repositories. A large number of new software projects are implemented, developed and maintained through these systems. This way, software engineers can collaborate directly with each other, organize effectively and maintain an up-to-date history of the project's evolution. Therefore, the volume of information stored is significant and its harnessing can lead to the development of smart and efficient systems.

Within the context of this diploma thesis a machine learning system is developed, which stores, processes and groups source code changes that have taken place during the development stage, with the goal of extracting source code changes patterns. These patterns can act as recommendations for new projects, in order to optimize code development and/or fix potential bugs found repeatedly in project repositories.

The proposed methodology was applied on the GitHub code hosting platform. GitHub tracks changes of source code files contained in a repository. These changes are represented as Abstract Syntax Trees (ASTs), so that the calculation of a similarity metric for the algorithmic structure can be achieved. Additionally, their semantic similarity is calculated and thus final clustering of source code changes is possible. Clusters that meet specific criteria, contain patterns of source code changes that can be used to provide recommendations for new software projects.

Odysseas Kyparissis

odykypar@gmail.com

Thessaloniki, September 2021

Περιεχόμενα

Λίστα Διαγραμμάτων

Λίστα Πινάκων

Λίστα Αρτικόλεξων

VCS	Version Control System
CR	Code Refactoring
RSSE	Recommendation Systems in Software Engineering
SA	Statistical Analysis
ML	Machine Learning
API	Application Programming Interface
CVCS	Centralized Version Control Systems
DVCS	Distributed Version Control Systems
OSS	Open-Source Software
IDE	Integrated Development Environment
OOP	Object Oriented Programming
KDD	Knowledge Discovery Databases
AI	Artificial Intelligence
BI	Business Intelligence
SL	Supervised Learning
UL	Unsupervised Learning
RL	Reinforcement Learning
DR	Dimensionality Reduction
PCA	Principal Component Analysis
AST	Abstract Syntactic Tree
MSR	Mining Software Repositories
SCMS	Software Configuration Management System
TED	Tree Edit Distance
AHC	Agglomerative Hierarchical Clustering
SSE	Sum of Squared Errors
PC	Principal Components
CCP	Code Change Patterns
VSM	Vector Space Modelling

Κεφάλαιο 1. Εισαγωγή

1.1. Κίνητρο

Στη σημερινή εποχή, οι σύγχρονες ανάγκες της κοινωνίας αυξάνουν συνεχώς τη ζήτηση για τεχνολογικές καινοτομίες, οι οποίες διευκολύνουν τη λειτουργία διάφορων οργανισμών και επιχειρήσεων. Για τον λόγο αυτό, η δημιουργία νέων έργων λογισμικού, αυξάνεται ολοένα και περισσότερο με την πάροδο του χρόνου.

Καθοριστικό ρόλο για την οργάνωση καθώς και για την ανάπτυξη των έργων λογισμικού, παίζουν τα συστήματα ελέγχου εκδόσεων (VCS - Version Control Systems), όπως το git[1]. Πέρα από την πρόσβαση σε τεράστιο όγκο δεδομένων ανοικτού κώδικα, τα συστήματα αυτά προσφέρουν τη δυνατότητα συνεργασίας μεταξύ των προγραμματιστών μιας ομάδας, για τη συνεχή αποθήκευση και βελτιστοποίηση του πηγαίου κώδικα. Κατά την αποθήκευση των αρχείων πηγαίου κώδικα, διατηρούνται επίσης οι τροποποιήσεις που έχει εφαρμόσει ο κάθε προγραμματιστής, δίνοντας με αυτόν τον τρόπο τη δυνατότητα σε έναν συνεργάτη του να παρακολουθήσει και να ελέγξει την εξέλιξη του έργου. Μεγάλος αριθμός των έργων αυτών, είναι ελεύθερα διαθέσιμος για όλους τους χρήστες του διαδικτύου με σκοπό τη επαναχρησιμοποίηση και την πρόταση βελτιώσεων.

Βέβαια, πολλές φορές, οι ομάδες μηχανικών λογισμικού διαφορετικών ομάδων καλούνται να δώσουν λύση σε προβλήματα ίδιας φύσεως, χρησιμοποιώντας παρόμοιες τεχνολογίες. Έτσι, αποτελεί σύνηθες φαινόμενο η αντιμετώπιση παρόμοιων σφαλμάτων λειτουργίας καθώς και η παρόμοια χρονική εξέλιξη του έργου. Παρ ’όλα αυτά, οι προγραμματιστές εξακολουθούν να αντιμετωπίζουν δυσκολίες όσον αφορά τη διόρθωση σφαλμάτων και την ανάπτυξη λογισμικού κατά τη διάρκεια ενός έργου.

Επομένως, ο τεράστιος όγκος πληροφοριών που σχετίζεται με τη χρονική εξέλιξη των έργων λογισμικού και είναι αποθηκευμένος στα συστήματα ελέγχου εκδόσεων, θα μπορούσε να χρησιμοποιηθεί για την ομαδοποίηση και την εξαγωγή προτύπων αλλαγών πηγαίου κώδικα που λαμβάνουν χώρα σε ένα μεγάλο σύνολο έργων. Μετατρέποντας, στη συνέχεια, τα πρότυπα αυτά σε μία γενικευμένη μορφή, καθίσταται εφικτή η επαναχρησιμοποίησή τους σε νέα ή υπό εξέλιξη έργα λογισμικού. Επιπρόσθετα, θα μπορούσε να επιτευχθεί η εκπαίδευση μοντέλων μηχανικής μάθησης, με σκοπό την αναγνώριση και διόρθωση σφαλμάτων που περιέχονται σε ένα αρχείο κώδικα με βάση τα

[1] https://git-scm.com/

πρότυπα αυτά. Γενικότερα, η αξία όλων αυτών των δεδομένων μπορεί να χρησιμοποιηθεί ώστε να δημιουργηθούν συστήματα τα οποία θα καθιστούν την ανάπτυξη λογισμικού ευκολότερη.

1.2. Περιγραφή Προβλήματος

Κατά τη διάρκεια του κύκλου ζωής του, ένα σύστημα λογισμικού υφίσταται επανειλημμένες τροποποιήσεις. Η εξέλιξη του επηρεάζεται στενά από διάφορες αλλαγές στο περιβάλλον καθώς και στις απαιτήσεις που πρέπει να καλύπτει. Δυστυχώς αρκετές φορές, οι προγραμματιστές δε διαθέτουν τον απαραίτητο χρόνο για να βεβαιωθούν πως οι τροποποιήσεις που πραγματοποιούν στα συστήματα ακολουθούν καλές πρακτικές σχεδίασης[1].Κατά συνέπεια, η ποιότητα λογισμικού συχνά μειώνεται, επιφέροντας κατ' αυτόν τον τρόπο περισσότερες δυσκολίες στη συντήρηση υπαρχόντων λογισμικών[2]. Αρκετές έρευνες έχουν υποδείξει στοιχεία τα οποία συσχετίζουν τη χαμηλή ποιότητα σχεδίασης λογισμικού με χαμηλότερη παραγωγικότητα, περισσότερο χρόνο διόρθωσης σφαλμάτων και γενικότερα περισσότερη προσπάθεια από την πλευρά των προγραμματιστών[3]. Μια λύση που μπορεί να βοηθήσει προς την κατεύθυνση της βελτίωσης της ποιότητας του λογισμικού αποτελεί ο επανασχεδιασμός του αρχικού κώδικα.

Λύση στο παραπάνω πρόβλημα μπορούν να προσφέρουν τα Συστήματα Προτάσεων Τεχνολογίας Λογισμικού (Recommendation Systems in Software Engineering - RSSE). Ένα RSSE είναι ένα λογισμικό που στοχεύει στην υποστήριξη των μηχανικών λογισμικού κατά την διαδικασία λήψης αποφάσεων σχετικά με την ανάπτυξη πηγαίου κώδικα, μελετώντας τις συνήθειες τους[4].Ο στόχος των συστημάτων αυτών είναι η βελτίωση της ποιότητας ενός έργου λογισμικού, κάνοντας τη διαδικασία ανάπτυξης του πιο αποτελεσματική και λιγότερο ακριβή, ελαττώνοντας το γνωστικό φορτίο των προγραμματιστών. Κατά κύριο λόγο υποστηρίζουν την επαναχρησιμοποίηση (reusability) λογισμικού, την αποσφαλμάτωση (debugging), τη σύνταξη καθώς και τη συντήρηση πηγαίου κώδικα.

Η υποστήριξη που προσφέρουν τα συστήματα αυτά, αποτελείται από την πρόταση διορθώσεων ή την προσθήκη νέου πηγαίου κώδικα ιδιωματικού χαρακτήρα και συμβάλει στην επαναχρησιμοποίηση και την αποσφαλμάτωση λογισμικού. Με τον όρο ιδίωμα αναφερόμαστε σε ένα κομμάτι κώδικα το οποίο εμφανίζεται σε μια πληθώρα διαφορετικών έργων λογισμικού και το οποίο έχει μοναδικό σημασιολογικό σκοπό[5]. Βέβαια, για την αποτελεσματική εξαγωγή των ιδιωμάτων αυτών, απαιτείται αρκετά μεγάλος όγκος δεδομένων ο οποίος χρειάζεται να είναι επεξεργασμένος προσεκτικά, ώστε ένα RSSE να προσφέρει προτάσεις και λύσεις ειδικού περιεχομένου είτε σε γενικευμένη, είτε σε ειδική μορφή.

1.3. Στόχοι της Διπλωματικής

Η παρούσα διπλωματική εργασία έχει ως στόχο τη δημιουργία ενός συστήματος, το οποίο αποσκοπεί στην εύρεση και εξαγωγή προτύπων αλλαγών πηγαίου κώδικα από τα δεδομένα που παρέχονται από τα συστήματα VCS και τα αποθετήρια ανοικτού λογισμικού με σκοπό την παροχή προτάσεων τροποποίησης του κώδικα σε νέα έργα. Σκοπό του συστήματος αποτελεί, επίσης, ο εντοπισμός γνωστών σφαλμάτων σε νέα αρχεία πηγαίου κώδικα, με στόχο την παροχή προτάσεων για την άμεση εξάλειψή τους. Τα στάδια λειτουργίας του συστήματος είναι τα εξής:

- Εξόρυξη δεδομένων (data mining) από τα 3.000 δημοφιλέστερα αποθετήρια λογισμικού (repositories) του GitHub.
- Αναγνώριση των μοτίβων που παρουσιάζονται στην αλγοριθμική δομή των τροποποιήσεων κώδικα σε γλώσσα προγραμματισμού Java[2], χρησιμοποιώντας στατιστική ανάλυση (Statistical Analysis - SA) και μηχανική μάθηση (Machine Learning - ML).
- Ομαδοποίηση των τροποποιήσεων με τη χρήση μετρικών ομοιότητας, ως προς την αλγοριθμική δομή και το λεξιλογικό τους περιεχόμενο.
- Εξαγωγή προτύπων αλλαγών κώδικα από τα τελικά αποτελέσματα της ομαδοποίησης.
- Δημιουργία συστήματος προτάσεων τροποποίησης λογισμικού σε γλώσσα προγραμματισμού Java, κάνοντας χρήση των προτύπων αλλαγών που αναφέρθηκαν παραπάνω.

Πιο συγκεκριμένα, υλοποιείται ένα εργαλείο εξαγωγής προτύπων αλλαγών πηγαίου κώδικα που λαμβάνουν χώρα στο εσωτερικό ολοκληρωμένων μεθόδων. Βασική πηγή πληροφορίας αποτελούν οι αλλαγές που πραγματοποιούνται σε διάφορα αρχεία κώδικα και καταγράφονται από τα VCS κατά την εξέλιξη ενός έργου λογισμικού. Ως τελικό αποτέλεσμα της εργασίας, εμφανίζονται προτάσεις αλλαγών στον κώδικα που ακολουθούν την ίδια λογική και έχουν ιδιωματικό χαρακτήρα. Τέλος, επιχειρείται η δημιουργία ενός συστήματος προτάσεων αλλαγών πηγαίου κώδικα για καινούργια αρχεία.

1.4. Διάρθρωση

Η διάρθρωση της διπλωματικής είναι η εξής:

- *Κεφάλαιο 1:* Στο πρώτο κεφάλαιο της παρούσας διπλωματικής εργασίας παρουσιάστηκε μια σύντομη εισαγωγή στο θέμα, αναλύοντας το κίνητρο για την εκπόνηση της εργασίας, την περιγραφή του προβλήματος και τους στόχους της διπλωματικής.

[2] https://www.java.com/

- *Κεφάλαιο 2:* Στο κεφάλαιο 2 περιγράφεται το θεωρητικό υπόβραθρο, το οποίο πρέπει να γνωρίζει ο αναγνώστης ώστε να κατανοήσει πλήρως τη διάρθρωση και τη λειτουργία της διπλωματικής εργασίας
- *Κεφάλαιο 3:* Στο συγκεκριμένο κεφάλαιο, γίνεται μια σύντομη ανάλυση της επιστημονικής έρευνας που βρίσκεται στη βιβλιογραφία και σχετίζεται με την εύρεση προτύπων αλλαγών σε πηγαίο κώδικα.
- *Κεφάλαιο 4:* Σε αυτήν την ενότητα, το πρόβλημα ορίζεται με μεγαλύτερη σαφήνεια και αναλύεται σε βάθος η μεθοδολογία ανάπτυξης του συστήματος.
- *Κεφάλαιο 5:* Στο κεφάλαιο 5 περιγράφονται τα πειράματα που διεξήχθησαν κατά τον έλεγχο λειτουργίας του συστήματος, καθώς και τα αποτελέσματα τους.
- *Κεφάλαιο 6:* Τέλος, περιγράφονται τα τελικά συμπεράσματα που εξάγονται από την παρούσα εργασία και γίνονται ορισμένες προτάσεις για τυχόν βελτιώσεις και επεκτάσεις του συστήματος.

Κεφάλαιο 2. Θεωρητικό Υπόβαθρο

2.1. Γενικά

Στο κεφάλαιο αυτό πραγματοποιείται η περιγραφή του θεωρητικού και γνωστικού υπόβαθρου, το οποίο πρέπει να γνωρίζει ο αναγνώστης ώστε να κατανοήσει πλήρως τη διάρθρωση και τη λειτουργία της διπλωματικής εργασίας. Οι παρακάτω ενότητες περιλαμβάνουν τον ορισμό εννοιών και τεχνολογιών όπως τα αποθετήρια λογισμικού, τα συστήματα ελέγχου εκδόσεων, την επαναχρησιμοποίηση και αποσφαλμάτωση λογισμικού, την εξόρυξη δεδομένων και τη μηχανική μάθηση καθώς και τα αφηρημένα συντακτικά δένδρα (Abstract Syntax Trees – ASTs).

2.2. Αποθετήρια Λογισμικού

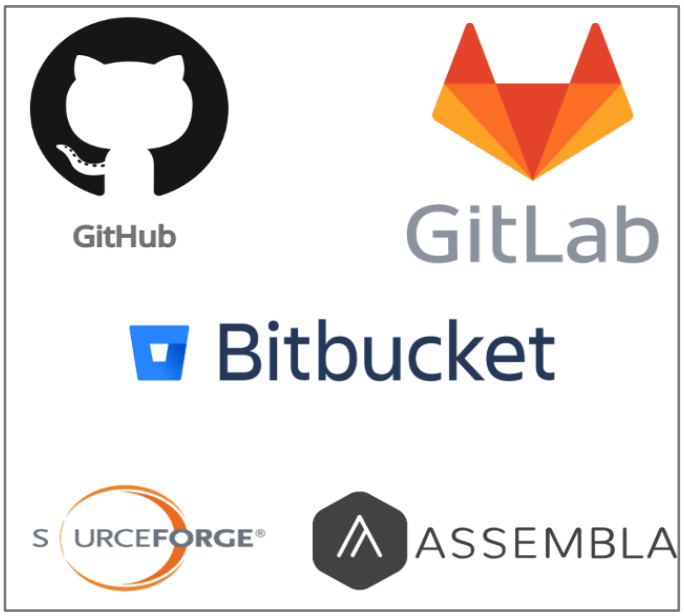
Ένα αποθετήριο λογισμικού (repository), είναι ουσιαστικά μια αποθήκη αρχείων, εγγράφων (documentation), ιστοσελίδων και άλλων έργων, τα οποία είναι προσβάσιμα είτε δημόσια είτε ιδιωτικά. Χρησιμοποιείται από τα συστήματα ελέγχου εκδόσεων – τα οποία περιγράφονται στην ενότητα 2.3 - για την αποθήκευση και επεξεργασία πολλαπλών εκδόσεων αρχείων. Ενώ ένα αποθετήριο μπορεί να ρυθμιστεί σε ένα τοπικό μηχάνημα για έναν μόνο χρήστη, αποθηκεύεται συχνά σε έναν διακομιστή, στον οποίο έχουν τη δυνατότητα παράλληλης πρόσβασης πολλαπλοί χρήστες. Επιπλέον, πολλά αποθετήρια παρέχουν ένα σύστημα εντοπισμού σφαλμάτων, λίστες αλληλογραφίας και έγγραφα τεκμηρίωσης τα οποία βασίζονται στην ανοιχτή εγκυκλοπαίδεια Wikipedia[3].

Ένα repository περιέχει τρία κύρια στοιχεία – έναν κορμό (trunk), κλάδους (branches) και ετικέτες (tags). Ο κορμός περιέχει την τρέχουσα έκδοση ενός έργου λογισμικού και μπορεί να περιλαμβάνει πολλαπλά αρχεία πηγαίου κώδικα, καθώς και άλλους πόρους, οι οποίοι χρησιμοποιούνται από το εκάστοτε λογισμικό. Οι κλάδοι χρησιμοποιούνται για την αποθήκευση νέων εκδόσεων του προγράμματος. Ένας προγραμματιστής έχει τη δυνατότητα δημιουργίας ενός νέου κλάδου όποτε υλοποιεί ουσιώδες αναθεωρήσεις (revisions) στο λογισμικό. Εάν ένας κλάδος περιλαμβάνει ανεπιθύμητες αλλαγές του λογισμικού, μπορεί να διακοπεί. Διαφορετικά, μπορεί να συγχωνευτεί (merge) ξανά στον κορμό ως τελευταία έκδοση. Οι ετικέτες χρησιμοποιούνται για την αποθήκευση εκδόσεων ενός έργου, αλλά δεν προορίζονται για ενεργή ανάπτυξη λογισμικού. Για παράδειγμα, ένας προγραμματιστής μπορεί να δημιουργεί μια "ετικέτα έκδοσης" κάθε φορά που κυκλοφορεί μια νέα έκδοση του λογισμικού.

[3] https://en.wikipedia.org/wiki

Ένα repository παρέχει στους προγραμματιστές έναν δομημένο τρόπο για αποθήκευση αρχείων ανάπτυξης λογισμικού. Αυτό μπορεί να είναι χρήσιμο για κάθε τύπο ανάπτυξης λογισμικού, αλλά είναι ιδιαίτερα σημαντικό για έργα μεγάλου μεγέθους. Με τη δέσμευση (commit) αλλαγών σε ένα αποθετήριο, οι προγραμματιστές μπορούν γρήγορα να επιστρέψουν σε προηγούμενη έκδοση ενός προγράμματος, εάν μια πρόσφατη ενημέρωση προκαλεί σφάλματα ή άλλα προβλήματα. Πολλά συστήματα ελέγχου εκδόσεων υποστηρίζουν ακόμη και παράλληλες συγκρίσεις διαφορετικών εκδόσεων αρχείων που έχουν αποθηκευτεί στο αποθετήριο, οι οποίες μπορούν να είναι χρήσιμες για τον εντοπισμό σφαλμάτων πηγαίου κώδικα.

Τα πιο δημοφιλή αποθετήρια ανοικτού λογισμικού παρουσιάζονται στην Εικόνα 2.1 και μερικές από τις τεχνολογίες που υποστηρίζουν στον Πίνακα 2.1.



Εικόνα 2.1 Τα πιο δημοφιλή Αποθετήρια Ανοικτού Λογισμικού.

Πίνακας 2.1 Χαρακτηριστικά των Δημοφιλέστερων Αποθετηρίων Ανοικτού Λογισμικού.

Repository	GitHub[4]	GitLab[5]	Bitbucket[6]	Source Forge[7]	Assembla[8]
Code Review	✓	✓	✓	✓	✓
Bug Tracking	✓	✓	✓	✓	✓

[4] https://github.com/
[5] https://gitlab.com/
[6] https://bitbucket.org/
[7] https://sourceforge.net/
[8] https://www.assembla.com/

Web Hosting	✓	✓	✓	✓	✓
Wiki	✓	✓	✓	✓	✓
Translation System	✗	✗	✗	✗	✓
Shell Server	✗	✗	✗	✓	✗
Mailing List	✗	✗	✗	✓	✗
Forum	✗	✗	✗	✓	✗
Personal Repository	✓	✓	✓	✓	✓
Private Repository	✓	✓	✓	✓	✓
Announce	✓	✓	✗	✓	✓
Build System	✓	✓	✓	✗	✓
Team	✓	✓	✓	✓	✓
Release Binaries	✓	✓	✗	✓	✓
Self Hosting	Com[9]	✓	Com	✓	✗

2.3. Συστήματα Ελέγχου Εκδόσεων

Στη μηχανική λογισμικού, ο έλεγχος έκδοσης (version control), επίσης γνωστός και ως έλεγχος αναθεώρησης (revision control) ή έλεγχος πηγής (source control) ή διαχείριση πηγαίου κώδικα (source code management), είναι μια κατηγορία συστημάτων, τα οποία είναι υπεύθυνα για τη διαχείριση αλλαγών σε προγράμματα υπολογιστών, έγγραφα, μεγάλους ιστότοπους ή άλλες συλλογές πληροφοριών[6].

Στη διαδικασία ανάπτυξης λογισμικού, είναι φυσιολογικό οι προγραμματιστές να κάνουν συνεχώς αλλαγές σε κομμάτια κώδικα και άλλων αρχείων που περιλαμβάνουν προσθήκη και διαγραφή διάφορων χαρακτηριστικών[7]. Αναμένεται να πραγματοποιηθούν πολλές αναθεωρήσεις (revisions) πριν από την παραγωγή της τελικής έκδοσης. Η διαχείριση και οργάνωση του πηγαίου κώδικα και των αρχείων καθίσταται αρκετά δύσκολη επειδή ο αριθμός των αναθεωρήσεων αυξάνεται λόγω των μεγαλύτερων και περιπλοκότερων συστημάτων[7]. Ως εκ τούτου, η ύπαρξη των συστημάτων ελέγχου εκδόσεων (Version Control Systems – VCSs) βοηθά πραγματικά τους μηχανικούς λογισμικού να επιταχύνουν και να απλοποιήσουν τη διαδικασία ανάπτυξης.

Αναλυτικότερα, υπάρχουν δύο διαφορετικές προσεγγίσεις για τα VCSs, οι οποίες είναι τα Συγκεντρωτικά Σύστημα Ελέγχου Εκδόσεων (Centralized Version Control Systems- CVCSs) και τα Κατανεμημένα Συστήματα Ελέγχου Έκδοσης (Distributed Version Control Systems - DVCSs)[8] . Το CVCS

[9] Commercial

είναι ένα συγκεντρωτικό μοντέλο με ένα κεντρικό αποθετήριο ενώ το DVCS είναι ένα κατανεμημένο μοντέλο χωρίς κεντρικό αποθετήριο, αλλά διαθέτει ένα τοπικό αποθετήριο για κάθε χρήστη. Τα εργαλεία CVCS που χρησιμοποιούνται πιο συχνά είναι τα CVS [10]και Subversion[11][9, 10]. Από την εμφάνιση των εργαλείων DVCS όπως το Git[12], το Mercurial[13], το Bazaar [14]και το BitKeeper[15], πολλά έργα ανοιχτού και κλειστού κώδικα τείνουν να μετακινηθούν ή έχουν ήδη μετακινήσει τα αποθετήρια πηγαίου κώδικα σε ένα DVCS[10]. Περίληψη της σύγκρισης μεταξύ CVCS και DVCS φαίνεται στον Πίνακα 2.2 παρακάτω, καθώς και τα λογότυπα των λογισμικών στην Εικόνα 2.2.

Πίνακας 2.2 Σύγκριση μεταξύ CVCSs - DVCSs.

Version Control System	CVCSs	DVCSs
Αποθετήριο	Υπάρχει μόνο ένα κεντρικό αποθετήριο που είναι ο διακομιστής (server).	Κάθε χρήστης έχει ένα πλήρες αποθετήριο που ονομάζεται τοπικό αποθετήριο στον προσωπικό τους Υπολογιστή.
Πρόσβαση Αποθετηρίου	Κάθε χρήστης που πρέπει να έχει πρόσβαση στο το αποθετήριο πρέπει να συνδεθεί μέσω δικτύου.	Το DVCS επιτρέπει σε κάθε χρήστη να λειτουργεί πλήρως εκτός σύνδεσης. Όμως ο χρήστης χρειάζεται ένα δίκτυο για να το μοιραστεί αποθετήρια με άλλους χρήστες.
Παραδείγματα Εργαλείων VCS	Subversion, Perforce Revision Control System.	Git, Mercurial, Bazaar, BitKeeper
Χαρακτηριστικά Λογισμικού	i. Έργα που επιτρέπουν μόνο μερικούς χρήστες να συνεισφέρουν στην ανάπτυξη λογισμικού. ii. Η ομάδα βρίσκεται σε έναν ιστότοπο.	i. Το DVCS είναι κατάλληλο για έναν ή περισσότερους προγραμματιστές επειδή το αποθετήριο του έργου διανέμεται σε όλους τους προγραμματιστές και αυτή η ικανότητα προσφέρει μεγάλη βελτίωση για τα έργα. ii. Μπορεί επίσης να εφαρμοστεί για μικρά ή μεγάλα έργα λογισμικού επειδή καθιστά λιγότερο δύσκολο για τους κανονικούς χρήστες να συνεισφέρουν στην ανάπτυξη.

[10] https://savannah.nongnu.org/projects/cvs
[11] https://subversion.apache.org/
[12] https://git-scm.com/
[13] https://www.mercurial-scm.org/
[14] https://bazaar.canonical.com/en/
[15] http://www.bitkeeper.org/

		iii. Ομάδα που βρίσκεται σε πολλούς ιστότοπους ή σε διαφορετικές χώρες και διαφορετικές ζώνες ώρας.

(a) *(b)*

Εικόνα 2.2 Τα πιο δημοφιλή VCSs. Στο (a) εμφανίζονται τα λογότυπα των λογισμικών DVCSs, ενώ στο (b) τα λογότυπα των CVCSs.

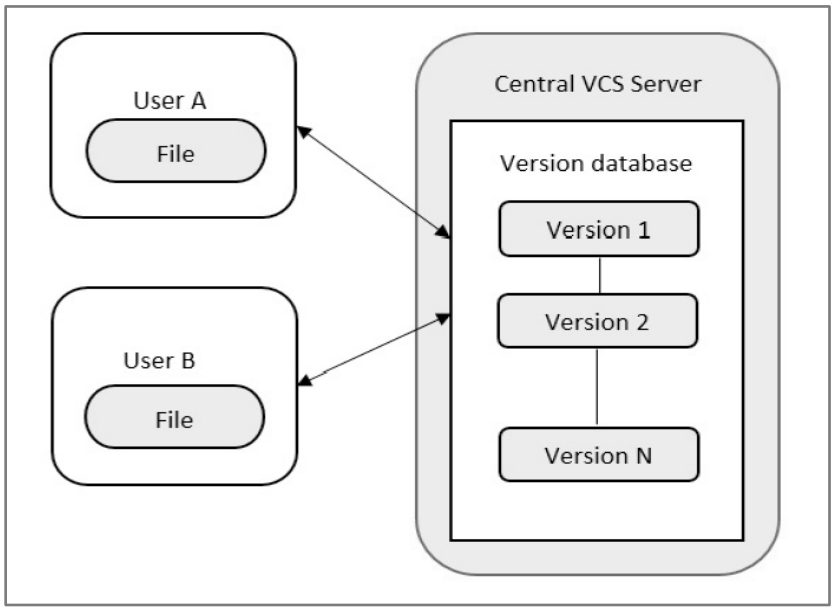
Αρχικά, η ανάπτυξη λογισμικού με DVCS ξεκινά με ένα νέο έργο. Βέβαια, οι μηχανικοί λογισμικού επιτρέπεται επίσης να εισάγουν υπάρχοντα έργα στο VCS. Στη συνέχεια, χρειάζεται να δημιουργήσουν μια έκδοση του έργου στη διεύθυνση εργασίας (working directory) τους, ώστε να έχουν τη δυνατότητα να εργαστούν και να κάνουν αλλαγές στα αρχεία κώδικα τοπικά. Όταν οι αλλαγές που έχουν υλοποιήσει είναι ικανοποιητικές, μπορούν να τις δεσμεύσουν (commit) στο repository με ένα επεξηγηματικό μήνυμα το οποίο τις περιγράφει. Στο επόμενο βήμα, οι μηχανικοί λογισμικού πρέπει να συγχρονίσουν την προσωπική έκδοση του έργου που μόλις επεξεργάστηκαν, με τα commits των άλλων μελών της ομάδας[7]. Τέλος, επισημαίνεται το όνομα της κυκλοφορίας (release's name) για να κυκλοφορήσει μια νέα τελική έκδοση.

2.3.1. Συγκεντρωτικά Σύστημα Ελέγχου Εκδόσεων (CVCSs)

Όπως συμβαίνει και με πολλά άλλα προγραμματιστικά πακέτα ή ιδέες, καθώς οι αναγκαίες προϋποθέσεις συνεχίζουν να αναπτύσσονται, οι πελάτες πιστεύουν ότι τα τοπικά συστήματα ελέγχου εκδόσεων περιορίζουν τις δραστηριότητές τους[11]. Οι άνθρωποι δεν είναι έτοιμοι να εργαστούν συνεργατικά σε ένα παρόμοιο έργο, καθώς το ιστορικό των εκδόσεων του λογισμικού τους αποθηκεύεται σε έναν κοντινό προσωπικό υπολογιστή κάποιου ατόμου και δεν είναι ανοιχτό σε άλλα άτομα που

εργάζονται σε παρόμοια έγγραφα. Τα CVCS αποτελούν μια προσέγγιση που επιτρέπει στους προγραμματιστές να εργαστούν συνεργατικά. Αποθηκεύει ένα κύριο αντίγραφο του ιστορικού των αρχείων και προκειμένου να διαβαστεί, να ανακτηθεί και να κατοχυρωθούν νέες αλλαγές σε ορισμένες εκδόσεις, πρέπει να υπάρχει επικοινωνία με έναν διακομιστή[12].

Όπως φαίνεται στην Εικόνα 2.3, το CVCS έχει ένα μοναδικό αποθετήριο που είναι βασικά ο διακομιστής[9, 11]. Όλα τα αρχεία διατηρούνται στον διακομιστή στον οποίο έχουν πρόσβαση όλοι από τον τοπικό υπολογιστή τους. Όλοι οι προγραμματιστές κάνουν αλλαγές σε αυτό το αποθετήριο ανακτώντας μόνο τη τελευταία έκδοση των αρχείων. Καθώς τα αρχεία αποθηκεύονται στο διακομιστή, αυτό σημαίνει ότι αλλαγές που ενδεχομένως πραγματοποιήθηκαν θα κοινοποιούνται αυτόματα και στους υπόλοιπους προγραμματιστές που έχουν πρόσβαση στο κύριο αποθετήριο.



Εικόνα 2.3 Λειτουργία των Συγκεντρωτικών Συστημάτων Ελέγχου Εκδόσεων.

Οι προγραμματιστές, αλλιώς γνωστοί ως συνεισφέροντες (contributors) ή "*committers*", πρέπει να είναι οι βασικοί προγραμματιστές (core developers) ώστε να έχουν δικαίωμα εκτέλεσης βασικών λειτουργιών, όπως δημιουργία κλάδων (branches), συγχώνευση κλάδων (merging branches) ή επαναφορά αλλαγών σε προηγούμενες καταστάσεις[13]. Παρ 'όλα αυτά, αυτός ο περιορισμός επηρεάζει τη συμμετοχή και το συγγραφικό δικαίωμα νέων committers, επειδή προκειμένου να εκτελέσουν βασικές εργασίες, πρέπει να ακολουθήσουν κάποια βήματα για να θεωρούνται βασικοί προγραμματιστές. Παρακάτω ακολουθούν τα βήματα που πρέπει να ακολουθήσει ο προγραμματιστής για να γίνει βασικός προκειμένου να αποκτήσει άδεια γραφής:

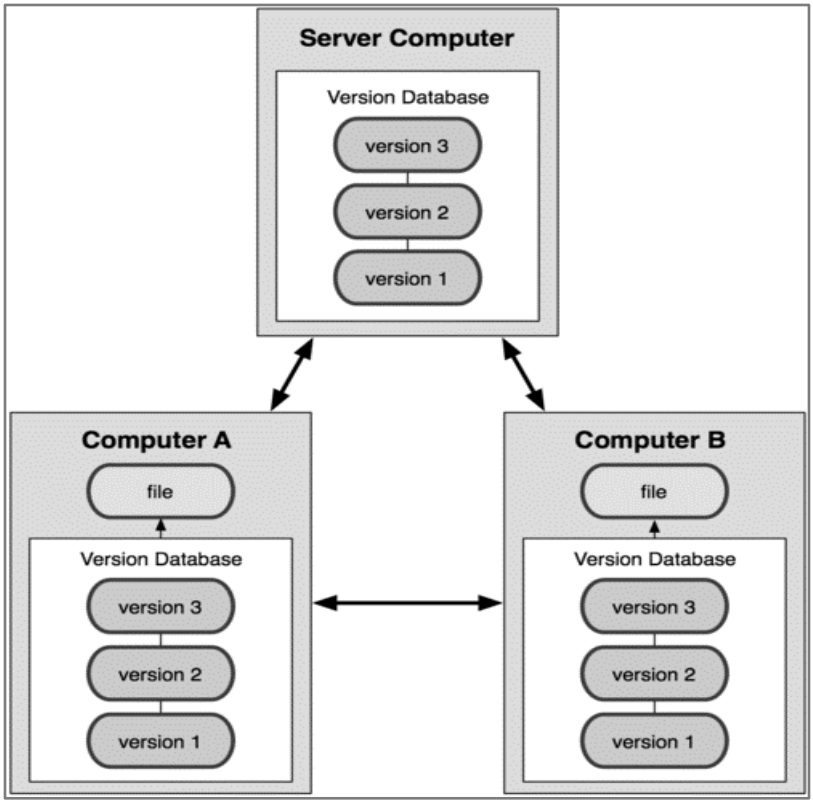
- Ένας προγραμματιστής λογισμικού, αρχικά, εγγράφεται στο αποθετήριο ως "παθητικός χρήστης" (passive user), που σημαίνει ότι δε μπορεί να εντοπιστεί η δραστηριότητά του. Σε αυτό το στάδιο, ο προγραμματιστής δε μπορεί να υποβάλει μηνύματα ηλεκτρονικού ταχυδρομείου ή αναφορές σφαλμάτων καθώς του επιτρέπεται μόνο να επισκέπτεται την ιστοσελίδα του έργου και να διαβάζει τη λίστα αλληλογραφίας.
- Από "παθητικοί χρήστες", οι προγραμματιστές εξελίσσονται σε "προχωρημένοι χρήστες" (advanced users) εάν εξοικειωθούν με το έργο λογισμικού. Σε αυτό το στάδιο, επιτρέπεται στους προγραμματιστές να στέλνουν ορισμένα μηνύματα στη λίστα αλληλογραφίας. Αυτή η δραστηριότητα μπορεί να παρακολουθείται.
- Οι προγραμματιστές που έχουν μερική γνώση σχετικά με το έργο ενδέχεται να αναφέρουν σφάλματα, να υποβάλουν σχόλια και να υλοποιήσουν μερικές τροποποιήσεις στο έργο.
- Σε αυτό το στάδιο, οποιοσδήποτε committer δίνει σημαντικές και ουσιαστικές συνεισφορές σχετικά με το έργο, λαμβάνει έναν λογαριασμό με άδεια συγγραφής στο αποθετήριο. Προγραμματιστές που έχουν άδεια συγγραφής έχουν το δικαίωμα καταχώρησης οποιασδήποτε αλλαγής έχει πραγματοποιηθεί στο έργο λογισμικού.
- Η τελευταία διαδικασία για να αποτελέσει ένας προγραμματιστής μέλος της ομάδας των βασικών προγραμματιστών, είναι η συνεισφορά στο έργο με υψηλού επιπέδου δραστηριότητα πάνω από ένα ορισμένο χρονικό διάστημα.

2.3.2. Κατανεμημένα Συστήματα Ελέγχου Εκδόσεων (DVCSs)

Σήμερα, τα έργα λογισμικού ανοικτού κώδικα (Open Source Software – OSS) υιοθετούν σε μεγάλο βαθμό τα DVCSs[14]. Αυτό οφείλεται στον κίνδυνο χρήσης Τοπικού 'Ελέγχου Έκδοσης (Local Version Control) και των CVCSs. Όταν ένα έργο χρησιμοποιεί CVCS, αυτό σημαίνει ότι το ιστορικό που σχετίζεται με το έργο θα διατηρηθεί σε ένα μόνο μέρος, ενώ οι προγραμματιστές δε θα έχουν τη δυνατότητα να εργαστούν ταυτόχρονα, λόγω του τοπικού ελέγχου έκδοσης. Έναν ακόμα υποκείμενο κίνδυνο αποτελεί η δημιουργία προβλημάτων κατά τη διαδικασία επικοινωνίας με τον διακομιστή[12]. Οι κίνδυνοι που αναφέρθηκαν είναι οι λόγοι για τους οποίους τα περισσότερα OSS υιοθετούν το νέο σύστημα ελέγχου εκδόσεων για καταγραφή, διαχείριση και εξέλιξη των αλλαγών του πηγαίου κώδικα.

Όπως φαίνεται στην Εικόνα 2.4, κατά βάση, το DVCS έχει σχεδιαστεί για να ενεργεί και με τους δύο τρόπους, καθώς αποθηκεύει ολόκληρη την ιστορία των αρχείων σε κάθε υπολογιστή τοπικά[15] και μπορεί επίσης να συγχρονίσει τις τοπικές αλλαγές που γίνονται από τον χρήστη πίσω στον διακομιστή οποτεδήποτε απαιτείται, έτσι ώστε οι αλλαγές να μπορούν να κοινοποιηθούν σε ολόκληρη την

ομάδα[11]. Οι *clients* - ηλεκτρονικοί υπολογιστές ή σταθμοί εργασίας που λαμβάνουν πληροφορίες και εφαρμογές από έναν διακομιστή - μπορούν να επικοινωνούν μεταξύ τους και να διατηρήσουν τους δικούς τους τοπικούς κλάδους, χωρίς να χρειάζεται να περάσουν από κάποιον κεντρικό διακομιστή ή αποθετήριο. Στη συνέχεια, ο συγχρονισμός λαμβάνει χώρα μεταξύ των χρηστών που αποφασίζουν ποια αλλαγή μπορεί να αντιγραφεί πλήρως στο αποθετήριο. Επομένως, εάν κάποιος διακομιστής καταρρεύσει, και αυτοί οι χρήστες συνεργάζονταν χρησιμοποιώντας τον, οποιοδήποτε από τα αποθετήρια των clients μπορεί να αντικατοπτριστεί στον διακομιστή για να το ανακτήσει.



Εικόνα 2.4 Λειτουργία των Διανεμημένων Συστημάτων Ελέγχου Εκδόσεων.

Κάθε κλώνος είναι ένα πλήρες αντίγραφο ασφαλείας όλων των δεδομένων. Παραδείγματα εργαλείων DVCS είναι τα Git, Bazaar, Mercurial, Bitkeeper, και Darcs[14, 16]. Το Bitkeeper και το Bazaar είναι δύο από τα πρώτα εργαλεία DVCS[17]. Το Git και το Mercurial είναι τα νεότερα εργαλεία DVCS που κατάφεραν να βελτιώσουν τις δυνατότητες συγχώνευσης και διακλάδωσης.

2.4. Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface – API)

Μία *Διεπαφή Προγραμματισμού Εφαρμογών* (Application Programming Interface - API) είναι μια σύνδεση μεταξύ υπολογιστών ή μεταξύ υπολογιστικών προγραμμάτων. Ουσιαστικά, είναι ένας τύπος

διεπαφής λογισμικού, που προσφέρει μια υπηρεσία σε άλλα κομμάτια λογισμικού[18]. Ένα έγγραφο ή ένα πρότυπο που περιγράφει τον τρόπο δημιουργίας μιας τέτοιας σύνδεσης ονομάζεται “API specification”. Ο όρος API μπορεί να αναφέρεται είτε στο παραπάνω έγγραφο είτε στην υλοποίηση της σύνδεσης. Σε αντίθεση με μια διεπαφή χρήστη (user interface), η οποία συνδέει έναν υπολογιστή με ένα άτομο, ένα API συνδέει υπολογιστές ή κομμάτια λογισμικού μεταξύ τους. Ένα API αποτελείται από διαφορετικά μέρη που λειτουργούν ως εργαλεία ή υπηρεσίες και είναι διαθέσιμα στον προγραμματιστή. Τα μέρη αυτά είναι συνήθως υπορουτίνες, μέθοδοι, αιτήματα (requests) ή τελικά σημεία (endpoints). Η λειτουργία όλων των παραπάνω περιγράφεται λεπτομερώς στο API specification.

Ένας σκοπός των APIs είναι να αποκρύψουν τις εσωτερικές λεπτομέρειες για το πώς λειτουργεί ένα σύστημα, εκθέτοντας μόνο εκείνα τα μέρη που είναι χρήσιμα για έναν προγραμματιστή. Επιπλέον, καλή πρακτική σχεδίασης ενός API αποτελεί η διατήρηση της συνεπούς λειτουργίας του, ακόμη και αν οι εσωτερικές λεπτομέρειες υφίστανται αλλαγές. Ένα API μπορεί να είναι προσαρμοσμένο για ένα συγκεκριμένο ζεύγος συστημάτων ή μπορεί να είναι ένα κοινό πρότυπο που επιτρέπει τη διαλειτουργικότητα μεταξύ πολλών συστημάτων. Τέλος, Ο όρος API χρησιμοποιείται συχνά ως αναφορά στα web APIs, τα οποία επιτρέπουν την επικοινωνία μεταξύ υπολογιστών που είναι συνδεδεμένοι στο διαδίκτυο. Υπάρχουν επίσης APIs για γλώσσες προγραμματισμού, βιβλιοθήκες λογισμικού, λειτουργικά συστήματα υπολογιστών και υλικό υπολογιστή.

2.5. Επαναχρησιμοποίηση Λογισμικού

Στην επιστήμη των υπολογιστών (computer science) και στη μηχανική λογισμικού (software engineering) επαναχρησιμοποίηση λογισμικού (software reuse) ή κώδικα (code reuse) θεωρείται η χρήση ήδη υπάρχοντος λογισμικού ή γνώσης λογισμικού, με σκοπό τη δημιουργία ενός νέου συστήματος[19], ακολουθώντας κάποιες αρχές επαναχρησιμοποίησης (reusability principles).

Η επαναχρησιμοποίηση κώδικα στοχεύει στην εξοικονόμηση χρόνου και πόρων, με την αξιοποίηση στοιχείων λογισμικού τα οποία έχουν ήδη δημιουργηθεί σε κάποια μορφή κατά τη διαδικασία ανάπτυξης ενός προϊόντος λογισμικού[20]. Η βασική ιδέα επαναχρησιμοποίησης είναι πως κομμάτια ενός προγράμματος τα οποία έχουν γραφεί μια φορά, μπορούν ή θα έπρεπε να χρησιμοποιηθούν στην κατασκευή άλλων προγραμμάτων αργότερα.

Καθώς ο κώδικας αποτελεί τον πιο βασικό πόρο που επιλέγεται για επαναχρησιμοποίηση, άλλα στοιχεία τα οποία παράγονται κατά τον κύκλο ανάπτυξης ενός έργου λογισμικού ίσως μπορούν να προσφέρουν ευκαιρίες επαναχρησιμοποίησης. Για παράδειγμα: τμήματα λογισμικού (software

components), test suites (συλλογή περιπτώσεων ελέγχου για την εύρυθμη λειτουργία και συμπεριφορά ενός προγράμματος λογισμικού σε συγκεκριμένες καταστάσεις), σχεδιασμοί λογισμικού (software designs), εγχειρίδια και ούτω καθεξής[21].

2.5.1. Πεδία Εφαρμογής

Παρακάτω αναφέρονται τα πιο σημαντικά πεδία στα οποία εφαρμόζεται η επαναχρησιμοποίηση κώδικα:

2.5.1.1. Βιβλιοθήκες Λογισμικού (Software Libraries)

Ένα πολύ βασικό παράδειγμα επαναχρησιμοποίησης κώδικα αποτελεί η τεχνική χρήσης βιβλιοθηκών λογισμικού. Πολύ συχνές λειτουργίες, όπως η μετατροπή πληροφορίας μεταξύ διαφορετικών μορφών, η πρόσβαση εξωτερικού αποθηκευτικού χώρου (external storage), η διασύνδεση με εξωτερικά προγράμματα ή ο κατάλληλος χειρισμός πληροφορίας (αριθμών, λέξεων, ονομάτων, τοποθεσιών, ημερομηνιών, κτλ.) χρησιμοποιούνται από πολλά διαφορετικά προγράμματα. Οι συγγραφείς νέων προγραμμάτων μπορούν να κάνουν χρήση του κώδικα μιας βιβλιοθήκης λογισμικού με σκοπό την υλοποίηση των παραπάνω λειτουργιών, αντί να δημιουργήσουν εκ νέου τον κώδικα που υλοποιεί την ίδια διαδικασία. Η υλοποίηση βιβλιοθηκών συχνά έχει το εξής πλεονέκτημα: ο κώδικας είναι ελεγμένος όσον αφορά τη λειτουργία του και την κάλυψη ασυνήθιστων περιπτώσεων. Μερικά μειονεκτήματα αποτελούν η ανικανότητα εφαρμογής μικροδιορθώσεων, οι οποίες μπορούν να επηρεάζουν την επίδοση του αλγορίθμου ή το επιθυμητό αποτέλεσμα, καθώς και ο χρόνος και το κόστος που απαιτούνται για την απόκτηση, εκμάθηση και διαμόρφωση της βιβλιοθήκης[21].

2.5.1.2. Πρότυπα Σχεδίασης (Design Patterns)

Στη μηχανική λογισμικού, ένα πρότυπο σχεδίασης λογισμικού είναι μια γενική επαναχρησιμοποιήσιμη λύση σε ένα γνωστό πρόβλημα στο πλαίσιο της σχεδίασης λογισμικού. Δεν είναι μια ολοκληρωμένη σχεδίαση η οποία μπορεί να μετατραπεί απευθείας σε πηγαίο κώδικα ή γλώσσα μηχανής, αντιθέτως είναι μια περιγραφή ή ένα πρότυπο, το οποίο περιγράφει το τρόπο που μπορεί να λυθεί ένα πρόβλημα και να χρησιμοποιηθεί σε πολλές διαφορετικές περιπτώσεις. Τα πρότυπα σχεδίασης είναι γνωστές πρακτικές που μπορούν να χρησιμοποιηθούν από έναν προγραμματιστή, με στόχο την επίλυση συνηθισμένων προβλημάτων κατά τη σχεδίαση μιας εφαρμογής ή ενός συστήματος.

2.5.1.3. Πλαίσια Λογισμικού - Software Frameworks

Στον προγραμματισμό υπολογιστών (computer programming), ένα πλαίσιο λογισμικού είναι ένα αφηρημένο σύστημα που: προσφέρει λογισμικό γενικευμένης λειτουργικότητας, αλλά μπορεί να

τροποποιηθεί επιλεκτικά προσθέτοντας κώδικα γραμμένο από κάποιον χρήστη ώστε να χρησιμοποιηθεί σε πιο εξειδικευμένες περιπτώσεις. Επιπρόσθετα προσφέρει έναν συμβατικό τρόπο κατασκευής και ανάπτυξης εφαρμογών και αποτελεί ένα καθολικό, επαναχρησιμοποιήσιμο περιβάλλον λογισμικού, το οποίο παρέχει συγκεκριμένη λειτουργικότητα ως κομμάτι μιας εκτενέστερης πλατφόρμας λογισμικού, με κύριο στόχο τη διευκόλυνση της ανάπτυξης εφαρμογών, προϊόντων και λύσεων. Ένα software framework μπορεί να περιλαμβάνει προγράμματα υποστήριξης (support programs), μεταγλωττιστές (compilers), βιβλιοθήκες κώδικα, εργαλειοθήκες (toolsets), και APIs που μπορούν να ενώσουν όλα τα διαφορετικά δομικά στοιχεία τα οποία είναι απαραίτητα για την ανάπτυξη ενός συστήματος ή ενός έργου.

Τα βασικά χαρακτηριστικά των frameworks που τα διαφοροποιούν από τις απλές βιβλιοθήκες παρουσιάζονται παρακάτω:

- Αντιστροφή Ελέγχου (Inversion of control): Στο εσωτερικό ενός framework, σε αντίθεση με τις βιβλιοθήκες ή τις τυπικές εφαρμογές, η συνολική ροή ελέγχου δεν υπαγορεύεται από τον καλούντα αλλά από το ίδιο το framework[22].
- Επεκτασιμότητα (Extensibility): Ένας χρήστης μπορεί να επεκτείνει το framework, συνήθως με επιλεκτική παράκαμψη, ή να προσθέσει εξειδικευμένο κώδικα για να παρέχει συγκεκριμένη λειτουργικότητα.
- Μη Τροποποιήσιμο Λογισμικό (Non-modifiable framework code): Το λογισμικό που προσφέρει ένα framework, γενικά, δεν πρέπει να τροποποιηθεί, αλλά δέχεται επεκτάσεις που υλοποιούνται από τον χρήση. Με άλλα λόγια, οι χρήστες μπορούν να επεκτείνουν το framework, αλλά δε μπορούν να τροποποιήσουν τον κώδικα του.

2.5.1.4. Αντικειμενοστραφής Προγραμματισμός (Object – Oriented Programming - OOP)

Πολλοί ερευνητές έχουν εργαστεί για να καταστήσουν την επαναχρησιμοποίηση κώδικα γρηγορότερη, ευκολότερη, πιο συστηματική και αναπόσπαστο μέρος της κανονικής διαδικασίας προγραμματισμού. Αυτοί είναι μερικοί από τους κύριους στόχους πίσω από την εφεύρεση του αντικειμενοστραφούς προγραμματισμού, ο οποίος έχει εξελιχθεί σε μια από τις πιο κοινές μορφές τυποποιημένης επαναχρησιμοποίησης. Ουσιαστικά, είναι ένα πρότυπο προγραμματισμού που βασίζεται στην έννοια των “αντικειμένων” το οποίο μπορεί να περιέχει δεδομένα και κώδικα:

- Δεδομένα με τη μορφή πεδίων (συχνά γνωστά ως χαρακτηριστικά ή ιδιότητες).
- Κώδικα με τη μορφή διαδικασιών (συχνά γνωστοί ως μέθοδοι).

Ένα χαρακτηριστικό στοιχείο των αντικειμένων είναι ότι οι διαδικασίες ενός αντικειμένου μπορούν να έχουν πρόσβαση και να τροποποιούν συχνά τα πεδία δεδομένων του (περιέχουν την έννοια του “this” ή του “self”). Στον OOP, τα προγράμματα υπολογιστών, σχεδιάζονται αποτελούμενα από αντικείμενα τα οποία αλληλοεπιδρούν το ένα με το άλλο[23, 24]. Οι γλώσσες OOP ποικίλουν, αλλά οι πιο δημοφιλείς είναι βασισμένες στη χρήση “κλάσεων”, πράγμα που σημαίνει ότι τα αντικείμενα αποτελούν παράγωγα αυτών και επίσης καθορίζουν τον τύπο τους.

Πολλές από τις πιο ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού (όπως C ++, Java, Python κ.λπ.), οι οποίες περιέχουν πολλαπλά πρότυπα προγραμματισμού, υποστηρίζουν αντικειμενοστραφή προγραμματισμό, συνήθως σε συνδυασμό με επιτακτικό (imperative), διαδικαστικό προγραμματισμό (procedural programming).

2.5.2. Πλεονεκτήματα & Μειονεκτήματα

Παρακάτω αναγράφονται τα πλεονεκτήματα και τα μειονεκτήματα της επαναχρησιμοποίησης λογισμικού. Αρχικά, όσον αφορά τα πλεονεκτήματα, πολύ σημαντικό ρόλο παίζει η μείωση του συνολικού κόστους και χρόνου παραγωγής ενός έργου λογισμικού ή ενός συστήματος[25]. Πέραν αυτού, σημειώνεται αύξηση της αξιοπιστίας του συστήματος, καθώς τα επαναχρησιμοποιημένα τμήματα λογισμικού είναι κατά βάση δοκιμασμένα και ελεγμένα. Επιπρόσθετα, ο χρόνος αποσφαλμάτωσης είναι σημαντικά μικρότερος, διότι η πιθανότητα εμφάνισης σφαλμάτων είναι πιο μικρή συγκριτικά με την εκ νέου δημιουργία λογισμικού.

Βέβαια, υπάρχουν και μερικά μειονεκτήματα τα οποία χαρακτηρίζουν την εν λόγω ιδιότητα. Ένα σημαντικό πρόβλημα αποτελεί η ύπαρξη ασυμφωνίας μεταξύ των διαφορετικών συστημάτων. Πιο συγκεκριμένα, ενδέχεται διαφορετικά έργα λογισμικού να πραγματεύονται παρόμοια θέματα, αλλά η προσέγγιση που ακολουθείται να είναι διαφορετική. Αυτό έχει ως αποτέλεσμα τμήματα λογισμικού να μην ταιριάζουν απόλυτα και στις δύο περιπτώσεις. Ακόμα, η συντήρηση έτοιμων κομματιών πηγαίου κώδικα είναι αρκετά δύσκολη και χρονοβόρα, λόγω έλλειψης ολοκληρωμένης τεκμηρίωσης (documentation). Τέλος, συχνό φαινόμενο αποτελεί το γεγονός πως οι προγραμματιστές για τη σωστή χρήση έτοιμου λογισμικού, χρειάζεται να δαπανήσουν αρκετό χρόνο για να κατανοήσουν σε βάθος την υλοποίηση της εκάστοτε τεχνολογίας που χρησιμοποιείται.

2.6. Αποσφαλμάτωση (Debugging)

Στην επιστήμη των υπολογιστών και την ανάπτυξη λογισμικού, η αποσφαλμάτωση αποτελεί τη διαδικασία εύρεσης και επίλυσης σφαλμάτων (ελαττώματα ή προβλήματα που εμποδίζουν τη σωστή

λειτουργία) έργων λογισμικού ή συστημάτων. Η τακτική εντοπισμού σφαλμάτων μπορεί να περιλαμβάνει διαδραστικό εντοπισμό σφαλμάτων (interactive debugging), ανάλυση ροής ελέγχου (control flow analysis), δοκιμή μονάδας (unit testing), δοκιμή ενσωμάτωσης (integration testing), ανάλυση αρχείων καταγραφής (log file analysis), παρακολούθηση σε επίπεδο εφαρμογής ή συστήματος (application or system monitoring), απορρίψεις μνήμης (memory dumps) και δημιουργία προφίλ (profiling). Πολλές γλώσσες προγραμματισμού και εργαλεία ανάπτυξης λογισμικού προσφέρουν επίσης προγράμματα που βοηθούν στην αποσφαλμάτωση λογισμικού, γνωστά ως "*debuggers*".

Ουσιαστικά η αποσφαλμάτωση κώδικα περιλαμβάνει την ανάλυση και ενδεχομένως την επέκταση ενός δεδομένου προγράμματος, το οποίο δεν πληροί τις κατάλληλες προδιαγραφές, με σκοπό την επίτευξη των προδιαγραφών αυτών. Συνεπώς, η αποσφαλμάτωση είναι η "διαδικασία που περιλαμβάνει τη διάγνωση της ακριβούς φύσης ενός γνωστού σφάλματος και τη διόρθωση του"[26].

Όπως είναι γνωστό μεταξύ των μηχανικών λογισμικού, το μεγαλύτερο μέρος της προσπάθειας για την αποσφαλμάτωση λογισμικού περιλαμβάνει τον εντοπισμό ελαττωμάτων. Ο εντοπισμός σφαλμάτων γίνεται σε πολύ μικρό βαθμό κατά τη διαδικασία συγγραφής των αρχείων πηγαίου κώδικα. Στα πρώτα χρόνια της ανάπτυξης λογισμικού, ελαττώματα τα οποία δεν εντοπίζονταν κατά τον έλεγχο των αρχείων λογισμικού (code reviewing), εμφανίζονταν κατά τη διαδικασία της μεταγλώττισης και της εκτέλεσης του κώδικα[27]. Μέσω μιας χρονοβόρας και δύσκολης διαδικασίας (όπως εισαγωγή εντολών εκτύπωσης στις κατάλληλες θέσεις του προγράμματος), ένας προγραμματιστής έπρεπε να εντοπίσει την ακριβή τοποθεσία του σφάλματος και να βρει μια κατάλληλη επιδιόρθωση.

Ακόμα και σήμερα, ο εντοπισμός σφαλμάτων παραμένει ένα είδος "τέχνης". Μεγάλο μέρος της κοινότητας της επιστήμης των υπολογιστών έχει αγνοήσει σε μεγάλο βαθμό το πρόβλημα εντοπισμού σφαλμάτων. Σε παλαιότερες έρευνες[28, 29] μελετήθηκαν 59 ατεκμηρίωτες εμπειρίες εντοπισμού σφαλμάτων και τα συμπεράσματά είχαν ως εξής:

- Ακριβώς πάνω από το 50% των προβλημάτων προέκυψαν λόγω του μεγάλου χρονικού διαστήματος που παρατηρείται μεταξύ της διαδικασίας ανάπτυξης του λογισμικού και της εμφάνισης των σφαλμάτων, ή λόγω ανεπάρκειας εργαλείων εντοπισμού σφαλμάτων.

Οι κυρίαρχες τεχνικές εύρεσης σφαλμάτων ήταν η συλλογή δεδομένων (π.χ. εκτυπώσεις δηλώσεων) και η χειροκίνητη προσομοίωση. Οι δύο μεγαλύτερες αιτίες σφαλμάτων ήταν η αντικατάσταση πληροφορίας στη μνήμη (memory overwrites) και τα ελαττώματα στο λογισμικό ή το υλικό που προέρχονταν από τους προμηθευτές[27].

Βέβαια πολλές νέες προσεγγίσεις έχουν προταθεί και πολλά περιβάλλοντα αποσφαλμάτωσης είναι πλέον διαθέσιμα στην αγορά για τον εντοπισμό των σφαλμάτων στο πρόγραμμα κατά τη διάρκεια συγγραφής. Τα ολοκληρωμένα περιβάλλοντα ανάπτυξης (Integrated Development Environments – IDEs) παρέχουν ένα σύστημα εντοπισμού μερικών από τα προκαθορισμένα σφάλματα που δημιουργούνται συνήθως σε συγκεκριμένες γλώσσες προγραμματισμού (π.χ., έλλειψη του χαρακτήρα τέλους μιας δήλωσης, μη καθορισμένες μεταβλητές και ούτω καθεξής), χωρίς την ανάγκη για μεταγλώττιση του κώδικα.

2.7. Εξόρυξη Δεδομένων (Data Mining)

Η εξόρυξη δεδομένων είναι μια διαδικασία ανακάλυψης και εξαγωγής πολύτιμων προτύπων σε μεγάλα σύνολα δεδομένων η οποία περιλαμβάνει συνδυαστικές μεθόδους μηχανικής μάθησης, στατιστικής και συστημάτων βάσεων δεδομένων[30]. Η εξόρυξη δεδομένων είναι ένα διεπιστημονικό πεδίο της επιστήμης των υπολογιστών και της στατιστικής, με συνολικό στόχο την εξαγωγή πληροφοριών (με έξυπνες μεθόδους) από ένα σύνολο δεδομένων και τη μετατροπή των πληροφοριών αυτών σε κατανοητή δομή για περαιτέρω χρήση[31]. Επιπρόσθετα, αποτελεί ένα βήμα της διαδικασίας "ανακάλυψης γνώσεων σε βάσεις δεδομένων" (KDD - Knowledge Discovery in Databases)[32]. Εκτός από το στάδιο της ανάλυσης πρωτότυπων δεδομένων, περιλαμβάνει γενικότερα μεθόδους διαχείρισης δεδομένων και βάσεων δεδομένων, προεπεξεργασίας, υπολογισμού μετρήσεων - σχετικών με τα δεδομένα - και θεμάτων πολυπλοκότητας. Οδηγεί στον σχεδιασμό μοντέλων και την εξαγωγή συμπερασμάτων, καθώς και στη μετα-επεξεργασία ανακαλυφθέντων δομών και στην οπτικοποίηση τους[30].

Ο όρος “εξόρυξη δεδομένων” δεν είναι πολύ ακριβής, επειδή ο στόχος είναι η εξαγωγή προτύπων και γνώσεων από μεγάλο όγκο δεδομένων και όχι η εξαγωγή (εξόρυξη) των ίδιων των δεδομένων[33]. Είναι επίσης λέξη-κλειδί και εφαρμόζεται συχνά σε οποιαδήποτε μορφή επεξεργασίας μεγάλης κλίμακας (large-scale) δεδομένων ή πληροφοριών (συλλογή, εξαγωγή, αποθήκευση, ανάλυση), καθώς και στην ανάπτυξη διαφόρων συστημάτων υποστήριξης αποφάσεων, συμπεριλαμβανομένης της τεχνητής νοημοσύνης (AI – Artificial Intelligence) και της επιχειρηματικής ευφυΐας (BI – Business Intelligence). Συχνά οι πιο γενικοί όροι, όπως ανάλυση δεδομένων μεγάλης κλίμακας ή, όταν αναφέρονται σε πραγματικές μεθόδους, τεχνητή νοημοσύνη και μηχανική μάθηση, είναι πιο κατάλληλοι.

Η πραγματική διεργασία εξόρυξης δεδομένων είναι είτε ημι-αυτόματη, είτε αυτόματη και αποτελείται από την ανάλυση μεγάλων ποσοτήτων δεδομένων, για εξαγωγή προηγουμένως άγνωστων προτύπων, όπως ανάλυση ομάδων δεδομένων (cluster analysis), ασυνήθιστες εγγραφές (ανίχνευση

ανωμαλιών) και εξαρτήσεις (εξόρυξη κανόνων συσχέτισης, διαδοχική εξόρυξη προτύπων). Αυτό συνήθως περιλαμβάνει τη χρήση διαφόρων τεχνικών βάσεων δεδομένων. Αυτά τα μοτίβα μπορούν στη συνέχεια να θεωρηθούν ως ένα είδος σύνοψης των αρχικών δεδομένων και μπορούν να χρησιμοποιηθούν σε περαιτέρω ανάλυση ή, για παράδειγμα, σε μοντέλα μηχανικής μάθησης και προγνώσεων. Για παράδειγμα, η διαδικασία εξόρυξης δεδομένων μπορεί να προσδιορίσει διαφορετικές συσχετιζόμενες ομάδες στα δεδομένα, οι οποίες μπορούν στη συνέχεια να χρησιμοποιηθούν για τη λήψη ακριβέστερων αποτελεσμάτων πρόβλεψης από ένα σύστημα υποστήριξης αποφάσεων. Ούτε η συλλογή και η προετοιμασία δεδομένων, αλλά ούτε η ερμηνεία και η αναφορά αποτελεσμάτων αποτελούν μέρος της εξόρυξης δεδομένων, αλλά ανήκουν στη συνολική διαδικασία KDD ως πρόσθετα βήματα.

Η διαφορά μεταξύ ανάλυσης δεδομένων και εξόρυξης δεδομένων είναι ότι η ανάλυση δεδομένων χρησιμοποιείται για τη δοκιμή μοντέλων και υποθέσεων στο σύνολο δεδομένων ανεξάρτητα από τον αριθμό τους π.χ. ανάλυση της αποτελεσματικότητας μιας καμπάνιας μάρκετινγκ. Αντίθετα, η εξόρυξη δεδομένων χρησιμοποιεί μηχανική μάθηση και στατιστικά μοντέλα για να αποκαλύψει κρυφά ή κρυμμένα μοτίβα σε μεγάλο όγκο δεδομένων[34].

Οι σχετικοί όροι data dredging, data fishing και data snooping αναφέρονται στη χρήση μεθόδων εξόρυξης δεδομένων, για τη δειγματοληψία τμημάτων ενός μεγαλύτερου συνόλου δεδομένων, τα οποία είναι πολύ μικρά για την έγκυρη ανακάλυψη μοτίβων και την εξαγωγή αξιόπιστων στατιστικών συμπερασμάτων. Αυτές οι μέθοδοι μπορούν, ωστόσο, να χρησιμοποιηθούν για τη δημιουργία νέων υποθέσεων για έλεγχο.

2.8. Μηχανική Μάθηση (Machine Learning)

Η μηχανική μάθηση (ML) είναι η μελέτη αλγορίθμων που εκπαιδεύονται μέσω της εμπειρίας και της χρήσης δεδομένων και αποτελεί μέρος της τεχνητής νοημοσύνης. Οι αλγόριθμοι μηχανικής μάθησης δημιουργούν ένα μοντέλο βασισμένο σε δείγματα δεδομένων, γνωστά ως "δεδομένα εκπαίδευσης", προκειμένου να πραγματοποιούν προβλέψεις ή να λαμβάνουν αποφάσεις χωρίς να έχουν προγραμματιστεί ρητά να το κάνουν[35]. Οι αλγόριθμοι μηχανικής μάθησης χρησιμοποιούνται σε μεγάλο εύρος εφαρμογών, όπως στην ιατρική, το φιλτράρισμα email, την αναγνώριση ομιλίας και τη μηχανική όραση, όπου είναι δύσκολο ή ανέφικτο να αναπτυχθούν συμβατικοί αλγόριθμοι για την εκτέλεση των απαιτούμενων εργασιών[36].

Ένα υποσύνολο της μηχανικής μάθησης σχετίζεται στενά με την υπολογιστική στατιστική (computational statistics), η οποία επικεντρώνεται στην πραγματοποίηση προβλέψεων

χρησιμοποιώντας υπολογιστές. Η μελέτη της μαθηματικής βελτιστοποίησης παρέχει μεθόδους, θεωρίες και πεδία εφαρμογών στον τομέα της μηχανικής μάθησης. Η εξόρυξη δεδομένων είναι ένας σχετικός τομέας μελέτης, που εστιάζει στη διερευνητική ανάλυση δεδομένων μέσω μη εποπτευόμενης μάθησης[37]. Στην εφαρμογή της σε επιχειρηματικά προβλήματα, η μηχανική μάθηση αναφέρεται επίσης ως προγνωστική ανάλυση.

2.8.1. Προσεγγίσεις Μηχανικής Μάθησης (Machine Learning Approaches)

Οι προσεγγίσεις μηχανικής μάθησης χωρίζονται παραδοσιακά σε τέσσερεις ευρείες κατηγορίες, ανάλογα με τη φύση του εκπαιδευτικού "σήματος" ή την "ανατροφοδότηση" που είναι διαθέσιμα σε ένα σύστημα μάθησης:

- Εποπτευόμενη μάθηση (Supervised Learning - SL): Το υπολογιστικό πρόγραμμα δέχεται τις παραδειγματικές εισόδους καθώς και τα επιθυμητά αποτελέσματα από έναν “δάσκαλο”, και ο στόχος είναι να μάθει έναν γενικό κανόνα προκειμένου να αντιστοιχίσει τις εισόδους με τα αποτελέσματα[38].
- Μη Εποπτευόμενη μάθηση (Unsupervised Learning - UL): Ο σκοπός αυτής της κατηγορίας είναι η εύρεση της δομής των δεδομένων εισόδου. Η μη εποπτευόμενη μάθηση μπορεί να είναι αυτοσκοπός, ανακαλύπτοντας κρυμμένα μοτίβα σε δεδομένα, ή μέσο επίτευξης (χαρακτηριστικό της μάθησης)[39].
- Ενισχυτική Μάθηση (Reinforcement Learning - RL): Ένα πρόγραμμα υπολογιστή αλληλοεπιδρά με ένα δυναμικό περιβάλλον στο οποίο πρέπει να επιτύχει έναν συγκεκριμένο στόχο (όπως οδήγηση οχήματος ή νίκη ενός παιχνιδιού εναντίον κάποιου αντιπάλου). Καθώς πλοηγείται στον χώρο του προβλήματος, το πρόγραμμα δέχεται ανατροφοδότηση, γνωστή και ως "ανταμοιβή", την οποία προσπαθεί να μεγιστοποιήσει. Η ανταμοιβή μπορεί να είναι είτε θετική είτε αρνητική, ανάλογα με την πορεία του προγράμματος προς τον στόχο[40].
- Μείωση της Διαστασιμότητας (Dimensionality Reduction – DR): Η μείωση διαστάσεων είναι μια διαδικασία μείωσης του αριθμού των τυχαίων μεταβλητών υπό εξέταση, λαμβάνοντας ένα σύνολο κύριων μεταβλητών κατά τη φάση της εκπαίδευσης ενός μοντέλου[41] και πολλές φορές θεωρείται υποκατηγορία της μη-εποπτευόμενης μάθησης. Με άλλα λόγια, πρόκειται για μια διαδικασία μείωσης της διάστασης του συνόλου λειτουργιών, που ονομάζονται επίσης "χαρακτηριστικά". Οι περισσότερες από τις τεχνικές μείωσης διαστάσεων καλούνται είτε τεχνικές εξάλειψης χαρακτηριστικών είτε τεχνικές εξαγωγής χαρακτηριστικών. Μία από τις δημοφιλείς μεθόδους μείωσης διαστάσεων ονομάζεται βασική ανάλυση συνιστωσών (Principal

Component Analysis – PCA)[42]. Για παράδειγμα, ο αλγόριθμος PCA περιλαμβάνει τον μετασχηματισμό δεδομένων υψηλότερων διαστάσεων (π.χ. 3D) σε μικρότερο χώρο (π.χ. 2D). Αυτό έχει ως αποτέλεσμα ένα σετ δεδομένων μικρότερης διάστασης, (2D αντί για 3D) διατηρώντας παράλληλα όλες τις αρχικές μεταβλητές στο μοντέλο χωρίς αλλαγή των δεδομένων.

2.9. Αφηρημένα Συντακτικά Δένδρα (Abstract Syntactic Trees – ASTs)

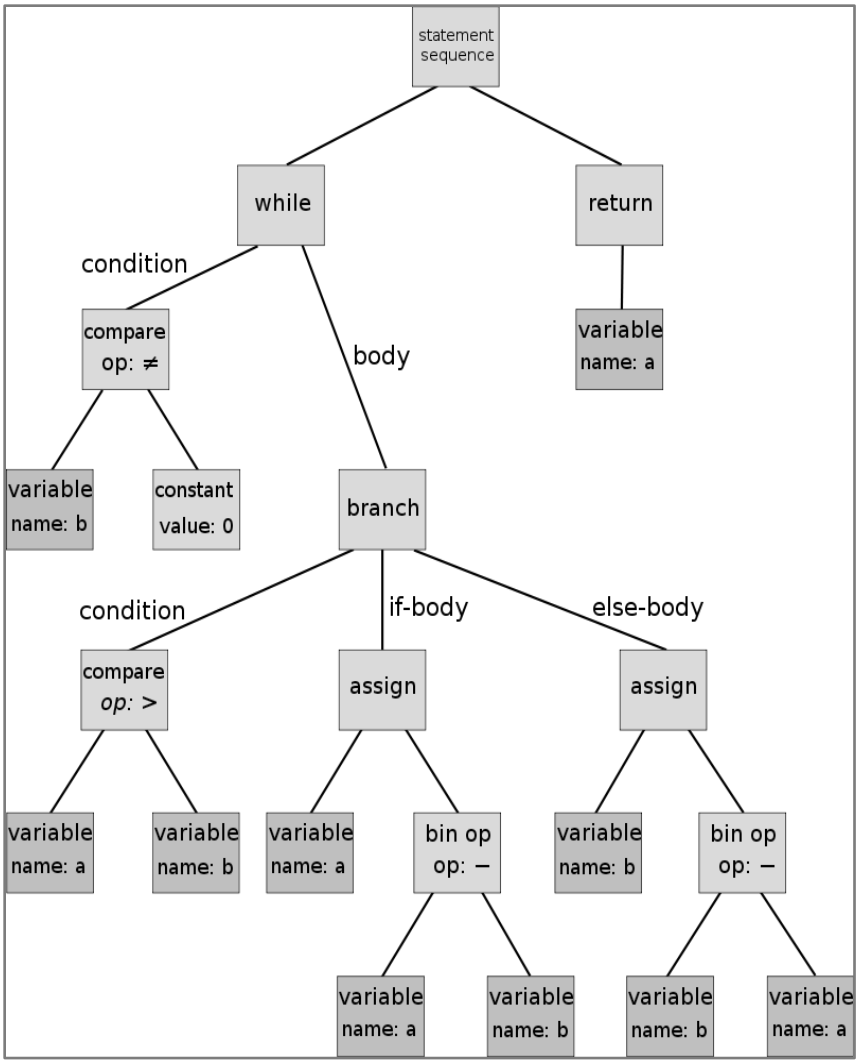
Στην επιστήμη των υπολογιστών, ένα αφηρημένο συντακτικό δέντρο (AST), είναι ένα δέντρο που αντιπροσωπεύει την συντακτική δομή του πηγαίου κώδικα με αφηρημένο τρόπο. Κάθε κόμβος του δέντρου υποδηλώνει τα διαφορετικά δομικά στοιχεία του πηγαίου κώδικα. Η σύνταξη είναι "αφηρημένη" με την έννοια ότι δεν αντιπροσωπεύει κάθε λεπτομέρεια που εμφανίζεται στην πραγματική σύνταξη, αλλά μόνο τις δομικές ή σχετικές με το περιεχόμενο λεπτομέρειες. Για παράδειγμα, η ομαδοποίηση παρενθέσεων δεν είναι εμφανής στη δομή του δέντρου, επομένως αυτές δε χρειάζεται να αναπαρασταθούν ως ξεχωριστοί κόμβοι. Ομοίως, ένα συντακτικό κατασκεύασμα όπως μια έκφραση if-condition-then μπορεί να συμβολίζεται μέσω ενός μόνο κόμβου με τρεις κλάδους. Αυτό διακρίνει τα αφηρημένα συντακτικά δέντρα από τα παραδοσιακά δέντρα σύνταξης (parse trees).

Τα parse trees δημιουργούνται συνήθως από έναν αναλυτή (parser) κατά τη διαδικασία μετάφρασης και μεταγλώττισης του πηγαίου κώδικα (translation and compiling process). Μόλις δημιουργηθεί το AST, προστίθενται πληροφορίες μέσω μεταγενέστερης επεξεργασίας, π.χ. ανάλυση συμφραζομένων. Τα αφηρημένα δέντρα σύνταξης χρησιμοποιούνται επίσης σε συστήματα ανάλυσης και μετασχηματισμού προγραμμάτων. Ένα παράδειγμα ενός AST δέντρου παρουσιάζεται στην Εικόνα 2.5, το οποίο απεικονίζει τον Ευκλείδειο αλγόριθμο του Πίνακα 2.3:

Πίνακας 2.3 Ευκλείδειος Αλγόριθμος

Ευκλείδειος Αλγόριθμος
1. while b ≠ 0: 2. if a > b: 3. a = a – b 4. else: 5. b = b – a 6. return a

Τα ASTs, επίσης, είναι δομές δεδομένων που χρησιμοποιούνται ευρέως σε μεταγλωττιστές για να αντιπροσωπεύσουν τη δομή του κώδικα προγράμματος. Ένα AST είναι συνήθως το αποτέλεσμα της φάσης ανάλυσης σύνταξης ενός μεταγλωττιστή. Συχνά χρησιμεύει ως ενδιάμεση αναπαράσταση του προγράμματος σε διάφορα στάδια που απαιτεί ο μεταγλωττιστής και έχει ισχυρό αντίκτυπο στην τελική απόδοση του μεταγλωττιστή.



Εικόνα 2.5 Αφηρημένο Συντακτικό Δέντρο για τον Euclidean Αλγόριθμο.

Κεφάλαιο 3. Σχετική Βιβλιογραφία

3.1. Γενικά

Η εφαρμογή τεχνικών εξόρυξης δεδομένων σε αποθετήρια ανοιχτού λογισμικού με σκοπό τη μελέτη της εξέλιξης τμημάτων κώδικα, είναι ένα σχετικά καινούριο πεδίο έρευνας, το οποίο έχει απασχολήσει σοβαρά τους διάφορους ερευνητές ανά τον κόσμο τα τελευταία χρόνια. Αναμφισβήτητα, φιλοδοξεί να θέσει τα θεμέλια της αυτόματης παραγωγής κώδικα από συστήματα τεχνητής νοημοσύνης, κάτι το οποίο μέχρι σήμερα μπορεί να προσεγγιστεί από την αναζήτηση παρόμοιων αλλαγών που είχαν διενεργηθεί από προγραμματιστές σε προηγούμενα έργα λογισμικού. Αυτά τα πρότυπα αλλαγών μπορεί να αναφέρονται σε προτάσεις διόρθωσης συγκεκριμένου τύπου σφαλμάτων, σε συχνές, συστηματικές τροποποιήσεις, σε βελτιώσεις στο εσωτερικό τμημάτων με τέτοιον τρόπο ώστε να μην επηρεάζεται η συνολική λειτουργία τους, σε χρήση υπαρχόντων σχολίων που περιγράφουν την ίδια ακριβώς διαδικασία αλλαγής, αλλά και σε παρεμβάσεις σε επίπεδο αρχιτεκτονικής.

Στη συνέχεια λοιπόν, παρουσιάζονται ορισμένες τεχνικές και εργαλεία, τα οποία χρησιμοποιήθηκαν για τη συλλογή των απαραίτητων δεδομένων από συστήματα ελέγχου εκδόσεων λογισμικού, με την εκμετάλλευση των οποίων, ερευνητές οδηγήθηκαν στην υλοποίηση συστημάτων που προτείνουν διαφόρων ειδών τροποποιήσεις σε ένα τμήμα κώδικα που δίνεται στην είσοδό τους.

3.2. Εξόρυξη Αποθετηρίων Λογισμικού (Mining Software Repositories – MSR)

Κατά την προηγούμενη δεκαετία, η λήψη αποφάσεων σε μεγάλους οργανισμούς που ασχολούνται με την ανάπτυξη λογισμικού ήταν απολύτως επηρεασμένη από προηγούμενες εμπειρίες των υπευθύνων ή από κυρίαρχα μοτίβα που ακολουθούσαν διάφοροι οργανισμοί της αγοράς. Οι επαγγελματίες συχνά βασίζονταν στην εμπειρία, τη διαίσθηση και το ένστικτο τους κατά τη λήψη σημαντικών αποφάσεων. Οι υπεύθυνοι των οργανισμών αυτών, παρείχαν στους προγραμματιστές πόρους ανάπτυξης και δοκιμών λογισμικού, με βάση την εμπειρία τους σε προηγούμενα έργα και τη διαίσθησή τους σχετικά με την πολυπλοκότητα του νέου έργου. Οι προγραμματιστές χρησιμοποιούσαν συνήθως την εμπειρία τους όταν επρόκειτο να υλοποιήσουν μια νέα λειτουργία ή να διορθώσουν κάποιο σφάλμα. Οι υπεύθυνοι δοκιμών συνήθως έδιναν προτεραιότητα στη δοκιμή χαρακτηριστικών, που ήταν γνωστό ότι ήταν επιρρεπή σε σφάλματα, βασιζόμενοι σε γνωστές αναφορές σφαλμάτων επί του θέματος.

Τα αποθετήρια λογισμικού, σήμερα, περιέχουν πληθώρα πολύτιμων πληροφοριών σχετικά με την ανάπτυξη λογισμικού. Χρησιμοποιώντας λοιπόν τις πληροφορίες που είναι αποθηκευμένες στα αποθετήρια, οι επαγγελματίες μπορούν πλέον να εξαρτηθούν λιγότερο από τη διαίσθηση και την

εμπειρία τους, αλλά περισσότερο από τα ιστορικά δεδομένα του εκάστοτε πεδίου. Το πεδίο του MSR αναλύει και διασυνδέει τα πλούσια δεδομένα που είναι διαθέσιμα σε αποθετήρια λογισμικού, με σκοπό την ανακάλυψη σημαντικών πληροφοριών σχετικά με διάφορα συστήματα και έργα λογισμικού[43]. Παραδείγματα αποθετηρίων λογισμικού είναι:

- Αποθετήρια Ιστορικού (Historical Repositories): Αποτελούνται από αποθετήρια ελέγχου πηγαίου κώδικα, αποθετήρια σφαλμάτων και αρχειοθετημένες επικοινωνίες στα οποία καταγράφονται πληροφορίες σχετικές με την εξέλιξη και την πρόοδο ενός έργου λογισμικού.
- Αποθετήρια Εκτέλεσης Λογισμικού (Run-Time Repositories): Απαρτίζονται από αρχεία καταγραφής ανάπτυξης λογισμικού (deployment logs) , τα οποία περιέχουν πληροφορίες σχετικά με την εκτέλεση και τη χρήση μιας εφαρμογής.
- Αποθετήρια Κώδικα (Code repositories): Παραδείγματα αποτελούν το Sourceforge.net[16] και το code.google[17], καθώς και όσα αναφέρθηκαν στο υποκεφάλαιο 2.2, τα οποία συμπεριλαμβάνουν τον πηγαίο κώδικα εφαρμογών που αναπτύχθηκαν από διάφορους προγραμματιστές.

Τα αποθετήρια λογισμικού χρησιμοποιούνται συνήθως στην πράξη ως αποθετήρια διατήρησης αρχείων και σπάνια χρησιμοποιούνται για υποστήριξη διαδικασιών λήψης αποφάσεων[43]. Βέβαια, το γεγονός αυτό καλείται να αντιμετωπίσει η εξόρυξη αποθετηρίων λογισμικού. Η έρευνα που πραγματοποιείται στο πεδίο του MSR επικεντρώνεται κυρίως σε δύο τομείς:

- Τη δημιουργία τεχνικών αυτοματοποίησης και βελτίωσης της εξαγωγής πληροφοριών από αποθετήρια.
- Την ανακάλυψη και την επικύρωση νέων τεχνικών και προσεγγίσεων για την εξόρυξη σημαντικών πληροφοριών από αυτά τα αποθετήρια.

Αν και οι δύο παραπάνω περιπτώσεις ακούγονται πανομοιότυπες, αποσκοπούν σε διαφορετικά αποτελέσματα. Η πρώτη είναι απαραίτητη για να διευκολυνθεί η υιοθεσία τεχνικών MSR από διάφορους οργανισμούς και εταιρείες τεχνολογίας, ενώ η δεύτερη καταδεικνύει τη σημασία και την αξία της πληροφορίας που μπορεί να εξαχθεί από αποθετήρια λογισμικού με σκοπό την εξέλιξη της επιστημονικής έρευνας επί του θέματος[43]. Τέλος, οι ερευνητές μηχανικής λογισμικού καθιστούν απολύτως σαφές, πως τα οφέλη της ανάπτυξης του MSR είναι πολύ σημαντικά για την υποστήριξη της

[16] https://sourceforge.net/
[17] https://code.google.com/

συντήρησης και της εξέλιξης των συστημάτων λογισμικού, τη βελτίωση της επαναχρησιμοποίησης κώδικα, καθώς και την εμπειρική επικύρωση νέων ιδεών και τεχνικών[44].

3.2.1. Εργαλεία Εξόρυξης Αποθετηρίων Λογισμικού

Οι περισσότεροι επιστήμονες, για τη διεξαγωγή των ερευνών που έχουν πραγματοποιηθεί στη διεθνή βιβλιογραφία στο θέμα της εξόρυξης αποθετηρίων, χρησιμοποίησαν ήδη αναπτυγμένα εργαλεία για εργασίες εξόρυξης δεδομένων σε αποθήκες μηχανικής λογισμικού. Λίγοι από αυτούς ανέπτυξαν νέα εργαλεία και πηγαίο κώδικα για την απάντηση των ερευνητικών τους ερωτημάτων.

Αυτά τα εργαλεία, τα οποία χρησιμοποιούνται κυρίως για την εξαγωγή δεδομένων από αποθετήρια, απαιτούν φιλτράρισμα δεδομένων, αναγνώριση προτύπων, μάθηση και πρόβλεψη. Μερικοί ερευνητές βέβαια, ανέπτυξαν εργαλεία με σκοπό τον σχεδιασμό της ροής εργασίας, από την ανάκτηση των δεδομένων έως την ολοκλήρωση των προβλέψεων, ενώ άλλοι είχαν ως στόχο τη συγγραφή προσαρμοσμένων αρχείων πηγαίου κώδικα, σε διάφορες γλώσσες προγραμματισμού[44].

Σε λίγες εργασίες, ερευνητές έχουν υλοποιήσει τον πηγαίο κώδικα σε γλώσσα προγραμματισμού python[18] ή Java, που χρησιμοποιούνται ως “plug-in” για το περιβάλλον eclipse[19]. Τα εργαλεία που αναφέρονται ως “newly developed or created” στον Πίνακα 3.1, ενσωματώνονται επίσης στο περιβάλλον eclipse ή σε ορισμένα ήδη καθιερωμένα εργαλεία εξόρυξης δεδομένων. Μερικοί ερευνητές υλοποίησαν την ανάπτυξη νέων εργαλείων, τα οποία χαρακτηρίζονται ως “prototypes”. Υπάρχουν επίσης ερευνητές που δεν αναφέρουν τα εργαλεία που χρησιμοποιήθηκαν για την ανάλυση που διεξήγαγαν. Αυτή η ανάλυση ποικίλλει σε μορφή, χρησιμοποιώντας διάφορες τεχνολογίες και αλγορίθμους, με κύριες περιπτώσεις την παλινδρόμηση (regression), τη συσχέτιση (correlation), τη λογιστική παλινδρόμηση (logistic regression) και άλλου είδους τεχνικές. Τα εργαλεία ταξινομούνται σε τέσσερις κατηγορίες: “newly developed tools”, “traditional data mining tools”, “prototype development/implementation” και “scripts”. Η κατηγοριοποιημένη λίστα εργαλείων παρουσιάζεται στον Πίνακα 3.1.

Πίνακας 3.1 MSR Tools[44].

Types	Tools
Newly Developed	Exempler, CloneTracker, sqminer, CallExtractor, J-REX on MapReduce, REXML (Ruby for XML), Ruby Porter Stammer and Ruby Classifier, FPLearner, FPClassifier, EvoOnt, iSPARQL, Deep Intellisense, Operation Recorder, SpotWeb, PopCon, CP-Miner, PatchMiner, Binary Analysis Tool (BAT), Anchored signature matching and

[18] https://www.python.org/
[19] https://www.eclipse.org/

	Rationalizer, SeCold, rodrigorgs2, checkstyle, Apache Pig, Marmoset, Evolizer, Swanson's Maintenace Classification.
Traditional data mining tools	CodeFinder, CodeBroker, R Tools, WEKA toolkit, GIT and GitMining Tools, KEEL, RapidMiner, SAS Text Miner, SPSS Clementine, GNUPlot, Matlab, CCFinder, SourceMonitor, MALLET, CTAGS, CHANGEDISTILLER and EVOLIZER suite, UCINET, UNDERSTAND, Unix Diff utility, CVSAnalY tool, FLOSSmole, Simian, Statistica.
Prototype	Bug report assignment on eclipse, Automatic Labeling of Software Components in Java, How developers work together in Tomcat and Junit, defect correction effort using the defect data collected from a Japanese multi-vendor information system development project COSE, which methods are cloned and which methods are changed in each transaction in DNSJava, Renaming recommendation system.
Scripts	Eclipse, FreeBSD, GNOME, Apache and Apache Ant, Java Generics adoption.

3.3. Εξόρυξη Τροποποιήσεων Πηγαίου Κώδικα

Σε μεγάλο αριθμό εργασιών τροποποίησης λογισμικού, ο πηγαίος κώδικας ενός συστήματος επιδέχεται αλλαγές από τους επαγγελματίες σε διαφορετικά μέρη [45]. Ένας προγραμματιστής, για να προσδιορίσει τα μέρη εκείνα του κώδικα, τα οποία σχετίζονται με την εργασία που έχει να φέρει εις πέρας, έχει τη δυνατότητα να χρησιμοποιήσει διάφορα εργαλεία που αναλύουν στατικά ή δυναμικά τις εξαρτήσεις μεταξύ τμημάτων του πηγαίου κώδικα (π.χ., [46, 47]). Η ανάλυση αυτή, μπορεί να βοηθήσει έναν προγραμματιστή να εντοπίσει τα σημεία του κώδικα που τον ενδιαφέρουν, αλλά δε μπορεί πάντα να προσδιορίσει όλα τα κομμάτια του λογισμικού που σχετίζονται με την αλλαγή. Για παράδειγμα, χρησιμοποιώντας αυτές τις αναλύσεις, είναι συνήθως δύσκολο να εντοπιστούν εξαρτήσεις μεταξύ "platform-dependent[20]" ενοτήτων καθώς και μεταξύ ενοτήτων που είναι γραμμένες σε διαφορετικές γλώσσες προγραμματισμού.

Επιπρόσθετα, ένα από τα πιο απαιτητικά προβλήματα που καλούνται να λύσουν οι μηχανικοί λογισμικού, είναι η εύρεση των κατάλληλων εντολών, με τις οποίες θα καταφέρουν να φέρουν εις πέρας την επίλυση διαφόρων σφαλμάτων που εμφανίζονται κατά τη διαδικασία ανάπτυξης ενός έργου λογισμικού. Βέβαια, φαίνεται πως αρκετές φορές τα σφάλματα αυτά επαναλαμβάνονται σε πληθώρα διαφορετικών έργων λογισμικού, κάποια εκ των οποίων ενδέχεται να δώσουν λύση σε παρόμοια προβλήματα. Στη συνέχεια παρουσιάζονται προτάσεις και έρευνες που πραγματεύονται τα εν λόγω ζητήματα.

[20] Συνήθως αναφέρεται σε εφαρμογές που λειτουργούν με ένα μόνο λειτουργικό σύστημα σε μια σειρά υπολογιστών.

3.3.1. Πρόβλεψη Αλλαγών Πηγαίου Κώδικα Εξορύσσοντας το Ιστορικό Αλλαγών

Μία προσέγγιση για την επίλυση του παραπάνω προβλήματος επιχειρούν να δώσουν το 2004 οι Annie T.T. Ying et al. με την ερευνητική εργασία "Predicting Source Code Changes by Mining Change History"[48]. Η προσπάθεια του έργου αυτού επικεντρώνεται κυρίως στην εξόρυξη προτύπων τροποποιήσεων – αρχεία τα οποία αλλάζουν σε ζεύγη συχνά στο παρελθόν – από το ιστορικό αλλαγών του πηγαίου κώδικα του συστήματος. Τα πρότυπα αυτά, επίσης, μπορούν να χρησιμοποιηθούν για την ένδειξη πιθανώς σχετικών αρχείων κατά τη διάρκεια μια εργασίας τροποποίησης λογισμικού.

Στο εν λόγο σύστημα έγινε χρήση του αλγορίθμου "frequent pattern mining"[49], ο οποίος βασίζεται στην αρίθμηση της συχνότητας εμφάνισης ενός προτύπου (association rule mining algorithm). Για την αξιολόγηση της χρησιμότητας της προσέγγισής που προτείνουν, οι Annie T.T. Ying et al. ελέγχουν τις συστάσεις του εργαλείου που κατασκεύασαν σε δύο μεγάλα έργα λογισμικού ανοιχτού κώδικα, το Eclipse[19] και το Mozilla[21], βασιζόμενοι στη προβλεψιμότητα (predictability) και το ενδιαφέρον (interestingness) των συστάσεων.

Η "προβλεψιμότητα" αξιολογεί ποσοτικά τις συστάσεις, δηλαδή αντικατοπτρίζει τον αριθμό εμφάνισης τους στο σύνολο των τροποποιήσεων. Το "ενδιαφέρον" των συστάσεων αξιολογείται κυρίως καθορίζοντας αν ένα προτεινόμενο αρχείο έχει ισχυρές ή όχι δομικές εξαρτήσεις με το αρχείο που ξεκινά να επεξεργάζεται ο προγραμματιστής. Αρχεία που δε σχετίζονται δομικά θεωρούνται πιο ενδιαφέροντα, καθώς δε μπορούν να προσδιοριστούν εύκολα χρησιμοποιώντας υπάρχουσες στατικές και δυναμικές αναλύσεις.

Το σύστημα που αναπτύχθηκε στα πλαίσια της έρευνας αποτελείται από τρία στάδια:

- Στο πρώτο στάδιο πραγματοποιούνται: η εξαγωγή των δεδομένων, από ένα σύστημα διαχείρισης διαμόρφωσης λογισμικού (Software Configuration Management System – SCMS) και η προεπεξεργασία τους, με στόχο την κατάλληλη διαμόρφωση ώστε να χρησιμοποιηθούν για είσοδο στον αλγόριθμο εξόρυξης δεδομένων - association rule mining algorithm.
- Στο δεύτερο στάδιο εφαρμόζεται ο αλγόριθμος με στόχο την ανακάλυψη των προτύπων στις τροποποιήσεις των αρχείων πηγαίου κώδικα.

[21] https://www.mozilla.org/el/

- Στο τρίτο και τελευταίο στάδιο, λαμβάνουν μέρος οι προτάσεις συσχετιζόμενων αρχείων που εξάγει το σύστημα, σύμφωνα με τον πηγαίο κώδικα που δίνεται ως είσοδος, χρησιμοποιώντας "query script" στα πρότυπα τροποποιήσεων.

Το πρώτο βήμα προεπεξεργασίας περιλαμβάνει τον προσδιορισμό των συνόλων αρχείων λογισμικού που καταχωρήθηκαν στο σύστημα μαζί. Το δεύτερο βήμα προεπεξεργασίας περιλαμβάνει την διαγραφή συναλλαγών που αποτελούνται από περισσότερα από 100 αρχεία, επειδή αυτές οι περιπτώσεις συνήθως δεν αντιστοιχούν σε ουσιαστικές αλλαγές, όπως για παράδειγμα οι διορθώσεις σφαλμάτων.

Ο αλγόριθμος που επιλέχθηκε στη συγκεκριμένη περίπτωση για την εύρεση συχνών προτύπων, χρησιμοποιεί μία συμπαγή δομή δεδομένων που ονομάζεται FP-tree για την κωδικοποίηση μιας βάσης δεδομένων[54]. Η ιδέα του αλγορίθμου είναι να βρίσκει πρότυπα χρησιμοποιώντας μια προσέγγιση βάθους (depthfirst approach), η οποία υλοποιεί αναδρομική εξόρυξη ενός προτύπου. Η μέθοδος αυτή αναζητά αναδρομικά τη εγγραφή με τη μεγαλύτερη τιμή εμφάνισης στη δομή δεδομένων FP-tree σε αντίθεση με την προσέγγιση πλάτους (breadth-first approach), η οποία αναζητά όλα τα πρότυπα με τον ίδιο αριθμό εμφάνισης πριν βρεθούν πρότυπα ενός μεγαλύτερου αριθμού εμφανίσεων.

Η εφαρμογή του αλγορίθμου στα προεπεξεργασμένα δεδομένα καταλήγει σε μια συλλογή προτύπων τροποποιήσεων. Κάθε πρότυπο τροποποίησης αποτελείται από σύνολα που περιλαμβάνουν τα ονόματα πηγαίων αρχείων που έχουν τροποποιηθεί σε ζεύγη συχνά στο παρελθόν. Για την παροχή μίας πρότασης αρχείων που σχετίζονται με μια συγκεκριμένη εργασία τροποποίησης πηγαίου κώδικα, ο προγραμματιστής πρέπει να παραθέσει το όνομα τουλάχιστον ενός αρχείου που πιθανώς εμπλέκεται σε αυτήν. Τα αρχεία που προτείνονται, χρησιμοποιώντας ένα "query script" πάνω στα δεδομένα των προτύπων, καθορίζονται ψάχνοντας τις εγγραφές που περιλαμβάνουν το προσδιορισμένο αρχικό αρχείο. Χρησιμοποιείται ο συμβολισμός:

$$Recomm(f_s) = f_R \tag{3.1}$$

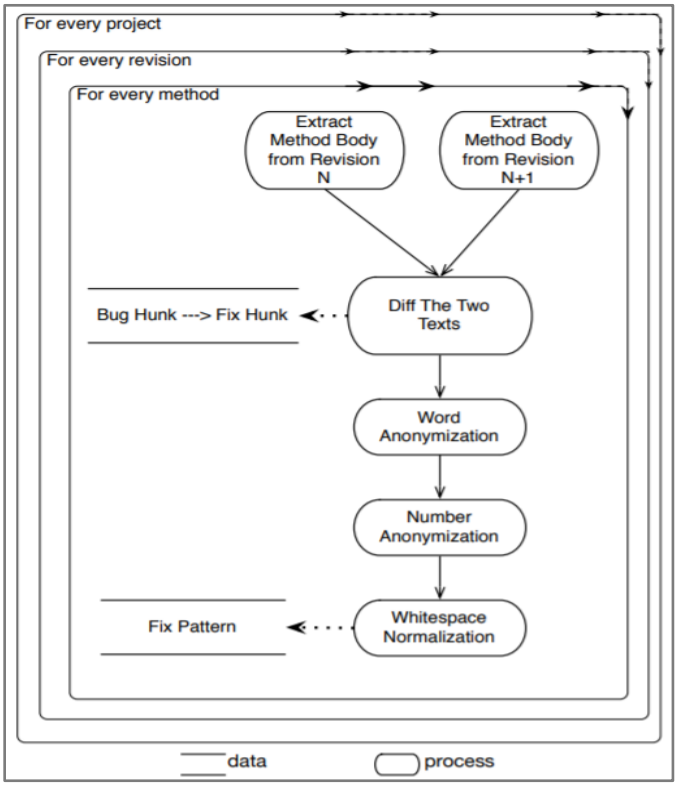
Αυτό σημαίνει ότι το σύνολο των αρχείων f_s οδηγεί στη σύσταση του συνόλου αρχείων f_R. Το σύνολο προτάσεων f_R είναι η ένωση αντιστοιχισμένων προτύπων - προτύπων με τουλάχιστον ένα αρχείο από το f_s — χωρίς να περιλαμβάνεται το f_s.

3.3.2. Εξόρυξη Συχνών Αλλαγών Διόρθωσης Σφαλμάτων

Το 2004 υλοποιείται από τους H.Osman et al. μία έρευνα με τίτλο "Mining Frequent Bug-Fix Code Changes"[55] για τον εντοπισμό και την εξαγωγή μοτίβων διόρθωσης σφαλμάτων (bug-fixing patterns) από 717 αποθετήρια ανοικτού λογισμικού σε γλώσσα Java. Οι συγγραφείς εξάγουν τμήματα κώδικα πριν

και μετά αντίστοιχα από τη διόρθωση ενός σφάλματος χρησιμοποιώντας τη βιβλιοθήκη JAPA[22] για την ανάλυση (parsing) του πηγαίου κώδικα. Ακολούθως, γίνεται μία αξιολόγηση των σημαντικότερων και επαναλαμβανόμενων προτύπων αλλαγών στον κώδικα, καθώς και των αιτιών εμφάνισης τους.

Συνοπτικά η μεθοδολογία λειτουργίας του συστήματος ακολουθεί τα παρακάτω βήματα. Αρχικά, γίνεται αντιγραφή (cloning) 717 δημοφιλών Java projects από το Github σε τοπικό επίπεδο. Στη συνέχεια ακολουθεί η ανάλυση του πηγαίου κώδικα και του ιστορικού αλλαγών (Change History), όπως φαίνεται στην Εικόνα 3.1.



Εικόνα 3.1 Διαδικασία Ανάλυσης Πηγαίου Κώδικα και Εξαγωγής Προτύπων Διορθώσεων

Γίνεται, δηλαδή, εντοπισμός των αλλαγών διόρθωσης σφαλμάτων (bug fixing commits), θεωρώντας πως πρέπει να εμπεριέχουν τη λέξη “fix” στο commit message. Ακολουθεί η εξαγωγή δύο εκδόσεων για κάθε αλλαγμένη μέθοδο, την έκδοση πριν από την αλλαγή $revision_N$ και την έκδοση μετά την αλλαγή $revision_{N+1}$. Οι δύο εκδόσεις αυτές συγκρίνονται και τελικά διατηρείται μόνο το κομμάτι του κώδικα που διαφέρει μεταξύ των δύο και αναπαριστά και την αλλαγή. Τέλος, γίνεται κανονικοποίηση των παραγόμενων προτύπων ως εξής: Όλες οι μεταβλητές αντικαθίστανται από το γράμμα “T”, εκτός από τις

[22] https://www.tabnine.com/code/java/packages/japa.parser

δεσμευμένες λέξεις της Java. Όλοι οι αριθμοί αντικαθίστανται από τον αριθμό “0” και οποιαδήποτε αλληλουχία κενών χαρακτήρων (white space character) με ένα κενό χαρακτήρα “ ”. Έτσι τελικώς παραμένουν μοτίβα αλλαγών της μορφής: T.T(T); >>>> T.T(T);

Η αξιολόγηση των μοτίβων διόρθωσης βασίζεται στην εμφάνιση τους μεταξύ διαφορετικών έργων λογισμικού, δηλαδή σε όσα περισσότερα αποθετήρια καταγράφεται μία αλλαγή τόσο σημαντικότερη θεωρείται. Οι συγγραφείς αναλύουν συνολικά 190.821 αλλαγές κώδικα που αντιστοιχούν σε 94.534 bug-fixing commits. Προέκυψε πως από αυτά, το 53% αφορά αλλαγές σε επίπεδο μίας γραμμής κώδικα, ενώ το 73% αφορά commits με αλλαγές το πολύ τεσσάρων γραμμών κώδικα. Επιπλέον, ένα ποσοστό 40% των αλλαγών αυτών απαντάται σε τουλάχιστον 10 διαφορετικά έργα λογισμικού, γεγονός που καθιστά μία αυτοματοποιημένη προσέγγιση διόρθωσης σφαλμάτων από τα έργα λογισμικού του GitHub δυνατή.

Στα μειονεκτήματα του συστήματος προσμετράται η απουσία χρήσης αφαιρετικών αναπαραστάσεων κώδικα (π.χ. ASTs) σε συνδυασμό με την ήδη υπάρχουσα μεθοδολογία για την παραγωγή καλύτερων αποτελεσμάτων. Επιπλέον, οι ευρετικές τεχνικές που χρησιμοποιούνται για την αρχική επιλογή των bug-fixing commits είναι αρκετά απλοϊκές και μπορούν να βελτιωθούν σημαντικά μέσω μίας σημασιολογικής ανάλυσης κειμένου.

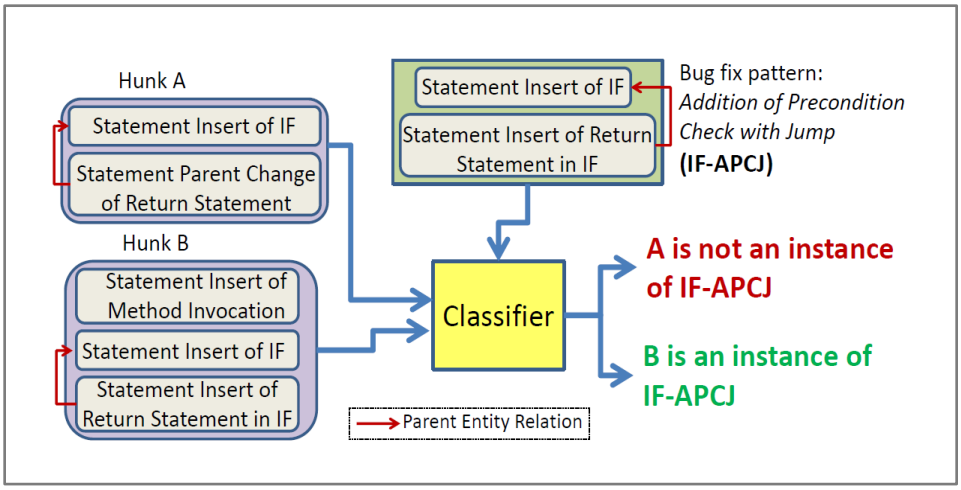
3.3.3. Αυτόματη Εξαγωγή Προτύπων Αλλαγών Πηγαίου Κώδικα με Ανάλυση AST

Κίνητρο της παρούσας έρευνας με τίτλο: “Automatically Extracting Instances of Code Change Patterns with AST Analysis”[56], η οποία πραγματοποιήθηκε από τους Matias Martinez et al. το 2014, αποτελεί η παροχή ενός γενικού τρόπου προσδιορισμού προτύπων αλλαγών πηγαίου κώδικα.

Η τεχνική που χρησιμοποιήθηκε βασίζεται στην αυτοματοποιημένη ανάλυση διαφορών μεταξύ αφηρημένων συντακτικών δένδρων (ASTs). Οι Matias Martinez et al. χρησιμοποιούν την ταξινομία αλλαγών που εισήγαγαν οι Fluri et al.[57], το 2007. Η ολοκληρωμένη ταξινομία αλλαγών μπορεί να βρεθεί εσωτερικά της έρευνας των Fluri et al.. Η ταυτοποίηση ενός προτύπου αλλαγών, πραγματοποιείται στις εκδόσεις, οι οποίες περιλαμβάνουν τις AST αλλαγές που περιέχονται και στο πρότυπο. Αυτό συμβαίνει με τον υπολογισμό των AST διαφορών κάθε ζεύγους αρχείων που τροποποιούνται στα commits (της νέας έκδοσης καθώς και των προγόνων της). Για το 2014, αυτός ο τρόπος ορισμού και η ποσοτικοποίηση των προτύπων αλλαγών ήταν καινοτόμος. Η ανάλυση οδήγησε σε 18 πρότυπα αλλαγών, από 6 έργα λογισμικού ανοικτού κώδικα που περιλαμβάνουν πηγαίο κώδικα γραμμένο σε Java, με συνολικά 23.597 Java *revisions*.

Αρχικά, υπολογίζονται τα ASTs για ένα δεδομένο ζεύγος διαδοχικών εκδόσεων ενός πηγαίου αρχείου. Στη συνέχεια εφαρμόζεται ένας αλγόριθμος διαφοροποίησης AST δένδρων, για την εξαγωγή της ουσίας της αλλαγής. Για να υπολογιστεί ένα σύνολο AST αλλαγών μεταξύ δύο πηγαίων αρχείων κώδικα, χρησιμοποιήθηκε ο αλγόριθμος διαφοροποίησης AST, ChangeDistiller[57]. Επιλέχθηκε ο συγκεκριμένος αλγόριθμός λόγω της αποτελεσματικότητας του, από άποψη χρόνου εκτέλεσης και της διαθεσιμότητας ενός σταθερού, open-source και επαναχρησιμοποιήσιμου αλγορίθμου για την ανάλυση AST αλλαγών σε κώδικα Java. Ο ChangeDistiller παρέχει λεπτομερείς πληροφορίες σχετικά με τις διαφορές των ASTs πηγαίων αρχείων, σε επίπεδο *δηλώσεων* (statements). Ορίζει 41 τύπους αλλαγών πηγαίου κώδικα, όπως η *εισαγωγή δήλωσης* (Statement Insertion) ή η *αλλαγή έκφρασης συνθήκης* (Condition Expression Change)[58], κτλ.

Προηγούμενες εργασίες δημιούργησαν την *υπόθεση τοπικής αλλαγής*[59]. Η υπόθεση αυτή δηλώνει ότι, οι περιπτώσεις προτύπων βρίσκονται εσωτερικά του ίδιου πηγαίου αρχείου και μάλιστα εσωτερικά ενός απλού *hunk*, δηλαδή, μέσα σε μια ακολουθία διαδοχικών τροποποιημένων γραμμών. Τα AST hunks είναι συν-τοπικές αλλαγές πηγαίου κώδικα, δηλαδή αλλαγές που βρίσκονται η μία κοντά στην άλλη. Στην παρούσα εργασία υλοποιήθηκε ένας ταξινομητής αλλαγών, που αποφασίζει αν υπάρχει ή όχι ένα δεδομένο πρότυπο μέσα σε ένα AST hunk. Η Εικόνα 3.2, παρουσιάζει την διαδικασίας ταξινόμησης.



Εικόνα 3.2 Διαδικασία Ταξινόμησης ενός AST hunk.

Οι Matias Martinez et al. ορίζουν 18 πρότυπα διόρθωσης σφαλμάτων, σε μορφή AST, που ανήκουν στις κατηγορίες If, Loops, Try, Switch, Method Declaration και Assignment. Συνοπτικά, τα 18 πρότυπα που παράγονται ως αποτέλεσμα σε αυτό το έργο παρουσιάζονται στον Πίνακα 3.2.

Πίνακας 3.2 Απόλυτος Αριθμός Εμφανίσεων Των Προτύπων Επιδιόρθωσης Σφαλμάτων.

Pattern Name	Abs
Change of If Condition Expression-IF-CC	4.444
Additions of a Method Declaration-MD-ADD	4.443
Additions of a Class Field-CD-ADD	2.427
Additions of an Else Branch-IF-ABR	2.053
Change of Method Declaration-MD-CHG	1.940
Removal of a Method Declaration-MD-RMV	1.762
Removal of a Class Field-CF-RMV	983
Addition of Precond, Chech with Jump-IF-APCJ	667
Addition of a Catch Block-TY-ARCB	497
Addition of Precondition Check-IF-APC	431
Addition of Switch Branch-SW-ARSB	348
Removal of a Catch Block-SW-ARSB	348
Removal of a Catch Block-TY_ARCB	343
Removal of an IF Predicate-IF-RMV	283
Change of Loop Predicate-LP-CC	233
Removal on an Else Branch-IF-RBR	190
Removal of Switch Branch-SWARSB	146
Removal of Try Statement-TY-ARTC	26
Addition of Try Statement-TY-ARTC	18
Total	**21.234**

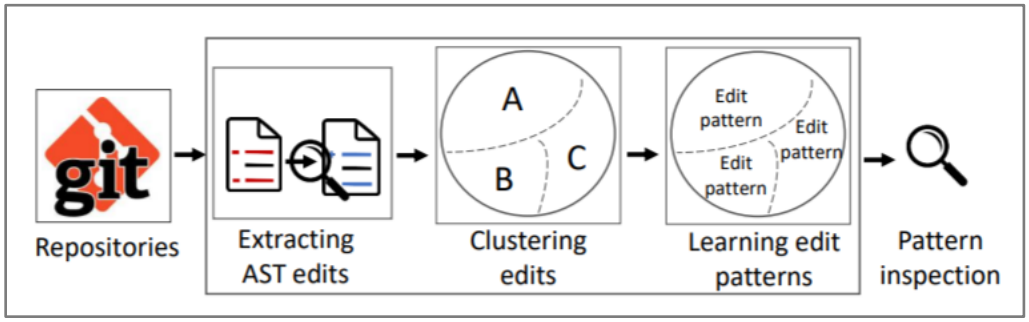
Μέσω του απόλυτου αριθμού εμφανίσεων των προτύπων επιδιόρθωσης σφαλμάτων (Abs), αποδεικνύεται ότι η συγκεκριμένη προσέγγισή κλιμακώνεται στα 23.597 Java revisions από το ιστορικό των 6 έργων ανοιχτού κώδικα. Αυτός ο πίνακας επιτρέπει τον προσδιορισμό της σημαντικότητας κάθε προτύπου διόρθωσης σφαλμάτων. Γενικά, η κατανομή των προτύπων είναι πολύ διαστρεβλωμένη, και δείχνει ότι μερικά από τα πρότυπα των Pan et al. είναι πραγματικά σπάνια στην πράξη. Είναι ενδιαφέρον ότι υπολογίστηκαν επίσης τα αποτελέσματα σε όλα τα revisions με την χρήση φίλτρου στο commit message και η κατανομή των προτύπων ήταν παρόμοια. Φαίνεται ότι η παρουσία των λέξεων bug-fix-patch στα commit messages δεν αποφέρει σημαντικά διαφορετικό σύνολο commits.

3.3.4. Εκμάθηση Γρήγορων Διορθώσεων από Αποθετήρια Κώδικα

Το 2018, ολοκληρώνεται από τους επιστήμονες R.Rolim et al. μια έρευνα με τίτλο “Learning Quick Fixes from Code Repositories”[60], η οποία υλοποιεί μια ακόμα μεθοδολογία για τον αυτόματο εντοπισμό συχνών μοτίβων επεξεργασίας κώδικα σε γλώσσα Java. Το σύστημα αυτό μπορεί να αναγνωρίσει ομάδες συχνών και γρήγορων διορθώσεων κώδικα λαμβάνοντας ως είσοδο τις διευθύνσεις αποθετηρίων κώδικα και το ιστορικό αλλαγών τους. Η μεθοδολογία βασίζεται στην αναπαράσταση του κώδικα σε ASTs, με τη χρήση του αλγορίθμου GumTree, στον υπολογισμό της απόστασης επεξεργασίας

δέντρου (tree edit distance - TED) και στην υλοποίηση ενός άπληστου αλγορίθμου ομαδοποίησης (greedy clustering algorithm).

Ακολουθεί μία περιεκτική περιγραφή της αρχιτεκτονικής του συστήματος όπως απεικονίζεται και στην Εικόνα 3.3.



Εικόνα 3.3 Αρχιτεκτονική του συστήματος REVISAR.

Αρχικά, το σύστημα αναλύει τις διαφορές μεταξύ διαδοχικών αλλαγών (revisions) r_i και r_{i+1} και παράγει τα ASTs τους t_i και t_{i+1}. Στη συνέχεια, εφαρμόζεται ο αλγόριθμος TED προκειμένου να αναγνωριστεί το σύνολο των αλλαγών, που εάν εφαρμοστούν στο δέντρο t_i παράγεται το t_{i+1}. Μία τέτοια αλλαγή σε επίπεδο δέντρου ανήκει σε τέσσερις διακριτές κατηγορίες: εισαγωγή, διαγραφή, ενημέρωση ή μετακίνηση ενός κόμβου του δέντρου. Σε επόμενη φάση, το σύστημα προσπαθεί να ομαδοποιήσει αυτά τα σύνολα αλλαγών και παράγει πρότυπα αλλαγών (edit patterns) που είναι συνεπή με όλες τις αλλαγές της εκάστοτε ομάδας. Ένα πρότυπο αλλαγών είναι ουσιαστικά ένας κανόνας της μορφής $r = \tau_i \Rightarrow \tau_{i,o}$, όπου $\tau_{i,o}$ είναι ένα πρότυπο (template) γενικευμένου δέντρου εισόδου και εξόδου αντίστοιχα.

Ένα AST t ταιριάζει σε ένα πρότυπο, εάν υπάρχει τρόπος να ανατεθούν τιμές σε κενές θέσεις φύλλων του δέντρου του προτύπου προκειμένου να παραχθεί το t. Για την παραγωγή των προτύπων εισόδου και εξόδου χρησιμοποιήθηκε αλγόριθμος “αντί-ενοποίησης” (anti-unification), ο οποίος δεδομένων δύο δένδρων t_1 και t_2 παράγει το λιγότερο γενικό template (τ) για το οποίο υπάρχουν αντικαταστάσεις α_1 και α_2 έτσι ώστε $\alpha_1(\tau) = t_1$ και $\alpha_2(\tau) = t_2$.

Ο αλγόριθμος ομαδοποίησης (clustering) χρησιμοποιεί μία άπληστη προσέγγιση πολυπλοκότητας $O(n^2)$, ξεκινώντας με ένα άδειο σύνολο από ομάδες. Στη συνέχεια, για κάθε υποψήφια αλλαγή και για κάθε ομάδα ελέγχεται αν η προσθήκη της αλλαγής αυτής στο σύνολο διακριτών αλλαγών της εκάστοτε ομάδας δημιουργεί ένα πρότυπο. Όταν η συνθήκη αυτή ικανοποιείται, το σύστημα υπολογίζει το κόστος ένταξης της αλλαγής αυτής σε όλες τις υποψήφιες ομάδες και τελικά εντάσσει την αλλαγή στην ομάδα

με το μικρότερο κόστος. Σε περίπτωση που δεν υπάρχουν υποψήφιες ομάδες, δημιουργείται μία νέα ομάδα με την αλλαγή αυτή.

Το σύστημα εντόπισε 288.899 τροποποιήσεις κώδικα που συγκεντρώθηκαν σε 110.384 ομάδες. Από αυτές τις ομάδες, 39.104 περιέχουν περισσότερες από μία αλλαγές, δηλαδή θα μπορούσαν πολλαπλά παραδείγματα να γενικευτούν σε ένα ενιαίο μοτίβο επεξεργασίας. Αυτά τα πρότυπα επεξεργασίας μπορούν να καλύψουν το 71% του συνόλου των τροποποιήσεων κώδικα. Στα μειονεκτήματα του συστήματος προσμετράται η παραγωγή προτύπων σε μορφή AST, ενώ ιδανική κρίνεται η παραγωγή άμεσα εκτελέσιμών γρήγορων διορθώσεων στον κώδικα. Επιπλέον, σημειώνεται πως το σύστημα πραγματοποίησε την εκπαίδευση του αναλύοντας το ιστορικό μόνο απο 9 έργα λογισμικού.

3.3.5. Coming: Ένα Εργαλείο Εξόρυξης Προτύπων Αλλαγών από Git Commits

Οι ερευνητές Matias Martinez et al., το 2018, ολοκληρώνουν μια έρευνα με τίτλο "Coming: a Tool for Mining Change Pattern Instances from Git Commits"[67] με στόχο την υλοποίηση ενός εργαλείου που αναλύει αποθετήρια Git με δύο κύριους στόχους: τον υπολογισμό λεπτομερών αλλαγών κώδικα μεταξύ διαδοχικών revisions και την ανίχνευση προτύπων αλλαγών κώδικα. Τα χαρακτηριστικά του εργαλείου είναι τα εξής:

- Επισκέπτεται ένα σύνολο από revisions (δηλ. commits).
- Φιλτράρει εκείνα τα revisions που είναι ενδιαφέροντα (π.χ. bug fix commits), σύμφωνα για παράδειγμα με το περιεχόμενο του commit messsage.
- Υπολογίζει τις αλλαγές μεταξύ ενός revision και του προηγούμενου του.
- Συνοψίζει τα αποτελέσματα (π.χ., υπολογίζει την πιθανότητα κάθε τύπου αλλαγής).
- Καταγράφει τα commits που εισάγουν ένα σύνολο προτύπων αλλαγών κώδικα.

Με λίγα λόγια, το Coming λαμβάνει ως είσοδο μια λίστα αναθεωρήσεων. Για κάθε ζεύγος διαδοχικών αναθεωρήσεων, r και r', υπολογίζει πρώτα τις αλλαγές μεταξύ τους σε λεπτομερές επίπεδο, χρησιμοποιώντας έναν αλγόριθμό διαφοράς AST. Στη συνέχεια, αναλύει τις αλλαγές για να εντοπίσει εάν αντιστοιχούν σε ένα πρότυπο αλλαγών που δίνεται ως είσοδος. Ένα πρότυπο αλλαγών προσδιορίζει μια διενέργεια (π.χ. εισαγωγή, αφαίρεση) που πραγματοποιήθηκε σε οντότητες κώδικα (π.χ. επικλήσεις, αναθέσεις). Ένα ζεύγος αναθεωρήσεων περιλαμβάνει ένα παράδειγμα ενός προτύπου εάν:

- Υπάρχουν όλες οι αλλαγές που περιλαμβάνει το πρότυπο στη διαφορά μεταξύ των δυο αναθεωρήσεων του πηγαίου κώδικα.
- Οι οντότητες που επηρεάζονται από τις αλλαγές ταιριάζουν με αυτές του προτύπου.

Τέλος, μετά την ανάλυση όλων των commits, το Coming μετα-επεξεργάζεται τα αποτελέσματα από κάθε ζεύγος αναθεωρήσεων (δηλαδή τα πρότυπα που βρέθηκαν και τη συχνότητα εμφάνισης τους) και τα εξάγει σε μορφή JSON.

Το εργαλείο αυτό αναλύει commits από ένα αποθετήριο Git, η διεύθυνση του οποίου δίνεται ως παράμετρος εισόδου. Αρχικά, περιηγείται στο ιστορικό Git χρησιμοποιώντας τη βιβλιοθήκη Eclipse GIT[23]. Επιπλέον εξετάζει τα commits του αποθετηρίου ξεκινώντας από το παλαιότερο. Για κάθε commit C, το Coming παίρνει κάθε αρχείο f το οποίο τροποποιείται εσωτερικά του C και δημιουργεί ένα ζεύγος αναθεώρησης (revision pair) με κάθε προηγούμενη έκδοση του f. Στη συνέχεια, πραγματοποιεί λεπτομερείς συγκρίσεις βασισμένες στα ASTs κάθε ζεύγους αναθεώρησης. Η διαδικασία αυτή αποτελείται από δύο βασικά βήματα:

- AST αναπαράσταση αρχείων: Δημιουργεί ένα AST από τον κώδικα κάθε αναθεώρησης. Εσωτερικά ενός AST κάθε κόμβος του δέντρου αντιστοιχεί σε ένα στοιχείο του κώδικα (π.χ., μια επίκληση, μια παράμετρο, κτλ.).
- Σύγκριση Διαφοράς Δέντρων: Για την εξαγωγή της διαφοράς μεταξύ δύο μοντέλων m_s και m_t, που προέρχονται από δύο αναθεωρήσεις r_s και r_t, το Coming εφαρμόζει έναν αλγόριθμο διαφοράς δέντρου. Από προεπιλογή το Coming χρησιμοποιεί τον αλγόριθμο GumTree.

Στη συνέχεια, γίνεται χρήση ενός αναλυτή (analyzer) για τον εντοπισμό προκαθορισμένων προτύπων αλλαγών κώδικα (δίνονται σαν είσοδος στο σύστημα σε μορφή XML) μέσα σε μία λίστα αλλαγών AST (έξοδος του προηγούμενου βήματος). Κάθε πρότυπο έχει μία λίστα από ενέργειες (pattern actions), που καθορίζουν μία συγκεκριμένη αλλαγή AST μεταξύ δύο revisions. Ένα pattern action έχει δύο πεδία: τον τύπο της ενέργειας (insert, mode, remove, update) και μία αναφορά σε μία οντότητα κώδικα (PatternEntity). Κάθε PatternEntity με τη σειρά του έχει τρία πεδία: τον τύπο του στοιχείου κώδικα που αφορά (π.χ.if, invocation, return), την τιμή του στοιχείου αυτού (π.χ. callMethod1(), return null) και μία αναδρομική αναφορά σε κάποιο άλλο PatternEntity το οποίο θεωρείται γονέας (parent) του. Τέλος το εργαλείο επεξεργάζεται τα πρότυπα και εξάγει τα αποτελέσματα σε αρχεία JSON. Το σύστημα υποστηρίζει και μερικές επιπλέον λειτουργίες οι οποίες διαφοροποιούν την προκαθορισμένη (default). Παρέχει διαφορετικές προσεγγίσεις και λειτουργίες σε σχέση με: το αρχικό φιλτράρισμα των commits, τη λειτουργία του αναλυτή των διαφορών και τη μορφοποίηση εξόδου των αποτελεσμάτων.

[23] https://projects.eclipse.org/projects/technology.egit

Όσον αφορά τα αποτελέσματα του εργαλείου κατά την διαδικασία αξιολόγησης, υπήρχαν πάρα πολλές εμφανίσεις προτύπων οι οποίες αναφέρονται στην προσθήκη, ανανέωση, μετακίνηση αλλά και διαγραφή εντολών ελέγχου συνθηκών (If Statemets).

3.4. Recommendation Systems for Software Engineering

3.4.1. Εισαγωγή στα Συστήματα Προτάσεων

Στη σημερινή εποχή ο ολοένα και αυξανόμενος αριθμός και όγκος πληροφοριών που διοχετεύεται στους διαδικτυακούς τόπους απαιτεί την ανασυγκρότηση της προσβασιμότητας της πληροφορίας, με στόχο την εξυπηρέτηση του χρήστη για τη λήψη βέλτιστων αποφάσεων. Σε αυτό το πλαίσιο, τα συστήματα προτάσεων με εφαρμογή στον κλάδο της τεχνολογίας λογισμικού, εξυπηρετούν αυτόν τον σκοπό και επιτρέπουν στους προγραμματιστές να δημιουργήσουν τα κατάλληλα αρχεία πηγαίου κώδικα. Επιπλέον τα συστήματα αυτά είναι ικανά να συνδέουν τα αποτελέσματά τους με τα εξατομικευμένα ενδιαφέροντα του εκάστοτε χρήστη.

Συγκεκριμένα, τα συστήματα προτάσεων έχουν τη δυνατότητα να προσφέρουν στους χρήστες χρήσιμες λύσεις με βάση τα αντικείμενα που τους ενδιαφέρουν και τα οποία έχουν εκφραστεί είτε άμεσα (μέσω ερωτημάτων) είτε έμμεσα (μέσω παράπλευρων ενεργειών που εκτελούν) σε παρελθοντικό χρόνο, αποφεύγοντας με αυτόν τον τρόπο τη διοχέτευση μη χρήσιμων πληροφοριών στη βάση δεδομένων. Οι προγραμματιστές μέσω των συστημάτων προτάσεων (RSSEs) μπορούν να βρουν τον σωστό κώδικα και το κατάλληλο API (application program interface) συνδυάζοντας πληροφορίες από βάσεις δεδομένων του συστήματος, βιβλιοθήκες λογισμικού, αναφορές σφαλμάτων (bug reports), ιστορικό εκδόσεων (version history) και λοιπή τεκμηρίωση (documentation).

Η ανάκτηση των προτιμήσεων του χρήστη, τόσο άμεσα μέσω ερωτημάτων που θέτει ο προγραμματιστής όσο και έμμεσα μέσω της εκτέλεσης παράπλευρων ενεργειών, εγείρει την εξής πρόκληση: τη δημιουργία ενός *πεδίου* στο οποίο θα λαμβάνονται υπόψιν όλες οι σχετικές πληροφορίες για τον χρήστη, για την εργασία του, το στάδιο του έργου ή της εργασίας σε πραγματικό χρόνο πρόβλεψης/προτάσεων.

Έτσι ένα RSSE πρέπει να συμπεριλάβει ένα ευρύτερο φάσμα πληροφοριών για τον προσδιορισμό του πεδίου αναζήτησης. Πέρα από τις πληροφορίες που αφορούν τα χαρακτηριστικά του χρήστη, όπως για παράδειγμα η περιγραφή της θέσης εργασίας, το επίπεδο εξειδίκευσης, προηγούμενη εργασιακή εμπειρία και γενικότερα το κοινωνικό του δίκτυο, οι παρελθοντικές ενέργειες του χρήστη (όπως αντικείμενα που είδε ή του προτάθηκαν προγενέστερα μέσω των αναζητήσεών του), είναι ιδιαίτερα

χρήσιμες για τον προσδιορισμό του πεδίου αναζήτησης. Επιπλέον, το είδος της εργασίας που πρέπει να ολοκληρωθεί, όπως η προσθήκη νέων χαρακτηριστικών ή η βελτιστοποίηση, και τα ειδικά χαρακτηριστικά της εργασίας (π.χ. επεξεργασμένος κώδικας), συνεισφέρουν σημαντικά στην ολοκληρωμένη δημιουργία του πεδίου αναζήτησης.

Προκειμένου να λειτουργήσει ένα σύστημα προτάσεων RSSE υπάρχουν ορισμένα διακριτά βήματα και στάδια που πρέπει να τεθούν σε εφαρμογή. Ωστόσο, λόγω της μεγάλης ποικιλομορφίας των λειτουργιών που υπάρχουν στον τομέα της τεχνολογίας λογισμικού και των διαφορετικών RSSEs που υιοθετούν διαφορετικές αναπαραστάσεις δεδομένων, δεν είναι δυνατόν να οριστεί μια καθολική και ολοκληρωμένη αρχιτεκτονική που να περιλαμβάνει όλα αυτά τα διαφορετικά συστήματα που έχουν προταθεί. Σε γενικές γραμμές, η λειτουργικότητα των συστημάτων RSSEs μπορεί να αποτυπωθεί μέσα από ορισμένα στάδια που περιλαμβάνουν συγκεκριμένα βήματα. Το πρώτο καίριο στάδιο για τη λειτουργία ενός συστήματος RSSE, είναι η συγκέντρωση και αποθήκευση δεδομένων από διάφορες πηγές. Ανάλογα με τον τύπο RSSE τα δεδομένα συγκεντρώνονται είτε από αποθετήρια λογισμικού, είτε από κοινότητες ερωταπαντήσεων, είτε από μηχανές αναζήτησης και αποθηκεύονται σε στατικές αποθήκες λογισμικού.

Το επόμενο στάδιο αφορά την προεπεξεργασία των δεδομένων που έχουν συλλεχθεί και αποθηκευτεί. Τα ακατέργαστα αποθηκευμένα δεδομένα πρέπει να μετασχηματιστούν ώστε να γίνουν όσο το δυνατόν περισσότερο ερμηνεύσιμα. Για παράδειγμα, τα δεδομένα μπορούν να μετασχηματιστούν είτε μέσω της διαχείρισης των ελλειπουσών τιμών (missing values), είτε με την ανίχνευση των ακραίων τιμών (outlier detection). Το επόμενο στάδιο μετά από την επιτυχή ολοκλήρωση της προεπεξεργασίας δεδομένων αφορά την εφαρμογή των κατάλληλων αλγορίθμων για την εξόρυξη της χρήσιμης πληροφορίας. Η εξόρυξη της χρήσιμης πληροφορίας είναι ίσως το κυριότερο βήμα για την τελική επιτυχία ενός συστήματος προτάσεων, καθώς μέσω αυτού του σταδίου επιτυγχάνεται η μεγιστοποίηση της εξαγόμενης σχετικής πληροφορίας στον εκάστοτε χρήστη που είναι άλλωστε και ο πυρήνας της φιλοσοφίας της εφαρμογής των συστημάτων προτάσεων. Στη συνέχεια αναφέρονται με περισσότερες λεπτομέρειες οι προϋποθέσεις για την επιτυχή ολοκλήρωση του βήματος της εξόρυξης της χρήσιμης πληροφορίας.

Πιο αναλυτικά, η εξόρυξη της χρήσιμης πληροφορίας επιτυγχάνεται μέσα από την εφαρμογή λογικών αναλύσεων και αλγορίθμων. Γενικότερα για τη σωστή εφαρμογή ενός συστήματος RSSE αλλά και ειδικότερα για την επιτυχή ολοκλήρωση του βήματος της εξόρυξης της χρήσιμης πληροφορίας, η επιλογή του κώδικα που θα χρησιμοποιηθεί είναι καίριας σημασίας. Ο κώδικας θα πρέπει να είναι τόσο

ερμηνεύσιμος όσο και αναζητήσιμος. Για να είναι ένας κώδικας ερμηνεύσιμος και αναζητήσιμος, είναι απαραίτητη η χρήση μίας κατάλληλης αναπαράστασης (representation) του κώδικα. Ορισμένες από τις πιο διαδεδομένες αναπαραστάσεις (representation) που χρησιμοποιούνται για την πρακτική εφαρμογή διάφορων RSSEs είναι:

- Η *Αναπαράσταση Συνόλου Αντικειμένων* (Bag of words representation), η οποία είναι μία διανυσματική αναπαράσταση, όπου κάθε διάσταση αναπαριστά τη συχνότητα ενός αντικειμένου.
- Οι *Ακολουθίες* (Sequences) στις οποίες αποτυπώνονται η σειρά με την οποία εκτελέστηκαν οι εντολές και επομένως έχει μία ακολουθιακή δομή.
- Τα Αφηρημένα Συντακτικά Δέντρα (Abstract Syntax Trees).
- Τα Αναλυτικά Δέντρα (ParseTrees).
- Οι *Κατευθυνόμενοι Άκυκλοι Γράφοι* (Directed Acyclic Graphs) αποτελούν μία γενίκευση της δεντρικής δομής όπου οι διαφορετικοί τύποι αντικειμένων αποτυπώνονται σε κόμβους και η εφαρμογή συναρτήσεων πάνω στα δεδομένα αποτυπώνεται σε ακμές.

Επομένως, από τη στιγμή που ένας κώδικας έχει την κατάλληλη αναπαράσταση είναι εφικτή η εφαρμογή των αλγορίθμων και των απαραίτητων τεχνικών εξόρυξης δεδομένων. Έτσι μέσα από τις παραπάνω διαδικασίες γίνεται η κατηγοριοποίηση των αποτελεσμάτων στα πλαίσια κάποιας συνάρτησης αξιολόγησης ανάλογα με τη σχετικότητα των τμημάτων του κώδικα με το ερώτημα του χρήστη, ώστε να είναι εφικτή η μετεπεξεργασία των δεδομένων. Το στάδιο της μετεπεξεργασίας δεδομένων στοχεύει στη βελτίωση των διαδικασιών που έχουν ήδη διεξαχθεί. Σε αυτό το βήμα, ορίζονται περαιτέρω διαδικασίες αξιολόγησης (π.χ μεταγλωττισιμότητα του κώδικα, ευρεστικές τεχνικές επεξεργασίας, ποιοτικός έλεγχος κ.α.) για τα αποτελέσματα που παράγονται και την ταξινόμηση τους και αφού οριστούν οι κατάλληλοι μετασχηματισμοί για τα αποτελέσματα, τότε είναι εφικτή και η παρουσίαση τους στον εκάστοτε χρήστη.

Η παρουσίαση των αποτελεσμάτων συνιστά το τελικό στάδιο της λειτουργίας ενός συστήματος RSSE. Στην πιο απλοποιημένη μορφή τα αποτελέσματα μπορούν να παρουσιαστούν στους χρήστες με μία μορφή λίστας. Σε αυτή τη λίστα τα αποτελέσματα ταξινομούνται σύμφωνα με την αντιστοιχία που έχουν με τις σχετικές αναζητήσεις του χρήστη. Σε πιο σύνθετη μορφή, τα αποτελέσματα παρουσιάζονται μέσω της αλληλεπίδρασης του χρήστη με τη διεπαφή του συστήματος. Η διεπαφή (interface) αλληλεπίδρασης του χρήστη με το σύστημα, μπορεί να επιτευχθεί μέσα από διάφορους τρόπους που απαντώνται συχνά σε πραγματικά συστήματα προτάσεων τεχνολογίας λογισμικού. Για παράδειγμα, μέσω της

επαναχρησιμοποίησης τμημάτων κώδικα (Component Reuse) ο χρήστης μπορεί να δώσει σαν είσοδο κάποιο μήνυμα ή ημιτελή κομμάτια κώδικα προκειμένου να του παρουσιαστούν τα αποτελέσματα. Αυτός ο τρόπος ενδείκνυται για χρήστες που δε γνωρίζουν τι κώδικα να χρησιμοποιήσουν προκειμένου να επιτελέσουν τις κατάλληλες λειτουργίες.

Ένας ακόμη τρόπος προκειμένου να επιτευχθεί η αλληλεπίδραση του χρήστη με το σύστημα αποτελεί η *αυτόματη συμπλήρωση κώδικα* (Auto Completion), όπου ο χρήστης δίνει σαν είσοδο ημιτελή τμήματα ενός κώδικα και χρησιμοποιεί το RSSE για να συμπληρώσει τα κενά προκειμένου να συντάξει τον κώδικά του. Σε περιπτώσεις όπου ο χρήστης επιθυμεί να πραγματοποιήσει έλεγχο για την εύρεση καλύτερης λύσης σχετικής με το πρόβλημα που αντιμετωπίζει ή να ελέγξει την ύπαρξη παρόμοιων υλοποιήσεων, δίνει σαν είσοδο στο σύστημα ένα ολοκληρωμένο τμήμα κώδικα. Ο τρόπος αυτός συνιστά τον επονομαζόμενο *Σχετικό Κώδικα* (Relevant Code). Επιπλέον, σε περιπτώσεις που επιδιώκεται η λήψη προτάσεων σχετικής τεκμηρίωσης ή κατάλληλων μηνυμάτων και σχολίων, γίνεται χρήση της μεθόδου της *Τεκμηρίωσης* (Documentation) κατά την οποία ο χρήστης δίνει σαν είσοδο κάποιο κομμάτι κώδικα που έχει ήδη υλοποιήσει. Τέλος, μέσω της χρήσης διεπαφών προγραμματισμού εφαρμογών (API Discovery), είναι δυνατή η επιστροφή τμημάτων κώδικα που λύνουν συγκεκριμένα προβλήματα με χρήση του API του ενδιαφέροντος του χρήστη. Ο τρόπος αυτός είναι ιδιαίτερα χρήσιμος για χρήστες που δε γνωρίζουν πως να χρησιμοποιήσουν ένα συγκεκριμένο API.

Μέχρι σήμερα έχουν αναπτυχθεί διάφορα σχετικά συστήματα προτάσεων όπως το RSSE που αναλύθηκε προηγουμένως. Η βιβλιογραφική επισκόπηση παρόμοιων συστημάτων, στοχεύει στην καλύτερη και βαθύτερη κατανόηση του θέματος που πραγματεύεται η παρούσα εργασία. Στη συνέχεια, θα αναλυθούν συνοπτικά κάποια από τα σημαντικότερα και πιο πρόσφατα συστήματα προτάσεων, η βασική λειτουργία τους καθώς και ο τρόπος εκτέλεσης τους από κάποιον χρήστη.

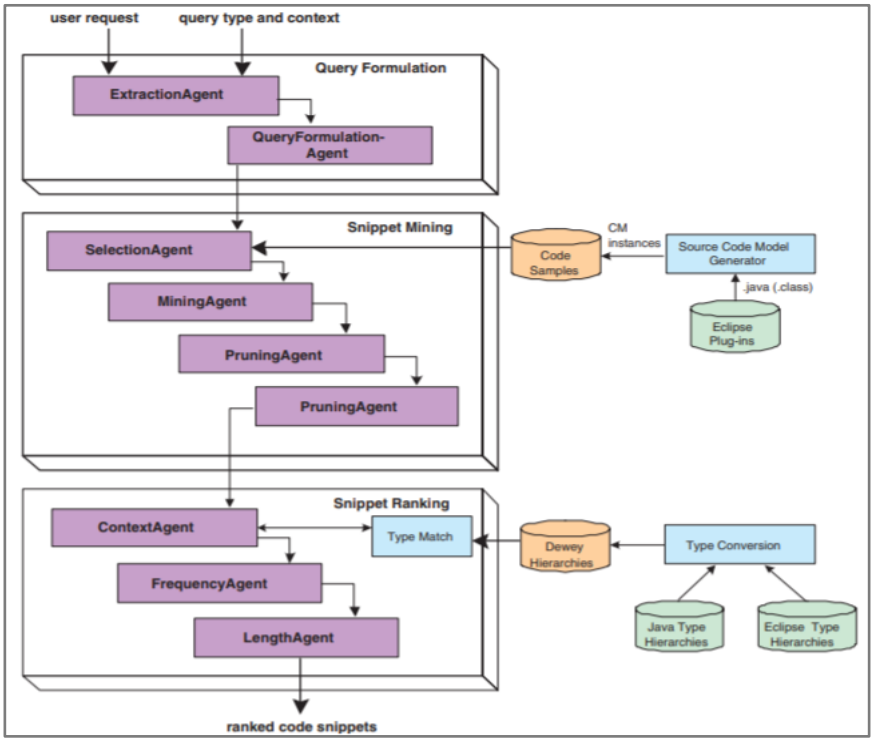
3.4.2. Υπάρχοντα Συστήματα Προτάσεων

3.4.2.1. Xsnippet

Το 2006, δημιουργήθηκε ένα από τα πρώτα σύστημα προτάσεων πηγαίου κώδικα επονομαζόμενο Xsnippet[69], μέσω του οποίου είναι δυνατή η αποστολή ερωτημάτων σε αποθετήρια λογισμικού και η λήψη αποσπασμάτων κώδικα που σχετίζονται με το έργο τους. Η αρχιτεκτονική του συστήματος παρουσιάζεται στην Εικόνα 3.4 και η λειτουργία του έγκειται στην ολοκλήρωση τριών βημάτων. Σε πρώτο στάδιο τίθεται ένα ερώτημα στο σύστημα, το οποίο μπορεί να είναι τόσο γενικό όσο και ειδικό και

παράγονται τα αντίστοιχα *snippets* τα οποία μπορεί να είναι ποσοτικά πολλά (σε γενικό ερώτημα) αλλά και σχετικά και συγκεκριμένα (σε ειδικό ερώτημα).

Στη συνέχεια, υλοποιείται η εξόρυξη τμημάτων κώδικα από τον αλγόριθμο BFSMINE[70] (Breadth-First Search Mining), του οποίου τα αποτελέσματα φιλτράρονται πριν το τελευταίο βήμα στο οποίο ουσιαστικά υφίσταται η αξιολόγηση των τμημάτων κώδικα που φιλτραρίστηκαν μέσα από τη χρήση *ευρετικών τεχνικών* (heuristics). Η αξιολόγηση μπορεί να ταξινομηθεί σε ορισμένες κατηγορίες όπως το περιεχόμενο, η συχνότητα εμφάνισης και το μέγεθος του κώδικα.

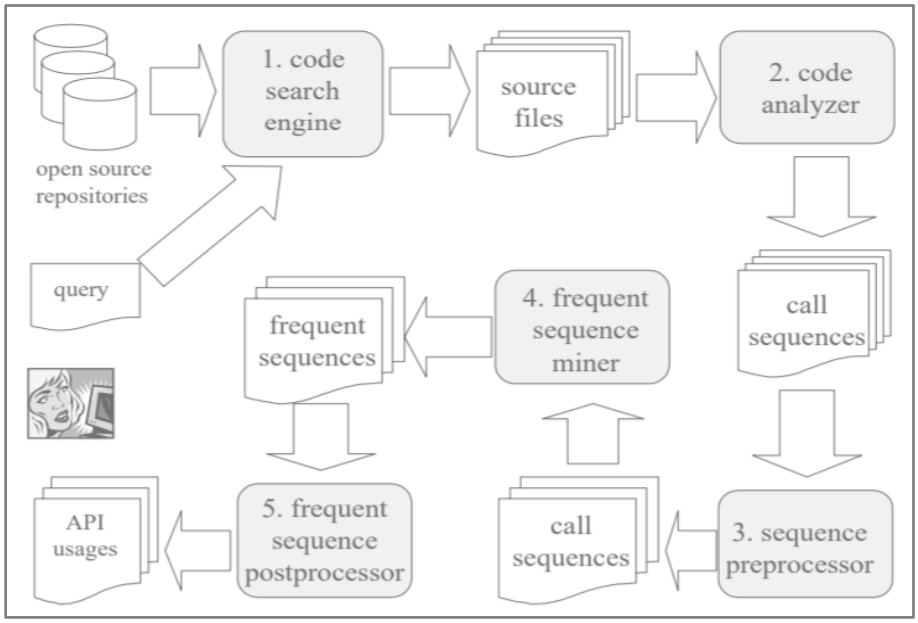


Εικόνα 3.4 Αρχιτεκτονική του Xsnippet.

3.4.2.2. MAPO & PARSEWeb

Το 2006 Οι Xiao και Pei δημιουργήσαν ένα σύστημα προτάσεων γνωστό ως MAPO[71]. Λίγα χρόνια αργότερα οι ίδιοι κατασκεύασαν ένα νέο σύστημα προτάσεων το PARSEWeb[72] το οποίο κύριο στόχο είχε να ξεπεραστεί η αδυναμία του MAPO, που περιγράφεται παρακάτω, και γενικότερα να λύσει το ζήτημα της ημιτελούς γνώσης ενός κώδικα κατά την παραγωγή του από τους προγραμματιστές. Συγκεκριμένα, σε ορισμένες περιπτώσεις ένας προγραμματιστής μπορεί να έχει γνώση του αντικειμένου που θέλει να υλοποιήσει αλλά να μη γνωρίζει πως να παράγει τον συγκεκριμένο κώδικα για να έχει το επιθυμητό αποτέλεσμα. Αρχικά, θα αναφερθεί ο τρόπος λειτουργίας του MAPO και στη συνέχεια ο τρόπο λειτουργίας του PARSEWeb το οποίο κατασκευάστηκε σε μεταγενέστερο στάδιο.

Ο κύριος στόχος που συνετέλεσε στη δημιουργία του MAPO ήταν η διευκόλυνση των προγραμματιστών στην κατανόηση της χρήσης ενός API ώστε να μπορούν να συντάσσουν καλύτερους κώδικες αξιοποιώντας τις ήδη υπάρχουσες μηχανές αναζήτησης. Δομικά, αποτελείται από πέντε βασικά τμήματα. Αναλυτικότερα, αποτελείται από μία *Μηχανή Αναζήτησης Κώδικα* (code search engine), από έναν *Αναλυτή Πηγαίου Κώδικα* (Source Code Analyzer), όπου ο κώδικας μοντελοποιείται σε μία ακολουθιακή αναπαράσταση και δεν απαιτείται τα αρχεία του κώδικα να είναι μεταγλωττισμένα, από έναν *Προεπεξεργαστή Ακολουθιών* (Sequence Preprocessor), από έναν *Εξορυκτή Συχνών Ακολουθιών* (Frequent Sequence Miner), ο οποίος έχει τη δυνατότητα εξόρυξης πληροφοριών χρησιμοποιώντας τη συχνότητα εμφάνισης των ακολουθιών και τέλος έναν *Μετεπεξαργαστή Συχνών Ακολουθιών* (Frequent Sequence Postprocessor) ο οποίος χρησιμεύει στη μείωση του αριθμού των αποτελεσμάτων διατηρώντας παράλληλα τα χρήσιμα δεδομένα. Στην Εικόνα 3.5 εμφανίζεται η αρχιτεκτονική του συστήματος.

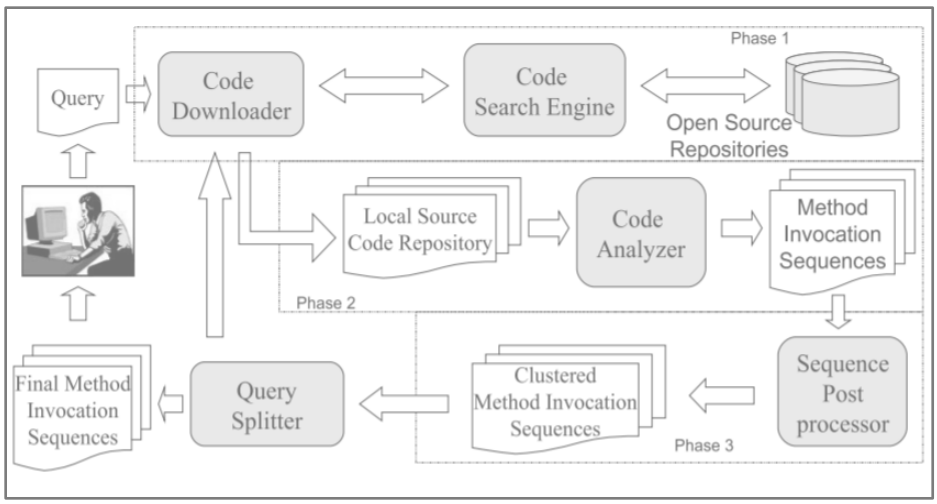


Εικόνα 3.5 Αρχιτεκτονική MAPO.

Πέρα από τα πλεονεκτήματα που παρουσιάζει η αρχιτεκτονική και η λειτουργία του MAPO υπάρχουν και ορισμένα μειονεκτήματα όπως το γεγονός ότι οι προγραμματιστές αδυνατούν να χειριστούν αυθαίρετα τμήματα κώδικα. Στα πλαίσια του MAPO, πρέπει να έχουν γνώση του API που θέλουν να χρησιμοποιήσουν και δεν υπάρχει η πληροφορία ροής ελέγχου (control-flow) που δημιουργήθηκε σε μεταγενέστερα συστήματα. Έτσι λοιπόν το σύστημα προτάσεων PARSEWeb έρχεται να αντιμετωπίσει ορισμένα από τα προβλήματα του MAPO.

Η λειτουργία του PARSEWeb βασίζεται στην αλληλεπίδραση με μία μηχανή αναζήτησης κώδικα, με στόχο την αναζήτηση δειγμάτων κώδικα σχετικά με τις χρήσεις των συγκεκριμένων τύπων αντικειμένων προέλευσης και προορισμού. Τα αποτελέσματα παραδειγμάτων κώδικα που παράγονται από τη μηχανή αναζήτησης ως αποτελέσμα, σχηματίζουν ουσιαστικά μία αποθήκη πηγαίου κώδικα και με την ανάλυση αυτής της αποθήκης εξάγονται σχετικές προτάσεις με τη χρήση ενός μετεπεξεργαστή ακολουθιών, οι οποίες αποτελούν τη λύση για το εκάστοτε ερώτημα που έχει τεθεί από τον προγραμματιστή. Είναι σημαντικό να αναφερθεί ότι στο συγκεκριμένο σύστημα προτάσεων τα αποτελέσματα ταξινομούνται με τη χρήση πολλών ευρετικών τεχνικών (heuristics). Επιπλέον, μέσω του PARSEWeb αντιμετωπίζεται το πρόβλημα διαχωρισμένων τμημάτων κώδικα που βρίσκονται σε διαφορετικά αρχεία πηγαίου κώδικα.

Σε σχέση με τα δομικά χαρακτηριστικά του PARSEWeb υπάρχουν πέντε βασικά τμήματα που το στελεχώνουν (όπως και το MAPO). Το πρώτο είναι η *Μηχανή Αναζήτησης Κώδικα* (Code Search Engine) όπου συλλέγονται τα δείγματα του κώδικα. Το δεύτερο είναι η *Μηχανή Λήψης Κώδικα* (Code Downloader), το οποίο δέχεται ένα ερώτημα από τον προγραμματιστή και είναι υπεύθυνο για τη δημιουργία μίας αποθήκης πηγαίου κώδικα. Το τρίτο τμήμα περιλαμβάνει τον *Αναλυτή Κώδικα* (Code Analyzer) ο οποίος αναλύει τα δείγματα του κώδικα που είναι αποθηκευμένα στην αποθήκη πηγαίου κώδικα και παράγει διάφορα MIS (Method Invocation Sequence). Με τον όρο MIS αναφερόμαστε σε συχνά χρησιμοποιούμενες ακολουθίες κλήσης μεθόδων που μπορούν να μετατρέψουν ένα αντικείμενο προέλευσης (Source) σε ένα αντικείμενο προορισμού (Destination). Ουσιαστικά τα αντικείμενα προέλευσης και προορισμού αποτελούν τα ερωτήματα και τα αποτελέσματα του συστήματος, αντίστοιχα. Το τέταρτο είναι ο *Μετεπεξεργαστής Ακολουθιών* (Sequence Postprocessor) ο οποίος συγκεντρώνει όλα τα παρόμοια MIS και δημιουργεί μία κατάταξη κατόπιν αξιολόγησης. Αν παραχθεί μία και μοναδική λύση που μπορεί να αποτελέσει χρήσιμο δεδομένο για το ερώτημα, τότε ο *Διαχωριστής Ερωτήματος*, ο οποίος αποτελεί και το τελευταίο δομικό χαρακτηριστικό, δίνει την εξαγωγή του αποτελέσματος. Σε διαφορετική περίπτωση, ο Διαχωριστής Ερωτήματος ξεχωρίζει το ερώτημα που έχει τεθεί από τον προγραμματιστή σε υποερωτήματα και η διαδικασία επαναλαμβάνεται από την αρχή μέχρι να συλλεχθούν τα αποτελέσματα, να μαζευτούν και να τα εξάγει σε ένα συνολικό αποτέλεσμα. Στην Εικόνα 3.6 παρουσιάζεται η αρχιτεκτονική του συστήματος.



Εικόνα 3.6 Αρχιτεκτονική PARSEWeb.

3.4.2.3. Blueprint

Το σύστημα προτάσεων Blueprint[73] το οποίο δημιουργήθηκε το 2010, επιτρέπει την πρόσβαση σε παραδείγματα κώδικα που βρίσκονται στο διαδίκτυο μέσα από το περιβάλλον ανάπτυξης (IDE) του χρήστη. Μέσω της ενσωμάτωσης αναζήτησης στο περιβάλλον ανάπτυξης (IDE), δίνεται η δυνατότητα της αξιοποίησης του περιβάλλοντος των χρηστών (μειώνοντας με αυτό τον τρόπο το κατασκευαστικό κόστος που απαιτείται για τη δημιουργία ενός καλού ερωτήματος) και της παραγωγής βέλτιστων και ποιοτικών αποτελεσμάτων. Πρόκειται για ένα σύστημα προτάσεων το οποίο υλοποιείται υπό την αρχιτεκτονική client-server. Το συγκεκριμένο σύστημα περιλαμβάνει ένα plug-in (client) το οποίο παρέχει ουσιαστικά τη διεπαφή χρήστη για αναζήτηση και περιήγηση των αποτελεσμάτων και ο server εκτελεί αναζητήσεις κώδικα. Η λειτουργία του έγκειται στο ότι παρέχει μία διεπαφή χρήστη για την παραγωγή ερωτημάτων και την εμφάνιση αποτελεσμάτων, στο ότι είναι ικανό να στέλνει πληροφορίες σχετικά με το περιβάλλον ανάπτυξης στον server και στο ότι ειδοποιεί τον χρήστη σε περίπτωση που αλλάξει η προέλευση παραδειγμάτων κώδικα που έχει χρησιμοποιήσει στο διαδίκτυο.

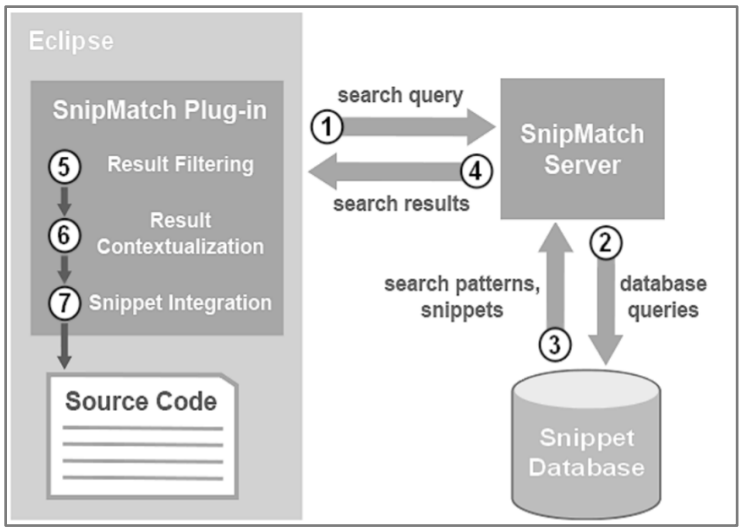
Αναλυτικότερα, ο χρήστης σε πρώτο στάδιο θέτει κάποιο ερώτημα στον server, ο οποίος εμπλουτίζει την ερώτηση του χρήστη ανάλογα με το περιβάλλον του και στη συνέχεια αποστέλλει το τελικό ερώτημα στη μηχανή αναζήτησης, η οποία επιστρέφει ένα σύνολο από ιστοσελίδες σε HTML. Στη συνέχεια, ο server ανακτά τις ιστοσελίδες και τις επεξεργάζεται για να εξάγει παραδείγματα πηγαίου κώδικα, τμηματοποιώντας την εκάστοτε σελίδα προκειμένου να ταξινομήσει κάθε τμήμα ως πηγαίο κώδικα ή διαφορετικό περιεχόμενο. Τα τελικά αποτελέσματα του κώδικα αξιολογούνται είτε με ευρετικές τεχνικές

(heuristics) είτε με αναλυτές κειμένου (language parsers). Η λειτουργία του συγκεκριμένου συστήματος είναι αρκετά χρονοβόρα και ενδείκνυται για τη χρήση μικρών κομματιών κώδικα.

3.4.2.4. Snipmatch

Οι D. Wightman και J. Brandt πρότειναν ένα ακόμη κομβικό σύστημα προτάσεων κώδικα, ονομαζόμενο Snipmatch[74]. Το Snipmatch σχεδιάστηκε το 2012, με απώτερο σκοπό να αντιμετωπίσει τις αδυναμίες προηγούμενων προσεγγίσεων, όπως αυτές που έχουν σχεδιαστεί για την εκμετάλλευση ενός προσεκτικά επιμελημένου συνόλου τμημάτων κώδικα και αυτές, που είτε συνθέτουν, είτε εξορύσσουν (mine) σχετικά snippets από μεγάλα αποθετήρια λογισμικού. Σε αυτό το πλαίσιο το Snipmatch χρησιμοποίησε μία καινοτόμα βασική σχεδιαστική ιδέα: τη χρήση μεταδεδομένων μέσω της οποίας επιτυγχάνεται η δημιουργία πολύ πιο ισχυρών εργαλείων για την ανάκτηση και προσαρμογή των snippets. Συγκεκριμένα, μαζί με ένα snippet συμπεριλαμβάνονται και πληροφορίες, όπως τα ονόματα των τοπικών μεταβλητών καθώς και το τοπικό περιβάλλον κώδικα (π.χ. τύπους μεταβλητών, εισαγόμενες βιβλιοθήκες, θέση του δρομέα (cursor) στο AST, εξαρτήσεις κώδικα), ώστε τα snippets να προσαρμόζονται αυτόματα και να εξασφαλίζεται η συμβατότητα με τον υπάρχοντα κώδικα. Στη συνέχεια περιγράφονται τα βασικά επτά βήματα για τη λειτουργία του συγκεκριμένου συστήματος προτάσεων.

Όπως φαίνεται στην Εικόνα 3.7, στο πρώτο βήμα ο χρήστης σχηματίζει ένα ερώτημα αναζήτησης στο σχετικό πεδίο και εμφανίζεται το αποτέλεσμα. Ο κατάλογος των αποτελεσμάτων αναζήτησης ενημερώνεται καθώς τροποποιούνται τα περιεχόμενα του πεδίου αναζήτησης. Αυτό παρέχει στο χρήστη άμεση ανατροφοδότηση, διευκολύνοντας την εύρεση αποσπάσματος και τη βαθμιαία βελτίωση των ερωτημάτων αναζήτησης.



Εικόνα 3.7 Διαδικασία Εύρεσης ενός Snippet μέσω του SnipMatch.

Έπειτα, ο server μεταφέρει το ερώτημα στη βάση και αιτείται επιστροφή σχετικών αποτελεσμάτων. Υπάρχουν τρεις τύποι αποτελεσμάτων αναζήτησης οι οποίοι αξιολογούνται με διαφορετικά κριτήρια.

- In-order: Είναι snippets στα οποία η ακολουθία αναζήτησης απαντάται αυτούσια στο αναζητήσιμο πεδίο τους (βήμα 2).
- Unordered: Snippets που περιέχουν έναν ή περισσότερους όρους του ερωτήματος αναζήτησης στο αναζητήσιμο πεδίο τους (βήμα 3).
- Unsigned: Snippets που δεν έχουν ακόμη αναζητήσιμο πεδίο, αλλά των οποίων ο κώδικας περιέχει έναν ή περισσότερους όρους του ερωτήματος αναζήτησης (βήμα 4).

Στη συνέχεια, ο client τροποποιεί τα αποτελέσματα αναζήτησης που επιστρέφονται πριν εμφανιστούν στον χρήστη. Η διαδικασία αυτή γίνεται σε δύο στάδια: το φιλτράρισμα των αποτελεσμάτων (Result Filtering) και την τοποθέτηση εννοιολογικού πλαισίου (Result Contextualization). Στη φάση του φιλτραρίσματος (βήμα 5) γίνεται ένας έλεγχος μεταγλώττισης καθώς και συμβατότητας με τον προϋπάρχοντα κώδικα του χρήστη. Επιπλέον, ελέγχεται εάν τηρούνται οι προϋποθέσεις ενσωμάτωσης με βάση τη markup γλώσσα. Στη φάση της τοποθέτησης εννοιολογικού πλαισίου (βήμα 6), το σύστημα μετονομάζει τις μεταβλητές του snippet σύμφωνα με τον κώδικα του χρήστη. Τέλος, γίνεται αυτόματη ενσωμάτωση του επιλεγμένου snippet στον κώδικα του χρήστη με τη βοήθεια της markup [24]γλώσσας καθώς και ορισμένων βοηθητικών κλάσεων (helper classes) (βήμα 7).

3.4.2.5. Bing Code Search

Το Bing Code Search[75] συνιστά ένα από τα πιο πρόσφατα συστήματα προτάσεων το οποίο παρουσιάστηκε για πρώτη φορά από τη Microsoft το 2015. Μέσω του συγκεκριμένου συστήματος οι προγραμματιστές έχουν τη δυνατότητα να γράψουν τις ερωτήσεις που επιθυμούν σε φυσική γλώσσα και να πάρουν τα ανάλογα αποσπάσματα κώδικα σε γλώσσα C# , τα οποία δίνουν τις κατάλληλες απαντήσεις στις ερωτήσεις που έχουν διατυπωθεί προηγουμένως σε φυσική γλώσσα. Το BCS υλοποιείται σε μια αρχιτεκτονική client-server. Ο client είναι ένα plug-in του Visual Studio και είναι υπεύθυνος να στέλνει ερωτήματα χρήστη στον server, να λαμβάνει προτάσεις αποσπασμάτων κώδικα, να καταγράφει τις ενέργειες του χρήστη και να αποστέλλει αυτές τις ενέργειες στον server ως ανατροφοδότηση (feedback). Συγκεκριμένα, η λειτουργία του BCS έγκειται στο ότι μπορεί να προτείνει κατάλληλα και προσαρμοσμένα αποσπάσματα (snippets) πριν την τελική παρουσίασή τους στον χρήστη. Η λειτουργία του συγκεκριμένου συστήματος μπορεί να κατανοηθεί μέσα από τα τέσσερα ακόλουθα στάδια.

[24] https://www.britannica.com/technology/markup-language

Σε πρώτο στάδιο επιτυγχάνεται η αναζήτηση μέσω του διαδικτύου για υποψήφια αποσπάσματα ώστε να παραχθούν τα κατάλληλα αποτελέσματα. Με βάση το ερώτημα που θα θέσει ο χρήστης, το BCS προσθέτει στο ερώτημα τη λέξη c# και στη συνέχεια μέσω του συστήματος Bing search engine ανακύπτουν οι σχετικές με το ερώτημα ιστοσελίδες (οι 20 καλύτερες και σχετικότερες). Στη συνέχεια μέσω του Microsoft Roslyn C# parser ο κώδικας διαχωρίζεται από το κείμενο και ο parser παράγει AST για τα τμήματα κώδικα, τα οποία χρησιμοποιούνται για να αναγνωριστούν κλάσεις, μέθοδοι και μεταβλητές.

Εν συνεχεία γίνεται η αξιολόγηση και η κατάταξη των αποτελεσμάτων μέσω της χρήσης χαρακτηριστικών του κώδικα. Για να βελτιωθεί η ποιότητα της κατάταξης των αποσπασμάτων, γίνεται χρήση μίας αξιολόγησης βασισμένης στη μηχανική μάθηση (machine learning) με βάση ορισμένα κριτήρια όπως η σχετικότητα του αποσπάσματος του κώδικα στο ερώτημα του χρήστη και η ποιότητα του αποσπάσματος του κώδικα. Είναι σημαντικό να αναφερθεί ότι οι μηχανές αναζήτησης έχουν τη δυνατότητα να ενσωματώνουν ήδη εξελιγμένες τεχνικές κατάταξης, οι οποίες λαμβάνονται υπόψη και βελτιώνονται επιπλέον κριτήρια όπου είναι αναγκαίο.

Επιπλέον, υπάρχει προσαρμογή των αποσπασμάτων στο περιβάλλον προγραμματισμού του χρήστη με κατάλληλη μετονομασία των μεταβλητών η οποία γίνεται αυτοματοποιημένα. Συγκεκριμένα, οι μεταβλητές που θέτει ο χρήστης στο ερώτημα αντιστοιχούνται αυτόματα σε μεταβλητές που υπάρχουν στον κώδικα που χρησιμοποιεί ο χρήστης. Είναι σημαντικό να επισημανθεί ότι υπάρχει μόνο μία σωστή αντιστοίχιση με τις υπόλοιπες να είναι λανθασμένες, γεγονός που οδηγεί σε πρόβλημα δυαδικής ταξινόμησης με δύο επίπεδα (Σωστό/Λάθος). Το συγκεκριμένο ζήτημα είναι δυνατόν να επιλυθεί με τη χρήση Support Vector Machines (SVMs). Ωστόσο η κατασκευή ενός αξιολογητή (ranker) για την ταξινόμηση όλων των δυνατών αντιστοιχίσεων θα αποτελούσε την ιδανική περίπτωση.

Τέλος, μετά την επιτυχή ολοκλήρωση των προηγούμενων βημάτων τα αποσπάσματα είναι έτοιμα να παρουσιαστούν στον χρήστη. Πριν την παρουσίαση τους είναι σημαντικό να αναφερθεί ότι παρέχεται ανατροφοδότηση στον χρήστη προκειμένου να παραχθούν οι καλύτερες προτάσεις. Συγκεκριμένα, το BCS προσαρμόζει την κατάταξη των αποτελεσμάτων προωθώντας αποσπάσματα που επιλέχθηκαν συχνότερα στο παρελθόν από το χρήστη. Αυτό επιτυγχάνεται μέσω της καταγραφής του αριθμού των φορών που επιλέχθηκε κάθε απόσπασμα από το χρήστη για κάθε ερώτημα.

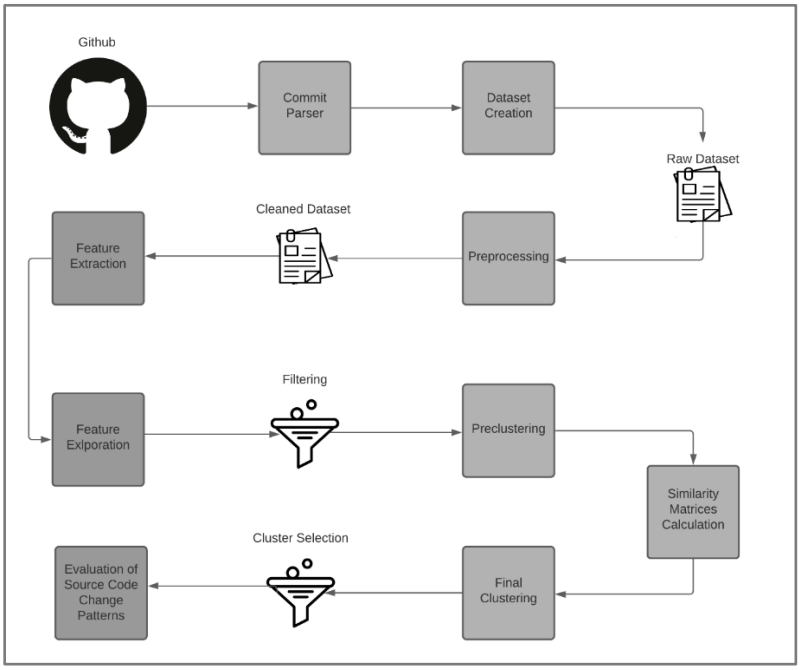
Κεφάλαιο 4. Μεθοδολογία

4.1. Γενικά

Στο κεφάλαιο αυτό παρουσιάζεται η μεθοδολογία υλοποίησης του συστήματος της παρούσας διπλωματικής. Όπως και οι έρευνες που πραγματοποιήθηκαν παραπάνω, η εργασία αυτή πραγματεύεται την αυτόματη εξαγωγή προτύπων αλλαγών κώδικα από αποθετήρια ανοικτού λογισμικού. Αρχικά καταγράφεται περιληπτικά η δομή του συστήματος και στη συνέχεια γίνεται εμβάθυνση σε κάθε διακριτό της βήμα. Η ανάπτυξη του συστήματος πραγματοποιήθηκε κάνοντας χρήση της γλώσσας προγραμματισμού python μέσω του IDE περιβάλλοντος Visual Studio Code[25].

4.2. Δομή Συστήματος

Βασική ιδέα της εργασίας, για την εξαγωγή των προτύπων αλλαγών κώδικα, αποτελεί η ομαδοποίηση αλλαγών που εμφανίζονται σε μεγάλο αριθμό έργων λογισμικού υψηλού επιπέδου, εσωτερικά μίας μεθόδου και μοιάζουν δομικά αλλά και λεξιλογικά. Επιπλέον το σύστημα, ελέγχει νέα αρχεία πηγαίου κώδικα Java με σκοπό την εύρεση τμημάτων κώδικα που εμφανίζονται στα πρότυπα αλλαγών ώστε να προτείνει τις κατάλληλες αλλαγές για να διορθωθούν πιθανά σφάλματα ή να βελτιστοποιηθεί η λειτουργία τους. Η συνολική δομή του συστήματος απεικονίζεται στην Εικόνα 4.1.



Εικόνα 4.1 Αρχιτεκτονική Συστήματος.

[25] https://code.visualstudio.com/

Πιο συγκεκριμένα, τα διάφορα στάδια ανάπτυξης του συστήματος είναι τα εξής:

- Επιλογή των commits (Commit Parser): Αρχικά, για τη δημιουργία των δεδομένων επιλέχθηκαν τα 900 πιο δημοφιλή αποθετήρια λογισμικού του GitHub, υλοποιημένα σε γλώσσα προγραμματισμού Java. Από αυτά, τα 600 αποτέλεσαν το σύνολο εκπαίδευσης (training set) και χρησιμοποιήθηκαν για την εξαγωγή των προτύπων αλλαγών κώδικα. Τα υπόλοιπα 300 σχηματίζουν το σύνολο αξιολόγησης (test set) και συντέλεσαν στην τεκμηρίωση των αποτελεσμάτων. Σε αυτό το στάδιο το σύστημα εξετάζει τα αποθετήρια, χρησιμοποιώντας το εργαλείο PyDriller[76], ανιχνεύοντας commits του βασικού κλάδου που προσπαθούν να διορθώσουν κάποιο σφάλμα. Η διαδικασία αυτή πραγματοποιείται ελέγχοντας τα μηνύματα που επισυνάπτουν οι προγραμματιστές κατά την καταχώρηση των commits. Εφόσον ανιχνευθούν, αποθηκεύεται το αντίστοιχο αναγνωριστικό τους για χρήση στα παρακάτω στάδια του συστήματος.
- Δημιουργία Συνόλου Δεδομένων (Dataset Creation): Στο σημείο αυτό πραγματοποιείται η διαδικασία παραγωγής των αρχικών δεδομένων. Καθώς στο προηγούμενο βήμα έχουν επιλεγεί τα commits του κάθε αποθετηρίου που διορθώνουν σφάλματα, τώρα ολοκληρώνεται η εξαγωγή των τροποποιήσεων του κάθε αρχείου (modifications). Οι αλλαγές των αρχείων που αποθηκεύονται δεν πρέπει να ξεπερνούν ένα άνω όριο γραμμών που τροποποιήθηκαν κατά την υλοποίηση του commit. Επιπλέον, αν σε ένα αρχείο έχουν γίνει πολλαπλές αλλαγές σε διαφορετικά σημεία, η σε διαφορετικές μεθόδους τότε αυτές διασπώνται και διατηρούνται ως ξεχωριστές εγγραφές (code changes) στο σύνολο δεδομένων. Στη συνέχεια ελέγχεται αν οι αλλαγές αυτές περιέχονται εσωτερικά μιας μεθόδου. Τέλος για κάθε αλλαγή (code change) αποθηκεύεται η μορφή της μεθόδου πριν και μετά, η αλλαγή αυτή καθαυτή, καθώς και η AST αναπαράσταση τους που παράγεται μέσω του ASTExtractor[26].
- Προεπεξεργασία Δεδομένων (Preprocessing): Στη συνέχεια, πραγματοποιείται ο διαχωρισμός των αλλαγών σε τρείς διαφορετικές κατηγορίες: αλλαγές που περιέχουν μόνο προσθέσεις κώδικα (only additions), αλλαγές που περιέχουν μόνο αφαιρέσεις κώδικα (only deletions) και τέλος αλλαγές που περιέχουν και τα δύο (both). Έπειτα, ολοκληρώνεται η διαγραφή των διπλοεγγραφών, δηλαδή ολόιδιων αλλαγών που υλοποιήθηκαν πολλαπλές φορές εσωτερικά ενός και μόνο αρχείου στο ίδιο commit.

[26] https://github.com/thdiaman/ASTExtractor

- Εξαγωγή Χαρακτηριστικών των Αλλαγών (Feature Extraction): Ακολουθώντας, για κάθε μία από τις τρείς κατηγορίες υπολογίζεται μια πληθώρα χαρακτηριστικών τα οποία σχετίζονται κυρίως με την αλγοριθμική δομή των AST των αλλαγών.
- Ανάλυση Χαρακτηριστικών (Feature Exploration): Σε αυτό το τμήμα του συστήματος, υπολογίζονται στατιστικά στοιχεία που περιγράφουν τις κατανομές των χαρακτηριστικών που αναφέρθηκαν παραπάνω, καθώς και τη μεταξύ τους *συσχέτιση* (correlation) και *αλληλεξάρτηση* (interrelationship).
- Φιλτράρισμα Δεδομένων (Data Filtering): Εφόσον, τα στατιστικά στοιχεία της προηγούμενης ενότητας εμφανίζουν μερικές ακραίες περιπτώσεις, οι οποίες μπορούν να χαρακτηρισθούν ως θόρυβος (noise), το σύνολο δεδομένων υφίσταται ένα τελικό φιλτράρισμα. Διαγράφονται δηλαδή ακραίες περιπτώσεις των εγγραφών, οι τιμές των οποίων για συγκεκριμένα χαρακτηριστικά ξεπερνούν ένα άνω όριο.
- Αρχική Ομαδοποίηση (Preclustering): Η αρχική ομαδοποίηση ολοκληρώνεται κάνοντας χρήση των τιμών των προαναφερθέντων χαρακτηριστικών, αφού πρώτα μετασχηματιστούν μέσω του state-of-the-art αλγορίθμου μείωσης της διαστασιμότητας PCA. Η χρησιμότητα της εν λόγω δομικής ομαδοποίησης, η οποία λαμβάνει χώρα χρησιμοποιώντας τον αλγόριθμο *k-means++*[77], είναι η μείωση του μεγέθους του συνόλου δεδομένων και για τις τρείς κατηγορίες αλλαγών, ώστε να είναι εφικτός, χωρικά και χρονικά, ο υπολογισμός ομοιότητας μεταξύ των ομάδων αυτών στο παρακάτω βήμα. Η επιλογή του κατάλληλου αριθμού ομάδων έγινε υπολογίζοντας το *άθροισμα των τετραγωνικών σφαλμάτων[78]* (Sum of Squared Errors).
- Υπολογισμός Πινάκων Ομοιότητας (Similarity Matrices Calculation): Στο στάδιο αυτό υπολογίζεται αρχικά η δομική ομοιότητα των AST των αλλαγών, για κάθε ομάδα του προηγούμενου βήματος, και για τις τρείς κατηγορίες, χρησιμοποιώντας τον αλγόριθμο pq-grams[79]. Στη συνέχεια υπολογίζεται και η λεξιλογική ομοιότητα του κειμένου των αλλαγών της κάθε ομάδας χρησιμοποιώντας τον συνδυασμό του αλγόριθμου TfidVectorizer[27] και της μετρικής *ομοιότητα συνημίτονού* (Cosine Similarity)[28]. Στο τέλος οι δύο αυτοί πίνακες ομοιότητας συνδυάζονται οδηγώντας σε έναν τελικό πίνακα με τον οποίο ολοκληρώνεται η τελική ομαδοποίηση των αλλαγών.

[27] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[28] https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm

- Ομαδοποίηση Αλλαγών (Final Clustering): Η τελική ομαδοποίηση των αλλαγών ολοκληρώνεται κάνοντας χρήση του ML αλγορίθμου *ιεραρχικής ομαδοποίησης συγχώνευσης*[29] (Agglomerative Hierarchical Clustering - AHC). Σε αυτή την περίπτωση η επιλογή του βέλτιστου αριθμού των τελικών ομάδων έγινε υπολογίζοντας σε κάθε επανάληψη τη μετρική Silhouette[80].
- Επιλογή Τελικών Ομάδων Αλλαγών (Cluster Selection): Από τις τελικές ομάδες που προέκυψαν επιλέγονται μόνον αυτές, οι αλλαγές των οποίων, προέρχονται από πολλά διαφορετικά αποθετήρια και ξεπερνούν ένα άνω όριο *συνοχής*[81] (cohesion).

4.2.1. Επιλογή των Commits

Στην ενότητα αυτή παρουσιάζεται το πρώτο βήμα της ανάπτυξης του συστήματος εξόρυξης προτύπων αλλαγών κώδικα, που είναι η επιλογή των κατάλληλων commits. Πηγή των δεδομένων αποτελεί ένα από τα πιο δημοφιλή DVCS, το GitHub. Η ιδέα της επιλογής των αποθετηρίων που προσφέρουν τα κατάλληλα δεδομένα στο σύστημα είναι να βρεθούν έργα λογισμικού υψηλού επιπέδου που θα είναι όσο το δυνατόν περισσότερο αντιπροσωπευτικά και θα περιέχουν ποιοτικό πηγαίο κώδικα. Για τον σκοπό αυτό, επιλέχθηκαν τα 900 πιο δημοφιλή αποθετήρια ανοικτού λογισμικού, υλοποιημένα σε γλώσσα προγραμματισμού Java, με βάση τον αριθμό των stars και των forks που έχουν λάβει στην πλατφόρμα του GitHub. Για τον έλεγχο και την προσπέλαση των αποθετηρίων χρησιμοποιήθηκε το εργαλείο PyDriller, όπως αναφέρθηκε παραπάνω.

Τα 600 από τα 900 έργα, που επιλέχθηκαν τυχαία για τη δημιουργία του συνόλου εκπαίδευσης, περιέχουν συνολικά 55.067 commits, τα οποία πραγματοποιήθηκαν στο main branch των αντίστοιχων έργων λογισμικού. Επιπλέον, στα μηνύματα τους περιέχουν τουλάχιστον μία από τις εξής λέξεις: fix, improve, change, bug, add, remove ή support. Η επιλογή των συγκεκριμένων λέξεων έγινε με βάση την ανάλυση της έρευνας με τίτλο " Improving source code readability: theory and practice"[82]. Κατά αυτό τον τρόπο γίνεται εμφανές πως οι αλλαγές είναι ποιοτικές και επίσης έγιναν στα αρχεία των έργων με στόχο την αποσφαλμάτωση ή τη βελτίωση της απόδοσης του πηγαίου κώδικα. Στη συνέχεια αποθηκεύεται για κάθε ένα από τα παραπάνω commits το μοναδικό του αναγνωριστικό, γνωστό ως *SHA* ή *hash*, ώστε να μπορούν να αναλυθούν απευθείας στο επόμενο βήμα του συστήματος.

4.2.2. Δημιουργία Συνόλου Δεδομένων

Στην ενότητα αυτή, παρουσιάζονται οι αποφάσεις που πάρθηκαν κατά τη διαδικασία διαχείρισης των παραπάνω commits με σκοπό τη δημιουργία του συνόλου δεδομένων που χρησιμοποιήθηκε για την

[29] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

υλοποίηση του συστήματος. Αρχικά το σύνολο των αρχείων καθώς και το ιστορικό των αλλαγών τους που περιέχεται εσωτερικά των commits αποθηκεύτηκε σε τοπικό επίπεδο με τη βοήθεια του Git και πιο συγκεκριμένα μέσω της διαδικασίας Git Cloning[30]. Στη συνέχεια χρησιμοποιώντας ξανά τη βιβλιοθήκη RepositoryMining του εργαλείου PyDriller, έγινε δυνατή η πρόσβαση των αλλαγών όλων των αρχείων που είχαν λάβει μέρος εσωτερικά των commits, για τα οποία είχαν αποθηκευτεί τα αναγνωριστικά τους.

Η αλλαγή κάθε ξεχωριστού αρχείου εσωτερικά ενός commit ονομάζεται modification. Τα modifications διακρίνονται σε έξι διακριτούς τύπους. Οι τύποι αυτοί είναι οι εξής: ADD, COPY, RENAME, DELETE, MODIFY, UNKNOWN. Ο τύπος ADD αναφέρεται σε commits κατά τα οποία προστέθηκαν νέα αρχεία στο αποθετήριο λογισμικού. Ο τύπος COPY αποτελείται από αντιγραφές αρχείων που πραγματοποιήθηκαν εσωτερικά του αντίστοιχου commit, ενώ ο τύπος RENAME αναφέρεται απλά σε μετονομασία αρχείων. Αντίστοιχα, με τον τύπο DELETE περιγράφονται commits που περιέχουν μόνο διαγραφή αρχείων από το αποθετήριο. Τέλος ο τύπος MODIFY αναφέρεται σε αλλαγές αρχείων κατά τις οποίες τροποποιήθηκε το περιεχόμενο τους και είναι αυτός που χρησιμοποιήθηκε για την τελική ανάλυση της εργασίας. Τα 600 έργα λογισμικού που χρησιμοποιήθηκαν περιείχαν συνολικά 470.861 modifications τύπου MODIFY με τα οποία και ολοκληρώθηκε το σύστημα.

Από το παραπάνω σύνολο των modifications, διατηρήθηκαν μόνο όσες περιπτώσεις περιείχαν αλλαγές οι οποίες επιδρούσαν σε λιγότερες από 100 γραμμές πηγαίου κώδικα. Η επιλογή αυτή διασφαλίζει τον εντοπισμό μικρών και ουσιαστικών διορθώσεων στον κώδικα που μπορούν εύκολα να επαναχρησιμοποιηθούν από άλλους προγραμματιστές. Κατά αυτό τον τρόπο απορρίφθηκαν ακριβώς 14.038 modifications από τον συνολικό αριθμό τους, δηλαδή ένα ποσοστό της τάξης του 3%.

Στη συνέχεια, ελέγχεται αν εσωτερικά ενός modification οι αλλαγές που πραγματοποιήθηκαν επηρεάζουν τμήματα του πηγαίου κώδικα, τα οποία εμπεριέχονται σε διαφορετικές μεθόδους του εκάστοτε αρχείου ή γενικότερα στο κύριο τμήμα του. Οι αλλαγές κάθε διακριτής μεθόδου αλλά και αυτές που βρίσκονται σε διαφορετικά σημεία του κύριου τμήματος του πηγαίου αρχείου αποθηκεύονται ως ξεχωριστές εγγραφές στο σύνολο των δεδομένων. Για παράδειγμα στις εικόνες Εικόνα 4.2, 4.3 παρουσιάζονται αλλαγές που πραγματοποιήθηκαν εσωτερικά του αρχείου BytecodeViewer.java σε δύο διαφορετικές μεθόδους, ενώ στην Εικόνα 4.4 παρουσιάζονται δύο αλλαγές που πραγματοποιήθηκαν στο κύριο τμήμα του, αλλά σε διαφορετικά σημεία. Το αρχείο αυτό ανήκει στο έργο λογισμικού bytecode-

[30] https://git-scm.com/docs/git-clone

viewer[31] το οποίο έχει υλοποιηθεί από τον χρήστη Konloch. Το αναγνωριστικό του commit που περιλαμβάνει το modification αυτό είναι το 30dc40a095fe9ce0a900897bef070f0a9e39c10e[32].

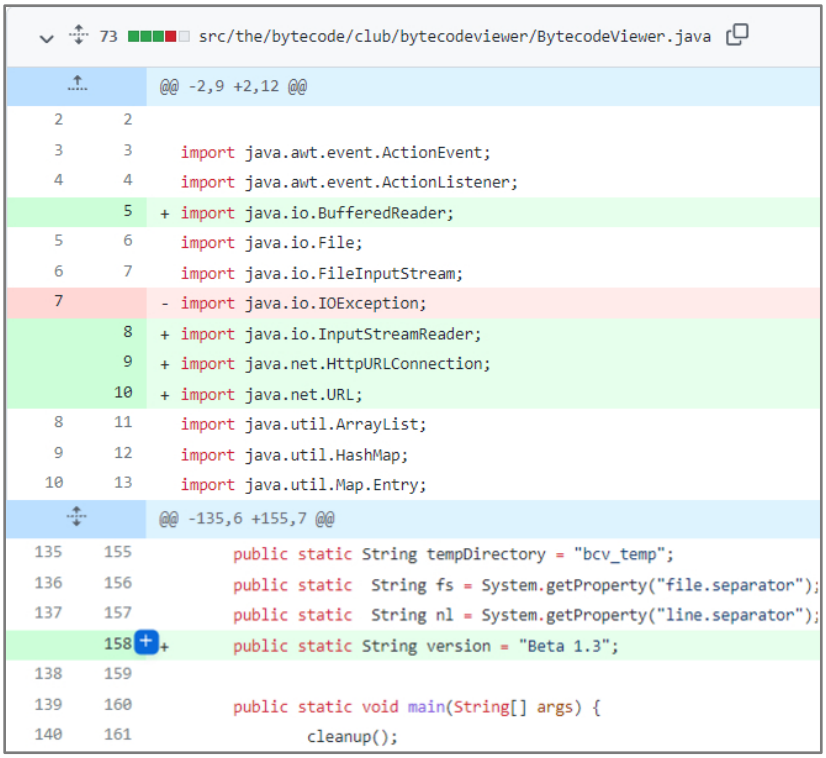
```
@@ -318,16 +359,14 @@ public void actionPerformed(ActionEvent e) {
318  359          public static void cleanup() {
319  360                  tempF = new File(tempDirectory);
320  361                  try {
321      -                       Thread.sleep(100);
322  362                           FileUtils.deleteDirectory(tempF);
323      -                       Thread.sleep(100);
324  363                  } catch (Exception e) {
325  364                  }
326  365
327  366                  while(!tempF.exists()) { //keep making dirs
328  367                          try {
329  368                                  tempF.mkdir();
330      -                               Thread.sleep(100);
     369  +                               Thread.sleep(1);
331  370                          } catch (Exception e) {
332  371                          }
333  372                  }
```

Εικόνα 4.2 Τροποποίηση της μεθόδου public void actionPerformed(ActionEvent e) εσωτερικά του αρχείου BytecodeViewer.java.

```
@@ -180,7 +219,7 @@ public static void openFiles(File[] files) {
180  219                      if (fn.endsWith(".jar")) {
181  220                          try {
182  221                              JarUtils.put(f, BytecodeViewer.loadedClasses);
183      -                       } catch (final IOException e) {
     222  +                       } catch (final Exception e) {
184  223                              e.printStackTrace();
185  224                          }
186  225
```

Εικόνα 4.3 Τροποποίηση της μεθόδου public void openFiles(File[] files) εσωτερικά του αρχείου BytecodeViewer.java.

[31] https://github.com/Konloch/bytecode-viewer
[32] https://github.com/Konloch/bytecode-viewer/commit/30dc40a095fe9ce0a900897bef070f0a9e39c10e



Εικόνα 4.4 Τροποποίηση του κύριου τμήματος του αρχείου BytecodeViewer.java σε δύο διαφορετικά σημεία.

Για να μη δημιουργείται σύγχυση με τη χρήση των όρων “αλλαγή” και “τροποποίηση”, οι παραπάνω αλλαγές, δηλαδή αυτές που αναφέρονται σε ξεχωριστές μεθόδους ή διαφορετικά σημεία του κύριου τμήματος κάθε πηγαίου αρχείου από εδώ και στο εξής θα αναφέρονται ως *code changes*. Ο συνολικός αριθμός των code changes που απαρτίζουν το αρχικό σύνολο δεδομένων είναι 730.320. Βέβαια κατά την επεξεργασία κάθε μεμονωμένου modification, μόλις διαχωρίζονται τα ξεχωριστά code changes, ελέγχεται η ομοιότητα των αλλαγών τους για την αποφυγή αποθήκευσης διπλοεγγραφών, καθώς και αν ανήκουν στο εσωτερικό μιας μεθόδου. Πιο συγκεκριμένα από τις αλλαγές του αρχείου BytecodeViewer.java, δημιουργούνται δύο διακριτές εγγραφές στο σύνολο των δεδομένων του συστήματος, οι οποίες αναγράφουν τις αλλαγές των εξής μεθόδων εφόσον δεν είναι όμοιες:

- public void actionPerformed(ActionEvent e) - Εικόνα 4.2.
- public static void openFiles(File[] files) - Εικόνα 4.3.

ενώ οι αλλαγές του κυρίου τμήματος διαγράφονται επειδή δεν ανήκουν σε κάποια μέθοδο. Συγκεκριμένα, παρατηρήθηκε σε πολλαπλές περιπτώσεις πως παρόλο που οι αλλαγές είχαν πραγματοποιηθεί σε διαφορετικές μεθόδους, ήταν ολόιδιες. Η διατήρηση όλων των όμοιων αλλαγών του ίδιου αρχείου, θα οδηγούσε στην ύπαρξη θορύβου κατά τη συνέχεια της υλοποίησης του συστήματος. Για τον λόγο αυτό, χρησιμοποιώντας τη συνάρτηση ratio() της κλάσης SequenceMatcher

της βιβλιοθήκης difflib[33] διαγράφηκαν οι πολλαπλές περιπτώσεις ίδιων αλλαγών διατηρώντας στο σύνολο των δεδομένων μόνο μια. Η συνάρτηση ratio() επιστρέφει μια δεκαδική μέτρηση της ομοιότητας δύο ακολουθιών στο εύρος [0,1].

$$SequenceMatcher.ratio(sequence1, sequence2) = {}^{2.0 * M}/_{T} \quad (4.1)$$

όπου Τ είναι ο συνολικός αριθμός των στοιχείων και των δύο ακολουθιών, και Μ είναι ο αριθμός των matches.

Είναι φανερό πως το αποτέλεσμα είναι ίσο με τη μονάδα όταν οι δύο ακολουθίες είναι όμοιες, και 0 όταν δεν έχουν κανένα κοινό στοιχείο. Θεωρείται πως ένα code change περιέχει ίδιες αλλαγές με ένα άλλο, όταν το αποτέλεσμα του SequenceMatcher για τον πηγαίο κώδικα που προστέθηκε καθώς και για τον πηγαίο κώδικα που αφαιρέθηκε ξεπερνάει ταυτόχρονα την τιμή 0.8. Η τιμή αυτή προέκυψε έπειτα από πειραματισμό και χειροκίνητο έλεγχο των αποτελεσμάτων. Ο αριθμός των διπλοεγγραφών εσωτερικά των modifications ήταν 89.622, ενώ τα code changes που δεν ανήκουν σε μία μέθοδο 453.257, περίπου δηλαδή 12,3% και 62% του συνολικού αριθμού, αντίστοιχα. Οπότε η δημιουργία του αρχικού συνόλου δεδομένων έχει ως αποτέλεσμα 187.441 code changes. Για κάθε μία από αυτές τις περιπτώσεις αποθηκεύονται τα χαρακτηριστικά που εμφανίζονται στον Πίνακα 4.1.

Πίνακας 4.1 Χαρακτηριστικά του Συνόλου Δεδομένων.

Χαρακτηριστικό	Περιγραφή
SHA	Το μοναδικό αναγνωριστικό του commit.
Filename	Η διεύθυνση του αρχείου πηγαίου κώδικα εσωτερικά του αποθετηρίου.
Repo_name	Το όνομα του αποθετηρίου στο οποίο ανήκει το αρχείο.
Commit_date	Η ημερομηνία διεξαγωγής του Commit που περιλαμβάνει το code change.
Commit_message	Το μήνυμα καταχώρησης του Commit που περιλαμβάνει το code change.
Code_diff	Ο πηγαίος κώδικας σε μορφή διαφορών.
Method_Code_Before	Ο πηγαίος κώδικας της μεθόδου στην οποία πραγματοποιείται το code change πριν την εφαρμογή του commit.
Method_Code_After	Ο πηγαίος κώδικας της μεθόδου στην οποία πραγματοποιείται το code change μετά την εφαρμογή του commit.
Method_Code_Before_AST	Η AST αναπαράσταση του Method_Code_Before.
Method_Code_After_AST	Η AST αναπαράσταση του Method_Code_After.

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

Code_Additions	Ο πηγαίος κώδικας που προστέθηκε στη μέθοδο κατά την εφαρμογή του commit.
Code_Deletions	Ο πηγαίος κώδικας που αφαιρέθηκε από τη μέθοδο κατά την εφαρμογή του commit.
Code_Additions_AST	Το AST του Code_Additions.
Code_Deletions_AST	Το AST του Code_Deletions.

Τα χαρακτηριστικά SHA, filename, repo_name, commit_date και commit_message παρέχονται απευθείας από τον PyDriller. Επίσης ο PyDriller, παρέχει την πληροφορία για τη μορφή του κάθε αρχείου πηγαίου κώδικα πριν την εφαρμογή του commit, το συνολικό code_diff του commit, καθώς και τη μορφή του αρχείου μετά την εφαρμογή του commit. Βέβαια, στην παρούσα εργασία αυτό ήταν χρήσιμο, μόνο για τις περιπτώσεις που οι τροποποιήσεις εσωτερικά του αρχείου συμβαίνουν στο εσωτερικό μιας και μόνο μεθόδου, διαφορετικά τα αντίστοιχα δεδομένα έπρεπε να υπολογιστούν μεμονωμένα. Ο υπολογισμός αυτός πραγματοποιήθηκε σε επίπεδο γραμμής, λαμβάνοντας υπόψιν το σύνολο των αλλαγών που περιλαμβάνονται στις παραπάνω πληροφορίες που παρέχει ο PyDriller. Τέλος ελέγχοντας τα νέα code_diff που παράγονται, ψάχνοντας στο εσωτερικό τους για τα σύμβολα “+” και “-“ εξάγονται και τα χαρακτηριστικά code_additions και code_deletions για κάθε code change.

Στη συνέχεια ακολουθεί η παραγωγή των ASTs των χαρακτηριστικών: method_code_before, method_code_after, code_additions και code_deletions. Αυτή πραγματοποιήθηκε με το εργαλείο ASTExtractor, ένα εργαλείο βασισμένο στον Eclipse Compiler που επιτρέπει την εξαγωγή ASTs για αρχεία κώδικα Java σε XML format, ενώ μπορεί να χρησιμοποιηθεί και ως βιβλιοθήκη. Παρακάτω στους πίνακες Πίνακας 4.2, Πίνακας 4.3 παρουσιάζεται ένα παράδειγμα των δεδομένων για την καλύτερη κατανόηση από τον αναγνώστη.

Πίνακας 4.2 Παράδειγμα Δεδομένων ενός Code Change.

Filename: src\main\groovy\org\gradle\execution\Dag.java	**Commit SHA: ca605c56c816a63025bbe62247d112358c17b98c**

Code Diff	@@ -233,7 +233,7 @@ public Dag execute() {
	private void execute(Set<DefaultTask> tasks) {
	for (DefaultTask task : tasks) {
	execute(new TreeSet(getChildren(task)));
	- if (!task.getExecuted()) {
	+ if (!task.isExecuted()) {
	logger.info("Executing: " + task);
	task.execute();
	}

Method Code Before	private void execute(Set<DefaultTask> tasks){ for (DefaultTask task : tasks) {

	execute(new TreeSet(getChildren(task)));
	If (!task.getExecuted()) {
	logger.info("Executing: " + task);
	task.execute();
	}
	}
	}

Method Code After	private void execute(Set<DefaultTask> tasks){
	for (DefaultTask task : tasks) {
	execute(new TreeSet(getChildren(task)));
	if (!task.isExecuted()) {
	logger.info("Executing: " + task);
	task.execute();
	}
	}
	}

Code Additions	if (!task.isExecuted()) {
Code Deletions	if (!task.getExecuted()) {

Πίνακας 4.3 AST των Προσθέσεων και Αφαιρέσεων του Παραδείγματος.

Code Additions AST	Code Deletions AST
<ROOT> <SimpleName>isExecuted</SimpleName> </ROOT>	<ROOT> <SimpleName>getExecuted</SimpleName> </ROOT>

4.2.3. Προεπεξεργασία Δεδομένων

Στην ενότητα αυτή πραγματοποιείται η διαδικασία της προεπεξεργασίας των δεδομένων, κατά την οποία διαχωρίζονται τα δεδομένα σε τρείς κατηγορίες, ανάλογα με τον τύπο των αλλαγών που πραγματοποιήθηκαν εσωτερικά των μεθόδων. Στη συνέχεια διαγράφονται ξανά περιπτώσεις διπλοεγγραφών, αλλά αυτή τη φορά, οι διπλοεγγραφές προέρχονται από διαφορετικά αρχεία που τροποποιήθηκαν στο ίδιο commit και όχι απλώς από το ίδιο αρχείο.

4.2.3.1. Διαχωρισμός του Συνόλου Δεδομένων

Πιο συγκεκριμένα, στα δεδομένα παρατηρούνται τρείς διαφορετικές περιπτώσεις αλλαγών του πηγαίου κώδικα στο εσωτερικό των μεθόδων. Οι περιπτώσεις αυτές χαρακτηρίζονται ως *only_additions, only_deletions* και *both.* Προφανώς, η ορολογία που ορίζεται αποσκοπεί στην απεικόνιση και τον χαρακτηρισμό των αλλαγών. Τα code changes που χαρακτηρίζονται ως only_additions αναφέρονται σε αλλαγές κατά τις οποίες προστέθηκαν νέες γραμμές πηγαίου κώδικα εσωτερικά της μεθόδου. Τα only_deletions χαρακτηρίζουν αλλαγές κατά τις οποίες αφαιρέθηκαν γραμμές πηγαίου κώδικα από μια μέθοδο, ενώ τα both περιλαμβάνουν και προσθέσεις και αφαιρέσεις. Στον Πίνακα 4.4 εμφανίζεται ο

αριθμός των εγγραφών των τριών κατηγοριών καθώς και το ποσοστό του συνολικού σετ δεδομένων που καλύπτει η κάθε μια.

Πίνακας 4.4 Αριθμός Εγγραφών των Τριών Κατηγοριών των Δεδομένων.

Κατηγορία	Αριθμός Εγγραφών	Ποσοστό Συνολικού Σετ Δεδομένων
Both	137.112	73,15 %
Only_additions	35.490	18,93 %
Only_deletions	14.839	7,92 %

Αφού ολοκληρωθεί ο διαχωρισμός των τριών κατηγοριών ακολουθεί η διαγραφή των διπλοεγγραφών. Κατά αυτό τον τρόπο, γίνεται αφαίρεση του θορύβου που δημιουργούν τα code changes από διαφορετικά αρχεία του ίδιου commit με ίδια code_additions και code_deletions. Συνήθως, όταν συναντώνται διαφορετικά αρχεία με ολόιδιες αλλαγές στο ίδιο commit, πρόκειται για αρχεία με συμπληρωματική λειτουργία, δηλαδή αρχεία που αλληλεξαρτώνται και πραγματοποιούνται κλήσεις του ενός στο εσωτερικό του άλλου. Σε αυτή την περίπτωση οι αλλαγές είναι ίδιες, ώστε να μη δημιουργούνται σφάλματα κατά την εκτέλεση τους. Το γεγονός αυτό βέβαια, θα δημιουργούσε πολλαπλές εμφανίσεις των ίδιων αλλαγών στην τελική ομαδοποίηση, το οποίο δε θεωρείται σωστό καθώς θα δυσχέραινε τη γενίκευση των αποτελεσμάτων.

4.2.3.2. Διαγραφή Διπλοεγγραφών Εσωτερικά του Ίδιου Commit

Τα παρακάτω βήματα, που αποτελούν τη διαδικασία που ακολουθήθηκε για τη διαγραφή των διπλοεγγραφών αυτών, επαναλαμβάνονται και για τις τρείς κατηγορίες, με μικρές διαφοροποιήσεις για την κάθε μία. Αρχικά, για κάθε μοναδικό SHA εσωτερικά των δεδομένων της κάθε κατηγορίας, δημιουργείται μια λίστα στην οποία αποθηκεύονται οι εκάστοτε αλλαγές του πηγαίου κώδικα σε AST μορφή, καθώς με τη χρήση των ASTs υπολογίζεται η δομική ομοιότητα μεταξύ δύο τμημάτων πηγαίου κώδικα. Για την κατηγορία των only_additions αποθηκεύονται ουσιαστικά όλα τα code_additions_ASTs που ανήκουν στο ίδιο commit. Για τα only_deletions αντίστοιχα αποθηκεύονται τα code_deletions_ASTs, ενώ για τα both δημιουργούνται δύο λίστες, μία για τα code_additions_ASTs και μια για τα code_deletions_ASTs.

Βέβαια τα ASTs χρειάζεται να μετατραπούν σε *ordered labeled trees* προκειμένου να ολοκληρωθεί ο υπολογισμός της ομοιότητας. Χαρακτηριστικά, για ένα δέντρο T με σύνολο κόμβων $V(T)$ και ακμές $E(T)$, σε κάθε κόμβο του $v \in V(T)$ προστίθεται μία ετικέτα (label) με ένα ASTNode που αντιστοιχεί σε ένα στοιχείο της γλώσσας προγραμματισμού Java (π.χ. MethodDeclaration, Block κ.τ.λ.). Τα δέντρα αυτά επίσης περιέχουν διατεταγμένες ακμές (ordered edges) καθώς για κάθε ακμή αντιστοιχεί ένα ζεύγος

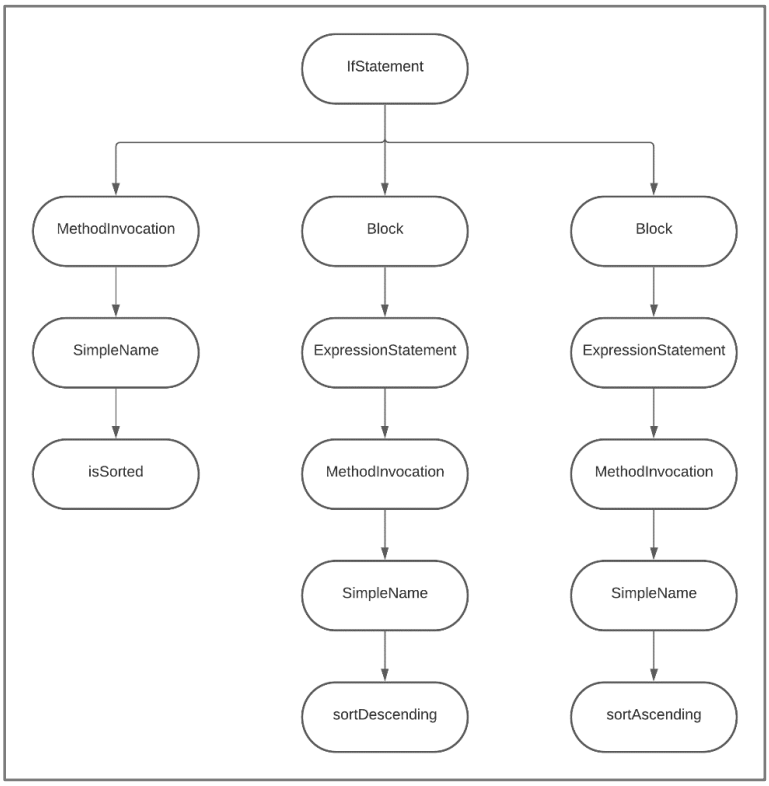
(p, c) με τον κόμβο p να είναι ο πατέρας του c. Επίσης, η διάταξη των δέντρων ορίζεται και ως προς τα παιδιά του κάθε κόμβου, χαρακτηριστικά ο κόμβος c αποτελεί το i-οστό παιδί του p όταν:

$$i = \{ x \in V(T) \mid (p, x) \in E(T), x \leq c \} \quad (4.2)$$

Για παράδειγμα στο τμήμα κώδικα της Εικόνας 4.5 αντιστοιχεί το ordered label tree που φαίνεται στην Εικόνα 4.6.

```
1 if (isSorted()) {
2     sortDescending();
3 }
4 else {
5     sortAscending();
6 }
```

Εικόνα 4.5 Παράδειγμα ενός Τμήματος Κώδικα Java.

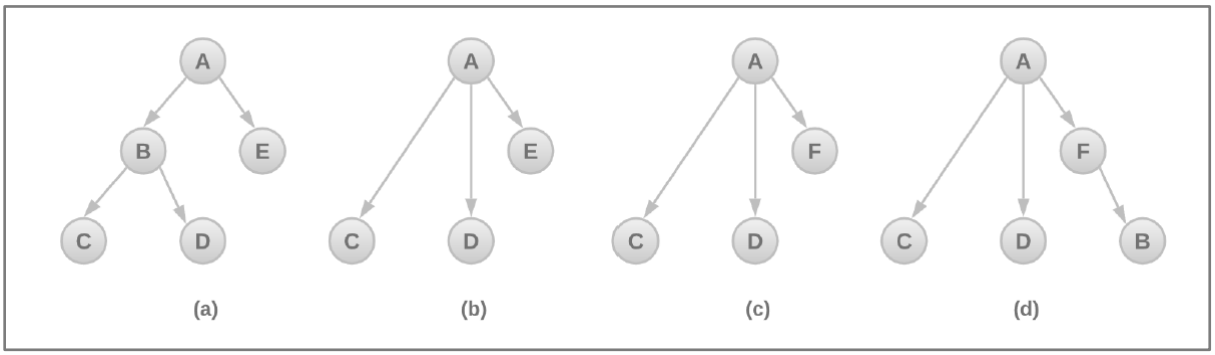


Εικόνα 4.6 Ordered Label AST.

Μετά τη δημιουργία των διατεταγμένων δέντρων, η σύγκριση των code_additions/deletions_ASTs βασίζεται στην *απόσταση επεξεργασίας δέντρου* (Tree Edit Distance - TED). Η TED χρησιμοποιείται για τον υπολογισμό της απόστασης μεταξύ δύο διατεταγμένων δέντρων T_1, T_2 (ordered label trees) και

ορίζεται ως ο ελάχιστος αριθμός των λειτουργιών επεξεργασίας (edit operations) που χρειάζεται να γίνουν στο δέντρο T_1 ώστε να μετατραπεί στο T2[83]. Οι λειτουργίες επεξεργασίας που επιτρέπεται να πραγματοποιηθούν εμφανίζονται στην Εικόνα 4.7 και είναι οι εξής:

- Μετονομασία ενός κόμβου, αλλάζοντας την ετικέτα (label).
- Διαγραφή ενός κόμβου v με πατέρα u, μετατρέποντας τα παιδία του v σε παιδιά του u.
- Εισαγωγή ενός κόμβου v ως παιδί του u, μετατρέποντας τα παιδιά του u σε παιδιά του v.



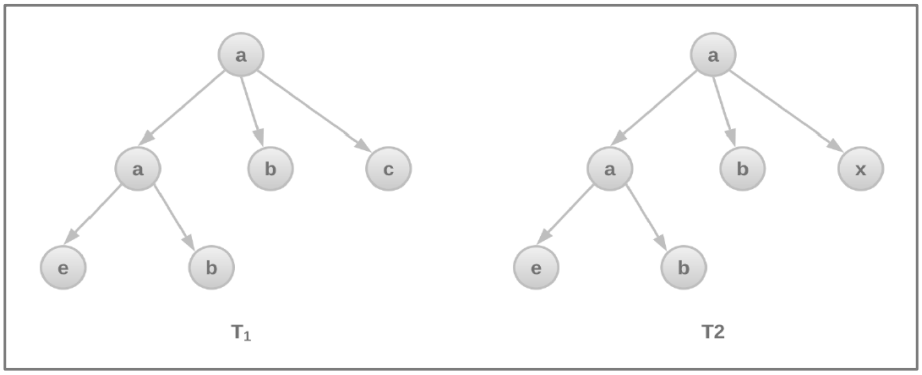
Εικόνα 4.7 Παράδειγμα δέντρου (a) και οι τρεις ενέργειες επεξεργασίας: (b) μετονομασία κόμβου, (c) διαγραφή κόμβου, (d) εισαγωγή κόμβου.

Η απόσταση επεξεργασίας δέντρου, αν και αποτελεί έναν από τους πιο δημοφιλείς τρόπους για τη σύγκριση δέντρων, έχει μεγάλη πολυπλοκότητα και, για συστήματα όπου ο αριθμός των συγκρίσεων είναι μεγάλος, συνήθως δεν προτιμάται. Για τον λόγο αυτό χρησιμοποιήθηκε ο αλγόριθμος pq-grams που υπολογίζει προσεγγιστικά την *απόσταση επεξεργασίας δέντρου* με πολύ μικρότερη πολυπλοκότητα. Ο αλγόριθμος αυτός παρουσιάστηκε το 2005 από τους N. Augusten et. al.[84] και βασίζεται σε δύο παραμέτρους p, q για τη σύγκριση των δέντρων. Για τη υλοποίηση της συγκεκριμένης μεθόδου υπολογισμού της απόστασης δέντρων έγινε χρήση της βιβλιοθήκης PyGram[34]. Αρχικά, για τον υπολογισμό της απόστασης μεταξύ δύο δέντρων T_1, T_2, όπως αυτά που φαίνονται στην Εικόνα 4.8, ο αλγόριθμος pq-grams δημιουργεί ένα δέντρο επέκτασης (extended tree) T^{pq} για κάθε ένα από αυτά. Για το σχηματισμό του δέντρου επέκτασης εισάγονται κενοί κόμβοι (null nodes), που συμβολίζονται με το χαρακτήρα * , με την εξής διαδικασία:

- $(p-1)$ null κόμβοι εισάγονται στη ρίζα του δέντρου ως πρόγονοι.
- $(q-1)$ null κόμβοι εισάγονται ως παιδιά, πριν το πρώτο και μετά το τελευταίο παιδί, κάθε ενδιάμεσου κόμβου (non-leaf node).

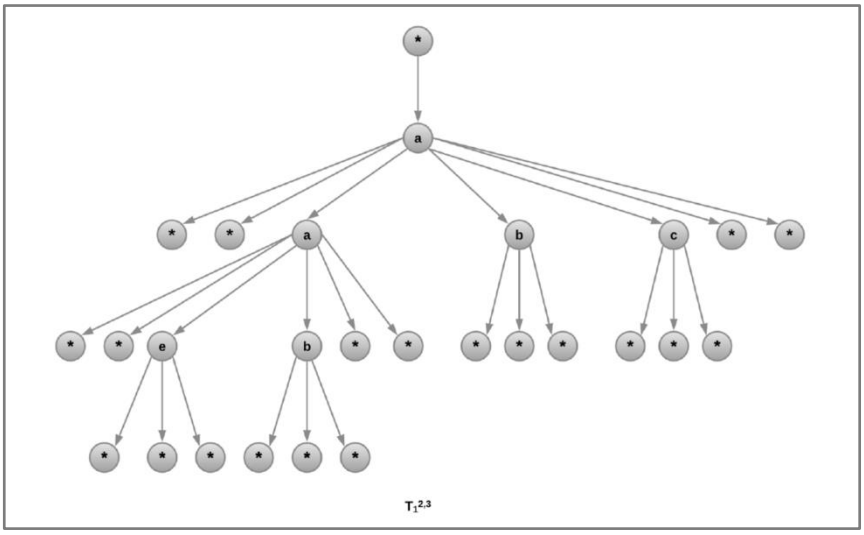
[34] https://github.com/TylerGoeringer/PyGram

- q παιδιά εισάγονται σε κάθε φύλλο του δέντρου.



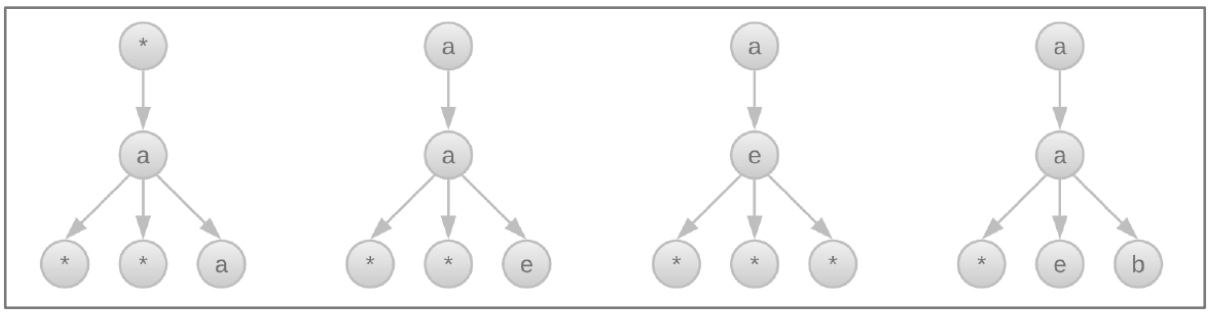
Εικόνα 4.8 Δέντρα T1 και T2.

Στη συνέχεια, με βάση το εκτεταμένο δέντρο, το οποίο εμφανίζεται στην Εικόνα 4.9, υπολογίζονται όλα τα pq-grams trees που εμπεριέχονται σε αυτό.



Εικόνα 4.9 Εκτεταμένο δέντρο του T1 με παραμέτρους p=2 και q=3.

Ως pq-gram tree ορίζεται το υποδέντρο που περιέχει έναν κόμβο με $p-1$ προγόνους και q παιδιά. Στην Εικόνα 4.10 φαίνονται μερικά παραδείγματα υποδέντρων του ${T_1}^{2,3}$. Τα pq-grams trees μπορούν να σημειωθούν ως ακολουθία labels διασχίζοντας το υποδέντρο με τη μέθοδο pre-order. Για τη διάσχιση του δέντρου με τη συγκεκριμένη μέθοδο αρχικά διασχίζεται η ρίζα του δέντρου και στη συνέχεια διασχίζονται τα υπόλοιπα υποδέντρα αναδρομικά από αριστερά προς τα δεξιά. Το σύνολο των pq-grams trees ενός εκτεταμένου δέντρου ονομάζεται προφίλ (pq-gram-Profile). Στην Εικόνα 4.11 παρουσιάζονται τα προφίλ των T_1, T_2 με τη μορφή ακολουθίας ετικετών.



Εικόνα 4.10 Παραδείγματα Υποδέντρων του $T_1^{2,3}$.

$P^{2,3}(T_1)$ labels	$P^{2,3}(T_2)$ labels
(*, a, *, *, a)	(*, a, *, *, a)
(a, a, *, *, e)	(a, a, *, *, e)
(a, e, *, *, *)	(a, e, *, *, *)
(a, a, *, e, b)	(a, a, *, e, b)
(a, b, *, *, *)	(a, b, *, *, *)
(a, a, e, b, *)	(a, a, e, b, *)
(a, a, b, *, *)	(a, a, b, *, *)
(*, a, *, a, b)	(*, a, *, a, b)
(a, b, *, *, *)	(a, b, *, *, *)
(*, a, a, b, c)	(*, a, a, b, x)
(a, c, *, *, *)	(a, x, *, *, *)
(*, a, b, c, *)	(*, a, b, x, *)
(*, a, c, *, *)	(*, a, x, *, *)

Εικόνα 4.11 Προφίλ των δέντρων T1 και T2 με παραμέτρους p=2 και q=3.

Για τον υπολογισμό της απόστασης, ο αλγόριθμος χρησιμοποιεί τα profiles των δέντρων που έχουν υπολογιστεί. Ο αριθμός των κοινών pq-grams που υπάρχουν στα profiles των δέντρων καθορίζει την τιμή της απόστασης. Συγκεκριμένα η τιμή της απόστασης μεταξύ δύο δέντρων καθορίζεται από τη παρακάτω σχέση:

$$\mathrm{d}^{p,q}(T_1, T_2) = 1 - 2\frac{|P^{p,q}(T_1) \cap P^{p,q}(T_2)|}{|P^{p,q}(T_1) \cup P^{p,q}(T_2)|} \quad (4.3)$$

όπου $|P^{p,q}(T_1) \cap P^{p,q}(T_2)|$ ο αριθμός κοινών pq-grams δέντρων που έχουν τα T_1, T_2 και $|P^{p,q}(T_1) \cup P^{p,q}(T_2)|$ το άθροισμα των pq-grams που έχουν τα T1, T2. Για παράδειγμα, για τα δέντρα T_1 και T_2 με βάση την Εικόνα 4.11 όπου φαίνονται τα προφίλ τους, προκύπτει ότι $|P^{p,q}(T_1) \cup P^{p,q}(T_2)|$ = 26 και $|P^{p,q}(T_1) \cap P^{p,q}(T_2)|$ = 9. Επομένως η απόσταση των δύο δένδρων είναι:

$$\mathrm{d}^{p,q}(T_1, T_2) = 1 - 2\frac{|P^{p,q}(T_1) \cap P^{p,q}(T_2)|}{|P^{p,q}(T_1) \cup P^{p,q}(T_2)|} = 1 - 2\frac{9}{26} = 0.31$$

Η απόσταση που προκύπτει από τη σχέση (4.3) είναι κανονικοποιημένη, συγκεκριμένα λαμβάνει τιμές στο διάστημα [0,1]. Για τον υπολογισμό της ομοιότητας αρκεί να αφαιρέσουμε την απόσταση $\mathrm{d}^{p,q}(T_1, T_2)$ από τη μονάδα οπότε και προκύπτει η παρακάτω σχέση:

$$sim^{p,q}(T_1, T_2) = 2\frac{|P^{p,q}(T_1) \cap P^{p,q}(T_2)|}{|P^{p,q}(T_1) \cup P^{p,q}(T_2)|} \quad (4.4)$$

Με βάση την παραπάνω ανάλυση, υπολογίστηκαν οι ομοιότητες για την κάθε κατηγορία. Όπως αναφέρθηκε και προηγουμένως η παραπάνω ανάλυση για τα only_additions ολοκληρώθηκε χρησιμοποιώντας τα code_additions_ASTs, για τα only_deletions τα code_deletions_ASTs, ενώ για την κατηγορία των both υπολογίστηκε η ομοιότητα και για τα code_additions_ASTs, αλλά και για τα code_deletions_ASTs. Τέλος, όσα ζεύγη εγγραφών, που περιέχουν αλλαγές μεθόδων οι οποίες πραγματοποιήθηκαν σε διαφορετικά αρχεία του ίδιου commit, είχαν τιμή 1 στην εξίσωση (4.4) διαγράφονται και από αυτά διατηρείται μόνο ένα. Στην κατηγορία των both γίνεται πρόσθεση της ομοιότητας των code_additions_AST και των code_deletions_AST και αν η τιμή είναι ίση με 2 (δηλαδή παρουσιάζουν απόλυτη ομοιότητα τόσο ως προς τα code_additions όσο και ως προς τα code_deletions), μόνο τότε θεωρείται πως εμφανίζονται διπλοεγγραφές.

4.2.4. Εξαγωγή Χαρακτηριστικών των Αλλαγών

Στην ενότητα αυτή παρουσιάζεται η διαδικασία υπολογισμού των χαρακτηριστικών των code changes. Ο αριθμός των χαρακτηριστικών φτάνει τα 23 για τις κατηγορίες των only_additions και only_deletions code changes, ενώ τα 46 για την κατηγορία των both. Αυτό συμβαίνει διότι υπολογίστηκαν δύο φορές τα χαρακτηριστικά στην προκειμένη κατηγορία, μία χρησιμοποιώντας την μεταβλητή code_additions_AST ή code_additions – ανάλογα με το εκάστοτε χαρακτηριστικό – και μία χρησιμοποιώντας τη μεταβλητή code_deletions_AST ή code_deletions αντίστοιχα. Στον Πίνακα 4.5 εμφανίζονται τα χαρακτηριστικά καθώς και η περιγραφή τους.

Πίνακας 4.5 Πίνακας Χαρακτηριστικών των Code Changes.

Χαρακτηριστικό	Περιγραφή
Number_of_lines_changed	Ο αριθμός των γραμμών πηγαίου κώδικα που περιλαμβάνει η μεταβλητή code_additions ή code_deletions.
Cyclomatic_complexity	Ο αριθμός των control flow statements[35].
Number_of_AST_children	Ο αριθμός των παιδιών του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_names	Ο αριθμός των στοιχείων Simple Names του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Unique_operators_count	Ο αριθμός των μοναδικών τελεστέων που περιέχει η μεταβλητή code_additions ή code_deletions.

[35] Control flow statements = [if, else, else if, while, for, and]

Number_of_method_invocations	Ο αριθμός των στοιχείων Method Invocations του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_if_statements	Ο αριθμός των στοιχείων If Statements του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_variable_statements	Ο αριθμός των στοιχείων Variable Statements του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_switch_statements	Ο αριθμός των στοιχείων Switch Statements του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_conditional_expression	Ο αριθμός των στοιχείων Conditional Expressions του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_class-instance_creations	Ο αριθμός των στοιχείων Class Instance Creations του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_while_statements	Ο αριθμός των στοιχείων While Statements του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_for_statements	Ο αριθμός των στοιχείων For Statements του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_try_statements	Ο αριθμός των στοιχείων Try Statements του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_return_statement	Ο αριθμός των στοιχείων Return Statements του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_single_variable_declaration	Ο αριθμός των στοιχείων Single Variable Declarations του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
Number_of_qualified_names	Ο αριθμός των στοιχείων Qualified Names του AST δέντρου της μεταβλητής code_additions_AST ή code_deletions_AST.
For_flag	Boolean μεταβλητή που υποδεικνύει την εμφάνιση ή όχι μιας εντολής For.
While_flag	Boolean μεταβλητή που υποδεικνύει την εμφάνιση ή όχι μιας εντολής While.
Do_flag	Boolean μεταβλητή που υποδεικνύει την εμφάνιση ή όχι μιας εντολής Do.
Switch_flag	Boolean μεταβλητή που υποδεικνύει την εμφάνιση ή όχι μιας εντολής Switch.
If_flag	Boolean μεταβλητή που υποδεικνύει την εμφάνιση ή όχι μιας εντολής If.

Try_flag	Boolean μεταβλητή που υποδεικνύει την εμφάνιση εντολής Try.

Πιο συγκεκριμένα τα χαρακτηριστικά καταμέτρησης των Simple Names, Method Invocations, If Statements, Variable Statements, Switch Statements, Conditional Expressions, Class Instance Creations, While Statements, For Statements, Try Statements, Return Statements, Single Variable Declarations και Qualified Names αναφέρονται στα συστατικά μέρη των ASTs που περιγράφουν την δομή ενός πηγαίου κώδικα Java. Πιο αναλυτικά, το στοιχείο Simple Name αποτελεί τον κόμβο του AST δέντρου ενός πηγαίου κώδικα, που περιλαμβάνει ως φύλλο, το όνομα μιας μεταβλητής, μιας μεθόδου ή μιας κλάσης που καλείται στο αντίστοιχο τμήμα του. Επιπρόσθετα, οι κόμβοι με όνομα Method Invocations αναφέρονται στις επικλήσεις μεθόδων που πραγματοποιούνται εσωτερικά του πηγαίου κώδικα. Τα χαρακτηριστικά που μετρούν τον αριθμό εμφάνισης των If, Switch, While, For, Try και Return Statements είναι εμφανές πως καταγράφουν τις περιπτώσεις χρήσης των συγκεκριμένων μπλοκ κώδικα στο εσωτερικό του δέντρου. Τα Variable Statements αναφέρονται στις περιπτώσεις δήλωσης μεταβλητών, ενώ τα Class Instance Creations σε περιπτώσεις δημιουργίας νέων κλάσεων. Οι κόμβοι Conditional Expressions απεικονίζουν τις εντολές διαχείρισης συνθηκών. Τέλος, τα Single Variable Declarations περιγράφουν την δήλωση πολλαπλών μεταβλητών στην ίδια γραμμή πηγαίου κώδικα, ενώ τα Qualified Names αποτελούν ονόματα κλάσεων τα οποία περιέχουν το πακέτο (package) από το οποίο προέρχεται η συγκεκριμένη κλάση.

Είναι εμφανές πως στα περισσότερα από τα παραπάνω χαρακτηριστικά γίνεται καταμέτρηση των εμφανίσεων των διαφορετικών στοιχείων που αποτελούν τα βασικά δομικά στοιχεία ενός AST δέντρου. Στόχο της παραπάνω καταμέτρησης αποτελεί η δημιουργία ενός νέου σετ δεδομένων που χαρακτηρίζει τα code changes, ώστε να μπορεί να πραγματοποιηθεί μια αρχική διαχώριση των δεδομένων σε μικρότερα groups τα οποία να έχουν παρόμοιο δομικό περιεχόμενο. Η επιλογή αυτή κρίθηκε αναγκαία, καθώς το μέγεθος του συνόλου δεδομένων και για τις τρείς κατηγορίες ήταν πολύ μεγάλο και θα δημιουργούσε προβλήματα χώρου και χρόνου εκτέλεσης στην τελική ομαδοποίηση των αλλαγών. Πέρα από τα χαρακτηριστικά που βασίζονται κυρίως στη δομή των ASTs, υπάρχουν και μερικά τα οποία απλά μετρούν ιδιότητες του πηγαίου κώδικα που προστέθηκε η αφαιρέθηκε, ή και τα δύο, στην εκάστοτε κατηγορία, όπως για παράδειγμα Unique operators count, Cyclomatic complexity, Number of lines changed. Σύμφωνα και με παλαιότερες έρευνες που αντιμετωπίζουν το ίδιο ή παρόμοια θέματα, φαίνεται πως τα συγκεκριμένα χαρακτηριστικά προσδίδουν μεγάλη διακριτική ικανότητα μεταξύ των code changes, το οποίο φαίνεται πως ισχύει και στα αποτελέσματα που παρουσιάζονται παρακάτω.

Τέλος, όσον αφορά την κατηγορία των both code changes, αφού οι τιμές των παραπάνω χαρακτηριστικών υπολογίστηκαν και για τις προσθέσεις και για τις αφαιρέσεις του κώδικα και των αντίστοιχων AST δέντρων, προστέθηκαν ώστε να μπορούν να υποστούν επεξεργασία πιο εύκολα και να μειωθεί η διαστασιμότητα του προβλήματος.

4.2.5. Ανάλυση Χαρακτηριστικών

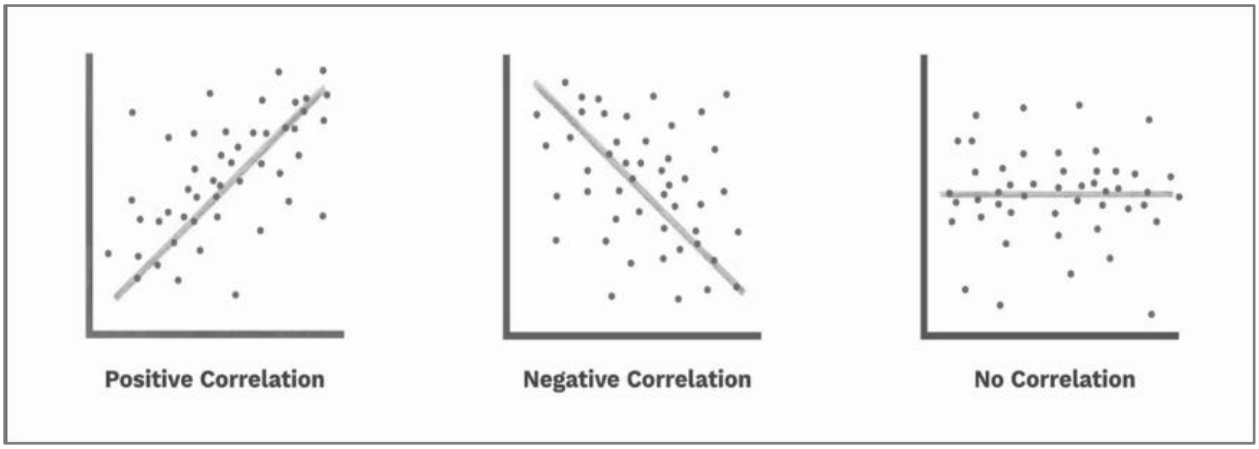
Στην ενότητα αυτή υλοποιείται στατιστική ανάλυση των χαρακτηριστικών που υπολογίστηκαν στο προηγούμενο βήμα, με στόχο τη διατήρηση αυτών που προσφέρουν τη μεγαλύτερη διακριτική ικανότητα. Τα συμπεράσματα αυτού του βήματος αποτελούν πολύ σημαντικό κομμάτι για τη μείωση της διαστασιμότητας του συνόλου των χαρακτηριστικών, καθώς και για την υλοποίηση ποιοτικού φιλτραρίσματος των δεδομένων. Δίχως αυτά τα βήματα η διαδικασία της ομαδοποίησης στην ενότητα 4.2.7 θα απαιτούσε τεράστιους υπολογιστικούς πόρους και οι ομάδες που προκύπτουν θα περιείχαν εγγραφές επηρεασμένες από θόρυβο. Η παρακάτω διαδικασία ακολουθήθηκε και για τις τρείς κατηγορίες δεδομένων (only_additions, only_deletions, both).

Αρχικά για το σύνολο των 23 χαρακτηριστικών υπολογίστηκε ο πίνακας αυτοσυσχέτισης (correlation matrix). Στη στατιστική, η αυτοσυσχέτιση ή η εξάρτηση είναι κάθε στατιστική σχέση, αιτιώδης ή μη, μεταξύ δύο τυχαίων μεταβλητών. Με την ευρύτερη έννοια η αυτοσυσχέτιση είναι οποιαδήποτε στατιστική συσχέτιση, αν και συνήθως αναφέρεται στον βαθμό στον οποίο ένα ζεύγος μεταβλητών συνδέονται γραμμικά. Ουσιαστικά, η αυτοσυσχέτιση είναι μια μετρική που περιγράφει τον τρόπο με τον οποίο δύο ή περισσότερες μεταβλητές σχετίζονται μεταξύ τους. Υπάρχουν αρκετοί συντελεστές αυτοσυσχέτισης (correlation coefficients), που συχνά συμβολίζονται ρ ή r, και μετρούν τον βαθμό συσχέτισης. Ο πιο συνηθισμένος από αυτούς είναι ο συντελεστής συσχέτισης Pearson[85], ο οποίος είναι ευαίσθητος μόνο σε μια γραμμική σχέση μεταξύ δύο μεταβλητών (η οποία μπορεί να υπάρχει ακόμη και όταν η μία μεταβλητή είναι μη γραμμική συνάρτηση της άλλης). Άλλοι συντελεστές συσχέτισης - όπως ο συσχετισμός κατάταξης Spearman - έχουν αναπτυχθεί για να είναι πιο ισχυροί από τον Pearson, δηλαδή πιο ευαίσθητοι σε μη γραμμικές σχέσεις[86, 87].

Στην προκειμένη περίπτωση υπολογίζεται ο Pearson συντελεστής αυτοσυσχέτισης δείγματος (Sample correlation coefficient) των χαρακτηριστικών. Δεδομένης μια σειράς από n μετρήσεις του ζεύγους (Xi, Yi), με δείκτη $i = 1, \dots, n$, ο Pearson συντελεστής αυτοσυσχέτισης δείγματος μπορεί να χρησιμοποιηθεί για την εκτίμηση του συντελεστή αυτοσυσχέτισης του πληθυσμού $\rho_{X,Y}$ μεταξύ του X και Y. Ο ορισμός φαίνεται παρακάτω:

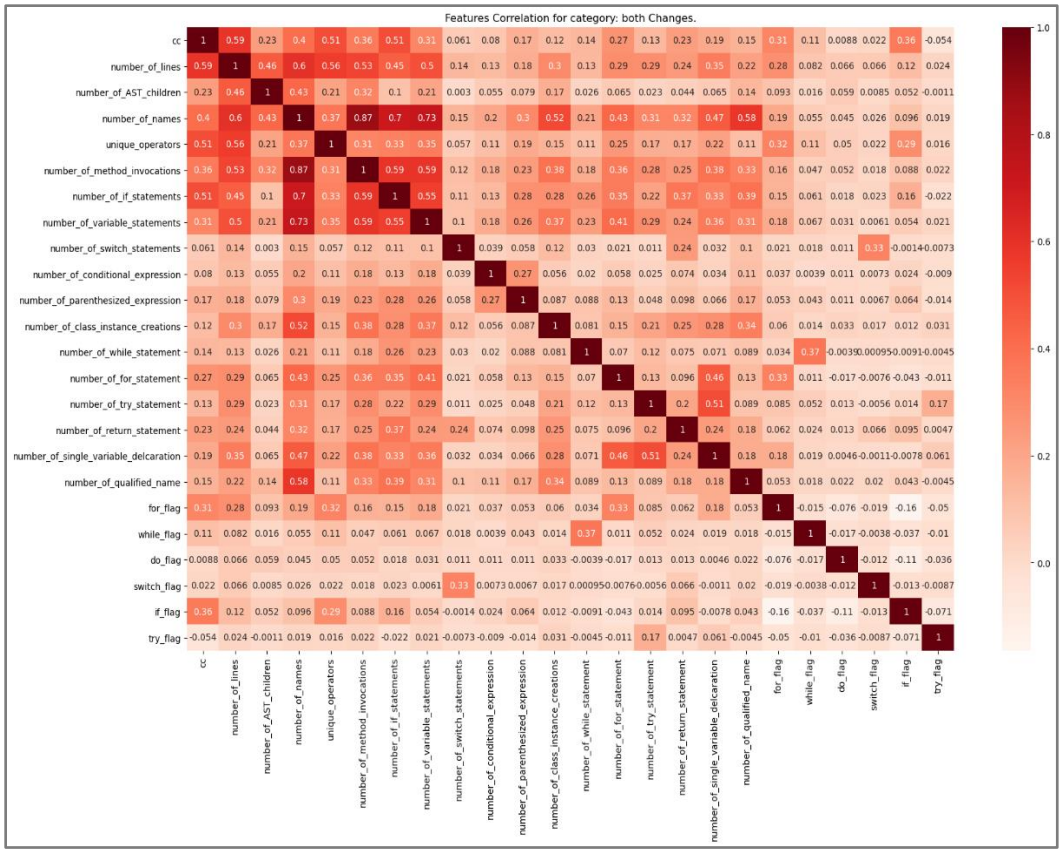
$$r_{X,Y} \stackrel{\text{def}}{=} \frac{\sum_1^n (x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} = \frac{\sum_1^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_1^n (x_i - \bar{x})^2 \ \sum_1^n (y_i - \bar{y})^2}} \quad (4.5)$$

όπου, $\bar{x}$ και $\bar{y}$ είναι οι δειγματικοί μέσοι όροι των X και Y, και s_x και s_y οι δειγματικές τους τυπικές αποκλίσεις. Οι τιμές της αυτοσυσχέτισης κυμαίνονται στις περισσότερες περιπτώσεις στο διάστημα [-1,1]. Αν δύο μεταβλητές εμφανίζουν τιμή αυτοσυσχέτισης ίση με -1, αυτό μας δείχνει μια τέλεια αρνητική συσχέτιση, ενώ αν η τιμή είναι 1 δείχνει μια τέλεια θετική συσχέτιση. Ένας συσχετισμός 0 δε δείχνει καμία γραμμική σχέση μεταξύ της κίνησης των δύο μεταβλητών. Οι περιπτώσεις αυτές εμφανίζονται στην Εικόνα 4.12.



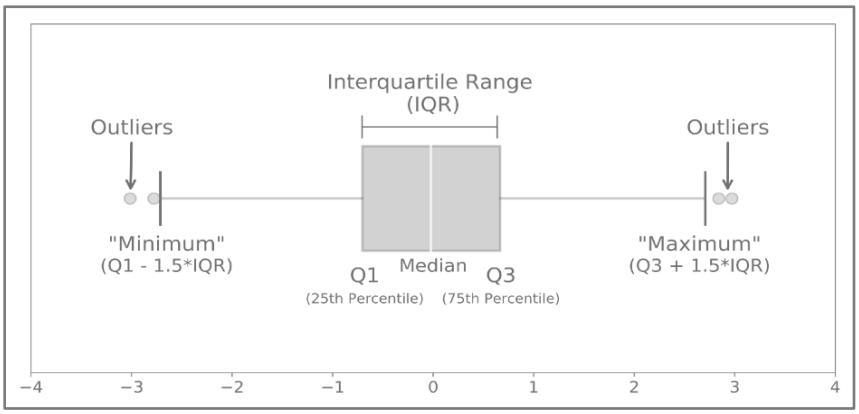
Εικόνα 4.12 Παραδείγματα Συσχέτισης δύο Μεταβλητών.

Στην Εικόνα 4.13 εμφανίζεται ο πίνακας αυτοσυσχέτισης του συνόλου των χαρακτηριστικών για την κατηγορία των both code changes. Ομοίως υπολογίστηκε και για τις υπόλοιπες δύο κατηγορίες και τα αποτελέσματα ήταν αρκετά όμοια. Όπως φαίνεται και στο σχήμα το χαρακτηριστικό των εγγραφών με όνομα number_of_method_invocations φαίνεται να εμφανίζει ισχυρή αυτοσυσχέτιση με μια πληθώρα διαφορετικών χαρακτηριστικών. Για τον λόγο αυτό διαγράφηκε από το σύνολο των χαρακτηριστικών και για τις τρείς κατηγορίες. Παρόλο που το ίδιο συμβαίνει και για το χαρακτηριστικό number_of_names κοιτώντας χειροκίνητα τα δεδομένα έγινε κατανοητό πως η παρουσία του στο σύνολο των δεδομένων ήταν απαραίτητη. Τα υπόλοιπα χαρακτηριστικά διατηρήθηκαν.



Εικόνα 4.13 Πίνακας Αυτοσυσχέτισης Χαρακτηριστικών.

Στη συνέχεια ακολούθησε ο σχεδιασμός των boxplots για κάθε ένα χαρακτηριστικό, ώστε να γίνει εξέταση της εμφάνισης περιπτώσεων που μπορούν να χαρακτηρισθούν ως outliers. Χρησιμοποιώντας λοιπόν τα boxplots υπολογίζονται και για τις τρείς κατηγορίες, η ελάχιστη τιμή, το 25ο percentile του δείγματος, η διάμεσος, το 75ο percentile και η μέγιστη τιμή. Η Εικόνα 4.14 περιγράφει τα συστατικά μέρη ενός boxplot.

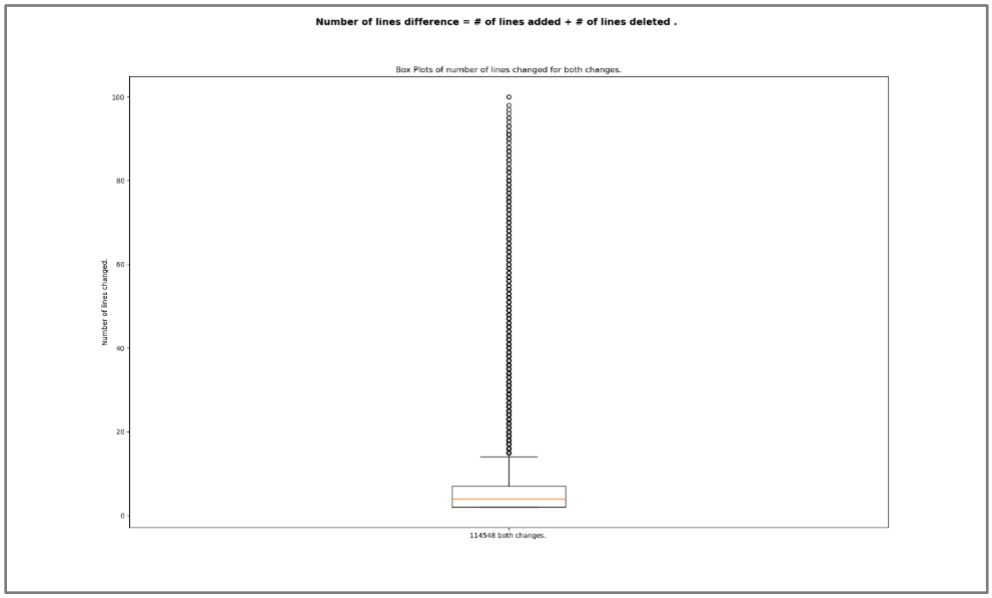


Εικόνα 4.14 Συστατικά Μέρη ενός Boxplot.

Πιο συγκεκριμένα:

- Median ($Q_2/50^{th}$ Percentile): Η διάμεσος του συνόλου δεδομένων.
- First quartile ($Q_1/25^{th}$Percentile): Ο μεσαίος αριθμός μεταξύ της μικρότερης τιμής (όχι το "minimum" όπως φαίνεται στο σχήμα) και της διαμέσου του συνόλου δεδομένων.
- Third quartile ($Q_3/75^{th}$ Percentile): Η μεσαία τιμή μεταξύ της διαμέσου και της υψηλότερης τιμής του συνόλου δεδομένων (όχι το "maximum" όπως εμφανίζεται στο σχήμα).
- Interquartile range (IQR): $Q_3 - Q_1$.
- Maximum: $Q_3 + 1.5IQR$.
- Minimum: $Q_1 - 1.5IQR$

Στην Εικόνα 4.15 παρουσιάζεται το boxplot της κατηγορίας των both code changes για το χαρακτηριστικό Number_of_lines_changed.



Εικόνα 4.15 Boxplot του Χαρακτηριστικού Numbers of Lines για την Κατηγορία των Both Code Changes.

4.2.6. Φιλτράρισμα Δεδομένων

Χρησιμοποιώντας λοιπόν όλη την πληροφορία που παράγεται στην προηγούμενη ενότητα οδηγούμαστε στο φιλτράρισμα των δεδομένων. Στόχο του βήματος αυτού αποτελεί το φιλτράρισμα των ακραίων περιπτώσεων, οι οποίες αποτρέπουν το σύστημα από το να παράγει γενικά και ποιοτικά αποτελέσματα. Έπειτα από μελέτη των στατιστικών δεδομένων που έχουν υπολογιστεί για τα χαρακτηριστικά, επιλέγονται μερικά όρια τιμών για συγκεκριμένες μεταβλητές. Πιο συγκεκριμένα τα πιο

ενδιαφέροντα δεδομένα παρουσιάζονται και για τις τρείς κατηγορίες στους πίνακες Πίνακας 4.6, Πίνακας 4.7 και Πίνακας 4.8.

Πίνακας 4.6 Στατιστικά Δεδομένα Χαρακτηριστικών των Both Code Changes.

Both	Mean	Std	Min	25%	50%	75%	Max
# Lines	6,2757	7,8715	2	2	4	7	**100**
# AST children	5,4471	6,7341	2	2	4	6	**297**
# Names	14,9569	28,7767	0	2	6	16	**2.175**
Cyclomatic Complexity	3,0015	2,3706	2	2	2	4	**104**
# Unique Operators	7,2116	3,2906	0	4	6	9	**24**
# If Statements	0,3968	1,5097	0	0	0	0	**106**
# Switch Statements	0,0054	0,0914	0	0	0	0	4
# While Statemets	0,0111	0,1499	0	0	0	0	**8**
# For Statemets	0,0722	0,4379	0	0	0	0	**26**
# Try Statements	0,0491	0,3234	0	0	0	0	**24**

Πίνακας 4.7 Στατιστικά Δεδομένα Χαρακτηριστικών των Only Additions Code Changes.

Only Additions	Mean	Std	Min	25%	50%	75%	Max
# Lines	3,5517	4,8400	1	1	2	4	**98**
# Names	10,4764	22,4401	0	3	5	10	**652**
# AST children	1,9673	2,8555	1	1	1	2	**68**
Cyclomatic Complexity	1,6464	1,3314	1	1	1	2	**35**
# Unique Operators	3,5556	2,0421	0	2	3	5	**12**
# If Statements	0,4824	1,2923	0	0	0	1	**55**
# Switch Statements	0,0024	0,0524	0	0	0	0	2
# While Statemets	0,0068	0,1011	0	0	0	0	4
# For Statemets	0,0544	0,3122	0	0	0	0	**9**
# Try Statements	0,0210	0,1883	0	0	0	0	**9**

Οι τονισμένες τιμές στους πίνακες, υποδεικνύουν τη μεγάλη διαφορά που υπάρχει για τα συγκεκριμένα χαρακτηριστικά, μεταξύ των ακραίων και των συνηθέστερων περιπτώσεων. Παρόλο που εμφανίζονται πολύ ακραίες περιπτώσεις και για τις τιμές των μεταβλητών “number of while statements”, “number of for statements” και “number of try statements” παρατηρείται πως οι τιμές τους διορθώνονται, καθώς φιλτράρονται οι ακραίες τιμές των χαρακτηριστικών “number of lines”, “number of AST children” και “number of names”.

Πίνακας 4.8 Στατιστικά Δεδομένα Χαρακτηριστικών των Only Deletions Code Changes.

Only Deletions	Mean	Std	Min	25%	50%	75%	Max
# Lines	3,0404	4,0396	1	1	2	3	**69**
# AST children	1,8467	2,3240	1	1	1	2	**52**
# Names	9,1043	20,1387	0	3	4	9	**942**
Cyclomatic Complexity	1,4514	1,0197	1	1	1	2	**15**

# Unique Operators	3,2725	1,9483	0	2	3	5	**12**
# If Statements	0,3419	1,0314	0	0	0	0	**46**
# Switch Statements	0,0018	0,0500	0	0	0	0	3
# While Statemets	0,0037	0,0707	0	0	0	0	3
# For Statemets	0,0477	0,3028	0	0	0	0	**13**
# Try Statements	0,0236	0,1902	0	0	0	0	4

Οι επιλογές που έγιναν βάση των παραπάνω δεδομένων και χειροκίνητου ελέγχου των αποτελεσμάτων είναι οι εξής :

- Για τα both code changes διαγράφονται οι εγγραφές που έχουν τιμές:
  - Number_of_lines > 16
  - Number_of_AST_children > 14
  - Number_of_names > 41
- Για τα only additions και τα only deletions code changes διαγράφονται οι εγγραφές με τιμές:
  - Number_of_lines > 8
  - Number_of_AST_children > 6
  - Number_of_names > 20

Από τις τιμές των πινάκων Πίνακας 4.9, Πίνακας 4.10 και Πίνακας 4.11 είναι εμφανές πως πλέον οι εγγραφές που αποτελούν το σύνολο των δεδομένων είναι ισορροπημένες. Το ίδιο ισχύει και για τα υπόλοιπα χαρακτηριστικά.

Πίνακας 4.9 Στατιστικά Δεδομένα Χαρακτηριστικών των Both Code Changes μετά το Φιλτράρισμα.

Both	Mean	Std	Min	25%	50%	75%	Max
# Lines	4,1802	2,9777	2	2	3	5	**16**
# AST children	3,9661	2,5627	2	2	3	5	**13**
# Names	8,1278	8,1991	0	2	5	11	**41**
Cyclomatic Complexity	2,6273	1,2689	2	2	2	3	**24**
# Unique Operators	6,6939	2,7912	0	4	6	8	**21**
# If Statements	0,1659	0,5077	0	0	0	0	**7**
# Switch Statements	0,0009	0,0350	0	0	0	0	2
# While Statemets	0,0031	0,0653	0	0	0	0	2
# For Statemets	0,0251	0,1986	0	0	0	0	**4**
# Try Statements	0,0215	0,1665	0	0	0	0	**4**

Πίνακας 4.10 Στατιστικά Δεδομένα Χαρακτηριστικών των Only Additions Code Changes μετά το Φιλτράρισμα.

Only Additions	Mean	Std	Min	25%	50%	75%	Max
# Lines	2,3255	1,7066	1	1	2	3	**8**
# AST children	1,4306	0,8870	1	1	1	2	**6**
# Names	5,4116	4,086	0	2	4	7	**20**
Cyclomatic Complexity	1,4136	0,7201	1	1	1	2	**9**

# Unique Operators	3,2191	1,7870	0	2	3	4	**12**
# If Statements	0,2897	0,5293	0	0	0	1	4
# Switch Statements	0,0004	0,0217	0	0	0	0	1
# While Statemets	0,0014	0,0387	0	0	0	0	2
# For Statemets	0,017	0,1352	0	0	0	0	2
# Try Statements	0,0056	0,074	0	0	0	0	1

Πίνακας 4.11 Στατιστικά Δεδομένα Χαρακτηριστικών των Only Deletions Code Changes μετά το Φιλτράρισμα.

Only Deletions	Mean	Std	Min	25%	50%	75%	Max
# Lines	2,0956	1,6137	1	1	1	3	**8**
# AST children	1,4155	0,8925	1	1	1	1	**6**
# Names	5,1251	3,9816	0	2	4	7	**20**
Cyclomatic Complexity	1,293	0,6244	1	1	1	1	**7**
# Unique Operators	2,9981	1,7099	0	2	3	4	**10**
# If Statements	0,2016	0,4504	0	0	0	0	4
# Switch Statements	0	0	0	0	0	0	0
# While Statemets	0,0011	0,0332	0	0	0	0	1
# For Statemets	0,0016	0,1302	0	0	0	0	2
# Try Statements	0,0071	0,0898	0	0	0	0	3

Τέλος, στον Πίνακα 4.12 παρουσιάζεται ο συνολικός αριθμός και το ποσοστό των εγγραφών που διαγράφηκε για κάθε κατηγορία αλλαγών.

Πίνακας 4.12 Συνολικός Αριθμός και Ποσοστό Φιλτραρισμένων Εγγραφών.

Filtering Category	Total Code Changes before Filtering	Total Code Changes Filtered	Percentage of Changes Filtered
Both	114.548	15.580	13,6%
Only_additions	29.385	4.099	13,95%
Only_deletions	9.173	1.052	11,46%

4.2.7. Αρχική Ομαδοποίηση

Η αρχική ομαδοποίηση σκοπεύει στη δομική διαχώριση του συνόλου δεδομένων, με σκοπό τη συγκέντρωση όμοιων code changes στις ίδιες ομάδες, καθώς και τη μείωση του αριθμού των συνολικών εγγραφών, ώστε να είναι δυνατή η τελική ομαδοποίηση για την εξαγωγή των τελικών προτύπων. Η ομαδοποίηση αυτή ολοκληρώνεται κάνοντας χρήση των τιμών των παραπάνω χαρακτηριστικών που υπολογίστηκαν για κάθε εγγραφή του φιλτραρισμένου συνόλου των δεδομένων. Σύμφωνα, όμως με τον αριθμό της διάστασης των χαρακτηριστικών, ο οποίος είναι έπειτα από την παραπάνω ανάλυση ίσος με 22 και για τις τρείς κατηγορίες, είναι απαραίτητος ο μετασχηματισμός των δεδομένων προκειμένου να ολοκληρωθεί με επιτυχία η αρχική ομαδοποίηση των δεδομένων.

4.2.7.1. Μείωσης της Διαστασιμότητας των Χαρακτηριστικών

Για τη μείωση της διαστασιμότητας όπως αναφέρθηκε και παραπάνω έγινε χρήση του αλγορίθμου PCA[36] της βιβλιοθήκης sklearn. Η ανάλυση κύριων συνιστωσών έχει στόχο την έυρεση ενός υποχώρου των δεδομένων, διάστασης $d < p$ (όπου p η διάσταση του χώρου των δεδομένων) που εξηγεί όσο το δυνατόν περισσότερο τη διασπορά των σημείων. Έστω λοιπόν $X = [x_1, x_2, \dots, x_n]^T$ το σύνολο των μεταβλητών με μέγεθος $n\ x\ p$ και $x_i \in R^p$. Η λειτουργία του αλγόριθμου περιγράφεται αναλυτικά στα παρακάτω βήματα. Αρχικά γίνεται επεξεργασία των δεδομένων σε δύο διακριτά βήματα και έπειτα ακολουθούν τα βήματα της διαδικασίας PCA:

- Κεντράρισμα των δεδομένων για κάθε συνιστώσα $j, j = 1, \dots, p$ και αντικατάσταση των τιμών $x_{i,j}$ από τις τιμές $x_{i,j} - \bar{x}_j$, όπου:

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \tag{4.6}$$

- Τυποποίηση των τιμών, δηλαδή εφόσον οι τιμές κεντραριστούν διαιρούνται με την τυπική τους απόκλιση σε κάθε συνιστώσα:

$$(x_{i,j} - \bar{x})/s_j \tag{4.7}$$

- Υπολογισμός του πίνακα διασπορών – συνδιασπορών, αν προηγηθεί τυποποίηση, τότε ο S είναι ο πίνακας συσχέτισης:

$$S = \frac{1}{n-1} X^T X \tag{4.8}$$

- Υπολογισμός των ιδιοτιμών $\lambda_1, \lambda_2, \dots, \lambda_p$του S, οι οποίες προκύπτουν από την επίλυση της εξίσωσης:

$$|S - \lambda I| = 0 \tag{4.9}$$

όπου $|x|$ είναι η ορίζουσα του πίνακα x και I ο μοναδιαίος πίνακας $p\ x\ P$.

- Υπολογισμός των ιδιοδιανυσμάτων $\alpha_1, \alpha_2, \dots, \alpha_p$ του S που προκύπτουν από την επίλυση των p εξισώσεων:

$$(S - \lambda_j I)\alpha_j = 0, j = 1, \dots, p \tag{4.10}$$

Με τη συνθήκη πως τα α_j είναι ορθοκανονικά, δηλαδή:

$$\alpha_i a_i^T = 1 \text{ και } \alpha_j a_i^T = 0 \tag{4.11}$$

Ισχύει, επίσης, $L = A^T S A$, όπου A πίνακας $p\ x\ p$ των ιδιοδιανυσμάτων και L διαγώνιος $p\ x\ p$ πίνακας των ιδιοτιμών. Οι ιδιοτιμές είναι διατεταγμένες σε φθίνουσα σειρά:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \tag{4.12}$$

[36] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

- Τα ιδιοδιανύσματα $\alpha_j,\ j = 1, \dots, p$ και οι κύριες συνιστώσες (principal components – PC) που ορίζουν τη νέα ορθοκανονική βάση (με περιστροφή των αξόνων)

$$y_j = \alpha_j^T x, j = 1, \dots, p \quad (4.13)$$

Και $\mathrm{x} = [x_1, x_2, \dots, x_p]$, $\mathrm{y} = [y_1, y_2, \dots, y_p]$ οι p αρχικές και νέες μεταβλητές. Τα στοιχεία του α_j εκφράζουν τις διευθύνσεις συνημίτονων που συνδέουν το αρχικό με το νέο σύστημα συντεταγμένων. Η κάθε νέα μεταβλητή y_j δίνεται ως γραμμικός συνδυασμός των αρχικών p μεταβλητών.

- Ο μετασχηματισμός των αρχικών παρατηρήσεων στο νέο σύστημα συντεταγμένων ορίζεται ως

$$Y = XA \quad (4.14)$$

όπου το Y περιλαμβάνει τα σκορ κυρίων συνιστωσών (PC Scores). Τα PC Scores είναι και αυτά κεντραρισμένα, όπως τα στοιχεία του X, και ασυσχέτιστα. Με αυτό τον τρόπο μπορούν να εκφραστούν οι αρχικές μεταβλητές ως προς τα PCs:

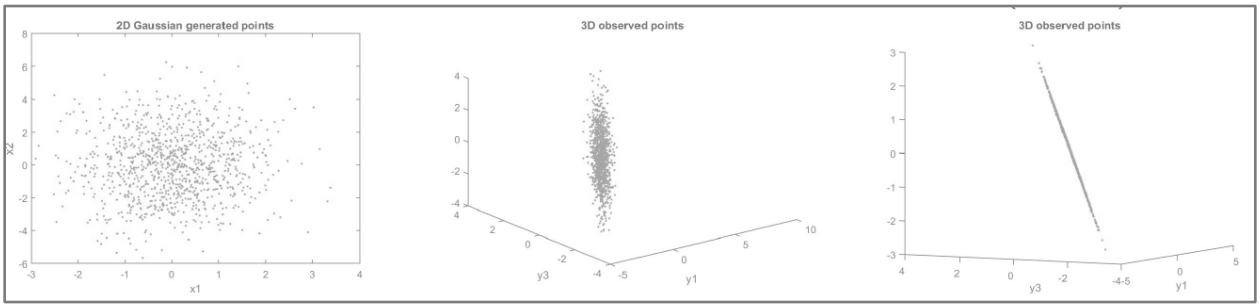
$$x = Ay \quad (4.15)$$

- Μόνο τα d PCs που αντιστοιχούν σε υψηλές ιδιοτιμές (εκφράζουν μεγάλο μέρος της διασποράς των δεδομένων) συμπεριλαμβάνονται για την παρακάτω ανάλυση και τα $p - d$ απορρίπτονται:

$$Y_d = XA_d \quad (4.16)$$

όπου το A_d περιλαμβάνει τα πρώτα d ιδιοδιανύσματα και έχει μέγεθος $p\ x\ d$, ενώ ο πίνακας Y_d έχει μέγεθος $n\ x\ d$.

Ένα παράδειγμα υλοποίησης του αλγόριθμου PCA, όπου μετασχηματίζονται δεδομένα τριών διαστάσεων σε δύο εμφανίζεται στην Εικόνα 4.16. Στο παρόν πρόβλημα, ο μετασχηματισμός των δεδομένων από τις 22 διαστάσεις σε 10, έπειτα από τη χρήση των κατάλληλων εργαλείων της python, υπολογίζεται πως καλύπτει το 95% της συνολικής πληροφορίας και για τις τρείς κατηγορίες.



Εικόνα 4.16 Παράδειγμα εφαρμογής του PCA σε Gaussian Generated Points.

4.2.7.2. Υλοποίηση της Αρχικής Ομαδοποίησης

Στη συνέχεια τα μετασχηματισμένα δεδομένα των 10 διαστάσεων δίνονται ως είσοδος στον αλγόριθμο k – means ++ για την ολοκλήρωση της αρχικής ομαδοποίησης. Αρχικά ο k-means [37] είναι ένας αλγόριθμος διαχωρισμού των δεδομένων σε ομάδες. Η κάθε ομάδα συνδέεται με ένα κεντρικό σημείο των δεδομένων αποκαλούμενο centroid. Για κάθε εγγραφή των δεδομένων ελέγχεται η απόσταση του σημείου της από τα κέντρα των ομάδων και αποδίδεται στην ομάδα με το πιο κοντινό κέντρο. Ο αριθμός των ομάδων, k, πρέπει να καθοριστεί. Στην Εικόνα 4.17 ορίζεται η διαδικασία του αλγορίθμου.

```
1: Select K points as the initial centroids.
2: repeat
3:     Form K clusters by assigning all points to the closest centroid.
4:     Recompute the centroid of each cluster.
5: until The centroids don't change
```

Εικόνα 4.17 Αλγόριθμος k-means.

Τα αρχικά κέντρα συνήθως επιλέγονται τυχαία. Βέβαια η επιλογή των αρχικών κέντρων παίζει καθοριστικό ρόλο στην ποιότητα των αποτελεσμάτων του αλγόριθμου. Για τον λόγο αυτό επιλέχθηκε ο k-means ++ στη συγκεκριμένη υλοποίηση και η διαφορά είναι πως, η επιλογή των αρχικών κέντρων γίνεται με τη χρήση της εξής διαδικασίας:

- Τυχαία επιλογή του αρχικού κέντρου από τα σημεία των δεδομένων.
- Για κάθε νέο σημείο p των δεδομένων που δεν έχει χρησιμοποιηθεί ακόμη, υπολογίζεται η Ευκλείδεια απόσταση $D(p, c)$, του p και του πλησιέστερου σημείου που έχει θεωρηθεί ήδη ως κέντρο c. Έστω ένα σημείο p με καρτεσιανές συντεταγμένες $(p_1, p_2, \dots, p_n)$, και ένα σημείο c που θεωρείται κέντρο με συντεταγμένες $(c_1, c_2, \dots, c_n)$, τότε:

$$D(p, c) = \sqrt{(c_1 - p_1)^2 + (c_2 - p_2)^2 + \cdots + (c_n - p_n)^2} \quad (4.17)$$

- Τυχαία επιλογή ενός νέου σημείου των δεδομένων ως νέο κέντρο και χρησιμοποιώντας μια σταθμισμένη κατανομή πιθανότητας (weighted probability distribution) επιλέγεται ως κέντρο το σημείο με πιθανότητα ανάλογη του $D(p, c)^2$.
- Επανάληψη των παραπάνω δύο βημάτων μέχρι να έχουν επιλεχθεί συνολικά k κέντρα.
- Στη συνέχεια ακολουθείται η υπόλοιπη διαδικασία του k-means.

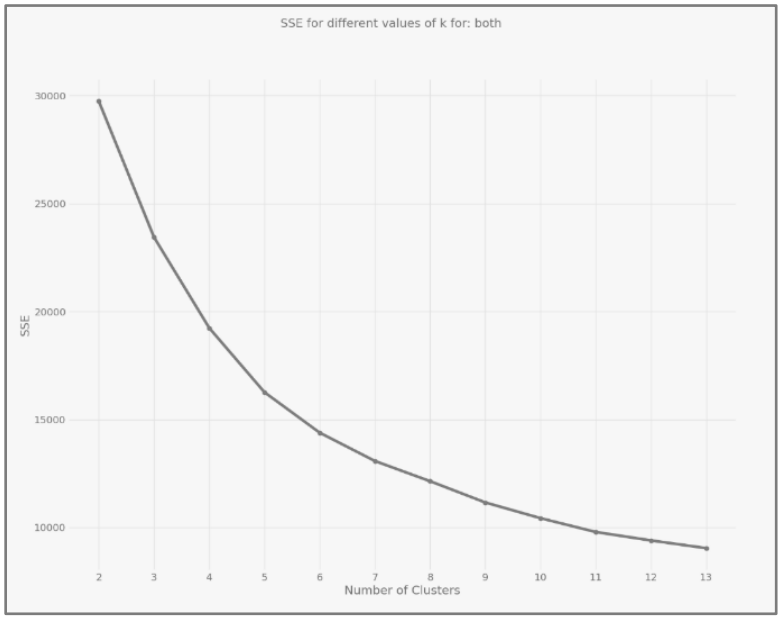
[37] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Για την επιλογή της κατάλληλης τιμής του αριθμού k, για τις τρείς διαφορετικές κατηγορίες, χρησιμοποιείται η μετρική του αθροίσματος των τετραγωνικών σφαλμάτων (Sum of Squared Errors – SSE). Για κάθε σημείο, το σφάλμα είναι η απόσταση του απο το κέντρο της ομάδας. Το SSE είναι το άθροισμα του τετραγώνου αυτών των σφαλμάτων:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} dist^2(m_i, x) \quad (4.18)$$

όπου x είναι ένα σημείο στην ομάδα C_iκαι m_i είναι το κέντρο της ομάδας αυτής.

Εφαρμόζοντας λοιπόν τα παραπάνω βήματα και ελέγχοντας τα αποτελέσματα υπολογισμού του SSE, επιλέγεται ο αριθμός k για τις τρείς κατηγορίες των δεδομένων. Στην Εικόνα 4.18 εμφανίζεται το διάγραμμα του SSE για την κατηγορία των both code changes.



Εικόνα 4.18 Διάγραμμα SSE για την Κατηγορία των Both Code Changes.

Αναλυτικότερα ο αριθμός των ομάδων καθώς και ο αριθμός των εγγραφών στο εσωτερικό της κάθε μίας και για τις τρείς διαφορετικές κατηγορίες περιγράφονται στους πίνακες Πίνακας 4.13 και Πίνακας 4.14 αντίστοιχα.

Πίνακας 4.13 Αριθμός Εγγραφών ανά Ομάδα της κάθε Κατηγορίας Δεδομένων.

Data Category	Number of Clusters								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Both	2.276	47.103	4.026	7.525	9.272	12.496	6.732	4.364	5.174
Only Additions	6.496	13.032	1.614	3.286	858	-	-	-	-

Only Deletions	325	1.466	532	4.672	1.126	-	-	-	-

Πίνακας 4.14 Επιλογή του Αριθμού k για Κάθε Κατηγορία Δεδομένων.

Data Category	Number of Clusters
Both	9
Only Additions	5
Only Deletions	5

4.2.8. Υπολογισμός Πινάκων Ομοιότητας

Στο στάδιο αυτό του συστήματος πραγματοποιείται ο υπολογισμός δύο πινάκων ομοιότητας για κάθε ομάδα που εμφανίζεται στον Πίνακα 4.13. Οι πίνακες αυτοί αφού συνδυαστούν, δίνονται ως είσοδος στον αλγόριθμο ομαδοποίησης της επόμενης ενότητας με σκοπό την παραγωγή των τελικών ομάδων (final clusters) που περιλαμβάνουν αλγοριθμικά και λεξιλογικά όμοια code changes, τα οποία χαρακτηρίζονται ως πρότυπα (Code Change Patterns – CCP). Γενικά, οι πίνακες αντικατοπτρίζουν την ομοιότητα της αλγοριθμικής δομής καθώς και τη λεξιλογική ομοιότητα του πηγαίου κώδικα των code changes.

4.2.8.1. Πίνακας Ομοιότητας Αλγοριθμικής Δομής των Code Changes

Αρχικά ο υπολογισμός του πίνακα ομοιότητας της αλγοριθμικής δομής ακολουθεί ακριβώς την ίδια διαδικασία με την ενότητα 4.2.3.2. Πιο συγκεκριμένα, προσδιορίζονται τα προφίλ των ASTs κάθε εγγραφής κάθε ξεχωριστής ομάδας και ακολουθεί ο υπολογισμός του πίνακα χρησιμοποιώντας τη σχέση (4.4) αναδρομικά. Όπως και στην ενότητα 4.2.3.2, για την υλοποίηση χρησιμοποιήθηκαν τα χαρακτηριστικά code_additions_ASTs και code_deletions_ASTs. Για την κατηγορία των only additions code changes χρησιμοποιήθηκαν οι τιμές του code_additions_AST, για τα only deletions code changes οι τιμές του code_deletions_AST ενώ για την κατηγορία των both οι τιμές και των δύο. Στην τελευταία περίπτωση, υπολογίστηκαν δύο πίνακες ομοιότητας, ένας για κάθε χαρακτηριστικό ξεχωριστά, και στο τέλος συνδυάστηκαν χρησιμοποιώντας τον μέσο όρο των τιμών.

Βέβαια στην ενότητα αυτή, ο αριθμός των εγγραφών ήταν πολύ μεγαλύτερος από την ενότητα 4.2.3 και για τον λόγο αυτό, ο υπολογισμός των πινάκων απαιτεί πολύ περισσότερο χρόνο κατά τη διαδικασία εκτέλεσης, καθώς και αποθηκευτικό χώρο στη μνήμη RAM. Για τον λόγο αυτό χρησιμοποιήθηκαν οι εξής δυο τεχνικές για μείωση του απαιτούμενου χρόνου και χώρου υπολογισμού:

- Αρχικά υπολογίστηκε μόνο ο άνω τριγωνικός πίνακας AST_T, εφόσον ο τελικός πίνακας ομοιότητας AST_M των δεδομένων είναι συμμετρικός και προκύπτει από τη σχέση:

$$AST\_M = AST\_T + AST\_T^T \quad (4.20)$$

- Επίσης, η διαδικασία των συγκρίσεων των code changes πραγματοποιήθηκε παράλληλα. Πιο αναλυτικά , έγινε χρήση της βιβλιοθήκης multiprocessing[38] της python, με την οποία ορίζεται ένα πλήθος workers που αναλαμβάνουν να κάνουν τις συγκρίσεις παράλληλα. Ουσιαστικά κάθε worker αναλαμβάνει να υπολογίσει μία γραμμή του άνω τριγωνικού πίνακα και στη συνέχεια μόλις τελειώσει συνεχίζει σε επόμενη γραμμή που δεν έχει υπολογιστεί.

4.2.8.2. Πίνακας Λεξιλογικής Ομοιότητας των Code Changes

Παρομοίως με τον πίνακα ομοιότητας της αλγοριθμικής δομής και σε αυτή την περίπτωση για την υλοποίηση χρησιμοποιήθηκαν τα χαρακτηριστικά code_additions και code_deletions. Για την κατηγορία των only additions code changes χρησιμοποιήθηκαν οι τιμές του code_additions, για τα only deletions code changes οι τιμές του code_deletions ενώ για την κατηγορία των both οι τιμές και των δύο. Στην τελευταία περίπτωση πάλι, υπολογίστηκαν δύο πίνακες ομοιότητας, ένας για κάθε χαρακτηριστικό ξεχωριστά και στο τέλος συνδυάστηκαν χρησιμοποιώντας τον μέσο όρο των τιμών. Πριν την εφαρμογή του μοντέλου ομοιότητας TF-IDF στα δεδομένα των αλλαγών υπό μορφή κειμένου για την παραγωγή του πίνακα λεξιλογικής ομοιότητας (code additions/code deletions), είναι απαραίτητη η εκτέλεση μιας σειράς ενεργειών προκειμένου οι αλλαγές ως κείμενο να περιέχουν μονάχα ουσιώδη πληροφορία. Η διαδικασία αυτή αποτελεί μέρος του κλάδου της *επεξεργασίας φυσικής γλώσσας* και για την υλοποίηση της έγινε χρήση του εργαλείου NLTK[39] της python.

Πιο αναλυτικά, οι τιμές των μεταβλητών code_additions και code_deletions υφίστανται επεξεργασία με τα ακόλουθα βήματα:

- Αφαίρεση σημείων στίξης (punctuation removal).
- Διάσπαση του περιεχομένου των μεταβλητών σε επιμέρους λέξεις (tokens).
- Αντιμετώπιση περιπτώσεων ενωμένων λέξεων χωρίς κενά (camel case).
- Απαλοιφή λέξεων που ανήκουν στο σύνολο κοινών λέξεων (stop words) του nltk.
- Μετατροπή κεφαλαίων σε πεζά γράμματα (lowercase).

Ένα παράδειγμα εφαρμογής της παραπάνω διαδικασίας εμφανίζεται στον Πίνακα 4.15:

Πίνακας 4.15 Επεξεργασία Κειμένου των Code Additions - Code Deletions.

Βήματα Επεξεργασίας	Κείμενο Αλλαγής
Αρχική Κατάσταση	“If (isSorted()) { sortDescending();}”

[38] https://docs.python.org/3/library/multiprocessing.html
[39] https://www.nltk.org/

1ο Βήμα	“If isSorted sortDescending”
2ο Βήμα	[“if”, “isSorted”, “sortDescending”]
3ο Βήμα	[“If”, “is”, “Sorted”, “sort”, “Descending”]
4ο Βήμα	[“Sorted”, “sort”, “Descending”]
5ο Βήμα	[“sorted”, ”sort” ,”descending”]

Αρχικά, τα σημεία στίξης δεν περιλαμβάνουν κάποια σημαντική πληροφορία για τη συγκεκριμένη ανάλυση και η αφαίρεση τους είναι απαραίτητη για την ολοκλήρωση των παρακάτω βημάτων επεξεργασίας. Η διάσπαση του περιεχομένου των μεταβλητών σε επιμέρους λέξεις (tokenization) είναι αναγκαία για την εύρυθμη λειτουργία του TF-IDF μοντέλου. Στη συνέχεια, αντιμετωπίζονται οι περιπτώσεις λέξεων οι οποίες είναι γραμμένες σε CamelCase, δηλαδή περιπτώσεων όπου διαφορετικές λέξεις είναι ενωμένες και συνδέονται εναλλάσσοντας πεζά με κεφαλαία γράμματα. Σε αυτό το βήμα, οι λέξεις αυτές διασπώνται ώστε να προκύψουν ξεχωριστά tokens. Για παράδειγμα, μία λέξη η οποία έχει τη μορφή “isOpen” θα διασπαστεί στα επιμέρους tokens “is” και “Open”. Για την υλοποίηση της διαδικασίας αυτής χρησιμοποιούνται regular expressions[88], τα οποία αποτελούν μια ακολουθία συμβόλων που ορίζουν ένα μοτίβο αναζήτησης σε μια μεταβλητή. Πιο συγκεκριμένα έγινε χρήση του regular expression: **'.+?(?:(?<=[a-z])(?=[A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|$)'**.

Επόμενο βήμα της διαδικασίας αποτελεί η απαλοιφή λέξεων που ανήκουν στο σύνολο των κοινών λέξεων που εμφανίζονται σε κάποιο κείμενο (stop words). Η λίστα των stop words που χρησιμοποιήθηκε στην προκειμένη περίπτωση εμφανίζεται στον Πίνακα 4.16 και αποτελεί μέρος της βιβλιοθήκης nltk.corpus όπου εμπεριέχονται οι πιο συνήθεις λέξεις της αγγλικής γλώσσας.

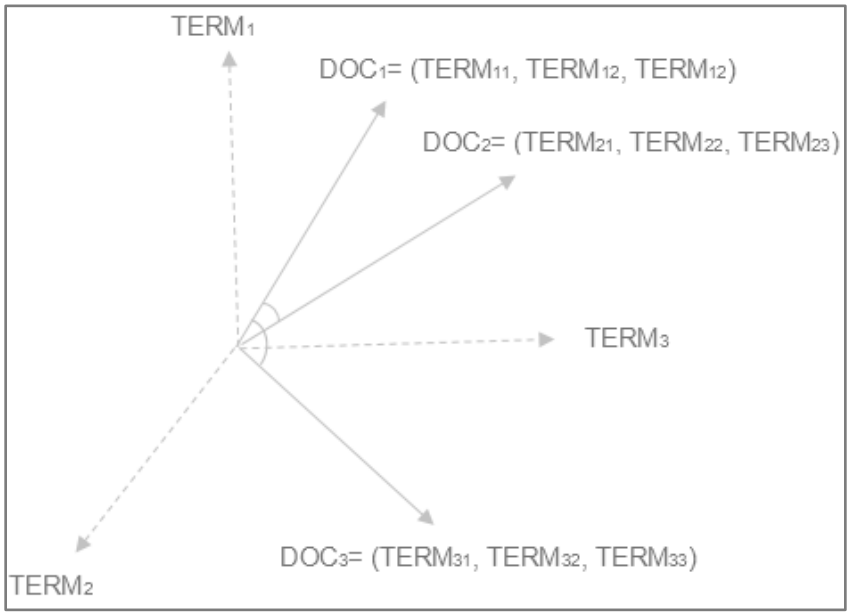
Πίνακας 4.16 Nltk English stopwords.

i	me	my	myself	we	our	ours	ourselves	you
you're	you've	you'll	you'd	your	yours	yourself	yourselves	he
him	his	himself	she	she's	her	hers	herself	it
it's	its	itself	they	them	their	theirs	themselves	what
which	who	whom	this	that	that'll	these	those	am
is	are	was	were	be	been	being	have	has
had	having	do	does	did	doing	a	an	the
and	but	if	or	because	as	until	while	of
at	by	for	with	about	against	between	into	through
during	before	after	above	below	to	from	up	down
in	out	on	off	over	under	again	further	then
once	here	there	when	where	why	how	all	any
both	each	few	more	most	other	some	such	no
nor	not	only	own	same	so	than	too	very
s	t	can	will	just	don	don't	should	should've

now	d	ll	m	o	re	ve	y	ain
aren	aren't	couldn	couldn't	didn	didn't	doesn	doesn't	hadn
hadn't	hasn	hasn't	haven	haven't	isn	isn't	ma	mightn
mightn't	mustn	mustn't	needn	needn't	shan	shan't	shouldn	shouldn't
wasn	wasn't	weren	weren't	won	won't	wouldn	wouldn't	ok

Βέβαια προκειμένου να υπάρχει η δυνατότητα εφαρμογής ενός αλγορίθμου ομοιότητας στο περιεχόμενο των αλλαγών σε μορφή κειμένου είναι απαραίτητη η μετατροπή τους σε μια αναπαράσταση η οποία θα καθιστά δυνατή τη σύγκριση αυτή. Μία από τις σημαντικότερες τεχνικές αναπαράστασης του κειμένου σε συγκρίσιμη μορφή είναι γνωστή ως Μοντελοποίηση Διανυσματικού Χώρου (Vector Space Modelling – VSM). Ουσιαστικά, η τεχνική αυτή ορίζει έναν πολυδιάστατο διανυσματικό χώρο με βάση το σύνολο των λέξεων (λεξιλόγιο) που απαντώνται σε ολόκληρο το εύρος των code changes. Στη συνέχεια, κάθε code change των ομάδων της ενότητας 4.2.7.2 αναπαρίσταται με ένα διάνυσμα στον πολυδιάστατό αυτό χώρο.

Η λογική πίσω από τη διανυσματική αναπαράσταση είναι πως οι αλλαγές κώδικα που βρίσκονται “κοντά” στον διανυσματικό χώρο περιγράφουν παρόμοια ζητήματα. Κάθε όρος του διανύσματος αυτού αντικατοπτρίζει και το πόσο σημαντική είναι η κάθε λέξη για το εκάστοτε code change. Η λειτουργία του VSM απεικονίζεται και στην Εικόνα 4.19.



Εικόνα 4.19 3D Απεικόνιση ενός Vector Space Model.

Η τεχνική VSM που χρησιμοποιήθηκε για τους σκοπούς της υλοποίησης βασίζεται στην καταμέτρηση συχνότητας όρων (term frequency counting). Ωστόσο, απορρίφθηκε η χρήση τεχνικών απόλυτης

καταμέτρησης συχνότητας όρων καθώς μπορεί να αποδώσουν εσφαλμένα μεγάλη σημασία σε μη ουσιώδεις όρους. Πιο συγκεκριμένα, επιλέχθηκε η χρήση ενός μοντέλου TF-IDF (Term Frequency-Inverse Document Frequency). Πιο συγκεκριμένα, το σκορ TF-IDF ενός όρου/λέξης (w) για ένα code change (cc) αυξάνεται με βάση τον αριθμό των φορών που εμφανίζεται ο εκάστοτε όρος στο code change, ενώ αντισταθμίζεται με βάση τον αριθμό των code changes που περιέχουν τον όρο αυτόν.

Αναλυτικότερα, το σκορ υπολογίζεται από την παρακάτω μαθηματική σχέση:

$$TF.IDF(w, cc) = TF(w, cc) * IDF(w) = TF(w, cc) * \log\left(\frac{TC}{DF(w)}\right) \quad (4.21)$$

όπου $TF(w, cc)$ η συνάρτηση συχνότητας του όρου w στο code change cc, TC ο συνολικός αριθμός των code changes, $DF(w)$ το πλήθος των code changes που περιέχουν τον όρο w έστω $cc_1 \leq TC$ και $IDF(w)$ η αντίστροφη συχνότητα εγγράφων, η οποία ουσιαστικά είναι ένα μέτρο της πληροφορίας που παρέχει μία λέξη επί του συνόλου όλων των code changes. Για παράδειγμα, για πολύ γενικές λέξεις που εμφανίζονται σχεδόν σε όλα τα μηνύματα υπολογίζεται $IDF(w) = 0 \Rightarrow TF.IDF(w, cc) = 0$, δηλαδή τους αποδίδεται μικρή σημασία, καθώς δεν περιέχουν ουσιώδη πληροφορία. Επιπλέον αναφέρεται πως υπάρχουν διάφορες μορφές συναρτήσεων συχνότητας όρων TF. Στη συγκεκριμένη περίπτωση, η συνάρτηση που επιλέχθηκε είναι η τετραγωνική ρίζα της συχνότητας εμφάνισης ενός όρου, δηλαδή:

$$TF(w, cc) = \sqrt{tf} \quad (4.22)$$

Για την καλύτερη κατανόηση της μεθόδου παραδίδεται ένα σύντομο παράδειγμα λειτουργίας της για τρία μικρά μηνύματα που παρουσιάζονται στον Πίνακα 4.17 από το σύνολο δεδομένων:

Πίνακας 4.17 Παράδειγμα εφαρμογής TF-IDF.

Αρχικό Μήνυμα	Tokens Πριν την Εφαρμογή TF_IDF
M1 = 'fixed bug with mute()'	['fix', 'bug', 'mute']
M2 = 'shutdown bug fix'	['shutdown', 'bug', 'fix']
M3 = 'fixed misnamed setter'	['fix', 'misnamed', 'setter']

Στον Πίνακα 4.18 αναγράφονται οι σχετικές συχνότητες όρων ενώ παρακάτω γίνεται ο αναλυτικός υπολογισμός του σκορ TF-IDF για τη λέξη *bug* και για τα τρία μηνύματα.

Πίνακας 4.18 Συχνότητες όρων (term frequencies) του παραδείγματος.

Μήνυμα	fix	bug	mute	shutdown	misnamed	setter
M1	TF(fix, M1) = 1	1	1	0	0	0
M2	1	1	0	1	0	0
M3	1	0	0	0	1	1

$$TF.IDF(bug, M1) = TF(bug, M1) * IDF(bug) = TF(bug, M1) * \log\left(\frac{|M|}{DF(bug)}\right) = 1 * \log\left(\frac{3}{2}\right) = 0{,}17$$

$$TF.IDF(bug, M2) = TF(bug, M2) * IDF(bug) = TF(bug, M2) * \log\left(\frac{|M|}{DF(bug)}\right) = 1 * \log\left(\frac{3}{2}\right) = 0{,}17$$

$$TF.IDF(bug, M3) = TF(bug, M3) * IDF(bug) = TF(bug, M3) * \log\left(\frac{|M|}{DF(bug)}\right) = 0 * \log\left(\frac{3}{2}\right) = 0$$

Για τους σκοπούς της εργασίας το μοντέλο TF-IDF υλοποιήθηκε μέσω της χρήσης της συνάρτησης TfidfVectorizer() της βιβλιοθήκης scikit-learn. Η συνάρτηση αυτή για κάθε code change που δέχεται σαν όρισμα επιστρέφει και ένα σχετικό διάνυσμα, με βάση κάποιες παραμέτρους. Πιο συγκεκριμένα, οι παράμετροι αυτοί είναι:

- tokenizer: Ορίζει τη συνάρτηση διάσπασης του κειμένου σε μεμονωμένες λέξεις (tokens). Στη δική μας υλοποίηση είναι η συνάρτηση επεξεργασίας των code changes που αναφέρθηκε παραπάνω.
- max_df (maximum document frequency): Ορίζει το ανώτατο όριο εμφάνισης (σε ποσοστό) ενός όρου στο σύνολο δεδομένων. Στη δική μας υλοποίηση, επιλέχθηκε 0.5, δηλαδή λέξεις που εμφανίζονται σε περισσότερο από το 50% των μηνυμάτων αγνοούνται ως πολύ γενικές.
- min_df (maximum document frequency): Ορίζει το κατώτατο όριο εμφάνισης (σε αριθμό code changes) ενός όρου στο σύνολο δεδομένων. Στη δική μας υλοποίηση, επιλέχθηκε 10, δηλαδή λέξεις που εμφανίζονται σε λιγότερα από 10 code changes αγνοούνται ως πολύ ειδικές.
- lowercase: Λογική τιμή που καθορίζει εάν θα γίνει μετατροπή των κεφαλαίων γραμμάτων σε πεζά. Στη συγκεκριμένη υλοποίηση, χρησιμοποιείται η τιμή True.

Μετά την ολοκλήρωση της διαδικασίας εκτέλεσης του μοντέλου, οι μεταβλητές code additions και code deletions των code changes έχουν πλέον αναπαρασταθεί με ένα διανύσμα στον n-διάστατο χώρο και μπορούν επομένως να συγκριθούν ώστε να παραχθεί ο πίνακας λεξιλογικής ομοιότητας των code changes. Για τον υπολογισμό του πίνακα ομοιότητας χρησιμοποιείται η ομοιότητα συνημίτονου. Ο χώρος αυτός ορίζεται από το σύνολο των όρων/λέξεων που απαντώνται σε όλα τα code changes, δηλαδή κάθε διάσταση αντιπροσωπεύει και έναν όρο. Αντίστοιχα, κάθε διάνυσμα στον χώρο αυτόν αντιπροσωπεύει ένα μοναδικό code change και το σύνολο των όρων που αυτό περιέχει. Κάθε μη μηδενική διάσταση ενός διανύσματος αντιστοιχεί στην εμφάνιση του αντίστοιχου όρου μέσα στο code change, ενώ η τιμή της καθορίζεται από τη τιμή $TF.IDF(w, cc)$ (όπου, w ο σχετικός όρος και cc το σχετικό code change).

Μία μέθοδος υπολογισμού της απόστασης μεταξύ διανυσμάτων σε n-διάστατο χώρο που βρίσκει εφαρμογές σε περιπτώσεις αραιών αναπαραστάσεων (πολλές μηδενικές διαστάσεις) είναι και η ομοιότητα συνημίτονου. Η ομοιότητα συνημίτονου εφαρμόζεται κυρίως σε θετικό χώρο (η τιμή $TF.IDF(w, cc)$ δε λαμβάνει αρνητικές τιμές) και κυμαίνεται στο διάστημα [0,1], γεγονός που τη καθιστά ένα χρήσιμο εργαλείο μέτρησης ομοιότητας. Η ομοιότητα συνημίτονου επιλέχθηκε καθώς δε λαμβάνει υπόψη τις μηδενικές διαστάσεις ενός διανύσματος, με αποτέλεσμα τη σημαντική μείωση της πολυπλοκότητας υπολογισμών.

Αναφέρουμε επιπλέον πως, όπως δηλώνει και το όνομά της, υπολογίζει το συνημίτονο της γωνίας μεταξύ δύο διανυσμάτων, δηλαδή διανύσματα με τον ίδιο προσανατολισμό έχουν ομοιότητα 1, ενώ διανύσματα με κάθετους προσανατολισμούς έχουν ομοιότητα 0. Η ομοιότητα συνημίτονου μεταξύ δύο code changes (cc_1, cc_2) δίνεται από την παρακάτω αναλυτική σχέση υπολογισμού:

$$\cos.\mathrm{sim}(cc_1, cc_2) = \frac{(cc_1 \cdot cc_2)}{(||cc_1|| * ||cc_2||)} = \frac{\sum_{i=1}^{N}\{tfi(w_i, cc_1) * tfi(w_i, cc_2)\}}{\sqrt{\sum_{i=1}^{N} tfi^2(w_i, cc_1)}\sqrt{\sum_{i=1}^{N} tfi^2(w_i, cc_2)}} \quad (4.23)$$

όπου $tfi(w_i, cc)$ η τιμή TF-IDF για τη λέξη w και το μήνυμα cc, και N το συνολικό πλήθος διαστάσεων του χώρου.

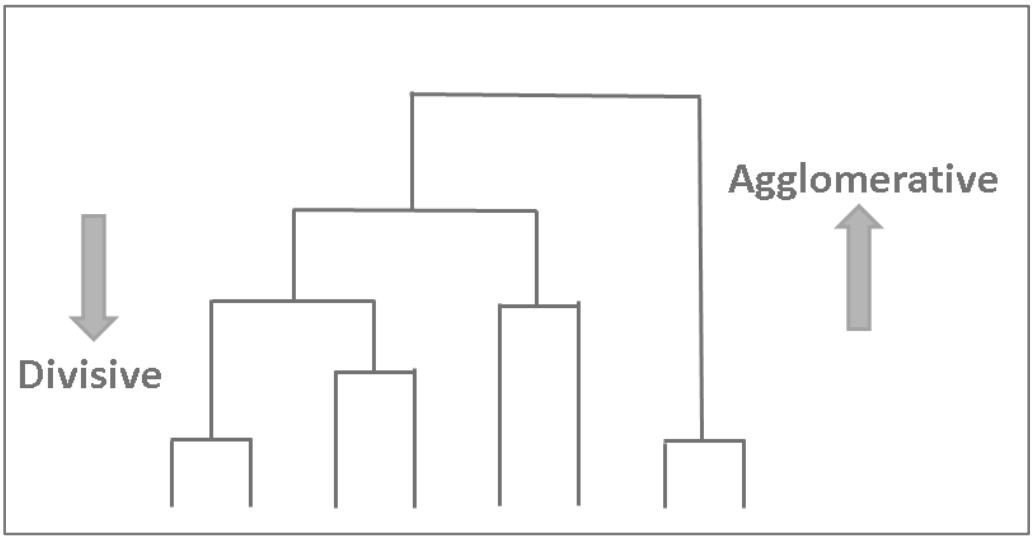
Για τους σκοπούς της υλοποίησης έγινε χρήση της συνάρτησης cosine_similarity[40] της βιβλιοθήκης scikit-learn. Ειδικότερα, η συνάρτηση αυτή δέχεται σαν είσοδο δύο διανύσματα των code changes σε μορφή $TF.IDF$ και επιστρέφει τη μεταξύ τους ομοιότητα συνημίτονου. Εναλλακτικά, μπορεί να δεχτεί ένα διάνυσμα εισόδου και μία λίστα με διανύσματα προς σύγκριση. Στην περίπτωση αυτή επιστρέφεται μία λίστα με όλες τις ομοιότητες συνημίτονου μεταξύ του διανύσματος και της λίστας που δόθηκαν ως ορίσματα. Κατά αυτό τον τρόπο έχουμε ως αποτέλεσμα έναν nxn συμμετρικό πίνακα ομοιότητας της λεξιλογικής δομής των code changes.

4.2.9. Ομαδοποίηση Αλλαγών

Στην ενότητα αυτή υλοποιείται η τελική ομαδοποίηση των αλλαγών για κάθε ξεχωριστή ομάδα των τριών κατηγοριών του συνόλου δεδομένων. Εφόσον για κάθε ομάδα έχουν υπολογισθεί οι δύο πίνακες ομοιότητας της παραπάνω ενότητας, είναι πλέον εφικτή η υλοποίηση του αλγορίθμου ομαδοποίησης. Για την τελική ομαδοποίηση έγινε χρήση του αλγορίθμου ιεραρχικής ομαδοποίησης[88] (Hierarchical Clustering).

[40] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

Η ιεραρχική ομαδοποίηση είναι μια μέθοδος ανάλυσης ομάδων που έχει ως στόχο τη δημιουργία μιας ιεραρχίας ομάδων. Γενικά υπάρχουν δύο κατηγορίες ιεραρχικής ομαδοποίησης οι οποίες απεικονίζονται στην Εικόνα 4.20.



Εικόνα 4.20 Δενδρόγραμμα Ιεραρχικής Ομαδοποίησης Συγχώνευσης και Διαχωρισμού.

Πιο συγκεκριμένα:

- Ιεραρχική Ομαδοποίηση Συγχώνευσης (Agglomerative): Αυτή η κατηγορία αποτελεί μια προσέγγιση από κάτω προς τα πάνω (bottom-up approach), δηλαδή αρχικά κάθε παρατήρηση του συνόλου δεδομένων αποτελεί ξεχωριστή ομάδα, και ζεύγη από ομάδες συγχωνεύονται όσο ο αλγόριθμος προχωρά προς υψηλότερα επίπεδα ιεραρχίας.
- Ιεραρχική Ομαδοποίηση Διαχωρισμού (Divisive): Αντίθετα, η κατηγορία αυτή είναι μια προσέγγιση από πάνω προς τα κάτω (top-down approach), δηλαδή αρχικά όλες οι παρατηρήσεις ανήκουν στην ίδια ομάδα και όσο ο αλγόριθμος προχωρά προς χαμηλότερα επίπεδα ιεραρχίας οι ομάδες διασπώνται.

Γενικά, οι διασπάσεις καθώς και οι συγχωνεύσεις των ομάδων στις δύο κατηγορίες καθορίζονται με έναν άπληστο τρόπο και τα αποτελέσματα του αλγορίθμου συνήθως παρουσιάζονται χρησιμοποιώντας ένα δενδρόγραμμα. Προκειμένου, λοιπόν ο αλγόριθμος να μπορεί να αποφασίσει ποιες ομάδες θα συγχωνευθούν (agglomerative) ή αν μια ομάδα πρέπει να διασπαστεί (divisive), είναι απαραίτητη η χρήση ενός μέτρου της ανομοιότητας μεταξύ των παρατηρήσεων των δεδομένων. Στις περισσότερες περιπτώσεις, αυτό πραγματοποιείται μέσω μιας μετρικής (ένα μέτρο της απόστασης κάθε ζεύγους

παρατηρήσεων) και ενός κριτηρίου σύνδεσης (linkage criterion) που καθορίζει την ανομοιότητα των συνόλων ως μια συνάρτηση των ζευγαρωτών αποστάσεων των παρατηρήσεων.

Οι παρακάτω μέθοδοι αποτελούν τις βασικές προσεγγίσεις που χρησιμοποιούνται για τον υπολογισμό της απόστασης μεταξύ δύο ομάδων a και b:

- Ευκλείδεια Απόσταση (Euclidean Distance):

$$||a - b||_2 = \sqrt{\sum_i^n (a_i - b_i)^2} \tag{4.24}$$

- Ευκλείδεια Τετραγωνική Απόσταση (Euclidean Squared Distance):

$$||a - b||_2^2 = \sum_i^n (a_i - b_i)^2 \tag{4.25}$$

- Απόσταση Μανχάταν (Manhattan Distance):

$$||a - b||_1 = \sum_i^n |a_i - b_i| \tag{4.26}$$

- Μέγιστη Απόσταση (Maximum Distance):

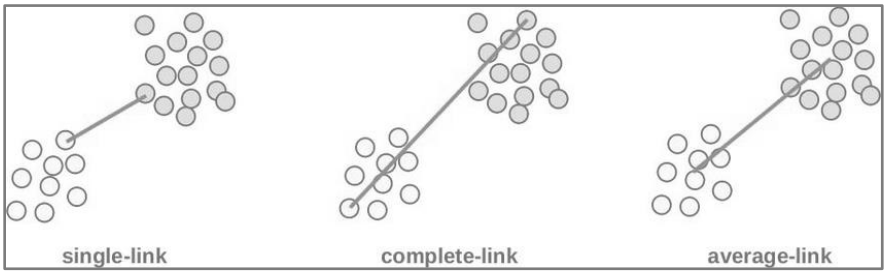
$$||a - b||_\infty = max_i |a_i - b_i| \tag{4.27}$$

- Απόσταση Mahalanobis (Mahalanobis Distance):

$$\sqrt{(a - b)^T S^{-1} (a - b)} \tag{4.28}$$

όπου S ο πίνακας συνδιακύμανσης.

Όσον αφορά τώρα τα κριτήρια σύνδεσης (linkage criterion) παρακάτω περιγράφονται μερικές από τις πιο συνηθισμένες περιπτώσεις που χρησιμοποιούνται μεταξύ δύο σετ παρατηρήσεων A και B και απεικονίζονται επίσης στην Εικόνα 4.21:



Εικόνα 4.21 Μέθοδοι Υπολογισμού Απόστασης Μεταξύ των Ομάδων.

- Maximum ή complete-linkage clustering:

$$\max\{d(a,b) : a \in A, b \in B\} \tag{4.29}$$

- Minimum ή single-linkage clustering:

$$\min\{d(a,b) : a \in A, b \in B\} \tag{4.30}$$

- Average linkage clustering:

$$\frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} d(a,b) \tag{4.31}$$

Στην υλοποίηση της παρούσας διπλωματικής ο ιεραρχικός αλγόριθμος ομαδοποίησης εφαρμόστηκε σε κάθε ομάδα ξεχωριστά, χρησιμοποιώντας τον Agglomerative Hierarchical Clustering με κριτήριο σύνδεσης Average Linkage. Βέβαια όπως αναφέρθηκε ο αλγόριθμος αυτός χρησιμοποιεί πίνακα απόστασης και όχι ομοιότητας. Για τον λόγο αυτό ήταν απαραίτητος ο συνδυασμός των πινάκων ομοιότητας της προηγούμενης ενότητας - του πίνακα ομοιότητας αλγοριθμικής δομής και της λεξιλογικής ομοιότητας - και ο υπολογισμός του συμπληρωματικού του πίνακα. Για τον συνδυασμό των πινάκων χρησιμοποιήθηκε η πράξη της πρόσθεσης (element-wise) και στη συνέχεια εφαρμόστηκε κλιμάκωση έτσι ώστε ο τελικός πίνακας απόστασης να είναι συμμετρικός και οι τιμές του να ανήκουν στο διάστημα [0,1].

Για την εύρεση του βέλτιστου αριθμού ομάδων χρησιμοποιήθηκε η μετρική Silhouette. Η Silhouette αποτελεί ένα μέτρο του κατά πόσο ένα αντικείμενο είναι όμοιο με την ομάδα στην οποία ανήκει (cohesion) σε σύγκριση με τις υπόλοιπες ομάδες (separation). Η μετρική αυτή ορίζεται ως προς ένα σημείο και παίρνει τιμές στο διάστημα [-1,1]. Για τη τιμή Silhouette ως προς ένα σημείο i, υπολογίζονται δύο παράμετροι:

- Η μέση απόσταση του σημείου από τα υπόλοιπα σημεία της ομάδας. Η τιμή αυτή δείχνει κατά πόσο ένα σημείο i έχει ανατεθεί σωστά στο συγκεκριμένο cluster C_i. Για τον υπολογισμό της μέσης απόστασης χρησιμοποιείται η σχέση (4.32).

$$a(i) = \frac{1}{|C_i| - 1} \sum_{\substack{j \in C_i \\ j \neq i}} d(i,j) \tag{4.32}$$

όπου $d(i,j)$ είναι η απόσταση μεταξύ των σημείων i και j στο cluster C_i.

- Η μικρότερη από τις μέσες αποστάσεις του σημείου i από όλα τα υπόλοιπα clusters C_k και υπολογίζεται με βάση τον παρακάτω τύπο:

$$b(i) = min \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j) \tag{4.33}$$

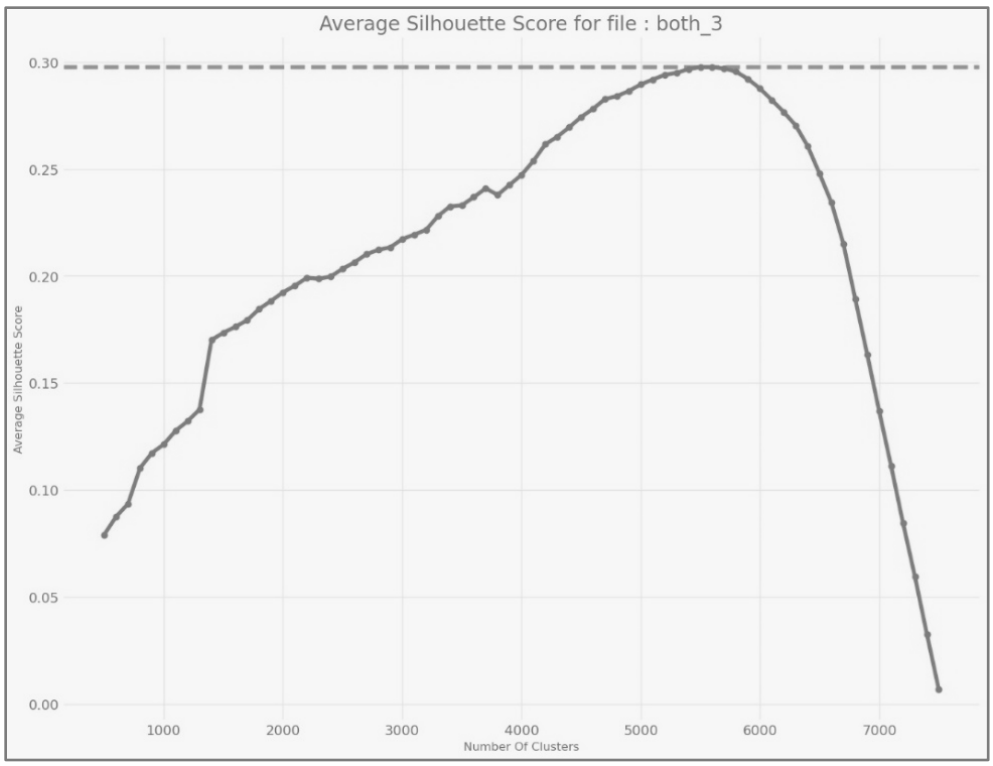
Με βάση αυτές τις δύο τιμές υπολογίζεται η τιμή της μετρικής Silhouette σύμφωνα με τη παρακάτω σχέση:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, if\ |C_i| > 1\ and\ s(i) = 0, if\ |C_i| = 1 \tag{4.34}$$

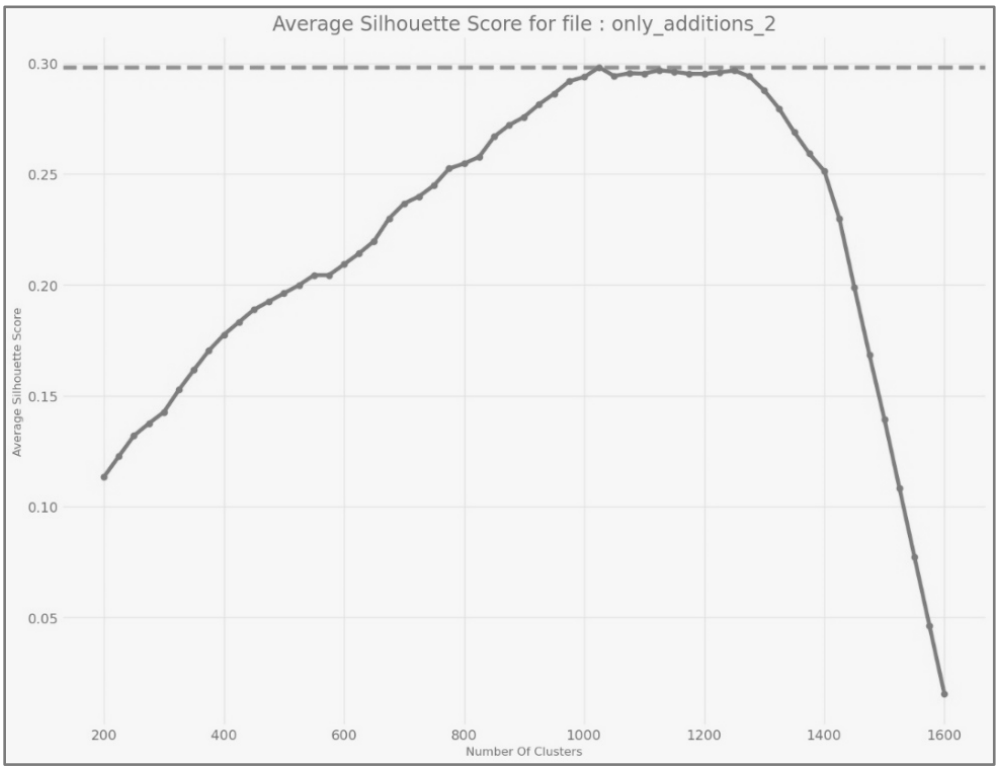
Από το τύπο αυτό είναι εμφανές πως η τιμή της μετρικής Silhouette κυμαίνεται στο διάστημα [-1,1]. Για τιμή της μετρικής silhouette κοντά στη μονάδα σημαίνει ότι το σημείο ανήκει στη σωστή ομάδα, ενώ αντίθετα όταν η τιμή αυτή είναι κοντά στο -1 το σημείο έπρεπε να ανήκει σε διαφορετικό cluster. Για την αξιολόγηση του συστήματος υπολογίσαμε το μέσο Silhouette $\bar{s}$ όλων των σημείων του συνόλου δεδομένων, δηλαδή των code changes στη συγκεκριμένη περίπτωση. Η διαδικασία της ομαδοποίησης πραγματοποιήθηκε επαναληπτικά για διαφορετικές τιμές του αριθμού ομάδων και τελικά επιλέχθηκε εκείνη με τη μεγαλύτερη μέση τιμή του Silhouette που αποτελεί και τη βέλτιστη τιμή του αριθμού των ομάδων.

$$k_{opt} = \{k: \max(\bar{s}(k))\} \tag{4.35}$$

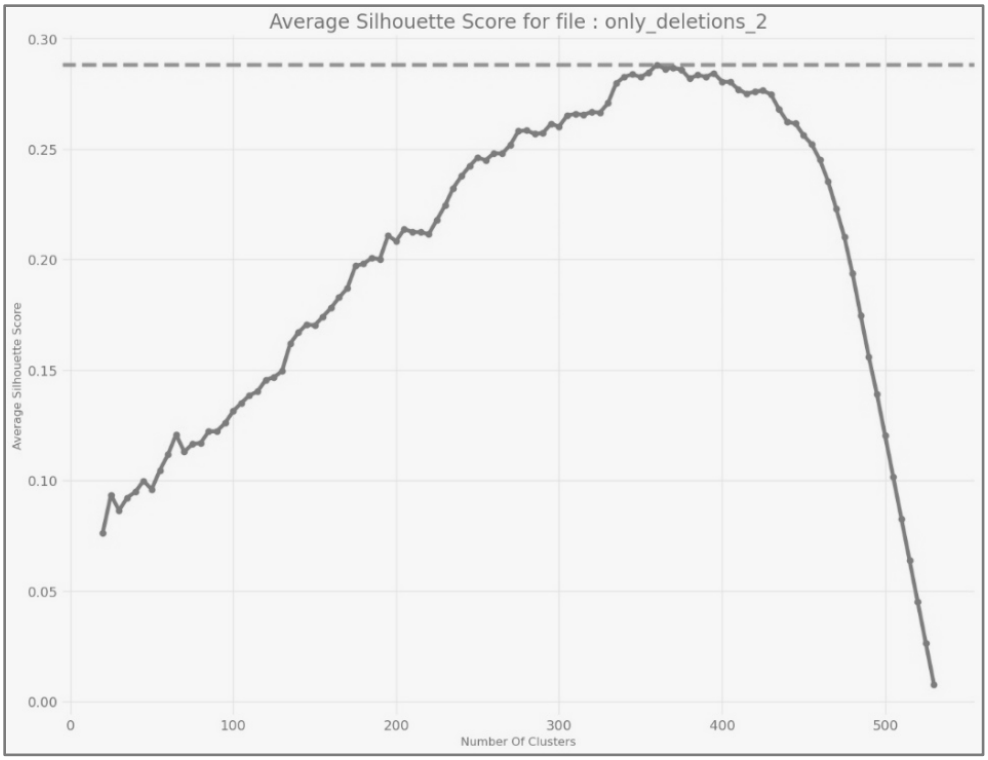
Παρακάτω, στις εικόνες Εικόνα 4.22, Εικόνα 4.23, Εικόνα 4.24 απεικονίζονται τα διαγράμματα καταγραφής της μέσης τιμής της μετρικής Silhouette με βάση τα οποία έγινε η επιλογή του k_{opt} για κάθε διαφορετική ομάδα που υπολογίστηκε στην ενότητα 4.2.7.2.. Για λόγους απλότητας εμφανίζεται ένα παράδειγμα από κάθε κατηγορία αλλαγών πηγαίου κώδικα.



Εικόνα 4.22 Μέση Τιμή της Μετρικής Silhouette για την Ομάδα both_3 της Κατηγορίας Both Code Changes.



Εικόνα 4.23 Μέση Τιμή της Μετρικής Silhouette για την Ομάδα only_additions_2 της Κατηγορίας Only Additions Code Changes.



Εικόνα 4.24 Μέση Τιμή της Μετρικής Silhouette για την Ομάδα only_deletions_2 της Κατηγορίας Only Deletions Code Changes.

4.2.10. Επιλογή Τελικών Ομάδων των Code Changes

Όπως αναφέρθηκε προηγουμένως η διαδικασία της τελικής ομαδοποίησης πραγματοποιήθηκε ξεχωριστά για κάθε ομάδα που είχε δημιουργηθεί κατά την ενότητα 4.2.7.2. Αυτό οδηγεί στη δημιουργία των τελικών ομάδων, στις οποίες περιέχονται όμοιες δομικά και λεξιλογικά αλλαγές κώδικα (code changes). Κρίνεται βέβαια απαραίτητο το φιλτράρισμα των τελικών ομάδων και η επιλογή μόνο αυτών οι οποίες ικανοποιούν κάποιες συγκεκριμένες συνθήκες. Λόγω της φύσης του ιεραρχικού αλγορίθμου ομαδοποίησης καθώς και του συνόλου δεδομένων που χρησιμοποιήθηκε, είναι λογικό οι τελικές ομάδες να περιλαμβάνουν περιπτώσεις που ο αριθμός των εγγραφών στο εσωτερικό τους, είναι πάρα πολύ μικρός. Αυτές οι περιπτώσεις δεν εμπεριέχουν σημαντική πληροφορία και ούτε μπορούν να χαρακτηριστούν οι εγγραφές τους, ως πρότυπα αλλαγών κώδικα, καθώς αυτά πρέπει να συναντώνται σε μια πληθώρα διαφορετικών έργων λογισμικού. Επιπλέον, δημιουργούνται συχνά ομάδες που περιέχουν μεγάλο πλήθος από code changes τα οποία προέρχονται από μικρό αριθμό διαφορετικών έργων λογισμικού. Αυτό συμβαίνει διότι υπάρχουν αποθετήρια κώδικα που χρησιμοποιούν πολλές φορές σχεδόν όμοια τμήματα κώδικα. Οι ομάδες αυτές δε βοηθούν στην εξαγωγή των προτύπων.

Ακόμα ένα χαρακτηριστικό που είναι σημαντικό για την επίλυση του προβλήματος είναι η συνοχή που παρουσιάζουν οι ομάδες. Για να θεωρηθεί ότι μια ομάδα αντιστοιχεί σε ένα πρότυπο αλλαγών

κώδικα είναι απαραίτητο να έχει συνεκτικότητα, δηλαδή η ομοιότητα μεταξύ των code changes να είναι υψηλή. Με βάση το σύνολο των παραπάνω παρατηρήσεων, η τελική επιλογή των ομάδων γίνεται με τη χρήση των παρακάτω παραμέτρων:

- Ορίζεται ένα κατώτατο όριο του αριθμού των code changes (code changes lower bound) που πρέπει να περιέχουν οι ομάδες.
- Ορίζεται ένα κατώτατο όριο του αριθμού των έργων λογισμικού (repos lower bound) από τα οποία προέρχονται οι εγγραφές της κάθε ομάδας. Κατά αυτό τον τρόπο εξασφαλίζεται η γενικότητα του προτύπου καθώς αποδεικνύεται πως χρησιμοποιείται ευρέως και δεν αποτελεί μεμονωμένη περίπτωση.
- Και τέλος, για τον έλεγχο της συνοχής των ομάδων υπολογίζεται η μέση ομοιότητα που παρουσιάζουν τα code changes της ομάδας από το centroid τους, τόσο αλγοριθμικά όσο και λεξιλογικά. Ο υπολογισμός αυτός υλοποιείται αρχικά με την εύρεση του centroid και στη συνέχεια με τον προσδιορισμό της συνοχής όπως περιγράφεται στις παρακάτω σχέσεις:

$$centroid = \min\left(\frac{1}{|C|-1}\right)\sum_{\substack{j\in C\\ j\neq i}} d(i,j) \tag{4.36}$$

$$cohesion = 1 - \frac{1}{|C|-1}\sum_{x\in C} d(x, centroid) \tag{4.37}$$

όπου αυτή τη φορά ο $d(i,j)$ αποτελεί τον τελικό πίνακα απόστασης καθώς το $d(x, centroid)$ είναι απόσταση μεταξύ κάθε εγγραφής της ομάδας από το centroid.

Οι τιμές που επιλέχθηκαν για τις παραπάνω τρεις κατηγορίες παραμέτρων διακρίνονται στον Πίνακα 4.19. Η επιλογή των παραμέτρων έγινε μέσω διαδικασίας δοκιμών διαφορετικών τιμών και σε κάθε περίπτωση οι παραγόμενες ομάδες ελέγχονται με βάση τον αριθμό εγγραφών τους, τη συνοχή τους και τον αριθμό των διαφορετικών αποθετηρίων λογισμικού.

Πίνακας 4.19 Τιμές των Παραμέτρων για την Επιλογή των Τελικών Ομάδων.

Precluster Name	Size Threshold	Cohesion Threshold	Repositories Threshold
only_additions_0	5	0,75	5
only_additions_1	10	0,75	7
only_additions_2	5	0,75	5
only_additions_3	10	0,75	7
only_additions_4	5	0,75	5
only_deletions_0	5	0,75	5
only_deletions_1	10	0,75	7
only_deletions_2	5	0,75	5

only_deletions_3	10	0,75	7
only_deletions_4	5	0,75	5
both_0	10	0,75	7
both_1	10	0,75	7
both_2	10	0,75	7
both_3	10	0,75	7
both_4	10	0,75	7
both_5	10	0,75	7
both_6	10	0,75	7
both_7	10	0,75	7
both_8	10	0,75	7

Κεφάλαιο 5. Πειράματα & Αποτελέσματα

5.1. Γενικά

Στο κεφάλαιο αυτό, περιγράφονται τα πειράματα που διεξάγονται κατά τον έλεγχο λειτουργίας του συστήματος καθώς και τα αποτελέσματα τους. Κύριος στόχος κατά την αξιολόγηση, ήταν η ανίχνευση των εξαγόμενων προτύπων αλλαγών κώδικα του συνόλου δεδομένων εκπαίδευσης, στα έργα λογισμικού από τα οποία δημιουργείται το σύνολο δεδομένων αξιολόγησης. Επιπρόσθετα, πέρα από την επαλήθευση της εμφάνισης των προτύπων σε μια πληθώρα έργων λογισμικού, έγινε και μια προσπάθεια πρότασης διορθώσεων σε νέα αρχεία πηγαίου κώδικα που εισάγονται στο σύστημα ως είσοδος. Στη συνέχεια, περιγράφεται πιο αναλυτικά η διαδικασία αξιολόγησης και παρουσιάζονται τα πρότυπα αλλαγών κώδικα και η λειτουργικότητα τους.

5.2. Αξιολόγηση Προτύπων Αλλαγών Κώδικα

Η διαδικασία αξιολόγησης των προτύπων αλλαγών κώδικα ολοκληρώθηκε για τις τρεις διαφορετικές κατηγορίες δεδομένων ξεχωριστά και αποτελείται από δύο διακριτά βήματα.

Κατά το πρώτο βήμα χρησιμοποιούνται τα centroids της κάθε ομάδας της ενότητας 4.2.10 για την εύρεση παρόμοιων αλλαγών στο σύνολο αξιολόγησης. Η εμφάνιση ενός προτύπου σε μια εγγραφή του συνόλου αξιολόγησης ορίζεται χρησιμοποιώντας την σχέση (4.1) της ενότητας 4.2.2. Επιπρόσθετα ορίζεται ένα κατώτατο όριο ομοιότητας το οποίο πρέπει να εμφανίζουν τα ζεύγη αλλαγών ώστε μια περίπτωση να χαρακτηριστεί ως *match.* Έπειτα από πειραματισμό με την τιμή του ορίου και ελέγχοντας τα παραγόμενα αποτελέσματα επιλέχθηκε η τιμή 0.8 και για τις τρεις κατηγορίες. Για τις κατηγορίες only additions και only deletions code changes υπολογίζεται η ομοιότητα των χαρακτηριστικών code additions και code deletions αντίστοιχα. Αντιθέτως για την κατηγορία both code changes υπολογίζεται η ομοιότητα και για τα δύο χαρακτηριστικά και χρησιμοποιείται ο μέσος όρος τους. Σε αυτό το βήμα γίνεται καταγραφή και αποθήκευση των δεικτών των εγγραφών του συνόλου αξιολόγησης και των centroid που συμμετέχουν στο match.

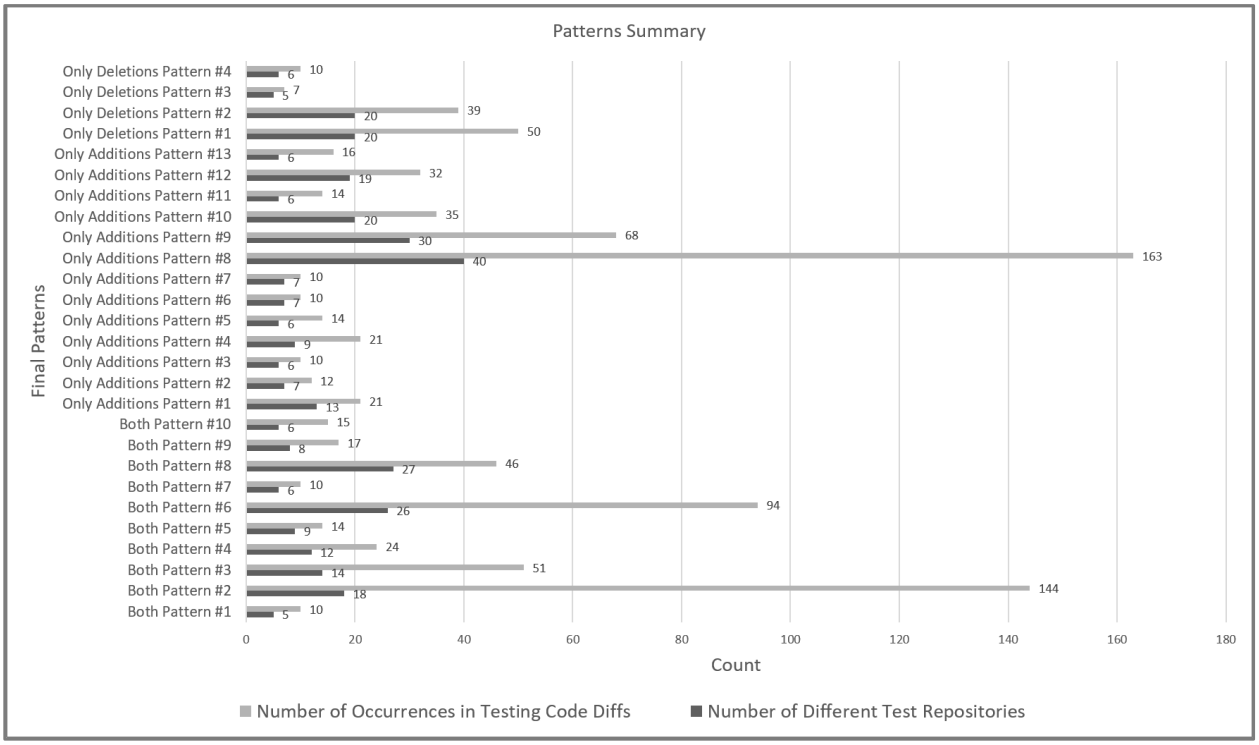
Στο επόμενο βήμα, χρησιμοποιώντας τους δείκτες του πρώτου βήματος, καταγράφονται οι παρακάτω μεταβλητές για κάθε match σε νέο αρχείο.

- Όνομα αρχείου του συνόλου αξιολόγησης που περιλαμβάνει την αλλαγή κώδικα που συμμετέχει στο match.

- Τον κώδικα διαφορών της αντίστοιχης αλλαγής του συνόλου αξιολόγησης, που παράγεται από την βιβλιοθήκη PyDriller.
- Τα code additions και code deletions τόσο για την εγγραφή του συνόλου αξιολόγησης όσο και του αντίστοιχου centroid.
- Τον αριθμό της ομάδας στην οποία ανήκει το συγκεκριμένο centroid.
- Το σκορ ομοιότητας που πέτυχε το συγκεκριμένο match.

5.3. Αποτελέσματα

Παρακάτω εμφανίζονται τα πιο σημαντικά αποτελέσματα που εξάγονται από το σύστημα της παρούσας διπλωματικής ανά κατηγορία αλλαγών πηγαίου κώδικα. Στην Εικόνα 5.1 παρουσιάζεται μια σύνοψη των τελικών αποτελεσμάτων που παρουσιάζονται στην επόμενη υποενότητα.



Εικόνα 5.1 Σύνοψη των Τελικών Αποτελεσμάτων.

5.3.1. Πρότυπα Αλλαγών της Κατηγορίας Both Code Changes

Στην ενότητα αυτή παρουσιάζονται τα πρότυπα αλλαγών πηγαίου κώδικα της κατηγορίας both code changes και περιγράφεται η λειτουργία τους καθώς και τα οφέλη που προσφέρουν όταν λαμβάνουν χώρα σε διάφορες μεθόδους. Επιπρόσθετα εμφανίζεται για το καθένα ο αριθμός των διαφορετικών

αποθετηρίων του συνόλου δεδομένων αξιολόγησης στα οποία εμφανίζεται το πρότυπο καθώς και ο αριθμός εμφάνισης τους στο περιεχόμενο των code diffs συνολικά.

5.3.1.1. Πρότυπο #1

Πίνακας 5.1 Πρότυπο #1 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions
Map<String, String> resultMap = new HashMap<String, String> ();	Map<String, String> resultMap = new LinkedHashMap<String, String> ();

Πίνακας 5.2 Χαρακτηριστικά του Προτύπου #1 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
5	10	both_1	334

Το LinkedHashMap είναι μια εναλλακτική κλάση του HashMap με μερικές πρόσθετες δυνατότητες. Η μεγαλύτερη διαφορά μεταξύ του HashMap και του LinkedHashMap είναι η ταξινόμηση των στοιχείων. Το LinkedHashMap παρέχει έναν τρόπο ταξινόμησης και ανίχνευσης των στοιχείων, ενώ το HashMap δεν υποστηρίζει αυτή τη λειτουργία. Ενώ το LinkedHashMap χρησιμοποιεί την ίδια εσωτερική εφαρμογή με το HashMap, χρησιμοποιεί επίσης μια διπλή σύνδεση σε όλες τις καταχωρήσεις του. Αυτή η συνδεδεμένη λίστα είναι χρήσιμη για την ταξινόμηση των στοιχείων. Κατά αυτό τον τρόπο η χρήση του LinkedHashMap παρέχει στον πηγαίο κώδικα μια πιο γρήγορη και αποτελεσματική δομή δεδομένων.

5.3.1.2. Πρότυπο #2

Πίνακας 5.3 Πρότυπο #2 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions
public String name () {	public String getName() {

Πίνακας 5.4 Χαρακτηριστικά του Προτύπου #2 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
18	144	both_1	1003

Οι μέθοδοι get και set είναι μέθοδοι που αποτελούν μέρη μιας κλάσης στον αντικειμενοστραφή προγραμματισμό. Είναι γενικά μια δημόσια (public) διεπαφή για την αλλαγή και τη λήψη των ιδιοτήτων μιας ιδιωτικής (private) κλάσης. Γενικότερα στη Java, οι μέθοδοι αυτοί ονομάζονται, καθολικά, χρησιμοποιώντας τις λέξεις get και set διότι κατά αυτό τον τρόπο το όνομα τους αντικατοπτρίζει πλήρως την λειτουργία τους. Επομένως το παραπάνω πρότυπο αποσκοπεί στη καλύτερη οργάνωση των μεθόδων μιας κλάσης και βοηθάει το έργο των προγραμματιστών όταν συγκεκριμένα έχουν να επεξεργαστούν και

να χρησιμοποιήσουν έναν τεράστιο αριθμό μεθόδων (getters & setters) κατά την υλοποίηση ενός έργου λογισμικού.

5.3.1.3. Πρότυπο #3

Πίνακας 5.5 Πρότυπο #3 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions
StringBuffer buffer = new StringBuffer ();	StringBuilder buffer = new StringBuilder ();

Πίνακας 5.6 Χαρακτηριστικά του Προτύπου #3 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
14	51	both_1	1899

Η Java παρέχει τρεις κλάσεις που αντιπροσωπεύουν μια ακολουθία χαρακτήρων: String, StringBuffer και StringBuilder. Η κλάση String είναι μια αμετάβλητη κλάση ενώ οι κλάσεις StringBuffer και StringBuilder είναι μεταβλητές. Η κύρια διαφορά μεταξύ StringBuffer και StringBuilder είναι πως η StringBuffer είναι συγχρονισμένη, δηλαδή δύο νήματα (threads) δεν μπορούν να καλέσουν τις μεθόδους της ταυτόχρονα, ενώ αντίθετα η StringBuilder υποστηρίζει multi-threading και ο λόγος αυτός την καθιστά πολύ πιο γρήγορη και αποδοτική.

5.3.1.4. Πρότυπο #4

Πίνακας 5.7 Πρότυπο #4 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions
ArrayList<Object> list = new ArrayList<> ();	List<Object> list = new ArrayList<>();

Πίνακας 5.8 Χαρακτηριστικά του Προτύπου #4 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
12	24	both_1	17208

Ο κύριος λόγος εμφάνισης του παραπάνω προτύπου, είναι η αποδέσμευση του κώδικά από μια συγκεκριμένη εφαρμογή της διεπαφής λίστας. Όταν χρησιμοποιείται η παραπάνω εντολή της στήλης code additions, ο υπόλοιπος κώδικας γνωρίζει μόνο ότι τα δεδομένα είναι τύπου List, το οποίο είναι προτιμότερο επειδή επιτρέπει στο μέλλον την εναλλαγή μεταξύ διαφορετικών εφαρμογών της διεπαφής λίστας με ευκολία. Ειδικά όταν πρόκειται για μεγάλα έργα λογισμικού η δυνατότητα αλλαγής της δομής στην οποία αποθηκεύονται τα δεδομένα χωρίς την ανάγκη αλλαγών στον υπάρχοντα κώδικα είναι πολύ σημαντικό προνόμιο.

5.3.1.5. Πρότυπο #5

Πίνακας 5.9 Πρότυπο #5 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions
} else {	} else if (value != null){

Πίνακας 5.10 Χαρακτηριστικά του Προτύπου #5 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
9	14	both_6	236

Στο συγκεκριμένο πρότυπο παρατηρείται η προσθήκη ενός κλάδου ελέγχου else if. Στο code addition του προτύπου προστίθεται ένας επιπλέον κόμβος ελέγχου, ο οποίος ελέγχει αν η τιμή μιας μεταβλητής είναι διάφορη της τιμής null, ώστε να προχωρήσει στις επόμενες εντολές οι οποίες βρίσκονται στο εσωτερικό του. Αντίθετα, στο code deletion ο κώδικας συνεχίζει κατευθείαν με την εκτέλεση των εντολών χωρίς τον έλεγχο της τιμής. Η χρήση του προτύπου αυτού οδηγεί στην αποφυγή σφαλμάτων τύπου “java.lang.NullPointerException”.

5.3.1.6. Πρότυπο #6

Πίνακας 5.11 Πρότυπο #6 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions
} catch (IOException e) {	} catch (Exception e) {

Πίνακας 5.12 Χαρακτηριστικά του Προτύπου #6 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
26	94	both_1	155

Γενικά οι προγραμματιστές κατά την υλοποίηση ενός έργου λογισμικού προσπαθούν συνεχώς να ελέγξουν περιπτώσεις κατά την εκτέλεση του κώδικα οι οποίες δημιουργούν εξαιρέσεις και μπορούν να οδηγήσουν σε κάποιο σφάλμα. Στην συγκεκριμένη περίπτωση, φαίνεται πως πολλές φορές είναι προτιμότερος ο έλεγχος πολλαπλών τύπων σφαλμάτων και όχι ενός συγκεκριμένου. Κατά αυτό τον τρόπο ο έλεγχος των εξαιρέσεων είναι γενικού περιεχομένου και θεωρείται πιο αποτελεσματικός.

5.3.1.7. Πρότυπο #7

Πίνακας 5.13 Πρότυπο #7 της Κατηγορίας Both Code Changes

Code Additions	Code Deletions
e.printStackTrace();	logger.error("", e);

Πίνακας 5.14 Χαρακτηριστικά του Προτύπου #7 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
6	10	both_1	1489

Με την κλήση της μεθόδου printStackTrace ένα σφάλμα εμφανίζεται στην οθόνη κατά την εκτέλεση. Αντιθέτως, η κλάση logger που αποτελεί ένα logging framework, επιτρέπει την ανακατεύθυνση του μηνύματος σφάλματος σε διαφορετικές διευθύνσεις, στην κονσόλα καθώς και σε ένα αρχείο. Ακολούθως, η εμφάνιση ενός σφάλματος στην διάρκεια εκτέλεσης ενός προγράμματος μπορεί να καταγραφεί, ώστε να χρησιμοποιηθεί αργότερα για έρευνα της αιτίας που προκάλεσε αρχικά το σφάλμα. Ουσιαστικά, το πρότυπο αυτό αποδεικνύει μια πιο οργανωμένη δομή καταγραφής σφαλμάτων σε ένα έργο λογισμικού.

5.3.1.8. Πρότυπο #8

Πίνακας 5.15 Πρότυπο #8 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions

out.close();	if (out!= null) {
	  out.close();
	}

Πίνακας 5.16 Χαρακτηριστικά του Προτύπου #8 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
27	46	both_6	57

Στο συγκεκριμένο πρότυπο παρατηρείται η προσθήκη ενός κλάδου ελέγχου if. Πιο συγκεκριμένα, στο code addition του προτύπου ελέγχεται αν μία μεταβλητή είναι διάφορη της τιμής null και σε αυτή την περίπτωση οδηγείται η μέθοδος στο κλείσιμο ενός αρχείου. Αντίθετα, προηγουμένως, το αρχείο έκλεινε και στην περίπτωση που η τιμή της μεταβλητής out ήταν κενή. Ομοίως με τη παράγραφο 5.3.1.5, το παράδειγμα αυτό έχει σκοπό την αποφυγή σφαλμάτων τύπου “java.lang.NullPointerException”.

5.3.1.9. Πρότυπο #9

Πίνακας 5.17 Πρότυπο #9 της Κατηγορίας Both Code Changes.

Code Deletions	Code Additions
@Test	@Test(timeOut = $CONSTANT)

Πίνακας 5.18 Χαρακτηριστικά του Προτύπου #9 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
8	17	both_1	1655

Σε αυτή την περίπτωση είναι εμφανές πως οι προγραμματιστές τείνουν να προσθέτουν ένα ανώτατο χρονικό όριο (timeOut = $CONSTANT) κατά την διαδικασία ελέγχου (testing) μιας μεθόδου. Όταν η εκτέλεση της μεθόδου ξεπεράσει αυτό το όριο εμφανίζεται στην οθόνη κατάλληλο μήνυμα, όπως για παράδειγμα: “java.lang.Exception: test timed out after $CONSTANT milliseconds”. Κατά αυτό τον τρόπο είναι πιο αποτελεσματικός και γρήγορος ο έλεγχος μιας μεθόδου, καθώς επιτρέπει τον ορισμό ενός χρονικού στόχου κατά το testing.

5.3.1.10. Πρότυπο #10

Πίνακας 5.19 Πρότυπο #10 της Κατηγορίας Both Code Changes.

Code Deletions	Code Deletions
TextView tv = (TextView) findViewById(R.id.aboutTextView);	TextView tv = findViewById(R.id.aboutTextView);

Πίνακας 5.20 Χαρακτηριστικά του Προτύπου #10 της Κατηγορίας Both Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
6	15	both_5	435

Το συγκεκριμένο πρότυπο αλλαγών, αποτρέπει την εμφάνιση μιας εξαίρεσης τύπου NullPointerException κατά την προσπάθεια ορισμού του κειμένου που θα πρέπει να εμφανίζεται στον χρήστη. Κατά την διαδικασία δημιουργίας ενός View σε ένα android σύστημα, η TextView δεν αποτελεί μέρος της ιεραρχίας των view ενός activity έως ότου κληθεί η εντολή setContentView(R.layout.main). Χρησιμοποιώντας λοιπόν την εντολή όπως εμφανίζεται στο code deletion, η διεπαφή TextView δεν έχει οριστεί ακόμα, οπότε ο εντολή displayTextView παίρνει την τιμή null και πολλές φορές οδηγεί σε σφάλμα.

5.3.2. Πρότυπα Αλλαγών της Κατηγορίας Only Additions Code Changes

Στην ενότητα αυτή παρουσιάζονται τα πρότυπα αλλαγών πηγαίου κώδικα της κατηγορίας only additions code changes και περιγράφεται η λειτουργία τους καθώς και ο λόγος που χρησιμοποιούνται σε διάφορες μεθόδους.

5.3.2.1. Πρότυπο #1

Πίνακας 5.21 Πρότυπο #1 της Κατηγορίας Only Additions Code Changes.

Code Additions
Thread.sleep($CONSTANT);

Πίνακας 5.22 Χαρακτηριστικά του Προτύπου #1 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
13	21	only_additions_1	331

Η μέθοδος Thread.sleep() της κλάσης Thread μπορεί να χρησιμοποιηθεί ώστε να σταματήσει την εκτέλεση του τρέχοντος thread για συγκεκριμένο χρόνο σε επίπεδο milliseconds. Το πρότυπο αυτό είναι ένα αποτελεσματικό μέσο κατά την εκτέλεση παράλληλων threads ώστε να υπάρχει διαθεσιμότητα χρόνου στον επεξεργαστή για την εκτέλεση των υπόλοιπων νημάτων μιας εφαρμογής ή άλλων εφαρμογών που ενδέχεται να εκτελούνται στο σύστημα ενός υπολογιστή.

5.3.2.2. Πρότυπο #2

Πίνακας 5.23 Πρότυπο #2 της Κατηγορίας Only Additions Code Changes.

Code Additions
file.deleteOnExit();

Πίνακας 5.24 Χαρακτηριστικά του Προτύπου #2 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
7	12	only_additions_1	9

Η μέθοδος java.io.File.deleteOnExit() διαγράφει το αρχείο ή τον κατάλογο που ορίζεται από το όνομα της αφηρημένης διαδρομής όταν ολοκληρώσει την λειτουργία της η εικονική μηχανή. Κατά κύριο λόγο η παραπάνω εντολή χρησιμοποιείται όταν κατά την διάρκεια εκτέλεσης ενός προγράμματος δημιουργούνται προσωρινά αρχεία, τα οποία βοηθούν στην ολοκλήρωση της ροής του πηγαίου κώδικα, αλλά η διατήρηση τους δεν είναι απαραίτητη μετά την ολοκλήρωση του.

5.3.2.3. Πρότυπο #3

Πίνακας 5.25 Πρότυπο #3 της Κατηγορίας Only Additions Code Changes.

Code Additions
method.setAccessible(true);

Πίνακας 5.26 Χαρακτηριστικά του Προτύπου #3 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
6	10	only_additions_1	600

Η μέθοδος setAccessible() της κλάσης Field ορίζει, για το αντικείμενο που χρησιμοποιείται κατά την κλήση της, την σημαία πρόσβασης (accessible flag) ίση με την αντίστοιχη boolean τιμή. Η τιμή true υποδεικνύει ότι, όταν χρησιμοποιείται το αντικείμενο αυτό, οι έλεγχοι πρόσβασης πρέπει να καταστέλλονται. Ουσιαστικά χρησιμοποιώντας το παραπάνω πρότυπο οι προγραμματιστές αποτρέπουν την εμφάνιση σφαλμάτων πρόσβασης σε μια μέθοδο κατά την διαδικασία ανάπτυξης ενός έργου.

5.3.2.4. Πρότυπο #4

Πίνακας 5.27 Πρότυπο #4 της Κατηγορίας Only Additions Code Changes.

Code Additions
e.printStackTrace();

Πίνακας 5.28 Χαρακτηριστικά του Προτύπου #4 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
9	21	only_additions_1	12

Η μέθοδος printStackTrace() της κλάσης Throwable εκτυπώνει το Throwable αντικείμενο και το backtrace του στην τυπική ροή σφαλμάτων. Το πρότυπο αυτό χρησιμοποιείται από τους προγραμματιστές για εύκολο και γρήγορο έλεγχο σφαλμάτων.

5.3.2.5. Πρότυπο #5

Πίνακας 5.29 Πρότυπο #5 της Κατηγορίας Only Additions Code Changes.

Code Additions

if ($Variable == null) {
  throw new NullPointerException("$Variable should not be null");
}

Πίνακας 5.30 Χαρακτηριστικά του Προτύπου #5 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
6	14	only_additions_0	142

Στο συγκεκριμένο πρότυπο, αν η τιμή μιας μεταβλητής είναι ίση με null, τότε καταγράφεται μια περίπτωση εξαίρεσης χρησιμοποιώντας την κλάση NullPointerException. Οι προγραμματιστές κάνουν συχνή χρήση του παραπάνω προτύπου ώστε να αποφευχθούν περιπτώσεις στις οποίες δημιουργούνται

σφάλματα όταν η τιμή μιας μεταβλητής είναι ίση με null. Μια εξαίρεση καταγράφεται κατά αυτό τον τρόπο όταν μια εφαρμογή προσπαθεί να χρησιμοποιήσει την τιμή null σε μια περίπτωση όπου απαιτείται ένα αντικείμενο.

5.3.2.6. Πρότυπο #6

Πίνακας 5.31 Πρότυπο #6 της Κατηγορίας Only Additions Code Changes.

Code Additions
long start = System.currentTimeMillis();

Πίνακας 5.32 Χαρακτηριστικά του Προτύπου #6 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
7	10	only_additions_1	889

Η μέθοδος currentTimeMillis() αποτελεί μέρος της κλάσης System. Επιστρέφει την τρέχουσα ώρα σε χιλιοστά του δευτερολέπτου. Η χρήση της εντολής αυτής επιλέγεται πολύ συχνά ώστε να αποθηκεύεται ο χρόνος έναρξης και λήξης της εκτέλεσης ενός προγράμματος ή μιας μεθόδου για την καταγραφή του συνολικού χρόνου εκτέλεσης τους.

5.3.2.7. Πρότυπο #7

Πίνακας 5.33 Πρότυπο #7 της Κατηγορίας Only Additions Code Changes.

Code Additions
} catch (IOException e) {   throw new RuntimeException(e);

Πίνακας 5.34 Χαρακτηριστικά του Προτύπου #7 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
7	10	only_additions_1	592

Στο συγκεκριμένο πρότυπο γίνεται χρήση των κλάσεων IOException και RuntimeException που αποτελούν υποκλάσεις της κλάσης Exception για την καταγραφή μιας ανεπιθύμητης εξαίρεσης. Η κλάση IOException σηματοδοτεί ότι έχει συμβεί κάποια εξαίρεση εισόδου/εξόδου. Αυτή η κατηγορία είναι η γενική κατηγορία εξαιρέσεων που δημιουργούνται από αποτυχημένες ή διακοπτόμενες λειτουργίες εισόδου/εξόδου. Οι προγραμματιστές χρησιμοποιούν πολύ συχνά το πρότυπο αυτό ώστε να αποφεύγουν σφάλματα τα οποία σχετίζονται με την ανάγνωση, την εγγραφή και την αναζήτηση ενός αρχείου ή μιας διεύθυνσης.

5.3.2.8. Πρότυπο #8

Πίνακας 5.35 Πρότυπο #8 της Κατηγορίας Only Additions Code Changes.

Code Additions
@SuppressWarnings("unchecked")

Πίνακας 5.36 Χαρακτηριστικά του Προτύπου #8 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
40	163	only_additions_1	653

Τα SuppressWarnings αποτελούν έναν τύπο Annotation της γλώσσας Java. Η εντολή αυτή υποδεικνύει ότι πρέπει να αγνοηθούν τα unchecked warnings του μεταγλωττιστή για το στοιχείο που ορίζεται χρησιμοποιώντας το συγκεκριμένο annotation (και για όλα τα στοιχεία προγράμματος που περιέχονται σε αυτό). Οι προγραμματιστές κατά την υλοποίηση μιας μεθόδου χρησιμοποιούν το παραπάνω πρότυπο για την αποφυγή εμφάνισης εξαιρέσεων που δεν ελέγχονται κατά την μεταγλώττιση του κώδικα. Αυτό συνηθίζεται όταν είναι σίγουρο πως η λειτουργία της μεθόδου είναι σωστή, αλλά για διάφορους λόγους ο μεταγλωττιστής θεωρεί πως η λογική του προγράμματος είναι εσφαλμένη κατά την εκτέλεση του.

5.3.2.9. Πρότυπο #9

Πίνακας 5.37 Πρότυπο #9 της Κατηγορίας Only Additions Code Changes.

Code Additions
@Override

Πίνακας 5.38 Χαρακτηριστικά του Προτύπου #9 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
30	68	only_additions_1	2732

Η εντολή @Override αποτελεί ένα ακόμα Annotation της γλώσσας προγραμματισμού Java. Ουσιαστικά η χρήση του παραπάνω προτύπου σκοπεύει στην ενημέρωση του μεταγλωττιστή πως το στοιχείο που ακολουθεί αντικαθιστά μια μέθοδο σε μια υποκλάση. Η χρήση του προτιμάται διότι ελέγχει αν υπάρχουν ορθογραφικά λάθη στο όνομα της νέας μεθόδου και αν οι παράμετροι τους ταιριάζουν. Επιπλέον κάνει τον πηγαίο κώδικα πιο κατανοητό σε περιπτώσεις αντικατάστασης μεθόδων.

5.3.2.10. Πρότυπο #10

Πίνακας 5.39 Πρότυπο #10 της Κατηγορίας Only Additions Code Changes.

Code Additions
this.$Variable = $Variable;

Πίνακας 5.40 Χαρακτηριστικά του Προτύπου #10 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
20	35	only_additions_1	513

Στο παρόν πρότυπο γίνεται χρήση του keyword this της γλώσσας προγραμματισμού Java, το οποίο αποτελεί τον πιο βασικό τρόπο κατασκευής ενός constructor αλλά και των αντίστοιχων μεθόδων μιας κλάσης. Σε μια μέθοδο ή έναν constructor η λέξη κλειδί this χρησιμοποιείται ως αναφορά στο τρέχον αντικείμενο. Ο κύριος λόγος χρήσης του παραπάνω προτύπου είναι η αποφυγή σύγχυσης μεταξύ των χαρακτηριστικών μιας κλάσης και των παραμέτρων της με το ίδιο όνομα.

5.3.2.11. Πρότυπο #11

Πίνακας 5.41 Πρότυπο #11 της Κατηγορίας Only Additions Code Changes.

Code Additions
logger.error("check error!", e);

Πίνακας 5.42 Χαρακτηριστικά του Προτύπου #11 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
6	14	only_additions_1	609

Το πρότυπο αυτό παρουσιάζει την καταγραφή ενός σφάλματος χρησιμοποιώντας τις κλάσεις Error και Logger. Η λειτουργία και ο λόγος προτίμησης της χρήσης του παραπάνω προτύπου από τους προγραμματιστές έχει ήδη περιγραφεί στο υποκεφάλαιο 5.3.1.7.

5.3.2.12. Πρότυπο #12

Πίνακας 5.43 Πρότυπο #12 της Κατηγορίας Only Additions Code Changes.

Code Additions
file.close();

Πίνακας 5.44 Χαρακτηριστικά του Προτύπου #12 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
19	32	only_additions_1	65

Η μέθοδος close() αποτελεί μέρος της κλάσης java.io.FileOutputStream. Η μέθοδος close κλείνει τη ροή εξόδου αρχείου και απελευθερώνει τους πόρους συστήματος που σχετίζονται με αυτήν τη ροή αφού έχει ολοκληρωθεί η καταχώρηση των δεδομένων. Η χρήση της εντολής αυτής συνίσταται ανεπιφύλακτα με σκοπό την απελευθέρωση πόρων του συστήματος, κυρίως της μνήμης, καθιστώντας πιο αποτελεσματική την εκτέλεση του.

5.3.2.13. Πρότυπο #13

Πίνακας 5.45 Πρότυπο #13 της Κατηγορίας Only Additions Code Changes.

Code Additions
mAdapter.notifyDataSetChanged();

Πίνακας 5.46 Χαρακτηριστικά του Προτύπου #13 της Κατηγορίας Only Additions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
6	16	only_additions_0	92

Η μέθοδος notifyDataSetChanged() αποτελεί μέρος της κλάσης BaseAdapter που χρησιμοποιείται στην ανάπτυξη λογισμικού συστημάτων android. Η μέθοδος notifyDataSetChanged() λοιπόν ενημερώνει τους χρήστες της εφαρμογής ότι τα υποκείμενα δεδομένα έχουν αλλάξει και κάθε προβολή (view) που αντικατοπτρίζει το σύνολο δεδομένων θα πρέπει να ανανεωθεί. Η λειτουργία της είναι πολύ χρήσιμη και χρησιμοποιείται συχνά από τους προγραμματιστές για τον ολοκληρωμένο σχεδιασμό μιας εφαρμογής, όσον αφορά την ενημέρωση των χρηστών για νέες αλλαγές των δεδομένων.

5.3.3. Πρότυπα Αλλαγών της Κατηγορίας Only Deletions Code Changes

Στην ενότητα αυτή παρουσιάζονται τα πρότυπα αλλαγών πηγαίου κώδικα της κατηγορίας only deletions code changes και περιγράφεται η λειτουργία τους καθώς και ο λόγος που χρησιμοποιούνται σε διάφορες μεθόδους. Γενικά, όπως φαίνεται και στην συνέχεια τα πρότυπα της κατηγορίας only deletions κυρίως αποτελούν εντολές εμφάνισης μηνυμάτων σφαλμάτων και εξαιρέσεων στην οθόνη. Είναι γνωστό πως η πλειοψηφία των προγραμματιστών κατά την ανάπτυξη λογισμικού χρησιμοποιούν τέτοιου είδους εντολές ώστε να ελέγχουν αν η λειτουργία του συστήματος είναι η επιθυμητή. Εφόσον είναι δεδομένο πως το σύστημα πληροί τις κατάλληλες προϋποθέσεις, συνήθως οι εντολές αυτές αφαιρούνται.

5.3.3.1. Πρότυπο #1

Πίνακας 5.47 Πρότυπο #1 της Κατηγορίας Only Deletions Code Changes.

Code Deletions
System.out.println();

Πίνακας 5.48 Χαρακτηριστικά του Προτύπου #1 της Κατηγορίας Only Deletions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
20	50	only_deletions_3	45

Η μέθοδος αυτή εκτυπώνει το κείμενο που δέχεται ως όρισμα στην κονσόλα και ο κέρσορας τοποθετείται στην αρχή της επόμενης γραμμής της κονσόλας. Η παραπάνω εντολή αποτελεί μια πολύ συχνή συνήθεια τον προγραμματιστών για την ευκολότερη αποσφαλμάτωση ενός πηγαίου κώδικα κατά την ανάπτυξη του. Εφόσον, το σύστημα έχει ολοκληρωθεί, αυτή η εντολή συνήθως διαγράφεται.

5.3.3.2. Πρότυπο #2

Πίνακας 5.49 Πρότυπο #2 της Κατηγορίας Only Deletions Code Changes

Code Deletions
e.printStackTrace();

Πίνακας 5.50 Χαρακτηριστικά του Προτύπου #2 της Κατηγορίας Only Deletions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
20	39	only_deletions_3	443

Το ίδιο πρότυπο εμφανίζεται και περιγράφεται στην υποενότητα 5.3.2.4.

5.3.3.3. Πρότυπο #3

Πίνακας 5.51 Πρότυπο #3 της Κατηγορίας Only Deletions Code Changes.

Code Deletions
} catch (Exception e) { throw e;

Πίνακας 5.52 Χαρακτηριστικά του Προτύπου #3 της Κατηγορίας Only Deletions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
5	7	only_deletions_3	246

Στην συγκεκριμένη περίπτωση χρησιμοποιείται η κλάση Exception για την καταγραφή εξαιρέσεων κατά την εκτέλεση ενός προγράμματος. Όπως αναφέρθηκε και προηγουμένως, αυτές οι εντολές αφαιρούνται, εφόσον έχει ελεγχθεί η λειτουργία του συστήματος και τα απαραίτητα κριτήρια καλύπτονται.

5.3.3.4. Πρότυπο #4

Πίνακας 5.53 Πρότυπο #4 της Κατηγορίας Only Deletions Code Changes

Code Deletions
long time = System.currentTimeMillis();

Πίνακας 5.54 Χαρακτηριστικά του Προτύπου #4 της Κατηγορίας Only Deletions Code Changes.

Number of Different Test Repositories	Number of Occurrences in Testing Code Diffs	Preclustering Group	Clustering Group
6	10	only_deletions_3	623

Ομοίως η περίπτωση αυτή έχει περιγραφεί στην υποενότητα 5.3.2.6. Συνήθως, όταν ολοκληρωθεί η διαδικασία ανάπτυξης και ελέγχου ενός πηγαίου κώδικα αφαιρούνται αυτού του είδους οι εντολές, διότι έχουν επιτευχθεί τόσο οι λειτουργικοί όσο και οι χρονικοί στόχοι της ομάδας ανάπτυξης λογισμικού.

Κεφάλαιο 6. Συμπεράσματα & Μελλοντική Εργασία

6.1. Συμπεράσματα

Η ανάλυση της παρούσας εργασίας καθώς και η διεθνής βιβλιογραφία, που χρησιμοποιήθηκε ως πηγή για τη διάρθρωση και την εξέλιξη του συγκεκριμένου συστήματος, οδηγεί σε κάποια πολύ σημαντικά συμπεράσματα. Αρχικά, έγινε απολύτως κατανοητό πως οι μηχανικοί λογισμικού αντιμετωπίζουν πληθώρα προβλημάτων κατά την διαδικασία ανάπτυξης ενός έργου λογισμικού. Τα τελευταία χρόνια, πηγή της βραδείας ανάπτυξης λογισμικού αποτελούσε η εμπειρική σύνταξη πηγαίου κώδικα, καθώς και η ελάχιστη προσπάθεια για δημιουργία επαναχρησιμοποιήσιμου λογισμικού. Επιπρόσθετα, συχνό φαινόμενο αποτελούσε η έλλειψη οργάνωσης και συντονισμού στον τρόπο λειτουργίας μιας ομάδας προγραμματιστών. Το γεγονός αυτό οδηγούσε επανειλημμένα, στην σπατάλη χρόνου για την δημιουργία, την αποσφαλμάτωση και την βελτίωση πηγαίου κώδικα, ο οποίος είχε περάσει απ’ όλα αυτά τα στάδια ανάπτυξης πολλαπλές φορές στο παρελθόν, είτε σε διαφορετικά έργα λογισμικού, είτε σε διαφορετικές ομάδες προγραμματιστών.

Η συνεχής εξέλιξη της επιστήμης των υπολογιστών και της τεχνολογίας λογισμικού έχει οδηγήσει τους μηχανικούς λογισμικού στην ανάγκη γρήγορης και αποδοτικής συγγραφής πηγαίου κώδικα. Τα συστήματα ελέγχου εκδόσεων και τα αποθετήρια ανοικτού λογισμικού αποτελούν τη βάση για τη λύση του παραπάνω ζητήματος, καθώς είναι τα κατάλληλα εργαλεία που προσφέρουν την δυνατότητα αποδοτικής συνεργασίας και οργάνωσης μεταξύ των προγραμματιστών. Επιπλέον, τα συστήματα αυτά περιέχουν τεράστιο όγκο πληροφορίας, ο οποίος θεωρείται πολύτιμος για την εξαγωγή προτύπων υλοποίησης διαφόρων έργων λογισμικού και μπορούν να οδηγήσουν σε συμπεράσματα που βοηθούν τους προγραμματιστές στη λήψη αποφάσεων κατά τη δημιουργία νέων έργων. Ακολούθως, η αποτελεσματική αξιοποίηση των παραπάνω δεδομένων βοηθά στην ανάπτυξη συστημάτων και μεθόδων με στόχο την εδραίωση επαναχρησιμοποιήσιμου λογισμικού.

Η υλοποίηση του παραπάνω συστήματος καλύπτει την ανάγκη εύρεσης και εκμετάλλευσης συχνών αλλαγών πηγαίου κώδικα που έχουν πραγματοποιηθεί στο παρελθόν σε μεγάλο αριθμό έργων λογισμικού, για την αποσφαλμάτωση και την βελτίωση του πηγαίου κώδικα. Χρησιμοποιώντας τα πρότυπα αυτά, οι προγραμματιστές μπορούν να αποφύγουν σφάλματα που έχουν συμβεί επανειλημμένα σε παρόμοιες υλοποιήσεις, αλλά και να λάβουν βοήθεια όσον αφορά την λήψη αποφάσεων κατά την διαδικασία ανάπτυξης νέων έργων. Επιπλέον, οι αλλαγές αυτές, οι οποίες

επιδιορθώνουν σφάλματα στον πηγαίο κώδικα, οδηγούν στην εδραίωση αποτελεσματικής και γρήγορης ανάπτυξης επαναχρησιμοποιήσιμου λογισμικού.

Η ανάπτυξη του συστήματος, βασίζεται στην συλλογή και την επεξεργασία όλων των τμημάτων πηγαίου κώδικα που περιλαμβάνονται στα commits των αποθετηρίων του συνόλου δεδομένων (Code Before, Code After, Code Additions, Code Deletions). Αφού αρχικά διατηρήθηκαν μόνο οι αλλαγές που περιλαμβάνονται στο εσωτερικό μεθόδων, ολοκληρώθηκε η διαδικασία εξαγωγής χαρακτηριστικών, από τα παραπάνω τμήματα των commits, και ακολούθησε η ανάλυση τους για την υλοποίηση ενός φίλτρου καθαρισμού διπλοεγγραφών αλλά και ακραίων τιμών.

Στη συνέχεια, πραγματοποιείται η εξαγωγή των αφηρημένων συντακτικών δέντρων (ASTs) του κάθε τμήματος, για τον υπολογισμό της δομικής ομοιότητας μεταξύ των αλλαγών. Πιο συγκεκριμένα, η ομοιότητα υπολογίστηκε με την εφαρμογή του αλγορίθμου απόστασης επεξεργασίας δέντρων (TED) pq-Grams στα ASTs των δεδομένων. Επιπρόσθετα, υλοποιήθηκε ο υπολογισμός μιας επιπλέον μετρικής ομοιότητας που αφορά το λεξιλογικό περιεχόμενο του πηγαίου κώδικα των αλλαγών. Ως τεχνική διανυσματοποίησης και κατ' επέκταση σύγκρισης των αλλαγών σε επίπεδο κειμένου χρησιμοποιήθηκε το μοντέλο TF-IDF εφόσον είχαν ολοκληρωθεί βήματα προεπεξεργασίας με την βιβλιοθήκη NLTK.

Επιπρόσθετα, με το συνδυασμό των παραπάνω μετρικών ομοιότητας υπολογίζεται ένας τελικός πίνακας απόστασης, ο οποίος χρησιμοποιείται από τον αλγόριθμο ιεραρχικής ομαδοποίησης συγχώνευσης (Agglomerative Hierarchical Clustering). Εφόσον έγινε η κατάλληλη βελτιστοποίηση των παραμέτρων του αλγορίθμου, τα αποτελέσματα που προκύπτουν είναι ομάδες αλλαγών που περιλαμβάνουν αλλαγές πηγαίου κώδικα με όμοιο δομικό και λεξιλογικό περιεχόμενο. Από τις ομάδες που προέκυψαν έγινε επιλογή των βέλτιστων, με τη χρήση ενός πλήθους παραμέτρων, και πραγματοποιήθηκε εύρεση του πιο αντιπροσωπευτικού προτύπου(centroid) για κάθε μία από αυτές.

Τέλος, τα αποτελέσματα που προέκυψαν και παρουσιάζονται στο Κεφάλαιο 4, αξιολογήθηκαν υπολογίζοντας τον αριθμό εμφάνισης τους στο σύνολο των commits στα δεδομένα αξιολόγησης, αλλά και μελετώντας χειροκίνητα την λειτουργία τους.

Συμπερασματικά, με βάση τα αποτελέσματα της προηγούμενης ενότητας είναι σαφές πως προέκυψαν χρήσιμα πρότυπα αλλαγών τα οποία μπορούν να προσφέρουν λύσεις στους μηχανικούς λογισμικού για την αντιμετώπιση προβλημάτων. Ωστόσο, υπάρχει η δυνατότητα εξέλιξης και βελτίωσης του συστήματος της παρούσας εργασίας ώστε να είναι πιο εύρωστο και αποδοτικό.

6.2. Μελλοντική Εργασία

Παρακάτω, είναι χρήσιμο να αναφερθούν μερικές βελτιώσεις και επεκτάσεις που θα μπορούσαν να αποτελέσουν αντικείμενο μελλοντικής εργασίας και θα οδηγούσαν σε πιο ποιοτικά αποτελέσματα. Μερικές βασικές ιδέες περιγράφονται παρακάτω:

- Επέκταση της μεθοδολογίας, ώστε να περιλαμβάνει στο σύνολο των δεδομένων, αλλαγές του πηγαίου κώδικα οι οποίες εμφανίζονται όχι μόνο εσωτερικά των μεθόδων του, αλλά και στα κύρια τμήματα του. Μια επιπλέον επέκταση που σχετίζεται με το σύνολο δεδομένων είναι η ανάλυση αλλαγών πηγαίου κώδικα που έχουν πραγματοποιηθεί σε περισσότερες γλώσσες προγραμματισμού. Κατά αυτό τον τρόπο τα τελικά πρότυπα θα ήταν πιο γενικά και θα μπορούσαν να προσφέρουν λύσεις σε πολλές περισσότερες περιπτώσεις.
- Χρήση βάσης δεδομένων για πιο αποτελεσματική αποθήκευση και ανάκτηση της πληροφορίας. Κατά αυτό τον τρόπο, οι απαιτήσεις μνήμης μπορούν να μειωθούν σημαντικά και επιπλέον ο χρόνος επεξεργασίας των πληροφοριών να μειωθεί ριζικά.
- Χρήση της κάρτας γραφικών κατά τον υπολογισμό των πινάκων ομοιότητας για την ελαχιστοποίηση του χρόνου εκτέλεσης. Παρόλο που έγινε χρήση βιβλιοθήκης παραλληλοποίησης της εν λόγω διαδικασίας, λόγω του μεγάλου αριθμού εγγραφών των ομάδων ο υπολογισμός των πινάκων απόστασης επιβαρύνει πάρα πολύ το σύστημα και το καθιστά αρκετά αργό.
- Υλοποίηση μιας γραφικής διεπαφής χρηστών (Graphical User Interface - GUI), ώστε η χρήση του συστήματος να είναι πιο φιλική προς τον χρήστη και να υπάρχει η δυνατότητα εκμετάλλευσης του συστήματος σε μορφή plug-in σε κάποιο IDE.

Βιβλιογραφία

[1] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE software,* vol. 27, no. 4, pp. 80-86, 2009.

[2] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Transactions on software engineering,* vol. 30, no. 2, pp. 126-139, 2004.

[3] G. Bavota, A. De Lucia, and R. Oliveto, "Identifying extract class refactoring opportunities using structural and semantic cohesion measures," *Journal of Systems and Software,* vol. 84, no. 3, pp. 397-414, 2011.

[4] M. Gasparic and A. Janes, "What recommendation systems for software engineering recommend: A systematic literature review," *Journal of Systems and Software,* vol. 113, pp. 101-113, 2016.

[5] M. Allamanis and C. Sutton, "Mining idioms from source code," in *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*, 2014, pp. 472-483.

[6] B. O'Sullivan, *Mercurial: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2009.

[7] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version control system: A review," *Procedia Computer Science,* vol. 135, pp. 408-415, 2018.

[8] S. Otte, "Version control systems," *Computer Systems and Telematics,* pp. 11-13, 2009.

[9] B. De Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?," in *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, 2009: IEEE, pp. 36-39.

[10] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, "Version Control with Subversion For Subversion 1.7 (Compiled from r6037)," 2002.

[11] R. Somasundaram, *Git: Version control for everyone*. Packt Publishing Ltd, 2013.

[12] A. Yip, B. Chen, and R. T. Morris, "Pastwatch: A Distributed Version Control System," in *NSDI*, 2006.

[13] C. Rodríguez-Bustos and J. Aponte, "How distributed version control systems impact open source software projects," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012: IEEE, pp. 36-39.

[14] P. C. Rigby, E. T. Barr, C. Bird, P. Devanbu, and D. M. German, "What effect does distributed version control have on oss project organization?," in *2013 1st International Workshop on Release Engineering (RELENG)*, 2013: IEEE, pp. 29-32.

[15] S. B. Chacon S, *Pro Git*. USA: Berkeley, 2014.

[16] K. Aslan, H. Skaf-Molli, and P. Molli, "Connecting Distributed Version Control Systems communities to linked open data," in *2012 International Conference on Collaboration Technologies and Systems (CTS)*, 2012: IEEE, pp. 242-250.

[17] N. R. Rao and K. C. Sekharaiah, "A Methodological Review Based Version Control System with Evolutionary Research for Software Processes," in *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, 2016, pp. 1-6.

[18] M. Reddy, *API Design for C++*. Elsevier, 2011.

[19] W. B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE transactions on Software Engineering,* vol. 31, no. 7, pp. 529-536, 2005.

[20] L. H. Group. "What Is Software Reuse?" https://web.archive.org/web/20141023013122/http://lombardhill.com/what_reuse.htm (accessed 22 June 2021, 2021).

[21] "Code reuse - DocForge Programming Wiki." https://web.archive.org/web/20110710143019/http:/docforge.com/wiki/Code_reuse (accessed 25 June, 2021).
[22] D. Riehle, "Framework design: A role modeling approach," ETH Zurich, 2000.
[23] E. Kindler and I. Krivy, "Object-oriented simulation of systems with sophisticated control," *International Journal of General Systems,* vol. 40, no. 3, pp. 313-343, 2011.
[24] J. Lewis and W. Loftus, "Java Software Solutions Foundations of Programming Design (Vol," ed: Madrid: Pearson Education Inc, 2008.
[25] A. L. Imoize, D. Idowu, and T. Bolaji, "A brief overview of software reuse and metrics in software engineering," *World Scientific News,* vol. 122, pp. 56-70, 2019.
[26] T. Anderson, "Software reliability: Principles and practices GJ Myers, Wiley, New York, 1976. No. of pages: 360. Price:£ l5. 8.5, $25.00," ed: Wiley Online Library, 1978.
[27] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal,* vol. 41, no. 1, pp. 4-12, 2002.
[28] H. Lieberman, "The debugging scandal and what to do about it," *Communications of the ACM,* vol. 40, no. 4, pp. 26-30, 1997.
[29] M. Eisenstadt, "My hairiest bug war stories," *Communications of the ACM,* vol. 40, no. 4, pp. 30-37, 1997.
[30] D. J. Hand, "Principles of data mining," *Drug safety,* vol. 30, no. 7, pp. 621-622, 2007.
[31] S. Chakrabarti *et al.*, "Data mining curriculum: A proposal (Version 1.0)," *Intensive Working Group of ACM SIGKDD Curriculum Committee,* vol. 140, pp. 1-10, 2006.
[32] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases (1996)," *Citado na,* p. 53, 2007.
[33] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.
[34] D. L. Olson, "Data mining in business services," *Service Business,* vol. 1, no. 3, pp. 181-193, 2007.
[35] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, "Automated design of both the topology and sizing of analog electrical circuits using genetic programming," in *Artificial Intelligence in Design'96*: Springer, 1996, pp. 151-170.
[36] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Transactions on Vehicular Technology,* vol. 69, no. 12, pp. 14413-14423, 2020.
[37] J. H. Friedman, "Data Mining and Statistics: What's the connection?," *Computing science and statistics,* vol. 29, no. 1, pp. 3-9, 1998.
[38] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Machine learning techniques for multimedia*: Springer, 2008, pp. 21-49.
[39] Z. Ghahramani, "Unsupervised learning," in *Summer School on Machine Learning*, 2003: Springer, pp. 72-112.
[40] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[41] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res,* vol. 10, no. 66-71, p. 13, 2009.
[42] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems,* vol. 2, no. 1-3, pp. 37-52, 1987.
[43] A. E. Hassan, "The road ahead for mining software repositories," in *2008 Frontiers of Software Maintenance*, 2008: IEEE, pp. 48-57.
[44] K. K. Chaturvedi, V. Sing, and P. Singh, "Tools in mining software repositories," in *2013 13th International Conference on Computational Science and Its Applications*, 2013: IEEE, pp. 89-98.
[45] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," in *Pioneers and Their Contributions to Software Engineering*: Springer, 1972, pp. 479-498.

[46] H. Agrawal and J. R. Horgan, "Dynamic program slicing," *ACM SIGPlan Notices,* vol. 25, no. 6, pp. 246-256, 1990.

[47] M. Weiser, "Program slicing," *IEEE Transactions on software engineering,* no. 4, pp. 352-357, 1984.

[48] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE transactions on Software Engineering,* vol. 30, no. 9, pp. 574-586, 2004.

[49] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, 1993, pp. 207-216.

[50] D. B. Leblang, "The {CM} Challenge: Configuration management That Works in Walter F. Tichy, editor, Configuration Management. Trends in Software," ed: John Wiley and Sons, 1994.

[51] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 11, no. 3, pp. 309-346, 2002.

[52] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, 1994, vol. 1215: Citeseer, pp. 487-499.

[53] J. S. Park, M.-S. Chen, and P. S. Yu, "Using a hash-based method with transaction trimming for mining association rules," *IEEE transactions on knowledge and data engineering,* vol. 9, no. 5, pp. 813-825, 1997.

[54] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM sigmod record,* vol. 29, no. 2, pp. 1-12, 2000.

[55] H. Osman, M. Lungu, and O. Nierstrasz, "Mining frequent bug-fix code changes," in *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014: IEEE, pp. 343-347.

[56] M. Martinez, L. Duchien, and M. Monperrus, "Automatically extracting instances of code change patterns with ast analysis," in *2013 IEEE international conference on software maintenance*, 2013: IEEE, pp. 388-391.

[57] B. Fluri, M. Wursch, M. PInzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Transactions on software engineering,* vol. 33, no. 11, pp. 725-743, 2007.

[58] B. Fluri and H. C. Gall, "Classifying change types for qualifying change couplings," in *14th IEEE International Conference on Program Comprehension (ICPC'06)*, 2006: IEEE, pp. 35-45.

[59] K. Pan, S. Kim, and E. J. Whitehead, "Toward an understanding of bug fix patterns," *Empirical Software Engineering,* vol. 14, no. 3, pp. 286-315, 2009.

[60] R. Rolim, G. Soares, R. Gheyi, T. Barik, and L. D'Antoni, "Learning quick fixes from code repositories," *arXiv preprint arXiv:1803.03806,* 2018.

[61] H. Zhong and Z. Su, "An empirical study on real bug fixes," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, vol. 1: IEEE, pp. 913-923.

[62] Q. Hanam, F. S. d. M. Brito, and A. Mesbah, "Discovering bug patterns in JavaScript," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 144-156.

[63] X. B. D. Le, D. Lo, and C. Le Goues, "History driven program repair," in *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, 2016, vol. 1: IEEE, pp. 213-224.

[64] F. Long and M. Rinard, "Automatic patch generation by learning correct code," in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2016, pp. 298-312.

[65] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013: IEEE, pp. 802-811.

[66] M. Soto and C. Le Goues, "Common statement kind changes to inform automatic program repair," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, 2018: IEEE, pp. 102-105.

[67] M. Martinez and M. Monperrus, "Coming: A tool for mining change pattern instances from git commits," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019: IEEE, pp. 79-82.

[68] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 437-440.

[69] N. Sahavechaphan and K. Claypool, "XSnippet: mining for sample code," in *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2006, pp. 413-430.

[70] C. E. Leiserson and T. B. Schardl, "A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)," in *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*, 2010, pp. 303-314.

[71] T. Xie and J. Pei, "MAPO: Mining API usages from open source repositories," in *Proceedings of the 2006 international workshop on Mining software repositories*, 2006, pp. 54-57.

[72] S. Thummalapenta and T. Xie, "Parseweb: a programmer assistant for reusing open source code on the web," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 204-213.

[73] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 513-522.

[74] D. Wightman, Z. Ye, J. Brandt, and R. Vertegaal, "Snipmatch: Using source code context to enhance snippet retrieval and parameterization," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*, 2012, pp. 219-228.

[75] Y. Wei, N. Chandrasekaran, S. Gulwani, and Y. Hamadi, "Building bing developer assistant," *Technical Report. MSR-TR-2015-36, Microsoft Research,* 2015.

[76] D. Spadini, M. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 908-911.

[77] K. Krishna and M. N. Murty, "Genetic K-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 29, no. 3, pp. 433-439, 1999.

[78] T. J. Archdeacon, *Correlation and regression analysis: a historian's guide*. Univ of Wisconsin Press, 1994.

[79] N. Augsten, M. Böhlen, and J. Gamper, "The pq-gram distance between ordered labeled trees," *ACM Transactions on Database Systems (TODS),* vol. 35, no. 1, pp. 1-36, 2008.

[80] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science,* vol. 290, no. 5500, pp. 2323-2326, 2000.

[81] E. Yourdon and L. L. Constantine, "Structured design. Fundamentals of a discipline of computer program and systems design," *Englewood Cliffs: Yourdon Press,* 1979.

[82] S. Fakhoury, D. Roy, A. Hassan, and V. Arnaoudova, "Improving source code readability: theory and practice," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019: IEEE, pp. 2-12.

[83] B. Paaßen, "Revisiting the tree edit distance and its backtracing: A tutorial," *arXiv preprint arXiv:1805.06869,* 2018.

[84] N. Augsten, M. Böhlen, and J. Gamper, "Approximate matching of hierarchical data using pq-grams," in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 301-312.

[85] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*: Springer, 2009, pp. 1-4.

[86] C. F. Dietrich, *Uncertainty, calibration and probability: the statistics of scientific and industrial measurement*. Routledge, 2017.

[87] F. E. Croxton and D. J. Cowden, "Applied general statistics," 1939.

[88] J. E. Friedl, *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.