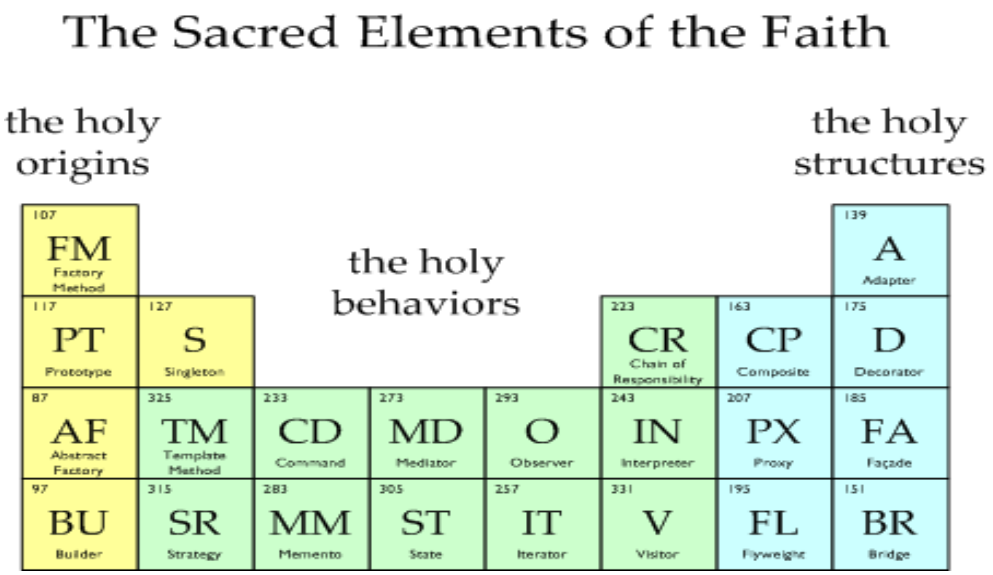
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

«ΕΜΠΕΙΡΙΚΗ ΜΕΛΕΤΗ ΤΗΣ ΕΠΙΔΡΑΣΗΣ ΤΩΝ ΠΡΟΤΥΠΩΝ ΣΧΕΔΙΑΣΗΣ ΣΤΑ ΣΦΑΛΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ ΠΑΙΧΝΙΔΙΩΝ»



Της φοιτήτριας
Αρβανίτου Ελβίρας-Μαρίας
Αρ. Μητρώου: 063067

Επιβλέπων Καθηγητής
Δεληγιάννης Ιγνάτιος

Θεσσαλονίκη 2011

ΠΡΟΛΟΓΟΣ

Η ανάπτυξη λογισμικού αποτελεί μια σύνθετη και πολύπλοκη διαδικασία. Στη πιο συχνή περίπτωση κατά την ολοκλήρωση κατασκευής του λογισμικού υπάρχουν αρκετά σφάλματα στον τρόπο λειτουργίας του, καθώς και αποκλίσεις από τις επιθυμητές απαιτήσεις. Μετά την ολοκλήρωση της ανάπτυξης ακολουθεί μια διαδικασία αποσφαλμάτωσης του λογισμικού (μέρος της διορθωτικής συντήρησης - corrective maintenance) η οποία έχει μεγάλο κόστος για τις επιχειρήσεις ανάπτυξης λογισμικού. Από τα παραπάνω συμπεραίνουμε ότι η ο αριθμός σφαλμάτων καθώς και ο ρυθμός διόρθωσης τους, αποτελούν βασικό παράγοντα της ποιότητας του λογισμικού.

Μια πολύ γνωστή τεχνική που φαίνεται να έχει θετική επίδραση στη ποιότητα λογισμικού είναι τα πρότυπα σχεδίασης. Παρά την πληθώρα της βιβλιογραφίας σχετικά με την επίδραση των προτύπων σχεδίασης σε μετρικές και χαρακτηριστικά αντικειμενοστραφούς ποιότητας, οι μελέτες σχετικά με την επίδραση της χρήσης προτύπων σχεδίασης στον αριθμό σφαλμάτων και στον ρυθμό αποσφαλμάτωσης του κώδικα είναι περιορισμένες.

Η διεξαγωγή της παρούσας μελέτης αποδείχθηκε ιδιαίτερα χρήσιμη για την εξοικείωση του συγγραφέα με προηγμένα θέματα τεχνολογίας λογισμικού, όπως τα πρότυπα σχεδίασης, καθώς και στη διενέργεια εμπειρικών μελετών με εκτεταμένη χρήση στατιστικής ανάλυσης. Επιπλέον, τα αποτελέσματα της εργασίας θεωρούνται σημαντικά και ευρύτερου επιστημονικού ενδιαφέροντος, από τη στιγμή που αξιολογούν τη χρήση των προτύπων σχεδίασης στη ποιότητα του λογισμικού, ζήτημα επίκαιρο στην κοινωνία ανάπτυξης λογισμικού. Τέλος τα ευρήματα της εργασίας χρησιμοποιήθηκαν ως μέρος ενός δημοσιευμένου επιστημονικού άρθρου, σε γνωστό συνέδριο του χώρου.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία ασχολείται με την μελέτη των συσχετίσεων μεταξύ της χρήσης προτύπων σχεδίασης και των σφαλμάτων στον κώδικα του λογισμικού. Για να επιτύχουμε τον προαναφερθέντα στόχο χρησιμοποιήσαμε μετρικές μεθόδους αξιολόγησης. Πιο συγκεκριμένα, διενεργήσαμε μια μελέτη περίπτωσης σε παιχνίδια ανοιχτού λογισμικού γραμμένα σε java. Αναλυτικότερα, εξετάσαμε αρκετά παιχνίδια ανοιχτού λογισμικού, εντοπίσαμε τον αριθμό των λαθών, τον βαθμό αποσφαλματωσης τους και στη συνέχεια εντοπίσαμε τα πρότυπα σχεδίασης που έχουν χρησιμοποιηθεί. Τα αποτελέσματα της μελέτης δείχνουν ότι ο συνολικός αριθμός στιγμιότυπων προτύπων σχεδίασης δεν συσχετίζετε με τον αριθμό των λαθών ούτε με την αποτελεσματικότητα της αποσφαλαμάτωσης. Όμως, αποδείχθηκε ότι συγκεκριμένα πρότυπα σχεδίασης εμφανίζουν σημαντική επίδραση και στους δυο προαναφερθέντες τομείς.

Επιπλέον, παρουσιάζουμε ένα παράδειγμα για το πως μπορούν τα πρότυπα σχεδίασης να είναι ως βάση για την ανάπτυξη λογισμικού. Πιο συγκεκριμένα, αναλύσαμε και σχεδιάσαμε ένα παιχνίδι που προσομοιώνει τη διαδικασία ανάπτυξης λογισμικού. Ο κύριος στόχος του λογισμικού είναι να βοηθήσει τους φοιτητές να κατανοήσουν βασικές αρχές της διοίκησης έργων λογισμικού.

ABSTRACT

In this thesis, we investigate the correlation between design pattern application and software defects. In order to achieve this goal we conducted an empirical study on java open source games. More specifically, we examined several successful open source games, identified the number of defects, the debugging rate and performed design pattern related measurements. The results of the study suggest that the overall number of design pattern instances is not correlated to defect frequency and debugging effectiveness. However, specific design patterns appear to have a significant impact on the number of reported bugs and debugging rate.

Additionally, we present an illustrative example on how design patterns can be used as a basis on software development. More specifically, we analyzed and design a computer game, which simulates the software development process. The main aim of this software will be to help students in understanding basic aspects of software project management.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω πολύ τον κ Αμπατζόγλου Απόστολο που από την πρώτη στιγμή της ανάθεσης της πτυχιακής μου ήταν δίπλα μου, με στήριξε και με βοήθησε πολύ στην υλοποίηση της πτυχιακής. Επίσης θα ήθελα να τον ευχαριστήσω γιατί μου έδωσε την ευκαιρία να συμμετέχω σε ένα άρθρο το οποίο δημοσιεύτηκε σε ένα επιστημονικό συνέδριο.

Επίσης, ένα ευχαριστώ και σε όσους ήταν δίπλα μου όλο αυτό τον καιρό της εκπόνησης της πτυχιακής μου εργασίας, στηρίζοντας και υπομένοντας εμένα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ – ΣΧΗΜΑΤΩΝ

1. ΕΙΣΑΓΩΓΗ

Η παρούσα πτυχιακή εργασία πραγματεύεται την επίδραση των προτύπων σχεδίασης στο πλήθος σφαλμάτων (bugs) και στον ρυθμό αποσφαλμάτωσης (debug rate) του λογισμικού. Για να μελετήσουμε το παραπάνω ερώτημα διενεργήσαμε μια μελέτη περίπτωσης σε έργα ανοιχτού λογισμικού. Για να επικυρώσουμε τα αποτελέσματα της προαναφερθείσας μελέτης, σχεδιάσαμε ένα εκπαιδευτικό παιχνίδι που στοχεύει στην προσομοίωση της λειτουργίας μιας εταιρίας παραγωγής λογισμικού. Στο διάγραμμα κλάσεων που παράχθηκε εντοπίσαμε πρότυπα σχεδίασης και αξιολογήσαμε ποιοτικά (qualitative evaluation) την επίδραση τους στο πλήθος σφαλμάτων και τον ρυθμό αποσφαλμάτωσης. Στη συνέχεια στο κεφάλαιο 1.1 γίνετε μια ιστορική αναδρομή στο ανοιχτό λογισμικό, ενώ στο κεφάλαιο 1.2, παρουσιάζουμε τα πρότυπα σχεδίασης που λάβαμε υπόψη στη μελέτη περίπτωσης.

1.1 Ανοιχτό Λογισμικό

Στα πρωταρχικά χρόνια του προγραμματισμού, το λογισμικό διανέμονταν μαζί με το πηγαίο κώδικα. Ειδικότερα, το λογισμικό ήταν άμεσα συνδεδεμένο με το υλικό του κατασκευαστή. Εξαιτίας της πολυπλοκότητας και του κόστους ανάπτυξης (καθώς και της σχετικά περιορισμένης ισχύος που διέθεταν οι πρώτοι υπολογιστές), των επιχειρηματικών μοντέλων των κατασκευαστών (μοντέλα που βασίζονταν στην πώληση του υλικού), καθώς και άλλων παραγόντων, οι χρήστες ελεύθερα διαμοιράζονταν τον πηγαίο κώδικα των εφαρμογών με συνεργατικό τρόπο, κάτι που οδήγησε στη δημιουργία ομάδων χρηστών, μερικές από τις οποίες είναι ακόμα ενεργές. Με την αποσύνδεση του υλικού από το λογισμικό, στις αρχές της δεκαετίας του 1970, εμφανίστηκαν στην αγορά τα πρώτα προϊόντα λογισμικού με τη μορφή “πακέτου”. Σταδιακά, λοιπόν, η διάδοση του πηγαίου κώδικα σταμάτησε να γίνεται ευρύτατα, καθώς οι πωλητές λογισμικού άρχισαν σταδιακά να αποκρύβουν τον κώδικα. Η εμπορευματοποίηση του λογισμικού που επικρατούσε παρείχε στους χρήστες κάποια οφέλη αλλά δεν υπήρχε παραμετροποίηση του λογισμικού στις ανάγκες του καθενός. Αντιλαμβανόμενος ο Richard Stallman αυτόν τον ανασταλτικό

παράγοντα στην ανάπτυξη του λογισμικού και στην πρόοδο της τεχνολογίας των υπολογιστών, ξεκίνησε το GNU project με σκοπό να παρέχει ελεύθερο UNIX. Στα χρόνια που προηγήθηκαν αυτού του βήματος, οι προγραμματιστές του GNU είχαν δημιουργήσει μια ποικιλία από καινοτόμα ανοιχτού κώδικα εργαλεία. Ο Richard Stallman ήταν και ο πρώτος που εισήγαγε τον όρο “ελεύθερο λογισμικό” (free software) και ίδρυσε το Ίδρυμα Ελεύθερου Λογισμικού (Free Software Foundation) το 1983 προκειμένου να βρεθεί ένας τρόπος διατήρησης της ελευθερίας των χρηστών να μελετούν, να κατανοούν και να τροποποιούν το λογισμικό, μέσα στο πλαίσιο του πνεύματος, που διέπει την κοινότητα των hackers και αφορά την απρόσκοπτη διάχυση των πληροφοριών. Αργότερα, ο οργανισμός “Open Source Iniative” πρότεινε τον όρο “λογισμικό ανοιχτού κώδικα” (open source software), πιθανότατα προκειμένου να αποφευχθεί η παρερμηνεία του αγγλικού όρου “free”, ο οποίος χρησιμοποιήθηκε από το FSF για την απόδοση της έννοιας της ελευθερίας και όχι της έννοιας δωρεάν.

Η δημιουργία του Linux το 1991, ενός ελεύθερου, συμβατού με το UNIX, λειτουργικού συστήματος, άλλαξε πολύ την στάση των προγραμματιστών προς τον κώδικα ανοιχτού λογισμικού. Ο δημιουργός του πυρήνα Linux, Linus Torvalds, πέτυχε τις προσδοκίες του, δημιουργώντας μια ομάδα προγραμματιστών και ατόμων που συνεισέφεραν στην ανάπτυξη μέσω του Internet. Ο Linus έδινε σε κυκλοφορία νέες εκδόσεις του λειτουργικού συστήματος και οι υπόλοιποι προγραμματιστές έκαναν αποσφαλμάτωση και πρόσθεταν νέες λειτουργίες. Ο Eric Raymond, ένας ευαγγελιστής του ανοιχτού-κώδικα, ονόμασε αυτή την προσέγγιση μοντέλο “Bazaar”, υποδηλώνοντας έτσι τη χαοτική αλλά αποτελεσματική οργάνωση των προγραμματιστών στο Internet, σε αντίθεση με τη παραδοσιακή προσέγγιση “Cathedral” που επικρατούσε μέχρι τότε. Παράλληλα, εξηγεί ότι η σταθερότητα των προγραμμάτων ανοιχτού-κώδικα είναι αποτέλεσμα του μεγάλου αριθμού προγραμματιστών και δοκιμαστών, καθώς οποιοδήποτε μεγάλο σφάλμα, φαίνεται μικρό κάτω από το βλέμμα πολλών ματιών.

Το κλειδί στο λογισμικό ανοιχτού-κώδικα είναι η διαθεσιμότητα του πηγαίου κώδικα, καθώς επιτρέπει στους χρήστες με ειδικές προγραμματιστικές ικανότητες να το προσαρμόσουν στις προσωπικές τους απαιτήσεις και να εξαλείψουν τα σφάλματα

που εμφανίζονται στη χρήση. Η ευρέως διαδεδομένη χρήση της άδειας ελεύθερου λογισμικού, GNU General Public License, εγγυάται ότι ο καθένας μπορεί να τροποποιήσει και να αναδιανείμει προγράμματα, έχοντας ως προϋπόθεση ότι δεν εμποδίζει άλλους από το να πράξουν το ίδιο.

Στις μέρες μας υπάρχουν αναρίθμητες εφαρμογές και προγράμματα ανοικτού-κώδικα. Γνωστά παραδείγματα είναι το λειτουργικό σύστημα Linux, ο Apache Server, η Perl, το Mozilla κ.α. Μερικά από τα προγράμματα αυτά βγάζουν εκτός συναγωνισμού εμπορικά προγράμματα αντίστοιχου είδους και πολλές εταιρίες όπως η Apple και η Sun Microsystems κάνουν χρήση των προγραμμάτων αυτών.

Δίνοντας έναν ορισμό για το ανοιχτό λογισμικό, θα πρέπει να αναφέρουμε το σύνολο των συνθηκών στο οποίο βασίζεται:

- Ελεύθερη Αναδιανομή
- Πηγαίος Κώδικας: Το πρόγραμμα που διατίθεται πρέπει να περιλαμβάνει τον πηγαίο κώδικα, ενώ συγχρόνως πρέπει να επιτρέπεται η διάθεσή του είτε ως πηγαίος κώδικας είτε σε μεταγλωττισμένη μορφή.
- Παραγόμενα Έργα: Η άδεια χρήσης πρέπει να επιτρέπει τροποποιήσεις του προγράμματος, καθώς και πιθανά παραγόμενα έργα, τα οποία πρέπει να διανέμονται με τους ίδιους όρους που διέπουν το αρχικό λογισμικό.
- Ακεραιότητα του πηγαίου κώδικα του συγγραφέα
- Καμία διάκριση εναντίον ατόμων ή ομάδων ατόμων
- Καμία διάκριση εναντίον κάποιων τομέων δραστηριοποίησης: Για παράδειγμα, δεν μπορεί να περιορίζει τη χρήση του προγράμματος για την εξυπηρέτηση των αναγκών μιας επιχείρησης ή μιας ερευνητικής ομάδας που εξετάζει ζητήματα γενετικής.
- Διανομή της άδειας χρήσης
- Η άδεια χρήσης δεν πρέπει να αφορά μόνο ένα συγκεκριμένο προϊόν
- Η άδεια χρήσης δεν πρέπει να περιορίζει άλλα λογισμικά
- Η άδεια χρήσης πρέπει να είναι τεχνολογικά ουδέτερη: Κανένας όρος της άδειας χρήσης δεν πρέπει να επιβάλλει τη χρήση συγκεκριμένων τεχνολογιών ή διεπαφών.

1.2 Πρότυπα Σχεδίασης

Στα τέλη της δεκαετίας του 1970 ένας αρχιτέκτονας ονόματι Christopher Alexander επιχείρησε να βρει και να καταγράψει αποδεδειγμένα ποιοτικούς σχεδιασμούς στον τομέα των κατασκευών. Έτσι μελέτησε πολλές διαφορετικές κατασκευές που εξυπηρετούσαν τον ίδιο σκοπό και προσπάθησε να ανακαλύψει κοινά στοιχεία, τα οποία κατηγοριοποίησε σε σχεδιαστικά πρότυπα. Το 1987 ο Kent Beck και ο Ward Cunningham άρχισαν να πειραματίζονται με την ιδέα της εύρεσης σχεδιαστικών προτύπων, η οποία εφαρμόστηκε για πρώτη φορά στη μηχανική λογισμικού και μέχρι τα μέσα της δεκαετίας του '90 η εν λόγω έννοια είχε καθιερωθεί και εξαπλωθεί, προσανατολισμένη πλέον προς τον αντικειμενοστρεφή προγραμματισμό. Στο (Gamma et al., 1995) καταγράφονται 23 πρότυπα που επιλύουν συνήθη προβλήματα στη σχεδίαση λογισμικού.

Έχουν οριστεί διάφορες κατηγορίες προτύπων, για διαφορετικά προβλήματα και κάθε κατηγορία περιλαμβάνει πολλαπλά στοιχεία. Έτσι, τα σχεδιαστικά πρότυπα κατηγοριοποιούνται σε κατασκευαστικά , δομικά και συμπεριφορικά.

- Κατασκευαστικά πρότυπα (Creational Patterns): Τα κατασκευαστικά πρότυπα αφορούν τυποποιημένους τρόπους δυναμικής κατασκευής αντικειμένων κατά τον χρόνο εκτέλεσης. Απώτερος στόχος τους είναι η ανεξαρτητοποίηση του κώδικα που χρησιμοποιεί κάποια αντικείμενα από τις κλάσεις που ορίζουν τα αντικείμενα αυτά και τον τρόπο που κατασκευάζονται στη μνήμη, σύμφωνα με την αρχή ανοιχτής-κλειστής σχεδίασης για ορθή αντικειμενοστρεφή σχεδίαση. Τέτοια κατασκευαστικά πρότυπα είναι το Factory, το Singleton και το Prototype.
- Δομικά πρότυπα (Structural Patterns): Τα δομικά πρότυπα αφορούν τυποποιημένους τρόπους δυναμικής κατασκευής σύνθετων αντικειμένων τα οποία χρησιμοποιούν υπάρχουσες ιεραρχίες κλάσεων. Τέτοια δομικά πρότυπα είναι το Adapter και το Decorator.
- Συμπεριφορικά πρότυπα (Behavioural Patterns): Τα συμπεριφορικά πρότυπα αφορούν τον καταμερισμό αρμοδιοτήτων σε διάφορες κλάσεις και τον ορισμό

του τρόπου επικοινωνίας μεταξύ των αντικειμένων τους κατά τον χρόνο εκτέλεσης. Σε αντίθεση με τα δομικά πρότυπα, τα συμπεριφορικά βρίσκουν εφαρμογή στον αρχικό σχεδιασμό μίας ιεραρχίας κλάσεων και όχι στην εκ των υστέρων επέκταση κάποιας υπάρχουσας ιεραρχίας. Τέτοια συμπεριφορικά πρότυπα είναι το Strategy, το State, το Observer, το Template και το Visitor.

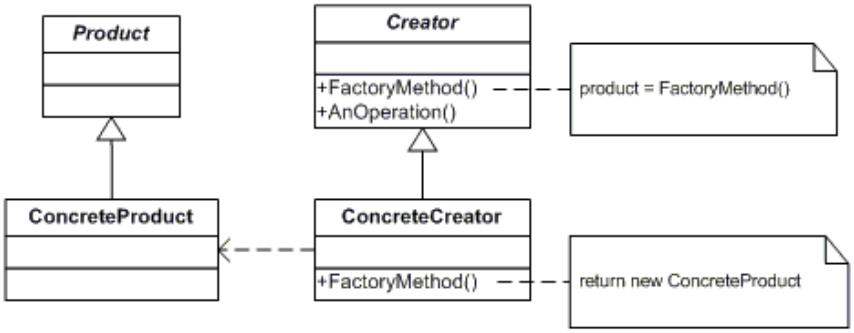
1.2.1 Πρότυπο Factory

Το πρότυπο αυτό έχει ως σκοπό την παροχή μιας διασύνδεσης για τη δημιουργία οικογενειών συσχετιζόμενων ή εξαρτημένων αντικειμένων χωρίς να προσδιορίζεται η συγκεκριμένη κλάση τους (Chatzigeorgiou, 2005). Οι μέθοδοι του Factory, όταν καλούνται, κατασκευάζουν ένα νέο αντικείμενο κατάλληλου τύπου και επιστρέφουν έναν αφηρημένο δείκτη προς αυτό, δηλαδή έναν δείκτη τύπος του οποίου είναι η γενική διασύνδεση, οπότε το εξωτερικό πρόγραμμα μπορεί να τις καλεί για να λαμβάνει το ζητούμενο κατά περίπτωση αντικείμενο χωρίς το ίδιο να χρειάζεται να γνωρίζει κάθε παραγόμενο τύπο δεδομένων που υλοποιεί τη διασύνδεση. Με αυτόν τον τρόπο το πρόγραμμα είναι κλειστό ως προς πιθανές επεκτάσεις και μόνο το Factory είναι ανοιχτό ως προς αυτές, καθώς μόνον ο δικός του κώδικας χρειάζεται να τροποποιηθεί σε περίπτωση π.χ. προσθήκης μίας νέας παραγόμενης κλάσης. Το Factory συνήθως παρέχει μία μέθοδο για κάθε δυνατό παραγόμενο τύπο, αλλά μία παραλλαγή ονόματι παραμετροποιημένο Factory περιέχει μία μοναδική μέθοδο η οποία επιλέγει το αντικείμενο που θα δημιουργήσει και θα επιστρέψει, αναλόγως με την τιμή ενός ορίσματος που δέχεται. Το όρισμα αυτό μπορεί π.χ. να διαβάζεται από κάποιο αρχείο ρυθμίσεων ή να μεταβιβάζεται ως όρισμα γραμμής εντολών στο εξωτερικό πρόγραμμα (στον πελάτη), έτσι ώστε το τελευταίο να είναι κλειστό ως προς το σύνολο των παραγόμενων κλάσεων και να μη χρειάζεται επαναμεταγλώττιση σε περίπτωση που τροποποιηθεί το σύνολο αυτό.

Ένας εναλλακτικός τύπος Factory είναι όταν ορίζεται ως αφηρημένη κλάση και η παρεχόμενη μέθοδος κατασκευής αντικειμένων υποσκελίζεται από υποκλάσεις του, έτσι ώστε η καθεμία από τις τελευταίες να κατασκευάζει αντικείμενο διαφορετικού τύπου. Σε κάθε περίπτωση στόχος είναι να μπορεί το πρόγραμμα να δημιουργεί στιγμιότυπα κλάσεων χωρίς να προσδιορίζει ρητά τον ακριβή τύπο τους και

αφήνοντας το Factory να τον αποφασίσει εσωτερικά το πρόγραμμα αρκεί να έχει γνώση του γενικού αφηρημένου τύπου.

Γενική Δομή

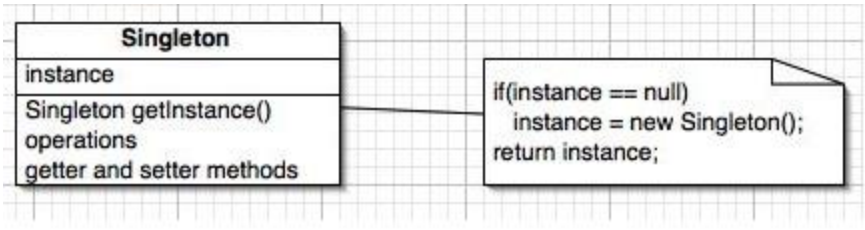


Σχήμα 1: Διάγραμμα κλάσεων προτύπου Factory

1.2.2 Πρότυπο Singleton

Το πρότυπο Singleton είναι ένα σχεδιαστικό πρότυπο το οποίο εξασφαλίζει ότι μια κλάση θα έχει μόνο ένα στιγμιότυπο και παρέχει ένα καθολικό σημείο πρόσβασης σε αυτό (Chatzigeorgiou, 2005). Με το πρότυπο Singleton η αρμοδιότητα για την ικανοποίηση αυτού του περιορισμού ανατίθεται στην ίδια την κλάση Α και δε μεταβιβάζεται στο εξωτερικό πρόγραμμα που τη χρησιμοποιεί. Αυτό γίνεται μέσω μίας στατικής δημόσιας μεθόδου της Α η οποία δημιουργεί το πρώτο στιγμιότυπο της κλάσης και ακολούθως, σε κάθε επόμενη κλήση της, επιστρέφει έναν δείκτη ή μία αναφορά προς αυτό. Το Singleton διαφέρει από μία απλή καθολική μεταβλητή (global variable), η οποία επίσης δημιουργείται αναγκαστικά μόνο μία φορά καθ' όλο το χρόνο εκτέλεσης του προγράμματος, καθώς το αντικείμενο Singleton δε δεσμεύει μνήμη μέχρι τη στιγμή της πρώτης κλήσης της μεθόδου κατασκευής. Από εκεί κι έπειτα όμως το αντικείμενο παραμένει στη μνήμη ως τον τερματισμό του προγράμματος γιατί είναι υλοποιημένο ως στατική μεταβλητή. Το πρότυπο Singleton χρησιμοποιείται όταν σε κάποιο σύστημα λογισμικού υπάρχει η απαίτηση από μια κλάση να δημιουργείται ένα και μόνο αντικείμενο.

Γενική Δομή



Σχήμα 2: Διάγραμμα κλάσεων προτύπου Singleton

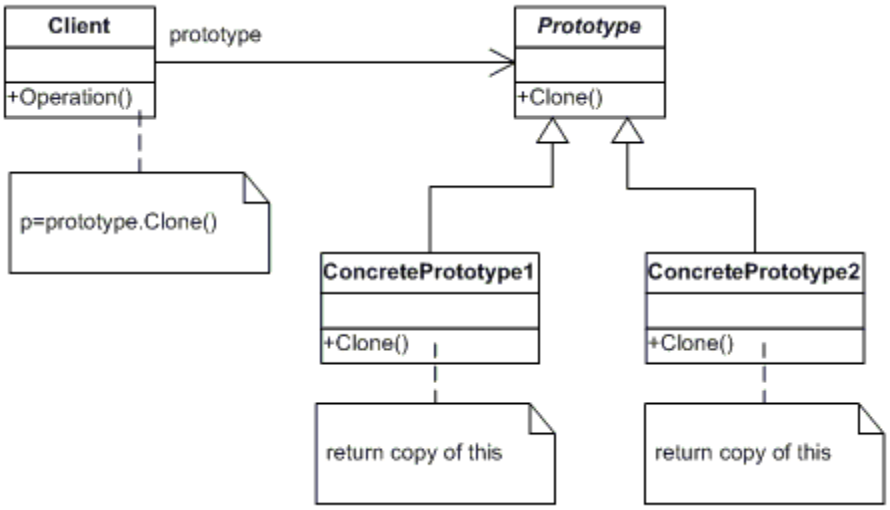
1.2.3 Πρότυπο Prototype

Το πρότυπο Prototype είναι ένα σχεδιαστικό πρότυπο το οποίο προσδιορίζει τους τύπους των αντικειμένων, δημιουργεί στιγμιότυπα χρησιμοποιώντας πρωτότυπα με σκοπό να δημιουργεί νέα αντικείμενα αντιγράφοντας αυτά τα πρότυπα. Το Prototype κατασκευάζει νέα αντικείμενα με "κλωνοποίηση" κάποιου υπάρχοντος. Η κλωνοποίηση αυτή γίνεται μέσω μίας μεθόδου clone() η οποία παρέχεται από μία αφηρημένη κλάση ή διασύνδεση A και υλοποιείται σε κάθε παραγόμενη κλάση B η οποία κληρονομεί την A. Έτσι η κλήση της clone() σε ένα στιγμιότυπο της B επιστρέφει ένα αντίγραφο του εν λόγω στιγμιότυπου, το οποίο αναλόγως με την υλοποίηση μπορεί είτε να περιέχει δείκτες προς τις εσωτερικές δομές δεδομένων του αρχικού στιγμιότυπου, είτε να περιέχει πλήρη, νεοδημιουργηθέντα αντίγραφα αυτών των δομών δεδομένων.

Το Prototype χρησιμοποιείται για την αποφυγή δημιουργίας υποκλάσεων ενός αντικειμένου δημιουργού στην εφαρμογή και την αποφυγή του κόστους κληρονομικότητας για την δημιουργία ενός νέου αντικειμένου με τον συμβατικό τρόπο (Gamma et al., 1995). Η μέθοδος clone() μπορεί να επικαλύπτει την κλήση του εκάστοτε κατασκευαστή αντιγράφου με ένα κοινό επίπεδο αφαίρεσης έτσι ώστε να μη

χρειάζεται το εξωτερικό πρόγραμμα να γνωρίζει όλους τους παραγόμενους τύπους δεδομένων που υλοποιούν τη διασύνδεση A, καθώς η clone() επιστρέφει αναφορά του αφηρημένου τύπου A.

Γενική Δομή



Σχήμα 3: Διάγραμμα κλάσεων προτύπου Prototype

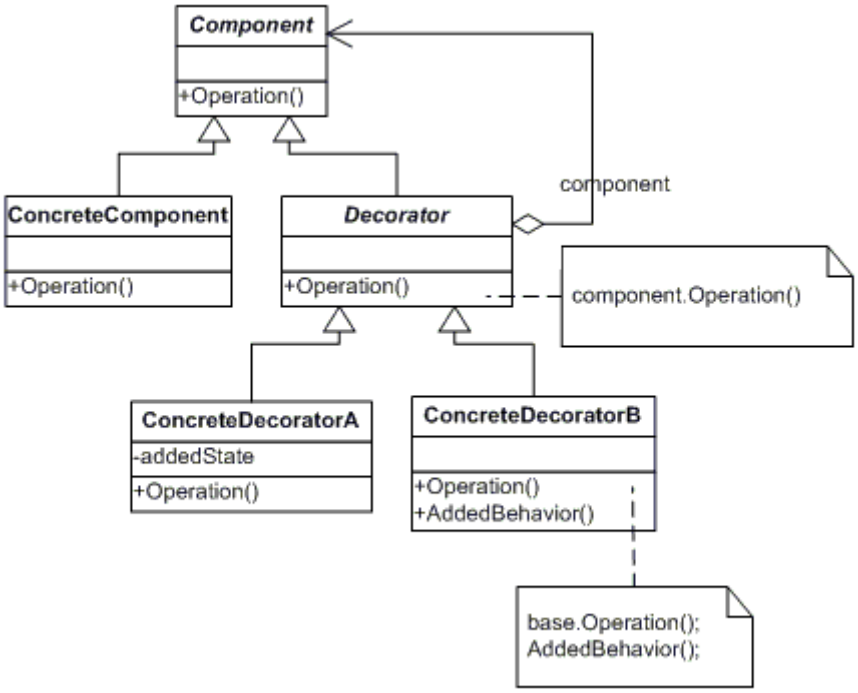
1.2.4 Πρότυπο Decorator

Το πρότυπο αυτό επιτρέπει την εύκολη και δυναμική επέκταση της λειτουργικότητας κάποιων υπαρχόντων κλάσεων A, B κλπ, οι οποίες υλοποιούν την ίδια διασύνδεση ή κληρονομούν την ίδια αφηρημένη κλάση (έστω Interface), σε χρόνο εκτέλεσης. Αυτό επιτυγχάνεται μέσω μίας νέας κλάσης Decorator, η οποία επίσης υλοποιεί την Interface αλλά περιέχει ως ιδιωτικό πεδίο και μία αναφορά σε ένα στιγμιότυπο του γενικού τύπου Interface (έστω το instance), η οποία τυπικά μεταβιβάζεται ως όρισμα στον κατασκευαστή της Decorator. Έτσι οι μέθοδοι της τελευταίας υλοποιούν εσωτερικά την καινούργια λειτουργικότητα αλλά για τις κοινές εργασίες καλούν τις αντίστοιχες μεθόδους του instance. Κατά τον χρόνο εκτέλεσης θα μπορούσε το αντικείμενο Decorator να κατασκευάζεται με όρισμα οποιοδήποτε

στιγμιότυπο τύπου Interface (ακόμα και του ίδιου του Decorator, αν και αυτό δε θα είχε ιδιαίτερο νόημα) ώστε κατά περίπτωση το αντικείμενο να παρέχει τη λειτουργικότητα οποιασδήποτε κλάσης τύπου Interface, είτε της Α είτε κάποιας άλλης, επεκτεταμένης με ένα συγκεκριμένο σύνολο δυνατοτήτων. Με αυτόν τον τρόπο γίνεται εφικτός ένας δυναμικός συνδυασμός λειτουργιών από στοιχειώδεις δομικούς λίθους κατά τον χρόνο εκτέλεσης.

Η εναλλακτική λύση, χωρίς χρήση κάποιου σχεδιαστικού προτύπου, θα ήταν η απλή κληρονομικότητα, με τον ορισμό κλάσεων οι οποίες επεκτείνουν τις Α, Β κλπ. και προσθέτουν τη νέα λειτουργικότητα. Ωστόσο η λύση αυτή δεν είναι εφικτή σε περίπτωση που οι Α, Β κλπ. δεν μπορούν να επεκταθούν με κληρονομικότητα (π.χ. αν δηλωθούν ως τελικές κλάσεις στην Java), ενώ σε άλλες περιπτώσεις δεν είναι καθόλου πρακτική, π.χ. αν έχουμε πολλαπλά διαφορετικά σύνολα νέων δυνατοτήτων τα οποία πρέπει να συνδυαστούν με τις Α, Β κλπ. Το πρόβλημα έγκειται στο ότι με την κληρονομικότητα όλοι οι πιθανοί συνδυασμοί δυνατοτήτων πρέπει να προβλεφθούν και να ληφθούν υπ’ όψιν κατά τη συγγραφή του προγράμματος. Αντιθέτως με την κλάση Decorator, η οποία δρα ως περίβλημα (wrapper) άλλων αντικειμένων τύπου Interface προς τα οποία περιέχει αναφορές / δείκτες, η σύνθεση νέων αντικειμένων ουσιαστικά γίνεται δυναμικά ενώ το πρόγραμμα εκτελείται (Gamma et al., 1995).

Γενική Δομή



Σχήμα 4: Διάγραμμα κλάσεων προτύπου Component

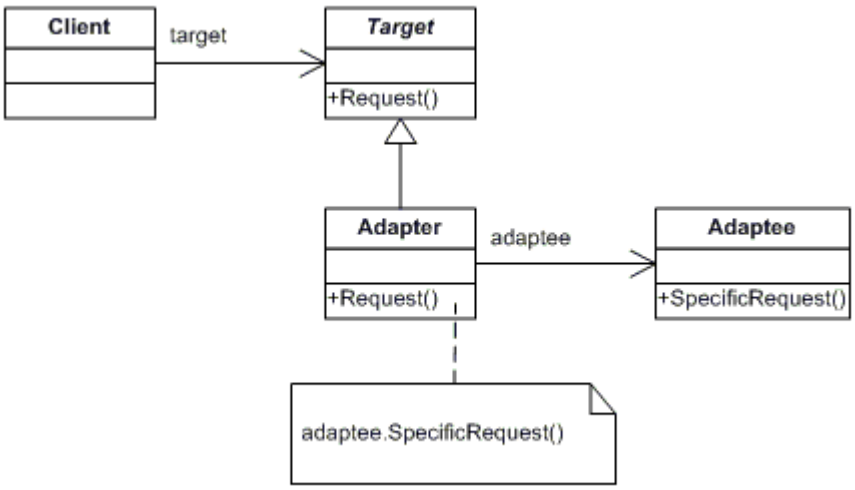
1.2.5 Πρότυπο Adapter

Το πρότυπο αυτό επιτρέπει την προσαρμογή της διασύνδεσης που εξάγει μία κλάση Α σε μία πρότυπη διασύνδεση, έστω Interface, που αναμένεται από ένα εξωτερικό πρόγραμμα ώστε να μην παραβιάζεται η αρχή ανοιχτής-κλειστής σχεδίασης. Αυτό γίνεται μέσω μίας ενδιάμεσης κλάσης, του Adapter, η οποία υλοποιεί το Interface και ταυτοχρόνως είτε κληρονομεί την Α (υλοποίηση με κληρονομικότητα), είτε περιέχει μία ιδιωτική αναφορά σε αντικείμενο τύπου Α (υλοποίηση με συνάθροιση). Σε κάθε περίπτωση ο Adapter έχει πρόσβαση στις μεθόδους της Α και οι δικές του μέθοδοι, οι υπογραφές των οποίων συμφωνούν με αυτές που αναμένονται από κάποιον πελάτη καθώς προδιαγράφονται από τη διασύνδεση / αφηρημένη κλάση Interface, τις καλούν εσωτερικά και επιστρέφουν τα αποτελέσματα, αποκρύπτοντας παράλληλα τη διαδικασία αυτή από το εκάστοτε πρόγραμμα πελάτη. Το πρότυπο αυτό χρησιμοποιείται όταν θέλουμε να χρησιμοποιήσουμε μια

υπάρχουσα κλάση, αλλά η διασύνδεσή της δεν συμβαδίζει με τις υπάρχουσες ανάγκες (Chatzigeorgiou, 2005).

Η υλοποίηση του προτύπου Adapter μέσω κληρονομικότητας επιτρέπει την προσαρμογή στη ζητούμενη διασύνδεση και των προστατευμένων μεθόδων της κλάσης Α, αντί μόνο των δημοσίων της, ενώ είναι και ελαφρώς πιο αποδοτική από την υλοποίηση με συνάθροιση, αφού κάθε κλήση στον Adapter «μεταφράζεται» σε κλήση σε μια άλλη μέθοδο του ιδίου αντί για κλήση σε μέθοδο άλλου αντικειμένου. Από την άλλη η υλοποίηση με συνάθροιση δεν αντιμετωπίζει προβλήματα ακόμα και αν η Α δεν μπορεί να κληρονομηθεί, ενώ επιτρέπει την προσαρμογή όχι μόνο της Α αλλά και κάθε υποκλάσης της λόγω του πολυμορφισμού.

Γενική Δομή



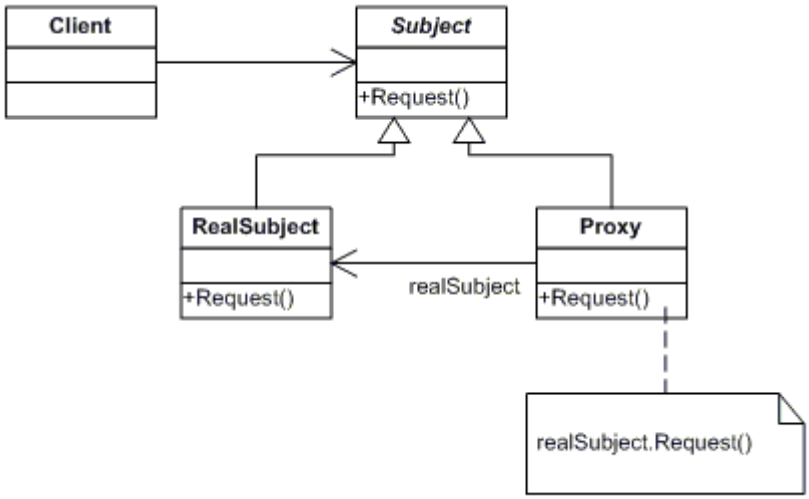
Σχήμα 5: Διάγραμμα κλάσεων προτύπου Adapter

1.2.6 Πρότυπο Proxy

Όταν η πρόσβαση ενός εξωτερικού προγράμματος σε αντικείμενα κάποιας συγκεκριμένης κλάσης (έστω της A) πρέπει να είναι ελεγχόμενη και να πληροί ορισμένες προϋποθέσεις (π.χ., για λόγους εξοικονόμησης μνήμης, να μη φορτώνεται πραγματικά η A μέχρι να κληθεί μία μέθοδος της), το πρότυπο Proxy παρέχει έναν τυποποιημένο τρόπο ώστε αυτό να γίνεται διατηρώντας την κλειστότητα του εξωτερικού προγράμματος ως προς την εν λόγω κλάση και τον τρόπο λειτουργίας της· ακόμα και αν η υλοποίηση της αλλάξει ο πελάτης είναι αδιάφορος απέναντι σε αυτές τις αλλαγές (δε χρειάζεται τροποποίηση) χάρη σε ένα επίπεδο αφαίρεσης που του παρέχεται από μία ενδιάμεση κλάση, τον Proxy, η οποία παίζει το ρόλο του μεσολαβητή πρόσβασης. Ο Proxy είναι που εκτελεί όλους τους απαραίτητους ελέγχους και προσπελαύνει πραγματικά τα αντικείμενα, ενώ ταυτόχρονα υλοποιεί την ίδια διασύνδεση με την A κι έτσι ο πελάτης, ο οποίος κατέχει μία αναφορά προς στιγμιότυπο του Proxy αντί της A, δεν αντιλαμβάνεται καν τη μεσολάβησή του· ο Proxy ουσιαστικώς εικονικοποιεί την πρόσβαση στη ζητούμενη κλάση. Συνήθως κατασκευάζεται, αντί για την A, από ένα Factory και εντελώς διαφανώς για το εξωτερικό πρόγραμμα, ενώ δεν είναι σπάνιο να περιέχει μία αναφορά στο πραγματικό στιγμιότυπο της A ως ιδιωτικό πεδίο.

Το σχεδιαστικό αυτό πρότυπο όταν αναπαριστά ένα μεγάλο αντικείμενο, μέχρις ότου ζητηθεί από την εφαρμογή να φορτωθεί το ίδιο το αντικείμενο, επιτυγχάνεται μεγαλύτερη ταχύτητα και λιγότερη φόρτωση της μνήμης της εφαρμογής που χρησιμοποιεί το πρότυπο (Gamma et al., 1995).

Γενική Δομή



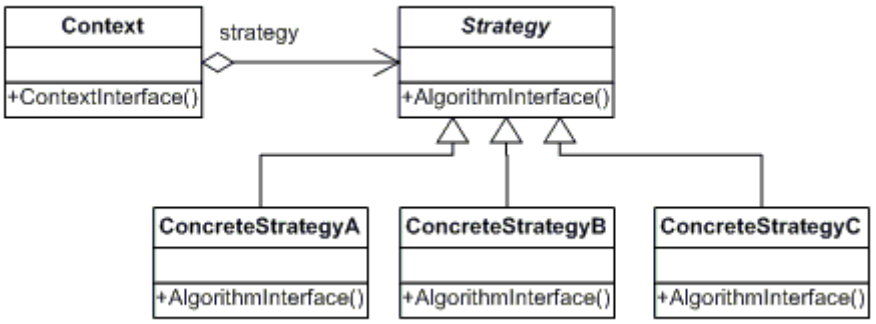
Σχήμα 6: Διάγραμμα κλάσεων προτύπου Proxy

1.2.7 Πρότυπο Strategy

Το πρότυπο αυτό ενθυλακώνει την κατάσταση ενός αντικειμένου, ώστε να μπορεί να αλλάξει τη συμπεριφορά του, όταν αλλάξει η εσωτερική κατάσταση του αντικειμένου (Gamma et al., 1995). Όταν είναι διαθέσιμοι πολλαπλοί τρόποι επίλυσης ενός προβλήματος (π.χ. διαφορετικοί αλγόριθμοι), είναι προτιμότερο ο καθένας από αυτούς να μην υλοποιείται μέσα στις κλάσεις-πελάτες που τον χρησιμοποιούν (π.χ. ως ιδιωτική μέθοδος), έτσι ώστε οι πελάτες να έχουν μικρότερη περιπλοκότητα και οι διάφοροι αλγόριθμοι να είναι επαναχρησιμοποιήσιμοι και προσπελάσιμοι από πολλαπλά εξωτερικά προγράμματα. Με το πρότυπο Strategy όλοι οι διαφορετικοί αλγόριθμοι ορίζονται ως ξεχωριστές κλάσεις που υλοποιούν μία κοινή διασύνδεση Α και οι πελάτες διατηρούν ως πεδίο μία αναφορά προς τον αφηρημένο τύπο Α. Στο πεδίο αυτό δίνεται τιμή μέσω του ορίσματος κάποιας μεθόδου του πελάτη (πιθανώς του κατασκευαστή του), έτσι ώστε η ταυτότητα του εκάστοτε χρησιμοποιούμενου αλγορίθμου από όλους τους διαθέσιμους για την εκτέλεση της ζητούμενης εργασίας

να είναι εύκολα παραμετροποιημένη, με μία απλή αλλαγή αυτού του ορίσματος κατά την κλήση της προαναφερθείσας μεθόδου.

Γενική Δομή



Σχήμα 7: Διάγραμμα κλάσεων προτύπου Strategy

1.2.8 Πρότυπο Composite

Το πρότυπο σχεδίασης Composite επιτρέπει τη σύνθεση αντικειμένων σε δενδρικές δομές για την αναπαράσταση ιεραρχιών τμήματος όλου. Έτσι επιτρέπει στα προγράμματα πελάτες να διαχειρίζονται με ενιαίο τρόπο τόσο τα ανεξάρτητα αντικείμενα, όσο και σύνθετα αντικείμενα.

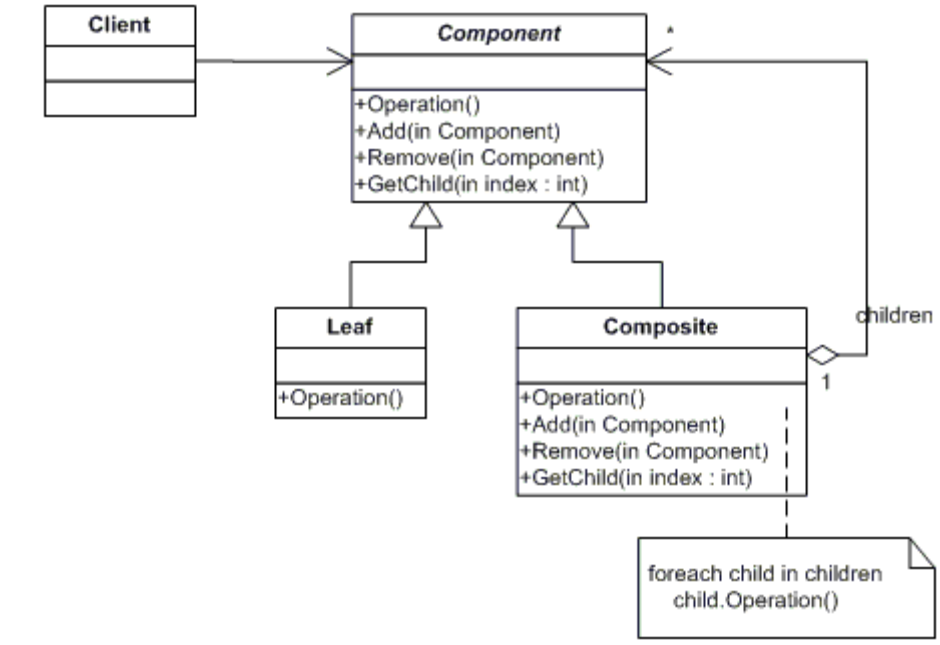
Το πρότυπο σχεδίασης Composite δίνει λύσεις με κομψό τρόπο σε προβλήματα χρήσης σχέσεων περιεκτικότητας μεταξύ της κλάσης που αντιπροσωπεύει το όλον(περικλείουσα κλάση) και των κλάσεων που αντιπροσωπεύουν τα τμήματα.

Το σημείο κλειδί στο πρότυπο είναι η ύπαρξη μιας αφηρημένης κλάσης που αναπαριστά τόσο τις πρωταρχικές όσο και τις περικλείουσες κλάσεις. Έτσι είναι δυνατή η δημιουργία οποιουδήποτε πρωταρχικού ή σύνθετου αντικειμένου επιτρέποντας ομοιόμορφο χειρισμό των αντικειμένων από ένα μόνο πρόγραμμα πελάτη.

Ο χρήστης είναι σε θέση να δημιουργήσει οποιαδήποτε σύνθετη οντότητα και να την προσθέσει στην εφαρμογή. Η σχεδίαση ενός σύνθετου αντικειμένου ουσιαστικά συνιστάται στη σχεδίαση των επιμέρους τμημάτων του. Για το λόγο αυτό

είναι επιθυμητή η ενιαία αντιμετώπιση όλων των αντικειμένων. Το πρότυπο Composite, επιτρέπει τον αναδρομικό ορισμό περιεκτικότητας, ώστε οι πελάτες να μην αντιλαμβάνονται τη διαφορά μεταξύ πρωταρχικών και σύνθετων αντικειμένων(Chatzigeorgiou,2005).

Γενική Δομή



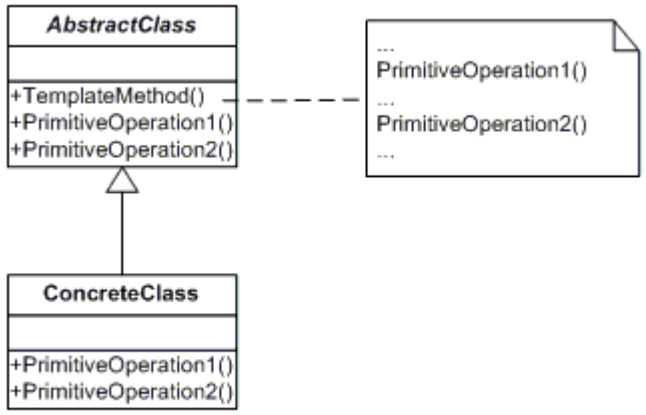
Σχήμα 8: Διάγραμμα κλάσεων προτύπου Composite

1.2.9 Πρότυπο Template Method

Το πρότυπο αυτό είναι εκλέπτυνση του Strategy για την περίπτωση που κάθε αλγόριθμος έχει πολλαπλές παραλλαγές οι οποίες διαφέρουν σε ορισμένα βήματα αλλά κάποια άλλα σημεία τους είναι κοινά. Τότε για κάθε αλγόριθμο μπορούμε να ορίσουμε ένα στοιχείο αφαίρεσης Β το οποίο υλοποιεί τη διασύνδεση Α και με τη σειρά του κληρονομείται από πολλές παραλλαγές του αλγορίθμου. Το κάθε Β περιέχει την υλοποίηση των αμετάβλητων βημάτων και, στα σημεία που οι παραλλαγές διαφέρουν, καλεί αφηρημένες προστατευμένες μεθόδους. Οι μέθοδοι αυτές υλοποιούνται διαφορετικά σε κάθε παραλλαγή που κληρονομεί το Β. Το πρότυπο

χρησιμοποιείται για τον ορισμό των αμετάβλητων τμημάτων και τη μετάθεση της υλοποίησης των μεταβλητών τμημάτων του αλγόριθμου σε παράγωγες κλάσεις (Chatzigeorgiou, 2005).

Γενική Δομή



Σχήμα 9: Διάγραμμα κλάσεων προτύπου Template Method

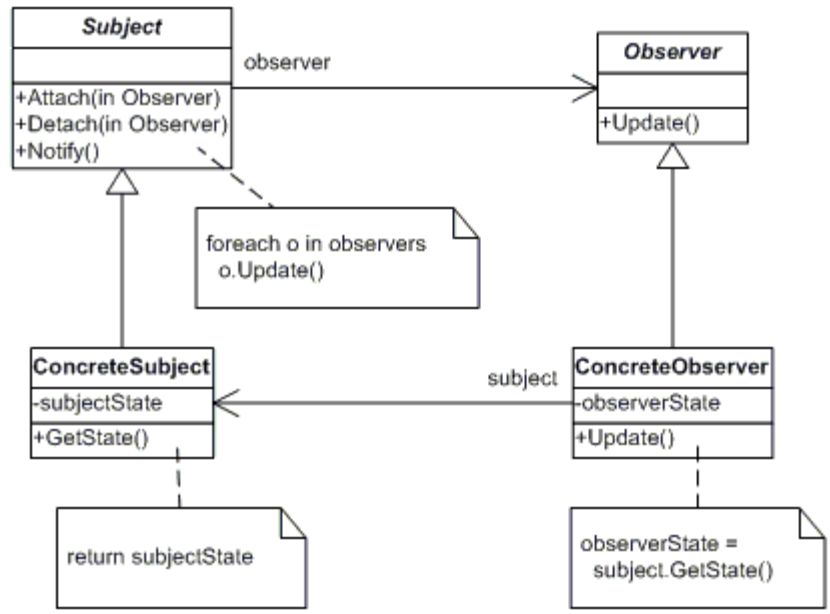
1.2.10 Πρότυπο Observer

Σε περιπτώσεις που κάποια αντικείμενα (παρατηρητές) ενδιαφέρονται να λαμβάνουν ειδοποιήσεις για τυχόν αλλαγές στην κατάσταση κάποιου άλλου αντικειμένου Α (π.χ. τροποποιήσεις τιμών κάποιων πεδίων του), υπάρχουν δύο προσεγγίσεις για την υλοποίηση αυτών των ενημερώσεων: είτε τις κατάλληλες στιγμές το Α να καλεί προκαθορισμένες μεθόδους των παρατηρητών και να τους μεταβιβάζει έτσι δεδομένα, είτε οι παρατηρητές να καλούν μία μέθοδο του Α για να λαμβάνουν κατά βούληση πληροφορίες. Η ενδιαφέρουσα περίπτωση είναι η πρώτη καθώς μόνον το Α γνωρίζει πότε υπάρχουν αλλαγές στην κατάσταση του και τέτοιες αλλαγές είναι που πρέπει να πυροδοτούν τις ενημερώσεις. Ο απλούστερος τρόπος είναι να διατηρεί μία συνδεδεμένη λίστα με όλους τους παρατηρητές που κατά καιρούς έχουν εκδηλώσει ενδιαφέρον και να καλεί τις κατάλληλες μεθόδους τους τις κατάλληλες στιγμές. Όμως αυτή η λύση προκαλεί προβλήματα κλειστότητας καθώς

ένας άλλος τύπος παρατηρητή μπορεί να έχει διαφορετικές μεθόδους, ενώ το Α παρουσιάζει υψηλή σύζευξη με άλλες κλάσεις.

Το πρότυπο Observer δίνει μία λύση σε αυτό το πρόβλημα: ορίζει μία διασύνδεση Observable, η οποία περιέχει μία μέθοδο register (για την εκούσια δήλωση παρατηρητών) και μία μέθοδο unregister (για την εκούσια αποχώρηση παρατηρητών), και μία διασύνδεση Observer, με μία μέθοδο synchronize() για τη μεταβίβαση ενημερωμένων δεδομένων. Η κλάση κάθε αντικειμένου Α πρέπει να υλοποιεί τη διασύνδεση Observable και η κλάση κάθε παρατηρητή τη διασύνδεση Observer. Ένα αντικείμενο Observable διατηρεί μία λίστα με αντικείμενα του αφηρημένου τύπου Observer τα οποία έχουν καλέσει τη μέθοδο register του εν λόγω αντικειμένου. Οι εγγραφές αυτές χρησιμοποιούνται τις κατάλληλες χρονικές στιγμές για την ειδοποίηση των παρατηρητών. Τέλος αξίζει να σημειωθεί, ότι η λίστα των παρατηρητών μπορεί να αλλάζει κατά την διάρκεια εκτέλεσης του προγράμματος (Chatzigeorgiou, 2005).

Γενική Δομή



Σχήμα 10: Διάγραμμα κλάσεων προτύπου Observer

2. ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΑΝΑΣΚΟΠΗΣΗ

Στο δεύτερο κεφάλαιο της εργασίας παρουσιάζουμε τη βιβλιογραφική ανασκόπηση σχετικά με τους βασικού όρους τεχνολογίας λογισμικού που εμπλέκονται στη πτυχιακή εργασία. Πιο συγκεκριμένα στο κεφάλαιο 2.1, παρουσιάζουμε μια βιβλιογραφική αναφορά σχετικά με τα σφάλματα λογισμικού (software defects), στο κεφάλαιο 2.2 γίνετε συνοπτική αναφορά στην επίδραση των προτύπων σχεδίασης στη ποιότητα του λογισμικού (software quality). Τέλος στο κεφάλαιο 2.3 παρουσιάζουμε τα αποτελέσματα παλαιότερων ερευνών σχετικά με την επίδραση των προτύπων σχεδίασης στον αριθμό σφαλμάτων λογισμικού.

2.1 Σφάλματα Λογισμικού

Μια από τις μεγαλύτερες προκλήσεις που παρουσιάζονται σε θέματα διασφάλισης της ποιότητας λογισμικού είναι η αποσφαλμάτωση. Σύμφωνα με την ανακοίνωση του Εθνικού Ινστιτούτου Προτύπων και Τεχνολογίας το 2002, τα σφάλματα του λογισμικού προκάλεσαν ζημιές 60 δισεκατομμυρίων για την οικονομία της ΗΠΑ [Tassey, 2002]. Η διασφάλιση της ποιότητας είναι περιορισμένη από το χρόνο και το κόστος, επομένως, οι διαχειριστές προσπαθούν να διαθέσουν τους πόρους για τα μέρη του κώδικα του προγράμματος που είναι πιο πιθανό να περιέχουν σφάλματα σε σχέση με τα άλλα μέρη του προγράμματος. Το ερώτημα είναι, ποιος είναι ο καλύτερος τρόπος για τον εντοπισμό των σφαλμάτων στα τμήματα του κώδικα για την κατανομή των πόρων. Τα σφάλματα του λογισμικού μπορεί να καθοριστούν πριν ή μετά την έκδοση του λογισμικού. Προφανώς ο χρήστης λογισμικού ασχολείται με τα σφάλματα μετά την ολοκλήρωση του λογισμικού. Πολλές προσπάθειες έχουν πραγματοποιήσει οι διαχειριστές του λογισμικού, προκειμένου να δοκιμάσουν τα λογισμικά και πάντα προσπαθούσαν να εντοπίσουν τα τμήματα κώδικα που περιλαμβάνουν τα σφάλματα, παρουσιάζοντας διάφορες μετρικές με πρόβλεψη σφαλμάτων. Αυτές οι μετρικές ήταν αποτελεσματικές. Μια ομάδα από αυτές τις μετρικές καθορίζουν τα σφάλματα του λογισμικού με τη μέτρηση των κωδικό - χαρακτήρων όπως η πολυπλοκότητα της μετρικής McCabe [Basili et al., 1996, Nagappan et al., 2006, Subramanyam et al., 2003]. Οι μετρικές αυτές είναι αποδεκτές από πολλούς

ερευνητές και μπορούν εύκολα να εξαχθούν από τον κώδικα. Μια άλλη ομάδα μετρικών βοηθούν τις ομάδες λογισμικού στην πρόβλεψη του σφάλματος, χρησιμοποιώντας την εξάρτηση μεταξύ κομματιών κώδικα, όπως ενότητες, κλάσεις και αρχεία.

Ο Zimmerman το 2008 έκανε μία έρευνα σχετικά με μετρικές δικτύου και έδειξαν ότι οι μετρικές βασίζονται στην εξάρτηση διπλής ανάκλησης όσο και στην ανάκληση της πολυπλοκότητας των μετρικών [Nagappan and Zimmermann, 2008].

Ο εντοπισμός των σφαλμάτων από την εξόρυξη των δεδομένων μπορούν να κατηγοριοποιηθούν σε τρεις ομάδες: πολυπλοκότητα μετρικών, ιστορικά δεδομένα και εξαρτήσεις.

Πολυπλοκότητα μετρικών (Complexity metrics) : Στο [Binkley and Schach, 1998] ο Binkley και ο Scotch παρουσίασαν την σύζευξη μεταξύ των μετρικών. Οι έρευνες τους έδειξαν ότι αυτές οι μετρικές είναι επιτυχής στη βελτίωση της ποιότητας του σχεδιασμού του λογισμικού. Απέδειξαν ότι η σύζευξη μεταξύ των μετρικών είναι πολύ πιο αποτελεσματική στην εκτέλεση πρόβλεψης του χρόνου αποτυχίας σε σχέση με την πολυπλοκότητα των μετρικών. Ο Alberg και ο Olson χρησιμοποίησαν την πολυπλοκότητα των μετρικών για να προβλέψουν αυτές τις ενότητες που προκαλούν αποτυχία χρόνου. Στο άρθρο [Alberg and Ohlsson, 1996] οι συγγραφείς ανακάλυψαν ότι στα μοντέλα πρόβλεψης τους μπορούσαν να εντοπίσουν το 20% των ενοτήτων κώδικα που περιείχαν 47% του συνόλου των σφαλμάτων. Ο Nagappan ανακάλυψε κάποιες μετρικές για την πρόβλεψη των σφαλμάτων ερευνώντας σε εργασίες στο Microsoft 5 καθώς και παρουσιάζοντας μια αναφορά σχετικά με το πώς η κυκλοφορία των σφαλμάτων μπορούν να δημιουργηθούν με βάση την ιστορία[Ball et al., 2006].Στο [Ball and Nagappan, 2005] ο συγγραφέας υποστηρίζει ότι ο κώδικας με υψηλό αριθμό εξαρτίσεων μεταξύ των κλάσεων μπορεί να προκαλέσει σφάλματα που γίνονται αντιληπτά μετά την ολοκλήρωση του λογισμικού. Η πρόβλεψη αυτή έγινε με τη βοήθεια της πολυπλοκότητας των μετρικών όπως τα εξαρτήματα του κώδικα και η αλλαγή της χρονικής έκτασης του κώδικα.

Ιστορικά δεδομένα (Historical data) : Ο Hodpel μπόρεσε και πρόβλεψε εάν μια ενότητα περιέχει σφάλματα ή όχι χρησιμοποιώντας το συνδυασμό των στοιχείων της πολυπλοκότητας των μετρικών και τα ιστορικά δεδομένα. Σε αυτή την έρευνα, ο

σχεδιασμός του λογισμικού των μετρικών χρησιμοποιήθηκε στον εντοπισμό των σφαλμάτων εκτός από τη χρήση επαναχρησιμοποιημένης πληροφορίας. Στο [Allen et al., 1996] θεωρήθηκε ότι οι νέες ή τροποποιημένες ενότητες είχαν μεγαλύτερη πυκνότητα σφαλμάτων. Ο Ostrand μελέτησε 2 τεράστια συστήματα λογισμικού με βάση τα ιστορικά δεδομένα χρησιμοποιώντας μέχρι και 17 δελτία για την πρόβλεψη των αρχείων με την υψηλότερη πυκνότητα σφαλμάτων. Στο [Bell et al., 2005] ο συγγραφέας υποστηρίζει ότι το μοντέλο πρόβλεψης σφαλμάτων προσδιόρισε ότι το 20% του συνόλου των φακέλων από τα 2 συστήματα λογισμικού, περιλαμβάνουν αρχεία τα οποία περιέχουν σφάλματα και θα μπορούσε να τα εντοπίσει κατά 71% και 92% του συνόλου των σφαλμάτων. Στο άρθρο [Allen et al.,1996] ο Khoshgoftar παρουσίασε τα μοντέλα πρόβλεψης, στα οποία εισάγονται τα μοντέλα με διάφορους αριθμούς γραμμών κώδικα σε διαφορετική θέση ως ελαττωματικές τμήματα. Στο [Graves et al., 2000] ο συγγραφέας έδειξε ότι οι μετρικές βασίζονται με βάση τα ιστορικά δεδομένα καλύτερα παρότι η πολυπλοκότητα των μετρικών χρησιμοποιώντας τες σε ένα μεγάλο σύστημα με μεγάλη χρονική διάρκεια. Όπως είναι φανερό από το αποτέλεσμα των ερευνών με βάση τα ιστορικά δεδομένα, οι μετρικές που βασίζονται στα ιστορικά δεδομένα έχουν καλύτερες επιδώσεις από τις μετρικές που βασίζονται στην πολυπλοκότητα τους. Υπάρχουν όμως και κάποια προβλήματα χρησιμοποιώντας τις μετρικές που βασίζονται στα ιστορικά δεδομένα. Η μεγάλη διάρκεια του προγράμματος με σφάλματα μπορεί να μην υπάρχει.

Εξαρτήσεις (Dependency): Ο Henry και ο Kafura το 1998 στο [Kafura et al., 1998] παρουσίασαν μία νέα μετρική, δίνοντας όνομα μετρική πολυπλοκότητας χρησιμοποιώντας τις μετρικές fan-in (number of the nodes that call a given node) and fan-out (number of the nodes that are called by a given node) και επέδειξαν ότι το πρόγραμμα σχεδιάστηκε από τον υψηλό ρυθμό των μετρικών fan- in και fan-out έχοντας ακατάλληλο σχεδιασμό.

Το 1990 ο Clarke και ο Pogdulsky παρουσίασαν ένα τυπικό μοντέλο. Η σύνδεση μεταξύ των δυο τμημάτων από τον κώδικα μπορεί να ανακαλυφθεί από το πηγαίο κώδικα βασισμένο στο συγκεκριμένο μοντέλο. Επίσης, προσπάθησαν να

χρησιμοποιήσουν αυτές τις εξαρτήσεις για το πειραματικό λογισμικό[Clarke et al., 1990]. Στο [Bevan and Whitehead, 2003] οι συγγραφείς συνδύασαν τις εξαρτήσεις με τα ιστορικά δεδομένα, το 2003 και το 2006 προκειμένου να προβλέψουν την αστάθεια στο λογισμικό. Έκριναν, επίσης, το είδος της εξάρτησης εκτός από την καταμέτρηση των μετρικών fan-in και fan-outs[Schroter et al., 2006]. Στο [Ball and Nagappan, 2007] ο συγγραφέας αναφέρει πως μία ομάδα της Microsoft το 2007 ερεύνησε και έδειξε ότι χρησιμοποιώντας τον κώδικα με υψηλό αριθμό εξαρτήσεων μεταξύ των κλάσεων μπορεί να προβλέψει τα σφάλματα μετά από την ολοκλήρωση του λογισμικού.

Στο [Nagappan and Zimmermann, 2008], οι συγγραφείς πραγματοποίησαν ένα πείραμα για την ανίχνευση σφαλμάτων με την χρήση των γραφημάτων εξάρτησης σε δύο επίπεδα EGO και Global, και έδειξαν ότι η ανάκληση στο πείραμά τους είναι διπλάσια από την ανάκλησή που πραγματοποιείται χρησιμοποιώντας την πολυπλοκότητα των μετρικών. Επίσης, στο [Bener et al., 2009] ο Turhan απέδειξε ότι η πρόβλεψη των σφαλμάτων χρησιμοποιώντας γραφήματα έχει πολύ μεγαλύτερη επιτυχία σε σύγκριση με την πρόβλεψη των σφαλμάτων χρησιμοποιώντας την πολυπλοκότητα των μετρικών σε μεγάλα προγράμματα, αλλά εάν το χρησιμοποιήσουμε σε ένα σύντομο πρόγραμμα δεν θα υπάρχει πολύ μεγάλη διαφορά.

2.2 Πρότυπα και Ποιότητα Λογισμικού

Η ενότητα αυτή περιλαμβάνει 30 άρθρα τα οποία ασχολούνται με τις συνέπειες της εφαρμογής προτύπων σχεδίασης στην ποιότητα του λογισμικού. Η ποιότητα λογισμικού αξιολογείται με βάση κάποια χαρακτηριστικά, όπως η σταθερότητα, η ευχρηστία, η εύκολη κατανόηση, η χρηστικότητα κ.α.

Τα (Di Penta et al., 2008, Aversano et al., 2007, Vokac, 2004 και Gatrell et Al., 2009) προβάλλουν εμπειρικές μελέτες σχετικά με το κατά πόσο διευκολύνεται η εφαρμογή αλλαγών σε ένα λογισμικό χρησιμοποιώντας πρότυπα σχεδίασης. Το (Di Penta et al., 2008) επιδιώκει στην κατανόηση σχετικά με το εάν υπάρχουν ρόλοι που επηρεάζονται περισσότερο στις αλλαγές καθώς και εάν υπάρχουν αλλαγές που συμβαίνουν συχνότερα σε συγκεκριμένους ρόλους. Η προσέγγιση αφορά ένα

σχεδιαστικό μοτίβο αναγνώρισης προτύπων ενώ περιγράφεται και η εξαγωγή των στοιχείων που απαιτούνταν για την ολοκλήρωση της μελέτης. Τα συγκεκριμένα στοιχεία λαμβάνονται από αποθετήρια πηγαίου κώδικα τριών συστημάτων καθώς και από 12 πρότυπα σχεδίασης. Τα αποτελέσματα εξακριβώνουν ότι κάποιοι ρόλοι είναι επιρρεπείς στις αλλαγές στα σχεδιαστικά πρότυπα καθώς και προειδοποιούν για την απαραίτητη σωστή σχεδίαση των τμημάτων ενός προτύπου όταν συμβαίνουν συχνές αλλαγές. Το (Aversano et al., 2007) παρουσιάζει παρόμοια αποτελέσματα. Το άρθρο αφορά την βελτίωση των προτύπων σχεδίασης σε τρία συστήματα ανοιχτού λογισμικού, τα οποία είναι γραμμένα σε Java, αναλύει την συχνότητα που τα πρότυπα τροποποιούνται, σε ποιές αλλαγές υποβάλλονται και ποιες κλάσεις αντικαθιστούνται με τα πρότυπα. Στο (Vokac, 2004) αναλύεται η εβδομαδιαία εξέλιξη και συντήρηση ενός μεγάλου εμπορικού προϊόντος σε διάρκεια τριών ετών, συγκρίνοντας τα επίπεδα των λαθών για τις κλάσεις που συμμετείχαν σε επιλεγμένα προϊόντα σχεδίασης με τον κώδικα γενικά. Τα αποτελέσματα αποδεικνύουν ότι τα ποσοστά των προτύπων έχουν σημαντικές διαφορές. Επίσης δημιουργείται ένα σύνολο νέων εργαλείων το οποίο εξάγει πληροφορίες για τα πρότυπα σχεδίασης.

Η επόμενη υποκατηγορία άρθρων που μελετήθηκαν ασχολείται με την χρήση προτύπων ως προς την ευκολία συντήρησης λογισμικού. Το (Vokac et al., 2004) επεκτείνει ένα άλλο πείραμα σχετικά με την συντήρηση προτύπων σχεδίασης. Τα αποτελέσματα της έρευνας αποδεικνύουν ότι κάθε πρότυπο από αυτά που χρησιμοποιήθηκαν πρέπει να χρησιμοποιούνται ανάλογα με το πρόβλημα που υπάρχει και την εμπειρία του προγραμματιστή αφού τα πρότυπα έχουν τα δικά τους χαρακτηριστικά. Η τεκμηρίωση των προτύπων σχεδίασης που χρησιμοποιούνται σε ένα πρόγραμμα μπορεί να βελτιώσει τόσο την ποιότητα όσο και την ταχύτητα των διαδικασιών συντήρησης του. Και τα (Prechelt et al., 2001, Ng et al., 2006) παρουσιάζουν πειράματα. Το (Prechelt et al., 2001) ερευνά σενάρια συντήρησης λογισμικού που χρησιμοποιούν διάφορα πρότυπα και συγκρίνει σχεδιαστικές λύσεις με πρότυπα και απλούστερες εναλλακτικές λύσεις. Ως συμπέρασμα από αυτή την έρευνα προκύπτει ότι η χρήση προτύπων σχεδίασης είναι χρησιμότερη παρότι οι εναλλακτικές λύσεις. Το (Ng et al., 2006) εργάζεται με την συντήρηση του JHotDraw, ενός συστήματος ανοικτού λογισμικού που έχει επεκταθεί με πολλαπλά πρότυπα. Τα

αποτελέσματα δείχνουν ότι για να πραγματοποιηθεί ένα καθήκον συντήρησης, ο χρόνος που ξοδεύτηκε ακόμη και από συντηρητές που ήταν άπειροι, ήταν πολύ πιο σύντομος στην επανασχεδιασμένη εκδοχή από αυτόν των πεπειραμένων συντηρητών στην αρχική έκδοση. Από την άλλη πλευρά, το (Wendorff, 2001) παρουσιάζει ένα μεγάλο εμπορικό πρόγραμμα όπου η συνεχής χρήση των προτύπων έχει συμβάλει σε σοβαρά προβλήματα συντήρησης. Στο άρθρο παρουσιάζοντα προβλήματα που μπορούν να παρουσιαστούν στον πηγαίο κώδικα από την λάθος χρήση των προτύπων και έπειτα παρουσιάζονται μια σειρά απλών βημάτων για την αξιολόγηση των προτύπων από μια προοπτική επανασχεδίασης. Τα αποτελέσματα αυτής της έρευνας μας δείχνουν ότι αν και η απομάκρυνση αυτών των προτύπων εμφανίζεται να είναι επιθυμητή, συχνά συνδέονται στενά με άλλα αντικείμενα του λογισμικού, με αποτέλεσμα η αφαίρεσή τους να μην είναι δυνατή.

Στα άρθρα (Ampatzoglou και Chatzigeorgiou, 2007, Kouskouras et al., 2008 και Ng et al., 2007) παρουσιάζεται η συμβολή των προτύπων σχεδίασης στη δυνατότητα επέκτασης ενός λογισμικού. Το (Ampatzoglou και Chatzigeorgiou, 2007) ερευνά τον τρόπο με τον οποίο τα πρότυπα σχεδίασης μπορούν να χρησιμοποιηθούν στα παιχνίδια για υπολογιστή και αξιολογεί τα οφέλη και τα μειονεκτήματα τους. Για την συγκεκριμένη έρευνα, αξιολογήθηκαν δύο παιχνίδια ανοιχτού λογισμικού. Για την ποσοτική αξιολόγηση, τα προγράμματα αναλύονται με τεχνικές αντίστροφης μηχανής και υπολογίζονται κάποιες μετρικές λογισμικού. Τα αποτελέσματα δείχνουν ότι τα πρότυπα μπορούν να είναι ευεργετικά όσον αφορά την ευκολία συντήρησης, αφού φαίνεται να μειώνουν τη σύζευξη και την πολυπλοκότητα και να αυξάνουν τη συνοχή του λογισμικού. Ως συμπέρασμα προκύπτει ότι πρέπει να ενθαρρύνεται η κατάλληλη εφαρμογή προτύπων σχεδίασης. Στο (Kouskouras et al., 2008) παρουσιάζεται η συμπεριφορά μιας αντικειμενοστραφούς εφαρμογής λογισμικού σε ένα συγκεκριμένο σενάριο επέκτασης. Αναλύοντας το πλαίσιο αξιολόγησης προβλήθηκαν τρεις εναλλακτικές λύσεις. Η πρώτη λύση σχολιάζει τα αποτελέσματα των μετρικών και άλλες ποιοτικές παρατηρήσεις για μια απλή λύση του συστήματος. Η δεύτερη λύση παρουσιάζει και εφαρμόζει το πρότυπο Registry κα η τρίτη λύση εφαρμόζει θεματοστρεφή προγραμματισμό. Ως αποτέλεσμα επιδιώκεται η αξιολόγηση των τριών εναλλακτικών λύσεων, σε ποιοτικό και σε ποσοτικό επίπεδο, προσδιορίζοντας τα

πρόσθετες σχεδιαστικές επιπτώσεις που απαιτούνται τόσο για την επέκταση όσο και για την αξιολόγηση της επίδρασης της επέκτασης σε διάφορες ποιοτικά χαρακτηριστικά της εφαρμογής. Το (Ng et al., 2007) ερευνά εάν υπάρχει οποιαδήποτε σχέση μεταξύ των προτύπων και της Αρχής της Ανοιχτής – Κλειστής σχεδίασης. Πραγματοποιήθηκε ένα πείραμα, προτείνοντας ότι η ικανοποίηση των προτύπων γενικά οδηγεί στην προσαρμογή στην Αρχή της Ανοιχτής – Κλειστής σχεδίασης. Ολοκληρώνοντας το πείραμα, βρέθηκαν τρεις περιπτώσεις εξαιρέσεων.

Το (Baudry et al., 2001) παρουσιάζει δύο διαμορφώσεις ενός αντικειμενοστραφούς σχεδίου που μπορεί να αποδυναμώσει τη δυνατότητα δοκιμών. Συζητά τα προβλήματα που μπορούν να προκύψουν κατά τη διάρκεια ελέγχου είτε από κακό σχεδιασμό είτε από εφαρμογή μιας αντικειμενοστραφούς αρχιτεκτονικής. Το άρθρο εστιάζει στα πρότυπα στα πρότυπα σχεδίασης ως συνεπή υποσύνολα για την αρχιτεκτονική, εξηγεί πως η χρήση τους μπορεί να παρέχει έναν τρόπο για την μείωση της πολυπλοκότητας των ελέγχων που αφορούν τα συγκρούσεις στο πρότυπο. Η δυνατότητα δοκιμών στον αντικειμενοστραφή σχεδιασμό απασχολεί και το άρθρο (Baudry et al., 2003). Το άρθρο μελετάει αφηρημένες αναπαραστάσεις προτύπων σχεδίασης μέσα σε UML metamodel, για να επιτευχθεί αυτόματη εφαρμογή των προτύπων σχεδίασης με έναν τρόπο που θα μπορεί να ελεγχθεί.

Το άρθρο (Nielson και Knutson, 2006) μελετά το ποιοτικό χαρακτηριστικό της προσαρμοστικότητας του λογισμικού, ως προς τον αντίκτυπο των προτύπων σχεδίασης κατά την μεταφορά του σχεδίου σε άλλη γλώσσα προγραμματισμού. Το άρθρο προτείνει ένα κύκλο συντήρησης προτύπων σχεδίασης ώστε να ανταποκρίνονται στις εξελικτικές γλωσσικές αλλαγές και να προσαρμόζονται στις OO++ γλώσσες. Οι προτάσεις απεικονίζονται με την επέκταση του προτύπου Iterator σε μια OO++ παραλλαγή. Το άρθρο (Elish, 2006) εκθέτει με παραδείγματα τον αντίκτυπο τεσσάρων structural προτύπων σχεδίασης (adapter, bridge, composite and faqade) στη σταθερότητα των διαγραμμάτων κλάσεων. Το (Beck et al., 1996) παρουσιάζει την ευκολία κατανόησης λογισμικού που κάνει χρήση προτύπων σχεδίασης. Η χρήση προτύπων σχεδίασης φαίνεται να ασκεί μεγάλη επίδραση στον τρόπο ανάπτυξης λογισμικού. Το (Malloy και Power, 2006) ασχολείται με το ποιοτικό χαρακτηριστικό της ευελιξίας και περιγράφει τη χρήση δύο προτύπων σχεδίασης για

την αυτοματοποίηση του ελέγχου των μη μεταβλητών κλάσεων μιας εφαρμογής σε C++. Τα αποτελέσματα αυτής της έρευνας δείχνουν ότι τα πρότυπα Visitor και Decorator που μελετούνται παρέχουν ευελιξία από άποψη της συχνότητας και του επιπέδου κατάτμησης για την επικύρωση σταθερών κλάσεων, που εκφράζονται σε OCL(Object Constraint Language). Το (Ellis et al., 2007) παρουσιάζει γιατί η δημιουργία αντικειμένων με τη χρήση των προτύπων Factory που χρησιμοποιούνται σε APIς είναι πολύ πιο χρονοβόρα από τους κατασκευαστές, ανεξάρτητα από το πλαίσιο ή το επίπεδο εμπειρίας του προγραμματιστή που χρησιμοποιεί API. Τέλος, συζητούνται οι λόγοι που συμβαίνει αυτό, τα εμπόδια που εμφανίζονται και τα πιθανά εναλλακτικά πρότυπα που μπορούν να χρησιμοποιηθούν.

Το (McNatt και Bieman, 2001) εξετάζει την έννοια της σύζευξης προτύπων («σφιχτής» ή «χαλαρής») για να ταξινομήσει τους τρόπους που τα σχέδια μπορούν να περιλάβουν συνδεμένα πρότυπα. Τα θετικά της σύζευξης προτύπων αξιολογούνται από τη σκοπιά των αποτελεσμάτων της στην ευκολία συντήρησης, την κατάτμηση και την ικανότητα επαναχρησιμοποίησης σε περιπτώσεις σύζευξης προτύπων με διάφορους τρόπους. Το (Ng et al., 2007) μελετά εμπειρικά εάν οι συντηρητές λογισμικού χρησιμοποιούν τα ήδη εφαρμοσμένα πρότυπα σχεδίασης, και αν ναι, με ποιες λειτουργίες ασχολούνται συχνότερα. Τα πειράματα δείχνουν ότι σχεδόν όλοι ασχολούνται με την προσθήκη των νέων συμμετεχόντων, λιγότεροι ασχολούνται με ενέργειες που αφορούν τους πελάτες, ενώ ακόμη και λιγότεροι ασχολούνται με ενέργειες που αφορούν τους πελάτες, ενώ ακόμη και λιγότεροι εκτελούν καθήκοντα που περιλαμβάνουν αφηρημένα την ομάδα των συμμετεχόντων. Το (Ng και Cheung, 2005) παραθέτει τους σχεδιαστικούς περιορισμούς στην εφαρμογή των προτύπων για την επίτευξη σωστής ενθυλάκωσης. Το (Prechelt et al., 2002) επικεντρώνεται στην συντήρηση και καταγράφει τις πειραματικές δοκιμές που έγιναν για την ακόλουθη ερώτηση: Βοηθά τον συντηρητή όταν τα πρότυπα σχεδίασης που χρησιμοποιούνται σε ένα κώδικα προγράμματος είναι ρητά τεκμηριωμένα έναντι ενός καλά-σχολιασμένου προγράμματος χωρίς ρητή αναφορά στα πρότυπα σχεδίασης; Περιλαμβάνει τα πρώτα ελεγχόμενα πειραματικά αποτελέσματα για τη χρήση προτύπων σχεδίασης και παρουσιάζει μια προσέγγιση, που είναι η λύση σε μια σημαντική κατηγορία σχεδιαστικών προβλημάτων σχετικά με την τεκμηρίωση,

που αφορούν πειράματα. Το (Bieman et al., 2003) έχει στόχο να χρησιμοποιηθούν οι μελέτες περίπτωσης, τόσο από την βιομηχανία όσο κα από την κοινότητα ανοιχτού λογισμικού, για να ερευνηθεί η σχέση μεταξύ των σχεδιαστικών δομών στο αντικειμενοστραφές λογισμικό και της ανάπτυξης και των αλλαγών κατά τη συντήρηση. Τα αποτελέσματα παρέχουν διορατικότητα στο πώς τα πρότυπα σχεδίαση χρησιμοποιούνται πραγματικά και μπορούν να βοηθήσουν στο να μάθουμε να αναπτύσσουμε σχέδια λογισμικού που προσαρμόζονται ευκολότερα. Το (Jain και Yang, 2003) εξετάζει ένα εξελισσόμενο αντικειμενοστραφές εμπορικό σύστημα και εξετάζει τη σχέση μεταξύ της σχεδιαστικής δομής και των αλλαγών λογισμικού. Η πιο σημαντική εξακρίβωση είναι η ύπαρξη μιας ισχυρής σχέσης μεταξύ του μεγέθους των κλάσεων και του αριθμού των αλλαγών που συμβαίνουν. Δυο αναπάντεχα αποτελέσματα που προέκυψαν είναι: α) οι κλάσεις που συμμετέχουν στα πρότυπα σχεδίασης δεν είναι λιγότερο επιρρεπείς σε αλλαγές και β) οι κλάσεις που επαναχρησιμοποιούνται, συνήθως μέσω κληρονομικότητας, τείνουν να είναι πιο επιρρεπής στις αλλαγές. Το (Khomh και Geueheneuce, 2008) ερευνά τον αντίκτυπο των προτύπων σχεδίασης στα ποιοτικά χαρακτηριστικά, στα πλαίσια της συντήρησης λογισμικού και της εξέλιξης. Μελετώντας 10 ποιοτικά χαρακτηριστικά αποδεικνύεται ότι τα πρότυπα σχεδίασης δεν βελτιώνουν πάντα την ποιότητα των συστημάτων και ότι πρέπει να χρησιμοποιούνται με σύνεση κατά τη διάρκεια της ανάπτυξης επειδή μπορούν να αποτελέσουν εμπόδιο στη συντήρηση και την εξέλιξη.

Επίσης, τα άρθρα που ακολουθούν περιλαμβάνουν μελέτες σχετικά με την ποιότητα λογισμικού. Το (Hsueh et al., 2008) προτείνει μια προσέγγιση για την ανάλυση και τον έλεγχο προτύπων σχεδίασης από άποψη ποιότητας. Επίσης, προτείνεται και ένας τρόπος μέτρησης της αποτελεσματικής βελτίωσης της ποιότητας ενός προτύπου σχεδίασης. Τέλος, παρουσιάζεται μια μελέτη περίπτωσης στην οποία αναπτύσσεται ένα εργαλείο λογισμικού για την υποστήριξη της προσέγγισης. Το (Huston, 2001) παρουσιάζει μεθόδους ανάλυσης που καταδεικνύουν τις επιδράσεις της εφαρμογής διαφόρων προτύπων μέσω ορισμένων αποτελεσμάτων μετρικών. Το (Gustafsson et al., 2002) επιδεικνύει πώς οι μετρικές λογισμικού και τα αρχιτεκτονικά πρότυπα μπορούν να χρησιμοποιηθούν για τη διαχείριση της εξέλιξης λογισμικού. Στο (Geuheneuc et al., 2004) προτείνεται μια πειραματική μελέτη για τις κλάσεις που

διαδραματίζουν τους ρόλους στα σχεδιαστικά μοντέλα. Για την μελέτη χρησιμοποιούνται μετρικές και ένας αλγόριθμος μηχανικής μάθησης για να επισημάνει τους ρόλους αυτούς. Το (Muraki και Saeki, 2001) παρουσιάζει ένα είδος μετρικών λογισμικού που μας βοηθούν να εφαρμόσουμε τα πρότυπα σχεδίασης σε διαδικασίες επανασχεδίασης. Τέλος, το (Khomh και Gueheneuc, 2009) παρουσιάζει μια περιγραφική και αναλυτική μελέτη των κλάσεων που διαδραματίζουν μέχρι δύο ρόλους σε έξι διαφορετικά πρότυπα σχεδίασης.

2.3 Πρότυπα Σχεδίασης και Σφάλματα

Ο στόχος αυτής της μελέτης περίπτωσης ήταν να διερευνηθεί η πιθανή σύνδεση μεταξύ της χρήσης ορισμένων προτύπων σχεδίασης και τα ποσοστά λαθών που υπάρχουν στον κώδικα. Για να επιτευχθεί αυτό, ήταν απαραίτητο να 1) επιλεχθούν τα πρότυπα που πρέπει να μελετηθούν, 2) να αποκτήσουν το κατάλληλο αντικείμενο μελέτης και 3)να βρεθεί ή να δημιουργηθεί ένα εργαλείο, το οποίο μπορεί να ανακτήσει τα πρότυπα σχεδίασης από τον κώδικα με ικανοποιητική ταχύτητα και ακρίβεια.

2.3.1 Το προϊόν SuperOffice CRM5

Το προϊόν SuperOffice CRM5, το οποίο είναι φτιαγμένο από την SuperOffice ASA στην Νορβηγία, επιλέχθηκε ως το αντικείμενο μελέτης. Πρόκειται για ένα αρκετά μεγάλο και ώριμο εμπορικό προϊόν. Η εταιρεία παρέχει πλήρη πρόσβαση στον πηγαίο κώδικα και στο ιστορικό του, καθώς και την βάση δεδομένων καταγραφής λαθών. Ο συγγραφέας ήταν ανώτερος στέλεχος της ομάδας ανάπτυξης που δημιούργησε την εκπαιδευόμενη έκδοση και γι' αυτό το λόγο έχει καλή γνώση του κώδικα.

Το προϊόν αυτή είναι ένα σύστημα διαχείρισης πελατειακών σχέσεων, το οποίο χρησιμοποιείται από τις εταιρείες για να παρακολουθούν τους πελάτες τους, τις καθημερινές πωλήσεις και δραστηριότητες. Το προϊόν τρέχει σε Microsoft Windows και είναι μία κλασική client/ server εφαρμογή. Η εταιρία δίνει μεγάλη έμφαση να είναι η εφαρμογή φιλική προς τον χρήστη και κατά συνέπεια να υπάρχουν πολλές οδηγίες

για την εφαρμογή. Η εφαρμογή γράφτηκε σε C++ και έχει σχεδόν ξανά γραφτεί το 2000.

Η εφαρμογή έχει πουληθεί στην τυπική του έκδοση σε 11 γλώσσες και εγκαταστάθηκε πάνω από 11.000 εγκαταστάσεις πελατών. Οι νέες εκδόσεις κυκλοφόρησαν ελεύθερα περίπου δύο φορές το χρόνο κατά τη διάρκεια της περιόδου της μελέτης (2001-2003).

Ο κώδικας διατηρήθηκε και επεκτάθηκε για τα τρία τελευταία χρόνια από μια σταθερή ομάδα προγραμματιστών, από τους οποίους περίπου οι μισοί συμμετείχαν στην συγγραφή του αρχικού κώδικα. Ένα σύστημα παρακολούθησης σφαλμάτων [TechExcel, 2004] χρησιμοποιήθηκε για τον εντοπισμό λαθών στον κώδικα, για το εσωτερικό κατά τη διάρκεια της επίσημης προκαταρκτικής δοκιμής ή στις εγκαταστάσεις των πελατών. Το σύστημα παρακολούθησης σφαλμάτων ενσωματώθηκε με το CVS (Control Version System) [Perforce, 2004]. Οι αλλαγές στον κώδικα ελέγχθηκαν στο σύστημα ελέγχου εκδόσεων σε συναλλαγές που συνήθως αντιστοιχούν σε μια λειτουργική αλλαγή, οι οποίες συνήθως είναι σωστές.

2.3.2 Προσδιορισμός των προτύπων σχεδίασης σε C++ κώδικα

Στα πρώτα στάδια της μελέτης, διατυπώθηκαν οι ακόλουθοι στόχοι με ένα εργαλείο για την αποκατάσταση προτύπου:

Πρέπει να είναι δυνατό για να περιγράψει μια διαρθρωτική υπογραφή και να το εξάγει σε C++ κώδικα βάση το σύνολο των κλάσεων που αντιστοιχούν στην υπογραφή. Δεδομένου ότι ορισμένα πρότυπα μπορούν να οδηγήσουν σε πολύπλοκες δομές, κάποια προσπάθεια προγραμματισμού θα πρέπει να αναμένεται.

Η μέθοδος πρέπει να κλιμακώνεται πολύ καλά, και να είναι σε θέση να χειριστεί μεγάλο μέγεθος κώδικα σε 100000000 LOC σειρά με χρόνους λειτουργίας της τάξης των ωρών ή τουλάχιστον μέσα σε ένα Σαββατοκύριακο. Κατά προτίμηση, η προσθήκη νέων μοντέλων δεν θα πρέπει να αναγκάζει μια επανάληψη της όλης διαδικασίας.

Τα δεδομένα εισόδου θα πρέπει να είναι με τη μορφή «μη επεξεργασμένα» αρχεία κώδικα, δηλαδή, ανεξάρτητα από τη δομή του πηγαίου έργου. Η έξοδος θα πρέπει να είναι σε μορφή που μπορεί εύκολα να μεταφερθεί σε στατιστικά πακέτα για περαιτέρω ανάλυση.

Υφιστάμενα εργαλεία που βρέθηκαν στις βιβλιογραφίες [Gueheneuc and Albin-Amiot, 2001], [Kramer and Prechelt, 1996], [Florijn et al., 1997], [Bansiya, 1998], [Antoniol et al., 1998], [Antoniol et al., 2001], [Keller et al., 1999], [Schauer and Keller, 1998], [Albin-Amiot, 2001], δεν έχουν τεκμηριωθεί να κατέχουν το συνδυασμό της ταχύτητας, την ανάκτηση και την ακρίβεια που απαιτείται για την περιπτωσιολογική μελέτη με την πιθανή εξαίρεση των εργασιών του Balanyi και Ferenc [Balanyi and Ferenc, 2003] η οποία εμφανίστηκε μόνο μετά από τη μελέτη η οποία ήταν σε εξέλιξη.

Λόγο της έλλειψης της ικανοποίησης, αποφάσισαν να κατασκευάσουν ένα δικό τους. Το εργαλείο δημιουργήθηκε χρησιμοποιώντας δευτερεύοντα στοιχεία για να χειριστούν τα διάφορα στάδια της διαδικασίας. Κατά τη διάρκεια της μελέτης περίπτωσής, το εργαλείο απέδωσε ικανοποιητικά. Περισσότερες λεπτομέρειες σχετικά με το εργαλείο μπορείτε να βρείτε στο [Vokac].

Στο εργαλείο αυτό, ο κώδικας σε C++ αναλύεται χρησιμοποιώντας ένα εμπορικό εργαλείο [Scientific Toolworks Inc., 2003] για να δημιουργήσουν μια βάση με μεταδεδομένα (λίστες από τύπους, ονόματα, μεταβλητές, μέθοδοι, κλπ) και οι σχέσεις μεταξύ τους, όπως η μέθοδος X καλεί την μέθοδο Y. Τα μεταδεδομένα, στη συνέχεια μεταφέρονται σε μία βάση SQL Server, αναπροσαρμόζονται και τα πρότυπα ανακτώνται μέσω των ερωτημάτων που αντιστοιχούν στις υπογραφές που παράγονται για κάθε πρότυπο. Με απλή ρύθμιση του δείκτη, πραγματοποιήθηκε εξαιρετική απόδοση , ενώ η χρήση της SQL κατέστησε εύκολο την αποσφαλμάτωση στην διαδικασία ανάκτησης, ενώ συνεργάζεται με τα πλήρη στοιχεία.

Επιπλέον, αν θέλουμε να εξετάσουμε την εξέλιξη του συστήματος με την πάροδο του χρόνου, δεδομένα που εξάγονται από το CVS μπορούν να προστεθούν στα μεταδεδομένα. Με το συνδυασμό CVS δεδομένων και με τα μεταδεδομένα του προγράμματος, η εξέλιξη της συμπεριφοράς με την πάροδο του

χρόνου μπορεί να εντοπιστεί η παρουσία ή το πρότυπο μπορεί να συσχετιστεί με τις δραστηριότητες, όπως η διορθωτική συντήρηση.

2.3.3 Εργαλείο Επίδοσης: Ανάκτηση, Ακρίβεια και Επεκτασιμότητα

Τα λάθη στην ανάκτηση των προτύπων από τον κώδικα, αποτελεί σημαντική απειλή για την εγκυρότητα αυτής της μελέτης. Το θέμα ως εκ τούτου αντιμετωπίζεται με αρκετή λεπτομέρεια σε αυτό το τμήμα. Υπάρχουν δυο τύποι λαθών που πρέπει να αξιολογηθούν: τα λάθη τα οποία έχουν θετική επίδραση και τα λάθη που έχουν αρνητική επίδραση. Τα λάθη με θετική επίδραση συμβαίνουν όταν ένα πρότυπο αναγνωρίζεται, ακόμη και αν οι κλάσεις που ενδιαφέρουν δεν είναι σύμφωνες με την δομή των προτύπων. Τα λάθη με αρνητική επίδραση συμβαίνουν όταν οι κλάσεις είναι σύμφωνες με το πρότυπο αλλά δεν αναγνωρίζονται.

Τα λάθη με θετική επίδραση είναι σχετικά εύκολα να ελεγχθούν από την επιθεώρηση του κώδικα που έχει προσδιοριστεί από το εργαλείο για να διαπιστωθεί εάν όντως συμμορφώνονται προς ένα πρότυπο. Τα λάθη με αρνητική επίδραση είναι όμως πολύ πιο δύσκολο να εντοπιστούν από τη στιγμή που θα πρέπει να προσδιοριστούν όλες οι εμφανίσεις τους σε όλους τους συνδυασμούς στον κώδικα και στη συνέχεια συγκρίνονται τα στοιχεία αυτά με την έξοδο του εργαλείου. Το λογισμικό οποιουδήποτε σημαντικού μεγέθους (>10 KLOC), δεν είναι στην πραγματικότητα έργο. Ως υποκατάστατο για μια συνολική ανάλυση, μπορούμε να εξετάσουμε ένα τυχαίο δείγμα από τις κλάσεις και να καθορίσουμε τα πρότυπα. Αυτό θα μας δώσει μία ένδειξη για τον αριθμό των λαθών τα οποία έχουν αρνητική επίδραση.

Γενικά,μια ακριβή και λεπτομερή περιγραφή των υπογραφών του προτύπου θα μειώσει τον αριθμό των λαθών με θετική επίδραση, αλλά θα αυξήσει τον κίνδυνο για τα αρνητικά λάθη. Η ακριβής υπογραφή είναι συνδεδεμένη με τη γλώσσα - τα ειδικά χαρακτηριστικά και επίσης εξαρτάται από το πόσο αυστηρά ερμηνεύουμε τη δομή του εν λόγω προτύπου, δηλαδή τις συζητήσεις σχετικά με το βάθος κληρονομικότητας και άλλων παραμέτρων στα [Florijn et al., 1997],[Antoniol et al., 2001],[Balanyi and Ferenc, 2003].

Προκαλείται μία πρόκληση από την παρουσία των μακροεντολών του προεπεξεργαστή στη C και στη C++ γλώσσα. Είναι δυνατό ο κώδικας να έχει περίπλοκες δηλώσεις και οι δομές των μακροεντολών, τα οποία στη συνέχεια είναι δύσκολα να αναλυθούν. Ενώ το πρόβλημα δεν είναι εντελώς δυσεπίλυτο, όπως φαίνεται από τον Badros και τον Notkin [Badros and Notkin, 2000], το εργαλείο ανάλυσης που χρησιμοποιούν έχει μόνο περιορισμένη υποστήριξη στον προεπεξεργαστή. Αυτή η πτυχή πρέπει να ληφθεί υπόψη κατά την επικύρωση της χρήσης του εργαλείου σε οποιοδήποτε δεδομένο σύνολο του λογισμικού.

Δεδομένου ότι ένα πρότυπο σχεδίασης είναι μια άτυπη προδιαγραφή μιας συνιστώμενης δομής, αυτό θα μεταφραστεί στον κώδικα του προγράμματος με διαφορετικό τρόπο σε διαφορετικά έργα. Οποιαδήποτε συζήτηση για τα ποσοστά λαθών πρέπει, ως εκ τούτου, να εξεταστούν σε σχέση με την εφαρμογή του εργαλείου σε ένα συγκεκριμένο σύνολο στοιχείων λογισμικού.

Είναι απαραίτητο να καθοριστεί ακριβώς τι εννοούμε με ένα παράδειγμα από ένα ορισμένο πρότυπο σχεδίασης. Ένα απλό παράδειγμα είναι το πρότυπο Factory: Μία κλάση του Factory μπορεί να έχει μεθόδους για να δημιουργήσουν μια ή περισσότερες παραγόμενες κλάσεις. Θα πρέπει να μετράμε κάθε συνδυασμό μεταξύ του Factory και την παραγόμενη κλάση ως παράδειγμα ή μπορεί μια κλάση Factory να μετράει ως ένα απλό παράδειγμα, ανεξάρτητα από τον αριθμό των προϊόντων; Παρόμοιες καταστάσεις συμβαίνουν στα περισσότερα πρότυπα, δεδομένου ότι προσδιορίζουν τις σχέσεις μεταξύ πολλών κλάσεων.

Στις αξιολογήσεις μας, έχουμε υιοθετήσει το απλό ορισμό, σύμφωνα με τον οποίο υπολογίζει την μια κλάση Factory ως μια μεμονωμένη εμφάνιση. Ομοίως, μετράμε για παράδειγμα το πρότυπο Observer για κάθε θέμα κλάσεων, μια για την Template μέθοδο, για κάθε template μέθοδο, μια Decorator μέθοδο και μια Singleton για κάθε Singleton κλάση.

Ήταν απαραίτητο να ρυθμιστεί το εργαλείο αποκατάστασης πρότυπων για δύο συγκεκριμένα πρότυπα: Observer και Decorator. Το πρότυπο Observer μπορεί να εφαρμοστεί σε δύο ριζικά διαφορετικούς τρόπους. Στην κλασική δομή, ορίζει ότι κάθε θέμα θα πρέπει να παρακολουθείται απ 'ευθείας από τους Observers,

χρησιμοποιώντας μια συλλογή από αναφορές σε αντικείμενα. Αυτό εισάγει μια αρκετά ισχυρή αμφίδρομη ζεύξη. Μια εναλλακτική προσέγγιση είναι να χρησιμοποιήσουμε κάποιο είδος διαμεσολαβητή μηνυμάτων για να χειριστούν οι σχέσεις μεταξύ των θεμάτων και των παρατηρητών, σπάζοντας έτσι την άμεση αμφίδρομη σύνδεση μεταξύ του θέματος και των Observer. Αυτή είναι η προσέγγιση που ακολουθείται σε παγκόσμιο επίπεδο στον κώδικα CRM5,και το εργαλείο προσαρμόστηκε σε αυτή τη δομή Subject/ Broker / Observer. Για τον Decorator, ένα σχετικό πρόβλημα υπάρχει, αυτό της συγκέντρωσης. Το εργαλείο ήταν προσαρμοσμένο στο είδος της συγκέντρωσης, γενικά χρησιμοποιείται στον CRM5 κώδικα.

2.4.1 Θετική επίδραση

Το ποσοστό των λαθών που έχουν θετική επίδραση καθορίστηκε από την εξέταση όλων των διαπιστωμένων περιπτώσεων σε όλους τα πρότυπα. Τα αποτελέσματα για όλα τα πρότυπα δίνονται στον πίνακα 1.

Πίνακας 1: Θετική Επίδραση

Pattern	Instances	N_{false}	Error rate
Factory	53	8	15.1%
Singleton	45	0	0.0%
Observer	20	3	15.0%
Template Method	163	2	1.2%
Decorator	9	0	0.0%

Τα περισσότερα από αυτά τα λάθη προσδιορίζονται για το πρότυπο Factory στο οποίο οι κλάσεις χρησιμοποιούνται ως εσωτερικές κλάσεις. Από το εξωτερικό, αυτό μοιάζει με ένα παράδειγμα από το Factory δεδομένου ότι η εσωτερική κλάση έχει δημιουργηθεί από την εξωτερική κλάση και από πουθενά αλλού.

Ωστόσο, αυτή η χρήση δεν ανταποκρίνεται στην πρόθεση του προτύπου Factory, οπότε έχει χαρακτηριστεί ως ένα λάθος με θετική επίδραση. Είναι αρκετά εφικτό να προσαρτηθεί ο όρος "οι παραγόμενες κλάσεις δεν θα πρέπει να

είναι ένθετες μέσα στην κλάση Factory" στις υπογραφές για το πρότυπο Factory σε μια εκλεπτυσμένη εκδοχή των κανόνων.

Στις περιπτώσεις του Observer, έχουμε να κάνουμε με μια «χαλαρή συνδεδεμένη" έκδοση του Observer κατά την οποία όλες οι κοινοποιήσεις γίνεται από μια κεντρική κατηγορία διαμεσολάβησης μηνυμάτων. Αυτό διαφέρει κάπως από το κλασικό, το απλό πρότυπο Observer, στο οποίο, κάθε θέμα παρακολουθεί τους παρατηρητές του ξεχωριστά. Υπάρχουν και άλλες μορφές αλληλεπίδρασης μέσω του διαμεσολαβητή μηνυμάτων, εκείνες που είναι σύμφωνες με το πρότυπο Observer, και τα τρία θετικά λάθη στις περιπτώσεις αυτές. Δεδομένου ότι η δομή του προτύπου είναι ήδη αρκετά περίπλοκη, η περαιτέρω βελτίωση είναι δύσκολη χωρίς να ρισκάρετε ένα μεγαλύτερο αριθμό λαθών με αρνητική επίδραση.

Τη δομή για τη Template μέθοδο επιτρέπει πολλαπλά επίπεδα κληρονομίας και δεν απαιτεί περισσότερες από μία κλήση προς ένα υποκείμενο, εικονική πρωτόγονη μέθοδος για να εξετάσει τον καλούντα μια μέθοδο γονέα. Είναι ένα θέμα προτίμησης αν θα θέλαμε να ενισχύσουμε τον ορισμό, δηλαδή, να απαιτήσουμε ότι θα υπάρχουν περισσότερες από μία εφαρμογές της πρωτόγονης μεθόδου ή περισσότερες από μια πρωτόγονη μέθοδο για κάθε Template μέθοδο. Ο αριθμός των λαθών με θετική επίδραση πιθανότατα θα μειωθεί, αλλά ο αριθμός των λαθών με αρνητική επίδραση μπορεί να αυξηθεί.

Το πρότυπο Decorator έχει μια κάπως προβληματική δομή υπό την έννοια ότι περιέχει μια συνάθροιση (το σύνολο των Decorators για μια decorated κλάση) μπορεί να εφαρμοστεί με πολλούς τρόπους. Η υπογραφή ήταν προσαρμοσμένη στο γνωστό είδος της εφαρμογής συνάθροιση σε αυτόν τον κώδικα και, έτσι, δεν θα πρέπει να λαμβάνουμε το μηδενικό ποσοστό λαθών ως εγγύηση σε μια διαφορετική ρύθμιση. Επίσης, τα λάθη με αρνητική επίδραση είναι πιο πιθανά για αυτό το πρότυπο.

Το πρότυπο Singleton είναι ένα πρότυπο που είναι αρκετά εύκολο να αναγνωριστεί και έτσι το χαμηλό ποσοστό θετικών λαθών είναι όπως αναμενόταν.

2.4.2 Αρνητική επίδραση

Για να καθοριστεί το πραγματικό ποσοστό των λαθών με αρνητική επίδραση, θα πρέπει να αξιολογηθούν όλες οι κατηγορίες και να βρεθούν όλες οι περιπτώσεις κατά τις οποίες μια κλάση συμμετέχει σε ένα πρότυπο, αλλά δεν έχει εντοπιστεί από το εργαλείο. Με περισσότερες από 2000 κλάσεις, αυτό δεν είναι ρεαλιστικά δυνατό.

Αν 'αυτού, επέλεξε να αξιολογήσει ένα τυχαίο δείγμα των κλάσεων για να πάρουν μια εκτίμηση το ποσοστό των λαθών με αρνητική επίδραση. Από την προηγούμενη γνώση του κώδικα, το χαμηλό ποσοστό των λαθών αναμενόταν. Για τον υπολογισμό του απαραίτητου μεγέθους του δείγματος, τα ακόλουθα κριτήρια είναι τα εξής:

- απαιτούμενη ισχύ: 90%
- υπέθεσε ποσοστό (ποσοστό αρνητικών λαθών): 20%
- εναλλακτική αναλογία (ποσοστό που πρέπει να ελεγχθεί) : 10%

Το γεγονός αυτό απέφερε ένα απαιτούμενο μέγεθος δείγματος 109^2. Για την προστασία από τυχαία επιλογή κλάσεων, οι οποίες μπορεί να είναι μικρές ή με άλλο τρόπο να μην αντιπροσωπεύονται, το πραγματικό μέγεθος του δείγματος αυξήθηκε σε 125. Οι κλάσεις επιλέχθηκαν χρησιμοποιώντας μία ομοιόμορφα κατανεμημένη γεννήτρια τυχαίων αριθμών. Οι επιλεχθείσες κλάσεις καλύπτουν όλες τις μεγάλες ενότητες του προγράμματος.

Οι 125 κλάσεις έχουν ελεγχθεί από τον συγγραφέα μαζί με ανώτερους προγραμματιστές από την εταιρεία. Υπήρχαν 9 λάθη με αρνητική επίδραση. Από αυτά, το ένα ήταν μέρος του Decorator, ένα άλλο του Observer και τα υπόλοιπα ήταν στις Template μέθοδοι.

Τα όρια για το ποσοστό των λαθών με αρνητική επίδραση, όπως προκύπτει από τις παρατηρήσεις αυτές, περιλαμβάνονται στον πίνακα 2.

Πίνακας 2: Αρνητική Επίδραση

Pattern	N_{false}	95%	P
Factory	0	2.3%	0.000
Singleton	0	2.3%	0.000

Observer	1	3.7%	0.000
Template Method	7	10.3%	0.000
Decorator	1	3.7%	0.000

Κατά την ερμηνεία αυτών των αποτελεσμάτων, πρέπει να έχουμε κατά νου ότι η εφαρμογή τους έγινε στο SuperOffice με κώδικα CRM5. Λαμβάνοντας υπόψη τον αριθμό των πιθανών τρόπων για την υλοποίηση της δομής που περιγράφεται από ένα πρότυπο, η ισχύς του εργαλείου πρέπει να ελέγχεται για κάθε νέο στυλ κωδικοποίησης.

2.3.4 Εξαγωγή λαθών και πρότυπα σχεδίασης από τον CRM5 κώδικα

Η παρουσία των λαθών καθορίστηκε από την ανάλυση της CVS, όπου εκεί είναι όλος ο κώδικας. Αυτό το σύστημα εν μέρει ενσωματώθηκε στο σύστημα παρακολούθησης λαθών που χρησιμοποιήθηκε κατά την διάρκεια της μελέτης. Μια περαιτέρω ανάλυση κειμένου έγινε από τον συγγραφέα των σχολίων στο CVS να ανακτήσει τα λάθη που δεν είχαν τέτοιες άμεσες συνδέσεις. Η προσέγγιση αυτή επικυρώθηκε από την αξιολόγηση ενός δείγματος επιλέγοντας τυχαία από το σύνολο των αλλαγών κώδικα για να βεβαιωθούν ότι δεν υπάρχουν λανθασμένες ταξινομήσεις.

2.3.5 Στατικών μοντέλων και ποσοτικά αποτελέσματα

Η ενότητα αυτή παρουσιάζει την εξέλιξη και τη λογική της στατιστικής του μοντέλου και τα καθαρά ποσοτικά αποτελέσματα που προέκυψαν από αυτό. Ο στόχος της μελέτης περίπτωσης ήταν να καθοριστεί αν η παρουσία ορισμένων προτύπων σχεδίασης συσχετίζονται με τη συχνότητα των λαθών του κώδικα. Τα ανεπεξέργαστα δεδομένα αποτελούνται από την C + + κώδικα, οι μετρικές μεγέθους για κάθε κατηγορία, και οι δείκτες για το εάν ή όχι η κλάση συμμετέχει σε ένα πρότυπο. Το ποσό του κώδικα που δεν ήταν μέλος μιας κλάσης ήταν τόσο μικρή ώστε να είναι αμελητέα.

2.3.5.1 Απλό μοντέλο

Για την ανάλυση των δεδομένων, ένα δυαδικό λογιστικό μοντέλο παλινδρόμησης επιλέχθηκε [Kleinbaum, 1994]. Σε αυτό το μοντέλο, υπάρχουν δύο πιθανά αποτελέσματα μιας παρατήρησης, ένας από αυτούς ονομάζεται « γεγονός» ή «επιτυχία».

Τα κύρια αποτελέσματα από μια λογιστική παλινδρόμηση είναι τα ποσοστά πιθανοτήτων, τα οποία η ερμηνεία τους έχουν ως εξής: δίνεται μια αλλαγή μιας μονάδας στο υποκείμενο παράγοντα και κρατώντας όλους τους άλλους παράγοντες αμετάβλητους, ο λόγος των πιθανοτήτων δίνει τη μεταβολή της πιθανότητας για την ύπαρξη ενός γεγονότος, στην περίπτωσή μας, το γεγονός είναι η διόρθωση ενός λάθους. Για παράδειγμα, μία από τις αποδόσεις αναλογίας της τάξης του 0,5 για το Factory θα σήμαινε ότι ο κώδικας που συμμετέχει στο πρότυπο Factory έχει τη μισή πιθανότητα λαθών όλων των άλλων κώδικα που έχει το ίδιο μέγεθος και συμμετοχή σε άλλα πρότυπα. Αυτό το μοντέλο αποδόσεις αρκετά ενδιαφέροντα αποτελέσματα, όπως φαίνεται στον Πίνακα 3.

Πίνακας 3: Ποσοτικά αποτελέσματα από την τοποθέτηση του μοντέλου παλινδρόμησης στα Observed δεδομένα

Coefficient		P	Odds Ratio	95% CI Lower	Upper
Constant	(β_0)	0.000			
Factory	(β_F)	0.000	0.66	0.54	0.81
Singleton	(β_S)	0.000	2.69	2.24	3.24
Observer	(β_O)	0.000	1.53	1.33	1.75
Template	(β_T)	0.002	0.61	0.44	0.83
Method Decorator	(β_D)	0.159	0.49	0.18	1.32
Size KLOC	(β_K)	0.000	1.98	1.85	2.11
Week	(β_W)	0.000	0.99	0.99	0.99

Πρώτον, τέσσερα μοντέλα έχουν σαφώς σημαντικά ποσοστά πιθανοτήτων, αλλά όχι στην ίδια κατεύθυνση. Το πρότυπο Factory και το Template method συσχετίζονται με μικρότερη συχνότητα λαθών, ενώ το Observer και κυρίως το Singleton συσχετίζονται με υψηλή συχνότητα λαθών.

Δεύτερον, η επίδραση του μεγέθους είναι ιδιαίτερα σημαντική, όπως αναμενόταν. Προσθέτοντας 1000 LOC σε μια κατηγορία διπλασιάζει σχεδόν την πιθανότητα ενός λάθους, όλες τις άλλες περιπτώσεις το μέγεθος παραμένει σταθερό. Τέλος, υπάρχει μια πολύ μικρή τάση μείωσης του αριθμού των λαθών στην πάροδο του χρόνου. Ωστόσο, η τάση αυτή δεν θεωρείται αρκετά μεγάλο για να ακυρωθούν τα άλλα αποτελέσματα.

2.3.5.2 Ένα πλήρες μοντέλο όπου συμπεριλαμβάνονται οι αλληλεπιδράσεις

Το μοντέλο της προηγούμενης ενότητας λαμβάνει υπόψη μόνο τα κύρια αποτελέσματα (τα πέντε πρότυπα υπό μελέτη) και δύο πιθανοί παράγοντες σύγχυσης (μέγεθος και χρόνος). Ωστόσο, η τάση αυτή δεν θεωρείται αρκετά μεγάλη για να ακυρωθούν τα άλλα αποτελέσματα.

Πρώτον, η συμμετοχή των κλάσεων στα πρότυπα δεν είναι μια απλή 1:1 ή 1:0 σχέση. Είναι δυνατό για μια κλάση να συμμετέχει σε περισσότερα από ένα πρότυπα, μάλιστα, τα περιστατικά των προτύπων και των συνδυασμών των προτύπων στο υλικό των δεδομένων, όπως φαίνεται στον πίνακα 7, δείχνουν ότι οι συνδυασμοί πρέπει να λαμβάνονται υπόψη.

Για να ληφθεί αυτό υπόψη, το μοντέλο είναι recoded. Αντί να χρησιμοποιεί μία χωριστή μεταβλητή δείκτη για κάθε πρότυπο, η συμμετοχή του προτύπου σε κάθε κλάση είναι εκφραζόμενη σαν ένα συνδυασμένο πρότυπο-μεταβλητή. Η μεταβλητή περιέχει ένα «F» εάν η κλάση συμμετέχει στο Factory, ένα «S» εάν η κλάση συμμετέχει στο Singleton, και τα λοιπά. Η κλάση που συμμετέχει και στο Factory και στο Singleton, η μεταβλητή του θα περιέχει ένα «FS».

Η παλινδρόμηση είναι μια επανεκτέλεση με πρότυπο όπως ο Factory, δηλαδή, κάθε ξεχωριστή αξία θεωρείται ένα ξεχωριστό συντελεστή. Όπως και πριν, η αρχική τιμή διαμορφώνεται από αυτές τις κλάσεις που δεν συμμετέχουν σε οποιοδήποτε πρότυπο.

Διαπιστώνουμε ότι δύο συνδυασμοί έχουν σημαντικά ποσοστά πιθανοτήτων: Factory + Singleton και Observer + Singleton. Αυτό σημαίνει ότι οι κλάσεις που συμμετέχουν ταυτόχρονα και στα δύο Factory και Singleton είναι δύο φορές πιο επιρρεπής σε λάθη σε σχέση με κλάσεις που δεν συμμετέχουν σε οποιοδήποτε πρότυπο. Παρόμοια αποτελέσματα δίνουν και οι κλάσεις που συμμετέχουν στις Observer και Singleton.

Ωστόσο, υπάρχει ένας άλλος πιθανός παράγοντας σύγχυσης που πρέπει να ληφθεί υπόψη: Η πιθανή αλληλεπίδραση μεταξύ του μεγέθους και του προτύπου. Αυτό θα συνέβαινε αν τα συγκεκριμένα πρότυπα συσχετίζονται με μεγαλύτερο ή μικρότερο μέγεθος από ό, τι άλλα πρότυπα ή οι κλάσεις σε γενικές γραμμές, αυτή η σχέση θα συνεπαγόταν στη συγγραμικότητα μεταξύ του προτύπου και το μέγεθος των συντελεστών.

Περιμένουμε ήδη ότι ορισμένα πρότυπα (Observer) θα οδηγήσουν σε μεγαλύτερες κλάσεις και από άλλα πρότυπα (Factory).

2.3.5.3 Τελικό μοντέλο

Το σύνολο των αλληλεπιδράσεων Μέγεθος x Πρότυπο παράγει μια σειρά συντελεστών μη σημαντική, και η παρουσία τους διαταράσσει το υπόλοιπο του μοντέλου. Συνεπώς, θα εξαλειφθούν οι όροι της αλληλεπίδρασης που δεν είναι σημαντικοί. Ομοίως και για το Πρότυπο x Πρότυπο.

2.3.6 Ερμηνεία των αποτελεσμάτων

Πίνακας 4: Ποσοτικά αποτελέσματα της καταλληλότητας του μοντέλου με συνδυασμούς προτύπων σε συγκριση με τα παρατηριθεντα αποτελμάτα

Coefficient		P	Odds Ratio	95% CI Lower	Upper
Constant	(β_0)	0.000			
Week	(β_W)	0.000	0.99	0.99	0.99
Size KLOC	(β_K)	0.000	1.69	1.53	1.87

Factory	(β_F)	0.000	0.63	0.51	0.77
Singleton	(β_S)	0.141	1.35	0.91	2.02
Observer	(β_O)	0.000	1.55	1.26	1.91
Template Method	(β_T)	0.048	0.72	0.52	1.00
Decorator	(β_D)	0.154	0.49	0.18	1.31
Singleton + Observer	(β_{SO})	0.000	0.32	1.21	0.48
Singleton X Size	(β_{SK})	0.000	13.18	6.29	27.61
Observer X Size	(β_{OK})	0.009	1.21	1.05	1.40

Από τα ποσοτικά αποτελέσματα του πίνακα 9, διαπιστώνουμε ότι δεν υπάρχει σημαντική συσχέτιση για το πρότυπο Decorator και Singleton. Η συσχέτιση των υπόλοιπων συντελεστών είναι σημαντικές.

Οι πιθανότητες των δεικτών πρέπει να αντιμετωπίζονται με κάποια επιφύλαξη. Η παρουσία μιας συσχέτισης δεν μπορεί από μόνη της ναι είναι εγγύηση, για αυτό το λόγο θα πρέπει να πάμε πίσω στον κώδικα και να εκτελεστεί μια πιο ποιοτική ανάλυση. Επιπλέον, οι επιπτώσεις της αλληλεπίδρασης τροποποιεί την ερμηνεία των αποτελεσμάτων. Ωστόσο, ακόμη και σε ποσοτικό επίπεδο, τα αποτελέσματα είναι ενδιαφέροντα. Μπορούμε να δούμε ότι η συστηματική εξέλιξη του αριθμού των λαθών με το χρόνο ήταν πολύ μικρή. Η εκτιμώμενη αξία των 0.99 δείχνει μια μικρή μείωση του δείκτη των λαθών κατά την πάροδο του χρόνου. Έτσι, ενώ το ποσοστό των λαθών πέφτει σημαντικά, το μέγεθος του επίδρασης είναι αμελητέα. Αντίθετα, προσθέτοντας 1000 γραμμές κώδικα σε μια κλάση αυξάνεται η πιθανότητα λαθών κατά τα δύο τρίτα (αναλογία 1,69) εκτός από το Observer και το Singleton πρότυπο, όπου η επίπτωση αλληλεπίδρασης πρέπει να ληφθεί υπόψη.

Οι πιθανότητες λαθών σε μια κλάση που συμμετέχει σε στο πρότυπο Factory είναι λιγότερο από τα δύο τρίτα (0,63) του φόντου ποσοστού λαθών. Την ασημαντότητα της αλληλεπίδρασης του Μεγέθους x

Factory και Factory X Singleton αυξάνει την εμπιστοσύνη μας ότι έχουμε εκ νέου πράγματι κοιτάζοντας μια πραγματική ενέργεια που σχετίζεται με το πρότυπο.

Για το Decorator, το πρότυπο δεν χρησιμοποιείται πολύ και επιπλέον τα δεδομένα είναι πολύ αραιά. Το Template Method, από την άλλη πλευρά, χρησιμοποιείται σε πολλά διαφορετικά μέρη στον κώδικα, που κυμαίνονται από μικρά και απλά σε σχετικά βαθιά και σύνθετα. Η ευρεία διάδοση είναι η αιτία για τα αδύναμα αποτελέσματα όπως εδώ P = 0,048. Ωστόσο, στο πλαίσιο της χρήσης της πολυπλοκότητας και του προτύπου, η παρατήρηση ότι υπάρχει μόνο μια αδύναμη σχέση είναι το ίδιο ενδιαφέρουσα. Το Template Method είναι ένα πρότυπο το οποίο μπορεί να κρύβει ένα σύνθετο σύστημα το οποίο το χρησιμοποιεί σε απλές περιπτώσεις. Ο Balanyi και ο Ferenc παρατήρησαν ότι το Template Method είναι ένα ασήμαντο πρότυπο [Balanyi and Ferenc, 2003], αλλά βλέπουμε εδώ ότι η χρήση του μπορεί να ποικίλλει, από ασήμαντα σε αρκετά περίπλοκο. Η τάση που παρατηρήθηκε εδώ είναι ότι δεν έχουν την τάση να χαμηλώσουν το ποσοστό, έτσι ώστε να χρησιμοποιείται περισσότερο για τις πιο απλές παρά για τις σύνθετες λειτουργίες.

Στο σύστημα CRM5, το πρότυπο Singleton χρησιμοποιείται για αντικείμενα που είναι περισσότερο ή λιγότερο παγκόσμιας εμβέλειας. Ένα παράδειγμα είναι μια κρυφή μνήμη των προτιμήσεων των χρηστών, άλλο είναι ένα σύνολο από διασυνδεδεμένα αντικείμενα που παρακολουθούν την κατάσταση του GUI. Στο μοντέλο παλινδρόμησης, δεν έχουμε σημαντικό αποτέλεσμα για το Singleton. Το Singleton είναι ένα πρότυπο που έχει δύο σημαντικά αποτελέσματα σε συνδυασμό με άλλους παράγοντες: το μέγεθος και το Observer πρότυπο. Σημαντικοί παράγοντες αλληλεπίδρασης σημαίνει ότι το αποτέλεσμα είναι μια συνάρτηση της μεταβλητής η οποία αλληλεπιδρά και ο κύριος συντελεστής των αποδόσεων όταν η μεταβλητή της αλληλεπίδρασης είναι 0. Σύμφωνα με την ερμηνεία που έδωσε ο Demaris [DeMaris,1991], ο Hosmer και ο Lemeshow [Hosmer and Lemeshow, 2000], και ο Jaccard [Jaccard, 2001] υπολογίζουν τον λόγο των πιθανοτήτων για το Singleton, δεδομένου την ταυτόχρονη συμμετοχή του Observer.

Η εξήγηση για την αλληλεπίδραση εντοπίστηκε από την ποιοτική ανάλυση των εν λόγω κλάσεων στον κώδικα. Ένα σύνολο από επτά μικρές

κλάσεις αποτελεί μια κρατική μηχανή που ελέγχει την παγκόσμια κατάσταση του GUI του συνόλου της εφαρμογής. Αυτές οι κλάσεις είναι στο πρότυπο Singleton και ταυτόχρονα και στο Observer. Είχαν σχεδιαστεί προσεκτικά από νωρίς στο έργο και από τις υποκείμενες απαιτήσεις δεν έχουν αλλάξει, έχουν παραμείνει πολύ σταθερές.

Ο λόγος των πιθανοτήτων του προτύπου Observer είναι μόνο 1,55. Το Observer είναι ένα σχετικά περίπλοκο πρότυπο που χρησιμοποιείται σε καταστάσεις με σύζευξη μεταξύ πολλών, μη τετριμμένων κλάσεων. Το υψηλότερο από το μέσο όρο συχνότητας λαθών είναι όπως αναμενόταν.

3. ΕΜΠΕΙΡΙΚΗ ΜΕΛΕΤΗ ΤΗΣ ΕΠΙΔΡΑΣΗΣ ΤΩΝ ΠΡΟΤΥΠΩΝ ΣΧΕΔΙΑΣΗΣ ΣΤΑ ΣΦΑΛΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ ΠΑΙΧΝΙΔΙΩΝ

Προκειμένου να εκτελέσουμε την επιστημονική έρευνα στην τεχνολογία λογισμικού, πρέπει να κατανοήσουμε τις μεθόδους που έχουμε στη διάθεσή μας, τα όριά τους και πότε μπορούν να εφαρμοστούν. Σύμφωνα με το [Glass, 1994], συνοψίζονται τέσσερις μέθοδοι έρευνας στον τομέα της μηχανικής λογισμικού.

Οι μέθοδοι είναι οι εξής:

- 1)επιστημονική μέθοδος (scientific method)
- 2)μηχανική μέθοδος (engineering method)
- 3)εμπειρική μέθοδος (empirical method)
- 4)αναλυτική μέθοδος (analytical method)

Στην επιστημονική μέθοδο παρατηρείται το περιβάλλον και δημιουργείται ένα μοντέλο με βάση την παρατήρηση, για παράδειγμα, ένα μοντέλο προσομοίωσης. Στην μηχανική μέθοδο οι υπάρχουσες λύσεις μελετώνται και προτείνονται οι αλλαγές, οι οποίες στη συνέχεια αξιολογούνται. Στην εμπειρική μέθοδο προτείνεται ένα μοντέλο και στη συνέχεια αξιολογείται με εμπειρικές μελέτες, όπως για παράδειγμα, μελέτες περιπτώσεων ή πειράματα. Τέλος, στην αναλυτική μέθοδο προτείνεται μια επίσημη θεωρία, συλλέγονται εμπειρικές παρατηρήσεις και στη συνέχεια με βάση τις παρατηρήσεις συγκρίνεται η προς διερεύνηση θεωρία. Η μηχανική μέθοδος και η εμπειρική μέθοδος μπορούν να θεωρηθούν ως παραλλαγές της επιστημονικής μεθόδου [Basili, 1993]. Κάθε μία από αυτές τις μεθόδους κρίνεται ως καταλληλότερη και εφαρμόζεται σε κάποιο συγκεκριμένο τομέα.

Υπάρχουν τρεις μεγάλες ερευνητικές μέθοδοι που χρησιμοποιούνται στις εμπειρικές μελέτες οι οποίες είναι οι εξής:

- μελέτη περίπτωσης (case study).
- έρευνα πεδίου(survey),
- τυπικό ή ελεγχόμενο πείραμα (formal experiment).

Η ενότητα αυτή, στοχεύει στην διερεύνηση της χρήσης αντικειμενοστραφών προτύπων σχεδίασης στα σφάλματα και στην αποτελεσματικότητα της αποσφαλμάτωσης έργων ανοιχτού λογισμικού.

3.1 Μεθοδολογία

Λαμβάνοντας υπόψη τη φύση, το αντικείμενο της έρευνάς και την πληθώρα των διαθέσιμων έργων ανοικτού λογισμικού, πιστεύουμε ότι μια μελέτη περίπτωσης είναι η πιο κατάλληλη για τις ανάγκες της έρευνας μας. Η μελέτη περίπτωσης της έρευνάς ήταν σύμφωνα με τις κατευθυντήριες γραμμές που περιγράφονται στο [Kitchenham et al., 1995]. Σύμφωνα με το [Kitchenham et al., 1995], τα βήματα για τη διεξαγωγή μιας μελέτης περίπτωσης περιλαμβάνουν :

(α) Καθορισμός υποθέσεων
(β) Επιλογή υποκειμένων έρευνας
(γ) Μέθοδος επιλογής-σύγκρισης
(δ) Ελαχιστοποίηση παραγόντων εκτός των ανεξαρτήτων μεταβλητών
(ε) Σχεδιασμός της μελέτης περίπτωσης
(στ) Παρακολούθηση της μελέτης περίπτωσης και
(ζ) Ανάλυση και αναφορά των αποτελεσμάτων

Οι υποθέσεις, δηλαδή το βήμα (α), ορίζονται στο κεφάλαιο 3.2.1 τα βήματα (β) και (δ), τα οποία ασχολούνται με το πρωτόκολλο επιλογής των έργων και τους παράγοντες σύγχυσης παρουσιάζονται στο κεφάλαιο 3.2.2, η οποία συνοδεύεται με το βήμα (ε). Οι μέθοδοι που χρησιμοποιούνται για την ανάλυση των δεδομένων, δηλαδή το βήμα (γ), παρουσιάζεται στο κεφάλαιο 3.3. Το βήμα (στ), όπως αυτό περιγράφεται στο [Kitchenham et al., 1995] αναλύεται στην παράγραφο 5. Τέλος, σχετικά με το βήμα (ζ), αναφέρουμε τα αποτελέσματα στο κεφάλαιο 3.3 και τα συμπεράσματα στο κεφάλαιο 6.

3.1.1 Τα ερωτήματα της έρευνας

Στην ενότητα αυτή θέτουμε τα ερωτήματα που ερευνούμε στη μελέτη μας:

RQ1: Είναι η χρήση των προτύπων σχεδίασης σχετιζόμενη με τον αριθμό των λαθών στα παιχνίδια ανοικτού λογισμικού ;

RQ2: Είναι η χρήση των προτύπων σχεδίασης σχετιζόμενη με την επιτυχής αποσφαλμάτωση στα παιχνίδια ανοικτού λογισμικού;

3.1.2 Πλάνο της μελέτης περίπτωσης

Σύμφωνα με το άρθρο [Basili et al., 1986], προκειμένου να διενεργηθεί σωστά μια μελέτη περίπτωσης, πρέπει να ορίσουμε προσεκτικά ένα πλάνο μελέτης. Στη συγκεκριμένη μελέτη περίπτωσης το πλάνο αυτό αποτελείται από μια διαδικασία πέντε βημάτων:

1.Εύρεση ενός αριθμού έργων που πληρούν τα κριτήρια επιλογής στο χώρο των παιχνιδιών για ηλεκτρονικούς υπολογιστές

2.Ανίχνευση των προτύπων σχεδίασης που χρησιμοποιούνται σε κάθε έργο που έχει επιλεγεί

3.Ανάλυση στην ανίχνευση λαθών των έργων, προκειμένου να προσδιοριστεί ο αριθμός των ανοιχτών λαθών και των λαθών που έχουν διορθωθεί.

4.Καταχώρηση αποτελεσμάτων σε πίνακες

5.Ανάλυση των δεδομένων όσον αφορά τα ερωτήματα της έρευνας

Από τις διαθέσιμες κατηγορίες λογισμικού ανοιχτού κώδικα OSS (Open Source Software) επιλέξαμε έργα που πληρούν τα εξής κριτήρια:

1.Είναι γραμμένα σε java, σύμφωνα με τους περιορισμούς του εργαλείου που χρησιμοποιούμε για την ανίχνευση των προτύπων [Tsantalis et al., 2006]

2.Διαθέτουν δυαδικό κώδικα, σύμφωνα με τους περιορισμούς του εργαλείου που χρησιμοποιούμε για την ανίχνευση των προτύπων

3.Λογισμικό που είχε περισσότερα από 10 λάθη για κάθε μία από τις εκδόσεις του

Ένας πιθανός παράγοντας σύγχυσης σε αυτή την μελέτη είναι ότι τόσο ο βαθμός χρήσης προτύπου σχεδίασης όσο και η συχνότητα των σφαλμάτων στον κώδικα συσχετίζονται με το μέγεθος του λογισμικού. Έτσι, τα μεγαλύτερα προγράμματα αναμένεται να έχουν περισσότερα σφάλματα και περισσότερες περιπτώσεις σχεδίασης.

Έτσι, πιθανά αποτελέσματα τα οποία δείχνουν ότι κάποια πρότυπα σχεδίασης συσχετίζονται θετικά με τον αριθμό των λαθών πρέπει να μελετηθούν περεταίρω. Επιπλέον, ένας άλλος παράγοντας που πρέπει να εξεταστεί κατά την ερμηνεία των αποτελεσμάτων, είναι ο βαθμός γνώσης των προγραμματιστών στα πρότυπα σχεδίασης καθώς και σε παιχνίδια ανοιχτού λογισμικού.

Θεωρείται ότι οι πιο έμπειροι προγραμματιστές είναι πιο πιθανό να χρησιμοποιούν περισσότερο πολύπλοκα σχεδιαστικά πρότυπα και οι λιγότερο έμπειροι προγραμματιστές να χρησιμοποιούν τα πιο απλά, όπως το Strategy και το Adapter. Επιπλέον, πιστεύουμε ότι οι πιο έμπειροι προγραμματιστές είναι πιο πιθανό να γράψουν κώδικα χωρίς λάθη.

3.1.3 Μέθοδοι ανάλυσης δεδομένων

Το σύνολο των δεδομένων που προέκυψαν μετά την ανίχνευση των προτύπων αποτελούν τα αριθμητικά μας δεδομένα. Ωστόσο, ορισμένες τεχνικές που χρησιμοποιήθηκαν κατά την ανάλυση των δεδομένων απαιτούσαν κατηγορικές ή δυαδικές μεταβλητές. Έτσι, έγιναν κάποιες μετατροπές στην μορφή των δεδομένων. Με την ολοκλήρωση της φάσης της προεπεξεργασίας κάθε εφαρμογή χαρακτηρίζεται από 64 μεταβλητές:

- όνομα
- έκδοση

- ανοιχτά σφάλματα (bugs open)
- διορθωμένα σφάλματα (bugs fixed)
- αποδοτικότητα αποσφαλμάτωσης (δηλαδή διορθωμένα σφάλματα δια ανοιχτά σφάλματα)
- αριθμός κλάσεων
- τρεις μεταβλητές για κάθε πρότυπο (number of pattern instances, αριθμός κλάσεων που συμμετέχουν στο πρότυπο, και το ποσοστό της συμμετοχής των κλάσεων σε κάθε πρότυπο). Συνολικά 33 μεταβλητές.
- δύο κατηγορικές μεταβλητές για κάθε πρότυπο. Συνολικά 22 μεταβλητές.
- συνολικό ποσοστό κλάσεων που συμμετέχουν σε πρότυπα. Είναι ο αριθμός των κλάσεων που συμμετέχουν σε πρότυπα προς το συνολικό αριθμό κλάσεων του έργου.
- δύο κατηγορικές μεταβλητές για τον χαρακτηρισμό του έργου, σύμφωνα με το συνολικό αριθμό των προτύπων που συμμετέχουν στις κλάσεις

Κατά τη διάρκεια της ανάλυσης χρησιμοποιήσαμε στατιστικές μεθόδους, όπως περιγραφική στατιστική, t-test και boxplots. Η στατιστική ανάλυση έγινε με τη χρήση SPSS.

3.2 Αποτελέσματα

Στην ενότητα αυτή, παρουσιάζουμε τα ευρήματα της εμπειρικής μελέτης. Το σύνολο των δεδομένων αποτελείται από 97 παιχνίδια ανοιχτού λογισμικού διαφόρων μεγεθών, τη συχνότητα λαθών και το βαθμό χρήσης προτύπων σχεδίασης.

Οι περιγραφικές μετρήσεις που περιγράφουν το σύνολο δεδομένων παρουσιάζονται στον Πίνακα 5, στον οποίο φαίνεται ο ελάχιστος αριθμός προτύπων που εντοπίσθηκαν σε μία εφαρμογή, ο μέγιστος αριθμός προτύπων που εντοπίσθηκαν σε μία εφαρμογή, η μέση τιμή για όλα τα πρότυπα σχεδίασης και η τυπική απόκλιση για την κάθε μεταβλητή.

Πίνακας 5: Περιγραφική στατιστική των δεδομένων

Variable	min	max	mean	std. dev	Variable	min	max	Mean	std. dev
bugs opened	10	253	59,82	48,20	*stateStrategy instances*	1	242	109,98	79,206
bugs fixed	3	228	59,76	44,49	*stateStrategy participants*	2	550	163,86	114,709
bugs fixed / bugs opened	0.15	3.26	1.10	0.572	*stateStrategy percentage in %*	4,12%	34,02 %	16,70%	6,03 %
number of classes	37	1964	909,38	464,42	*template instances*	0	27	11,68	5,878
overall pattern participants	12.46%	50.51%	30.74%	6.99	*template participants*	0	155	75,82	53,869
factory instances	0	24	4,95	4,73	*template percentage in %*	0,00%	16,37 %	7,76 %	4,84 %
factory participants	0	136	19,44	28,14	*decorator instances*	0	26	2,62	4,338
factory participants in %	0,00 %	6,95 %	1,53 %	1,63 %	*decorator participants*	0	105	13,49	17,816
singleton instances	0	63	23,77	15,609	*decorator percentage in %*	0,00%	5,36%	1,28 %	1,07 %
singleton participants	0	63	23,77	15,609	*prototype instances*	0	411	14,07	62,375
singleton participants in %	0,00 %	21,62%	3,04 %	3,27 %	*prototype participants*	0	464	28,54	79,063
composite instances	0	7	1,16	1,320	*prototype percentage*	0,00%	24,33	2,02 %	4,22 %

					in %		%		
composite participants	0	66	5,14	10,831	*proxy instances*	0	23	10,66	8,807
composite percentage in %	0,00 %	3,36 %	0,45 %	0,69 %	*proxy participants*	0	37	12,53	9,931
adapter instances	1	224	98,58	52,409	*proxy percentage in %*	0,00%	9,09%	1,48 %	1,52 %
adapter participants	2	264	105,24	56,638	*Proxy2 instances*	0	2	0,31	,727
adapter percentage in %	3,77 %	24,74%	11,95%	4,76 %	*Proxy2 participants*	0	4	0,62	1,454
observer instances	0	15	7,90	5,011	*Proxy2 percentage in %*	0,00%	0,28	0,04 %	0,10 %
observer participants	0	152	52,78	53,828					
observer percentage in %	0,00 %	21,82%	5,22 %	4,76 %					

Επιπλέον, εκτελέσαμε Pearson x^2 test ώστε να μελετηθεί η συσχέτιση μεταξύ των εξαρτημένων μεταβλητών(δηλαδή τη συχνότητα των λαθών και την αποτελεσματικότητα της αποσφαλμάτωσης) με τις αριθμητικές ανεξάρτητες μεταβλητές(δηλαδή το πλήθος στιγμιότυπων προτύπων, το πλήθος ων κλάσεων που συμμετέχουν σε πρότυπα και το ποσοστό των κλάσεων του συστήματος που συμμετέχουν στα στα πρότυπα). Τα στατιστικά σημαντικά αποτελέσματα παρουσιάζονται στον Πίνακα 6.

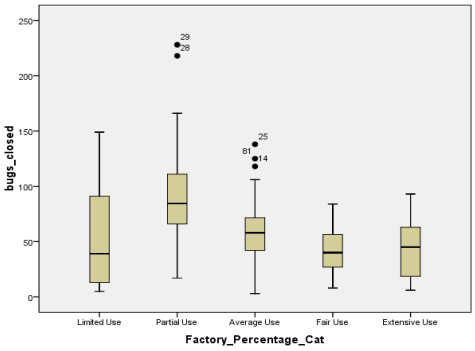
Πίνακας 6: Στατιστικά Σημαντικά Συσχτισμένες Μεταβλητές

Variable	**test**	**bugs opened**	**Debugging efficiency**	**Variable**	**test**	**bugs opened**	**Debugging efficiency**

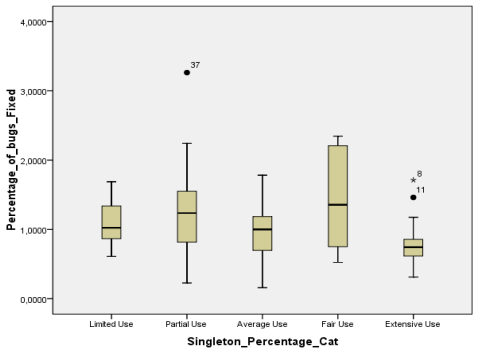
Factory instances	pearson correlation	-,367**	,011	*Observer instances*	pearson correlation	-,397**	,185
	sig. (2-tailed)	,000	,918		sig. (2-tailed)	,000	,070
Factory participants	pearson correlation	-,270**	-,037	*Observer participants*	pearson correlation	-,259*	,163
	sig. (2-tailed)	,008	,721		sig. (2-tailed)	,010	,111
Factory percentage	pearson correlation	-,293**	,039	*Observer participation pct*	pearson correlation	-,338**	,219*
	sig. (2-tailed)	,004	,705		sig. (2-tailed)	,001	,032
Singleton participation pct	pearson correlation	-,052	-,217*	*Template participants*	pearson correlation	,253*	,014
	sig. (2-tailed)	,612	,033		sig. (2-tailed)	,012	,891
Composite instances	pearson correlation	-,398**	,045	*Template participation pct*	pearson correlation	,511**	-,014
	sig. (2-tailed)	,000	,663		sig. (2-tailed)	,000	,892
Composite participants	pearson correlation	-,234*	-,106	*Prototype participants*	pearson correlation	-,205*	-,103
	sig. (2-tailed)	,021	,304		sig. (2-tailed)	,044	,315
Composite participation pct	pearson correlation	-,203*	-,059	*Prototype participation pct*	pearson correlation	-,242*	-,038
	sig. (2-tailed)	,046	,565		sig. (2-tailed)	,017	,710
Adapter participants	pearson correlation	,276**	-,109	*Proxy instances*	pearson correlation	-,448**	,193
	sig. (2-tailed)	,006	,289		sig. (2-tailed)	,000	,058
Adapter	pearson	,531**	-,224*	*Proxy*	pearson	-,427**	,140

participation pct	correlation sig. (2-tailed)	,000	,028	*participants*	correlation sig. (2-tailed)	,000	,173
State/ Strategy instances	pearson correlation	-,335**	,173	*Proxy participation pct*	pearson correlation	-,365**	,132
	sig. (2-tailed)	,001	,091		sig. (2-tailed)	,000	,198
Decorator participation pct	pearson correlation	,191	-,212*				

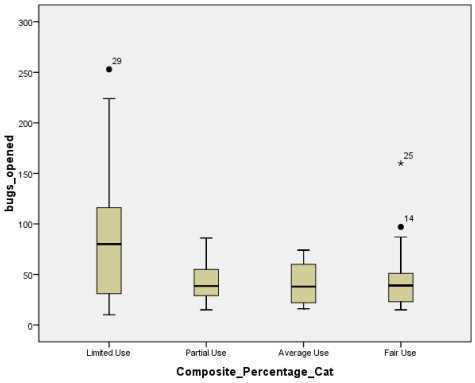
Για να απεικονίσει την επίδραση κάθε ανεξάρτητης μεταβλητής στην εξαρτημένη μεταβλητή, δημιουργήσαμε boxplots τις σημαντικότερες συσχετίσεις.



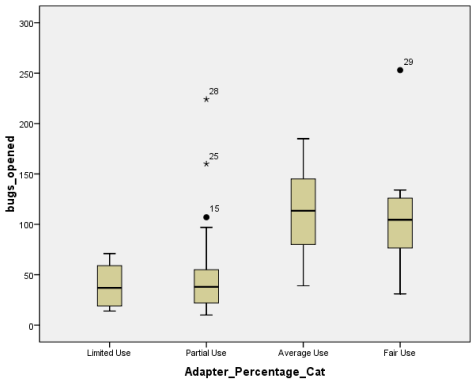
Σχήμα11: Boxplot για τη συχνότητα βλάβης και χρήση του προτύπου Factory



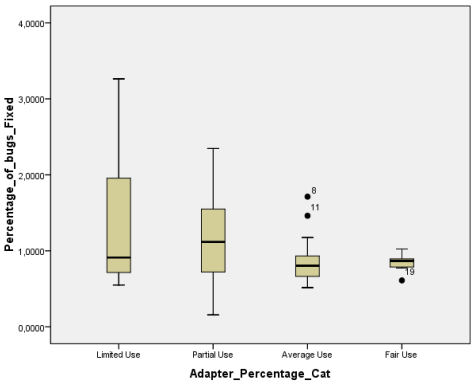
Σχήμα 12. Boxplot για τον εντοπισμό σφαλμάτων και χρήση του προτύπου Singleton



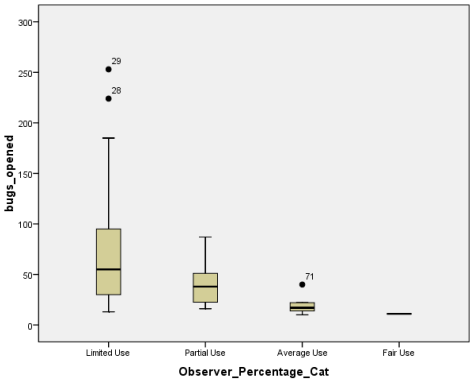
Σχήμα 13. Boxplot για τη συχνότητα βλάβης και χρήση του προτύπου Composite



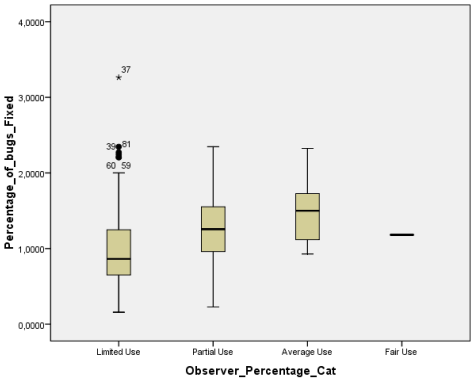
Σχήμα 14. Boxplot για τη συχνότητα βλάβης και χρήση του προτύπου Adapter



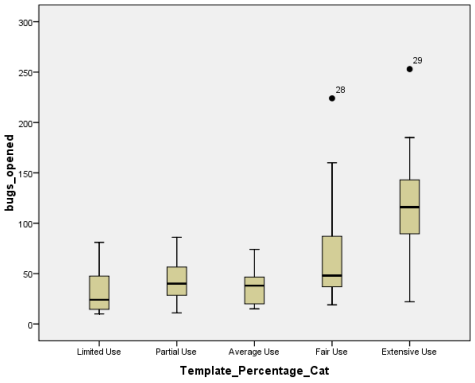
Σχήμα 15. Boxplot για τον εντοπισμό σφαλμάτων και χρήση του προτύπου Adapter



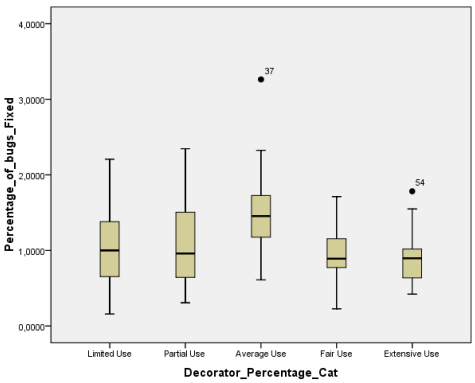
Σχήμα 16. Boxplot για τη συχνότητα βλάβης και χρήση του προτύπου Observer



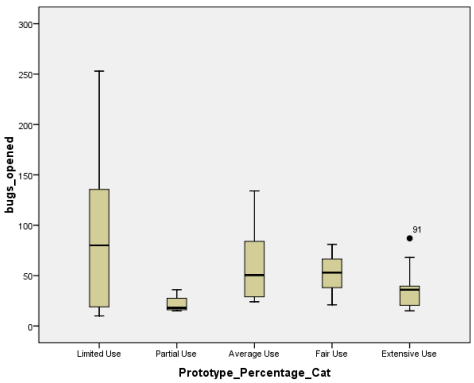
Σχήμα 17. Boxplot για τον εντοπισμό σφαλμάτων και χρήση του προτύπου Observer



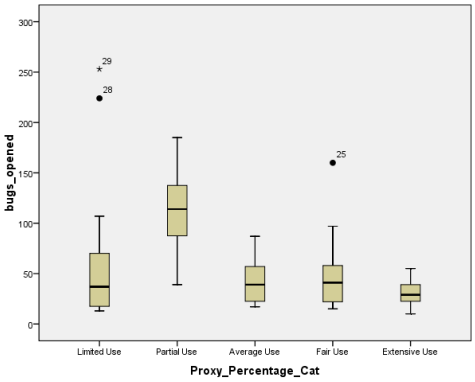
Σχήμα 18. Boxplot για τη συχνότητα βλάβης και χρήση του προτύπου Template Method



Σχήμα 19. Boxplot για τον εντοπισμό σφαλμάτων και τη χρήση του προτύπου Decorator



Σχήμα 20. Boxplot για τη συχνότητα βλάβης και τη χρήση του προτύπου Prototype



Σχήμα 11. Boxplot για τη συχνότητα βλάβης και τη χρήση προτύπου Proxy

Τέλος, προκειμένου να διερευνήσουμε εάν υπάρχει στατιστικά σημαντική διαφορά στην απόδοση αποσφαλμάτωσης μεταξύ διαφορετικών προτύπων πάνω στο βαθμό χρήσης, πραγματοποιήσαμε independend sample t-tests . Τα σημαντικότερα ευρήματα αναφέρονται στον Πίνακα 7.

Πίνακας 7. Η διαφορά στη μέση τιμή της απόδοσης αποσφαλμάτωσης μεταξύ των προτύπων και στον βαθμό χρήσης τους.

Pattern	**Application Intensity**	**diff**	**sig**
Factory	Partial Use – Average Use	-0.441	0.02
	Partial Use – Fair Use	-0.366	0.05
Singleton	Limited Use – Fair Use	-0.374	0.04
	Limited Use – Extensive Use	0.294	0.00
	Average Use – Fair Use	-0.484	0.01
	Average Use – Extensive Use	0.184	0.00
Adapter	Partial Use – Extensive Use	0.428	0.00

	Fair Use – Extensive Use	0.403	0.01
Observer	Limited Use – Extensive Use	-0.560	0.00
	Average Use – Extensive Use	-0.444	0.00
	Fair Use – Extensive Use	-0.412	0.05
Template Method	Partial Use – Fair Use	-0.405	0.05
	Fair Use – Extensive Use	0.526	0.00
Decorator	Average Use – Fair Use	0.554	0.00
	Average Use – Extensive Use	0.595	0.00
Prototype	Limited Use – Average Use	-0.365	0.01
Proxy	Partial Use – Average Use	-0.262	0.03
	Partial Use – Fair Use	-0.121	0.01

4. ΠΕΡΙΓΡΑΦΗ ΕΝΔΕΙΚΤΙΚΟΥ ΠΑΡΑΔΕΙΓΜΑΤΟΣ

Η πτυχιακή εργασία αφορά τη δημιουργία ενός online παιχνιδιού, το οποίο δίνει την δυνατότητα στον χρήστη να δοκιμάσει τις γνώσεις του στην Ανάπτυξη Λογισμικού. Στο παιχνίδι θα υπάρχουν δύο είδη χρηστών: ο καθηγητής και ο φοιτητής. Ο ρόλος του χρήστη θα καθορίζεται κατά την σύνδεση(από το όνομα χρήστη και τον κωδικό πρόσβασης) που θα πρέπει να κάνει στο παιχνίδι. Οι χρήστες που θα εισέρχονται σαν φοιτητές θα παίρνουν τον ρόλο του project manager.

4.1 Περιγραφή Παιχνιδιού

Η εταιρία θα δίνει στον φοιτητή ένα αρχικό προϋπολογισμό(budget), το οποίο θα πρέπει να το διαχειριστεί κατάλληλα. Ο χρήστης θα μπορεί να επιλέξει το έργο που θα εκπονήσει ανάμεσα σε ένα πλήθος από διαθέσιμα project έχοντας την δυνατότητα να επιλέξει όσα θέλει και να τα ολοκληρώσει σειριακά. Για παράδειγμα, εάν ένας φοιτητής πάρει 50,000 ευρώ budget, μπορεί να επιλέξει 3 project όπου το ένα θα κοστίζει 5,000 ευρώ, το άλλο 10,000 ευρώ και το τελευταίο 20,000 ευρώ. Για να μπορέσει ο φοιτητής να αξιολογήσει και να κοστολογήσει τα project που του προσφέρονται, θα έχει πρόσβαση σε κάποια παραδείγματα που θα δείχνουν μέσα σε πόσο χρονικό διάστημα ολοκληρώθηκαν, πόσοι υπάλληλοι αξιοποιήθηκαν και πόσο χρονικό διάστημα κράτησε η κάθε φάση. Στη συνέχεια αφού έχει επιλέξει το project, θα ορίσει τις ανθρωποώρες που θα διαρκέσει η κάθε φάση. Στη συνέχεια θα πρέπει να προσλάβει υπαλλήλους, οι οποίοι δουλεύουν με την ώρα, και να μοιράσει αρμοδιότητες.

Το παιχνίδι θα διαθέτει ένα πλήθος υπαλλήλων, όπου ο κάθε υπάλληλος θα έχει τέσσερις ειδικότητες:
α) Αναλυτής, β) Σχεδιαστής, γ) Προγραμματιστής και δ) Tester.

Ο υπάλληλος θα έχει αξιολογηθεί και θα χαρακτηρίζεται από έναν αριθμό ως βαθμό ικανότητας σε κάθε μια από τις παραπάνω ειδικότητες. Αυτός ο αριθμός θα κυμαίνεται από το 0(καθόλου) έως το 100(πολύ).

Εκτός από τις τέσσερις ειδικότητες που έχει ένας υπάλληλος, διαθέτει και κάποια χαρακτηριστικά προσωπικότητας, τα οποία καταγράφουν τα ιδιαίτερα γνωρίσματα κάθε ανθρώπου σε ότι αφορά τον τρόπο που αυτός σκέφτεται, δρα, λειτουργεί, συμπεριφέρεται, αντιδρά κάτω από διαφορετικά ερεθίσματα, σχέσεις και καταστάσεις της ζωής και της εργασίας του.

Η μέθοδος MBTI [Myers, 1962] αναφέρεται στις προτιμήσεις προσωπικότητας, οι οποίες δηλώνουν αυτό που αρέσει, αυτό που συνήθως προτιμά και επιλέγει ο κάθε άνθρωπος. Οι προτιμήσεις προσωπικότητας διακρίνονται σε τέσσερεις τομείς – διχοτομίες. Για κάθε τομέα υπάρχουν δύο προτιμήσεις αντίθετες μεταξύ τους, γι’ αυτό κάθε τομέας ονομάζεται και διχοτομία. Αυτές είναι:

α) Εξωστρέφεια (Extraversion [E]) - Εσωστρέφεια (Introversion [I]) : από ποιόν κόσμο ο άνθρωπος αντλεί ενέργεια, τον εξωτερικό κόσμο ή το δικό του εσωτερικό κόσμο.

β) Αίσθηση (Sensing [S]) - Διαίσθηση (INtuition [N]) : Ο τρόπος που αντιλαμβάνεται ο άνθρωπος την πληροφορία που υπάρχει παντού γύρω του. Μέσω των αισθήσεων ή της διαίσθησης.

γ) Σκέψη (Thinking [T]) - Συναίσθημα (Feeling [F]) : Με ποιό τρόπο ο άνθρωπος παίρνει αποφάσεις. Εξετάζει τα θέματα με λογική και συνέπεια ή ακολουθεί τα συναισθήματα του.

δ) Κρίση (Judging [J]) - Παρατήρηση (Perceiving [P]) : Στη συναλλαγή του με τον εξωτερικό κόσμο ο άνθρωπος προτιμά τα πράγματα να είναι δομημένα με μια σειρά ή είναι ανοιχτός σε νέες πληροφορίες και επιλογές.

Κάθε τύπος προσωπικότητας ορίζεται σαν ένας κωδικός τεσσάρων γραμμάτων, κάθε γράμμα αντιπροσωπεύει μία από τις τέσσερεις διχοτομίες. Χρησιμοποιώντας τον κωδικό των τεσσάρων γραμμάτων για όλες τις προτιμήσεις προκύπτουν 16 συνδυασμοί προτιμήσεων, κάθε συνδυασμός παριστάνει ένα τύπο προσωπικότητας. Για παράδειγμα ένας τύπος προσωπικότητας ενός υπαλλήλου θα μπορούσε να ήταν ENTP. Χρησιμοποιώντας αυτή την μέθοδο, ο φοιτητής θα βλέπει συνολικά 5 χαρακτηριστικά για κάθε υπάλληλο (4 ειδικότητες και 1 τύπο προσωπικότητας). Η κατανομή των τύπων προσωπικότητας των διαθέσιμων

υπαλλήλων, έγινε με βάση μια έρευνα που μελέτησε τους πιο συχνούς τύπους τεχνολόγων λογισμικού στις ΗΠΑ. Η συχνότητα εμφάνισης του κάθε τύπου έγινε με βάση τον Πίνακα 8 [Capretz, 2003].

Πίνακας 8: Συχνότητα εμφάνισης του κάθε τύπου προσωπικότητας

ISTJ 24% $R = 2.08$	ISFJ 2% $R = 0.14$	INFJ 1% $R = 0.68$	INTJ 7% $R = 3.40$
ISTP 8% $R = 1.49$	ISFP 5% $R = 0.57$	INFP 2% $R = 0.46$	INTP 8% $R = 2.46$
ESTP 8% $R = 1.87$	ESFP 1% $R = 0.12$	ENFP 3% $R = 0.37$	ENTP 7% $R = 2.19$
ESTJ 15% $R = 1.73$	ESFJ 4% $R = 0.33$	ENFJ 1% $R = 0.41$	ENTJ 4% $R = 2.23$

Η συγκεκριμένη έρευνα έδειξε ότι υπήρχαν περισσότεροι τύποι Εσωστρεφείς(I=57%) παρά Εξωστρεφείς (E=43%). Αρκετά περισσότεροι τύποι με προσωπικότητα Αίσθησης (S=67%) παρά Διαίσθησης (N=33%), Σκέψης (T=81%) παρά Συναίσθημα (F=19%) και Κρίσης (J=58%) παρά Παρατήρηση (P=42%). Πιο συγκεκριμένα, αυτή η έρευνα έδειξε ότι οι τύποι ISTJ, ISTP, ESTP και ESTJ συνθέτουν πάνω από 50% του δείγματος, ενώ οι τύποι INFJ, ESFP και ENFJ είναι ανεπαρκή. Ο τύπος προσωπικότητας που χρησιμοποιείται περισσότερο είναι ο ISTJ (24%).

Σε επιχειρησιακό επίπεδο, η μελέτη και σκιαγράφηση του ατομικού προφίλ υπαλλήλων είναι σημαντική, έτσι ώστε να τοποθετείται ο κατάλληλος άνθρωπος στην κατάλληλη θέση και να επιτυγχάνεται το μέγιστο της απόδοσης. Όταν συμβαίνει αυτό, δηλαδή ο βέλτιστος συνδυασμός ανθρώπου – ρόλου, τότε η επιχείρηση απολαμβάνει την μέγιστη παραγωγικότητα και την καλύτερη αξιοποίηση του Ανθρωπίνου δυναμικού. Επιπλέον, σύμφωνα με το [Sfetsos et al., 2009] οι ομάδες ανάπτυξης λογισμικού οφείλουν να είναι ετερογενείς ώστε να αποδίδουν καλύτερα.

Πιο συγκεκριμένα, στα [Capretz and Gorla, 2004] οι συγγραφείς υποστηρίζουν ότι οι μηχανικοί λογισμικού είναι πλέον πιθανοί να είναι τύπου STs ή TJs ή NTs. Τα αποτελέσματα είναι σημαντικά στους εργοδότες που ψάχνουν τους επαγγελματίες λογισμικού και στους σπουδαστές που ψάχνουν τις σταδιοδρομίες. Σύμφωνα με τη θεωρία MBTI, το NTs τείνει να είναι δημιουργικότερο από τον τύπο STs επειδή το NS βλέπει τις δυνατότητες πέρα από τα δεδομένα γεγονότα, και ψάχνει τα σχέδια και τις σχέσεις. Είναι πιο έμπειροι στον προσδιορισμό των ελλοχευουσών αρχών από, τι στην απομνημόνευση των συγκεκριμένων στοιχείων. Το NTs συνδέει ένα θεωρητικό πλαίσιο και την τάση να παρατείνει πέρα από τις λεπτομέρειες, έτσι ώστε οι νέες αρχές να μπορούν να δουν. Αφ' ενός, πολλοί NFs και SFs σύρονται στους τομείς όπως την ψυχολογία και τη σχολική διδασκαλία λόγω της ανησυχίας τους για τους άλλους. Από τα στοιχεία αυτά, μπορεί να διαπιστωθεί ότι η πλειοψηφία των μηχανικών λογισμικού (ISTJ) είναι τεχνικά προσανατολισμένη και προτιμά τη συνεργασία με τα γεγονότα και το λόγο παρά με τους ανθρώπους. Σύμφωνα με τις ιδιότητες που συνδέθηκαν με κάθε τύπο από τη θεωρία MBTI, θα μπορούσαμε να συμπεράνουμε ότι οι τύποι προσωπικότητας ESTJs, ESTPs, ENTJs και ENTPs φαίνονται να κάνουν τους καλούς αναλυτές συστημάτων λόγω της δυνατότητας σκέψης τους να λύσουν τα οργανωτικά προβλήματα και να επικοινωνήσουν με άλλους ανθρώπους. Το ESTPs και ENTPs θα προτιμήσουν συχνά να αφήσουν την εφαρμογή των σχεδίων τους σε άλλους. Αφ' ενός, το ESTJs και ENTJs είναι πιθανό να ακολουθήσει ένα πρόγραμμα μέσω του τέλους κατά συνέπεια, επιτυγχάνοντας την περάτωση που ο Js επιδιώκει. Ο τύπος ISTPs εμφανίζεται να είναι άριστος προγραμματιστής δεδομένου ότι έχει τις μεγάλες δεξιότητες για να επισημάνει το κέντρο ενός προβλήματος και να φανεί να βρίσκει τις πρακτικές λύσεις. Από τα προηγούμενα αποτελέσματα, θεωρούνται ότι το INTPs αποδίδει καλύτερα στον επιστημονικό προγραμματισμό. Το INTJs έχει μια υψηλή ανάγκη να επιτύχει, αν και μια χαμηλή κίνηση για να κοινωνικοποιήσουν με άλλους ανθρώπους. Συνεπεία της προτίμησης J, το ISTJs και το ESTJs τείνουν να είναι οργανωμένα και να κάνουν περισσότερο προγραμματισμό από άλλους. Στη βιομηχανία λογισμικού, το ESTJs και το ISTJs είναι πιθανότερο να επιδιώξουν ενεργά μια διοικητική θέση, ενώ τα ISTPs,

ESTPs, INTPs και ENTPs θα ήταν συχνά πολύ ευτυχή ακολουθώντας μια τεχνική πορεία σταδιοδρομίας.

Προσωπικότητα αρχηγών ομάδας

Η έρευνά μας επέδειξε πώς οι αρχηγοί ομάδας που σημειώθηκαν στη διάσταση συλλογής πληροφοριών (Αίσθηση(S)/Διαίσθηση(N)) είχαν έναν σημαντικό αντίκτυπο στην απόδοση ομάδων. Οι ομάδες που είχαν αρχηγό με τύπο προσωπικότητας Διαίσθηση(N) ξεπέρασε εκείνες τις ομάδες με αρχηγό με τύπο προσωπικότητας Αίσθηση(S). Οι διαισθητικοί τύποι είναι σε ευρεία βάση, ολόκληρος-εικόνα-προσανατολισμένος σχετικά με τα συστήματα και τα υποσυστήματα, και έχουν μια καινοτόμο δυνατότητα να δημιουργήσουν και να αξιολογήσουν τις εναλλασσόμενες λύσεις. Αυτά είναι επιθυμητά χαρακτηριστικά ενός αρχηγού ομάδας στις μικρές ομάδες, δεδομένου ότι μπορεί επίσης να πρέπει να εκτελέσει τις ευθύνες ενός αναλυτή συστημάτων. Όσον αφορά τη διάσταση λήψης αποφάσεων, τα αποτελέσματά μας δείχνουν ότι οι ομάδες με έναν αρχηγό ομάδας τύπων Συναισθήματος (F) ξεπέρασαν εκείνων με έναν τύπου Σκέψη (T) αρχηγό ομάδας τύπων. Το γεγονός ότι το αισθανόμενο πρόσωπο είναι λαϊκό και καθιστά τις αποφάσεις βασισμένες στον τρόπο με τον οποίο έχουν επιπτώσεις στα άτομα, μπορεί να βοηθήσει να κάνει εκείνο τον τύπο προσωπικότητας έναν αποτελεσματικό διευθυντή. Αυτή η άποψη υποστηρίχθηκε από μια δήλωση από έναν από τους προγραμματιστές σε μια υψηλής απόδοσης ομάδα.

Προσωπικότητα αναλυτών συστημάτων

Μόνο η διάσταση λήψης αποφάσεων (Αίσθηση(S)/Διαίσθηση(N)) της προσωπικότητας αναλυτών συστημάτων είχε μια σημαντική επιρροή στην απόδοση ομάδων. Οι ομάδες με σκεπτόμενο τύπο προσωπικότητας (T) αναλυτή συστημάτων ξεπέρασαν εκείνων με έναν αναλυτή συστημάτων τύπων συναισθήματος (F). Εμφανίζεται ότι οι αναλυτικές δεξιότητες ενός αναλυτή συστημάτων είναι σημαντικότερες από τις συμπεριφοριστικές δεξιότητες στις μικρές ομάδες. Ενώ στις μεγαλύτερες ομάδες, ο αναλυτής συστημάτων μπορεί να διοριστεί πρώτιστα σε τέτοιους στόχους όπως τον προσδιορισμό απαίτησης και την προδιαγραφή

συστημάτων, στις μικρότερες ομάδες ο αναλυτής συστημάτων πρέπει να εκτελέσει τους πολλαπλάσιους στόχους. Μπορεί να πρέπει να συμμετέχει στο σχέδιο συστημάτων και προγραμματισμός εκτός από τα συστήματα της ανάλυσης. Η προσωπικότητα τύπων σκέψης ταιριάζει καλύτερα σε μια τέτοια σειρά των αναλυτικών στόχων.

Προσωπικότητα προγραμματιστών

Μόνο η κοινωνική διάσταση αλληλεπίδρασης (ενδοστρέφεια/εξωστρέφεια) της προσωπικότητας προγραμματιστών αφορούσε έντονα την απόδοση ομάδων. Αυτή η εύρεση ήταν εκπληκτική, αλλά οι ομάδες με τους εξωστρεφείς (E) προγραμματιστές ξεπέρασαν εκείνους με τους εσωστρεφείς (I) τύπους. Αυτό μπορεί να εξηγηθεί από τους προγραμματιστές γεγονότος που πρέπει να αλληλεπιδράσει με διάφορα συμβαλλόμενα μέρη, συμπεριλαμβανομένων των αναλυτών συστημάτων, των σχεδιαστών, άλλους προγραμματιστές, χειριστές υπολογιστών, και χειριστές εισαγωγών δεδομένων. Επιπλέον, στις μικρές ομάδες προγράμματος, οι προγραμματιστές μπορούν επίσης να ενεργήσουν ως αναλυτές συστημάτων και να αλληλεπιδράσουν με το διάφορο προσωπικό τμημάτων χρηστών. Ένας εξωστρεφής προγραμματιστής μπορεί να ταιριάξει καλύτερα για τέτοιους ευρέως κυμαινόμενους στόχους.

Πίνακας 9: Απόδοση Προσωπικότητας Υπαλλήλου ανά Φάση

Προσωπικότητα	***Ανάλυση***	***Σχεδίαση***	***Υλοποίηση***	***Έλεγχος***
ESTJ	1,20	1,20	1,10	1,10
ESTP	1,20	1,00	0,90	1,00
ESFJ	0,90	0,90	1,00	0,90
ESFP	0,90	0,90	1,00	0,90
ENTJ	1,20	1,20	1,10	1,10
ENTP	1,20	1,00	0,90	1,10
ENFJ	0,90	0,90	1,00	0,90
ENFP	0,90	0,90	1,00	0,90

ISTJ	1,10	1,10	0,90	1,00
ISTP	0,90	0,90	1,20	1,00
ISFJ	0,90	0,90	1,00	0,90
ISFP	0,90	0,90	1,00	0,90
INTJ	0,90	0,90	1,20	1,10
INTP	1,10	1,10	1,20	1,10
INFJ	0,90	0,90	1,00	0,90
INFP	0,90	0,90	1,00	1,00

Ο κάθε υπάλληλος έχει ένα συγκεκριμένο ωρομίσθιο. Στη βάση θα υπάρχουν κάποιοι υπάλληλοι οι οποίοι θα είναι αρκετά καλοί στα περισσότερα χαρακτηριστικά σε συνδυασμό με χαμηλό κόστος. Ο φοιτητής που θα αρχίσει νωρίς το παιχνίδι θα μπορεί να διαλέγει τους κατάλληλους υπαλλήλους με χαμηλό κόστος, ενώ οι άλλοι φοιτητές που θα αργήσουν να αρχίσουν το παιχνίδι θα πρέπει να προσλάβουν υπαλλήλους από αυτούς που έμειναν. Ο φοιτητής θα κρίνει πόσα άτομα θα πρέπει να προσλάβει για την κάθε φάση (Ανάλυση, Σχεδίαση, Προγραμματισμός, Testing) καθώς και πόσες ώρες θα τους αξιοποιήσει σε κάθε φάση. Αυτή η διαδικασία είναι η βάση του project, γιατί με αυτή την επιλογή θα κριθεί εάν ο φοιτητής έχει χρησιμοποιήσει κατάλληλα το budget που του έχει δοθεί για να ολοκληρώσει το έργο.

Αφού ο φοιτητής έχει προσλάβει τους υπαλλήλους, επιλέγει ποιός υπάλληλος θα απασχοληθεί σε ποιά φάση, βάσει των χαρακτηριστικών του κάθε υπαλλήλου. Για παράδειγμα, εάν ένας υπάλληλος στο χαρακτηριστικό Αναλυτής έχει βαθμό ίσο με 70, αυτό σημαίνει ότι θα είναι αρκετά καλός στην φάση της Ανάλυσης, οπότε ο φοιτητής μπορεί να τον αξιοποιήσει σε αυτή την φάση. Ο φοιτητής κρίνει εάν θα απασχολεί όλους τους υπαλλήλους σε κάθε φάση. Δηλαδή μπορεί να έχει επιλέξει τρεις υπαλλήλους στην φάση της Ανάλυσης αλλά να χρησιμοποιήσει και άλλους δύο, οι οποίοι να μην είναι αρκετά καλοί(να έχουν στο χαρακτηριστικό Αναλυτής βαθμό ίσο με 50). Αυτή η επιλογή μπορεί να βοηθήσει τον φοιτητή να αξιοποιήσει καλά τους υπαλλήλους του αλλά μπορεί και να προκαλέσει ζημιά στο project του.

Υπάρχει περίπτωση το project να αποτύχει για διάφορους λόγους, όπως η λανθασμένη επιλογή των υπαλλήλων. Ο φοιτητής που θα έχει επιλέξει τους

κατάλληλους υπαλλήλους με χαμηλό κόστος σημαίνει ότι έχει κάνει σωστή επιλογή ενώ ο φοιτητής που επέλεξε υπαλλήλους είτε καλοί είτε μέτριοι με υψηλό κόστος μπορεί να προκαλέσει την αποτυχία του project. Άλλη περίπτωση αποτυχίας είναι η λάθος επιλογή αρμοδιοτήτων στους υπαλλήλους.

Ο φοιτητής εκτός από τις αρμοδιότητες που δίνει σε κάθε υπάλληλο, θα πρέπει να ορίσει και πόσο χρονικό διάστημα κρατάει η κάθε φάση. Η μετάβαση στην επόμενη φάση ανάπτυξης γίνεται κατά την ολοκλήρωση της προηγούμενης φάσης.

Η αξιολόγηση του χρονοπρογραμματισμού του έργου για κάθε φάση γίνεται ως εξής:

ΙΠΕΦ = Ιδανικό budget $*$ ποσοστό_ φάσης$_i$

ΙΠΕΦ = Ιδανικό ποσό επένδυσης στη φάση$_i$

$$\text{ΙΩΕΦ} = \frac{\text{ΙΠΕΦ}}{\text{ΜΟΜΑ}}$$

ΜΟΜΑ = Μέσος όρος μισθού αναλυτή στη βάση
ΙΩΕΦ = Ιδανικές ώρες επένδυσης στη φάση$_i$

Για την εταιρία του φοιτητή:

$$\text{ΠΕΦ} = \sum_{i=0}^{i=\pi\lambda\acute{\eta}\theta o\varsigma\_\alpha\nu\alpha\lambda\upsilon\tau\acute{\omega}\nu} \Omega E_i * \text{ΙΥ}_i * \text{ΑΠΑ}_i$$

ΠΕΦ = Προσπάθεια εταιρίας για κάθε Φάση
ΩΕ = Ώρες εργασίας
ΙΥ = Ικανότητα Υπαλλήλου

ΑΠΑ = Απόδοση Προσωπικότητας Υπαλλήλου: Υπολογίζετε για κάθε φάση με βάση τον Πίνακα 9

$$\text{Αξιολόγηση Ομάδας} = \frac{\sum_{i=0}^{i=\pi\lambda\eta\theta o\varsigma\_\zeta\varepsilon\upsilon\gamma\alpha\rho\iota\acute{\omega}\nu\_\upsilon\pi\alpha\lambda\lambda\acute{\eta}\lambda\omega\nu} \Pi\Sigma\Pi}{\Pi Z Y * 4}$$

ΠΣΠ = Πλήθος στοιχείων προσωπικότητας που διαφέρει ο Y_i από τον Y_{i+1}

ΠΖΥ = Πλήθος ζευγαριών υπαλλήλων

ΑΠΟΔΟΣΗ = Ανάλυση $*$ Αξιολόγηση Ομάδας

Αξιολόγηση Ομάδας : Ποσοστό Διαφορετικότητας Χαρακτήρων στην Ομάδα

$$\text{ΑΞΙΟΛΟΓΗΣΗ} = \frac{\Pi E \Phi}{I \Omega E \Phi}$$

Ανάλογα με τον βαθμό που βγαίνει, χρησιμοποιώντας την κλίμακα αξιολόγησης, διαπιστώνεται ότι όσο πιο κοντά είναι στο 1 τόσο καλύτερο είναι το project.

Κλίμακα αξιολόγησης:

(0 έως 0,2) κακό
(0,2 έως 0,4) μέτριο
(0,4 έως 0,6) καλό
(0,6 έως 0,8) πολύ καλό
(0,8 έως 1,5) άριστο
(1,5 <) καλό

Αφού ο χρονοπρογραμματισμός του έργου έχει αξιολογηθεί, ξεκινάει η φάση της Ανάλυσης. Μέσα στο χρονικό διάστημα που έχει ορίσει ο φοιτητής για αυτή την φάση, θα δέχεται τέσσερα είδη διαγραμμάτων, κατά σειρά (α).Διάγραμμα Π.Χ, (β).Προδιαγραφές Π.Χ, (γ).Διάγραμμα Ακολουθίας Συστήματος και (δ).Εννοιολογικό Μοντέλο και θα πρέπει να τα αξιολογεί.

Συγκεκριμένα, θα δέχεται ένα email όπου θα ενημερώνεται ότι το συγκεκριμένο διάγραμμα έχει ολοκληρωθεί από τους υπαλλήλους του και ότι του έχει σταλεί. Το πρώτο διάγραμμα που θα λάβει θα είναι το Διάγραμμα Π.Χ, στο οποίο θα έχει ανατεθεί μία βαθμολογία με βάση την αξιολόγηση της κάθε φάσης. Δηλαδή, για το Διάγραμμα Π.Χ υπάρχουν 5 διαφορετικά διαγράμματα και ανάλογα σε ποιο διάστημα βρίσκεται ο αριθμός στέλνεται και το αντίστοιχο διάγραμμα.

Ο φοιτητής δεν γνωρίζει την αξιολόγηση του διαγράμματος, οπότε δεν γνωρίζει εάν το διάγραμμα που του έχει σταλεί είναι ικανοποιητικής ποιότητας. Θα μπορεί όμως να κάνει αλλαγές πάνω στο διάγραμμα, κάτι που μπορεί να αυξήσει την βαθμολογία του διαγράμματος ή να την μειώσει. Στη συνέχεια θα κάνει upload το διάγραμμα. Ο καθηγητής θα εισέρχεται στο σύστημα και θα βλέπει το διάγραμμα που έχει στείλει ο φοιτητής. Ο καθηγητής θα μπορεί να δει τον βαθμό που έχει πάρει εξ' αρχής το διάγραμμα που του έχει σταλεί. Στη συνέχεια βλέπει τις αλλαγές που έχει πραγματοποιήσει ο φοιτητής και έχει το δικαίωμα να αυξήσει ή να μειώσει τον βαθμό κατά 0,1 μονάδα. Για παράδειγμα, εάν το διάγραμμα είχε βαθμό 0,6 (με άριστα το 1) και οι αλλαγές που έκανε ήταν σωστές τότε ο βαθμός του από 0,6 θα πάει στο 0,7. Στη συνέχεια ο φοιτητής μετά από κάποιες μέρες, θα δεχθεί το επόμενο email όπου θα τον ενημερώνει για τις Προδιαγραφές Π.Χ. Εάν ο φοιτητής στο πρώτο διάγραμμα έχει πάρει βαθμό ίσο με 0,7 στο δεύτερο διάγραμμα θα παραλάβει ένα διάγραμμα με βαθμολογία 0,7. Η διαδικασία συνεχίζεται έως ότου τελειώσουν τα διαγράμματα στη φάση της Ανάλυσης.

Για τη μετάβαση από τη μία φάση στην επόμενη ακολουθείται η παρακάτω διαδικασία:

$$Α.Β.Φ_{i+1} = ΑΞΙΟΛΟΓΗΣΗ_{i+1} * Τ.Β.Φ_{i+1}$$

Α.Β.Φ = Αρχικός Βαθμός Φάσης

Τ.Β.Φ = Τελικός Βαθμός Φάσης

Παραδείγματος χάριν, έστω ότι η φάση της Ανάλυσης ολοκληρώνεται με βαθμό αξιολόγησης ίσο με 0,7. Η αξιολόγηση της φάσης Σχεδίασης στον χρονοπρογραμματισμό είναι 0,9. Άρα, το πρώτο διάγραμμα της φάσης Σχεδίασης θα βαθμολογηθεί με $0.9 * 0.7 = 0,63$.

Μέσα στο χρονικό διάστημα που έχει ορίσει ο φοιτητής για αυτή την φάση, θα δέχεται δύο είδη διαγραμμάτων κατά σειρά (α).Διάγραμμα Ακολουθίας και (β).Διάγραμμα Κλάσεων και θα πρέπει να τα αξιολογήσει. Η όλη διαδικασία είναι όμοια με την διαδικασία στην φάση της Ανάλυσης. Η διαδικασία τελειώνει όταν σταλούν και τα δύο διαγράμματα πάντα μέσα στο χρονικό διάστημα που έχει ορίσει ο φοιτητής ότι διαρκεί η φάση της Σχεδίασης.

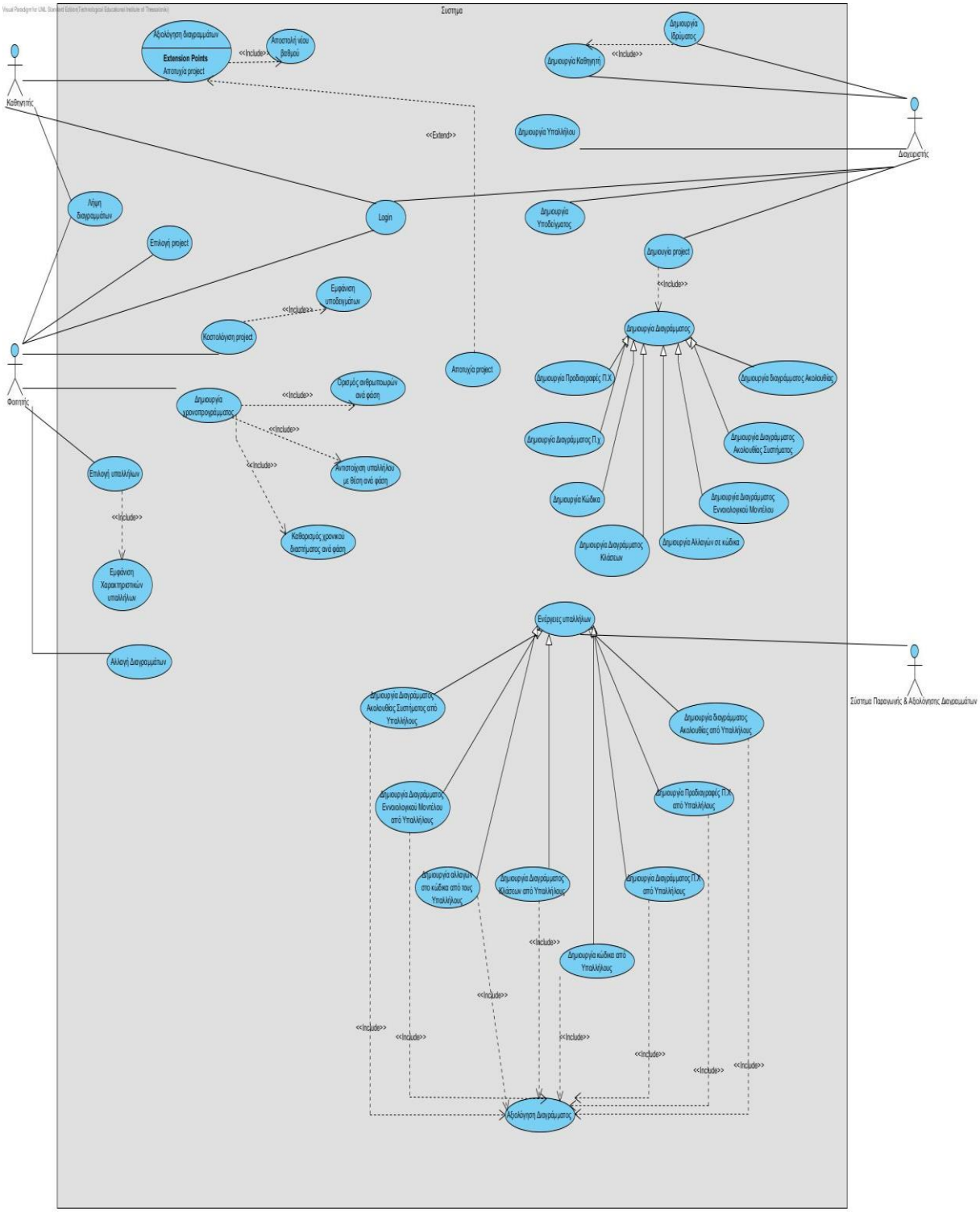
Στη συνέχεια πηγαίνουμε στην φάση της Υλοποίησης. Για τη μετάβαση από τη φάση της Σχεδίασης στη φάση της Υλοποίησης πραγματοποιείται η ίδια διαδικασία με παραπάνω. Ανάλογα με το βαθμό εκκίνησης της φάσης, ο φοιτητής δέχεται ένα τμήμα κώδικα και 5 προτάσεις για να αλλάξει τον κώδικα που είχε λάβει προηγουμένως. Ο φοιτητής επιλέγει έως τρεις προτάσεις αναδόμησης και τεκμηριώνει την επιλογή του. Ο καθηγητής αξιολογεί τις επιλογές του και αυξάνει ή μειώνει κατά 0,1 μονάδα. Ο βαθμός αναπαριστά την δομική ποιότητα του κώδικα(structural code quality). Ανάλογα με την ποιότητα του λογισμικού σε συνδυασμό με την αξιολόγηση του χρονοπρογραμματισμού στη φάση του testing θα του επιστρέφει το πρόγραμμα ένα συγκεκριμένο κομμάτι κώδικα και ο φοιτητής θα πρέπει να το διορθώσει και στη συνέχεια να το ξανά στείλει. Για παράδειγμα, εάν τα λάθη του είναι 10 θα του επιστρέφει 1 λάθος για να το διορθώσει. Ο καθηγητής βλέπει το πλήθος των λαθών που έχει το project και σε συνδυασμό και με τις διορθώσεις που έχει κάνει ο φοιτητής μπορεί να προσθέσει ή να μειώσει τον τελικό βαθμό κατά 0,1 μονάδα.

Τέλος, το πρόγραμμα ελέγχει τον τελικό αριθμό και κρίνει εάν το project πέτυχε ή απέτυχε. Το όριο επιτυχίας ή αποτυχίας του κάθε project υπάρχει κατοχυρωμένο στη βάση. Εάν πέτυχε, τότε ο φοιτητής λαμβάνει κάποια αμοιβή και ανάλογα με το χρονικό διάστημα που του έχει απομείνει μπορεί να πάρει και επόμενο project με τα

χρήματα που διαθέτει. Στην περίπτωση που το project αποτύχει, ο φοιτητής έχει δικαίωμα να ασχοληθεί με άλλο project με το budget που του έχει απομείνει και μέσα στο χρονικό διάστημα που του έχει απομείνει.

4.2 Διάγραμμα Περιπτώσεων Χρήσης

Σχήμα 22: Διάγραμμα Π.Χ



4.3 Περιγραφή Περιπτώσεων Χρήσης

Πίνακας 10: Δημιουργία Διαγράμματος Π.Χ

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ Π.Χ			
Use Case ID	UC-A1		
Super Use Case			
Primary Actor	Διαχειριστής		
Secondary Actor(s)			
Brief Description	Το σύστημα δημιουργεί το διάγραμμα Π.Χ για να το δει ο φοιτητής		
Preconditions			
Flow of Events		**Actor Input**	**System Response**
	1	Ο διαχειριστής επιθυμεί να δημιουργήσει διαγράμματα Π.Χ.	
	2		Το σύστημα ζητάει από τον διαχειριστή τα πέντε Διαγράμματα Π.Χ.
	3	Ο διαχειριστής δίνει τα αρχεία που του ζητάει το σύστημα.	
	4		Αποθηκεύονται τα αρχεία στο server.
	5		Επιστροφή στην φόρμα «Δημιουργία Project»
Εναλλακτική ροή (Σενάριο):			

4α

	Actor Input	System Response
1		Ο χρήστης δεν έχει συμπληρώσει και τα πέντε πεδία. Επιστροφή στο Βήμα «1»

4β

	Actor Input	System Response
1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1»

Πίνακας 11: Δημιουργία Προδιαγραφών Π.Χ

Full – ΔΗΜΙΟΥΡΓΙΑ ΠΡΟΔΙΑΓΡΑΦΕΣ Π.Χ

Use Case ID	UC-A2
Super Use Case	
Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Το σύστημα ανεβάζει τις Προδιαγραφές Π.Χ για να τις δει ο φοιτητής
Preconditions	Θα πρέπει να έχει γίνει η Π.Χ «Δημιουργία Διαγράμματος Π.Χ»

Flow of Events

	Actor Input	System Response
1	Ο διαχειριστής επιθυμεί να δημιουργήσει Προδιαγραφές Π.Χ.	
2		Το σύστημα ζητάει από τον διαχειριστή τις πέντε Προδιαγραφές Π.Χ.
3	Ο διαχειριστής δίνει τα αρχεία που του ζητάει το σύστημα.	
4		Αποθηκεύονται τα αρχεία στο server.

5		Επιστροφή στην φόρμα «Δημιουργία Project».

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Ο χρήστης δεν έχει συμπληρώσει και τα πέντε πεδία. Επιστροφή στο Βήμα «1»

4β

	Actor Input	System Response
1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1»

Πίνακας 12: Δημιουργία Διαγράμματος Ακολουθίας

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΑΚΟΛΟΥΘΙΑΣ	
Use Case ID	UC-A3
Super Use Case	
Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Το σύστημα ανεβάζει το Διάγραμμα Ακολουθίας για να το δει ο φοιτητής
Preconditions	

Flow of Events

	Actor Input	System Response
1	Ο διαχειριστής επιθυμεί να δημιουργήσει Διαγράμματα Ακολουθίας.	
2		Το σύστημα ζητάει από τον διαχειριστή τα πέντε Διαγράμματα Ακολουθίας.
3	Ο διαχειριστής δίνει τα αρχεία που του ζητάει το σύστημα.	
4		Αποθηκεύονται τα αρχεία στο server.
5		Επιστροφή στην φόρμα «Δημιουργία Project».

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Ο χρήστης δεν έχει συμπληρώσει και τα πέντε πεδία. Επιστροφή στο Βήμα «1»

4β

	Actor Input	System Response
1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1»

Πίνακας 13: Δημιουργία Διαγράμματος ακολουθίας Συστήματος

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΑΚΟΛΟΥΘΙΑΣ ΣΥΣΤΗΜΑΤΟΣ

Use Case ID	UC-A4
Super Use Case	
Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Το σύστημα ανεβάζει το Διάγραμμα Ακολουθίας Συστήματος για να το δει ο φοιτητής
Preconditions	

Flow of Events

	Actor Input	System Response
1	Ο διαχειριστής επιθυμεί να δημιουργήσει Διαγράμματα Ακολουθίας Συστήματος.	
2	.	Το σύστημα ζητάει από τον διαχειριστή τα πέντε Διαγράμματα Ακολουθίας Συστήματος.
3	Ο διαχειριστής δίνει τα αρχεία που του ζητάει το σύστημα	
4		Αποθηκεύονται τα αρχεία στο server.
5		Επιστροφή στην φόρμα «Δημιουργία Project».

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Ο χρήστης δεν έχει συμπληρώσει και τα πέντε πεδία. Επιστροφή στο Βήμα «1»

4β

	Actor Input	System Response
1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».

3		Επιστροφή στο Βήμα «1»

Πίνακας 14: Δημιουργία Διαγράμματος Κλάσεων

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΚΛΑΣΕΩΝ			
Use Case ID	UC-A5		
Super Use Case			
Primary Actor	Διαχειριστής		
Secondary Actor(s)			
Brief Description	Το σύστημα ανεβάζει το Διάγραμμα Κλάσεων για να το δει ο φοιτητής		
Preconditions			
Flow of Events		**Actor Input**	**System Response**
	1	Ο διαχειριστής επιθυμεί να δημιουργήσει Διαγράμματα Κλάσεων.	
	2		Το σύστημα ζητάει από τον διαχειριστή τα πέντε Διαγράμματα Κλάσεων.
	3	Ο διαχειριστής δίνει τα αρχεία που του ζητάει το σύστημα.	
	4		Αποθηκεύονται τα αρχεία στο server.
	5		Επιστροφή στην φόρμα «Δημιουργία Project».

<u>Εναλλακτική ροή (Σενάριο):</u>

		Actor Input	**System Response**
4α	1		Ο χρήστης δεν έχει συμπληρώσει και τα

			πέντε πεδία. Επιστροφή στο Βήμα «1»
4β		**Actor Input**	**System Response**
	1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
	2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».
	3		Επιστροφή στο Βήμα «1»

Πίνακας 15: Δημιουργία Διαγράμματος Εννοιολογικού Μοντέλου

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΕΝΝΟΙΟΛΟΓΙΚΟΥ ΜΟΝΤΕΛΟΥ

Use Case ID	UC-A6
Super Use Case	
Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Το σύστημα ανεβάζει το Διάγραμμα Εννοιολογικού Μοντέλου για να το δει ο φοιτητής
Preconditions	

Flow of Events

	Actor Input	**System Response**
1	Ο διαχειριστής επιθυμεί να δημιουργήσει Διαγράμματα Εννοιολογικού Μοντέλου.	
2		Το σύστημα ζητάει από τον διαχειριστή τα πέντε Διαγράμματα Εννοιολογικού Μοντέλου.
3	Ο διαχειριστής δίνει τα αρχεία που του ζητάει το σύστημα.	
4		Αποθηκεύονται τα αρχεία στο server.

5		Επιστροφή στην φόρμα «Δημιουργία Project».

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Ο χρήστης δεν έχει συμπληρώσει και τα πέντε πεδία. Επιστροφή στο Βήμα «1»

4β

	Actor Input	System Response
1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1»

Πίνακας 16: Δημιουργία Ανέβασμα Κώδικα

Full – ΔΗΜΙΟΥΡΓΙΑ ΑΝΕΒΑΣΜΑ ΚΩΔΙΚΑ	
Use Case ID	UC-A7
Super Use Case	
Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Το σύστημα ανεβάζει ένα κομμάτι κώδικα για να το δει ο φοιτητής
Preconditions	

Flow of Events

	Actor Input	System Response
1	Ο διαχειριστής επιθυμεί να δημιουργήσει κώδικα.	
2		Το σύστημα ζητάει από τον διαχειριστή τα πέντε κομμάτια κώδικα.
3	Ο διαχειριστής δίνει τα αρχεία που του ζητάει το σύστημα.	
4		Αποθηκεύονται τα αρχεία στο server.
5		Επιστροφή στην φόρμα «Δημιουργία Project».

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Ο χρήστης δεν έχει συμπληρώσει και τα πέντε πεδία. Επιστροφή στο Βήμα «1»

4β

	Actor Input	System Response
1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1»

Πίνακας 17: Δημιουργία Αλλαγών στον Κώδικα

Full – ΔΗΜΙΟΥΡΓΙΑ ΑΛΛΑΓΩΝ ΣΤΟΝ ΚΩΔΙΚΑ

Use Case ID UC-A8

Super Use Case

Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Το σύστημα ανεβάζει τις αλλαγές του κώδικα.
Preconditions	

Flow of Events

	Actor Input	System Response
1	Ο διαχειριστής επιθυμεί να κάνει αλλαγές στον κώδικα.	
2		Το σύστημα ζητάει από τον διαχειριστή να ανεβάσει τις πέντε προτάσεις αλλαγής κώδικα που αντιστοιχεί σε ένα κομμάτι κώδικα.
3	Ο διαχειριστής δίνει τις αλλαγές κώδικα που του ζητάει το σύστημα.	
4		Αποθηκεύονται τα αρχεία στο server.
5		Επιστροφή στην φόρμα «Δημιουργία Project».

<u>Εναλλακτική ροή (Σενάριο):</u>

4α

	Actor Input	System Response
1		Ο χρήστης δεν έχει συμπληρώσει και τα πέντε πεδία. Επιστροφή στο Βήμα «1»

4β

	Actor Input	System Response
1		Τα αρχεία δεν είναι δυνατόν να αποθηκευτούν
2		Εμφάνιση μηνύματος «Τα αρχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1»

Πίνακας 18: Δημιουργία διαγράμματος Π.Χ από τους Υπαλλήλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ Π.Χ ΑΠΟ ΤΟΥΣ ΥΠΑΛΛΗΛΟΥΣ			
Use Case ID	UC-A9		
Super Use Case			
Primary Actor			
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων		
Brief Description	Οι υπάλληλοι δημιουργούν το διάγραμμα Π.Χ για να το δει ο φοιτητής		
Preconditions			
Flow of Events		**Actor Input**	**System Response**
	1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν ένα Διάγραμμα Π.Χ	
	2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος»
	3		Το σύστημα επιστρέφει στον φοιτητή το Διάγραμμα Π.Χ που δημιουργήθηκε στο βήμα «2».

Πίνακας 19: Δημιουργία Προδιαγραφών Π.Χ από τους Υπαλλήλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΠΡΟΔΙΑΓΡΑΦΕΣ Π.Χ ΑΠΟ ΤΟΥΣ ΥΠΑΛΛΗΛΟΥΣ			
Use Case ID	UC-A10		
Super Use Case			
Primary Actor			
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων		
Brief Description	Οι υπάλληλοι δημιουργούν τις Προδιαγραφές Π.Χ για να τις δει ο φοιτητής		
Preconditions	Ο καθηγητής έχει ολοκληρώσει την Π.Χ «Αποστολή νέου βαθμού» για το Διάγραμμα Π.Χ.		
Flow of Events		**Actor Input**	**System Response**
	1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν μια	

		Προδιαγραφή Π.Χ.	
	2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος»
	3		Το σύστημα επιστρέφει στον φοιτητή την Προδιαγραφή Π.Χ που δημιουργήθηκε στο βήμα «2».

Πίνακας 20: Δημιουργία Διαγράμματος Ακολουθίας από τους Υπαλληλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΑΚΟΛΟΥΘΙΑΣ ΑΠΟ ΤΟΥΣ ΥΠΑΛΛΗΛΟΥΣ	
Use Case ID	UC-A11
Super Use Case	
Primary Actor	
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων
Brief Description	Οι υπάλληλοι δημιουργούν το Διάγραμμα Ακολουθίας για να το δει ο φοιτητής
Preconditions	Ο καθηγητής έχει ολοκληρώσει την Π.Χ «Αποστολή νέου βαθμού» για το Διάγραμμα Εννοιολογικού Μοντέλου.

Flow of Events

	Actor Input	System Response
1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν ένα Διάγραμμα Ακολουθίας.	
2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος»
3		Το σύστημα επιστρέφει στον φοιτητή το Διάγραμμα Ακολουθίας που δημιουργήθηκε στο βήμα «2».

Πίνακας 21: Δημιουργία Διαγράμματος Ακολουθίας Συστήματος από Υπαλλήλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΑΚΟΛΟΥΘΙΑΣ ΣΥΣΤΗΜΑΤΟΣ ΑΠΟ ΥΠΑΛΛΗΛΟΥΣ	
Use Case ID	UC-A12
Super Use Case	
Primary Actor	

Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων		
Brief Description	Οι υπάλληλοι δημιουργούν το Διάγραμμα Ακολουθίας Συστήματος για να το δει ο φοιτητής		
Preconditions	Ο καθηγητής έχει ολοκληρώσει την Π.Χ «Αποστολή νέου βαθμού» για τις Προδιαγραφές Π.Χ.		
Flow of Events		**Actor Input**	**System Response**
	1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν ένα Διάγραμμα Ακολουθίας Συστήματος.	
	2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος».
	3		Το σύστημα επιστρέφει στον φοιτητή το Διάγραμμα Ακολουθίας Συστήματος που δημιουργήθηκε στο βήμα «2».

Πίνακας 22: Δημιουργία Διαγράμματος Κλάσεων από Υπαλλήλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΚΛΑΣΕΩΝ ΑΠΟ ΥΠΑΛΛΗΛΟΥΣ			
Use Case ID	UC-A13		
Super Use Case			
Primary Actor			
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων		
Brief Description	Οι υπάλληλοι δημιουργούν το Διάγραμμα Κλάσεων για να το δει ο φοιτητής		
Preconditions	Ο καθηγητής έχει ολοκληρώσει την Π.Χ «Αποστολή νέου βαθμού» για το Διάγραμμα Ακολουθίας.		
Flow of Events		**Actor Input**	**System Response**
	1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν ένα Διάγραμμα Κλάσεων.	

	2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος».
	3		Το σύστημα επιστρέφει στον φοιτητή το Διάγραμμα Κλάσεων που δημιουργήθηκε στο βήμα «2».

Πίνακας 23: Δημιουργία Διαγράμματος Εννοιολογικού Μοντέλου από Υπαλλήλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ ΕΝΝΟΙΟΛΟΓΙΚΟΥ ΜΟΝΤΕΛΟΥ ΑΠΟ ΥΠΑΛΛΗΛΟΥΣ

Use Case ID	UC-A14
Super Use Case	
Primary Actor	
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων
Brief Description	Οι υπάλληλοι δημιουργούν το Διάγραμμα Εννοιολογικού Μοντέλου για να το δει ο φοιτητής
Preconditions	Ο καθηγητής έχει ολοκληρώσει την Π.Χ «Αποστολή νέου βαθμού» για το Διάγραμμα Ακολουθίας Συστήματος.

Flow of Events

	Actor Input	System Response
1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν ένα Διάγραμμα Εννοιολογικού Μοντέλου.	
2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος».
3		Το σύστημα επιστρέφει στον φοιτητή το Διάγραμμα Εννοιολογικού Μοντέλου που δημιουργήθηκε στο βήμα «2».

Πίνακας 24: Δημιουργία Ανέβασμα Κώδικα από τους Υπαλλήλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΑΝΕΒΑΣΜΑ ΚΩΔΙΚΑ ΑΠΟ ΤΟΥΣ ΥΠΑΛΛΗΛΟΥΣ			
Use Case ID	UC-A15		
Super Use Case			
Primary Actor			
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων		
Brief Description	Οι υπάλληλοι δημιουργούν ένα κομμάτι κώδικα για να το δει ο φοιτητής		
Preconditions	Ο καθηγητής έχει ολοκληρώσει την Π.Χ «Αποστολή νέου βαθμού» για το Διάγραμμα Κλάσεων.		
Flow of Events		**Actor Input**	**System Response**
	1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν ένα κομμάτι κώδικα.	
	2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος».
	3		Το σύστημα επιστρέφει στον φοιτητή το κομμάτι κώδικα που δημιουργήθηκε στο βήμα «2».

Πίνακας 25: Δημιουργία Αλλαγών στον Κώδικα από τους Υπαλλήλους

Full – ΔΗΜΙΟΥΡΓΙΑ ΑΛΛΑΓΩΝ ΣΤΟΝ ΚΩΔΙΚΑ ΑΠΟ ΤΟΥΣ ΥΠΑΛΛΗΛΟΥΣ			
Use Case ID	UC-A16		
Super Use Case			
Primary Actor			
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων		
Brief Description	Οι υπάλληλοι δημιουργούν τις αλλαγές του κώδικα.		
Preconditions	Ο καθηγητής έχει ολοκληρώσει την Π.Χ «Αποστολή νέου βαθμού» για το κομμάτι κώδικα.		
Flow of Events		**Actor Input**	**System Response**

	1	Το σύστημα ζητάει από τους υπαλλήλους να δημιουργήσουν πέντε προτάσεις αλλαγής κώδικα που αντιστοιχεί σε ένα κομμάτι κώδικα.	
	2		Το σύστημα καλεί την Π.Χ «Αξιολόγηση Διαγράμματος».
	3		Το σύστημα επιστρέφει στον φοιτητή τις αλλαγές κώδικα που δημιουργήθηκε στο βήμα «2».

Πίνακας 26: Δημιουργία Καθηγητή

Full – ΔΗΜΙΟΥΡΓΙΑ ΚΑΘΗΓΗΤΗ			
Use Case ID	UC-A17		
Super Use Case			
Primary Actor	Διαχειριστής		
Secondary Actor(s)			
Brief Description	Δημιουργία καθηγητή από τον διαχειριστή.		
Preconditions	Να είναι εγγεγραμένο το εκπαιδευτικό ίδρυμα στο σύστημα στο οποίο ανήκει ο καθηγητής.		
Flow of Events		**Actor Input**	**System Response**
	1	Ο διαχειριστής ζητάει να εισάγει ένα νέο καθηγητή.	
	2		Το σύστημα εμφανίζει την φόρμα «Δημιουργία Καθηγητή»
	3	Ο διαχειριστής εισάγει τα στοιχεία του καθηγητή στο σύστημα.	
	4		Τα στοιχεία είναι σωστά. Το σύστημα ελέγχει εάν ο καθηγητής δεν είναι καταχωρημένος.

	5		Ο καθηγητής δεν είναι καταχωρημένος. Το σύστημα δημιουργεί και εμφανίζει σε μια φόρμα το username, το password, την ημερομηνία λήξης άδειας χρήσης καθώς και το mail του καθηγητή.
	5		Το σύστημα ζητάει το πλήθος των φοιτητών που θα παρακολουθήσουν το μάθημα.
	6	Ο διαχειριστής εισάγει το πλήθος των φοιτητών και ένα αρχείο με τα στοιχεία τω εγγεγραμμένων φοιτητών.	
	7		Το σύστημα δημιουργεί εμφανίζει σε μία φόρμα όλα τα username, password για τους φοιτητές.
	8		Το σύστημα στέλνει mail για τα στοιχεία., καλώντας την Π.Χ «Αποστολή email»

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Τα στοιχεία είναι λάθος. Ο διαχειριστής επιστρέφει στο Βήμα <<2>> και ρωτάει τον καθηγητή ξανά τα στοιχεία του.

5α

	Actor Input	System Response

	1		Το σύστημα εμφανίζει το μήνυμα «Είναι ήδη καταχωρημένος»
	2		Ο διαχειριστής επιστρέφει στο Βήμα <<2>> και ρωτάει τον καθηγητή ξανά τα στοιχεία του.

Πίνακας 27: Δημιουργία Ιδρύματος

Full – ΔΗΜΙΟΥΡΓΙΑ ΙΔΡΥΜΑΤΟΣ			
Use Case ID	UC-A18		
Super Use Case			
Primary Actor	Διαχειριστής		
Secondary Actor(s)			
Brief Description	Δημιουργία ιδρύματος από τον διαχειριστή.		
Preconditions			
Flow of Events		**Actor Input**	**System Response**
	1	Ο διαχειριστής ζητάει να εισάγει ένα νέο ίδρυμα.	
	2		Το σύστημα εμφανίζει την φόρμα «Δημιουργία Ιδρύματος»
	3	Ο διαχειριστής εισάγει τα στοιχεία του ιδρύματος στο σύστημα.	
	4		Τα στοιχεία είναι σωστά. Το σύστημα ελέγχει εάν το ίδρυμα είναι ήδη καταχωρημένο.
	5		Το ίδρυμα δεν είναι καταχωρημένο. Το σύστημα εμφανίζει σε μία φόρμα το όνομα του ιδρύματος, το όνομα του μαθήματος, το username καθώς και το password.

	6		Γίνεται include στην Π.Χ «Δημιουργία Καθηγητή»

4α **Εναλλακτική ροή (Σενάριο):**

	Actor Input	System Response
1		Τα στοιχεία είναι λάθος. Ο διαχειριστής επιστρέφει στο Βήμα <<2>> και ρωτάει το στοιχεία του ιδρύματος ξανά.

5α

B	Actor Input	System Response
1		Το σύστημα εμφανίζει το μήνυμα «Είναι ήδη καταχωρημένο το ίδρυμα»
2		Ο διαχειριστής επιστρέφει στο Βήμα <<2>> και ρωτάει το στοιχεία του ιδρύματος ξανά.

Πίνακας 28: Δημιουργία Υποδείγματος

Full – ΔΗΜΙΟΥΡΓΙΑ ΥΠΟΔΕΙΓΜΑΤΟΣ			
Use Case ID	UC-A19		
Super Use Case			
Primary Actor	Διαχειριστής		
Secondary Actor(s)			
Brief Description	Δημιουργία υποδειγμάτων στην βάση από τον διαχειριστή συστήματος.		
Preconditions			
Flow of Events		**Actor Input**	**System Response**
	1	Ο διαχειριστής θέλει να εισάγει ένα νέο υπόδειγμα.	

2		Το σύστημα εμφανίζει την φόρμα «Δημιουργία Υποδείγματος»
3	Ο διαχειριστής εισάγει στο σύστημα το νέο υπόδειγμα.	
4	Εισάγει στο σύστημα τον αντίστοιχο προυπολογισμό και το χρονοπρογραμματισμό του έργου.	
5		Το σύστημα αποθηκεύει τα στοιχεία του υποδείγματος στην βάση.
6		Το σύστημα ταξινομεί τα υποδείγματα ανά βαθμό δυσκολίας(εύκολο έως δύσκολο) με κριτήριο τον προϋπολογισμό που δίνει.

Εναλλακτική ροή (Σενάριο):

5α

	Actor Input	System Response
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν.
2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1»

Πίνακας 29: Δημιουργία Υπαλλήλου

Full – ΔΗΜΙΟΥΡΓΙΑ ΥΠΑΛΛΗΛΟΥ	
Use Case ID	UC-A20
Super Use Case	
Primary Actor	Διαχειριστής

Secondary Actor(s)

Brief Description Δημιουργία υπαλλήλων στην βάση από τον διαχειριστή συστήματος.

Preconditions

Flow of Events

	Actor Input	System Response
1	Ο διαχειριστής ζητάει να εισάγει ένα νέο υπάλληλο.	
2		Το σύστημα εμφανίζει την φόρμα «Δημιουργία Υπαλλήλου».
3	Ο διαχειριστής εισάγει στο σύστημα το όνομα και το επίθετο του υπαλλήλου.	
4		Το σύστημα ελέγχει εάν ο υπάλληλος είναι ήδη καταχωρημένος.
5		Ο υπάλληλος δεν είναι καταχωρημένος. Το σύστημα εμφανίζει σε μία φόρμα το όνομα και το επίθετο του υπαλλήλου.
6	Ο διαχειριστής εισάγει το ωρομίσθιο και τις υπερωρίες.	
7		Το σύστημα ελέγχει εάν το ωρομίσθιο και τις υπερωρίες είναι αριθμός.
8		Το ωρομίσθιο και οι υπερωρίες είναι αριθμός. Το σύστημα εμφανίζει στην φόρμα το ωρομίσθιο και τις υπερωρίες.

9	Ο διαχειριστής εισάγει τον βαθμό ικανότητας ανά φάση.	
10		Το σύστημα ελέγχει εάν ο βαθμός κυμαίνεται μεταξύ 0 και 100.
11		Ο βαθμός κυμαίνεται μεταξύ 0 και 100. Το σύστημα εμφανίζει στην φόρμα τον βαθμό ικανότητας ανά φάση.
12	Ο διαχειριστής εισάγει τον τύπο προσωπικότητας του υπαλλήλου.	
13		Το σύστημα ελέγχει εάν ο τύπος προσωπικότητας είναι ένας αποδεκτός συνδυασμός 4 γραμμάτων.
14		Ο τύπος προσωπικότητας είναι ένας αποδεκτός συνδυασμός 4 γραμμάτων. Το σύστημα εμφανίζει στην φόρμα τον τύπο προσωπικότητας του υπαλλήλου.
15		Το σύστημα αποθηκεύει τα στοιχεία του υπαλλήλου στην βάση.

Εναλλακτική ροή (Σενάριο):

5α

	Actor Input	System Response
1	.	Το σύστημα εμφανίζει το μήνυμα «Είναι ήδη καταχωρημένος ο υπάλληλος»
2	Ο διαχειριστής επιστρέφει στο «Βήμα 1»	

8α

	Actor Input	System Response
1	.	Το σύστημα εμφανίζει το μήνυμα «Το ωρομίσθιο και οι υπερωρίες δεν είναι αριθμός».
2	Ο διαχειριστής επιστρέφει στο «Βήμα 6»	

11α

	Actor Input	System Response
1		Το σύστημα εμφανίζει το μήνυμα «Ο βαθμός ικανότητας ανά φάση δεν κυμαίνεται μεταξύ 0 και 100»
2	Ο διαχειριστής επιστρέφει στο «Βήμα 9»	

14α

	Actor Input	System Response
1		Το σύστημα εμφανίζει το μήνυμα «Ο τύπος προσωπικότητας δεν είναι ένας κωδικός 4 γραμμάτων. »
2	Ο διαχειριστής επιστρέφει στο «Βήμα 12»	

15α

	Actor Input	System Response
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν.
2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».

3		Επιστροφή στο Βήμα «1».

Πίνακας 30: Login

Full – LOGIN	
Use Case ID	UC-A21
Super Use Case	
Primary Actor	Καθηγητής, Φοιτητής, Διαχειριστής
Secondary Actor(s)	
Brief Description	Οι χρήστες κάνουν login στο σύστημα.
Preconditions	

Flow of Events		**Actor Input**	**System Response**
	1	Ο χρήστης ανοίγει το σύστημα.	
	2		Το σύστημα εμφανίζει την φόρμα «Login»
	3	Ο χρήστης εισάγει το username και το password.	
	4		Το σύστημα ελέγχει εάν τα στοιχεία ανήκουν σε κάποιον χρήστη.
	5		Τα στοιχεία είναι σωστά. Το σύστημα ελέγχει τον τύπο του χρήστη.
	6		Ο χρήστης είναι ο διαχειριστής. Το σύστημα εμφανίζει την φόρμα του διαχειριστή.

Εναλλακτική ροή (Σενάριο):

5α

	Actor Input	System Response
1		Το σύστημα εμφανίζει το μήνυμα «Λάθος username»
2		Επιστροφή στο Βήμα «2».

5β

	Actor Input	System Response
1		Το σύστημα εμφανίζει το μήνυμα «Λάθος password».
2		Επιστροφή στο Βήμα «2».

6α

	Actor Input	System Response
1		Ο χρήστης είναι ο καθηγητής. Το σύστημα εμφανίζει την φόρμα του καθηγητή.

6β

	Actor Input	System Response
1		Ο χρήστης είναι ο φοιτητής. Το σύστημα εμφανίζει την φόρμα του φοιτητή.

Πίνακας 31: Ενέργειες Υπαλλήλων

Full – ΕΝΕΡΓΕΙΕΣ ΥΠΑΛΛΗΛΩΝ	
Use Case ID	UC-A22
Super Use Case	
Primary Actor	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων
Secondary Actor(s)	
Brief Description	Ενέργειες υπαλλήλων στο σύστημα
Preconditions	

Flow of Events		**Actor Input**	**System response**
		Το σύστημα ζητάει από τους υπαλλήλους να εκτελέσουν μια ενέργεια.	
	2		Ανάλογα με την φάση ανάπτυξης και το διάγραμμα που έχει μόλις ολοκληρωθεί, καλείται η αντίστοιχη Π.Χ «Ενέργεια Υπαλλήλου».

Πίνακας 32: Δημιουργία Project

Full – ΔΗΜΙΟΥΡΓΙΑ PROJECT	
Use Case ID	UC-A23
Super Use Case	
Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Δημιουργία project από τον διαχειριστή
Preconditions	

Flow of Events

	Actor Input	System response
1	Ο διαχειριστής ζητάει να εισάγει ένα νέο project.	
2		Το σύστημα εμφανίζει την φόρμα «Δημιουργία project».
3	Ο διαχειριστής εισάγει τα στοιχεία του project (όνομα, περιγραφή, κέρδος επιτυχίας ολοκλήρωσης έργου).	
4		Το σύστημα αποθηκεύει τα στοιχεία του project στην βάση.
5		Γίνεται include στην Π.Χ «Δημιουργία Διαγράμματος»

Εναλλακτική ροή (Σενάριο):

	Actor Input	System Response
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν.
2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
3		Επιστροφή στο Βήμα «1».

Πίνακας 33: Αξιολόγηση Διαγράμματος

Full – ΑΞΙΟΛΟΓΗΣΗ ΔΙΑΓΡΑΜΜΑΤΟΣ	
Use Case ID	UC-A24
Super Use Case	

Primary Actor			
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων		
Brief Description	Αξιολογείται το κάθε διάγραμμα που ανεβαίνει στο σύστημα		
Preconditions			
Flow of Events		**Actor Input**	**System response**
	1	Το σύστημα ζητάει να παραχθεί το διάγραμμα ανάλογα με την Π.Χ «Ενέργειες Υπαλλήλου» που την έχει καλέσει.	
	2		Το σύστημα αξιολογεί με βάση τον χρονοπρογραμματισμό του έργου και τον βαθμό του προηγούμενου διαγράμματος, το διάγραμμα που θα παραχθεί.
	3		Το σύστημα αναζητά στη βάση το κατάλληλο διάγραμμα ανάλογα με την παραπάνω αξιολόγηση.
	4		Το σύστημα επιστρέφει το διάγραμμα.

Πίνακας 34: Αξιολόγηση Διαγραμμάτων

Full – ΑΞΙΟΛΟΓΗΣΗ ΔΙΑΓΡΑΜΜΑΤΩΝ			
Use Case ID	UC-A25		
Super Use Case			
Primary Actor	Καθηγητής		
Secondary Actor(s)			
Brief Description	Ο καθηγητής αξιολογεί τα διαγράμματα που κάνει upload ο φοιτητής		
Preconditions	Να έχει πραγματοποιηθεί η Π.Χ «Λήψη διαγραμμάτων»		
Flow of Events		**Actor Input**	**System Response**
	1	Ο καθηγητής ζητάει να αξιολογήσει το διάγραμμα που έχει κάνει upload ο φοιτητής.	

	2		Το σύστημα εμφανίζει το διάγραμμα του φοιτητή σε μια φόρμα «Λήψη Διαγραμμάτων».
	3	Ο καθηγητής αξιολογεί το διάγραμμα και αυξάνει ή μειώνει κατά 0,1 στον βαθμό που έχει το διάγραμμα.	
	4		Το σύστημα καλεί την Π.Χ «Αποστολή νέου βαθμού»

Πίνακας 35: Αποστολή Νέου Βαθμού

Full – ΑΠΟΣΤΟΛΗ ΝΕΟΥ ΒΑΘΜΟΥ			
Use Case ID	UC-A26		
Super Use Case			
Primary Actor	Καθηγητής		
Secondary Actor(s)			
Brief Description	Ο καθηγητής αφού έχει αξιολογεί τα διαγράμματα που έχει κάνει upload ο φοιτητής, το σύστημα στέλνει το νέο βαθμό στην βάση.		
Preconditions	Να έχει ολοκληρωθεί η Π.Χ «Αξιολόγηση διαγραμμάτων»		
Flow of Events		**Actor Input**	**System Response**
	1	Ο καθηγητής επιλέγει ότι θέλει να αποστείλει τον νέο βαθμό ενός φοιτητή.	
	2		Το σύστημα εμφανίζει το νέο βαθμό και ζητάει από τον καθηγητή να τον επικυρώσει.
	3	Ο καθηγητής επικυρώνει τον βαθμό.	

	4		Το σύστημα αποθηκεύει στην βάση το νέο βαθμό που αξιολόγησε ο καθηγητής το τρέχων διάγραμμα.

3α		**Actor Input**	**System Response**
	1	Ο καθηγητής δεν επικυρώνει τον βαθμό.	
	2		Επιστρέφει στο Βήμα «1».

4α		**Actor Input**	**System Response**
	1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν.
	2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
	3		Επιστροφή στο Βήμα «1»

Πίνακας 36: Λήψη Διαγραμμάτων

Full – ΛΗΨΗ ΔΙΑΓΡΑΜΜΑΤΩΝ			
Use Case ID	UC-A27		
Super Use Case			
Primary Actor	Καθηγητής, Φοιτητής		
Secondary Actor(s)			
Brief Description	Ο καθηγητής κατεβάζει τα διαγράμματα που έχει κάνει upload ο φοιτητής για να τα αξιολογήσει. Ο φοιτητής κατεβάζει τα διαγράμματα για να τα δει και να προσθέσει κάποια πράγματα εάν επιθυμεί.		
Preconditions	Να έχουν σταλθεί οι αλλαγές του διαγράμματος.		
Flow of Events		**Actor Input**	**System Response**
	1	Ο χρήστης ζητάει από το σύστημα να κατεβάσει τα διαθέσιμα διαγράμματα.	
	2		Το σύστημα κατεβάζει το διάγραμμα και ρωτάει τον χρήστη εάν θέλει να το αποθηκεύσει ή να το ανοίξει από την τρέχουσα τοποθεσία.

	3	Ο χρήστης επιλέγει τον τρόπο για να ανοίξει το διάγραμμα.	
	4		Ο χρήστης επιλέγει να το ανοίξει. Το σύστημα εμφανίζει το διάγραμμα.

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Ο χρήστης επιλέγει να το αποθηκεύσει.
2	Ο χρήστης πηγαίνει και επιλέγει να ανοίξει το διάγραμμα εκεί που αποθηκεύτηκε.	
3		Το σύστημα εμφανίζει το διάγραμμα.

Πίνακας 37: Επιλογή Project

Full – ΕΠΙΛΟΓΗ PROJECT	
Use Case ID	UC-A28
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής επιλέγει ποιο project επιθυμεί να εκπονήσει ανάμεσα σε ένα πλήθος από διαθέσιμα project.
Preconditions	Να έχουν δημιουργηθεί τα project.

Flow of Events

	Actor Input	System Response
1	Ο χρήστης επιλέγει ότι θέλει να ξεκινήσει ένα νέο project.	
2		Το σύστημα δίνει στον φοιτητή ένα αρχικό προϋπολογισμό (budget).
3	Ο φοιτητής ζητάει να δεί τα διαθέσιμα project.	
4		Το σύστημα εμφανίζει την φόρμα «Διαθέσιμα project»
5	Ο φοιτητής επιλέγει ένα project.	
6		Το σύστημα εμφανίζει την φόρμα «Εμφάνιση project»
7	Ο φοιτητής επιλέγει να του εμφανίσει τα αντίστοιχα υποδείγματα.	
8		Το σύστημα εμφανίζει την φόρμα «Εμφάνιση υποδειγμάτων»
9	Ο φοιτητής επιλέγει αν θα αναλάβει το συγκεκριμένο project.	
10		Το σύστημα αποθηκεύει τις ενέργειες στην βάση.
		Τα βήματα 1-9 επαναλαμβάνονται, αν ο φοιτητής επιλέξει ότι θέλει να αναλάβει περισσότερα project.

Εναλλακτική ροή (Σενάριο):

4α

A	Actor Input	System Response
1		Το σύστημα εμφανίζει ένα μήνυμα «Δεν υπάρχουν διαθέσιμα project»

10α

A	Actor Input	System Response
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν.
		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
		Επιστροφή στο Βήμα «1»

Πίνακας 38: Κοστολόγηση Project

Full – ΚΟΣΤΟΛΟΓΗΣΗ PROJECT	
Use Case ID	UC-A29
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής κοστολογεί το project, χρησιμοποιώντας την βοήθεια των υποδειγμάτων.
Preconditions	Να έχει ολοκληρωθεί η Π.Χ «Δημιουργία project»

Flow of Events

	Actor Input	System Response
1	Ο φοιτητής πριν κοστολογήσει το project, ζητάει από το σύστημα να δει τα αντίστοιχα υποδείγματα.	
2		Το σύστημα καλεί την Π.Χ «Εμφάνιση Υποδειγμάτων»

	3	Ο φοιτητής επιλέγει ποιο από τα διαθέσιμα project θέλει να κοστολογήσει.	
	4		Το σύστημα εμφανίζει την φόρμα «Κοστολόγηση project».
	5	Ο φοιτητής ανάλογα με το κόστος του project που διάλεξε και με το budget του, εισάγει την κοστολόγηση του δικού του project.	
	6		Το σύστημα ελέγχει εάν η κοστολόγηση είναι αριθμός.
	7		Η κοστολόγηση είναι αριθμός. Το σύστημα εμφανίζει στην φόρμα την κοστολόγηση του project.
	8		Το σύστημα αποθηκεύει στη βάση την κοστολόγηση του project.

Εναλλακτική Ροή:

		Actor Input	System Response
6α	1		Η κοστολόγηση δεν είναι αριθμός. Το σύστημα εμφανίζει το μήνυμα «Η κοστολόγηση δεν είναι αριθμός. »
	2		Επιστρέφει στο Βήμα «4»
		Actor Input	**System Response**
8α	1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν.
	2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».

3		Επιστροφή στο Βήμα «1»

Πίνακας 39: Εμφάνιση Υποδειγμάτων

Full – ΕΜΦΑΝΙΣΗ ΥΠΟΔΕΙΓΜΑΤΩΝ			
Use Case ID	UC-A30		
Super Use Case			
Primary Actor			
Secondary Actor(s)			
Brief Description	Το σύστημα εμφανίζει τα υποδείγματα.		
Preconditions	Να έχει ολοκληρωθεί η Π.Χ «Δημιουργία υποδειγμάτων».		
Flow of Events		**Actor Input**	**System Response**
	1	Ο χρήστης επιλέγει να δει τα υποδείγματα για ένα project.	
	2		Το σύστημα αναζητά τα υποδείγματα που ταιριάζουν στο project του φοιτητή.
	3	.	Το σύστημα εμφανίζει την φόρμα «Εμφάνιση υποδειγμάτων» για το αντίστοιχο project, η οποία περιλαμβάνει όλα τα σχετικά υποδείγματα.

Πίνακας 40: Δημιουργία Χρονοπρογράμματος

Full – ΔΗΜΙΟΥΡΓΙΑ ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΟΣ

Use Case ID	UC-A31
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής δημιουργεί τον χρονοπρογραμματισμό του έργου.
Preconditions	

Flow of Events

	Actor Input	System Response
1	Ο φοιτητής ζητά να χρονοπρογραμματίσει το project που έχει αναλύσει.	
2		Το σύστημα εμφανίζει τη φόρμα «Χρονοπρογραμματισμός» της πρώτης φάσης ανάπτυξης.
3		Το σύστημα καλεί την Π.Χ «Καθορισμός χρονικού διαστήματος ανά φάση».
4		Το σύστημα καλεί την Π.Χ «Ορισμός Ανθρωποωρών».
5		Το σύστημα καλεί την Π.Χ «Αντιστοίχηση υπαλλήλου με θέση ανά φάση».
6		Τα βήματα 2-5 επαναλαμβάνονται για κάθε φάση ανάπτυξης.

Πίνακας 41: Ορισμός Ανθρωποωρών ανά Φάση

Full – ΟΡΙΣΜΟΣ ΑΝΘΡΩΠΟΩΡΩΝ ΑΝΑ ΦΑΣΗ	
Use Case ID	UC-A32
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής ορίζει τις ανθρωποώρες για κάθε φάση.

Preconditions

Flow of Events

	Actor Input	System Response
1	Ο φοιτητής θέλει να εισάγει τις ανθρωποώρες για την τρέχουσα φάση.	
2		Το σύστημα εμφανίζει την φόρμα «Ορισμός Ανθρωποωρών».
3	Ο φοιτητής εισάγει τον αριθμό ωρών που θα απασχοληθούν οι υπάλληλοι για την τρέχουσα φάση.	
4		Το σύστημα ελέγχει εάν οι ώρες είναι αριθμοί.
5		Οι ώρες είναι αριθμοί. Το σύστημα εμφανίζει στην φόρμα τις ώρες που θα απασχοληθούν συνολικά οι υπάλληλοι για κάθε φάση.
6	Ο φοιτητής επιβεβαιώνει τα στοιχεία στην φόρμα.	
7		Το σύστημα αποθηκεύει στη βάση τα στοιχεία του ορισμού ανθρωποωρών.

Εναλλακτική ροή (Σενάριο):

5α

	Actor Input	System Response
1		Το σύστημα εμφανίζει το μήνυμα «Οι ώρες δεν είναι αριθμός»

	2	Ο φοιτητής επιστρέφει στο Βήμα «2»	

7α

	Actor Input	System Response
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν στην βάση.
2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
2		Επιστρέφει στο Βήμα «1».

Πίνακας 42: Αντιστοίχιση Υπαλλήλου ανά Φάση

Full – ΑΝΤΙΣΤΟΙΧΙΣΗ ΥΠΑΛΛΗΛΟΥ ΜΕ ΘΕΣΗ ΑΝΑ ΦΑΣΗ	
Use Case ID	UC-A33
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής ορίζει τους υπαλλήλους σε ποια θέση θα απασχοληθεί
Preconditions	

Flow of Events

	Actor Input	System Response
1	Ο φοιτητής επιθυμεί να εισάγει υπαλλήλους σε κάθε φάση.	
2		Το σύστημα εμφανίζει την φόρμα «Αντιστοίχιση υπαλλήλου με θέση ανά φάση» η οποία παρουσιάζει τους διαθέσιμους υπαλλήλους.
3	Ο φοιτητής επιλέγει ποιοι υπάλληλοι θα δουλέψουν σε ποια φάση και πόσες ώρες.	
4		Το σύστημα αποθηκεύει στη βάση τα στοιχεία.

5		Τα βήματα 2-4 επαναλαμβάνονται μέχρι ο φοιτητής να εξαντλήσει τους υπαλλήλους του ή να επιλέξει ότι δεν θέλει να χρησιμοποιήσει άλλους.

Εναλλακτική ροή (Σενάριο):

4α

	Actor Input	System Response
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν στην βάση.
2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
3		Επιστρέφει στο Βήμα «1».

Πίνακας 43: Καθορισμός Χρονικού Διαστήματος ανά Φάση

Full – ΚΑΘΟΡΙΣΜΟΣ ΧΡΟΝΙΚΟΥ ΔΙΑΣΤΗΜΑΤΟΣ ΑΝΑ ΦΑΣΗ	
Use Case ID	UC-A34
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής καθορίζει το χρονικό διάστημα που θα διαρκέσει η κάθε φάση
Preconditions	

Flow of Events

	Actor Input	System Response
1	Ο φοιτητής ζητάει να εισάγει τον χρονοπρογραμματισμό.	
2		Το σύστημα εμφανίζει την φόρμα. «Καθορισμός χρονικού διαστήματος ανά φάση»

3	Ο φοιτητής εισάγει την ημερομηνία έναρξης της τρέχουσας φάσης.	
4		Το σύστημα ελέγχει εάν η ημερομηνία είναι έγκυρη.
5		Η ημερομηνία είναι έγκυρη. Το σύστημα εμφανίζει στην φόρμα την ημερομηνία.
6	Ο φοιτητής εισάγει την ημερομηνία λήξης της κάθε φάσης.	
7		Το σύστημα ελέγχει εάν η ημερομηνία είναι έγκυρη.
8	.	Η ημερομηνία είναι έγκυρη. Το σύστημα εμφανίζει στην φόρμα την ημερομηνία λήξης.

Εναλλακτική ροή (Σενάριο):

5α

A	**Actor Input**	**System Response**
1		Το σύστημα εμφανίζει το μήνυμα «Η ημερομηνία δεν είναι έγκυρη».
2	Ο φοιτητής επιστρέφει στο Βήμα «3»	

8α

B	**Actor Input**	**System Response**
1		Το σύστημα εμφανίζει το μήνυμα «Η ημερομηνία δεν είναι έγκυρη».
2	Ο φοιτητής επιστρέφει στο Βήμα «3»	

Πίνακας 44: Καθορισμός Αξιολόγηση Χρονοπρογραμματισμού

Full – ΑΞΙΟΛΟΓΗΣΗ ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Use Case ID	UC-A35
Super Use Case	
Primary Actor	
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων
Brief Description	Το σύστημα αξιολογεί τις ενέργειες του φοιτητή στη Π.Χ «Δημιουργία Χρονοπρογράμματος»
Preconditions	

Flow of Events

	Actor Input	System Response
1	Το σύστημα πρέπει να αξιολογήσει τον χρονοπρογραμματισμό του φοιτητή για το πρώτο project.	
2		Το σύστημα με βάση την έξοδο της Π.Χ «Δημιουργία Χρονοπρογράμματος» υπολογίζει για κάθε φάση την μεταβλητή αξιολόγηση.
3		Το σύστημα αποθηκεύει τα στοιχεία στην βάση.

Εναλλακτική ροή (Σενάριο):

3α

	Actor Input	System Response
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν στην βάση.
2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
3		Επιστρέφει στο βήμα «1»

Πίνακας 45: Επιλογή Υπαλλήλων

Full – ΕΠΙΛΟΓΗ ΥΠΑΛΛΗΛΩΝ	
Use Case ID	UC-A36
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής επιλέγει υπαλλήλους για κάθε φάση
Preconditions	

Flow of Events		**Actor Input**	**System Response**
	1	Ο φοιτητής να επιλέξει φοιτητές.	
	2		Το σύστημα εμφανίζει την φόρμα «Υπάλληλοι»
	3		Γίνεται include στην Π.Χ «Εμφάνιση Χαρακτηριστικών Υπαλλήλων»
	4	Ο φοιτητής επιλέγει τους υπαλλήλους που θέλει να προσλάβει.	
	5		Το σύστημα αποθηκεύει τους υπαλλήλους στην βάση.

Εναλλακτική ροή (Σενάριο):

		Actor Input	**System Response**
	1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν στην βάση.
	2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
	3		Επιστρέφει στο βήμα «1»

Πίνακας 46: Εμφάνιση Χαρακτηριστικών Υπαλλήλων

Full – ΕΜΦΑΝΙΣΗ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΥΠΑΛΛΗΛΩΝ	
Use Case ID	UC-A37
Super Use Case	
Primary Actor	
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων

Brief Description	Το σύστημα εμφανίζει τα χαρακτηριστικά των υπαλλήλων		
Preconditions			
Flow of Events		**Actor Input**	**System Response**
	1	Το σύστημα ζητάει να εμφανίσει τα χαρακτηριστικά των υπαλλήλων.	
	2		Το σύστημα εμφανίζει στην φόρμα «Υπάλληλοι» τον βαθμό ικανότητας για κάθε υπάλληλο και για κάθε φάση.
	3		Το σύστημα εμφανίζει στη φόρμα τον τύπο προσωπικότητας του κάθε υπαλλήλου.

Πίνακας 47: Αποτυχία Project

Full – ΑΠΟΤΥΧΙΑ PROJECT	
Use Case ID	UC-A38
Super Use Case	
Primary Actor	
Secondary Actor(s)	Σύστημα Παραγωγής και Αξιολόγησης Διαγραμμάτων
Brief Description	Το project αποτυγχάνει στην περίπτωση Αξιολόγηση διαγράμματος καθώς και στην περίπτωση Αξιολόγηση χρονοπρογραμματισμού.
Preconditions	

Flow of Events

	Actor Input	System Response
1	Το σύστημα εντοπίζει την αποτυχία ενός project.	
2		Το σύστημα ακυρώνει το project από τη βάση.
3		Το σύστημα ενημερώνει τον φοιτητή για τον λόγο αποτυχίας.

Πίνακας 48: Αλλαγή Διαγραμμάτων

Full – ΑΛΛΑΓΗ ΔΙΑΓΡΑΜΜΑΤΩΝ	
Use Case ID	UC-A39
Super Use Case	
Primary Actor	Φοιτητής
Secondary Actor(s)	
Brief Description	Ο φοιτητής βλέπει το διάγραμμα που του έχει σταλεί και έχει την δυνατότητα να κάνει αλλαγές .
Preconditions	

	Actor Input	System Response
1	Ο φοιτητής ζητάει να δει το διάγραμμα που έχουν ετοιμάσει οι υπάλληλοι για την τρέχουσα φάση.	
2		Το σύστημα εμφανίζει το τρέχων διάγραμμα.

3	Ο φοιτητής επιθεωρεί το διάγραμμα και επιλέγει να κάνει αλλαγές στο διάγραμμα.	
4		Το σύστημα εμφανίζει την φόρμα ανεβάσματος νέου διαγράμματος.
5	Ο φοιτητής ανεβάζει το αναθωρημένο διάγραμμα.	
6		Το σύστημα αποθηκεύει το διάγραμμα και ενημερώνει τον καθηγητή.

Εναλλακτική ροή (Σενάριο):

6α	**Actor Input**	**System Response**
1		Τα στοιχεία δεν είναι δυνατόν να αποθηκευτούν στην βάση.
2		Εμφάνιση μηνύματος «Τα στοιχεία δεν μπορούν να αποθηκευτούν».
3		Επιστρέφει στο βήμα «1»

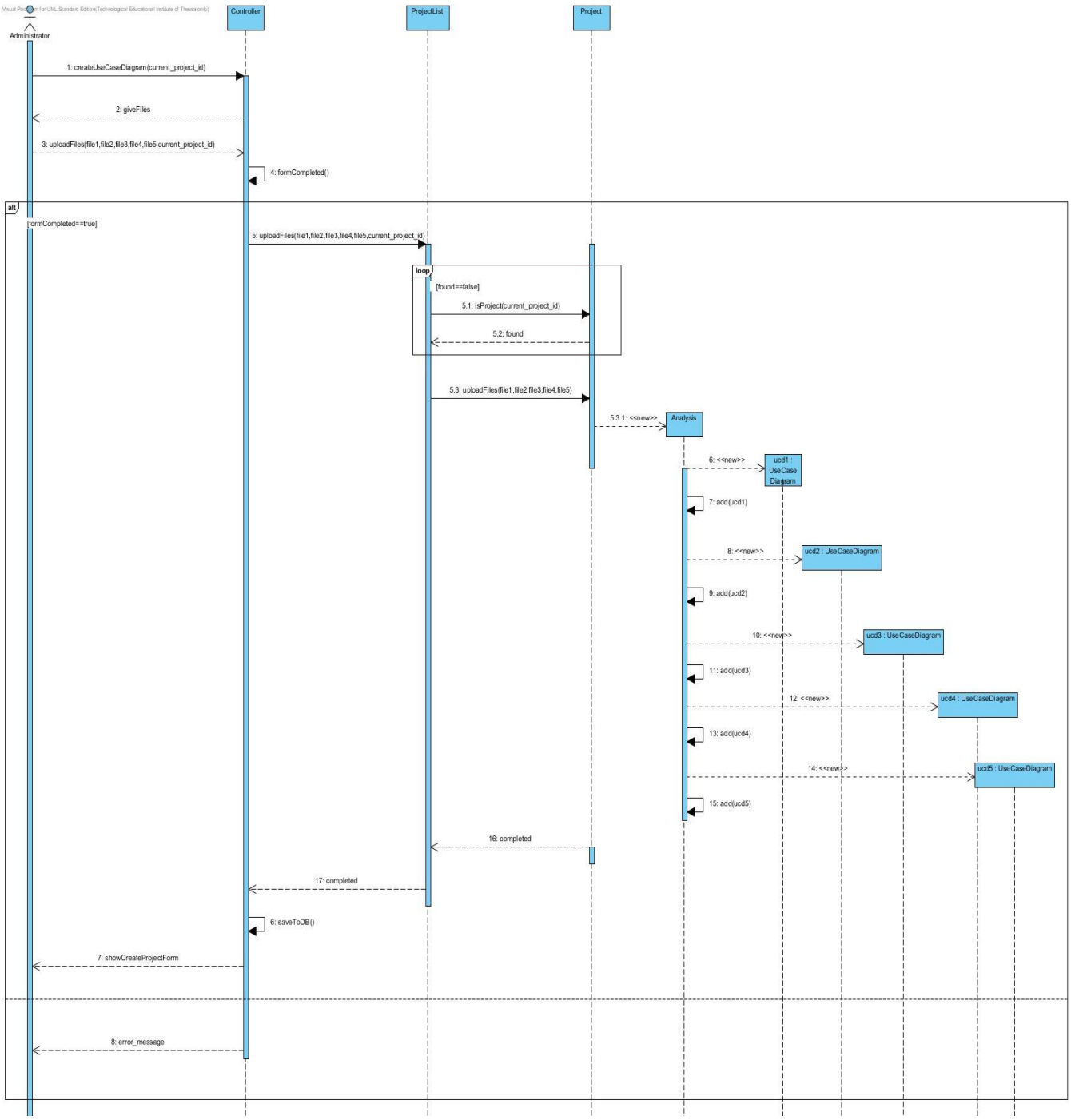
Πίνακας 49: Δημιουργία Διαγράμματος

Full – ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΓΡΑΜΜΑΤΟΣ	
Use Case ID	UC-A40
Super Use Case	
Primary Actor	Διαχειριστής
Secondary Actor(s)	
Brief Description	Ο διαχειριστής επιλέγει ένα διάγραμμα
Preconditions	Να έχει κληθεί η Π.Χ «Δημιουργία project»

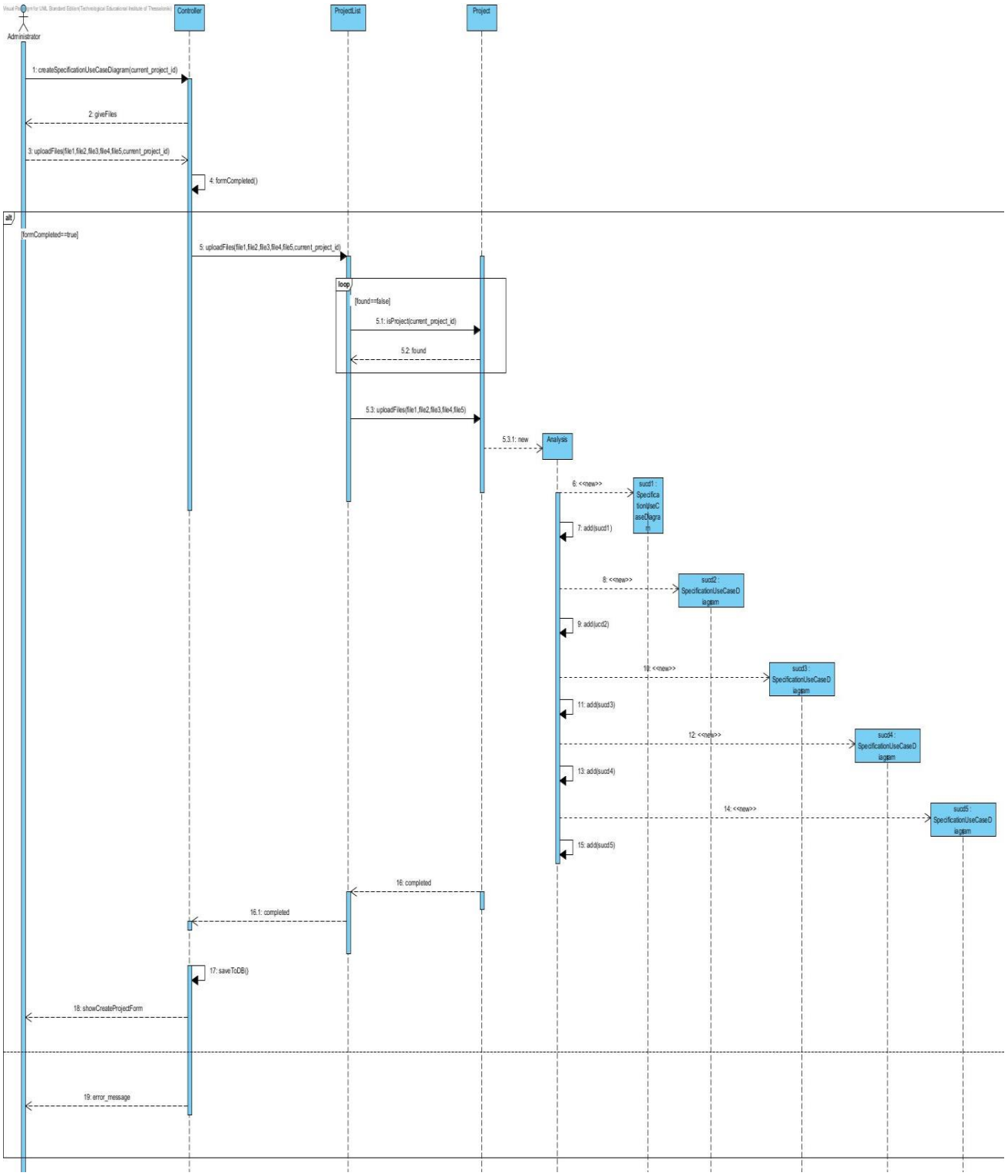
Flow of Events		**Actor Input**	**System Response**
	1	Ο διαχειριστής επιλέγει να δημιουργήσει ένα διάγραμμα.	
	2		Το σύστημα ζητάει τον τύπο του διαγράμματος που θα αποθηκευτεί.
	3	Ο διαχειριστής επιλέγει τον τύπο του διαγράμματος.	
	4		Το σύστημα καλεί την αντίστοιχη Π.Χ για το διάγραμμα που θα δημιουργηθεί.

4.4 Διαγράμματα Ακολουθίας

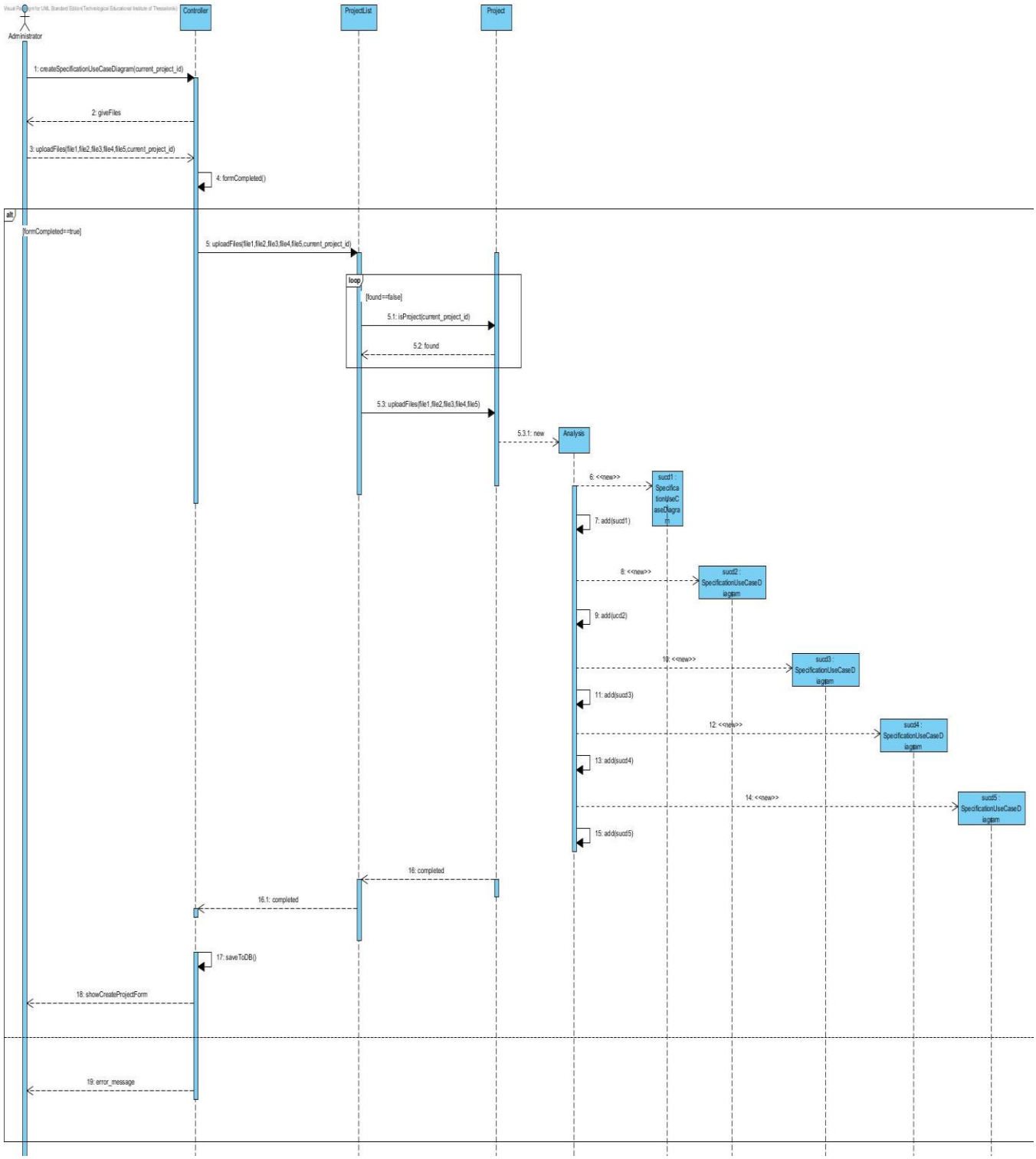
Σχήμα 23: Δημιουργία Διαγράμματος Π.Χ



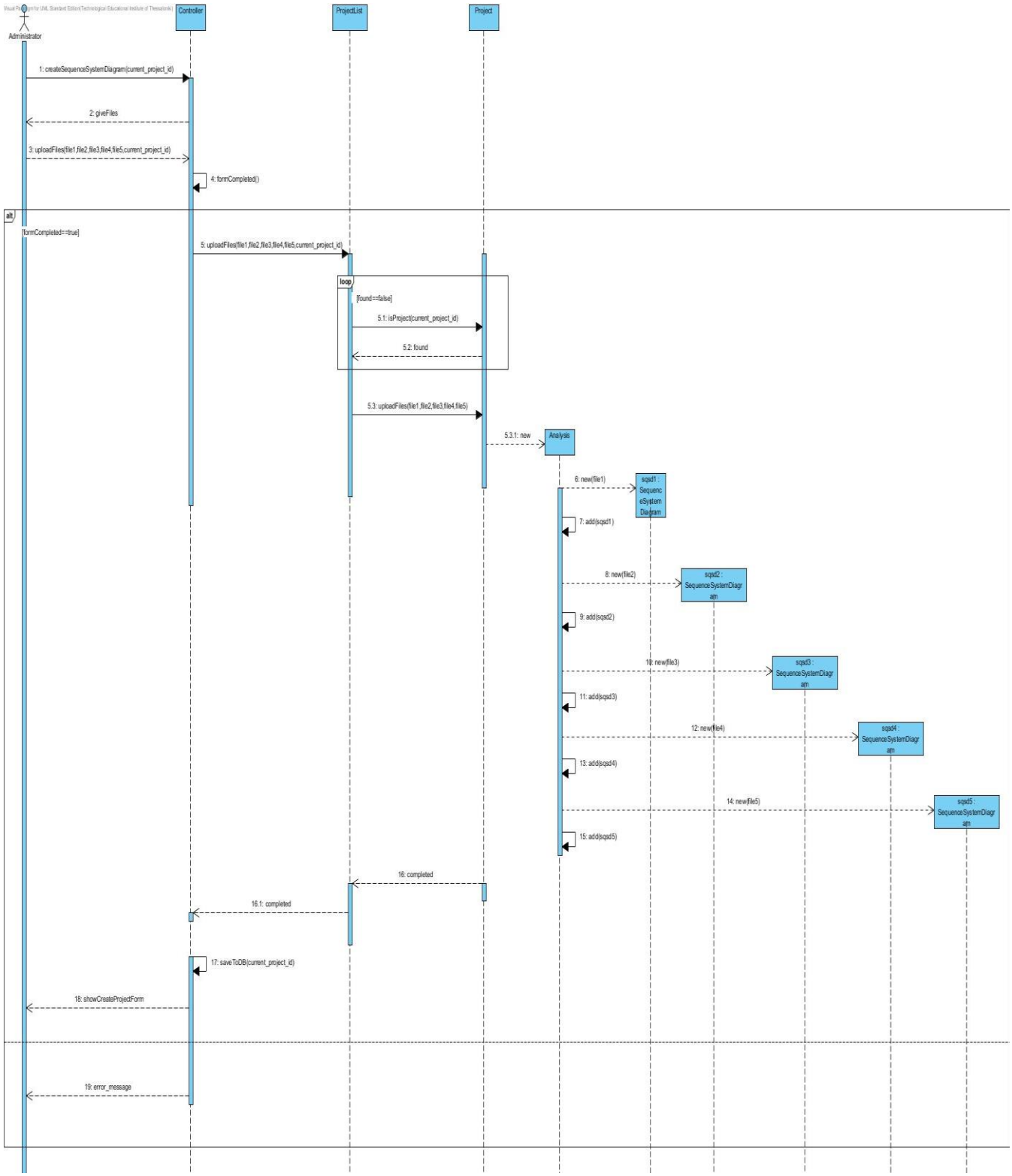
Σχήμα 24: Δημιουργία Προδιαγραφών Π.Χ



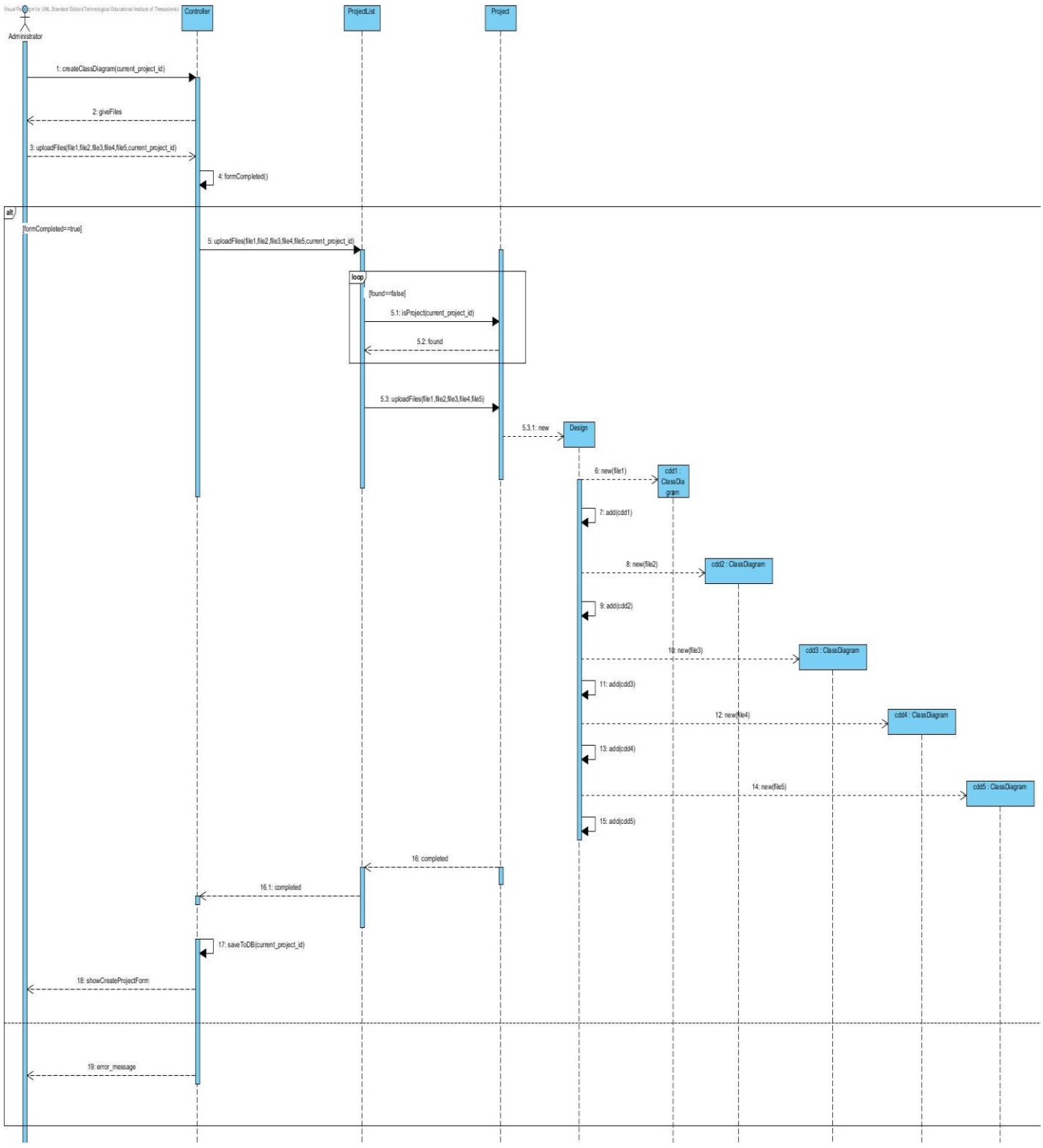
Σχήμα 25:Δημιουργία Διαγράμματος Ακολουθίας



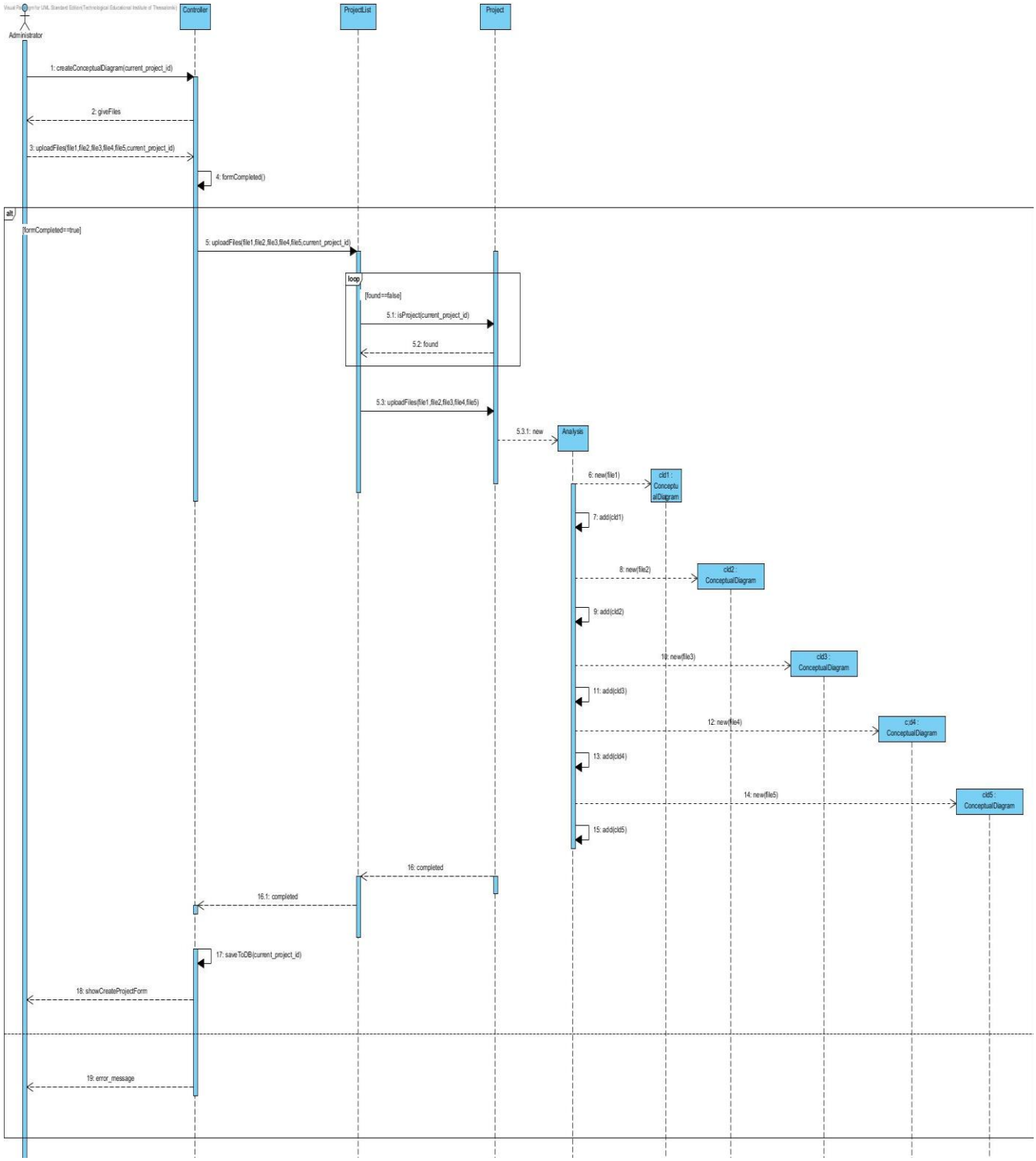
Σχήμα 26:Δημιουργία Διαγράμματος Ακολουθίας Συστήματος



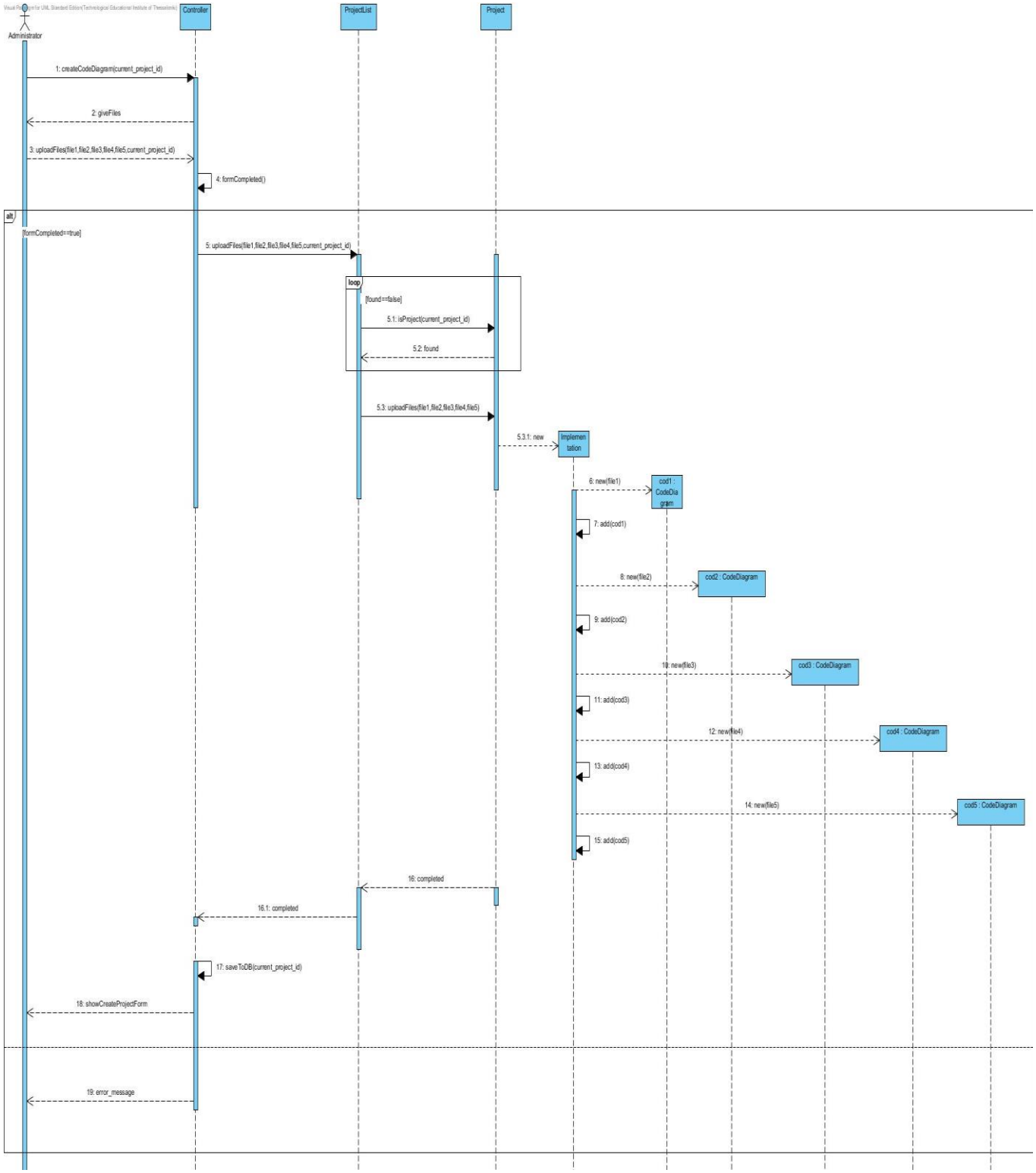
Σχήμα 27: Δημιουργία Διαγράμματος Κλάσεων



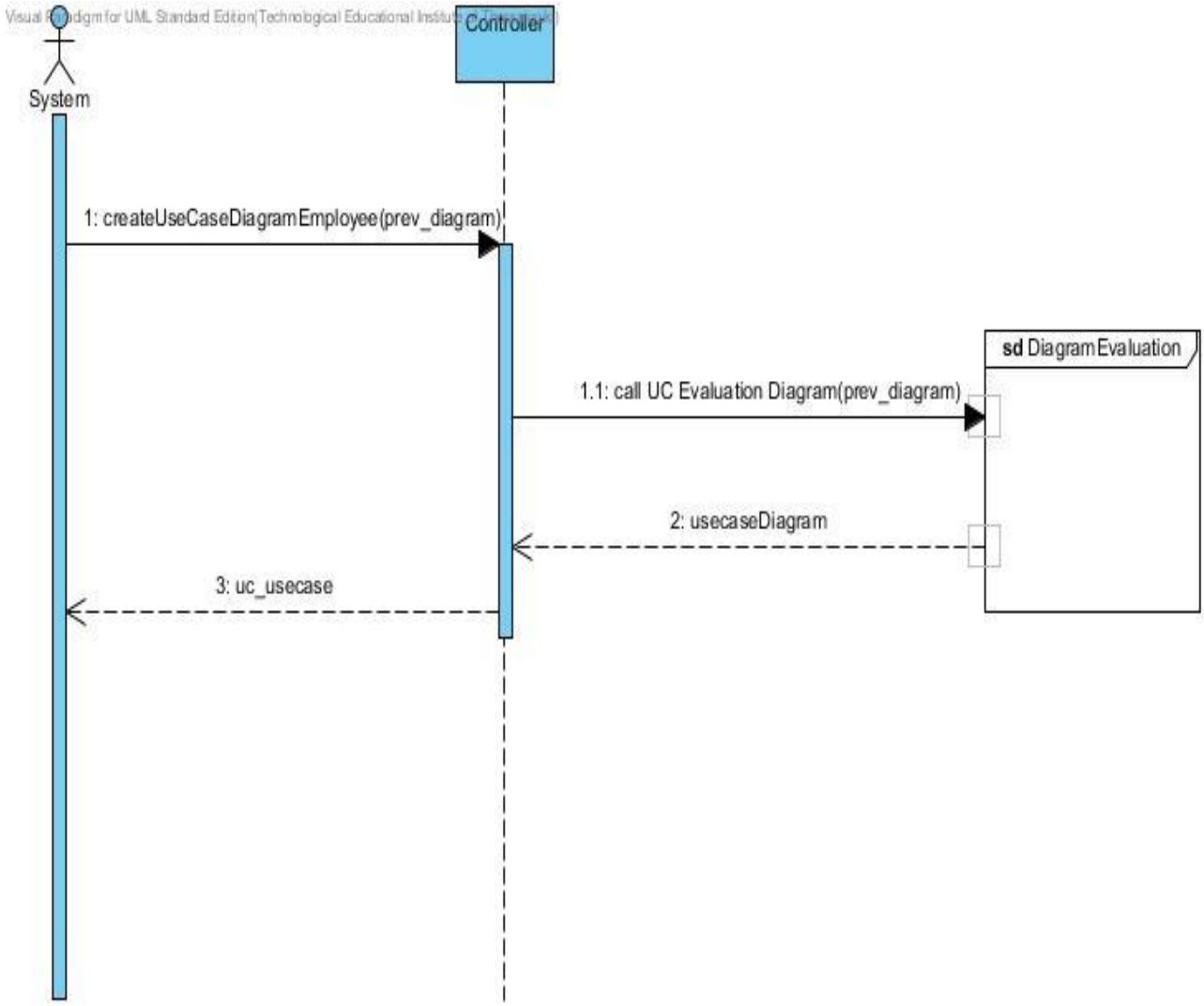
Σχήμα 28: Δημιουργία Διαγράμματος Εννοιολογικού Μοντέλου



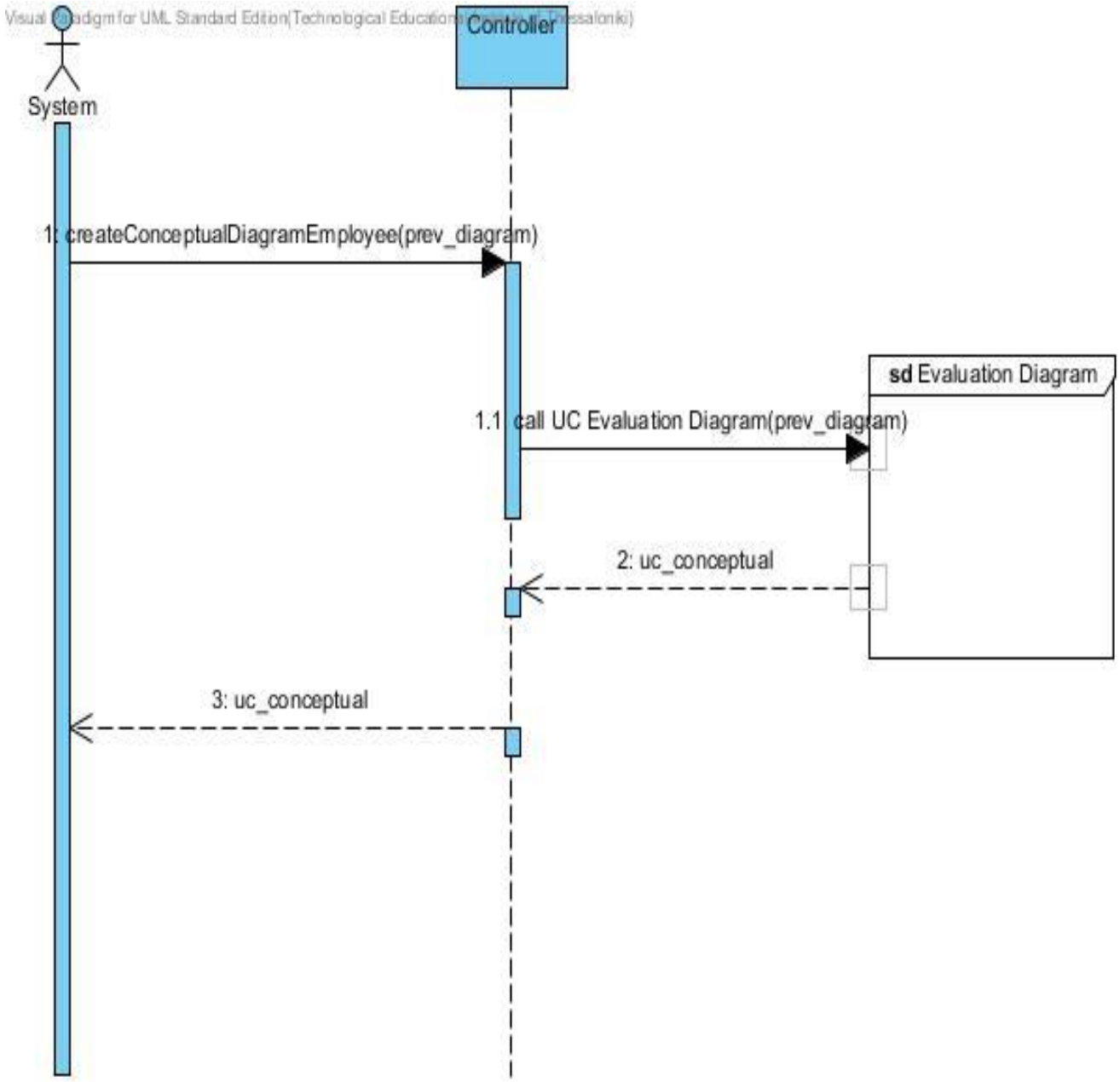
Σχήμα 29: Δημιουργία Ανέβασμα Κώδικα



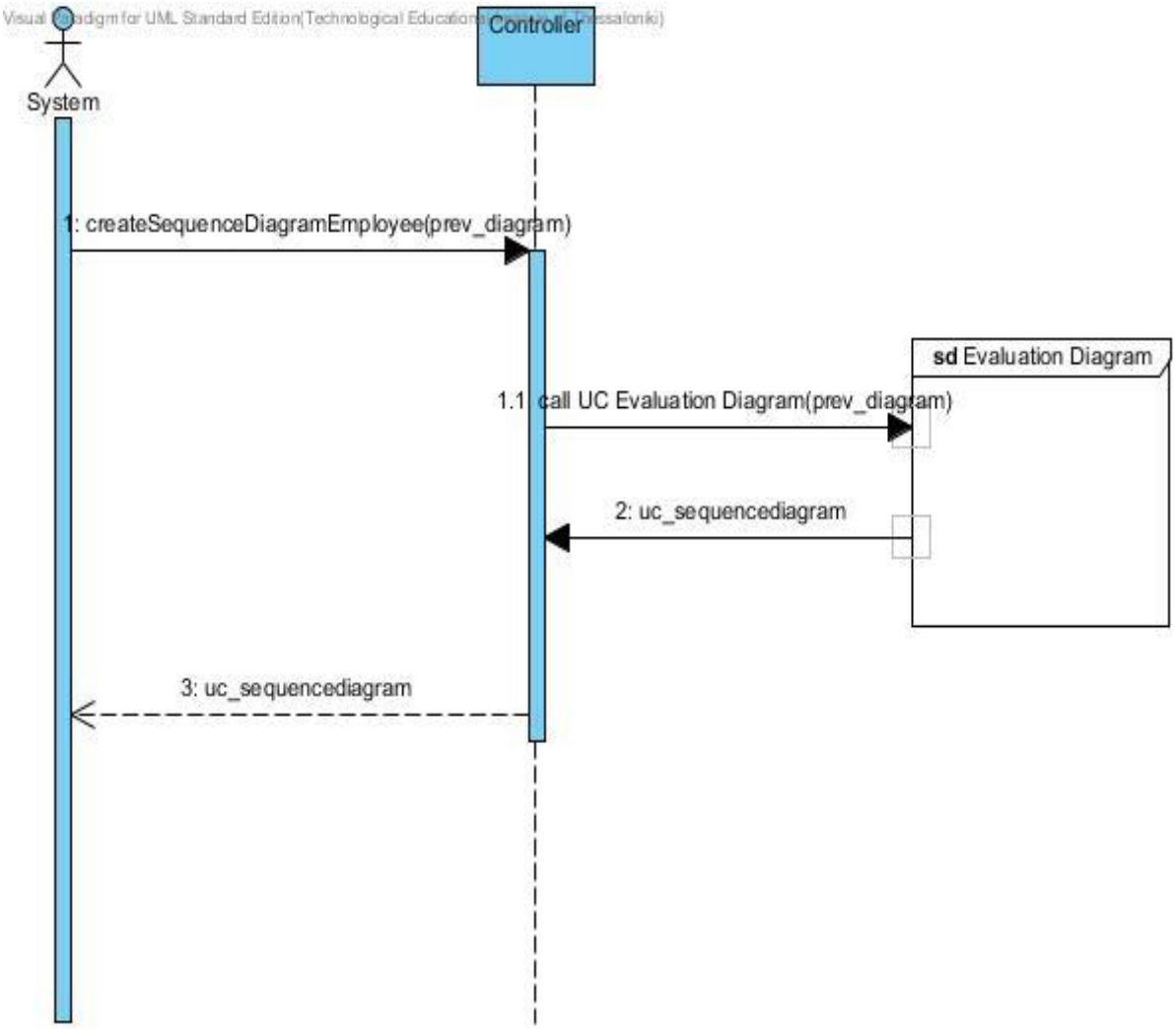
Σχήμα 30: Δημιουργία Διαγράμματος Π.Χ Από Τους Υπαλλήλους



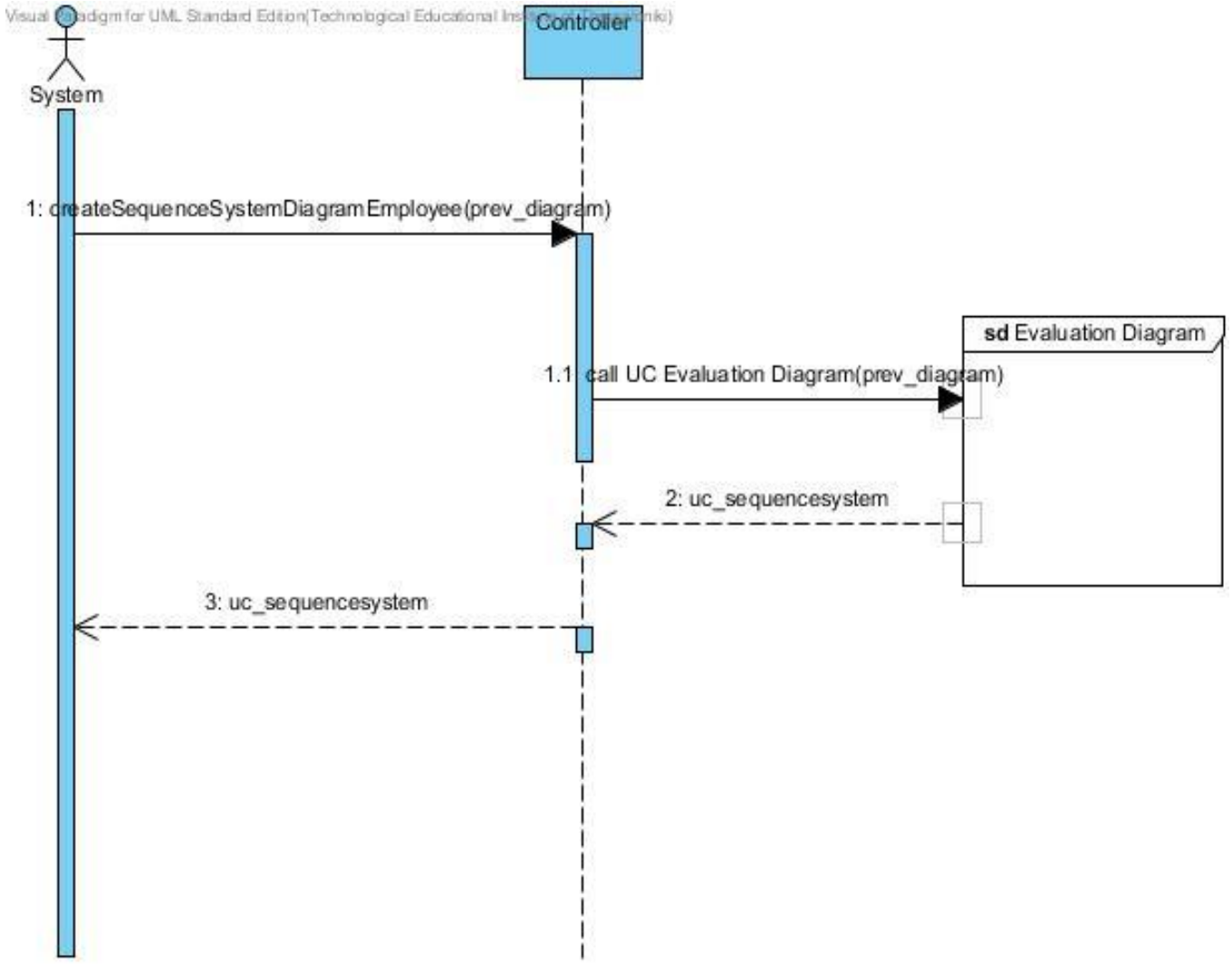
Σχήμα 31: Δημιουργία Προδιαγραφών Π.Χ Από Τους Υπαλλήλους



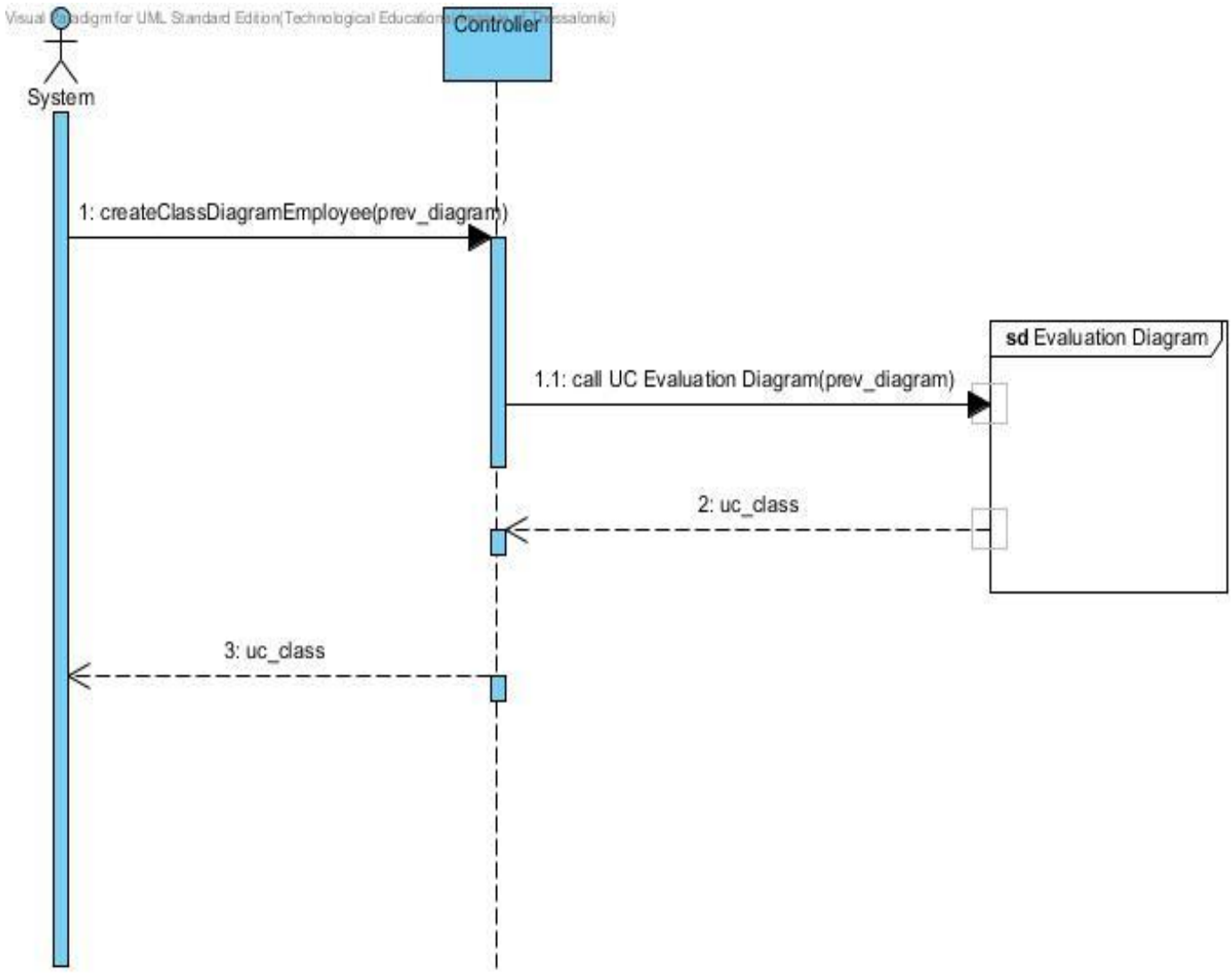
Σχήμα 32: Δημιουργία Διαγράμματος Ακολουθίας Από Τους Υπαλλήλους



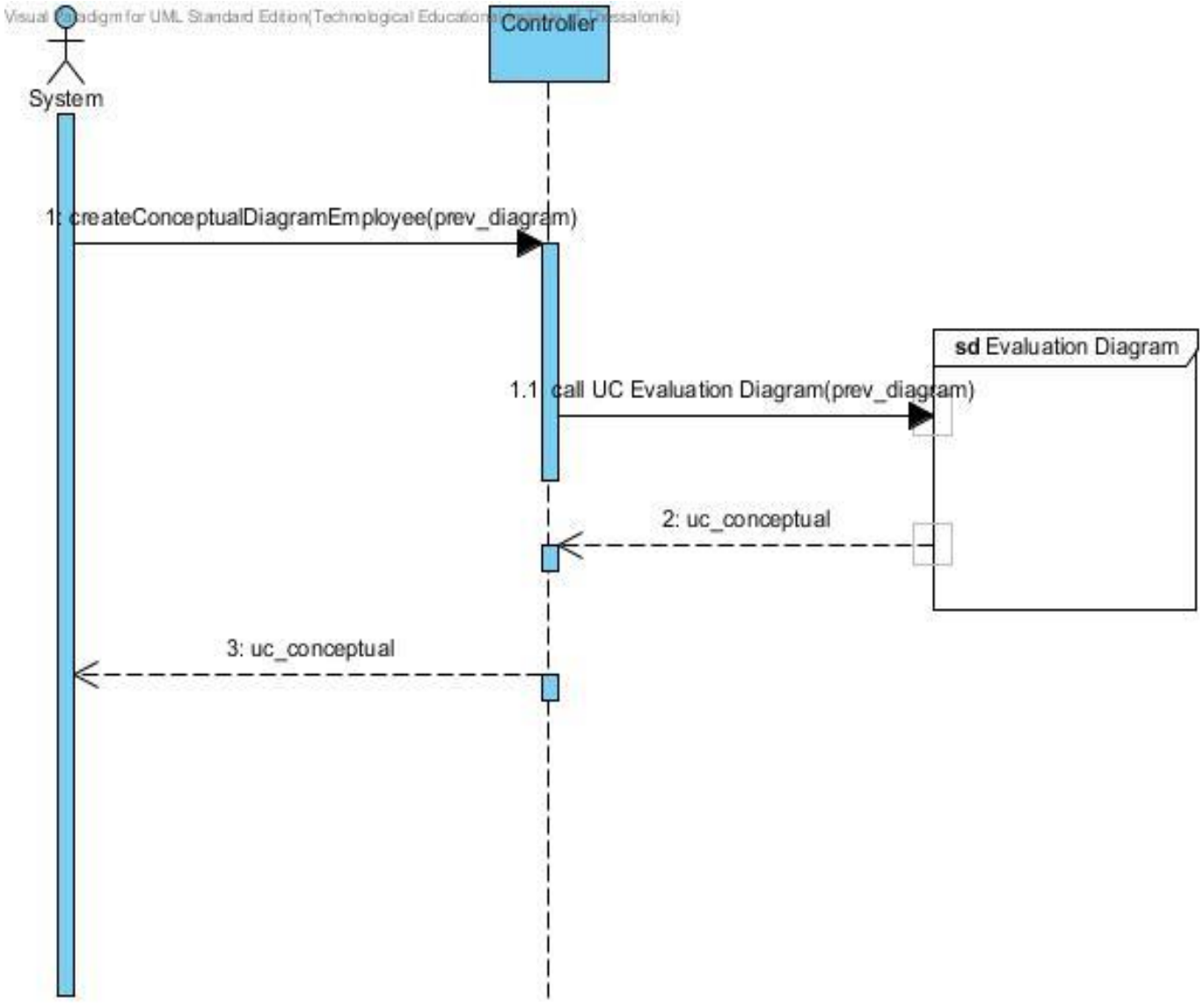
Σχήμα 33: Δημιουργία Διαγράμματος Ακολουθίας Συστήματος Από Τους Υπαλλήλους



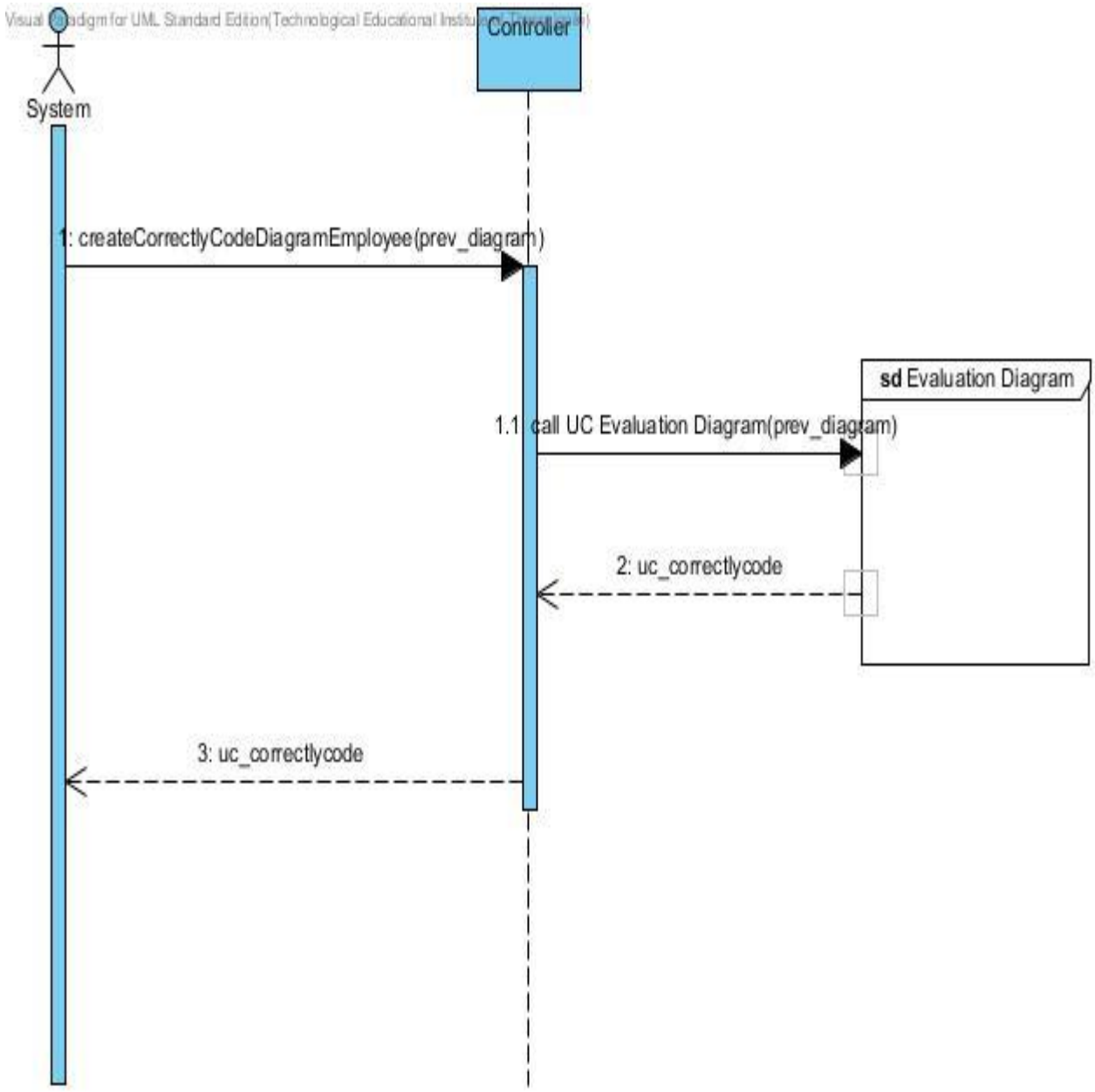
Σχήμα 34: Δημιουργία Διαγράμματος Κλάσεων Από Τους Υπαλλήλους



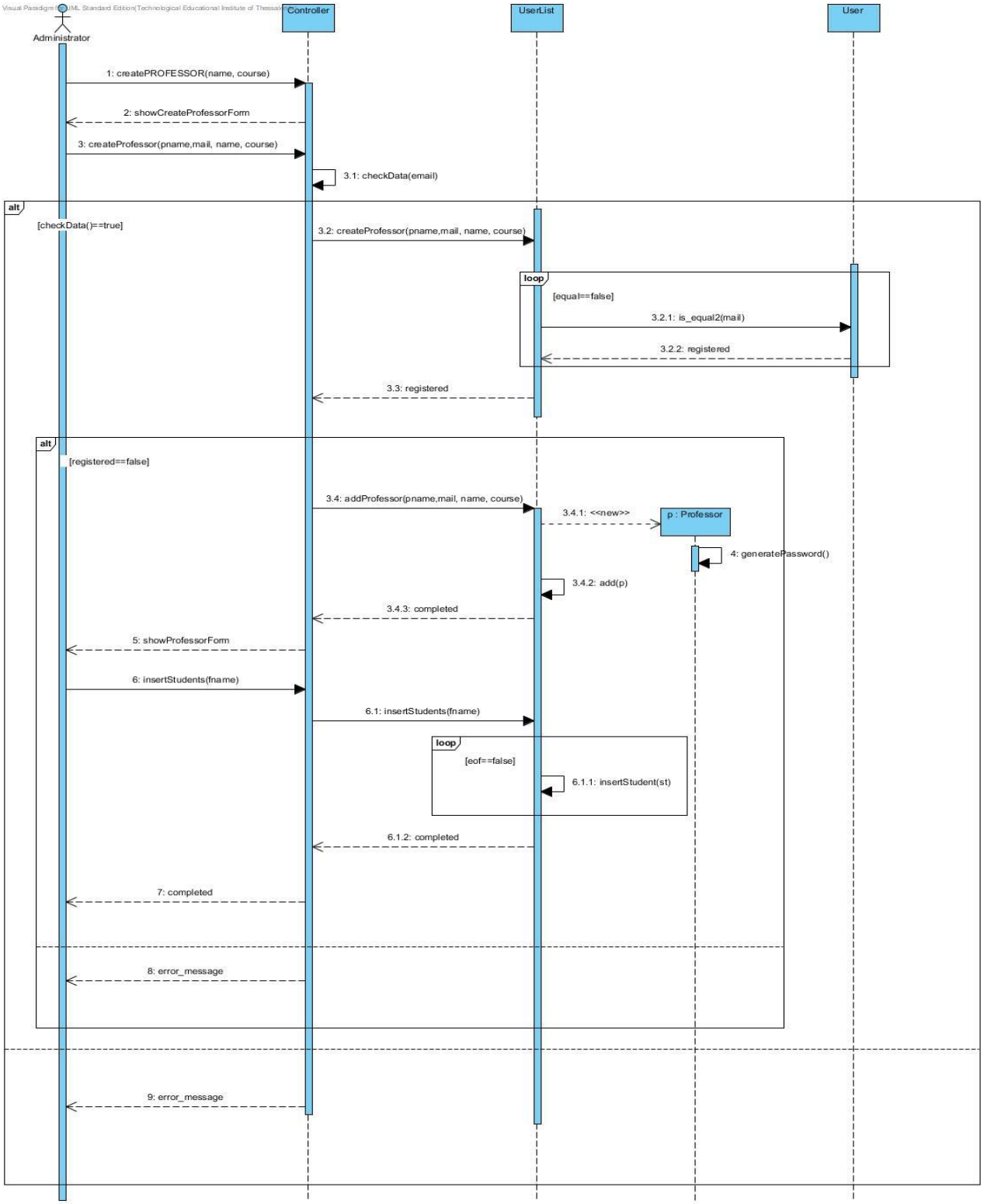
Σχήμα 35: Δημιουργία Διαγράμματος Εννοιολογικού Μοντέλου Από Τους Υπαλλήλους



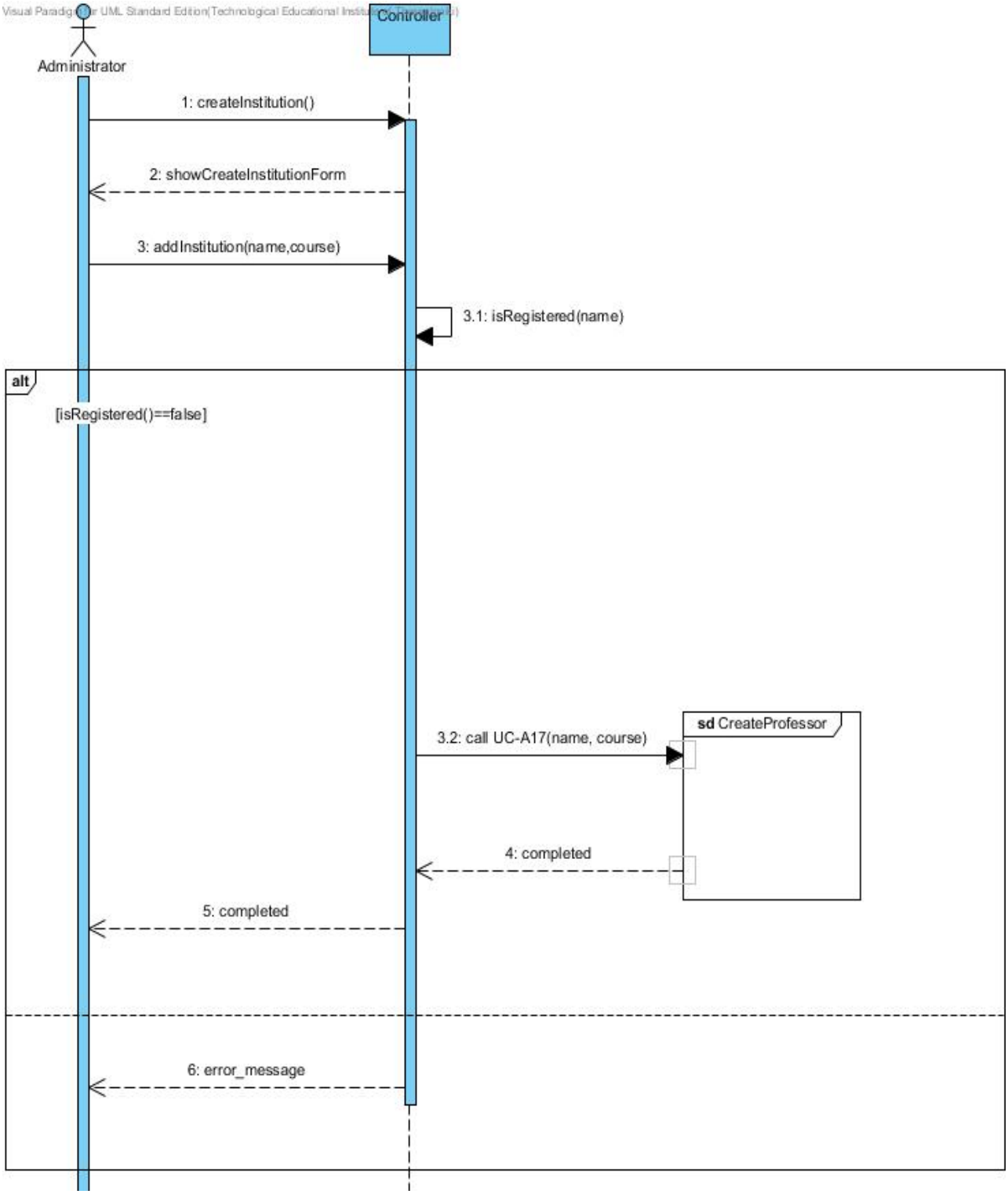
Σχήμα 36: Δημιουργία Διαγράμματος Ανέβασμα Κώδικα Από Τους Υπαλλήλους



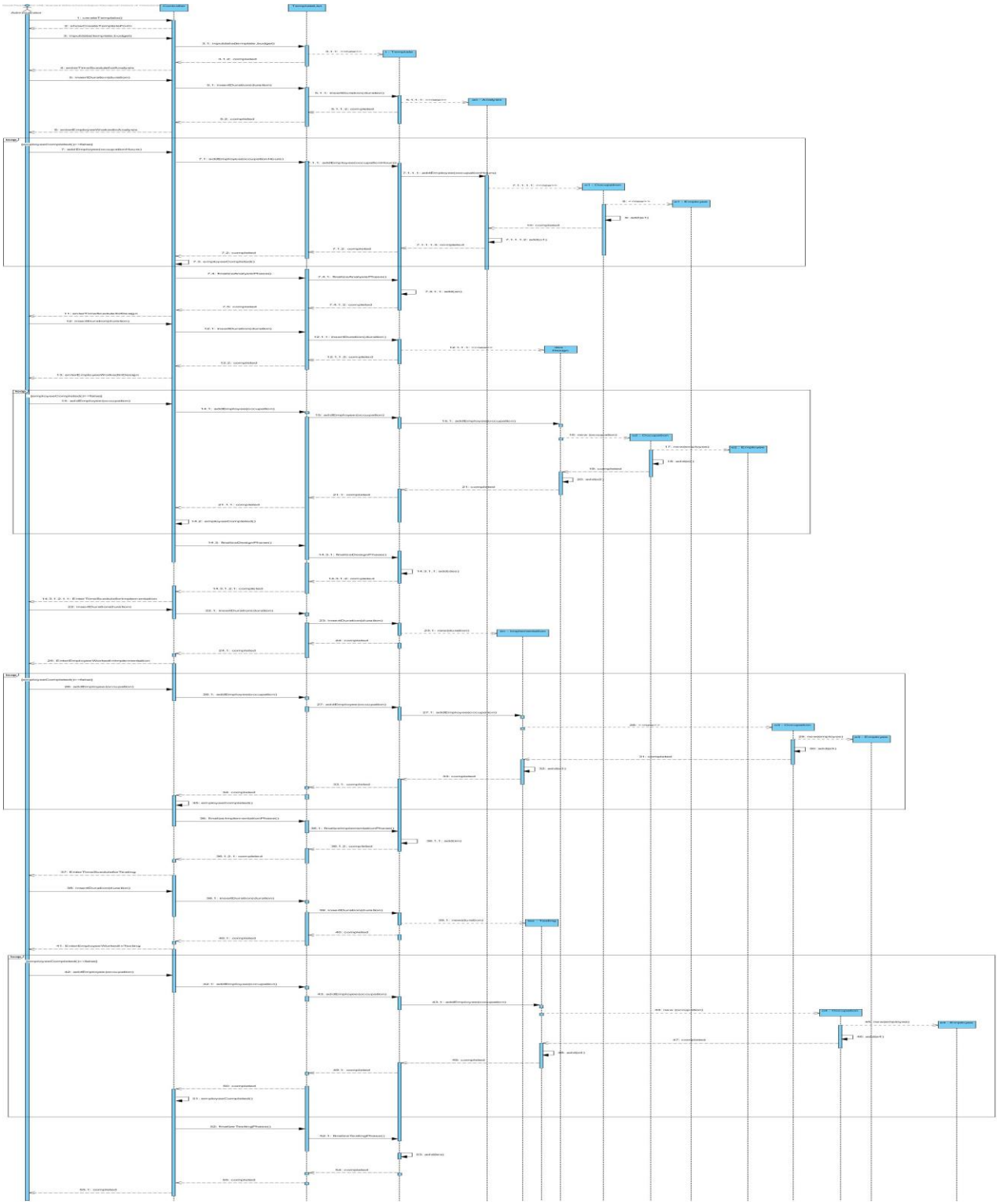
Σχήμα 37: Δημιουργία Καθηγητή



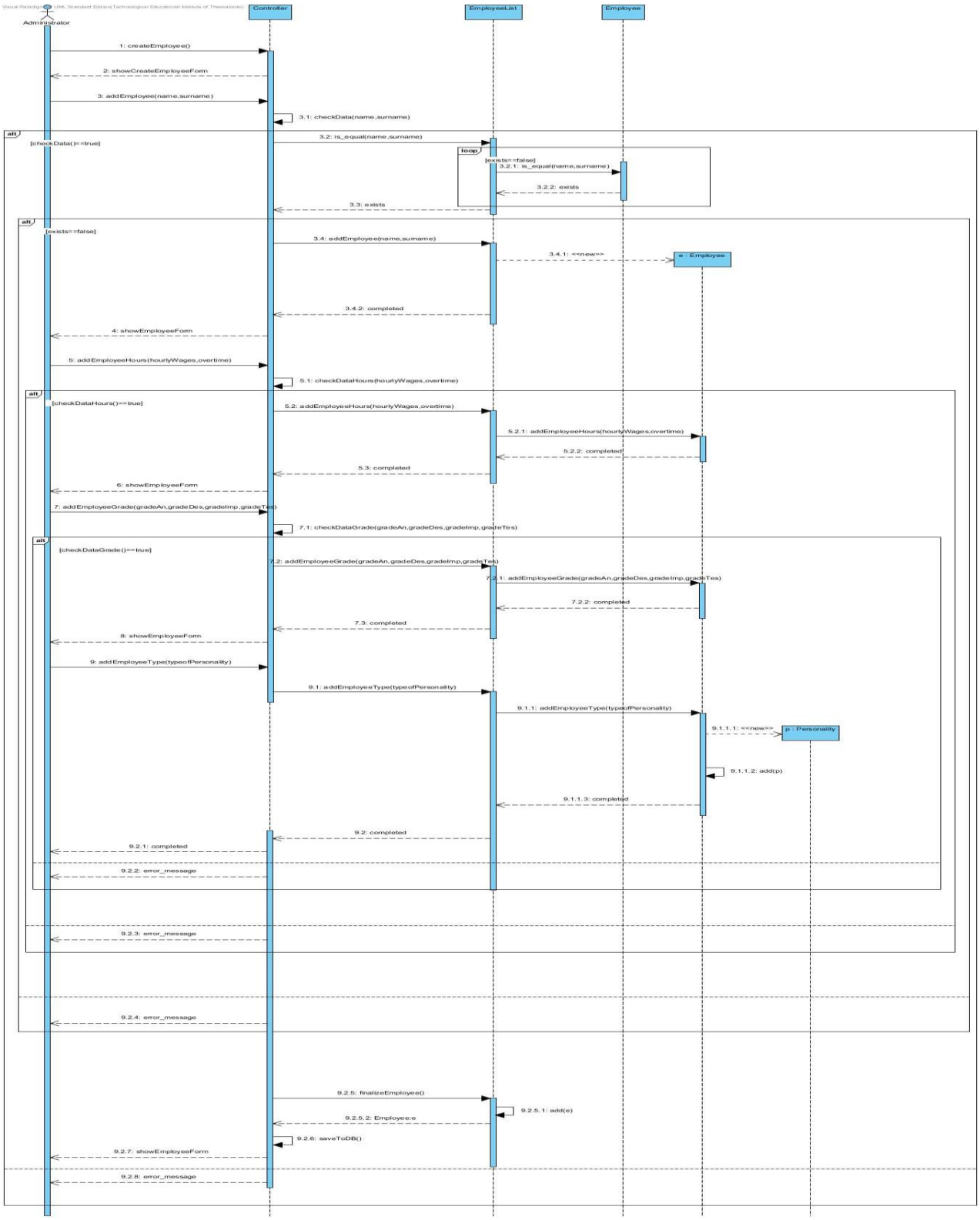
Σχήμα 38: Δημιουργία Ιδρύματος



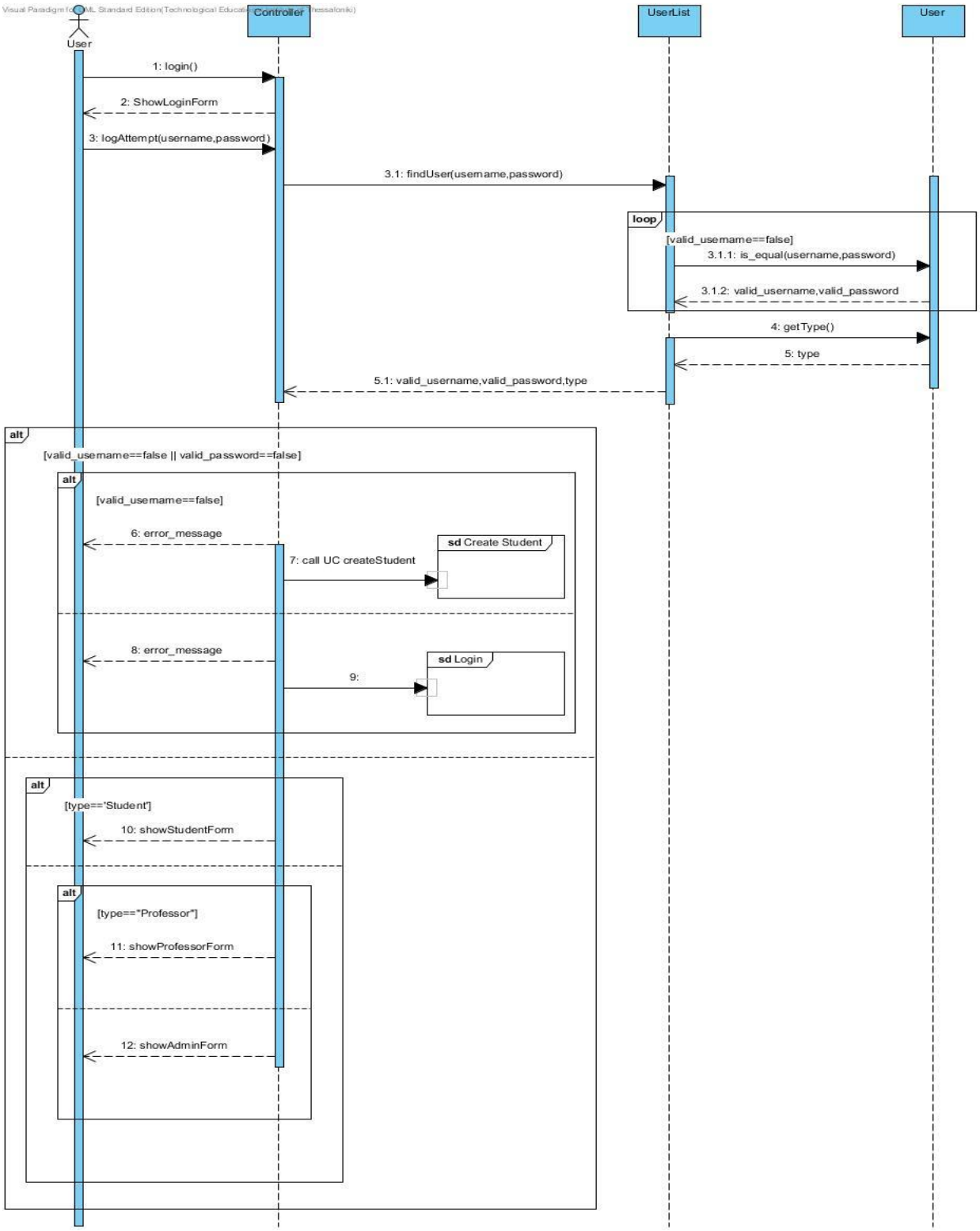
Σχήμα 38: ΔημιουργίαΥποδείγματος



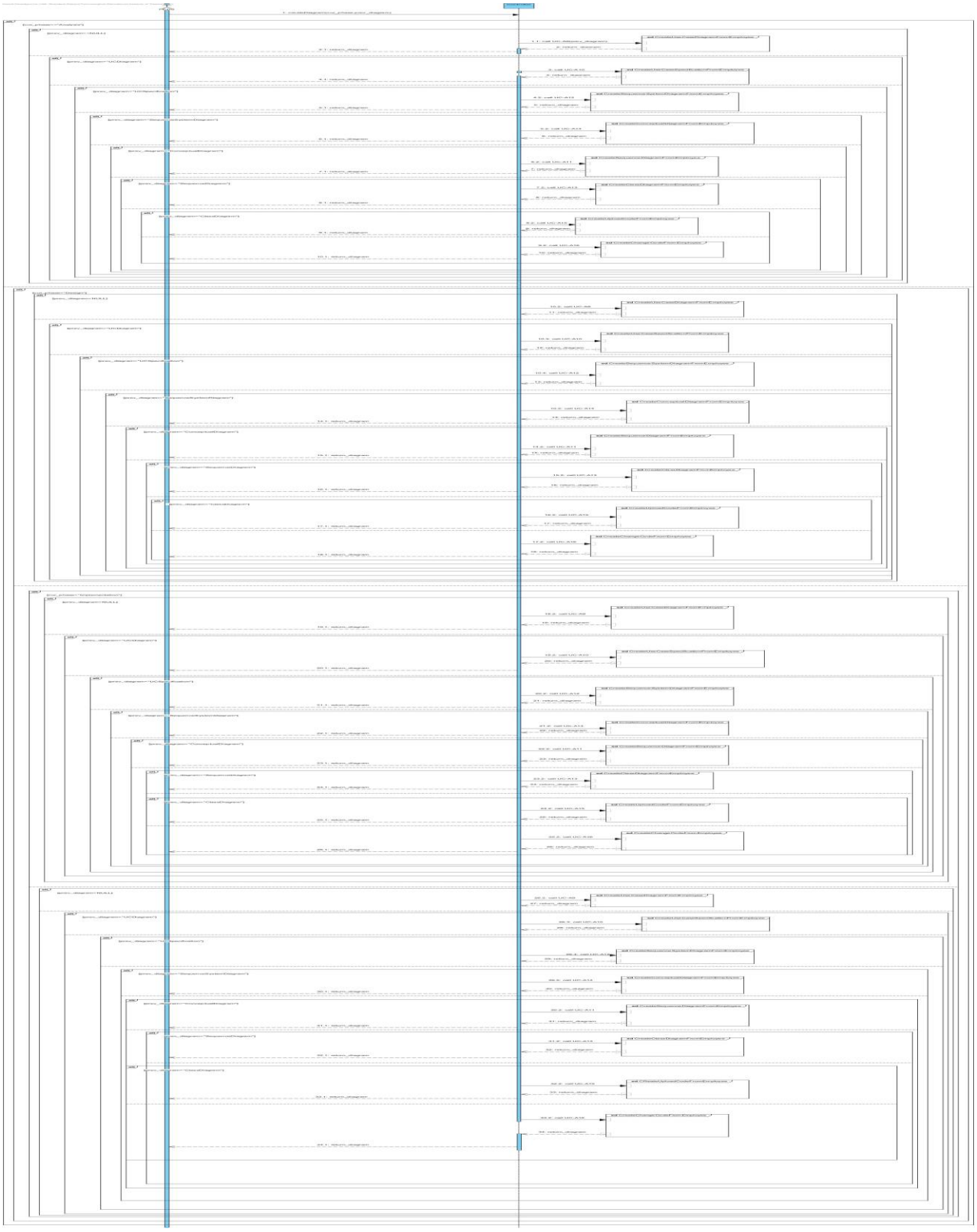
Σχήμα 40: Δημιουργία Υπαλλήλου



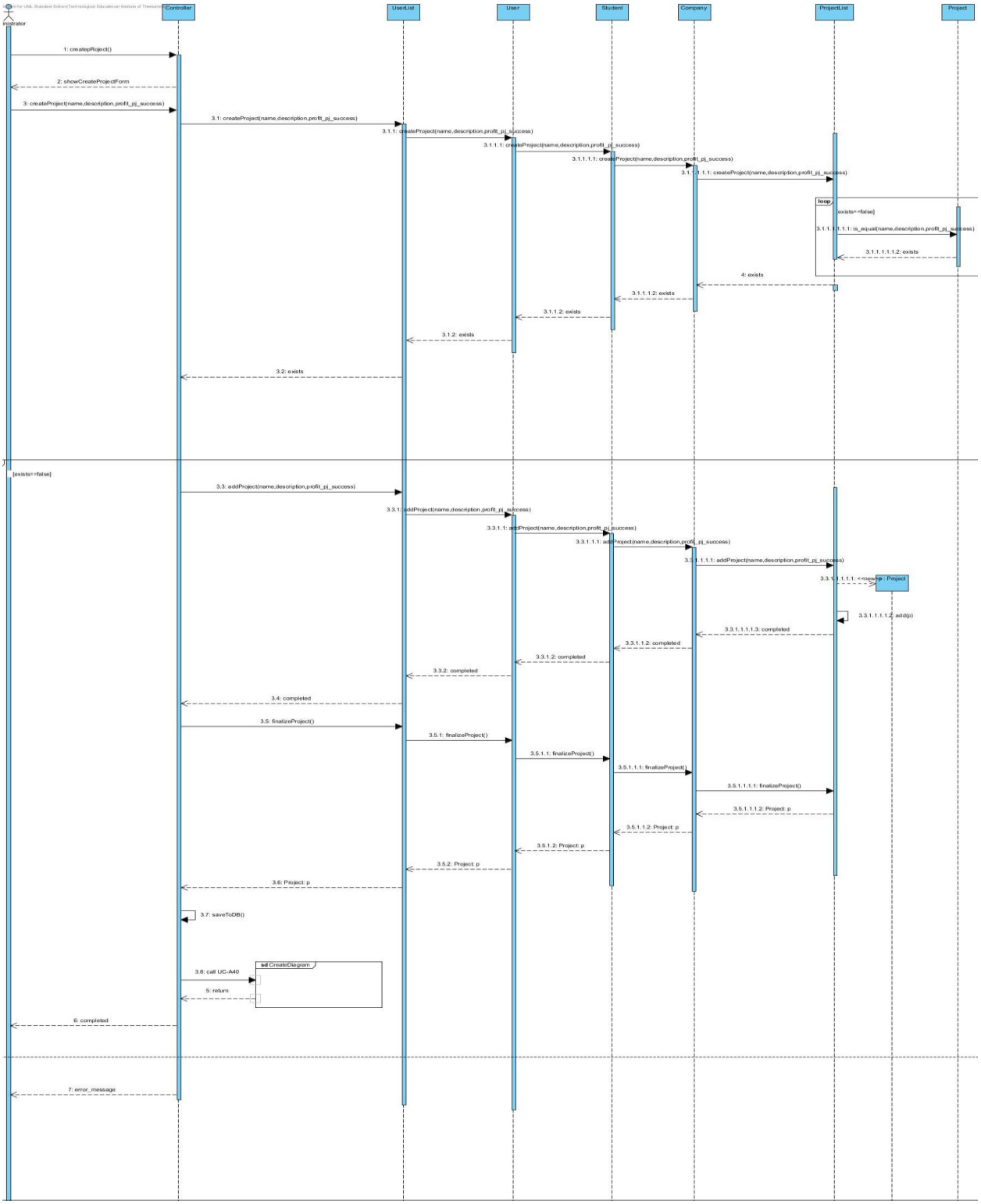
Σχήμα 41: Login



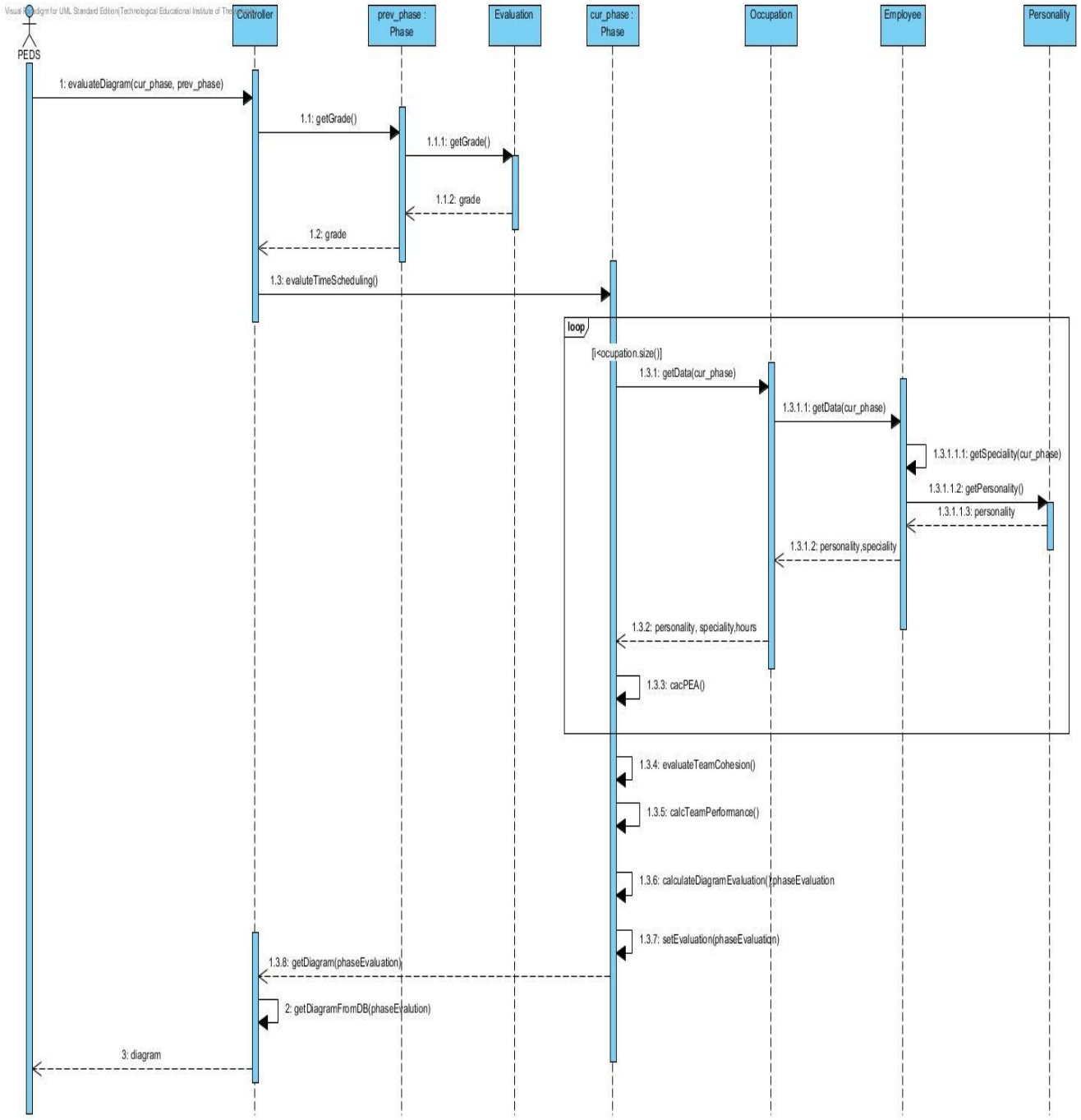
Σχήμα 42: Ενέργειες Υπαλλήλων



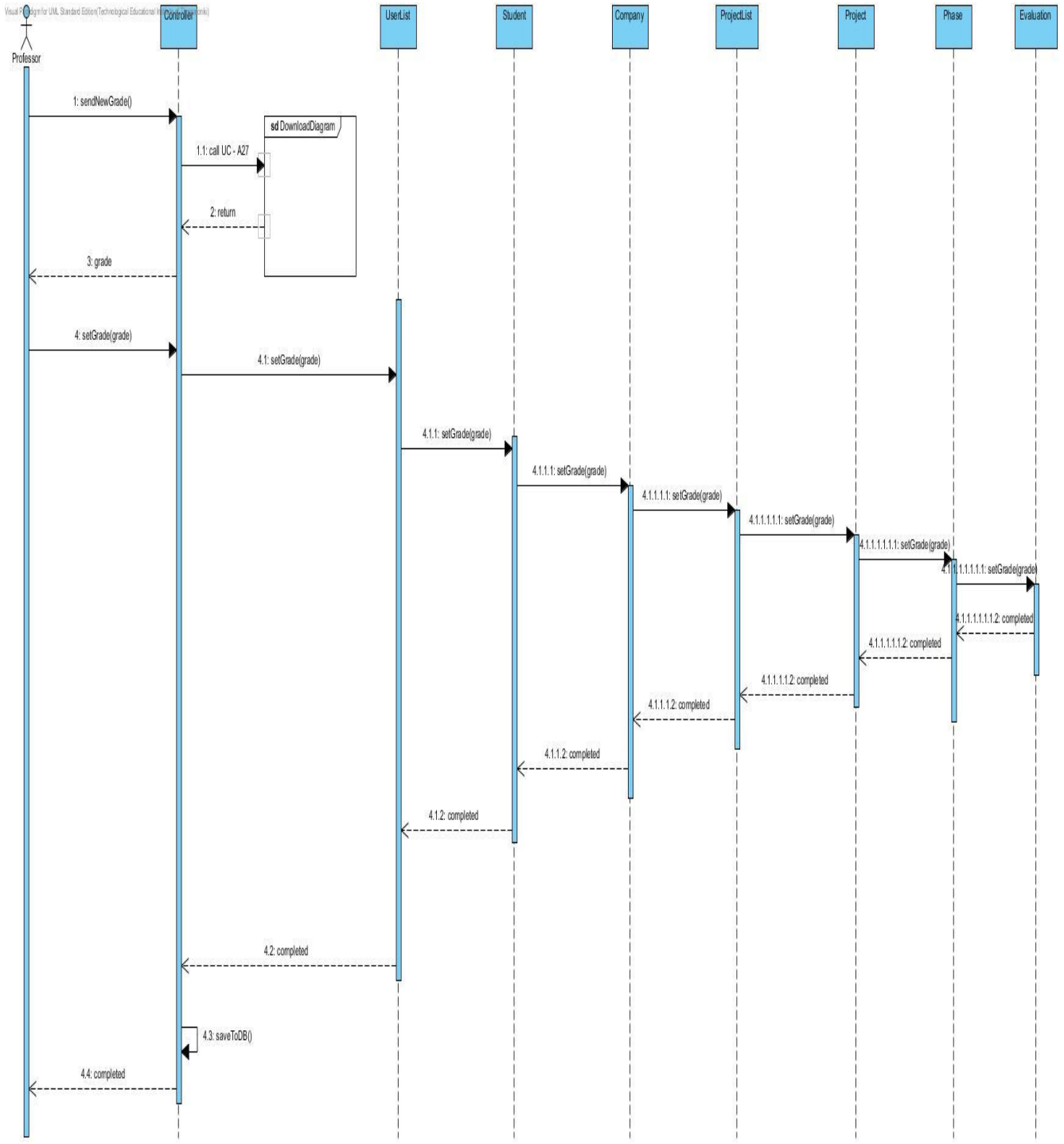
Σχήμα 43: Δημιουργία Project



Σχήμα 44: Αξιολόγηση Διαγράμματος

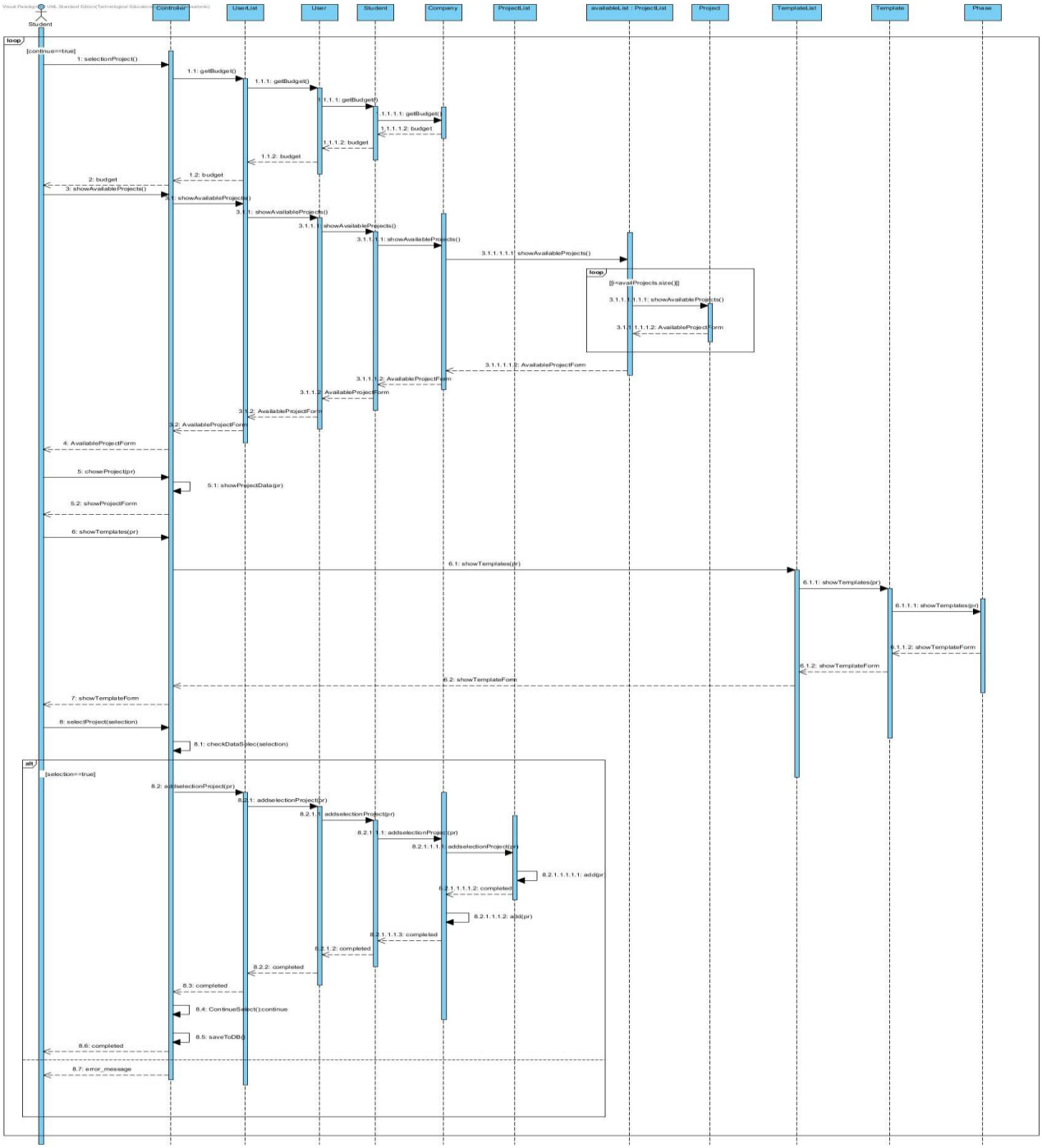


Σχήμα 45: Αξιολόγηση Διαγραμμάτων

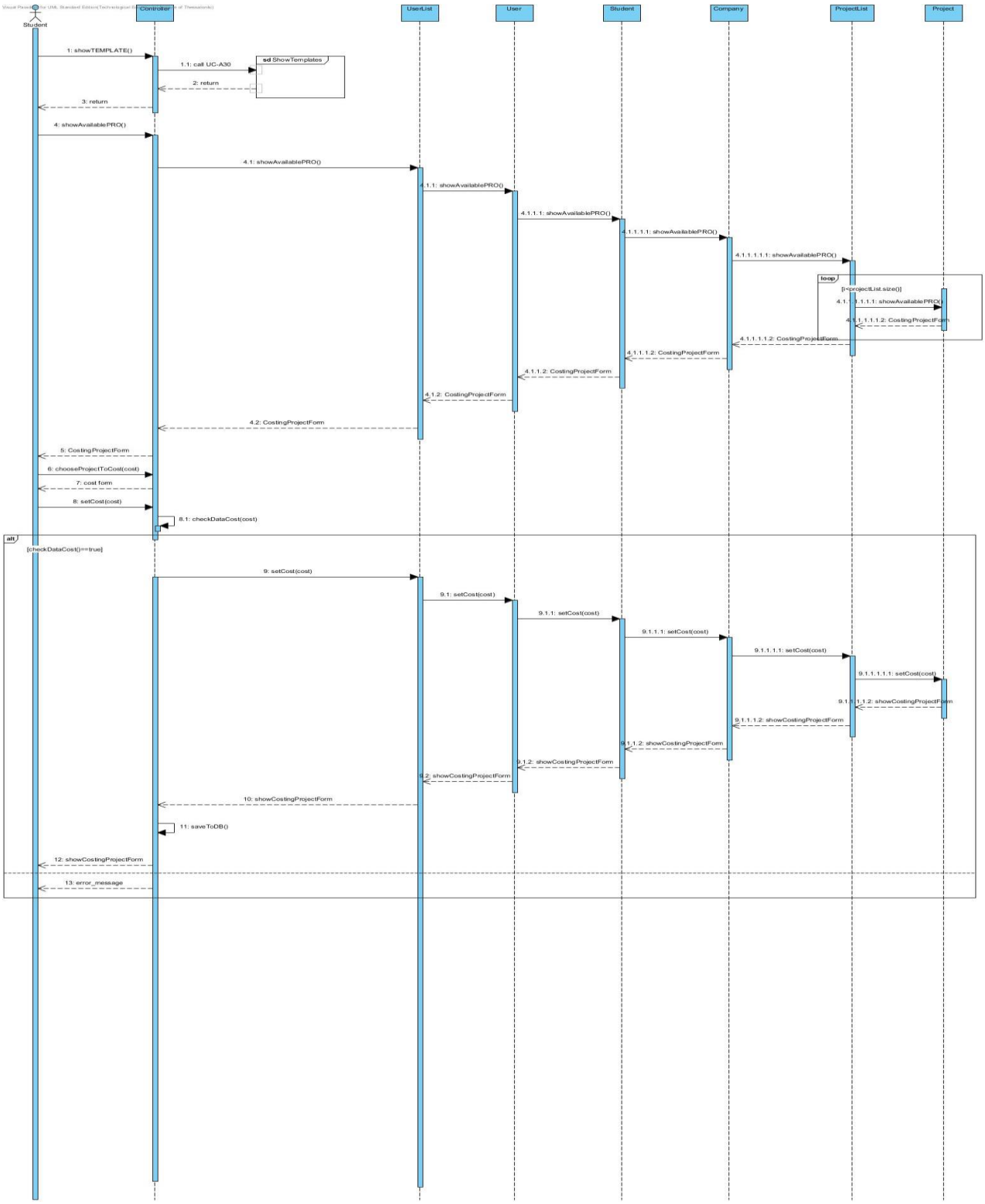


Σχήμα 46: Λήψη Διαγραμμάτων

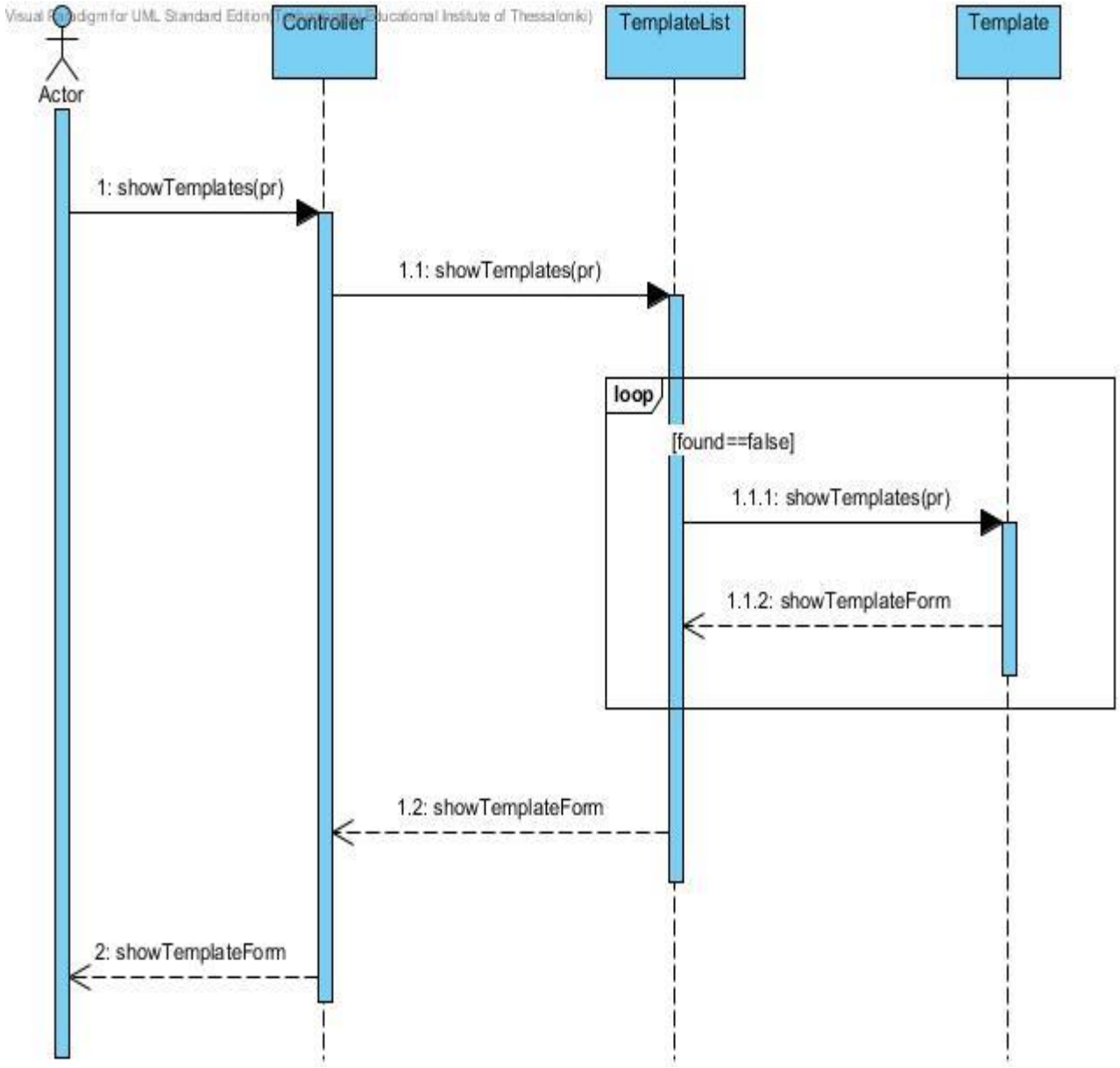
Σχήμα 47: Επιλογή Project



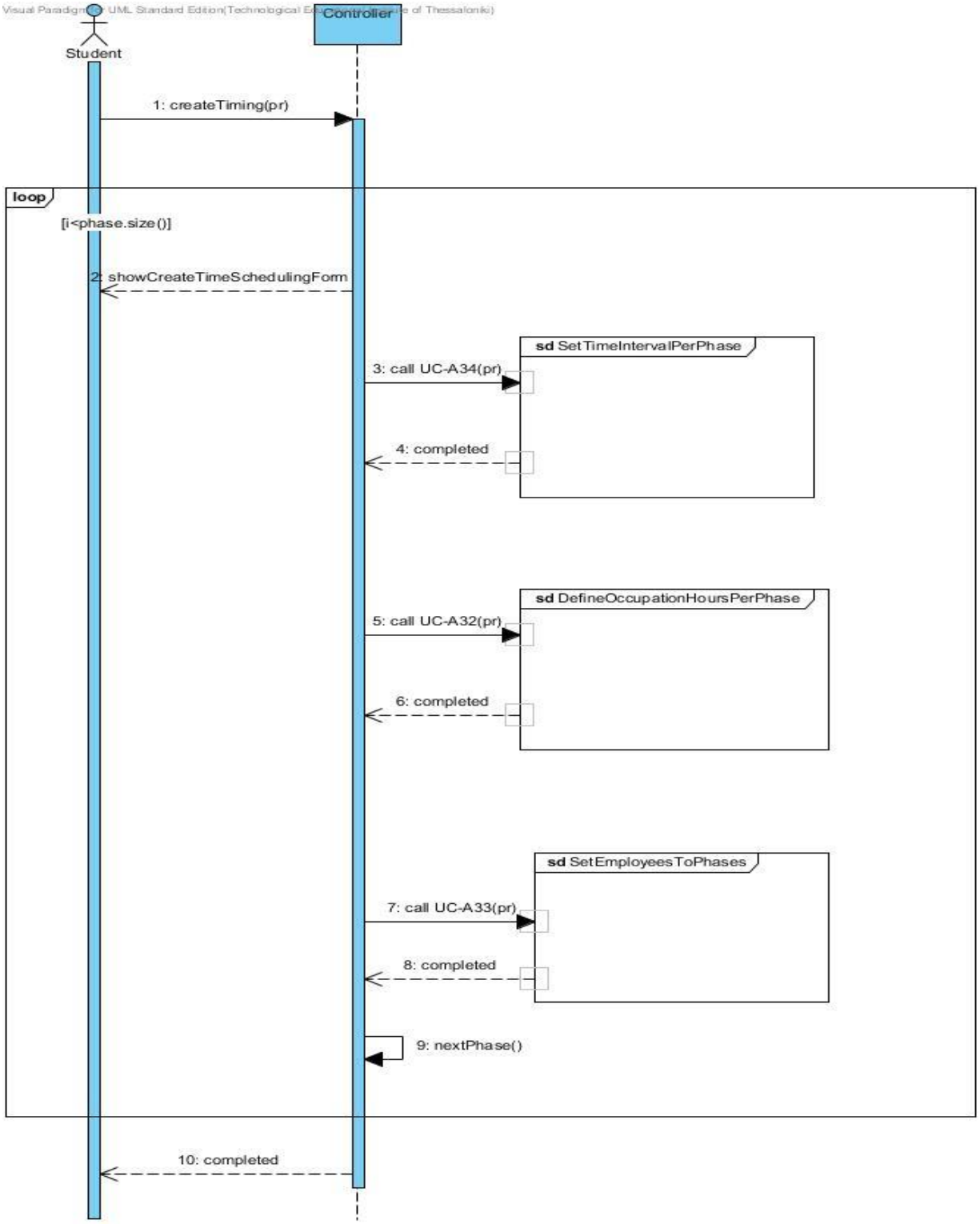
Σχήμα 48: Κοστολόγηση Project



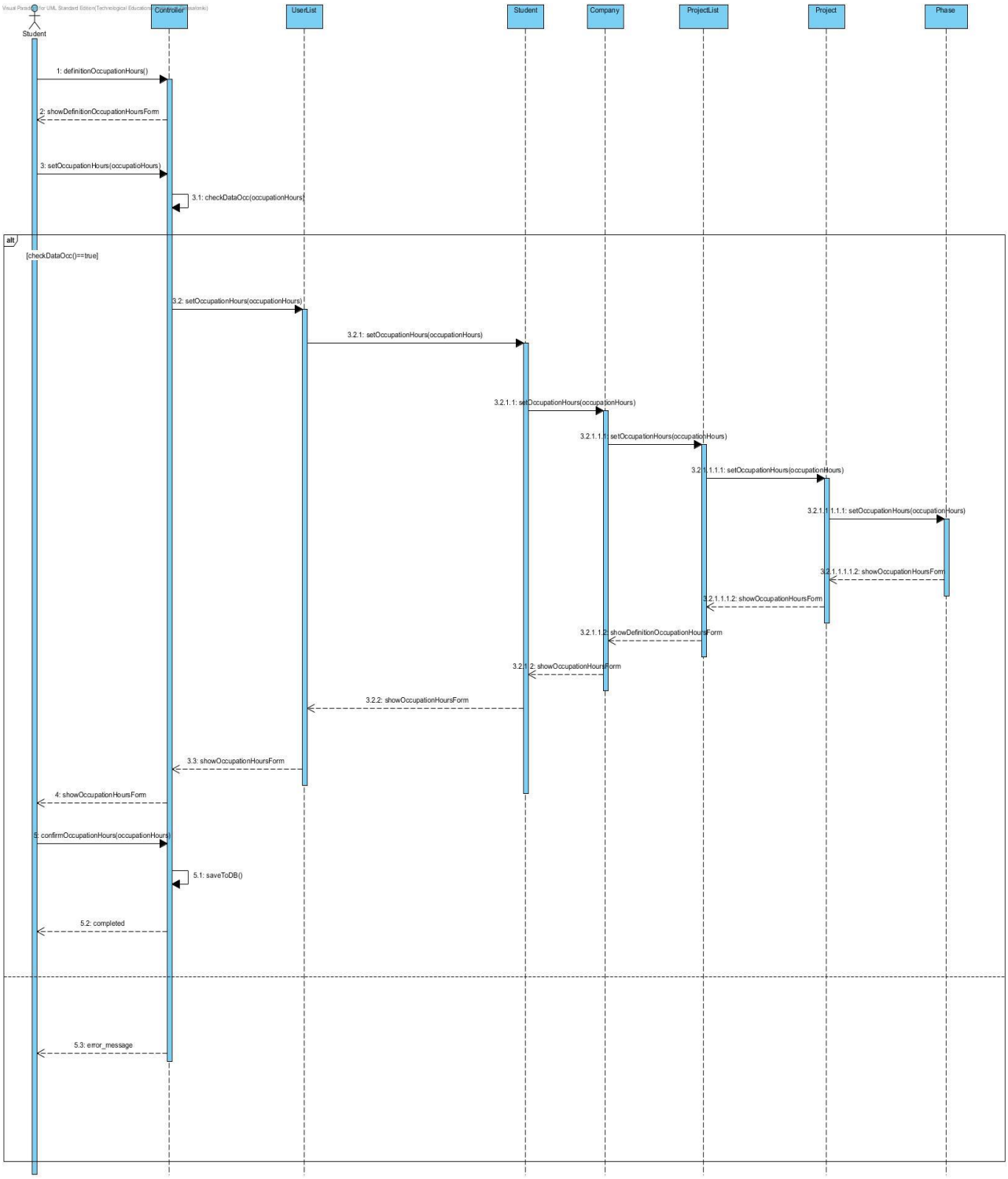
Σχήμα 49: Εμφάνιση Υποδειγμάτων



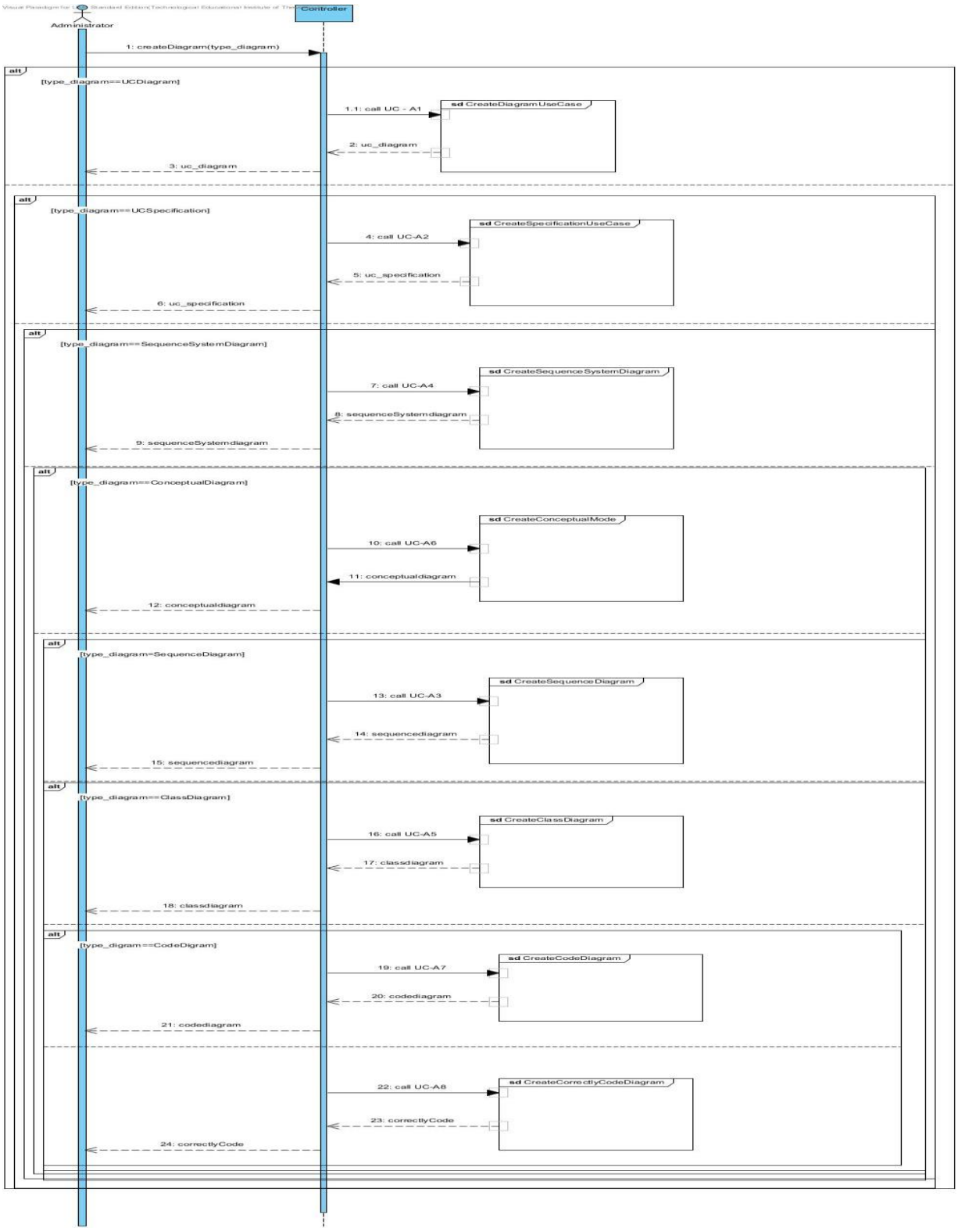
Σχήμα 50: Δημιουργία Χρονοπρογράμματος



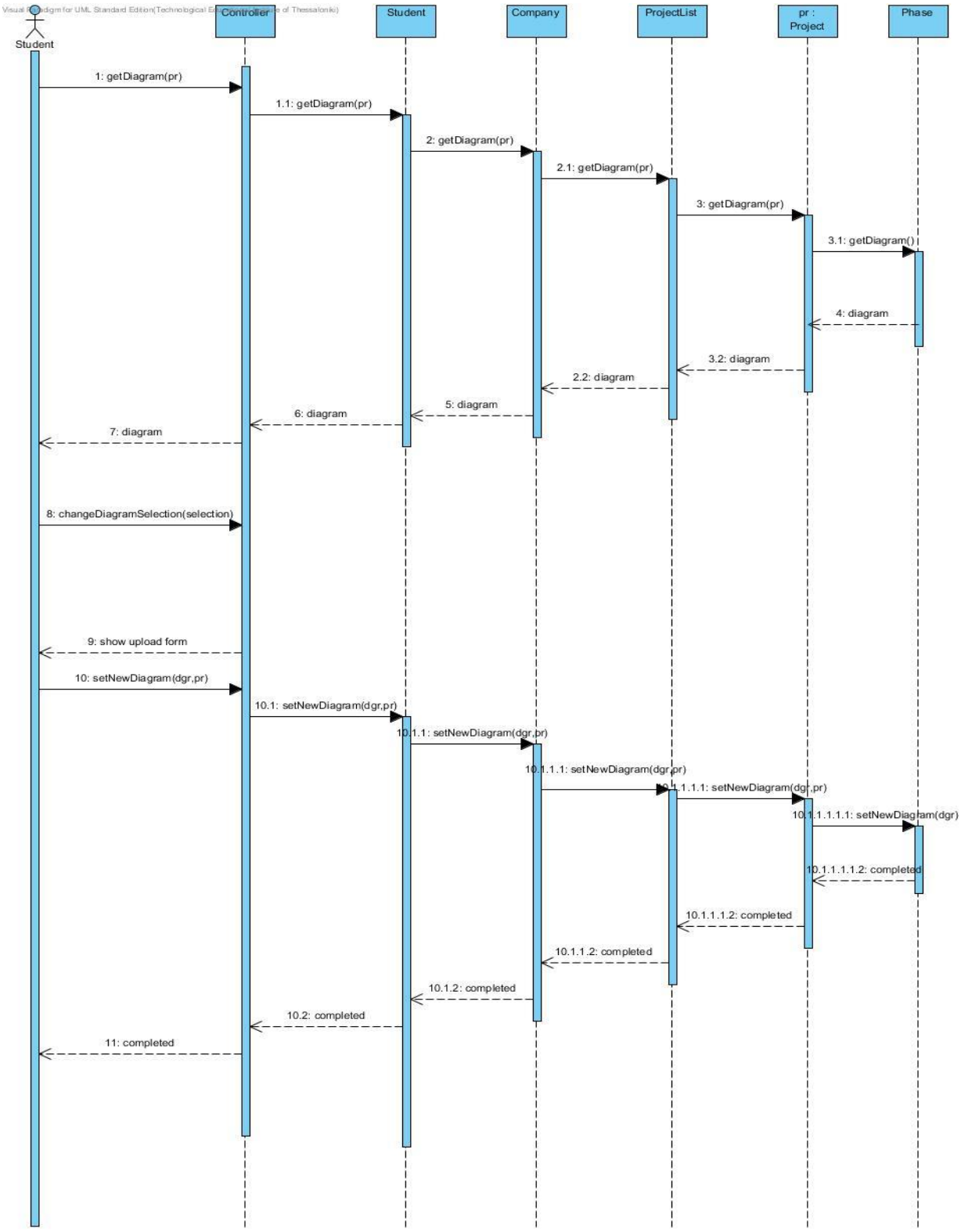
Σχήμα 51: Ορισμός Ανθρωποωρών Ανα Φάση



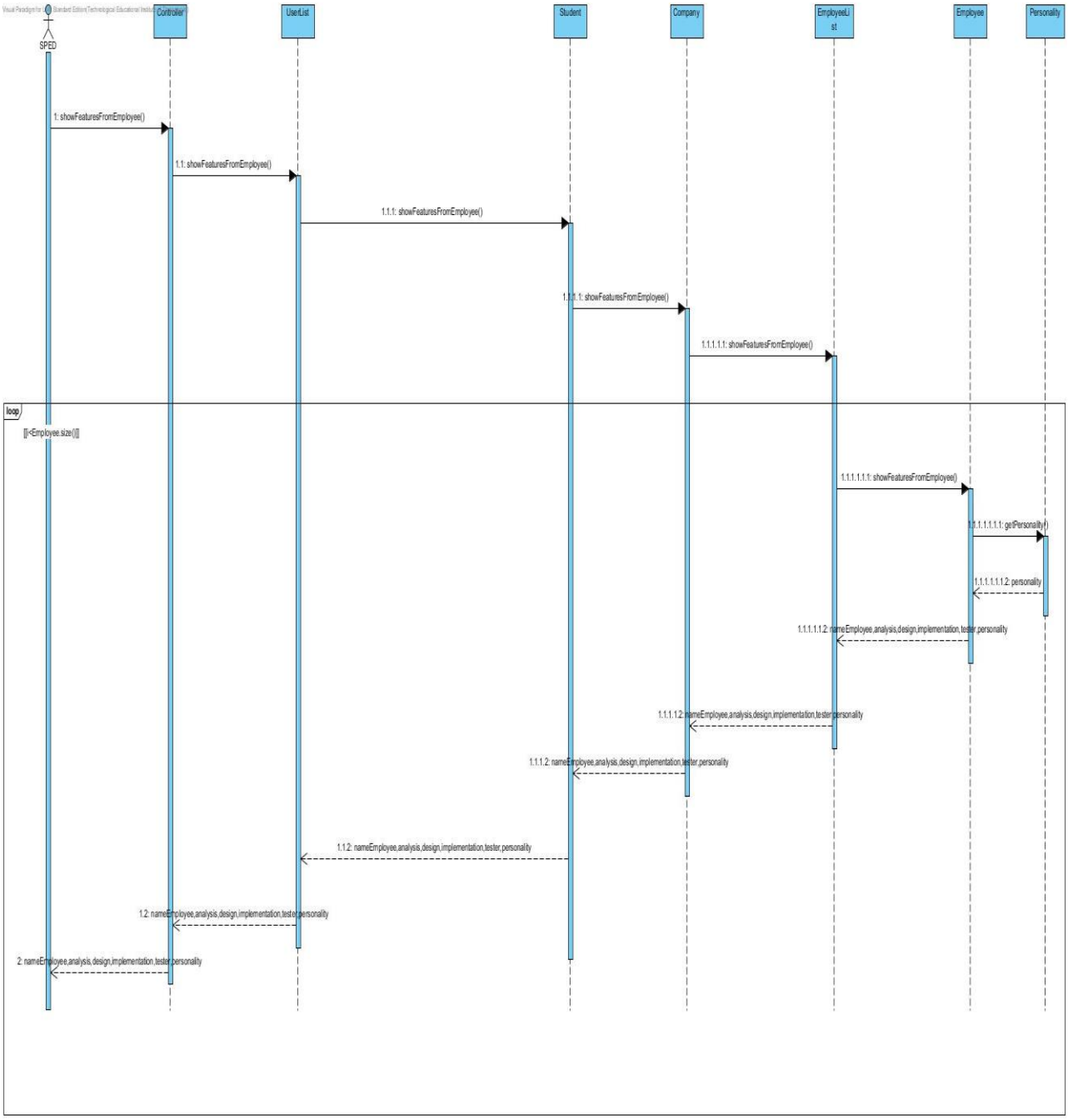
Σχήμα 52: Δημιουργία Διαγράμματος



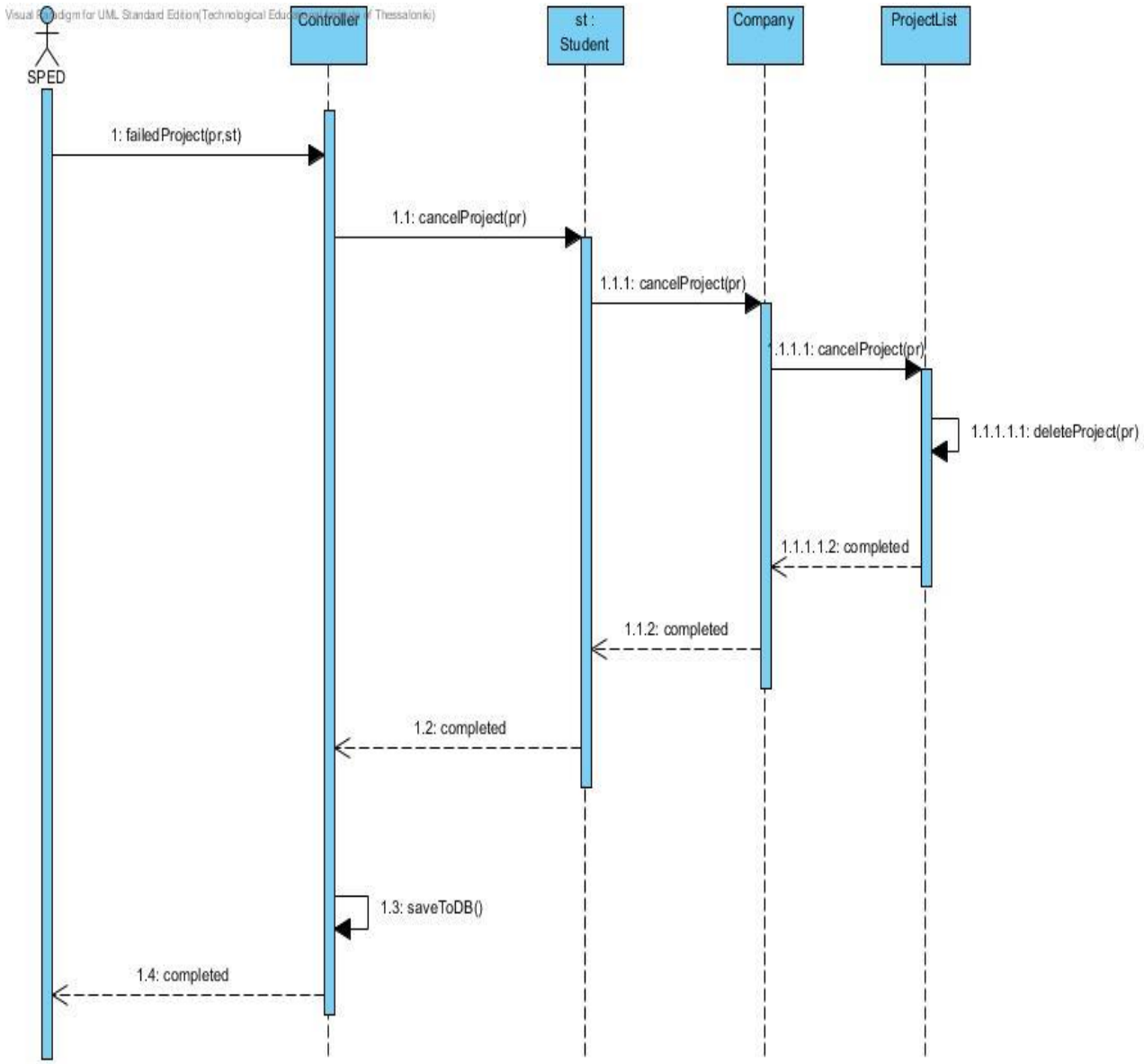
Σχήμα 53: Αλλαγή Διαγραμμάτων



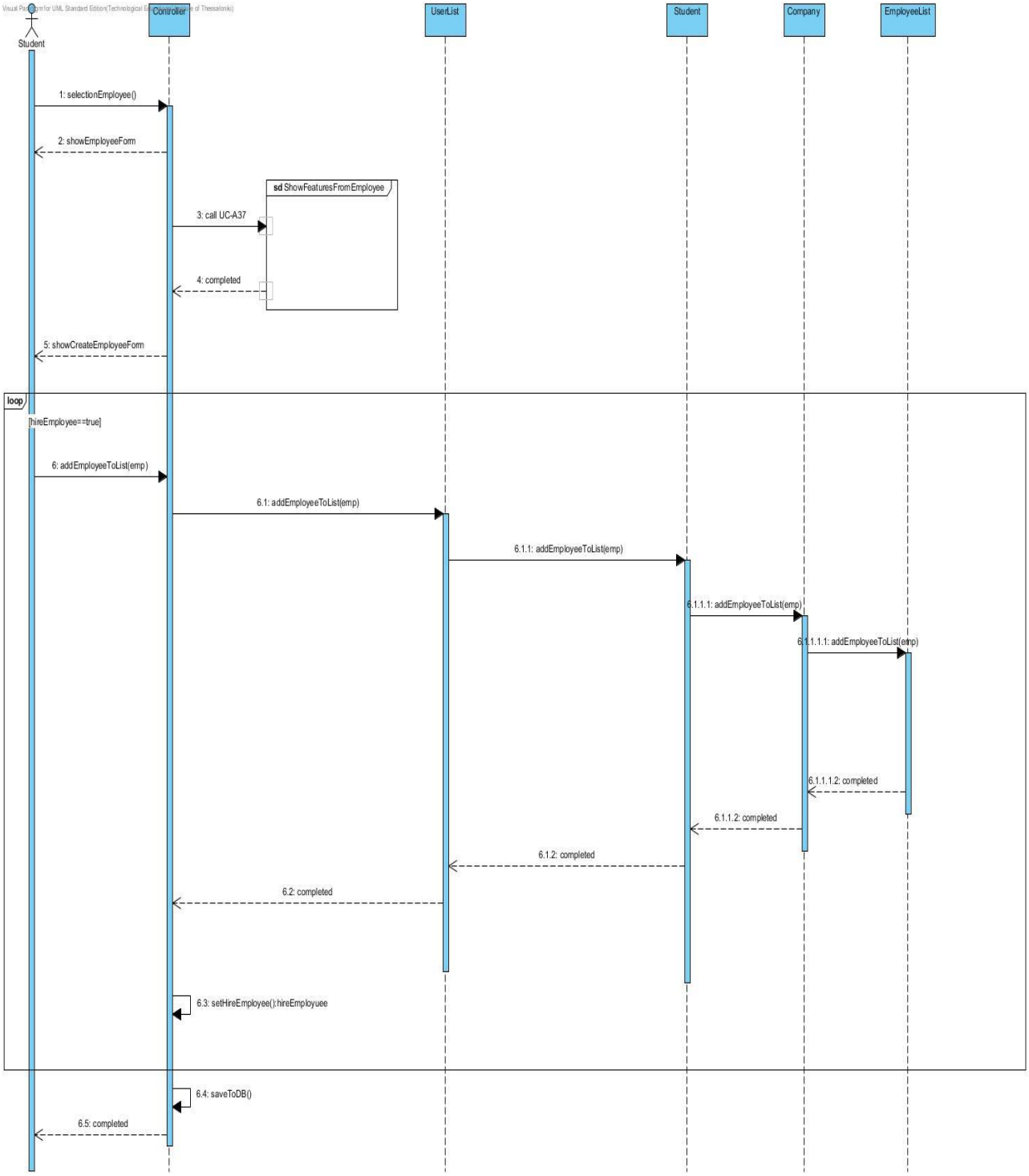
Σχήμα 54: Εμφάνιση Χαρακτηριστικών Υπαλλήλων



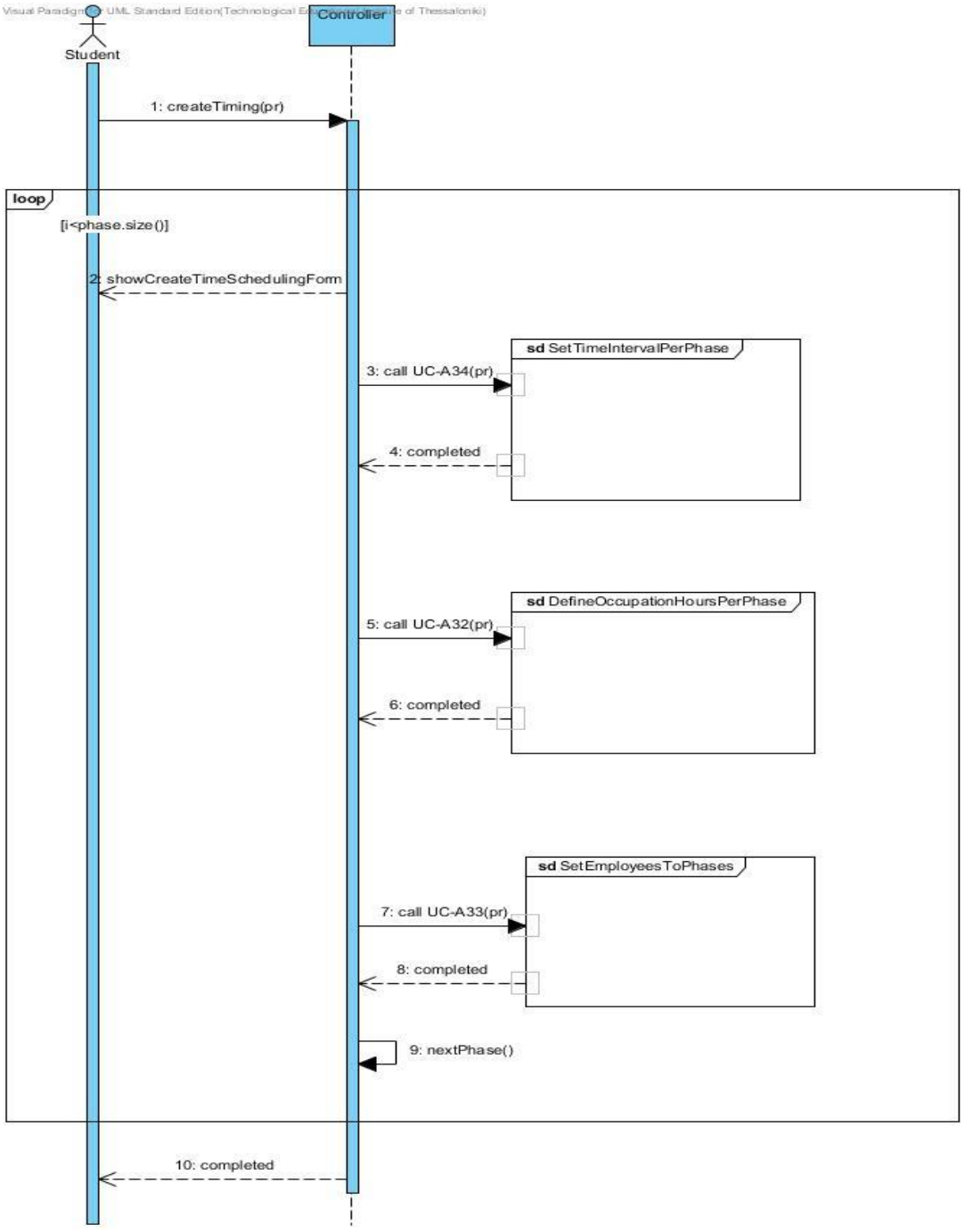
Σχήμα 55: Αποτυχία Project



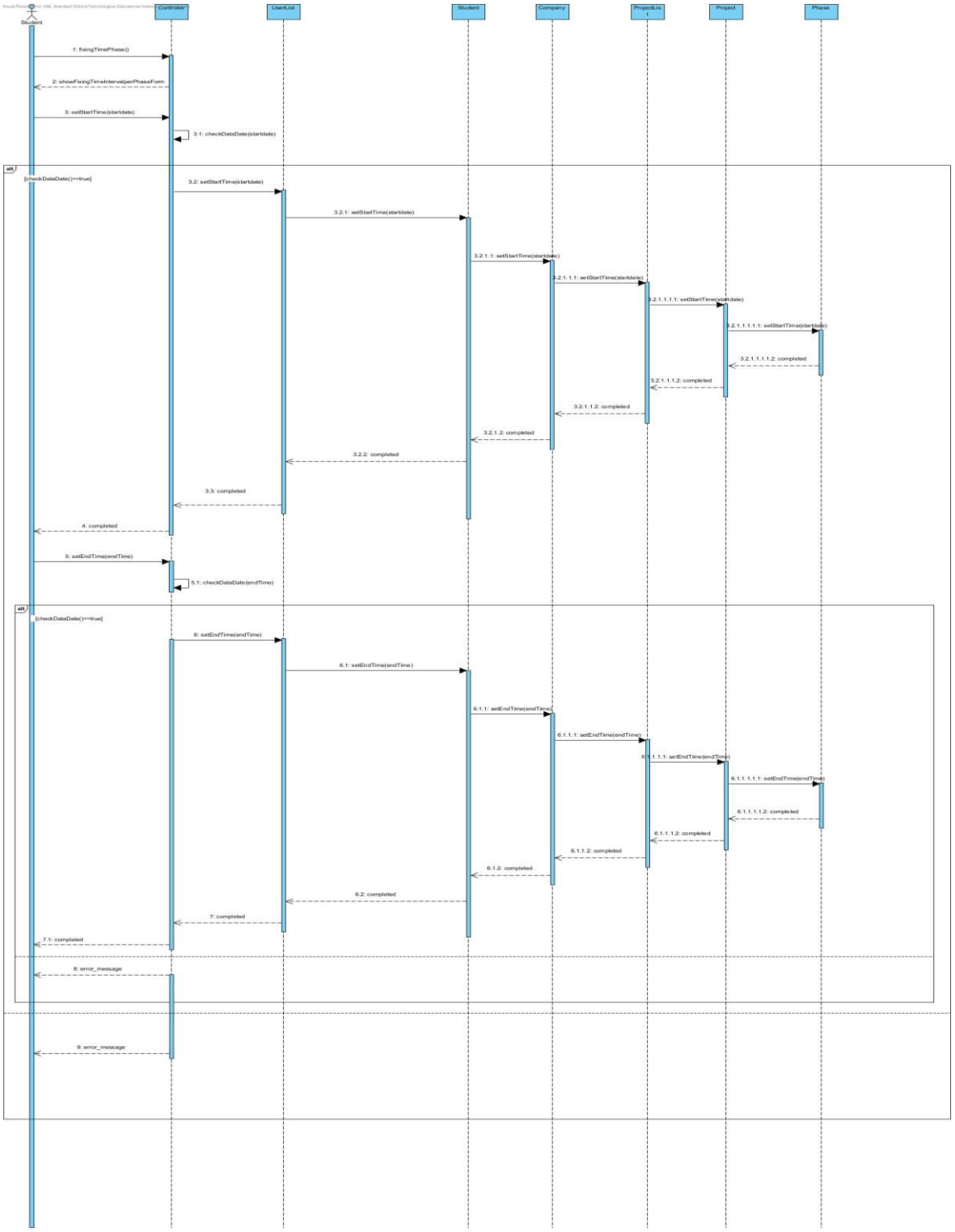
Σχήμα 56: Επιλογή Υπαλλήλων



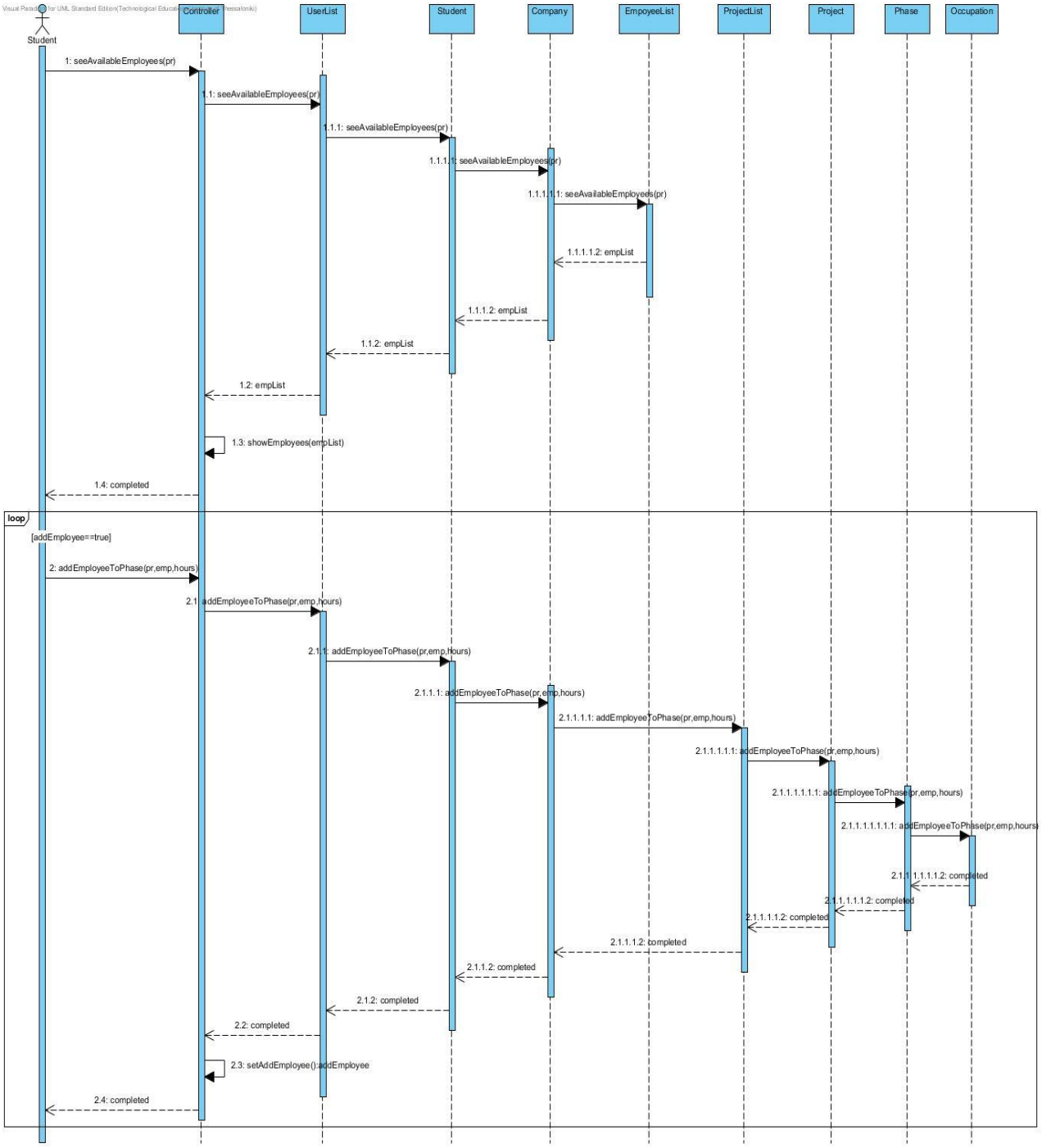
Σχήμα 57: Αξιολόγηση Χρονοπρογραμματισμού



Σχήμα 58: Καθορισμός Χρονικού Διαστήματος ανά Φάση



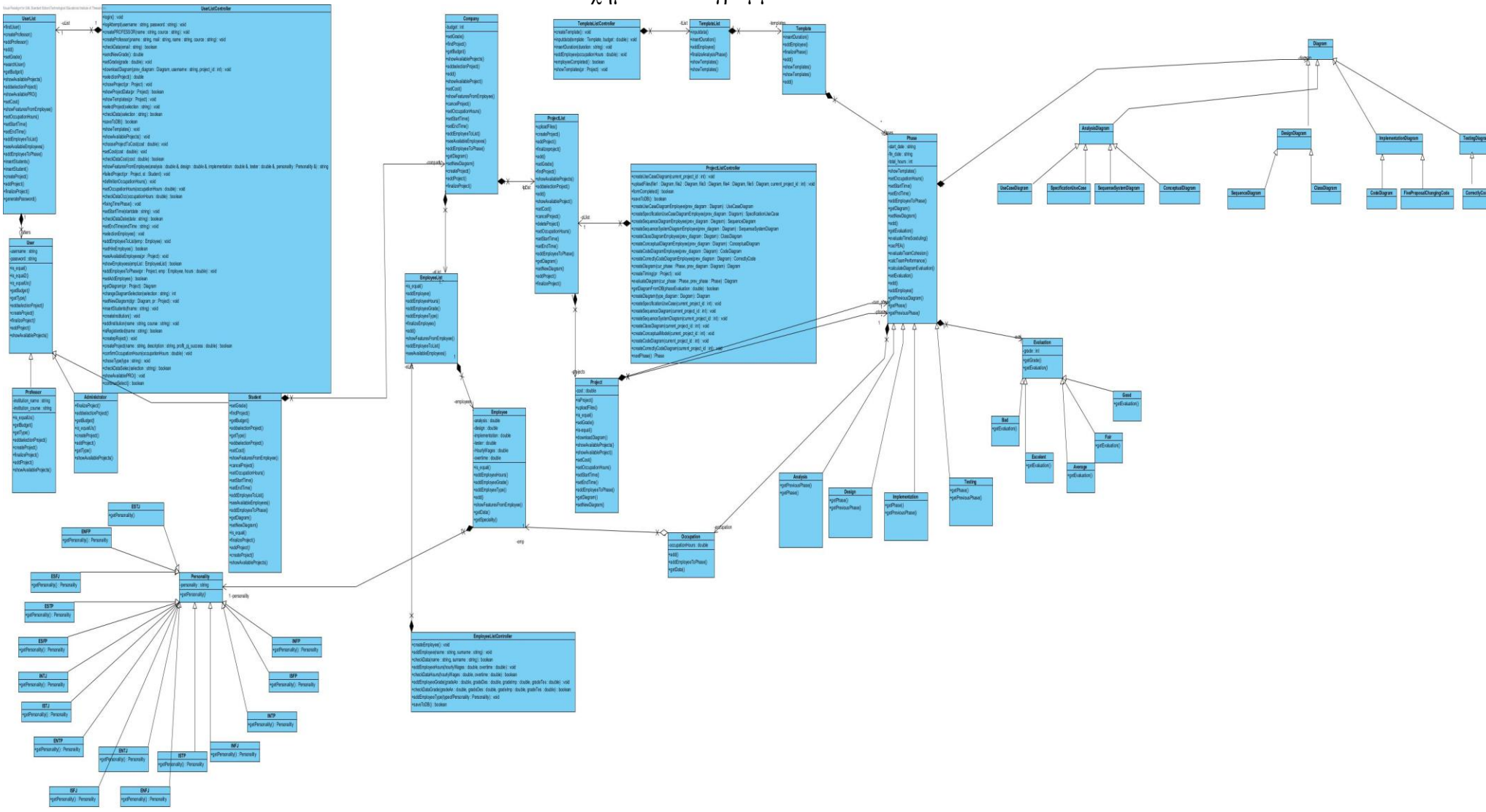
Σχήμα 59: Αντιστοίχιση υπαλλήλου με θέση ανά φάση



4.5 Διαγράμματα Κλάσεων

Στη συνέχεια παρουσιάζεται το διγραμμα κλάσεων του συστήματος στο οποίο περιέχονται τα πρότυπα σχεδίασης Στρατηγική (Strategy) και Κατάσταση (State). Το συνολικό διάγρμμα κλάσεων παρουσιάζεται στο σχήμα 60.

Σχημα 60: Διάγραμμα Κλάσεων



Για παράδειγμα ένα στιγμιότυπο Strategy pattern υπάρχει στο σημείο του κώδικα όπου γίνεται ο υπολογισμός της απόδοσης του συνόλου των υπαλλήλων για κάθε φάση ανάπτυξης του λογισμικού. Πιο συγκεκριμένα, στο συγκεκριμένο pattern instance συμμετέχουν οι κλάσεις Project, Phase, Analysis, Design, Implementation και Testing. Όλες οι υποκλάσεις της Phase έχουν την αρμοδιότητα να υλοποιήσουν τον αλγόριθμο υπολογισμού της προσπάθειας εταιρίας στη κάθε φάση (ΠΕΦ, calcPEF(), κεφάλαιο 4.1). Σε κάθε φάση, σύμφωνα με τον πίνακα 9, η προσωπικότητα του κάθε υπαλλήλου έχει διαφορετική επίδραση. Συμπερασματικά, κάθε υποκλάση πρέπει να χειρίζεται με διαφορετικό τρόπο το αποτέλεσμα της συνάρτησης getPersonality(). Παρ' όλα αυτά οι κανόνες της «καλής» σχεδίασης απαιτούν όλες οι υποκλάσεις της Phase να έχουν κοινό interface ώστε η Project να τις χειρίζεται με ενιαίο τρόπο. Τη λύση στο συγκεκριμένο πρόβλημα δίνει το Strategy pattern κάνοντας χρήση δυναμικού πολυμορφισμού.

Επιπλέον, η χρήση του προτύπου Στρατηγικής όπως εμφανίζεται από τα αποτελέσματα του πίνακα 6, έχει θετικές συνέπειες ως προς την αποσφαλμάτωση καθώς και ως προς τον αριθμό λαθών. Το γεγονός αυτό επιβεβαιώνεται και διαισθητικά λόγω της φύσης του προτύπου. Αναλυτικότερα, η πολυμορφική συμπεριφορά του συστήματος επιτρέπει τον διαχωρισμό των υλοποιήσεων των διαφορετικών αλγορίθμων σε διαφορετικές συναρτήσεις. Το γεγονός αυτό καθιστά πιο κατανοητό τον κώδικα και αυξάνει την ευελιξία του.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Η ενότητα αυτή, ασχολείται με την παρουσίαση των κινδύνων εγκυρότητας (threats to validity) της μελέτης. Σε κάθε εμπειρική μελέτη υπάρχουν διάφοροι κίνδυνοι για την εγκυρότητα των αποτελεσμάτων, όπως για παράδειγμα η γενίκευση τους εκτός του πεδίου εφαρμογής της μελέτης. Στην παρούσα μελέτη, τα αποτελέσματα δεν μπορούν να γενικευθούν στα υπόλοιπα από τα 23 GoF πρότυπα σχεδίασης, από τη στιγμή που στη μελέτη εξετάστηκαν μόνο 11 από αυτά. Επιπλέον, τα αποτελέσματα δεν μπορούν να χρησιμοποιηθούν για λογισμικό κλειστού κώδικα,

για τα παιχνίδια γραμμένα σε γλώσσες προγραμματισμού, εκτός από java και για κατηγορίες ανοιχτού κώδικα, εκτός από τα παιχνίδια.

Η εργασία αυτή έχει ως στόχο τον προσδιορισμό πιθανών συσχετίσεων μεταξύ του βαθμού εφαρμογής των προτύπων σχεδίασης, της συχνότητας λαθών και την αποτελεσματικότητα αποσφαλμάτωσης σε παιχνίδια ανοιχτού λογισμικού. Για το λόγο αυτό διεξάχθηκε μια μελέτη περίπτωσης σε 97 παιχνίδια ανοιχτού λογισμικού γραμμένα σε java.

Τα αποτελέσματα της μελέτης επιβεβαιώνουν ότι αρκετά πρότυπα σχεδίασης συσχετίζονται με τον αριθμό των λαθών σε παιχνίδια ανοιχτού λογισμικού, ενώ άλλα πρότυπα συσχετίζονται με το ποσοστό των δραστηριοτήτων αποσφαλμάτωσης. Για παράδειγμα, το πρότυπο Adapter εμφανίζεται ως ένα πρότυπο που έχει αρνητική επίδραση τόσο στην συχνότητα λαθών όσο και στην αποτελεσματικότητα της αποσφαλμάτωσσης. Μια πιθανή εξήγηση αυτού του αποτελέσματος είναι ότι ο Adapter χρησιμοποιείται πιο συχνά σε δραστηριότητες επαναχρησιμοποιημένου κώδικα. Πιο συγκεκριμένα, κατά την επαναχρησιμοποίηση κώδικα, ο προγραμματιστής μπορεί να μην έχει πλήρη επίγνωση του κώδικα που επαναχρησιμοποιεί. Έτσι, μπορεί να έχει προβλήματα στον εντοπισμό των λαθών. Ταυτόχρονα, είναι πιθανό να επαναχρησιμοποιηθούν σφάλματα των τμημάτων κώδικα τα οποία προστίθενται σε ανοιχτά λάθη του υπό ανάπτυξη συστήματος.

Από την άλλη πλευρά, η συστηματική χρήση στιγμιότυπων του προτύπου Observer μπορεί οδηγήσει σε μείωση των λαθών τα οποία είναι ανοιχτά σε ένα παιχνίδι ανοιχτού λογισμικού και επιταχύνουν τη διαδικασία αποσφαλμάτωσης. Μια πιθανή εξήγηση για αυτό είναι ότι οι προγραμματιστές που χρησιμοποιούν το πρότυπο Observer, είναι έμπειροι σε δραστηριότητες σχεδίασης, και ως εκ τούτου είναι λιγότερο επιρρεπείς σε λάθη. Επιπλέον, το γεγονός ότι το πρότυπο Observer οριοθετεί σαφώς τις ενότητες του παιχνιδιού, οδηγεί σε ευκολότερη αποσφαλμάτωση.

Παρ΄ολα αυτά δεν είναι σαφές, αν η χρήση των προτύπων σχεδίασης είναι η βασική αιτία ύπαρξης λαθών και της αποδοτικότητας κατά την διόρθωση τους σε παιχνίδια ανοιχτού λογισμικού. Μια εναλλακτική πιθανή εξήγηση συσχετίζει τη χρήση των προτύπων σχεδίασης με άλλα σημαντικά χαρακτηριστικά του έργου, όπως το η

εμπειρία της κοινότητας, η βιωσιμότητα του έργου κτλ. Τα εν λόγω θέματα αποτελούν αντικείμενο περαιτέρω έρευνας.

Μελλοντικά ερευνητικά σχέδια περιλαμβάνουν την επανάληψη της μελέτης περίπτωσης σε μεγαλύτερη ποικιλία έργων και σε διαφορετικούς τομείς. Μια τέτοια προσπάθεια θα παρέχει βαθύτερη κατανόηση σχετικά με το εάν τα αποτελέσματα αυτής της μελέτης σχετίζονται με τα παιχνίδια ή αν τα αποτελέσματα χαρακτηρίζουν το ανοιχτό λογισμικό γενικότερα. Επιπλέον, η φύση των λαθών πρόκειται να αξιολογηθεί και η σοβαρότητά τους πρόκειται να συσχετιστούν με τα πρότυπα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Alberg, H.,Ohlsson, N. (1996):" Predicting fault-prone software modules in telephone switches". *IEEE Transactions on Software Engineering*, 22, 886—894

Allen, E.B., Aud, S.J., Hudepohl, J.P., Khoshgoftaar, T.M., Mayrand, J. (1996): Emerald: "Software metrics and models on the desktop". *IEEE Software*. 13, 56—60

Allen, E.,B., Goel, N., Khoshgoftaar, T.M., McMullan, J., Nandi, A. (1996): "Detection of software modules with high debug code churn in a very large legacy system". In: *Seventh International Symposium on Software Reliability Engineering*, pp.364-371, NY

Aversano L., Canfora G., Cerulo L., Grosso C.D., Penta Di M., (2007): "An empirical study on the evolution of design patterns*", Foundations of Software Engineering (FSE' 7), ACM*, pp. 385-394, Dubrovnik, Croatia, 3-7 September 2007

Ampatzogloy A., Chatzigeorgiou A. (2007), "Evaluation of object-oriented design patterns in game development",In *Information and Software Technology*, Elsevier, Vol. 49, No 5, pp. 445-454

Antoniol G., Fiutem R., Cristoforetti L., (1998): "Using Metrics to Identify Design Patterns in Object-Oriented Software*", Proceedings Metrics 1998: Fifth International Software Metrics Symposium,* pp. 23-34

Antoniol G., Casazza G., DiPenta M., Fiutem R., (2001): "Object-Oriented Design Patterns Recovery", *J. Systems and Software*, vol. 59, no.2, pp. 181-196

Albin-Amiot H., Cointe P., Gueheneuc Y.G, Jussien N., (2001): "Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together*", Proceedings 16th Ann. International Conference Automated Software Engineering*, pp. 26-29

Badros G.J., Notkin D., (2000): "A Framework for Preprocessor-Aware C Source Code analyses", *Software-Practice and Experience*, vol. 30, no.8, pp. 907-924

Ball, T., Nagappan, N. (2005): "Use of Relative Code Churn Measures to Predict System Defect Density". In: *International Conference on Software Engineering*, pp. 284—292, St. Louis, MO

Ball, T., Nagappan, N., Zeller, A. (2006): "Mining Metrics to Predict Component Failures". In: *International Conference on Software Engineering*, pp. 452—461, China

Ball, T., Nagappan, N. (2007): "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Emperical Case Study". In: *International Symposium on Empirical Software Engineering and Measurement*, pp.364-373, Madrid, Spain

Balanyi Z., Ferenc R., (2003): “Mining Design Patterns from C++ Source Code”, *Proceedings International Conference Software Maintenance*, pp. 305-315

Basili, V.R., Briand, L.C., Melo, V.L. (1996): “A Validation of Object Orient Design Metrics as Quality Indicators”, *IEEE Transactions on Software Engineering*, 22, 75—761.

Baudry B., Sunye Y. Le., Jezequel M., (2001): “Towards a ‘Safe’ Use of Design Patterns to Improve OO Software Testability*”, Proceedings of the 12th International Symposium on Software Reliability Engineering, IEEE*, pp. 324, Hong Kong, China, 27-30 November 2001

Baudry B., Traon Y. Le., Sunye G., Jezequel (2003):”Measuring and Improving Design Patterns Testability”, *Proceedings of the 12th International Symposium on Software Metrics, IEEE* pp. 50, Sydney, Australia, 03-05 September 2003

Bansiya J., (1998): “Automating Design Pattern Identification”, *Dr. Dobb’s J*., vol 23, no 6, pp. 20-28, June 1998

Basili V. R., (1993): “The Experimental Paradigm in Software Engineering”, *Experimental Software Engineering Issues: Critical Assessment and Future Directives*

Basili V. R., R.W. Selby, D.H. Hutchens, (1986): *In IEEE Transactions on Software Engineering,* “Experimentation in Software Engineering”, IEEE Computer Society

Bell, R.M., Ostrand, T., Weyuker, E. (2005): “Predicting the location and number of faults in large software systems”. *IEEE Transactions In Software Engineering* 31, 340-355

Bener, A., Tosun, A., Turhan, B. (2009): “Validation of network measures as indicators of defective modules in software systems”. In: *5th Inthernational Conference on Predictor Models in Software Engineering*, Vancouver , Canada

Bevan, J., Whitehead, E.J. (2003): “Identification of Software Instabilities”. In: *10th Working Conference on Reverse Engineering*, pp.134-145, Victoria B.C., Canada

Beck K., Crocker R., Meszaros G., Vlissides J., Coplien J. O., Dominic L., Paulisch F., (1996): “Industrial experience with design patterns”, *Proceedings of the 18th International Conference on Software Engineering (ICSE ‘96),* IEEE, pp. 103-114, Berlin, Germany, 25-29 March 1996

Bieman J. M., Straw G., Wang H., Munger P. W., Alexander R. T., (2003): “Design Patterns and Change Proneness: An Examination of Five Evolving Systems”,

Proceedings of the 9th International Symposium on Software Metrics, IEEE, pp. 40, Sydney, Australia, 03-05 September 2003

Binkley A.B., Schach, S.R (1998): "Validation of the coupling dependency metric as a predictor of failures and maintenance measures". In: *International Conference on Software Engineering*, pp.452—455, Kyoto

Capretz L.F., "Personality Types in Software Engineering," *Int'l J. Human-Computer Studies*, vol. 58, no. 2, 2003, pp. 207–214.

Chatzigeorgiou A. (2005), "Object –Oriented Design: UML, Principles, Patterns and Heuristics", *Kleidarithmos*, Athens, 1st edition.

Clarke, L.A., Podgurski, A. (1990): "Formal Model of Program Dependences and its Impications for Software Testing, Debugging and Maintenance". *IEEE Transactions on Software Engineering*. 16, 965—979

Di Penta M., Cerulo L., Gueheneuc Y. G., Antoniol G. (2008) : "An Empirical Study of Relationships between Design Pattern Roles and Class Change Proneness", In: *ICSM 2008, 24th International Conference on Software Maintenance, IEEE Computer Society*, Beijing¸China, September 28 October 4, 2008, pp. 217-226

DeMaris A., (1991): "A Framework for the Interpretation of First-Order Interaction in Logit Modeling", *Psychological Bulletin*, vol.110, no.3, pp. 557-57

Elish M., (2006): "Do Structural Design Patterns Promote Design Stability*?", Proceedings of the 30th Annual International Computer Software and Applications Conference- Volume 01* (COMPSAC '06), IEEE, pp. 215-220, Illionois, 17-21 September 2006

Ellis B., Stylos J., Myers B., (2007): "The Factory Pattern in API Design: A Usability Evaluation", *Proceedings of the 29th International Conference on Software Engineering, IEEE*, pp. 302-312, Minneapolis, Minnesota, 20-26 May 2007

Florijn G., Meijers M., Winsen P. van, (1997): "Tool Support for Object-Oriented Patterns", *Proceedings Object-Oriented Programming*, vol, 1241, pp. 472-495

Gamma E, Helms R, Johnson R, Vissides J. (1995), "Design Patterns: Elements of Reusable Object-Oriented Software", *Addison-Wesley Professional*, Reading, MA, 1st edition.

Graves, T.L., Karr, A.F., Marron, J.S., Siy, H. (2000): "Predicting Fault Incidence Using Software Change History". *IEEE Transactions on Software Engineering*, pp.653-661

Gatrell M., Counsell S., Hall T. (2009) : "Design Patterns and Change Proneness: A Replication Using Proprietary C# Software*", Proceedings of the 2009 16th Working Conference on Reverse Engineering*, pp. 160-164, Lille, France, 13-16 October 2009

Glass R., (1994): "The Software Research Crisis", *IEEE Software*, pp. 42-47, November 1994

Gustafsson J., Paakki J., Nenonen L., Verkamo A. I., (2002): "Architecture-Centric Software Evolution by Software Metrics and Design Patterns", *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, IEEE, pp. 108, Budapest, Hungary, 11-13 March 2002

Geuheneuc J., Sahraoui H., Zaidi F., (2004): "Fingerprinting Design Patterns", *Proceedings of the 11th Working Conference on Reverse Engineering,* pp. 172-181, delft, The Netherlands, 08-12 November 2004

Gueheneuc Y.-G., Albin-Amiot H.,(2001): "Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects," *Proceedings of the 39th International Conference and Exhibition Technology of Object-Oriented Language and Systems*, pp. 296-305

Gorla Narasimhaiah and Yan Wah Lam. (2004): "Who should work with whom?: building effective software project teams". *Commun. ACM 47*, 6 (June 2004), 79-82.

Hsueh N. L., Chu P. H., Chu W., (2008): "A quantitative approach for evaluating the quality of design patterns", Journal of Systems and Software, Elsevier, 81 (8), pp. 1430-1439, August 2008

Hosmer D.W. and Lemeshow S., (2000): *Applied Logistic Regression, second ed. John Wiley and Sons*

Huston B., (2001): "The effects of design pattern application on metric scores", *Journal of Systems and Software, Elsevier*, 58 (3), pp. 261-269, September 2001

Jain D., Yang H. J., (2003): "OO Design Patterns, Design Structuring and Program Changes: An Industrial Case Study", *Proceedings of the IEEE International Conference on Software Maintenance* (ICSE '01),IEEE, pp. 580,Florence, Italy, 07-09 November 2003

Jaccard J., (2001): *International Effects in Logistic Regression. Sage Publications*

Kafura,D.G., Henry, S.M. (1998):" Software Structure Metrics based on Information Flow". *IEEE Transactions on Software Engineering*. 7, 510—518
Kramer C., Prechelt L., (1996): "Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software", *Proceedings of the Third Working Conference Reverse Engineering*, pp. 208-215

Kleinbaum D.G., (1994): "Logistic Regression: A self-Learning Text"

Kouskoyras K., Chatzigeorgiou A., Stephanides G., (2008): "Facilitating software extension with design patterns and Aspect-Oriented Programming*", Journal of Systems and Software,* Elsevier, 81 (10), pp 1725-1737, October 2008

Khomh F., Gueheneuc Y. G., (2009): "Playing roles in design patterns: An empirical descriptive and analytic study*", Proceedings of the 25th IEEE International Conference on Software Maintenance*, IEEE, pp, 83-92, Edmonton, Canada, 20-26 September 2009

Khomh F., Gueheneuce Y. G., (2008): "Do Design Patterns Impact Software Quality Positively?", Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering, IEEE, pp. 274-278, Athens, Greece, 01-04 April 2008

Keller R., Schauer R., Robitaille S., Page P., (1999): "Pattern-Based Reverse-Engineering of Design Componentns", *Proceedings 1999 International Conference Software Engineering,* pp. 226-235

Kitchenham B., L. Pickard, S.L. Pfleeger, (1995): *In IEEE Software*, "Case Studies for Method and Tool Evaluation".

Myers I (1962) Manual: "the Myers-Briggs type indicator". *Consulting Psychologists Press*, Palo Alto, California.

Malloy B. A., Power J. F., (2006): "Exploiting design patterns to automate validation of class invariants: Research articles*", Software Testing Verification and Reliability, Wiley Interscience,* 16 (2), pp. 71-95, June 2006

McNatt W. B., Bieman J. M., (2001): "Coupling of Design patterns: Common Practices and Their Benefits", *25th Annual International Computer Software and Applications Conference (COMPSAC '01),* IEEE, pp.574, Chicago, Illinois, 08-12 October 2001

Muraki T., Saeki M., (2001): “Metrics for applying GOF design patterns in refactoring processes”, *Proceedings of the 4th International Workshop on Principles of Software Evolution,* IEEE, pp. 27-36, Vienna, Austria, 10-11 September 2001

Nagappan, N., Zimmermann, T. (2008): “Predicting defects using network analysis on dependency graph”. In: *the 30th International Conference on Software Engineering* (ICSE ’08), pp. 531—540, Leipzig, Germany

Nagappan, N., Ball, T., Zeller, A. (2006): “Mining metrics to predict component failure”. In: *International Conference on Software Engineering* , pp. 452—461, China

Nielson S. J., Knuston C. D., (2006): “Design dysphasia and the pattern maintenance cycle”, *Information and Software Technology*, Elsevier, 48 (8), pp. 660-675, August 2006

Ng T. H., S.C. Cheung, W. . Chan, Yu Y.T (2006) : “Toward effective deployment of design patterns for software extension: a case study*”, Proceedings of the 2006 international workshop on Software quality (ICSE’06)*, IEEE, pp. 51-56, Shanghai, China, 21 May 2006

Ng T. H., Cheung S. C., Chan W. K., Yu Y. T., (2007): “Do Maintainers Utilize Deployed Design Patterns Effectively?”, *International Conference on Software Engineering, IEEE*, pp. 168-177, Minneapolis, Minnesota, 20-26 May 2007

Ng T. H., Cheung S.C., (2005): “Enhancing class commutability in the deployment of design patterns”, *Information and Software Technology*, Elsevier, 47 (12), pp. 797-804, September 2005

Perforce, Inc., (2004): “Perforce Software Configuration Management System”, *http://www.perforce.com*

Prechlet L., Unger B., Tichy W. F., Brossler P., Votta L. G. (2001): “A controlled experiment in maintenance comparing design patterns to simpler solutions”, *In IEEE Transactions on Software Engineering*, IEEE Computer Society, Vol. 27, pp. 1134-1144

Prechelt L., Unger-Lamprecht B., Philipsen M., Tichy W. F., (2002): “Two Controlled Experiments Assesing the Usefulness of Design Pattern Documentation in Program Maintenance”, *IEEE Transactions on Software Engineering*, IEEE, 28 (6), pp. 595-606, June 2002

Subramanyam, R., Krishnan, M.S. (2003): “Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Impications for Software Defects”. *IEEE Transactions on Software Engineering*. 29, 29—310

Schroter,A., Zeller, A., Zimmermann, T. (2006): ”Predicting Component Failures at Design Time”. In: *International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil

Schauer R., Keller R., (1998): “Pattern Visualization for Software Comprehension”, *Proceedings sixth International Workshop Program Comprehension*, pp. 4-12

Scientific Toolworks Inc., (2003): “Understand for C++”, *http:// www.scitools.com*

Sfetsos Panagiotis, Stamelos Ioannis, Angelis Lefteris, Deligiannis Ignatios, (2009): “An experimental investigation of personality types impact on pair effectiveness in pair programming”, *Empirical Software Engineering, Computer Science, Springer Netherlands*, vol. 14, pp. 187-226

Tassey, G. (2002): “The Economic Impacts of Inadequate Insrastructure for Software Testing”, *National Institute of Standards and Technology*.

TechExcel,(2004):”DevTrack Defect Tracking Tool*”, http:// www.techexcel.com/products/devtrack/dtoverview.html*.

Tsantalis N., A. Chatzigeorgiou, G. Stephanides and S.T. Halkidis, (2006): *In IEEE Transaction on Software Engineering,* "Design Pattern Detection using Similarity Scoring", IEEE Computer Society

Vokac M., Tichy W., D.I.K. Sjoberg, Arisholm E, Aldrin M., (2004) : “A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns – A Replication in a Real Programming Environment”, *Empirical Software Engineering, Springer, 9(3),* pp 149-195, September 2004

Vokac M. (2004) : “Defect Frequency and Design: An Empirical Study of Industrial Code”, *IEEE Transactions on Software Engineering , IEEE, 30(12),* pp. 904-917, December 2004

Vokac M., : “A tool for Recovering Design Patterns from C++ Code, and its Application in a Case Study”, *J. Object Technology*

Wendorff P. (2001): “Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project”, In *CSMR 2001, 5th European Conference on Software Maintenance and Reengineering, IEEE Computer Society*, Lisbon, Portugal, March 14-16, 2001, pp. 77