ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΜΠΕΙΡΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΤΕΧΝΙΚΟΥ ΧΡΕΟΥΣ ΣΕ ΛΟΓΙΣΜΙΚΟ ΥΨΗΛΗΣ ΥΠΟΛΟΓΙΣΤΙΚΗΣ ΙΣΧΥΟΣ

Διπλωματική Εργασία

του

Νικολαΐδη Νικόλαου

Θεσσαλονίκη, Φεβρουάριος 2021

ΕΜΠΕΙΡΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΤΕΧΝΙΚΟΥ ΧΡΕΟΥΣ ΣΕ ΛΟΓΙΣΜΙΚΟ ΥΨΗΛΗΣ ΥΠΟΛΟΓΙΣΤΙΚΗΣ ΙΣΧΥΟΣ

Νικολαΐδης Νικόλαος

Πτυχίο Εφαρμοσμένης Πληροφορικής, ΠΑΜΑΚ, 2018

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Αμπατζόγλου Απόστολος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή 26/2/2021

Αμπατζόγλου Απόστολος	Χατζηγεωργίου Αλέξανδρος	Ξυνόγαλος Στυλιανός
.....	.....	.....

Νικολαΐδης Νικόλαος

.....

Περίληψη

Τα τελευταία χρόνια υπάρχει μεγάλη αύξηση του ενδιαφέροντος για τη διαχείρισης του τεχνικού χρέους, τόσο από τους ερευνητές, όσο και από την βιομηχανία. Το ενδιαφέρον αυτό πηγάζει από τη θετική συμβολή της σωστής διαχείρισης του τεχνικού χρέους στη συντήρηση του λογισμικού, αλλά και στις αρνητικές επιπτώσεις που μπορούν να προκύψουν αν δεν αποτελεί κομμάτι της ανάπτυξης λογισμικού. Ένας τομέας που δεν έχει ακόμη υιοθετήσει τεχνικές διαχείρισης τεχνικού χρέους, είναι οι ομάδες ανάπτυξης λογισμικού σε συστήματα υψηλής υπολογιστικής ισχύος. Σε αυτή την εργασία προσεγγίζεται η ποσοτικοποίηση και αποπληρωμή του τεχνικού χρέους σε τέτοιου είδους έργα λογισμικού. Επιπλέον, παρουσιάζεται το εργαλείο που αναπτύχθηκε για να υποβοηθήσει τους προγραμματιστές στην διαχείριση του τεχνικού χρέους. Επίσης, παρουσιάζεται η έρευνα που έγινε όσον αφορά την αλληλεπίδραση που έχει η χρήση εργαλείων βελτιστοποίησης ταχύτητας και φορητότητας στο τεχνικό χρέος, όπου δεν φάνηκε κάποια ιδιαίτερη αντιστάθμιση. Τέλος, επισημαίνονται τα αποτελέσματα της χρήσης του εργαλείου μας για την αποπληρωμή, σε τέσσερα βιομηχανικά λογισμικά υψηλής κλίμακας, όπου η μείωση του τεχνικού χρέους ήταν αισθητά μεγάλη.

Λέξεις Κλειδιά: συστήματα υψηλής υπολογιστικής ισχύος, διαχείριση τεχνικού χρέους, βελτιστοποίηση λογισμικού, εργαλείο διαχείρισης τεχνικού χρέους

Abstract

In recent years there has been a great increase in the interest of technical debt management by researchers and the industry. This is based on the positive influence that proper management can offer and the negative influence that can arise if it is not part of the software development. An area that technical debt management has not been adopted so far is high performance computing (HPC). In this thesis we introduce an approach to adjust technical debt quantification and repayment in such software projects. In addition, we present a tool developed to help HPC developers to manage the technical debt in an easy way. Moreover, we present a research on the effects of using performance and portability optimization tools on technical debt, where no important trade-offs have been identified. Finally, we present the results of applying the proposed tool for repayment, in four real-world HPC software applications, where the reduction of technical debt was significant.

Keywords: high performance computing, technical debt management, software optimization, technical debt management tool

Πρόλογος – Ευχαριστίες

Είναι πολύ σημαντικό να γνωρίζεις άτομα στην ζωή σου τα οποία πιστεύουν σε εσένα και σε βοηθούν να αναπτυχθείς. Νιώθω τυχερός που έχω την τύχη και δυνατότητα να συνεργάζομαι με δύο τέτοιους καθηγητές τον κ. Χατζηγεωργίου Αλέξανδρο και τον κ. Αμπατζόγλου Απόστολο.

Θα ήθελα να ευχαριστήσω τον Αμπατζόγλου Απόστολο για την βοήθεια και τις συμβουλές του κατά την διάρκεια της εκπόνησης αυτής της εργασίας. Καθώς και τον Χατζηγεωργίου Αλέξανδρο, ο οποίος είναι ο επιστημονικός υπεύθυνος του ευρωπαϊκού έργου EXA2PRO (exa2pro.eu), όπου και βασίζεται η εργασία αυτή. Επιπλέον, θα ήθελα να ευχαριστήσω όλους τους συνεργάτες μου από το εργαστήριο τεχνολογίας λογισμικού του ΠΑΜΑΚ που συνέβαλαν σε διάφορα μέρη της έρευνας και της τελικής δημοσίευσης στο συνέδριο Euromicro DSD/SEAA 2020.

Τέλος, θα ήθελα να ευχαριστήσω τους προγραμματιστές των έργων CO_2Capture, MetalWalls, LQCD, KKRnano, Pastix, QR-mumps, Rodinia, Parse, που μας παρείχαν πρόσβαση στον κώδικα τους καθώς και πολύτιμες πληροφορίες για την αξιολόγηση των προτεινόμενων αλλαγών.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Ο προγραμματισμός υψηλών επιδόσεων έχει σκοπό την ανάπτυξη εφαρμογών λογισμικού για επιστημονικούς σκοπούς, αλλά στην πληθώρα των περιπτώσεων, κατά την ανάπτυξη τους δεν γίνεται χρήση των πρακτικών τεχνολογίας λογισμικού. Βλέποντας τα οφέλη που έχουν οι πρακτικές τεχνολογίας λογισμικού, όταν χρησιμοποιούνται, είναι λογικό οι προγραμματιστές επιστημονικού λογισμικού να προσπαθούν σταδιακά να τις ενστερνιστούν. Μια από αυτές τις πρακτικές είναι η διαχείριση του τεχνικού χρέους.

Το τεχνικό χρέος από την εισαγωγή του στην κοινότητα το 1992, έχει προσελκύσει το ενδιαφέρον του ερευνητικού και του βιομηχανικού κόσμου. Παρ' όλο που υπάρχουν πολλοί ορισμοί για την μεταφορά του τεχνικού χρέους, όλοι έχουν ένα κοινό χαρακτηριστικό, την έλλειψη ποιότητας στον κώδικα και τις επιπτώσεις που επιφέρει αυτό. Γνωρίζοντας την σημαντικότητα του τεχνικού χρέους αναγνωρίζεται και η σημαντικότητα της διαχείρισης του. Για αυτόν το λόγο, πολλές έρευνες και εργαλεία έχουν αναπτυχθεί για την πιο αποτελεσματική διαχείριση του τεχνικού χρέους.

Το τεχνικό χρέος και η διαχείρισή του δεν έχει ακόμη πλήρως υιοθετηθεί στο πεδίο του προγραμματισμού υψηλών επιδόσεων και οι έρευνες προς αυτή την κατεύθυνση είναι περιορισμένες. Έχοντας υπόψη τις γλώσσες προγραμματισμού που χρησιμοποιούνται στον προγραμματισμό υψηλών επιδόσεων, φαίνεται η ανάγκη για μία προσαρμογή αρκετών εννοιών, τρόπου υπολογισμού και διαχείρισης του τεχνικού χρέους.

Το ερευνητικό έργο EXA2PRO έχει ως σκοπό την ανάπτυξη ενός περιβάλλοντος προγραμματισμού που θα επιτρέπει την εύκολη ανάπτυξη παράλληλων εφαρμογών σε συστήματα μεγάλης κλίμακας. Έχοντας υπόψη μας την συνεργασία με το Πανεπιστήμιο Μακεδονίας περιορίσαμε τα εξεταζόμενα έργα σε αυτά των πάροχων του EXA2PRO[1], λόγω των συγκεκριμένων ιδιοτήτων και της εύκολης επικοινωνίας με τους προγραμματιστές.

1.2 Σκοπός – Στόχοι

Η εργασία αυτή έχει ως σκοπό τον ορισμό του τρόπου διαχείρισης του τεχνικού χρέους και της αποτελεσματικότητας του. Πιο συγκεκριμένα, θα οριστεί ο τρόπος με τον

[1] https://exa2pro.eu/

οποίο θα γίνεται η ποσοτικοποίηση και η αποπληρωμή του τεχνικού χρέους, καθώς και το πόσο αποτελεσματικοί είναι οι τρόποι αποπληρωμής που παρουσιάζουμε σε συγκεκριμένα έργα. Επίσης, για την διαχείριση του τεχνικού χρέους, θα δημιουργηθεί και μια εφαρμογή από όπου οι δραστηριότητες της διαχείρισης θα γίνονται πιο εύκολα.

Επιπλέον, μέσω της ποσοτικοποίησης του τεχνικού χρέους θα μπορέσουμε να βρούμε τις επιδράσεις κάποιων υπερσύγχρονων τεχνολογιών που χρησιμοποιούνται σε προγράμματα υψηλής επίδοσης. Αυτές οι τεχνολογίες αφορούν την βελτιστοποίηση του χρόνου και της φορητότητας των εφαρμογών.

1.3 Διάρθρωση της μελέτης

Στο δεύτερο κεφάλαιο, πραγματοποιείται μια βιβλιογραφική επισκόπηση και παρουσιάζεται το θεωρητικό υπόβαθρο της έρευνας. Αναλύονται τα βασικά χαρακτηριστικά του τεχνικού χρέους και ο βαθμός με τον οποίο τα προγράμματα υψηλής επίδοσης χρησιμοποιούν πρακτικές της τεχνολογίας λογισμικού.

Στο τρίτο κεφάλαιο, ορίζεται ο τρόπος με τον οποίο γίνεται η ποσοτικοποίηση του τεχνικού χρέους. Συγκεκριμένα επικεντρωνόμαστε στο TX σε επίπεδο κώδικα, το οποίο πρέπει να υπολογιστεί εμπειρικά πάνω σε συγκεκριμένα έργα.

Στο τέταρτο κεφάλαιο, παρουσιάζεται η προσέγγιση μας για την αποπληρωμή του τεχνικού χρέους μέσω των αναδιαμορφώσεων Extract File / Module και Extract Procedure. Παρουσιάζονται οι αλγόριθμοι, με τους οποίους γίνεται η αυτόματη εύρεση ευκαιριών αναδιαμορφώσεων καθώς και οι τροποποιήσεις που έχουμε υποβάλλει σε αυτούς.

Στο πέμπτο κεφάλαιο, πραγματοποιείται μελέτη για την εύρεση των επιπτώσεων διάφορων έργων, όπου πραγματοποιήθηκαν σύγχρονες βελτιστοποιήσεις απόδοσης και φορητότητας, στο τεχνικό χρέος.

Στο έκτο κεφάλαιο, παρουσιάζεται το εργαλείο που αναπτύχθηκε για την διαχείριση του τεχνικού χρέους.

Τέλος, στο έβδομο κεφάλαιο, πραγματοποιείται η αξιολόγηση της χρήση του εργαλείου μας για την εύρεση της επίδρασης που έχει στην αποπληρωμή του τεχνικού χρέους σε πραγματικά έργα.

2 Θεωρητικό Υπόβαθρο

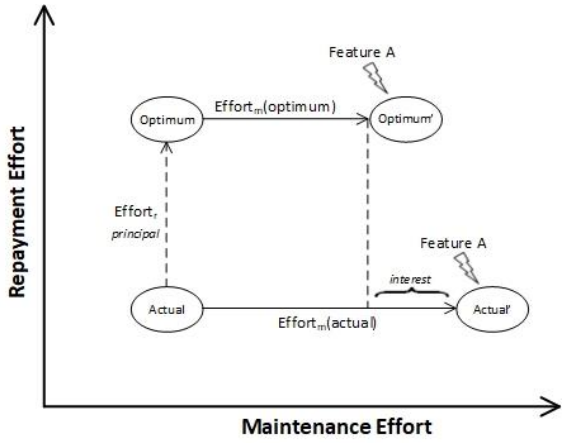
2.1 Τεχνικό Χρέος

Η πρώτη εμφάνιση της μεταφοράς του τεχνικού χρέους (ΤΧ) στην βιβλιογραφία της τεχνολογίας λογισμικού εμφανίζεται το 1992, από τον Ward Cunningham. Ο Cunningham χρησιμοποίησε αυτή την μεταφορά, για να δώσει έμφαση στις αντισταθμίσεις μεταξύ της παράδοσης κώδικα που βρίσκετε σε αρχικά στάδια, φέρνοντας τους όρους «χρέος» (debt) και «επιτόκιο» (interest) στην ανάπτυξη λογισμικού (Cunningham (1992)). Τις επόμενες δεκαετίες, η μεταφορά του τεχνικού χρέους έγινε πολύ γνωστή, προσελκύοντας το ενδιαφέρον της ερευνητικής κοινότητας, καθώς και της βιομηχανίας. Το κύριο σημείο ανάδειξης ήταν η ικανότητά του να λειτουργεί ως μέσο επικοινωνίας μεταξύ τεχνικών και διαχειριστικών φορέων, υποδηλώνοντας την ανάγκη βελτίωσης της ποιότητας του λογισμικού (Kruchten et al. (2012)).

2.1.1 Βασικές Έννοιες Τεχνικού Χρέους

Οι πυλώνες του ΤΧ είναι δύο έννοιες που οι τεχνολόγοι λογισμικού δανείστηκαν από την επιστήμη της οικονομίας: το κεφάλαιο (principal) και ο τόκος (interest). Μία εταιρία λογισμικού εξοικονομεί προσπάθεια, που θεωρείται ως το κεφάλαιο (του δανείου) με την παραγωγή ανώριμου λογισμικού (π.χ., σχεδιασμός ή κώδικας) και πληρώνει την «*ποινή*» για αυτά ως πρόσθετη προσπάθεια συντήρησης (τόκος).

Στην Εικόνα 2-1, απεικονίζεται η σχέση μεταξύ αυτών των δύο εννοιών, με βάση τη μελέτη των Chatzigeorgiou et al. (2015). Για οποιοδήποτε «Actual» («πραγματικό») σύστημα, ο άξονας y απεικονίζει το επίπεδο ποιότητας του συστήματος. Επιπλέον, φαίνεται η απόσταση μεταξύ της πραγματικής ποιότητας και της «Optimum» («βέλτιστής») ποιότητας. Η προσπάθεια που απαιτείται από την ομάδα ανάπτυξης να επιτύχει τη βέλτιστη ποιότητα, αντιπροσωπεύει το Κεφάλαιο ΤΧ, δηλαδή την κατακόρυφη απόσταση μεταξύ «Actual» και «Optimum». Ο Τόκος ΤΧ είναι η αρνητική επίπτωση του ΤΧ και αντιπροσωπεύει την πρόσθετη προσπάθεια που απαιτείται για τη συντήρηση του λογισμικού στην πραγματική κατάσταση, σε σύγκριση με την προσπάθεια που θα χρειαζόταν εάν το σύστημα ήταν βέλτιστης ποιότητας.



Εικόνα 2-1: Οπτικοποίηση Ορολογίας ΤΧ (Chatzigeorgiou et al., 2015)

Το SonarQube θεωρείται το πιο συχνά χρησιμοποιούμενο εργαλείο για την εκτίμηση του Κεφαλαίου ΤΧ (Yli-Huumo et al. (2016)), τουλάχιστον για το ΤΧ σε επίπεδο κώδικα, παρόλο που έχουν προταθεί και άλλα εργαλεία στη βιβλιογραφία. Στο SonarQube, το Κεφάλαιο ΤΧ αξιολογείται μέσω του αριθμού των μη-βέλτιστα υλοποιημένων κομματιών στον πηγαίο κώδικα, και του χρόνου που απαιτείται για την επίλυση αυτών. Ο αλγόριθμος της πλατφόρμας με τον οποίο υπολογίζεται το ΤΧ, βασίζεται στο άθροισμα του κόστους αποκατάστασης όλων των ζητημάτων ποιότητας.

Από την άλλη πλευρά, ο Τόκος ΤΧ είναι πιο δύσκολο να εκτιμηθεί, καθώς αναφέρεται στη διαφορά της προσπάθεια που απαιτείται μεταξύ της συντήρησης του πραγματικού και ενός υποθετικού βέλτιστου συστήματος. Παρά το γεγονός ότι ο Τόκος ΤΧ δεν μπορεί να ποσοτικοποιηθεί με βεβαιότητα, στη βιβλιογραφία μπορούμε να εντοπίσουμε διάφορες προσεγγίσεις. Οι Seaman και Guo (2011) προτείνουν ότι το τεχνικό χρέος πρέπει να αποπληρωθεί για να «αποφευχθεί ο τόκος με τη μορφή μειωμένης συντηρησιμότητας». Οι Conejero et al. (2018) αναγνωρίσαν ότι «η συντηρησιμότητα είναι ένα από τα κύρια χαρακτηριστικά που συμβάλλουν στον τόκο του τεχνικού χρέους». Ομοίως, οι Zazworka et al. (2014) χρησιμοποιήσαν το επήρεια σε λάθη και αλλαγές ως δείκτες για τον Τόκο ΤΧ και αναγνώρισαν την σύνδεσή τους με το μελλοντικό κόστος συντήρησης. Οι MacCormack και Sturtevant (2016) ανέλυσαν τη σχέση μεταξύ αποφάσεων σχεδιασμού και κόστους συντήρησης, που θεωρείται ότι αντιπροσωπεύουν τον Τόκο ΤΧ. Με βάση τα παραπάνω, οι μετρικές που αξιολογούν τη συντηρησιμότητα σε επίπεδο σχεδιασμού, όπως η κληρονομικότητα, η συνοχή, η σύζευξη, η πολυπλοκότητα και το μέγεθος, μπορούν να χρησιμοποιηθούν ως δείκτες για την εκτίμηση του Τόκου ΤΧ. Ωστόσο, η πραγματική ποσοτικοποίησή παραμένει μια ανοιχτή ερευνητική κατεύθυνση.

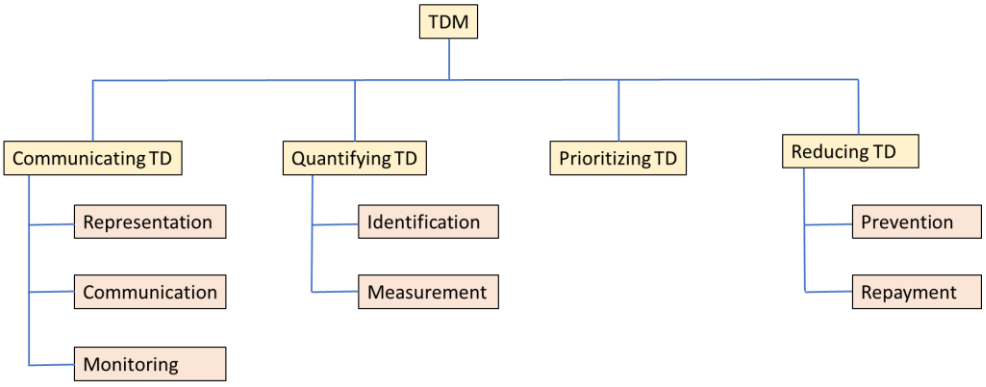
2.1.2 Διαχείριση Τεχνικού Χρέους

Η συντήρηση μπορεί να θεωρηθεί ως η πιο δαπανηρή δραστηριότητα στην ανάπτυξη λογισμικού, καθώς αποτελεί το 50 - 75% της συνολικής προσπάθειας κατά τη διάρκεια ζωής ενός λογισμικού (van Vliet (2008)). Ορισμένες δραστηριότητες συντήρησης, όπως η προσθήκη νέων λειτουργιών, η διόρθωση σφαλμάτων, η βελτίωση του χρόνου εκτέλεση κτλ., δεν μπορούν να αγνοηθούν ή να αναβληθούν. Από την άλλη πλευρά, δραστηριότητες συντήρησης που σχετίζονται με τη βελτίωση της σχεδιαστικής ποιότητας συνήθως παραμελούνται ή αναβάλλονται για μελλοντικές επαναλήψεις. Παρ' όλα αυτά, εάν οι προγραμματιστές παραμελούν αυτές τις δραστηριότητες συντήρησης, με την πάροδο του χρόνου η ποιότητα του συστήματος λογισμικού θα εξασθενεί σταδιακά (Parnas et al. (1994)). Αυτό μπορεί να οδηγήσει στη δημιουργία ενός οικονομικού κόστους, το οποίο ονομάζεται τεχνικό χρέος, το οποίο μπορεί να γίνει τόσο μεγάλο εάν η διαχείριση του ΤΧ δεν γίνει σωστά, που μπορεί τελικά να χρεοκοπήσει την εταιρία ανάπτυξης λογισμικού (Avgeriou et al. (2016)).

Σύμφωνα με τους Li et al. (2015), η αποτελεσματική διαχείριση τεχνικού χρέους (TDM) αποτελείται από οκτώ δραστηριότητες:

- ***Αποπληρωμή*** (repayment), που αφορά την μείωση του ποσού του συσσωρευμένου ΤΧ μέσω διάφορων τεχνικών ανακατασκευής.
- ***Αναγνώριση*** (identification), που αφορά τον εντοπισμός των τμημάτων που υποφέρουν από ΤΧ.
- ***Μέτρηση*** (measurement), ποσοτικοποίηση του ΤΧ τόσο του κεφαλαίου όσο και του τόκου.
- ***Παρακολούθηση*** (monitoring), δηλαδή η καταγραφή και αξιολόγηση της εξέλιξης του ΤΧ.
- ***Προτεραιότητα*** (prioritization), που αφορά τον προσδιορισμό των στοιχείων με μεγάλο ΤΧ που θα εξοφληθούν πρώτα
- ***Επικοινωνία*** (communication), εξήγηση της φύσης, συνέπιες και τρόπους αντιμετώπισης ΤΧ στα μέλη της εταιρίας
- ***Πρόληψη*** (prevention), η οποία αφορά την αποφυγή της συσσώρευσης πρόσθετου ΤΧ
- ***Αναπαράσταση / Τεκμηρίωση*** (representation / documentation) καταγραφή όλων των αποφάσεων, ενεργειών, μετρήσεων, που σχετίζονται με την διαχείριση του τεχνικού χρέους.

Αυτές οι οκτώ δραστηριότητες μπορούν να ομαδοποιηθούν σε τέσσερις δραστηριότητες υψηλού επιπέδου (Arvanitou (2019)), σύμφωνα με τον στόχο τους, όπως απεικονίζεται στην Εικόνα 2-2.



Εικόνα 2-2: Ομαδοποίηση δραστηριοτήτων διαχείριση του τεχνικού χρέους (Arvanitou et al. (2019))

Όπως σημειώνεται από τον Eisenberg (2013), η πλήρης αποπληρωμή του ΤΧ θεωρείται μη ρεαλιστική. Για τον λόγο αυτό, το Τεχνικό Χρέος θα πρέπει να παρακολουθείται και να διαχειρίζεται συνεχώς, καθώς συσσωρεύεται, και μπορεί να βλάψει όλες τις δραστηριότητες ανάπτυξης (Kruchten et al. (2012)). Από όλους τους τύπους ΤΧ, το ΤΧ σε επίπεδο κώδικα αναφέρεται ως ο πιο συχνά μελετημένος τύπος (Alves et al. (2016)) και ένας τύπος πολύ κρίσιμος για την βιομηχανία (Ampatzoglou et al. (2016)).

2.2 Προγραμματισμός Υψηλών Επιδόσεων

Ο προγραμματισμός υψηλών επιδόσεων (HPC) αναφέρεται στην ανάλυση, τον σχεδιασμό, την δοκιμή και την ανάπτυξη εφαρμογών λογισμικού για επιστημονικούς σκοπούς (π.χ. φυσική, βιολογία, ιατρική ανάλυση και επιστήμη δεδομένων). Η ανάγκη για συνεχή πειραματισμό και επικύρωση διάφορων τεχνικών πριν από την κυκλοφορία των επιστημονικών αποτελεσμάτων, έχει οδηγήσει στην ανάπτυξη του επιστημονικού λογισμικού ως σημαντική μέθοδο για την επιτυχία των ερευνητικών προσπαθειών (Birdsall και Langdon (1991)). Έχοντας υπόψη τα οφέλη της χρήσης πρακτικών τεχνολογίας λογισμικού στην ανάπτυξη επιστημονικού λογισμικού, υπάρχει ένα αυξανόμενο ενδιαφέρον μεταξύ των προγραμματιστών. Αυτό το αυξανόμενο ενδιαφέρον, έχει αρχίσει να επηρεάζει τη βιβλιογραφία σε αυτόν τον τομέα.

Η βιβλιογραφική ανασκόπηση των Heaton και Carver (2015) στόχευσε στον εντοπισμό του τρόπου με τον οποίο οι προγραμματιστές επιστημονικού λογισμικού

χρησιμοποιούν τις πρακτικές της τεχνολογίας λογισμικού στο HPC. Τα αποτελέσματα έδειξαν ότι: (α) Παρακολούθηση Ζητημάτων (Issue Tracking) και Συστήματα Ελέγχου Έκδοσης (Version Control Systems) είναι οι πρακτικές που υιοθετήθηκαν περισσότερο, και (β) Επαλήθευση και Επικύρωση (Verification and Validation) και Δοκιμή (Testing) είναι οι πρακτικές που οι επιστημονικοί προγραμματιστές λογισμικού θεωρούν σημαντικές, ωστόσο δεν έχουν ακόμη υιοθετηθεί ευρέως.

Η βιβλιογραφική ανασκόπηση των Sletholt et al. (2011) είχε να κάνει με τις πρακτικές agile και τις επιπτώσεις τους στην επιστημονική ανάπτυξη λογισμικού. Αυτοί διερεύνησαν τον βαθμό στον οποίο τα επιστημονικά προγράμματα λογισμικού έχουν χρησιμοποιήσει τέτοιες πρακτικές και επιπλέον, τον αντίκτυπο που είχαν αυτές. Τα αποτελέσματα της έρευνας αυτής, δείχνουν, ότι επιστημονικά έργα που υιοθετούν πρακτικές agile επιτυγχάνουν καλύτερα επίπεδα δοκιμών. Επίσης, οι συγγραφείς παρατήρησαν μια θετική επίδραση στις δραστηριότητες που σχετίζονται και με τις απαιτήσεις.

Με βάση τις παραπάνω έρευνες, φαίνεται ότι υπάρχει ιδιαίτερο ενδιαφέρον από την μεριά των προγραμματιστών υψηλών επιδόσεων για τις πρακτικές της τεχνολογίας λογισμικού, καθώς φαίνεται να επηρεάζουν θετικά το αναπτυσσόμενο λογισμικό.

3 Ποσοτικοποίηση Τεχνικού Χρέους

Η ποσοτικοποίηση του τεχνικού χρέους χωρίζεται σε δύο κατευθύνσεις, το ΤΧ σε επίπεδο κώδικα και σε επίπεδο σχεδίασης. Για την ποσοτικοποίηση του ΤΧ σε επίπεδο κώδικα όπως είναι και ευρέος γνωστό στην βιβλιογραφία, υπάρχουν διάφορα προγράμματα τα οποία υπολογίζουν τον χρόνο επίλυσης παραβιάσεων κανόνων. Στην περίπτωση μας χρησιμοποιήθηκε το SonarQube μαζί με την βοήθεια των plugin: (α) icode CNES, για την υποστήριξη της γλώσσας προγραμματισμού FORTRAN, και (β) SonarQube C++ Community για την γλώσσα προγραμματισμού C / C++.

Το ΤΧ σε επίπεδο σχεδίασης υπολογίζεται ως το χρηματικό ποσό που αντιστοιχεί στην προσπάθεια που απαιτείται για την επίλυση των σχεδιαστικών προβλημάτων (Li et al. (2015)). Δεδομένου ότι το SonarQube δεν είναι σε θέση να υπολογίσει τον σχεδιασμό και την αρχιτεκτονική (Avgeriou et al. (2020)), απαιτείται μια νέα προσέγγιση. Για την ποσοτικοποίηση του Κεφαλαίου ΤΧ σε επίπεδο σχεδίασης, ακολουθήσαμε τα εξής βήματα:

- ορισμός λίστας σχεδιαστικών προβλημάτων που θα εντοπιστούν
- αναγνώριση στοιχείων (π.χ. αρχεία, modules, διαδικασίες) που υποφέρουν από αυτά τα σχεδιαστικά προβλήματα
- εκτίμηση του χρόνου που απαιτείται για την επίλυση κάθε προβλήματος
- άθροιση του χρόνου που απαιτείται για την επίλυση των σχεδιαστικών προβλημάτων όλων των στοιχείων

3.1 Ορισμός Σχεδιαστικών Προβλημάτων

Ξεκινήσαμε την έρευνα μας για τον εντοπισμό σχεδιαστικών προβλημάτων στον κώδικα ενός έργου από το βιβλίο για refactorings του Fowler (1997). Αφού μελετήσαμε τα προβλήματα σχεδιασμού που παρουσιάζονται στο βιβλίο αυτό και λαμβάνοντας υπόψη το τρόπο με τον οποίο γίνεται η συγγραφή κώδικα σε εφαρμογές μεγάλης κλίμακας, όπου η χρήση αντικειμενοστραφή προγραμματισμού είναι μειωμένη, αποφασίσαμε να επικεντρωθούμε σε τέσσερα σχεδιαστικά προβλήματα που ταιριάζουν στον συναρτησιακό προγραμματισμό (π.χ. εξαιρούμενης της κληρονομικότητας και πολυμορφισμού):

- ***πολύπλοκα αντικείμενα (complex artifacts)***: Ο κώδικας ορισμένων διαδικασιών (procedures) παρουσιάζει υπερβολικά επίπεδα πολυπλοκότητας, όσον αφορά τους κόμβους αποφάσεων ή επανάληψης. Τέτοια κομμάτια κώδικα (και τα αρχεία που

τα περιέχουν) είναι δύσκολο να κατανοηθούν και να διατηρηθούν (McCabe et al. (1976)).

- ***υπερβολικά συνδεδεμένα αντικείμενα (over-coupled artifacts)***: Ορισμένα αρχεία (files) ή λειτουργικές μονάδες (modules) περιλαμβάνουν συνδέσεις με άλλα αρχεία σε υπερβολικό αριθμό, καθώς χρειάζονται τις πληροφορίες τους για να μεταγλωττιστούν. Τέτοια αρχεία είναι επιρρεπή σε αλυσιδωτές αντιδράσεις, δηλαδή χρειάζονται επανεξέταση κάθε φορά που αλλάζει ένα εξαρτώμενο αρχείο, αυτό οδηγεί σε επιπλέον προσπάθεια συντήρησης (Arvanitou et al. (2017)).
- ***μεγάλα αντικείμενα (Large artifacts)***: Κάποια αντικείμενα (modules, files, or procedures) είναι μεγάλου μεγέθους (συνήθως ως προς τις γραμμές κώδικα). Αυτά τα αντικείμενα έχουν περισσότερους από έναν λόγους αλλαγής (δηλ. ευθύνες). Αυτά τα αντικείμενα παραβιάζουν την (Chidamber and Kemerer (1994)) Αρχή της Ενιαίας Ευθύνης (SRP) (Martin (2002)) και επομένως είναι πιο πιθανό να υποβληθούν σε συντήρηση και να παράγουν Τόκο ΤΧ.

Είναι σημαντικό να σημειώσουμε ότι η παραπάνω λίστα δεν είναι ολοκληρωμένη, και ως εκ τούτου το Κεφάλαιο ΤΧ σε επίπεδο σχεδίασης που θα υπολογιστεί θα είναι ένα κομμάτι του πραγματικού. Ωστόσο, θεωρούμε αυτή τη λίστα κατάλληλη, δεδομένου ότι: (α) καταγράφει τις σημαντικότερες (μη αντικειμενοστραφείς) ιδιότητες της δυνατότητας συντήρησης λογισμικού (Riaz (2009)), και (β) δεν ήταν δυνατή η καταγραφή όλων των πιθανών τύπων προβλημάτων σχεδιασμού κατά τη διάρκεια της διπλωματικής. Συνοψίζοντας και λαμβάνοντας υπόψη ότι τα αντικείμενα, σε γλώσσες που δεν είναι αντικειμενοστραφείς, είναι συνήθως αρχεία και διαδικασίες, πρέπει να αντιμετωπιστούν τα ακόλουθα προβλήματα σχεδιασμού: (α) ***Complex Procedures*** (πολύπλοκές διαδικασίες), (β) ***Over Coupled Files/Modules*** (υπερβολικά συνδεδεμένα αρχεία/λειτουργικές μονάδες), (γ) ***Large Files/Modules*** (μεγάλα αρχεία/λειτουργικές μονάδες), και (δ) ***Long Procedures*** (μεγάλες διαδικασίες).

3.2 Εντοπισμός Σχεδιαστικών Προβλημάτων

Για τον εντοπισμό σχεδιαστικών προβλημάτων σε διάφορα έργα απαιτείται μια κλιμακούμενη προσέγγιση που μπορεί να αυτοματοποιηθεί. Για αυτό τον λόγο, επιλέξαμε μια προσέγγιση με βάση τον υπολογισμό μετρικών, για τον εντοπισμό προβλημάτων (Marinescu (2004)), δηλαδή, υπολογίσαμε τις τιμές των μετρικών για κάθε τύπο προβλήματος, ταξινομήσαμε τα αντικείμενα (αρχεία και διαδικασίες) για κάθε μετρική

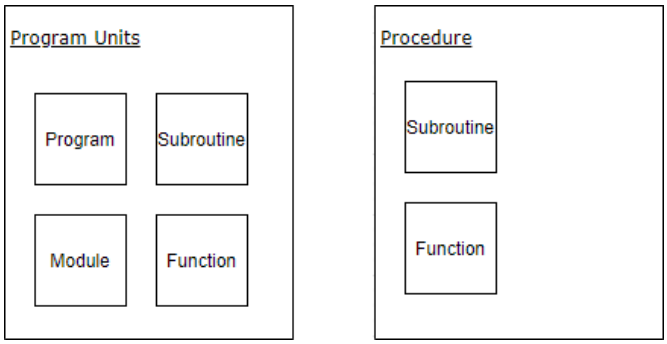
ξεχωριστά και τα χειρότερα ορίζονται ως προβληματικά. Η χρήση ορίων στις μετρικές (thresholds) ως δείκτες προβληματικών αντικειμένων υποστηρίζεται σε μεγάλο βαθμό από την βιβλιογραφία (Ferreira et al. (2012)). Στην Ενότητα 3.2.1, παρουσιάζουμε τη διαδικασία επιλογής των μετρικών, καθώς και λεπτομέρειες σχετικά με τον υπολογισμό τους. Στην Ενότητα 3.2.2, παρουσιάζουμε την προσέγγιση για την εξαγωγή των μετρικών ορίων και των πραγματικών τιμών που έχουμε ανακτήσει.

3.2.1 Επιλογή Μετρικών και Διαδικασία Υπολογισμού

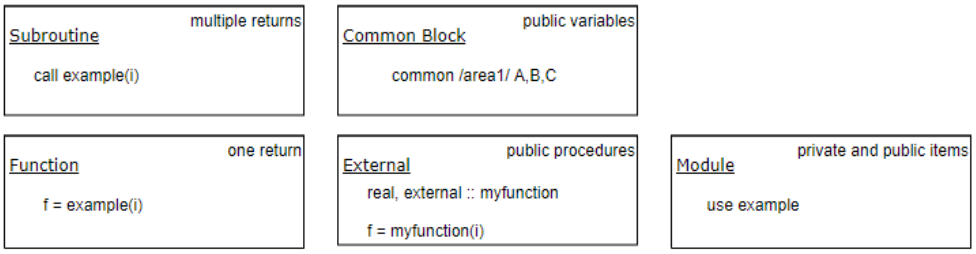
Ως πρώτο βήμα προς την εφαρμογή της προτεινόμενης μεθοδολογίας, πρέπει να επιλέξουμε τις μετρικές που θα χρησιμοποιήσουμε για τον εντοπισμό προβλημάτων σχεδιασμού. Με βάση τα προβλήματα που έχουμε ορίσει στην Ενότητα 3.1, το πρόγραμμα του έχει υλοποιηθεί υπολογίζει πέντε μετρικές: (α) cyclomatic complexity (CC) (McCabe 1976)—για ***Complex Procedures***, (β) coupling between files (CBF)—για ***Over Coupled Files / Modules***, (γ) lines of code (LOC) (Li and Henry (1993))—για ***Large Files / Modules***, (δ) lack of cohesion of lines (LCOL) (Charalampidou et al. (2016))—για ***Long Procedure***, και (ε) lack of cohesion of procedures (LCOP) —για ***Large Files/Modules***. Από την παραπάνω λίστα, 3 μετρικές (συγκεκριμένα: CC, LOC και LCOL) χρησιμοποιούνται όπως έχουν προταθεί στη βιβλιογραφία. Ενώ οι άλλες δύο (CBF και LCOP) εισάγονται στην διπλωματική με καινούργιο τρόπο. Ωστόσο, πρέπει να σημειώσουμε ότι και οι δύο μετρικές δεν έχουν αναπτυχθεί από το μηδέν, καθώς βασίζονται στη σύζευξη μεταξύ αντικειμένων (CBO) και στην έλλειψη συνοχής μεθόδων (LCOM) (Chidamber and Kemerer (1994)). Συγκεκριμένα: (α) Η CBF αναφέρεται στον αριθμό εξωτερικών συσχετίσεων file / modules και (β) Η LCOP αναφέρεται στον αριθμό των διαδικασιών που δεν συσχετίζεται με χρήση των μεταβλητών του module. Για να μπορέσουμε να υπολογίσουμε τις προαναφερθείσες μετρικές, πρέπει να γίνει μια διάκριση μεταξύ FOTRAN και C, λόγο του ότι έχουν μια διαφορετική προσέγγιση για τη διαχείριση του πεδίου ορισμού και χρήσης των μεταβλητών, η οποία επηρεάζει άμεσα στον τρόπο με τον οποίο γίνεται αντιληπτή / ορίζεται η σύζευξη και η συνοχή στις δύο γλώσσες.

<u>*Σύζευξη και συνοχή στην γλώσσα προγραμματισμού FORTRAN*</u>: Για τον ορισμό αυτών των μετρικών στη γλώσσα προγραμματισμού FOTRAN, πρέπει πρώτα να κατανοήσουμε μερικά από τα βασικά στοιχεία αυτής της γλώσσας και τον τρόπο με τον οποίο διαρθρώνονται σε ένα πρόγραμμα. Αρχικά, απαιτείται ένα στοιχείο προγράμματος (Program) το οποίο είναι ο ελεγκτής ολόκληρης της εφαρμογής, με την έννοια ότι είναι το

σημείο εισόδου της εκτέλεσης - δηλαδή, όπως η κύρια λειτουργία main σε C / C ++ και Java. Επιπλέον υπάρχει το υποπρόγραμμα (subprogram) το όποιο είναι ένα αρχείο στο οποίο ο προγραμματιστής δηλώνει και προσδιορίζει την λειτουργεία διαδικασιών (procedures), οι οποίες μπορούν να κληθούν από το πρόγραμμα ή άλλες διαδικασίες. Οι διαδικασίες μπορούν να έχουν τη μορφή υπορουτίνας (subroutine) ή συνάρτησης (function). Και οι δύο χρησιμοποιούνται για να περιγράψουν μια συγκεκριμένη εργασία ή υπολογισμό, αλλά η διαφορά μεταξύ τους είναι ότι η υπορουτίνα επιστρέφει πολλές τιμές, ενώ η συνάρτηση επιστρέφει μόνο μία. Όσο αφορά το πεδίο εφαρμογής μιας μεταβλητής, σε ένα υποπρόγραμμα, είναι η διαδικασία στην οποία έχει δηλωθεί. Για το λόγο αυτό υπάρχει το Common Block, το οποίο χρησιμοποιείται για να δηλώσει δημόσιες μεταβλητές ή ακόμη και δομές μεταβλητών όπου το πεδίο εφαρμογής τους είναι ολόκληρο το έργο. Το πεδίο εφαρμογής μιας διαδικασίας είναι πάλι πολύ περιορισμένο και είναι προσβάσιμο μόνο από το πρόγραμμα ή το υποπρόγραμμα στο οποίο ορίζεται, για το λόγο αυτό, χρησιμοποιείται η δήλωση “external”. Ωστόσο η χρήση διαδικασιών με την δήλωση “external” δεν είναι καλή πρακτική και δεν συνιστάτε για συχνή χρήση. Ο βασικός λόγος χρήσης της δήλωσης αυτής είναι κυρίως για πρόσβαση σε διαδικασίες που είναι υλοποιημένες σε διαφορετικές γλώσσες προγραμματισμού. Ένας άλλος τρόπος με τον οποίο είναι δυνατή η πρόσβαση σε μια διαδικασία, χωρίς τη χρήση της δήλωσης “external”, είναι μέσω του συνδέτη (linker) στην κατασκευή και μεταγλώττιση του προγράμματος. Με την αναβάθμιση της Fortran 77 στην έκδοση 90 παρουσιάστηκε ένα νέο βασικό χαρακτηριστικό προκειμένου να γίνει μία προσπάθεια μίμησης των νεότερων γλωσσών με αντικειμενοστραφή προσέγγιση. Η Fortran παρουσίασε το μοντέλο αντικειμένων (object-based) που εφαρμόζεται με τη δημιουργία Modules όπου επιτρέπετε η χρήση αντικειμένων (objects) και ενθυλάκωσης (encapsulation). Τα βασικά μέρη ενός προγράμματος FORTRAN φαίνονται στην Εικόνα 3-1 και τα βασικά στοιχεία που αναφέρθηκαν απεικονίζονται στην Εικόνα 3-2.

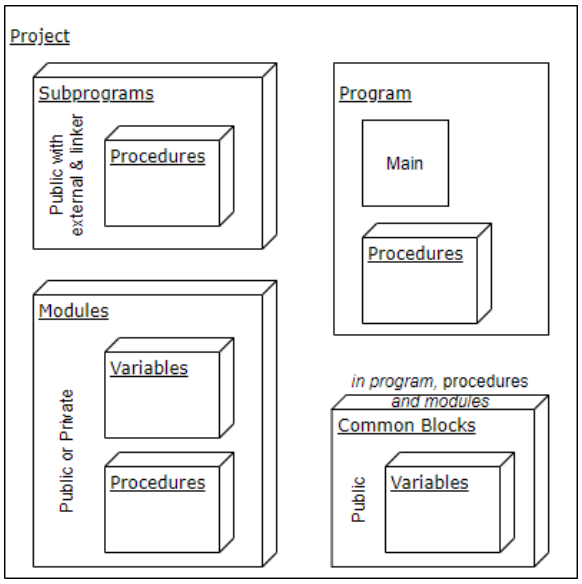


Εικόνα 3-1: Βασικά μέρη ενός προγράμματος FORTRAN



Εικόνα 3-2: Βασικά στοιχεία της γλώσσας προγραμματισμού FORTRAN

Έχοντας εξηγήσει τα βασικά στοιχεία της γλώσσας προγραμματισμού Fortran, στην Εικόνα 3-3 απεικονίζουμε μια συχνή δομή ενός έργου. Κάθε έργο (Project) έχει: (α) ένα πρόγραμμα (Program) που περιέχει ένα αντικείμενο main και διάφορες διαδικασίες (Procedures), (β) υποπρογράμματα (Subprograms) με διάφορες διαδικασίες (Procedures) οι οποίες μπορεί να έχουν δημόσια πρόσβαση με τη χρήση του external ή του linker, (γ) modules με δημόσιες και ιδιωτικές μεταβλητές (Variables) και διαδικασίες (Procedures), και (δ) common blocks με δημόσιες μεταβλητές που μπορούν να υπάρχουν μέσα στο πρόγραμμα, τις διαδικασίες και τα modules.



Εικόνα 3-3: Δομή ενός προγράμματος FORTRAN

Σύζευξη μεταξύ αρχείων, είναι ο αριθμός των αρχείων που σχετίζονται με ένα δεδομένο αρχείο, το οποίο στην γλώσσα προγραμματισμού Fortran μπορεί να επιτευχθεί με τη χρήση μιας εξωτερικής διαδικασίας, τη χρήση διαδικασιών ή μεταβλητών των modules και τη χρήση common blocks. Επομένως, η μετρική της σύζευξη μεταξύ αρχείων (CBF) είναι ο αριθμός των εμφανίσεων των μεθόδων αυτών που δείχνουν σε ένα μοναδικό αρχείο.

Ο υπολογισμός της συνοχής απαιτεί την ύπαρξη ενός στοιχείου στο οποίο συνυπάρχουν διαδικασίες και τοπικές μεταβλητές. Επομένως, η μετρική της συνοχής μπορεί να οριστεί μόνο σε modules που περιέχουν μεταβλητές, σε όλες τις υπόλοιπες οντότητες της γλώσσας η συνοχή δεν ορίζεται. Η μετρική έλλειψης συνοχής διαδικασιών (LCOP) βασίζεται κυρίως στη μετρική LCOM1 (Chidamber and Kemerer (1994)): Για κάθε module, για τον υπολογισμό της μετρικής, αφαιρούμε τον αριθμό ζευγών μεθόδων που δεν μοιράζονται μεταβλητές από τον αριθμό ζευγών μεθόδων που μοιράζονται μεταβλητές. Όπως και η μετρική LCOM1, εάν το αποτέλεσμα της αφαίρεσης είναι αρνητικό, η μετρική ορίζεται ίση με το μηδέν.

Σύζευξη και συνοχή στην γλώσσα προγραμματισμού C: Όσον αφορά τη γλώσσα προγραμματισμού C, δύο αρχεία μπορούν να συσχετιστούν μόνο με τη χρήση των λέξεων-κλειδιών include και extern. Η μετρική της σύζευξης μεταξύ αρχείων (CBF) ορίζεται ως ο αριθμός εμφανίσεων των δηλώσεων συμπερίληψης ενός αρχείου από ένα άλλο. Τέλος, επειδή σε αυτή τη γλώσσα προγραμματισμού έχουμε μόνο έναν τύπο αρχείου και ένα τύπο αρχείου κεφαλίδας (header file) με μεθόδους και μεταβλητές, ο ορισμός της συνοχής ήταν απλούστερος. Βασιζόμενοι ξανά στην μετρική LCOM1 (Chidamber and Kemerer (1994)) και για κάθε αρχείο (που περιέχει μεταβλητές επιπέδου αρχείου) αφαιρούμε τον αριθμό ζευγών μεθόδων που δεν μοιράζονται μεταβλητές από τον αριθμό ζευγών μεθόδων που μοιράζονται μεταβλητές. Ομοίως με το LCOM1, δεν επιτρέπουμε τιμές κάτω από το μηδέν και σε αυτές τις περιπτώσεις η μετρική παίρνει την τιμή μηδέν.

Συνοψίζοντας, στους Πίνακες 3-1 με 3-3, παρέχουμε μια επισκόπηση των υπολογισμένων μετρικών, για μονάδες προγράμματος, γλώσσες προγραμματισμού και τα σχεδιαστικά προβλήματα τα οποία χαρτογραφούνται. Στις περιπτώσεις όπου χρησιμοποιούνται περισσότερες από μία μετρικές για τον εντοπισμό της ύπαρξης ενός συγκεκριμένου προβλήματος, πραγματοποιείται ένωση των στοιχείων.

Πίνακας 3-1: Μετρικές για την γλώσσα προγραμματισμού FORTRAN 90

Σχεδιαστικό Πρόβλημα	CC	CBF	LOC	LCOL	LCOP
Complex Procedures	Procedures				
Long Procedures				Procedures	

Over Coupled Files / Modules		Modules / Files			
Large Files / Modules			Modules / Files		Modules / Files

Πίνακας 3-2: Μετρικές για την γλώσσα προγραμματισμού FORTRAN 77

Σχεδιαστικό Πρόβλημα	CC	CBF	LOC	LCOL
Complex Procedures	Procedures			
Long Procedures				Procedures
Over Coupled Files / Modules		Files		
Large Files / Modules			Files	

Πίνακας 3-3: Μετρικές για την γλώσσα προγραμματισμού C

Σχεδιαστικό Πρόβλημα	CC	CBF	LOC	LCOL	LCOP
Complex Procedures	Procedures				
Long Procedures				Procedures	
Over Coupled Files / Modules		Files			
Large Files / Modules			Files		Files

3.2.2 Προσδιορισμός Μέγιστων Επιτρεπτών Τιμών

Δεδομένου του ότι τα έργα που επιλέχτηκαν για ανάλυση (και γενικά τα έργα υψηλής υπολογιστικής ισχύος) μπορεί να είναι πολύ αποκλίνουσες μεταξύ τους οι τιμές των μετρικών όσον αφορά το μέγεθος, την πολυπλοκότητα κ.λπ.. Γι' αυτό το λόγο προτιμήσαμε να θέσουμε συγκεκριμένα όρια επιτρεπτών τιμών για κάθε έργο και όχι

κάποια γενικευμένα. Αυτή η απόφαση βασίζεται και στο γεγονός ότι, ανεξάρτητα από το πόσο «καλή» ή «κακή» είναι η ποιότητα κώδικα ενός συνόλου έργων, ο υπολογισμός αντικειμένων που χρήζουν αναδιαμορφώσεις (refactorings) είναι περιορισμένος και δεν μπορεί να εξαπλωθεί σε μεγάλο αριθμό αντικειμένων που υπάρχουν στο έργο. Για τον υπολιγισμό των επιτρεπτών ορίων κάθε μετρικής, υπολογίζεται αρχικά η τιμή κάθε μετρικής για όλα τα αντικείμενα και το 10% των χειρότερων αντικειμένων ανά μετρική ορίζεται ως μη επιτρεπτό. Τα όρια των μερτικών για κάθε ένα από τα εξεταζόμενα έργα για κάθε μετρική παρουσιάζονται στον παρακάτω Πίνακα 3-4.

Πίνακας 3-4: Όριο επιτρεπτών τιμών ανά έργο

Project	CC	CBF	LOC	LCOL	LCOP
LQCD—SU(3) Multiplications	5	3	27	51	26
LQCD—Metropolis update of a pure Yang-Mills theory	5	3	30	214	26
LQCD—Matrix-vector multiplication using Wilson fermions	6	3	30	193	26
KKRnano	1	4	24	2283	36
Metalwalls	7	11	75	1309	1
CO_2Capture	2	5	49	1358	78

3.3 Εκτίμηση Χρόνου Επίλυσης Σχεδιαστικών Προβλημάτων

Για λύση στα προβλήματα που ορίζονται στην Ενότητα 3.1, προτείνουμε την εφαρμογή δύο γνωστών αναδιαμορφώσεων (refactorings): εξαγωγή διαδικασίας (Extract Procedure) και εξαγωγή αρχείου / module (Extract File / Module). Συγκεκριμένα, η αναδιαμόρφωση Extract Procedure στοχεύει τα προβλήματα σχεδιασμού Long Procedures και Complex Procedures, ενώ η αναδιαμόρφωση Extract File / Module αναμένεται να

επιλύσει τα προβλήματα σχεδίασης Large File / Module και Over-Coupled File / Module. Ο στόχος αυτής της υποενότητας είναι να εκτιμήσουμε τον χρόνο που απαιτείται για την εκτέλεση αυτών των αναδιαμορφώσεων, χωρίς την χρήση αυτοματοποιημένων εργαλείων, ώστε να εκτιμηθεί ο χρόνος που απαιτείται για την εξάλειψη ενός σχεδιαστικού προβλήματος. Για την επίτευξη αυτού του στόχου εργαστήκαμε στον κώδικα δύο μεγάλης κλίμακας έργων: Metalwalls (υλοποιημένο στην γλώσσα προγραμματισμού C) και CO_2Capture (υλοποιημένο στην γλώσσα FORTRAN). Για την συστηματοποίηση της διαδικασία εύρεσης του απαιτούμενου χρόνου για τις αναδιαμορφώσεις, εφαρμόσαμε την ακόλουθη διαδικασία:

- Εύρεση αντικειμένων που αντιμετωπίζουν σχεδιαστικά προβλήματα
- Σχεδιασμός λύσης για την επίλυση του προβλήματος – καταγραφή χρόνου σε λεπτά μέχρι την εύρεση της λύσης
- Εφαρμογή της λύσης στον κώδικα – καταγραφή χρόνου σε λεπτά για την εφαρμογή της λύσης
- Πολλαπλασιασμός του αθροίσματος των παραπάνω χρόνων με τον μέσο μισθό ενός προγραμματιστή

Σε αυτό το σημείο πρέπει να αναφέρουμε ότι σε περίπτωση που χρησιμοποιείτε κάποιο εργαλείο που κάνει εντοπισμό αναδιαμορφώσεων, στο παραπάνω άθροισμα θα πρέπει να ληφθεί υπόψη μόνο η προσπάθεια της εύρεσης της αναδιαμόρφωσης. Επιπλέον, σημειώνουμε ότι ο πολλαπλασιασμός του 4ου βήματος πραγματοποιείται με βάση τον παγκόσμιο μέσο όρο μισθού ανά ώρα, αλλά μπορεί να προσαρμοστεί σε διαφορετικό κόστος μίσθωσης συγκεκριμένων εταιρειών ή χωρών.

3.3.1 Αποτίμηση του ανασχεδιασμού Extract Procedure

Όσον αφορά τον ανασχεδιασμό Extract Procedure (εξαγωγή διαδικασίας), εντοπίσαμε με μη αυτόματο τρόπο 84 διαφορετικά σημεία εφαρμογής στον κώδικα του Metalwalls και 47 στο έργο CO_2Capture. Πρέπει να σημειωθεί ότι εντοπίσαμε τέτοιες ευκαιρίες, βάσει του προβλήματος σχεδιασμού Long Process. Η απόφαση αυτή δεν προκαλεί κάποιο πρόβλημα στα αποτελέσματα μας, καθώς ο χρόνος που απαιτείται για την εφαρμογή ενός ανασχεδιασμού, δεν σχετίζεται με το σχεδιαστικό πρόβλημα που δημιουργεί την ανάγκη για την εφαρμογή του, αλλά μόνο με τους μηχανισμούς του ίδιου του ανασχεδιασμού. Τα 84 σημεία εφαρμογής Extract Procedure για το έργο Metalwalls παρουσιάζονται στον Πίνακα 3-5, μαζί με το συνολικό Κεφάλαιο ΤΧ για κάθε αρχείο.

Πίνακας 3-5: Το Κεφάλαιο ΤΧ για τα αρχεία του MetalWalls

Source File	Principal (σε λεπτά)
system.F90	226
configuration.F90	156
species.F90	72
command.F90	39
localwork.F90	63
ewald.F90	6
timers.F90	16
coulomb_lr.F90	57
coulomb_lr.cpp	51
coulomb.F90	55
coulomb_keq0.F90	89
electrode_parameters.F90	6

Ως παράδειγμα στις παραπάνω περιπτώσεις, εστιάζουμε στο αρχείο System.F90. Σε αυτό το αρχείο έχουν εντοπιστεί 21 ευκαιρίες για την εφαρμογή του Extract Procedure. Στον Πίνακα 3-6, παρουσιάζουμε το Κεφάλαιο ΤΧ που απαιτείται για την επίλυση αυτού του σχεδιαστικού προβλήματος. Η ανάλυση για τις υπόλοιπες περιπτώσεις μέτρησης του χρόνου εφαρμογής του ανασχεδιασμού παρουσιάζεται στο Παράρτημα Α.

Πίνακας 3-6: Ανάλυση του Κεφαλαίου ΤΧ για το αρχείο System.F90

Long Procedure	Extract Procedure	Χρόνος Επίλυσης (σε λεπτά)
read_data()	open_file()	6
	read_header()	8
	check_if_value_set()	10
	check_all_keywords()	6
	validate_values()	8
	read_box_parameters()	5
	write_box_lengthparameters()	10
	validate_array_coordinates_dimensions()	15

	read_coordinates()	5
	read_ions()	13
	read_atoms	9
	validate_array_velocities()	7
	read_velocities()	8
	read_ions_velocities()	6
	check_atoms_velocities()	11
	read_forces()	16
	read_thermostat_parameters()	8
	read_electrode_atom_charges()	10
	finilize_system_setup()	18
deallocate_data_arrays()	perform_deallocation()	23
	setup_do_output()	24

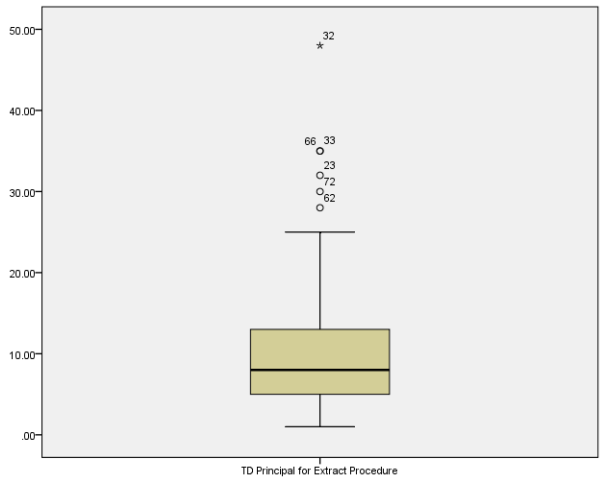
Ένα παράδειγμα παρουσιάζεται στην Εικόνα 3-4: στην αριστερή πλευρά του σχήματος παρουσιάζουμε τον κωδικό πριν από την εφαρμογή του refactoring, ενώ στη δεξιά πλευρά ο κώδικας μετά. Επειδή το συγκεκριμένο κομμάτι κώδικα (στα αριστερά) εκτελούσε μια συγκεκριμένη διαδικασία (ανάγνωση ενός αρχείου δεδομένων από το σύστημα) αποφασίσαμε να εκτελέσουμε μια αναδιαμόρφωση Extract Procedure, δημιουργώντας μια διαδικασία που διαβάζει ένα αρχείο δεδομένων από το σύστημα. Αργότερα, γενικεύσαμε τη χρήση αυτής της διαδικασίας, προκειμένου να διαβάσουμε ένα αρχείο από το σύστημα (είτε αρχείο τύπου config είτε αρχείο δεδομένων) και το μεταφέραμε στο αρχείο fileunit.F90.

```
! Open file
call W_fileunit_get_new_unit(funit)
open(unit=funit,file=datafile,
  access="SEQUENTIAL",action="READ",
  position="REWIND",orm="FORMATTED",
  status="OLD", iostat=ierr)
if (ierr /= 0) then
   call MW_errors_open_error
   ("read_data","configuration.f90",
     datafile, ierr)
end if
```

```
! Open file
call
MW_fileunit_open_file(funit,
ierr, datafile)
```

Εικόνα 3-4: Παράδειγμα εφαρμογής αναδιαμόρφωση Extract Refactoring

Η στατιστική ανάλυση σχετικά με την αποτίμηση του Κεφαλαίου ΤΧ για την εφαρμογή αναδιαμόρφωσης Long Procedure υποδηλώνει ότι, κατά μέσο όρο, κάθε περίπτωση απαιτεί 10,15 λεπτά για να ολοκληρωθεί. Η ελάχιστη τιμή είναι 1 λεπτό, η μέγιστη τιμή είναι 48, ενώ η τυπική απόκλιση είναι 8,97. Η ανάλυση παρουσιάζεται γραφικά στην παρακάτω εικόνα όπου απεικονίζει το θηκόγραμμα.



Εικόνα 3-5: Θηκόγραμμα για το Κεφάλαιο ΤΧ του MetalWalls για Extract Procedure

Οι 47 ευκαιρίες για το έργο CO_2Capture παρουσιάζονται στον Πίνακα 3-7, μαζί με το συνολικό Κεφάλαιο ΤΧ για κάθε αρχείο. Ως παράδειγμα στις παραπάνω περιπτώσεις, εστιάζουμε στο αρχείο frpmodel.f. Σε αυτό το αρχείο έχουν εντοπιστεί 26 ευκαιρίες εφαρμογής Extract Procedure. Στον Πίνακα 3-8, παρουσιάζουμε το Κεφάλαιο ΤΧ που απαιτείται για την επίλυση κάθε σχεδιαστικού προβλήματος Long Process. Η ανάλυση για τις υπόλοιπες περιπτώσεις παρουσιάζεται στο Παράρτημα Α.

Πίνακας 3-7: Το Κεφάλαιο ΤΧ για τα αρχεία του CO_2Capture

Source File	Principal (σε λεπτά)
element.f	177
frpmodel.f	223
frpmainmn.f	43
heater.f	10
heater2.f	3
read_reproducible.F90	7
write_reproducible.F90	5
funobj_cost_estimators.f	12
funobj_real_cost_estimators.f	23

Πίνακας 3-8: Ανάλυση του Κεφαλαίου TX για το αρχείο frpmodel.f

Long Procedure	Extract Procedure	Χρόνος Επίλυσης (σε λεπτά)
HEXCH()	HECMB_Rich_Stream()	13
	HECMB_Lean_Stream()	12
	HETMB_Rich_Stream()	11
	HETMB_Lean_Stream()	14
	Cold_Inlet_from_Absorber()	10
	Cold_Oulet_to_Stripper()	12
	Hot_Inlet_from_Stripper()	18
	Hot_Outlet_to_Absorber()	14
	Heat_Exchanger_Energy_Balance()	24
	Heat_Exchanger_Area_Calculation()	10
HEATER()	Heater_MEA_Partial_Pressure()	23
	Heater_Mass_Balance()	17
	Heater_Reboiler_Mass_Balance()	10
	Heater_Equilibrium_Relations()	13
	Heater_Equilibrium()	7
	Heater_Equilibrium_Relations_H2O()	6
	Heater_Reboiler_Energy_Balance()	6
	Heater_Summation_Equations()	3
HEATER2()	Heater2_MEA_Partial_Pressure()	10
	Heater2_Mass_Balance()	2
	Heater2_Reboiler_Mass_Balance()	1
	Heater2_Equilibrium_Relations()	1
	Heater2_Equilibrium()	1
	Heater2_Equilibrium_Relations_H2O()	1
	Heater2_Reboiler_Energy_Balance()	1
	Heater2_Summation_Equations()	1

Ένα παράδειγμα παρουσιάζεται στην Εικόνα 3-6, όπου στο πάνω μέρος παρουσιάζουμε τον κωδικό πριν από την εφαρμογή του refactoring, ενώ στο κάτω μέρος τον κώδικα μετά την εφαρμογή.

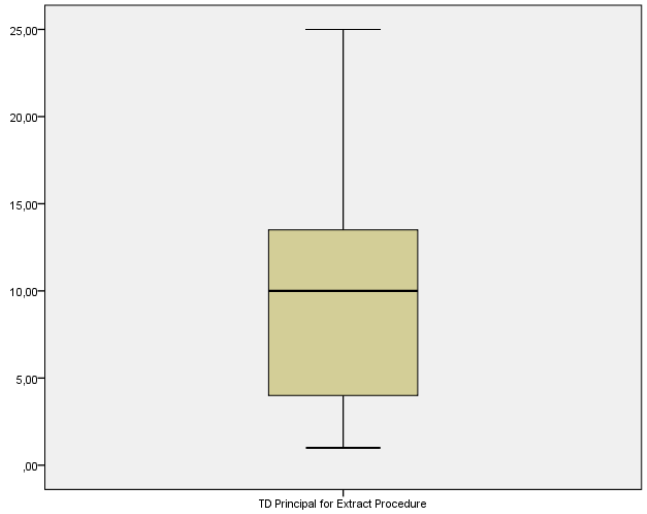
```
      C------------ Cost of Utilities ---------
      MW_am = MW(3)
      C_steam = (Steam * C_st * 3.65d2 * 2.4d1 * 3.6d3) * 1536.5d0
/&104d0
      Ψ_water = (WATMK * 1.8d1 * 1d-3 * 1.16d-4 * 3.65d2 * 2.4d1*3.6d3+
&WATER * 1.8d1 * 1d-3 * 1.16d-4) * 1536.5d0 / 1104d0
      C_amine = AMNMK * MW_am * 1d-3 * C_am * 3.65d2 * 2.4d1 * 3.6d3 +
& AMINE * MW_am * 1d-3 * C_am

      IF (GRADIENT) THEN
         dC_steamdQb = dSteamdQb * C_st * 3.65d2 * 2.4d1 * 3.6d3 * &
1536.5d0 / 1104d0
         dC_steamdTb = dSteamdTb * C_st * 3.65d2 * 2.4d1 * 3.6d3 * &
1536.5d0 / 1104d0
         dC_waterdF = 1.8d1 * 1d-3 * 1.16d-4 * 3.65d2 * 2.4d1 * 3.6d3
* & 1536.5d0 / 1104d0
         dC_waterdF2 = 1.8d1 * 1d-3 * 1.16d-4 * 1536.5d0 / 1104d0
         dC_aminedF = MW_am * 1d-3 * C_am * 3.65d2 * 2.4d1 * 3.6d3
         dC_aminedF2 = MW_am * 1d-3 * C_am
      ENDIF
```

```
      C-------------- Cost of Utilities --------
      CALL Funobj_Utilities_Cost(MW, MW_am, C_steam, C_st, C_water,
           & C_am, C_amine, Steam, WATMK, WATER, AMNMK,
           & AMINE, dC_steamdQb, dSteamdQb,
           & dC_steamdTb, dSteamdTb, dC_waterdF,
           & dC_waterdF2, dC_aminedF, dC_aminedF2,
           & GRADIENT)
```

Εικόνα 3-6: Παράδειγμα εφαρμογής αναδιαμόρφωση Extract Procedure

Δεδομένου ότι αυτό το συγκεκριμένο κομμάτι κώδικα (πριν την αναδιαμόρφωση) εκτελούσε μια συγκεκριμένη διαδικασία (υπολογίζοντας το κόστος των υπηρεσιών), αποφασίσαμε να εκτελέσουμε ένα refactoring Extract Procedure, δημιουργώντας μια διαδικασία (υπό-ρουτίνα) που κάνει τον υπολογισμό αυτόν. Η στατιστική ανάλυση σχετικά με την αποτίμηση του Κεφαλαίου ΤΧ για την αναδιαμόρφωση των Long Procedure φανέρωσε ότι κατά μέσο όρο κάθε περίπτωση απαιτεί 9,81 λεπτά για

αναδιαμορφωθεί. Η ελάχιστη τιμή είναι 1 λεπτό, η μέγιστη τιμή είναι 25, ενώ η τυπική απόκλιση είναι 6,71. Στην παρακάτω εικόνα απεικονίζετε το αντίστοιχο θηκόγραμμα.



Εικόνα 3-7: Θηκόγραμμα για το Κεφάλαιο ΤΧ του CO_2Capture για Extract Procedure

3.3.2 Αποτίμηση του ανασχεδιασμού Extract File / Module

Όσον αφορά τον ανασχεδιασμό του Extract File / Module, εντοπίσαμε χειροκίνητα 5 ευκαιρίες στον κώδικα του έργο Metalwalls και 6 στον κώδικα του CO_2Capture. Σημειώνουμε ότι εντοπίσαμε αυτές τις ευκαιρίες, βάσει του προβλήματος σχεδιασμού Large File / Module. Η απόφαση αυτή δεν προκαλεί προβλήματα στα αποτελέσματα μας, με την έννοια ότι ο χρόνος που απαιτείται για την εφαρμογή ενός refactoring, σχετίζεται μόνο με τους μηχανισμούς της ίδιας της αναδιάρθρωσης. Οι 5 ευκαιρίες του Metalwalls παρουσιάζονται στον Πίνακα 3-9, μαζί με το συνολικό Κεφάλαιο ΤΧ για κάθε αρχείο.

Πίνακας 3-9: Το Κεφάλαιο ΤΧ για τα αρχεία του MetalWalls

Source File	Principal (σε λεπτά)
output.F90	62
system.F90	20
species.F90	14

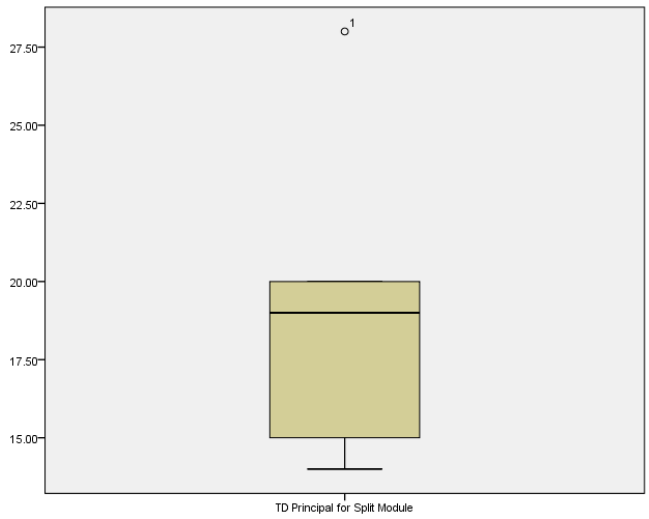
Ως παράδειγμα για τις παραπάνω περιπτώσεις, εστιάζουμε στο αρχείο System.F90. Σε αυτό το αρχείο έχει βρεθεί 1 ευκαιρία για την εφαρμογή ανασχεδιασμού. Στον Πίνακα 3-10, παρουσιάζουμε το Κεφάλαιο ΤΧ που απαιτείται για την επίλυση κάθε σχεδιαστικού προβλήματος Large File / Module. Στο Παράρτημα Β έχουν προστεθεί οι υπόλοιπες

πληροφορίες για όλα τα Extract File / Modules refactorings που εκτελέστηκαν μαζί με τους απαιτούμενους χρόνους.

Πίνακας 3-10: Ανάλυση του Κεφαλαίου TX για το αρχείο Output.F90

Large File/Module	Extracted File	Χρόνος Επίλυσης (σε λεπτά)
system.F90	input_data.F90	20

Η στατιστική ανάλυση για την αποτίμηση του Κεφαλαίου TX για την αναδιαμόρφωση των μεγάλων αρχείων / module φανερώνει ότι κατά μέσο όρο, κάθε περίπτωση απαιτεί 19,20 λεπτά για να αναδιαμορφωθεί. Η ελάχιστη τιμή είναι 14 λεπτά, η μέγιστη τιμή είναι 28, ενώ η τυπική απόκλιση είναι 5,54. Το θηκόγραμμα για την στατιστική ανάλυση απεικονίζεται στην Εικόνα 3-8.



Εικόνα 3-8: Θηκόγραμμα για το Κεφάλαιο TX του MetalWalls για Extract File / Module

Οι 6 ευκαιρίες που βρέθηκαν για το έργο CO_2Capture παρουσιάζονται στον Πίνακα 3-11, μαζί με το Κεφάλαιο TX για κάθε αρχείο. Ως παράδειγμα για αυτές τις περιπτώσεις, εστιάζουμε στο αρχείο reproducible.F90. Σε αυτό το αρχείο έχουν εντοπιστεί 2 ευκαιρίες για Extract File / Module. Στον Πίνακα 3-12, παρουσιάζουμε το Κεφάλαιο TX που απαιτείται για την επίλυση κάθε σχεδιαστικού προβλήματος Large File / Module. Οι επιπλέον ευκαιρίες για αναδιαμορφώσεις που πραγματοποιήθηκαν παραθέτουνε στο Παραρτημα Β.

Πίνακας 3-11: Το Κεφάλαιο TX για τα αρχεία του CO_2Capture

Source File	Principal (σε λεπτά)

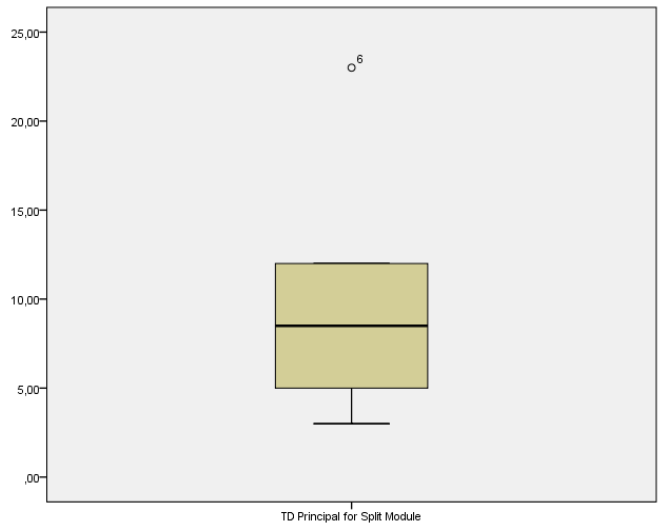
frpmodel.f	13
reproducible.F90	12
frpmainmn.f	12
funobj_real.f	23

Πίνακας 3-12: Ανάλυση του Κεφαλαίου TX για το αρχείο reproducible.F90

Large File/Module	Extracted File	Time to Resolve (σε λεπτά)
reproducible.F90	read_reproducible.F90	7
	write_reproducible.F90	5

Αυτό το συγκεκριμένο Extract File / Module του Πίνακα 3-12 προέκυψε από το γεγονός ότι εντοπίστηκαν δύο λειτουργίες με διαφορετικό σκοπό στο αρχικό module, η λειτουργία διαβάσματος reproducible και η εγράφη reproducible. Το αρχείο reproducible.F90 ήταν αρκετά μεγάλο σε μέγεθος γραμμών κώδικα, οπότε αποφασίσαμε να το χωρίσουμε σε 2 + 1 modules (δύο modules που αναφέρονται παραπάνω + ένα γενικό reproducible module που κάνει την κατάλληλη προετοιμασία).

Η στατιστική ανάλυση σχετικά με την αποτίμηση του Κεφαλαίου TX για την αναδιαμόρφωση Large Files / Modules υποδηλώνει ότι κατά μέσο όρο κάθε περίπτωση απαιτεί 10 λεπτά για να αναδιαμορφωθεί. Η ελάχιστη τιμή είναι 3 λεπτά, η μέγιστη τιμή είναι 23, ενώ η τυπική απόκλιση είναι 7,16. Στην Εικόνα 3-9 εμφανίζεται το θηκόγραμμα από την στατιστική ανάλυση.



Εικόνα 3-9: Θηκόγραμμα για το Κεφάλαιο TX του CO_2Capture για Extract File / Module

3.4 Τελική Αξιολόγηση Του Κεφαλαίου ΤΧ Σε Επίπεδο Σχεδίασης

Το τελικό βήμα αυτής της διαδικασίας είναι πολύ απλό, υπό την έννοια ότι αντιστοιχεί σε ένα σταθμισμένο άθροισμα των εμφανίσεων κάθε προβλήματος σχεδιασμού το οποίο πολλαπλασιάζεται με το κόστος για την επίλυση κάθε προβλήματος. Για να συνθέσουμε τα αποτελέσματα των δύο έργων σε έναν κοινό τύπο, εξετάζουμε πρώτα εάν υπάρχει στατιστικά σημαντική διαφορά στο μέσο χρόνο που απαιτείται για την επίλυση κάθε σχεδιαστικού προβλήματος στα δύο έργα. Και στις δύο περιπτώσεις οι διαφορές στα δύο έργα δεν είναι στατιστικά σημαντικές (μπορεί να ελεγχθεί και οπτικά με τα θηκογράμματα σε ζευγάρια). Επομένως, ως απαιτούμενος χρόνος για την επίλυση κάθε σχεδιαστικού προβλήματος, χρησιμοποιούμε τη μέση τιμή των δύο έργων (9,98 λεπτά για την αναδιαμόρφωση Extract Procedure και 14,60 για την αναδιαμόρφωση Extract File / Module). Για να μετατρέψουμε την προσπάθεια που απαιτείται για την επίλυση από λεπτά σε νόμισμα (δηλαδή, ευρώ) χρησιμοποιούμε τη μέση αμοιβή των υπαλλήλων των έργων που αναλύουμε, η οποία ανέρχεται στα 39,44 ευρώ ανά ώρα. Επομένως, το Technical Debt Design (TDD) principal μπορεί να υπολογιστεί με τον παρακάτω τρόπο (σε ευρώ), λαμβάνοντας υπόψη ότι το κόστος για την εφαρμογή της διαδικασίας Extract Procedure είναι 6,56 ευρώ, ενώ το κόστος για την εφαρμογή Extract File / Module είναι 9,59 ευρώ:

$$
\begin{aligned}
TDD_{Principal} = {} & \left(occurences_{long\ procedure} + occurences_{complex\ proceduce}\right) \\
& * cost_{extract\ procedure} \\
& + \left(occurences_{large\ file/module} + occurences_{overcouple\ file/module}\right) \\
& * cost_{extract_{file/module}} \\
= {} & \left(occurences_{long\ procedure} + occurences_{complex\ proceduce}\right) * 6.56 \\
& + \left(occurences_{large\ file/module} + occurences_{overcouple\ file/module}\right) \\
& * 9.59
\end{aligned}
$$

4 Αποπληρωμή Τεχνικού Χρέους

Σε αυτήν την ενότητα παρουσιάζουμε τις δύο προσεγγίσεις που προτείνουμε για τον αυτοματοποιημένο προσδιορισμό ευκαιριών για την εφαρμογή Extract Procedure και Extract File / Module. Οι δύο προσεγγίσεις είναι προσαρμοσμένες εκδόσεις των προσεγγίσεων που παρουσιάστηκαν αρχικά από τους Charalampidou et al. (2016) και Fokaefs et al. (2009).

4.1 Extract Procedure βάση της Αρχής της Μοναδικής Αρμοδιότητας

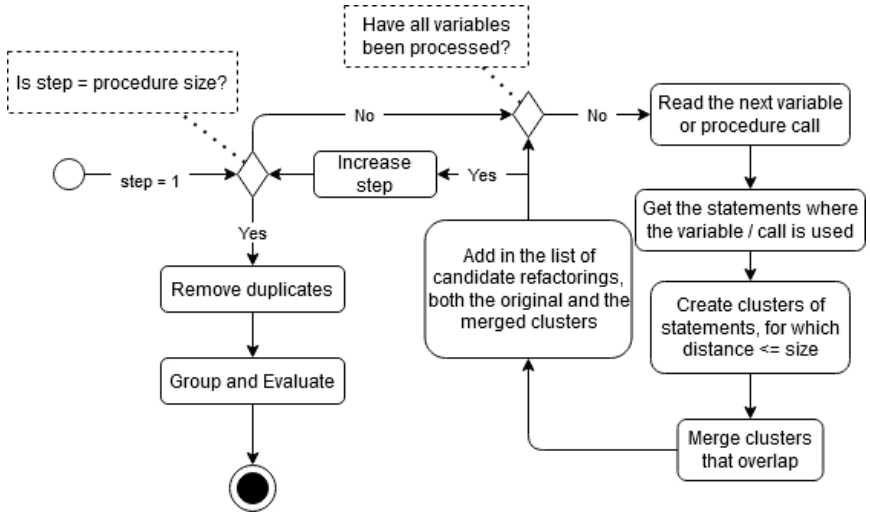
Η προσέγγιση που χρησιμοποιούμε για τον διαχωρισμό μιας μεγάλης σε μέγεθος διαδικασία βασίζεται στην αρχή της μοναδικής αρμοδιότητας (SRP) (Martin (2002)), η οποία εμπνεύστηκε από την αποσύνθεση της κάθε λειτουργίας σε ξεχωριστή μονάδας, που εισήγαγε ο De Marco (1979). Συγκεκριμένα, βασιστήκαμε στον τρόπο με τον οποίο το SRP έχει εφαρμοστεί από τους Charalampidou et al. (2016), που έκανε μια προσέγγιση για την δημιουργία του SRP Extract Method Identification (SEMI). Η προσέγγιση χρησιμοποίει τη σχέση μεταξύ τμημάτων κώδικα που συνεργάζονται για την ολοκλήρωση μιας λειτουργικότητας, αξιολογώντας τη συνοχή μεταξύ τους. Με τη μείωση του μεγέθους της μεθόδου μέσω εξαγωγής μεθόδου, η συνοχή αυξάνεται, οδηγώντας σε μια συνολικά πιο συμβατή με SRP μονάδα (Charalampidou et al. (2016)). Η διαδικασία SEMI αποσυντίθεται σε δύο κύρια μέρη: (α) προσδιορισμός ευκαιριών εξαγωγή μεθόδων, και (β) ομαδοποίηση και κατάταξη των ευκαιριών. Παρακάτω παρουσιάζουμε με μεγαλύτερη λεπτομέρεια την προσέγγιση, καθώς και τις ενημερώσεις που έχουμε κάνει. Καθώς αναλύουμε αρχεία κώδικα της γλώσσας C και FORTRAN, εφαρμόζουμε την προτεινόμενη διαδικασία σε διαδικασίες και όχι σε μεθόδους.

Εύρεση Ευκαιριών Extract Procedure: Το πρώτο μέρος της διαδικασίας SEMI, αφορά τον εντοπισμό συνεργαζόμενων διαδοχικών δηλώσεων, που παρέχουν μια συγκεκριμένη λειτουργικότητα στο σύστημα. Σύμφωνα με τους De Marco et al. (1979), η συνοχή μπορεί να συνδεθεί με τον αριθμό των διακριτών λειτουργιών για τις οποίες είναι υπεύθυνη μια ενότητα. Οι Charalampidou et al. επικεντρώθηκαν στη συνοχή σε επίπεδο μεθόδου και πιο συγκεκριμένα, στη συνοχή των διάφορων εντολών. Με βάση την προηγούμενη μελέτη, οι Charalampidou et al. χαρακτήρισαν δύο εντολές ως συνεκτικές με βάση την ικανοποίηση τουλάχιστον μιας από τις δύο προϋποθέσεις (2015). Τέλος

σημειώνουμε ότι οι παρακάτω ορισμοί έχουν υιοθετηθεί με συγκεκριμένο τρόπο ώστε να ταιριάζουν με τις ιδιαιτερότητες των γλωσσών C και FORTRAN (βλ. Ενότητα 3.2).

- *Πρόσβαση στην ίδια μεταβλητή.* Αυτή η συνθήκη βασίζεται στον ορισμό του Dallal για όλες τις μετρικές συνοχής μεθόδου-μεταβλητής (2010), όπου ο υπολογισμός της συνοχής βασίζεται στο κατά πόσο οι δύο μέθοδοι, ή εντολές στην περίπτωση μας, έχουν πρόσβαση στην ίδια μεταβλητή. Οι Charalampidou et al. (2016) στη μελέτη τους, κατηγοριοποίησαν ως μεταβλητές ενδιαφέροντος όλες οι μεταβλητές που είναι προσβάσιμες από το κύριο μέρος μιας μεθόδου (για παράδειγμα ιδιότητες, τοπικές μεταβλητές και παράμετροι της μεθόδου).
- *Κλήση της ίδιας μεθόδου.* Αυτή η συνθήκη βασίζεται σε διάφορους ορισμούς των μετρητών συνοχής, της Lack of Cohesion of Methods (Hitz and Montazeri (1995)), Tight Class Cohesion and Loose Class Cohesion (Bieman and Kang (1995)), Degree of Cohesion-Direct and Degree of Cohesion-Indirect (Badri (2004)). Ο λόγος για τον οποίο δύο δηλώσεις θεωρούνται συνεκτικές είναι ότι εκτελούν την ίδια λειτουργία (καλούν την ίδια διαδικασία), το οποίο υποδηλώνει ότι εξαρτώνται από την ίδια υπηρεσία, ανεξάρτητα από το αν εκτελούνται σε διαφορετικά δεδομένα. Αυτό υποστηρίζεται από τους Charalampidou er al. (2015) που διαπίστωσε ότι οι μετρικές Lack of Cohesion of Methods και Degree of Cohesion-Direct είναι πολύ αποδοτικές.

Βάση τα παραπάνω, το SEMI προσδιορίζει όλα τα πιθανά συνεκτικά σύνολα διαδοχικών εντολών, ακολουθώντας τη διαδικασία που φαίνεται στην Εικόνα 4-1.



Εικόνα 4-1: Διάγραμμα ροής για την εύρεση ευκαιριών Extract Procedure

Στη συνέχεια, παρουσιάζουμε ένα επεξηγηματικό παράδειγμα, για κάθε βήμα του γραφήματος για το Extract Procedure. Ο κώδικας που χρησιμοποιείτε στο παράδειγμα είναι μιας διαδικασίας από ένα αρχείο κώδικα της γλώσσας προγραμματισμού C, το οποί απεικονίζεται στην Εικόνα 4-2. Το πρώτο βήμα της διαδικασίας είναι να προσδιοριστούν οι προσβάσιμες στη διαδικασία μεταβλητές και οι κλήσεις διαδικασιών που γίνονται. Σημειώνουμε ότι η προσέγγιση λαμβάνει υπόψη μοναδικές προσβάσεις μεταβλητών ανα γραμμή, επομένως δεν καταγράφεται (και εν τέλει επηρεάζει) εάν μια μεταβλητή χρησιμοποιείται περισσότερες από μία φορές, σε μία μόνο γραμμή κώδικα. Το δεύτερο βήμα, είναι η δημιουργία ενός πίνακα που περιέχει τον αριθμό της γραμμής κώδικα μαζί με τη μεταβλητή ή διαδικασία που χρησιμοποιείτε σε αυτή.

```
void FileTraceLogWriter::write(const char* str, size_t len) {
    auto ptr = str;
    int remaining = len;
    bool needsResolve = false;

    while (remaining) {
        int ret = __write( traceFileFD, ptr, remaining );
        if (ret > 0) {
            lastError(0);
            remaining -= ret;
            ptr += ret;
            if (needsResolve) {
                issues->resolveIssue("trace_log_file_write_error");
                needsResolve = false;
            }
        } else {
            issues->addIssue("trace_log_file_write_error");
            needsResolve = true;
            printf(stderr, "Unx err [%d] flushing trace log.\n",
                   errno);
           lastError(errno);
           threadSleep(0.1);
        }
    }
}
```

Εικόνα 4-2: Κώδικας παραδείγματος

Κατά τη δημιουργία του πίνακα, πρέπει να έχουμε κατά νου τη χρήση δομών ελέγχου. Για τον λόγο ότι σε περίπτωση μιας συνθήκης ελέγχου else στην γραμμή 16 της Εικόνας 4-2 χρησιμοποιείτε έμμεσα η μεταβλητή της αρχικής συνθήκης ελέγχου. Επομένως, όπως είναι φανερό στον επόμενο πίνακα, η γραμμή 16 με την χρήση του ***else*** φαίνεται ότι χρησιμοποιεί την ίδια μεταβλητή με την γραμμή 8, όπου βρίσκετε το ***if***.

Πίνακας 4-1: Παράδειγμα χρήση μεταβλητών / διαδικασιών αν γραμμή

Αριθμός Γραμμών	Μεταβλητές / Διαδικασίες
2	prt; str
3	remaining; len
4	needsResolve
6	remaining
7	ret; __write; traceFileFD; ptr; remaining
8	ret
9	lastError
10	remaining; ret
11	prt; ret
12	needsResolve
13	resolveIssue
14	needsResolve
16	ret
17	addIssue
18	needsResolve
19	errno
20	lastError; errno
21	threadSleep

Με βάση τις πληροφορίες του παραπάνω πίνακα, το SEMI μπορεί να αρχίσει να κατηγοριοποιεί τις γραμμές κώδικα. Η διαδικασία ομαδοποίησης συνεχίζεται αυξάνοντας την απόσταση γραμμών κατά μία (με κάθε επανάληψη ομαδοποίησης). Στην περίπτωση που απαιτείται συγχώνευση, πραγματοποιείται με βάση την προσέγγιση συσσωματωμένης ιεραρχικής ομαδοποίησης (agglomerative hierarchical clustering) (Hitz and Montazeri (1995)). Η διαδικασία συνεχίζεται έως ότου επιτευχθεί ο μέγιστος αριθμός επαναλήψεων

(δηλαδή, το βήμα ισούται με το μέγεθος της διαδικασίας). Μετά την εξαγωγή όλων των πιθανών ευκαιριών, καταργούνται τυχόν μη έγκυρες και διπλές συστάδες.

Τροποποίηση του αλγορίθμου SEMI: Ένα από τα μειονεκτήματα της αρχικής προσέγγισης του αλγορίθμου SEMI ήταν ότι είναι αρκετά χρονοβόρα, ειδικά για τις περιπτώσεις στις οποίες τοποθετούνται οι εμφολευμένες συνθήκες ελέγχου ή βρόχοι επανάληψης. Επομένως, τροποποιήσαμε το SEMI, ώστε να επιτύχουμε μια ταχύτερη και πιο επεκτάσιμη έκδοση, ενσωματώνοντας τις συντακτικές προϋποθέσεις όπως αναφέρονται από τους Silva et al. (2014). Η προϋπόθεση που αναφέρεται, δηλώνει ότι τα επιλεγμένα κομμάτια κώδικα που προορίζονται για εξαγωγή πρέπει να αποτελούνται από πλήρη κομμάτια κώδικα. Για παράδειγμα στην Εικόνα 4-2, γνωρίζουμε ότι υπάρχει ένα μπλοκ if και αποτελείται από τις γραμμές 8-16. Εάν εξετάσουμε τις διαθέσιμες ομαδοποιήσεις, η ομαδοποίηση 12-14 είναι έγκυρη, καθώς περιέχεται στο μπλοκ if και δεν παραβιάζει την προϋπόθεση που αναφέρθηκε παραπάνω. Αυτό όμως δεν ισχύει για την ομαδοποίηση 6-11 όπου η προϋπόθεση παραβιάζεται, καθώς οι γραμμές 8-11 αποτελούν μέρος ενός μπλοκ κώδικα if. Επομένως η αλλαγή που κάνουμε είναι ότι ομαδοποιούμε μπλοκ κώδικα που δεν περιέχουν μέσα τους άλλα μπλοκ κώδικα, προκειμένου να βελτιωθεί η διαδικασία αναγνώρισης ευκαιριών. Για την επίλυση αυτού του προβλήματος, στην ενημερωμένη έκδοση, θεωρούμε όλες τις γραμμές εντός των συνθηκών ελέγχου / βρόχοι επανάληψης ως μία γραμμή. Στην καινούργια αυτή γραμμή ορίζεται ότι χρησιμοποιείται η ΄ένωση όλων των μεταβλητών / συναρτήσεων που ανήκαν στο μπλοκ αυτό. Λαμβάνοντας υπόψη τα παραπάνω, ο ενημερωμένος πίνας με την χρήση μεταβλητών / διαδικασιών ανά γραμμή παρουσιάζεται παρακάτω.

Πίνακας 4-2: Ενημερωμένο παράδειγμα χρήσης μεταβλητών / διαδικασιών αν γραμμή

Line Number	Accessed Variable / Called Procedures
2	prt; str
3	remaining; len
4	needsResolve
6	remaining
7	ret; __write; traceFileFD; ptr; remaining
8	ret
9	lastError

10	remaining; ret
11	prt; ret
12-15	needsResolve; resolveIssue
16-21	ret; addIssue; needsResolve; errno; lastError; threadSleep

Ομαδοποίηση και Ταξινόμηση των Ευκαιριών: Το δεύτερο μέρος του SEMI, αφορά την ομαδοποίηση και μετά την ταξινόμηση της λίστας των ευκαιριών που βρέθηκαν (δεδομένου ότι συνήθως είναι πάρα πολλές). Αυτή η φάση εξυπηρετεί το σκοπό της ομαδοποίησης των επικαλυπτόμενων ευκαιριών. Για παράδειγμα, εάν δύο ευκαιρίες μοιράζονται ένα σημαντικό ποσοστό των γραμμών κώδικα τους, θεωρείται ότι οι δύο αυτές παρέχουν σχεδόν την ίδια λειτουργικότητα. Ως αποτέλεσμα, οι δύο παραπάνω ευκαιρίες θεωρούνται πιθανές εναλλακτικές η μια για την άλλη. Για να ομαδοποιηθούν (σαν εναλλακτικές ομαδοποιήσεις), πρέπει να ακολουθηθεί μια διαδικασία. Ο αλγόριθμος ομαδοποίησης παρουσιάζεται σε απλά βήματα από τους Charalampidou et al. (2016).

Με τις υπόλοιπες τρεις παραμέτρους, γίνετε η να αξιολόγηση των ευκαιριών. Κάθε ευκαιρία έχει ένα όφελος, που αντιπροσωπεύει τη βελτίωση που κάνει στις μετρήσεις του κώδικα. Υπολογίζοντας το όφελος, μπορούν να ταξινομηθούν οι ευκαιρίες βάση το όφελος και να βρεθεί η καλύτερη. Η ταξινόμηση χρησιμοποιεί δύο μετρικές υπολογισμού του οφέλους (πρωτεύον και δευτερεύον). Ο τρόπος με τον οποίο επηρεάζουν το αποτέλεσμα παρουσιάζεται παρακάτω, αλλά πρώτα πρέπει να εξηγήσουμε ποιες είναι οι δύο μετρικές. Ως το πρωτεύων μετρική χρησιμοποιείται η μέτρηση συνοχής μιας διαδικασίας (LCOM2). Η επιλογή αυτής της διαδικασίας ως το πρωταρχική μετρική οφέλους κρίθηκε κατάλληλη για διάφορους λόγους που παρουσιάζει ο Charampidou et al. (2016). Το όφελος υπολογίζεται ως η μεταβολή της συνοχής, μετά την εφαρμογή της προτεινόμενης ευκαιρίας αναδιάρθρωσης. Ο τύπος είναι:

$$LCOM2_{Benefit} = LCOM2_{Original} - MAX(LCOM2_{opportunity}, LCOM2_{after\ refactoring})$$

Ως το δευτερεύον μετρική ο Charampidou επέλεξε το μέγεθος της διαδικασίας (σε αριθμό εντολών) αφού προτιμάται η εξαγωγή μεγαλύτερου αριθμού εντολών (της ίδιας συνοχής).

4.2 Extract File / Module με την χρήση του Agglomerative Clustering

Ο αλγόριθμος ομαδοποίησης που χρησιμοποιούμε για την αποσύνθεση αρχείων / modules είναι ο agglomerative αλγόριθμος, ένας τύπος ιεραρχικής ομαδοποίησης. Η

Ιεραρχική Ομαδοποίηση (Hierarchical Clustering) επιδιώκει να δημιουργήσει μια ιεραρχία συστάδων και βασίζεται στη ιδέα των οντοτήτων που σχετίζονται περισσότερο με γειτονικές οντότητες παρά με οντότητες που βρίσκονται πιο μακριά. Επομένως, αυτοί οι αλγόριθμοι συνδέουν οντότητες για να σχηματίσουν συστάδες με βάση την απόστασή τους. Ο αλγόριθμος Agglomerative Clustering μπορεί λειτουργεί ως εξής: Ως αρχικοποίηση εκχωρεί κάθε οντότητα σε ένα μόνο σύμπλεγμα. Σε κάθε επανάληψη, συγχωνεύει τις δύο συστάδες που έχουν την μικρότερη απόσταση μεταξύ τους. Τέλος, ο αλγόριθμος τερματίζεται όταν όλες οι οντότητες περιέχονται σε ένα σύμπλεγμα. Για να μπορέσουμε να αποφασίσουμε τις πραγματικές συστάδες, πρέπει να επιλέξουμε μια τιμή ορίου για την ελάχιστη απόσταση ως τιμή αποκοπής. Η ιεραρχία των συστάδων παρουσιάζεται συνήθως από ένα δενδρόγραμμα. Τα φύλλα του δέντρου αντιπροσωπεύουν τις οντότητες, η ρίζα είναι το τελικό σύμπλεγμα και οι ενδιάμεσοι κόμβοι είναι οι συστάδες που επιλέχτηκαν στα ενδιάμεσα βήματα. Το ύψος του δέντρου αντιπροσωπεύει τα διαφορετικά επίπεδα του ορίου απόστασης στο οποίο συγχωνεύθηκαν δύο συστάδες.

Υπάρχουν πολλές μέθοδοι για την επιλογή των κοντινότερων οντοτήτων. Στην περίπτωση μας επιλέξαμε τη μέθοδο Average Linkage, στην οποία η απόσταση μεταξύ ενός συμπλέγματος και ενός άλλου θεωρείται ίση με τη μέση απόσταση όλων των μελών ενός συμπλέγματος με όλα τα μέλη του άλλου συμπλέγματος. Όσον αφορά την τιμή ορίου (cut-off) για την ελάχιστη απόσταση, δεν ορίζουμε μια σταθερή, αλλά εφαρμόζουμε τον αλγόριθμο για ένα εύρος τιμών ορίου (από 0,1 έως 1,0) και παρουσιάζουμε τα αποτελέσματα. Παρατηρήσαμε ότι υψηλότερα όρια (που κυμαίνονται από 0,85 έως 1,0) τείνουν να παράγουν καλύτερα αποτελέσματα από τα χαμηλότερα όρια. Η μετρική για την απόσταση που επιλέξαμε να χρησιμοποιήσουμε είναι η απόσταση Jaccard, η οποία παράγει αξιοπρεπή αποτελέσματα. Για να καθορίσουμε την απόσταση Jaccard μεταξύ δύο διαδικασιών, χρησιμοποιούμε την έννοια των συνόλων των οντοτήτων. Το σύνολο οντοτήτων μιας διαδικασίας περιέχει όλες τις διαδικασίες (functions και subroutines) που καλούνται από αυτή, και όλες οι ιδιότητες που έχουν πρόσβαση σε αυτήν. Έτσι, αφού ορίσαμε την έννοια των συνόλων των οντοτήτων, υπολογίζουμε την απόσταση Jaccard μεταξύ δύο συνόλων οντοτήτων A και B όπως φαίνεται παρακάτω:

$$Jaccard\ Distance_{A,B} = 1 - \frac{|A\ \cap B|}{|A\ \cup B|}$$

Για την ευκολότερη κατανόηση της παραπάνω μεθοδολογίας παρουσιάζεται το παρακάτω παράδειγμα, όπου ο κώδικας φαίνεται στην Εικόνα 4-8. Το παράδειγμα είναι για module, αλλά η διαδικασία είναι η ίδια και για αρχεία.

```
module User
    implicit none
    private
    public :: change_email
    public :: get_email
    public :: init_user
    public :: delete_user
    public :: User_t

    type User_t
       integer :: id
       character(30) :: email
       character(30) :: password
       character(30) :: name
    end type User_t
 contains
   character(30) function get_email(this)
        implicit none
        type(User_t), intent(inout) :: this
        get_email = this%email
    end function get_email
    subroutine change_email(this, new_email)
       implicit none
       type(User_t), intent(inout) :: this
       character(30), intent(in) :: new_email

       this%name = new_email
    end subroutine change_email
    subroutine init_user(this, id, name, email, password)
        implicit none
        type(User_t), intent(inout) :: this
        integer, intent(in) :: id
        character(30), intent(in) :: name
        character(30), intent(in) :: email
        character(30), intent(in) :: password
        this%id = id
        this%name = name
        this%email = email
        this%password = password
    end subroutine init_user
    subroutine delete_user(this)
        implicit none
        type(User_t), intent(inout) :: this
        this%id = -1
        this%name = ""
        this%email = ""
        this%password = ""
    end subroutine delete_user
end module User
```

Εικόνα 4-3: Κώδικας παραδείγματος Agglomerative Clustering

Σε αυτό το παράδειγμα έχουμε ένα module με 4 ιδιότητες και 4 διαδικασίες:

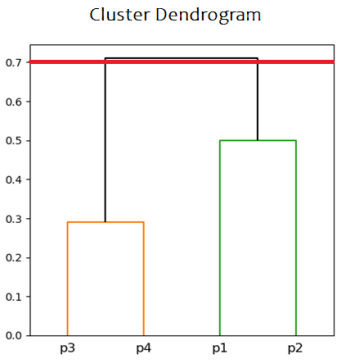
Πίνακας 4-3: Ιδιότητες και διαδικασίες παραδείγματος Agglomerative Clustering

Ιδιότητες	Διαδικασίες
id	change_email
email	get_email
password	init_user
name	delete_user

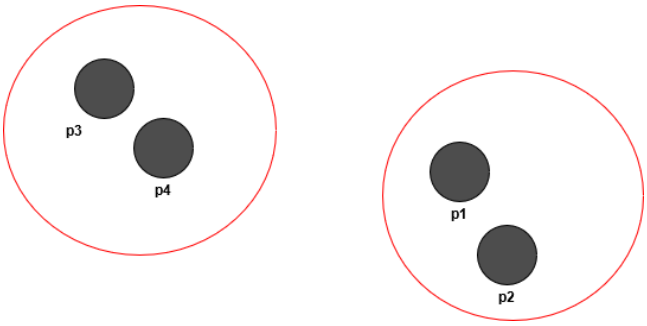
Στον Πίνακας 4-4 φαίνονται οι αποστάσεις για αυτό το παράδειγμα, και την Εικόνα 4-9 παρουσιάζεται το δενδρόγραμμα που παράγεται από τον αλγόριθμο ομαδοποίησης. Το δενδρόγραμμά μας βοηθά να καθορίσουμε τον αριθμό των συστάδων, δεδομένης μιας τιμής ορίου για την ελάχιστη απόσταση μεταξύ των οντοτήτων. Τέλος, η Εικόνα 4-10 δείχνει με γραφική αναπαράσταση της ενότητες μας και τις ομαδοποιήσεις. Οι τέσσερις κόμβοι αντιπροσωπεύουν τις διαδικασίες του module: υπάρχουν δύο διακριτές συστάδες που εμφανίζονται σε κύκλους. Καταφέραμε να εντοπίσουμε και τις δύο ομαδοποιήσεις χρησιμοποιώντας τον αλγόριθμο Agglomerative Clustering και μια τιμή κατωφλίου απόστασης 0,7. Σύμφωνα με τις δύο προϋποθέσεις που απαιτούνται για τη διασφάλιση ενός ορισμένου βαθμού λειτουργικότητας, το module που προτείνεται να εξαχθεί θα πρέπει: (α) να περιέχει τουλάχιστον μία διαδικασία και (β) να έχει άμεση πρόσβαση στις ιδιότητες του αρχικού module.

Πίνακας 4-4: Αποστάσεις διαδικασιών παραδείγματος

	p1	p2	p3
p1			
p2	0.5		
p3	0.71	0.71	
p4	0.71	0.71	0.29



Εικόνα 4-4: Δενδρόγραμμα παραδείγματος



Εικόνα 4-5: Γραφική απεικόνιση των διαδικασιών παραδείγματος

5 Επίδραση Βελτιστοποιήσεων Απόδοσης και Φορητότητας στο Τεχνικό Χρέος

Για να ερευνηθεί η σχέση μεταξύ: (α) της εφαρμογής βελτιστοποιήσεων απόδοσης / φορητότητας, και (β) την επίδρασή τους στη συντηρησιμότητα, πραγματοποιήσαμε μια μελέτη σε έξι έργα υψηλής υπολογιστικής ισχύος (τα οποία ανακτήθηκαν από τους παρόχους περιπτώσεων του EXA2PRO). Σε αυτήν την ενότητα περιγράφουμε το σχεδιασμό της μελέτης περίπτωσης και τα αποτελέσματα που βρέθηκαν. Ο λόγος για τη διεξαγωγή μελέτης περιπτώσεων ήταν ότι στοχεύσαμε στη διερεύνηση πραγματικών έργων που εφαρμόζουν βελτιστοποιήσεις απόδοσης και φορητότητας (όπως θα εκτελούσαν στην πράξη), χωρίς να γνωρίζουμε (ή να ελέγξουμε) τις συνέπειες στη συντήρηση.

5.1 Σχεδιασμός Μελέτης Περίπτωσης

Ο στόχος αυτής της μελέτης, περιγράφεται χρησιμοποιώντας τη διατύπωση Goal-Question-Metric (GQM) είναι: να αναλύσει τις βελτιστοποιήσεις φορητότητας και απόδοσης με σκοπό την εύρεση πιθανών αλλαγών στη συντήρηση του λογισμικού, στα πλαίσια των έργων μεγάλης κλίμακας. Αν και η έρευνα στον τομέα της διαχείρισης του τεχνικού χρέους (TDM) είναι πολύ αναπτυγμένη τα τελευταία χρόνια και έχουν προταθεί διάφορες μεθοδολογίες, εργαλεία και τεχνικές για την παρακολούθηση και τη διαχείριση του ΤΧ, η συσσώρευση και οι συνέπειες του ΤΧ μέσω της ανάπτυξης συστημάτων μεγάλης κλίμακας έχει δεν έχει διερευνηθεί στη βιβλιογραφία (Saarimaki et al. (2019)). Για την καλύτερη οργάνωση αυτής της μελέτης, λαμβάνουμε υπόψη το γεγονός ότι η χρήση προσεγγίσεων βελτιστοποίησης, όπως SkePU ή StarPU, οδηγεί είτε στην τροποποίηση (αναδιαμόρφωση) των υπαρχόντων αρχείων είτε στην προσθήκη νέου κώδικα. Ο λόγος για τη διαφοροποίηση των δύο τύπων αρχείων (refactored και new) είναι για να γίνει έλεγχος αν ο νέος κωδικός που εισάγεται για κατά την εφαρμογή της βελτιστοποίησης είναι διαφορετικός από τον refactored κώδικα (λόγω της εφαρμογής της βελτιστοποίησης). Οι διαφορές αυτές του ΤΧ μεταξύ νέου και refactored κώδικα έχουν συζητηθεί από τους Arvanitou et al. (2019). Με βάση τα παραπάνω, η έρευνα επικεντρώθηκε σε στις δύο κύριες έννοιες του ΤΧ, δηλαδή: κεφάλαιο και τόκο.

Τα δύο ερωτήματα που θα απαντηθούν σε αυτό το κεφάλαιο είναι:

- Ποια είναι η επίδραση της βελτιστοποίησης απόδοσης και φορητότητας στο τεχνικό χρέος, στα αναδιαμορφωμένα μέρη του πηγαίου κώδικα;
- Ποια είναι η επίδραση της βελτιστοποίησης απόδοσης και φορητότητας στο τεχνικό χρέος, στα νέα μέρη του πηγαίου κώδικα;

Για να απαντήσουμε σε αυτές τις ερωτήσεις, συγκρίνουμε τα επίπεδα του Κεφαλαίου και Τόκου ΤΧ των αρχείων που τροποποιούνται και εισάγονται κατά τη διαδικασία βελτιστοποιήσεων. Τα αποτελέσματα σχετικά με την απόδοση (SkePU) και τη φορητότητα (StarPU) αντιμετωπίζονται ξεχωριστά, καθώς συμμορφώνονται με διαφορετικούς κανόνες μετασχηματισμού. Με το πρώτο ερώτημα στοχεύουμε στη διερεύνηση του εάν οι τροποποιήσεις του πηγαίου κώδικα, λόγω της εφαρμογής βελτιστοποιήσεων, οδηγούν σε περισσότερο ή λιγότερο διατηρήσιμο πηγαίο κώδικα. Αντίστοιχα, με το δεύτερο στοχεύουμε στη διερεύνηση του εάν τα αρχεία που εισάγονται κατά τη διάρκεια του μετασχηματισμού παρουσιάζουν υψηλότερη ή χαμηλότερη συντήρηση σε σύγκριση με το μέσο όρο των υφιστάμενων αρχείων.

Σύμφωνα με τους Runeson et al. (2012), η μελέτη μας χαρακτηρίζεται ως ενσωματωμένη μελέτη πολλαπλών περιπτώσεων, καθώς ερευνούμε πολλές μονάδες ανάλυσης (δηλ. αρχεία) που εξάγονται από διάφορες περιπτώσεις (δηλ. έργα μεγάλης κλίμακας). Παρά την πληθώρα ανοιχτού κώδικα έργα μεγάλης κλίμακας, επιλέξαμε να περιορίσουμε την επιλογή περιπτώσεων σε έργα που:

- γνωρίζουμε το commit στο οποίο έχει πραγματοποιηθεί ένας μετασχηματισμός SkePU ή StarPU.
- γνωρίζουμε ότι στο προαναφερθέν commit, έχουν πραγματοποιηθεί περιορισμένες (ή καμία) άλλες αλλαγές (εκτός από τις βελτιστοποιήσεις).

Λόγω των περιορισμών αυτών, έπρεπε να αναφερθούμε σε συγκεκριμένους παρόχους εφαρμογών μεγάλης κλίμακας. Συγκεκριμένα, χρησιμοποιήσαμε έξι έργα που ανήκουν στο ευρωπαϊκό έργο EXA2PRO, τα οποία παρουσιάζεται στον Πίνακα 5-1. Εκτός από το όνομα του έργου, για κάθε ένα αναφέρουμε τις γλώσσες προγραμματισμού, το εργαλείο που χρησιμοποιήθηκε για τη βελτιστοποίηση, τον αριθμό των αρχείων πριν (β) και μετά (α) της βελτιστοποίησης καθώς και τον ιδιοκτήτη του έργου.

Πίνακας 5-1: Εξεταζόμενα έργα της μελέτης

Έργο	Γλώσσα Προγραμματισμού	Εργαλείο	#Αρχεία		Ιδιοκτήτης
			(b)	(a)	

CO_2Capture	Fortran	SkePU	25	35	CERTH[2]
Metalwalls	Fortran / C++	SkePU StarPU	42 41	42 42	CNRS[3]
Pastix	Fortran / C	StarPU	100	100	INRIA[4]
QR-mumps	Fortran / C	StarPU	91	102	JULICH[5]
Rodinia	C / C++	StarPU	4	14	LIU[6]
ParseC	C / C++	SkePU	36	65	LIU

Συλλογή δεδομένων και επεξεργασία. Για κάθε έργο έχουμε αναλύσει δύο εκδόσεις: πριν από τη βελτιστοποίηση και μετά τη βελτιστοποίηση. Για αυτές τις εκδόσεις καταγράψαμε τις μεταβλητές που φαίνονται παρακάτω για κάθε αρχείο.

- Number of Code Smells—NCS (Δείκτης για το Κεφάλαιο TX)
- Number of Functions—NOF (σχετίζεται με τον Τόκο TX)
- Complexity—CC (σχετίζεται με τον Τόκο TX)
- Lines of Code—LOC (σχετίζεται με τον Τόκο TX)
- Comments Ratio—CR (σχετίζεται με τον Τόκο TX)
- Fan-Out—FO (σχετίζεται με τον Τόκο TX)
- Lack of Cohesion of Lines—LCOL (σχετίζεται με τον Τόκο TX)

Στη συνέχεια, συγκρίνοντας τα ονόματα και τα μεγέθη των αρχείων στις εκδόσεις πριν και μετά, χαρακτηρίσαμε κάθε αρχείο ως: NEW (καινούργιο), REFACTORED (αλλαγμένο), ή UNCHANGED (ίδιο). Στη συνέχεια, πραγματοποιήσαμε τους ακόλουθους μετασχηματισμούς δεδομένων:

- για κάθε αρχείο REFACTORED, για κάθε μετρική, υπολογίζουμε τη διαφορά μεταξύ της έκδοσης πριν και μετά. Για να διασφαλιστεί η ομοιόμορφη ερμηνεία της διαφοράς, αποφασίσαμε τη σειρά της αφαίρεσης, έτσι ώστε οι αρνητικές διαφορές να αντιστοιχούν στο αρνητικό αποτέλεσμα του μετασχηματισμού και οι θετικές διαφορές στο θετικό αποτέλεσμα. Επομένως, σχετικά με την αναλογία

[2] www.certh.gr

[3] http://www.cnrs.fr/

[4] www.inria.fr

[5] www.fz-juelich.de

[6] https://liu.se/

σχολίων (CR), υπολογίσαμε το DIFF (διαφορά) ως ΜΕΤΑ-ΠΡΙΝ, ενώ για τα υπόλοιπα ΠΡΙΝ - ΜΕΤΑ.

- για κάθε αρχείο NEW, για κάθε μετρική, υπολογίζουμε πρώτα τη μέση τιμή της μετρικής στην προηγούμενη έκδοση. Στη συνέχεια, υπολογίζουμε τη διαφορά της μετρικής μεταξύ της μεταγενέστερης έκδοσης και της μέσης τιμής στην προηγούμενη έκδοση. Επομένως, όσον αφορά το CR, υπολογίσαμε το DIFF ως AFTER-MEAN, ενώ για τα υπόλοιπα ως MEAN-AFTER.

Στη συνέχεια, αναγνωρίζοντας την ανάγκη να συνθέσουμε τους έξι δείκτες του Τόκου ΤΧ σε μια μεταβλητή, βασιστήκαμε στο framework FItTeD και υπολογίσαμε τον Τόκο ΤΧ. Επομένως, το τελικό σύνολο δεδομένων της μελέτης περιέχει τις ακόλουθες μεταβλητές: [V1] Όνομα αρχείου; [V2] Εργαλείο βελτιστοποίησης (SkePU ή StarPU). [V3] Τύπος αρχείου (NEW, REFACTORED, ή UNCHANGED) [V4] Δείκτης για Κεφάλαιο ΤΧ (DIFF), και [V5] Δείκτης για τον Τόκο ΤΧ.

Ανάλυση δεδομένων. Ως πρώτο βήμα για τη διαδικασία ανάλυσης δεδομένων, πραγματοποιήσαμε μια στατιστική ανάλυση για τα δύο σύνολα δεδομένων (πριν και μετά τη βελτιστοποίηση) για όλες τις προαναφερθείσες μετρικές, για κάθε έργο. Στη συνέχεια, για να απαντήσουμε στις αρχικές ερωτήσεις, πραγματοποιήσαμε στατιστικούς ελέγχους υπόθεσης (t-tests) για να διερευνήσουμε εάν οι μεταβλητές [V4] και [V5] διαφέρουν από το μηδέν. Ώστε να διερευνήσουμε εάν το μέσο αποτέλεσμα της βελτιστοποίησης διαφέρει στατιστικά σημαντικά από το να μην έχει αποτέλεσμα (DIFF = 0,0). Πιο συγκεκριμένα, για την απάντηση στην πρώτη ερώτηση, φιλτράρουμε το σύνολο δεδομένων χρησιμοποιώντας την μεταβλητή [V3], επιλέγοντας μόνο αρχεία τύπου REFACTORED. Ενώ για την δεύτερη ερώτηση, διατηρήσαμε μόνο αρχεία τύπου NEW.

5.2 Αποτελέσματα

Σε αυτήν την ενότητα, παρουσιάζουμε τα αποτελέσματα της μελέτης περίπτωσης. Ως πρώτο βήμα για την ανάλυση, στον Πίνακα 5-2, παρέχουμε τα στατιστικά στοιχεία για όλες τις μετρικές που αναλύθηκαν. Πιο συγκεκριμένα, παρουσιάζουμε τις τιμές πριν και μετά από κάθε μετρική (σειρές), για κάθε έργο (στήλες), καθώς και το εργαλείο βελτιστοποίησης που χρησιμοποιήθηκε (πρώτη σειρά του πίνακα). Ο πίνακας αναφέρει μαζί τα REFACTORED και τα NEW αρχεία. Για κάθε ζεύγος τιμής πριν και μετά, επισημαίνουμε (έντονες γραμματοσειρές) την τιμή που είναι καλύτερη (για παράδειγμα η χαμηλότερη τιμή για την πολυπλοκότητα αλλά η υψηλότερη για την αναλογία σχολίων).

Από τα αποτελέσματα του Πίνακα 5-2, μπορούμε να παρατηρήσουμε ότι οι περιπτώσεις στις οποίες η πρώτη έκδοση είναι καλύτερη είναι 47% και οι περιπτώσεις που η δεύτερη έκδοση είναι καλύτερη είναι 53%. Στις προσεχείς υποενότητες παρουσιάζουμε μια λεπτομερή ανάλυση για να διερευνήσει: (α) εάν οι μέσοι όροι παρουσιάζουν στατιστικά σημαντικές διαφορές, (β) εάν εμφανίζονται οι ίδιες διαφορές για τον αλλαγμένο και τον νέο κώδικα μεμονωμένα, και (γ) τις διαφορές στον τόκο όταν συντίθενται όλες οι μετρικές μαζί.

Πίνακας 5-2: Περιγραφική στατιστική κάθε έργου

Tool		SkePU			StarPU			
Metrics		ParseC	CO_2Capture	Metalwalls		Pastix	QR-mumps	Rodinia
NCS	Before	***61.69***	219.40	***59.81***	***63.66***	***153.13***	***132.09***	233.00
	After	62.15	***165.97***	63.07	67.17	162.18	165.51	***45.57***
NOF	Before	***4.89***	***5.32***	5.69	5.78	***2.86***	***4.42***	19.00
	After	4.92	5.60	5.69	***5.67***	2.90	4.74	***4.36***
CC	Before	14.36	***37.84***	22.38	***23.44***	***25.92***	***17.60***	33.00
	After	***12.57***	39.40	***22.12***	23.45	26.40	18.74	***16.00***
LOC	Before	83.28	***375.32***	331.19	337.78	***145.03***	238.59	426.00
	After	***72.97***	413.66	***325.83***	***332.31***	147.36	***235.52***	***119.79***
CR	Before	***46.76***	***15.89***	12.89	***13.11***	55.75	22.85	5.77
	After	46.17	10.81	***14.58***	12.77	***55.99***	***24.76***	***41.27***
FO	Before	***1.44***	***0.46***	5.52	5.54	3.34	***2.75***	***0.00***
	After	1.70	0.63	***5.43***	***5.43***	***3.21***	3.48	1.64
LCOL	Before	113.91	4965.19	425.66	***476.81***	***1108.75***	***100.98***	290.23
	After	***100.67***	***4403.89***	***365.11***	581.66	1163.06	146.93	***49.12***

5.2.1 Επίδραση των Μετασχηματισμών SkePU / StarPU στον Refactored Κώδικα

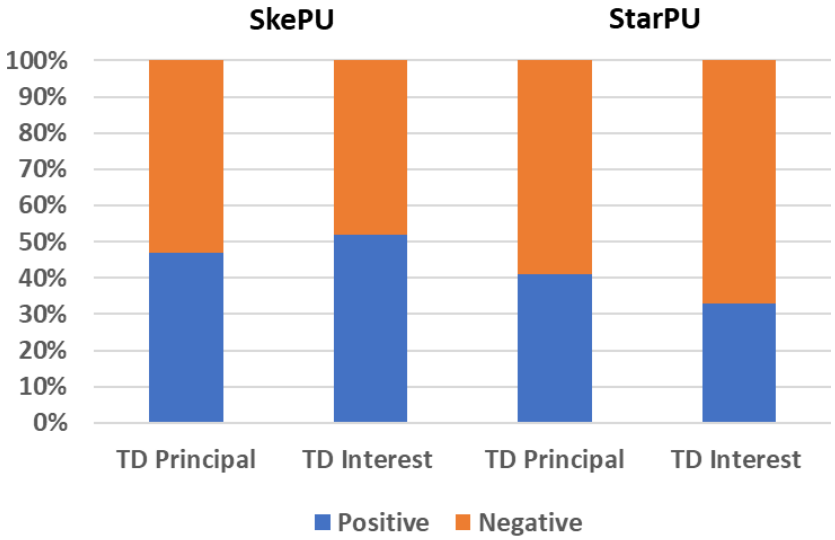
Στον Πίνακα 5-3, παρουσιάζουμε τα αποτελέσματα της ανάλυσης t-test στη μέση διαφορά της μεταβλητής πριν και μετά τις βελτιστοποιήσεις, από το μηδέν. Και οι δύο διαφορές υπολογίζονται πρώτα σε επίπεδο αρχείου και στη συνέχεια υπολογίζεται ένας μεγάλος μέσος όρος (σε όλα τα αρχεία όλων των έργων).

Πίνακας 5-3: Επίδραση του μετασχηματισμού στο TX για Refactored κώδικα

Βελτιστοποίηση	TX	Μέση Τιμή	t-value	sig.
SkePU	TD Principal	19.8%	2.591	*0.014*
	TD Interest	2.8%	0.989	0.336
StarPU	TD Principal	16.6%	3.256	*0.001*
	TD Interest	-5.2%	-2.397	*0.019*
** Across projects the range of difference values for principal and interest is [-390%, 200%] and [-230%, 160%]*				

Τα αποτελέσματα υποδηλώνουν ότι όλες οι μέσες διαφορές (εκτός από τον Τόκο TX για το StarPU) είναι θετικές, δηλαδή το Κεφάλαιο TX (TD principal) ή ο Τόκος TX (TD interest) έχει μειωθεί (δηλαδή, βελτιώθηκε) λόγω της βελτιστοποίησης. Αυτό το αποτέλεσμα είναι στατιστικά σημαντικό σε όλες τις περιπτώσεις, εκτός από τον Τόκο TX για SkePU. Ωστόσο, οι διαφορές (στήλη Μέσης Τιμής) σε απόλυτες τιμές φαίνεται να είναι μικρές για το Κεφάλαιο TX και οριακές για τον Τόκο TX.

Επιπλέων αναλύουμε τις προαναφερθείσες περιπτώσεις, όχι από την άποψη των μέσων τιμών των μετρήσεων, αλλά στις συχνότητες των refactored αρχείων. Στην Εικόνα 5-1 παρουσιάζουμε το ραβδόγραμμα σχετικά με τη συχνότητα των αρχείων που έχουν επηρεαστεί θετικά ή αρνητικά από τον μετασχηματισμό. Το αριστερό μέρος του σχήματος αναφέρεται στον μετασχηματισμό SkePU, ενώ το δεξί μέρος στο αποτέλεσμα του μετασχηματισμού StarPU. Συνδυάζοντας τα αποτελέσματα του Πίνακα 5-3 και της Εικόνας 5-1, μπορούμε να παρατηρήσουμε ότι όσον αφορά τη συχνότητα, οι μετασχηματισμοί φαίνεται να έχουν αρνητική επίδραση σε τουλάχιστον τα μισά αρχεία τόσο για SkePU όσο και για StarPU. Δεδομένου ότι για το Κεφάλαιο TX (στην περίπτωση του StarPU) η μέση τιμή (που περιλαμβάνει το 60% των αρνητικών αριθμών και το 40% των θετικών αριθμών) είναι θετική, μπορούμε να συμπεράνουμε σε απόλυτες τιμές, ότι οι θετικοί αριθμοί είναι υψηλότεροι από τους αρνητικούς. Αυτή η παρατήρηση οδηγεί στο συμπέρασμα ότι η θετική επίδραση του StarPU στο TX (όταν εμφανίζεται) είναι υψηλότερη (σε μέγεθος), σε σύγκριση με τις περιπτώσεις αρνητικού αποτελέσματος. Αυτή η παρατήρηση ισχύει επίσης σε όλες τις άλλες περιπτώσεις.



Εικόνα 5-1: Συχνότητα θετικής και αρνητικής επίδρασης

5.2.2 Επίδραση των Μετασχηματισμών SkePU / StarPU στον New Κώδικα

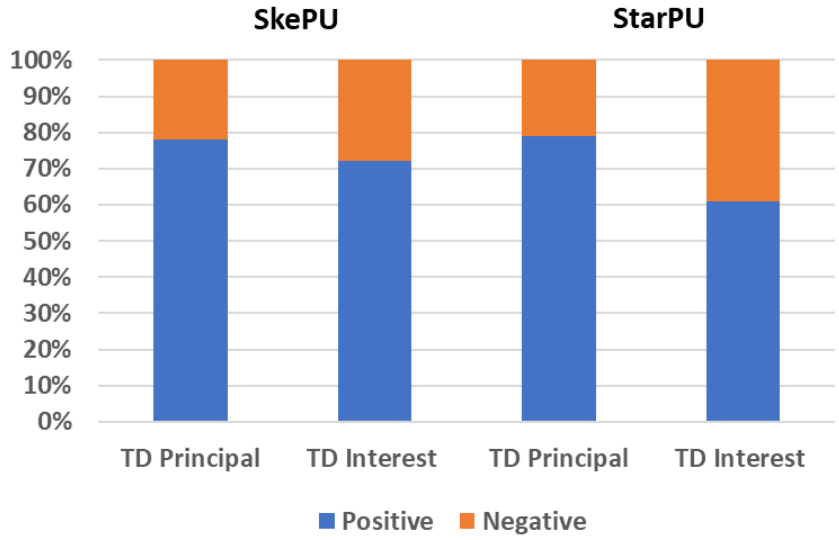
Ομοίως με την προηγούμενη υποενότητα, στον Πίνακα 5-4, παρουσιάζουμε τα αποτελέσματα της ανάλυσης t-test, που αξιολογεί το Κεφάλαιο και Τόκο του TX, σε νέα αρχεία που εισήχθησαν κατά την εκτέλεση μετασχηματισμών SkePU / StarPU, σε σύγκριση με τα υπόλοιπα αρχεία. Από τα αποτελέσματα του πίνακα, μπορούμε να παρατηρήσουμε ότι όλες οι μέσες διαφορές είναι θετικές (δηλαδή, οι μετασχηματισμοί βελτιώνουν το Κεφάλαιο και τον Τόκο του TX) και ότι όλες οι διαφορές είναι στατιστικά σημαντικές.

Πίνακας 5-4: Επίδραση του μετασχηματισμού στο TX για New κώδικα

Βελτιστοποίηση	TX	Μέση Τιμή	t-value	sig.
SkePU	TD Principal	34.7%	2.466	0.018
	TD Interest	103.7%	3.229	0.003
StarPU	TD Principal	35.0%	3.403	0.001
	TD Interest	31.6%	2.030	0.047
* Across projects the range of difference values for principal and interest is [-120%, 99%] and [-99%, 226%]				

Ακολουθώντας την ίδια στρατηγική όπως και πριν, στην Εικόνα 5-2 παρουσιάζουμε τη συχνότητα των νέων αρχείων που έχουν υψηλότερο ή χαμηλότερο Κεφάλαιο και Τόκο TX σε σύγκριση με το υπόλοιπο σύστημα κατά την εκτέλεση των μετασχηματισμών SkePU και StarPU. Η επίδραση του νέου κώδικα φαίνεται χαμηλότερη από τις μέσες τιμές και τις συχνότητες. Με βάση την Εικόνα 5-2, περίπου το 80% των αρχείων που εισάγονται μαζί με τον μετασχηματισμό SkePU έχουν χαμηλότερο Κεφάλαιο

και Τόκο TX, σε σύγκριση με τις μέσες τιμές του συστήματος. Μια παρόμοια παρατήρηση μπορεί να γίνει και για τα νέα αρχεία μετασχηματισμών τύπου StarPU, αλλά με χαμηλότερο ρυθμό, ειδικά για τον Τόκο TX, για το οποίο τα αποτελέσματα είναι οριακά.



Εικόνα 5-2: Συχνότητα θετικής και αρνητικής επίδρασης

5.3 Συζήτηση Αποτελεσμάτων

Σε αυτήν την ενότητα συνοψίζουμε τα κύρια ευρήματα αυτής της μελέτης, ερμηνεύοντας τα αποτελέσματα, συγκρίνοντάς τα με την υπάρχουσα βιβλιογραφία και παρέχοντας επιπτώσεις σε ερευνητές και επαγγελματίες. Ως πρώτο βήμα προς μια αποτελεσματική συζήτηση, στον Πίνακα 5-5, συνοψίζουμε τα κύρια ευρήματα της μελέτης μας.

Πίνακας 5-5: Περίληψη αποτελεσμάτων

Βελτιστοποίηση		TX	Συχνότητα επίδρασης	Επίδραση
SkePU	Refactored Code	TD Principal	Ίση	Strong for Positive Cases
		TD Interest	Ίση	Strong for Positive Cases
	New Code	TD Principal	**Θετική**	Strong
		TD Interest	**Θετική**	Strong
StarPU	Refactored Code	TD Principal	**Αρνητική**	Strong for Positive Cases
		TD Interest	**Αρνητική**	Strong
	New Code	TD Principal	**Θετική**	Strong
		TD Interest	**Θετική**	Limited

Ερμηνεία των αποτελεσμάτων. Λαμβάνοντας υπόψη την παραπάνω περίληψη των αποτελεσμάτων, σχετικά με το SkePU μπορούμε να ισχυριστούμε ότι δεν παρατηρούνται σημαντικές αντισταθμίσεις, με την έννοια ότι οι βελτιώσεις φορητότητας που προσφέρει το SkePU δεν βλάπτουν ουσιαστικά την συντηρησιμότητα του συστήματος από την μεριά του TX. Πιο συγκεκριμένα, τα αποτελέσματα υποδηλώνουν ότι ο νέος κώδικας που εισάγεται είναι καλύτερης ποιότητας σε σύγκριση με τον υπάρχοντα και για τον refactored κώδικα τα αποτελέσματα είναι ισορροπημένα (περίπου τα μισά αρχεία επηρεάζονται θετικά και τα άλλα μισά επηρεάζονται αρνητικά) . Επιπλέον, φαίνεται ότι τα αρχεία που επηρεάζονται θετικά επηρεάζονται πιο έντονα, καθιστώντας τη συνολική αξιολόγηση θετική και στατιστικά σημαντική. Παρ 'όλα αυτά, υπάρχει ακόμη περιθώριο βελτίωσης, ειδικά όσον αφορά τα αρχεία που έχουν αναδιαμορφωθεί, και ιδίως όσον αφορά τον τόκο που παράγουν. Ένα παράδειγμα του τρόπου με τον οποίο οι μετασχηματισμοί SkePU βελτιώνουν τον Τόκο TX παρουσιάζεται στην Εικόνα 5-3. Στο πάνω της εικόνας αντιστοιχεί ο πηγαίο κώδικα της υλοποίησης με SkePU, ενώ το κάτω μέρος χωρίς την βελτιστοποίηση.

```
float prod(float a, float b) {
      return a * b;
}
Vector<float> vector_prod(Vector<float> &v1, Vector<float> &v2) {
      auto vsum = Map<2>(prod);
      Vector<float> result(v1.size());
      return vsum(result, v1, v2);
}
```

```
float prod(float a, float b) {
      return a * b;
}
Vector<float> vector_prod(Vector<float> &v1, Vector<float> &v2) {
      Vector<float> result;
      for (int i=0; i<v1.size(); i++) {
          result.push_back(prod(v1[i], v2[i]));
      }
      return result;
}
```

Εικόνα 5-3: Παράδειγμα βελτιστοποίησης

Και οι δύο υλοποιήσεις έχουν τις ίδιες μετρικές CR, FO, NOF και LCOL. Αλλά η κυκλωματική πολυπλοκότητα (CC) του κώδικα με την εφαρμογή του SkePU είναι 1, ενώ για τον κώδικα χωρίς την βελτιστοποίηση το CC ισούται με 2. Επίσης, το LOC του κώδικα χωρίς την εφαρμογή του SkePU είναι μεγαλύτερο κατά 1 γραμμή. Επομένως, ο Τόκος TX χωρίς την χρήση του SkePU είναι υψηλότερο.

Αφ' ετέρου όσον αφορά το StarPU, οι αντισταθμίσεις είναι πιο εμφανείς: η χρήση του StarPU εγγυάται την καλύτερη απόδοση του συστήματος, αλλά φαίνεται να βλάπτει τη συντηρησιμότητα των αναδιαμορφωμένων αρχείων (πάνω από το 60% από αυτά). Ωστόσο, τα νέα αρχεία που προστίθενται φαίνεται να έχουν καλύτερα επίπεδα Κεφαλαίου και Τόκου TX σε σύγκριση με τον υπόλοιπο κώδικα. Επομένως, σε αυτό το είδος μετασχηματισμού υπάρχει και πάλι περιθώριο βελτίωσης, δίνοντας ιδιαίτερη έμφαση στον refactored κώδικα και στο παραγόμενο τόκο. Η ύπαρξη ανταλλάγματος μεταξύ βελτιστοποιήσεων χρόνου εκτέλεσης και συντηρητικότητας είναι ένα αναμενόμενο αποτέλεσμα, με την έννοια ότι η βιβλιογραφία έχει ήδη αναφέρει παρόμοια ευρήματα (Barney et al. (2012), Buyens et al. (2009), Feitosa et al. (2015)). Επίσης είναι σημαντικό να τονίσουμε ότι ο Τόκος TX φαίνεται να διαχειρίζεται λίγο πιο δύσκολα και φαίνεται πιο ευάλωτος σε αντισταθμίσεις σε σύγκριση με το Κεφάλαιο TX. Μια εξήγηση σχετικά με αυτό είναι το γεγονός ότι ο Τόκος TX υπολογίζεται ως μια συλλογή συνήθως αντικρουόμενων μετρικών (όπως σύζευξη και συνοχή, μέγεθος και πολυπλοκότητα κ.λπ.) σε αντίθεση με το Κεφάλαιο TX.

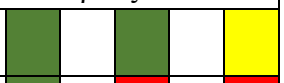
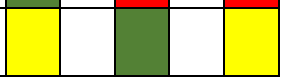
Επιπτώσεις για τους επαγγελματίες. Συγκεκριμένα, θεωρώντας ότι η βελτίωση των χαρακτηριστικών ποιότητας χρόνου εκτέλεσης είναι ένας στόχος που δεν είναι διαπραγματεύσιμος σε εφαρμογές HPC, μπορούμε να επισημάνουμε τις πιο συχνές παγίδες, ενώ εφαρμόζουμε μετασχηματισμούς SkePU και StarPU, έτσι ώστε οι προγραμματιστές να τους έχουν υπόψιν και να τις αποφεύγουν σε μελλοντικές εφαρμογές.

Στον Πίνακα 5-6, παραθέτουμε τους πιο συχνούς τύπους Στοιχείων Τεχνικού Χρέους (παραβιάσεις κανόνων SonarQube στα εξεταζόμενα έργα) που εισάγονται από μετασχηματισμούς SkePU / StarPU. Παραθέτουμε όλες τις παραβιάσεις των κανόνων που εισάγονται στον νέο κώδικα και τις παραβιάσεις των κανόνων που έχουν εισαχθεί στα αναδιαμορφωμένα αρχεία. Συγκεκριμένα, τις παραβιάσεις των κανόνων που εμφανίζονται στις 10 πιο συχνές από κάθε έργο (με κόκκινη σκίαση), στις 20 πιο συχνές από κάθε έργο (με κίτρινη σκίαση) και αυτές που συναντιούνται λιγότερο συχνά σε κάθε έργο (με πράσινη σκίαση). Το κριτήριο για να συμπεριληφθεί ένας κανόνας στον πίνακα ήταν η

εμφάνισή του σε τουλάχιστον 3 έργα. Με βάση τα ευρήματα του Πίνακα 5-6, μπορούμε να ενθαρρύνουμε τους ασκούμενους να προσπαθήσουν να αποφύγουν τις παραβιάσεις των κανόνων ""“Magic Number”, “Missing Curly Braces”, “Missing Include File”, “String Literal Duplicated”, και “Undocumented API” για μετασχηματισμούς StarPU και SkePU σε κώδικα C / C ++ και οι παραβιάσεις των κανόνων “Float Compare”, “Check Code Return”, και “Exit Loop” κατά την εφαρμογή βελτιστοποιήσεων απόδοσης και φορητότητας σε κώδικα Fortran.

Πίνακας 5-6: Συχνότητα code smells

Code Smell	Πρότζεκ					
c:ClassName						
c:CommentedCode						
c:FileHeader						
c:FunctionCognitiveComplexity						
c:FunctionComplexity						
c:FunctionName						
c/cxx:MagicNumber						
c/cxx:MissingCurlyBraces						
c/cxx:MissingIncludeFile						
c:ReservedNames						
c/cxx:StringLiteralDuplicated						
c/cxx:TabCharacter						
c:TooLongLine						
c/cxx:TooManyParameters						
c:TooManyStatementsPerLine						
c/cxx:UndocumentedApi						
common-c:DuplicatedBlocks						
common-c:InsufficientCommentDensity						
common-c:InsufficientLineCoverage						

Code Smell	Πρότζεκ
F-rules:COM.DATA.FloatCompare	
F-rules:COM.FLOW.CheckCodeReturn	
F-rules:COM.FLOW.ExitLoop	

Επιπτώσεις για τους ερευνητές. Όσον αφορά τους ερευνητές, μπορούν να βρεθούν πολλές ενδιαφέρουσες μελλοντικές ευκαιρίες εργασίες. Η πιο ενδιαφέρουσα μελλοντική κατεύθυνση που σκοπεύουμε να ακολουθήσουμε είναι να ερευνήσουμε περαιτέρω τα αποτελέσματα αυτής της μελέτης και να αποκαλύψουμε τους λόγους για τους οποίους εισάγονται τα code smells που παρουσιάζονται στον Πίνακα 5-6. Πιθανές εξηγήσεις για αυτό θα μπορούσαν να είναι: (α) η γενική συχνότητα αυτών των κανόνων, (β) η κουλτούρα ή οι ιδιαιτερότητες των ομάδων, ή (γ) οι ιδιαιτερότητες των μετασχηματισμών των δύο εργαλείων. Επιπλέον, ενθαρρύνουμε την περαιτέρω διερεύνηση των σχεδιαστικών επιπτώσεων της χρήσης του StarPU και του SkePU, καθώς έχουν ιδιαίτερη επίδραση στον τόκο.

5.4 Απειλές Για Την Εγκυρότητα

Τα αποτελέσματα που παρουσιάσαμε για αυτή την έρευνα υπόκεινται στην απειλή της γενίκευσης. Η αδυναμία που υπάρχει για την γενίκευση των αποτελεσμάτων μας αφορά τόσα τα έργα που αναλύθηκαν όσο και τα εργαλεία βελτιστοποίηση. Με άλλα λόγια, δεν μπορούμε να υποστηρίξουμε ότι οι βελτιστοποιήσεις StarPU και SkePU θα μειώσουν το Κεφάλαιο TX σε κάθε έργο λογισμικού μεγάλης κλίμακας, καθώς στην μελέτη χρησιμοποιήθηκαν μόνο έξι έργα από συγκεκριμένους πάρωρους. Παρομοίως, δεν μπορεί να υποστηριχθεί ότι οποιοδήποτε άλλο είδος βελτιστοποίησης απόδοσης ή φορητότητας, πέραν αυτών που εφαρμόζονται από το SkePU και StarPU θα οδηγήσει σε παρόμοια αποτελέσματα. Απαιτείται περαιτέρω έρευνα για την επικύρωση αυτών των ευρημάτων και για τη διερεύνηση των λόγων που οι βελτιστοποιήσεις αυτές επηρεάζουν τις μετρικές του λογισμικού.

Όσον αφορά τις απειλές εγκυρότητάς των αποτελεσμάτων, πρέπει να τονίσουμε ότι για να εκτιμηθεί το αντίκτυπο στο κεφάλαιο και στον τόκο του τεχνικού χρέους, χρησιμοποιήθηκε ένα συγκεκριμένο εργαλείο (SonarQube) και επιλεγμένες μετρικές σχεδίασης. Ένα αρνητικό του SonarQube είναι ότι δίνει λιγότερη έμφαση στην αρχιτεκτονική του συστήματος που εξετάζει. Ωστόσο, λόγο της φύσης των βελτιστοποιήσεων απόδοσης και φορητότητας, δεν προβλέπεται να υπάρχουν ιδιαίτερες

αλλαγές στην αρχιτεκτονική του συστήματος. Για τον τόκο του τεχνικού χρέους βασιστήκαμε σε δείκτες, καθώς η ποσοτικοποίηση του τόκου είναι δύσκολη. Αυτό μπορεί να συνέβαλε στα σχετικά ασυνεπή αποτελέσματα του τόκου, ωστόσο οι δείκτες που χρησιμοποιήθηκαν είναι ευρέως αναγνωρισμένα από την κοινότητα σαν δείκτες συντηρησιμότητας.

6 Ανάπτυξη Εργαλείου

Σε αυτήν την ενότητα, παρουσιάζουμε το εργαλείο που έχει αναπτυχθεί για τον υπολογισμό και τη διαχείριση ΤΧ / μετρικών για εφαρμογές μεγάλης κλίμακας. Αυτό το εργαλείο υποστηρίζει τις γλώσσες προγραμματισμού: FORTRAN (εκδώσεις 77 και 90), C, και C++.

6.1 Εργαλεία Που Χρησιμοποιήθηκαν Για Την Υλοποίηση

Η Java παρουσίασε την βιβλιοθήκη Swing στην βασική έκδοση της Java από την Java 1.2. Η Swing είναι ένα API που παρέχει στους προγραμματιστές έναν εύκολο τρόπο να δημιουργούν γραφικά στοιχεία, χρησιμοποιώντας εύκολα προσαρμόσιμα στοιχεία. Παρόλο που στις μέρες μας υπάρχει μια πιο καινούργια προσέγγιση που προσφέρει μια πιο σύγχρονη τελική εμφάνιση GUI, μέσω τις εξωτερική βιβλιοθήκη του JavaFX, για λόγους απλότητας επιλέξαμε την Swing. Επιπλέον το περιβάλλον που χρησιμοποιήθηκε για την υλοποίηση ήταν το Netbeans, το οποίο ανήκει στην Apache, και το πρόγραμμα μας χρησιμοποιεί το εργαλείο αυτοματοποίησης κατασκευής Maven. Τέλος, επειδή το εργαλείο μας είναι απλώς μια εφαρμογή Java, έχει δημιουργηθεί με τέτοιο τρόπο ώστε να μπορεί να εκτελείται σε υπολογιστές Windows και Linux με το ίδιο εξαγόμενο αρχείο .jar.

6.2 Εξαρτήσεις Εργαλείου

Το εργαλείο μας εξαρτάται από τρία διαφορετικά προγράμματα, τα οποία πρέπει να έχουν ήδη εγκατασταθεί στο μηχάνημα του τελικού χρήστη. Τα προγράμματα είναι SonarQube με εγκατεστημένα δύο plugins, το Apache Maven, και τέλος την γλώσσα προγραμματισμού Python.

Η ανάλυση ενός έργου θα μπορούσε να χωριστεί σε δύο διαφορετικές υπο-αναλύσεις: 1) τον προσδιορισμό των θεμάτων του κώδικα και τον υπολογισμό του τεχνικού χρέους, και 2) τον υπολογισμό των μετρικών κάθε αρχείου. Το SonarQube μπορεί να αναλύσει έργα γραμμένα στη γλώσσα προγραμματισμού Fortran, C και C ++ μόνο μετά την εγκατάσταση των αντίστοιχων plugin. Τα plugin αυτά είναι το iCode CNES και SonarQube C ++. Μετά την εγκατάσταση, για την εκτέλεση μιας ανάλυσης από το εργαλείο μας πρέπει να έχει ξεκινήσει το SonarQube πιο πριν. Τέλος, την πρώτη φορά που εκτελείται το εργαλείο, πρέπει να διαμορφωθεί από τον χρήστη με την εγκατάσταση του στο SonarQube.

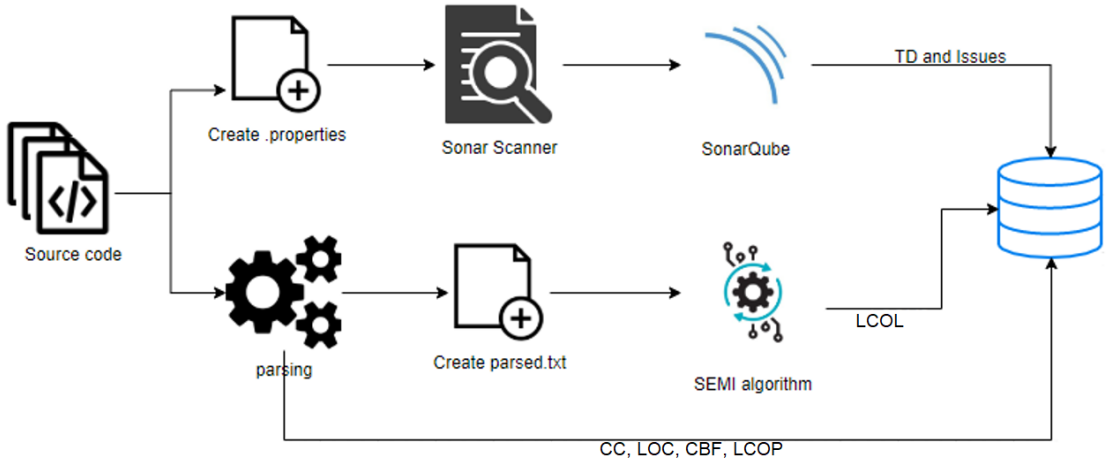
Η τιμή του SonarQube (ή οποιουδήποτε εργαλείου που εκτιμά το τεχνικό χρέος με βάση την προσπάθεια διόρθωσης θεμάτων του κώδικα) μπορεί να αυξηθεί εάν ο χρόνος για την επίλυση μιας παραβίασης κανόνα μπορεί να προσαρμοστεί στα μέτρα της κάθε ομάδας ανάπτυξης λογισμικού. Για το σκοπό αυτό, έχουμε προσθέσει μια δυνατότητα έτσι ώστε για την ανάλυση του κώδικα Fortran με τη βοήθεια του plugin iCode CNES, ο χρήστης μπορεί να διαμορφώσει τον χρόνο επιδιόρθωσης για κάθε τύπο προβλήματος (για το plugin SonarQube C ++ αυτή η λειτουργικότητα είναι ήδη διαθέσιμη).

Η αρχικές τιμές για την επίλυση των παραβιάσεων των κανόνων για την γλώσσα προγραμματισμού FORTRAN είναι οι τιμές που βρέθηκαν από την έρευνα που έγινε στα πλαίσια του ευρωπαϊκού έργου EXA2PRO. Όπου μέσα από ένα ερωτηματολόγιο σε διάφορες ομάδες προγραμματισμού μεγάλης κλίμακας έργων βρέθηκε ο μέσω χρόνος επίλυσης για κάθε παραβίαση κανόνα. Η αλλαγή του χρόνου αποκατάστασης των κανόνων απαιτεί την αλλαγή και έπειτα ανακατασκευή του plugin iCode CNES. Δεδομένου ότι το plugin χρησιμοποιεί το εργαλείο αυτοματοποίησης Maven, μια εξωτερική εξάρτηση για το εργαλείο μας είναι το εργαλείο Apache Maven, ώστε να μπορέσει να αυτοματοποιηθεί η όλη διαδικασία.

Τέλος, ο υπολογισμός αναδιαμορφώσεων τύπου Extract File / Module ο οποίος αναφέρθηκε στα προηγούμενα κεφάλαια γίνεται με την βοήθεια του αλγορίθμου Agglomerative Clustering. Ο αλγόριθμος αυτός δημιουργήθηκε στην γλώσσα προγραμματισμού Python, όπου υπάρχει πληθώρα έτοιμων βιβλιοθηκών για την υλοποίηση που χριζόμαστε.

6.3 Ροή Εργασιών Για Ανάλυση Έργων

Όταν ένας χρήστης δημιουργεί ένα νέο έργο για ανάλυση ή επιθυμεί να αναλύσει μια καινούργια έκδοση, το εργαλείο ακολουθεί τα ίδια βήματα. Πρώτα απ’ όλα, το εργαλείο εντοπίζει όλα τα αρχεία πηγαίου κώδικα στον φάκελο και τους υποφακέλους του έργου. Στη συνέχεια, η κύρια ροή εργασίας χωρίζεται σε δύο μέρη: (α) τον υπολογισμό του τεχνικού χρέους με το SonarQube και (β) τον υπολογισμό των μετρικών των αρχείων και διαδικασιών. Τα λεπτομερή βήματα απεικονίζονται στην Εικόνα 6-1 παρακάτω.



Εικόνα 6-1: Ροή εργασιών για την ανάλυση έργου

Για τον υπολογισμό του τεχνικού χρέους (principal) και τις παραβιάσεις κανόνων ενός έργου το εργαλείο μας εκτελεί τα εξής:

- Για κάθε ανάλυση SonarQube, πρέπει να συντάσσεται ένα αρχείο "sonar-project.properties". Το εργαλείο μας δημιουργεί το αρχείο αυτό αυτόματα με τις καταχωρίσεις "sonar.projectKey" και "sonar.projectName" με το όνομα του έργου, "sonar.projectVersion", το οποίο για κάθε νέα έκδοση αυξάνεται κατά ένα και "sonar.sources" όπου ορίζεται ο αρχικός φάκελος του έργου. Σε περίπτωση που το έργο χρησιμοποιεί τη γλώσσα προγραμματισμού FORTRAN, εισάγονται δύο επιπλέον καταχωρήσεις για να μπορέσει να ξεκινήσει το plugin iCode και να εντοπιστεί η θέση του προγράμματος γραμμής εντολών που πρέπει να υπάρχει στον τοπικό υπολογιστή του χρήστη.
- Για να πραγματοποιηθεί μια ανάλυση στο SonarQube, απαιτείται ένας σαρωτής από την μεριά του χρήστη. Το εργαλείο μας συνοδεύεται από το SonarScanner 4.2 για Windows και Linux και η κατάλληλη εντολή εκτελείται στο σύστημα για να ξεκινήσει η ανάλυση αυτόματα.
- Με το τέλος της ανάλυσης, τα αποτελέσματα του σαρωτή μεταφέρονται στο SonarQube από όπου το εργαλείο μας έχει πρόσβαση στη μέτρηση του τεχνικού χρέους και τις παραβιάσεις των κανόνων, μέσω των κατάλληλων κλήσεων API.

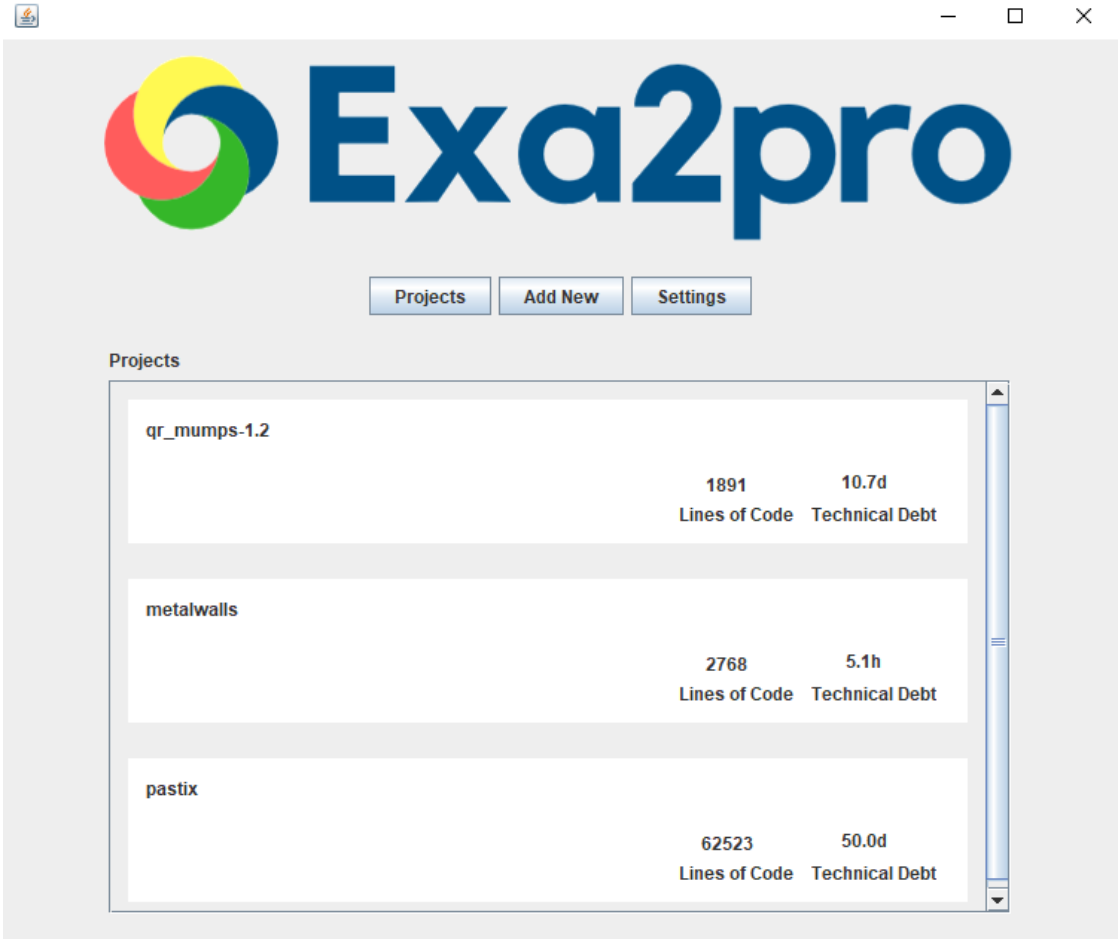
Για τον υπολογισμό των μετρικών σύζευξης (CBF), γραμμών κώδικα (LOC) και της έλλειψης συνοχής των διαδικασιών (LCOP) για όλα τα αρχεία και τις μετρικές κυκλωματικής πολυπλοκότητας (CC) και έλλειψη συνοχής των γραμμών (LCOL) για όλες τις διαδικασίες, ακολουθούνται τα ακόλουθα βήματα:

- Όλα τα αρχεία αναλύονται και υπολογίζονται οι μετρήσεις των CC, LOC, CBF και LCOP.
- Με τις αναλυμένες πληροφορίες, δημιουργούνται επιπλέων αρχεία .txt για κάθε αρχείο του έργου. Αυτό το αρχείο είναι ένα αρχείο κειμένου που περιέχει πληροφορίες από τον πηγαίο κώδικα, όπου κάθε γραμμή έχει τις μεταβλητές ή διαδικασίες που χρησιμοποιούνται στην αντίστοιχη γραμμή κώδικα.
- Τα αρχεία που δημιουργήθηκαν στο προηγούμενο βήμα είναι σε αυτή την συγκεκριμένη μορφή η οποία είναι απαραίτητη από τον αλγόριθμο SEMI. Το SEMI χρησιμοποιείτε για την μετρική της έλλειψης συνοχής όπου για κάθε αρχείο υπολογίζετε με βάση την ομοιότητα μεταξύ διαδοχικών εντολών.

6.4 Γραφικό Περιβάλλον Χρήστη

Το κύριο σκεπτικό πίσω από την ανάπτυξη του γραφικού περιβάλλοντος ήταν η παροχή ενός εύχρηστου και αυτονόητου εργαλείου που θα απαιτούσε πολύ περιορισμένη επένδυση χρόνου για την κατανόηση και χρήση του. Ώστε να μην είναι δύσκολου για άτομα που είναι εξοικειωμένα στον προγραμματισμό λογισμικού HPC. Στην Εικόνα 6-2 παρουσιάζουμε την κύρια οθόνη του εργαλείου μας. Η κύρια οθόνη περιλαμβάνει δύο κύρια στοιχεία:

- Το panel με τα κουμπιά, όπου υπάρχει τον Projects, Add New and Settings.
- Το panel με τα έργα, το οποίο δείχνει σε μία λίστα όλα τα έργα που έχουν αναλυθεί και είναι αποθηκευμένα τοπικά. Σε κάθε έργο φαίνεται το όνομα, οι γραμμές κώδικα, και το τεχνικό χρέος (σε λεπτά, ώρες ή μέρες).

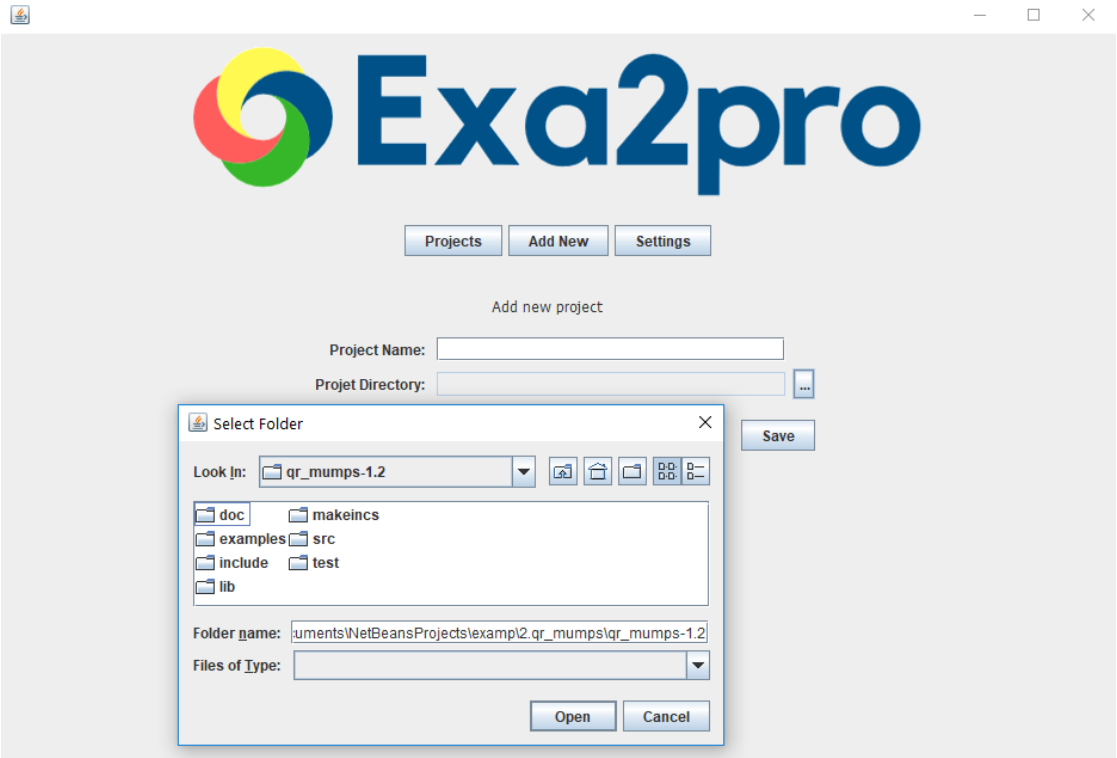


Εικόνα 6-2: Κύρια οθόνη του εργαλείου

6.4.1 Εισαγωγή Έργου

Για την ανάλυση ενός νέου έργου, ο χρήστης πλοηγείτε στο panel Add New, όπου θα πρέπει να δώσει ένα όνομα έργου και την θέση στην οποία βρίσκετε στον τοπικό υπολογιστή, όπως φαίνεται στην Εικόνα 6-3. Το όνομα που θα δοθεί θα χρησιμοποιηθεί εσωτερικά ως το όνομα του έργου, αλλά και για SonarQube ως sonar.projectName και sonar.projectKey.

Ο χρήστης μπορεί να αναλύσει μια επόμενη έκδοση ενός έργου (για να ανακτήσει δεδομένα εξέλιξης του έργου), ανοίγοντας αυτό το έργο, επιλέγοντας το panel Manage Project και τέλος το κουμπί "Ανάλυση" (Analyze). Για απλότητα, υποθέτουμε ότι οποιαδήποτε μελλοντική έκδοση αυτού του έργου θα βρίσκεται στην ίδια τοποθεσία φακέλου με το αρχικό. Με κάθε νέα ανάλυση, η έκδοση του έργου αυξάνεται κατά μία (και επισημαίνεται ως τρέχουσα), ενώ οι μετρήσεις των προηγούμενων εκδόσεων αποθηκεύονται ως ιστορικές, και χρησιμοποιούνται μόνο στον πίνακα εξέλιξης.



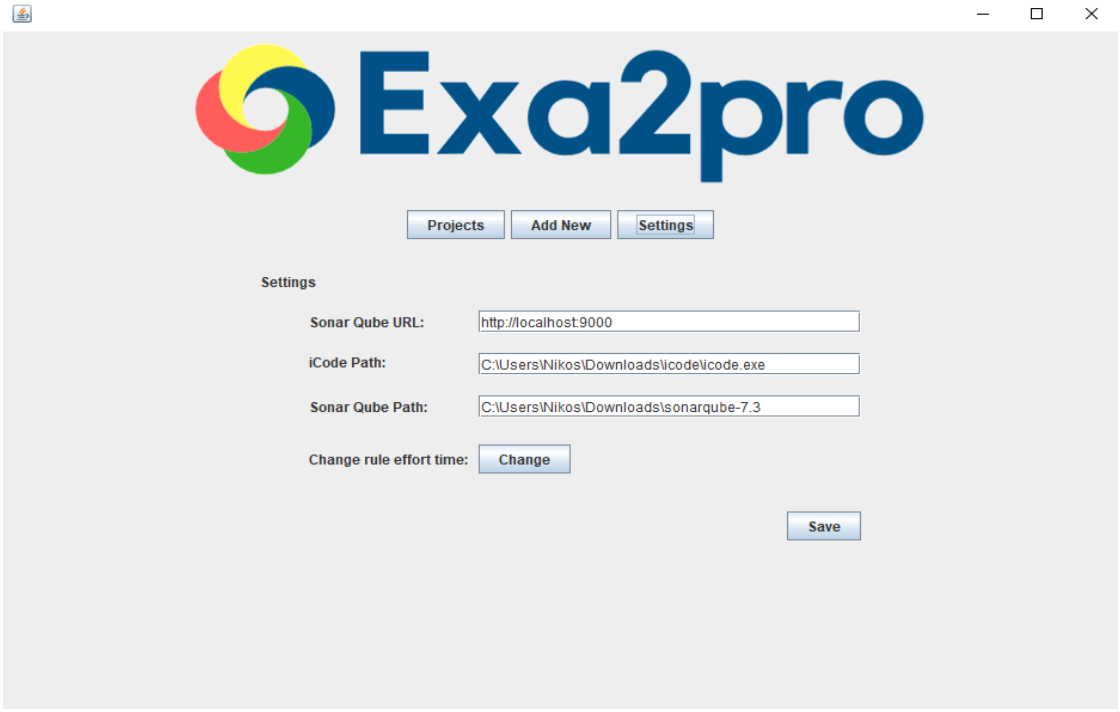
Εικόνα 6-3: Εισαγωγή καινούργιου έργου

6.4.2 Ρυθμίσεις

Όπως αναφέρθηκε προηγουμένως, την πρώτη φορά που θα εκτελεστεί το εργαλείο, ο χρήστης θα πρέπει να το διαμορφώσει ώστε να μπορεί να συνδεθεί με το SonarQube του χρήστη. Μεταβαίνοντας στο panel Settings, ο χρήστης βλέπει τις απαιτούμενες πληροφορίες που πρέπει να παρέχει, όπως φαίνεται στην Εικόνα 6-4. Οι πληροφορίες αυτές είναι:

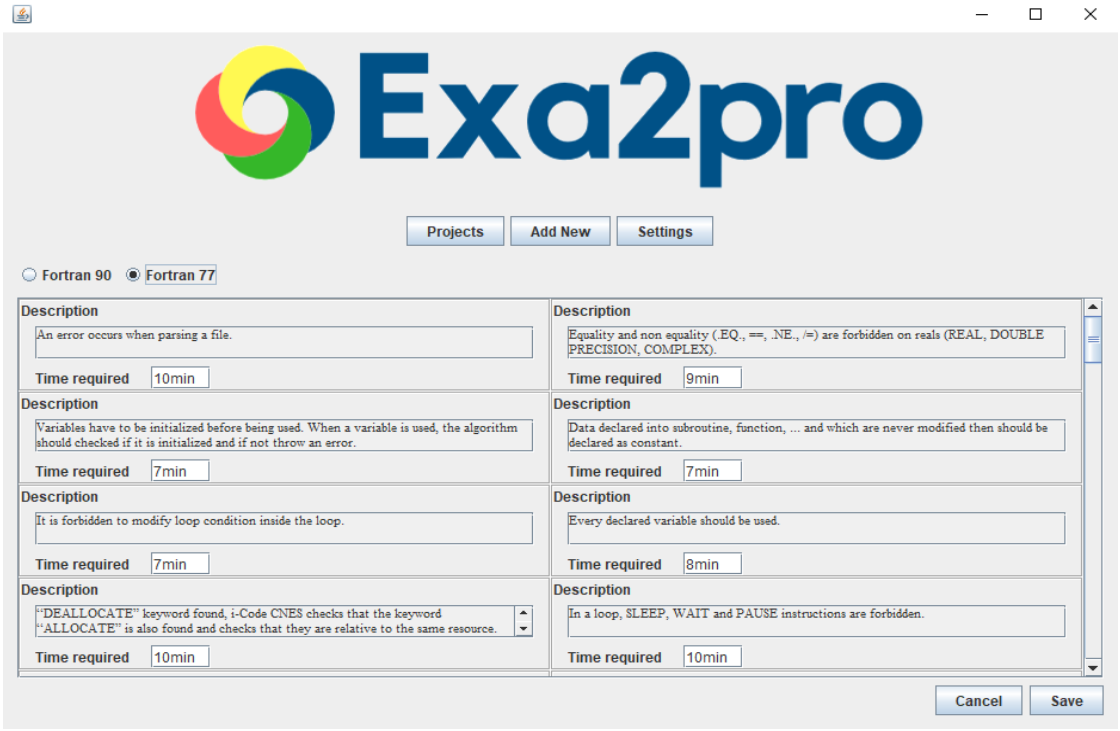
- Το URL του SonarQube για τη σύνδεση με το API του. Για παράδειγμα, "http: // localhost: 9000", αλλά αυτό μπορεί να αλλάξει ώστε να αντιστοιχεί στη διεύθυνση URL της εγκατάστασης του SonarQube που χρησιμοποιείται από τον χρήστη.
- Την τοποθεσία του iCode στον υπολογιστή του χρήστη. Κατά την εγκατάσταση των απαιτούμενων plugins για το SonarQube ο χρήστης πρέπει να κατεβάσει και την εφαρμογή γραμμής εντολών iCode. Η τοποθεσία της εφαρμογής γραμμής εντολών iCode χρησιμοποιείται μέσα στο αρχείο "sonar-project.properties" όπου είναι απαραίτητη ώστε κάποιος να εκτελέσει μια ανάλυση για έργα που χρησιμοποιούν τη γλώσσα προγραμματισμού Fortran. Στο εργαλείο μας η δημιουργία του αρχείου "sonar-project.properties" είναι αυτόματη, αλλά η διαδρομή της εφαρμογής iCode εξακολουθεί να απαιτείται.

- Η διαδρομή του SonarQube στο μηχάνημα του χρήστη. Σε περίπτωση που ο χρήστης χρησιμοποιεί το SonarQube τοπικά και θέλει να μπορεί να αλλάξει τον χρόνο αποκατάστασης των ζητημάτων για τους κανόνες Fortran, θα πρέπει να συμπληρώσει την διαδρομή της εγκατάστασης SonarQube.



Εικόνα 6-4: Ρυθμίσεις εργαλείου

Στο panel Settings επιλέγοντας το κουμπί με την ένδειξη “Change” ο χρήστης μπορεί να αποκτήσει πρόσβαση στη δυνατότητα αλλαγής του απαιτούμενου χρόνου αποκατάστασης των κανόνων όπως φαίνεται στην Εικόνα 6-5. Αποθηκεύοντας τυχόν αλλαγές, το plug-in ξαναχτίζεται και μετά την επανεκκίνηση του SonarQube, ο χρήστης μπορεί να δει τις αλλαγές.



Εικόνα 6-5: Αλλαγή απαιτούμενων χρόνων αποκατάστασης

6.4.3 Πληροφορίες Έργου

Όταν ο χρήστης επιλέγει ένα έργο από την κύρια οθόνη, ανοίγει ένα επιπλέον παράθυρο με όλες τις πληροφορίες σχετικά με το επιλεγμένο έργο.

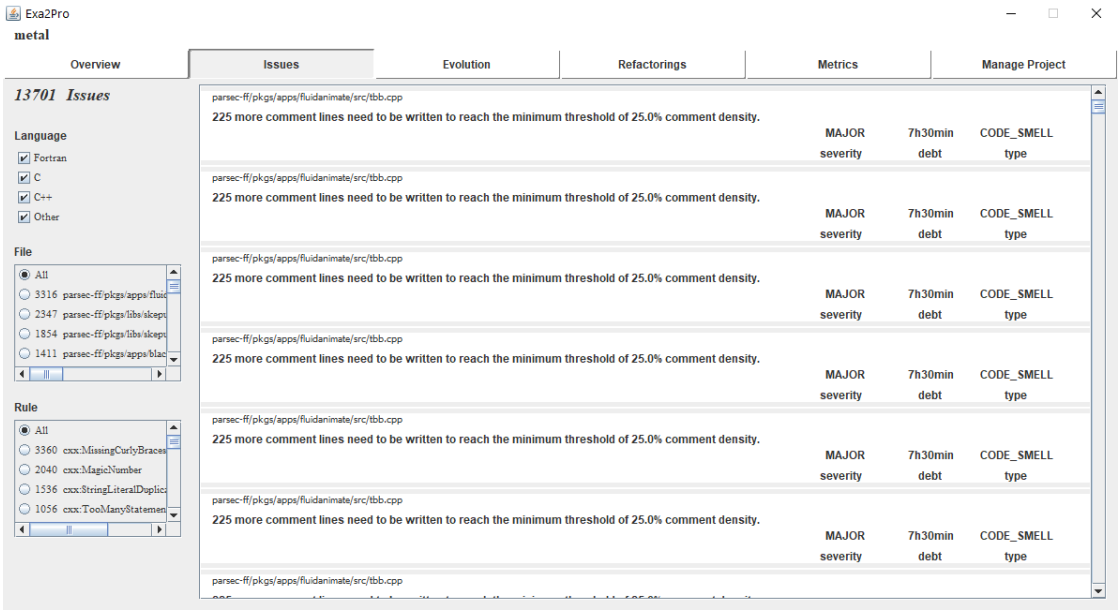
Στον πρώτο panel, ο χρήστης μπορεί να δει μερικές γενικές πληροφορίες για το έργο όπως γραμμές κώδικα, τις γλώσσες προγραμματισμού που χρησιμοποιούνται καθώς και την ημερομηνία της τελευταίας ανάλυσης του έργου, όπως φαίνεται στην Εικόνα 6-6. Επίσης, είναι διαθέσιμη μια επισκόπηση των υπολογισμένων μετρικών του συστήματος μαζί με το τεχνικό χρέος, τον Τόκο και το Κεφάλαιο ΤΧ του κώδικα (Source Code Debt) και του σχεδιασμού (Design Debt). Τα γραφήματα πίτας δείχνουν το ποσοστό των υγιών και προβληματικών αρχείων / διαδικασιών του έργου, για κάθε μετρική που υπολογίσαμε. Καθώς οι μετρικές είναι προσαρμοσμένες για χρήση στις συγκεκριμένες γλώσσες προγραμματισμού, δεν υπάρχουν ανώτατα επιτρεπτά όρια στη βιβλιογραφία. Έτσι, ορίζουμε το όριο πάνω από το οποίο μια τιμή θεωρείται «προβληματική», όταν ανήκει στο ανώτατο 10% της κατανομής της μετρικής, λαμβάνοντας υπόψη όλα τα έργα που αναλύθηκαν. Με αυτόν τον τρόπο, υιοθετούμε μια προσέγγιση στην οποία όλα τα έργα αξιολογούνται ουσιαστικά με βάσης τα ήδη αναλυμένα.

Εικόνα 6-6: Overview panel

Ο χρήστης μπορεί να δει όλα τα ζητήματα τεχνικού χρέους (εκείνα που συμβάλλουν στο Κεφάλαιο ΤΧ) του έργου στον επόμενο panel, όπως φαίνεται στην Εικόνα 6-7. Τα ζητήματα αυτά προέρχονται από την ανάλυση στο SonarQube με τη βοήθεια του API που προσφέρετε. Κάθε ζήτημα συνοδεύεται από τις ακόλουθες πληροφορίες:

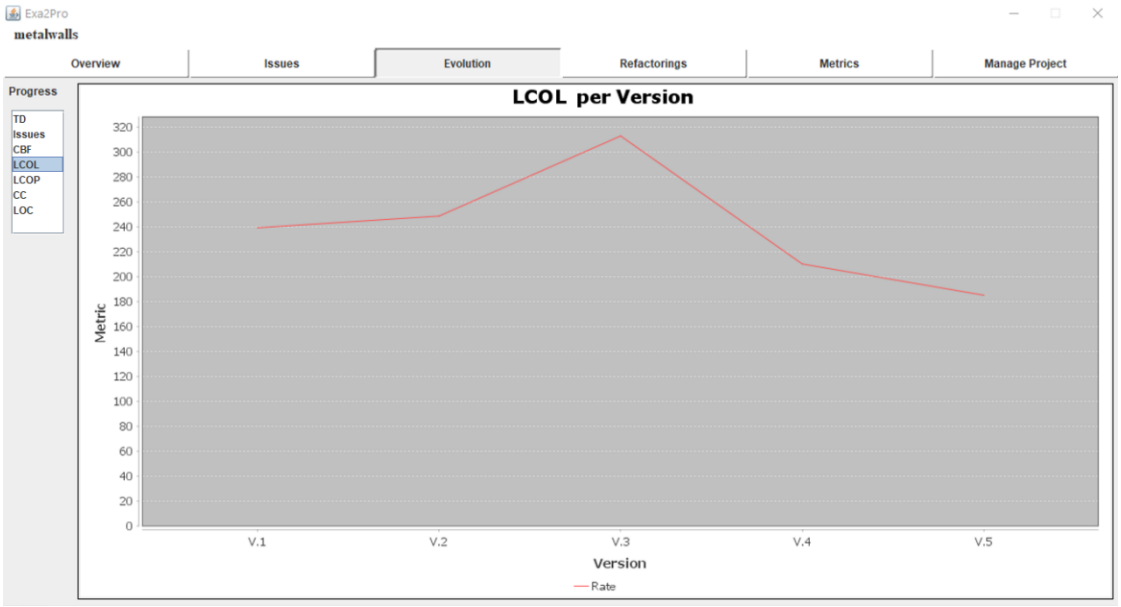
- Το αρχείο όπου υπάρχει το ζήτημα, για παράδειγμα “temp_fortan_example.f90”.
- Η περιγραφή του ζητήματος, π.χ. “*It is not allowed to use the format * for reals like r*”.
- Η σοβαρότητα του ζητήματος, για παράδειγμα “MAJOR”.
- Ο χρόνος αποκατάστασης που απαιτείται για την επίλυση.
- Ο τύπος του ζητήματος (π.χ. CODE SMELL).

Σε αυτόν το panel, ο χρήστης μπορεί να φιλτράρει τα εμφανιζόμενα ζητήματα με τρεις διαφορετικές επιλογές (μπορούν να πραγματοποιηθούν και συνδυασμοί): (α) ανά γλώσσα, (β) ανά αρχείο, και (γ) κατά κανόνα.



Εικόνα 6-7: Issues panel

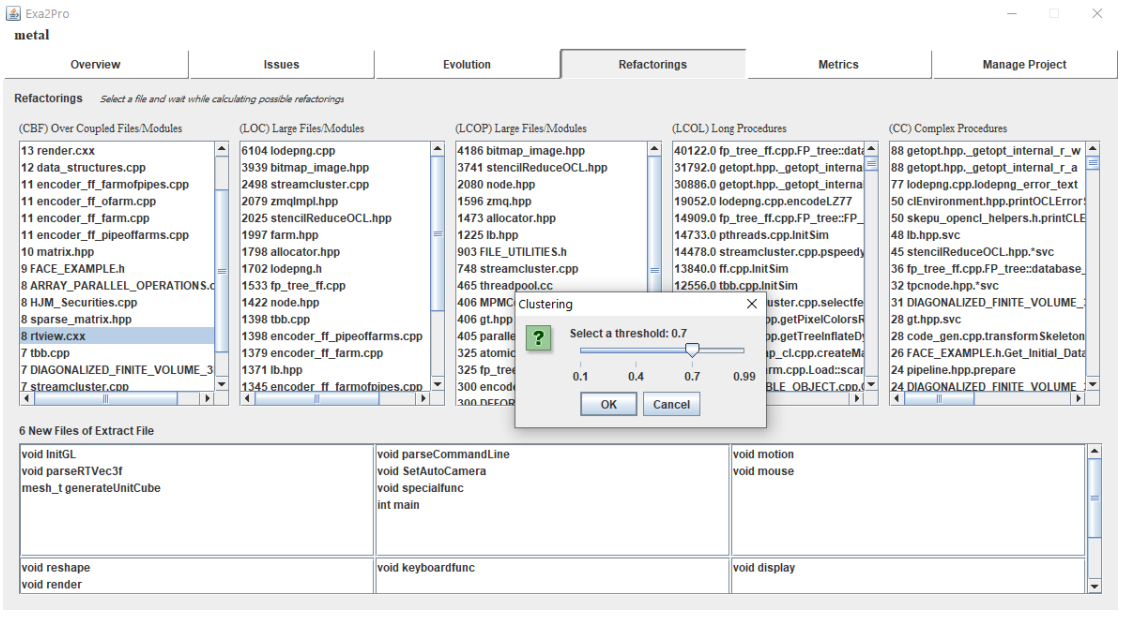
Στη συνέχεια, ο χρήστης μπορεί να δει την εξέλιξη των μετρικών του έργου και του τεχνικού χρέους σε όλες τις εκδόσεις του έργου (Εικόνα 6-8). Συγκεκριμένα, ο χρήστης έχει πρόσβαση στα γραφήματα του TX, αριθμό ζητημάτων, μέσος όρος CBF, LOC και LCOP αρχείων και μέσος όρος CC και LCOL διαδικασιών.



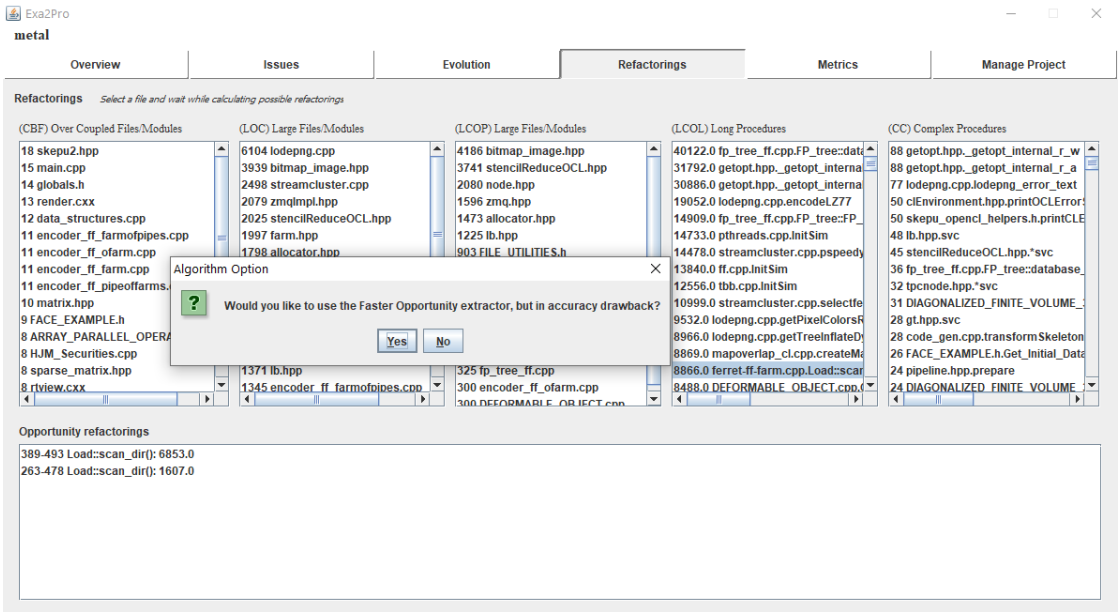
Εικόνα 6-8: Evolution panel

Για την διαχείριση του Τόκου TX, ο χρήστης πρέπει να κάνει διάφορες αναδιαμορφώσεις στον κώδικα του, βασισμένος στις τιμές των μετρικών (CBF, LOC, LCOP, CC, and LCOL) όπως φαίνεται στις παρακάτω εικόνες. Με τον ίδιο τρόπο όπως στα γραφήματα πίτας που αναφέρθηκαν προηγουμένως, βρίσκουμε τα αρχεία και τις

διαδικασίες που είναι «προβληματικά» αν οι μετρικές τους ανήκουν στο άνω 10% και τα δείχνουμε σε φθίνουσα σειρά σύμφωνα με την τιμή του. Επομένως, τα αρχεία και οι διαδικασίες για τις οποίες είναι πιο επείγον να επιλυθούν τα ζητήματά τους αναφέρονται πρώτα. Για τις μετρικές CBF, LOC, και LCOP οι οποίες είναι επιπέδου αρχείου η αντιμετώπιση από την μεριά το χρήστη είναι η εφαρμογή αναδιαμορφώσεων Extract File / Module. Όταν ο χρήστης επιλέγει ένα συγκεκριμένο αρχείο από αυτές τις λίστες, εμφανίζεται ένα μήνυμα για την επιλογή του ορίου για τον αλγόριθμο που θα εφαρμοστεί (ενότητα 3.3.2) και ο χρήστης στην συνέχεια βλέπει τα αποτελέσματα όπως φαίνονται στο κάτω μέρος της Εικόνας 6-9. Τα αποτελέσματα αυτά είναι χωρισμένα σε καινούργια Files / Modules που πρέπει να δημιουργηθούν και τις διαδικασίες που πρέπει να έχει το κάθε ένα από αυτά. Για τις μετρικές LCOL και CC που υπολογίζονται σε επίπεδο διαδικασίας η αντιμετώπιση είναι εφικτή μέσω αναδιαμορφώσεις τύπου Extract Procedure. Όταν ο χρήστης επιλέγει μία διαδικασία από τις αντίστοιχες στήλες εμφανίζεται ένα μήνυμα για την επιλογή του αλγόριθμου εφαρμογής (ενότητα 3.3.1) και τα αποτελέσματα είναι ορατά όπως στην Εικόνα 6-10. Τα αποτελέσματα είναι πιθανές εφαρμογές Extract Procedure δίνοντας στον χρήστη τις γραμμές εξαγωγής, και την μείωση που θα πετύχει η εφαρμογή αυτή.

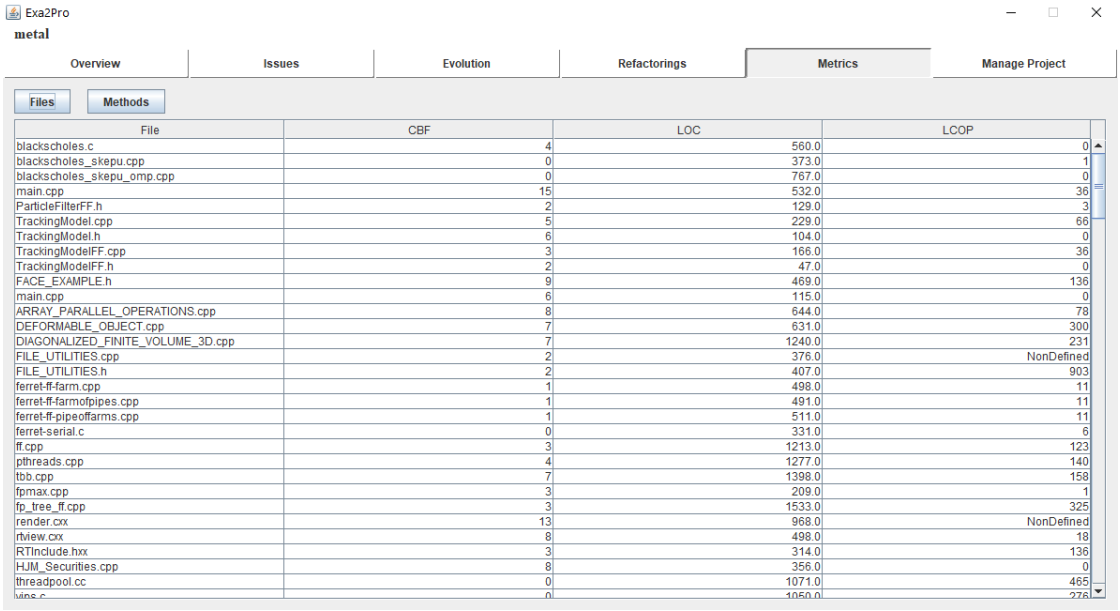


Εικόνα 6-9: Refactoring panel, Extract File / Module



Εικόνα 6-10: Refactoring panel, Extract Procedure

Στο panel Refactorings, ο χρήστης μπορεί να δει τις μετρικές μόνο για τα αρχεία και τις διαδικασίες που χρειάζονται κάποιο είδος αναδιαμόρφωσης. Εάν ο χρήστης θέλει να δει την υπολογισμένη μετρική για όλα τα αρχεία και διαδικασίες, θα πρέπει να μεταβεί στον panel Metrics, όπως παρουσιάζεται στην Εικόνα 6-11. Μέσω των κουμπιών “Files” και “Methods” ο χρήστης έχει πρόσβαση σε όλες τις μετρικές. Οι μετρικές CBF, LOC, και LCOP υπολογίζονται για κάθε αρχείο, ενώ οι CC και LCOL για κάθε διαδικασία.

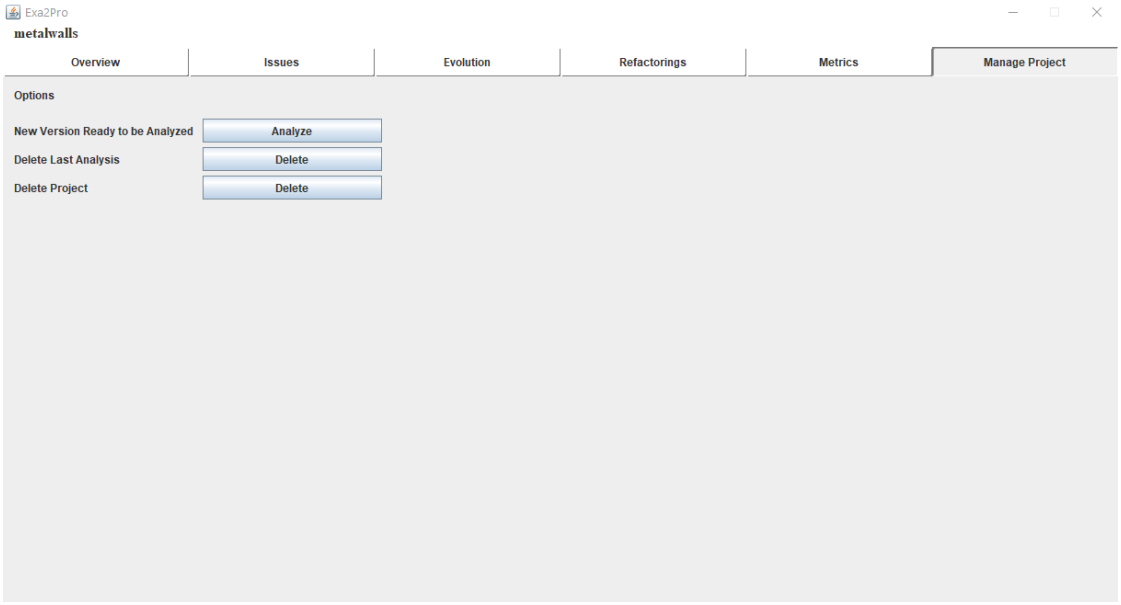


File	CBF	LOC	LCOP
blackscholes.c	4	560.0	0
blackscholes_skepu.cpp	0	373.0	1
blackscholes_skepu_omp.cpp	0	767.0	0
main.cpp	15	532.0	36
ParticleFilterFF.h	2	129.0	3
TrackingModel.cpp	5	229.0	66
TrackingModel.h	6	104.0	0
TrackingModelFF.cpp	3	166.0	36
TrackingModelFF.h	2	47.0	0
FACE_EXAMPLE.h	9	469.0	136
main.cpp	6	115.0	0
ARRAY_PARALLEL_OPERATIONS.cpp	8	644.0	78
DEFORMABLE_OBJECT.cpp	7	631.0	300
DIAGONALIZED_FINITE_VOLUME_3D.cpp	7	1240.0	231
FILE_UTILITIES.cpp	2	376.0	NonDefined
FILE_UTILITIES.h	2	407.0	903
ferret-ff-farm.cpp	1	498.0	11
ferret-ff-farmofpipes.cpp	1	491.0	11
ferret-ff-pipeoffarms.cpp	1	511.0	11
ferret-serial.c	0	331.0	6
ff.cpp	3	1213.0	123
pthreads.cpp	4	1277.0	140
tbb.cpp	7	1398.0	158
fpmax.cpp	3	209.0	1
fp_tree_ff.cpp	3	1533.0	325
render.cxx	13	968.0	NonDefined
rtview.cxx	8	498.0	18
RTInclude.hxx	3	314.0	136
HJM_Securities.cpp	8	356.0	0
threadpool.cc	0	1071.0	465

Εικόνα 6-11: Metrics panel

Τέλος, οι ενέργειες που μπορεί να κάνει ένας χρήστης για την διαχείριση ενός έργου εμφανίζονται στον panel Manage Project, Εικόνα 6-12. Όταν ο χρήστης έχει

ολοκληρώσει τις αλλαγές στα αρχεία του έργου και θέλει να τρέξει εκ νέου την ανάλυση στη νέα έκδοση του έργου, μπορεί να επιλέξει το κουμπί “Analyze”. Εκτός από το κουμπί της ανάλυσης νέας έκδοσης, υπάρχουν δύο επιπλέων κουμπιά “Delete” που δίνουν τη δυνατότητα στον χρήστη να διαγράψει την τελευταία ανάλυση ή ολόκληρο το έργο.



Εικόνα 6-12: Manage panel

7 Εμπειρικά Αποτελέσματα Αποπληρωμής Τεχνικού Χρέους

Σε αυτήν την ενότητα παρουσιάζουμε τα αποτελέσματα της χρήσης του εργαλείου μας για τη διαχείριση τεχνικού χρέους σε εφαρμογές που ανήκουν στο ευρωπαϊκό έργο EXA2PRO. Συγκεκριμένα, αναφέρουμε τα εξής:

- Αριθμός προβλημάτων σχεδιασμού που βρέθηκαν
- Σύνολο τεχνικού χρέους Design
- Ευκαιρίες αναδιαμορφώσεων που βρέθηκαν / εφαρμόστηκαν
- Αξιολόγηση της εφαρμογής των αναδιαμορφώσεων: εννοιολογική αξιολόγηση (καταλληλόλητα της αναδιαμόρφωσης) και αξιολόγηση του τεχνικού χρέους (μείωση του τεχνικού χρέους Design)

Τα αποτελέσματα είναι οργανωμένα σε τρεις υποενότητες, με βάση τους τρεις πάρωρους των εφαρμογών.

7.1 Εφαρμογή Στο CO_2Capture

Το εργαλείο μας εφαρμόστηκε δύο φορές στο πρότζεκ CO_2Capture για δύο διαφορετικές εκδόσεις. Η πρώτη φορά ήταν στο commit αμέσως μετά την υιοθέτηση του SkePu, προκειμένου να προταθούν κυρίως ευκαιρίες Extract Procedure (Εξαγωγής Διαδικασίας), ενώ η δεύτερη έπειτα από κάποιες εσωτερικές αλλαγές και επικεντρώθηκε στις αναδιαμορφώσεις Extract Files / Modules (Εξαγωγής Αρχείων / Modules).

7.1.1 Πρώτη Εκτέλεση

Κατά την διάρκεια εκτέλεσης του εργαλείου μας στην εφαρμογή CO_2Capture, στην πρώτη έκδοση, καταφέραμε να εντοπίσουμε 4.937 ζητήματα SonarQube και 51 ζητήματα σε επίπεδο σχεδίασης (Πίνακες 7-1 με 7-5). Το συνολικό Κεφάλαιο ΤΧ αντιστοιχεί σε 41.059 ευρώ για χρέος πηγαίου κώδικα και 329 ευρώ για χρέος σχεδιασμού. Ο Τόκος ΤΧ ισούται με: 664,94 ευρώ ανά νέα έκδοση. Μια επισκόπηση των Design Issues απεικονίζεται στους Πίνακες 7-1 με 7-5, ενώ πρέπει να σημειώσουμε ότι για τους πίνακες των μετρήσεων CC και LOC, υπολογίστηκε ένας μέσος όρος των μετρικών των διαδικασιών κάθε αρχείου, καθώς είναι μετρικές επιπέδου διαδικασίας.

Πίνακας 7-1: Πρώτη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις CC

Αρχεία	CC
x2p_frpmodel.cpp	76.6

frpmodel.f	52.5
frpinit.f	37
x2p_copy.cpp	34
meamainpitcon.f	30.5
frprops2.f	24
frpmainmn.f	22
funobj_real.f	18
itest.f	13.5
sa_moves.f	12.5
run_once.f	12

Πίνακας 7-2: Πρώτη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις LCOL

Αρχεία	LCOL
funobj_real.f	195173
frpmodel.f	76307
frpinit.f	15945
frpmainmn.f	6980
meamainpitcon.f	9568
itest.f	3949
frprops2.f	2285

Πίνακας 7-3: Πρώτη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις LOC

Αρχεία	LOC
x2p_frpmodel.cpp	3736
frpmodel.f	3246
itest.f	882
funobj_real.f	765

Πίνακας 7-4: Πρώτη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις CBF

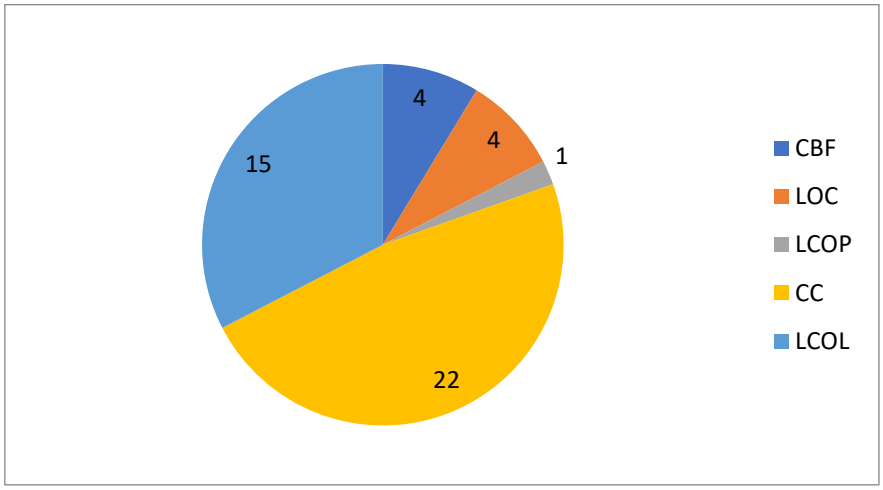
Αρχεία	CBF
itest.f	15
main_anneal_par.f	14
master_mpi.f	8

meamainpitcon.f	8

Πίνακας 7-5: Πρώτη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις LCOP

Αρχεία	LCOP
x2p_variables.cpp	144

Στη συνέχεια, παρουσιάζεται η κατανομή σχετικά με τον τύπο προβλημάτων σχεδιασμού, μέσω ενός γραφήματος πίτας (Εικόνα 7-1).



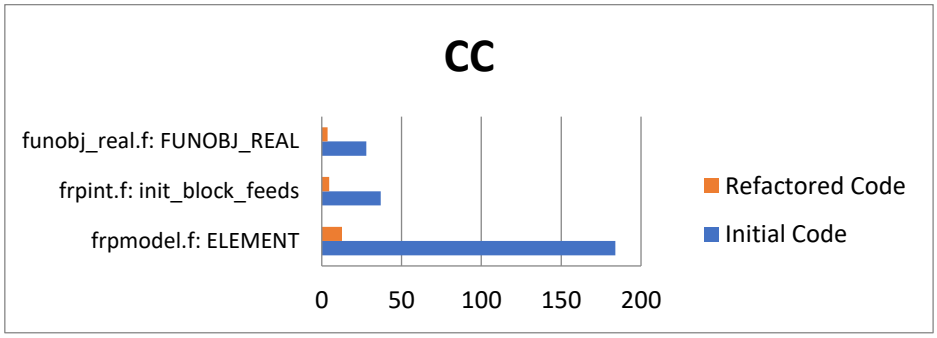
Εικόνα 7-1: Αριθμός σχεδιαστικών προβλημάτων ανά τύπο προβλήματος

Από τις παραπάνω ευκαιρίες αναδιαμόρφωσης, εφαρμόσαμε 25 Extract Procedures και 1 Extract Files / Module. Στη συνέχεια, περιγράφουμε: (α) την εννοιολογική αξιολόγηση των refactorings - δηλαδή, το βαθμό στον οποίο είχαν νόημα για τους προγραμματιστές του CO_2Capture, και (β) την ποσοτική ανάλυση των μετρήσεων των αποτελεσμάτων της εφαρμογής και ειδικότερα του Τόκου TX και των μετρήσεων.

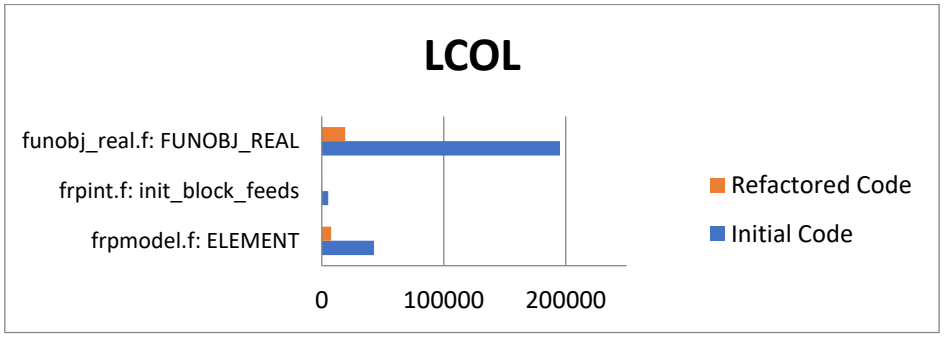
Μετά την εφαρμογή των προτεινόμενων refactorings, ακολούθησε μια σύντομη συνέντευξη με τους προγραμματιστές των έργων, μαζί με ένα ερωτηματολόγιο για την αξιολόγηση των αλλαγών. Πρώτα απ' όλα, πρέπει να σημειώσουμε ότι όλα τα refactorings μας τεκμηριώθηκαν ως σωστά, εννοιολογικά και δομικά, σύμφωνα με τους προγραμματιστές του έργου. Επιπλέον, αναμένουμε ότι το αντίκτυπο των refactorings στη συντήρηση είναι υψηλότερο όταν το κώματι κώδικα όπου εφαρμόζετε το refactoring είναι επιρρεπής σε αλλαγές. Σύμφωνα με αυτήν την υπόθεση, εκτιμούμε ότι θα υπάρξει σημαντικό αντίκτυπο στη συντηρησιμότητα του συνολικού προγράμματος, διότι σε 25 περιπτώσεις ο αρχικός κώδικας του αλλάζει πολύ συχνά. Επιπλέων, μια από τις υψηλότερες σε προτεραιότητα ιδιότητα, σε μια εφαρμογή HPC, είναι η απόδοση. Μετά τις εφαρμογές αναδιαμόρφωσης μετρήσαμε τον αντίκτυπο στον χρόνο εκτέλεσης με την

βοήθεια των προγραμματιστών του έργου. Οι αλλαγές μας στη βάση κώδικα δεν έβλαψαν την απόδοση, αντίθετα, η εκτέλεση του λογισμικού ήταν 0,4% ταχύτερη.

Η εφαρμογή των refactorings οδήγησε σε μείωση του αριθμού των ζητημάτων επιπέδου σχεδίασης, σε μείωση των μετρικών σε συγκεκριμένα αρχεία και σε αυτές του συνολικού έργου. Στις επόμενες εικόνες, παρουσιάζουμε τις αλλαγές στις μετρικές σχεδίασης, όπου η μετρική του αλλαγμένου κώδικα υπολογίστηκε ως ο μέσος όρος των μετρικών όλων των διαδικασιών ή αρχείων που άλλαξαν ή προστέθηκαν κατά την εφαρμογή του refactoring. Στις Εικόνες 7-2 και 7-3 μπορούμε να δούμε μετρικές επιπέδου μεθόδου που μειώθηκαν αρκετά καθώς η κύρια αναδιαμόρφωση της πρώτης εκτέλεσης ήταν η Extract Procedure.

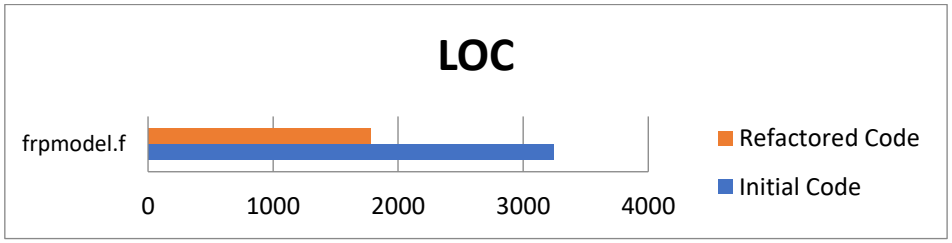


Εικόνα 7-2: Πρώτη εκτέλεση CO_2Capture, μετρική CC



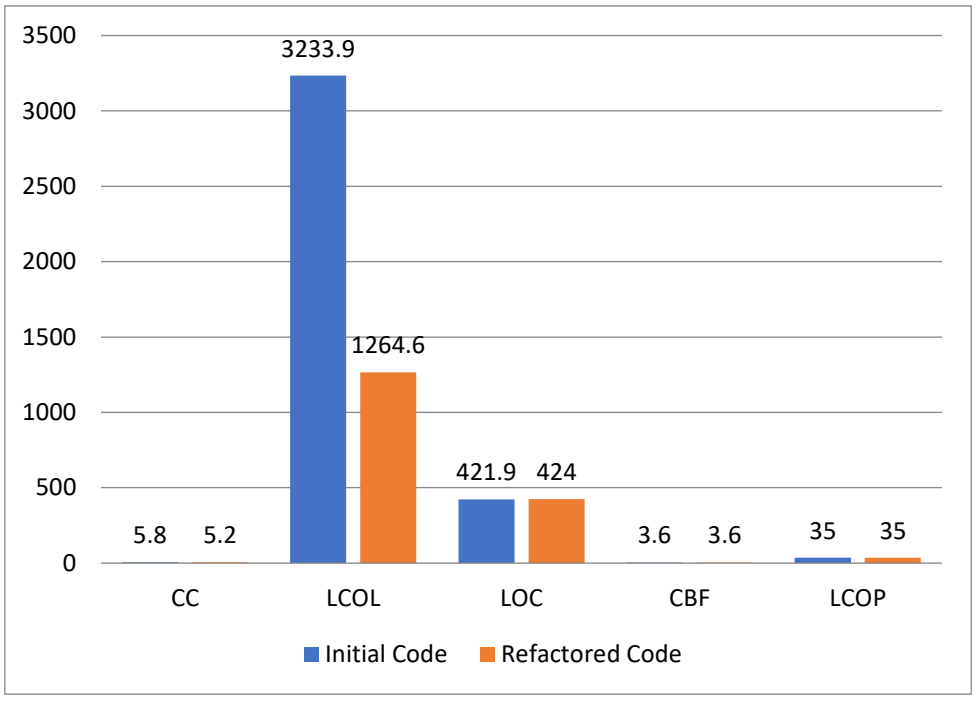
Εικόνα 7-3: Πρώτη εκτέλεση CO_2Capture, μετρική LCOL

Εκτός από τα τις αναδιαμορφώσεις Extract Procedure, εφαρμόσαμε επίσης μια Extract File / Module με αποτέλεσμα τη μείωση της μέτρησης LOC αυτού του αρχείου. Η αλλαγή αυτή είναι ορατή στην Εικόνα 7-4, αλλά οι υπόλοιπες μετρικές επιπέδου αρχείου / module δεν επηρεάστηκαν και επομένως δεν απεικονίζονται σε γραφήματα. Οι μετρικές CBF και LCOP μπορούν να αλλάξουν μόνο με την εκτέλεση refactoring Extract File / Module, η οποία δεν ήταν δυνατή σε αυτήν την εκτέλεση με μόνο μία, αρκετά μικρή σε έκταση, αναδιαμόρφωση αυτού του τύπου.



Εικόνα 7-4: Πρώτη εκτέλεση CO_2Capture, μετρική LOC

Οι μετρικές ολόκληρου του έργου απεικονίζονται στην Εικόνα 7-5 όπου μπορούμε να δούμε ότι οι αλλαγές είναι ορατές ειδικά στη μετρική LCOL, αλλά υπάρχει ακόμη περιθώριο βελτίωσης, καθώς οι μετρικές πρέπει να αλλάξουν σε περισσότερα αρχεία / διαδικασίες για να έχουν αντίκτυπο στο συνολικό έργο. Πρέπει να σημειώσουμε ότι η μετρική LOC αυξήθηκε ελαφρώς ως αντιστάθμιση από τα Extract Procedures και μόνο μια αναδιαμόρφωση τύπου Extract File / Module δεν ήταν αρκετή για τη μείωση του. Με βάση τα παραπάνω, ο Τόκος ΤΧ μειώθηκε κατά 126,6 ευρώ και τώρα ισούται με 538,34 ευρώ ανά νέα έκδοση.



Εικόνα 7-5: Πρώτη εκτέλεση CO_2Capture, μετρικές συστήματος

7.1.2 Δεύτερη Εκτέλεση

Αφού η ομάδα προγραμματιστών του ΕΚΕΤΑ υιοθέτησε τα refactorings που προτάθηκαν κατά την πρώτη εκτέλεση (μαζί με την προσθήκη αρκετών άλλων αλλαγών), πραγματοποιήθηκε μια δεύτερη εκτέλεση του εργαλείου μας. Πρέπει να σημειώσουμε ότι κάναμε τους υπολογισμούς μας και εφαρμόσαμε τα refactorings μόνο στον κύριο κώδικα

του προγράμματος, εξαιρώντας όλες τις βιβλιοθήκες ή τον κώδικα τρίτων. Σε αυτήν την εκτέλεση, καταφέραμε να εντοπίσουμε 6.083 ζητήματα SonarQube και 60 ζητήματα σε επίπεδο σχεδίασης. Το συνολικό Κεφάλαιο ΤΧ αντιστοιχεί σε 44.305 ευρώ για χρέος πηγαίου κώδικα και 447.1 ευρώ για χρέος σχεδιασμού. Ο Τόκος ΤΧ ισούται με: 1.694,58 ευρώ ανά νέα έκδοση. Τα αρχεία με θέματα σχεδίασης απεικονίζονται στους Πίνακες 7-6 με 7-10.

Πίνακας 7-6: Δεύτερη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις CC

Αρχεία	CC
x2p_frpmodel.cpp	76.6
element.f	37.4
frpmodel.f	31
x2p_copy.cpp	34
frpmainmn.f	22
frprops2.f	22
model_variables.F90	15
funobj_real.f	14
sa_moves.f	12.5
helpers.F90	12
model.F90	12

Πίνακας 7-7: Δεύτερη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις LCOP

Αρχεία	LCOP
controllability_assessment.F90	136
x2p_frpmodel.cpp	114

Πίνακας 7-8: Δεύτερη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις LCOL

Αρχεία	LCOL
element.f	49697.1
funobj_real.f	31575.3
frpmodel.f	25333.2
frpmainmn.f	6933
controllability_assessment.F90	3801
init.f	1725

frprops2.f	1656
slave_node.F90	1533
master_node.F90	1496

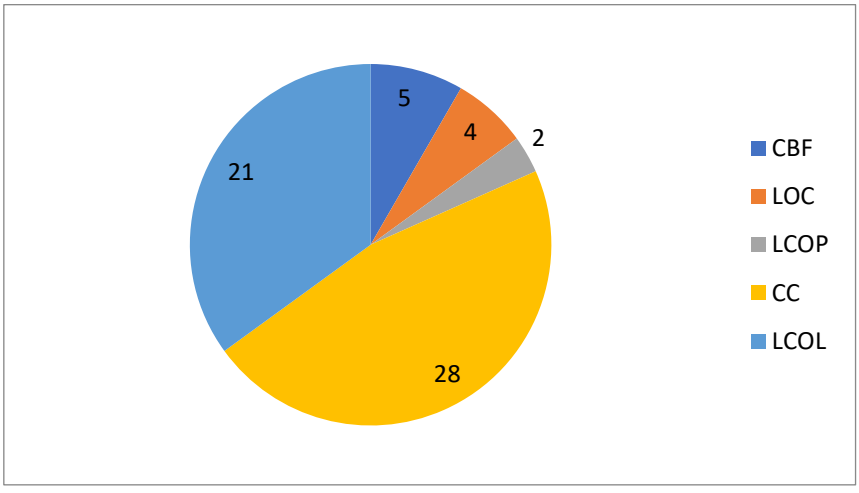
Πίνακας 7-9: Δεύτερη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις LOC

Αρχεία	LOC
x2p_frpmodel.cpp	3669
element.f	3086
frpmodel.f	1942
funobj_real.f	889

Πίνακας 7-10: Δεύτερη εκτέλεση CO_2Capture, αρχεία με μεγάλες μετρήσεις CBF

Αρχεία	CBF
main_anneal_par.f	14
main_nompi.F90	12
slave_node.F90	10
process_design.F90	9
controllability_assessment.F90	9

Στη συνέχεια, παρουσιάζεται η κατανομή σχετικά με τον τύπο προβλημάτων σχεδιασμού, μέσω ενός γραφήματος πίτας (Εικόνα 7-1).

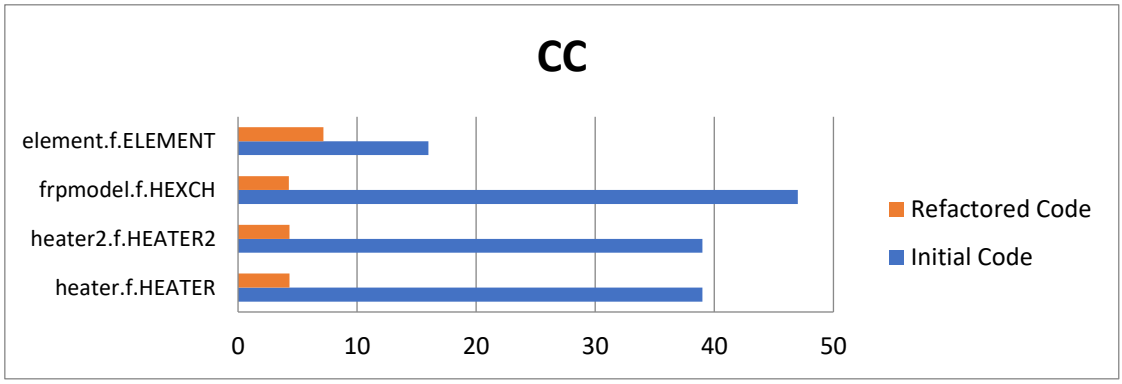


Εικόνα 7-6: Αριθμός σχεδιαστικών προβλημάτων ανά τύπο προβλήματος

Από τις προαναφερθείσες ευκαιρίες refactoring, εφαρμόσαμε 39 Extract Procedure και 6 Extract Files / Module. Όπως και την προηγούμενη φορά, μετά την εφαρμογή των προτεινόμενων refactorings, οργανώθηκε μια συνέντευξη μαζί με ένα ερωτηματολόγιο με τους προγραμματιστές του έργου, προκειμένου να βοηθήσουν στην αξιολόγηση των

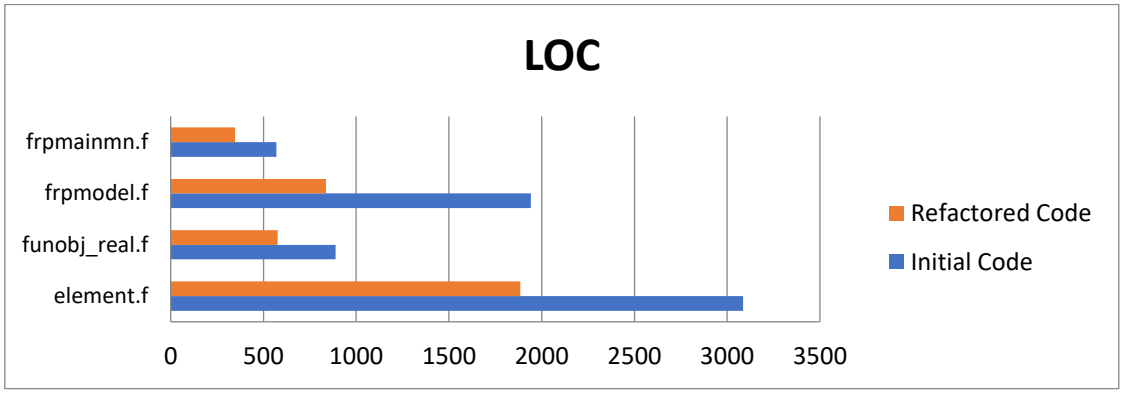
αλλαγών. Από τα 45 προτεινόμενα refactorings, μόνο 5 Extract Procedures δεν θεωρήθηκαν απολύτως σωστές, όσον αφορά την εννοιολογική σημασία: οι προγραμματιστές του έργου θα το προτιμούσαν ένα διαφορετικό τρόπο. Στο αρχείο "heater.f" 2 διεργασίες εξαγωγής συγχωνεύθηκαν ως μία και παρόμοια, στο "heater2.f" πραγματοποιήθηκε μια ακόμη συγχώνευση δύο διαδικασιών. Έτσι, ο τελικός αριθμός των refactorings είναι 36 Extract Procedures και 6 Extract Files / Module, οι οποίες αναδιαμορφώσεις θα υιοθετηθούν όλες στην κύριο κώδικα της εφαρμογής. Επιπλέον, όπως εξηγείται στην πρώτη εκτέλεση, υπάρχει υψηλότερος αντίκτυπος όταν τα refactorings πραγματοποιούνται σε ένα μέρος του κώδικα το οποίο είναι επιρρεπή σε αλλαγές. Σύμφωνα με αυτήν την υπόθεση, οι προτεινόμενες αναδιαρθρώσεις αυτής της επανάληψης αναμένεται να έχουν σημαντική επίδραση στη συντηρησιμότητα του συνολικού έργου. Ο λόγος είναι ότι η πλειονότητα των επηρεαζόμενων αρχείων είναι επιρρεπή σε αλλαγές και επιπλέον, χρησιμοποιούνται και σε άλλα έργα (μόνο 3 αρχεία δεν είναι επιρρεπή σε αλλαγές και μόνο 2 αρχεία δεν χρησιμοποιούνται σε άλλα έργα). Τέλος, όπως και στην πρώτη εκτέλεση των refactorings, μετρήθηκε και χρόνος εκτέλεσης του προγράμματος. Σε αυτήν την περίπτωση, ο χρόνος ήταν 0,5% χειρότερος μετά τα refactorings, αλλά οι προγραμματιστές το βρήκαν αποδεκτό.

Στις επόμενες εικόνες, αναφέρουμε τις μετρήσεις πριν και μετά τις αναδιαμορφώσεις για τα αρχεία / modules και τις διαδικασίες που άλλαξαν οι μετρικές τους.



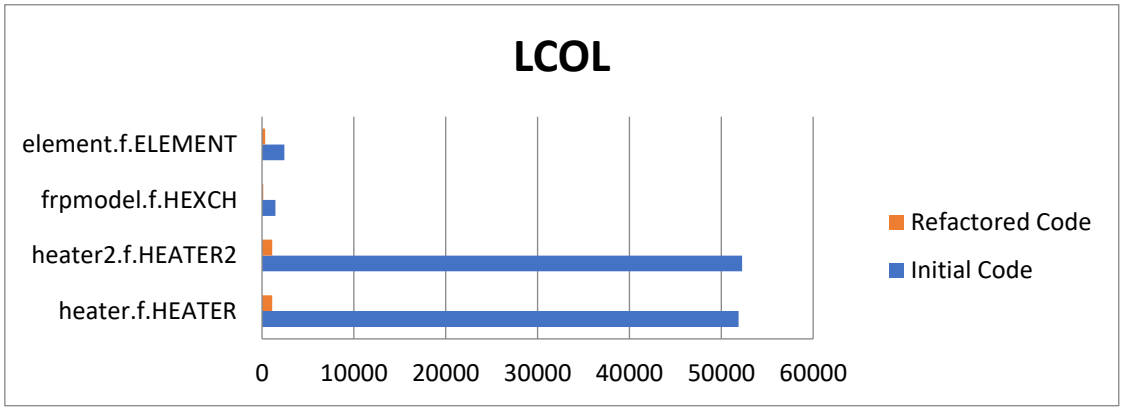
Εικόνα 7-7: Δεύτερη εκτέλεση CO_2Capture, μετρική CC

Όπως μπορεί να παρατηρηθεί, σε όλες τις διαδικασίες της Εικόνας 7-7, η τιμή της μετρικής κυκλωματικής πολυπλοκότητας μειώνεται, λόγω των Extract Procedure που εφαρμόζονται στις συγκεκριμένες διαδικασίες (βρόγχοι ελέγχου διανεμήθηκαν σε διαφορετικές νέες διαδικασίες).



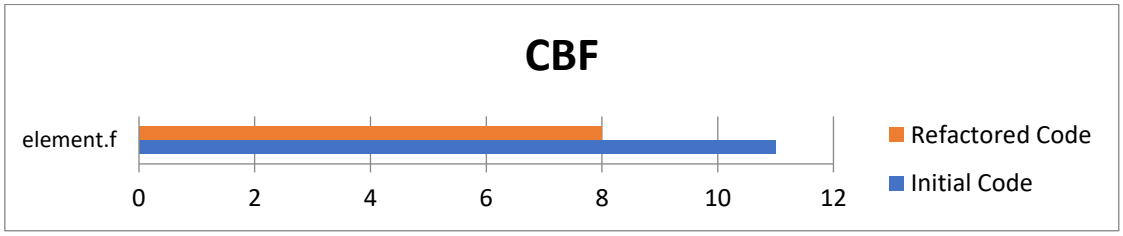
Εικόνα 7-8: Δεύτερη εκτέλεση CO_2Capture, μετρική LOC

Παρομοίως, η τιμή της μετρικής LOC μειώθηκε για όλα τα αρχεία που παρουσιάζονται στην Εικόνα 7-8, λόγω των αναδιαμορφώσεων Extract File / Module του (α) frpmainmn.f (σε funobj_cost_estimators.f), (β) frpmodel.f (στο heater.f & heater2.f) και (γ) funobj_real.f (στο funobj_real_cost_estimators.f). Στο αρχείο element.f εφαρμόσαμε μόνο refactorings Extract Procedure και καταργήσαμε την υπορουτίνα "ELEMENT_ORIGINAL", καθώς χρησιμοποιούταν μόνο για δοκιμές από τους προγραμματιστές στο παρελθόν.



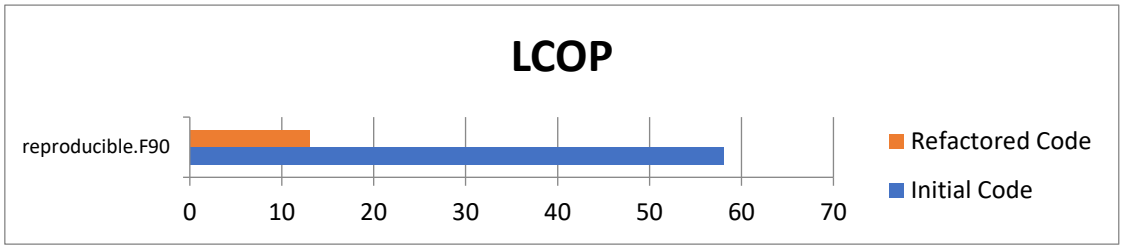
Εικόνα 7-9: Δεύτερη εκτέλεση CO_2Capture, μετρική LCOL

Η τιμή της μερικής LCOL βελτιώθηκε (μειώθηκε) για καθεμία από τις διαδικασίες που παρουσιάζονται στην Εικόνα 7-9, λόγω των αναδιαμορφώσεων κώδικα (Extract Procedure) που εφαρμόστηκαν στα συγκεκριμένα αρχεία (πολλές γραμμές κώδικα μοιράστηκαν σε διάφορες διαδικασίες).



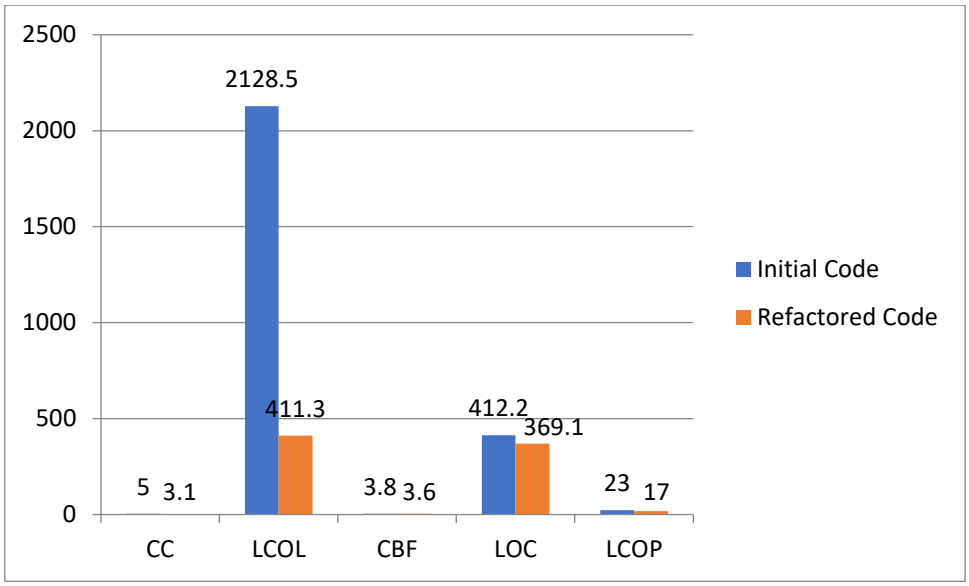
Εικόνα 7-10: Δεύτερη εκτέλεση CO_2Capture, μετρική CBF

Για το αρχείο που φαίνεται στην Εικόνα 7-10, η τιμή της μετρικής CBF μειώνεται, λόγω της αφαίρεσης της υπορουτίνας "ELEMENT_ORIGINAL", όπως αναφέραμε και παραπάνω.



Εικόνα 7-11: Δεύτερη εκτέλεση CO_2Capture, μετρική LCOP

Η τιμή της μετρικής LCOP μειώθηκε για το αρχείο reproducible.F90 που φαίνεται στην Εικόνα 7-11, λόγο της αναδιαμόρφωσης Extract File / Module (σε read_reproducible.F90 & write_reproducible.F90). Όπως απεικονίζεται στην Εικόνα 7-12, όλες οι μετρικές του συστήματος βελτιώθηκαν (μειώθηκαν), λόγω των Extract File / Module και Extract Procedure. Τέλος, πρέπει να σημειώσουμε ότι ο Τόκος TX έχει μειωθεί στα 1.322,58 ευρώ ανά νέα έκδοση (μείωση περίπου 300 ευρώ).



Εικόνα 7-12: Δεύτερη εκτέλεση CO_2Capture, μετρικές συστήματος

7.2 Εφαρμογή Στο MetalWalls

Εφαρμόζοντας το εργαλείο μας στην εφαρμογή MetalWalls, καταφέραμε να εντοπίσουμε 1.753 ζητήματα SonarQube και 71 ζητήματα επιπέδου σχεδίασης. Το συνολικό Κεφάλαιο TX ανέρχεται σε 6.176 ευρώ για χρέη πηγαίου κώδικα και 474.8 ευρώ για σχεδιαστικό χρέος. Ο Τόκος TX ισούται με: 776,22 ευρώ ανά νέα έκδοση. Τα αρχεία με θέματα σχεδίασης απεικονίζονται στους Πίνακες 7-11 με 7-15.

Πίνακας 7-11: MetalWalls, αρχεία με μεγάλες μετρήσεις CC

Αρχεία	CC
species.F90	21
configuration.F90	20.6
electrode_charge.F90	14
electrode_charge.cpp	14
cg.cpp	13
localwork.F90	12
coulomb_lr.cpp	11
configuration_line.F90	10
system.F90	9
box.F90	9
timers.F90	9
command.F90	8.6
electrode_parameters.F90	8
coulomb.cpp	8
coulomb_sr.cpp	7

Πίνακας 7-12: MetalWalls, αρχεία με μεγάλες μετρήσεις LCOL

Αρχεία	LCOL
system.F90	200218
species.F90	7554
configuration.F90	6385.3
coulomb_lr.F90	4139.5
coulomb.F90	2317.5
timers.F90	2059
localwork.F90	2047.3
output.F90	1649
coulomb_keq0.F90	1367.5
cg.F90	1553
electrode_parameters.F90	1272

Πίνακας 7-13: MetalWalls, αρχεία με μεγάλες μετρήσεις LOC

Αρχεία	LOC
configuration.F90	951
system.F90	896
coulomb_lr.cpp	820
localwork.F90	755
output.F90	709
command.F90	475
ewald.F90	435

Πίνακας 7-14: MetalWalls, αρχεία με μεγάλες μετρήσεις CBF

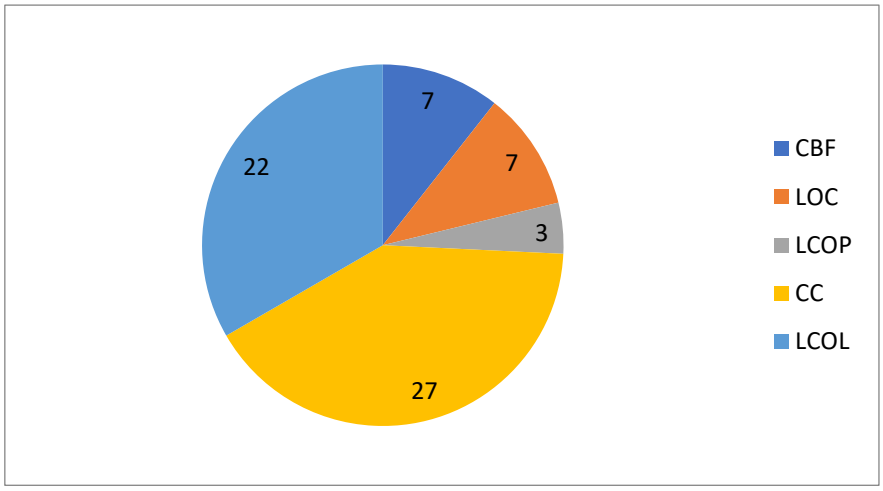
Αρχεία	CBF
19 system.F90	19
19 main.F90	19
19 configuration.F90	19
16 coulomb.F90	16
13 coulomb.h	13
13 electrode_charge.h	13
11 coulomb_sr.F90	11

Πίνακας 7-15: MetalWalls, αρχεία με μεγάλες μετρήσεις LCOP

Αρχεία	LCOP
326 output.F90	326
36 kinds.F90	36
1 stdio.F90	1

Η κατανομή σχετικά με τον τύπο των προβλημάτων σχεδιασμού παρουσιάζεται μέσω ενός γραφήματος πίτας στην Εικόνα 7-13. Από τις προαναφερθείσες ευκαιρίες αναδιαμόρφωσης, εφαρμόστηκαν 79 Extract Procedure και 5 Extract Files / Modules. Από τα συνολικά 84 προτεινόμενα refactorings, μόνο 7 Extract Procedure δεν ορίστηκαν ως απολύτως σωστές, με την έννοια ότι εννοιολογικά οι προγραμματιστές του έργου θα τα προτιμούσαν με διαφορετικό τρόπο. Εκτός αυτού, 11 Extract Procedure δεν ενσωματώθηκαν στον τελικό κώδικα, επειδή τα refactorings αυτά είχαν επίδραση στην λειτουργικότητα και στα τελικά αποτελέσματα του προγράμματος. Από τα συνολικά 5

refactorings Extract Files / Modules, υιοθετήθηκαν μόνο τα 2, λόγω του ότι οι προγραμματιστές θα τα προτιμούσαν με διαφορετικό τρόπο. Εννοιολογικά, ωστόσο, συμφωνήθηκε ότι όλα τα Refactorings Extract Files / Modules είχαν νόημα.



Εικόνα 7-13: Αριθμός σχεδιαστικών προβλημάτων ανά τύπο προβλήματος

Οι συνολικοί χρόνοι εκτέλεσης του προγράμματος πριν και μετά τις αναδιαμορφώσεις παρουσιάζεται τον Πίνακα 7-16, ο οποίος παρουσιάζει τους χρόνους εκτέλεσης (σε δευτερόλεπτα) για έναν / είκοσι πυρήνες για τρεις περιπτώσεις (Supercap, Blue Energy, Graphene). Με βάση τις τιμές στον πίνακα, μπορούμε να ισχυριστούμε ότι οι αναδιαμορφώσεις κώδικα δεν επηρεάζουν σημαντικά τη συνολική απόδοση του κώδικα.

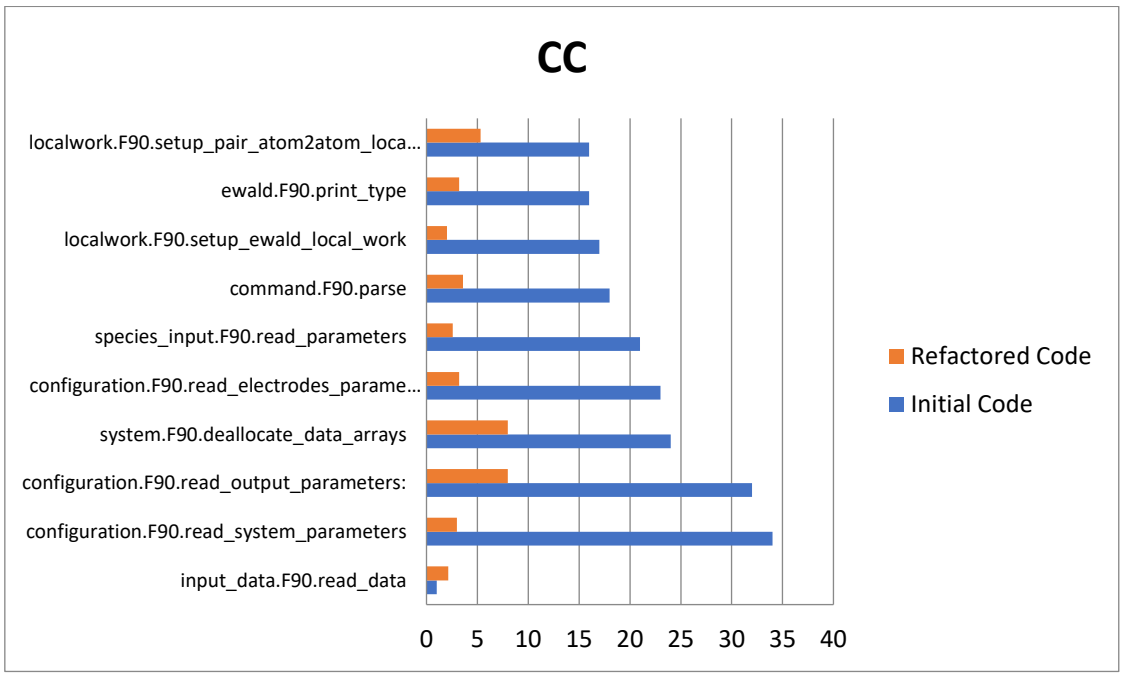
Πίνακας 7-16: Χρόνοι εκτέλεσης MetalWalls

Version	Supercap		Blue Energy		Graphene	
	1 Core	20 Cores	1 Core	20 Cores	1 Core	20 Cores
Before Refactoring	127.1	6.86	420.9	23.5	15,416	813
After Refactoring	126.6	6.91	420.3	22.6	15,417	828

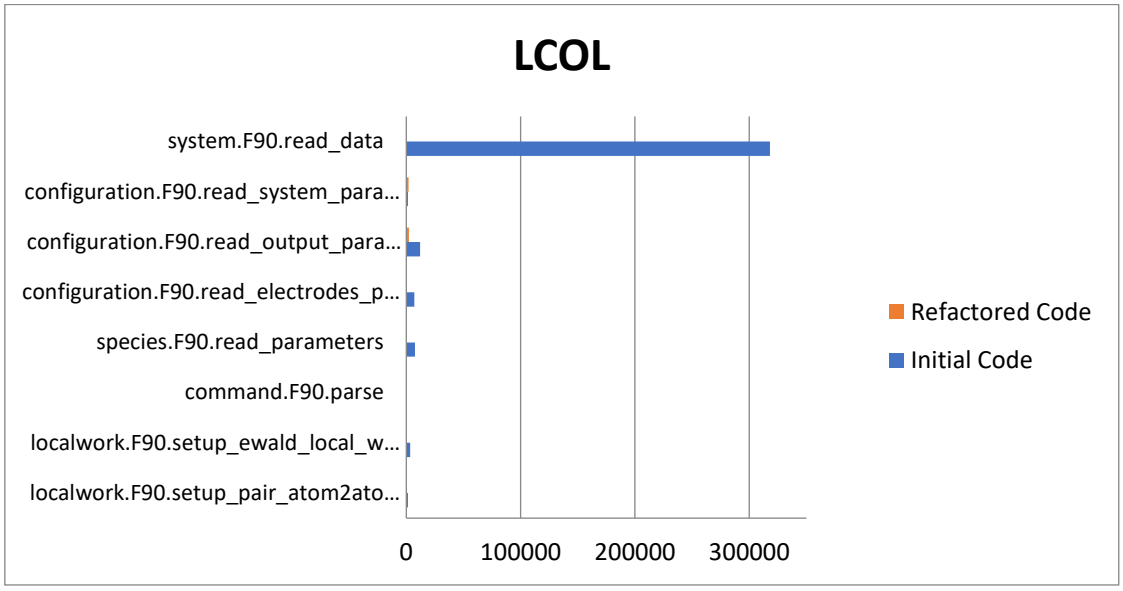
Επομένως, από την άποψη της απόδοσης, τα refactorings που προτείνουμε και εφαρμόσαμε είναι αποδεκτά. Τα αποτελέσματα της εφαρμογής των refactorings μας συζητούνται λεπτομερώς παρακάτω μαζί με μια γραφικές αναπαραστάσεις των αλλαγών στις μετρικές. Πρώτα, παρουσιάζουμε τις μετρικές επιπέδου διαδικασίας (CC και LCOL) και μετά τις μετρικές επιπέδου αρχείου (LOC, CBF, LCOP). Θα πρέπει να σημειωθεί ότι οι μετρήσεις του αναδιαρθρωμένου κώδικα υπολογίστηκαν κάθε φορά με τον μέσο όρο

της αντίστοιχης μετρική για όλες τις διαδικασίες ή τα αρχεία που άλλαξαν ή να προστέθηκαν μέσω του refactoring.

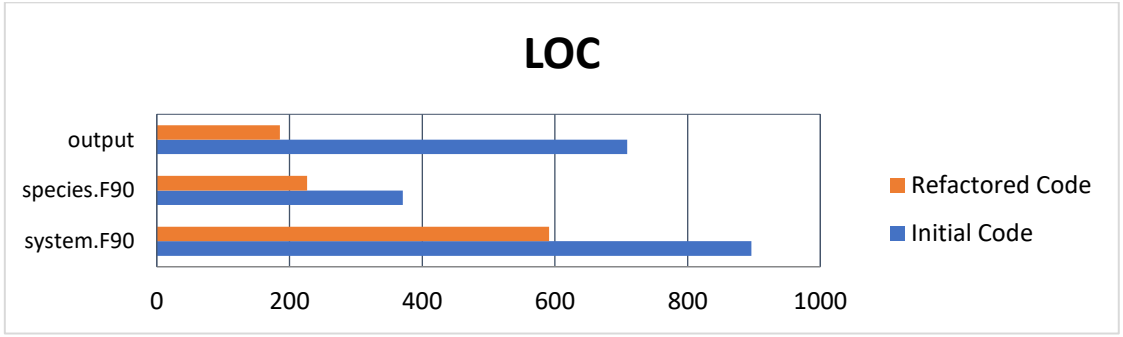
Όπως μπορεί να παρατηρηθεί, στη συντριπτική πλειονότητα των διαδικασιών στην Εικόνα 7-14, η τιμή της μετρικής της κυκλωματικής πολυπλοκότητας μειώνεται, λόγω των αναδιαμορφώσεων κώδικα (Extract Process & Extract File / Module) που εφαρμόζονται στις συγκεκριμένες διαδικασίες. Παρομοίως, η τιμή της μετρικής LOC (γραμμές κώδικα) μειώνεται για καθεμία από τις διαδικασίες που παρουσιάζονται στην Εικόνα 7-15, εξαιτίας των αναδιαμορφώσεων Extract File/Module (α) output.F90 (σε output_dump.F90, output_dump_freq.F90 και output_setup.F90), (β) species.F90 (σε species_input.F90) και (γ) system.F90 (input_data.F90). Η τιμή της μετρικής LCOL (Έλλειψη συνοχής των γραμμών) μειώνεται για καθεμία από τις διαδικασίες που παρουσιάζονται στην Εικόνα 7-16, λόγω των αναδιαμορφώσεων κώδικα (Extract Procedures & Extract File / Module) που εφαρμόζονται στα συγκεκριμένα αρχεία (πολλές γραμμές ο κώδικας διανεμήθηκε σε διαφορετικές νέες διαδικασίες). Σε όλα τα αρχεία / modules που υπάρχουν στην Εικόνα 7-17, η τιμή της μετρικής CBF μειώνεται ή παραμένει σταθερή, καθώς τα refactorings που εφαρμόστηκαν (Extract Process & Extract File / Module) δεν είχαν σκοπό βα την τροποποιήσουν. Στην πραγματικότητα, η CBF είναι το αποτέλεσμα των εφαρμοζόμενων αναδιαμορφώσεων (μειωμένη συνοχή, ελαφρώς αυξημένη σύζευξη).



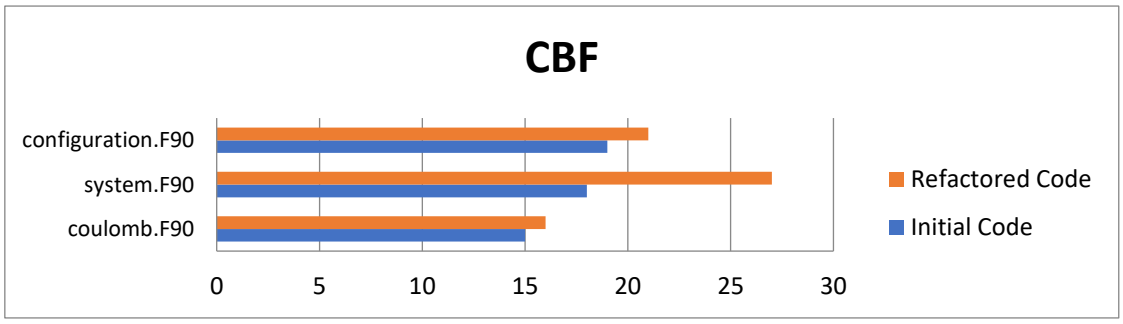
Εικόνα 7-14: Εκτέλεση MetalWalls, μετρική CC



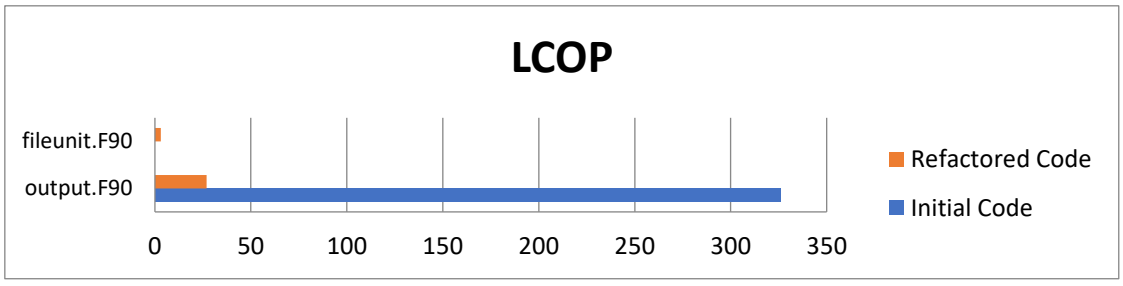
Εικόνα 7-15: Εκτέλεση MetalWalls, μετρική LCOL



Εικόνα 7-16: Εκτέλεση MetalWalls, μετρική LOC

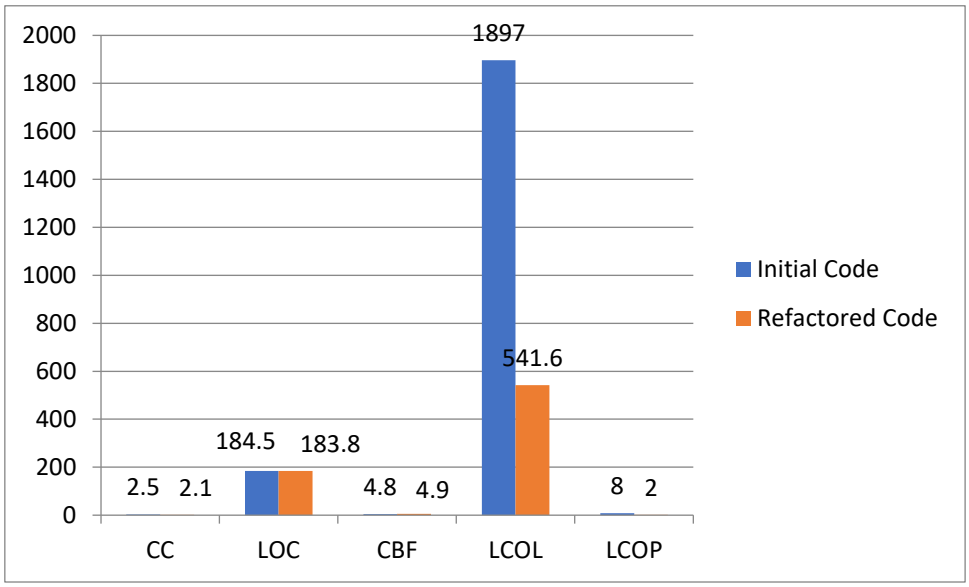


Εικόνα 7-17: Εκτέλεση MetalWalls, μετρική CBF



Εικόνα 7-18: Εκτέλεση MetalWalls, μετρική LCOP

Όπως μπορεί να παρατηρηθεί στην Εικόνα 7-18, η τιμή της μετρικής LCOP (έλλειψη συνοχής διαδικασιών) μειώνεται για το αρχείο output.F90, λόγω των refactorings (Extract Files / Modules & Extract διαδικασίες), αλλά αυξάνεται κατά 3 για το αρχείο fileunit.F90. Η αύξηση της τιμής της μετρικής LCOP για το αρχείο fileunit.F90, οφείλεται σε μια πρόσθετη διαδικασία (open_file) που αποφασίσαμε ότι εννοιολογικά ανήκει σε αυτό το module.



Εικόνα 7-19: Εκτέλεση MetalWalls, μετρικές συστήματος

Όπως μπορεί να παρατηρηθεί στην Εικόνα 7-19, σχεδόν όλες οι μτρικές του συστήματος, βελτιώθηκαν (μειώθηκαν) ή παρέμειναν σταθερές, λόγω των αναδιαμορφώσεων που εφαρμόσαμε (Extract File / Module and Extract Procedure). Μπορούμε να παρατηρήσουμε ότι η τιμή της μετρικής CBF έχει αυξηθεί ελαφρώς, ως αντιστάθμιση στα refactorings Extract Files / Modules που έχουμε κάνει. Με βάση τα παραπάνω, ο Τόκος ΤΧ μειώθηκε κατά 244,64 ευρώ και ισούται με 531,58 ευρώ ανά νέα έκδοση.

7.3 Εφαρμογή Στο LQCD Και KKRNano

Στο ευρωπαϊκό έργο EXA2PRO εντάσσετε και το ερευνητικό κέντρο Forschungszentrum Jülich όπου εκτελέστηκε το εργαλείο μας. Το JUELICH αναπτύσσει δύο έργα μεγάλης κλίμακας, το LQCD και το KKRNano

7.3.1 LQCD

Για την εφαρμογή LQCD, καταφέραμε να εντοπίσουμε 390 ζητήματα SonarQube και 15 ζητήματα σε επίπεδο σχεδίασης. Το συνολικό Κεφάλαιο ΤΧ αντιστοιχεί σε 1.724

ευρώ για το χρέος πηγαίου κώδικα και 106,99 ευρώ για το χρέος σε επίπεδο σχεδίασης. Ο Τόκος TX ισούται με: 103,53 ευρώ ανά νέα έκδοση. Στους παρακάτω πίνακες, τα αρχεία που εντοπίστηκαν ότι έχουν προβλήματα σχεδίασης απεικονίζονται μαζί με τις μετρικές τους (για μετρικές επιπέδου διαδικασίας παρέχεται ο μέσος όρος του αρχείου).

Πίνακας 7-17: LQCD, αρχεία με μεγάλες μετρήσεις CC

File	CC
fermion-lattice.cc	30
gauge-lattice.cc	16
c34.cc	8
unittest.cc	6
gauge-lattice.cc	6

Πίνακας 7-18: LQCD, αρχεία με μεγάλες μετρήσεις LCOL

File	LCOL
unittest.cc	1116.0
main.cc	193.0
gauge-lattice.cc	152.0

Πίνακας 7-19: LQCD, αρχεία με μεγάλες μετρήσεις LOC

File	LOC
c34.cc	255
gauge-lattice.cc	215

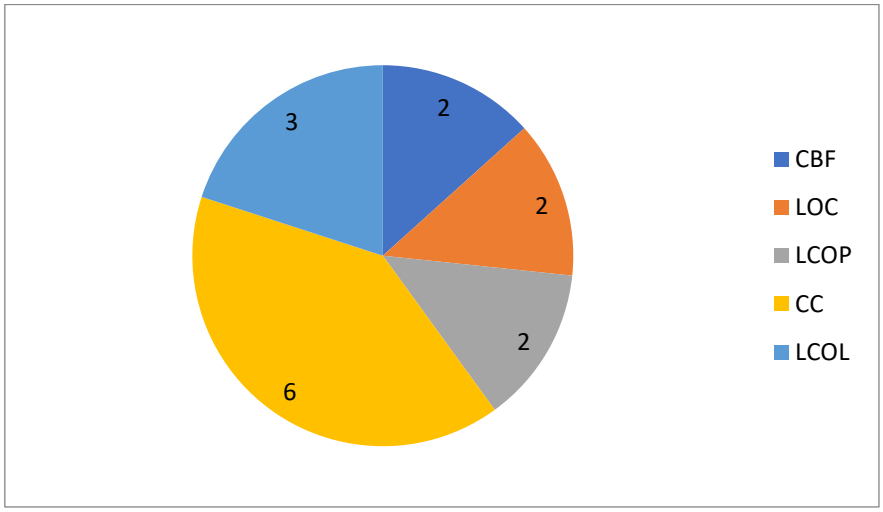
Πίνακας 7-20: LQCD, αρχεία με μεγάλες μετρήσεις CBF

File	CBF
gauge-lattice.h	3
gauge-lattice.cc	3

Πίνακας 7-21: LQCD, αρχεία με μεγάλες μετρήσεις LCOP

File	LCOP
c34.cc	58
c33.cc	26

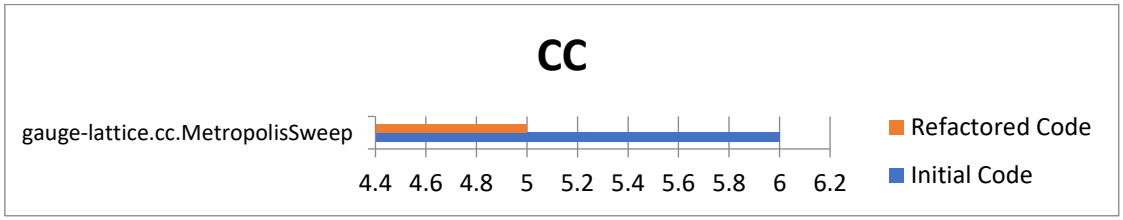
Ο αριθμός των σχεδιαστικών προβλημάτων ανά τύπο προβλήματος παρουσιάζεται μέσω ενός γραφήματος πίτας στην Εικόνα 7-20.



Εικόνα 7-20: Αριθμός σχεδιαστικών προβλημάτων ανά τύπο προβλήματος

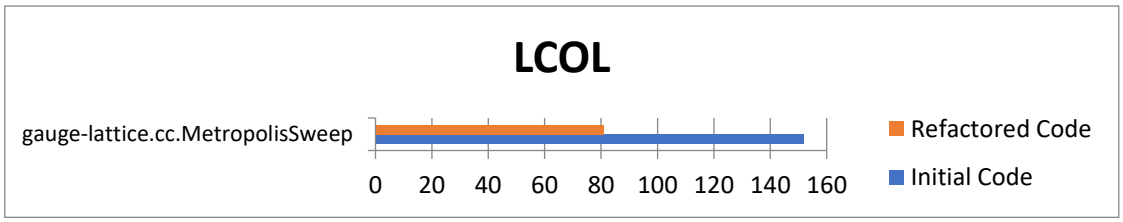
Πρέπει να σημειώσουμε ότι υπήρχαν λιγότερες ευκαιρίες, καθώς αυτό το έργο είναι μικρότερο σε έκταση από τα υπόλοιπα. Από τις προαναφερθείσες ευκαιρίες refactoring, εφαρμόσαμε μόνο 1 Extract Procedure. Με μια πρώτη ματιά από τον αντίστοιχο προγραμματιστή, η αναδιάρθρωση αυτή δεν ήταν ελκυστική λόγω του διαχωρισμού που έγιναν στα σχόλια του κώδικα. Αλλά εξετάζοντας περεταίρω τις αντιδράσεις του προγραμματιστή με τις απαντήσεις στο ερωτηματολόγιο, μπορούμε να δούμε ότι ο προγραμματιστής μπόρεσε να δώσει ένα συγκεκριμένο όνομα στη διαδικασία που δημιουργήσαμε, το οποίο σημαίνει ότι αυτό το κομμάτι κώδικα έχει μια συγκεκριμένη λειτουργικότητα. Από την στιγμή που υπάρχει συγκεκριμένη λειτουργικότητα σε αυτό το κομμάτι κώδικα, καθιστά την προτεινόμενη αναδιαμόρφωση εννοιολογικά σωστή. Επιπλέον, τα αποτελέσματα στη συντηρησιμότητα θα είναι σημαντικά καθώς ο αρχικός κώδικας βρίσκεται σε ένα πολύ κεντρικό σημείο του προγράμματος που είναι επιρρεπή σε αλλαγές. Τέλος, από την άποψη της απόδοσης, ο αναδιαμορφωμένος κώδικας είναι ένα χιλιοστό του δευτερολέπτου πιο αργός, το οποίο μπορεί να θεωρηθεί και αμελητέο.

Τα αποτελέσματα της εφαρμογής των refactorings μας συζητούνται λεπτομερώς παρακάτω μαζί με γραφικές αναπαραστάσεις των αλλαγών στις μετρικές. Αρχικά, παρουσιάζουμε τις μετρικές επιπέδου διαδικασίας (CC και LCOL) και έπειτα τις μετρικές επιπέδου αρχείου (LOC, CBF, LCOP). Θα πρέπει να σημειωθεί ότι η μέτρηση του refactored κώδικα υπολογίστηκε κάθε φορά με τον μέσο όρο της νέας μέτρησης όλων των διαδικασιών ή αρχείων που άλλαξαν ή προστέθηκαν.



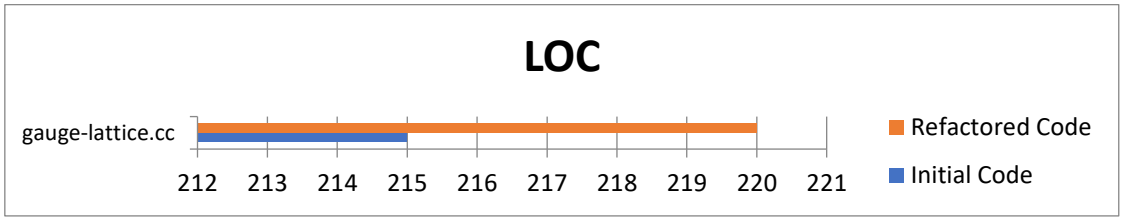
Εικόνα 7-21: Εκτέλεση LQCD, μετρική CC

Όπως μπορεί να παρατηρηθεί, στη διαδικασία στην Εικόνα 7-21, η τιμή της μετρικής της κυκλωματικής πολυπλοκότητας μειώνεται, λόγω της αναδιαμόρφωσης κώδικα (Extract Process) που εφαρμόστηκε στο "MetropolisSweep" (βρόγχοι ελέγχου διανεμήθηκαν σε πολλές διαφορετικές νέες διαδικασίες).



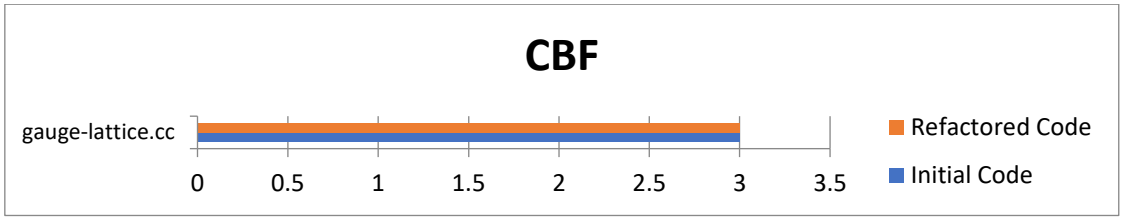
Εικόνα 7-22: Εκτέλεση LQCD, μετρική LCOL

Η τιμή της μετρικής LCOL (Έλλειψη συνοχής γραμμών) μειώθηκε στη διαδικασία που παρουσιάζεται στην Εικόνα 7-22, λόγω της αναδιαμόρφωσης κώδικα (Εξαγωγή διαδικασίας) που εφαρμόστηκε στο συγκεκριμένο αρχείο (πολλές γραμμές κώδικα διανεμήθηκαν σε διαφορετικές νέα διαδικασίες).



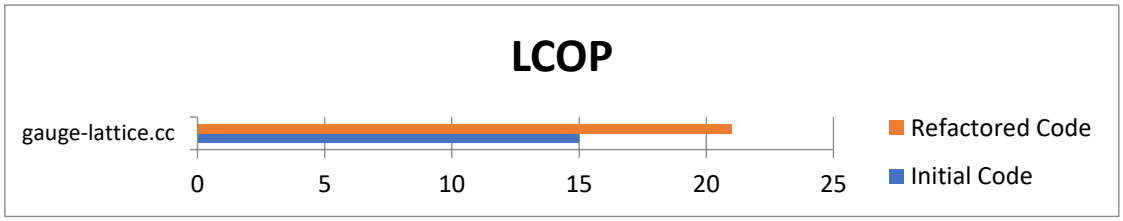
Εικόνα 7-23: Εκτέλεση LQCD, μετρική LOC

Η τιμή της μετρικής LOC αυξάνεται στο αρχείο που παρουσιάζεται στην Εικόνα 7-23, λόγω της αναδιαμόρφωσης Extract Procedure που εφαρμόσαμε στη διαδικασία "MetropolisSweep" (ο αρχικός κωδικός συν την κλήση της καινούργιας διαδικασίας και οι δηλώσεις των μεταβλητων).



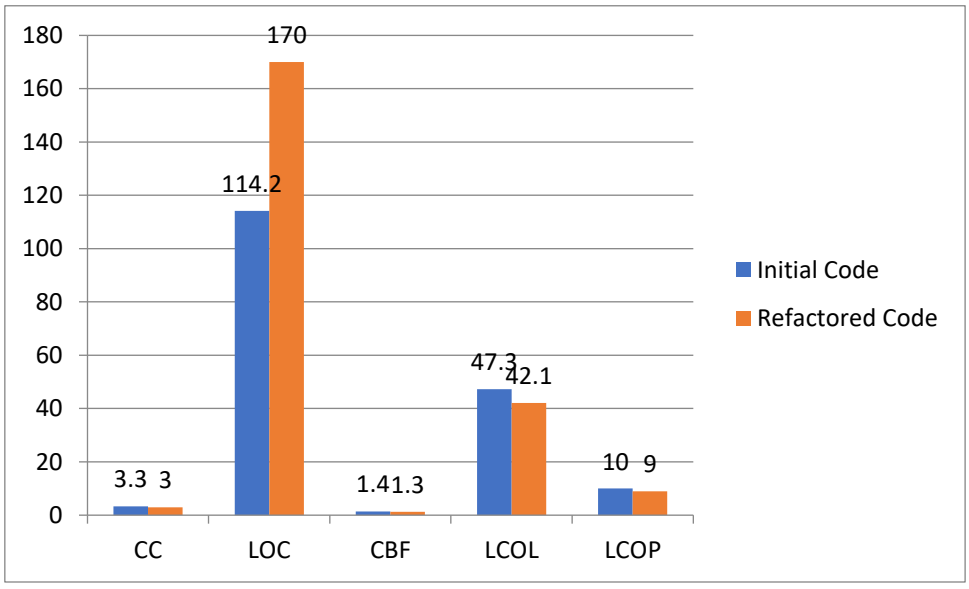
Εικόνα 7-24: Εκτέλεση LQCD, μετρική CBF

Στο αρχείο που παρουσιάζεται στην Εικόνα 7-24, η τιμή της μετρικής CBF παραμένει ως έχει, καθώς το refactoring (Extract Procedure) δεν προοριζόταν να μειώσει την τιμή της μετρικής CBF.



Εικόνα 7-25: Εκτέλεση LQCD, μετρική LCOP

Τέλος, όπως φαίνεται στην Εικόνα 7-25, η τιμή της μετρικής LCOP (έλλειψη συνοχής διαδικασιών) του αρχείου gauge-lattice.cc, αυξάνεται ελαφρώς, λόγω της διαδικασίας που δημιουργήθηκε (define_updates), η οποία αποφασίσαμε εννοιολογικά ανήκει σε αυτή την κλάση. Παρ 'όλα αυτά, όπως μπορεί να παρατηρηθεί στην Εικόνα 7-26, σχεδόν όλες οι μετρικές του συστήματος, έχουν βελτιωθεί (μειωθεί), λόγω της μιας αναδιαμόρφωσης που εφαρμόσαμε (Extract Procedure). Μπορούμε να παρατηρήσουμε ότι η τιμή της μετρικής LOC έχει αυξηθεί κατά σχεδόν 56 μονάδες, ως αντιστάθμιση στο refactoring που κάναμε. Τέλος, ο Τόκος TX παρέμεινε σχεδόν το ίδιο λόγω του μικρού αριθμού των refactorings



Εικόνα 7-26: Εκτέλεση LQCD, μετρικές συστήματος

7.3.2 KKRnano

Για την εφαρμογή KKRnano, καταφέραμε να εντοπίσουμε 1.379 ζητήματα SonarQube και 93 ζητήματα σε επίπεδο σχεδίασης. Το συνολικό Κεφάλαιο TX αντιστοιχεί

σε 6.982,8 ευρώ για χρέος πηγαίου κώδικα και 636,85 ευρώ για χρέος σχεδιασμού. Ο Τόκος TX ισούται με: 685,74 ευρώ ανά νέα έκδοση. Στους πίνακες 7-22 με 7-26, τα αρχεία με θέματα σχεδίασης απεικονίζονται μαζί με τις μετρικές τους. Πρέπει να σημειώσουμε ότι για τους πίνακες σχετικά με τις μετρήσεις CC και LCOL παρουσιάζουμε μόνο τα 20 αρχεία με τις μεγαλύτερες μετρικές λόγω περιορισμένου χώρου.

Πίνακας 7-22: KKRnano, αρχεία με μεγάλες μετρήσεις CC

File	CC
zgemm_avx2.c	16
SparseMatrix_mod.F90	8.5
tfQMR_mod.F90	7.1
bsrmm_mod.F90	6.1
zrandn.f90	6
test_zgemm.c	5
viz.F90	4
mod_compute.f90	3.5
zgemm_sve.c	3
TimerMpi_mod.F90	2.3
zgemm_neon.c.	2
KKROperator_mod.F90	2
nf_codelets.f90	2
zdotu_neon.c	1
zaxpy_sve.c	1
zxpay_sve.c	1
zxpay_avx2.c	1
zxpay_neon.c	1
zaxpy_neon.c	1
zdotu_avx2.c	1

Πίνακας 7-23: KKRnano, αρχεία με μεγάλες μετρήσεις LCOL

File	LCOL
test_zgemm.c	12728.0
tfQMR_mod.F90	8143.6

zgemm_kernel_inner_avx2.h	5860.0
zgemm_neon.c	3912.0
zgemm_sve.c	2283.0
zdotu_avx2.c	1958.0
zxpay_avx2.c	1452.0
zaxpy_avx2.c	1452.0
test_zaxpy.c	1337.0
zdotu_sve.c	1274.0
test_zdotu.c	1119.0
fstarpu_mod.f90	1075.5
test_zxpay.c	760.0
bsrmm_mod.F90	748.6
zxpay_sve.c	633.0
zaxpy_sve.c	633.0
SparseMatrix_mod.F90	476.5
zdotu_neon.c	429.0
nf_codelets.f90	315.0
viz.F90	293.0

Πίνακας 7-24: KKRnano, αρχεία με μεγάλες μετρήσεις LOC

File	LOC
fstarpu_mod.f90	1745
tfQMR_mod.F90	464
bsrmm_mod.F90	424
zgemm_kernel_inner_avx2.h	353
test_zgemm.c	263

Πίνακας 7-25: KKRnano, αρχεία με μεγάλες μετρήσεις CBF

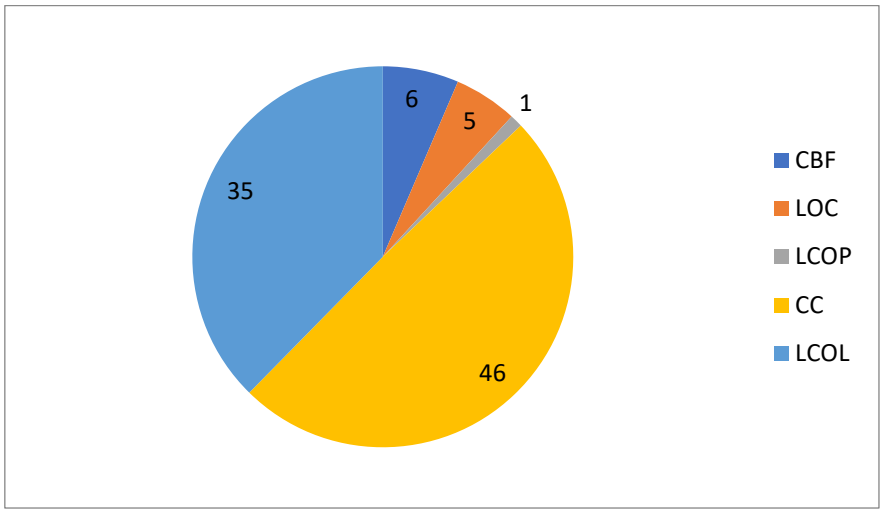
File	CBF
tfQMR_mod.F90	11
bsrmm_mod.F90	8
miniKKR.F90	6

mod_compute.f90	4
KKROperator_mod.F90	4
zgemm_f.F90	4

Πίνακας 7-26: KKRnano, αρχεία με μεγάλες μετρήσεις LCOP

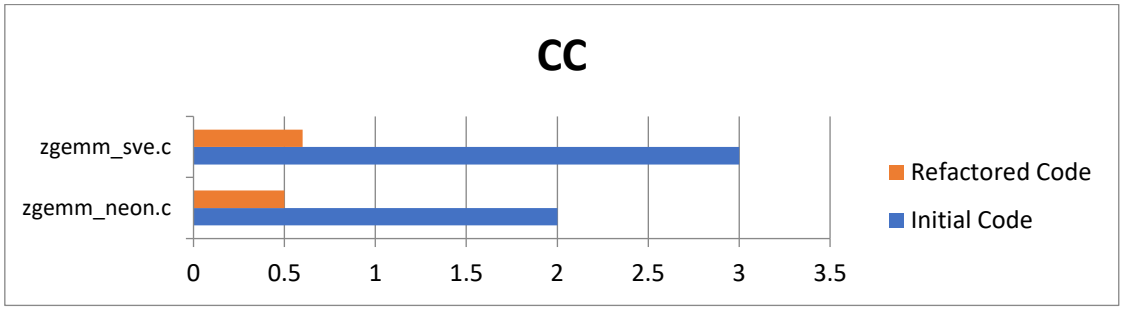
File	LCOP
fstarpu_mod.f90	38781

Ο αριθμός των σχεδιαστικών προβλημάτων ανά τύπο προβλήματος παρουσιάζεται μέσω ενός γραφήματος πίτας στην Εικόνα 7-27.



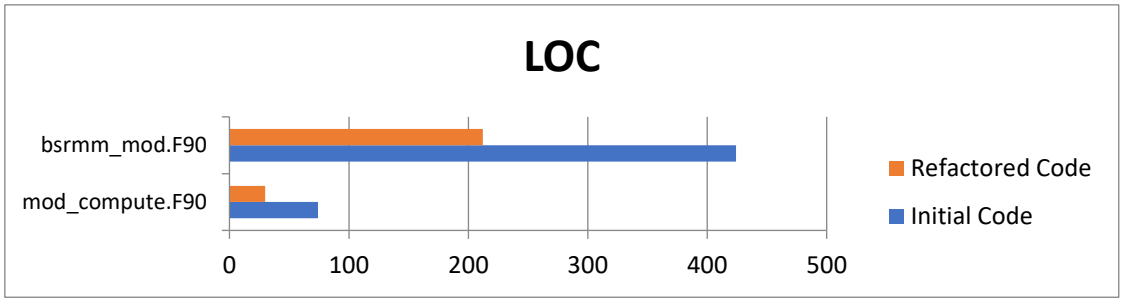
Εικόνα 7-27: Αριθμός σχεδιαστικών προβλημάτων ανά τύπο προβλήματος

Από τις προαναφερθείσες ευκαιρίες refactoring, εφαρμόσαμε 7 Extract Procedure και 1 Extract Files / Module. Για τα refactorings αυτού του έργου, δεν έχουμε πλήρη πληροφόρηση από το ερωτηματολόγιο, αλλά μπορούμε να πούμε ότι παρά την αρχική απόρριψη των refactorings στη συνέντευξη, μπορούμε να δούμε ότι στις απαντήσεις του ερωτηματολόγιου υπάρχει μια θετική πλευρά. Πρώτα απ' όλα, το Extract File / Module, το οποίο είναι test cases, σημειώθηκε ότι μπορεί να υιοθετηθεί από τον προγραμματιστή. Όσον αφορά τις διαδικασίες εξαγωγής (Extract Procedures), ο προγραμματιστής μπόρεσε να δώσει ένα όνομα για την διαδικασία του εξαγόμενου κώδικα, το οποίο αποδεικνύει την εννοιολογική ορθότητα των προτεινόμενων refactorings. Τέλος, τα refactorings χρίζουν περισσότερη επικύρωση για να μπορέσουν οι προγραμματιστές να τα ενσωματώσουν στον κώδικα τους, καθώς και στον υπολογισμό της απόδοσης.



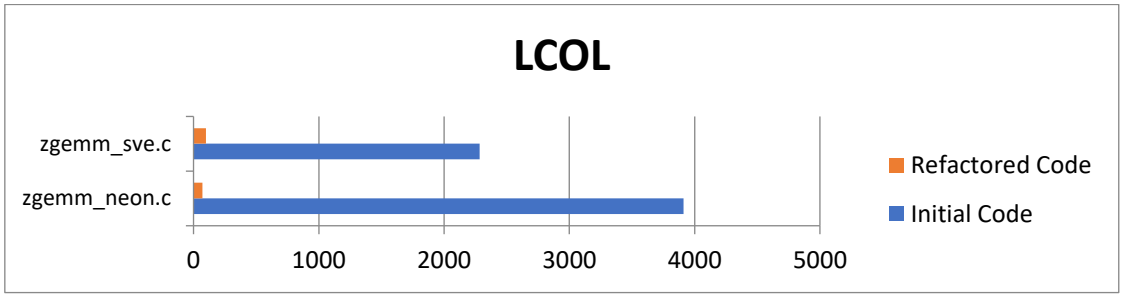
Εικόνα 7-28: Εκτέλεση KKRnano, μετρική CC

Όπως μπορεί να παρατηρηθεί, σε όλες τις διαδικασίες στην Εικόνα 7-28, η τιμή της μετρικής της κυκλωματικής πολυπλοκότητας μειώνεται, λόγω των αναδιαμορφώσεων κώδικα (Extract Procedure & Extract File / Module) που εφαρμόστηκαν.



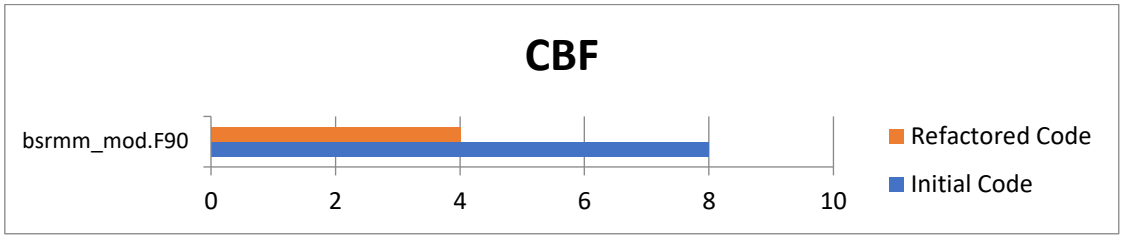
Εικόνα 7-29: Εκτέλεση KKRnano, μετρική LOC

Παρομοίως, η τιμή της μετρικής LOC μειώνεται για καθεμία από τα αρχεία που παρουσιάζονται στην Εικόνα 7-29, εξαιτίας των αναδιαμορφώσεων Extract File / Module του (α) mod_compute.F90 (σε mod_compute_copy.F90 και mod_compute_loop. F90) και (β) bsrmm_mod.F90 (σε bsrmm_test.F90).



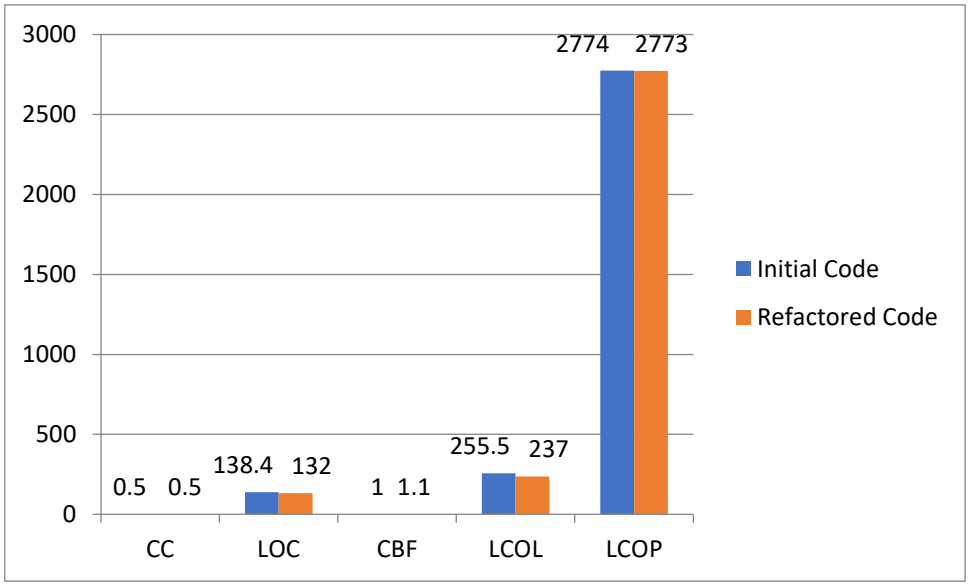
Εικόνα 7-30: Εκτέλεση KKRnano, μετρική LCOL

Η τιμή της μετρικής LCOL (Έλλειψη συνοχής γραμμών) μειώνεται για καθένα από τα αρχεία που παρουσιάζονται στην Εικόνα 7-30, λόγω των αναδιαμορφώσεων κώδικα (Extract Procedures & Extract File / Module) που εφαρμόζονται στα συγκεκριμένα αρχεία (πολλές γραμμές ο κώδικας διανεμήθηκαν σε διαφορετικές νέες διαδικασίες ή αρχεία).



Εικόνα 7-31: Εκτέλεση KKRnano, μετρική CBF

Επιπλέων στην Εικόνα 7-31 βλέπουμε την μείωση της CBF μετρικής λόγο του Extract File / Module. Τέλος όπως φαίνεται παρακάτω, σχεδόν όλες οι τιμές των μετρικών του συστήματος βελτιώθηκαν (μειώθηκαν) ή παρέμειναν οι ίδιες, λόγω των αναδιαμορφώσεων που εφαρμόσαμε (Extract File / Module και Extract Procedure). Μπορούμε να παρατηρήσουμε ότι η τιμή της μετρικής CBF έχει αυξηθεί ελαφρώς, ως αντιστάθμιση των αναδιαμορφώσεων Extract Files / Modules (σε mod_compute.F90 και bsrmm_mod.F90). Τέλος, αξίζει να σημειωθεί ότι ο Τόκος TX είχε μια μικρή βελτίωση, καθώς μειώθηκε από 685,74 σε 652,37 ευρώ ανά νέα έκδοση.



Εικόνα 7-32: Εκτέλεση KKRnano, μετρικές συστήματος

8 Επίλογος

8.1 Σύνοψη και συμπεράσματα

Σε αυτή την εργασία αρχικά παρουσιάστηκε μια προσέγγιση για την ποσοτικοποίηση του τεχνικού χρέους για προγράμματα υψηλής επίδοσης ώστε να εντοπίζονται προβλήματα σχεδιαστικού επιπέδου. Συγκεκριμένα, εκτιμήθηκε το Κεφάλαιο ΤΧ για κάθε τύπο σχεδιαστικού προβλήματος. Επιπλέον παρουσιάστηκαν οι τεχνικές αναδιαμόρφωσης για την αντιμετώπιση των προαναφερθέντων προβλημάτων σχεδιασμού. Συγκεκριμένα, αναφερθήκαμε στον αλγόριθμο SEMI με κάποιες τροποποιήσεις για την αποσύνθεση μεγάλων διαδικασιών και προσαρμόσαμε την τεχνική Agglomerative Clustering για την αποσύνθεση μεγάλων αρχείων / modules σε πιο συνεκτικά. Επίσης, χρησιμοποιώντας τους υπολογισμούς Τόκου και Κεφαλαίου του ΤΧ, καταφέραμε να εκτιμήσουμε τις αντισταθμίσεις των βελτιστοποιήσεων χρόνου και φορητότητας που προσφέρουν τα εργαλεία StarPU και SkePU. Τα αποτελέσματα έδειξαν ότι στην πλειονότητα των περιπτώσεων, η επίδραση των βελτιστοποιήσεων στο ΤΧ είναι ουδέτερη ή αρνητική, ειδικά στον κώδικα που έχει αναδιαμορφωθεί. Ακόμη, το εργαλείο που αναπτύχθηκε είναι ένα εύκολο στην χρήση εργαλείο με αρκετές λειτουργίες ενσωματωμένες και αυτές που δεν είναι, ενσωματώθηκαν με τέτοιο ώστε να είναι ευκολά προσαρμόσιμες.

Όσο αφορά τα εμπειρικά αποτελέσματα της αποπληρωμής τεχνικού χρέους μέσω εφαρμογής των προτεινόμενων αναδιαμορφώσεων, αποκάλυψε ότι η συντήρηση των έργων μπορεί να βελτιωθεί σημαντικά. Για παράδειγμα, τα refactorings που εφαρμόστηκαν στις εφαρμογές CO_2Capture, Metalwalls και KKRnano μείωσαν τον Τόκο ΤΧ κατά 21,9%, 31,5% και 4,8%, αντίστοιχα. Ταυτόχρονα, η εφαρμογή αυτών των refactorings στην απόδοση των αντίστοιχων εφαρμογών ήταν ελάχιστη: Για το CO_2Capture παρατηρήσαμε βελτίωση 0,4% και επιδείνωση 0,5% στο χρόνο εκτέλεσης, ανάλογα με τα refactorings που έχουν εφαρμοστεί. Για το Metalwalls τα αποτελέσματα ποικίλλουν ανάλογα με τα δεδομένα της περίπτωσης (Supercap, Blue Energy, Graphene) και από το αν χρησιμοποιήσαμε 1 ή 20 πυρήνες. Τα αποτελέσματα αυτά κυμαίνονταν από βελτίωση κατά 3,98% έως επιδείνωση 1,81%. Έτσι, υπάρχουν επαρκή αποδεικτικά στοιχεία για να υποστηριχθεί ο ισχυρισμός ότι οι βελτιώσεις σε επίπεδο σχεδιασμού και κώδικα, σε εφαρμογών υψηλής υπολογιστικής ισχύος μπορούν να αυξήσουν το επίπεδο συντήρησής τους χωρίς να βλάψουν την απόδοσή τους.

8.2 Όρια και περιορισμοί της έρευνας

Ο κύριος περιορισμός αυτής της εργασίας είναι ο αριθμός των έργων που χρησιμοποιήθηκαν. Όσο αφορά τα εμπειρικά αποτελέσματα για την αποπληρωμή του τεχνικού χρέους χρησιμοποιήθηκαν τέσσερα έργα, ενώ για την επίδραση τον βελτιστοποιήσεων ταχύτητας και φορτικότητας στο τεχνικό χρέος χρησιμοποιήθηκαν έξι έργα. Ο αριθμός των έργων είναι μικρός, για να μπορέσουμε να καταλήξουμε σε στατιστικά σημαντικά αποτελέσματα, παρόλο που τα αποτελέσματα φαίνεται να είναι ομοιόμορφα. Αυτήν η απόφαση πάρθηκε λόγο της επικοινωνίας που ήμασταν σε θέση να έχουμε με τους προγραμματιστές των έργων αυτόν. Επιπλέον, τα αποτελέσματα σχετικά με τις αναδομήσεις λογισμικού δεν μπορούν να γενικευθούν σε άλλες αναδομήσεις, ή τεχνικές βελτίωσης ποιότητας.

8.3 Μελλοντικές Επεκτάσεις

Η μελλοντικές επεκτάσεις της εργασίας αυτής μπορούν να χωριστούν σε δύο κατευθύνσεις. Πρώτον, όσον αφορά το ερευνητικό κομμάτι θα μπορούσαν να βρεθούν και άλλα έργα ώστε να αυξηθεί το πλήθος και η αξιοπιστία των αποτελεσμάτων, καθώς και να εξεταστούν και άλλα εργαλεία βελτιστοποίησης. Επιπλέων, θα μπορούσε να επεκταθεί το εργαλείο που αναπτύχθηκε ώστε να παρέχονται περισσότερες πληροφορίες μέσω από αυτό και η διαχείριση του τεχνικού χρέους να είναι πιο εύκολη. Συγκεκριμένα υπάρχουν ήδη σχέδια και ανάπτυξη σε αρχικά στάδια για την ενσωμάτωση λειτουργιών πρόβλεψης του τεχνικού χρέους και αρχιτεκτονική λήψη αποφάσεων ως χρηματοοικονομική επένδυση.

9 Βιβλιογραφία

Al Dallal, J., 2010. Measuring the discriminative power of object-oriented class cohesion metrics. IEEE Transactions on Software Engineering, 37(6), pp.788-804.

Alves, N.S., Mendes, T.S., de Mendonça, M.G., Spínola, R.O., Shull, F. and Seaman, C., 2016. Identification and management of technical debt: A systematic mapping study. Information and Software Technology, 70, pp.100-121.

Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P., Abrahamsson, P., Martini, A., Zdun, U. and Systa, K., 2016, October. The perception of technical debt in the embedded systems domain: an industrial case study. In 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD) (pp. 9-16). IEEE.

Arvanitou, E.M., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A. and Stamelos, I., 2019. Monitoring technical debt in an industrial setting. In Proceedings of the Evaluation and Assessment on Software Engineering (pp. 123-132).

Arvanitou, E.M., Ampatzoglou, A., Chatzigeorgiou, A. and Avgeriou, P., 2017, June. A method for assessing class change proneness. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (pp. 186-195).

Avgeriou, P., Kruchten, P., Ozkaya, I. and Seaman, C., 2016. Managing technical debt in software engineering (dagstuhl seminar 16162). In Dagstuhl Reports (Vol. 6, No. 4). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Avgeriou, P.C., Taibi, D., Ampatzoglou, A., Fontana, F.A., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, N., Pigazzini, I. and Saarimaki, N., 2020. An overview and comparison of technical debt measurement tools. IEEE Software.

Badri, L. and Badri, M., 2004. A Proposal of a new class cohesion criterion: an empirical study. Journal of Object Technology, 3(4), pp.145-159.

Barney, S., Petersen, K., Svahnberg, M., Aurum, A. and Barney, H., 2012. Software quality trade-offs: A systematic map. Information and software technology, 54(7), pp.651-662.

Bieman, J.M. and Kang, B.K., 1995. Cohesion and reuse in an object-oriented system. ACM SIGSOFT Software Engineering Notes, 20(SI), pp.259-262.

Birdsall, C.K. and Langdon, A.B., 2004. Plasma physics via computer simulation. CRC press.

Buyens, K., Scandariato, R. and Joosen, W., 2009, October. Measuring the interplay of security principles in software architectures. In 2009 3rd International Symposium on Empirical Software Engineering and Measurement (pp. 554-563). IEEE.

Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., Gkortzis, A. and Avgeriou, P., 2016. Identifying extract method refactoring opportunities based on functional relevance. IEEE Transactions on Software Engineering, 43(10), pp.954-974.

Charalampidou, S., Ampatzoglou, A. and Avgeriou, P., 2015, October. Size and cohesion metrics as indicators of the long method bad smell: An empirical study. In Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering (pp. 1-10).

Chatzigeorgiou, A., Ampatzoglou, A., Ampatzoglou, A. and Amanatidis, T., 2015, October. Estimating the breaking point for technical debt. In 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD) (pp. 53-56). IEEE.

Chidamber, S.R. and Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6), pp.476-493.

Conejero, J.M., Rodríguez-Echeverría, R., Hernández, J., Clemente, P.J., Ortiz-Caraballo, C., Jurado, E. and Sánchez-Figueroa, F., 2018. Early evaluation of technical debt impact on maintainability. Journal of Systems and Software, 142, pp.92-114.

Cunningham, W., 1992. The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger, 4(2), pp.29-30.

DeMarco, T., 1979. Structure analysis and system specification. In Pioneers and Their Contributions to Software Engineering (pp. 255-288). Springer, Berlin, Heidelberg.

Eisenberg, R., 2013, October. Management of technical debt: a lockheed martin experience report. In 3rd International Workshop on Managing Technical Debt (MTD'13), Baltimore, USA.

Feitosa, D., Ampatzoglou, A., Avgeriou, P. and Nakagawa, E.Y., 2015, May. Investigating quality trade-offs in open source critical embedded systems. In 2015 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA) (pp. 113-122). IEEE.

Ferreira, K.A., Bigonha, M.A., Bigonha, R.S., Mendes, L.F. and Almeida, H.C., 2012. Identifying thresholds for object-oriented software metrics. Journal of Systems and Software, 85(2), pp.244-257.

Fokaefs, M., Tsantalis, N., Chatzigeorgiou, A. and Sander, J., 2009, September. Decomposing object-oriented class modules using an agglomerative clustering technique. In 2009 IEEE International Conference on Software Maintenance (pp. 93-101). IEEE.

Fowler, M., 1997, June. Refactoring: Improving the design of existing code. In 11th European Conference. Jyväskylä, Finland.

Heaton, D. and Carver, J.C., 2015. Claims about the use of software engineering practices in science: A systematic literature review. Information and Software Technology, 67, pp.207-219.

Hitz, M. and Montazeri, B., 1995. Measuring coupling and cohesion in object-oriented systems (pp. 25-27). na.

Kruchten, P., Nord, R.L. and Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. Ieee software, 29(6), pp.18-21.

Li, Z., Avgeriou, P. and Liang, P., 2015. A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101, pp.193-220.

Li, W. and Henry, S., 1993. Object-oriented metrics that predict maintainability. Journal of systems and software, 23(2), pp.111-122.

MacCormack, A. and Sturtevant, D.J., 2016. Technical debt and system architecture: The impact of coupling on defect-related activity. Journal of Systems and Software, 120, pp.170-182.

Marinescu, R., 2004, September. Detection strategies: Metrics-based rules for detecting design flaws. In 20th IEEE International Conference on Software Maintenance, 2004. Proceedings. (pp. 350-359). IEEE.

Martin, R.C., 2002. Agile software development: principles, patterns, and practices. Prentice Hall.

McCabe, T.J., 1976. A complexity measure. IEEE Transactions on software Engineering, (4), pp.308-320.

Parnas, D.L., 1994, May. Software aging. In Proceedings of 16th International Conference on Software Engineering (pp. 279-287). IEEE.

Riaz, M., Mendes, E. and Tempero, E., 2009, October. A systematic review of software maintainability prediction and metrics. In 2009 3rd International Symposium on Empirical Software Engineering and Measurement (pp. 367-377). IEEE.

Runeson, P., Host, M., Rainer, A. and Regnell, B., 2012. Case study research in software engineering: Guidelines and examples. John Wiley & Sons.

Saarimaki, N., Baldassarre, M.T., Lenarduzzi, V. and Romano, S., 2019, August. On the accuracy of sonarqube technical debt remediation time. In 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 317-324). IEEE.

Seaman, C. and Guo, Y., 2011. Measuring and monitoring technical debt. In Advances in Computers (Vol. 82, pp. 25-46). Elsevier.

Silva, D., Terra, R. and Valente, M.T., 2014, June. Recommending automated extract method refactorings. In Proceedings of the 22nd International Conference on Program Comprehension (pp. 146-156).

Sletholt, M.T., Hannay, J., Pfahl, D., Benestad, H.C. and Langtangen, H.P., 2011, May. A literature review of agile practices and their effects in scientific software

development. In Proceedings of the 4th international workshop on software engineering for computational science and engineering (pp. 1-9).

Van Vliet, H., Van Vliet, H. and Van Vliet, J.C., 2008. Software engineering: principles and practice (Vol. 13). Hoboken, NJ: John Wiley & Sons.

Yli-Huumo, J., Maglyas, A. and Smolander, K., 2016. How do software development teams manage technical debt?–An empirical study. Journal of Systems and Software, 120, pp.195-218.

Zazworka, N., Izurieta, C., Wong, S., Cai, Y., Seaman, C. and Shull, F., 2014. Comparing four approaches for technical debt identification. Software Quality Journal, 22(3), pp.403-426.

10 Παράρτημα - Ευκαιρίες Extract Procedure

Πίνακας 10-1: Extract Procedure στο αρχείο configuration.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
read_system_parameters()	open_file()	5
	check_if_defined()	32
	check_if_specified_correctly()	8
	validate_values()	8
	check_if_gaussian_charge_distributed()	5
	check_if_mobile()	5
read_output_parameters()	read_parameters_values()	48
read_electrodes_parameters()	void_current_values()	6
	set_up_electrode_types()	15
	store_values()	10
	define_algorithm_parameters()	14

Πίνακας 10-2: Extract Procedure στο αρχείο species.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
read_parameters()	read_length_parameters()	35
	read_name_parameters()	8
	read_charge_type_keyword()	10
	read_charge_parameters()	14
	read_count_parameters()	5

Πίνακας 10-3: Extract Procedure στο αρχείο species.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
parse()	get_arg_max_length()	7

	read_arguments()	10
	check_positional_arguements()	13
	check_last_option()	9

Πίνακας 10-4: Extract Procedure στο αρχείο localwork.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
setup_ewald_local_work()	calculate_num_modes()	6
	determine_num_of_effective_modes()	25
	balance_comp_effort()	8
	determine_kstart()	13
set_pair_atom2atom_local_work()	distribute_diagonal_blocks()	8
	distribute_offdiagonal_blocks()	3

Πίνακας 10-5: Extract Procedure στο αρχείο localwork.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
print_type()	print_ion_to_ion()	3
	print_atom_to_atom()	1
	print_ion_to_atom()	1
	print_atom_to_ion()	1

Πίνακας 10-6: Extract Procedure στο αρχείο localwork.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
MW_timers_print()	print_breakdown_header()	2
	print_timer_main()	4
	print_icons2atoms_potential()	2
	print_atoms2atoms_potential()	2

	print_ions_coulomb_forces()	1
	print_ions_coulomb_potential()	1
	print_rattles()	1
	print_lennard_jones()	1
	print_diagnostics()	1
	print_mpi()	1

Πίνακας 10-7: Extract Procedure στο αρχείο coulomb_lr.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
melt2elec_potential()	melt2elec_compute_ions()	28
	melt2elec_compute_elec()	7
elec2elec_potential()	elec2elec_compute_1()	15
	elec2elec_compute_2()	7

Πίνακας 10-8: Extract Procedure στο αρχείο coulomb.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
melt2elec_potential()	melt2elec_long_range_contr()	8
	melt2elec_long_range_contr_from_k_zero()	3
	melt2elec_short_range_contr()	3
elec2elec_potential()	elec2elec_long_range_contr()	30
	elec2elec_long_range_contr_from_k_zero()	4
	elec2elec_short_range_contr()	4
	self_interaction_correction()	3

Πίνακας 10-9: Extract Procedure στο αρχείο coulomb_lr.cpp

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
	MW_coulomb_lr_melt2elec_compute_ions()	35

MW_coulomb_lr_melt2elec_potential()	MW_coulomb_lr_melt2elec_compute_elec()	9
MW_coulomb_lr_elec2elec_potential()	MW_coulomb_lr_elec2elec_compute_elec()	7

Πίνακας 10-10: Extract Procedure στο αρχείο coulomb_lr.cpp

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
melt2elec_potential()	melt2elec_long_range_contr()	7
	melt2elec_compute_blocks_with_full_interactions()	20
elec2elec_potential()	elec2elec_long_range_contr_from_k_zero()	7
	elec2elec_compute_other_blocks_with_full_interactions()	20
	elec2elec_compute_diag_blocks()	15
	elec2elec_compute_tri_blocks_with_full_interactions()	20

Πίνακας 10-11: Extract Procedure στο αρχείο cg.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
solve()	setup()	6
	finalize()	11

Πίνακας 10-12: Extract Procedure στο αρχείο electrode_parameters.F90

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
read_parameters()	check_if_defined()	6

Πίνακας 10-13: Extract Procedure στο αρχείο element.f

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)

ELEMENT_ORIGINAL()	Liquid_Inlets_Connection()	25
	Vapor_Inlets_Connection()	13
	Nonlinear_Constraints_Evaluation()	12
	N2_Mass_Balances()	13
	Liquid_Bottom_Stream()	14
	CO2_Partial_Pressure()	17
	MEA_Partial_Pressure()	16
	H2O_Partial_Pressure()	25
	Collocation_Points()	14
	Vn_Enthalpy_calculation()	10
	Lo_Enthalpy()	8
	Bottom_Total_Liquid()	6
	Top_Total_Vapor()	4

Πίνακας 10-14: Extract Procedure στο αρχείο frpmodel.f

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
HEXCH()	HECMB_Rich_Stream()	13
	HECMB_Lean_Stream()	12
	HETMB_Rich_Stream()	11
	HETMB_Lean_Stream()	14
	Cold_Inlet_from_Absorber()	10
	Cold_Oulet_to_Stripper()	12
	Hot_Inlet_from_Stripper()	18
	Hot_Outlet_to_Absorber()	14
	Heat_Exchanger_Energy_Balance()	24
	Heat_Exchanger_Area_Calculation()	10
HEATER()	Heater_MEA_Partial_Pressure()	23
	Heater_Mass_Balance()	17
	Heater_Reboiler_Mass_Balance()	10

	Heater_Equilibrium_Relations()	13
	Heater_Equilibrium()	7
	Heater_Equilibrium_Relations_H2O()	6
	Heater_Reboiler_Energy_Balance()	6
	Heater_Summation_Equations()	3
read_electrodes_parameters()	Heater2_MEA_Partial_Pressure()	10
	Heater2_Mass_Balance()	2
	Heater2_Reboiler_Mass_Balance()	1
	Heater2_Equilibrium_Relations()	1
	Heater2_Equilibrium()	1
	Heater2_Equilibrium_Relations_H2O()	1
	Heater2_Reboiler_Energy_Balance()	1
	Heater2_Summation_Equations()	1

Πίνακας 10-15: Extract Procedure στο αρχείο frpmainmn.f

Μεγάλη Διαδικασία	Extracted Διαδικασία	Χρόνος Επίλυσης (σε λεπτά)
FUNOBJ()	Funobj_Hex_Cost_Estimation()	5
	Funobj_Absorber_Cost_Estimation()	4
	Funobj_Stripper_Cost_Estimation()	4
	Funobj_Reboiler_Cost_Estimation()	3
	Funobj_Centrifugal_Pump_Cost_Estimation()	6
	Funobj_Utilities_Cost()	5
	Funobj_CO2_Emissions_Cost()	3
	Funobj_Objective_Function()	13

11 Παράρτημα - Ευκαιρίες Extract File / Module

Πίνακας 11-1: Extract File / Module στο αρχείο system.F90

Μεγάλο Αρχείο	Extracted Αρχείο	Χρόνος Επίλυσης (σε λεπτά)
system.F90	input_data.F90	20

Πίνακας 11-2: Extract File / Module στο αρχείο species.F90

Μεγάλο Αρχείο	Extracted Αρχείο	Χρόνος Επίλυσης (σε λεπτά)
species.F90	species_input.F90	14

Πίνακας 11-3: Extract File / Module στο αρχείο frpmodel.f

Μεγάλο Αρχείο	Extracted Αρχείο	Χρόνος Επίλυσης (σε λεπτά)
frpmodel.f	heater.f heater2.f	13

Πίνακας 11-4: Extract File / Module στο αρχείο reproducible.F90

Μεγάλο Αρχείο	Extracted Αρχείο	Χρόνος Επίλυσης (σε λεπτά)
reproducible.F90	read_reproducible.F90 write_reproducible.F90	12

Πίνακας 11-5: Extract File / Module στο αρχείο frpmodel.f

Μεγάλο Αρχείο	Extracted Αρχείο	Χρόνος Επίλυσης (σε λεπτά)
frpmainmn.f	funobj_cost_estimators.f funobj_real_cost_estimators.f	35