η

Πανεπιστήμιο Μακεδονίας

Τμήμα: Εφαρμοσμένης Πληροφορικής

Μάθημα: Ποιότητα Λογισμικού

DNAnalyzer

DNAnalyzer

Έγγραφο Διαχείρισης Ποιότητας

<Αρχοντής Κωστής – Χρήστος Χαχούδης>

YouTube Link (Click Me!)

Περιεχόμενα

1. Εισαγωγή

Σε αυτό το κεφάλαιο, παρέχουμε μια επισκόπηση του έργου λογισμικού που πρόκειται να αναλύσουμε, τη δομή του εγγράφου και τους στόχους ποιότητας.

Για να βοηθήσουμε στην κατανόηση του περιεχομένου της εργασίας, παρέχουμε επίσης ορισμούς όρων που χρησιμοποιούνται συχνά στο έγγραφο.

1.1. Σκοπός

Σκοπός της εργασίας είναι η μέτρηση, παρακολούθηση και αντιμετώπιση του τεχνικού χρέους του DNAnalyzer, ενός εργαλείου ανοιχτού κώδικα για την ανάλυση και ερμηνεία αλληλουχιών DNA. Το τεχνικό χρέος θα μετρηθεί και θα παρακολουθηθεί με τη χρήση του MetricsCalculator και του DesigniteJava, τα οποία είναι εργαλεία για την ανάλυση της ποιότητας κώδικα Java.

Το τεχνικό χρέος θα αποπληρωθεί μέσω της εφαρμογής Design Patterns και της αναδόμησης του κώδικα.

1.2. Περιγραφή Συστήματος

Το DNAnalyzer είναι ένα εργαλείο γραμμένο σε Java για την ανάλυση και ερμηνεία αλληλουχιών DNA. Διαθέτει Command Line Interface (CLI) και Γραφικό Περιβάλλον (GUI) για να επιτρέπει σε γιατρούς να αλληλεπιδρούν εύκολα με το λογισμικό και να εντοπίζουν πιθανές γενετικές μεταλλάξεις που σχετίζονται με ασθένειες.

Ορισμένα βασικά χαρακτηριστικά του DNAnalyzer περιλαμβάνουν:

- Δυνατότητα αναγνώρισης κωδικονίων έναρξης και παύσης
- Δυνατότητα πρόβλεψης περιοχών υψηλής κάλυψης και εντοπισμού αλληλουχιών, υποκινητών και σχετικών ρυθμιστικών αλληλουχιών
- Σειρά εργαλείων, όπως επεξεργαστής ακολουθιών DNA.
- Δυνατότητα χειρισμού αρχείων FASTA

1.3. Στόχοι Ποιότητας

1.3.1. Χρήση Αρχών Σχεδίασης

Ακολουθώντας τις καθιερωμένες αρχές σχεδιασμού, θα προσπαθήσουμε να βελτιώσουμε ορισμένα χαρακτηριστικά και μετρικές του λογισμικού. Για παράδειγμα, αν αποφασίσουμε να εφαρμόσουμε την αρχή Open-Closed, θα επικεντρωθούμε στη βελτίωση της επεκτασιμότητας του συστήματος. Ακολουθώντας αυτές τις αρχές, στόχος μας είναι να βελτιώσουμε την συνολική ποιότητα και συντηρησιμότητα της βάσης κώδικα

1.3.2. Χρήση Προτύπων Σχεδίασης

Με τη χρήση δημοφιλών προτύπων σχεδίασης, στοχεύουμε στη βελτίωση της συνοχής του συστήματος. Ένας συνεκτικός σχεδιασμός θα διευκολύνει την κατανόηση και συντήρηση του codebase.

1.3.3. Αναδόμηση Κώδικα

Για να βελτιώσουμε την αναγνωσιμότητα, συντηρησιμότητα και αποτελεσματικότητα του συστήματος, θα εφαρμόσουμε τεχνικές αναδόμησης κώδικα, όπως η μεταφορά

λειτουργικότητας μεταξύ των κλάσεων και η αντικατάσταση μεγάλων τμημάτων κώδικα με ιεραρχίες. Αυτές οι προσπάθειες αναδόμησης αναμένεται να επηρεάσουν τις μετρικές που σχετίζονται με την επεκτασιμότητα, σύζευξη και συντηρησιμότητα.

1.4. Ορισμοί

1.4.1. Ορισμοί Τεχνικού Χρέους

- **Τεχνικό Χρέος** (Technical Debt): Το κόστος της πρόσθετης προσπάθειας που απαιτείται για τη συντήρηση ή βελτίωση ενός συστήματος λογισμικού λόγω της επιλογής μιας εύκολης λύσης ή κακών σχεδιαστικών αποφάσεων που έγιναν στο παρελθόν.
- **Κεφάλαιο** (Principal): Η απόσταση μεταξύ της εσωτερικής ποιότητας του συστήματος και του βέλτιστου επιπέδου ποιότητας
- **Τόκος** (Interest): Η επιπλέον προσπάθεια που απαιτείται για τη διατήρηση ενός συστήματος με συσσωρευμένο τεχνικό χρέος.
- **Σημείο Θραύσης** (Breaking Point): Το χρονικό σημείο στο οποίο οι συσσωρευμένοι τόκοι του τεχνικού χρέους θα φθάσουν το ποσό του αντίστοιχου κεφαλαίου.

1.4.2. Ορισμοί Μετρικών

- **DIT** (Depth of Inheritance Tree): Αυτή η μετρική μετρά το βάθος κληρονομικότητας μιας κλάσης στην ιεραρχία κλάσεων. Μια υψηλή τιμή DIT υποδηλώνει μια πολύπλοκη ιεραρχία κλάσεων, η οποία μπορεί να κάνει την κλάση πιο δύσκολη στην κατανόηση και συντήρηση.
- **NOCC** (Number Of Class Children): Αυτή η μετρική μετρά τον αριθμό των κλάσεων-παιδιών που έχει μια κλάση. Μια υψηλή τιμή NOCC θα μπορούσε να υποδεικνύει μια κλάση που επαναχρησιμοποιείται σε πολλά σημεία, γεγονός που θα μπορούσε να καταστήσει πιο δύσκολη τη τροποποίηση και συντήρησή της.
- **CBO** (Coupling Between Objects): Αυτή η μετρική μετρά τον αριθμό άλλων κλάσεων από τις οποίες εξαρτάται μια κλάση. Μια υψηλή τιμή CBO θα μπορούσε να υποδεικνύει μια κλάση που εξαρτάται σε μεγάλο βαθμό από άλλες κλάσεις, γεγονός που θα μπορούσε να καταστήσει το έργο πιο εύθραυστο και πιο δύσκολο στη συντήρηση.
- **LCOM** (Lack of Cohesion in Methods): Αυτή η μετρική μετρά τον βαθμό στον οποίο οι μέθοδοι μιας κλάσης σχετίζονται μεταξύ τους. Μια υψηλή τιμή LCOM υποδεικνύει ότι οι μέθοδοι σε μια κλάση μπορεί να μην έχουν ισχυρή συνεκτική σχέση, γεγονός που μπορεί να καταστήσει την κλάση πιο δύσκολη στην κατανόηση και συντήρηση.
- **WMC** (Weighted Methods per Class): Αυτή η μετρική μετρά την πολυπλοκότητα μιας κλάσης, λαμβάνοντας υπόψη τον αριθμό και την πολυπλοκότητα των μεθόδων της κλάσης. Μια υψηλή τιμή WMC μπορεί να υποδηλώνει μια πολύπλοκη κλάση που μπορεί να είναι δύσκολο να κατανοηθεί και να συντηρηθεί.
- **NOM** (Number Of Methods): Αυτή η μετρική μετρά τον αριθμό των μεθόδων σε μια κλάση. Μια υψηλή τιμή NOM μπορεί να υποδεικνύει μια κλάση με μεγάλο αριθμό αρμοδιοτήτων, γεγονός που μπορεί να την καταστήσει πιο δύσκολη στην κατανόηση και συντήρησή της.
- **SIZE1** (Lines Of Code/LOC): Αυτή η μετρική μετράει το μέγεθος μιας κλάσης ως προς τον αριθμό των γραμμών κώδικα. Μια υψηλή τιμή SIZE1 θα μπορούσε να υποδηλώνει μια μεγάλη κλάση που θα μπορούσε να είναι δύσκολο να κατανοηθεί και να συντηρηθεί.

1.4.3. Ορισμοί Αρχών Σχεδίασης

- **Single Responsibility Principle** (SRP/Αρχή Μοναδικής Αρμοδιοτήτας): Η αρχή αυτή δηλώνει ότι μια κλάση πρέπει να έχει έναν και μόνο έναν λόγο για να αλλάξει, δηλαδή πρέπει να έχει μόνο μια ευθύνη/αρμοδιότητα. Αυτό συμβάλλει στη διατήρηση της απλής σχεδίασης του συστήματος και διευκολύνει στην κατανόηση, testing και συντήρησή του.
- **Open Closed Principle** (OCP/Αρχή Ανοιχτής-Κλειστής Σχεδίασης): Αυτή η αρχή δηλώνει ότι μια κλάση θα πρέπει να είναι ανοικτή για επέκταση αλλά κλειστή για τροποποίηση, αυτό σημαίνει ότι η νέα λειτουργικότητα θα πρέπει να προστίθεται μέσω υποκλάσεων και υλοποίησης νέων μεθόδων αντί της τροποποίησης των υπαρχουσών. Αυτό συμβάλλει στην προστασία του υπάρχοντος κώδικα, καθιστώντας το σύστημα πιο σταθερό και εύκολο στη συντήρηση.

1.4.4. Άλλοι Ορισμοί

- **Code Smell** (Οσμή Κώδικα): είναι ένα χαρακτηριστικό του πηγαίου κώδικα που μπορεί να υποδεικνύει κάποιο πρόβλημα. Δεν αποτελεί απαραίτητα πρόβλημα από μόνο του, αλλά μάλλον σύμπτωμα ενός ή περισσότερων προβλημάτων που θα μπορούσαν να επηρεάσουν αρνητικά την ποιότητα και συντηρησιμότητα του κώδικα.
- **Remediation Time** (Χρόνος Αποκατάστασης): Ο χρόνος που απαιτείται για τη διόρθωση ή αναδιαμόρφωση του κώδικα ώστε να εξαλειφθεί κάποιο code smell. Αυτό μπορεί να εξαρτάται από διάφορους παράγοντες, όπως η πολυπλοκότητα του code smell, το μέγεθος της βάσης κώδικα και το επίπεδο δεξιοτήτων των προγραμματιστών που εργάζονται πάνω στο πρόβλημα. Σε ορισμένες περιπτώσεις, ο χρόνος αποκατάστασης μπορεί να είναι σχετικά σύντομος, όπως μερικές ώρες, αλλά σε άλλες περιπτώσεις μπορεί να είναι μεγαλύτερος, όπως μερικές ημέρες ή ακόμη και εβδομάδες.
- **Code Duplication** (Επανάληψη Κώδικα): Η επανάληψη κώδικα σε πολλά σημεία μέσα σε ένα πρόγραμμα ή σύστημα. Αυτό μπορεί να οδηγήσει σε αυξημένο κόστος συντήρησης, καθώς οι αλλαγές στον επαναλαμβανόμενο κώδικα πρέπει να γίνουν σε πολλές θέσεις.
- **Code Reuse** (Επαναχρησιμοποίηση Κώδικα): Η πρακτική της χρήσης υπάρχοντος κώδικα σε μια νέα ή διαφορετική εφαρμογή ή σε ένα νέο feature, αντί της συγγραφής νέου κώδικα από το μηδέν. Αυτό μπορεί να οδηγήσει σε μείωση του χρόνου ανάπτυξης και σε αυξημένη συντηρησιμότητα και αξιοπιστία του κώδικα.

2. Μέτρηση Τεχνικού Χρέους

Σε αυτή την ενότητα, θα συζητήσουμε τις μεθόδους και τα εργαλεία που χρησιμοποιήθηκαν για τη μέτρηση του τεχνικού χρέους του έργου. Θα δούμε επίσης μια επισκόπηση της διαδικασίας μέτρησης του τεχνικού χρέους, συμπεριλαμβανομένου του υπολογισμού του Principal, Interest και Breaking Point. Οι πληροφορίες αυτές θα χρησιμοποιηθούν για τον εντοπισμό περιοχών στο έργο που μπορεί να χρειάζονται βελτίωση προκειμένου να αποπληρωθεί αποτελεσματικά το τεχνικό χρέος.

2.1. Προεργασία μέτρησης Τεχνικού Χρέους

Προκειμένου να μετρηθεί και να παρακολουθηθεί με ακρίβεια το τεχνικό χρέος του DNAnalyzer, είναι απαραίτητο να υπολογιστούν το κεφάλαιο και οι τόκοι σε νομισματικές μονάδες. Για να γίνει αυτό, πρέπει να προσδιορίσουμε το μέσο ημερομίσθιο ενός Java

Developer, καθώς αυτό θα χρησιμοποιηθεί για τον υπολογισμό του κόστους του τεχνικού χρέους σε χρόνο και χρηματικές μονάδες.

2.1.1. Υπολογισμός του μέσου ημερήσιου μισθού Java Developer

Για τον προσδιορισμό του μέσου ημερήσιου μισθού ενός Java Developer, συγκεντρώσαμε στοιχεία για τους μέσους ετήσιους μισθούς προγραμματιστών Java σε εννέα ευρωπαϊκές χώρες από την Payscale. Από αυτά τα δεδομένα υπολογίσαμε τον μέσο ετήσιο μισθό, τον μέσο μηνιαίο μισθό και τον μέσο ημερήσιο μισθό και τον μέσο μισθό ανά ώρα. Αυτοί οι υπολογισμοί θα χρησιμεύσουν ως μια προσέγγιση του μέσου ημερίσιου μισθού για τους σκοπούς της εργασίας και θα χρησιμοποιηθούν για τον υπολογισμό του κόστους του τεχνικού χρέους σε χρηματικές μονάδες.

Χώρα	Ετήσιο Εισόδημα Java Developer
Βέλγιο	80,700 €
Ηνωμένο Βασίλειο	66,000 €
Ισπανία	56,000 €
Ελβετία	110,000 €
Ιταλία	60,000 €
Γερμανία	80,500 €
Γαλλία	69,800 €
Ελλάδα	44,600 €
Βουλγαρία	19,700 €

Πίνακας 1: Μέσοι Μισθοί Java Developer σε χώρες της Ευρώπης

Μέσο Ετήσιο Εισόδημα	65,256 €
Μέσος Μισθός ανά μήνα	5,437.96 €
Μέσο Ημερομίσθιο	181.27 €
Μέσο Ωρομίσθιο	22.66€

Πίνακας 2: Μέσο Εισόδημα προγραμματιστών Java

Με βάση αυτούς τους υπολογισμούς, ο μέσος ημερήσιος μισθός ενός προγραμματιστή Java είναι 181,27 € ανα ημέρα και 22,66€ ανα ώρα. Αυτες είναι και οι τιμές που θα χρησιμοποιήσουμε για του υπολογισμούς μας.

2.2. Μέτρηση Τεχνικού Χρέους

Σε αυτή την ενότητα, θα δούμε τις μεθόδους και τα εργαλεία που χρησιμοποιήσαμε για τη μέτρηση του τεχνικού χρέους. Το κύριο εργαλείο που χρησιμοποιήθηκε ήταν το DesigniteJava, ένα εργαλείο ανοικτού κώδικα για την ανάλυση κώδικα Java και τον εντοπισμό code smells. Τα code smells είναι μοτίβα στον κώδικα που υποδηλώνουν κάποιο πρόβλημα και μπορούν να χρησιμεύσουν ως δείκτες τεχνικού χρέους.

2.2.1. DesigniteJava

Το DesigniteJava χρησιμοποιήθηκε για τη σάρωση της βάσης κώδικα του DNAnalyzer και την ανίχνευση διαφόρων code smells. Εντοπίστηκαν οι ακόλουθοι τύποι code smells:

- **Magic Number:** Ένας «μαγικός αριθμός» είναι ένας αριθμός που χρησιμοποιείται στον κώδικα χωρίς καμία εξήγηση (συνήθως σε συνθήκες δομών επιλογής). Αυτό μπορεί να

κάνει τον κώδικα πιο δύσκολο στην κατανόηση και συντήρηση, καθώς δεν είναι σαφές για ποιο λόγο χρησιμοποιήθηκε ο αριθμός ή τι αντιπροσωπεύει.

- **Long Statement:** μια γραμμή κώδικα που είναι υπερβολικά πολύπλοκη ή δύσκολα κατανοητή. Αυτό μπορεί να κάνει τον κώδικα πιο δύσκολο στην ανάγνωση και συντήρηση, καθώς απαιτεί μεγαλύτερη προσπάθεια για την ανάλυση και ερμηνεία του.
- **Unnecessary Abstraction:** Η αφαίρεση είναι μια χρήσιμη τεχνική για την οργάνωση και την απλούστευση πολύπλοκων συστημάτων, μπορεί όμως να φτάσει σε περιττές ακρότητες. Η περιττή αφαίρεση καταστεί τον κώδικα πιο δύσκολο στην κατανόηση και συντήρηση, καθώς μπορεί να αποκρύψει την υποκείμενη λογική και να δυσχεράνει τη συνολική εικόνα.
- **Deficient Encapsulation:** Η ενθυλάκωση είναι μια τεχνική για την ομαδοποίηση δεδομένων και μεθόδων που λειτουργούν με αυτά τα δεδομένα μέσα σε μια ενιαία μονάδα (όπως μια κλάση). Η ανεπαρκής ενθυλάκωση συμβαίνει όταν τα δεδομένα δεν προστατεύονται σωστά ή όταν οι αρμοδιότητες μιας κλάσης δεν είναι καλά καθορισμένες. Αυτό μπορεί να οδηγήσει σε κώδικα που είναι πιο δύσκολος στην κατανόηση και συντήρηση, καθώς μπορεί να είναι επιρρεπής σε σφάλματα ή ασυνέπειες.
- **Unutilized Abstraction:** Η αφαίρεση μπορεί να είναι ένα ισχυρό εργαλείο για την οργάνωση και την απλούστευση πολύπλοκων συστημάτων, αλλά μπορεί επίσης να μην αξιοποιείται επαρκώς. Η αχρησιμοποίητη αφαίρεση εμφανίζεται όταν δημιουργούνται αφαιρέσεις αλλά δεν χρησιμοποιούνται αποτελεσματικά. Αυτό μπορεί να κάνει τον κώδικα πιο δύσκολο στην κατανόηση και συντήρηση, καθώς εισάγει περιττή πολυπλοκότητα και σύγχυση.
- **Broken Modularization:** το Modularization είναι η διαδικασία διαίρεσης ενός συστήματος σε μικρότερες, ανεξάρτητες ενότητες που μπορούν να αναπτυχθούν και να δοκιμαστούν ξεχωριστά. Αυτό το code smell προκαλείται όταν τα modules δεν είναι καλά καθορισμένα ή όταν δεν λειτουργούν όπως προβλέπεται. Αυτό μπορεί να καταστήσει τον κώδικα πιο δύσκολο να κατανοηθεί και να συντηρηθεί, καθώς οδηγεί σε συγκρούσεις ή εξαρτήσεις μεταξύ διαφορετικών τμημάτων του συστήματος.

2.2.2. Μέτρηση Κεφαλαίου (Principal)

Η μέτρηση του Κεφαλαίου γίνεται απο τον ακόλουθο τύπο:

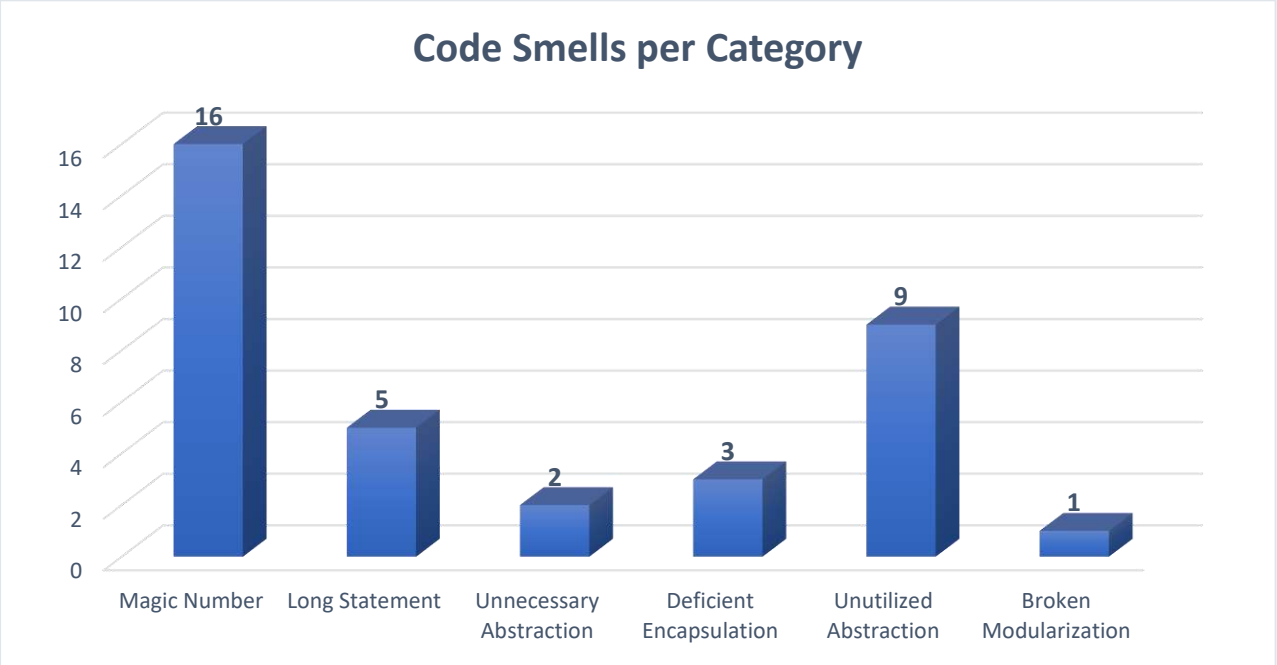
$$TechDebt_{Principal} = \sum_{i=0}^{codeSmells} \#instances \times remediationTime(i)$$

Τύπος 1: Υπολογισμός Κεφαλαίου Τεχνικού Χρέους

Κατά τη διάρκεια της ανάλυσης με το DesigniteJava, εντοπίστηκαν 32 code smells. Η κατανομή των code smells αυτών ανά κατηγορία παρουσιάζεται στον ακόλουθο πίνακα:

	Magic Number	Long Statement	Unnecessary Abstraction	Deficient Encapsulation	Unutilized Abstraction	Broken Modularization	TOTAL
# Code Smells	16	5	2	3	9	1	36

Πίνακας 3: Αριθμός Code Smell ανά Κατηγορία



Γράφημα 1: Αριθμός Code Smell ανά Κατηγορία

Για τον υπολογισμό του Remediation Time κάθε code smell θα χρησιμοποιήσουμε τον ακόλουθο τύπο:

$$RemediationTime_{minutes} = DifficultyFactor \times SizeFactor$$

Τύπος 2: Υπολογισμός Remediation Time

Όπου:

- Το $\boldsymbol{DifficultyFactor}$ είναι μια υποκειμενική βαθμολογία που αποδίδεται από εμάς σε μια κλίμακα από το 1 έως το 5, με το 1 να είναι η πιο εύκολη διόρθωση και το 5 η πιο δύσκολη.
- Το $\boldsymbol{SizeFactor}$ είναι επίσης μια υποκειμενική βαθμολογία που αποδίδεται από εμάς σε μια κλίμακα από το 1 έως το 5, με το 1 να είναι το μικρότερο σε μέγεθος ή πεδίο εφαρμογής και το 5 το μεγαλύτερο.

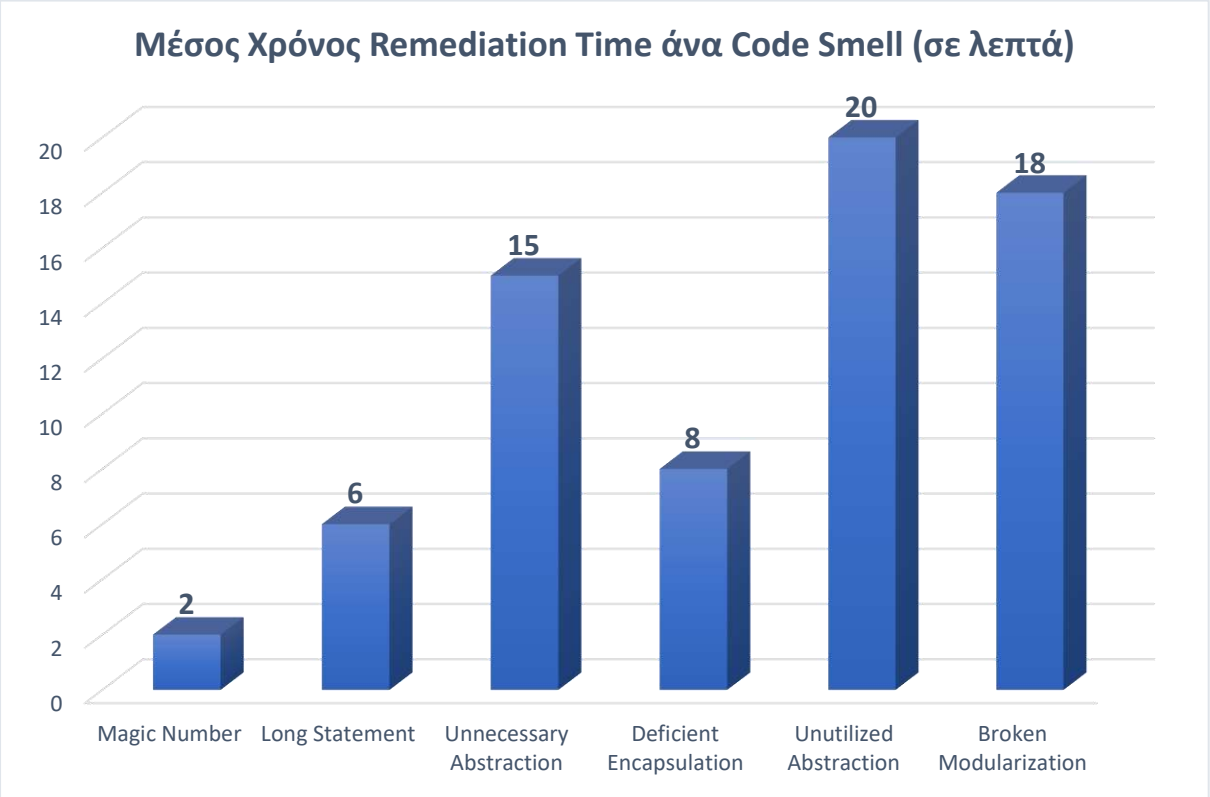
Χρησιμοποιώντας το $\boldsymbol{DifficultyFactor}$ και το $\boldsymbol{SizeFactor}$ στον υπολογισμό μας μπορούμε να έχουμε μια εκτίμηση του χρόνου που απαιτείται για την αντιμετώπιση κάθε code smell. Αυτό μας επιτρέπει να αποφύγουμε την ανάγκη να διορθώσουμε ένα code smell κάθε τύπου και στη συνέχεια να χρησιμοποιήσουμε τον μέσο χρόνο που απαιτείται για τη διόρθωση κάθε οσμής ως μέτρο χρόνου.

Με βάση αυτούς τους υπολογισμούς, προκύπτει ο ακόλουθος πίνακας με Remediation Times:

Code Smell	Difficulty	Size	Remediation Time (mins)
Magic Number	1	2	2
Long Statement	2	3	6
Unnecessary Abstraction	3	5	15

Deficient Encapsulation	2	4	8
Unutilized Abstraction	4	5	20
Broken Modularization	3	6	18

Πίνακας 4: Μέσος Χρόνος Remediation Time ανα Code Smell (σε λεπτά)



Γράφημα 2: Μέσος Χρόνος Remediation Time ανα Code Smell (σε λεπτά)

Χρησιμοποιώντας τον τύπο για τον υπολογισμό του κεφαλαίου (Τύπος 1) και τον τύπο για το Remediation Time, υπολογίζουμε:

$$Principal_{MagicNumber} = 16 \times 2 = 32$$

$$Principal_{LongStatement} = 5 \times 6 = 30$$

$$Principal_{UnnecessaryAbstraction} = 2 \times 15 = 30$$

$$Principal_{DeficientEncapsulation} = 3 \times 8 = 24$$

$$Principal_{UnutilizedAbstraction} = 9 \times 20 = 180$$

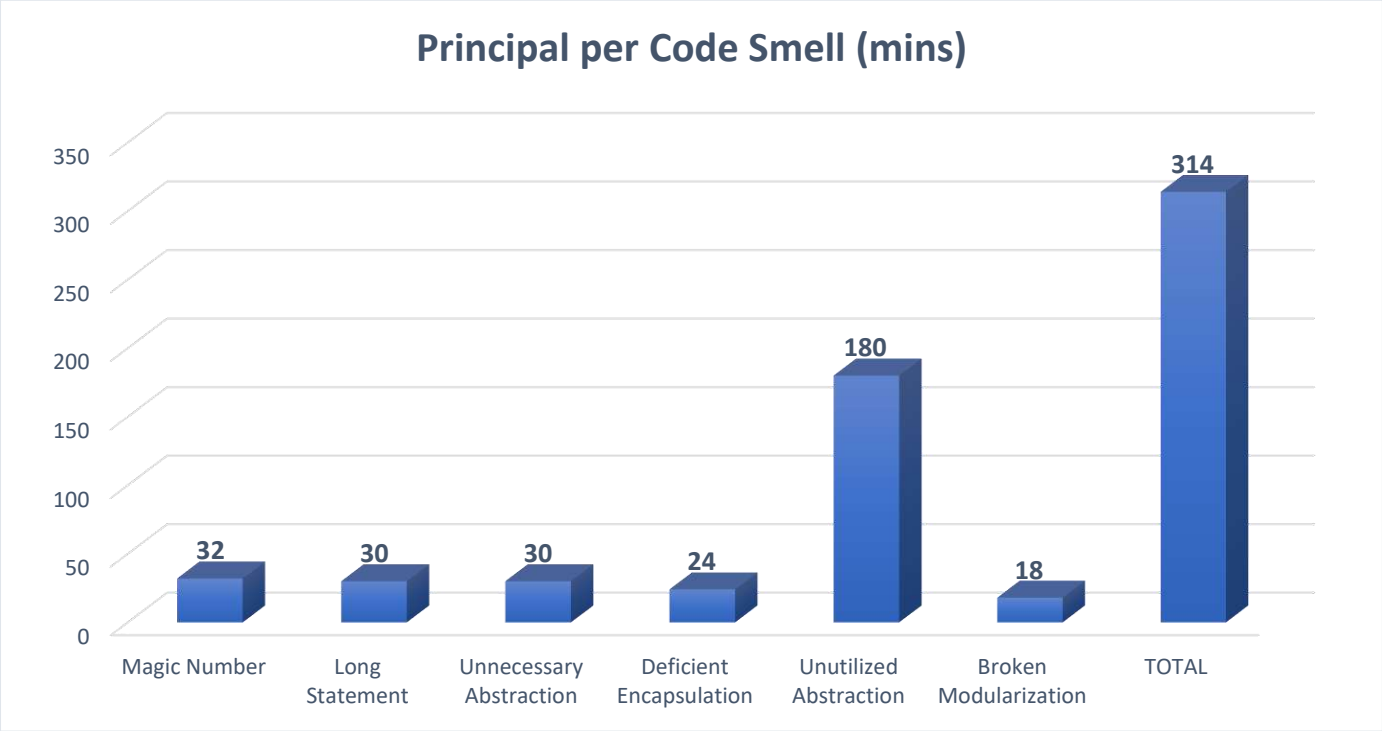
$$Principal_{BrokenModularization} = 1 \times 18 = 18$$

$$Principal_{AllCodeSmells} = Principal_{MagicNumber} + Principal_{LongStatement} + Principal_{UnnecessaryAbstraction} + Principal_{DeficientEncapsulation} + Principal_{BrokenModularization}$$

$$Principal_{AllCodeSmells} = 32 + 30 + 30 + 24 + 180 + 18 = 314\ \lambda\epsilon\pi\tau\acute{\alpha} \approx 6\ \acute{\omega}\rho\epsilon\varsigma$$

Με βάση τους παραπάνω υπολογισμούς, κατασκευάσαμε τον ακόλουθο πίνακα:

	Magic Number	Long Statement	Unnecessary Abstraction	Deficient Encapsulation	Unutilized Abstraction	Broken Modularization	TOTAL
Principle (mins)	32	30	30	24	180	18	314



Πίνακας 5: Principal ανά Κατηγορία Code Smell

Επομένως για τη διόρθωση όλων των code smells, εκτιμάται ότι θα χρειαστούν 314 λεπτά, δηλαδή περίπου 6 ώρες.

2.2.3 Κεφάλαιο σε Χρηματικές Μονάδες

Για να υπολογίσουμε το κόστος της αντιμετώπισης των code smells σε χρηματικές μονάδες, πρέπει να γνωρίζουμε το μέσο ωρομίσθιο ενός Java Developer. Όπως προσδιορίστηκε προηγουμένως (Κεφάλαιο 2.1.1), ο μέσος μισθός είναι 181,27€.

Μπορούμε να χρησιμοποιήσουμε αυτή την τιμή για να υπολογίσουμε αρχικά τον μέσο μισθό ανα ώρα και στην συνέχεια το συνολικό κόστος πολλαπλασιάζοντάς τον μισθό ανά ώρα με τον αριθμό των ωρών προσπάθειας που απαιτούνται για την ολοκλήρωση της αντιμετώπισης των code smells.

Υπολογισμός Μέσου Ωρομίσθιου

$$HourlySalary = DailyIncome \div 8$$

$$HourlySalary = 187.27 \div 8 = 23.41€$$

Υπολογισμός Κεφαλαίου σε Χρηματικές Μονάδες

$$Principal_{EUROS} = Principal_{HOURS} \times HourlySalary$$

$$Principal_{EUROS} = 6 \times 23.41 = 140.46€$$

2.2.3. Μέτρηση Τόκου (Interest)

Η μέτρηση του τόκου για μια συγκεκριμένη χρονική περίοδο (π.χ. ενός μήνα, ενός έτους κλπ.) γίνεται από τον παρακάτω τύπο:

$$Interest = Distance_{fromOptimumMaintainability} \times Period(x) \times LoadOfMaintenance_{History}$$

2.2.3.1. Απόσταση από το Βέλτιστο

Για βρούμε τι θεωρούμε βέλτιστο, επιλέξαμε να χρησιμοποιήσουμε την κλάση `Properties.java` ως σημείο αναφοράς.

Πιο συγκεκριμένα, για να μετρήσουμε την ομοιότητα μεταξύ των κλάσεων, χρησιμοποιήσαμε τη μετρική SIZE1 (LOC). Αυτή η μετρική αντιπροσωπεύει τον αριθμό των γραμμών κώδικα σε μια κλάση και χρησιμοποιείται ευρέως ως μέτρο πολυπλοκότητας μιας κλάσης.

Υπολογίσαμε την ομοιότητα κάθε κλάσης με την κλάση αναφοράς χρησιμοποιώντας τον ακόλουθο τύπο:

$$Similarity = \left| \left(loc_{class} - loc_{referenceClass} \right) \div loc_{referenceClass} \right|$$

Αυτός ο τύπος επιστρέφει μια τιμή μεταξύ 0 και 1, με το 1 να υποδεικνύει ότι η κλάση έχει τον ίδιο αριθμό γραμμών κώδικα με την κλάση αναφοράς και το 0 να υποδεικνύει ότι η κλάση έχει εντελώς διαφορετικό αριθμό γραμμών κώδικα. Ταξινομώντας τον πίνακα των μετρικών σε επίπεδο κλάσης με βάση την τιμή ομοιότητας, μπορούμε να προσδιορίσουμε ποιες κλάσεις είναι πιο παρόμοιες με την κλάση αναφοράς.

Χρησιμοποιώντας αυτό τον τύπο, δημιουργήσαμε τον ακόλουθο πίνακα με τις κλάσεις ταξινομημένες με βάση την ομοιότητα:

Name	SIZE1	SIMILARITY (%)
Properties.java	61	100%
DNAnalyzerGUIFXMLController.java	49	80%
CmdArgs.java	44	72%
ProteinFinder.java	40	66%
ReadingFrames.java	33	54%
ProteinAnalysis.java	25	41%
BasePairCounter.java	24	39%
Main.java	21	34%
DNAnalyzerGUI.java	18	30%
ProteinFinderst.java	14	23%
Utils.java	6	10%
AminoAcidFactory.java	5	8%
CodonDataConstants.java	5	8%
MainTest.java	3	5%
CodonDataUtils.java	3	5%
AminoAcid.java	0	0%
DNATools.java	0	0%
CodonFrame.java	0	0%
DNAAnalysis.java	0	0%
Traits.java	0	0%

Πίνακας 6: Ομοιότητα μεταξύ των Κλάσεων

Στην περίπτωσή μας, η κλάση `DNAnalyzerGUIFXMLController.java` ήταν η πιο όμοια κλάση με τιμή ομοιότητας 80% και η λιγότερο όμοια κλάση ήταν η `Traits.java` με τιμή ομοιότητας 0%.

Επιλέξαμε την κλάση `Properties.java`, που είναι το σημείο αναφοράς μας, και τις επόμενες πέντε πιο παρόμοιες κλάσεις. Στην συνέχεια πήραμε την καλύτερη τιμή κάθε μετρικής μεταξύ αυτών των έξι κλάσεων και την αντιμετωπίσαμε ως την τιμή που θα είχε η βέλτιστη κλάση μας (η κλαση αυτή δεν υπάρχει στην πραγματική βάση κώδικα του έργου).

	DNAnalyzer GUIFXML Controller	CmdArgs	ProteinFinder	ReadingFrames	ProteinAnalysis	Properties	Optimal
DIT	0	0	0	0	0	0.000	0.000
CBO	2	4	2	1	1	2.000	1.000
LCOM	10	1	1	1	1	28.000	1.000
WMC	1.166666627	2	2	2	1.333333373	1.000	1.000
NOM	5	2	2	2	2	8.000	2.000
SIZE1	49	44	40	33	25	61.000	25.000

Πίνακας 7: Κλάσεις υπό έλεγχο και Βέλτιστη Κλάση (Με πράσινο οι καλύτερες τιμές καθε μετρικής)

Από αυτό τον πίνακα, υπολογίσαμε την απόσταση απο το βέλτιστο και κατασκευάσαμε τον ακόλουθο:

Properties	OptimalClass	Distance From Optimal
0.000	0.000	0.000
2.000	1.000	1.000
28.000	1.000	27.000
1.000	1.000	0.000
8.000	2.000	6.000
61.000	25.000	36.000

Πίνακας 8: Απόσταση από το Βέλτιστο

2.2.3.2. Υπολογισμός Τόκου ανά Γραμμή Κώδικα

Για να βρούμε τον τόκο ανά γραμμή κώδικα, ακολουθήσαμε τον εξής τύπο:

$$InterestForMetric_{perLOC} = distanceFromOptimal \div MetricValue_{OptimalClass}$$

Στη συνέχεια πήραμε το μέσο όρο των τιμών που προέκυψαν, αυτός ο μέσος όρος είναι η τιμή του Interest per LOC:

Metric	OptimalClass	Distance From Optimal	Interest per LOC
DIT	0	0.000	0
CBO	1	1.000	1
LCOM	1	27.000	27
WMC	1	0.000	0
NOM	2	6.000	3
SIZE1	25	36.000	1.44

AVERAGE	6.49

Table 9: Τόκος ανά Γραμμή Κώδικα

Επομένως, ο τόκος ανά γραμμή κώδικα είναι 6.49 γραμμές.

2.2.3.3. Υπολογισμός ανά Μετάβαση σε Νέα Έκδοση

Για να εκτιμήσουμε τον τόκο που συσσωρεύεται με κάθε έκδοση, προσδιορίσαμε τον μέσο αριθμό γραμμών κώδικα που προστίθενται ανά version.

Στον παρακάτω πίνακα, παρουσιάζουμε τις εκδόσεις του έργου και τον συνολικό αριθμό γραμμών κώδικα για κάθε έκδοση (LOC/SIZE1). Στην τελευταία γραμμή του πίνακα εμφανίζεται ο μέσος όρος των διαφορών σε LOC από έκδοση σε έκδοση, αυτός ο μέσος όρος θα χρησιμοποιηθεί ως παράγοντας για τον υπολογισμό του τόκου.

VERSION	LOC
1.0.0	
1.2.0	61
2.0.0	53
2.1.1	61
AVERAGE	15.25

Όπως φαίνεται από τον πίνακα, παρατηρούμε ότι κατά μέσο όρο, 15.25 γραμμές κώδικα προστίθενται με κάθε νέα έκδοση.

2.2.3.4. Τόκος σε Ώρες

Για να προσδιορίσουμε το ποσό των τόκων που θα συγκεντρωθούν με την πάροδο του χρόνου, πρέπει να γνωρίζουμε τον μέσο αριθμό γραμμών κώδικα που μπορεί να γράψει ένας προγραμματιστής ανά ώρα. Για τους σκοπούς της εργασίας θεωρούμε οτι ο μέσος προγραμματιστής γράφει 25 γραμμές ανά ημέρα.

Επομένως έχουμε:

$$Interest_{HOURS} = \frac{LOC_{addedPerVersion}}{25}$$

$$Interest_{HOURS} = \frac{15.25}{25} = 0.61\ Ώρες$$

Επομένως, ο τόκος του τεχνικού χρέους σε ώρες ειναι 0,61 ώρες (περίπου 37 λεπτά).

2.2.3.5. Τόκος σε Χρηματικές Μονάδες

Για να μπορέσουμε να υπολογίσουμε τον τόκο σε χρηματικές μονάδες, και πιο συγκεκριμένα σε Ευρώ (€), θα πρέπει να γνωρίζουμε τον μέσο μισθό ενός Java Developer ανα ώρα. Ώπως είδαμε προηγουμένος ο μέσος μισθός ενός προγραμματιστή Java είναι 22.66€.

Επομένως έχουμε:

$$CostPerLine_{EUROS} = Interest_{HOURS} \times DevWage_{perDay}$$

$$CostPerLine_{EUROS} = 0.61 \times 22.66 = 13{,}82€$$

$$Interest_{EUROS} = AverageInterest_{perLOC} * CostPerLine_{EUROS} = 6.49 \times 13.82 = 89.68€$$

Επομένως ο τόκος του τεχνικού χρέους σε Ευρώ είναι 89,68€.

2.3. Σημείο Θραύσης (Breaking Point)

Έχοντας υπολογίσει τις τιμές του κεφαλαίου καθώς και του τόκου που έχει συσσωρευθεί σε μία έκδοση, μπορούμε πλέον να υπολογίσουμε με ευκολία τον αριθμό των ετών στον οποίο θα φτάσουμε το Σημείο Θράυσης. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε τον ακόλουθο τύπο:

$$BreakingPoint_{Years} = \frac{Principal_{EUROS}}{Interest_{EUROS}}$$

Επομένως έχουμε:

$$BreakingPoint_{Years} = \frac{140,46€}{89,68€} = 1.5\ έτη$$

Μέ αυτούς τους υπολογισμούς έχουμε ότι θα φτάσουμε στο σημείο θράυσης μετα από 1,5 έτη.

3. Παρακολούθηση Ποιότητας του Έργου

Σε αυτό το κεφάλαιο, παρέχει μια επισκόπηση της εξέλιξης διαφόρων μετρικών ποιότητας μεταξύ διαφορετικών εκδόσεων του υπο μελέτη λογισμικού. Αρχικά, θα παρουσιάσουμε την εξέλιξη της ποιότητας χρησιμοποιώντας διαγράμματα μετρικών. Στη συνέχεια θα παρουσιάσουμε πίνακες που συγκρίνουν τον κώδικα σε νεότερες εκδόσεις με τον υπάρχοντα κώδικα σε κάθε δεδομένη χρονική στιγμή. Τέλος θα παρέχεται μία αξιολόγηση της συνολικής ποιότητας του έργου καθώς και προτάσεις για βελτίωση.

3.1. Διαγράμματα

Συνολικά, τα δεδομένα δείχνουν ότι το λογισμικό έχει υποστεί κάποιες αλλαγές στην πολυπλοκότητα μεταξύ των εκδόσεων 1.0.0 και 2.0.0, με ορισμένες μετρικές να μειώνονται και άλλες να αυξάνονται. Στην έκδοση 2.1.1, υπήρξε μια μικρή αντιστροφή αυτών των τάσεων.

Για να έχουμε μία καλύτερη εικόνα του φαινομένου αυτού θα παρουσιάσουμε διαγράμματα που δείχνουν την εξέλιξη κάθε μετρικής με την πάροδο του χρόνου και θα σχολιάσουμε τις αλλαγές που βλέπουμε.

3.1.1. Μετρικές

- **DIT (Depth of Inheritance)**

  Η μετρική **DIT** παρέμεινε στο 0 για όλες τις εκδόσεις, κάτι που μπορεί να είναι καλό, καθώς υποδηλώνει μια πιο επίπεδη ιεραρχία κληρονομικότητας και ενδεχομένως απλούστερο σχεδιασμό κλάσεων.

  Ωστόσο, είναι επίσης πιθανό ότι το έργο δεν χρησιμοποιεί αποτελεσματικά την κληρονομικότητα, γεγονός που μπορεί να οδηγήσει σε έλλειψη reuseability του κώδικα και ενδεχομένως σε αυξημένη πολυπλοκότητα σχεδιασμού.

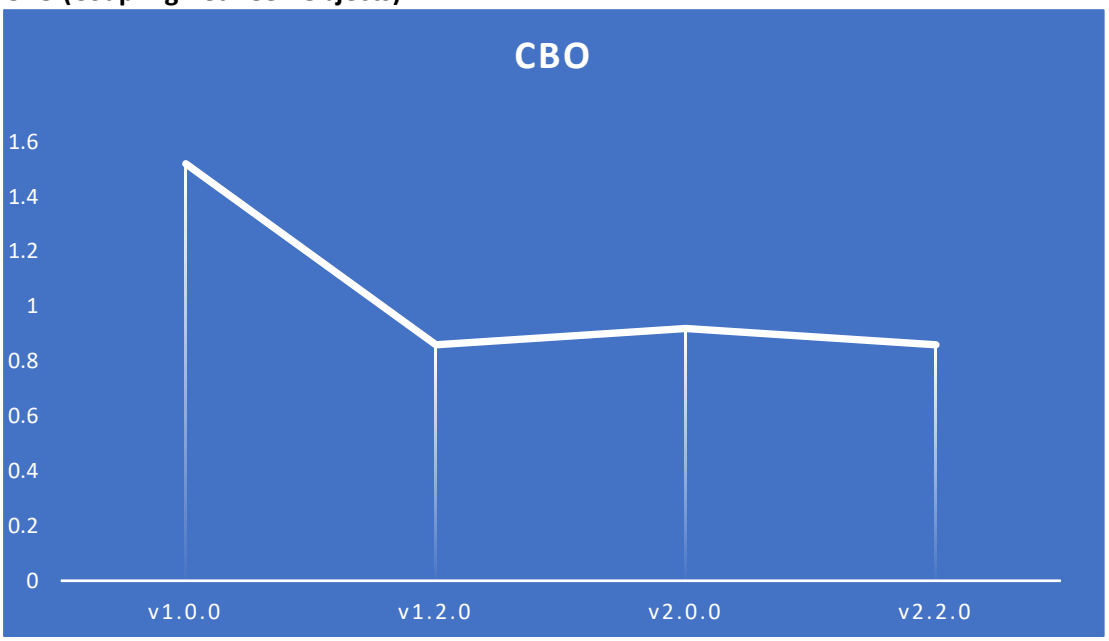
- **NOCC (Number Of Class Children)**

Η μετρική NOCC παρέμεινε και αυτή στο 0 για όλες τις εκδόσεις, γεγονός που μπορεί να υποδηλώνει ένα σχετικά χαμηλό επίπεδο επαναχρησιμοποίησης κώδικα και ενδεχομένως έναν απλούστερο σχεδιασμό κλάσεων.

Ενώ η έλλειψη επαναχρησιμοποίησης κώδικα (code reuse) μπορεί να είναι επωφελής από την άποψη της απλότητας και της ευκολίας κατανόησης, μπορεί επίσης να οδηγήσει σε αυξημένη επανάληψη κώδικα (code duplication) και υψηλότερο κόστος συντήρησης μακροπρόθεσμα.

Για παράδειγμα, αν προσθέταμε ένα νέο feature στο DNAnalyzer που απαιτούσε παρόμοια λειτουργικότητα με μια υπάρχουσα κλάση, θα έπρεπε να γράψουμε τον κώδικα από την αρχή αντί να είμαστε σε θέση να επαναχρησιμοποιήσουμε και να επεκτείνουμε τον υπάρχοντα κώδικα. Κάτι τέτοιο θα παραβίαζε την Αρχή της Ανοιχτής-Κλειστής Σχεδίασης. Σε αυτή την περίπτωση, ίσως αξίζει να εξετάσουμε το ενδεχόμενο αναδιαμόρφωσης του υπάρχοντος κώδικα για να επιτρέψουμε μεγαλύτερη επαναχρησιμοποίηση του κώδικα και να βελτιώσουμε τη συντηρησιμότητα του έργου.

- **CBO (Coupling Between Objects)**



Μια άλλη μετρική που πρέπει να προσέξουμε είναι η μετρική CBO, η οποία ξεκίνησε με μια αρκετά υψηλή τιμή, μειώθηκε στην έκδοση 1.2.0, στη συνέχεια αυξήθηκε ξανά στην εκδόση 2.0.0 και ξαναέπεσε στην έκδοση 2.1.1.

Πιο συγκεκριμένα:

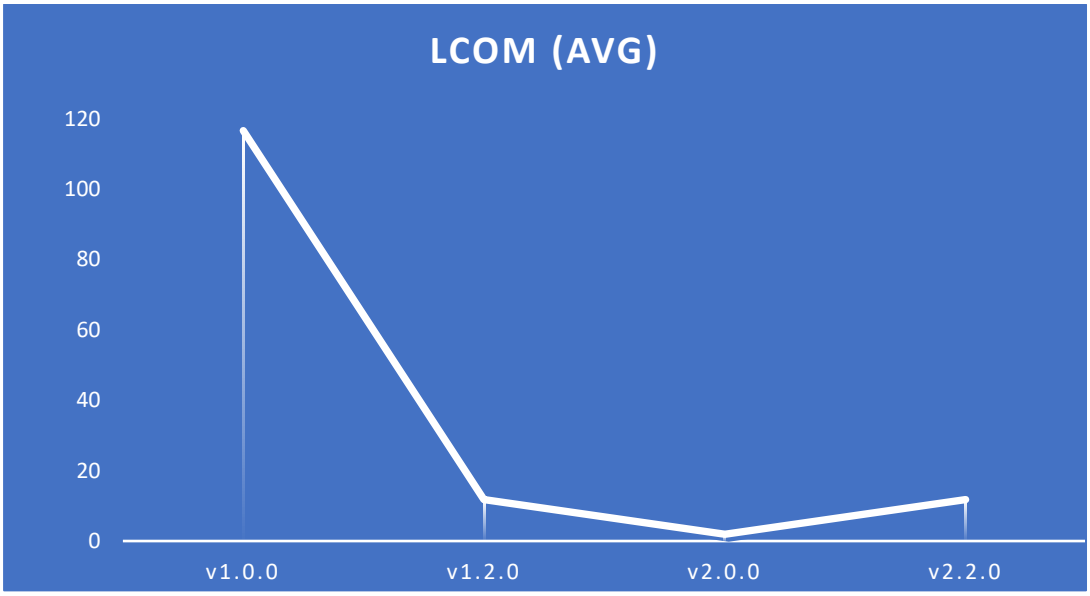
- Η CBO, στην έκδοση 1.0.0, ήταν 1.52, η οποία είναι σχετικά υψηλή και υποδηλώνει ένα υψηλό επίπεδο αλληλεξάρτησης μεταξύ των κλάσεων του έργου.
- Στην έκδοση 1.2.0, υπάρχει μια σημαντική πτώση στην τιμή της CBO στο 0,86, υποδεικνύοντας απλοποίηση των αλληλεξαρτήσεων μεταξύ των κλάσεων.
- Στην έκδοση 2.0.0, ο CBO αυξήθηκε και πάλι σε 0,9, πριν μειωθεί ελαφρώς στην έκδοση 2.1.1, ξανά σε 0,86.

Επομένως, οι καλύτερες εκδόσεις από άποψη CBO θα είναι οι εκδόσεις 1.2.0 και 2.1.1.

Οι υψηλές τιμές CBO μπορούν να υποδηλώνουν υψηλό βαθμό αλληλεξάρτησης μεταξύ των κλάσεων, γεγονός που μπορεί να καταστήσει το λογισμικό πιο δύσκολο στη συντήρηση και την τροποποίηση, καθώς οι αλλαγές σε μια κλάση μπορεί να έχουν αντίκτυπο σε άλλες κλάσεις. Για παράδειγμα, αν προσθέταμε ένα νέο feature που απαιτούσε την τροποποίηση πολλών κλάσεων, αυτό θα παραβίαζε την Single Responsibility Principle, καθώς θα σήμαινε ότι αυτές οι κλάσεις έχουν περισσότερους από έναν λόγους να αλλάξουν. Αυτό θα μπορούσε να οδηγήσει σε υψηλότερο κίνδυνο εισαγωγής σφαλμάτων και να καταστήσει πιο δύσκολη την κατανόηση και συντήρηση του κώδικα.

Αξίζει λοιπόν να εξεταστεί γιατί οι τιμές CBO άλλαξαν με αυτόν τον τρόπο και αν υπάρχουν ευκαιρίες μείωσης της σύζευξης στον κώδικα.

- **LCOM (Lack of Cohesion in Methods)**



Η μετρική που ξεχωρίζει είναι η LCOM, η οποία μειώθηκε σημαντικά μεταξύ των εκδόσεων 1.0.0 και 1.2.0, συνέχισε να έχει πτωτική τάση μεταξύ των εκδόσεων 1.2.0 και 2.0.0, αλλά αυξήθηκε ξανά στην έκδοση 2.1.1 .

Πιο συγκεκριμένα:

- Στην έκδοση 1.0.0, η LCOM ήταν 116,61, η οποία είναι σχετικά υψηλή και υποδηλώνει ένα χαμηλό επίπεδο συνοχής μεταξύ των μεθόδων των κλάσεων του έργου.
- Στην έκδοση 1.2.0, η LCOM έχει μία τεράστια πτώση στο 11.82, υποδεικνύοντας βελτίωση της συνοχής μεταξύ των μεθόδων των κλάσεων του έργου.
- Στην έκδοση 2.0.0, το LCOM μειώθηκε περαιτέρω σε 1,92, υποδεικνύοντας υψηλότερο επίπεδο συνοχής μεταξύ των μεθόδων των κλάσεων του έργου.
- Τέλος, στην έκδοση 2.1.1, το LCOM αυξήθηκε εκ νέου σε 11,82. Ένα υψηλό LCOM, γεγονός που καθιστά το έργο δύσκολο στην κατανόηση και τη συντήρησή του.

Στην περίπτωση μας θεωρούμε πως το version 2.0.0 είναι το ευκολότερο ως προς την κατανόηση και συντήρηση (όσον αφορά τις τιμές της LCOM) καθώς έχει και τον μικρότερο βαθμό LCOM.

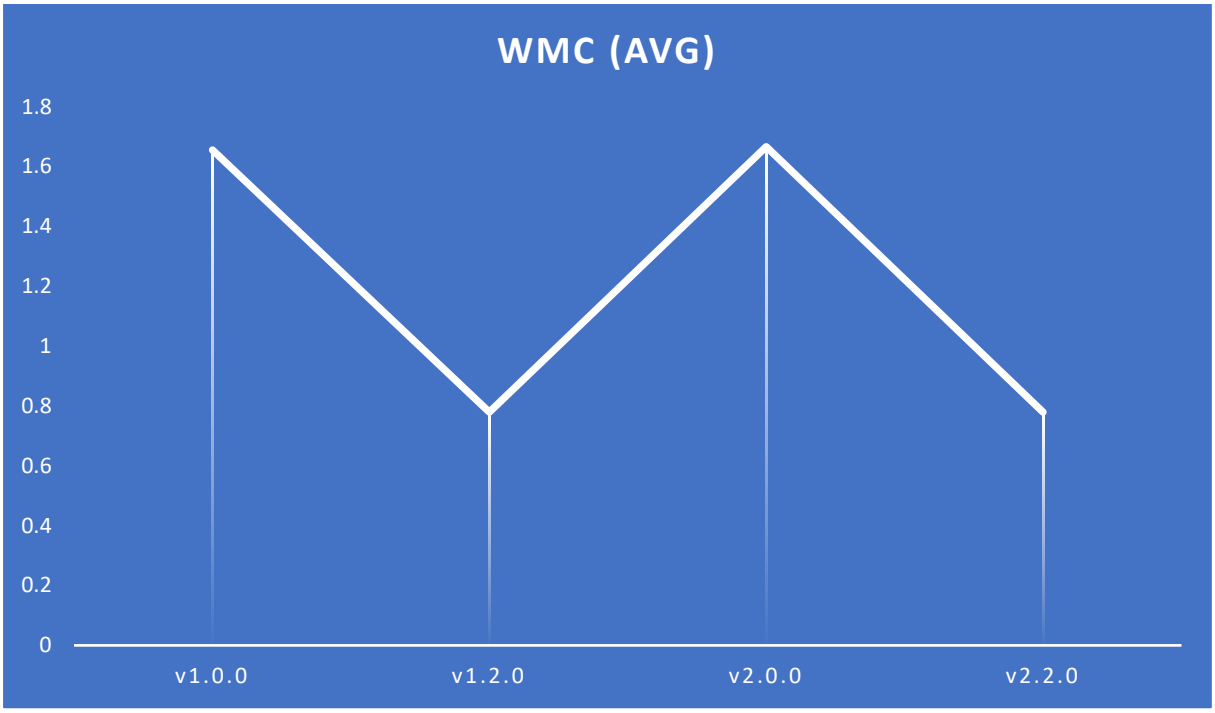
Οι υψηλές τιμές LCOM υποδεικνύουν ότι το έργο έχει χαμηλή συνοχή μεταξύ των μεθόδων των κλάσεων του και είναι ένα σημάδι ότι οι κλάσεις προσπαθούν να κάνουν πάρα πολλά και μπορεί να

παραβιάζουν την Αρχή της Ενιαίας Ευθύνης (SRP). Αυτό μπορεί να καταστήσει το λογισμικό πιο δύσκολο στη συντήρηση και τροποποίηση, αφού οι αλλαγές σε ένα μέρος της κλάσης μπορεί να έχουν ακούσιες συνέπειες σε άλλα μέρη της κλάσης.

Ακόμη, αν θέλαμε να προσθέσουμε ένα νέο feature που απαιτούσε την τροποποίηση μιας υπάρχουσας κλάσης με υψηλές τιμές LCOM, πιθανότατα να έπρεπε να γίνουν πολλαπλές αλλαγές στην κλάση, και ενδεχομένως να παραβιάζονταν η Αρχή Ανοικτής Κλειστής Σχεδίασης.

Αξίζει λοιπόν να εξεταστεί το ενδεχόμενο αναδιαμόρφωσης του κώδικα για να βελτιωθεί η συνοχή των κλάσεων και η συντηρησιμότητα του έργου.

- **WMC (Weighted Methods per Class)**



Άλλη μια μετρική που ξεχωρίζει είναι η WMC που παρουσιάζει κάποιες διακυμάνσεις με την πάροδο του χρόνου.

- Στην έκδοση 1.0.0, η WMC ήταν 1,66, η οποία είναι σχετικά υψηλή και υποδηλώνει ένα υψηλό επίπεδο πολυπλοκότητας.
- Στην συνέχεια στην έκδοση 1.2.0, η WMC πέφτει σημαντικά σε 0,78, υποδηλώνοντας απλοποίηση του κώδικα.
- Στην έκδοση 2.0.0, ο μέσος όρος WMC αυξήθηκε και πάλι σε 1,67, πριν μειωθεί ξανά ελαφρώς σε 0,78, για την έκδοση 2.1.1.

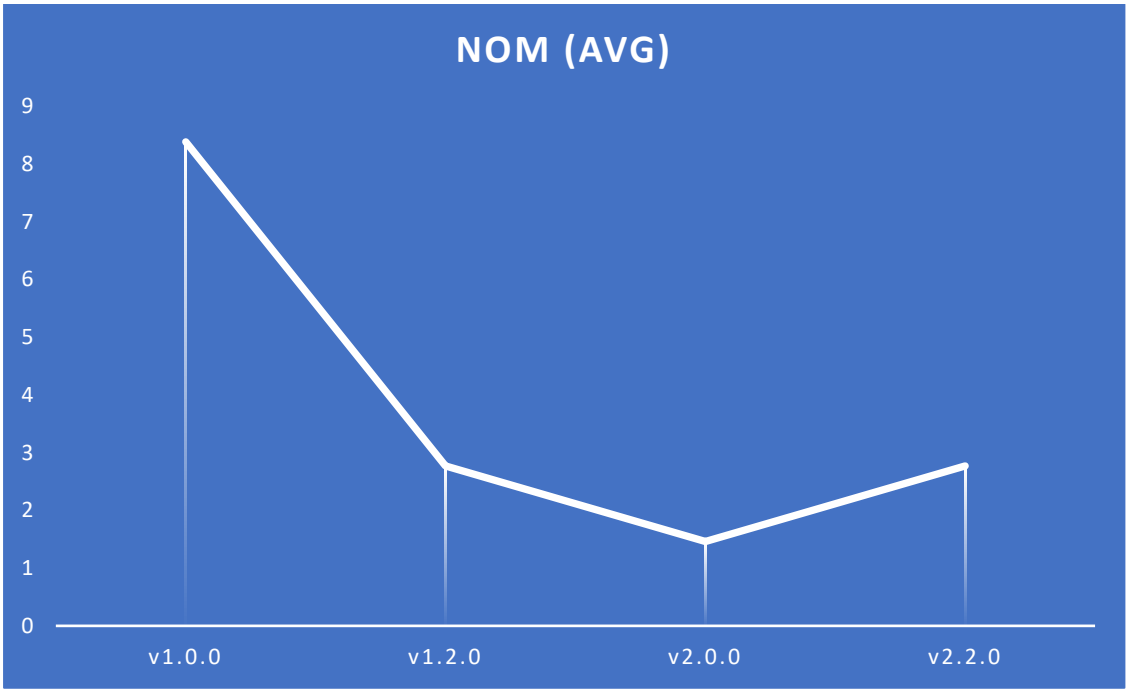
Είναι ενδιαφέρον να σημειωθεί το μοτίβο που φαίνεται να ακολουθούν οι αλλαγές στην WMC, με μια μείωση που ακολουθείται από μια αύξηση.

Οι υψηλές τιμές WMC μπορεί να υποδεικνύουν ότι μια κλάση προσπαθεί να κάνει πάρα πολλά και είναι κακοσχεδιασμένη, γεγονός που μπορεί να κάνει τον κώδικα πιο δύσκολο να κατανοηθεί και να συντηρηθεί. Για παράδειγμα, αν θέλαμε να προσθέσουμε ένα νέο feature σε μια κλάση με υψηλή τιμή WMC, αυτό πιθανότατα να απαιτούσε σημαντική τροποποίηση της κλάσης, παραβιάζοντας την Αρχή Ανοιχτού Κλειστού Σχεδιασμού. Από την άλλη πλευρά, οι

χαμηλότερες τιμές WMC υποδηλώνουν έναν απλούστερο και πιο «αρθρωτό» σχεδιασμό, ο οποίος μπορεί να κάνει τον κώδικα πιο εύκολο στην κατανόηση και συντήρηση.

Επομένως, αξίζει να εξεταστούν οι λόγοι αυτών των αλλαγών και κατά πόσον υπάρχουν ευκαιρίες για περαιτέρω απλοποίηση του κώδικα ή αντιμετώπιση ζητημάτων που συμβάλλουν στην πολυπλοκότητα των κλάσεων.

- **NOM (Number Of Methods)**



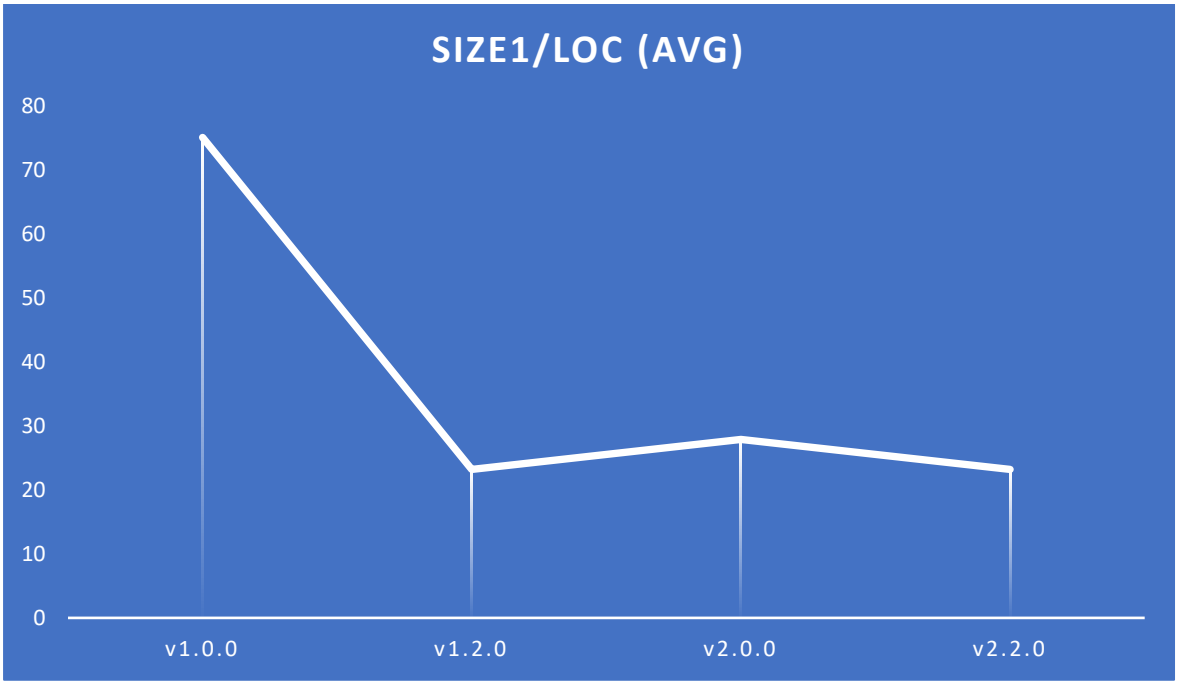
Για τη μετρική NOM, μπορούμε να δούμε ότι υπάρχει μείωση του μέσου αριθμού μεθόδων ανά κλάση από την έκδοση 1.0.0 έως την έκδοση 1.2.0, ακολουθούμενη από αύξηση στην έκδοση 2.0.0 και στη συνέχεια επιστροφή στα επίπεδα της έκδοσης 1.2.0 στην έκδοση 2.1.1.

Οι υψηλές τιμές NOM υποδηλώνουν ότι μια κλάση μεγάλο αριθμό μεθόδων και κατα συνέπεια έχει πολλές αμοδιότητες και προσπαθεί να κάνει πάρα πολλά, γεγονός που μπορεί να κάνει την κατανόηση και συντήρησή της πιο δύσκολη.

Αξίζει να διερευνηθεί γιατί οι τιμές NOM άλλαξαν με αυτόν τον τρόπο και αν υπάρχουν ευκαιρίες για αναδιαμόρφωση του κώδικα ώστε να βελτιωθεί ο σχεδιασμός των κλάσεων και να μειωθούν οι ευθύνες τους, ακολουθώντας την Αρχή της Ενιαίας Ευθύνης.

Για παράδειγμα, αν θέλουμε να προσθέσουμε ένα νέο feature στο έργο, θα μπορούσαμε να εξετάσουμε το ενδεχόμενο να δημιουργήσουμε μια νέα κλάση ειδικά για να χειριστεί αυτή την λειτουργία αντί να προσθέσουμε τον κώδικα σε μια υπάρχουσα κλάση με υψηλή τιμή NOM, αυτό θα βοηθούσε στη διατήρηση της συντηρησιμότητας του κώδικα.

- **SIZE1 (Lines Of Code/LOC)**



Η μετρική LOC μειώθηκε σημαντικά μεταξύ των εκδόσεων 1.0.0 και 1.2.0, αλλά αυξήθηκε ελαφρώς μεταξύ των εκδόσεων 2.0.0 και 2.2.0.

- Στην έκδοση 1.0.0, ο LOC ήταν 75.16, υποδεικνύοντας ένα μεγάλο και πολύπλοκο έργο.
- Στην έκδοση 1.2.0, ο LOC μειώθηκε δραματικά σε 23,23, υποδεικνύοντας απλοποίηση της βάσης κώδικα.
- Στην έκδοση 2.0.0, ο LOC αυξήθηκε και πάλι σε 27,92, πριν ξαναπέσει και πάλι σε 23,22 στην έκδοση 2.1.1.

Μια μεγάλη τιμή LOC υποδηλώνει ένα μεγάλο και πολύπλοκο έργο. Θέλουμε η έκδοση να έχει όσο το δυνατόν μικρότερο LOC για εύκολη συντηρησιμότητα. Σε αυτή την περίπτωση, θεωρούμε ότι η έκδοση 2.1.1 είναι η καλύτερη έκδοση από άποψη LOC με τιμή 23,22.

Η μετρική Lines Of Code (LOC) παρουσίασε σημαντική πτώση μεταξύ των εκδόσεων 1.0.0 και 1.2.0, προτού σταθεροποιηθεί στις επόμενες εκδόσεις. Είναι ενδιαφέρον να σημειωθεί η δραστική μείωση των γραμμών κώδικα, η οποία μπορεί να υποδηλώνει την εστίαση στη βελτιστοποίηση του κώδικα και της αποδοτικότητας.

Αξίζει να εξεταστεί γιατί οι τιμές LOC άλλαξαν με αυτόν τον τρόπο και αν υπάρχουν ευκαιρίες για αναδιαμόρφωση του κώδικα ώστε να μειωθεί η πολυπλοκότητα και να βελτιωθεί η συντηρησιμότητα του έργου. Για παράδειγμα, αν θέλουμε να προσθέσουμε ένα νέο feature που απαιτεί πολύπλοκη επεξεργασία, ίσως χρειαστεί να προσθέσουμε σημαντική ποσότητα κώδικα στο έργο, γεγονός που θα μπορούσε να αυξήσει την τιμή LOC και ενδεχομένως να καταστήσει τον κώδικα πιο δύσκολο στην κατανόηση και τη συντήρηση.

Προκειμένου να αποφευχθεί η άσκοπη αύξηση της τιμής LOC, καλό θα ήταν να ακολοθουνται αρχές όπως η Single Responsibility Principle, και η Open-Closed. Ακολουθώντας αυτές τις αρχές, μπορούμε να βοηθήσουμε στην διασφάλιση ότι ο κώδικας που προσθέτουμε είναι συντηρήσιμος και εύκολα κατανοητός, ενώ παράλληλα ανταποκρίνεται στις απαιτήσεις του έργου.

4. Αναγνώριση και Επίλυση Προβλημάτων Ποιότητας

4.1. Εισαγωγή

Στόχος αυτής της ενότητας είναι να εξηγήσει τις αλλαγές που έγιναν στην βάση κώδικα και τον αντίκτυπο των αλλαγών αυτών στη συνολική ποιότητα του έργου. Πριν από τον επανασχεδιασμό, το έργο υπέφερε από μια σειρά ζητημάτων που εμπόδιζαν την κατανόηση, συντηρησιμότητα και επεκτασιμότητά του. Μετά το refactoring, το έργο αναδιαρθρώθηκε με τρόπο που το καθιστά πιο οργανωμένο, συντηρήσιμο και επεκτάσιμο.

Η αρχική βάση κώδικα δεν ακολουθούσε τον αντικειμενοστραφή προγραμματισμό (OOP) και τις αρχές σχεδιασμού. Το πρόγραμμα ήταν δύσκολο να κατανοηθεί και να συντηρηθεί και ήταν δύσκολο να προστεθούν νέα χαρακτηριστικά. Το πρόγραμμα είχε επίσης υψηλό βαθμό σύζευξης μεταξύ των κλάσεων, γεγονός που καθιστούσε δύσκολη τη δοκιμή και την αποσφαλμάτωση.

Πιο συγκεκριμένα εντοπίσαμε τα εξής προβλήματα:

- **Πρόβλημα 1**: Έλλειψη ειδικών κλάσεων για το DNA, τις πρωτεΐνες και τα αμινοξέα
- **Πρόβλημα 2**: Έλλειψη διαχωρισμού μεταξύ επεξεργασίας δεδομένων και εκτύπωσης στο CLI
- **Πρόβλημα 3**: Έλλειψη σωστής μορφοποίησης και εκτύπωσης των αποτελεσμάτων
- **Πρόβλημα 4**: Έλλειψη αφαίρεσης στο χειρισμό αρχείων
- **Πρόβλημα 5**: Έλλειψη σωστού χειρισμού και ενθυλάκωσης των arguments της γραμμής εντολών
- **Πρόβλημα 6**: Έλλειψη της αρχής της ενιαίας ευθύνης (SRP) στην κλάση DNAnalyzer (1)
- **Πρόβλημα 7**: Έλλειψη της αρχής της ενιαίας ευθύνης (SRP) στην κλάση DNAnalyzer (2)

Για τους σκοπούς αυτής της εργασίας, επικεντρωθήκαμε στη διόρθωση του CLI του προγράμματος. Οι αλλαγές που έγιναν βελτίωσαν σημαντικά τη συνολική ποιότητα του προγράμματος, καθιστώντας το πιο κατανοητό, συντηρήσιμο και επεκτάσιμο.

4.2. Πρόβλημα 1: Έλλειψη ειδικών κλάσεων για το DNA, τις πρωτεΐνες και τα αμινοξέα (Χρήστος Χαχούδης)

4.2.1. Δήλωση Προβλήματος

Η αρχική βάση κώδικα δεν διέθετε ειδικές κλάσεις για τον χειρισμό αλληλουχιών για το DNA, τις πρωτεΐνες και τα αμινοξέα, αλλά χρησιμοποιούσε συμβολοσειρές για την αναπαράστασή τους (από τα ορίσματα της γραμμής εντολών), πράγμα που είναι προβληματικό, καθώς δεν παρέχει την απαραίτητη αφαίρεση και ενθυλάκωση για αυτούς τους τύπους αλληλουχιών.

4.2.2. Λύση

Για την αντιμετώπιση αυτού του προβλήματος, εισάγαμε ειδικές κλάσεις για το χειρισμό ακολουθιών για DNA, Πρωτεΐνες και αμινοξέα.

Αυτές οι κλάσεις χρησιμοποιούν κληρονομικότητα (εκτος της κλάσης για τα αμινοξεα) καθώς είναι πολύ παρόμοιες, όλες έχουν μια ιδιότητα ακολουθίας και κοινές λειτουργίες. Μεταφέραμε επίσης τη λειτουργικότητα από το **DNATools** σε μια νέα κλάση που ονομάζεται "**DNAToolkit**". Αυτή η κλάση είναι ειδικά υπεύθυνη για το χειρισμό των ακολουθιών DNA. Επιπλέον, υλοποιήσαμε μια πιο σωστή έκδοση του προτύπου **AminoAcidFactory** που δημιουργεί και αποθηκεύει περιπτώσεις της κλάσης **AminoAcid** σε ένα **HashMap**, επιτρέποντας την εύκολη ανάκτηση των περιπτώσεων με βάση το όνομα.

4.2.3. Αλλαγές

1. Δημιουργήθηκε ένα νέο πακέτο με όνομα model, που θα φιλοξενήσει τις κλάσσεις που θα φτιάξουμε.
2. **Sequence**: Μια αφηρημένη κλάση από την οποία κληρονομούν οι κλάσεις **DNA** και **Protein**.
3. **DNA**: Μια κλάση για το χειρισμό αλληλουχιών DNA.
4. **Protein**: Μια κλάση για το χειρισμό αλληλουχιών πρωτεϊνών.
5. **AminoAcid**: Μια κλάση για το χειρισμό αλληλουχιών αμινοξέων.
6. **AminoAcidFactory**: Μια κλάση που ακολούθει το προτυπο Factory για μαζικη κατασκευη των αμινοξεων (ηταν σε ενα enum στην αρχικη βαση κώδικα)

4.2.4. Οφέλη των Αλλαγών

1. Το έργο είναι πλέον πιο συντηρήσιμο, καθώς οι νέες κλάσεις και το πρότυπο factory εξασφαλίζουν σωστή ενθυλάκωση και συντηρησιμότητα.
2. Το έργο είναι πλέον πιο κατανοητό, καθώς είναι πλέον σαφές τι τύπους ακολουθιών χειρίζεται το πρόγραμμα και πώς τις χειρίζεται.
3. Οι νέες κλάσεις διευκολύνουν την προσθήκη νέων χαρακτηριστικών και τη διόρθωση σφαλμάτων στο μέλλον.
4. Η κλάση **DNAToolkit** είναι πλέον ειδικά υπεύθυνη για τον χειρισμό των ακολουθιών DNA, γεγονός που βελτιώνει την αναγνωσιμότητα του κώδικα.
5. Το πρότυπο **AminoAcidFactory** συμβάλλει στη διατήρηση του κώδικα οργανωμένου, εύχρηστου και κατανοητού.

4.2.4. Κώδικας

Πρίν την αναδιαμόρφωση:

Η Κλάση **DNATools**:

```
public record DNATools(String dna) {
    public void isValid() {
        if (!dna.matches("[atgc]+")) {
            throw new IllegalArgumentException("Invalid characters present in
DNA sequence.");
        }
    }

    public DNATools replace(final String input, final String replacement) {
        return new DNATools(this.dna.replace(input, replacement));
    }

    public DNATools reverse() {
        return new DNATools(new StringBuilder(dna).reverse().toString());
```

```
    }

    public String getDna() {
        return dna;
    }
}
```

Το enum **AminoAcid**:

```
public enum AminoAcid {
    ALANINE("Alanine", List.of("a", "alanine", "ala"), List.of("GCT", "GCC",
"GCA", "GCG")),
    CYSTEINE("Cysteine", List.of("c", "cysteine", "cys"), List.of("TGT",
"TGC")),
    ASPARTIC_ACID("Aspartic acid", List.of("d", "aspartic acid", "asp"),
List.of("GAT", "GAC")),

    // rest of aminoacids

    private final String fullName;
    private final List<String> names;

    private final List<String> codonData;

    AminoAcid(final String fullName, final List<String> abbreviations, final
List<String> codonData) {
        this.fullName = fullName;
        this.names = abbreviations;
        this.codonData = codonData;
    }

    public List<String> getNames() {
        return names;
    }

    public String getFullName() {
        return fullName;
    }

    public List<String> getCodonData() {
        return codonData;
    }
}
```

Η Κλάση **AminoAcidFactory**:

```
public class AminoAcidFactory {

    public static AminoAcid getAminoAcid(final String aminoAcid) {
        return Arrays.stream(AminoAcid.values())
                .filter(acid ->
acid.getNames().contains(aminoAcid.toLowerCase()))
                .findFirst()
                .orElseThrow(() -> new IllegalArgumentException("Unknown amino
acid."));
    }
}
```

Μετά την Αναδιαμόρφωση:

Η κλάση **Sequence**:

```
package model;

public abstract class Sequence {
    private String sequence;

    public Sequence(String sequence) {
        this.sequence = sequence;
    }

    public int length() {
        return sequence.length();
    }

    public String getSequence() {
        return sequence;
    }

    public void setSequence(String sequence) {
        this.sequence = sequence;
    }
}
```

Η κλάση **DNA**:

```
public class DNA extends Sequence {

    public DNA(String sequence) {
        super(sequence);
    }

    @Override
    public String toString() {
        return "DNA{" +
                "sequence=" + super.getSequence() + '\'' +
                '}';
    }
}
```

Η Κλάση **Protein**:

```
package model.protein;

import model.Sequence;

public class Protein extends Sequence {
    public Protein(String sequence) {
        super(sequence);
    }
}
```

Η Κλάση **AminoAcid**:

```
public class AminoAcid {
    private String fullName;
    private List<String> abbreviations;
    private List<String> codons;
```

```
    public AminoAcid(String fullName, List<String> abbreviations,
List<String> codons) {
        this.fullName = fullName;
        this.abbreviations = abbreviations;
        this.codons = codons;
    }

    public String getFullName() {
        return fullName;
    }
    public List<String> getCodons() {
        return codons;
    }

}
```

Η Κλάση **AminoAcidFactory**:

```
package model.aminoAcid;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class AminoAcidFactory {
    private static final Map<String, AminoAcid> aminoAcids = new
HashMap<>();

    static {
        AminoAcid alanine = new AminoAcid("Alanine", List.of("a",
"alanine", "ala"), List.of("GCT", "GCC", "GCA", "GCG"));
        aminoAcids.put("ALANINE", alanine);

        AminoAcid cysteine = new AminoAcid("Cysteine", List.of("c",
"cysteine", "cys"), List.of("TGT", "TGC"));
        aminoAcids.put("CYSTEINE", cysteine);

       // rest of acids
    }

    public static AminoAcid getAminoAcid(String name) {
        return aminoAcids.get(name);
    }
    public static Map<String, AminoAcid> getAminoAcids() { return
aminoAcids; }
}
```

4.2.6. Συμπεράσματα

Οι αλλαγές που έγιναν στην αρχική βάση κώδικα του έργου αντιμετώπισαν το ζήτημα της έλλειψης ειδικών κλάσεων για το DNA, τις πρωτεΐνες και τα αμινοξέα. Οι νέες κλάσεις, το **DNAToolkit** και το πρότυπο factory βελτίωσαν τη συντηρησιμότητα, την κατανόηση και την οργάνωση του έργου. Η νέα δομή καθιστά ευκολότερη την κατανόηση της λειτουργικότητας κάθε κλάσης και την πραγματοποίηση αλλαγών στο έργο στο μέλλον. Με τις νέες κλάσεις και το πρότυπο factory, το έργο ακολουθεί πλέον περισσότερο την Αρχή της Ενιαίας Ευθύνης (Single Responsibility Principle - SRP), η οποία είναι μια σημαντική αρχή σχεδιασμού λογισμικού.

4.3. Πρόβλημα 2: Έλλειψη διαχωρισμού μεταξύ επεξεργασίας δεδομένων και εκτύπωσης στο CLI (Χρήστος Χαχούδης)

4.3.1. Δήλωση Προβλήματος

Η αρχική βάση κώδικα δεν διαχώριζε τον τρόπο επεξεργασίας και εκτύπωσης των δεδομένων. Η λειτουργικότητα ήταν διασκορπισμένη σε πολλές κλάσεις και γενικά δεν ακολουθούσε τον αντικειμενοστραφή προγραμματισμό (OOP) και τις αρχές σχεδιασμού. Το πρόγραμμα χειριζόταν όλα τα δεδομένα ως `Strings` και οι ίδιες μέθοδοι/κλάσεις που χειρίζονταν τα δεδομένα ήταν υπεύθυνες και για την εκτύπωση.

4.3.2. Λύση

Για την αντιμετώπιση αυτού του προβλήματος, προσδιορίσαμε πρώτα τους διαφορετικούς τύπους δεδομένων που ήθελε να εκτυπώσει το πρόγραμμα. Διαπιστώσαμε ότι η μέθοδος run της κλάσης `CmdArgs` είχε κώδικα που εκτύπωνε συγκεκριμένα δεδομένα σχετικά με το DNA, όπως η περιεκτικότητα σε GC, ο αριθμός νουκλεοτιδίων, οι περιοχές υψηλής κάλυψης, η μεγαλύτερη πρωτεΐνη και άλλα.

Για να διαχωρίσουμε τον χειρισμό των δεδομένων και την εκτύπωση, δημιουργήσαμε μια ξεχωριστή κλάση που ονομάζεται `AnalysisResult`. Αυτή η κλάση κρατάει όλα τα δεδομένα που θέλει να εκτυπώσει το πρόγραμμα, είναι ένα αντικείμενο δεδομένων που κρατάει τις πληροφορίες με πιο οργανωμένο τρόπο. Θα μπορούσαμε να πούμε οτι αυτή η κλάση είναι μια κλάσση POJO.

4.3.3. Αλλαγές

1. Δημιουργήθηκε μια νέα κλάση με όνομα `AnalysisResult`, η οποία κρατάει όλα τα δεδομένα που το πρόγραμμα θέλει να εκτυπώσει.
2. Η κλάση `AnalysisResult` είναι πλέον υπεύθυνη για την κατοχή των δεδομένων, γεγονός που βελτίωσε την αναγνωσιμότητα του κώδικα.

4.3.4. Οφέλη των Αλλαγών

1. Το έργο είναι πλέον πιο συντηρήσιμο, καθώς η νέα κλάση βελτιώνει την ενθυλάκωση των δεδομένων.
2. Το έργο είναι πλέον πιο οργανωμένο, καθώς τα δεδομένα αποθηκεύονται πλέον σε μια συγκεκριμένη κλάση, γεγονός που καθιστά εύκολη την κατανόηση των δεδομένων που εκτυπώνονται και του τρόπου χειρισμού τους.
3. Το έργο είναι πλέον πιο ευέλικτο, καθώς είναι ευκολότερο να προστεθούν νέα δεδομένα στην κλάση `AnalysisResult` και να εκτυπωθούν στο μέλλον.
4. Ο διαχωρισμός του χειρισμού των δεδομένων και της εκτύπωσης βελτιώνει την αναγνωσιμότητα του κώδικα και διευκολύνει την προσθήκη νέων χαρακτηριστικών.

4.3.5. Κώδικας

Πρίν την αναδιαμόρφωση:

Δεν άλλαξε κάτι από τον κώδικα πριν την αναδιαμόρφωση. Αφού απλά προστέθηκε μια νέα κλάσση.

Μετά την Αναδιαμόρφωση:

Η Κλάση **AnalysisResults**

```
public class AnalysisResult {
    private String inputAminoAcid;
    private List<Protein> dnaProteins;
    private double gcContent;
    private Map<String, Long> nucleotideCountMap;
    private Map<String, Integer> highCoverageRegions;
    private Protein longestProtein;
    private ReadingFrames readingFrames;
    private int proteinSequenceIndex;
    private boolean dnaIsRandom = false;

    public AnalysisResult(String inputAminoAcid) {
        this.inputAminoAcid = inputAminoAcid;
    }

    // Getters - Setters

}
```

4.3.6. Συμπεράσματα

Για να βελτιωθεί η ποιότητα του έργου, η αντιμετωπίσαμε το ζήτημα της έλλειψης διαχωρισμού μεταξύ της επεξεργασίας δεδομένων και της εκτύπωσης στη γραμμή εντολών (CLI) δημιουργώντας μια νέα κλάση με την ονομασία **AnalysisResult**. Αυτή η κλάση περιέχει όλα τα δεδομένα που το πρόγραμμα θέλει να εκτυπώσει και βελτίωσε την αναγνωσιμότητα του κώδικα. Ο διαχωρισμός της επεξεργασίας δεδομένων και της εκτύπωσης βελτιώνει την αναγνωσιμότητα του κώδικα και διευκολύνει την προσθήκη νέων χαρακτηριστικών. Αυτές οι αλλαγές βελτίωσαν σημαντικά τη συνολική ποιότητα του προγράμματος, καθιστώντας το πιο συντηρήσιμο, οργανωμένο και ευέλικτο.

4.4. Πρόβλημα 3: Έλλειψη σωστής μορφοποίησης και εκτύπωσης των αποτελεσμάτων (Χρήστος Χαχούδης)

4.4.1. Δήλωση Προβλήματος

Η αρχική βάση κώδικα δεν διέθετε συγκεκριμένη κλάση ή μηχανισμό για τη μορφοποίηση και την εκτύπωση των αποτελεσμάτων της ανάλυσης DNA. Ο κώδικας που ήταν υπεύθυνος για την εκτύπωση των αποτελεσμάτων ήταν διασκορπισμένος σε πολλές κλάσεις και δεν ήταν ενθυλακωμένος ή εύκολα κατανοητός. Ακόμη, σε πολλές περιπτώσεις οι μέθοδοι που επεξεργάζονταν τα δεδομένα ήταν και υπεύθυνες και για την εκτύπωση τους.

4.4.2. Λύση

Για την αντιμετώπιση αυτού του προβλήματος, δημιουργήσαμε μια νέα κλάση με την ονομασία **ResultPrinter**. Αυτή η κλάση είναι ειδικά υπεύθυνη για τη μορφοποίηση και την εκτύπωση των αποτελεσμάτων της ανάλυσης DNA, χρησιμοποιώντας τα δεδομένα που είναι αποθηκευμένα στην κλάση **AnalysisResult** που αναφέρθηκε ως λύση σε ένα από τα παραπάνω προβλήματα.

4.4.3. Αλλαγές

1. Δημιουργήθηκε μια νέα κλάση με όνομα **ResultPrinter**, η οποία είναι υπεύθυνη για τη μορφοποίηση και την εκτύπωση των αποτελεσμάτων της ανάλυσης DNA.

2. Η κλάση **ResultPrinter** χρησιμοποιεί τα δεδομένα από την κλάση **AnalysisResult** για την εκτύπωση των αποτελεσμάτων.
3. Η κλάση **ResultPrinter** περιέχει διάφορες μεθόδους που εκτυπώνουν συγκεκριμένες πληροφορίες σχετικά με την ανάλυση DNA.
4. Η κλάση **Terminal** (θα αναφερθει από το άλλο μέλος της ομάδας) χρησιμοποιεί τώρα μια αναφορά της κλάσης **ResultPrinter** για να εκτυπώσει τα αποτελέσματα της ανάλυσης στην κονσόλα.

4.4.4. Οφέλη των Αλλαγών

1. Το έργο είναι πλέον πιο συντηρήσιμο, καθώς η νέα κλάση βελτιώνει την ενθυλάκωση των δεδομένων και της λογικής εκτύπωσης.
2. Το έργο είναι πλέον πιο οργανωμένο, καθώς τα δεδομένα αποθηκεύονται πλέον σε μια συγκεκριμένη κλάση και εκτυπώνονται από μια άλλη κλάση, γεγονός που καθιστά εύκολη την κατανόηση των δεδομένων που εκτυπώνονται και του τρόπου μορφοποίησής τους.
3. Το έργο είναι τώρα πιο ευέλικτο, καθώς είναι ευκολότερο να προστεθούν νέα δεδομένα στην κλάση **AnalysisResult** και να τα εκτυπώσουμε στο μέλλον.
4. Ο διαχωρισμός της μορφοποίησης των δεδομένων και της εκτύπωσης βελτιώνει την αναγνωσιμότητα του κώδικα και διευκολύνει την προσθήκη νέων χαρακτηριστικών.

4.4.5. Κώδικας

Πρίν την αναδιαμόρφωση:

Δεν άλλαξε κάτι από τον κώδικα πριν την αναδιαμόρφωση. Αφού απλά προστέθηκε μια νέα κλάση.

Παρόλα αυτά θα δείξουμε ένα κομμάτι κώδικα για να δείξουμε μια κλάση που ταυτόχρονα ανέλυε δεδομένα και εκτύπωνε αποτελέσματα.

```
public class ProteinAnalysis {

   public static void printHighCoverageRegions(final List<String> geneList,
PrintStream out) {
      short count = 1;

      // print the list of genes with the highest GC content
      out.println("\nHigh coverage regions: ");
      out.println("-----------------------------------------------------");

      for (final String gene : geneList) {

         // High GC content range
         final float MIN_GC_CONTENT = 0.40f;
         final float MAX_GC_CONTENT = 0.60f;
         if ((Properties.getGCContent(gene) > MIN_GC_CONTENT) &&
(Properties.getGCContent(gene) < MAX_GC_CONTENT)) {
            out.println(count + ". " + gene);
            count++;
         }
      }
   }

   public static void printLongestProtein(final List<String> proteinList,
PrintStream out) {
      String longestProtein = "";
      for (final String protein : proteinList) {
```

```
            if (protein.length() > longestProtein.length()) {
                longestProtein = protein;
            }
        }
        out.println("\nLongest gene (" + longestProtein.length() + "
nucleotides): " + longestProtein);
    }
}
```

Σε αυτή την κλάση βλέπουμε ότι οι μέδοδοι **printLongestProtein** και **printHighCoverageRegions** και εκτελουν την λειτουργια ευρεσης αλλα και εκτυπωσης.

Με τις αλλαγές που εισάγουμε θέλουμε να έχουμε ένα εννιαίο αντικείμενο για τα αποτελέσματα, και οι κλάσεις που επεξεργάζονται τα δεδομένα θα αναδιαμορφωθούν σε επόμενη ενότητα ωστε μόνο να επεξεργάζονται δεδομένα και όχι να εκτυπώνουν. Η Κλάση **Terminal** θα αναλάβει την εκτύπωση των δεδομένων που βρίσκονται μέσα στην κλάση **AnalysisResult**.

Μετά την Αναδιαμόρφωση:

```
public class ResultPrinter {
    private AnalysisResult result;

    public void printProteinAnalysis() {
        // Changes the 1 letter or 3 letter abbreviation of the amino acids
into the
        // full name
        final AminoAcid acid =
AminoAcidFactory.getAminoAcid(result.getInputAminoAcid());

        System.out.println("Proteins coded for: " + acid.getFullName() + ":
");
        System.out.println("-----------------------------------------------
------------");

        short count = 1;
        for (final Protein gene : result.getDnaProteins()) {
            System.out.println(count + ". " + gene.getSequence());
            count++;
        }
    }

    public void printHighCoverageAreas() {
        System.out.println("\nHigh coverage regions: ");
        System.out.println();

        Map<String, Integer> highCoverageRegions =
result.getHighCoverageRegions();
        for (Map.Entry<String, Integer> mapEntry :
highCoverageRegions.entrySet()){
            String gene = mapEntry.getKey();
            int count = mapEntry.getValue();
            System.out.println(count + ". " + gene);
        }
    }

    public void printLongestProtein() {
        String proteinSequence = result.getLongestProtein()
                .getSequence();
        int proteinLength = result.getLongestProtein()
                .length();
```

```
        System.out.println(
                "\nLongest gene ("
                        + proteinLength
                        + " nucleotides): "
                        + proteinSequence);
    }

    public void printNucleotideCount() {
        Map<String, Long> nucleotideCountMap =
result.getNucleotideCountMap();
        for(Map.Entry<String, Long> nucleotide :
nucleotideCountMap.entrySet()) {
            String nucleotideName = nucleotide.getKey();
            Long nucleotideCount = nucleotide.getValue();

            System.out.println(
                    nucleotideName + ": "
                            + nucleotideCount + " %"
            );
        }

    }

    public void printReadingFrames() {
        CodonFrame codonFrame = result.getReadingFrames()
                .getCodonFrame();
        Map<String, Integer> codonCounts = result.getReadingFrames()
                .getCodonCounts();

        String title = "Codons in reading frame " +
codonFrame.getReadingFrame() + " (" + codonFrame.getMin() + "-"
                + codonFrame.getMax() + " occurrences)" + ":";


        System.out.println(title);
        System.out.println("----------------------------------------------
------------");

        for (final Map.Entry<String, Integer> entry :
codonCounts.entrySet()) {
            if (codonCounts.get(entry.getKey()) >= codonFrame.getMin()
                    && codonCounts.get(entry.getKey()) <=
codonFrame.getMax()) {
                String codonKey = entry.getKey().toUpperCase();
                Integer codonCount = codonCounts.get(entry.getKey());
                System.out.println( codonKey + ": " + codonCount);
            }
        }
    }

    public void printProteinSequenceResults() {
        int sequenceIndex = result.getProteinSequenceIndex();

        if(sequenceIndex == -1)
            System.out.println("\nProtein sequence not found in the DNA
sequence.");
        else
            System.out.println("\nProtein sequence found at index " +
sequenceIndex + " in the DNA sequence.");
```

```
    }

    public void printRandomDnaInfo() {
        if(result.isDnaIsRandom())
            System.out.println("\nThe DNA Sequence Detected to be
Random.");
    }

    public AnalysisResult getResult() {
        return result;
    }

    public void setResult(AnalysisResult result) {
        this.result = result;
    }
}
```

Για την υλοποίηση των μεθόδων βρήκαμε πώς το πρόγραμμα εκτυπωνει τα αποτελεσματα (απο τις κλασσεις που εκτελουσαν και τις δυο λειτουργίες) και ως δεδομένα εκτυπώνουμε τα δεδομένα στην AnalysisResult

Η κλάση αυτή λοιπόν θα είναι υπεύθυνη για την εκτύπωση των αποτελεσμάτων στην κονσόλα.

4.5. Πρόβλημα 4: Έλλειψη αφαίρεσης στο χειρισμό αρχείων (Αρχοντής Κωστής)

4.5.1. Δήλωση Προβλήματος

Ο στόχος αυτής της ενότητας είναι να περιγράψει τις αλλαγές που έγιναν στην αρχική βάση κώδικα ενός έργου για την αντιμετώπιση του προβλήματος της έλλειψης αφαίρεσης στο χειρισμό αρχείων.

Η αρχική βάση κώδικα δεν διέθετε συγκεκριμένο μηχανισμό για το χειρισμό αρχείων και είχε μία μόνο μέθοδο για το χειρισμό αρχείων, τη μέθοδο **`readFile`** στην κλάση **`Utils`**, η οποία διάβαζε τα αρχεία με τρόπο που να μπορούμε να πάρουμε δεδομένα μόνο από αρχεία fasta και όχι από οποιοδήποτε άλλο τύπο αρχείου. Επίσης, αυτή η κλάση είχε μόνο την αρμοδιότητα διαβάσματος απο αρχεία και επέστρεφε μια αλληλουχία DNA, καθιστώντας το όνομα Utils ακατάλληλο για το σκοπό της κλάσης.

Όλα αυτά έκαναν το έργο άκαμπτο και δύσκολο στη συντήρηση, καθώς δεν ήταν δυνατό να προσθέσουμε εύκολα νέους τύπους αρχείων ή να χειριστούμε αρχεία με ευέλικτο και αποτελεσματικό τρόπο.

4.5.2. Λύση

Για την αντιμετώπιση αυτού του προβλήματος, αναδιαμορφώσαμε την κλάση **`Utils`** μετονομάζοντάς την σε **`FastaReader`** και δημιουργώντας μια ένα νέο Interface με όνομα **`FileReader`** το οποίο υλοποιεί η **`FastaReader`**. Αυτή η αλλαγή επιτρέπει την εύκολη προσθήκη νέων τύπων αρχείων στο μέλλον, καθώς και μεγαλύτερη ευελιξία στο χειρισμό αρχείων. Η κλάση

`FastaReader` και η διεπαφή `FileReader` μεταφέρθηκαν επίσης σε ένα νέο πακέτο που ονομάζεται io και μέσα σε ένα νέο υποπακέτο file.

4.5.3. Αλλαγές

1. Δημιουργήθηκε ένα νέο πακέτο με όνομα file, το οποίο περιέχει την αναμορφωμένη κλάση `FastaReader` και τη νέα διεπαφή `FileReader`.
2. Μετονομάστηκε η `Utils` σε `FastaReader` και υλοποιήθηκε η διεπαφή `FileReader`.
3. Υλοποιήθηκε μια μέθοδος `read()` στη διεπαφή `FileReader`.
4. Μεταφέρθηκε η κλάση `FastaReader` και η διεπαφή `FileReader` σε ένα νέο πακέτο με όνομα 'io' και μέσα στο υπερπακέτο ‘file’.

4.5.4. Οφέλη των Αλλαγών

1. Το έργο είναι πλέον πιο συντηρήσιμο, καθώς η προσθήκη νέων τύπων αρχείων μπορεί να επιτευχθεί εύκολα με την υλοποίηση της διεπαφής `FileReader`.
2. Το έργο είναι πλέον πιο ευέλικτο, καθώς είναι δυνατή η διαχείριση αρχείων με πιο αποδοτικό τρόπο.
3. Το έργο είναι τώρα πιο οργανωμένο, καθώς οι αναδιαμορφωμένες κλάσεις μεταφέρθηκαν σε ένα νέο πακέτο, το οποίο βοηθάει στην εύκολη πλοήγηση στο έργο.
4. Η ονομασία της κλάσης είναι πλέον κατάλληλη για τον σκοπό της, γεγονός που καθιστά εύκολη την κατανόηση της λειτουργικότητας της κλάσης.

4.5.5. Κώδικας

Πρίν την αναδιαμόρφωση:

```
public class Utils {
   public static String readFile(final File file) {
      try {
         return Files.readString(file.toPath()).replace("\n",
"").toLowerCase();
      } catch (IOException e) {
         return null;
      }
   }
}
```

Μετά την Αναδιαμόρφωση:

- Το νέο `FileReader` Interface

```
public interface FileReader {
    DNA read(File file);
}
```

- Η νέα `FastaReader` κλάση

```
public class FastaReader implements FileReader{

    @Override
    public DNA read(File file) {
        String sequence = "";
        try {
            sequence = Files.readString(file.toPath()).replace("\n",
"").toLowerCase();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return new DNA(sequence);
    }
}
```

Η μέθοδος **`read(File file)`** επιστρέφει αντικείμενα της κλάσσης **DNA**.

4.5.6. Συμπεράσματα

Οι αλλαγές που έγιναν στην αρχική βάση κώδικα του έργου αντιμετώπισαν το ζήτημα της έλλειψης αφαίρεσης στο χειρισμό αρχείων και βελτίωσαν την ευελιξία, τη συντηρησιμότητα και την οργάνωση του έργου. Το έργο είναι πλέον σε θέση να χειρίζεται αρχεία με πιο αποτελεσματικό τρόπο και είναι καλύτερα εξοπλισμένο για να φιλοξενήσει νέους τύπους αρχείων στο μέλλον.

4.6. Πρόβλημα 5: Έλλειψη σωστού χειρισμού και ενθυλάκωσης των arguments της γραμμής εντολών (Αρχοντής Κωστής)

4.6.1. Δήλωση Προβλήματος

Η αρχική βάση κώδικα είχε έλλειψη κατάλληλου χειρισμού και ενθυλάκωσης των arguments της γραμμής εντολών, γεγονός που είχε ως αποτέλεσμα τη στενή σύζευξη μεταξύ των διαφόρων τμημάτων της βάσης κώδικα και την έλλειψη κατάλληλης ενθυλάκωσης της λογικής χειρισμού των arguments της γραμμής εντολών. Αυτό καθιστούσε δύσκολη τη συντήρηση και την επέκταση του έργου.

4.6.2. Λύση

Για την αντιμετώπιση αυτού του προβλήματος, η δημιουργήσαμε ένα νέο πακέτο με την ονομασία controller, το οποίο περιέχει τις ακόλουθες κλάσεις:

1. Κλάση **`ArgumentParser`**: η οποία είναι υπεύθυνη για την ανάλυση των ορισμάτων της γραμμής εντολών και την επιστροφή ενός συνόλου επιλογών.
2. Κλάση **`Controller`**: η οποία είναι υπεύθυνη για την εκτέλεση του προγράμματος DNAnalyzer.

Μεταφέραμε επίσης τον κώδικα που σχετίζεται με το GUI στη μέθοδο **`controller.startGUI()`** και τον κώδικα που σχετίζεται με το CLI στη μέθοδο **`controller.startCLI()`**. Αυτό βοήθησε στην ενθυλάκωση της λογικής χειρισμού των ορισμάτων της γραμμής εντολών και στη μείωση της στενής σύζευξης μεταξύ των διαφόρων τμημάτων της βάσης κώδικα.

Ακόμη φτιάξαμε ένα νέο πακέτο με όνομα cli το οποίο τοποθετήσαμε στο υπερπακέτο με όνομα io. Σε αυτό το πακέτο προσθέσαμε μία ακόμη κλάση, την `Terminal` στην οποία μεταφέραμε την μέθοδο για την εκαθάριση της κονσόλας που προηγουμένως βρισκόταν στην `Main`. Η κλάση αυτή θα είναι υπεύθυνη για την εκτύπωση μυνημάτων στην γραμμή εντολών. Η κλάση `Terminal` ακολουθεί το Singleton Pattern ωστε να έχουμε μόνο μια αναφορά της σε όλο το πρόγραμμα. Τέλος προσθέσαμε μία μέθοδο στην `Terminal` με όνομα `printError(String Message)`. Όταν θα υπάρχει κάποιο Exception στο πρόγραμμα θα καλούμε αυτή την μέθοδο η οποία θα εκτυπώνει ένα μύνημα και στην συνέχεια θα τερματίζει το πρόγγραμα.

4.6.3. Αλλαγές

Δημιουργήθηκε ένα νέο πακέτο με όνομα controller το οποίο περιέχει:

1. Κλάση `ArgumentParser` η οποία είναι υπεύθυνη για την ανάλυση των ορισμάτων της γραμμής εντολών και την επιστροφή ενός συνόλου επιλογών.
2. Κλάση `Controller` η οποία είναι υπεύθυνη για την εκτέλεση του προγράμματος DNAnalyzer.
3. Κλάση `Terminal` η οποία είναι υπεύθυνη για την εκτύπωση μυνημάτων στην γραμμή εντολών
4. Μεταφέρθηκε ο κώδικας που σχετίζεται με το γραφικό περιβάλλον στη μέθοδο `controller.startGUI()`.
5. Μεταφέρθηκε ο κώδικας που σχετίζεται με το CLI στη μέθοδο `controller.startCLI()`.
6. Η μέθοδος main του έργου εκτελεί τώρα την κλάση `Controller`.

4.6.4. Οφέλη των Αλλαγών

1. Το έργο είναι πλέον πιο συντηρήσιμο, καθώς οι αλλαγές στο χειρισμό των arguments χρειάζεται να γίνουν μόνο στο πακέτο controller.
2. Το έργο είναι πλέον πιο ευέλικτο, καθώς είναι δυνατός ο χειρισμός των arguments με πιο αποτελεσματικό τρόπο.
3. Το έργο είναι πλέον πιο οργανωμένο, καθώς η λογική χειρισμού των arguments είναι πλέον ενθυλακωμένη στο πακέτο controller, γεγονός που βοηθάει στη διατήρηση της εύκολης πλοήγησης στο έργο.
4. Ο διαχωρισμός του κώδικα GUI και CLI βελτιώνει την αναγνωσιμότητα του κώδικα και διευκολύνει την προσθήκη νέων χαρακτηριστικών.

4.6.5. Κώδικας

Πρίν την αναδιαμόρφωση:

Κλάση `Main`:

```
public class Main {
   // Other code
   public static void clearTerminal() throws InterruptedException,
IOException {
   if (System.getProperty("os.name").contains("Windows")) { // if the os is
Windows
      new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
   } else {
      System.out.print("\u001b[H\u001b[2J"); // unicode string to clear
```

```
everything logged above this
        System.out.flush();
    }
}


    public static void main(final String[] args) {
        configureSentry();
        new CommandLine(new CmdArgs()).execute(args);
    }
}
```

Κλάση **CmdArgs**:

```
@Command(name = "DNAnalyzer", mixinStandardHelpOptions = true, description
= "A program to analyze DNA sequences.")
public class CmdArgs implements Runnable {
    @Option(names = { "--gui" }, description = "Start in GUI mode")
    Boolean startGUI = false;

    @Option(names = { "--amino" }, description = "The amino acid
representing the start of a gene.")
    String aminoAcid;

    @Option(names = { "--min" }, description = "The minimum count of the
reading frame.")
    int minCount = 0;

    @Option(names = { "--max" }, description = "The maximum count of the
reading frame.")
    int maxCount = 0;

    @Parameters(paramLabel = "DNA", description = "The FASTA file to be
analyzed.")
    File dnaFile;

    @Option(names = { "--find" }, description = "The DNA sequence to be
found within the FASTA file.")
    File proteinFile;

    @Option(names = { "--reverse", "-r" }, description = "Reverse the DNA
sequence before processing.")
    boolean reverse;


    @Override
    public void run() {
        if (startGUI == true) {
            String[] args = new String[0];
            DNAnalyzerGUI.launchIt(args);
        } else {
            DNAAnalysis dnaAnalyzer = dnaAnalyzer(aminoAcid)
                    .isValidDna()
                    .replaceDNA("u", "t");

            if (reverse == true) {
                dnaAnalyzer = dnaAnalyzer.reverseDna();
            }

            dnaAnalyzer
                    .printProteins(System.out)
```

```
                .printHighCoverageRegions(System.out)
                .outPutCodons(minCount, maxCount, System.out)
                .printLongestProtein(System.out);

        if (Properties.isRandomDNA(dnaAnalyzer.dna().getDna())) {
            System.out.println("\n" + dnaFile.getName() + " has been
detected to be random.");
        }
    }
}

private DNAAnalysis dnaAnalyzer(final String aminoAcid) {
    try {
        String protein = null;
        Main.clearTerminal();
        final String dna = Utils.readFile(dnaFile);
        if (proteinFile != null) {
            protein = Utils.readFile(proteinFile);
        }
        return new DNAAnalysis(new DNATools(dna), protein, aminoAcid);
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
        return new DNAAnalysis(null, null, aminoAcid);
    }
}
}
```

Μετά την αναδιαμόρφωση:

Η κλαση `ArgumentParser`:

```
@Command(name = "DNAnalyzer", mixinStandardHelpOptions = true, description
= "A program to analyze DNA sequences.")
public class ArgumentParser {
    public static class Options {
        @Option(names = {"--gui"}, description = "Start in GUI mode")
        Boolean startGUI = false;

        @Option(names = {"--amino"}, description = "The amino acid
representing the start of a gene.")
        String aminoAcid;

        @Option(names = {"--min"}, description = "The minimum count of the
reading frame.")
        int minCount = 0;

        @Option(names = {"--max"}, description = "The maximum count of the
reading frame.")
        int maxCount = 0;

        @Option(names = {"--dna"}, description = "The FASTA file to be
analyzed.")
        File dnaFile = new
File("C:\\Users\\Archontis\\Desktop\\test.fasta");

        @Option(names = {"--find"}, description = "The DNA sequence to be
found within the FASTA file.")
        File proteinFile;
```

```
        @Option(names = {"--reverse", "-r"}, description = "Reverse the DNA
sequence before processing.")
        boolean reverse;
    }


    public Options parse(String[] arguments) {
        Options options = new Options();
        CommandLine commandLine = new CommandLine(options);
        commandLine.parse(arguments);
        return options;
    }
}
```

Η κλάση **Terminal**:

```
public class Terminal {
    private static final String CLEAR_COMMAND_WINDOWS = "cmd /c cls";
    private static final String CLEAR_COMMAND_MAC_LINUX =
"\u001b[H\u001b[2J";
    private static Terminal instance;
    private String operatingSystem;
    private ResultPrinter resultPrinter;

    private Terminal() {
        this.operatingSystem = System.getProperty("os.name");
        this.resultPrinter = new ResultPrinter();
    }

    public static Terminal getInstance() {
        if (instance == null)
            instance = new Terminal();
        return instance;
    }

    public void printError(String message) {
        System.out.println("DNAnalyzer encountered a fatal Error. The
program will now close..." + System.lineSeparator());
        System.out.println("ERROR MESSAGE: " + message +
System.lineSeparator());
        System.exit(0);
    }

    public void clear() throws IOException, InterruptedException {
        if(operatingSystem.contains("Windows"))
            clearForWindows();
        else
            clearForMacLinux();
    }

    public void printLine(String textToPrint) {
        System.out.println(textToPrint);
    }

    private void clearForWindows() throws IOException,
InterruptedException{
        Runtime.getRuntime().exec(CLEAR_COMMAND_WINDOWS).waitFor();
    }
```

```
    private void clearForMacLinux() {
        System.out.println(CLEAR_COMMAND_MAC_LINUX);
        System.out.flush();
    }

    public void printAnalysisResults(AnalysisResult results) {
        this.resultPrinter.setResult(results);

        this.resultPrinter.printProteinAnalysis();
        this.resultPrinter.printHighCoverageAreas();
        this.resultPrinter.printLongestProtein();
        this.resultPrinter.printReadingFrames();
        this.resultPrinter.printProteinSequenceResults();
        this.resultPrinter.printRandomDnaInfo();
    }
}
```

Η Κλάση **Terminal** χρησιμοποιεί τις κλάσεις **ResultPrinter** και **AnalysisResult** που αναφέρθηκαν σε προήγουμενη ενότητα.

Η κλάση **Controller**:

```
public class Controller implements Runnable{
    private ArgumentParser argumentParser;
    private ArgumentParser.Options options;
    private Terminal terminal;

    public Controller(String[] args) {
        argumentParser = new ArgumentParser();
        options = argumentParser.parse(args);
        terminal = Terminal.getInstance();
    }

    @Override
    public void run() {
        if(options.startGUI)
            startGUI();
        else
            startCLI();
    }

    private void startGUI() {
      // GUI CODE
    }

    private void startCLI() {
        // CLI CODE
    }

}
```

Η Κλάση **Main**:

```
import controller.Controller;

public class Main {
    public static void main(final String[] args) {
        Controller controller = new Controller(args);
```

```
            controller.run();
        }
}
```

4.3.6. Συμπεράσματα

Οι αλλαγές που έγιναν στην αρχική βάση κώδικα του έργου αντιμετώπισαν το ζήτημα της έλλειψης κατάλληλου χειρισμού και ενθυλάκωσης των arguments γραμμής εντολών. Το νέο πακέτο με την ονομασία controller, το οποίο περιέχει τις κλάσεις `ArgumentParser` και `Controller`, βελτίωσε την ευελιξία, τη συντηρησιμότητα και την οργάνωση του έργου. Το έργο είναι πλέον σε θέση να χειρίζεται τα ορίσματα της γραμμής εντολών με πιο αποτελεσματικό τρόπο και είναι καλύτερα εξοπλισμένο για να φιλοξενήσει νέα ορίσματα της γραμμής εντολών στο μέλλον.

Με αυτόν τον τρόπο, η κλάση `ArgumentParser` είναι υπεύθυνη για την ανάλυση των επιχειρημάτων και την παροχή πρόσβασης στις τιμές τους, ενώ η κλάση Terminal είναι υπεύθυνη για την εκτύπωση του μηνύματος στην κονσόλα και η κλάση Controller για την εκκίνηση του προγράμματος μας. Αυτός ο διαχωρισμός των αρμοδιοτήτων καθιστά τον κώδικα πιο συντηρήσιμο και testable, καθώς η κλάση `ArgumentParser` μπορεί εύκολα να ελεγχθεί μεμονωμένα, ενώ η κλάση `Terminal` μπορεί εύκολα να επαναχρησιμοποιηθεί σε άλλα μέρη της εφαρμογής.

Αξίζει επίσης να αναφερθεί ότι θα μπορούσαμε να χρησιμοποιήσουμε την κλάση `CommandLine` στην κλάση `Terminal` και να αναλύσουμε τα ορίσματα εκεί, αλλά σε αυτή την περίπτωση η κλάση `Terminal` θα είχε την ευθύνη της ανάλυσης των επιχειρημάτων και της εκτύπωσης του μηνύματος, γεγονός που θα καθιστούσε λιγότερο επαναχρησιμοποιήσιμη και δυσκολότερη στο Testing.

4.7. Πρόβλημα 6: Έλλειψη της αρχής της ενιαίας ευθύνης (SRP) στην κλάση DNAnalysis (1) (Αρχοντής Κωστής)

4.7.1. Δήλωση Προβλήματος

Η αρχική βάση κώδικα είχε μια κλάση που ονομαζόταν `DNAnalysis`, η οποία ήταν υπεύθυνη για πολλά διαφορετικά πράγματα. Αυτό καθιστούσε δύσκολη την κατανόηση και τη συντήρηση της κλάσης, καθώς είχε πολλές αρμοδιότητες.

4.7.2. Λύση

Για την αντιμετώπιση αυτού του προβλήματος, η προσδιορίσαμε πρώτα τις διάφορες αρμοδιότητες της κλάσης `DNAnalysis`. Διαπιστώνοντας ότι η κλάση ήταν υπεύθυνη για την επικύρωση του DNA, την αντικατάσταση του DNA, την αντιστροφή του DNA, τη δημιουργία λίστας πρωτεϊνών, την εκτύπωση πρωτεϊνών, την περιεκτικότητα σε GC, τον αριθμό νουκλεοτιδίων, τις περιοχές υψηλής κάλυψης, τη μεγαλύτερη πρωτεΐνη και την εξαγωγή κωδικωνίων. Για την εφαρμογή της SRP, δημιουργήσαμε ξεχωριστές κλάσεις για πολλές από τις λειτουργίες της κλάσης `DNAnalysis` και άλλες μέθοδοι της μεταφέρθηκαν σε άλλες κλάσεις.

4.7.3. Αλλαγές

- Οι μέθοδοι `isValid()`, reverse και `replaceDNA()` μεταφέρθηκαν στη νέα κλάση DNA.
- Η μέθοδος `getProteins` του `DNAnalysis` αφαιρεθηκε και η μέθοδος `ProteinFinder.`*`getProtein`*`(DNATools dna, String aminoAcid)` μεταφέρθηκε στην κλάση `DNA` και αναδιαμορφώθηκε ώστε να χρησιμοποιεί το αντικείμενο

της κλασσης **DNA** για να βρει τις προτείνες, και να επιστρέφει δεδομένα της κλάσης **`Protein`** που φτιάξαμε.

- Δημιουργήθηκε μια νέα κλάση με την ονομασία **`DNAToolkit`**, η οποία διαθέτει τη λειτουργικότητα για τις **`checkIfDnaIsRandom`**, **`replaceDnaSequence`** και **`findProteins`**. Αυτές οι μέθοδοι αντικαθιστούν τη λογική από τις αντίστοιχες μεθόδους της κλάσης **`DNAnalyzer`** και **`DNATools`**.
- Η κλάση **`ProteinFinder`** διατηρήθηκε και αναδιαμορφώθηκε ώστε να λειτουργεί με το νέο αντικείμενο **DNA**.
- Η κλάση **`BasePairCounter`** αντικαταστάθηκε με μια πιο συγκεκριμένη κλάση που ονομάζεται **`NucleotideCounter`** και η μέθοδος **`countBasePairs`** αφαιρέθηκε.
- Η κλάση Terminal θα χρησιμοποιεί την κλάση **`ResultsPrinter`** για να εκτυπώσει τα αποτελέσματα από το αντικείμενο **`AnalysisResults`**, αφού τα πάρει από την κλάση Analysis.

4.7.4. Οφέλη των Αλλαγών

1. Η κλάση **DNA** είναι υπεύθυνη για την επικύρωση του DNA, την αντικατάσταση του DNA, την αντιστροφή του DNA κλπ, γεγονός που διευκολύνει την κατανόηση και τη συντήρησή της.
2. Η κλάση **`DNAToolkit`** είναι υπεύθυνη για το reversal του DNA, την περιεκτικότητα σε GC, τον αριθμό νουκλεοτιδίων, τις περιοχές υψηλής κάλυψης, τη μεγαλύτερη πρωτεΐνη και την εξαγωγή κωδικονίων, γεγονός που διευκολύνει την κατανόηση και τη συντήρησή της.
3. Η κλάση **`ProteinFinder`** είναι υπεύθυνη για την εύρεση πρωτεϊνών, γεγονός που διευκολύνει την κατανόηση και τη συντήρηση.
4. Η κλάση **`NucleotideCounter`** είναι υπεύθυνη για την καταμέτρηση των νουκλεοτιδίων, γεγονός που καθιστά ευκολότερη την κατανόηση και τη συντήρηση.
5. Η κλάση **`Terminal`** είναι υπεύθυνη για την εκτύπωση των αποτελεσμάτων, γεγονός που καθιστά ευκολότερη την κατανόηση και τη συντήρηση.
6. Ο διαχωρισμός των ανησυχιών οδηγεί σε πιο επαναχρησιμοποιήσιμο και συντηρήσιμο κώδικα.
7. Ο νέος σχεδιασμός επιτρέπει ευκολότερη δοκιμή και αποσφαλμάτωση των επιμέρους στοιχείων.
8. Ο νέος σχεδιασμός επιτρέπει καλύτερη επεκτασιμότητα, καθώς νέα χαρακτηριστικά μπορούν να προστεθούν ή να τροποποιηθούν χωρίς να επηρεαστούν άλλα τμήματα της βάσης κώδικα.

4.7.5. Κώδικας

Πρίν την αναδιαμόρφωση:

Η Κλάση DNAnalyzer:

```
public record DNAAnalysis(DNATools dna, String protein, String aminoAcid) {
```

```
   public DNAAnalysis isValidDna() {
      dna.isValid();
      return this;
   }

   public DNAAnalysis replaceDNA(final String input, final String
replacement) {
      return new DNAAnalysis(dna.replace(input, replacement), protein,
aminoAcid);
   }

   public DNAAnalysis reverseDna() {
      return new DNAAnalysis(dna.reverse(), protein, aminoAcid);
   }

   // Create protein list
   // Output the proteins, GC content, and nucleotide cnt found in the DNA
   public DNAAnalysis printProteins(PrintStream out) {
      ofNullable(dna).map(DNATools::getDna).ifPresent(dna -> {
         Properties.printProteinList(getProteins(aminoAcid), aminoAcid,
out);

         out.println("\nGC-content (genome): " +
Properties.getGCContent(dna) + "\n");
         Properties.printNucleotideCount(dna, out);
      });
      return this;
   }

   // Outputs the high coverage regions of a DNA
   public DNAAnalysis printHighCoverageRegions(PrintStream out) {
      ofNullable(dna).map(DNATools::dna).ifPresent(dna -> {
         ProteinAnalysis.printHighCoverageRegions(getProteins(aminoAcid),
out);
      });
      return this;
   }

   // used as helper method for output-codons, used to generate reading
frames
   public void configureReadingFrames(final int minCount, final int
maxCount, PrintStream out) {
      final short READING_FRAME = 1;
      final String dna = this.dna.getDna();
      final ReadingFrames aap = new ReadingFrames(new CodonFrame(dna,
READING_FRAME, minCount, maxCount));
      out.print("\n");
      aap.printCodonCounts(out);
   }

   // used as helper method for output codons, handles protein decisions
   public void proteinSequence(PrintStream out) {
      final String dna = this.dna.getDna();

      if (protein != null) {
         final Pattern p = Pattern.compile(protein);
         final Matcher m = p.matcher(dna);
         if (m.find()) {
            out.println(
                  "\nProtein sequence found at index " + m.start() + " in
```

```
the DNA sequence.");
            } else {
               out.println("\nProtein sequence not found in the DNA
sequence.");
            }
         }
      }

      public DNAAnalysis outPutCodons(final int minCount, final int maxCount,
PrintStream out) {
         configureReadingFrames(minCount, maxCount, out);
         proteinSequence(out);

         return this;
      }

      public void printLongestProtein(PrintStream out) {
         ProteinAnalysis.printLongestProtein(getProteins(aminoAcid), out);
      }

      private List<String> getProteins(final String aminoAcid) {
         return ProteinFinder.getProtein(dna.getDna(), aminoAcid);
      }


      public static long[] countBasePairs(String dnaString) {
         return new BasePairCounter(dnaString)
               .countAdenine()
               .countThymine()
               .countGuanine()
               .countCytosine()
               .getCounts();
      }

      public static class BasePairIndex {

         public static final int ADENINE = 0;
         public static final int THYMINE = 1;
         public static final int GUANINE = 2;
         public static final int CYTOSINE = 3;
      }


      public static class AsciiInt {

         public static final int UPPERCASE_A = 65;
         public static final int UPPERCASE_T = 84;
         public static final int UPPERCASE_G = 71;
         public static final int UPPERCASE_C = 67;
         public static final int LOWERCASE_A = 97;
         public static final int LOWERCASE_T = 116;
         public static final int LOWERCASE_G = 103;
         public static final int LOWERCASE_C = 99;

      }
}
```

Η Κλάσση DNATools:

```
public record DNATools(String dna) {
   public void isValid() {
      if (!dna.matches("[atgc]+")) {
         throw new IllegalArgumentException("Invalid characters present in
DNA sequence.");
      }
   }
   public DNATools replace(final String input, final String replacement) {
      return new DNATools(this.dna.replace(input, replacement));
   }
   public DNATools reverse() {
      return new DNATools(new StringBuilder(dna).reverse().toString());
   }
   public String getDna() {
      return dna;
   }
}
```

Η Κλάσση ProteinFinder:

```
public class ProteinFinder {

   private ProteinFinder() {
   }

   public static List<String> getProtein(final String dna, final String
aminoAcid) {

      final List<String> proteinList = new ArrayList<>();
      final List<String> aminoAcidList = new
ArrayList<>(CodonDataUtils.getAminoAcid(aminoAcid));

      for (final String startCodon : aminoAcidList) {
         addProtein(dna, proteinList,
dna.indexOf(startCodon.toLowerCase()));
      }

      // if no proteins are found, return null
      if (proteinList.isEmpty()) {

         System.out.println("No proteins found");
         return Collections.emptyList();
      }


      return proteinList;
   }

   private static void addProtein(final String dna, final List<String>
proteinList, final int startIndex) {


      final List<String> stopCodonList = CodonDataConstants.STOP;
      int stopIndex;
      for (final String stopCodon : stopCodonList) {

         stopIndex = dna.indexOf(stopCodon.toLowerCase(), startIndex + 3);
```

```
            if ((startIndex != -1) && (stopIndex != -1)) {
                proteinList.add(dna.substring(startIndex, stopIndex +
3).toUpperCase());
                break;
            }

        }
    }

}
```

Μετά την Αναδιαμόρφωση:

Η Κλαση DNA:

```
public class DNA extends Sequence {

    public DNA(String sequence) {
        super(sequence);
    }

    public boolean isValid() {
        String sequence = super.getSequence();
        if(!sequence.matches("[atgc]+"))
            throw new IllegalArgumentException("Invalid characters present
in DNA sequence.");
        return true;
    }

    public void replaceSequence(String charToReplace, String replacement) {
        String newSequence = DNAToolkit.replaceDnaSequence(this,
charToReplace, replacement);
        super.setSequence(newSequence);
    }

    public void reverse() {
        String sequence = super.getSequence();
        String reversedDna = new
StringBuilder(sequence).reverse().toString();
        this.setSequence(reversedDna);
    }

    public List<Protein> findProteins(String aminoAcid) {
        return DNAToolkit.findProteins(this, aminoAcid);
    }

    public boolean isRandom() {
        return DNAToolkit.checkIfDnaIsRandom(this);
    }

    @Override
    public String toString() {
        return "DNA{" +
                "sequence=" + super.getSequence() + '\'' +
                '}';
    }
}
```

Η Κλαση DNAToolkit:

```
public class DNAToolkit {

    public static boolean checkIfDnaIsRandom(DNA dna) {
        NucleotideCounter counter = new NucleotideCounter(dna);

        long[] nucleotideCount = counter.count();

        Arrays.sort(nucleotideCount);

        // Only calculate 2 Percentages, as only the highest difference
(max - min) is
        // relevant
        int maxPercent = (int) (nucleotideCount[0] / dna.length() * 100);
        int minPercent = (int) (nucleotideCount[3] / dna.length() * 100);

        // If the percentage of each nucleotide is between 2% of one
another, then it is
        // random
        return Math.abs(maxPercent - minPercent) <= 2;
    }

    public static String replaceDnaSequence(DNA dna, String charToReplace,
String replacement) {
        return dna.getSequence()
                .replace(charToReplace, replacement);
    }

    public static List<Protein> findProteins(DNA dna, String aminoAcid) {
        return ProteinFinder.findProteins(dna, aminoAcid);
    }
}
```

Η Κλάση NucleotideCounter:

```
package analysis.analyzers;

import model.dna.DNA;

import java.util.Map;

public class NucleotideCounter {
    private DNA dna;

    public NucleotideCounter(DNA dna) {
        this.dna = dna;
    }

    public long[] count() {
        long adenine = countAdenine();
        long thymine = countThymine();
        long guanine = countGuanine();
        long cytosine = countCytosine();

        return new long[] {adenine, thymine, guanine, cytosine};
    }

    private long countAdenine() {
```

```
        String dnaSequence = dna.getSequence();
        return dnaSequence.chars()
                .filter(c -> (c == 'A' || c == 'a'))
                .count();
    }

    private long countThymine() {
        String dnaSequence = dna.getSequence();
        return dnaSequence.chars()
                .filter(c -> (c == 'T' || c == 't'))
                .count();
    }

    private long countGuanine() {
        String dnaSequence = dna.getSequence();
        return dnaSequence.chars()
                .filter(c -> (c == 'G' || c == 'g'))
                .count();
    }

    private long countCytosine() {
        String dnaSequence = dna.getSequence();
        return dnaSequence.chars()
                .filter(c -> (c == 'C' || c == 'c'))
                .count();
    }

}
```

Η Κλάση ProteinFinder:

```
public class ProteinFinder {
    private static final List<String> STOP = List.of("TAA", "TAG", "TGA");

    private ProteinFinder() {
    }

    public static List<Protein> findProteins(DNA dna, String aminoAcid) {

        final List<Protein> proteinList = new ArrayList<>();
        final List<String> aminoAcidList = new
ArrayList<>(CodonDataUtils.getAminoAcid(aminoAcid));

        // Outer loop loops through the start codons for the amino acids
that the user
        // entered.
        // store the start index
        for (final String startCodon : aminoAcidList) {
            addProtein(dna, proteinList,
dna.getSequence().indexOf(startCodon.toLowerCase()));
        }

        // if no proteins are found, return null
        if (proteinList.isEmpty()) {
            // Return null if no protein found in DNA sequence
            System.out.println("No proteins found");
            return Collections.emptyList();
        }
```

```
        // Return list of proteins found in the DNA sequence
        return proteinList;
    }

    public static void addProtein(DNA dna, Collection<Protein> proteinList,
int startIndex) {

        // Inner loop that loops through the stop that the user entered.
        // store the stopIndex
        // if index is not -1 then store the substring of dna with start
and stop index
        // in the protein list
        final List<String> stopCodonList = CodonDataConstants.STOP;
        int stopIndex;
        for (final String stopCodon : stopCodonList) {

            stopIndex = dna.getSequence().indexOf(stopCodon.toLowerCase(),
startIndex + 3);

            if ((startIndex != -1) && (stopIndex != -1)) {
                String proteinSequence =
dna.getSequence().substring(startIndex, stopIndex + 3).toUpperCase();
                proteinList.add(new Protein(proteinSequence));
                break;
            }

        }
    }

}
```

4.8. Πρόβλημα 7: Έλλειψη της αρχής της ενιαίας ευθύνης (SRP) στην κλάση DNAnalyzer (2) (Αρχοντής Κωστής)

4.8.1. Δήλωση Προβλήματος

Όπως είδαμε και παραπάνω αρχική βάση κώδικα είχε μία κλάση με όνομα **DNAnalyzer**, η οποία ήταν υπεύθυνη για το χειρισμό και την ανάλυση αλληλουχιών DNA και πρωτεϊνών. Ωστόσο, η κλάση είχε πολλές αρμοδιότητες, γεγονός που την καθιστούσε δύσκολη στην κατανόηση και τη συντήρησή της.

Η κλάση **DNAnalyzer** ήταν υπεύθυνη για πολλές διαφορετικές εργασίες, όπως επικύρωση DNA, αντικατάσταση DNA κλπ. Αυτό καθιστούσε δύσκολη την κατανόηση και τη συντήρηση της κλάσης, καθώς είχε πολλές αρμοδιότητες.

4.8.2. Λύση

Για την αντιμετώπιση αυτού του προβλήματος, δημιουργήσαμε μια νέα κλάση με το όνομα **Analyzer**, η οποία χειρίζεται την ανάλυση αλληλουχιών DNA και πρωτεϊνών χρησιμοποιώντας ένα αντικείμενο **Analysis**, τις κλάσεις **ProteinAnalyzer** και **NucleotideCounter**.

Επιπλέον, δημιουργήσαμε επίσης μια νέα κλάση με όνομα **Analysis** η οποία χρησιμοποιείται για την αποθήκευση των δεδομένων που χρησιμοποιούνται στην ανάλυση.

Η κλάση **Analyzer** χρησιμοποιεί το αντικείμενο **Analysis** για την αποθήκευση των δεδομένων και χρησιμοποιεί τις κλάσεις **ProteinAnalyzer** και **NucleotideCounter** για την εκτέλεση της

ανάλυσης. Η κλάση **Analyzer** έχει επίσης μια αναφορά σε ένα αντικείμενο result (**AnalysisResult**), το οποίο αποθηκεύει τα αποτελέσματα της ανάλυσης. Μετά την ολοκλήρωση της ανάλυσης, το αντικείμενο result μεταβιβάζεται ως παράμετρος στη μέθοδο **printAnalysisResults** της κλάσης **Terminal**, όπου εκτυπώνονται τα αποτελέσματα.

4.8.3. Αλλαγές

1. Δημιουργήσαμε μια νέα κλάση με όνομα **Analyzer** η οποία χειρίζεται την ανάλυση αλληλουχιών DNA και πρωτεϊνών χρησιμοποιώντας ένα αντικείμενο **Analysis**, τις κλάσεις **ProteinAnalyzer** και **NucleotideCounter**.
2. Δημιουργήσαμε μια νέα κλάση με όνομα **Analysis** η οποία χρησιμοποιείται για την αποθήκευση των δεδομένων που χρησιμοποιούνται στην ανάλυση.
3. Αντικαταστήσαμε την κλάση **DNAnalysis** με την κλάση **Analyzer**, η οποία χρησιμοποιεί το αντικείμενο **Analysis**, τις κλάσεις **ProteinAnalyzer** και **NucleotideCounter** για την εκτέλεση της ανάλυσης.
4. Δημιουργήσαμε ένα αντικείμενο result στην κλάση **Analyzer** για την αποθήκευση των αποτελεσμάτων της ανάλυσης.
5. Μετά την ολοκλήρωση της ανάλυσης, το αντικείμενο result περνά ως παράμετρος στη μέθοδο **printAnalysisResults** της κλάσης **Terminal**, όπου εκτυπώνονται τα αποτελέσματα.
6. Μεταφέρθηκε η λογική για τον υπολογισμό του περιεχομένου GC και του μήκους της πρωτεΐνης στην κλάση **Protein**, βελτιώνοντας την ενθυλάκωση και τη συντηρησιμότητα του κώδικα.
7. Μεταφέρθηκε η μέθοδος **configureReadingFrames** στην κλάση **Analyzer**, αναδιαμορφώνοντας τον κώδικα ώστε να χρησιμοποιεί τα συγκεκριμένα αντικείμενα που δημιουργήσαμε.
8. Μεταφέρθηκε η μέθοδος **proteinSequence** στην κλάση Analyzer, αναδιαμορφώνοντας τον κώδικα ώστε να χρησιμοποιεί τα συγκεκριμένα αντικείμενα που δημιουργήσαμε.
9. Τοποθετήθηκε η ιδιωτική μέθοδος **init()** στον κατασκευαστή, προκειμένου να υπάρχει ένας αμυντικός προγραμματιστικός τρόπος αντιμετώπισης του μη έγκυρου DNA.
10. Η κλάση Analysis έχει μια δημόσια μέθοδο **begin()** η οποία χειρίζεται την αντικατάσταση και την αντιστροφή του DNA.
11. Η κλάση Analysis έχει μέθοδους όπως η **analyzeProteins()** που χρησιμοποιούν τις νέες κλάσεις που δημιουργήσαμε για να χειριστεί την ανάλυση και αποθηκεύει τα αποτελέσματα στο αντικείμενο **AnalysisResult**.
12. Η κλάση **Controller** χρησιμοποιεί τώρα την κλάση **Analysis** για να ξεκινήσει την ανάλυση και την κλάση **Analyzer** για να παράγει αποτελέσματα.
13. Η κλάση **Terminal** χρησιμοποιεί τώρα την κλάση **ResultsPrinter** για την εκτύπωση των αποτελεσμάτων.

4.8.4. Οφέλη των Αλλαγών

1. Οι νέες κλάσεις **Analyzer** και **Analysis** βελτιώνουν την οργάνωση και την κατανόηση του κώδικα και τον καθιστούν πιο σπονδυλωτό και εύκολο στη συντήρησή του.
2. Η χρήση του αντικειμένου **Analysis** και των κλάσεων **ProteinAnalyzer** και **NucleotideCounter** βελτιώνει το διαχωρισμό των προβλημάτων, καθιστώντας τον κώδικα πιο σπονδυλωτό και εύκολα κατανοητό.
3. Η χρήση ενός αντικειμένου αποτελέσματος και η παράδοσή του ως παραμέτρου στη μέθοδο **printAnalysisResults** της κλάσης **Terminal** βελτιώνει την επεκτασιμότητα

του κώδικα, καθιστώντας ευκολότερη την προσθήκη νέων χαρακτηριστικών και λειτουργιών στο μέλλον.

4. Η ενθυλάκωση των δεδομένων και της λογικής σε συγκεκριμένες κλάσεις βελτιώνει τη συντηρησιμότητα του κώδικα.

4.8.5. Κώδικας

Πρίν την αναδιαμόρφωση:

`DNAnalysis` Class:

```
public record DNAAnalysis(DNATools dna, String protein, String aminoAcid) {

   public DNAAnalysis isValidDna() {
      dna.isValid();
      return this;
   }

   public DNAAnalysis replaceDNA(final String input, final String
replacement) {
      return new DNAAnalysis(dna.replace(input, replacement), protein,
aminoAcid);
   }

   public DNAAnalysis reverseDna() {
      return new DNAAnalysis(dna.reverse(), protein, aminoAcid);
   }

      public DNAAnalysis printProteins(PrintStream out) {
      ofNullable(dna).map(DNATools::getDna).ifPresent(dna -> {
         Properties.printProteinList(getProteins(aminoAcid), aminoAcid,
out);

         out.println("\nGC-content (genome): " +
Properties.getGCContent(dna) + "\n");
         Properties.printNucleotideCount(dna, out);
      });
      return this;
   }

   public DNAAnalysis printHighCoverageRegions(PrintStream out) {
      ofNullable(dna).map(DNATools::dna).ifPresent(dna -> {
         ProteinAnalysis.printHighCoverageRegions(getProteins(aminoAcid),
out);
      });
      return this;
   }

   public void configureReadingFrames(final int minCount, final int
maxCount, PrintStream out) {
      final short READING_FRAME = 1;
      final String dna = this.dna.getDna();
      final ReadingFrames aap = new ReadingFrames(new CodonFrame(dna,
READING_FRAME, minCount, maxCount));
      out.print("\n");
      aap.printCodonCounts(out);
   }

     public void proteinSequence(PrintStream out) {
```

```
      final String dna = this.dna.getDna();

      if (protein != null) {
         final Pattern p = Pattern.compile(protein);
         final Matcher m = p.matcher(dna);
         if (m.find()) {
            out.println(
                  "\nProtein sequence found at index " + m.start() + " in
the DNA sequence.");
         } else {
            out.println("\nProtein sequence not found in the DNA
sequence.");
         }
      }
   }


   public DNAAnalysis outPutCodons(final int minCount, final int maxCount,
PrintStream out) {
      configureReadingFrames(minCount, maxCount, out);
      proteinSequence(out);

      return this;
   }

   public void printLongestProtein(PrintStream out) {
      ProteinAnalysis.printLongestProtein(getProteins(aminoAcid), out);
   }

   private List<String> getProteins(final String aminoAcid) {
      return ProteinFinder.getProtein(dna.getDna(), aminoAcid);
   }


   public static long[] countBasePairs(String dnaString) {
      return new BasePairCounter(dnaString)
            .countAdenine()
            .countThymine()
            .countGuanine()
            .countCytosine()
            .getCounts();
   }


   public static class BasePairIndex {

      public static final int ADENINE = 0;
      public static final int THYMINE = 1;
      public static final int GUANINE = 2;
      public static final int CYTOSINE = 3;
   }


   public static class AsciiInt {

      public static final int UPPERCASE_A = 65;
      public static final int UPPERCASE_T = 84;
      public static final int UPPERCASE_G = 71;
      public static final int UPPERCASE_C = 67;
      public static final int LOWERCASE_A = 97;
```

```
        public static final int LOWERCASE_T = 116;
        public static final int LOWERCASE_G = 103;
        public static final int LOWERCASE_C = 99;

    }
}
```

CmdArgs Class:

```
@Command(name = "DNAnalyzer", mixinStandardHelpOptions = true, description
= "A program to analyze DNA sequences.")
public class CmdArgs implements Runnable {
    @Option(names = { "--gui" }, description = "Start in GUI mode")
    Boolean startGUI = false;

    @Option(names = { "--amino" }, description = "The amino acid
representing the start of a gene.")
    String aminoAcid;

    @Option(names = { "--min" }, description = "The minimum count of the
reading frame.")
    int minCount = 0;

    @Option(names = { "--max" }, description = "The maximum count of the
reading frame.")
    int maxCount = 0;

    @Parameters(paramLabel = "DNA", description = "The FASTA file to be
analyzed.")
    File dnaFile;

    @Option(names = { "--find" }, description = "The DNA sequence to be
found within the FASTA file.")
    File proteinFile;

    @Option(names = { "--reverse", "-r" }, description = "Reverse the DNA
sequence before processing.")
    boolean reverse;


    @Override
    public void run() {
        if (startGUI == true) {
            String[] args = new String[0];
            DNAnalyzerGUI.launchIt(args);
        } else {
            DNAAnalysis dnaAnalyzer = dnaAnalyzer(aminoAcid)
                    .isValidDna()
                    .replaceDNA("u", "t");

            if (reverse == true) {
                dnaAnalyzer = dnaAnalyzer.reverseDna();
            }

            dnaAnalyzer
                    .printProteins(System.out)
                    .printHighCoverageRegions(System.out)
                    .outPutCodons(minCount, maxCount, System.out)
                    .printLongestProtein(System.out);

            if (Properties.isRandomDNA(dnaAnalyzer.dna().getDna())) {
```

```
            System.out.println("\n" + dnaFile.getName() + " has been
detected to be random.");
         }
      }
   }

   private DNAAnalysis dnaAnalyzer(final String aminoAcid) {
      try {
         String protein = null;
         Main.clearTerminal();
         final String dna = Utils.readFile(dnaFile);
         if (proteinFile != null) {
            protein = Utils.readFile(proteinFile);
         }
         return new DNAAnalysis(new DNATools(dna), protein, aminoAcid);
      } catch (IOException | InterruptedException e) {
         e.printStackTrace();
         return new DNAAnalysis(null, null, aminoAcid);
      }
   }
}
```

Μετά την Αναδιαμόρφωση:

`Analysis` Class:

```
public class Analysis {
    private DNA dna;
    private String aminoAcid;
    private String protein;

    public Analysis(DNA dna, String aminoAcid, String protein) {
        this.dna = dna;
        this.aminoAcid = aminoAcid;
        this.protein = protein;
    }

    public DNA getDna() {
        return dna;
    }

    public String getAminoAcid() {
        return aminoAcid;
    }

    public void setAminoAcid(String aminoAcid) {
        this.aminoAcid = aminoAcid;
    }

    public String getProtein() {
        return protein;
    }

    public void setProtein(String protein) {
        this.protein = protein;
    }
}
```

`Analyzer` Class:

```
public class Analyzer {
    private Analysis analysis;
    private ProteinAnalyzer proteinAnalyzer;
    private NucleotideCounter nucleotideCounter;
    private AnalysisResult result;

    public Analyzer(Analysis analysis) {
        this.analysis = analysis;
        this.result = new AnalysisResult(analysis.getAminoAcid());
        init(analysis);
    }

    private void init(Analysis analysis) {
        String aminoAcid = analysis.getAminoAcid();
        DNA dna = analysis.getDna();

        if (!dna.isValid())
            throw new IllegalArgumentException("Invalid characters present
in DNA sequence.");

        List<Protein> dnaProteins = dna.findProteins(aminoAcid);

        this.proteinAnalyzer = new ProteinAnalyzer(dnaProteins);
        this.nucleotideCounter = new NucleotideCounter(dna);
    }

    public void begin(boolean reverseIsSelected) {
        DNA dna = analysis.getDna();
        dna.replaceSequence("u", "t");

        if (reverseIsSelected)
            dna.reverse();
    }

    public void analyzeProteins() {
        DNA dna = analysis.getDna();

        String aminoAcid = analysis.getAminoAcid();
        this.result.setDnaProteins(dna.findProteins(aminoAcid));

        double gcContent = dna.calculateGCContent();
        this.result.setGcContent(gcContent);

        // Find Nucleotide Count
        long[] nucleotideCount = nucleotideCounter.count();
        // Put the results to a map
        Map<String, Long> resultsMap = new HashMap<>();
        resultsMap.put("A", nucleotideCount[0]);
        resultsMap.put("T", nucleotideCount[2]);
        resultsMap.put("G", nucleotideCount[3]);
        resultsMap.put("C", nucleotideCount[4]);

        this.result.setNucleotideCountMap(resultsMap);
    }

    public void searchForHighCoverageRegions() {
        Map<String, Integer> highCoverageRegions =
                proteinAnalyzer.findHighCoverageRegions();

        this.result.setHighCoverageRegions(highCoverageRegions);
```

```
    }

    public void findLongestProtein() {
        String aminoAcid = analysis.getAminoAcid();
        List<Protein> proteins = analysis.getDna()
                .findProteins(aminoAcid);

        Protein longestProtein =
proteinAnalyzer.findLongestProtein(proteins);
        this.result.setLongestProtein(longestProtein);
    }

    public void configureReadingFrames(int minCount, int maxCount) {
        final short READING_FRAME = 1;

        DNA dna = analysis.getDna();
        CodonFrame codonFrame = new CodonFrame(dna, READING_FRAME,
minCount, maxCount);
        ReadingFrames readingFrames = new ReadingFrames(codonFrame);

        this.result.setReadingFrames(readingFrames);
    }

    public void analyzeProteinSequence() {
        if(analysis.getProtein() == null)
            throw new IllegalArgumentException("Protein cannot be null.");

        final Pattern pattern = Pattern.compile(analysis.getProtein());
        final Matcher matcher =
pattern.matcher(analysis.getDna().getSequence());

        if (matcher.find())
            this.result.setProteinSequenceIndex(matcher.start());

        this.result.setProteinSequenceIndex(matcher.start());
    }

    public void checkForRandomDNA() {
        if (analysis.getDna().isRandom())
            this.result.setDnaIsRandom(true);
    }

    public Analysis getAnalysis() {
        return analysis;
    }
    public void setAnalysis(Analysis analysis) {
        this.analysis = analysis;
    }
    public AnalysisResult getResult() {
        return result;
    }
}
```

Controller Class: (Μας ενδιαφέρει κυρίως η **startCLI()**)

```
public class Controller implements Runnable{
    private ArgumentParser argumentParser;
    private ArgumentParser.Options options;
    private Terminal terminal;
```

```
    public Controller(String[] args) {
        argumentParser = new ArgumentParser();
        options = argumentParser.parse(args);
        terminal = Terminal.getInstance();
    }

    @Override
    public void run() {
        if(options.startGUI)
            startGUI();
        else
            startCLI();
    }

    private void startGUI() {
//        String[] args = new String[0];
//        DNAnalyzerGUI.launchIt(args);
    }

    private void startCLI() {
        terminal.printLine("DNAnalyzer Started Running...");

        Analysis analysis = startAnalysis();
        Analyzer analyzer = new Analyzer(analysis);

        analyzer.begin(options.reverse);
        analyzer.analyzeProteins();

        analyzer.searchForHighCoverageRegions();
        analyzer.configureReadingFrames(options.minCount,
options.maxCount);
        analyzer.analyzeProteinSequence();
        analyzer.findLongestProtein();
        analyzer.checkForRandomDNA();

        AnalysisResult analysisResult =
                analyzer.getResult();

        terminal.printAnalysisResults(analysisResult);
    }

    private Analysis startAnalysis() {
        try {
            String protein = null;
            terminal.clear();

            FastaReader fastaReader = new FastaReader();
            DNA dna = fastaReader.read(new
File("C:\\Users\\Archontis\\Desktop"));

            if (options.proteinFile != null)
                protein = fastaReader.read(options.proteinFile)
                        .getSequence();

            Analysis analysis = new Analysis(dna, options.aminoAcid,
protein);
            return analysis;
        } catch (IOException | InterruptedException e) {
```

```
                Terminal.getInstance()
            .printError("There was an error reading the files...");

            }
        }
}
```

4.8.6. Συμπεράσματα

Συμπερασματικά, καταφέραμε να αντιμετωπίσουμε το πρόβλημα της έλλειψης διαχωρισμού των ανησυχιών στην κλάση **DNAnalyzer** εφαρμόζοντας την αρχή της ενιαίας ευθύνης και δημιουργώντας ξεχωριστές κλάσεις για κάθε ευθύνη.

Οι νέες κλάσεις **Analyzer** και **Analysis** βελτίωσαν την οργάνωση και την κατανόηση του κώδικα, καθιστώντας τον πιο σπονδυλωτό και εύκολο στη συντήρησή του. Χρησιμοποιώντας το αντικείμενο **Analysis** και τις κλάσεις **ProteinAnalyzer** και **NucleotideCounter**, βελτιώσαμε το διαχωρισμό των ανησυχιών και την ενθυλάκωση των δεδομένων και της λογικής σε συγκεκριμένες κλάσεις, γεγονός που με τη σειρά του βελτίωσε τη συντηρησιμότητα του κώδικα.

Η χρήση ενός αντικειμένου αποτελέσματος και η μεταβίβασή του ως παραμέτρου στη μέθοδο **printAnalysisResults** της κλάσης **Terminal** βελτίωσε επίσης την επεκτασιμότητα του κώδικα, διευκολύνοντας την προσθήκη νέων χαρακτηριστικών και λειτουργιών στο μέλλον.

4.9. Νέο File Structure / Τεκμηρίωση (Μαζί)

4.9.1. Νέο File Structure

Κατά την εφαρμογή των αλλαγών, η ομάδα αποφάσισε να αναδιοργανώσει τη δομή των αρχείων του έργου για να γίνει πιο οργανωμένη και εύκολη στην πλοήγηση.

Η δομή πριν το Refactoring είχε της εξής μορφή:

- DNAnalyzer
  - core
    - DNAAnalysis.java
    - Properties.java
  - data
    - aminoAcid
      - AminoAcid.java
      - AminoAcidFactory.java
    - codon
      - CodonDataConstants.java
      - CodonDataUtils.java
      - CodonFrame.java
  - ui
    - cli
      - CmdArgs.java
    - gui
      - DNAnalyzerGUI.java
      - DNAnalyzerGUIFXMLController.java

        - utils
            - core
                - BasePairCounter.java
                - DNATools.java
                - ReadingFreames.java
                - Utils.java
            - protein
                - ProteinAnalysis.java
                - ProteinFinder.java
            - traits
                - Traits.java

Η νέα δομή είναι η εξής:

- analysis
    - analyzers
        - Analyzer.java
        - NucleotideCounter.java
        - ProteinAnalyzer.java
    - Analysis.java
    - AnalysisResult.java
- controller
    - ArgumentParser.java
    - Controller.java
- io
    - cli
        - ResultPrinter.java
        - Terminal.java
    - file
        - FastaReader.java
        - FileReader.java
- model
    - aminoAcid
        - AminoAcid.java
        - AminoAcidFactory.java
    - codon
        - CodonDataConstants.java
        - CodonDataUtils.java
        - CodonFrame.java
    - dna
        - DNA.java
        - DNAToolkit.java
    - protein
        - Protein.java
        - ProteinFinder.java
    - ReadingFrames.java
    - Sequence.java
- Main.java

Η νέα δομή των αρχείων του έργου έχει βελτιώσει σημαντικά την οργάνωση και την κατανόηση του κώδικα. Με την ομαδοποίηση των κλάσεων που σχετίζονται με μια συγκεκριμένη λειτουργικότητα στο ίδιο πακέτο, έγινε ευκολότερη η πλοήγηση και η κατανόηση της βάσης κώδικα.

Επιπλέον, η νέα δομή των αρχείων κατέστησε πιο αρθρωτή και πιο εύκολη την προσθήκη νέων χαρακτηριστικών και λειτουργιών στο μέλλον. Με τη διατήρηση των κλάσεων που σχετίζονται με μια συγκεκριμένη λειτουργικότητα μαζί, γίνεται ευκολότερος ο εντοπισμός των σημείων όπου πρέπει να προστεθεί νέος κώδικας, μειώνοντας τις πιθανότητες εισαγωγής σφαλμάτων και καθιστώντας τη διαδικασία ανάπτυξης πιο αποτελεσματική.

Ακόμη, η νέα δομή πακέτων βοήθησε επίσης στη βελτίωση της αναγνωσιμότητας του κώδικα. Κρατώντας τις κλάσεις με παρόμοιες αρμοδιότητες μαζί, έγινε ευκολότερη η κατανόηση του σκοπού κάθε κλάσης, καθιστώντας ευκολότερη την κατανόηση της συνολικής αρχιτεκτονικής του έργου και του τρόπου με τον οποίο οι διάφορες κλάσεις αλληλεπιδρούν μεταξύ τους.

4.9.2. Τεκμηρίωση

Δημιουργήσαμε επίσης την τεκμηρίωση για κάθε κλάση (στα αγγλικά), η οποία βρίσκεται σε έναν φάκελο που ονομάζεται "documentation" και μέσα στον φάκελο "docs" του έργου. Αυτή η τεκμηρίωση παρέχει περαιτέρω πληροφορίες σχετικά με τον σκοπό και τη χρήση κάθε κλάσης, καθιστώντας τον κώδικα ακόμη πιο κατανοητό και εύκολο στην πλοήγηση.

4.9.3. Συμπεράσματα

Συνοπτικά, η νέα δομή αρχείων βελτίωσε σημαντικά την οργάνωση και την κατανόηση του κώδικα, καθιστώντας τον πιο σπονδυλωτό, πιο εύκολο στην πλοήγηση και πιο αποτελεσματικό στην προσθήκη νέων χαρακτηριστικών και λειτουργιών στο μέλλον. Η τεκμηρίωση των κλάσεων συμβάλλει επίσης σε αυτό, παρέχοντας σαφείς και περιεκτικές πληροφορίες σχετικά με τις κλάσεις στο έργο. Επιπλέον, έχει επίσης βελτιωθεί η αναγνωσιμότητα του κώδικα και την κατανόηση της αρχιτεκτονικής του έργου.

4.10. Μετρικές πρίν και μέτα το Refactoring

4.10.1. Εισαγωγή

Μετά την αναδόμηση, το έργο έχει αναδομηθεί με τρόπο που το καθιστά πιο οργανωμένο, συντηρήσιμο και επεκτάσιμο. Η αρχική βάση κώδικα δεν ακολουθούσε τον αντικειμενοστραφή προγραμματισμό (OOP) και τις αρχές σχεδιασμού. Το πρόγραμμα ήταν δύσκολο να κατανοηθεί και να συντηρηθεί και ήταν δύσκολο να προστεθούν νέα χαρακτηριστικά. Το πρόγραμμα είχε επίσης υψηλό βαθμό σύζευξης μεταξύ των κλάσεων, γεγονός που καθιστούσε δύσκολη τη δοκιμή και την αποσφαλμάτωση. Ένας από τους βασικούς τομείς εστίασης κατά τη διαδικασία αναδιαμόρφωσης ήταν ο διαχωρισμός των ανησυχιών και η εφαρμογή των αρχών SOLID.

Η λύση ήταν η αναδιαμόρφωση της βάσης κώδικα ώστε να ακολουθεί τις αρχές της OOP και του καλού σχεδιασμού. Αυτό περιελάμβανε τη δημιουργία νέων κλάσεων, τη μετακίνηση μεθόδων μεταξύ κλάσεων και την αναδιάρθρωση του προγράμματος ώστε να γίνει πιο οργανωμένο, συντηρήσιμο και επεκτάσιμο.

Ανακεφαλαιώνοντας, έγιναν οι ακόλουθες αλλαγές:

- Δημιουργήθηκαν νέες κλάσεις για τη διαχείριση συγκεκριμένων αρμοδιοτήτων, όπως οι κλάσεις `Analyzer`, `AnalysisResult` και `ResultPrinter`. Αυτό βελτίωσε την ενθυλάκωση των δεδομένων και της λογικής, καθιστώντας το πρόγραμμα πιο συντηρήσιμο και πιο κατανοητό.
- Μετακινήσαμε μεθόδους και λογική μεταξύ κλάσεων για να βελτιωθεί τη συνοχή του προγράμματος. Αυτό έκανε το πρόγραμμα πιο οργανωμένο και πιο κατανοητό.
- Δημιούργησε μια νέα κλάση με όνομα `Controller` για να χειριστεί τη ροή του προγράμματος και να τη διαχωρίσει από την κλάση `CmdArgs`.
- Χρησιμοποιήσαμε τις κλάσεις `Analyzer`, `AnalysisResult`, `ResultPrinter` και στην κλάση `Terminal` για να χειριστούμε την ανάλυση και εκτύπωση των αποτελεσμάτων.
- Η κλάση DNAnalyzer έφυγε και αντικαταστάθηκε με την `Analyzer` που ακολουθεί την Αρχή της Ενιαίας Ευθύνης (Single Responsibility Principle - SRP) και να βελτιώσει τη συνοχή του προγράμματος.

4.10.2. Μετρικές

Οι ακόλουθοι πίνακες παρουσιάζουν τις μετρικές του έργου μεταξύ της τελευταίας έκδοσης και της αναθεωρημένης έκδοσης. Τα αποτελέσματα δείχνουν τις βελτιώσεις που έγιναν στην βάση κώδικα όσον αφορά τη συντηρησιμότητα, την αναγνωσιμότητα και την επεκτασιμότητα.

v2.1.1.

	DIT	NOCC	CBO	LCOM	WMC*	NOM	SIZE1
SUM	0	0	19	260	16.36905	61	511
AVERAGE	0	0	0.863636	11.81818	0.779478	2.772727	23.22727
MAX	0	0	4	190	2	20	138

Refactored Version:

	DIT	NOCC	CBO	LCOM	WMC	NOM	SIZE1
SUM	3	0	38	83	16.41666669	50	396
AVERAGE	0.13	0.08696	2.3913	17.9565	1.010232429	4.3478	28.3043
MAX	1	0	5	15	2	6	62

4.10.3. Συμπεράσματα

Είναι σημαντικό να σημειωθεί ότι κατά τη σύγκριση των μετρικών της αναδιαμορφωμένης έκδοσης με την αρχική έκδοση, μπορεί να φαίνεται ότι η αναδιαμορφωμένη έκδοση έχει χειρότερες μετρικές. Ωστόσο, αυτό δεν ισχύει. Η αύξηση σε μετρικές όπως οι DIT, CBO, LCOM, WMC, NOM και LOC είναι αποτέλεσμα του αυξημένου αριθμού κλάσεων και μεθόδων που προστέθηκαν στην αναδιαμορφωμένη έκδοση.

Αυτό ήταν αναμενόμενο, καθώς η διαδικασία refactoring περιελάμβανε τη δημιουργία νέων κλάσεων και μεθόδων για τη διαχείριση συγκεκριμένων αρμοδιοτήτων, καθιστώντας τον κώδικα πιο οργανωμένο και πιο κατανοητό.

Είναι επίσης σημαντικό να σημειωθεί ότι οι μετρικές δεν αποτελούν τη μοναδική ένδειξη της ποιότητας του κώδικα. Η αναδιαμορφωμένη έκδοση του έργου DNAnalyzer έχει αρκετές βελτιώσεις

που την καθιστούν καλύτερη έκδοση από την αρχική. Ορισμένες από αυτές τις βελτιώσεις είναι οι εξής:

- **Διαχωρισμός των ανησυχιών/ Εφαρμογή των αρχών SOLID**: Η αρχική έκδοση της κλάσης `DNAnalyzer` είχε μεθόδους που χρησιμοποιούνταν μόνο μέσα σε άλλες μεθόδους της κλάσης, γεγονός που καθιστούσε δύσκολη την κατανόηση και τη συντήρησή της. Η διαδικασία αναδόμησης εισήγαγε νέες κλάσεις για τη διαχείριση συγκεκριμένων αρμοδιοτήτων, καθιστώντας τον κώδικα πιο οργανωμένο και εύκολα κατανοητό.
- **Βελτιωμένη ενθυλάκωση/ Βελτίωση της αναγνωσιμότητας**: Η αρχική έκδοση της κλάσης `DNAnalyzer` είχε μεθόδους που χρησιμοποιούνταν μόνο μέσα σε άλλες μεθόδους της κλάσης, καθιστώντας την δύσκολη στην κατανόηση και τη συντήρησή της. Η διαδικασία αναδόμησης εισήγαγε νέες κλάσεις για να χειρίζονται συγκεκριμένες αρμοδιότητες, καθιστώντας τον κώδικα πιο οργανωμένο και εύκολα κατανοητό. Ακόμη η λογικη ανάλυσης και εκτύπωσης διαχωρίστηκε

Συμπερασματικά, η αναδιαμόρφωση της βάσης κώδικα βελτίωσε τη συνολική ποιότητα του προγράμματος, καθιστώντας το πιο οργανωμένο, συντηρήσιμο και επεκτάσιμο. Το πρόγραμμα είναι πλέον πιο σπονδυλωτό και πιο κατανοητό, γεγονός που διευκολύνει την προσθήκη νέων χαρακτηριστικών και την αποσφαλμάτωση. Η αύξηση των μετρήσεων μετά την αναδιαμόρφωση είναι φυσική συνέπεια των αλλαγών που έγιναν και δεν αντανακλά αρνητικά στην ποιότητα του προγράμματος.