

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ**

**ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ**

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**Εργαλεία λογισμικού για αξιολόγηση και βελτίωση ποιότητας κώδικα**

**Φοίβος Γ. Παπαπαναγιωτάκης Μπουσύ**

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 code smell[[1]](#footnote-1) σε προγράμματα γραμμένα στην γλώσσα C++. Το σύστημα οφείλει να εντοπίζει ενδείξεις κακού και δυσανάγνωστου κώδικα και να τις παρουσιάζει στον χρήστη με όσο το δυνατό πιο κατανοητά μηνύματα και στατιστικά. Επιπρόσθετα το σύστημα πρέπει να εύκολα επεκτάσιμο στο επίπεδο ανίχνευσης των code smell καθώς και στον οραματισμό τους.

Για την διεκπεραίωση της εργασίας ασχοληθήκαμε με θεωρητικά προβλήματα όπως το τι είναι ένα code smell όσο και με τεχνικά προβλήματα όπως την ανίχνευση των code smell, την διεπαφή χρήστη 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 πολύ μεγάλα προγράμματα και είναι επεκτάσιμο σύμφωνα με τις λειτουργικές προδιαγραφές.

Περιεχόμενα

[1 Εισαγωγή 5](#_Toc80279215)

[1.1 Αντικείμενο της εργασίας 5](#_Toc80279216)

[1.2 Διάρθρωση της εργασίας 5](#_Toc80279217)

[2 Θεωρητικό υπόβαθρο 6](#_Toc80279218)

[2.1 Δυσοσμία κώδικα 6](#_Toc80279219)

[3 Περιγραφή Συστήματος 7](#_Toc80279220)

[3.1 Προδιαγραφές 7](#_Toc80279221)

[3.2 Αρχιτεκτονική 7](#_Toc80279222)

[3.3 Υλοποίηση 7](#_Toc80279223)

[3.3.1 Εργαλεία & Βιβλιοθήκες 7](#_Toc80279224)

[3.3.2 Δημιουργία πίνακα συμβόλων με το Clang 7](#_Toc80279225)

[3.3.3 Δυναμική εγκατάσταση των ανιχνευτών δυσοσμίας 7](#_Toc80279226)

[3.3.4 Ανιχνευτές δυσοσμίας 8](#_Toc80279227)

[3.3.5 Διεπαφή χρήστη 15](#_Toc80279228)

[4 Οδηγίες Χρήσης 17](#_Toc80279229)

[4.1 Εγκατάσταση 17](#_Toc80279230)

[4.1.1 Εγκατάσταση του LLVM 17](#_Toc80279231)

[4.1.2 Εγκατάσταση του Symbol Table Export 18](#_Toc80279232)

[4.1.3 Εγκατάσταση του Code Smell Detector 19](#_Toc80279233)

[4.2 Χρήση 20](#_Toc80279234)

[4.2.1 Χρήση του Symbol Table Export 20](#_Toc80279235)

[4.2.2 Χρήση του Code Smell Detector 21](#_Toc80279236)

[4.3 Επέκταση 22](#_Toc80279237)

[5 Συμπεράσματα 22](#_Toc80279238)

[5.1 Γνωστά προβλήματα 22](#_Toc80279239)

[5.2 Πιθανές βελτιώσεις 22](#_Toc80279240)

[6 Βιβλιογραφία 22](#_Toc80279241)

# 1 Εισαγωγή

## Αντικείμενο της εργασίας

## Διάρθρωση της εργασίας

# Θεωρητικό υπόβαθρο

## Δυσοσμία κώδικα

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

Δεν υπάρχει απόλυτος ορισμός που να καθορίζει τι είναι δυσοσμία κώδικα και τι όχι, ωστόσο, πολλοί ερευνητές του θέματος συμφωνούν ότι τα παρακάτω είναι δυσοσμίες:

* Συναρτήσεις με υπερβολικό πλήθος παραμέτρων.
* Συναρτήσεις με υπερβολικά literals[[2]](#footnote-2).
* Συναρτήσεις με υπερβολικές τοπικές δηλώσεις.
* Συναρτήσεις με πολλές επαναλήψεις ή branching statements[[3]](#footnote-3)
* Συναρτήσεις με πολλά εμφωλευμένα μπλοκ κώδικα.
* Υπερβολικά μεγάλες συναρτήσεις.
* Αναγνωριστικά πολύ μεγάλου μήκους.
* Αναγνωριστικά που δεν ακολουθούν κάποια τυπογραφική σύμβαση.
* Αναγνωριστικά που δεν εκφράζουν την χρησιμότητα του αντικειμένου το οποίο αναπαριστούν.
* Υπερβολικά μεγάλες γραμμές κώδικα.
* Υπερβολικά μεγάλα αρχεία κώδικα.
* Κυκλική εξάρτηση
* Δομές (classes) 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 δυσοσμίες. Μερικές από αυτές για την γλώσσα C++ είναι:

* Συναρτήσεις και τελεστές που υπερφορτώνονται υπερβολικά συχνά
* Downcasting
* …

Κάποιες από τις δυσοσμίες, εκ φύσεως, είναι εύκολες στην ανίχνευση από έναν προγραμματιστή ή έναν αναλυτή πηγαίου κώδικα (ή μεταγλωττιστή).

# Περιγραφή Συστήματος

## Προδιαγραφές

## Αρχιτεκτονική

## Υλοποίηση

### Εργαλεία & Βιβλιοθήκες

### Δημιουργία πίνακα συμβόλων με το Clang

### Δυναμική εγκατάσταση των ανιχνευτών δυσοσμίας

Στα πλαίσια αυτής της εργασίας έχουν υλοποιηθεί 17 ανιχνευτές δυσοσμίας. Ο κάθε ανιχνευτής έχει μια συνάρτηση η οποία καλείται για την εύρεση των δυσοσμιών. Αυτή η συνάρτηση καλείται με δύο παραμέτρους, τον πίνακα συμβόλων και ένα αντικείμενο “args” το οποίο έχει τις παραμέτρους του ανιχνευτή, εάν αυτός έχει τέτοιες.

### Ανιχνευτές δυσοσμίας

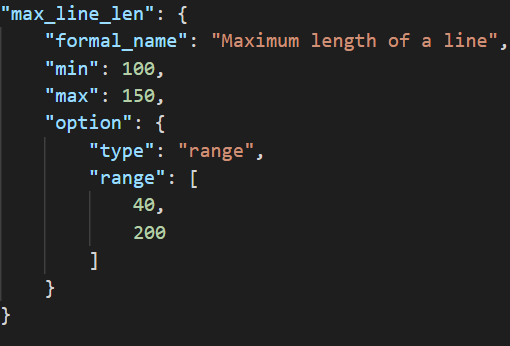
Κάθε ανιχνευτής δέχεται τις παραμέτρους του μέσω της δεύτερης μεταβλητής του, την args. Αυτές οι παράμετροι μπορούν να έχουν έναν από τους τρεις τύπους: range, enum ή boolean.

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.

Για παράδειγμα 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 λιγότερο η ίσο με 100 χαρακτήρες δεν θα χαρακτηρισθεί ως δυσοσμία. Οι γραμμές κώδικα με μήκος μεταξύ 100 και 150 χαρακτήρων θα προκαλέσουν δυσοσμία με ένταση από 0.01 έως 9.99. Τέλος οι γραμμές με μήκος περισσότερο ή ίσο με 150 χαρακτήρες θα δημιουργήσουν δυσοσμία με την μέγιστη ένταση, 10.

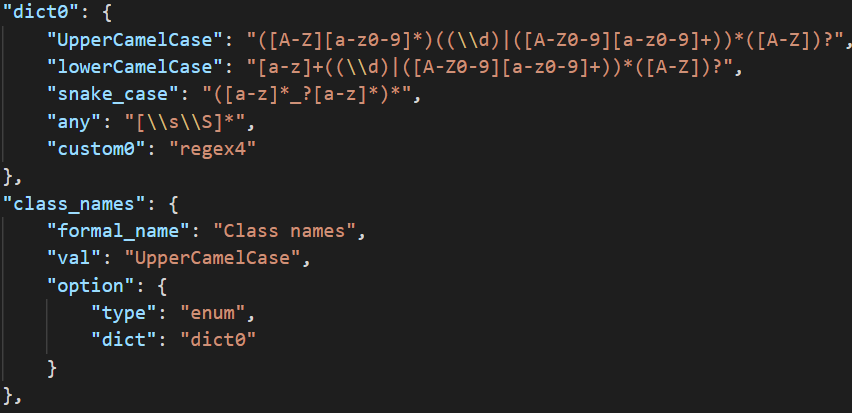


Εικόνα : Δήλωση μιας παραμέτρου τύπου range.

O μαθηματικός τύπος για τον υπολογισμό της έντασης της δυσοσμίας που προκύπτει από μία μεταβλητή x όταν η παράμετρος με τύπο “range” που της αντιστοιχεί έχει τιμή [Y, Z] είναι:

Enum:

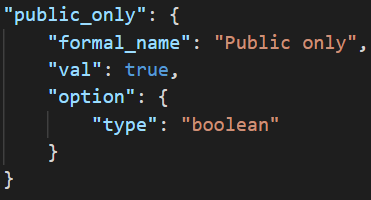
Ο τύπος enum χρησιμοποιείται όπως τα enumerations στην γλώσσα C++ με την διαφορά ότι δεν αντιστοιχεί σε αριθμό αλλά σε οποιοδήποτε αντικείμενο. Για την υλοποίηση αυτού υπάρχει τουλάχιστον άλλη μία “κρυφή” παράμετρος που έχει το ρόλο ενός λεξικού που κρατάει ζεύγη κλειδιού – τιμής. Επομένως, μία παράμετρος με τύπο enum πρέπει να έχει ένα κλειδί και το όνομα του λεξικού στο οποίο αναφέρεται αυτό το κλειδί. Παρακάτω βρίσκεται ένα παράδειγμα από τις παραμέτρους του ανιχνευτή “Naming conventions” με το λεξικό dict0 και την μεταβλητή class\_names τύπου enum.



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

Boolean:

Ο τύπος boolean είναι ο πιο απλός από τους τρείς καθώς οι παράμετροι με τον τύπο αυτό δέχονται δύο τιμές: true (αληθής) ή false (ψευδής). Ένα παράδειγμα χρήσης αυτού του τύπου είναι η παράμετρος public\_only του ανιχνευτή “Large class”. Η δήλωση της παραμέτρου φαίνεται παρακάτω.



Εικόνα : Δήλωση μιας παραμέτρου τύπου boolean.

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

#### Too many method parameters

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_args | [ 4, 7 ] |

Πίνακας 1: 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.

#### Too many method literals

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_literals | [ 15, 30 ] |

Πίνακας 2: 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.

#### Too many method locals

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_locals | [ 10, 24 ] |

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

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

#### Long method

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_statements | [ 60, 100 ] |
| range | max\_lines | [ 70, 120 ] |

Πίνακας 4: Παράμετροι του ανιχνευτή "Long method"

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

#### Cyclomatic complexity

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_loops | [ 6, 12 ] |
| range | max\_branches | [ 16, 28 ] |

Πίνακας 5: Παράμετροι του ανιχνευτή "Cyclomatic complexity"

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

#### Excessively long identifier

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_id\_len | [ 25, 45 ] |

Πίνακας 6: Παράμετροι του ανιχνευτή "Excessively long identifier"

Μετράει το πλήθος των χαρακτήρων του κάθε αναγνωριστικού στον πίνακα συμβόλων. Δημιουργείται μία δυσοσμία για κάθε αναγνωριστικό το οποίο έχει περισσότερους χαρακτήρες από το κάτω άκρο του εύρους max\_id\_len.

#### Excessively long line of code

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_line\_len | [ 130, 190 ] |

Πίνακας 7: 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 η οποία έχει περισσότερους χαρακτήρες από το κάτω άκρο του εύρους max\_line\_len.

#### 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 8: Παράμετροι του ανιχνευτή "Naming conventions"

Προσπαθεί να συσχετίσει κάθε αναγνωριστικό με την αντίστοιχή 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.

Δηλαδή, με τις συγκεκριμένες παραμέτρους, οι μέθοδοι “YYmethod\_a” και “YYmethod\_aY” έχουν το ίδιο μήκος μεγαλύτερης υπό-ακολουθίας το οποίο είναι 2.

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 την δυνατότητα να προσθέσει δικές του κανονικές εκφράσεις με όνομα της επιλογής του στο DetectorsConfig.json.

#### Nested scopes

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_nested\_scopes | [ 4, 7 ] |

Πίνακας 9: Παράμετροι του ανιχνευτή "Nested scopes"

Μετράει το μέγιστο πλήθος εμφωλευμένων μπλοκ κώδικα της κάθε μεθόδου του πίνακα συμβόλων. Δημιουργείται μία δυσοσμία για κάθε μέθοδο η οποία έχει περισσότερα εμφωλευμένα μπλοκ κώδικα από το κάτω άκρο του εύρους max\_nested\_scopes.

#### Large class

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_methods | [ 8, 17 ] |
| range | max\_fields | [ 6, 16 ] |
| boolean | public\_only | true |

Πίνακας 10: Παράμετροι του ανιχνευτή "Large class"

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

Όταν η μεταβλητή “public\_only” έχει την τιμή true, στον υπολογισμό του πλήθος των μεθόδων και των πεδίων της δομής χρησιμοποιούνται μόνο οι μέθοδοι και τα πεδία που έχουν δηλωθεί ως public.

#### Large file

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_src\_lines | [ 500, 1200 ] |
| range | max\_header\_lines | [ 200, 400 ] |

Πίνακας 11: 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 δυσοσμία για κάθε αρχείο που έχει περισσότερες γραμμές από το κάτω άκρο των αντίστοιχων ευρών.

#### Inappropriate intimacy

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_member\_expr | [ 50, 120 ] |

Πίνακας 12: Παράμετροι του ανιχνευτή "Inappropriate intimacy"

Μετράει το πλήθος των member expressions[[4]](#footnote-4) από κάθε μία δομή του πίνακα συμβόλων σε οποιαδήποτε άλλη. Δημιουργείται μία δυσοσμία κάθε φορά που αυτή η μέτρηση ξεπερνάει το κάτω άκρο του εύρους max\_nested\_scopes.

#### Non-virtual base destructor

Αυτός ο ανιχνευτής δεν δέχεται παραμετροποίηση. Δημιουργεί μια δυσοσμία για κάθε δομή η οποία έχει κληρονόμους αλλά ο destructor[[5]](#footnote-5) της δεν είναι virtual[[6]](#footnote-6).

#### Redundant protected access

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_unneeded\_protected | [ 0, 10 ] |

Πίνακας 13: Παράμετροι του ανιχνευτή "Redundant protected access"

Μετράει το πλήθος των πεδίων και των μεθόδων με protected access[[7]](#footnote-7) τις κάθε δομής του πίνακα συμβόλων που δεν έχει κληρονόμους. Δημιουργείται μία δυσοσμία κάθε φορά που αυτό το πλήθος ξεπερνάει το κάτω άκρο του εύρους max\_unneeded\_protected.

#### Excessive function overloading

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_overload | [ 6, 14 ] |

Πίνακας 14: Παράμετροι του ανιχνευτή "Excessive function overloading"

Μετράει το πλήθος των υπερφορτώσεων που δέχεται κάθε μέθοδος του πίνακα συμβόλων. Δημιουργείται μία δυσοσμία κάθε φορά που αυτό το πλήθος ξεπερνάει το κάτω άκρο του εύρους max\_overload.

#### Too many dependencies

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_direct\_dependencies | [ 6, 14 ] |

Πίνακας 15: Παράμετροι του ανιχνευτή "Excessive function overloading"

Μετράει το πλήθος των άμεσων εξαρτήσεων που έχει κάθε δομή του πίνακα συμβόλων σε άλλες δομές. Δημιουργείται μία δυσοσμία κάθε φορά που αυτό το πλήθος ξεπερνάει το κάτω άκρο του εύρους max\_direct\_dependencies.

#### Circular dependency

|  |  |  |
| --- | --- | --- |
| Τύπος | Όνομα | Ενδεικτική τιμή |
| range | max\_circle\_len | [ 6, 14 ] |

Πίνακας 16: Παράμετροι του ανιχνευτή "Circular dependency"

Χρησιμοποιώντας τον πίνακα συμβόλων δημιουργεί ένα κατευθυνόμενο γράφο εξαρτήσεων των δομών. Στην συνέχεια βρίσκει όλους τους στοιχειώδης κύκλους του γράφου χρησιμοποιώντας τον αλγόριθμο του Donald B. Johnson [3]. Δημιουργείται μία δυσοσμία κάθε τέτοιο κύκλο του οποίου το μήκος είναι μεγαλύτερο από το κάτω άκρο του εύρους max\_circle\_len.

### Διεπαφή χρήστη

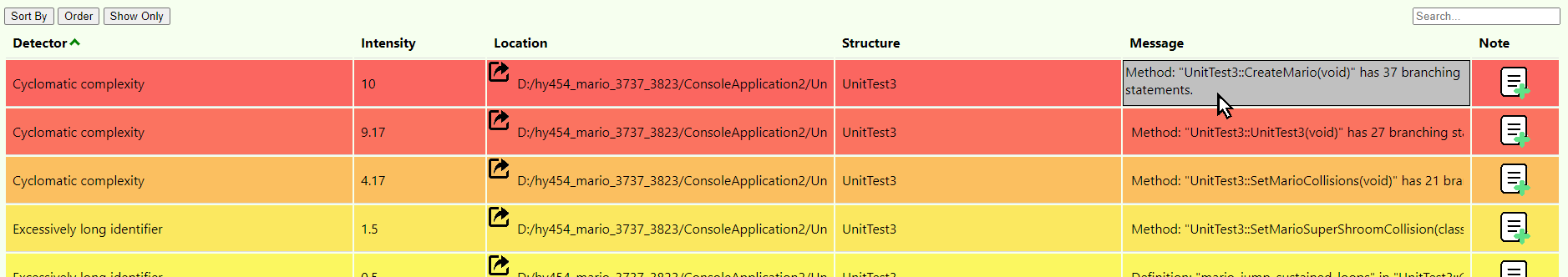
Η διεπαφή χρήστη είναι υλοποιημένη χρησιμοποιώντας το Electron. Το Electron χρησιμοποιεί το chromium engine[[8]](#footnote-8) για να γίνει δημιουργήσει την διεπαφή χρήστη και συνδυάζεται με το node.js για να προσφέρει όλες τις λειτουργίες μίας εφαρμογής desktop.

Η αρχικοποίηση του Electron.js κατορθώνεται με ένα script το οποίο έχει την ευθύνη να δημιουργήσει το παράθυρο της διεπαφής και να θέσει το index.html. Στην 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 κλάσεων[[9]](#footnote-9): SmellRenderer, DetectorRenderer, StatsRenderer. Η κάθε μία από αυτές τις κλάσεις σε συνδυασμό με το index.html και αντίστοιχο CSS αρχείο υλοποιεί μία διαφορετική καρτέλα.

#### Καρτέλα λίστας δυσοσμιών



Η καρτέλα λίστας δυσοσμιών υλοποιείται από την κλάση “SmellRenderer”. Η κλάση αυτή έχει μία κύρια συνάρτηση, την render, της οποίας ο ρόλος είναι να δημιουργεί την λίστα των δυσοσμιών 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 0: 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. Στην ίδια γραμμή, στα δεξιά, το πεδίο κειμένου “Search” προσφέρει στον χρήστη την δυνατότητα να βάλει κάποιο κείμενο στην μεταβλητή “text\_filter” η οποία σε συνδυασμό με την “shown\_detectors” θα καθορίσει ποιες δυσοσμίες θα εμφανιστούν τελικά στην λίστα.

Στην 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 την δυσοσμία

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

#### Καρτέλα ρυθμίσεων ανιχνευτών

#### Καρτέλα στατιστικών

# Οδηγίες Χρήσης

## 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.

### Εγκατάσταση του LLVM

Το Symbol Table Export δημιουργεί ένα πίνακα συμβόλων χρησιμοποιώντας το Clang που είναι το front-end του μεταγλωττιστή LLVM. Παρακάτω βρίσκονται οι οδηγίες εγκατάστασης του LLVM σε Windows 10 με χρήση Visual Studio. Το στάδιο της μεταγλώττισης του LLVM ενδέχεται να διαρκέσει πολλές ώρες και να απαιτήσει έως και 85GB από τον σκληρό δίσκο.

Προαπαιτούμενα:

* [Git](https://git-scm.com/)
* [CMake](https://cmake.org/)

1. Δημιουργείστε ένα καινούργιο φάκελο στον οποίο θα εγκατασταθεί ο LLVM.
2. Χρησιμοποιώντας το τερματικό:
   1. Μεταβείτε σε αυτό τον φάκελο ( $ cd C:\... )
   2. Τρέξτε την εντολή:

$ git clone --config core.autocrlf=false <https://github.com/llvm/llvm-project.git>

* 1. Μεταβείτε στον φάκελο που έχει δημιουργηθεί ( $ cd llvm-project)
  2. Τρέξτε την εντολή:

cmake -DLLVM\_ENABLE\_PROJECTS=clang -G "Visual Studio 16 2019" -A x64 -Thost=x64 llvm

1. Ανοίξτε το llvm-project\build\LLVM.sln χρησιμοποιώντας την ίδια έκδοση του Visual Studio που χρησιμοποιήσατε στο cmake (π.χ. VS 2019).
2. Από το Visual Studio:
   1. Από την πάνω μπάρα επιλέξτε configuration Release & x64
   2. Δεξιά από το Solution Explorer ανοίξτε τον φάκελο CMakePredefinedTargets
   3. 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.
   4. Πατήστε πάλι με δεξί κλικ πάνω στο “ALL\_BUILD” και πατήστε build ( η μεταγλώττιση του LLVM μπορεί να πάρει 3-7 ώρες).
   5. Τέλος το μήνυμα “build success” θα σας ενημερώσει για την επιτυχής μεταγλώττιση.

### 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://vcpkg.io/en/index.html))

1. Εγκατάσταση του jsoncpp

* Εάν το vcpkg είναι εγκατεστημένο στον υπολογιστή, η εγκατάσταση γίνεται μέσο 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).

1. Χρησιμοποιώντας το τερματικό, τρέξτε την παρακάτω εντολή στην περιοχή που θέλετε να εγκαταστήσετε το σύστημα:

$ git clone <https://github.com/PhivPap/Code-Smell-Detector>

1. Χρησιμοποιώντας την ίδια έκδοση του Visual Studio με την οποία μεταγλωττίσατε το LLVM δημιουργήστε ένα άδειο C++ project.
2. Με το “Solution Explorer” προσθέστε στο project όλα τα αρχεία κώδικα που βρίσκονται κάτω από τον φάκελο “\GraphGenerator\”.
3. Από τις ιδιότητες (properties) του project:
   * Στην κατηγορία C/C++ > Additional Include Directories προσθέστε τα path:
     1. Για τους φακέλους κάτω από το “\GraphGenerator\” που περιέχουν header files.
     2. “path to”\llvm\llvm\include
     3. “path to”\llvm\clang\include
     4. “path to”\llvm\build\include
     5. “path to”\llvm\build\tools\clang\include
     6. Για το include directory του jsoncpp ( εάν αυτό είναι εγκατεστημένο με το vcpkg τότε είναι: “path to”\vcpkg\installed\x64-windows\include )
   * Linker > General > Additional Library Directories. Add the path where the .lib files of llvm-project are located (\llvm project\build\Debug\lib)
   * Linker > Input > Additional Dependencies. Add the .lib files of llvm-project. Add the version.lib and jsoncpp.lib too.
   * ……

### Εγκατάσταση του Code Smell Detector

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

1. Εγκατάσταση του Node.js

* Εάν το vcpkg είναι εγκατεστημένο στον υπολογιστή, η εγκατάσταση γίνεται μέσο της εντολής: $ vcpkg install jsoncpp:x64-windows

1. Εγκατάσταση του Electron
2. …

## Χρήση

### 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.

#### 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 ένα αρχείο κειμένου το οποίο περιέχει ανά γραμμή διαδρομές (path). Οι διαδρομές αυτές καθορίζουν ποια αρχεία δεν πρέπει να υπολογισθούν για την δημιουργία του πίνακα συμβόλων. Για παράδειγμα εάν το αρχείο κειμένου περιέχει την διαδρομή: “C:/project/skip/”, κανένα περιεχόμενο των αρχείων που βρίσκονται μέσα στον φάκελο “skip” δεν θα καταλήξει 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, ένα αρχείο κειμένου το οποίο περιέχει ανά γραμμή namespaces. Κανένα περιεχόμενο αυτών των namespaces δεν θα καταλήξει 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 που και με ποιο όνομα θα αποθηκευτεί ο πίνακας συμβόλων.

##### 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 με λίστα από εντολές. Κάθε μία από αυτές υποδεικνύει τον τρόπο με τον οποίο κάθε τμήμα του project θα μεταγλωττιστεί. Μπορεί να δημιουργηθεί με το CMake[[10]](#footnote-10) προσθέτοντας στην εντολή την παράμετρο “-D CMAKE\_EXPORT\_COMPILE\_COMMANDS=ON” και χρησιμοποιώντας τα generators “Ninja” ή “Makefile Generators”. Αλλιώς, για Visual Studio projects, μπορεί να δημιουργηθεί χρησιμοποιώντας την open-source επέκταση “SourceTrail”.

To Compilation Database θα αποθηκευτεί σε ένα αρχείο με το 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 την συμβολοσειρά “--cmp-db” και ως δεύτερη παράμετρο την διαδρομή για τον φάκελο ο οποίος περιέχει το αρχείο “compile\_commands.json”.

##### Είσοδος: Project Directory

Η δεύτερη αυτή μέθοδος εισόδου στο Symbol Table Export, είναι χρήσιμη μόνο σε μικρά projects τα οποία έχουν όλες τις διαδρομές στις “#include” ντιρεκτίβες να είναι σχετικές ή απόλυτες διαδρομές. Συχνά αυτή η μέθοδος δεν δουλεύει επειδή περιβάλλοντα προγραμματισμού, όπως το Visual Studio, για λόγους ευκολίας, προσφέρουν την δυνατότητα στον προγραμματιστή να ορίσει επιπλέον φακέλους ως “#include” ντιρεκτίβες. Όμως, η πληροφορία αυτή δεν υπάρχει στα αρχεία πηγαίου κώδικα.

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.

#### Έξοδος

Στο τέλος της εκτέλεσης του Symbol Table Export θα τυπωθεί στο τερματικό μήνυμα το οποίο επισημαίνει την επιτυχία ή την αποτυχία μεταγλώττισης. Στην περίπτωση επιτυχίας στην διαδρομή που έχει ορίσει ο χρήστης στην 5ή παράμετρο θα υπάρχει ο πίνακας συμβόλων ο οποίος στην συνέχεια θα χρησιμοποιηθεί από το Code Smell Detector.

### Χρήση του Code Smell Detector

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

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



Εικόνα : Το αρχείο MainConfig.json

Εκεί, πρέπει να θέσει στο πεδίο “ST\_Path” την σχετική διαδρομή για το JSON αρχείο που έχει παραχθεί από το Symbol Table Export. Μπορεί να αλλάξει την τιμή του τρίτου πεδίου εάν θέλει να επιλέξει άλλο αρχείο δηλώσεων των ανιχνευτών δυσοσμίας, ή δομή του οποίου περιγράφεται στην ενότητα 3.3.3. Τέλος, στην τιμή του 4ού πεδίου, έχει την δυνατότητα να βάλει άλλη 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 “<insert startup command here>”.

## Επέκταση

# Συμπεράσματα

## Γνωστά προβλήματα

## Πιθανές βελτιώσεις

# 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] | D. B. Johnson, «Finding all the elementary circuits of a directed graph,» *SIAM Journal on Computing,* τόμ. IV, αρ. 1, pp. 79-80, 1975. |
| [4] | K. Busbee, «Branching Statements,» Rebus Press, [Ηλεκτρονικό]. Available: https://press.rebus.community/programmingfundamentals/chapter/branching-statements/. [Πρόσβαση 1 August 2021]. |

1. code smell: τα συμπτώματα είτε κακών σχεδιαστικών επιλογών είτε κακής υλοποίησης. [1] [↑](#footnote-ref-1)
2. Literal: σταθερές τιμές στο πρόγραμμα (πχ. true, 2, “str”). [↑](#footnote-ref-2)
3. Branching statement: μία εντολή η οποία μπορεί να αλλάξει την ροή του προγράμματος [4]. [↑](#footnote-ref-3)
4. Member expression: είναι στην C++ οι εκφράσεις τύπου M->x ή M.x, όπου το M είναι δομή ή τύπος ένωσης και το x είναι πεδίο. [↑](#footnote-ref-4)
5. Destructor στην C++ είναι … [↑](#footnote-ref-5)
6. Virtual…. [↑](#footnote-ref-6)
7. Protected access … [↑](#footnote-ref-7)
8. Chromium engine: --------------- [↑](#footnote-ref-8)
9. Singleton class: είναι μία κλάση που κατά την εκτέλεση του προγράμματος θα υπάρξει ένα μοναδικό στιγμιότυπο. [↑](#footnote-ref-9)
10. To CMake είναι ένα προγραμματιστικό εργαλείο που αυτοματοποιεί την διαδικασία μεταγλώττισης πηγαίου κώδικα που ενδέχεται να είναι πολύπλοκή διαδικασία με μεγάλα προγράμματα. [↑](#footnote-ref-10)