Κατηγοριοποίηση Σχεδιαστικών Προτύπων Αντικειμενοστρεφούς Σχεδίασης Βασισμένη σε Μετρικές Ποιότητας

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Παναγιώτη Γιαννάκη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

# ΑΦΙΕΡΩΣΗ

Στους γονείς μου

# ΕΥΧΑΡΙΣΤΙΕΣ

Ένα μεγάλο ευχαριστώ στον επιβλέποντα καθηγητή μου κ. Απόστολο Ζάρρα για την εμπιστοσύνη, την συνεχή παρακολούθηση και καθοδήγηση κατά την διάρκεια της εκπόνησης της μεταπτυχιακής μου εργασίας. Οι επισημάνσεις και συμβουλές του ήταν καθοριστικές για την επιτυχή ολοκλήρωση του μεταπτυχιακού προγράμματος σπουδών μου. Ενώ μέσω της άριστης συνεργασίας μας απέκτησα γνώσεις και εμπειρία που συμπλήρωσαν τον μεταπτυχιακό κύκλο σπουδών μου.

Στους γονείς μου οι οποίοι στήριξαν και στηρίζουν τις επιλογές μου καθ' όλη την διάρκεια της ακαδημαϊκής μου πορείας. Η συμπαράσταση και η υπομονή που επέδειξαν, αποτέλεσαν ένα σπουδαίο βοήθημα για την ολοκλήρωση της μεταπτυχιακής μου εργασίας και τέλος στην αδελφή μου για την συμπαράσταση της κατά την διάρκεια των σποδών μου.

# ΠΕΡΙΕΧΟΜΕΝΑ

# ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

# ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

# ΠΕΡΙΛΗΨΗ

Παναγιώτης Γιαννάκης του Γεωργίου και της Διαμάντως. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιανουάριος, 2010. Κατηγοριοποίηση Σχεδιαστικών Προτύπων Αντικειμενοστρεφους Σχεδιασης Βασισμένη σε Μετρικές Ποιότητας.

Επιβλέποντας: Ζαρρας Απόστολος.

Τα σχεδιαστικά πρότυπα (software design patterns) παρέχουν έτοιμες λύσεις σε προβλήματα σχεδίασης λογισμικού. Ο σκοπός των προτύπων είναι η αποτύπωση σχεδιαστικής εμπειρίας που προέρχεται από την σχεδίαση παλαιότερων συστημάτων λογισμικού. Παράλληλα προάγουν και την επαναχρησιμοποίηση των λύσεων αυτών σε παρόμοια προβλήματα. Ο αυξημένος αριθμός των σχεδιαστικών προτύπων αναδεικνύει την ανάγκη οργάνωσης και κατηγοριοποίησης τους με τέτοιον τρόπο ώστε να γίνεται η ευκολότερη η εύρεση του κατάλληλου προτύπου για την αντιμετώπιση ενός δεδομένου σχεδιαστικού προβλήματος. Όμως τα κριτήρια των κατηγοριοποιήσεων αυτών δεν χρησιμοποιούν κάποιο μετρίσιμο μέγεθος. Συνέπεια αυτού είναι οι κατηγοριοποιήσεις που προκύπτουν να εμπεριέχουν υποκειμενικές κρίσεις.

Στην παρούσα εργασία αναλύουμε την συνεισφορά των προτύπων στην βελτίωση μιας σχεδίασης βάσει μετρικών ποιότητας. Η προαναφερθείσα ανάλυση οδηγεί στην δημιουργία μίας κατηγοριοποίησης τα κριτήρια της οποίας είναι οι μετρικές ποιότητας. Τέλος χρησιμοποιώντας συνδυαστική ανάλυση αναδεικνύονται πιο λεπτομερώς οι ιδιότητες που έχουν τα σχεδιαστικά πρότυπα.

# EXTENDED ABSTRACT IN ENGLISH

Giannakis, Panagiotis, G. MSc, Computer Science Department, University of Ioannina, Greece. January, 2010. A Design Patterns Classification Based on Quality Metrics.

Thesis Supervisor: Zarras Apostolos.

Software design patterns describe proven solutions to recurring software design problems. The main idea behind design patterns is to support the reuse of design information thus allowing developers to communicate more effectively. A growing number of people consider design patterns to be a promising approach to system development, which addresses the aforementioned problems, especially in object-oriented systems. New design patterns are being discovered, described and applied by several research groups. Knowledge of these patterns increases designers' abilities, leads to cleaner and more easily maintained software, speeds up implementation and test, and helps programmers document and communicate their designs.

Over the past years, the number of documented software design patterns has increased greatly. Multitude brings with it a need to organize and classify. Many research groups have proposed diferrentes types of classifications but these classifications are not based on objective criteria and they do not quantify the object oriented concept.

In this thesis we measure the contribution of GoF design patterns in the software design process. Moreover we measure this contribution in terms of object – oriented metrics. By this way we can classify the patterns using more objective criteria. We developed a clustering algorithm that associates every pattern with a feature vector.

These vectors have components a value for every object – oriented metric. Finally, we used Apriori algorithm to find frequent items sets that associate with a set of patterns. By this way we can examine the attributes of the patterns in-depth.

# ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ



## 1.1. Στόχοι

Τα σχεδιαστικά πρότυπα περιγράφουν μια αποδεδειγμένα καλή λύση για την αντιμετώπιση ενός προβλήματος σχεδίασης λογισμικού. Βασικό σκοπό έχουν να αποτυπώσουν εμπειρική γνώση για την αντιμετώπιση ενός προβλήματος έτσι ώστε η λύση αυτή να είναι επαναχρησιμοποιήσιμη. Η γνώση των προτύπων αυτών, βελτιώνει τις δεξιότητες των σχεδιαστών λογισμικού. Πιο συγκεκριμένα, οδηγεί στην χρήση απλών τρόπων σχεδίασης ενώ διευκολύνει και τις διαδικασίες συντήρησης ενός λογισμικού. Παράλληλα επιταχύνει την διαδικασία της εφαρμογής αλλά και της δοκιμής ενός λογισμικού. Τέλος διευκολύνει την επικοινωνία μεταξύ προγραμματιστών με στόχο την κατανόηση τις σχεδίασης που ακολουθεί ο κάθε ένας.

Με το πέρασμα των χρόνων, ο αριθμός των προτύπων αυξήθηκε σημαντικά. Έτσι λοιπόν εμφανίστηκε η ανάγκη οργάνωσης αυτών των προτύπων με στόχο την ευκολότερη εύρεση του κατάλληλου προτύπου για την αντιμετώπιση ενός προβλήματος. Μέχρι τώρα στην βιβλιογραφία υπάρχουν διάφορες κατηγοριοποιήσεις που χρησιμοποιούν διαφορετικά κριτήρια ενώ παράλληλα έχουν και διαφορετικούς σκοπούς.

Στόχος της παρούσας εργασίας είναι η δημιουργία μια νέας κατηγοριοποίησης για τα πρότυπα του κατέγραψε η GoF [11]. Η κατηγοριοποίηση αυτή ως βασικό κριτήριο θα έχει την συμβολή του κάθε προτύπου στην βελτίωση της σχεδίασης ενός συστήματος.

Ακόμα λαμβάνει υπόψη και την επιβάρυνση που μπορεί να προκαλέσει σε κάποιες μονάδες του συστήματος το κάθε πρότυπο. Τέλος βασικό στοιχείο της κατηγοριοποίησης είναι να επηρεάζεται όσο τον δυνατόν λιγότερο από υποκειμενικά στοιχεία.

### 1.2. Δομή της Διατριβής

Η διατριβή περιέχει 6 κεφάλαια και η οργάνωση της παρουσιάζεται παρακάτω: στο κεφάλαιο 2 αναλύεται το θεωρητικό υπόβαθρο του θέματος που πραγματεύεται η διατριβή. Θέματα δηλαδή που έχουν να κάνουν με την έννοια και τις αρχές της αντικειμενοστρεφούς σχεδίασης, ακόμα γίνεται αναφορά στα σχεδιαστικά πρότυπα ως έννοια αλλά και ως χρησιμότητα στα πλαίσια της σχεδίασης ενός λογισμικού. Τέλος παρουσιάζονται θέματα κατηγοριοποίησης σχεδιαστικών προτύπων. Στο κεφάλαιο 3 παρουσιάζεται η προηγούμενη δουλειά που έχει γίνει για το συγκεκριμένο θέμα. Δηλαδή παρουσιάζεται ένα σύνολο κατηγοριοποιήσεων που αποτελείται από αυτές της GoF [11], του Tichy [22], του Zimmer [23] και του Eckel [10]. Ακόμα γίνεται και μια κριτική ανασκόπηση των κατηγοριοποιήσεων αυτών προκειμένου να αναδειχθούν τα δυνατά και τα αδύνατα τους σημεία και να γίνει πιο κατανοητή η συνεισφορά της συγκεκριμένης διατριβής. Στο κεφάλαιο 4 γίνεται η αξιολόγηση των προτύπων βάσει των κριτηρίων που έχουν τεθεί ενώ παράλληλα παρουσιάζεται και η μεθοδολογία που θα ακολουθηθεί για την κατηγοριοποίηση των προτύπων. Στο κεφάλαιο 5 παρουσιάζονται τα αποτελέσματα της ανάλυσης των δεδομένων του προηγούμενου κεφαλαίου. Τέλος το κεφάλαιο 6 αποτελεί τον επίλογο και παρουσιάζει μια σύντομη ανακεφαλαίωση της διατριβής.

# ΚΕΦΑΛΑΙΟ 2. ΥΠΟΒΑΘΡΟ



## 2.1. Αντικειμενοστρεφής Σχεδίαση

Η έννοια της αντικειμενοστρεφούς σχεδίασης είναι άρρηκτα συνδεδεμένη με τον αντικειμενοστρεφή προγραμματισμό. Αποτελεί μια τεχνική ανάπτυξης λογισμικού βάσει της οποίας τα επιμέρους τμήματα του συστήματος εκφράζονται ως αντικείμενα, ενώ παράλληλα περιγράφει και την αλληλεπίδραση μεταξύ των αντικειμένων αυτών. Η χρήση μιας αντικειμενοστρεφούς γλώσσας προγραμματισμού σε συνδυασμό με το αντικειμενοστρεφές μοντέλο σχεδίασης παρέχουν ένα πλαίσιο ανάπτυξης λογισμικού που απλοποιεί τις διαδικασίες συντήρησης χωρίς επιπτώσεις στην ποιότητα του λογισμικού.

### *2.1.1. Χαρακτηριστικά της Σχεδίασης*

Όπως προαναφέρθηκε, η χρήση της αντικειμενοστραφούς σχεδίασης οδηγεί στην υιοθέτηση εννοιών που έχουν σχέση με τον αντικειμενοστρεφή προγραμματισμό. Έννοιες όπως “στοιχείο αφαίρεσης”, “κλάση” και “μέθοδοι”, περιγράφουν τα επιμέρους τμήματα του λογισμικού ενώ τα αντικείμενα αποτελούν τις δομικές μονάδες του συστήματος. Επιπλέον, προκειμένου να υπάρξει σαφής καθορισμός, η αντικειμενοστρεφής σχεδίαση περιλαμβάνει έννοιες που χαρακτηρίζουν την εσωτερική της ποιότητα χρησιμοποιώντας όρους αντικειμενοστρεφή

προγραμματισμού. Τέτοιες έννοιες είναι η συνεκτικότητα, η σύζευξη, η κληρονομικότητα, η απόκρυψη πληροφοριών και η ενθυλάκωση. Στην συνέχεια παρουσιάζουμε τις συγκεκριμένες έννοιες εκτενέστερα.

Η *συνεκτικότητα* (cohesion) χαρακτηρίζει την εσωτερική συνάφεια των κλάσεων του συστήματος. Πιο συγκεκριμένα, η συνεκτικότητα μετράει τον βαθμό σχετικότητας των διαφορετικών μεθόδων μιας κλάσης. Δηλαδή εξετάζει τα ιδιοχαρακτηριστικά και πως αυτά σχετίζονται με το σύνολο μεθόδων της κλάσης. Έτσι λοιπόν μια κλάση έχει υψηλή συνεκτικότητα όταν οι μέθοδοι της σχετίζονται με την αντιμετώπιση κοινού προβλήματος. Τέλος η υψηλή συνεκτικότητα βοηθάει στην δημιουργία εύκολα κατανοητού και επαναχρησιμοποιήσιμου κώδικά.

Η *σύζευξη* (coupling) μετράει την αλληλεξάρτηση που παρουσιάζουν οι δομικές μονάδες του συστήματος. Δυο αντικείμενα παρουσιάζουν σύζευξη στην περίπτωση που ανταλλάζουν μηνύματα. Ενώ αποστολή μηνύματος από ένα αντικείμενο σε κάποιο άλλο σημαίνει την κλήση μιας μεθόδου του άλλου αντικειμένου. Δηλαδή η σύζευξη μετράει τη διασύνδεση των δομικών ενοτήτων που υπάρχουν σε ένα λογισμικό.

Η *κληρονομικότητα* (inheritance) αποτελεί μηχανισμό βάση του οποίου ένα αντικείμενο αποκτά χαρακτηριστικά που ανήκουν σε κάποιο άλλο. Κάνοντας χρήση της κληρονομικότητας είναι δυνατόν να πραγματοποιηθεί ο διαμοιρασμός ιδιοχαρακτηριστικών και μεθόδων μεταξύ κλάσεων που είναι οργανωμένες σε μια ιεραρχία.

Η *απόκρυψη πληροφορίας* (information hiding) είναι η διαδικασία βάση τις οποίας δεδομένα ενός αντικειμένου χαρακτηρίζονται ως ιδιωτικά (private). Με αυτόν τον τρόπο τα δεδομένα αυτά δεν είναι προσβάσιμα από μεθόδους που βρίσκονται εξωτερικά της κλάσης.

Επιπλέον, η *ενθυλάκωση* (encapsulation) αποτελεί μηχανισμό απόκρυψης των λεπτομερειών της υλοποίησης ενός αντικειμένου. Τα ιδιοχαρακτηριστικά αλλά και οι μέθοδοι μιας κλάσεις στις οποίες επιτρέπεται η πρόσβαση ορίζουν την *διεπαφή*

(interface) της κλάσης μέσω της οποίας η συγκεκριμένη κλάση επικοινωνεί με τις υπόλοιπες. Με αυτόν τον τρόπο η ενθυλάκωση υποστηρίζει τον διαχωρισμό της διεπαφής από τις υλοποιήσεις. Τέλος η ενθυλάκωση στοχεύει στην ελαχιστοποίηση των εξαρτήσεων μεταξύ των δομικών μονάδων ενός συστήματος που με την σειρά της οδηγεί στην συντήρηση του συστήματος με λιγότερο κόστος.

*2.1.2. Αρχές της Σχεδίασης*

Η διαδικασία της αντικειμενοστρεφούς σχεδίασης υποστηρίζεται από ένα σύνολο αρχών. Οι αρχές αυτές παρέχουν ένα σύνολο οδηγιών οι οποίες βοηθούν στην αποφυγή κακής σχεδίασης ενώ η δημιουργία τους οφείλεται στην συσσωρευμένη γνώση και εμπειρία πολλών χρόνων και διαφορετικών προσώπων ενώ σκοπός τους είναι η υπόδειξη πρακτικών καλής σχεδίασης αλλά και λάθη προς αποφυγή. Στην συνέχεια παρουσιάζουμε επτά αρχές οι οποίες είναι οι εξής: η αρχή της μοναδικής αρμοδιότητας, της αντικατάστασης Liskov, της αντιστροφής εξαρτήσεων, της ανοιχτής – κλειστής σχεδίασης και του διαχωρισμού των διασυνδέσεων, ενώ υπάρχουν και δυο πιο γενικές αρχές οι οποίες είναι αυτή της ενσωμάτωσης και της χαμηλής ζεύξης.

Η *αρχή της μοναδικής αρμοδιότητας* (single – responsibility principle) στοχεύει στην εξασφάλιση υψηλής συνεκτικότητας των δομικών μονάδων ενός συστήματος. Σύμφωνα με την αρχή αυτή, μια κλάση πρέπει να έχει μόνο ένα λόγο για να αλλάξει [9]. Πιο συγκεκριμένα, μια και μόνο αρμοδιότητα μπορεί να χαρακτηρίζει κάθε κλάση δηλαδή να εξυπηρετεί ένα και μόνο σκοπό. Έτσι λοιπόν οι αλλαγές των απαιτήσεων του συστήματος έχουν την μορφή αλλαγής των αρμοδιοτήτων των κλάσεων. Σε περίπτωση που μία κλάση έχει περισσότερες από μια αρμοδιότητες θα έχει περισσότερους από έναν λόγους να αλλάξει. Όμως σε τέτοιες περιπτώσεις οι αλλαγές σε μια αρμοδιότητα μπορεί να εμποδίσουν αλλαγές σε κάποια άλλη αρμοδιότητα ή ακόμα να δημιουργηθούν και σφάλματα στις υπόλοιπες αρμοδιότητες, καθιστώντας την διαδικασία συντήρησης δύσκολη.

Η *αρχή της υποκατάστασης* (Liskov substitution principle) υποστηρίζει την δημιουργία σωστά δομημένων ιεραρχιών κλάσεων. Σύμφωνα με αυτήν την αρχή, οι

παράγωγοι τύποι πρέπει να μπορούν να υποκαθιστούν τους βασικούς τύπους. Έτσι λοιπόν κάθε υποκλάση Β μια κλάσης Α θα πρέπει να έχει όμοια συμπεριφορά με αυτήν της κλάσης Α. Πιο συγκεκριμένα, μια κλάση Β αποτελεί παράγωγο τύπο της κλάσης Α, μόνο στην περίπτωση που κάθε κώδικας Κ, ο οποίος λειτουργεί με αντικείμενα Α, συμπεριφέρεται με των ίδιο τρόπο και με αντικείμενα Β [14]. Τέλος σε περίπτωση που υπάρξει παραβίαση της συγκεκριμένης αρχής τα προγράμματα πελάτες μπορεί να λειτουργούν ορθά με μια βασική κλάση, όμως μπορεί να πέφτουν σε σφάλματα χρησιμοποιώντας παράγωγες κλάσεις.

Η *αρχή της αντιστροφής εξαρτήσεων* (dependency inversion principle) έχει ως στόχο την μείωση των εξαρτήσεων μεταξύ στοιχείων υψηλού και χαμηλού επιπέδου. Πιο συγκεκριμένα οι μονάδες υψηλού επιπέδου δεν θα πρέπει να εξαρτώνται από μονάδες χαμηλού επιπέδου ενώ οι λεπτομέρειες θα πρέπει να εξαρτώνται από στοιχεία αφαίρεσης [19]. Δηλαδή κάθε κλάση υψηλού επιπέδου επικοινωνεί με κλάσεις χαμηλού επιπέδου μέσω ενός στοιχείου αφαίρεσης και έτσι η κλάση υψηλού επιπέδου δεν διατηρεί απευθείας αναφορές σε κλάσεις χαμηλού επιπέδου. Με αυτόν τον τρόπο τα στοιχεία υψηλού επιπέδου δεν είναι ευάλωτα σε αλλαγές που μπορεί να προκύψουν σε στοιχεία χαμηλού επιπέδου και έτσι η συγκεκριμένη αρχή προάγει την επαναχρησιμοποίηση του κώδικα ενώ παράλληλα διευκολύνει και τις διαδικασίες συντήρησης.

Η *αρχή της ανοιχτής – κλειστής σχεδίασης* (open – close principle) θεωρεί ότι σημαντικό ρόλο στην σχεδίαση του συστήματος παίζει η μείωση της σύζευξης μεταξύ των μονάδων. Πιο συγκεκριμένα οι οντότητες λογισμικού θα πρέπει να είναι ανοικτές σε επέκταση, αλλά κλειστές σε τροποποίηση [18]. Ως ανοιχτές σε επέκταση ορίζονται οι κλάσεις οι λειτουργίες των οποίων μπορούν να επεκταθούν για την κάλυψη των νέων απαιτήσεων με κλειστό τρόπο, δηλαδή η επέκταση των λειτουργιών δεν οδηγεί σε αλλαγές του κώδικα. Η βασική έννοια που υποστηρίζει την συγκεκριμένη αρχή είναι η αφαίρεση. Ένα στοιχείο αφαίρεσης χρησιμοποιείται για την απόκρυψη πληροφοριών που έχουν να κάνουν με την υλοποίηση μιας λειτουργία, ενώ σκοπός έχει την καλύτερη κατανόηση των λεπτομερειών και της δομής της λειτουργίας. Ακόμα ένα δομικό στοιχείο μπορεί να συσχετίζεται με ένα στοιχείο αφαίρεσης, η συγκεκριμένη μονάδα μπορεί να είναι κλειστή για τροποποίηση λόγω

της εξάρτισης από την αφαίρεση, όμως είναι ανοιχτή σε επέκταση μέσω νέων υλοποιήσεων του στοιχείου αφαίρεσης. Αυτό σημαίνει ότι η συγκεκριμένη αρχή προάγει την επέκταση των απαιτήσεων ενός συστήματος αποφεύγοντας αλλαγές σε υπάρχον κώδικα και έτσι επιτυγχάνεται η απλοποίηση των διαδικασιών συντήρησης.

Η *αρχή του διαχωρισμού των διασυνδέσεων* (interface segregation principle) υποστηρίζει την ύπαρξη πολλών εξειδικευμένων διασυνδέσεων αντί μιας γενικής [7]. Με αυτόν των τρόπο οι χρήστες της διασύνδεσης δεν έχουν εξάρτηση από μεθόδους που δεν χρησιμοποιούν. Συνεπώς η συγκεκριμένη αρχή αποτρέπει την σύζευξη μεταξύ των χρηστών της διεπαφής. Έτσι αλλαγές στον κώδικα ενός χρήστη δεν επηρεάζουν του υπόλοιπους χρηστές και αυτό γιατί δεν χρησιμοποιούν μια γενική διασύνδεση αλλά εξειδικευμένες διασυνδέσεις που ανταποκρίνονται στις απαιτήσεις του κάθε χρήστη.

Η *αρχή της ενθυλάκωσης* (encapsulation principle) θεωρεί ότι η εσωτερική κατάσταση ενός αντικειμένου πρέπει να είναι τροποποιήσιμη μόνο μέσω της δημόσιας διασύνδεσης του. Δηλαδή οι ιδιοτιμές δεν πρέπει να είναι προσπελάσιμες εξωτερικά τις κλάσης ενώ οι δημόσια διαθέσιμες μέθοδοι παρέχουν την δυνατότητα τροποποίηση των τιμών αυτών. Με αυτόν τον τρόπο οι δομικές ενότητες του συστήματος παραμένουν ανεξάρτητες από τις ιδιότητες οποιασδήποτε κλάσης και έτσι σε περίπτωσης αλλαγής της αναπαράστασης των δεδομένων της κλάσης το σύστημα παραμένει ανεπηρέαστο.

Η *αρχή της χαμηλής ζεύξης* (low coupling principle) υποστηρίζει την ανάγκη για μειωμένο βαθμό εξάρτησης μεταξύ των κλάσεων ενός συστήματος [21]. Πιο συγκεκριμένα σε ένα σχέδιο λογισμικού πρέπει να επιδιώκεται η επίτευξη της μικρότερης δυνατής σύζευξης των συστατικών. Έτσι λοιπόν η επιδίωξη της χαμηλής ζεύξης εξασφαλίζει ανεξαρτησία μεταξύ των δομικών μονάδων του συστήματος. Συνεπώς οι διαδικασίες ελέγχου, συντήρησης αλλά και υλοποίησης πραγματοποιούνται πιο εύκολα.

## 2.2. Μετρικές

### *2.2.1. Εισαγωγή*

Οι μετρικές στα πλαίσια τις σχεδίασης ενός λογισμικού μπορούν να χρησιμοποιηθούν ως εργαλεία ποσοτικοποίησης τις ποιότητας της σχεδίασης. Βάσει των μετρικών είναι δυνατόν να πραγματοποιηθεί έλεγχος ποιότητας της σχεδίασης και να πραγματοποιηθούν αλλαγές προκειμένου να μειωθεί η πολυπλοκότητα ή να αναπτυχθούν οι δυνατότητες.

Η διαδικασίες ελέγχου ποιότητας μπορούν να πραγματοποιηθούν σε διαφορετικά στάδια του κύκλου ζωής ενός λογισμικού. Ανάλογα με την οπτική που ένα λογισμικό παρακολουθείται, μπορούν να χρησιμοποιηθούν διαφορετικές μετρικές. Πιο συγκεκριμένα ο διαχωρισμός των μετρικών γίνεται σε δυο κατηγορίες, οι εσωτερικές μετρικές εξετάζουν το λογισμικό από στατική άποψη και υπολογίζουν μεγέθη που έχουν να κάνουν με χαρακτηριστικά του παραγόμενου κώδικα και μόνο. Αναφορικά με την σχεδίαση οι εσωτερικές μετρικές εξετάζουν χαρακτηριστικά όπως το μέγεθος, την συντηρησιμότητα, την φορητότητα και την δυνατότητα αξιολόγισης θέματα δηλαδή που άπτονται στις ανάγκες των προγραμματιστών. Αντίθετα οι εξωτερικές μετρικές εξετάζουν το λογισμικό ως εκτελέσιμο προϊόν και το κατά πόσο ανταποκρίνεται στο περιβάλλον του. Δηλαδή οι εξωτερικές μετρικές συνδέονται με τις ανάγκες των τελικών χρηστών ενός προγράμματος, ανάγκες όπως η λειτουργικότητα, η ευχρηστία, η αποδοτικότητα, η αξιοπιστία, η ευελιξία, η απλότητα και η φιλικότητα προς τον χρήστη είναι χαρακτηριστικά που εύκολα μπορούν να εντοπιστούν κατά την διάρκεια της χρήσης ενός λογισμικού. Έτσι λοιπόν οι εξωτερικές μετρικές δεν εξετάζουν το λογισμικό ως οντότητα, αλλά ποσοτικοποιούν την αλληλεπίδραση με τους χρήστες και το υπόλοιπο περιβάλλον. Ενώ οι εσωτερικές μετρικές, ποσοτικοποιούν μεγέθη που χαρακτηρίζουν την σχεδίαση και την υλοποίηση ενός λογισμικού.

Η ανάπτυξη της αντικειμενοστραφής σχεδίασης ανέδειξε την ανάγκη ποσοτικοποίησης μεγεθών που χαρακτηρίζουν την έννοια της αντικειμενοστρέφειας. Έτσι λοιπόν πέρα από μετρικές κατάλληλες για δομημένη σχεδίαση, μετρικές δηλαδή

για τις γραμμές κώδικα και γραμμές σχόλιων, υπάρχουν και μετρικές για αντικειμενοστραφή σχεδίαση που υπολογίζουν την κληρονομικότητα, την συνεκτικότητα και την σύζευξη. Η επόμενη ενότητα παρουσιάζει ένα σύνολο μετρικών κατάλληλο για την υποστήριξη της αντικειμενοστραφούς σχεδίασης.

### *2.2.2. Οι μετρικές των Chidamber και Kemerer*

Ένα από τα πιο γνωστά σύνολα μετρικών που έχει προταθεί για την αντικειμενοτρεφή σχεδίαση είναι αυτό των Chidamber και Kemerer [6]. Σύμφωνα με τους εισηγητές του, η χρήση αυτού του μοντέλου βοηθάει τους σχεδιαστές στην λήψη αποφάσεων αναφορικά με θέματα σχεδίασης. Οι μετρικές CK παρουσιάζουν ερευνητικό ενδιαφέρον ενώ παράλληλα τυγχάνουν ευρείας αποδοχής. Οι Chidamber και Kemerer πρότειναν έξι μετρικές η περιγραφή των οποίων γίνεται παρακάτω.

Η μετρική Weighted Method per Class (WMC) υπολογίζει την πολυπλοκότητα της κλάσης. Πιο συγκεκριμένα, προκειμένου να βρούμε την πολυπλοκότητα κάποιας κλάσης μπορούμε να εξετάσουμε τον αριθμό των μεθόδων που υπάρχουν σε αυτήν.
Έτσι λοιπόν δεδομένου μιας κλάσης $C_1$ η οποία έχει μεθόδους $M_1,M_2,...,M_n$ και υποθέτοντας ότι η αντίστοιχη πολυπλοκότητα σε κάθε μεθόδου είναι $c_1,c_2,\ldots,c_n$. τότε το WMC είναι το άθροισμα της πολυπλοκότητας των μεθόδων της κλάσης.
Δηλαδή:

$$WMC(C_1)=\sum_{i=1}^{n} c_i$$

Βάση αυτού του τύπου δεν υπάρχουν λεπτομέρειες για τον υπολογισμό της πολυπλοκότητας κάθε μεθόδου. Σε μια γενική περίπτωση μπορούμε να θεωρήσουμε ότι όλες οι μέθοδοί έχουν την ίδια πολυπλοκότητα. Έτσι λοιπόν το WMC της κλάσης είναι ίδιο με τον αριθμό των μεθόδων.
Δηλαδή:

$$WMC(C_1)=n$$

Όμως είναι γεγονός ότι όλες οι μέθοδοι μιας κλάσης δεν έχουν την ίδια πολυπλοκότητα άρα ο παραπάνω τρόπος υπολογισμού μπορεί να απέχει από την πραγματικότητα. Για μεγαλύτερη ακρίβεια στον υπολογισμό της πολυπλοκότητας

των μεθόδων μπορεί να γίνει χρήση της κυκλωματικής πολυπλοκότητας (McCabe Cyclomatic Complexity) [16]. Σύμφωνα λοιπόν με την προσέγγιση αυτή η πολυπλοκότητα μίας μεθόδου ισοδυναμεί με το άθροισμα των ελέγχων του αντίστοιχου διαγράμματος ροής.

Ο αριθμός των μεθόδων αλλά και η πολυπλοκότητα τους έχουν άμεση σύνδεση με τον χρόνο και την προσπάθεια που πρέπει να καταβληθεί ώστε να δημιουργηθεί αλλά και να συντηρηθεί η κλάση στην όποια υπάρχουν [15]. Επιπλέον όσο μεγαλύτερος είναι ο αριθμός των μεθόδων που έχει μια κλάση τόσο μεγαλύτερο αντίκτυπο έχει στις παράγωγες κλάσεις και αυτό γιατί οι κλάσεις αυτές κληρονομούν αυτές τις μεθόδους. Επιπρόσθετα, κλάσεις με μεγάλο πλήθος μεθόδων είναι πιθανότερο να είναι πιο ειδικά ορισμένες για την συγκεκριμένη εφαρμογή και έτσι περιορίζεται η δυνατότητα επαναχρησιμοποίησης του κώδικα. Ενώ η αύξηση του WMC αυξάνει και την πυκνότητα σφαλμάτων παράλληλα μειώνει την ποιότητα τον κώδικα [4]. Όμως δεν υπάρχει κάποιος ορισμός για τον αριθμό των μεθόδων σε μια κλάση. Η ανάλυση του *Chidamber* δείχνει ότι οι περισσότερες κλάσεις ενός συστήματος έχουν ως 10 μεθόδους ενώ μόνο ένα 10% των κλάσεων μπορούν να έχουν παραπάνω από 24 μεθόδους. Τέλος από τα παραπάνω γίνεται σαφές ότι είναι επιθυμητό να έχουμε μικρή πολυπλοκότητα των κλάσεων δηλαδή το WMC είναι προτιμότερο να είναι μικρό.

Η μετρική Depth of Inheritance Tree (DIT) υπολογίζει το μήκος της μέγιστης διαδρομής από μια κλάση παράγωγο ως την βασική κλάση της ιεραρχίας και βρίσκει εφαρμογή σε ιεραρχίες κλάσεων. Επιπλέον βάσει του DIT μπορεί να υπολογιστεί πόσες κλάσεις προγόνων μπορούν να έχουν επιπτώσεις στην υπό εξέταση κλάση. Σαν ιδιότητα η κληρονομικότητα βοηθάει στην διαχείριση της πολυπλοκότητας και όχι στην αύξησή της. Έτσι λοιπόν τα μεγάλα δένδρα προωθούν την επαναχρησιμοποίηση εκμεταλλευόμενα την κληρονομικότητα. Αυξημένο DIT προκαλεί και μεγάλο αριθμό κληρονομούμενων μεθόδων το οποίο όμως θα έχει ως αποτέλεσμα δυσκολίες στην κατανόηση και στην συντήρηση της κάθε κλάσης [15] ενώ παράλληλα οδηγεί και στην αύξηση της σχεδιαστικής πολυπλοκότητας. Ακόμα, αυξημένο ύψους του δένδρου συνεπάγεται και μεγαλύτερη πιθανότητα αύξησης των σφαλμάτων του κώδικα [4]. Βέβαια αυτό δεν σημαίνει απαραίτητα ότι οι βαθύτερες κλάσεις σε ένα

δένδρο θα έχουν τον μεγαλύτερο αριθμό λαθών. Στην πραγματικότητα οι μεσαίες κλάσεις σε ένα δένδρο είναι πιο επιρρεπείς σε σφάλματα [12]. Σχετικά με την τιμή που πρέπει να έχει το DIT υπάρχουν διάφορες προσεγγίσεις. Οι Chidamber και Kemerer προτείνουν ως επιθυμητό DIT ίσο ή μικρότερο του 10, ενώ μια άλλη προσέγγιση περιορίζει το DIT σε τιμές μικρότερες του 8 [12]. Τέλος εγχειρίδια του Visual Studio .NET προτείνουν το DIT να είναι μικρότερο ή ίσο με 5. Ειδικές περιπτώσεις του DIT έχουμε όταν μια κλάση κληρονομεί απευθείας από το σύστημα μεθόδους τότε το DIT=0 και όταν μια κλάση κληρονομεί από μια άγνωστη κλάση τότε το DIT=2 και αυτό επειδή η άγνωστη κλάση θα κληρονομεί από το σύστημα και το 2 είναι η ελάχιστη τιμή. Στην πραγματικότητα το DIT μπορεί να είναι και μεγαλύτερο από 2.

Γενικεύοντας μπορούμε να πούμε ότι η ύπαρξη DIT στο σύστημα βοηθάει στην μείωση της πολυπλοκότητας ενώ ενισχύει την επαναχρησιμοποίηση του κώδικα και την συντήρηση του συστήματος. Όμως στην περίπτωσης που υπάρξει πολύ μεγάλο DIT οδηγούμαστε σε πολύπλοκες ιεραρχίες με κλάσεις δύσκολα κατανοητές και συντηρίσιμες. Τέλος αυξημένο DIT αυξάνει και την πυκνότητα των σφαλμάτων ενώ μειώνει την ποιότητα του συστήματος.

Η μετρική Number of Children (NOC) υπολογίζει τον αριθμό των παιδιών που κληρονομούν από την κλάση πατέρα και βρίσκει εφαρμογή σε ιεραρχίες κλάσεων. Ο NOC υπολογίζει το πλήθος των παράγωγων κλάσεων που επεκτείνουν μια βασική κλάση, ενώ ο DIT υπολογίζει το ύψος μιας παραγωγής κλάσης από την ρίζα της ιεραρχίας των κλάσεων. Γενικά το ύψος είναι πιο χρήσιμο από το εύρος και αυτό γιατί προωθεί την επαναχρησιμοποίηση σε πολλά επίπεδα μέσω της κληρονομικότητας. Επιπλέον υπάρχει συσχέτιση μεταξύ DIT και NOC και αυτό γιατί οι δυο μετρικές βρίσκουν εφαρμογή μόνο σε ιεραρχίες κλάσεων. Ο αριθμός του NOC δείχνει περίπου το επίπεδο επαναχρησιμοποίησης της βασικής κλάσης [6]. Έτσι λοιπόν σε περίπτωση που έχουμε μεγάλη τιμή NOC τότε έχουμε και μεγάλο βαθμό επαναχρησιμοποίησης. Από την άλλη μεριά το αυξημένο NOC μιας βασικής κλάσης, υπονοεί σύζευξη μεταξύ της συγκεκριμένης κλάσης και περισσότερων υποκλάσεων, έτσι μια αλλαγή σε κάποια μέθοδο της βασικής κλάσης επηρεάζει περισσότερες παράγωγες κλάσεις [15]. Επιπλέον μπορεί να υπάρξει κακή χρήση των υποκλάσεων

και σε αυτήν την περίπτωση είναι πιο χρήσιμη η ομαδοποίηση σχετικών κλάσεων και η εισαγωγή ενός νέου επιπέδου κληρονομικότητα. Αντίθετα μια κλάση με αυξημένο NOC και WMC δηλώνει υψηλή πολυπλοκότητα στην κορυφή της ιεραρχίας των κλάσεων. Μια τέτοια κλάση επηρεάζει έναν μεγάλο αριθμό απογόνων το οποίο μπορεί να σημαίνει κακή σχεδίαση.

Συνοψίζοντας μπορούμε να πούμε ότι το NOC δηλώνει επαναχρησιμοποίηση κώδικά και βοηθάει στην επέκταση του συστήματος. Όμως πολύ μεγάλος αριθμός NOC αντικατοπτρίζει την προσπάθεια που απαιτείται προκειμένου να γίνει ο έλεγχος μια κλάσης ενώ παράλληλα μπορεί να οδηγήσει σε κακή χρήση της κληρονομικότητας.

Η μετρική Coupling Between Objects (CBO) υπολογίζει την σύζευξη μεταξύ κλάσεων αντικειμένων. Πιο συγκεκριμένα μια κλάση έχει σύζευξη με μια άλλη κλάση στην περίπτωση που αντικείμενα των δυο αυτών κλάσεων αλληλεπιδρούν. Δηλαδή στην περίπτωση που έχουμε δυο κλάσεις θεωρούμε ότι υπάρχει σύζευξη όταν μέθοδοι που έχουν οριστεί σε μια κλάση χρησιμοποιούν μεθόδους ή ιδιοχαρακτηριστικά που είναι ορισμένα στην δεύτερη κλάση. Ακόμα η πολλαπλή πρόσβαση σε μια κλάση μετράει σαν μια πρόσβαση δηλαδή το CBO μιας κλάσης διαμορφώνεται από τις μεθόδους και τα ιδιοχαρακτηριστικά που καλεί. Το αυξημένο CBO είναι ανεπιθύμητο, η υπερβολική εξάρτηση μεταξύ των κλάσεων αποτρέπει την επαναχρησιμοποίηση. Έτσι λοιπόν όσο πιο ανεξάρτητες είναι οι κλάσεις μεταξύ τους, τόσο πιο εύκολο είναι να επαναχρησιμοποιηθούν σε άλλες εφαρμογές. Προκειμένου να υποστηριχθεί η ενθυλάκωση και ο πολυμορφισμός οι εξαρτήσεις των κλάσεων πρέπει να παραμένουν χαμηλές. Όσο πιο μεγάλος είναι ο αριθμός των εξαρτήσεων τόσο μεγαλύτερη είναι η ευαισθησία στις αλλαγές σε διάφορα σημεία του συστήματος, συνεπώς και πιο δύσκολη η συντήρηση. Επιπλέον αυξημένο CBO δηλώνει και μεγαλύτερη πιθανότητα σε σφάλματα κώδικα και συνεπώς δημιουργείται η ανάγκη για πιο σχολαστικό έλεγχο. Σχετικά με τις προτεινόμενες τιμές εξάρτησης που μπορεί να έχει μια κλάση, έχει προταθεί ότι το CBO δεν πρέπει να είναι μεγαλύτερο του 14 [10]. Συνοψίζοντας μπορούμε να πούμε ότι ο αριθμός του CBO είναι αντιστρόφως ανάλογος με τον βαθμό επαναχρησιμοποίησης μιας κλάσης. Συνεπώς το CBO της κάθε κλάσης θα πρέπει να είναι όσο το δυνατόν μικρότερο γίνεται.

Η μετρική Response For a Class (RFC) δηλώνει τον αριθμό μεθόδων που πρέπει να κληθούν προκειμένου να απαντηθεί ένα μήνυμα μιας άλλης κλάσης. Ο υπολογισμός του είναι το άθροισμα των μεθόδων της κλάσης με το πλήθος των εξωτερικών μεθόδων που καλούν οι μέθοδοι της κλάσης. Έτσι λοιπόν ο τύπος που χρησιμοποιείται είναι:

$$RFC = M + R$$

Όπου:

M = ο αριθμός των μεθόδων στην κλάση. Μια μέθοδος υπολογίζεται μόνο μια φόρα ακόμα και αν εκτελεστεί πολλές φορές από άλλες μεθόδους

R = ο αριθμός των εξωτερικών μεθόδων που καλεί η κλάση.

Ο RFC ουσιαστικά μετράει το πλήθος των μεθόδων που καλούνται εξωτερικά συνεπώς καταγράφει έμμεσα την επικοινωνία των εμπλεκόμενων κλάσεων. Κλάσεις με αυξημένο RFC παρουσιάζουν και αυξημένη πολυπλοκότητα. Δηλαδή όσο μεγαλύτερο το RFC μιας κλάσης τόσο πιο δύσκολη η συντήρηση της και ο έλεγχος της. Ο λόγος είναι ότι καλώντας έναν μεγάλο αριθμό εξωτερικών μεθόδων είναι πιο δύσκολος ο εντοπισμός κάποιου προβλήματος που θα προκύψει [15].

Συνοψίζοντας αναφορικά με το RFC είναι προτιμότερο να παρουσιάζει μικρές τιμές [4] καθώς έτσι υπάρχει λιγότερη επικοινωνία μεταξύ των κλάσεων ενώ η μικρότερη επικοινωνία των κλάσεων του συστήματος βοηθά τις διαδικασίες συντήρησης και ελέγχου.

Τέλος η μετρική Lack of Cohesion in Methods (LCOM) κάνει χρήση του βαθμού συνάφειας των μεθόδων μιας κλάσης. Υπολογίζει την έλλειψη συνεκτικότητας της κλάσης, πόσο καλά δηλαδή έχει σχεδιαστεί ένα σύστημα και πόσο πολύπλοκη είναι μια κλάση. Ο LCOM υπολογίζει το πλήθος των ζευγαριών των μεθόδων που έχουν μηδενική συνάφεια μείον το πλήθος των ζευγαριών των μεθόδων που έχουν μη μηδενική συνάφεια. Η συνάφεια των μεθόδων φαίνεται από τις μεταβλητές που χρησιμοποιούν. Έτσι λοιπόν για κάθε κλάση C με μεθόδους $M_1, M_2 \ldots M_n$ οι οποίες

χρησιμοποιούν ένα σύνολο μεταβλητών $I_1, I_2 \ldots I_n$. Για κάθε μέθοδο $M_i$ μπορούμε να ορίσουμε μια διαμέριση τέτοια ώστε να ισχύει:

$$P = \{(I_i, I_j) \mid I_i \cap I_j = \varnothing\} \text{ και } Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \varnothing\}$$

Τότε το LCOM υπολογίζεται βάσει του τύπου LCOM = |P| - |Q| αν |P| > |Q| αλλιώς LCOM = 0. Έτσι λοιπόν γίνεται σαφές ότι το LCOM μετράει τα ζεύγη των μεθόδων που έχουν συνάφεια μηδέν. Όσο μικρότερη η τιμή του LCOM μιας κλάσης τόσο πιο εύκολη είναι η συντήρηση της. Ακόμα αν όλες οι μέθοδοι μιας κλάσης χρησιμοποιούν ανεξάρτητα σύνολα ιδιοχαρακτηριστικών τότε η κλάση παρουσιάζει κακή σχεδίαση [15]. Συνοψίζοντας, αναφορικά με την επιθυμητή τιμή του LCOM μπορούμε να πούμε ότι είναι το μηδέν δηλαδή αυτό που επιθυμούμε είναι να μην υπάρχει έλλειψη συνεκτικότητας.

## 2.3. Σχεδιαστικά Πρότυπα

### *2.3.1. Ορισμός της έννοιας*

Ένα *σχεδιαστικό πρότυπο* (design pattern) αποτελεί μια λύση η οποία έχει εφαρμοστεί επιτυχώς σε πολλές περιπτώσεις για την αντιμετώπιση ενός συγκεκριμένου προβλήματος. Ο Christoper Alexander [2] εισήγαγε την έννοια του σχεδιαστικού προτύπου στον χώρο της Αρχιτεκτονικής. Έτσι λοιπόν ένα σχεδιαστικό πρότυπο περιγράφει ένα πρόβλημα το οποίο καλούμαστε να αντιμετωπίσουμε πολλές φορές ενώ στην συνέχεια προτείνει την λύση του προβλήματος. Πέρα όμως από τον χώρο της αρχιτεκτονικής τα σχεδιαστικά πρότυπα ως έννοια μπορούν να προτείνουν λύσεις και στον χώρο της αντικειμενοστρεφούς σχεδίασης. Αντί λοιπόν η περιγραφή των προτύπων να γίνεται με όρους αρχιτεκτονικής όπως πόρτα και σκάλα, πραγματοποιείται με έννοιες όπως αντικείμενο, στοιχείο αφαίρεσης και κλάση.

Ένα πρότυπο στην αντικειμενοστρεφή σχεδίαση περιγράφει ένα συχνά επαναλαμβανόμενο σχεδιαστικό πρόβλημα το οποίο προκύπτει σε συγκεκριμένο σχεδιαστικό πλαίσιο. Επιπρόσθετα παρουσιάζει μια γενική περιγραφή της λύσης,

ενώ στην συνέχεια ακολουθεί η περιγραφή των βασικών δομικών μονάδων, των μεταξύ τους σχέσεων καθώς και των αλληλεπιδράσεων [5]. Η γενική περιγραφή της λύσης φανερώνει ότι το πρότυπο παρέχει μια οδηγία για τον τρόπο επίλυσης του προβλήματος το οποίο μπορεί να προκύψει σε διαφορετικά συστήματα. Ενώ το κάθε σύστημα είναι αυτό που εξειδικεύει την λύση δημιουργώντας την κατάλληλη σχεδίαση που στην συνέχεια μετασχηματίζεται σε κώδικα.

Στόχος των σχεδιαστικών προτύπων είναι η συστηματική καταγραφή λύσεων σε συνηθισμένα προβλήματα λογισμικού. Οι λύσεις αυτές είναι κοινά αποδεκτές καθώς έχει παρατηρηθεί η εφαρμογή τους σε διαφορετικά συστήματα για την αντιμετώπιση του ίδιου προβλήματος. Η περιγραφή του κάθε πρότυπου είναι χωρισμένη σε ενότητες που τονίζουν διαφορετικές πτυχές της έννοιας. Οι ενότητες αυτές περιλαμβάνουν το όνομα (name) το οποίο βοηθάει τον προσδιορισμό και την αναφορά στο πρότυπο, τον σκοπό (intent) για την περιγραφή του προβλήματος αλλά και την προτεινόμενη λύση και τέλος τις συνέπειες (consequences) για την περιγραφή των οφελών αλλά και των επιπτώσεων από την χρήση του προτύπου.

### *2.3.2. Η χρησιμότητα των προτύπων*

Η έννοια του σχεδιαστικού προτύπου έχει στο επίκεντρο την αποτύπωση εμπειρικής γνώσης αναφορικά με θέματα σχεδίασης. Από τον ορισμό τους, για να χαρακτηριστεί μια πρακτική σχεδίασης ως πρότυπο θα πρέπει να έχει εφαρμοστεί σε διαφορετικά περιβάλλοντα για την αντιμετώπιση του ίδιου προβλήματος. Επιπλέον τα σχεδιαστικά πρότυπα προτείνουν μια στρατηγική για την αντιμετώπιση του προβλήματος. Παράλληλα δίνουν έμφαση στον εντοπισμό του προβλήματος και στην κωδικοποίηση του με τρόπο εύκολα κατανοητό με σκοπό την εφαρμογή τους στην διαδικασία της σχεδίασης. Ενώ λοιπόν οι σχεδιαστικές αρχές παρέχουν οδηγίες για την αποφυγή κακής σχεδίασης, τα πρότυπα ορίζουν το σχεδιαστικό πρόβλημα και περιγράφουν την λύση του.

Τα πρότυπα μπορούν να δράσουν σε πολλά επίπεδα. Ενώ η περιγραφή της λύση είναι γενική έτσι ώστε να βρίσκει εφαρμογή σε διαφορετικές συνθήκες. Με αυτόν τον τρόπο έχουν την δυνατότητα να προτείνουν τεχνικές αντιμετώπισης περιορισμών που

μπορεί να προέρχονται από την γλώσσα προγραμματισμού [8]. Ένα παράδειγμα είναι οι περιορισμοί αναφορικά με την πολλαπλή κληρονομικότητα. Υπάρχουν γλώσσες που δεν υποστηρίζουν κληρονομικότητα από διαφορετικές κλάσεις. Έτσι λοιπόν η διαδικασία επέκτασης μίας δομικής μονάδας μπορεί να μην είναι πραγματοποιήσιμη. Σε αυτήν την περίπτωση το πρότυπο Bridge παρέχει μια λύση βάση της οποία η διαδικασία συντήρησης παραμένει ανεξάρτητη από τα χαρακτηριστικά της γλώσσας.

Η αντικειμενοστρεφής σχεδίαση ενισχύει την οργάνωση των μεθόδων με κατανεμημένο τρόπο, ο οποίος με την σειρά του οδηγεί στην ισχυρή σύζευξη διαφορετικών μονάδων του συστήματος. Τα σχεδιαστικά πρότυπα αντιμετωπίζουν το πρόβλημα της σύζευξης σε επίπεδο συστήματος, ενώ η λύση που προτείνουν επιδρά σε όλες τις δομικές μονάδες και βελτιώνουν το σύστημα συνολικά. Ένα παράδειγμα είναι το πρότυπο Mediator το οποίο μειώνει την σύζευξη μεταξύ τον κλάσεων του συστήματος και αναλαμβάνει τον ρόλο του συντονιστή της κατανεμημένης λειτουργίας. Ακόμα η σχεδίαση μπορεί να θεωρηθεί ως μια διαδικασία σύνθεσης, μια διαδικασία συγκέντρωσης μονάδων ή μια διαδικασία συνδυασμού μονάδων. Βάσει αυτής της προσέγγισης στα πλαίσια της αντικειμενοστρεφούς σχεδίασης η δημιουργία του συστήματος θα πραγματοποιηθεί δημιουργώντας τις κλάσεις ενώ στην συνέχεια ακολουθεί ο συνδυασμός των κλάσεων με τα πρότυπα να παρεμβαίνουν στην διαδικασία του συνδυασμού.

Συνοψίζοντας αναφορικά με την χρησιμότητα των σχεδιαστικών προτύπων γίνεται φανερό ότι αποτελούν μια ακόμα μέθοδο σχεδίασης, ενώ παράλληλα επεκτείνουν τις αρχές της αντικειμενοστραφής σχεδίασης. Τέλος η υιοθέτηση των προτύπων στην διαδικασία της σχεδίασης βοηθάει στον έλεγχο και στη συντήρηση του συστήματος ενώ ταυτόχρονα ενισχύει και την επαναχρησιμοποίηση του κώδικα.

## 2.4. Κατηγοριοποίηση των Προτύπων

### *2.4.1. Η χρησιμότητα των κατηγοριοποιήσεων*

Τα σχεδιαστικά πρότυπα αναφορικά με την αντικειμενοστρεφή σχεδίαση μπορούν να εφαρμοστούν σε διαφορετικά επίπεδα και σε διαφορετικά στάδια της διαδικασίας σχεδίασης. Παράλληλα τα σχεδιαστικά πρότυπα αντιμετωπίζουν διαφορετικά προβλήματα ενώ η συνεργασία διαφορετικών προτύπων δίνει λύση σε πιο περίπλοκα προβλήματα. Μια κατηγοριοποίηση έχει ως σκοπό την οργάνωση αυτής της γνώσης σε ομάδες. Έτσι λοιπόν σχεδιαστικά πρότυπα που αντιμετωπίζουν παρόμοια προβλήματα ή μπορούν να συνεργαστούν για την αντιμετώπιση κάποιου προβλήματος μπορούν να ενταχθούν σε μια κατηγόρια.

Στόχος των κατηγοριοποιήσεων είναι η παροχή ενός σχετικά εύκολου τρόπου εύρεσης του κατάλληλου προτύπου προς αντιμετώπιση ενός συγκεκριμένου προβλήματος. Πιο συγκεκριμένα όσο πιο μεγάλο είναι το πλήθος των προτύπων, τόσο πιο δύσκολη η κατανόηση και η χρήση τους. Το βασικό πρόβλημα λοιπόν είναι η εύρεση ενός τρόπου με τον οποίο οι σχεδιαστές λογισμικού μπορούν να διαχειριστούν την γνώση που απορρέει από τα πρότυπα. Όπως γίνεται αντιληπτό, είναι δύσκολο για έναν σχεδιαστή λογισμικού, κάθε φορά που προκύπτει κάποιο σχεδιαστικό πρόβλημα να ανατρέχει στα πρότυπα, να τα μελετάει ως ανεξάρτητες μονάδες, να τα κατανοεί σε βάθος και ύστερα να επιλέγει το κατάλληλο για την λύση του προβλήματος του. Επιπλέον ο συνεχώς αυξανόμενος αριθμός των προτύπων που προκύπτει από την ανάπτυξη νέων τεχνολογιών κάνει πιο επιτακτική την ανάγκη ανάπτυξη μεθοδολογιών και τεχνικών που να βοηθούν την οργάνωση των προτύπων.

Οι τεχνικές κατηγοριοποίησης των σχεδιαστικών προτύπων, αποτελούν έναν τρόπο οργάνωσης των προτύπων σε σύνολα. Τα σύνολα αυτά έχουν ως χαρακτηριστικό τον διαμοιρασμό κοινών ιδιοτήτων. Ενώ το είδος των ιδιοτήτων που χαρακτηρίζουν τα σύνολα δεν είναι κάτι δεδομένο, ενώ παράλληλα μπορούν να περιλαμβάνουν κριτήρια όπως την δομή του προτύπου των σκοπό ή το πεδίο εφαρμογής του. Συνεπώς το είδος της κατηγοριοποίησης των προτύπων είναι συνάρτηση των κριτηρίων με τα οποία θα υλοποιηθεί η διαδικασία.

*2.4.2. Τα χαρακτηριστικά των κατηγοριοποιήσεων*

Τα διάφορα σχήματα κατηγοριοποίησης έχουν ως χαρακτηριστικό διαφορετικές διαστάσεις. Για παράδειγμα ένα σχήμα δυο διαστάσεων χρησιμοποιεί δυο κριτήρια για την υλοποίηση της κατηγοριοποίησης των προτύπων. Συνήθως όσο περισσότερες διαστάσεις έχει ένα σχήμα τόσο πιο λεπτομερείς κατηγοριοποίηση παράγει. Ακόμα κάθε σχήμα κατηγοριοποίησης έχει ως σκοπό την ανάδειξη των κύριων χαρακτηριστικών του κάθε σχεδιαστικού προτύπου. Παράλληλα ένα σχήμα κατηγοριοποίησης μπορεί μέσω των διαστάσεων του να αναδείξει τις συσχετίσεις μεταξύ των διαφορετικών προτύπων.

Σημαντικό είναι και ο εντοπισμός των ιδιοτήτων που πρέπει να έχει ένα σχήμα κατηγοριοποίησης, προκειμένου να είναι ποιοτικό και να βοηθάει τους σχεδιαστές στην διαδικασία εύρεσης του κατάλληλου για κάποιο πρόβλημα. Σύμφωνα με μια προσέγγιση ένα σχήμα κατηγοριοποίησης πρέπει να χαρακτηρίζεται από ένα σύνολο πέντε ιδιοτήτων [5]. Δηλαδή πρέπει να είναι απλό και εύκολο στην κατανόηση, να αποτελείται από σχετικά λίγα κριτήρια, να αναδεικνύει τις ιδιότητες των προτύπων, να οδηγεί στην σωστή επιλογή και να είναι ανοιχτό στην εισαγωγή νέων προτύπων.

Πιο συγκεκριμένα αναφορικά με την απλότητα και την ευκολία στην κατανόηση ένα σχήμα ομαδοποίησης, το κάθε σχήμα πρέπει να είναι εύκολο τόσο στην διαδικασία της κατηγοριοποίησης όσο και στην διαδικασία της επιλογής του κατάλληλου. Ακόμα προκειμένου να μην υπάρχει μεγάλη πολυπλοκότητα, το κάθε σχήμα πρέπει να αποτελείται από σχετικά λίγα κριτήρια, με αυτόν τον τρόπο υιοθετεί ένα συμπαγές σχήμα διαστάσεων που στόχο έχει τον περιορισμό σε λίγα χαρακτηριστικά και όχι στις δευτερεύουσες πτυχές του κάθε προτύπου. Επιπλέον και η ανάδειξη των βασικών ιδιοτήτων ως χαρακτηριστικό δίνει έμφαση στα κυρία χαρακτηριστικά του κάθε προτύπου όπως για παράδειγμα το ακριβές πρόβλημα του αντιμετωπίζει το πρότυπο. Παράλληλα η καθοδήγηση στην σωστή επιλογή έχει να κάνει με την με το πόσο εύκολη είναι η διαδικασία επιλογής του σωστού προτύπου για ένα δεδομένο πρόβλημα. Τέλος ο ανοιχτός χαρακτήρας στην εισαγωγή νέων προτύπων, έχει να κάνει με την ευκολία εισαγωγής νέων προτύπων στο ήδη υπάρχων σχήμα χωρίς να απαιτούνται διαδικασίες επαναπροσδιορισμού του σχήματος.

# ΚΕΦΑΛΑΙΟ 3. ΣΧΕΤΙΚΗ ΔΟΥΛΕΙΑ



## 3.1. Η κατηγοριοποίηση GoF

Όπως προαναφέρθηκε η GoF [11] πέρα από την συστηματική καταγραφή ενός συνόλου προτύπων, πρότειναν και μια κατηγοριοποίηση με σκοπό την ευκολότερη εκμάθηση τους. Έτσι λοιπόν μέσω της κατηγοριοποίησης των προτύπων, ορίζονται ομάδες που περιέχουν σχετικά πρότυπα.

Όπως δείχνει και ο πίνακας 3.1 η GoF πρότειναν μια κατηγοριοποίηση δυο διαστάσεων. Το πρώτο κριτήριο είναι ο σκοπός (purpose) και περιγραφεί το βασικό πρόβλημα που αντιμετωπίζει το κάθε πρότυπο. Με την χρήση αυτού του κριτηρίου, τα πρότυπα μπορούν να κατηγοριοποιηθούν σε τρείς ομάδες, πρότυπα δημιουργίας (Creational Patterns), πρότυπα δόμησης (Structural Patterns) και πρότυπα συμπεριφοράς (Behavioral Pattern). Το δεύτερο κριτήριο είναι το πλαίσιο (scope) εφαρμογής, το οποίο κατηγοριοποιεί τα πρότυπα ανάλογα με το αν εφαρμόζονται σε κλάσεις ή σε αντικείμενα.

Πίνακας 3.1 η Κατηγοριοποίηση GoF [11]

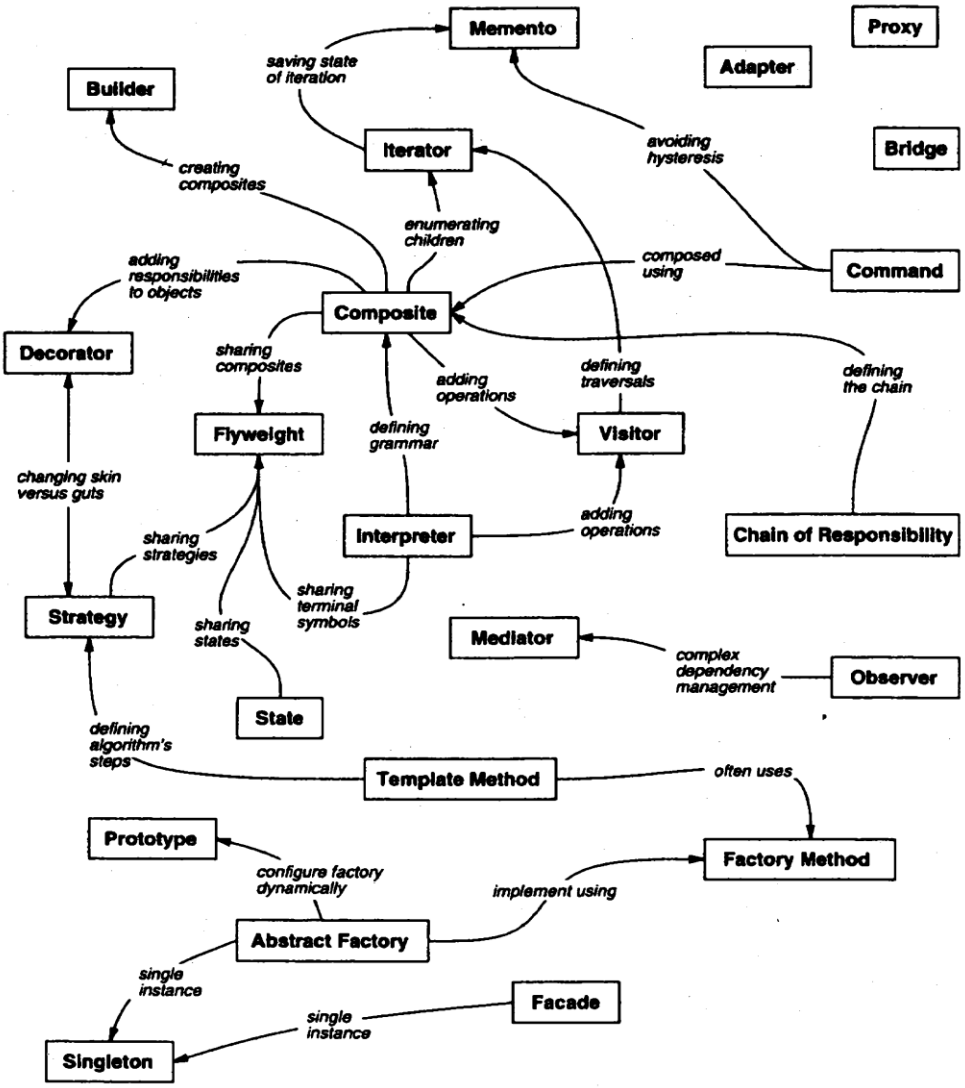
| | | **Purpose** | | |
|---|---|---|---|---|
| | | **Creational** | **Structural** | **Behavioral** |
| **Scope** | **Class** | Factory Method | Adapter (class) | Interpreter<br>Template method |
| | **Object** | Abstract Factory<br>Builder<br>Prototype<br>Singleton | Adapter (object)<br>Bridge<br>Composite<br>Decorator<br>Façade<br>Flyweight<br>Proxy | Chain of Responsibility<br>Command<br>Iterator<br>Mediator<br>Memento<br>Observer<br>State<br>Strategy<br>Visitor |

Αναφορικά με το πλαίσιο εφαρμογής, τα πρότυπα κλάσεων εξετάζουν θέματα που χαρακτηρίζουν τις σχέσεις των κλάσεων με τις υποκλάσεις τους. Ο καθορισμός των σχέσεων αυτών γίνεται μέσω της κληρονομικότητας, όμως η υλοποίηση των μεθόδων των υποκλάσεων οδηγεί στην υπερφόρτωση των μεθόδων των κλάσεων. Συνεπώς η διαδικασία της μεταγλώττισης πέρα από τις υποκλάσεις επηρεάζει και τις κλάσεις του προτύπου. Αντίθετα, τα πρότυπα αντικειμένων εξετάζουν θέματα αναφορικά με τα αντικείμενα των κλάσεων, έτσι λοιπόν οι τροποποιήσεις μπορούν να γίνουν σε πραγματικό χρόνο και η διαδικασία της μεταγλώττισης επηρεάζει μόνο τις κλάσεις που τροποποιήθηκαν.

Τα πρότυπα δημιουργίας αντιμετωπίζουν το πρόβλημα της δημιουργίας ενός αντικειμένου, ενώ ως σκοπό έχουν την ανεξαρτητοποίηση του συστήματος από την διαδικασία δημιουργίας ενός αντικειμένου. Αυτό σημαίνει ότι με την χρήση του προτύπου, το σύστημα δεν έχει γνώση διαδικασιών δημιουργίας, σύνθεσης και αναπαράστασης ενός αντικειμένου. Τα πρότυπα δόμησης, αντιμετωπίζουν το πρόβλημα κατασκευής σύνθετων κλάσεων και αντικειμένων. Παράλληλα τα πρότυπα αυτά προσφέρουν λύση για την ενοποίηση ανεξάρτητων κλάσεων ή ιεραρχιών με

στόχο την συνεργασία τους στα πλαίσια ενός συστήματος. Τέλος τα πρότυπα συμπεριφοράς, αντιμετωπίζουν προβλήματα κατανομής αρμοδιοτήτων ανάμεσα σε αντικείμενα και κλάσεις, ενώ παράλληλα εξετάζουν και τους τρόπους επικοινωνίας των αντικειμένων και θέματα υλοποίησης αλγορίθμων.

Πέρα όμως από την κατηγοριοποίηση με κριτήρια τον σκοπό και το πλαίσιο εφαρμογής, η GoF πρότεινε κι ένα μοντέλο συσχέτισης των προτύπων. Σε αντίθεση λοιπόν με την οργάνωση των προτύπων σε κατηγορίες που σκοπό έχουν την ανάδειξη των βασικών χαρακτηριστικών των προτύπων, το μοντέλο συσχέτισης των προτύπων αναδεικνύει σημαντικές ομοιότητες, διαφορές καθώς και συνεργασίες προτύπων που μπορούν να πετύχουν κάποιον στόχο.




Σχήμα 3.1 το Μοντέλο Συσχετίσεων των Προτύπων [11].

Το σχήμα 3.1 δείχνει τις συσχετίσεις που υπάρχουνε μεταξύ των διαφορετικών προτύπων. Ο συγκεκριμένος τρόπος οργάνωσης των προτύπων, προκύπτει από τις αναφορές που έχει το κάθε πρότυπο σε οποιοδήποτε άλλο. Ενώ οι αναφορές αυτές είναι καταγεγραμμένες στην ενότητα "Related Patterns" του κάθε προτύπου.

### 3.2. Η κατηγοριοποίηση Walter Zimmer

Ο Zimmer [23] πρότεινε μια νέα κατηγοριοποίηση για τα πρότυπα που παρουσίασε η GoF. Στόχος του είναι να μην μπει σε λεπτομερή καταγραφή των συσχετίσεων που διέπουν τις σχέσεις μεταξύ των προτύπων, αλλά να δώσει μια γενική προσέγγιση των σχέσεων αυτών. Με τον τρόπο αυτόν ο Zimmer δημιουργεί μια κατηγοριοποίηση χρησιμοποιώντας τις συσχετίσεις των προτύπων.

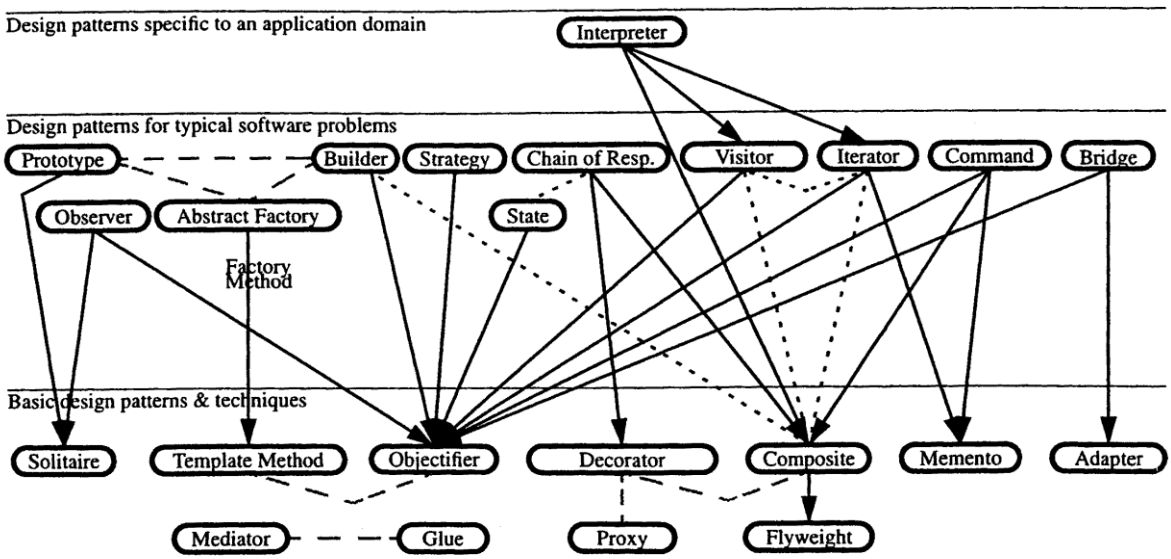
Ως έναυσμα για την συγκεκριμένη δουλειά υπήρξε ο ορισμός των συσχετίσεων μεταξύ των προτύπων όπως της παρουσίασε η GoF και φαίνονται στο σχήμα 3.1. Ζητούμενο είναι ένας τρόπος με τον οποίο θα μπορούσε να γίνει ομαδοποίηση αυτών των σχέσεων. Τελικά προκύπτει μια κατηγοριοποίηση δυο διαστάσεων, η μια διάσταση δημιουργεί κατηγορίες ανάλογα με τις συσχετίσεις που υπάρχουν, ενώ η άλλη ανάλογα με το επίπεδο που ανήκει το κάθε πρότυπο.

Αναφορικά με τις συσχετίσεις ο Zimmer διέκρινε τρείς κατηγορίες. Η σχέση X χρησιμοποιεί Y στην προτεινόμενη λύση του (X uses Y in its solution), χρησιμοποιείται στην περίπτωση που ένα πρότυπο X που λύνει ένα συγκεκριμένο πρόβλημα, εφαρμόζει ένα άλλο πρότυπο Y για την λύση ενός υποπροβλήματος. Συνεπώς η λύση που προτείνει το πρότυπο Y αποτελεί ένα τμήμα της λύση που παρέχει το πρότυπο X. Επιπλέον ο Zimmer προτείνει διαχωρισμό της σχέσης αυτής σε δυο κατηγορίες δηλαδή, X πρέπει να χρησιμοποιήσει Y (X must use Y) και X μπορεί να χρησιμοποιήσει Y (X might use Y). Ακόμα εντοπίζει την σχέση X είναι όμοιο του Y ( X is similar to Y) σύμφωνα με την οποία τα δυο πρότυπα αντιμετωπίζουν το ίδιο πρόβλημα. Πιο συγκεκριμένα η σχέση αυτή υπάρχει μεταξύ προτύπων που σκοπό έχουν την αντιμετώπιση του ιδίου προβλήματος, ενώ η λύση που προτείνουν μπορεί να διαφέρει. Τέλος η σχέση X μπορεί να συνδυαστεί με Y (X can be combined with Y) συμφώνα με την οποία τα δυο πρότυπα μπορούν να

συνδυαστούν προκειμένου να πετύχουν κάποιο αποτέλεσμα. Η σχέση συνεργασίας (can be combined) διαφέρει από την σχέση χρήσης (uses). Η πιο σημαντική διαφορά είναι ότι η σχέση συνεργασίας δεν υπονοεί ότι το ένα πρότυπο εμπεριέχει το άλλο ως μέρος της λύσης του αλλά ότι τα δυο πρότυπα εφαρμόζονται ανεξάρτητα ενώ ο συνδυασμός τους παράγει ένα διαφορετικό αποτέλεσμα.

Πέρα από των ορισμό των κατηγοριοποιήσεων ο Zimmer προτείνει και επιπλέον αλλαγές. Μια από αυτές είναι ο ορισμός ενός νέου προτύπου, το πρότυπο Objectifier στο σχήμα των συσχετίσεων. Πιο συγκεκριμένα, υπάρχουν πρότυπα που ως λύση προτείνουν την υλοποίηση λειτουργιών μέσω της συνάθροισης αντικειμένων μια ιεραρχίας. Η διαδικασία αυτή μπορεί να θεωρηθεί ως ένα ξεχωριστό πρότυπο το οποίο χρησιμοποιείται για την λύση ενός υποπροβλήματος ενός άλλου προτύπου. Τέλος ορίζει τρία επίπεδά σημασιολογικού διαχωρισμού των προτύπων ανάλογα με την πεδίο χρήσης τους. Το επίπεδο «βασικά σχεδιαστικά πρότυπα και τεχνικές» ( Basic design patterns and techniques ), περιλαμβάνει πρότυπα γενικής χρήσης τα οποία έχουν εφαρμογή σε ένα μεγάλο εύρος προβλημάτων αντικειμενοστρεφούς σχεδίασης. Το επίπεδο «σχεδιαστικά πρότυπα για τυπικά προβλήματα λογισμικού» (Design patterns for typical software problems), περιλαμβάνει τα πρότυπα εκείνα που παρέχουν μια λύση σε πιο συγκεκριμένα προβλήματα ενώ ταυτόχρονα έχουν και καθορισμένο πεδίο εφαρμογής τους το οποίο όμως περιλαμβάνει ένα μεγάλο εύρος προγραμμάτων. Τέλος το επίπεδο «σχεδιαστικά πρότυπα συγκεκριμένου πεδίου εφαρμογής» (Design patterns specific to application domain), περιλαμβάνει τα πρότυπα που έχουνε πολύ συγκεκριμένο πεδίο εφαρμογής.

Το σχήμα 3.2 δείχνει πώς διαμορφώνεται η κατηγοριοποίηση των σχέσεων που παρουσίασε η GoF βάσει της προσέγγισης του Zimmer. Όπως δείχνει το σχήμα οι σχέσεις ορίζονται με διαφορετικό βέλος, ενώ παράλληλα υπάρχουν και μερικές τροποποιήσεις. Στη κατηγοριοποίηση αυτή το δυο πρότυπα παρουσιάζονται με διαφορετικό όνομα αναφορικά με την κατηγοριοποίηση της GoF, το πρότυπο Singleton ονομάζεται Solitaire και το πρότυπο Façade ονομάζεται Glue.

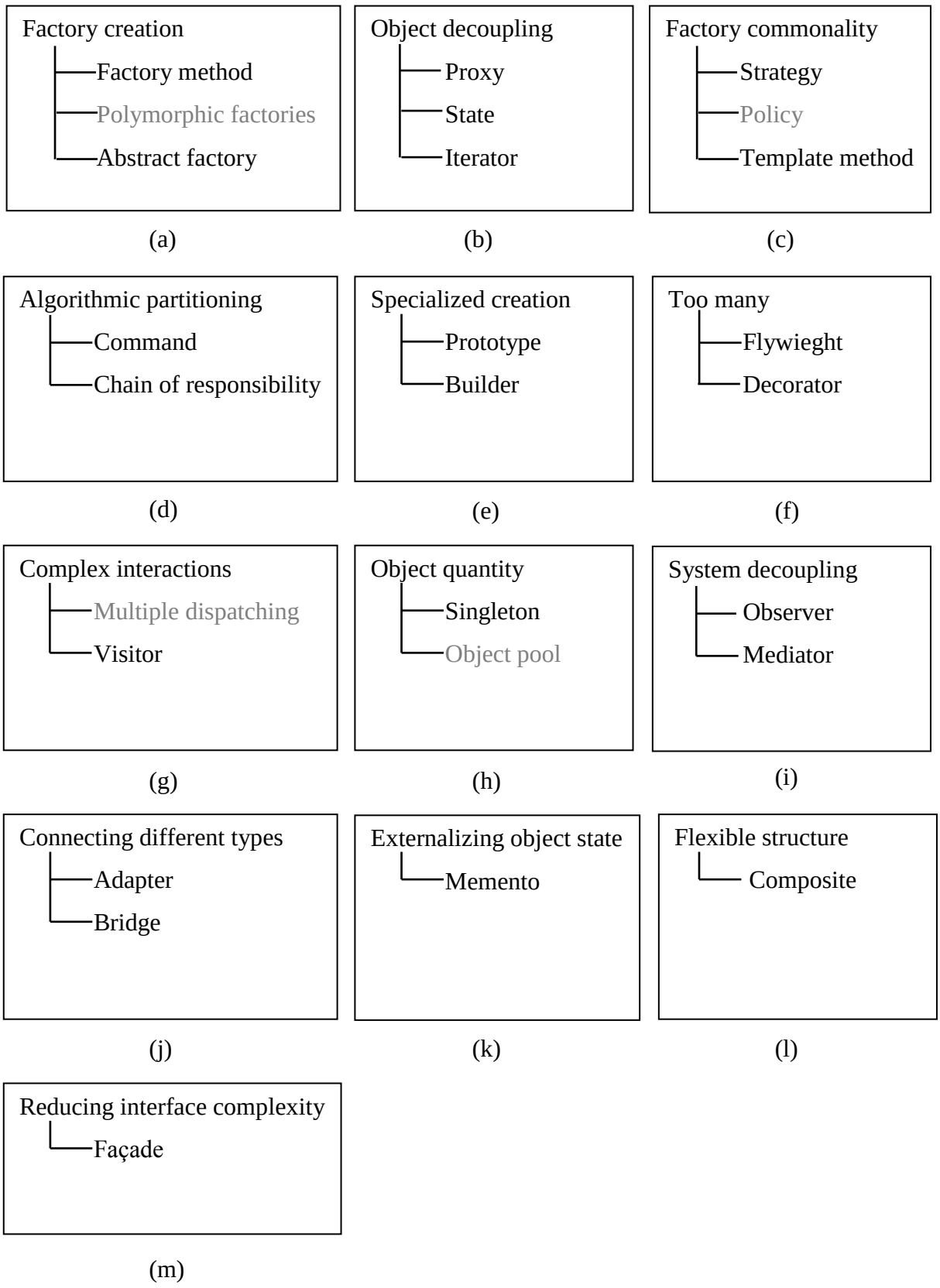



Σχήμα 3.2 το Μοντέλο Συσχετίσεων των Προτύπων [23].

Βασική τροποποίηση είναι η εισαγωγή του προτύπου Objectifier στο επίπεδο πρότυπα και τεχνικές. Επιπλέον υπάρχει σχέση χρήσης μεταξύ των προτύπων Observer, Builder, Strategy, State, Visitor, Iterator, Command, Bridge και του Objectifier. Ακόμα υπάρχει σχέση ομοιότητας μεταξύ των προτύπων Template Method και Objectifier λόγω του ότι και τα δυο έχουν ως σκοπό τον πολυμορφισμό. Μια ακόμα αλλαγή είναι η διαγραφή το Factory Method και αυτό γιατί όπως υποστηρίζει ο Zimmer το Factory Method δεν αποτελεί ξεχωριστό πρότυπο αλλά ουσιαστικά είναι μια σχέση χρήσης μεταξύ του Abstract Factory και του Template Method.

### 3.3. Η κατηγοριοποίηση Bruce Eckel

Μια ακόμα κατηγοριοποίηση των πρότυπων που παρουσιάστηκαν από την GoF προτείνει ο Eckel [10]. Κριτήριο αυτής της κατηγοριοποίησης αποτελεί το πρόβλημα στο οποίο δίνει λύση το κάθε πρότυπο. Σκοπός της συγκεκριμένης κατηγοριοποίησης είναι η παροχή βοήθειας στους σχεδιαστές συστημάτων. Έτσι αφού εντοπίσουν κάποιο πρόβλημα να μπορούν εύκολα να φτάνουν στην λύση του προβλήματος, ψάχνοντας τα πρότυπα μιας μόνο κατηγορίας. Το σχήμα 3.3 δίνει μια άποψη των κατηγόριων που προτείνει ο Eckel καθώς και τα πρότυπα που υπάρχουν σε κάθε μια από αυτές.




Σχήμα 3.3 η Κατηγοριοποίηση του Eckel [10].

Η κατηγορία Object quantity περιλαμβάνει πρότυπα που ελέγχουν την ποσότητα των αντικειμένων που δημιουργούνται. Στην κατηγορία Object Decoupling υπάρχουν πρότυπα τα οποία παρέχουν μια κλάση υποκατάστατο της πραγματικής μέσω της οποίας γίνεται η επικοινωνία της πραγματικής κλάσης με τον Client του συστήματος. Η κατηγορία Factoring commonality περιέχει πρότυπα τα οποία υλοποιούν τα βήματα εκτέλεσης μιας διαδικασίας σε διαφορετικές κλάσεις από αυτήν που τα χρησιμοποιεί. Πρότυπα που η διαδικασία δημιουργίας ενός αντικειμένου παρέχεται από ένα στοιχείο αφαίρεσης, συγκροτούν την κατηγορία Encapsulating creation, ενώ Specialized creation είναι τα πρότυπα που χρησιμοποιούν ειδικευμένους τρόπος δημιουργίας αντικειμένων. Ακόμα η κατηγορία Too many περιέχει πρότυπα που διαχειρίζονται περιπτώσεις πολλών αντικειμένων η κλάσεων σε ένα σύστημα. Η κατηγορία Connecting different types περιέχει πρότυπα που ενοποιούν διαφορετικές κλάσεις ή λειτουργίες σε μια ενιαία δομή. Η κατηγορία Flexible structure επικεντρώνεται στον τρόπο δόμησης σύνθετων αντικειμένων, ενώ η κατηγορία System decoupling περιέχει πρότυπα που υποστηρίζουν μειωμένη σύζευξη μεταξύ των αντικειμένων. Ακόμα η κατηγορία Reducing interface complexity επικεντρώνεται στην δημιουργία interface μέσω του οποίου μια κλάση επικοινωνεί με μια άλλη η με ένα υποσύστημα. Ακόμα η κατηγορία Algorithmic partitioning περιέχει πρότυπα στα οποία η διαδικασία επιλογής ενός αντικειμένου υπάρχει κατανεμημένη σε διάφορες κλάσεις. Η κατηγορία Externalizing object state ασχολείται με την διαχείριση των καταστάσεων μιας κλάσης, εξωτερικά από την ίδια την κλάση. Τέλος η κατηγορία Complex interactions περιέχει πρότυπα που εξετάζουν τον τρόπο επικοινωνίας των κλάσεων ενός συστήματος.

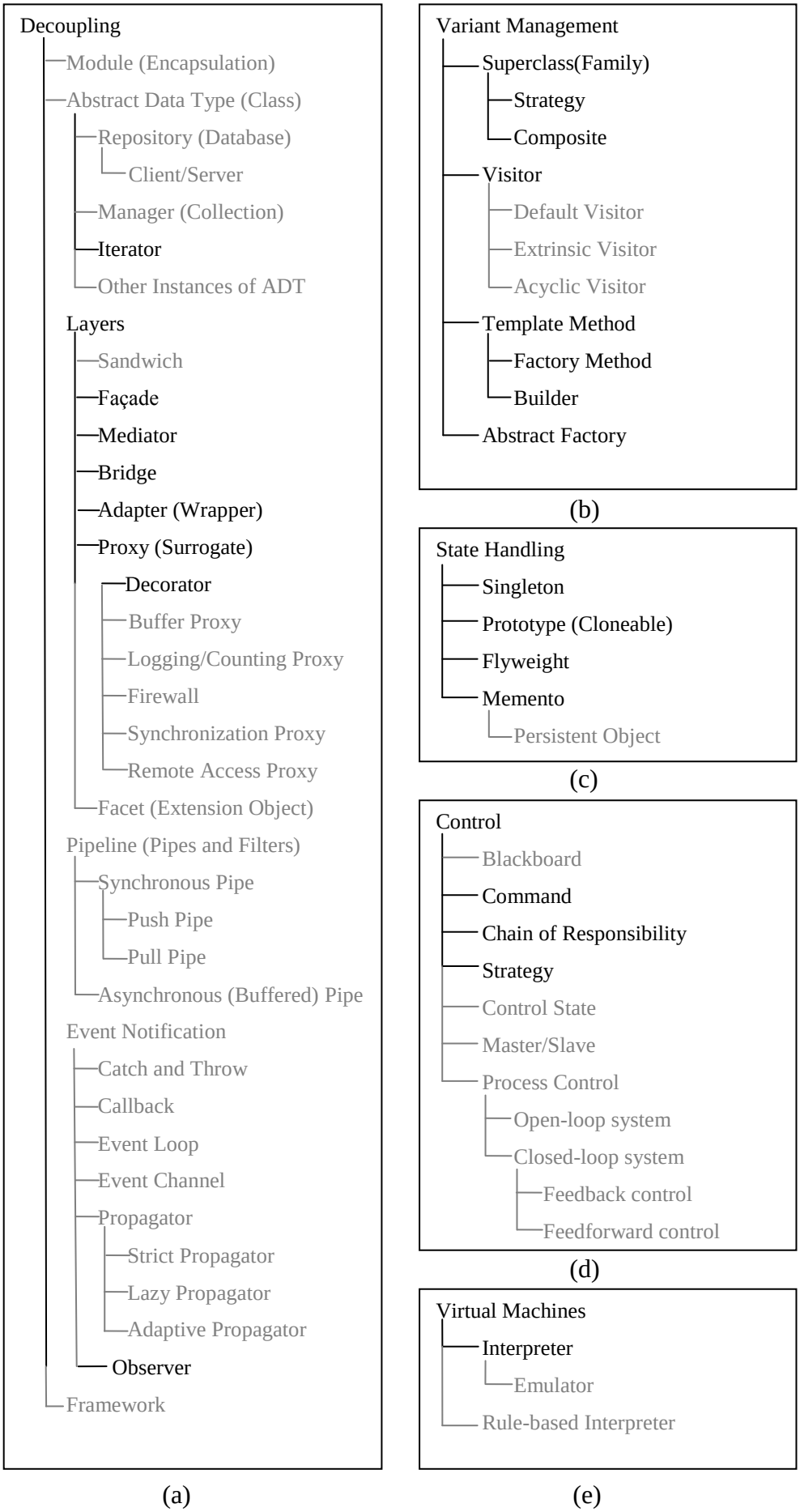
### 3.4. Η κατηγοριοποίηση Walter Tichy

Ο Tichy προτείνει μια κατηγοριοποίηση η οποία περιλαμβάνει περισσότερα από εκατό πρότυπα γενικής χρήσης [22]. Το σύνολο αυτό περιλαμβάνει και τα πρότυπα που παρουσιάζει η GoF, ενώ το κριτήριο σύμφωνα με το οποίο έγινε η κατηγοριοποίηση είναι η χρησιμότητα του προτύπου και πιο συγκεκριμένα τα προβλήματα τα οποία λύνουν. Η κατηγοριοποίηση αυτή στοχεύει στον πιο εύκολο εντοπισμό του κατάλληλου προτύπου για την αντιμετώπιση ενός δεδομένου σχεδιαστικού προβλήματος. Το συγκεκριμένο σχήμα χαρακτηρίζεται από κατηγορίες

και υποκατηγορίες. Οι υποκατηγορίες αυτές ειδικεύουν κάποια χαρακτηριστικά των προτύπων ενώ οι κατηγορίες παρουσιάζουν μια γενικότερη εικόνα του προβλήματος που αντιμετωπίζουν.

Σύμφωνα λοιπόν με το κριτήριο αξιολόγησης, ο Ticky παρουσιάζει εννέα κατηγορίες προτύπων για την αντιμετώπιση εννέα διαφορετικών προβλημάτων. Η κατηγορία Decoupling έχει ως βασικό χαρακτηριστικό την ανεξαρτησία των δομικών μονάδων ενός συστήματος. Με αυτόν τον τρόπο διευκολύνεται η διαδικασία επέκτασης, συντήρησης και επαναχρησιμοποίησης ενός λογισμικού. Η κατηγορία Variant Management περιλαμβάνει πρότυπα με βασικό χαρακτηριστικό τον χειρισμό διαφορετικών αντικειμένων, που παρουσιάζουν όμως κοινά χαρακτηριστικά, με όμοιες διαδικασίες. Στην κατηγορία State Handling, ανήκουν πρότυπα που αντιμετωπίζουν θέματα χειρισμού καταστάσεων ενός αντικειμένου. Το βασικό χαρακτηριστικό που διακρίνει την κατηγορία Control είναι το πρόβλημα της υλοποίησης διαδικασιών ελέγχου στα πλαίσια ενός συστήματος. Τα πρότυπα αυτής της κατηγορίας στοχεύουν στον αποδοτικότερο έλεγχο των εκτελέσεων και της επιλογής των κατάλληλων μεθόδων για την εκτέλεση μιας λειτουργίας. Στην κατηγορία Virtual Machines υπάρχουν πρότυπα που προσομοιώνουν έναν επεξεργαστή. Πιο συγκεκριμένα, πρότυπα που ανήκουν σε αυτήν την κατηγορία παρέχουν τρόπους διερμήνευσης προγραμμάτων που είναι γραμμένα σε άλλες γλώσσες.

Πέρα από τις κατηγορίες που προαναφέρθηκαν υπάρχουν και μερικές άλλες, οι οποίες όμως δεν περιλαμβάνουν κανένα από τα σχεδιαστικά πρότυπα της GoF. Η κατηγόρια Convenience Patterns στοχεύει στην απλοποίηση τον κώδικα μέσω της συγκέντρωσης επαναλαμβανόμενων κομματιών σε ένα σημείο. Η κατηγορία Compound Patterns περιλαμβάνει πρότυπα που συνθέτονται από άλλα απλούστερα. Πρότυπα που ελέγχουν παράλληλη και ταυτόχρονη εκτέλεση περιλαμβάνει η κατηγορία Concurrency. Τέλος η κατηγορία Distribution χαρακτηρίζεται από την αντιμετώπιση προβλημάτων που εμφανίζονται σε κατανεμημένα συστήματα. Το σχήμα 3.4 παρουσιάζει την κατηγοριοποίηση που προτείνει ο Tichy δίνοντας περισσότερη έμφαση στα πρότυπα που καταγράφηκαν από την GoF.


Decoupling
Module (Encapsulation)
Abstract Data Type (Class)
Repository (Database)
Client/Server
Manager (Collection)
Iterator
Other Instances of ADT
Layers
Sandwich
Façade
Mediator
Bridge
Adapter (Wrapper)
Proxy (Surrogate)
Decorator
Buffer Proxy
Logging/Counting Proxy
Firewall
Synchronization Proxy
Remote Access Proxy
Facet (Extension Object)
Pipeline (Pipes and Filters)
Synchronous Pipe
Push Pipe
Pull Pipe
Asynchronous (Buffered) Pipe
Event Notification
Catch and Throw
Callback
Event Loop
Event Channel
Propagator
Strict Propagator
Lazy Propagator
Adaptive Propagator
Observer
Framework
(a)
Variant Management
Superclass(Family)
Strategy
Composite
Visitor
Default Visitor
Extrinsic Visitor
Acyclic Visitor
Template Method
Factory Method
Builder
Abstract Factory
(b)
State Handling
Singleton
Prototype (Cloneable)
Flyweight
Memento
Persistent Object
(c)
Control
Blackboard
Command
Chain of Responsibility
Strategy
Control State
Master/Slave
Process Control
Open-loop system
Closed-loop system
Feedback control
Feedforward control
(d)
Virtual Machines
Interpreter
Emulator
Rule-based Interpreter
(e)


Σχήμα 3.4 η Κατηγοριοποίηση του Tichy[22].

### 3.5. Κριτική ανασκόπηση των κατηγοριοποιήσεων

Στο κεφάλαιο αυτό έγινε παρουσίαση των σημαντικότερων κατηγοριοποιήσεων που έχουν προταθεί για την οργάνωση των σχεδιαστικών πρότυπων που καταγράφηκαν από την GoF ή και από άλλους. Οι κατηγοριοποιήσεις των προτύπων μπορεί να διαφέρουν ανάλογα με τα κριτήρια ενώ τα κριτήρια ορίζονται βάση του σκοπού που εξυπηρετεί η κάθε κατηγοριοποίηση. Έτσι υπάρχουν κατηγοριοποιήσεις βάσει του

προβλήματος που αντιμετωπίζουν, αναδεικνύοντας έτσι την χρησιμότητα, κατηγοριοποιήσεις ανάλογα με την πρόθεση του προτύπου ή κατηγοριοποιήσεις ανάλογα με τις σχέσεις που υπάρχουν ανάμεσα στα πρότυπα.

Η δισδιάστατη κατηγοριοποίηση της GoF απαντάει στο ερώτημα του τι κάνει ένα πρότυπο. Συνεπώς η χρήση μιας τέτοιας κατηγοριοποίησης μπορεί να βοηθήσει τον αναγνώστη στην γρηγορότερη κατανόηση των προτύπων. Όμως, είναι λιγότερο χρήσιμη σε περιπτώσεις αναζήτησης κάποιου προτύπου που λύνουν ένα συγκεκριμένο πρόβλημα. Το δεύτερο κριτήριο που ορίζουν, το πλαίσιο εφαρμογής δηλαδή, δεν προσφέρει κανένα στοιχείο στους σχεδιαστές που ψάχνουν κάποιο πρότυπο. Αυτό συμβαίνει γιατί το συγκεκριμένο κριτήριο δεν παρέχει κάποια πληροφορία για σχεδιαστικές δραστηριότητες. Ακόμα υπάρχει δυσκολία στον διαχωρισμό των κατηγοριών Behavioral και Structural, πιο συγκεκριμένα το πρότυπο Composite κατατάσσεται στα Structural ενώ το Interpreter στα Behavioral.

Η κατηγοριοποίηση του Zimmer προσφέρει χρήσιμες πληροφορίες για τις συσχετίσεις μεταξύ των προτύπων. Πιο συγκεκριμένα βοηθάει στην καλύτερη κατανόηση των λύσεων που προτείνουν δίνοντας έμφαση στις ομοιότητες και στις συνεργασίες μεταξύ των προτύπων. Παράλληλα ο ορισμός των διαφορετικών επιπέδων ενισχύσει την διαδικασίας της μελέτης. Έτσι λοιπόν ένας σχεδιαστής που θέλει να ασχοληθεί με τα πρότυπα μπορεί να ξεκινήσει από τα πιο απλά, που βρίσκονται στο επίπεδο «βασικά σχεδιαστικά πρότυπα και τεχνικές», ενώ στην συνέχεια έχοντας κατανοήσει τις απλές τεχνικές, μπορεί να προχωρήσει σε πιο σύνθετα πρότυπα. Το βασικό πρόβλημα που αντιμετωπίζει αυτή η κατηγοριοποίηση είναι ότι δεν προσφέρει καμία πληροφορία για το πρόβλημα που αντιμετωπίζει το κάθε πρότυπο. Έτσι λοιπόν είναι δύσκολη η εύρεση του κατάλληλου προτύπου για την αντιμετώπιση ενός δεδομένου προβλήματος. Συνοψίζοντας, μπορούμε να πούμε ότι η συγκεκριμένη κατηγοριοποίηση βοηθάει στην βαθύτερη κατανόηση των προτύπων αλλά δεν παρέχει πληροφορίες για το πρόβλημα που αντιμετωπίζουν.

Οι κατηγοριοποιήσεις των Eckel και Tichy έχουν ως κριτήριο το πρόβλημα που επιλύει το κάθε πρότυπο. Συνεπώς, παρέχουν έναν ευκολότερο τρόπο για την εύρεση του κατάλληλου προτύπου προς αντιμετώπιση ενός προβλήματος. Το βασικό

πρόβλημα που αντιμετωπίζουν είναι στην μεθοδολογία που ακολουθούν για τον ορισμό των κατηγόριων. Πιο συγκεκριμένα, αν και οι δυο κατηγοριοποιήσεις χρησιμοποιούν το ίδιο κριτήριο, παρουσιάζουν πολλές διαφορές στο τελικό αποτέλεσμα. Οι διαφορές αυτές εντοπίζονται τόσο στον διαφορετικό αριθμό των κατηγοριών όσο και στα διαφορετικά πρότυπα που υπάρχουν σε κάθε κατηγορία. Ένα συγκεκριμένο παράδειγμα είναι ο τρόπος που ερμηνεύουν την χρησιμότητα του προτύπου Singleton. Ο Tichy θεώρει ότι το συγκεκριμένο πρότυπο παρέχει μια λύση που αφορά το χειρισμό των καταστάσεων ενός αντικειμένου και γι' αυτό, κατατάσσει το πρότυπο στην κατηγορία State Handling. Από την άλλη πλευρά το ίδιο πρότυπο ο Eckel θεωρεί ότι παρέχει λύση στην διασφάλιση του πλήθους των αντικειμένων μιας κλάσης και γι' αυτό εντάσσει το πρότυπο στην κατηγορία Object quantity. Έτσι λοιπόν γίνεται φανερό ότι σε περιπτώσεις που το υπό εξέταση πρότυπο δεν καθορίζει σαφώς την κύρια ιδιότητα του, εμφανίζει δυσκολίες και στην κατηγοριοποίηση [22]. Παράλληλα σε τέτοιες περιπτώσεις η κατηγοριοποίηση επηρεάζεται και από υποκειμενικά στοιχεία που καθορίζονται από αυτόν που προτείνει την κατηγοριοποίηση.

Ζητούμενο είναι η δημιουργία μιας κατηγοριοποίησης των προτύπων χρήσιμη στην διαδικασία της αντικειμενοστεφής σχεδίασης, σαφώς καθορισμένη και χρησιμοποιώντας ποσοτικά χαρακτηριστικά για την απεικόνιση των ιδιοτήτων του κάθε προτύπου. Με αυτόν τον τρόπο θα υπάρξει μείωση της υποκειμενικότητας των κατηγοριοποιήσεων.

# ΚΕΦΑΛΑΙΟ 4. ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ ΒΑΣΙΣΜΕΝΗ ΣΤΙΣ ΜΕΤΡΙΚΕΣ



## 4.1. Μεθοδολογία αξιολόγησης των προτύπων

Βασικό στοιχείο για την κατηγοριοποίηση των προτύπων αποτελεί η αξιολόγηση τους στα πλαίσια ενός συστήματος, ενώ μέτρο για την αξιολόγηση τους αποτελούν οι μετρικές της αντικειμενοστρεφούς σχεδίασης. Έτσι λοιπόν για κάθε πρότυπο παρουσιάζεται το πρόβλημα που αντιμετωπίζει, στην συνέχεια ακολουθεί ένα παράδειγμα ως μια πρώτη προσέγγιση, στην οποία δεν υπάρχει η χρήση του προτύπου, ενώ τέλος παρουσιάζεται η χρήση του προτύπου για την αντιμετώπιση του προβλήματος πάνω στο ίδιο παράδειγμα.

Μεγάλη σημασία έχει και το παράδειγμα βάσει του οποίου θα γίνει η αξιολόγηση. Το κάθε παράδειγμα που επιλέγεται αποτελεί μια απλή περίπτωση σχεδίασης στην οποία όμως φαίνεται ξεκάθαρα η τάση που παρουσιάζουν οι μετρικές. Πιο συγκεκριμένα, ένα παράδειγμα μπορεί να θεωρηθεί ως αντιπροσωπευτικό της χρήσης του προτύπου στην περίπτωση που οι μετρικές θα παρουσιάσουν την ίδια τάση σε οποιοδήποτε σενάριο συντήρησης. Για την αξιολόγηση των προτύπων είναι απαραίτητο να λάβουμε υπόψη διαδικασίες συντήρησης ενώ σε αντίθετη περίπτωση, εξετάζοντας την πιο απλή περίπτωση για ένα πρότυπο, είναι πολύ πιθανόν να μην προκύψει κάποια βελτίωση αλλά αντίθετα το πρότυπο μπορεί να επιβαρύνει το σύστημα με επιπλέον πολυπλοκότητα.

Η αξιολόγηση πραγματοποιείται μέσω των μεταβολών που παρουσιάζουν οι μετρικές του συστήματος. Πιο συγκεκριμένα για κάθε κλάση του συστήματος υπολογίζεται η τιμή που θα έχει η κάθε μετρική πριν και μετά την χρήση του προτύπου. Η μείωση κάποιας μετρικής μετά την χρήση του προτύπου οδηγεί σε βελτίωση. Όμως η μείωση αυτή χαρακτηρίζει μια συγκεκριμένη κλάση του συστήματός ενώ η αξιολόγηση της χρήσης του προτύπου αφορά το σύστημα συνολικά και όχι κάποιες μεμονωμένες κλάσεις. Έτσι λοιπόν ένα χαρακτηριστικό μέγεθος για τον σκοπό μας είναι ο μέσος όρος της κάθε μετρικής στο σύστημα. Δηλαδή η τιμή μιας μετρικής για ολόκληρο το σύστημα ορίζεται βάσει του μέσου όρου. Όταν η χρήση του προτύπου συνοδεύεται με μείωση της τιμής αυτής το σύστημα βελτιώνεται. Σε περίπτωση όμως αύξησης το σύστημα επιβαρύνεται.

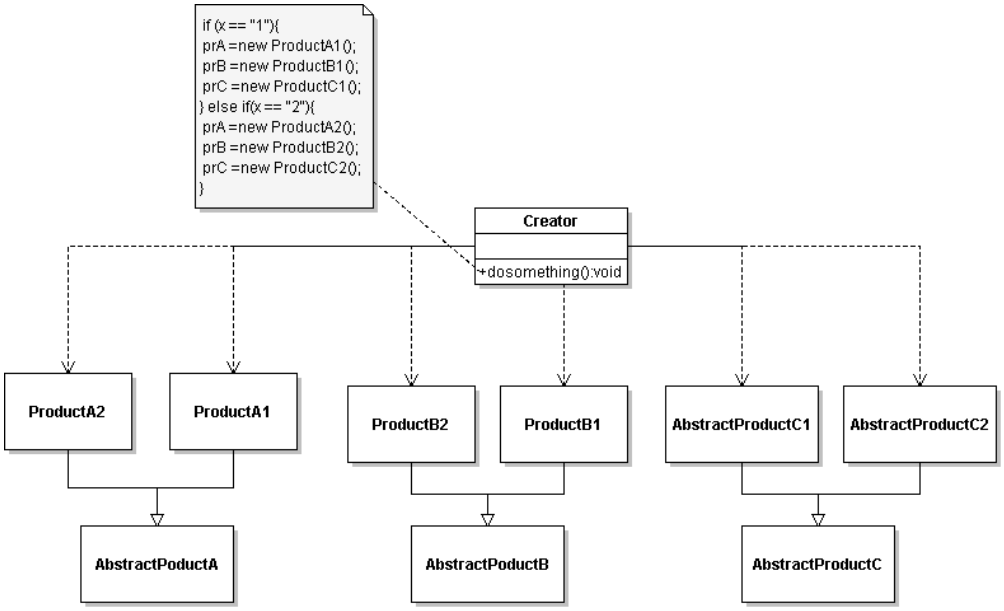
Οι μετρικές που θα χρησιμοποιηθούν για την αξιολόγηση των προτύπων είναι το σύνολο των έξι μετρικών που πρότειναν οι Chidamber και Kemerer. Οι συγκεκριμένες μετρικές προσφέρουν σημαντική πληροφόρηση στην περίπτωση που ακολουθούνται οι αρχές της αντικειμενοστρεφούς σχεδίασης, πράγμα που συμβαίνει και στα πλαίσια των σχεδιαστικών προτύπων. Τέλος, οι μετρικές αυτές γνωρίζουν μεγάλη αποδοχή από την ερευνητική κοινότητα ενώ παράλληλα αποτελούν το πιο γνωστό σύνολο που έχει προταθεί.

## 4.2. Η αξιολόγηση των προτύπων

### *4.2.1. Abstract Factory*

Το πρότυπο Abstract Factory παρέχει μια λύση σε συστήματα αποτελούμενα από ένα σύνολο ιεραρχιών και μια κλάση Creator, μια κλάση δηλαδή που ενδιαφέρεται για τη δημιουργία αντικειμένων από κλάσεις των ιεραρχιών. Το ζητούμενο ενός τέτοιου συστήματος είναι ένας αποδοτικός τρόπος για την δημιουργία αντικειμένων των διαφορετικών ιεραρχιών στον Creator. Μια πρώτη προσέγγιση είναι η απευθείας δημιουργία των αντικειμένων των ιεραρχιών στον Creator. Το σχήμα 4.1 περιγράφει την διαδικασία δημιουργίας χωρίς την χρήση του προτύπου. Βάσει αυτής της

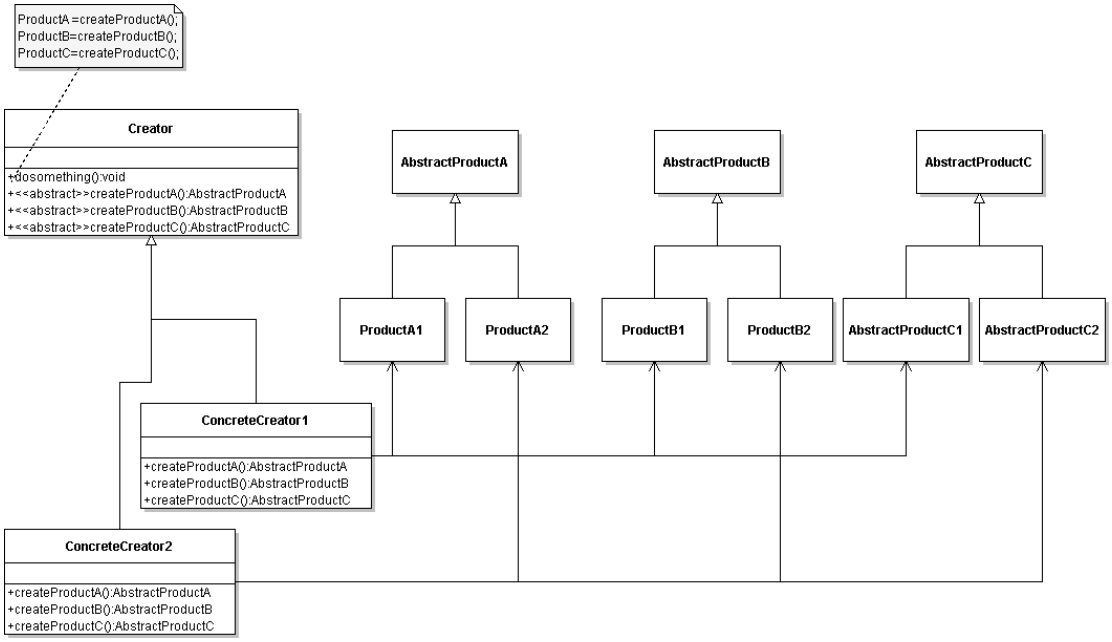
προσέγγισης, ο Creator έχει γνώση για την διαδικασία δημιουργίας αντικειμένων της κάθε υλοποίησης.




Σχήμα 4.1 Δημιουργώντας τα Αντικείμενα χωρίς το Πρότυπο

Το πρότυπο δίνει μια λύση σύμφωνα με την οποία ο Creator αποτελεί μια αφηρημένη κλάση η οποία περιλαμβάνει υλοποιήσεις για το σύνολο των μεθόδων του, όμως τα αντικείμενα της ιεραρχίας δημιουργούνται από διαφορετικές υλοποιήσεις μιας αφηρημένης μεθόδου. Στο σχήμα 4.2 φαίνεται η χρήση του προτύπου. Ο Creator έχει υλοποίηση μόνο για την μέθοδο dosomething(), ενώ η δημιουργία των αντικειμένων γίνεται βάσει των διαφορετικών υλοποιήσεων των μεθόδων createProduct(). Επιπλέον ο αριθμός των αφηρημένων μεθόδων createProduct() που υπάρχουν στον Creator είναι ίσος με τον αριθμό των ιεραρχιών που υπάρχουν στο σύστημα. Αυτό συμβαίνει γιατί ο Creator ενδιαφέρεται για την δημιουργία ενός και μόνο αντικειμένου από κάθε ιεραρχία. Ενώ παράλληλα αποκλείει και την δημιουργία άλλων αντικειμένων της κάθε ιεραρχίας.

Πιο συγκεκριμένα η κάθε υλοποίηση του Creator δημιουργεί ένα σύνολο από αντικείμενα. Το σύνολο αυτό περιλαμβάνει ένα τύπο αντικειμένου από κάθε ιεραρχία. Έτσι λοιπόν αν θεωρήσουμε ένα σύστημα με m ιεραρχίες, η σχέση που υπάρχει μεταξύ των υλοποιήσεων του Creator και των υλοποιήσεων των ιεραρχιών είναι ένα προς m.




Σχήμα 4.2 Δημιουργώντας τα Αντικείμενα Χρησιμοποιώντας Abstract Factory

Εξετάζοντας τις μετρικές παρατηρούμε ότι το CBO τείνει να βελτιώνεται όσο αυξάνει ο αριθμός των υλοποιήσεων της κάθε ιεραρχίας. Όπως φαίνεται και στα σχήματα 4.1 και 4.2 στην περίπτωση που δεν γίνει χρήση του προτύπου, η εξάρτηση του Creator είναι από όλες τις υλοποιήσεις των ιεραρχιών του συστήματος. Δηλαδή ο Creator έχει γνώση για την διαδικασία δημιουργίας όλων των αντικειμένων. Αντίθετα η χρήση του προτύπου καθιστά τον Creator ανεξάρτητο της διαδικασίας, δηλαδή δεν υπάρχουν εξαρτήσεις μεταξύ Creator και υλοποιήσεων. Όμως οι εξαρτήσεις αυτές μεταφέρονται στις υλοποιήσεις του Creator. Έτσι λοιπόν ο κάθε ConcreteCreator έχει εξάρτηση από μια υλοποίηση ανά ιεραρχία. Όπως φαίνεται και στον πίνακα 4.1 οι εξαρτήσεις ως άθροισμα παραμένουν ίδιες. Αυτό που αλλάζει είναι ο μέσος όρος του CBO ανά κλάση όπου η χρήση του προτύπου βελτιώνει το σύστημα. Πρακτικά αυτό μπορούμε να το ερμηνεύσουμε ως αποδοτικότερη κατανομή των εξαρτήσεων μεταξύ των κλάσεων του συστήματος. Ακόμα τάση βελτίωσης παρουσιάζει και το WMC. Όπως φαίνεται και στο σχήμα 4.3 για να γίνει η δημιουργία της κάθε οικογένειας συσχετιζόμενων αντικειμένων υπάρχει μια δομή επιλογής. Ανάλογα με την τιμή της παραμέτρου πραγματοποιείται και η δημιουργία διαφορετικής ομάδας αντικειμένων. Ενώ όπως φαίνεται στον σχήμα 4.4 με την χρήση του προτύπου υπάρχει μείωση της πολυπλοκότητας αφού δεν υπάρχει πλέον η δομή επιλογής αναφορικά με τις

υπόλοιπες κλάσεις όπως φαίνεται και στον πίνακα 4.1 δεν υπάρχει καμία μεταβολή της πολυπλοκότητας. Συνεπώς, λόγω της αφαίρεσης από τον Creator της δομής επιλογής, η χρήση του προτύπου οδηγεί στην συνολική βελτίωση του συστήματος σε σχέση με το WMC. Σχετικά με το LCOM όπως φαίνεται στο σχήμα 4.2 η συνεκτικότητα των κλάσεων στις ιεραρχίες παραμένει αμετάβλητη αυτό συμβαίνει γιατί δεν υπάρχει εισαγωγή νέων μεθόδων στις ιεραρχίες που θα επηρέαζε το LCOM κάθε κλάσης. Ακόμα οι νέες κλάσεις ConcreteCreator δεν παρουσιάζουν έλλειψη συνεκτικότητας αφού οι μέθοδοι που υλοποιούν δεν χρησιμοποιούν κάποιες ιδιοτιμές. Συνεπώς το LCOM παραμένει αμετάβλητο, ενώ μικρή αύξηση παρουσιάζει το RFC. Η αύξηση αυτή είναι λόγο της κλήσης των μεθόδων createProduct από την μέθοδο doSomething(). Αναφορικά με το NOC υπάρχει αύξηση, όπως φαίνεται και στο σχήμα 4.2. Ο Creator αποκτά δυο κλάσεις παιδιά, ενώ στις υπόλοιπες μονάδες του συστήματος παραμένει σταθερό. Έτσι λοιπόν όπως φαίνεται και στον πίνακα 4.1 το NOC ανά κλάση συστήματος παρουσιάζει αύξηση. Τέλος το DIT παρουσιάζει ανάλογη αύξηση και όπως φαίνεται από το σχήμα 4.2 το σύστημα απόκτα δυο νέες κλάσεις με DIT ενώ βάσει αυτού και όπως φαίνεται στον πίνακα 4.1 υπάρχει αύξηση του DIT ανά κλάση.

Βάσει λοιπόν των μετρικών η χρήση του προτύπου Abstract Factory οδηγεί στην μείωση του WMC και του CBO, ενώ παράλληλα προκαλεί μια ανάλογη αύξηση στα DIT και NOC. Για το σύστημα αυτό σημαίνει ότι υποστηρίζει σενάρια επέκτασης με κλειστό τρόπο. Δηλαδή για την εισαγωγή στο σύστημα μιας νέας οικογένειας συσχετιζόμενων κλάσεων το μόνο που πρέπει να γίνει είναι η επέκταση των ιεραρχιών έτσι ώστε οι νέες κλάσεις να εφαρμόζουν τα στοιχεία αφαίρεσης AbstractProduct. Στην περίπτωση που δεν γίνει χρήση του προτύπου και για το ίδιο σενάριο συντήρησης θα πρέπει να επέμβουμε στον Creator έτσι ώστε να υποστηρίζει την δημιουργία νέων αντικειμένων.

```
public class Client {

    String x = "2";
    AbstractProductA prA;
    AbstractProductB prB;

    public void operation1()
    {
        if (x == "1"){
            prA =new
ProductA1("ProductA1");
            prB =new
ProductB1("ProductB1");
            }
        else if(x == "2"){
            prA =new
ProductA2("ProductA2");
            prB =new
ProductB2("ProductB2");
            }
    }}
```

Σχήμα 4.3 η Κλάση Creator χωρίς Abstract Factory

```
public abstract class Creator {

    public void operation1(){
    AbstractProductA prA =createProductA();
 AbstractProductB prB =createProductB();
    }
    abstract AbstractProductA
createProductA();
    abstract AbstractProductB
createProductB();
}
```

Σχήμα 4.4 η Κλάση Creator με Χρήση Abstract Factory

```
class ConcreteCreator1 extends Creator{
public AbstractProductA createProductA(){
          return new ProductA1("ProductA1");
}
     public AbstractProductB createProductB(){
          return new ProductB1("ProductB1");
}
}
class ConcreteCreator2 extends Creator{
```

```
        AbstractProductA createProductA(){
                return new
ProductA2("ProductA2");
        }
        AbstractProductB createProductB(){
                return new
ProductB2("ProductB2");
        }
}
```

Σχήμα 4.5 οι Επεκτάσεις της Κλάσης Creator

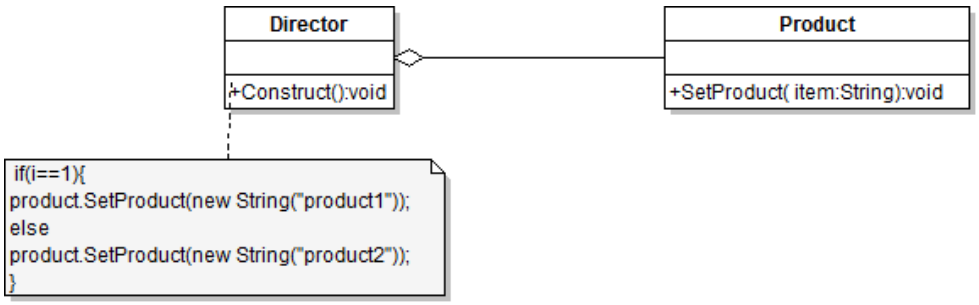
Πίνακας 4.1 οι Μεταβολές των Μετρικών με και χωρίς το Abstract Factory

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Creator | 6 | 0 | 2 | 0 | 0 | 0 | 1 | 4 | 0 | 2 | 0 | 0 |
| Concrete Creator1 | | 3 | | 0 | | 0 | | 3 | | 0 | | 1 |
| Concrete Creator2 | | 3 | | 0 | | 0 | | 3 | | 0 | | 1 |
| Abstract ProductA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| Concrete ProductA1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Concrete ProductA2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Abstract ProductB | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| Concrete ProductB1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Concrete ProductB2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Abstract ProductC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| Concrete ProductC1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Concrete ProductC2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Overall Mean | 0.6 | 0.5 | 0.2 | 0 | 0 | 0 | 0.1 | 0.9 | 0.6 | 0.66 | 0.6 | 0.66 |

*4.2.2. Builder*

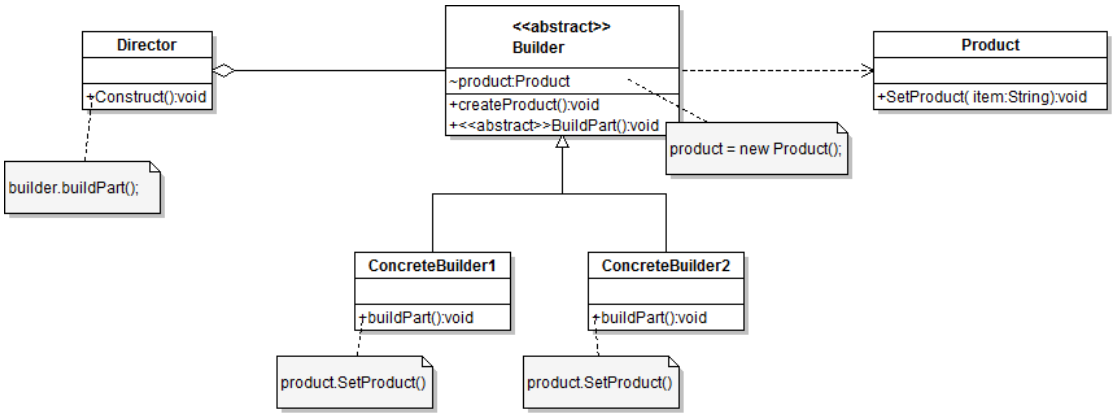
Το πρότυπο Builder δίνει μια λύση σε συστήματα που χρησιμοποιούν διαδικασίες δημιουργίας συνθετών αντικειμένων, αντικείμενα δηλαδή που αποτελούν ένα σύνολο από πιο απλά αντικείμενα. Σε τέτοια συστήματα ζητούμενο είναι ένας αποδοτικός

τρόπος δημιουργίας των σύνθετων αντικειμένων. Μια αρχική προσέγγιση στο πρόβλημα φαίνεται στο σχήμα 4.6. Σύμφωνα με αυτήν την προσέγγιση ο Director, η κλάση δηλαδή που ενδιαφέρεται για την δημιουργία του σύνθετου αντικειμένου, έχει πλήρη γνώση της διαδικασίας δημιουργίας. Αυτό σημαίνει ότι η κλάση Director θα έχει και γνώση για την εσωτερική δομή των αντικειμένων της κλάσης Product, πράγμα που στην προκειμένη περίπτωση δεν είναι απαραίτητο και αυτό γιατί ο Director ενδιαφέρεται μόνο για το σύνθετο αντικείμενο.




Σχήμα 4.6 η Δημιουργία του Σύνθετου Αντικειμένου χωρίς Builder

Το πρότυπο θεωρεί ένα στοιχείο αφαίρεσης Builder το οποίο έχει την ευθύνη της δημιουργίας του σύνθετου αντικειμένου και επεκτείνεται βάσει διαφορετικών υλοποιήσεων ConcreteBuilder. Όπως φαίνεται και στο σχήμα 4.7, οι υλοποιήσεις αυτές αναλαμβάνουν την διαδικασία δημιουργίας των επιμέρους αντικειμένων. Έτσι λοιπόν σε ένα σενάριο βάσει του οποίου πρέπει να αλλάξει η εσωτερική δομή του σύνθετου αντικειμένου το μόνο που χρειάζεται να γίνει είναι η επέκταση του στοιχείου αφαίρεσης με μια νέα υλοποίηση.




Σχήμα 4.7 η Δημιουργία του Σύνθετου Αντικειμένου με Builder

Παρατηρώντας τον πίνακά 4.2 μπορούμε να διαπιστώσουμε ότι η μετρική CBO ανά κλάση παρουσιάζει βελτίωση. Η χρήση του προτύπου συνέβαλε στην ανεξαρτητοποίηση του Director από το Product. Πλέον ο Director έχει εξάρτηση μόνο από το στοιχείο αφαίρεσης Builder. Αναφορικά με το WMC όπως φαίνεται και στον πίνακα 4.2 παρουσιάζει βελτίωση. Ο Director δεν έχει γνώση για την δημιουργία του σύνθετου αντικειμένου, παρέχει όμως έναν μηχανισμό για την αναπαράσταση του αντικείμενου ο οποίος μάλιστα μπορεί να χρησιμοποιηθεί για κάθε εσωτερική αναπαράσταση. Εξετάζοντας το LCOM και όπως φαίνεται και από το σχήμα 4.7 δεν υπάρχει εισαγωγή νέων μεθόδων συνεπώς δεν υπάρχει και μεταβολή της συγκεκριμένη μετρικής πριν και μετά την χρήση του προτύπου. Ενώ το RFC παρουσιάζει αύξηση, η εισαγωγή της ιεραρχίας στο σύστημα είχε ως αποτέλεσμα περισσότερη επικοινωνία μεταξύ των κλάσεων προκειμένου να γίνει η δημιουργία του σύνθετου αντικειμένου. Επιπλέον, αύξηση παρουσιάζει και το NOC όπως φαίνεται και στο σχήμα 4.7. Η χρήση του προτύπου οδηγεί στην εισαγωγή μιας ιεραρχίας κλάσεων. Έτσι λοιπόν πριν την χρήση του προτύπου δεν υπήρχαν ιεραρχίες, ενώ με την χρήση υπάρχει ένα στοιχείο αφαίρεσης, το NOC, του οποίου είναι ίσο με το πλήθος των διαφορετικών εσωτερικών αναπαραστάσεων του σύνθετου αντικειμένου. Συμπερασματικά, όπως φαίνεται και στον πίνακα 4.2 υπάρχει αύξηση του μέσου NOC στο σύστημα. Ανάλογα και με το DIT, η χρήση του προτύπου οδηγεί στην υλοποίηση κλάσεων με DIT ίσο με ένα.

Βάσει λοιπόν των μετρικών η χρήση του προτύπου Builder οδηγεί στην βελτίωση των CBO και WMC στο σύστημα. Πιο συγκεκριμένα το πρότυπο βοηθάει στην καλύτερη κατανομή των παραπάνω μετρικών σε όλο το σύστημα. Συνέπεια αυτού είναι ότι ο Director μένει ανεξάρτητος από την διαδικασία δημιουργίας σύνθετων αντικείμενων. Δεν είναι απαραίτητη δηλαδή η γνώση της διαδικασίας δημιουργίας των σύνθετων αντικειμένων. Έτσι λοιπόν ένα σενάριο συντήρησης εκεί που βοηθά το πρότυπο, είναι στην δημιουργία του σύνθετου αντικείμενου με διαφορετικές εσωτερικές αναπαραστάσεις. Σε αυτή την περίπτωση και βάσει του προτύπου η συντήρηση γίνεται με κλειστό τρόπο, με την επέκταση του στοιχείου αφαίρεσης Builder με νέες υλοποιήσεις, ενώ χωρίς την χρήση του προτύπου θα έπρεπε να επέμβουμε στον Director έτσι ώστε να υποστηρίζει την νέα σύνθεση για το σύνθετο αντικείμενο.

```
public class Director {
   private int i=1;
   private Product product;
   Director(Product product){
   this.product= product;
   }
   public void Construct(){
   if (i ==1){
   product.SetProduct(new String("product1"));
   else
   product.SetProduct(new String("product2"));
   }}}
```

Σχήμα 4.8 ο Director χωρίς Builder

```
public class Director {
  private Builder builder;
   Director(Builder builder){
   this.builder= builder;
   }

   public void Construct(){
      builder.createProduct();
      builder.buildPart();
      }}}
```

Σχήμα 4.9 ο Director με την Χρήση Builder

```
abstract class Builder {
   public Product product;
   public void createProduct(){
```

```
        product = new Product();
    }
     abstract public void buildPart();
}

public class ConcreteBuilder extends Builder {
    public void buildPart(){
    product.SetProduct(new String("product1"));
    }}

public class ConcerteBuilder2 extends Builder{
    public void buildPart(){
       product.SetProduct(new String("product2"));
    }}
```

Σχήμα 4.10 η Ιεραρχία του Προτύπου

```
public class Product {
    private String elements;

    public void SetProduct(String item){
        elements=item;
    }}
```

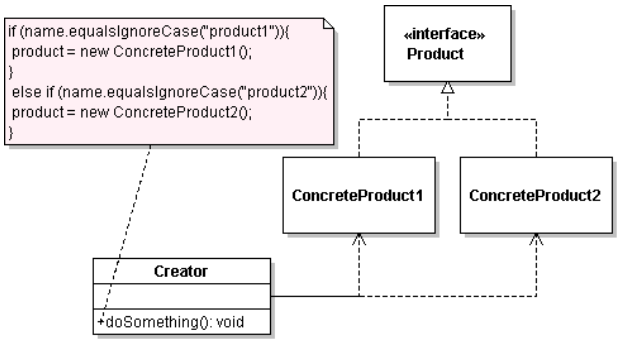
Σχήμα 4.11 το Σύνθετο Αντικείμενο

Πίνακας 4.2 οι Μεταβολές των Μετρικών με και χωρίς το Builder

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Director | 1 | 0 | 1 | 1 | 0 | 0 | 2 | 3 | 0 | 0 | 0 | 0 |
| Builder | | 1 | | 0 | | 0 | | 1 | | 2 | | 0 |
| ConcreteBuilder1 | | 0 | | 0 | | 0 | | 2 | | 0 | | 1 |
| Concrete Builder2 | | 0 | | 0 | | 0 | | 2 | | 0 | | 1 |
| Product | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Overall Mean | 0.5 | 0.2 | 0.5 | 0.2 | 0 | 0 | 1.5 | 1.8 | 0 | 0.4 | 0 | 0.4 |

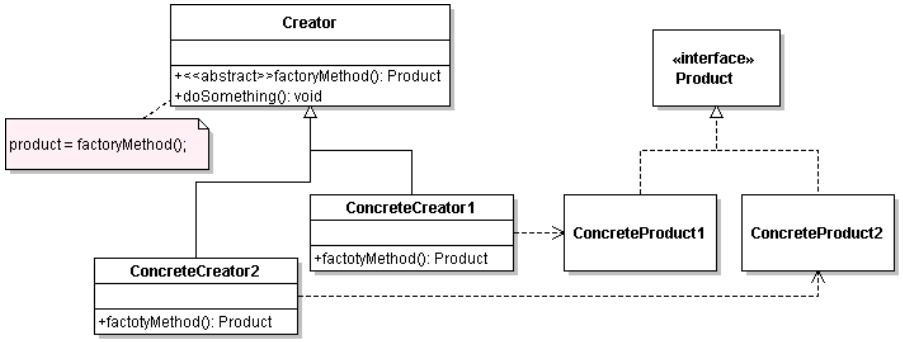
*4.2.3. Factory Method*

Το πρότυπο Factory Method παρέχει μια λύση σε συστήματα αποτελούμενα από μια ιεραρχία κλάσεων και μια κλάση Creator, μια κλάση δηλαδή που ενδιαφέρεται για την δημιουργία αντικειμένων της ιεραρχίας. Ζητούμενο του συστήματος είναι η δημιουργία αντικειμένων της ιεραρχία στον Creator. Μια πρώτη προσέγγιση είναι η δημιουργία των αντικειμένων στον Creator. Στο σχήμα 4.12 φαίνεται το διάγραμμα κλάσεων που περιγράφει αυτήν την προσέγγιση. Η κλάση που ενδιαφέρεται για την δημιουργία των αντικειμένων της ιεραρχίας εξαρτάται από κάθε υλοποίηση του στοιχείου αφαίρεσης Product. Δηλαδή η κλάση Creator έχει γνώση των διαδικασιών δημιουργίας αντικειμένων της κάθε υλοποίησης.




Σχήμα 4.12 Δημιουργώντας τα Αντικείμενα χωρίς Factory Method

Το πρότυπο δίνει μια λύση σύμφωνα με την οποία ο Creator αποτελεί μια αφηρημένη κλάση η οποία περιλαμβάνει υλοποιήσεις για το σύνολο των μεθόδων όμως τα αντικείμενα της ιεραρχίας δημιουργούνται από διαφορετικές υλοποιήσεις μιας αφηρημένης μεθόδου. Όπως φαίνεται και στο σχήμα 4.13 η αφηρημένη κλάση Creator ορίζει την μέθοδο factoryMethod(), ενώ η δημιουργία των αντικειμένων της ιεραρχίας γίνεται βάσει των διαφορετικών υλοποιήσεων της factoryMethod(). Με αυτόν τον τρόπο κάθε υλοποίηση έχει την δυνατότητα να δημιουργεί αντικείμενα μιας και μόνο κλάσης της ιεραρχίας. Δηλαδή η σχέση που υπάρχει ανάμεσα στις υλοποιήσεις του Creator και του στοιχείου αφαίρεσης της ιεραρχίας είναι ένα προς ένα.




Σχήμα 4.13 Δημιουργώντας τα Αντικείμενα Χρησιμοποιώντας Factory Method

Το CBO του συστήματος τείνει να βελτιώνεται όσο αυξάνει ο αριθμός των υλοποιήσεων της ιεραρχίας. Σύμφωνα και με το σχήμα 4.14, χωρίς την χρήση του προτύπου ο Creator πρέπει να έχει γνώση των διαδικασιών δημιουργίας διαμορφώνοντας το CBO του Creator ίσο με τον πληθάριθμο των υλοποιήσεων της ιεραρχίας. Ενώ χρησιμοποιώντας το πρότυπο επιτυγχάνεται η ανεξαρτησία του Creator από την ιεραρχία όπως αυτό φαίνεται στο σχήμα 4.15 παράλληλα οι εξαρτήσεις αυτές μεταφέρονται στις υλοποιήσεις της μεθόδου factoryMethod(). Λόγω της ένα προς ένα εξάρτησης μεταξύ των υλοποιήσεων του στοιχείου αφαίρεσης Creator και αυτών του Product, το CBO της κάθε υλοποίησης του Creator είναι ίσο με ένα. Ανάλογη τάση βελτίωσης παρουσιάζει και το WMC, χωρίς την χρήση του προτύπου, όπου ο Creator θα πρέπει να είναι εφοδιασμένος με μια πολλαπλή δομή επιλογής προκειμένου να γίνεται έλεγχος για την δημιουργία του σωστού αντικειμένου. Χρησιμοποιώντας το πρότυπο παρουσιάζεται μείωση της πολυπλοκότητας και αυτό γιατί δεν υπάρχει η συγκεκριμένη δομή επιλογής όπως δείχνει και το σχήμα 4.15 ενώ οι υποκλάσεις του Creator καθορίζουν την δημιουργία των αντικειμένων της ιεραρχίας. Αναφορικά με την συνεκτικότητα των κλάσεων δεν υπάρχει καμία μεταβολή. Σύμφωνα και με το σχήμα 4.13 το LCOM των κλάσεων της ιεραρχίας παραμένει αμετάβλητων, αυτό συμβαίνει γιατί δεν υπάρχει καμία μεταβολή στον βαθμό συνάφειας των μεθόδων στην κάθε κλάση της ιεραρχίας, παράλληλα οι κλάσεις ConcreteCreator δεν παρουσιάζουν έλλειψη συνεκτικότητας αφού υλοποιούν μόνο μια μέθοδο, συνεπώς στο σύστημα δεν παρουσιάζει καμία μεταβολή στο μέσο LCOM. Ακόμα μικρή αύξηση παρουσιάζει το RFC. Η αύξηση αυτή οφείλεται στην κλήση της μεθόδου factoryMethod() από την μέθοδο του Creator. Επιπλέον χρησιμοποιώντας το πρότυπο, υπάρχει αύξηση του NOC όπως φαίνεται και στο

σχήμα 4.13 η κλάση Creator έχει NOC ίσο με το πλήθος των υλοποιήσεων της μεθόδου factoryMethod(). Τέλος αύξηση παρουσιάζει και το DIT, οι νέες αυτές υλοποιήσεις παρουσιάζουν DIT ίσο με ένα και αυτό γιατί επεκτείνουν την κλάση Creator.

```
public class Creator {
   Product product;
   String name ="product1";
   public void anOperation (){
      if (name.equalsIgnoreCase("product1")){
        product = new ConcreteProduct1();
        product.ProductOpeation();
      }
      Else if (name.equalsIgnoreCase("product2")){
         product = new ConcreteProduct2();
         product.ProductOpeation();
      }}}
```

Σχήμα 4.14 η Κλάση Creator χωρίς το Πρότυπο.

```
public abstract class Creator {


   public void anOperation(){
Product product1 = factoryMethod();
        product1.ProductOpeation();
 }
   protected abstract Product factoryMethod();
}
```

Σχήμα 4.15 η Κλάση Creator με Χρήση του Προτύπου

```
public class ConcreteFactory1 extends Creator  {
   protected Product factoryMethod()
         {
         return new ConcreteProduct1();
         }
}

public class ConcreteFactory2 extends Creator  {
   protected Product factoryMethod()
         {
         return new ConcreteProduct2();
         }
}
```

Σχήμα 4.16 οι Επεκτάσεις του Creator

```
public interface Product {
   public void ProductOperation();
}
public class ConcreteProduct1 implements Product{
public ConcreteProduct1(){
System.out.println("this is");
}
   public void ProductOperation(){
   System.out.println("product1");
}}
public class ConcreteProduct1 implements Product{
public ConcreteProduct2(){
System.out.println("this is");}
 public void ProductOperation(){
   System.out.println("product2");
}}
```

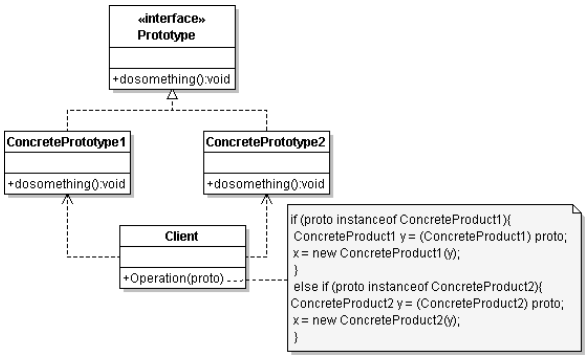
Σχήμα 4.17 Η Ιεραρχία του Συστήματος

Πίνακας 4.3 οι Μεταβολές των Μετρικών με και χωρίς το Factory Method

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Creator | 2 | 0 | 2 | 0 | 0 | 0 | 1 | 1 | 0 | 2 | 0 | 0 |
| ConcreteCreator1 | | 1 | | 0 | | 0 | | 1 | | 0 | | 1 |
| ConcreteCreator2 | | 1 | | 0 | | 0 | | 1 | | 0 | | 1 |
| Product | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| ConcreteProduct1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| ConcreteProduct2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Overall Mean | 1/2 | 1/3 | 0.5 | 0 | 0 | 0 | 0.25 | 0.5 | 0.5 | 0.6 | 0.5 | 0.6 |

*4.2.4. Prototype*

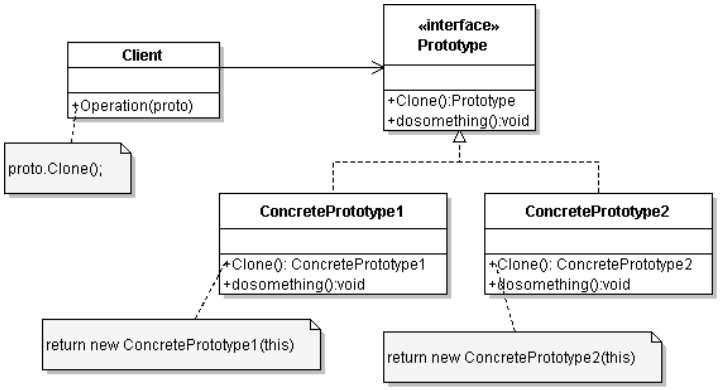
Το συγκεκριμένο πρότυπο παρέχει μια λύση σε περιπτώσεις στις οποίες θέλουμε να δημιουργήσουμε αντίγραφα κάποιου αντικειμένου. Έτσι λοιπόν αρχικά πρέπει να δημιουργηθεί με κάποιον τρόπο ένα αντικείμενο και στην συνέχεια να εφαρμοστεί κάποια τεχνική βάσει της οποίας θα δημιουργηθούν τα αντίγραφα. Έχοντας ως δεδομένη την δημιουργία ενός αντικειμένου της ιεραρχίας το ζητούμενο είναι πώς θα δημιουργηθούν τα αντίγραφα του αντικειμένου αυτού. Το σχήμα 4.18 παρουσιάζει μια πρώτη προσέγγιση του προβλήματος σύμφωνα με την οποία ο ίδιος ο Client αναλαμβάνει αυτήν την διαδικασία δημιουργίας αντιγράφου. Βάσει αυτής της προσέγγισης είναι απαραίτητος ο έλεγχος του τύπου του αντικειμένου και στην συνέχεια γίνεται η δημιουργία του αντιγράφου. Η όλη διαδικασία έχει ως αποτέλεσμα την εξάρτηση του Client από όλα τα στοιχεία της ιεραρχίας.




Σχήμα 4.18 Δημιουργία Αντιγράφων χωρίς Prototype

Η λύση που δίνει το πρότυπο Prototype φαίνεται στο σχήμα 4.19. Βάσει λοιπόν αυτής της προσέγγισης οι κλάσεις της ιεραρχία του συστήματος παρέχουν μια επιπλέον

μέθοδο μέσω της οποίας γίνεται η δημιουργία των αντιγράφων. Η προσέγγιση αυτή παρέχει ανεξαρτησία μεταξύ των υλοποιήσεων της ιεραρχίας και του Client. Πιο συγκεκριμένα όπως φαίνεται και από το σχήμα 4.21, ο Client καλεί την μέθοδο clone() πάνω σε ένα αντικείμενο τύπου Prototype. Με αυτόν τον τρόπο γίνεται η δημιουργία του αντιγράφου χωρίς να υπάρχει κάποιος έλεγχος για το πρωτότυπο.




Σχήμα 4.19 η Δημιουργία Αντιγράφων με χρήση Prototype

Εξετάζοντας το πρότυπο βάσει των μετρικών, είναι φανερό ότι υπάρχει βελτίωση στο CBO το οποίο είναι αποτέλεσμα της ανεξαρτησίας του Client από τις υλοποιήσεις της ιεραρχίας. Βελτίωση παρουσιάζει και το WMC, όπως φαίνεται και στο σχήμα 4.21. Με την χρήση του πρότυπου ο Client δεν δημιουργεί τα αντίγραφα, συνεπώς δεν έχει και δομή επιλογής προκειμένου να ελέγχει τον τύπο κάθε αντικειμένου. Αντίθετα το RFC παρουσιάζει αύξηση λόγο της κλήσης της μεθόδου clone από τον Client. Ακόμα αύξηση παρουσιάζει και το LCOM λόγω της εισαγωγής της μεθόδου clone. Στην πραγματικότητα αυτή η νέα μέθοδος δεν χρησιμοποιεί κανένα κοινό πεδίο με τις μεθόδους που προϋπήρχαν στην ιεραρχία. Τέλος οι μετρικές NOC και DIT δεν παρουσιάζουν καμιά μεταβολή. Το σχήμα 4.19 δείχνει ότι η χρήση του προτύπου δεν επηρεάζει τις κλάσεις της ιεραρχίας.

```
public class Client {
    Prototype x;
    public void Operation (Prototype proto){
    if (proto instanceof ConcretePrototype1){
        ConcretePrototype1 y = (ConcretePrototype1) proto;
         x = new ConcretePrototype1(y);
```

```
    }
    else if (proto instanceof ConcretePrototype2){
        ConcretePrototype2 y = (ConcretePrototype2) proto;
         x = new ConcretePrototype2(y);
    }}}
```

Σχήμα 4.20 ο Client χωρίς την Χρήση του Prototype

```
public class Client {
   Prototype x;
   public Prototype Operation (Prototype proto){
    x = proto;
   return x.clone();
   }}
```

Σχήμα 4.21 ο Client με Χρήση του Prototype

```
abstract class Prototype {
   abstract void anOperation();
}

public class ConcretePrototype1 extends Prototype {
   private ConcretePrototype1 a;
   ConcretePrototype1 (){}
   ConcretePrototype1(ConcretePrototype1 proto){
   a = proto;
   }
   public void anOperation(){
   System.out.println("This is an operation from ConcreteProduct1");
   }}

public class ConcretePrototype2 extends Prototype {
   private ConcretePrototype2 a;
```

```
   ConcretePrototype2 (){}
   ConcretePrototype2(ConcretePrototype2 proto){
   a = proto;
   }
  public void anOperation(){
  System.out.println("This is an operation from ConcreteProduct2");
  }}
```

Σχήμα 4.22 η Ιεραρχία του Συστήματος

```
abstract class Prototype {
   abstract public void anOperation();
   abstract public Prototype clone();
}

public class ConcretePrototype1 extends Prototype {
   private ConcretePrototype1 a;
   ConcretePrototype1 (){}
   ConcretePrototype1(ConcretePrototype1 proto){
   a = proto;
   }
   public void anOperation(){
   System.out.println("This is an operation from ConcreteProduct1");
   }

   public Prototype clone(){
      System.out.println("creating copy ConcreteProduct1");
      return new ConcretePrototype1(this);
   }}

public class ConcretePrototype2 extends Prototype {
   private ConcretePrototype2 a;
   ConcretePrototype2 (){}
```

```
ConcretePrototype2(ConcretePrototype2 proto){
a = proto;
}
public void anOperation(){
System.out.println("This is an operation from ConcreteProduct2");
}

public Prototype clone(){
   System.out.println("creating copy ConcreteProduct2");
   return new ConcretePrototype2(this);
}}
```

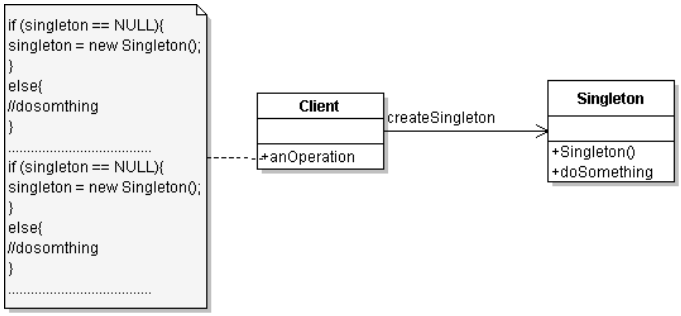
Σχήμα 4.23 ο Client χωρίς Χρήση Prototype

Πίνακας 4.4 οι Μεταβολές των Μετρικών με και χωρίς το Prototype

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 2 | 0 | 2 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| Prototype | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| Concrete Prototype1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0 | 0 | 1 | 1 |
| Concrete Prototype2 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0 | 0 | 1 | 1 |
| Overall Mean | 0.5 | 0 | 0.5 | 0 | 0 | 0.5 | 0.75 | 1.5 | 0.5 | 0.5 | 0.5 | 0.5 |

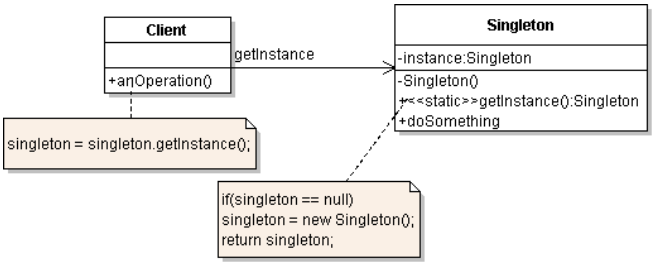
*4.2.5. Singleton*

Το πρότυπο Singleton χρησιμοποιείται σε περίπτωση που επιτρέπεται η δημιουργία ενός και μόνου αντικειμένου μιας συγκεκριμένης κλάσης. Παράλληλα υπάρχει η δυνατότητα πρόσβασης στο συγκεκριμένο αντικείμενο από διάφορα σημεία του κώδικα του Client. Το σχήμα 4.24 δείχνει μια πρώτη προσέγγιση βάσει της οποίας ο Client αναλαμβάνει την ευθύνη του έλεγχου για την δημιουργία του αντικειμένου. Έτσι λοιπόν για οποιοδήποτε λειτουργία του Client που χρησιμοποιεί το αντικείμενο θα πρέπει να υπάρχει και ο ανάλογος έλεγχος.




Σχήμα 4.24 ο Client χωρίς Χρήση Singleton

Το πρότυπο δίνει μια λύση βάσει της οποίας η κλάση ελέγχει την μοναδικότητα του αντικειμένου. Η διαδικασία αυτή γίνεται ορίζοντας τον constructor ως private και χρησιμοποιώντας μια static μέθοδο η οποία επιστρέφει στιγμιότυπα της κλάσης.




Σχήμα 4.25 ο Client με Χρήση Singleton

Αναφορικά με τις μετρικές, βελτίωση παρουσιάζει το WMC. Αυτό συμβαίνει γιατί με τη χρήση του προτύπου, ο κώδικας του Client δεν χρειάζεται να περιλαμβάνει δομές ελέγχου της μοναδικότητας του αντικειμένου, ενώ αύξηση παρουσιάζει το RFC, όπως φαίνεται και στο σχήμα 4.27. Ο Client καλεί μια επιπλέον μέθοδο εξωτερικά ενώ σύμφωνα με το σχήμα 4.26 η κλάση του μοναδικού αντικειμένου έχει μια παραπάνω μέθοδο, την στατική μέθοδο δηλαδή, που δημιουργεί το μοναδικό αντικείμενο. Η μέθοδος αυτή ευθύνεται για την αύξηση του LCOM στο σύστημα. Τέλος οι υπόλοιπες μετρικές δεν παρουσιάζουν καμία μεταβολή.

```
public class singleton {

 private static singleton single = null;
public void doSomething(){
```

```
System.out.println("this is an operation")
}
 private singleton(){
 System.out.println("this is the singleton object");
 }
 public static singleton getinstance(){
   if (single == null){
   single = new singleton();
   }
   return single;
 }
}
```

Σχήμα 4.26 η Κλάση με Χρήση Singleton

```
public class Client {
   public static void main(String[] args) {
      singleton sing1 =singleton.getinstance();
      System.out.println(sing1);
      System.out.println("//////////////////////////////");
      System.out.println("/////////////code/////////////");
      System.out.println("/////////////code/////////////");
      System.out.println("//////////////////////////////");

      sing1 =singleton.getinstance();
      System.out.println(sing1);
   }
}
```

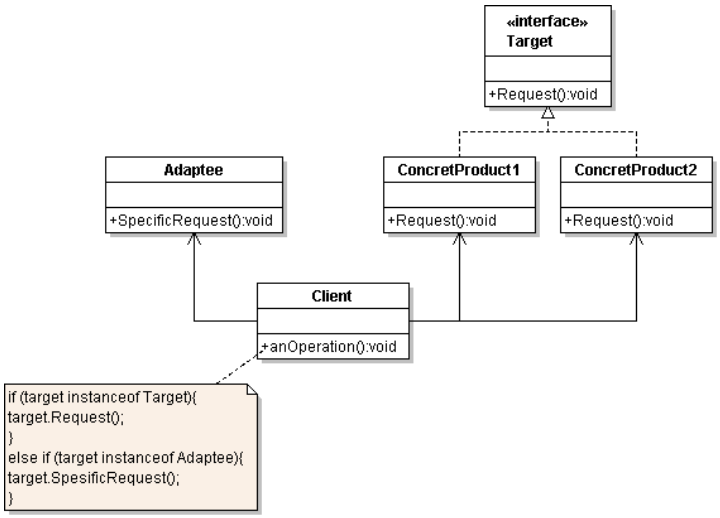
Σχήμα 4.27 ο Client με Χρήση Singleton

Πίνακας 4.5 οι Μεταβολές των Μετρικών με και χωρίς το Singleton

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 1 | 1 | 2 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| Singleton | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 0 | 0 | 0 | 0 |
| Overall Mean | 0.5 | 0.5 | 1 | 0.5 | 0 | 0.5 | 1 | 2 | 0 | 0 | 0 | 0 |

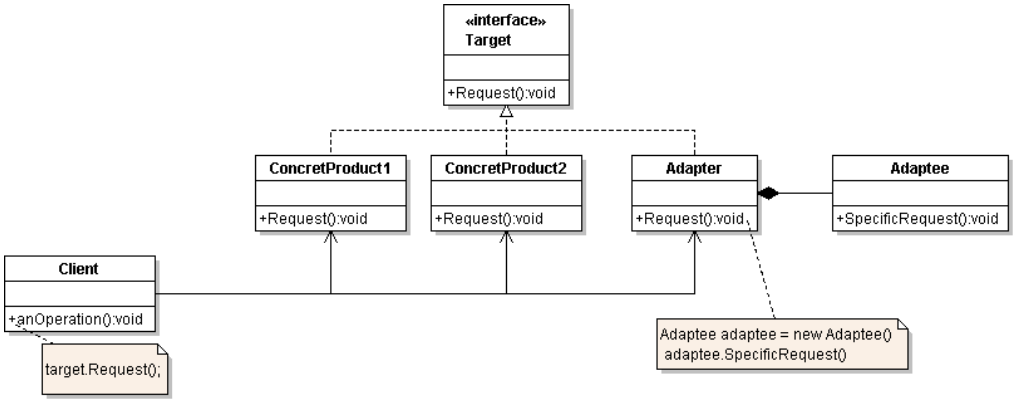
*4.2.6. Adapter*

Το πρότυπο Adapter δίνει μια λύση σε συστήματα τα οποία αποτελούνται από μια ιεραρχία κλάσεων και τον Client. Έστω λοιπόν ότι θέλουμε να επεκτείνουμε την ιεραρχία με νέες υλοποιήσεις. Οι υλοποιήσεις αυτές υποστηρίζουν τις ίδιες λειτουργίες, υλοποιούν όμως διαφορετική διεπαφή. Το ζητούμενο λοιπόν είναι ο τρόπος με τον οποίο θα εισάγουμε στο σύστημα τις νέες αυτές κλάσεις. Μια πρώτη προσέγγιση φαίνεται και στο σχήμα 4.28, σύμφωνα με την οποία υπάρχει απευθείας συσχέτιση του Client με τέτοιες υλοποιήσεις. Το βασικό πρόβλημα που παρουσιάζει αυτή η προσέγγιση φανερώνεται στον Client. Όπως δείχνει και το σχήμα 4.34 κάθε φορά που ο Client θέλει να εκτελέσει την συγκεκριμένη λειτουργία θα πρέπει να ελέγξει τον τύπο του αντικειμένου και στην συνέχεια να καλεί την αντίστοιχη μέθοδο.




Σχήμα 4.28 το Σύστημα χωρίς Χρήση Adapter

Το πρότυπο δίνει μια λύση χρησιμοποιώντας το χαρακτηριστικό αυτής της κλάσης. Δηλαδή το ότι παρέχει την ίδια λειτουργία με την ιεραρχία αλλά χρησιμοποιεί διαφορετική διεπαφή. Σύμφωνα με το σχήμα 4.29, χρησιμοποιώντας το πρότυπο οι επεκτάσεις αυτές μπορούν να γίνουν συμβατές με το στοιχείο αφαίρεσης της ιεραρχίας. Έτσι λοιπόν για κάθε επέκταση του συστήματος ορίζεται μια κλάση adapter η οποία είναι συμβατή και υλοποιεί το στοιχείο αφαίρεσης της ιεραρχίας. Σε κάθε κλάση adapter υπάρχει αναφορά σε ένα αντικείμενο από τις μη συμβατές επεκτάσεις. Έτσι λοιπόν καλώντας τις μεθόδους της κλάσης adapter εκτελούνται οι μέθοδοι της μη συμβατής επέκτασης.




Σχήμα 4.29 το Σύστημα με Χρήση Adapter

Εξετάζοντας το σύστημα βάσει των μετρικών, βελτίωση παρουσιάζει το WMC. Όπως φαίνεται και στον πίνακα 4.5 το WMC ανά κλάση μειώνεται. Αυτό συμβαίνει γιατί ο Client δεν πραγματοποιεί έλεγχο προκείμενου να διαπιστώσει τον τύπο του αντικειμένου και να καλέσει την αντίστοιχη μέθοδο. Στο σχήμα 4.33 φαίνεται η κλήση της μεθόδου με όμοιο τρόπο για όλες τις κλάσεις του συστήματος. Παράλληλα βελτιώνεται και το RFC, η χρήση του πρότυπου κάνει τον Client να καλεί μόνο μια μέθοδο εξωτερικά για την εκτέλεση της λειτουργίας. Ακόμα εξετάζοντας το CBO γίνεται φανερό ότι η χρήση του προτύπου διατηρεί τις υπάρχουσες συσχετίσεις και χρησιμοποιεί νέες συνθέσεις. Όπως φαίνεται και στο σχήμα 4.32 η κλάση Adapter δημιουργεί ένα αντικείμενο Adaptee. Συνεπώς το CBO ανά κλάση στο σύστημα επιβαρύνεται. Τέλος οι μετρικές NOC και DIT παρουσιάζουν αύξηση. Ο λόγος είναι ότι βάσει του προτύπου οι νέες κλάσεις προσαρμόζονται την ιεραρχία. Σύμφωνα με το σχήμα 4.32 η κλάση Adaptee υλοποιεί την διεπαφή του συστήματος συνεπώς έχει DIT ίσο με ένα, ενώ προκαλεί αύξηση στο NOC της διεπαφής. Άρα, όπως φαίνεται και στον πίνακα 4.5 ο μέσος όρος των μετρικών της ιεραρχίας ανά κλάση αυξάνεται.

```
public interface Target {
   public void anOperation();
}


public class ConcreteProduct1 implements Target {
   public void Request ()
   {
```

```
    System.out.println("this is Concrete Product 1");
    }
}

public class ConcreteProduct2 implements Target {
    public void Request ()
    {
    System.out.println("this is Concrete Product 2");
    }
}
```

Σχήμα 4.30 η Ιεραρχία του Συστήματος

```
public class Adaptee {

    public void SpecificRequest (){
        System.out.print("this is an Adaptee ");
    }}
```

Σχήμα 4.31 η Κλάση Adaptee

```
public class Adapter implements Target{
Adaptee ad;

    public Adapter()
    {
     ad = new Adaptee();
    }
 public void Request (){
       ad. SpecificRequest ();
  }
}
```

Σχήμα 4.32 η Κλάση Adapter

```
public class Client {
    public static void main(String[] args) {
        Target target = new Adapter();
        target.Request();
    }}
```

Σχήμα 4.33 ο Client με Χρήση Adapter

```
public class Client {
    public static void main(String[] args) {
        Object target = null;

        if (target instanceof Target ){
        Target t1=(Target)target;
        t1.Request();
        }
        else if (target instanceof Adaptee){
        Adaptee t2 = (Adaptee)target;
        t2.SpecificRequest();
        }}  }
```

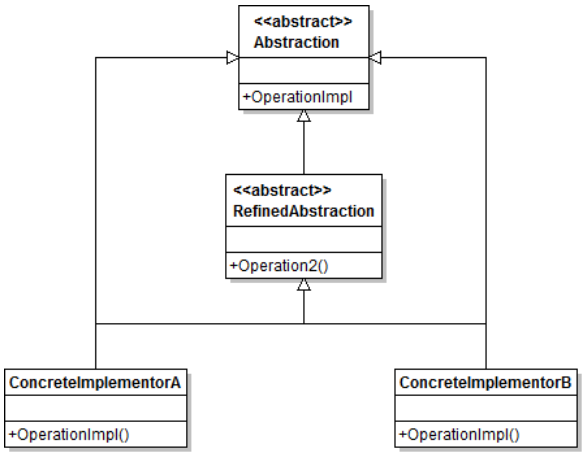
Σχήμα 4.34 ο Client χωρίς Χρήση Adapter

Πίνακας 4.6 οι Μεταβολές των Μετρικών με και χωρίς το Adapter

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 3 | 3 | 2 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 | 0 |
| Target | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 | 0 | 0 |
| ConcreteProduct1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ConcreteProduct2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Adaptee | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Adapter | | 1 | | 0 | | 0 | | 2 | | 0 | | 1 |
| Overall Mean | 0.6 | 0.66 | 0.4 | 0 | 0 | 0 | 1.2 | 1.16 | 2/5 | 3/6 | 2/5 | 3/6 |

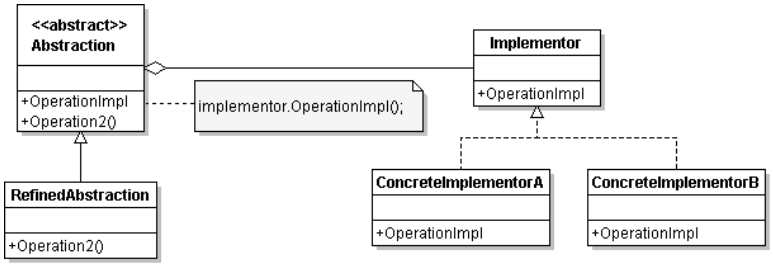
*4.2.7. Bridge*

Το πρότυπο Bridge δίνει μια λύση σε περιπτώσεις που έχουμε μια ιεραρχία κλάσεων και θέλουμε να επεκτείνουμε την διεπαφή με τέτοιον τρόπο που οι κλάσεις της ιεραρχίας να υποστηρίζουν νέες λειτουργίες. Ζητούμενο λοιπόν είναι ο τρόπος που θα γίνει η επέκταση με όσο το δυνατόν μικρότερο αντίκτυπο στην δομή της ιεραρχίας. Μια πρώτη προσέγγιση φαίνεται στο σχήμα 4.35 βάσει αυτής. Η εισαγωγής της νέας λειτουργίας στην ιεραρχία πραγματοποιείται μέσω επέκτασης του στοιχείου αφαίρεσης RefineAbstraction. Αυτό έχει ως αποτέλεσμα αυξημένη σύζευξη μεταξύ των κλάσεων τις ιεραρχίας καθώς εφαρμόζουν την διεπαφή αλλά ταυτόχρονα επεκτείνουν και την αφηρημένη κλάση.




Σχήμα 4.35 η Ιεραρχία χωρίς Χρήση Bridge

Το πρότυπο δίνει μια λύση όμοια με το σχήμα 4.36, δηλαδή γίνεται διαχωρισμός της ιεραρχίας και των επεκτάσεων του στοιχείου αφαίρεσης. Το πρότυπο υποστηρίζει την χρήση δυο ιεραρχιών, η πρώτη συσχετίζει στοιχεία αφαίρεσης μέσω κληρονομικότητας, ενώ η δεύτερη συσχετίζει κλάσεις που χρησιμοποιούνται στην υλοποίηση των κλάσεων της πρώτης ιεραρχίας. Η σύνδεση των δυο ιεραρχιών πραγματοποιείται με συνάθροιση, καθώς στο στοιχείο αφαίρεσης Abstraction, υπάρχει μια αναφορά Implementor.




Σχήμα 4.36 η Ιεραρχία με Χρήση Bridge

Ως αποτέλεσμα της χρήση του προτύπου είναι η ανεξαρτησία μεταξύ ιεραρχίας και των επεκτάσεων των λειτουργιών. Παρατηρώντας τις μετρικές στον πίνακα 4.7, φαίνεται βελτίωση στις μετρικές NOC και DIT. Η μείωση στην τιμή αυτών των μετρικών οδηγεί στην ελάττωση των ζεύξεων που οφείλονται στην κληρονομικότητα και βοηθούν στην δημιουργία μιας πιο απλής δομής για την ιεραρχία. Η χρήση του προτύπου έχει επιπτώσεις στο RFC. Το πρότυπο Bridge απαιτεί επικοινωνία μεταξύ της ιεραρχίας και των επεκτάσεων και πιο συγκεκριμένα η αφηρημένη κλάση Abstraction καλεί μια εξωτερική μέθοδο, η κλήση της οποίας προκαλεί την αύξηση στο RFC του συστήματος ανά κλάση. Τέλος οι υπόλοιπες μετρικές δεν παρουσιάζουν καμία μεταβολή.

```
abstract class Abstraction {
   abstract void OperationImpl();
}

abstract class RefinedAbstraction extends Abstraction {
 void Operation2(){
 System.out.println("operation2");
 }}

public class ConcreteImplementorA extends
RefinedAbstraction extends Abstraction {
   public void OperationImpl(){
      System.out.println("ConcreteImplementorA");
   }}
```

```
public class ConcreteImplementorB extends
RefinedAbstraction extends Abstraction {
   public void OperationImpl(){
      System.out.println("ConcreteImplementorB");
   }}
```

Σχήμα 4.37 η Ιεραρχία χωρίς Χρήση Bridge

```
public interface Implementor {
   void OperationImpl();
}

public class ConcreteImplementorA implements Implementor {
   public void OperationImpl(){
      System.out.println("ConcreteImplementorA");
   }}

public class ConcreteImplementorB implements Implementor {
   public void OperationImpl(){
      System.out.println("ConcreteImplementorB");
   }}
```

Σχήμα 4.38 η Ιεραρχία με Χρήση Bridge

```
public abstract class Abstraction {
  Implementor implementor;

   public void Operation(){
   implementor.OperationImpl();
   }
   abstract void Operation2();
}
```

```
public class RefinedAbstraction extends Abstraction {

RefinedAbstraction(Implementor implementor){
this.implementor = implementor;
}
public void Operation2(){
System.out.println("operation2");
}}
```

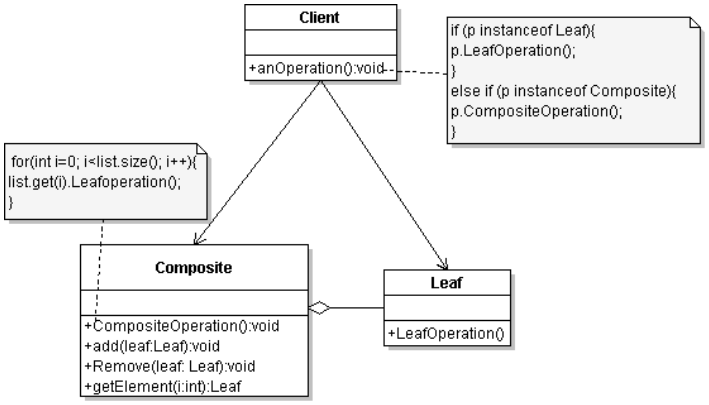
Σχήμα 4.39 η Επέκταση Βάσει του Προτύπου Bridge

Πίνακας 4.7 οι Μεταβολές των Μετρικών με και χωρίς το Bridge

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Abstraction | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 | 1 | 0 | 0 |
| Concrete ImplementorA | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 |
| Concrete ImplementorB | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 |
| Refined Abstraction | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0 | 1 | 1 |
| Implementor | | 0 | | 0 | | 0 | | 0 | | 2 | | 0 |
| Overall Mean | 0 | 0 | 0 | 0 | 0 | 0 | 0.75 | 1 | 1.25 | 0.6 | 1.25 | 0.6 |

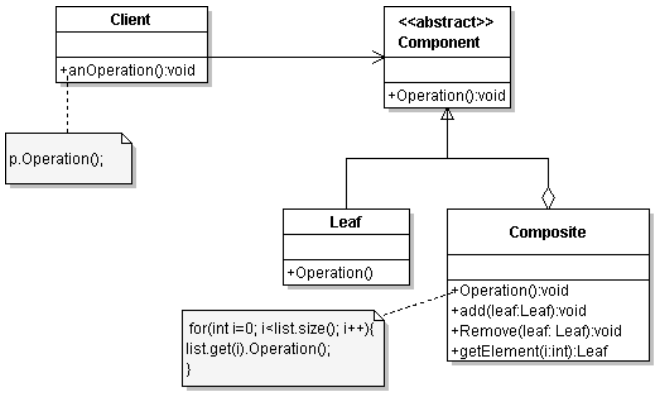
### *4.2.8. Composite*

Το πρότυπο Composite έχει εφαρμογή σε συστήματα αποτελούμενα από απλά και σύνθετα αντικείμενα. Ως σύνθετα αντικείμενα θεωρούνται εκείνα τα αντικείμενα που συναθροίζουν απλά αντικείμενα. Η σχεδίαση ενός τέτοιου συστήματος θα μπορούσε να γίνει με τον τρόπο που δείχνει το σχήμα 4.40, χρήση δηλαδή της συνάθροισης μεταξύ απλών και σύνθετων αντικειμένων. Το βασικό πρόβλημα που έχει αυτή η προσέγγιση είναι ότι δεν επιτρέπει στον Client έναν ομοιόμορφο τρόπο διαχείρισης των διαφορετικών αντικειμένων. Όπως δείχνει και το σχήμα 4.44 ο Client πρέπει πρώτα να ελέγξει των τύπο του αντικειμένου και στην συνέχεια να καλέσει την αντίστοιχη μέθοδο.




Σχήμα 4.40 το Σύστημα χωρίς Χρήση Composite

Το πρότυπο δίνει μια λύση οργανώνοντας τις κλάσεις τους συστήματος σε μια δενδροειδή μορφή όπως φαίνεται στο σχήμα 4.41. Βάσει αυτής της προσέγγισης ο Client του συστήματος επικοινωνεί με τις κλάσεις μέσω μιας κοινής διεπαφής ενώ όλες οι κλάσεις, απλές και σύνθετες, υλοποιούν την ίδια διεπαφή. Τέλος όπως δείχνει και το σχήμα 4.43 η δημιουργία ενός σύνθετου αντικειμένου γίνεται μέσω συνάθροισης αντικειμένων τις ιεραρχίας αυτής.




Σχήμα 4.41 το Σύστημα με Χρήση Composite

Η χρήση του προτύπου βελτιώνει κάποιες από τις μετρικές του συστήματος. Όπως φαίνεται και από τον πίνακα 4.8 βελτίωση παρουσιάζει το RFC. Ο ενιαίος τρόπος με τον οποίο ο Client καλεί την συγκεκριμένη λειτουργία των αντικειμένων, μειώνει τις εξωτερικές μεθόδους που καλεί. Επιπλέον η χρήση της διεπαφής μειώνει τις συσχετίσεις του Client με τα αντικείμενα συνεπώς υπάρχει και μείωση του CBO ανά κλάση. Ακόμα η χρήση του προτύπου δεν απαιτεί έλεγχο του Client για τον τύπο του

αντικειμένου προκειμένου να εκτελέσει την κατάλληλη μέθοδο και συνεπώς παρουσιάζει και μείωση της πολυπλοκότητας λόγο της απουσίας δομών ελέγχου. Αναφορικά με το LCOM δεν υπάρχει κάποια μεταβολή αφού ο αριθμός των μεθόδων στις κλάσεις δεν αλλάζει. Τέλος επιβάρυνση παρουσιάζουν οι μετρικές DIT και NOC καθώς η οργάνωση των κλάσεων σε μορφή ιεραρχίας οδήγησε στην αύξηση αυτών των μετρικών. Χρησιμοποιώντας το πρότυπο Composite, το σύστημα θα έχει μια διεπαφή με NOC ίσο με το πλήθος των κλάσεων ενώ η κάθε κλάση θα έχει DIT ίσο με ένα και αυτό γιατί θα υλοποιεί την κοινή διεπαφή.

```
public class leaf{
    public void Operation(){
    System.out.println("Leaf");
    }}
```

Σχήμα 4.42 η Κλάση leaf

```
public class Composite {
    private ArrayList<leaf> Arcomp;

    Composite(ArrayList<leaf> Arcomp){
      this.Arcomp =Arcomp;
    }
   public void CompositeOperation(){
    for (int i =0; i<Arcomp.size();i++){
       Arcomp.get(i).Operation();
    } }
    public void Add(leaf leaf){
       Arcomp.add(leaf);
    }
    public void Remove (leaf leaf){
       Arcomp.remove(leaf);
   }
    public leaf GetChild (int i){
```

```
    return Arcomp.get(i);
    }}
```

Σχήμα 4.43 η Κλάση Composite χωρίς το Πρότυπο

```
public class Client {
   public void anOperation{
      if (object instanceof leaf ){
       l = (leaf)object;
       l.Operation();
      }
      else if (object instanceof Composite ){
       c = (Composite)object;
       c.CompositeOperation();
      }}}
```

Σχήμα 4.44 ο Client χωρίς το Πρότυπο

```
public interface Component {
   public void Operation();
}

public class leaf implements  Component {
   private String id;
   private String name;
   leaf (String id, String name){
   this.id = id;
   this.name = name;
   }

   public void Operation(){
    System.out.println("leaf");
   }}
```

```
public class Composite implements Component{
   private ArrayList<Component> Arcomp;
   Composite(ArrayList<Component> Arcomp){
    this.Arcomp =Arcomp;
   }
   public void Operation(){
   System.out.println("Composite");
   }
   public void Add(Component component){
      Arcomp.add(component);
   }
   public void Remove (Component component){
      Arcomp.remove(component);
   }
   public Component GetChild (int i){
   return Arcomp.get(i);
   }}
```

Σχήμα 4.45 ο Οργάνωση των Κλάσεων leaf και Composite σε Ιεραρχία

```
public class Client {
      Component c1 = new leaf();
       anOperation(){
            c1.Operation();
         }}
```

Σχήμα 4.46 ο Client του Συστήματος με Χρήση του Προτύπου

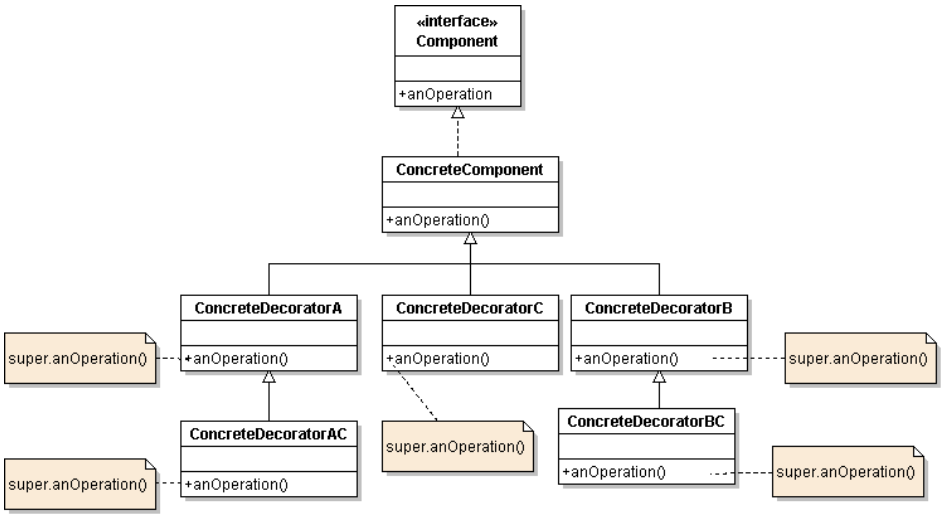
Πίνακας 4.8 οι Μεταβολές των Μετρικών με και χωρίς το Composite

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 2 | 0 | 2 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 | 0 |
| Composite | 1 | 0 | 1 | 1 | 0 | 0 | 5 | 5 | 0 | 0 | 0 | 1 |
| Leaf | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Component | | 0 | | 0 | | 0 | | 0 | | 2 | | 0 |
| Overall Mean | 1 | 0 | 1 | 0.25 | 0 | 0 | 3 | 2 | 0 | 0.5 | 0 | 0.5 |

*4.2.9. Decorator*

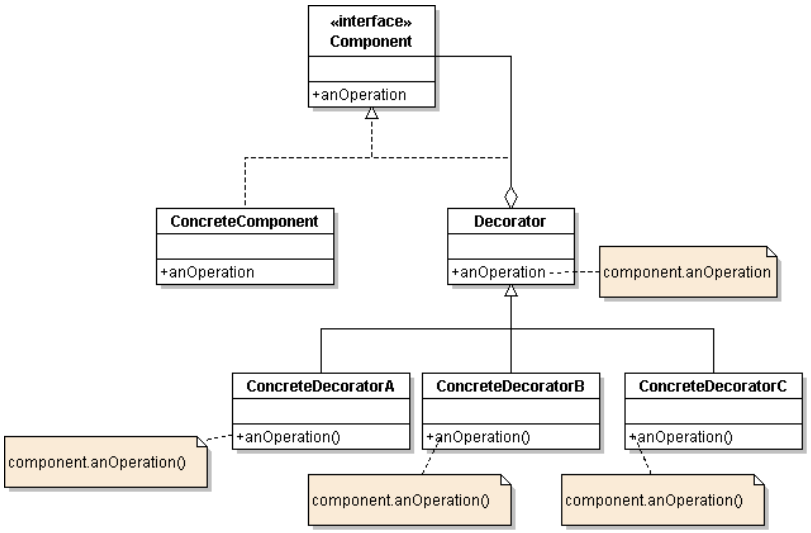
Το πρότυπο Decorator παρέχει μια λύση σε συστήματα που οι κλάσεις τους είναι οργανωμένες σε μια ιεραρχία. Το πρόβλημα που αντιμετωπίζει είναι η επέκταση των μεθόδων των κλάσεων έτσι ώστε να εκτελούν πρόσθετες λειτουργίες. Μια πρώτη προσέγγιση δείχνει το σχήμα 4.47. Σύμφωνα με αυτήν η επέκταση των δυνατοτήτων των μεθόδων μπορεί να γίνει μέσω επέκτασης των κλάσεων. Έτσι λοιπόν στην περίπτωση που θέλουμε να εισάγουμε τρείς νέες επεκτάσεις, θα πρέπει να υλοποιήσουμε πέντε νέες κλάσεις. Πιο συγκεκριμένα, το ζητούμενο δεν είναι μόνο οι επεκτάσεις των μεθόδων ξεχωριστά, αλλά μας ενδιαφέρουν και οι μεταξύ τους συνδυασμοί.




Σχήμα 4.47 η Ιεραρχία χωρίς Χρήση Decorator

Το πρότυπο δίνει μια λύση βάσει της οποίας γίνεται χρήση μια κλάσης Decorator η οποία υλοποιεί το στοιχείο αφαίρεσης της ιεραρχίας. Η κλάση αυτή έχει μια αναφορά στο στοιχείο αφαίρεσης και με αυτόν τον τρόπο είναι δυνατόν να συναθροίζει αντικείμενα των κλάσεων της ιεραρχίας, ενώ δημιουργούνται και υποκλάσεις στην Decorator για κάθε επέκταση. Όπως φαίνεται και στο σχήμα 4.51 οι επεκτάσεις είναι τρείς και αυτό συμβαίνει γιατί δεν χρειάζονται υλοποιήσεις για τους συνδυασμούς

των επεκτάσεων. Έτσι λοιπόν η κλήση της μεθόδου του Decorator έχει ως αποτέλεσμα την κλήση της μεθόδου του αντικειμένου που συναθροίζει ενώ παράλληλα εκτελείται και ο κώδικάς της επέκτασης. Τέλος λόγω της συνάθροισης των αντικειμένων της ιεραρχίας στον Decorator είναι δυνατόν να γίνει οποιοσδήποτε συνδυασμός μεταξύ υλοποίησης της ιεραρχίας και νέων επεκτάσεων.




Σχήμα 4.48 η Ιεραρχία με Χρήση Decorator

Εξετάζοντας τις μετρικές όπως αυτές παρουσιάζονται στον πίνακα 4.9 σημαντική βελτίωση φαίνεται στις μετρικές DIT και NOC. Αυτό συμβαίνει λόγο της μείωσης των επεκτάσεων της ιεραρχίας. Δηλαδή το γεγονός ότι δεν υπάρχουν οι κλάσεις που υλοποιούν συνδυασμό επεκτάσεων, δεν επιβαρύνει την ιεραρχία με κλάσεις που παρουσιάζουν μεγάλο βάθος κληρονομικότητας. Κάτι ανάλογο συμβαίνει και με το RFC. Όπως φαίνεται και στον πίνακα 4.9 οι κλάσεις που υποστηρίζουν συνδυασμό επεκτάσεων επιβαρύνουν το σύστημα με RFC ίσο με δυο και συνεπώς η απουσία τους οδηγεί στην βελτίωση αυτής της μετρικής. Τέλος οι υπόλοιπες μετρικές δεν επηρεάζονται αφού δεν υπάρχει καμία αλλαγή αναφορικά με την πολυπλοκότητα αλλά ούτε και εισαγωγή νέων μεθόδων σε υπάρχουσες κλάσεις.

```
public interface Componet {
   public void anOperation ();
}
```

```
public class ConcreteComponet implements Componet {
   ConcreteComponet(){
   System.out.println("This is ConcreteComponet1");
    }
    public void anOperation (){
       System.out.println("This is anOperation");
    }}
```

Σχήμα 4.49 το Στοιχείο Αφαίρεσης και η Υλοποίηση

```
public class ConecreteDecoratorA extends ConcreteComponet{
  public void anOperation(){
   super.anOperation();
   System.out.println("this is the extend ConecreteDecoratorA");
}}

public class ConecreteDecoratorB extends ConcreteComponet{
  public void anOperation(){
   super.anOperation();
   System.out.println("this is the extend ConecreteDecoratorB");
}}

public class ConecreteDecoratorC extends ConcreteComponet {
   public void anOperation(){
   super.anOperation();
   System.out.println("this is the extend ConecreteDecoratorC");
}}

public class ConecreteDecoratorAC extends ConecreteDecoratorA {
   public void anOperation(){
   super.anOperation();
   System.out.println("this is the extend ConecreteDecoratorC");
  }}
```

```
public class ConecreteDecoratorBC extends ConcreteComponet{
  public void anOperation(){
   super.anOperation();
   System.out.println("this is the extend ConecreteDecoratorC");
  }}
```

Σχήμα 4.50 οι Επεκτάσεις χωρίς Decorator

```
public class Decorator implements Componet {
    Componet compon;
    public Decorator(Componet other_compon){
        compon = other_compon;
    }
    public void anOperation(){
     compon.anOperation();
    }}

public class ConecreteDecoratorA extends Decorator{
   public ConecreteDecoratorA(Componet other_compon){
   super(other_compon);
   }
   public void anOperation(){
    compon.anOperation();
    System.out.println("this is the extend ConecreteDecorator1");
   }}

public class ConecreteDecoratorB extends Decorator
{
  public ConecreteDecoratorB(Componet other_compon){
   super(other_compon);
   }
   public void anOperation(){
    compon.anOperation();
```

```
   System.out.println("this is the extend ConecreteDecorator1");
  }}

public class ConecreteDecoratorC extends Decorator{
public ConecreteDecoratorC(Componet other_compon){
  super(other_compon);
  }
   public void anOperation(){
   compon.anOperation();
   System.out.println("this is the extend ConecreteDecorator1");
  }}
```

Σχήμα 4.51 οι Επεκτάσεις με Χρήση Decorator

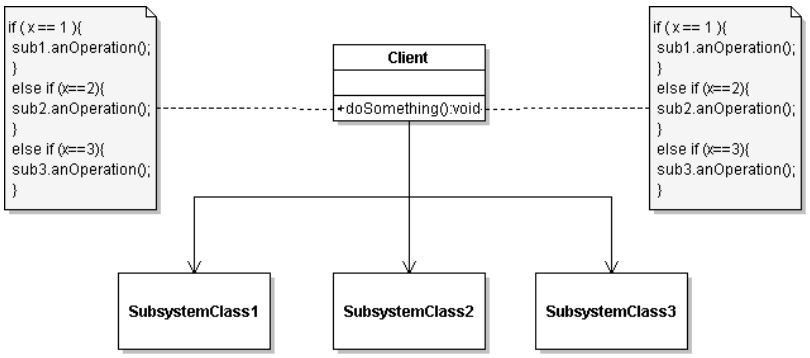
Πίνακας 4.9 οι Μεταβολές των Μετρικών με και χωρίς το Decorator

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Component | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 |
| Concrete Component | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 0 | 1 | 1 |
| Decorator | | 0 | | 0 | | 0 | | 2 | | 3 | | 1 |
| Concrete DecoratorA | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 1 | 0 | 2 | 2 |
| Concrete DecoratorB | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 1 | 0 | 2 | 2 |
| Concrete DecoratorC | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 2 | 2 |
| Concrete DecoratorAC | 0 | | 0 | | 0 | | 2 | | 0 | | 3 | |
| Concrete DecoratorBC | 0 | | 0 | | 0 | | 2 | | 0 | | 3 | |
| Overall Mean | 0 | 0 | 0 | 0 | 0 | 0 | 1.57 | 1.5 | 0.85 | 0.83 | 13/7 | 8/6 |

*4.2.10. Façade*

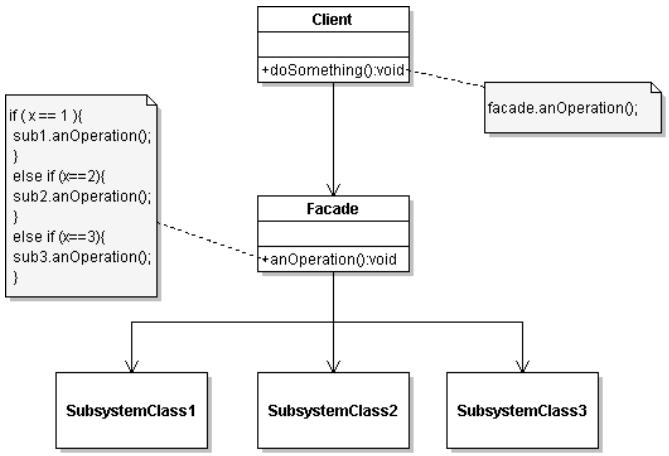
Το πρότυπο Façade αντιμετωπίζει το πρόβλημα της χρήσης ενός υποσυστήματος από έναν Client. Ένα υποσύστημα είναι ένα σύνολο κλάσεων, οι οποίες παρέχουν λειτουργίες που σε συνδυασμό παράγουν ένα επιθυμητό αποτέλεσμα. Το ζητούμενο που προκύπτει είναι ο τρόπος με τον οποίο θα επικοινωνεί ο Client με το υποσύστημα. Μια πρώτη προσέγγιση φαίνεται στο σχήμα 4.52, η προσέγγιση αυτή υποστηρίζει την απευθείας επικοινωνία του Client με τη κλάση του υποσυστήματος. Μια τέτοια σχεδίαση θα έχει ως αποτέλεσμα την γνώση του Client για τον τρόπο λειτουργίας των κλάσεων του υποσυστήματος. Έτσι λοιπόν στην περίπτωση που ο

Client καλεί πολλές φορές την συγκεκριμένη λειτουργία του υποσυστήματος, θα επιβαρύνεται με την γνώση αυτής της πολύπλοκης διαδικασίας .




Σχήμα 4.52 η Χρήση του Υποσυστήματος χωρίς Façade

Το πρότυπο Façade παρέχει μια διεπαφή υψηλότερου επιπέδου η οποία κάνει πιο απλή την χρήση των κλάσεων του υποσυστήματος από τον Client. Δηλαδή χρησιμοποιώντας το πρότυπο Façade ο Client δεν έχει γνώση του τρόπου λειτουργίας των κλάσεων άλλα όπως δείχνει και το σχήμα 4.54 η πολύπλοκη αυτή διαδικασία επιβαρύνει την κλάση Façade. Με αυτόν τον τρόπο δεν υπάρχουν συσχετίση μεταξύ Client και κλάσεων του συστήματος, ενώ παράλληλα ο Client δεν συντηρεί πολύπλοκες δομές για την γνώση της λειτουργίας που παρέχει το υποσύστημα.




Σχήμα 4.53 η Χρήση του Υποσυστήματος μέσω Façade

Από την εξέταση της χρήση του προτύπου στο σύστημα και αναφορικά με τις μετρικές προκύπτει βελτίωση στο WMC. Πιο συγκεκριμένα όπως δείχνει και ο πίνακας 4.10 η μείωση στο WMC στο σύστημα είναι αποτέλεσμα της μείωσης του WMC στον Client. Στην συγκεκριμένη περίπτωση η μέθοδος του Client, χρησιμοποιεί δυο φορές την λειτουργία του υποσυστήματος. Ως αποτέλεσμα αυτού θα είναι η συντήρηση της πολύπλοκής διαδικασίας δυο φόρες στον κώδικα. Αντίθετα με την χρήση του προτύπου, η κλάση Façade συντηρεί μια φορά αυτήν την διαδικασία. Από την άλλη πλευρά αύξηση παρουσιάζουν οι μετρικές CBO και RFC όπως δείχνει και ο πίνακας 4.10. Η χρήση του προτύπου προσθέτει συσχετίσεις στο σύστημα, ενώ αναφορικά με το RFC η χρήση του προτύπου οδηγεί σε επιπλέον κλήσεις εξωτερικών μεθόδων. Τέλος οι μετρικές LCOM, NOC και DIT δεν παρουσιάζουν καμία μεταβολή αφού δεν εισάγονται νέες μέθοδοι σε υπάρχουσες κλάσεις αλλά ούτε προτείνεται η χρήση κάποιας ιεραρχίας κλάσεων.

```
public class Client {
     int x;
     SubsystemClass1 sub1;
    SubsystemClass2 sub2;
    SubsystemClass3 sub3;

   public void doSomething() {
       if ( x == 1 ){
          sub1.anOperation();
       }
       else if (x==2){
          sub2.anOperation();
       }
       else if (x==3){
          sub3.anOperation();
       }}}
```

Σχήμα 4.54 ο Client χωρίς Χρήση Façade

```
public class Facade{
   private SubsystemClass1 sub1;
   private SubsystemClass2 sub2;
   private SubsystemClass3 sub3;
   private int x;
   public Facade(int x){
   this.x = x;
   }
   public void use_subsystem(){
      if ( x == 1 ){
         sub1.anOperation();
      }
      else if (x==2){
         sub2.anOperation();
      }
      else if (x==3){
         sub3.anOperation();
      }
   }}
```

Σχήμα 4.55 η Κλάση Façade

```
public class Client {
   public void anOperation(){
    Facade facade = new Facade(1);
    facade.use_subsystem();
   }}
```

Σχήμα 4.56 ο Client με Χρήση Facade

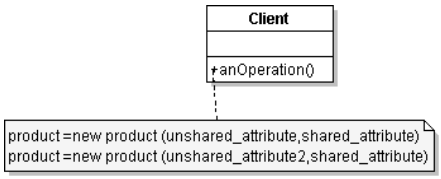
Πίνακας 4.10 οι Μεταβολές των Μετρικών με και χωρίς το Facade

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 3 | 1 | 2 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 0 |
| SubSystem1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SubSystem2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| SubSystem3 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Facade | | 3 | | 0 | | 0 | | 4 | 0 | 0 | 0 | 0 |
| Overall Mean | 0.75 | 1 | 0.5 | 0 | 0 | 0 | 1.75 | 1.8 | 0 | 0 | 0 | 0 |

*4.2.11. Flyweight*

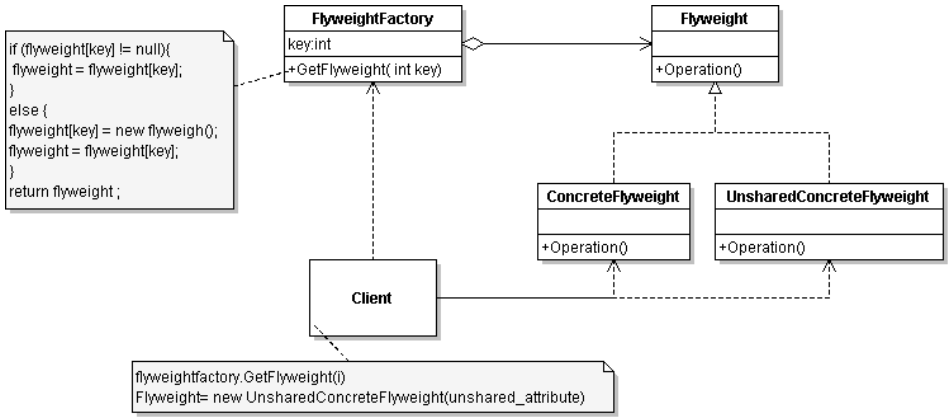
Το πρότυπο Flyweight αντιμετωπίζει προβλήματα που έχουν να κάνουν με την αποθήκευση αντικειμένων στην μνήμη. Πιο συγκεκριμένα σε ένα σύστημα η πληροφορία που αποθηκεύεται στα αντικείμενα μιας κλάσης μπορεί να αποτελεί κοινή πληροφορία για πολλά αντικείμενα της συγκεκριμένης κλάσης ή να αποτελεί μοναδική πληροφορία η οποία διακρίνει το ένα αντικείμενο από τα υπόλοιπα της ίδιας κλάσης. Το ζητούμενο είναι ο τρόπος με τον οποίο θα γίνει η διαχείριση αυτών των αντικειμένων με στόχο την καλύτερη απόδοση του συστήματος. Μια πρώτη προσέγγιση δείχνει το σχήμα 4.57. Σύμφωνα με αυτήν ο Client δημιουργεί αντικείμενα μια κλάσης χωρίς να εξετάζει τα κοινά και τα διαφοροποιούμενα πεδία. Αυτή η προσέγγιση οδηγεί σε σπατάλη μνήμης και ενέχει κινδύνους για το σύστημα. Μάλιστα όσο περισσότερα είναι τα αντικείμενα που δημιουργούνται τόσο μεγαλύτερη η σπατάλη της μνήμη.




Σχήμα 4.57 το Σύστημα χωρίς Χρήση του Προτύπου Flyweight

Το πρότυπο Flyweight εκμεταλλεύεται την κοινή πληροφορία που υπάρχει στα αντικείμενα μιας κλάσης με σκοπό την καλύτερη διαχείριση της μνήμης του συστήματος. Πιο συγκεκριμένα, υποστηρίζει τον διαχωρισμό της κοινής πληροφορίας σε ένα μοναδικό αντικείμενο Flyweight. Στην συνέχεια ακολουθεί ο συσχετισμός του Flyweight με το υποσύνολο των αντικειμένων της κλάσης που το χρησιμοποιούν. Με αυτόν τον τρόπο ένα συγκεκριμένο Flyweight χρησιμοποιείται από ένα υποσύνολο αντικειμένων. Όπως δείχνει και το σχήμα 4.58 το πρότυπο υποστηρίζει και την χρήση

μια κλάσης FlyweightFactory, η οποία είναι υπεύθυνη για την διαχείρισης των αντικειμένων Flyweight ενώ στο σχήμα 4.60 φαίνεται μια υλοποίηση της κλάσης FlyweightFactory.




Σχήμα 4.58 το Σύστημα με Χρήση Flyweight

Εξετάζοντας την εφαρμογή του προτύπου αναφορικά με τις μετρικές γίνεται φανερό ότι δεν υπάρχει κάποια βελτίωση στο σύστημα. Πιο συγκεκριμένα όπως δείχνει και ο πινάκας 4.11 υπάρχει μια επιβάρυνση σε όλες τις μετρικές του συστήματος εκτός βέβαια από την μετρική LCOM. Έτσι λοιπόν η χρήση του πρότυπου επιβαρύνει την σχεδίαση του συστήματος με επιπλέον πολυπλοκότητα. Βέβαια η χρήση του προτύπου ωφελεί το σύστημα αφού από τον ορισμό του το πρότυπο Flyweight δίνει λύση σε θέματα που αφορούν την καλύτερη διαχείριση τον πόρων του συστήματος. Δηλαδή μειώνει την χρήση της μνήμης η οποία επιτυγχάνεται μέσω μίας εναλλακτικής σχεδίασης του συστήματός.

Συνοψίζοντας, το πρότυπο Flyweight μειώνει την χρήση της μνήμης, το οποίο όμως δεν φαίνεται από κάποια μετρική, ενώ παράλληλα επιβαρύνει την πολυπλοκότητα του συστήματος το οποίο μπορεί να γίνει αντιληπτό και μέσω των μετρικών του συστήματος.

```
public interface Flyweight {
   public void report();
}
class ConcreteFlyweight implements Flyweight {
```

```
    private int i;
    private String atr1;
    private String atr2;
    private int atr3;
    private String atr4;
    private String atr5;
   public ConcreteFlyweight( int i,String atr1, String atr2) {
        this.i=i;
        this.atr1 = atr1;
        this.atr2 = atr2;
   }
   public void report() {
     System.out.print( " " + atr1+atr2);
}  }

public class UnsharedConcreteFlyweight implements Flyweight {
    private String Unshared_atr;

    public UnsharedConcreteFlyweight(String Unshared_atr){
    this.Unshared_atr=Unshared_atr;
    }
    public void report(){
    System.out.print(Unshared_atr);
    }}
```

Σχήμα 4.59 η Ιεραρχία του Προτύπου Flyweight

```
class FlyweightFactory {
   private Flyweight[] pool;
   public FlyweightFactory() {
    pool = new Flyweight[2];
   }
   public void createFlyweight (int theRow,String atr1, String atr2){
```

```
      pool[theRow] = new ConcreteFlyweight( theRow, atr1, atr2);


  }
  public Flyweight getFlyweight( int theRow,String atr1, String atr2) {
    if (pool[theRow] == null)
      pool[theRow] = new ConcreteFlyweight( theRow, atr1, atr2);
    return pool[theRow];
} }
```

Σχήμα 4.60 η Κλάση FlyweightFactory

```
public class Client {
    FlyweightFactory flyweightfactory;
    UnsharedConcreteFlyweight un;


  anOperation(){
    flyweightfactory = new FlyweightFactory();
    flyweightfactory.createFlyweight(0, "atr1", "atr2");
    un = new UnsharedConcreteFlyweight("name1");
    un.report();
    flyweightfactory.getFlyweight(0, "department1", "address1").report();
  }}
```

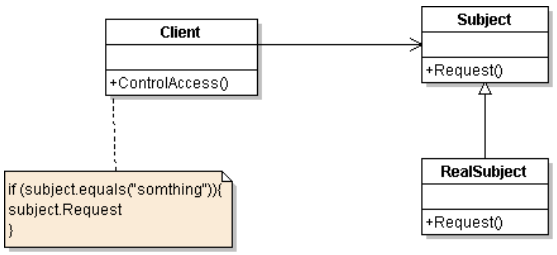
Σχήμα 4.61 ο Client με Χρήση Flyweight

Πίνακας 4.11 οι Μεταβολές των Μετρικών με και χωρίς το Flyweight

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 1 | 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Flyweight | | 0 | | 0 | 0 | 0 | | 0 | | 2 | | 0 |
| ConcreteFlyweight | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 1 |
| UnsharedConcreteFlyweight | | 0 | | 0 | | 0 | | 1 | | 0 | | 1 |
| FlyweightFacrory | | 0 | | 2 | 0 | 0 | | 1 | | 0 | | 0 |
| Overall Mean | 0.5 | 0.8 | 0 | 2/5 | 0 | 0 | 1 | 4/5 | 0 | 2/5 | 0 | 2/5 |

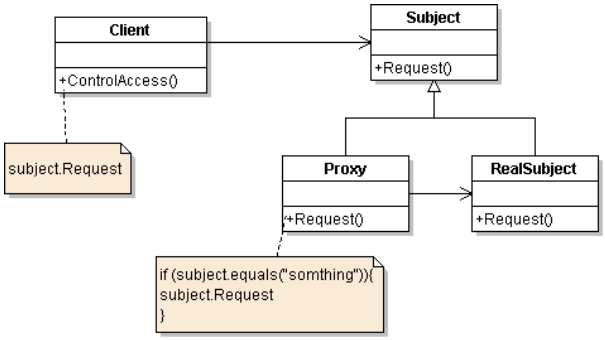
*4.2.12. Proxy*

Το πρότυπο Proxy αντιμετωπίζει προβλήματα ελέγχου προσπέλασης που είναι πιθανόν να προκύψουν σε συστήματα. Πιο συγκεκριμένα υπάρχουν περιπτώσεις στις οποίες ο Client του συστήματος μπορεί να χρησιμοποιήσει κάποιο αντικείμενο μόνο υπό συνθήκες. Έτσι λοιπόν προκύπτει ένα ερώτημα για το ποια κλάση θα πραγματοποιεί τους ελέγχους αυτούς. Μια πρώτη προσέγγιση είναι ο ίδιος ο Client να έχει την ευθύνη για τον έλεγχο της συνθήκης. Όπως δείχνει το σχήμα 4.62 ο ίδιος ο Client έχει δομή επιλογής προκειμένου να ελέγξει την συνθήκη. Όμως στην πραγματικότητα μια τέτοια δομή επιλογής πρέπει να υπάρχει σε οποιοδήποτε σημείο που ο Client καλεί το αντικείμενο. Συνεπώς, η συγκεκριμένη προσέγγιση επηρεάζει την πολυπλοκότητα του Client.




Σχήμα 4.62 το Σύστημα χωρίς Proxy

Το πρότυπο Proxy παρουσιάζει μια λύση σύμφωνα με την οποία ο έλεγχος της συνθήκης είναι ανεξάρτητος από των κώδικα του Client. Το πρότυπο επεκτείνει την ιεραρχία με μια νέα υλοποίηση proxy, ενώ ο Client του συστήματος αντί να συσχετίζεται με αντικείμενα τύπου RealSubject συσχετίζεται με αντικείμενα τύπου proxy. Όπως δείχνει και το σχήμα 4.65 η κλήση μεθόδου του RealSubject μπορεί να πραγματοποιηθεί μέσω του proxy εφόσον ισχύει η συνθήκη. Το σχήμα 4.63 δίνει την συνολική εικόνα του συστήματος με την χρήση του προτύπου.




Σχήμα 4.63 το Σύστημα με Χρήση Proxy

Εξετάζοντας την χρήση του προτύπου σε σχέση με της μετρικές, βελτίωση παρουσιάζει το WMC. Όπως δείχνει και ο πίνακάς 4.12 εξαγωγής του έλεγχου της συνθήκης από τον Client επιφέρει την βελτίωση του συστήματός καθώς υπάρχει μείωση του WMC ανά κλάση. Αντίθετα αύξηση παρουσιάζει το CBO και αυτό γιατί ο Proxy συντηρεί μια αναφορά στο RealSubject. Επιπλέον η χρήση του Proxy επιβαρύνει και το RFC του συστήματος καθώς η νέα υλοποίηση καλεί μια εξωτερική μέθοδο που συνεπώς προκαλεί αύξηση του RFC ανά κλάση για όλο το σύστημα. Ακόμα αύξηση παρουσιάζουν οι μετρικές DIT και NOC. Ο λόγος είναι ότι η κλάση Proxy υλοποιεί την διεπαφή του συστήματος με συνέπεια η κλάση αυτή να έχει DIT ίσο με ένα ενώ ταυτόχρονα το στοιχείο αφαίρεσης Subject παρουσιάζει αύξηση της τιμής του NOC. Η αύξηση που ωφελείται στην κλάση Proxy οδηγούν στην επιβάρυνση του μέσου NOC και DIT του συστήματος. Τέλος το LCOM παραμένει αμετάβλητο καθώς δεν υπάρχει κάποια αλλαγή σε μεθόδους.

```
public interface Subject{
public void Request()
};

public class RealSubject implements Subject {
   public void Request(){
      System.out.println("Request");
   }}
```

Σχήμα 4.64 το Στοιχείο Αφαίρεσης Subject και η Υλοποίηση RealSubject

```
public class Proxy implements Subject{
   public void Request(){
   int x=0;
   System.out.println("check");
   if (x==0){
    Subject subject = new RealSubject();
    subject.Request();
   }}}
```

Σχήμα 4.65 η Κλάση Proxy

```
public class Client {
        Subject subject = new  RealSubject();
      if (subject instanceof RealSubject){
       subject.Request();
      }}
```

Σχήμα 4.66 ο Client χωρίς Χρήση Proxy

```
public class client {
      Subject s = new proxy();
      s.Request();
   }
```

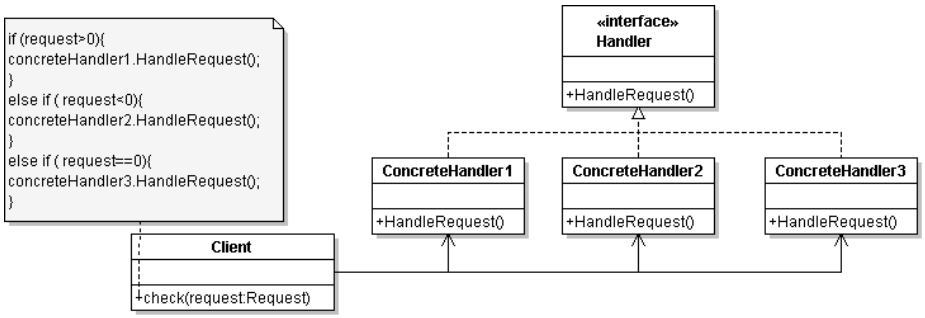
Σχήμα 4.67 ο Client με την Χρήση του Proxy

Πίνακας 4.12 οι Μεταβολές των Μετρικών με και χωρίς το Proxy

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 1 | 1 | 2 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 |
| Subject | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 |
| RealSubject | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Proxy | | 1 | | 1 | | 0 | | 2 | | 0 | | 1 |
| Overall Mean | 1/3 | 2/4 | 0.66 | 0.25 | 0 | 0 | 1 | 5/4 | 1/3 | 0.5 | 1/3 | 0.5 |

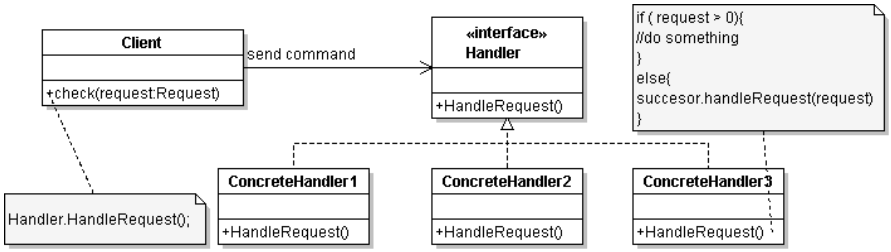
*4.2.13. Chain of Responsibility*

Το πρότυπο Chain of Responsibility προσφέρει μια λύση σε συστήματα στα οποία ο Client στέλνει μηνύματα σε αντικείμενα κάποιων άλλων κλάσεων προκειμένου να χρησιμοποιηθούν από αυτά. Τα μηνύματα αυτά μπορεί να έχουν την μορφή αντικειμένων κάποιας άλλης κλάσης ενώ για να τα χρησιμοποιήσει κάποιο αντικείμενο χειριστής, θα πρέπει να ισχύει κάποια συνθήκη. Σε αντίθετη περίπτωση το μήνυμα πρέπει να προωθείται στο επόμενο αντικείμενο χειριστή. Ζητούμενο είναι ο τρόπος με τον οποίο το σύστημα θα υλοποιεί των έλεγχο και την αποστολή του μηνύματος στο κατάλληλο αντικείμενο χειριστή. Μια πρώτη προσέγγιση φαίνεται στο σχήμα 4.68. Σύμφωνα με αυτήν ο Client του συστήματος έχει την ευθύνη για τον έλεγχο του μηνύματος. Έτσι λοιπόν ο έλεγχος στον Client καθορίζει σε ποιο αντικείμενο χειριστή θα σταλεί το μήνυμα.




Σχήμα 4.68 το Σύστημα χωρίς Χρήση CoR

Το πρότυπο δίνει μια λύση βάσει της οποίας τα αντικείμενα ConcreteHandler λειτουργούν ως αλυσίδα μέσω της οποίας το μήνυμα μεταδίδεται σε όλα τα αντικείμενα. Κάθε αντικείμενο ConcreteHandler παίρνει το μήνυμα και έχει την δυνατότητα να το περάσει στο επόμενο αντικείμενο της αλυσίδας. Όπως δείχνει και το σχήμα 4.69 με αυτόν τον τρόπο το μήνυμα χρειάζεται να μεταδοθεί στο πρώτο αντικείμενο της αλυσίδας ενώ κάθε αντικείμενο γνωρίζει σε ποιο αντικείμενο πρέπει να στείλει το μήνυμα. Έτσι λοιπόν δεν υπάρχει εξάρτηση του Client από της υλοποιήσεις της ιεραρχίας. Ενώ παράλληλα υπάρχει και μείωση της πολυπλοκότητας.




Σχήμα 4.69 το Σύστημα με Χρήση CoR

Εξετάζοντας το σύστημα αναφορικά με τις μετρικές, σημαντική βελτίωση παρουσιάζει το CBO. Όπως φαίνεται και στο σχήμα 4.73 η διαδικασία αποστολής του μηνύματος δεν επιβαρύνει τον Client με εξαρτήσεις από τις υλοποιήσεις. Έτσι λοιπόν έχοντας ως δεδομένο ότι η αλυσίδα των αντικειμένων χειριστών έχει δημιουργηθεί με κάποιον τρόπο, το πρότυπο μειώνει το CBO του Client. Όπως δείχνει ο πίνακας 4.13 η μείωση στο CBO του Client οδηγεί στην βελτίωση του CBO ανά κλάση στο σύστημα. Αντίθετα το RFC επιβαρύνεται καθώς με την χρήση του προτύπου, κάθε υλοποίηση καλεί επιπλέον μια εξωτερική μέθοδο. Οι υπόλοιπες μετρικές δεν παρουσιάζουν κάποια μεταβολή. Όπως δείχνει ο πίνακας 4.13 το WMC στο Client μειώνεται και η χρήση του προτύπου οδηγεί σε μια ανάλογη αύξηση στις υλοποιήσεις της ιεραρχίας. Συνεπώς το μέσο WMC ανά κλάση δεν μεταβάλλεται. Ακόμα το LCOM παραμένει σταθερό και αυτό γιατί η χρήση του προτύπου δεν οδηγεί στην μεταβολή του αριθμού των μεθόδων σε κάποια κλάση. Τέλος οι μετρικές NOC και DIT παραμένουν σταθερές αφού το πρότυπο δεν απαιτεί κάποια αλλαγή αναφορικά με τις υλοποιήσεις της ιεραρχίας.

```
public abstract class Handler{
  public abstract void handleRequest();
}

public class ConcreteHandler1 extends Handler{
  public void handleRequest(){
    System.out.println("Negative values are handled by ConcreteHandlerOne:");
  }}
```

```
public class ConcreteHandler2 extends Handler{
  public void handleRequest(){
    System.out.println("Positive values are handled by ConcreteHandlerTwo:");
  }}

public class ConcreteHandler3 extends Handler{
   public void handleRequest(){
       System.out.println("Zero values are handled by ConcreteHandlerThree:");
 }}
```

Σχήμα 4.70 η Ιεραρχία του Συστήματος χωρίς CoR

```
public abstract class Handler{
          protected Handler m_successor;
          public void setSuccessor(Handler successor ){
            m_successor = successor;
          }
          public abstract void HandleRequest(Request request);
}

public class ConcreteHandler1 extends Handler{
     public void HandleRequest(Request request){
        if (request.getValue() < 0){
           System.out.println("Negative values are handled by ConcreteHandlerOne:");
        }
         else
           m_successor.HandleRequest(request);
        }}

public class ConcreteHandler2 extends Handler{
          public void HandleRequest(Request request){
             if (request.getValue() > 0){
             System.out.println("Positive values are handled by ConcreteHandlerTwo:");
```

```
            }
            else
            m_successor.HandleRequest(request);
            }}

public class ConcreteHandler3 extends Handler{
  public void HandleRequest(Request request){
      if (request.getValue() >= 0){
         System.out.println("Zero values are handled by ConcreteHandlerThree:");
      }
      else
         m_successor.HandleRequest(request);
      }}
```

Σχήμα 4.71 η Ιεραρχία του Συστήματος με CoR

```
public class Main {
    public static void main(String[] args) {
       Handler handler;
       Request request = new Request("111",0);
       if (request.getValue()<0){
        handler = new ConcreteHandler1();
        handler.handleRequest();
       }
       else if (request.getValue()>0){
        handler = new ConcreteHandler2();
        handler.handleRequest();
       }
        else if(request.getValue()==0){
        handler = new ConcreteHandler3();
        handler.handleRequest();
       }}}
```

Σχήμα 4.72 ο Client του Συστήματος χωρίς Χρήση CoR

```
public class Client {
 public static void main(String[] args) {
    /*Setup Chain of Responsibility
    Handler h1 = new ConcreteHandler1();
    Handler h2 = new ConcreteHandler2();
    Handler h3 = new ConcreteHandler3();
    h1.setSuccessor(h2);
    h2.setSuccessor(h3);
    */
    // Send requests to the chain
     Request request = new Request("Value", -1);
     h1.HandleRequest(request);
     }}
```

Σχήμα 4.73 ο Client του Συστήματος με Χρήση του Προτύπου

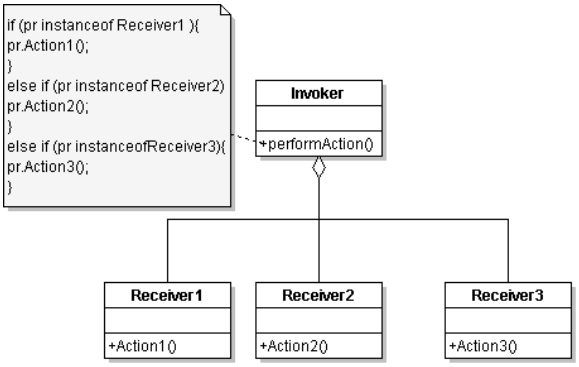
Πίνακας 4.13 οι Μεταβολές των Μετρικών με και Χωρίς το CoR

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 3 | 0 | 3 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 0 |
| Handler | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 0 | 0 |
| ConcreteHandler1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 1 |
| ConcreteHandler2 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 1 |
| ConcreteHandler3 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 1 |
| Overall Mean | 3/5 | 0 | 3/5 | 3/5 | 0 | 0 | 7/5 | 8/5 | 3/5 | 3/5 | 3/5 | 3/5 |

### *4.2.14. Command*

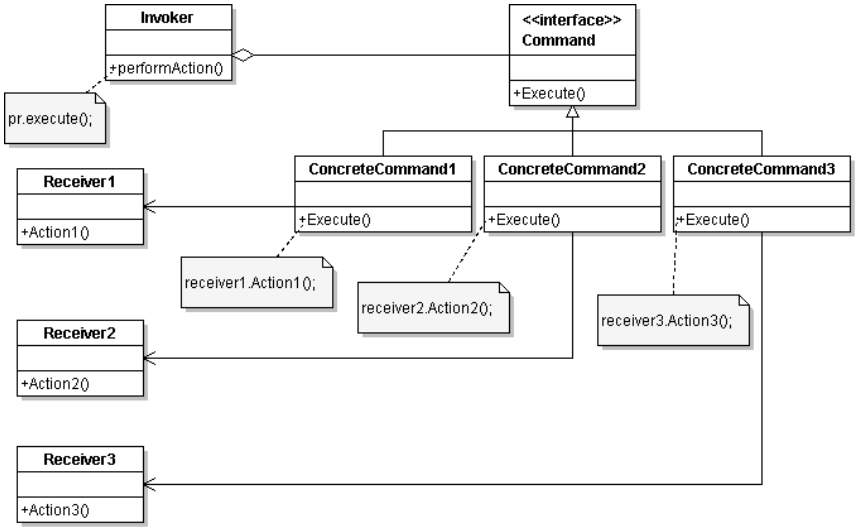
Το πρότυπο Command δίνει λύσεις σε περίπτωση που στο σύστημα που θα υλοποιηθεί χρησιμοποιεί αντικείμενα κάποιων κλάσεων προκειμένου να εκτελεστούν κάποιες λειτουργίες τους. Δηλαδή ανάλογα με τον τύπο του αντικειμένου που καλείται, εκτελούνται και διαφορετικές λειτουργίες του συστήματος. Το ζητούμενο είναι ο τρόπος με τον οποίο η κλάση, η οποία είναι επιφορτισμένη με την εκτέλεση τον λειτουργιών του συστήματος, θα πραγματοποιεί αυτή την διαδικασία. Μια πρώτη προσέγγιση φαίνεται στο σχήμα 4.74, η κλάση Invoker έχει γνώση για την εκτέλεση

αυτών των λειτουργιών και ανάλογα με το αντικείμενο που δέχεται σαν όρισμα εκτελεί την αντίστοιχη λειτουργία.




Σχήμα 4.74 το Σύστημα χωρίς Χρήση Command

Το πρότυπο δίνει μια λύση βάσει της οποίας η κλάση Invoker μένει ανεξάρτητη από τις κλάσεις Receiver του συστήματος. Όπως φαίνεται στο σχήμα 4.75 στο στοιχείο αφαίρεσης Command υλοποιείται διαφορετικά για κάθε Receiver. Κάθε υλοποίηση Execute καλεί την αντίστοιχη μέθοδο Action του Receiver. Με αυτόν τον τρόπο όπως φαίνεται και στο σχήμα 4.79 ο Invoker αντιμετωπίζει με έναν ενιαίο τρόπο την εκτέλεση των λειτουργιών του κάθε Receiver. Δηλαδή η κλάση Invoker ενθυλακώνει ένα αντικείμενο Command και εκτελεί την μέθοδο Execute.




Σχήμα 4.75 το Σύστημα με Χρήση Command

Εξετάζοντας το σύστημα βάσει των μετρικών, βελτίωση παρουσιάζουν οι μετρικές CBO, WMC και RFC. Χρησιμοποιώντας το πρότυπο η κλάση Invoker δεν έχει γνώση για τα αντικείμενα Receiver συνεπώς δεν υπάρχει καμία συσχέτιση. Σύμφωνα με τον πίνακα 4.14 η μείωση το CBO του Invoker οδηγεί στην μείωση του μέσου CBO στο σύστημα. Ανάλογα και για το WMC, όπως φαίνεται στο σχήμα 4.79, η κλάση Invoker δεν ελέγχει την διαδικασία εκτέλεσης των λειτουργιών, αλλά ο έλεγχος αυτός πραγματοποιείται κατανεμημένα από την ιεραρχία που εισάγει το πρότυπο. Συνεπώς υπάρχει μείωση της πολυπλοκότητας του Invoker που οδηγεί στην μείωση του μέσου WMC στο σύστημα. Ακόμα το RFC παρουσιάζει βελτίωση επειδή η κλάση Invoker χρησιμοποιεί έναν όμοιο τρόπο για την εκτέλεση των εξωτερικών λειτουργιών. Δηλαδή η κλάση Invoker δεν καλεί διαφορετικές εξωτερικές μεθόδους για τον κάθε Receiver αλλά καλεί μια μέθοδο που του παρέχει η ιεραρχία του προτύπου. Ενώ το LCOM δεν παρουσιάζει καμία μεταβολή αφού δεν υπάρχει μεταβολή στην συνεκτικότητα των κλάσεων. Τέλος αύξηση παρουσιάζει το NOC και το DIT. Η ιεραρχία που χρησιμοποιεί το πρότυπο έχει ως αποτέλεσμα το σύστημα να έχει ένα στοιχείο αφαίρεσης με NOC ίσο με τρία και τρείς υλοποιήσεις με DIT ίσο με ένα. Έτσι υπάρχει αύξηση του μέσου DIT και NOC σε όλο το σύστημα.

```
public class Receiver1 {
    public void Action1(){
       System.out.println(" Action1 is excuted");
   }}

public class Receiver2 {
   public void Action2(){
       System.out.println(" Action2 is excuted");

   }}

public class Receiver3 {
   public void Action3(){
       System.out.println(" Action3 is excuted");
```

```
}}
```

Σχήμα 4.76 οι Κλάσεις Receiver

```
public class Invoker {
    private Object pr;
    Invoker(Object pr){
    this.pr=pr;
    }
    public void performAction(){
        if (pr instanceof Receiver1){
            Receiver1 pr1 = (Receiver1)pr;
            pr1.Action1();
        }
        else if (pr instanceof Receiver2){
            Receiver2 pr2 = (Receiver2)pr;
            pr2.Action2();
        }
        else if (pr instanceof Receiver3){
            Receiver3 pr3 = (Receiver3)pr;
            pr3.Action3();
        }}}
```

Σχήμα 4.77 η Κλάση Invoker χωρίς Χρήση Command

```
public interface Command {
 public void Execute ();
}
class ConcreteCommand1 implements Command {
    private Receiver1 r;
    public ConcreteCommand1 (Receiver1 r) {
        this.r = r;
    }
```

```
    public void execute( ) {
        r.Action1();
    }}

class ConcreteCommand2 implements Command {
    private Receiver2 r;
    public ConcreteCommand2 (Receiver2 r) {
    this.r = r;
    }
    public void execute( ) {
        r.Action2();
    }}

public class ConcreteCommand3 {
    private Receiver3 r;
    public ConcreteCommand3 (Receiver3 r) {
        this.r = r;
    }
    public void execute( ) {
        r.Action3();
    }}
```

Σχήμα 4.78 η Ιεραρχία Command

```
class Invoker {
    private Command com;
    public Invoker(Command com) {
        this.com = com;
    }
    void performAction() {
        this.com.execute();
    }}
```

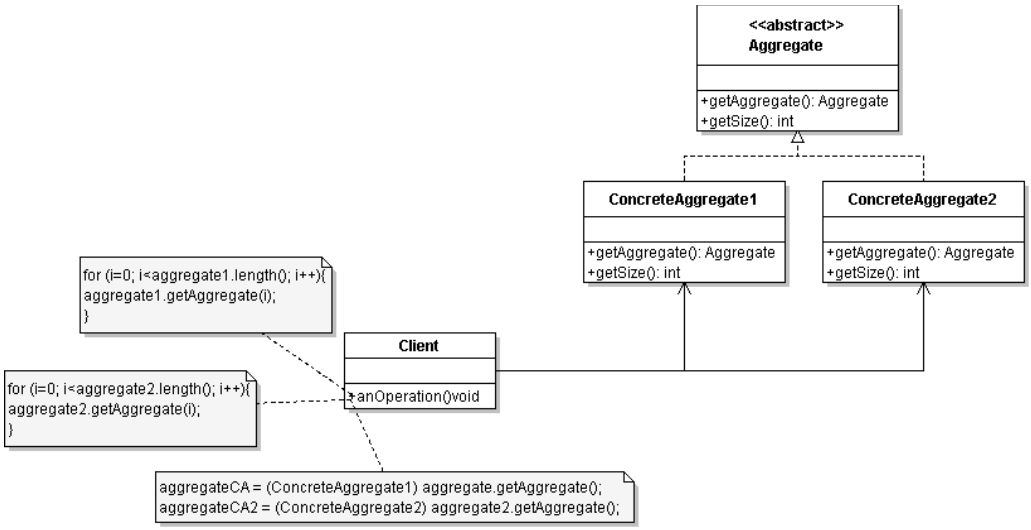
Σχήμα 4.79 η Κλάση Invoker με Χρήση Command

Πίνακας 4.14 οι Μεταβολές των Μετρικών με και χωρίς το Command

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Involker | 3 | 0 | 3 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 0 |
| Receiver1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Receiver2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Receiver3 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Command | | 0 | | 0 | | 0 | | 0 | | 3 | | 0 |
| Concrete Command1 | | 1 | | 0 | | 0 | | 2 | | 0 | | 1 |
| Concrete Command2 | | 1 | | 0 | | 0 | | 2 | | 0 | | 1 |
| Concrete Command3 | | 1 | | 0 | | 0 | | 2 | | 0 | | 1 |
| Overall Mean | 0.75 | 0.37 | 0.75 | 0 | 0 | 0 | 1.75 | 1.375 | 0 | 0.37 | 0 | 0.37 |

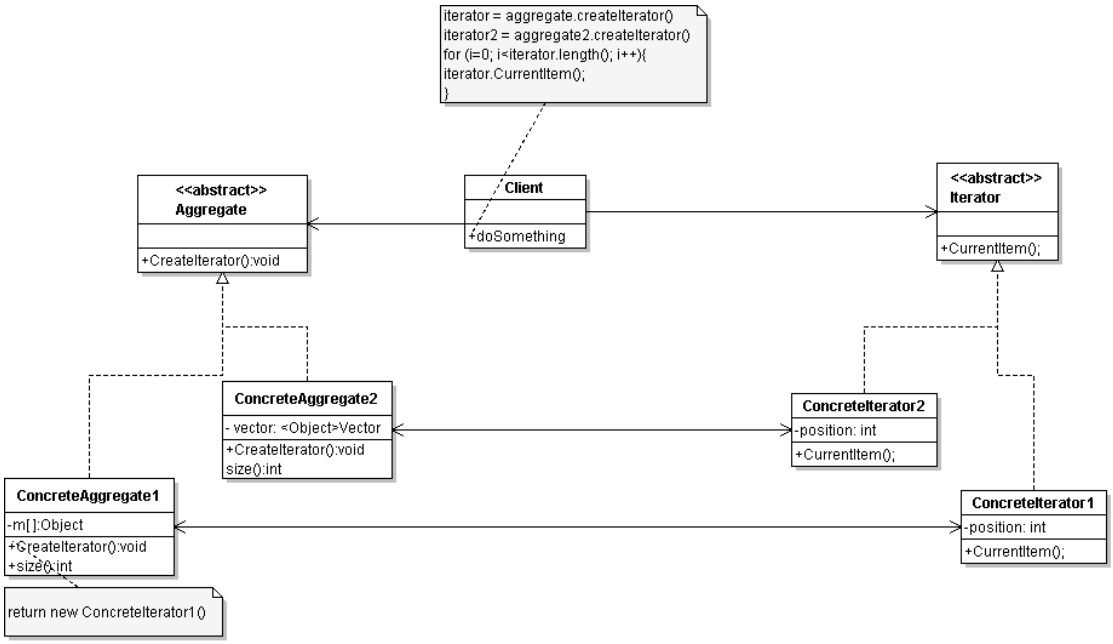
*4.2.15. Iterator*

Το πρότυπο Iterator δίνει λύση σε συστήματα τα οποία χρησιμοποιούν σύνθετες δομές δεδομένων. Πιο συγκεκριμένα, τα συστήματα αυτά χρησιμοποιούν δομές δεδομένων όπως είναι οι λίστες και οι πίνακες, οι οποίες περιέχουν ένα σύνολο από αντικείμενα. Το ζητούμενο είναι πως ο Client του συστήματος θα έχει πρόσβαση στα αντικείμενα που υπάρχουν σε αυτές τις δομές. Μια αρχική προσέγγιση φαίνεται στο σχήμα 4.81. Συμφώνα με αυτή ο Client έχει γνώση της διαδικασίας που πρέπει να ακολουθηθεί για την προσπέλαση της κάθε σύνθετης, δηλαδή έχει μηχανισμούς που υποστηρίζουν την πρόσβαση στα αντικείμενα των διαφορετικών δομών δεδομένων. Αυτή η γνώση του Client για την διαχείριση των δομών δεδομένων έχει ως αποτέλεσμα αυξημένη πολυπλοκότητα.




Σχήμα 4.80 το Σύστημα χωρίς Χρήση Iterator

Η λύση που παρέχει το πρότυπο φαίνεται στο σχήμα 4.81. Συμφώνα με αυτήν η ευθύνη για την διαχείριση των αντικειμένων μιας σύνθετης δομής δεδομένων περνάει σε μία υλοποίηση ConcreteIterator. Επιπλέον κάθε τέτοια υλοποίηση συσχετίζεται με ένα ConcreteAggregate ενώ σκοπός της υλοποίησης αυτής είναι η παροχή λειτουργιών για την πρόσβαση στα αντικείμενα της κάθε δομής. Πιο συγκεκριμένα, το πρότυπο εισάγει μια δεύτερη ιεραρχία αποτελούμενη από ένα στοιχείο αφαίρεσης Iterator και ένα σύνολο υλοποιήσεων ConcreteIterator. Το στοιχείο αφαίρεσης ορίζει τις λειτουργίες χειρισμού των δομών αυτών, ενώ οι ConcreteIterator υλοποιούνται έχοντας γνώση για τις ιδιαιτερότητες για κάθε δομή δεδομένων. Με αυτόν τον τρόπο ο Client έχει επιπλέον λειτουργίες για την προσπέλαση της κάθε δομής δεδομένων.




Σχήμα 4.81 το Σύστημα με Χρήση Iterator

Μελετώντας το σύστημα αναφορικά με τις μετρικές βελτίωση παρουσιάζουν οι μετρικές CBO, WMC και RFC. Όπως φαίνεται στον πίνακα 4.15, υπάρχει μείωση στο CBO του Client αφού πλέον δε συσχετίζεται με ConcreteAggregate ενώ η κάθε υλοποίηση του Iterator συσχετίζεται με ένα ConcreteAggregate. Επιπλέον όπως δείχνει και στο σχήμα 4.82 κάθε ConcreteIterator χαρακτηρίζεται ως private ενώ υλοποιείται εσωτερικά στην αντίστοιχη κλάση ConcreteAggregate. Αναφορικά με το

WMC, η βελτίωση οφείλεται στην μειωμένη πολυπλοκότητα του Client. Όπως δείχνει το σχήμα 4.85, ο Client συντηρεί έναν μηχανισμό πρόσβασης σε δομή δεδομένων, και βάσει αυτού του μοναδικού μηχανισμού μπορεί να διαχειρίζεται διαφορετικούς τύπους δομών. Ακόμα το RFC παρουσιάζει βελτίωση γιατί η χρήση του προτύπου οδηγεί στη κλήση λιγότερων εξωτερικών μεθόδων από μέρος του Client. Συνεπώς το μέσο RFC του συστήματος μειώνεται. Σχετικά με το LCOM δεν υπάρχει καμία διαφοροποίηση αφού οι νέες κλάσεις του συστήματος δεν παρουσιάζουν έλλειψη συνεκτικότητας. Τέλος αύξηση παρουσιάζουν οι μετρικές DIT και NOC. Η χρήση του προτύπου οδηγεί στην εισαγωγή μιας νέας ιεραρχίας, συνεπώς το μέσο DIT και NOC του συστήματος θα αυξηθεί.

```
interface Aggregate{
   public Iterator createIterator();
   public int size();
}

public class ConcreteAggregate implements Aggregate {
 public String m_titles[] = {"Design Patterns","1","2","3","4"};
    public Object getAggregate(){
    return m_titles;
    }
    public int getSize(){
        return m_titles.length;
    }}

public class ConcreteAggregate2 implements Aggregate{
    private Vector <String> vector = new Vector <String>();

    ConcreteAggregate2(){
        vector.add("Design Patterns");
        vector.add("1");
        vector.add("2");
```

```
        vector.add("3");
        vector.add("4");
    }
    public Object  getAggregate(){
    return vector;
    }
    public int getSize(){
        return vector.size();
    }}
```

Σχήμα 4.82 η Ιεραρχία του Συστήματος

```
interface Iterator{
    public Object CurrentItem();
}

private class ConcreteIterator implements Iterator{
   private int m_position;
public Object CurrentItem(){
   return m_titles[m_position];
   m_position++;
}}

private class ConcreteIterator2 implements Iterator{
private int m_position;

public Object CurrentItem(){
return vector.get(m_position);
m_position++;
}
```

Σχήμα 4.83 η Ιεραρχία του Προτύπου Iterator

```
public class Client {
public void doSomething() {

        Aggregate aggregate = new ConcreteAggregate();
        String b[] = (String[]) aggregate.getAggregate();
        //accessForConcreteAggregate1
         for (int i = 0; i < b.length; i++ ){
         System.out.println(b[i]);
          }

        Aggregate aggregate2 = new ConcreteAggregate2();
        Vector b2 = (Vector) aggregate2.getAggregate();
        //accessForConcreteAggregate2
         for (int j = 0; j<b2.size(); j++){
          System.out.println(b2.get(i));
         }}}
```

Σχήμα 4.84 ο Client χωρίς την Χρήση Iterator

```
public class Client {
public void doSomething (Aggregate aggregate){
        Iterator iterator = aggregate.createIterator();
       for (int i=0 ; i<aggregate.size(); i++){
       iterator.CurrentItem();
      }} }
```

Σχήμα 4.85 ο Client με την Χρήση Iterator

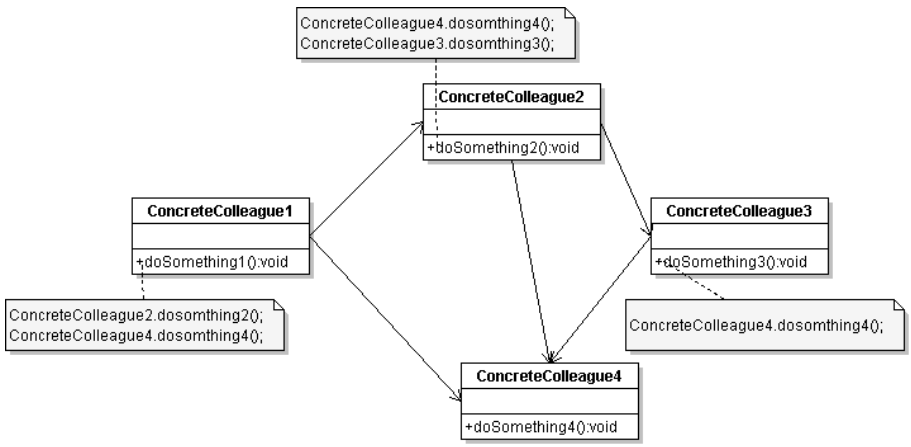
Πίνακας 4.15 οι Μεταβολές των Μετρικών με και χωρίς το Iterator

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 2 | 0 | 2 | 1 | 0 | 0 | 7 | 4 | 0 | 0 | 0 | 0 |
| Aggregate | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| Concrete Aggregate1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 1 | 1 |
| Concrete Aggregate 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 1 | 1 |
| Iterator | | 0 | | 0 | | 0 | | 0 | | 2 | | 0 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Concrete Iterator1 | | 1 | | 0 | | 0 | | 1 | | 0 | | 1 |
| Concrete Iterator2 | | 1 | | 0 | | 0 | | 1 | | 0 | | 1 |
| Overall Mean | 0.5 | 0.28 | 0.5 | 0.14 | 0 | 0 | 2.75 | 1.4 | 0.5 | 0.57 | 0.5 | 0.57 |

*4.2.16. Mediator*

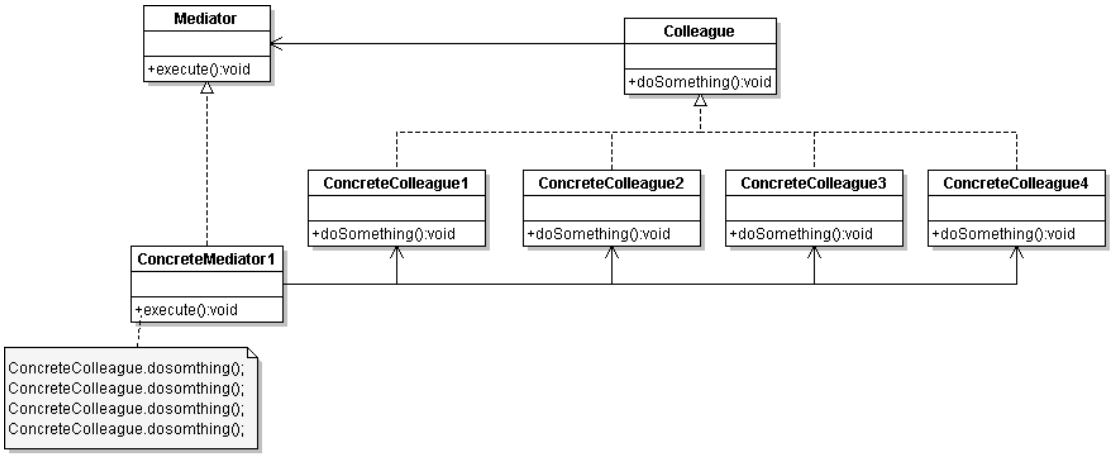
Το πρότυπο Mediator χρησιμοποιείται στην περίπτωση που έχουμε ένα σύστημα και η εκτέλεση των λειτουργιών του απαιτεί των συντονισμό αντικειμένων διαφορετικών κλάσεων. Ζητούμενο είναι ένας τρόπος συσχέτισης των αντικειμένων αυτών των κλάσεων με σκοπό την πραγματοποίηση των λειτουργιών του συστήματος. Δεδομένου λοιπόν ότι η μια λειτουργία του συστήματος είναι κατανεμημένη στις διαφορετικές κλάσεις ConcreteColleague, είναι απαραίτητος για το σύστημα ένας τρόπος με τον οποίο θα γίνεται ο έλεγχος και ο συντονισμός τους με στόχο την επιτυχή εκτέλεση των λειτουργιών του συστήματος. Μια πρώτη προσέγγιση φαίνεται στο σχήμα 4.86. Σύμφωνα με αυτή υπάρχει απευθείας συσχέτιση μεταξύ των κλάσεων του συστήματος δηλαδή κάθε κλάση έχει αναφορές προς τα αντικείμενα που συνεργάζονται προκειμένου να πραγματοποιηθεί η λειτουργία. Η συγκεκριμένη προσέγγιση οδηγεί σε αυξημένη σύζευξη μεταξύ των κλάσεων.




Σχήμα 4.86 το Σύστημα χωρίς Χρήση Mediator

Το πρότυπο δίνει μια λύση στο πρόβλημα βάσει της οποίας η εύθυνη για τον έλεγχο και των συντονισμό των αντικειμένων, προκειμένου να εκτελεσθεί η λειτουργία, περνάει σε μια νέα ιεραρχία κλάσεων. Όπως φαίνεται στο σχήμα 4.87 η ιεραρχία αυτή αποτελείται από ένα στοιχείο αφαίρεσης το οποίο ορίζει τις μεθόδους που είναι

απαραίτητες για την επικοινωνία και των συντονισμό των ConcreteColleague και ένα σύνολο υλοποιήσεων οι οποίες υλοποιούν το στοιχείο αφαίρεσης. Έτσι λοιπόν η διαδικασία συντονισμού των κλάσεων από κατανεμημένη, που είναι πριν την χρήση του προτύπου, γίνεται εντατικοποιημένη σε μια υλοποίηση του Mediator.




Σχήμα 4.87 το Σύστημα με Χρήση Mediator

Εξετάζοντας το σύστημα αναφορικά με τις μετρικές, βελτίωση παρουσιάζουν οι μετρικές CBO και RFC. Σύμφωνα με τον πίνακα το CBO των κλάσεων του συστήματος μηδενίζεται. Αντίθετα το CBO της υλοποίησης του Mediator παρουσιάζει τιμή ίση με τον αριθμό των κλάσεων του συστήματος. Επιπλέον μείωση παρουσιάζει και το RFC. Με την χρήση του προτύπου οι κλάσεις ConcreteColleague δεν καλούν εξωτερικές μεθόδους συνεπώς υπάρχει μείωση RFC στις κλάσεις του συστήματος, ενώ το RFC του ConcreteMediator επιβαρύνεται από την κλήση εξωτερικών μεθόδων, ο αριθμός των οποίων είναι ίσος με το πλήθος των κλάσεων. Οι μετρικές WMC και LCOM δεν παρουσιάζουν καμία μεταβολή αφού το πρότυπο δεν επηρεάζει την πολυπλοκότητα αλλά ούτε την συνεκτικότητα των κλάσεων. Τέλος οι μετρικές DIT και NOC, παρουσιάζουν αύξηση. Οι οργάνωση των κλάσεων του συστήματος σε ιεραρχία, καθώς και η εισαγωγή του στοιχείου αφαίρεσης Mediator, οδηγεί στην επιβάρυνση του NOC και του DIT. Όπως δείχνει και το σχήμα 4.86, η αρχική εικόνα του συστήματος δεν έχει κάποια ιεραρχία, αλλά η χρήση του προτύπου οργανώνει τις κλάσεις σε δυο ιεραρχίες. Συνεπώς η χρήση του προτύπου οδηγεί στην αύξηση του NOC και DIT ανά κλάση.

```
public class ConcreteColleague1 {
  ConcreteColleague2 cc2 = new ConcreteColleague2();
  Collegue cc4 = new ConcreteColleague4();
  public void execute() {
   cc2.execute();
   cc4.execute();
  }}

public class ConcreteColleague2 {
  Collegue cc3 = new ConcreteColleague3();
  Collegue cc4 = new ConcreteColleague4();
  public void execute() {
      cc3.execute();
      cc4.execute();
  }}

public class ConcreteColleague3 {
  Collegue cc4 = new ConcreteColleague4();
  public void execute() {
    cc4.execute();
}}

public class ConcreteColleague4 {
   public void execute() {
    System.out.println("the ConcreteColleague4().execute");
  }}
```

Σχήμα 4.88 το Σύστημα χωρίς Χρήση Mediator

```
public abstract class Collegue {
 Mediator mediator;
 abstract public void execute();
}
```

```
public class ConcreteColleague1 extends Collegue{
  public ConcreteColleague1(ConcreteMediator m) {
    mediator = m;
  }
  public void execute() {
    System.out.println("");
}}

public class ConcreteColleague2 extends Collegue{
  public ConcreteColleague2(ConcreteMediator m) {
    mediator = m;
  }
  public void execute() {
    System.out.println("");
  }}

public class ConcreteColleague3 extends Collegue {
  public ConcreteColleague3(ConcreteMediator m) {
    mediator = m;
  }
  public void execute() {
    System.out.println("");
  }}

public class ConcreteColleague4 extends Collegue {
  public ConcreteColleague4(ConcreteMediator m) {
    mediator = m;
  }
  public void execute() {
    System.out.println("The ConcreteColleague4().execute");
  }}
```

Σχήμα 4.89 η Οργάνωση των Κλάσεων ConcreteColleague σε Ιεραρχία

```
public interface Mediator {
    public void execute();
}
public class ConcreteMediator implements Mediator {
  ConcreteColleague1 cc1=new ConcreteColleague1(this);
  ConcreteColleague2 cc2 = new ConcreteColleague2(this);
  ConcreteColleague3 cc3 = new ConcreteColleague3(this);
  ConcreteColleague4 cc4 = new ConcreteColleague4(this);

  public void execute () {
      cc1.execute();
      cc2.execute();
      cc3.execute();
      cc4.execute();
      cc4.execute();
      cc4.execute();
    }}
```

Σχήμα 4.90 η Ιεραρχία του Προτύπου Mediator

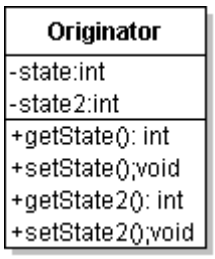
Πίνακας 4.16 οι Μεταβολές των Μετρικών με και χωρίς το Mediator

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mediator | | 0 | | 0 | | 0 | | 0 | | 1 | | 0 |
| Concrete Mediator1 | | 4 | | 0 | | 0 | | 5 | | 0 | | 1 |
| Colleague | | 0 | | 0 | | 0 | | 0 | | 4 | | 0 |
| Concrete Colleague1 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 0 | 0 | 0 | 1 |
| Concrete Colleague2 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 0 | 0 | 0 | 1 |
| Concrete Colleague3 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 1 |
| Concrete Colleague4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Overall Mean | 1.25 | 0.5 | 0 | 0 | 0 | 0 | 2.25 | 1.28 | 0 | 0.71 | 0 | 0.71 |

### *4.2.17. Memento*

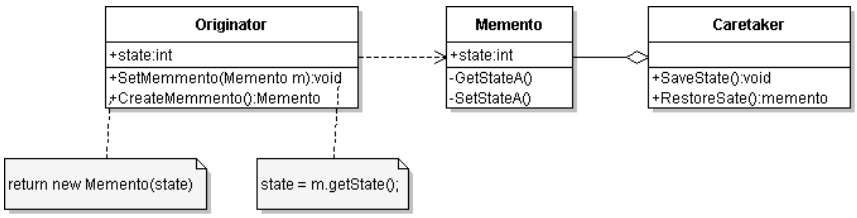
Το πρότυπο Memento δίνει μια λύση σε συστήματα στα οποία είναι απαραίτητη η εγγραφή των καταστάσεων από τις οποίες περνάει κάποιο αντικείμενο. Οι καταστάσεις αυτές πρέπει με κάποιον τρόπο να κρατούνται αποθηκευμένες

προκειμένου να υπάρχει η δυνατότητα επαναχρησιμοποίησης τους. Επιπλέον οι καταστάσεις αυτές θα πρέπει με κάποιον τρόπο να προστατεύονται έτσι ώστε δικαίωμα χρήσης να έχει μόνο το συγκεκριμένο αντικείμενο. Μια πρώτη προσέγγιση δείχνει το σχήμα 4.91. Συμφώνα με αυτήν οι καταστάσεις της κλάσης Originator υπάρχουν στην ίδια την κλάση και η διαχείριση τους γίνεται χρησιμοποιώντας διαφορετικές μεθόδους get και set. Το βασικό πρόβλημα που παρουσιάζει αυτή η προσέγγιση είναι αναφορικά με την συνεκτικότητα της κλάσης.




Σχήμα 4.91 η Κλάση Originator χωρίς Χρήση Memento

Το πρότυπο δίνει μια λύση σύμφωνα με την οποία η αποθήκευση των καταστάσεων του Originator γίνεται με την μορφή αντικειμένου Memento. Όπως δείχνει το σχήμα 4.92 το αντικείμενο Originator δημιουργεί ένα αντικείμενο Memento ενώ η κλάση Caretaker κάνει διαχείριση των καταστάσεων αποθηκεύοντας αντικείμενα τύπου Memento. Στο σχήμα 4.94 φαίνεται η κλάση Memento που υλοποιείται εσωτερικά της κλάσης Originator ενώ οι μέθοδοι της χαρακτηρίζονται private. Με αυτόν τον τρόπο πρόσβαση στις καταστάσεις έχει μόνο το αντικείμενο Originator.




Σχήμα 4.92 το Σύστημα με Χρήση Memento

Εξετάζοντας την χρήση του προτύπου Memento και το πώς επηρεάζει τις μετρικές στο σύστημα, βελτίωση παρουσιάζει το LCOM. Αυτό συμβαίνει γιατί η κλάση Originator δεν έχει πολλές μεθόδους get και set ενώ οι νέες κλάσεις του συστήματος δεν παρουσιάζουν έλλειψη συνεκτικότητας. Ακόμα οι μετρικές CBO και RFC

παρουσιάζουν επιβάρυνση. Αυτό συμβαίνει γιατί η εισαγωγή νέων κλάσεων στο σύστημα οδηγεί στην εμφάνιση συσχετίσεων ενώ η διαδικασία αποθήκευσης και ανάκτησης των καταστάσεων προϋποθέτει περισσότερη επικοινωνία. Τέλος οι μετρικές WMC, NOC και DIT παραμένουν αμετάβλητες καθώς το πρότυπο δεν επηρεάζει την πολυπλοκότητα κάποιας κλάσης ενώ παράλληλα δεν υποστηρίζει την χρήση ιεραρχιών.

```
public class Originator {
    private int state, state2;
   private void setState(int state) {
       state = state;
       }
   private String getState(){
    return state;
   }
   private void setState2(int state2) {
       state = state;
       }
   private String getState2(){
    return state;
```

Σχήμα 4.93 η Κλάση Originator χωρίς Χρήση Memento

```
public class Originator {
    private int state;
   public void setMemmento(int state) {
       System.out.println("Originator: Setting state to "+state);
       this.state = state;
   }

   public Memento CreateMemmento() {
       System.out.println("Originator: Saving to Memento.");
       return new Memento(state);
   }
```

```
    public class Memento {
      private int state2;
       private Memento(int stateToSave) {
       state2 = stateToSave;
       }
      private void setState(int state){
      state2 = state;
     }
     private int getdState() {
         return state;
     }
}}
```

Σχήμα 4.94 η Κλάση Originator με την Εσωτερική Κλάση Memento

```
public class Caretaker {
   private ArrayList<Originator.Memento> savedStates
= new ArrayList<Originator.Memento>();
  public void SaveState(Originator.Memento m) {
     savedStates.add(m); }
  public Originator.Memento RestoreSate(int index) {
     return savedStates.get(index);
  }}
```

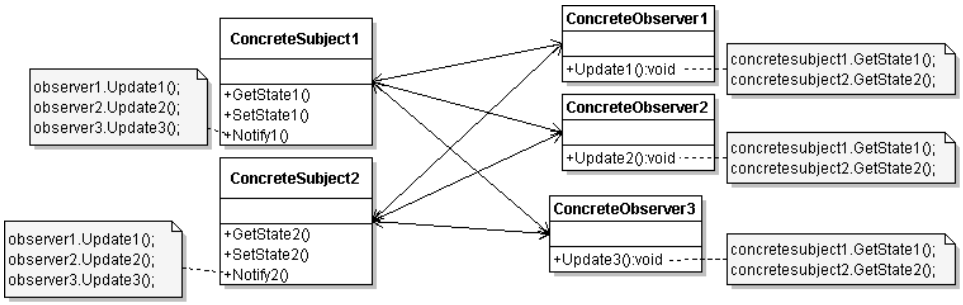
Σχήμα 4.95 η Κλάση Caretaker με Αναφορά σε Αντικείμενα Memento

Πίνακας 4.17 οι Μεταβολές των Μετρικών με και χωρίς το Memento

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Originator | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 4 | 0 | 0 | 0 | 0 |
| Memento | | 0 | | 0 | | 0 | | 2 | | 0 | | 0 |
| Caretaker | | 1 | | 0 | | 0 | 4 | 4 | | 0 | | 0 |
| Overall Mean | 0 | 0.67 | 0 | 0 | 1 | 0 | 3 | 3.3 | 0 | 0 | 0 | 0 |

*4.2.18. Observer*

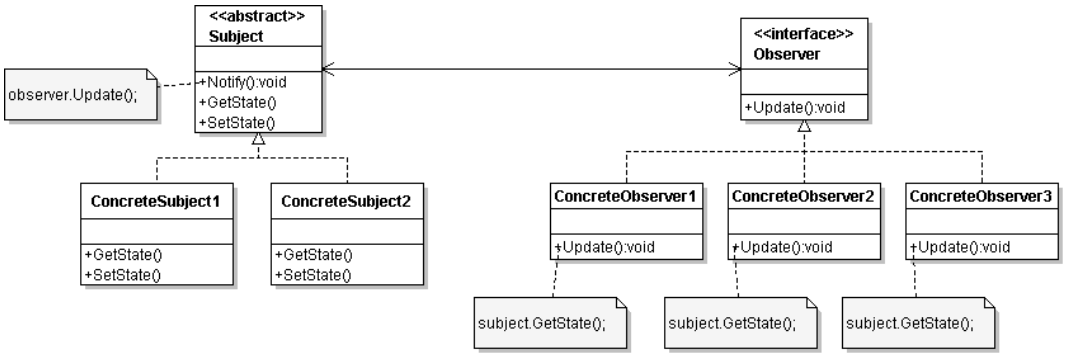
Το πρότυπο Observer χρησιμοποιείται σε περιπτώσεις που έχουμε κλάσεις οι οποίες ενδιαφέρονται για αλλαγές στην κατάσταση άλλων κλάσεων. Δηλαδή σε περιπτώσεις που γίνει κάποια αλλαγή στην κατάσταση ενός αντικειμένου μιας κλάσης, αυτόματα θα πρέπει να υπάρξει ενημέρωση των αντικειμένων των εξαρτώμενων κλάσεων. Έτσι λοιπόν το σύστημα αποτελείται από ένα σύνολο ConcreteSubject η εσωτερική κατάσταση των οποίων αλλάζει και ένα σύνολο από ConcreteObserver αντικειμένα, που ενδιαφέρονται δηλαδή για αυτές τις αλλαγές. Το πρόβλημα που πρέπει να αντιμετωπιστεί είναι η ενημέρωση των ConcreteObserver για αλλαγές που μπορεί να προκύψουν στα αντικείμενα ConcreteSubject. Ένας τρόπος για να γίνει αυτό φαίνεται στο σχήμα 4.96. Σύμφωνα με αυτήν την προσέγγιση οι κλάσεις ConcreteSubject έχουν αναφορές σε όλους τους τύπους των ενδιαφερόμενων αντικειμένων αλλά και το αντίθετο, δηλαδή οι κλάσεις ConcreteObserver έχουν αναφορές σε όλες τις κλάσεις ConcreteSubject. Το βασικό πρόβλημα που παρουσιάζει αυτή η προσέγγιση, είναι μεγάλη συσχέτιση μεταξύ των κλάσεων του συστήματος.




Σχήμα 4.96 το Σύστημα χωρίς Χρήση Observer

Η λύση που δίνει το πρότυπο φαίνεται στο σχήμα 4.97 και σύμφωνα με αυτήν την προσέγγιση, υπάρχει καθορισμένος τρόπος επικοινωνίας μεταξύ των κλάσεων που αλλάζει την εσωτερική τους κατάσταση και των κλάσεων που ενδιαφέρονται για τις αλλαγές αυτές. Πιο συγκεκριμένα ο καθορισμός της επικοινωνίας γίνεται στοιχείων αφαίρεσης, οι κλάσεις των αντικειμένων στις οποίες η εσωτερική κατάσταση αλλάζει, επεκτείνουν μια αφηρημένη κλάση βάσει της οποίας γίνεται η εκδήλωση του ενδιαφέροντος, ενώ η ενημέρωση των ενδιαφερόμενων κλάσεων πραγματοποιείται μέσω μιας αφηρημένης κλάσης που επεκτείνουν. Έτσι λοιπόν το πρότυπο Observer

εφαρμόζει την αρχή αντιστροφής εξαρτήσεων και επιτυγχάνει την μείωση των συσχετίσεων μεταξύ των κλάσεων του συστήματος.




Σχήμα 4.97 το Σύστημα με Χρήση Observer

Εξετάζοντας την χρήση του προτύπου στο σύστημα αναφορικά με τις μετρικές, βελτίωση παρουσιάζουν οι μετρικές CBO και RFC. Όπως προαναφέρθηκε, το πρότυπο υλοποιεί ένα ενιαίο τρόπο επικοινωνίας μεταξύ των ConcreteSubject και των ConcreteObserver. Η επικοινωνία δηλαδή πραγματοποιείται μέσω στοιχείων αφαίρεσης. Όπως φαίνεται και στον πίνακα 4.18 η χρήση του προτύπου έχει ως συνέπεια το μηδενισμό των συσχετίσεων. Παράλληλα η χρήση των στοιχείων αφαίρεσης βοηθάει τις κλάσεις του συστήματος ώστε η πρόσβασή σε εξωτερικές μεθόδους να πραγματοποιείται με όμοιο τρόπο. Έτσι επιτυγχάνεται και η μείωση του μέσου RFC του συστήματος. Αντίθετα αύξηση παρουσιάζουν οι μετρικές NOC και DIT. Η εισαγωγή των δυο αφηρημένων κλάσεων στο σύστημα έχει ως αποτέλεσμα την αύξηση του DIT στις κλάσεις του συστήματος και επιπλέον υπάρχει αύξηση του NOC λόγω των αφηρημένων κλάσεων. Τέλος οι μετρικές LCOM και WMC παραμένουν αμετάβλητες.

```
public class ConcreteSubject1 {
    private String string1;
    private String string2;
    ConcreteObserver1 ob1;
    ConcreteObserver2 ob2;
    ConcreteObserver3 ob3;
```

```
    public void Notify1() {
        ob1.Update1();
        ob2.Update2();
        ob3.Update3();
    }

    public  void setState1() {
string1 = "Concrete";
        string2 = "Subject1";
}
public String getState1() {
String tmp = new String();
tmp = string1 + " " + string2;
return tmp;
}}

public class ConcreteSubject2 {
    private String string1;
    private String string2;
    ConcreteObserver1 ob1;
    ConcreteObserver2 ob2;
    ConcreteObserver3 ob3;

public void Notify2() {
ob1.Update1();
ob2.Update2();
ob3.Update3();
}
public  void setState2() {
string1 = "Concrete";
string2 = "Subject2";
}
```

```
public String getState2() {
String tmp = new String();
tmp = string1 + " " + string2;
return tmp;
}}
```

Σχήμα 4.98 οι Κλάσεις ConcreteSubject

```
public class ConcreteObserver1 {
    ConcreteSubject1 next;
    ConcreteSubject2 next2;

   ConcreteObserver1 (ConcreteSubject1 next,ConcreteSubject2 next2) {
   this.next = next;
   this.next2= next2;
}
public void Update1() {
next.getState1();
next2.getState2();
}}

public class ConcreteObserver2 {
    ConcreteSubject1 next;
    ConcreteSubject2 next2;

    ConcreteObserver2 (ConcreteSubject1 next,ConcreteSubject2 next2) {
    this.next = next;
    this.next2= next2;
}
   public void Update2() {
  next.getState1();
  next2.getState2();
```

```
}}

public class ConcreteObserver3 {
   ConcreteSubject1 next;
   ConcreteSubject2 next2;
ConcreteObserver3 (ConcreteSubject1 next,ConcreteSubject2 next2) {
this.next = next;
this.next2= next2;
}
public void Update3() {
next.getState1();
next2.getState2();
}}
```

Σχήμα 4.99 οι Κλάσεις ConcreteObserver

```
abstract public class  Observer{
Subject next;
Observer (Subject netx) {
this.next = next;
}
abstract public void Update();
}

public class ConcreteObserver1 extends Observer {
ConcreteObserver1(Subject next){
super(next);
}
public void Update() {
next.getState();
}}

public class ConcreteObserver2 extends Observer {
```

```
ConcreteObserver2 (Subject next){
super(next);
}
public void Update() {
next.getState();
}}

public class ConcreteObserver3 extends Observer {
ConcreteObserver3(Subject next){
super(next);
}
public void Update() {
next.getState();
}}
```

Σχήμα 4.100 το Στοιχείο Αφαίρεσης Observer και οι Υλοποιήσεις του

```
abstract  public class Subject{
 private ArrayList observers = new ArrayList();
 public void notifyObservers() {
Iterator elements = observers.iterator();
  while (elements.hasNext()) {
     ((Observer)elements.next()).Update();
  }}
  abstract public void setState();
  abstract public String getState();
}

public class ConcreteSubject1 extends Subject
{
private String string1;
private String string2;
public  void setState() {
```

```
string1 = "Concrete";
string2 = "Subject1";
}
public String getState() {
String tmp = new String();
tmp = string1 + " " + string2;
return tmp;
}}

public class ConcreteSubject2 extends Subject{
private String string1;
private String string2;
public  void setState() {
string1 = "Concrete";
 string2 = "Subject2";
}
public String getState() {
String tmp = new String();
tmp = string1 + " " + string2;
return tmp;
}}
```

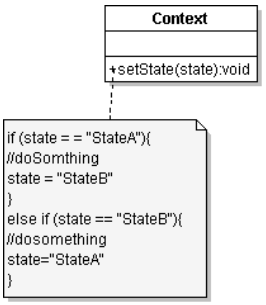
Σχήμα 4.101 το Στοιχείο Αφαίρεσης Subject και οι Υλοποιήσεις του

Πίνακας 4.18 οι Μεταβολές των Μετρικών με και Χωρίς το Observer

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ConcreteSubject1 | 3 | 0 | 0 | 0 | 0 | 0 | 6 | 2 | 0 | 0 | 0 | 1 |
| ConcreteSubject2 | 3 | 0 | 0 | 0 | 0 | 0 | 6 | 2 | 0 | 0 | 0 | 1 |
| ConcreteObserever1 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 | 1 |
| ConcreteObserever2 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 | 1 |
| ConcreteObserever3 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 | 1 |
| Subject | | 0 | 0 | 0 | 0 | 0 | | 2 | | 2 | | 0 |
| Obserever | | 0 | 0 | 0 | 0 | 0 | | 0 | | 3 | | 0 |
| Overall Mean | 2.4 | 0 | 0 | 0 | 0 | 0 | 4.2 | 1.7 | 0 | 0.7 | 0 | 0.7 |

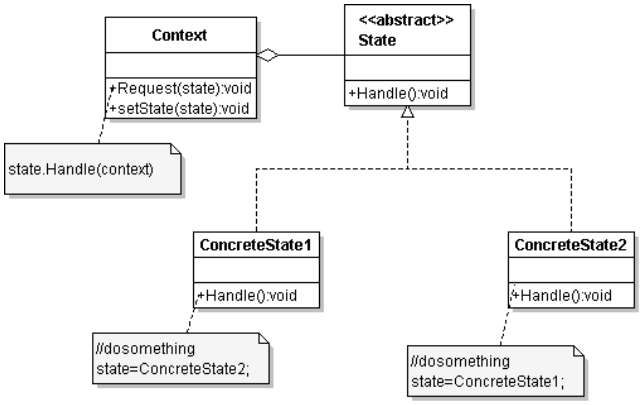
*4.2.19. State*

Πολλές φορές αντικείμενα κλάσεων χαρακτηρίζονται από διαφορετικές καταστάσεις. Οι καταστάσεις αυτές χαρακτηρίζουν το αντικείμενο για την συγκεκριμένη χρονική στιγμή. Επιπλέον η κατάσταση που βρίσκεται το αντικείμενο μια χρονική στιγμή μπορεί να εξαρτάται από την προηγούμενη του κατάσταση. Το πρότυπο δίνει μια λύση σε περιπτώσεις στις οποίες έχουμε αντικείμενα τα οποία μπορούν να βρεθούν σε πολλές διαφορετικές καταστάσεις. Ζητούμενο είναι ο τρόπος με τον οποίο το αντικείμενο θα μεταβαίνει στις διαφορετικές καταστάσεις. Ένας τρόπος είναι να υλοποιήσουμε μια μέθοδο εσωτερικά στην κλάση γι αυτόν το σκοπό. Όπως φαίνεται στο σχήμα 4.102, η μέθοδος δέχεται μια παράμετρο και ανάλογα με την τιμή της, το αντικείμενο μεταβαίνει στην σωστή κατάσταση. Δηλαδή με την χρήση μιας πολλαπλής δομής επιλογής διαπιστώνεται η κατάσταση του αντικειμένου και έπειτα πραγματοποιείται η αλλαγή της κατάστασης. Μια τέτοια προσέγγιση θα έχει ως αποτέλεσμα μεγάλη πολυπλοκότητα της κλάσης και η πολυπλοκότητα αυτή θα είναι συνέπεια της πολλαπλής δομής επιλογής που χρησιμοποιήθηκε προκειμένου να γίνει η διαπίστωση της κατάστασης.




Σχήμα 4.102 η Κλάση Context χωρίς Χρήση State

Το πρότυπο δίνει μια λύση χρησιμοποιώντας μια ιεραρχία κλάσεων αντικείμενα της οποίας ορίζουν τις καταστάσεις. Με αυτόν τον τρόπο αντί να χρησιμοποιείται δομή έλεγχου για τον ορισμό της κατάστασης του αντικειμένου, χρησιμοποιούνται αντικείμενα της ιεραρχίας ενώ με την χρήση της ενθυλάκωσης ορίζεται η τρέχουσα κατάσταση.




Σχήμα 4.103 το Σύστημα με Χρήση State

Εξετάζοντας την χρήση του προτύπου, βελτίωση παρουσιάζει η μετρική WMC όπως δείχνει και το σχήμα 4.106 καθώς με την χρήση του προτύπου η κλάση Context δε εφαρμόζει κάποια δομή επιλογής. Συνεπώς υπάρχει μηδενισμός στην πολυπλοκότητα της κλάσης, η οποία οδηγεί στην βελτίωση του μέσου WMC στο σύστημα. Αντίθετα αύξηση παρουσιάζει το RFC. Το πρότυπο υποστηρίζει την κλήση εξωτερικών μεθόδων για τον ορισμό της κατάστασης του κάθε αντικείμενου. Όπως φαίνεται στον πίνακα 4.109 υπάρχει αύξηση του RFC στην κλάση Context και συνέπεια αυτού είναι η επιβάρυνση του μέσου RFC στο σύστημα. Επιπλέον αύξηση παρουσιάζουν και οι μετρικές DIT και NOC και αυτό συμβαίνει γιατί η χρήση του προτύπου προϋποθέτει τον υλοποίηση των καταστάσεων σε κλάσεις και την οργάνωση τους σε ιεραρχία. Τέλος δεν υπάρχει μεταβολή στις μετρικές CBO και LCOM.

```
public class Context {

   private String state;
   Context (String state){
      this.state = state;
   }

   public void setState (){

      if (state =="StateA"){
         System.out.println("this is StateA");
```

```
        state = "StateA";
      }
      else if (state =="StateB"){
        System.out.println("this is StateB");
        state = "StateB";
      }}}
```

Σχήμα 4.104 η Κλάση Context χωρίς Χρήση State

```
public interface State {
    public void Handle();
}

public class ConcreteStateA implements State {
  public void Handle(){
    System.out.println("this is StateA");
  }}

public class ConcreteStateB {
  public void Handle(){
    System.out.println("this is StateB");
  }}
```

Σχήμα 4.105 η Ιεραρχία του Προτύπου

```
public class Context {
private State state;
Context (State state){
this.state = state;
}
 public void setState (){
 Request ();
 }
```

```
public void Request (){
 state.Handle();
}}
```

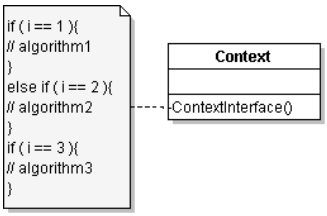
Σχήμα 4.106 η Κλάση Context με Χρήση State

Πίνακας 4.19 οι Μεταβολές των Μετρικών με και χωρίς το State

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Context | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 3 | 0 | 0 | 0 | 0 |
| State | | 0 | | 0 | 0 | 0 | | 0 | | 2 | | 0 |
| ConcreteState1 | | 0 | | 0 | 0 | 0 | | 1 | | 0 | | 1 |
| ConcreteState2 | | 0 | | 0 | 0 | 0 | | 1 | | 0 | | 1 |
| Overall Mean | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 1.25 | 0 | 0.5 | 0 | 0.5 |

*4.2.20. Strategy*

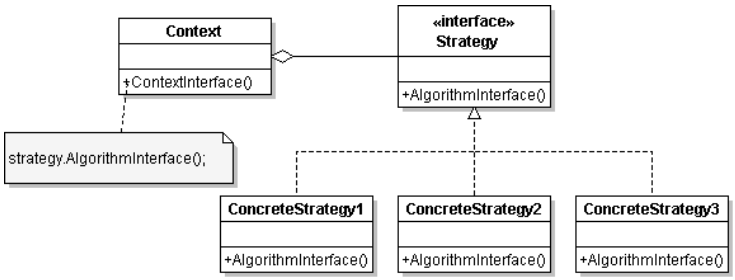
Το πρότυπο Strategy παρέχει μια λύση σε συστήματα που η εκτέλεση των λειτουργιών τους απαιτεί την χρήση διαφορετικών αλγορίθμων. Πιο συγκεκριμένα το σύστημα περιέχει μια κλάση η οποία για την εκτέλεση των λειτουργιών της χρησιμοποιεί ένα σύνολο από αλγορίθμους. Ζητούμενο είναι ένας τρόπος βάσει του οποίου θα γίνει η οργάνωση των διαφορετικών αλγορίθμων αλλά και το πώς θα γίνει η χρήση τους από την κλάση. Στο σχήμα 4.107 φαίνεται μια πρώτη προσέγγιση σύμφωνα με την οποία, η υλοποίηση των αλγορίθμων γίνεται μέσα στην κλάση. Ακόμα υπάρχει και μια δομή επιλογής η οποία καθορίζει ποιος αλγόριθμος θα εκτελεσθεί. Το βασικό πρόβλημα που παρουσιάζει αυτή η προσέγγιση είναι η μεγάλη πολυπλοκότητα που παρουσιάζει η συγκεκριμένη κλάση.




Σχήμα 4.107 η Κλάση Context χωρίς Χρήση Strategy

Το πρότυπο υποστηρίζει την οργάνωση των διαφορετικών αλγορίθμων σε μια ιεραρχία. Όπως φαίνεται και στο σχήμα 4.108, το πρότυπο υποστηρίζει την χρήση

ενός στοιχείου αφαίρεσης το οποίο υλοποιείται ανάλογα με τους αλγορίθμους που υποστηρίζει το σύστημα. Ακόμα η κλάση Context ενθυλακώνει αντικείμενα της συγκεκριμένης ιεραρχίας προκειμένου να εκτελέσει τον κατάλληλο αλγόριθμο.




Σχήμα 4.108 το Σύστημα με χρήση Strategy

Αναφορικά με τις μετρικές, η χρήση του προτύπου βελτιώνει το WMC. Όπως δείχνει και το σχήμα 4.111 το προτύπου απαλλάσσει την κλάση από την χρήση της δομής επιλογής. Συνεπώς η κλάση Context παρουσιάζει μείωση της μετρικής WMC ενώ η μείωση αυτή οδηγεί στην βελτίωση του μέσου WMC του συστήματος. Αντίθετα αύξηση παρουσιάζουν οι μετρικές NOC και DIT λόγω της εισαγωγής ιεραρχίας στο σύστημα. Τέλος αμετάβλητες μένουν οι μετρικές RFC και LCOM. Αναφορικά με την μετρική RFC όπως δείχνει και το σχήμα 4.111 η κλάση Context καλεί μια εξωτερική μέθοδο, όμως όπως δείχνει ο πίνακας 4.20, η αύξηση αυτή δεν επηρεάζει το μέσο RFC στου σύστημα και αυτό γιατί το πλήθος των κλάσεων βοηθάει στην καλύτερη κατανομή του RFC ανά κλάση.

```
public class Context{
private int i;
 public void ContextInterface(){

   if (this.i == 1){
   //Algorithm1
   }
   else if (this.i == 2){
      //Algorithm2
```

```
    }
    else if (this.i ==3){
      //Algorithm3
    }
    else{
    System.out.println("no algorithm");
    }}
```

Σχήμα 4.109 η Κλάση Context χωρίς Χρήση του Strategy

```
public interface Strategy {
   public void Algorithminterface();
}

public class ConcreteStrategy1 implements Strategy{
   public void Algorithminterface(){
   System.out.println("Algorithm 1");
   }}

public class ConcreteStrategy2 implements Strategy{
   public void Algorithminterface(){
   System.out.println("Algorithm 2");
   }}

public class ConcreteStrategy3 implements Strategy{
   public void Algorithminterface(){
   System.out.println("Algorithm 3");
   }}
```

Σχήμα 4.110 η Ιεραρχία του Προτύπου Strategy

```
public class Context {
   Strategy strategy;
```

```
  Context(Strategy strategy){
    this.strategy = strategy;
    }
public void ContextInterface(){
System.out.print("This context use ");
strategy.Algorithminterface();
}}
```

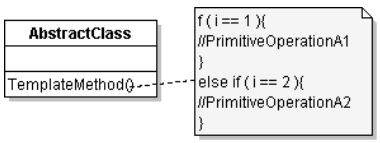
Σχήμα 4.111 η Κλάση Context με Χρήση του Strategy

Πίνακας 4.20 οι Μεταβολές των Μετρικών με και χωρίς το Strategy

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Context | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| Strategy | | 0 | | 0 | | 0 | | 0 | | 3 | | 0 |
| ConcreteStrategy 1 | | 0 | | 0 | | 0 | | 1 | | 0 | | 1 |
| ConcreteStrategy 2 | | 0 | | 0 | | 0 | | 1 | | 0 | | 1 |
| ConcreteStrategy 3 | | 0 | | 0 | | 0 | | 1 | | 0 | | 1 |
| Overall Mean | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 1 | 0 | 0.6 | 0 | 0.6 |

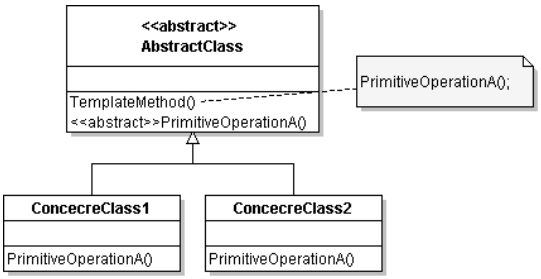
*4.2.21. Template Method*

Το πρότυπο Template Method δίνει λύση σε συστήματα τα οποία χρησιμοποιούν ένα συγκεκριμένο αλγόριθμο. Επιπλέον ο αλγόριθμός αυτός μπορεί να διαφοροποιείται σε κάποια βήματα ανάλογα με την εκτέλεση. Ζητούμενο σε τέτοια συστήματα είναι ο τρόπος με τον οποίο θα υλοποιούνται τα βήματα που διαφοροποιούνται ανάλογα με την εκτέλεση. Μια πρώτη προσέγγιση φαίνεται στο σχήμα 4.112. Σύμφωνα με αυτή, τα βήματα στα οποία διαφοροποιείται ο αλγόριθμος υλοποιούνται μέσα στην ίδια κλάση. Η συγκεκριμένη προσέγγιση έχει αυξημένη πολυπλοκότητα η οποία οφείλεται στην δομή επιλογής που καθορίζει πιο βήμα του αλγορίθμου θα εκτελεστεί.




Σχήμα 4.112 η Κλάση χωρίς Χρήση Template Method

Το πρότυπο δίνει μια λύση βάσει της οποίας ο αλγόριθμος σχεδιάζεται ως μια ιεραρχία κλάσεων. Το στοιχείο αφαίρεσης της ιεραρχίας περιέχει την υλοποίηση του κορμού και τα βήματα που δεν μεταβάλλονται, ενώ το σύνολο των υλοποιήσεων των αφηρημένων μεθόδων καθορίζει τα βήματα που μεταβάλλονται. Στο σχήμα 4.113 φαίνεται η σχεδίαση του αλγορίθμου με την χρήση του προτύπου.




Σχήμα 4.113 η Κλάση με Χρήση Template Method

Οι μετρικές του συστήματος δείχνουν ότι η χρήση του προτύπου μειώνει την πολυπλοκότητα. Πιο συγκεκριμένα όπως δείχνει και το σχήμα 4.115 το πρότυπο δεν χρησιμοποιεί καμία δομή επιλογής και συνεπώς το WMC της κλάσης που υλοποιεί τον αλγόριθμο θα μηδενιστεί. Όπως δείχνει ο πίνακας 4.21 η μείωση αυτή οδηγεί στην βελτίωση του μέσου WMC του συστήματος. Αντίθετα αύξηση παρουσιάζουν οι μετρικές DIT και NOC. Η χρήση του προτύπου υποστηρίζει την επέκταση της κλάσης με υποκλάσεις. Ακόμα αύξηση παρουσιάζει και η μετρική RFC, όπως δείχνει και το σχήμα 4.115. Η χρήση του προτύπου απαιτεί κλήση μεθόδων που υλοποιούνται εξωτερικά της κλάσης. Τέλος οι μετρικές CBO και LCOM δεν παρουσιάζουν καμία μεταβολή στο σύστημα.

```
public class AbstractClass {
   String choice;
   AbstractClass(String choice){
   this.choice = choice;
   }
   public void TemplateMethod(){
      System.out.println("this is the TemplateMethod");
```

```
if (this.choice.equalsIgnoreCase("OperationA1")){
//PrimitiveOperationA1
}
else if (this.choice.equalsIgnoreCase("OperationA2")){
//PrimitiveOperationA2
}}
```

Σχήμα 4.114 η Κλάση χωρίς την Χρήση του Προτύπου Template Method

```
abstract class AbstractClass {
   public void TemplateMethod(){
      System.out.println("this is the TemplateMethod");
   PrimitiveOperationA();
   }
   abstract public  void PrimitiveOperationA();
}

public class ConcreteClass1 extends AbstractClass {
public  void PrimitiveOperationA(){
System.out.println("with ConcreteClassA1");
}}

public class ConcreteClass2 extends AbstractClass {
public  void PrimitiveOperationA(){
System.out.println("with ConcreteClassA2");
}}
```

Σχήμα 4.115 η Ιεραρχία μετά την Χρήση του Template Method

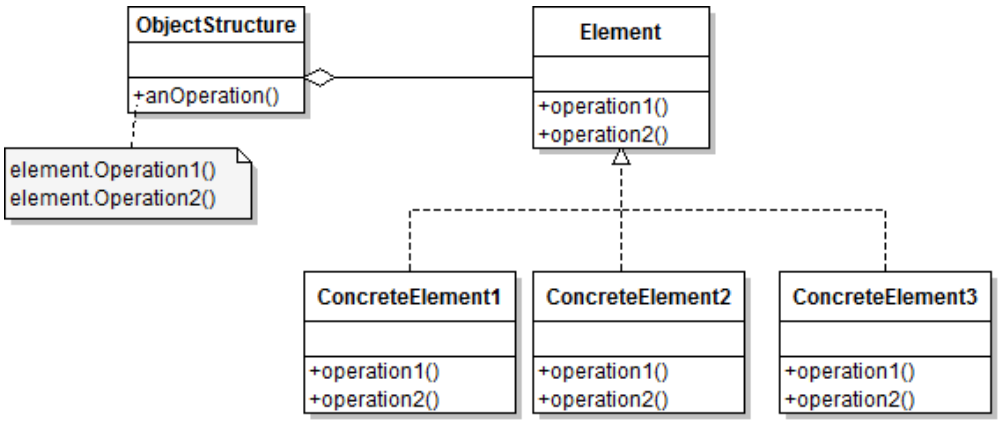
Πίνακας 4.21 οι Μεταβολές των Μετρικών με και χωρίς το Template Method

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AbstractClass | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 0 |
| ConcreteClass1 | | 0 | | 0 | | 0 | | 1 | | 0 | | 1 |
| ConcreteClass2 | | 0 | | 0 | | 0 | | 1 | | 0 | | 1 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Overall Mean | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 1.3 | 0 | 0.6 | 0 | 0.6 |

*4.2.22. Visitor*

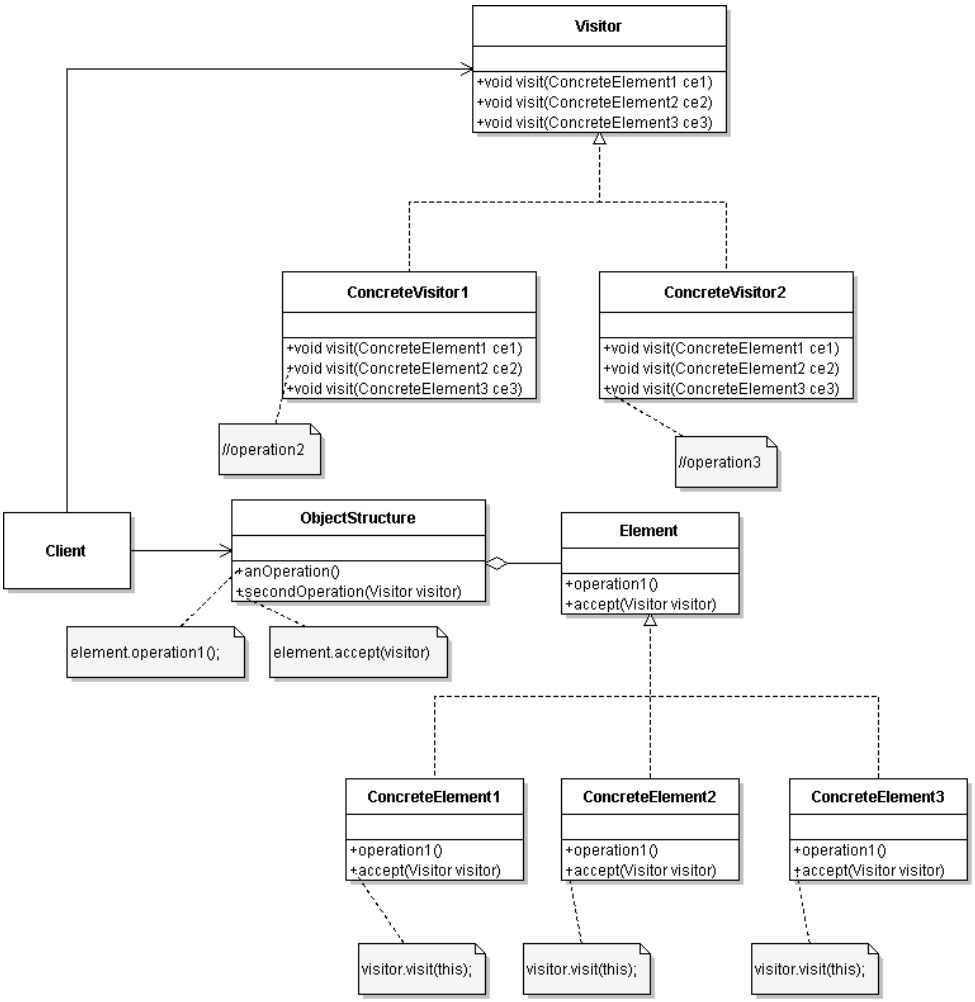
Το πρότυπο Visitor δίνει λύσεις στο πρόβλημα της εισαγωγής μιας νέας λειτουργίας σε μια εφαρμογή που βασίζεται σε μια ιεραρχία κλάσεων. Το ζητούμενο είναι ο τρόπος με τον οποίο θα γίνει η προσθήκη της νέας αυτής λειτουργίας στην ιεραρχία των κλάσεων έτσι ώστε τα αντικείμενα της ιεραρχίας αυτής να μπορούν να εκτελέσουν την νέα λειτουργία στην εφαρμογή. Το σχήμα 4.116 δείχνει μια πρώτη προσέγγιση σύμφωνα με την οποία για την νέα λειτουργία πρέπει να γίνει προσθήκη μιας μεθόδου σε όλες τις κλάσεις της ιεραρχίας. Το βασικό πρόβλημα αυτής της προσέγγισης, πέρα από ην παραβίαση της αρχής της κλειστότητας, είναι ότι παρουσιάζει μείωση της συνεκτικότητας στις κλάσεις της ιεραρχίας.




Σχήμα 4.116 το Σύστημα χωρίς Χρήση Visitor

Το πρότυπο υποστηρίζει την ομαδοποίηση των μεθόδων που αντιστοιχούν στην νέα λειτουργία του συστήματος σε μια κλάση. Όπως δείχνει το σχήμα 4.117 για την εκτέλεση της νέας λειτουργίας, θα πρέπει να γίνει κλήση των μεθόδων visit() από τις κλάσεις τις ιεραρχίας. Επιπλέον κάθε κλάση της ιεραρχίας θα πρέπει να καλεί την κατάλληλη μέθοδο visit και για τον λόγο αυτό κάθε κλάση της ιεραρχίας είναι εφοδιασμένη με μια μέθοδο accept (Visitor visitor). Όπως δείχνει και το σχήμα 4.121 μέσω της μεθόδου accept (Visitor visitor) το αντικείμενο της ιεραρχίας δείχνει τον

τύπο του σε αντικείμενα Visitor και με αυτόν τον τρόπο γίνεται η κλήση της σωστής μεθόδου visit().




Σχήμα 4.117 το Σύστημα με Χρήση Visitor

Αναφορικά με τις μετρικές, η χρήση του προτύπου βελτιώνει την μετρική LCOM. Όπως δείχνει και ο πίνακας 4.22 η χρήση του προτύπου μειώνει την μετρική LCOM στις κλάσεις της ιεραρχίας. Η μείωση αυτή οδηγεί στην βελτίωση του μέσου LCOM του συστήματος. Αντίθετα αύξηση παρουσιάζουν οι μετρικές NOC και DIT καθώς η χρήση του προτύπου απαιτεί την εισαγωγή μιας νέας ιεραρχίας στο σύστημα η οποία υλοποιεί τις λειτουργίες. Ακόμα αύξηση παρουσιάζει και η μετρική RFC αφού για την εκτέλεση της νέας λειτουργίας το πρότυπο απαιτεί επικοινωνία μεταξύ των κλάσεων της ιεραρχίας και της κλάσης που παρέχει την νέα λειτουργία. Τέλος οι μετρικές WMC και CBO παραμένουν αμετάβλητες.

```
public interface Element {
   public void operation1();
   public void operation2();
}

public class ConcreteElement1 implements Element {
public void operation1(){
   System.out.println("operation1, ConcreteElement1");
   }
   public void operation2(){
   System.out.println("operation2, ConcreteElement1");
   }}

public class ConcreteElement2 implements Element {
public void operation1(){
   System.out.println("operation1, ConcreteElement2");
   }
   public void operation2(){
   System.out.println("operation2, ConcreteElement2");
   }}

public class ConcreteElement3 implements Element {
public void operation1(){
   System.out.println("operation1, ConcreteElement3");
   }
   public void operation2(){
   System.out.println("operation2, ConcreteElement3");
   }}
```

Σχήμα 4.118 η Ιεραρχία χωρίς Visitor

```
public class ObjectStructure {
    private Element element;
    ObjectStructure( Element element){
    this.element = element;
    }
    public void anOperation(){
    element.operation1();
    element.operation2();
    }}
```

Σχήμα 4.119 η Κλάση ObjectStructure χωρίς Visitor

```
public interface Visitor {
    public void visit(ConcreteElement1 ce1);
    public void visit(ConcreteElement2 ce2);
    public void visit(ConcreteElement3 ce3);
}

public class ConcreteVisitor1 implements Visitor{
    public void visit(ConcreteElement1 ce1){
    System.out.println("ConcreteVisitor1, ConcreteElement1");
    }
    public void visit (ConcreteElement2 ce2){
    System.out.println("ConcreteVisitor1, ConcreteElement2");
    }
    public void visit (ConcreteElement3 ce3){
    System.out.println("ConcreteVisitor3, ConcreteElement3");
    }}

public class ConcreteVisitor2 implements Visitor{
    public void visit(ConcreteElement1 ce1){
    System.out.println("ConcreteVisitor2, ConcreteElement1");
    }
```

```
public void visit (ConcreteElement2 ce2){
System.out.println("ConcreteVisitor2, ConcreteElement2");
}
public void visit (ConcreteElement3 ce3){
System.out.println("ConcreteVisitor2, ConcreteElement3");
}}
```

Σχήμα 4.120 η Ιεραρχία του Visitor

```
public interface Element {
   public void operation1();
   public void accept(Visitor visitor);
}

public class ConcreteElement1 implements Element{
   public void operation1(){
   System.out.println("operation1, ConcreteElement1");
   }
   public void accept(Visitor visitor){
    visitor.visit(this);
   }}

public class ConcreteElement2 implements Element {
   public void operation1(){
   System.out.println("operation1, ConcreteElement2");
   }
   public void accept(Visitor visitor){
   visitor.visit(this);
   }}

public class ConcreteElement3 implements Element{
   public void operation1(){
   System.out.println("operation1, ConcreteElement2");
```

```
}
public void accept(Visitor visitor){
visitor.visit(this);
}}
```

Σχήμα 4.121 η Ιεραρχία του Συστήματος με Χρήση Visitor

```
public class ObjectStructure {
   private Element element;
   ObjectStructure( Element element){
   this.element = element;
   }
   public void anOperation(){
   element.operation1();
   }
   public void secondOperation(Visitor visitor){
   element.accept(visitor);
   }}
```

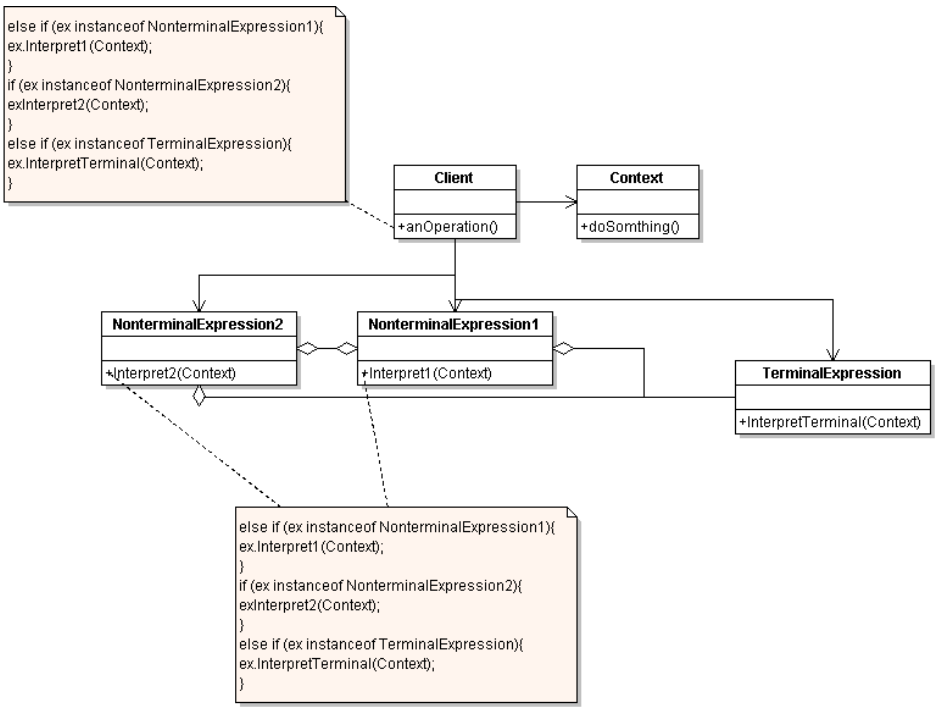
Σχήμα 4.122 η Κλάση ObjectStructure με Χρήση Visitor

Πίνακας 4.22 οι Μεταβολές των Μετρικών με και χωρίς το Visitor

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ObjectStructure | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 4 | 0 | 0 | 0 | 0 |
| Element | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 0 | 0 |
| Concrete Element 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 3 | 0 | 0 | 1 | 1 |
| Concrete Element 2 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 3 | 0 | 0 | 1 | 1 |
| Concrete Element 3 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 3 | 0 | 0 | 1 | 1 |
| Visitor | | 0 | | 0 | | 0 | | 0 | | 2 | | 0 |
| Concrete Visitor1 | | 0 | | 0 | | 0 | | 3 | | 0 | | 1 |
| Concrete Visitor2 | | 0 | | 0 | | 0 | | 3 | | 0 | | 1 |
| Overall Mean | 0 | 0 | 0 | 0 | 0.6 | 0 | 1.8 | 2 | 0.6 | 0.625 | 0.6 | 0.625 |

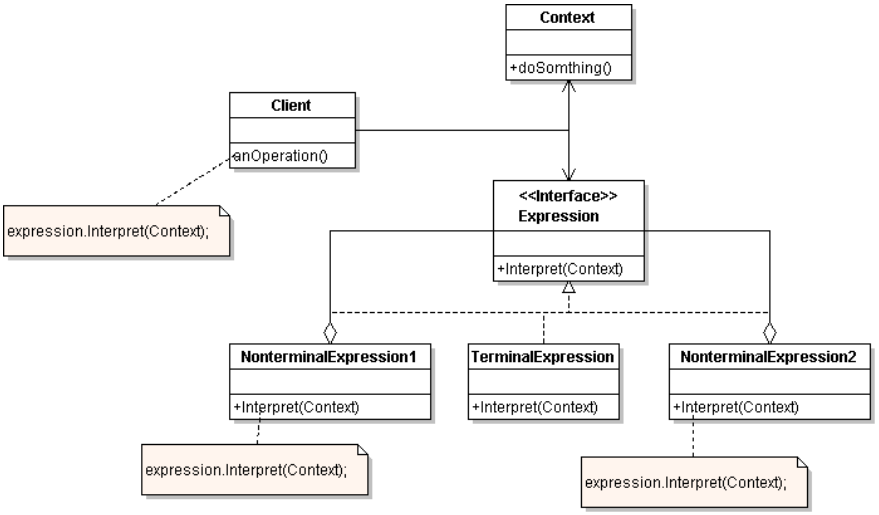
*4.2.23. Interpreter*

Το πρότυπο Interpreter δίνει μια λύση σε συστήματα τα οποία χρησιμοποιούν μια γλώσσα και θέλουν να διερμηνεύσουν προτάσεις σε αυτήν την γλώσσα. Το ζητούμενο είναι ο τρόπος με τον οποίο θα γίνεται η αναπαράσταση της γραμματικής αυτής προκειμένου να χρησιμοποιηθεί μαζί με τον διερμηνέα για την διερμήνευση των προτάσεων στην γλώσσα. Μια πρώτη προσέγγιση δείχνει η εικόνα 4.123 σύμφωνα με αυτή οι μη τερματικές εκφράσεις ενθυλακώνουν άλλες εκφράσεις που μπορεί να είναι τερματικές ή μη τερματικές. Η προσέγγιση αυτή οδηγεί σε υψηλή σύζευξη μεταξύ των τερματικών και των μη τερματικών εκφράσεων.




Σχήμα 4.123 το Σύστημα χωρίς Χρήση Interpreter

Το πρότυπο δίνει μια λύση σύμφωνα με την οποία οι τερματικές και η μη τερματικές εκφράσεις οργανώνονται σε μια ιεραρχία. Όπως δείχνει και το σχήμα 4.124, οι εκφράσεις, υλοποιούν ένα στοιχείο αφαίρεσης ενώ οι μη τερματικές εκφράσεις έχουν αναφορές σε αυτό το στοιχείο.




Σχήμα 4.124 το Σύστημα με Χρήση Interpreter

Εξετάζοντας το σύστημα αναφορικά με τις μετρικές, η χρήση του προτύπου βελτιώνει τις μετρικές CBO, WMC και RFC. Όπως δείχνει και ο πίνακας 4.23 η χρήση του προτύπου μηδενίζει το CBO των εκφράσεων. Επιπλέον στις ίδιες κλάσεις μηδενισμό έχουμε και στη μετρική WMC, αυτό συμβαίνει γιατί οι κλάσεις των μη τερματικών εκφράσεων δε περιέχουν κάποια δομή επιλογής. Ακόμα μείωση παρουσιάζει και το RFC, η οργάνωση των εκφράσεων σε ιεραρχία βοηθάει στην ομοιόμορφη επικοινωνία του Client με τις τερματικές και μη τερματικές εκφράσεις. Αντίθετα αύξηση παρουσιάζουν οι μετρικές DIT και NOC που οφείλεται στην χρήση του στοιχείου αφαίρεσης. Τέλος η συνεκτικότητα των κλάσεων δεν μεταβάλλεται, συνεπώς το μέσο LCOM του συστήματος παραμένει σταθερό.

```
public class TerminalExpression {
public void interpretTerminal(){
System.out.println("TerminalExpression,interpret");
}}

public class NonTerminalExpression1 {
   private Object ex;
   NonTerminalExpression1(Object ex ){
   this.ex=ex;
```

```
    }
    public void interpreter2(){
    if (ex instanceof TerminalExpression){
    TerminalExpression ex1 = (TerminalExpression)ex;
    ex1.interpretTerminal();
    }
    else if (ex instanceof NonTerminalExpression1){
    NonTerminalExpression1 ex1 = (NonTerminalExpression1)ex;
    ex1.interpreter2();
    }
    else if (ex instanceof TerminalExpression){
    TerminalExpression ex1 = (TerminalExpression)ex;
    ex1.interpretTerminal();
    }}}

public class NonTerminalExpression2 {
    private Object ex;
    NonTerminalExpression2(Object ex ){
    this.ex=ex;
    }
    public void interpreter2(){
    if (ex instanceof TerminalExpression){
    TerminalExpression ex1 = (TerminalExpression)ex;
    ex1.interpretTerminal();
    }
    else if (ex instanceof NonTerminalExpression1){
    NonTerminalExpression1 ex2 = (NonTerminalExpression1)ex;
    ex2.interpreter2();
    }
    else if (ex instanceof TerminalExpression){
    TerminalExpression ex3 = (TerminalExpression)ex;
    ex3.interpretTerminal();
```

```
}}}
```

Σχήμα 4.125 οι Εκφράσεις χωρίς Interpreter

```
public class Context {
    public void doSomething(){
    System.out.println("Context");
    }}
```

Σχήμα 4.126 η Κλάση Context

```
public class Client {
private Object ex;
public void anOperation(Context context){
   if (ex instanceof TerminalExpression){
   TerminalExpression ex1 = (TerminalExpression)ex;
   ex1.interpretTerminal();
   }
   else if (ex instanceof NonTerminalExpression1){
   NonTerminalExpression1 ex1 = (NonTerminalExpression1)ex;
   ex1.interpreter2();
   }
   else if (ex instanceof TerminalExpression){
   TerminalExpression ex1 = (TerminalExpression)ex;
   ex1.interpretTerminal();
   }}}
```

Σχήμα 4.127 ο Client χωρίς Interpreter

```
public interface Expression {
    public void Interpret(Context context);
}
```

```
public class NonTerminalExpression1 implements Expression{
    private Expression ex1;
    private Expression ex2;
    NonTerminalExpression1(Expression ex1){
    this.ex1 =ex1;
    }
    public void Interpret(Context context){
        System.out.println("TerminalExpression,Interpret");
        ex1.Interpret(context);
    }}

public class NonTerminalExpression2 implements Expression {
    private Expression ex1;

    NonTerminalExpression2(NonTerminalExpression1 ex1){
    this.ex1=ex1;
   }
    public void Interpret(Context context){
        System.out.println("NonTerminalExpression,Interpret");
        ex1.Interpret(context);
    }}

public class TerminalExpression implements Expression {
       public void Interpret(Context context){
        System.out.println("TerminalExpression,Interpret");
       }}
```

Σχήμα 4.128 οι Εκφράσεις με Χρήση Interpreter

```
public class Client {
    private Expression ex;
    public void anOperation(Context context){
        ex.Interpret(context);
```

```
}}
```

Σχήμα 4.129 ο Client με Χρήση Interpreter

Πίνακας 4.23 οι Μεταβολές των Μετρικών με και χωρίς το Interpreter

| | CBO | | WMC | | LCOM | | RFC | | NOC | | DIT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client | 4 | 1 | 3 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 0 |
| Context | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Nonterminal Expression1 | 2 | 0 | 3 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 1 |
| Nonterminal Expression2 | 2 | 0 | 3 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 1 |
| TerminalExpression | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Expression | | 0 | | 0 | | 0 | | 0 | | 3 | 0 | 0 |
| Overall Mean | 1.6 | 0.16 | 1.8 | 0 | 0 | 0 | 2.8 | 1.3 | 0 | 0.5 | 0 | 0.5 |

## 4.3. Μεθοδολογία ανάλυσης των δεδομένων

### *4.3.1. Καθορισμός του συνόλου δεδομένων*

Βασικό ζήτημα για την ανάλυση της δομής και των σχέσεων μεταξύ των προτύπων είναι ο προσδιορισμός των δεδομένων που πρόκειται να χρησιμοποιηθούν, το τι αντιπροσωπεύουν για τα πρότυπα αυτά τα δεδομένα καθώς και ο ορισμός του τύπου που τα χαρακτηρίζει. Πιο συγκεκριμένα μια σωστή ανάλυση απαιτεί τα δεδομένα που θα επιλέξουμε να ανταποκρίνονται στο πρόβλημα που θα αντιμετωπίσει η διαδικασία της ανάλυσης [17].

Στα πλαίσια μιας διαδικασίας ανάκτησης αρχιτεκτονικής λογισμικού (software architecture recovery), μια πτυχή της οποίας δίνουν τα σχεδιαστικά πρότυπα [13], μπορούν να χρησιμοποιηθούν επίσημα και ανεπίσημα χαρακτηριστικά γνωρίσματα [3]. Επίσημα χαρακτηριστικά αποτελούν τα γνωρίσματα που αναφέρονται στον τύπο της σχεδίασης, όπως η συνεκτικότητα, η κληρονομικότητα και η κλήση μεθόδων από μια κλάση. Ενώ ως ανεπίσημα ορίζονται χαρακτηριστικά όπως οι γραμμές και τα σχόλια του κώδικα και τα ονόματα των προγραμματιστών. Όπως προαναφέραμε, τα σχεδιαστικά πρότυπα χαρακτηρίζονται από ένα σύνολο μετρικών. Οι μετρικές αυτές περιγράφουν τις σχέσεις που υπάρχουν μεταξύ των δομικών μονάδων ενός

συστήματος. Συνεπώς η χρήση των μετρικών στα πλαίσια της ανάλυση των σχεδιαστικών προτύπων καλύπτει τον στόχο της ομαδοποίησης των προτύπων βάσει της προσφοράς τους στο σύστημα.

Αναφορικά με τον τύπο των χαρακτηριστικών, θα χρησιμοποιηθούν κατηγορικές μεταβλητές. Σκοπός της ανάλυσης είναι ο εντοπισμός της τάσης που παρουσιάζει μια μετρική βάσει της χρήσης κάθε προτύπου. Πιο συγκεκριμένα το ζητούμενο είναι αν η χρήση του κάθε προτύπου προκαλεί αύξηση, μείωση ή αφήνει αμετάβλητη κάθε μετρική. Παράλληλα επειδή εξετάζουμε την συνεισφορά του προτύπου στο σύστημα συνολικά χρησιμοποιούμε ως μέτρο το μέσο όρο της κάθε μετρικής στο σύστημα. Ο πίνακας 4.24 δίνει μια συνολική εικόνα των προτύπων και των τάσεων που παρουσιάζουν αναφορικά με τη μέση τιμή της κάθε μετρικής στο σύστημα.

Πίνακας 4.24 τα Πρότυπα και οι Τάσεις των Μετρικών

| | CBO | WMC | LCOM | RFC | NOC | DIT |
|---|---|---|---|---|---|---|
| Abstract Factory | down | down | same | up | up | up |
| Builder | down | down | same | up | up | up |
| Factory Method | down | down | same | up | up | up |
| Prototype | down | down | up | up | same | same |
| Singleton | same | down | up | up | same | same |
| Adapter | up | down | same | down | up | up |
| Bridge | same | same | same | up | down | down |
| Composite | down | down | same | down | up | up |
| Decorator | same | same | same | down | down | down |
| Façade | up | down | same | up | same | same |
| Flyweight | up | up | same | down | up | up |
| Proxy | up | down | same | up | up | up |
| Chain of Responsibility | down | same | same | up | same | same |
| Command | down | down | same | down | up | up |
| Iterator | down | down | same | down | up | up |
| Mediator | down | same | same | down | up | up |

| | | | | | | |
|---|---|---|---|---|---|---|
| Memento | up | same | down | up | same | same |
| Observer | down | same | same | down | up | up |
| State | same | down | same | up | up | up |
| Strategy | same | down | same | same | up | up |
| Template Method | same | down | same | up | up | up |
| Visitor | same | same | down | up | up | up |
| Interpreter | down | down | same | down | up | up |

*4.3.2. Συσταδοποίηση των προτύπων*

Πέρα από τον καθορισμό των δεδομένων που θα χρησιμοποιήσουμε θα πρέπει να καθοριστεί και ο τρόπος με τον οποίο θα γίνει η συσταδοποίηση των στοιχείων με σκοπό να αναδειχθούν οι κατηγορίες προτύπων με κοινά χαρακτηριστικά. Έτσι λοιπόν για την περίπτωση μας δημιουργήσαμε έναν ιεραρχικό αλγόριθμο. Η χρήση ενός ιεραρχικού αλγορίθμου στην διαδικασία της συσταδοποίησης προσφέρει μια πολυεπίπεδη εικόνα του τρόπου με τον οποίον μπορούν να οργανωθούν τα σχεδιαστικά πρότυπα. Επιπλέον η χρήση ενός τέτοιου αλγορίθμου, αποδεσμεύει την διαδικασία από τον ορισμό του αριθμού των ομάδων.

Όπως φαίνεται και στο σχήμα 4.130 μια διαδικασία ιεραρχικής συσταδοποίησης αποτελείται από ένα σύνολο βημάτων. Αρχικά υπάρχει ο καθορισμός των οντοτήτων και των διανυσμάτων. Στην περίπτωση που εξετάζουμε οι οντότητες της διαδικασίας είναι τα πρότυπα ενώ τα διανύσματα αποτελούνται από τις τιμές των μετρικών. Κάθε πρότυπο αντιπροσωπεύεται από ένα διάνυσμα μετρικών, ενώ οι συνιστώσες των διανυσμάτων μπορούν να πάρουν τρεις διακριτές τιμές. Επόμενο βήμα είναι ο υπολογισμός της ομοιότητας κάθε ζεύγους διανυσμάτων η οποία βασίζεται σε ένα προκαθορισμένο κριτήριο ομοιότητας και στην συνέχεια ακολουθεί η ομαδοποίηση των δυο πιο όμοιων προτύπων. Πέρα από την ομαδοποίηση των προτύπων δημιουργείται και ένα διάνυσμα αντιπροσωπευτικό για την νέα αυτή ομάδα. Τέλος υπολογίζεται η ομοιότητα μεταξύ του νέου διανύσματος με τα υπόλοιπα διανύσματα του συστήματος. Η διαδικασία αυτή επαναλαμβάνεται μέχρι όλες οι συστάδες των προτύπων να ομαδοποιηθούν σε μια. Είναι πολύ πιθανόν σε κάποιο σημείο της διαδικασίας να εμφανιστούν περισσότερα από δυο πρότυπα ή συστάδες προτύπων με

τον ίδιο βαθμό ομοιότητας. Σε αυτή την περίπτωση ο αλγόριθμός κάνει μια αυθαίρετη επιλογή των δυο οντοτήτων που θα ομαδοποιήσει.

/* $1^{η}$ Φάση – Προετοιμασία */

1. Συσχέτιση κάθε σχεδιαστικού προτύπου $P_i$ με ένα διάνυσμα μετρικών $M_{ij}$ το οποίο έχει $m_{i1},m_{i2},\ldots,m_{i6}$ διαστάσεις
2. Υπολογισμός της ομοιότητας μεταξύ κάθε ζευγαριού σχεδιαστικών προτύπων.

/* $2^{η}$ Φάση – Επαναλήψεις */

3. Repeat
   a. Ομαδοποίηση των δυο πιο όμοιων προτύπων $M_x$ και $M_y$.
   b. Δημιουργία ενός νέου διανύσματος $M_{xy}$ αντιπροσώπου των διανυσμάτων $M_x$ και $M_y$.
   c. Υπολογισμός της ομοιότητας μεταξύ του νέου διανύσματος και των υπόλοιπων.

   Until

   ο αριθμός των ομάδων να γίνει ίσος με ένα

Σχήμα 4.130 τα Βήματα ενός Ιεραρχικού Αλγορίθμου Συσταδοποίησης

Πέρα όμως από τον ορισμό της διαδικασίας, είναι απαραίτητος ο προσδιορισμός του κριτηρίου ομοιότητας. Στο σχήμα 4.131 φαίνεται ο τρόπος υπολογισμού της ομοιότητας που έχουμε επιλέξει. Η ομοιότητα βασίζεται στο πόσες μετρικές του προτύπου παρουσιάζουν βελτίωση και πόσες επιβάρυνση αναφορικά με την χρήση τους σε ένα σύστημα, ενώ στη μέτρηση δεν υπολογίζονται αυτές που παραμένουν αμετάβλητες. Ακόμα η διαδικασία υπολογισμού ομοιότητας χρησιμοποιεί βάρη και με αυτόν τον τρόπο η διαδικασία ευνοεί τις μετρικές που παρουσιάζουν τιμή down, δηλαδή τις μετρικές που παρουσιάζουν βελτίωση. Το βάρος για τις μετρικές που έχουν τιμή down είναι 5/6 ενώ για την τιμή up είναι 1/6. Έτσι λοιπόν σε κάθε περίπτωση ακόμα και αν το πλήθος των μετρικών που έχουν ομοιότητα στην τιμή up υπερτερεί σε σχέση με το πλήθος που έχουν τιμή down, η ομοιότητα διαμορφώνεται πρωτίστως από τις μετρικές του παρουσιάζουν ομοιότητα στην τιμή down και δευτερευόντως από αυτές που παρουσιάζουν ομοιότητα up.

```
1.    int Sim(Pi,Pj ){
2.      Wdown = 5/6;
3.      Wup = 1/6
4.      for (x = 0; x<6;x++ )
5.          if (Pi[x] == Pj[x] && Pi[x] ==”down” ) ++ downSum
6.          else if (Pi[x]==Pj[x]&& Pi[x] ==”up” )  ++ upSum
7.       return (downSum*Wdown + upSum*Wup)/6;  }
```

Σχήμα 4.131 Υπολογισμός Ομοιότητας μεταξύ Προτύπων

Τέλος πρέπει να ορισθεί και ο τρόπος εξαγωγής του αντιπροσωπευτικού διανύσματος για κάθε νέα συστάδα που δημιουργεί το σύστημα. Το σχήμα 4.131 δείχνει την διαδικασία δημιουργίας του διανύσματος αντιπροσώπου κάθε νέας συστάδας. Σύμφωνα με αυτήν την διαδικασία αν οι δυο οντότητες που έχουν ομαδοποιηθεί παρουσιάζουν ίδια τιμή σε μια συγκεκριμένη μετρική, τότε το διάνυσμα αντιπρόσωπος θα έχει την ίδια τιμή για την μετρική αυτή. Στην περίπτωση όμως που δεν παρουσιάζουν ίδια τιμή, τότε το διάνυσμα αντιπρόσωπος και για την μετρική αυτή θα έχει ως τιμή *, αυτό σημαίνει ότι η συγκεκριμένη συστάδα δεν παρέχει καμία πληροφορία για την τάση αυτής της μετρικής.

```
1.    P newEntity(Pi,Pj){
2.      for (x = 0; x<6;x++ )
3.          if (Pi[x] == Pj[x]) P[x] = Pi [x];
4.          else P[x] =”*”
5.    return P;}
```

Σχήμα 4.132 η Δημιουργία του Αντιπροσωπευτικού Διανύσματος

*4.3.3. Συνδυαστική ανάλυση*

Για την ανάλυση των δεδομένων θα γίνει χρήση και της συνδυαστικής ανάλυσης με σκοπό την ανακάλυψη σχέσεων που υπάρχουν μεταξύ των τιμών που μπορούν να πάρουν οι μετρικές. Οι αποκαλυπτόμενες σχέσεις που προκύπτουν μέσω της

συνδυαστικής ανάλυσης, μπορούν να εκφραστούν ως κανόνες συσχέτισης ή ως σύνολα συχνά εμφανιζόμενων στοιχείων.

Αναφορικά με το σύνολο των στοιχείων, στην περίπτωση που υπάρχει ένα υπερσύνολο στοιχείων $I = \{i_1, i_2, \ldots i_d\}$ και ένα σύνολο συναλλαγών $T = \{t_1, t_2, \ldots t_N\}$ τότε κάθε συναλλαγή $t_i$ θα περιέχει ένα υποσύνολο από στοιχεία που ανήκουν στο I. Στην περίπτωση που εξετάζουμε τις μετρικές του συστήματος με τις διακριτές τιμές που μπορούν να πάρουν, αποτελούν το υπερσύνολο των στοιχείων ενώ τα σχεδιαστικά πρότυπα αποτελούν το σύνολο των συναλλαγών. Συχνό σύνολο στοιχείων χαρακτηρίζεται το σύνολο εκείνο που εμφανίζεται σε έναν προκαθορισμένο αριθμό συναλλαγών. Έτσι λοιπόν το ζητούμενο είναι να βρούμε σύνολα μετρικών με συγκεκριμένες τιμές που εμφανίζονται σε περισσότερα του ενός προτύπων.

Για την εύρεση των συχνών συνόλων στοιχείων θα γίνει χρήση του αλγορίθμου Apriori [1]. Η εύρεση των συχνών συνόλων γίνεται βάσει του υπολογισμού της υποστήριξης (support) συνόλου. Η υποστήριξη φανερώνει πόσο συχνά εμφανίζεται ένα συγκεκριμένο υποσύνολο στοιχείων στις συναλλαγές. Ενώ η βασική αρχή του συγκεκριμένου αλγορίθμου είναι ότι κάθε υποσύνολο ενός συχνού συνόλου, είναι κι αυτό με την σειρά του συχνό σύνολο. Η χρήση του συγκεκριμένου αλγόριθμου στο πρόβλημα μας, θα αναδείξει συχνά σύνολα τιμών που παρουσιάζουν οι μετρικές στο σύνολο των προτύπων του εξετάζουμε.

Ο αλγόριθμος Apriori αρχικά διαβάζει τον πίνακα των προτύπων και μετράει την υποστήριξη των συχνών συνόλων μεγέθους ένα (1-itemsets) και βρίσκει πια από αυτά ξεπερνούν το κατώφλι ελάχιστης υποστήριξη που έχουμε θέσει. Σε κάθε επόμενο βήμα, γίνεται χρήση των συχνών συνόλων του προηγούμενου βήματος, με σκοπό την δημιουργία νέων συνόλων. Τα νέα αυτά σύνολα αποτελούν τα υποψήφια συχνά σύνολα (candidate itemsets) και αυτό γιατί δεν γνωρίζουμε την υποστήριξη που έχουν και κατ’ επέκταση το αν είναι συχνά. Για τον λόγο αυτό μετριέται η υποστήριξη μέσω ενός περάσματος από τον αρχικό πίνακα. Ενώ στο τέλος του κάθε βήματος αποφασίζεται ποια σύνολα είναι συχνά και αυτά χρησιμοποιούνται στο επόμενο βήμα.

Στο σχήμα 4.133 φαίνεται ο αλγόριθμος Apriori. Αρχικά γίνεται ένα πέρασμα τον πίνακα των προτύπων P και εντοπίζονται τα συχνά σύνολα μεγέθους ένα (1-itemsets) που ξεπερνάνε το κατώφλι υποστήριξη (minsup) που έχουμε ορίσει (εντολή 1). Σε κάθε επόμενο πέρασμα (εντολή 2), εκτελούνται δύο φάσεις. Σε πρώτη φάση γίνεται η εύρεση των υποψήφιων συχνών συνόλων. Η πρώτη φάση αφορά την παραγωγή υποψήφιων συνόλων μεγέθους k από τα συχνά σύνολα (k-1) τα συχνά σύνολα δηλαδή που βρέθηκαν στην προηγούμενη φάση. Ενώ την διαδικασία αυτή αναλαμβάνει η συνάρτηση apriori-gen (εντολή 3). Η δεύτερη φάση αφορά τον υπολογισμό του support count για τα υποψήφια συχνά σύνολα. Πιο συγκεκριμένα, για κάθε πρότυπο (εντολή 4) βρίσκονται τα υποψήφια συχνά σύνολα που περιέχονται σ' αυτό (εντολή 5) και ο μετρητής αυξάνεται κατά 1 (εντολή 7). Η συνάρτηση subset (εντολή 5) υπολογίζει το σύνολο $C_t$, δηλαδή υπολογίζει τα υποψήφια συχνά σύνολα που περιέχονται στο πρότυπο p. Ενώ στο τέλος του περάσματος υπολογίζεται στο σύνολο $L_k$ (εντολή 9) και απορρίπτει τα σύνολα του $C_k$ τα οποία δεν είναι συχνά. Τέλος (εντολή 11) ο αλγόριθμος επιστρέφει όλα τα συχνά σύνολα τιμών που παρουσιάζουν οι μετρικές των προτύπων.

| | |
|---|---|
| 1. | $L_1$ = {large 1-itemsets} |
| 2. | for (k=2; $L_{k-1}\neq\emptyset$; k++) do begin |
| 3. | $C_k$ = apriori-gen($L_{k-1}$); |
| 4. | forall patterns $p\in P$ do begin |
| 5. | $C_t$ = subset ($C_k$,t); |
| 6. | For each candidates $c\in C_t$ do |
| 7. | c.count++; |
| 8. | end |
| 9. | $L_k$ = {$c\in C_t$\|c.count≥minsup} |
| 10. | end |
| 11. | return $\cup_k L_k$; |

Σχήμα 4.133 ο Αλγόριθμος Apriori

Η συνάρτηση apriori-gen στην γραμμή τέσσερα παίρνει παράμετρο το συχνό σύνολο στοιχείων που υπολογίστηκε στο προηγούμενο πέρασμα ($F_{k-1}$) και επιστρέφει ένα υπερσύνολο του συχνού συνόλου στοιχείων του k-οστού σταδίου. Αντίθετα, η συνάρτηση subset δέχεται ως παράμετρο το σύνολο υποψήφιων συνόλων αντικειμένων και επιστρέφει τα σύνολα των στοιχείων που εμφανίζονται στην συγκεκριμένη συναλλαγή.

# ΚΕΦΑΛΑΙΟ 5. ΑΠΟΤΕΛΕΣΜΑΤΑ



## 5.1. Κατηγοριοποίηση

Σε αυτήν την ενότητα παρουσιάζουμε τα αποτελέσματα της ιεραρχικής ανάλυσης των δεδομένων. Σκοπός είναι η παρουσίαση όλης της διαδικασίας της ανάλυσης που οδηγεί στην δημιουργία μιας νέας κατηγοριοποίησης για τα πρότυπα που κατέγραψε η GoF. Ο πίνακας 5.1 δείχνει λεπτομερώς την διαδικασία ομαδοποίησης των προτύπων. Πιο συγκεκριμένα η πρώτη στήλη δηλώνει την επανάληψη στην οποία είναι ο αλγόριθμός, στην συνέχεια ακολουθεί μια στήλη στην οποία φαίνονται τα ονόματα των προτύπων που έχουν ομαδοποιηθεί καθώς και το όνομα της νέας ομάδας που δημιουργούν. Ακόμα υπάρχει και μια στήλη για την παρουσίαση της ομοιότητας μεταξύ των εμπλεκόμενων προτύπων και ομάδων. Τέλος για την κάθε επανάληψη και την δημιουργία μιας νέας ομάδας προτύπων, υπάρχει μια αναφορά στην επόμενη επανάληψη που θα εμπλακεί η συγκεκριμένη ομάδα.

Τα πρότυπα Composite, Command, Iterator και Interpreter παρουσιάζουν την μεγαλύτερη ομοιότητα και ομαδοποιούνται πρώτα κατά την διάρκεια της εκτέλεσης. Ανάλογα, τα πρότυπα Abstract Factory, Builder και Factory Method, παρουσιάζουν την δεύτερη μεγαλύτερη ομοιότητα και ομαδοποιούνται στην τέταρτη και πέμπτη επανάληψη.

Πίνακας 5.1 Υπολογισμός της Ομοιότητας

| Iteration | Cluster Combined | | | Similarity | Next Iteration |
|---|---|---|---|---|---|
| | Cluster 1 | Cluster 2 | Name | | |
| 1 | Composite | Command | NOD1 | 0.472 | 3 |
| 2 | Iterator | Interpreter | NOD2 | 0.472 | 3 |
| 3 | NOD1 | NOD2 | NOD3 | 0.472 | 6 |
| 4 | Abstract Factory | Builder | NOD4 | 0.361 | 5 |
| 5 | Factory Method | NOD4 | NOD5 | 0.361 | 8 |
| 6 | Adapter | NOD3 | NOD6 | 0.333 | 12 |
| 7 | Mediator | Observer | NOD7 | 0.333 | 14 |
| 8 | Prototype | NOD5 | NOD8 | 0.305 | 16 |
| 9 | Bridge | Decorator | NOD9 | 0.277 | 20 |
| 10 | Proxy | State | NOD10 | 0.222 | 11 |
| 11 | Template Method | NOD10 | NOD11 | 0.222 | 13 |
| 12 | Flyweight | NOD6 | NOD12 | 0.194 | 14 |
| 13 | Strategy | NOD11 | NOD13 | 0.194 | 18 |
| 14 | NOD7 | NOD12 | NOD14 | 0.194 | 20 |
| 15 | Singleton | Façade | NOD15 | 0.166 | 18 |
| 16 | Chain of Responsibility | NOD8 | NOD16 | 0.166 | 19 |
| 17 | Memento | Visitor | NOD17 | 0.166 | 19 |
| 18 | NOD13 | NOD15 | NOD18 | 0.138 | 21 |
| 19 | NOD16 | NOD17 | NOD19 | 0.027 | 21 |
| 20 | NOD9 | NOD14 | NOD20 | 0.0 | 22 |
| 21 | NOD18 | NOD19 | NOD21 | 0.0 | 22 |
| 22 | NOD20 | NOD21 | NOD22 | 0.0 | - |

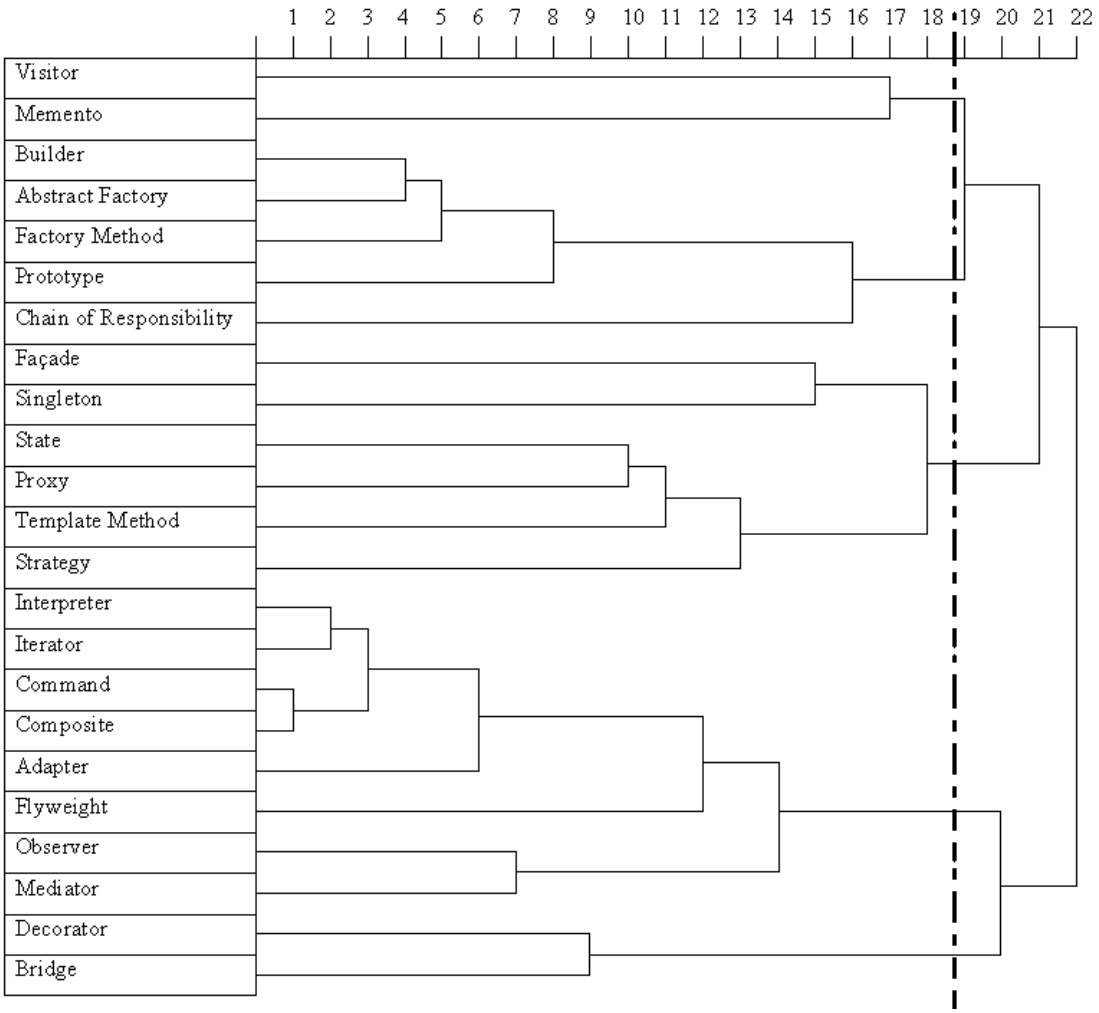
Αναφορικά με τις ομαδοποιήσεις των προτύπων από την εικοστή έως την εικοστή δεύτερη εκτέλεση παρατηρούμε ότι η ομοιότητα τον προτύπων έχει τιμή μηδέν. Αυτό σημαίνει ότι δεν υπάρχει κάποιο κοινό στοιχείο το οποίο να οδηγεί στην συγχώνευση των συγκεκριμένων ομάδων. Συνεπώς δεν εξετάζουμε το αποτέλεσμα της διαδικασίας στις τρείς τελευταίες επαναλήψεις για να μην οδηγηθούμε στην δημιουργία ομάδων που δεν χαρακτηρίζονται από κοινά στοιχεία.

Πέραν όμως από την διαδικασία ομαδοποίησης των προτύπων, ενδιαφέρον παρουσιάζουν και τα διανύσματα μετρικών που χαρακτηρίζουν τις ομάδες που δημιουργεί η διαδικασία σε κάθε βήμα εκτέλεσης. Ο πίνακας 5.2 παρουσιάζει τα ονόματα των ομάδων με το αντίστοιχο διάνυσμα που χαρακτηρίζει το κάθε ένα από αυτά.

Πίνακας 5.2 τα Διανύσματα των Ομάδων

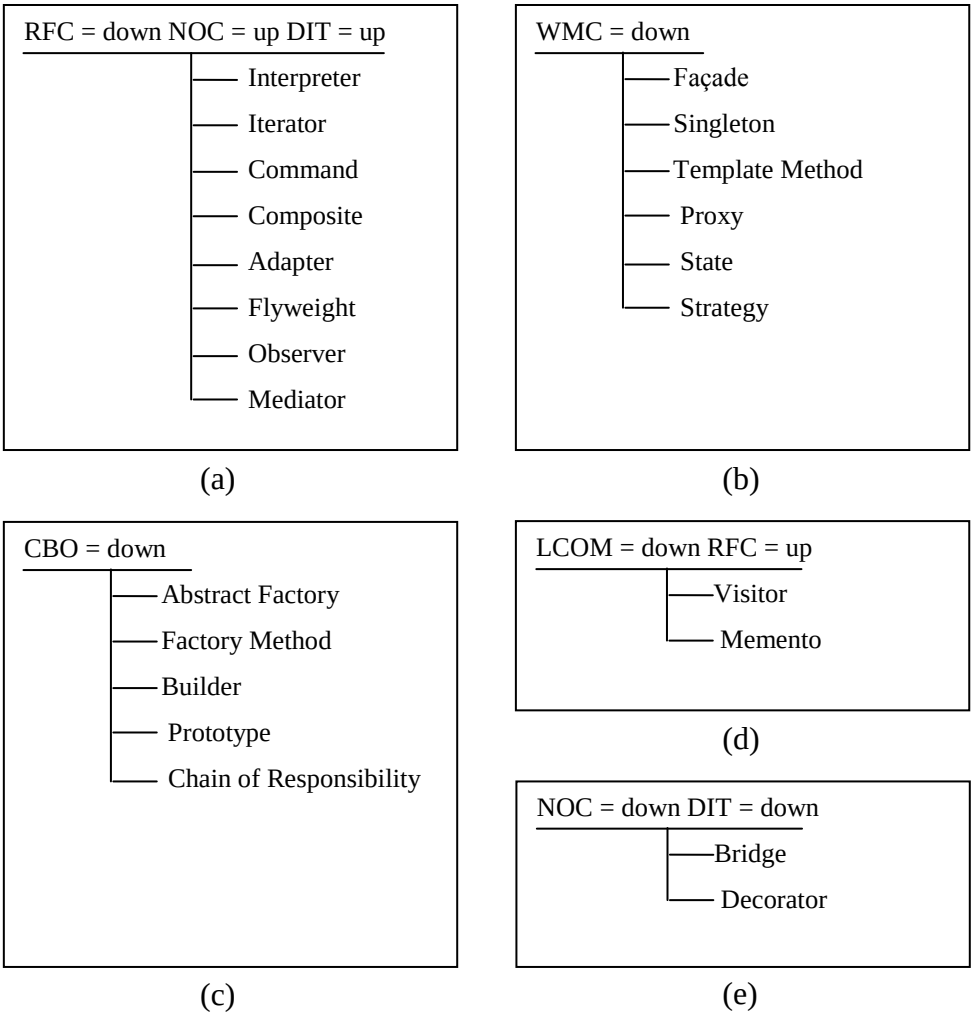
| Cluster Combined | New Metrics Vector | | | | | |
|---|---|---|---|---|---|---|
| | CBO | WMC | LCOM | RFC | NOC | DIT |
| NOD1 | down | down | same | down | up | up |
| NOD2 | down | down | same | down | up | up |
| NOD3 | down | down | same | down | up | up |
| NOD4 | down | down | same | up | up | up |
| NOD5 | down | down | same | up | up | up |
| NOD6 | * | down | same | down | up | up |
| NOD7 | down | same | same | down | up | up |
| NOD8 | down | down | * | up | * | * |
| NOD9 | same | same | same | * | down | down |
| NOD10 | * | down | same | up | up | up |
| NOD11 | * | down | same | up | up | up |
| NOD12 | * | * | same | down | up | up |
| NOD13 | * | down | same | * | up | up |
| NOD14 | * | * | same | down | up | up |
| NOD15 | * | down | * | up | same | same |
| NOD16 | down | * | * | up | * | * |
| NOD17 | * | same | down | up | * | * |
| NOD18 | * | down | * | * | * | * |
| NOD19 | * | * | * | up | | |
| NOD20 | * | * | same | * | * | * |
| NOD21 | * | * | * | * | * | * |
| NOD22 | * | * | * | * | * | * |

Τα διανύσματα των μετρικών που χαρακτηρίζουν τις ομάδες έχουν σημασία για την διαδικασία γιατί μέσω αυτών μπορεί να γίνει κατανοητό ποια είναι τα κοινά χαρακτηριστικά που έχουν τα πρότυπα της κάθε ομάδας. Όσο προχωράει η διαδικασία τόσο μειώνεται και η πληροφορία που χαρακτηρίζει την κάθε ομάδα. Πιο συγκεκριμένα οι ομάδες NOD16 έως NOD22 διαθέτουν πληροφορίες σε λιγότερες από τις μισές μετρικές. Ακόμα η ομάδα NOD19 παρέχει πληροφορίες μόνο για την μετρική RFC συμφώνα με την οποία η συγκεκριμένη ομάδα προτύπων παρουσιάζει αύξηση στην μετρική RFC. Όμως στόχος είναι η δημιουργία ομάδων που χαρακτηρίζονται από την βελτίωση κάποιας μετρικής. Συνεπώς η διαδικασία ομαδοποίησης θα σταματήσει πριν την δεκάτη ένατη επανάληψη. Το σχήμα 5.1 δείχνει την διαδικασία της εκτέλεσης του αλγορίθμου.




Σχήμα 5.1 το Δενδρόγραμμα της Ιεραρχικής Ομαδοποίησης.

Ο τερματισμός της εκτέλεσης του αλγορίθμου πριν την δέκατη ένατη επανάληψη οδηγεί στην δημιουργία πέντε διαφορετικών ομάδων. Η μεγαλύτερη ομάδα αποτελείται από οκτώ πρότυπα ενώ υπάρχουν και δυο ομάδες που περιέχουν από δυο πρότυπα η κάθε μια. Στο σχήμα 5.2 φαίνονται οι ομάδες καθώς και η τιμή της μετρικής που χαρακτηρίζει την κάθε ομάδα.




Σχήμα 5.2 η Κατηγοριοποίηση των Προτύπων.

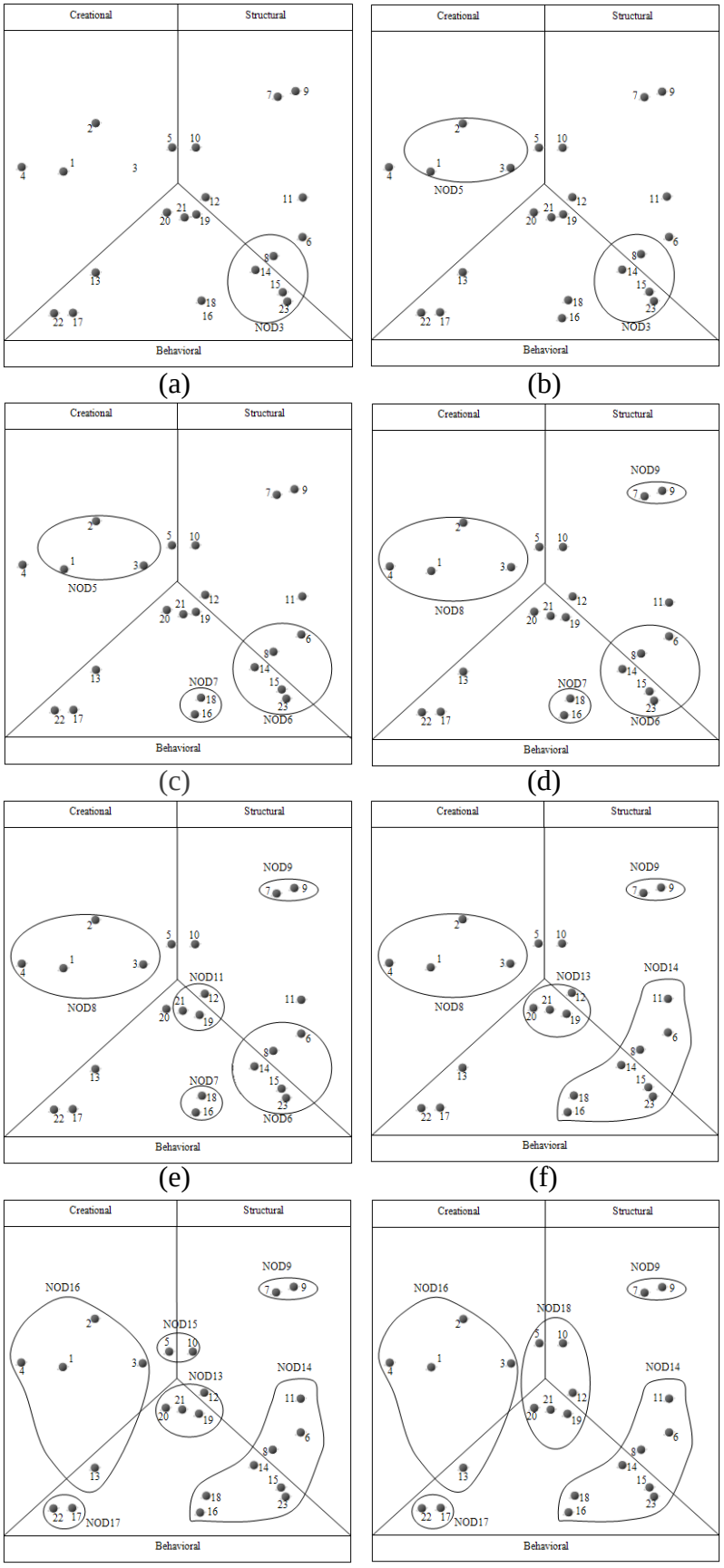
Το βασικό χαρακτηριστικό που έχει η κατηγορία που φαίνεται στο σχήμα 5.2 a είναι η βελτίωση της μετρικής RFC. Πιο συγκεκριμένα τα πρότυπα που ανήκουν σε αυτήν την κατηγορία παρέχουν τρόπους για την ομοιόμορφη πρόσβαση σε μια λειτουργία, που υλοποιεί ένα σύνολο κλάσεων. Με αυτόν τον τρόπο απλοποιείται η επικοινωνία των κλάσεων που παρέχουν την λειτουργία και της κλάσης που ενδιαφέρεται για την

λειτουργία αυτή. Τα πρότυπα αυτά χρησιμοποιούν ένα στοιχείου αφαίρεσης μέσω του οποίου πραγματοποιείται η επικοινωνία. Η χρήση ενός τέτοιου στοιχείου οδηγεί στην αύξηση των μετρικών DIT και NOC. Βάσει λοιπόν αυτών των χαρακτηριστικών μπορούμε να πούμε ότι η συγκεκριμένη ομάδα υποστηρίζει σενάρια συντήρησης με κλειστό τρόπο. Δηλαδή κάθε νέα κλάση που εισάγουμε στο σύστημα για την υποστήριξη μιας λειτουργίας υλοποιεί το στοιχείο αφαίρεσης. Ακόμα η κάθε κλάση που ενδιαφέρεται για τις λειτουργίες συσχετίζεται μόνο με το στοιχείο αφαίρεσης, και έτσι δεν χρειάζονται αλλαγές στον κώδικα του εξαιτίας του σεναρίου συντήρησης που πραγματοποιήθηκε.

Το σχήμα 5.2 b παρουσιάζει την κατηγορία που μειώνει την μετρική WMC. Δηλαδή τα πρότυπα που ανήκουν σε αυτήν την κατηγορία προσφέρουν λύσεις για την βελτίωση της πολυπλοκότητας ενός συστήματος. Η βελτίωση της πολυπλοκότητας οφείλεται στην μείωση των δομών ελέγχου. Για παράδειγμα τα πρότυπα Singleton και Proxy ανεξαρτητοποιούν τον Client από ελέγχους για την δημιουργία ή την προσπέλαση ενός αντικειμένου. Ενώ τα πρότυπα Strategy και Template Method ανεξαρτητοποιούν μια κλάση από την επιλογή κάποιου αλγορίθμου ή κάποιου βήματος στην εκτέλεση του αλγορίθμου.

Στο σχήμα 5.2 c φαίνεται η κατηγορία προτύπων που βελτιώνει το CBO. Δηλαδή τα πρότυπα αυτής της κατηγορίας δίνουν λύση σε προβλήματα υψηλής ζεύξης μεταξύ δομικών μονάδων ενός συστήματος. Ακόμα στο σχήμα 5.2 d φαίνεται η κατηγορία των προτύπων που βελτιώνουν την μετρική LCOM. Πιο συγκεκριμένα τα πρότυπα Memento και Visitor παρέχουν λύσεις για την βελτίωση της συνεκτικότητας των κλάσεων ενός συστήματος. Όμως οι λύσεις αυτές επιβαρύνουν την μετρική RFC. Τέλος η κατηγορία του σχήματος 5.2 e περιλαμβάνει πρότυπα που βελτιώνουν τις μετρικές NOC και DIT ενώ η χρήση των προτύπων αυτών οδηγεί στην απλοποίηση των ιεραρχιών του συστήματος.

Περά όμως από την παρουσίαση της κατηγοριοποίησης, ενδιαφέρον έχει και η σύγκριση της με την κατηγοριοποίηση που πρότεινε η GoF. Το σχήμα 5.3 δίνει μια εικόνα των δυο αυτών κατηγοριοποιήσεων και των βημάτων που οδηγούν στην τελική μορφή.




## Creational

1 Abstract Factory
2 Builder
3 Factory Method
4 Prototype
5 Singleton

## Structural

6 Adapter
7 Bridge
8 Composite
9 Decorator
10 Façade
11 Flyweight
12 Proxy

## Behavioral

13 Chain of Responsibility
14 Command
15 Iterator
16 Mediator
17 Memento
18 Observer
19 State
20 Strategy
21 Template Method
22 Visitor
23 Interpreter

Σχήμα 5.3 οι Κατηγοριοποίηση Συγκριτικά με αυτή της GoF.

Οι δυο κατηγοριοποιήσεις παρουσιάζουνε διαφορές στον αριθμό των κατηγοριών. Η πρόταση της GoF υποστηρίζει τρεις διαφορετικές κατηγορίες, ενώ η κατηγοριοποίηση που βασίζεται στις μετρικές οδηγεί σε πέντε κατηγορίες. Επιπλέον όπως δείχνει το σχήμα 5.3 h υπάρχουν κατηγορίες που περιλαμβάνουν πρότυπα διαφορετικών κατηγοριών σύμφωνα με την GoF. Πιο συγκεκριμένα η κατηγορία που ορίζει η ομαδοποίηση NOD14 περιλαμβάνει πέντε Behavioral και τρία Structural πρότυπα, κάτι ανάλογο δείχνει και η ομαδοποίηση NOD18 η οποία περιλαμβάνει τρία Behavioral, ένα Creational και δυο Structural πρότυπα. Το γεγονός ότι Behavioral και Structural πρότυπα εμφανίζονται σε ίδιες κατηγορίες ενισχύει την άποψη που υποστηρίζει ότι ο τρόπος που χαρακτηρίζονται τα πρότυπα ως Structural και Behavioral δεν είναι ξεκάθαρος (Tichy,1997) [5]. Αντίθετα η ομαδοποίηση NOD9 περιλαμβάνει μόνο Structural πρότυπα και η ομαδοποίηση NOD17 αποτελείται από δυο Behavioral πρότυπα. Τέλος τα τέσσερα από τα πέντε Creational πρότυπα ομαδοποιούνται στο NOD16.

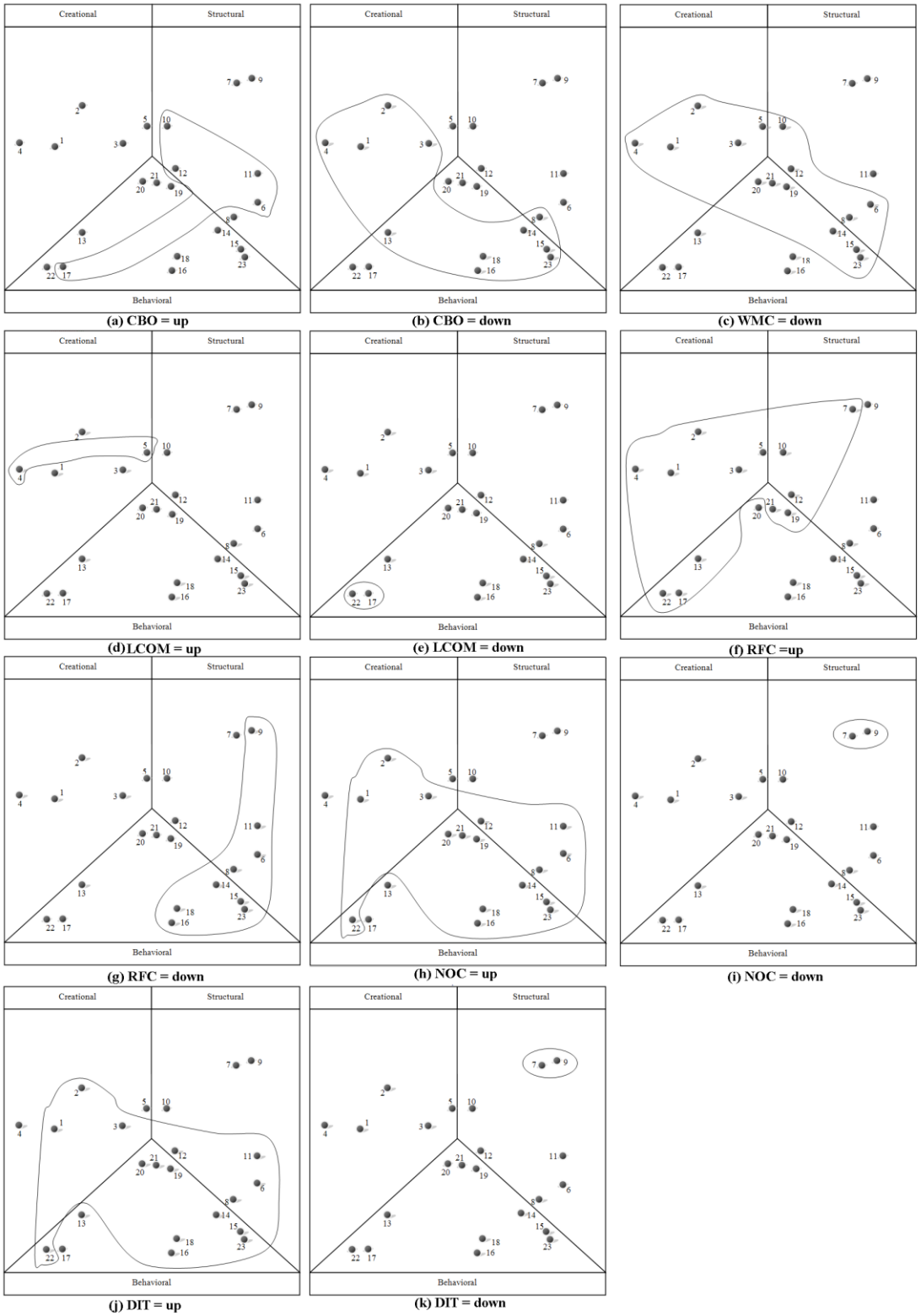
### 5.2. Συχνά σύνολα

Στην ενότητα αυτή παρουσιάζονται τα αποτελέσματα της συνδυαστικής ανάλυσης και πιο συγκεκριμένα δίνεται μια εικόνα των συχνών συνόλων και πως αυτά εμφανίζονται στα πρότυπα. Με αυτόν τον τρόπο μπορεί να γίνει μια πιο λεπτομερής παρουσίαση των ιδιοτήτων που έχουν κάποια πρότυπα ή κάποιες ομάδες προτύπων.

Συχνό σύνολο θεωρείται μια τιμή μιας συγκεκριμένης μετρικής που εμφανίζεται σε δύο ή περισσότερα πρότυπα. Η εκτέλεση του αλγορίθμου Apriori εμφάνισε εβδομήντα συχνά σύνολα, ενώ το μέγιστο μέγεθος των συνόλων αυτών περιλαμβάνει πέντε τιμές. Ακόμα με μέγεθος δυο και τριών τιμών, υπάρχουν από είκοσι τρία συχνά σύνολα. Αναλυτικά η παρουσίαση των συχνών συνόλων γίνεται στο παράρτημα Β ενώ στην συνέχεια ακολουθεί αναλυτική περιγραφή των συχνών συνόλων ταξινομημένα βάσει μεγέθους. Ενώ τα σχήματα βοηθούν στην σύγκριση των συχνών συνόλων με βάση την κατηγοριοποίηση της GoF.

Το σχήμα 5.4 δίνει μια εικόνα για τα συχνά σύνολα μεγέθους ένα, δηλαδή παρουσιάζει συχνά σύνολα τιμών που περιλαμβάνουν μόνο μια μετρική.

1 Abstract Factory 2 Builder 3 Factory Method 4 Prototype 5 Singleton 6 Adapter 7 Bridge 8 Composite 9 Decorator 10 Façade 11 Flyweight 12 Proxy 13 Chain of Responsibility 14 Command 15 Iterator 16 Mediator17 Memento 18 Observer 19 State 20 Strategy 21 Template Method 22 Visitor 23 Interpreter




Σχήμα 5.4 τα Συχνά Σύνολα Μεγέθους Ένα.

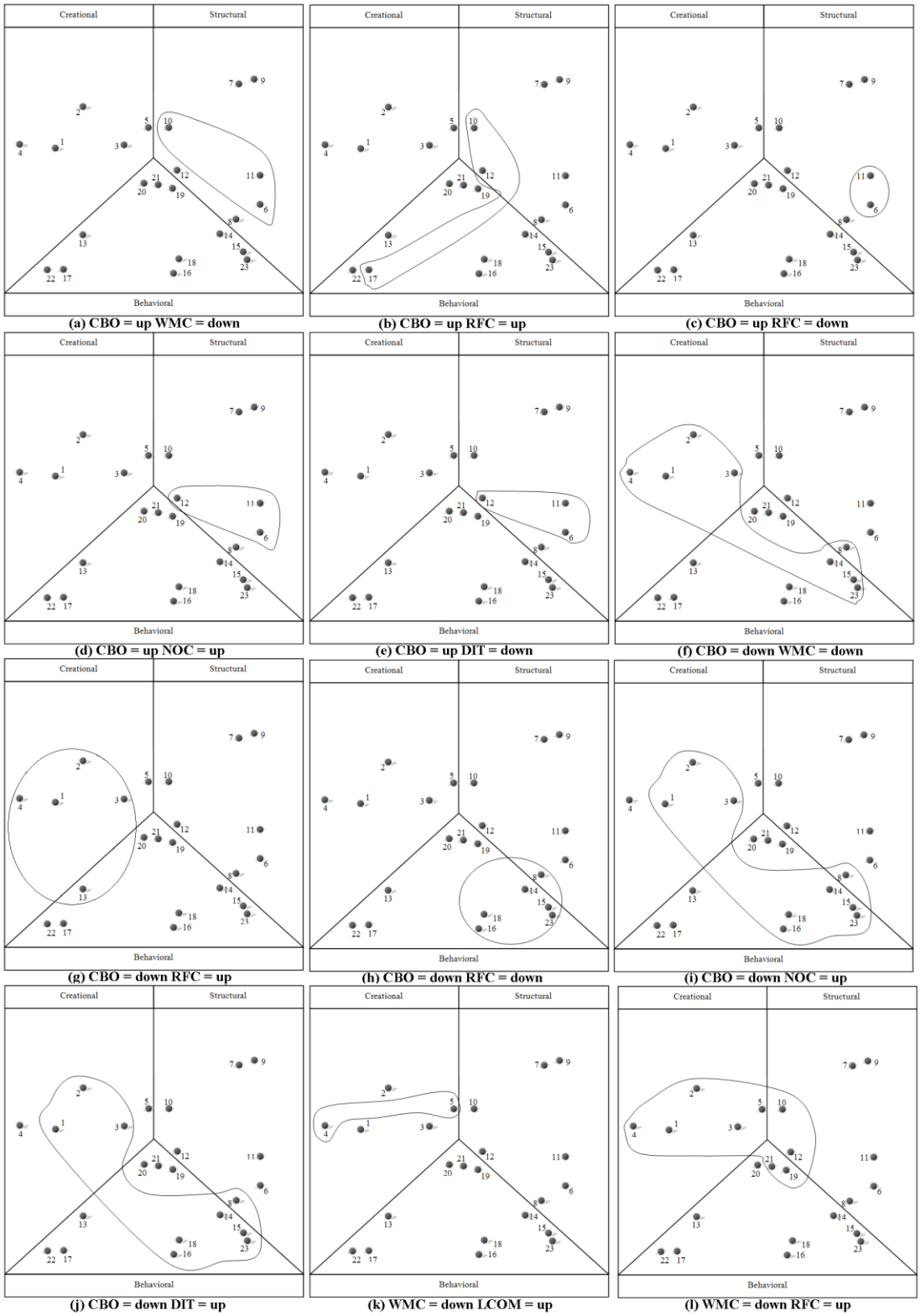
Σχεδόν τα μισά συχνά σύνολα μεγέθους ένα εμφανίζονται σε πρότυπά διαφορετικών κατηγόριων. Όπως φαίνεται και στα σχήματα 5.4 b,c,f,h και j τα συχνά σύνολα που παρουσιάζουν, περιλαμβάνουν πρότυπα που ανήκουν και στις τρείς κατηγορίες που έχει προτείνει η GoF. Αντίθετα παρατηρούνται τέσσερις περιπτώσεις όπου τα συχνά σύνολα περιλαμβάνουν πρότυπα μιας και μόνο κατηγορίας.

Το σχήμα 5.4 d παρουσιάζει το συχνό σύνολο LCOM = up. Το συγκεκριμένο συχνό σύνολο εντοπίζεται μόνο σε Creational πρότυπα και πιο συγκεκριμένα στο Prototype και στο Singleton. Επιπλέον το σχήμα 5.4 e δείχνει το συχνό σύνολο LCOM = down. Σε αυτήν την περίπτωση, τα πρότυπα που περιλαμβάνονται σε αυτό το σύνολο ανήκουν στην κατηγορία Behavioral πρότυπα και είναι το Visitor και το Memento. Ακόμα στο σχήμα 4.5 i, φαίνεται το συχνό σύνολό NOC = down το οποίο υποστηρίζεται από τα πρότυπα Decorator και Bridge, πρότυπα δηλαδή που ανήκουν στην κατηγορία Structural. Τέλος ανάλογη εικόνα παρουσιάζει και το σχήμα 4.5 k το οποίο παρουσιάζει το σχετικό σύνολο DIT = down το οποίο μπορούμε να το εντοπίσουμε επίσης στα πρότυπα Decorator και Bridge.

Ακόμα υπάρχουν και δυο συχνά σύνολα που υποστηρίζονται από Behavioral και Structural πρότυπα. Στο σχήμα 5.4 a φαίνεται το συχνό σύνολο CBO = up το οποίο μπορούμε να το εντοπίσουμε σε ένα υποσύνολό των Structural προτύπων και στο Memento, το οποίο είναι Behavioral πρότυπο. Ανάλογα και το συχνό υποσύνολο RFC = down το οποίο παρουσιάζεται στο σχήμα 5.4 g και περιλαμβάνει Behavioral και Structural πρότυπα.

Τέλος ενδιαφέρον παρουσιάζει μια συσχέτιση που αποκαλύπτεται από τα σχήματα 5.4 h και j. Παρατηρώντας τα σχήματα μπορούμε να διαπιστώσουμε ότι τα δύο αυτά συχνά σύνολα που χαρακτηρίζουν τις μετρικές DIT και NOC ως up αποτελούνται από τα ίδια ακριβώς πρότυπα. Ενώ το ίδιο συμβαίνει και με τα συχνά σύνολα των σχημάτων 5.4 i και k τα οποία χαρακτηρίζουν τις ίδιες μετρικές ως down. Το σχήμα 5.5 που ακολουθεί παρουσιάζει τα συχνά σύνολα μεγέθους δυο. Η εκτέλεση του αλγορίθμου ανέδειξε είκοσι τρία τέτοια συχνά σύνολα και αποτελεί την πιο μεγάλη ομάδα συχνών συνόλων.

1 Abstract Factory 2 Builder 3 Factory Method 4 Prototype 5 Singleton 6 Adapter 7 Bridge 8 Composite 9 Decorator 10 Façade 11 Flyweight 12 Proxy 13 Chain of Responsibility 14 Command 15 Iterator 16 Mediator17 Memento 18 Observer 19 State 20 Strategy 21 Template Method 22 Visitor 23 Interpreter



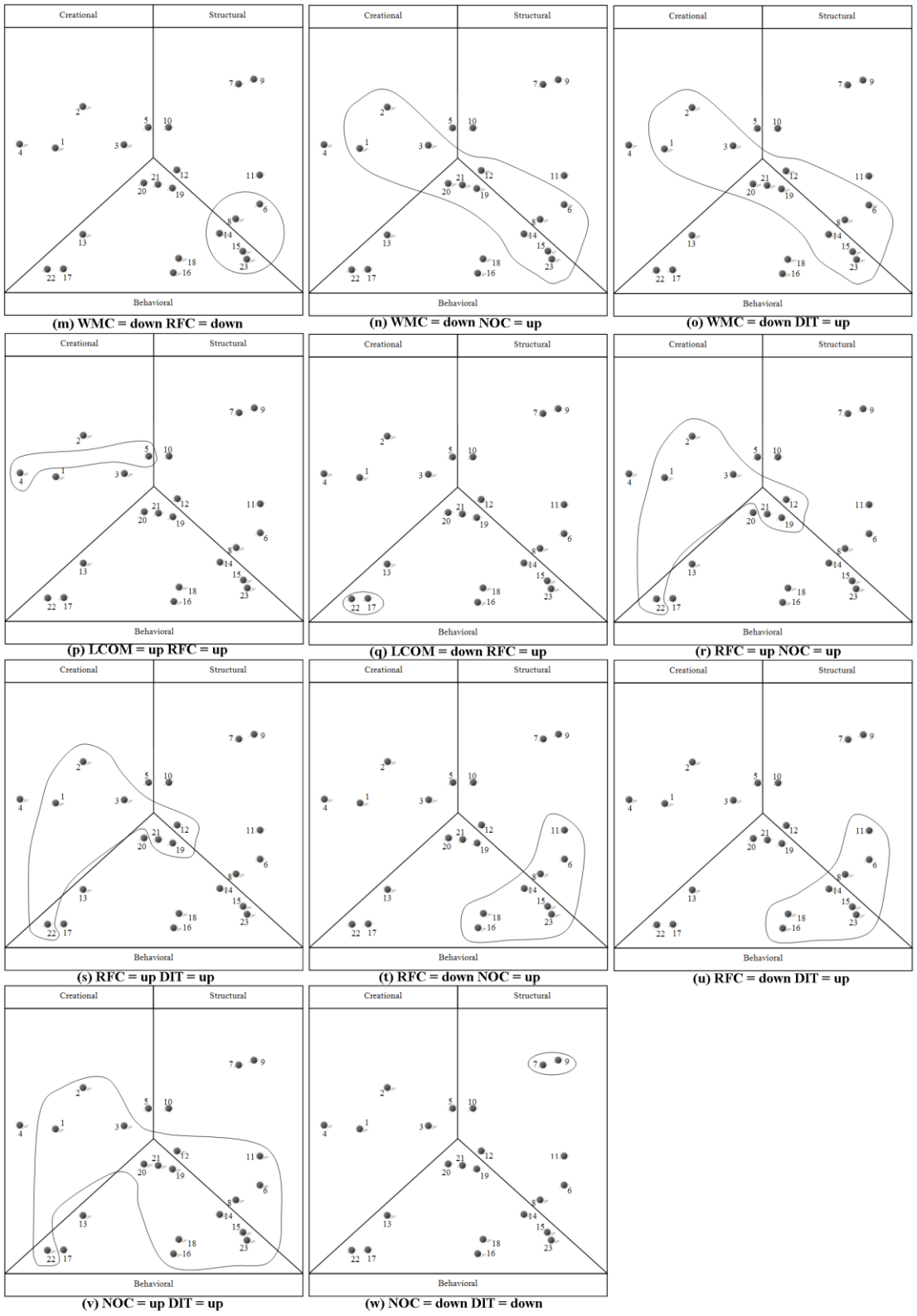

(a) CBO = up WMC = down (b) CBO = up RFC = up (c) CBO = up RFC = down

(d) CBO = up NOC = up (e) CBO = up DIT = down (f) CBO = down WMC = down

(g) CBO = down RFC = up (h) CBO = down RFC = down (i) CBO = down NOC = up

(j) CBO = down DIT = up (k) WMC = down LCOM = up (l) WMC = down RFC = up

1 Abstract Factory 2 Builder 3 Factory Method 4 Prototype 5 Singleton 6 Adapter 7 Bridge 8 Composite 9 Decorator 10 Façade 11 Flyweight 12 Proxy 13 Chain of Responsibility 14 Command 15 Iterator 16 Mediator17 Memento 18 Observer 19 State 20 Strategy 21 Template Method 22 Visitor 23 Interpreter




(m) WMC = down RFC = down (n) WMC = down NOC = up (o) WMC = down DIT = up

(p) LCOM = up RFC = up (q) LCOM = down RFC = up (r) RFC = up NOC = up

(s) RFC = up DIT = up (t) RFC = down NOC = up (u) RFC = down DIT = up

(v) NOC = up DIT = up (w) NOC = down DIT = down

Σχήμα 5.5 τα Συχνά Σύνολα Μεγέθους Δύο.

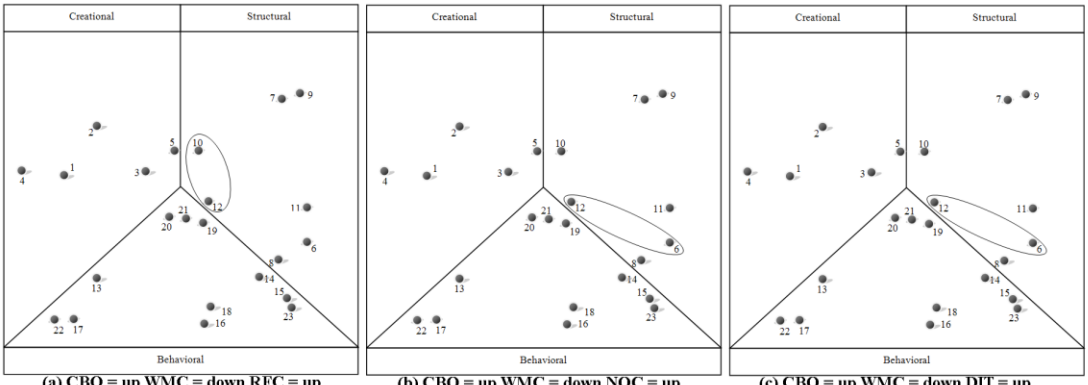
Το παραπάνω σχήμα δείχνει ότι υπάρχουν οκτώ συχνά σύνολα που υποστηρίζονται από πρότυπα που ανήκουν στην ίδια κατηγορία. Τα σχήματα 5.5 a,c,d,e και w παρουσιάζουν συχνά σύνολα τα οποία αποτελούνται από πρότυπα που ανήκουν στην κατηγορία Structural. Ακόμα στα σχήματα 5.5 k και p φαίνονται συχνά σύνολα που περιλαμβάνουν τα πρότυπα Singleton και Prototype. Δηλαδή τα συχνά σύνολα αυτά υποστηρίζονται από Creational πρότυπα, ενώ το συχνό σύνολο LCOM = down και RFC = up που υπάρχει στο σχήμα 5.5 q αποτελείται από τα Behavioral πρότυπα Visitor και Memento.

Ακόμα, υπάρχουν εννέα συχνά σύνολα που περιλαμβάνουν πρότυπα που ανήκουν σε διαφορετικές κατηγορίες, ενώ τα υπόλοιπα έξι συχνά σύνολα υπάρχουν σε πρότυπα που ανήκουν σε δυο διαφορετικές κατηγορίες. Πιο συγκεκριμένα τα πέντε από αυτά τα οποία φαίνονται στα σχήματα 5.5 b, h, m, t και u υποστηρίζονται από πρότυπα που ανήκουν στις κατηγορίες Behavioral και Structural. Ενώ υπάρχει και το συχνό σύνολο CBO = down και RFC = up το οποίο δείχνει το σχήμα 5.5 και υποστηρίζεται από τέσσερα Creational πρότυπα, δηλαδή τα πρότυπα Abstract Factory, Factory Method, Builder, Prototype, και από το Chain of Responsibility το οποίο ανήκει στην κατηγορία των Behavioral.

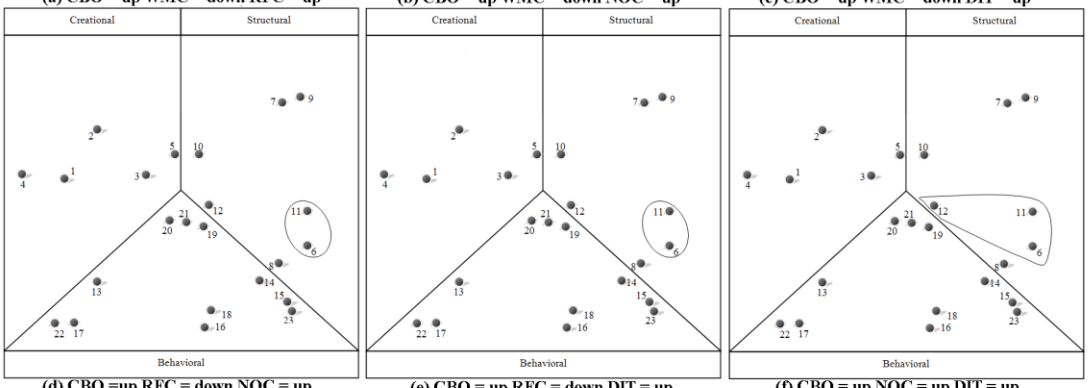
Τέλος το συχνό σύνολο με την μεγαλύτερη υποστήριξη φαίνεται στο σχήμα 5.5 v. Το σύνολο αυτό είναι το NOC = up και DIT = up. Το συγκεκριμένο σύνολο έχει υποστήριξη από δώδεκα πρότυπα τα οποία μάλιστα ανήκουν και στις τρεις κατηγορίες που έχουν οριστεί από την GoF. Το γεγονός ότι πάνω από τα μισά πρότυπα που εξετάζουμε έχουν τάση αύξησης στις μετρικές των ιεραρχιών οδηγεί στο συμπέρασμα ότι μια γενική αρχή που διέπει τα πρότυπα αυτά είναι αυτή της ανοιχτής – κλειστής σχεδίασης.

Το σχήμα 5.6 παρουσιάζει τα συχνά σύνολα μεγέθους τρία. Η κατηγορία αυτή αποτελείται από είκοσι τρία σύνολα και είναι η μεγαλύτερη κατηγορία συχνών συνόλων μαζί με την προηγούμενη.

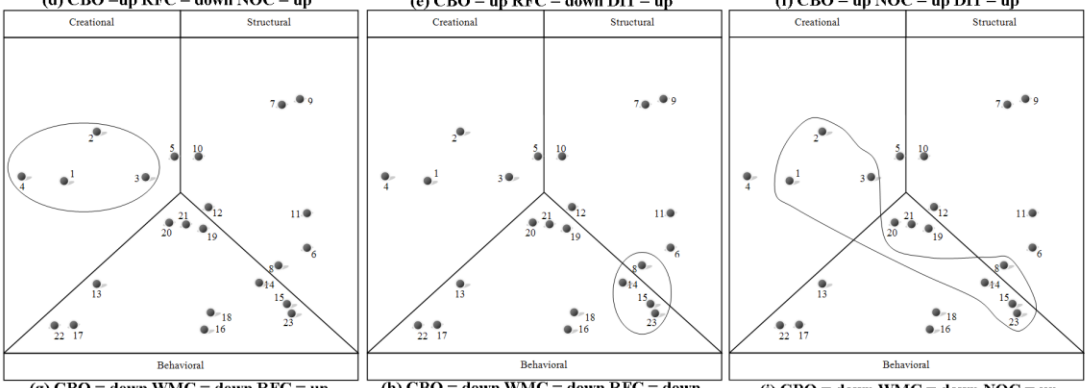
1 Abstract Factory 2 Builder 3 Factory Method 4 Prototype 5 Singleton 6 Adapter 7 Bridge 8 Composite 9 Decorator 10 Façade 11 Flyweight 12 Proxy 13 Chain of Responsibility 14 Command 15 Iterator 16 Mediator17 Memento 18 Observer 19 State 20 Strategy 21 Template Method 22 Visitor 23 Interpreter



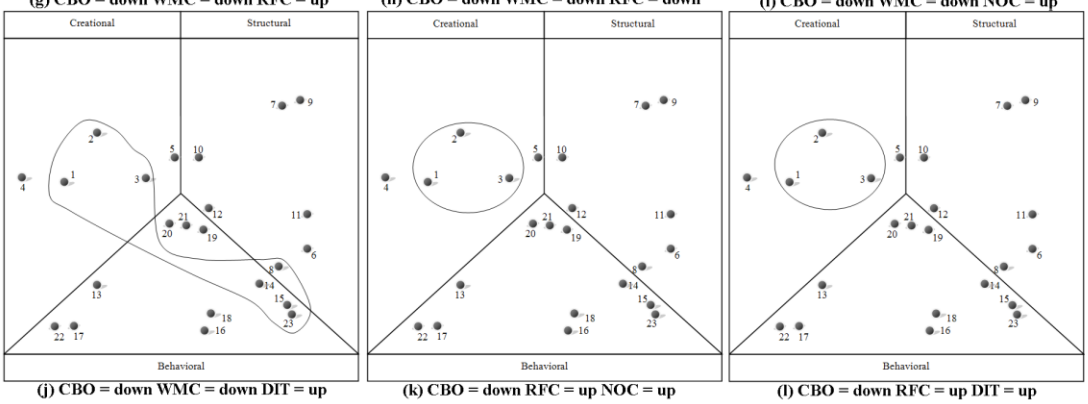

(a) CBO = up WMC = down RFC = up (b) CBO = up WMC = down NOC = up (c) CBO = up WMC = down DIT = up




(d) CBO =up RFC = down NOC = up (e) CBO = up RFC = down DIT = up (f) CBO = up NOC = up DIT = up

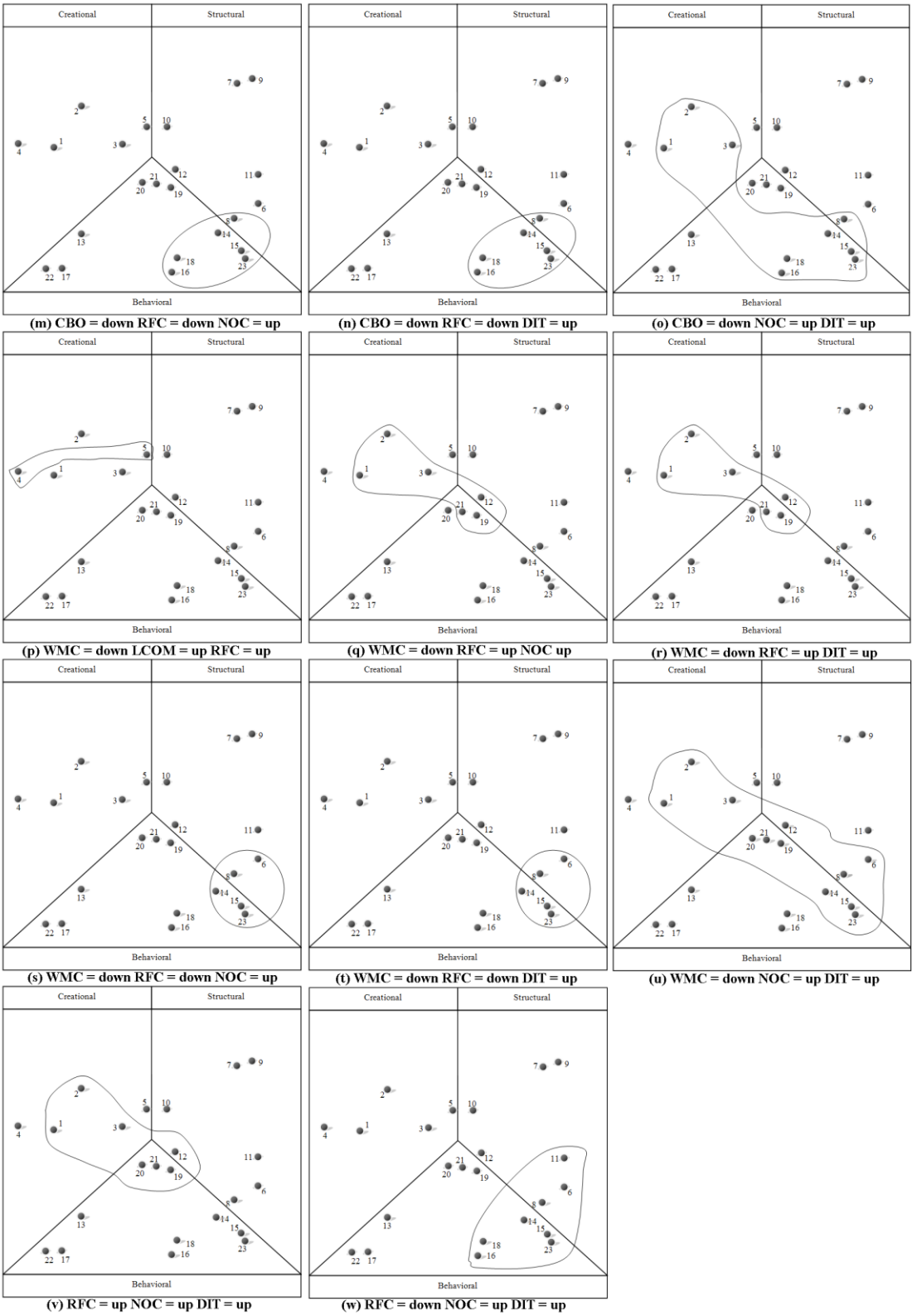



(g) CBO = down WMC = down RFC = up (h) CBO = down WMC = down RFC = down (i) CBO = down WMC = down NOC = up




(j) CBO = down WMC = down DIT = up (k) CBO = down RFC = up NOC = up (l) CBO = down RFC = up DIT = up

1 Abstract Factory 2 Builder 3 Factory Method 4 Prototype 5 Singleton 6 Adapter 7 Bridge 8 Composite 9 Decorator 10 Façade 11 Flyweight 12 Proxy 13 Chain of Responsibility 14 Command 15 Iterator 16 Mediator17 Memento 18 Observer 19 State 20 Strategy 21 Template Method 22 Visitor 23 Interpreter




(m) CBO = down RFC = down NOC = up (n) CBO = down RFC = down DIT = up (o) CBO = down NOC = up DIT = up

(p) WMC = down LCOM = up RFC = up (q) WMC = down RFC = up NOC up (r) WMC = down RFC = up DIT = up

(s) WMC = down RFC = down NOC = up (t) WMC = down RFC = down DIT = up (u) WMC = down NOC = up DIT = up

(v) RFC = up NOC = up DIT = up (w) RFC = down NOC = up DIT = up

Σχήμα 5.6 τα Συχνά Σύνολα Μεγέθους Τρία.

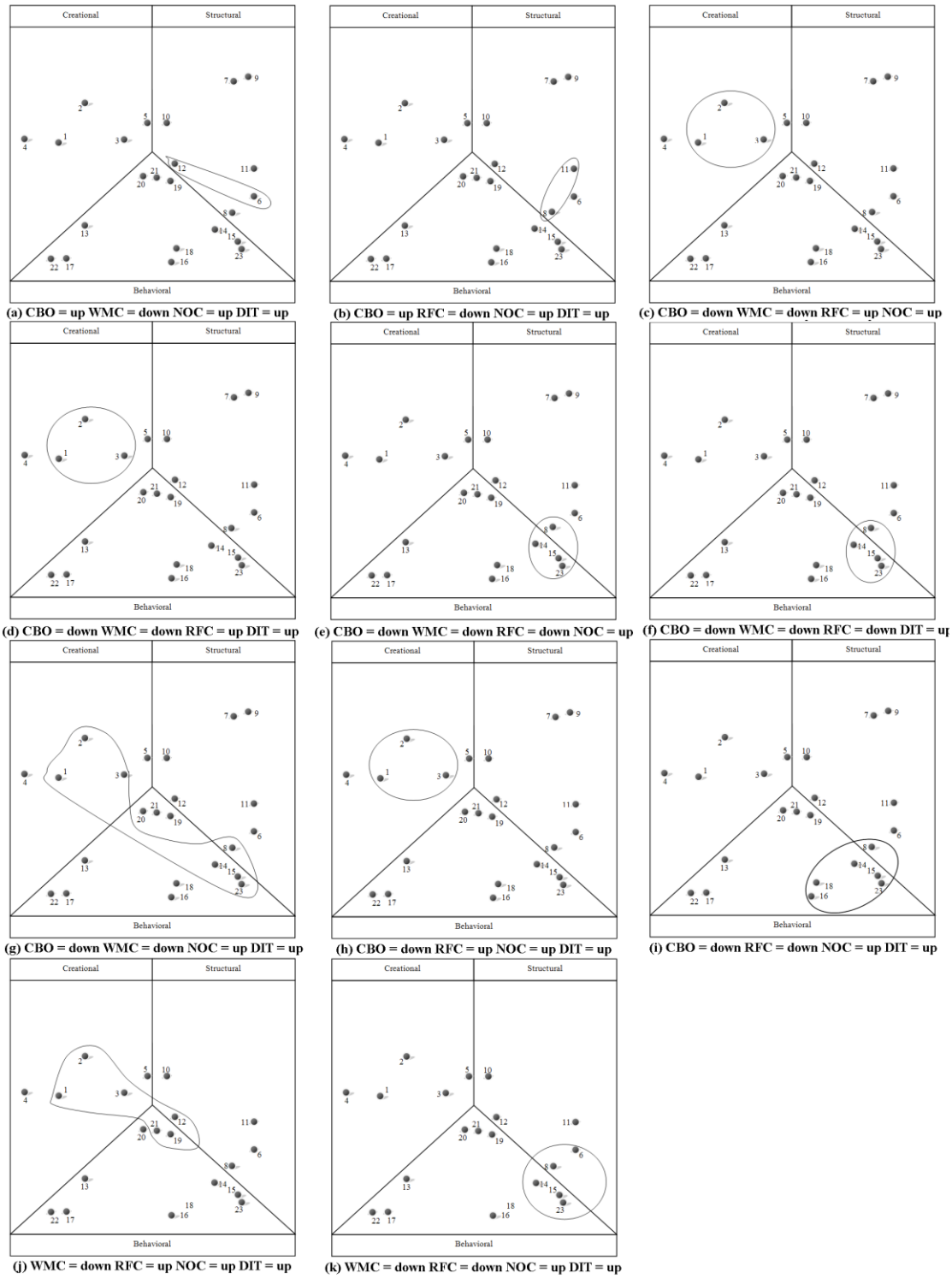
Η υποστήριξη του κάθε ενός από τα παραπάνω συχνά σύνολα γίνεται από σχεδιαστικά πρότυπα που ανήκουν σε μια συγκεκριμένη κατηγορία. Πιο συγκεκριμένα στα σχήματα 5.6 a,b,c,d,f φαίνονται συχνά σύνολα τα οποία έχουν υποστήριξη από Structural πρότυπα. Ακόμα τα συχνά σύνολα που υπάρχουν στα σχήματα 5.6 g,k,l και p έχουν υποστήριξη μόνο από πρότυπα της κατηγορίας Creational. Ενώ δεν υπάρχει κάποιο συχνό σύνολο που να έχει υποστήριξη από Behavioral πρότυπα.

Επιπλέον, υπάρχουν επτά συχνά σύνολα που υποστηρίζονται από πρότυπα που ανήκουν και στις τρεις κατηγορίες. Τέλος υπάρχουν έξι συχνά σύνολα τα οποία υποστηρίζονται από πρότυπα δυο κατηγόριων. Όπως δείχνουν τα σχήματα 5.6 h,n,m,s,t και w η υποστήριξη των συγκεκριμένων συχνών συνόλων γίνεται από Behavioral και Structural πρότυπα. Μεγαλύτερο ενδιαφέρον παρουσιάζουν τα συχνά σύνολα των σχημάτων h,m και n καθώς τα σύνολα αυτά υποστηρίζονται από Behavioral πρότυπα και από το Composite το οποίο ανήκει στην κατηγορία Structural. Παρόμοια παρατήρηση κάνει και ο Buschmann ο οποίος θεωρεί ότι δεν είναι ξεκάθαρος ο διαχωρισμός των προτύπων στις κατηγορίες Structural και Behavioral και δίνει ως παράδειγμα τον χαρακτηρισμό του προτύπου Composite ως Structural ενώ το πρότυπο Interpreter θεωρείται ως Behavioral.

Τα συχνά σύνολα μεγέθους τρία εμφανίζουν την μεγαλύτερη υποστήριξη από πρότυπα της ίδια κατηγορίας. Όπως προαναφέρθηκε τα δέκα από τα είκοσι τρία συχνά σύνολα μεγέθους τρία, υποστηρίζονται από πρότυπα μιας κατηγορίας. Ενώ η αντίστοιχη αναλόγια στα συχνά σύνολα μεγέθους δύο είναι οκτώ στα είκοσι τρία. Τέλος η ίδια αναλογία για τα συχνά σύνολα που έχουν μέγεθος ένα, είναι τέσσερα στα έντεκα.

Το σχήμα 5.7 παρουσιάζει τα συχνά σύνολα μεγέθους τέσσερα. Η συνδυαστική ανάλυση ανέδειξε έντεκα τέτοια σύνολα, όσα δηλαδή ήταν και τα συχνά σύνολα μεγέθους ένα.

| |
|---|
| 1 Abstract Factory 2 Builder 3 Factory Method 4 Prototype 5 Singleton 6 Adapter 7 Bridge 8 Composite 9 Decorator 10 Façade 11 Flyweight 12 Proxy 13 Chain of Responsibility 14 Command 15 Iterator 16 Mediator17 Memento 18 Observer 19 State 20 Strategy 21 Template Method 22 Visitor 23 Interpreter |



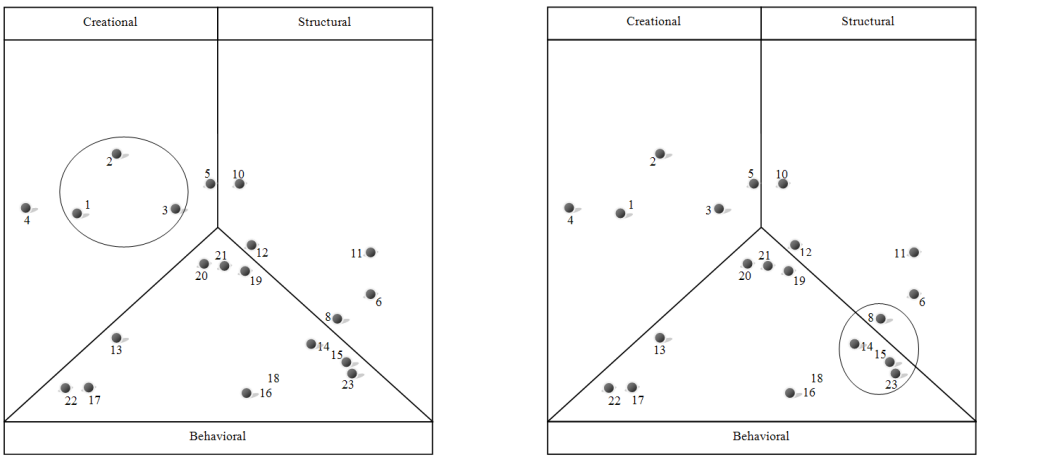

Σχήμα 5.7 τα Συχνά Σύνολα Μεγέθους Τέσσερα.

Τα σχήματα 5.7 a,b,c,d και h δείχνουν συχνά σύνολα με υποστήριξη από μια κατηγορία προτύπων το κάθε ένα. Πιο συγκεκριμένα δυο συχνά σύνολα υποστηρίζονται από Structural πρότυπα και φαίνονται στα σχήματα 5.7 α και b ενώ τα άλλα τρία έχουν υποστήριξη από τα πρότυπα Builder, Abstract Factory και Factory Method τα οποία είναι Creational.

Επιπλέον υπάρχουν τέσσερα συχνά σύνολα που υποστηρίζονται από Behavioral και Structural πρότυπα. Όπως φαίνεται στα σχήματα 5.7 e, f και g, υπάρχει και σε αυτήν την περίπτωση υποστήριξη συχνών συνόλων από μια ομάδα Behavioral προτύπων και από το Composite. Τέλος δυο από τα συχνά σύνολα υποστηρίζονται από πρότυπα και των τριών κατηγοριών.

Το σχήμα 5.8 δείχνει τα συχνά σύνολα μεγέθους πέντε. Είναι η τελευταία ομάδα συχνών συνόλων και αποτελείται από δυο τέτοια σύνολα.

1 Abstract Factory 2 Builder 3 Factory Method 4 Prototype 5 Singleton 6 Adapter 7 Bridge 8 Composite 9 Decorator 10 Façade 11 Flyweight 12 Proxy 13 Chain of Responsibility 14 Command 15 Iterator 16 Mediator17 Memento 18 Observer 19 State 20 Strategy 21 Template Method 22 Visitor 23 Interpreter




**(a)**
**CBO = down WMC = down RFC = up NOC = up DIT = up**

**(b)**
**CBO = down WMC = down RFC = down NOC = up DIT = up**

Σχήμα 5.8 τα Συχνά Σύνολα Μεγέθους Πέντε.

Όπως δείχνει το σχήμα 5.8 a το συχνό σύνολο αποτελείται από τα Creational πρότυπα Builder, Abstract Factory και Factory Method. Ενώ το συχνό σύνολο του σχήματος 5.8 b περιλαμβάνει τα Behavioral πρότυπα Command, Iterator, Interpreter και από το Composite το οποίο είναι Structural. Όπως φαίνεται στο σχήμα ο αριθμός των συχνών συνόλων μεγέθους πέντε είναι πολύ μικρότερος συγκριτικά με τις υπόλοιπες κατηγορίες. Όμως δίνει πολλές πληροφορίες για τα πρότυπα και αυτό γιατί εντοπίζει κάποια τιμή για τις πέντε από τις έξι μετρικές που εξετάζουμε.

Ο μεγαλύτερος αριθμός συχνών συνόλων χαρακτηρίζει τις κατηγορίες μεγέθους δυο και τρία οι οποίες έχουν από είκοσι τρία και δίνουν πληροφορίες για πολλές από τις ιδιότητες των προτύπων. Ακόμα οι κατηγορίες συχνών συνόλων μεγέθους ένα και τέσσερα περιλαμβάνουν έντεκα τέτοια σύνολα. Αντιθέτως, το συχνό σύνολο μεγέθους δυο NOC = up DIT = up έχει βαθμό υποστήριξης δεκαέξι ο οποίος είναι ο μεγαλύτερος που υπάρχει. Αυτό δείχνει μια συσχέτιση μεταξύ των μετρικών NOC και DIT η οποία ενισχύεται από το σχετικό σύνολο NOC = down DIT = down με υποστήριξη δυο. Πιο συγκεκριμένα στα δεκαοκτώ πρότυπα που εξετάζουμε η τιμή που παρουσιάζει η μια από της δυο αυτές μετρικές σε κάθε πρότυπο είναι ίδια με την τιμή της άλλης μετρικής.

Η χρήση των συχνών συνόλων δίνει μια πιο λεπτομερή εικόνα για τις ιδιότητες των προτύπων αναφορικά με τις μετρικές. Με αυτόν τον τρόπο μπορούν να παρουσιαστούν διαφορετικές προσεγγίσεις για την χρησιμότητα ενός προτύπου με στόχο την αντιμετώπιση κάποιου προβλήματος. Η χρήση των συχνών συνόλων παρουσιάζει περισσότερη πολυπλοκότητα στην διαδικασία εντοπισμού της κατάλληλης κατηγορίας προτύπων. Δηλαδή κάποιος που ενδιαφέρεται για τον εντοπισμό μιας κατηγορίας προτύπων που να έχει κάποιες συγκεκριμένες τιμές σε ένα υποσύνολο μετρικών θα πρέπει να ψάξει τα συχνά σύνολα, το πλήθος των οποίων είναι εβδομήντα.

# ΚΕΦΑΛΑΙΟ 6. ΕΠΙΛΟΓΟΣ

Τα σχεδιαστικά πρότυπα παρέχουν τεχνικές για την αντιμετώπιση προβλημάτων που προκύπτουν κατά την διαδικασία σχεδίασης ενός λογισμικού. Η γνώση των προτύπων αυτών είναι ιδιαίτερα σημαντική για τους σχεδιαστές λογισμικού και αυτό συμβαίνει γιατί τα σχεδιαστικά πρότυπα υποστηρίζουν μια αποδεδειγμένα καλή λύση για την αντιμετώπιση κάποιου προβλήματος. Πιο συγκεκριμένα οι λύσεις που προτείνουν έχουν εφαρμοστεί με επιτυχία στο παρελθόν κατά την σχεδίαση διαφορετικών λογισμικών. Συνεπώς εγγυώνται μια καλή σχεδίαση σε λογισμικά που πρόκειται να σχεδιαστούν μελλοντικά.

Το βασικό πρόβλημα που υπάρχει στην διαχείριση της γνώσης που παρέχουν τα σχεδιαστικά πρότυπα, είναι η οργάνωση της πληροφορίας που δίνουν με αποδοτικό τρόπο. Πιο συγκεκριμένα ζητούμενο είναι ο τρόπος με τον οποίο θα διαχειριζόμαστε τα πρότυπα έτσι ώστε, ο κάθε σχεδιαστής λογισμικού να οδηγείται εύκολα στην εύρεση του κατάλληλου προτύπου για την αντιμετώπιση του προβλήματος. Η κατηγοριοποίηση των προτύπων δίνει μια λύση στο πρόβλημα της διαχείρισης και αυτό γιατί χωρίζει τα πρότυπα σε διαφορετικές κατηγορίες χρησιμοποιώντας κάποια κριτήρια. Έχουν προταθεί διαφορετικές κατηγοριοποιήσεις οι οποίες βασίζονται σε διαφορετικά κριτήρια. Πιο συγκεκριμένα, η GoF κατηγοριοποιεί τα πρότυπα με κριτήρια τον σκοπό και το πλαίσιο εφαρμογής τους ενώ ο Zimmer έχει ως κριτήριο τις συσχετίσεις μεταξύ των προτύπων. Τέλος η κατηγοριοποίηση του Eckel έχει ως κριτήριο το πρόβλημα στο οποίο δίνει λύση το κάθε πρότυπο.

Όμως τα υπάρχοντα κριτήρια των κατηγοριοποιήσεων δεν βασίζονται στην χρήση κάποιων μετρικών. Συνεπώς οι κατηγοριοποιήσεις που προκύπτουν εμπεριέχουν και υποκειμενικές κρίσεις των ατόμων που προτείνουν την κάθε κατηγοριοποιήσει.

Ακόμα υπάρχουν και περιπτώσεις που ο διαχωρισμός μεταξύ των κατηγοριών δεν είναι ευδιάκριτος.

Στην παρούσα εργασία προτάθηκε μια κατηγοριοποίηση η οποία βασίζεται σε κριτήρια τα οποία είναι λιγότερο υποκειμενικά. Το σύνολο των μετρικών που πρότειναν οι Chidamber και Kemerer αποτελεί το κριτήριο για την κατηγοριοποίηση των σχεδιαστικών προτύπων. Στόχος μας ήταν η δημιουργία μιας κατηγοριοποίησης η οποία δεν επηρεάζεται από υποκειμενικά στοιχεία επιπλέον βασικό χαρακτηριστικό της κατηγοριοποίησης είναι ότι ομαδοποιεί πρότυπα που βελτιώνουν συγκεκριμένες μετρικές.

Για την ανάλυση των δεδομένων χρησιμοποιήθηκαν δυο τεχνικές. Η συσταδοποίηση των προτύπων οδηγεί με ιεραρχικό τρόπο στην ανάδειξη πέντε κατηγοριών οι οποίες χαρακτηρίζονται από τις μετρικές που βελτιώνονται με την χρήση των προτύπων, ακόμα υπάρχουν κατηγορίες που έχουν και επιπλέον χαρακτηριστικά μετρικές που επιβαρύνονται με την χρήση κάποιων προτύπων. Έτσι λοιπόν ένας σχεδιαστής λογισμικού που ενδιαφέρεται να βελτιώσει κάποια μετρική σε ένα σύστημα χρησιμοποιώντας κάποιο σχεδιαστικό πρότυπο, δεν είναι απαραίτητο να ψάξει στο σύνολο των προτύπων αλλά μόνο σε αυτά που ανήκουν στην αντίστοιχη κατηγορία.

Η συνδυαστική ανάλυση των δεδομένων που προκύπτουν από τα πρότυπα οδηγεί στην ανάδειξη συχνών συνόλων που έχουν να κάνουν με τις τιμές που παίρνουν οι μετρικές. Η χρήση των συχνών συνόλων οδηγεί σε μια πιο λεπτομερή καταγραφή των ιδιοτήτων που έχουν τα σχεδιαστικά πρότυπα. Έτσι λοιπόν ένας σχεδιαστής μπορεί να θέτει πολύπλοκα ερωτήματα αναφορικά με τις μετρικές που τον ενδιαφέρουν και να δέχεται ως απάντηση ένα πιο περιορισμένο σύνολο προτύπων. Δηλαδή η κατηγοριοποίηση που προκύπτει από την ιεραρχική συσταδοποίηση των προτύπων οδηγεί στην ανάδειξη μερικών σημαντικών ιδιοτήτων των προτύπων ενώ οι υπόλοιπες ιδιότητες χάνονται κατά την διαδικασία δημιουργίας των κατηγοριών. Αντίθετα, τα συχνά σύνολα μετρικών που προκύπτουν μέσω της συνδυαστικής ανάλυσης, αναδεικνύουν όλες τις ιδιότητες των προτύπων. Βέβαια ο αριθμός των συχνών συνόλων είναι μεγάλος συνεπώς η διαδικασία εύρεσης του κατάλληλου προτύπου θα πρέπει να υποστηρίζεται και από κάποιον αλγόριθμο.

# ΑΝΑΦΟΡΕΣ

[1] R. Agrawal and R. Srikant. “Fast Algorithms for Mining Association Rules in Large Databases”, VLDB, 487-499, 1994

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel. “A pattern Language”, Oxford University Press, New Work, 1977

[3] P. Andritsos and V. Tzerpos, “Information-Theoretic Software Clustering,” IEEE Transactions on Software Engineering, Vol. 31(2), pp 150-165, February 2005.

[4] V. Basili, L. Briand and W. Melo. “A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Transactions on Software Engineering. vol. 22, no. 10, pp 89-104, October 1996.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland and M. Stal. ”Pattern-Oriented Software Architecture: A System of Patterns”. John Wiley & Sons Ltd. , 1996.

[6] S. R. Chidamber, C. F. Kemerer. “A Metrics suite for Object Oriented design”, IEEE Transactions on Software Engineering, vol 20(6), pp 476-493, June 1994.

[7] J. O. Coplien. “Setting the Stage. *C++ Report”*, vol 6(8), pp 8-16, October 1994.

[8] J. O. Coplien. “Software Patterns”, SIGS Books & Multimedia, 2000.

[9] T. Demarco. “Structured Analysis and System Specification”, Prentice Hall, ISBN 0138543801, 1979.

[10] B. Eckel, “Thinking in Patterns”, Revision 0.9 , May 20, 2003, HTTP://WWW.MINDVIEWINC.COM/BOOKS/

[11] E. Gamma, R. Helm R. Johnson and J. Vlissides. “Design Patterns: Elements of Reusable Object Oriented Software”, Addison-Wesley Professional Computing Series, 1994.

[12] D. Glasberg, K. El Emam, W. Melo and N. Madhavji. “Validating Object-Oriented Design Metrics on a CommercialJava Application”, Technical Report, NRC/ERB-1080 (National Research Council of Canada), September 2000.

[13] R.Keller, R.Schauer, S.Robitaille, P.Page, “Pattern Based Reverse - Engineering of Design Components”. In Proc. 21st International Conference on Software Engineering, 1999.

[14] B. Liskov. “Data Abstraction and Hierarchy”, Conference on Object Oriented Programming Systems Languages and Applications, pp 16-34, October 1987.

[15] W. Li and S. Henry. “Object-Oriented Metric that Predict Maintainability”, Journal of Systems and Software, Volume 23(2), pp 111-122, November 1993.

[16] T.J. McCabe. “A Complexity Measure”, IEEE Transactions on Software Engineering Vol. 2, No. 4, p. 308 (1976)

[17] O. Maqbool and H. A. Babri. “Hierarchical Clustering for Software Architecture Recovery”, IEEE Transactions on Software Engineering, Vol 33(11), pp 759-780, November 2007.

[18] B. Meyer. “Object-Oriented Software Construction”, Prentice Hall, Hemel Hempstead, 1988.

[19] D.L.Parnas.” On the Criteria to Be Used in Decomposing Systems into Modules”, Communications of the ACM, vol 15, no 12,pp 1053-1058 December 1972.

[20] H. A. Sahraoui, R. Godin and T. Miceli. “Can Metrics Help Bridging the Gap Between the Improvement of OO Design Quality and Its Automation?”, 16th IEEE International Conference on Software Maintenance (ICSM'00), October 200.

[21] W. P. Stevens, G.J. Myers, L. L. Constantine. “Structured Design”,IBM Systems Journal, vol 13, no 2, pp 115-139, 1974.

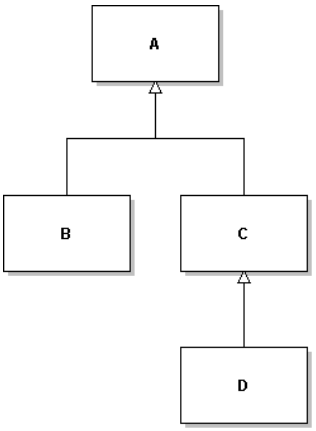
[22]W. F. Tichy, “A catalogue of General-Purpose Software Design Patterns”, IEEE Computer Society, Technology of Object-Oriented Languages and Systems, pp 330-339,1997.

[23] W. Zimmer, “Relationships between Design Patterns”. *Pattern Languages of Program Design*, ACM Press/Addison-Wesley Publishing Co, pp 345 – 364, 1995.

# ΠΑΡΑΡΤΗΜΑ Α

Στο παράρτημα αυτό υπάρχουν παραδείγματα υπολογισμού των μετρικών σε διαφορετικά διαγράμματα κλάσεων.

DIT = το μήκος της μέγιστης διαδρομής από κάποιον κόμβο ως την ρίζα της ιεραρχίας.

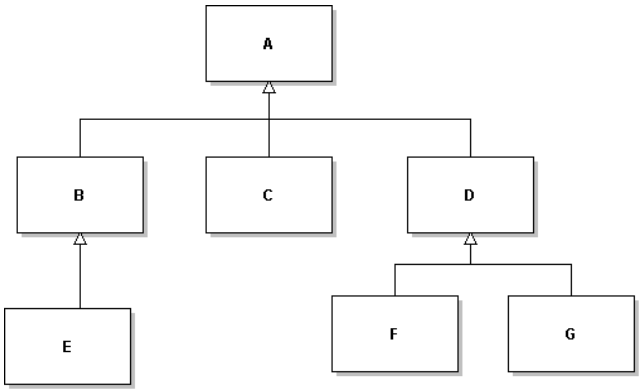



Σχήμα Α.1 μια Ιεραρχία Κλάσεων

Πίνακας Α.3 Υπολογισμό της Μετρική DIT

| | DIT |
|---|---|
| A | 0 |
| B | 1 |
| C | 1 |
| D | 2 |

NOC = ο αριθμός των παιδιών που κληρονομούν από την κλάση πατέρα

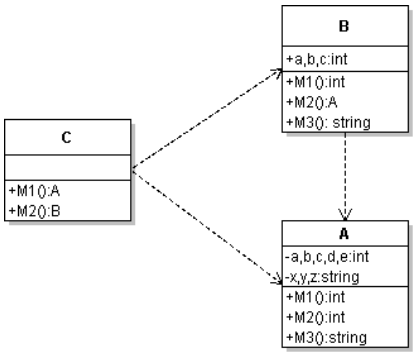



Σχήμα Α.2 Δεύτερη Ιεραρχία Κλάσεων

Πίνακας Α.4 Υπολογισμό των Μετρικών DIT και NOC.

| | NOC | DIT |
|---|---|---|
| A | 3 | 0 |
| B | 1 | 1 |
| C | 0 | 1 |
| D | 2 | 1 |
| E | 0 | 2 |
| F | 0 | 2 |
| G | 0 | 2 |

LCOM = βαθμός συνάφειας των μεθόδων μιας κλάσης




Σχήμα Α.3 Σύστημα τριών Κλάσεων

```
public class A{
private int a,b,c,d,e;
private string x,y,z;
  public int M1(){
    a = b+c+d+e;
  return a;}
  public int M2(){
   a = b*e;
  return a;}
  public string M3(){
   x = y+z;
  return x;}
}
```

```
public class B{
private int a,b,c;

  public int M1(){
    a = b+c;
  return a;}
  public A M2(){
  return new A();
  }
  public void M3(){
  System.out.println("M3");
  }
}
```

```
public class C{

  public void M1(){
   return new A();
  }

  public int M2(){
   return new B();
  }
}
```

Σχήμα Α.4 οι Υλοποιήσεις των τριών Κλάσεων.

Υπολογισμός LCOM για την κλάση A:

Το σύνολο των μεταβλητών είναι I ={a,b,c,d,e,x,y,z} κάθε μέθοδο M1,M2 και M3 έχει τα αντίστοιχα σύνολα I1={a,b,c,d,e},I2={a,b,e} και I3={x,y,z} τότε $I_1 \cap I_2$={a,b,e}, $I_1 \cap I_3 = \varnothing$ και $I_2 \cap I_3 = \varnothing$.

Άρα το P = 2 και Q = 1, LCOM = $|P| - |Q| = 2 - 1 = 1$.

Υπολογισμός LCOM για την κλάση B:

Το σύνολο των μεταβλητών είναι I ={a,b,c,} κάθε μέθοδο M1,M2 και M3 έχει τα αντίστοιχα σύνολα I1={a,b,c},I2=∅ και I3=∅ τότε $I_1 \cap I_2 = \varnothing$, $I_1 \cap I_3 = \varnothing$ και $I_2 \cap I_3$ δεν ορίζεται.

Άρα το P = 3 και Q = 0, LCOM = $|P| - |Q| = 2 - 0 = 2$.

Υπολογισμός LCOM για την κλάση C:

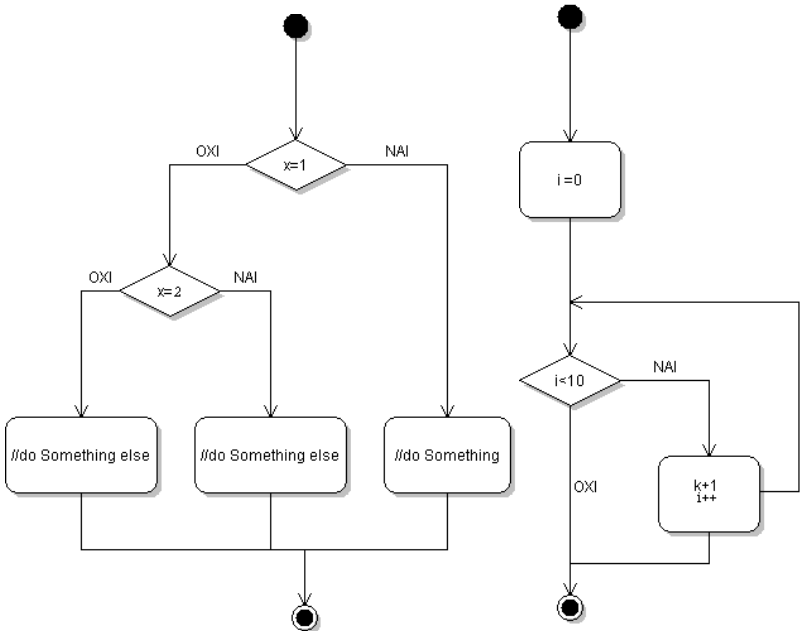
Το σύνολο των μεταβλητών είναι I =∅ κάθε μέθοδο M1 και M2 έχει τα αντίστοιχα σύνολα I1=∅ και I2=∅ τότε $I_1 \cap I_2$ δεν ορίζεται.

Άρα δεν ορίζεται έλλειψη συνεκτικότητας για την κλάση C.

WMC = πολυπλοκότητα της κλάσης

```
public class Product(){
  void operation1(int x){
     if (x==1){ //do Something}
     else if (x==2){//do Something else}
     else // do nothing
   }
   void operation2 (int k){
     for (int i=0; i<10;i++){
        k=k+1
      }
   }
   void operation3(){
     System.out.println(“operation3”) ;
   }
}
```

Σχήμα Α.5 Υλοποίηση της Κλάσης Product




a) operation1 b) operation2

Σχήμα Α.6 τα Διαγράμματα Ροής των Μεθόδων

Το WMC της κλάσης Product είναι ίσο με το άθροισμα των πολυπλοκοτήτων των τριών μεθόδων. Ενώ η πολυπλοκότητα της κάθε μεθόδου υπολογίζεται βάσεις του κυκλωματικού αριθμού (CC) το σχήμα Π.6 μας δείχνει τα διαγράμματα ροής για τις μεθόδους operation1 και operation2. Ο κυκλωματικός ορισμός μιας μεθόδου ορίζεται από το άθροισμα των έλεγχων που υπάρχουν στο αντίστοιχο διάγραμμα. Άρα σύμφωνα με τα παραπάνω, η μετρική WMC στην κλάση θα υπολογίζεται από τον παρακάτω τύπο.

$WMC_{Product} = CC_{operation1} + CC_{operation2} + CC_{operation3}$

$CC_{operation1} = 1+1 = 2$ ( 1 από x=1 και 1 από x=2)

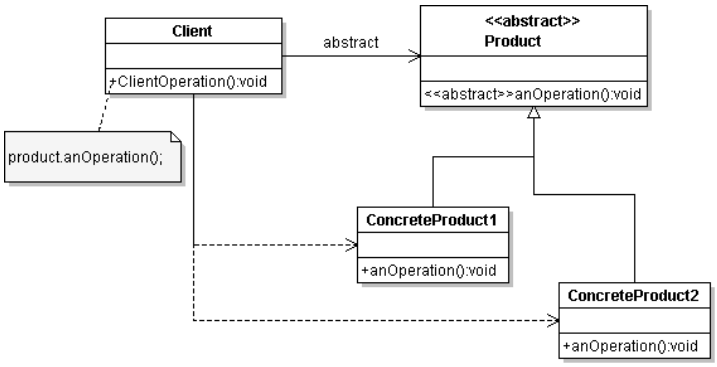
$CC_{operation2} = 1$ (1 από i<10)

$CC_{operation3} = 0$ (κανένας έλεγχος)

Άρα $WMC_{Product} = 3$

CBO = η σύζευξη μεταξύ των αντικειμένων. Στον υπολογισμό της μετρικής CBO δεν μετράει η σύζευξη με στοιχεία αφαίρεσης, αλλά μόνο από κλάσεις.

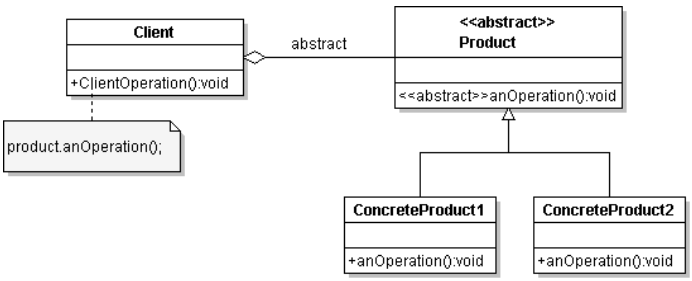
RFC = το άθροισμα των μεθόδων μιας κλάσης μαζί με το άθροισμα των εξωτερικών μεθόδων που καλεί η συγκεκριμένη κλάση. Σε αυτήν την περίπτωση η κλήση μιας εξωτερικής μεθόδου που είναι αφηρημένη υπολογίζεται κανονικά.




Σχήμα Α.7 οι Client και η Ιεραρχία Κλάσεων Product

Πίνακας Α.5 Υπολογισμός Μετρικών ενός Συστήματος

| | CBO | RFC | NOC | DIT |
|---|---|---|---|---|
| Client | 2 | 2 | 0 | 0 |
| Product | 0 | 0 | 2 | 0 |
| ConcreteProduct1 | 0 | 0 | 0 | 1 |
| ConcreteProduct2 | 0 | 0 | 0 | 1 |




Σχήμα Α.8 ο Client Συναθροίζοντας Αντικείμενα Τύπου Product

Πίνακας Α.6 Υπολογισμός Μετρικών του Συστήματος

| | CBO | RFC | NOC | DIT |
|---|---|---|---|---|
| Client | 0 | 2 | 0 | 0 |
| Product | 0 | 0 | 2 | 0 |
| ConcreteProduct1 | 0 | 0 | 0 | 1 |
| ConcreteProduct2 | 0 | 0 | 0 | 1 |

# ΠΑΡΑΡΤΗΜΑ Β

Τα συχνά σύνολα που βρέθηκαν με την εκτέλεση του αλγορίθμου apriori. Στην εκτέλεση αυτή δεν υπολογίζονται οι μετρικές που έχουν τιμή same.

```
Minimum support: 0.1 (2 patterns)

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Large Itemsets L(1):
CBO=up 5
CBO=down 11
WMC=down 15
LCOM=up 2
LCOM=down 2
RFC=up 13
RFC=down 9
NOC=up 16
NOC=down 2
DIT=up 16
DIT=down 2

Size of set of large itemsets L(2): 23

Large Itemsets L(2):
CBO=up WMC=down 3
CBO=up RFC=up 3
CBO=up RFC=down 2
CBO=up NOC=up 3
CBO=up DIT=up 3
CBO=down WMC=down 8
CBO=down RFC=up 5
CBO=down RFC=down 6
CBO=down NOC=up 9
CBO=down DIT=up 9
WMC=down LCOM=up 2
WMC=down RFC=up 9
WMC=down RFC=down 5
WMC=down NOC=up 12
WMC=down DIT=up 12
LCOM=up RFC=up 2
LCOM=down RFC=up 2
RFC=up NOC=up 7
RFC=up DIT=up 7
RFC=down NOC=up 8
```

```
RFC=down DIT=up 8
NOC=up DIT=up 16
NOC=down DIT=down 2

Size of set of large itemsets L(3): 23

Large Itemsets L(3):
CBO=up WMC=down RFC=up 2
CBO=up WMC=down NOC=up 2
CBO=up WMC=down DIT=up 2
CBO=up RFC=down NOC=up 2
CBO=up RFC=down DIT=up 2
CBO=up NOC=up DIT=up 3
CBO=down WMC=down RFC=up 4
CBO=down WMC=down RFC=down 4
CBO=down WMC=down NOC=up 7
CBO=down WMC=down DIT=up 7
CBO=down RFC=up NOC=up 3
CBO=down RFC=up DIT=up 3
CBO=down RFC=down NOC=up 6
CBO=down RFC=down DIT=up 6
CBO=down NOC=up DIT=up 9
WMC=down LCOM=up RFC=up 2
WMC=down RFC=up NOC=up 6
WMC=down RFC=up DIT=up 6
WMC=down RFC=down NOC=up 5
WMC=down RFC=down DIT=up 5
WMC=down NOC=up DIT=up 12
RFC=up NOC=up DIT=up 7
RFC=down NOC=up DIT=up 8

Size of set of large itemsets L(4): 11

Large Itemsets L(4):
CBO=up WMC=down NOC=up DIT=up 2
CBO=up RFC=down NOC=up DIT=up 2
CBO=down WMC=down RFC=up NOC=up 3
CBO=down WMC=down RFC=up DIT=up 3
CBO=down WMC=down RFC=down NOC=up 4
CBO=down WMC=down RFC=down DIT=up 4
CBO=down WMC=down NOC=up DIT=up 7
CBO=down RFC=up NOC=up DIT=up 3
CBO=down RFC=down NOC=up DIT=up 6
WMC=down RFC=up NOC=up DIT=up 6
WMC=down RFC=down NOC=up DIT=up 5

Size of set of large itemsets L(5): 2

Large Itemsets L(5):
CBO=down WMC=down RFC=up NOC=up DIT=up 3
CBO=down WMC=down RFC=down NOC=up DIT=up 4
```

# ΠΑΡΑΡΤΗΜΑ Γ

Τα συχνά σύνολα που βρέθηκαν με την εκτέλεση του αλγορίθμου apriori. Στην εκτέλεση αυτή υπολογίζονται και οι μετρικές που έχουν τιμή same.

```
Minimum support: 0.1 (2 patterns)

Generated sets of large itemsets:

Size of set of large itemsets L(1): 16

Large Itemsets L(1):
CBO=up 5
CBO=down 11
CBO=same 7
WMC=down 15
WMC=same 7
LCOM=up 2
LCOM=down 2
LCOM=same 19
RFC=up 13
RFC=down 9
NOC=up 16
NOC=down 2
NOC=same 5
DIT=up 16
DIT=down 2
DIT=same 5

Size of set of large itemsets L(2): 64

Large Itemsets L(2):
CBO=up WMC=down 3
CBO=up LCOM=same 4
CBO=up RFC=up 3
CBO=up RFC=down 2
CBO=up NOC=up 3
CBO=up NOC=same 2
CBO=up DIT=up 3
CBO=up DIT=same 2
CBO=down WMC=down 8
CBO=down WMC=same 3
CBO=down LCOM=same 10
CBO=down RFC=up 5
CBO=down RFC=down 6
CBO=down NOC=up 9
CBO=down NOC=same 2
CBO=down DIT=up 9
```

```
CBO=down DIT=same 2
CBO=same WMC=down 4
CBO=same WMC=same 3
CBO=same LCOM=same 5
CBO=same RFC=up 5
CBO=same NOC=up 4
CBO=same NOC=down 2
CBO=same DIT=up 4
CBO=same DIT=down 2
WMC=down LCOM=up 2
WMC=down LCOM=same 13
WMC=down RFC=up 9
WMC=down RFC=down 5
WMC=down NOC=up 12
WMC=down NOC=same 3
WMC=down DIT=up 12
WMC=down DIT=same 3
WMC=same LCOM=down 2
WMC=same LCOM=same 5
WMC=same RFC=up 4
WMC=same RFC=down 3
WMC=same NOC=up 3
WMC=same NOC=down 2
WMC=same NOC=same 2
WMC=same DIT=up 3
WMC=same DIT=down 2
WMC=same DIT=same 2
LCOM=up RFC=up 2
LCOM=up NOC=same 2
LCOM=up DIT=same 2
LCOM=down RFC=up 2
LCOM=same RFC=up 9
LCOM=same RFC=down 9
LCOM=same NOC=up 15
LCOM=same NOC=down 2
LCOM=same NOC=same 2
LCOM=same DIT=up 15
LCOM=same DIT=down 2
LCOM=same DIT=same 2
RFC=up NOC=up 7
RFC=up NOC=same 5
RFC=up DIT=up 7
RFC=up DIT=same 5
RFC=down NOC=up 8
RFC=down DIT=up 8
NOC=up DIT=up 16
NOC=down DIT=down 2
NOC=same DIT=same 5

Size of set of large itemsets L(3): 96

Large Itemsets L(3):
CBO=up WMC=down LCOM=same 3
CBO=up WMC=down RFC=up 2
CBO=up WMC=down NOC=up 2
CBO=up WMC=down DIT=up 2
CBO=up LCOM=same RFC=up 2
CBO=up LCOM=same RFC=down 2
CBO=up LCOM=same NOC=up 3
CBO=up LCOM=same DIT=up 3
```

```
CBO=up RFC=up NOC=same 2
CBO=up RFC=up DIT=same 2
CBO=up RFC=down NOC=up 2
CBO=up RFC=down DIT=up 2
CBO=up NOC=up DIT=up 3
CBO=up NOC=same DIT=same 2
CBO=down WMC=down LCOM=same 7
CBO=down WMC=down RFC=up 4
CBO=down WMC=down RFC=down 4
CBO=down WMC=down NOC=up 7
CBO=down WMC=down DIT=up 7
CBO=down WMC=same LCOM=same 3
CBO=down WMC=same RFC=down 2
CBO=down WMC=same NOC=up 2
CBO=down WMC=same DIT=up 2
CBO=down LCOM=same RFC=up 4
CBO=down LCOM=same RFC=down 6
CBO=down LCOM=same NOC=up 9
CBO=down LCOM=same DIT=up 9
CBO=down RFC=up NOC=up 3
CBO=down RFC=up NOC=same 2
CBO=down RFC=up DIT=up 3
CBO=down RFC=up DIT=same 2
CBO=down RFC=down NOC=up 6
CBO=down RFC=down DIT=up 6
CBO=down NOC=up DIT=up 9
CBO=down NOC=same DIT=same 2
CBO=same WMC=down LCOM=same 3
CBO=same WMC=down RFC=up 3
CBO=same WMC=down NOC=up 3
CBO=same WMC=down DIT=up 3
CBO=same WMC=same LCOM=same 2
CBO=same WMC=same RFC=up 2
CBO=same WMC=same NOC=down 2
CBO=same WMC=same DIT=down 2
CBO=same LCOM=same RFC=up 3
CBO=same LCOM=same NOC=up 3
CBO=same LCOM=same NOC=down 2
CBO=same LCOM=same DIT=up 3
CBO=same LCOM=same DIT=down 2
CBO=same RFC=up NOC=up 3
CBO=same RFC=up DIT=up 3
CBO=same NOC=up DIT=up 4
CBO=same NOC=down DIT=down 2
WMC=down LCOM=up RFC=up 2
WMC=down LCOM=up NOC=same 2
WMC=down LCOM=up DIT=same 2
WMC=down LCOM=same RFC=up 7
WMC=down LCOM=same RFC=down 5
WMC=down LCOM=same NOC=up 12
WMC=down LCOM=same DIT=up 12
WMC=down RFC=up NOC=up 6
WMC=down RFC=up NOC=same 3
WMC=down RFC=up DIT=up 6
WMC=down RFC=up DIT=same 3
WMC=down RFC=down NOC=up 5
WMC=down RFC=down DIT=up 5
WMC=down NOC=up DIT=up 12
WMC=down NOC=same DIT=same 3
WMC=same LCOM=down RFC=up 2
```

```
WMC=same LCOM=same RFC=up 2
WMC=same LCOM=same RFC=down 3
WMC=same LCOM=same NOC=up 2
WMC=same LCOM=same NOC=down 2
WMC=same LCOM=same DIT=up 2
WMC=same LCOM=same DIT=down 2
WMC=same RFC=up NOC=same 2
WMC=same RFC=up DIT=same 2
WMC=same RFC=down NOC=up 2
WMC=same RFC=down DIT=up 2
WMC=same NOC=up DIT=up 3
WMC=same NOC=down DIT=down 2
WMC=same NOC=same DIT=same 2
LCOM=up RFC=up NOC=same 2
LCOM=up RFC=up DIT=same 2
LCOM=up NOC=same DIT=same 2
LCOM=same RFC=up NOC=up 6
LCOM=same RFC=up NOC=same 2
LCOM=same RFC=up DIT=up 6
LCOM=same RFC=up DIT=same 2
LCOM=same RFC=down NOC=up 8
LCOM=same RFC=down DIT=up 8
LCOM=same NOC=up DIT=up 15
LCOM=same NOC=down DIT=down 2
LCOM=same NOC=same DIT=same 2
RFC=up NOC=up DIT=up 7
RFC=up NOC=same DIT=same 5
RFC=down NOC=up DIT=up 8

Size of set of large itemsets L(4): 67

Large Itemsets L(4):
CBO=up WMC=down LCOM=same RFC=up 2
CBO=up WMC=down LCOM=same NOC=up 2
CBO=up WMC=down LCOM=same DIT=up 2
CBO=up WMC=down NOC=up DIT=up 2
CBO=up LCOM=same RFC=down NOC=up 2
CBO=up LCOM=same RFC=down DIT=up 2
CBO=up LCOM=same NOC=up DIT=up 3
CBO=up RFC=up NOC=same DIT=same 2
CBO=up RFC=down NOC=up DIT=up 2
CBO=down WMC=down LCOM=same RFC=up 3
CBO=down WMC=down LCOM=same RFC=down 4
CBO=down WMC=down LCOM=same NOC=up 7
CBO=down WMC=down LCOM=same DIT=up 7
CBO=down WMC=down RFC=up NOC=up 3
CBO=down WMC=down RFC=up DIT=up 3
CBO=down WMC=down RFC=down NOC=up 4
CBO=down WMC=down RFC=down DIT=up 4
CBO=down WMC=down NOC=up DIT=up 7
CBO=down WMC=same LCOM=same RFC=down 2
CBO=down WMC=same LCOM=same NOC=up 2
CBO=down WMC=same LCOM=same DIT=up 2
CBO=down WMC=same RFC=down NOC=up 2
CBO=down WMC=same RFC=down DIT=up 2
CBO=down WMC=same NOC=up DIT=up 2
CBO=down LCOM=same RFC=up NOC=up 3
CBO=down LCOM=same RFC=up DIT=up 3
CBO=down LCOM=same RFC=down NOC=up 6
CBO=down LCOM=same RFC=down DIT=up 6
```

```
CBO=down LCOM=same NOC=up DIT=up 9
CBO=down RFC=up NOC=up DIT=up 3
CBO=down RFC=up NOC=same DIT=same 2
CBO=down RFC=down NOC=up DIT=up 6
CBO=same WMC=down LCOM=same RFC=up 2
CBO=same WMC=down LCOM=same NOC=up 3
CBO=same WMC=down LCOM=same DIT=up 3
CBO=same WMC=down RFC=up NOC=up 2
CBO=same WMC=down RFC=up DIT=up 2
CBO=same WMC=down NOC=up DIT=up 3
CBO=same WMC=same LCOM=same NOC=down 2
CBO=same WMC=same LCOM=same DIT=down 2
CBO=same WMC=same NOC=down DIT=down 2
CBO=same LCOM=same RFC=up NOC=up 2
CBO=same LCOM=same RFC=up DIT=up 2
CBO=same LCOM=same NOC=up DIT=up 3
CBO=same LCOM=same NOC=down DIT=down 2
CBO=same RFC=up NOC=up DIT=up 3
WMC=down LCOM=up RFC=up NOC=same 2
WMC=down LCOM=up RFC=up DIT=same 2
WMC=down LCOM=up NOC=same DIT=same 2
WMC=down LCOM=same RFC=up NOC=up 6
WMC=down LCOM=same RFC=up DIT=up 6
WMC=down LCOM=same RFC=down NOC=up 5
WMC=down LCOM=same RFC=down DIT=up 5
WMC=down LCOM=same NOC=up DIT=up 12
WMC=down RFC=up NOC=up DIT=up 6
WMC=down RFC=up NOC=same DIT=same 3
WMC=down RFC=down NOC=up DIT=up 5
WMC=same LCOM=same RFC=down NOC=up 2
WMC=same LCOM=same RFC=down DIT=up 2
WMC=same LCOM=same NOC=up DIT=up 2
WMC=same LCOM=same NOC=down DIT=down 2
WMC=same RFC=up NOC=same DIT=same 2
WMC=same RFC=down NOC=up DIT=up 2
LCOM=up RFC=up NOC=same DIT=same 2
LCOM=same RFC=up NOC=up DIT=up 6
LCOM=same RFC=up NOC=same DIT=same 2
LCOM=same RFC=down NOC=up DIT=up 8

Size of set of large itemsets L(5): 25

Large Itemsets L(5):
CBO=up WMC=down LCOM=same NOC=up DIT=up 2
CBO=up LCOM=same RFC=down NOC=up DIT=up 2
CBO=down WMC=down LCOM=same RFC=up NOC=up 3
CBO=down WMC=down LCOM=same RFC=up DIT=up 3
CBO=down WMC=down LCOM=same RFC=down NOC=up 4
CBO=down WMC=down LCOM=same RFC=down DIT=up 4
CBO=down WMC=down LCOM=same NOC=up DIT=up 7
CBO=down WMC=down RFC=up NOC=up DIT=up 3
CBO=down WMC=down RFC=down NOC=up DIT=up 4
CBO=down WMC=same LCOM=same RFC=down NOC=up 2
CBO=down WMC=same LCOM=same RFC=down DIT=up 2
CBO=down WMC=same LCOM=same NOC=up DIT=up 2
CBO=down WMC=same RFC=down NOC=up DIT=up 2
CBO=down LCOM=same RFC=up NOC=up DIT=up 3
CBO=down LCOM=same RFC=down NOC=up DIT=up 6
CBO=same WMC=down LCOM=same RFC=up NOC=up 2
CBO=same WMC=down LCOM=same RFC=up DIT=up 2
```

```
CBO=same WMC=down LCOM=same NOC=up DIT=up 3
CBO=same WMC=down RFC=up NOC=up DIT=up 2
CBO=same WMC=same LCOM=same NOC=down DIT=down 2
CBO=same LCOM=same RFC=up NOC=up DIT=up 2
WMC=down LCOM=up RFC=up NOC=same DIT=same 2
WMC=down LCOM=same RFC=up NOC=up DIT=up 6
WMC=down LCOM=same RFC=down NOC=up DIT=up 5
WMC=same LCOM=same RFC=down NOC=up DIT=up 2

Size of set of large itemsets L(6): 4

Large Itemsets L(6):
CBO=down WMC=down LCOM=same RFC=up NOC=up DIT=up 3
CBO=down WMC=down LCOM=same RFC=down NOC=up DIT=up 4
CBO=down WMC=same LCOM=same RFC=down NOC=up DIT=up 2
CBO=same WMC=down LCOM=same RFC=up NOC=up DIT=up 2
```

# ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Παναγιώτης Γιαννάκης γεννήθηκε το 1986 στην Αθηνά το 2004 αποφοίτησε από το 3$^{ο}$ Λύκειο Αλίμου. Το 2008 ολοκλήρωσε τις προπτυχιακές του σπουδές στο τμήμα Διδακτικής της Τεχνολογίας και Ψηφιακών Συστημάτων με κατεύθυνση Ηλεκτρονικές Υπηρεσίες του Πανεπιστημίου Πειραιώς το 2010 ολοκλήρωσε τις μεταπτυχιακές του σπουδές στην Πληροφορική με Εξειδίκευση στο Λογισμικό του τμήματος της Πληροφορικής του Πανεπιστημίου Ιωαννίνων.

Τα ερευνητικά του ενδιαφέροντα σχετίζονται με τεχνολογία λογισμικού, αντικειμενοστρεφή σχεδίαση και σχεδιαστικά πρότυπα (Design Patterns).