



ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εργαλεία λογισμικού για αξιολόγηση και βελτίωση ποιότητας κώδικα

Φοίβος Γ. Παπαπαναγιωτάκης Μπουσού
Α.Μ.: 3823

Επιβλέπων: Αντώνιος Σαββίδης, Καθηγητής Επιστήμης Υπολογιστών

ΑΘΗΝΑ

ΑΥΓΟΥΣΤΟΣ 2021

ΠΕΡΙΛΗΨΗ

Σκοπός αυτής της εργασίας είναι η κατασκευή ενός επεκτάσιμου συστήματος για την ανίχνευση δυσοσμίων κώδικα (code smells) σε προγράμματα υλοποιημένα στην γλώσσα C++. Δυσοσμία κώδικα σε ένα πρόγραμμα είναι ενδείξεις είτε κακών σχεδιαστικών επιλογών είτε κακής υλοποίησης [1]. Το σύστημα οφείλει να εντοπίζει τέτοιες ενδείξεις και να τις παρουσιάζει στον χρήστη με όσο το δυνατό πιο κατανοητή εικονοποίηση και στατιστικά. Επιπρόσθετα, το σύστημα πρέπει να είναι εύκολα επεκτάσιμο στο επίπεδο ανίχνευσης των δυσοσμίων κώδικα καθώς και στον οραματισμό τους.

Για την διεκπεραίωση της εργασίας ασχοληθήκαμε με θεωρητικά προβλήματα όπως το τι είναι μια δυσοσμία κώδικα όσο και με τεχνικά προβλήματα όπως την ανίχνευση τους, την διεπαφή χρήστη, την παραμετροποίηση και την επεκτάσιμη φύση του συστήματος.

Τέλος, καταλήγουμε σε πολύ ικανοποιητικά αποτελέσματα, παρατηρώντας ότι το σύστημα παρουσιάζει αποτελεσματικά στον χρήστη περιοχές κώδικα που πράγματι χρειάζονται αναθεώρηση, δουλεύει ακόμη και με είσοδο μεγάλα προγράμματα και είναι επεκτάσιμο σύμφωνα με τις λειτουργικές προδιαγραφές.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον κύριο Αντώνιο Σαββίδη για την βοήθεια και καθοδήγηση που προσέφερε στην σχεδίαση, υλοποίηση και συγγραφή αυτής της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω την Κρυσταλλία Σαββάκη, καθώς μεγάλο τμήμα του πρώτου συστήματος (Code Smell Detector) που υλοποιήθηκε στα πλαίσια αυτής της εργασίας, προέρχεται από την δική της μεταπτυχιακή εργασία με τίτλο “Εξαγωγή της Αρχιτεκτονικής Λογισμικού από τον Πηγαίο Κώδικα”.

Περιεχόμενα

| | |
|--|----|
| 1. Εισαγωγή | 6 |
| 1.1 Αντικείμενο της εργασίας | 6 |
| 1.2 Διάρθρωση της εργασίας | 6 |
| 2. Θεωρητικό υπόβαθρο | 7 |
| 2.1 Δυσσομία κώδικα | 7 |
| 3. Περιγραφή Συστήματος | 8 |
| 3.1 Προδιαγραφές | 8 |
| 3.2 Αρχιτεκτονική | 8 |
| 3.3 Υλοποίηση | 9 |
| 3.3.1 Εργαλεία & Βιβλιοθήκες | 9 |
| 3.3.2 Δημιουργία πίνακα συμβόλων με το Clang | 10 |
| 3.3.3 Διεπαφή προγραμματισμού εφαρμογών των ανιχνευτών δυσσομίας | 12 |
| 3.3.4 Ανιχνευτές δυσσομίας | 16 |
| 3.3.5 Διεπαφή χρήστη | 21 |
| 4. Οδηγίες Χρήσης | 24 |
| 4.1 Εγκατάσταση | 24 |
| 4.1.1 Εγκατάσταση του Clang | 24 |
| 4.1.2 Εγκατάσταση του Symbol Table Export | 25 |
| 4.1.3 Εγκατάσταση του Code Smell Detector | 26 |
| 4.2 Χρήση | 27 |
| 4.2.1 Χρήση του Symbol Table Export | 27 |
| 4.2.2 Χρήση του Code Smell Detector | 28 |
| 4.3 Επέκταση | 29 |
| 5. Συμπεράσματα | 30 |
| 6. Βιβλιογραφία | 31 |

1. Εισαγωγή

1.1 Αντικείμενο της εργασίας

Μετά την τεχνολογική ανάπτυξη του υλικού των υπολογιστών κατά το δεύτερο μισό του 20^{ου} αιώνα, ακολούθησε η δημιουργία όλο και μεγαλύτερων συστημάτων λογισμικού. Η διαχείριση, συντήρηση και επέκταση τέτοιων γιγαντιαίων συστημάτων είναι πλέον, ένα πολύ γνωστό πρόβλημα το οποίο ταλαιπωρεί αδιάκοπα μεγάλες εταιρίες και οργανισμούς.

Μια από τις πηγές του προβλήματος είναι η αφηρημένη φύση του λογισμικού μεγάλης κλίμακας. Για να αποφευχθεί η χαοτική δομή ενός τέτοιου συστήματος, πρέπει κατά τη δημιουργία του, να τηρηθούν αυστηρά κανόνες σχεδιασμού και υλοποίησης. Σε αυτή την εργασία εξετάζουμε κακές πρακτικές προγραμματισμού καθώς και μεθόδους για την αυτοματοποιημένη ανίχνευσή τους απευθείας από τον πηγαίο κώδικα.

1.2 Διάρθρωση της εργασίας

Η εργασία ξεκινάει στο δεύτερο κεφάλαιο όπου καλύπτεται το θεωρητικό υπόβαθρο και ορίζεται το πρόβλημα το οποίο καλούμαστε να επιλύσουμε. Στο τρίτο κεφάλαιο βρίσκεται η περιγραφή του συστήματος η οποία περιλαμβάνει τις λειτουργικές προδιαγραφές, την αρχιτεκτονική και την περιγραφή της υλοποίησης των διάφορων τμημάτων. Το τέταρτο κεφάλαιο περιέχει τις οδηγίες εγκατάστασης, χρήσης και επέκτασης του συστήματος. Τέλος στο πέμπτο κεφάλαιο βρίσκονται τα σχόλια περί των αποτελεσμάτων, μικροπροβλήματα και πιθανές βελτιώσεις που θα μπορούσαν να ερευνηθούν σε επόμενες εκδόσεις.

2. Θεωρητικό υπόβαθρο

2.1 Δυσσοσμία κώδικα

Μία δυσσοσμία κώδικα είναι ένα σύμπτωμα είτε κακών σχεδιαστικών επιλογών είτε κακής υλοποίησης [1]. Ενώ μία δυσσοσμία κώδικα ενδέχεται να μην αποτελεί δυσχέρεια, είναι συνήθως ένδειξη κάποιου βαθύτερου προβλήματος [2].

Καθώς δεν υπάρχει αντικειμενικός ορισμός κακής σχεδίασης λογισμικού, δεν είναι εφικτό να υπάρξει απόλυτος ορισμός που να καθορίζει τι είναι δυσσοσμία κώδικα και τι όχι.

Ωστόσο, ως δυσσοσμίες κώδικα, μεταξύ άλλων, έχουν επικρατήσει τα παρακάτω:

- Συναρτήσεις με υπερβολικό πλήθος παραμέτρων
- Συναρτήσεις με υπερβολικά literals¹
- Συναρτήσεις με υπερβολικές τοπικές δηλώσεις
- Συναρτήσεις με πολλές επαναλήψεις ή branching statements²
- Συναρτήσεις με πολλά εμφωλευμένα μπλοκ κώδικα
- Υπερβολικά μεγάλες συναρτήσεις
- Αναγνωριστικά πολύ μεγάλου μήκους
- Αναγνωριστικά που δεν ακολουθούν κάποια σύμβαση γραφής
- Αναγνωριστικά που δεν εκφράζουν την χρησιμότητα του αντικειμένου το οποίο αναπαριστούν
- Γραμμές κώδικα μεγάλου μήκους
- Υπερβολικά μεγάλα αρχεία κώδικα
- Κυκλική εξάρτηση
- Δομές ή αρχεία με πολλές εξαρτήσεις
- Υπερβολικά μεγάλες δομές
- Δομές που χρησιμοποιούν υπερβολικά συχνά λειτουργίες μιας άλλης δομής
- Δομές που είναι υπερβολικά μικρές
- Μεταβλητές που εννοιολογικά πρέπει να ομαδοποιηθούν σε μια δομή αλλά παραμένουν ανεξάρτητες
- Αντικείμενα με πολλαπλές ευθύνες
- Διπλότυπος κώδικας
- Υπερβολική χρήση σχολίων

Από τις ειδικές λειτουργίες και χαρακτηριστικά μιας γλώσσας προγραμματισμού μπορούν να προκύψουν καινούργιες δυσσοσμίες. Μερικές από αυτές για την γλώσσα C++ είναι:

- Συναρτήσεις και τελεστές που υπερφορτώνονται υπερβολικά συχνά
- Downcasting
- Υπερβολική χρήση πολλαπλής κληρονομικότητας

¹ Literal: σταθερές τιμές στο πρόγραμμα (πχ. true, 2, "str").

² Branching statement: μία εντολή η οποία μπορεί να αλλάξει την ροή του προγράμματος [5].

3. Περιγραφή Συστήματος

3.1 Προδιαγραφές

Όπως προαναφέρθηκε, η δυσοσμία κώδικα είναι πολύ γενική ορολογία. Είναι εύκολο κάποιος να θεωρήσει ως δυσοσμία κώδικα κάτι αποκλειστικά υποκειμενικό, επομένως, ένα πρόγραμμα με σκοπό την ανίχνευση τέτοιων, οφείλει να είναι ευκολά επεκτάσιμο. Είναι σημαντικό να μπορεί ένας προγραμματιστής με ελάχιστη γνώση του συστήματος να επεκτείνει και να παραμετροποιεί το τμήμα ανίχνευσης δυσοσμιών. Για αυτό τον λόγο, η κάθε κατηγορία δυσοσμίας πρέπει να έχει τον δικό της ανιχνευτή που έχει τις δικές του παραμέτρους και να επικοινωνεί με το σύστημα μέσω μιας αυστηρά καθορισμένης διεπαφής προγραμματισμού εφαρμογών.

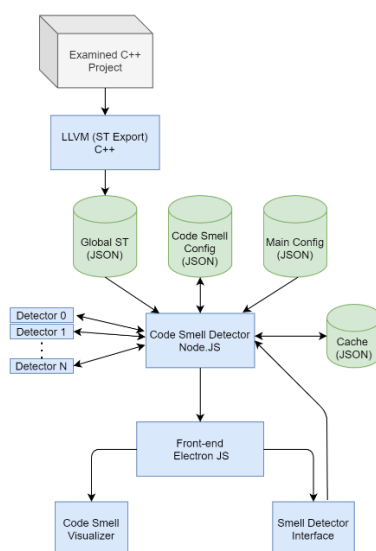
Επιπρόσθετα, στόχος μας, είναι το σύστημα να μπορεί να χειριστεί μεγάλες εισόδους καθώς η ανίχνευση δυσοσμιών έχει ιδιαίτερο κέρδος σε μεγάλα συστήματα.

Τέλος, το τμήμα της διεπαφής χρήστη που υλοποιεί την αναπαράσταση των στατιστικών πρέπει να είναι και αυτό επεκτάσιμο με εύκολη προσθήκη καινούργιων γραφημάτων και στατιστικών.

3.2 Αρχιτεκτονική

Το σύστημα που υλοποιήθηκε για την εκπλήρωση των στόχων αυτής της εργασίας χωρίζεται σε δύο υποσυστήματα, το Symbol Table Export και το Code Smell Detector. Το πρώτο είναι ένα C++ πρόγραμμα το οποίο χρησιμοποιεί το Clang για να εξάγει πληροφορίες από το εξεταζόμενο σύστημα και στη συνέχεια τις αποθηκεύει ως ένα αρχείο τύπου JSON το οποίο αργότερα αποκαλείται “Πίνακας Συμβόλων”. Το δεύτερο σύστημα είναι ένα Node.js πρόγραμμα το οποίο αναλαμβάνει την ανίχνευση και απεικόνιση των δυσοσμιών κώδικα με βάση τον πίνακα συμβόλων που έχει προηγουμένως κατασκευαστεί από το Symbol Table Export.

Παρακάτω βρίσκεται μια εικόνα η οποία προβάλλει την αλληλεπίδραση των διάφορων τμημάτων των δύο υποσυστημάτων. Ο διαχωρισμός των δύο γίνεται στον πίνακα συμβόλων *Global ST (JSON)*.



Εικόνα 1: Η γενική αρχιτεκτονική του συστήματος.

3.3 Υλοποίηση

3.3.1 Εργαλεία & Βιβλιοθήκες

3.3.1.1 Εργαλεία & Βιβλιοθήκες του Symbol Table Export

Clang

Το κύριο εργαλείο του Symbol Table Export είναι το Clang το οποίο είναι ένα front-end του μεταγλωττιστή LLVM για την οικογένεια γλωσσών C [3]. Παρέχει μια συλλογή εργαλείων τα οποία, στα πλαίσια αυτής της εργασίας, τα χρησιμοποιούμε μέσω μιας διεπαφής προγραμματισμού εφαρμογών (API) για να συλλέξουμε κάθε είδους πληροφορίες από τον πηγαίο κώδικα του project που εξετάζουμε.

JsonCpp

Το JsonCpp είναι μια open source βιβλιοθήκη για την γλώσσα C++ η οποία υποστηρίζει τον χειρισμό JSON³ τιμών. Η βιβλιοθήκη αυτή χρησιμοποιείται από το Symbol Table Export για την αποθήκευση του πίνακα συμβόλων σε μορφή κειμένου.

CMake

Το CMake είναι ένα προγραμματιστικό εργαλείο που αυτοματοποιεί την διαδικασία μεταγλώττισης πηγαίου κώδικα η οποία ενδέχεται να είναι πολύπλοκη για μεγάλα προγράμματα. Το CMake μπορεί να δημιουργήσει ένα αρχείο οδηγιών για την μεταγλώττιση, που αργότερα θα χρησιμοποιηθεί από το Symbol Table Export.

Visual Studio - Sourcetrail

Το Sourcetrail είναι ένα open-source extension του Visual Studio το οποίο επίσης, όπως το CMake, μπορεί να δημιουργήσει το αρχείο οδηγιών μεταγλώττισης που θα χρησιμοποιηθεί από το Symbol Table Export.

3.3.1.2 Εργαλεία & Βιβλιοθήκες του Code Smell Detector

Node.js

Το Node.js είναι ένα open-source περιβάλλον εκτέλεσης για την γλώσσα JavaScript το οποίο δουλεύει ανεξάρτητα από περιηγητή ιστού. Πάνω σε αυτό είναι υλοποιημένο ολόκληρο το back-end του Code Smell Detector καθώς και κομμάτια του front-end.

Electron

Το Electron είναι ένα open-source framework το οποίο χρησιμοποιείται για την δημιουργία γραφικής διεπαφής χρήστη. Η χρήση του γίνεται αντίστοιχα με σελίδα περιηγητή, με HTML και CSS. Πάνω σε αυτό είναι υλοποιημένη η διεπαφή χρήστη του Code Smell Detector.

jQuery

Το jQuery είναι μια μεγάλη βιβλιοθήκη για την γλώσσα JavaScript της οποίας ο σκοπός είναι η απλοποίηση της χρήσης HTML αντικειμένων κατά την υλοποίηση του front-end σε

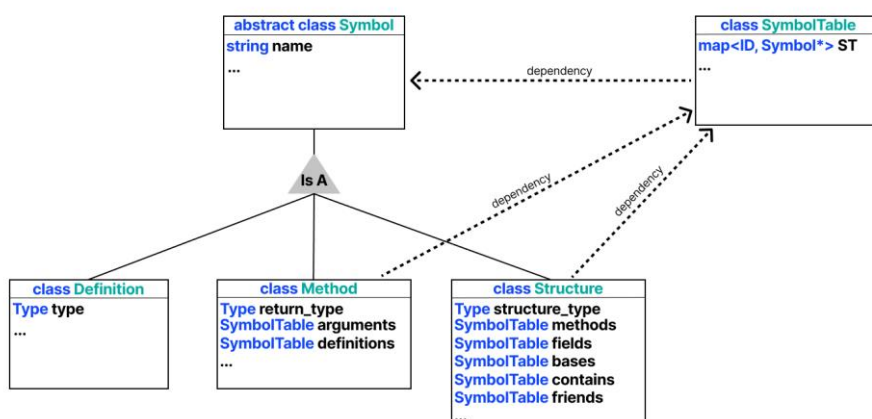
³ JSON, ή αλλιώς JavaScript Object Notation, είναι ένα απλό μορφότυπο για την αναπαράσταση πληροφοριών σε μορφή κειμένου [7].

εφαρμογές περιηγητή. Σε αυτή την εργασία χρησιμοποιείται για την υλοποίηση διαφόρων τμημάτων του front-end.

3.3.2 Δημιουργία πίνακα συμβόλων με το Clang

Το Symbol Table Export δέχεται ως είσοδο ένα C++ πρόγραμμα, το μεταγλωττίζει χρησιμοποιώντας τον μεταγλωττιστή LLVM και με τη χρήση του Clang αντλεί δεδομένα από τον πηγαίο κώδικα του προγράμματος τα οποία, στη συνέχεια, θα αποθηκευτούν σε έναν πίνακα συμβόλων ο οποίος εκτυπώνεται σε ένα αρχείο τύπου .json.

Στο Symbol Table Export υπάρχει μια κλάση *SymbolTable* η οποία είναι η υλοποίηση ενός πίνακα συμβόλων. Κάθε αντικείμενο σε αυτόν το πίνακα είναι τύπου *Symbol* και το κλειδί που τους αντιστοιχεί είναι το αναγνωριστικό τους. Η κλάση *Symbol* είναι μια βασική κλάση χωρίς αντικείμενα (abstract class) που κληρονομείται από τις κλάσεις *Definition*, *Method* και *Structure* οι οποίες αντιπροσωπεύουν δηλώσεις, μεθόδους και δομές αντίστοιχα. Οι κλάσεις *Method* και *Structure* έχουν πεδία τα οποία είναι και αυτά τύπου *SymbolTable* καθώς μέθοδοι και δομές μπορούν εσωτερικά να έχουν δικές τους δηλώσεις. Παρακάτω βρίσκεται ένα απλοποιημένο διάγραμμα οντοτήτων - σχέσεων που δείχνει πώς οι κλάσεις αυτές αλληλοεπιδρούν μεταξύ τους.



Εικόνα 2: Διάγραμμα οντοτήτων - σχέσεων που περιγράφει τη χρήση της κλάσης *SymbolTable*.

Ο τρόπος ο οποίος το πρόγραμμα δημιουργεί τον πίνακα συμβόλων είναι ο εξής: αρχικά δημιουργείται ένα αντικείμενο τύπου *SymbolTable* στο οποίο θα αποθηκευτούν όλες οι δομές του προγράμματος που βρίσκονται σε global scope. Στη συνέχεια, το πρόγραμμα θα γεμίσει κάθε πεδίο της κάθε κλάσης αναδρομικά έτσι ώστε στο τέλος να βρίσκονται μέσα στον πίνακα συμβόλων όλες οι κλάσεις και τα περιεχόμενά τους. Το αντικείμενο τύπου *SymbolTable* που αναφέρεται σε καθολική εμβέλεια θα αποθηκευτεί στο παραγόμενο αρχείο (JSON) σε πεδίο με όνομα *structures*.

Παρακάτω αναγράφονται αναλυτικά τα πεδία που θα αποθηκευτούν για κάθε δομή, μέθοδο και δήλωση. Το πρόγραμμα εσωτερικά χρειάζεται και άλλα πεδία, ωστόσο παρακάτω βρίσκονται μόνο αυτά που θα καταλήξουν στο αρχείο που δημιουργείται.

Structures

| Πεδίο | Τύπος τιμής | Περιγραφή τιμής |
|----------|-----------------------------|---|
| contains | SymbolTable | Αντικείμενο SymbolTable που περιέχει μόνο αντικείμενα τύπου Structure (τις εσωτερικές δομές της δομής). |
| fields | SymbolTable | Αντικείμενο SymbolTable που περιέχει μόνο αντικείμενα τύπου Definition. |
| friends | SymbolTable | Αντικείμενο SymbolTable που περιέχει αντικείμενα τύπου Method και Structure. |
| methods | SymbolTable | Αντικείμενο SymbolTable που περιέχει μόνο αντικείμενα τύπου Method. |
| src_info | SourceInfo | Αντικείμενο το οποίο περιέχει το αρχείο, τη γραμμή και τη στήλη όπου δηλώθηκε η δομή. |

Methods

| Πεδίο | Τύπος τιμής | Περιγραφή τιμής |
|-------------|-----------------------------|--|
| access | string | Συμβολοσειρά με τιμές “private”, “protected” ή “public”. |
| args | SymbolTable | Αντικείμενο SymbolTable που περιέχει μόνο αντικείμενα τύπου Definition. |
| branches | int | Το πλήθος των branching statements της μεθόδου. |
| definitions | SymbolTable | Αντικείμενο SymbolTable που περιέχει μόνο αντικείμενα τύπου Definition. |
| lines | int | Το πλήθος των γραμμών της μεθόδου. |
| literals | int | Το πλήθος των literals της μεθόδου. |
| loops | int | Το πλήθος των επαναλήψεων της μεθόδου. |
| max_scope | int | Το βάθος του βαθύτερου μπλοκ κώδικα της μεθόδου. |
| ret_type | string | Συμβολοσειρά με το τύπο επιστροφής της μεθόδου. |
| src_info | SourceInfo | Αντικείμενο το οποίο περιέχει το αρχείο, τη γραμμή και τη στήλη όπου δηλώθηκε η μέθοδος. |
| statements | int | Το πλήθος των statement της μεθόδου. |
| virtual | boolean | Τιμή <i>true</i> μόνο εάν η μέθοδος έχει δηλωθεί ως virtual. |

Definitions

| Πεδίο | Τύπος τιμής | Περιγραφή τιμής |
|--------|------------------------|---|
| access | string | Συμβολοσειρά με τιμές “private”, “protected” ή “public”. Εάν η δήλωση αυτή δεν είναι πεδίο (field) μίας δομής τότε δεν υπάρχει το πεδίο access. |
| type | string | Συμβολοσειρά με τον τύπο της δήλωσης. |

Το παραγόμενο αρχείο, πέρα από το πεδίο *structures* που περιέχει τον μεγάλο πίνακα συμβόλων έχει τρία επιπλέον πεδία, τα *headers*, *sources* και *dependencies*. Τα πεδία *headers* και *sources* θα γεμίσουν με μία λίστα από διαδρομές αρχείων. Πιο συγκεκριμένα, το *headers* θα αποτελείται από τη λίστα διαδρομών των αρχείων header (κατάληξης .h / .hpp) και το *sources* από τις διαδρομές των αρχείων πηγαίου κώδικα (κατάληξης .cpp). Η τιμή του πεδίου *dependencies* είναι μια λίστα αντικειμένων που αντιπροσωπεύουν μια σχέση εξάρτησης από μια δομή σε μια άλλη. Κάθε τέτοιο JSON αντικείμενο έχει ένα πεδίο *from* στο οποίο βρίσκεται το όνομα της εξαρτώμενης δομής, ένα πεδίο *to* στο οποίο βρίσκεται το όνομα της δομής από την οποία εξαρτάται και ένα πεδίο *types* του οποίου η τιμή είναι ένα αντικείμενο που περιέχει ζεύγη τύπου και πλήθους εξαρτήσεων. Κάθε αντικείμενο εξάρτησης έχει τουλάχιστον έναν από τους παρακάτω 11 τύπους εξαρτήσεων: *Inherit*, *Friend*, *NestedClass*, *ClassField*, *ClassTemplateParent*, *ClassTemplateArg*, *MethodReturn*, *MethodArg*, *MethodDefinition*, *MemberExpr*, *MethodTemplateArgs*.

```
{
  "from" : "UnitTest3",
  "to" : "Sprite",
  "types" :
  {
    "ClassField" : 1,
    "MemberExpr" : 166,
    "MethodArg" : 9,
    "MethodDefinition" : 10
  }
},
```

Εικόνα 3: Παράδειγμα αντικειμένου εξάρτησης

3.3.3 Διεπαφή προγραμματισμού εφαρμογών των ανιχνευτών δυσοσμίας

Παρακάτω βρίσκεται η περιγραφή της διεπαφής προγραμματισμού εφαρμογών (API) των ανιχνευτών δυσοσμίας.

Ένας ανιχνευτής δυσοσμίας πρέπει να δηλωθεί μαζί με τις παραμέτρους του σε ένα αρχείο παραμετροποίησης. Μεταξύ άλλων, η δήλωση του ανιχνευτή, πρέπει να έχει ένα πεδίο με τη διαδρομή για το αρχείο που τον υλοποιεί.

Στο αρχείο υλοποίησής του, θα πρέπει να βρίσκεται μια συνάρτηση η οποία δέχεται δύο παραμέτρους, τον πίνακα συμβόλων και ένα αντικείμενο το οποίο έχει τις παραμέτρους του ανιχνευτή, εάν αυτός έχει τέτοιες. Αυτή η συνάρτηση θα κληθεί από τον Code Smell Detector και θα πρέπει να επιστρέψει την λίστα δυσοσμιών που θα δημιουργήσει ο ανιχνευτής.

3.3.3.1 Δήλωση ανιχνευτή

Στο JSON αρχείο που ορίζεται από το πεδίο *DetectorsConfig_Path* του αρχείου παραμετροποίησης "*\\CodeSmellDetector\\MainConfig.json*" υπάρχει μία λίστα δηλώσεων που λέγεται *detectors*. Στη λίστα αυτή, θα πρέπει να προστεθεί η δήλωση του καινούργιου ανιχνευτή.

Το JavaScript αντικείμενο που δηλώνει τον ανιχνευτή υποχρεωτικά θα έχει τα εξής τέσσερα πεδία: *name*, *file*, *description*, *args*. Οι τιμές των τριών πρώτων πεδίων είναι συμβολοσειρές

οι οποίες αποτελούνται από το όνομα του ανιχνευτή, τη σχετική διαδρομή για το αρχείο υλοποίησής του και μια σύντομη περιγραφή αντίστοιχα. Το πεδίο *args* είναι ένα αντικείμενο το οποίο περιγράφει τις παραμέτρους που δέχεται ο συγκεκριμένος ανιχνευτής. Η δήλωση μιας τέτοιας παραμέτρου περιγράφεται στην ενότητα 3.3.3.2.

```
{
  "name": "Too many parameters",
  "file": "Method_argc.js",
  "description": "Detects methods with an excessive amount of parameters",
  "args": {
    "max_argc": {
      "formal_name": "Maximum method arguments",
      "type": "range",
      "range": [6, 10],
      "limits": [2, 15]
    }
  }
},
```

Εικόνα 4: Παράδειγμα δήλωσης ανιχνευτή

3.3.3.2 Δήλωση παραμέτρων ανιχνευτή

Εάν ένας ανιχνευτής δέχεται παραμέτρους, τότε αυτές θα πρέπει να δηλωθούν ως JavaScript αντικείμενα στο πεδίο *args* της δήλωσης του ανιχνευτή. Τα αντικείμενα αυτά έχουν υποχρεωτικά δύο πεδία: *formal_name* και *type*. Το πρώτο πεδίο είναι το όνομα με το οποίο θα εμφανιστεί η παράμετρος στην διεπαφή χρήστη και το δεύτερο πεδίο δηλώνει τον τύπο της παραμέτρου. Υπάρχουν τρεις διαθέσιμοι τύποι: *range*, *enum* και *boolean*.

Range

Ο τύπος *range* είναι ο πιο κοινός, καθώς είναι απαραίτητος στην δημιουργία κλίμακας στην ένταση των δυσοσμιών. Είναι ένα εύρος με κάτω και πάνω άκρα. Το κάτω άκρο συμβολίζει την οριακή τιμή έως την οποία μια μεταβλητή δεν προκαλεί δυσοσμία ενώ το πάνω άκρο ορίζει την ελάχιστη τιμή που προκαλεί μέγιστη ένταση δυσοσμίας.

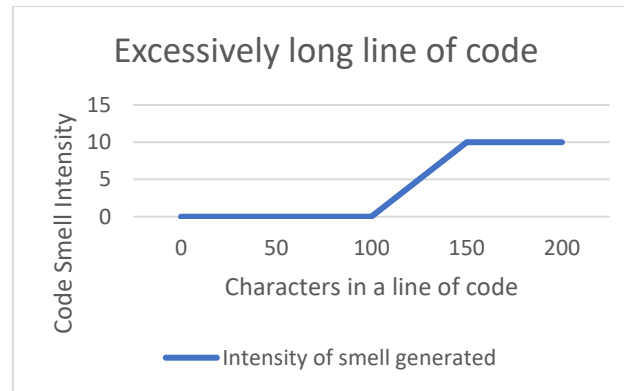
Η παράμετρος του τύπου *range*, πέραν των δυο πεδίων που έχει ήδη, θα πρέπει να ορίζει δυο ακόμη πεδία, τα *range* και *limits*. Το πεδίο *range* είναι το ίδιο το εύρος και η τιμή του είναι ένας πίνακας ο οποίος περιέχει στην πρώτη και δεύτερη θέση του το κάτω και το πάνω άκρο του εύρους αντίστοιχα. Το πεδίο *limits* είναι οι ακρότατες τιμές που μπορεί να δεχτεί το εύρος και αυτές αποθηκεύονται σε έναν πίνακα δύο θέσεων, όπου στην πρώτη θέση βρίσκεται το κάτω ακρότατο ενώ στην δεύτερη θέση το πάνω.

```
"max_line_len": {
  "formal_name": "Maximum length of a line in characters",
  "type": "range",
  "range": [100, 150],
  "limits": [40, 200]
}
```

Εικόνα 5: Δήλωση μιας παραμέτρου τύπου *range*.

Ένα παράδειγμα που εξηγεί τη χρησιμότητα του τύπου αυτού είναι το εξής: θεωρούμε τον ανιχνευτή *“Excessively long line of code”* ο οποίος δέχεται την παράμετρο *max_line_len* (μέγιστο μήκος γραμμής) με τύπο *range* και με την τιμή [100, 150]. Οποιαδήποτε γραμμή

κώδικα με μήκος μικρότερο ή ίσο από 100 χαρακτήρες δεν θα χαρακτηριστεί ως δυσσομία. Οι γραμμές κώδικα με μήκος μεταξύ 100 και 150 χαρακτήρων θα προκαλέσουν δυσσομία με ένταση από 0.01 έως 9.99. Τέλος οι γραμμές με μήκος μεγαλύτερο ή ίσο από 150 χαρακτήρες θα δημιουργήσουν δυσσομία με τη μέγιστη ένταση, δηλαδή 10.



Ο μαθηματικός τύπος για τον υπολογισμό της έντασης της δυσσομίας που προκύπτει από μία μεταβλητή x όταν η παράμετρος με τύπο `range` που της αντιστοιχεί έχει τιμή $[Y, Z]$ είναι:

$$Intensity(x, [Y, Z]) = \begin{cases} 0, & x \leq Y \\ 10, & x \geq Z \\ \frac{x - Y}{Z - Y} * 10, & x \in (Y, Z) \end{cases}$$

Enum

Ο τύπος `enum` χρησιμοποιείται όπως τα `enumerations` στη γλώσσα C++ με τη διαφορά ότι δεν αντιστοιχεί σε αριθμό, αλλά σε οποιοδήποτε αντικείμενο. Για την υλοποίηση αυτού υπάρχει τουλάχιστον άλλη μια «κρυφή» παράμετρος που έχει το ρόλο ενός λεξικού που κρατάει ζεύγη κλειδιού – τιμής. Επομένως, μία παράμετρος με τύπο `enum` πρέπει να έχει ένα κλειδί και το όνομα του λεξικού στο οποίο αναφέρεται αυτό το κλειδί.

Οι παράμετροι τύπου `enum` απαιτούν δυο επιπλέον πεδία πέρα των βασικών. Το πρώτο είναι το πεδίο `dict` του οποίου η τιμή πρέπει να είναι το όνομα του λεξικού στο οποίο αντιστοιχεί η παράμετρος. Το δεύτερο πεδίο έχει όνομα `val` και η τιμή του πρέπει να είναι ένα από τα κλειδιά του λεξικού που αναφέρει το πεδίο `dict`.

Τα λεξικά που χρησιμοποιούνται σε κάποιον ανιχνευτή πρέπει να δηλωθούν ως παράμετροι αλλά δεν πρέπει να έχουν τα πεδία `formal_name` και `type`.

```
"dict0": {
  "UpperCamelCase": "([A-Z][a-z0-9]*)(\\d|([A-Z0-9][a-z0-9]+))*([A-Z])?",
  "lowerCamelCase": "[a-z]+(\\d|([A-Z0-9][a-z0-9]+))*([A-Z])?",
  "snake_case": "([a-z]*_?[a-z]*)*",
  "any": "[\\s\\S] *"
},
"class_names": {
  "formal_name": "Class names",
  "val": "UpperCamelCase",
  "type": "enum",
  "dict": "dict0"
},
```

Εικόνα 6: Δήλωση μιας παραμέτρου τύπου `enum` και του λεξικού που της αντιστοιχεί.

Boolean

Ο τύπος `boolean` είναι ο πιο απλός από τους τρεις καθώς οι παράμετροι με τον τύπο αυτό δέχονται δύο τιμές: αληθής ή ψευδής. Η δήλωση μιας τέτοιας παραμέτρου θα πρέπει να έχει και ένα πεδίο `val` του οποίου η τιμή μπορεί να είναι `true` ή `false`.

```
"public_only": {
  "formal_name": "Public only",
  "val": true,
  "type": "boolean"
}
```

Εικόνα 7: Δήλωση μιας παραμέτρου τύπου `boolean`.

3.3.3.3 Υλοποίηση ανιχνευτή

Η υλοποίηση του ανιχνευτή γίνεται στο JavaScript αρχείο που αναγράφεται στο πεδίο `file` της δήλωσης του ανιχνευτή. Εκεί, μέσα στο αντικείμενο `exports.module`⁴, πρέπει να οριστεί μια ασύγχρονη συνάρτηση με όνομα `callback` η οποία δέχεται δύο παραμέτρους.

Η ασύγχρονη συνάρτηση `callback` θα κληθεί από το πρόγραμμα με πρώτη παράμετρο τον πίνακα συμβόλων, η δομή του οποίου περιγράφεται στην ενότητα 3.2.2, και με δεύτερη παράμετρο το αντικείμενο με τις παραμέτρους του συγκεκριμένου ανιχνευτή. Δεν επιτρέπεται να μεταβάλλει το περιεχόμενο του πίνακα συμβόλων καθώς αυτός χρησιμοποιείται από όλους τους ανιχνευτές. Η συνάρτηση αυτή αναμένεται να επιστρέψει μια λίστα από δυσοσμίες οι οποίες ανιχνευτήκαν. Η κάθε δυσοσμία είναι ένα αντικείμενο το οποίο πρέπει να ακολουθεί αυστηρά την εξής καθορισμένη δομή:

```
{
  msg:  ,
  src: {
    file:  ,
    line:  ,
    col:  ,
    struct:  ,
    method:
  },
  lvl:
}
```

Εικόνα 8: Πρότυπο του αντικειμένου μιας δυσοσμίας κώδικα.

⁴ Το αντικείμενο `module.exports` είναι μια ειδική μεταβλητή που βρίσκεται σε κάθε JavaScript αρχείο μιας Node.js εφαρμογής. Ό,τι ανατεθεί σε αυτή την μεταβλητή θα είναι ορατό σε όποιο κομμάτι κώδικα χρησιμοποιεί το συγκεκριμένο `module` [6].

| Πεδίο | Τύπος τιμής | Περιγραφή τιμής |
|--------|-------------|---|
| msg | string | Μήνυμα που περιγράφει τη δυσοσμία. |
| file | string | Η πλήρης διαδρομή για το αρχείο με τη δυσοσμία. |
| line | int | Η γραμμή του αρχείου στην οποία υπάρχει η δυσοσμία. Εάν είναι άγνωστη ο αριθμός 1 συμβολίζει την αρχή του αρχείου. |
| col | int | Η στήλη του αρχείου στην οποία υπάρχει η δυσοσμία. Εάν είναι άγνωστη ο αριθμός 1 συμβολίζει την αρχή του αρχείου. |
| struct | string | Η δομή στην οποία βρέθηκε η δυσοσμία. Εάν δεν πηγάζει από δομή τότε πρέπει να έχει null τιμή. |
| method | string | Η μέθοδος στην οποία βρέθηκε η δυσοσμία. Εάν δεν πηγάζει από μέθοδο τότε πρέπει να έχει null τιμή. |
| lvl | number | Πραγματικός αριθμός που συμβολίζει την ένταση της δυσοσμίας. Πρέπει να έχει 2 δεκαδικά ψηφία και να ισχύει $0 < lvl \leq 10$. |

Πίνακας 1: Τα πεδία ενός αντικειμένου δυσοσμίας κώδικα

Στο module Utility.js υπάρχουν βοηθητικές συναρτήσεις που χρησιμοποιούνται από τους περισσότερους ανιχνευτές για τη δημιουργία του αντικειμένου δυσοσμίας, υπολογισμό έντασης δυσοσμίας και εξαγωγή δεδομένων από τον πίνακα συμβολών. Η τεκμηρίωση τους βρίσκεται εκεί.

3.3.4 Ανιχνευτές δυσοσμίας

Παρακάτω βρίσκεται η λίστα των 17 ανιχνευτών δυσοσμίας που υλοποιήθηκαν στα πλαίσια αυτής της εργασίας, καθώς και οι παράμετροι που δέχονται και μια σύντομη εξήγηση της λειτουργίας τους.

3.3.4.1 Too many method parameters

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|----------|-----------------|
| range | max_args | [4, 7] |

Πίνακας 2: Παράμετροι του ανιχνευτή "Too many method parameters"

Μετράει το πλήθος των παραμέτρων της κάθε μεθόδου στον πίνακα συμβόλων. Δημιουργείται μία δυσοσμία για κάθε μέθοδο η οποία έχει περισσότερες παραμέτρους από το κάτω άκρο του εύρους max_args.

3.3.4.2 Too many method literals

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|--------------|-----------------|
| range | max_literals | [15, 30] |

Πίνακας 3: Παράμετροι του ανιχνευτή "Too many method literals"

Μετράει το πλήθος των σταθερών τιμών της κάθε μεθόδου στον πίνακα συμβόλων. Δημιουργείται μία δυσοσμία για κάθε μέθοδο η οποία έχει περισσότερες σταθερές τιμές από το κάτω άκρο του εύρους max_literals.

3.3.4.3 Too many method locals

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|------------|-----------------|
| range | max_locals | [10, 24] |

Πίνακας 4: Παράμετροι του ανιχνευτή "Too many method literals"

Μετράει το πλήθος των τοπικών δηλώσεων της κάθε μεθόδου στον πίνακα συμβόλων. Δημιουργείται μία δυσσομία για κάθε μέθοδο η οποία έχει περισσότερες τοπικές δηλώσεις από το κάτω άκρο του εύρους *max_locals*.

3.3.4.4 Long method

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|----------------|-----------------|
| range | max_statements | [60, 100] |
| range | max_lines | [70, 120] |

Πίνακας 5: Παράμετροι του ανιχνευτή "Long method"

Μετράει το πλήθος των statements και των γραμμών της κάθε μεθόδου στον πίνακα συμβόλων. Δημιουργείται μια δυσσομία για κάθε μέθοδο η οποία έχει περισσότερα statements ή γραμμές από τα κάτω άκρα των αντίστοιχων ευρών. Στην περίπτωση που προκύψει δυσσομία και από τους δύο ελέγχους, χρησιμοποιείται η πιο ισχυρή από τις δύο για τον υπολογισμό της συνολικής έντασης.

3.3.4.5 Cyclomatic complexity

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|--------------|-----------------|
| Range | max_loops | [6, 12] |
| Range | max_branches | [16, 28] |

Πίνακας 6: Παράμετροι του ανιχνευτή "Cyclomatic complexity"

Μετράει το πλήθος των επαναλήψεων και των branching statements της κάθε μεθόδου στον πίνακα συμβόλων. Δημιουργείται μια δυσσομία για κάθε μέθοδο που έχει περισσότερες επαναλήψεις ή branching statements από τα κάτω άκρα των αντίστοιχων ευρών. Στην περίπτωση που προκύψει δυσσομία και από τους δύο ελέγχους, χρησιμοποιείται η πιο ισχυρή από τις δύο για τον υπολογισμό της συνολικής έντασης.

3.3.4.6 Excessively long identifier

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|------------|-----------------|
| Range | max_id_len | [25, 45] |

Πίνακας 7: Παράμετροι του ανιχνευτή "Excessively long identifier"

Μετράει το πλήθος των χαρακτήρων του κάθε αναγνωριστικού στον πίνακα συμβόλων. Δημιουργείται μία δυσσομία για κάθε αναγνωριστικό που έχει περισσότερους χαρακτήρες από το κάτω άκρο του εύρους *max_id_len*.

3.3.4.7 Excessively long line of code

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|--------------|-----------------|
| Range | max_line_len | [130, 190] |

Πίνακας 8: Παράμετροι του ανιχνευτή "Excessively long identifier"

Μετράει το πλήθος των χαρακτήρων της κάθε γραμμής κώδικα είτε από source file είτε από header file. Αυτή η καταμέτρηση παραλείπει οποιοδήποτε είδος σχολίου. Δημιουργείται μία δυσοσμία για κάθε γραμμή που έχει περισσότερους χαρακτήρες από το κάτω άκρο του εύρους *max_line_len*.

3.3.4.8 Naming conventions

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|-------------------|-----------------|
| Enum | class_names | UpperCamelCase |
| Enum | method_names | snake_case |
| enum | var_names | any |
| Range | max_chars_ingored | [0, 15] |

Πίνακας 9: Παράμετροι του ανιχνευτή "Naming conventions"

Προσπαθεί να συσχετίσει κάθε αναγνωριστικό με την αντίστοιχη κανονική έκφραση που ορίζει ο χρήστης μέσω των παραμέτρων *class_names*, *method_names* και *var_names*. Εάν κάποιο αναγνωριστικό δεν καλύπτεται πλήρως από την κανονική έκφραση που του αντιστοιχεί, το μήκος της μεγαλύτερης υπό-ακολουθίας χαρακτήρων που δεν ταιριάζει με την κανονική έκφραση χρησιμοποιείται για τον υπολογισμό της έντασης της δυσοσμίας.

Ο χρήστης του συστήματος μπορεί να επιλέξει μεταξύ των κανονικών εκφράσεων που καλύπτουν τα προγραμματιστικά στυλ: "*UpperCamelCase*", "*lowerCamelCase*", "*snake_case*". Μπορεί επίσης να επιλέξει και το "*any*" το οποίο δέχεται οποιοδήποτε τρόπο γραφής των αναγνωριστικών και έχει τη δυνατότητα να προσθέσει δικές του κανονικές εκφράσεις με όνομα της επιλογής του στο αρχείο παραμετροποίησης *DetectorsConfig.json*.

3.3.4.9 Nested scopes

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|-------------------|-----------------|
| Range | max_nested_scopes | [4, 7] |

Πίνακας 10: Παράμετροι του ανιχνευτή "Nested scopes"

Μετράει το μέγιστο πλήθος εμφωλευμένων μπλοκ κώδικα της κάθε μεθόδου του πίνακα συμβόλων. Δημιουργείται μία δυσοσμία για κάθε μέθοδο που έχει περισσότερα εμφωλευμένα μπλοκ κώδικα από το κάτω άκρο του εύρους *max_nested_scopes*.

3.3.4.10 Large class

| Τύπος | Όνομα | Ενδεικτική τιμή |
|---------|-------------|-----------------|
| Range | max_methods | [8, 17] |
| Range | max_fields | [6, 16] |
| Boolean | public_only | true |

Πίνακας 11: Παράμετροι του ανιχνευτή "Large class"

Μετράει το πλήθος των μεθόδων και των πεδίων της κάθε δομής στον πίνακα συμβόλων. Δημιουργείται μια δυσσομία για κάθε δομή που έχει περισσότερες μεθόδους ή πεδία από τα κάτω άκρα των αντίστοιχων ευρών. Στην περίπτωση που προκύψει δυσσομία και από τους δύο ελέγχους, χρησιμοποιείται η πιο ισχυρή από τις δύο για τον υπολογισμό της συνολικής έντασης.

Όταν η μεταβλητή *public_only* έχει την τιμή true, για τον υπολογισμό του πλήθους των μεθόδων και των πεδίων της δομής χρησιμοποιούνται μόνο οι μέθοδοι και τα πεδία που έχουν δηλωθεί ως public.

3.3.4.11 Large file

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|------------------|-----------------|
| Range | max_src_lines | [500, 1200] |
| Range | max_header_lines | [200, 400] |

Πίνακας 12: Παράμετροι του ανιχνευτή "Large file"

Μετράει το πλήθος των γραμμών κάθε αρχείου επικεφαλίδα και πηγαίου κώδικα των οποίων τα μονοπάτια βρίσκονται σε ξεχωριστές λίστες στον πίνακα συμβόλων. Δημιουργείται μια δυσσομία για κάθε αρχείο που έχει περισσότερες γραμμές από το κάτω άκρο των αντίστοιχων ευρών.

3.3.4.12 Inappropriate intimacy

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|-----------------|-----------------|
| Range | max_member_expr | [50, 120] |

Πίνακας 13: Παράμετροι του ανιχνευτή "Inappropriate intimacy"

Μετράει το πλήθος των member expressions από κάθε μια δομή του πίνακα συμβόλων σε οποιαδήποτε άλλη. Δημιουργείται μία δυσσομία κάθε φορά που αυτή η μέτρηση ξεπερνάει το κάτω άκρο του εύρους *max_nested_scopes*.

3.3.4.13 Non-virtual base destructor

Αυτός ο ανιχνευτής δεν δέχεται παραμετροποίηση. Δημιουργεί μια δυσσομία για κάθε δομή που έχει κληρονόμους αλλά ο destructor της δεν είναι virtual.

3.3.4.14 Redundant protected access

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|------------------------|-----------------|
| Range | max_unneeded_protected | [0, 10] |

Πίνακας 14: Παράμετροι του ανιχνευτή "Redundant protected access"

Μετράει το πλήθος των πεδίων και των μεθόδων με protected access της κάθε δομής του πίνακα συμβόλων που δεν έχει κληρονόμους. Δημιουργείται μία δυσσομία κάθε φορά που αυτό το πλήθος ξεπερνάει το κάτω άκρο του εύρους *max_unneeded_protected*.

3.3.4.15 Excessive function overloading

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|--------------|-----------------|
| Range | max_overload | [6, 14] |

Πίνακας 15: Παράμετροι του ανιχνευτή "Excessive function overloading"

Μετράει το πλήθος των υπερφορτώσεων που δέχεται κάθε μέθοδος του πίνακα συμβόλων. Δημιουργείται μία δυσσομία κάθε φορά που αυτό το πλήθος ξεπερνάει το κάτω άκρο του εύρους *max_overload*.

3.3.4.16 Too many dependencies

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|-------------------------|-----------------|
| Range | max_direct_dependencies | [6, 14] |

Πίνακας 16: Παράμετροι του ανιχνευτή "Excessive function overloading"

Μετράει το πλήθος των άμεσων εξαρτήσεων που έχει κάθε δομή του πίνακα συμβόλων σε άλλες δομές. Δημιουργείται μία δυσσομία κάθε φορά που αυτό το πλήθος ξεπερνάει το κάτω άκρο του εύρους *max_direct_dependencies*.

3.3.4.17 Circular dependency

| Τύπος | Όνομα | Ενδεικτική τιμή |
|-------|----------------|-----------------|
| Range | max_circle_len | [6, 14] |

Πίνακας 17: Παράμετροι του ανιχνευτή "Circular dependency"

Χρησιμοποιώντας τον πίνακα συμβόλων δημιουργεί ένα κατευθυνόμενο γράφο εξαρτήσεων των δομών. Στην συνέχεια βρίσκει όλους τους στοιχειώδεις κύκλους του γράφου χρησιμοποιώντας τον αλγόριθμο του Donald B. Johnson [4]. Δημιουργείται μία δυσσομία για κάθε τέτοιο κύκλο του οποίου το μήκος είναι μεγαλύτερο από το κάτω άκρο του εύρους *max_circle_len*.

3.3.5 Διεπαφή χρήστη

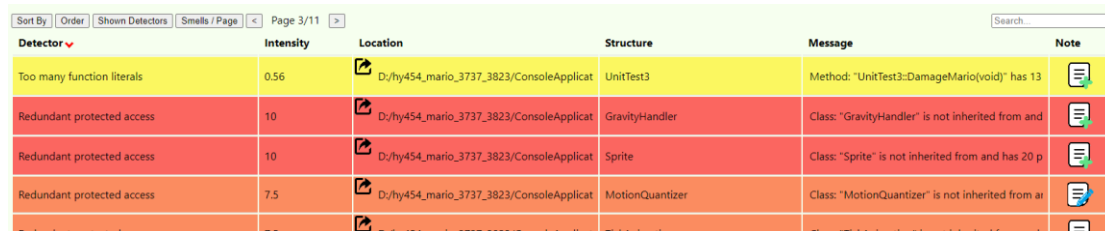
Η διεπαφή χρήστη είναι υλοποιημένη χρησιμοποιώντας το Electron. Το Electron χρησιμοποιεί το chromium engine⁵ για να δημιουργήσει την διεπαφή χρήστη και συνδυάζεται με το Node.js με σκοπό να προσφέρει όλες τις λειτουργίες μιας εφαρμογής desktop.

Η αρχικοποίηση του Electron κατορθώνεται με ένα script το οποίο έχει την ευθύνη να δημιουργήσει το παράθυρο της διεπαφής και να θέσει το index.html. Στη συνέχεια ο προγραμματιστής μπορεί να γράψει HTML, CSS και JavaScript κώδικα όπως θα έκανε για έναν περιηγητή ή για το Node.js.

Το Code Smell Detector είναι single page application (SPA) το οποίο σημαίνει ότι έχει ένα μοναδικό αρχείο HTML και η εναλλαγή μεταξύ καρτελών επιτυγχάνεται με JavaScript κώδικα ο οποίος κρύβει και εμφανίζει διάφορα τμήματα της σελίδας.

Αφού τελειώσει η αρχικοποίηση του backend, δημιουργούνται τα μοναδικά αντικείμενα των τριών singleton κλάσεων⁶: *SmellRenderer*, *DetectorRenderer*, *StatsRenderer*. Η κάθε μία από αυτές τις κλάσεις σε συνδυασμό με το index.html και το αντίστοιχο CSS αρχείο υλοποιεί μία διαφορετική καρτέλα.

3.3.5.1 Καρτέλα λίστας δυσοσμίων



| Detector | Intensity | Location | Structure | Message | Note |
|----------------------------|-----------|--|-----------------|--|------|
| Too many function literals | 0.56 | D:/hy454_mario_3737_3823/ConsoleApplicat | UnitTest3 | Method: "UnitTest3::DamageMario(void)" has 13 | |
| Redundant protected access | 10 | D:/hy454_mario_3737_3823/ConsoleApplicat | GravityHandler | Class: "GravityHandler" is not inherited from and | |
| Redundant protected access | 10 | D:/hy454_mario_3737_3823/ConsoleApplicat | Sprite | Class: "Sprite" is not inherited from and has 20 p | |
| Redundant protected access | 7.5 | D:/hy454_mario_3737_3823/ConsoleApplicat | MotionQuantizer | Class: "MotionQuantizer" is not inherited from a | |
| Redundant protected access | 7.5 | D:/hy454_mario_3737_3823/ConsoleApplicat | TileAnimation | Class: "TileAnimation" is not inherited from and | |

Εικόνα 9: Καρτέλα λίστας δυσοσμίων.

Η καρτέλα λίστας δυσοσμίων υλοποιείται από την κλάση *SmellRenderer*. Η κλάση αυτή έχει μια κύρια συνάρτηση, την *render*, της οποίας ο ρόλος είναι να δημιουργεί τη λίστα των δυσοσμίων και να την εμφανίζει στον χρήστη. Η κλάση κρατάει επίσης πληροφορίες για το φιλτράρισμα της λίστας, τις οποίες τις χειρίζεται ο χρήστης μέσω της διεπαφής. Αυτές είναι:

| | |
|--------|-----------------|
| enum | sort_by |
| enum | order |
| list | shown_detectors |
| string | text_filter |

Πίνακας 18: *smell renderer* filtering data

Τα κουμπιά *Sort By*, *Order* και *Show only* είναι dropdown menus τα οποία έχουν action listeners που επιτρέπουν στον χρήστη να αλλάζει τις μεταβλητές από τον Πίνακα 0. Στο ίδιο ύψος, στα δεξιά, το πεδίο κειμένου *Search* προσφέρει στον χρήστη τη δυνατότητα να βάλει κάποιο κείμενο στη μεταβλητή *text_filter* η οποία σε συνδυασμό με την *shown_detectors* θα καθορίσει ποιες δυσοσμίες θα εμφανιστούν τελικά στη λίστα.

⁵ Chromium engine: -----

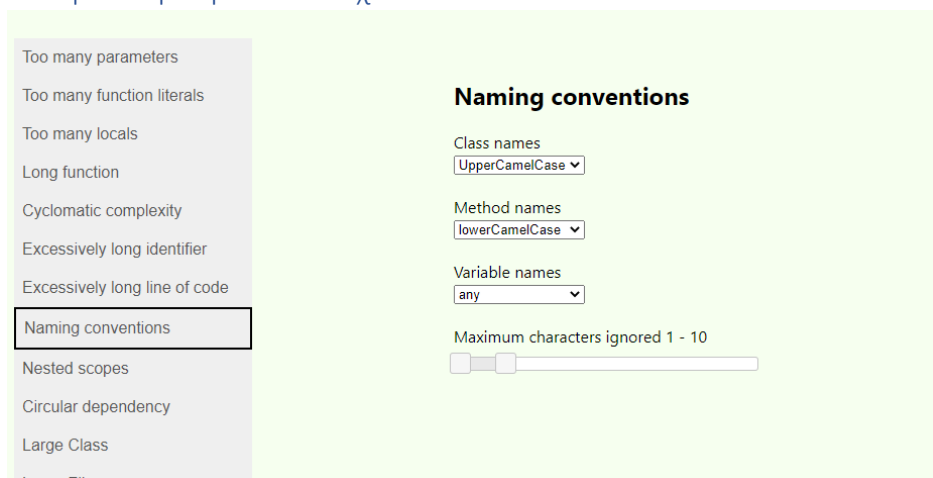
⁶ Singleton class: είναι μία κλάση που κατά την εκτέλεση του προγράμματος θα υπάρξει ως ένα μοναδικό στιγμιότυπο.

Στη λίστα, για κάθε δυσοσμία παρουσιάζονται με την ίδια σειρά οι εξής πληροφορίες:

- Ανιχνευτής
- Ένταση δυσοσμίας (δεκαδικός αριθμός μεταξύ 0 και 10)
- Πηγή
 - Αρχείο
 - Γραμμή
 - Στήλη
- Λεπτομερές μήνυμα με πληροφορίες για τη δυσοσμία

Ο χρήστης έχει τη δυνατότητα να διαβάσει, προσθέσει, διαγράψει ή να επεξεργαστεί μια σημείωση για κάθε δυσοσμία με το εικονίδιο στα δεξιά.

3.3.5.2 Καρτέλα ρυθμίσεων ανιχνευτών



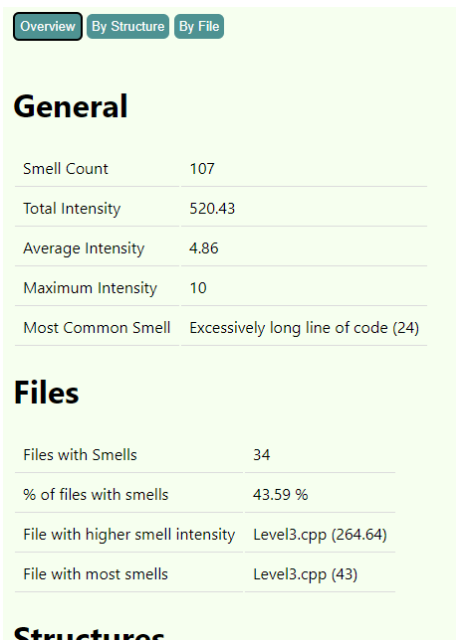
Εικόνα 10: Καρτέλα ρυθμίσεων ανιχνευτών (Ανιχνευτής “Naming conventions”).

Η καρτέλα ρυθμίσεων των ανιχνευτών δυσοσμίας υλοποιείται από την κλάση *DetectorRenderer*. Η κλάση αυτή έχει μια κύρια συνάρτηση, την *render*, η οποία καλείται μια φορά κατά την αρχικοποίηση του προγράμματος και ο ρόλος της είναι να δημιουργεί την γραφική διεπαφή των ανιχνευτών με βάση τις παραμέτρους που έχουν οριστεί στο αρχείο “*DetectorsConfig.json*”.

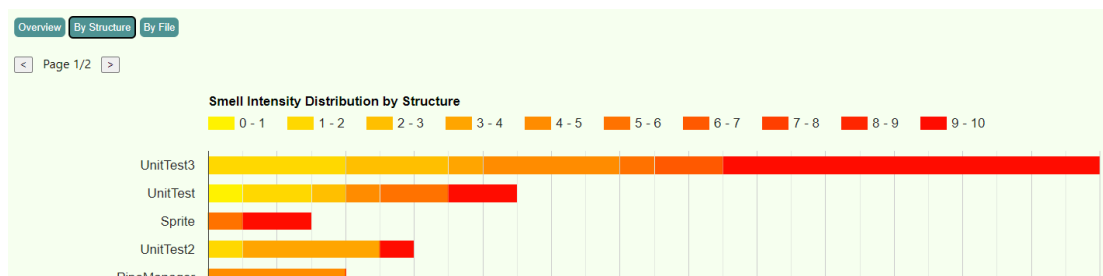
3.3.5.3 Καρτέλα στατιστικών

Η καρτέλα στατιστικών έχει εσωτερικά τις τρεις καρτέλες: *Overview*, *By Structure*, *By File*

Η καρτέλα *Overview* εμφανίζει γενικά στατιστικά που προκύπτουν από την ανίχνευση δυσοσμιών τα οποία εμφανίζονται σε μορφή λίστας. Οι καρτέλες *By Structure* και *By File* εμφανίζουν ένα γράφημα κατανομής της έντασης των δυσοσμιών κατά δομή και κατά αρχείο αντίστοιχα.



Εικόνα 11: Καρτέλα στατιστικών - Overview



Εικόνα 12: Καρτέλα στατιστικών - By Structure

4. Οδηγίες Χρήσης

4.1 Εγκατάσταση

Το σύστημα χρησιμοποιεί αρκετά εργαλεία εκ των οποίων μερικά έχουν πολύπλοκη εγκατάσταση. Παρακάτω βρίσκονται αναλυτικές οδηγίες για την εγκατάσταση κάθε μεγάλου τμήματος του συστήματος. Κάθε τμήμα έχει δικές του εξαρτήσεις οι οποίες αναγράφονται.

4.1.1 Εγκατάσταση του Clang

Το Symbol Table Export δημιουργεί έναν πίνακα συμβόλων χρησιμοποιώντας το Clang που είναι το front-end του μεταγλωττιστή LLVM. Παρακάτω βρίσκονται οι οδηγίες εγκατάστασης του Clang σε Windows 10 με χρήση Visual Studio.

Προαπαιτούμενα:

- [Git](#)
- [CMake](#)
- [Python 3.0+](#)

1. Δημιουργείτε έναν καινούργιο φάκελο στον οποίο θα εγκατασταθεί ο LLVM.
2. Χρησιμοποιώντας το τερματικό:
 - a. Μεταβείτε σε αυτό τον φάκελο (`$ cd C:\...`)
 - b. Τρέξτε την εντολή:
`$ git clone --config core.autocrlf=false https://github.com/llvm/llvm-project.git`
 - c. Μεταβείτε στον φάκελο που έχει δημιουργηθεί (`$ cd llvm-project`)
 - d. Τρέξτε την εντολή:
`cmake -DLLVM_ENABLE_PROJECTS=clang -G "Visual Studio 16 2019" -Ax64 -Thost=x64 llvm`
3. Ανοίξτε το `"llvm-project\LLVM.sln"` χρησιμοποιώντας την ίδια έκδοση του Visual Studio που χρησιμοποιήσατε στο cmake (π.χ. VS 2019).
4. Από το Visual Studio:
 - a. Από την πάνω μπάρα επιλέξτε configuration Release & x64
 - b. Δεξιά από το Solution Explorer ανοίξτε τον φάκελο CMakePredefinedTargets
 - c. Πατήστε με δεξί κλικ πάνω στο `ALL_BUILD`, επιλέξτε properties και στο General > C++ Language Standard επιλέξτε το "ISO C++ 17 Standard". Κάνετε αποθήκευση και βγαίνετε από τα properties.
 - d. Πατήστε πάλι με δεξί κλικ πάνω στο `ALL_BUILD` και πατήστε `build` (η μεταγλώττιση του LLVM μπορεί να πάρει 1-3 ώρες).
 - e. Τέλος το μήνυμα "build success" θα σας ενημερώσει για την επιτυχή μεταγλώττιση.

4.1.2 Εγκατάσταση του Symbol Table Export

Το Symbol Table Export είναι ένα C++ πρόγραμμα που χρησιμοποιεί τον LLVM για να μεταγλωττίσει και να αντλήσει πληροφορίες από το εξεταζόμενο πρόγραμμα τις οποίες στη συνέχεια θα τις αποθηκεύσει σε έναν πίνακα συμβόλων που θα χρησιμοποιηθεί αργότερα από τον Code Smell Detector. Παρακάτω βρίσκονται οι οδηγίες εγκατάστασής του στο Visual Studio.

Προαπαιτούμενα:

- LLVM
- πρόσβαση στο [PhivPap/Code-Smell-Detector \(github.com\)](https://github.com/PhivPap/Code-Smell-Detector)
- (προαιρετικά [vcpkg](https://github.com/PhivPap/vcpkg))

1. Εγκατάσταση του jsoncpp
 - Εάν το vcpkg είναι εγκατεστημένο στον υπολογιστή, η εγκατάσταση γίνεται μέσω της εντολής: `$ vcpkg install jsoncpp:x64-windows`
 - Αλλιώς, η διαδικασία εγκατάστασης υπάρχει εδώ: [open-source-parsers/jsoncpp: A C++ library for interacting with JSON. \(github.com\)](https://github.com/open-source-parsers/jsoncpp).
2. Χρησιμοποιώντας το τερματικό, τρέξτε την παρακάτω εντολή στην περιοχή που θέλετε να εγκαταστήσετε το σύστημα:
`$ git clone https://github.com/PhivPap/Code-Smell-Detector`
3. Χρησιμοποιώντας την ίδια έκδοση του Visual Studio με την οποία μεταγλωττίσατε το LLVM δημιουργήστε ένα άδειο C++ project.
4. Με το "Solution Explorer" προσθέστε στο project όλα τα αρχεία κώδικα που βρίσκονται κάτω από τον φάκελο "`\GraphGenerator\`".
5. Από την πάνω μπάρα επιλέξτε configuration Release & x64.
6. Από τις ιδιότητες (properties) του project:
 - Στο General > C++ Language Standard επιλέξτε το "ISO C++ 17 Standard".
 - Στην κατηγορία C/C++ > Additional Include Directories προσθέστε τα path:
 - i. Για τους φακέλους κάτω από το "`\GraphGenerator\`" που περιέχουν header files.
 - ii. "`\Ivm-project\Ivm\include`"
 - iii. "`\Ivm-project\clang\include`"
 - iv. "`\Ivm-project\tools\clang\include`"
 - v. "`\Ivm-project\include`"
 - vi. Για το include directory του jsoncpp (εάν αυτό είναι εγκατεστημένο με το vcpkg τότε είναι: "`\vcpkg\installed\x64-windows\include`")
 - Στην κατηγορία Linker > General > Additional Library Directories προσθέστε τις διαδρομές:
 - i. Για τον φάκελο που περιέχει το jsoncpp.lib.
 - ii. "`\Ivm-project\Release\lib`"
 - iii. Στην κατηγορία Linker > Input > Additional Dependencies προσθέστε όλα τα αρχεία με κατάληξη .lib από τον φάκελο "`\Ivm-project\Release\lib`", καθώς και τα version.lib και jsoncpp.lib.

4.1.3 Εγκατάσταση του Code Smell Detector

Το Code Smell Detector είναι η κύρια εφαρμογή που χρησιμοποιεί τον πίνακα συμβόλων για την ανίχνευση των δυσосμιών και τις προβάλλει στο χρήστη μέσω γραφικής διεπαφής. Το back-end του υλοποιείται με το Node.js και το front-end με το Electron. Παρακάτω βρίσκονται οι οδηγίες για την πλήρη εγκατάσταση της εφαρμογής.

1. Εγκαταστήστε το Node.js χρησιμοποιώντας τον κατάλληλο installer από αυτόν τον σύνδεσμο: [Download | Node.js \(nodejs.org\)](#).
2. Εγκατάσταση του Electron:
 - a) Με το τερματικό μεταβείτε στον φάκελο "`\CodeSmellDetector\code-smell-detector-GUI`".
 - b) Τρέξτε τις εντολές:
 - i. `$ npm install --save-dev @electron-forge/cli`
 - ii. `$ npx electron-forge import`
 - iii. `$ npm install`
 - c) Σε περίπτωση που οι παραπάνω εντολές δεν δούλεψαν λόγω έκδοσης του npm μπορείτε να βρείτε αναλυτικές οδηγίες εγκατάστασης σε αυτόν τον σύνδεσμο: [Import Existing Project - Electron Forge](#).

4.2 Χρήση

4.2.1 Χρήση του Symbol Table Export

Αυτό το πρόγραμμα παράγει τον πίνακα συμβόλων που χρησιμοποιείται αργότερα για την ανίχνευση δυσοσμιών κώδικα. Δέχεται ως είσοδο ένα C++ Project το οποίο πρέπει να μπορεί να το μεταγλωττίσει το LLVM και παράγει ως έξοδο έναν πίνακα συμβόλων σε μορφή JSON αρχείου.

4.2.1.1 Είσοδος

Το project που θα δεχθεί ως είσοδο μπορεί να δοθεί στο πρόγραμμα με δύο τρόπους:

1. Compilation Database
2. Project Directory

Η επιλογή καθορίζεται από τις 2 πρώτες παραμέτρους (command line arguments) του Symbol Table Export, όπως αναγράφεται παρακάτω.

Η τρίτη παράμετρος που δέχεται, είναι η διαδρομή για ένα αρχείο κειμένου το οποίο περιέχει διαδρομές (paths) ανά γραμμή. Οι διαδρομές αυτές, καθορίζουν τα αρχεία που θα αγνοηθούν για τη δημιουργία του πίνακα συμβόλων. Για παράδειγμα, εάν το αρχείο κειμένου περιέχει τη διαδρομή: `"C:/project/skip/"`, τα περιεχόμενα των αρχείων που βρίσκονται μέσα στον φάκελο `skip` δε θα συμπεριληφθούν στον πίνακα συμβόλων. Το αρχείο αυτό μπορεί να είναι άδειο, ωστόσο δεν μπορεί να περιέχει κενές γραμμές.

Η τέταρτη παράμετρος είναι, όπως και η τρίτη, μια διαδρομή για ένα αρχείο κειμένου το οποίο περιέχει namespaces ανά γραμμή. Τα περιεχόμενα αυτών των namespaces δεν θα συμπεριληφθούν στον πίνακα συμβόλων. Το αρχείο αυτό μπορεί να είναι άδειο ωστόσο δεν μπορεί να περιέχει κενές γραμμές.

Η πέμπτη παράμετρος είναι η διαδρομή η οποία καθορίζει πού και με ποιο όνομα θα αποθηκευτεί ο πίνακας συμβόλων.

4.2.1.1.1 Είσοδος: Compilation Database

Αυτή η μέθοδος είναι η προτεινόμενη, καθώς δουλεύει και με εξωτερικές εξαρτήσεις. Ένα Compilation Database είναι ένα JSON αρχείο το οποίο περιέχει μια λίστα από εντολές. Κάθε μια από αυτές υποδεικνύει τον τρόπο με τον οποίο θα μεταγλωττιστεί κάθε τμήμα του project. Μπορεί να δημιουργηθεί με το CMake προσθέτοντας στην εντολή την παράμετρο `"-D CMAKE_EXPORT_COMPILE_COMMANDS=ON"` και χρησιμοποιώντας τα generators *Ninja* ή *Makefile Generators*. Αλλιώς, για Visual Studio projects, το Compilation Database μπορεί να δημιουργηθεί χρησιμοποιώντας την open-source επέκταση *SourceTrail*.

Το Compilation Database θα αποθηκευτεί σε ένα αρχείο με όνομα `"compile_commands.json"`. Μετά από αλλαγή σε αρχείο πηγαίου κώδικα θα πρέπει να ενημερωθεί πριν την εκτέλεση του Symbol Table Export.

Για την επιλογή αυτής της μεθόδου input, ο χρήστης πρέπει να θέσει ως πρώτη παράμετρο τη συμβολοσειρά `"--cmp-db"` και ως δεύτερη παράμετρο τη διαδρομή για τον φάκελο ο οποίος περιέχει το αρχείο `"compile_commands.json"`.

4.2.1.1.2 Είσοδος: Project Directory

Η δεύτερη μέθοδος εισόδου στο Symbol Table Export, είναι χρήσιμη μόνο σε μικρά projects των οποίων οι οδηγίες “#include” αποτελούνται αποκλειστικά από σχετικές ή απόλυτες διαδρομές. Συχνά, αυτή η μέθοδος δεν δουλεύει, καθώς για λόγους ευκολίας πολλά περιβάλλοντα προγραμματισμού, όπως το Visual Studio, προσφέρουν την δυνατότητα στον προγραμματιστή να καθορίσει τους φακέλους αναζήτησης των “#include” οδηγιών. Οι επιλογές αυτές, δεν βρίσκονται σε αρχεία πηγαίου κώδικα, άρα, ένας άλλος μεταγλωττιστής δεν θα μπορέσει να μεταγλωττίσει το πρόγραμμα.

Για την επιλογή αυτής της μεθόδου, ο χρήστης πρέπει να θέσει ως πρώτη παράμετρο την συμβολοσειρά “--src” και ως δεύτερη παράμετρο την διαδρομή για έναν φάκελο ο οποίος πρέπει να περιέχει όλα τα αρχεία πηγαίου κώδικα του project.

4.2.1.2 Έξοδος

Στο τέλος της εκτέλεσης του Symbol Table Export θα τυπωθεί στο τερματικό ένα μήνυμα το οποίο επισημαίνει την επιτυχία ή την αποτυχία της μεταγλώττισης. Σε περίπτωση αποτυχίας ο μεταγλωττιστής θα τυπώσει στο τερματικό “Fatal Error” (το μήνυμα “error” μπορεί να αγνοηθεί). Στην περίπτωση επιτυχίας, στη διαδρομή που έχει ορίσει ο χρήστης στην πέμπτη παράμετρο, θα υπάρχει ο πίνακας συμβόλων ο οποίος στη συνέχεια θα χρησιμοποιηθεί από το Code Smell Detector.

4.2.2 Χρήση του Code Smell Detector

Το Code Smell Detector είναι ένα JavaScript πρόγραμμα που δέχεται ως είσοδο τον πίνακα συμβόλων και οπτικοποιεί τις δυσοσμίες στο χρήστη με βοηθητικό τρόπο.

Πριν την έναρξη, ο χρήστης πρέπει να ανοίξει το “\CodeSmellDetector\MainConfig.json”. Το αρχείο αυτό έχει την εξής μορφή:

```
{
  "ST_Path": "../ST0.json",
  "SmellsCache_Path": "../SmellReports.json",
  "DetectorsConfig_Path": "../DetectorsConfig.json",
  "TextEditorStartUpCommand": "code -g"
}
```

Εικόνα 13: Το αρχείο MainConfig.json

Στο πεδίο *ST_Path*, ο χρήστης πρέπει να θέσει τη σχετική διαδρομή για το JSON αρχείο που έχει παραχθεί από το Symbol Table Export. Εάν θέλει να επιλέξει άλλο αρχείο δηλώσεων των ανιχνευτών δυσοσμίας, μπορεί να αλλάξει την τιμή του τρίτου πεδίου, η δομή του οποίου περιγράφεται στην ενότητα 3.3.3. Τέλος, στην τιμή του τέταρτου πεδίου, έχει την δυνατότητα να βάλει διαφορετική εντολή εκκίνησης επεξεργαστή κειμένου. Η εντολή αυτή χρησιμοποιείται για την ανακατεύθυνση του χρήστη στον πηγαίο κώδικα.

Στην συνέχεια, ο χρήστης μπορεί να εκκινήσει το πρόγραμμα από το τερματικό με την εντολή “npm start”.

4.3 Επέκταση

Το Code Smell Detector μπορεί εύκολα να επεκταθεί σε τρεις τομείς: στους ανιχνευτές δυσοσμίας, στις παραμέτρους των ανιχνευτών δυσοσμίας και στα στατιστικά.

Ένας προγραμματιστής με ελάχιστη γνώση του συστήματος είναι πολύ εύκολο να προσθέσει καινούργιους ανιχνευτές τηρώντας τους κανόνες και ακολουθώντας τις οδηγίες που βρίσκονται στην διεπαφή προγραμματισμού εφαρμογών (API) των ανιχνευτών δυσοσμίας στο κεφάλαιο 3.3.3 και διαβάζοντας την δομή του πίνακα συμβόλων η οποία περιγράφεται στο κεφάλαιο 3.3.2.

Εύκολη επίσης, είναι η προσθήκη καινούργιων τύπων παραμέτρων για τους ανιχνευτές δυσοσμίας. Όπως αναφέρεται και στο κεφάλαιο 3.3.3 ο κάθε ανιχνευτής μπορεί να έχει παραμέτρους. Οι παράμετροι έχουν υποχρεωτικά έναν από τους τρεις προκαθορισμένους τύπους, αυτοί είναι: *range*, *enum* και *boolean*. Η προσθήκη καινούργιου τύπου είναι απλή, ο προγραμματιστής θα πρέπει να ορίσει τα πεδία που χρειάζεται αυτός ο τύπος και στη συνέχεια θα πρέπει να υλοποιήσει την γραφική απεικόνιση και την αλληλεπίδραση του με τον χρήστη. Στη συνάρτηση *render* της κλάσης *DetectorRenderer* υπάρχει μια επανάληψη η οποία διατρέχει όλες τις παραμέτρους όλων των ανιχνευτών που βρίσκονται στο αρχείο *“DetectorsConfig.json”*. Εκεί θα πρέπει να γίνει η αρχικοποίηση της γραφικής διεπαφής της κάθε παραμέτρου του κάθε ανιχνευτή. Τέλος, κατά την αλλαγή κάποιου πεδίου μίας παραμέτρου πρέπει πάντα να καλείται η συνάρτηση *detector_config_changed*.

5. Συμπεράσματα

Είναι σημαντικό να αναφερθεί ότι τα συστήματα που σχεδιάστηκαν και υλοποιήθηκαν επιφέρουν αποτελέσματα που συμπίπτουν με τις αρχικές μας προσδοκίες. Σύμφωνα με τις λειτουργικές προδιαγραφές το *Code Smell Detector* είναι πράγματι πλήρως επεκτάσιμο στο επίπεδο των ανιχνευτών δυσοσμίας, καθώς η προσθήκη ενός νέου ανιχνευτή είναι διαδικασία που ενώ απαιτεί γνώση της διεπαφής προγραμματισμού εφαρμογών και την δομή του πίνακα συμβόλων, δεν απαιτεί καμία γνώση της υλοποίησης του συστήματος. Επιπλέον, όπως αναφέρεται στις λειτουργικές προδιαγραφές, η γραφική διεπαφή είναι εύκολα επεκτάσιμη στο επίπεδο των στατιστικών και τα συστήματα είναι ικανά να λειτουργήσουν ορθά και αποτελεσματικά ακόμη και με μεγάλες εισόδους.

Τέλος, πρέπει να σχολιαστεί η επεκτασιμότητα του συστήματος με βάση έναν πιο γενικό γνώμονα. Ενώ το σύστημα λειτουργεί σύμφωνα με τις προδιαγραφές του, η προσθήκη κάποιων ανιχνευτών είναι αδύνατη χωρίς την επέκταση του πίνακα συμβόλων. Αυτό οφείλεται στο ότι το *Code Smell Detector* είναι περιορισμένο από την πληροφορία που δέχεται. Αυτό το πρόβλημα δεν είναι εύκολα επιλύσιμο, επειδή όσο περισσότερη πληροφορία προστίθεται στον πίνακα συμβόλων, τόσο δυσκολότερη γίνεται η υλοποίηση των ανιχνευτών. Στην πιο ακραία περίπτωση ο πίνακας θα περιείχε ολόκληρο τον πηγαίο κώδικα το οποίο θα σήμαινε ότι το *Code Smell Detector* θα έπρεπε να τον μεταγλωττίσει. Σε αυτήν την περίπτωση όμως, η προσθήκη νέου ανιχνευτή θα απαιτούσε βαθιά γνώση του συστήματος, της γλώσσας C++ και των μεταγλωττιστών.

6. Βιβλιογραφία

- [1] M. Fowler, K. Beck, J. Brant, W. Opdyke και D. Roberts, σε *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
- [2] M. Fowler, «Martin Fowler,» 9 February 2006. [Ηλεκτρονικό]. Available: <https://martinfowler.com/bliki/CodeSmell.html>. [Πρόσβαση 31 July 2021].
- [3] «Clang LLVM,» [Ηλεκτρονικό]. Available: <https://clang.llvm.org>. [Πρόσβαση 17 August 2021].
- [4] D. B. Johnson, «Finding all the elementary circuits of a directed graph,» *SIAM Journal on Computing*, τόμ. IV, αρ. 1, pp. 79-80, 1975.
- [5] K. Busbee, «Branching Statements,» Rebus Press, [Ηλεκτρονικό]. Available: <https://press.rebus.community/programmingfundamentals/chapter/branching-statements/>. [Πρόσβαση 1 August 2021].
- [6] J. Hibbard, «Sitepoint,» 26 November 2019. [Ηλεκτρονικό]. Available: <https://www.sitepoint.com/understanding-module-exports-exports-node-js/>. [Πρόσβαση 21 August 2021].
- [7] ECMA International, «ECMAScript,» σε *ECMA-404 The JSON Data Interchange Standard*, 2017.