ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΧΝΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ (MACHINE LEARNING) ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΠΡΟΒΛΗΜΑΤΩΝ ΣΧΕΔΙΑΣΗΣ Ή ΑΛΛΩΝ ΠΑΡΑΜΕΤΡΩΝ ΣΕ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

Διπλωματική Εργασία

του

Σωτηρίου Γεώργιου

Θεσσαλονίκη, Σεπτέμβριος 2022

ΤΕΧΝΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ (MACHINE LEARNING) ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΠΡΟΒΛΗΜΑΤΩΝ ΣΧΕΔΙΑΣΗΣ Ή ΑΛΛΩΝ ΠΑΡΑΜΕΤΡΩΝ ΣΕ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

Σωτηρίου Γεώργιος

Μηχανικός Πληροφορικής Τεχνολογικής Εκπαίδευσης, Σχολή Τεχνολογικών Εφαρμογών Τμήμα Πληροφορικής ΑΤΕΙ, 2015

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Χατζηγεωργίου Αλέξανδρος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την __/__/____

Χατζηγεωργίου Αλέξανδρος	Σατρατζέμη Μάγια	Ξυνόγαλος Στυλιανός
.....	.....	.....

Σωτηρίου Γεώργιος

.....

Περίληψη

Η ανίχνευση των παραβιάσεων των θεμελιωδών αρχών σχεδιασμού και η ανακατασκευή του πηγαίου κώδικα μίας μονάδας λογισμικού είναι ζωτικής σημασίας για τη διατήρηση της ποιότητας, τη μείωση της πολυπλοκότητας και την αύξηση της αποτελεσματικότητας αυτής. Τα τελευταία χρόνια, αναπτύσσονται εργαλεία που βασίζονται στη μηχανική μάθηση, τα οποία αναλύουν αμιγώς τον πηγαίο κώδικα των υπολογιστικών συστημάτων.

Στόχος της παρούσας διπλωματικής εργασίας είναι η διερεύνηση των τεχνικών και των σχετικών τους εργαλείων που βασίζονται σε τεχνικές μηχανικής μάθησης για τον εντοπισμό μονάδων λογισμικού που εμφανίζουν συμπτώματα κακής σχεδίασης, υψηλό τεχνικό χρέος ή σφάλματα, δηλ. code smells και anti-patterns.

Αρχικά πραγματοποιήθηκε βιβλιογραφική έρευνα με στόχο την διερεύνηση των μεθόδων και την καταγραφή των αλγορίθμων μηχανικής μάθησης που έχουν εφαρμοστεί για την υλοποίηση ανιχνευτών code smells και anti-patterns σε μονάδες λογισμικού. Σε συνέχεια της βιβλιογραφικής έρευνας, αναζητήθηκαν και εντοπίστηκαν συνολικά 11 εργαλεία ανίχνευσης code smells και anti-patterns που βασίζονται στην μηχανική μάθηση. Από αυτά επιλέχθηκαν 4, τα οποία αξιολογήθηκαν τόσο συλλογικά, όσο και επιμέρους. Η αξιολόγησή τους πραγματοποιήθηκε με τη χρήση 3ων έργων Android εφαρμογών που διατίθενται ως ελεύθερο λογισμικό.

Τα αποτελέσματα έδειξαν ότι η πλειοψηφία των εργαλείων χρησιμοποιούν αλγορίθμους βαθιάς μάθησης. Όσον αφορά τα εργαλεία που χρησιμοποιούν τους ίδιους αλγορίθμους, διαπιστώθηκε ότι η αποτελεσματικότητά τους διαφέρει, ενώ συχνά δεν δίνουν τα ίδια αποτελέσματα. Επιπλέον, παρατηρήθηκε ότι το μέγεθος του έργου επηρεάζει σημαντικά την αποτελεσματικότητα των εργαλείων. Τέλος, αξίζει να σημειωθεί ότι από τα 22 διαφορετικά είδη code smells, τα επιλεγμένα εργαλεία μπορούν να διακρίνουν μόλις 5 από αυτά.

Λέξεις Κλειδιά: Μηχανική Μάθηση, Code Smell, Anti-patterns

Abstract

Detecting violations of fundamental design principles and refactoring the source code of a software module is critical to maintaining quality, reducing complexity, and increasing efficiency. In recent years, machine learning-based tools have been developed that analyze the source code of computing systems.

This thesis investigates the tools based on machine learning techniques to detect software modules that indicate bad design, high technical debt, or bugs, i.e., code smells and anti-patterns.

Initially, a literature study was conducted to investigate the methods and identify the machine learning algorithms used to implement the code smell and anti-pattern detectors in software modules. This study led to 11 code smells and anti-pattern detection tools utilizing machine learning. Out of these, 4 tools were selected and were evaluated both collectively and individually. Their evaluation was performed via 3 open-source Android projects.

The results showed that most of the tools use deep learning algorithms. However, the detectors using the same ML algorithms did not have the same accuracy and often did not give the same results. In addition, it was noticed that the project's size significantly affects the tools' effectiveness. Finally, out of 22 different types of code smells, the selected tools can distinguish only 5 of them.

Keywords: Machine Learning, Code Smell, Anti-patterns

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Συμβολισμοί

Συμβολισμός	Επεξήγηση
AL	Active Learning Ενεργή Μάθηση
BC	Brain Class Κλάση Εγκεφάλου
Bi-LSTM	Bidirectional Long-Short Term Memory Αμφίδρομα Δίκτυα Μακράς Βραχύχρονης Μνήμης
BM	Brain Method Μέθοδος Εγκεφάλου
CNN	Convolutional Neural Network Συνελικτικό Νευρωνικό Δίκτυο
GRU	Gated Recurrent Unit Ανατροφοδοτούμενη Μονάδα με Πύλη
LSTM	Long-Short Term Memory Δίκτυα Μακράς Βραχύχρονης Μνήμης
ML	Machine Learning Μηχανική Μάθηση
MCC	Matthew’s Correlation Coefficient Συντελεστής Συσχέτισης Matthews
MLP	Multi-layer Perceptron Πολυστρωματικό Perceptron

NLP	Natural Language Processing Επεξεργασία Φυσικής Γλώσσας
ROC	Receiver Operating Characteristic Καμπύλη Λειτουργικού Χαρακτηριστικού Δέκτη
RNN	Recurrent Neural Network Επαναλαμβανόμενο Νευρωνικό Δίκτυο
SVM	Support Vector Machine Μηχανή Διανύσματος Υποστήριξης
TF-IDF	Term Frequency–Inverse Document Frequency Συχνότητα Όρων-Αντίστροφη Συχνότητα Εγγράφων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Κατά την ανάπτυξη υπολογιστικών συστημάτων και εφαρμογών συχνά παρατηρείται η χρήση προτύπων ανάπτυξης λογισμικού (anti-patterns) που θεωρούνται κακές πρακτικές προγραμματισμού, καθώς προκαλούν αρνητικές συνέπειες. Επιπλέον, παρατηρούνται χαρακτηριστικά στον πηγαίο κώδικα, τα οποία υποδηλώνουν παραβίαση των θεμελιωδών αρχών σχεδιασμού [1]. Τα χαρακτηριστικά αυτά αναφέρονται με τον όρο code smells και θεωρούνται βασική απειλή για τον κύκλο ζωής των υπολογιστικών συστημάτων. Η ανίχνευση των code smells και των anti-patterns, καθώς και η ανακατασκευή του πηγαίου κώδικα είναι ζωτικής σημασίας για τη διατήρηση της ποιότητας, τη μείωση της πολυπλοκότητας και την αύξηση της αποτελεσματικότητας μιας εφαρμογής λογισμικού. Διαφορετικά εργαλεία μπορούν να χρησιμοποιήσουν διαφορετικές τεχνικές ανίχνευσης για την ανίχνευση code smells και anti-patterns.

Οι περισσότερες παραδοσιακές μέθοδοι για την ταξινόμηση των code smells και anti-patterns βασίζονται αποκλειστικά σε δομικές αντικειμενοστρεφείς μετρήσεις και χειροκίνητα σχεδιασμένους ευρετικούς κανόνες. Οι Perocelli et. al. [2] διαπίστωσαν ότι τα περισσότερα από τα αυτοματοποιημένα εργαλεία βασίζονται σε μετρήσεις. Αυτά τα εργαλεία υπολογίζουν ένα σύνολο μετρήσεων στον πηγαίο κώδικα του υπολογιστικού συστήματος, τις συνδυάζουν σε ένα σύνολο κανόνων και συγκρίνουν τις μετρήσεις με εμπειρικά προσδιορισμένα όρια, καταλήγοντας στο ότι υπάρχει ένα code smell ή/και anti-pattern, εάν ξεπεραστεί κάποιο από τα προκαθορισμένα όρια. Συνεπώς σε αυτού του είδους τα εργαλεία τα αποτελέσματα ποικίλλουν σημαντικά ανάλογα με τα καθορισμένα όρια και τις μετρικές μετρήσεις [3], [4], [5], [6], [7], [8], [9].

Ωστόσο, σύμφωνα με τους Alkharabsheh et. al. [10] η αυξανόμενη πολυπλοκότητα και οι ραγδαίες εξελίξεις στην ανάπτυξη συστημάτων λογισμικού απαιτούν περισσότερες νέες προσεγγίσεις, τεχνικές, πρακτικές και εργαλεία που μπορούν να βοηθήσουν στον εντοπισμό των code smells και των anti-patterns. Έτσι, τα τελευταία χρόνια, αναπτύσσονται εργαλεία που βασίζονται στη μηχανική μάθηση, τα οποία αναλύουν αμιγώς τον πηγαίο κώδικα των υπολογιστικών συστημάτων [2], [11], [12], [13], [14], [15], [16], [17], [18].

1.2 Σκοπός – Στόχοι

Στόχος της παρούσας διπλωματικής εργασίας είναι η διερεύνηση των εργαλείων που βασίζονται σε τεχνικές μηχανικής μάθησης για τον εντοπισμό μονάδων λογισμικού που εμφανίζουν συμπτώματα κακής σχεδίασης, υψηλό τεχνικό χρέος ή σφάλματα, δηλ. code smells και anti-patterns. Η εργασία περιλαμβάνει σύγκριση τεσσάρων (4) εργαλείων έργων ανοιχτού κώδικα.

1.3 Ερωτήματα – Υποθέσεις

Η παρούσα ερευνητική εργασία αποσκοπεί να μελετήσει τα ακόλουθα ερευνητικά ερωτήματα:

1. Ποιες τεχνικές μηχανικής μάθησης έχουν εφαρμοστεί στην προσπάθεια εντοπισμού code smells και anti-patterns σε μονάδες λογισμικού;
2. Υπάρχουν εργαλεία ανίχνευσης code smells και anti-patterns τα οποία να έχουν αναπτυχθεί χρησιμοποιώντας αλγορίθμους μηχανικής μάθησης;
3. Ποια είδη code smells και anti-patterns μπορούν να εντοπιστούν από τα διαθέσιμα εργαλεία ανίχνευσης που βασίζονται στην μηχανική μάθηση;
4. Μπορούν τα διαθέσιμα εργαλεία ανίχνευσης code smells και anti-patterns να θεωρηθούν επαρκή για τον εντοπισμό παραβιάσεων των θεμελιωδών αρχών σχεδιασμού λογισμικού σε αντικειμενοστραφή έργα;

1.4 Διάρθρωση της μελέτης

Η διπλωματική εργασία αποτελείται από πέντε (5) κεφάλαια. Στο πρώτο κεφάλαιο γίνεται μία σύντομη εισαγωγή και παρουσιάζονται ο στόχος της διπλωματικής εργασίας, τα ερευνητικά ερωτήματα που θα διερευνηθούν στο πλαίσιο αυτής, καθώς και η δομή της. Το δεύτερο κεφάλαιο αποτελεί το θεωρητικό μέρος της διπλωματικής, καθώς σε αυτό περιγράφονται εν συντομία όλες οι βασικές έννοιες και οι ορισμοί, ενώ παρατίθεται η ανασκόπηση της βιβλιογραφίας. Στο τρίτο κεφάλαιο περιγράφεται η μεθοδολογία της έρευνας που διεξήχθη. Το τέταρτο κεφάλαιο περιλαμβάνει την συλλογική, αλλά και την επιμέρους αξιολόγηση των χαρακτηριστικών των επιλεγμένων ανιχνευτών code smells και anti-patterns. Τέλος, στο πέμπτο κεφάλαιο γίνεται μία σύνοψη της διπλωματικής εργασίας και των αποτελεσμάτων.

2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

2.1 Θεωρητικό Υπόβαθρο

Παρακάτω παρατίθεται το θεωρητικό υπόβαθρο της εργασίας, στο οποίο περιγράφονται οι ορισμοί όλων των βασικών στοιχείων που είναι απαραίτητα στον αναγνώστη για την κατανόηση του πειραματικού μέρους της παρούσας διπλωματικής εργασίας.

Αρχικά, προκειμένου ο αναγνώστης να μπορεί να κατανοήσει τι είναι ένα ποιοτικό λογισμικό δίνεται ο ορισμός του λογισμικού υπολογιστών και περιγράφονται εν συντομία ποια είναι τα ποιοτικά χαρακτηριστικά του. Στην συνέχεια, αναλύονται τα code smells και anti-patterns, δηλαδή τα χαρακτηριστικά ενός λογισμικού τα οποία επιδρούν αρνητικά στην ποιότητα του. Τέλος, δίνεται ο ορισμός της Μηχανικής Μάθησης, δηλαδή του εργαλείου που εστιάζει η διπλωματική εργασία και χρησιμοποιείται για την ανίχνευση των code smells και anti-patterns και περιγράφονται οι μετρικές αξιολόγησης των αλγορίθμων Μηχανικής Μάθησης, με την χρήση των οποίων αξιολογούνται τα επιλεγμένα εργαλεία στο πειραματικό μέρος της εργασίας.

2.1.1 Λογισμικό Υπολογιστών

Με τον όρο λογισμικό υπολογιστών ή λογισμικό (software) ορίζεται ένα σύνολο προγραμμάτων και τεκμηρίωσης. Τα προγράμματα περιλαμβάνουν δομές δεδομένων για την διαχείριση της πληροφορίας και εντολές που όταν εκτελούνται παρέχουν στους χρήστες επιθυμητές λειτουργίες και επιδόσεις. Η τεκμηρίωση περιγράφει τον τρόπο λειτουργίας και χρήσης των προγραμμάτων. [19]

2.1.2 Ποιοτικά Χαρακτηριστικά Λογισμικού

Σύμφωνα με τις βασικές αρχές της Τεχνολογίας Λογισμικού ένα λογισμικό θα πρέπει να προσαρμόζεται στις εξελισσόμενες συνθήκες του περιβάλλοντός του [19]. Σύμφωνα με το πρότυπο ISO/IEC 25010:2011 [20] τα σημαντικότερα ποιοτικά χαρακτηριστικά που θα πρέπει να διαθέτει ένα λογισμικό υπολογιστών είναι:

- Λειτουργική καταλληλότητα (Functional Suitability): Αντιπροσωπεύει τον βαθμό στον οποίο ένα προϊόν ή ένα σύστημα παρέχει λειτουργίες που ικανοποιούν δηλωμένες και υπονοούμενες ανάγκες όταν χρησιμοποιείται

υπό συγκεκριμένες συνθήκες. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:

- Λειτουργική Πληρότητα (Functional completeness)
- Λειτουργική Ορθότητα (Functional correctness)
- Λειτουργική Καταλληλότητα (Functional appropriateness).

- Αποδοτικότητα Επίδοσης (Performance efficiency): Αντιπροσωπεύει την απόδοση σε σχέση με την ποσότητα των πόρων που χρησιμοποιούνται υπό καθορισμένες συνθήκες. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:
  - Χρονική Συμπεριφορά (Time behavior)
  - Χρήση Πόρων (Resource utilization)
  - Χωρητικότητα (Capacity)
- Συμβατότητα (Compatibility): Ο βαθμός στον οποίο ένα προϊόν, σύστημα ή στοιχείο μπορεί να ανταλλάσσει πληροφορίες με άλλα προϊόντα, συστήματα ή στοιχεία ή/και να εκτελεί τις απαιτούμενες λειτουργίες του ενώ μοιράζεται το ίδιο περιβάλλον υλικού ή λογισμικού. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:
  - Συνύπαρξη (Co-existence)
  - Διαλειτουργικότητα (Interoperability)
- Ευχρηστία (Usability): Ο βαθμός στον οποίο ένα προϊόν ή ένα σύστημα μπορεί να χρησιμοποιηθεί από συγκεκριμένους χρήστες για την επίτευξη συγκεκριμένων στόχων με αποτελεσματικότητα, αποδοτικότητα και ικανοποίηση σε ένα συγκεκριμένο πλαίσιο χρήσης. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:
  - Καταλληλότητα/Αναγνωρισιμότητα (Appropriateness/Recognizability)
  - Δυνατότητα εκμάθησης (Learnability)
  - Λειτουργικότητα (Operability)
  - Προστασία σφαλμάτων χρήστη (User error protection)
  - Αισθητική διεπαφής χρήστη (User interface aesthetics)
  - Προσβασιμότητα (Accessibility)
- Αξιοπιστία (Reliability): Ο βαθμός στον οποίο ένα σύστημα, προϊόν ή εξάρτημα εκτελεί συγκεκριμένες λειτουργίες υπό καθορισμένες συνθήκες

για μια συγκεκριμένη χρονική περίοδο. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:

- Ωριμότητα (Maturity)
- Διαθεσιμότητα (Availability)
- Ανοχή σε σφάλματα (Fault tolerance)
- Δυνατότητα επαναφοράς (Recoverability)

- Ασφάλεια (Security): Ο βαθμός στον οποίο ένα προϊόν ή σύστημα προστατεύει πληροφορίες και δεδομένα έτσι ώστε τα άτομα ή άλλα προϊόντα ή συστήματα να έχουν τον βαθμό πρόσβασης στα δεδομένα που είναι κατάλληλος για τους τύπους και τα επίπεδα εξουσιοδότησής τους. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:
  - Εμπιστευτικότητα (Confidentiality)
  - Ακεραιότητα (Integrity)
  - Μη αποποίηση (Non-repudiation)
  - Ευθύνη (Accountability)
  - Αυθεντικότητα (Authenticity)
- Συντηρησιμότητα (Maintainability): Αντιπροσωπεύει τον βαθμό αποτελεσματικότητας και αποδοτικότητας με τον οποίο ένα προϊόν ή σύστημα μπορεί να τροποποιηθεί για να το βελτιωθεί, να διορθωθεί ή να το προσαρμοστεί στις αλλαγές στο περιβάλλον και στις απαιτήσεις. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:
  - Τμηματοποίηση (Modularity)
  - Επαναχρησιμοποίηση (Reusability)
  - Δυνατότητα Ανάλυσης (Analyzability)
  - Δυνατότητα Τροποποίησης (Modifiability)
  - Δυνατότητα Δοκιμής (Testability)
- Φορητότητα (Portability): Ο βαθμός αποτελεσματικότητας και αποδοτικότητας με τον οποίο ένα σύστημα, προϊόν ή στοιχείο μπορεί να μεταφερθεί από ένα υλικό, λογισμικό ή άλλο λειτουργικό ή περιβάλλον χρήσης σε άλλο. Αποτελείται από τα ακόλουθα επιμέρους χαρακτηριστικά:
  - Προσαρμοστικότητα (Adaptability)
  - Δυνατότητα Εγκατάστασης (Installability)
  - Δυνατότητα Αντικατάστασης (Replaceability)

2.1.3 Code Smells

Ο όρος code smell εισήχθη για πρώτη φορά το 1999 από τους Martin Fowler και τον Kent Beck [21] και αναφέρεται στα χαρακτηριστικά ενός πηγαίου κώδικα, που υποδηλώνουν παραβίαση των θεμελιωδών αρχών σχεδιασμού. Τα code smells συνήθως δεν είναι σφάλματα ή τεχνικά λάθη και δεν εμποδίζουν τη λειτουργία του προγράμματος. Αντίθετα, υποδεικνύουν αδυναμίες στο σχεδιασμό που μπορεί να επιβραδύνουν την ανάπτυξη ή να αυξήσουν τον κίνδυνο σφαλμάτων ή αποτυχιών στο μέλλον. Τα συχνότερα και σύμφωνα με τους Fowler και Beck τα βασικότερα code smells είναι 22, και είναι τα:

- Διπλότυπος Κώδικας (Duplicated Code): Ένα από τα πιο συχνά εμφανιζόμενα code smells. Συμβαίνει όταν η ίδια δομή κώδικα μπορεί να βρεθεί σε περισσότερα από ένα σημεία ενός λογισμικού. Για παράδειγμα μπορεί να είναι μία έκφραση που υπάρχει σε δύο μεθόδους μίας κλάσης ή σε δύο μεθόδους αδερφών υποκλάσεων. Ο κώδικας που είναι παρόμοιος αλλά όχι ακριβώς ο ίδιος ή οι μέθοδοι που κάνουν το ίδιο πράγμα με διαφορετικό αλγόριθμο θεωρούνται επίσης διπλότυπος κώδικας.
- Μεγάλη Μέθοδος (Long Method): Μία μέθοδος μπορεί να θεωρηθεί μεγάλη όταν έχει πάρα πολλές γραμμές κώδικα ή περιέχει πολλαπλούς όρους και βρόχους. Οι μεγάλες μέθοδοι είναι πιο δύσκολο να κατανοηθούν από τις πιο σύντομες μεθόδους. Έτσι, συχνά είναι αναγκαίο αυτές να χωριστούν σε πολλές μικρότερες μεθόδους.
- Μεγάλη Κλάση (Large Class): Οι μεγάλες κλάσεις, συχνά, φέρουν πάρα πολλές μεταβλητές στιγμιότυπων. Μπορούν επίσης να περιέχουν πάρα πολύ κώδικα, ο οποίος μπορεί να οδηγήσει σε διπλότυπο κώδικα και άλλα ζητήματα συντήρησης. Αυτές οι κλάσεις πρέπει να χωρίζονται σε πολλαπλές κλάσεις ή υποκλάσεις ή από αυτές θα πρέπει να εξάγονται διεπαφές με στόχο την απλούστευση τους. Οι μεγάλες κλάσεις συχνά αναφέρονται και ως Κλάσεις-Θεοί (God Classes).
- Μεγάλη Λίστα Παραμέτρων (Long Parameter List): Προκειμένου οι μέθοδοι να έχουν ως είσοδο όλα τα απαραίτητα δεδομένα για την ορθή λειτουργία τους, παρατηρείται συχνά οι λίστες παραμέτρων να είναι ιδιαιτέρως μεγάλες. Οι μεγάλες λίστες παραμέτρων είναι δύσκολο να κατανοηθούν και να διατηρηθούν, καθώς η συνεχής αλλαγή στα απαραίτητα δεδομένα τις καθιστά ασυνεπείς. Ωστόσο, σε

αντικειμενοστραφή προγράμματα, οι λίστες παραμέτρων μπορεί να είναι πολύ μικρότερες λόγω της χρήσης των αντικειμένων.

- Αποκλίνουσα Αλλαγή (Divergent Change): Η Αποκλίνουσα Αλλαγή παρουσιάζεται είτε όταν μια κλάση αλλάζει με πολλούς τρόπους για διαφορετικούς λόγους είτε σε περίπτωση που υπάρχει ανάγκη να γίνουν αλλαγές σε ένα σύστημα, αλλά είναι εξαιρετικά δύσκολος ο ακριβής εντοπισμός των σημείων που πρέπει να αλλάξουν. Επιπλέον, παρουσιάζεται όταν πολλές μη-σχετιζόμενες μέθοδοι αλλάζουν μετά από μία αλλαγή σε μία κλάση, ενώ κάθε αντικείμενο θα έπρεπε να αλλάζει μόνο ως αποτέλεσμα ενός είδους αλλοίωσης. Η εξαγωγή κλάσης μπορεί να χρησιμοποιηθεί για να διορθώσει αυτό το πρόβλημα και να συνδυάσει όλες τις μεταβαλλόμενες πτυχές σε μια νέα κλάση.
- Shotgun Surgery: Το Shotgun Surgery παρουσιάζεται στην περίπτωση που μία αλλαγή στον πηγαίο κώδικα οδηγεί σε πολλές μικρές αλλαγές σε πολλές διαφορετικές κλάσεις. Στο Divergent Change μια κλάση περνάει από πολλά είδη αλλαγών, αλλά με το Shotgun Surgery γίνεται το αντίθετο – μια αλλαγή αλλάζει πολλές κλάσεις. Για την επίλυση του Shotgun Surgery οι αλλαγές θα πρέπει να μετακινηθούν σε μία κλάση.
- Παράλληλες Ιεραρχίες Κληρονομικότητας (Parallel Inheritance Hierarchies): Πρόκειται για μια ειδική περίπτωση Shotgun Surgery. Σε αυτήν την περίπτωση, όταν υλοποιείται μια νέα υποκλάση σε μία κλάση, τότε θα πρέπει επίσης να υλοποιηθεί μία νέα υποκλάση και σε κάποια άλλη κλάσης. Αυτό δημιουργεί περιττές επικαλύψεις. Για να την εξάλειψη αυτού του code smell, οι περιπτώσεις της μιας ιεραρχίας πρέπει να αναφέρονται στις περιπτώσεις της άλλης. Στη συνέχεια, η ιεραρχία στην αναφερόμενη κλάση μπορεί να αφαιρεθεί.
- Feature Envy: Το Feature Envy παρουσιάζεται, όταν μια μέθοδος χρησιμοποιεί δεδομένα κυρίως από άλλες κλάσεις αντί της ίδιας. Για παράδειγμα, Feature Envy υπάρχει όταν μια μέθοδος της κλάσης Α καλεί πάρα πολλές μεθόδους get για ένα αντικείμενο της κλάσης Β. Για την επίλυση του προβλήματος αυτού η μέθοδος της κλάσης Α θα πρέπει να μετακινηθεί στην κλάση Β.

- Συστάδες Δεδομένων (Data Clumps): Τα στοιχεία δεδομένων, που εμφανίζονται μαζί σε πολλές θέσεις στον κώδικα, ονομάζονται συστάδες δεδομένων. Μία συστάδα πολλαπλών πεδίων μπορεί να μετατραπεί σε ένα αντικείμενο. Κατά την κλήση μίας μεθόδου αυτά τα στοιχεία δεδομένων, που θα οδηγούσαν σε μία μακριά λίστα παραμέτρων, μπορούν να περιοριστούν με την εισαγωγή του αντίστοιχου αντικειμένου παραμέτρων. Με αυτόν τον τρόπο οι λίστες παραμέτρων μπορούν να συντομευθούν και η κλήση μεθόδων απλοποιείται.
- Primitive Obsession: Τα Primitive Obsession συμβαίνουν, όταν αντί για ξεχωριστά αντικείμενα, μικρές εργασίες αντιμετωπίζονται με primitives. Σε αυτές τις περιπτώσεις, οι τιμές δεδομένων πρέπει να αντικατασταθούν με αντικείμενα και οι τύποι με κλάσεις. Τα primitives μπορούν επίσης να εξαχθούν ως κλάση ή να εισαχθούν ως αντικείμενο παραμέτρων.
- Εντολές Switch (Switch Statements): Συχνά οι ίδιες εντολές Switch υπάρχουν σε πολλά μέρη του κώδικα και προκύπτει το πρόβλημα της αντιγραφής. Εάν προστεθεί μια νέα ρήτρα σε μία εντολή Switch, τότε θα πρέπει να αλλάξει και σε όλες τις άλλες αντίστοιχες εντολές Switch. Συνήθως, ο πολυμορφισμός θα πρέπει να λαμβάνεται υπόψη αντί για τις δηλώσεις εναλλαγής για τη βελτίωση της οργάνωσης του κώδικα.
- Lazy Class: Αυτός ο όρος χρησιμοποιείται σε περίπτωση ύπαρξης μιας κλάσης, η οποία δεν εξυπηρετεί στο μέγιστο τον σκοπό της. Συνήθως αφορά είτε κλάσεις που συρρικνώθηκαν κατά την ανακατασκευή, είτε κλάσεις που προστέθηκαν με πρόθεση την μεταγενέστερη εφαρμογή τους, η οποία ωστόσο δεν πραγματοποιήθηκε ποτέ. Οι κλάσεις, που είναι σχεδόν άχρηστες, μπορούν να διαγραφούν αφού μεταφερθούν όλα τα εγγενή τα χαρακτηριστικά τους σε άλλη μία άλλη κλάση. Επιπλέον, οι υποκλάσεις που έχουν λίγες λειτουργίες μπορούν να συγχωνευθούν με τις μητρικές τους κλάσεις.
- Speculative Generality: Το Speculative Generality παρατηρείτε όταν ένα μέρος του κώδικα γράφεται με την ελπίδα μεταγενέστερης εφαρμογής. Αυτό δημιουργεί κλάσεις, μεθόδους και πεδία που δεν χρησιμοποιούνται και ο κώδικας είναι πιο δύσκολο να διατηρηθεί και να κατανοηθεί. Σε αυτή

την περίπτωση, συχνά οι μόνοι χρήστες μιας μεθόδου ή μιας κλάσης είναι οι ίδιοι οι προγραμματιστές.

- Προσωρινά Πεδία (Temporary Field): Μερικές φορές, ένα αντικείμενο ορίζεται με σκοπό να χρησιμοποιηθεί για μία συγκεκριμένη περίπτωση. Αυτό το είδος κώδικα μπορεί να είναι δύσκολο να κατανοηθεί, καθώς ορισμένες μεταβλητές φαίνεται να μην χρησιμοποιούνται, για αυτό μπορεί να δημιουργηθεί μια νέα κλάση για τις αχρησιμοποίητες μεταβλητές ή ο υπό όρους κώδικας μπορεί να εξαλειφθεί δημιουργώντας ένα εναλλακτικό στοιχείο.
- Αλυσίδες Μηνυμάτων (Message Chains): Οι αλυσίδες μηνυμάτων συμβαίνουν, όταν μια σειρά κλήσεων λαμβάνει χώρα με τη μορφή πελάτη (client) που ζητά ένα αντικείμενο εκ μέρους ενός άλλου αντικειμένου, το οποίος στη συνέχεια ζητά ένα άλλο αντικείμενο και ούτω καθεξής. Συχνά αυτό έρχεται με τη μορφή μιας σειράς μεθόδων get ή ως μια ακολουθία temps. Οποιεσδήποτε αλλαγές σε αυτού του είδους τις σχέσεις οδηγούν σε αλλαγές στον πελάτη.
- Middle Man: Μια κλάση τύπου Middle Man υπάρχει μόνο για την ανάθεση εργασιών από μια κλάση σε μία άλλη. Η κλάση τύπου Middle Man μπορεί να αφαιρεθεί ή να μετατραπεί σε υποκατηγορία ενός πραγματικού αντικειμένου για να επεκτείνει τη συμπεριφορά του.
- Inappropriate Intimacy: Το Inappropriate Intimacy παρουσιάζεται όταν μια κλάση χρησιμοποιεί σε μεγάλο βαθμό τα εσωτερικά πεδία και τις μεθόδους μιας άλλης κλάσης. Οι κλάσεις, για να είναι πιο εύκολο να διατηρηθούν, δεν πρέπει να γνωρίζουν για τις υλοποιήσεις των άλλων κλάσεων. Για να μειωθεί το πρόβλημα αυτό, οι μέθοδοι και τα πεδία μπορούν να μετακινηθούν από τη μια κλάση στην άλλη. Οι υποκλάσεις και οι γονικές τους κλάσεις μπορούν να διαχωριστούν κατά την αντικατάσταση της ανάθεσης με κληρονομικότητα.
- Εναλλακτικές Κλάσεις με Διαφορετικές Διεπαφές (Alternative Classes with Different Interfaces): Πρόκειται για κλάσεις που έχουν τις ίδιες λειτουργίες και σκοπό, αλλά διαφορετικές υπογραφές. Οι μέθοδοι που κάνουν το ίδιο αλλά έχουν διαφορετικά ονόματα, μπορούν να μετονομαστούν. Επιπλέον, οι μέθοδοι μπορούν να μετακινηθούν για να

μετακινήσουν τη συμπεριφορά τους και να κάνουν την υπογραφή των κλάσεων ίδια. Με την ανακατασκευή πολλές φορές μπορεί να διαγραφεί μία από τις κλάσεις.

- Μη Ολοκληρωμένες Κλάσεις Βιβλιοθήκες (Incomplete Library Class): Οι κλάσεις βιβλιοθήκες μπορεί να γίνουν ακατάλληλες για τις ανάγκες των χρηστών, καθώς δεν μπορούν να αλλάξουν για την κάλυψη των αναγκών των χρηστών. Ο μόνος τρόπος τροποποίησης τέτοιου είδους βιβλιοθηκών είναι να δημιουργηθούν νέες μέθοδοι ή ακόμα να εισαχθούν ξένες μέθοδοι. Για μεγαλύτερου εύρους αλλαγές τότε θα πρέπει να εισαχθεί μία τοπική επέκταση.
- Κλάση Δεδομένων (Data Class): Πρόκειται για κλάσεις που περιλαμβάνουν αποκλειστικά πεδία τιμών και μεθόδους τύπου «get» και «set». Ουσιαστικά λειτουργούν ως containers δεδομένων, αφού δεν έχουν καμία πρόσθετη λειτουργικότητα. Ούτως ώστε οι κλάσεις δεδομένων να αποκτήσουν σκοπό, προτείνεται οι μέθοδοι της κλάσης χρήστη να μετακινηθούν στην κλάση δεδομένων.
- Απόρριψη Κληροδοτήματος (Refused Bequest): Η απόρριψη κληροδοτήματος συμβαίνει, όταν οι υποκλάσεις κληρονομούν τα δεδομένα των γονικών τους κλάσεων, όμως χρειάζονται μόνο μερικά από αυτά τα δεδομένα, ενώ τα υπόλοιπα μένουν αχρησιμοποίητα. Εάν η υποκλάση δεν υποστηρίζει τη διεπαφή της υπερκλάσης, η διεπαφή πρέπει να καταργηθεί. Εάν η κληρονομικότητα είναι κατάλληλη, μπορεί να δημιουργηθεί μία νέα αδερφική κλάση , στην οποία να δοθούν όλες οι μέθοδοι που δεν χρησιμοποιούνται.
- Σχόλια (Comments): Η πληθώρα σχολίων στον πηγαίο κώδικα μίας εφαρμογής μπορεί να αποτελεί ένδειξη ύπαρξης code smells, καθώς τα σχόλια λειτουργούν ως «κάλυμμα» σε ζητήματα που αφορούν κυρίως τον σχεδιασμό της. Όταν υπάρχει ανάγκη για σχόλια, συνήθως ο κώδικας πρέπει να αναδιαρθρώνεται ή να αναδιαμορφώνεται για να γίνει πιο διαισθητικός.

Σύμφωνα με του Mantyla et al. [22] τα code smells μπορούν να ταξινομηθούν σε 6 κατηγορίες:

- Bloaters: Τα bloaters παρουσιάζονται όταν ο πηγαίος κώδικας είναι εκτεταμένος με αποτέλεσμα να είναι ιδιαιτέρως δύσκολος ο χειρισμός και η διατήρησή του. Τα code smells που υπάγονται σε αυτή την κατηγορία είναι τα: Long Method, Large Class, Primitive Obsession, Long Parameter List και Data Clumps.
- Καταχραστές Αντικειμενοστρέφειας (Object-Orientation Abusers): Σε αυτή την κατηγορία περιλαμβάνονται οι περιπτώσεις κώδικα , στις οποίες δεν χρησιμοποιούνται πλήρως οι αρχές και οι δυνατότητες του αντικειμενοστραφούς προγραμματισμού. Τέτοια παραδείγματα αποτελούν η μη χρήση υποκλάσεων σε περιπτώσεις που αυτές θεωρούνται αναγκαίες, η παραβίαση της αρχής απόκρυψης πληροφοριών ή η κακή χρήση της κληρονομικότητας. Τα code smells που υπάγονται σε αυτή την κατηγορία είναι τα: Switch Statements, Temporary Field, Refused Bequest, Alternative Classes with Different Interfaces και Parallel Inheritance Hierarchies.
- Προλήπτες Αλλαγών (Change Preventers): Στην περίπτωση των Change Preventers η τροποποίηση του κώδικα μακροπρόθεσμα καθίσταται δυσκολότερη, καθώς παραβιάζεται ο κανόνας ότι οι πιθανές αλλαγές και οι κλάσεις πρέπει να έχουν σχέση ένα προς ένα. Τα code smells που υπάγονται σε αυτή την κατηγορία είναι τα: Divergent Change και Shotgun Surgery.
- Αναλώσιμος Κώδικας (Dispensables): Τα code smells αυτής της κατηγορίας αντιπροσωπεύουν κώδικα, ο οποίος δεν είναι απαραίτητος είτε επειδή δεν χρησιμοποιείται είτε επειδή δεν ικανοποιεί πλήρως τις απαιτήσεις των χρηστών και πρέπει να αφαιρεθεί. Τα code smells που υπάγονται σε αυτή την κατηγορία είναι τα: Lazy Class, Data Class, Duplicate Code και Speculative Generality.
- Ενθυλακωτές (Encapsulators): Οι ενθυλακωτές αφορούν προβλήματα σχετικά με την επικοινωνία μεταξύ διαφορετικών αντικειμένων και με τον τρόπο που τα αντικείμενα αυτά έχουν πρόσβαση στα δεδομένα. Τα code smells που υπάγονται σε αυτή την κατηγορία είναι τα: Message Chains και Middle Man.
- Συζευκτές (Couplers): Οι συζευκτές σχετίζονται με περιπτώσεις υψηλής σύζευξης, οι οποίες αντίκεινται στις αρχές του αντικειμενοστραφούς

σχεδιασμού. Τα code smells που υπάγονται σε αυτή την κατηγορία είναι τα: Feature Envy και Inappropriate Intimacy.

2.1.4 Anti-patterns

Σύμφωνα με τους Brown et. al. [23] ο όρος anti-pattern αναφέρεται σε πρότυπα ανάπτυξης λογισμικού που θεωρούνται κακές πρακτικές προγραμματισμού, καθώς προκαλούν αρνητικές συνέπειες. Τα anti-patterns ταξινομούνται σε 4 κατηγορίες [24]:

- Anti-patterns Ανάπτυξης (Development Anti-patterns): Αφορούν τεχνικά προβλήματα και λύσεις που αντιμετωπίζουν οι προγραμματιστές.
- Anti-patterns Αρχιτεκτονικής (Architectural Anti-patters): Αφορούν κοινά προβλήματα στον τρόπο δόμησης των υπολογιστικών συστημάτων.
- Περιβαλλοντικά Anti-patterns (Environmental Anti-patterns): Αφορούν προβλήματα που προέρχονται από μία κυρίαρχη δομή ή κάποιο κοινωνικό μοντέλο, τα οποία είναι αποτέλεσμα κοινωνικοπολιτικών δυνάμεων.
- Διαχειριστικά Anti-patterns (Managerial Anti-patterns): Αφορούν προβλήματα που οφείλονται στον τρόπο που διαχειρίστηκαν διάφορα θέματα τα διευθυντικά στελέχη μίας εταιρείας ή ενός οργανισμού που δραστηριοποιείται στον τομέα της ανάπτυξης λογισμικού.

Αξίζει να σημειωθεί ότι τα anti-patterns σε αντίθεση με τα code smells, δεν αφορούν μόνο λάθη ή ελλείψεις σε επίπεδο πηγαίου κώδικα, αλλά και σε σχεδιαστικό επίπεδο, κάνοντάς τα συχνά δύσκολα ή ακόμα και αδύνατα ως προς την επίλυσή τους.

2.1.5 Μηχανική Μάθηση

Ως Μηχανική Μάθηση (Machine Learning) ορίζεται η ικανότητα ενός υπολογιστή να προσαρμόζεται σε νέες περιστάσεις και να εντοπίζει ή να συμπεραίνει πρότυπα. Ο τομέας της μηχανικής μάθησης διακρίνει συνήθως τρεις περιπτώσεις:

- Την Επιβλεπόμενη Μάθηση (Supervised Learning) που περιλαμβάνει την μάθηση μίας συνάρτησης από παραδείγματα των εισόδων και των εξόδων του πράκτορα. Χρησιμοποιείται σε προβλήματα: Ταξινόμησης (Classification), Πρόγνωσης (Prediction) και Διερμηνείας (Interpretation).
- Την Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning) που περιλαμβάνει μάθηση προτύπων εισόδου, χωρίς να παρέχονται συγκεκριμένες τιμές εξόδου. Χρησιμοποιείται σε προβλήματα: Ανάλυσης Συσχετισμών (Association Analysis) και Ομαδοποίησης (Clustering).

- Την Ενισχυτική Μάθηση (Reinforcement Learning), όπου ο αλγόριθμος μαθαίνει μια στρατηγική ενεργειών μέσα από άμεση αλληλεπίδραση με το περιβάλλον. Χρησιμοποιείται κυρίως σε προβλήματα Σχεδιασμού (Planning).

Για κάθε πρόβλημα προς επίλυση στον χώρο της Μηχανικής Μάθησης υπάρχει ένας κατάλληλος τρόπος μάθησης και για κάθε τρόπο μάθησης υπάρχει τουλάχιστον ένας αλγόριθμος που μπορεί να χρησιμοποιηθεί [25].

2.1.5.1 Μετρικές Αξιολόγησης Αλγορίθμων Μηχανικής Μάθησης

Για την αξιολόγηση αλγορίθμων μηχανικής μάθησης υπάρχουν μετρικές που χρησιμοποιούνται, ώστε να είναι δυνατή η σύγκριση μεταξύ διαφορετικών προσεγγίσεων.

Η μήτρα σύγχυσης (confusion matrix) είναι ένας πίνακας που απεικονίζει την απόδοση ενός αλγόριθμου (εποπτευόμενης μάθησης). Κάθε στήλη του πίνακα αντιπροσωπεύει τις περιπτώσεις σε μια προβλεπόμενη κατηγορία, ενώ κάθε σειρά αντιπροσωπεύει τις περιπτώσεις σε μια πραγματική κατηγορία (ή αντίστροφα). Το όνομα προέρχεται από το γεγονός ότι είναι εύκολο να δούμε αν το σύστημα συγχέει τη μια κατηγορία με την άλλη. Στην Εικόνα 1 παρουσιάζεται μια Μήτρα Σύγχυσης.

- Ως Αληθώς Θετικός (True Positive) ορίζεται ο πραγματικός αριθμός των θετικών περιπτώσεων στα δεδομένα που προβλέφθηκαν ως θετικές.

	Πραγματική Κατηγορία	
Προβλεπόμενη Κατηγορία	Αληθώς Θετικό (True Positive)	Ψευδώς Θετικό (False Positive)
	Ψευδώς Αρνητικό (False Negative)	Αληθώς Αρνητικό (True Negative)

Εικόνα 1: Μήτρα Σύγχυσης

- Ως Ψευδώς Θετικός (False Positive) ορίζεται ο πραγματικός αριθμός των θετικών περιπτώσεων στα δεδομένα που προβλέφθηκαν ως αρνητικές.

- Ως Αληθώς Αρνητικός (True Negative) ορίζεται ο πραγματικός αριθμός των αρνητικών περιπτώσεων στα δεδομένα που προβλέφθηκαν ως αρνητικές.
- Ως Ψευδώς Αρνητικός (False Negative) ορίζεται ο πραγματικός αριθμός των αρνητικών περιπτώσεων στα δεδομένα που προβλέφθηκαν ως θετικές.

Όπως παρουσιάζεται στον Πίνακα 1, με βάση την μήτρα σύγχυσης υπολογίζονται οι τέσσερις (4) βασικές μετρικές αξιολόγησης αλγορίθμων εποπτευόμενης μάθησης. Οι μετρικές αυτές είναι οι : Ορθότητα (Accuracy), Ακρίβεια (Precision), Ανάκληση (Recall) και F1 Score.

Πίνακας 1: Μετρικές Αξιολόγησης Αλγορίθμων Εποπτευόμενης Μάθησης

Μετρική	**Τύπος**
Accuracy	$\frac{True\ Positive+True\ Negative}{True\ Positive+True\ Negative+False\ Positive+False\ Negative}$
Precision	$\frac{True\ Positive}{True\ Positive+False\ Positive}$
Recall	$\frac{True\ Positive}{True\ Positive+False\ Negative}$
F1 Score	$\frac{2*Precision*Recall}{Precision+Recall}$

2.2 Βιβλιογραφική Επισκόπηση

Όπως προαναφέρθηκε τα τελευταία χρόνια όλο και περισσότερες μελέτες παρουσιάζουν διαφορετικές προσεγγίσεις που χρησιμοποιούν αλγορίθμους μηχανικής μάθησης με στόχο την ανάπτυξη εργαλείων ανίχνευσης code smells και anti-patterns.

Το 2005 ο Kreimer J. [11] εισήγαγε ένα αυτοματοποιημένο σύστημα εντοπισμού ελαττωμάτων σχεδιασμού (anti-patterns) σε αντικειμενοστραφές λογισμικό συνδυάζοντας γνωστές προσεγγίσεις βασισμένες σε αντικειμενοστραφείς μετρήσεις με τεχνικές μηχανικής μάθησης και πρότεινε μια προσαρμοστική και μαθησιακή προσέγγιση.

Το 2010, οι Hassaine S. et. al. [12] παρουσίασαν τον πρώτο συστηματικό παραλληλισμό μεταξύ τεχνητού ανοσοποιητικού συστήματος και ανίχνευσης τεχνικών κακού σχεδιασμού. Το σύστημα τους παρουσίασε σχετικά καλό precision και recall όσον αφορά την εύρεση παρόμοιων αλλά όχι πανομοιότυπων κλάσεων.

Ένα χρόνο αργότερα οι Maneerat N. et. al. [13] χρησιμοποιώντας 7 διαφορετικά σύνολα δεδομένων, τα οποία αποτελούνταν από 27 μετρικές σχεδιασμού λογισμικού και περιλάμβαναν 7 είδη code smells (Feature Envy, Long Method, Long Perimeter Lists, Lazy Class, Message Chains, Middle Man και Switch Statements), εκπαίδευσαν και αξιολόγησαν επτά διαφορετικούς αλγορίθμους μηχανικής μάθησης (Naive Bayes, Logistic Regression, IB1, IBk, VF1, J48 και Random Forest). Έδειξαν ότι η μεθοδολογία που ακολούθησαν έχει εγγύτητα σε πραγματικές συνθήκες. Επιπλέον, έδειξαν ότι οι αλγόριθμοι Naive Bayes, VFI και J48, σε αντίθεση με τους αλγορίθμους Logistic regression, IBI, IBk και Random Forest, δεν είναι κατάλληλοι για ανίχνευση code smells, καθώς ο μέσος όρος των ποσοστών πρόβλεψης που πέτυχαν ήταν χαμηλός (<90%).

Την ίδια χρονιά , σε μία άλλη έρευνα, χρησιμοποιήθηκαν δίκτυα πεποιθήσεων Bayes (Bayesian belief networks) για την ανάπτυξη της προσέγγισης BDTEX [14] με στόχο την ανίχνευση τριών τύπων anti-patterns (Blob, Functional Decomposition & Spaghetti Code) σε δύο έργα Java. Έπειτα έγινε σύγκριση του εργαλείου που βασίζεται στην μέθοδο BDTEX με ένα εργαλείο που βασίζεται στην στη μέθοδο DECOR και χρησιμοποιήθηκε ως εργαλείο αναφοράς.

Λίγα χρόνια αργότερα, οι Azadi, U., et. al. [35] παρουσίασαν μια ημι-αυτοματοποιημένη ροή εργασίας μέσω του WekaNose, η οποία στοχεύει στην ανίχνευση code smells με τη χρήση τεχνικών μηχανικής μάθησης. Όπως αναφέρουν, το κύριο πλεονέκτημα αυτής της διαδικασίας είναι ότι χρησιμοποιούνται επιλεγμένοι κανόνες ή

επεξεργασμένοι αλγόριθμοι που μπορούν να διακρίνουν την ύπαρξη code smells ή όχι, όχι μόνο με υποκειμενικό τρόπο, αλλά με τρόπο που βασίζεται στην εμπειρία.

Την ίδια χρονιά οι Liu, H. et. al. [28] πρότειναν πρώτοι μια νέα προσέγγιση βασισμένη στη βαθιά μάθηση για την ανίχνευση ενός από τα πιο κοινά code smells, του feature envy. Η βασική ιδέα αυτής της προσέγγισης είναι ότι τα βαθιά νευρωνικά δίκτυα και οι προηγμένες τεχνικές βαθιάς μάθησης θα μπορούσαν να επιλέγουν αυτοματοποιημένα τα χαρακτηριστικά του πηγαίου κώδικα (ειδικά τα χαρακτηριστικά κειμένου) για την ανίχνευση feature envy και θα μπορούσαν να δημιουργούν τη σύνθετη αντιστοίχιση μεταξύ τέτοιων χαρακτηριστικών και προβλέψεων. Επιπλέον, πρότειναν μια αυτοματοποιημένη προσέγγιση για τη δημιουργία ετικετοποιημένων δεδομένων εκπαίδευσης για τους ταξινομητές που βασίζονται σε νευρωνικά δίκτυα, η οποία δεν απαιτεί ανθρώπινη παρέμβαση. Τα αποτελέσματα αξιολόγησης σε εφαρμογές ανοιχτού κώδικα υποδηλώνουν ότι η προσέγγισή βελτίωνε σημαντικά τις προγενέστερες μεθόδους εντοπισμού των feature envy χαρακτηριστικών όσο και τους τους αντίστοιχους ταξινομητές code smells.

Έναν χρόνο αργότερα, οι Pecorelli, F. et. al. [2] πραγματοποίησαν μία εμπειρική μελέτη κατά την οποία συνέκριναν την απόδοση μίας προσέγγισης βασιζόμενης στη μηχανική μάθηση (χρησιμοποιήθηκε ο αλγόριθμος Naive Bayes) και μίας ευρετικής μεθόδου για την ανίχνευση code smells λαμβάνοντας υπόψη πέντε διαφορετικούς τύπους code smells. Για την ανάλυση χρησιμοποιήθηκε ένα σύνολο δεδομένων που αποτελείτο από 125 εκδόσεις δεκατριών συστημάτων λογισμικού ανοιχτού κώδικα και περιλάμβανε 17 μετρικές κώδικα και 11 είδη code smells. Τα δύο κεντρικά αποτελέσματα αυτής της έρευνας συνοψίζονται ως εξής:

1. Οι ευρετικές μέθοδοι επιτυγχάνουν οριακά καλύτερη απόδοση σε σχέση με τις διαφορετικές τεχνικές μηχανικής μάθησης, όσον αφορά την ανίχνευση code smells. Ωστόσο, υπάρχει ανάγκη για περεταίρω διερεύνηση καθώς και στις δύο περιπτώσεις οι μετρικές αξιολόγησης της απόδοσης παραμένουν χαμηλές.
2. Η ύπαρξη προβλημάτων στο σύνολο των δεδομένων προς ανάλυση, όπως για παράδειγμα η ανισορροπία ή η κακή ακρίβεια αυτών, κάνουν την εφαρμογή και των δύο προσεγγίσεων αρκετά δύσκολη.

Δεν αρκεί η αξιολόγηση των διαφορετικών προσεγγίσεων μόνο με βάση τα μέτρα αξιολόγησης Accuracy και F1 Score, λόγω της ανισορροπίας των δεδομένων. Προτείνεται,

λοιπόν, να ελέγχονται επικουρικά άλλες μετρικές όπως το Kappa του Cohen και οι δείκτες ROC και MCC.

Το 2019 οι Alkharabsheh K. et. al. [15], [10] πραγματοποίησαν ένα σύνολο πειραμάτων-χρησιμοποιώντας 28 ταξινομητές μηχανικής μάθησης που στόχευαν στην ανίχνευση ύπαρξης God Class. Αυτά διεξήχθησαν χρησιμοποιώντας ένα σύνολο δεδομένων που σχηματίστηκε από 12.587 κλάσεις 24ων συστημάτων λογισμικού, στα οποία επικυρώθηκαν χειροκίνητα 1.958 κλάσεις. Κατέληξαν, στο ότι οι περισσότεροι ταξινομητές που εξέτασαν πέτυχαν υψηλές επιδόσεις, με πρώτο τον Cat Boost. Επίσης, σε αντίθεση με προηγούμενες μελέτες [2], υποστήριξαν ότι η εξισορρόπηση δεδομένων δεν έχει καμία σημαντική επίδραση στην ακρίβεια της ανίχνευσης και συνεπώς ότι οι προσεγγίσεις που βασίζονται σε μηχανική μάθηση μπορούν να χρησιμοποιούνται με ασφάλεια σε πραγματικά σενάρια όπου τα δεδομένα είναι συνήθως μη-ισορροπημένα λόγω της εγγενούς φύσης των code smells.

Την ίδια χρονιά, οι Barbez, A. et. al. [27] διαπίστωσαν ότι αν και οι προγενέστερες έρευνες που αφορούν τεχνικές ανίχνευσης code smells και anti-patterns βοήθησαν τους προγραμματιστές στον εντοπισμό στοιχείων κώδικα που χρήζουν αναδόμησης, αυτές βασίζονται σε διαφορετικές πηγές πληροφοριών και έτσι, προσδιορίζουν διαφορετικά σύνολα που εμφανίζουν anti-patterns. Ως εκ τούτου, αγνοώντας είτε τις δομικές είτε τις ιστορικές πτυχές των συστημάτων λογισμικού, οι προγενέστερες προσεγγίσεις χάνουν ορισμένες πολύτιμες πληροφορίες που περιορίζουν τις επιδόσεις τους. Έτσι, πρότειναν το CAME (Convolutional Analysis of code Metrics Evolution), μια προσέγγιση βασισμένη στη βαθιά μάθηση που βασίζεται τόσο σε δομικές όσο και σε ιστορικές πληροφορίες για τον εντοπισμό anti-patterns. Πιο αναλυτικά, το CAME εκμεταλλεύεται μετρικές που αφορούν δομημένο κώδικα και χρησιμοποιεί έναν ταξινομητή συνελικτικού νευρωνικού δικτύου για να ανιχνεύσει την ύπαρξη ή μη anti-patterns (κυρίως God Class). Λίγο αργότερα, οι ίδιοι παρουσίασαν το εργαλείο SMAD [26] (SMart Aggregation of Anti-patterns Detectors), που βασίζεται σε μία τεχνική μηχανική μάθησης που χρησιμοποιεί πολλά εργαλεία ανίχνευσης για να παράγει μια βελτιωμένη πρόβλεψη ανίχνευσης anti-pattern βάσει των εσωτερικών κανόνες ανίχνευσής τους από έναν εύλογο αριθμό παραδειγμάτων εκπαίδευσης. Το SMAD επικεντρώνεται στον εντοπισμό δύο ειδών anti-patterns, των God Class και Feature Envy.

Το 2020, τους Pecorelli F. et. al. [16], έρχονται να συμπληρώσουν οι Alkharabsheh K. et. al. [15], καθώς διερεύνησαν πέντε προσεγγίσεις για τον μετριασμό των

προβλημάτων ανισορροπίας δεδομένων για να κατανοήσουν τον αντίκτυπό τους στις τεχνικές εντοπισμού code smells και anti-patterns που βασίζονται σε μηχανική μάθηση και αφορούν αντικειμενοστραφή συστήματα ή συστήματα που εφαρμόζουν το μοτίβο Model-View-Controller. Έδειξαν ότι η αποφυγή της εξισορρόπησης δεν επηρεάζει δραματικά την ακρίβεια. Οι υπάρχουσες τεχνικές εξισορρόπησης δεδομένων είναι ανεπαρκείς για την ανίχνευση code smell και ευθύνεται για τις χαμηλές επιδόσεις των προσεγγίσεων που βασίζονται στη μηχανική μάθηση. Επιπλέον, υποστήριξαν ότι είναι αναγκαία η ανάδειξη νέων μετρικών κώδικα για την εκμετάλλευση διαφορετικών χαρακτηριστικών λογισμικού και νέων τεχνικών για τον αποτελεσματικό συνδυασμό τους.

Το ίδιο έτος, σε μία άλλη έρευνα οι συγγραφείς παρουσίασαν μία προσέγγιση ικανή να παρέχει στις σύγχρονες εταιρείες ανάπτυξης λογισμικού τη δυνατότητα να χρησιμοποιούν παραδείγματα για να διδάξουν έναν αλγόριθμο να αναγνωρίζει παραβιάσεις των οδηγιών κώδικα/σχεδίασης και, συνεπώς, να αυξήσει τον αριθμό των αναθεωρήσεων που πραγματοποιούνται πριν από την κυκλοφορία ενός προϊόντος, με αποτέλεσμα την αυξημένη ποιότητα του τελικού λογισμικού [31]. Πιο αναλυτικά, κατασκεύασαν ένα εργαλείο ανάλυσης κώδικα που βασίζεται στη μηχανική μάθηση, μέσω του οποίοι οι προγραμματιστές λογισμικού και οι αρχιτέκτονες σε μεγάλες και μεσαίες εταιρείες μπορούν να χρησιμοποιήσουν μερικά παραδείγματα γραμμών πηγαίου κώδικα που παραβιάζουν τις αρχές ανάπτυξης κώδικα και σχεδίασης (έως 700 γραμμές κώδικα) για να εκπαιδεύσουν έναν ταξινομητή (δέντρο αποφάσεων) για να εντοπίσουν παρόμοιες παραβιάσεις στις βάσεις κωδικών τους (έως 3 εκατομμύρια γραμμές κώδικα). Κατάφεραν να επιτύχουν μέση ακρίβεια πάνω από 99% και μέσο F1-score ίσο με 0,80 για έργα ανοιχτού κώδικα χρησιμοποιώντας περίπου 40.000 γραμμές κώδικα για την εκπαίδευση του εργαλείου. Παρόμοιο αποτελέσματα πέτυχαν και στον χρησιμοποιώντας μόνο έως 700 γραμμές κώδικα ως σύνολο δεδομένων εκπαίδευσης.

Το 2021 οι Sharma, T., et. al. [18] αξιοποίησαν τα Συνελικτικά (Convolution Neural Networks) και τα Επαναλαμβανόμενα Νευρωνικά Δίκτυα (Recurrent Neural Networks) συναρτήσει μοντέλων αυτόματου κωδικοποιητή (autoencoder models), με στόχο τη διερεύνηση της σκοπιμότητας της εφαρμογής μοντέλων βαθιάς εκμάθησης για την ανίχνευση οσμών χωρίς εκτεταμένη μηχανική χαρακτηριστικών, και της δυνατότητας εφαρμογής της εκμάθησης μεταφοράς στο πλαίσιο της ανίχνευσης code smells, καταλήγοντας στο ότι τα προαναφερθέντα μοντέλα μπορούν να χρησιμοποιηθούν για την ανίχνευση code smell, αν και με διαφορετική απόδοση.

Την ίδια χρονιά οι Zhang, Y., & Dong, C. [34], παρουσίασαν μια νέα προσέγγιση που ονομάζεται MARS για την ανίχνευση των κλάσεων και των μεθόδων εγκεφάλου (brain classes/ methods). Το MARS βελτιώνει την υποβάθμιση της κλίσης χρησιμοποιώντας ένα βελτιωμένο υπολειμματικό δίκτυο. Αυξάνει την τιμή βάρους αυτών των σημαντικών μετρικών κώδικα για την επισήμανση code smells, εισάγοντας έναν μηχανισμό μετρικής προσοχής. Για την εκπαίδευση του MARS, χρησιμοποιήθηκε το σύνολο δεδομένων BrainCode, το οποίο περιλαμβάνει 270.000 δείγματα από 20 εφαρμογές πραγματικού κόσμου. Μέσα από συγκριτική μελέτη προέκυψε ότι η μέση ακρίβεια του MARS είναι 2,01 % υψηλότερη από αυτή των προϋπάρχουσων προσεγγίσεων.

Τέλος, οι Zhang, Y. et. al. [33], πρόσφατα, πρότειναν μια νέα προσέγγιση για την ανίχνευση code smells με βάση ένα μοντέλο βαθιάς μάθησης που ονομάζεται DeleSmell. Το DeleSmell συλλέγει τόσο δομικά χαρακτηριστικά μέσω του iPlasma [38] όσο και σημασιολογικά χαρακτηριστικά μέσω λανθάνουσας σημασιολογικής ανάλυσης και word2vec [39]. Το μοντέλο του DeleSmell περιλαμβάνει μια διακλάδωση Συνελικτικού Νευρωνικού Δικτύου (Convolutional Neural Network) και έναν κλάδο προσοχής Ανατροφοδοτούμενης Μονάδας με Πύλη (Gate Recurrent Unit). Η τελική ταξινόμηση διεξάγεται από μια Μηχανή Διανύσματος Υποστήριξης (Support Vector Machine).

3 Μεθοδολογία

Για την πραγματοποίηση του πειραματικού μέρους αυτής της διπλωματικής ακολουθήθηκε η κάτωθι μεθοδολογία με την οποία αποσκοπούμε στην αναζήτηση, διερεύνηση, ταξινόμηση, και επιλογή εργαλείων ανοιχτού κώδικα, που βασίζονται σε τεχνικές μηχανικής μάθησης. Ως μέρος της προαναφερθείσας μεθοδολογίας περιγράφουμε επίσης, τον εντοπισμό και επιλογή μονάδων λογισμικού για την εφαρμογή των εργαλείων μηχανικής μάθησης σε αυτά, με απώτερο σκοπό την παραγωγή μετρικών αποδοτικότητας και την εξαγωγή αποτελεσμάτων.

Αρχικά, η αναζήτηση των εργαλείων βασίστηκε σε βιβλιογραφική έρευνα που πραγματοποιήθηκε στο Google Scholar. Ως λέξεις κλειδιά εφαρμόστηκαν μεμονωμένα ή σε συνδυασμό οι όροι: “code smells”, “code smell”, “code bad smells”, “bad code smells” “bad smells”, “anomalies”, “anti-patterns”, “antipattern”, “design defect”, “design-smells”, “design flaw”, “machine learning”, “machine-learning-based”, “learning”, “ML”, “prediction”, “detection”, “identification”, “prediction model”, “model”, “tool”, “tools”, “software”, “software engineering”.

Στην συνέχεια, με την χρήση των προαναφερθέντων λέξεων κλειδιών, πραγματοποιήθηκε πρόσθετη αναζήτηση στο Διαδίκτυο. Η αναζήτηση πραγματοποιήθηκε τόσο στο Google, όσο και στην πλατφόρμα GitHub. Αυτή η αναζήτηση ανέδειξε αποτελέσματα για διαφορετικά εργαλεία που βρέθηκαν από πολλές πηγές: προηγούμενα ερευνητικά έργα, φόρουμ, αναρτήσεις ιστολογίου, ιστότοπους για σχετικά εμπορικά προϊόντα και άρθρα. Πρέπει να σημειωθεί ότι αυτή η αναζήτηση οδήγησε σε εργαλεία που στην πλειοψηφία τους χρησιμοποιούνται σε έρευνες που εντοπίστηκαν προηγουμένως.

Κάθε προτεινόμενο εργαλείο, βασισμένο σε Μηχανική Μάθηση καταγράφηκε ως πιθανό αποτέλεσμα. Συνολικά βρέθηκαν 11 πιθανά εργαλεία ανίχνευσης code smells και anti-patterns που βασίζονται σε μηχανική μάθηση και παρουσιάζονται στον Πίνακα 2.

Πίνακας 2: Εργαλεία ανίχνευσης code smells και anti-patterns που βασίζονται σε μηχανική μάθηση.

Α/Α	**Όνομα Εργαλείου**	**Πηγαίος Κώδικας**

1	SMAD [26]	https://github.com/antoineBarbez/SMAD
2	CAME [27]	https://github.com/antoineBarbez/CAME
3	FeatureEnvy [28]	https://github.com/liuhuigmail/FeatureEnvy
4	ML_CloneValidationFramework [29]	https://github.com/pseudoPixels/ML_CloneValidationFramework
5	Ccflex [30], [31]	https://github.com/mochodek/py-ccflex
6	Deep Smell Detection [32]	https://github.com/liuhuigmail/DeepSmellDetection
7	DeleSmell [33]	N/A
8	MARS [34]	N/A

9	WekaNose [35]	https://github.com/uazadi/WekaNose
10	Deep-Multimodal-Architectures-Code-Smell-Classification	https://github.com/anushka-code/Deep-Multimodal-Architectures-Code-Smell-Classification
11	ML-code-smell-CSharp [41]	https://github.com/Clean-CaDET/ML-code-smell-CSharp

Προκειμένου να πραγματοποιηθεί η ανάλυση των επιλεγμένων εργαλείων ανίχνευσης code smells, επιλέχθηκαν εφαρμογές Android για τη δοκιμή τους. Τα κριτήρια για τις επιλογές εφαρμογών ήταν ότι οι εφαρμογές έπρεπε να είναι ανοιχτού κώδικα. Ο πηγαίος κώδικας έπρεπε να είναι ελεύθερα προσβάσιμος για να μπορεί να εκτελεί τους ανιχνευτές code smell. Οι επιλεγμένες εφαρμογές έπρεπε να είναι διαφορετικών μεγεθών για να επιτρέπεται η σύγκριση για την ανάλυση. Ωστόσο, δεν επιλέχθηκαν σημαντικά έργα, καθώς η ανάλυση θα ήταν πολύ εκτεταμένη. Επιπλέον, ήταν σημαντικό όλες οι εφαρμογές που επιλέχθηκαν να διαφέρουν μεταξύ τους στην περιγραφή τους, προκειμένου να διασφαλιστεί ότι ο κώδικας που αναλύθηκε ήταν διαφορετικός. Οι εφαρμογές Android επιλέχθηκαν από την πλατφόρμα F-Droid [36], η οποία είναι ένα αποθετήριο για δωρεάν λογισμικό ανοιχτού κώδικα για την πλατφόρμα Android. Ήταν σημαντικό ότι οι εφαρμογές είχαν σχετικά πρόσφατες ενημερώσεις, καθώς έπρεπε να εκτελούνται. Επιλέχθηκαν 3 εφαρμογές, από το αποθετήριο F-Droid, που θα χρησιμοποιηθούν στην παρούσα διπλωματική εργασία και περιγράφονται στον Πίνακα 3.

Πίνακας 3: Περιγραφή των επιλεγμένων εφαρμογών

Όνομα Εφαρμογής	**Περιγραφή**	**Μέγεθος**

Notepad	Εφαρμογή καταγραφής σημειώσεων.	Μικρή
Personal Stuff	Εφαρμογή παρακολούθησης, διαχείρισης και υπενθύμισης προσωπικών θεμάτων.	Μεσαία
Wikipedia	Επίσημο πρόγραμμα προβολής της δωρεάν διαδικτυακής εγκυκλοπαίδειας Wikipedia.	Μεγάλη

Πρέπει να σημειωθεί ότι οι εφαρμογές επιλέχθηκαν ώστε να έχουν διαφορές στο μέγεθος τους, ως προς τον αριθμό αρχείων και τον αριθμό γραμμών.

Η εφαρμογή Notepad αποτελείται από 52 αρχεία, τα οποία εμπεριέχουν συνολικά 2.717 γραμμές, εκ των οποίων οι 284 είναι κενές, οι 309 είναι σχόλια και οι 1.953 είναι κώδικας. Στο Πίνακα 4 παρουσιάζεται η κατανομή αρχείων και γραμμών ανά γλώσσα προγραμματισμού για την εφαρμογή Notepad.

Πίνακας 4: Η κατανομή αρχείων και γραμμών ανά γλώσσα προγραμματισμού για την εφαρμογή Notepad

Γλώσσα Προγραμματισμού	**Αριθμός Αρχείων**	**Αριθμός γραμμών κώδικα**	**Αριθμός γραμμών σχολίων**	**Αριθμός κενών γραμμών**	**Συνολικός αριθμός γραμμών εφαρμογής**
Java	11	1,030	260	284	1,574
XML	34	810	27	138	975
Groovy	3	47	1	8	56
Markdown	1	38	0	12	50

YAML	1	23	6	8	37
Properties	2	5	15	5	25

Η εφαρμογή Personal Stuff αποτελείται από 266 αρχεία, τα οποία εμπεριέχουν συνολικά 37.074 γραμμές, εκ των οποίων οι 2.039 είναι κενές, οι 450 είναι σχόλια και οι 37.074 είναι κώδικας. Στο Πίνακα 5 παρουσιάζεται η κατανομή αρχείων και γραμμών ανά γλώσσα προγραμματισμού για την εφαρμογή Personal Stuff.

Πίνακας 5: Η κατανομή αρχείων και γραμμών ανά γλώσσα προγραμματισμού για την εφαρμογή Personal Stuff.

Γλώσσα Προγραμματισμού	**Αριθμός Αρχείων**	**Αριθμός γραμμών κώδικα**	**Αριθμός γραμμών σχολίων**	**Αριθμός κενών γραμμών**	**Συνολικός αριθμός γραμμών εφαρμογής**
HTML	1	17,685	76	8	17,769
Java	139	9,942	334	1,693	11,969
XML	104	5,256	14	258	5,528
JSON	5	786	0	0	786
YAML	4	556	0	15	571
Groovy	9	308	7	53	368

Markdown	2	43	0	11	54
Properties	2	9	19	1	29

Η εφαρμογή Wikipedia αποτελείται από 804 αρχεία, τα οποία εμπεριέχουν συνολικά 146.407 γραμμές, εκ των οποίων οι 3.548 είναι κενές, οι 2.590 είναι σχόλια και οι 140.269 είναι κώδικας. Στο Πίνακα 6 παρουσιάζεται η κατανομή αρχείων και γραμμών ανά γλώσσα προγραμματισμού για την εφαρμογή Wikipedia.

Πίνακας 6: Η κατανομή αρχείων και γραμμών ανά γλώσσα προγραμματισμού για την εφαρμογή Wikipedia.

Γλώσσα Προγραμματισμού	**Αριθμός Αρχείων**	**Αριθμός γραμμών κώδικα**	**Αριθμός γραμμών σχολίων**	**Αριθμός κενών γραμμών**	**Συνολικός αριθμός γραμμών εφαρμογής**
XML	691	114,241	2,379	2,942	119,562
JSON	60	23,423	0	6	23,429
Java	26	1,410	69	375	1,854
Python	5	445	114	131	690
Groovy	6	295	21	55	371
SQL	2	286	0	2	288

YAML	5	135	0	17	152
Properties	5	16	5	4	25
Shell Script	2	10	2	7	19
Markdown	2	8	0	9	17

Οι γραμμές κώδικα για τις εφαρμογές υπολογίστηκαν χρησιμοποιώντας το εργαλείο VS Code Counter [37].

Τέλος, η ανάλυση των επιλεγμένων εργαλείων ανίχνευσης code smell και anti-patterns γίνετε σε δύο μέρη. Αρχικά γίνεται μία γενική σύγκριση των επιδόσεων του συνόλου των ανιχνευτών και έπειτα πραγματοποιείται σύγκριση των αποτελεσμάτων που πέτυχε κάθε ανιχνευτής ξεχωριστά για τις τρεις επιλεγμένες εφαρμογές Android.

4 Αποτελέσματα

Αυτή η ενότητα παρουσιάζει τα αποτελέσματα του πειραματικού μέρους της διπλωματικής εργασίας. Αρχικά, παρουσιάζονται τα συνολικά 11 εργαλεία που εντοπίστηκαν. Στην συνέχεια, περιγράφεται ο τρόπος επιλογής των εργαλείων που επιλέχθηκαν για περεταίρω διερεύνηση. Εν συνεχεία, παρουσιάζονται τα συλλογικά αποτελέσματα αξιολόγησης της χρηστικότητάς των τεσσάρων (4) εργαλείων που τελικώς επιλέχθηκαν. Τέλος, παρατίθενται και αξιολογούνται τα επιμέρους αποτελέσματα για κάθε επιλεγμένο ανιχνευτή code smell ή anti-pattern κατηγοριοποιημένα ως προς το είδος των παραβιάσεων που εντοπίζουν.

4.1 Περιγραφή εργαλείων

4.1.1 SMAD

Το SMAD [26] είναι μια μέθοδος συνόλου που βασίζεται στη μηχανική μάθηση για τη συγκέντρωση διαφόρων προσεγγίσεων ανίχνευσης anti-patterns με βάση τον εσωτερικό κανόνα ανίχνευσης. Στόχος του SMAD είναι να δημιουργήσει έναν βελτιωμένο ταξινομητή σε σχέση με της αυτόνομες προσεγγίσεις.

Η βασική ιδέα πίσω από το SMAD είναι ο υπολογισμός των βασικών μετρήσεων διαφόρων προσεγγίσεων για κάθε οντότητα εισόδου και η χρήση αυτών των μετρήσεων για την τροφοδοσία του ταξινομητή που βασίζεται στο ML. Ο ταξινομητής που χρησιμοποιείται από το SMAD για να προβλέψει εάν μια οντότητα είναι anti-pattern ή όχι είναι ένα Multi-Layer Perceptron (MLP), δηλαδή ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο προώθησης τροφοδοσίας. Για την εκπαίδευση και την αξιολόγηση του SMAD δημιουργήθηκε μία βάση δεδομένων που περιείχε τα περιστατικά God Class και Feature Envy σε οκτώ συστήματα ανοιχτού κώδικα.

Τα βασικά αποτελέσματα των πειραμάτων που διενεργήθηκαν για την αξιολόγηση του SMAD από τους δημιουργούς του, έδειξαν ότι αυτό παρουσιάζει καλύτερα αποτελέσματα όσον αφορά τον εντοπισμό των God Class και Feature Envy τόσο συγκριτικά με άλλα εργαλεία, όσο και με ανταγωνιστικές μεθόδους συνόλου

4.1.2 CAME

Το CAME (Convolutional Analysis of code Metrics Evolution) [27] κατασκευάστηκε από τους Barbez, A., Khomh, F., & Guéhéneuc, Y. G. Παρουσιάστηκε πρώτη φορά το 2019 στο συνέδριο International Conference on Software Maintenance and Evolution (ICSME) της IEEE. Πρόκειται για ένα εργαλείο ανίχνευσης code smells και anti-patterns που βασίζεται σε βαθιά μάθηση. Το CAME χρησιμοποιεί μια αρχιτεκτονική Συνελικτικού Νευρωνικού Δικτύου για τον εντοπισμό anti-patterns τόσο από δομικές όσο και από ιστορικές πληροφορίες που εξάγονται από συστήματα ελέγχου έκδοσης (π.χ. Git). Χρησιμοποιείτε για την ανίχνευση του God Class.

Σύμφωνα με τους κατασκευαστές του, για την εκπαίδευση και αξιολόγηση του CAME χρησιμοποιήθηκε ένα σύνολο δεδομένων που περιείχε anti-patterns από 8 έργα ανοιχτού κώδικα γραμμένα σε Java. Για την επαλήθευση του CAME αξιοποιήθηκαν ιστορικές πληροφορίες σχετικά με τις μετρήσεις κώδικα εφαρμογών με διαφορετικά μήκη ιστορικού (δηλαδή, τον αριθμό των αναθεωρήσεων). Το CAME πέτυχε κατά μέσο όρο 71% precision, 86% recall και 77% F1-score.

4.1.3 Feature Envy

Το εργαλείο Feature Envy [28] παρουσιάστηκε για πρώτη φορά το 2018 στο 33ο διεθνές συνέδριο των ACM και IEEE με τίτλο Automated Software Engineering από τους Liu, H., Xu, Z. και Zou, Y. Πρόκειται για ένα εργαλείο βασισμένο στη βαθιά μάθηση που στοχεύει στην ανίχνευση ενός από τα πιο κοινά code smells, το feature envy.

Η βασική ιδέα αυτής της προσέγγισης, είναι ότι τα βαθιά νευρωνικά δίκτυα και οι προηγμένες τεχνικές βαθιάς μάθησης θα μπορούσαν να επιλέγουν αυτοματοποιημένα τα χαρακτηριστικά του πηγαίου κώδικα (ειδικά τα χαρακτηριστικά κειμένου) για την ανίχνευση Feature Envy και θα μπορούσαν να δημιουργούν τη σύνθετη αντιστοίχιση μεταξύ τέτοιων χαρακτηριστικών και προβλέψεων. Το Feature Envy αξιολογήθηκε σε εφαρμογές ανοιχτού κώδικα και πέτυχε κατά μέσο όρο 39,79% precision, 79,27% recall και 52,98% F1-score.

4.1.4 ML_CloneValidationFramework

Το ML_CloneValidationFramework [29] χρησιμοποιεί μοντέλα μηχανικής μάθησης για την πρόβλεψη της επικύρωσης κλώνου κώδικα για έναν συγκεκριμένο χρήστη. Για τη χρήση του ML_CloneValidationFramework, αυτό θα πρέπει αρχικά να εκπαιδευτεί από τον χρήστη με βάση τα μη αυτόματα επικυρωμένα σύνολα κλώνων κώδικα. Στη συνέχεια, τα εκπαιδευμένα μοντέλα χρησιμοποιούνται για τη βελτίωση των αναφερόμενων κλώνων κώδικα από εργαλεία ανίχνευσης κλώνων, προβλέποντας τα ειδικά μοτίβα επικύρωσης του χρήστη. Το ML_CloneValidationFramework επιπλέον παρέχει μια αρχιτεκτονική βασισμένη στο σύννεφο (cloud) για την αξιοποίηση των πλεονεκτημάτων υπολογισμού υψηλής απόδοσης και συμβατότητας με οποιοδήποτε από τα υπάρχοντα εργαλεία ανίχνευσης κλώνων κώδικα.

4.1.5 CCflex

Το εργαλείο CCFlex (Flexible Code Counter/Classifier) [30], [31] σχεδιάστηκε αρχικά για την εκτέλεση μέτρησης μεγέθους λογισμικού, ωστόσο αργότερα, προσαρμόστηκε ώστε να εντοπίζει παραβιάσεις των θεμελιωδών αρχών σχεδιασμού λογισμικού σε έναν πηγαίο κώδικα. Η υλοποίησή του πραγματοποιήθηκε σε συνεργασία με δύο μεγάλες Σκανδιναβικές εταιρείες που δραστηριοποιούνται στον τομέα ανάπτυξης λογισμικού.

Το CCFlex βασίζεται σε μία αρχιτεκτονική σωλήνων και φίλτρων (pipes-and-filters), βάση της οποίας χρησιμοποιούνται μια σειρά από ανεξάρτητα στοιχεία (φίλτρα) που θα μπορούσαν να οργανωθούν σε αγωγούς επεξεργασίας.

Εκτός από τα φίλτρα, το CCFlex παρέχει μια σουίτα μεμονωμένων εργαλείων που επιτρέπουν την εκτέλεση ανεξάρτητων εργασιών, όπως η επιλογή γραμμών για επισήμανση χρησιμοποιώντας το Active Learning (AL) ή ο έλεγχος της συνέπειας των ετικετών γραμμής που παρέχονται από έναν άνθρωπο (δηλαδή, αναγνώριση γραμμών που έχουν παρόμοια αναπαράσταση όσον αφορά τα χαρακτηριστικά τους, αλλά διαφορετικές ετικέτες).

Εξαιτίας των παραπάνω, το CCFlex είναι ικανό να αναγνωρίσει παραβιάσεις των κατευθυντήριων γραμμών κώδικα χρησιμοποιώντας χαρακτηριστικά που εξάγονται απευθείας από το κείμενο χωρίς την ανάγκη ανάλυσης ή μεταγλώττισης του κώδικα.

Επιπλέον, μπορεί να εκπαιδευτεί ώστε να αναγνωρίζει παραβιάσεις διαφορετικών οδηγιών κωδικοποίησης και διαφορετικών γλωσσών προγραμματισμού (C, C++ και Java).

Η βέλτιστη μέση απόδοση που έχει επιτύχει το CCFlex όσον αφορά τα F1-Score και Recall είναι 0,78 και 0,97 αντίστοιχα. Τα καλύτερα αποτελέσματα λήφθηκαν για τους κανόνες που απαιτούν την κατανόηση του περιβάλλοντος μιας γραμμής, ενώ οι κανόνες που απαιτούσαν την κατανόηση του περιβάλλοντος πολλών γραμμών ήταν πολύ πιο δύσκολο να εκπαιδευτούν.

4.1.6 Deep Smell Detection

Το Deep Smell Detection είναι ένα εργαλείο ανίχνευσης code smells που βασίζεται σε αλγορίθμους μηχανικής μάθησης. Το Deep Smell Detection διατίθεται ως λογισμικό ανοιχτού κώδικα μέσω της πλατφόρμας GitHub. Οι χρήστες που έχουν συνεισφέρει στην ανάπτυξή του είναι οι TyrantJin και liuhuigmail. Η πρώτη έκδοση του Deep Smell Detection δημοσιεύθηκε τον Απρίλιο του 2019, ενώ η πιο πρόσφατη τον Δεκέμβριο του 2021. Δεν υπάρχει δημοσίευση που να πραγματοποιείται αξιολόγηση του εργαλείου.

4.1.7 DeleSmell

Το DeleSmell [33] στοχεύει στην ανίχνευση code smells με βάση ένα μοντέλο βαθιάς μάθησης. Το μοντέλο του DeleSmell περιλαμβάνει μια διακλάδωση Συνελικτικού Νευρωνικού Δικτύου (CNN) και ενός κλάδου προσοχής Ανατροφοδοτούμενης Μονάδας με Πύλη (GRU). Εντός του DeleSmell η τελική ταξινόμηση διεξάγεται από μια Μηχανή Διανύσματος Υποστήριξης (SVM). Το DeleSmell συλλέγει τόσο δομικά χαρακτηριστικά μέσω του iPlasma [38] όσο και σημασιολογικά χαρακτηριστικά μέσω λανθάνουσας σημασιολογικής ανάλυσης και word2vec [39].

Για την εκπαίδευση των μοντέλων εντός του DeleSmell δημιουργήθηκε ένα σύνολο δεδομένων με εξαγωγή δειγμάτων από 24 έργα.

Τα πειραματικά αποτελέσματα δείχνουν ότι το DeleSmell βελτιώνει την ακρίβεια της ανίχνευσης μυρωδιών της Κλάσης Εγκεφάλου (BC) και της Μεθόδου Εγκεφάλου (BM) έως και 4,41%.

4.1.8 MARS

Το MARS [34] στοχεύει στην ανίχνευση BC και MC. Το MARS βελτιώνει την υποβάθμιση της κλίσης χρησιμοποιώντας ένα βελτιωμένο υπολειμματικό δίκτυο. Επιπλέον, αυξάνει την τιμή βάρους αυτών των σημαντικών κωδικών μετρήσεων για την επισήμανση δειγμάτων με code smells, εισάγοντας έναν μηχανισμό μετρικής προσοχής.

Για την εκπαίδευσης του MARS, παράχθηκε ένα σύνολο δεδομένων που ονομάζεται BrainCode. Αυτό αποτελείτο από περισσότερα από 270.000 δείγματα, που προέρχονται από 20 διαφορετικές εφαρμογές.

Σε σύγκριση με άλλες προσεγγίσεις που βασίζονται στη μηχανική μάθηση και στη βαθιά μάθηση, τα πειραματικά αποτελέσματα έδειξαν ότι η μέση ακρίβεια του MARS είναι 2,01 % υψηλότερη από αυτή των προηγούμενων προσεγγίσεων.

4.1.9 WekaNose

Το WekaNose δημιουργήθηκε από τους Umberto Azadi, Francesca Arcelli Fontana και Marco Zanoni. Παρουσιάστηκε για πρώτη φορά στο 40ο διεθνές συνέδριο στην Μηχανική Λογισμικού που πραγματοποιήθηκε στην Νέα Υόρκη. Πρόκειται για ένα ολοκληρωμένο εργαλείο ανίχνευσης code smells που βασίζεται σε τεχνικές μηχανικής μάθησης. Το WekaNose ανά περίπτωση επιλέγει κανόνες ή εκπαιδευμένους αλγόριθμους-βάση παραδειγμάτων, που μπορούν να ταξινομήσουν μια μέθοδο ή κλάση ως προς την παρουσία ή μη, code smells.

4.1.10 Deep-Multimodal-Architectures-Code-Smell-Classification

Το Deep-Multimodal-Architectures-Code-Smell-Classification είναι μία πολυτροπική προσέγγιση βαθιάς μάθησης που συνδυάζει δομικές και σημασιολογικές πληροφορίες για να ανιχνεύσει δύο code smells τις Long Parameter Lists και Switch Statements.

Η αρχιτεκτονική που ακολουθεί εφαρμόζει την εκμάθηση μεταφοράς DistilBERT για τη δημιουργία διανυσματικών ενσωματώσεων που αντιπροσωπεύουν κλάσεις και μεθόδους που συνδέονται με αριθμητικές μετρήσεις για κοινή εξαγωγή χαρακτηριστικών με χρήση CNN, για τη δημιουργία μιας σύνθετης αντιστοίχισης μεταξύ των χαρακτηριστικών και την πρόβλεψη της εξόδου ως code smell ή μη. Περιλαμβάνει επίσης

την υλοποίηση δύο αγωγών πολυτροπικής μηχανικής εκμάθησης, ο πρώτος χρησιμοποιεί ένα sci-kit [40] Learning TF-IDF Vectorizer με Random Forest Classifier και ο δεύτερος συγχωνεύει το CNN με το Bi-LSTM. Το Deep-Multimodal-Architectures-Code-Smell-Classification σύμφωνα με τους δημιουργούς του επιτυγχάνει ακρίβεια 91,2%.

4.1.11 ML-code-smell-CSharp

Το ML-code-smell-CSharp [41] εξετάζει την ενσωμάτωση νευρικού πηγαίου κώδικα τελευταίας τεχνολογίας CodeT5 για ανίχνευση οσμής κώδικα σε C#, πιο συγκεκριμένα εξετάζει και συγκρίνει δύο αναπαραστάσεις χαρακτηριστικών για την εκπαίδευση μοντέλων ML: (1) μετρήσεις κώδικα και (2) ενσωματώσεις CodeT5. Οι δημιουργοί του υποστηρίζουν ότι ενώ τα μοντέλα που εκπαιδεύτηκαν με μετρήσεις κώδικα έχουν καλύτερη απόδοση, η βελτίωση της απόδοσης σε σχέση με τα χαρακτηριστικά του CodeT5 είναι αμελητέα συνυπολογίζοντας τα πλεονεκτήματα της αυτόματης εξαγωγής λειτουργιών.

4.2 Επιλογή εργαλείων

Από τα 11 εργαλεία που βρέθηκαν, επιλέχθηκαν 4 ανιχνευτές για περαιτέρω ανάλυση. Οι ανιχνευτές που δεν επιλέχθηκαν και οι αιτίες αποκλεισμού τους παρουσιάζονται στον Πίνακα 7.

Πίνακας 7: Αιτίες αποκλεισμού μη επιλεγμένων ανιχνευτών

Όνομα Ανιχνευτή	**Αιτία Αποκλεισμού**
SMAD	Αδυναμία εκτέλεσης λόγω μη συμβατότητας.
ML_CloneValidation Famework	Αδυναμία εκτέλεσης λόγω μη συμβατότητας.

Ccflex	Αδυναμία εκτέλεσης λόγω μη συμβατότητας.
DeleSmell	Ο πηγαίος κώδικας δεν ήταν διαθέσιμος στο διαδίκτυο.
MARS	Ο πηγαίος κώδικας δεν ήταν διαθέσιμος στο διαδίκτυο.
Deep-Multimodal-Architectures-Code-Smell-Classification	Αδυναμία εκτέλεσης λόγω μη συμβατότητας.
ML-code-smell-CSharp	Η εφαρμογή υποστηρίζει την ανάλυση πηγαίου από έργα C#.

Τα εργαλεία που επιλέχθηκαν ήταν τα: CAME, FeatureEnvy, Deep Smell Detection και WekaNose.

4.3 Συλλογικά αποτελέσματα

Αρχικά, αναζητήθηκαν και διερευνήθηκαν τα εργαλεία εντοπισμού code smells και anti-patterns. Όπως παρουσιάζεται στον Πίνακα 8, αυτοί οι 4 επιλεγμένοι ανιχνευτές υποσχέθηκαν συλλογικά να ανιχνεύσουν 5 διαφορετικά code smells και anti-patterns. Πιο αναλυτικά, οι ανιχνευτές CAME και Feature Envy στοχεύουν στην ανίχνευση ενός μόνο είδους code smell, των large class και feature envy αντίστοιχα. Από την άλλη μεριά βλέπουμε ότι τα εργαλεία Deep Smell Detection και WekaNose έχουν αναπτυχθεί με τέτοιο τρόπο, ώστε να μπορούν να ανιχνεύσουν αντίστοιχα 4 και 5 διαφορετικά είδη code smell.

Επιπλέον, παρατηρείται ότι οι ανιχνευτές είναι ικανοί να ανταποκριθούν σε τέσσερις διαφορετικές κατηγορίες code smell (Bloaters, Couplers και Dispensables), με πιο συχνή την Bloaters, που σημαίνει ότι τα εργαλεία βρίσκουν πολύ μεγάλο κώδικα. Δεν εντοπίζονται code smells κατηγορίας Encapsulators, Change Preventers και Object-Orientation Abusers, συνεπώς δεν μπορούν να εντοπιστούν προβλήματα σχετικά με την

επικοινωνία μεταξύ διαφορετικών αντικειμένων, με τον τρόπο που τα αντικείμενα αυτά έχουν πρόσβαση στα δεδομένα, με μελλοντικές τροποποιήσεις κώδικα και με τη μη εφαρμογή των αρχών και των δυνατοτήτων του αντικειμενοστραφούς προγραμματισμού.

Όσον αφορά τους αλγορίθμους μηχανικής μάθησης που χρησιμοποιούνται, αξίζει να σημειωθεί ότι τα 3 εργαλεία χρησιμοποιούν αλγορίθμους βαθιάς μάθησης, ένα εκ των οποίων πραγματοποιεί Επεξεργασία Φυσικής Γλώσσας (NLP). Ο συχνότερος αλγόριθμος βαθιάς μάθησης που χρησιμοποιείται είναι αυτός των Συνελικτικών Νευρωνικών Δικτύων. (CNN).

Πίνακας 8: Επισκόπηση των code smells anti-patterns που βρέθηκαν από επιλεγμένους ανιχνευτές και των αλγορίθμων μηχανικής μάθησης που χρησιμοποιήθηκαν.

Όνομα Ανιχνευτή / Code smell	**CAME**	**Feature Envy**	**Deep Smell Detection**	**WekaNose**
Long Method			DenseNN	Boosted J48 Unpruned
Large Class	CNN		Dense LSTM	Naïve Bayes
Long Parameter List				Boosted J48 Unpruned
Feature Envy		CNN	CNN	Boosted JRip
Data Class			NLP	Boosted J48 pruned

4.4 Αξιολόγηση ανιχνευτών ανά είδος code smell

Οι επιλεγμένοι ανιχνευτές κώδικα code smells και anti-pattern δοκιμάστηκαν στις 3 επιλεγμένες εφαρμογές Android, με στόχο να αναλυθεί η χρηστικότητα και η χρησιμότητά τους στην εύρεση code smells και anti-patterns. Προκειμένου να μπορέσουν να υπολογιστούν οι μετρικές αξιολόγησης, πριν ξεκινήσει η εκπαίδευση, σε κάθε ένα από τα έργα Android πραγματοποιήθηκε χειροκίνητα ετικετοποίηση του πηγαίου κώδικα.

Πιο αναλυτικά, ακολουθώντας την μεθοδολογία των Ochodek, M., et.al. [30], [31] τέθηκαν συγκεκριμένες προϋποθέσεις για κάθε μορφή code smell και τοποθετήθηκαν στην αρχή και στο τέλος του πηγαίου κώδικα που παραβιάζει κάποιο από τα ποιοτικά χαρακτηρίστηκα λογισμικού τρείς χαρακτήρες ως εξής: <!@ Long Method @!>, <!# Large Class #!>, <!$ Long Parameter List $!>, <!% Feature Envy $!>, <!& Data Class &!>.

4.4.1 Long Method

Από τους επιλεγμένους ανιχνευτές τα Deep Smell Detection και WekaNose μπορούν να χρησιμοποιηθούν για την ανίχνευση μεγάλων μεθόδων. Στον Πίνακα 9 παρουσιάζονται τα αποτελέσματα αξιολόγησης για κάθε μία από τις τρεις επιλεγμένες εφαρμογές Android.

Πίνακας 9: Αποτελέσματα αξιολόγησης για την ανίχνευση μεγάλων μεθόδων.

	Deep Smell Detection			**WekaNose**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Notepad	75,0%	69,2%	72,0%	37,5%	75,0%	50,0%
Personal Stuff	40,0%	100,0%	57,1%	80,0%	66,7%	23,5%

Wikipedia	27,6%	72,7%	40,0%	14,3%	66,7%	23,5%
Μέσος Όρος	47,5%	80,6%	56,4%	43,9%	69,4%	48,8%

Όπως φαίνεται το εργαλείο Deep Smell Detection παρουσιάζει καλύτερα αποτελέσματα, όσον αφορά την ανίχνευση μεγάλων μεθόδων. Επιπλέον, παρατηρείται ότι και οι 2 ανιχνευτές μπορούν να ανταποκριθούν καλύτερα σε προγράμματα μικρότερου μεγέθους.

4.4.2 Large Class

Από τους επιλεγμένους ανιχνευτές τα CAME, Deep Smell Detection και WekaNose μπορούν να χρησιμοποιηθούν για την ανίχνευση μεγάλων κλάσεων. Στον Πίνακα 10 παρουσιάζονται τα αποτελέσματα αξιολόγησης για κάθε μία από τις τρεις επιλεγμένες εφαρμογές Android.

Πίνακας 10: Αποτελέσματα αξιολόγησης για την ανίχνευση μεγάλων κλάσεων.

	CAME			**Deep Smell Detection**			**WekaNose**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Notepad	33,3%	50,0%	40,0%	16,7%	50,0%	25,0%	25,0%	50,0%	33,3%
Personal Stuff	50,0%	25,0%	33,3%	75,0%	60,0%	66,7%	20,0%	25,0%	22,2%
Wikipedia	71,4%	38,5%	50,0%	66,7%	54,5%	60,0%	23,5%	44,4%	30,8%
Μέσος Όρος	51,6%	37,8%	41,1%	52,8%	54,8	60,0%	22,8%	39,8%	28,8%

Όπως φαίνεται το εργαλείο Deep Smell Detection παρουσιάζει καλύτερα αποτελέσματα, όσον αφορά την ανίχνευση μεγάλων κλάσεων. Επιπλέον, παρατηρείται ότι το μέγεθος των προγραμμάτων δεν επηρεάζει ιδιαίτερα το αποτέλεσμα της αξιολόγησης.

4.4.3 Long Parameter List

Από τους επιλεγμένους ανιχνευτές μόνο ο WekaNose μπορεί να χρησιμοποιηθεί για την ανίχνευση ύπαρξης μεγάλης λίστας παραμέτρων. Στον Πίνακα 11 παρουσιάζονται τα αποτελέσματα αξιολόγησης του WekaNose για κάθε μία από τις τρεις επιλεγμένες εφαρμογές Android.

Πίνακας 11: Αποτελέσματα αξιολόγησης για την ανίχνευση μεγάλης λίστας παραμέτρων.

	WekaNose		
	Precision	**Recall**	**F1-Score**
Notepad	20,0%	50,0%	28,6%
Personal Stuff	Τα True Positive ισούνταν με 0.		
Wikipedia	83,3%	41,7%	55,6%
Μέσος Όρος	51,65%	45,85%	42,1%

Όπως φαίνεται το εργαλείο WekaNose παρουσιάζει καλύτερα αποτελέσματα, σε ότι αφορά προγράμματα μεγάλου μεγέθους.

4.4.4 Feature Envy

Από τους επιλεγμένους ανιχνευτές τα Feature Envy, Deep Smell Detection και WekaNose μπορούν να χρησιμοποιηθούν για την ανίχνευση μεγάλων κλάσεων. Στον Πίνακα 12 παρουσιάζονται τα αποτελέσματα αξιολόγησης για κάθε μία από τις τρεις επιλεγμένες εφαρμογές Android.

Πίνακας 12: Αποτελέσματα αξιολόγησης για την ανίχνευση feature envy.

	Feature Envy			**Deep Smell Detection**			**WekaNose**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Notepad	10,0%	50,0%	16,7%	12,5%	50,0%	20,0%	25,0%	50,0%	33,3%
Personal Stuff	20,0%	50,0%	28,6%	12,5%	14,3%	13,3%	16,7%	50,0%	25,0%
Wikipedia	59,3%	57,1%	58,2%	7,2%	31,7%	44,1%	6,3%	4,3%	5,1%
Μέσος Όρος	29,8%	52,4%	34,5%	32,4%	32,0%	25,8%	16,0%	34,8%	21,2%

Όπως φαίνεται το εργαλείο Feature Envy παρουσιάζει καλύτερα αποτελέσματα, όσον αφορά την ανίχνευση feature envy. Επιπλέον, παρατηρείται ότι και οι ανιχνευτές Feature Envy και Deep Smell Detection μπορούν να ανταποκριθούν καλύτερα σε προγράμματα μεγαλύτερου μεγέθους, σε αντίθεση με το WekaNose που παρουσιάζεται να μην μπορεί να ανιχνεύσει τέτοιου είδους code smells.

4.4.5 Data Class

Από τους επιλεγμένους ανιχνευτές τα Deep Smell Detection και WekaNose μπορούν να χρησιμοποιηθούν για την ανίχνευση κλάσεων δεδομένων. Στον Πίνακα 14

παρουσιάζονται τα αποτελέσματα αξιολόγησης για κάθε μία από τις τρεις επιλεγμένες εφαρμογές Android.

Πίνακας 13: Αποτελέσματα αξιολόγησης για την ανίχνευση κλάσεων δεδομένων.

	Deep Smell Detection			**WekaNose**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Notepad	100,0%	50,0%	66,7%	50,0%	50,0%	50,0%
Personal Stuff	7,1%	33,3%	11,8%	7,1%	33,3%	11,8%
Wikipedia	16,7%	50,0%	25,0%	20,0%	66,7%	30,7%
Μέσος Όρος	41,3%	44,4%	34,5%	25,7%	50,0%	30,8%

Όπως φαίνεται το εργαλείο Deep Smell Detection παρουσιάζει καλύτερα αποτελέσματα, όσον αφορά την ανίχνευση κλάσεων δεδομένων. Επιπλέον, παρατηρείται ότι και οι 2 ανιχνευτές μπορούν να ανταποκριθούν καλύτερα σε προγράμματα μικρότερου μεγέθους.

5 Συζήτηση & Μελλοντικές Επεκτάσεις

Στην παρούσα διπλωματική εργασία διερευνήθηκαν εργαλεία που βασίζονται σε τεχνικές μηχανικής μάθησης για τον εντοπισμό μονάδων λογισμικού που εμφανίζουν συμπτώματα κακής σχεδίασης, υψηλό τεχνικό χρέος ή σφάλματα, δηλ. code smells και anti-patterns.

Με στόχο να απαντηθούν τα τέσσερα (4) ερευνητικά ερωτήματα που τέθηκαν στα πρώιμα στάδια της παρούσας έρευνας, αρχικά πραγματοποιήθηκε βιβλιογραφική έρευνα για την εύρεση των μεθόδων και την καταγραφή των αλγορίθμων μηχανικής μάθησης που έχουν εφαρμοστεί για την υλοποίηση ανιχνευτών code smells και anti-patterns σε μονάδες λογισμικού. Σε συνέχεια της βιβλιογραφικής έρευνας, αναζητήθηκαν και εντοπίστηκαν εργαλεία ανίχνευσης code smells και anti-patterns που βασίζονται στην μηχανική μάθηση. Από αυτά επιλέχθηκαν 4, τα οποία αξιολογήθηκαν τόσο συλλογικά, όσο και επιμέρους. Η αξιολόγησή τους πραγματοποιήθηκε με τη χρήση τριών Android εφαρμογών που διατίθενται ως ελεύθερο λογισμικό.

Η έρευνά μας έδειξε ότι τα τελευταία χρόνια όλο και περισσότερες μελέτες παρουσιάζουν διαφορετικές προσεγγίσεις που χρησιμοποιούν αλγορίθμους μηχανικής μάθησης με στόχο την ανάπτυξη εργαλείων ανίχνευσης code smells και anti-patterns. Η πλειοψηφία αυτών αξιοποιεί αλγορίθμους βαθιάς μάθησης, όπως για παράδειγμα τα RNN, CNN [18], [33], LSTM, NLP [32], ωστόσο δεν είναι λίγες οι μελέτες που αξιοποιούν αλγορίθμους μηχανικής μάθησης με πιο αποδοτικούς του αλγορίθμους τους Naive Bayes, VFI, J48 [13], [35] και Cat Boost [15], [10].

Επιπλέον, διαπιστώθηκε ότι στην πλειοψηφία των προσεγγίσεων η ύπαρξη προβλημάτων στο σύνολο των δεδομένων προς ανάλυση, όπως για παράδειγμα η ανισορροπία ή η κακή ακρίβεια αυτών επηρεάζουν αρνητικά την απόδοση των αλγορίθμων, για αυτό και πληθώρα ανιχνευτών συνοδεύονται από προσεγγίσεις για τον μετριασμό των προβλημάτων ανισορροπίας δεδομένων.

Αξίζει να σημειωθεί ότι οι διαφορετικές προσεγγίσεις, ενώ επιδιώκουν στην επίλυση του ίδιου προβλήματος, δεν ακολουθούν την ίδια μεθοδολογία. Χαρακτηριστικό παράδειγμα είναι η μέθοδος επιλογής των δεδομένων για την εκπαίδευση των ML μοντέλων καθώς ορισμένες προσεγγίσεις εστιάζουν στα δομικά χαρακτηρίστηκα [18] του κώδικα, άλλες στα σημασιολογικά χαρακτηριστικά [32] αυτού, άλλες σε μετρικές κώδικα [33] άλλες σε συνδυασμό των προαναφερθέντων [38].

Η αξιολόγηση των μοντέλων γίνεται κυρίως με τις μετρικές Precision, Recall και F1-Measure, ωστόσο ορισμένοι ερευνητές επιλέγουν να χρησιμοποιήσουν επικουρικά και άλλους δείκτες αξιολόγησης, όπως για παράδειγμα τον Kappa του Cohen, το ROC και το MCC [2].

Όσον αφορά τα είδη code smells και anti-pattern που μπορούν να εντοπιστούν από τους ανιχνευτές που βασίζονται στην μηχανική μάθηση, προέκυψε ότι μπορούν να ανιχνευθούν συνολικά 4 anti-patterns, τα God Class [15], [10], [26], Blob, Functional Decomposition και Spaghetti Code [14] και μόλις 8 από τα 22 διαφορετικά είδη code smells, τα Feature Envy [13], [26], [28], [32], [35], Data Class, Long Method[32], [35], Long Parameter Lists [35], Large Class [27], [32], [35], Message Chains, Middle Man και Switch Statements [13]. Τα επιλεγμένα εργαλεία που διερευνήθηκαν είχαν την δυνατότητα να εντοπίζουν 5 από αυτά, τα Long Method, Large Class, Long Parameter List, Feature Envy και Data Class.

Όσον αφορά τη πειραματική ανάλυση, αναμενόταν ότι οι ανιχνευτές, που βρίσκουν τα ίδια code smells, θα δίνουν τα ίδια αποτελέσματα. Ωστόσο, αυτό δεν επιβεβαιώθηκε από την έρευνα. Αντίθετα, φάνηκε ότι τα αποτελέσματα διαφέρουν σημαντικά μεταξύ τους. Επιπλέον, τα αποτελέσματα έδειξαν ότι κατά περίπτωση το μέγεθος του έργου επηρεάζει την αποτελεσματικότητα των εργαλείων. Τέλος, επιβεβαιώνεται ότι η ανισορροπία στα δεδομένα επηρεάζει τα αποτελέσματα.

Σε συνέχεια της παρούσας διπλωματικής εργασίας, η έρευνα θα μπορούσε να συνεχιστεί εστιάζοντας στην ανάπτυξη ενός εργαλείου ανίχνευσης code smell το οποίο θα βασίζεται στην μηχανική μάθηση. Πιο αναλυτικά, θα μπορούσε να αναπτυχθεί ένα εργαλείο, το οποίο θα χρησιμοποιεί την μέθοδο NLP και θα αναλύει τα σχόλια στον πηγαίο κώδικα ενός λογισμικού με στόχο τον εντοπισμό περιττών ή πολύ περιγραφικών σχολίων με βάση τον κώδικα στον οποίο αναφέρονται. Ο λόγος που επιλέχθηκε αυτή η μορφή code smell είναι αφενός επειδή η πληθώρα σχολίων στον πηγαίο κώδικα μίας εφαρμογής αποτελεί υπό συνθήκες code smell, καθώς τα σχόλια λειτουργούν ως «κάλυμμα» σε ζητήματα που αφορούν κυρίως τον σχεδιασμό της και αφετέρου επειδή η βιβλιογραφική έρευνα που διενεργήθηκε έδειξε ότι δεν υπάρχει κάποια έρευνα ή εργαλείο που να ασχολείται με τον εντοπισμό αυτού του είδους code smell.

Βιβλιογραφία

1. Girish, S., Ganesh, S. and Tushar, S. (2014). Refactoring for Software Design Smells: Managing Technical Debt. Morgan Kaufmann Publishers Inc.340 Pine Street, Sixth Floor San Francisco CA United States, p.258.

2. Pecorelli, F., Palomba, F., Nucci, D. D., Andrea, D. L. (2019). Comparing heuristic and machine learning approaches for metric-based code smell detection. Proceedings of the 27th International Conference on Program Comprehension (ICPC '19).

3. Tahvildar L., Kontogiannis K. (2004). Improving design quality using meta-pattern transformations: a metric-based approach. J. Softw.: Evol. Process., 16 (4–5), pp. 331-361.

4. Munro M. J. (2005). Product metrics for automatic identification of "bad smell" design problems in java source-code. Intl. Conf. Software Metrics, p. 15.

5. Marinescu C., Marinescu R., Mihancea P.F., Wettel R. IPlasma (2005). An integrated platform for quality assessment of object-oriented design. Intl. Conf. Software Maintenance - Industrial and Tool Volume, pp. 77-80.

6. Choinzon M., Ueda Y. (2006). Detecting defects in object-oriented designs using design metrics. J. Conf. on Knowledge-Based Software Engineering, pp. 61-72.

7. Moha N., Guéhéneuc Y.-G. (2007). DECOR: a tool for the detection of design defects. Intl. Conf. on Automated Software Engineering, pp. 527-528.

8. Fourati R., Bouassida N., Abdallah H. (2011). A metric-based approach for anti-pattern detection in UML designs. Comput. Inf. Sci., pp. 17-33.

9. Shatnawi R. (2015). Deriving metrics thresholds using log transformation. J. Softw.: Evol. Process., 27 (2), pp. 95-113.

10. Alkharabsheh, K., Alawadi, S., Kebande, V., Crespo, Y., Fernández-Delgado, M., & Taboada, J. (2022). A comparison of machine learning algorithms on design smell detection using balanced and imbalanced dataset: A study of God class. Information And Software Technology, 143, 106736.

11. Kreimer J. (2005). Adaptive detection of design flaws Electron. Notes Theor. Comput. Sci., 141 (4), pp. 117-136.

12. Hassaine S., Khomh F., Guéhéneuc Y.- G., Hamel S. (2010) Ids: an immune-inspired approach for the detection of software design smells. Intl. Conf. Quality of Information and Communications Technology, pp. 343-348.

13. Maneerat N., Muenchaisri P. (2011). Bad-smell prediction from software design model using machine learning techniques. Intl. J. Conf. on Computer Science and Software Engineering, pp. 331-336

14. Khomh F. Vaucher S., Guéhéneuc Y.-G., Sahraoui H. (2011). BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. J. Syst. Softw., 84 (4), pp. 559-572.

15. Alkharabsheh K., Crespo Y., Fernández- Delgado M., Cotos J.M., Taboada J.A. (2019). Assessing the influence of size category of the project in God class detection, an experimental approach based on machine learning (MLA). International Conference on Software Engineering & Knowledge Engineering, pp. 361-366.

16. Pecorelli F., Di Nucci D., De Roover C., De Lucia A. (2020) A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. J. Syst. Softw., Article 110693.

17. Romeo L. Quoi, JR. (2020). Deep Learning-Based Code Smell Detection Using CNN and RNN. Master Thesis, School of Computer Science & Technology, Beijin Institute of Technology

18. Sharma, T., Efstathiou, V., Louridas, P., & Spinellis, D. (2021). Code smell detection by deep direct-learning and transfer-learning. Journal Of Systems and Software, 176, 110936.

19. Γιακουμάκης, Ε. και Διαμαντίδης, Ν., (2009). Τεχνολογία Λογισμικού. 1η εκδ.. Αθήνα: Εκδόσεις Σταμούλη Α.Ε., σ.25-26.

20. International Organization for Standardization (ISO) (2011). Systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - System and software quality models, ISO/IEC 25010:2011.

21. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999) Refactoring. Improving the Design of Existing Code. Boston: Addison-Wesley Professional.

22. Mantyla, M., Vanhanen, J., Lassenius, C. (2003). A taxonomy and an initial empirical study of bad smells in code. Proceedings of the International Conference on Software Maintenance, 2003. ICSM 2003. Amsterdam, The Netherlands, pp. 381-384.

23. Brown WJ, Malveau RC, Brown WH, McCormick HW III, Mowbray TJ (1998) Anti patterns: refactoring software, architectures, and projects in crisis, 1st edn. Wiley, New York

24. Settas D. (2011). Software Project Antipattern Knowledge Management. PhD thesis, Department of Informatics, Aristotle University of Thessaloniki, p. 18.

25. Shi, B., & Iyengar, S.S. (2019). Mathematical Theories of Machine Learning - Theory and Applications.

26. Barbez, A., Khomh, F., & Guéhéneuc, Y. (2020). A machine-learning based ensemble method for anti-patterns detection. Journal Of Systems And Software, 161, 110486.

27. Barbez, A., Khomh, F., & Guéhéneuc, Y. G. (2019). Deep learning anti-patterns from code metrics history. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 114-124). IEEE.

28. Liu, H., Xu, Z., & Zou, Y. (2018). Deep learning-based feature envy detection. In Proceedings of the 33rd ACM/IEEE international conference on automated software engineering (pp. 385-396).

29. Mostaeen, G., Svajlenko, J., Roy, B., Roy, C. K., & Schneider, K. A. (2018). On the use of machine learning techniques towards the design of cloud based automatic code clone validation tools. In 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM) (pp. 155-164). IEEE.

30. Ochodek, M., Staron, M., Bargowski, D., Meding, W., & Hebig, R. (2017, February). Using machine learning to design a flexible LOC counter. In Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on (pp. 14-20). IEEE.

31. Ochodek, M., Hebig, R., Meding, W., Frost, G., & Staron, M. (2020). Recognizing lines of code violating company-specific coding guidelines using machine learning. Empirical Software Engineering, 25(1), 220-265.

32. Hui Liu. (2021). Large-npy-Files4ModelTraining (v1.0). Zenodo.

33. Zhang, Y., Ge, C., Hong, S., Tian, R., Dong, C., & Liu, J. (2022). DeleSmell: Code smell detection based on deep learning and latent semantic analysis. Knowledge-Based Systems, 255, 109737.

34. Zhang, Y., & Dong, C. (2021). MARS: Detecting brain class/method code smell based on metric–attention mechanism and residual network. Journal of Software: Evolution and Process, e2403.

35. Azadi, U., Fontana, F. A., & Zanoni, M. (2018). Poster: machine learning based code smell detection through WekaNose. In 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) (pp. 288-289). IEEE.

36. F-Droid - Free and Open-Source Android App Repository. (2022). Retrieved September 2022, from https://f-droid.org/

37. VS Code Counter - Visual Studio Marketplace. (2022). Retrieved September 2022, from https://marketplace.visualstudio.com/items?itemName=uctakeoff.vscode-counter

38. Marinescu, C., Marinescu, R., Mihancea, P.F., Ratiu, D., & Wettel, R. (2005). iPlasma: An Integrated Platform for Quality Assessment of Object-Oriented Design. ICSM.

39. word2vec | TensorFlow Core. (2022). Retrieved September 2022, from https://www.tensorflow.org/tutorials/text/word2vec

40. scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation. (2022). Retrieved October 2022, from https://scikit-learn.org/stable/

41. Kovačević, Aleksandar; Luburić, Nikola; Slivka, Jelena; Prokić, Simona; Grujić, Katarina-Glorija; Vidaković, Dragan; et al. (2022): Automatic detection of code smells using metrics and CodeT5 embeddings: a case study in C#. TechRxiv. Preprint.

Παράρτημα Α- Μήτρες Σύγχυσης και Ορθότητα πειραμάτων

Στους Πίνακες που ακολουθούν παρατίθενται τα confusion matrices και το accuracy για όλα τα πειράματα που πραγματοποιήθηκαν στο πλαίσιο της παρούσας διπλωματικής εργασίας.

Long Method						
		True Positive	**True Negative**	**False Positive**	**False Negative**	**Accuracy**
Deep Smell Detection	**Notebook**	9	18	3	4	0,79411765
	Personal Stuff	4	23	6	0	0,81818182
	Wikipedia	8	38	21	3	0,65714286
WekaNose	**Notebook**	3	20	5	1	0,79310345
	Personal Stuff	4	24	1	2	0,90322581
	Wikipedia	4	38	24	2	0,61764706

Large Class						
		True Positive	**True Negative**	**False Positive**	**False Negative**	**Accuracy**
CAME	**Notebook**	1	6	2	1	0,7
	Personal Stuff	2	20	2	6	0,73333333

	Wikipedia	5	18	2	8	0,6969697
Deep Smell Detection	**Notebook**	1	3	5	1	0,4
	Personal Stuff	3	18	1	2	0,875
	Wikipedia	6	18	3	5	0,75
WekaNose	**Notebook**	1	3	3	1	0,5
	Personal Stuff	1	25	4	3	0,78787879
	Wikipedia	4	48	13	5	0,74285714

Long Parameter List						
		True Positive	**True Negative**	**False Positive**	**False Negative**	**Accuracy**
WekaNose	**Notebook**	1	29	4	1	0,85714286
	Personal Stuff	0				-
	Wikipedia	25	5	5	35	0,42857143

Feature Envy						
		True Positive	**True Negative**	**False Positive**	**False Negative**	**Accuracy**

FeatureEnvy	**Notebook**	1	24	9	1	0,71428571
	Personal Stuff	1	23	4	1	0,82758621
	Wikipedia	16	12	11	12	0,54901961
Deep Smell Detection	**Notebook**	1	24	9	1	0,71428571
	Personal Stuff	1	23	4	1	0,82758621
	Wikipedia	16	12	11	12	0,54901961
WekaNose	**Notebook**	1	23	3	1	0,85714286
	Personal Stuff	1	31	5	1	0,84210526
	Wikipedia	1	45	15	22	0,55421687

Data Class						
		True Positive	**True Negative**	**False Positive**	**False Negative**	**Accuracy**
Deep Smell Detection	**Notebook**	1	5	0	1	0,85714286
	Personal Stuff	1	17	13	2	0,54545455
	Wikipedia	5	23	25	5	0,48275862
WekaNose	**Notebook**	1	4	1	1	0,71428571

	Personal Stuff	1	12	13	2	0,46428571
	Wikipedia	2	15	8	1	0,65384615