ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΑΣ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΓΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΣΤΗΝ ΔΙΟΙΚΗΣΗ ΚΑΙ ΣΤΗΝ ΟΙΚΟΝΟΜΙΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΛΟΠΟΙΗΣΗ ΔΙΑΔΙΚΤΥΑΚΗΣ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΤΗΝ ON-LINE ΚΑΤΑΘΕΣΗ ΔΗΛΩΣΕΩΝ ΕΡΓΑΣΤΗΡΙΩΝ

WEB APPLICATION SOFTWARE ENGINEERING FOR STUDENTS' ON-LINE LESSON REGISTRATION

Σπουδαστές: Ρήγας Λάμπρος

Παναγιωτόπουλος Ανδρέας

Εισηγητής: Τρυφωνόπουλος Αθηνόδωρος

ΑΜΑΛΙΑΔΑ ΟΚΤΩΒΡΙΟΣ 2010

ΠΕΡΙΕΧΟΜΕΝΑ

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή μας κ. Αθηνόδωρο Τρυφωνόπουλο που μας εμπιστεύθηκε και στάθηκε αρωγός και συνεπίκουρος στην προσπάθεια εκπόνησης της εργασίας. Οι γνώση και η πείρα του σε θέματα software engineering μας διαφώτισε και έδωσε λύσεις σε πολλές περιπτώσεις.

Επίσης ευχαριστούμε τις οικογένειές μας για την υπομονή και συμπαράσταση που επέδειξαν κατά την διάρκεια των σπουδών μας.

Ρήγας Λ.- Παναγιωτόπουλος Ανδ.

ΠΕΡΙΛΗΨΗ

Στόχος της πτυχιακής μας είναι η υλοποίηση μιας εφαρμογής για την on-line δήλωση των εργαστηριακών ομάδων των μαθημάτων του ΤΕΙ, μέσα από την διαδικασία σχεδιασμού και υλοποίησης λογισμικού. Έτσι μελετώντας τις βασικές αρχές του software engineering ,όπως τα βασικά πρότυπα μοντέλα διαδικασίας υλοποίησης λογισμικού, τις αρχές καταγραφής των απαιτήσεων και την θεωρία αρχιτεκτονικής σχεδίασης λογισμικού προχωρήσαμε στην μελέτη περίπτωσης που αφορά στην συγκεκριμένη εφαρμογή. Μέσα από την εφαρμογή ο χρήστης- φοιτητής θα μπορεί να επιλέγει τις εργαστηριακές ομάδες που προσφέρονται για κάθε μάθημα εξαμήνου, να δημιουργεί λίστα με τις επιλογές του και να την υποβάλλει. Παράλληλα ο χρήστης-διαχειριστής μετά από διαδικασία ταυτοποίησης θα μπορεί να διαχειρίζεται την βάση δεδομένων της εφαρμογής πχ τροποποιώντας τις δηλώσεις, προσθέτοντας ή αφαιρώντας εργαστηριακή ομάδα, φοιτητή κτλ. Επίσης, η τεκμηρίωση της υλοποίησης έγινε επίσημα, όπως συμβαίνει σε κάθε έργο που αναλαμβάνει μια εταιρία παραγωγής λογισμικού. Πιστεύουμε πως έτσι, από την μια δώσαμε λύση στο πρόβλημα δήλωσης εργαστηρίων στο ΤΕΙ που έως τώρα γίνεται με ενυπόγραφες καταστάσεις φοιτητών, αλλά και αποκομίσαμε πολύτιμες εμπειρίες και γνώσεις πάνω στον σχεδιασμό και την υλοποίηση εφαρμογών λογισμικού.

ABSTRACT

The objective of our final is the implementation of an application for on-line registration of laboratories of courses of our technological college, via process of software engineering. Thus studying the basic disciplines of software engineering, as the basic software process models, the theory of requirements specification and the theory of architectural designing of software we advanced in the case study which concerns the particular application. Through the application the user- student might select the laboratorial teams that are offered for each course of semester , create list with his choices and submit them. At the same time the user-administrator after process of identification might manage the data base of the application i.e. modifying the registrations, adding or removing laboratorial team, student etc. Also, the documentation of the software produced is done officially, as it happens in each work that undertakes a company of production of software. We believe that thus, from the one we gave solution in the problem of laboratories registration in our techonological college that until now takes place with signed lists of students, but we also acquired precious experiences and knowledge on designing and implementation of software application.

ΕΙΣΑΓΩΓΗ

Στις μέρες μας πλέον, όλες οι χώρες εξαρτώνται από σύνθετα υπολογιστικά συστήματα. Οι εθνικές υποδομές και εφαρμογές στηρίζονται στα υπολογιστικά συστήματα και τα περισσότερα ηλεκτρικά προϊόντα περιλαμβάνουν έναν υπολογιστή και ένα λογισμικό ελέγχου. Η βιομηχανική κατασκευή και τα δίκτυα είναι εντελώς αυτοματοποιημένες, όπως είναι και το οικονομικό σύστημα. Επομένως, η οικονομική παραγωγή και διατήρηση του λογισμικού είναι ουσιαστικές για τη λειτουργία των εθνικών και διεθνών οικονομιών.

Η τεχνολογία λογισμικού είναι μια αρχή σχεδιασμού και υλοποίησης της οποίας εστίαση είναι η οικονομική ανάπτυξη υψηλής ποιότητας συστημάτων λογισμικού. Το λογισμικό είναι αφηρημένο και άυλο. Δεν περιορίζεται από τα υλικά, ή κυβερνάται από τους φυσικούς νόμους ή από τις διαδικασίες παραγωγής. Με κάποιους τρόπους, αυτό απλοποιεί την σχεδίαση και υλοποίηση λογισμικού δεδομένου ότι δεν υπάρχει κανένας φυσικός περιορισμός στις δυνατότητες του λογισμικού. Εντούτοις, αυτή η έλλειψη φυσικών περιορισμών σημαίνει ότι το λογισμικό μπορεί εύκολα να γίνει εξαιρετικά σύνθετο και ως εκ τούτου πολύ δύσκολο να κατανοηθεί.

Η έννοια του σχεδιασμού και υλοποίησης λογισμικού (software engineering) προτάθηκε αρχικά το 1968 σε μια διάσκεψη που έγινε για να συζητηθεί η λεγόμενη τότε «κρίση λογισμικού»'. Αυτή η κρίση λογισμικού προέκυψε άμεσα από την εισαγωγή του νέου υλικού υπολογιστών βασισμένου σε ενσωματωμένα κυκλώματα. Η δύναμή τους κατέστησε τις έως τώρα ανεφάρμοστες εφαρμογές υπολογιστών μια εφικτή πρόταση. Το προκύπτον λογισμικό ήταν πολύ μεγαλύτερα και περισσότερο πολύπλοκα από τα προηγούμενα συστήματα λογισμικού.

Η προηγούμενη εμπειρία στην οικοδόμηση αυτών των συστημάτων έδειξε ότι η άτυπη ανάπτυξη λογισμικού δεν ήταν αρκετά καλή. Σημαντικά προγράμματα ήταν μερικές φορές έτη πίσω. Το λογισμικό κόστισε πολύ περισσότερο από ότι προβλέφθηκε, ήταν αναξιόπιστο, ήταν δύσκολο να διατηρηθεί και απέδωσε κακώς. Η ανάπτυξη λογισμικού ήταν στην κρίση. Τα κόστη υλικού έπεφταν διαρκώς, ενώ τα κόστη λογισμικού αυξάνονταν γρήγορα. Νέες τεχνικές και οι μέθοδοι απαιτήθηκαν για να ελέγξουν την έμφυτη πολυπλοκότητα στα μεγάλα συστήματα λογισμικού.

Αυτές οι τεχνικές έχουν γίνει μέρος της τεχνολογίας λογισμικού και πλέον χρησιμοποιούνται ευρέως. Εντούτοις, δεδομένου ότι η δυνατότητά μας να παράγουμε λογισμικό έχει αυξηθεί, τόσο επίσης έχει αυξηθεί και η πολυπλοκότητα των συστημάτων λογισμικού που χρειαζόμαστε. Οι νέες τεχνολογίες που πηγάζουν από την σύγκλιση των συστημάτων υπολογιστών και επικοινωνιών και των σύνθετων γραφικών διεπαφών με τον χρήστη τοποθετούν νέες απαιτήσεις στους μηχανικούς λογισμικού. Δεδομένου ότι πολλές επιχειρήσεις ακόμα δεν εφαρμόζουν τις τεχνικές τεχνολογίας λογισμικού αποτελεσματικά, πάρα πολλά έργα παράγουν ακόμα λογισμικό που είναι αναξιόπιστο, παραδίδεται αργά και κοστίζει πέρα από τον προϋπολογισμό.

Επισημαίνεται, ότι έχουμε σημειώσει τεράστια πρόοδο από το 1968 και ότι η ανάπτυξη του σχεδιασμού και υλοποίησης του λογισμικού έχει βελτιώσει εμφανώς το λογισμικό μας. Έχουμε μια πολύ καλύτερη κατανόηση των δραστηριοτήτων που περιλαμβάνονται στη ανάπτυξη λογισμικού. Έχουμε αναπτύξει τις πλέον αποτελεσματικές μεθόδους προδιαγραφής, σχεδιασμού και υλοποίησης λογισμικού. Οι νέες σημειώσεις και τα εργαλεία μειώνουν την προσπάθεια που απαιτείται για να παράγουμε μεγάλα και σύνθετα συστήματα.

Στην εργασία μας αυτή επιχειρούμε να παρουσιάσουμε στον αναγνώστη το θεωρητικό υπόβαθρο που απαιτείται να έχει ένας σχεδιαστής λογισμικού, τόσο κατά την μελέτη σχεδιασμού όσο και για την υλοποίηση του. Εν συνεχεία, μέσα από την μελέτη περίπτωσης σχεδίασης και υλοποίησης μιας διαδικτυακής εφαρμογής για την on-line δήλωση εργαστηρίων των φοιτητών , ο αναγνώστης εξοικειώνεται καλύτερα και έχει την δυνατότητα να παρατηρήσει πως εφαρμόζονται στην πράξη, οι βασικές αρχές σχεδίασης και υλοποίησης κατά την διάρκεια όλων των φάσεων παραγωγής του λογισμικού.

1. Σχεδιασμός και υλοποίηση λογισμικού-Γενικές έννοιες

1.1 Γενικά

Στο κεφάλαιο αυτό της πτυχιακής θα προσπαθήσουμε να προσεγγίσουμε μερικά θεμελιώδη ζητήματα της τεχνολογίας λογισμικού , ώστε να μπορέσει να εξοικειωθεί ο αναγνώστης με τις βασικές αρχές. Στον πίνακα 1, παραθέτουμε τις βασικές έννοιες της τεχνολογίας λογισμικού με την μορφή των συχνών ερωτημάτων (frequently asked questions), μια προσέγγιση που είναι φιλική προς τον αναγνώστη ειδικά αυτόν που δεν είναι εξοικειωμένος με τις έννοιες.

1.2 Λογισμικό και σχεδιασμός λογισμικού

Πολλοί άνθρωποι εξισώνουν τον όρο λογισμικό με τα προγράμματα υπολογιστών. Εντούτοις, θα προτιμούσαμε έναν ευρύτερο ορισμό σύμφωνα με τον οποίο το λογισμικό είναι όχι μόνο τα προγράμματα αλλά και όλο τα σχετικά στοιχεία τεκμηρίωσης και διαμόρφωσης που απαιτούνται για να κάνουν αυτά τα προγράμματα να λειτουργήσουν σωστά. Ένα σύστημα λογισμικού αποτελείται συνήθως από διάφορα χωριστά προγράμματα, αρχεία διαμόρφωσης, που χρησιμοποιούνται στην οργάνωση αυτών των προγραμμάτων, τεκμηρίωση συστημάτων, η οποία περιγράφει τη δομή του συστήματος, και την τεκμηρίωση χρηστών, η οποία εξηγεί πώς να χρησιμοποιηθεί το σύστημα και οι ιστόχωροι για τους χρήστες ώστε να κατέβουν πρόσφατες πληροφορίες για τα προϊόντα.

Οι μηχανικοί λογισμικού ενδιαφέρονται για την ανάπτυξη των προϊόντων λογισμικού, δηλ., λογισμικά που μπορούν να πωληθούν σε έναν πελάτη. Υπάρχουν δύο θεμελιώδεις τύποι προϊόντων λογισμικού:

1. Τα γενικά προϊόντα Αυτά είναι αυτόνομα συστήματα που παράγονται από έναν οργανισμό ανάπτυξης και πωλούνται στην ελεύθερη αγορά σε οποιοδήποτε πελάτη που είναι σε θέση να τους αγοράσει. Παραδείγματα αυτού του τύπου προϊόντων αποτελούν το λογισμικά για τα PC όπως οι βάσεις δεδομένων, επεξεργαστές κειμένου, πακέτα σχεδίασης και τα εργαλεία διαχείρισης έργων.

2. Τα προσαρμοσμένα προϊόντα Αυτά είναι συστήματα που ανατίθενται από έναν ιδιαίτερο πελάτη. Ένας ανάδοχος λογισμικού αναπτύσσει το λογισμικό ειδικά για εκείνο τον πελάτη. Τα παραδείγματα αυτού του τύπου λογισμικού περιλαμβάνουν τα συστήματα ελέγχου για τις ηλεκτρονικές συσκευές, συστήματα που γράφονται για να υποστηρίξουν μια ιδιαίτερη επιχειρησιακή διαδικασία και τα συστήματα ελέγχου εναέριας κυκλοφορίας.

Μια σημαντική διαφορά μεταξύ αυτών των τύπων λογισμικών είναι ότι, στα γενικά προϊόντα, ο οργανισμός που αναπτύσσει το λογισμικό ελέγχει τις προδιαγραφές λογισμικού. Για τα προϊόντα προσαρμογής (custom) , οι προδιαγραφές αναπτύσσονται συνήθως και ελέγχονται από την επιχείρηση που αγοράζει το λογισμικό. Οι προγραμματιστές λογισμικού πρέπει να εργαστούν πάνω σε αυτές τις προδιαγραφές.

Η κατασκευή (σχεδιασμός και υλοποίηση) λογισμικού είναι μια αρχή σχεδιασμού και υλοποίησης που σχετίζεται με όλες τις απόψεις παραγωγής λογισμικού από τα πρώτα στάδια καταγραφής των απαιτήσεων του συστήματος μέχρι και την συντήρησή του μετά την διάθεσή του προς χρήση. Στον παραπάνω ορισμό υπάρχουν δυο φράσεις κλειδιά:

1. Αρχή σχεδιασμού και υλοποίησης Οι μηχανικοί κάνουν τα πράγματα να δουλεύουν. Εφαρμόζουν τις θεωρίες, μεθοδολογίες και εργαλεία όπου αυτά είναι απαραίτητα, αλλά τα χρησιμοποιούν πάντα επιλεκτικά και προσπαθούν πάντα να ανακαλύψουν τις λύσεις στα προβλήματα ακόμα και όταν δεν υπάρχουν εφαρμόσιμες θεωρίες και μεθοδολογίες. Οι μηχανικοί επίσης αναγνωρίζουν ότι πρέπει να λειτουργήσουν με περιορισμούς επιπέδου και φύσης οργανισμού και οικονομίας. Έτσι ψάχνουν τις λύσεις μέσα σε αυτούς τους περιορισμούς.

2. Όλες οι απόψεις παραγωγής λογισμικού Οι μηχανικοί παραγωγής λογισμικού ενδιαφέρονται όχι μόνο για τις τεχνικές διαδικασίες της ανάπτυξης λογισμικού αλλά και για τις δραστηριότητες όπως η διαχείριση του προγράμματος λογισμικού και για την ανάπτυξη των εργαλείων, μεθόδων και των θεωριών για να υποστηρίξουν την παραγωγή λογισμικού.

Ερώτηση	Απάντηση
Τι είναι λογισμικό;	Τα προγράμματα υπολογιστών και η συσχετιζόμενη τεκμηρίωση. Τα προϊόντα λογισμικού μπορεί να αναπτυχθούν για συγκεκριμένο πελάτη η και για μια γενικευμένη αγορά
Τι είναι σχεδιασμός και υλοποίηση λογισμικού;	Ο σχεδιασμός και υλοποίηση λογισμικού είναι μια αρχή κατασκευής που σχετίζεται με όλες τις απόψεις της παραγωγής λογισμικού
Ποια είναι η διαφορά μεταξύ κατασκευής λογισμικού και επιστήμης των υπολογιστών;	Η επιστήμη των υπολογιστών σχετίζεται με θεωρία και θεμελιώδεις αρχές. Η κατασκευή λογισμικού σχετίζεται με τις πρακτικές ανάπτυξης και απόδοσης χρήσιμου λογισμικού
Ποια είναι η διαφορά μεταξύ κατασκευής λογισμικού και κατασκευής συστήματος;	Η κατασκευή συστήματος σχετίζεται με όλες τις απόψεις της ανάπτυξης συστημάτων που βασίζονται σε υπολογιστές, συμπεριλαμβανομένου των υλικών, λογισμικού και κατασκευής διαδικασιών. Η κατασκευή λογισμικού είναι μέρος αυτής.
Τι είναι διαδικασία λογισμικού;	Ένα σύνολο εργασιών που σκοπό έχουν την ανάπτυξη ενός λογισμικού
Τι είναι μοντέλο διαδικασίας λογισμικού;	Μια απλουστευμένη αναπαράσταση μιας διαδικασίας λογισμικού, που παρουσιάζεται από συγκεκριμένη προοπτική

Πίνακας 1 : **Βασικές έννοιες** [Sommerville, 2008, Software Engineering 8th edition]

Γενικά, οι μηχανικοί λογισμικού υιοθετούν μια συστηματική και οργανωμένη προσέγγιση στην εργασία τους, όπως αυτό είναι συχνά ο πλέον αποτελεσματικός τρόπος να παραχθεί υψηλής ποιότητας λογισμικό. Εντούτοις, ο σχεδιασμός και η υλοποίηση είναι πάνω από όλα η επιλογή της πιο κατάλληλης μεθόδου για ένα σύνολο καταστάσεων και μια δημιουργικότερη, λιγότερο επίσημη προσέγγιση στην ανάπτυξη μπορεί να είναι αποτελεσματική κάτω από κάποιες περιστάσεις. Η λιγότερο επίσημη ανάπτυξη είναι ιδιαίτερα κατάλληλη για την ανάπτυξη των βασισμένων στο WEB συστημάτων, η οποία απαιτεί ένα μίγμα δεξιοτήτων πάνω στο λογισμικό και στην γραφική σχεδίαση.

1.3 Διαδικασία λογισμικού

Η διαδικασία λογισμικού είναι το σύνολο δραστηριοτήτων και σχετικών αποτελεσμάτων που παράγουν ένα προϊόν λογισμικού. Υπάρχουν τέσσερις θεμελιώδεις δραστηριότητες διαδικασίας , που είναι κοινές για όλες τις διαδικασίες λογισμικού. Αυτές είναι:

1. Ανάλυση προδιαγραφών λογισμικού ,όπου οι πελάτες και οι μηχανικοί λογισμικού καθορίζουν το λογισμικό που θα παραχθεί και τους περιορισμούς στη λειτουργία του.

2. Ανάπτυξη λογισμικού ,όπου το λογισμικό σχεδιάζεται και προγραμματίζεται.

3. Επικύρωση λογισμικού ,όπου το λογισμικό ελέγχεται για να εξασφαλιστεί ότι είναι αυτό που ο πελάτης απαιτεί.

4. Η εξέλιξη λογισμικού ,όπου το λογισμικό τροποποιείται για να προσαρμοστεί στις αλλαγές των απαιτήσεων των πελατών και της αγοράς.

Διαφορετικοί τύποι συστημάτων απαιτούνται για τις διαφορετικές διαδικασίες ανάπτυξης. Παραδείγματος χάριν, το λογισμικό πραγματικού χρόνου σε ένα αεροσκάφος πρέπει να καθοριστεί εντελώς σε σχέση με τις προδιαγραφές του προτού να αρχίσει η ανάπτυξη ενώ, στα συστήματα ηλεκτρονικού εμπορίου, η προδιαγραφή και το πρόγραμμα αναπτύσσονται συνήθως παράλληλα. Συνεπώς, αυτές οι γενικές δραστηριότητες μπορούν να

οργανωθούν με διαφορετικούς τρόπους και να περιγραφούν σε διαφορετικά επίπεδα λεπτομέρειας για τους διαφορετικούς τύπους λογισμικών. Εντούτοις, η χρήση μιας ακατάλληλης διαδικασίας λογισμικού μπορεί να μειώσει την ποιότητα ή τη χρησιμότητα του προϊόντος λογισμικού που αναπτύσσεται ή/και να αυξήσει τις δαπάνες ανάπτυξης.

1.4 Μοντέλο ανάπτυξης λογισμικού

Το μοντέλο ανάπτυξης λογισμικού είναι μια απλοποιημένη περιγραφή μιας διαδικασίας λογισμικού που παριστάνει μια άποψη της διαδικασίας. Τα μοντέλα ανάπτυξης μπορεί να περιλαμβάνουν δραστηριότητες που είναι μέρος της διαδικασίας λογισμικού, προϊόντα λογισμικού και τους ρόλους των ατόμων που εμπλέκονται στον σχεδιασμό και στην υλοποίησή του. Μερικά παραδείγματα των τύπων των μοντέλων που μπορεί να παραχθούν είναι:

1. Ένα μοντέλο ροής εργασιών Αυτό δείχνει την αλληλουχία δραστηριοτήτων στην διαδικασία μαζί με τις εισόδους, εξόδους και εξαρτήσεις. Οι δραστηριότητες σε αυτό το μοντέλο αναπαριστούν ανθρώπινες ενέργειες.

2. Ένα μοντέλο ροής πληροφοριών ή δραστηριοτήτων Αυτό αναπαριστά τη διαδικασία ως σύνολο δραστηριοτήτων, κάθε μια από τις οποίες πραγματοποιεί κάποιο μετασχηματισμό δεδομένων. Δείχνει πώς η εισαγωγή στη διαδικασία, όπως μια προδιαγραφή, μετασχηματίζεται σε μια έξοδο, όπως ένα σχέδιο. Οι δραστηριότητες μπορούν εδώ να αντιπροσωπεύσουν τους μετασχηματισμούς που πραγματοποιούνται από τους ανθρώπους ή από τους υπολογιστές.

3. Ένα μοντέλο ρόλου/δράσης Αυτό αναπαριστά τους ρόλους των ανθρώπων που συμμετέχουν στη διαδικασία λογισμικού και τις δραστηριότητες για τις οποίες είναι υπεύθυνοι.

Τα περισσότερα πρότυπα διαδικασίας ανάπτυξης λογισμικού είναι βασισμένα στο ένα από τρία γενικά πρότυπα ή τα παραδείγματα της ανάπτυξης λογισμικού:

1. Η προσέγγιση του καταρράκτη Αυτό παίρνει τις ανωτέρω δραστηριότητες και τις αναπαριστά ως χωριστές φάσεις διαδικασίας όπως η προδιαγραφή απαιτήσεων, ο σχεδιασμός

λογισμικού, η υλοποίηση, ο έλεγχος και ούτω καθεξής. Αφότου καθορίζεται κάθε στάδιο, υπογράφεται, θεωρείται τετελεσμένο και η ανάπτυξη πηγαίνει προς το επόμενο στάδιο.

2. Η επαναληπτική ανάπτυξη Αυτή η προσέγγιση παρακάμπτει προσωρινά τις δραστηριότητες της προδιαγραφής, της ανάπτυξης και της επικύρωσης. Ένα αρχικό σύστημα αναπτύσσεται γρήγορα από τις πολύ γενικές προδιαγραφές. Αυτό τελειοποιείται έπειτα με τα δεδομένα των πελατών για να παραγάγει ένα σύστημα που ικανοποιεί τις ανάγκες των πελατών. Το σύστημα μπορεί έπειτα να παραδοθεί. Εναλλακτικά, μπορεί να υλοποιηθεί ξανά χρησιμοποιώντας μια πιο καλά δομημένη προσέγγιση ώστε να παράγει ένα πιο ισχυρό και συντηρήσιμο σύστημα.

3. Η βασισμένη στα συστατικά ανάπτυξη λογισμικού (CBSE- component based software engineering) Αυτή η τεχνική υποθέτει ότι τα μέρη του συστήματος υπάρχουν ήδη. Η αναπτυξιακή διαδικασία του συστήματος εστιάζει στην ενσωμάτωση αυτών των μερών παρά την ανάπτυξη τους από την αρχή.

Εκτιμώντας την σημασία και την χρησιμότητα των μοντέλων ανάπτυξης λογισμικού στην σχεδίαση και υλοποίησή του, αφιερώνουμε ξεχωριστό κεφάλαιο στην πτυχιακή μας όπου και τα αναλύουμε ενδελεχώς.

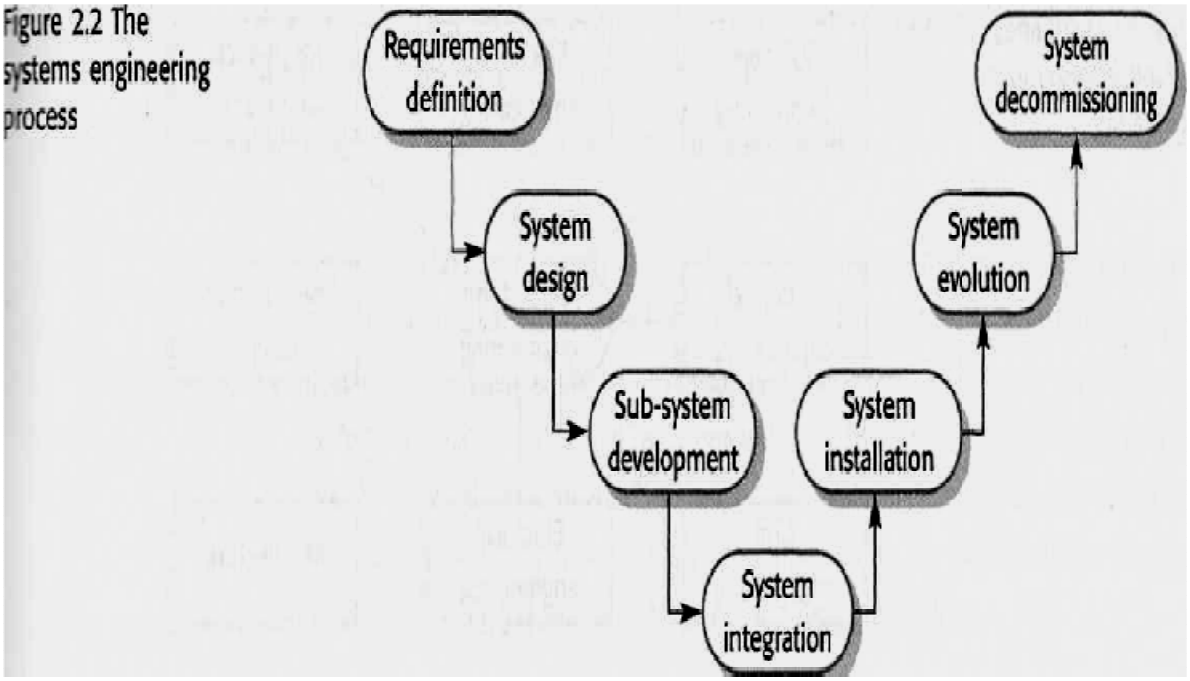
1.5 Σχεδιασμός και υλοποίηση συστημάτων

Ο σχεδιασμός και υλοποίηση συστημάτων είναι η δραστηριότητα της διευκρίνισης προδιαγραφών, του σχεδιασμού, της εφαρμογής, της επικύρωσης, της ανάπτυξης και της διατήρησης των κοινωνικό-τεχνικών συστημάτων. Με τον όρο κοινωνικό- τεχνικά συστήματα εννοούμε τα συστήματα που περιλαμβάνουν ανθρώπους, λογισμικό και υλικό.

Οι μηχανικοί τέτοιων συστημάτων δεν ενδιαφέρονται μόνο για το λογισμικό αλλά και για το υλικό και τις αλληλεπιδράσεις του συστήματος με τους χρήστες και το περιβάλλον του. Πρέπει να σκεφτούν για τις υπηρεσίες που το σύστημα παρέχει, τους περιορισμούς κάτω από τους οποίους το σύστημα πρέπει να χτιστεί και να χρησιμοποιηθεί και τους τρόπους με τους οποίους το σύστημα χρησιμοποιείται για να εκπληρώσει το σκοπό του. Όπως είπαμε, οι

μηχανικοί λογισμικού χρειάζονται μια κατανόηση του σχεδιασμού και υλοποίησης των συστημάτων επειδή τα προβλήματα της κατασκευής είναι συχνά ένα αποτέλεσμα των αποφάσεων των μηχανικών συστημάτων (Thayer, 1997 Thayer, 2002).

Οι φάσεις της διαδικασίας σχεδιασμού και υλοποίησης συστημάτων παρουσιάζονται στο σχήμα 1. Αυτή η διαδικασία ήταν μια σημαντική επιρροή στο waterfall πρότυπο ή αλλιώς μοντέλο καταρράκτη της διαδικασίας λογισμικού που αναλύουμε στο 2^{ο} κεφάλαιο της πτυχιακής μας.



Σχήμα 1 Η διαδικασία σχεδιασμού και υλοποίησης συστημάτων

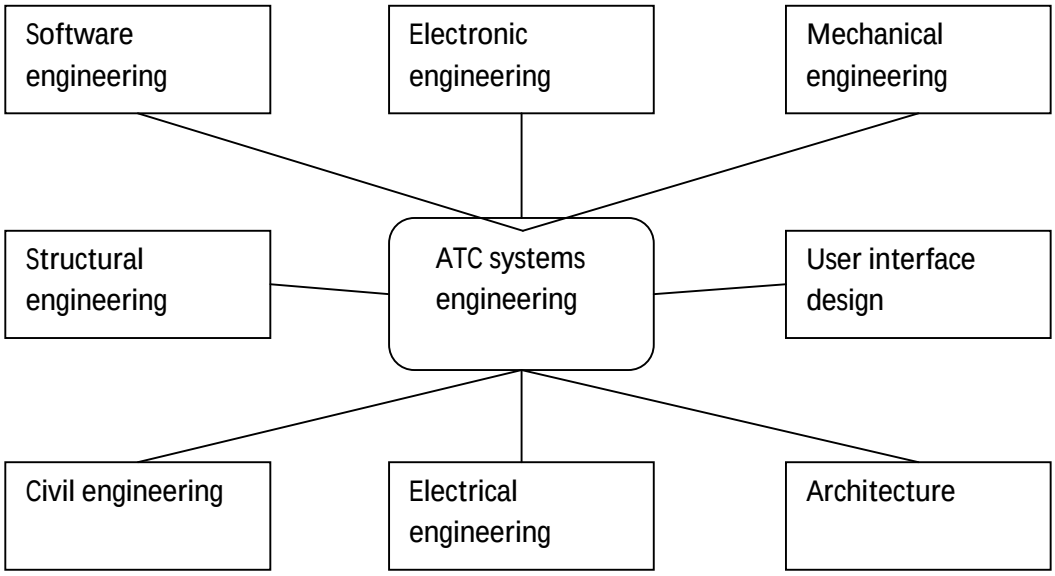
Υπάρχουν σημαντικές διακρίσεις μεταξύ της διαδικασίας σχεδιασμού και υλοποίησης συστημάτων και της διαδικασίας του σχεδιασμού και υλοποίησης λογισμικού:

1. Περιορισμένο πεδίο για ανακατασκευή κατά τη διάρκεια της ανάπτυξης συστημάτων Μόλις ληφθούν μερικές αποφάσεις κατασκευής συστημάτων, όπως η τοποθέτηση των σταθμών βάσης σε ένα κινητό τηλεφωνικό σύστημα, απαιτεί μεγάλο κόστος να αλλάξουν. Η ανακατασκευή συστημάτων για να λύσει αυτά τα προβλήματα είναι σπάνια εφικτή. Ένας λόγος που το λογισμικό έχει γίνει τόσο σημαντικό στα συστήματα είναι ότι επιτρέπει στις αλλαγές να γίνονται κατά τη διάρκεια της ανάπτυξης των συστημάτων, σε απάντηση στις νέες απαιτήσεις.

2. Η διεπιστημονική συμμετοχή Πολλές αρχές σχεδιασμού και υλοποίησης μπορούν να περιληφθούν στον σχεδιασμό και στην υλοποίηση συστημάτων. Υπάρχει πολύ πεδίο για παρανόηση καθώς διαφορετικοί μηχανικοί χρησιμοποιούν διαφορετική ορολογία και συμβάσεις.

Η κατασκευή συστημάτων είναι μια διεπιστημονική δραστηριότητα που περιλαμβάνει ομάδες που προέρχονται από διάφορα υπόβαθρα. Οι ομάδες υλοποίησης συστημάτων απαιτούνται λόγω της ευρείας γνώσης που απαιτείται για να εξετάσουμε όλες τις πτυχές των αποφάσεων σχεδίασης των συστημάτων.

Στο σχήμα 2 απεικονίζονται μερικές από τις αρχές που μπορούν να εμπλέκονται σε μια ομάδα υλοποίησης ενός συστήματος ελέγχου εναέριας κυκλοφορίας (ATC) που χρησιμοποιεί τα ραντάρ και άλλους αισθητήρες για να καθοριστούν οι θέσεις των αεροσκαφών.



Σχήμα 2: **Οι αρχές που εμπλέκονται στην υλοποίηση ενός συστήματος**

Για πολλά συστήματα, υπάρχουν σχεδόν άπειρες δυνατότητες για τις ανταλλαγές μεταξύ των διαφορετικών τύπων υποσυστημάτων. Οι διαφορετικές αρχές διαπραγματεύονται

να αποφασίσουν πώς η λειτουργικότητα πρέπει να παρασχεθεί. Συχνά δεν υπάρχει «σωστή»' απόφαση σχετικά με τον τρόπο με τον οποίο ένα σύστημα πρέπει να αποσυντεθεί. Μάλλον, μπορείτε να έχετε διάφορες πιθανές εναλλακτικές λύσεις, αλλά μπορείτε να μην είστε σε θέση να επιλέξετε την καλύτερη τεχνική λύση. Έστω ότι μια εναλλακτική λύση σε ένα σύστημα ελέγχου εναέριας κυκλοφορίας είναι να χτιστούν τα νέα ραντάρ παρά να επισκευαστούν οι υπάρχουσες εγκαταστάσεις. Εάν οι πολιτικοί μηχανικοί που συμμετέχουν σε αυτήν την διαδικασία δεν έχουν πολλή άλλη εργασία, μπορούν να ευνοήσουν αυτήν την εναλλακτική λύση επειδή επιτρέπει σε αυτούς να κρατήσουν τις εργασίες τους. Μπορούν έπειτα να υποστηρίξουν λογικά αυτήν την επιλογή με τα τεχνικά επιχειρήματα.

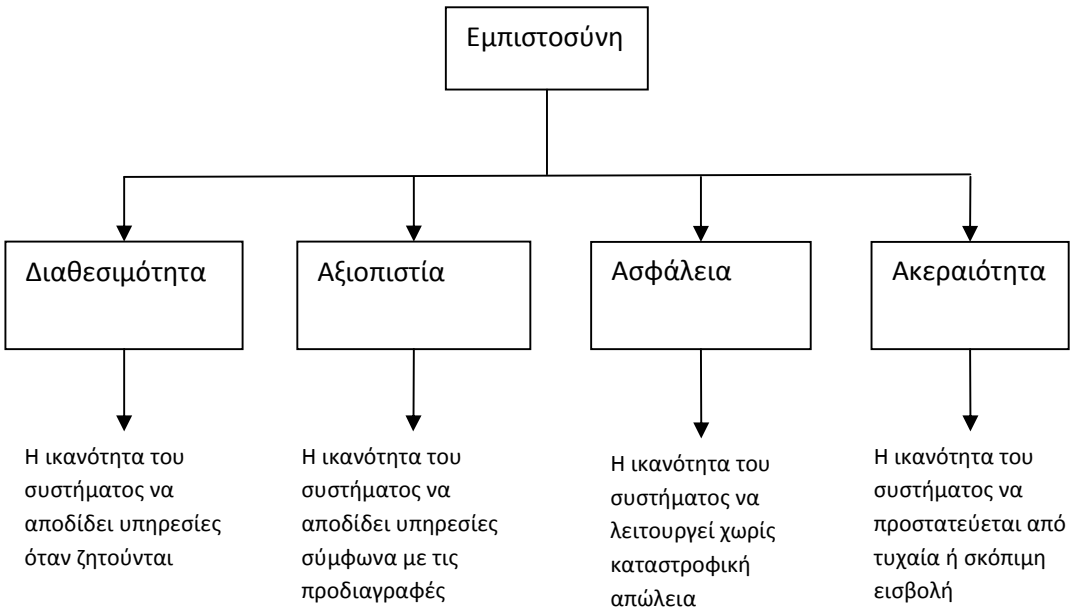
1.6 Εμπιστοσύνη (Αξιοπιστία)συστήματος

Όλοι μας γνωρίζουμε, ότι υπάρχει ο κίνδυνος απώλειας του λειτουργικού συστήματος. Χωρίς προφανή λόγο, τα συστήματα υπολογιστών πολλές φορές καταστρέφονται και αποτυγχάνουν να αποδώσουν τις υπηρεσίες που έχουν ζητηθεί. Τα προγράμματα που τρέχουν σε αυτούς τους υπολογιστές μπορεί να μην λειτουργούν όπως αναμένεται και περιστασιακά μπορεί να βλάπτουν τα δεδομένα που διαχειρίζεται το σύστημα. Έχουμε μάθει να ζούμε με αυτές τις απώλειες και λίγοι από εμάς εμπιστευόμαστε απόλυτα το υπολογιστικό σύστημα που συνήθως χειριζόμαστε.

Η αξιοπιστία ενός συγκροτήματος ηλεκτρονικών υπολογιστών είναι μια ιδιότητα του συστήματος που ισοδυναμεί με την εμπιστοσύνη του. Η εμπιστοσύνη σημαίνει ουσιαστικά το βαθμό αξιοπιστίας των χρηστών, ότι το σύστημα θα λειτουργήσει όπως αναμένουν και ότι το σύστημα δεν θα αποτύχει' σε κανονική χρήση. Αυτή η ιδιότητα δεν μπορεί να εκφραστεί αριθμητικά, αλλά χρησιμοποιούμε τους σχετικούς όρους όπως ' όχι βάσιμο' , ' πολύ βάσιμο' και ' εξαιρετικά-βάσιμο' για να απεικονίσει τους βαθμούς εμπιστοσύνης που μπορούμε να έχουμε σε ένα σύστημα.

Η εμπιστοσύνη και η χρησιμότητα δεν είναι, φυσικά, το ίδιο πράγμα. Δεν νομίζω ότι ο επεξεργαστής λέξεων που χρησιμοποίησα για να γράψω αυτό το βιβλίο είναι ένα πολύ αξιόπιστο σύστημα, αλλά είναι πολύ χρήσιμο. Εντούτοις, για να απεικονίσουμε την έλλειψη

εμπιστοσύνης μας στο σύστημα συχνά κάνουμε save την εργασία και κρατάμε πολλαπλά εφεδρικά αντίγραφα από αυτήν. Αντισταθμίζω την έλλειψη αξιοπιστίας συστήματος με ενέργειες που περιορίζουν τη ζημία που θα μπορούσε να προκληθεί εάν το σύστημα αποτύγχανε.



Σχήμα 3: Οι τέσσερις διαστάσεις της βασιμότητας των συστημάτων

[Sommerville, 2008, Software Engineering 8th edition]

Υπάρχουν τέσσερις κύριες διαστάσεις στην αξιοπιστία, όπως φαίνεται στο σχήμα 3:

1.Διαθεσιμότητα Ανεπίσημα, η διαθεσιμότητα ενός συστήματος είναι η πιθανότητα ότι θα είναι σε λειτουργία και ικανό να παραδώσει τις χρήσιμες υπηρεσίες οποιαδήποτε στιγμή.

2. Αξιοπιστία Ανεπίσημα, η αξιοπιστία ενός συστήματος είναι η πιθανότητα, κατά τη διάρκεια μιας δεδομένης χρονικής περιόδου το σύστημα να παραδώσει σωστά τις υπηρεσίες όπως αναμένεται στο χρήστη.

3. Ασφάλεια, Ανεπίσημα, η ασφάλεια ενός συστήματος είναι μια κρίση για το πόσο πιθανό είναι το σύστημα να προκαλέσει ζημία στους ανθρώπους ή το περιβάλλον του.

4. Ακεραιότητα, Ανεπίσημα, η ακεραιότητα ενός συστήματος είναι μια κρίση για το πόσο πιθανό είναι το σύστημα να μπορεί να αντισταθεί στις τυχαίες ή σκόπιμες παρεισφρήσεις.

Αυτές είναι περίπλοκες ιδιότητες που μπορεί να αναλυθούν σε ένα αριθμό άλλων, πιο απλών ιδιοτήτων. Για παράδειγμα, η ασφάλεια περιέχει την ακεραιότητα (που διαβεβαιώνει ότι τα προγράμματα και δεδομένα του συστήματος δεν βλάπτονται) και την εμπιστευτικότητα (που διαβεβαιώνει ότι οι πληροφορίες μπορεί να προσπελαύνονται μόνο από άτομα που είναι εξουσιοδοτημένα). Η αξιοπιστία περιέχει την διορθωσιμότητα (βεβαιώνεται ότι οι υπηρεσίες του συστήματος προσδιορίζονται), την ακρίβεια (βεβαιώνοντας ότι η πληροφορία διανέμεται σε ένα ικανοποιητικό επίπεδο λεπτομέρειας) και αμεσότητα (διαβεβαιώνοντας πως η πληροφορία φτάνει όταν πρέπει και απαιτείται).

Όπως και αυτές οι τέσσερις κύριες διαστάσεις, άλλες ιδιότητες συστημάτων μπορούν επίσης να είναι υπό τον τίτλο της αξιοπιστίας:

1. Επισκευασιμότητα Οι διακοπές του συστήματος είναι αναπόφευκτες, αλλά η διακοπή που προκαλείται από την απώλεια μπορεί να ελαχιστοποιηθεί εάν το σύστημα μπορεί να επισκευαστεί γρήγορα. Για να συμβεί αυτό, πρέπει να είναι δυνατό να εντοπιστεί το πρόβλημα, να υπάρχει πρόσβαση στο συστατικό που έχει αποτύχει και να γίνουν οι απαραίτητες ενέργειες επιδιόρθωσης του συστατικού. Η επισκευασιμότητα στα λογισμικά ενισχύεται όταν η επιχείρηση που χρησιμοποιεί το σύστημα έχει πρόσβαση στον πηγαίο κώδικα και έχει τις δεξιότητες για να κάνει τις αλλαγές σε αυτόν. Δυστυχώς, αυτό γίνεται όλο και περισσότερο ασυνήθιστο καθώς κινούμαστε προς την ανάπτυξη συστημάτων χρησιμοποιώντας τρίτο οργανισμό, τμήματα μαύρων κουτιών .

2. Η συντηρησιμότητα Καθώς τα συστήματα χρησιμοποιούνται, νέες απαιτήσεις προκύπτουν. Είναι σημαντικό να διατηρηθεί η χρησιμότητα ενός συστήματος με την αλλαγή του για να προσαρμοστεί σε αυτές τις νέες απαιτήσεις. Το συντηρήσιμο λογισμικό είναι λογισμικό που μπορεί να προσαρμοστεί οικονομικά για να αντιμετωπίσει τις νέες απαιτήσεις

και στο οποίο υπάρχει μια χαμηλή πιθανότητα κάνοντας αλλαγές να εισαγάγουμε νέα λάθη στο σύστημα.

3. Επιβιωσιμότητα Μια πολύ σημαντική ιδιότητα για συστήματα που βασίζονται στο διαδίκτυο, που είναι στενά συνδεδεμένη με την ασφάλεια και τη διαθεσιμότητα (Ellison, και λοιποί., 1999). Η ικανότητα επιβίωσης είναι η δυνατότητα ενός συστήματος να συνεχίσει να αποδίδει την υπηρεσία ενώ είναι κάτω από επίθεση και, ενδεχομένως, ενώ μέρος του συστήματος τίθεται εκτός λειτουργίας. Η εργασία για την ικανότητα επιβίωσης εστιάζει στον προσδιορισμό των βασικών τμημάτων συστήματος και την εξασφάλιση ότι μπορεί να παραδώσει την ελάχιστη υπηρεσία. Τρεις στρατηγικές χρησιμοποιούνται για να ενισχύσουν την ικανότητα επιβίωσης- συγκεκριμένα, αντίσταση στην επίθεση, η αναγνώριση επίθεσης και η αποκατάσταση από τη ζημία που προκαλείται από μια επίθεση (Ellison, και λοιποί., 1999 Ellison, και λοιποί., 2002).

4. Ανοχή στο λάθος Η ιδιότητα αυτή, μπορεί να εξεταστεί ως τμήμα της δυνατότητας χρησιμοποίησης και απεικονίζει το βαθμό στον οποίο το σύστημα έχει σχεδιαστεί έτσι ώστε σφάλματα που εισάγονται από τον χρήστη να αποφεύγονται και να τα ανέχεται το σύστημα. Όταν τα λάθη χρηστών εμφανίζονται, το σύστημα πρέπει, όσο το δυνατόν περισσότερο, να ανιχνεύσει αυτά τα λάθη και είτε να τα διορθώσει αυτόματα, είτε να ζητήσει από το χρήστη για να εισαγάγει ξανά τα στοιχεία του.

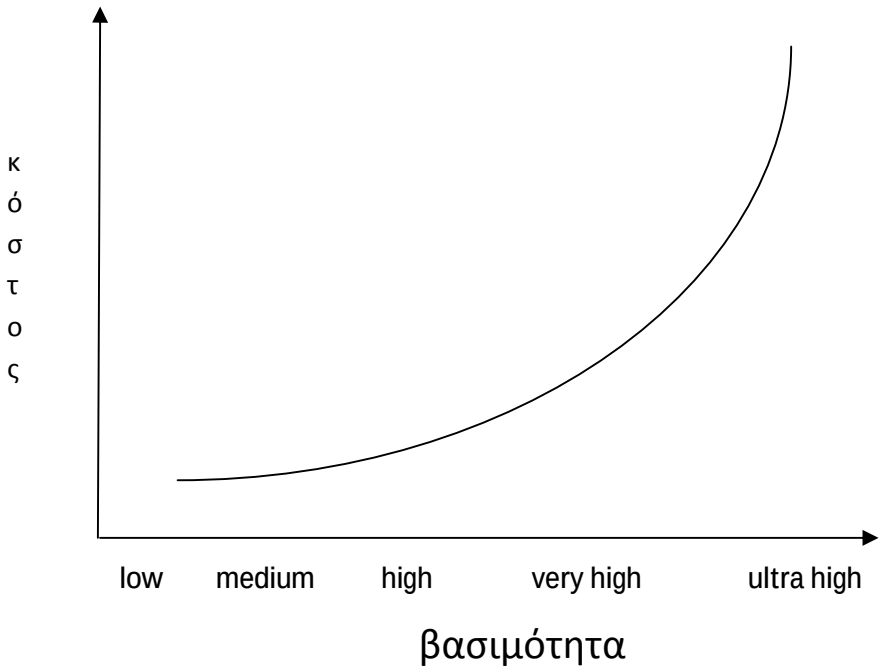
Επειδή η διαθεσιμότητα, η αξιοπιστία, η ασφάλεια και η ακεραιότητα είναι θεμελιώδεις ιδιότητες , επικεντρωνόμαστε σε αυτές σε αυτό το κεφάλαιο.

Φυσικά, αυτές οι ιδιότητες βασιμότητας δεν είναι όλες εφαρμόσιμες σε όλα τα συστήματα. Για παράδειγμα για ένα σύστημα λειτουργίας μιας αντλίας νερού οι σημαντικότερες ιδιότητες είναι η διαθεσιμότητα (πρέπει να λειτουργήσει σε περίπτωση ανάγκης), αξιοπιστία (πρέπει να παραδώσει τη σωστή δόση νερού) και ασφάλεια (δεν πρέπει ποτέ να συμβεί ηλεκτρική εκκένωση).

Οι σχεδιαστές πρέπει συνήθως να κάνουν μια ανταλλαγή μεταξύ της απόδοσης συστημάτων και της αξιοπιστίας. Γενικά, τα υψηλά επίπεδα της αξιοπιστίας μπορούν μόνο να

επιτευχθούν εις βάρος της απόδοσης συστημάτων. Το αξιόπιστο λογισμικό περιλαμβάνει επιπλέον, συχνά άφθονο, κώδικα για να εκτελέσει τον απαραίτητο έλεγχο για τις εξαιρέσεις συστημάτων και για να ανακτήσει το σύστημα από τα ελαττώματα . Αυτό μειώνει την απόδοση συστημάτων και αυξάνει τον αποθηκευτικό χώρο που απαιτείται από το λογισμικό. Προσθέτει επίσης σημαντικά στις δαπάνες της ανάπτυξης συστημάτων.

Λόγω του πρόσθετου σχεδίου, οι δαπάνες εφαρμογής και επικύρωσης, που αυξάνουν την αξιοπιστία ενός συστήματος μπορούν σημαντικά να αυξήσουν τις δαπάνες ανάπτυξης. Συγκεκριμένα οι δαπάνες επικύρωσης είναι υψηλές για τα κρίσιμα συστήματα. Όπως και η επικύρωση ότι το σύστημα καλύπτει τις απαιτήσεις του, η διαδικασία επικύρωσης μπορεί να πρέπει να αποδείξει σε έναν εξωτερικό ρυθμιστή όπως για παράδειγμα έναν φορέα πιστοποίησης ότι το σύστημα είναι αξιόπιστο.



Σχήμα 4: καμπύλη κόστους-αξιοπιστίας

Το σχήμα 4 παρουσιάζει τη σχέση μεταξύ των δαπανών και των αυξητικών βελτιώσεων στην αξιοπιστία. Όσο υψηλότερη η αξιοπιστία που χρειάζεστε, τόσο περισσότερο πρέπει να ξοδέψετε στη δοκιμή για να ελέγξετε ότι έχετε φθάσει σε εκείνο το επίπεδο. Εξαιτίας της εκθετικής φύσης της σχέσης κόστους/ αξιοπιστίας, δεν είναι δυνατό να δηλώσουμε ότι ένα

σύστημα είναι 100% αξιόπιστο, καθώς τα κόστη της βεβαίωσης βασιμότητας μπορεί να είναι άπειρα.

1.7 Διαθεσιμότητα και Αξιοπιστία

Η διαθεσιμότητα και η αξιοπιστία συστημάτων είναι στενά συνδεδεμένες ιδιότητες που μπορούν και οι δύο να εκφραστούν ως αριθμητικές πιθανότητες. Η αξιοπιστία ενός συστήματος είναι η πιθανότητα οι υπηρεσίες του συστήματος να παραδοθούν σωστά όπως προδιαγράφονται. Η διαθεσιμότητα ενός συστήματος είναι η πιθανότητα το σύστημα να είναι σε υπηρεσία για να παραδώσει αυτές τις υπηρεσίες στους χρήστες όταν τις ζητούν.

Αν και είναι στενά συνδεδεμένα, δεν μπορείτε να υποθέσετε ότι τα αξιόπιστα συστήματα θα είναι πάντα διαθέσιμα και αντίστροφα. Παραδείγματος χάριν, μερικά συστήματα μπορούν να έχουν μια υψηλή απαίτηση διαθεσιμότητας αλλά μια πολύ χαμηλότερη απαίτηση αξιοπιστίας. Εάν οι χρήστες αναμένουν τη συνεχή υπηρεσία έπειτα οι απαιτήσεις διαθεσιμότητας είναι υψηλές. Εντούτοις, εάν οι συνέπειες μιας αποτυχίας είναι ελάχιστες και το σύστημα μπορεί να ανακτήσει γρήγορα από αυτές τις αποτυχίες έπειτα το ίδιο σύστημα μπορεί να έχει τις χαμηλές απαιτήσεις αξιοπιστίας.

Ένα παράδειγμα ενός συστήματος όπου η διαθεσιμότητα είναι κρισιμότερη από την αξιοπιστία είναι ένας διακόπτης ανταλλαγής τηλεφώνου. Οι χρήστες αναμένουν έναν τόνο κλήσης όταν παίρνουν ένα τηλέφωνο έτσι το σύστημα έχει υψηλές απαιτήσεις διαθεσιμότητας. Εντούτοις, εάν ένα ελάττωμα αναγκάζει μια σύνδεση να αποτύχει, αυτό είναι συχνά ανακτήσιμο. Οι διακόπτες ανταλλαγής περιέχουν συνήθως τις ευκολίες επισκευής που μπορούν να επανεκκινήσουν το σύστημα και να ξαναδοκιμάσουν την προσπάθεια σύνδεσης. Αυτό μπορεί να γίνει πολύ γρήγορα, και ο χρήστης δεν μπορεί καν να παρατηρήσει ότι μια αποτυχία έχει εμφανιστεί. Επομένως, διαθεσιμότητα παρά αξιοπιστία είναι η βασική απαίτηση βασιμότητας για αυτά τα συστήματα.

Μια περαιτέρω διάκριση μεταξύ αυτών των χαρακτηριστικών είναι ότι η διαθεσιμότητα δεν εξαρτάται απλά από το ίδιο το σύστημα αλλά και από το χρόνο που απαιτείται για να επισκευάσει τα ελαττώματα που καθιστούν το σύστημα μη διαθέσιμο. Επομένως, εάν το σύστημα Α αποτυγχάνει μία φορά το χρόνο, και το σύστημα Β αποτυγχάνει μια φορά το μήνα, κατόπιν το Α είναι σαφώς πιο αξιόπιστο από το Β. Εντούτοις, υποθέστε ότι εκείνο το σύστημα Α διαρκεί τρεις ημέρες στο καινούριο ξεκίνημα μετά από μια αποτυχία, ενώ το σύστημα Β διαρκεί 10 λεπτά στο καινούριο ξεκίνημα. Η διαθεσιμότητα του συστήματος Β κατά τη διάρκεια του έτους (120 λεπτά του εκ/εν - χρόνου) είναι πολύ καλύτερη από αυτή του συστήματος Α (4.320 λεπτά εκ/εν - χρόνου).

Η αξιοπιστία και η διαθεσιμότητα συστημάτων μπορούν να καθοριστούν ακριβέστερα ως εξής:

1. Αξιοπιστία η πιθανότητα της χωρίς-αποτυχία λειτουργίας κατά τη διάρκεια ενός καθορισμένου χρόνου σε ένα δεδομένο περιβάλλον για έναν συγκεκριμένο σκοπό.

2. Διαθεσιμότητα η πιθανότητα ότι ένα σύστημα, σε μια συγκεκριμένη στιγμή, θα είναι λειτουργικό και ικανό να παραδώσει τις απαιτούμενες υπηρεσίες.

Ένα από τα πρακτικά προβλήματα στην ανάπτυξη των αξιόπιστων συστημάτων είναι ότι οι έννοιές της αξιοπιστίας και της διαθεσιμότητας είναι μερικές φορές ευρύτερες από τους παραπάνω περιορισμένους ορισμούς. Ο καθορισμός της αξιοπιστίας δηλώνει ότι το περιβάλλον στο οποίο το σύστημα χρησιμοποιείται και ο σκοπός για τον οποίο χρησιμοποιείται θα πρέπει να ληφθεί υπόψη. Εάν μετράτε την αξιοπιστία συστημάτων σε ένα περιβάλλον, δεν μπορείτε να υποθέσετε ότι η αξιοπιστία θα είναι η ίδια σε ένα άλλο περιβάλλον όπου το σύστημα χρησιμοποιείται με έναν διαφορετικό τρόπο.

Παραδείγματος χάριν, ας πούμε ότι μετράτε την αξιοπιστία ενός επεξεργαστή λέξεων σε ένα περιβάλλον γραφείου όπου οι περισσότεροι χρήστες είναι αδιάφοροι στη λειτουργία

του λογισμικού. Ακολουθούν τις οδηγίες για τη χρήση του και δεν προσπαθούν να πειραματιστούν με το σύστημα. Εάν μετράτε την αξιοπιστία του ίδιου συστήματος σε ένα πανεπιστημιακό περιβάλλον, τότε η αξιοπιστία μπορεί να είναι αρκετά διαφορετική. Εδώ, οι σπουδαστές μπορούν να ερευνήσουν τα όρια του συστήματος και να χρησιμοποιήσουν το σύστημα με απροσδόκητους τρόπους. Αυτό μπορεί να οδηγήσει σε διακοπές του συστήματος που δεν εμφανίστηκαν στο περιορισμένο περιβάλλον γραφείου.

Οι ανθρώπινες αντιλήψεις και τα σχέδια της χρήσης είναι επίσης σημαντικές. Παραδείγματος χάριν, έστω ένα αυτοκίνητο έχει ένα ελάττωμα στο σύστημα υαλοκαθαριστήρων του που οδηγεί στις διαλείπουσες αποτυχίες των σφουγγαριών να λειτουργήσουν σωστά στη δυνατή βροχή. Η αξιοπιστία εκείνου του συστήματος σύμφωνα με έναν οδηγό εξαρτάται από το που ζουν και πως χρησιμοποιούν το αυτοκίνητο. Ένας οδηγός στο Σιάτλ (υγρό κλίμα) πιθανώς περισσότερο θα επηρεαστεί από αυτήν την αποτυχία από έναν οδηγό στο Λας Βέγκας (ξηρό κλίμα). Το Σιάτλ η αντίληψη του οδηγού θα είναι ότι το σύστημα είναι αναξιόπιστο, ενώ ο οδηγός στο Λας Βέγκας δεν μπορεί ποτέ να παρατηρήσει το πρόβλημα.

Μια περαιτέρω δυσκολία με αυτούς τους ορισμούς είναι ότι δεν λαμβάνουν υπόψη τη δριμύτητα της αποτυχίας ή τις συνέπειες της α-διαθεσιμότητας. Οι άνθρωποι, φυσικά, ανησυχούν για τις διακοπές του συστήματος που έχουν τις σοβαρές συνέπειες, και η αντίληψή τους της αξιοπιστίας των συστημάτων επηρεάζεται από αυτές τις συνέπειες. Παραδείγματος χάριν, έστω μια αποτυχία εκκίνησης που οφείλεται στο λογισμικό της μηχανής ενός αυτοκινήτου, που (σβήνει λίγο μετά αφού πήρε μπροστά), αλλά λειτουργεί σωστά μετά από ένα καινούριο ξεκίνημα (επανεκκίνηση) που διορθώνει το πρόβλημα έναρξης. Αυτό δεν έχει επιπτώσεις στην κανονική λειτουργία του αυτοκινήτου, και πολλοί οδηγοί δεν θα σκέφτονταν ότι μια επισκευή απαιτείται. Σε αντίθεση, οι περισσότεροι οδηγοί θα σκεφτούν ότι μια μηχανή που σβήνει ενώ οδηγούν στη υψηλή ταχύτητα μια φορά το μήνα (έστω) είναι και αναξιόπιστη και επισφαλής και πρέπει να επισκευαστεί.

Ένας αυστηρός ορισμός της αξιοπιστίας συσχετίζει την εφαρμογή του συστήματος με τις προδιαγραφές του. Δηλαδή το σύστημα συμπεριφέρεται αξιόπιστα εάν η συμπεριφορά του είναι σύμφωνη με αυτήν που καθορίζεται στην προδιαγραφή. Εντούτοις, μια κοινή αιτία προφανούς αναξιοπιστίας είναι ότι η προδιαγραφή συστημάτων δεν ταιριάζει με τις

προσδοκίες των χρηστών του συστήματος. Δυστυχώς, πολλές προδιαγραφές είναι ελλιπείς ή ανακριβείς και αφήνεται στους μηχανικούς λογισμικού να ερμηνεύσουν πώς το σύστημα πρέπει να συμπεριφερθεί. Δεδομένου ότι δεν είναι έμπειροι στον τομέα, μπορούν, επομένως, να μην εφαρμόσουν τη συμπεριφορά που οι χρήστες αναμένουν.

Η αξιοπιστία και η διαθεσιμότητα εκτίθενται από τις διακοπές του συστήματος. Αυτές μπορούν να είναι μια αποτυχία για να παρασχεθεί μια υπηρεσία, μια αποτυχία για να παραδοθεί μια υπηρεσία όπως προδιαγράφεται, ή η παροχή μιας υπηρεσίας κατά τέτοιο τρόπο ώστε είναι επισφαλής ή ευπρόσβλητη. Μερικές από αυτές τις αποτυχίες είναι μια συνέπεια των λαθών ή των αποτυχιών προδιαγραφών στα σχετικά συστήματα όπως ένα σύστημα τηλεπικοινωνιών. Εντούτοις, πολλές αποτυχίες είναι μια συνέπεια της λανθασμένης συμπεριφοράς του συστήματος που προέρχεται από τα ελαττώματα στο σύστημα. Όταν γίνεται λόγος για αξιοπιστία, είναι χρήσιμο να γίνεται διάκριση μεταξύ των όρων του ελαττώματος, του λάθους και της αποτυχίας. Στον πίνακα 2 γίνεται η διάκρισή τους.

Όρος	Περιγραφή
Αποτυχία συστήματος	Γεγονός που συμβαίνει σε κάποιο χρόνο όπου το σύστημα δεν αποδίδει την λειτουργία που αναμένεται από τους χρήστες του
Ελάττωμα συστήματος (error)	Μια κατάσταση ελαττωματικού συστήματος που μπορεί να οδηγήσει σε συμπεριφορά του συστήματος που δεν αναμένουν οι χρήστες του
Λάθος συστήματος (fault)	Χαρακτηριστικό του λογισμικού συστήματος που μπορεί να οδηγήσει σε ελάττωμα.
Ανθρώπινο σφάλμα ή λάθος	Ανθρώπινη συμπεριφορά που οδηγεί στην εισαγωγή λαθών στο σύστημα

Πίνακας 2 [Sommerville, 2008, Software Engineering 8th edition]

1.8 Ασφάλεια (safety) και ασφάλιση (security)

Τα ασφάλειας-κρίσιμα συστήματα (safety-critical systems) είναι συστήματα όπου είναι απαραίτητο ότι η λειτουργία τους να είναι πάντα ασφαλής. Δηλαδή το σύστημα δεν πρέπει ποτέ να βλάψει τους ανθρώπους ή το περιβάλλον του συστήματος ακόμα κι αν το σύστημα αποτυγχάνει. Παραδείγματα των ασφάλειας-κρίσιμων συστημάτων είναι τα συστήματα καταγραφής και ελέγχου στα αεροσκάφη, τα συστήματα ελέγχου διαδικασίας στις χημικές και φαρμακευτικές εγκαταστάσεις και τα αυτοκινητικά συστήματα ελέγχου.

Ο έλεγχος υλικού των ασφάλειας-κρίσιμων συστημάτων είναι απλούστερος να εφαρμοστεί και να αναλυθεί από τον έλεγχο λογισμικού. Εντούτοις, χτίζουμε τώρα τα συστήματα τέτοιας πολυπλοκότητας που δεν μπορούν να ελεγχθούν από το υλικό μόνο. Κάποιος έλεγχος λογισμικού είναι αναγκαίος λόγω της ανάγκης να ρυθμιστούν οι μεγάλοι αριθμοί αισθητήρων και ενεργοποιητών με τους σύνθετους νόμους ελέγχου. Ένα παράδειγμα τέτοιας πολυπλοκότητας βρίσκεται στο προηγμένο, αεροδυναμικά ασταθές στρατιωτικό αεροπλάνο. Απαιτούν τη συνεχή λογισμικά-ελεγχόμενη ρύθμιση των επιφανειών πτήσης τους για να εξασφαλίσουν ότι δεν συντρίβουν.

Το ασφάλειας-κρίσιμο λογισμικό περιέρχεται σε δύο κατηγορίες:

1. *Το αρχικό, ασφάλειας-κρίσιμο λογισμικό* Αυτό είναι λογισμικό που ενσωματώνεται ως ελεγκτής σε ένα σύστημα. Πιθανή δυσλειτουργία τέτοιου λογισμικού μπορεί να προκαλέσει δυσλειτουργία υλικού, το οποίο οδηγεί στον ανθρώπινη τραυματισμό ή τη ατμοσφαιρική ρύπανση. Εστιάζω σε αυτόν τον τύπο λογισμικού.

2. *Το δευτεροβάθμιο ασφάλειας-κρίσιμο λογισμικό* Αυτό είναι λογισμικό που μπορεί έμμεσα να οδηγήσει στον τραυματισμό. Τα παραδείγματα τέτοιων συστημάτων είναι με τη βοήθεια υπολογιστή συστήματα σχεδίου εφαρμοσμένης μηχανικής των οποίων πιθανή δυσλειτουργία μπορεί να οδηγήσει σε ένα ελάττωμα σχεδίου στο αντικείμενο που σχεδιάζεται. Αυτό το ελάττωμα μπορεί να προκαλέσει τον τραυματισμό στους ανθρώπους

εάν το σύστημα που σχεδιάζεται δυσλειτουργήσει. Ένα άλλο παράδειγμα ενός δευτεροβάθμιου ασφάλειας-κρίσιμου συστήματος είναι μια βάση δεδομένων ιατρείου όπου κρατούνται λεπτομέρειες των φαρμάκων που χορηγούνται στους ασθενείς. Τα λάθη σε αυτό το σύστημα μπορεί να οδηγήσουν σε μια ανακριβή δόση φαρμάκων που χορηγείται.

Η ασφάλεια (security) με την έννοια της ασφάλισης, είναι μια ιδιότητα του συστήματος που απεικονίζει τη δυνατότητα του συστήματος να προστατευθεί από τις εξωτερικές επιθέσεις που μπορούν να είναι τυχαίες ή σκόπιμες. Η ασφάλεια έχει γίνει αυξητικά σημαντική όπως όλο και περισσότερα συστήματα συνδέονται με το Διαδίκτυο. Οι συνδέσεις στο Διαδίκτυο παρέχουν την πρόσθετη λειτουργία συστημάτων (π.χ., οι πελάτες μπορούν να είναι σε θέση να έχουν πρόσβαση στους τραπεζικούς λογαριασμούς τους άμεσα), αλλά η σύνδεση στο Διαδίκτυο επίσης σημαίνει ότι το σύστημα μπορεί να επιτεθεί από τους ανθρώπους με πιθανές εχθρικές προθέσεις. Η σύνδεση στο Διαδίκτυο επίσης σημαίνει ότι οι λεπτομέρειες των συγκεκριμένων ευπαθειών των συστημάτων μπορούν να διαδοθούν εύκολα έτσι ώστε περισσότεροι άνθρωποι μπορούν να είναι σε θέση να επιτεθούν στο σύστημα. Ομοίως, εντούτοις, η σύνδεση μπορεί να επιταχύνει τη διανομή των μπαλωμάτων συστημάτων για να επισκευάσει αυτές τις ευπάθειες.

Παραδείγματα των επιθέσεων μπορεί να είναι ιοί, αναρμόδια χρήση των υπηρεσιών συστημάτων και αναρμόδια τροποποίηση του συστήματος ή των δεδομένων του. Η ασφάλεια είναι σημαντική για όλα τα κρίσιμα συστήματα. Χωρίς ένα λογικό επίπεδο ασφάλειας, η διαθεσιμότητα, η αξιοπιστία και η ασφάλεια του συστήματος μπορούν να εκτεθούν εάν οι εξωτερικές επιθέσεις προκαλούν κάποια ζημιά στο σύστημα.

Ο λόγος για αυτό είναι ότι όλες οι μέθοδοι που βεβαιώνουν την διαθεσιμότητα, την αξιοπιστία και την ασφάλεια στηρίζονται στο γεγονός ότι το λειτουργικό σύστημα είναι το ίδιο με το σύστημα που εγκαταστάθηκε αρχικά. Εάν αυτό το εγκατεστημένο σύστημα έχει εκτεθεί με κάποιο τρόπο (παραδείγματος χάριν, εάν το λογισμικό έχει τροποποιηθεί για να περιλάβει έναν ιό), κατόπιν τα αιτήματα για την αξιοπιστία και την ασφάλεια που έγιναν αρχικά δεν μπορούν πλέον να κρατηθούν. Το λογισμικό του συστήματος μπορεί να αλλοιωθεί και μπορεί να συμπεριφερθεί με έναν απρόβλεπτο τρόπο.

Αντιθέτως, τα λάθη στην ανάπτυξη ενός συστήματος μπορούν να οδηγήσουν στις ασφαλιστικές δικλείδες. Εάν ένα σύστημα δεν αποκρίνεται στις απροσδόκητες εισαγωγές ή εάν τα όρια σειράς δεν ελέγχονται, τότε οι εισβολείς μπορεί να εκμεταλλευτούν αυτές τις

αδυναμίες ώστε να αποκτήσουν πρόσβαση στο σύστημα. Τα σημαντικότερα περιστατικά ασφάλειας όπως το αυθεντικό σκουλήκι του διαδικτύου (Spafford,1989) και το σκουλήκι κόκκινου κώδικα (Berghel, 2001) εκμεταλλεύτηκαν το γεγονός ότι τα προγράμματα στη C δεν περιέχουν έλεγχο ορίων σειράς. Έγραφαν πάνω σε μέρος της μνήμης που επέτρεπε μη εξουσιοδοτημένη προσπέλαση στο σύστημα.

Φυσικά, σε μερικά κρίσιμα συστήματα, η ασφάλεια είναι η σημαντικότερη διάσταση της αξιοπιστίας συστημάτων. Τα στρατιωτικά συστήματα, συστήματα για το ηλεκτρονικό εμπόριο και συστήματα που περιλαμβάνουν την επεξεργασία και την ανταλλαγή της εμπιστευτικής πληροφορίας πρέπει να σχεδιαστούν έτσι ώστε επιτυγχάνουν ένα υψηλό επίπεδο της ασφάλειας. Εάν ένα αεροπορικό σύστημα κράτησης (έστω) είναι μη διαθέσιμο, αυτό προκαλεί τη δυσχέρεια και μερικές καθυστερήσεις με την έκδοση των εισιτηρίων. Εντούτοις, εάν το σύστημα είναι επισφαλές και μπορεί να δεχτεί τις πλαστές κρατήσεις έπειτα η αερογραμμή που είναι κύρια του συστήματος μπορεί να χάσει πολλά χρήματα.

Υπάρχουν τρεις τύποι ζημιών που μπορούν να προκληθούν μέσω της εξωτερικής επίθεσης:

1. *Η άρνηση της υπηρεσίας* Το σύστημα μπορεί να αναγκαστεί σε μια κατάσταση όπου οι κανονικές του λειτουργίες γίνονται μη διαθέσιμες. Αυτό, προφανώς, έχει επιπτώσεις έπειτα στη διαθεσιμότητα του συστήματος.

2. *Τη φθορά των προγραμμάτων ή των δεδομένων* Τα τμήματα του λογισμικού του συστήματος μπορεί να αλλάξουν με έναν αναρμόδιο τρόπο. Αυτό μπορεί να έχει επιπτώσεις στη συμπεριφορά του συστήματος και ως εκ τούτου να βλαπτούν η αξιοπιστία και η ασφάλειά του. Εάν η ζημία είναι αυστηρή, η διαθεσιμότητα του συστήματος μπορεί να επηρεαστεί.

3. *Η κοινοποίηση της εμπιστευτικής πληροφορίας* Οι πληροφορίες που διαχειρίζονται από το σύστημα μπορεί να είναι εμπιστευτικές, και η εξωτερική επίθεση μπορεί να εκθέσει αυτό σε άτομα μη εξουσιοδοτημένα. Ανάλογα με τον τύπο δεδομένων, αυτό θα μπορούσε να έχει επιπτώσεις στην ασφάλεια του συστήματος και μπορεί να επιτρέψει αργότερα σε επιθέσεις που έχουν επιπτώσεις στη διαθεσιμότητα ή την αξιοπιστία του συστήματος.

Όπως με άλλες πτυχές της αξιοπιστίας, υπάρχει μια εξειδικευμένη ορολογία που σχετίζεται με την ασφάλεια. Μερικοί σημαντικοί όροι, όπως τεκμηριώθηκαν από τον Pfleeger (Pfleeger, 1997), καθορίζονται στον πίνακα 3.

Όρος	Περιγραφή
Έκθεση (exposure)	Πιθανή απώλεια ή βλάβη σε ένα υπολογιστικό σύστημα. Μπορεί να είναι απώλεια ή βλάβη σε δεδομένα ή απώλεια χρόνου και προσπάθειας αν η ανάκαμψη απαιτείται μετά από ένα βρόγχο ασφαλείας
Τρωτότητα (vulnerability)	Μια αδυναμία σε ένα υπολογιστικό σύστημα που μπορεί κάποιος να εκμεταλλευτεί ώστε να προκαλέσει απώλεια ή ζημιά
Επίθεση (attack)	Εκμετάλλευση της τρωτότητας του συστήματος. Γενικά προέρχεται εξωτερικά του συστήματος και είναι σκόπιμη προσπάθεια να προκαλέσει κάποια ζημιά
Απειλές (threats)	Καταστάσεις που έχουν την δυναμική να προκαλέσουν απώλεια ή ζημιά. Μπορείτε να τις σκεφτείτε σαν τρωτότητα του συστήματος που υπόκειται σε επίθεση.
Έλεγχος (control)	Ένα μέτρο προστασίας που μειώνει την τρωτότητα του συστήματος. Η κρυπτογράφηση είναι ένα παράδειγμα μείωσης τρωτότητας σε ένα σύστημα ασθενούς ελέγχου προσπέλασης.

Πίνακας 3 [Sommerville, 2008, Software Engineering 8th edition]

Υπάρχει μια σαφής αναλογία εδώ με μερικούς από τους όρους της ασφάλειας έτσι ώστε μια έκθεση είναι ανάλογη με ένα ατύχημα και μια ευπάθεια είναι ανάλογη με έναν κίνδυνο.

Επομένως, υπάρχουν συγκρίσιμες προσεγγίσεις που μπορούν να χρησιμοποιηθούν για να βεβαιώσουν την ασφάλεια ενός συστήματος:

1. *Η αποφυγή ευπάθειας* Το σύστημα σχεδιάζεται έτσι ώστε οι ευπάθειες δεν εμφανίζονται. Παραδείγματος χάριν, εάν ένα σύστημα δεν συνδέεται με ένα εξωτερικό δημόσιο δίκτυο τότε δεν υπάρχει καμία δυνατότητα μιας επίθεσης από τα μέλη του κοινού.

2. *Η ανίχνευση και η ουδετεροποίηση επίθεσης* Το σύστημα έχει σχεδιαστεί με σκοπό να ανιχνεύσει τα τρωτά σημεία και να τα αφαιρέσουν προτού να οδηγηθεί σε μια έκθεση. Ένα παράδειγμα της ανίχνευσης και της αφαίρεσης τρωτότητας είναι η χρήση ενός ελεγκτή ιών που αναλύει τα εισερχόμενα αρχεία για τους ιούς και τροποποιεί αυτά τα αρχεία για να αφαιρέσει τον ιό.

2. Διαδικασίες Λογισμικού

2.1 Γενικά

Διαδικασία λογισμικού είναι μια σειρά ενεργειών που οδηγούν στην παραγωγή ενός προϊόντος λογισμικού. Αυτές οι ενέργειες μπορεί να εμπεριέχουν την ανάπτυξη ενός λογισμικού από την αρχή σε μια τυπική γλώσσα προγραμματισμού, όπως η JAVA ή η C. Όμως , ολοένα και περισσότερο, νέα λογισμικά αναπτύσσονται επεκτείνοντας και τροποποιώντας ήδη υπάρχοντα καθώς και σχηματίζοντας και ολοκληρώνοντας έτοιμα και άμεσα διαθέσιμα λογισμικά ή συστατικά συστημάτων.

Οι διαδικασίες λογισμικού είναι πολύπλοκες και όπως όλες οι διανοητικές και δημιουργικές διαδικασίες , βασίζονται σε αποφάσεις και κρίσεις που λαμβάνονται από ανθρώπους. Εξαιτίας της ανάγκης για αποφάσεις και δημιουργικότητα, οι προσπάθειες για αυτοματοποίηση των διαδικασιών ανάπτυξης λογισμικών δεν έχουν μεγάλη επιτυχία. Τα εργαλεία σχεδίασης λογισμικού με χρήση υπολογιστών (computer-aided software engineering tools/CASE tools) μπορούν να υποστηρίξουν ορισμένες διαδικασίες. Βέβαια, δεν υπάρχει πιθανότητα, τουλάχιστον στα επόμενα χρόνια, να υπάρξει πιο εκτενής αυτοματοποίηση όπου λογισμικό να καταλαμβάνει δημιουργική σχεδίαση από τους σχεδιαστές που εμπλέκονται στην διαδικασία λογισμικού.

Ένας λόγος για τον οποίο η αποτελεσματικότητα των CASE tools είναι περιορισμένη είναι η τεράστια ποικιλία των διαδικασιών λογισμικού. Δεν υπάρχει ιδανική διαδικασία , και πολλοί οργανισμοί έχουν αναπτύξει την δικιά τους προσέγγιση για την ανάπτυξη λογισμικού. Οι διαδικασίες έχουν αναπτυχθεί για να εκμεταλλεύονται τις ικανότητες των ανθρώπων σε

μια επιχείρηση και των συγκεκριμένων χαρακτηριστικών των συστημάτων που αναπτύσσονται. Για ορισμένα συστήματα όπως τα κρίσιμα συστήματα απαιτείται μια καλά δομημένη διαδικασία ανάπτυξης λογισμικού. Για επιχειρηματικά συστήματα, μια διαδικασία που να αλλάζει ραγδαία τις απαιτήσεις, μια ευέλικτη, ευκίνητη είναι πιο αποτελεσματική.

2.2 Δραστηριότητες διαδικασίας λογισμικού

Αν και υπάρχουν πολλές διαδικασίες λογισμικού , ορισμένες θεμελιώδεις δραστηριότητες είναι κοινές σε όλες αυτές:

Καθορισμός απαιτήσεων, όπου η λειτουργικότητα και οι περιορισμοί του λογισμικού πρέπει να καθοριστούν.

Σχεδιασμός και ολοκλήρωση λογισμικού, όπου πρέπει να παραχθεί το λογισμικό που καλύπτει τις απαιτήσεις.

Έλεγχος αξιοπιστίας λογισμικού, όπου ελέγχεται αν κάνει αυτό που απαιτεί ο πελάτης

Εξέλιξη λογισμικού, όπου το λογισμικό πρέπει να εξελιχθεί ώστε να καλύψει τις αλλαγές στις απαιτήσεις του πελάτη.

Αν και δεν υπάρχει ιδανική διαδικασία λογισμικού, υπάρχει ο στόχος της βελτίωσης των διαδικασιών σε πολλούς οργανισμούς. Οι διαδικασίες μπορεί να περιλαμβάνουν ξεπερασμένες τεχνικές ή να μην εκμεταλλεύονται την βέλτιστη εφαρμογή στην βιομηχανική σχεδίαση λογισμικού. Στην πραγματικότητα πολλοί οργανισμοί ακόμη δεν εκμεταλλεύονται τις μεθόδους ανάπτυξης λογισμικού στην ανάπτυξη του λογισμικού τους.

Οι διαδικασίες λογισμικού μπορούν να βελτιωθούν μέσω της κανονικοποίησης των διαδικασιών όπου η ποικιλία των διαδικασιών λογισμικού μέσα σε ένα οργανισμό περιορίζεται. Αυτό οδηγεί στην βελτίωση της επικοινωνίας και σε μείωση του χρόνου εκπαίδευσης και καθιστά την αυτοματοποιημένη διαδικασία πιο οικονομική. Η

κανονικοποίηση είναι επίσης ένα σημαντικό πρώτο βήμα στην εισαγωγή νέων μεθόδων σχεδιασμού λογισμικού και εφαρμογών καλής σχεδίασης λογισμικού.

2.3 Μοντέλα ανάπτυξης λογισμικού

Το μοντέλο ανάπτυξης λογισμικού είναι μια θεωρητική αναπαράσταση μιας διαδικασίας λογισμικού. Κάθε μοντέλο διαδικασίας αντιπροσωπεύει μια διαδικασία από μια συγκεκριμένη προοπτική παρέχοντας έτσι μόνο μερική πληροφόρηση για την διαδικασία. Παρακάτω γίνεται εισαγωγή των πολύ γενικών μοντέλων διαδικασιών (παραδειγμάτων διαδικασιών) και παρουσιάζονται από αρχιτεκτονικής πλευράς. Αυτό αποτελεί το πλαίσιο εργασίας για την ανάπτυξη του λογισμικού αλλά όχι οι λεπτομέρειες συγκεκριμένων δραστηριοτήτων.

Αυτά τα γενικά μοντέλα δεν αποτελούν οριστικές και απόλυτες περιγραφές διαδικασιών λογισμικού. Μάλλον είναι θεωρίες της διαδικασίας που χρησιμοποιείται για να εξηγηθούν διαφορετικές προσεγγίσεις στην ανάπτυξη λογισμικού. Μπορούμε να τις θεωρήσουμε ως πλαίσια εργασίας που μπορούν να επεκταθούν και να υιοθετηθούν ώστε να δημιουργηθεί πιο συγκεκριμένη και λεπτομερής διαδικασία σχεδιασμού λογισμικού.

Τα μοντέλα διαδικασιών που καλύπτονται παρακάτω είναι:

Το μοντέλο καταρράκτη (waterfall model) Αυτό παίρνει τις θεμελιώδεις διαδικασίες των προδιαγραφών, της ανάπτυξης , της επαλήθευσης και της εξέλιξης και τις αναπαριστά ως ξεχωριστές φάσεις όπως ο προσδιορισμός των απαιτήσεων, ο σχεδιασμός του λογισμικού, η ολοκλήρωση του συστήματος, ο έλεγχος, κτλ.

Η εξελικτική ανάπτυξη (evolutionary development) Αυτή η προσέγγιση παρεμβάλει - διαστρωματώνει θα λέγαμε- τις δραστηριότητες των προδιαγραφών, της ανάπτυξης και της επαλήθευσης. Ένα αρχικό σύστημα ραγδαία εξελίσσεται από αόριστες προδιαγραφές. Αυτό έπειτα βελτιώνεται με την «είσοδο» των πελατών.

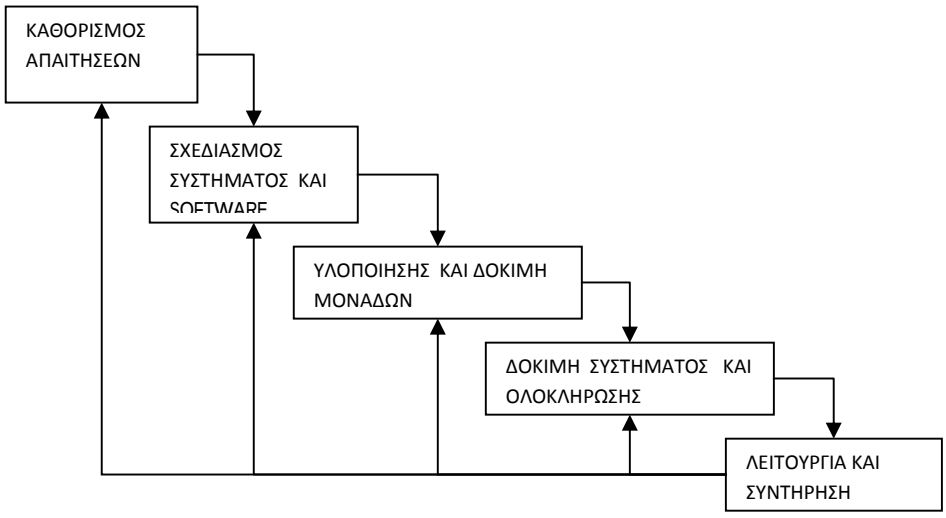
Ο σχεδιασμός λογισμικού που βασίζεται στα συστατικά του (component-based software engineering) Αυτή η προσέγγιση βασίζεται στην ύπαρξη ενός σημαντικού αριθμού επαναχρησιμοποιήσιμων συστατικών. Η διαδικασία ανάπτυξης του συστήματος εστιάζει στην ενοποίηση αυτών των συστατικών σε ένα σύστημα από το να τα αναπτύξουμε από την αρχή.

Αυτά τα τρία γενικά μοντέλα χρησιμοποιούνται ευρέως στον σύγχρονο σχεδιασμό λογισμικού στην πράξη. Δεν είναι αμοιβαία αποκλειόμενα και συχνά χρησιμοποιούνται μαζί, ειδικά για την ανάπτυξη μεγάλων συστημάτων. Υποσυστήματα μέσα σε ένα ευρύτερο σύστημα μπορούν να αναπτυχθούν χρησιμοποιώντας διαφορετικές προσεγγίσεις. Συνεπώς αν και είναι άβολο να εξετάζουμε τα δυο συστήματα ξεχωριστά, οφείλουμε να κατανοήσουμε ότι στην πράξη συχνά συνδυάζονται.

Όλα τα είδη αυτών των γενικών μοντέλων προτείνονται συχνά και μπορούν να χρησιμοποιηθούν από ορισμένους οργανισμούς. Το πιο σημαντικό είδος είναι πιθανώς η ανάπτυξη του επισήμου συστήματος, όπου ένα τυπικό μαθηματικό μοντέλο δημιουργείται. Αυτό το μοντέλο έπειτα μετασχηματίζεται χρησιμοποιώντας μαθηματικούς μετασχηματισμούς που διατηρούν την ακεραιότητά του σε εκτελέσιμο κώδικα.

2.3.1 Το μοντέλο καταρράκτη (waterfall model)

Πρόκειται για το πρώτο μοντέλο ανάπτυξης λογισμικού που δημοσιεύτηκε και πηγάζει από τα γενικότερες διαδικασίες σχεδιασμού συστημάτων. Κάτι τέτοιο φαίνεται στα παρακάτω σχήματα (Σχήμα 1 & 2). Εξαιτίας της διαδοχικότητας από την μια φάση στην άλλη αυτό το μοντέλο ονομάστηκε μοντέλο καταρράκτη ή μοντέλο κύκλου ζωής λογισμικού.



ΣΧΗΜΑ 1. Ο κύκλος ζωής του λογισμικού (μοντέλο καταρράκτη)

Τα κύρια επίπεδα του μοντέλου καθορίζουν και θεμελιώδεις δραστηριότητες της ανάπτυξης λογισμικού:

Καθορισμός και ανάλυση απαιτήσεων Οι υπηρεσίες του συστήματος, οι περιορισμοί και οι στόχοι καθορίζονται πάντα με γνώμονα τους χρήστες του συστήματος. Έπειτα καθορίζονται λεπτομερώς και θεωρούνται ως προδιαγραφές του συστήματος.

Σχεδιασμός συστήματος και λογισμικού Η διαδικασία σχεδιασμού του συστήματος διαχωρίζει τις απαιτήσεις είτε ως hardware είτε ως software του συστήματος. Εγκαθιστά μια

αρχιτεκτονική ενός ενοποιημένου συστήματος. Ο σχεδιασμός λογισμικού εμπεριέχει την ταυτοποίηση, προσδιορισμό και περιγραφή θεμελιωδών εννοιών λογισμικού και των συσχετίσεών τους.

Υλοποίηση και δοκιμή μονάδων Σε αυτό το στάδιο, ο σχεδιασμός συστήματος υλοποιείται ως ένα σύνολο προγραμμάτων ή μονάδων προγραμμάτων. Η δοκιμή μονάδων συστήματος περιέχει επαλήθευση ότι κάθε μονάδα ταιριάζει με τις προδιαγραφές.

Ολοκλήρωση και δοκιμή συστήματος Η κάθε μονάδα συστήματος ή πρόγραμμα ολοκληρώνεται και ελέγχεται ως ένα ολοκληρωμένο σύστημα ώστε να διασφαλιστεί ότι οι απαιτήσεις του συστήματος ικανοποιούνται. Μετά τον έλεγχο το σύστημα λογισμικού δίνεται στον πελάτη.

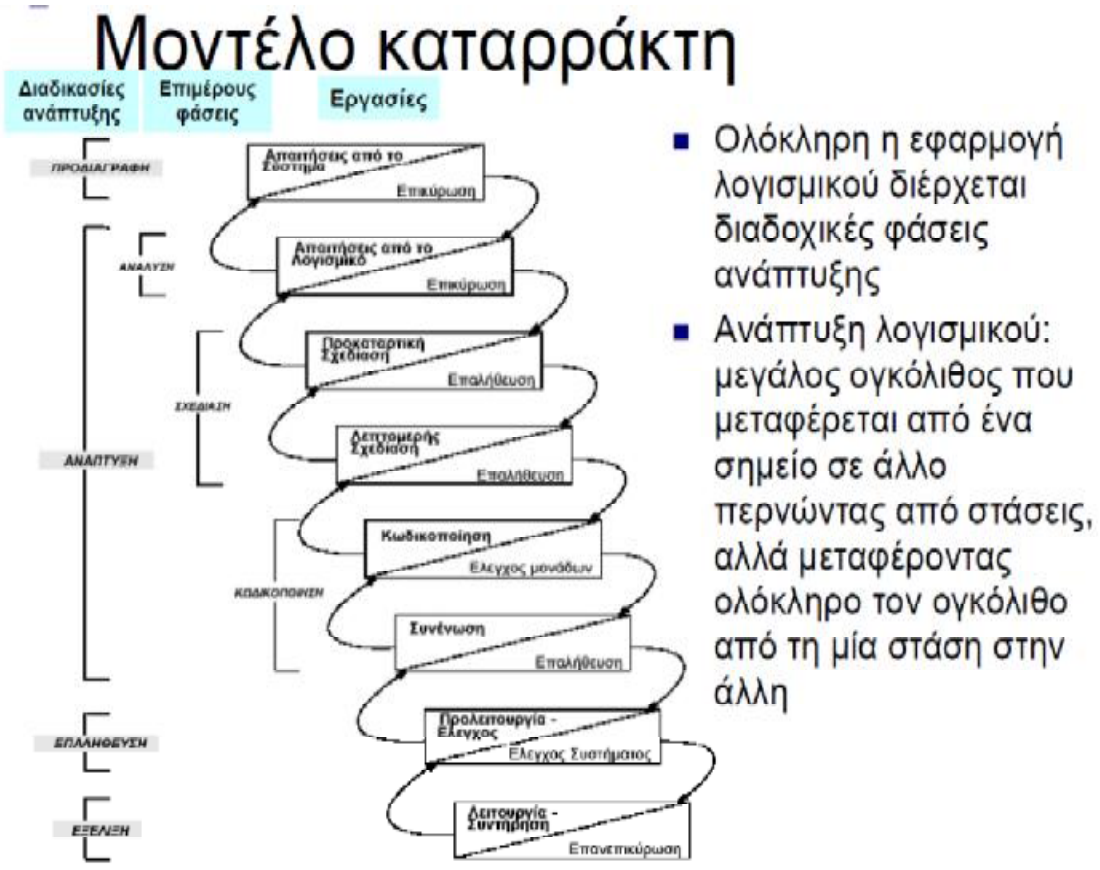
Λειτουργία και συντήρηση Κανονικά, (αν και όχι απαραίτητα) αυτή είναι η πιο μακρόχρονη φάση του κύκλου ζωής. Το σύστημα εγκαθίσταται και τίθεται σε λειτουργία στην πράξη. Η συντήρηση εμπεριέχει διόρθωση λαθών που ανακαλύπτονται στα πρώιμα στάδια του κύκλου ζωής, βελτιώσεις ολοκλήρωσης των μονάδων του συστήματος και εμπλουτισμούς των υπηρεσιών του συστήματος, καθώς ανακαλύπτονται νέες απαιτήσεις.

Κατ' αρχήν , το αποτέλεσμα της κάθε φάσης είναι ένα ή περισσότερα έγγραφα εγκρίνονται (υπογράφονται). Η επόμενη φάση δεν ξεκινά αν δεν τελειώσει η προηγούμενη. Στην πράξη αυτές οι φάσεις αλληλεπικαλύπτονται και η μια τροφοδοτεί την άλλη πληροφορίες. Κατά την διάρκεια της σχεδίασης, προσδιορίζονται προβλήματα σχετικά με τις απαιτήσεις. Κατά την κωδικοποίηση τα προβλήματα σχεδίασης ανακαλύπτονται κ.ο.κ. Η διαδικασία λογισμικού δεν είναι ένα απλό γραμμικό μοντέλο αλλά περιέχει μια σειρά επαναλήψεων των δραστηριοτήτων σχεδίασης.

Εξαιτίας του κόστους παραγωγής και έγκρισης εγγράφων, οι επαναλήψεις είναι δαπανηρές και περιέχουν σημαντική αναθεώρηση. Συνεπώς , έπειτα από ένα μικρό αριθμό επαναλήψεων, είναι δόκιμο να «παγώνουμε» τμήματα της ανάπτυξης, όπως οι απαιτήσεις, ώστε να ασχολούμαστε με τα τελευταία στάδια ανάπτυξης. Τυχόν προβλήματα αφήνονται για μετέπειτα ανάλυση, αγνοούνται ή ξεπερνιούνται. Αυτό το πρώιμο «πάγωμα» των απαιτήσεων, μπορεί να σημαίνει ότι το σύστημα δεν θα κάνει αυτό που προσδοκά ο χρήστης.

Μπορεί ακόμη να οδηγήσει σε άσχημα δομημένα συστήματα, καθώς τα προβλήματα σχεδίασης υπερπηδούνται με χρήση τεχνασμάτων ολοκλήρωσης.

Κατά τη διάρκεια του τελευταίου σταδίου του κύκλου ζωής (λειτουργία και συντήρηση) το λογισμικό τίθεται σε χρήση. Λάθη και παραλείψεις στις αρχικές απαιτήσεις του συστήματος ανακαλύπτονται. Σφάλματα προγράμματος και σχεδιασμού επείγουν και αναγνωρίζεται η ανάγκη για νέα λειτουργικότητα. Το σύστημα πρέπει συνεπώς να εξελιχθεί ώστε να παραμείνει χρήσιμο. Κάνοντας αυτές τις αλλαγές (συντήρηση λογισμικού) είναι δυνατόν να απαιτηθεί επανάληψη προηγούμενων σταδίων της διαδικασίας.



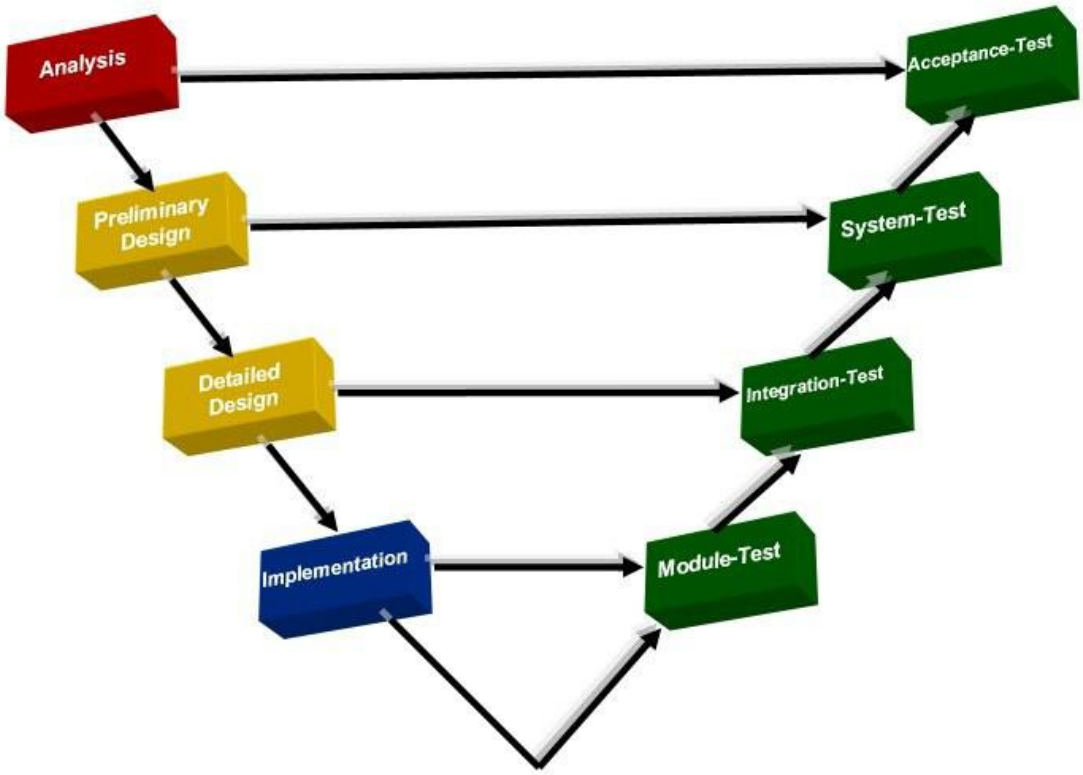
ΣΧΗΜΑ 2. Ο κύκλος ζωής του λογισμικού (μοντέλο καταρράκτη)

Τα πλεονεκτήματα του μοντέλου του καταρράκτη είναι τα έγγραφα που παράγονται σε κάθε φάση και ότι ταιριάζει με τα άλλα μοντέλα σχεδιασμού. Το βασικό του μειονέκτημα είναι ο δύσκαμπτος διαμελισμός του έργου σε ξεχωριστά τμήματα. Οι υποχρεώσεις πρέπει να πραγματοποιούνται σε κάποιο πρώιμο στάδιο της διαδικασίας, πράγμα που το κάνει δύσκολο να ανταποκρίνεται στις απαιτήσεις του πελάτη.

Συνεπώς, το μοντέλο καταρράκτη πρέπει μόνο να χρησιμοποιείται όταν οι απαιτήσεις είναι καλά κατανοητές και απίθανο να αλλάξουν ριζικά κατά την διάρκεια ανάπτυξης του συστήματος. Όμως το μοντέλο καταρράκτη αντικατοπτρίζει τον τύπο των μοντέλων διαδικασιών που χρησιμοποιούνται σε άλλα έργα σχεδίασης. Ως αποτέλεσμα, οι διαδικασίες λογισμικού που βασίζονται σε αυτήν την προσέγγιση ακόμη χρησιμοποιούνται για ανάπτυξη λογισμικού, ειδικά όταν το έργο είναι τμήμα ενός ευρύτερου έργου σχεδιασμού συστήματος.

2.3.2 Το μοντέλο V

Το μοντέλο V (Σχήμα 3) είναι μια παραλλαγή του γραμμικού μοντέλου μοντέλου καταρράκτη, η οποία αναδεικνύει τον τρόπο με τον οποίο οι δραστηριότητες των δοκιμών σχετίζονται με την ανάλυση και τη σχεδίαση. Η κωδικοποίηση παριστάνεται στην κορυφή ενός σχήματος V, με την ανάλυση και τη σχεδίαση στο αριστερό σκέλος V και τις δοκιμές και τη συντήρηση στο δεξιό.



Σχήμα 3: Το μοντέλο V

Το μοντέλο V προτείνει τη χρήση των **δοκιμών δομικών μονάδων** και ενοποίησης και για την επαλήθευση του σχεδίου των προγραμμάτων. Με άλλα λόγια, κατά τη διάρκεια αυτών των δοκιμών, τα μέλη των ομάδων κωδικοποίησης και ελέγχου θα πρέπει να εξασφαλίζουν ότι έχουν υλοποιηθεί σωστά στον κώδικα όλες οι πτυχές της σχεδίασης των προγραμμάτων. Οι δοκιμές του συστήματος θα πρέπει να επαληθεύουν το σχέδιο του συστήματος, εξασφαλίζοντας ότι όλες οι πτυχές αυτού του σχεδίου έχουν υλοποιηθεί σωστά.

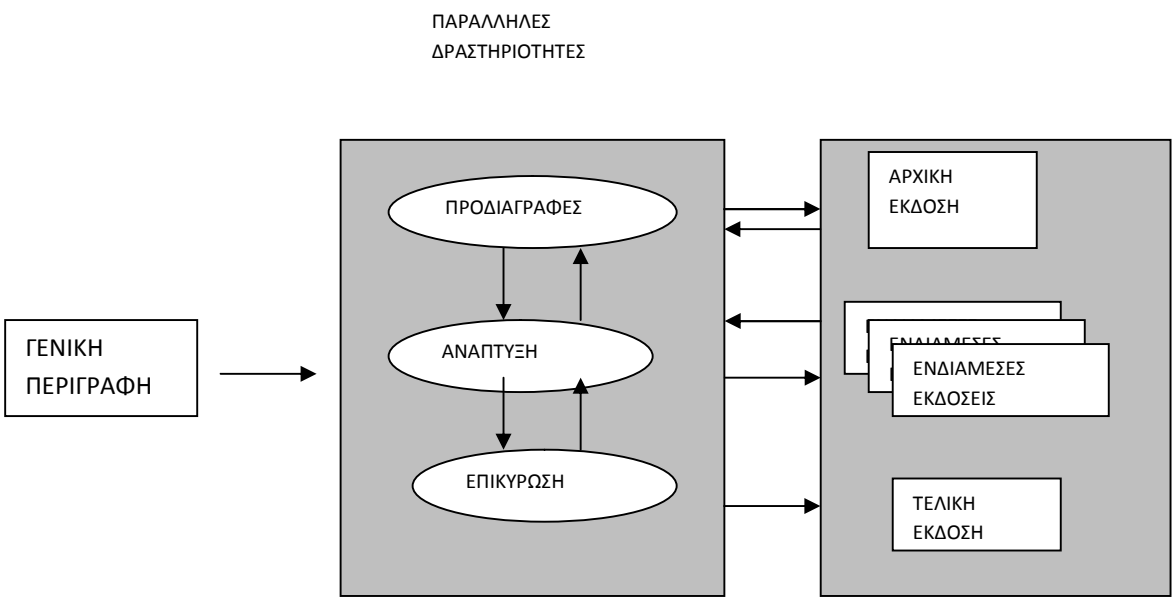
Οι έλεγχοι αποδοχής, οι οποίες γίνονται από τον πελάτη και όχι από τους δημιουργούς, επικυρώνουν τις προδιαγραφές συσχετίζοντας κάθε βήμα των δοκιμών με το αντίστοιχο στοιχείο των προδιαγραφών. Αυτού του είδους οι δοκιμές ελέγχουν αν έχει υλοποιηθεί πλήρως όλες οι προδιαγραφές πριν γίνει αποδεκτό το σύστημα και πληρωθούν οι δημιουργοί.

Η σύνδεση που επιχειρείται σε αυτό το μοντέλο μεταξύ του αριστερού και του δεξιού σκέλους του V υπονοεί ότι, αν εντοπιστούν προβλήματα κατά τη διάρκεια της επαλήθευσης και της επικύρωσης, το αριστερό σκέλος του V θα μπορεί να εκτελεστεί ξανά προκειμένου να διορθωθούν και να βελτιωθούν οι προδιαγραφές, το σχέδιο και ο κώδικας πριν ξεκινήσουν και πάλι τα βήματα των δοκιμών του δεξιού σκέλους του V.

Με το μοντέλο V ορίζονται με κατηγορηματικότερο τρόπο ορισμένες επαναληπτικές και αναθεωρητικές εργασίες οι οποίες κρύβονται στην αναπαράσταση του γραμμικού μοντέλου. Ενώ το γραμμικό μοντέλο εστιάζει συχνά στα έγγραφα και τα παραγόμενα προϊόντα, το μοντέλο V εστιάζει στις δραστηριότητες και την ορθότητά τους.

2.3.3 Το μοντέλο της εξελικτικής ανάπτυξης

Η εξελικτική ανάπτυξη λογισμικού βασίζεται στην ιδέα της ανάπτυξης μιας αρχικής προσέγγισης της εφαρμογής, που σχολιάζεται και διορθώνεται από τον χρήστη περνώντας από διάφορες εκδόσεις μέχρι να αναπτυχθεί ένα επαρκές και πλήρες σύστημα (Σχήμα 4).



ΣΧΗΜΑ 4. Εξελικτική ανάπτυξη

Οι απαιτήσεις , η ανάπτυξη και η επαλήθευση ως λειτουργίες δεν είναι πλέον ξεχωριστές αλλά εναλλάσσονται, με άμεση ανατροφοδότηση κατά την διάρκεια των δραστηριοτήτων.

Υπάρχουν δύο θεμελιώδη είδη εξελικτικής ανάπτυξης:

1.Διερευνητική ανάπτυξη, όπου σκοπός της διεργασίας είναι να εργαστεί με τον πελάτη ώστε να εξερευνήσει τις απαιτήσεις του και να παραδώσει ένα τελικό σύστημα. Η ανάπτυξη ξεκινά από μέρη του συστήματος που είναι κατανοητά και στην πορεία εμπλουτίζεται από χαρακτηριστικά που προτείνονται από τον πελάτη.

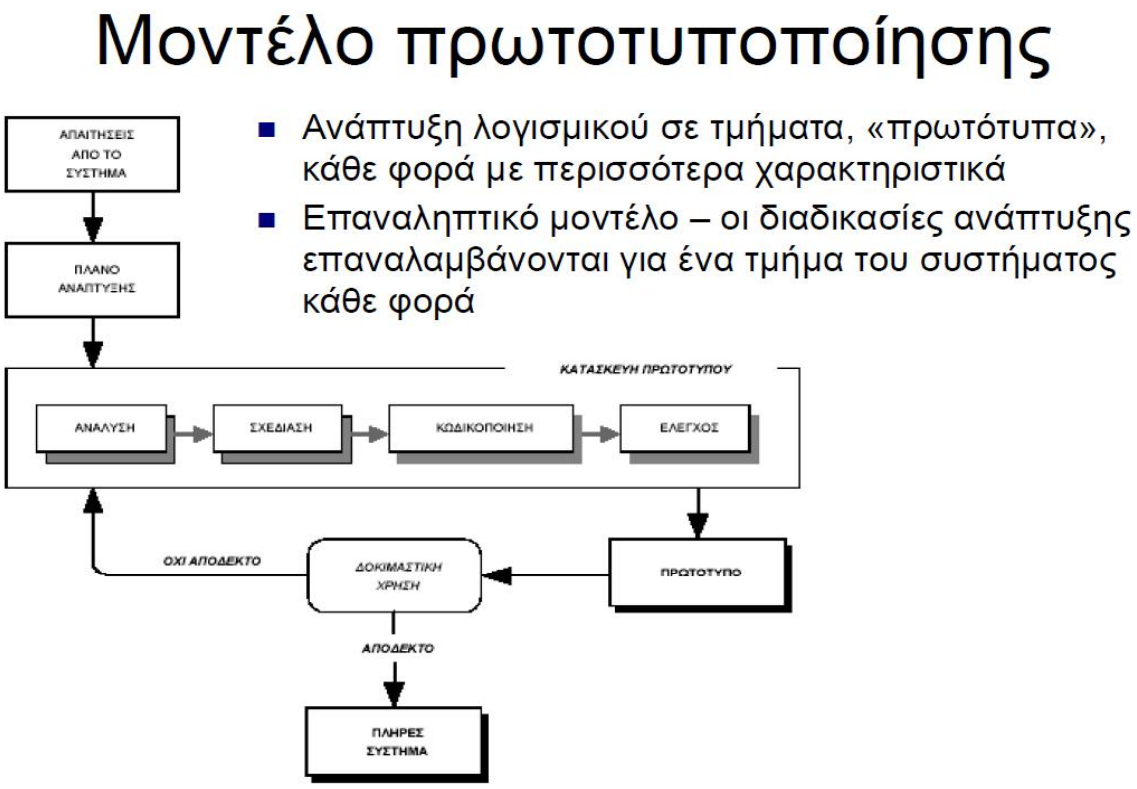
2. Σταδιακή Διαμόρφωση πρωτοτύπου ή πρωτοτυποποίησης (Σχήμα 5) , όπου ο σκοπός της εξελικτικής διαδικασίας ανάπτυξης είναι να κατανοήσει τις απαιτήσεις του πελάτη και έτσι να αναπτύξει καλύτερο προσδιορισμό απαιτήσεων για το σύστημα. Το πρωτότυπο

επικεντρώνεται στον πειραματισμό με τις απαιτήσεις του πελάτη που είναι ελάχιστα κατανοητές.

Πολλές φορές η εξελικτική ανάπτυξη είναι πιο αποτελεσματική από το μοντέλο καταρράκτη στην παραγωγή μοντέλων που ικανοποιούν τις άμεσες ανάγκες των πελατών. Το πλεονέκτημα μιας διαδικασίας ανάπτυξης λογισμικού βασισμένης σε εξελικτική προσέγγιση είναι ότι οι απαιτήσεις μπορούν να αναπτυχθούν αυξητικά. Το γεγονός ότι οι χρήστες κατανοούν καλύτερα το πρόβλημα τους, αντανακλάται και στο σύστημα. Βέβαια από πλευράς σχεδιασμού και διαχείρισης, η εξελικτική ανάπτυξη έχει δυο προβλήματα:

1. Η διαδικασία δεν φαίνεται. Οι μάνατζερ χρειάζονται συγκεκριμένα παραδοτέα ώστε να μετρήσουν την πρόοδο. Αν τα συστήματα αναπτύσσονται γρήγορα, δεν είναι οικονομικό το να παράγεις συνέχεια έγγραφα που αντανακλούν την κάθε μορφή του συστήματος.

2. Τα συστήματα είναι συχνά φτωχά δομημένα. Οι συνεχείς αλλαγές τείνουν να διαβρώνουν την δομή του λογισμικού. Ενσωματωμένες αλλαγές στο λογισμικό γίνονται υπερβολικά δύσκολες και κοστίζουν.



ΣΧΗΜΑ 5. Μοντέλο Πρωτοτυποποίησης

Για μικρού και μεσαίου μεγέθους συστήματα (έως και 500.000 γραμμές κώδικα), η εξελικτική προσέγγιση είναι η καλύτερη προσέγγιση ανάπτυξης. Τα προβλήματα της εξελικτικής ανάπτυξης είναι οξεία σε μεγάλα, πολύπλοκα, μεγάλης διάρκειας ζωής συστήματα, όπου διαφορετικές ομάδες διαμορφώνουν διαφορετικά τμήματα του συστήματος. Είναι δύσκολο να θεσπίσεις μια σταθερή αρχιτεκτονική του συστήματος χρησιμοποιώντας αυτή την προσέγγιση, πράγμα που σημαίνει ότι είναι δύσκολο να ενσωματωθούν οι συνεισφορές των ομάδων.

Για μεγάλα συστήματα, προτείνεται μια μικτή διαδικασία που εμπεριέχει τα καλύτερα χαρακτηριστικά του μοντέλου καταρράκτη και της εξελικτικής προσέγγισης. Αυτό μπορεί να περιέχει την ανάπτυξη ενός πρωτοτύπου χρησιμοποιώντας την εξελικτική ανάπτυξη ώστε να

επιλυθούν οι αβεβαιότητες στις απαιτήσεις του συστήματος. Μπορείτε να επαναπροσδιορίσετε το σύστημα χρησιμοποιώντας μια πιο καλά δομημένη προσέγγιση. Μέρη του συστήματος που είναι κατανοητά μπορούν να προσεγγιστούν και να αναπτυχθούν με την βοήθεια μιας βασισμένης στο μοντέλο καταρράκτη διαδικασίας. Άλλα τμήματα του συστήματος όπως οι διεπαφές χρηστών , που είναι δύσκολο να προσδιοριστούν εκ των προτέρων, πρέπει πάντα να αναπτύσσονται με χρήση της διερευνητικής προσέγγισης .

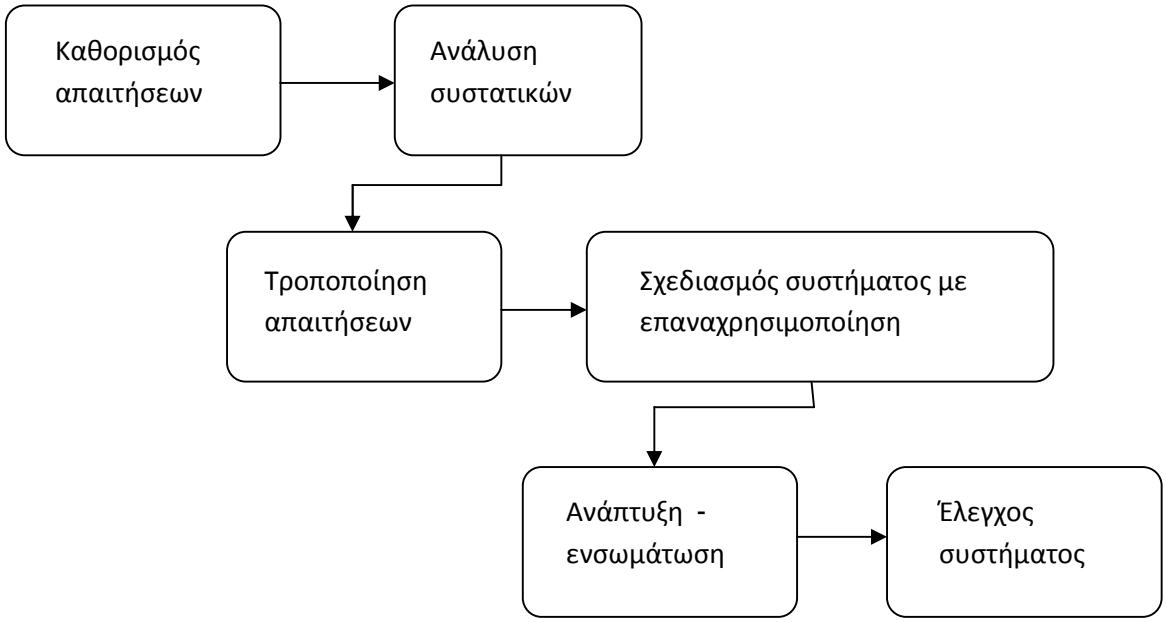
2.3.4 Το μοντέλο ανάπτυξης λογισμικού βασισμένο στα συστατικά

Στην πλειοψηφία των έργων λογισμικού υπάρχει επαναχρησιμοποίηση λογισμικού. Αυτό συνήθως συμβαίνει συνήθως όταν οι άνθρωποι που εργάζονται στο έργο γνωρίζουν από τεχνικές ή κώδικα που είναι εφάμιλλες των απαιτουμένων. Τον αναζητούν, τροποποιούν όπως απαιτείται και τον ενσωματώνουν στο σύστημά τους.

Στην εξελικτική μέθοδο η επαναχρησιμοποίηση είναι συχνά απαραίτητη για άμεση ανάπτυξη του συστήματος.

Αυτή η ανεπίσημη επαναχρησιμοποίηση συμβαίνει ανεξαρτήτως του μοντέλου που χρησιμοποιείται. Βέβαια, τα τελευταία χρόνια, μια προσέγγιση που βασίζεται στην επαναχρησιμοποίηση και καλείται ανάπτυξη λογισμικού βασισμένη στα στοιχεία, είναι ευρέως διαδεδομένη.

Η βασισμένη στην επαναχρησιμοποίηση προσέγγιση στηρίζεται σε μια μεγάλη βάση λογισμικών στοιχείων που δύνανται να επαναχρησιμοποιηθούν καθώς και κάποια ενσωματωμένα πλαίσια για αυτά τα συστατικά στοιχεία. Σε αρκετές περιπτώσεις αυτά τα στοιχεία είναι ξεχωριστά συστήματα που μπορούν να παρέχουν συγκεκριμένες λειτουργίες όπως για παράδειγμα μορφοποίηση κειμένου ή αριθμητικός υπολογισμός . Η γενική μορφή της διαδικασίας φαίνεται παρακάτω (Σχήμα 6).



ΣΧΗΜΑ 6. Ανάπτυξη μοντέλου βασισμένη στα συστατικά στοιχεία

Παρατηρούμε ότι τα στάδια του προσδιορισμού των απαιτήσεων και του ελέγχου συστήματος είναι ίδια και στα άλλα μοντέλα, ενώ τα ενδιάμεσα στάδια είναι διαφορετικά. Τα στάδια αυτά είναι:

Ανάλυση συστατικών στοιχείων. Με δεδομένο τον καθορισμό των απαιτήσεων, γίνεται μια έρευνα για στοιχεία που να καλύπτουν αυτές τις απαιτήσεις. Συνήθως δεν υπάρχει πλήρης ταύτιση και τα στοιχεία που χρησιμοποιούνται παρέχουν μόνο μερική από την λειτουργικότητα που απαιτείται.

Τροποποίηση απαιτήσεων. Σε αυτό το στάδιο, οι απαιτήσεις αναλύονται χρησιμοποιώντας πληροφορίες από τα στοιχεία που έχουν ανακαλυφθεί. Έπειτα τροποποιούνται ώστε να αντικατοπτρίζουν τα διαθέσιμα συστατικά στοιχεία. Οπουδήποτε οι τροποποιήσεις είναι αδύνατες, η δραστηριότητα ανάλυσης των στοιχείων επαναπροσδιορίζεται ώστε να αναζητηθούν νέες λύσεις.

Σχεδιασμός συστήματος με επαναχρησιμοποίηση. Σε αυτή τη φάση, το γενικό πλαίσιο του συστήματος σχεδιάζεται ή επαναχρησιμοποιείται ένα γνωστό ήδη πλαίσιο. Οι σχεδιαστές λαμβάνουν υπόψη τα συστατικά στοιχεία και οργανώνουν κατάλληλα το γενικό πλαίσιο ώστε

να ταιριάξει. Είναι πιθανό να δημιουργηθεί κάποιο νέο λογισμικό εάν ήδη υπάρχοντα στοιχεία δεν είναι δυνατόν να χρησιμοποιηθούν ξανά.

Ανάπτυξη και ενσωμάτωση. Το λογισμικό που δεν είναι δυνατόν να βρεθεί αναπτύσσεται και τα συστατικά στοιχεία και επιμέρους συστήματα ενσωματώνονται ώστε να δημιουργηθεί το νέο σύστημα. Σε αυτό το μοντέλο η ενσωμάτωση του συστήματος μπορεί να είναι και τμήμα της όλης διαδικασίας ανάπτυξης παρά μια ξεχωριστή διαδικασία.

Η κατασκευή λογισμικού με την μέθοδο ανάπτυξης βασισμένη στα συστατικά στοιχεία έχει το προφανές πλεονέκτημα της μείωσης της ποσότητας λογισμικού που αναπτύσσεται και έτσι ελαττώνονται κόστη και κίνδυνοι. Συχνά οδηγεί και σε γρηγορότερη απόδοση του λογισμικού. Όμως , οι παραδοχές και συμβιβασμοί των απαιτήσεων είναι αναπόφευκτοι και αυτό μπορεί να οδηγήσει σε ένα σύστημα που δεν καλύπτει πραγματικά τον χρήστη. Επιπρόσθετα, πιθανός έλεγχος εξέλιξης του συστήματος χάνεται, καθώς οι νέες εκδόσεις των συστατικών στοιχείων που επαναχρησιμοποιούνται δεν είναι υπό τον έλεγχο του οργανισμού που τους χρησιμοποιεί.

Το μοντέλο αυτό μοιάζει περισσότερο με ένα αναδυόμενο μοντέλο που στηρίζεται σε ενσωματωμένες διαδικτυακές λειτουργίες από μια γκάμα παροχέων.

2.4 Διαδικασίες Επανάληψης

Οι αλλαγές είναι αναπόφευκτες στα μεγάλα έργα υλοποίησης λογισμικών. Οι απαιτήσεις του συστήματος αλλάζουν καθώς η επιχείρηση που το συντηρεί αντιδρά σε

εξωτερικές πιέσεις. Οι προτεραιότητες των μανατζερς αλλάζουν διαρκώς. Καθώς νέες τεχνολογίες είναι διαθέσιμες, ο σχεδιασμός και η υλοποίηση αλλάζουν. Αυτό σημαίνει ότι η διαδικασία σχεδιασμού λογισμικού δεν γίνεται «μια και έξω». Αντιθέτως οι επιμέρους δραστηριότητες της διαδικασίας επαναλαμβάνονται συνήθως και το όλο σύστημα ξαναδουλεύεται , όσο οι απαιτήσεις αλλάζουν.

Η επαναληπτική ανάπτυξη είναι θεμελιώδεις για τα λογισμικά. Κρίνουμε σκόπιμο να αναφερθούμε ιδιαίτερα για αυτές στην πτυχιακή μας εργασία. Έτσι στο τρέχον κεφάλαιο περιγράφοντας δύο μοντέλα διαδικασιών, εισάγουμε τις επαναληπτικές διαδικασίες. Αυτά τα μοντέλα σχεδιάστηκαν αποκλειστικά για να υποστηρίξουν την επανάληψη στον σχεδιασμό των λογισμικών:

Το μοντέλο λειτουργικής επαύξησης , όπου οι απαιτήσεις λογισμικού, σχεδιασμός και υλοποίηση επιμερίζονται σε ένα σύνολο τμημάτων που καθένα αναπτύσσεται ξεχωριστά.

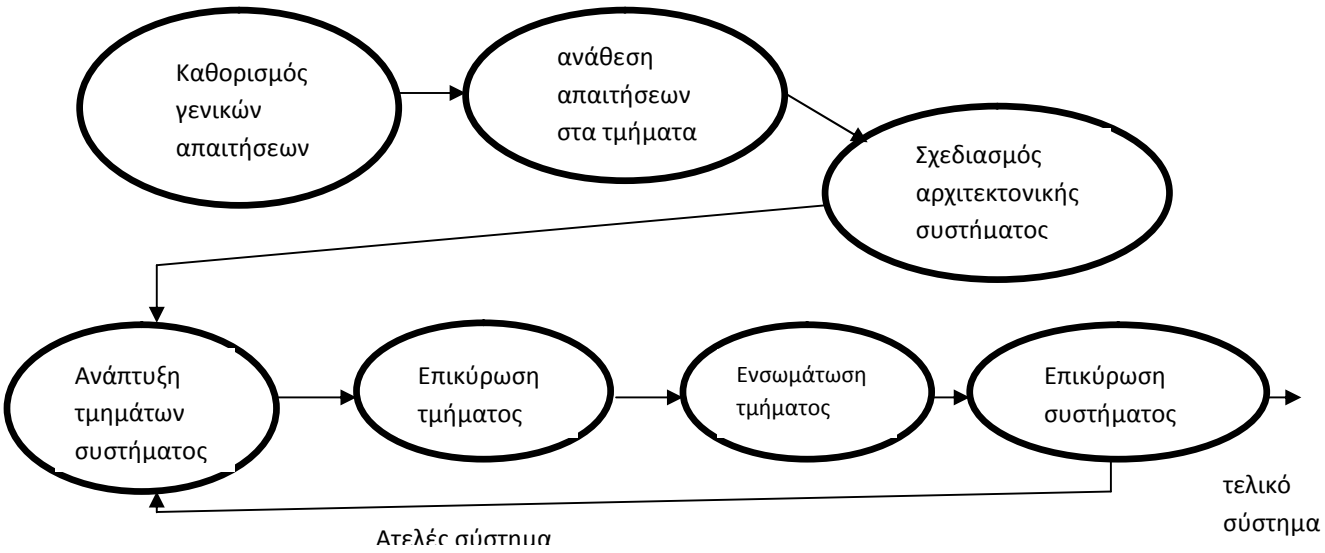
Η σπειροειδής ανάπτυξη, όπου η ανάπτυξη του συστήματος γίνεται προς τα έξω ξεκινώντας από ένα αρχικό προσχέδιο και καταλήγοντας σε ένα τελικό και αναπτυγμένο σύστημα.

Η ουσία των διαδικασιών επανάληψης είναι ότι οι απαιτήσεις αναπτύσσονται σε συνδυασμό με το σύστημα. Όμως αυτό έρχεται σε σύγκρουση με το εφοδιαστικό μοντέλο πολλών επιχειρήσεων όπου οι απαιτήσεις του συνολικού συστήματος είναι μέρος του συμβολαίου ανάπτυξης του συστήματος. Στην αυξητική προσέγγιση, δεν υπάρχουν απαιτήσεις συνολικού συστήματος αν πρώτα δεν ολοκληρωθεί το ίδιο το σύστημα. Αυτό απαιτεί ένα καινούριο είδος συμβολαίου, το οποίο μεγάλης κλίμακας πελάτες όπως κυβερνητικοί οργανισμοί μπορεί να δυσκολευτούν να συμμορφωθούν

2.4.1 Το μοντέλο λειτουργικής επαύξησης

Το μοντέλο καταρράκτη απαιτεί από τους πελάτες για ένα σύστημα να καταλήξουν σε ένα σύνολο απαιτήσεων πριν αρχίσει ο σχεδιασμός και από τον σχεδιαστή να καταλήξει σε συγκεκριμένες στρατηγικές σχεδίασης πριν αρχίσει η υλοποίηση. Πιθανές αλλαγές στις απαιτήσεις απαιτούν ξανά να δουλευτούν οι απαιτήσεις , ο σχεδιασμός και η υλοποίηση. Όμως ο διαχωρισμός σχεδιασμού και υλοποίησης θα έπρεπε να οδηγήσει σε καλά τεκμηριωμένα συστήματα που είναι υποκείμενα σε αλλαγές. Αντιθέτως , μια εξελικτική προσέγγιση ανάπτυξης επιτρέπει στις αποφάσεις απαιτήσεων και σχεδιασμού να καθυστερήσουν αλλά επίσης οδηγεί σε λογισμικό που μπορεί να είναι φτωχά δομημένο, δυσνόητο και δύσκολο να συντηρηθεί.

Το μοντέλο της λειτουργικής επαύξησης (Σχήματα 7 & 8) είναι μια ενδιάμεση προσέγγιση που συνδυάζει τα πλεονεκτήματα των παραπάνω μοντέλων. Σε μια διαδικασία λειτουργικής επαύξησης οι πελάτες αναγνωρίζουν, σε γενικές γραμμές, τις υπηρεσίες που παρέχονται από το σύστημα. Προσδιορίζουν ακόμη ποιες από τις υπηρεσίες είναι οι πιο σημαντικές και ποιες λιγότερο για αυτούς. Κατόπιν καθορίζεται ένας αριθμός επιμέρους τμημάτων, καθένα από τα οποία παρέχει ένα υποσύνολο λειτουργιών του συνολικού συστήματος. Ο καταμερισμός των λειτουργιών σε τμήματα εξαρτάται από την προτεραιότητα που τους έχει δοθεί, με τις πιο σημαντικές λειτουργίες να παραδίδονται πρώτες.



ΣΧΗΜΑ 7. Μοντέλο λειτουργικής επαύξησης

Αμέσως μόλις καθοριστούν τα επιμέρους τμήματα, καθορίζονται οι απαιτήσεις των λειτουργιών που σχετίζονται με το πρώτο τμήμα με λεπτομέρεια και αναπτύσσεται το τμήμα αυτό. Κατά την διάρκεια της ανάπτυξης καθορίζονται οι απαιτήσεις των επόμενων τμημάτων αλλά όμως πιθανές αλλαγές στις απατήσεις του τρέχοντος τμήματος δεν γίνονται αποδεκτές.

Εφόσον κάποιο τμήμα ολοκληρωθεί και παραδοθεί στον πελάτη, αυτός μπορεί να το θέσει σε λειτουργία. Αυτό σημαίνει ότι ο πελάτης μπορεί να εκμεταλλευτεί άμεσα τη λειτουργικότητα του κάθε τμήματος που του παραδίδεται. Έχουν την δυνατότητα να πειραματιστούν με το σύστημα και με αυτόν τον τρόπο να κατανοήσουν τις απαιτήσεις για τα επόμενα τμήματα αλλά και για νεότερες εκδόσεις του τρέχοντος τμήματος. Καθώς νέα τμήματα παραδίδονται, ενσωματώνονται στο σύστημα και έτσι η συνολική λειτουργικότητα του συστήματος διαρκώς βελτιώνεται. Οι κοινές υπηρεσίες είναι δυνατό να ολοκληρωθούν στην αρχή της διαδικασίας ή να υλοποιηθούν τμηματικά καθώς η λειτουργικότητα απαιτείται για ένα τμήμα.

Το μοντέλο της λειτουργικής επαύξησης κατέχει σημαντικό αριθμό πλεονεκτημάτων:

Οι πελάτες δεν είναι αναγκαίο να περιμένουν την ολοκλήρωση του συστήματος για να καρπώνονται τα οφέλη του. Το πρώτο τμήμα ικανοποιεί τις πιο σημαντικές απαιτήσεις τους και έτσι μπορούν να χρησιμοποιήσουν το λογισμικό αμέσως.

Οι πελάτες μπορούν να χρησιμοποιήσουν τα πρώτα τμήματα ως πρωτότυπα και να αποκτήσουν εμπειρία που τους πληροφορεί για τις απαιτήσεις των επόμενων τμημάτων του συστήματος.

Ο κίνδυνος απώλειας ολόκληρου του έργου είναι μικρότερος. Αν και μπορεί να προκύψουν προβλήματα σε μερικά τμήματα, το πιο πιθανό είναι να παραδοθούν με επιτυχία στον πελάτη.

Καθώς τα σημαντικότερης προτεραιότητας τμήματα παραδίδονται πρώτα και τα επόμενα τμήματα ενσωματώνονται με αυτά, αναπόφευκτα, τα πιο σημαντικά τμήματα

επιδέχονται και τον περισσότερο έλεγχο. Αυτό σημαίνει ότι οι πελάτες είναι λιγότερο πιθανό να αντιμετωπίσουν απώλεια ή σφάλματα του λογισμικού στα πιο σημαντικά μέρη του συστήματος.

ΣΧΗΜΑ 8. Μοντέλο λειτουργικής επαύξησης

Όμως υπάρχουν κάποια προβλήματα που προκύπτουν με την χρήση του μοντέλου της λειτουργικής επαύξησης. Τα επιμέρους τμήματα θα πρέπει να είναι σχετικά μικρά (όχι πάνω από 20000 γραμμές κώδικα) και κάθε τμήμα θα πρέπει να παραδίδει και κάποια λειτουργικότητα. Μπορεί να είναι δύσκολο να χαρτογραφηθούν οι απαιτήσεις του πελάτη σε

τμήματα ακριβούς μεγέθους. Επιπρόσθετα, τα περισσότερα συστήματα απαιτούν ένα σύνολο βασικών εγκαταστάσεων που χρησιμοποιούνται από διαφορετικά τμήματα του συστήματος. Καθώς οι απαιτήσεις δεν προσδιορίζονται με ακρίβεια μέχρι να υλοποιηθεί το τμήμα, μπορεί να είναι δύσκολο να προσδιοριστούν οι κοινές εγκαταστάσεις που απαιτούνται από όλα τα τμήματα του συστήματος.

Μια παραλλαγή αυτής της αυξητικής προσέγγισης που έχει αναπτυχθεί καλείται extreme programming (Beck,2000). Αυτή στηρίζεται στην ανάπτυξη και ολοκλήρωση πολύ μικρών τμημάτων λειτουργικοτήτων, στην ανάμιξη του πελάτη στην διαδικασία, σε σταθερή βελτίωση του κώδικα και στον παράλληλο προγραμματισμό.

2.4.2 Το σπειροειδές μοντέλο

Το σπειροειδές μοντέλο της διαδικασίας σχεδιασμού και ανάπτυξης λογισμικού (Σχήμα 9) αρχικά προτάθηκε από τον Boehm (Boehm, 1988). Η διαδικασία του λογισμικού δεν αναπαριστάται πια σαν μια ακολουθία δραστηριοτήτων με μετάβαση από την μια στην άλλη , αλλά σαν ένα σπειροειδές σχήμα.

ΣΧΗΜΑ 9. Σπειροειδές Μοντέλο

Κάθε στροφή στο σπείρωμα αναπαριστά μια φάση της διαδικασίας λογισμικού. Έτσι η πιο εσωτερική στροφή σχετίζεται με την εφικτότητα του συστήματος, η επόμενη με τον καθορισμό των απαιτήσεων, η επόμενη με τον σχεδιασμό του συστήματος κοκ.

Κάθε στροφή του σπειρώματος χωρίζεται σε τέσσερις τομείς:

Ρύθμιση σκοπού, όπου καθορίζονται συγκεκριμένοι στόχοι για αυτή τη φάση του έργου. Καθορίζονται οι περιορισμοί του προϊόντος και της διαδικασίας και σχεδιάζεται ένα

λεπτομερές πλάνο διαχείρισης. Καθορίζονται οι κίνδυνοι του έργου. Εναλλακτικές στρατηγικές που εξαρτώνται από τους κινδύνους μπορούν να καθοριστούν.

Προσδιορισμός κινδύνου και μείωσή του, όπου για κάθε προσδιοριζόμενο κίνδυνο εξάγεται μια λεπτομερής ανάλυση. Λαμβάνονται μέτρα για την μείωση του κινδύνου. Για παράδειγμα, εάν υπάρχει κίνδυνος πως οι απαιτήσεις είναι ακατάλληλες, μπορεί να αναπτυχθεί ένα πρωτότυπο σύστημα.

Ανάπτυξη και επικύρωση, όπου έπειτα από τον υπολογισμό των κινδύνων, επιλέγεται ένα μοντέλο ανάπτυξης για το σύστημα. Για παράδειγμα , εάν δεσπόζουν κίνδυνοι για τις διεπαφές χρηστών (user interface), ένα κατάλληλο μοντέλο ανάπτυξης μπορεί να είναι η εξελικτική προτυποποίηση. Εάν οι κίνδυνοι ασφάλειας είναι επικρατέστεροι, η ανάπτυξη που βασίζεται σε επίσημες μετατροπές μπορεί να είναι η πιο ενδεικτική κοκ. Το μοντέλο του καταρράκτη ενδείκνυται όταν το κυριότερο πρόβλημα είναι η υλοποίηση των υποσυστημάτων.

Προγραμματισμός, όπου το έργο επιθεωρείται και λαμβάνεται η απόφαση εάν θα προχωρήσουμε σε επόμενη στροφή του σπειρώματος. Εάν αποφασιστεί να προχωρήσουμε, σχεδιάζονται πλάνα για την επόμενη φάση του έργου.

Η κυριότερη διαφορά μεταξύ του σπειροειδούς μοντέλου και των άλλων μοντέλων διαδικασιών λογισμικών, είναι η αποκλειστική αναγνώριση του κινδύνου στο σπειροειδές μοντέλο. Ανεπίσημα, ο κίνδυνος σημαίνει απλά ότι κάτι μπορεί να πάει στραβά. Για παράδειγμα, αν υπάρχει πρόθεση να χρησιμοποιηθεί μια νέα γλώσσα προγραμματισμού, ένας κίνδυνος μπορεί να είναι ότι οι διαθέσιμοι μεταγλωττιστές είναι αναξιόπιστοι ή δεν παράγουν ικανή ποσότητα ορθού κώδικα. Οι κίνδυνοι έχουν σαν συνέπειες προβλήματα στο έργο, όπως προγραμματισμός και κόστος συνεπώς η μείωση του κινδύνου αποτελεί σημαντική ενέργεια της διαχείρισης του έργου.

Ένας κύκλος στο σπείρωμα ξεκινά μετά από λεπτομερή επεξεργασία αντικειμένων όπως η λειτουργικότητα και η εκτέλεση. Κατόπιν αριθμούνται οι εναλλακτικοί τρόποι ώστε να ικανοποιούνται αυτοί οι στόχοι και οι περιορισμοί που έχουν τεθεί πάνω σε αυτούς. Κάθε εναλλακτική προσδιορίζεται απέναντι σε κάθε στόχο και αναγνωρίζονται οι πηγές πιθανών κινδύνων. Το επόμενο βήμα είναι να εξαλειφθούν αυτοί οι κίνδυνοι με την βοήθεια δραστηριοτήτων συλλογής πληροφοριών ‘όπως για παράδειγμα μια λεπτομερής ανάλυση κινδύνων, προτυποποίηση και εξομοίωση. Από τη στιγμή που οι κίνδυνοι έχουν προσδιοριστεί , εξελίσσεται και η ανάπτυξη, ακολουθούμενη από ενέργειες σχεδιασμού για την επόμενη φάση της διαδικασίας.

2.5 Δραστηριότητες διαδικασίας

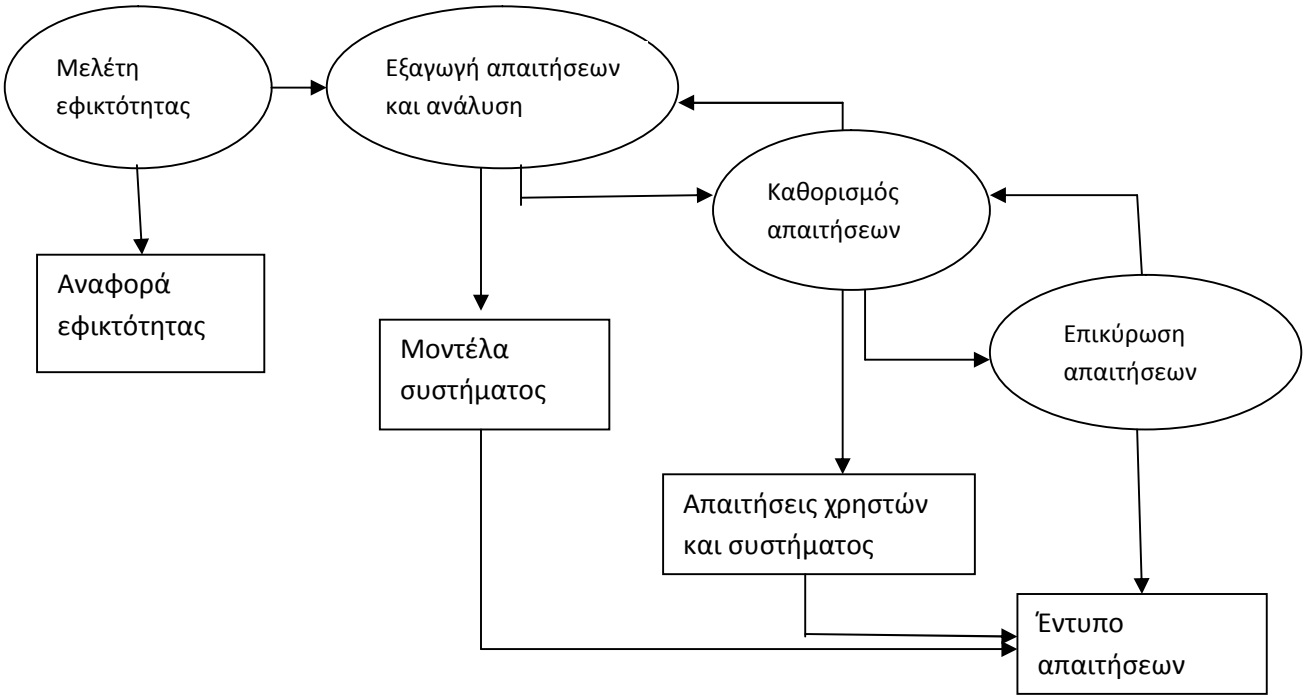
Οι τέσσερις βασικές λειτουργίες του καθορισμού των απαιτήσεων, της ανάπτυξης, της επικύρωσης και της εξέλιξης οργανώνονται διαφορετικά σε διαφορετικές διαδικασίες. Στο μοντέλο καταρράκτη, οργανώνονται σε σειρά, ενώ στην εξελικτική ανάπτυξη μπερδεύονται. Το πώς γίνονται αυτές οι λειτουργίες εξαρτάται από το είδος του λογισμικού, τους ανθρώπους και τις δομές της επιχείρησης που εμπλέκονται. Δεν υπάρχει σωστός ή λανθασμένος τρόπος να οργανώσεις αυτές τις λειτουργίες και ο σκοπός μας σε αυτό το κεφάλαιο είναι απλά να εισάγουμε το πώς αυτές μπορεί να οργανωθούν.

2.6 Απαιτήσεις συστήματος

Οι προδιαγραφές συστήματος ή σχεδιασμός και ανάλυση απαιτήσεων είναι η διαδικασία της κατανόησης και καθορισμού των υπηρεσιών που απαιτούνται από το σύστημα και της αναγνώρισης των περιορισμών της λειτουργίας και ανάπτυξης του συστήματος.

Η διαδικασία ανάλυσης και καθορισμού των απαιτήσεων φαίνεται στο σχήμα 10. Αυτή η διαδικασία οδηγεί στην παραγωγή του εντύπου απαιτήσεων το οποίο αποτελεί και τις

προδιαγραφές του συστήματος. Οι απαιτήσεις συνήθως παρουσιάζονται σε δυο επίπεδα λεπτομέρειας στο έντυπο. Τελικοί χρήστες και πελάτες απαιτούν υψηλού επιπέδου τεκμηρίωση των απαιτήσεων. Οι σχεδιαστές του συστήματος απαιτούν μια πιο λεπτομερή ανάλυση απαιτήσεων.



ΣΧΗΜΑ 10. Η διαδικασία σχεδιασμού των απαιτήσεων

Υπάρχουν τέσσερις κύριες φάσεις στην διαδικασία σχεδιασμού των απαιτήσεων:

Η μελέτη εφικτότητας. Γίνεται μια εκτίμηση αν οι καθορισμένες ανάγκες των χρηστών μπορούν να ικανοποιηθούν χρησιμοποιώντας τις τρέχουσες τεχνολογίες λογισμικού και εξοπλισμού μελέτη εξετάζει αν το προτεινόμενο σύστημα είναι οικονομικά ανεκτό από την επιχείρηση και αν μπορεί να αναπτυχθεί λαμβάνοντας υπόψη τους υπάρχοντες οικονομικούς περιορισμούς των κεφαλαίων. Μια μελέτη εφικτότητας πρέπει να είναι σχετικά οικονομική

και γρήγορη. Το αποτέλεσμα θα πρέπει να καθορίζει την απόφαση αν θα προχωρήσουμε ή όχι με μια πιο λεπτομερή ανάλυση.

Η εξαγωγή και ανάλυση των απαιτήσεων. Είναι η διαδικασία εξαγωγής των απαιτήσεων του συστήματος μέσα από την παρατήρηση των υπαρχόντων συστημάτων, συζητήσεις με τους μελλοντικούς χρήστες και διαχειριστές, ανάλυση έργου κτλ. Αυτό μπορεί να περιέχει και την ανάπτυξη ενός ή περισσοτέρων μοντέλων συστημάτων και προτύπων. Αυτό βοηθά τον αναλυτή να κατανοήσει το σύστημα του οποίου οι απαιτήσεις θα καθοριστούν.

Ο καθορισμός των απαιτήσεων. Η ενέργεια της μετάφρασης των πληροφοριών που συλλέχτηκαν κατά την διάρκεια της διαδικασίας ανάλυσης σε ένα έντυπο που καθορίζει ένα σύνολο απαιτήσεων. Μπορεί να περιέχονται δυο είδη απαιτήσεων στο έντυπο. Οι απαιτήσεις των χρηστών είναι γενικές απαιτήσεις συστήματος για τον πελάτη και τελικό χρήστη του συστήματος. Οι απαιτήσεις συστήματος είναι μια πιο λεπτομερής περιγραφή των λειτουργικοτήτων που παρέχονται.

Η επικύρωση των απαιτήσεων. Αυτή η δραστηριότητα ελέγχει τις απαιτήσεις πόσο ρεαλιστικές είναι, πόσο επαρκείς και αυτοτελείς. Κατά τη διάρκεια αυτής της διαδικασίας αναπόφευκτα ανακαλύπτονται σφάλματα στο έντυπο των απαιτήσεων. Πρέπει κατόπιν να τροποποιηθούν ώστε να γίνει η διόρθωσή τους.

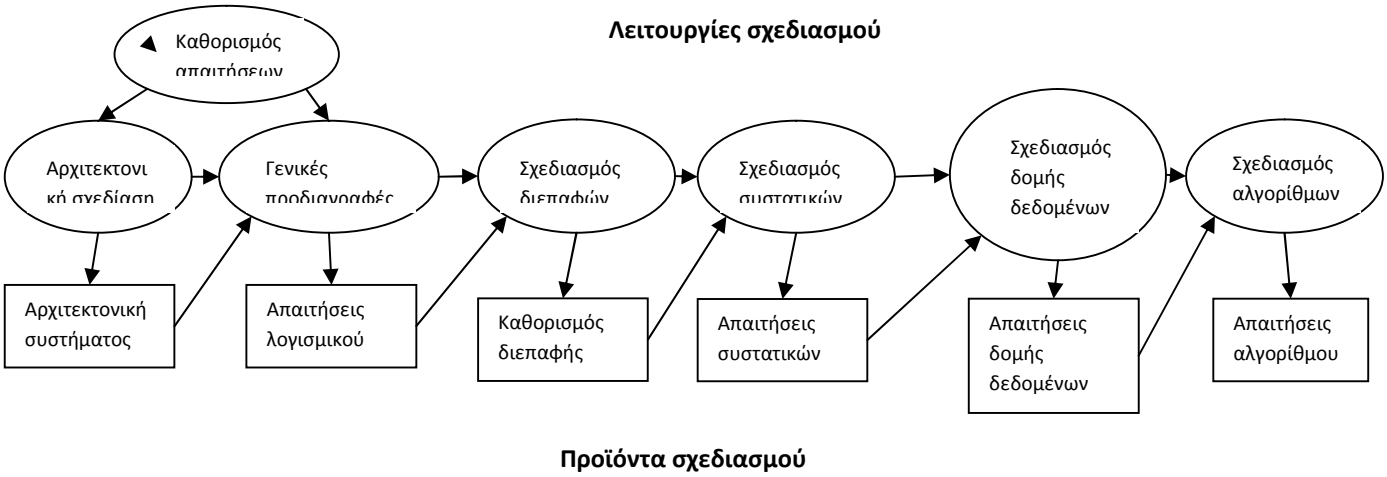
Φυσικά οι δραστηριότητες στην διαδικασία καθορισμού των απαιτήσεων δεν γίνονται σε ακριβή ακολουθία. Η ανάλυση των απαιτήσεων συνεχίζει κατά την διάρκεια του καθορισμού και της ανάλυσης των προδιαγραφών και νέες απαιτήσεις έρχονται στο φως κατά την διαδικασία. Επομένως, οι ενέργειες της ανάλυσης, ορισμού και καθορισμού των απαιτήσεων μπλέκονται. Σε ευέλικτες μεθόδους όπως extreme programming οι απαιτήσεις αναπτύσσονται αυξητικά ανάλογα με τις προτεραιότητες των χρηστών και η εκμαίευση των απαιτήσεων έρχεται από χρήστες που είναι μέλη της ομάδας ανάπτυξης.

2.7 Σχεδιασμός και υλοποίηση λογισμικού

Το στάδιο της εφαρμογής της ανάπτυξης του λογισμικού είναι η διαδικασία της μετατροπής των απαιτήσεων του συστήματος σε ένα εκτελέσιμο σύστημα. Περιέχει πάντα διαδικασίες σχεδίασης και προγραμματισμού αλλά, αν χρησιμοποιείται μια εξελικτική προσέγγιση ανάπτυξης, μπορεί επίσης να περιέχει διόρθωση και τελειοποίηση της ανάλυσης των απαιτήσεων.

Ο σχεδιασμός του λογισμικού είναι μια περιγραφή της δομής του λογισμικού που θα υλοποιηθεί, των δεδομένων που είναι μέρος του συστήματος, των διεπαφών μεταξύ των συστατικών στοιχείων του συστήματος και μερικές φορές των αλγορίθμων που χρησιμοποιούνται. Οι σχεδιαστές δεν φτάνουν αμέσως σε έναν τελικό σχεδιασμό αλλά διαρκώς αναπτύσσουν τον σχεδιασμό τους μέσω ενός αριθμού εκδόσεων. Η διαδικασία σχεδιασμού καθώς προχωράει προϋποθέτει την προσθήκη επισημότητας και λεπτομέρειας με σταθερή μετάβαση πίσω ώστε να διορθωθούν παλιότερες εκδόσεις.

Η διαδικασία σχεδιασμού μπορεί να περιλαμβάνει την ανάπτυξη ορισμένων μοντέλων του συστήματος σε διαφορετικά επίπεδα εννοιών. Καθώς ένας σχεδιασμός αποσυντίθεται, λάθη και παραλείψεις στα ενωρίτερα στάδια φαίνονται. Αυτές οι ανατροφοδοτήσεις είναι που επιτρέπουν στα προηγούμενα μοντέλα να βελτιώνονται. Στην εικόνα 11, φαίνεται ένα μοντέλο αυτής της διαδικασίας που δείχνει τις περιγραφές σχεδιασμού που προκύπτουν στα διάφορα στάδια σχεδίασης. Αυτό το διάγραμμα προτείνει ότι τα στάδια σχεδιασμού είναι συνεχόμενα. Στην πραγματικότητα, δεν είναι. Η ανατροφοδότηση από ένα στάδιο σε ένα άλλο και η συνακόλουθη εργασία ανασχεδιασμού είναι αναπόφευκτες σε όλες τις διαδικασίες σχεδιασμού.

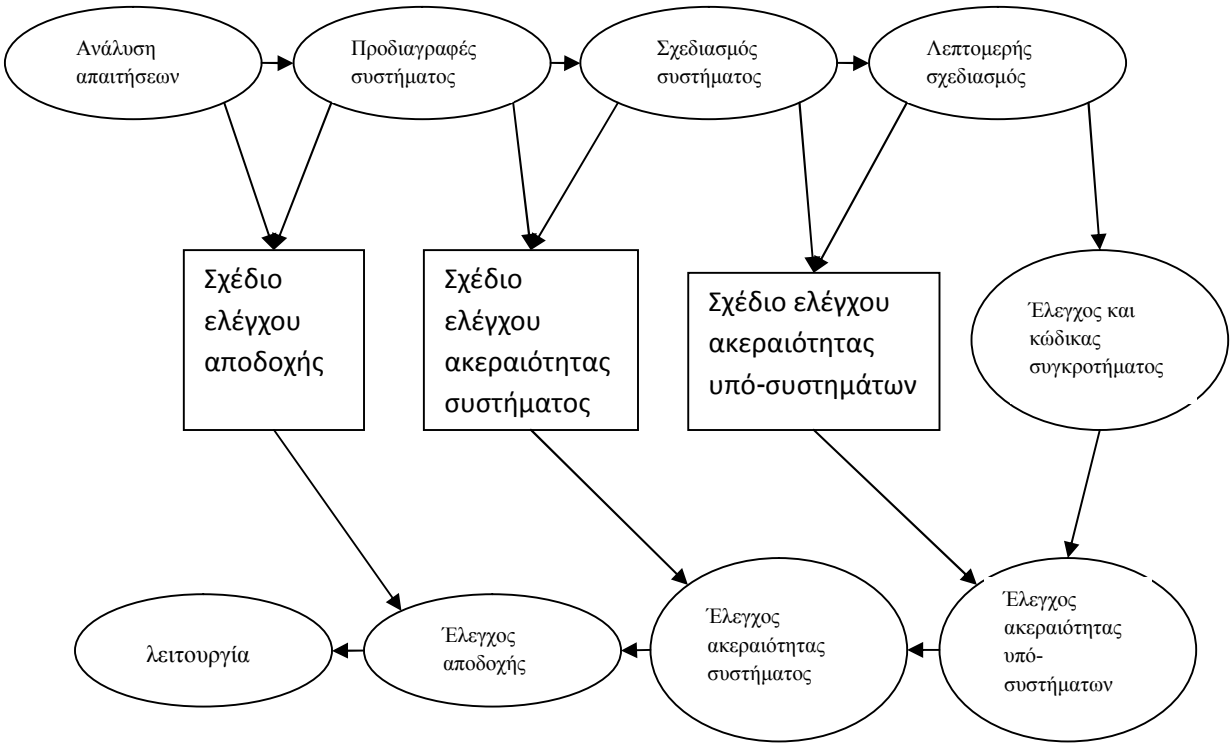


Σχήμα 11:Ένα γενικό μοντέλο της διαδικασίας σχεδιασμού

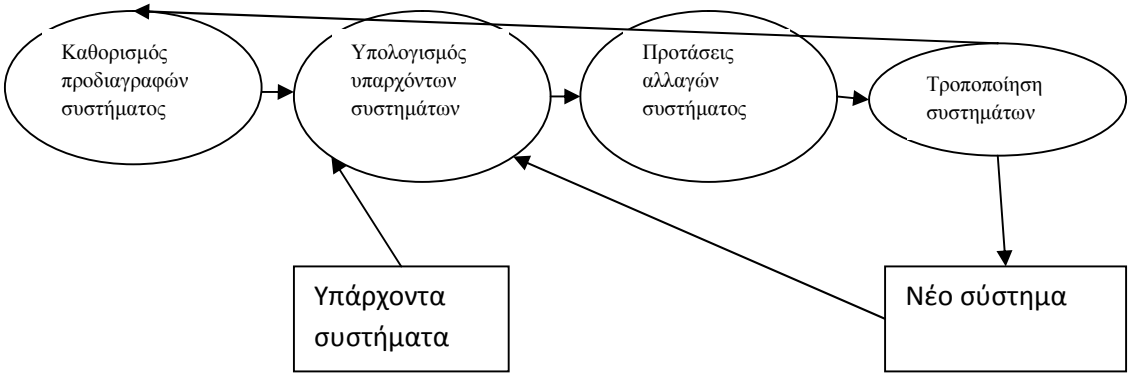
Συνήθως στην πορεία της ανάπτυξης του λογισμικού και κατά την διάρκεια των ελέγχων του κώδικα του λογισμικού , οι προγραμματιστές ανακαλύπτουν σφάλματα. Η διαδικασία απαλλαγής του λογισμικού από αυτά καλείται αποσφαλμάτωση και τα στάδιά της φαίνονται στο σχήμα 12.

Στο σχήμα 13 μπορεί κανείς να δει τα επιμέρους στάδια που ακολουθούνται κατά την διαδικασία ελέγχου του λογισμικού. Η διαδικασία αυτή οδηγεί στην επικύρωση του λογισμικού ή γενικότερα όπως λέμε στην επαλήθευση και επικύρωσή του.

Στο Σχήμα 14 φαίνονται τα στάδια της διαδικασίας εξέλιοξης του λογισμικού που παράγεται. Τυχόν νέες απαιτήσεις από τους χρήστες ή το περιβάλλον, μπορεί να απαιτήσουν την μετεξέλιξη του λογισμικού και την περαιτέρω βελτίωσή του.



Σχήμα 13: Οι φάσεις ελέγχου στην διαδικασία λογισμικού



Σχήμα 14 : Εξέλιξη συστημάτων

2.8 Rational Unified Process (RUP)

Η λογικά ενοποιημένη διαδικασία ή Rational Unified Process (RUP) είναι ένα μοντέρνο μοντέλο διαδικασίας ανάπτυξης και υλοποίησης λογισμικού είναι πρακτικά ο οδηγός της αποτελεσματικής χρήσης της Unified Modeling Language (UML) στη διαδικασία ανάπτυξης λογισμικού. Η RUP υποστηρίζεται από εργαλεία που αυτοματοποιούν μεγάλο τμήμα της διαδικασίας. Τα εργαλεία χρησιμοποιούνται για κατασκευή και συντήρηση δημιουργημάτων/παραδοτέων της διαδικασίας της τεχνολογίας λογισμικού. Τέτοια παραδοτέα μπορεί να είναι μοντέλα γραφικής απεικόνισης, έγγραφα η κώδικες. Τα εργαλεία που προβλέπονται από τη RUP είναι εξαιρετικά χρήσιμα στην υποστήριξη της διαδικασίας τήρησης της μεθοδολογίας.

Οι δραστηριότητες της RUP δημιουργούν και συντηρούν μοντέλα και στοχεύουν όχι στην παραγωγή πολλών εγγράφων αλλά δίνουν έμφαση στην ανάπτυξη και συντήρηση μοντέλων που είναι σημασιολογικά πλούσιες αναπαραστάσεις του υπό ανάπτυξη συστήματος λογισμικού.

Παρακάτω περιγράφονται οι φάσεις του μοντέλου διαδικασίας RUP:

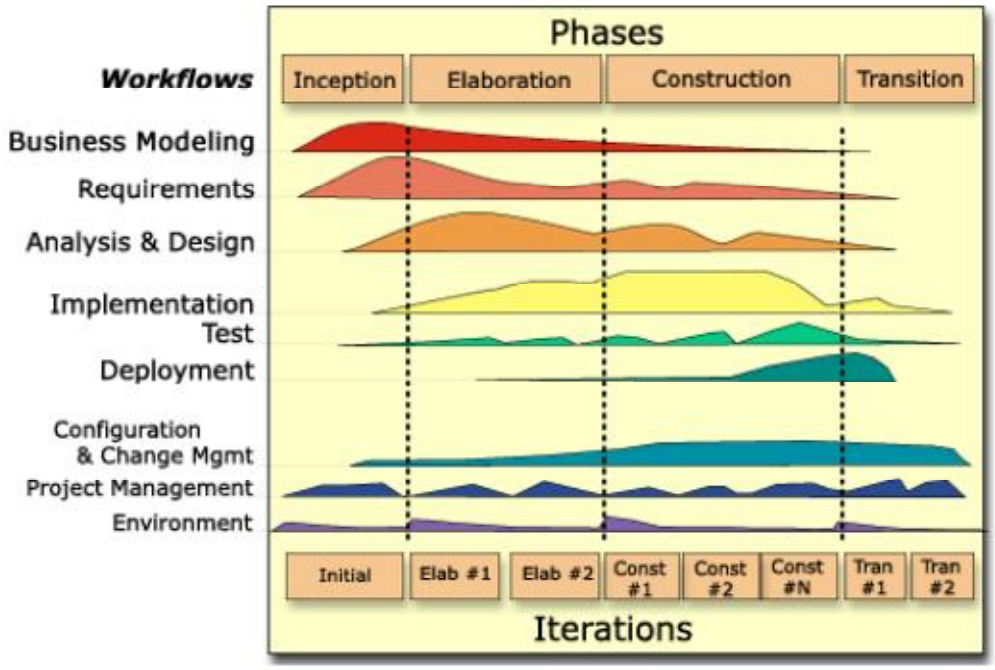
1. Σύλληψη (Inception) – Ορισμός του έργου και της έκτασής του. Παρουσιάζεται η αρχική ιδέα του συστήματος μέχρι του σημείου που είναι πολύ καλά θεμελιωμένη για να επιτρέψει την είσοδο στη φάση επεξεργασίας
2. Επεξεργασία (Elaboration) – Προσδιορισμός βασικών περιπτώσεων χρήσης, κατάστρωση μεθόδου υλοποίησης έργου, μοντελοποίηση χαρακτηριστικών έργου, ορισμός αρχιτεκτονικής συστήματος, εντοπισμός μεγάλων κινδύνων
3. Κατασκευή (Construction) – Υλοποίηση του έργου

4. Μετάβαση (Transition) – Εγκατάσταση συστήματος στο περιβάλλον χρήσης του, εκπαίδευση χρηστών, έλεγχος συστήματος από χρήστες – έναρξη φάσης συντήρησης λογισμικού και όχι τέλος διαδικασίας ανάπτυξης

Τα βασικά στοιχεία που χαρακτηρίζουν τη μεθοδολογία ανάπτυξης RUP είναι τα εξής:

1. Εξελικτικό μοντέλο με ανάδραση
2. Καθοδηγείται από περιπτώσεις χρήσης (use cases)
3. Είναι αρχιτεκτονικό-κεντρικό μοντέλο (4+1 άποψεις –views)
4. Χρησιμοποιεί τη UML σαν γλώσσα μοντελοποίησης
5. Πλούσιο πλαίσιο υποστήριξης της διαδικασίας

Η στατική άποψη του μοντέλου RUP εστιάζει στις δραστηριότητες που εξελίσονται κατά την διαδικασία ανάπτυξης και φαίνεται στην παρακάτω εικόνα:



Η RUP περιγράφει πώς είναι εφικτό να υλοποιηθούν εμπορικά δοκιμασμένες προσεγγίσεις ανάπτυξης λογισμικού από τις ομάδες ανάπτυξης λογισμικού γνωστές και ως «βέλτιστες πρακτικές» (best practices) που είναι:

1. Επαναληπτική ανάπτυξη λογισμικού – Develop iteratively

2. Διαχείριση απαιτήσεων – Manage requirements

3. Χρήση αρχιτεκτονικών που βασίζονται σε δομικά στοιχεία – Use component architectures

4. Μοντελοποίηση λογισμικού με τη χρήση γραφικών εργαλείων – Model visually

5. Επαλήθευση ποιότητας λογισμικού – Verify software quality

6. Έλεγχος και διαχείριση αλλαγών λογισμικού – Control changes

Στη συνέχεια απαριθμούνται τα πλεονεκτήματα και τα μειονεκτήματα χρησιμοποίησης του μοντέλου ανάπτυξης RUP.

ΠΛΕΟΝΕΚΤΗΜΑΤΑ

1. Δυνατότητα ελέγχου του κινδύνου.
2. Ανάπτυξη του έργου κάνοντας συνδυασμένη σταδιακή, ανάλυση – σχεδιασμό – υλοποίηση.
3. Δίνει τη δυνατότητα σε όλες της να συμμετέχουν στο έργο και να συνεισφέρουν στην εμπειρία τους.
4. Οι επαναλήψεις δημιουργούν τη δυνατότητα της διόρθωσης και του επαναπροσδιορισμού των επιλογών.
5. Ανάλυση και ανάπτυξη ανά component.

ΜΕΙΟΝΕΚΤΗΜΑΤΑ

1. Δεν καλύπτει έννοιες όπως η συντήρηση και αναβάθμιση λογισμικού.
2. Δεν υποστηρίζει ρητά τις προσπάθειες ανάπτυξης υποδομής σε όλο τον οργανισμό και δεν ανταποκρίνεται στην μεγάλη κλίμακα επαναχρησιμοποίησης συστατικών μέσα στην επιχείρηση.
3. Δεν μπορεί να αυτοματοποιήσει κάθε πτυχή της διαδικασίας λογισμικού.
4. Αδυναμίες επίσης εστιάζονται στην διαχείριση μετρήσεων , επαναχρησιμοποίησης, διαχείριση ανθρώπων και διενέργεια δοκιμών.
5. Δεν δίνει επίσης απαραίτητη έμφαση στην επιχειρηματική λογική αλλά μόνο στην λογική της κατασκευής του λογισμικού.

3. Απαιτήσεις Λογισμικού

3.1 Γενικά

3.1.1 Ορισμός

Οι απαιτήσεις για ένα σύστημα είναι οι περιγραφές των υπηρεσιών που παρέχονται από το σύστημα και τους λειτουργικούς περιορισμούς του. Αυτές οι απαιτήσεις απεικονίζουν τις ανάγκες των πελατών για ένα σύστημα που βοηθούν να λυθεί κάποιο πρόβλημα όπως ο έλεγχος μιας συσκευής, η τοποθέτηση μιας διαταγής ή η εύρεση των πληροφοριών. Η διαδικασία, και αυτούς τους υπηρεσίες και περιορισμούς καλείται σχεδιασμός απαιτήσεων.

Ο όρος απαιτήσεις δεν χρησιμοποιείται στη βιομηχανία λογισμικού με έναν συνεπή τρόπο. Σε μερικές περιπτώσεις, μια απαίτηση είναι απλά μια υψηλού επιπέδου, αφηρημένη δήλωση μιας υπηρεσίας που το σύστημα πρέπει να παρέχει ή ένας περιορισμός στο σύστημα. Σε άλλη ακραία, είναι ένας λεπτομερής, επίσημος καθορισμός μιας λειτουργίας συστημάτων.

3.1.2 Διαχωρισμός Απαιτήσεων

Με τον όρο απαιτήσεις χρηστών επικεντρωνόμαστε στις υψηλού επιπέδου γενικές απαιτήσεις ενώ λέγοντας απαιτήσεις συστημάτων εννοούμε μια λεπτομερή περιγραφή του τι θα κάνει το σύστημα. Οι απαιτήσεις χρηστών και οι απαιτήσεις συστημάτων μπορούν να καθοριστούν ως εξής:

1. Οι απαιτήσεις χρηστών είναι δηλώσεις, σε μια φυσική γλώσσα συν τα διαγράμματα, ποιων υπηρεσιών το σύστημα αναμένεται για να παρέχει και οι περιορισμοί κάτω από τους οποίους πρέπει να λειτουργήσει.

2. Οι απαιτήσεις συστημάτων καθορίζουν τις λειτουργίες του συστήματος, τις υπηρεσίες και τους λειτουργικούς περιορισμούς λεπτομερώς. Το έγγραφο απαιτήσεων συστημάτων (μερικές φορές αποκαλούμενο λειτουργική προδιαγραφή) πρέπει να είναι ακριβές. Πρέπει να καθορίσει ακριβώς τι είναι να εφαρμοστεί. Μπορεί να είναι μέρος της σύμβασης μεταξύ του αγοραστή συστημάτων και των προγραμματιστών λογισμικού.

3.2 Λειτουργικές και μη λειτουργικές απαιτήσεις

Οι απαιτήσεις συστημάτων λογισμικού είναι συχνά ταξινομημένες ως λειτουργικές απαιτήσεις, μη λειτουργικές απαιτήσεις ή απαιτήσεις περιοχών:

1. Οι λειτουργικές απαιτήσεις Είναι δηλώσεις των υπηρεσιών που το σύστημα πρέπει να παρέχει, πώς το σύστημα πρέπει να αντιδράσει στις ιδιαίτερες εισαγωγές και πώς το σύστημα πρέπει να συμπεριφερθεί σε συγκεκριμένες καταστάσεις. Σε μερικές περιπτώσεις, οι λειτουργικές απαιτήσεις μπορούν επίσης ρητά να δηλώσουν τι το σύστημα δεν πρέπει να κάνει.

2. Οι μη λειτουργικές απαιτήσεις Είναι περιορισμοί στις υπηρεσίες ή τις λειτουργίες που προσφέρονται από το σύστημα. Περιλαμβάνουν τους περιορισμούς συγχρονισμού, τους περιορισμούς στη αναπτυξιακή διαδικασία και τα πρότυπα. Οι μη λειτουργικές απαιτήσεις ισχύουν συχνά για το σύστημα συνολικά. Συνήθως ακριβώς δεν ισχύουν για τα μεμονωμένα χαρακτηριστικά γνωρίσματα συστημάτων ή τις υπηρεσίες.

3. Οι απαιτήσεις περιοχών Είναι απαιτήσεις που προέρχονται από την περιοχή εφαρμογής του συστήματος και που απεικονίζουν τα χαρακτηριστικά και τους περιορισμούς εκείνης της περιοχής. Μπορούν να είναι λειτουργικές ή μη λειτουργικές απαιτήσεις.

Στην πραγματικότητα, η διάκριση μεταξύ των διαφορετικών τύπων απαιτήσεων δεν είναι τόσο ευδιάκριτη όσο αυτοί οι απλοί ορισμοί προτείνουν. Μια απαίτηση χρηστών ενδιαφερόμενη για την ασφάλεια μπορεί, για παράδειγμα, να εμφανιστεί να είναι μια μη λειτουργική απαίτηση. Εντούτοις, όταν αναπτύσσεται λεπτομερέστερα, αυτή η απαίτηση μπορεί να παραγάγει άλλες απαιτήσεις που είναι σαφώς λειτουργικές, όπως η ανάγκη να περιληφθούν οι εγκαταστάσεις επικύρωσης χρηστών στο σύστημα.

3.3 Απαιτήσεις Χρηστών

Οι απαιτήσεις χρηστών για ένα σύστημα πρέπει να περιγράψουν τις λειτουργικές και μη λειτουργικές απαιτήσεις, έτσι ώστε να είναι κατανοητές από τους χρήστες συστημάτων χωρίς λεπτομερείς τεχνικές γνώσεις. Πρέπει μόνο να διευκρινίσουν την εξωτερική συμπεριφορά του συστήματος και πρέπει να αποφύγουν, όσο το δυνατόν περισσότερο, τα χαρακτηριστικά σχεδίου συστημάτων. Συνεπώς, εάν γράφετε τις απαιτήσεις χρηστών, δεν πρέπει να χρησιμοποιήσετε την επαγγελματική γλώσσα λογισμικού, τις δομημένες σημειώσεις ή τις επίσημες σημειώσεις, ή να περιγράψετε την εφαρμογή του συστήματος. Πρέπει να γράψετε τις απαιτήσεις χρηστών σε απλή γλώσσα, με απλούς πίνακες και μορφές και με βοηθητικά διαγράμματα.

Εντούτοις, τα διάφορα προβλήματα μπορούν να προκύψουν όταν γράφονται οι απαιτήσεις στις προτάσεις φυσικής γλώσσας σε ένα έγγραφο κειμένων, όπως για παράδειγμα έλλειψη σαφήνειας, σύγχυση και συγχώνευση απαιτήσεων.

Οι απαιτήσεις χρηστών που περιλαμβάνουν πάρα πολλές πληροφορίες περιορίζουν την ελευθερία του υπεύθυνου για την ανάπτυξη συστημάτων ώστε να παρασχεθούν οι καινοτόμες λύσεις στα προβλήματα χρηστών και είναι δύσκολες να κατανοηθούν. Οι απαιτήσεις χρηστών πρέπει απλά να εστιάσουν στις βασικές ευκολίες που παρέχονται.

Όποτε είναι δυνατόν, πρέπει να προσπαθήσετε να συνδέσετε μια λογική με κάθε απαίτηση χρηστών. Η λογική πρέπει να εξηγήσει γιατί η απαίτηση έχει περιληφθεί και είναι ιδιαίτερα χρήσιμη όταν αλλάζουν τις απαιτήσεις.

Για να ελαχιστοποιήσουμε τις παρανοήσεις κατά το γράψιμο των απαιτήσεων χρηστών, ακολουθούμε τις παρακάτω απλές οδηγίες:

1. Εφευρίσκουμε ένα τυποποιημένο σχήμα και εξασφαλίζουμε ότι όλοι οι ορισμοί απαίτησης ταιριάζουν σε αυτό το σχήμα. Η τυποποίηση του σχήματος καθιστά τις παραλείψεις λιγότερο πιθανές και τις απαιτήσεις ευκολότερο να ελέγθούν.

2. Χρησιμοποιούμε τη γλώσσα με συνέπεια. Πρέπει πάντα να διακρίνουμε μεταξύ των υποχρεωτικών και επιθυμητών απαιτήσεων. Οι υποχρεωτικές απαιτήσεις είναι απαιτήσεις που το σύστημα πρέπει να υποστηρίξει και γράφονται συνήθως χρησιμοποιώντας «πρέπει». Οι επιθυμητές απαιτήσεις δεν είναι ουσιαστικές και γράφονται χρησιμοποιώντας «μπορεί», «θα έπρεπε» κτλ.

3. Χρησιμοποιούμε κείμενο τονισμένο (άρθρο σε εφημερίδα με μαύρους χαρακτήρες, σχόλιο σε εφημερίδα με πλάγιους χαρακτήρες ή χρώμα) για να διαλέξουμε τα μέρη κλειδί της απαίτησης.

4. Αποφεύγουμε, όσο το δυνατόν περισσότερο, τη χρήση της επαγγελματικής γλώσσας υπολογιστών. Αναπόφευκτα, εντούτοις, οι λεπτομερείς τεχνικοί όροι θα συρθούν στις απαιτήσεις χρηστών.

3.4 Απαιτήσεις Συστημάτων

Οι απαιτήσεις συστημάτων είναι διευρυμένες εκδόσεις των απαιτήσεων χρηστών που χρησιμοποιούνται από τους μηχανικούς λογισμικού ως αφετηρία για το σχέδιο συστημάτων. Προσθέτουν τη λεπτομέρεια και εξηγούν πώς οι απαιτήσεις χρηστών πρέπει να παρασχεθούν από το σύστημα. Μπορούν να χρησιμοποιηθούν ως τμήμα της σύμβασης για την εφαρμογή του συστήματος και πρέπει επομένως να είναι μια πλήρης και συνεπής προδιαγραφή ολόκληρου του συστήματος.

Ιδανικά, οι απαιτήσεις συστημάτων πρέπει απλά να περιγράψουν την εξωτερική συμπεριφορά του συστήματος και των λειτουργικών περιορισμών του. Δεν πρέπει να ενδιαφερθούν για το πώς το σύστημα πρέπει να σχεδιαστεί ή να εφαρμοστεί. Εντούτοις, στο επίπεδο λεπτομέρειας που απαιτείται να διευκρινιστεί εντελώς ένα σύνθετο σύστημα λογισμικού, είναι αδύνατο, στην πράξη, να αποκλειστούν όλες οι πληροφορίες σχεδίου. Υπάρχουν διάφοροι λόγοι για αυτό:

1. Μπορεί να πρέπει να σχεδιάσετε μια αρχική αρχιτεκτονική του συστήματος για να βοηθήσετε να κτίσετε την προδιαγραφή απαιτήσεων. Οι απαιτήσεις συστημάτων

οργανώνονται σύμφωνα με τα διαφορετικά υποσυστήματα που αποτελούν το σύστημα. Ο αρχιτεκτονικός καθορισμός είναι ουσιαστικός εάν θέλετε να επαναχρησιμοποιήσετε τα τμήματα λογισμικού κατά την εφαρμογή του συστήματος.

2. Στις περισσότερες περιπτώσεις, τα συστήματα πρέπει να επικοινωνήσουν με άλλα υπάρχοντα συστήματα. Αυτό περιορίζει τον σχεδιασμό, και αυτοί οι περιορισμοί επιβάλλουν τις απαιτήσεις στο νέο σύστημα.

3. Η χρήση μιας συγκεκριμένης αρχιτεκτονικής για να ικανοποιήσει τις μη λειτουργικές απαιτήσεις (όπως ο προγραμματισμός ν-έκδοσης για να επιτύχει την αξιοπιστία) μπορεί να είναι απαραίτητη. Ένας εξωτερικός ρυθμιστής που πρέπει να πιστοποιήσει ότι το σύστημα είναι ασφαλές μπορεί να διευκρινίσει ότι ένα αρχιτεκτονικό σχέδιο που έχει πιστοποιηθεί ήδη χρησιμοποιείται.

Η φυσική γλώσσα χρησιμοποιείται συχνά για να γράψει τις προδιαγραφές απαιτήσεων συστημάτων καθώς επίσης και τις απαιτήσεις χρηστών. Εντούτοις, επειδή οι απαιτήσεις συστημάτων είναι πιο λεπτομερείς από τις απαιτήσεις χρηστών, οι προδιαγραφές φυσικής γλώσσας μπορούν προκαλέσουν σύγχυση ή να είναι δυσνόητες

3.5 Απαιτήσεις Διεπαφών

Σχεδόν όλα τα συστήματα λογισμικού πρέπει να λειτουργήσουν με τα υπάρχοντα συστήματα που έχουν εφαρμοστεί ήδη και έχουν εγκατασταθεί σε ένα περιβάλλον. Εάν το νέο σύστημα και τα υπάρχοντα συστήματα πρέπει να λειτουργήσουν μαζί, οι διεπαφές των υπαρχόντων συστημάτων πρέπει να διευκρινιστούν με ακρίβεια. Αυτές οι προδιαγραφές πρέπει να καθοριστούν νωρίς στη διαδικασία και να περιληφθούν (ίσως ως παράρτημα) στο έγγραφο απαιτήσεων.

Υπάρχουν τρεις τύποι διεπαφών που μπορεί να πρέπει να καθοριστούν:

1. Διαδικαστικές διεπαφές όπου τα υπάρχοντα προγράμματα ή τα υποσυστήματα προσφέρουν μια σειρά των υπηρεσιών που προσεγγίζονται με την κλήση των διαδικασιών διεπαφών. Αυτές οι διεπαφές καλούνται μερικές φορές διεπαφές προγραμματισμού εφαρμογής (APIs).

2. Δομές δεδομένων που περνούν από ένα υποσύστημα σε άλλο. Τα γραφικά πρότυπα στοιχείων είναι οι καλύτερες σημειώσεις για αυτόν τον τύπο περιγραφής. Εάν είναι απαραίτητο, οι περιγραφές προγράμματος στην Java ή C++ μπορούν να παραχθούν αυτόματα από αυτές τις περιγραφές.

3. Αντιπροσωπεύσεις των δεδομένων που έχουν καθιερωθεί για ένα υπάρχον υποσύστημα. Αυτές οι διεπαφές είναι οι πιό κοινές στο ενσωματωμένο, σε πραγματικό χρόνο σύστημα. Μερικές γλώσσες προγραμματισμού όπως το ADA (αν και όχι η Java) υποστηρίζει αυτό το επίπεδο προδιαγραφής. Εντούτοις, ο καλύτερος τρόπος να περιγραφούν αυτοί είναι πιθανώς να χρησιμοποιηθεί ένα διάγραμμα της δομής με τους σχολιασμούς που εξηγούν την λειτουργία κάθε ομάδας bits.

Οι επίσημες σημειώσεις, επιτρέπουν στις διεπαφές για να καθοριστούν με έναν σαφή τρόπο, αλλά η εξειδικευμένη φύση τους σημαίνει ότι δεν είναι κατανοητές χωρίς πρόσθετη κατάρτιση. Χρησιμοποιούνται σπάνια στην πράξη για τις απαιτήσεις διεπαφών αν και, κατά την άποψή μου, είναι ιδανικά ταιριαγμένοι για αυτόν το λόγο. Μια γλώσσα προγραμματισμού όπως η Java μπορεί να χρησιμοποιηθεί για να περιγράψει τη σύνταξη της διεπαφής.

3.6 Το έντυπο απαιτήσεων Λογισμικού

Το έγγραφο απαιτήσεων λογισμικού (μερικές φορές αποκαλούμενο απαιτήσεις λογισμικού η προδιαγραφή ή SRS) είναι η επίσημη ανακοίνωση αυτού που οι υπεύθυνοι για την ανάπτυξη συστημάτων πρέπει να εφαρμόσουν. Πρέπει να περιλάβει και τις απαιτήσεις χρηστών για ένα σύστημα και λεπτομερή προδιαγραφή των απαιτήσεων συστημάτων. Σε μερικές περιπτώσεις, οι απαιτήσεις χρηστών και συστημάτων μπορούν να ενσωματωθούν σε μια ενιαία περιγραφή. Σε άλλες περιπτώσεις, οι απαιτήσεις χρήστη καθορίζονται σε μια εισαγωγή στις απαιτήσεις συστημάτων . Εάν υπάρχει ένας μεγάλος αριθμός απαιτήσεων, οι λεπτομερείς απαιτήσεις συστημάτων μπορούν να παρουσιαστούν σε ένα χωριστό έγγραφο.

Το έγγραφο απαιτήσεων έχει ένα διαφορετικό σύνολο χρηστών, που κυμαίνονται από τη διοίκηση της οργάνωσης που πληρώνει για το σύστημα στους υπεύθυνους μηχανικούς για την ανάπτυξη του λογισμικού.

Η ποικιλομορφία των πιθανών χρηστών σημαίνει ότι το έγγραφο απαιτήσεων πρέπει να είναι ένας συμβιβασμός μεταξύ της διαβίβασης των απαιτήσεων στους πελάτες, που καθορίζουν τις απαιτήσεις με ακριβείς λεπτομέρειες για τους υπεύθυνους για την ανάπτυξη και τους ελεγκτές, και συμπεριλαμβανομένων των πληροφοριών για την πιθανή εξέλιξη συστημάτων. Οι πληροφορίες για τις προσδοκώμενες αλλαγές μπορούν να βοηθήσουν τους σχεδιαστές συστήματος να αποφύγουν τις περιοριστικές αποφάσεις σχεδίου και να βοηθήσουν τους τεχνικούς συντήρησης συστημάτων, που πρέπει να προσαρμόσουν το σύστημα στις νέες απαιτήσεις.

Το επίπεδο λεπτομέρειας που πρέπει να περιλάβετε σε ένα έγγραφο απαιτήσεων εξαρτάται από τον τύπο συστήματος που αναπτύσσεται και την χρησιμοποιούμενη αναπτυξιακή διαδικασία. Όταν το σύστημα θα αναπτυχθεί από έναν εξωτερικό ανάδοχο, οι κρίσιμες απαιτήσεις συστημάτων πρέπει να είναι ακριβείς και πολύ λεπτομερείς. Όταν υπάρχει περισσότερη ευελιξία στις απαιτήσεις και όπου μια εσωτερική, επαναληπτική αναπτυξιακή διαδικασία χρησιμοποιείται, το έγγραφο απαιτήσεων μπορεί να είναι πολύ λιγότερο λεπτομερές και οποιαδήποτε ασάφεια λύνεται κατά την ανάπτυξη του συστήματος. Διάφορες μεγάλες οργανώσεις, όπως το αμερικανικό υπουργείο Αμύνης και IEEE, έχουν καθορίσει τα πρότυπα για τα έγγραφα απαιτήσεων.

4. Η αρχιτεκτονική λογισμικού

4.1 Ορισμός

Η αρχιτεκτονική λογισμικού περιγράφει τα στοιχεία από τα οποία αποτελείται ένα σύστημα, τις αλληλεπιδράσεις ανάμεσα σε αυτά τα στοιχεία, καθώς και πρότυπα και περιορισμούς που καθοδηγούν τη σύνθεση αυτών των στοιχείων. Σε γενικές γραμμές, ένα σύστημα περιγράφεται σε επίπεδο αρχιτεκτονικής σαν μια συλλογή υποσυστημάτων / ψηφίδων (**components**) και τις αλληλεπιδράσεις (**connectors**) ανάμεσα σε αυτά τα υποσυστήματα / ψηφίδες. Ένα σύστημα μπορεί να είναι υποσύστημα σε κάποια άλλη (μεγαλύτερη και πολυπλοκότερη) εφαρμογή. Παρέχει ένα τρόπο ανάλυσης και περιγραφής του συστήματος σε υψηλό αφαιρετικό επίπεδο και εστιάζει στις βασικές σχεδιαστικές αποφάσεις που λαμβάνουμε για ένα σύστημα

Η αρχιτεκτονική λογισμικού (*software architecture*) ασχολείται με (Shan and Garlan, 1996):

Την οργάνωση του συστήματος ως σύνθεση εξαρτημάτων.

Καθολικές δομές ελέγχου.

Πρωτόκολλα επικοινωνίας, συγχρονισμού και πρόσβασης σε αποθηκευμένα δεδομένα.

Την ευθυγράμμιση των λειτουργιών με τα στοιχεία του σχεδίου.

Τη σύνθεση των στοιχείων του σχεδίου.

Τη φυσική υλοποίηση του συστήματος.

Την απόδοση και την ανταπόκριση σε αυξανόμενες απαιτήσεις (*scaling*).

Τις δυνατότητες εξέλιξης.

Την επιλογή μεταξύ εναλλακτικών σχεδίων.

4.2 Στόχοι

Η αρχιτεκτονική ενός συστήματος λογισμικού περιγράφει το σύστημα ως σύνολο υπολογιστικών εξαρτημάτων και συνδέσεων μεταξύ αυτών των εξαρτημάτων. Αναλυτικότερα οι στόχοι που εξυπηρετεί είναι:

Η αρχιτεκτονική λογισμικού στοχεύει στον εντοπισμό και την καταγραφή, μέσα από την συσσωρευμένη πείρα των μηχανικών λογισμικού, συγκεκριμένων αρχιτεκτονικών μοτίβων και προτύπων (*architectural styles, architectural patterns*) που περιγράφουν συστήματα λογισμικού σε υψηλό επίπεδο αφαίρεσης.

Στην αρχή του σχεδιασμού ενός συστήματος λογισμικού εντοπίζουμε το αρχιτεκτονικό πρότυπο που ταιριάζει καλύτερα στο σύστημα.

Προσπαθούμε να μην επανεφευρίσκουμε τον τροχό αλλά να καθοδηγούμαστε από δοκιμασμένες από τον χρόνο και από την πράξη λύσεις.

Καθοδηγούμαστε έτσι ώστε να αποφύγουμε τον καταποντισμό μέσα σε μία θάλασσα αντικειμένων (Brushmann et al., 1996).

Τα αρχιτεκτονικά μοτίβα και πρότυπα χρησιμεύουν επίσης και ως μέσο επικοινωνίας μεταξύ των μηχανικών λογισμικού, αφού αποτελούν ένα κοινό λεξιλόγιο

Αποτελεί την βάση της διεργασίας ανάπτυξης λογισμικού. Αμέσως μετά την ανάλυση των απαιτήσεων επιλέγεται η κατάλληλη για το σύστημα αρχιτεκτονική , αναλύεται και παρουσιάζεται στους εταίρους. Κατόπιν το σύστημα σχεδιάζεται και υλοποιείται σύμφωνα με αυτήν και επιβεβαιώνεται ότι η υλοποίηση είναι σύμφωνη με την επιλεχθείσα αρχιτεκτονική.

Είναι πιθανό το σύστημα να μπορεί να αναλυθεί σε υποσυστήματα που το καθένα υλοποιείται με διαφορετική αρχιτεκτονική, ή να χρησιμοποιηθούν διάφορες αρχιτεκτονικές σε διαφορετικά επίπεδα αφαίρεσης.

4.3 Κριτήρια ποιότητας

Μπορούμε να ξεχωρίσουμε τα παρακάτω κριτήρια ποιότητας μιας αρχιτεκτονικής (Bush et al. 1998):

Η αρχιτεκτονική πρέπει να είναι καλά τεκμηριωμένη, με τη χρήση συμβόλων που καταλαβαίνουν όλοι οι εταίροι με την ελάχιστη προσπάθεια.

Οι εταίροι πρέπει να επιθεωρήσουν την αρχιτεκτονική.

Πρέπει να εντοπιστούν ποσοτικές μετρικές (όπως αριθμός δοσοληψιών στη μονάδα του χρόνου) όπως και ποιοτικές μετρικές (όπως συντηρησιμότητα) προτού να είναι πολύ αργά για να γίνουν αλλαγές.

Η αρχιτεκτονική πρέπει να είναι υλοποιήσιμη σε έναν αρχέτυπο σύστημα που του λείπουν οι λειτουργίες αλλά επιδεικνύει τα βασικά χαρακτηριστικά της. Το σύστημα αυτό θα πρέπει να μπορεί να εξελιχθεί βηματικά.

Η αρχιτεκτονική πρέπει να οδηγεί σε ένα μικρό σύνολο περιορισμών οι οποίοι είναι καταγεγραμμένοι και τεκμηριωμένοι.

Η αρχιτεκτονική πρέπει να αποτελείται από καλά ορισμένα τμήματα που να είναι όσο το δυνατόν ανεξάρτητα και να αποκρύπτουν όσο το δυνατόν τα εσωτερικά τους χαρακτηριστικά.

Διαφορετικές ομάδες θα πρέπει να μπορούν να εργαστούν παράλληλα σε διαφορετικά τμήματα της αρχιτεκτονικής.

Η αρχιτεκτονική πρέπει να έχει την ελάχιστη δυνατή εξάρτηση από το υλικό.

Η αρχιτεκτονική δεν πρέπει ποτέ να εξαρτάται από κάποιο εμπορικό εργαλείο ή προϊόν.

Τα τμήματα που παράγουν δεδομένα πρέπει να είναι διαφορετικά από αυτά που καταναλώνουν δεδομένα.

Τα σενάρια χρήσης της αρχιτεκτονικής πρέπει να είναι λίγα και απλά.

4.4 Είδη αρχιτεκτονικής σχεδίασης

Στην συνέχεια περιγράφονται ορισμένα είδη αρχιτεκτονικής σχεδίασης που χρησιμοποιούνται στην αρχιτεκτονική λογισμικού.

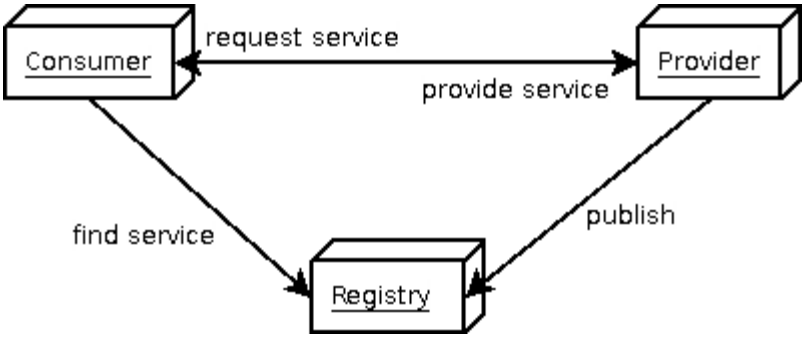
4.4.1 Η υπηρεσιακοστραφής αρχιτεκτονική

Η υπηρεσιακοστραφής αρχιτεκτονική (*service-oriented architecture, SOA*) αντιμετωπίζει το λογισμικό ως σύνολο υπηρεσιών. Διακρίνει:

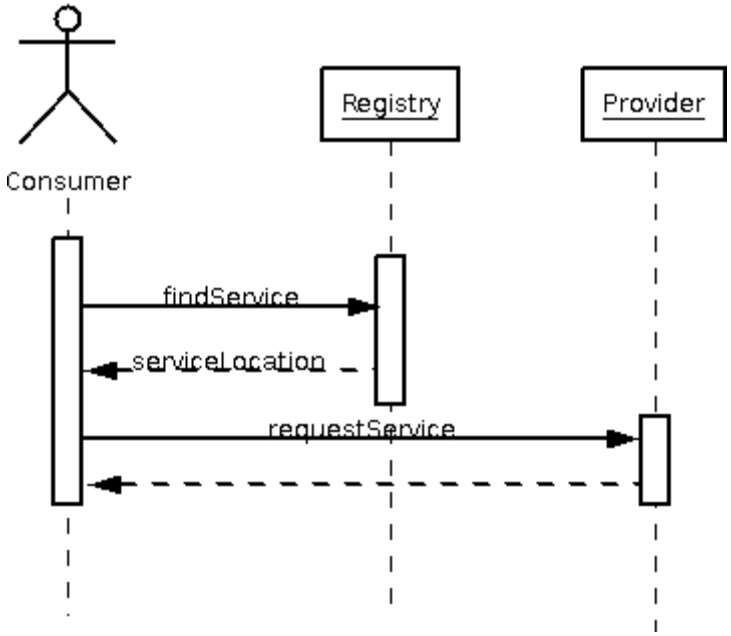
Παροχείς υπηρεσιών (*service providers*)

Καταναλωτές (*consumers*)

Κατάλογοι διαθέσιμων υπηρεσιών (*registries*)



Υπηρεσιακοστραφής Αρχιτεκτονική



Διάγραμμα Ακολουθίας Υπηρεσιακοστραφής Αρχιτεκτονικής

Πλεονεκτήματα

Πλήρης διαχωρισμός καταναλωτή-παροχέα.

Ανταγωνιστικοί παροχείς μπορεί να προσφέρουν την ίδια υπηρεσία με διαφορετικά χαρακτηριστικά ποιότητας, αξιοπιστίας, τιμής, κ.λπ.

Βασίζεται σε ανοιχτά πρότυπα (SOAP, REST, WSDL, UDDI, κ.λπ).

Η ανταλλαγή δεδομένων γίνεται με χρήση της XML.

Υπάρχουσες υπηρεσίες μπορούν να συνδυάζονται για τη δημιουργία πιο σύνθετων υπηρεσιών και ολόκληρων επιχειρηματικών διαδικασιών.

Μειονεκτήματα

Ο ορισμός υπηρεσιών και ο συνδυασμός τους είναι περίπλοκος και τα εμπλεκόμενα πρότυπα πολλά και αλληλοεπικαλυπτόμενα.

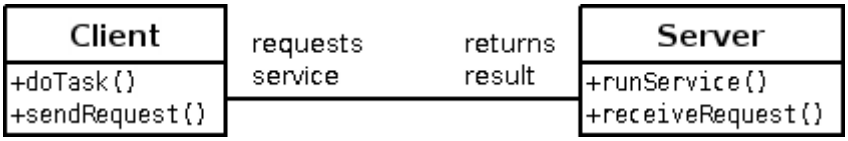
Μερικές από τις απαραίτητες τεχνολογίες δεν είναι ακόμα έτοιμες

Δεν έχουν δημιουργηθεί ακόμα κατάλογοι διαθέσιμων υπηρεσιών, και ίσως δεν δημιουργηθούν ποτέ για επιχειρησιακούς λόγους. Οι υπάρχουσες υλοποιήσεις είναι υπηρεσίες ιστού (***web services***) που περιλαμβάνουν μόνο καταναλωτές και παροχείς υπηρεσιών.

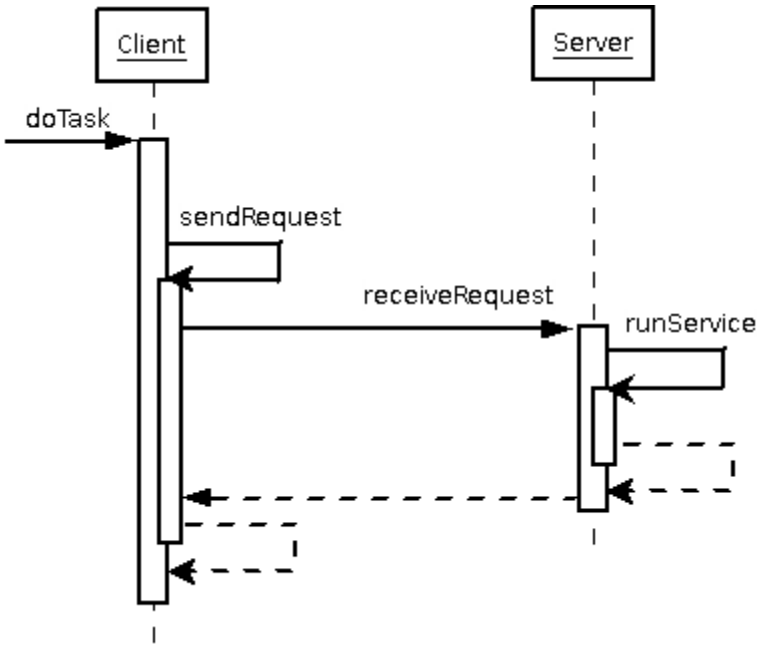
Η χρήση XML μπορεί να οδηγήσει σε ανταλλαγή τεράστιων όγκων δεδομένων.

4.4.2 Η αρχιτεκτονική πελάτη-εξυπηρετητή

Στην αρχιτεκτονική πελάτη-εξυπηρετητή το σύστημα δέχεται αιτήσεις για τις υπηρεσίες του από μια οντότητα εκτός συστήματος και προωθεί τις απαντήσεις στην οντότητα αυτή.



Κλάσεις Πελάτη-Εξυπηρετητή (προσαρμοσμένο από Buschmann et al., 1996, σ. 326)



Διάγραμμα Ακολουθίας Πελάτη-Εξυπηρετητή (προσαρμοσμένο από Buschmann et al., 1996, σ. 327)

Η αρχιτεκτονική πελάτη-εξυπηρετητή συναντάται σε:

Βάσεις δεδομένων

Παγκόσμιος Ιστός (*World Wide Web*)

Πληροφοριακά Συστήματα (*Information Systems*)

Παραλλαγές.

thin client, όπου ο πελάτης είναι υπεύθυνος μόνο για την διεπαφή με τον χρήστη.

thick client, όπου ο πελάτης είναι υπεύθυνος για την επιχειρησιακή λογική και για την διεπαφή με τον χρήστη, ενώ ο εξυπηρετητής είναι υπεύθυνος μόνο για την επεξεργασία δεδομένων.

Πλεονεκτήματα

Μεταφερσιμότητα (*portability*)

Απόδοση (*performance*)

Διαχειρισιμότητα (*ease of administration*)

Ικανότητα κλιμάκωσης (*scalability*)

Προσαρμοστικότητα στη διεπαφή με τον χρήστη (*user interface flexibility*)

Αξιοπιστία (*reliability*)

Μειονεκτήματα

Πιθανώς αυξημένες ανάγκες μεταφοράς δεδομένων μέσω δικτύου

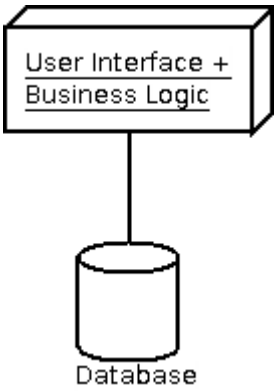
Προβλήματα λόγω καθυστερήσεων στην μεταφορά δεδομένων

Πιθανώς δύσκολο προγραμματιστικό μοντέλο

Πιθανώς να απαιτείται επικοινωνία μεταξύ ισότιμων μερών (*peer-to-peer*)

4.4.3 Αρχιτεκτονικές Πελάτη-Εξυπηρετητή Δύο Επιπέδων

Τα πληροροφιακά συστήματα μέχρι περίπου το 1990 δομούνταν σε δύο επίπεδα (*two-tier architecture*):



Two-tier architecture

Πλεονεκτήματα

Γρήγορη υλοποίηση.

Ανάπτυξη με χρήση ολοκληρωμένων εργαλείων που δίνονται από τον προμηθευτή.

Κατάλληλο για ομογενή περιβάλλοντα.

Μειονεκτήματα

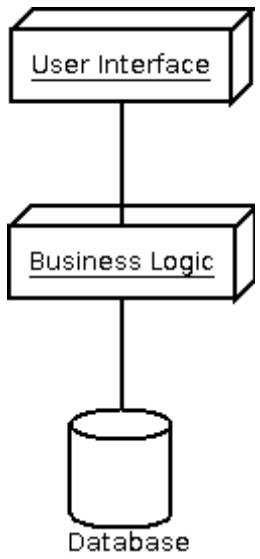
Δεν προσαρμόζεται σε ετερογενή περιβάλλοντα.

Οδηγεί σε κλειστές λύσεις, δεμένες με συγκεκριμένο προμηθευτή.

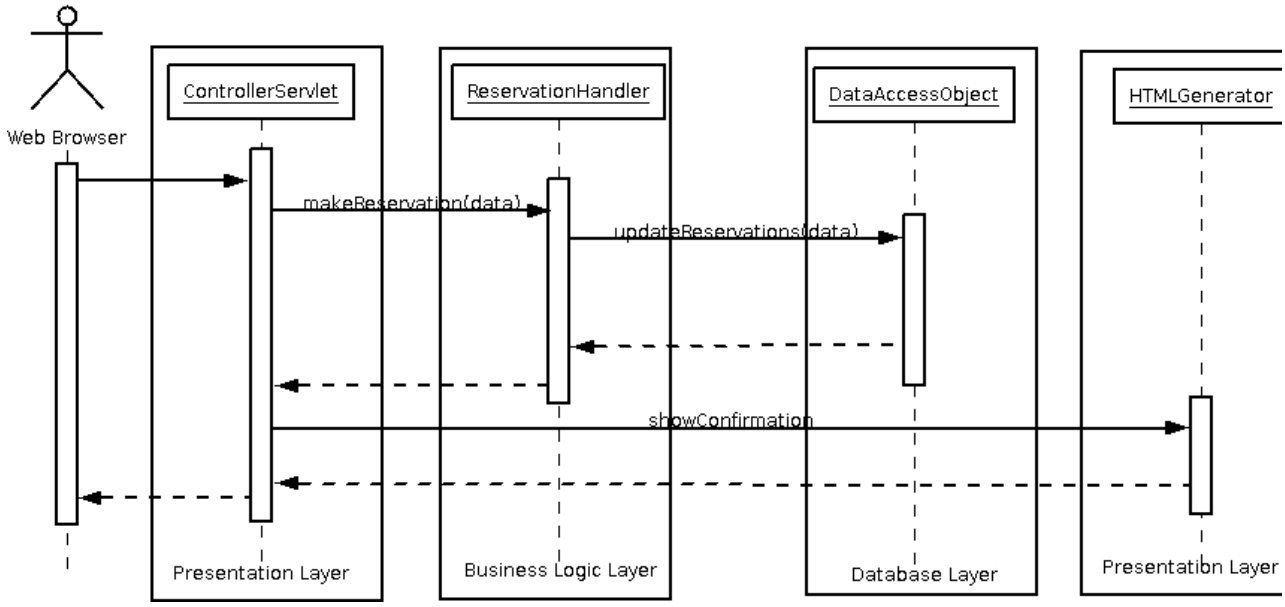
Δεν υπάρχει λόγος να επιβαρύνεται ο κάθε πελάτης με την επιχειρησιακή λογική, η οποία είναι ίδια για όλους και θα μπορούσε να εκτελείται από τον εξυπηρετητή.

4.4.4 Αρχιτεκτονικές Πελάτη-Εξυπηρετητή Τριών Επιπέδων

Σήμερα τις περισσότερες φορές τα πληροφοριακά συστήματα δομούνται σε τρία επίπεδα (*three-tier architecture*):



Three-tier architecture



Διάγραμμα ακολουθίας three-tier architecture

Παρουσίαση (*presentation layer*)

Επιχειρηματική λογική (*business logic*)

Βάση δεδομένων (*database*)

Με τη διαστρωμάτωση αυτή διαχωρίζεται η διεπαφή με τον χρήστη από τη λογική του συστήματος και αυτή με τη σειρά της από την απόθηκευση των δεδομένων.

Πλεονεκτήματα

Ανεξαρτησία διεπαφής από την υπόλοιπη εφαρμογή.

Κατάλληλο για ετερογενή περιβάλλοντα (μπορούν να υπάρχουν διαφορετικές διεπαφές για διαφορετικούς χρήστες).

Ελαχιστοποιούνται οι απαιτήσεις στον υπολογιστή του χρήστη, καθώς εκτελείται μόνο κώδικας διεπαφής.

Καλύτερη διαχείριση πόρων, καθώς μπορεί να καταμεριστεί και η πρόσβαση στα δεδομένα και η επιχειρησιακή λογική.

Μειονεκτήματα

Περίπλοκη υλοποίηση.

Δεν υπάρχει λόγος να επιβαρύνεται ο κάθε πελάτης με την επιχειρησιακή λογική, η οποία είναι ίδια για όλους και θα μπορούσε να εκτελείται από τον εξυπηρετητή.

Αυξημένες ανάγκες μεταφοράς δεδομένων στο δίκτυο.

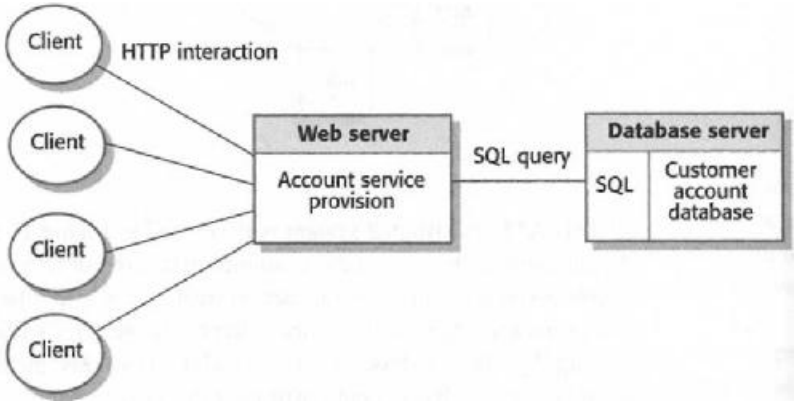


Figure 12.8 The distribution architecture of an Internet banking system

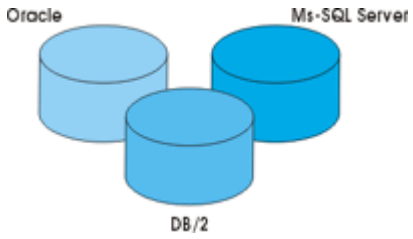
Στο παραπάνω σχήμα φαίνεται ένα παράδειγμα αρχιτεκτονικής τριών επιπέδων (διαδικτυακό σύστημα τράπεζας)

Ας περιγράψουμε τώρα πιο αναλυτικά τα τρια επίπεδα της 3-Tier αρχιτεκτονικής:

Πρώτο Επίπεδο (First Tier) - Database Server

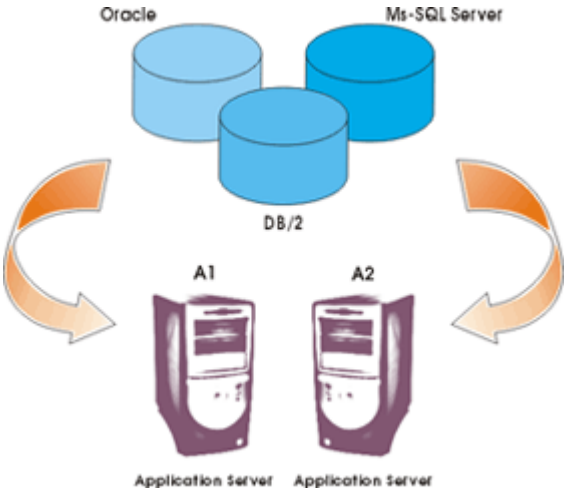
Αποτελώντας το βασικότερο επίπεδο του συστήματος, ο Database Server παρέχει όλες τις απαραίτητες λειτουργίες για την αποθήκευση, ανάκτηση, ενημέρωση και συντήρηση των

δεδομένων του συστήματος καθώς επίσης και όλους τους απαραίτητους μηχανισμούς για την ακεραιότητα των δεδομένων (Data Integrity).



Δεύτερο Επίπεδο (Second Tier) - Application Server

Αποτελεί το κύριο τμήμα του λογισμικού, στο οποίο εκτελούνται οι περισσότερες λειτουργίες, εκτός εκείνων που σχετίζονται με τη διαμόρφωση των οθονών εργασίας. Υπάρχει δυνατότητα εγκατάστασης περισσότερων του ενός Application Servers σε διαφορετικά μηχανήματα, αξιοποιώντας, με τον τρόπο αυτό, οποιαδήποτε διαθέσιμη υπολογιστική ισχύ και εξασφαλίζοντας εξαιρετικά αποτελέσματα ανταπόκρισης, αξιοπιστίας και επεκτασιμότητας.



Με την κατανομή των Application Servers σε ανεξάρτητα μηχανήματα, επιτυγχάνεται αποσυμφόρηση του συνολικού φόρτου του συστήματος, αφού κάθε Application Server είναι σε θέση να υποστηρίξει ένα υποσύνολο του συνολικού αριθμού των Remote Clients (π.χ. Ο Application Server Α θα εξυπηρετεί τους Clients του υποκαταστήματος Α, ενώ ο Application

Server Β θα εξυπηρετεί τους Clients του υποκαταστήματος

Τρίτο Επίπεδο (Third Tier) - Client

Το τρίτο επίπεδο του λογισμικού αποτελεί τη επαφή του χρήστη με το σύστημα (User Interface). Στο επίπεδο αυτό, πραγματοποιείται η διαχείριση των Οθονών Εργασίας (User Screens) καθώς επίσης και η μορφοποίηση των δεδομένων που εμφανίζονται. Η επικοινωνία του Client με τον Application ή τους Application Servers πραγματοποιείται κάνοντας χρήση ενός μόνο πακέτου δεδομένων κάθε φορά. Έτσι, επιτυγχάνεται ο βέλτιστος χρόνος απόκρισης μεταξύ του Client και του Application Server, δεδομένου ότι τα δυο αυτά επίπεδα μπορούν να λειτουργήσουν πάνω σε μια τηλεπικοινωνιακή γραμμή (Leased Line, Dialup, Internet Connection), εξασφαλίζοντας έτσι μικρούς χρόνους απόκρισης σε όλο το σύστημα.

Η αρχιτεκτονική Client - Server τριών επιπέδων (Three Tier) έχει διεθνώς αποδειχθεί ως η πλέον κατάλληλη για δικτυακές εγκαταστάσεις, σε αντίθεση με την αρχιτεκτονική Client - Server δύο επιπέδων (Two Tier), είτε Fat-Client, είτε Fat-Server.

Η συγκρότηση του συστήματος σε τρία επίπεδα εξασφαλίζει:

Την ελαχιστοποίηση της επιβάρυνσης του δικτύου λόγω μεταφοράς μεγάλου όγκου δεδομένων π.χ. η εκτέλεση ενός Query για την ανάκτηση μερικών εγγραφών από έναν πίνακα με δεκάδες χιλιάδες εγγραφές γίνεται στο διακομιστή εφαρμογής (Application Server), από τον οποίο μεταφέρεται στο χρήστη μόνο το αποτέλεσμα

Τη δυνατότητα διαχωρισμού του διακομιστή δεδομένων (Database Server) από το διακομιστή ή τούς διακομιστές εφαρμογής (Application Servers), ώστε να εκτελούνται σε διαφορετικά μηχανήματα. Κατά συνέπεια, ο καθορισμός

των κρίσιμων μεγεθών απόδοσης των αντίστοιχων μηχανών (sizing) μπορεί να γίνεται ανεξάρτητα, ενώ παράλληλα εξασφαλίζεται απεριόριστη επεκτασιμότητα, χωρίς ανακατασκευή, του λογισμικού

Τη μέγιστη ευελιξία στην επιλογή του διακομιστή δεδομένων, καθώς επιτρέπεται η χρήση οποιουδήποτε μηχανήματος με οποιοδήποτε λειτουργικό σύστημα (π.χ. Windows NT ή UNIX etc), με μοναδική απαίτηση τη δυνατότητα επικοινωνίας δια μέσου TCP/IP πρωτοκόλλου. Έτσι, είναι δυνατή η μεταγενέστερη αναβάθμιση ως προς τη βάση δεδομένων με την αλλαγή / αναβάθμιση του μηχανήματος, χωρίς να επηρεάζεται το υπόλοιπο σύστημα.

4.4.5 Υπηρεσίες ιστού

Οι υπηρεσίες ιστού είναι αρχιτεκτονικές πελάτη-εξυπηρετητή στις οποίες:

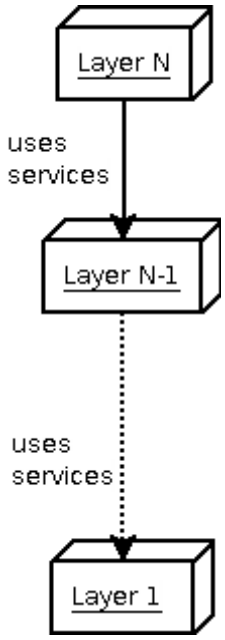
Όλα τα δεδομένα ανταλλάσσονται με τη μορφή XML (SOAP, REST).

Για τη μεταφορά των δεδομένων χρησιμοποιούνται πρωτόκολλα του διαδικτύου όπως το HTTP.

Η διεπαφή μεταξύ πελάτη-εξυπηρετητή περιγράφεται συνήθως σε διάλεκτο της XML (WSDL).

Χρησιμοποιούνται από εταιρείες για την επικοινωνία μεταξύ ετερογενών συστημάτων, όπως και για την επικοινωνία με εξωτερικούς εταίρους (π.χ., Google, Yahoo, Amazon).

Το αρχιτεκτονικό πρότυπο της διαστρωμάτωσης (*layers*, Buschmann et al., 1996, σ. 31--51) είναι κατάλληλο για τη δόμηση εφαρμογών που μπορούν να αναλυθούν σε ομάδες λειτουργιών όπου η κάθε ομάδα βρίσκεται σε ένα συγκεκριμένο επίπεδο αφαίρεσης.



Three-tier architecture

Το κάθε επίπεδο:

Προσφέρει υπηρεσίες στο αμέσως ανώτερο επίπεδο.

Χρησιμοποιεί τις υπηρεσίες του αμέσως χαμηλότερου επιπέδου.

Πλεονεκτήματα

Επαναχρησιμοποίηση επιπέδων.

Υποστήριξη τυποποίησης.

Περιορισμός των εξαρτήσεων.

Ανταλλαξιμότητα.

Μειονεκτήματα

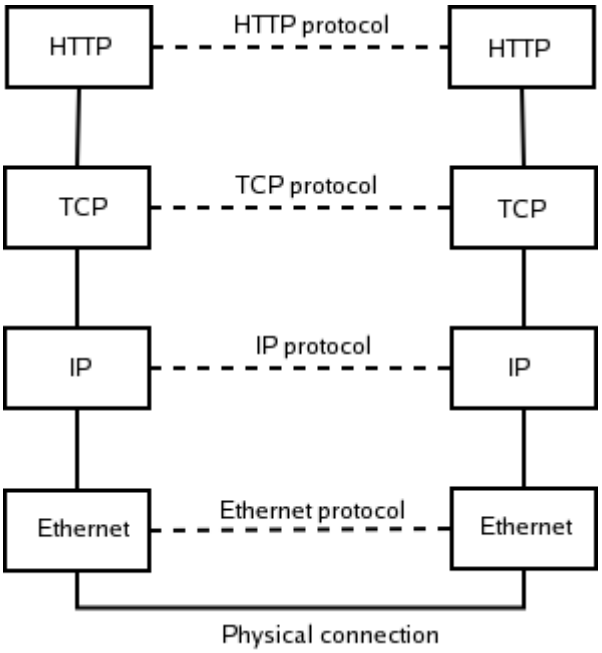
Μετάδοση αλλαγών.

Χαμηλότερη απόδοση.

Εκτέλεση περιττών εργασιών σε κάποιο επίπεδο.

Δυσκολία στον εντοπισμό των σωστών επιπέδων.

Το διαδίκτυο ακολουθεί την ακόλουθη διαστρωμάτωση:



Τα επίπεδα του διαδικτύου

5. Μελέτη περίπτωσης (Case Study)

5.1 Ιστορικό μελέτης περίπτωσης

Η εγγραφή των φοιτητών στις εργαστηριακές ομάδες των μαθημάτων στο τμήμα Εφαρμογών Πληροφορικής στην Διοίκηση και Οικονομία στο ΤΕΙ Αμαλιάδας γίνεται αυτήν την περίοδο με το χέρι. Οι σπουδαστές συμπληρώνουν το ονοματεπώνυμό τους στην έντυπη κατάσταση της αντίστοιχης ομάδας και οι λίστες μετά στέλνονται στην γραμματεία του ΤΕΙ Οι υπάλληλοι εισάγουν έπειτα τις επιλογές σε μια βάση δεδομένων και έτσι καταρτίζονται οι διάφορες εργαστηριακές ομάδες, λίστες με τα ονόματα των σπουδαστών που παρακολουθούν ένα εργαστήριο, δημιουργούνται νέες εργαστηριακές ομάδες αν απαιτείται κτλ. Η διαδικασία εγγραφής ολοκληρώνεται μέσα σε δυο έως τρεις εβδομάδες. Το πανεπιστήμιο αποφάσισε να ερευνήσει τη χρήση ενός on-line συστήματος εγγραφής. Αυτό το σύστημα θα χρησιμοποιείται από τους καθηγητές για να δείξουν τις εργαστηριακές ομάδες που διδάσκουν, από τους σπουδαστές για να επιλεχτούν οι εργαστηριακές ομάδες, και από τη γραμματεία για να διαχειριστεί τις δηλώσεις και να ολοκληρωθεί η διαδικασία εγγραφής.

5.2 Δήλωση του προβλήματος on-line δήλωση εργαστηρίων

Στην αρχή κάθε εξαμήνου οι σπουδαστές μπορούν να ζητήσουν έναν κατάλογο σειράς μαθημάτων που περιέχει τα μαθήματα που αντιστοιχούν σε κάθε εξάμηνο και τις αντίστοιχες εργαστηριακές ομάδες που προσφέρονται από τους καθηγητές ώστε να προχωρήσει στην δήλωσή τους. Για κάθε εργαστηριακή ομάδα χρειάζονται και συγκεκριμένες πληροφορίες ώστε να μπορέσουν να λάβουν σωστές. Μέσω του νέου on-line συστήματος θα μπορούν οι σπουδαστές να βλέπουν όλες τις αντίστοιχες πληροφορίες για το κάθε μάθημα. Μόλις ολοκληρωθεί η διαδικασία δήλωσης για έναν σπουδαστή, το σύστημα στέλνει τις πληροφορίες πίσω στην βάση δεδομένων και η δήλωση του φοιτητή αποθηκεύεται και είναι μοναδική. Οι καθηγητές μπορούν να έχουν πρόσβαση στην εφαρμογή ως διαχειριστές και να τροποποιήσουν κάποια εργαστηριακή ομάδα ή να προσθέσουν νέα αν προσφέρεται. Θα μπορούν επίσης να δουν λίστα με τους φοιτητές που επέλεξαν τις εργαστηριακές τους

ομάδες. Ο διαχειριστής θα μπορεί να έχει πρόσβαση στην εφαρμογή για να τροποποιήσει κάποια λανθασμένη δήλωση φοιτητή ή καθηγητή αλλά αυτό θα μπορεί όμως να γίνει εντός περιορισμένου χρονικού διαστήματος, ώστε να ολοκληρώνεται σύντομα η διαδικασία.

5.3 Ο ρόλος των εργαλείων

Οποιαδήποτε μέθοδος ανάπτυξης λογισμικού υποστηρίζεται καλύτερα από ένα εργαλείο. Στην περίπτωσή μας χρησιμοποιήθηκε το visual studio 2010. Μέσω αυτού χαρτογραφήθηκαν τα επιμέρους βήματα της διαδικασίας, με την βοήθεια διαγραμμάτων περίπτωσης, διαγραμμάτων ροής και ακολουθίας. Επιπλέον προσδιορίστηκαν οι οντότητες τους συστήματος και η βάση δεδομένων.

5.4 Σύνοψη του Έργου

Το έργο υλοποιήθηκε με την βοήθεια της διαδικασίας RUP (Rational Unified Process), μια μίξη των γενικών μοντέλων και των μοντέλων επανάληψης. Κατά την αρχική φάση (Inception phase) ξεκίνησε η σχεδίαση των διαγραμμάτων περιπτώσεων χρήσης και ωρίμασαν κατά την φάση της υλοποίησης (elaboration phase). Σε όλες τις φάσεις του κύκλου ζωής του έργου χρησιμοποιήθηκαν τα συστατικά της διαδικασίας ανάλυση απαιτήσεων, σχεδιασμός ,υλοποίηση και έλεγχος.

5.5 Αρχική Φάση (Inception phase)

Το πρώτο θέμα που εξετάζεται είναι η ανάγκη για ένα νέο σύστημα εγγραφής σε εργαστήρια. Έχει το ΤΕΙ τους πόρους που απαιτούνται για να σχεδιαστεί και να εφαρμοστεί το νέο σύστημα; Εκτός από την αξιολόγηση της ανάγκης για το σύστημα, οι κίνδυνοι που τίθενται από το νέο σύστημα προσδιορίζονται. Στην περίπτωση ενός on-line συστήματος εγγραφής σε εργαστηριακές ομάδες, ένας από τους σημαντικότερους κινδύνους είναι η

δυνατότητα να αποθηκευτούν οι πληροφορίες με έναν τρόπο που είναι ασφαλής και γρήγορα προσιτός από όλους. Για τους σκοπούς αυτής της περιπτωσιολογικής μελέτης αποφασίστηκε ότι το νέο σύστημα πρέπει να χτιστεί. Συμπληρώθηκαν πρωτότυπα να εξεταστούν οι κίνδυνοι της βάσης δεδομένων.

5.6 Καθορισμός των ρόλων

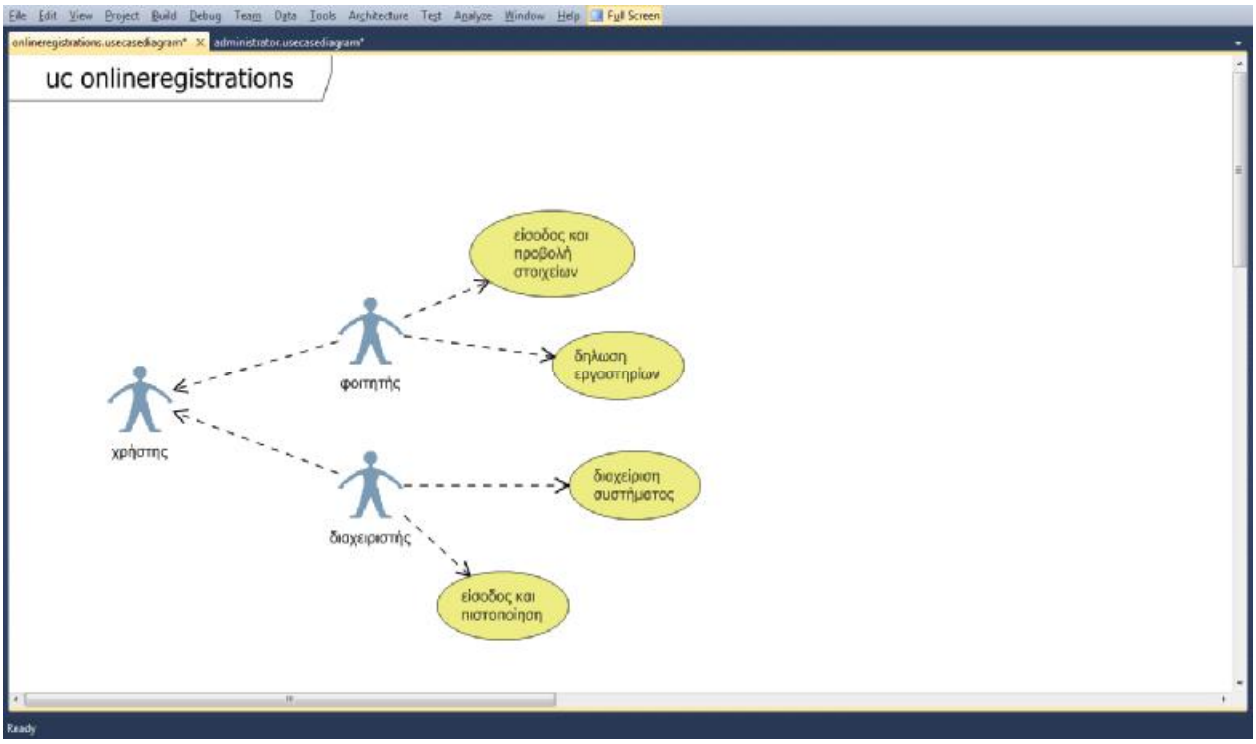
Καθορίστηκαν οι παρακάτω ρόλοι για το πρόβλημα:

1. Φοιτητής
2. Διαχειριστής

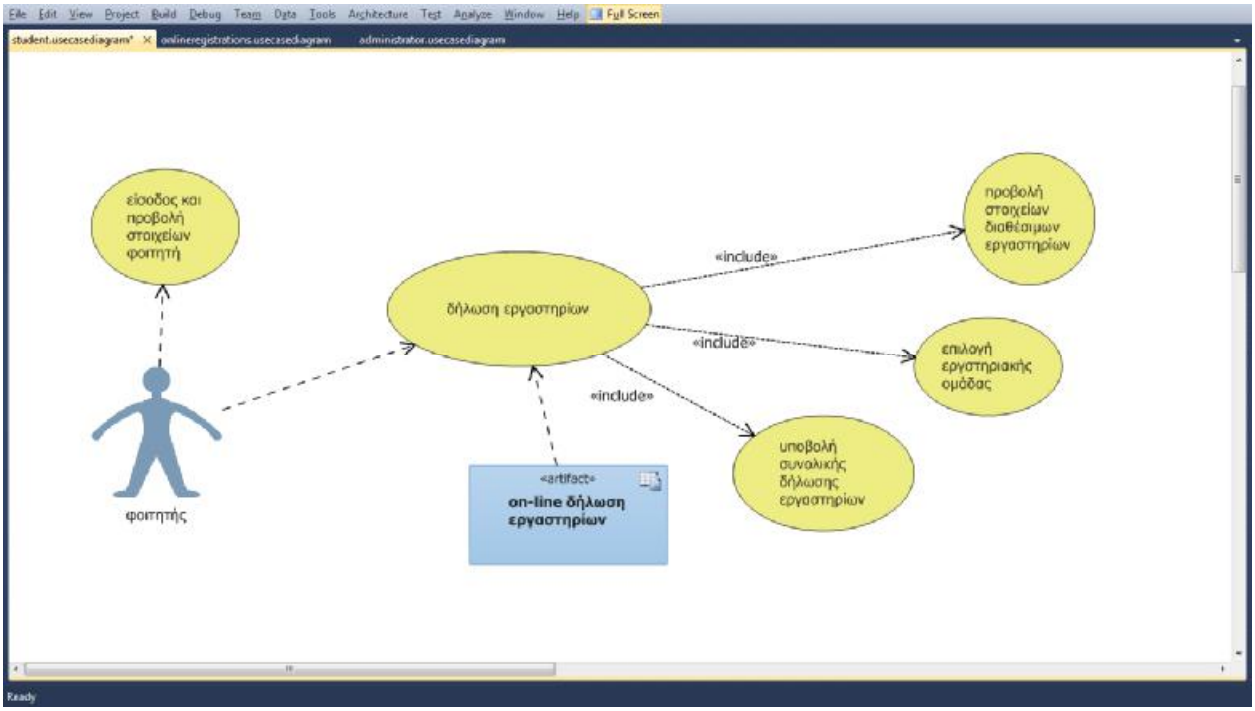
Για λόγους ευκολίας στην σχεδίαση και υλοποίηση της εφαρμογής οι καθηγητές δεν ορίζονται σαν ξεχωριστός ρόλος στο σύστημα, αλλά κάνουμε την παραδοχή ότι μπορούν να ικανοποιούν τις ανάγκες τους μέσω των διαχειριστών ή και σαν διαχειριστές εφόσον έχουν κωδικό και passwords διαχειριστή.

5.6.1 Σχεδιάζοντας τα use case diagrams

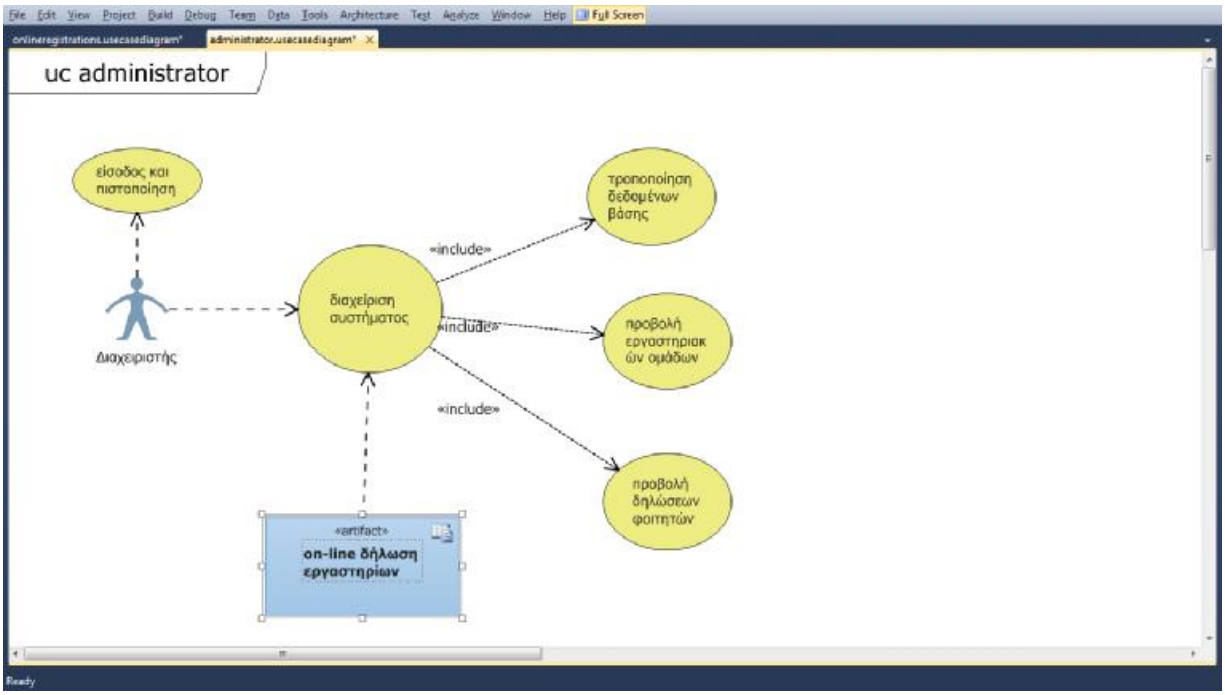
Τα διαγράμματα περιπτώσεων χρήσης φαίνονται παρακάτω (Εικόνες 1,2 &3). Διακρίνουμε τους ρόλους (φοιτητής, διαχειριστής) και τις χρήσεις (βλέπε οβάλ σχήματα).



Εικόνα 1



Εικόνα 2



Εικόνα 3

Για κάθε περίπτωση χρήσης συντάσσεται μια σύντομη περιγραφή, η οποία τοποθετείται στο πεδίο τεκμηρίωση προδιαγραφής για την περίπτωση χρήσης. Οι περιγραφές αυτές έχουν ως εξής:

- Δήλωση εργαστηρίου: Αυτή η περίπτωση χρήσης ξεκινά από τον φοιτητή. Παρέχει την δυνατότητα προσθήκης, διαγραφής, προβολής και υποβολής των επιλεγμένων εργαστηριακών ομάδων στον φοιτητή και όλες οι υποβαλλόμενες πληροφορίες επιστρέφουν στην βάση δεδομένων του ΤΕΙ.

- Διαχείριση εφαρμογής: Αυτή η περίπτωση χρήσης ξεκινά από τον διαχειριστή. Παρέχει την δυνατότητα τροποποίησης των δεδομένων εργαστηριακών ομάδων, φοιτητών, εξαμήνων και δηλώσεων. Επιπλέον παρέχει την δυνατότητα προβολής των δηλώσεων των φοιτητών ή και των εργαστηριακών ομάδων.

5.7 Φάση Υλοποίησης

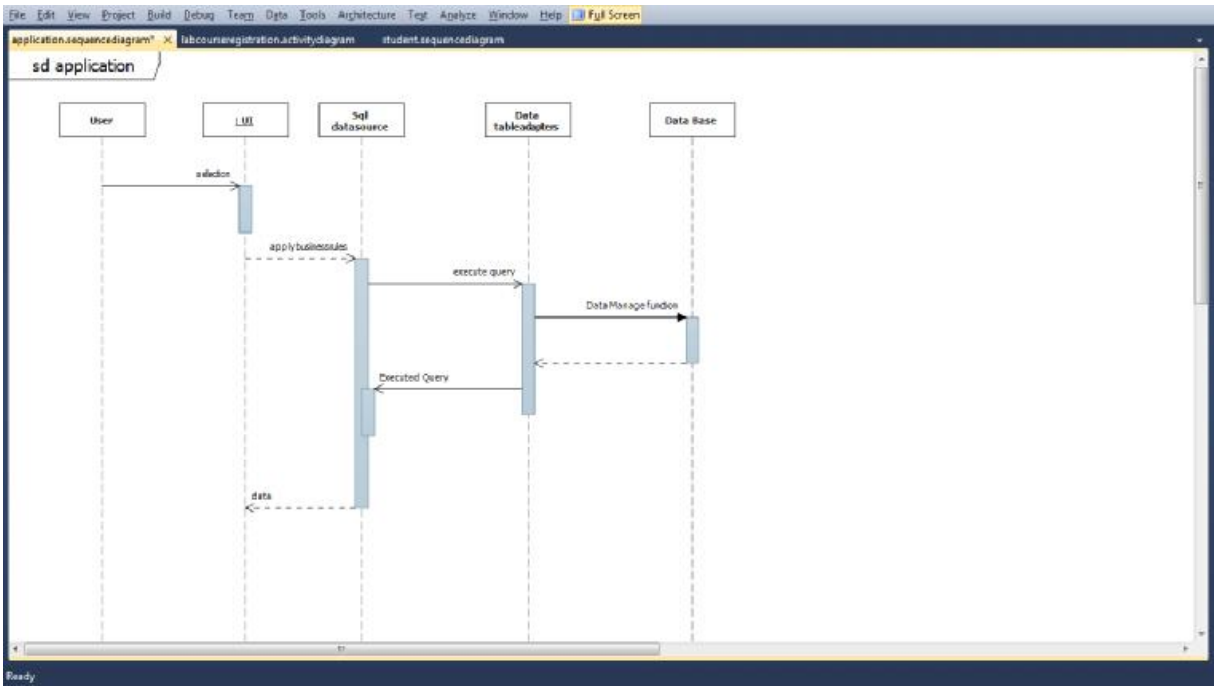
Κατά τη διάρκεια της υλοποίησης, εφαρμόζονται μερικές από τις σημαντικότερες και κρίσιμες περιπτώσεις χρήσης . Κατά τη διάρκεια αυτής της φάσης, γίνεται εστίαση στην σωστή δομή και αρχιτεκτονική.

5.7.1 Ανάπτυξη των σεναρίων

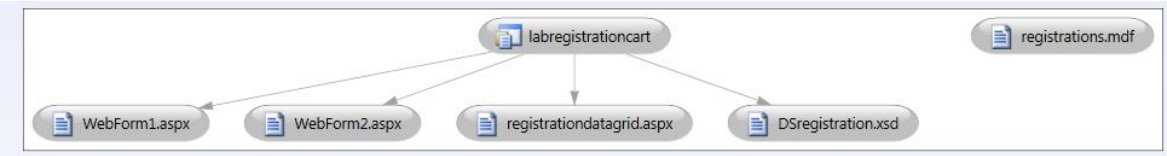
Κάθε περίπτωση χρήσης είναι ένας Ιστός των σεναρίων. Τα σενάρια είναι τεκμηριωμένα χρησιμοποιώντας τα διαγράμματα ακολουθίας. Τα αντικείμενα αντιπροσωπεύονται ως κάθετες γραμμές και τα μηνύματα μεταξύ των αντικειμένων παρουσιάζονται ως απευθείας οριζόντιες γραμμές.

Στην υλοποίηση της εφαρμογής ακολουθήθηκε η φιλοσοφία που φαίνεται στο διάγραμμα ακολουθίας στην Εικόνα 4.

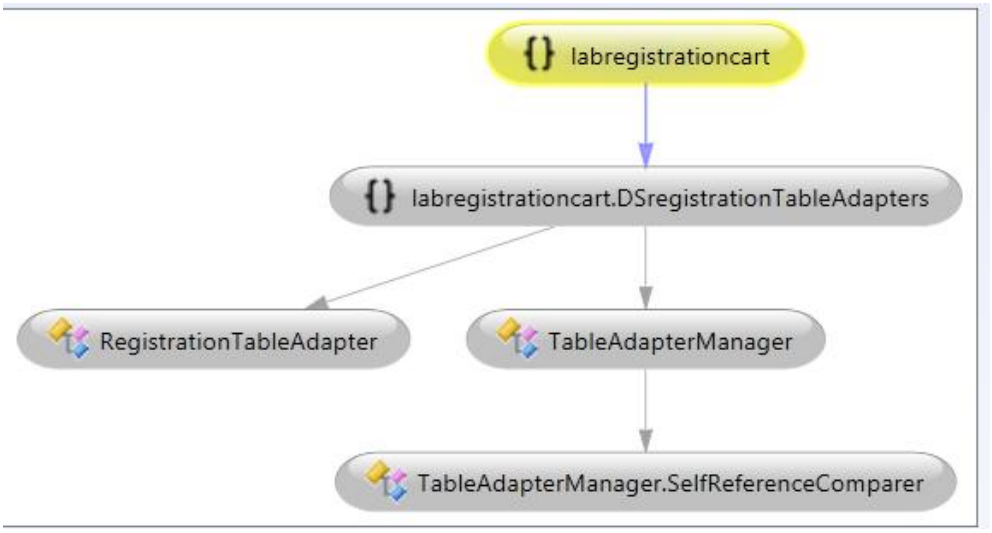
Παρακάτω στις εικόνες 5,6 &7 δίνονται οι αρχιτεκτονικοί σχεδιασμοί σε επίπεδο φορμών και κλάσεων που ακολουθήθηκαν.



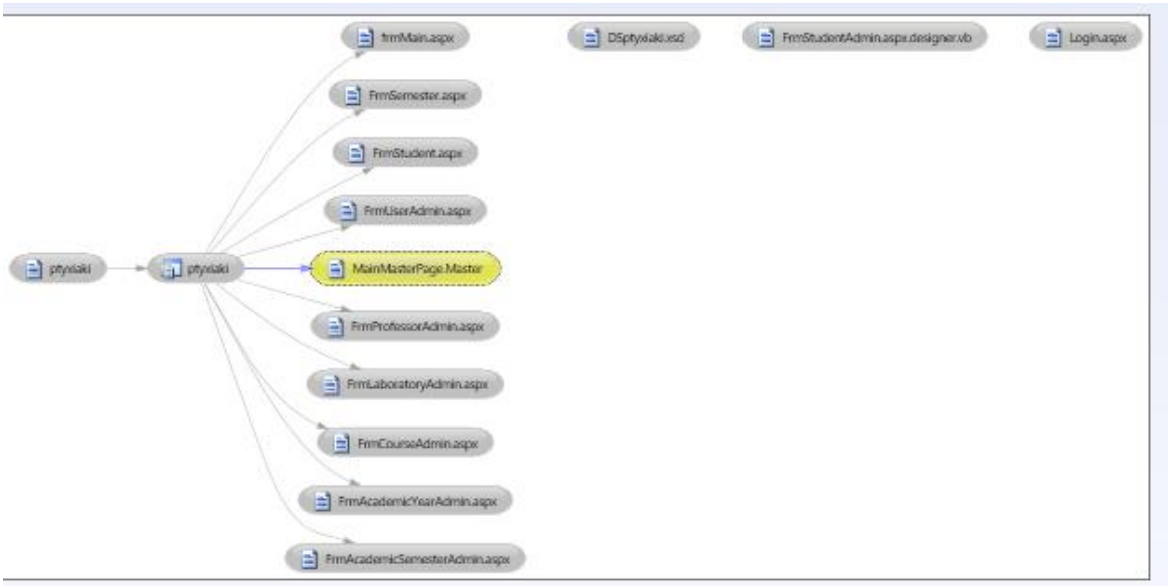
Εικόνα 4



Εικόνα 5



Εικόνα 6



Εικόνα 7

5.8 Περιγραφή της Βάσης Δεδομένων της εφαρμογής

5.8.1 Γενικά

Η βάση δεδομένων της εφαρμογής σχεδιάστηκε στο visual studio σε SQL και ονομάζεται registrations.db και σχεδιάστηκε με βάση ορισμένους κανόνες και τροποποιήθηκε στην πορεία ανάλογα με τις ανάγκες υλοποίησης της εφαρμογής.

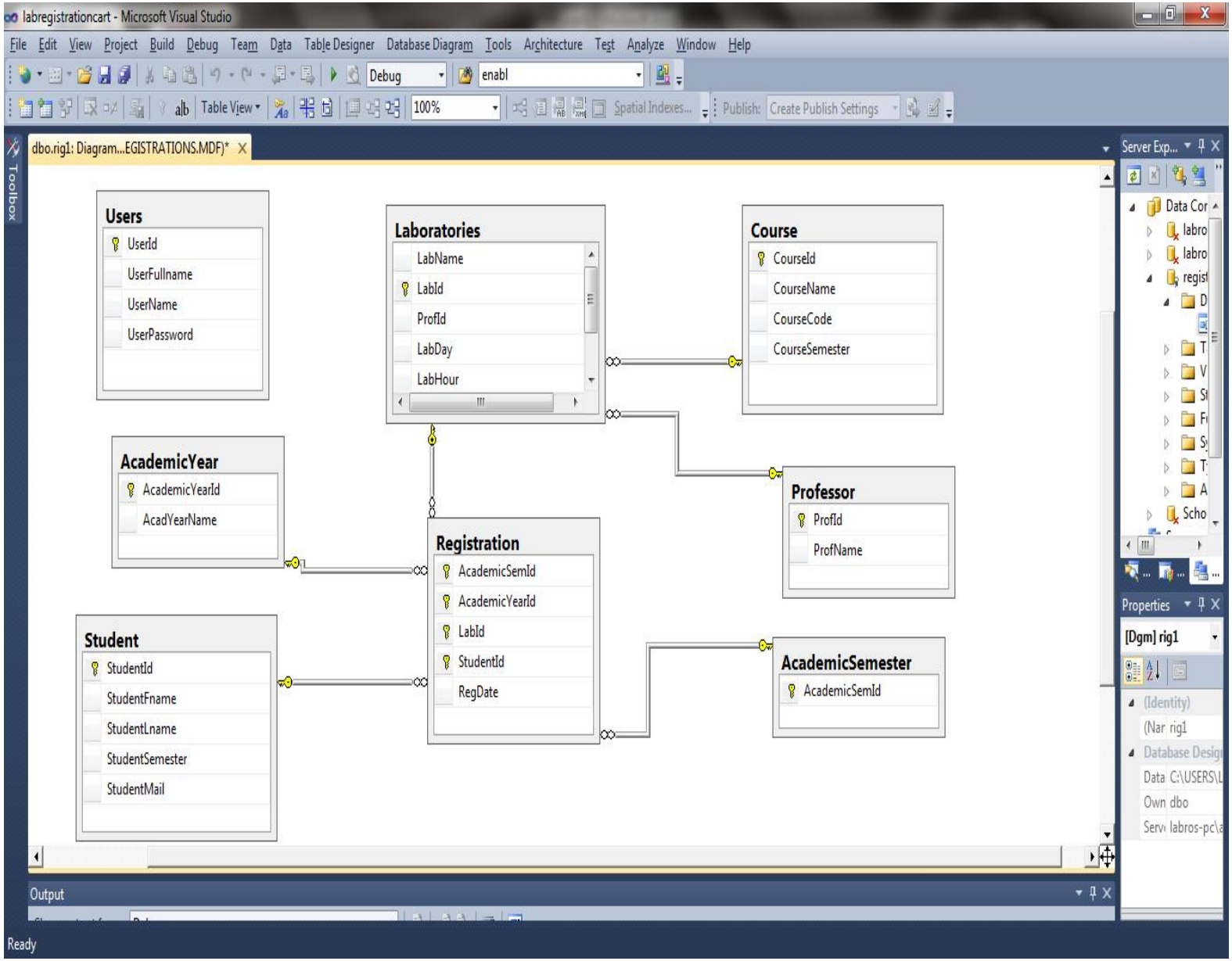
Έτσι αρχικά σχεδιάστηκαν όλοι οι απαραίτητοι πίνακες με τα αντίστοιχα πεδία τους:

1. Πίνακας AcademicYear (Ακαδημαϊκό Έτος)
2. Πίνακας AcademicSemester (Ακαδημαϊκό Εξάμηνο)
3. Πίνακας Professor (Καθηγητής)
4. Πίνακας Student (Φοιτητής)
5. Πίνακας Course (Μάθημα)
6. Πίνακας Laboratories (Εργαστήρια)
7. Πίνακας Registration (Δηλώσεις)
8. Πίνακας Users (Χρήστες)

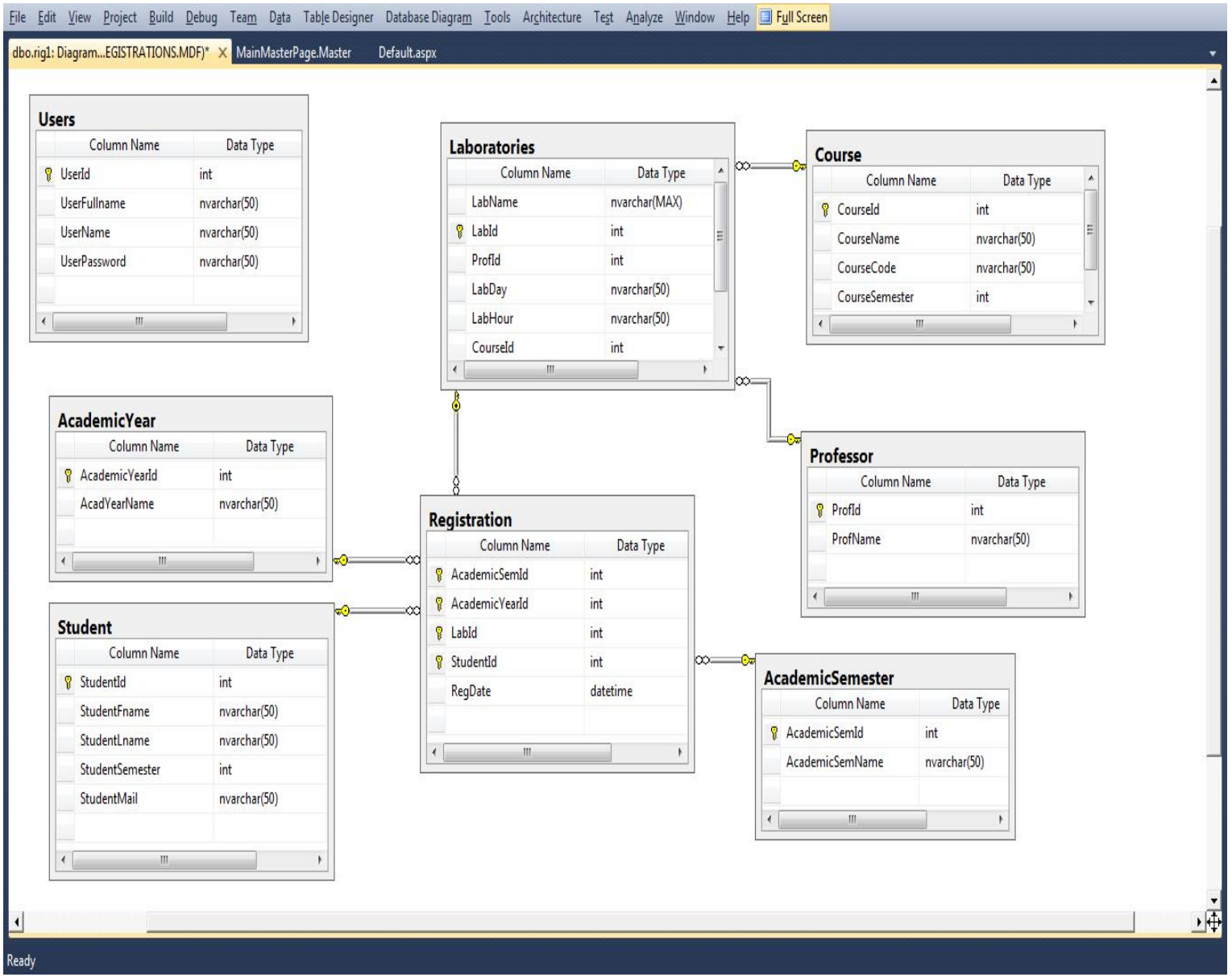
Σημειώνεται ότι στον πίνακα Registration καταχωρούνται οι δηλώσεις των φοιτητών, για αυτό άλλωστε περιλαμβάνει συσχετίσεις με τα primary keys όλων των πινάκων. (βλέπε και σχήμα βάσης δεδομένων). Ο πίνακας users σχεδιάστηκε για να αποθηκεύονται τα ονόματα χρηστών και οι κωδικοί, ώστε να γίνεται ταυτοποίηση χρηστών (login).

Στην Εικόνα 1 παρακάτω φαίνεται η βάση δεδομένων και οι σχέσεις μεταξύ των πινάκων, ενώ στην Εικόνα 2 φαίνεται η βάση με τους πίνακες σε γενικευμένη προβολή με τους τύπου των πεδίων τους.

Στην συνέχεια θα προχωρήσουμε σε αναλυτική παρουσίαση του κάθε πίνακα της βάσης δεδομένων.



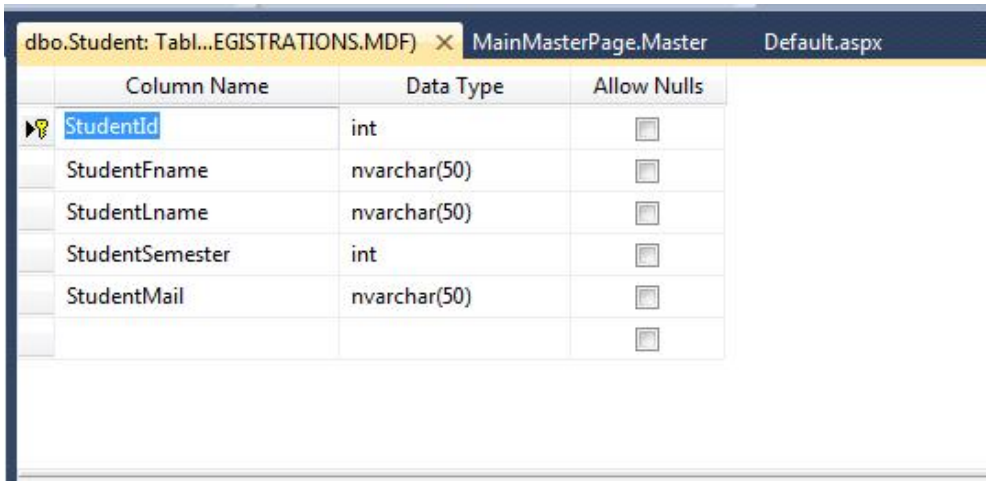
Εικόνα 8 Το Σχήμα της Βάσης Δεδομένων (απλές προβολές πινάκων)



Εικόνα 9 Το Σχήμα της Βάσης Δεδομένων (γενικευμένες προβολές πινάκων)

5.8.2 Περιγραφή Πινάκων

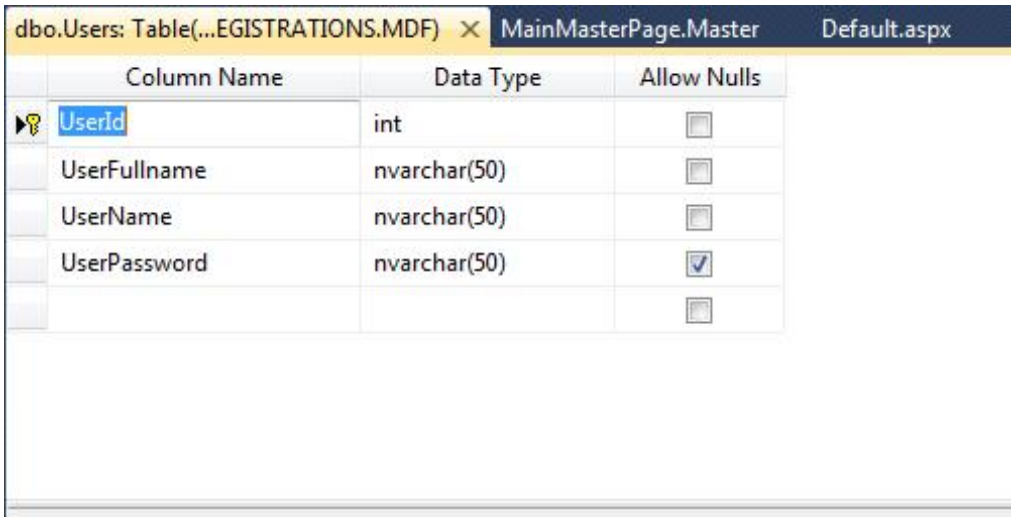
i. Πίνακας Φοιτητής (Εικόνα 3)



Εικόνα 10

Είναι αποθηκευμένα τα στοιχεία των φοιτητών (Αριθμός Μητρώου, Όνομα, Επίθετο, Ακαδημαϊκό Εξάμηνο και e-mail επικοινωνίας. Η μοναδικότητα του Α.Μ μας επιτρέπει και μας διευκολύνει να το θέσουμε σαν πρωτεύον κλειδί στον πίνακα.

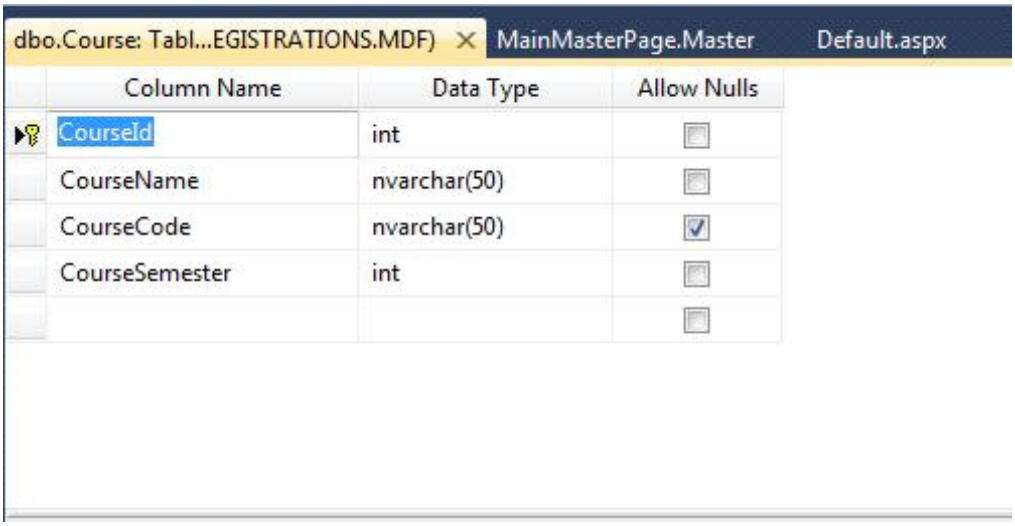
ii. Πίνακας Χρήστες (Εικόνα 4)



Εικόνα 11

Είναι αποθηκευμένα ονόματα χρηστών και κωδικοί για να γίνεται ταυτοποίηση κατά την είσοδό τους στην εφαρμογή. Δεν συνδέεται με σχέση με κάποιον πίνακα την βάσης και δημιουργήθηκε βοηθητικά.

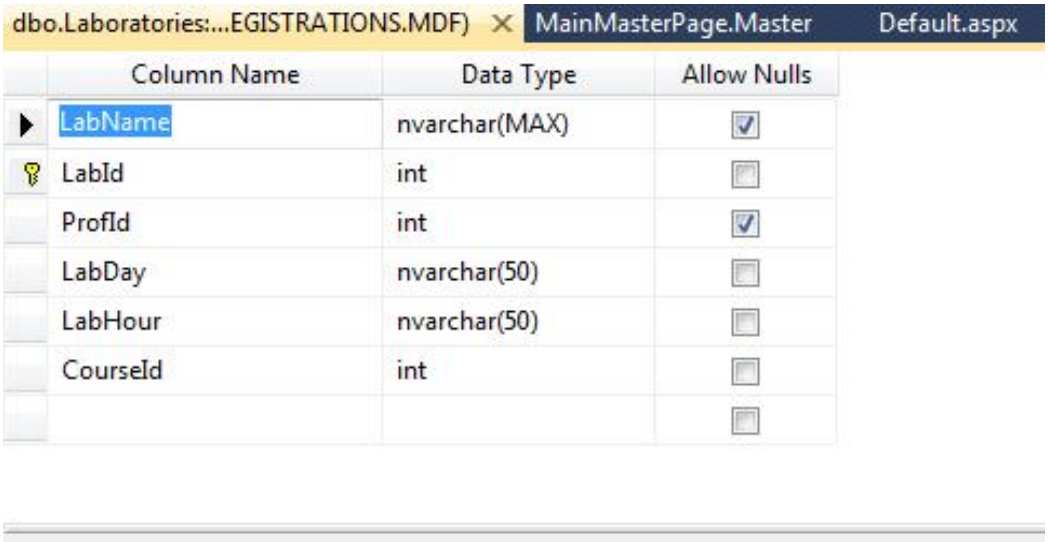
iii. Πίνακας Μαθήματα (Εικόνα 5)



Εικόνα 12

Είναι αποθηκευμένα τα στοιχεία των μαθημάτων (Όνομα, Κωδικός,Εξάμηνο) και έχει δημιουργηθεί ένα πεδίο μοναδικής καταγραφής ως πρωτεύον κλειδί (CourseId) .

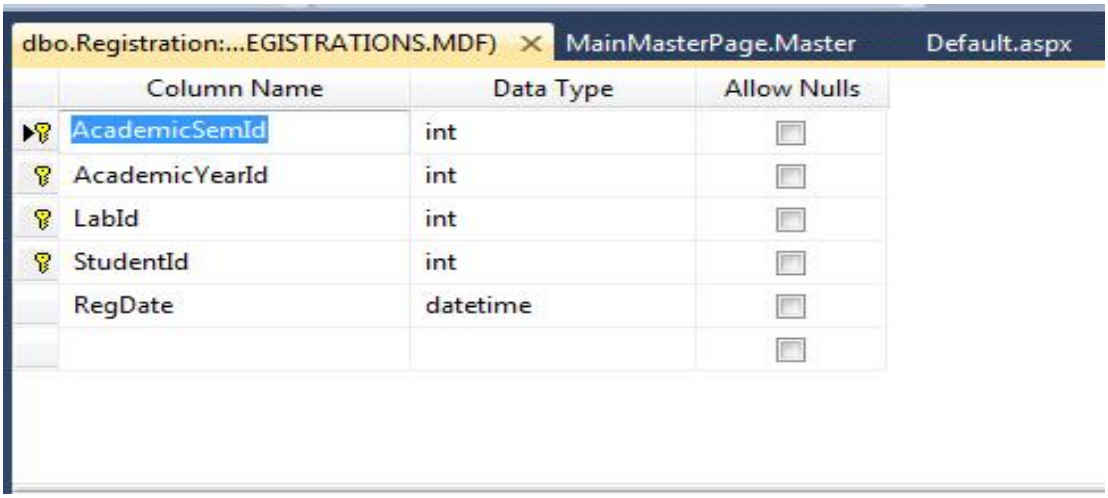
iv. Πίνακας Εργαστήρια (Εικόνα 6)



Εικόνα 13

Είναι αποθηκευμένα τα στοιχεία (Όνομα, Ημέρα, Ώρα)των εργαστηριακών ομάδων που αφορούν σε κάθε μάθημα , οπότε και συνδέεται με τον πίνακα Μάθημα. Τα πεδία ProfId και CourseId δημιουργήθηκαν για να επιτευχθεί η σύνδεση με τους πίνακες Καθηγητής και Μάθημα αντίστοιχα.

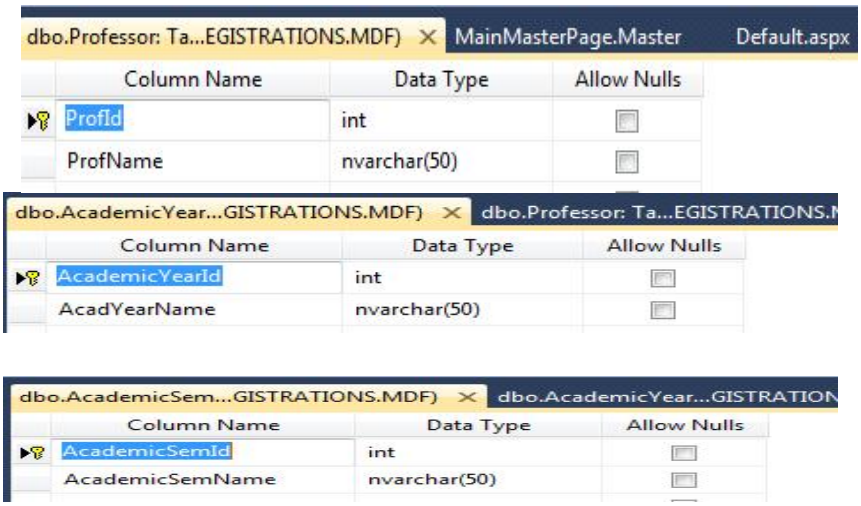
v. Πίνακας Δηλώσεις (Εικόνα 7)



Εικόνα 14

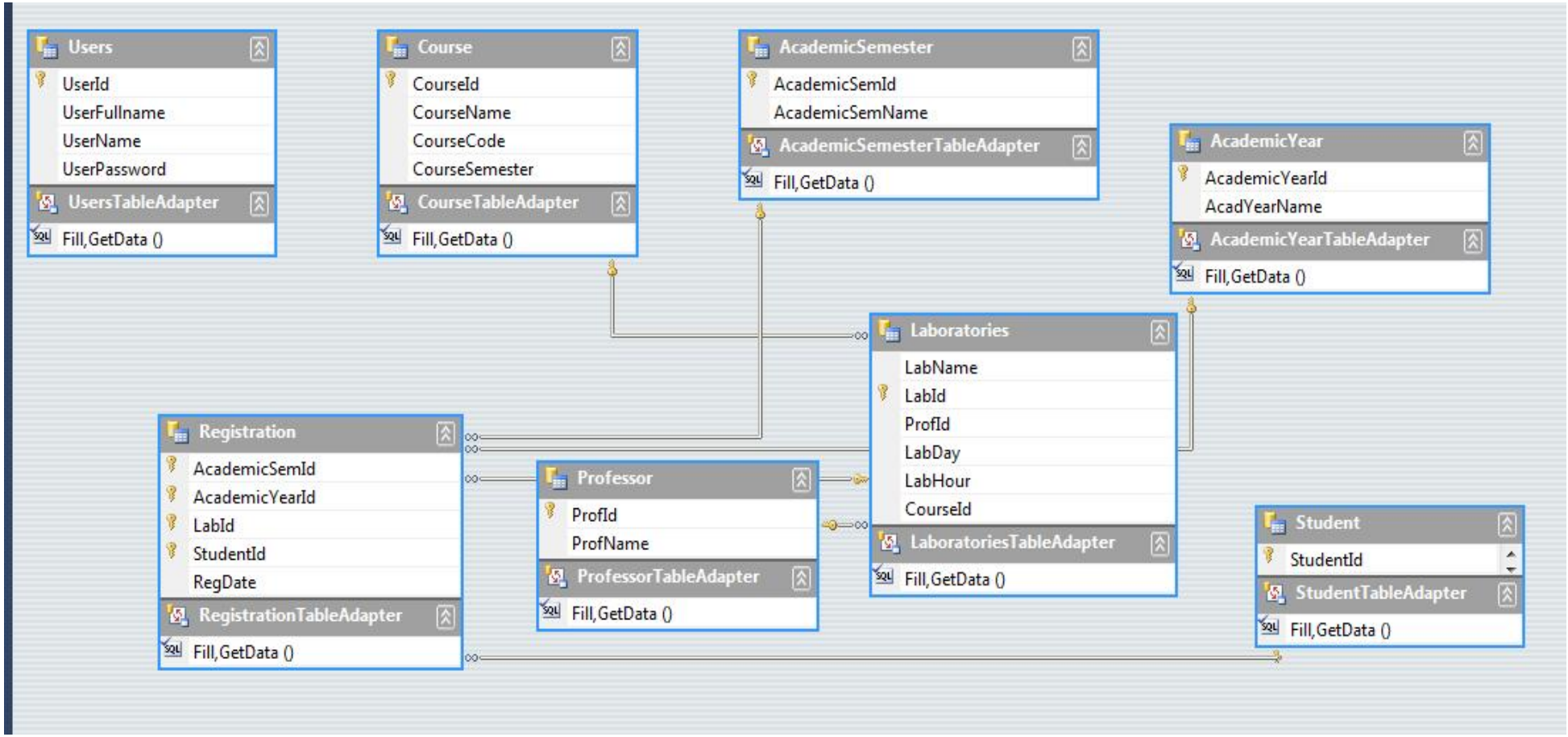
Περιέχει τα στοιχεία της κάθε δήλωσης, οπότε συνδέεται με όλα τα πρωτεύοντα κλειδιά των πινάκων Φοιτητής, Εργαστήριο, Έτος και Εξάμηνο. Η ημερομηνία καταχωρείται την στιγμή που υποβάλλεται η δήλωση του φοιτητή.

vi. Πίνακες Καθηγητής, Ακαδημαϊκό Ατός και Ακαδημαϊκό Εξάμηνο (Εικόνα 8)



Εικόνα 8

Αποθηκεύονται αντίστοιχα τα δεδομένα των οντοτήτων καθηγητής, Ακαδημαϊκό Έτος και Εξάμηνο.



Σχήμα 2 : Σχέδιο της Βάσης όπου φαίνονται οι προσαρμογείς των πινάκων

6. Τεχνικά εγχειρίδια και εγχειρίδια χρήσης εφαρμογής

6.1 Έντυπο καταγραφής απαιτήσεων

για την

«Υλοποίηση Εφαρμογής On-line δήλωσης εργαστηριακών μαθημάτων στο ΑΤΕΙ Αμαλιάδας»

Έκδοση 2.0

Συντάκτες: Ρήγας Λάμπρος – Παναγιωτόπουλος Ανδρέας

Τμήμα Εφαρμογών Πληροφορικής στην Διοίκηση και Οικονομία

ΑΤΕΙ Αμαλιάδας

15 Οκτ. 2010

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

6.1.1 Εισαγωγή

Στόχος

Σε αυτό το έγγραφο αναλύονται οι απαιτήσεις προδιαγραφών για την υλοποίηση της εφαρμογής On-line δήλωσης εργαστηριακών μαθημάτων του τμήματος Εφαρμογών Πληροφορικής στην Διοίκηση και Οικονομία του ΑΤΕΙ Αμαλιάδας.

Κοινό που απευθύνεται και τρόπος ανάγνωσης

Το παρόν έγγραφο απευθύνεται στους δημιουργούς, διαχειριστές και χρήστες της εφαρμογής. Στη συνέχεια ακολουθεί μια γενική περιγραφή της εφαρμογής, αναλύονται τα χαρακτηριστικά του συστήματος και των επιμέρους υποσυστημάτων καθώς επίσης και οι λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος. Με αυτόν τον τρόπο ο αναγνώστης αποκτά ολοκληρωμένη εικόνα του συνόλου των προδιαγραφών.

Σκοπός της Εφαρμογής

Στόχος της εφαρμογής είναι η δημιουργία μιας εφαρμογής με την βοήθεια της οποίας θα δίνεται η δυνατότητα αφενός στον φοιτητή να επιλέγει ηλεκτρονικά τα εργαστηριακά μαθήματα του εξαμήνου του και αφετέρου στον διαχειριστή να διαχειρίζεται τις εργαστηριακές ομάδες που προσφέρονται . Έτσι οι εγγραφές θα γίνονται εύκολα και γρήγορα χωρίς να ταλαιπωρούνται τόσο οι φοιτητές όσο και το προσωπικό του ΑΤΕΙ που επεξεργάζεται τις δηλώσεις. Επιπλέον μετριάζονται τα σφάλματα και η όλη διαδικασία της δήλωσης εργαστηρίων απλουστεύεται.

Παραπομπές

Το παρόν έγγραφο συντάχθηκε με βάση το πρότυπο έγγραφο Software Requirements Specification **(www.processimpact.com)** και τις κατευθύνσεις του συγγράμματος Software Engineering 8-I.Sommerville(Chapter6,par.5 software requirements document).

6.1.2 Γενική Περιγραφή

Προοπτική του Προϊόντος

Η εφαρμογή που περιγράφεται σε αυτό το έγγραφο υλοποιήθηκε στα πλαίσια εκπόνησης πτυχιακής εργασίας των συντακτών και αποτελεί νέο και ανεξάρτητο προϊόν.

Χαρακτηριστικά του Προϊόντος

Μέσω της παρούσας εφαρμογής ο φοιτητής έχει την δυνατότητα να εγγράφεται ηλεκτρονικά στα εργαστηριακά μαθήματα του εξαμήνου που θα παρακολουθήσει. Επιπλέον το προσωπικό του ΤΕΙ δύναται να συγκεντρώνει όλες τις δηλώσεις των φοιτητών και να έχει εικόνα του συνόλου των εργαστηριακών ομάδων.

Τύποι Χρηστών και χαρακτηριστικά Χρηστών

Οι χρήστες της εφαρμογής είναι οι απλοί χρήστες-φοιτητές και οι διαχειριστές του συστήματος-προσωπικό ΤΕΙ. Οι φοιτητές (απλοί χρήστες) μπορούν να χρησιμοποιήσουν την εφαρμογή για να κάνουν δήλωση των εργαστηρίων τους. Οι διαχειριστές μπορούν να συγκεντρώνουν τα στοιχεία και είναι υπεύθυνοι για την σωστή λειτουργία της εφαρμογής καθώς έχουν πρόσβαση στην βάση δεδομένων.

Περιβάλλον Λειτουργίας

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκαν ο συνδυασμός των εργαλείων SQL-Visual Basic-Internet Information Server.

Περιορισμοί Σχεδίασης και Υλοποίησης

Επισημαίνεται ότι τα εργαστηριακά μαθήματα ανά εξάμηνο και ο τρόπος δήλωσής τους τέθηκαν στη βάση δεδομένων όπως ακριβώς ορίζονται στον κανονισμό σπουδών του τμήματος. Επίσης δεν δίνεται η δυνατότητα στον φοιτητή για εκ νέου δήλωση εργαστηρίων ή τροποποίηση της υπάρχουσας από το σύστημα. Τέλος η συντήρηση της εφαρμογής δεν καλύπτεται στο παρόν έγγραφο.

Εγχειρίδια Χρηστών

Το εγχειρίδιο χρήσης της εφαρμογής παρέχει κατευθυντήριες οδηγίες στους χρήστες ώστε με απλά βήματα να εξοικειωθούν και να μπορούν να κάνουν ασφαλή χρήση. Δίδονται οδηγίες τόσο για τον φοιτητή όσο και για τον διαχειριστή του συστήματος.

Παραδοχές και Εξαρτήσεις

Για την σωστή λειτουργία του συστήματος κρίνεται επιτακτική η πρόσβαση του προσωπικού του ΤΕΙ ως διαχειριστές του συστήματος στην βάση δεδομένων και η ύπαρξη (διάθεση) των απαραίτητων υπολογιστικών εργαλείων. Επιπλέον οι καθηγητές θα μπορούν να χρησιμοποιούν την εφαρμογή για να προσφέρουν ή να διαγράφουν εργαστηριακές ομάδες είτε μέσω της γραμματείας είτε ως διαχειριστές αφού πάρουν όνομα διαχειριστή και κωδικό.

6.1.3 Χαρακτηριστικά Συστήματος

Το διάγραμμα use case που φαίνεται στην εικόνα 1 του case study , συνοψίζει τις λειτουργικές απαιτήσεις του προϊόντος και τις κύριες υπηρεσίες που παρέχει. Οι περιπτώσεις χρήσης χωρίζονται σε χρήση από τν φοιτητή για υποβολή δήλωσης εργαστηρίων και χρήση από τον διαχειριστή για διαχείριση της βάσης δεδομένων της εφαρμογής.

Υποβολή Δήλωσης Φοιτητή

3.1.1 Περιγραφή και Προτεραιότητα

Ο φοιτητής μέσα από την χρήση της εφαρμογής θα μπορεί να επιλέξει και να υποβάλλει μια λίστα εργαστηριακών ομάδων. Υψηλής προτεραιότητας καθώς αποτελεί τον κύριο σκοπό της εφαρμογής.

3.1.2 Stimulus/Response Sequences

Όπως φαίνεται και στο use case διάγραμμα στο Σχήμα 2 του case study, ο φοιτητής εισάγει τον αριθμό μητρώου του και εμφανίζονται τα στοιχεία του. Μετά προχωρά στη δήλωση των εργαστηρίων αφού επιλέξει ομάδες εργαστηρίων από αυτές που εμφανίζονται.

3.1.3 Functional Requirements

REQ-1: Ο φοιτητής εισάγει το Α.Μ. του και εμφανίζονται τα στοιχεία του.

REQ-2: Κάθε φορά που επιλέγει μάθημα εξαμήνου φαίνονται τα αντίστοιχα προσφερόμενα εργαστήρια.

REQ-3: Επιλέγει την ομάδα που θέλει και δημιουργεί την λίστα εργαστηρίων του.

REQ-4: Υποβάλει την λίστα επιλογών .

Διαχείριση Εφαρμογής

3.2.1 Περιγραφή και Προτεραιότητα

Ο διαχειριστής μέσα από την χρήση της εφαρμογής θα μπορεί να επιλέξει τον αντίστοιχο πίνακα και να εισάγει, διαγράφει ή τροποποιεί δεδομένα. Υψηλής προτεραιότητας καθώς αποτελεί αναγκαία λειτουργία, ώστε η εφαρμογή να λειτουργεί σωστά.

3.2.2 Stimulus/Response Sequences

Όπως φαίνεται και στο use case διάγραμμα στο Σχήμα 3 του case study, ο διαχειριστής, αφού κάνει log in ,επιλέγει τον πίνακα που θέλει να τροποποιήσει και εμφανίζεται η αντίστοιχη φόρμα.

3.2.3 Functional Requirements

REQ-1: Ο διαχειριστής κάνει login και εμφανίζεται η φόρμα διαχείρισης εφαρμογής.

REQ-2: Επιλέγει ποιον πίνακα θέλει να τροποποιήσει και εμφανίζεται η αντίστοιχη φόρμα επεξεργασίας.

REQ-3: Έξοδος από την εφαρμογή.

6.1.4 Μη λειτουργικές Απαιτήσεις

Απαιτήσεις Χρηστών

Οι διεπαφές χρηστών θα σχεδιαστούν να είναι φιλικές προς τους χρήστες. Θα γίνει χρήση controls που παρέχονται από το visual studio και η αλληλεπιδράσεις των φορμών θα συνοδεύονται από μηνύματα αποδοχής ή σφάλματος όπου είναι απαραίτητο. Θα χρησιμοποιηθεί μια master page ως φόρμα-οδηγός και με βάση αυτό το πρότυπο θα σχεδιαστούν οι άλλες nested pages.Θα γίνει χρήση cascaded style sheets ώστε το αποτέλεσμα να είναι καλαίσθητο και εντυπωσιακό.

Απαιτήσεις Hardware

Το λογισμικό θα διατίθεται σε περιβάλλον Windows και θα εγκαθίσταται ιδιαίτερα εύκολα σε οποιοδήποτε μηχάνημα με υποστήριξη και αρχείου README το οποίο θα επεξηγεί και θα διευκρινίζει τα θέματα εγκατάστασης και λειτουργίας της εφαρμογής.

Απαιτήσεις Software

Θα χρησιμοποιηθεί το Microsoft Visual Studio 2010 για τον σχεδιασμό και την υλοποίηση της εφαρμογής, ένα πολυεργαλείο προγραμματισμού , όπου είναι δυνατός ο σχεδιασμός διαγραμμάτων UML , ο σχεδιασμός και υλοποίηση της Βάσης Δεδομένων σε γλώσσα SQL και η ανάπτυξη web project σε γλώσσα Visual Basic.NET

Απαιτήσεις Επικοινωνίας

Η υποβαλλόμενη δήλωση θα στέλνεται στον χρήστη φοιτητή με e-mail. Σε περίπτωση που τα στοιχεία του δεν είναι έγκυρα πρέπει να εγκαταλείπει την εφαρμογή και να επικοινωνεί με την γραμματεία.

Απαιτήσεις Εκτέλεσης

Το λογισμικό αρχικά θα εγκατασταθεί σε έναν server με ελάχιστη απαίτηση Pentium IV, 128 MB RAM και θα έχει την δυνατότητα εγκατάστασης σε πολλά τερματικά PCs (Pentium II) με λειτουργικό Windows στα οποία θα έχει εγκατασταθεί MySQL.

Απαιτήσεις Ασφαλείας και Διασφάλισης Απορρήτου

Η χρήση του Α.Μ. των φοιτητών γίνεται μόνο για την διασφάλιση μοναδικότητας της δήλωσής τους. Ο χρήστης διαχειριστής του συστήματος μπορεί να κάνει χρήση αυτής με όνομα χρήστη και κωδικό και αυτό για να αποφευχθεί η πιθανότητα ο φοιτητής να τροποποιήσει τα στοιχεία της βάσης δεδομένων της εφαρμογής.

Παράρτημα Α: Μοντέλα Ανάλυσης

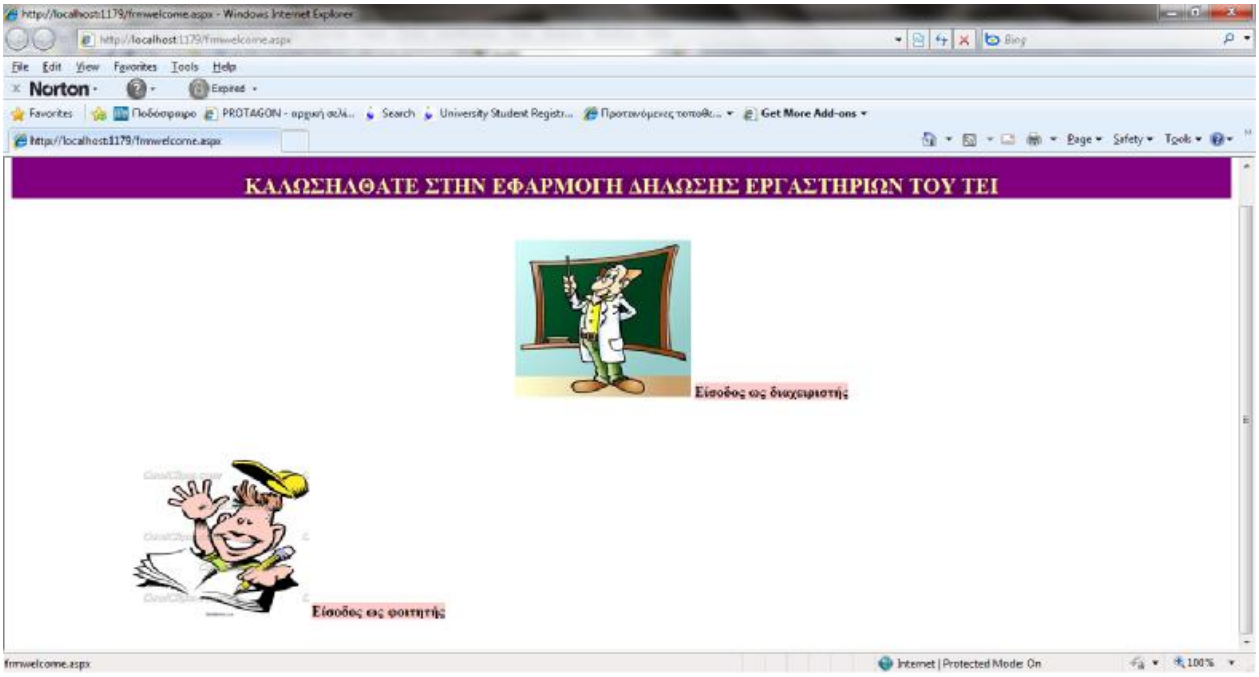
Μελέτη περίπτωσης: On line δήλωση εργαστηρίων φοιτητών

Παράρτημα Β: Λίστα Εκδόσεων

Εγχειρίδιο Χρήσης Εφαρμογής

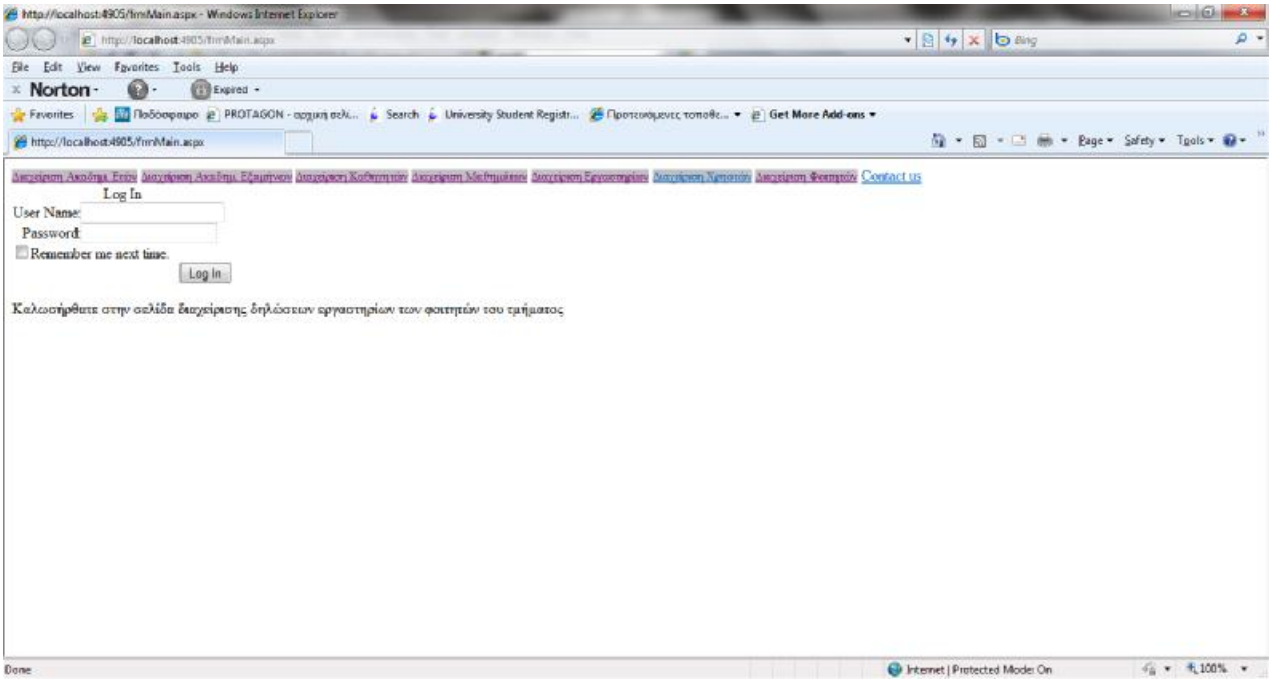
6.2 Εγχειρίδιο Χρηστών Εφαρμογής on-line δήλωσης εργαστηρίων

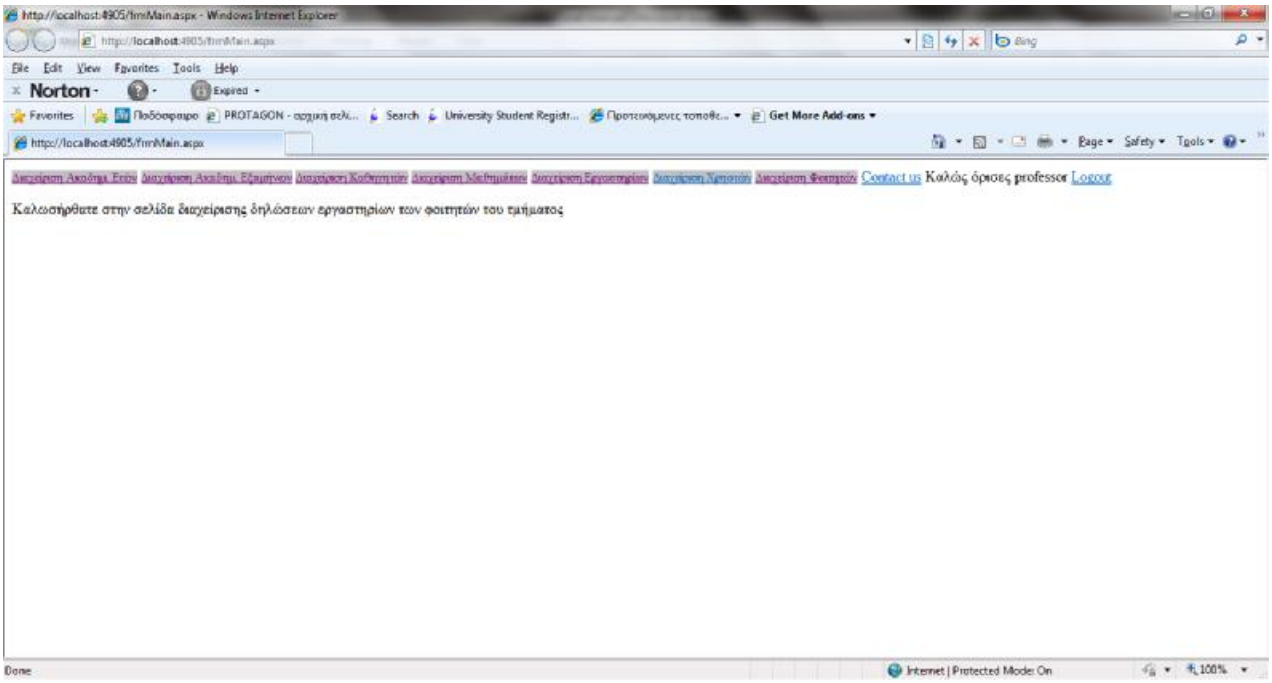
Αφού εισέλθετε στην εφαρμογή σας εμφανίζεται ένα μήνυμα χαιρετισμού και πατώντας στο αντίστοιχο imagebutton *επιλέγετε να εισέλθετε είτε ως φοιτητής είτε ως διαχειριστής οπότε αντίστοιχα εμφανίζεται η αρχική φόρμα της κάθε διεπαφής.*



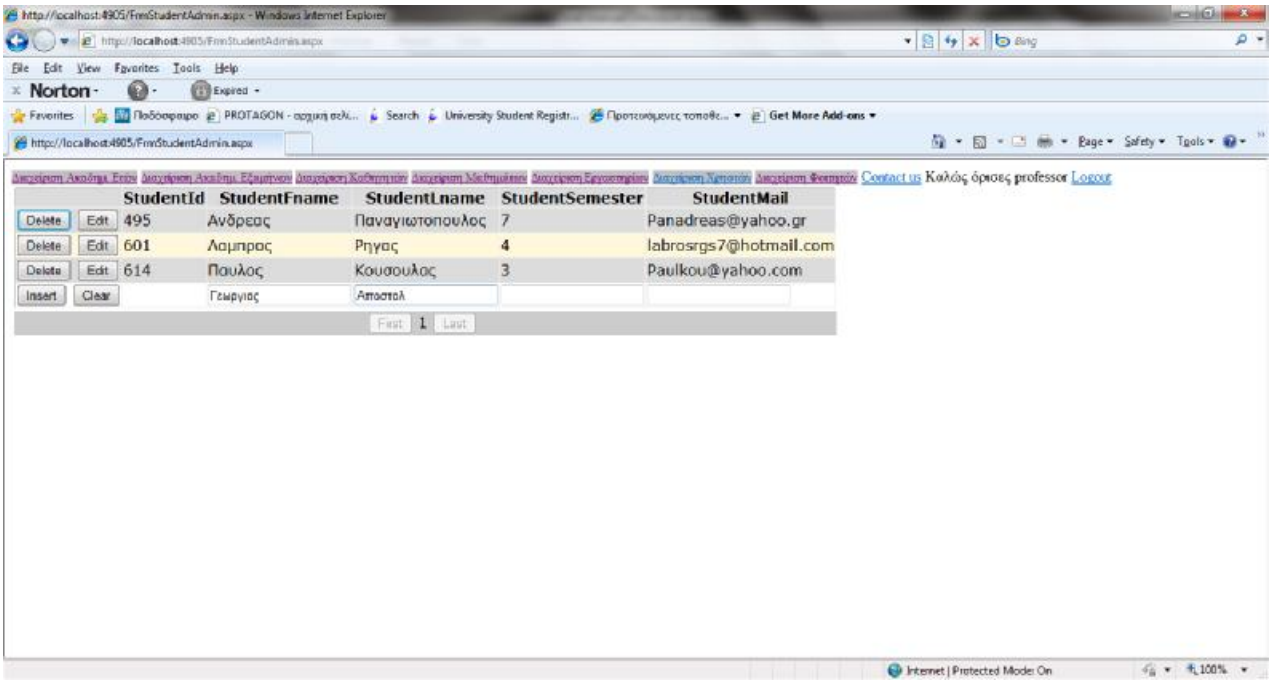
6.2.1 Α.Χρήστης-Διαχειριστής

Μπαίνοντας στην αρχική φόρμα διαχείρισης της βάσης δεδομένων του συστήματος ως διαχειριστής, βλέπετε στο επάνω μέρος την μπάρα με της επιλογές διαχείρισης που έχετε και την επιλογή επικοινωνίας με τους υπευθύνους της εφαρμογής, καθώς επίσης textboxes *και* button *για* login. *Πρέπει να εισαγάγετε το όνομα χρήστη και τον κωδικό που τα έχετε εφοδιαστεί από την γραμματεία και να πατήσετε το* button. *Κάνοντας επιτυχές* login *σας εμφανίζεται το όνομα χρήστη με το μήνυμα υποδοχής.*

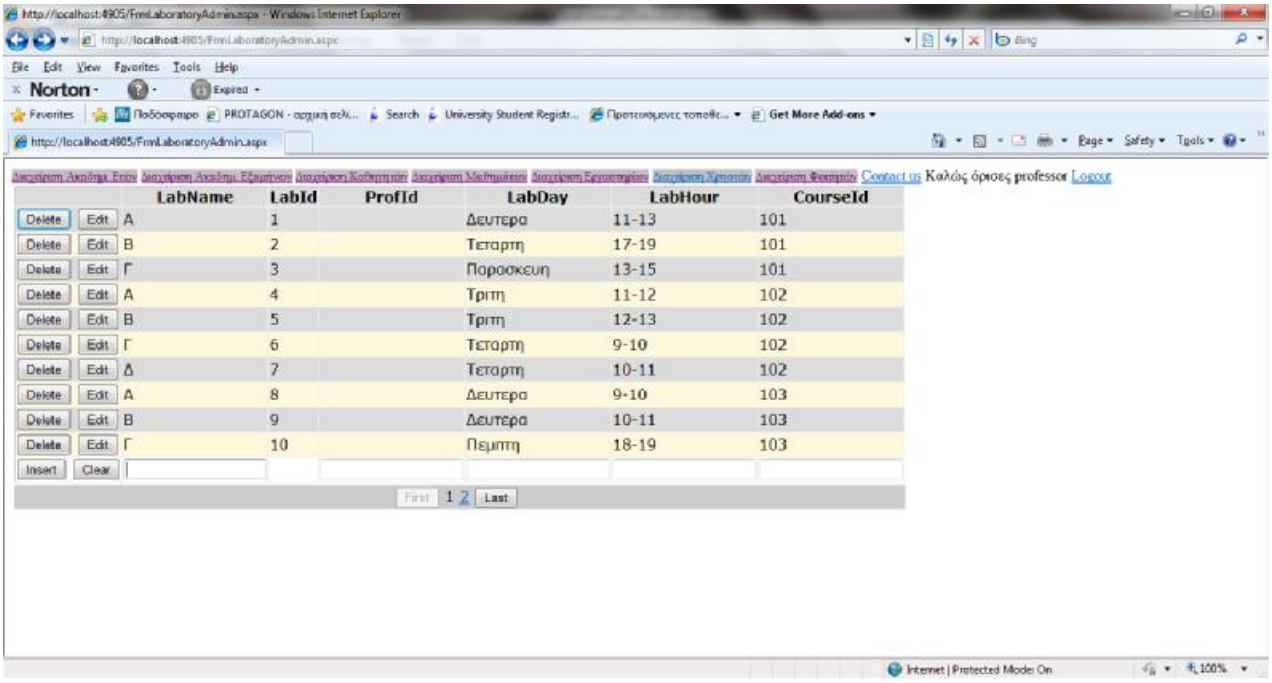




Μπορείτε τώρα να επιλέξετε οποιαδήποτε από τις επιλογές Ακαδημαϊκό εξάμηνο, καθηγητής, μάθημα, ακαδημαϊκό έτος, φοιτητής, εργαστήριο και χρήστες , πατώντας στο αντίστοιχο link *της μπάρας επιλογών και θα μεταβείτε στη φόρμα επιλογών διαχείρισης. Όλες οι φόρμες λειτουργούν με τον ίδιο τρόπο. Αν θέλουμε για παράδειγμα να προσθέσουμε έναν φοιτητή πληκτρολογούμε τα στοιχεία του και πατάμε το* button Insert.



Με παρόμοιο τρόπο αν θέλουμε για παράδειγμα να διαγράψουμε μια εργαστηριακή ομάδα το κάνουμε πατώντας το button delete *στα αριστερά του πίνακα .Με το* button Edit *τροποποιούμε μια ήδη υπάρχουσα καταγραφή.*

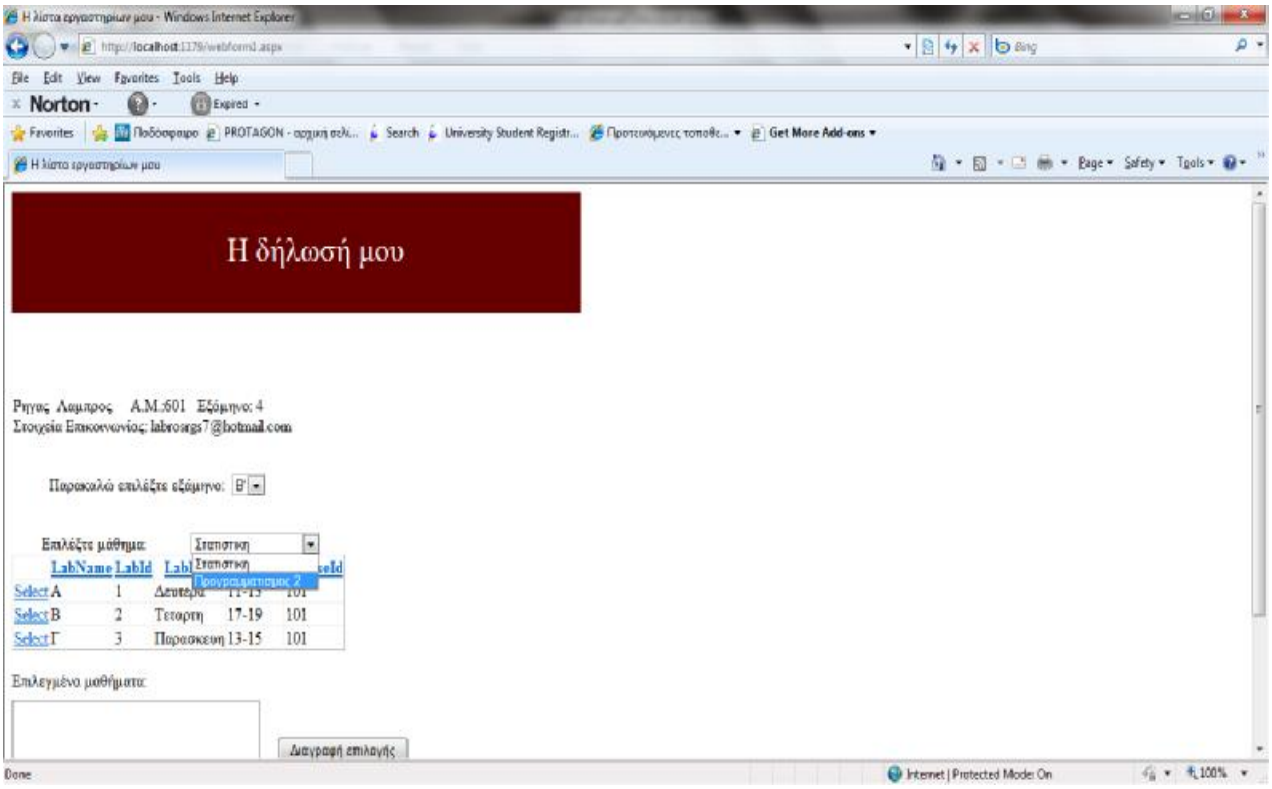


Αφού ολοκληρώσουμε τις αλλαγές μας κάνουμε logout *και μεταβαίνουμε στην αρχική φόρμα της εφαρμογής*

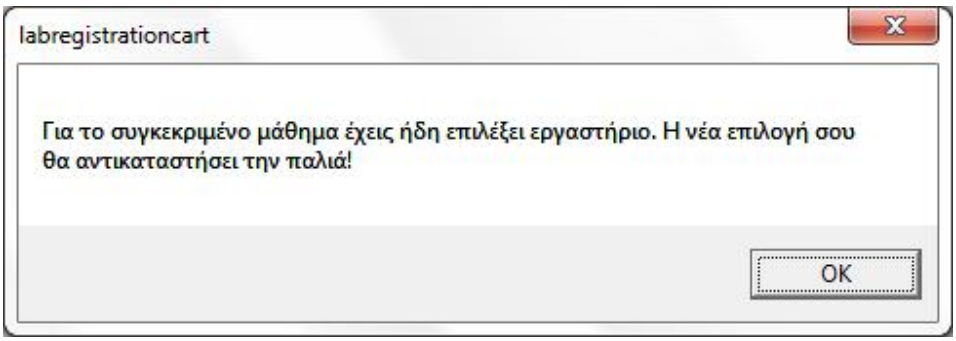
6.2.2 Β. Χρήστης-Φοιτητής

Εφόσον έχουμε εισέλθει ως φοιτητές στην εφαρμογή πατώντας στο αντίστοιχο imagebutton*, εμφανίζεται ένα πεδίο κειμένου όπου πρέπει να εισάγουμε τον αριθμό μητρώου που είναι και μοναδικός. Κατόπιν εμφανίζονται τα προσωπικά στοιχεία του φοιτητή όνομα, επώνυμο, εξάμηνο και έγκυρο* mail *επικοινωνίας.*

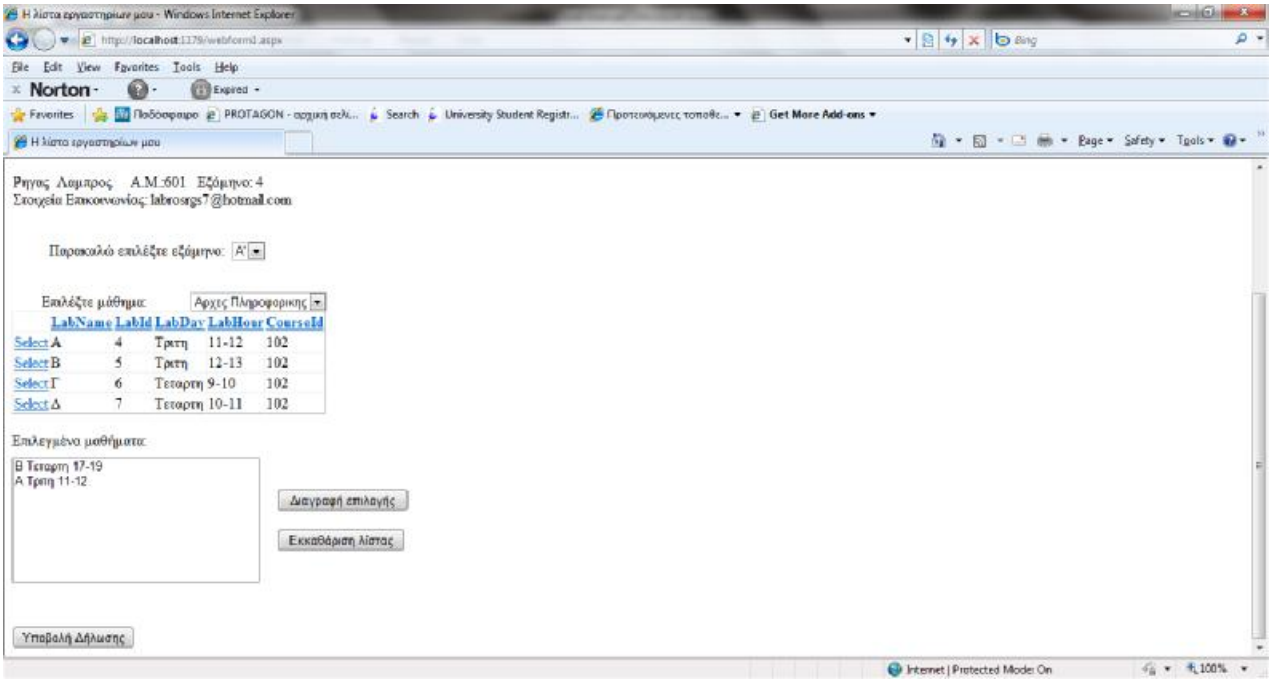
Αν τα στοιχεία σας δεν είναι έγκυρα εγκαταλείψτε την εφαρμογή και επικοινωνήστε με την γραμματεία, καθώς η δήλωσή σας θα είναι εσφαλμένη. Αν είναι σωστά πατήστε το button *και θα οδηγηθείτε στην φόρμα επιλογής εργαστηριακών ομάδων.*



Επιλέγοντας κάθε φορά εξάμηνο και μάθημα στα αντίστοιχα dropdownlists *, στην λίστα δεδομένων εμφανίζονται οι εργαστηριακές ομάδες που προσφέρονται με όλα τους τα στοιχεία. Μπορείτε να επιλέξετε μόνο μια πατώντας στο* button select *στα*



αριστερά. Κάθε φορά που επιλέγετε άλλη του ίδιου μαθήματος, η νέα σας επιλογή θα αντικαθιστά την παλιά. Οι επιλογές σας αυτόματα καταγράφονται στην λίστα που βρίσκεται στο κάτω μέρος της φόρμας.



Υπάρχουν buttons *με επιλογές διαγραφής κάποιας επιλογής καθώς επίσης και εκκαθάρισης ολόκληρης της λίστας.*

Αφού τελειώσετε τις επιλογές σας μπορείτε να υποβάλετε την δήλωση πατώντας στο αντίστοιχο button.*Οι επιλογές σας έχουν αποθηκευτεί στην βάση της γραμματείας και θα σας σταλούν με* e-mail *στην διεύθυνση που έχετε δώσει.*

ΣΥΠΜΕΡΑΣΜΑΤΑ

Με την παρούσα πτυχιακή πιστεύουμε ότι πετύχαμε τον αρχικό στόχο, δηλαδή την μελέτη των βασικών αρχών του software engineering όπως και τα βασικά πρότυπα μοντέλα διαδικασίας υλοποίησης λογισμικού με σκοπό την ανάλυση και υλοποίηση διαδικτυακής εφαρμογής δήλωσης εργαστηριακών μαθημάτων.

Η εφαρμογή αυτή θα καλύπτει τις βασικές λειτουργίες ενός χρήστη – φοιτητή για την δήλωση εργαστηριακών μαθημάτων και ομάδων. Όπως προκύπτει από τα παραπάνω η διαδικτυακή εφαρμογή θα διευκολύνει τους φοιτητές αφού δεν θα αναγκάζονται να πηγαίνουν στο ΤΕΙ για την δήλωση εργαστηρίων αλλά θα την κάνουν από οποιοδήποτε μέρος. Έτσι οι δηλώσεις θα γίνονται με πιο εύκολο και κατανοητό τρόπο και δεν θα υπάρχουν καθυστερήσεις αλλά ούτε προβλήματα όσον αφορά τις ομάδες αφού μέχρι τώρα οι δηλώσεις γίνονταν με ενυπόγραφες καταστάσεις και η διαδικασία ήταν χρονοβόρα. Πιστεύουμε ότι φθάσαμε σε ένα στάδιο το οποίο θα μπορέσουμε να λύσουμε ορισμένα προβλήματα και αν όχι όλα μερικά από αυτά και να εξυπηρετήσουμε εκτός από τους φοιτητές και τους καθηγητές αφού θα έχουν μια πιο ολοκληρωμένη και γενική εικόνα όσον αφορά τα εργαστήρια.

ΠΑΡΑΡΤΗΜΑ

(φόρμα Master)

```
<%@ Master Language="VB" AutoEventWireup="false"
CodeBehind="MainMasterPage.master.vb" Inherits="ptyxiaki.MainMasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="StyleSheet1.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
    <div id="nav">
                <a href="FrmAcademicYearAdmin.aspx"><span class="divstyle">Διαχείριση
Ακαδημ. Ετών</span></a>
                <a href="FrmAcademicSemesterAdmin.aspx" class="divstyle">Διαχείριση Ακαδημ.
Εξαμήνων</a>
                <a href="FrmProfessorAdmin.aspx" class="divstyle">Διαχείριση Καθηγητών</a>
                <a href="FrmCourseAdmin.aspx" class="divstyle">Διαχείριση Μαθημάτων</a>
                <a href="FrmLaboratoryAdmin.aspx" class="divstyle">Διαχείριση
Εργαστηρίων</a>
        <a href="FrmUserAdmin.aspx" class="divstyle">Διαχείριση Χρηστών</a>
        <a href="FrmStudentAdmin.aspx" class="divstyle">Διαχείριση Φοιτητών</a>
                <a class="lastchild" href="http://www.freewebsitetemplates.com">Contact us</a>
            <asp:LoginView ID="LoginView1" runat="server">
        <AnonymousTemplate>
          <asp:Login ID="Login1" runat="server">
          </asp:Login>
```

```
        </AnonymousTemplate>
                <LoggedInTemplate>
           Καλώς όρισες <asp:LoginName ID="LoginName1" runat="server" />
           <asp:LoginStatus ID="LoginStatus1" runat="server" />
        </LoggedInTemplate>
        </asp:LoginView>
           </div>    <div>
        <asp:ContentPlaceHolder ID="ContentPlaceHolderMainMasterPage" runat="server">
        </asp:ContentPlaceHolder>
    </div>    </form>
</body></html>
```

(φόρμα laboratoryadmin)

```
<%@ Page Title="" Language="vb" AutoEventWireup="false"
MasterPageFile="~/MainMasterPage.Master" CodeBehind="FrmLaboratoryAdmin.aspx.vb"
Inherits="ptyxiaki.FrmLaboratoryAdmin" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolderMainMasterPage"
runat="server">
 <asp:LoginView ID="LoginView1" runat="server">
    <AnonymousTemplate>
    Παρακαλώ κάντε login για να έχετε πρόσβαση στην σελίδα.
    </AnonymousTemplate>
    <LoggedInTemplate>
      <asp:ListView ID="lstvLaboratory" runat="server" DataKeyNames="LabId"
       DataSourceID="odsLaboratory" InsertItemPosition="LastItem">
       <AlternatingItemTemplate>
          <tr style="background-color:#FFF8DC;">
             <td>
```

```
            <asp:Button ID="DeleteButton" runat="server" CommandName="Delete"
              Text="Delete" />
            <asp:Button ID="EditButton" runat="server" CommandName="Edit" Text="Edit" />
          </td>            <td>
            <asp:Label ID="LabNameLabel" runat="server" Text='<%# Eval("LabName") %>' />
          </td>            <td>
            <asp:Label ID="LabIdLabel" runat="server" Text='<%# Eval("LabId") %>' />
          </td>            <td>
            <asp:Label ID="ProfIdLabel" runat="server" Text='<%# Eval("ProfId") %>' />
          </td>            <td>
            <asp:Label ID="LabDayLabel" runat="server" Text='<%# Eval("LabDay") %>' />
          </td>            <td>
            <asp:Label ID="LabHourLabel" runat="server" Text='<%# Eval("LabHour") %>' />
          </td>            <td>
            <asp:Label ID="CourseIdLabel" runat="server" Text='<%# Eval("CourseId") %>' />
          </td>          </tr>
      </AlternatingItemTemplate>
      <EditItemTemplate>
        <tr style="background-color:#008A8C;color: #FFFFFF;">
          <td>
            <asp:Button ID="UpdateButton" runat="server" CommandName="Update"
              Text="Update" />
            <asp:Button ID="CancelButton" runat="server" CommandName="Cancel"
              Text="Cancel" />
          </td>            <td>
            <asp:TextBox ID="LabNameTextBox" runat="server" Text='<%# Bind("LabName")
%>' />
          </td>            <td>
            <asp:Label ID="LabIdLabel1" runat="server" Text='<%# Eval("LabId") %>' />
          </td>            <td>
```

```
            <asp:TextBox ID="ProfIdTextBox" runat="server" Text='<%# Bind("ProfId") %>' />
        </td>            <td>
            <asp:TextBox ID="LabDayTextBox" runat="server" Text='<%# Bind("LabDay") %>'
/>
        </td>            <td>
            <asp:TextBox ID="LabHourTextBox" runat="server" Text='<%# Bind("LabHour")
%>' />
        </td>            <td>
            <asp:TextBox ID="CourseIdTextBox" runat="server"
                Text='<%# Bind("CourseId") %>' />
        </td>          </tr>
    </EditItemTemplate>
    <EmptyDataTemplate>
      <table runat="server"
        style="background-color: #FFFFFF;border-collapse: collapse;border-color:
#999999;border-style:none;border-width:1px;">
        <tr>              <td>
            No data was returned.</td>
        </tr>
      </table>
    </EmptyDataTemplate>
    <InsertItemTemplate>
      <tr style="">
        <td>
          <asp:Button ID="InsertButton" runat="server" CommandName="Insert"
            Text="Insert" />
          <asp:Button ID="CancelButton" runat="server" CommandName="Cancel"
            Text="Clear" />
        </td>            <td>
```

```
            <asp:TextBox ID="LabNameTextBox" runat="server" Text='<%# Bind("LabName")
%>' />
        </td>
        <td>                 </td>
        <td>
            <asp:TextBox ID="ProfIdTextBox" runat="server" Text='<%# Bind("ProfId") %>' />
        </td>            <td>
            <asp:TextBox ID="LabDayTextBox" runat="server" Text='<%# Bind("LabDay") %>'
/>
        </td>
        <td>
            <asp:TextBox ID="LabHourTextBox" runat="server" Text='<%# Bind("LabHour")
%>' />
        </td>            <td>
            <asp:TextBox ID="CourseIdTextBox" runat="server"
                Text='<%# Bind("CourseId") %>' />
        </td>          </tr>
    </InsertItemTemplate>
    <ItemTemplate>
      <tr style="background-color:#DCDCDC;color: #000000;">
        <td>
            <asp:Button ID="DeleteButton" runat="server" CommandName="Delete"
                Text="Delete" />
            <asp:Button ID="EditButton" runat="server" CommandName="Edit" Text="Edit" />
        </td>            <td>
            <asp:Label ID="LabNameLabel" runat="server" Text='<%# Eval("LabName") %>' />
        </td>            <td>
            <asp:Label ID="LabIdLabel" runat="server" Text='<%# Eval("LabId") %>' />
        </td>            <td>
            <asp:Label ID="ProfIdLabel" runat="server" Text='<%# Eval("ProfId") %>' />
```

```
            </td>            <td>
                <asp:Label ID="LabDayLabel" runat="server" Text='<%# Eval("LabDay") %>' />
            </td>            <td>
                <asp:Label ID="LabHourLabel" runat="server" Text='<%# Eval("LabHour") %>' />
            </td>            <td>
                <asp:Label ID="CourseIdLabel" runat="server" Text='<%# Eval("CourseId") %>' />
            </td>          </tr>
        </ItemTemplate>
        <LayoutTemplate>
            <table runat="server">
                <tr runat="server">
                    <td runat="server">
                        <table ID="itemPlaceholderContainer" runat="server" border="1"
                            style="background-color: #FFFFFF;border-collapse: collapse;border-color:
#999999;border-style:none;border-width:1px;font-family: Verdana, Arial, Helvetica, sans-serif;">
                            <tr runat="server" style="background-color:#DCDCDC;color: #000000;">
                                <th runat="server">
                                </th>                            <th runat="server">
                                    LabName</th>
                                <th runat="server">
                                    LabId</th>
                                <th runat="server">
                                    ProfId</th>
                                <th runat="server">
                                    LabDay</th>
                                <th runat="server">
                                    LabHour</th>
                                <th runat="server">
                                    CourseId</th>
                            </tr>                        <tr ID="itemPlaceholder" runat="server">
```

```
                    </tr>                    </table>
                </td>                </tr>
            <tr runat="server">
                <td runat="server"
                    style="text-align: center;background-color: #CCCCCC;font-family: Verdana, Arial,
Helvetica, sans-serif;color: #000000;">
                    <asp:DataPager ID="DataPager1" runat="server">
                        <Fields>
                            <asp:NextPreviousPagerField ButtonType="Button"
ShowFirstPageButton="True"
                                ShowNextPageButton="False" ShowPreviousPageButton="False" />
                            <asp:NumericPagerField />
                            <asp:NextPreviousPagerField ButtonType="Button"
ShowLastPageButton="True"
                                ShowNextPageButton="False" ShowPreviousPageButton="False" />
                        </Fields>
                    </asp:DataPager>
                </td>                </tr>
        </table>
    </LayoutTemplate>
    <SelectedItemTemplate>
        <tr style="background-color:#008A8C;font-weight: bold;color: #FFFFFF;">
            <td>
                <asp:Button ID="DeleteButton" runat="server" CommandName="Delete"
                    Text="Delete" />
                <asp:Button ID="EditButton" runat="server" CommandName="Edit" Text="Edit" />
            </td>                <td>
                <asp:Label ID="LabNameLabel" runat="server" Text='<%# Eval("LabName") %>' />
            </td>                <td>
                <asp:Label ID="LabIdLabel" runat="server" Text='<%# Eval("LabId") %>' />
```

```
        </td>            <td>
            <asp:Label ID="ProfIdLabel" runat="server" Text='<%# Eval("ProfId") %>' />
        </td>            <td>
            <asp:Label ID="LabDayLabel" runat="server" Text='<%# Eval("LabDay") %>' />
        </td>            <td>
            <asp:Label ID="LabHourLabel" runat="server" Text='<%# Eval("LabHour") %>' />
        </td>            <td>
            <asp:Label ID="CourseIdLabel" runat="server" Text='<%# Eval("CourseId") %>' />
        </td>          </tr>
    </SelectedItemTemplate>
</asp:ListView>
<asp:ObjectDataSource ID="odsLaboratory" runat="server" DeleteMethod="Delete"
    InsertMethod="Insert" OldValuesParameterFormatString="original_{0}"
    SelectMethod="GetData"
    TypeName="ptyxiaki.DataSet1TableAdapters.LaboratoriesTableAdapter"
    UpdateMethod="Update">
    <DeleteParameters>
        <asp:Parameter Name="Original_LabId" Type="Int32" />
    </DeleteParameters>
    <InsertParameters>
        <asp:Parameter Name="LabName" Type="String" />
        <asp:Parameter Name="ProfId" Type="Int32" />
        <asp:Parameter Name="LabDay" Type="String" />
        <asp:Parameter Name="LabHour" Type="String" />
        <asp:Parameter Name="CourseId" Type="Int32" />
    </InsertParameters>
    <UpdateParameters>
        <asp:Parameter Name="LabName" Type="String" />
        <asp:Parameter Name="ProfId" Type="Int32" />
        <asp:Parameter Name="LabDay" Type="String" />
```

```
            <asp:Parameter Name="LabHour" Type="String" />
            <asp:Parameter Name="CourseId" Type="Int32" />
            <asp:Parameter Name="Original_LabId" Type="Int32" />
        </UpdateParameters>
    </asp:ObjectDataSource> </LoggedInTemplate>
    </asp:LoginView>
</asp:Content>
```

(φόρμα δήλωσης-αρχική)

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="WebForm2.aspx.vb"
Inherits="labregistrationcart.WebForm2" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head><body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server"
            Text="Πληκτρολογηστε τον ΑΜ σας
            ωστε να εμφανιστούν τα στοιχεία σας :  "></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
            SelectCommand="SELECT * FROM [Student] WHERE ([StudentId] = @StudentId)">
            <SelectParameters>
```

```
            <asp:ControlParameter ControlID="TextBox1" Name="StudentId"
PropertyName="Text"
                Type="Int32" />
        </SelectParameters>
    </asp:SqlDataSource>
    <br />
    <asp:GridView ID="GridView2" runat="server" AutoGenerateColumns="False"
      DataKeyNames="StudentId" DataSourceID="SqlDataSource1">
      <Columns>
        <asp:CommandField ShowSelectButton="True" />
        <asp:BoundField DataField="StudentId" HeaderText="StudentId"
          InsertVisible="False" ReadOnly="True" SortExpression="StudentId" />
        <asp:BoundField DataField="StudentFname" HeaderText="StudentFname"
          SortExpression="StudentFname" />
        <asp:BoundField DataField="StudentLname" HeaderText="StudentLname"
          SortExpression="StudentLname" />
        <asp:BoundField DataField="StudentSemester" HeaderText="StudentSemester"
          SortExpression="StudentSemester" />
        <asp:BoundField DataField="StudentMail" HeaderText="StudentMail"
          SortExpression="StudentMail" />
      </Columns>
    </asp:GridView>
  </div>    </form></body></html>
```

(κώδικας vb)

```
Public Class WebForm2
  Inherits System.Web.UI.Page

  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
  End Sub
```

```
    Protected Sub GridView2_SelectedIndexChanged(ByVal sender As Object, ByVal e As
EventArgs) Handles GridView2.SelectedIndexChanged
        Session("StudentId") = GridView2.SelectedRow.Cells(1).Text
        Response.Redirect("webform1.aspx")
    End Sub
End Class
```

(φόρμα δήλωσης – λίστα δηλώσεων)

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="WebForm1.aspx.vb"
Inherits="labregistrationcart.WebForm1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Η λίστα εργαστηρίων μου</title>
    <style type="text/css">
        .style1
        {
            margin-left: 40px;
        }
    </style></head><body>
    <form id="form1" runat="server">
    <div>    <div style="font-size: xx-large; vertical-align: baseline; width: 614px; color: #ffffff;
            direction: ltr; height: 112px; background-color: #660000; text-align: left; font-variant:
normal">
             
            <br />
                               
                Η δήλωσή μου</div>
        <br /><br />       <br />
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
```

```
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
    SelectCommand="SELECT * FROM [Student] WHERE ([StudentId] = @StudentId)">
    <SelectParameters>
        <asp:SessionParameter Name="StudentId" SessionField="StudentId" Type="Int32" />
    </SelectParameters>
</asp:SqlDataSource>
<asp:FormView ID="FormView1" runat="server" DataKeyNames="StudentId"
    DataSourceID="SqlDataSource1">
    <EditItemTemplate>
        StudentId:
        <asp:Label ID="StudentIdLabel1" runat="server"
            Text='<%# Eval("StudentId") %>' />
        <br />
        StudentFname:
        <asp:TextBox ID="StudentFnameTextBox" runat="server"
            Text='<%# Bind("StudentFname") %>' />
        <br />
        StudentLname:
        <asp:TextBox ID="StudentLnameTextBox" runat="server"
            Text='<%# Bind("StudentLname") %>' />
        <br />
        StudentSemester:
        <asp:TextBox ID="StudentSemesterTextBox" runat="server"
            Text='<%# Bind("StudentSemester") %>' />
        <br />
        StudentMail:
        <asp:TextBox ID="StudentMailTextBox" runat="server"
            Text='<%# Bind("StudentMail") %>' />
        <br />
        <asp:LinkButton ID="UpdateButton" runat="server" CausesValidation="True"
```

```
        CommandName="Update" Text="Update" />
     <asp:LinkButton ID="UpdateCancelButton" runat="server"
        CausesValidation="False" CommandName="Cancel" Text="Cancel" />
</EditItemTemplate>
<InsertItemTemplate>
    StudentFname:
    <asp:TextBox ID="StudentFnameTextBox" runat="server"
        Text='<%# Bind("StudentFname") %>' />
    <br />
    StudentLname:
    <asp:TextBox ID="StudentLnameTextBox" runat="server"
        Text='<%# Bind("StudentLname") %>' />
    <br />
    StudentSemester:
    <asp:TextBox ID="StudentSemesterTextBox" runat="server"
        Text='<%# Bind("StudentSemester") %>' />
    <br />
    StudentMail:
    <asp:TextBox ID="StudentMailTextBox" runat="server"
        Text='<%# Bind("StudentMail") %>' />
    <br />
    <asp:LinkButton ID="InsertButton" runat="server" CausesValidation="True"
        CommandName="Insert" Text="Insert" />
     <asp:LinkButton ID="InsertCancelButton" runat="server"
        CausesValidation="False" CommandName="Cancel" Text="Cancel" />
</InsertItemTemplate>          <ItemTemplate>
    <br />
    <asp:Label ID="StudentLnameLabel" runat="server"
        Text='<%# Bind("StudentLname") %>' />
     <asp:Label ID="StudentFnameLabel" runat="server"
```

```
            Text='<%# Bind("StudentFname") %>' />
              A.M.:<asp:Label ID="StudentIdLabel" runat="server"
            Text='<%# Eval("StudentId") %>' />
            Εξάμηνο:
          <asp:Label ID="StudentSemesterLabel" runat="server"
            Text='<%# Bind("StudentSemester") %>' />
          <br />
          Στοιχεία Επικοινωνίας:
          <asp:Label ID="StudentMailLabel" runat="server"
            Text='<%# Bind("StudentMail") %>' />
          <br />          </ItemTemplate>
      </asp:FormView>
      <br />       <div>
        <p class="style1">
          Παρακαλώ επιλέξτε εξάμηνο: 
          <asp:DropDownList ID="DDLSemester" runat="server" AutoPostBack="True">
            <asp:ListItem Value="1">A&#39;</asp:ListItem>
            <asp:ListItem Value="2">B&#39;</asp:ListItem>
          </asp:DropDownList>
        </p>       </div>
      <asp:SqlDataSource ID="SqlDSSemester" runat="server"
        ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
            SelectCommand="SELECT [CourseName] FROM [Course] WHERE
([CourseSemester] = @CourseSemester)">
        <SelectParameters>
          <asp:ControlParameter ControlID="DDLSemester" Name="CourseSemester"
            PropertyName="SelectedValue" Type="Int32" />
        </SelectParameters>
      </asp:SqlDataSource>       <br />
```

```
        Επιλέξτε μάθημα:           
    <asp:DropDownList ID="DDLCourseName" runat="server" AutoPostBack="True"
      DataSourceID="SqlDSSemester" DataTextField="CourseName"
      DataValueField="CourseName">
    </asp:DropDownList>
    <br />
    <asp:SqlDataSource ID="SqlDSLabsAvailable" runat="server"
      ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
            SelectCommand="SELECT Laboratories.* FROM Course INNER JOIN
Laboratories ON Course.CourseId = Laboratories.CourseId WHERE (Course.CourseName =
@CourseName)">
      <SelectParameters>
        <asp:ControlParameter ControlID="DDLCourseName" Name="CourseName"
          PropertyName="SelectedValue" Type="String" />
      </SelectParameters>
    </asp:SqlDataSource>
    <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
      AllowSorting="True" AutoGenerateColumns="False" DataKeyNames="LabId"
      DataSourceID="SqlDSLabsAvailable">
      <Columns>
        <asp:CommandField ShowSelectButton="True" />
        <asp:BoundField DataField="LabName" HeaderText="LabName"
          SortExpression="LabName" />
        <asp:BoundField DataField="LabId" HeaderText="LabId" InsertVisible="False"
          ReadOnly="True" SortExpression="LabId" />
        <asp:BoundField DataField="ProfId" HeaderText="ProfId"
          SortExpression="ProfId" Visible="False" />
        <asp:BoundField DataField="LabDay" HeaderText="LabDay"
          SortExpression="LabDay" />
```

```
            <asp:BoundField DataField="LabHour" HeaderText="LabHour"
               SortExpression="LabHour" />
            <asp:BoundField DataField="CourseId" HeaderText="CourseId"
               SortExpression="CourseId" />
         </Columns>
      </asp:GridView>
      <br />
      Επιλεγμένα μαθήματα:<br />
      <table style="width: 500px" cellspacing="0"
            cellpadding="0" border="0">
         <tr>
            <td style="width: 286px; height: 153px">
               <asp:ListBox ID="lstCart" runat="server"
                  Width="267px" Height="135px">                </asp:ListBox>
            </td>
            <td style="height: 153px">
               <asp:Button ID="btnRemove" runat="server"
                  Width="142px" Text="Διαγραφή επιλογής" /><br /><br />
               <asp:Button ID="btnEmpty" runat="server"
                  Width="138px" Text="Εκκαθάριση λίστας" />
            </td>          </tr>       </table>       <br />       <br />
      <asp:Button ID="Button1" runat="server" Text="Υποβολή Δήλωσης" />
   </div>    </form></body></html>
```

(κώδικας vb)

```
Imports System.Data.SqlClient
Public Class WebForm1
   Inherits System.Web.UI.Page
   Dim newcol As Collection
   Private Sub GridView1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles GridView1.SelectedIndexChanged
```

```
        newcol = CType(Session("RegLabcol"), Collection)
        Dim lab As New Laboratory
        lab.LabName = GridView1.SelectedRow.Cells(1).Text
        lab.LabID = CInt(GridView1.SelectedRow.Cells(2).Text)
        'lab.ProfId = CInt(GridView1.SelectedRow.Cells(3).Text)
        lab.LabDay = GridView1.SelectedRow.Cells(4).Text
        lab.LabHour = GridView1.SelectedRow.Cells(5).Text
        lab.CourseId = CInt(GridView1.SelectedRow.Cells(6).Text)
        Dim found As Boolean = False
        Dim foundLabId As Integer = 0
        For i As Integer = 1 To newcol.Count
            Dim labnew As New Laboratory

            labnew = newcol(i)
            If labnew.CourseId = lab.CourseId Then
                found = True
                foundLabId = i
                Exit For
            End If
        Next
        If found Then
            MsgBox("Για το συγκεκριμένο μάθημα έχεις ήδη επιλέξει εργαστήριο. Η νέα επιλογή σου
θα αντικαταστήσει την παλιά!")
            newcol.Remove(foundLabId)
        End If
        "(GridView1.SelectedValue)
        newcol.Add(lab, lab.LabID)
        Session("RegLabcol") = newcol
        DisplayCart()
    End Sub
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
Button1.Click
        newcol = CType(Session("RegLabcol"), Collection)
        For i As Integer = 1 To newcol.Count
            Dim lab As New Laboratory          lab = newcol.Item(i)

            Dim commandText As String = "INSERT INTO REGISTRATION
(AcademicSemId,AcademicYearId,LabId,StudentId,RegDate) VALUES (" &
Session("AcademicSemId") & "," & Session("AcademicYearId") & "," & lab.LabID & "," &
Session("StudentId") & "," & Now.Date.ToString

            Using connection As New SqlConnection(ConnectionStringSettings)
                Dim command As New SqlCommand(commandText, connection)
                Try
                    connection.Open()
```

```
                Dim rowsAffected As Integer = command.ExecuteNonQuery()
            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try
        End Using
    Next
    Response.Redirect("webform2.aspx")
  End Sub
  Private Sub DisplayCart()
    lstCart.Items.Clear()
    ' newcol = CType(Session("RegLabcol"), Collection)
    For i As Integer = 1 To newcol.Count
      Dim lab As New Laboratory

      lab = newcol.Item(i)
      lstCart.Items.Add(lab.LabName & " " & lab.LabDay & " " & lab.LabHour)
    Next
  End Sub
  Protected Sub btnRemove_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnRemove.Click
    newcol = CType(Session("RegLabcol"), Collection)
    newcol.Remove(lstCart.SelectedIndex)
    Session("RegLabcol") = newcol
    DisplayCart()
  End Sub
  Protected Sub btnEmpty_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnEmpty.Click
    newcol = CType(Session("RegLabcol"), Collection)

    newcol.Clear()

    Session("RegLabcol") = newcol

    DisplayCart()
  End Sub

  Private Function ConnectionStringSettings() As String
    Throw New NotImplementedException
  End Function

End Class
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Ian Sommerville, M. K. (2008). *Software engineering.* Pearson Education Limited.
- Per Kroll, P. K. (2003). *The rational unified process made easy: a practitioner's guide to the RUP.* Addison-Wesley.
- Andrew Moore,(2010). Visual Studio 2010 ALL-IN-ONE for DUMMIES
- Joe Mayo, (2010). Visual Sudio 2010 beginners guide
- George Shepherd, (2010) Microsoft ASP.NET Step byStep
- Michael Halvorson, (2010) Microsoft Visual Basic.NET βήμα βήμα.
- Πρόδρομος Χατζόγλου, (2005) Τεχνικές Ανάλυσης και Σχεδίασης Πληροφοριακών Συστημάτων
- Harvey M.,Deitel, Paul J.,Deitel, T.R.,Nieto, Visual Basic.NET, (2002) , Pearson Education Limited
- Deitel, Nieto, VisualBasic.NET, (2003), Γκιούρδας Μ.
- Petkovic, Dusan, (2006), Οδηγός του SQL Server 2005 ,Γκιούρδας Μ.

ΠΗΓΕΣ ΔΙΑΔΙΚΤΥΟΥ

www.msdn.com

www.asp.com

www.w3schools.com

http://www.homeandlearn.co.uk/NET/vbNet.html (Αρκετά καλό για αρχάριους στην vb)

http://www.vbtutor.net/vb2008/vb2008_lesson1.html

http://www.programmersheaven.com/2/VB-NET-School