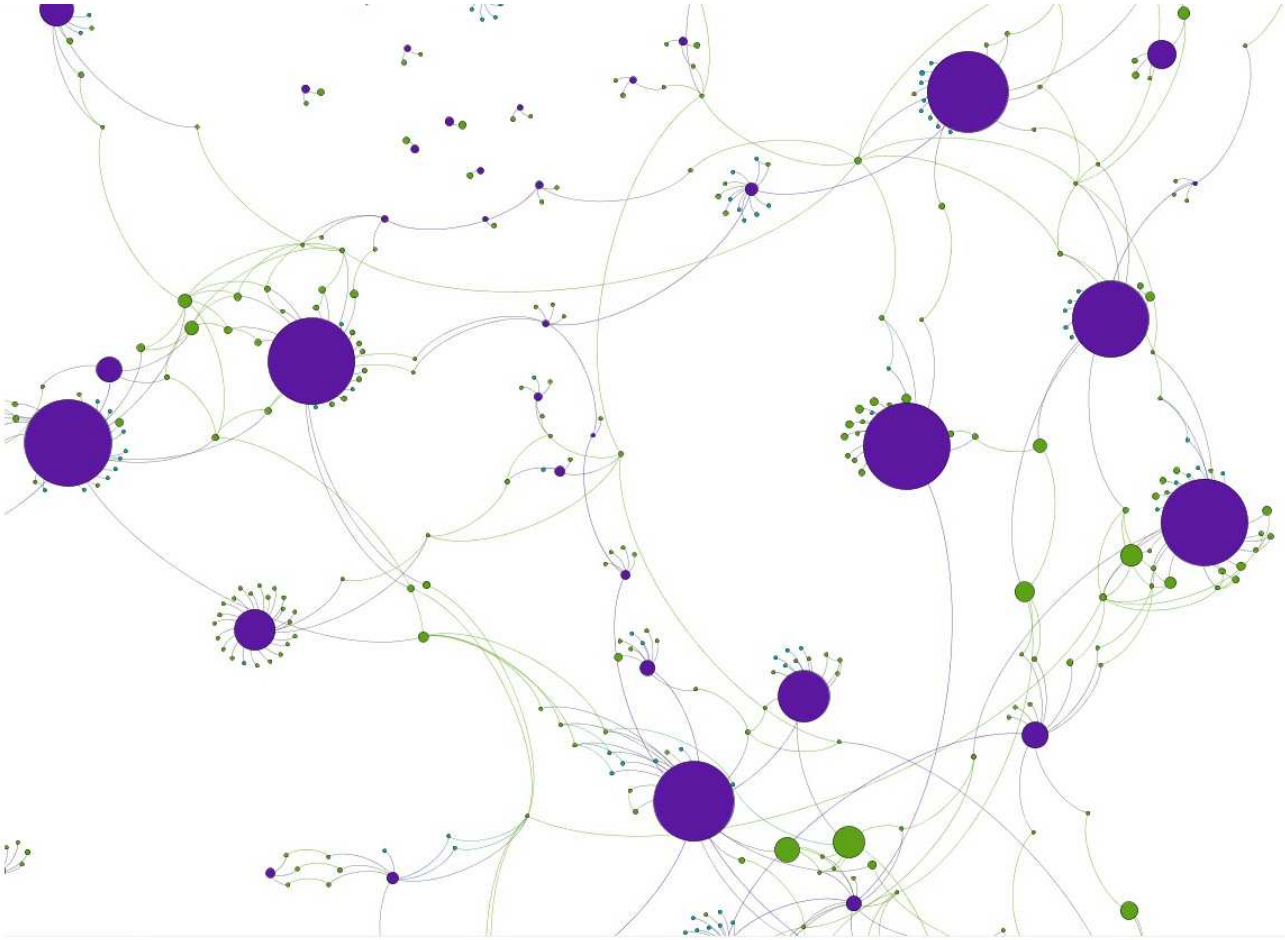
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΣΤΡΑΤΗΓΙΚΗ ΕΠΙΛΟΓΗΣ ΚΛΑΣΕΩΝ ΜΕ ΣΤΟΧΟ ΤΗΝ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΤΗΣ ΠΟΙΟΤΗΤΑΣ ΤΟΥ ΕΠΙΛΕΓΜΕΝΟΥ ΚΩΔΙΚΑ»



Του φοιτητή

Γκορτζή Αντωνίου

Αρ. Μητρώου: 042648

Επιβλέποντες καθηγητές

Δρ. Δεληγιάννης Ιγνάτιος

Δρ. Αμπατζόγλου Απόστολος

Θεσσαλονίκη 2012

Πρόλογος

Η ανάπτυξη λογισμικού αποτελεί μια σύνθετη και πολύπλοκη διαδικασία. Η ανάπτυξη λογισμικού βασισμένη σε «υποψήφια συστατικά» προσφέρει μία ελπιδοφόρα λύση στην παράγωγή μεγάλων και σύνθετων συστημάτων λογισμικού αρχικά χωρίζοντας πολύπλοκες διαδικασίες σε μικρότερες και στην συνέχεια την υλοποίηση αυτών με επαναχρησιμοποίηση έτοιμων «συστατικών λογισμικού». Στη συγκεκριμένη εργασία προτείνεται και χρησιμοποιείται μια μεγάλη πηγή «υποψήφιων συστατικών» που εξάγονται από το ανοιχτό λογισμικό, λόγω της δυνατότητας χρήσης τους σε επίπεδο επαναχρησιμοποίησης «λευκού» και «μαύρου» κουτιού.

Μεγάλη σημασία για την επιλογή συστατικών λογισμικού κατέχει η μέτρηση των ποιοτικών χαρακτηριστικών που επιδεικνύει ένα «υποψήφιο συστατικό», διαδικασία που μπορεί να συντελέσει σημαντικά στη βελτίωση της ποιότητας του λογισμικού στο οποίο θα επαναχρησιμοποιηθεί, όσο και στην ευκολία της διαδικασίας επαναχρησιμοποίησης του. Η μέθοδος που χρησιμοποιείται για το σκοπό αυτό είναι οι μετρικές.

Στην βελτίωση της ποιότητας του λογισμικού, αποσκοπεί και η εύρεση «υποψήφιων συστατικών» βασισμένα στη χρήση των προτύπων σχεδίασης. Τα πρότυπα σχεδίασης, είναι μηχανισμοί που παρέχουν λύσεις σε συνήθη σχεδιαστικά προβλήματα, που παρουσιάζονται κατά τις φάσεις της σχεδίασης, της ανάπτυξης ή της τροποποίησης του λογισμικού. Τα πρότυπα, βελτιώνουν τα ποιοτικά χαρακτηριστικά του συστήματος, παρέχοντας ευελιξία, προσαρμοστικότητα, ευκολία στη συντήρηση, δυνατότητα επαναχρησιμοποίησης συστατικών του συστήματος κ.α.

Η διεξαγωγή της παρούσας μελέτης και η ανάπτυξη του εργαλείου «εξαγωγής υποψήφιων συστατικών» αποδείχθηκε ιδιαίτερα χρήσιμη για την εξοικείωση του συγγραφέα με προηγμένα θέματα τεχνολογίας λογισμικού όπως ο σχεδιασμός, η υλοποίηση, η επέκταση, βελτιστοποίηση του κώδικα και η συντήρηση μιας ολοκληρωμένης εφαρμογής λογισμικού. Επίσης προσέφερε σημαντική εμπειρία στις

μεθόδους εμπειρικής αξιολόγησης μιας μελέτης και τη συστηματική βιβλιογραφική ανασκόπηση ενός ερευνητικού πεδίου.

Τέλος τα ευρήματα της εργασίας χρησιμοποιήθηκαν ως μέρος μιας υπο κρίση εργασίας, υποβεβλημένη σε ένα επιστημονικό συνέδριο στο χώρο της τεχνολογίας λογισμικού για παιχνίδια.

Περίληψη

Η Μηχανική Λογισμικού βασισμένη σε «Συστατικά Λογισμικού» εστιάζει στην παραγωγή «υποψήφιων συστατικών» με σκοπό την αξιοποίησή τους από περισσότερα υπό ανάπτυξη συστήματα και όχι μόνο από αυτό για το οποίο αρχικά σχεδιάστηκαν καθώς και στην ανάπτυξη εφαρμογών εξ ολοκλήρου από επαναχρησιμοποιήσιμα συστατικά. Η συγκεκριμένη πτυχιακή εργασία αποσκοπεί στην παρουσίαση μίας μεθοδολογίας για την εξαγωγή επαναχρησιμοποιήσιμων «συστατικών λογισμικού» από εφαρμογές ανοιχτού λογισμικού. Τα παραγόμενα «υποψήφια συστατικά» αξιολογήθηκαν με την διεξαγωγή μιας εμπειρικής μελέτης. Επιπρόσθετα, τα «υποψήφια συστατικά» που εξήχθησαν είναι διαθέσιμα για ελεύθερη επαναχρησιμοποίηση μέσω μιας διαδικτυακής υπηρεσίας. Για τις ανάγκες της πτυχιακής εργασίας σχεδιάστηκε και αναπτύχθηκε ένα εργαλείο που εφαρμόζει την προτεινόμενη μεθοδολογία και παράγει «υποψήφια συστατικά» προς επαναχρησιμοποίηση.

Abstract

Component-Based Software Engineering (CBSE) focuses on the development of components in order to enable their reuse in more systems, rather than only to be used to the original ones for which they have been implemented in the first place (i.e. development for reuse) and the development of new systems with reusable components (i.e. development with reuse). This thesis aims at introducing a methodology on extracting reusable software components from open source software. The extracted components have been empirically evaluated through a case study. Additionally, the components that have been extracted are available for reuse through a web service that is freely available in the web. For the purposes of the thesis a tool applying the proposed methodology and extracting reusable components has been developed.

Ευχαριστίες

Θεωρώ την εκπόνηση της συγκεκριμένης πτυχιακής εργασίας ως την ανώτερη δημιουργική στιγμή της φοιτητικής μου πορείας. Θα ήθελα λοιπόν να ευχαριστήσω θερμά τους επιβλέποντες καθηγητές μου κ. Δεληγιάνη Ιγνάτιο και κ. Αμπατζόγλου Απόστολο που μου προσέφεραν αυτή τη δυνατότητα. Ιδιαίτερα θα ήθελα να ευχαριστήσω τον Απόστολο που για περισσότερο από ένα χρόνο στάθηκε σαν δάσκαλος, συνεργάτης και φίλος σε αυτή την αντικειμενικά δύσκολη πορεία και υπό την καθοδήγησή του έγινε πραγματικότητα η εφαρμογή που παρουσιάζουμε στην πτυχιακή εργασία. Επίσης θα ήθελα να τον ευχαριστήσω για τις ευκαιρίες που μου έδωσε να ασχοληθώ με ερευνητικά θέματα και να συμμετάσχω στην διαδικασία συγγραφής ενός επιστημονικού άρθρου και την δημοσίευση αυτού σε διεθνές συνέδριο του χώρου. Η συνεργασία μαζί του μου προσέφερε ανεκτίμητης αξίας εμπειρίες και γνώσεις.

Επίσης θα ήθελα να ευχαριστήσω την οικογένειά και τους φίλους μου, που έδειξαν κατανόηση στις δυσκολίες των τελευταίων εξαμήνων και με στήριξαν με κάθε τρόπο.

Τέλος, θα ήθελα να ευχαριστήσω την ομάδα του εργαστηρίου SWENG του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης για τις συμβουλές και λύσεις που μου πρότειναν σε πολλά από τα αδιέξοδα που παρουσιάστηκαν κατά την διάρκεια της εκπόνησης της πτυχιακής εργασίας και της παρουσίας μου στο εργαστήριό τους στα πλαίσια της εξάμηνης Πρακτικής μου άσκησης.

Περιεχόμενα

Ευρετήριο Σχημάτων

Ευρετήριο Πινάκων

1. Εισαγωγή

Στο πρώτο κεφάλαιο κρίνεται απαραίτητο να γίνει μία βιβλιογραφική ανασκόπηση στο «ανοιχτό λογισμικό» από το οποίο αντλούνται όλα τα δεδομένα προς ανάλυση και σύγκριση στη συγκεκριμένη πτυχιακή εργασία. Επίσης γίνεται μία βιβλιογραφική αναφορά στη διαδικασία της «ανάπτυξης λογισμικού βασισμένη σε συστατικά». Στη συνέχεια γίνεται μια σύντομη ανάλυση για κάθε ένα από τα δεκαοκτώ (18) πρότυπα σχεδίασης που μελετώνται και ανακτώνται από το εργαλείο που αναπτύχθηκε στα πλαίσια της Πτυχιακής εργασίας.

1.1. Ανοιχτό Λογισμικό

Ο όρος "ανοιχτό λογισμικό" (free software) εισήχθη από τον Richard Stallman στις αρχές της δεκαετίας του ΄80 ενώ παράλληλα ίδρυσε και το Ίδρυμα Ελεύθερου Λογισμικού (free Software Foundation) το 1983 προκειμένου να βρεθεί ένας τρόπος διατήρησης της ελευθερίας των χρηστών να μελετούν, να κατανοούν και να τροποποιούν το λογισμικό μέσα στα πλαίσια των κανόνων και του πνεύματος που διέπει τις ομάδες αυτές. Η δημιουργία του Ιδρύματος Ελεύθερου Λογισμικού και η προσπάθεια για την διάδοση αυτού συμπίπτει χρονικά με την περίοδο της αποσύνδεσης του υλικού από το λογισμικό και της προσπάθειας για εμπορευματοποίηση του δεύτερου.

Το 1998 παρουσιάστηκε μια νέα τάση στην παραγωγή λογισμικού, η διαδικασία ανάπτυξης κώδικα ανοιχτού λογισμικού (Open Source Software), την οποία μελέτησαν οι συγγραφείς στο (Feller and Fitzgerald, 2002), παρουσιάζοντας πολύ αξιόλογα και πετυχημένα έργα ανοιχτού λογισμικού όπως το λειτουργικό Linux, το πακέτο εργαλείων γραφείου Libre Office, ο φυλλομετρητής Mozilla Firefox καθώς και ο εξυπηρετητής Apache Server.

Η ουσία στην ιδέα του ανοιχτού λογισμικού βρίσκεται στην διάθεση του πηγαίου κώδικα καθώς και στην συνεργασία των μελών της κοινότητας για την περαιτέρω ανάπτυξή του. Αυτός ο τρόπος ανάπτυξης έχει πλεονεκτήματα αλλά και μειονεκτήματα. Το κόστος ανάπτυξης παραμένει σε πολύ χαμηλά επίπεδα αλλά το τελικό προϊόν υστερεί σε τεκμηρίωση και τεχνική υποστήριξη. Από την άλλη πλευρά όμως

χαρακτηριστικό του ανοιχτού λογισμικού είναι η αξιοπιστία καθώς και η διάθεση του πηγαίου κώδικα σε οποιοδήποτε θέλει να το παραμετροποιήσει και να το προσαρμόσει στις δικές του ανάγκες όπως αναφέρεται στο (Samoladas et al., 2004). Η ευρέως διαδεδομένη χρήση της άδειας ελεύθερου λογισμικού, GNU General Public License, εγγυάται ότι ο καθένας μπορεί να τροποποιήσει και να αναδιανείμει προγράμματα, έχοντας ως προϋπόθεση ότι δεν εμποδίζει άλλους από το να πράξουν το ίδιο.

1.2. Ανάπτυξη Λογισμικού Βασισμένη σε Συστατικά

Για να κατανοήσουμε την βασισμένη σε συστατικά ανάπτυξη λογισμικού πρέπει πρώτα να κατανοήσουμε τον όρο "Συστατικά Λογισμικού". Σύμφωνα με τον Milton Keyes στο (Keyes M, 1999) τα συστατικά λογισμικού πρωτοεμφανίστηκαν μαζί με την χρήση των υπορουτίνων στις πρώτες γλώσσες μηχανής. Ήταν κομμάτια κώδικα τα οποία μπορούσαν να συνενωθούν σε ένα μεγαλύτερο πακέτο και να εκτελεστούν παρέχοντας κάποια λειτουργικότητα. Μπορούσαν να τοποθετηθούν μέσα σε βιβλιοθήκες λογισμικού καθώς και να συνδεθούν απευθείας πάνω σε υπάρχοντα κώδικα. Το 1968 ο McLllroy στο (McIllroy M.D, 1967) χαρακτήρισε αυτά τα πακέτα ως "συστατικά λογισμικού".

Η ανάπτυξη λογισμικού βασισμένη σε συστατικά έχει γίνει τα τελευταία χρόνια μια ευρέως διαδεδομένη μέθοδος ανάπτυξης λογισμικού. Σύμφωνα με τη μέθοδο αυτή αντί να ξεκινάμε την συγγραφή του κώδικα μιας εφαρμογής από το μηδέν στην ουσία αναζητούμε ανεξάρτητα κομμάτια κώδικα σε ήδη υπάρχουσες εφαρμογές τα οποία καλύπτουν τις λειτουργικές μας απαιτήσεις και μπορούμε να προσαρμόσουμε στην εφαρμογή μας όπως αναφέρεται και στο (Staringer W, 1994). Σε αυτό έχουν συντελέσει και τα πολλά αποθετήρια συστατικών (repositories) που μπορεί ο κάθε χρήστης να επισκεφτεί και να αναζητήσει συστατικά με τις επιθυμητές λειτουργίες.

Ο συγκεκριμένη μέθοδος ανάπτυξης λογισμικού βασίζεται στην "Επαναχρησιμοποίηση Λογισμικού" και περιλαμβάνει ένα συνδυασμό από τεχνικές, μεθόδους και διαδικασίες. Ενώ η επαναχρησιμοποίηση λογισμικού φαίνεται ελκυστική, πολλές τεχνικά και μη, προβλήματα κάνουν την χρήση της όχι και τόσο εύκολη. Για παράδειγμα:

1. Όταν οι προδιαγραφές του συστήματος δεν υπάρχουν ή δεν είναι αρκετά συγκεκριμένες ώστε να κατανοήσομε τις παρεχόμενες λειτουργίες χωρίς να χρειάζεται να ανατρέξουμε στον κώδικα.
2. Το κόστος για να προσαρμόσουμε το ήδη υπάρχον σύστημα στο δικό μας είναι μεγαλύτερο από το κόστος της υλοποίησής του από την αρχή.
3. Όταν υπάρχει λογισμικό που να εκτελεί την λειτουργία που θέλουμε αλλά είναι πολύ γενικό ώστε στην περίπτωσή μας να είναι ασύμφορη η χρήση του.
4. Τέλος δεν υπάρχει ενιαίος τρόπος διαχείρισης των δομών δεδομένων με αποτέλεσμα να περιορίζεται η επικοινωνία μεταξύ των συστατικών ενός συστήματος.

Βάση των παραπάνω γίνεται κατανοητό πως πρέπει να δοθεί ιδιαίτερη βαρύτητα στον σχεδιασμό επαναχρησιμοποιήσιμων συστατικών λογισμικού καθώς και στην οργάνωση και αποθήκευσή τους με τρόπο που να διευκολύνει την εύρεση του κατάλληλου συστατικού για κάθε λειτουργία που ζητάμε. Ο (McLllroy, 1974) έθεσε το εξής ερώτημα. "Γιατί το λογισμικό (software) δεν μοιάζει περισσότερο στο υλικό (hardware)? Γιατί κάθε υλοποίηση πρέπει να ξεκινάει από το μηδέν?", προτείνοντας την λύση με καταλόγους οι οποίοι θα περιέχουν συστατικά λογισμικού διαθέσιμα για ελεύθερη χρήση ή και αγορά.

1.2.1. Επαναχρησιμοποίηση Λογισμικού

Η επαναχρησιμοποίηση κώδικα ανοιχτού λογισμικού είναι μια ολοένα και πιο συχνά εμφανιζόμενη τεχνική από τις εταιρίες λογισμικού. Η επαναχρησιμοποίηση συστατικών ανοιχτού λογισμικού σε άλλα έργα ανοιχτού λογισμικού, είναι εξαιρετικά έντονη· η επαναχρησιμοποίηση κώδικα από 1.311 έργα αντιπροσωπεύει περίπου 316.000 εργατοέτη και δεκάδες εκατομμύρια δολάρια σε κόστος ανάπτυξης[1]. Επίσης, συστατικά ανοιχτού λογισμικού επαναχρησιμοποιούνται σε χιλιάδες άλλα έργα, για παράδειγμα το log4j έχει επαναχρησιμοποιηθεί σε περισσότερα από 5.500 έργα[2].

Στη βιβλιογραφία, η επαναχρησιμοποίηση λογισμικού εμφανίζεται με δυο βασικές μορφές, τη συστηματική (Jansen et. al, 2008) και την οπορτουνιστική (Morison et. al,

[1] http://www.blackducksoftware.com/news/releases/2009-03-30
[2] http://www.blackducksoftware.com/news/releases/2008-12-09

2000). Τα αποτελέσματα σχετικά με το ποιος από τους δύο τρόπους είναι πιο αποδοτικός είναι διχασμένα, ανάλογα με το μέγεθος της επιχείρησης [Henry and Faller, 1995, Jansen et. al, 2008 και McConnell, 1996]. Στην πραγματικότητα, όταν ένας δημιουργός λογισμικού επαναχρησιμοποιεί κώδικα οπορτουνιστικά, επικολλά κλάσεις ή πακέτα στο έργο που αναπτύσσει. Επιπλέον, στο (Mockus, 2007) αναφέρεται ότι οι περισσότερες μονάδες επαναχρησιμοποίησης έχουν τροποποιηθεί σημαντικά ή λιγότερο σημαντικά, ώστε να ενσωματωθούν στο τελικό έργο. Το γεγονός ότι οι οντότητες που επαναχρησιμοποιούνται τροποποιούνται πριν τη χρήση δείχνει ότι έχουν χρησιμοποιηθεί τεχνικές επαναχρησιμοποίησης λευκού κουτιού (white-box reuse).

Σε αυτή την ενότητα μελετήθηκε η επαναχρησιμοποίηση τμημάτων κώδικα, στο πλαίσιο της οπορτουνιστικής επαναχρησιμοποίησης λογισμικού, με στόχο να εντοπιστεί το σύνολο κλάσεων το οποίο αποτελεί το βέλτιστο σημείο εκκίνησης για επαναχρησιμοποίηση. Ο όρος βέλτιστο, ορίζεται ως το σύνολο κλάσεων με δομικά χαρακτηριστικά τα οποία ευνοούν την κατανοησιμότητα και την προσαρμοστικότητα τους. Σε αρκετές πραγματικές καταστάσεις, η προσπάθεια αναγνώρισης ενός τμήματος κώδικα προς επαναχρησιμοποίηση κατευθύνει το σχεδιαστή σε μια συγκεκριμένη κλάση. Στην περίπτωση που η συγκεκριμένη κλάση συμμετέχει σε κάποιο πρότυπο σχεδίασης, ο σχεδιαστής έχει τρεις κύριες εναλλακτικές ως προς την επιλογή του συνόλου κλάσεων που θα επαναχρησιμοποιήσει, (α) να επαναχρησιμοποιήσει την κλάση, (β) να επαναχρησιμοποιήσει το πρότυπο και (γ) να επαναχρησιμοποιήσει το πακέτο στο οποίο ανήκει η κλάση. Αν και συνήθως η λειτουργικότητα είναι ο κύριος παράγοντας για την επαναχρησιμοποίηση σε πρώτο επίπεδο, η επιλογή σύμφωνα με συγκεκριμένα χαρακτηριστικά ποιότητας, όπως η επαναχρησιμοποιησιμότητα, είναι ο βασικός παράγοντας για την επιλογή μεταξύ ισοδύναμων λειτουργικά τμημάτων κώδικα.

1.3. Πρότυπα Σχεδίασης

Ο όρος "πρότυπα σχεδίασης" παρουσιάστηκε πρώτη φορά από τον αρχιτέκτονα Cristopher Alexander το 1977. Ο Alexander παρατήρησε ότι υπάρχουν συγκεκριμένα αρχιτεκτονικά σχεδιαστικά προβλήματα που μπορούν να διαχειριστούν με κοινές λύσεις. Έτσι κατέγραψε αυτά τα ζευγάρια προβλημάτων και λύσεων προτείνοντας την

επαναχρησιμοποίηση τους για την επίτευξη καλών ποιοτικά σχεδίων (Alexander et al., 1977).

Στην τεχνολογία λογισμικού, ένα σχεδιαστικό πρότυπο είναι μία γενική επαναλαμβανόμενη λύση για ένα εμφανιζόμενο πρόβλημα σε κάποια σχεδίαση λογισμικού (software design). Ένα σχεδιαστικό πρότυπο δεν είναι κάποια ολοκληρωμένη σχεδίαση η οποία μπορεί να μετασχηματιστεί άμεσα σε έναν κώδικα. Αποτελεί μία περιγραφή ή ένα πρότυπο για το πώς μπορεί να λυθεί ένα πρόβλημα, το οποίο μπορεί να χρησιμοποιηθεί σε πολλές διαφορετικές καταστάσεις. Πρόκειται για αφαιρέσεις υψηλού επιπέδου που αποτελούν πλήρη υποσυστήματα, κατάλληλα ρυθμισμένα για την επίλυση συγκεκριμένων προβλημάτων και έτοιμα για χρήση. Είναι μια περιγραφή για το πώς να λυθεί ένα πρόβλημα που μπορεί να χρησιμοποιηθεί σε πολλές διαφορετικές καταστάσεις. Τα αντικειμενοστραφή σχεδιαστικά πρότυπα παρουσιάζουν τις σχέσεις και τις αλληλεπιδράσεις μεταξύ των κλάσεων ή των αντικειμένων χωρίς διευκρίνιση των τελικών κλάσεων ή των αντικειμένων εφαρμογής που περιλαμβάνονται. Κάθε πρότυπο σχεδίασης κατονομάζει τη συγκεκριμένη λύση, παρέχει μια περιγραφή του προβλήματος στο οποίο μπορεί να εφαρμοστεί, και προδιαγράφει τη λύση, συνήθως σε επίπεδο αρχιτεκτονικής σχεδίασης. (Chatzigeorgiou, 2005).

Έχουν ορισθεί τρεις διαφορετικές κατηγορίες προτύπων για την αντιμετώπιση διαφορετικών τύπων προβλημάτων. Πιο συγκεκριμένα τα:

- Κατασκευαστικά Πρότυπα (Creational Patterns) τα οποία παρουσιάζουν τρόπους δυναμικής κατασκευής αντικειμένων με στόχο την ανεξαρτητοποίηση του κώδικα που χρησιμοποιεί κάποια αντικείμενα από τις κλάσεις που ορίζουν τα αντικείμενα αυτά.
- Δομικά Πρότυπα (Structural Patterns) τα οποία επίσης παρουσιάζουν τρόπους δυναμικής κατασκευής σύνθετων αντικειμένων τα οποία χρησιμοποιούν υπάρχουσες ιεραρχίες κλάσεων. Έχουν στόχο να μειώσουν την σύζευξη μεταξύ των κλάσεων του συστήματος και να προσφέρουν εναλλακτικές λύσεις στην κληρονομικότητα.

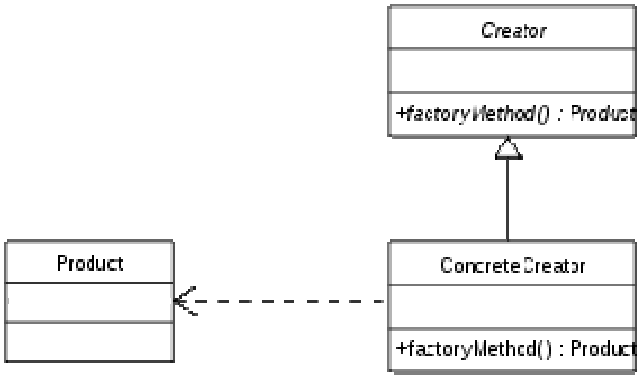
- Συμπεριφορικά Πρότυπα (Behavioral Patterns) τα οποία ορίζουν τον καταμερισμό αρμοδιοτήτων σε διάφορες κλάσεις και τον τρόπο επικοινωνίας μεταξύ των αντικειμένων τους κατά τον χρόνο εκτέλεσης. (Gamma et al., 1995).

Παρακάτω αναλύονται κάποια βασικά πρότυπα τα οποία χρησιμοποιούνται για την δημιουργία "Υποψήφιων Συστατικών".

1.3.1. Μέθοδος Εργοστάσιο (Factory Method)

Το πρότυπο σχεδίασης "Μέθοδος Εργοστάσιο" συγκεντρώνει σε μια κατάλληλη κλάση όλη τη λειτουργικότητα κατασκευής στιγμιότυπων ενός συνόλου παραγόμενων υποκλάσεων που κληρονομούν κάποια κοινή υπερκλάση ή υλοποιούν την ίδια διασύνδεση. Ανήκει στην κατηγορία των κατασκευαστικών προτύπων.

Χρησιμοποιείται όταν μία κλάση δεν γνωρίζει τον ακριβή τύπο των αντικειμένων που πρέπει να δημιουργήσει όπως επίσης όταν μία κλάση θέλει οι παραγόμενές της κλάσεις να ορίζουν τον τύπο των αντικειμένων που θα δημιουργούνται. (Gamma et al., 1995).



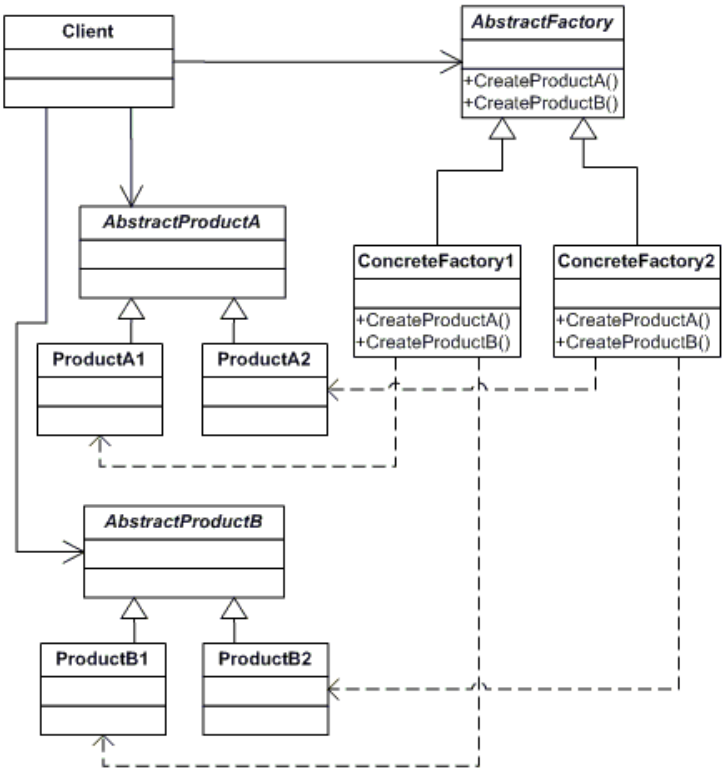
Σχήμα 1. Διάγραμμα κλάσεων προτύπου Factory

1.3.2. Αφηρημένο εργοστάσιο (Abstract Factory)

Σκοπός του προτύπου σχεδίασης "Αφηρημένο Εργοστάσιο" είναι η παροχή μιας διασύνδεσης για τη δημιουργία οικογενειών, συσχετιζόμενων ή εξαρτημένων αντικειμένων, χωρίς να προσδιορίζεται η υλοποίηση των συγκεκριμένων κλάσεών τους.

Ανήκει στην κατηγορία των κατασκευαστικών προτύπων και ως εκ τούτου επιτρέπει τη συγγραφή μεθόδων που δημιουργούν νέα αντικείμενα, χωρίς την άμεση χρήση ιδιωμάτων (π.χ. τελεστής new), όπως συμβαίνει στις αντικειμενοστραφείς γλώσσες προγραμματισμού. Το γεγονός αυτό επιτρέπει την ανάπτυξη μεθόδων που παράγουν ομάδες διαφορετικών αντικειμένων καθώς και την επέκτασή τους για νέα αντικείμενα χωρίς την τροποποίηση του κώδικα των μεθόδων.

Το πρότυπο Factory χρησιμοποιείται για την αποφυγή εξάρτησης από συγκεκριμένες κλάσεις όταν απαιτείται η δημιουργία αντικειμένων καθώς και για την ομαδοποίηση μεθόδων που δημιουργούν συσχετιζόμενα αντικείμενα σε μια αφηρημένη κλάση. (Chatzigeorgiou, 2005).

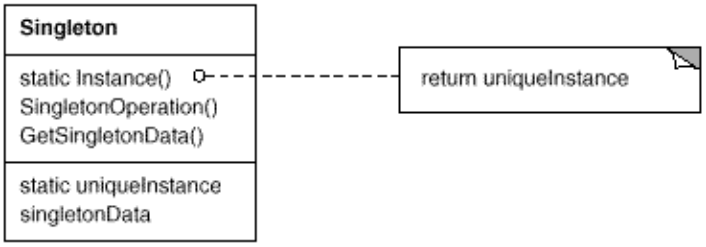


Σχήμα 2. Διάγραμμα κλάσεων προτύπου Abstract Factory

1.3.3. Μοναδιαίο (Singleton)

Το πρότυπο σχεδίασης "Μοναδιαίο" εξασφαλίζει ότι μια κλάση θα έχει μόνο ένα στιγμιότυπο και παρέχει ένα καθολικό σημείο πρόσβασης σε αυτό. Ανήκει στην κατηγορία των κατασκευαστικών προτύπων.

Χρησιμοποιείται όταν σε κάποιο σύστημα λογισμικού υπάρχει η απαίτηση από μια κλάση να δημιουργείται ένα και μόνο αντικείμενο. (Chatzigeorgiou, 2005).

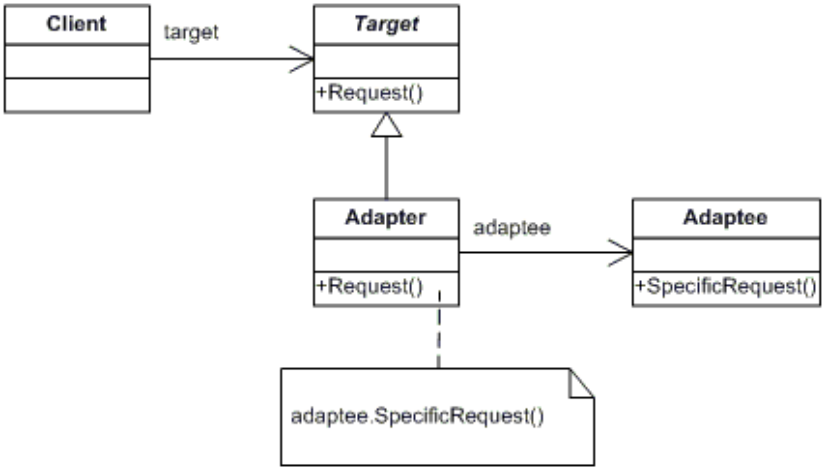


Σχήμα 3. Διάγραμμα κλάσεων προτύπου Singleton

1.3.4. Προσαρμογέας (Adapter)

Το πρότυπο σχεδίασης "Προσαρμογέας" έχει ως στόχο την μετατροπή της διασύνδεσης μιας κλάσης σε μια άλλη που αναμένει το πρόγραμμα πελάτης. Ο προσαρμογέας επιτρέπει τη συνεργασία κλάσεων, η οποία σε διαφορετική περίπτωση θα ήταν αδύνατη λόγω ασύμβατων διασυνδέσεων. Ανήκει στην κατηγορία των δομικών προτύπων (structural).

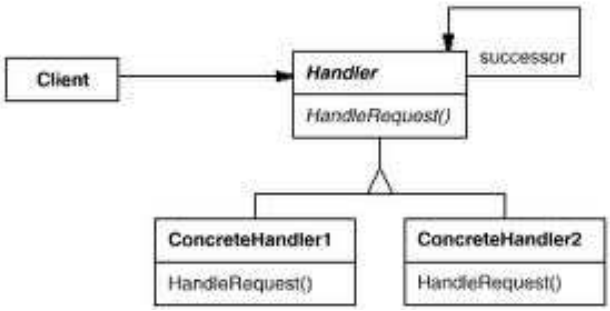
Χρησιμοποιείται σε περιπτώσεις που η διασύνδεση μιας κλάσης που μας ενδιαφέρει δεν συμβαδίζει με τις ανάγκες μας. Ένας προσαρμογέας κλάσης (class adapter) χρησιμοποιεί πολλαπλή κληρονομικότητα για να προσαρμόσει μια διασύνδεση σε μια άλλη. Ενώ ένας προσαρμογέας αντικειμένου (object adapter) βασίζεται στη σύνθεση αντικειμένων και στη διαβίβαση μηνυμάτων. (Chatzigeorgiou, 2005).



Σχήμα 4. Διάγραμμα κλάσεων προτύπου Adapter

1.3.5. Αλυσίδα Ευθύνης (Chain of Responsibility)

Το πρότυπο σχεδίασης "Αλυσίδα Ευθύνης" προσφέρει την δυνατότητα να αποφύγουμε την άμεση σύζευξη μεταξύ του αποστολέα μιας αίτησης και του παραλήπτη της, ορίζοντας περισσότερα του ενός αντικείμενα που μπορούν να διαχειριστούν την αίτηση. Δημιουργεί δηλαδή μια αλυσίδα μεταξύ των εμπλεκόμενων αντικειμένων και προωθεί μία αίτηση έως ότου ο εξουσιοδοτημένος παραλήπτης να την επεξεργαστεί. Ανήκει στην κατηγορία των συμπεριφορικών προτύπων. (Gamma et al., 1995).

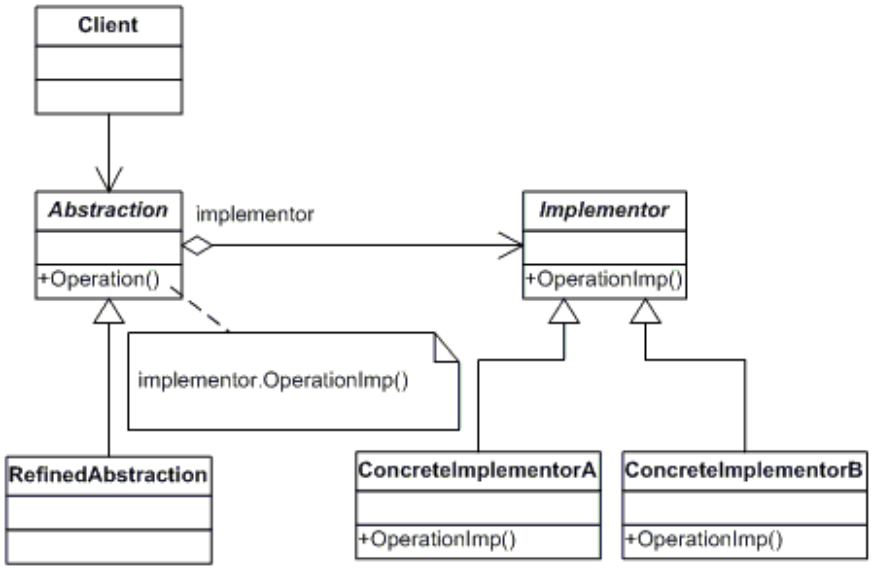


Σχήμα 5. Διάγραμμα κλάσεων προτύπου Chain of Responsibility

1.3.6. Γέφυρα (Bridge)

Το πρότυπο σχεδίασης "Γέφυρα" έχει ως στόχο την αποσύνδεση μιας αφαίρεσης από την υλοποίησή της, ώστε να μπορούν να μεταβάλλονται ανεξάρτητα. Όταν μια αφαίρεση μπορεί να έχει περισσότερες από μία υλοποιήσεις, ο συνήθης τρόπος οργάνωσης είναι με τη χρήση κληρονομικότητας. Με τον όρο αφαίρεση νοείται μια αφηρημένη κλάση που ορίζει μια διασύνδεση, ενώ υλοποιήσεις είναι οι συγκεκριμένες παράγωγες κλάσεις οι οποίες υλοποιούν τις μεθόδους της αφηρημένης κλάσης. Ανήκει στην κατηγορία των δομικών προτύπων (structural).

Μια "Γέφυρα" χρησιμοποιείται όταν μια έννοια (αφαίρεση) μπορεί να σχετίζεται με διάφορες υλοποιήσεις οι οποίες μπορεί να τροποποιούνται κατά τη διάρκεια της εκτέλεσης. Επίσης χρησιμοποιείται όταν οι αφαιρέσεις και οι υλοποιήσεις τους θα πρέπει να είναι επεκτάσιμες μέσω κληρονομικότητας. (Chatzigeorgiou, 2005).



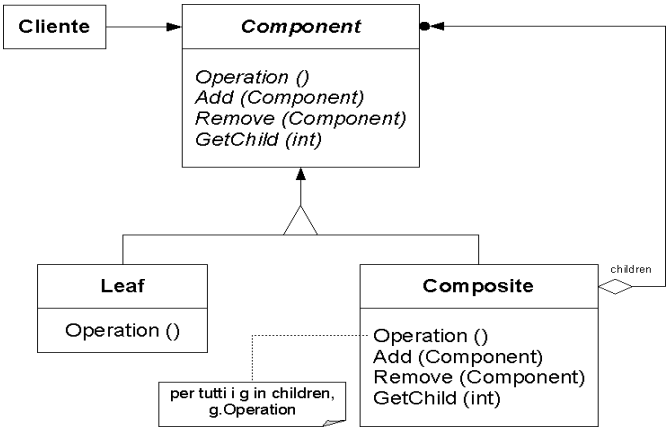
Σχήμα 6. Διάγραμμα κλάσεων προτύπου Bridge

1.3.7. Σύνθετο (Composite)

Το πρότυπο σχεδίασης "Σύνθετο" επιτρέπει τη σύνθεση αντικειμένων σε δενδροειδείς δομές για την αναπαράσταση ιεραρχιών τμήματος-όλου. Το πρότυπο σχεδίασης "Σύνθετο" επιτρέπει στα προγράμματα πελάτες να διαχειρίζονται με ενιαίο

τρόπο τόσο τα ανεξάρτητα αντικείμενα όσο και τις συνθέσεις αντικειμένων. Ανήκει στην κατηγορία των δομικών προτύπων (structural).

Χρησιμοποιείται όταν θέλουμε τα προγράμματα πελάτες να χειρίζονται μεμονωμένα αντικείμενα και σύνθετα αντικείμενα με τον ίδιο τρόπο. (Chatzigeorgiou, 2005).



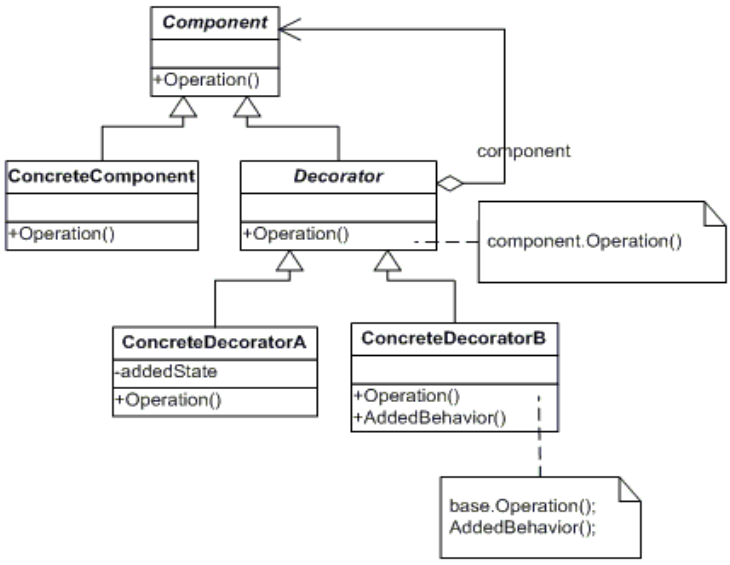
Σχήμα 7. Διάγραμμα κλάσεων προτύπου Composite

1.3.8. Διακοσμητής (Decorator)

Το πρότυπο "διακοσμητής" προσθέτει λειτουργίες σε κάποιο αντικείμενο δυναμικά. Οι "διακοσμητές" προσφέρουν μια ευέλικτη εναλλακτική λύση για την επέκταση των λειτουργιών μιας κλάσης αντί της μεθόδου δημιουργίας υποκλάσεων. Ανήκει στην κατηγορία των δομικών προτύπων (structural).

Εάν για παράδειγμα θέλουμε να εφαρμόσουμε μια συγκεκριμένη λειτουργία σε ένα αντικείμενό μας δεν χρειάζεται να πειράξουμε ολόκληρη την κλάση "διασύνδεσή" του αλλά αρκεί να επισυνάψουμε το αντικείμενό μας σε ένα άλλο "αντικείμενο διακοσμητής" που θα παρέχει την επιθυμητή λειτουργία. (Gamma et al., 1995).

Επίσης προσφέρει την δυνατότητα προσθήκης λειτουργιών σε κλάσης και που αποκρύπτονται οι λειτουργίες τους και δεν επιτρέπεται η επέκτασή τους. (Gamma et al., 1995).

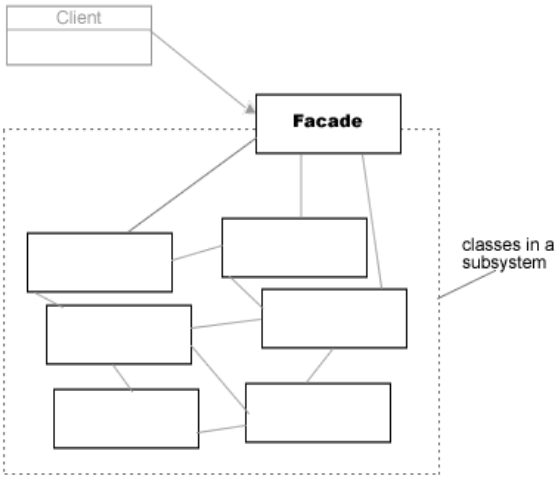


Σχήμα 8. Διάγραμμα κλάσεων προτύπου Decorator

1.3.9. Πρόσοψη (Facade)

Το πρότυπο σχεδίασης "Πρόσοψη" παρέχει μία ενιαία διασύνδεση για ένα σύνολο από διασυνδέσεις ενός υποσυστήματος. Προσδιορίζει δηλαδή μία υψηλού-επιπέδου διασύνδεση που διευκολύνει την χρήση ενός υποσυστήματος. Ανήκει στην κατηγορία των δομικών προτύπων.

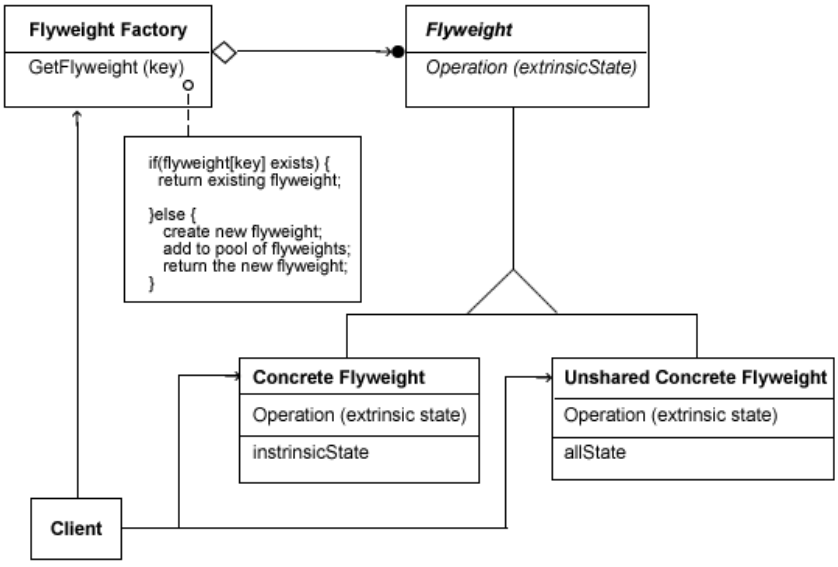
Χρησιμοποιείται όταν έχουμε μεγάλη σύζευξη μεταξύ των πελατών και των υλοποιήσεων κάποιων διασυνδέσεων με σκοπό να μειώσει την σύζευξη αυτή αλλά και να προσφέρει και μία ανεξαρτητοποίηση στο υποσύστημα. (Gamma et al., 1995).



Σχήμα 9. Διάγραμμα κλάσεων προτύπου Πρόσοψη

1.3.10. Ελαφρού Τύπου (Flyweight)

Το πρότυπο σχεδίασης "Flyweight" προτείνει την κοινή χρήση κάποιων αντικειμένων ώστε να είναι αποδοτικές και διαχειρίσιμες καταστάσεις στις οποίες υπάρχει πολύ μεγάλος αριθμός αντικειμένων. Ένα flyweight δηλαδή, είναι ένα αντικείμενο το οποίο μπορεί να χρησιμοποιηθεί ταυτόχρονα από πολλές συλλογές αντικειμένων. Ανήκει στην κατηγορία των δομικών προτύπων. (Gamma et al., 1995).

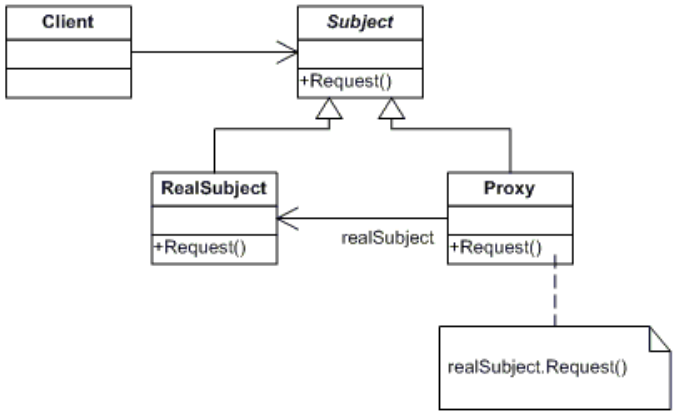


Σχήμα 10. Διάγραμμα κλάσεων προτύπου Flyweight

1.3.11. Πληρεξούσιο (Proxy)

Το πρότυπο σχεδίασης "Πληρεξούσιο" λειτουργεί ως υποκατάστατο ή ως ένας τρόπος κράτησης της θέσης ενός άλλου αντικειμένου. Ανήκει στην κατηγορία των δομικών προτύπων.

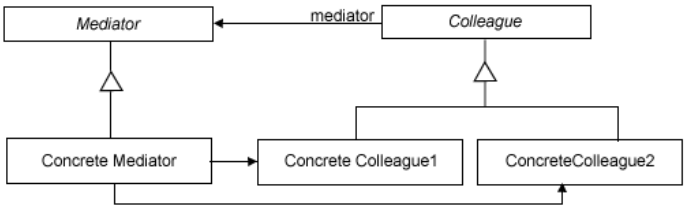
Μπορεί να χρησιμοποιηθεί με πολλούς τρόπους, είτε λειτουργώντας τοπικά ως αντιπρόσωπος ενός αντικειμένου, είτε αναπαριστώντας ένα μεγάλο αντικείμενο που πρέπει να φορτωθεί εφόσον ζητηθεί, είτε προστατεύοντας την πρόσβαση προς ένα ευαίσθητο αντικείμενο. Τα πρότυπα Proxy παρέχουν ένα επίπεδο ανακατεύθυνσης σε συγκεκριμένες ιδιότητες των αντικειμένων. Έτσι μπορούν να απαγορέψουν, να ενισχύσουν ή να αλλάξουν αυτές τις ιδιότητες. (Gamma et al., 1995).



Σχήμα 11.Διάγραμμα κλάσεων προτύπου Proxy

1.3.12. Μεσολαβητής (Mediator)

Το πρότυπο σχεδίασης "Μεσολαβητής" προσδιορίζει ένα αντικείμενο που ενσωματώνει ένα σύνολο από άλλα αντικείμενα και ορίζει το πως αυτά θα επικοινωνούν μεταξύ τους. Το πρότυπο αυτό μειώνει την σύζευξη με το να μην επιτρέπει την άμεση επικοινωνία μεταξύ δύο αντικειμένων και επιτρέποντας στον χρήστη να ορίσει τον τρόπο επικοινωνίας. Ανήκει στη κατηγορία των συμπεριφορικών προτύπων. (Gamma et al., 1995).

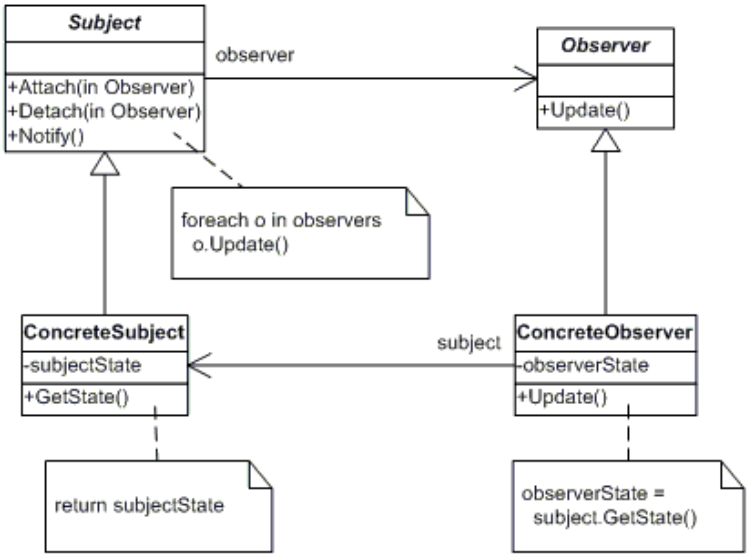


Σχήμα 12. Διάγραμμα κλάσεων προτύπου Mediator

1.3.13. Παρατηρητής (Observer)

Το πρότυπο σχεδίασης "Παρατηρητής" είναι επίσης γνωστό ως πρότυπο "Δημοσίευση-Εγγραφή" (Publish-Subscribe). Ορίζει μια σχέση εξάρτησης ένα-προς-πολλά μεταξύ αντικειμένων έτσι ώστε όταν μεταβάλλεται η κατάσταση ενός αντικειμένου, όλα τα εξαρτώμενα αντικείμενα να ενημερώνονται και τροποποιούνται αυτόματα. Ανήκει στην κατηγορία των συμπεριφορικών προτύπων.

Το πρότυπο Observer χρησιμοποιείται όταν η αλλαγή της κατάστασης ενός αντικειμένου (υποκείμενο) απαιτεί την ειδοποίηση άλλων αντικειμένων (παρατηρητών) , χωρίς το υποκείμενο να πρέπει να κάνει καμία υπόθεση για το ποια είναι τα αντικείμενα-παρατηρητές. Τέλος αξίζει να σημειωθεί, ότι η λίστα των παρατηρητών μπορεί να αλλάζει κατά τη διάρκεια εκτέλεσης του προγράμματος (Chatzigeorgiou, 2005).

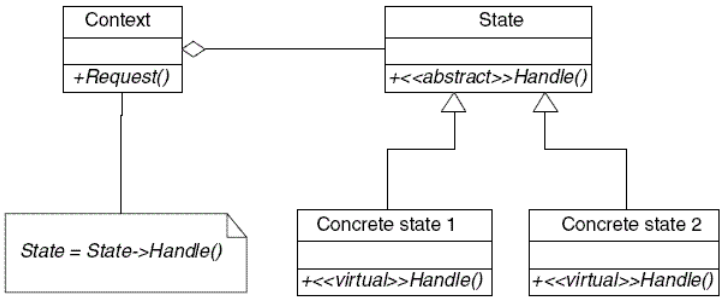


Σχήμα 13. Διάγραμμα κλάσεων προτύπου Observer

1.3.14. Κατάσταση (State)

Το πρότυπο σχεδίασης "Κατάσταση" ενθυλακώνει την κατάσταση ενός αντικειμένου, ώστε να μπορεί να αλλάξει τη συμπεριφορά του, όταν αλλάξει η εσωτερική κατάσταση του αντικειμένου. Ανήκει στην κατηγορία των συμπεριφορικών προτύπων.

Το πρότυπο State δίνει τη δυνατότητα σε ένα αντικείμενο να συμπεριφέρεται σαν να αλλάζει η κλάση του, κάτι που στις περισσότερες αντικειμενοστραφείς γλώσσες είναι αδύνατο. Στο πρότυπο, η κλάση πελάτης, περιέχει μια αφηρημένη κλάση, η οποία όμως δεν αντιπροσωπεύει μια στρατηγική αλλά μια κατάσταση. Οι παράγωγες κλάσεις υλοποιούν τις διάφορες καταστάσεις και κατά συνέπεια, η κλάση πελάτης, μπορεί να εναλλάξει την κατάστασή της αλλάζοντας την τιμή του δείκτη αναφοράς προς την επιθυμητή περιεχόμενη κατάσταση (Gamma et al., 1995).

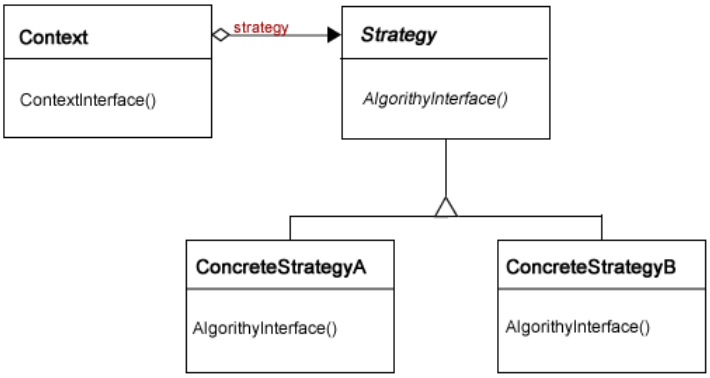


Σχήμα 14. Διάγραμμα κλάσεων προτύπου State

1.3.15. Στρατηγική (Strategy)

Το πρότυπο σχεδίασης "Στρατηγική" ορίζει μια οικογένεια αλγορίθμων, τους οποίους ενσωματώνει και επιτρέπει την εναλλαγή μεταξύ αυτών. Το πρότυπο δίνει την δυνατότητα μεταβολής των αλγορίθμων, ανεξάρτητα από τους πελάτες που τους χρησιμοποιούν. Ανήκει στην κατηγορία των συμπεριφορικών προτύπων.

Χρησιμοποιείται όταν ένας αλγόριθμος αναμένεται να έχει πολλές εναλλακτικές υλοποιήσεις ή ενδέχεται να προστεθούν κι άλλες στο μέλλον. (Chatzigeorgiou, 2005).

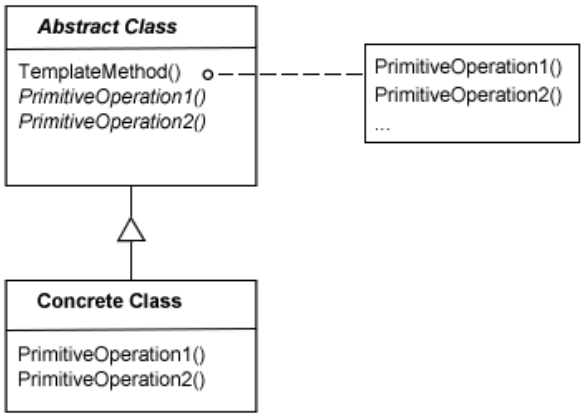


Σχήμα 15. Διάγραμμα κλάσεων προτύπου Stategy

1.3.16. Μέθοδος Υπόδειγμα (Template Method)

Το πρότυπο σχεδίασης "Μέθοδος Υπόδειγμα" ορίζει το περίγραμμα ενός αλγορίθμου σε μια λειτουργία, αφήνοντας ορισμένα βήματα στις παράγωγες κλάσεις. Το πρότυπο επιτρέπει στις παράγωγες κλάσεις να επαναορίσουν ορισμένα βήματα του αλγορίθμου χωρίς να αλλάξουν τη δομή του. Ανήκει στην κατηγορία των συμπεριφορικών προτύπων.

Χρησιμοποιείται για τον ορισμό των αμετάβλητων τμημάτων ενός αλγορίθμου σε μια λειτουργία μιας κλάσης και την μετάθεση της υλοποίησης των μεταβλητών τμημάτων του αλγορίθμου σε παράγωγες κλάσεις. (Chatzigeorgiou, 2005).

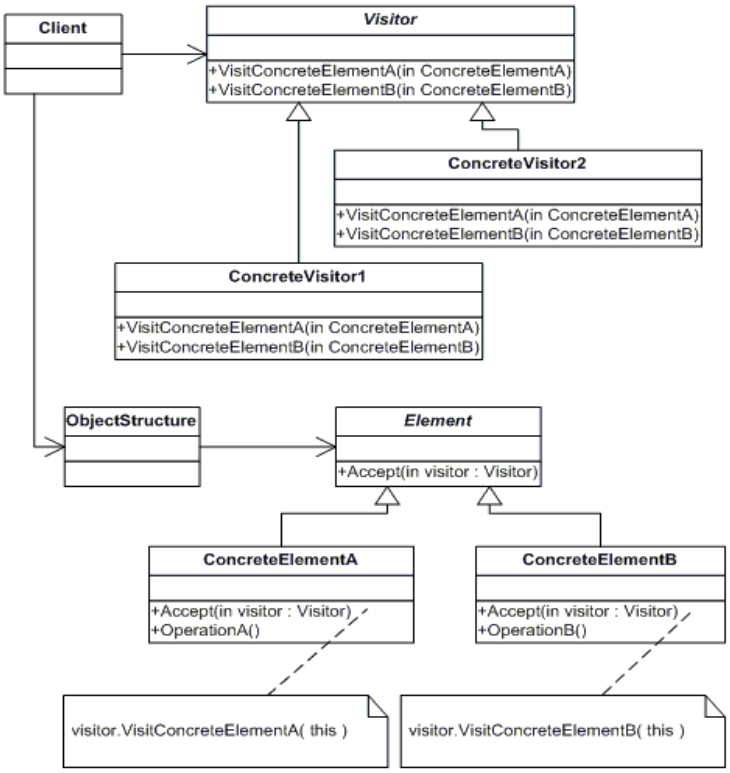


Σχήμα 16. Διάγραμμα κλάσεων προτύπου Template Method

1.3.17. Επισκέπτης (Visitor)

Το πρότυπο σχεδίασης "Επισκέπτης" έχει ως στόχο την αναπαράσταση μιας λειτουργίας που πρόκειται να πραγματοποιηθεί στα στοιχεία μιας δομής αντικειμένων. Το πρότυπο επιτρέπει τον ορισμό μιας νέας λειτουργίας χωρίς την τροποποίηση των κλάσεων των στοιχείων στα οποία επιδρά. Ανήκει στην κατηγορία των συμπεριφορικών προτύπων.

Χρησιμοποιούμε το πρότυπο "Επισκέπτης" όταν θέλουμε να προσθέσουμε λειτουργίες στα αντικείμενα μιας ιεραρχίας χωρίς να "μολύνουμε" τις κλάσεις τους με αυτές τις λειτουργίες. Η χρήση του προτύπου μας επιτρέπει να διατηρήσουμε σχετιζόμενες λειτουργίες σε ένα μέρος ορίζοντας αυτές σε μια ξεχωριστή κλάση. (Chatzigeorgiou, 2005).

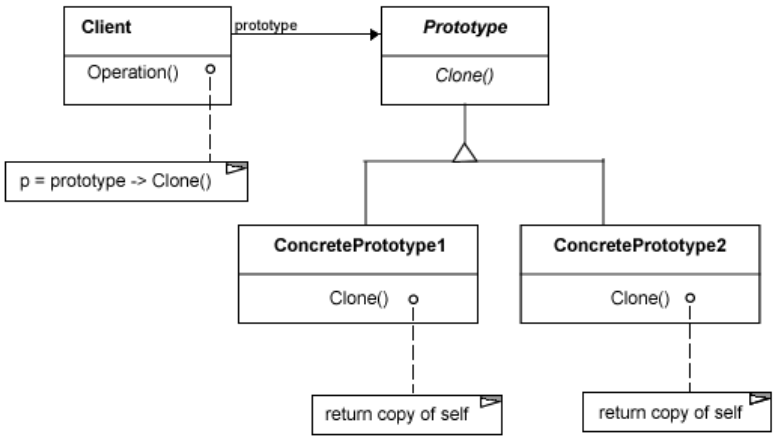


Σχήμα 17. Διάγραμμα κλάσεων προτύπου Visitor

1.3.18. Πρωτότυπο (Prototype)

Το πρότυπο σχεδίασης "Πρωτότυπο" προσδιορίζει τους τύπους των αντικειμένων που θα δημιουργηθούν βάση μιας πρωτότυπης διασύνδεσης, και δημιουργεί νέα αντικείμενα αντιγράφοντας το πρωτότυπο. Ανήκει στην κατηγορία των κατασκευαστικών προτύπων.

Το πρότυπο αυτό, χρησιμοποιείται για την αποφυγή δημιουργίας υποκλάσεων ενός αντικειμένου δημιουργού στην εφαρμογή και την αποφυγή του κόστους κληρονομικότητας για την δημιουργία ενός νέου αντικειμένου με το συμβατικό τρόπο (Gamma et al., 1995).



Σχήμα 18. Διάγραμμα κλάσεων προτύπου Prototype

1.4. Ανίχνευση Προτύπων Σχεδίασης

Πολλές διαφορετικές προσεγγίσεις και άρθρα έχουν δημοσιευτεί για την διαδικασία της αναγνώρισης προτύπων σχεδίασης μέσα στον κώδικα εφαρμογών. Βάση της βιβλιογραφικής μελέτης παρουσιάζονται οι σημαντικότερες προσεγγίσεις οι οποίες ποικίλλουν ανάλογα με τον τρόπο επεξεργασίας και ανάλυσης του κώδικα από τον οποίο θέλουμε να κάνουμε εξαγωγή προτύπων σχεδίασης.

Για παράδειγμα οι (De Lucia et al., 2009 και Wendehals και Orso, 2006), ασχολήθηκαν με την ανάκτηση προτύπων συμπεριφοράς συνδυάζοντας την στατική και δυναμική ανάλυση καθώς και οι (Antoniol et al., 1998 , Antoniol et al., 2001 και De Lucia et al., 2007) οι οποίοι ασχολήθηκαν με την ανάκτηση κατασκευαστικών προτύπων. Στα (Antoniol et al., 1998 και Antoniol et al., 2001) παρουσιάζεται μια προσέγγιση πολλών επιπέδων για την εξαγωγή κατασκευαστικών προτύπων σχεδίασης από αντικειμενοστραφή αντικείμενα, είτε σχέδια είτε κώδικα. Στο (De Lucia et al., 2007) παρουσιάζεται μια προσέγγιση δύο φάσεων για την ανάκτηση των κατασκευαστικών προτύπων σχεδίασης. Στην πρώτη φάση το διάγραμμα κλάσεων που εξάγεται από τον πηγαίο κώδικα αναλύεται για τον προσδιορισμό των σχεδιαστικών δομών που είναι υποψήφιες περιπτώσεις προτύπων. Αυτό ολοκληρώνεται χρησιμοποιώντας μια τεχνική ανάκτησης βασισμένη στην οπτική γλωσσική ανάλυση. Η δεύτερη φάση είναι η επέκταση της προηγούμενης προσέγγισης, που στοχεύει στη βελτίωση των αποτελεσμάτων της απόδοσης, της ακρίβειας και του χρόνου. Στο (Costagliola et al., 2005) οι συγγραφείς προτείνουν μια αντικειμενοστραφή προσέγγιση ανάκτησης προτύπων σχεδίασης που χρησιμοποιεί μια βιβλιοθήκη προτύπων σχεδίασης, εκφράζεται από την σκοπιά οπτικών γραμματικών, και βασίζεται σε μια τεχνική οπτικής γλωσσικής ανάλυσης. Παρουσιάζεται επίσης ένα οπτικό περιβάλλον που υποστηρίζει τη διαδικασία, ανακτώντας αυτόματα τα πρότυπα σχεδίασης από τα εισαγόμενα διαγράμματα κλάσεων σε UML. Μία επέκταση της προηγούμενης προσέγγισης επιχειρείται στο (Costagliola et al., 2006). Το (Amiot et al., 2001) παρουσιάζει ένα σύνολο εργαλείων και τεχνικών για να βοηθήσει στη σχεδίαση, κατανόηση, και επανασχεδίαση ενός τμήματος λογισμικού, χρησιμοποιώντας πρότυπα σχεδίασης. Στο (Niere et al., 2002) έχει σχεδιαστεί μια προσέγγιση ειδικά για να υπερνικήσει τα προβλήματα εκλεξιμότητας που προκαλούνται από τις πολλές παραλλαγές των προτύπων σχεδίασης σε επίπεδο σχεδιασμού και εφαρμογής. Είναι βασισμένο σε έναν νέο αλγόριθμο αναγνώρισης που λειτουργεί επ-αυξητικά αντί να προσπαθεί να αναλύσει ένα μεγάλο σύστημα λογισμικού σε ένα μόνο πέρασμα χωρίς οποιαδήποτε ανθρώπινη επέμβαση. Στο άρθρο (Wang και Tzerpos, 2005) παρουσιάζεται μια προσέγγιση για την ανίχνευση προτύπων σχεδίασης σε συστήματα Eiffel που ασχολούνται και με τη στατική δομή και με τη δυναμική συμπεριφορά των συνθετικών.

Τα (Kramer και Prechelt, 1996, Balanyi και Ferenc, 2003, Asencio et al., 2002 και Shi και Olsson, 2006) αναφέρονται σε μεθοδολογίες ανάκτησης απευθείας από πηγαίο κώδικα. Το (Kramer και Prechelt, 1996) παρουσιάζει μια προσέγγιση, αποκαλούμενη σύστημα Pat, το οποίο εξάγει σχεδιαστικές πληροφορίες απευθείας από C++ αρχεία επικεφαλίδων και τις αποθηκεύει. Τα πρότυπα εκφράζονται με κανόνες Prolog και οι σχεδιαστικές πληροφορίες μεταφράζονται σε γεγονότα. Έπειτα χρησιμοποιείται ένα απλό ερώτημα Prolog για την αναζήτηση όλων των προτύπων. Το (Balanyi και Ferenc, 2003) παρουσιάζει μια νέα μέθοδο για τα την ανακάλυψη προτύπων σχεδίασης μέσα σε πηγαίο κώδικα C++, που περιλαμβάνει την ανίχνευση μεταβίβασης κλήσεων μεθόδων, της δημιουργίας αντικειμένων και της υπερφόρτωσης συναρτήσεων και παρέχει μια ακριβή προδιαγραφή για το πώς λειτουργούν τα πρότυπα. Στο (Asencio et al., 2002) αναπτύσσονται κάποια εργαλεία, που εξετάζουν στατικά τον πηγαίο κώδικα, αναγνωρίζουν αυτόματα τη χρήση προτύπων σχεδίασης και συσχετίζουν τη χρήση προτύπων με την ποιότητα λογισμικού, κωδικοποιώντας τους στόχους και τις προσδοκίες εφαρμοσμένης μηχανικής συστημάτων για τον πηγαίο κώδικα, και δίνοντας έτσι πρόσβαση για να αποκαλυφθούν σημαντικές σχεδιαστικές αποφάσεις. Όλες οι παραπάνω προσεγγίσεις έχουν εφαρμοστεί σε συστήματα ανοιχτού λογισμικού και έχουν ελεγχθεί για την αποτελεσματικότητά τους.

1.4.1. Εργαλεία Ανίχνευσης Προτύπων Σχεδίασης

Για την ανίχνευση και την εξαγωγή προτύπων από τα συστήματα ανοιχτού λογισμικού με σκοπό την παραγωγή «υποψήφιων συστατικών (components candidates)» χρησιμοποιήθηκαν δύο εργαλεία υλοποιούν δύο διαφορετικές προσεγγίσεις. Το εργαλείο «Design Pattern Detection Using Similarity Scoring» είναι γραμμένο στην γλώσσα προγραμματισμού Java και έχει την ικανότητα να αναγνωρίζει τα πρότυπα Προσαρμογέας, Σύνθετο, Διακοσμητής, Μέθοδος Εργοστάσιο, Παρατηρητής, Πρωτότυπο, Μοναδιαίο, Πληρεξούσιο, Κατάσταση/Στρατηγική, Μέθοδος Υπόδειγμα και επισκέπτης μελετώντας τον μεταγλωττισμένο κώδικα (bytecode) εφαρμογών γραμμένων στην Java. Οι συγγραφείς που το ανέπτυξαν προτείνουν μια μεθοδολογία, που βασίζεται στην καταγραφή των ομοιοτήτων μεταξύ των κορυφών ορισμένων γραφικών παραστάσεων (Tsantalis et al., 2006). Η αξιολόγηση σε τρία

προγράμματα ανοιχτού λογισμικού κάνει εμφανή την ακρίβεια και την αποδοτικότητα της προτεινόμενης μεθόδου.

Το δεύτερο εργαλείο «Pattern Inference and Recovery Tool (PINOT)», ένα πρωτότυπο εργαλείο που χρησιμοποιείται για την εφαρμογή μιας νέας, πλήρως αυτοματοποιημένης προσέγγιση ανίχνευσης προτύπων (Shi και Olsson, 2006), που βασίζεται σε μια νέα επαναταξινόμηση των GoF προτύπων σύμφωνα με τις προθέσεις του καθενός, η οποία λέγεται ότι ταιριάζει καλύτερα στην αντίστροφη μηχανική.. Το PINOT ανιχνεύει όλα τα πρότυπα GoF που έχουν σαφείς ορισμούς καθοδηγούμενους από τη δομή του κώδικα ή τη συμπεριφορά του συστήματος και είναι ένα πλήρως αυτοματοποιημένο εργαλείο ανίχνευσης προτύπων που είναι γρηγορότερο, ακριβέστερο, και περιεκτικότερο από τα υπάρχοντα εργαλεία. Το μεγαλύτερο μέρος της ανάπτυξής του έγινε σε γλώσσα C++ ενώ κάποιες λειτουργίες υλοποιούνται στις γλώσσες Java και Perl. Τέλος, το εργαλείο επεξεργάζεται τον πηγαίο κώδικα (source code) εφαρμογών γραμμένων σε Java.

1.5. Εμπειρικές Μελέτες

Η διεξαγωγή εμπειρικών μελετών αποσκοπεί στην απόκτηση αντικειμενικών και στατιστικά τεκμηριωμένων αποτελεσμάτων, όσον αφορά την κατανόηση, τον έλεγχο, την πρόβλεψη και τη βελτίωση της ανάπτυξης λογισμικού. Στις εμπειρικές μελέτες υπάρχουν δύο τύποι παραδειγμάτων ερευνών που ακολουθούν διαφορετικές προσεγγίσεις: οι ποιοτικές έρευνες (qualitative research) και οι ποσοτικές (quantitative research). Η ποιοτική έρευνα σχετίζεται με τη μελέτη αντικειμένων στις φυσικές τους συνθήκες. Ένας ποιοτικός ερευνητής προσπαθεί να ερμηνεύσει ένα φαινόμενο βασισμένος στις ερμηνείες που δίνουν οι άνθρωποι. Η ποσοτική έρευνα σχετίζεται με την ποσοτικοποίηση μιας σχέσης ή με τη σύγκριση δύο ή περισσότερων συνόλων. Σκοπός της είναι η δόμηση μιας σχέσης αιτίου αποτελέσματος. Μια ποσοτική έρευνα διεξάγεται μέσα από ελεγχόμενα πειράματα ή συλλογή δεδομένων από μελέτες περιπτώσεων και είναι κατάλληλη όταν ελέγχονται τα αποτελέσματα μιας δραστηριότητας ή ενός χειρισμού. Ένα πλεονέκτημά της είναι ότι τα ποσοτικά δεδομένα επιτρέπουν τις συγκρίσεις των αποτελεσμάτων και τη στατιστική ανάλυση. Σύμφωνα με το (Wohlin et al., 2000), υπάρχουν τρεις βασικές ερευνητικές μέθοδοι που

χρησιμοποιούνται στις εμπειρικές μελέτες, οι μελέτες πεδίου, οι μελέτες περίπτωσης και τα πειράματα. Η επιλογή της προσέγγισης που θα ακολουθηθεί γίνεται συνήθως λαμβάνοντας υπόψη τη φύση και το αντικείμενο της εκάστοτε έρευνάς.

Η τεχνική των μελετών περίπτωσης χρησιμοποιείται για την παρακολούθηση έργων, ενεργειών ή εργασιών. Καθ' όλη τη διάρκεια της μελέτης συλλέγονται δεδομένα για έναν συγκεκριμένο σκοπό τα οποία συνήθως περνούν από στατιστική ανάλυση για την εξαγωγή αποτελεσμάτων. Τα πειράματα διεξάγονται συνήθως σε εργαστηριακό περιβάλλον, που παρέχει υψηλά επίπεδα ελέγχου. Τα υποκείμενα του πειράματος που έχουν επιλεγεί με βάση ορισμένα κριτήρια, εξετάζονται τυχαία σε κάποια καθήκοντα. Έπειτα ακολουθεί στατιστική ανάλυση των αποτελεσμάτων του πειράματος, ενώ υπολογίζεται και ο βαθμός χειραγώγησης ορισμένων μεταβλητών. Η διαφορά μεταξύ μιας μελέτης περίπτωσης και ενός πειράματος είναι ότι το δείγμα των μεταβλητών που χρησιμοποιούνται σε ένα πείραμα μπορεί να χειραγωγηθεί, ενώ σε μια μελέτη περίπτωσης το δείγμα των μεταβλητών προέρχεται από την αναπαράσταση μιας πραγματικής κατάστασης. Τέλος, μια μελέτη πεδίου, χρησιμοποιείται συνήθως για την διεξαγωγή μιας αναδρομικής εξέτασης, όταν για παράδειγμα ένα εργαλείο ή μια τεχνική χρησιμοποιείται για ένα χρονικό διάστημα. Η μελέτη γίνεται με χρήση ερωτηματολογίων που διανέμονται σε ένα αντιπροσωπευτικό δείγμα πληθυσμού που θέλουμε να μελετήσουμε. Τα αποτελέσματα της έρευνας αναλύονται και στην τελική φάση γενικεύονται για τον πληθυσμό απ' όπου προέρχονταν το δείγμα.

Για την ανάγκες της συγκεκριμένης έρευνας η μέθοδος της «μελέτης περίπτωσης» κρίθηκε ως η πλέον κατάλληλη λόγω του τεράστιου όγκου πληροφοριών που ήταν απαραίτητο να επεξεργαστούμε καθώς και για την αυτοματοποίηση της διαδικασίας. Γι αυτό το λόγο ακολουθεί και μια εκτενέστερη ανάλυση της μεθοδολογίας.

1.5.1. Μελέτη Περίπτωσης (Case Study)

Μια μελέτη περίπτωσης συντάσσεται για να μελετήσει μια οντότητα ή ένα φαινόμενο μέσα σε συγκεκριμένα χρονικά πλαίσια (Wohlin et al., 2000). Ο ερευνητής συγκεντρώνει λεπτομερείς πληροφορίες συχνά εφαρμόζοντας διάφορες διαδικασίες συλλογής πληροφοριών. Σε μια μελέτη περίπτωσης, πρώτα εντοπίζονται οι κύριοι

παράγοντες, όπως είσοδοι, περιορισμοί, πόροι και έξοδοι, που μπορούν να επηρεάσουν το αποτέλεσμα κάποιας δραστηριότητας, και στη συνέχεια τεκμηριώνεται ο καθένας από αυτούς ξεχωριστά. Ο σκοπός μιας μελέτης περίπτωσης είναι η σύγκριση μιας κατάστασης με κάποια άλλη παρόμοια, όπως για παράδειγμα η σύγκριση των αποτελεσμάτων που προκύπτουν από τη χρήση μεθόδων ή εργαλείων. Αποφασίζεται εκ των προτέρων τι ακριβώς θα ερευνηθεί κι έπειτα εντοπίζονται οι προς έλεγχο παράγοντες και οργανώνεται ο τρόπος συλλογής των δεδομένων.

Οι μελέτες περίπτωσης έχουν πλεονεκτήματα και μειονεκτήματα. Είναι ιδιαίτερα σημαντικές, γιατί ενσωματώνουν χαρακτηριστικά, όπως η κλιμάκωση, η πολυπλοκότητα και ο δυναμισμός, που δεν μπορούν να απεικονιστούν με τις άλλες μεθόδους. Το μεγαλύτερο μειονέκτημα της μεθόδου είναι η έλλειψη ελέγχου της κατάστασης που διερευνάται, με συνέπεια τα αποτελέσματα να μη γίνονται αποδεκτά με βεβαιότητα. Επίσης, οι μικρές ή απλοποιημένες μελέτες περίπτωσης σπάνια αποτελούν ένα καλό όργανο για την ανακάλυψη αρχών και τεχνικών που αφορούν την τεχνολογία λογισμικού.

1.5.2. Μεθοδολογία Διενέργειας Μελέτης Περίπτωσης

Σύμφωνα με το (Kitchenham et al., 1995) τα βήματα που απαιτούνται για να συντάξει κανείς μια μελέτη περίπτωσης περιλαμβάνουν:

1) Ορισμό μιας υπόθεσης.
2) Επιλογή ορισμένων εφαρμογών.
3) Επιλογή της μεθόδου σύγκρισης.
4) Ελαχιστοποίηση των παραγόντων σύγχυσης.
5) Σχεδιασμό πλάνου για την μελέτη περίπτωσης.
6) Παρακολούθηση της μελέτης περίπτωσης.
7) Ανάλυση και αναφορά των αποτελεσμάτων

Για να ορίσουμε την υπόθεση, ξεκινάμε ορίζοντας την επίδραση που περιμένουμε να έχει η μέθοδος. Ο ορισμός αυτός πρέπει να είναι αρκετά λεπτομερής, και να ξεκαθαρίζει τις μετρήσεις που πρέπει να γίνουν για να προκύψει το αποτέλεσμα. Επίσης, είναι σημαντικό να οριστεί τι δεν αναμένεται να συμβεί. Αυτό είναι ιδιαίτερα

σημαντικό γιατί επίσημα δεν μπορούμε να αποδείξουμε ότι μια υπόθεση αληθεύει. Μπορούμε μόνο να την καταρρίψουμε. Γι αυτό δηλώνουμε και μια μηδενική υπόθεση για να δείξουμε ότι δεν υπάρχει διαφοροποίηση στην μεταχείριση.

Κατά την επιλογή των εφαρμογών, είναι σημαντικό να επιλέξουμε εφαρμογές του ίδιου τύπου με αυτές που μας ενδιαφέρουν. Η επιλογή πρέπει να βασίζεται όχι μόνο στον τύπο της εφαρμογής, αλλά και στην συχνότητα που ο κάθε τύπος αναπτύσσεται.

Για το τρίτο βήμα πρέπει να έχουμε κατά νου, ότι η μελέτη περίπτωσης είναι από την φύση της συγκριτική μέθοδος όσον αφορά τα αποτελέσματά δύο ή περισσοτέρων μεθόδων. Για να επιβεβαιώσουμε την εσωτερική εγκυρότητα, πρέπει να βρούμε μια έγκυρη βάση για την αξιολόγηση των αποτελεσμάτων της μελέτης περίπτωσης. Για να επιτευχθεί αυτό υπάρχουν τρείς τρόποι: (1) Να επιλέξουμε ένα παρόμοιο έργο για αν συγκρίνουμε τα αποτελέσματα, (2) να συγκρίνουμε τα αποτελέσματα χρησιμοποιώντας μια νέα μέθοδο αντίθετη με τη γραμμή της μεθόδου που εμείς χρησιμοποιούμε και (3) εφόσον η μέθοδος μπορεί να εφαρμοστεί σε ανεξάρτητα συστατικά, να την εφαρμόσουμε τυχαία σε ορισμένα μόνο συστατικά προϊόντων.

Το τέταρτο βήμα αφορά την ελαχιστοποίηση των παραγόντων σύγχυσης. Τέτοιοι παράγοντες μπορεί να είναι η εκμάθηση του τρόπου χρήσης μιας μεθόδου ή ενός εργαλείου κατά τη διάρκεια της προσπάθειας αξιολόγησής του, η χρήση είτε πολύ ενθουσιώδους είτε πολύ δύσπιστου προσωπικού σχετικά με τη χρήση της μεθόδου ή του εργαλείου, η σύγκριση διαφορετικών τύπων εφαρμογών, κ.α. Ορισμένες φορές, μπορούμε να ελέγξουμε τη σύγχυση, μετρώντας τον παράγοντα σύγχυσης και ρυθμίζοντας τα αποτελέσματα ανάλογα.

Στην επόμενη φάση, το πλάνο αναγνωρίζει και καταγράφει όλες τις πτυχές που πρέπει να διευθετηθούν για την ομαλή διεξαγωγή της αξιολόγησης της μεθόδου. Σε αυτές, συμπεριλαμβάνονται τα απαραίτητα μέτρα, οι διαδικασίες συλλογής δεδομένων και το ανθρώπινο δυναμικό που είναι υπεύθυνο για τη συλλογή και ανάλυση των δεδομένων.

Η παρακολούθηση της μελέτης περίπτωσης σύμφωνα με το πλάνο του προηγούμενου βήματος, επιβεβαιώνει ότι οι μέθοδοι ή τα εργαλεία που εξετάζονται

χρησιμοποιούνται σωστά, και ότι όλοι οι παράγοντες που θα μπορούσαν να προδιαθέσουν τα αποτελέσματα καταγράφονται. Αυτό, θα βοηθήσει στο τέλος της έρευνας στην συγγραφή μιας αναφοράς αξιολόγησης με σκοπό την πρόταση αλλαγών στις διαδικασίες.

Τέλος, για την ανάλυση και αναφορά των αποτελεσμάτων, η διαδικασία που ακολουθείται κάθε φορά, εξαρτάται από τον αριθμό των δεδομένων που πρέπει να αναλυθούν.

2. Κατασκευή Λογισμικού Ανίχνευσης Υποψήφιων Συστατικών

Η διαδικασία της ανίχνευσης των υποψήφιων συστατικών στηρίζεται σε ένα σύνολο εφαρμογών ανοιχτού λογισμικού αλλά και εργαλείων που αναπτύχθηκαν για το σκοπό αυτό κατά την διάρκεια της Πτυχιακής. Οι εφαρμογές ανοιχτού λογισμικού που χρησιμοποιήσαμε είναι οι εξής:

- Classycle (http://classycle.sourceforge.net/) η οποία αναλύει τον κώδικα εφαρμογής και εξετάζει τις συνδέσεις μεταξύ των κλάσεων της και των πακέτων της.
- Chidamber and Kemerer Java Metrics - ckjm (http://www.spinellis.gr/sw/ckjm/) η οποία υπολογίζει τις μετρικές αντικειμενοστραφούς σχεδίασης που προτείνουν οι Chidamber and Kemerer αναλύοντας τον μεταγλωττισμένο κώδικα μιας εφαρμογής γραμμένη σε java.
- Design Pattern Detection Tool (http://java.uom.gr/~nikos/files/pattern-detection/pattern4.jar) η οποία ανακτά πρότυπα σχεδίασης αναλύοντας τον μεταγλωττισμένο κώδικα μιας java εφαρμογής.
- Pattern Inference and Recovery Tool - pinot (http://www.cs.ucdavis.edu/~shini/research/pinot/) η οποία ανακτά πρότυπα σχεδίασης αναλύοντας τον πηγαίο κώδικα μιας java εφαρμογής.

Είναι πολύ σημαντικό να ξεκαθαρίσουμε τα βήματα με τα οποία εκτελείται το τελικό εργαλείο και συνδυάζει όλες τις παραπάνω εφαρμογές. Για την ολοκληρωμένη χρήση του τελικού εργαλείου είναι απαραίτητη η ύπαρξη του πηγαίου κώδικα και του εκτελέσιμου αρχείου της εφαρμογής από την οποία θέλουμε να ανακτήσουμε "υποψήφια συστατικά λογισμικού". Επίσης είναι σημαντικό τα αρχεία που περιέχονται στον πηγαίο και μεταγλωττισμένο κώδικα να συμφωνούν μεταξύ τους. Παρακάτω αναλύονται συνοπτικά τα βήματα εκτέλεσης.

Βήμα 1ο. Αρχικοποίηση του project.

1. Στην αρχικοποίηση του Project ορίζονται τα μονοπάτια των απαραίτητων προς ανάλυση αρχείων καθώς και τον φακέλων που θα τοποθετηθούν τα

παραγόμενα αρχεία. Δημιουργείται ο φάκελος του Project με μονοπάτι "*./project/*".

2. Στη συνέχεια αποσυμπιέζεται το εκτελέσιμο αρχείο της εφαρμογής έτσι ώστε να είναι σε μορφή που μπορεί να επεξεργαστεί το Design Pattern Detection Tool. Τα αποσυμπιεσμένα αρχεία αποθηκεύονται στο φάκελο"*./project/bin/*".
3. Τέλος δημιουργείται ένας φάκελος στον οποίο αποθηκεύεται ένα αντίγραφο του πηγαίου κώδικα των αρχείων της εφαρμογής που θα αναλυθεί "*./project/project/*".

Βήμα 2ο. Ανάλυση των κλάσεων του project.

Εκτελείται το εργαλείο classycle βάση του οποίου αναλύεται το project από το οποίο θέλουμε να εξάγουμε "υποψήφια συστατικά λογισμικού". Παράγεται το αρχείο *"./project/project_classycle.xml"* το οποίο περιέχει όλες τις κλάσεις του project καθώς και τις εξαρτήσεις τους από άλλες κλάσεις του Project ή των βιβλιοθηκών του.

Βήμα 3ο. Εύρεση Μετρικών.

Εκτέλεση του εργαλείου "Chidamber and Kemerer Java Metrics" με παράμετρο τον μεταγλωττισμένο κώδικα του Project και παραγωγή του αρχείου *"./project/project_r_index_Metrics.txt"* το οποίο περιέχει όλες τις κλάσεις του Project με τις μετρικές τους.

Βήμα 4ο. Δημιουργία της "Λίστας Κλάσεων" (classList)

1. Διαβάζοντας το αρχείο που παράχθηκε στο βήμα 2 δημιουργείται μια λίστα από κλάσεις που ονομάζεται "classList" και περιέχει όλες τις κλάσεις του υπό ανάλυση project.
2. Βάση του ίδιου αρχείου προσθέτουμε σε κάθε κλάση της λίστας τις κλάσεις από τις οποίες εξαρτάται (Fan Out Classes) αλλά και τις κλάσεις από τις οποίες εξαρτιέται (Fan In Classes).

3. Διαβάζεται το αρχείο που παράχθηκε στο βήμα 3 και προστίθενται οι μετρικές που βρέθηκαν από το εργαλείο ckjm στην κάθε κατάλληλη κλάση.

Βήμα 5ο. Επικύρωση λίστας των κλάσεων (classList)

1. Σε κάθε κλάση της λίστας προσθέτουμε σε ένα νέο πεδίο (εάν υπάρχει) το μονοπάτι του αντίστοιχου αρχείου με τον πηγαίο κώδικα που υπάρχει στον αντίστοιχο φάκελο με τα αρχεία του πηγαίου κώδικα του project.
2. Αντίστοιχα με το παραπάνω προσθέτουμε σε ένα νέο πεδίο (εάν υπάρχει) το μονοπάτι του αντίστοιχου αρχείου με τον μεταγλωττισμένο κώδικα που υπάρχει στον αντίστοιχο φάκελο με τα αρχεία που δημιουργήθηκαν στο βήμα 1.2
3. Όσες από τις κλάσεις δεν αντιστοιχίστηκαν στα αρχεία τους χαρακτηρίζονται ως "μη έγκυρες" και διαγράφονται από την λίστα έτσι ώστε να μην συμμετέχουν στην δημιουργία των "υποψήφιων συστατικών".
4. Τέλος σε κάθε μία από τις εναπομείνασες κλάσεις αντιστοιχίζεται ένας χαρακτηριστικός μοναδικός ακέραιος αριθμός (uid).

Βήμα 6ο. Ανάκτηση Προτύπων Σχεδίασης

1. Εκτέλεση του εργαλείου "Design Pattern Detection Tool" για την ανάκτηση προτύπων σχεδίασης αναλύοντας τον πηγαίο κώδικα της εφαρμογής. Τα πρότυπα σχεδίασης που βρέθηκαν αποθηκεύονται στο αρχείο *"./project/project_TsantalisPatternResults.xml".*
2. Εκτέλεση του εργαλείου "Pattern Inference and Recovery Tool" για την ανάκτηση προτύπων σχεδίασης αναλύοντας τον μεταγλωττισμένο κώδικα της εφαρμογής. Τα πρότυπα σχεδίασης που βρέθηκαν αποθηκεύονται στο αρχείο *"./project/project_pinotPatternResults.xml".*

Βήμα 7ο. Ανάλυση αποτελεσμάτων Προτύπων Σχεδίασης

1. Διαβάζουμε το αρχείο που παράχθηκε στο βήμα 6.1 και αφού ξεκαθαρίσουμε τα πρότυπα σχεδίασης των οποίων όλες οι

συμμετέχουσες κλάσεις είναι έγκυρες τότε τα αποθηκεύουμε στο αρχείο *"./project/project_ TsantalisResults_csv.txt".*

2. Διαβάζουμε το αρχείο που παράχθηκε στο βήμα 6.2 και αφού ξεκαθαρίσουμε τα πρότυπα σχεδίασης των οποίων όλες οι συμμετέχουσες κλάσεις είναι έγκυρες τότε τα αποθηκεύουμε στο αρχείο *"./project/project_ pinotPattern_Results_csv.txt".*

Βήμα 8ο. Δημιουργία «Υποψήφιων Συστατικών»

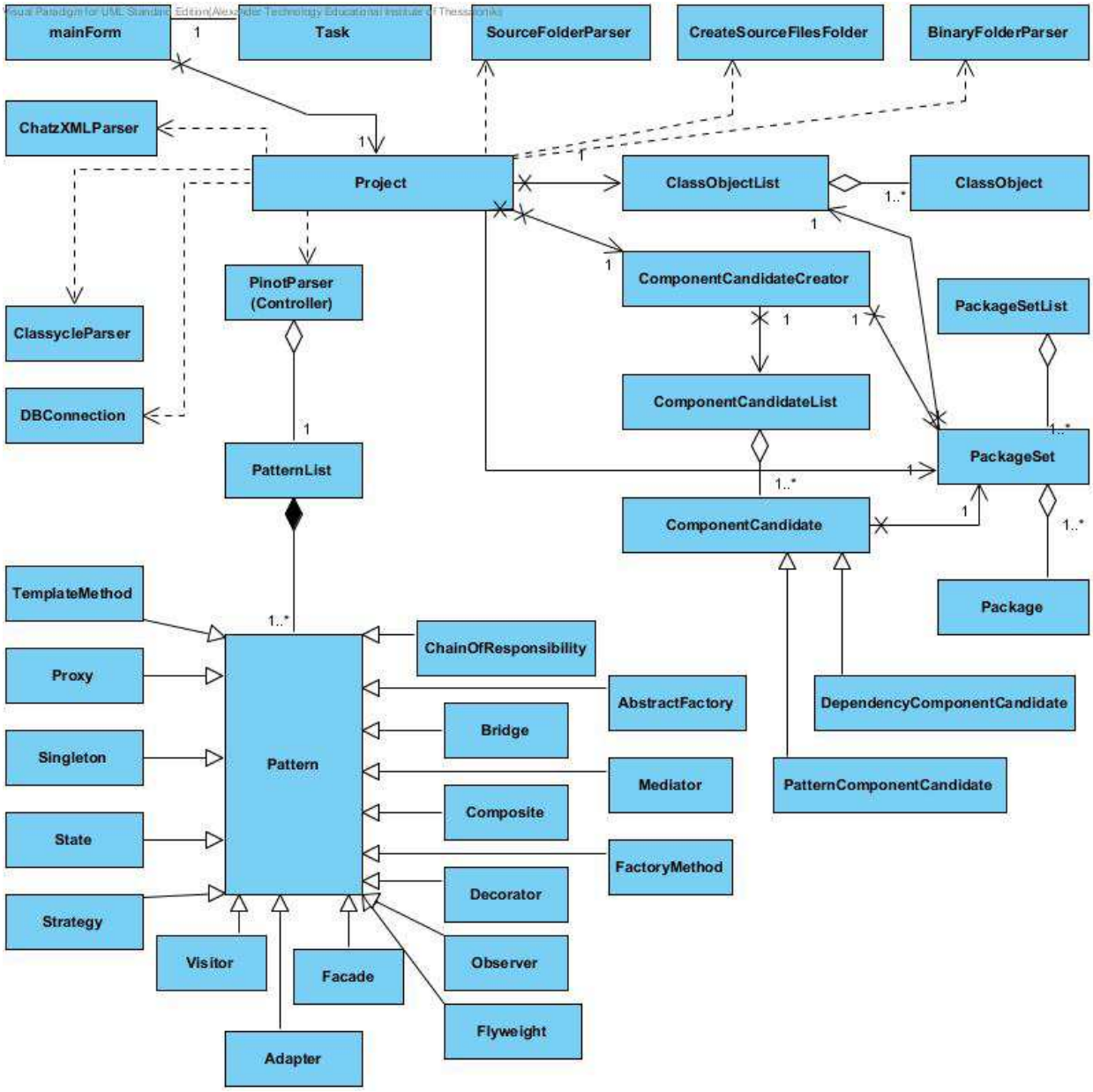
1. Διαβάζουμε το αρχείο που δημιουργήθηκε στο βήμα 7.1, δημιουργούμε τα «υποψήφια συστατικά» που βασίζονται σε πρότυπα σχεδίασης και τα αποθηκεύουμε στη λίστα.
2. Διαβάζουμε το αρχείο που δημιουργήθηκε στο βήμα 7.2, δημιουργούμε τα «υποψήφια συστατικά» που βασίζονται σε πρότυπα σχεδίασης και τα αποθηκεύουμε στη λίστα.
3. Για κάθε κλάση της λίστας που δημιουργήθηκε στο βήμα 4, εκτελούμε την διαδικασία παραγωγής «υποψήφιων συστατικών» -που βασίζονται σε εξαρτήσεις- και περιγράφεται αναλυτικά στο κεφάλαιο 2.1.
4. Αφού ολοκληρωθεί η παραπάνω διαδικασία, για κάθε «υποψήφιο συστατικό» που παρήχθη, εύρεση του συνόλου των πακέτων στα οποία ανήκουν οι κλάσεις που συμμετέχουν σε αυτό.
5. Για κάθε πακέτο και σύνολο πακέτων γίνεται ο υπολογισμός των μετρικών NOC, Fan In. Fan Out και Reusability Metric που τις χαρακτηρίζουν. Οι μετρικές αυτές περιγράφονται στο κεφάλαιο 2.3.2.

Βήμα 9ο. Αποθήκευση των «Υποψήφιων Συστατικών» στη Βάση Δεδομένων

1. Γίνεται η αρχικοποίηση του project από το οποίο δημιουργήσαμε τα «υποψήφια συστατικά» και στη συνέχεια αποθηκεύονται οι κλάσεις, τα πακέτα τέλος τα «υποψήφια συστατικά» στους κατάλληλους πίνακες της βάσης δεδομένων.

Παραθέτουμε και το διάγραμμα κλάσεων ολόκληρου του εργαλείου. Οι οντότητες των κλάσεων παρουσιάζονται χωρίς τις λεπτομέρειές τους (πεδία και μέθοδοι) έτσι ώστε να

είναι διαχειρίσιμο το μέγεθός του. Στα επόμενα κεφάλαια παρουσιάζουμε αναλυτικά όλα τα διαγράμματα κλάσεων κάθε υπο-εργαλείου ξεχωριστά καθώς γίνεται και μία σύντομη περιγραφή των λειτουργιών της κάθε κλάσης.



Σχήμα 19. Διάγραμμα κλάσεων εργαλείου "Ανίχνευσης Υποψήφιων Συστατικών" (χωρίς λεπτομέρειες)

- **mainForm** – Είναι η κλάση που υλοποιεί την διεπαφή χρήστη (γραφικό περιβάλλον) και διαχειρίζεται όλα τα γεγονότα (events) κατά την διάδραση του χρήστη με την εφαρμογή. Κατά την εκτέλεση της ανάλυσης ενός Project

αρχικοποιεί ένα Task τα πλεονεκτήματα της χρήσης του οποίου αναλύονται παρακάτω.

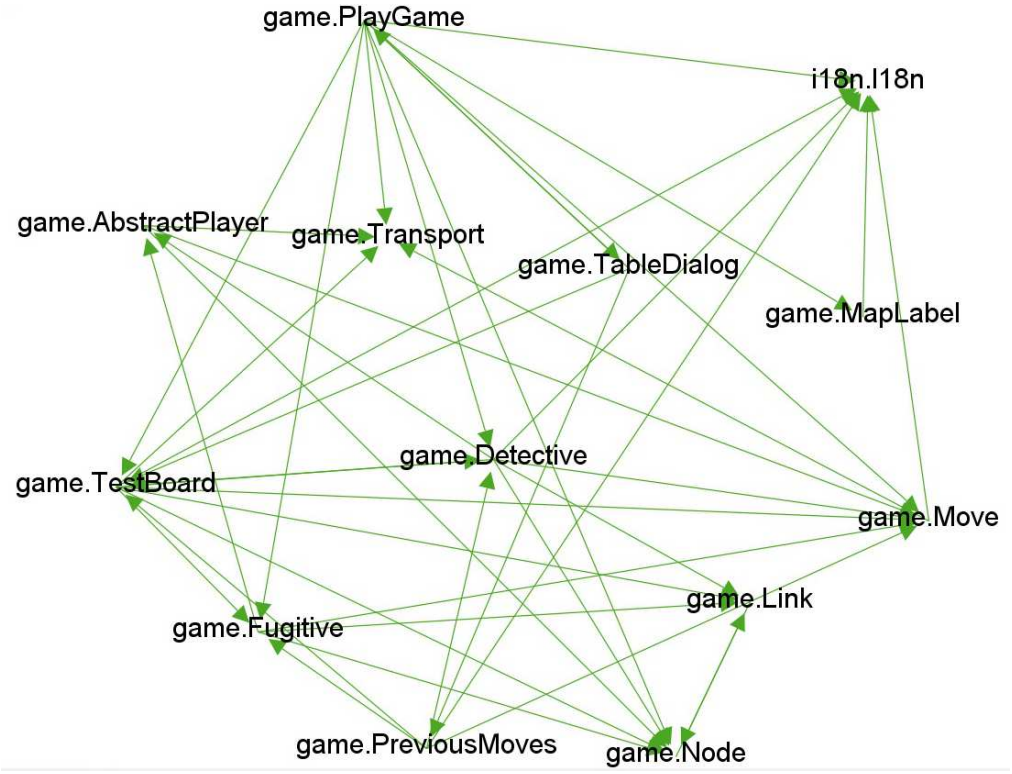
- **Task** – Επιτρέπει τον προγραμματισμό πολλαπλών νημάτων. Δίνει την δυνατότητα δηλαδή να εκτελεστεί η ανάλυση στο υπόβαθρο (background) ενώ οι λειτουργίες του γραφικού περιβάλλοντος είναι ενεργές στο χρήστη.
- **Project** – Είναι η κεντρική κλάση ελεγκτής (controller) όλου του συστήματος. Είναι υπεύθυνη για την μεταφορά των εντολών από το γραφικό περιβάλλον στο υπόλοιπο σύστημα. Αρχικοποιεί τις ιδιότητες του υπό ανάλυση project καθώς και εκτελεί όλα τα επί μέρους εργαλεία.
- **ClassycleParser** – Είναι η υπεύθυνη κλάση για την ανάλυση του xml αρχείου που περιέχει τις εξαρτήσεις των κλάσεων που παράγεται από το εργαλείο classycle.
- **ClassObject** – Αναπαριστά μία κλάση που συμμετέχει στο σύστημα. Περιέχει όλες τις απαραίτητες ιδιότητες όπως το όνομά της, το πακέτο στο οποίο ανήκει, τις κλάσεις από τις οποίες εξαρτάται και τις κλάσεις οι οποίες εξαρτώνται από αυτή. Επίσης διατηρεί λίστες με τις ιδιότητες και τις μεθόδους της κλάσης καθώς και τις τιμές όλων των μετρικών. Τέλος διατηρεί δύο πεδία με το μονοπάτι του πηγαίου κώδικα και του μεταγλωττισμένου κώδικα της συγκεκριμένης κλάσης και σε περίπτωση που κάποιο από αυτά τα δύο λείπει χαρακτηρίζεται ως «μη έγκυρη».
- **ClassObjectList** – Διατηρεί μία λίστα από αντικείμενα (ClassObjects) για τα οποία είναι υπεύθυνη να αρχικοποιήσει, προσθέσει και ελέγξει για την εγκυρότητά τους. Τέλος προσφέρει την δυνατότητα εγγραφής τους στη βάση δεδομένων.
- **BinaryFolderParser** – Είναι μία κλάση που αναλαμβάνει την ανάλυση του φακέλου με τα μεταγλωττισμένα αρχεία κώδικα του εκάστοτε υπό ανάλυση project. Αντιστοιχίζει σε κάθε κλάση το αντίστοιχο μονοπάτι για το αρχείο μεταγλωττισμένου κώδικά της.
- **SourceFolderParser** – Είναι η κλάση που αναλαμβάνει την ανάλυση του φακέλου με τα αρχεία πηγαίου κώδικα του εκάστοτε υπό ανάλυση project.

Αντιστοιχίζει σε κάθε κλάση το αντίστοιχο μονοπάτι για το αρχείο πηγαίου κώδικά της.

- **CreateSourceFilesFolder** – Είναι η κλάση που αναλαμβάνει την ανάλυση του φακέλου με τα αρχεία πηγαίου κώδικα του εκάστοτε υπό ανάλυση project. Αντιγράφει μόνο τα αρχεία που περιέχουν κώδικα java βάση της αρχικής δενδρικής μορφής.
- **ChatzXMLParser** – Είναι η κλάση που διαβάζει και αναλύει το xml αρχείο που παρήχθη από το εργαλείο ανάκτηση προτύπων σχεδίασης “Pattern Detection Tool”. Βρίσκει τους συμμετέχοντες για κάθε ένα από τα πρότυπα σχεδίασης που ανέκτησε το εργαλείο και εάν το κάθε πρότυπο μετά την ανάλυσή του χαρακτηρίζεται ως «έγκυρο» τότε αυτό εγγράφεται στο κατάλληλο αρχείο κειμένου από το οποίο θα δημιουργηθούν αργότερα τα «υποψήφια συστατικά» που βασίζονται σε πρότυπα σχεδίασης.

2.1. Μεθοδολογία Ανίχνευσης Υποψήφιων Συστατικών Βασισμένα σε Εξαρτήσεις

Ο ευκολότερος τρόπος για να περιγράψουμε τον τρόπο δημιουργίας των «υποψήφιων συστατικών» είναι μέσω των μονοπατιών ενός γράφου. Μετά την εκτέλεση των πρώτων πέντε βημάτων του εργαλείου έχουμε δημιουργήσει μία λίστα από κλάσεις «έγκυρες» που μπορούν να συμμετέχουν στην δημιουργία νέων «υποψήφιων συστατικών». Για κάθε μία από αυτές τις κλάσεις υπάρχουν και οι εξαρτήσεις τους καθώς και τα ποιοτικά τους στοιχεία. Επομένως έχουμε δημιουργήσει έναν γράφο που περιγράφει το σύστημα έχοντας σαν κόμβους τα ονόματα των κλάσεων και σαν κορυφές τις εξαρτήσεις μεταξύ τους. Ένα παράδειγμα τέτοιου γράφου για ολόκληρο το σύστημα του παιχνιδιού ανοικτού λογισμικού “ScotlandYard” παρουσιάζεται παρακάτω.



Σχήμα 20. Γράφος Εξαρτήσεων παιχνιδιού "Scotland Yard"

Η προτεινόμενη μεθοδολογία εφαρμόζεται για κάθε κλάση του γράφου (συστήματος) και παράγει ένα σύνολο από μοναδικά «υποψήφια συστατικά λογισμικού». Για κάθε κόμβο (κλάση) εφαρμόζονται τα παρακάτω βήματα.

Βήμα 1ο. Δημιουργούμε τη δομή δεδομένων (σύνολο) βάση της οποίας περιορίζουμε τα παραγόμενα «υποψήφια συστατικά». Η δομή αυτή είναι ένας δισδιάστατος πίνακας δυναμικά μεταβαλλόμενος. Το πλήθος των γραμμών του πίνακα ορίζει τον μέγιστο αριθμό των κλάσεων που μπορούν να συμμετέχουν σε ένα «υποψήφιο συστατικό», ενώ ο αριθμός των στηλών ορίζει το πλήθος των «υποψήφιων συστατικών» που επιτρέπεται να δημιουργηθούν για κάθε αριθμό κλάσεων. Κατασκευάζουμε το πρώτο «υποψήφιο συστατικό» από τον κόμβο (κλάση) που ξεκινήσαμε την διαδικασία.

Βήμα 2ο. Βρίσκουμε τις εξαρτήσεις της κλάσης που ορίζει το αρχικό «υποψήφιο συστατικό». Εάν το υποψήφιο συστατικό έχει περισσότερους, του ενός, συμμετέχοντες τότε θεωρούμε το συστατικό αυτό ως αυτόνομο σύστημα

και βρίσκουμε τις εξαρτήσεις που είναι οι εξωτερικές κλάσεις στις οποίες αναφέρεται αυτό.

Βήμα 3ο. Τοποθετούμε τις εξαρτήσεις που βρέθηκαν στο βήμα 2 σε μία λίστα, βάση του αριθμού των εξαρτήσεών τους, με αύξουσα σειρά. Δηλαδή προτεραιότητα θα έχουν αυτές με τις λιγότερες εξαρτήσεις (Fan Out).

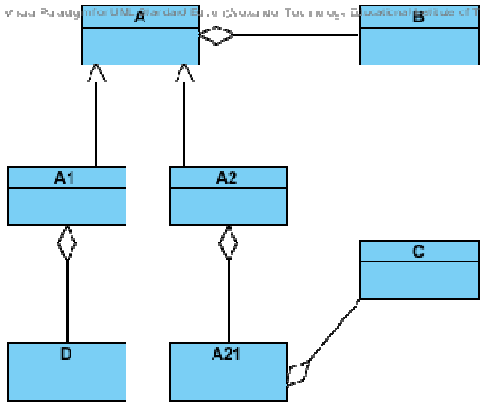
Βήμα 4ο. Για κάθε εξάρτηση της λίστας εξαρτήσεων δημιουργούμε ένα καινούριο συστατικό μεγαλύτερο κατά ένα στο πλήθος κλάσεων από το πατρικό του και το τοποθετούμε στην λίστα με τα υπόλοιπα «υποψήφια συστατικά» αφού ελέγξουμε ότι δεν υπάρχει ήδη. Στη συνέχεια αυξάνουμε τον αριθμό στο κατάλληλο κελί της δομής που περιορίζει την παραγωγή δεδομένων. Τα «υποψήφια συστατικά» που παράγονται σε αυτό το βήμα θεωρούνται αποθηκεύονται σε μία λίστα ως παιδιά του «υποψήφιου συστατικού» από το οποίο προήλθαν.

Βήμα 5ο. Επιστρέφουμε στο βήμα 2 για κάθε υποψήφια συστατικό που δημιουργήθηκε στο βήμα 3 και αποθηκεύτηκε στην λίστα με τα «υποψήφια συστατικά» με τη σειρά που αυτά αποθηκεύτηκαν. Σε περίπτωση που υπερβούμε τους περιορισμούς που θέτει η δομή ελέγχου τότε σταματάμε την εκτέλεση του διαδικασίας.

Για την καλύτερη κατανόηση και λεπτομερέστερη περιγραφή της διαδικασίας θα εφαρμόσουμε τα παραπάνω βήματα σε ένα δοκιμαστικό project με όνομα "Example Project".

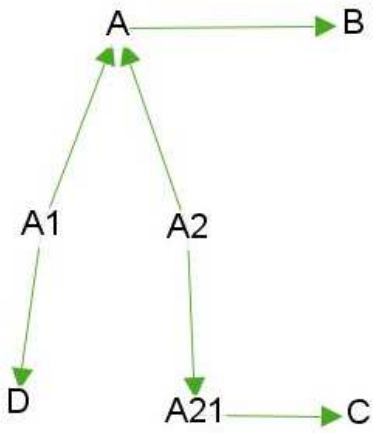
2.1.1. Παραδείγματα

Θα μελετήσουμε τα βήματα της δημιουργίας "υποψήφιων συστατικών" θεωρώντας αρχικά ως κόμβο εκκίνησης στο γράφο μας την κλάση A του "Example Project". Το διάγραμμα κλάσης του “Example Project”.



Σχήμα 21. Διάγραμμα κλάσεων του "Example Project"

Για να γίνουν ποιο κατανοητές οι εξαρτήσεις μεταξύ των κλάσεων του “ Example Project” παραθέτουμε και τον γράφο με τις κλάσεις ως κόμβους και τις εξαρτήσεις με βέλη.



Σχήμα 22. Γράφος του "Example Project"

Στο πρώτο βήμα θα δημιουργηθεί το "υποψήφιο συστατικό" μεγέθους 1 που αποτελείται από την κλάση εκκίνησης.

Πίνακας 1. Δημιουργία «Υποψήφιων Συστατικών» με κόμβο εκκίνησης την κλάση A (Πέρασμα 1ο)

Μέγεθος 1	A		
Μέγεθος 2			

Επειδή η κλάση A έχει μόνο μία εξάρτηση (κλάση B) πως στο δεύτερο βήμα θα δημιουργηθεί ένα μόνο "υποψήφιο συστατικό" μεγέθους 2 το οποίο θα αποτελείται από τις κλάσεις A και B.

Πίνακας 2. Δημιουργία «Υποψήφιων Συστατικών» με κόμβο εκκίνησης την κλάση Α (Πέρασμα 2ο)

Μέγεθος 1	A		
Μέγεθος 2	A,B		

Είναι προφανές πως επειδή καμία από τις κλάσεις Α και Β δεν έχει άλλη εξάρτηση η ο κύκλος της δημιουργίας υποψήφιων συστατικών με κόμβο εκκίνησης την κλάση Α ολοκληρώνεται εδώ. Για να γίνουν πιο ξεκάθαρα τα βήματα «περάσματα» δημιουργίας «υποψήφιων συστατικών» θα παρουσιάσουμε μία πιο σύνθετη περίπτωση, αυτή με κόμβο εκκίνησης την κλάση Α1. Όπως και στην πρώτη περίπτωση το αρχικό «υποψήφιο συστατικό» που θα δημιουργηθεί είναι αυτό που αποτελείται μόνο από τον κόμβο (κλάση) εκκίνησης, στην περίπτωσή μας η Α1.

Πίνακας 3. Δημιουργία «Υποψήφιων Συστατικών» με κόμβο εκκίνησης την κλάση Α1 (Πέρασμα 1ο)

Μέγεθος 1	A1			

Η κλάση Α2 έχει ως εξαρτήσεις (Fan Out classes) τις κλάσεις Α1 και D. Επομένως από αυτές θα φτιάξει τα επόμενα «υποψήφια συστατικά» μεγέθους 2. Το πρώτο που θα δημιουργήσει θα είναι το A1,D διότι η κλάση D έχει Fan Out μηδέν, μικρότερο από την κλάση Α που έχει ένα.

Πίνακας 4. Δημιουργία «Υποψήφιων Συστατικών» με κόμβο εκκίνησης την κλάση Α1 (Πέρασμα 2ο)

Μέγεθος 1	A1			
Μέγεθος 2	A1,D	A,A1		

Όπως αναφέρθηκε και στην μεθοδολογία η διαδικασία γίνεται αναδρομικά ή αλλιώς τα μονοπάτια του γράφου δημιουργούνται κατά βάθος, επομένως στο 3^{ο} πέρασμα θα θεωρηθεί ως κόμβος εκκίνησης ο συνδυασμός των κλάσεων A1,D. Αυτός ο συνδυασμός κλάσεων («υποψήφιο συστατικό») έχει ως εξάρτηση την κλάση Α με αποτέλεσμα στο συγκεκριμένο πέρασμα να δημιουργηθεί μόνο το υποψήφιο συστατικό A,A1,D.

Πίνακας 5. Δημιουργία «Υποψήφιων Συστατικών» με κόμβο εκκίνησης την κλάση Α1 (Πέρασμα 3ο)

Μέγεθος 1	A1			
Μέγεθος 2	A1,D	A,A1		
Μέγεθος 3	A,A1,D			

Ακριβώς όπως και παραπάνω ο κόμβος εκκίνησης τώρα θα θεωρείται ο συνδυασμός A,A1,D και βάση των εξαρτήσεων αυτού (κλάση B), θα δημιουργηθούν τα επόμενα «υποψήφια συστατικά» μεγέθους 4.

Πίνακας 6. Δημιουργία «Υποψήφιων Συστατικών» με κόμβο εκκίνησης την κλάση A1 (Πέρασμα 4ο)

Μέγεθος 1	A1			
Μέγεθος 2	A1,D	A,A1		
Μέγεθος 3	A,A1,D			
Μέγεθος 4	A,A1,B,D			

Επειδή ο συνδυασμός A,A1,B,D δεν έχει άλλη εξάρτηση τώρα θα θεωρηθεί ως κόμβος εκκίνησης ο συνδυασμός του αμέσως μικρότερου μεγέθους ο οποίος δεν έχει ελεγχθεί. Στην περίπτωσή μας επειδή δεν υπάρχει μη ελεγμένος συνδυασμός κλάσεων μεγέθους 3 θα πάμε ακόμα ένα βήμα πίσω και θα ελέγξουμε τους συνδυασμούς μεγέθους 2. Μη ελεγμένος υπάρχει μόνο ο συνδυασμός A,A1 ο οποίος έχει εξαρτήσεις τις κλάσεις B και D. Τα νέα υποψήφια συστατικά μεγέθους 3 που θα δημιουργηθούν είναι τα A,A1,B και A,A1,D με το τελευταίο να μην προστίθεται στη λίστα αφού ήδη υπάρχει. Επομένως η λίστα θα είναι η εξής:

Πίνακας 7. Δημιουργία «Υποψήφιων Συστατικών» με κόμβο εκκίνησης την κλάση A1 (Πέρασμα 5ο)

Μέγεθος 1	A1			
Μέγεθος 2	A1,D	A,A1		
Μέγεθος 3	A,A1,D	A,A1,B		
Μέγεθος 4	A,A1,B,D			

Όπως φαίνεται και από τον τελευταίο πίνακα δεν υπάρχει άλλο μονοπάτι στον γράφο που να ξεκινάει από την κλάση A1 και είναι μοναδικό. Επομένως η δημιουργία «υποψήφιων συστατικών» με κόμβο εκκίνησης την κλάση A1 ολοκληρώθηκε με την δημιουργία έξι «υποψήφιων συστατικών» με τα παρακάτω χαρακτηριστικά.

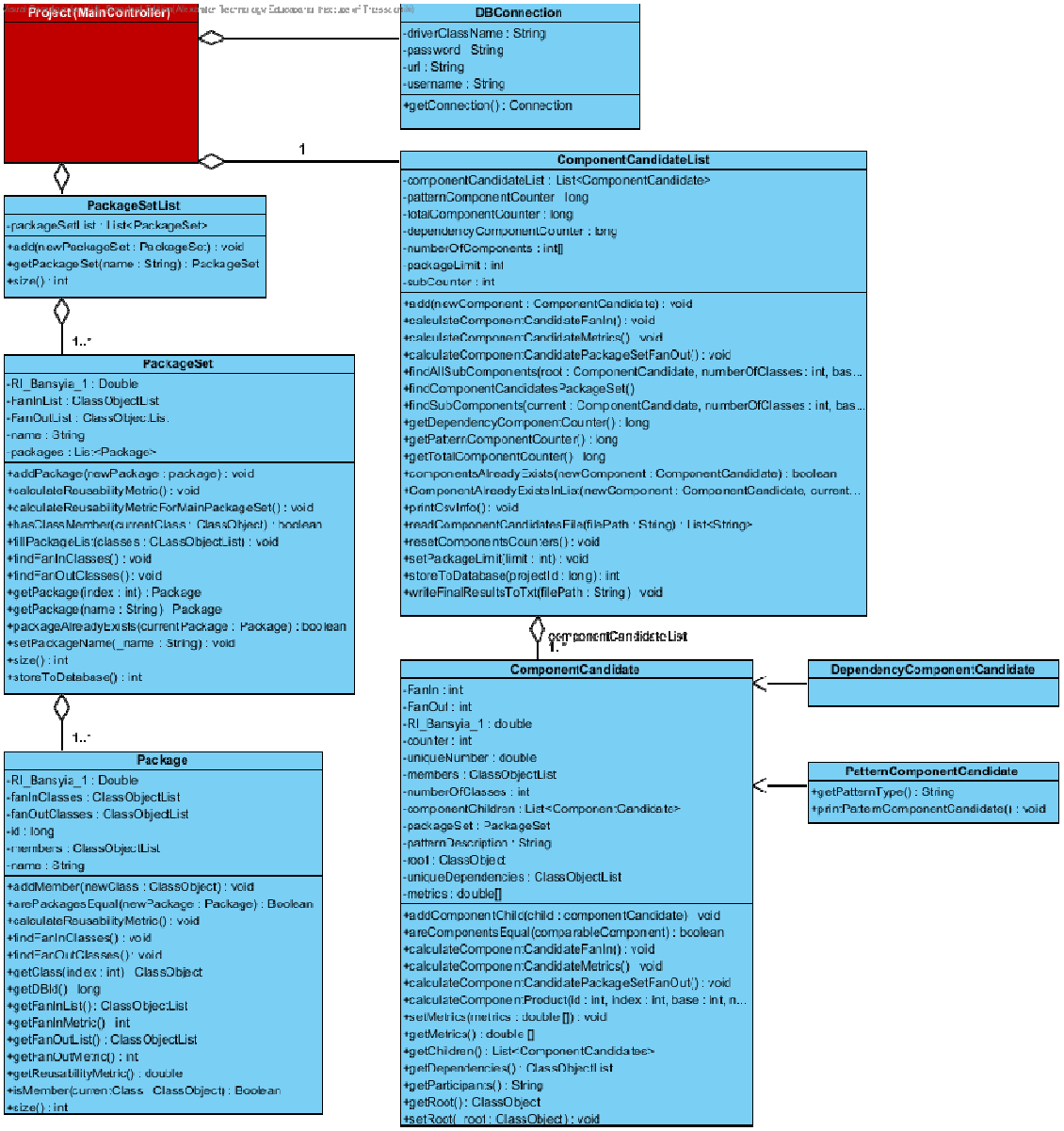
Πίνακας 8. Υποψήφια Συστατικά

Υποψήφιο Συστατικό	**Εξωτερικές Εξαρτήσεις (Fan Out)**	**Μέγεθος**
A1	2	1
A1, A	2	2

A1, D	1	2
A1, A, D	1	3
A1, A, B	1	3
A1, A, B, D	0	4

2.1.2. Διάγραμμα Κλάσεων

Παρουσιάζεται το διάγραμμα κλάσεων του εργαλείου που εκτελεί την διαδικασία παραγωγής «υποψήφιων συστατικών» καθώς και μία σύντομη περιγραφή για τις λειτουργίες της κάθε κλάσης του.



Σχήμα 23. Διάγραμμα κλάσεων εργαλείου Component Candidate Creator

- **ComponentCandidate** – Είναι μία κλάση η οποία αναπαριστά ένα «υποψήφιο συστατικό» ορίζοντας όλες τις απαραίτητες ιδιότητές του, όπως οι μετρικές, οι εξαρτήσεις, η κλάση «ρίζα», τις συμμετέχουσες κλάσεις και τους μοναδικούς χαρακτηριστικούς αριθμούς αναγνώρισής του. Ορίζει επίσης βασικές λειτουργίες όπως η αρχικοποίηση του συστατικού, τη σύγκριση ομοιότητας μεταξύ δύο συστατικών, τον υπολογισμό των μετρικών τους καθώς και όλες τις μεθόδους που προσφέρουν την δυνατότητα μεταβολής και ανάκτησης των ιδιοτήτων της.
- **DependencyComponentCandidate** – Είναι η επέκταση της κλάσης “ComponentCandidate” και όπως φαίνεται από το διάγραμμα κλάσεων δεν προσφέρει κάποια παραπάνω λειτουργία σε σχέση με την πατρική της κλάση. Ο λόγος ύπαρξής της είναι η διαφοροποίηση των «υποψήφιων συστατικών» που βασίζονται σε εξαρτήσεις από αυτά που βασίζονται σε πρότυπα σχεδίασης.
- **PatternComponentCandidate** – Είναι η επέκταση της κλάσης “ComponentCandidate” και προσθέτει την ιδιότητα του τύπου του προτύπου σχεδίασης που είναι απαραίτητο κατά την εγγραφή των «υποψήφιων συστατικών» στη βάση δεδομένων.
- **ComponentCandidateList** – Είναι η κλάση που αρχικοποιεί και διατηρεί μία λίστα από αντικείμενα (υποψήφια συστατικά) και είναι υπεύθυνη για την δημιουργία τους και την διαχείρισή τους. Η κλάση αυτή δημιουργεί και προσθέτει «υποψήφια συστατικά» βάση του αναδρομικού αλγορίθμου που περιγράφεται αναλυτικά στη μεθοδολογία. Τέλος προσφέρει και τις απαραίτητες λειτουργίες για την εγγραφή των υποψήφιων συστατικών στη βάση δεδομένων.
- **Package** – Αναπαριστά ένα πακέτο κλάσεων όπως αυτό περιγράφεται από τη γλώσσα προγραμματισμού java. Δηλαδή διατηρεί μία λίστα από κλάσεις που ανήκουν σε αυτό το πακέτο καθώς και από μία λίστα για τις εξαρτήσεις κλάσεων από και προς το πακέτο. Η κλάση πακέτο αναπαριστά ένα σύνολο κλάσεων και αντίστοιχα με τα «υποψήφια συστατικά» διατηρεί και υπολογίζει τις κατάλληλες τιμές για τις μετρικές που το χαρακτηρίζουν.
- **PackageSet** – Αναπαριστά ένα σύνολο πακέτων. Αυτό το σύνολο έχει ύπαρξη μόνο εάν τα μέλη ενός «υποψήφιου συστατικού» ανήκουν σε περισσότερα του ενός πακέτα. Εκτελεί ελέγχους για την εύρεση και αφαίρεση των

διπλοεγγεγραμμένων κλάσεων μελών τους ώστε οι τιμές στις μετρικές τους (όπως στα πακέτα και τα «υποψήφια συστατικά») να είναι έγκυρες. Επίσης προσφέρει και την δυνατότητα προσθήκης των πακέτων που το απαρτίζουν στη βάση δεδομένων.

- **PackageSetList** – Διατηρεί μία λίστα από τα σύνολα των πακέτων (packageSet) έτσι ώστε σύνολα πακέτων να δημιουργούνται μόνο μία φορά και να επιταχύνεται η εκτέλεση της εφαρμογής κατά τον υπολογισμό των απαραίτητων μετρικών για κάθε πακέτο (package) και σύνολο πακέτων (packageSet).
- **DBConnection** – Προσφέρει την δυνατότητα της αρχικοποίησης μία σύνδεσης στην βάση δεδομένων η οποία είναι κοινή από όλη την εφαρμογή.

2.1.3. Αντιμετώπιση Προβλημάτων και Βελτιστοποίηση Κώδικα

Η υλοποίηση του αλγορίθμου κατά τη συγγραφή του κώδικα του εργαλείου ήταν μιας συνεχής σχεδιαστική και προγραμματιστική πρόκληση. Αρχικά αξίζει να αναφερθούμε στην πρώτη προσέγγιση για την υλοποίηση του εργαλείου που έγινε σε γλώσσα PHP. Η συγκεκριμένη γλώσσα επιλέχθηκε με σκοπό το εργαλείο να γίνει μία διαδικτυακή εφαρμογή πραγματικού χρόνου (real-time online service) με δυνατότητα σύνδεσης πολλών χρηστών και ανάλυση οποιουδήποτε project κατά απαίτηση. Η ανάλυση γίνονταν βάση του πηγαίου κώδικα (source code) του project και περιελάμβανε τα παρακάτω βήματα:

1. Ανέβασμα (upload) ενός συμπιεσμένου αρχείου στον εξυπηρετητή, που περιείχε το κώδικα του project.
2. Αποσυμπίεση του συμπιεσμένου αρχείου και καταγραφή σε μια λίστα όλων των ονομάτων των αρχείων που περιείχαν πηγαίο κώδικα σε Java.
3. Άνοιγμα όλων των αρχείων της λίστας και καταγραφή των εξαρτήσεων τους προς άλλες κλάσεις, σε ένα εικονικό γράφο, βάση των ονομάτων της ίδιας λίστας.
4. Επιλογή μιας κλάσης ως ρίζα και εύρεση όλων των πιθανών συνδυασμών, βασισμένα στις συνδέσεις του γράφου, που αργότερα ονομάστηκαν υποψήφια συστατικά (component candidates).

Όπως αναφέραμε και παραπάνω το εργαλείο θα λειτουργούσε ως μια εφαρμογή διαδικτύου με προοπτική να γίνει εφαρμογή πραγματικού χρόνου. Βασικό εμπόδια σε αυτό το στόχο στάθηκε ο χρόνος που χρειάζονταν για το ανέβασμα ενός project στον εξυπηρετητή και τα όρια χρόνου που ορίζει ο κάθε διαχειριστής. Η διαδικασία αυτή εξαρτάται από πολλούς παράγοντες, όπως η ποιότητα της σύνδεσης στο ίντερνετ και ο φόρτος του εκάστοτε εξυπηρετητή, καθιστώντας τον χρόνο ανεβάσματος (upload time) συνήθως απαγορευτικό για εφαρμογή πραγματικού χρόνου. Επίσης, πέραν του Βήματος 2, όλες οι υπόλοιπες διεργασίες απαιτούν μεγάλο ποσοστό από τους πόρους του συστήματος και τις περισσότερες φορές για αρκετό χρονικό διάστημα λόγω των πολύπλοκων ελέγχων που πραγματοποιούνται. Για παράδειγμα, το project apache ant με 1800 κλάσεις χρειάζονταν πάνω από 3 gb μνήμης για αρκετές ώρες για να εκτελεστεί. Ο αλγόριθμος για το πρόβλημά μας δεν μας επέτρεπε να χρησιμοποιήσουμε πολυνηματικό προγραμματισμό οπότε οι επεξεργαστές τελευταίας τεχνολογίας με πολλούς πυρήνες επεξεργασίας έμεναν στην ουσία ανεκμετάλλευτοι.

Πέρα από τα θέματα απόδοσης, κύριο πρόβλημα που μας οδήγησε να βρούμε άλλο τρόπο υλοποίησης ήταν οι αρκετές περιπτώσεις στις οποίες στο ίδιο project υπήρχαν κλάσεις με το ίδιο όνομα και ενώ ήταν σε διαφορετικό πακέτο ήταν αδύνατο για τον αλγόριθμο να τις ξεχωρίσει. Το αποτέλεσμα ήταν σε κάποιες περιπτώσεις να δημιουργούνται εξαρτήσεις που στην πραγματικότητα δεν υπήρχαν και σε άλλες περιπτώσεις να εξαιρούνται εξαρτήσεις διότι στην λίστα υπάρχει ήδη το όνομα της συγκεκριμένης κλάσης ενώ στην πραγματικότητα είναι άλλη. Επομένως υπήρχαν αρκετές περιπτώσεις που ο τελικός γράφος (λίστα με εξαρτήσεις) δεν ήταν έγκυρος.

Εξαιτίας των παραπάνω και λαμβάνοντας υπόψη πως το εργαλεία που βρίσκει μετρικές του κώδικα και το εργαλείο που ανακτά πρότυπα σχεδίασης από τον κώδικα είναι γραμμένα σε Java πάρθηκε η απόφαση να υλοποιηθεί η εφαρμογή εξολοκλήρου σε Java με σκοπό να ενιαία διαχείριση των ενδιάμεσα παραγόμενων αρχείων από το κάθε εργαλείο. Τα προβλήματα που εμφανίστηκαν και αντιμετωπίστηκαν κατά την διάρκεια της ανάπτυξης μπορούν να χωριστούν σε:

- Προβλήματα απόδοσης
- Προβλήματα που οφείλονται στην ιδιαιτερότητα των δεδομένων

2.1.3.1. Προβλήματα απόδοσης

Τα προβλήματα απόδοσης οφείλονταν κυρίως στον τεράστιο όγκο πληροφοριών που έπρεπε να διαχειριστούμε καθώς και στους υπεράριθμους ελέγχους και υπολογισμούς που συμβαίνουν κατά την εκτέλεση της ανάλυσης από το εργαλείο. Όπως περιγράφεται και στην μεθοδολογία η παραγωγή των υποψήφιων συστατικών βασίζεται στον γράφο που αποτελείται από τις κλάσεις του υποψήφιου Project. Είναι φανερό πως οι διαφορετικοί τρόποι για να διατρέξουμε τον γράφο με τις παραμέτρους του αλγορίθμου είναι πάρα πολλοί. Ποικίλουν από μερικές εκατοντάδες μέχρι και δεκάδες χιλιάδες. Για κάθε πιθανή διαδρομή από αυτές δημιουργείται και μία λίστα από κόμβους που θα αποτελούν το «υποψήφιο συστατικό λογισμικού». Θέλουμε όμως κάθε «υποψήφιο συστατικό λογισμικού» να είναι μοναδικό οπότε πρέπει να το συγκρίνουμε με όλα τα μέχρι εκείνη στιγμή «υποψήφια συστατικά» που έχουμε ήδη δημιουργήσει. Για να γίνει αυτός ο έλεγχος πρέπει να συγκρίνουμε την λίστα με τις κλάσεις που συμμετέχουν στο εκάστοτε «υποψήφιο συστατικό» με την αντίστοιχη λίστα κάθε «υποψήφιου συστατικού» που υπάρχει ήδη αποθηκευμένο. Είναι φανερό πως η διαδικασία αυτή είναι χρονοβόρα κα απαιτεί τεράστια υπολογιστική ισχύ. Για το λόγο αυτό χρησιμοποιήσαμε κάποιες τεχνικές που θα μείωναν την πολυπλοκότητα καθώς και τις απαιτήσεις σε υπολογιστική ισχύ. Τα σημεία που χρειάστηκε να βελτιωθούν ήταν αυτά της

- σύγκρισης δύο «υποψήφιων συστατικών»,
- μείωσης του αριθμού των συγκρίσεων του νέου συστατικού με τα ήδη υπάρχοντα αποθηκευμένα και τέλος
- ο περιορισμός του αριθμού των παραγόμενων «υποψήφιων συστατικών»

<u>Για την βελτιστοποίηση της σύγκρισης</u> δύο «υποψήφιων συστατικών» αρχικά μετά το τέλος της δημιουργίας της λίστας με τις «έγκυρες» κλάσεις του project ορίζετε ένας μοναδικός αριθμός (classID) για κάθε κλάση. Κατά τη δημιουργία ενός νέου «υποψήφιου συστατικού» η προσθήκη των κλάσεων στην λίστα με τις κλάσεις που συμμετέχουν σε αυτό γίνετε με ταξινομημένα βάση του ονόματός τους. Στη συνέχεια οι κλάσεις που συμμετέχουν στο «υποψήφιο συστατικό» χωρίζονται ανά πέντε και για

κάθε πεντάδα δημιουργείται ένας αριθμός «παράγωγο (product)» βάση του αλγορίθμου του οποίου ο κώδικας παρουσιάζεται παρακάτω όπου:

- *newMembers* είναι η λίστα με τις κλάσεις που θα συμμετέχουν στο νέο «υποψήφιο συστατικό».
- *base* είναι ο αριθμός «βάση» που δηλώνει το σύνολο των κλάσεων του εκάστοτε υπο-ανάλυση project.

```
public ComponentCandidate(ClassObjectList newMembers, int base) {
        this.firstUniqueNumber = 0;
        this.secondUniqueNumber = 0;
        this.thirdUniqueNumber = 0;
        this.fourthUniqueNumber = 0;
        this.fifthUniqueNumber = 0;
        this.sixthUniqueNumber = 0;
        this.seventhUniqueNumber = 0;
        this.eightthUniqueNumber = 0;
        this.ninthUniqueNumber = 0;

        for (int i = 0; i < newMembers.size(); i++) {
            ClassObject currentNewMember = newMembers.get(i);

            if (i < 5) {
                firstUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        i,
                        base,
                        newMembers.size());
            } else if (i < 10) {
                secondUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        (i - 5),
                        base,
                        newMembers.size());
            } else if (i < 15) {
                thirdUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        (i - 10),
                        base,
                        newMembers.size());
            } else if (i < 20) {
                fourthUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        (i - 15),
                        base,
                        newMembers.size());
```

```
            } else if (i < 25) {
                fifthUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        (i - 20),
                        base,
                        newMembers.size());
            } else if (i < 30) {
                sixthUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        (i - 25),
                        base,
                        newMembers.size());
            } else if (i < 35) {
                seventhUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        (i - 30),
                        base,
                        newMembers.size());
            } else if (i < 40) {
                eightthUniqueNumber += calculatePackageProduct(
                        currentNewMember.getClassObjectId(),
                        (i - 35), //correction...
                        base,
                        newMembers.size());
            }
        }

public static double calculatePackageProduct(int id, int index, int base, int
nocInPackage) {
        double product = (double) id / Math.pow(base, nocInPackage);
        return product * Math.pow(base, index);
}
```

Ο έλεγχος για την περίπτωση που το «υποψήφιο συστατικό» υπάρχει ήδη στη λίστα μας γίνεται πλέον μόνο βάση των μοναδικών αριθμών του κάθε «υποψήφιου συστατικού» με την κλήση του παρακάτω κώδικα.

```
public boolean packageAlreadyExistsInList(ComponentCandidate newComponent,
ArrayList<ComponentCandidate> candidatesList) {
      for (int i = 0; i < currentList.size(); i++) {
            ComponentCandidate currentPackage = currentList.get(i);
            if (currentComponent.areComponentsEqual(newComponent)) {
                  return true;
            }
      }
      return false;
}
```

```
public boolean areComponentsEqual(ComponentCandidate newComponent) {
      return (firstUniqueNumber == newComponent.firstUniqueNumber
            && secondUniqueNumber == newComponent.secondUniqueNumber
            && thirdUniqueNumber == newComponent.thirdUniqueNumber
            && fourthUniqueNumber == newComponent.fourthUniqueNumber
            && fifthUniqueNumber == newComponent.fifthUniqueNumber
            && sixthUniqueNumber == newComponent.sixthUniqueNumber
            && seventhUniqueNumber == newComponent.seventhUniqueNumber
            && eightthUniqueNumber == newComponent.eightthUniqueNumber);
}
```

Είναι προφανές πως από τη στιγμή που οι συμμετέχουσες κλάσεις είναι ταξινομημένες βάση του ονόματός τους και ο χαρακτηριστικός αριθμός της κάθε κλάσης είναι μοναδικός πως για κάθε «υποψήφιο συστατικό» η ακολουθία των αριθμών firstUniqueNumber, secondUniqueNumber, thirdUniqueNumber, fourthUniqueNumber, fifthUniqueNumber, sixthUniqueNumber, seventhUniqueNumber, eightthUniqueNumber είναι μοναδικός. Ίδια μπορεί να είναι μόνο στην περίπτωση που δύο «υποψήφια συστατικά» έχουν τον ίδιο αριθμό κλάσεων και αυτές είναι ίδιες μία προς μία, ανεξάρτητα από την σειρά με την οποία προστέθηκαν στη λίστα των κλάσεων του «υποψήφιου συστατικού». Το κέρδος στην απόδοση του εργαλείου οφείλεται στον χρόνο που το σύστημα χρειάζεται για την σύγκριση δύο μεταβλητών. ο αριθμός των συμμετεχουσών κλάσεων ποικίλει από μία έως και σαράντα. Είναι φανερό πως σε «υποψήφια συστατικά» με περισσότερες από οκτώ κλάσεις ο χρόνος σύγκρισης είναι πολύ μεγαλύτερος από τον χρόνο σύγκρισης των οκτώ μοναδικών αριθμών. Ένας λόγος που ενισχύει την ανάγκη χρήσης της συγκεκριμένης τεχνικής είναι και η σημαντική διαφορά στον χρόνο μεταξύ της σύγκρισης δύο αριθμών και της σύγκρισης δύο αλφαριθμητικών, με την πρώτη σύγκριση να γίνεται σαφώς γρηγορότερα καταναλώνοντας παράλληλα λιγότερους πόρους από το σύστημα. Μία ιδέα που φάνηκε εξαιρετικά αποδοτική αλλά τελικά εγκαταλείφτηκε ως μη εφαρμόσιμη στο σύστημά μας ήταν η διατήρηση ενός πίνακα με τους αριθμούς κατακερματισμού (hash set) των αντικειμένων («υποψήφιων συστατικών»). Η τεχνική αυτή δεν εφαρμόζεται στην περίπτωσή μας διότι κάθε φορά τα «υποψήφια συστατικά» δημιουργούνται εκ νέου δημιουργώντας νέο μοναδικό αριθμό κατακερματισμού με αποτέλεσμα την ύπαρξη δύο ίδιων «υποψήφιων συστατικών» με διαφορετικούς μοναδικούς αριθμούς και διπλό εγγραφές.

Για την μείωση των ελέγχων στη λίστα των «υποψήφιων συστατικών» χρησιμοποιήσαμε επίσης μία ιεραρχία λιστών. Αρχικά παρατηρήθηκε πως μετά το 30% της εκτέλεσης του εργαλείου δημιουργίας νέων «υποψήφιων συστατικών» η απόδοση του συστήματος μειώνονταν σημαντικά καθώς και πως η μείωση της απόδοσης είχε αυξητικό ρυθμό. Αυτό συνέβαινε διότι αυξάνονταν συνεχώς ο αριθμός των αποθηκευμένων «υποψήφιων συστατικών» οπότε κάθε νέο «υποψήφιο συστατικό» έπρεπε να συγκριθεί με όλα τα προηγούμενα όταν αυτά ήταν αποθηκευμένα μέσα σε μία μονοδιάστατη λίστα. Η μονοδιάστατη λίστα με τα αποθηκευμένα «υποψήφια συστατικά» είχε την εξής μορφή:

Πίνακας 9. Απεικόνιση μονοδιάστατης λίστας για την αποθήκευση των υποψήφιων συστατικών

List<ComponentCandidate>	Candidate 1 Candidate 2 Candidate 3 …. Candidate n

Έτσι αποφασίστηκε η χρήση ενός ακόμα επιπέδου λιστών με χαρακτηριστικό τον αριθμό των κλάσεων του κάθε «υποψήφιου συστατικού». Με αυτό τον τρόπο ένα νέο «υποψήφιο συστατικό» συγκρίνεται μόνο με τα ήδη υπάρχοντα «υποψήφια συστατικά» που έχουν τον ίδιο αριθμό κλάσεων με αυτό. Το αποτέλεσμα είναι η σημαντική μείωση του αριθμού των συγκρίσεων και κατ επέκταση του συνολικού χρόνου που διαρκούν οι συγκρίσεις. Η δισδιάστατη λίστα με τα αποθηκευμένα «υποψήφια συστατικά» είχε την εξής μορφή:

Πίνακας 10. Απεικόνιση δισδιάστατης λίστας για την αποθήκευση των υποψήφιων συστατικών

List<List>	List<ComponentCandidate> (size 1)	Candidate 1 Candidate 2 …. Candidate n
	List<ComponentCandidate> (size 2)	Candidate 1 Candidate 2 …. Candidate n
	List<ComponentCandidate> (size 3)	Candidate 1 Candidate 2 ….

		Candidate n
	…..	…..
	List<ComponentCandidate> (size 40)	Candidate 1 Candidate 2 …. Candidate n

Το αποτέλεσμα ήταν η περεταίρω μείωση του αριθμού των ελέγχων και του συνολικού χρόνου. Παρόλη τη βελτίωση της απόδοσης κρίθηκε αναγκαίο να προσθέσουμε ακόμη δύο επίπεδα λιστών ώστε ο χρόνος εκτέλεση του εργαλείου να μειωθεί στα επίπεδα μιας εφαρμογής πραγματικού χρόνου. Το μήκος των δύο νέων λιστών ισούται με τον αριθμό των χαρακτήρων της κλάσης με το μεγαλύτερο πλήθος χαρακτήρων. Για την απεικόνιση των πινάκων θα θεωρήσουμε πως το μέγιστο πλήθος χαρακτήρων σε μία κλάση είναι 30 και θα το ονομάσουμε *length*. πχ (στην *package.subpackage.Test είναι 23*).

Πίνακας 11. Απεικόνιση τετραδιάστατης λίστας για την αποθήκευση των υποψήφιων συστατικών

List<List<List<List>>>	List<List<List>> (size 1)	List<List > (length 1)	List<ComponentCandidate> (length 1)	Candidate 1 …. Candidate n
			List<ComponentCandidate> (length 2)	Candidate 1 …. Candidate n
			⋮	⋮
			List<ComponentCandidate> (length 30)	Candidate 1 …. Candidate n
		⋮	⋮	⋮
		List<List > (length 30)	List<ComponentCandidate> (length 1)	Candidate 1 …. Candidate n
			List<ComponentCandidate> (length 2)	Candidate 1 …. Candidate n
			⋮	⋮
			List<ComponentCandidate> (length 30)	Candidate 1 …. Candidate n
	⋮	⋮	⋮	⋮
	List<List<List>> (size 40)	List<List > (length 1)	List<ComponentCandidate> (length 1)	Candidate 1 …. Candidate n
			List<ComponentCandidate> (length 2)	Candidate 1 …. Candidate n
			⋮	⋮
			List<ComponentCandidate> (length 30)	Candidate 1 …. Candidate n
		⋮	⋮	⋮
		List<List > (length 30)	List<ComponentCandidate> (length 1)	Candidate 1 …. Candidate n
			List<ComponentCandidate> (length 2)	Candidate 1 …. Candidate n
			⋮	⋮
			List<ComponentCandidate> (length 30)	Candidate 1 …. Candidate n

Ακολουθεί το κομμάτι του κώδικα που αρχικοποιεί την ιεραρχία των λιστών.

```
public ComponentCandidateList(int noc, int longestNodeName) {
	long count = 0;
	this.componentCandidateList = new
	ArrayList<ArrayList<ArrayList<ArrayList<ComponentCandidate>>>>();
	this.numberOfComponents = new int[noc + 1];
	for (int i = 0; i <= noc; i++) {
		this.componentCandidateList.add(new
		ArrayList<ArrayList<ArrayList<ComponentCandidate>>>());
		this.numberOfComponents[i] = 0;
	}
	for (int i = 1; i < componentCandidateList.size(); i++) {
		for (int j = 0; j < longestNodeName; j++) {
			this.componentCandidateList.get(i).add(new
			ArrayList<ArrayList<ComponentCandidate>>());
		}
	}
	for (int i = 0; i < componentCandidateList.size(); i++) {
		for (int j = 0; j < componentCandidateList.get(i).size(); j++) {
			for (int k = 0; k < longestNodeName; k++) {
				this.componentCandidateList.get(i).get(j).add(new
				ArrayList<ComponentCandidate>());
			}
		}
	}
}
```

Για να γίνει πιο κατανοητή και μετρήσιμη η βελτιστοποίηση του κώδικα και του αλγορίθμου παρουσιάζουμε σε ένα πίνακα πως με κάθε προσθήκη μιας ιεραρχίας λιστών επηρεάζεται και ο χρόνος εκτέλεσης. Οι συγκριτικές δοκιμές γίνονται στην ανάλυση ενός παιχνιδιού “Super Mario” που αποτελείται από 33 κλάσεις. Ορίζουμε ως όριο την δημιουργία «υποψήφιων συστατικών» μέχρι οκτώ κλάσεις. Το πλήθος των παραγόμενων «υποψήφιων συστατικών» ήταν 453.190.

Πίνακας 12. Σύγκριση χρόνων εκτέλεσης με την προσθήκη ιεραρχιών λιστών

Διαστάσεις λίστας	**x1**	**x2**	**x3**	**x4**	**x4**
Μέθοδος σύγκρισης	Βάση ονομάτων	Βάση ονομάτων	Βάση ονομάτων	Βάση ονομάτων	Βάση αριθμών
Χρόνος Εκτέλεσης	431 min (>7h)	273 min (>4h)	30 min	7 min	1,5 min

Ο περιορισμός του αριθμού των παραγόμενων «υποψήφιων συστατικών» ήταν μία ανάγκη που παρουσιάστηκε όσο βελτιώναμε την απόδοση του εργαλείου και καταφέρναμε να το τρέξουμε για μεγαλύτερο πλήθος κλάσεων. Για ένα project με 33 κλάσεις όπως το παραπάνω παράγονται περισσότερα από 453.000 «υποψήφια συστατικά» με πλήθος κλάσεων από μία έως και οκτώ. Είναι φανερό πως αν θέλαμε να δημιουργήσουμε «υποψήφια συστατικά» με πλήθος κλάσεων μεγαλύτερο από οκτώ τότε ο αριθμός αυτών θα εκτοξεύονταν ακόμη περισσότερο. Πόσο μάλλον για Projects που αποτελούνται από μερικές εκατοντάδες ή και χιλιάδες κλάσεις. Επομένως αποφασίσαμε να περιορίσουμε τον αριθμό των παραγόμενων «υποψήφιων συστατικών» θέτοντας όριο σε κάθε πέρασμα του γράφου στην δημιουργία μέχρι κάποιο συγκεκριμένο αριθμό «υποψήφιων συστατικών» ανά πλήθος κλάσεων. Ο αριθμός αυτός ορίζεται από τον χρήστη κατά την εκτέλεση του εργαλείου. Για παράδειγμα εάν ο χρήστης ορίσει μέγιστο αριθμό δέκα «υποψήφιων συστατικών» ανά πλήθος κλάσεων σε κάθε πέρασμα του γράφου τότε εάν το πέρασμα του γράφου έχει κόμβο εκκίνησης την κλάση Χ θα παραχθούν τα εξής. Δέκα «υποψήφια συστατικά» με πλήθος κλάσεων 1, δέκα με πλήθος κλάσεων 2 κλπ έως και δέκα με πλήθος κλάσεων το μέγιστο που έχει ορίσει ο χρήστης πριν την ανάλυση του κάθε project. Ο αλγόριθμος επιλογής της επόμενης κλάσης για την δημιουργία νέου συστατικού ταξινομεί τις υποψήφιες προς συμμετοχή κλάσεις βάσει του χαμηλότερου Fan Out έτσι ώστε τα "υποψήφια συστατικά" που θα δημιουργηθούν να έχουν την μεγαλύτερη δυνατή αυτονομία.

Με την χρήση των παραπάνω τεχνικών καταφέραμε να εξασφαλίσουμε:

- Την εκτέλεση όλης της ανάλυσης ενός project σε αρκετά ικανοποιητικό χρόνο.
- Την μείωση την κατανάλωση των πόρων του συστήματος,
- και τέλος τη παραγωγή ενός διαχειρίσιμου αριθμού "υποψήφιων συστατικών" με τα καλύτερα δυνατά ποιοτικά χαρακτηριστικά.

2.1.3.2. Προβλήματα που οφείλονται στην ιδιαιτερότητα των δεδομένων

Όπως αναφέρθηκε και παραπάνω το εργαλείο δημιουργίας «Υποψήφιων Συστατικών Λογισμικού» απαιτεί την ύπαρξη του πηγαίου και του μεταγλωττισμένου

κώδικα για κάθε project που θα αναλυθεί. Σε αυτό το σημείο δημιουργούνται τα περισσότερα προβλήματα εξαιτίας της φύσης του ανοιχτού λογισμικού. Μελετώντας τα 3 μεγαλύτερα αποθετήρια κώδικα εφαρμογών ανοιχτού λογισμικού παρατηρήσαμε πως για τα περισσότερα projects υπάρχει ασυμφωνία μεταξύ του πηγαίου κώδικα και του μεταγλωττισμένου κώδικα. Οι ποιο συχνές περιπτώσεις ιδιαιτερότητας των δεδομένων είναι οι εξής:

- Έλλειψη αρχείων πηγαίου κώδικα. Σε αυτή τη περίπτωση, στον παρεχόμενο από τον σχεδιαστή φάκελο με τον πηγαίο κώδικα του project, λείπει μία ή και περισσότερες κλάσεις από αυτές που υπάρχουν μεταγλωττισμένες στο εκτελέσιμο αρχείο.
- Έλλειψη αρχείων μεταγλωττισμένου κώδικα: Είναι η περίπτωση στην οποία λείπει ένα ή και περισσότερα αρχεία μεταγλωττισμένων κλάσεων από το εκτελέσιμο αρχείο (jar) του project που θέλουμε να αναλύσουμε.
- Ασυμφωνία μονοπατιών (file paths) των αρχείων του πηγαίου και του μεταγλωττισμένου κώδικα: Για να γίνει κατανοητή η συγκεκριμένη περίπτωση θα δοθεί ένα παράδειγμα. Θεωρούμε μια κλάση με όνομα “*Test”.* Ο πηγαίος κώδικας της παραπάνω κλάσης (java file) βρίσκεται στο πακέτο *src/package/subpackage* οπότε η τελική της απεικόνιση είναι *package.subpackage.Test.java*. Ο μεταγλωττισμένος κώδικας βρίσκεται στο αρχείο Test.class και βρίσκεται στο πακέτο org/*package/subpackage* οπότε η τελική της απεικόνιση είναι org.*package.subpackage.Test.class.* χωρίς τις καταλήξεις των αρχείων έχουμε τα ονόματα *package.subpackage.Test* για το πηγαίο κώδικα και org.*package.subpackage.Test* για τον μεταγλωττισμένο κώδικα.

Όλες οι παραπάνω περιπτώσεις επηρεάζουν την διεξαγωγή μιας ομαλής ανάλυσης των δεδομένων. Ποιο συγκεκριμένα στην περίπτωση που λείπει κάποιο αρχείο πηγαίου κώδικα υπάρχει πρόβλημα στην εξαγωγή του αντίστοιχου «υποψήφιου συστατικού λογισμικού» αφού δεν υπάρχει καταχωρημένη η διαδρομή του αρχείου (file path) πηγαίου κώδικα στη βάση δεδομένων. Στην περίπτωση που λείπει κάποιο αρχείο μεταγλωττισμένου κώδικα τότε τα προβλήματα που δημιουργούνται είναι ακόμη

μεγαλύτερα. Η λίστα με τις κλάσεις που συμμετέχουν σε ένα project και από την οποία θα δημιουργηθεί ο εικονικός γράφος του συστήματος δημιουργείται από το εργαλείο classycle το οποίο αναλύει τον μεταγλωττισμένο κώδικα του project. Επίσης η εύρεση των μετρικών των κλάσεων που βρίσκονται στην λίστα γίνεται με την χρήση του εργαλείου ckjm το οποίο επίσης αναλύει τον μεταγλωττισμένο κώδικα του project. Οπότε και στην περίπτωση που προσθέταμε τις κλάσεις που λείπουν μέσω του πηγαίου κώδικα τότε αυτές δεν θα είχαν μετρικές οπότε θα ήταν αδύνατη η αξιολόγησή τους και ως επέκταση η αξιολόγηση των «υποψήφιων συστατικών λογισμικού» στα οποία θα συμμετέχουν. Στην τελευταία περίπτωση όπου υπάρχει ασυμφωνία μεταξύ των διαδρομών των αρχείων πηγαίου κώδικα και των αντίστοιχων αρχείων μεταγλωττισμένου κώδικα δεν μπορεί να γίνει η σωστή αντιστοίχηση διότι δεν είμαστε βέβαιοι πως πρόκειται για την ίδια κλάση και όχι για διαφορετικές κλάσεις με ίδιο όνομα. Για παράδειγμα εάν υπάρχουν δύο κλάσεις με όνομα “*Test.class*” στον μεταγλωττισμένο κώδικα στα πακέτα org/package/ & org/subpackage/ ενώ στον φάκελο με τον πηγαίο κώδικα υπάρχει μόνο μία κλάση με τέτοιο όνομα Test.java στο πακέτο org/, δεν μπορούμε να συμπεράνουμε με ποια από τις δύο κλάσεις του μεταγλωττισμένου κώδικα θα κάνουμε την αντιστοίχιση. Όσο δεν μπορεί να γίνει με απόλυτη ασφάλεια αυτή η αντιστοίχιση δεν μπορούμε να είμαστε σίγουροι πως το «υποψήφιο συστατικό λογισμικού» που θα ανακτήσουμε από την βάση δεδομένων θα έχει τον σωστό πηγαίο κώδικα.

Όλες οι παραπάνω περιπτώσεις απειλούσαν την εγκυρότητα των παραγόμενων «συστατικών λογισμικού». Για να μειώσουμε στο ελάχιστο αυτές τις περιπτώσεις αποφασίσαμε στην αντικειμενοστραφή απεικόνιση της κλάσης «αντικείμενο ClassObject» να συμπεριλάβουμε, πέραν των άλλων, τα πεδία “className”, “sourceName” τα οποία αντιπροσωπεύουν τα ονόματα των αρχείων του πηγαίου και του μεταγλωττισμένου κώδικα αντίστοιχα. Μετά την δημιουργία της βασικής λίστας με τα ονόματα των κλάσεων συμπληρώνονται και τα δύο αυτά πεδία μετά από ανάλυση του φακέλου με τα αρχεία πηγαίου κώδικα και του εκτελέσιμου αρχείου που περιλαμβάνει τα αρχεία με τον μεταγλωττισμένο κώδικα. Είναι τα βήματα 5 και 6 («Επικύρωση λίστας των κλάσεων» και «Ανάκτηση Προτύπων Σχεδίασης») της μεθοδολογίας που περιγράφεται στην αρχή του δεύτερου κεφαλαίου. Μετά και από αυτή την ανάλυση όσες

κλάσεις εμφανίσουν κάποια από τις παραπάνω περιπτώσεις λάθους χαρακτηρίζονται ως «μη έγκυρες (invalid)» και στη συνέχεια διαγράφονται από τη λίστα οπότε δεν λαμβάνονται υπόψη στην δημιουργία του γράφου άρα και στην παραγωγή των «υποψήφιων συστατικών λογισμικού». Ακολουθεί παράδειγμα από τον κώδικα.

```
public boolean isInvalid() {
	return (this.hasSourceFile == false || this.hasBinaryFile == false);
}

public void removeInvalidClasses() {
	boolean flag;
	int invalidClassCounter = 0;
	do {
		flag = false;
		for (int i = 0; i < this.classObjectList.size(); i++) {
			if (this.classObjectList.get(i).isInvalid()) {
				this.classObjectList.remove(i);
				flag = true;
				invalidClassCounter++;
				break;
			}
		}
	} while (flag);
	System.err.println(invalidClassCounter + " invalid classes removed!");
}
```

Με αυτό τον τρόπο εξασφαλίζουμε για τα παραγόμενα «συστατικά λογισμικού» τη δυνατότητα της:

- αξιολόγησης,
- ανάκτησης τους από τη βάση δεδομένων,
- τέλος εξασφαλίζουμε την εγκυρότητά τους.

2.2. Μεθοδολογία Ανίχνευσης Υποψήφιων Συστατικών Βασισμένα σε Πρότυπα Σχεδίασης

Στο κεφάλαιο αυτό παρουσιάζονται ορισμένα παραδείγματα για την καλύτερη κατανόηση του τρόπου που ανακτώνται τα πρότυπα σχεδίασης και της δημιουργίας «υποψήφιων συστατικών» από αυτά. Επίσης, γίνεται αναφορά στα προβλήματα που υπήρξαν κατά την διαδικασία αυτή και ο τρόπος αντιμετωπίστηκαν. Τέλος, γίνεται παράθεση ενός ενδεικτικού, για την διαδικασία, μέρους κώδικα.

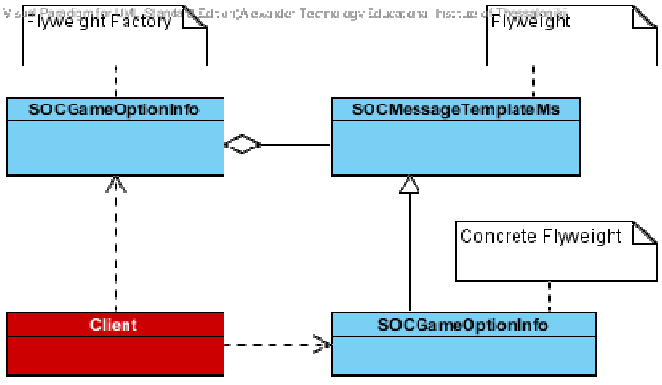
2.2.1. Παραδείγματα

Για την καλύτερη κατανόηση του τρόπου που ανακτώνται πρότυπα σχεδίασης και της δημιουργίας «υποψήφιων συστατικών» από αυτά παρουσιάζουμε κάποια παραδείγματα από τα projects που αναλύσαμε στα πλαίσια της Πτυχιακής εργασίας.

Από το project jSettlers2 ανακτήσαμε εάν στιγμιότυπο του προτύπου σχεδίασης Flyweight του οποίου το διάγραμμα κλάσεων παρουσιάζουμε. Η μορφή του στα αρχεία που παράγονται κατά την ανάλυση του project στο βήμα 7 («Ανάλυση αποτελεσμάτων Προτύπων Σχεδίασης») είναι η εξής:

```
Flyweight;Factory;soc.message.SOCGameOptionInfo;Flyweight;soc.message.SOCMess
ageTemplateMs;Subclass;soc.message.SOCGameOptionInfo
```

Ενώ το διάγραμμα Κλάσεων του συγκεκριμένου στιγμιότυπου του προτύπου είναι το εξής.

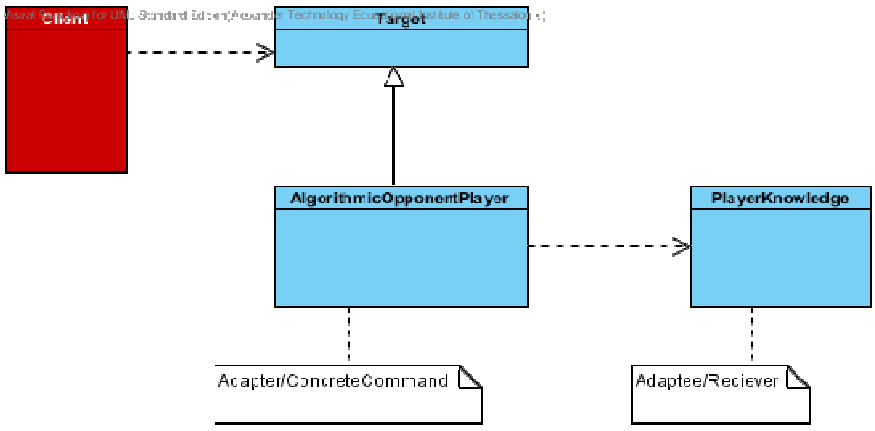


Σχήμα 24. Διάγραμμα Κλάσεων Παραδείγματος Προτύπου Σχεδίασης "Flyweight"

Από το project jSkat παρουσιάζουμε ένα στιγμιότυπο του προτύπου σχεδίασης «Προσαρμογέας (Adapter)» το οποίο καταγράφηκε ως εξής:

```
(Object)Adapter_Command;Adaptee/Receiver;org.jskat.ai.PlayerKnowledge;Adapter
/ConcreteCommand;org.jskat.ai.algorithmic.AlgorithmicOpponentPlayer
```

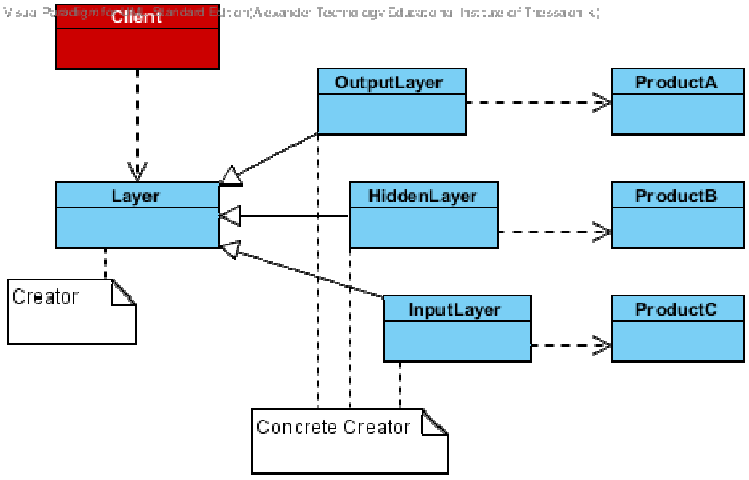
Το διάγραμμα κλάσεών του είναι το εξής.



Σχήμα 25. Διάγραμμα Κλάσεων Παραδείγματος Προτύπου Σχεδίασης "Προσαρμογέας (Adapter)"

Τέλος από το ίδιο project jSkat παρουσιάζουμε ένα στιγμιότυπο του προτύπου σχεδίασης «Μέθοδος Εργοστάσιο (Factory Method)» το οποίο είναι καταγεγραμμένο ως εξής:

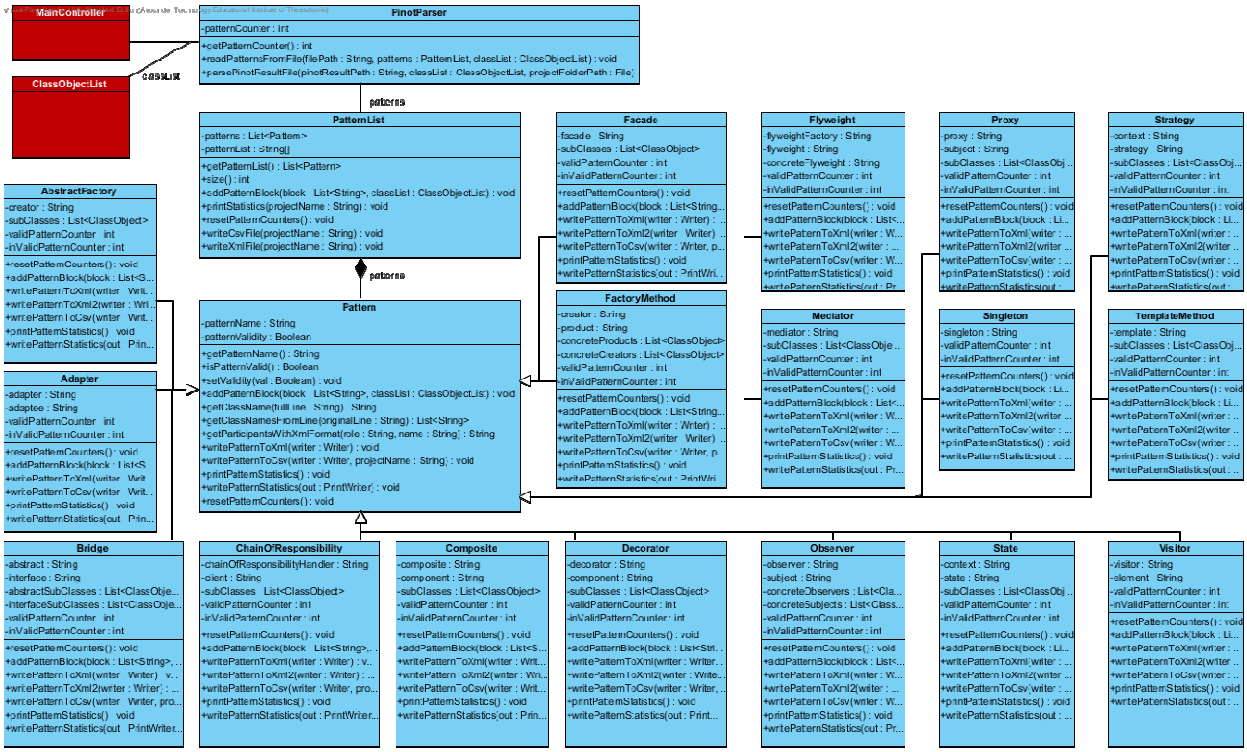
```
FactoryMethod;Creator;org.jskat.ai.nn.util.Layer;Subclass;org.jskat.ai.nn.uti
l.OutputLayer;Subclass;org.jskat.ai.nn.util.HiddenLayer;Subclass;org.jskat.ai
.nn.util.InputLayer
```



Σχήμα 26. Διάγραμμα Κλάσεων Παραδείγματος Προτύπου Σχεδίασης "Μέθοδος Εργοστάσιο (Factory Method)"

2.2.2. Διαγράμματα Κλάσεων

Παρουσιάζεται το διάγραμμα κλάσεων του εργαλείου που αναλύει τα αποτελέσματα των ανακτημένων προτύπων σχεδίασης από το εργαλείο Pinot καθώς γίνεται και μια σύντομη περιγραφή των κλάσεων του εργαλείου.



Σχήμα 27. Διάγραμμα κλάσεων του εργαλείου "Ανάλυσης Προτύπων Σχεδίασης"

- **Pattern** – Είναι μία αφηρημένη (abstract) κλάση που αναπαριστά ένα πρότυπο σχεδίασης ορίζοντας βασικές του ιδιότητες όπως το όνομα του και βασικές λειτουργίες όπως ο έλεγχος της εγκυρότητάς του καθώς και η εγγραφή των συμμετεχόντων του προτύπου σε ένα αρχείο κειμένου.
- **“ConcretePattern” Class** – Είναι ένα σύνολο κλάσεων που επεκτείνουν τις λειτουργίες της παραπάνω αφηρημένης κλάσης “Pattern” προσθέτοντας τις απαραίτητες πληροφορίες και ιδιότητες που χρειάζεται το κάθε ένα από τα δεκαεπτά (17) πρότυπα που αναγνωρίζει το εργαλείο Pinot. Οι συμμετέχοντες του κάθε προτύπου αναγνωρίζονται μέσα από το block κειμένου με το οποίο τροφοδοτείται το κάθε πρότυπο από την κλάση λίστα (patternList).

- **PatternList** – Η συγκεκριμένη κλάση διατηρεί μία λίστα από αντικείμενα (πρότυπα σχεδίασης) και είναι υπεύθυνη για την διαχείρισή τους καθώς και για την εγγραφή των τελικών αποτελεσμάτων στα κατάλληλα αρχεία κειμένου. Τροφοδοτείται από την PinotParser με block στοιχείων προτύπων σχεδίασης και ανάλογα με τον τύπο τους δημιουργεί το κατάλληλο αντικείμενο (πρότυπο σχεδίασης) το οποίο τροφοδοτεί παράλληλα με τα στοιχεία του προτύπου.
- **PinotParser** – Είναι η κλάση ελεγκτής (controller) του συγκεκριμένου εργαλείου καθώς διαχειρίζεται την λίστα με τα αντικείμενα (πρότυπα σχεδίασης) και είναι υπεύθυνη για το διάβασμα των blocks με το κάθε πρότυπο που έχει ανακτήσει το εργαλείο Pinot. Στη συνέχεια τροφοδοτεί την λίστα (patternList) με αυτά τα blocks.

2.2.3. Προβλήματα που Αντιμετωπίστηκαν

Τα προβλήματα που εμφανίστηκαν κατά την επεξεργασία των ανακτημένων προτύπων σχεδίασης από τον πηγαίο και μεταγλωττισμένο κώδικα, οφείλονταν στην ιδιαιτερότητα των δεδομένων, πρόβλημα που αναλύθηκε διεξοδικά στο κεφάλαιο 2.1.3.2. Ποιό συγκεκριμένα, κλάσεις που συμμετείχαν σε κάποιο πρότυπο το οποίο ανακτήθηκε, είτε από τον πηγαίο κώδικα είτε από το εκτελέσιμο αρχείο, είχαν χαρακτηριστεί στο βήμα 5 της εκτέλεσης του εργαλείου ως "μη έγκυρες" και είχαν διαγραφεί εξαιτίας της έλλειψης του πηγαίου ή εκτελέσιμου κώδικά τους. Επομένως είναι αδύνατη η δημιουργία "υποψήφιων συστατικών" που βασίζονται σε πρότυπα σχεδίασης και έτσι τα συγκεκριμένα προτεινόμενα "υποψήφια συστατικά" απορρίπτονται άμεσα από το σύστημα.

Επίσης ένα ακόμη πρόβλημα που προέκυψε είναι η συμμετοχή κλάσεων σε πρότυπα σχεδίασης οι οποίες ήταν "εσωτερικές" (inner classes). Στην περίπτωση αυτή έπρεπε να αντιστοιχίσουμε την εσωτερική κλάση στην αντίστοιχη εξωτερική -εάν αυτή ήταν "έγκυρη"- και στην περίπτωση που δεν ήταν δυνατό να απορρίψουμε το συγκεκριμένο στιγμιότυπο του προτύπου σχεδίασης. Παραδείγματα κώδικα από την ανάλυση των ανακτημένων στιγμιότυπων προτύπων σχεδίασης παρατίθενται παρακάτω.

2.2.4. Ενδεικτικός Κώδικας

Στο πρώτο παράδειγμα κώδικα γίνεται η ανάλυση ενός στιγμιότυπου του προτύπου σχεδίασης "Στρατηγική (Strategy)" που ανακτήθηκε από το εργαλείο pinot και αποθηκεύτηκε στο αντίστοιχο αρχείο αποτελεσμάτων του βήματος 6.2 της εκτέλεσης του εργαλείου καθώς και η εγγραφή των αποτελεσμάτων σε ένα csv αρχείο.

```
public void addPatternBlock(ArrayList<String> block, ClassObjectList
      classList) {
      for (int i = 0; i < block.size(); i++) {
            if (block.get(i).contains("is the Context class")) {
                  contextClass =
            classList.getMostValidClassName(getClassName(block.get(i)));
            } else if (block.get(i).contains("is the Strategy interface")) {
                  strategyInterface =
            classList.getMostValidClassName(getClassName(block.get(i)));
            } else if (block.get(i).contains("Concrete Strategy classes:")) {
            if (contextClass != null && strategyInterface != null) {
            ArrayList<String> temp = getClassNamesFromLine(
            block.get(i).substring(block.get(i).indexOf(":") + 1));
            if (!temp.isEmpty()) {
                  for (int j = 0; j < temp.size(); j++) {
                  if (temp.get(j) != null) {
                        String currentSubject =
                              classList.getMostValidClassName(temp.get(j));
                  if (currentSubject != null) {
                        concreteStrategies.add(currentSubject);
                                          }
                                    }
                              }
                        }
                  }
            }
      }

      // Checking pattern validity..
      if (contextClass == null
                  || strategyInterface == null
                  || concreteStrategies.isEmpty()) {
            setValidity(false);
            inValidPatternCounter++;
      } else {
            validPatternCounter++;
      }
}
```

```
public void writePatternToCsv(Writer writer, String projectName) throws
      IOException {
      writer.write(projectName + ";" + getPatternName());
      writer.write(";" + "Context" + ";" + contextClass);
      writer.write(";" + "Strategy" + ";" + strategyInterface);
      for (int i = 0; i < concreteStrategies.size(); i++) {
            writer.write(";" + "Subclass" + ";" + concreteStrategies.get(i));
      }
      writer.write("\n");
}
```

Στη συνέχεια παραθέτουμε ένα κομμάτι από τον κώδικα που αναλύει ένα στιγμιότυπο του προτύπου σχεδίασης που ανακτήθηκε από το εργαλείο pinot και αποθηκεύτηκε στο αντίστοιχο αρχείο αποτελεσμάτων του βήματος 6.2 της εκτέλεσης του εργαλείου καθώς και η εγγραφή των αποτελεσμάτων σε ένα csv αρχείο.

```
public static void parseAdapterPattern(Element currentPatternElement,
      String projectName, ClassObjectList classList, BufferedWriter out)
            throws IOException {
      NodeList currentInstanceList =
                  currentPatternElement.getElementsByTagName("instance");
      String patternName = currentPatternElement.getAttribute("name");

      for (int s1 = 0; s1 < currentInstanceList.getLength(); s1++) {
            System.out.println("\tparsing " + patternName);
            Node currentInstanceNode = currentInstanceList.item(s1);
            String adapter = "";
            String adaptee = "";
            if (currentInstanceNode.getNodeType() == Node.ELEMENT_NODE) {

                  Element currentInstanceElement = (Element)
                        currentInstanceNode;
                  NodeList roleList =
                        currentInstanceElement.getElementsByTagName("role");
                  for (int s2 = 0; s2 < roleList.getLength(); s2++) {
                        Node fstNode2 = roleList.item(s2);
                        if (fstNode2.getNodeType() == Node.ELEMENT_NODE) {
                              Element currentRoleElement = (Element)
                                    fstNode2;
                              String roleName =
                                    currentRoleElement.getAttribute("name");
                              String roleElement =
                              currentRoleElement.getAttribute("element");
                              if (roleName.equals("Adaptee/Receiver")) {
                                    adaptee = roleElement;
                              } else if
                              (roleName.equals("Adapter/ConcreteCommand")) {
```

```
                                    adapter = roleElement;
                              }
                        }
                  }
            }
            if (classList.getClass_Name(adapter) != null
                        && classList.getClass_Name(adaptee) != null) {
                  out.write(projectName + ";" + patternName + ";"
                              + "Adaptee/Receiver" + ";" + adaptee + ";"
                              + "Adapter/ConcreteCommand" + ";" + adapter +
                                    "\n");
                  ++patternCount;
            } else {
                  System.out.println("\t\t (ERROR at retrieving Pattern
                        Participants from ClassList");
            }
      }
}
```

2.3. Αξιολόγηση Υποψήφιων Συστατικών

Η επαναχρησιμοποίηση λογισμικού θεωρείται μία σημαντική και ευρείας χρήσης μέθοδο αντιμετώπισης προβλημάτων στην ανάπτυξη λογισμικού. Θεωρείται πως αυξάνει την παραγωγικότητα και βελτιώνει την ποιότητα της ανάπτυξης λογισμικού βάση των (Barnes et. al, 1988), (Birgerstaff and Richter, 1987) και (Schafer et. al, 1994). Με ποιο τρόπο όμως επιλέγει ο ενδιαφερόμενος ένα ήδη υπάρχον ανοιχτό λογισμικό ώστε να το επαναχρησιμοποιήσει?

Όλες οι μεθοδολογίες για την επιλογή εφαρμογών λογισμικού που έχουν προταθεί μέχρι σήμερα συγκρίνουν τις απαιτήσεις των χρηστών με τις παρεχόμενες από το υποψήφιο λογισμικό λειτουργίες. Υπάρχουν διαφορετικά ήδη απαιτήσεων όπως οι «διαχειριστικές», οι «πολιτικές» και φυσικά οι ποιοτικές απαιτήσεις. οι τελευταίες είναι δύσκολο να ελεγχθούν λόγω της φύσεώς τους. Έχουν γίνει αρκετές προσπάθειες ώστε να μοντελοποιηθούν οι ποιοτικές απαιτήσεις ώστε να εφαρμόζονται σε οποιαδήποτε εφαρμογή λογισμικού. Ένα παράδειγμα τέτοιας προσέγγισης γίνεται στο (Bansiya and Davis, 2002).

Στο επόμενο υπο-κεφάλαιο γίνεται μια σύντομη περιγραφή των μετρικών που ποσοτικοποιούν τα χαρακτηριστικά ποιότητας του λογισμικού καθώς και τις επιλεγμένες

μετρικές που χρησιμοποιούνται για την αξιολόγηση των «υποψήφιων συστατικών» που βασίζονται σε εξαρτήσεις.

2.3.1. Ποιότητα Λογισμικού

Υπάρχουν διάφοροι τρόποι να ορίσει κανείς την έννοια της ποιότητας λογισμικού. Σύμφωνα με το (Kitchenham and Pfleeger, 1996) η ποιότητα είναι μια σύνθετη και πολύπλευρη ιδέα που μπορεί να οριστεί από πέντε διαφορετικές οπτικές. Ορίζεται ως κάτι που μπορεί να αναγνωριστεί αλλά όχι να οριστεί. Από τη σκοπιά του χρήστη, η ποιότητα αφορά την καταλληλότητα ως προς κάποιον συγκεκριμένο σκοπό. Από κατασκευαστική άποψη, ποιότητα σημαίνει συμμόρφωση με τις προδιαγραφές. Από άποψη προϊόντος, η ποιότητα αφορά τη σύνδεση των έμφυτων χαρακτηριστικών του προϊόντος, ενώ έχοντας ως βάση της αξία, η ποιότητα εξαρτάται από το ποσό που ένας πελάτης είναι διατεθειμένος να πληρώσει.

Η ποιότητα του λογισμικού διαχωρίζεται σε δυο κατηγορίες, (α) την εσωτερική ποιότητα και (β) την εξωτερική ποιότητα (Kitchenham and Pfleeger, 1996). Η εσωτερική ποιότητα ασχολείται με τα χαρακτηριστικά του λογισμικού που δεν μπορούνε να παρατηρηθούν από το χρήστη ή τον προγραμματιστή, όπως για παράδειγμα η συνοχή, η πολυπλοκότητα και η σύζευξη. Η εσωτερική ποιότητα είναι μετρήσιμη μέσω των μετρικών, που μπορούν να λαμβάνονται απευθείας είτε από τον πηγαίο κώδικα, είτε από σχεδιαστικά τεχνουργήματα, όπως διαγράμματα κλάσεων ή ακολουθίας. Η εξωτερική ποιότητα ασχολείται με τις πτυχές του λογισμικού που γίνονται αντιληπτές είτε από τον προγραμματιστή είτε από τον χρήστη, όπως η ευκολία συντήρησης, η λειτουργικότητα και η χρηστικότητα. Τα χαρακτηριστικά της εξωτερικής ποιότητας είναι μη μετρήσιμα, και προκειμένου να έχουν πρόσβαση σε αυτά, οι προγραμματιστές χρησιμοποιούν ένα μοντέλο ποιότητας ώστε να αποτυπώσουν τα χαρακτηριστικά της εσωτερικής ποιότητας σε εξωτερικά χαρακτηριστικά.

Το μοντέλο αυτό περιλαμβάνεται στο διεθνές πρότυπο, ISO 9126, που έχει αναπτυχθεί με σκοπό την εκτίμηση της ποιότητας του λογισμικού. Το ISO 9126 προσδιορίζει ορισμένα ποιοτικά χαρακτηριστικά και τις μετρικές που τα υπολογίζουν. (Jung et al., 2004) Τα έξι βασικά χαρακτηριστικά που ορίζει το πρότυπο είναι η

λειτουργικότητα, η αξιοπιστία, η χρηστικότητα, η αποτελεσματικότητα, η ευκολία για συντήρηση και η φορητότητα, ενώ ορίζει και 27 υποκατηγορίες που αναλύουν αυτά τα χαρακτηριστικά. Το πρότυπο ορίζει επίσης μετρικές για την μέτρηση των 27 χαρακτηριστικών. Για παράδειγμα, όσον αφορά το ποιοτικό χαρακτηριστικό της λειτουργικότητας, μπορεί να αναπαρασταθεί από τις τιμές των 5 υποκατηγοριών από τις οποίες συντίθεται, και χρησιμοποιώντας μια κατάλληλη μέθοδο σύνθεσης των αποτελεσμάτων να προκύψει μια τιμή που θα προσδιορίζει τη λειτουργικότητα του συστήματος. Επειδή το ποιοτικό μοντέλο που παρέχει το πρότυπο είναι πολύ γενικό, μπορεί να εφαρμοστεί σε οποιοδήποτε λογισμικό.

Αναλύοντας τώρα τις μετρικές σε επίπεδο κώδικα, βασικό πλεονέκτημά τους είναι ότι παρέχουν στους προγραμματιστές διορατικότητα στο εσωτερικό του κώδικα που αναπτύσσουν και τους βοηθούν να κατανοήσουν ποιοι κομμάτια κώδικα πρέπει να ανακατασκευαστούν ή να ελεγχθούν σχολαστικά. Βοηθούν στην αναγνώριση ενδεχόμενων κινδύνων, στην κατανόηση της εκάστοτε κατάστασης ενός έργου και στην καταγραφή της προόδου κατά την ανάπτυξη του λογισμικού. Τέτοιες μετρικές είναι (1) η WMPC (Weighted Methods Per Class), που υπολογίζει την πολυπλοκότητα μιας κλάσης, μετρώντας τον αριθμό των μεθόδων που την αποτελούν, (2) η DIT (Depth of Inheritance Tree), που συσχετίζει τον βάθος μιας κλάσης στην ιεραρχία με την πολυπλοκότητα της, θεωρώντας ότι όσο μεγαλύτερο το βάθος στην ιεραρχία, τόσο πιο πιθανή η κληρονομικότητα κλάσεων υψηλότερων στην ιεραρχία, που έχει ως αποτέλεσμα την αύξηση του αριθμού των μεθόδων στην κλάση και συνεπώς την αύξηση της πολυπλοκότητας. (3) η NOC (Number Of Children) που μετράει πόσες υποκλάσεις πρόκειται να κληρονομήσουν τις μεθόδους των κλάσεων γονέων, (4) η CBO (Coupling Between Object Classes) που υπολογίζει το κατά πόσο οι κλάσεις εξαρτώνται από άλλες κλάσεις για να λειτουργήσουν ή είναι αυτόνομες, γεγονός που επηρεάζει πολύ την δυνατότητα επαναχρησιμοποίησης, (5) η RFC (Response For a Class), που μετράει τον αριθμό των μεθόδων που ανταποκρίνονται κατά τη λήψη ενός μηνύματος από μια κλάση. Όσο μεγαλύτερος ο αριθμός αυτός, τόσο πιο σύνθετη αναμένεται να είναι η διαδικασία της αποσφαλμάτωσης και του ελέγχου, αφού για τη λειτουργία της κλάσης εμπλέκονται και μέθοδοι που καλούνται από εξωτερικές κλάσεις, αυξάνοντας έτσι την πολυπλοκότητα. (6) η LCOM (Lack of Cohesion in Methods) που

μετράει τη συνοχή των μεθόδων μιας κλάσης, και είναι επιθυμητή αφού προωθεί την ενθυλάκωση και μειώνει πολυπλοκότητα και την πιθανότητα λαθών κατά τη διαδικασία της ανάπτυξης (Chidamber and Kemerer, 1994). Οι μετρικές σε επίπεδο κώδικα χαρακτηρίζονται από μεγάλη ακρίβεια, αλλά μπορούν να υπολογιστούν μόνο κατά την φάση εκτέλεσης.

Αντίθετα, οι μετρικές σε σχεδιαστικό επίπεδο, δεν είναι τόσο ακριβείς, μπορούν όμως να υπολογιστούν νωρίτερα, και να παρέχουν ενδείξεις για την τελική ποιότητα του λογισμικού. Για να μπορέσουν να χρησιμοποιηθούν πρέπει το αντικειμενοστραφές σχέδιο να περιέχει πληροφορίες, όπως ορισμό των κλάσεων, ιεραρχία των κλάσεων, διευκρινήσεις για τη λειτουργικότητα των μελών των κλάσεων και διευκρινήσεις σχετικά με τις παραμέτρους, τους τύπους, και τις ιδιότητες που χρησιμοποιούνται (Bansiya and Davis, 2002). Σύμφωνα με μια έρευνα που έχει διεξαχθεί, υπάρχουν ορισμένες μετρικές που μπορούν να τροποποιηθούν και να χρησιμοποιηθούν για την αξιολόγηση των σχεδιαστικών χαρακτηριστικών, όπως η αφαίρεση, η ανταλλαγή μηνυμάτων και η κληρονομικότητα. Υπάρχουν όμως και κάποια σχεδιαστικά χαρακτηριστικά, όπως η ενθυλάκωση και η σύνθεση για τα οποία δεν υπάρχουν αντικειμενοστραφείς σχεδιαστικές μετρικές. Μετρικές υπάρχουν ήδη για την μέτρηση της συνοχής και της σύζευξης που απαιτούν ωστόσο μια σχεδόν ολοκληρωμένη εφαρμογή των κλάσεων για να μπορέσουν να υπολογιστούν. Ορισμένες μετρικές σε επίπεδο σχεδίου είναι οι εξής: (1) η DSC (Design Size in Classes) που μετράει το σύνολο των κλάσεων στο σχέδιο. (2) η NOH (Number Of Hierarchies) που μετράει τον αριθμό των ιεραρχιών των κλάσεων στο σχέδιο. (3) η ANA (Average Number of Ancestors) που επισημαίνει τον μέσο όρο των κλάσεων από τις οποίες μια κλάση κληρονομεί πληροφορίες. Υπολογίζεται προσδιορίζοντας όλες τις κλάσεις ξεκινώντας από την ρίζα μιας ιεραρχικής δομής. (4) η DAM (Data Access Metric), το ποσοστό δηλαδή των ιδιωτικών ιδιοτήτων προς το σύνολο των ιδιοτήτων μιας κλάσης. Επιθυμητά είναι τα μεγάλα ποσοστά της μετρικής. (5) η DCC (Direct Class Coupling) που υπολογίζει των αριθμό των διαφορετικών κλάσεων με τις οποίες σχετίζεται άμεσα μια κλάση. Η μετρική περιλαμβάνει κλάσεις που σχετίζονται άμεσα με κάποια ιδιότητα, δήλωση ή παράμετρο μιας μεθόδου. (6) η CAM (Cohesion Among Methods of Class) που υπολογίζει τη συσχέτιση μεταξύ των μεθόδων μιας κλάσης και της λίστας παραμέτρων της. (7) η MOA (Measure Of

Aggregation) που μετράει τον αριθμό των δηλωτικών δεδομένων που οι τύποι τους είναι κλάσεις ορισμένες από τον χρήστη. (8) η MFA (Measure of Functional Abstraction) το ποσοστό δηλαδή των μεθόδων που κληρονομούνται από μια κλάση προς τον συνολικό αριθμό των μεθόδων που είναι προσβάσιμες από τις μεθόδους μια κλάσης. (9) η NOP (Number of Polymorphic Methods) που μετράει τις μεθόδους που εμφανίζουν πολυμορφική συμπεριφορά. (10) η CIS (Class Interface Size) που μετράει των αριθμό των δημόσιων μεθόδων σε μια κλάση. (11) η NOM (Number Of Methods) που μετράει όλες τις μεθόδους που ορίζονται σε μια κλάση.

2.3.2. Επιλεγμένες Μετρικές

Οι μετρικές που καταγράφονται και χρησιμοποιούνται από το εργαλείο είναι οι: FanIn, FanOut, CBO, CAMC, CIS, NOC και αναλύονται παρακάτω όσες δεν αναλύθηκαν στο παραπάνω κεφάλαιο 2.3.1.

2.3.2.1. Fan Out

Η μετρική Fan-out εκτιμάει την σύζευξη του λογισμικού. Μετρά τον αριθμό των τύπων αναφορών που χρησιμοποιούνται σε δηλώσεις ιδιοτήτων, επίσημες παραμέτρους, τύπους επιστροφής. Στην ουσία είναι ο αριθμός των εξωτερικών κλάσεων των οποίων υλοποιεί ή απλά χρησιμοποιεί το σύστημά μας. Επομένως, συστήματα ή κλάσεις με μεγάλο fan-out απαιτούν μεγάλο κόστος για συντήρηση και έχουν μικρή αυτονομία έξω από το ολοκληρωμένο σύστημα. Σε επίπεδο «υποψήφιου συστατικού» είναι το σύνολο των κλάσεων στις οποίες αναφέρονται οι κλάσεις που συμμετέχουν στο «υποψήφιο συστατικό».

2.3.2.2. Fan In

Η μετρική Fan In μετράει το πόσες κλάσεις εξαρτώνται από την κλάση που μελετάμε. Η μετρική αυτή μπορεί να αναχθεί και σε επίπεδο ενός συστήματος ή ενός "υποψήφιου συστατικού" μετρώντας το πόσες εξωτερικές προς το σύνολό μας κλάσεις, αναφέρονται σε αυτό. Σε επίπεδο κλάσης δηλώνει το πόσες κλάσεις χρειάζονται την υπό μελέτη κλάση ώστε να μεταγλωττιστούν σωστά. Στο εργαλείο καταγράφεται ως το ο αριθμός των εξωτερικών, ως προς το "υποψήφιο συστατικό" μας, κλάσεων του project

προς τον συνολικό αριθμό κλάσεων. Δηλαδή αντιπροσωπεύει το ποσοστό των κλάσεων του συστήματος που χρησιμοποιούν κλάσεις από το "υποψήφιο συστατικό" μας.

2.3.2.3. CBO

Η μετρική "Σύζευξη μεταξύ Αντικειμένων" (Coupling Between Object Classes) αναλύθηκε στο κεφάλαιο 2.3.1. Στο εργαλείο μας είναι απαραίτητο να την υπολογίσουμε σε επίπεδο "υποψήφιου συστατικού", πακέτου και συνόλου πακέτων έτσι τα αποτελέσματά της να χρησιμοποιηθούν για την εύρεση της μετρική R (Επαναχρησιμοποίησης). Στο επίπεδο "υποψήφιου συστατικού", πακέτου και συνόλου πακέτων υπολογίζεται ως ο μέσος όρος των μετρήσεων της CBO για κάθε κλάση που συμμετέχει στα παραπάνω σύνολα κλάσεων.

2.3.2.4. CAMC (Cohesion Among Methods of Class)

Η CAMC υπολογίζει τη συνοχή μεταξύ των μεθόδων μιας κλάσης εξετάζοντας την λίστα των παραμέτρων της. Όπως και η CBO, έτσι και αυτή υπολογίζεται σε επίπεδο "υποψήφιου συστατικού", πακέτου και συνόλου πακέτων κρατώντας το μέσο όρο της μετρικής CAMC της κάθε κλάσης που συμμετέχει σε κάποιο από τα παραπάνω σύνολα κλάσεων.

2.3.2.5. CIS (Class Interface Size)

Η CIS μετράει των αριθμό των δημόσιων μεθόδων σε μια κλάση, δηλαδή όλους τους πιθανούς τρόπους με τους οποίους μπορεί να χρησιμοποιηθεί από άλλες κλάσεις. Υπολογίζεται σε επίπεδο "υποψήφιου συστατικού", πακέτου και συνόλου πακέτων κρατώντας το μέσο όρο της μετρικής CIS της κάθε κλάσης που συμμετέχει σε κάποιο από τα παραπάνω σύνολα κλάσεων.

2.3.2.6. NOC (Number of Classes)

Αυτή η μετρική υπολογίζει τον αριθμό των κλάσεων σε κάθε "υποψήφιο συστατικό", πακέτο κλάσεων ή σύνολο πακέτων κλάσεων. Τα αποτελέσματα της μετρικής μπορούν να χρησιμοποιηθούν για την εκτίμηση αλλαγών στην εφαρμογή και τη σύγκρισή της με άλλες εφαρμογές με κριτήριο τον αριθμό των κλάσεών τους. Συνήθως εφαρμογές με μεγάλο αριθμό κλάσεων προσφέρουν μεγάλη λειτουργικότητα αλλά παρουσιάζουν δυσκολίες στην επεκτασιμότητα και συντηρησιμότητα.

2.3.2.7. Μέτρηση Επαναχρησιμοποίησης

Η επιλογή των κλάσεων που μπορούν να ενταχθούν σε ένα συστατικό λογισμικού απαιτεί την αξιολόγηση αρκετών χαρακτηριστικών των «υποψήφιων συστατικών» (Franch and Carvallo, 2003) και (Kontio, 2006). Οι συγγραφείς στα (Andreou and Tziakouris, 2007), (Cho et. al, 2001), (Fahmi and Choi), και (Yu et. al, 2009) προτείνουν ότι ένας αποδοτικός τρόπος επιλογής συστατικών είναι η επιλογή πακέτων με βάση τα δομικά χαρακτηριστικά ποιότητας. Στη συγκεκριμένη μελέτη, η επαναχρησιμοποιησιμότητα επιλέχθηκε να χρησιμοποιηθεί ως μέτρο σύγκρισης των υποψήφιων συστατικών. Σύμφωνα με τον Bansyia *«Επαναχρησιμοποιησιμότητα Λογισμικού είναι η παρουσία των αντικειμενοστραφών χαρακτηριστικών που επιτρέπουν σε κάποιο σύστημα να ανεφάρμοστη σε κάποιο άλλο πρόβλημα, χωρίς σημαντική* προσπάθεια» (Bansiya and Davis, 2002). Τα χαρακτηριστικά τα οποία σύμφωνα με τη βιβλιογραφία επηρεάζουν την επαναχρησιμοποιησιμότητα συνοψίζονται στον Πίνακα 13.

Πίνακας 13. Δομικά Χαρακτηριστικά Ποιότητας - Επαναχρησιμοποίηση

Χαρακτηριστικό	**Επίδραση**	**Αναφορά**
Σύζευξη	-	(Bansiya and Davis, 2002) (Barnard, 1998) (Gui and Scott, 2007) (Sandlhu te. al, 2008)
Συνοχή	+	(Bansiya and Davis, 2002) (Sandlhu te. al, 2008)
Πέρασμα Μηνυμάτων	+	(Bansiya and Davis, 2002)
Μέγεθος	+	(Bansiya and Davis, 2002)
Κληρονομικότητα	-	(Barnard, 1998) (Sandlhu te. al, 2008)
Πολυπλοκότητα	-	(Barnard, 1998) (Sandlhu te. al, 2008)

Στο (Bansiya and Davis, 2002), οι συγγραφείς προτείνουν το μοντέλο ποιότητας QMOOD. Μεταξύ άλλων, το μοντέλο υπολογίζει την επαναχρησιμοποιησιμότητα του λογισμικού, μέσω μετρικών ποιότητας χαμηλού επιπέδου σε αρχικά στάδια της

σχεδίασης του λογισμικού. Για τον υπολογισμό της επαναχρησιμοποιησιμότητας χρησιμοποιείται μια γραμμική εξίσωση με όρους γνωστές αντικειμενοστραφείς μετρικές. Το προτεινόμενο ιεραρχικό μοντέλο, επικυρώθηκε μέσω ενός τυπικού πειράματος με επαγγελματίες μηχανικούς λογισμικού.

Επιπλέον, στο (Barnard, 1998) ο συγγραφέας προτείνει κάποιες οριακές τιμές σε επίπεδο κώδικα και στυλ κωδικοποίησης τα οποία, όταν υπερβαίνονται, η επαναχρησιμοποίηση του συστήματος γίνεται δύσκολη. Όμως, η αυτόματη εφαρμογή του μοντέλου δεν είναι εφικτή λόγω κάποιων υποκειμενικών μετρήσεων, όπως «όνομα ιδιότητας με νόημα». Στο (Gui and Scott, 2007) μελετήθηκε η συσχέτιση μεταξύ αρκετών μετρικών σύζευξης και της επαναχρησιμοποιησιμότητας συστατικών. Επιπλέον, οι συγγραφείς του (Andreou and Tziakouris, 2007) προτείνουν ένα μοντέλο αξιολόγησης της ποιότητας των συστατικών βασισμένο στο ISO 9126, αλλά η προσέγγιση είναι κυρίως ποιοτική και κατά συνέπεια δύσκολα αυτοματοποιήσιμη.

Στο (Washizaki et. al, 2003), οι συγγραφείς προτείνουν ότι κάποιοι ευρετικοί κανόνες, όπως η ύπαρξη μετα-πληροφοριών μπορούν να αποδειχθούν χρήσιμοι στη μέτρηση της επαναχρησιμοποιησιμότητας συστατικών λογισμικού. Παρόμοια με το (Barnard, 1998), η χρήση του συγκεκριμένου μοντέλου δεν είναι εφικτή γιατί δεν αυτοματοποιείται. Τέλος, το (Sandlhu te. al, 2008) προτείνει το συνδυασμό διάφορων μετρικών μέσω νευρωνικών δικτύων, με στόχο να εκτιμήσει την επαναχρησιμοποιησιμότητα αντικειμενοστραφών συστημάτων. Όμως, η συγκεκριμένη μέθοδος δεν μπορεί να φανεί χρήσιμη για τη συγκεκριμένη έρευνα, η οποία χρειάζεται αριθμητικά δεδομένα.

Συμπερασματικά, στη συγκεκριμένη μελέτη αποφασίστηκε να χρησιμοποιηθεί το μοντέλο QMOOD (Bansiya and Davis, 2002) διότι μπορεί να αυτοματοποιηθεί, δεν περιέχει υποκειμενικά κριτήρια και είναι επαρκώς αξιολογημένο. Επιπλέον, το μοντέλο QMOOD βασίζεται σε μετρικές οι οποίες έχουν χρησιμοποιηθεί σε αρκετές δημοσιευμένες εργασίες του χώρου (Counsell et. al, 2006), (Hsueh et. al, 2008), (Genero et. al 2007), (Etzkorn et. al, 2004), (Khomh and Gueheneuc, 2008), (Marcus et. al, 2008) και (Plague et. al, 2007). Επιπλέον, η επαναχρησιμοποιησιμότητα όπως ορίζεται στο επιλεχθέν μοντέλο, λαμβάνει υπόψη της δομικά χαρακτηριστικά, όπως η

σύζευξη και η συνοχή, τα οποία είναι πολύ σημαντικά στην εφαρμογή της επαναχρησιμοποίησης λευκού κουτιού. Ο υπολογισμός της επαναχρησιμοποιησιμότητας σύμφωνα με το QMOOD είναι ο εξής:

$$\text{reusability} = 0.25 * cohesion + 0.5 * messaging + 0.5 * size - 0.25 * coupling$$

Σε επίπεδο συστατικού, πακέτου ή συνόλου πακέτων υπολογίζουμε την επαναχρησιμοποιησιμότητα ως το μέσο όρο της επαναχρησιμοποιησιμότητας των κλάσεων που συμμετέχουν σε αυτό.

$$reusabilit\ y = \frac{\sum_{i=1}^{NOC} reusabilit\ y\_of\ \_class\ \_i}{NOC}$$

3. Εμπειρική Αξιολόγηση Συστατικών

3.1. Μεθοδολογία Μελέτης Περίπτωσης

Σε αυτό το κεφάλαιο αξιολογούμε τα "συστατικά λογισμικού" που ανακτήθηκαν σύμφωνα με τη μεθοδολογία. Η μέθοδος αξιολόγησης (validation method) είναι μία μελέτη περίπτωσης. Μία μελέτη περίπτωσης αποτελεί κατάλληλη μέθοδο αξιολόγησης όταν έχουμε διαθέσιμο ένα πολύ μεγάλο πλήθος δεδομένων προς ανάλυση και όταν το περιβάλλον στο οποίο εφαρμόζεται η μέθοδος δεν είναι ελεγχόμενο. Η οργάνωση της μελέτης περίπτωσης είναι όμοια με αυτή που εφαρμόζουν οι συγγραφείς στο (Ampatzoglou et al., 2011) με σκοπό την αξιολόγηση της επαναχρησιμοποιησιμότητας των προτύπων σχεδίασης.

3.1.1. Πλάνο της Μελέτης Περίπτωσης

Ο στόχος της συγκεκριμένης μελέτης περίπτωσης είναι να μελετήσει τα ποιοτικά χαρακτηριστικά των "συστατικών λογισμικού (components)" που ανακτήθηκαν εφαρμόζοντας την προτεινόμενη μεθοδολογία, σε σχέση με "συστατικά λογισμικού που βασίζονται σε πακέτα (packages)". Τα βήματα της εκτέλεσης της μελέτης περίπτωσης είναι τα εξής:

- Προσδιορισμός των ερωτημάτων της έρευνας.
- Δημιουργία του συνόλου δεδομένων.
- Εύρεση της μεθόδου ανάλυσης δεδομένων.
- Εκτέλεση της μελέτης περίπτωσης.
- Ανάλυση και καταγραφή των αποτελεσμάτων.

3.1.2. Τα Ερωτήματα της Έρευνας

Το ερώτημα της έρευνας της Πτυχιακής εργασίας μπορεί να περιγραφεί από το ακόλουθο σενάριο. "Ένας προγραμματιστής θέλει να αναπτύξει μία λειτουργία. Έχοντας αναζητήσει σε άλλα projects του συγκεκριμένου τομέα έχει βρει μία κλάση που του παρέχει την λειτουργία. Ποιές κλάσεις από αυτές που εξαρτάται η κλάση που επέλεξε, πρέπει να επιλεγούν, μετασχηματιστούν και επαναχρησιμοποιηθούν στο project που αναπτύσσει?" Στην έρευνά μας μελετάμε τρεις εναλλακτικές επιλογές για τον προγραμματιστή:

- Επιλογή ενός "συστατικού λογισμικού" βασισμένο στη μεθοδολογία που αναπτύξαμε και περιέχει την κλάση που μας ενδιαφέρει. *[Εναλλακτική επιλογή Α1 - "Συστατικό Λογισμικού (Component)"]*
- Επιλογή του πακέτου στο οποίο ανήκει η κλάση που έχουμε επιλέξει. *[Εναλλακτική επιλογή Α2 - "Πακέτο Κλάσεων (Package)"]*
- Επιλογή του συνόλου των πακέτων στα οποία ανήκουν οι κλάσεις που απαρτίζουν το "συστατικό λογισμικού" που επιλέξαμε στην "Εναλλακτική επιλογή Α1". *[Εναλλακτική επιλογή Α3 - "Σύνολο Πακέτο Κλάσεων (Package Set)"]*

3.1.3. Δημιουργία του συνόλου δεδομένων

Στην μελέτη περίπτωσής μας, εφαρμόσαμε την προτεινόμενη μεθοδολογία σε δεκατέσσερα (14) παιχνίδια ανοιχτού λογισμικού γραμμένα σε java. Τα projects επιλέχθηκαν τυχαία από αποθετήρια ανοιχτού λογισμικού. Αφού εφαρμόσαμε την προτεινόμενη μεθοδολογία σε όλες τις κλάσεις των Projects (2.803), προτείναμε 577.319 "υποψήφια συστατικά λογισμικού". Με σκοπό την αξιολόγηση των "υποψήφιων συστατικών" δημιουργήσαμε ένα πίνακα αποτελεσμάτων, με 12 στήλες για κάθε ένα "υποψήφιο συστατικό". Οι στήλες του πίνακα είναι οι εξής:

- Μέγεθος της Εναλλακτικής A1 ("Component")
- Εξωτερικές εξαρτήσεις (Fan Out) της Εναλλακτικής A1 ("Component")
- Λειτουργικότητα (Fan In) της Εναλλακτικής A1 ("Component")
- Επαναχρησιμοποιησιμότητα (R metric) της Εναλλακτικής A1 ("Component")
- Μέγεθος της Εναλλακτικής A2 ("Package")
- Εξωτερικές εξαρτήσεις (Fan Out) της Εναλλακτικής A2 ("Package")
- Λειτουργικότητα (Fan In) της Εναλλακτικής A2 ("Package")
- Επαναχρησιμοποιησιμότητα (R metric) της Εναλλακτικής A2 ("Package")
- Μέγεθος της Εναλλακτικής A3 ("Package Set")
- Εξωτερικές εξαρτήσεις (Fan Out) της Εναλλακτικής A3 ("Package Set")
- Λειτουργικότητα (Fan In) της Εναλλακτικής A3 ("Package Set")
- Επαναχρησιμοποιησιμότητα (R metric) της Εναλλακτικής A3 ("Package Set")

3.1.4. Μέθοδοι Ανάλυσης Δεδομένων

Με σκοπό την σύγκριση των τριών εναλλακτικών, επιλέξαμε μετρικές ποιοτικών χαρακτηριστικών όπως το μέγεθος, η ανεξαρτησία, η λειτουργικότητα και η επαναχρησιμοποιησιμότητα για κάθε "υποψήφιο συστατικό" για κάθε μία από τις τρεις εναλλακτικές προσεγγίσεις. Οι επιλεγμένες μετρικές παρουσιάζονται στον παρακάτω πίνακα. Οι συγκρίσεις των τριών εναλλακτικών έγιναν βάση περιγραφικής στατιστικής όπως με πίνακες συχνοτήτων και διαγράμματα.

Πίνακας 14. Μετρικές που χρησιμοποιήθηκαν στην αξιολόγηση των "Υποψήφιων Συστατικών"

Χαρακτηριστικό	**Μετρική**	**Ορισμός**
Μέγεθος	Αριθμός Κλάσεων (NOC)	Μετράει τον αριθμό των κλάσεων που συμμετέχουν στο "υποψήφιο συστατικό".
Εξαρτήσεις	Fan Out (FO)	Μετράει τις εξωτερικές -προς το "υποψήφιο συστατικό"- κλάσεις οι οποίες είναι απαραίτητες ώστε αυτό να μεταγλωττιστεί.
Λειτουργικότητα	Weighted Fan In (WFI)	Παρουσιάζει το ποσοστό των εξωτερικών - προς το "υποψήφιο συστατικό" - κλάσεων που χρησιμοποιούν τουλάχιστον μία από τις κλάσεις του, από το σύνολο των κλάσεων του project.
Επαναχρησιμοποιησιμότητα	Επαναχρησι-μοποιησιμότητα (R)	Είναι μία συνάρτηση που υπολογίζει την ευκολία με την οποία ένα σύνολο από κλάσεις μπορεί να μεταφερθεί σε ένα άλλο. Υπολογίζεται βάση των παραπάνω μετρικών. (Bansiya and Davis, 2002)

3.2. Αποτελέσματα

Στον πίνακα 14 παρουσιάζουμε τη περιγραφική στατιστική για το δείγμα της μελέτης περίπτωσης. Παρατηρήθηκε πως ο μέσος όρος του αριθμού των κλάσεων στα (14) παιχνίδια ανοιχτού λογισμικού που μελετήσαμε ήταν 340 κλάσεις με μέσο όρο στα χαρακτηριστικά που περιγράψαμε παραπάνω να παρουσιάζονται στον παρακάτω πίνακα 15.

Πίνακας 15. Σύγκριση ποιοτικών χαρακτηριστικών μεταξύ των τριών Εναλλακτικών

Ποιοτικό Χαρακτηριστικό	**Εναλλακτική Επιλογή A1**	**Εναλλακτική Επιλογή A2**	**Εναλλακτική Επιλογή A3**
Μέγεθος (Μέσος όρος)	20	68	165
Εξαρτήσεις (Μέσος όρος)	18	20	19
Λειτουργικότητα (Μέσος όρος - Ποσοστό)	34%	8%	24.8%
Επαναχρησιμοποιησιμότητα (Μέσος όρος)	5.4	3.1	3.6

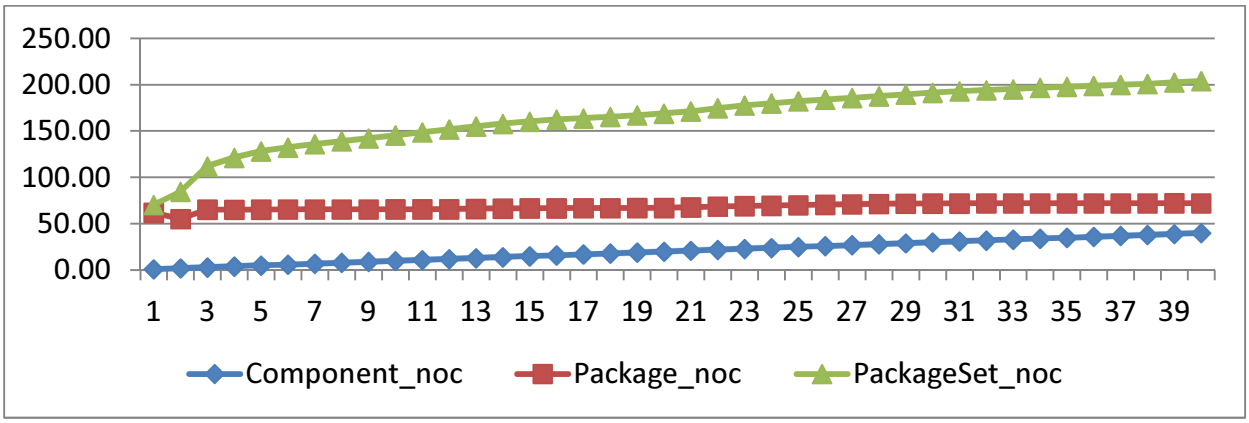
Ο πίνακας 16 παρέχει μια αποτύπωση των δομικών ποιοτικών χαρακτηριστικών του επιλεγμένου «υποψήφιου συστατικού» βάση της εναλλακτικής μεθόδου που επιλέχθηκε. Ωστόσο, για να μελετήσουμε εάν τα παρακάτω αποτελέσματα μπορούν να γενικευτούν για κάθε μέγεθος «υποψήφιου συστατικού», έπρεπε να διαχωρίσουμε το σύνολο των δεδομένων σε μικρότερα υποσύνολα και να τα μελετήσουμε ξεχωριστά.

Πίνακας 16. Περιγραφική Στατιστική

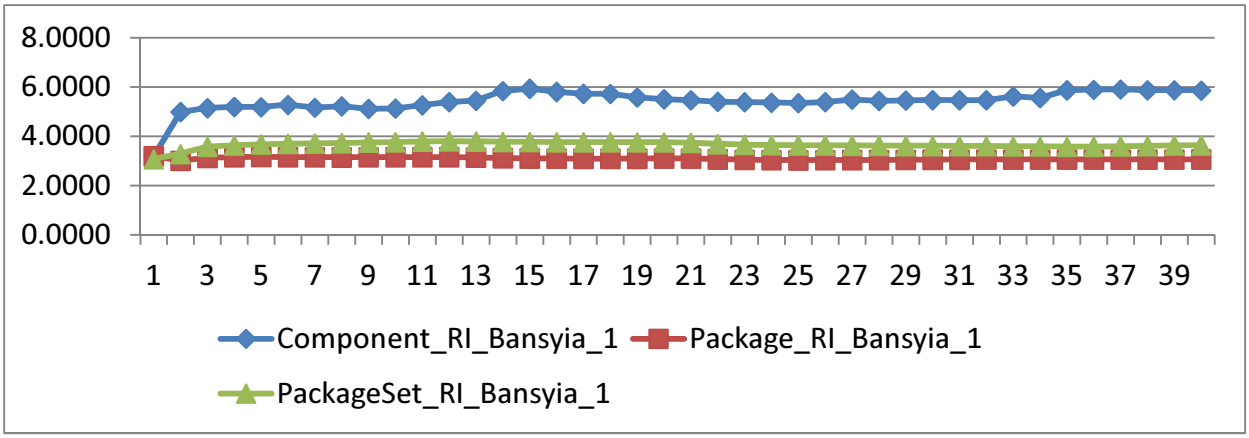
	N	**min**	**max**	**mean**	**std. dev.**
Μέγεθος (NOC)	576803	13.000	590.000	340.520	179.880
Εναλλακτική A1 (NOC)	576803	1.000	40.000	20.310	11.190
Εναλλακτική A2 (NOC)	576803	1.000	323.000	68.300	99.654
Εναλλακτική A3 (NOC)	576803	1.000	576.000	166.540	127.843
Εναλλακτική A1 (R)	576803	-16.460	177.794	5.476	4.426
Εναλλακτική A2 (R)	576803	- 1.937	35.162	3.090	2.590
Εναλλακτική A3 (R)	576803	- 1.937	35.162	3.672	1.700
Εναλλακτική A1 (FO)	576803	0.000	119.000	18.370	20.667
Εναλλακτική A2 (FO)	576803	0.000	108.000	20.210	23.251

Εναλλακτική Α3 (FO)	576803	0.000	250.000	19.040	25.586
Εναλλακτική Α1 (WFI)	576802	0.000	1.000	0.376	0.172
Εναλλακτική Α2 (WFI)	543514	0.000	0.500	0.083	0.100
Εναλλακτική Α3 (WFI)	540218	0.000	0.857	0.248	0.195

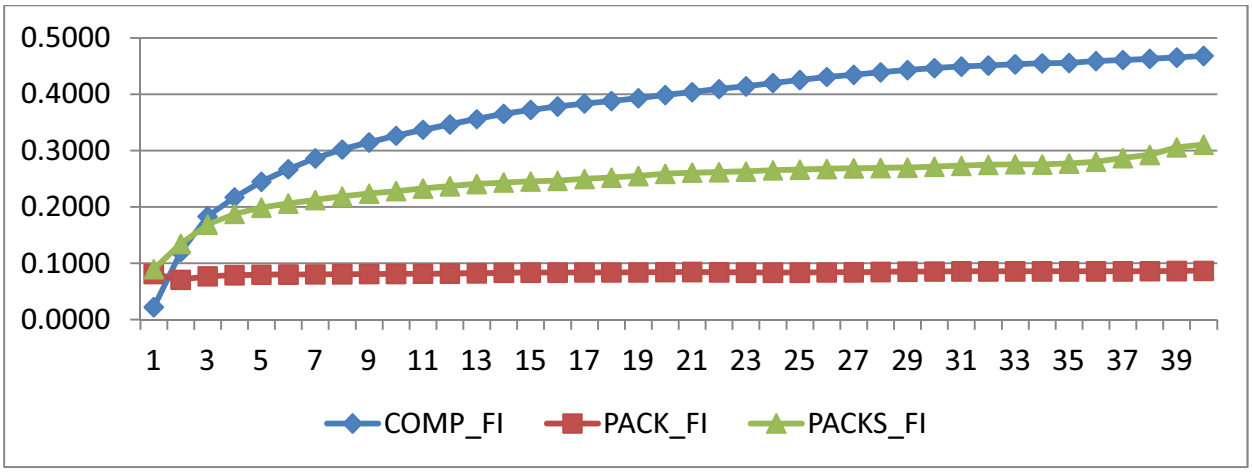
Στα σχήματα 28 έως 31 παρουσιάζουμε τα γραφήματα που συμβολίζουν το μέσο μέγεθος (NOC), το μέσο αριθμό εξαρτήσεων (Fan Out), τη μέση λειτουργικότητα (Fan In) και τη μέση επαναχρησιμοποιησιμότητα (R) των «υποψήφιων συστατικών» που επιλέχθηκαν με τις τρεις εναλλακτικές μεθόδους. Στον άξονα των x τοποθετείται ο αριθμός του μεγέθους των «υποψήφιων συστατικών», ενώ στον άξονα των y τοποθετείται η τιμή της εκάστοτε μέτρησης (χαρακτηριστικού ποιότητας).



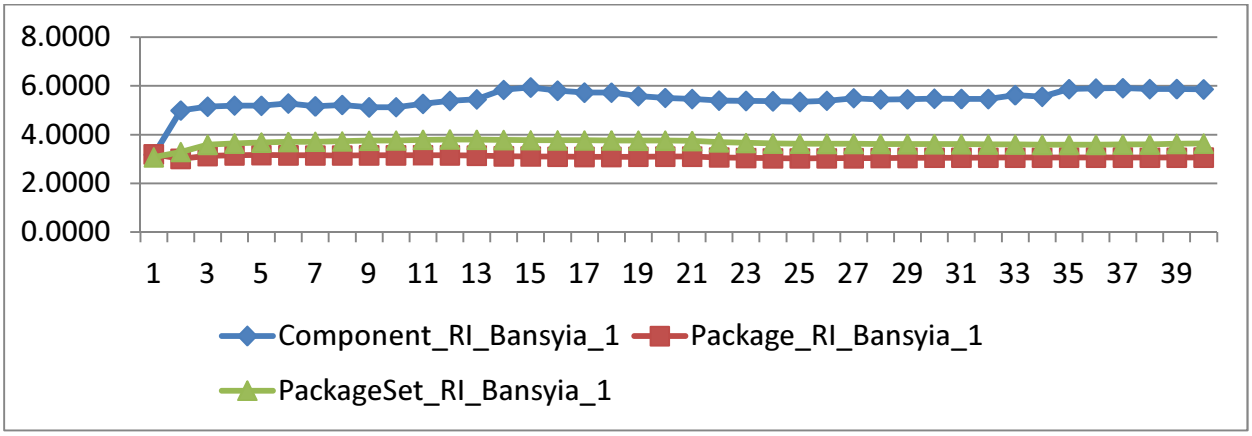
Σχήμα 28. Μέγεθος (noc) των "Υποψήφιων Συστατικών"



Σχήμα 29. Εξαρτήσεις (Fan Out) των "Υποψήφιων Συστατικών"



Σχήμα 30. Λειτουργικότητα (Fan In) των "Υποψήφιων Συστατικών"



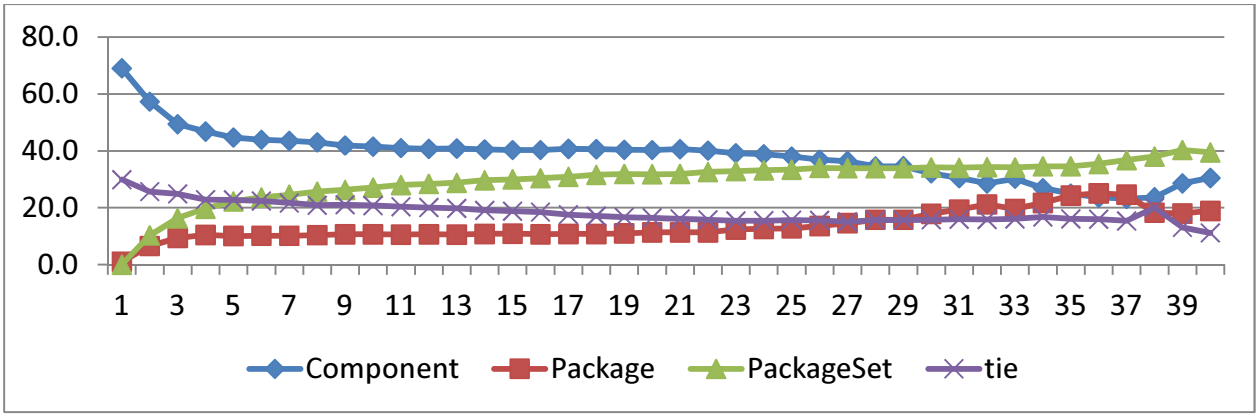
Σχήμα 31. Επαναχρησιμοποιησιμότητα R των "Υποψήφιων Συστατικών"

Στον παρακάτω πίνακα 17 παρουσιάζουμε το ποσοστό των περιπτώσεων όπου κάποια από τις εναλλακτικές A1, A2, A3 είναι η βέλτιστη στρατηγική επιλογής "υποψήφιου συστατικού" βασισμένη στις τιμές του μεγέθους, εξαρτήσεων, λειτουργικότητας και επαναχρησιμοποιησιμότητας.

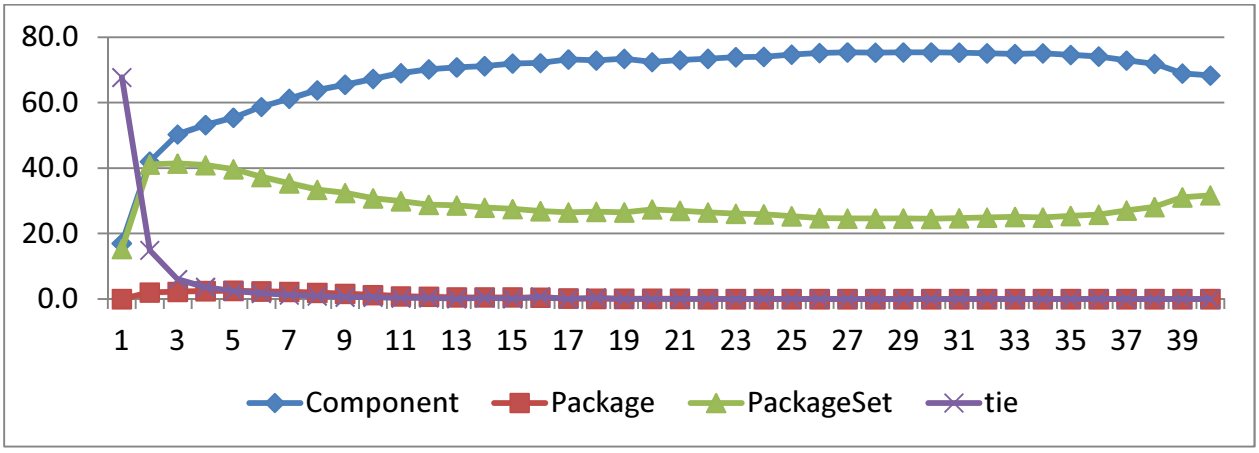
Πίνακας 17. Βέλτιστη επιλογή Εναλλακτικής

	Fan Out (Εξαρτήσεις)	**Fan In (Λειτουργικότητα)**	**Reusability (Επαναχρησιμοποιησιμότητα)**
Εναλλακτική A1 (NOC)	37.8%	69.2%	64.2%
Εναλλακτική A2(NOC)	13.7%	0.6%	15.3%
Εναλλακτική A3 (NOC)	30.3%	29.1%	13.7%
Ισοπαλία	18.1%	1.1%	6.7%

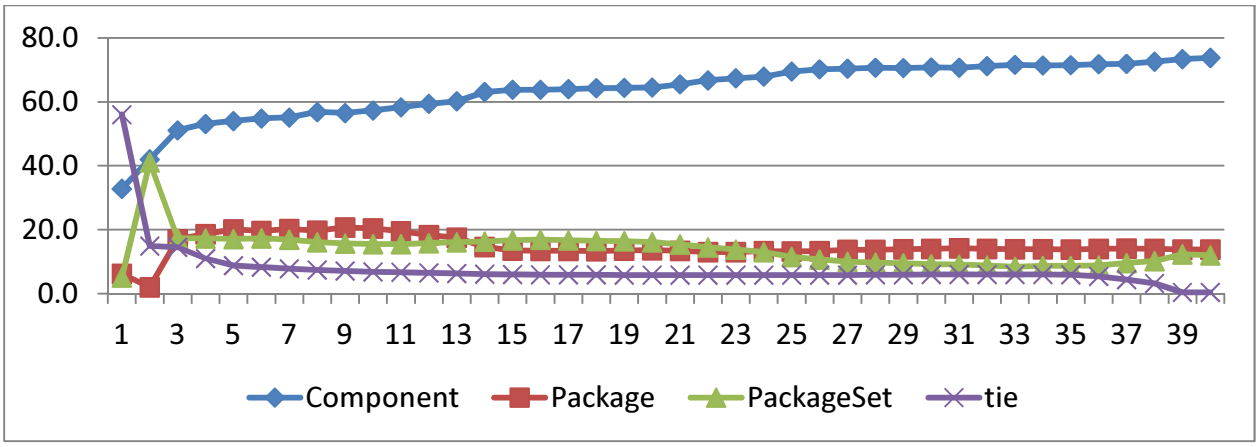
Ακολουθεί μια γραφική αναπαράσταση των πληροφοριών του πίνακα 16, λαμβάνοντας υπόψη το μέγεθος του "υποψήφιου συστατικού" που ανακτήθηκε βάση της Εναλλακτικής Α1, στα σχήματα 32 έως 34. Στις γραφικές παραστάσεις ο άξονας των χ παίζει τον ρόλο του μεγέθους του "υποψήφιου συστατικού" και ο άξονας των y αντιπροσωπεύει την τιμή της μέτρησης. Τα αποτελέσματα στην πλειονότητα των περιπτώσεων παρουσιάζουν την εναλλακτική Α1 ως την βέλτιστη στρατηγική επιλογής ενός συνόλου κλάσεων λαμβάνοντας υπόψη όλες τις μετρικές.



Σχήμα 32 Συχνότητα Εμφάνισης Βέλτιστης Στρατηγικής Επιλογής Κλάσεων (Εξαρτήσεις - Fan Out)



Σχήμα 33. Συχνότητα Εμφάνισης Βέλτιστης Στρατηγικής Επιλογής Κλάσεων (Λειτουργικότητα - Fan In)



Σχήμα 34. Συχνότητα Εμφάνισης Βέλτιστης Στρατηγικής Επιλογής Κλάσεων (Επαναχρησιμοποιησιμότητα -R)

Παρατηρώντας τα παραπάνω διαγράμματα γίνεται σαφές πως υπάρχει ένα εύρος στο μέγεθος των "υποψήφιων συστατικών" όπου με την προτεινόμενη μεθοδολογία μπορούμε να ανακτήσουμε "υποψήφια συστατικά" με τα βέλτιστα ποιοτικά χαρακτηριστικά. Βασιζόμενοι στο χαρακτηριστικό των εξαρτήσεων (Fan Out - FO) η προτεινόμενη μεθοδολογία συνιστά την επιλογή "υποψήφιων συστατικών" με μέγεθος μικρότερο των είκοσι-εννέα (29) κλάσεων. Επιπρόσθετα, εάν λάβουμε υπόψη τα χαρακτηριστικά της λειτουργικότητας (Fan In) και επαναχρησιμοποιησιμότητας (R), η προτεινόμενη μεθοδολογία συστήνει την επιλογή "υποψήφιων συστατικών" με πλήθος κλάσεων μεγαλύτερο του τρία (3). Οπότε εάν το "υποψήφιο συστατικό" έχει πλήθος κλάσεων μεταξύ 3 και 29, ο προγραμματιστής θα προτιμήσει την ανάκτηση του μέσω της εφαρμογής της εναλλακτικής Α1 με σκοπό την επαναχρησιμοποίηση.

3.3. Ενδεικτική Εφαρμογή της Μεθοδολογίας

Για την παρουσίαση μιας ενδεικτικής εφαρμογής της μεθοδολογίας, θεωρούμε πως ο προγραμματιστής σχεδιάζει ένα παιχνίδι στρατηγικής τύπου “Risk” και θέλει να υλοποιήσει λειτουργίες και οντότητες που σχετίζονται με τις «Χώρες». Ο προγραμματιστής χρησιμοποιεί την μηχανή αναζήτησης του Percerons (www.percerons.com) ταξινομώντας τα αποτελέσματα των «υποψήφιων συστατικών» βάση του μεγέθους (noc) και των εξαρτήσεών τους (Fan Out). Οι μετρήσεις των ποιοτικών χαρακτηριστικών για το επιλεγμένο «υποψήφιο συστατικό» παρουσιάζονται στον επόμενο πίνακα σε σύγκριση με τις μετρήσεις για το αντίστοιχο πακέτο

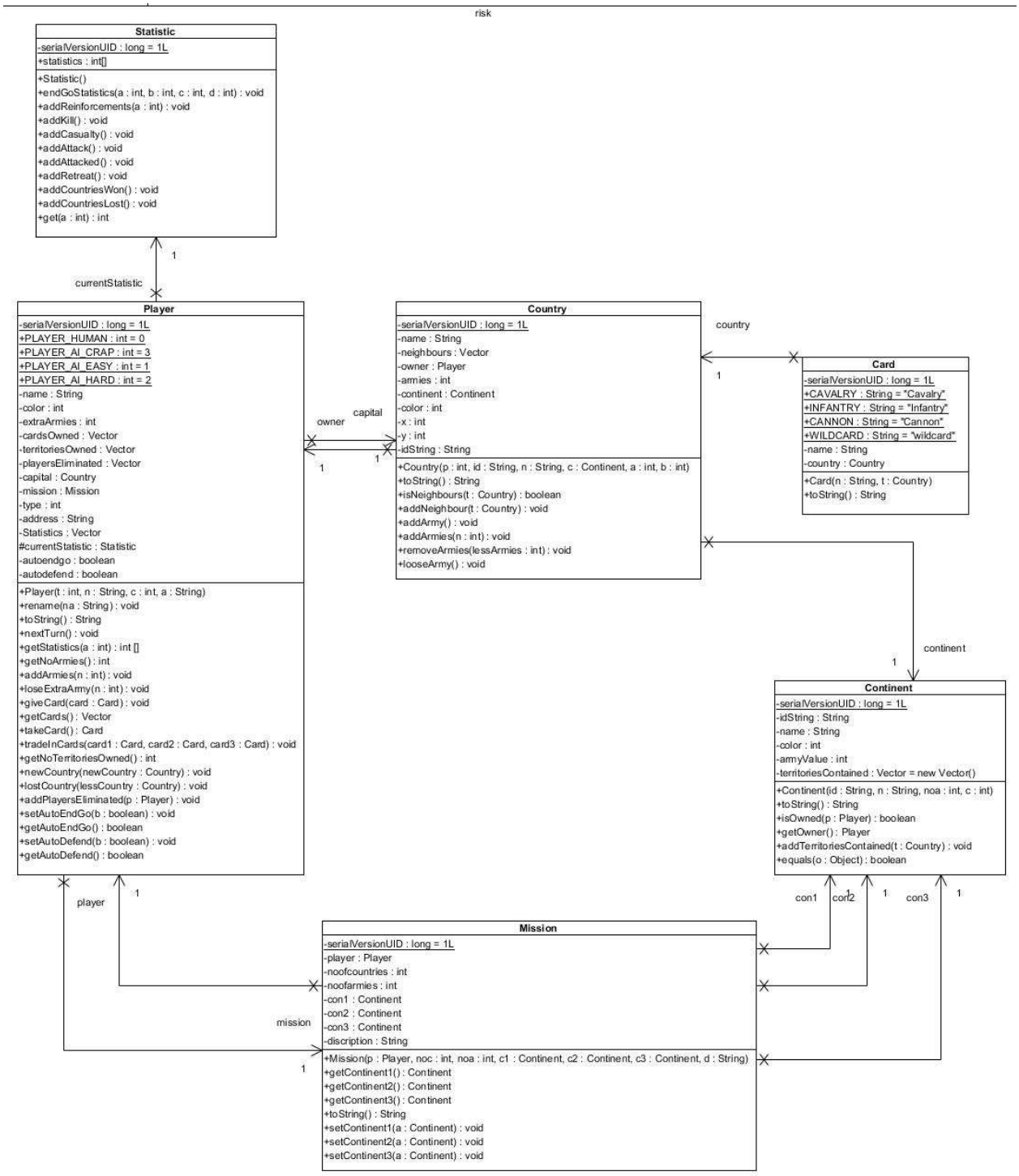
net.yura.domination.engine.core. Από την σύγκριση είναι εμφανείς οι καλύτερες οι τιμές των ποιοτικών χαρακτηριστικών του «υποψήφιου συστατικού» από αυτές του αντίστοιχου πακέτου.

Πίνακας 18. Σύγκριση Ποιοτικών Χαρακτηριστικών μεταξύ των Εναλλακτικών Α1 και Α2

	«Υποψήφιο Συστατικό»	**Πακέτο**
Μέγεθος (noc)	6	7
Εξαρτήσεις (Fan Out)	0	2
Λειτουργικότητα (Fan In)	22	7
Επαναχρησιμοποιησιμότητα (R)	8.8283	9.56

Το «υποψήφιο συστατικό» που εξήχθη προσφέρει λειτουργίες όπως «χώρα» και διαχείριση «ηπείρων», ορίζοντας συνορεύουσες χώρες, πρωτεύουσες περιοχών, στρατιωτικές μονάδες και έλεγχο αποστολών. Επιπρόσθετα ανανεώνει αυτόματα τα στατιστικά του παιχνιδιού μετά από κάθε κίνηση ενός παίκτη του παιχνιδιού. Το «υποψήφιο συστατικό» δεν συμμετέχει στη «μηχανική» του παιχνιδιού ούτε στην γραφική του αναπαράσταση. Επομένως, ο προγραμματιστής μπορεί να δημιουργήσει ο ίδιος την «μηχανική του παιχνιδιού» όπως την επιθυμεί, ορίζοντας τους κανόνες του παιχνιδιού και σχεδιάζοντας την γραφική του αναπαράσταση. Το διάγραμμα κλάσεων του προτεινόμενου «υποψήφιου συστατικού» παρουσιάζεται στο σχήμα 35. Ενώ αμέσως παρακάτω παρουσιάζεται ο κώδικας που γράφετε από τον προγραμματιστή έτσι ώστε να δοκιμάσει τις λειτουργίες του «υποψήφιου συστατικού».

```
public class RiskExecutionScenario {
    public static void main(String[] args) {
        Continent europe = new Continent("1", "Europe", 10, 1);
        Country france = new Country(1, "1", "France", europe, 0, 0);
        Country spain = new Country(2, "2", "Spain", europe, 15, 15);
        Continent asia = new Continent("2", "Asia", 15, 2);
        Country india = new Country(3, "3", "India", asia, 50, 50);
        Country china = new Country(4, "4", "China", asia, 40, 100);
        Card card1 = new Card("Infantry", india);
        Card card2 = new Card("Cannon", france);
        Card card3 = new Card("Cavalry", spain);
        Card card4 = new Card("Infantry", china);
        Player player = new Player(1, "angor", 1, "player1address");
        player.addArmies(5);
        player.setCapital(france);
}
```



Σχήμα 35. Διάγραμμα κλάσεων του προτεινόμενου "Υποψήφιου Συστατικού" από το project “Risk”

3.4. Κίνδυνοι Εγκυρότητας

Η ενότητα αυτή ασχολείται με την παρουσίαση των εσωτερικών και εξωτερικών κινδύνων που μπορούν να επηρεάσουν την εγκυρότητα της έρευνάς μας. Κατ' αρχάς, εφόσον τα αντικείμενα μελέτης της δικής μας μελέτης περίπτωσης είναι εφαρμογές ανοιχτού λογισμικού, τα αποτελέσματα ίσως να μην αφορούν αντίστοιχες εφαρμογές κλειστού λογισμικού. Είναι σημαντικό να αναφερθεί ότι το μέγεθος του δείγματος που πήραμε είναι αρκετά μικρό συγκριτικά με το συνολικό πλήθος των εφαρμογών ανοιχτού λογισμικού, διότι η μελέτη έχει γίνει αρχικά μόνο στην κατηγορία των παιχνιδιών, και η γενίκευση των αποτελεσμάτων που προέκυψαν από το δείγμα για το τι ισχύει στο σύνολο του πληθυσμού δεν αποτελεί πολύ ασφαλή τρόπο όσον αφορά την εγκυρότητα των αποτελεσμάτων. Επίσης, το σύνολο των δεδομένων που μελετήθηκαν ήταν αποκλειστικά παιχνίδια γραμμένα σε Java, επειδή τα εργαλεία που χρησιμοποιούνται για την ανίχνευση προτύπων σχεδίασης καθώς και τα εργαλεία που χρησιμοποιούνται για την ανάλυση των εξαρτήσεων εύρεσης μετρικών μπορούν να αναλύσουν μόνο αρχεία μεταγλωττισμένου κώδικα java (class files). Επιπλέον, χρησιμοποιήθηκε μόνο ένα αποθετήριο κώδικα, που ονομάζεται sourceforge (www.sourceforge.net).

3.5. Συμπεράσματα

Η Πτυχιακή εργασία παρουσιάζει μία μεθοδολογία που μπορεί να χρησιμοποιηθεί για την εξαγωγή "υποψήφιων συστατικών λογισμικού" από παιχνίδια ανοιχτού λογισμικού. Η μέθοδος εξαγωγής των "υποψήφιων συστατικών" βασίζεται στις εξαρτήσεις μεταξύ των κλάσεων με σκοπό την ελαχιστοποίηση των εξωτερικών προς το "υποψήφιο συστατικό" εξαρτήσεων. Η μεθοδολογία αξιολογήθηκε συγκρίνοντας 577.319 "υποψήφια συστατικά λογισμικού" με τα αντίστοιχα πακέτα των κλάσεών τους. Στην πλειονότητα των περιπτώσεων, τα εξαγμένα με την μεθοδολογία "υποψήφια συστατικά", εμφανίζονταν να έχουν το μικρότερο αριθμό εξωτερικών εξαρτήσεων, τη μέγιστη λειτουργικότητα και τη μέγιστη επαναχρησιμοποιησιμότητα. Τα "υποψήφια συστατικά" αποθηκεύονται σε μία βάση δεδομένων με σκοπό την αξιοποίησή τους από κάθε ενδιαφερόμενο.

4. Βιβλιογραφία

1. Chatzigeorgiou A. (2005), “Object-Oriented Design: UML, Principles, Patterns and Heuristics”, Kleidarithmos, Athens, 1st edition.
2. Alexander C., Ishikawa S., Silverstein M. (1977), “A Pattern Language – Town, Buildings, Construction”, Oxford University Press, New York.
3. Gamma E, Helms R, Johnson R, Vlissides J. (1995), “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley Professional, Reading, MA, 1st edition.
4. Sommerville I. (2009), "Software Engineering", Harlow: Addison Wesley, 8th edition.

5. Αναφορές

1. H. A. Amiot, P. Cointe, Y. G. Gueheneuc. and N. Jussien, “Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together”, *Proceedings of the 16th IEEE international conference on Automated software engineering,* ACM, pp.166, San Diego, California, 26-29 November 2001
2. A. Ampatzoglou, A. Kritikos, G. Kakarontzas and I. Stamelos, “An Empirical Investigation on the Reusability of Design Patterns and Software Packages”, Journal of Systems and Software, Elsevier, December 2011, volume 84, pp. 2265-2283.
3. A. Andreou and M. Tziakouris, “A quality framework for developing and evaluating original software components”, Information and Software Technology, Elsevier, 49 (2), pp. 122-141, February 2007.
4. G. Antoniol, G. Casazza, M .Di Penta and R .Fiutem, “Object-Oriented design patterns recovery”, *Journal of Systems and Software,* Elsevier, 59 (2), pp.181-196, November 2001
5. G. Antoniol, R. Fiutem and L. Christoforetti, “Using Metrics to Identify Design Patterns in Object-Oriented Software”, *Proceedings of the 5th International Symposium on Software Metrics,* IEEE, pp. 23, Bethesda, Maryland, 20-21 March 1998
6. A. Asencio, S. Cardman, D. Harris and E. Laderman, “Relating Expectations to Automatically Recovered Design Patterns”, *Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02),* pp. 87, Richmond, Virginia, 29 October – 01 November 2002.
7. Z. Balanyi and R. Ferenc, “MiningDesign Patterns from C++ Source Code”, *Proceedings of the International Conference on Software Maintenance,* IEEE, Amsterdam, The Netherlands, 22-26 September 2003
8. J. Bansiya, C. Davis, “A Hierarchical Model for Object-Oriented Design Quality Assessment”, IEEE Transaction on Software Engineering, IEEE Computer Society, 28 (1), pp. 4-17, 2002.
9. J. Barnard, “A new reusability metric for object-oriented software”, Software Quality Journal, Springer, 7 (1), pp. 35-50, March 1998
10. B. Barnes, T. Durek, J. Gaffney, and A. Pyster. A Framework and Economic Foundation for Software Reuse. In: Tutorial: Software Reuse: Emerging Technology, ed.W. Tracz. Washington: IEEE Computer Society, 1988. pp.77-88.
11. T. Birgerstaff and C. Richter, Reusability Framework, Assessment, and Directions, IEEE Soflware, vol. March. pp. 41-49, 1987.
12. E.S. Cho, M.S. Kim and S.D. Kim, “Component Metrics to Measure Component Quality”, Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC’ 01), IEEE Computer Society, pp. 419-426, Seul, South Korea, 4-7 December 2001.
13. G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino and M. Risi, “Design Pattern Recovery by Visual Language Parsing”, *Proceedings of the 29th international conference on Software Engineering,* IEEE, pp 102-111, Manchester, UK, 21-23 March 2005

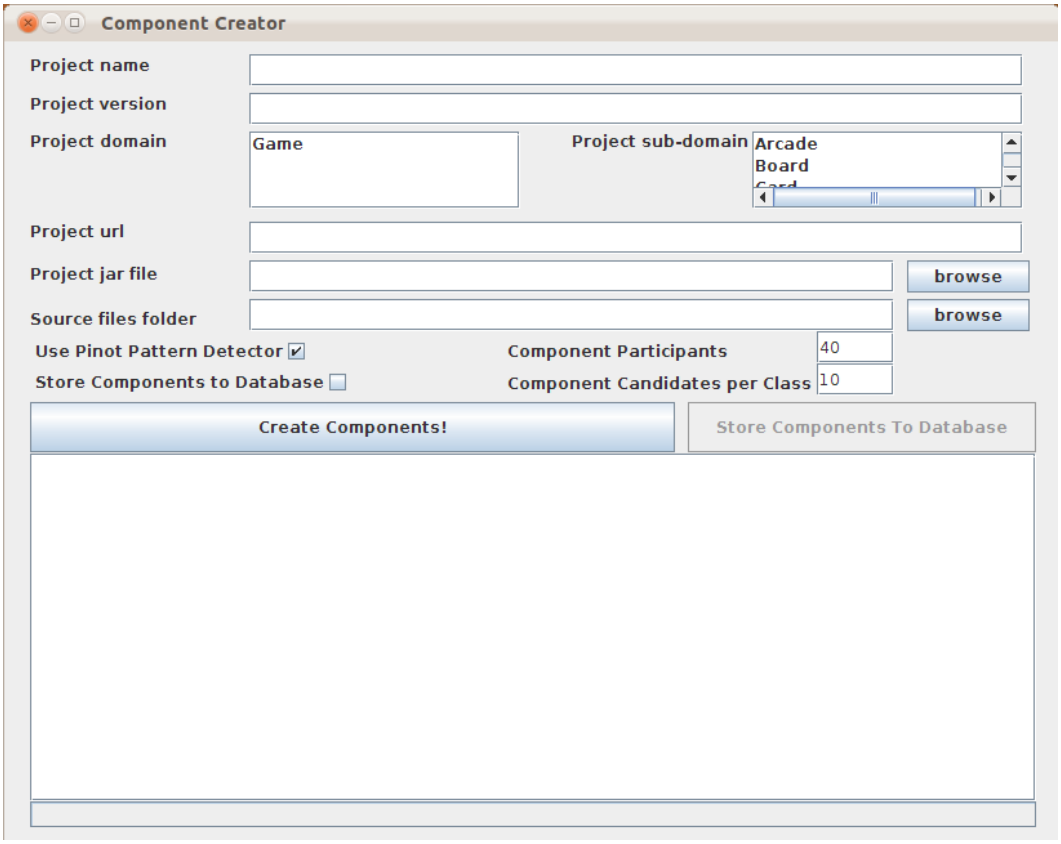
14. G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino and M. Risi, “Case Studies of Visual Language Based Design Patterns Recovery”, *Proceedings of the Conference on Software Maintenance and Reengineering,* IEEE, pp. 165-174, Bari, Italy*,* 22-24 March 2006
15. S. Counsell, S. Swift and J. Crampton, “The interpretation and utility of three cohesion metrics for object-oriented design”, Transactions on Software Engineering and Methodology, Association of Computing Machinery, 15 (2), pp.123-149, April 2006.
16. A. De Lucia, V. Deufemia, C. Gravino and M. Risi, “Behavioral Pattern Identification through Visual Language Parsing and Code Instrumentation”, *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering,* IEEE, pp. 99-108, Kaiserslautern, Germany, 24-27 March 2009
17. A. De Lucia, V. Deufemia, C. Gravino and M. Risi, “A Two Phase Approach to Design Pattern Recovery”, *Proceedings of the 11th European Conference on Software Maintenance and Reengineering,* IEEE, pp. 297-306, Amsterdam, the Netherlands, 21-23 March 2007
18. L. H. Etzkorn, S. E. Gholston, J. L. Fortune, C. E. Stein, D. Utley, P. A. Farrington, G. W. Cox, “A comparison of cohesion metrics for object-oriented systems”, Information and Software Technology, 46 (10), pp. 677-687, August 2004.
19. S.A. Fahmi and H.J. Choi, “Life cycles for component based software development”, Proceedings of the 8th Conference on Computer and Information Technology, IEEE Computer Society, pp. 637-642, pp. 637-642, Sydney, Australia, 8-11 July 2008.
20. Feller J., Fitzgerald B. (2002), “Understanding open source software development”, Addison-Wesley Longman, Boston, MA, 1st edition
21. X. Franch and J. P. Carvallo, “Using Quality Models in Software Package Selection”, Software, IEEE Computer Society, 20 (1), pp. 34-41, January/ February 2003.
22. M. Genero, E. Manso, A. Visaggio, G. Canfora and Mario Piattini, “Building measure-based prediction models for UML class diagram maintainability”, Empirical Software Engineering, Springer, 12 (5), pp. 517 – 549, October 2007.
23. G. Gui and P.D. Scott, “Ranking reusability of software components using coupling metrics”, Journal of Systems and Software, Elsevier, 80 (9), pp. 1450-1459, September 2007.
24. E. Henry and B. Faller, "Large-Scale Industrial Reuse to Reduce Cost and Cycle Time", Software, IEEE Computer Society, 12 (5), pp. 47–53, September 1995.
25. N.L. Hsueh, P.H. Chu, W. Chu, “A quantitative approach for evaluating the quality of design patterns”, Journal of Systems and Software, Elsevier, 81 (8), August 2008, pp 1430-1439.
26. S. Jansen, S. Brinkkemper, I. Hununk and C. Demir, “Pragmatic and opportunistic reuse in innovative start-up companies”, Software, IEEE Computer Society, 25 (6), pp. 2-9, November/December 2008.
27. KeyesM. (1999), "Architecture-driven component reuse", Information and Software Technology, Elsevier, 41, pp.963-968, 1999

28. F. Khomh and Y.G. Gueheneuc, “Do design patterns impact software quality positively?”, IEEE Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR 2008), pp.274-278, 1-4 April 2008, Athens, Greece
29. Kitchenham B., Pickard L., Pfleeger S.L. (1995), “Case Studies for Method and Tool Evaluation”, *In IEEE Software*, Vol. 12, No 04, pp. 52-62.
30. C. Kramer and L. Prechelt, “Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software”, *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96),* IEEE, pp. 208, Monterey, CA, 8-10 November 1996.
31. J. Kontio, “A case study in applying a systematic method for COTS selection”, Proceedings of the 28th International Conference on Software Engineering (ICSE’06), Association of Computing Machinery, pp. 201-209, Shanghai, China, 20-28 May 2006.
32. A. Marcus, D. Poshyvanyk and R. Ferenc, “Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems”, Transaction on Software Engineering, IEEE Computer Society, 34 (2), pp. 287-300, March/April 2008.
33. M.D. McIllroy, Mass-produced software components, in: P. Naur, B.Randell (Eds.), Software Engineering; Report on a Conference by the NATO Science Committee (Garmisch, Germany, October), Software Engineering Concepts and Techniques, Petrocelli/Charter, Belgium, 1976, pp. 88–98.
34. S. McConnell, “Rapid Development: Taming Wild Software Schedules”, Microsoft Press, Redmond, Washington, USA, 1996.
35. A. Mockus, “Large-scale code reuse in open-source software”, 1st International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS’07), IEEE Computer Society, pp. 7-12, Minesota, USA, 20-26 May 2007.
36. M. Morison, C. Tully and M. Ezra, “Diversity in Reuse Processes”, *Software,* IEEE Computer Society, 17 (4), pp. 56-63, July/August 2000.
37. J. Niere, W. Schafer, J. P. Wadsack, L. Wendehals and J. Welsh, “Towards pattern-based design recovery”, *Proceedings of the 24th International Conference on Software Engineering,,* IEEE, pp. 338-348, Orlando, Florida, 19-25 May 2002 .
38. H. M. Plague, L. H. Etzkorn, S. Gholston and S. Quattlebaum, “Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes”, Transactions on Software Engineering, IEEE Computer Society, 33 (6), pp. 402 – 419, June 2007.
39. Samoladas I., Stamelos I, Angelis L., Oikonomou A. (2004), “Open source software development should strive for even greater code maintainability”, In Communications of the ACM, Association of Computing Machinery, Vol. 47, No 10, pp. 83-87.

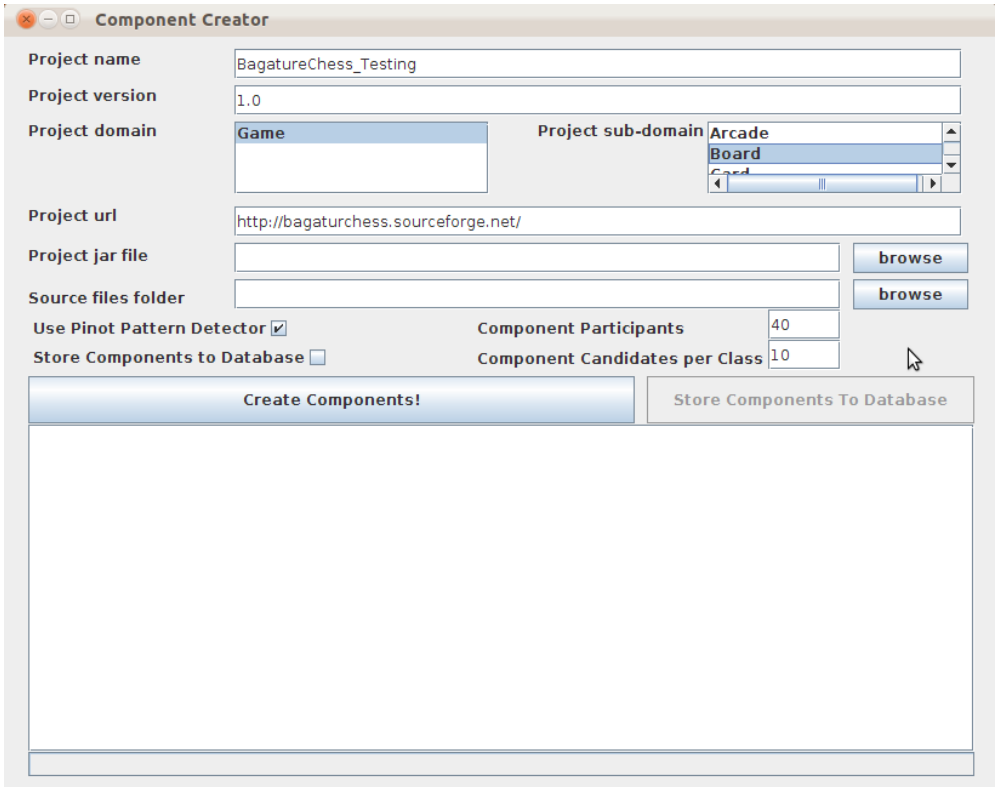
40. P.S. Sandlhu, H. Kaur and A. Singh, “Modelling reusability of object-oriented software systems”, World Academy of Sciences Engineering and Technology, Academic Science Research, 56, pp. 162-165, August 2009.
41. W. Schafer, R. Prieto-Dfaz, and M. Matsumoto. Software Reusability, W. Schiifer, R. Prieto-Dhz, and M. Matsumoto (Eds). Hemel Hempstead: Ellis Horwood, 1994.
42. N. Shi and R. A. Olsson, “Reverse Engineering of Design Patterns from Java Source Code”, *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering,* ACM, pp. 123-134, Tokyo, Japan, 18-22 September 2006
43. Staringer W., " Constructing Applications from Reusable Components ", *IEEE Software, 1994, pp. 61-68*
44. Tsantalis N., Chatzigeorgiou V, Stephanides G., Halkidis S. T. (2006). "Design Pattern Detection using Similarity Scoring", *In IEEE Transaction on Software Engineering,* IEEE Computer Society, Vol. 32, No 11, pp. 896-909.
45. L.Yu, K. Chen and S. Ramaswamy, “Multiple-parameter coupling metrics for layered component based software”, Software Quality Journal, Springer, 17 (1), pp. 5-24, March 2009.
46. W. Wang and V. Tzerpos, “Design Pattern Detection in Eiffel Systems”, *Proceedings of the 12th Working Conference on Reverse Engineering,* pp.165-174, Pittsburgh, Pennsylvania, 07-11 November 2005
47. H. Washizaki, H. Yamamoto and Y. Fukazawa, “A metric suite for measuring reusability of software components”, Proceedings of the 9th International Software Metrics Symposium (METRICS’03), IEEE Computer Society, pp. 211-223, Sydney, Australia, 03-05 September 2003.
48. L. Wendehals and A.Orso, “Recognizing behavioural patterns at run time using finite automata”, *Proceedings of the 2006 international workshop on Dynamic systems analysis (ICSE’06*), IEEE, pp.33-40, Shanghai, China, 23 May 2006
49. Wohlin C., Runeson P., Host M., Ohlsson M.C., Regnell B., Wesslen A. (2000), “Experimentation in Software Engineering”, *Kluwer Academic Publishers*, Boston/ Dordrecht/ London, 1st edition.

Παράρτημα 'Α - Στιγμιότυπα οθόνης (screen shots) από την εκτέλεση του εργαλείου

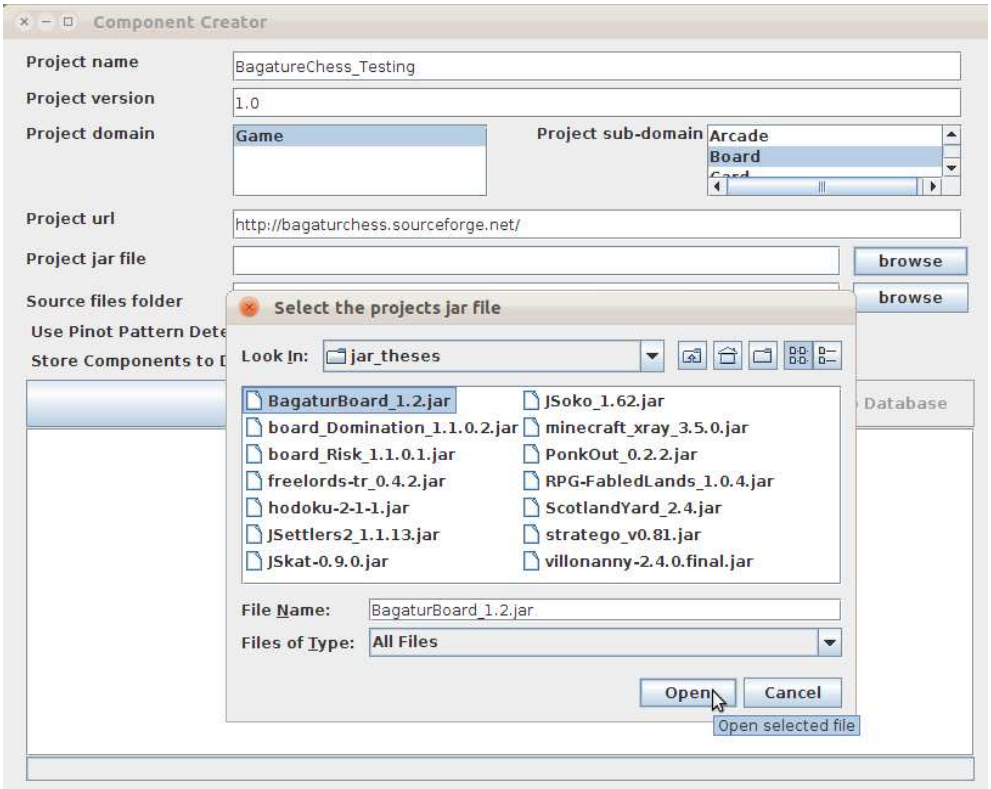
Τα στιγμιότυπα οθόνης (screen shots) που παραθέτουμε αποθηκεύτηκαν κατά τη διάρκεια ανάλυσης του Project "BagatureChess" με αριθμό κλάσεων 142 από το οποίο παράχθηκαν 24.312 "υποψήφια συστατικά" εκ των οποίων 55 βασίζονται σε πρότυπα σχεδίασης.



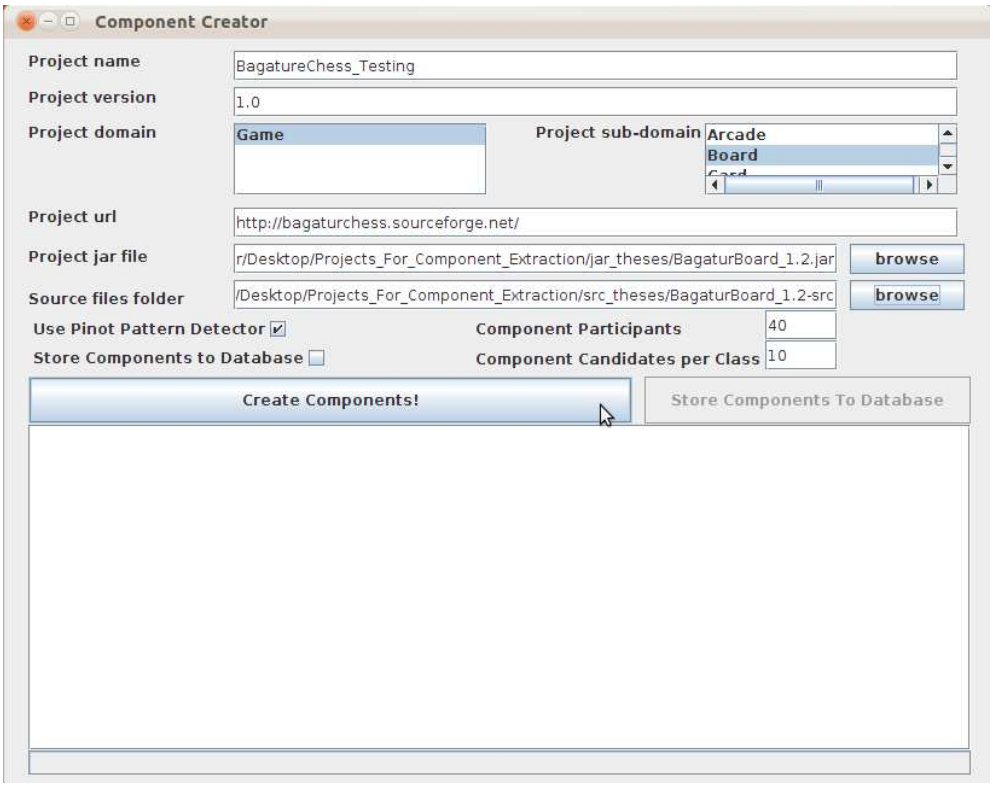
Σχήμα 36. Στιγμιότυπο από την εκτέλεση του εργαλείου



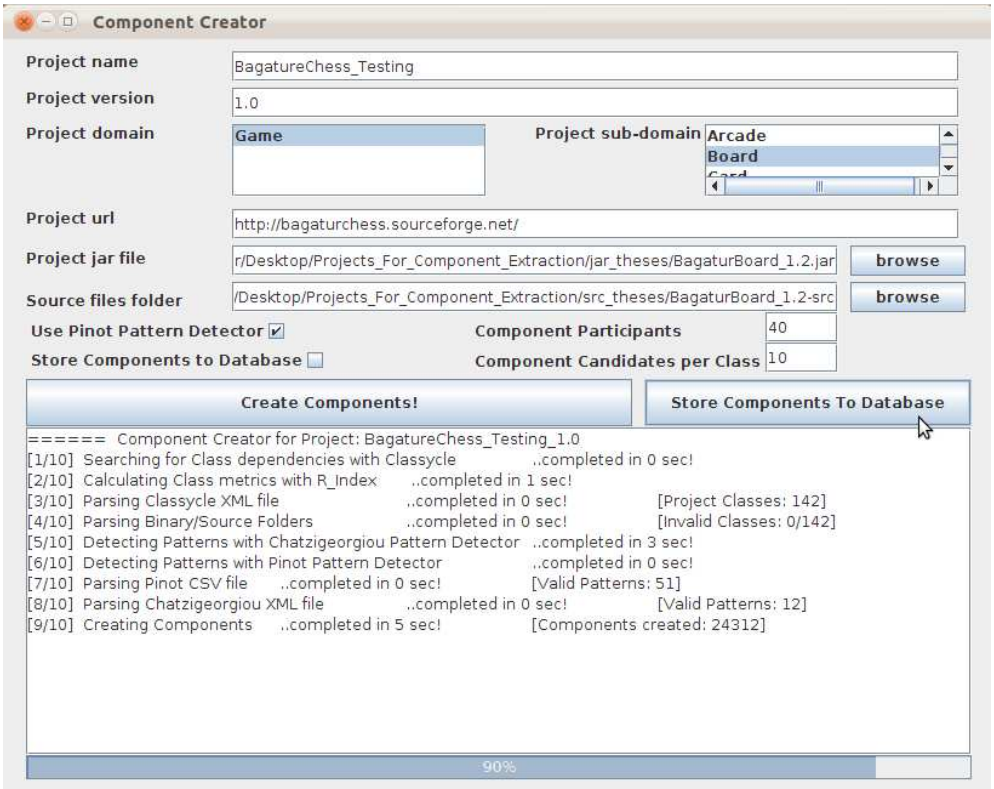
Σχήμα 37. Στιγμιότυπο κατά την εκτέλεση της ανάλυσης του project "Bagature Chess"



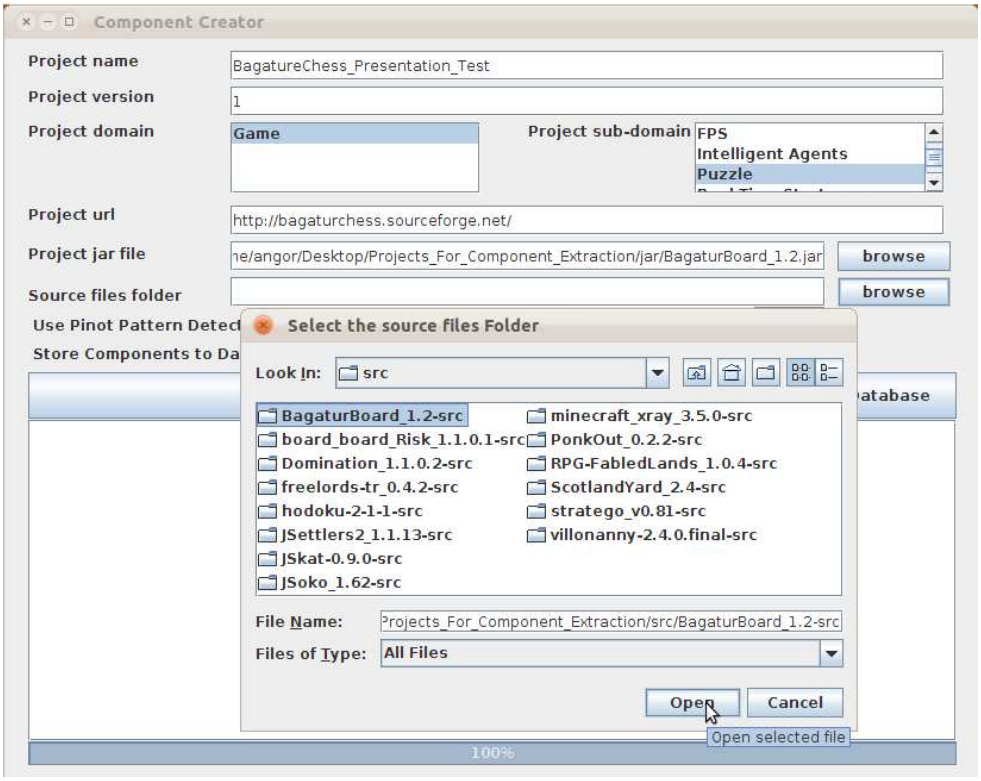
Σχήμα 38. Στιγμιότυπο κατά την εκτέλεση της ανάλυσης του project "Bagature Chess"



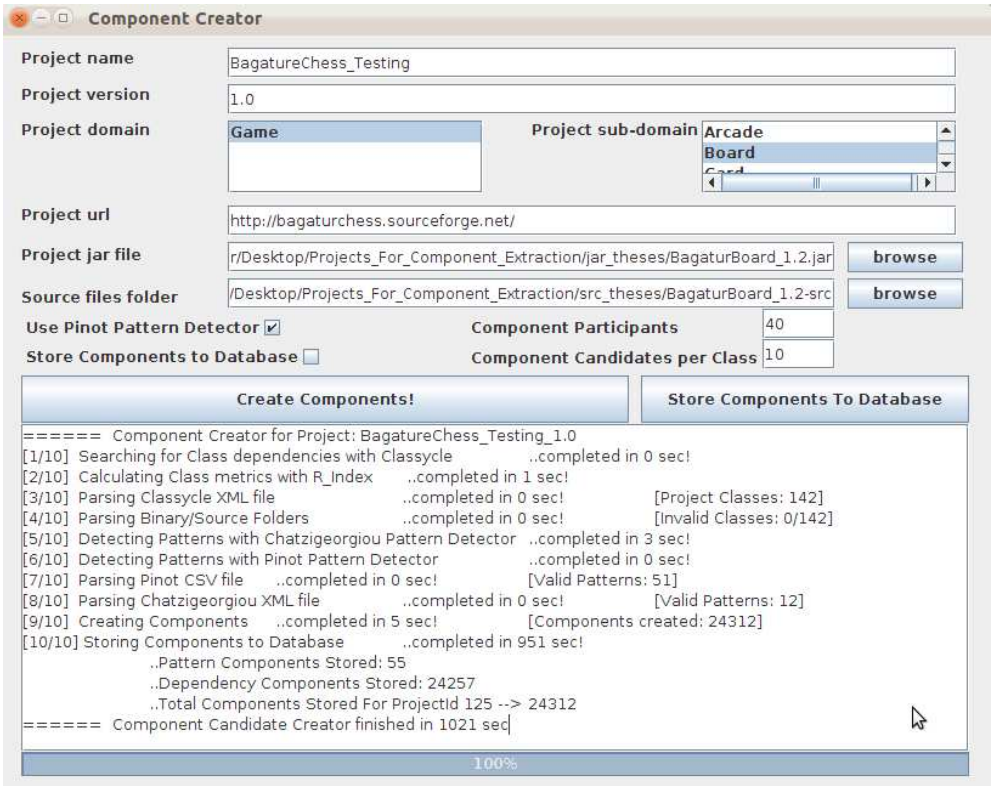
Σχήμα 39. Στιγμιότυπο κατά την εκτέλεση της ανάλυσης του project "Bagature Chess"



Σχήμα 40. Στιγμιότυπο κατά την εκτέλεση της ανάλυσης του project "Bagature Chess"



Σχήμα 41. Στιγμιότυπο κατά την εκτέλεση της ανάλυσης του project "Bagature Chess"



Σχήμα 42.Στιγμιότυπο κατά την εκτέλεση της ανάλυσης του project "Bagature Chess"

Παράρτημα 'Β – Εργαλεία που χρησιμοποιήθηκαν

Ένα πλήθος εφαρμογών και τεχνολογιών -κυρίως ανοιχτού λογισμικού- χρησιμοποιήθηκαν για την ανάπτυξη του εργαλείου «Ανίχνευσης και Ανάκτησης Υποψήφιων Συστατικών». Μία σύντομη περιγραφή τους καθώς και οι σύνδεσμοι για την τοποθεσία τους παρατίθενται παρακάτω.

- **Netbeans IDE** – Χρησιμοποιήθηκε για την ανάπτυξη του εργαλείου και την εγγραφή του κώδικα σε γλώσσα java. (http://www.netbeans.org/)
- **Eclipse IDE** – Χρησιμοποιήθηκε για την ανάπτυξη του εργαλείου και την εγγραφή του κώδικα σε γλώσσα java. (http://www.eclipse.org/)
- **MySql Community Server** - Χρησιμοποιήθηκε για την δημιουργία της βάσης δεδομένων στην οποία αποθηκεύονταν όλες οι παραγόμενες, από το εργαλείο, πληροφορίες. (http://www.mysql.com/downloads/mysql/)
- **MySql Workbench** – Χρησιμοποιήθηκε για την διαχείριση της βάσης δεδομένων. (http://www.mysql.com/downloads/workbench/)
- **Gephi** – Χρησιμοποιήθηκε για την δημιουργία και την απεικόνιση των γράφων που παρουσιάζονται στην Πτυχιακή εργασία. (http://gephi.org/)
- **Gimp** – Χρησιμοποιήθηκε για την επεξεργασία και μορφοποίηση των εικόνων και των σχημάτων που παρουσιάζονται στην Πτυχιακή εργασία. (http://www.gimp.org/)
- **LibreOffice** - Χρησιμοποιήθηκε για την συγγραφή του κειμένου της Πτυχιακής εργασίας καθώς και των γραφικών παραστάσεων που παρουσιάζονται στο κεφάλαιο των αποτελεσμάτων. (http://www.libreoffice.org/download/)
- **Visual Paradigm** – Χρησιμοποιήθηκε για την δημιουργία των διαγραμμάτων κλάσεων της Πτυχιακής εργασίας. (http://www.visual-paradigm.com/)
- **Microsoft Office** - Χρησιμοποιήθηκε για την συγγραφή του κειμένου της Πτυχιακής εργασίας καθώς και των γραφικών παραστάσεων που παρουσιάζονται στο κεφάλαιο των αποτελεσμάτων.