ΤΕΙ ΔΥΤΙΚΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑ

ΤΜΗΜΑ ΔΙΟΙΚΗΣΗ ΕΠΙΧΕΙΡΗΣΕΩΝ

Ανάλυση και Σχεδίαση Λογισμικού για την Αξιολόγηση Μαθήματος Διδασκαλίας από τους φοιτητές

Σπουδαστές:

Χατζηδάκης Ανδρέας

Ντερβίσι Λόρεντς

ΕΠΟΠΤΗΣ ΚΑΘΗΓΗΤΗΣ:

Χρήστος Πιερρακέας

Πάτρα

Φεβρουάριος 2014

Περιεχόμενα

Α. ΓΕΝΙΚΟ ΜΕΡΟΣ

1. Γνωσιολογικοί Στόχοι

1.1 Σκοπός - Περιεχόμενο

Σκοπός της παρούσας μελέτης περίπτωσης που αναλύεται υπό μορφή υπερκειμένου είναι η παρουσίαση των σταδίων ανάπτυξης μιας απλής εφαρμογής Αξιολόγησης Μαθήματος Διδασκαλίας από τους φοιτητές σύμφωνα με τη μεθοδολογία ICONIX. Η προσέγγιση έχει ως στόχο αφενός να περιγράψει θεωρητικά τα βήματα της μεθοδολογίας, εστιάζοντας στην καταγραφή των απαιτήσεων, την ανάλυση και την σχεδίαση του συστήματος. Αφετέρου, παρουσιάζεται η εφαρμογή της μεθοδολογίας στη συγκεκριμένη μελέτη περίπτωσης χρησιμοποιώντας ως εργαλείο τα αντίστοιχα διαγράμματα της Ενοποιημένης Γλώσσας Μοντελοποίησης (UML). Παράλληλα εξετάζεται η αξιολόγηση της ποιότητας του παραγόμενου σχεδίου, θέμα που απασχολεί την ομάδα ανάπτυξης οποιουδήποτε έργου λογισμικού. Τέλος, για την ολοκληρωμένη επίδειξη του συστήματος Αξιολόγησης Μαθήματος Διδασκαλίας από τους φοιτητές πραγματοποιείται μια συνοπτική επισκόπηση της ανάπτυξης γραφικής διασύνδεσης χρήστη.

1.2 Προσδοκώμενα Αποτελέσματα

Μετά την μελέτη του παραδείγματος ο αναγνώστης θα είναι σε θέση:

- να κατανοήσει τη χρησιμότητα μιας αντικειμενοστραφούς μεθοδολογίας ανάπτυξης λογισμικού όπως η ICONIX
- να πραγματοποιήσει με συστηματικό τρόπο την αντικειμενοστραφή ανάλυση και σχεδίαση για ένα πρόβλημα
- να χρησιμοποιεί τα διαγράμματα της Ενοποιημένης Γλώσσας Μοντελοποίησης για τη δημιουργία των κατάλληλων μοντέλων σε κάθε φάση της ανάπτυξης
- να αξιολογεί την ποιότητα της σχεδίασης
- να αναπτύσσει τη γραφική διασύνδεση χρήστη
- να αξιοποιεί τις δυνατότητες εργαλείων υποστήριξης της διαδικασίας ανάπτυξης

1.3 Λέξεις Κλειδιά

Αντικειμενοστραφής Ανάπτυξη Λογισμικού

Ενοποιημένη Γλώσσα Μοντελοποίησης (UML)

ICONIX

Ανάλυση και Σχεδίαση

Μετρικές Λογισμικού

Ποιότητα Σχεδίαση

Β. Εισαγωγή

2.1 Γενικά

Όλες σχεδόν οι χώρες εξαρτώνται σήμερα από σύνθετα συστήματα που βασίζονται σε υπολογιστές. Οι εθνικές υποδομές και υπηρεσίες στηρίζονται σε υπολογιστικά συστήματα, ενώ τα περισσότερα ηλεκτρικά προϊόντα περιλαμβάνουν έναν υπολογιστή και κάποιο λογισμικό ελέγχου. Η βιομηχανική παραγωγή και διανομή έχουν αυτοματοποιηθεί, όπως άλλωστε και το οικονομικό σύστημα. Κατά συνέπεια, η οικονομικά αποτελεσματική ανάπτυξη και συντήρηση του λογισμικού είναι ουσιώδης για τη λειτουργία των εθνικών οικονομιών και της διεθνούς οικονομίας.

Η τεχνολογία λογισμικού είναι ένας τεχνικός κλάδος που εστιάζεται στην οικονομικά αποτελεσματική ανάπτυξη συστημάτων λογισμικού υψηλής ποιότητας. Το λογισμικό είναι αφηρημένο και άυλο. Δεν περιορίζεται από υλικά, ούτε διέπεται από φυσικούς νόμους ή διαδικασίες. Το γεγονός αυτό κατά κάποιο τρόπο απλουστεύει την τεχνολογία λογισμικού, αφού δεν υφίστανται φυσικοί περιορισμοί για τις δυνατότητες του λογισμικού. Όμως αυτή έλλειψη φυσικών περιορισμών σημαίνει ότι το λογισμικό μπορεί εύκολα να γίνει εξαιρετικά πολύπλοκο και επομένως πολύ δυσνόητο.

Η έννοια της τεχνολογίας λογισμικού (Software engineering) προτάθηκε για πρώτη φορά το 1968,σε ένα συνέδριο που πραγματοποιήθηκε με σκοπό να συζητηθεί το φαινόμενο που την εποχή εκείνη ονόμαζαν ‘κρίση λογισμικού. Η κρίση αυτή ήταν άμεση συνέπεια της εμφάνισης νέου υπολογιστικού υλικού βασισμένο σε ολοκληρωμένα κυκλώματα. Η ισχύς του εξοπλισμού αυτού έκανε δυνατές ορισμένες υπολογιστικές εφαρμογές που μέχρι τότε ήταν ανέφικτες. Το λογισμικό που προέκυπτε ήταν κατά πολλές τάξεις μεγέθους μεγαλύτερο και πιο περίπλοκο από τα προηγούμενα συστήματα λογισμικού.

Οι πρώτες εμπειρίες από την κατασκευή τέτοιων συστημάτων έδειξαν ότι η ανάπτυξη λογισμικού με εμπειρικές και μη τυποποιημένες διαδικασίες ήταν ανεπαρκής. Σημαντικά έργα λογισμικού συχνά καθυστερούσαν χρόνια ολόκληρα. Το παραγόμενο λογισμικό κόστιζε πολύ περισσότερο από τις προβλέψεις, ήταν αναξιόπιστο, δύσκολο να συντηρηθεί, και έχει χαμηλή απόδοση. Η ανάπτυξη λογισμικού βρισκόταν σε κρίση. Ενώ το κόστος του υλικού κατρακυλούσε, το κόστος του λογισμικού αυξανόταν γρήγορα. Υπήρχε ανάγκη για νέες τεχνικές και μεθόδους για τον έλεγχο της πολυπλοκότητας που είναι έμφυτη στα μεγάλα συστήματα λογισμικού.

Τέτοιου είδους τεχνικές έχουν γίνει πλέον μέρος της τεχνολογίας λογισμικού και χρησιμοποιούνται ευρύτατα σήμερα. Ωστόσο, όπως έχει αυξηθεί η δυνατότητά μας να

παράγουμε λογισμικό, αντίστοιχα έχει αυξηθεί και η πολυπλοκότητα των συστημάτων λογισμικού που χρειαζόμαστε. Οι νέες τεχνολογίες που προκύπτουν από τη σύγκλιση υπολογιστικών και τηλεπικοινωνιακών συστημάτων, καθώς και σύνθετων γραφικών διασυνδέσεων χρήστη, θέτουν νέες απαιτήσεις στους μηχανικούς λογισμικού. Καθώς πολλές εταιρείες δεν εφαρμόζουν ακόμα αποτελεσματικά τις τεχνικές της τεχνολογίας λογισμικού, πάρα πολλά έργα εξακολουθούν να παράγουν λογισμικό λειτουργικά αναξιόπιστο, με καθυστερήσεις στο χρόνο παράδοσης, και με υπερβάσεις του προϋπολογισμού.

Έχει σημειωθεί τεράστια πρόοδο από το 1968, και ότι η ανάπτυξη της τεχνολογίας λογισμικού έχει βελτιώσει αισθητά το λογισμικό που διαθέτουμε σήμερα. Έχουμε επιτύχει πολύ καλύτερη κατανόηση των δραστηριοτήτων που εμπεριέχει η ανάπτυξη λογισμικού. Έχουμε αναπτύξει αποτελεσματικές μεθόδους εξαγωγής προδιαγραφών, σχεδιασμού, και υλοποίησης λογισμικού, ενώ νέες σημειογραφίες και εργαλεία μειώνουν την προσπάθεια που απαιτείται για την παραγωγής μεγάλων και σύνθετων συστημάτων.

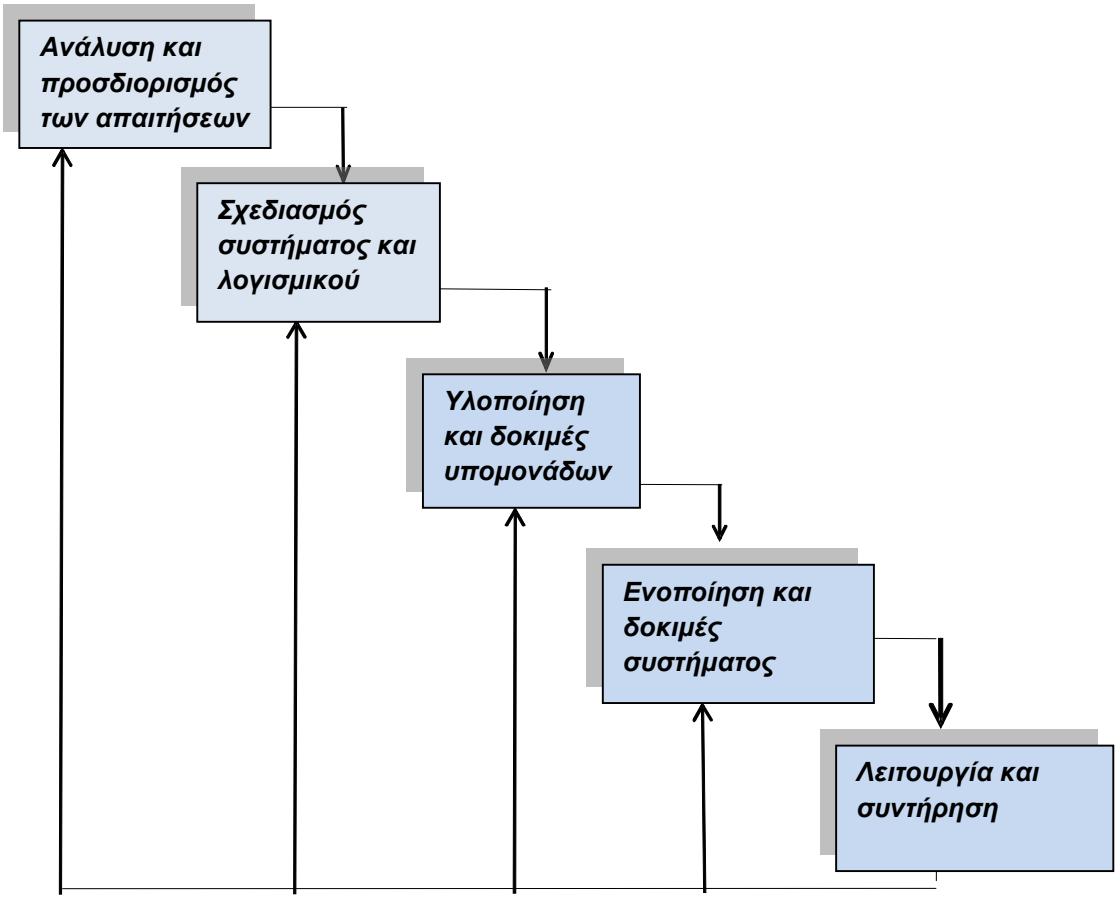
Οι μηχανικοί λογισμικού μπορούν δικαιωματικά να υπερηφανεύονται για τα επιτεύγματά τους, Χωρίς σύνθετα συστήματα λογισμικού δε θα είχαμε εξερευνήσει το διάστημα, δε θα είχαμε το Διαδίκτυο και τις σύγχρονες τηλεπικοινωνίες, και όλοι οι τρόποι ταξιδιού θα ήταν πιο επικίνδυνοι και ακριβοί. Η τεχνολογία λογισμικού έχει προσφέρει πολλά και η συνεισφορά της στον 21ο αιώνα θα είναι ακόμα μεγαλύτερη.

2.2 Κυριότερες Μεθοδολογίες

Σήμερα γνωρίζουμε ότι δεν υπάρχει μία και μοναδική προσέγγιση για την ανάπτυξη λογισμικού. Η μεγάλη ποικιλία των διαφόρων τύπων συστημάτων και των οργανισμών που χρησιμοποιούν αυτά τα συστήματα σημαίνει ότι χρειαζόμαστε μια αντίστοιχη ποικιλία προσεγγίσεων για την ανάπτυξη λογισμικού. Υπάρχουν όμως ορισμένες θεμελιώδεις έννοιες οργάνωσης των διαδικασιών και των συστημάτων στις οποίες βασίζονται όλες αυτές οι τεχνικές, και οι έννοιες αυτές αποτελούν την ουσία της τεχνολογίας λογισμικού.

2.3 Μοντέλο Καταρράκτη

Το πρώτο μοντέλο διαδικασίας ανάπτυξης λογισμικού που δημοσιεύτηκε προήλθε από γενικότερες πρακτικές της τεχνολογίας συστημάτων (Royce, 1970), όπως φαίνετε και στο Εικόνα.1. Λόγω της μετάπτωσης από την μια φάση στην άλλη, το μοντέλο αυτό έγινε γνωστό ως μοντέλο καταρράκτη (waterfall model) ή ως κύκλος ζωής λογισμικού (Software life cycle). Τα κύρια στάδια αυτού του μοντέλου αντιστοιχούν στις θεμελιώδεις δραστηριότητες ανάπτυξης:



Εικόνα.1(waterfall model)

2.3.1 Στάδια μοντέλου καταρράκτη

Ανάλυση και προσδιορισμός των απαιτήσεων: Οι υπηρεσίες, οι περιορισμοί και οι στόχοι του συστήματος προσδιορίζονται μέσω συζήτησης με τους χρήστες. Στη συνέχεια καθορίζονται λεπτομερώς και χρησιμεύουν ως προδιαγραφές του συστήματος.

Σχεδιασμός συστήματος και λογισμικού: Η διαδικασία του σχεδιασμού του συστήματος χωρίζει τις απαιτήσεις σε συστήματα υλικού και λογισμικού. Αυτό καταλήγει σε μια συνολική αρχιτεκτονική του συστήματος. Ο σχεδιασμός λογισμικού περιλαμβάνει τον προσδιορισμό και την περιγραφή των θεμελιωδών αφηρημένων τμημάτων του συστήματος, καθώς και των σχέσεων μεταξύ τους.

Υλοποίηση και δοκιμές υποομάδων : Κατά το στάδιο αυτό, ο σχεδιασμός του λογισμικού υλοποιείται ως ένα σύνολο προγραμμάτων ή προγραμματιστικών υποομάδων.

Οι δοκιμές υποομάδων συνίσταται στην επαλήθευση ότι κάθε υποομάδα ικανοποιεί τις προδιαγραφές της.

Ενοποίηση και δοκιμές συστήματος: Οι μεμονωμένες προγραμματιστικές υποομάδες ή προγράμματα ενοποιούνται και δοκιμάζονται ως ενιαίο σύστημα, για να διασφαλιστεί ότι οι απαιτήσεις λογισμικού έχουν εκπληρωθεί. Μετά τη δοκιμή, το σύστημα λογισμικού παραδίδετε στον πελάτη.

Λειτουργία και συντήρηση: Κανονικά (αλλά όχι απαραίτητα) η φάση αυτή είναι η ποιο μακρόχρονη φάση του κύκλου ζωής. Το σύστημα εγκαθιστάτε και τίθεται σε χρήση. Η συντήρηση συνιστάτε στην διόρθωση σφαλμάτων που δεν εντοπίστηκαν στα προηγούμενα στάδια του κύκλου ζωής τη βελτίωση της υλοποίησης των μονάδων του συστήματος, όπως και τη βελτίωση των υπηρεσιών του συστήματος καθώς παρουσιάζονται νέες απαιτήσεις.

Γενικά το αποτέλεσμα της κάθε φάσης είναι ένα ή περισσότερα εγκεκριμένα έγγραφα. Η επόμενη φάση δεν πρέπει να ξεκινήσει μέχρι να τελειώσει η προηγούμενη. Στην πράξη, όμως τα στάδια αυτά επικαλύπτονται και ανταλλάσσουν πληροφορίες. Κατά τον σχεδιασμό εντοπίζονται προβλήματα των απαιτήσεων, κατά την συγγραφή κώδικα λογισμικού εντοπίζονται προβλήματα του σχεδιασμού κ.ο.κ

Η διαδικασία παραγωγής λογισμικού δεν ακολουθεί ένα απλό γραμμικό μοντέλο αλλά εμπεριέχει μια ακολουθία κυκλικών επαναλήψεων των δραστηριοτήτων ανάπτυξης. Εξαιτίας του κόστους παραγωγής και έγκρισης των εγγράφων, οι κυκλικές επαναλήψεις είναι δαπανηρές και εμπεριέχουν πολλή δουλειά. Γι' αυτόν τον λόγο, μετά από ένα μικρό αριθμό επαναληπτικών κύκλων, συνηθίζεται να «παγώνουν» ορισμένα τμήματα ανάπτυξης, όπως η εξαγωγή προδιαγραφών, και η διαδικασία συνεχίζεται με επόμενα στάδια. Κάποια προβλήματα αφήνονται για να επιλυθούν αργότερα, αγνοούνται ή παρακάμπτονται με την βοήθεια κώδικα. Αυτό το πρόωρο «πάγωμα» των απαιτήσεων μπορεί να έχει συνέπεια το σύστημα να μην κάνει αυτά που θέλει ο χρήστης.

Μπορεί επίσης να οδηγήσει σε σύστημα με κακή δομή, εφόσον προβλήματα του σχεδιασμού έχουν παρακαμφθεί με τεχνάσματα της υλοποίησης. Στην τελική φάση του κύκλου ζωής (την φάση της λειτουργίας και συντήρησης), το λογισμικό τίθεται σε χρήση. Διαπιστώνονται σφάλματα και παραλείψεις των αρχικών απαιτήσεων του λογισμικού. Προκύπτουν σφάλματα των προγραμμάτων και του σχεδιασμού, και διαπιστώνετε η ανάγκη για νέες λειτουργίες και δυνατότητες. Γι' αυτούς τους λόγους, το σύστημα πρέπει να

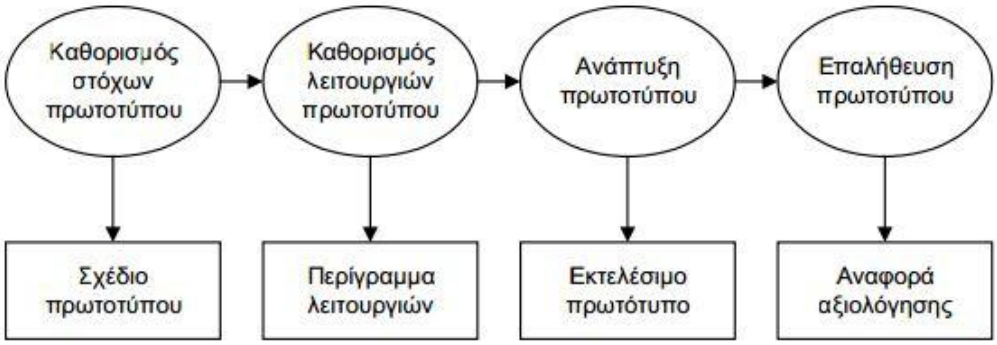
εξελιχθεί για να παραμείνει χρήσιμο. Η πραγματοποίηση αυτών τον αλλαγών (η συντήρηση λογισμικού) μπορεί να απαιτεί την επανάληψη προηγούμενων σταδίων της διαδικασίας.

Το πλεονέκτημα του μοντέλου καταρράκτη είναι ότι παράγει τεκμηρίωση σε κάθε φάση και ότι ταιριάζει με άλλα μοντέλα που χρησιμοποιούνται σε τεχνικά έργα. Το σημαντικότερο πρόβλημα είναι ο άκαμπτος διαμερισμός του έργου σε ξεχωριστά στάδια. Απαιτεί να λαμβάνονται αποφάσεις σε πρώιμα στάδια της διαδικασίας, γεγονός που μπορεί να δυσχεράνει την ανταπόκριση σε αλλαγές των απαιτήσεων του πελάτη. Επομένως, το μοντέλο καταρράκτη πρέπει να χρησιμοποιείται μόνο όταν οι απαιτήσεις είναι καλά κατανοητές και δεν είναι πιθανόν να αλλάξουν ριζικά κατά την ανάπτυξη του συστήματος. Από την άλλη, όμως το μοντέλο καταρράκτη αντανακλά το είδος μοντέλου διαδικασίας που χρησιμοποιείται σε άλλα τεχνικά έργα. Γι' αυτόν το λόγο, διαδικασίες παραγωγής λογισμικού που βασίζονται σε αυτήν την προσέγγιση εξακολουθούν να χρησιμοποιούνται γενικότερα στην ανάπτυξη λογισμικού, ιδιαίτερα όταν το έργο λογισμικού αποτελεί μέρος ενός μεγαλύτερου έργου κατασκευής συστημάτων.

2.4 Μοντέλο Πρωτοτυποποίησης

Ένας τρόπος να δούμε την προτυποποίηση είναι ως μια τεχνική για μείωση του κινδύνου. Ο πιο σημαντικός κίνδυνος στην ανάπτυξη λογισμικού είναι τα λάθη και περισσότερο οι παραλείψεις που προκύπτουν από μη σαφείς απαιτήσεις των χρηστών για το τελικό σύστημα. Το κόστος της διόρθωσης αυτών των λαθών και παραλείψεων σε επόμενα στάδια μπορεί να είναι πολύ υψηλό. Είναι προφανές ότι η δημιουργία ενός πρωτοτύπου μπορεί να μειώσει τον αριθμό των προβλημάτων των απαιτήσεων και ως εκ τούτου να μειώσει το συνολικό κόστος ανάπτυξης

Μια σχηματική αναπαράσταση της διαδικασίας ανάπτυξης ενός πρωτοτύπου φαίνεται στην εικόνα 2



Εικόνα 2. Διαδικασίας ανάπτυξης ενός πρωτοτύπου

Αρχικά θα πρέπει να καθοριστούν επακριβώς οι στόχοι του πρωτότυπου. Το πρωτότυπο σύστημα μπορεί να αφορά τη διαφάνεια χρήστη ή να περιέχει τις λειτουργίες εκείνες που θεωρούνται περισσότερο κρίσιμες. Είναι προφανές ότι ένα πρωτότυπο δεν μπορεί να καλύπτει όλες τις απαιτήσεις του σωστότατος. Για τον λόγο αυτό κάθε φορά θα πρέπει να ορίζονται πλήρως οι απαιτήσεις που αυτό θα καλύπτει, αλλιώς τελικά μπορεί να μην λάβουμε τα πλεονεκτήματα που μας προσφέρει η μέθοδος αυτή.

Το επόμενο στάδιο αφορά στην επιλογή των λειτουργιών εκείνων από το σύνολο των λειτουργιών του τελικού συστήματος που θα συμπεριληφθούν στο προς δημιουργία πρωτότυπο (άρα και αυτών που δεν θα συμπεριληφθούν). Ο λόγος για την επιλογή αυτή είναι, γιατί προφανώς έχει πολύ υψηλό κόστος εάν το πρωτότυπο δημιουργηθεί με όλες τις λειτουργίες του τελικού συστήματος. Βέβαια θα μπορούσε να αποφασιστεί να περιλαμβάνονται όλες οι λειτουργίες που έχει αποφασιστεί αλλά σε μειωμένο επίπεδο (πχ. χωρίς διαχείριση λαθών).

Η τελευταία φάση, μετά την ανάπτυξη του πρωτοτύπου, είναι η επαλήθευση του πρωτότυπου και είναι ίσως η πιο σημαντική φάση. Θα πρέπει να καταγραφούν συμπεράσματα για το πως νιώθουν οι χρήστες με το σύστημα, αν γίνεται κατανοητό το περιβάλλον και η λειτουργία του και να βρεθούν τυχόν λάθη και προβλήματα. Τα πλεονεκτήματα της χρήσης πρωτότυπου είναι ότι ανακαλύπτονται και διορθώνονται:

- Παρεξηγήσεις μεταξύ των χρηστών και των δημιουργών.
- Παραλειπόμενες υπηρεσίες στο σύστημα
- Δυσκολίες στη χρήση

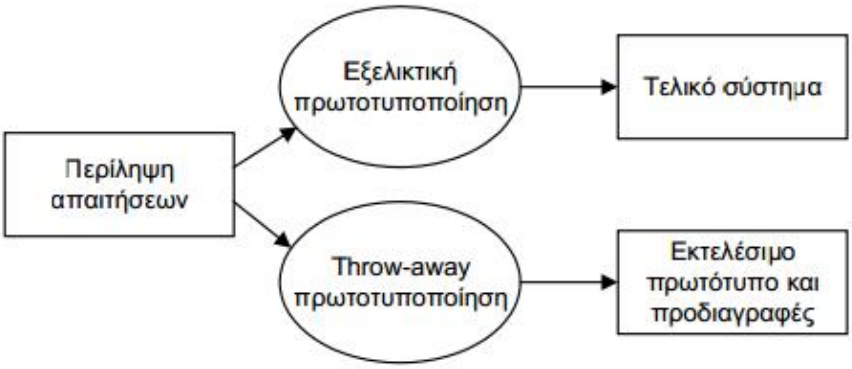
• Ασυνέχειες και κενά στις προδιαγραφές

Η προτυποποίηση μπορεί να χρησιμοποιηθεί και για άλλους σκοπούς, όπως στην εκπαίδευση των χρηστών, δηλαδή το πρωτότυπο να χρησιμοποιηθεί ως εκπαιδευτικό εργαλείο για την εκμάθηση του τελικού συστήματος, αλλά ταυτόχρονα είναι και ένας τρόπος μείωσης του κινδύνου, αφού περιορίζονται τα λάθη και οι παραλείψεις. Αν τα λάθη αφεθούν για διόρθωση στις τελευταίες φάσεις του κύκλου ζωής το κόστος αυξάνεται κατακόρυφα.

Το βασικό μειονέκτημα του μοντέλου της προτυποποίησης είναι ότι το κόστος ανάπτυξής του αποτελεί ένα μεγάλο μέρος του συνολικού κόστους του συστήματος που αναπτύσσεται. Πολλές φορές είναι οικονομικά πιο συμφέρον να μεταβληθεί το τελικό προϊόν από το να δημιουργηθεί ένα πρωτότυπο.

Είναι προφανές, ότι είναι πολύ δύσκολο να προβλεφθεί ποιες ακριβώς δυσκολίες θα αντιμετωπίσει ο τελικός χρήστης από την καθημερινή χρήση ενός νέου συστήματος λογισμικού. Ιδιαίτερα εάν αναφερόμαστε σε μεγάλα συστήματα λογισμικού η δυσκολία αυτή μπορεί να καταφανεί μόνο όταν το ολοκληρωμένο σύστημα αναπτυχθεί και τεθεί σε λειτουργία. Για να αντιμετωπιστεί αυτή η δυσκολία μπορεί να χρησιμοποιηθεί η εξελικτική (evolutionary) προτυποποίησης. Κατά τη προσέγγιση αυτή δημιουργείται μια περιορισμένη (ατελής) έκδοση του συστήματος πάνω στην οποία γίνονται οι διορθώσεις και οι προσθήκες καθώς απαιτήσεις είτε διευκρινίζονται είτε ανακαλύπτονται είτε βελτιώνονται μέχρι να καταλήξουμε σε μια έκδοση που να είναι επαρκής και ικανοποιητική.

Το βασικό πρόβλημα του μοντέλου της εξελικτικής Πρωτοτυποποίησης είναι ότι με τις συνεχείς διορθωτικές και προσθετικές παρεμβολές παράγεται «μπλεγμένος» κώδικας (spaghetti code) που είναι πολύ δύσκολα συντηρήσιμος ιδιαίτερα όταν δεν τηρούνται κανόνες ποιότητας λογισμικού Εναλλακτικά και προκειμένου να αποφύγουμε το φαινόμενο του μπλεγμένου κώδικα’, θα μπορούσε να δημιουργηθεί ένα πρωτότυπο το οποίο θα είχε αποκλειστικό σκοπό την αποσαφήνιση των απαιτήσεων και την παροχή πληροφοριών για την εκτίμηση του κινδύνου του τελικού συστήματος. Μετά την αξιολόγηση το πρωτότυπο σύστημα ‘πετιέται’ και δεν χρησιμοποιείται για την ανάπτυξη του συστήματος (throw-away prototype) (Εικόνα 3).



Εικόνα 3. Throw-away prototype

2.5 Ενοποιημένη Διεργασία (Unified Process)

Η ενοποιημένη διεργασία είναι μια αντικειμενοστραφή προσέγγιση στον τρόπο σχεδίασης και ανάπτυξης συστημάτων λογισμικού. Μια διαδικασία ανάπτυξης λογισμικού περιγράφει μια προσέγγιση στο χτίσιμο, στην ανάπτυξη, και ενδεχομένως στη συντήρηση του λογισμικού. Η ενοποιημένη διαδικασία έχει προκύψει ως δημοφιλής διαδικασία ανάπτυξης λογισμικού για αντικειμενοστραφή συστήματα..

Η ενοποιημένη διαδικασία (UP) συνδυάζει κοινώς αποδεκτές καλύτερες πρακτικές, όπως έναν επαναληπτικό κύκλο ζωής και μια κίνδυνο-καθοδηγημένη ανάπτυξη, αλλά και μια συνεκτική και καλά τεκμηριωμένη περιγραφή του συστήματος που βρίσκεται υπό ανάπτυξη. Συνεπώς, χρησιμοποιείται στη συνεχεία ως παράδειγμα μέσα στο οποίο γίνεται η εισαγωγή στην αντικειμενοστραφή ανάλυση και σχεδίαση

Η Ενοποιημένη Διεργασία (UP) χρησιμοποιεί μερικές από τις καλύτερες πρακτικές για την ανάλυση και σχεδίαση συστημάτων λογισμικού. Η σημαντικότερη όμως πρακτική είναι η επαναληπτική ανάπτυξη. Η ανάπτυξη οργανώνεται σε μια σειρά από σύντομα, καθορισμένου μήκους (παραδείγματος χάριν, τεσσάρων εβδομάδων) μίνι-projects που ονομάζονται επαναλήψεις. Το αποτέλεσμα κάθε μιας από αυτές τις επαναλήψεις είναι ένα δοκιμασμένο, ολοκληρωμένο και εκτελέσιμο σύστημα. Κάθε επανάληψη περιλαμβάνει την ανάλυση απαιτήσεων, τη σχεδίαση, την εφαρμογή και τις δραστηριότητες δοκιμής.

Ο επαναληπτικός τρόπος ανάπτυξης είναι βασισμένος στη διαδοχική διεύρυνση και αύξησης της ποιότητας του συστήματος μέσω των πολλαπλών επαναλήψεων, με κυκλική ανατροφοδότηση και αναπροσαρμογή. Το σύστημα αυξάνεται προοδευτικά με την πάροδο του χρόνου, από επανάληψη σε επανάληψη, έτσι αυτή η προσέγγιση είναι επίσης γνωστή ως επαναληπτική και αυξητική ανάπτυξη.

Το αποτέλεσμα κάθε επανάληψης είναι ένα εκτελέσιμο αλλά ελλιπές σύστημα δεν είναι έτοιμος να παραδώσει στην παραγωγή. Το σύστημα μπορεί να μην είναι έτοιμο να δοθεί στην παραγωγή πριν την ολοκλήρωση πολλών επαναλήψεων παραδείγματος χάριν, 10 ή 15 επαναλήψεις. Το αποτέλεσμα μιας επανάληψης δεν είναι κάτι πειραματικό ή ένα πρωτότυπο που είναι για πέταμα, επίσης η επαναληπτική ανάπτυξη δεν είναι διαμόρφωση πρωτοτύπου. Θα μπορούσε να πει κανείς ότι το αποτέλεσμα είναι ένα ολοκληρωμένο υποσύνολο του τελικού συστήματος. Αν και, γενικά, κάθε επανάληψη αντιμετωπίζει τις νέες απαιτήσεις και αυξητικά επεκτείνει το σύστημα, μπορεί περιστασιακά σε μια επανάληψη να αναθεωρηθεί το υπάρχον λογισμικό και να βελτιωθεί. παραδείγματος χάριν, μια επανάληψη μπορεί να εστιάσει στη βελτίωση της απόδοσης ενός υποσυστήματος, παρά στην επέκταση του με τα νέα χαρακτηριστικά γνωρίσματα . Μια άλλη κεντρική ιδέα στην Ενοποιημένη Διεργασία (UP) είναι η χρήση των αντικειμενοστραφών τεχνολογιών κάτι που περιλαμβάνει αντικειμενοστραφή ανάλυση και σχεδίαση αλλά και αντικειμενοστραφή προγραμματισμό.

Μερικές πρόσθετες καλύτερες πρακτικές και βασικές έννοιες στην Ενοποιημένη Διεργασία (UP) περιλαμβάνουν:

• Αντιμετώπιση των υψηλών κινδύνων και μεγάλης αξίας ζητήματα στις πρόωρες επαναλήψεις

• Συνεχής όχληση των χρηστών και γενικά των εμπλεκομένων για αξιολόγηση, ανατροφοδότηση, και επαναπροσδιορισμό των απαιτήσεων

• Χτίσιμο μια συνεκτική, αρχιτεκτονική πυρήνων στις πρώτες επαναλήψεις

• Συνεχής έλεγχος της ποιότητας εξετάστε νωρίς, συχνά, και ρεαλιστικά

• Εφαρμογή των περιπτώσεων χρήσης

• Οπτική διαμόρφωση του λογισμικού (με τη UML)

• Προσεκτική διαχείριση των απαιτήσεων

• Αίτημα αλλαγής πρακτικής και διαχείριση διαμόρφωσης.

Ένα πλάνο ανάπτυξης μιας εφαρμογής βασισμένο στην Ενοποιημένη Διεργασία (UP) οργανώνει την εργασία και τις επαναλήψεις σε τέσσερις σημαντικές φάσεις:

1. Αρχική μελέτη (Inception) Κατά προσέγγιση όραμα έναρξης, επιχειρησιακή περίπτωση, πεδίο, ασαφείς εκτιμήσεις.

2. Επεξεργασία (Elaboration) καθαρισμένο όραμα, επαναληπτική εφαρμογή του πυρήνα αρχιτεκτονικής, ψήφισμα των υψηλών κινδύνων, προσδιορισμός των περισσότερων απαιτήσεων και πεδίο, ρεαλιστικότερες εκτιμήσεις.

3. Κατασκευή (Construction) επαναληπτική εφαρμογή του υπόλοιπου χαμηλότερου κινδύνου και ευκολότερα στοιχεία, και προετοιμασία για την επέκταση.

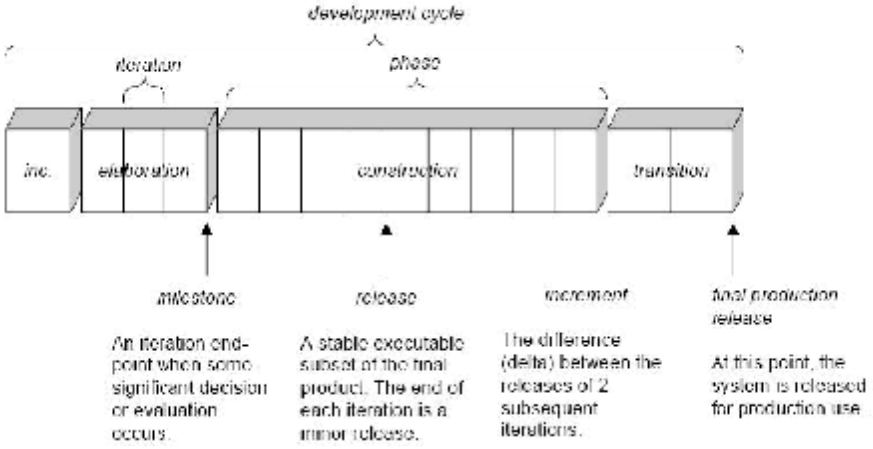
4. Μετάβαση (Transition) δοκιμές του λογισμικού, επέκταση.

Αυτό δεν είναι ο παλαιός «καταρράκτης» ή ο διαδοχικός κύκλος ζωής όπου πρώτα πρέπει να καθοριστούν όλες οι απαιτήσεις, και μετά να γίνει όλος ή το μεγαλύτερο μέρος του σχεδιασμού.

Η έναρξη δεν είναι η φάση των απαιτήσεων, είναι ένα είδος φάσης ανάλυσης του τι μπορεί να γίνει, όπου γίνεται ακριβώς όση έρευνα χρειάζεται για να παρθεί μια απόφαση αν η ομάδα θα συνεχίσει την ανάπτυξη ή θα σταματήσει.

Ομοίως, η επεξεργασία δεν είναι η φάση απαιτήσεων ή η φάση σχεδίασης, αλλά είναι η φάση όπου η κεντρική αρχιτεκτονική εφαρμόζεται επαναληπτικά, και μετριάζονται τα ζητήματα υψηλού κινδύνου.

Η Εικόνα 4 επεξηγεί τους συνήθεις όρους που έχουν σχέση με την αλληλουχία των ενεργειών στην Ενοποιημένη Διεργασία (UP) . Παρατηρήστε ότι ένας κύκλος ανάπτυξης αποτελείται από πολλές επαναλήψεις.



Εικόνα 4. (Κύκλος ανάπτυξης Ενοποιημένης Διεργασίας)

2.6 Ακραίος Προγραμματισμός (eXtreme Programming-XP)

Ο ακραίος προγραμματισμός είναι η πιο δημοφιλής από τις Ευέλικτες Μεθόδους και δημιουργήθηκε στα τέλη της δεκαετίας του '90 από τον Kent Beck. Είναι μία πειθαρχημένη προσέγγιση που υποστηρίζει την ταχύτερη ολοκλήρωση του έργου σε μικρούς επαναληπτικούς κύκλους, επιτρέποντας στους υπεύθυνους ανάπτυξης να ανταποκρίνονται στις μεταβαλλόμενες απαιτήσεις του πελάτη καθ' όλη τη διάρκεια του κύκλου παραγωγής (Beck, 2000). Βασίζεται σε τέσσερεις αρχές και δώδεκα πρακτικές οι οποίες βελτιώνουν και απλοποιούν την ανάπτυξη. Οι τέσσερεις αρχές είναι:

• Επικοινωνία (communication). Η κοινή αντιμετώπιση των προβλημάτων απαιτεί κοινή κατανόηση μέσω διαπροσωπικής επικοινωνίας.

• Απλότητα (simplicity). Η ομάδα εκτελεί πάντα το πιο απλό σχέδιο που πιθανόν θα δουλέψει. Συνεχώς απλοποιεί και βελτιώνει τον κώδικα με αναδόμηση. Όταν ο κώδικας είναι απλός, είναι περισσότερο κατανοητός και καλύτερα τεκμηριωμένος, ενώ τα λάθη είναι πολύ λιγότερα.

• Ανατροφοδότηση (feedback). Η τακτική παράδοση λογισμικού στον πελάτη, σχεδόν ανά εβδομάδα, βοηθά στην συνειδητοποίηση προβλημάτων πριν την ολοκλήρωση του έργου.

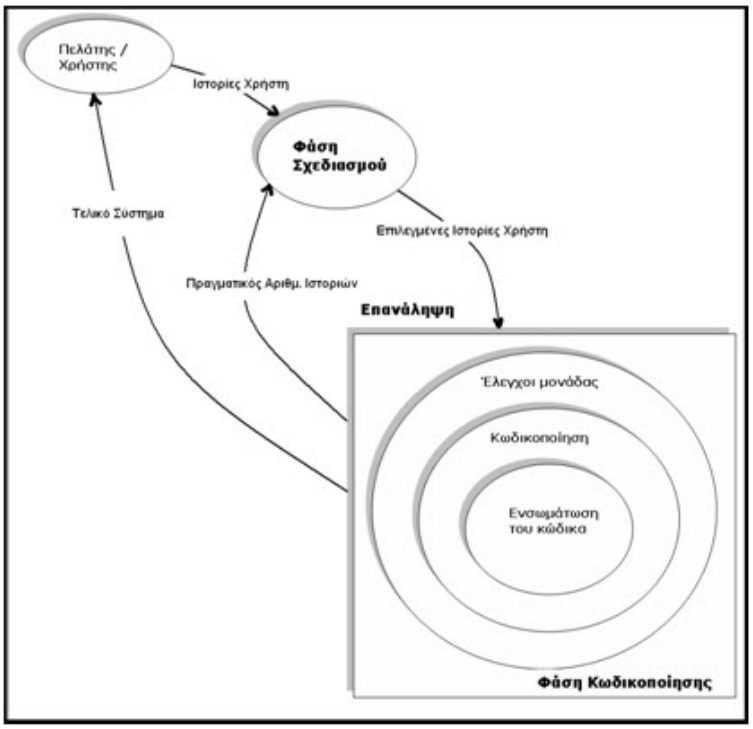
- Κουράγιο (courage). Το κουράγιο και η καλή ψυχολογία είναι δύο σημαντικοί παράγοντες. Επιτυχημένες ομάδες ανάπτυξης λογισμικού συχνά ενεργούν στην κόψη του ξυραφιού, καθώς πρέπει να εργάζονται πολύ γρήγορα, δίχως όμως να χάνουν τον έλεγχο. Αυτό σημαίνει πως συχνά αποτυγχάνουν. Αν οι προγραμματιστές φοβούνται την αποτυχία, θα ενεργούν πολύ αργά. Σε μια αποτυχία η ομάδα πρέπει να βρίσκει το κουράγιο να συνεχίζει ή να ξεκινά από την αρχή.

Οι αξίες αυτές διαμορφώνουν ένα διαφορετικό πλαίσιο προσέγγισης της ανάπτυξης λογισμικού. Στο πλαίσιο αυτό τα στελέχη της ανάπτυξης επικοινωνούν μεταξύ τους και με τους πελάτες μέσω συνεχούς διαπροσωπικής επικοινωνίας, εξασφαλίζοντας την άμεση ανατροφοδότηση. Χρησιμοποιούν απλό σχεδιασμό διατηρώντας τον κώδικα απλό και βρίσκουν το κουράγιο να συνεχίσουν μετά από κάποια αποτυχία. Το πλαίσιο αυτό συμπληρώνεται με δώδεκα πρακτικές που αλληλοεπιδρούν μεταξύ τους και πρέπει να εφαρμόζονται ενιαία σε όλες τις φάσεις της ανάπτυξης. Αυτές είναι:

- Το παιχνίδι του σχεδιασμού (planning game)
- Προγραμματισμός σε ζεύγη (pair programming)
- Έλεγχοι πριν την κωδικοποίηση (test-first-design)
- Πρότυπα κωδικοποίησης (coding standards)
- Απλός σχεδιασμός (simple design)
- Ανακατασκευή κώδικα (refactoring)
- Διαρκείς ενσωματώσεις στον κώδικα (continuous integration)
- Συλλογική ιδιοκτησία του κώδικα (collective code ownership)
- Διαρκής παρουσία πελάτη (on-site customer)
- Μικρές εκδόσεις (small releases)
- Υποφερτός ρυθμός εργασίας (sustainable pace)
- Συνολική εικόνα του συστήματος (system metaphor)

Οι αξίες μαζί με τις πρακτικές τίθενται σε καθημερινή δοκιμασία στην διαδικασία ανάπτυξης του λογισμικού. Το έργο λογισμικού ξεκινά με μία φάση διερεύνησης, όπου οι πελάτες παραθέτουν στους προγραμματιστές και στον διευθυντή του έργου τις απαιτήσεις του συστήματος υπό μορφή γραπτών κειμένων - ιστοριών χρήστη (user stories). Κάθε μία από αυτές τις ιστορίες χρήστη περιγράφει μία απλή λειτουργία του συστήματος και διασπάται από την ομάδα σε τμήματα εργασίας, τις προγραμματιστικές διεργασίες (tasks). Για κάθε μία από αυτές τις διεργασίες εκτιμάται ο χρόνος υλοποίησης χρησιμοποιώντας σαν

μέτρο μέτρησης τις ιδανικές προγραμματιστικές μέρες (ideal programming days). Αν κάποια ιστορία χρήστη είναι πολύπλοκη, τότε διασπάται σε μικρότερες ιστορίες. Στην συνέχεια οι ιστορίες χρήστη ταξινομούνται σύμφωνα με πιά προτεραιότητα έχουν στην ανάπτυξη και ανάλογα με τη βαρύτητα που προσδίδουν σε αυτές οι πελάτες. Από την αρχή της διερευνητικής φάσης ξεκινά το 'παιχνίδι του σχεδιασμού' σύμφωνα με το οποίο θα αποφασιστεί από την ομάδα το πλήθος των επί μέρους εκδόσεων (small releases) (3-6 μήνες) και των επί μέρους επαναλήψεων (iterations) (1-3 εβδομάδες) κάθε έκδοσης (βλ. Εικόνα. 5).



Εικόνα 5. Προγραμματιστικές διεργασίες (tasks)

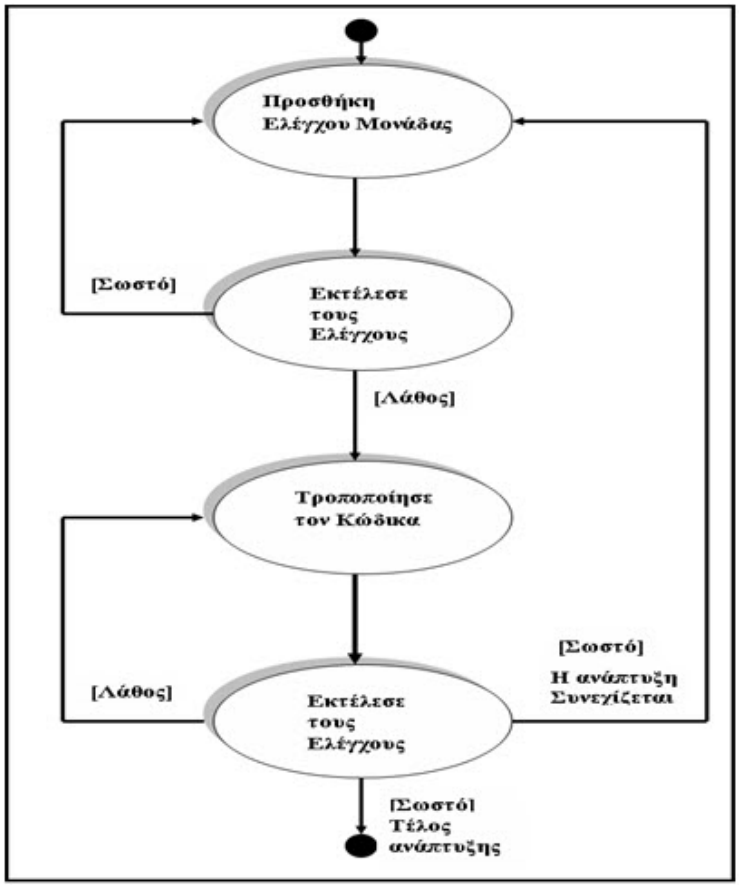
Η καθοδηγούμενη από ελέγχους ανάπτυξη (Test Driven Development) αποτελεί το κέντρο βάρους της ανάπτυξης λογισμικού του ακραίου προγραμματισμού. Αυτή η προσέγγιση έγινε αποδεκτή ακόμη και από τους θιασώτες των παραδοσιακών μεθόδων και υλοποιείται από δύο σημαντικές πρακτικές που ήδη εφαρμόζονται και σαν μεμονωμένες πρακτικές στην βιομηχανία λογισμικού: του προγραμματισμού σε ζεύγη προγραμματιστών και τους ελέγχους πριν την κωδικοποίηση.

Προγραμματισμός σε ζεύγη σημαίνει ότι η ομάδα χωρίζεται σε ζεύγη προγραμματιστών και το κάθε ζεύγος αναλαμβάνει την ολοκλήρωση μιας διεργασίας. Κάθε ζεύγος προγραμματιστών χρησιμοποιεί έναν υπολογιστή. Ο ένας προγραμματιστής ο οδηγός (driver) κωδικοποιεί και ο άλλος ο πλοηγός (navigator) ελέγχει τον κώδικα συνεχώς για την

αποφυγή λαθών και προτείνει λύσεις για την ανάπτυξη. Οι προγραμματιστές αλλάζουν συνεχώς ρόλους αλλά και συνεργάτες, όταν το απαιτούν κάποιες διεργασίες. Τα πλεονεκτήματα του προγραμματισμού σε ζεύγη είναι η ταχύτερη ανάπτυξη του κώδικα και ο περιορισμός των λαθών, που σημαίνει ποιοτικότερο κώδικα. Επιπλέον η συχνή αλλαγή συνεργάτη βοηθά στη μεταφορά της εμπειρίας και γνώσης σε όλα τα μέλη της ομάδας. Τα πλεονεκτήματα του προγραμματισμού σε ζεύγη καθώς και η σωστή εφαρμογής της πρακτικής αποτελούν θέματα έρευνας τα τελευταία χρόνια.

Οι έλεγχοι του κώδικα είναι συνεχείς, ποιοτικοί και αυτοματοποιημένοι, παρέχοντας μία δικλείδα ασφαλείας σε κατασκευαστές και πελάτες.

Πριν από κάθε κομμάτι κώδικα γράφεται πρώτα το αντίστοιχο κομμάτι ελέγχου μονάδας ή ενότητας (unit test) (βλ. Εικόνα 6).



Εικόνα 6. (unit test)

Ο κώδικας του ελέγχου μονάδας είναι ο ίδιος με τον παραγόμενο κώδικα, με την διαφορά της προσθήκης των εντολών ελέγχου. Οι έλεγχοι αυτού του τύπου είναι πολλαπλοί, μικρής εμβέλειας και συγκρίνουν τα αναμενόμενα αποτελέσματα με τα πραγματικά. Αποτελούν ένα ρητό μέρος του κώδικα, που διασφαλίζει και τεκμηριώνει τη σωστή λειτουργία του. Η συγγραφή και εκτέλεση των ελέγχων μονάδας γίνεται με γρήγορο και αυτοματοποιημένο τρόπο, συνήθως με την χρήση κάποιου εργαλείου.

2.7 Η γλώσσα UML

Η UML (Unified Modelling Language) είναι μια τυποποιημένη γλώσσα για τον καθορισμό, την οπτικοποίηση, την κατασκευή, και την τεκμηρίωση εκθεμάτων των συστημάτων λογισμικού, καθώς και για τη μοντελοποίηση των επιχειρήσεων και άλλων μη-συστημάτων λογισμικού. Η UML αντιπροσωπεύει μια συλλογή από τα καλύτερα μηχανικά κομμάτια που έχουν αποδειχθεί επιτυχείς στη μοντελοποίηση των μεγάλων και σύνθετων συστημάτων. Αποτελεί ένα πολύ σημαντικό μέρος της ανάπτυξης αντικειμένων προσανατολισμένου λογισμικού και της διαδικασίας ανάπτυξης λογισμικού. Επίσης, χρησιμοποιεί γραφικά σύμβολα για να εκφράσει το σχεδιασμό των έργων λογισμικού. Χρησιμοποιώντας τη UML βοηθά τις ομάδες έργου να επικοινωνούν, να διερευνούν δυναμικά σχέδια, και να επικυρώνουν τον αρχιτεκτονικό σχεδιασμό του λογισμικού.

Η UML αποτυπώνει τόσο τη στατική δομή, όσο και τη δυναμική συμπεριφορά ενός συστήματος. Ένα αντικειμενοστραφές σύστημα μοντελοποιείται ως μία συλλογή αντικειμένων που αλληλοεπιδρούν για την εκτέλεση μιας λειτουργίας η οποία είναι τελικά αξιοποιήσιμη από τον χρήστη του συστήματος. Η στατική δομή καθορίζει τα είδη των αντικειμένων που είναι σημαντικά για το σύστημα καθώς και τις συσχετίσεις μεταξύ τους. Η δυναμική συμπεριφορά προσδιορίζει την εξέλιξη των αντικειμένων σε σχέση με τον χρόνο και την επικοινωνία μεταξύ τους. Η κεντρική ιδέα πίσω από τη χρήση της UML είναι να συλλάβει τις σημαντικές λεπτομέρειες σχετικά με ένα σύστημα. Προσφέρει ένα πλούσιο συμβολισμό για τα οπτικά συστήματα λογισμικού μοντελοποίησης διευκολύνοντας έτσι αυτή τη διαδικασία. Η UML δεν προβλέπει μόνο την ένδειξη για τις βασικές δομικές μονάδες, αλλά παρέχει επίσης τρόπους για να εκφράσουν τις περίπλοκες σχέσεις μεταξύ των βασικών δομικών στοιχείων. Οι σχέσεις μπορεί να έχουν στατικό ή δυναμικό χαρακτήρα.

Οι Στατικές σχέσεις κυρίως περιστρέφονται γύρω από τις δομικές πτυχές ενός συστήματος. Η σχέση Κληρονομικότητας μεταξύ ενός ζευγαριού των κλάσεων, οι διεπαφές

που εφαρμόζονται από μια κλάση και η εξάρτηση από μια άλλη κατηγορία είναι τα παραδείγματα των στατικών σχέσεων. Οι Δυναμικές σχέσεις, από την άλλη, ανησυχούν με την συμπεριφορά του συστήματος και εξαιτίας αυτού υπάρχουν σε χρόνο εκτέλεσης. Τα μηνύματα που ανταλλάσσονται σε μια ομάδα κλάσεων, για να καλυφθούν πλήρως κάποιες ευθύνες και η ροή ελέγχου μέσα σε ένα σύστημα, είναι συμπεριλαμβανόμενες στο πλαίσιο της δυναμικής των σχέσεων που υπάρχουν μέσα σε ένα σύστημα. Και οι στατικές και οι δυναμικές πτυχές ενός συστήματος περιλαμβάνονται με τη μορφή των διαγραμμάτων της UML.

Η UML περιλαμβάνει τρία βασικά στοιχεία:

1. Οντότητες
2. Σχέσεις
3. Διαγράμματα

2.7.1 Διαγράμματα

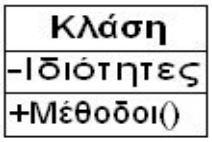
Η UML ορίζει τα παρακάτω διαγράμματα:

- Διάγραμμα περιπτώσεων χρήσης (use case diagram)
- Διαγράμματα δομής
- Διάγραμμα κλάσεων (class diagram)
- Διάγραμμα αντικειμένων (object diagram)
- Διαγράμματα συμπεριφοράς
- Διάγραμμα καταστάσεων (statechart diagram)
- Διάγραμμα δραστηριοτήτων (activity diagram)
- Διαγράμματα αλληλεπίδρασης
- Διάγραμμα ακολουθίας (sequence diagram)
- Διάγραμμα συνεργασίας (collaboration diagram)
- Διαγράμματα δομής υλοποίησης
- Διάγραμμα εξαρτημάτων (component diagram)
- Διάγραμμα ανάπτυξης (deployment diagram)

2.7.2 Κλάσεις

Οι κλάσεις αποτελούν τη βάση της κατασκευής οποιουδήποτε αντικειμενοστραφούς συστήματος. Ενσωματώνουν τα δεδομένα καθώς και τις λειτουργίες που επενεργούν στα δεδομένα αυτά. Συμβολίζονται με ένα παραλληλόγραμμο που έχει τρία διαμερίσματα. Στο πάνω διαμέρισμα αναγράφεται το όνομα της κλάσης, στο μεσαίο διαμέρισμά απεικονίζονται

οι ιδιότητες και στο κάτω διαμέρισμά οι λειτουργίες. Ανάλογα με τη οπτική γωνία σχεδίασης που χρησιμοποιούμε, η κλάση μπορεί να είναι μία κλάση λογισμικού ή μία έννοια του πεδίου του προβλήματος. Αντίστοιχα, οι λειτουργίες μπορεί να είναι μέθοδοι ή υποχρεώσεις υψηλού επιπέδου. Στην εικόνα 1 βλέπουμε μία κλάση.



Εικόνα 7 (Παράδειγμα σχεδίασης κλάσης)

Το συνηθισμένο σε μία κλάση είναι ότι οι ιδιότητες είναι ιδιωτικές (private) και οι λειτουργίες δημόσιες (public). Ιδιωτικά είναι τα χαρακτηριστικά μιας κλάσης, ιδιότητες ή λειτουργίες, που δεν είναι προσπελάσιμά από άλλες κλάσεις, ενώ δημόσια αυτά που είναι προσπελάσιμά από άλλες κλάσεις. Το ότι τα δεδομένα είναι ιδιωτικά σημαίνει ότι αντικείμενα άλλων κλάσεων δεν έχουν τη δυνατότητα να προσπελάσουν απευθείας τις ιδιότητες μίας κλάσης, αλλά χρησιμοποιούν για αυτό τις μεθόδους της κλάσης. Η τακτική αυτή ονομάζεται αρχή απόκρυψης των δεδομένων (Information hiding principle). Το σύμβολο «-» μπροστά από τις ιδιότητες και τα χαρακτηριστικά χρησιμοποιείται για να δηλώσει ιδιωτική πρόσβαση, ενώ το «+» σημαίνει δημόσια πρόσβαση. Το σύμβολο «#», που συμβολίζει την προστατευμένη πρόσβαση, έχει τη σημασία πως η ιδιότητα ή λειτουργία είναι προσπελάσιμά από την κλάση και τις τυχόν υποκλάσεις της. Η πρόσβαση σε επίπεδο πακέτου έχει την επισήμανση «~» και τη σημασία πως η ιδιότητα ή λειτουργία είναι προσπελάσιμα από την κλάση στην οποία δηλώνεται και τις άλλες κλάσεις που βρίσκονται στο ίδιο πακέτο με αυτήν.

2.7.3 Σχέσεις κλάσεων

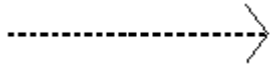
Μια συσχέτιση (association) μεταξύ δύο κλάσεων απεικονίζει μια στατική σχέση μεταξύ των δύο κλάσεων. Η ανάγκη μιας συσχέτισης προκύπτει από τη διαπίστωση πως για τη λειτουργία μιας κλάσης απαιτείται η συνεργασία της με μία ή περισσότερες άλλες κλάσεις. Αν αυτή η συνεργασία απαιτείται να είναι σε μόνιμή βάση, τότε χρησιμοποιούμε συσχέτιση, αν είναι παροδική τότε χρησιμοποιούμε εξάρτηση. μια την αναπαράσταση μια συσχέτισης χρησιμοποιούμε μία γραμμή μεταξύ των δύο κλάσεων. Κάθε συσχέτιση έχει δύο πέρατα συσχέτισης, κάθε πέρας είναι συνδεδεμένο με μία από τις τάξεις της συσχέτισης. Ένα πέρας μπορεί να ονομαστεί με μία ετικέτα. Αυτή η ετικέτα ονομάζεται όνομα ρόλου (Romeo

name) και τα πέρατα των συσχετίσεων συχνά ονομάζονται ρόλοι. Το πέρας μιας συσχέτισης έχει επίσης πολλαπλότητα (multiplicity), η οποία είναι μία ένδειξη για το πόσα αντικείμενα μπορούν να συμμετέχουν σε αυτή τη σχέση. Οι γραμμές συσχέτισης έχουν βέλη τα οποία δείχνουν τη δυνατότητα πλοήγησης (navigability). Αν μια δυνατότητα πλοήγησης υπάρχει μόνο στην μια κατεύθυνση, ονομάζουμε τη συσχέτιση μονόδρομή (unidirectional association), ενώ αν υπάρχει δυνατότητα πλοήγησης και προς τις δύο κατευθύνσεις τότε η συσχέτιση ονομάζεται αμφίδρομή.

Στη UML ορίζονται τέσσερεις βασικές σχέσεις:

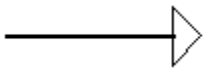
Εξάρτηση

Η εξάρτηση δηλώνει πως μια αλλαγή σε μια οντότητα θα επηρεάσει μια άλλη αλλά όχι απαραίτητα και το αντίστροφο. Παριστάνεται με μια διακεκομμένη γραμμή με ανοιχτό βέλος που δείχνει προς την οντότητα που υπάρχει εξάρτηση:



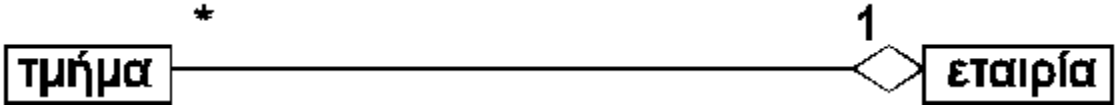
Γενίκευση

Η γενίκευση δηλώνει μια σχέση ανάμεσα σε κάτι γενικό (τη βασική κλάση ή αλλιώς γονέα) και κάτι ειδικό (μια υποκλάση ή αλλιώς παιδί της). Παριστάνεται με μια συνεχή γραμμή με κλειστό βέλος που δείχνει προς τη βασική κλάση:



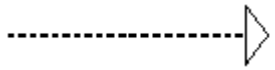
Σύνδεση

Η σύνδεση αναφέρεται σε αντικείμενα τα οποία συνδέονται με κάποιο τρόπο με άλλα. Όταν δύο κλάσεις είναι συνδεδεμένες μπορεί κανείς να μεταβεί από αντικείμενα της μιας σε αντικείμενα της άλλης. Η σύνδεση παριστάνεται με μια ευθεία γραμμή ανάμεσα στα δύο αντικείμενα.



Υλοποίηση

Η υλοποίηση δηλώνει πως ο εξυπηρετούμενος (αυτός που βρίσκεται στην ουρά του βέλους) υποστηρίζει τη δ επαφή (τουλάχιστον όλες τις πράξεις) που ορίζονται από τον παροχέα (αυτόν που βρίσκεται στην κεφαλή του βέλους):



Είδη εξαρτήσεων

Με τη χρήση μιας εξάρτησης εκφράζουμε σημασιολογικές (semantic) σχέσεις ανάμεσα σε στοιχεία του μοντέλου. Σε τέτοιες περιπτώσεις μια αλλαγή σε ένα στοιχείο της εξάρτησης μπορεί να έχει επίπτωση στο άλλο. Διακρίνουμε τα παρακάτω είδη εξάρτησης:

Πρόσβαση (access): Άδεια σε κάποια στοιχεία από ένα τμήμα να έχουν πρόσβαση σε στοιχεία από άλλο τμήμα (access).

Σύνδεση (binding): Παροχή τιμών σε ένα πρότυπο για να δημιουργήσει ένα νέο στοιχείο (bind).

Κλήση (call): Μια μέθοδος καλεί μια άλλη (call).

Απόρροια (derivation): Ένα στοιχείο που μπορεί να υπολογιστεί από κάποιο άλλο (derive).

Friend: Ένα στοιχείο μπορεί να έχει πρόσβαση σε στοιχεία άλλης κλάσης παρά τους όποιους περιορισμούς (friend).

Εισαγωγή (import): Άδεια σε ένα τμήμα να εισάγει και να χρησιμοποιήσει τα στοιχεία ενός άλλου τμήματος (import).

Δημιουργία (instantiation): Μια μέθοδος μιας κλάσης δημιουργεί αντικείμενα μιας άλλης κλάσης (instantiate).

Παράμετρος (parameter): Σχέση ανάμεσα σε μια λειτουργία και τις παραμέτρους της (parameter).

Δημιουργία (realization):Σχέση ανάμεσα σε μια προδιαγραφή και την υλοποίησή της (realize).

Εκλέπτυνση (refinement): Δήλωση για την ύπαρξη απεικόνισης ανάμεσα σε δύο σημασιολογικά επίπεδα (refine).

Αποστολή (send): Σχέση ανάμεσα στον αποστολέα και τον παραλήπτη ενός μηνύματος(send).

Ίχνος (trace): Σχέση ανάμεσα σε δύο στοιχεία δύο διαφορετικών μοντέλων που δεν αποτελεί όμως απεικόνιση (trace).

Χρήση (usage): Ένα στοιχείο απαιτεί την ύπαρξη ενός άλλου στοιχείο για τη λειτουργία του (π.χ. call, instantiate, parameter, send) (use).

2.8 ICONIX

Η ICONIX είναι μια μεθοδολογία ανάπτυξης λογισμικού που επιτρέπει τη συστηματική μετάβαση από τις αρχικές απαιτήσεις όπως αυτές διατυπώνονται από τον πελάτη, στον κώδικα που τελικά υλοποιεί τις απαιτήσεις αυτές. Είναι μια απλούστερη εκδοχή της ευρέως διαδεδομένης Ενοποιημένης Προσέγγισης (Unified Process).

Το μείζον χαρακτηριστικό της μεθοδολογίας ICONIX είναι η επαναληπτικότητα. Αφενός, η διαδικασία είναι επαναληπτική διότι επιτρέπει την παραγωγή λειτουργικού κώδικα για κάθε μια περίπτωση χρήσης του συστήματος ξεχωριστά. Σε κάθε επανάληψη, εξετάζεται μια νέα περίπτωση χρήσης που καταλήγει στην προσθήκη λειτουργικότητας στο τελικό προϊόν. Αφετέρου, η διαδικασία είναι επαναληπτική διότι επιτρέπει (και υποβοηθά) την επιστροφή από ένα στάδιο της διαδικασίας ανάπτυξης (π.χ. το σχεδιασμό) σε προηγούμενα (π.χ. την ανάλυση απαιτήσεων) με σκοπό την αποσαφήνιση, βελτίωση και διόρθωση προηγούμενων ενεργειών.

Η επαναληπτικότητα βρίσκεται στο κέντρο του αντικειμενοστραφούς υποδείγματος προγραμματισμού καθώς αναγνωρίζει ότι ένα μεγάλο σύστημα λογισμικού δεν μπορεί να αναπτυχθεί με μιας και ότι οι αλλαγές σε προηγούμενες επιλογές είναι αναπόφευκτες.

Η φιλοσοφία της μεθοδολογίας ICONIX αποτυπώνεται στο διάγραμμα της Εικόνα 8. Το ζητούμενο είναι από τις απαιτήσεις του χρήστη να παραχθεί το τελικό προϊόν, δηλαδή ο λειτουργικός κώδικας. Ο κώδικας παράγεται με τη διαδοχική εκλέπτυνση και βελτίωση δύο μοντέλων:

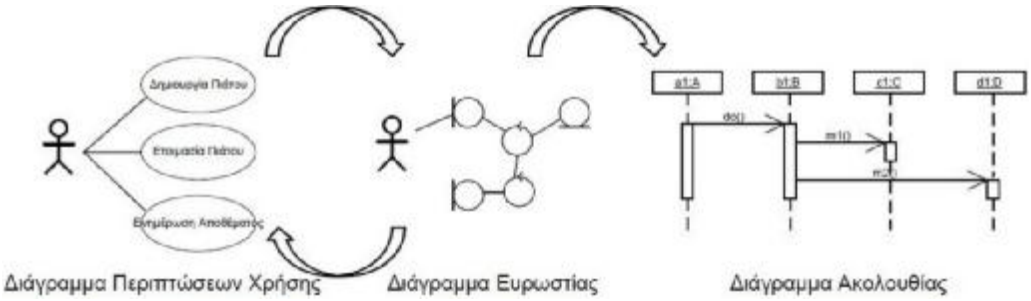
α) του στατικού μοντέλου που περιγράφει την αρχιτεκτονική του συστήματος, δηλαδή τις δομικές του μονάδες και τις σχέσεις μεταξύ τους και

β) του δυναμικού μοντέλου που περιγράφει τον τρόπο με τον οποίο οι μονάδες αλληλοεπιδρούν για την επίτευξη της επιθυμητής λειτουργικότητας. Οι απαιτήσεις του πελάτη/χρήστη του συστήματος θεωρείται ότι διατυπώνονται σε κάποιο αρχικό κείμενο (γνωστό ως απαιτήσεις υψηλού επιπέδου) και ενδεχομένως σε κάποια σκίτσα της επιθυμητής γραφικής διασύνδεσης χρήστη.

Η μεθοδολογία ICONIX βασίζεται στην αξιοποίηση ενός υποσυνόλου της UML για τη δημιουργία διαγραμμάτων ως ενδιάμεσα προϊόντα κατά την εξέλιξη του δυναμικού και στατικού μοντέλου του συστήματος που αναπτύσσεται. Συγκεκριμένα, από το σύνολο των διαγραμμάτων της UML, χρησιμοποιούνται τα διαγράμματα περιπτώσεων χρήσης, τα διαγράμματα κλάσεων, τα διαγράμματα ευρωστίας και τα διαγράμματα ακολουθίας.

Ο στόχος της iconix

ΔΥΝΑΜΙΚΟ ΜΟΝΤΕΛΟ ΣΥΣΤΗΜΑΤΟΣ



ΣΤΑΤΙΚΟ ΜΟΝΤΕΛΟ ΣΥΣΤΗΜΑΤΟΣ

Εικόνα 8.: Γραφική Επισκόπηση της Μεθοδολογίας ICONIX

Από τη στιγμή που οι απαιτήσεις του πελάτη δοθούν στην εταιρεία ανάπτυξης λογισμικού (και η εταιρεία αναλάβει το έργο) η διαδικασία ακολουθεί τις εξής φάσεις:

• **Ανάλυση Απαιτήσεων**

Το πρώτο βήμα στην ανάλυση των απαιτήσεων είναι η καταγραφή των οντοτήτων του πεδίου που διαπραγματεύεται το σύστημα που πρόκειται να αναπτυχθεί (πεδίο προβλήματος) και των σχέσεων μεταξύ τους. Οι οντότητες αυτές αποτελούν τη βάση του στατικού αντικειμενοστραφούς μοντέλου καθώς η λειτουργία του λογισμικού βασίζεται στην αλληλεπίδραση μεταξύ τους. Έχοντας το μοντέλο του πεδίου προβλήματος στη διάθεσή μας, επόμενο βήμα στην ανάλυση απαιτήσεων είναι η λεπτομερής και σαφής περιγραφή των απαιτήσεων του πελάτη υπό μορφή περιπτώσεων χρήσης.

• **Ανάλυση – Αρχική Σχεδίαση**

Σε επίπεδο δυναμικού μοντέλου κύριο εργαλείο στη φάση αυτή είναι τα διαγράμματα ευρωστίας για τη διερεύνηση των ενεργειών που υπαγορεύονται από τις περιπτώσεις χρήσης και κυρίως την βελτίωση του κειμένου των ίδιων των περιπτώσεων χρήσης. Με το πέρας της ανάλυσης ευρωστίας προκύπτει συνήθως ένα αναθεωρημένο και εμπλουτισμένο διάγραμμα κλάσεων (ως μετεξέλιξη του μοντέλου πεδίου προβλήματος) που περιλαμβάνει επιπρόσθετες κλάσεις καθώς και ορισμένες από τις ιδιότητες των κλάσεων.

• **Σχεδίαση**

Στη φάση της σχεδίασης επιχειρείται η ακριβής διερεύνηση της δυναμικής συμπεριφοράς του συστήματος και η κατανομή της λειτουργικότητας στις κλάσεις. Κύριο εργαλείο για το σκοπό αυτό είναι τα διαγράμματα ακολουθίας. Με το πέρας της σχεδίασης η ομάδα

ανάπτυξης παράγει το τελικό διάγραμμα κλάσεων που αποτελεί την είσοδο για τη φάση της υλοποίησης του λογισμικού.

• **Υλοποίηση**

Το σημαντικότερο προϊόν της διαδικασίας ανάπτυξης δεν είναι φυσικά τα διαγράμματα αλλά ο ίδιος ο κώδικας που ικανοποιεί τις απαιτήσεις του πελάτη. Στη φάση της υλοποίησης αναπτύσσεται ο κώδικας βάσει της σχεδίασης που προηγήθηκε (σε μεγάλο βαθμό υπάρχει η δυνατότητα παραγωγής κώδικα από τα διαγράμματα κλάσεων και ακολουθίας). Παρόλο που δεν εξετάζεται στην παρούσα μελέτη, σημαντικό μέρος της υλοποίησης είναι και ο έλεγχος μονάδων (unit testing) δηλαδή η εξασφάλιση της ορθής λειτουργίας των κλάσεων που δημιουργήθηκαν.

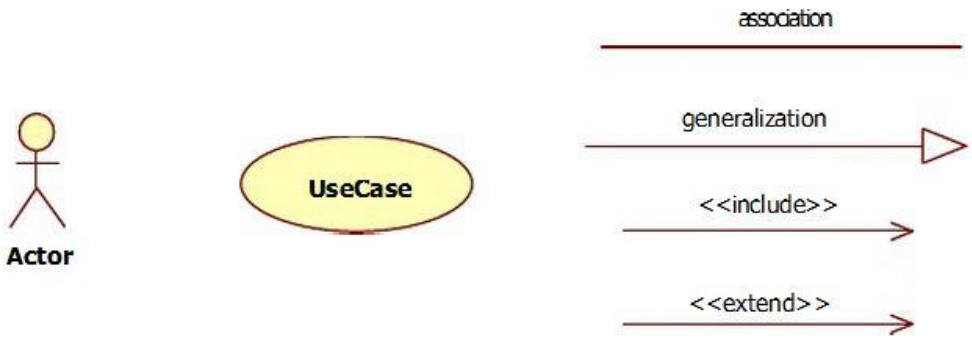
Σύμφωνα με τη μεθοδολογία ICONIX αλλά και τις γενικότερες επιταγές της Τεχνολογίας Λογισμικού, στο τέλος κάθε φάσης θα πρέπει να πραγματοποιείται μια συστηματική **επισκόπηση** (inspection/review). Στις επισκοπήσεις αυτές τα μέλη της ομάδας ανάπτυξης (αλλά ενδεχομένως και εκπρόσωποι του πελάτη ή της διοίκησης) ελέγχουν την ορθότητα των μοντέλων που δημιουργήθηκαν και τη συνέπεια προς τις απαιτήσεις

Όπως αναφέραμε παραπάνω η μεθοδολογία ICONIX από το σύνολο των διαγραμμάτων της UML, χρησιμοποιεί τα διαγράμματα περιπτώσεων χρήσης, τα διαγράμματα κλάσεων, τα διαγράμματα ευρωστίας και τα διαγράμματα ακολουθίας. Έτσι θα κάνουμε μια ποιο αναλυτική αναφορά αφού θα είναι και τα διαγράμματα που θα χρησιμοποιήσουμε στην παραγωγή του λογισμικού που θα παρουσιάσουμε στην συνέχεια.

2.9 Διάγραμμα περιπτώσεων χρήσης

Το διάγραμμα περιπτώσεων χρήσης στη UML χρησιμοποιείται για την μοντελοποίηση της λειτουργικότητας ενός συστήματος, όπως αυτή γίνεται αντιληπτή από τον εξωτερικό χρήστη. Τα διαγράμματα αυτά διαμερίζουν τη λειτουργικότητα του συστήματος σε συναλλαγές που έχουν νόημα για τους χρήστες του συστήματος ή αλλιώς χειριστές (actors). Τα επιμέρους τμήματα της λειτουργικότητας ονομάζονται περιπτώσεις χρήσης (use cases). Το σύνολο των περιπτώσεων χρήσης συνιστούν τη συμπεριφορά του συστήματος. Τα βασικά διαγραμματικά στοιχεία του διαγράμματος περιπτώσεων χρήσης

είναι το σύστημα, ο χειριστής, η περίπτωση χρήσης και οι σχέσεις μεταξύ τους. Τα στοιχεία αυτά φαίνονται παρακάτω στην Εικόνα 9:



Εικόνα 9. Βασικά διαγραμματικά στοιχεία του διαγράμματος περιπτώσεων χρήσης.

Η αξία του διαγράμματος περιπτώσεων χρήσης είναι ιδιαίτερα σημαντική, διότι καθορίζει τις λειτουργικές απαιτήσεις, οι οποίες θα αποτελέσουν σημείο αναφοράς καθ' όλη τη διάρκεια ανάπτυξης του συστήματος. Ο σημαντικότερος ρόλος του συγκεκριμένου διαγράμματος είναι ότι αποτελεί ένα μέσο επικοινωνίας μεταξύ πελατών και σχεδιαστών, όσον αφορά στη λειτουργικότητα του συστήματος. Η απλότητα των συμβολισμών το καθιστά ιδανικό για αυτό το σκοπό, παρέχοντας τη δυνατότητα εύκολης αντίληψης του συνόλου των λειτουργιών καθώς και εύκολης τροποποίησής τους.

Ο χειριστής αντιπροσωπεύει μια εξωτερική οντότητα, άνθρωπο ή σύστημα, η οποία αλληλοεπιδρά με το σύστημα. Ο χειριστής αναπαριστά ένα ρόλο, όχι έναν μεμονωμένο χρήστη του συστήματος, μιας και ο ίδιος χρήστης μπορεί να αλληλοεπιδρά με το σύστημα με πολλαπλούς ρόλους. Οι χειριστές είναι κλάσεις με το στερεότυπο «actor», όπου το όνομα της κλάσης γενικά αναπαριστά το ρόλο του χειριστή. Το σύμβολο του χειριστή φαίνεται στην Εικόνα 9. Στο διάγραμμα περιπτώσεων χρήσης χρησιμοποιείται μόνο η σχέση γενίκευσης ανάμεσα σε χειριστές, προκειμένου να περιγραφεί η κοινή συμπεριφορά ανάμεσα τους, την οποία και κληρονομούν από μια πρόγονο κλάση χειριστή.

Ο τυπικός ορισμός μιας περίπτωσης χρήσης είναι μια ακολουθία ενεργειών που πραγματοποιείται από το σύστημα για την παραγωγή μετρήσιμων αποτελεσμάτων που έχουν νόημα για τον χρήστη. Η περίπτωση χρήσης ορίζει ένα συγκεκριμένο τρόπο χρησιμοποίησης του συστήματος, προσδιορίζοντας την αλληλεπίδραση ανάμεσα σε έναν ή περισσότερους χειριστές και το σύστημα. Το στιγμιότυπο μιας περίπτωσης χρήσης ονομάζεται σενάριο

(scenario), και αναπαριστά ένα συγκεκριμένο μονοπάτι εκτέλεσης (execution path) μέσα στο σύστημα. Ο συμβολισμός φαίνεται στην Εικόνα 9

Η περίπτωση χρήσης έχει τα ακόλουθα χαρακτηριστικά:1. Ξεκινάει πάντα από ένα χειριστή.2. Πρέπει να επιστρέφει κάποιου είδους απτή πληροφορία στο χρήστη.3. Μια περίπτωση χρήσης είναι πλήρης, με την έννοια ότι αποτελεί μια πλήρη περιγραφή. Μια περίπτωση χρήσης δε θεωρείται ότι έχει ολοκληρωθεί μέχρις ότου η τελική πληροφορία παραχθεί, ακόμη κι αν απαιτούνται γι' αυτό πολλαπλές αλληλεπιδράσεις μεταξύ των αντικειμένων. Ένα σύνηθες λάθος είναι η διαίρεση μιας περίπτωσης χρήσης σε μικρότερες, οι οποίες παράγουν ενδιάμεσα αποτελέσματα. Ανάμεσα στις περιπτώσεις χρήσης υπάρχουν τρία είδη σχέσεων: η επέκταση (extends), η συμπερίληψη (uses ή includes) και η ομαδοποίηση (grouping). Οι σχέσεις αυτές φαίνονται στην Εικόνα 9 Η σχέση της επέκτασης είναι μια σχέση γενίκευσης που χρησιμοποιείται στην περίπτωση όπου μια περίπτωση χρήσης συμπεριλαμβάνει ένα τμήμα, όχι απαραίτητα ολόκληρη, την συμπεριφορά της περίπτωσης χρήσης που επεκτείνει. Τέτοιου είδους περιπτώσεις χρήσης χρησιμοποιούνται στο χειρισμό εξαιρέσεων. Η σχέση της συμπερίληψης είναι και αυτή μια σχέση γενίκευσης που χρησιμοποιείται στην περίπτωση όπου μια περίπτωση χρήσης συμπεριλαμβάνει την πλήρη λειτουργικότητα μιας άλλης.

Όταν ένα σύνολο περιπτώσεων χρήσης παρουσιάζουν σε κάποια τμήματα κοινή συμπεριφορά, η σχέση αυτή χρησιμοποιείται για τη μοντελοποίηση αυτής της κοινής συμπεριφοράς σε μια περίπτωση χρήσης που χρησιμοποιείται από τις υπόλοιπες. Τέλος με τη σχέση της ομαδοποίησης, περιπτώσεις χρήσης, οι οποίες διαθέτουν παρόμοια συμπεριφορά ή σχετίζονται με κάποιο τρόπο μεταξύ τους, οργανώνονται σε πακέτα. Ωστόσο, για λόγους απλότητας των διαγραμμάτων περιπτώσεων χρήσης η τελευταία σχέση συνήθως δε χρησιμοποιείται.

Η περιγραφή των περιπτώσεων χρήσης γίνεται με τη μορφή κειμένου στην ορολογία του χρήστη και αποτελεί μια απλή και συνεπή τεκμηρίωση. Η τεκμηρίωση κάθε περίπτωσης χρήσης, για να είναι πλήρης, θα πρέπει να περιλαμβάνει μια ακολουθία γεγονότων που λαμβάνουν χώρα για την υλοποίηση της επιθυμητής συμπεριφοράς. Επικεντρώνεται στην εξωτερική συμπεριφορά του συστήματος, αγνοώντας τον τρόπο υλοποίησης και την εσωτερική δομή του. Μερικά σημεία τα οποία θα πρέπει να συμπεριλαμβάνονται στην περιγραφή είναι ο στόχος της περίπτωσης χρήσης, από ποιόν χειριστή ξεκινάει, η ακολουθία των μηνυμάτων μεταξύ χειριστή και συστήματος, εναλλακτική ροή γεγονότων σε

περιπτώσεις εξαιρέσεων, και τέλος το πώς η περίπτωση χρήσης τερματίζεται επιστρέφοντας κάποια τιμή στο χειριστή.

2.10 Διάγραμμα κλάσεων

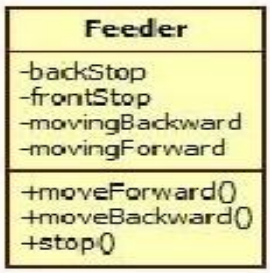
Το διάγραμμα κλάσεων είναι ο πρώτος τύπος διαγράμματος της UML που θα δούμε, και ο οποίος έχει άμεση σχέση με τα αντικειμενοστραφή συστήματα. Τα διαγράμματα περιπτώσεων χρήσης που είδαμε προηγουμένως είναι διαγράμματα καταγραφής προδιαγραφών και είναι χρήσιμα για κάθε τύπο συστήματος. Σε ένα αντικειμενοστρεφές σύστημα τα δομικά στοιχεία του είναι οι κλάσεις και οι σχέσεις μεταξύ των κλάσεων, οι οποίες επιτρέπουν τη συνεργασία αντικειμένων που δημιουργούνται ως στιγμιότυπα των κλάσεων. Το διάγραμμα κλάσεων αποτελείται από τις κλάσεις του συστήματος και τις μεταξύ τους συσχετίσεις, περιγράφοντας με αυτό τον τρόπο τη στατική δομή του συστήματος. Το διάγραμμα κλάσεων μπορεί να χρησιμοποιηθεί σε διάφορες φάσεις της ανάπτυξης του συστήματος. Στο αρχικό στάδιο της ανάλυσης απαιτήσεων οι κατασκευαστές αρχίζουν να αποκτούν γνώση για το πεδίο του προβλήματος του συστήματος. Αυτή η αρχική κατανόηση των εννοιών του πεδίου του προβλήματος καταγράφεται σε ένα διάγραμμα κλάσεων, το οποίο ονομάζεται μοντέλο του πεδίου προβλήματος (problem domain model).

Στο μοντέλο αυτό καταγράφονται ως κλάσεις οι έννοιες του πεδίου του προβλήματος και οι μεταξύ τους συσχετίσεις. Έπειτα, στο στάδιο της ανάλυσης, με οδηγό το μοντέλο του πεδίου προβλήματος, κατασκευάζεται ένα διάγραμμα κλάσεων, το οποίο αναπαριστά τη βασική αρχιτεκτονική δομή του συστήματος. Σε αυτό το στάδιο οι κλάσεις πρέπει να επιδιώκουν την αναπαράσταση του συστήματος που μοντελοποιείται με την ελάχιστη δυνατή πληροφορία, χωρίς να επιχειρείται αναφορά σε θέματα υλοποίησης. Στη συνέχεια, μεταβαίνοντας στο στάδιο της σχεδίασης, η περιγραφή των κλάσεων συμπληρώνεται με τις λειτουργίες που υλοποιούν τη συμπεριφορά των αντικειμένων και με επιπρόσθετες ιδιότητες ή συσχετίσεις, που επιβάλλονται από το περιβάλλον υλοποίησης. Τέλος, κατά την υλοποίηση του συστήματος, είναι δυνατόν να επέλθουν τροποποιήσεις στη δομή των κλάσεων λόγω απαιτήσεων που σχετίζονται με απόκρυψη πληροφορίας, ορατότητα και άλλες μη λειτουργικές απαιτήσεις, όπως π.χ. απόδοση και ασφάλεια. Σε μερικές περιπτώσεις, το διάγραμμα κλάσεων είναι το μόνο είδος διαγράμματος της UML που χρησιμοποιείται, λόγω των πληροφοριών που παρέχει σχετικά με τον πηγαίο κώδικα. Όπως θα αναφερθεί

παρακάτω, υπάρχει η δυνατότητα αυτόματης παραγωγής τμημάτων κώδικα από το διάγραμμα κλάσεων, καθώς και η αυτόματη δημιουργία διαγραμμάτων κλάσεων λαμβάνοντας ως είσοδο τον πηγαίο κώδικα.

Για το λόγο αυτό, ο κάθε συμβολισμός είναι σημαντικός, ακόμα κι αν υποδηλώνεται με ένα στοιχειώδες σύμβολο στο διάγραμμα κλάσεων. Στη συνέχεια παρουσιάζονται τα βασικά διαγραμματικά στοιχεία ενός διαγράμματος κλάσεων.

Οι κλάσεις αποτελούν τη βάση της κατασκευής οποιουδήποτε αντικειμενοστραφούς συστήματος. Ενσωματώνουν δεδομένα, καθώς και τις λειτουργίες που επενεργούν στα δεδομένα αυτά. Ο συμβολισμός της κλάσης φαίνεται στην Εικόνα 10. Αν μια κλάση είναι αφηρημένη το όνομα της κλάσης σημειώνεται με πλάγιους χαρακτήρες. Στο παρακάτω σχήμα φαίνεται η κλάση που αναπαριστά το φυσικό αντικείμενο Feeder (*Εικόνα 10:*):



Εικόνα 10: Η κλάση Feeder

Το συντακτικό για τη δήλωση ιδιοτήτων στη UML είναι:

ορατότητα όνομα : Τύπος = Αρχική Τιμή

Ενώ το συντακτικό για τη δήλωση λειτουργιών είναι: ορατότητα όνομα (λίστα παραμέτρων): Επιστρεφόμενος τύπος .

Η ορατότητα απεικονίζεται με τα σύμβολα “-”, “+”, “#”,“~” τα οποία δηλώνουν ιδιωτική, δημόσια, προστατευμένη καθώς και πρόσβαση σε επίπεδο πακέτου αντίστοιχα. Μια στατική ιδιότητα ή λειτουργία, που ανήκει δηλαδή στην κλάση και όχι στα στιγμιότυπά της, υποδηλώνεται στη UML υπογραμμίζοντας το όνομα της ιδιότητας ή της μεθόδου αντίστοιχα. Μια συσχέτιση μεταξύ δύο κλάσεων απεικονίζει μια στατική σχέση μεταξύ τους. Αν η σχέση αυτή μεταξύ των κλάσεων υφίσταται σε μόνιμη βάση, τότε χρησιμοποιούμε τη συσχέτιση, ενώ αν η σχέση είναι παροδική (π.χ. όταν τα αντικείμενα μιας κλάσης είναι παράμετροι σε

μια μέθοδο μιας άλλης κλάσης) χρησιμοποιούμε την εξάρτηση. Ενώ μια συσχέτιση στη UML συνδέει δύο κλάσεις ενός μοντέλου, ένα στιγμιότυπο μιας συσχέτισης συνδέει δύο συγκεκριμένα στιγμιότυπα κλάσεων και ονομάζεται σύνδεση (link). Προαιρετικά μπορούμε να έχουμε σε μία συσχέτιση τα εξής στοιχεία: Όνομα συσχέτισης, το οποίο θα πρέπει να υποδηλώνει με σαφήνεια το νόημα της συσχέτισης. Ονόματα άκρων συσχέτισης: τα οποία υποδηλώνουν το ρόλο αυτής της κλάσης στη συσχέτιση.

Πολλαπλότητα: Η πολλαπλότητα αφορά σε ένα άκρο μιας συσχέτισης και είναι το πλήθος των αντικειμένων που μπορεί να σχετίζονται με ένα αντικείμενο της άλλης κλάσης.

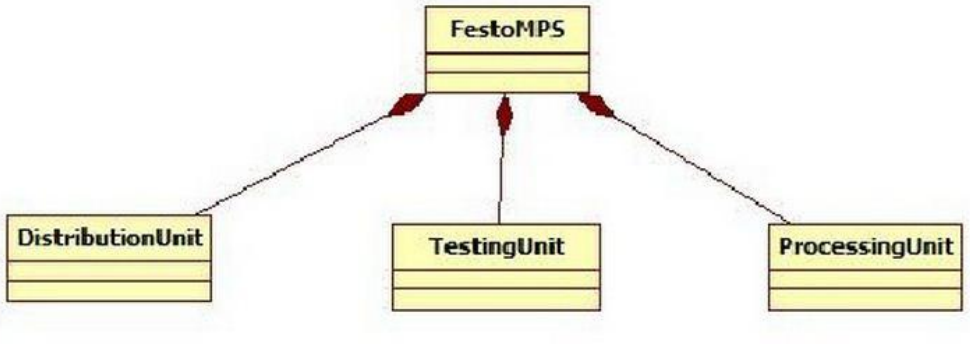
Πλοϊμότητα: Συμβολίζεται με ένα βέλος στο πέρας της συσχέτισης και υποδηλώνει πλοϊμότητα μόνο προς τη φορά του βέλους. Αφορά στη δυνατότητα που έχουμε από μία κλάση να ανακτήσουμε αντικείμενα της άλλης σε μια συσχέτιση. Όταν δεν υπάρχει πλοϊμότητα υπονοείται πλοϊμότητα και προς τις δύο κατευθύνσεις. Στη συνέχεια αναφέρονται διάφοροι ειδικότεροι τύποι συσχετίσεων.

2.10.1 Γενίκευση:

Η γενίκευση είναι μια ειδική μορφή συσχέτισης, η οποία αποτελεί μια σχέση μεταξύ μιας γενικής περιγραφής και μιας ειδικότερης περιγραφής που την επεκτείνει. Η γενίκευση αξιοποιεί το μηχανισμό της κληρονομικότητας και επιτρέπει πολυμορφική συμπεριφορά.

Η συσσωμάτωση είναι μια σχέση ειδικής μορφής που αναπαριστά μια σχέση συνόλου-τμήματος ή όλου-μέρους (whole-part). Η σύνθεση είναι μιας ισχυρότερης μορφής συσχέτιση στην οποία το σύνολο έχει την αποκλειστική ευθύνη διαχείρισης των τμημάτων , όπως τη δημιουργία και τη διαγραφή τους. Αν διαγραφεί για παράδειγμα η κλάση που αντιστοιχεί στο σύνολο, διαγράφονται και οι κλάσεις των τμημάτων. Στην Εικόνα 11. φαίνεται ένα παράδειγμα της σχέσης σύνθεσης από το FestoMPS .

(Περισσότερες πληροφορείς για το μηχανικό σύστημα ***Festo MPS*** *κάντε κλικ εδώ* *http://seg.ee.upatras.gr/seg/dev/FestoMPS.htm#Introduction_)*



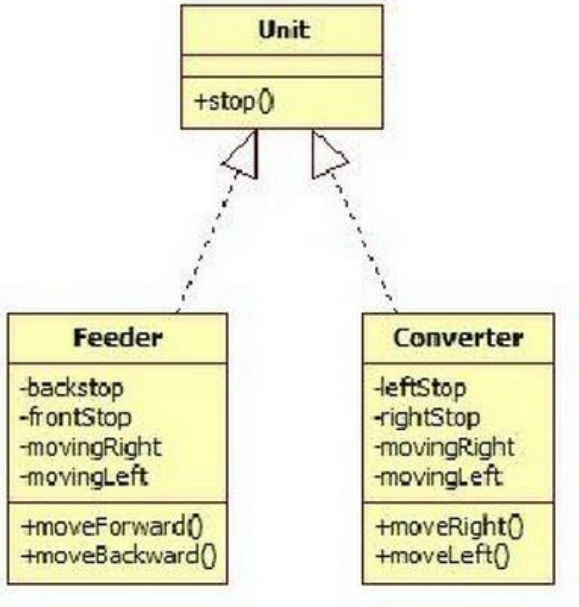
Εικόνα 11: Παράδειγμα σχέσης σύνθεσης

2.10.2 Εξάρτηση:

Μια εξάρτηση υποδηλώνει σημασιολογική σχέση μεταξύ δύο ή περισσοτέρων στοιχείων ενός μοντέλου. Αν δυο κλάσεις Α και Β συνδέονται με μια σχέση εξάρτησης από την Α προς τη Β, υποδηλώνεται ότι, παρόλο που η κλάση Α δε δημιουργεί ούτε «έχει» τη Β, απαιτεί την ύπαρξη της Β για την αποστολή μηνυμάτων προς αυτή. Αν η κλάση Β τροποποιηθεί, ενδεχομένως να απαιτείται και η τροποποίηση της κλάσης Α.

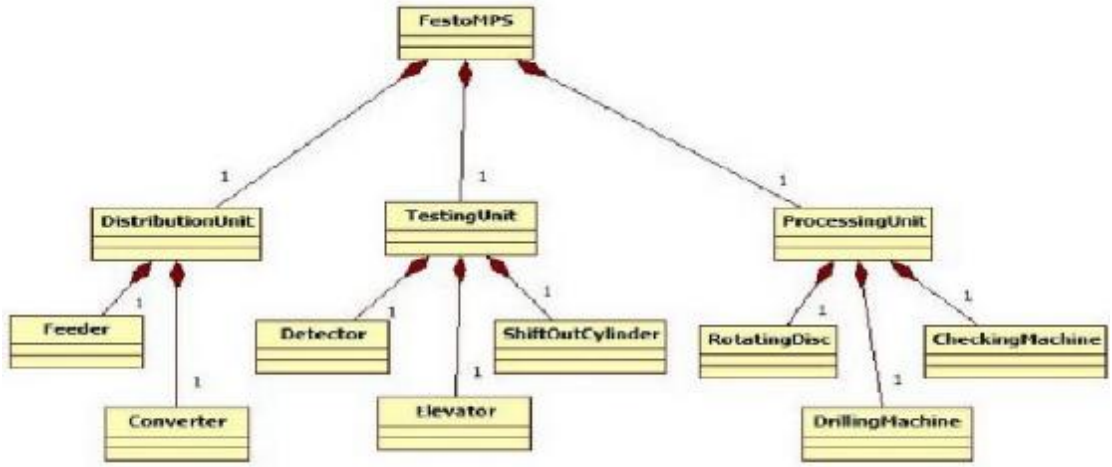
Διασύνδεση – σχέση πραγμάτωσης:

Η διασύνδεση (interface) είναι ένας τύπος που παρέχει λειτουργίες που είναι στο σύνολό τους αφαιρετικές. Η κλάση η οποία πραγματώνει μια διασύνδεση συνδέεται μαζί της με τη σχέση πραγμάτωσης (realization). Στο παρακάτω σχήμα φαίνεται ένα παράδειγμα σχέσης πραγμάτωσης από το Festo MPS: (Εικόνα 12.)



Εικόνα 12.Παράδειγμα σχέσης πραγμάτωσης

Παρόλο που ένα μοντέλο στη UML αναπαρίσταται γραφικά, απαιτείται συχνά χρήση κειμένου για μέγιστη δυνατή διαφάνεια. Ένας περιορισμός είναι μια λογική συνθήκη (έκφραση Boole) που πρέπει να είναι αληθής για να λάβει χώρα μια ενέργεια ή για να υπάρξει μια συσχέτιση. Η UML επιτρέπει τον καθορισμό περιορισμών με οποιοδήποτε τρόπο, αρκεί η περιγραφή να βρίσκεται μέσα σε άγκιστρα {}. Ωστόσο, η UML περιλαμβάνει τον ορισμό μιας τυπικής γλώσσας περιορισμών (Object Constraint Language – OCL). Παρακάτω φαίνεται το διάγραμμα κλάσεων για το μηχανικό σύστημα Festo MPS(Εικόνα 13.)



Εικόνα 13 :Διάγραμμα κλάσεων για το μηχανικό σύστημα FestoMPS

2.11 Διάγραμμα ακολουθίας

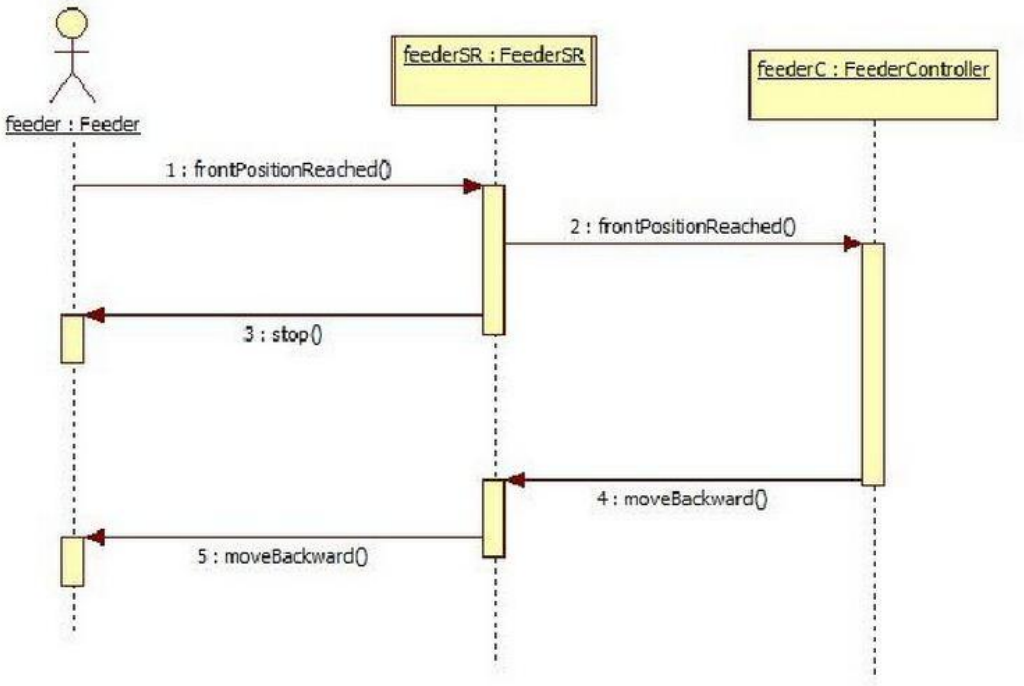
Το διάγραμμα ακολουθίας παρουσιάζει την αλληλεπίδραση μεταξύ αντικειμένων σε δύο διαστάσεις. Η κάθετη διάσταση αντιστοιχεί στην κλίμακα του χρόνου, ενώ στην οριζόντια διάσταση συμβολίζονται τα ανεξάρτητα αντικείμενα. Τα αντικείμενα συμβολίζονται με παραλληλόγραμμα μέσα στα οποία μπορεί να σημειωθεί το όνομα του στιγμιότυπου του αντικειμένου που συμμετέχει στο σενάριο που απεικονίζεται και ακολουθεί μετά από άνω-κάτω τελεία το όνομα της κλάσης στην οποία ανήκει το αντικείμενο. Σε κάθε αντικείμενο αντιστοιχεί μια κάθετη γραμμή που ονομάζεται γραμμή ζωής (lifeline). Τα αντικείμενα ανταλλάσσουν μηνύματα, τα οποία στην επίσημη ορολογία της UML ονομάζονται ερεθίσματα (stimuli). Ένα μήνυμα που αποστέλλεται μεταξύ των αντικειμένων συμβολίζεται ως ένα βέλος από τη γραμμή ζωής ενός αντικειμένου προς τη γραμμή ζωής ενός άλλου. Μήνυμα μπορεί να είναι οτιδήποτε από τα εξής:

Κλήση μιας λειτουργίας: όταν ένα αντικείμενο καλεί μια λειτουργία ενός άλλου αντικειμένου. Πρόκειται για σύγχρονο μήνυμα, δηλαδή ο αποστολέας του μηνύματος θα πρέπει να περιμένει την ολοκλήρωση της λειτουργίας για να συνεχίσει. Η κεφαλή του βέλους είναι γεμισμένη με μαύρο χρώμα. Πάνω από το βέλος αναγράφεται το όνομα της λειτουργίας που καλείται, με τις ενδεχόμενες παραμέτρους σε παρενθέσεις. Ειδική περίπτωση κλήσης είναι η αυτοκλήση, η οποία ξεκινάει από το αντικείμενο και καταλήγει πάλι σε αυτό.

Σήμα: όταν ένα αντικείμενο αποστέλλει ένα ασύγχρονο μήνυμα σε ένα άλλο αντικείμενο. Τυπικά ασύγχρονα μηνύματα συναντάμε σε πολυνηματικές εφαρμογές ,όπου ένα μήνυμα τοποθετείται σε κάποια ουρά ενός νήματος εκτέλεσης ενώ το ενεργό αντικείμενο-παραλήπτης θα επεξεργαστεί το μήνυμα σε κάποια επόμενη χρονική στιγμή. Η διαφορά με την κλήση λειτουργίας στον συμβολισμό είναι πως η κατάληξη είναι ένα ανοιχτό βέλος.

Επιστροφή κλήσης: είναι ένα διακεκομμένο βέλος το οποίο συμβολίζει την επιστροφή από μία κλήση λειτουργίας. Πάνω στο διακεκομμένο βέλος αναγράφεται συνήθως η τιμή επιστροφής, αν υπάρχει..

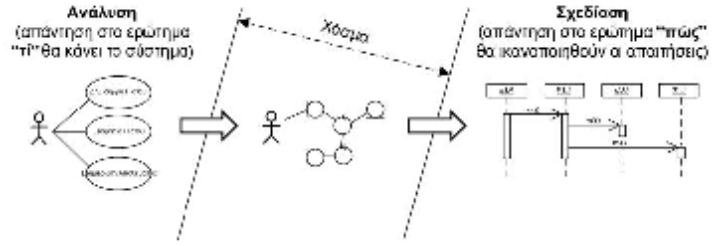
Μηνύματα υπό συνθήκη: Στα μηνύματα υπό συνθήκη τοποθετούνται αγκύλες μέσα στις οποίες αναγράφεται μία συνθήκη που μπορεί να είναι αληθής ή ψευδής . Η σημασία του συμβολισμού είναι ότι το μήνυμα θα αποσταλεί μόνο αν η συνθήκη είναι αληθής. Αν θέλουμε ταυτόχρονα να δείξουμε μια αποστολή εναλλακτικού μηνύματος στην περίπτωση που η συνθήκη είναι ψευδής, τότε δείχνουμε τα δύο αμοιβαία αποκεκομμένα μηνύματα σαν μηνύματα με το ίδιο σημείο εκκίνησης και γράφουμε στο πρώτο τη συνθήκη και στο δεύτερο τη φράση [else], όπως δείχνει το παρακάτω σχήμα. Σε ένα διάγραμμα ακολουθίας είναι δυνατόν να προστεθούν και περισσότεροι συμβολισμοί που υποδηλώνουν βρόχους επανάληψης, μηνύματα που αποστέλλονται πολλαπλές φορές, συγχρονισμό νημάτων κ.ο.κ. Ωστόσο, ο σκοπός των διαγραμμάτων ακολουθίας δεν είναι να αποτυπώσουν τις λεπτομέρειες ενός αλγορίθμου αλλά να αναπαραστήσουν με απλό και κατανοητό τρόπο τα σενάρια συνεργασίας μεταξύ αντικειμένων . Στην Εικόνα 14 φαίνεται το διάγραμμα ακολουθίας για την περίπτωση χρήσης “Front position reached” με actor τον Feeder.



Εικόνα 14:Διάγραμμα ακολουθίας για την περίπτωση χρήσης "Front position reached"

2.12 Διαγράμματα Ευρωστίας

Η ανάλυση ευρωστίας (robustness analysis) αποτελεί μια τεχνική, ενταγμένη στη φάση της ανάλυσης απαιτήσεων, για τη μετάβαση από τις περιπτώσεις χρήσης σε ένα λεπτομερές σχέδιο. Η γεφύρωση του χάσματος μεταξύ της ανάλυσης (που απαντά στο ερώτημα "τι" θα κάνει το σύστημα) και της σχεδίασης (που απαντά στο ερώτημα "πώς" θα ικανοποιηθούν οι απαιτήσεις του πελάτη) αναφέρεται στη μεθοδολογία ICONIX ως ο πρωταρχικός στόχος της ανάλυσης ευρωστίας (Εικόνα 15). Ως στάδιο, δεν έχει κάποιο παραδοτέο που να είναι απαραίτητο ή υποχρεωτικό για τη συνέχεια σε άλλες φάσεις της διαδικασίας ανάπτυξης. Είναι όμως μια εξαιρετική τεχνική για την εκλέπτυνση και αποσαφήνιση του κειμένου των περιπτώσεων χρήσης και τον εντοπισμό ενός αρχικού συνόλου αλληλοεπιδρώντων αντικειμένων (κλάσεων) για την ικανοποίηση της ζητούμενης λειτουργικότητας.



Εικόνα 15: Σημασία της ανάλυσης ευρωστίας στον κύκλο ζωής Λογισμικού

Κατά την ανάλυση ευρωστίας το κείμενο των περιπτώσεων χρήσης μεταφράζεται σταδιακά (πρόταση προς πρόταση) σε μια γραφική απεικόνιση κλάσεων και συμπεριφοράς (διάγραμμα ευρωστίας). Κάθε οντότητα η οποία σύμφωνα με το μοντέλο πεδίου προβλήματος (ή σύμφωνα με νεότερη κρίση των αναλυτών-σχεδιαστών) αποτελεί μια

κλάση του συστήματος, απεικονίζεται στο διάγραμμα ευρωστίας, κατηγοριοποιώντας την με βάση ένα από τα ακόλουθα τρία στερεότυπα κλάσεων:

- συνοριακές κλάσεις: κλάσεις που αποτελούν διασύνδεση μεταξύ του συστήματος και του "εξωτερικού κόσμου", δηλαδή των χειριστών του
- κλάσεις οντοτήτων: κλάσεις του μοντέλου πεδίου προβλήματος
- κλάσεις ελέγχου: κλάσεις που αποτελούν την "κόλλα" μεταξύ των συνοριακών και των κλάσεων οντοτήτων και αναπαριστούν τη συμπεριφορά του συστήματος.

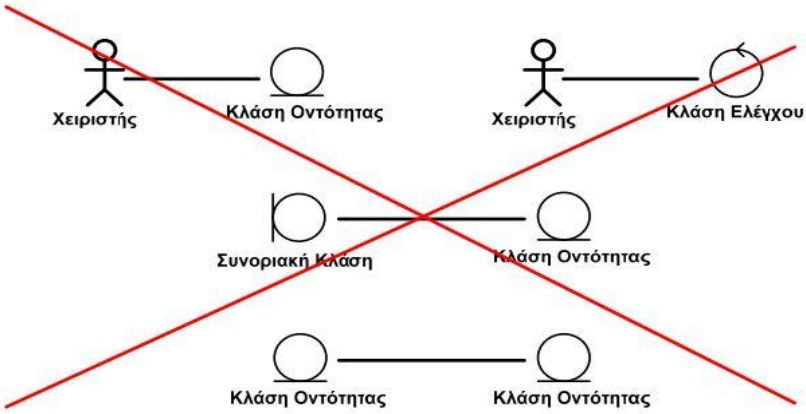
Σε ένα διάγραμμα ευρωστίας οι κλάσεις συσχετίζονται μεταξύ τους, υποδηλώνοντας την εννοιολογική συσχέτιση που υπάρχει ή υπονοείται στο κείμενο των περιπτώσεων χρήσης.

Κανόνες σχεδίασης διαγραμμάτων ευρωστίας

Κατά τη σχεδίαση διαγραμμάτων ευρωστίας οφείλουν να τηρούνται οι ακόλουθοι τρεις απλοί κανόνες:

- ουσιαστικά (χειριστές, συνοριακές κλάσεις και κλάσεις
- οντοτήτων) δεν μπορούν να συσχετίζονται απευθείας με άλλα ουσιαστικά
- ουσιαστικά μπορούν να συσχετίζονται με ρήματα (κλάσεις ελέγχου) και το αντίθετο
- ρήματα μπορούν να συσχετίζονται με άλλα ρήματα.

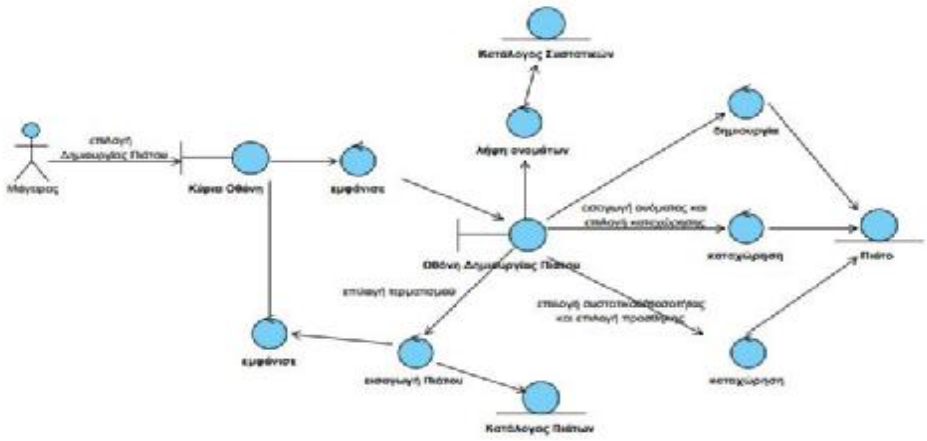
Εναλλακτικά, δεν συνιστώνται οι συσχετίσεις που απεικονίζονται στην ακόλουθη εικόνα:



Εικόνα 16: Μη συνιστάμενες συσχετίσεις σε διάγραμμα ευρωστίας

Οι κανόνες αυτοί ουσιαστικά επιβάλλουν την οργάνωση του κειμένου των περιπτώσεων χρήσης κατά τέτοιο τρόπο ώστε να είναι δυνατή στη συνέχεια η παραγωγή λεπτομερούς σχεδίου υπό μορφή διαγραμμάτων ακολουθίας.

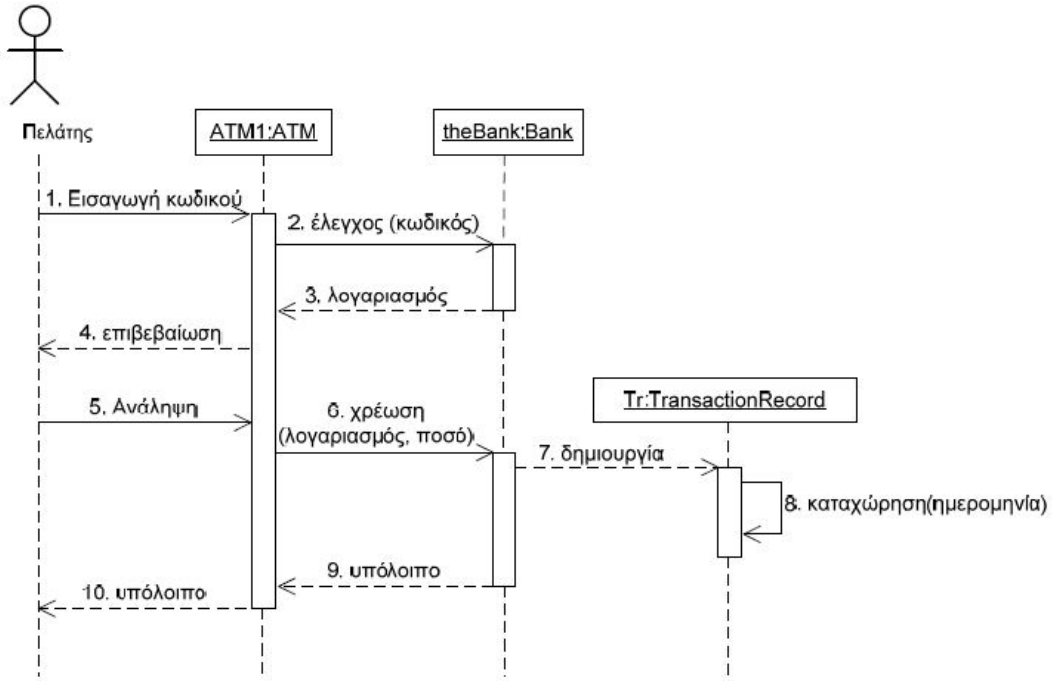
Στο παρακάτω σχήμα θα δώσουμε μια απεικόνιση ενός διαγράμματος ευρωστίας για την περίπτωση χρήσης «Δημιουργία Πιάτου» ενός λογισμικού που δημιουργεί παραγγελίες για εστιατόρια.



Εικόνα 17. Διάγραμμα Ευρωστίας για την Περίπτωση Χρήσης Δημιουργία

2.12 Παράδειγμα διαγράμματος ακολουθίας

Τα συνηθέστερα διαγραμματικά στοιχεία που εμφανίζονται σε ένα διάγραμμα ακολουθίας απεικονίζονται στην Εικόνα 18, που αναφέρεται σε ένα σύστημα τραπεζικών συναλλαγών μέσω ATM.



Εικόνα 18: Στοιχεία διαγράμματος ακολουθίας

Στο ανωτέρω διάγραμμα τα μηνύματα μεταξύ των αντικειμένων έχουν αριθμηθεί για λόγους ευκολότερης αναφοράς. Οι γραμμές που εκτείνονται κάτω από κάθε αντικείμενο του διαγράμματος ονομάζονται γραμμές ζωής (lifeline). Με τη λήψη ενός μηνύματος η γραμμή ζωής αντικαθίσταται με ένα ορθογώνιο που ονομάζεται πλαίσιο ενεργοποίησης και αντιστοιχεί στη διάρκεια εκτέλεσης της λειτουργίας που εξυπηρετεί το μήνυμα που λήφθηκε. Η αναπαράσταση του πλαισίου ενεργοποίησης σύμφωνα με πολλούς συγγραφείς θα πρέπει να αποφεύγεται καθώς προσθέτει περιττή πολυπλοκότητα στο διάγραμμα, υπό την έννοια ότι δεν συμβάλλει στην κατανομή της λειτουργικότητας σε κλάσεις.

2.13.1 Τύποι μηνυμάτων σε διαγράμματα ακολουθίας

Ο συνηθέστερος τύπος μηνύματος αντιστοιχεί στην κλήση μιας λειτουργίας ενός αντικειμένου και συμβολίζεται ως μια προσανατολισμένη ακμή με το βέλος από τον αποστολέα προς τον παραλήπτη (τέτοια μηνύματα είναι για παράδειγμα το 2 και το 6 στο

ανωτέρω σχήμα). Ο συμβολισμός αφορά σύγχρονα μηνύματα, δηλαδή μηνύματα όπου ο αποστολέας αναμένει την ολοκλήρωση της λειτουργίας από τον παραλήπτη για να συνεχίσει.

Επάνω από κάθε μήνυμα σημειώνεται το όνομα της λειτουργίας που καλείται καθώς και οι ενδεχόμενες παράμετροι που απαιτούνται Η ολοκλήρωση της λειτουργίας που ζητήθηκε από ένα αντικείμενο απεικονίζεται ως μία διακεκομμένη ακμή που επιστρέφει από το κληθέν αντικείμενο σε αυτό που πραγματοποίησε την κλήση και ονομάζεται επιστροφή μηνύματος. Επάνω στην ακμή σημειώνεται προαιρετικά η τιμή (ή οι τιμές) που επιστρέφονται, αν υπάρχουν. Τέτοια μηνύματα είναι τα μηνύματα 3 και 9 στο ανωτέρω σχήμα. Συνήθως, οι επιστροφές μηνυμάτων δεν απεικονίζονται στα διαγράμματα ακολουθίας.

Σε περίπτωση που η αποστολή μηνύματος προκαλεί τη δημιουργία ενός αντικειμένου πρόκειται για μήνυμα δημιουργίας. Απεικονίζεται ως ακμή όπου στο όνομα του μηνύματος σημειώνεται ρητά ο όρος "δημιουργία" μαζί με τυχόν παραμέτρους που απαιτούνται από τον κατασκευαστή της κλάσης. Για λόγους ευκολότερης κατανόησης, κατά τη δημιουργία αντικειμένου, το αντικείμενο που υλοποιείται τοποθετείται χαμηλότερα από τα υπόλοιπα, στο ύψος του μηνύματος, συμβολίζοντας ότι το αντικείμενο δεν υφίστατο πριν την αποστολή του μηνύματος. Τέτοιο μήνυμα δημιουργίας είναι το υπ' αριθμό 7 στο σχήμα 6.1.

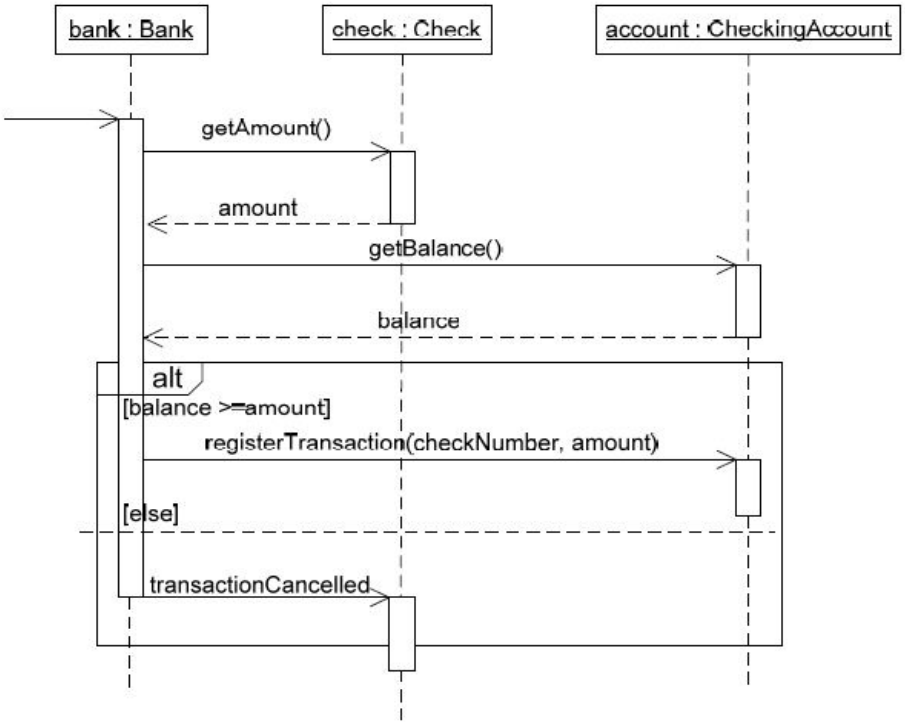
Το μήνυμα υπ' αριθμό 8 στο διάγραμμα ακολουθίας του σχήματος 6.1 αποτελεί μία αυτοκλήση. Με άλλα λόγια το μήνυμα αποστέλλεται από ένα αντικείμενο στον εαυτό του. Προγραμματιστικά αντιστοιχεί στην κλήση μιας μεθόδου (δημόσιας ή συνηθέστερα ιδιωτικής) του ιδίου αντικειμένου. Μηνύματα αυτοκλήσεων δεν κρίνεται σκόπιμο να απεικονίζονται για όλες τις κλήσεις μεθόδων εντός μιας κλάσης, αλλά μόνο στις περιπτώσεις που είναι επιθυμητό να δοθεί έμφαση σε κάποια λειτουργία που εκτελείται.

Προγραμματιστικά, η αποστολή ενός μηνύματος από ένα αντικείμενο (αποστολέας) προς κάποιο άλλο (αποδέκτης), προϋποθέτει την ύπαρξη μιας κατάλληλης μεθόδου στην κλάση του αποδέκτη, ώστε να μπορεί να αποκριθεί στο αίτημα που υποβάλλεται από τον αποστολέα. Η σύμβαση αυτή αποτελεί τη βάση για την κατανομή της λειτουργικότητας του συστήματος. Υπενθυμίζεται, ότι για κάθε περίπτωση χρήσης, οι χειριστές που εμπλέκονται, τα αντικείμενα που αντιστοιχούν σε συνοριακές κλάσεις αλλά και τα αντικείμενα των κλάσεων οντοτήτων μεταφέρονται αυτούσια στο διάγραμμα ακολουθίας. Δεν μεταφέρονται όμως και τα αντικείμενα των κλάσεων ελεγκτών καθώς συνήθως οι κλάσεις αυτές μεταφράζονται σε μηνύματα μεταξύ των αντικειμένων.

Σημειώνεται ότι στα διαγράμματα ακολουθίας πρέπει γενικά να αποφεύγεται η καταγραφή της ροής ελέγχου του συστήματος. Τα διαγράμματα ακολουθίας δεν αντικαθιστούν τα παλαιότερα διαγράμματα ροής όπου αναλύεται λεπτομερώς ένας αλγόριθμος, καταγράφοντας όλες τις πιθανές διαδρομές ανάλογα με τις τιμές διαφόρων συνθηκών (το ρόλο αυτό στη UML έχουν τα διαγράμματα δραστηριότητας). Σε εξαιρετικές μόνο περιπτώσεις πρέπει να χρησιμοποιούνται συμβολισμοί για την απεικόνιση επαναληπτικών δομών ή αποστολής μηνυμάτων υπό συνθήκη. Παρόλο που πολλοί συγγραφείς είναι τελείως αντίθετοι με την απεικόνιση της ροής ελέγχου σε διαγράμματα ακολουθίας, η νεότερη έκδοση του προτύπου της Ενοποιημένης Γλώσσας Μοντελοποίησης (UML 2) παρέχει τη δυνατότητα απεικόνισης των δομών της επανάληψης και της επιλογής. Για το σκοπό αυτό έχει εισαχθεί ο συμβολισμός του συνδυασμένου τμήματος (combined fragment) που χρησιμοποιείται για την ομαδοποίηση ενός συνόλου μηνυμάτων που εκτελούνται υπό συνθήκη. Σύμφωνα με το πρότυπο UML 2 υπάρχουν 11 περιπτώσεις για τα συνδυασμένα τμήματα. Στη συνέχεια θα παρουσιαστούν δύο συνήθεις περιπτώσεις που αξιοποιούνται και στα διαγράμματα ακολουθίας του συστήματος ηλεκτρονικής αξιολόγησης
.

2.13.2 Εναλλακτικές

Οι εναλλακτικές (alternatives) χρησιμοποιούνται για να υποδηλώσουν έναν αμοιβαίο αποκλεισμό μεταξύ δύο ή περισσοτέρων ακολουθιών από μηνύματα. Επιτρέπουν τη μοντελοποίηση της κλασσικής "ifthenelse" λογικής. Ένα παράδειγμα διαγράμματος ακολουθίας όπου χρησιμοποιείται ο συμβολισμός του συνδυασμένου τμήματος των εναλλακτικών παρουσιάζεται στην *Εικόνα 19*

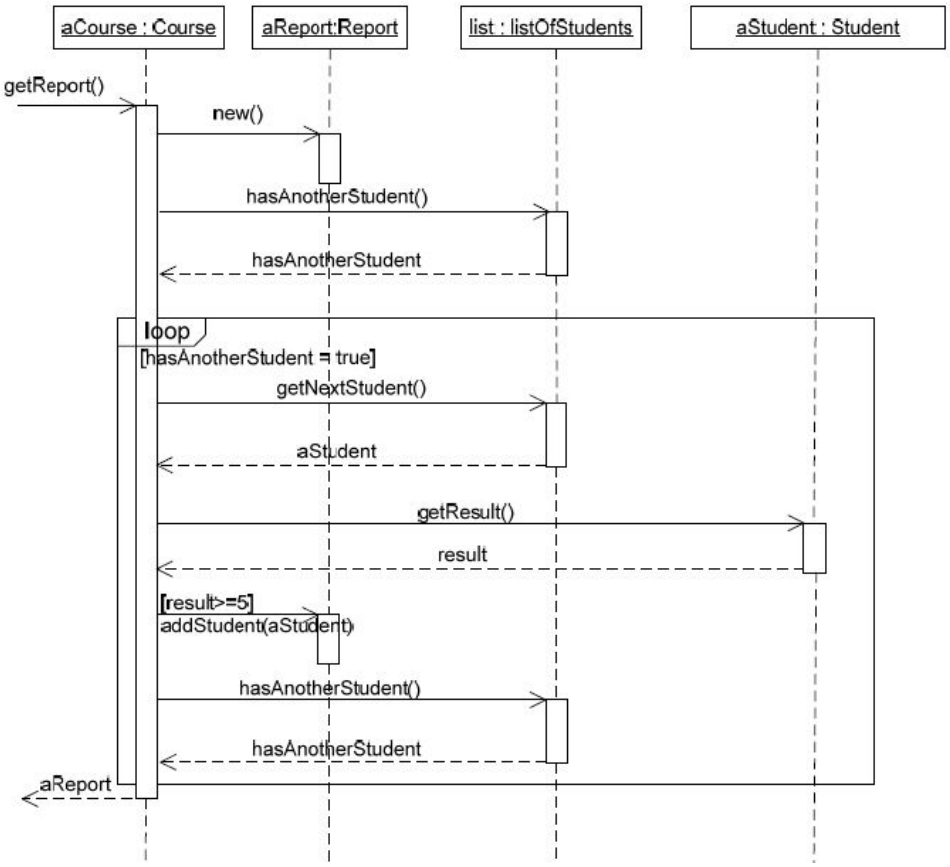


Εικόνα 19: Αναπαράσταση λογικής if/else σε διάγραμμα ακολουθίας

Το συνδυασμένο τμήμα εναλλακτικών αναπαρίσταται ως ένα πλαίσιο με το όνομα "alt". Το ευρύτερο ορθογώνιο διαχωρίζεται σε δύο περιοχές που σύμφωνα με τη UML 2 ονομάζονται τελεστέοι. Οι τελεστέοι διαχωρίζονται από μία διακεκομμένη γραμμή. Σε κάθε τελεστέο υπάρχει μία συνθήκη-φρουρός (σημειώνεται μέσα σε [ ]). Αν κατά την εκτέλεση του σεναρίου η τιμή της συνθήκης-φρουρού αποτιμάται ως αληθής εκτελείται ο αντίστοιχος τελεστέος. Στο ανωτέρω διάγραμμα, αν το υπόλοιπο του λογαριασμού (balance) είναι μεγαλύτερο ή ίσο προς το ποσό που ζητείται από το τσεκ (amount), αποστέλλεται το μήνυμα registerTransaction() από το αντικείμενο bank προς το αντικείμενο account. Σε αντίθετη περίπτωση, αποστέλλεται το μήνυμα transactionCancelled() από το αντικείμενο bank προς το αντικείμενο check.

2.13.3 Βρόχοι

Οι βρόχοι (loops) χρησιμοποιούνται για να υποδηλώσουν μια επαναληπτική διαδικασία. Επιτρέπουν τη μοντελοποίηση της κλασσικής λογικής "while(συνθήκη)". Ένα παράδειγμα διαγράμματος ακολουθίας όπου χρησιμοποιείται ο συμβολισμός του συνδυασμένου τμήματος των βρόχων παρουσιάζεται στην *Εικόνα 30*



Εικόνα 20: Αναπαράσταση επαναληπτικών δομών σε διάγραμμα ακολουθίας

Το συνδυασμένο τμήμα βρόχου αναπαρίσταται ως ένα πλαίσιο με το όνομα "loop". Στο πάνω αριστερό τμήμα του πλαισίου τοποθετείται μία συνθήκη-φρουρός. Η ακολουθία των μηνυμάτων που περιλαμβάνονται στο πλαίσιο εκτελείται επαναληπτικά όσο η τιμή της συνθήκης αποτιμάται ως αληθής. Στο ανωτέρω παράδειγμα όσο υπάρχουν επιπλέον φοιτητές (hasAnotherStudent = true), το αντικείμενο aCourse αποστέλλει το μήνυμα getNextStudent() στο αντικείμενο list, getResult() στο αντικείμενο aStudent, το μήνυμα addStudent() στο αντικείμενο aReport (εάν και μόνο αν η τιμή result είναι μεγαλύτερη του 5) και τέλος το μήνυμα hasAnotherStudent() στο αντικείμενο list για να ενημερωθεί η τιμή βάσει της οποίας γίνεται ο έλεγχος της συνθήκης για τις επαναλήψεις.

Για τις περιπτώσεις χρήσης του συστήματος ηλεκτρονικής αξιολόγησης, δημιουργούνται τα ακόλουθα διαγράμματα ακολουθίας χρησιμοποιώντας ως οδηγό τα αντίστοιχα διαγράμματα ευρωστίας (για το πρώτο διάγραμμα η κατάστρωση του παρουσιάζεται αναλυτικά).

Β. ΕΙΔΙΚΟ ΜΕΡΟΣ

3. Απαιτήσεις της Εφαρμογής

Το λογισμικό που πρόκειται να αναπτυχθεί αφορά μια διαδικτυακή εφαρμογή οι όποια να επιτρέπει την αξιολόγηση καθηγητών - φοιτητών- υποδομών (ενιαίο ερωτηματολόγιο) του ΤΕΙ ΠΑΤΡΩΝ από τους φοιτητές αυτού. Το σύστημα θα είναι το ίδιο για όλα τα τμήματα του ΤΕΙ ΠΑΤΡΩΝ. Οι φοιτητές εισέρχονται προς συγκεκριμένο χρονικό διάστημα στην εφαρμογή αύτη μέσω (LDAP) και έχουν την δυνατότητα να δουν και να αξιολογήσουν τα μαθήματα την σχολής τους που είναι εγγεγραμμένοι (λίστα μαθημάτων).

Κάθε φοιτητής μπορεί να κάνει αξιολόγηση σε κάθε μάθημα όσες φόρες θέλει κάνοντας αποθήκευση των στοιχείων που έχει εισάγει μέχρι να κάνει οριστική υποβολή . Μετά την οριστική υποβολή η εφαρμογή δεν του επιτρέπει πλέον την εισαγωγή στο συγκεκριμένο μάθημα . Αν τυχόν κάποια αξιολόγηση δεν έχει ολοκληρώσει την οριστική υποβολή του μέσα στο χρονικό διάστημα της αξιολόγησης τότε θα ακυρώνεται. Η εφαρμογή θα πρέπει να δίνει βοήθεια στον χρήστη - φοιτητή σε κάθε βήμα και στάδιο της. Η εφαρμογή θα πρέπει να δίνει στατιστικά αποτελέσματα (μέσους όρους - τοπικές απόκλισης - πινάκες συχνοτήτων) τουλάχιστον για όλες τις επιλογές του ερωτηματολογίου. Αυτά τα αποτελέσματα τα παίρνει η Μονάδας Διασφάλισης Ποιότητας (ΜΟ.ΔΙ.Π) για να τα αξιοποιήσει στον μέλλον.

Το LDAP είναι ένα πρωτόκολλο ανοικτού προτύπου για την πρόσβαση σε υπηρεσίες καταλόγου X.500. Το πρωτόκολλο τρέχει πάνω από το επίπεδο μεταφοράς ενός δικτύου, στην περίπτωση του Διαδικτύου αυτό είναι το TCP. Χρησιμοποιεί τη δικτυακή διαστρωμάτωση TCP/IP για τα επίπεδα δικτύου και μεταφοράς, σε αντίθεση με την περίπλοκη διαστρωμάτωση του μοντέλου OSI. Επίσης υιοθετεί και άλλες απλουστεύσεις, όπως η αναπαράσταση τιμών γνωρισμάτων και δομές πληροφορίας του πρωτοκόλλου ως αλφαριθμητικά κειμένου (strings), τα οποία σχεδιάστηκαν ώστε να γίνεται η υλοποίηση περισσότερο απλή και εύκολη. Οι υπηρεσίες καταλόγου είναι μία βάση δεδομένων η οποία οργανώνει εγγραφές, και είναι βελτιστοποιημένη για διαδικασίες ανάγνωσης και αναζήτησης.

3.1Ανάλυση Απαιτήσεων

Από την περιγραφή απαιτήσεων υψηλού επιπέδου που δόθηκε, προκύπτει η ακόλουθη αρχική λίστα ουσιαστικών και πιθανών κλάσεων πεδίου προβλήματος (έχοντας μετατρέψει το πρόσωπο σε πρώτο ενικό):

Λίστα Ουσιαστικών	
Σύστημα αξιολόγησης	Καθηγητές
Φοιτητής	Υποδομές
Μάθημα	ΜΟ.ΔΙ.Π
Τμήμα	Ερωτηματολόγιο
Λίστα μαθημάτων	Αποτελέσματα
Επιλογή μαθημάτων	

Με βάση το ανωτέρω πίνακα οι υποψήφιες κλάσεις του πεδίου προβλήματος είναι:

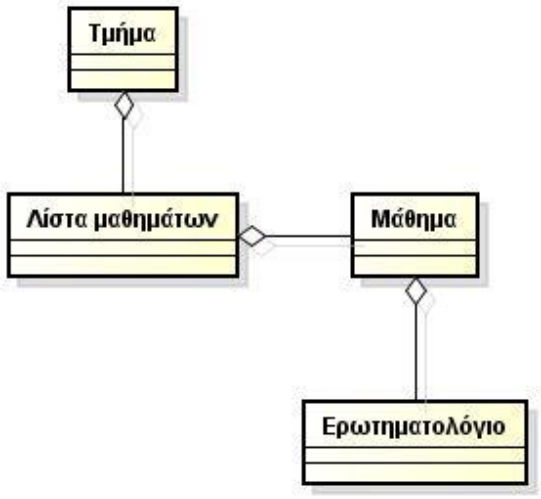
Υποψήφιες κλάσεις
Μάθημα
Ερωτηματολόγιο
Λίστα μαθημάτων
Τμήμα

Πρωταρχικός στόχος κατά την κατασκευή του μοντέλου του πεδίου προβλήματος είναι ο εντοπισμός σχέσεων μεταξύ των υποψηφίων κλάσεων Ωστόσο, καθώς στην αντικειμενοστραφή σχεδίαση υπάρχουν πολλά είδη σχέσεων μεταξύ κλάσεων, δεν κρίνεται σκόπιμο στο στάδιο αυτό να απεικονιστούν ιδιαίτερες λεπτομέρειες. Συνήθως αρκεί η απεικόνιση στο μοντέλο σχέσεων τύπου "έχει" (has) και σχέσεων τύπου "είναι" (is). Το πρώτο είδος αναφέρεται σε σχέσεις περιεκτικότητας μεταξύ δύο κλάσεων που στην απλούστερη μορφή καλούνται συναρμολογήσεις (ή συσσωματώσεις). Μια συναρμολόγηση υποδηλώνει ότι αντικείμενα μιας κλάσης περιέχουν αντικείμενα μιας άλλης κλάσης. Η συναρμολόγηση απεικονίζεται ως μια γραμμή μεταξύ των κλάσεων με έναν λευκό ρόμβο στο άκρο της περιέχουσας κλάσης. Το δεύτερο είδος αναφέρεται σε σχέσεις κληρονομικότητας μεταξύ κλάσεων. Μια σχέση κληρονομικότητας υποδηλώνει ότι μια κλάση (υποκλάση)

αποτελεί ειδικότερη κατηγορία μιας άλλης (υπερκλάση) και κληρονομεί τις ιδιότητες και τη συμπεριφορά της. Η κληρονομικότητα απεικονίζεται ως μια γραμμή με ένα τρίγωνο στο άκρο της υπερκλάσης.

Από το κείμενο των απαιτήσεων υψηλού επιπέδου προκύπτει ότι μια Λίστα μαθημάτων μπορεί να περιλαμβάνει Μαθήματα, το Τμήμα περιλαμβάνει τις Λίστες μαθημάτων και ένα Μάθημα περιλαμβάνει τα ερωτήματα οποία αποτελείται το Ερωτηματολόγιο. Σχέσεις κληρονομικότητας δεν εντοπίζονται με βάση την περιγραφή που δόθηκε καθώς δεν υπάρχει κάποια οντότητα που να αποτελεί ειδικότερη κατηγορία κάποιας άλλης.

Με βάση τις ανωτέρω πληροφορίες το αρχικό διάγραμμα του πεδίου προβλήματος που προκύπτει είναι :

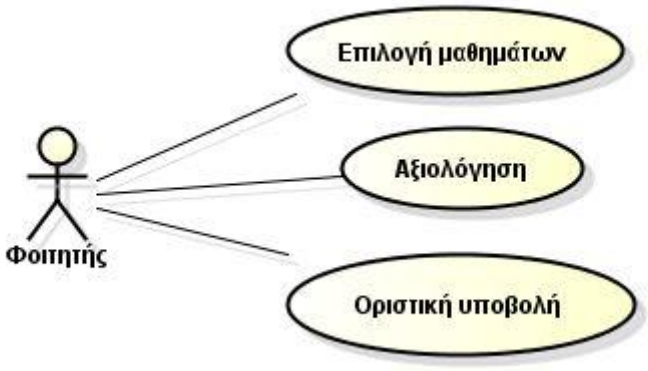


Εικόνα 21.Διάγραμμα Πεδίου Προβλήματος (αρχικό)

3.2 Περιπτώσεις χρήσης

Το μοντέλο περιπτώσεων χρήσης είναι μείζονος σημασίας στις αντικειμενοστραφείς μεθοδολογίες ανάπτυξης λογισμικού όπως η ICONIX και η RUP καθώς θέτει τις βάσεις για τη δημιουργία συστημάτων που δίνουν έμφαση στις απαιτήσεις του πελάτη. Στο μοντέλο περιπτώσεων χρήσης καταγράφονται οι απαιτήσεις των χρηστών διερευνώντας εξαντλητικά όλα τα πιθανά σενάρια χρήσης του συστήματος. Μία περίπτωση χρήσης (use case) είναι μια ακολουθία ενεργειών που ένας χρήστης του συστήματος (συνήθως πρόκειται για άνθρωπο αλλά μπορεί να είναι οποιαδήποτε εξωτερική οντότητα όπως ένα άλλο σύστημα) πραγματοποιεί στο σύστημα για να επιτύχει ένα συγκεκριμένο σκοπό. Μία περίπτωση χρήσης απεικονίζεται διαγραμματικά ως μία έλλειψη, μέσα στην οποία αναγράφεται το όνομα της. Το σύνολο των περιπτώσεων χρήσης ενός συστήματος συνιστούν το διάγραμμα περιπτώσεων χρήσης. Στα διαγράμματα αυτά είναι σημαντικό εκτός από τις περιπτώσεις χρήσης να απεικονιστούν οι χρήστες του συστήματος που συμμετέχουν σε κάθε περίπτωση. Οι χρήστες απεικονίζονται ως σχηματικά ανθρωπάκια (stick persons). Η συσχέτιση μεταξύ χρήστη και περίπτωσης χρήσης απεικονίζεται με μια γραμμή μεταξύ τους (η οποία καλό είναι να μην έχει οποιαδήποτε κατεύθυνση για την αποφυγή παρερμηνειών).

Για το σύστημα διαχείρισης Ερωτηματολόγιου το διάγραμμα περιπτώσεων χρήσης φαίνεται στην εικόνα 19:



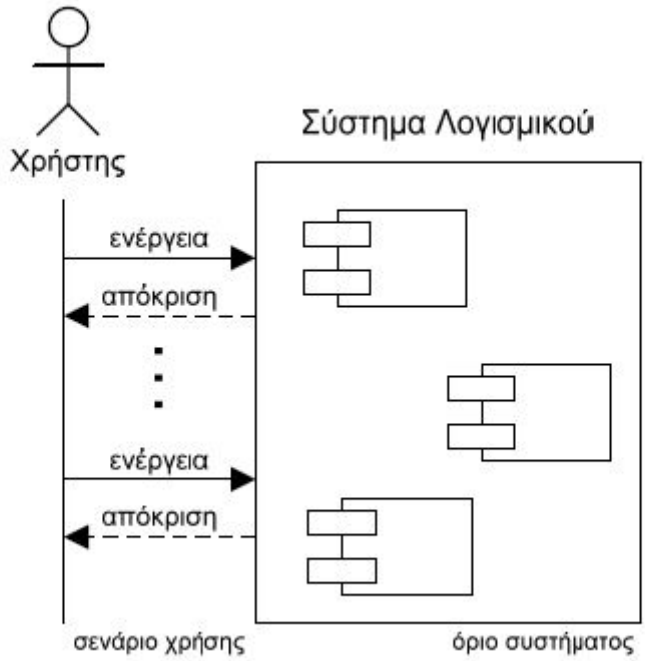
Εικόνα 22.Διάγραμμα Περιπτώσεων Χρήσης Συστήματος

Ο ορισμός μιας περίπτωσης χρήσης περιλαμβάνει όλες τις δυνατές συμπεριφορές, περιλαμβανομένης της κανονικής ή βασικής ροής (όπου για παράδειγμα εκτελούνται όλα όσα αναμένει ο χρήστης για την επίτευξη του στόχου) αλλά και όλων των "εναλλακτικών" ροών, όπου κάτι για παράδειγμα αποκλίνει από την "κανονική" συμπεριφορά. Μια περίπτωση χρήσης είναι επιτυχής όταν προδιαγράφει μόνο "ποια" είναι η συμπεριφορά του συστήματος χωρίς καμία αναφορά στο "πώς" αυτή επιτυγχάνεται, χωρίς δηλαδή περιγραφή τεχνικών λεπτομερειών υλοποίησης.

3.2.1 Τεκμηρίωση περιπτώσεων χρήσης

Προφανώς, διαγράμματα όπως στην εικόνα 19 παρέχουν λίγη πληροφορία σχετικά με τις απαιτήσεις από το σύστημα. Για το λόγο αυτό κάθε περίπτωση χρήσης πρέπει να συνοδεύεται από κατάλληλη τεκμηρίωση, όπου καταγράφονται τόσο η βασική όσο και όλες οι εναλλακτικές ροές. Για την τεκμηρίωση των περιπτώσεων χρήσης συνιστάται η τήρηση των εξής κανόνων:

- η φιλοσοφία της περιγραφής πρέπει να εστιάζει στην περιγραφή ενός σεναρίου χρήσης ως σύνολο ενεργειών-αποκρίσεων. Με άλλα λόγια, ο χρήστης πραγματοποιεί κάποια ενέργεια, το σύστημα αποκρίνεται, ο χρήστης προβαίνει σε κάποια νέα ενέργεια κ.ο.κ. μέχρις ότου επιτευχθεί ο σκοπός για τον οποίο ο χρήστης αξιοποιεί το σύστημα λογισμικού (Εικόνα 23)



Εικόνα 23.Περίπτωση χρήσης ως σύνολο ενεργειών/αποκρίσεων

- Οι προτάσεις είναι καλό να είναι γραμμένες σε ενεργητική φωνή, στον ενεστώτα και να έχουν τη μορφή υποκείμενο-ρήμα-αντικείμενο. Για παράδειγμα, Ο Φοιτητής επιλέγει το πλήκτρο "Σύνδεση" (Login).Η μορφή αυτή συμβάλλει στην αποφυγή περιγραφής τεχνικών λεπτομερειών του συστήματος και κυρίως επιτρέπει την ευκολότερη μετάβαση στα επόμενα στάδια της ανάλυσης και σχεδίασης. Στην ιδανική περίπτωση το υποκείμενο και το αντικείμενο κάθε πρότασης αντιστοιχούν σε αντικείμενα των κλάσεων του συστήματος και το ρήμα στο μήνυμα που ανταλλάσσεται μεταξύ τους για την επίτευξη της επιθυμητής λειτουργικότητας.
- Είναι επιθυμητό να χρησιμοποιούνται στη διατύπωση των περιπτώσεων χρήσης όσο το δυνατόν περισσότερο οι όροι του μοντέλου του πεδίου προβλήματος. Λαμβάνοντας υπόψη ότι ένα αντικειμενοστραφές σύστημα είναι κατ' ουσίαν τα αντικείμενα των κλάσεων του πεδίου προβλήματος (που βεβαίως εξελίσσεται και εμπλουτίζεται), είναι χρήσιμο να διατυπωθούν οι απαιτήσεις του προβλήματος όσο γίνεται πιο πρώιμα με τους όρους αυτών των αντικειμένων.
- Η περιγραφή της περίπτωσης χρήσης μπορεί να περιλαμβάνει αναφορές σε βασικά στοιχεία της γραφικής διασύνδεσης χρήστη (όπως οθόνες, πλήκτρα κλπ). Για το λόγο αυτό, είναι εξαιρετικά χρήσιμο σε πολλές περιπτώσεις, πριν

από την καταγραφή των περιπτώσεων χρήσης, να δημιουργηθούν σε συνεργασία με τους τελικούς χρήστες πρόχειρα σχέδια της αναμενόμενης γραφικής διασύνδεσης, τα οποία αναφέρονται στη βιβλιογραφία ως *πρωτότυπα* (prototypes). Στα πλαίσια αυτά, τα στοιχεία της γραφικής διασύνδεσης (πρωτότυπα) που χρησιμοποιούνται πρέπει να αναφέρονται με κάποιο όνομα και όχι αφηρημένα (π.χ. *ο χρήστης εισάγει στην Οθόνη Σύνδεση το όνομα του* και όχι *ο χρήστης εισάγει σε κάποια οθόνη*…). Ωστόσο, δεν θα πρέπει να γίνεται αναφορά σε τεχνικές ή σχεδιαστικές λεπτομέρειες καθώς ο στόχος της χρήσης πρώιμων δειγμάτων της γραφικής διασύνδεσης είναι αποκλειστικά η διερεύνηση των απαιτήσεων του πελάτη.

- Τέλος, σημειώνεται ότι στο στάδιο της καταγραφής των περιπτώσεων χρήσης είναι πολλές φορές κουραστική η αναζήτηση όλων των πιθανών εναλλακτικών ροών που μπορούν να εμφανιστούν σε ένα σενάριο χρήσης. Ωστόσο, κρίνεται ιδιαιτέρως σημαντικό η διερεύνηση αυτή να γίνει σε αυτό το στάδιο, παρά σε μεταγενέστερο στάδιο όπως η σχεδίαση ή η υλοποίηση. Η έγκαιρη διατύπωση όλων των απαιτήσεων είναι κατά πολύ οικονομικότερη και ασφαλέστερη από την τροποποίηση του σχεδίου ή του κώδικα στη συνέχεια.

3.2.2 Πρότυπα τεκμηρίωσης περιπτώσεων χρήσης

Στη βιβλιογραφία έχουν προταθεί διάφορα πρότυπα για την τεκμηρίωση των περιπτώσεων χρήσης. Τα πρότυπα μπορούν να ομαδοποιηθούν σε τρεις γενικές κατηγορίες:

1. Απλές περιγραφές κειμένου χωρίς ιδιαίτερη δομή. Οι υποστηρικτές αυτού του προτύπου δίνουν έμφαση στο ότι οι περιπτώσεις χρήσης πρέπει να εστιάζουν στις απαιτήσεις του πελάτη και να μην εμπλέκουν τον αναλυτή του συστήματος σε άλλες (μη απαιτούμενες) δραστηριότητες. Προτείνεται η περιγραφή να μην ξεπερνά τις δύο παραγράφους ανά περίπτωση χρήσης. Το πρότυπο αυτό χρησιμοποιείται στη συνέχεια για την τεκμηρίωση των περιπτώσεων χρήσης του συστήματος αξιολόγησης.

2. Πιο εκτεταμένες περιγραφές όπου διατυπώνεται ρητά ποια είναι η βασική ροή και ποιες είναι οι εναλλακτικές ροές. Συνήθως χρησιμοποιείται αρίθμηση για κάθε ενέργεια του χρήστη ή απόκριση του συστήματος. Σε περίπτωση εναλλακτικών ροών

χρησιμοποιείται ο αντίστοιχος αριθμός για να υποδηλώσει το στάδιο του σεναρίου χρήσης όπου αυτή εφαρμόζεται. Ένα τέτοιο παράδειγμα είναι:

Βασική Ροή

1. Ο χρήστης επιλέγει το πλήκτρο "Πληρωμή μέσω Κάρτας"
2. Το σύστημα εμφανίζει την οθόνη "Πληρωμή μέσω Κάρτας"
3. Ο χρήστης εισάγει τον αριθμό της κάρτας και το ποσό και επιλέγει Αποστολή
4. Το σύστημα εμφανίζει μήνυμα ολοκλήρωσης της συναλλαγής

Εναλλακτική Ροή

4.α.1 Ο αριθμός της κάρτας δεν είναι έγκυρος

4.α.2 Το σύστημα εμφανίζει μήνυμα σφάλματος

4.α.3 Η περίπτωση χρήσης συνεχίζεται από το βήμα 2 της βασικής ροής

3. Λεπτομερή πρότυπα όπου για κάθε περίπτωση χρήσης αναφέρονται:

α. **Σύντομη περιγραφή**

β. **Προ-συνθήκες**. Συνθήκες που πρέπει να ισχύουν ώστε να είναι δυνατή η έναρξη της περίπτωσης χρήσης.

γ. **Βασική Ροή**

δ. **Εναλλακτικές Ροές**

ε. **Μετά-συνθήκες**. Συνθήκες που θα ισχύουν μετά την ολοκλήρωση της περίπτωσης χρήσης.

3.3 Ενδεικτικές Οθόνες του Συστήματος

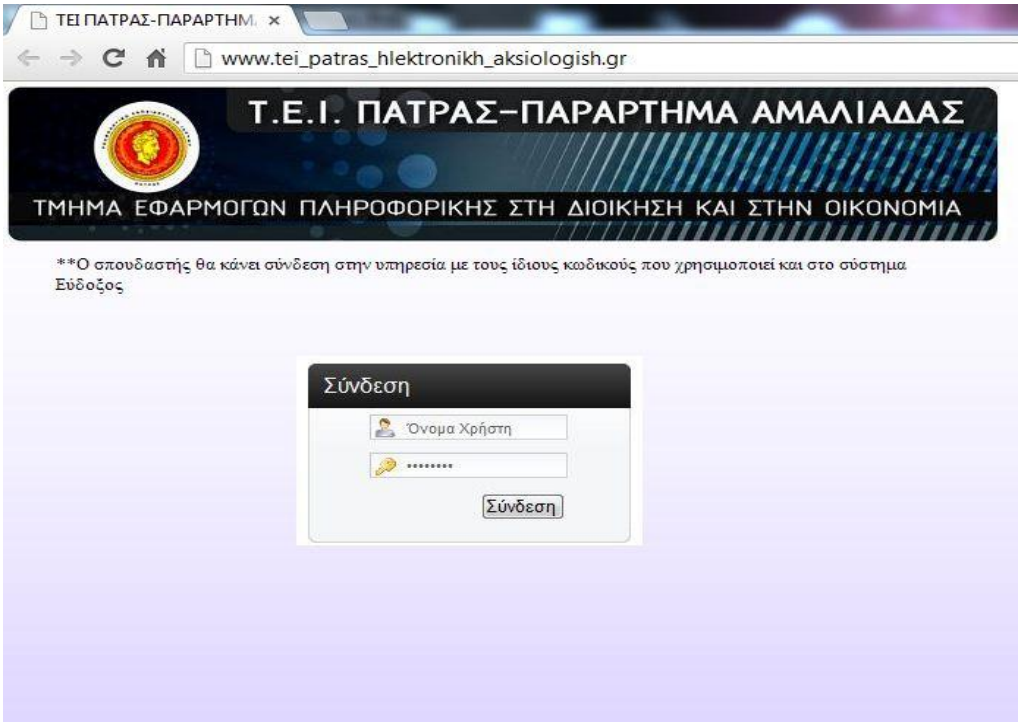
Στη λεκτική περιγραφή των περιπτώσεων χρήσης που ακολουθεί γίνεται αναφορά σε οθόνες του συστήματος που θα αναπτυχθεί. Η πρόχειρη σχεδίαση διεπαφών (οθονών) αποτελεί τμήμα της ανάλυσης των απαιτήσεων, όπου επιχειρούμε να δείξουμε στον μελλοντικό χρήστη του συστήματος την αναμενόμενη συμπεριφορά του υπό ανάπτυξη συστήματος λογισμικού. Τα σχέδια αυτά δεν αποτελούν μια λεπτομερή και ακριβή αποτύπωση της γραφικής διασύνδεσης χρήστη που θα έχει τελικά το λογισμικό που θα αναπτυχθεί. Απλά θεωρείται ότι αποτελούν ένα μέσο για την καλύτερη δυνατή συνεννόηση μεταξύ του τελικού χρήστη και του αναλυτή με σκοπό την αποσαφήνιση της λειτουργικότητας και τη διευκρίνιση τυχόν ασαφειών στις απαιτήσεις.

Οι ενδεικτικές οθόνες που παρουσιάζονται στη φάση αυτή δεν είναι αναλυτικές (δηλαδή δεν περιλαμβάνονται όλα τα πλήκτρα, πεδία, χρώματα, μηνύματα κλπ), καθώς κάτι

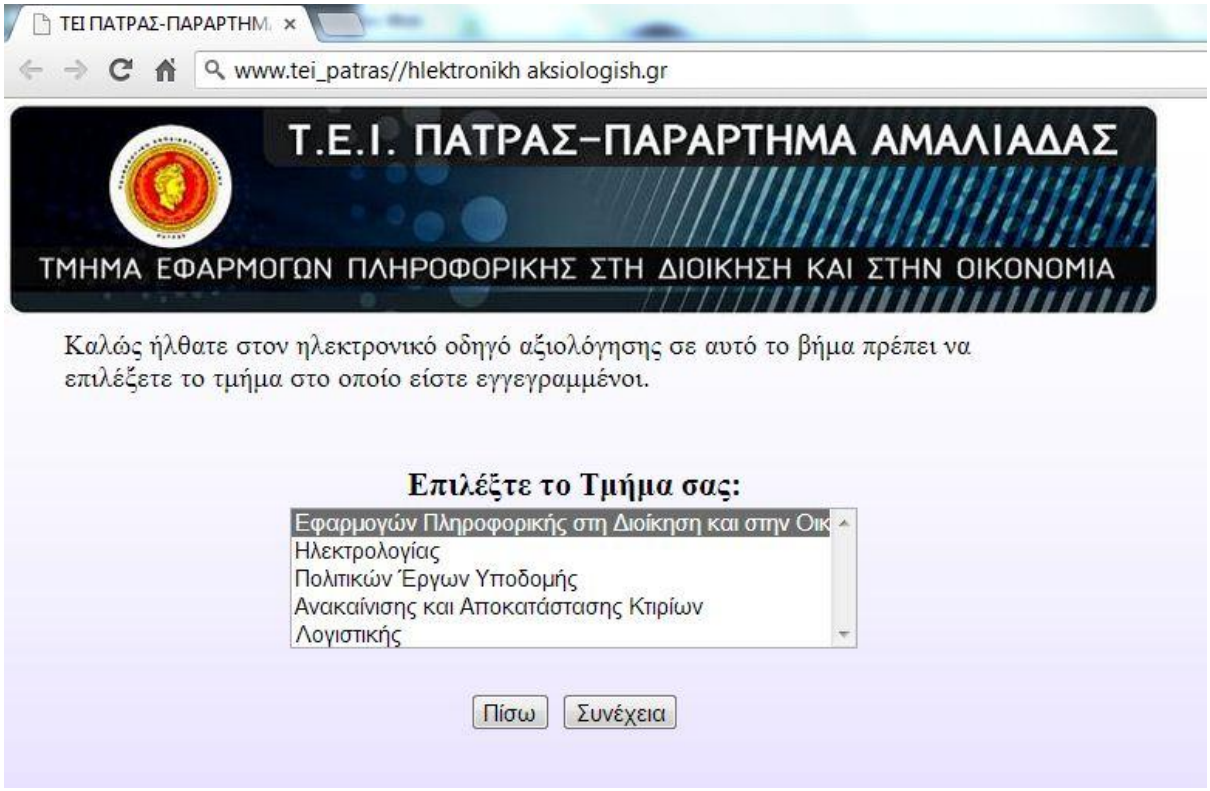
τέτοιο θα ήταν δεσμευτικό για την υλοποίηση, στοιχείο που δεν είναι επιθυμητό στο στάδιο της ανάλυσης των απαιτήσεων. Επίσης, οι οθόνες δεν καλύπτουν το σύνολο των διεπαφών μεταξύ χρήστη και συστήματος αλλά τις διεπαφές εκείνες που κρίνεται σκόπιμο να διερευνηθούν. Οι οθόνες αυτές μπορούν να δημιουργηθούν είτε με εργαλεία γραφικής σχεδίασης, είτε με γλώσσες ταχείας προτυποποίησης που επιτρέπουν την εύκολη ανάπτυξη γραφικής διασύνδεσης είτε ακόμη και ως πρόχειρα σχέδια που μπορούν να σκαναριστούν.

Οι ενδεικτικές οθόνες που παρουσιάζονται στη συνέχεια είναι:

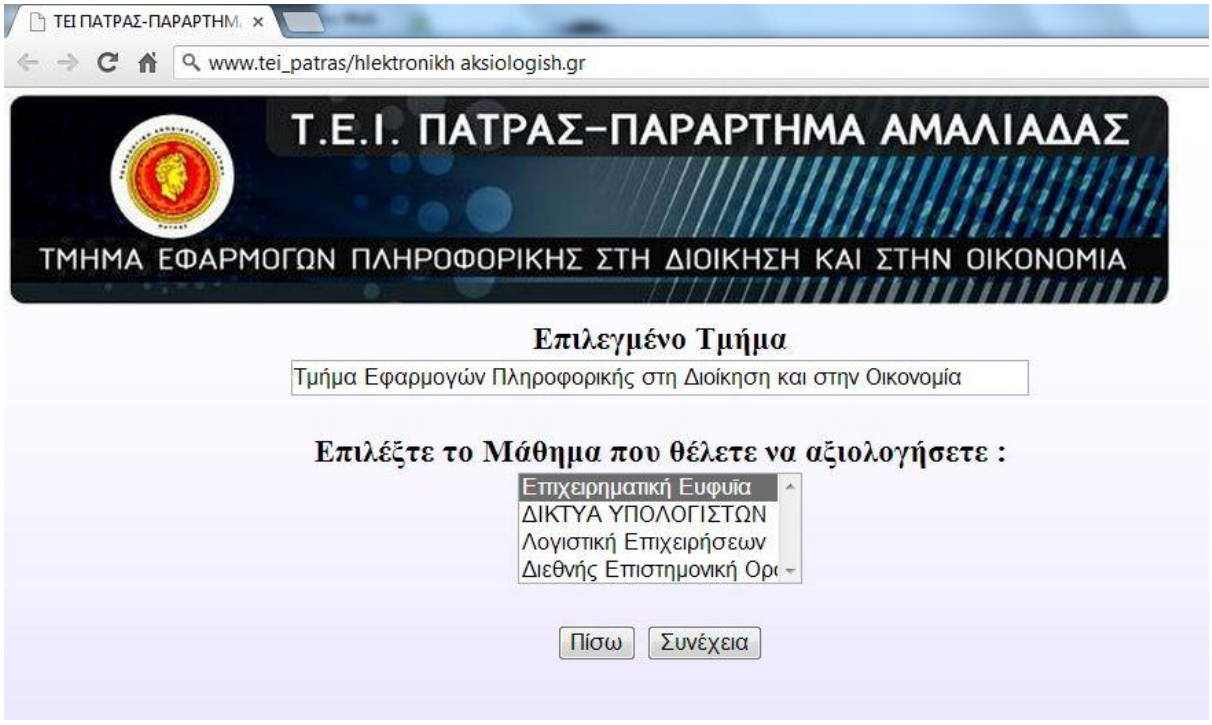
- η Κύρια Οθόνη της εφαρμογής (είναι η οθόνη σύνδεσης του χρηστή στη εφαρμογή),
- η οθόνη "Επιλογή Τμήματος" που επιτρέπει στο χρηστή να επιλέξει το τμήμα του,
- η οθόνη "Επιλογή Μαθήματος" που επιτρέπει στο χρηστή να επιλέξει το μάθημα,
- η οθόνη "Αξιολόγηση Μαθήματος" που επιτρέπει στο χρηστή να αξιολόγηση το μάθημα,
- η οθόνη "Οριστική υποβολή" που επιτρέπει στο χρηστή να κάνει αποθήκευση η οριστική υποβολή της αξιολόγησης.
- η οθόνη "Ολοκλήρωσης αξιολόγησης" είναι η οθόνη που επιβεβαιώνει την αποθήκευση η την οριστική υποβολή της αξιολόγησης.



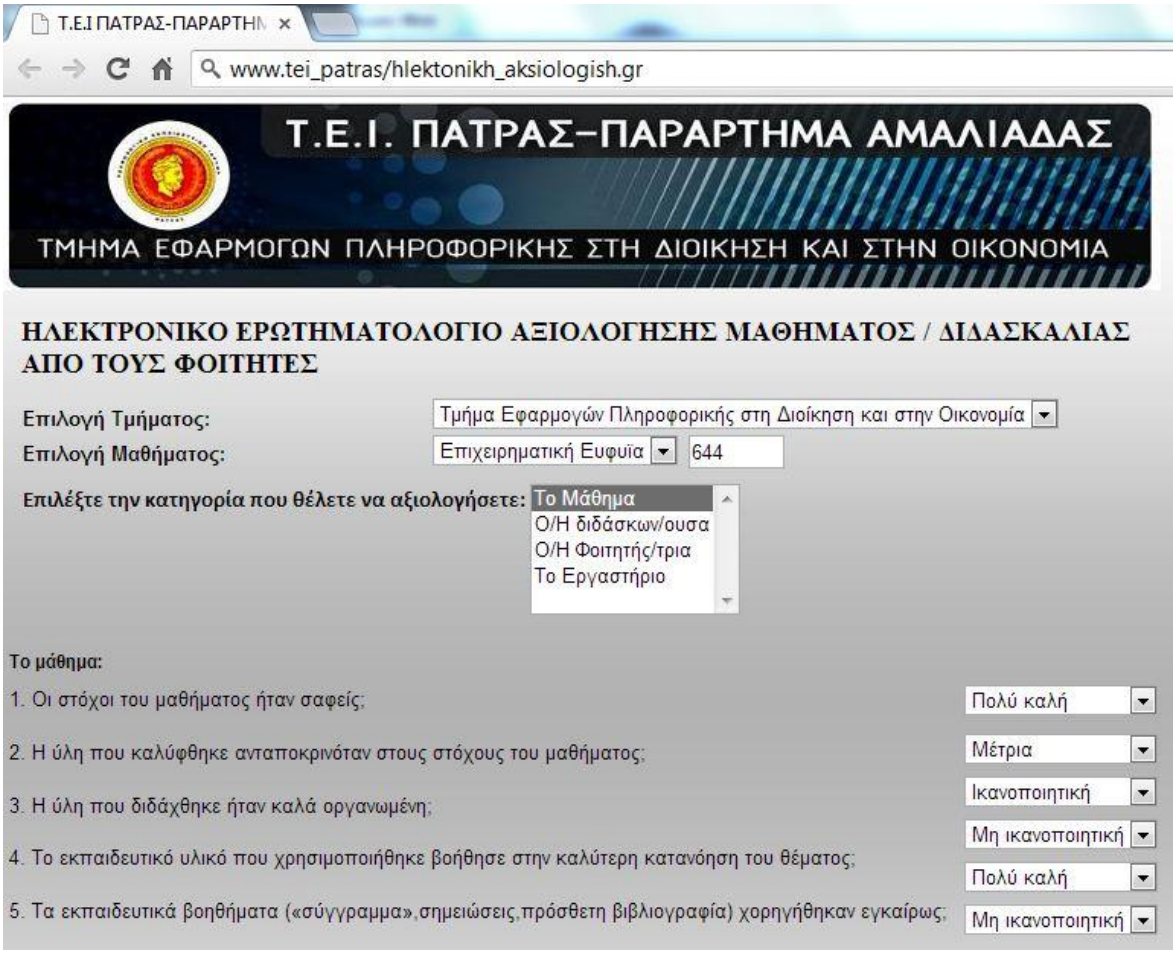
Εικόνα 24. "Οθόνη σύνδεσης"



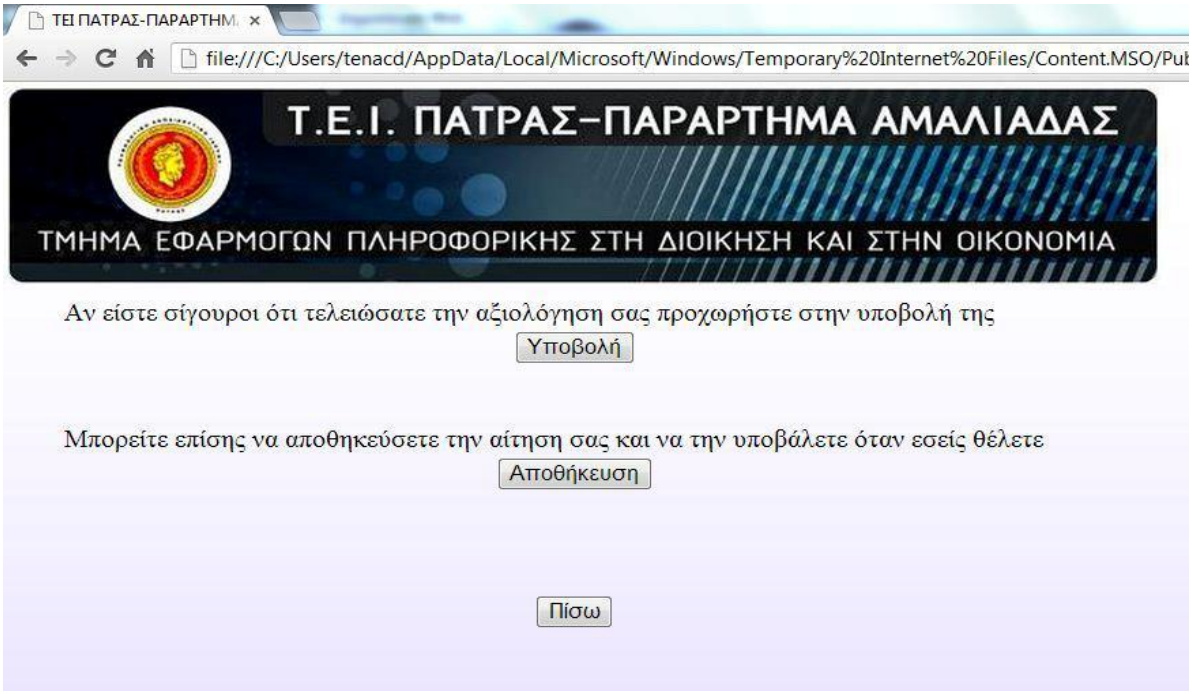
Εικόνα 25."Επιλογή Τμήματος"



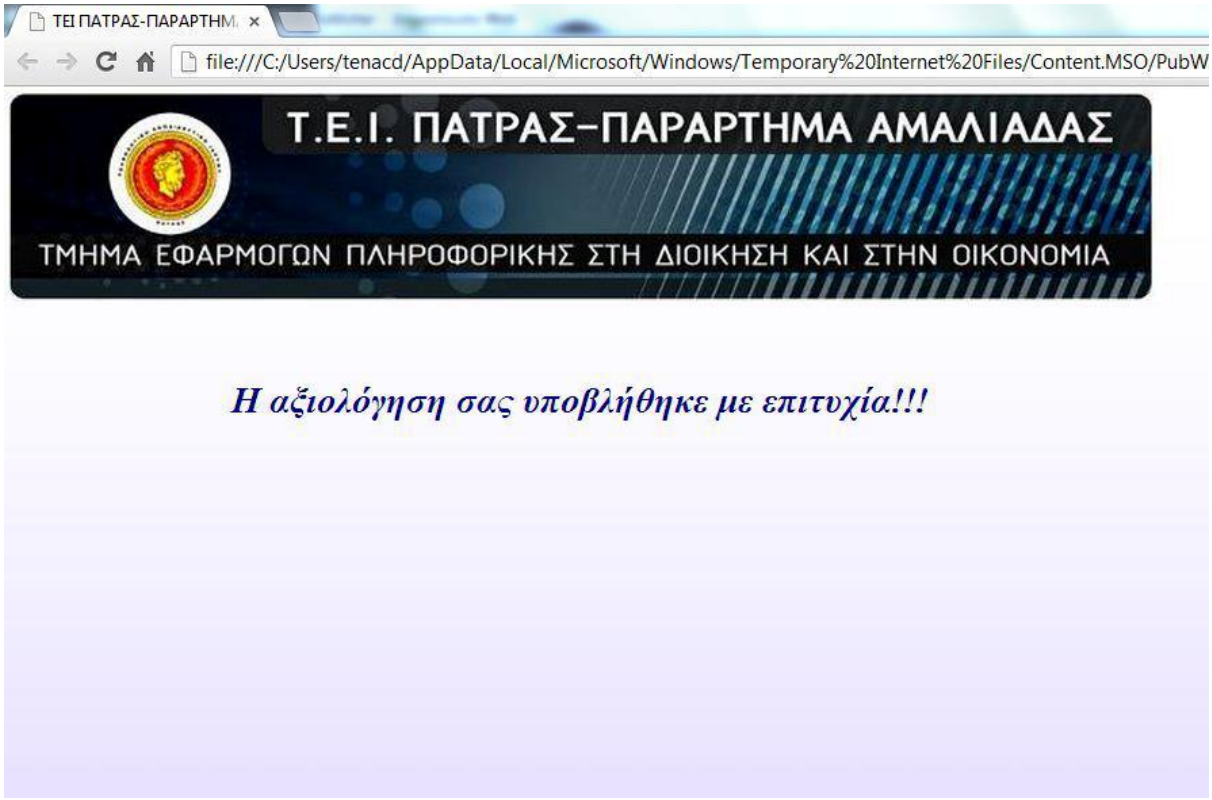
Εικόνα 26."Επιλογή Μαθήματος"



Εικόνα 27."Αξιολόγηση Μαθήματος"



Εικόνα 28."Οριστική υποβολή"



Εικόνα 29."Ολοκλήρωσης αξιολόγησης"

3.3.1 Τεκμηρίωση Περιπτώσεων Χρήσης

Στη συνέχεια παρουσιάζεται η τεκμηρίωση των περιπτώσεων χρήσης του συστήματος Αξιολόγησης σύμφωνα με το πρώτο και το δεύτερο πρότυπο. Η ανάλυση βασίζεται στις προδιαγραφές υψηλού επιπέδου που διατυπώθηκαν από τον πελάτη αλλά και σε (υποθετική) διευκρίνιση ασαφειών που πραγματοποιήθηκε σε συνεργασία με τους τελικούς χρήστες του συστήματος. Χάριν απλότητας, οι εναλλακτικές ροές που έχουν καταγραφεί είναι περιορισμένες.

3.4 Περίπτωση χρήσης 1:Δημιουργία και αποστολή αξιολόγησης

3.4.1 Πρότυπο 1

Με την εισαγωγή του χρήστη στην υπηρεσία το σύστημα εμφανίζει την οθόνη ‘Οθόνη σύνδεσης’. Ο χρήστης καταχωρεί τα προσωπικά στοιχεία user name και password που του έχουν δοθεί από την γραμματεία του τμήματος του, για να συνδεθεί στην υπηρεσία. Αφού γίνει η σύνδεση, το σύστημα εμφανίζει την οθόνη ‘Επιλογή Τμήματος’ στην οποία ο χρήστης επιλέγει το τμήμα στο οποίο είναι εγγεγραμμένος. Μόλις ο χρήστης επιλέξει το πλήκτρο το σύστημα εμφανίζει την οθόνη ‘Επιλογή Μαθήματος’ όπου ο χρήστης επιλέγει το μάθημα που θέλει να αξιολογήσει.

Στην συνέχεια ο χρήστης επιλέγει το πλήκτρο Επόμενο και το σύστημα εμφανίζει την οθόνη ‘Αξιολόγηση Μαθήματος’. Εκεί ο χρήστης επιλέγει την κατηγορία που θέλει να αξιολογήσει ‘Το μάθημα’ ‘Το Εργαστήριο’, ’Ο/Η Φοιτητής/τρια’, ’Ο/Η Διδάσκον/ουσα’. Έτσι ανάλογα με την επιλογή του χρήστη εμφανίζετε και το αντίστοιχο ερωτηματολόγιο με την δυνατότητα αξιολόγησης στις ερωτήσεις σε «Μη ικανοποιητική» «Μέτρια» «Καλή» «Πολύ καλή». Έτσι όταν ο χρήστης ολοκληρώσει την αξιολόγηση και επιλέξει το πλήκτρο επόμενο το σύστημα του εμφανίζει την οθόνη ‘Οριστική υποβολή’ σε αυτό το σημείο ο χρήστης επιλέγει το πλήκτρο ‘Υποβολή’ και το σύστημα στέλνει την αξιολόγηση στην βάση δεδομένων της γραμματείας. Το σύστημα εμφανίζει την οθόνη ‘Ολοκλήρωση Αξιολόγησης’ με το μήνυμα «Η αξιολόγηση σας υποβλήθηκε με επιτυχία»

3.4.2 Βασική ροή

- Ο χρήστης κάνει σύνδεση στο σύστημα με του προσωπικούς του κωδικούς στην ‘Οθόνη σύνδεσης’
- Το σύστημα εμφανίζει την οθόνη ‘Επιλογή Τμήματος’
- Ο χρήστης επιλέγει το τμήμα στο οποίο είναι εγγεγραμμένος και επιλέγει το πλήκτρο επόμενο
- Το σύστημα εμφανίζει την οθόνη ‘Επιλογή Μαθήματος’
- Ο χρήστης επιλέγει το μάθημα που θέλει να αξιολογήσει και επιλέγει το πλήκτρο επόμενο
- Το σύστημα εμφανίζει την οθόνη ‘Αξιολόγηση Μαθήματος’
- Ο χρήστης επιλέγει ποια κατηγορία θέλει να αξιολογήσει ανάμεσα στις κατηγορίες ‘Το μάθημα’ ‘Το Εργαστήριο’, ’Ο/Η Φοιτητής/τρια’, ’Ο/Η Διδάσκον/ουσα’.
- Το σύστημα εμφανίζει το αντίστοιχο ερωτηματολόγιο
- Ο χρήστης κάνει την αξιολόγηση στο ερωτηματολόγιο επιλέγοντας «Μη ικανοποιητική» «Μέτρια» «Καλή» «Πολύ καλή» και επιλέγει το πλήκτρο συνέχεια.
- Το σύστημα εμφανίζει την οθόνη ‘Οριστική υποβολή’
- Ο χρήστης επιλέγει το πλήκτρο Υποβολή
- Το σύστημα στέλνει την αξιολόγηση στην βάση δεδομένων της γραμματείας και εμφανίζει την οθόνη ‘Ολοκλήρωση Αξιολόγησης’ με το μήνυμα «Η αξιολόγηση σας υποβλήθηκε με επιτυχία».

3.5 Περίπτωση χρήσης 2:Δημιουργία και αποθήκευση αξιολόγησης

3.5.1 Πρότυπο 2

Με την εισαγωγή του χρήστη στην υπηρεσία το σύστημα εμφανίζει την οθόνη ‘Οθόνη σύνδεσης’. Ο χρήστης καταχωρεί τα προσωπικά στοιχεία user name και password που του έχουν δοθεί από την γραμματεία του τμήματος του, για να συνδεθεί στην υπηρεσία. Αφού γίνει η σύνδεση, το σύστημα εμφανίζει την οθόνη ‘Επιλογή Τμήματος’ στην οποία ο χρήστης επιλέγει το τμήμα στο οποίο είναι εγγεγραμμένος. Μόλις ο χρήστης επιλέξει το πλήκτρο το σύστημα εμφανίζει την οθόνη ‘Επιλογή Μαθήματος’ όπου ο χρήστης επιλέγει το μάθημα που θέλει να αξιολογήσει. Στην συνέχεια ο χρήστης επιλέγει το πλήκτρο Επόμενο και το σύστημα εμφανίζει την οθόνη ‘Αξιολόγηση Μαθήματος’. Εκεί ο χρήστης επιλέγει την κατηγορία που θέλει να αξιολογήσει ‘Το μάθημα’ ‘Το Εργαστήριο’, ’Ο/Η Φοιτητής/τρια’, ’Ο/Η Διδάσκον/ουσα’.

Έτσι ανάλογα με την επιλογή του χρήστη εμφανίζετε και το αντίστοιχο ερωτηματολόγιο με την δυνατότητα αξιολόγησης στις ερωτήσεις σε «Μη ικανοποιητική» «Μέτρια» «Καλή» «Πολύ καλή». Έτσι όταν ο χρήστης ολοκληρώσει την αξιολόγηση και επιλέξει το πλήκτρο επόμενο το σύστημα του εμφανίζει την οθόνη ‘Οριστική υποβολή’ σε αυτό το σημείο ο χρήστης επιλέγει το πλήκτρο ‘Αποθήκευση’ και το σύστημα αποθηκεύει την αξιολόγηση. Το σύστημα εμφανίζει την οθόνη ‘Ολοκλήρωση Αξιολόγησης’ με το μήνυμα «Η αξιολόγηση σας αποθηκεύτηκε με επιτυχία»

3.5.2Βασική ροή

1. Ο χρήστης κάνει σύνδεση στο σύστημα με του προσωπικούς του κωδικούς στην ‘Οθόνη σύνδεσης’
2. Το σύστημα εμφανίζει την οθόνη ‘Επιλογή Τμήματος’
3. Ο χρήστης επιλέγει το τμήμα στο οποίο είναι εγγεγραμμένος και επιλέγει το πλήκτρο επόμενο
4. Το σύστημα εμφανίζει την οθόνη ‘Επιλογή Μαθήματος’
5. Ο χρήστης επιλέγει το μάθημα που θέλει να αξιολογήσει και επιλέγει το πλήκτρο επόμενο
6. Το σύστημα εμφανίζει την οθόνη ‘Αξιολόγηση Μαθήματος’
7. Ο χρήστης επιλέγει ποια κατηγορία θέλει να αξιολογήσει ανάμεσα στις κατηγορίες ‘Το μάθημα’ ‘Το Εργαστήριο’, ’Ο/Η Φοιτητής/τρια’, ’Ο/Η Διδάσκον/ουσα’.

8. Το σύστημα εμφανίζει το αντίστοιχο ερωτηματολόγιο
9. Ο χρήστης κάνει την αξιολόγηση στο ερωτηματολόγιο επιλέγοντας «Μη ικανοποιητική» «Μέτρια» «Καλή» «Πολύ καλή» και επιλέγει το πλήκτρο συνέχεια.
10. Το σύστημα εμφανίζει την οθόνη ‘Οριστική υποβολή’
11. Ο χρήστης επιλέγει το πλήκτρο Υποβολή
12. Το σύστημα αποθηκεύει την αξιολόγηση και εμφανίζει την οθόνη ‘Ολοκλήρωση Αξιολόγησης’ με το μήνυμα «Η αξιολόγηση σας αποθηκεύτηκε με επιτυχία».

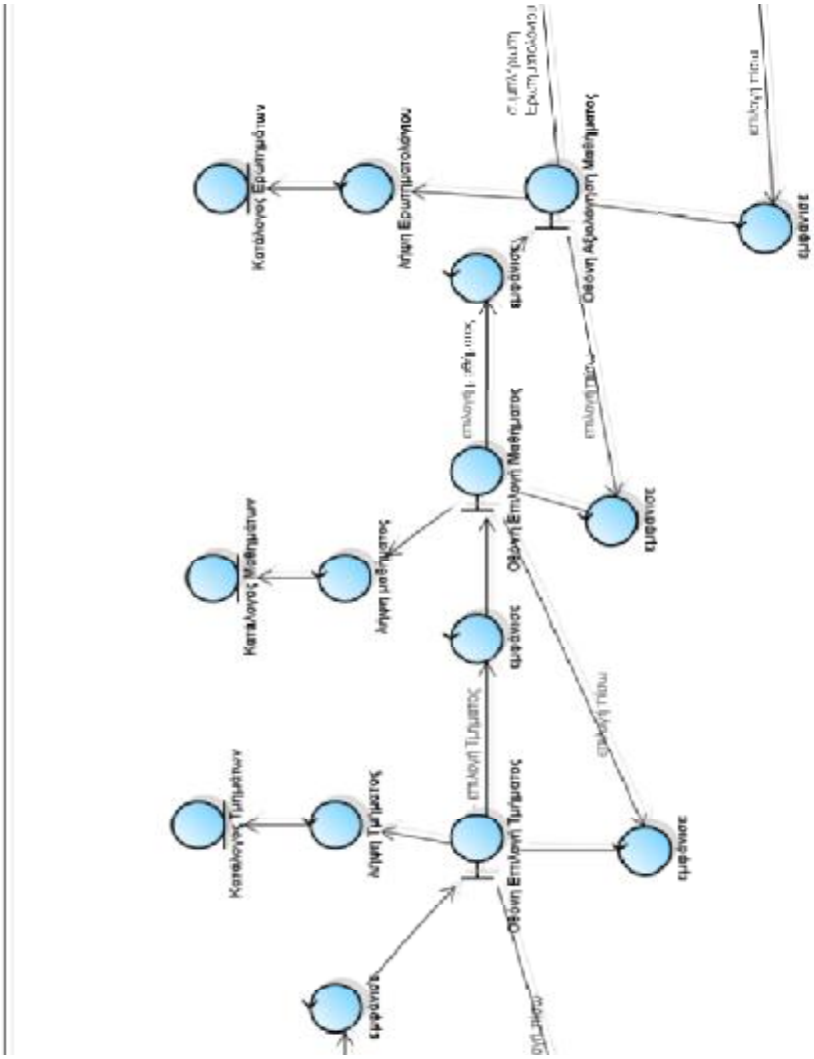
4.Ανάλυση

4.1 Διαγράμματα Ευρωστίας Συστήματος

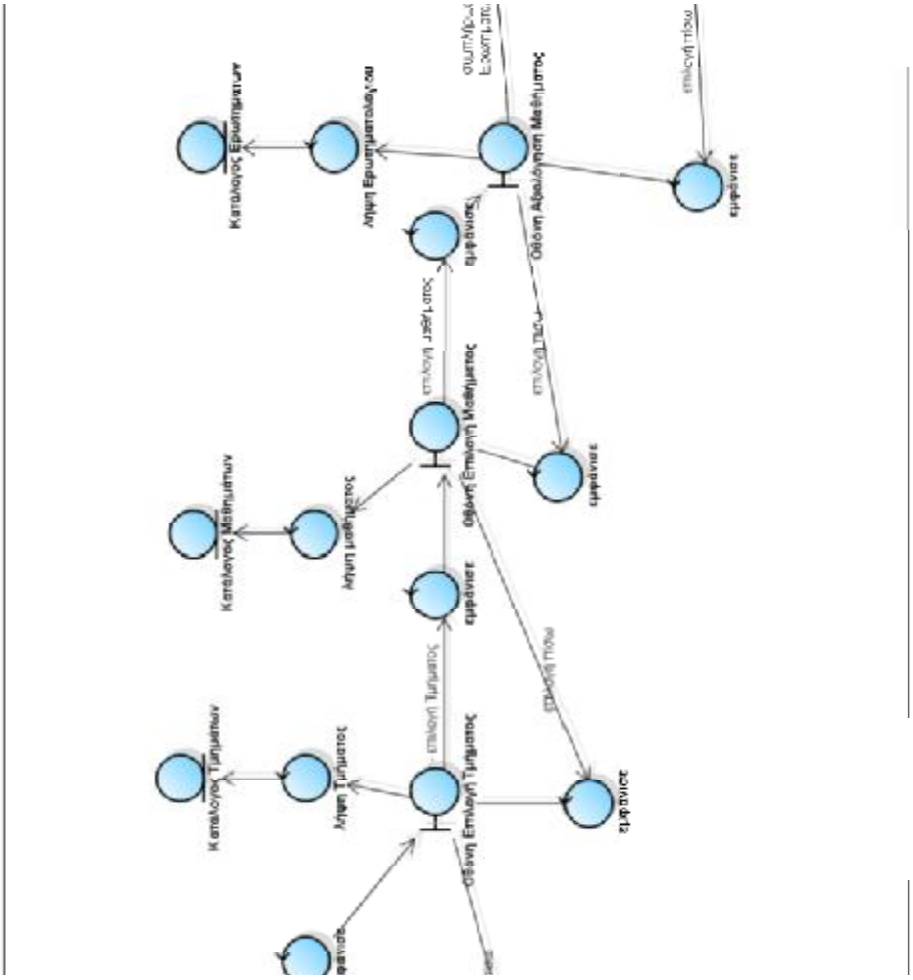
Για το υπό ανάπτυξη Σύστημα αξιολόγησης τα διαγράμματα ευρωστίας για κάθε περίπτωσης χρήσης είναι (για την πρώτη περίπτωση χρήσης η εξαγωγή του διαγράμματος ευρωστίας εξετάζεται αναλυτικά ενώ για τις υπόλοιπες περιπτώσεις η διαδικασία που ακολουθείται είναι ανάλογη):

4.1.1 Διάγραμμα Ευρωστίας 1: Ολοκλήρωσης αξιολόγησης

Η φράση ‘οι φοιτητές εισέρχονται προς συγκεκριμένο χρονικό διάστημα στην εφαρμογή’ απεικονίζεται ως μια ακμή μεταξύ του χρήστη και της συνοριακής κλάσης που αντιστοιχεί στην Κύρια Οθόνη. Η φράση " το σύστημα εμφανίζει την οθόνη ‘Οθόνη σύνδεσης’" υποδηλώνει την ύπαρξη στο διάγραμμα ευρωστίας μιας συνοριακής κλάσης για την " Οθόνη σύνδεσης". Τηρώντας τον κανόνα που δεν επιτρέπει την απευθείας συσχέτιση μεταξύ συνοριακών κλάσεων, εισάγεται μια κλάση ελέγχου που αντιστοιχεί στην ενέργεια της εμφάνισης της οθόνης (από την Οθόνη σύνδεσης), ως απόκριση στην επιλογή του χρήστη. Η φράση ‘‘το σύστημα εμφανίζει την οθόνη ‘Επιλογή Τμήματος’ στην οποία ο χρήστης επιλέγει το τμήμα στο οποίο είναι εγγεγραμμένος.’’ επιβάλλει την εισαγωγή μια κλάσης οντότητας που αντιστοιχεί στον Κατάλογο Τμημάτων και μιας κλάσης ελέγχου που αντιστοιχεί στην ενέργεια της λήψης των ονομάτων των Τμημάτων από τον κατάλογο.



Εικόνα 30." Διάγραμμα Ευρωστίας για την Περίπτωση Χρήσης Ολοκλήρωσης αξιολόγησης"



Εικόνα 31."Διάγραμμα Ευρωστίας για την Περίπτωση Χρήσης αποθήκευση αξιολόγησης"

5 Σχεδίαση

5.1 Διαγράμματα Ακολουθίας (Sequence diagrams)

5.1.1 Κατανομή λειτουργικότητας

Με την ολοκλήρωση των σταδίων της ανάλυσης οι προδιαγραφές του συστήματος υπό μορφή περιπτώσεων χρήσης έχουν παγιωθεί και μπορούν να θεωρηθούν πλήρεις, ορθές, λεπτομερείς και σαφείς. Επιπλέον έχουν εντοπιστεί οι περισσότερες κλάσεις, δηλαδή ολοκληρώθηκε το μεγαλύτερο τμήμα της στατικής δομής του λογισμικού.

Ωστόσο, λίγες ενέργειες στη φάση της ανάλυσης είχαν ως στόχο την κατανομή της συμπεριφοράς στις κλάσεις του συστήματος. Ο ακριβής καθορισμός του τρόπου με τον οποίο οι κλάσεις (τα αντικείμενά τους) αλληλοεπιδρούν μεταξύ τους, δηλαδή ο προσδιορισμός μέρους της δυναμικής συμπεριφοράς του συστήματος, αποτελούν αντικείμενο της σχεδίασης. Ως επακόλουθο αυτής της κατανομής λειτουργικότητας προκύπτει το αναθεωρημένο διάγραμμα κλάσεων όπου εκτός από τις κλάσεις και τις ιδιότητές τους, απεικονίζονται οι μέθοδοί τους και τυχόν νέες σχέσεις μεταξύ τους. Το τελικό αυτό διάγραμμα κλάσεων αποτελεί την είσοδο για την έναρξη της κωδικοποίησης του συστήματος (αν και είναι αναμενόμενο το διάγραμμα κλάσεων να τροποποιηθεί ως ένα βαθμό κατά την υλοποίηση).

Η κατανομή της λειτουργικότητας στις κλάσεις επιτυγχάνεται με την κατάστρωση διαγραμμάτων ακολουθίας. Ένα διάγραμμα ακολουθίας απεικονίζει τα μηνύματα που ανταλλάσσουν τα αντικείμενα του συστήματος μεταξύ τους για την ικανοποίηση της λειτουργικότητας ενός σεναρίου χρήσης (που συνήθως αντιστοιχεί σε μια περίπτωση χρήσης). Στα πλαίσια του αντικειμενοστραφούς μοντέλου ένα μήνυμα προς ένα αντικείμενο Α αντιστοιχεί σε αίτημα για την εκτέλεση μιας λειτουργίας από πλευράς του Α. Η οργάνωση των διαγραμμάτων ακολουθίας γίνεται σε δύο διαστάσεις. Σε οριζόντια διάταξη παρατίθενται τα αντικείμενα των κλάσεων που αλληλοεπιδρούν στο υπό εξέταση σενάριο. Η κάθετη διάσταση αντιστοιχεί στην κλίμακα του χρόνου, δηλαδή η εμφάνιση ενός μηνύματος σε χαμηλότερη θέση από κάποιο άλλο, υποδηλώνει και την αποστολή του σε μεταγενέστερο χρόνο.

5.2 Διάγραμμα Ακολουθίας Υποβολή Αξιολόγησης

Το διάγραμμα ακολουθίας καταστρώνεται ως εξής: Στον οριζόντιο άξονα των αντικειμένων τοποθετούνται, εξετάζοντας το αντίστοιχο διάγραμμα ευρωστίας, οι χειριστές

της περίπτωσης χρήσης (στην προκειμένη περίπτωση ο φοιτητής), τα αντικείμενα των συνοριακών κλάσεων οι οθόνες (τα αντικείμενα των συνοριακών κλάσεων, ο οθόνη "σύνδεσης", "Επιλογή Τμήματος", "Επιλογή Μαθήματος", "Αξιολόγηση Μαθήματος", "Οριστική υποβολή" και οθόνη "Ολοκλήρωσης αξιολόγησης") και τα αντικείμενα των κλάσεων οντοτήτων (λίστα μαθημάτων, λίστα τμημάτων και το ερωτηματολόγιο αξιολόγησης των μαθημάτων). Οι κλάσεις ελέγχου δεν εμφανίζονται στη συνήθη περίπτωση ως αντικείμενα στο διάγραμμα ακολουθίας αλλά "μεταφράζονται" σε μηνύματα μεταξύ των υπολοίπων κλάσεων. Λαμβάνοντας υπόψη ότι σε ένα διάγραμμα ακολουθίας απεικονίζεται η αλληλεπίδραση μεταξύ χρήστη και συστήματος για ένα συγκεκριμένο σενάριο χρήσης, το πρώτο μήνυμα (με αρίθμηση 1) είναι η εισαγωγή των στοιχείων του χρήστη στην οθόνη "σύνδεσης".

Στη συνέχεια, και με βάση το διάγραμμα ευρωστίας οθόνη "σύνδεσης" αποστέλλει το μήνυμα εμφάνισης (αντικείμενο ελεγκτή "εμφάνισε" στην Οθόνη Εισαγωγή στοιχείων-αντικείμενο συνοριακής κλάσης). Εκλαμβάνοντας τις ενέργειες που απεικονίζονται στο διάγραμμα ευρωστίας με δεξιόστροφη σειρά προτεραιότητας συμπεραίνεται ότι το πρώτο μήνυμα που αποστέλλει η οθόνη "Επιλογή Τμήματος" είναι το αίτημα λήψης Τμημάτων προς τον κατάλογο τμημάτων. Το όνομα του κάθε τμήματος μπορεί να "αντληθεί" αποστέλλοντας μήνυμα σε κάθε αντικείμενο της κλάσης Τμήματα που περιλαμβάνει ο κατάλογος τμημάτων.

Για το λόγο αυτό, παρόλο που κάτι τέτοιο δεν καταγράφεται στο διάγραμμα ευρωστίας, επιλέγουμε να απεικονίσουμε στο διάγραμμα ακολουθίας την επαναληπτική αποστολή μηνυμάτων λήψης ονόματος από το αντικείμενο της κλάσης Κατάλογος Τμημάτων προς αντικείμενα της κλάσης Τμήματα (το οποίο για το λόγο αυτό προστίθεται στον οριζόντιο άξονα του διαγράμματος ακολουθίας). Όπως γίνεται φανερό, είναι επιτρεπτό και αναμενόμενο στις περισσότερες περιπτώσεις, το διάγραμμα ακολουθίας να εμπλουτιστεί με περαιτέρω στοιχεία δυναμικής συμπεριφοράς του συστήματος, από αυτά που καταγράφηκαν σε προηγούμενες φάσεις. Με το πέρας της λήψης των ονομάτων από τον κατάλογο Τμημάτων και την επιστροφή της λίστας των ονομάτων στην οθόνη Επιλογή Τμήματος, η οθόνη εμφανίζει τα ονόματα των Τμημάτων που υπάρχουν στο σύστημα στον χρήστη.

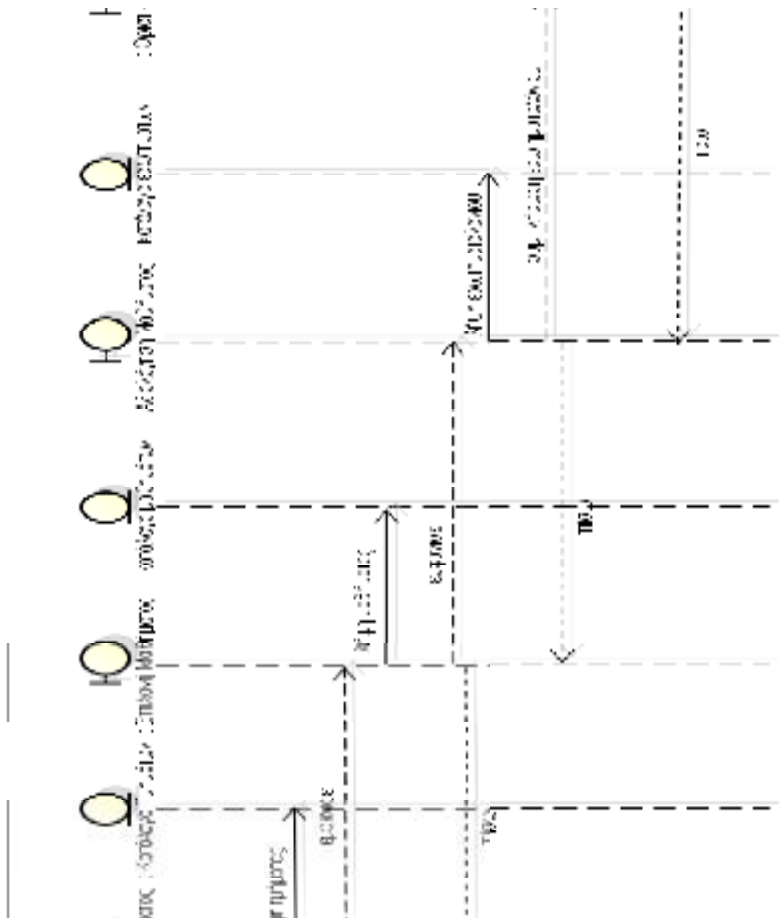
Η επόμενη ενέργεια που υποδηλώνεται στο διάγραμμα ευρωστίας είναι η επιλογή ενός μαθήματος . Το όνομα του κάθε μαθήματος μπορεί να "αντληθεί" αποστέλλοντας μήνυμα σε κάθε αντικείμενο της κλάσης μαθήματα που περιλαμβάνει ο κατάλογος

μαθημάτων. Με το πέρας της λήψης των ονομάτων από τον κατάλογο μαθημάτων και την επιστροφή της λίστας των ονομάτων στην οθόνη Επιλογή μαθήματος , η οθόνη εμφανίζει τα ονόματα των μαθημάτων που υπάρχουν στο σύστημα στον χρήστη.

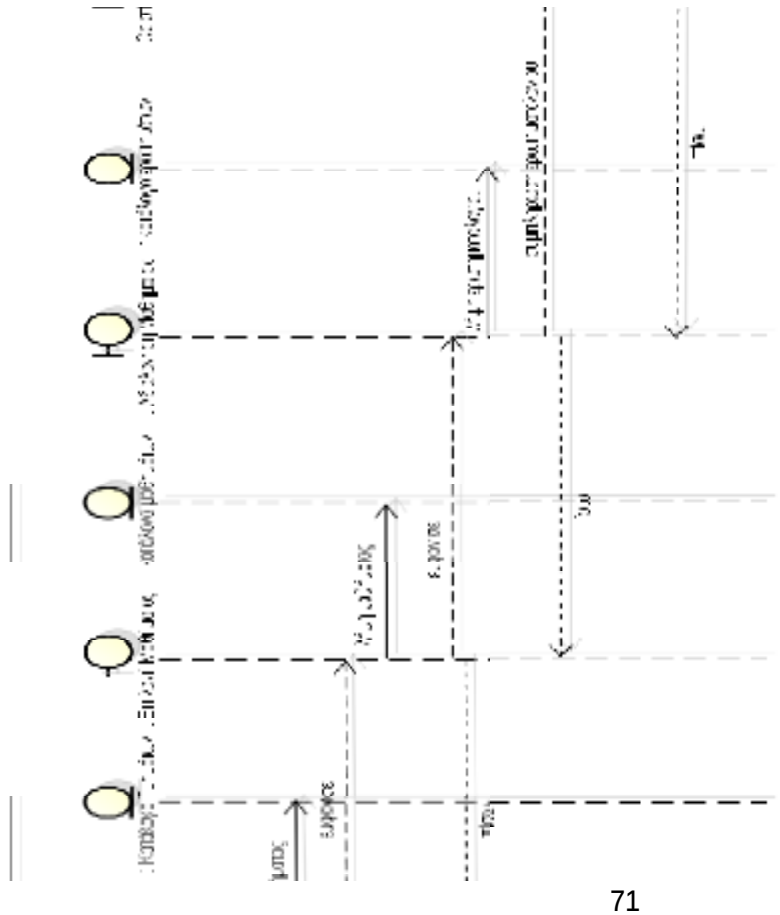
Η επόμενη ενέργεια που υποδηλώνεται στο διάγραμμα ευρωστίας είναι η εμφάνιση-συμπλήρωση του ερωτηματολογίου από την οθόνη Αξιολόγηση Μαθήματος. Με το πέρας της λήψης του ερωτηματολογίου από τον κατάλογο ερωτημάτων και την επιστροφή της λίστας των κατηγοριών στην οθόνη συμπλήρωση ερωτηματολόγιου , η οθόνη εμφανίζει τα ερωτήματα που υπάρχουν στο σύστημα στον χρήστη. Μετά την συμπληρώσει του ερωτηματολογίου το σύστημα εμφανίζει την επόμενη ενέργεια «Υποβολή» στην οθόνη Οριστική Υποβολή.

Η επόμενη ενέργεια που υποδηλώνεται στο διάγραμμα ευρωστίας είναι η δημιουργία μιας υποβολής , γεγονός που απεικονίζεται στο διάγραμμα ακολουθίας με ένα μήνυμα δημιουργίας ενός αντικειμένου προς την κλάση Αξιολόγησης (η τοποθέτηση της κλάσης οντότητας Αξιολόγησης στο επίπεδο του μηνύματος υποδηλώνει ότι το αντικείμενο δεν υφίστατο πριν την αποστολή του μηνύματος). Ωστόσο, η επιλογή καταχώρησης της υποβολής (που ισοδυναμεί με το πάτημα ενός πλήκτρου άρα και την εμφάνιση ενός συμβάντος) απεικονίζεται ως ακμή από τον χρήστη προς την οθόνη αξιολόγηση μαθήματος. Με βάση το διάγραμμα ευρωστίας προκύπτει ότι αμέσως επόμενη ενέργεια είναι η καταχώρηση της υποβολής στο αντικείμενο της κλάσης αξιολόγηση.

Τέλος, η επιλογή του πλήκτρου οριστική υποβολή από το χρήστη, με βάση το διάγραμμα ευρωστίας προκαλεί την εισαγωγή της αξιολόγησης που δημιουργήθηκε στον κατάλογο ερωτημάτων στο σύστημα . Το γεγονός αυτό απεικονίζεται στο διάγραμμα ακολουθίας ως ένα μήνυμα με παράμετρο το αντικείμενο της κλάσης αξιολόγησης από την οθόνη Οριστική Υποβολή. Η αμέσως επόμενη ενέργεια είναι η εμφάνιση της οθόνης ολοκλήρωση αξιολόγησης που απεικονίζεται ως μήνυμα εμφάνισης προς την Οθόνη οριστική υποβολή. Το ολοκληρωμένο διάγραμμα ακολουθίας για την 1η περίπτωση χρήσης παρουσιάζεται στην Εικόνα 32:



Εικόνα 32: Διάγραμμα ακολουθίας για την 1η περίπτωση



Εικόνα 33: Διάγραμμα ακολουθίας για την 2η περίπτωση

5.3 Διάγραμμα Κλάσεων

5.3.1 Στατικό μοντέλο συστήματος

Ο κύριος σκοπός δημιουργίας των διαγραμμάτων ακολουθίας είναι η κατανομή της λειτουργικότητας (μεθόδων) στις κλάσεις του συστήματος και κατά δεύτερο λόγο ο εντοπισμός τυχόν κλάσεων, ιδιοτήτων και μεθόδων που δεν αναδείχθηκαν στο στάδιο της ανάλυσης. Το στατικό μοντέλο του υπό ανάπτυξη συστήματος, δηλαδή το διάγραμμα κλάσεων που περιγράφει την αρχιτεκτονική του είναι επομένως αναμενόμενο να αναθεωρείται μετά την ολοκλήρωση των διαγραμμάτων ακολουθίας. Στο αναθεωρημένο διάγραμμα κλάσεων περιλαμβάνονται πλέον οι μέθοδοι κάθε κλάσης με την πλήρη υπογραφή τους, δηλαδή μαζί με τυχόν παραμέτρους που απαιτούνται και τον επιστρεφόμενο τύπο τους.
Η λήψη ενός μηνύματος από ένα αντικείμενο μιας κλάσης σε ένα διάγραμμα ακολουθίας υποδηλώνει την ύπαρξη μιας μεθόδου στην κλάση που είναι ο αποδέκτης του μηνύματος. Η μέθοδος συνιστά τη λειτουργία που θα εκτελείται στην κλάση-αποδέκτη με τη λήψη του αντίστοιχου μηνύματος.

5.3.2 Ποιότητα Σχεδίασης

Η αρχιτεκτονική του συστήματος που προέκυψε από την εφαρμογή της μεθοδολογίας ICONIX (δηλαδή η κατανομή της λειτουργικότητας σε κλάσεις και ο καθορισμός των σχέσεων μεταξύ τους) προφανώς ικανοποιεί τις απαιτήσεις του πελάτη, αφού η διαδικασία οδηγείται αυστηρά από τις περιπτώσεις χρήσης. Κάτι αντίθετο θα ήταν ούτως ή άλλως παράλογο, αφού η πρωταρχική απαίτηση από το λογισμικό είναι να παρέχει ποιότητα από την οπτική γωνία του χρήστη. Ωστόσο, το λογισμικό αξιολογείται από πλευράς ποιότητας και από την πλευρά του κατασκευαστή και στα πλαίσια αυτά επιχειρείται η αξιολόγηση της ποιότητας του σχεδίου.

Ωστόσο, η ίδια η ποιότητα εντός αντικειμενοστραφούς σχεδίου είναι δύσκολο να καθοριστεί. Εν γένει, είναι δύσκολο να αποφανθεί κανείς αν μία σχεδίαση είναι "καλή". Αντίθετα, οι επιπτώσεις από μια σχεδίαση "κακής" ποιότητας γίνονται εμφανείς πολύ ξεκάθαρα, ιδιαίτερα σε έργα λογισμικού που πρόκειται να εξελιχθούν, δηλαδή να παράγουν πολλαπλές γενιές του ιδίου προϊόντος λόγω μεταβαλλόμενων και νέων απαιτήσεων. Η έλλειψη ποιότητας στην αρχιτεκτονική σχεδίαση οδηγεί με βεβαιότητα στα ακόλουθα συμπτώματα:

• σε δυσκολία κατανόησης του συστήματος από τα μέλη της ομάδας ανάπτυξης που θα κληθούν μελλοντικά να προσθέσουν ή να τροποποιήσουν τη λειτουργικότητα

• σε δυσκολία συντήρησης που μεταφράζεται σε υπερβολική προσπάθεια από πλευράς χρόνου και κόστους για την υλοποίηση μελλοντικών αλλαγών στο λογισμικό

• σε δυσκολία ελέγχου του συστήματος που συνίσταται στην επαλήθευση (verification) της ορθότητας των λειτουργιών και επικύρωση (validation) ότι το λογισμικό ικανοποιεί τις απαιτήσεις του πελάτη

• σε δυσκολία επαναχρησιμοποίησης τμημάτων του λογισμικού που θα ήταν ενδεχομένως αξιοποιήσιμα σε διαφορετικές εφαρμογές ή και περιβάλλοντα

Η αξιολόγηση της σχεδίασης αντικειμενοστραφών συστημάτων στην πράξη πραγματοποιείται από τους πλέον έμπειρους σχεδιαστές-προγραμματιστές των ομάδων ανάπτυξης, λίγο πριν την έναρξη υλοποίησης του κώδικα. Οι σχεδιαστές αυτοί ελέγχουν την αρχιτεκτονική του συστήματος βάσει εμπειρικών κανόνων που οι ίδιοι έχουν σχηματίσει και προτείνουν τυχόν βελτιώσεις ή τροποποιήσεις. Προφανώς μια τέτοια διαδικασία αξιολόγησης είναι υποκειμενική και παράγει αμφίβολης αξιοπιστίας αποτελέσματα. Επιπλέον, η αξιολόγηση κατ' αυτόν τον τρόπο δεν επιδέχεται αυτοματοποίηση. Για το λόγο αυτό, επιχειρήθηκε η αντικατάσταση της προσφυγής στην γνώμη των ειδικών από αντικειμενικότερες και ει δυνατόν αυτοματοποιήσιμες τεχνικές αξιολόγησης.

Στα πλαίσια των τεχνικών αυτών, η αξιολόγηση αντικειμενοστραφών σχεδίων μπορεί να πραγματοποιηθεί με τρεις τρόπους: α) συλλέγοντας και αξιολογώντας τις τιμές επιλεγμένων μετρικών λογισμικού, β) ελέγχοντας τη συμμόρφωση του σχεδίου με καθιερωμένες αρχές σχεδίασης και γ) ελέγχοντας την τυχόν παραβίαση ευρετικών κανόνων.

5.3.3 Αξιολόγηση Σχεδίασης με χρήση Μετρικών Λογισμικού

Ο σκοπός των μετρήσεων είναι η ανάθεση αριθμητικών τιμών ή συμβόλων σε χαρακτηριστικά οντοτήτων που υπάρχουν στην πραγματικότητα. Για την πραγματοποίηση των μετρήσεων απαιτείται η ύπαρξη ενός μοντέλου το οποίο να καθορίζει τους κανόνες με βάση τους οποίους πρέπει να εκτελούνται οι μετρήσεις και να ερμηνεύονται τα αποτελέσματά τους. Σε ένα έργο λογισμικού, όπως και σε οποιοδήποτε τεχνικό έργο, οι μετρήσεις επιτρέπουν την καλύτερη διαχείριση, παρακολούθηση και αξιολόγηση των παραγόμενων προϊόντων και παραδοτέων. Σύμφωνα με τη ρήση του DeMarco, "δεν μπορείς να ελέγξεις αυτό που δεν μπορείς να μετρήσεις" ("you cannot control what you cannot measure").

Ο όρος "μετρική" στην Τεχνολογία Λογισμικού αναφέρεται στην ποσοτική εκτίμηση του βαθμού κατά τον οποίο ένα προϊόν ή μία διαδικασία κατέχει ένα συγκεκριμένο χαρακτηριστικό. Στη βιβλιογραφία έχουν προταθεί εκατοντάδες μετρικές και στην πράξη είναι δύσκολο να αξιολογήσει κανείς ποια είναι η καλύτερη επιλογή. Ωστόσο, μια ιδανική μετρική λογισμικού θα πρέπει να είναι:

• Απλή και υπολογίσιμη: Η εκμάθηση του τρόπου εξαγωγής της μετρικής θα πρέπει να είναι σχετικά απλή και ο υπολογισμός της όχι υπερβολικά δαπανηρός από πλευράς προσπάθειας και χρόνου.

• Εμπειρικά και διαισθητικά πειστική: Η χρησιμοποιούμενη μετρική θα πρέπει να ικανοποιεί την διαίσθηση του μηχανικού λογισμικού σχετικά με το υπό εξέταση χαρακτηριστικό (π.χ. η αριθμητική τιμή του μέτρου της συνεκτικότητας μιας μονάδας θα πρέπει να αυξάνει όσο αυξάνει το επίπεδο συνεκτικότητας)

• Συνεπής και αντικειμενική: Τα αποτελέσματα που προκύπτουν από την εφαρμογή της μετρικής στο ίδιο λογισμικό και για ίδια δεδομένα εισόδου, σε διαφορετικές χρονικές

στιγμές, θα πρέπει να είναι πάντοτε τα ίδια. Οποιοσδήποτε χρήστης της μετρικής θα πρέπει να καταλήγει στα ίδια αποτελέσματα.

• Συνεπής ως προς τη χρήση μονάδων και διαστάσεων: Η χρήση μαθηματικών μεθόδων σε μία μετρική δεν θα πρέπει να καταλήγει σε "περίεργα" αποτελέσματα από πλευράς μονάδων μέτρησης. Για παράδειγμα ο πολλαπλασιασμός των ατόμων μιας ομάδας σχεδίασης με τον αριθμό μεταβλητών του προγράμματος οδηγεί σε μονάδες μέτρησης που δεν είναι διαισθητικά αποδεκτές.

• Ανεξάρτητη από τη γλώσσα προγραμματισμού: Οι μετρικές θα πρέπει να βασίζονται στην ανάλυση του αλγορίθμου ή στη δομή ενός προγράμματος και όχι στις ιδιαιτερότητες σύνταξης και σημασιολογίας κάθε γλώσσας προγραμματισμού.

• Ένας ουσιαστικός μηχανισμός ανάδρασης: Μία μετρική θα πρέπει να παρέχει στο σχεδιαστή λογισμικού κατάλληλη πληροφορία ώστε να μπορεί να βελτιώσει την ποιότητα του παραγόμενου προϊόντος.

Στην Τεχνολογία Λογισμικού υπάρχουν ήδη από τις αρχές της δεκαετίας του 1970 δύο θεμελιώδεις έννοιες που χρησιμοποιούνται για την αξιολόγηση της αρχιτεκτονικής δομής ενός συστήματος λογισμικού (όχι μόνο αντικειμενοστραφούς λογισμικού). Η έννοια της σύζευξης και της συνεκτικότητας.

5.3.4 Σύζευξη

Η σύζευξη (coupling) αναφέρεται στο βαθμό αλληλεξάρτησης μεταξύ των μονάδων ενός συστήματος (κλάσεων για αντικειμενοστραφή συστήματα). Όσο μικρότερη είναι η αλληλεξάρτηση, τόσο ευκολότερη είναι η κατανόηση, η συντήρηση, ο έλεγχος και η επαναχρησιμοποίηση κάθε μονάδας. Κατά συνέπεια, βασικός στόχος κατά τη διάρκεια της σχεδίασης είναι η ελαχιστοποίηση της σύζευξης. Ο στόχος αυτός δεν είναι εύκολο να επιτευχθεί ακολουθώντας μια συγκεκριμένη διαδικασία. Ωστόσο, είναι σχετικά εύκολη η μέτρηση της σύζευξης ενός συστήματος λογισμικού (είτε αυτό υπάρχει ως διάγραμμα κλάσεων είτε ως κώδικας) και η παρακολούθηση της εξέλιξής της. Η εμφάνιση υψηλών τιμών σύζευξης είτε για ολόκληρο το σύστημα είτε για μεμονωμένες κλάσεις λειτουργεί ως ένα σήμα προειδοποίησης που επιβάλλει την αναδιοργάνωση του σχεδίου.

5.3.5 Συνεκτικότητα

Η συνεκτικότητα (cohesion) αναφέρεται στο βαθμό εσωτερικής συνοχής των τμημάτων μιας μονάδας λογισμικού. Στα πλαίσια των αντικειμενοστραφών συστημάτων ποσοτικοποιεί το βαθμό λειτουργικής συνάφειας των μεθόδων μιας κλάσης. Όσο πιο συνεκτική είναι μια κλάση, τόσο πιο πολύ οι μέθοδοί της σχετίζονται μεταξύ τους και συνεργάζονται για την επίτευξη ενός και μόνο κοινού σκοπού. Για λόγους ευκολότερης κατανόησης, συντήρησης και ελέγχου μιας κλάσης, είναι γενικά επιθυμητό μια κλάση να μην αναλαμβάνει διαφορετικές αρμοδιότητες. Υπό αυτή την έννοια μια κλάση πρέπει να είναι όσο το δυνατόν πιο συνεκτική και αν είναι εφικτό, όλες οι μέθοδοί της να είναι συνεκτικές μεταξύ τους. Δύο μέθοδοι θεωρούνται συνεκτικές εάν προσπελαύνουν έστω και μία κοινή ιδιότητα της κλάσης, θεωρώντας ότι η χρήση κοινών ιδιοτήτων υποδηλώνει και κάποιου είδους λειτουργική συνάφεια.

6.Συμπεράσματα

Η εφαρμογή της αντικειμενοστρεφούς μεθοδολογίας ανάλυσης και σχεδίασης κατέδειξε ότι με απλά και συστηματικά βήματα είναι δυνατόν να μεταβούμε από τις αρχικές απαιτήσεις χρήστη σε ένα λεπτομερές σχέδιο του συστήματος λογισμικού και εν συνεχεία να υλοποιήσουμε τον κώδικα. Βεβαίως η διαδικασία αυτή απαιτεί εμπειρία και γνώσεις από την πλευρά του αναλυτή-σχεδιαστή-προγραμματιστή και για το λόγο αυτό δεν είναι αυτοματοποιήσιμη. Ωστόσο, η τήρηση των βημάτων της μεθοδολογίας ICONIX, η αξιοποίηση της Ενοποιημένης Γλώσσας Μοντελοποίησης για τη δημιουργία των απαραίτητων μοντέλων και η χρήση σύγχρονων εργαλείων CASE διευκολύνουν σημαντικά τη διαδικασία ανάπτυξης λογισμικού. Εκτός της παραγωγής λειτουργικού κώδικα που ικανοποιεί πλήρως τις απαιτήσεις του πελάτη, αξίζει να σημειωθεί ότι η προσήλωση σε μια μεθοδολογία ανάπτυξης οδηγεί και στην κατασκευή ποιοτικού λογισμικού, που μπορεί να κατανοηθεί, να ελεγχθεί, να συντηρηθεί και να επαναχρησιμοποιηθεί με μικρό κόστος και προσπάθεια.

7. Πίνακας εικόνων

8.Βιβλιογραφία

8.1 Ξενόγλωσση

1. G. Booch, J. Rumbaugh and I. Jacobson, *the Unified Modeling Language User Guide*, Addison Wesley, 1999.
2. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
3. C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edn., Prentice Hall, 2005.
4. D. Rosenberg and M. Stephens, *Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007.
5. J. Rumbaugh, I. Jacobson and G. Booch, *Unified Modeling Language Reference Manual*, 2nd edn. Addison-Wesley, 2004.

8.2 Ελληνική

1. . Βεσκούκης, *Τεχνολογία Λογισμικού ΙΙ*, Ελληνικό Ανοικτό Πανεπιστήμιο, Πάτρα 2001.
2. . Γερογιάννης, Γ. Κακαρόντζας, Α. Καμέας, Γ. Σταμέλος, Π. Φιτσιλής, *Αντικειμενοστρεφής Ανάπτυξη Λογισμικού με τη UML*, Κλειδάριθμος, 2006.
3. Fowler M. and Scott K., *Εισαγωγή στη UML* (*UML Distilled: A Bried Guide to the Standard Object Modeling Language*), Κλειδάριθμος, 2001.