ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΑΣ
ΠΑΡΑΡΤΗΜΑ ΑΜΑΛΙΑΔΑΣ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ

ΕΦΑΡΜΟΓΕΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΣΤΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΟΙΚΟΝΟΜΙΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΙΟΥΡΓΙΑ ΙΣΤΟΣΕΛΙΔΑΣ ΓΙΑ ΤΗ ΠΡΟΒΟΛΗ ΜΙΑΣ ΣΥΓΚΕΚΡΙΜΕΝΗΣ ΕΤΑΙΡΕΙΑΣ – ΕΠΙΧΕΙΡΗΣΗΣ

WEB-SITE DESIGN FOR A COMPANY OR AN ENTERPRISE

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ: ΛΕΝΟΥ ΔΗΜΗΤΡΑ

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ: ΤΡΥΦΩΝΟΠΟΥΛΟΣ ΑΘΗΝΟΔΩΡΟΣ

ΤΟΠΟΣ-ΧΡΟΝΟΛΟΓΙΑ: ΑΘΗΝΑ 2012

Περιεχομενα

ΚΕΦΑΛΑΙΟ 1: **ΓΝΩΡΙΜΙΑ ΜΕ ΤΗΝ ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ** 1

1.1 **Υπολογιστες και Λογισμικο** **1**
1.2 **Τεχνικες κατασκευες και Λογισμικο** 2
1.3 **Κριση Λογισμικου** 3
1.3.1 Τι είναι λογισμικό *5*
1.3.2 Τι είναι Τεχνολογία Λογισμικού *6*
1.3.3 Ποια η διαφορά μεταξύ τεχνολογίας λογισμικού και επιστήμης των υπολογιστών; *7*
1.3.4 Ποια είναι η διαφορά μεταξύ τεχνολογίας λογισμικού και τεχνολογίας συστημάτων. *7*
1.4 **Τι ειναι διαδικασια παραγωγης λογισμικου** 8
1.4.1 Τι είναι μοντέλο διαδικασίας παραγωγής λογισμικού; *9*
1.4.2 Ποιο είναι το κόστος της τεχνολογίας λογισμικού; *10*
1.4.3 Τι είναι μέθοδοι της τεχνολογίας λογισμικού; *13*
1.4.4 Ποιες προκλήσεις αντιμετωπίζει η τεχνολογία λογισμικού; *15*

ΚΕΦΑΛΑΙΟ 2 : **ΜΟΝΤΕΛΑ ΚΥΚΛΟΥ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ** 18

2.1 **Η εννοια του μοντελου κυκλου ζωης** 18
2.2 **Μοντελο Κυκλου Ζωης Λογισμικου** 19
2.2.1 Μοντέλα διαδικασιών παραγωγής λογισμικού *22*
2.3 **Το μοντελο καταρρακτη** 24
2.4 **Εξελικτικη αναπτυξη** 28
2.5 **Το μοντελο πρωτυποποιησης** 32
2.6 **Το μοντελο λειτουργικης επαυξησης** 33
2.7 **Επαναληπτικες διαδικασιες** 34
2.8 **Βαθμιαια παραδοση** 36
2.9 **Το σπειροειδης μοντελο** 38
2.10 **Το μοντελο του πιδακα** 40

ΚΕΦΑΛΑΙΟ 3 : **ΠΡΟΔΙΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ** 43

3.1 **Λειτουργικες και μη λειτουργικες απαιτησεις** 46
3.1.1 Λειτουργικές απαιτήσεις *47*
3.1.2 Μη λειτουργικές απαιτήσεις *49*
3.2 **Απαιτησεις πεδιου** 54
3.3 **Απαιτησεις χρηστη** 55
3.4 **Απαιτησεις συστηματος** **58**
3.5 **Προδιαγραφες σε δομημενη γλωσσα** **61**
3.6 **Προδιαγραφες διασυνδεσης** **66**
3.7 **Το εγγραφο των απαιτησεων λογισμικου** **67**
3.8 **Μηχανικη Απαιτησεων** **73**
3.8.1 Βήματα στον προσδιορισμό απαιτήσεων *74*
3.9 **Αναλυση απαιτησεων** **76**
3.9.1 Προδιαγραφή Απαιτήσεων *78*

ΚΕΦΑΛΑΙΟ 4 : **ΣΧΕΔΙΑΣΗ** 82

4.1 **ΣΚΟΠΟΣ ΤΗΣ ΣΧΕΔΙΑΣΗΣ** 82
4.1.1 Σχέδιο λογισμικού 82
4.2 **ΤΕΧΝΟΤΡΟΠΙΕΣ ΣΧΕΔΙΑΣΗΣ** 84
4.2.1 Δομημένη σχεδίαση 85
4.2.2 Αντικεινενοστρεφής σχεδίαση 85
4.3 **ΑΝΤΙΚΕΙΜΕΝΟ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΣΧΕΔΙΑΣΗΣ** 86
4.3.1 Αρχιτεκτονική σχεδίαση 89
4.3.2 Σχεδίαση διαπροσωπειών 90
4.3.3 Λεπτομερής σχεδίαση μονάδων 91
4.3.4 Σχεδίαση δεδομένων 92
4.3.5 Το έγγραφο περιγραφής του σχεδίου του λογισμικού 93
4.4 **ΔΙΑΤΑΞΕΙΣ ΛΟΓΙΣΜΙΚΟΥ** 94
4.4.1 Διάταξη λογισμικού 95
4.4.2 Η μονολιθική διάταξη 96
4.4.3 Η διάταξη πελάτη-εξυπηρετητή 96
4.4.4 Η τριμερής Διάταξη 97
4.4.5 Πολυμερής διάταξη 98
4.5 **ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΧΕΔΙΑΣΗ** 99
4.5.1 Κεντρικός μετασχηματισμός 100
4.5.2 Βήματα κατασκευής διαγραμμάτων δομής 105
4.6 **ΛΕΠΤΟΜΕΡΗΣ ΣΧΕΔΙΑΣΗ ΜΟΝΑΔΩΝ** 106
4.7 **ΣΧΕΔΙΑΣΗ ΔΕΔΟΜΕΝΩΝ** 109

ΚΕΦΑΛΑΙΟ 5 : **ΠΑΡΑΓΩΓΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ** 111

5.1 **ΑΠΟ ΤΗΝ ΣΧΕΔΙΑΣΗ ΣΤΗΝ ΚΩΔΙΚΟΠΟΙΗΣΗ** 111
5.1.1 Λογισμικό χωρίς σφάλματα 111
5.1.2 Εργαλεια κωδικοποίησης 112
5.2 **ΕΠΙΘΥΜΗΤΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ** 114
5.2.1 Επάρκεια 114
5.2.2 Επίδοση 115
5.2.3 Αναγνωσιμότητα 115
5.2.4 Τεκμηρίωση 116
5.2.5 Μεταφερσιμότητα 117
5.2.6 Δυνατότητα επαναχρησιμοποίησης 117
5.3 **ΤΕΧΝΙΚΕΣ ΣΥΓΓΡΑΦΗΣ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ** 117
5.3.1 Τεχνικές αποφυγής σφαλμάτων 118
5.3.2 Προδιαγραφή απαιτήσεων 118
5.3.3 Αξιοποίηση της γλώσσας προγραμματισμού 119
5.3.4 Επιδιωξη ποιοτητας 119
5.4 **ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗ ΜΟΝΑΔΩΝ ΠΡΟΓΡΑΜΜΑΤΟΣ** 122

ΚΕΦΑΛΑΙΟ 6 : **ΈΛΕΓΧΟΣ ΚΑΙ ΔΙΟΡΘΩΣΗ ΣΦΑΛΜΑΤΩΝ** 123

6.1 **ΓΕΝΙΚΟ ΠΛΑΙΣΙΟ ΕΛΕΓΧΟΥ** 129
6.1.1 Προγραμματισμός του ελέγχου 130
6.2 **ΤΕΧΝΙΚΕΣ ΕΛΕΓΧΟΥ** 131

6.2.1 Στρατηγική του μαύρου κουτιού ... *133*
6.2.2 Προσέγγιση της ισοδύναμης διαμέρισης ... *134*
6.2.2.1 Προσέγγιση συνοριακών τιμών ... 136
6.2.2.2 Προσέγγιση αιτίου-Αποτέσματος ... 137
6.2.2.3 Προσέγγιση μαντέματος ... 137
6.2.3 Στρατηγική του γυάλινου κουτιού ... *138*
6.3 **ΕΚΤΕΛΕΣΗ ΕΛΕΓΧΟΥ** ... **140**
6.3.1 Έλεγχος μονάδας ... *140*
6.3.1.1 Έλεγχος συνένωσης ... 142
6.3.1.2 Πλεονεκτήματα και μειονεκτήματα ... 144
6.3.1.3 Έλεγχος συστήματος ... 145
6.3.1.4 Έλεγχος αποδοχής ... 146
6.4 **ΑΝΑΦΟΡΕΣ ΕΛΕΓΧΟΥ** ... **147**
6.5 **ΔΙΟΡΘΩΣΗ ΣΦΑΛΜΑΤΩΝ** ... **148**

ΚΕΦΑΛΑΙΟ 7: **ΣΥΝΤΗΡΗΣΗ ΣΥΣΤΗΜΑΤΟΣ** ... 150

7.1 **ΔΥΝΑΜΙΚΗ ΤΗΣ ΕΞΕΛΙΞΗΣ ΤΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ** ... **152**
7.2 **ΣΥΝΤΗΡΗΣΗ ΛΟΓΙΣΜΙΚΟΥ** ... **155**
7.3 **ΠΡΟΒΛΕΨΗ ΣΥΝΤΗΡΗΣΗΣ** ... **160**
7.4 **ΔΙΑΔΙΚΑΣΙΕΣ ΕΞΕΛΙΞΗΣ** ... **163**
7.5 **ΑΝΑΚΑΤΑΣΚΕΥΗ ΣΥΣΤΗΜΑΤΩΝ** ... **166**

ΚΕΦΑΛΑΙΟ 8:**ΑΝΑΠΤΥΞΗ ΔΙΚΤΥΑΚΗΣ ΕΦΑΡΜΟΓΗΣ ΜΕ ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ ΠΕΡΙΕΧΟΜΕΝΟΥ** . 172

8.1 **ΕΠΙΛΟΓΗ ΣΥΣΤΗΜΑΤΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΠΕΡΙΕΧΟΜΕΝΟΥ** (CMS) ... **173**
8.2 **ΠΕΡΙΓΡΑΦΗ ΤΟΥ** "JOOMLA!" ... **175**
8.2.1 Χαρακτηριστικά του "Joomla!" ... *175*
8.2.2 Προδιαγραφές Εγκατάστασης ... *176*
8.2.3 Η Δομή του joomla ... *176*
8.2.4 PHP ... *177*
8.2.5 MySQL ... *179*
8.2.6 CSS ... *179*
8.2.7 Σύστημα Διαχείρισης Περιεχομένου ... *180*

CASE STUDY ... 181

ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΔΙΚΤΥΑΚΟΥ ΤΟΠΟΥ ΤΟΥ ΤΜΗΜΑΤΟΣ ΕΠΔΟ ... **181**
ΜΕΘΟΔΟΛΟΓΙΑ ΑΝΑΠΤΥΞΗΣ ΚΑΙ ΦΑΣΕΙΣ ΕΦΑΡΜΟΓΩΝ ΔΙΑΔΙΚΤΥΟΥ ΓΙΑ ΤΟ SITE ... **182**
ΒΕΛΤΙΩΣΕΙΣ / ΕΠΕΚΤΑΣΕΙΣ ΣΥΣΤΗΜΑΤΟΣ ... **199**

ΣΥΜΠΕΡΑΣΜΑΤΑ ... 200

ΠΑΡΑΡΤΗΜΑ ... 202

ΚΕΦΑΛΑΙΟ 1 ... 206

ΙΣΤΟΣΕΛΙΔΑ ΤΟΥ ΠΑΡΑΡΤΗΜΑΤΟΣ ΑΜΑΛΙΑΔΟΣ ΤΟΥ Τ.Ε.Ι. ΠΑΤΡΩΝ – ΣΥΝΟΠΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΚΑΙ ΔΙΑΡΘΡΩΣΗ ... 206

ΓΕΝΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ... **206**

ΔΙΕΠΑΦΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΜΕ ΤΟΥΣ ΧΡΗΣΤΕΣ **207**
ΧΡΗΣΤΕΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ **207**
ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΥΨΗΛΟΥ ΕΠΙΠΕΔΟΥ ΑΠΑΙΤΗΣΕΩΝ **208**
1.1.1 Μεθοδολογία καταγραφής απαιτήσεων *208*
ΚΕΦΑΛΑΙΟ 2 209
ΑΝΑΛΥΤΙΚΗ ΚΑΤΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ ΣΥΣΤΗΜΑΤΟΣ 209
ΑΠΑΙΤΗΣΕΙΣ **209**
2.1.1 Γενικές απαιτήσεις *209*
2.1.2 Εξειδίκευση απαιτήσεων *211*
ΚΕΦΑΛΑΙΟ 3 ..... **215**
ΔΙΑΓΡΑΜΜΑΤΑ UML **ΣΥΣΤΗΜΑΤΟΣ** **215**
ΒΙΒΛΙΟΓΡΑΦΙΑ 219
ΠΗΓΕΣ 222

ΠΕΡΙΛΗΨΗ

Η πτυχιακή αυτή εργασία ασχολείται με το σχεδιασμό και την ανάπτυξη του δικτυακού τόπου του τμήματος Εφαρμογών Πληροφορικής στη Διοίκηση και Οικονομία του Παραρτήματος Αμαλιάδας του Ανώτατου Τεχνολογικού Εκπαιδευτικού Ιδρύματος Πατρών. Ο προϋπάρχων δικτυακός τόπος είχε κατασκευασθεί πρόχειρα, για την κάλυψη ορισμένων άμεσων αναγκών που είχαν δημιουργηθεί με το πέρασμα του χρόνου στο Ίδρυμα. Αυτό που έπρεπε να δημιουργηθεί λοιπόν, ήταν κάτι πιο σύγχρονο και συνάμα πιο εύχρηστο και δυναμικό.

Γίνεται εκτενής αναφορά σε αυτή τη νέα και πολυσυζητημένη επιστήμη της Μηχανικής που ονομάζεται «Τεχνολογία Λογισμικού», καθώς η κατασκευή λογισμικού διαπιστώθηκε ότι αποτελεί ιδιαίτερα σημαντικό παράγοντα για τη δημιουργία άρτιων Πληροφοριακών Συστημάτων. Για το λόγο αυτό αναλύονται διάφορα μοντέλα κύκλου ζωής λογισμικού και γίνεται αναφορά στις φάσεις ανάπτυξης που αυτά υποστηρίζουν. Επιπλέον, αναφέρονται οι διαδικασίες, οι οποίες λαμβάνουν χώρα κατά τις γενικές φάσεις «σύλληψη, κατασκευή, χρήση - συντήρηση, απόσυρση», προσδιορίζοντας τις επιμέρους φάσεις στις οποίες αυτές αναλύονται, τα προϊόντα που παράγονται σε καθεμία από αυτές, καθώς και τη σειρά εκτέλεσής τους.

Καθ’ όλη τη διάρκεια της έρευνας που πραγματοποιήθηκε επάνω στην Τεχνολογία Λογισμικού καθώς επίσης και στο σύστημα διαχείρισης περιεχομένου ανοικτού κώδικα «Joomla!», αποφασίστηκε ότι ο δικτυακός τόπος που επρόκειτο να κατασκευαστεί, θα είχε ως πρότυπο το μοντέλο ανάπτυξης λογισμικού «καταρράκτη», χρησιμοποιώντας τεχνολογίες αιχμής ανοιχτού κώδικα και ειδικότερα, το ανωτέρω συγκεκριμένο δημοφιλές σύστημα διαχείρισης περιεχομένου (CMS), λαμβάνοντας υπόψη τις προσφερόμενες δυνατότητες και την αναγκαία παραμετροποίηση, στο πλαίσιο των τεθέντων προδιαγραφών.

Ολοκληρώνοντας, η εργασία αναλύει τον τρόπο υιοθέτησης του επιλεγμένου μοντέλου κύκλου ζωής λογισμικού κατά την ανάπτυξη του δικτυακού τόπου του Ιδρύματος, καθώς επίσης περιγράφει τη διαδικασία υλοποίησής της με χρήση «Joomla!».

INTRODUCTION

The thesis work is concerned with the design and development of the website of the Department of Applied Informatics in Administration and Economy of Annex Amaliada of the Technological Educational Institute of Patras. The preexisting site was constructed makeshift, to meet some immediate needs that have arisen over the years to the Foundation. What have we created was something more modern, yet user friendly and dynamic.

Extensive reference to this new and much vaunted science of mechanics called "Software", as the software maker found that a particularly important factor in creating robust Information Systems. For this reason analyzed various models of software life cycle and refer to these phases of development support. In addition, indicate the procedures which take place during the general phases "conception, construction, use - maintenance, removal," identifying the individual phases in which they analyzed the products produced in each of them and turn their execution.

Throughout the course of the investigation which took place on the Software as well as content management system open source «Joomla!», Decided that the site to be built, would have the standard software development model "waterfall", using open source technologies and in particular, this specific popular content management system (CMS), taking into account the opportunities offered and the necessary configuration, in the stated specifications.

In conclusion, the work discusses how the adoption of selected software life cycle model in developing the Foundation's website and also describes the implementation process of using «Joomla!».

ΕΙΣΑΓΩΓΗ

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

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

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

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

Τέτοιου είδους τεχνικές έχουν γίνει πλέον μέρος της τεχνολογίας λογισμικού και χρησιμοποιούνται ευρύτατα σήμερα. Ωστόσο, όπως έχει αυξηθεί η δυνατότητά μας να παράγουμε λογισμικό, αντίστοιχα έχει αυξηθεί και η πολυπλοκότητα των συστημάτων λογισμικού που χρειαζόμαστε. Οι νέες τεχνολογίες που προκύπτουν από την σύγκλιση υπολογιστικών και τηλεπικοινωνιακών συστημάτων, καθώς και σύνθετων γραφικών διασυνδέσεων χρήστη, θέτουν νέες απαιτήσεις στους μηχανικούς του λογισμικού. Καθώς πολλές εταιρείες δεν εφαρμόζουν ακόμα αποτελεσματικά τις τεχνικές της τεχνολογίας λογισμικού, πάρα πολλά έργα εξακολουθούν να παράγουν λογισμικό λειτουργικά αναξιόπιστο, με καθυστερήσεις στο χρόνο παράδοσης, και με υπερβάσεις του προϋπολογισμού.

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

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

Οι μηχανικοί λογισμικού μπορούν δικαιωματικά να υπερηφανεύονται για τα επιτεύγματά τους. Χωρίς σύνθετα συστήματα λογισμικού δεν θα είχαμε εξερευνήσει το διάστημα, δε θα είχαμε το Διαδίκτυο και τις σύγχρονες τηλεπικοινωνίες, και όλοι οι τρόποι ταξιδιού θα ήταν πιο επικίνδυνοι και ακριβοί. Η τεχνολογία λογισμικού έχει προσφέρει πολλά και καθώς ωριμάζει ως επιστημονικό πεδίο, η συνεισφορά της στον $21^{ο}$ αιώνα θα είναι ακόμα μεγαλύτερη.[**C**]

Ο σκοπός ανάπτυξης ενός δικτυακού τόπου Περιεχομένου στη περίπτωσή μου ήταν ο πιο κατάλληλος, καθώς ο στόχος μου για την ανάπτυξη και ανανέωση του δικτυακού τόπου του ΤΕΙ Αμαλιάδας καθορίστηκε, πέρα από την έμφυτη διάθεση για δημιουργία που είχα, κυρίως από τις ανάγκες της ενημέρωσης προς στους φοιτητές, στην ευρύτερή τους διάσταση και κατά δεύτερο λόγο από την ανάγκη πληρότητας και συμμετρικής ανάπτυξής του site.

Για την ανάπτυξη της ιστοσελίδας χρειάστηκε να καλυφθούν οι ανάγκες τις οποίες είχε ο κάθε φοιτητής, καθηγητής, χρήστες που ενδιαφερόντουσαν για τη σχολή, κοκ. Γι'αυτό λοιπόν κι εγώ πρώτα περιηγήθηκα στα ιδιαίτερα χαρακτηριστικά που είχε το ΤΕΙ μου και ύστερα προχώρησα στην ανάπτυξη του ιστοτόπου μου.

Τα χαρακτηριστικά που έχει το τμήμα του ΤΕΙ που φοιτώ είναι τα ακόλουθα:

- Το περιεχόμενο σπουδών του τμήματος καλύπτει τα γνωστικά αντικείμενα της Πληροφορικής, της Διοικητικής επιστήμης, των Οικονομικών και του marketing με εξειδίκευση στις εφαρμογές της πληροφορικής στη Διοίκηση και την Οικονομία στο πλαίσιο της οργάνωσης και λειτουργίας επιχειρήσεων και οργανισμών, τόσο του δημόσιου, όσο και του ιδιωτικού τομέα.
- Η δομή του προγράμματος σπουδών περιλαμβάνει τα γνωστικά αντικείμενα που στοχεύουν στη παροχή βασικών γνώσεων κατά τη διάρκεια των πρώτων εξαμήνων, ενώ στη συνέχεια και προς την ολοκλήρωσή του, τα γνωστικά αντικείμενα αφορούν πιο εξειδικευμένους επιστημονικά φορείς.
- Έχει σαν αποστολή την εκπαίδευση των φοιτητών ώστε να καταστούν επιστήμονες οι οποίοι με τη θεωρητική και εφαρμοσμένη κατάρτισή τους να μεταφέρουν, να χρησιμοποιούν και να προάγουν σύγχρονες τεχνολογίες ΤΠΕ στον τομέα της διοίκησης επιχειρήσεων και οργανισμών.
- Επίσης εντάσσεται η δυνατότητα δημιουργίας ενός κλίματος ερευνητικών δραστηριοτήτων και επιστημονικής αναζήτησης με σκοπό την ανάπτυξη ακαδημαϊκής κουλτούρας και την προαγωγή του επιπέδου των σπουδών.
- Μετά την ολοκλήρωση των σπουδών του οι πτυχιούχοι αποκτούν τις απαραίτητες επιστημονικές και τεχνολογικές γνώσεις, οι οποίες τους επιτρέπουν να δραστηριοποιηθούν επαγγελματικά, υποστηρίζοντας με επιτυχία την αξιοποίηση των τεχνολογιών της Πληροφορικής και των Επικοινωνιών σε όλους τους τομείς της οικονομικής και επιχειρηματικής δραστηριότητας.

- Επιπλέον παρέχει στους πτυχιούχους υψηλό επίπεδο γνώσεων στα γνωστικά επιστημονικά αντικείμενα της Πληροφορικής Επιστήμης, της Διοίκησης και Οργάνωσης Επιχειρήσεων, του Marketing, της Χρηματοδότησης και της Διαχείρισης Ανθρωπίνων Πόρων και δύναται να σχεδιάσουν, αναπτύξουν και εφαρμόσουν σύγχρονα Πληροφοριακά Συστήματα Διοίκησης.
- Ακόμα οι πτυχιούχοι του τμήματος απασχολούνται τόσο στο δημόσιο όσο και στον ιδιωτικό τομέα, είτε αυτοδύναμα, είτε σε συνεργασία με άλλους επαγγελματίες και επιστήμονες, σε όλους τους τομείς αξιοποίησης των Πληροφοριακών Συστημάτων Διοίκησης, σε θέματα που σχετίζονται με την μελέτη, την ανάπτυξη, τη διαχείριση, την εφαρμοσμένη έρευνα, την εκπαίδευση και την κατάρτιση.
- Τέλος, ένα ακόμη προνόμιο που παρέχει στους πτυχιούχους είναι η παρακολούθηση προγραμμάτων σπουδών για απόκτηση μεταπτυχιακού ή διδακτορικού διπλώματος σε πανεπιστημιακά ιδρύματα της χώρας μας ή του εξωτερικού για την εμβάθυνση των γνώσεών τους επάνω στο αντικείμενο που τους ενδιαφέρει.

ΚΕΦΑΛΑΙΟ 1: Γνωριμία με την Τεχνολογία Λογισμικού

1.1 Υπολογιστές και Λογισμικό

Ένα από τα σημαντικότερα γεγονότα που σηματοδότησαν τον αιώνα που πέρασε ήταν η εφεύρεση του Ηλεκτρονικού Υπολογιστή (Η/Υ). Με τη βοήθεια του Η/Υ έγινε δυνατή η αυτοματοποίηση της εκτέλεσης πολλών κουραστικών, ανιαρών και επιρρεπών σε λάθη εργασιών, καθώς και η εκτέλεση άλλων, η οποία στο παρελθόν ήταν πρακτικά αδύνατη. Από την εποχή που κατασκευάστηκαν οι πρώτοι Η/Υ μέχρι σήμερα σημειώθηκε τεράστια βελτίωση των χαρακτηριστικών και των δυνατοτήτων τους. Κανένα άλλο ανθρώπινο κατασκεύασμα δε σημείωσε τόσο σημαντική πρόοδο σε τόσο μικρό χρονικό διάστημα. Ένα από τα πρακτικά αποτελέσματα αυτής της εξέλιξης ήταν ότι οι Η/Υ έγιναν προσιτοί σε μεγάλες μάζες ανθρώπων και αναφαίρετο εργαλείο της καθημερινής επαγγελματικής αλλά και ιδιωτικής ζωής για πολλούς από αυτούς. Παράλληλα, έγινε δυνατή η ενσωμάτωση Η/Υ σε πάρα πολλές συσκευές καθημερινής χρήσης, χωρίς αυτό να είναι πάντα αντιληπτό από τους ίδιους τους χρήστες. Σήμερα, σε πολλές από τις καθημερινές μας εργασίες χρησιμοποιούμε Η/Υ χωρίς να το γνωρίζουμε, ενώ συχνά η ίδια μας η ζωή εξαρτάται από Η/Υ (υγεία, μέσα μεταφοράς, υπηρεσίες όπως έλεγχος οδικής και εναέριας κυκλοφορίας κ.ά.).

Η σημερινή εποχή μπορεί να χαρακτηριστεί ως μεταβατική σε μια νέα κατάσταση στην οποία όλοι οι Η/Υ θα είναι διασυνδεδεμένοι μέσω δικτύων και θα εκτελούν σύνθετες εργασίες. Πολλές από τις σύγχρονες δικτυακές εφαρμογές μπορούν να μεταβάλουν κρίσιμες πλευρές του πολιτισμού μας, όπως την επικοινωνία, την εκπαίδευση και την κατάρτιση, αλλά και αυτή την ίδια τη λειτουργία των δημοκρατικών πολιτευμάτων. Η νέα κατάσταση που διαμορφώνεται αναφέρεται ως *κοινωνία της πληροφορίας* (information society) και έχουμε ήδη εισέλθει στο εξελικτικό της στάδιο με το Internet και τις περί αυτό εφαρμογές να παίζουν πρωταγωνιστικό ρόλο στη διαδικασία. Όλες αυτές οι εξελίξεις γίνονται δυνατές χάρη στην ύπαρξη και λειτουργία ενός συνόλου πολύπλοκων εφαρμογών λογισμικού.

Η εξάπλωση του ηλεκτρονικού υπολογιστή σε ολοένα και περισσότερες πλευρές της ανθρώπινης ζωής δε θα ήταν δυνατή χωρίς τη χρήση λογισμικού. Ο ηλεκτρονικός υπολογιστής ως συσκευή μπορεί μόνο να εκτελέσει ορισμένες πολύ απλές λειτουργίες με πάρα πολύ υψηλή ταχύτητα, όμως με τρόπο ιδιαίτερα δυσπρόσιτο στον άνθρωπο. Το

λογισμικό είναι εκείνο που καθιστά χρήσιμη και αποδίδει στοιχεία «συμπεριφοράς» στη συσκευή Η/Υ. Ο άνθρωπος δεν αξιοποιεί τον ηλεκτρονικό υπολογιστή άμεσα ως συσκευή, αλλά μόνο μέσω του λογισμικού. Έχοντας κατά νου τα παραπάνω, δεν είναι εύκολο να δοθεί ένας πλήρης και καθολικά αποδεκτός ορισμός της έννοιας «λογισμικό». Η πρακτική αξία, αλλά και η διαχρονικότητα ενός θεωρητικού ορισμού μπορεί να αμφισβητηθεί σχετικά εύκολα. Μια ικανοποιητική προσέγγιση είναι ο ορισμός του λογισμικού ως ακολούθως, τον οποίο και θα αποδεχτούμε ως επαρκή στο βιβλίο αυτό:

_ Λογισμικό:[1]

(1) *εντολές (προγράμματα ηλεκτρονικού υπολογιστή) οι οποίες όταν εκτελούνται επιτυγχάνουν επιθυμητά αποτελέσματα και επιδόσεις,*

(2) *δομές δεδομένων που επιτρέπουν σε προγράμματα να διαχειριστούν με επάρκεια πληροφορίες και*

(3) *κείμενα, διαγράμματα κτλ. που περιγράφουν τη λειτουργία και χρήση των προγραμμάτων.*

1.2 Τεχνικές κατασκευές και Λογισμικό

Το λογισμικό είναι ένα πολύπλοκο τεχνικό κατασκεύασμα, το οποίο δεν έχει αυτοτελή υπόσταση, παρά μόνο όταν χρησιμοποιείται για να καθοδηγήσει έναν ηλεκτρονικό υπολογιστή στην πραγματοποίηση συγκεκριμένων λειτουργιών. Παρά τις αρκετές ομοιότητες που μπορεί κανείς να αναζητά συχνά μεταξύ λογισμικού και λοιπών τεχνικών κατασκευών, υπάρχουν και σημαντικές διαφορές. Η πρώτη είναι η μη απτή φύση του λογισμικού. Μια τεχνική κατασκευή είναι ορατή και απτή, ενώ το λογισμικό δεν είναι αυτό καθεαυτό «ορατό». Μόνο τα αποτελέσματα της χρήσης του μπορούν να είναι αντιληπτά. Η δομή του λογισμικού, τόσο σε μικροσκοπικό όσο και σε μακροσκοπικό επίπεδο, είναι και αυτή ένα νοητό κατασκεύασμα, που μπορεί να γίνει με διαφορετικούς τρόπους αντιληπτό.

[1] Ανατρέξτε στο Βιβλίο του Βασίλειου Βεσκούκη, Τεχνολογία Λογισμικού Ι,σελ **1**

Η δεύτερη σημαντική διαφορά μπορεί να περιγραφεί από μια παρομοίωση: Σε αντίθεση με τα τεχνικά έργα, η κατασκευή των οποίων συνήθως ακολουθεί μια προκαθορισμένη οδό, γνωστή από την αρχή, η ανάπτυξη του λογισμικού ομοιάζει με *σκόπευση κινούμενου στόχου από κινούμενο έδαφος και με όπλο που συνεχώς αλλάζει τη συμπεριφορά του*. Ο στόχος είναι κινούμενος γιατί οι απαιτήσεις των χρηστών συνεχώς μμεταβάλλονται, ακόμα και μέσα στη διαδικασία ανάπτυξης μιας εφαρμογής που προορίζεται να τις ικανοποιήσει. Το έδαφος είναι κινούμενο γιατί το περιβάλλον ανάπτυξης του λογισμικού είναι και το ίδιο συνεχώς εξελισσόμενο μαζί με το υλικό, αλλά και μαζί με τις επιλογές και την έκβαση των μη τεχνικών διαμαχών στο χώρο της αγοράς τεχνογνωσίας και τεχνολογίας πληροφορικής. Σαν να μην έφταναν τα παραπάνω, το όπλο με το οποίο γίνεται η «σκόπευση», δηλαδή οι μμεθοδολογίες, τα εργαλεία και τα περιβάλλοντα ανάπτυξης και λειτουργίας του λογισμικού, είναι επίσης ραγδαία μμεταβαλλόμενα με το χρόνο. Όλοι μας γινόμαστε μάρτυρες ενός καταιγισμού προϊόντων, εξέλιξης λειτουργικών συστημάτων, γλωσσών προγραμματισμού, περιβαλλόντων και τεχνολογιών ανάπτυξης. Ο καταιγισμός αυτός φαίνεται να μην έχει ορατό τέλος, μιας και είναι οι νόμοι του ανταγωνισμού που σε πολλές περιπτώσεις προωθούν τις εξελίξεις, αλλά και η ίδια η πρόοδος της τεχνολογίας των υπολογιστών, η οποία είναι τουλάχιστον εντυπωσιακή.

Μολονότι η κατασκευή πολλών τεχνικών έργων είναι δυνατό να τυποποιηθεί σε αρκετά μεγάλο βαθμό, δεν ισχύει το ίδιο με την ανάπτυξη του λογισμικού. Η ανάπτυξη αυτή μέχρι σήμερα δεν έχει γίνει δυνατό να αυτοματοποιηθεί και το λογισμικό παραμένει ένα από τα πολυπλοκότερα και δυσκολότερα τεχνικά κατασκευάσματα του ανθρώπου, στην κατασκευή του οποίου συναντώνται επί μακρόν σημαντικά προβλήματα, τα οποία από πολλούς χαρακτηρίζονται ως «χρόνια».

1.3 Κρίση Λογισμικού

Το σύνολο αυτών των χρόνιων προβλημάτων αναφέρεται ως «κρίση λογισμικού». Ενδεχομένως, η χρήση όρων όπως «κρίση» ή «χρόνια προβλήματα» να μπορεί να χαρακτηριστεί υπερβολική, αυτό όμως δεν αναιρεί ούτε τη σοβαρότητα ούτε την παρατεταμένη διάρκεια εκδήλωσης των προβλημάτων που έχουν καταγραφεί και καθημερινά επιβεβαιώνονται στην ανάπτυξη του λογισμικού. Είναι χαρακτηριστικό ότι το λογισμικό είναι ένα από τα ελάχιστα ανθρώπινα κατασκευάσματα που πωλείται «ως έχει», χωρίς *καμία απολύτως* εγγύηση για τις ζημιές που μπορεί να προκαλέσει η χρήση του, οσοδήποτε

σημαντικές και αν είναι *αυτές*. Ο Πίνακας 1.1 απεικονίζει τα σημαντικότερα από τα προβλήματα αυτά.

Εξαιρετικά δύσκολη διαδικασία κατασκευής.	Δεν είναι πάντα σαφές ποια βήματα πρέπει να γίνουν, με ποια σειρά, με ποια ενδιάμεσα προϊόντα κτλ.
Ανεπαρκής ή και κακή ποιότητα τελικού προϊόντος	Λάθη στην κατασκευή, μη ικανοποίηση του σκοπού.
Μη τήρηση χρονοδιαγραμμάτων.	Υπερβολικές και «αδικαιολόγητες» καθυστερήσεις.
Υπερβάσεις προϋπολογισμών.	Κακές αρχικές εκτιμήσεις κόστους. Τελικά προϊόντα με πολλαπλάσιο κόστος από το αρχικά προϋπολογισθέν.
Μεγάλη δυσκολία και συνεπαγόμενο κόστος συντήρησης.	Παρενέργειες μεταβολών σε στοιχεία που πριν λειτουργούσαν, πρόχειρες λύσεις.
Δύσκολη κατανόηση εγγράφων, σχεδίων κτλ. από διαφορετικούς κατασκευαστές.	Στην πράξη, η κατανόηση ενός συστήματος λογισμικού από τρίτους, πλην των κατασκευαστών του, είναι συχνά αδύνατη ή ιδιαίτερα ασύμφορη.

ΠΙΝΑΚΑΣ 1 *Μερικά βασικά σημεία της "κρίσης λογισμικού"*

Προβλήματα όπως τα παραπάνω έχουν εντοπιστεί εδώ και δεκαετίες από την κοινότητα κατασκευαστών και ακαδημαϊκών ερευνητών στη γνωστική περιοχή του λογισμικού και έχουν διατυπωθεί με πολλούς τρόπους και με πολλές ευκαιρίες. Συχνά, νέες τεχνολογίες ή προϊόντα που προτείνονται για την ανάπτυξη του λογισμικού κάνουν επίκληση των προβλημάτων αυτών, ισχυριζόμενα ότι διαθέτουν ικανοποιητικές λύσεις. Για ένα διάστημα, οι λύσεις αυτές φέρονταν ως «ασημένια σφαίρα» που θα σκότωνε το «τέρας» των προβλημάτων ανάπτυξης λογισμικού. Κάτι τέτοιο δεν έγινε και η φιλοσοφία της «ασημένιας σφαίρας» εγκαταλείφθηκε, για να πάρουν τη θέση της πιο συνετές επιστημονικές προσεγγίσεις στην ανάπτυξη του λογισμικού.

Ως αποτέλεσμα, αναπτύχθηκε ένας ειδικός κλάδος της επιστήμης της πληροφορικής, που ονομάστηκε *Τεχνολογία Λογισμικού* (Software Engineering). Πρόσφατα προτάθηκε η Τεχνολογία Λογισμικού να από-τελέσει εξειδίκευση της επιστήμης του μμηχανικού, οπότε μια πιο εύστοχη απόδοση στα ελληνικά είναι αυτή της *Μηχανικής Λογισμικού*. Στο βιβλίο αυτό θα αποδεχτούμε την απόδοση «Τεχνολογία Λογισμικού» ως επικρατέστερη για την ελληνική πραγματικότητα και θα αναφερόμαστε στην ίδια τεχνική επιστημονική περιοχή,

ανεξάρτητα από το αν πρόκειται για τεχνολογία, για μμηχανική ή για. τέχνη, όπως συχνά υποστηρίζεται σε ακαδημαϊκές συζητήσεις.(35)

1.3.1 Τι είναι λογισμικό

Πολλοί εξισώνουν τον όρο «λογισμικό» (software) με τα προγράμματα υπολογιστών. Λογισμικό δεν είναι μόνο τα προγράμματα αλλά και όλη η σχετική τεκμηρίωση, καθώς και τα δεδομένα διευθέτησης που απαιτούνται για να κάνουν τα προγράμματα να λειτουργούν σωστά. Ένα σύστημα λογισμικού αποτελείται συνήθως από ένα σύνολο προγραμμάτων, αρχείων διευθέτησης για την ρύθμιση προγραμμάτων, τεκμηρίωσης του συστήματος η οποία περιγράφει τη δομή του, τεκμηρίωσης χρήστη η οποία εξηγεί τον τρόπο χρήσης του συστήματος, καθώς και ιστοτόπους από όπου οι χρήστες μπορούν να κατεβάζουν πρόσφατες πληροφορίες για το προϊόν.

Οι μηχανικοί λογισμικού ασχολούνται με την ανάπτυξη προϊόντων λογισμικού, δηλαδή λογισμικού που μπορεί να πωληθεί σε κάποιον πελάτη. Υπάρχουν δύο βασικοί τύποι προϊόντων λογισμικού:

1. Προϊόντα γενικής χρήσης. Πρόκειται για αυτοτελή συστήματα λογισμικού που παράγονται από κάποιον κατασκευαστή και πωλούνται στην αγορά σε οποιονδήποτε πελάτη είναι σε θέση να τα αγοράσει. Παραδείγματα προϊόντων αυτού του τύπου είναι το λογισμικό των προσωπικών υπολογιστών όπως οι βάσεις δεδομένων, οι επεξεργαστές κειμένου, τα σχεδιαστικά πακέτα, και τα εργαλεία παρακολούθησης έργων.
2. Προϊόντα κατά παραγγελία. Πρόκειται για συστήματα λογισμικού που έχουν παραγγελθεί από ένα συγκεκριμένο πελάτη. Ένας ανάδοχος κατασκευαστής λογισμικού αναπτύσσει το ζητούμενο λογισμικό ειδικά για το συγκεκριμένο πελάτη. Παραδείγματα λογισμικού αυτού του τύπου είναι τα συστήματα ελέγχου ηλεκτρονικών συσκευών, τα συστήματα για την υποστήριξη μιας συγκεκριμένης επιχειρηματικής διαδικασίας, καθώς και τα συστήματα ελέγχου εναέριας κυκλοφορίας.

Μια σημαντική διαφορά μεταξύ αυτών των δύο τύπων λογισμικού είναι ότι, στα προϊόντα γενικής χρήσης, ο κατασκευαστής του λογισμικού έχει τον έλεγχο των προδιαγραφών του

λογισμικού. Για τα προϊόντα κατά παραγγελία οι προδιαγραφές συνήθως διαμορφώνονται και ελέγχονται από την πλευρά του αγοραστή, και οι κατασκευαστές του λογισμικού πρέπει να εργαστούν σύμφωνα με αυτές.

Πάντως η διαχωριστική γραμμή μεταξύ των δύο τύπων προϊόντων γίνεται σήμερα όλο και πιο θολή. Όλο και περισσότερες εταιρείες λογισμικού ξεκινούν με ένα σύστημα γενικής χρήσης και το προσαρμόζουν στις ιδιαίτερες ανάγκες ενός συγκεκριμένου πελάτη. Τα συστήματα προγραμματισμού επιχειρησιακών πόρων (ERP), όπως το σύστημα SAP, είναι τα καλύτερα παραδείγματα αυτής της προσέγγισης. Ένα μεγάλο και σύνθετο σύστημα προσαρμόζεται για μια επιχείρηση, ενσωματώνοντας πληροφορίες για επιχειρηματικούς κανόνες και διαδικασίες, απαιτούμενες αναφορές, κοκ.

1.3.2 Τι είναι Τεχνολογία Λογισμικού

Η Τεχνολογία Λογισμικού είναι ένας τεχνικός κλάδος που ασχολείται με όλες τις πτυχές της παραγωγής λογισμικού, από τα πρώτα στάδια της εξαγωγής προδιαγραφών ενός συστήματος λογισμικού μέχρι τη συντήρηση του συστήματος μετά την διάθεσή του για χρήση. Ο ορισμός αυτός περιέχει δύο φράσεις-κλειδιά:

1. Είναι τεχνικός κλάδος. Δουλειά των μηχανικών είναι να κάνουν τα πράγματα να δουλεύουν. Εφαρμόζουν θεωρίες, μεθόδους, και χρησιμοποιούν εργαλεία, όμως το κάνουν με επιλεκτικό τρόπο και προσπαθούν πάντοτε να ανακαλύπτουν λύσεις σε προβλήματα ακόμα και όπου δεν υπάρχουν κατάλληλες θεωρίες και μέθοδοι. Οι μηχανικοί αναγνωρίζουν επίσης ότι πρέπει να εργάζονται κάτω από συγκεκριμένους εταιρικούς και οικονομικούς περιορισμούς, και επομένως αναζητούν λύσεις μέσα στα πλαίσια αυτών των περιορισμών.

2. Όλες οι πτυχές της παραγωγής λογισμικού. Η τεχνολογία λογισμικού δεν ασχολείται μόνο με τις τεχνικές διαδικασίες της ανάπτυξης λογισμικού, αλλά εκτείνεται και σε δραστηριότητες όπως η διαχείριση έργου λογισμικού, καθώς και η ανάπτυξη εργαλείων, μεθόδων, και θεωριών για την υποστήριξη της παραγωγής λογισμικού.

Γενικά, οι μηχανικοί λογισμικού υιοθετούν μια συστηματική και οργανωμένη προσέγγιση στη δουλειά τους, καθώς αυτός είναι συχνά ο πιο αποτελεσματικός τρόπος για να παραχθεί

λογισμικό υψηλής ποιότητας. Όμως το βασικό μέλημα της δουλειάς του μηχανικού είναι η επιλογή της πιο κατάλληλης μεθόδου για τις συγκεκριμένες κάθε φορά περιστάσεις, οπότε μια περισσότερο δημιουργική και λιγότερη τυπική προσέγγιση στην ανάπτυξη μπορεί να είναι αποτελεσματική σε μερικές περιπτώσεις. Ο λιγότερο τυπικός τρόπος ανάπτυξης προσφέρεται ιδιαίτερα για την ανάπτυξη συστημάτων για τον Ιστό, η οποία απαιτεί ένα μίγμα δεξιοτήτων ανάπτυξης λογισμικού και γραφικών τεχνών.

1.3.3 Ποια η διαφορά μεταξύ τεχνολογίας λογισμικού και επιστήμης των υπολογιστών;

Κατά βάση, η επιστήμη των υπολογιστών ενδιαφέρεται για τη θεωρία και τις μεθόδους που συγκροτούν το υπόβαθρο των υπολογιστών και των συστημάτων λογισμικού, ενώ η τεχνολογία λογισμικού ενδιαφέρεται για τα πρακτικά προβλήματα της παραγωγής λογισμικού. Κάποια γνώση της επιστήμης των υπολογιστών είναι απαραίτητη για τους μηχανικούς λογισμικού, όπως ακριβώς κάποια γνώση της Φυσικής είναι απαραίτητη για τους ηλεκτρολόγους μηχανικούς.
Ιδανικά, όλη η τεχνολογία λογισμικού θα έπρεπε να βασίζεται σε θεωρίες της επιστήμης των υπολογιστών, αλλά στην πραγματικότητα αυτό δεν ισχύει. Οι μηχανικοί λογισμικού χρειάζεται συχνά να χρησιμοποιούν περιστασιακές προσεγγίσεις για την ανάπτυξη του ζητούμενου λογισμικού. Οι κομψές θεωρίες της επιστήμης των υπολογιστών δεν μπορούν πάντα να εφαρμοστούν σε πραγματικά, περίπλοκα προβλήματα που απαιτούν μια λύση λογισμικού.

1.3.4 Ποια είναι η διαφορά μεταξύ τεχνολογίας λογισμικού και τεχνολογίας συστημάτων.

Η τεχνολογία συστημάτων ασχολείται με όλες τις πτυχές της ανάπτυξης και της εξέλιξης σύνθετων συστημάτων, στα οποία το λογισμικό παίζει ένα σημαντικό ρόλο. Επομένως, η τεχνολογία συστημάτων ενδιαφέρεται για την ανάπτυξη υλικού, για το σχεδιασμό πολιτικών και διαδικασιών, και για την εγκατάσταση του συστήματος, καθώς βέβαια και για την τεχνολογία λογισμικού. Οι μηχανικοί συστημάτων εμπλέκονται στην εξαγωγή των προδιαγραφών του συστήματος, στον ορισμό της γενικής αρχιτεκτονικής του, και κατόπιν στην ολοκλήρωση των διαφορετικών τμημάτων ώστε να δημιουργηθεί το τελικό σύστημα. Ενδιαφέρονται λιγότερο για την τεχνολογία των συστατικών στοιχείων του συστήματος (υλικό, λογισμικό, κ.λπ.).

Η τεχνολογία συστημάτων είναι ένας κλάδος παλαιότερος από την τεχνολογία λογισμικού. Οι άνθρωποι προδιαγράφουν και συναρμολογούν σύνθετα βιομηχανικά συστήματα, όπως αεροσκάφη και εργοστάσια χημικών προϊόντων, εδώ και περισσότερα από εκατό χρόνια. Καθώς όμως η παρουσία του λογισμικού στα συστήματα έχει αυξηθεί, διάφορες τεχνικές της τεχνολογίας λογισμικού όπως η μοντελοποίηση περιπτώσεων χρήσης και η διαχείριση διευθετήσεων χρησιμοποιούνται και στην τεχνολογία συστημάτων.

1.4 Τι είναι διαδικασία παραγωγής λογισμικού

Διαδικασία παραγωγής λογισμικού είναι ένα σύνολο δραστηριοτήτων και αντίστοιχων αποτελεσμάτων που παράγουν ένα προϊόν λογισμικού. Υπάρχουν τέσσερις θεμελιώδεις δραστηριότητες, κοινές σε όλες τις διαδικασίες παραγωγής λογισμικού. Οι δραστηριότητες αυτές είναι οι εξής:

1. Η εξαγωγή προδιαγραφών του λογισμικού (software specification), κατά την οποία οι πελάτες μαζί με τους μηχανικούς ορίζουν το λογισμικό που θα παραχθεί και τους περιορισμούς της λειτουργίας του.
2. Η ανάπτυξη του λογισμικού (software development), κατά την οποία σχεδιάζεται το λογισμικό και κατασκευάζονται τα σχετικά προγράμματα.
3. Η επικύρωση του λογισμικού (software validation), κατά την οποία το λογισμικό ελέγχεται ώστε να εξασφαλιστεί ότι είναι αυτό που ζητάει ο πελάτης.
4. Η εξέλιξη του λογισμικού (software evolution), κατά την οποία το λογισμικό τροποποιείται ώστε να προσαρμοστεί στις μεταβαλλόμενες απαιτήσεις του πελάτη και της αγοράς.

Κάθε τύπος συστημάτων λογισμικού χρειάζεται και διαφορετικές διαδικασίες ανάπτυξης. Το λογισμικού πραγματικού χρόνου ενός αεροσκάφους, για παράδειγμα, πρέπει να έχει προδιαγραφεί πλήρως πριν αρχίσει η ανάπτυξη, ενώ στα συστήματα ηλεκτρονικού εμπορίου οι προδιαγραφές και το πρόγραμμα συνήθως αναπτύσσονται μαζί. Κατά συνέπεια, οι παραπάνω γενικές δραστηριότητες μπορεί να οργανώνονται με διαφορετικούς τρόπους και να περιγράφονται σε διαφορετικά επίπεδα λεπτομέρειας ανάλογα τον τύπο του παραγόμενου λογισμικού. Από την άλλη πλευρά, η χρήση μιας ακατάλληλης διαδικασίας παραγωγής

λογισμικού μπορεί να υποβαθμίσει την ποιότητα ή τη χρησιμότητα του παραγόμενου προϊόντος ή και να αυξήσει το κόστος ανάπτυξης.

1.4.1 Τι είναι μοντέλο διαδικασίας παραγωγής λογισμικού;

Μοντέλο διαδικασίας παραγωγής λογισμικού ονομάζεται μια απλοποιημένη περιγραφή κάποιας διαδικασίας παραγωγής λογισμικού, η οποία παρουσιάζει μια συγκεκριμένη άποψη της διαδικασίας αυτής. Ένα μοντέλο διαδικασίας μπορεί να περιλαμβάνει δραστηριότητες που αποτελούν μέρος της διαδικασίας παραγωγής λογισμικού, προϊόντα λογισμικού, και τους ρόλους των ατόμων που συμμετέχουν στο έργο του λογισμικού. Παραδείγματα μερικών τύπων μοντέλων διαδικασιών παραγωγής λογισμικού που μπορεί να δημιουργηθούν είναι τα εξής:

1. Μοντέλο ροής εργασιών (workflow). Δείχνει την ακολουθία των δραστηριοτήτων της διαδικασίας, καθώς και τις εισόδους, τις εξόδους, και τις εξαρτήσεις αυτών των δραστηριοτήτων. Σε ένα τέτοιο μοντέλο οι δραστηριότητες αντιπροσωπεύουν ανθρώπινες πράξεις.
2. Μοντέλο ροής εργασιών (dataflow) ή δραστηριοτήτων (activity). Αναπαριστά τη διαδικασία ως σύνολο δραστηριοτήτων, κάθε μία από τις οποίες πραγματοποιεί κάποιο μετασχηματισμό δεδομένων. Ένα τέτοιο μοντέλο δείχνει με ποιον τρόπο τα δεδομένα εισόδου της διαδικασίας, όπως ένα σύνολο προδιαγραφών, μετασχηματίζονται σε δεδομένα εξόδου, όπως ένα σχέδιο. Οι δραστηριότητες εδώ αντιπροσωπεύουν μετασχηματισμούς που πραγματοποιούνται από ανθρώπους ή από υπολογιστές.
3. Μοντέλο ρόλων/ενεργειών (role/action). Αναπαριστά τους ρόλους των ανθρώπων που συμμετέχουν στη διαδικασία παραγωγής λογισμικού, καθώς και τις δραστηριότητες για τις οποίες είναι υπεύθυνοι.

Τα περισσότερα μοντέλα διαδικασιών παραγωγής λογισμικού βασίζονται σε ένα από τα τρία γενικά μοντέλα ή υποδείγματα ανάπτυξης λογισμικού:

1. Προσέγγιση καταρράκτη (waterfall). Η προσέγγιση αυτή παίρνει τις παραπάνω δραστηριότητες και τις αναπαριστά ως ξεχωριστές φάσεις της διαδικασίας, όπως

εξαγωγή προδιαγραφών απαιτήσεων, σχεδιασμός, του λογισμικού, υλοποίηση, δοκιμές κοκ. Μόλις οριστεί ένα στάδιο «κλείνει», και η ανάπτυξη προχωρά στο επόμενο στάδιο.

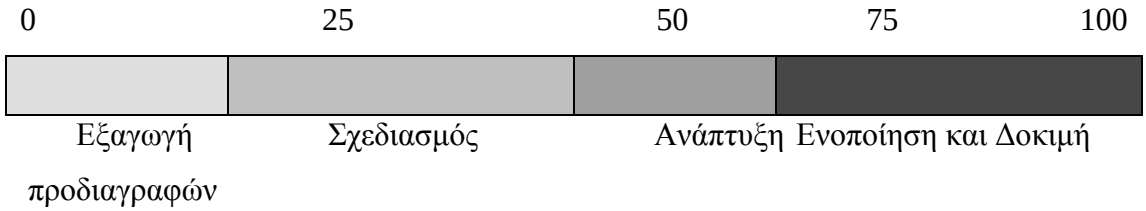
2. Επαναληπτική ανάπτυξη (iterative). Η προσέγγιση αυτή διαπλέκει τις δραστηριότητες της εξαγωγής προδιαγραφών, της ανάπτυξης, και της επικύρωσης. Αναπτύσσεται γρήγορα ένα αρχικό σύστημα από πολύ αφαιρετικές προδιαγραφές, το οποίο στη συνέχεια βελτιώνεται σύμφωνα με τα σχόλια του πελάτη, μέχρι να καταλήξει σε ένα σύστημα που να ικανοποιεί τις ανάγκες του πελάτη. Τότε το σύστημα μπορεί να παραδοθεί ως έχει, ή εναλλακτικά να υλοποιηθεί ξανά με μια πιο δομημένη προσέγγιση ώστε να παραχθεί ένα πιο ανθεκτικό και πιο συντηρήσιμο σύστημα.
3. Τεχνολογία λογισμικού βάσει συστατικών στοιχείων (component-based software engineering, CBSE). Η τεχνική αυτή προϋποθέτει ότι τα συστατικά στοιχεία του συστήματος υπάρχουν ήδη. Η διαδικασία ανάπτυξης του συστήματος εστιάζεται στην ολοκλήρωση αυτών των υφισταμένων μερών, αντί για την εξαρχής ανάπτυξή τους.

1.4.2 Ποιο είναι το κόστος της τεχνολογίας λογισμικού;

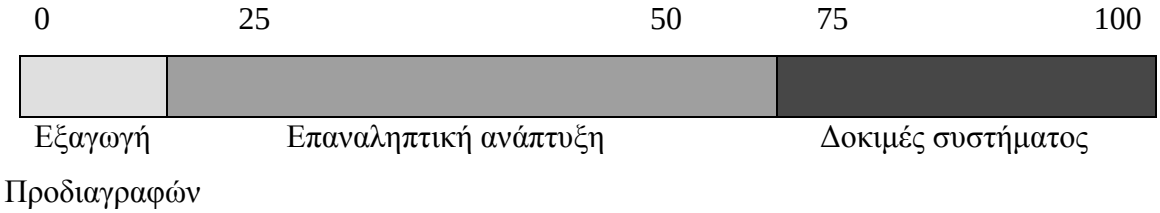
Δεν υπάρχει απλή απάντηση στην ερώτηση αυτή, καθώς η κατανομή του κόστους στις διαφορετικές δραστηριότητες της διαδικασίας παραγωγής λογισμικού εξαρτάται από τη συγκεκριμένη διαδικασία που χρησιμοποιείται από τον τύπο του λογισμικού που αναπτύσσεται. Το λογισμικού πραγματικού χρόνου, για παράδειγμα, συνήθως απαιτεί εκτενέστερη επικύρωση και δοκιμή από ότι οι εφαρμογές για τον Ιστό. Ωστόσο, κάθε μία από τις διάφορες γενικές προσεγγίσεις της ανάπτυξης λογισμικού παρουσιάζει και διαφορετική κατανομή κόστους στις επιμέρους δραστηριότητες της διαδικασίας. Αν υποτεθεί ότι το συνολικό κόστος ανάπτυξης ενός σύνθετου συστήματος λογισμικού είναι 100 μονάδες, η 1.2 εικόνα δείχνει πόσα ξοδεύονται στις δραστηριότητες των διαφόρων δραστηριοτήτων της διαδικασίας.

Στην προσέγγιση καταρράκτη, το κόστος της εξαγωγής προδιαγραφών, του σχεδιασμού, της υλοποίησης, και της ενοποίησης μπορεί να μετρηθεί χωριστά. Αξίζει να παρατηρήσετε ότι η ενοποίηση και οι δοκιμές του συστήματος είναι οι ακριβότερες δραστηριότητες ανάπτυξης. Υπό κανονικές συνθήκες, οι δύο αυτές δραστηριότητες αντιπροσωπεύουν περίπου το 40% του συνολικού κόστους ανάπτυξης, αλλά σε ορισμένα κρίσιμα συστήματα είναι πιθανό να ξεπεράσουν το 50% του κόστους ανάπτυξης

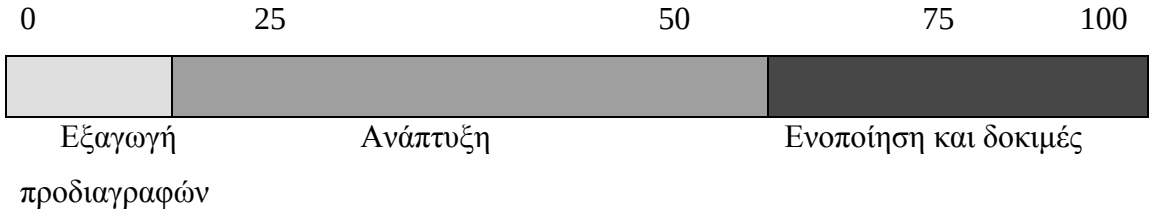
Μοντέλο Καταρράκτη



Επαναληπτική Ανάπτυξη



Ανάπτυξη λογισμικού βάσει συστατικών στοιχείων



Κόστος ανάπτυξης και εξέλιξης για μακρόβια συστήματα λογισμικού

ΠΙΝΑΚΑΣ 1.2 *Κατανομή κόστους δραστηριοτήτων της τεχνολογίας λογισμικού*

Αν η ανάπτυξη του λογισμικού γίνει με την επαναληπτική προσέγγιση, δεν υπάρχει ξεκάθαρη διαχωριστική γραμμή μεταξύ της εξαγωγής προδιαγραφών, του σχεδιασμού, και της ανάπτυξης. Το κόστος της εξαγωγής προδιαγραφών, ο σχεδιασμός, η υλοποίηση, η ενοποίηση, και οι δοκιμές διεκπεραιώνονται παράλληλα, μέσα στα πλαίσια μιας ενιαίας δραστηριότητας ανάπτυξης. Και εδώ όμως χρειάζεται μια ανεξάρτητη δραστηριότητα για τις δοκιμές του συστήματος, αφού ολοκληρωθεί η αρχική υλοποίηση.

Η κατασκευή λογισμικού βάσει συστατικών στοιχείων έχει αρχίσει να χρησιμοποιείται σε ευρεία κλίμακα σχετικά πρόσφατα. Δεν έχουμε λοιπόν ακριβείς αριθμούς για το κόστος των

διαφόρων δραστηριοτήτων ανάπτυξης λογισμικού με αυτή την προσέγγιση. Γνωρίζουμε όμως ότι το κόστος της ανάπτυξης είναι μειωμένο σε σχέση με το κόστος της ενοποίησης και των δοκιμών. Τα τελευταία είναι αυξημένα επειδή χρειάζεται να εξασφαλιστεί ότι τα συστατικά στοιχεία που χρησιμοποιούνται ικανοποιούν όντως τις προδιαγραφές τους και συνεργάζονται μεταξύ τους όπως αναμένεται.

Πέρα από το κόστος ανάπτυξης, δημιουργείται επίσης ένα κόστος από αλλαγές στο λογισμικό αφού έχει τεθεί σε χρήση. Το κόστος εξέλιξης του λογισμικού εξαρτάται σε πολύ μεγάλο βαθμό από τον τύπο του συστήματος. Για μακρόβια συστήματα λογισμικού, όπως τα συστήματα διοίκησης και ελέγχου (command and control), τα οποία μπορεί να χρησιμοποιούνται για 10 ή περισσότερα χρόνια, το κόστος εξέλιξης είναι πιθανό να ξεπεράσει το κόστος ανάπτυξης κατά ένα συντελεστή 3 ή 4, όπως φαίνεται στην τελευταία ράβδο της παραπάνω εικόνας. Όμως τα μικρότερα επιχειρηματικά συστήματα έχουν πολύ μικρότερο χρόνο ζωής και αντίστοιχα μειωμένο κόστος εξέλιξης.

Αυτές οι κατανομές κόστους ισχύουν για τα προϊόντα λογισμικού κατά παραγγελία, τα οποία προδιαγράφονται από τον πελάτη και αναπτύσσονται από τον ανάδοχο. Για τα προϊόντα λογισμικού που πωλούνται (κυρίως) για προσωπικούς υπολογιστές, το προφίλ κόστους είναι πιθανό να διαφέρει. Τέτοιου είδους προϊόντα συνήθως αναπτύσσονται από κάποιες γενικές προδιαγραφές με τη χρήση μιας προσέγγισης εξελικτικής ανάπτυξης. Επομένως το κόστος της εξαγωγής προδιαγραφών είναι σχετικά χαμηλό. Επειδή όμως προορίζονται για χρήση σε ένα φάσμα διαφορετικών συνθέσεων εξοπλισμού, πρέπει να δοκιμάζονται εκτενώς. Στην παρακάτω εικόνα παρουσιάζεται το προφίλ κόστους που θα μπορούσε να αναμένεται για τέτοια προϊόντα.

Το κόστος εξέλιξης των προϊόντων λογισμικού γενικής χρήσης είναι ιδιαίτερα δύσκολο να εκτιμηθεί. Σε πολλές περιπτώσεις, η τυπική εξέλιξη ενός προϊόντος είναι πολύ μικρή. Από την στιγμή που θα κυκλοφορήσει μια έκδοση του προϊόντος, ξεκινάει η δουλειά για την επόμενη έκδοση η οποία, για λόγους μάρκετινγκ, είναι πιθανό να παρουσιαστεί ως νέο (αλλά συμβατό) προϊόν και όχι ως τροποποιημένη έκδοση ενός προϊόντος που ο χρήστης έχει ήδη αγοράσει. Επομένως, αντίθετα με ότι συμβαίνει στην περίπτωση του λογισμικού κατά παραγγελία, το κόστος εξέλιξης δεν εκτιμάται ξεχωριστά, αλλά είναι απλώς το κόστος ανάπτυξης της επόμενης έκδοσης του συστήματος.

Κατανομή κόστους ανάπτυξης προϊόντος

0 25 50 75 100

Εξαγωγή προδιαγραφών	Ανάπτυξη	Δοκιμές συστήματος

ΠΙΝΑΚΑΣ 1.3 *Κατανομή κόστους ανάπτυξης προϊόντος*

1.4.3 Τι είναι μέθοδοι της τεχνολογίας λογισμικού;

Μέθοδος της τεχνολογίας λογισμικού είναι μια δομημένη προσέγγιση στην ανάπτυξη λογισμικού η οποία στοχεύει στη διευκόλυνση της παραγωγής λογισμικού υψηλής ποιότητας με τρόπο οικονομικά αποδοτικό. Μέθοδος όπως η δομημένη ανάλυση (DeMarco[10], 1978) και η JSD (Jackson, 1983) αναπτύχθηκαν για πρώτη φορά τη δεκαετία του 1970. Αυτές οι μέθοδοι επιχειρούσαν να προσδιορίσουν τα βασικά λειτουργικά συστατικά στοιχεία ενός συστήματος. Τέτοιες μέθοδοι προσανατολισμένες στις συναρτήσεις χρησιμοποιούνται ακόμα. Στις δεκαετίες του 1980 και του 1990 αυτές οι συναρτησιοστρεφείς (function-oriented) μέθοδοι συμπληρώθηκαν με αντικειμενοστραφείς (object oriented) μεθόδους, όπως εκείνες που προτάθηκαν από τον Booch[6] (Booch, 1994) και τον Rumbaugh (Rumbaugh, κ.α., 1991[28]). Όλες οι παραπάνω προσεγγίσεις έχουν σήμερα ενσωματωθεί σε μια ενιαία προσέγγιση βασισμένη στη γλώσσα UML(Unified Modeling Language, Ενοποιημένη Γλώσσα Μοντελοποίησης)(Booch[7], κ.α., 1999, Rumbaugh[29], κ.α., 1999a, Rumbaugh, κ.α., 1999b[30]).

Συστατικό Στοιχείο	Περιγραφή	Παράδειγμα
Περιγραφές Μοντέλων Συστήματος	Περιγραφές των μοντέλων συστήματος που πρέπει να αναπτυχθούν και της σημειογραφίας που θα χρησιμοποιηθεί για τον ορισμό αυτών των μοντέλων.	Μοντέλα αντικειμένων, μοντέλα ροής δεδομένων, μοντέλα μηχανών καταστάσεων κ.α.
Κανόνες	Περιορισμοί που ισχύουν πάντοτε για τα μοντέλα του συστήματος.	Κάθε οντότητα ενός μοντέλου συστήματος πρέπει να έχει ένα μοναδικό όνομα.
Συστάσεις	Ευρετικοί κανόνες που χαρακτηρίζουν την καλή πρακτική σχεδιασμού στη συγκεκριμένη μέθοδο. Η υιοθέτηση αυτών των συστάσεων θα πρέπει να οδηγεί σε καλά οργανωμένα μοντέλα συστήματος.	Κανένα αντικείμενο δεν πρέπει να συνδέεται με περισσότερα από επτά άλλα υποαντικείμενα.
Καθοδήγηση για την Διαδικασία	Περιγραφές και οργάνωση των δραστηριοτήτων που μπορεί κανείς να ακολουθήσει για την ανάπτυξη των μοντέλων συστήματος.	Οι ιδιότητες ενός αντικειμένου πρέπει να τεκμηριωθούν πριν τον ορισμό των λειτουργιών που συνδέονται με το αντικείμενο αυτό

ΠΙΝΑΚΑΣ 1.4 *Συστατικά στοιχεία μιας μεθόδου*

Ιδανική μέθοδος δεν υπάρχει, και κάθε μέθοδος έχει διαφορετικά πεδία εφαρμογής. Οι αντικειμενοστραφείς μέθοδοι, για παράδειγμα, είναι συχνά κατάλληλες για αλληλεπιδραστικά συστήματα αλλά όχι και για συστήματα με αυστηρές απαιτήσεις απόκρισης σε πραγματικό χρόνο.

Όλες οι μέθοδοι βασίζονται στην ιδέα της ανάπτυξης μοντέλων ενός συστήματος τα οποία να μπορούν να αναπαρασταθούν με γραφικό τρόπο και να αξιοποιηθούν ως προδιαγραφές ή σχεδιασμός του συστήματος. Κάθε μέθοδος περιλαμβάνει ένα σύνολο διαφορετικών συστατικών στοιχείων (Εικόνα 1.4).

1.4.4 Ποιες προκλήσεις αντιμετωπίζει η τεχνολογία λογισμικού;

Η τεχνολογία λογισμικού στο 21^{ο} αιώνα έρχεται αντιμέτωπη με τρεις βασικές προκλήσεις:

1. Πρόκληση της ετερογένειας. Υπάρχει όλο και μεγαλύτερη απαίτηση να λειτουργούν τα συστήματα λογισμικού ως κατανεμημένα συστήματα σε δίκτυα που περιλαμβάνουν διαφορετικούς τύπους υπολογιστών και διαφορετικού είδους υποδομές υποστήριξης. Συχνά επίσης είναι απαραίτητος ο συνδυασμός νέου λογισμικού με παλαιότερα κληρονομημένα συστήματα, γραμμένα σε διαφορετικές γλώσσες προγραμματισμού. Η πρόκληση της ετερογένειας συνίσταται στο να αναπτυχθούν τεχνικές παραγωγής φερέγγυου λογισμικού που να είναι ταυτόχρονα αρκετά ευέλικτο ώστε να αντιμετωπίζει αυτή την ετερογένεια.
2. Πρόκληση της γρήγορης παράδοσης. Πολλές παραδοσιακές τεχνικές της τεχνολογίας λογισμικού είναι χρονοβόρες. Ο χρόνος που απαιτούν είναι αναγκαίος προκειμένου να διασφαλιστεί η ποιότητα του λογισμικού. Όμως οι επιχειρήσεις σήμερα πρέπει να αποκρίνονται στις συνθήκες και να μεταβάλλονται πολύ γρήγορα, και το λογισμικό στο οποίο βασίζονται πρέπει να μεταβάλλεται εξίσου γρήγορα. Η πρόκληση του χρόνου παράδοσης συνιστάται στο να συντομευθεί ο χρόνος παράδοσης των μεγάλων και σύνθετων συστημάτων, χωρίς όμως να υποβαθμιστεί η ποιότητά τους.
3. Πρόκληση της εμπιστοσύνης. Καθώς το λογισμικό συνυφαίνεται με όλες τις πτυχές της ζωής μας, γίνεται απαραίτητο να μπορούμε να το εμπιστευόμαστε. Αυτό ισχύει ιδιαίτερα για τα συστήματα λογισμικού τηλεπρόσβασης μέσω μιας διασύνδεσης ιστοσελίδας ή υπηρεσίας Ιστού. Η πρόκληση της εμπιστοσύνης συνίσταται στο να αναπτυχθούν τεχνικές που να καταδεικνύουν ότι οι χρήστες μπορούν να εμπιστεύονται το λογισμικό.

Οι προκλήσεις αυτές δεν είναι βέβαια ανεξάρτητες μεταξύ τους. Θα μπορούσε, για παράδειγμα, να χρειαστεί να πραγματοποιηθούν γρήγορα αλλαγές σε ένα κληρονομημένο

σύστημα προκειμένου να προστεθεί μια διασύνδεση υπηρεσίας Ιστού. Για να αντιμετωπιστούν αυτές οι προκλήσεις, θα χρειαστούν νέα εργαλεία και τεχνικές, καθώς και καινοτόμοι τρόποι συνδυασμού και χρήσης των υφισταμένων μεθόδων της τεχνολογίας λογισμικού.

Χαρακτηριστικό Προϊόντος	Περιγραφή
Συντηρησιμότητα (Maintainability)	Το λογισμικό πρέπει να είναι γραμμένο έτσι ώστε να μπορεί να εξελίσσεται για να ανταποκρίνεται στις μεταβαλλόμενες ανάγκες των πελατών. Το γνώρισμα αυτό είναι κρίσιμο, καθώς η αλλαγή του λογισμικού είναι αναπόφευκτη συνέπεια ενός μεταβαλλόμενου επιχειρηματικού περιβάλλοντος.
Φερεγγυότητα (Dependability)	Η φερεγγυότητα του λογισμικού περιλαμβάνει ένα σύνολο χαρακτηριστικών: την αξιοπιστία (reliability), την προστασία από εξωτερικούς κινδύνους (security), και την ασφάλεια (safety). Το φερέγγυο λογισμικό δεν θα πρέπει να προκαλεί υλικές ή οικονομικές ζημιές σε περίπτωση αστοχίας του συστήματος.
Αποδοτικότητα (Efficiency)	Το λογισμικό δεν πρέπει να σπαταλάει πόρους του συστήματος, όπως η μνήμη και οι κύκλοι του επεξεργαστή. Η αποδοτικότητα, επομένως, περιλαμβάνει το βαθμό απόκρισης, το χρόνο επεξεργασίας, την αξιοποίηση της μνήμης κ.λπ.
Χρηστικότητα (Usability)	Το λογισμικό πρέπει να μπορεί να χρησιμοποιηθεί από τους τύπους χρηστών στους οποίους απευθύνεται

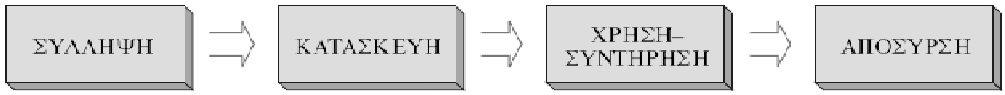
	χωρίς να απαιτεί αδικαιολόγητα μεγάλη προσπάθεια. Αυτό σημαίνει ότι πρέπει να διαθέτει μια κατάλληλη διασύνδεση χρήστη και επαρκή τεκμηρίωση.

ΠΙΝΑΚΑΣ 1.5 *Βασικά γνωρίσματα του καλού λογισμικού*

ΚΕΦΑΛΑΙΟ 2 : Μοντέλα Κύκλου Ζωής Λογισμικού

2.1 Η έννοια του μοντέλου κύκλου ζωής

Κάθε εφαρμογή λογισμικού, από τη σύλληψη μέχρι την απόσυρσή της, διέρχεται από διάφορες φάσεις, σε καθεμιά εκ των οποίων πρέπει να γίνονται ορισμένες εργασίες ώστε να επιτυγχάνεται το επιθυμητό αποτέλεσμα. Σε μακροσκοπικό επίπεδο οι πολύ γενικές φάσεις είναι: σύλληψη, κατασκευή, χρήση / συντήρηση και απόσυρση και όπως είναι εύκολα αντιληπτό, λαμβάνουν χώρα με τη σειρά αυτή. Μια εικόνα των γενικών αυτών φάσεων φαίνεται στο 2.1, που ακολουθεί.



ΣΧΗΜΑ 2.1 *Γενικές φάσεις του κύκλου ζωής του λογισμικού*

Πριν αναφερθούμε στον ορισμό του μοντέλου κύκλου ζωής και στα σημαντικότερα τέτοια μοντέλα που χρησιμοποιούνται σήμερα, είναι σκόπιμο να δοθούν ορισμένοι χρήσιμοι ορισμοί, οι οποίοι θα χρησιμοποιηθούν εκτεταμένα στη συνέχεια. Σε κάποιους από τους ορισμούς αυτούς ενδεχομένως ο αναγνώστης να συναντήσει και διαφορετικές
απόψεις στη βιβλιογραφία. Όπως έχουμε ήδη αναφέρει, ο πλουραλισμός ελάχιστα διαφοροποιούμενων απόψεων δεν είναι σπάνιος στην Τεχνολογία Λογισμικού, η δε αναφορά πολλών διαφορετικών απόψεων σε αυτό το σημείο, δεν εξυπηρετεί παρά τη σύγχυση.

Δραστηριότητα ανάπτυξης λογισμικού

*Μια **δραστηριότητα** ή **διαδικασία ανάπτυξης λογισμικού**[2] (software process) καθορίζει ποιες ενέργειες πρέπει να γίνουν για να επιτευχθεί ένα επιθυμητό αποτέλεσμα σε κάποια από τις φάσεις του κύκλου ζωής. Μια δραστηριότητα μπορεί να αναλύεται σε περισσότερες από μία επιμέρους **φάσεις**.*

Η έννοια «ανάπτυξη» στον προηγούμενο ορισμό περιγράφει μια γενική διαδικασία στην οποία υπόκειται το λογισμικό και όχι υποχρεωτικά κατασκευή εκ του μηδενός.

[2] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη, Τεχνολογία Λογισμικού Ι, σελ **31.**

Μεθοδολογία ανάπτυξης

*Μια **μεθοδολογία**[3] (software development methodology) καθορίζει το **πώς** θα πρέπει να εκτελούνται οι δραστηριότητες ανάπτυξης, δηλαδή ποιες επιμέρους ενέργειες περιλαμβάνουν, ποια βήματα γίνονται σε καθεμιά, ποια προϊόντα παράγονται, καθώς και πότε αυτές θεωρούνται περατωθείσες.*

Εργαλείο

*Ένα **εργαλείο λογισμικού**[4] (CASE: Computer.Aided Software Engineering) είναι ένα σύστημα (συνήθως είναι και το ίδιο εφαρμογή λογισμικού) το οποίο υποστηρίζει τη μερική ή (σπάνια)ολική αυτοματοποίηση των εργασιών που λαμβάνουν χώρα κατά την εφαρμογή των μεθοδολογιών ανάπτυξης λογισμικού.* Με βάση τα προηγούμενα, μπορούμε να δώσουμε τον ορισμό του Μοντέλου Κύκλου Ζωής Λογισμικού.

2.2 Μοντέλο Κύκλου Ζωής Λογισμικού

Ένα Μοντέλο Κύκλου Ζωής Λογισμικού[5] είναι μια περιγραφή των δραστηριοτήτων και των επιμέρους φάσεων από τις οποίες διέρχεται μια εφαρμογή λογισμικού από τη σύλληψη μέχρι την απόσυρσή της, καθώς και των εργασιών που λαμβάνουν χώρα σε καθεμιά από τις φάσεις αυτές.

Στο Σχήμα 2.2 φαίνεται η σχέση μεταξύ των εννοιών «μοντέλο κύκλου ζωής», «διαδικασία ανάπτυξης», «μεθοδολογία», καθώς και «εργαλείο», οι οποίες ορίστηκαν προηγουμένως. Μια έννοια που βρίσκεται χαμηλότερα στην πυρίτιδα αποτελεί το υπόβαθρο πάνω στο οποίο βασίζεται η έννοια που βρίσκεται στο αλέσω ψηλότερο Σητεία Κ.Ο.

[3] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **31.**

[4] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **32.**

[5] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **32.**

ΣΧΗΜΑ 2.2 *Σχέσεις εννοιών στην ανάπτυξη του λογισμικού*

Τα μοντέλα κύκλου ζωής λογισμικού προσδιορίζουν τις διαδικασίες ανάπτυξης οι οποίες λαμβάνουν χώρα κατά τις γενικές φάσεις «κατασκευή» και «χρήση - συντήρηση» (Σχήμα 2.1), προσδιορίζοντας τις επιμέρους φάσεις στις οποίες αυτές αναλύονται, τα προϊόντα που παράγονται σε καθεμία από αυτές, καθώς και τη σειρά εκτέλεσής τους.
Σε κάθε *διαδικασία ανάπτυξης* μπορούμε να διακρίνουμε περισσότερες από μία *επιμέρους φάσεις*, ενώ σε κάθε επιμέρους φάση μπορούμε να διακρίνουμε περισσότερες από μία *εργασίες*. Οι διαδικασίες ανάπτυξης λογισμικού μπορούν να ταξινομηθούν ως ακολούθως:

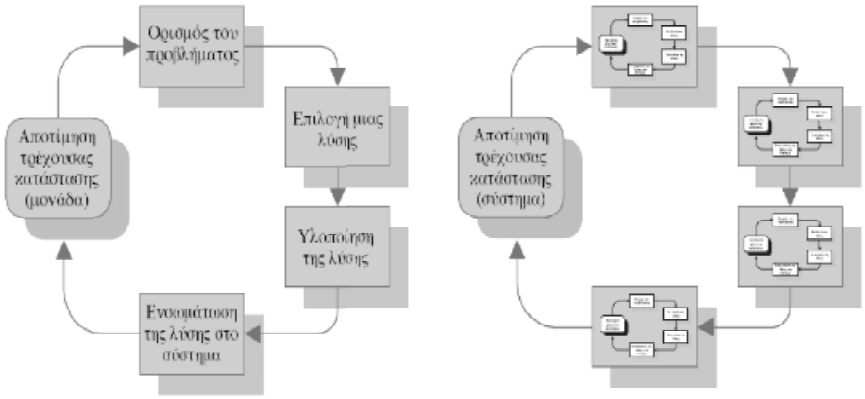
- **Προδιαγραφή**, δηλαδή καθορισμός των εργασιών που θα επιτελεί το λογισμικό, καθώς και των περιορισμών και των παραδοχών που ισχύουν.
- **Ανάπτυξη**, δηλαδή κατασκευή του λογισμικού. Εδώ, σε όλα τα μοντέλα κύκλου ζωής μπορούμε να διακρίνουμε τρεις επιμέρους φάσεις: την **ανάλυση**, τη **σχεδίαση** και τη **συγγραφή του πηγαίου κώδικα** (source code), την οποία στη συνέχεια θα ονομάζουμε και *κωδικοποίηση*.
- **Επαλήθευση**, δηλαδή επιβεβαίωση της ικανοποίησης των προδιαγραφών και της μη ύπαρξης σφαλμάτων.
- **Εξέλιξη**, δηλαδή επαύξηση των λειτουργικών χαρακτηριστικών του λογισμικού ή τροποποίηση υπαρχουσών, προκειμένου να ικανοποιούνται οι μεταβαλλόμενες ανάγκες.

Ένα μοντέλο κύκλου ζωής λογισμικού στοχεύει στην καθοδήγηση του κατασκευαστή προκειμένου αυτός να επιτύχει την καλύτερη δυνατή υλοποίηση των διαδικασιών ανάπτυξης λογισμικού. Λέγοντας «καλύτερη δυνατή», εννοούμε περισσότερο παραγωγική, με τα λιγότερα δυνατά σφάλματα και το μικρότερο δυνατό ρίσκο στις εκάστοτε συνθήκες. Τα παραπάνω μπορούν να διαφοροποιούνται ανάλογα με το μέγεθος και το θεματικό πεδίο κάθε εφαρμογής λογισμικού, με την εμπειρία και τα ιδιαίτερα χαρακτηριστικά του κάθε κατασκευαστή και ασφαλώς, με το εκάστοτε περιβάλλον ανάπτυξης.

Μια σημαντική παράμετρος που καταδεικνύει τη σημασία των μοντέλων κύκλου ζωής είναι το κόστος, ιδωμένο με την ευρύτερη σημασία του. Το κόστος αναθεώρησης αποφάσεων ή/και διόρθωσης σφαλμάτων είναι τόσο μεγαλύτερο, όσο μεγαλύτερη είναι και η απαιτούμενη οπισθοδρόμηση της διαδικασίας που αυτή συνεπάγεται. Το κόστος αυτό δεν αφορά μόνο οικονομικούς πόρους που αποδίδονται στο έργο, αλλά και χρόνο καθυστέρησης,

που δεν είναι πάντα διαθέσιμος σε πραγματικές συνθήκες. Επίσης, είναι συχνό φαινόμενο οι παρενέργειες στο υπόλοιπο σύστημα λογισμικού (side.effects), οι οποίες μπορούν να μεταβάλλουν προς το χειρότερο τα ποιοτικά του χαρακτηριστικά και δεν είναι εύκολο να εντοπιστούν από την αρχή.

Υπάρχουν αρκετά μοντέλα κύκλου ζωής, τα οποία διαφοροποιούνται ως προς τη σύλληψη της ιδέας του τρόπου κατασκευής, αλλά και ως προς τις επιμέρους φάσεις που προτείνουν, την επαναληπτικότητα και την εμβέλεια των εργασιών αυτών, τα ενδιάμεσα προϊόντα. συστατικά λογισμικού και την περιγραφή τους, τις οικονομικές και επιχειρηματικές πλευρές της χρήσης τους κ.ά. Καθεμία από τις ενέργειες που περιγράφεται σε ένα μοντέλο κύκλου ζωής είναι μια *διαδικασία επίλυσης προβλημάτων* (problem solving process). Τα βήματα κάθε τέτοιας διαδικασίας φαίνονται στο Σχήμα 2.3. Ο μηχανικός λογισμικού εκτελεί συνεχώς τέτοιες διαδικασίες επίλυσης προβλημάτων, τόσο σε μικροσκοπικό επίπεδο, δηλαδή στις μονάδες τού υπό κατασκευή λογισμικού (αριστερό τμήμα του σχήματος), όσο και σε μακροσκοπικό επίπεδο, δηλαδή για ολόκληρο το σύστημα (δεξί τμήμα του σχήματος).



ΣΧΗΜΑ 2.3 *Επίλυση προβλημάτων στο μικροσκοπικό (αριστερά) και μακροσκοπικό επίπεδο (δεξιά) ενός συστήματος λογισμικού*

Ακολούθως γίνεται μια σύντομη αναφορά στα σημαντικότερα μοντέλα κύκλου ζωής λογισμικού που έχουν εμφανιστεί, με σκοπό την παρουσίαση της κεντρικής ιδέας και των ποιοτικών χαρακτηριστικών αυτών και όχι την αναλυτική παρουσίασή τους, για την οποία ο αναγνώστης παραπέμπεται στη βιβλιογραφία. Στα μοντέλα κύκλου ζωής που θα παρουσιαστούν εμφανίζονται επιμέρους φάσεις και εργασίες αντίστοιχες και με παρόμοια σειρά, μόνο που παριστάνονται με διαφορετικό σχηματικό τρόπο. Αυτό δεν σημαίνει ότι επαναλαμβάνονται τα ίδια πράγματα και ότι μόνη διαφορά είναι η σχηματική απεικόνιση.

Ούτως ή άλλως, οι γενικές εργασίες κατά την ανάπτυξη του λογισμικού είναι οι τέσσερις που προαναφέρθηκαν, δηλαδή προδιαγραφή ανάπτυξη, επαλήθευση, εξέλιξη.

Το πρώτο και γηραιότερο μοντέλο κύκλου ζωής λογισμικού, αυτό του καταρράκτη, αντιμετωπίζει την ανάπτυξη του λογισμικού ως τη μεταφορά ενός μεγάλου ογκόλιθου από ένα Σητεία σε κάποιο άλλο περνώντας από ενδιάμεσες στάσεις, αλλά μεταφέροντας από τη μία στάση στην άλλη ολόκληρο τον ογκόλιθο. Επειδή οι εφαρμογές λογισμικού είναι περισσότερο εύπλαστες και ευμετάβλητες από τους ογκόλιθους, σύντομα παρουσιάστηκαν προβλήματα στην ιδέα, οπότε εμφανίστηκαν και άλλα μοντέλα κύκλου ζωής, τα οποία μεταφέρουν με διαφορετικούς και πιο ευέλικτους τρόπους μικρότερα τμήματα του «ογκόλιθου».

Αυτό λοιπόν, που διαφοροποιεί τα διάφορα μοντέλα κύκλου ζωής λογισμικού είναι η εμβέλεια, δηλαδή η έκταση του υπό κατασκευή συστήματος λογισμικού στην οποία αυτές οι διαδικασίες εφαρμόζονται, η επαναληπτικότητα των εργασιών, καθώς και οι ενδιάμεσες αποτιμήσεις από τον πελάτη ή τον κατασκευαστή. Έτσι, σε κάθε μοντέλο κύκλου ζωής είναι με διαφορετικό τρόπο δυνατός ο εντοπισμός της ανάγκης και η ενσωμάτωση τροποποιήσεων στα χαρακτηριστικά του λογισμικού προτού να ολοκληρωθεί πλήρως η κατασκευή του, οπότε μπορεί να είναι αργά από πλευράς χρόνου και κόστους.

2.2.1 Μοντέλα διαδικασιών παραγωγής λογισμικού

Όπως είδαμε στο Κεφάλαιο 1, τα μοντέλα διαδικασίας παραγωγής λογισμικού είναι αφηρημένες αναπαραστάσεις κάποιας διαδικασίας παραγωγής λογισμικού. Κάθε μοντέλο αναπαριστά μια διαδικασία από κάποια συγκεκριμένη σκοπιά, και επομένως παρέχει πληροφορίες για ορισμένες μόνο πλευρές της διαδικασίας. Στην ενότητα αυτή, θα συναντήσουμε μια σειρά πολύ γενικών μοντέλων διαδικασιών, που μερικές φορές λέγονται υποδείγματα διαδικασιών (process paradigms), και θα τα παρουσιάσουμε από μια αρχιτεκτονική προοπτική. Θα αναφερθούμε δηλαδή στο πλαίσιο των διαδικασιών, και όχι στις λεπτομέρειες των συγκεκριμένων δραστηριοτήτων τους.

Αυτά τα γενικά μοντέλα δεν είναι ρητές περιγραφές διαδικασιών παραγωγής λογισμικού. Είναι αφηρημένες περιγραφές διαδικασιών οι οποίες μπορούν να χρησιμοποιηθούν για την εξήγηση διαφορετικών προσεγγίσεων της ανάπτυξης λογισμικού. Μπορούν να θεωρηθούν ως πλαίσια διαδικασιών, τα οποία είναι σε θέση να επεκταθούν και να προσαρμοστούν έτσι ώστε να προκύψουν πιο συγκεκριμένες διαδικασίες παραγωγής λογισμικού.

Τα μοντέλα διαδικασιών που θα καλύψουμε είναι τα παρακάτω:

1. Το μοντέλο καταρράκτη (waterfall model). Το μοντέλο αυτό παίρνει τις θεμελιώδεις δραστηριότητες της διαδικασίας, δηλαδή την εξαγωγή προδιαγραφών, την ανάπτυξη, την επικύρωση, και την εξέλιξη, και τις αναπαριστά ως ξεχωριστές φάσεις, όπως ο καθορισμός απαιτήσεων, ο σχεδιασμός λογισμικού, η υλοποίηση, οι δοκιμές κ.λπ.
2. Εξελικτική ανάπτυξη (evolutionary development). Η προσέγγιση αυτή αναμιγνύει τις δραστηριότητες της εξαγωγής προδιαγραφών, της ανάπτυξης, και της επικύρωσης. Καταρχήν, αναπτύσσεται γρήγορα ένα αρχικό σύστημα από αφηρημένες προδιαγραφές. Στη συνέχεια, το σύστημα αυτό βελτιώνεται με την ανάδραση των πελατών, ώστε να παραχθεί ένα σύστημα που να ικανοποιεί τις ανάγκες τους.
3. Τεχνολογία λογισμικού βάσει συστατικών στοιχείων (component-based software engineering): Η προσέγγιση αυτή στηρίζεται στη ύπαρξη ενός σημαντικού αριθμού επαναχρησιμοποιήσιμων συστατικών στοιχείων του λογισμικού. Η διαδικασία ανάπτυξης του συστήματος εστιάζεται περισσότερο στο συνδυασμό τέτοιων στοιχείων ώστε να αποτελέσουν ένα σύστημα, και όχι στην ανάπτυξή τους από μηδενική βάση.

Αυτά τα τρία μοντέλα διαδικασιών χρησιμοποιούνται σε ένα ευρύ φάσμα της τρέχουσας πρακτικής της τεχνολογίας λογισμικού. Δεν αποκλείουν το ένα το άλλο, και συχνά χρησιμοποιούνται μαζί, ιδιαίτερα σε έργα ανάπτυξης μεγάλων συστημάτων. Έχουν προταθεί κάθε είδους παραλλαγές αυτών των γενικών διαδικασιών, και πολλές από αυτές μπορεί να χρησιμοποιούνται σε κάποιες εταιρείες. Η σπουδαιότερη ίσως από αυτές τις παραλλαγές είναι οι τυπικές μέθοδοι ανάπτυξης συστημάτων, που στηρίζονται στην δημιουργία ενός τυπικά ορισμένου μαθηματικού μοντέλου του συστήματος. Το μοντέλο αυτό στη συνέχεια μεταφράζεται σε εκτελέσιμο κώδικα, με τη χρήση μαθηματικών μετασχηματισμών οι οποίοι διατηρούν τη συνέπειά του.

Το πιο γνωστό παράδειγμα τυπικής διαδικασίας ανάπτυξης είναι η διαδικασία καθαρής αίθουσας (cleanroom), που αναπτύχθηκε αρχικά από την IBM (mills[23], κ.α., 1987, Selby[31], κ.α., 1987, Linger[22], 1994, Prowell[26], κ.α., 1999). Στη διαδικασία αυτή, κάθε επαύξηση του λογισμικού προδιαγράφεται κατά τυπικό τρόπο, και η προδιαγραφή αυτή μετασχηματίζεται σε υλοποίηση. Η ορθότητα του λογισμικού επιδεικνύεται με τη χρήση μιας τυπικής προσέγγισης. Η διαδικασία δεν περιλαμβάνει δοκιμές για ατέλειες, ενώ οι δοκιμές του συστήματος επικεντρώνονται στην εκτίμηση της αξιοπιστίας του.

Τόσο η προσέγγιση καθαρής αίθουσας όσο και μια άλλη τυπική προσέγγιση που βασίζεται στη μέθοδο Β (Wordsworth[24], 1996) είναι ιδιαίτερα κατάλληλες για την ανάπτυξη συστημάτων που έχουν αυστηρές απαιτήσεις ασφάλειας, αξιοπιστίας, ή προστασίας από εξωτερικούς κινδύνους. Οι τυπικές προσεγγίσεις διευκολύνουν την παραγωγή μιας μελέτης ασφάλειας ή προστασίας, η οποία επιδεικνύει στους πελάτες ή στις επιτροπές πιστοποίησης ότι το σύστημα ανταποκρίνεται πράγματι στις απαιτήσεις ασφάλειας ή προστασίας.

Έξω από αυτά τα εξειδικευμένα πεδία εφαρμογής, όμως, οι διαδικασίες που βασίζονται σε τυπικούς μετασχηματισμούς δεν έχουν ευρεία χρήση. Απαιτούν ειδικές γνώσεις και στην πραγματικότητα, για την πλειοψηφία των συστημάτων, δεν προσφέρουν σημαντικά πλεονεκτήματα κόστους ή ποιότητας έναντι των άλλων προσεγγίσεων της ανάπτυξης συστημάτων.

2.3 Το μοντέλο καταρράκτη

Το πρώτο μοντέλο διαδικασίας ανάπτυξης λογισμικού που δημοσιεύτηκε προήλθε από γενικότερες πρακτικές της τεχνολογίας συστημάτων (Royce[27], 1970), όπως φαίνεται στην Εικόνα 2.4. Λόγω της μετάπτωσης από μία φάση στην άλλη, το μοντέλο αυτό έγινε γνωστό ως μοντέλο καταρράκτη (waterfall model) ή ως κύκλος ζωής λογισμικού (software life cycle). Τα κύρια στάδια αυτού του μοντέλου αντιστοιχούν στους θεμελιώδεις δραστηριότητες ανάπτυξης:

1. Ανάλυση και ο καθορισμός απαιτήσεων. Οι υπηρεσίες, οι περιορισμοί, και οι στόχοι του συστήματος προσδιορίζονται μέσω συζήτησης με τους χρήστες του. Στη συνέχεια καθορίζονται λεπτομερώς και χρησιμεύουν ως προδιαγραφές του συστήματος.
2. Σχεδιασμός συστήματος και λογισμικού. Η διαδικασία του σχεδιασμού του συστήματος χωρίζει τις απαιτήσεις σε συστήματα υλικού και λογισμικού. Αυτό καταλήγει σε μια συνολική αρχιτεκτονική του συστήματος. Ο σχεδιασμός λογισμικού περιλαμβάνει τον προσδιορισμό και την περιγραφή των θεμελιωδών αφηρημένων τμημάτων του συστήματος, καθώς και των σχέσεων μεταξύ τους.
3. Υλοποίηση και δοκιμές υπομονάδων. Κατά το στάδιο αυτό, ο σχεδιασμός του λογισμικού υλοποιείται ως ένα σύνολο προγραμμάτων ή προγραμματιστικών

υπομονάδων. Οι δοκιμές υπομονάδων συνίστανται στην επαλήθευση ότι κάθε υπομονάδα ικανοποιεί τις προδιαγραφές της.

4. Ενοποίηση και δοκιμές συστήματος. Οι μεμονωμένες προγραμματιστικές υπομονάδες ή προγράμματα ενοποιούνται και δοκιμάζονται ως ενιαίο σύστημα, για να διασφαλιστεί ότι οι απαιτήσεις λογισμικού έχουν εκπληρωθεί. Μετά τη δοκιμή, το σύστημα λογισμικού παραδίδεται στον πελάτη.
5. Λειτουργία και συντήρηση. Κανονικά (αλλά όχι απαραίτητα) η φάση αυτή είναι η πιο μακρόχρονη φάση του κύκλου ζωής. Το σύστημα εγκαθίστανται και τίθεται σε χρήση. Η συντήρηση συνίσταται στη διόρθωση σφαλμάτων που δεν εντοπίστηκαν στα προηγούμενα στάδια του κύκλου ζωής, τη βελτίωση της υλοποίησης των μονάδων του συστήματος, όπως και την βελτίωση των υπηρεσιών του συστήματος καθώς παρουσιάζονται νέες απαιτήσεις.

Γενικά, το αποτέλεσμα της κάθε φάσης είναι ένα ή περισσότερα εγκεκριμένα έγγραφα. Η επόμενη φάση δεν θα πρέπει να ξεκινήσει μέχρι να τελειώσει η προηγούμενη. Στην πράξη, όμως, τα στάδια αυτά επικαλύπτονται και ανταλλάσουν πληροφορίες. Κατά το σχεδιασμό εντοπίζονται προβλήματα των απαιτήσεων, κατά τη συγγραφή κώδικα λογισμικού εντοπίζονται προβλήματα του σχεδιασμού κοκ. Η διαδικασία παραγωγής λογισμικού δεν ακολουθεί ένα απλό γραμμικό μοντέλο, αλλά εμπεριέχει μια ακολουθία κυκλικών επαναλήψεων των δραστηριοτήτων ανάπτυξης.

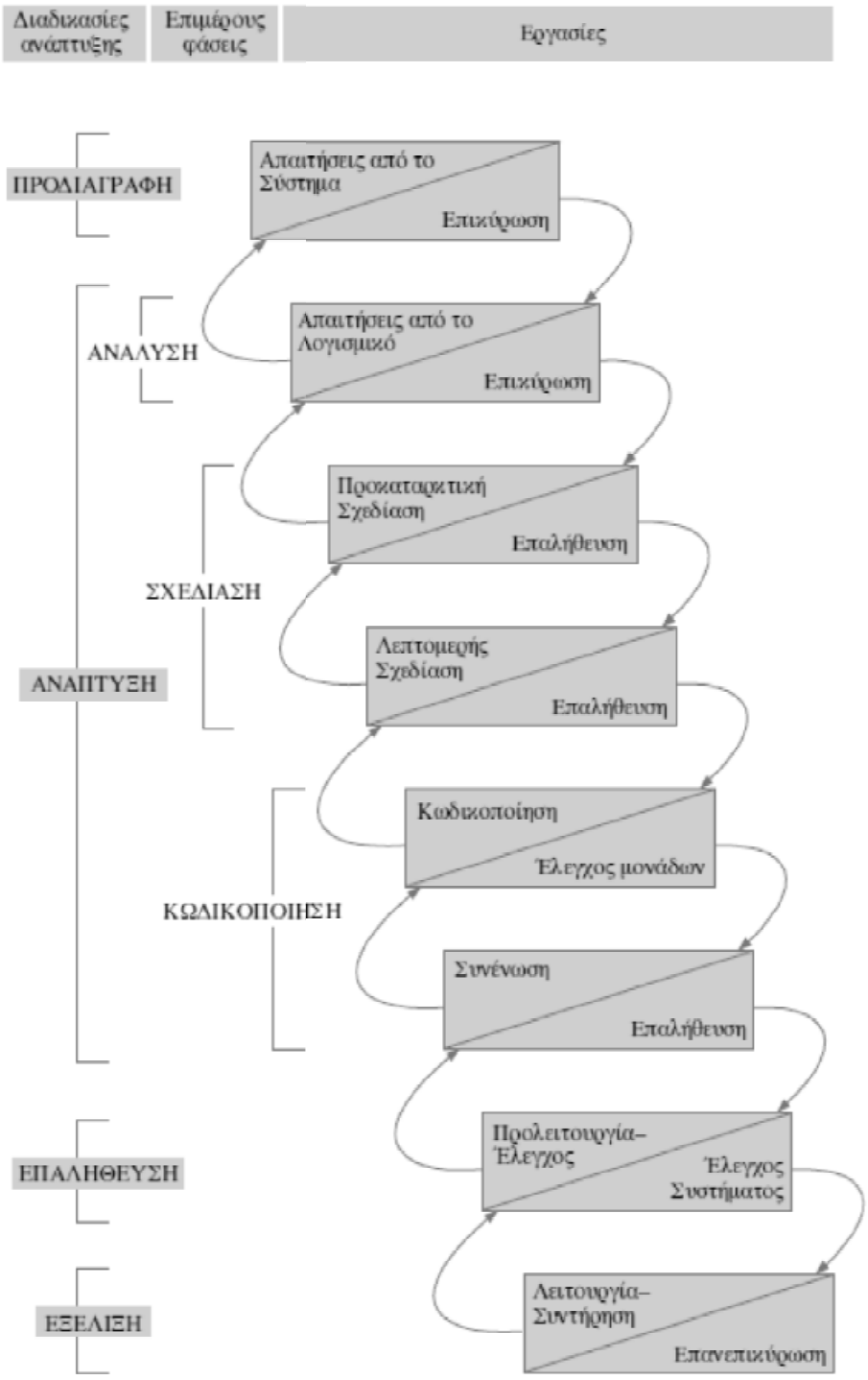
Εξαιτίας του κόστους παραγωγής και έγκρισης των εγγράφων, οι κυκλικές επαναλήψεις είναι δαπανηρές και εμπεριέχουν πολλή δουλειά. Γι’ αυτόν το λόγο, μετά από ένα μικρό αριθμό επαναληπτικών κύκλων, συνηθίζεται να «παγώνουν» ορισμένα τμήματα της ανάπτυξης, όπως η εξαγωγή προδιαγραφών, και η διαδικασία συνεχίζεται με επόμενα στάδια. Κάποια προβλήματα αφήνονται για να επιλυθούν αργότερα, αγνοούνται, ή παρακάμπτονται με τη βοήθεια κώδικα. Αυτό το πρόωρο «πάγωμα» των απαιτήσεων μπορεί να έχει συνέπεια το σύστημα να μην κάνει αυτά που θέλει ο χρήστης. Μπορεί επίσης να οδηγήσει σε συστήματα με κακή δομή, εφόσον προβλήματά του σχεδιασμού έχουν παρακαμφθεί με τεχνάσματα της υλοποίησης.

Στην τελική φάση του κύκλου ζωής (τη φάση της λειτουργίας και συντήρησης), το λογισμικό τίθεται σε χρήση. Διαπιστώνονται σφάλματα και παραλείψεις των αρχικών απαιτήσεων του λογισμικού. Προκύπτουν σφάλματα των προγραμμάτων και του σχεδιασμού, και διαπιστώνεται η ανάγκη για νέες λειτουργικές δυνατότητες. Γι’ αυτούς τους λόγους, το σύστημα πρέπει να εξελιχθεί για να παραμείνει χρήσιμο. Η πραγματοποίηση αυτών των

αλλαγών (η συντήρηση του λογισμικού) μπορεί να απαιτεί την επανάληψη προηγούμενων σταδίων της διαδικασίας.

Τα πλεονεκτήματα του μοντέλου καταρράκτη είναι ότι παράγει τεκμηρίωση σε κάθε φάση και ότι ταιριάζει με άλλα μοντέλα που χρησιμοποιούνται σε τεχνικά έργα. Το σημαντικότερο πρόβλημά του είναι ο άκαμπτος διαμερισμός του έργου σε ξεχωριστά στάδια. Απαιτεί να λαμβάνονται αποφάσεις σε πρώιμα στάδια της διαδικασίας, γεγονός που μπορεί να δυσχεραίνει την ανταπόκριση σε αλλαγές των απαιτήσεων του πελάτη.

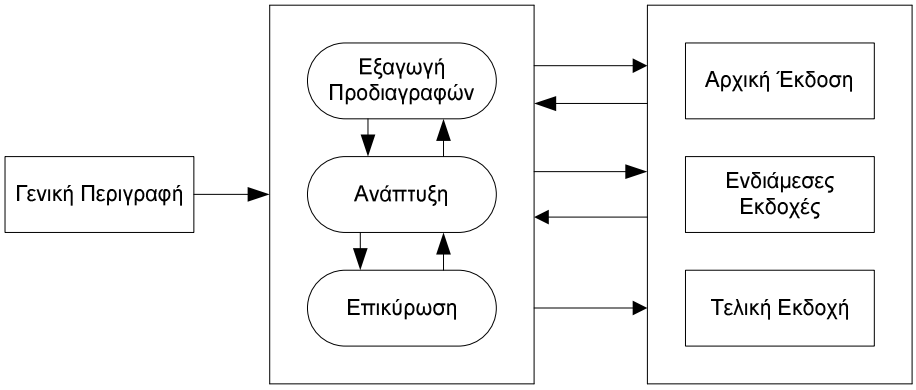
Επομένως, το μοντέλο καταρράκτη πρέπει να χρησιμοποιείται μόνο όταν οι απαιτήσεις είναι καλά κατανοητές και δεν είναι πιθανό να αλλάξουν ριζικά κατά την ανάπτυξη του συστήματος. Από την άλλη, όμως, το μοντέλο καταρράκτη αντανακλά το είδος μοντέλου διαδικασίας που χρησιμοποιείται σε άλλα τεχνικά έργα. Γι’ αυτόν το λόγο, διαδικασίες παραγωγής λογισμικού που βασίζονται σε αυτήν την προσέγγιση εξακολουθούν να χρησιμοποιούνται γενικότερα στην ανάπτυξη λογισμικού, ιδιαίτερα όταν το έργο λογισμικού αποτελεί μέρος ενός μεγαλύτερου έργου κατασκευής συστημάτων.



ΕΙΚΟΝΑ 2.4 *Το μοντέλο του καταρράκτη*

2.4 Εξελικτική ανάπτυξη

Η εξελικτική ανάπτυξη (evolutionary development) βασίζεται στην ανάπτυξη μιας αρχικής υλοποίησης, την έκθεσή της στους χρήστες για ανατροφοδότηση, και τη βελτίωσή της μέσω πολλών διαδοχικών εκδοχών, μέχρι να αναπτυχθεί ένα επαρκές σύστημα (Εικόνα 2.5). Οι δραστηριότητες της εξαγωγής προδιαγραφών, της ανάπτυξης και της επικύρωσης δεν είναι ξεχωριστές αλλά αλληλένδετες, με ταχύτατη ανταλλαγή πληροφοριών μεταξύ τους.



ΕΙΚΟΝΑ 2.5 *Εξελικτική ανάπτυξη*

Υπάρχουν δύο βασικοί τύποι εξελικτικής ανάπτυξης:

1. Διερευνητική ανάπτυξη (exploratory development), όπου η διαδικασία έχει στόχο τη συνεργασία με τον πελάτη προκειμένου να διερευνηθούν οι απαιτήσεις του και να του παραδοθεί ένα τελικό σύστημα. Η ανάπτυξη ξεκινάει από τα μέρη του συστήματος που είναι καλά κατανοητά. Το σύστημα εξελίσσεται με την προσθήκη νέων δυνατοτήτων που προτείνονται από τον πελάτη.
2. Δημιουργία αναλώσιμων πρωτοτύπων (throwaway prototyping), όπου η διαδικασία της εξελικτικής ανάπτυξης έχει στόχο την κατανόηση των απαιτήσεων του πελάτη και συνεπώς την ανάπτυξη ενός καλύτερου ορισμού απαιτήσεων του συστήματος. Το πρωτότυπο εστιάζεται στον πειραματισμό με τις απαιτήσεις του πελάτη που δεν είναι καλά κατανοητές.

Η εξελικτική προσέγγιση της ανάπτυξης λογισμικού είναι συχνά πιο αποτελεσματική από την προσέγγιση καταρράκτη για την παραγωγή συστημάτων που ικανοποιούν τις άμεσες ανάγκες των πελατών. Το πλεονέκτημα μιας διαδικασίας παραγωγής λογισμικού που βασίζεται στην εξελικτική προσέγγιση είναι ότι οι προδιαγραφές μπορούν να αναπτυχθούν

σταδιακά. Καθώς οι χρήστες αναπτύσσουν μια καλύτερη κατανόηση του προβλήματός τους, αυτό μπορεί να αντικατοπτριστεί στο σύστημα λογισμικού. Ωστόσο, από τεχνική και διοικητική άποψη, η εξελικτική προσέγγιση έχει δύο προβλήματα:

1. Η διαδικασία είναι αδιαφανής. Οι διευθυντές έργων χρειάζονται τακτικά παραδοτέα αποτελέσματα για να μετρήσουν την πρόοδο. Αν όμως τα συστήματα αναπτύσσονται με γρήγορο ρυθμό, δεν είναι αποδοτικό από άποψη κόστους να παράγεται τεκμηρίωση για κάθε εκδοχή τους.
2. Τα συστήματα συχνά έχουν κακή δομή. Οι συνεχείς αλλαγές τείνουν να καταστρέφουν τη δομή του λογισμικού. Γι' αυτόν το λόγο, η πραγματοποίηση αλλαγών στο λογισμικό γίνεται όλο και πιο δύσκολη και πιο δαπανηρή.

Για συστήματα μικρού ή μέσου μεγέθους (μέχρι 500.000 γραμμές κώδικα), η εξελικτική προσέγγιση είναι μάλλον η καλύτερη προσέγγιση ανάπτυξης. Τα προβλήματα της εξελικτικής ανάπτυξης λογισμικού γίνονται ιδιαίτερα έντονα στα μεγάλα, σύνθετα συστήματα με μεγάλη διάρκεια ζωής, όπου διαφορετικές ομάδες αναπτύσσουν διαφορετικά μέρη του συστήματος. Είναι δύσκολο να καθοριστεί μια σταθερή αρχιτεκτονική συστήματος με τη χρήση αυτής της προσέγγισης, πράγμα που δυσκολεύει την ενσωμάτωση των συνεισφορών των ομάδων στο σύνολο.

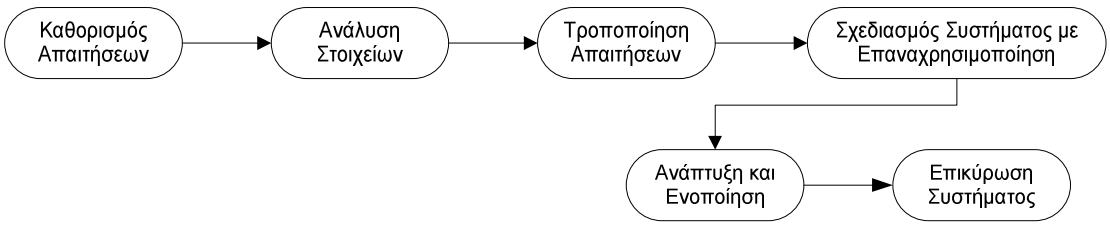
Για τα μεγάλα συστήματα, είναι προτιμότερη μια μικτή διαδικασία που να περιλαμβάνει τα καλύτερα χαρακτηριστικά των δύο μοντέλων, του καταρράκτη και της εξελικτικής ανάπτυξης. Αυτό μπορεί να σημαίνει την ανάπτυξη ενός αναλώσιμου πρωτοτύπου με τη χρήση της εξελικτικής προσέγγισης, προκειμένου να επιλυθούν κάποιες αβεβαιότητες των προδιαγραφών του συστήματος. Κατόπιν μπορείτε να επαναλάβετε την υλοποίηση του συστήματος χρησιμοποιώντας μια δομημένη προσέγγιση. Τα μέρη του συστήματος που είναι καλά κατανοητά μπορούν να προδιαγράφονται και να αναπτύσσονται με μια διαδικασία βασισμένη σε μοντέλο καταρράκτη. Άλλα μέρη του συστήματος, όπως είναι η διασύνδεση χρήστη, τα οποία είναι δύσκολο να προδιαγραφούν από την αρχή, θα πρέπει πάντα να αναπτύσσονται με τη χρήση μιας προσέγγισης διερευνητικού προγραμματισμού.

2.4.1 Τεχνολογία λογισμικού βάσει συστατικών στοιχείων

Στην πλειοψηφία των έργων λογισμικού, υπάρχει πάντοτε κάποιος βαθμός επαναχρησιμοποίησης λογισμικού. Αυτό συνήθως γίνεται άτυπα, όταν οι άνθρωποι που δουλεύουν στο έργο έχουν υπόψη τους σχεδιασμούς ή κώδικα που είναι παρεμφερή με τα ζητούμενα. Βρίσκουν αυτά τα στοιχεία, τα τροποποιούν κατάλληλα, και τα ενσωματώνουν στο σύστημά τους. Στην εξελικτική προσέγγιση, την οποία συζητήσαμε παραπάνω, η επαναχρησιμοποίηση είναι συχνά απαραίτητη για τη γρήγορη ανάπτυξη ενός συστήματος.

Αυτή η άτυπη επαναχρησιμοποίηση γίνεται ανεξάρτητα από τη διαδικασία ανάπτυξης που χρησιμοποιείται. Τα τελευταία χρόνια, όμως, έχει εμφανιστεί και χρησιμοποιείται όλο και περισσότερο μια προσέγγιση της ανάπτυξης λογισμικού που βασίζεται στην επαναχρησιμοποίηση, η οποία ονομάζεται τεχνολογία λογισμικού βάσει συστατικών στοιχείων (component-based software engineering, CBSE).

Αυτή η προσανατολισμένη στην επαναχρησιμοποίηση προσέγγιση στηρίζεται σε μια μεγάλη βάση επαναχρησιμοποιήσιμων στοιχείων λογισμικού και σε ορισμένα πλαίσια εργασίας για το συνδυασμό αυτών των στοιχείων. Μερικές φορές, τα συστατικά στοιχεία είναι και αυτά πλήρη συστήματα (έτοιμα συστήματα του εμπορίου, ή COTS-commercial off-the-shelf-systems), τα οποία μπορεί να παρέχουν κάποιες συγκεκριμένες λειτουργικές δυνατότητες, όπως μορφοποίηση κειμένου ή αριθμητικούς υπολογισμούς. Το γενικό μοντέλο διαδικασίας της τεχνολογίας λογισμικού βάσει συστατικών στοιχείων παρουσιάζεται στην Εικόνα 2.6.



ΕΙΚΟΝΑ 2.6 *Κατασκευή λογισμικού βάσει συστατικών στοιχείων*

Αν και τα στάδια του αρχικού καθορισμού απαιτήσεων και της επικύρωσης είναι ανάλογα με εκείνα άλλων διαδικασιών, τα ενδιάμεσα στάδια μιας διαδικασίας προσανατολισμένης στην επαναχρησιμοποίηση είναι διαφορετικά. Τα στάδια αυτά είναι τα εξής:

1. Ανάλυση συστατικών στοιχείων. Με δεδομένο ένα σύνολο προδιαγραφών απαιτήσεων, αναζητούνται στοιχεία που τις υλοποιούν. Συνήθως δεν υπάρχει

ακριβής ταύτιση με τις απαιτήσεις, και τα στοιχεία που μπορούν να χρησιμοποιηθούν καλύπτουν μόνο ένα μέρος των ζητούμενων λειτουργικών δυνατοτήτων.

2. Τροποποίηση απαιτήσεων. Κατά το στάδιο αυτό, οι απαιτήσεις αναλύονται λαμβάνοντας υπόψη τις πληροφορίες για τα συστατικά στοιχεία που εντοπίστηκαν. Στη συνέχεια, οι απαιτήσεις τροποποιούνται έτσι ώστε να αντανακλούν τα διαθέσιμα στοιχεία. Όπου η τροποποίηση είναι αδύνατη, μπορεί να επαναληφθεί η ανάλυση των συστατικών στοιχείων ώστε να αναζητηθούν εναλλακτικές λύσεις.
3. Σχεδιασμός συστήματος με επαναχρησιμοποίηση. Κατά τη φάση αυτή, σχεδιάζεται το πλαίσιο εργασίας του συστήματος ή επαναχρησιμοποιείται ένα υπάρχον πλαίσιο εργασίας. Οι σχεδιαστές λαμβάνουν υπόψη τα επαναχρησιμοποιούμενα στοιχεία και οργανώνουν το πλαίσιο εργασίας έτσι ώστε να τα εξυπηρετεί. Αν δεν υπάρχουν διαθέσιμα επαναχρησιμοποιήσιμα στοιχεία, μπορεί να απαιτηθεί ο σχεδιασμός κάποιου νέου λογισμικού.
4. Ανάπτυξη και ενοποίηση. Αναπτύσσεται το λογισμικό που δεν μπορεί να αποκτηθεί μέσω αγοράς, και κατόπιν συστατικά στοιχεία και έτοιμα συστήματα ενοποιούνται προκειμένου να δημιουργηθεί το νέο σύστημα. Στο μοντέλο αυτό, η ενοποίηση του συστήματος μπορεί να είναι μέρος της διαδικασίας ανάπτυξης, και όχι ξεχωριστή δραστηριότητα.

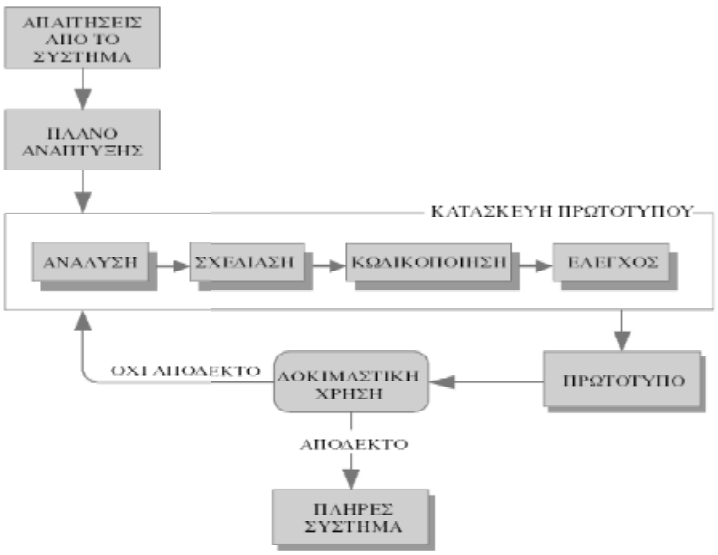
Η κατασκευή λογισμικού βάσει συστατικών στοιχείων έχει το προφανές πλεονέκτημα ότι μειώνει την ποσότητα λογισμικού που χρειάζεται να αναπτυχθεί, και επομένως μειώνει το κόστος και τους κινδύνους. Συνήθως οδηγεί επίσης σε ταχύτερη παράδοση του λογισμικού. Ωστόσο, οι συμβιβασμοί στις απαιτήσεις είναι αναπόφευκτοι, και αυτό μπορεί να οδηγήσει σε ένα σύστημα που δεν ικανοποιεί τις πραγματικές ανάγκες των χρηστών. Ακόμα, μέχρι κάποιο βαθμό χάνεται ο έλεγχος της εξέλιξης του συστήματος, καθώς Κι οι νέες εκδόσεις των επαναχρησιμοποιήσιμων στοιχείων δεν υπόκεινται στον έλεγχο του οργανισμού που τα χρησιμοποιεί.

Η τεχνολογία λογισμικού βάσει συστατικών στοιχείων έχει πολλά κοινά με μια νέα αναδυόμενη προσέγγιση της ανάπτυξης συστημάτων, η οποία βασίζεται στην ενοποίηση υπηρεσιών Ιστού από διάφορους προμηθευτές.

2.5 Το μοντέλο πρωτυποποίησης

Η κεντρική ιδέα του *μοντέλου πρωτοτυποποίησης* (prototyping model) είναι η ανάπτυξη του λογισμικού όχι εξολοκλήρου, αλλά σε τμήματα, που ονομάζονται «πρωτότυπα». Οι διαδικασίες ανάπτυξης επαναλαμβάνονται για ένα τμήμα του συστήματος κάθε φορά και, για το λόγο αυτό, το μοντέλο χαρακτηρίζεται ως *επαναληπτικό*. Κάθε πρωτότυπο περιλαμβάνει τις βασικές από τις λειτουργίες που προορίζεται να εκτελεί το λογισμικό και τίθεται σε δοκιμασία από τον πελάτη. Από εκεί συλλέγονται παρατηρήσεις και η διαδικασία κατασκευής νέου πρωτοτύπου επαναλαμβάνεται μέχρις ότου ένα πρωτότυπο να ικανοποιεί τις απαιτήσεις, δηλαδή να εκτελεί τις επιθυμητές λειτουργίες του λογισμικού με τρόπο ικανοποιητικό και να γίνεται αποδεκτό από τον πελάτη (Σχήμα 2.7). Από το Σητεία αυτό και μετά μπορούν να προστεθούν και οι υπόλοιπες λειτουργίες, ώστε το λογισμικό να ολοκληρωθεί.

Ένα σημαντικό πλεονέκτημα του μοντέλου αυτού είναι η δυνατότητα απόκτησης άποψης για την εφαρμογή λογισμικού νωρίτερα απ'ό,τι στο μοντέλο του καταρράκτη. Αυτό μπορεί να γλιτώσει την ανάπτυξη από καθυστερήσεις (και συνεπαγόμενα κόστη) ή ακόμη και από ολική αποτυχία, τα οποία θα επέρχονταν, αν ο κατασκευαστής αναγκαζόταν να οπισθοδρομήσει την ανάπτυξη, ενώ αυτή είχε προχωρήσει πολύ. Παράλληλα, ιδιαίτερη σημασία αποκτά η διοίκηση του έργου, η οποία πρέπει να εξασφαλίζει την υλοποιησιμότητα του πρωτοτύπου και την εύκολη τροποποίησή του. Κάθε κατασκευή πρωτοτύπου μπορεί να θεωρηθεί ως ένα μικρό έργο λογισμικού το οποίο κατασκευάζεται με διαδικασίες που μπορούν να ακολουθούν άλλα μοντέλα κύκλου ζωής, όπως αυτό του καταρράκτη.

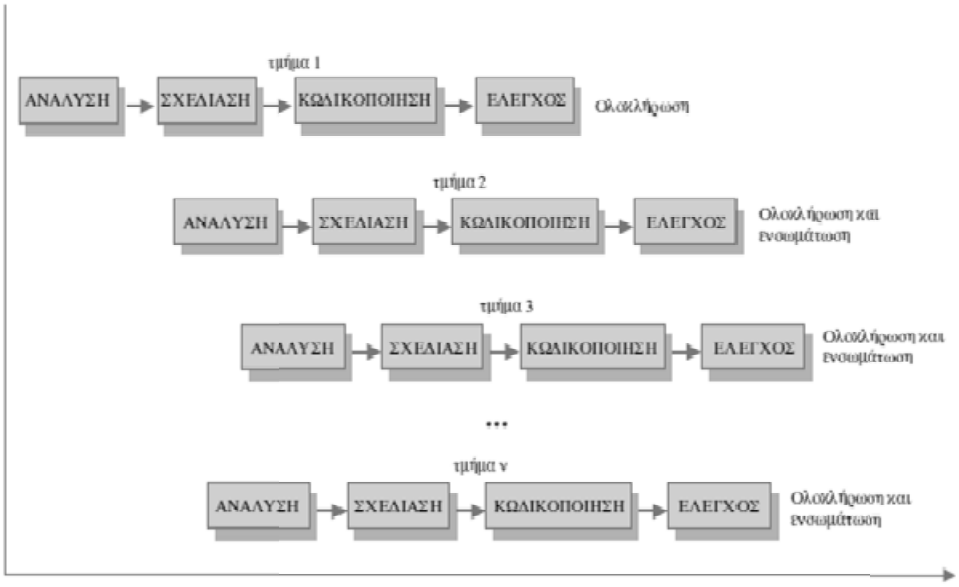


ΣΧΗΜΑ 2.7 *Το μοντέλο της πρωτοτυποποίησης*

Με βάση τις παραπάνω παρατηρήσεις, το μοντέλο πρωτοτυποποίησης χρησιμοποιείται στην ανάπτυξη εφαρμογών λογισμικού για τις απαιτήσεις από τις οποίες δεν υπάρχει βεβαιότητα στην αρχή της ανάπτυξης, οπότε δεν μπορούν να συμφωνηθούν και να παγιοποιηθούν. Τέτοιες είναι εφαρμογές που κατασκευάζονται για πρώτη φορά ή που είναι στενά εξαρτημένες από τον πελάτη, χωρίς να υπάρχει αποδεκτό προηγούμενο παράδειγμα. Ωστόσο, το μέγεθος των εφαρμογών αυτών δεν μπορεί να είναι ιδιαίτερα μεγάλο, διότι ο χρόνος ανάπτυξης κάθε πρωτοτύπου μεγαλώνει και η απαιτούμενη ευελιξία μειώνεται.

2.6 Το μοντέλο λειτουργικής επαύξησης

Το *μοντέλο της λειτουργικής επαύξησης* (incremental model) συνδυάζει την ακολουθιακή ανάπτυξη του μοντέλου του καταρράκτη με την τμηματική ανάπτυξη του μοντέλου της πρωτοτυποποίησης. Κεντρική ιδέα είναι η κατάτμηση του υπό κατασκευή λογισμικού σε τμήματα που αναπτύσσονται ανεξάρτητα, ακολουθώντας το καθένα ακολουθιακή ανάπτυξη σύμφωνα με το μοντέλο του καταρράκτη, όπως φαίνεται στο Σχήμα 2.8. Κατά την αρχική φάση ανάλυσης και σχεδίασης αποφασίζονται τα τμήματα στα οποία θα κατατμηθεί η εφαρμογή, η ανάπτυξη των οποίων γίνεται στη συνέχεια ανεξάρτητα και παράλληλα. Όταν ολοκληρώνεται η ανάπτυξη κάθε τμήματος, αυτό ενσωματώνεται στο σύνολο της εφαρμογής, διαδικασία η οποία δικαιολογεί και τον τίτλο «λειτουργική επαύξηση».



ΣΧΗΜΑ 2.6 *Το μοντέλο της λειτουργικής επαύξησης*

Πλεονεκτήματα της ιδέας είναι η δυνατότητα παράλληλης ανάπτυξης, η οποία τελικά καταλαμβάνει μικρότερο χρόνο, καθώς και ο διαδοχικός εμπλουτισμός των λειτουργικών χαρακτηριστικών του λογισμικού. Τα βασικά μειονεκτήματα του μοντέλου είναι τα ακόλουθα:

- Η αρχική κατάτμηση και γενική σχεδίαση του συστήματος αποκτά ιδιαίτερη βαρύτητα. Σφάλματα σε αυτή μπορούν να έχουν σημαντικές επιπτώσεις στο λογισμικό που θα κατασκευαστεί στη συνέχεια.
- Σε περίπτωση μεταβολής των λειτουργικών απαιτήσεων κατά τη χρήση του ημιτελούς συστήματος, μπορεί η αρχιτεκτονική αυτού να μεταβληθεί σε βαθμό που να κλονιστεί η ανάπτυξη των υπόλοιπων τμημάτων αυτού.

Το μοντέλο της λειτουργικής επαύξησης χρησιμοποιείται στην ανάπτυξη μεγάλων εφαρμογών λογισμικού για τις οποίες ισχύουν οι απαιτήσεις του μοντέλου του καταρράκτη, δηλαδή σαφής γνώση και μικρή ή καθόλου μεταβλητότητα των απαιτήσεων κατά την ανάπτυξη.

2.7 Επαναληπτικές διαδικασίες

Η αλλαγή είναι αναπόφευκτη σε όλα τα μεγάλα έργα λογισμικού. Οι απαιτήσεις του συστήματος μεταβάλλονται καθώς η εταιρεία-αγοραστής του συστήματος αποκρίνεται σε εξωτερικές πιέσεις. Οι διοικητικές προτεραιότητες αλλάζουν. Καθώς γίνονται διαθέσιμες νέες τεχνολογίες, οι σχεδιασμοί και η υλοποίηση αλλάζουν επίσης. Αυτό σημαίνει ότι η διαδικασία παραγωγής λογισμικού δεν είναι μια «εφάπαξ» διαδικασία-αντίθετα, οι δραστηριότητές της επαναλαμβάνονται τακτικά καθώς το σύστημα αναθεωρείται έτσι ώστε να ανταποκρίνεται σε ζητούμενες αλλαγές.

Η επαναληπτική ανάπτυξη (iterative development) είναι κάτι τόσο θεμελιώδες για το λογισμικό. Σε αυτή την ενότητα, θα κάνουμε μια παρουσίαση του θέματος, περιγράφοντας δύο μοντέλα διαδικασιών που είναι σχεδιασμένα ειδικά για την υποστήριξη μιας επαναληπτικής διαδικασίας:

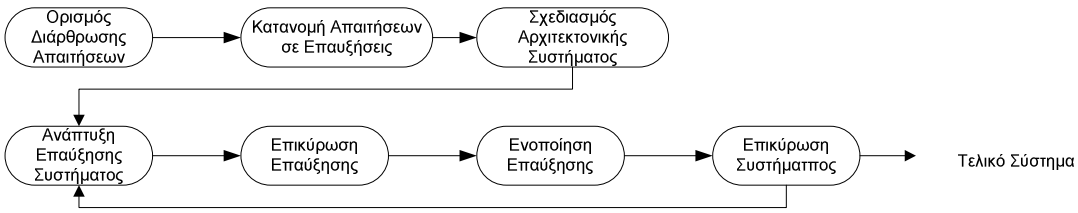
1. Βαθμιαία παράδοση (incremental delivery). Η εξαγωγή προδιαγραφών, ο σχεδιασμός, και η υλοποίηση του λογισμικού χωρίζονται σε μια σειρά από διαδοχικά βήματα (επαυξήσεις), κάθε ένα από τα οποία αναπτύσσεται με τη σειρά του.
2. Σπειροειδής ανάπτυξη (spiral development). Η ανάπτυξη του συστήματος εξελίσσεται ελικοειδώς προς τα έξω, ξεκινώντας από μια αρχική διάρθρωση και φτάνοντας στο τελικό σύστημα.

Η ουσία των επαναληπτικών διαδικασιών είναι ότι οι προδιαγραφές αναπτύσσονται σε συνδυασμό με το λογισμικό. Αυτό όμως έρχεται σε αντίθεση με το μοντέλο προμηθειών που ακολουθούν πολλοί οργανισμοί, όπου οι πλήρες προδιαγραφές του συστήματος αποτελούν μέρος του συμβολαίου ανάπτυξης. Στη βαθμιαία προσέγγιση δεν υπάρχουν πλήρεις προδιαγραφές μέχρι να καθοριστεί και η τελευταία επαύξηση. Αυτό απαιτεί μια νέα μορφή συμβολαίου, την οποία ίσως δυσκολευτούν να υιοθετήσουν οι μεγάλοι πελάτες, όπως δημόσιες υπηρεσίες.

2.8 Βαθμιαία παράδοση

Η ανάπτυξη με το μοντέλο καταρράκτη απαιτεί από τους πελάτες να έχουν καταλήξει οριστικά σε ένα σύνολο απαιτήσεων για τα σύστημα προτού αρχίσει ο σχεδιασμός, και από τους σχεδιαστές να έχουν καταλήξει σε συγκεκριμένες στρατηγικές πριν από την υλοποίηση. Αν πραγματοποιηθούν αλλαγές στις απαιτήσεις, είναι αναγκαίο να αναθεωρηθούν απαιτήσεις, σχεδιασμός, και υλοποίηση. Παρόλα αυτά, ο διαχωρισμός μεταξύ σχεδιασμού και υλοποίησης οδηγεί γενικά σε καλά τεκμηριωμένα συστήματα, επιδεκτικά σε αλλαγές. Αντίθετα, μια εξελικτική προσέγγιση ανάπτυξης επιτρέπει τη λήψη καθυστερημένων αποφάσεων για τις απαιτήσεις και το σχεδιασμό, αλλά ωστόσο οδηγεί σε λογισμικό που μπορεί να διαθέτει κακή δομή, να είναι δυσνόητο, και να συντηρείται δύσκολα.

Η βαθμιαία παράδοση (Εικόνα 2.8) είναι μια ενδιάμεση προσέγγιση που συνδυάζει τα πλεονεκτήματα των μοντέλων αυτών. Κατά τη διαδικασία της βαθμιαίας παράδοσης, οι πελάτες προσδιορίζουν σε γενικές γραμμές τις υπηρεσίες που πρέπει να παρέχονται από το σύστημα. Καθορίζουν ποιες από τιε υπηρεσίες είναι οι σημαντικότερες και ποιες είναι λιγότερο σημαντικές γι' αυτούς. Στη συνέχεια ορίζεται μια σειρά από παραδοτέες επαυξήσεις (increments), κάθε μια από τις οποίες παρέχει ένα υποσύνολο των λειτουργιών του συστήματος. Η κατανομή των υπηρεσιών στις επαυξήσεις εξαρτάται από την προτεραιότητα της κάθε υπηρεσίας. Οι υπηρεσίες υψηλότερης προτεραιότητας παραδίδονται πρώτες.



ΕΙΚΟΝΑ 2.8 *Βαθμιαία παράδοση*

Αφού προσδιοριστούν οι επαυξήσεις του συστήματος, ορίζονται λεπτομερώς οι απαιτήσεις για τις υπηρεσίες που θα πρέπει να παραδοθούν με την πρώτη επαύξηση, και κατόπιν η επαύξηση αυτή αναπτύσσεται. Κατά την ανάπτυξη, μπορεί να πραγματοποιηθεί περαιτέρω ανάλυση απαιτήσεων για τις επόμενες επαυξήσεις, όμως αλλαγές απαιτήσεων της τρέχουσας επαύξησης δεν είναι αποδεκτές.

Αφού μια επαύξηση ολοκληρωθεί και παραδοθεί, οι πελάτες μπορούν να τη χρησιμοποιήσουν. Αυτό σημαίνει ότι ένα μέρος των λειτουργικών δυνατοτήτων του

συστήματος παραδίδεται από νωρίς. Οι πελάτες μπορούν να πειραματιστούν με το σύστημα, πράγμα που τους βοηθάει να αποσαφηνιστούν τις απαιτήσεις τους για επόμενες επαυξήσεις και για επόμενες εκδόσεις της τρέχουσας επαύξησης. Καθώς οι νέες επαυξήσεις ολοκληρώνονται, ενοποιούνται με τις υπάρχουσες και έτσι η λειτουργικότητα του συστήματος βελτιώνεται με κάθε επαύξηση που παραδίδεται. Οι κοινές υπηρεσίες μπορεί να υλοποιηθούν σε πρώιμο στάδιο της διαδικασίας, ή να υλοποιούνται βαθμιαία, ανάλογα με τις απαιτήσεις λειτουργικότητας που έχει κάθε επαύξηση.

Αυτή η διαδικασία βαθμιαίας ανάπτυξης διαθέτει μια σειρά πλεονεκτήματα:

1. Οι πελάτες δεν χρειάζεται να περιμένουν μέχρι να παραδοθεί ολόκληρο το σύστημα για να είναι σε θέση να το αξιοποιήσουν. Η πρώτη επαύξηση ικανοποιεί τις πιο κρίσιμες απαιτήσεις τους, ώστε να μπορούν να χρησιμοποιήσουν το λογισμικό αμέσως.
2. Οι πελάτες μπορεί να χρησιμοποιούν τις πρώτες επαυξήσεις ως πρωτότυπα και να αποκτούν πείρα στην οποία θα βασίζουν τις απαιτήσεις τους για επόμενες επαυξήσεις του συστήματος.
3. Υπάρχει μικρότερος κίνδυνος ολικής αποτυχίας του έργου. Αν και είναι δυνατό να παρουσιαστούν προβλήματα σε κάποιες επαυξήσεις, το πιθανότερο είναι κάποιες επαυξήσεις να παραδοθούν στον πελάτη με επιτυχία.
4. Επειδή οι υπηρεσίες υψηλότερης προτεραιότητας παραδίδονται πρώτες, και οι επόμενες επαυξήσεις ενοποιούνται με εκείνες, είναι αναπόφευκτο οι σημαντικότερες υπηρεσίες του υπηρεσίες του συστήματος να δοκιμάζονται περισσότερο. Αυτό σημαίνει ότι είναι λιγότερο πιθανό οι πελάτες να αντιμετωπίσουν αστοχίες του λογισμικού στα πιο σημαντικά μέρη του συστήματος.

Ωστόσο, η βαθμιαία παράδοση έχει και προβλήματα. Οι επαυξήσεις θα πρέπει να είναι σχετικά μικρές (όχι πάνω από 20.000 γραμμές κώδικα), και κάθε μία τους πρέπει να παρέχει κάποιες λειτουργικές δυνατότητες του συστήματος. Ο επιμερισμός των απαιτήσεων του πελάτη σε επαυξήσεις του σωστού μεγέθους μπορεί να αποδεχτεί δύσκολος. Επίσης, τα περισσότερα συστήματα απαιτούν ένα σύνολο βασικών βοηθητικών λειτουργιών που χρησιμοποιούνται από διάφορα μέρη τους. Επειδή οι απαιτήσεις δεν ορίζονται λεπτομερώς παρά μόνο όταν πρόκειται να υλοποιηθεί μια

επαύξηση, μπορεί να είναι δύσκολο να προσδιοριστούν οι κοινές βοηθητικές λειτουργίες τις οποίες χρειάζονται όλες οι επαυξήσεις.

Έχει αναπτυχθεί μια παραλλαγή της βαθμιαίας προσέγγισης, που ονομάζεται ακραίος προγραμματισμός-extreme programming-(Beck[2], 2000). Η προσέγγιση αυτή στηρίζεται στην ανάπτυξη και παράδοση πολύ μικρών επαυξήσεων λειτουργικών δυνατοτήτων, τη συμμετοχή του πελάτη στη διαδικασία, τη σταθερή βελτίωση του κώδικα, και τον προγραμματισμό σε ζεύγη.

2.9 Το σπειροειδής μοντέλο

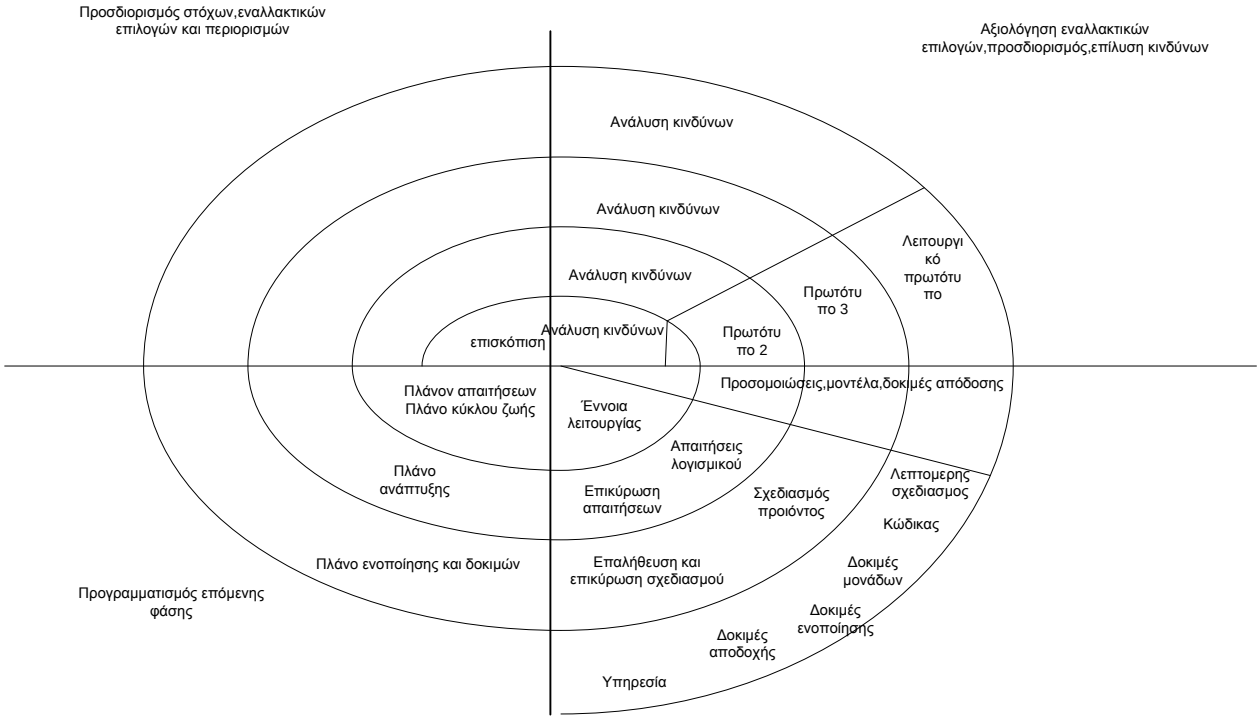
Το σπειροειδής μοντέλο (spiral model) της διαδικασίας παραγωγής λογισμικού (Εικόνα 2.9) προτάθηκε αρχικά από τον Boehm[3] (Boehm, 1988). Αντί η διαδικασία παραγωγής λογισμικού να αναπαριστά ως ακολουθία δραστηριοτήτων με κάποιες επανόδους από τη μία δραστηριότητα στην άλλη, έχει τη μορφή ελικοειδούς γραμμής. Κάθε βρόχος του σπειροειδούς αντιπροσωπεύει και μία φάση της διαδικασίας παραγωγής λογισμικού. Έτσι, ο εσώτερος βρόχος θα μπορούσε να αφορά την μελέτη σκοπιμότητας του συστήματος, ο επόμενος βρόχος τον καθορισμό των απαιτήσεων, ο επόμενος το σχεδιασμό του συστήματος κ.λπ.

Κάθε βρόχος του σπειροειδούς μοντέλου διαιρείται σε τέσσερις τομείς:

1. Καθορισμός στόχων. Ορίζονται συγκεκριμένοι στόχοι για τη δεδομένη φάση του έργου. Προσδιορίζονται περιορισμοί για τη διαδικασία και για το προϊόν, και καταρτίζεται ένα λεπτομερές πλάνο διαχείρισης. Προσδιορίζονται οι κίνδυνοι του έργου. Μπορεί να προγραμματιστούν εναλλακτικές στρατηγικές, ανάλογα με τους κινδύνους.
2. Αξιολόγηση και περιορισμός κινδύνων. Για κάθε προσδιορισμένο κίνδυνο του έργου, διενεργείται λεπτομερής ανάλυση, και λαμβάνονται μέτρα για τον περιορισμό του κινδύνου. Για παράδειγμα, αν υπάρχει κίνδυνος να αποδειχτούν οι απαιτήσεις ακατάλληλες, μπορεί να αναπτυχθεί ένα πρωτότυπο του συστήματος.
3. Ανάπτυξη και επικύρωση. Μετά την εκτίμηση των κινδύνων, επιλέγεται ένα μοντέλο για την ανάπτυξη του συστήματος. Για παράδειγμα, αν οι βασικότεροι κίνδυνοι αφορούν τη διασύνδεση χρήστη, ένα κατάλληλο μοντέλο ανάπτυξης θα ήταν η εξελικτική κατασκευή πρωτοτύπων. Αν το

κύριο μέλημα είναι οι κίνδυνοι της ασφάλειας, ένα μοντέλο ανάπτυξης που βασίζεται σε τυπικούς μετασχηματισμούς μπορεί να είναι καταλληλότερο κοκ. Σε περίπτωση που ο κυριότερος κίνδυνος που έχει προσδιοριστεί είναι η ενοποίηση των υποσυστημάτων, το καταλληλότερο μοντέλο ανάπτυξης θα μπορούσε να είναι το μοντέλο καταρράκτη.

4. Προγραμματισμός. Γίνεται μια επισκόπηση του έργου, και αποφασίζεται αν θα προχωρήσει στο επόμενο βρόχο του σπειροειδούς. Αν αποφασιστεί η συνέχεια του έργου, καταρτίζονται πλάνα για την επόμενη φάση.



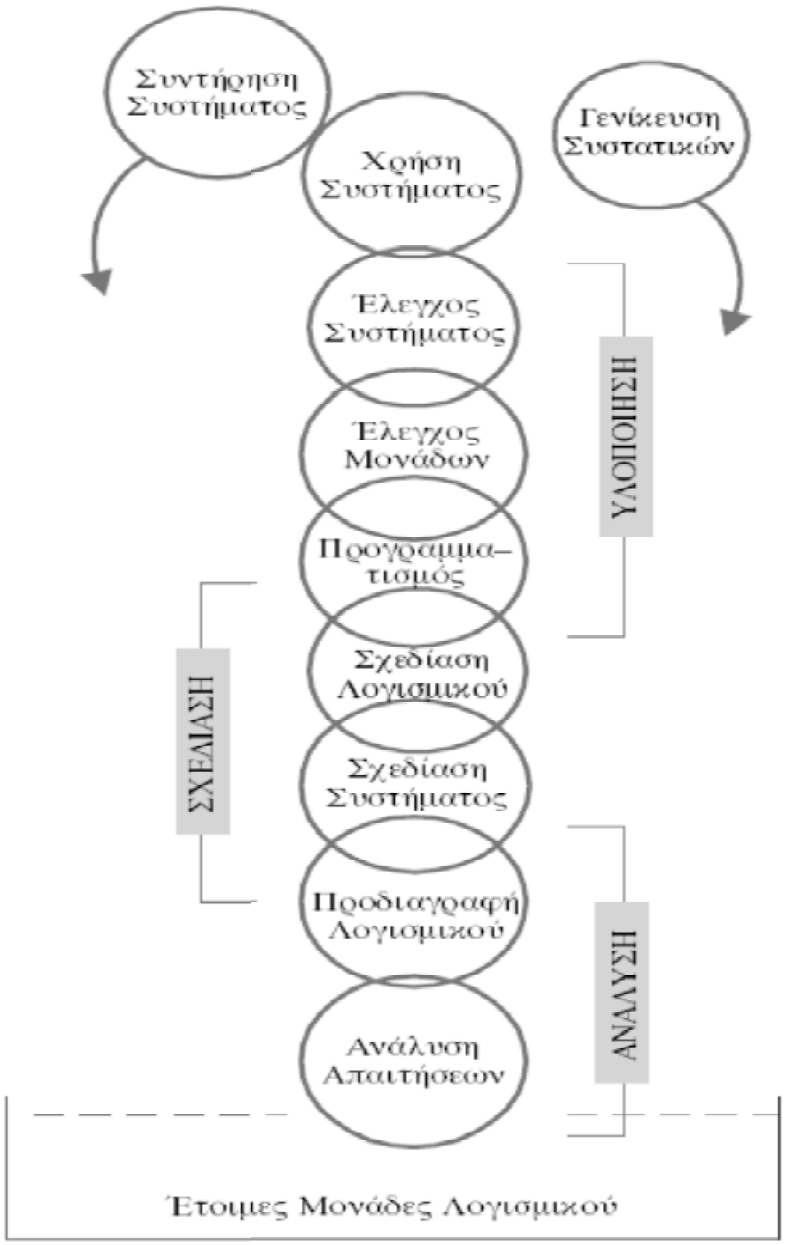
ΕΙΚΟΝΑ 2.9 *Το σπειροειδές μοντέλο κύκλου ζωής λογισμικού*

Η σημαντικότερη διαφορά μεταξύ του σπειροειδούς μοντέλου και των άλλων μοντέλων διαδικασιών παραγωγής λογισμικού είναι ότι το πρώτο αναγνωρίζει ρητά τους κινδύνους. Σε απλή γλώσσα, κίνδυνος είναι απλώς κάτι που μπορεί να πάει στραβά. Για παράδειγμα, αν υπάρχει πρόθεση να χρησιμοποιηθεί μια νέα γλώσσα προγραμματισμού, υπάρχει κίνδυνος οι διαθέσιμοι μεταγλωττιστές, να είναι αναξιόπιστοι ή να μην παράγουν αρκετά αποδοτικό αντικειμενικό κώδικα. Οι κίνδυνοι οδηγούν σε προβλήματα του έργου, όπως είναι υπερβάσεις του χρονοδιαγράμματος και του προϋπολογισμού, γι' αυτό και η ελαχιστοποίησή τους είναι μια πολύ σημαντική δραστηριότητα της διαχείρισης έργου.

Ένας κύκλος του σπειροειδούς σχήματος ξεκινάει με την λεπτομερή διατύπωση στόχων όπως η απόδοση και οι λειτουργικές δυνατότητες. Απαριθμούνται οι εναλλακτικοί τρόποι για την επίτευξη αυτών των στόχων, και μετά περιγράφονται οι περιορισμοί που επιβάλλονται σε κάθε στόχο. Αξιολογούνται όλες οι εναλλακτικές επιλογές για κάθε στόχο και προσδιορίζονται οι πηγές κινδύνων για το έργο. Το επόμενο βήμα είναι να επιλυθούν αυτοί οι κίνδυνοι με δραστηριότητες συγκέντρωσης πληροφοριών, όπως η λεπτομερής ανάλυση, η κατασκευή πρωτοτύπων, και η προσομοίωση. Αφού αξιολογηθούν οι κίνδυνοι, πραγματοποιείται ένα τμήμα της ανάπτυξης, ακολουθούμενο από δραστηριότητες προγραμματισμού για την επόμενη φάση της διαδικασίας.

2.10 Το μοντέλο του πίδακα

Αρκετά μοντέλα κύκλου ζωής που έχουν προταθεί αποτελούν παραλλαγές αυτών που αναφέρθηκαν, τα χαρακτηριστικά των οποίων υποβάλλονται από τις μεθοδολογίες ανάπτυξης. Οι πρώτες προσεγγίσεις του θέματος με βάση την *αντικειμενοστρεφή* (object.oriented) τεχνολογία διαφοροποίησαν το παραπάνω σχήμα βασιζόμενες σε δύο ιδιαίτερα γνωρίσματά της: πρώτον, ότι οι έννοιες «ανάλυση - σχεδίαση. κωδικοποίηση» έρχονται στο αντικειμενοστρεφές παράδειγμα πολύ πιο κοντά και δεύτερον, ότι το αποτέλεσμα κάθε διαδικασίας κατασκευής λογισμικού είναι όχι μόνο ένα σύστημα, αλλά και επαναχρησιμοποιήσιμες μονάδες, οι οποίες μπορούν να χρησιμοποιηθούν από τις πρώτες φάσεις της ανάπτυξης μελλοντικών συστημάτων. Με τον τρόπο αυτό προέκυψε το *μοντέλο του πίδακα* (fountain model), που φαίνεται στο Σχήμα 2.10.



ΣΧΗΜΑ 2.10 *Το μοντέλο κύκλου ζωής του πίδακα, το οποίο βασίζεται στην αντικειμενοστρεφή τεχνολογία ανάπτυξης λογισμικού*

Κατά την ανάπτυξη παρατηρούνται επικαλύψεις των φάσεων «ανάλυση - σχεδίαση. κωδικοποίηση», οι οποίες φαίνονται με την επικάλυψη των κύκλων στο σχήμα. Κατά το τέλος της ανάπτυξης, ορισμένα από τα συστατικά λογισμικού που έχουν παραχθεί ενσωματώνονται σε μια «δεξαμενή» συστατικών και διατίθενται για να χρησιμοποιηθούν στην ανάπτυξη και νέων συστημάτων. Η ιδέα του μοντέλου κύκλου ζωής του πίδακα τονίζει περισσότερο τα επιθυμητά χαρακτηριστικά της μεθοδολογίας κατασκευής του λογισμικού σύμφωνα με την αντικειμενοστρεφή λογική, ήταν δε αρκετά επίκαιρη κατά την έκρηξη ενδιαφέροντος για την αντικειμενοστρεφή τεχνολογία στα τέλη της δεκαετίας του 80 και στις αρχές της δεκαετίας του 90.

ΚΕΦΑΛΑΙΟ 3 : Προδιαγραφή Απαιτήσεων

Οι απαιτήσεις ενός συστήματος λογισμικού είναι οι περιγραφές των υπηρεσιών που παρέχονται από το σύστημα και οι λειτουργικοί περιορισμοί του. Οι απαιτήσεις αυτές αντανακλούν τις ανάγκες των πελατών για ένα σύστημα λογισμικού που βοηθάει στην επίλυση κάποιων προβλημάτων, όπως ο έλεγχος μιας συσκευής, η υποβολή μιας παραγγελίας, ή η ανεύρεση πληροφοριών. Η διαδικασία του εντοπισμού, της ανάλυσης, της τεκμηρίωσης, και του ελέγχου αυτών των υπηρεσιών και των περιορισμών ονομάζεται τεχνολογία απαιτήσεων (requirements engineering). Το παρόν κεφάλαιο εστιάζεται στις ίδιες τις απαιτήσεις και στο πώς τις περιγράφουμε.

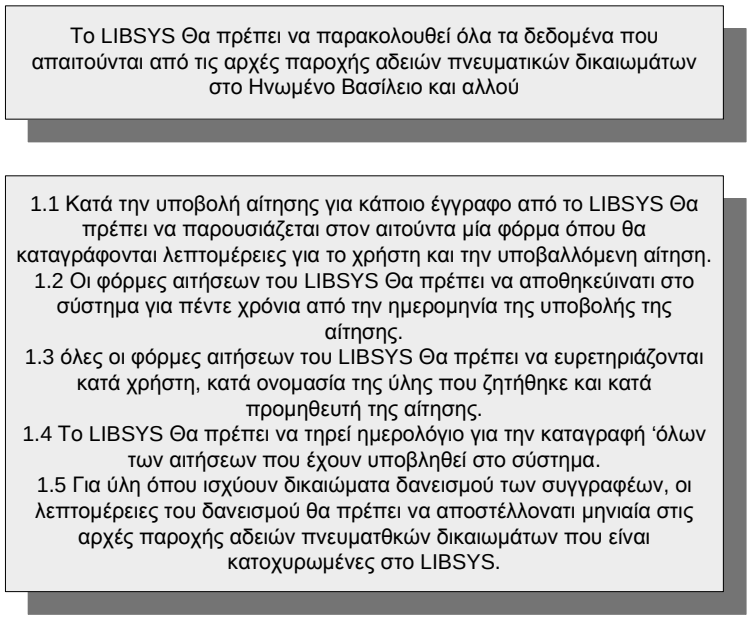
Ο όρος απαίτηση (requirement) δε χρησιμοποιείται με συνεπή τρόπο στη βιομηχανία λογισμικού. Σε μερικές περιπτώσεις, απαίτηση είναι απλώς μια υψηλού επιπέδου αφηρημένη δήλωση μιας υπηρεσίας την οποία θα πρέπει να παρέχει το σύστημα ή ενός περιορισμού του συστήματος. Στο άλλο άκρο, απαίτηση είναι ένας λεπτομερής, τυπικός ορισμός μιας λειτουργίας του συστήματος. Ο Davis[9] (Davis, 1993) εξηγεί γιατί υπάρχουν αυτές οι διαφορές:

Αν μια εταιρεία επιθυμεί να συνάψει σύμβαση για ένα μεγάλο έργο ανάπτυξης λογισμικού, πρέπει να ορίσει τις ανάγκες της με έναν αρκετά αφηρημένο τρόπο ώστε η λύση να μην είναι προκαθορισμένη. Οι απαιτήσεις πρέπει να γραφούν έτσι ώστε να επιτρέψουν σε πολλούς εργολάβους να κάνουν προσφορά για την σύμβαση, προσφέροντας, ίσως, διαφορετικούς τρόπους ικανοποίησης των αναγκών της εταιρείας-πελάτη. Αφού ανατεθεί η σύμβαση σε κάποιον εργολάβο, εκείνος πρέπει να γράψει έναν ορισμό συστήματος για τον πελάτη με περισσότερες λεπτομέρειες, ώστε ο πελάτης να μπορεί να καταλάβει και να επαληθεύσει τι θα κάνει το λογισμικό. Και τα δύο αυτά έγγραφα μπορεί να ονομαστούν έγγραφα απαιτήσεων για το σύστημα.

Κάποιο από τα προβλήματα που προκύπτουν στη διαδικασία παραγωγής απαιτήσεων είναι αποτέλεσμα της αδυναμίας να γίνει σαφής διαχωρισμός μεταξύ αυτών των διαφορετικών επιπέδων περιγραφής. Για τη διάκρισή τους, θα χρησιμοποιήσουμε τον όρο απαιτήσεις χρήστη (user requirements) για τις αφηρημένες απαιτήσεις υψηλού επιπέδου, και απαιτήσεις συστήματος (system requirements) για τη λεπτομερή περιγραφή για το τι πρέπει να κάνει το σύστημα. Οι απαιτήσεις χρήστη και οι απαιτήσεις συστήματος μπορούν να οριστούν ως εξής:

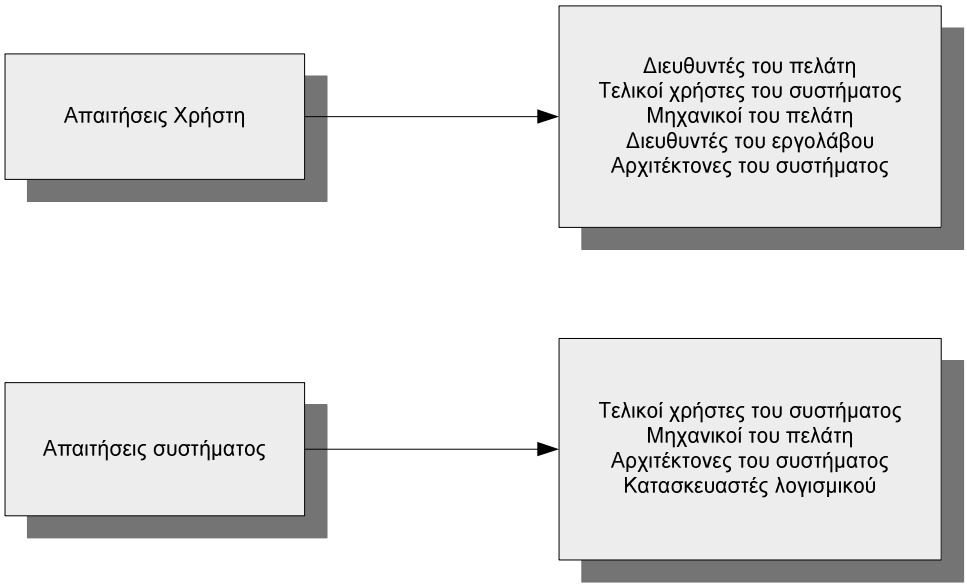
1. Οι απαιτήσεις χρήστη είναι δηλώσεις σε φυσική γλώσσα και διαγράμματα των υπηρεσιών που αναμένεται να παρέχει το σύστημα και των περιορισμών κάτω από τους οποίους πρέπει να λειτουργεί.
2. Οι απαιτήσεις συστήματος περιγράφουν με λεπτομέρειες τις λειτουργίες, τις υπηρεσίες, και τους λειτουργικούς περιορισμούς του συστήματος. Το έγγραφο απαιτήσεων συστήματος (το οποίο μερικές φορές ονομάζεται λειτουργικές προδιαγραφές) θα πρέπει να είναι ακριβές. Είναι απαραίτητο να ορίζει με ακρίβεια τι πρόκειται να υλοποιηθεί. Μπορεί να αποτελεί μέρος της σύμβασης μεταξύ του αγοραστή του συστήματος και των κατασκευαστών του λογισμικού.

Τα διαφορετικά επίπεδα προδιαγραφών συστήματος είναι χρήσιμα, γιατί παρέχουν πληροφορίες για το σύστημα σε διαφορετικές κατηγορίες αναγνωστών. Η Εικόνα 3.1 παρουσιάζει τη διάκριση μεταξύ απαιτήσεων χρήστη και συστήματος. Το παράδειγμα αυτό, που προέρχεται από ένα σύστημα βιβλιοθήκης, δείχνει πώς μια απαίτηση χρήστη μπορεί να αναπτυχθεί σε πολλές απαιτήσεις συστήματος. Στην Εικόνα 3.1 μπορείτε να δείτε ότι η απαίτηση χρήστη είναι πιο αφηρημένη, ενώ οι απαιτήσεις συστήματος προσθέτουν λεπτομέρειες, επεξηγώντας τις υπηρεσίες και τις λειτουργίες που πρέπει να παρέχονται από το σύστημα το οποίο πρόκειται να αναπτυχθεί.



ΕΙΚΟΝΑ 3.1 *Απαιτήσεις χρήστη και συστήματος*

Οι απαιτήσεις πρέπει να γράφονται σε διάφορα επίπεδα λεπτομέρειας, γιατί χρησιμοποιούνται με διαφορετικό τρόπο από κάθε κατηγορία αναγνωστών. Η Εικόνα 3.2 παρουσιάζει τις κατηγορίες αναγνωστών για τις απαιτήσεις χρήστη και συστήματος. Οι αναγνώστες των απαιτήσεων χρήστη συνήθως δεν ασχολούνται με το πώς θα υλοποιηθεί το σύστημα και μπορεί να είναι διευθύνοντες οι οποίοι δεν ενδιαφέρονται για τις λεπτομερείς δυνατότητές του. Οι αναγνώστες των απαιτήσεων συστήματος πρέπει να γνωρίζουν με μεγαλύτερη ακρίβεια τι θα κάνει το σύστημα, επειδή ενδιαφέρονται για τον τρόπο που θα υποστηρίξει τις επιχειρηματικές διαδικασίες ή επειδή συμμετέχουν άμεσα στην υλοποίηση του συστήματος.



ΕΙΚΟΝΑ 3.2 *Αναγνώστες των διαφορετικών τύπων προδιαγραφών*

3.1 Λειτουργικές και μη λειτουργικές απαιτήσεις

Οι απαιτήσεις ενός συστήματος λογισμικού συνήθως κατατάσσονται σε λειτουργικές απαιτήσεις, μη λειτουργικές απαιτήσεις, ή απαιτήσεις πεδίου:

1. Λειτουργικές απαιτήσεις (functional requirements). Πρόκειται για δηλώσεις που ορίζουν ποιες υπηρεσίες θα πρέπει να παρέχει το σύστημα, πώς θα πρέπει να αντιδρά σε συγκεκριμένες εισόδους, και πώς θα πρέπει να συμπεριφέρεται σε συγκεκριμένες καταστάσεις. Σε μερικές περιπτώσεις, οι λειτουργικές απαιτήσεις μπορούν επίσης να δηλώσουν ρητά τι δε θα πρέπει θα κάνει το σύστημα.
2. Μη λειτουργικές απαιτήσεις (non-functional requirements). Πρόκειται για περιορισμούς στις υπηρεσίες ή τις λειτουργίες που προσφέρει το σύστημα. Περιλαμβάνουν χρονικούς περιορισμός, περιορισμός της διαδικασίας ανάπτυξης, και πρότυπα. Οι μη λειτουργικές απαιτήσεις συχνά έχουν εφαρμογή στο σύστημα ως σύνολο, και δεν αφορούν μεμονωμένα χαρακτηριστικά ή υπηρεσίες του.
3. Απαιτήσεις πεδίου (domain requirements). Πρόκειται για απαιτήσεις που προέρχονται από το πεδίο εφαρμογής του συστήματος και να αντανακλούν

χαρακτηριστικά και περιορισμούς αυτού του πεδίου. Μπορεί να είναι λειτουργικές ή μη λειτουργικές απαιτήσεις.

Στην πραγματικότητα, η διάκριση μεταξύ των διαφόρων τύπων απαιτήσεων δεν είναι τόσο ξεκάθαρη όσο υπονοούν αυτοί οι απλοί ορισμοί. Μια απαίτηση χρήστη που αφορά την ασφάλεια, για παράδειγμα, μπορεί να φαίνεται μη λειτουργική. Ωστόσο, όταν αναπτυχθεί σε περισσότερες λεπτομέρειες, η απαίτηση αυτή μπορεί να δημιουργήσει άλλες απαιτήσεις που είναι σαφώς λειτουργικές, όπως η ανάγκη να συμπεριληφθούν στο σύστημα δυνατότητες πιστοποίησης της ταυτότητας των χρηστών.

3.1.1 Λειτουργικές απαιτήσεις

Οι λειτουργικές απαιτήσεις ενός συστήματος περιγράφουν τι θα πρέπει να κάνει το σύστημα. Οι απαιτήσεις αυτές εξαρτώνται από τον τύπο του λογισμικού που αναπτύσσεται, από τους αναμενόμενους χρήστες του λογισμικού, και από τη γενική προσέγγιση που ακολουθεί ο οργανισμός όταν γράφει απαιτήσεις. Όταν εκφράζονται ως απαιτήσεις χρήστη, οι απαιτήσεις συνήθως περιγράφονται με αρκετά αφηρημένο τρόπο. Όταν όμως είναι λειτουργικές απαιτήσεις συστήματος, περιγράφουν με λεπτομέρειες τη λειτουργία του συστήματος, τις εισόδους και τις εξόδους του, τις εξαιρέσεις κοκ.

Οι λειτουργικές απαιτήσεις ενός συστήματος λογισμικού μπορούν να εκφραστούν με διάφορους τρόπους. Λόγου χάρη, δείτε μερικά παραδείγματα, λειτουργικών απαιτήσεων για ένα πανεπιστημιακό σύστημα βιβλιοθήκης που ονομάζεται LIBSYS,το οποίο χρησιμοποιείται από τους φοιτητές και το εκπαιδευτικό προσωπικό για την παραγγελία βιβλίων και εγγράφων από άλλες βιβλιοθήκες.

1. Ο χρήστης θα πρέπει να είναι σε θέση να διενεργεί αναζητήσεις είτε σε ολόκληρο το αρχικό σύνολο βάσεων δεδομένων, είτε να επιλέγει ένα υποσύνολό τους.
2. Το σύστημα θα πρέπει να παρέχει στο χρήστη κατάλληλο λογισμικό προβολής για την ανάγνωση εγγράφων από την αποθήκη εγγράφων.
3. Σε κάθε παραγγελία θα πρέπει να αποδίδεται ένα μοναδικό αναγνωριστικό (ORDER_ID), το οποίο ο χρήστης θα μπορεί να αντιγράφει στο μόνιμο αποθηκευτικό χώρο του λογαριασμού.

Αυτές οι λειτουργικές απαιτήσεις χρήστη ορίζουν συγκεκριμένες δυνατότητας που θα παρέχονται από το σύστημα. Έχουν ληφθεί από το έγγραφο των απαιτήσεων χρήστη, και δείχνουν ότι οι λειτουργικές απαιτήσεις μπορούν να εκφράζονται σε διαφορετικά επίπεδα λεπτομέρειας (συγκρίνετε τις απαιτήσεις 1 και 3).

Το σύστημα LIBSYS είναι μια διασύνδεση για μια σειρά βάσεων δεδομένων άρθρων. Επιτρέπει στους χρήστες να λαμβάνουν αντίγραφα άρθρων δημοσιευμένων σε περιοδικά, εφημερίδες, και επιστημονικές εκδόσεις.

Η έλλειψη ακρίβειας στις προδιαγραφές απαιτήσεων αποτελεί αιτία πολλών προβλημάτων στην τεχνολογία λογισμικού. Είναι φυσικό για κάποιον που αναπτύσσει ένα σύστημα να ερμηνεύσει μια ασαφή απαίτηση με τέτοιον τρόπο ώστε να απλοποιήσει την υλοποίησή της. Πολλές φορές όμως, δεν είναι αυτό που θέλει ο πελάτης. Χρειάζεται να οριστούν νέες απαιτήσεις και να γίνουν αλλαγές στο σύστημα. Φυσικά, αυτό καθυστερεί την παράδοση του συστήματος και αυξάνει το κόστος.

Ας εξετάσουμε την δεύτερη απαίτηση στο παράδειγμα του συστήματος βιβλιοθήκης, όπου γίνετε λόγος για «κατάλληλο λογισμικό προβολής» που θα παρέχεται από το σύστημα. Το σύστημα βιβλιοθήκης μπορεί να παραδίδει έγγραφα σε μια ποικιλία μορφών, ο σκοπός αυτής της απαίτησης είναι να εξασφαλίσει ότι θα υπάρχει διαθέσιμο λογισμικό προβολής για όλες αυτές τις μορφές. Όμως η απαίτηση είναι διατυπωμένη με ασαφή τρόπο, δεν ξεκαθαρίζει ότι θα πρέπει να υπάρχει λογισμικό προβολής για κάθε μορφή εγγράφου. Ένας προγραμματιστής, κάτω από την πίεση του χρονοδιαγράμματος, μπορεί απλώς να προσθέσει μια λειτουργία προβολής κειμένου και να ισχυριστεί ότι η απαίτηση ικανοποιήθηκε.

Βασικά, οι προδιαγραφές των λειτουργικών απαιτήσεων ενός συστήματος θα πρέπει να είναι πλήρεις και συνεπείς. Πληρότητα (completeness) σημαίνει ότι θα πρέπει να έχουν οριστεί όλες οι υπηρεσίες που απαιτούνται από το χρήστη. Συνέπεια (consistency) σημαίνει ότι οι απαιτήσεις δεν πρέπει να έχουν ορισμούς που αντιφάσκουν. Στην πράξη, σε μεγάλα και σύνθετα συστήματα είναι πρακτικά αδύνατο να επιτευχθεί και πληρότητα και συνέπεια των απαιτήσεων.

Ένας λόγος για τον οποίο συμβαίνει αυτό είναι ότι είναι εύκολο να γίνουν λάθη και παραλείψεις όταν γράφονται προδιαγραφές για τέτοιου είδους συστήματα. Ένας άλλος λόγος είναι ότι οι ενδιαφερόμενοι του συστήματος έχουν διαφορετικές και συχνά αντιφατικές ανάγκες. Αυτές οι αντιφάσεις μπορεί να μην είναι προφανείς όταν καθορίζονται αρχικά απαιτήσεις, με αποτέλεσμα να περιλαμβάνονται στις προδιαγραφές αντιφατικές

απαιτήσεις. Τα προβλήματα μπορεί να αναδυθούν μόνο μετά από βαθύτερη ανάλυση ή, μερικές φορές, μετά την ολοκλήρωση της ανάπτυξης και την παράδοση του συστήματος στον πελάτη.

3.1.2 Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις, όπως υπονοεί το όνομά τους, είναι απαιτήσεις που δεν αφορούν άμεσα τις συγκεκριμένες λειτουργίες που παρέχονται από το σύστημα. Μπορεί να σχετίζονται με ανακύπτουσες ιδιότητες του συστήματος, όπως η αξιοπιστία, ο χρόνος απόκρισης, και ο αποθηκευτικός χώρος. Ή αλλιώς, μπορεί να ορίζουν περιορισμούς για το σύστημα, όπως οι δυνατότητες των συσκευών εισόδου-εξόδου και οι αναπαραστάσεις δεδομένων που χρησιμοποιούνται στις διασυνδέσεις του συστήματος.

Οι μη λειτουργικές απαιτήσεις σπανίως σχετίζονται με μεμονωμένα χαρακτηριστικά του συστήματος. Οι απαιτήσεις αυτές καθορίζουν ή περιορίζουν τις ανακύπτουσες ιδιότητες του συστήματος ως σύνολο, όπως είδαμε στο Κεφάλαιο 2. Επομένως, μπορεί να καθορίζουν την απόδοση, την προστασία από εξωτερικούς κινδύνους, τη διαθεσιμότητα, και άλλες ανακύπτουσες ιδιότητες του συστήματος. Αυτό σημαίνει ότι συχνά είναι πιο κρίσιμες από τις μεμονωμένες λειτουργικές απαιτήσεις. Οι χρήστες του συστήματος συνήθως βρίσκουν τρόπους να αναπληρώσουν κάποια λειτουργία του συστήματος που δεν ικανοποιεί πραγματικά τις ανάγκες τους. Όμως η αποτυχία να ικανοποιηθεί μια μη λειτουργική απαίτηση μπορεί να σημαίνει ότι ολόκληρο το σύστημα είναι άχρηστο. Για παράδειγμα, αν ένα σύστημα αεροσκάφους δεν ικανοποιεί τις απαιτήσεις αξιοπιστίας, δε θα πιστοποιηθεί ως λειτουργικά ασφαλές, αν ένα σύστημα ελέγχου πραγματικού χρόνου δεν ικανοποιεί τια απαιτήσεις απόδοσης, οι λειτουργίες ελέγχου δε θα λειτουργούν σωστά.

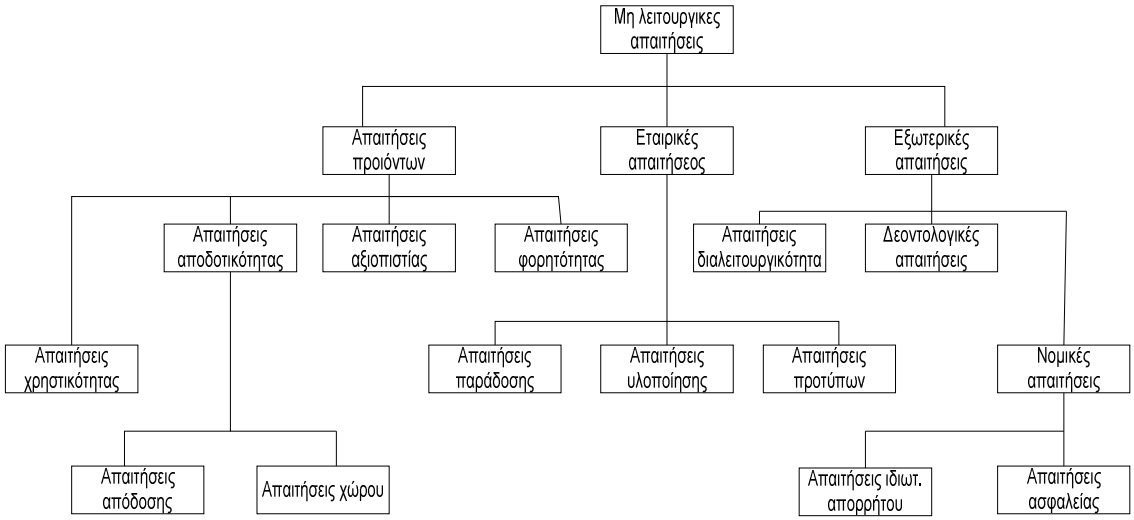
Οι μη λειτουργικές απαιτήσεις δεν αφορούν μόνο το σύστημα λογισμικού που θα αναπτυχθεί. Κάποιες μη λειτουργικές απαιτήσεις μπορεί να θέτουν περιορισμούς στη διαδικασία που πρέπει να χρησιμοποιηθεί για την ανάπτυξη του συστήματος. Παραδείγματα τέτοιων απαιτήσεων είναι μια προδιαγραφή για τα πρότυπα ποιότητας που πρέπει να χρησιμοποιηθούν στη διαδικασία, μια προδιαγραφή που ορίζει ότι ο σχεδιασμός πρέπει να παραχθεί με μια συγκεκριμένη εργαλειοθήκη CASE, και μια περιγραφή της διαδικασίας που θα πρέπει να ακολουθηθεί.

Μη λειτουργικές απαιτήσεις προκύπτουν από τις ανάγκες των χρηστών, από περιορισμούς του προϋπολογισμού, από πολιτικές της εταιρείας, από την ανάγκη λειτουργικότητας με άλλα συστήματα λογισμικού ή υλικού, ή από εξωτερικούς παράγοντες

όπως οι κανονισμοί προστασίας ή η νομοθεσία περί προσωπικού απορρήτου. Η Εικόνα 3.3 παρουσιάζει μια κατηγοριοποίηση των μη λειτουργικών απαιτήσεων. Από αυτό το διάγραμμα μπορείτε να δείτε ότι οι μη λειτουργικές απαιτήσεις ίσως να προέρχονται από απαιτούμενα χαρακτηριστικά του λογισμικού (απαιτήσεις προϊόντος), από την εταιρεία που αναπτύσσει το λογισμικό (εταιρικές απαιτήσεις), ή από εξωτερικές πηγές.

Οι τύποι των μη λειτουργικών απαιτήσεων είναι οι εξής:

1. Απαιτήσεις προϊόντων. Οι απαιτήσεις αυτές καθορίζουν τη συμπεριφορά του προϊόντος. Τέτοια παραδείγματα είναι: οι απαιτήσεις απόδοσης για το ποσό γρήγορα πρέπει να λειτουργεί το σύστημα και πόση μνήμη χρειάζεται, οι απαιτήσεις αξιοπιστίας που ορίζουν την αποδεκτή συχνότητα αστοχιών, οι απαιτήσεις φορητότητας, οι απαιτήσεις χρηστικότητας.
2. Εταιρικές απαιτήσεις. Οι απαιτήσεις αυτές πηγάζουν από την πολιτική και τις διαδικασίες της εταιρείας-πελάτη και της εταιρείας-κατασκευαστή. Τέτοια παραδείγματα είναι: τα πρότυπα διαδικασιών που πρέπει να χρησιμοποιηθούν, οι απαιτήσεις της υλοποίησης όπως η γλώσσα προγραμματισμού ή η μέθοδος σχεδιασμού που θα χρησιμοποιηθεί, οι απαιτήσεις παράδοσης που καθορίζουν το πότε θα παραδοθούν το προϊόν και η τεκμηρίωσή του.
3. Εξωτερικές απαιτήσεις. Αυτή η μεγάλη κατηγορία καλύπτει όλες τις απαιτήσεις που προέρχονται από παράγοντες εξωτερικούς προς το σύστημα και τη διαδικασία ανάπτυξής του. Τέτοιες απαιτήσεις μπορεί να είναι: οι απαιτήσεις διαλειτουργικότητας που ορίζουν πώς αλληλεπιδρά το σύστημα με συστήματα άλλων οργανισμών, οι νομικές απαιτήσεις που πρέπει να ακολουθηθούν για να διασφαλιστεί ότι το σύστημα λειτουργεί στα πλαίσια του νόμου, οι δεοντολογικές απαιτήσεις. Οι δεοντολογικές απαιτήσεις είναι απαιτήσεις που διασφαλίζουν ότι το σύστημα θα είναι αποδεκτό από τους χρήστες του και γενικά από το κοινό.



ΕΙΚΟΝΑ 3.3 *Τύποι μη λειτουργικών απαιτήσεων*

Ένα συνηθισμένο πρόβλημα με τις λειτουργικές απαιτήσεις είναι ότι μπορεί να είναι δύσκολο να επαληθευτούν. Οι χρήστες ή οι πελάτες συχνά διατυπώνουν αυτές τις απαιτήσεις ως γενικούς στόχους όπως η ευκολία χρήσης, η ικανότητα του συστήματος να ανακάμπτει από αστοχίες, ή η ταχεία απόκριση στο χρήση. Αυτοί οι αόριστοι στόχοι προκαλούν προβλήματα στους κατασκευαστές του συστήματος, καθώς αφήνουν περιθώρια διαφορετικών ερμηνειών με επακόλουθες διαφωνίες όταν παραδοθεί το σύστημα. Για δείτε μια τέτοια περίπτωση, εξετάστε την Εικόνα 3.4. Περιέχει ένα στόχο που σχετίζεται με τη χρηστικότητα ενός συστήματος ελέγχου κυκλοφορίας, και αποτελεί τυπική περίπτωση του πώς ένας χρήστης μπορεί να εκφράσει τις απαιτήσεις χρηστικότητας. Παρόλο που αντικειμενική επαλήθευση του στόχου είναι αδύνατη, μπορούμε να σχεδιάσουμε δοκιμές οι οποίες καταμετρούν τα λάθη που κάνουν οι ελεγκτές με τη χρήση ενός προσομοιωτή συστήματος.

Στόχος συστήματος

Το σύστημα θα πρέπει να μπορεί να χρησιμοποιηθεί εύκολα από έμπειρους ελεγκτές και να είναι οργανωμένο με τέτοιον τρόπο ώστε να ελαχιστοποιούνται τα λάθη των χρηστών.

Επαληθεύσιμη μη λειτουργική απαίτηση

Οι έμπειροι ελεγκτές θα πρέπει να είναι σε θέση να χρησιμοποιούν όλες τις λειτουργίες του συστήματος μετά από συνολική εκπαίδευση δύο ωρών. Μετά από αυτή την εκπαίδευση, ο μέσος όρος των λαθών που διαπράττονται από έμπειρους

χρήστες θα πρέπει να μην υπερβαίνει τα δύο ημερησίως.

ΕΙΚΟΝΑ 3.4 *Στόχοι του συστήματος και επαληθεύσιμες απαιτήσεις*

Όπου είναι δυνατό, θα πρέπει ν εκφράζετε τις μη λειτουργικές απαιτήσεις με ποσοτικό τρόπο, ώστε να μπορούν να δοκιμαστούν αντικειμενικά. Η εικόνα 6.6 δείχνει μια σειρά από δυνατές μετρήσεις που μπορείτε να χρησιμοποιήσετε για να καθορίσετε μη λειτουργικές ιδιότητες του συστήματος. Αυτά τα χαρακτηριστικά μπορούν να μετρηθούν κατά τις δοκιμές του συστήματος, ώστε να ελεγχθεί αν το σύστημα ικανοποιεί ή όχι τις μη λειτουργικές του απαιτήσεις.

Στην πράξη, όμως, οι πελάτες ενός συστήματος μπορεί να μην έχουν τη δυνατότητα να μεταφράσουν τους στόχους τους σε ποσοτικές απαιτήσεις. Για κάποιους στόχους, όπως η συντηρησιμότητα (maintainability), δεν υπάρχουν μετρικές που μπορούν να χρησιμοποιηθούν. Σε άλλες περιπτώσεις, ακόμη και όταν είναι δυνατές οι ποσοτικές προδιαγραφές, οι πελάτες μπορεί να μην είναι σε θέση να συσχετίσουν τις ανάγκες τους με αυτές. Δεν καταλαβαίνουν τι σημαίνει κάποιος αριθμός που ορίζει, για παράδειγμα, την απαιτούμενη αξιοπιστία, με βάση την καθημερινή τους εμπειρία σε συστήματα υπολογιστών. Επιπλέον, το κόστος της αντικειμενικής επαλήθευσης των ποσοτικών μη λειτουργικών απαιτήσεων μπορεί να είναι πολύ υψηλό, και οι πελάτες που πληρώνουν για το σύστημα μπορεί να θεωρήσουν ότι αυτό το κόστος δεν δικαιολογείται.

Γι’ αυτό, τα έγγραφα απαιτήσεων συχνά περιλαμβάνουν δηλώσεις στόχων ανάμικτα με απαιτήσεις. Οι στόχοι αυτοί μπορεί να είναι χρήσιμοι για τους κατασκευαστές επειδή παρέχουν ενδείξεις για τις προτεραιότητες του πελάτη. Ωστόσο, θα πρέπει πάντοτε να λέγετε στους πελάτες ότι επιδέχονται παρερμηνείες και δεν μπορούν να επαληθευθούν αντικειμενικά.

Οι μη λειτουργικές απαιτήσεις συχνά έρχονται σε αντίθεση και αλληλεπιδρούν με άλλες λειτουργικές ή μη λειτουργικές απαιτήσεις. Για παράδειγμα, μπορεί να αποτελεί απαίτηση ότι η μέγιστη μνήμη που χρησιμοποιείται από ένα σύστημα δε θα πρέπει να υπερβαίνει τα 4 Mbyte. Οι περιορισμοί μνήμης είναι συνηθισμένη σε ενσωματωμένα συστήματα (embedded systems), όπου ο χώρος ή το βάρος είναι περιορισμένα και ο αριθμός των τσιπ μνήμης ROM όπου αποθηκεύεται το λογισμικό του συστήματος πρέπει να ελαχιστοποιηθεί. Μια άλλη απαίτηση μπορεί να είναι ότι το σύστημα ότι το σύστημα θα πρέπει να γραφεί σε γλώσσα Ada, μια γλώσσα προγραμματισμού για την ανάπτυξη κρίσιμου

λογισμικού πραγματικού χρόνου. Ίσως όμως η μεταγλώττιση ενός προγράμματος Ada με την απαιτούμενη λειτουργικότητα σε λιγότερα από 4Mbyte να μην είναι δυνατή. Εκεί επομένως θα πρέπει να γίνει ένας συμβιβασμός μεταξύ των απαιτήσεων: να επιλεγεί μια εναλλακτική γλώσσα ανάπτυξης ή να προστεθεί στο σύστημα επιπλέον μνήμη.

Είναι χρήσιμο, αν μπορείτε, να ξεχωρίζετε τις λειτουργικές από τις μη λειτουργικές απαιτήσεις στο έγγραφο των απαιτήσεων. Στην πράξη, αυτό είναι δύσκολο. Αν οι μη λειτουργικές απαιτήσεις διατυπωθούν ξεχωριστά από τις λειτουργικές απαιτήσεις, μερικές φορές είναι δύσκολο να δει κανείς τις σχέσεις μεταξύ τους. Αν έχουν διατυπωθεί μαζί με τις λειτουργικές απαιτήσεις, μπορεί να δυσκολευτείτε να διαχωρίσετε τα λειτουργικά από τα μη λειτουργικά ζητήματα και να προσδιορίσετε τις απαιτήσεις που σχετίζονται με το σύστημα ως το σύνολο. Ωστόσο, θα πρέπει να επισημαίνετε ρητά τις απαιτήσεις που σχετίζονται σαφώς με ανακύπτουσες ιδιότητες του συστήματος, όπως η απόδοση ή η αξιοπιστία. Αυτό μπορείτε να το κάνετε τοποθετώντας τις σε μια ξεχωριστή ενότητα του εγγράφου των απαιτήσεων ή ξεχωρίζοντας τις με κάποιον τρόπο από τις άλλες απαιτήσεις του συστήματος.

Μη λειτουργικές απαιτήσεις, όπως οι απαιτήσεις ασφάλειας και προστασίας από εξωτερικούς κινδύνους, είναι ιδιαίτερα σημαντικές για κρίσιμα συστήματα.

Ιδιότητα	Μετρική
Ταχύτητα	Συναλλαγές/Δευτερόλεπτο Χρόνος απόκρισης χρήστη Χρόνος ανανέωσης οθόνης
Μέγεθος	K byte Αριθμός τσιπ μνήμης RAM
Ευχρηστία	Χρόνος εκπαίδευσης Αριθμός πλαισίων βοήθειας
Αξιοπιστία	Μέσος χρόνος μεταξύ αστοχιών Πιθανότητα μη διαθεσιμότητας Συχνότητα εμφάνισης αστοχιών Διαθεσιμότητα
Ανθεκτικότητα	Χρόνος επανεκκίνησης μετά από αστοχία Ποσοστό συμβάντων που προκαλούν αστοχία Πιθανότητα βλάβης δεδομένων από αστοχία

Φορητότητα	Ποσοστό εντολών που εξαρτώνται από το σύστημα προορισμού Αριθμός συστημάτων προορισμού

ΕΙΚΟΝΑ 3.5 *Μετρικές για τον καθορισμό μη λειτουργικών απαιτήσεων*

3.2 Απαιτήσεις πεδίου

Οι απαιτήσεις πεδίου(domain requirements) προέρχονται από το πεδίο εφαρμογής του συστήματος, και όχι από συγκεκριμένες ανάγκες των χρηστών του. Συνήθως περιλαμβάνουν εξειδικευμένη ορολογία του πεδίου ή αναφορές σε έννοιες του πεδίου. Μπορεί να είναι νέες λειτουργικές απαιτήσεις από μόνες τους, να περιορίζουν υπάρχουσες λειτουργικές απαιτήσεις, ή να καθορίζουν το πώς πρέπει να εκτελούνται κάποιοι συγκεκριμένοι υπολογισμοί. Επειδή οι απαιτήσεις αυτές είναι εξειδικευμένες, συχνά οι μηχανικοί λογισμικού δυσκολεύονται να καταλάβουν τις σχέσεις τους με άλλες απαιτήσεις του συστήματος.

Οι απαιτήσεις πεδίου είναι σημαντικές επειδή συχνά αντανακλούν θεμελιώδεις έννοιες του πεδίου εφαρμογής. Αν δεν ικανοποιούνται αυτές οι απαιτήσεις, το σύστημα μπορεί να είναι αδύνατο να λειτουργήσει ικανοποιητικά. Το σύστημα LIBSYS περιλαμβάνει μια σειρά από απαιτήσεις πεδίου:

Θα πρέπει να υπάρχει μια τυποποιημένη διασύνδεση χρήστη για όλες τις βάσεις δεδομένων.
Λόγω περιορισμών πνευματικών δικαιωμάτων, κάποια έγγραφα πρέπει να διαγράφονται αμέσως μετά την άφιξή τους. Ανάλογα με τις απαιτήσεις του χρήστη, τα έγγραφα αυτά είτε θα τυπώνονται τοπικά στο διακομιστή του συστήματος για φυσική προώθηση στο χρήστη, είτε θα δρομολογούνται σε ένα δικτυακό εκτυπωτή.

Η πρώτη απαίτηση είναι σχεδιαστικός περιορισμός. Ορίζει ότι η διασύνδεση χρήστη προς τη βάση δεδομένων πρέπει να υλοποιηθεί σύμφωνα με ένα συγκεκριμένο πρότυπο των βιβλιοθηκών. Επομένως, οι κατασκευαστές θα πρέπει να μάθουν τα σχετικά με αυτό πρότυπο πριν αρχίσουν να σχεδιάζουν τη διασύνδεση. Η δεύτερη απαίτηση τέθηκε λόγω των νόμων περί πνευματικής ιδιοκτησίας που ισχύουν για την ύλη που χρησιμοποιείται στις βιβλιοθήκες. Ορίζει ότι το σύστημα πρέπει να περιλαμβάνει μια λειτουργία αυτόματης διαγραφής κατά την

εκτύπωση για κάποιες κατηγορίες εγγράφων. Αυτό σημαίνει ότι οι χρήστες του συστήματος βιβλιοθήκης δεν μπορούν να κρατήσουν ένα δικό τους ηλεκτρονικό αντίγραφο του εγγράφου.

Για ένα παράδειγμα απαίτησης πεδίου που καθορίζει το πώς εκτελείται ένας υπολογισμός, δείτε την Εικόνα 3.6 που προέρχεται από τις προδιαγραφές απαιτήσεων ενός αυτόματου συστήματος προστασίας τρένων. Το σύστημα αυτό σταματάει αυτόματα ένα τρένο αν περάσει από κόκκινο σήμα. Η απαίτηση ορίζει το πώς υπολογίζεται η επιβράδυνση του τρένου από το σύστημα. Χρησιμοποιεί εξειδικευμένη ορολογία του πεδίου. Για να την καταλάβετε, πρέπει να έχετε κάποια γνώση της λειτουργίας των σιδηροδρομικών συστημάτων και των χαρακτηριστικών των τρένων.

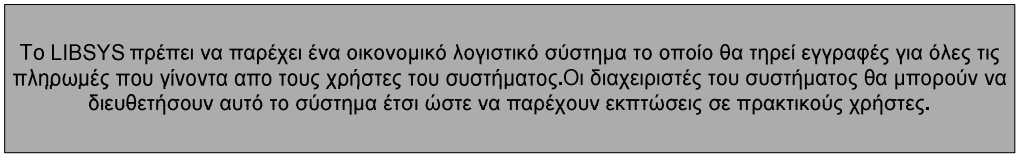
Η απαίτηση για το σύστημα τρένων επιδεικνύει ένα καίριο πρόβλημα των απαιτήσεων πεδίου. Είναι γραμμένες στη γλώσσα του πεδίου εφαρμογής (στη συγκεκριμένη περίπτωση, με τη μορφή μαθηματικών εξισώσεων), και συχνά είναι δύσκολα κατανοητές από τους μηχανικούς λογισμικού. Οι ειδικοί του πεδίου μπορεί να παραλείψουν πληροφορίες από μια απαίτηση επειδή απλώς είναι εντελώς προφανείς σε αυτούς. Ίσως όμως να μην είναι προφανείς στους κατασκευαστές του συστήματος, οι οποίοι κινδυνεύουν έτσι να υλοποιήσουν την απαίτηση με εσφαλμένο τρόπο.

3.3 Απαιτήσεις χρήστη

Το έγγραφο των απαιτήσεων χρήστη ενός συστήματος θα πρέπει να περιγράφει τις λειτουργικές και μη λειτουργικές απαιτήσεις με τέτοιον τρόπο ώστε να είναι κατανοητές από τους χρήστες του συστήματος χωρίς να χρειάζονται λεπτομερείς τεχνικές γνώσεις. Είναι απαραίτητο να καθορίζει μόνο την εξωτερική συμπεριφορά του συστήματος και να αποφεύγει, όσο είναι δυνατόν, σχεδιαστικά χαρακτηριστικά. Αν λοιπόν γράφετε απαιτήσεις χρήστη, δεν πρέπει να χρησιμοποιείτε ορολογία λογισμικού, δομημένους ή τυπικούς συμβολισμούς, ή να περιγράφετε την υλοποίηση του συστήματος. Πρέπει απλώς να γράφετε τις απαιτήσεις χρήστη σε απλή γλώσσα, με απλούς πίνακες και φόρμες και με κατανοητά διαγράμματα.

Η επιβράδυνση του τρένου θα πρέπει να υπολογίζεται ως:
$D_{train}=D_{control}+D_{gradient}$
Όπου $D_{gradient}$ είναι 9,81 ms^2 Ισοσταθμισμένη κλίση/ Alpha και όπου οι τιμές 9,81 ms^2/alpha είναι γνωστές για τους διαφόρους τύπους τρένων

ΕΙΚΟΝΑ 3.6 *Μία απαίτηση πεδίου από ένα σύστημα προστασίας τρένων*



ΕΙΚΟΝΑ 3.7 *Απαίτηση χρήστη για ένα λογιστικό σύστημα στο LIBSYS*

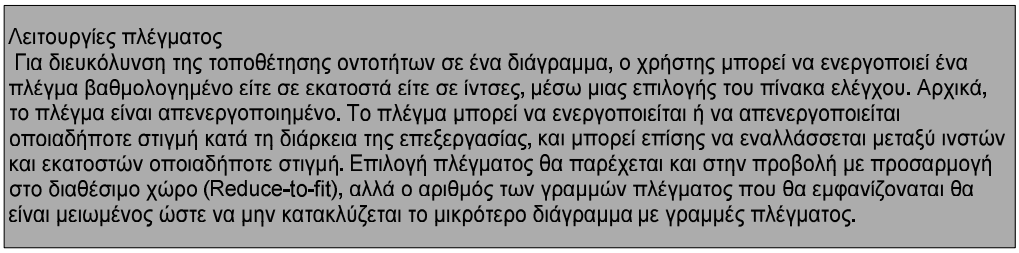
Ωστόσο, όταν οι απαιτήσεις γράφονται σε φυσική γλώσσα μπορεί να προκύψουν διάφορα προβλήματα:

1. Έλλειψη σαφήνειας. Μερικές φορές είναι δύσκολο να χρησιμοποιηθεί η γλώσσα με ακριβή και μονοσήμαντο τρόπο χωρίς να γίνει το έγγραφο φλύαρο και δυσανάγνωστο.
2. Σύγχυση των απαιτήσεων. Οι λειτουργικές απαιτήσεις, οι μη λειτουργικές απαιτήσεις, οι στόχοι του συστήματος, και οι σχεδιαστικές πληροφορίες μπορεί να μη διακρίνονται σαφώς.
3. Συνδυασμός απαιτήσεων. Πολλές διαφορετικές απαιτήσεις μπορεί να εκφράζονται μαζί, με τη μορφή μιας μόνο απαίτησης.

Για να δείτε κάποια από αυτά τα προβλήματα, εξετάστε μία από τις απαιτήσεις για τη βιβλιοθήκη στην Εικόνα 3.7.

Η απαίτηση αυτή περιλαμβάνει και εννοιολογικές και λεπτομερείς πληροφορίες. Εκφράζει την έννοια ότι ένα λογιστικό σύστημα πρέπει να συμπεριληφθεί εγγενές τμήμα του LIBSYS. Ωστόσο, περιλαμβάνει επίσης την λεπτομέρεια ότι το λογιστικό σύστημα πρέπει να υποστηρίζει εκπτώσεις για τους τακτικούς χρήστες του LIBSYS. Αυτή η λεπτομέρεια θα ήταν καλύτερο να είχε αφεθεί για τις προδιαγραφές των απαιτήσεων συστήματος.

Ο διαχωρισμός των απαιτήσεων χρήστη από τις πιο λεπτομερείς απαιτήσεις συστήματος στο έγγραφο των απαιτήσεων αποτελεί καλή πρακτική. Διαφορετικά, οι αναγνώστες των απαιτήσεων χρήστη που δεν έχουν τεχνικές γνώσεις μπορεί να κατακλυστούν με λεπτομέρειες οι οποίες στην πραγματικότητα ενδιαφέρουν μόνο τους τεχνικούς. Η Εικόνα 6.9 επιδεικνύει αυτή τη σύγχυση. Το παράδειγμα προέρχεται από ένα πραγματικό έγγραφο απαιτήσεων για ένα εργαλείο CASE το οποίο προορίζεται για την επεξεργασία μοντέλων σχεδιασμού λογισμικού. Ο χρήστης μπορεί να ορίσει ότι θα πρέπει να εμφανίζεται ένα πλέγμα έτσι ώστε οι οντότητες να μπορούν να τοποθετούνται στο διάγραμμα με ακρίβεια.



ΕΙΚΟΝΑ 3.8 *Απαίτηση χρήστη για πλέγμα επεξεργασίας*

Η πρώτη πρόταση αναμιγνύει τρία είδη απαιτήσεων.

1. Μια εννοιολογική, λειτουργική απαίτηση που ορίζει ότι το σύστημα επεξεργασίας θα πρέπει να παρέχει ένα πλέγμα. Παρουσιάζει μια αιτιολογία για αυτό.
2. Μια μη λειτουργική απαίτηση που δίνει λεπτομερείς πληροφορίες σχετικά με τις μονάδες του πλέγματος (εκατοστά ή ίντσες).
3. Μια μη λειτουργική απαίτηση για τη διασύνδεση χρήστη, που ορίζει πώς θα ενεργοποιείται ή θα απενεργοποιείται το πλέγμα από το χρήστη.

Η απαίτηση της Εικόνας 3.8 δίνει επίσης μερικές αλλά όχι όλες τις πληροφορίες για την αρχική κατάσταση. Ορίζει ότι αρχικά το πλέγμα θα είναι απενεργοποιημένο, ωστόσο δεν ορίζει ποιες μονάδες θα χρησιμοποιούνται κατά την ενεργοποίησή του. Παρέχει κάποιες λεπτομερείς πληροφορίες – ότι ο χρήστης μπορεί να εναλλάσσει τις μονάδες – αλλά όχι την απόσταση μεταξύ των γραμμών του πλέγματος.

Οι απαιτήσεις χρήστη που περιλαμβάνουν πάρα πολλές πληροφορίες περιορίζουν την ελευθερία του κατασκευαστή του συστήματος να δώσει καινοτόμες λύσεις σε προβλήματα του χρήστη, και είναι δυσνόητες. Μια απαίτηση χρήστη πρέπει απλώς να εστιάζεται στις πιο σημαντικές δυνατότητες που θα παρέχονται.

Όποτε είναι δυνατό, θα πρέπει να προσπαθείτε να δίνετε μια αντίστοιχη αιτιολογία σε κάθε απαίτηση χρήστη. Η αιτιολογία θα εξηγεί γιατί έχει τεθεί η απαίτηση, και είναι ιδιαίτερα χρήσιμη όταν αλλάζουν οι απαιτήσεις. Για παράδειγμα, η αιτιολογία της Εικόνας 3.10 αναγνωρίζει ότι ενώ ένα ενεργητικό πλέγμα, όπου τα τοποθετούμενα αντικείμενα στοιχίζονται αυτόματα επάνω σε μια γραμμή, θα ήταν χρήσιμο, αυτή η λύση έχει απορριφθεί σκοπίμως για χάρη της μη αυτόματης τοποθέτησης. Αν κάποια στιγμή αργότερα εμφανιστεί μια απαίτηση για αλλαγή αυτού του σημείου, θα είναι σαφές ότι η χρήση παθητικού πλέγματος ήταν σκόπιμη και όχι μια απόφαση υλοποίησης.

2.6.1 Λειτουργίες πλέγματος
Το πρόγραμμα θα πρέπει να παρέχει μια λειτουργία η οποία θα εμφανίζει ένα πλέγμα από οριζόντιες και κάθετες γραμμές στο φόντο του παραθύρου επεξεργασίας. Το πλέγμα πρέπει να είναι παθητικό, δηλαδή η στοίχιση των οντοτήτων θα αποτελεί ευθύνει του χρήστη.
Αιτιολογία: το πλέγμα βοηθάει το χρήστη να δημιουργήσει ένα τακτοποιημένο διάγραμμα με οντότητες σε κανονικές αποστάσεις. Αν και ένα ενεργητικό πλέγμα θα ήταν χρήσιμο, όπου οι οντότητες θα στοιχίζονταν αυτόματα επάνω στις γραμμές τους, η τοποθέτηση δεν θα ήταν ακριβής. Ο πλέον κατάλληλος για να αποφασίσει που θα τοποθετηθούν οι οντότητες είναι ο χρήστης.

ΕΙΚΟΝΑ 3.9 *Ορισμός λειτουργίας πλέγματος επεξεργασίας*

3.4 Απαιτήσεις συστήματος

Οι απαιτήσεις συστήματος είναι επεκτάσεις των απαιτήσεων χρήστη οι οποίες χρησιμοποιούνται από τους μηχανικούς λογισμικού ως σημείο εκκίνησης για το σχεδιασμό του συστήματος. Περιλαμβάνουν επιπλέον λεπτομέρειες και επεξηγούν πώς το σύστημα θα πρέπει να παρέχει τις απαιτήσεις χρήστη. Μπορούν να χρησιμοποιηθούν ως τμήμα της σύμβασης για την υλοποίηση του συστήματος, και επομένως πρέπει να αποτελούν μια πλήρη και συνεπή προδιαγραφή ολόκληρου του συστήματος.

Ιδανικά, οι απαιτήσεις του συστήματος θα πρέπει απλώς να περιγράφουν την εξωτερική συμπεριφορά του συστήματος και τους λειτουργικούς του περιορισμούς. Δεν πρέπει να ασχολούνται με το πώς θα σχεδιαστεί ή θα υλοποιηθεί το σύστημα. Ωστόσο, στο επίπεδο της λεπτομέρειας που είναι αναγκαίο για να καθοριστεί πλήρως ένα σύνθετο σύστημα λογισμικού, είναι πρακτικά αδύνατο να αποκλειστούν όλες οι σχεδιαστικές λεπτομέρειες. Αυτό οφείλεται σε πολλούς λόγους:

1. Ίσως χρειαστεί να σχεδιάσετε μια αρχική αρχιτεκτονική του συστήματος για να διευκολύνετε τη δόμηση της προδιαγραφής απαιτήσεων. Οι απαιτήσεις συστήματος οργανώνονται σύμφωνα με τα διάφορα υποσυστήματα που αποτελούν το όλο σύστημα. Ο αρχιτεκτονικός ορισμός έχει μεγάλη σημασία αν θέλετε να επαναχρησιμοποιήσετε συστατικά στοιχεία λογισμικού κατά την υλοποίηση του συστήματος.
2. Στις περισσότερες περιπτώσεις, τα συστήματα πρέπει να συνεργάζονται με άλλα συστήματα που ήδη υπάρχουν. Αυτό θέτει περιορισμούς στο σχεδιασμό, και οι περιορισμοί αυτοί επιβάλλουν απαιτήσεις στο νέο σύστημα.
3. Μπορεί να είναι αναγκαία η χρήση μιας συγκεκριμένης αρχιτεκτονικής για την ικανοποίηση μη λειτουργικών απαιτήσεων (όπως ο προγραμματισμός Ν εκδοχών για

την επίτευξη αξιοπιστίας). Κάποια εξωτερική εποπτική αρχή, που χρειάζεται να πιστοποιεί ότι το σύστημα είναι ασφαλές, θα μπορούσε να ορίσει ότι πρέπει να χρησιμοποιηθεί ένας ήδη πιστοποιούμενος αρχιτεκτονικός σχεδιασμός.

Για τη σύνταξη προδιαγραφών των απαιτήσεων συστήματος, συχνά χρησιμοποιείται φυσική γλώσσα, όπως ακριβώς γίνεται και για τις απαιτήσεις χρήστη. Επειδή όμως οι απαιτήσεις συστήματος είναι πιο λεπτομερείς από τις απαιτήσεις χρήστη, οι προδιαγραφές σε φυσική γλώσσα μπορεί να δημιουργήσουν σύγχυση και να φανούν δυσνόητες:

1. Η κατανόηση της φυσικής γλώσσας προϋποθέτει ότι οι αναγνώστες και οι συγγραφείς της προδιαγραφής χρησιμοποιούν τις ίδιες λέξεις για τις ίδιες έννοιες. Αυτό οδηγεί σε παρανοήσεις, λόγω της πολυσημίας της φυσικής γλώσσας. Ο Jackson (Jackson, 1995) δίνει ένα εξαιρετικό τέτοιο παράδειγμα, όταν περιγράφει τις πινακίδες στη βάση μιας κυλιόμενης σκάλας, οι οποίες ήταν της μορφής «Shoes must be worn» και «Dogs must be carried»[6]. Μπορείτε να διακρίνετε τις διφορούμενες ερμηνείες αυτών των φράσεων.
2. Μια προδιαγραφή απαιτήσεων σε φυσική γλώσσα είναι υπέρ του δέοντος ευέλικτη. Μπορείτε να λέτε το ίδιο πράγμα με τελείους διαφορετικούς τρόπους. Εξαρτάται από τον αναγνώστη να αποφασίσει πότε οι απαιτήσεις είναι οι ίδιες και πότε είναι διαφορετικές.
3. Οι απαιτήσεις σε φυσική γλώσσα δεν είναι εύκολο να οργανωθούν σε ξεχωριστές υπομονάδες. Ο εντοπισμός όλων των σχετικών απαιτήσεων μπορεί να είναι δύσκολος. Για να διαπιστώσετε τις συνέπειες μιας αλλαγής, ίσως χρειαστεί να εξετάσουμε κάθε απαίτηση, και όχι μόνο μια ομάδα σχετικών απαιτήσεων.

Εξαιτίας αυτών των προβλημάτων, οι προδιαγραφές απαιτήσεων που είναι γραμμένες σε φυσική γλώσσα είναι επιρρεπείς σε παρανοήσεις. Οι παρανοήσεις αυτές συχνά δε διαπιστώνονται παρά μόνο στις τελευταίες φάσεις της διαδικασίας παραγωγής λογισμικού, και τότε η επίλυσή τους μπορεί να έχει μεγάλο κόστος.

[6] Η πρώτη πρόταση σημαίνει «Φοράτε πάντοτε τα παπούτσια σας», ενώ η δεύτερη «Κρατήστε τα μικρά ζώα στα χέρια». Ωστόσο, θα μπορούσαν να ερμηνευτούν ως εξής: «Φορέστε τα παπούτσια σας» (σε περίπτωση που φτάσατε μέχρι εδώ ξυπόλητοι), και «Κρατάτε πάντοτε μικρά ζώα στα χέρια» (ως προϋπόθεση για να χρησιμοποιήσετε τη σκάλα).

Οι απαιτήσεις χρήστη πρέπει να γράφονται σε μια γλώσσα που να είναι κατανοητή από μη ειδικούς. Ωστόσο, οι απαιτήσεις συστήματος μπορούν να εκφράζονται με τη χρήση πιο εξειδικευμένων σημειογραφιών (Εικόνα 3.9). Τέτοιες σημειογραφίες είναι μια τυποποιημένη και δομημένη φυσική γλώσσα, γραφικά μοντέλα των απαιτήσεων όπως περιπτώσεις χρήσης, ή ακόμα και τυπικές μαθηματικές προδιαγραφές.

Σημειογραφία	Περιγραφή
Δομημένη φυσική γλώσσα	Η προσέγγιση αυτή βασίζεται στον ορισμό τυποποιημένων φορμών ή προτύπων εγγράφων για την έκφραση των προδιαγραφών των απαιτήσεων.
Γλώσσες περιγραφής σχεδιασμού	Η προσέγγιση αυτή χρησιμοποιεί μια γλώσσα σαν τις γλώσσες προγραμματισμού, αλλά με πιο αφηρημένες δυνατότητες, για τον καθορισμό των απαιτήσεων μέσω ενός λειτουργικού μοντέλου του συστήματος. Αυτή η προσέγγιση δεν χρησιμοποιείται ευρέως σήμερα, αν και μπορεί να είναι χρήσιμη για προδιαγραφές διασύνδεσης.
Σημειογραφίες γραφικών	Ο ορισμός των λειτουργικών απαιτήσεων για το σύστημα γίνεται με τη χρήση μιας γλώσσας γραφικών, η οποία συνοδεύεται από σχόλια κειμένου. Ένα πρώιμο παράδειγμα τέτοιας γλώσσας γραφικών ήταν η SADT (Ross, 1977) (Schoman και Ross,1977). Σήμερα χρησιμοποιούνται συνήθως περιγραφές περιπτώσεων χρήσης (Jacobsen[15],κ.α., 1993) και διαγράμματα ακολουθίας (Stevens και Pooley, 1999).
Μαθηματικές προδιαγραφές	Υπάρχουν σημειογραφίες που βασίζονται σε μαθηματικές έννοιες, όπως οι μηχανές πεπερασμένων καταστάσεων ή τα

	σύνολα. Τέτοιες μονοσήμαντες προδιαγραφές μειώνουν τις διαφωνίες μεταξύ πελάτη και εργολάβου σχετικά με τη λειτουργικότητα του συστήματος. Ωστόσο, οι περισσότεροι πελάτες δεν καταλαβαίνουν τις τυπικές προδιαγραφές και είναι απρόθυμοι να τις αποδεκτούν ως σύμβαση του συστήματος.

ΕΙΚΟΝΑ 3.10 *Σημειογραφίες για την προδιαγραφή των απαιτήσεων*

3.5 Προδιαγραφές σε δομημένη γλώσσα

Ένας τρόπος συγγραφής των απαιτήσεων συστήματος είναι η δομημένη φυσική γλώσσα, στην οποία η ελευθερία του συντάκτη των απαιτήσεων είναι περιορισμένων και όλες οι απαιτήσεις εκφράζονται με κάποιον τυποποιημένο τρόπο. Το πλεονέκτημα αυτής της προσέγγισης είναι ότι διατηρεί το μεγαλύτερο μέρος της εκφραστικότητας και της κατανοησιμότητας της φυσικής γλώσσας, ενώ ταυτόχρονα διασφαλίζει ότι επιβάλλεται κάποιος βαθμός ομοιομορφίας στις προδιαγραφές. Οι σημειογραφίες δομημένης γλώσσας περιορίζουν την ορολογία που μπορεί να χρησιμοποιηθεί, και χρησιμοποιούν πρότυπα έγγραφα για τον καθορισμό των απαιτήσεων συστήματος. Μπορεί να περιλαμβάνουν δομές ελέγχου που προέρχονται από τις γλώσσες προγραμματισμού, καθώς και γραφική σήμανση για το διαμερισμό των προδιαγραφών.

Ο Heninger[13] (Heninger, 1980) περιγράφει ένα πρώιμο έργο που χρησιμοποιούσε δομημένη φυσική γλώσσα για τον καθορισμό των απαιτήσεων συστήματος. Για την περιγραφή της εισόδου, της εξόδου, και των λειτουργιών ενός συστήματος λογισμικού αεροσκαφών, σχεδιάστηκαν ειδικές φόρμες, και οι απαιτήσεις συστήματος καθορίστηκαν με τη χρήση αυτών των φορμών.

Για να καθορίσετε τις απαιτήσεις συστήματος μα τη χρήση φορμών, πρέπει να ορίσετε μία ή περισσότερες τυποποιημένες φόρμες ή πρότυπα έγγραφα επάνω στα οποία θα εκφραστούν οι απαιτήσεις. Η προδιαγραφεί μπορεί να δομηθεί γύρω από τα αντικείμενα που χειρίζεται το σύστημα, τις λειτουργίες που πραγματοποιούνται από το σύστημα, ή από τα συμβάντα που επεξεργάζεται το σύστημα. Στην Εικόνα 3.10 παρουσιάζεται ένα τέτοιο παράδειγμα προδιαγραφής με φόρμα.

Η αντλία ινσουλίνης βασίζει τους υπολογισμούς της για τις ανάγκες του χρήστη σε ινσουλίνη στο ρυθμό μεταβολής των επιπέδων του σακχάρου στο αίμα. Ο υπολογισμός αυτών των ρυθμών μεταβολής γίνεται με βάση την τρέχουσα και τις προηγούμενες ενδείξεις.

Αντλία Ινσουλίνης/Λογισμικό ελέγχου/SRS/3.3.2	
Λειτουργία	Υπολογισμός δόσης ινσουλίνης: Ασφαλές επίπεδο σακχάρου
Περιγραφή	Υπολογίζει τη δόση της ινσουλίνης που θα χορηγηθεί όταν το τρέχον επίπεδο σακχάρου είναι μέσα στην ασφαλή ζώνη, μεταξύ 3 και 7 μονάδων
Είσοδοι	Τρέχουσα μέτρηση σακχάρου (r2), οι δύο προηγούμενες μετρήσεις (r0 και r1)
Προέλευση	Τρέχουσα μέτρηση σακχάρου από τον αισθητήρα. Οι άλλες μετρήσεις από τη μνήμη
Έξοδοι	CompDose- η δόση ινσουλίνη που θα χορηγηθεί
Προορισμός	Κύριος βρόχος ελέγχου
Ενέργεια: Η υπολογιζόμενη δόση CompDose είναι μηδέν αν το επίπεδο σακχάρου είναι σταθερό ή μειώνεται, ή αν το επίπεδο αυξάνεται αλλά ο ρυθμός αύξησης μειώνεται. Αν το επίπεδο αυξάνεται και ο ρυθμός αύξησης αυξάνεται, τότε το CompDose υπολογίζεται με διαίρεση δια 4 της διαφοράς μεταξύ του τρέχοντος επιπέδου σακχάρου και του προηγούμενου επιπέδου, και με στρογγυλοποίηση του αποτελέσματος. Αν το στρογγυλοποιημένο αποτέλεσμα είναι μηδέν, τότε το CompDose ορίζεται στην ελάχιστη δόση που μπορεί να χορηγηθεί.	
Απαιτεί	Δύο προηγούμενες μετρήσεις, ώστε να μπορεί να υπολογιστεί ο ρυθμός μεταβολής των επιπέδων του σακχάρου
Προσυνθήκη	Το δοχείο ινσουλίνης περιέχει τουλάχιστον τη μέγιστη επιτρεπόμενη δόση ινσουλίνης
Μετασυνθήκη	Η μέτρηση r0 από την r1 και η r1 από την r2
Παρενέργειες	Καμία

ΕΙΚΟΝΑ 3.11 *Προδιαγραφή των απαιτήσεων συστήματος με τη χρήση τυποποιημένης φόρμας*

Όταν χρησιμοποιείται μια τυποποιημένη φόρμα για τον καθορισμό των λειτουργικών απαιτήσεων, πρέπει να συμπεριλαμβάνονται οι ακόλουθες πληροφορίες:

1. Περιγραφή της λειτουργίας ή της οντότητας που προδιαγράφεται.
2. Προδιαγραφή των εισόδων και από πού προέρχονται.
3. Προδιαγραφή των εξόδων και για πού προορίζονται.
4. Αναφορά των άλλων οντοτήτων που χρησιμοποιούνται (το τμήμα «απαιτεί»).
5. Περιγραφή της ενέργειας που θα πραγματοποιείται.
6. Αν χρησιμοποιείται λειτουργική προσέγγιση, μια προσυνθήκη (pre-condition) που ορίζει τι πρέπει να ισχύει πριν την κλήση της λειτουργίας, και μια μετασυνθήκη (post-condition) που καθορίζει τι θα ισχύει μετά την κλήση της λειτουργίας.
7. Περιγραφή των παρενεργειών της λειτουργίας (αν υπάρχουν).

Η χρήση μορφοποιημένων προδιαγραφών μας απαλλάσσει από κάποια προβλήματα της προδιαγραφής σε φυσική γλώσσα. Μειώνεται η μεταβλητότητα της προδιαγραφής και οι απαιτήσεις οργανώνονται πιο αποτελεσματικά. Ωστόσο, η συγγραφή απαιτήσεων με μονοσήμαντο τρόπο εξακολουθεί να είναι δύσκολη, ιδιαίτερα όταν απαιτούνται πολύπλοκοι υπολογισμοί. Αυτό μπορείτε να το δείτε στην περιγραφή που φαίνεται στην Εικόνα 6.12, όπου δε γίνεται σαφές τι συμβαίνει αν δεν ικανοποιείται η προσυνθήκη.

Για την αντιμετώπιση αυτού του προβλήματος, μπορείτε να προσθέσετε επιπλέον πληροφορίες στις απαιτήσεις σε φυσική γλώσσα, χρησιμοποιώντας πίνακες ή γραφικά μοντέλα του συστήματος. Αυτά μπορούν να δείχνουν πώς γίνονται οι υπολογισμοί, πώς μεταβάλλεται η κατάσταση του συστήματος, πώς αλληλεπιδρούν οι χρήστες με το σύστημα, και πώς εκτελούνται οι ακολουθίες των ενεργειών.

Οι πίνακες είναι ιδιαίτερα χρήσιμοι όταν υπάρχουν πολλές πιθανότητες εναλλακτικές καταστάσεις και χρειάζεται να περιγράψετε τις ενέργειες που θα αναληφθούν για κάθε μία από αυτές. Η Εικόνα 3.11 είναι μια αναθεωρημένη περιγραφή του υπολογισμού της δόσης ινσουλίνης.

Συνθήκη	Ενέργεια
Το επίπεδο σακχάρου μειώνεται (r2<r1)	CompDose=0
Το επίπεδο σακχάρου μένει σταθερό (r2=r1)	CompDose=0
Το επίπεδο σακχάρου αυξάνεται και ο ρυθμός αύξησης μειώνεται ((r2-r1)<(r1-r0))	CompDose=0

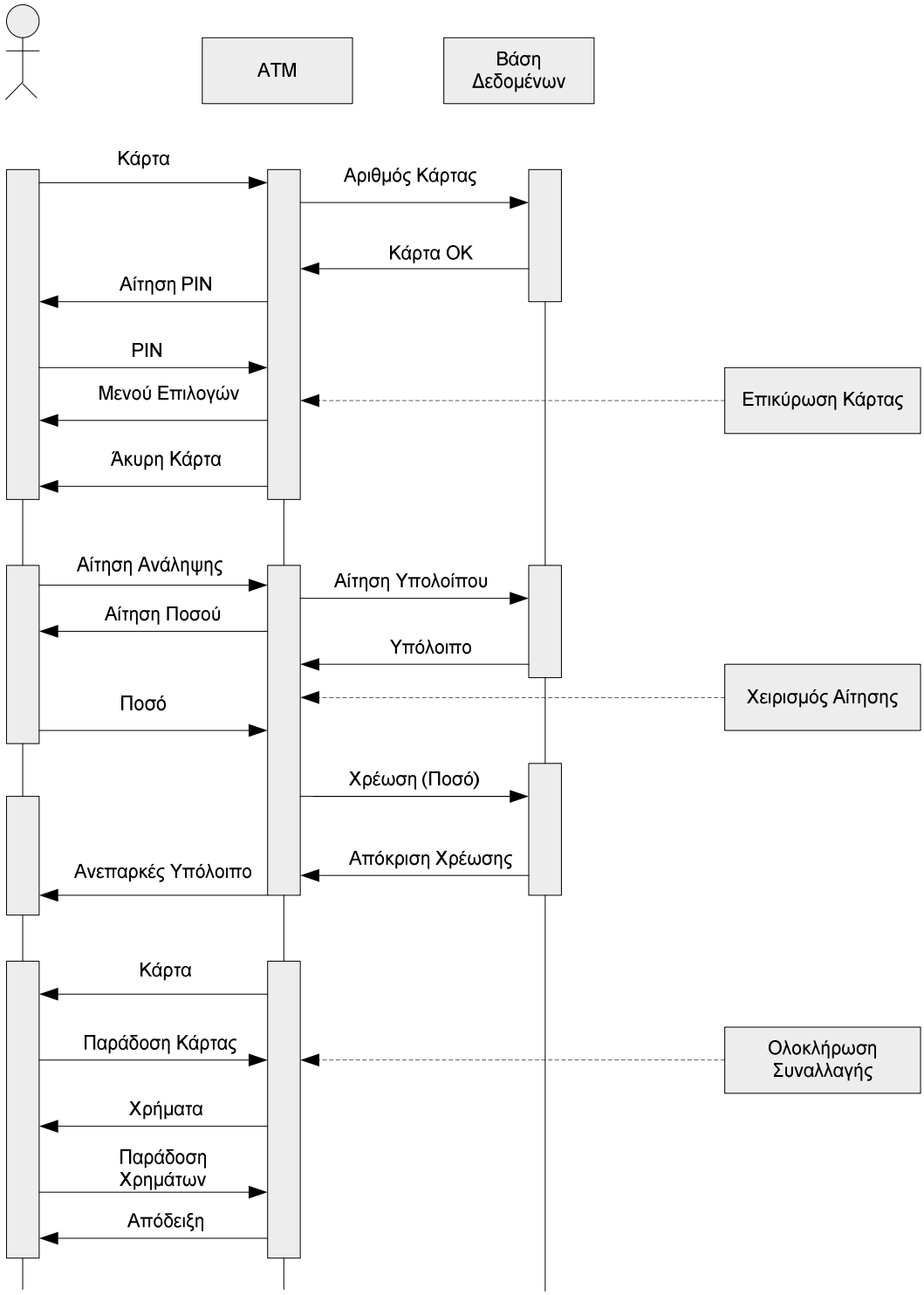
Το επίπεδο σακχάρου αυξάνεται και ο ρυθμός αύξησης μένει σταθερός ή αυξάνεται ((r2-r1)>(r1-r0))	CompDose=round ((r2-r1)/4) Αν μετά τη στρογγυλοποίηση το αποτέλεσμα είναι 0 τότε CompDose=MinimumDose

ΕΙΚΟΝΑ 3.12 *Προδιαγραφή υπολογισμού σε μορφή πίνακα*

Τα γραφικά μοντέλα είναι πιο χρήσιμα όταν χρειάζεται να δείξουμε πώς μεταβάλλεται η κατάσταση ή όταν χρειάζεται να περιγράψουμε μια ακολουθία ενεργειών. Η Εικόνα 3.12 επιδεικνύει την ακολουθία ενεργειών που λαμβάνει χώρα όταν ένας χρήστης επιθυμεί να κάνει ανάληψη χρημάτων από ένα αυτόματο μηχάνημα τραπεζικών συναλλαγών.

Για να δούμε τη σειρά με την οποία πραγματοποιούνται οι ενέργειες, θα πρέπει να διαβάσουμε το διάγραμμα ακολουθίας από την κορυφή προς το τέλος. Στην Εικόνα 3.12, υπάρχουν τρεις βασικές υπο-ακολουθίες:

1. Επικύρωση κάρτας. Η κάρτα του χρήστη επικυρώνεται με έλεγχο του αριθμού της κάρτας και του αριθμού PIN του χρήστη.



ΕΙΚΟΝΑ 3.13 *Διάγραμμα ακολουθίας για ανάληψη χρημάτων από ATM*

2. Χειρισμός αίτησης. Το σύστημα χειρίζεται την αίτηση του χρήστη. Για να γίνει ανάληψη, πρέπει να ερωτηθεί η βάση δεδομένων ώστε να ελεγχθεί το υπόλοιπο του λογαριασμού του χρήστη και να χρεωθεί με το ποσό της ανάληψης. Παρατηρήστε εδώ την εξαίρεση (στερεότυπο «exception») όταν ο αιτών δεν έχει αρκετά χρήματα στο λογαριασμό του.

3. Ολοκλήρωση συναλλαγής. Η κάρτα του χρήστη επιστρέφεται και, όταν αφαιρεθεί από το μηχάνημα, παραδίδονται τα χρήματα και η απόδειξη

3.6 Προδιαγραφές διασύνδεσης

Σχεδόν όλα τα συστήματα λογισμικού πρέπει να συνεργάζονται με υφιστάμενα συστήματα που έχουν ήδη υλοποιηθεί και εγκατασταθεί σε κάποιο περιβάλλον. Σε μια τέτοια περίπτωση, οι διασυνδέσεις των υφισταμένων συστημάτων πρέπει να προδιαγραφούν με ακρίβεια. Οι προδιαγραφές αυτές είναι απαραίτητο να οριστούν νωρίς κατά τη διαδικασία και να συμπεριληφθούν (ίσως με τη μορφή παραρτήματος) στο έγγραφο των απαιτήσεων.

Υπάρχουν τρεις τύποι διασυνδέσεων που ίσως χρειαστεί να οριστούν:

1. Διαδικασιακές διασυνδέσεις, όπου τα υφιστάμενα προγράμματα ή υποσυστήματα προσφέρουν ένα φάσμα υπηρεσιών οι οποίες προσπελάζονται με την κλήση διαδικασιών της διασύνδεσης. Οι διασυνδέσεις αυτές ονομάζονται και διασυνδέσεις προγραμματισμού εφαρμογών (Application Programming Interfaces, API).
2. Δομές δεδομένων που μεταβιβάζονται από το ένα υποσύστημα στο άλλο. Η καλύτερη σημειογραφία για αυτόν τον τύπο περιγραφής είναι τα γραφικά μοντέλα δεδομένων. Αν χρειαστεί, αυτές οι περιγραφές μπορούν να χρησιμεύσουν ως προέλευση για την αυτόματη παραγωγή περιγραφών προγραμμάτων σε Java ή C++.
3. Αναπαραστάσεις δεδομένων (όπως η διάταξη των bit) που έχουν καθιερωθεί για κάποιο υφιστάμενο υποσύστημα. Οι διασυνδέσεις αυτές συνηθίζονται περισσότερο σε ενσωματωμένα συστήματα πραγματικού χρόνου. Ορισμένες γλώσσες προγραμματισμού, όπως η Ada, (όχι όμως η Java), υποστηρίζουν αυτό το επίπεδο προδιαγραφών. Ο καλύτερος όμως τρόπος περιγραφής τους είναι μάλλον η χρήση ενός διαγράμματος δομής με σχόλια που επεξηγούν τη λειτουργία της κάθε ομάδας bit.

Οι τυπικές σημειογραφίες, επιτρέπουν το σαφή καθορισμό διασυνδέσεων, αλλά η εξειδικευμένη φύση τους σημαίνει ότι δεν είναι κατανοητές χωρίς ειδική εκπαίδευση. Σπανίως χρησιμοποιούνται στην πράξη για προδιαγραφές διασυνδέσεων αν και, κατά την άποψή μου, είναι ιδανικές γι’ αυτόν το σκοπό. Για την περιγραφή της σύνταξης της διασύνδεσης μπορεί να χρησιμοποιηθεί μια γλώσσα προγραμματισμού, όπως η Java. Αυτό

όμως πρέπει να συνοδεύεται από περαιτέρω περιγραφές, που να εξηγούν τη σημασιολογία της κάθε λειτουργίας που ορίζεται.

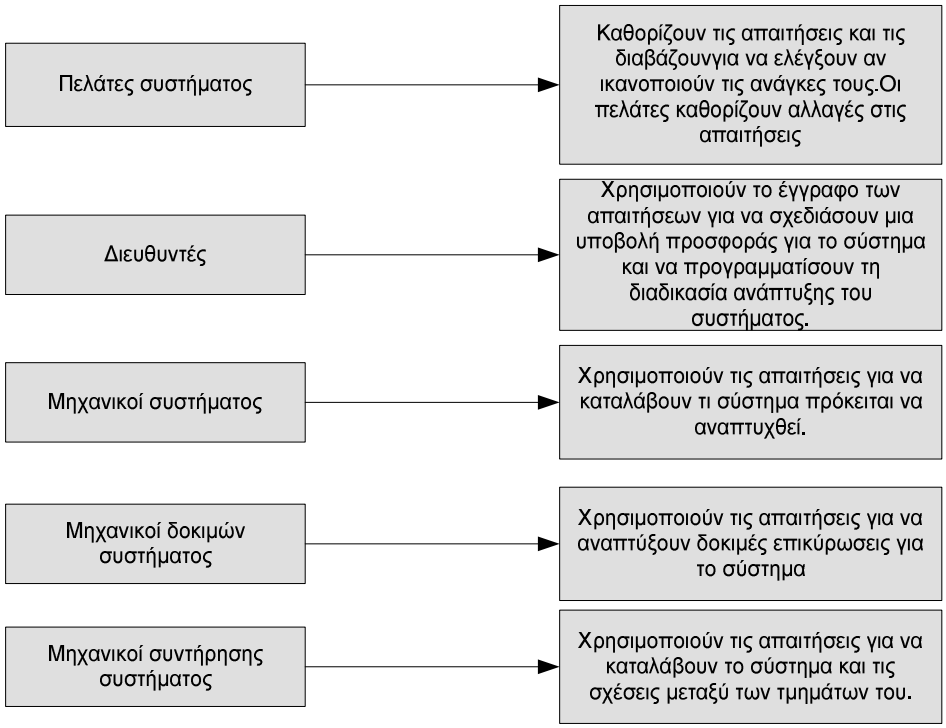
```
Interfase printserver {
// Ορίζει έναν αφηρημένο διακομιστή εκτυπώσεων
// Απαιτεί: Inerfase printer, Interfase printDoc
// Παρέχει: Initialize, print, displayPrintQueue, cancelprintjob, switch printer

void initialize (Printer p);
void print (Printer p, PrintDoc d);
void displayPrintQueue (Print p);
Void cancelPrintJob (Printer p, PrintDoc d);
Void switchPrinter (Printer p1, Printer p2, PrintDoc d);
}//PrintServer
```

ΕΙΚΟΝΑ 3.14 *Η περιγραφή μιας διασύνδεσης διακομιστή εκτυπώσεων σε java PDL*

Η Εικόνα 3.13 είναι ένα παράδειγμα ορισμού διαδικασιακής διασύνδεσης σε Java. Στη συγκεκριμένη περίπτωση, πρόκειται για τη διαδικασιακή διασύνδεση που παρέχεται από ένα διακομιστή εκτυπώσεων, ο οποίος διαχειρίζεται μια ουρά απαιτήσεων για την εκτύπωση αρχείων σε διάφορους εκτυπωτές. Οι χρήστες μπορούν να εξετάζουν την ουρά εκτυπώσεων που αντιστοιχεί σε κάποιον εκτυπωτή και ενδεχομένως να αφαιρούν τις δικές τους εργασίες εκτύπωσης από την ουρά. Μπορούν επίσης να μεταφέρουν εργασίες από τον έναν εκτυπωτή στον άλλο. Οι λειτουργικές δυνατότητες της διασύνδεσης μπορούν να οριστούν με τη χρήση κάποιας δομημένης φυσικής γλώσσας ή με πινακοποιημένες περιγραφές.

3.7 Το έγγραφο των απαιτήσεων λογισμικού

Το έγγραφο των απαιτήσεων λογισμικού, που μερικές φορές ονομάζεται και προδιαγραφή απαιτήσεων λογισμικού (Software requirements specification, SRS), είναι η επίσημη δήλωση του τι θα πρέπει να υλοποιήσουν οι κατασκευαστές του συστήματος. Θα πρέπει να περιλαμβάνει τόσο τις απαιτήσεις χρήστη για το σύστημα όσο και μια λεπτομερή προδιαγραφή των απαιτήσεων του συστήματος. Σε μερικές περιπτώσεις, οι απαιτήσεις χρήστη και οι απαιτήσεις συστήματος μπορούν να ενοποιηθούν σε μια μόνο περιγραφή. Σε άλλες περιπτώσεις, οι απαιτήσεις χρήστη ορίζονται σε μια εισαγωγική ενότητα της προδιαγραφής των απαιτήσεων του συστήματος. Αν υπάρχει μεγάλος αριθμός απαιτήσεων, οι λεπτομερείς απαιτήσεις συστήματος μπορεί να παρουσιάζονται σε ξεχωριστό έγγραφο.



ΕΙΚΟΝΑ 3.15 *Χρήστες ενός εγγράφου απαιτήσεων*

Το έγγραφο των απαιτήσεων απευθύνεται σε ένα ετερογενές σύνολο χρηστών, που ξεκινάει από τη διεύθυνση της εταιρείας που πληρώνει για το σύστημα και φτάνει μέχρι τους μηχανικούς που είναι υπεύθυνοι για την ανάπτυξη του λογισμικού. Η Εικόνα 3.14, που παρουσιάζει τους πιθανούς χρήστες του εγγράφου και τον τρόπο με τον οποίο το χρησιμοποιούν.

Η ποικιλία των πιθανών χρηστών σημαίνει ότι το έγγραφο των απαιτήσεων πρέπει να αποτελεί ένα συμβιβασμό ανάμεσα στην παρουσίαση των απαιτήσεων στον πελάτη, τον ορισμό των απαιτήσεων με επακριβείς λεπτομέρειες για τους υπεύθυνους ανάπτυξης και δοκιμών, και τη συμπερίληψη πληροφοριών για την πιθανή εξέλιξη του συστήματος. Οι πληροφορίες για αναμενόμενες αλλαγές μπορούν να βοηθήσουν τους σχεδιαστές του συστήματος να αποφύγουν περιοριστικές αποφάσεις όσον αφορά το σχεδιασμό, καθώς και τους μηχανικούς συντήρησης του συστήματος οι οποίοι θα πρέπει να προσαρμόσουν το σύστημα σε νέες απαιτήσεις.

Το επίπεδο λεπτομερειών που πρέπει να συμπεριλαμβάνεται σε ένα έγγραφο απαιτήσεων εξαρτάται από τον τύπο του συστήματος που αναπτύσσεται και τη διαδικασία ανάπτυξης που χρησιμοποιείται. Όταν το σύστημα πρόκειται να αναπτυχθεί από κάποιον εξωτερικό εργολάβο, οι κρίσιμες προδιαγραφές του θα πρέπει να είναι ακριβείς και εξαιρετικά λεπτομερείς. Όταν υπάρχει περισσότερη ευελιξία στις απαιτήσεις και όταν

χρησιμοποιείται μια επαναληπτική διαδικασία ανάπτυξης στο εσωτερικό της εταιρείας, το έγγραφο των απαιτήσεων μπορεί να είναι πολύ λιγότερο λεπτομερές και οι τυχόν ασάφειες να επιλύονται κατά το στάδιο της ανάπτυξης.

Μια σειρά από μεγάλους οργανισμούς, όπως το Υπουργείο Άμυνας των ΗΠΑ και το IEEE, έχουν ορίσει πρότυπα για τα έγγραφα απαιτήσεων. Ο Davis[9] (Davis, 1993) εξετάζει κάποια από αυτά τα πρότυπα και συγκρίνει τα περιεχόμενά τους. Το πιο διαδεδομένο πρότυπο είναι το IEEE/ANSI 830-1998 (IEEE, 1998). Αυτό το πρότυπο του IEEE προτείνει την ακόλουθη δομή για τα έγγραφα απαιτήσεων:

1. Εισαγωγή
    1.1 Σκοπός του εγγράφου απαιτήσεων
    1.2 Αντικείμενο του προϊόντος
    1.3 Ορισμοί, ακρωνύμια, και συντμήσεις
    1.4 Αναφορές
    1.5 Επισκόπηση του υπόλοιπου εγγράφου
2. Γενική περιγραφή
    2.1 Θεώρηση του προϊόντος
    2.2 Λειτουργίες του προϊόντος
    2.3 Χαρακτηριστικά των χρηστών
    2.4 Γενικοί περιορισμοί
    2.5 Παραδοχές και εξαρτήσεις
3. Ειδικές απαιτήσεις. Το τμήμα αυτό καλύπτει τις λειτουργικές και μη λειτουργικές απαιτήσεις και τις απαιτήσεις διασύνδεσης. Αποτελεί προφανώς τι πιο ουσιώδες μέρος του εγγράφου αλλά, λόγω της μεγάλης ποικιλίας που παρουσιάζουν οι πρακτικές των οργανισμών, δεν είναι σκόπιμο να οριστεί μια τυποποιημένη δομή για αυτή την ενότητα. Οι απαιτήσεις μπορεί να τεκμηριώνουν εξωτερικές διασυνδέσεις, να περιγράφουν τη λειτουργικότητα και την απόδοση του συστήματος, να καθορίζουν λογικές απαιτήσεις βάσεων δεδομένων, να σχεδιάζουν περιορισμούς, ανακύπτουσες ιδιότητες του συστήματος, και χαρακτηριστικά ποιότητας.
4. Παραρτήματα
5. Ευρετήριο

Αν και το πρότυπο του IEEE δεν είναι ιδανικό, περιέχει πολλές καλές συμβουλές για το πώς να γράφετε απαιτήσεις και να αποφεύγετε προβλήματα. Είναι πολύ γενικό για να αποτελέσει από μόνο του πρότυπο για κάποιον οργανισμό. Αποτελεί απλώς ένα πλαίσιο που μπορεί να τροποποιηθεί και να προσαρμοστεί έτσι ώστε να καθορίσει ένα πρότυπο

προσανατολισμένο στις ανάγκες ενός συγκεκριμένου οργανισμού. Η Εικόνα 3.15 επιδεικνύει μια πιθανή οργάνωση ενός εγγράφου απαιτήσεων που βασίζετε στο πρότυπο του IEEE. Ωστόσο, όπως επισήμανε ο Heninger[13] (Heninger, 1980) επεκτείνουμε το πρότυπο ώστε να συμπεριλάβουμε πληροφορίες σχετικά με την προβλεπόμενη εξέλιξη του συστήματος, όπου βοηθάμε τους συντηρητές και τους σχεδιαστές του συστήματος να συμπεριλάβουν υποστήριξη για μελλοντικές δυνατότητες του συστήματος.

Κεφάλαιο	Περιγραφή
Πρόλογος	Πρέπει να ορίζει τους αναμενόμενους αναγνώστες του εγγράφου και να περιγράφει το ιστορικό εκδόσεών του, καθώς και μια αιτιολογία για τη δημιουργία νέας έκδοσης και μια περίληψη των αλλαγών που έγιναν σε κάθε έκδοση.
Εισαγωγή	Πρέπει να περιγράφει την ανάγκη για το σύστημα, να αναφέρει συνοπτικά τις λειτουργίες του, και να εξηγεί πώς θα συνεργάζεται με άλλα συστήματα. Θα πρέπει επίσης να περιγράφει ποια είναι η θέση του συστήματος μέσα στους επιχειρηματικούς ή στρατηγικούς στόχους του οργανισμού που χρειάζεται το λογισμικό.
Γλωσσάρι	Πρέπει να γνωρίζει τους τεχνικούς όρους που χρησιμοποιούνται στο έγγραφο. Δεν είναι σωστό να γίνονται παραδοχές σχετικά με την πείρα ή τις γνώσεις του αναγνώστη.
Ορισμός απαιτήσεων χρήστη	Σε αυτή την ενότητα πρέπει να περιγράφονται οι υπηρεσίες που παρέχονται στο χρήστη και οι μη λειτουργικές απαιτήσεις του συστήματος. Η περιγραφή αυτή μπορεί να χρησιμοποιεί φυσική γλώσσα, διαγράμματα, ή άλλες σημειογραφίες που είναι κατανοητές στον πελάτη. Θα πρέπει να καθορίζονται τα πρότυπα προϊόντος και διαδικασιών που θα ακολουθηθούν.
Αρχιτεκτονική συστήματος	Το κεφάλαιο αυτό θα πρέπει να παρουσιάζει μια επισκόπηση υψηλού επιπέδου της αναμενόμενης αρχιτεκτονικής του συστήματος, η οποία να δείχνει την κατανομή των λειτουργιών στις υπομονάδες του συστήματος. Τα αρχιτεκτονικά στοιχεία που επαναχρησιμοποιούνται θα πρέπει να επισημαίνονται.
Προδιαγραφή απαιτήσεων	Πρέπει να περιγράφει τις λειτουργικές και μη λειτουργικές απαιτήσεις με περισσότερες λεπτομέρειες. Αν χρειάζεται,

συστήματος	μπορούν να προστεθούν περισσότερες λεπτομέρειες στις μη λειτουργικές απαιτήσεις, π.χ. να οριστούν διασυνδέσεις με άλλα συστήματα.
Μοντέλα συστήματος	Πρέπει να περιγράφει ένα ή περισσότερα μοντέλα συστήματος που να δείχνουν τις σχέσεις μεταξύ των στοιχείων του συστήματος και μεταξύ του συστήματος και του περιβάλλοντός του. Μπορεί να είναι μοντέλα αντικειμένων, μοντέλα ροής δεδομένων, και σημασιολογικά μοντέλα δεδομένων.
Εξέλιξη συστήματος	Θα πρέπει να περιγράφει τις θεμελιώδεις παραδοχές στις οποίες βασίζεται το σύστημα, και τις αναμενόμενες αλλαγές λόγω εξέλιξης του υλικού, λόγω μεταβολής των αναγκών των χρηστών κ.λπ.
Παραρτήματα	Παρέχουν λεπτομερείς, εξειδικευμένες πληροφορίες που σχετίζονται με την εφαρμογή που αναπτύσσεται. Παραδείγματα παραρτημάτων που μπορούν να συμπεριληφθούν είναι περιγραφές για το υλικό και τις βάσεις δεδομένων. Οι απαιτήσεις υλικού ορίζουν την ελάχιστη και την βέλτιστη σύνθεση για το σύστημα. Οι απαιτήσεις βάσεων δεδομένων ορίζουν τη λογική οργάνωση των δεδομένων που χρησιμοποιούνται από το σύστημα και τις σχέσεις μεταξύ των δεδομένων.
Ευρετήριο	Το έγγραφο μπορεί να περιλαμβάνει διάφορα ευρετήρια. Εκτός από το κανονικό αλφαβητικό ευρετήριο, μπορεί να υπάρχει ευρετήριο διαγραμμάτων, ευρετήριο λειτουργιών, κ.λπ.

ΕΙΚΟΝΑ 3.16 *Η δομή ενός εγγράφου απαιτήσεων*

Φυσικά, οι πληροφορίες που περιλαμβάνονται σε ένα έγγραφο απαιτήσεων πρέπει να βασίζονται στον τύπο του λογισμικού που αναπτύσσεται και στην προσέγγιση που χρησιμοποιείται για την ανάπτυξη. Αν, για παράδειγμα, υιοθετεί μια εξελικτική προσέγγιση για ένα προϊόν λογισμικού, το έγγραφο των απαιτήσεων θα πρέπει να παραλείψει πολλά από τα λεπτομερή κεφάλαια που προτάθηκαν παραπάνω. Το επίκεντρο θα είναι ο ορισμός των απαιτήσεων χρήστη και των υψηλού επιπέδου, μη λειτουργικών απαιτήσεων συστήματος. Σε τέτοιες περιπτώσεις, οι σχεδιαστές και οι προγραμματιστές χρησιμοποιούν την κρίση τους για να αποφασίσουν πώς θα ικανοποιηθεί το γενικό περίγραμμα των απαιτήσεων χρήστη για το σύστημα.

Αντίθετα, όταν το λογισμικό είναι μέρος ενός μεγάλου έργου κατασκευής που περιλαμβάνει την αλληλεπίδραση συστημάτων υλικού και λογισμικού, συχνά είναι απαραίτητο να ορίσουμε τις απαιτήσεις με λεπτομέρειες. Αυτό σημαίνει ότι τα έγγραφα των απαιτήσεων μάλλον θα είναι πολύ μεγάλα και θα πρέπει να περιλαμβάνουν τα περισσότερα κεφάλαια της Εικόνας 3.15, αν όχι όλα. Στα μεγάλα έγγραφα, είναι ιδιαίτερα σημαντικό να περιλαμβάνονται εκτενής πίνακας περιεχομένων και ευρετήριο, ώστε να μπορούν οι αναγνώστες να εντοπίζουν τις πληροφορίες που χρειάζονται.

Τα έγγραφα απαιτήσεων είναι απαραίτητα όταν το σύστημα λογισμικού αναπτύσσεται από κάποιον εξωτερικό εργολάβο. Από την άλλη, σύμφωνα με τις λεγόμενες ευέλικτες μεθόδους ανάπτυξης, οι απαιτήσεις αλλάζουν τόσο γρήγορα που ένα έγγραφο απαιτήσεων είναι ήδη ξεπερασμένο μόλις γραφτεί, οπότε η προσπάθεια σε μεγάλο βαθμό πάει χαμένη. Αντί για το τυπικό έγγραφο, ορισμένες προσεγγίσεις όπως ο ακραίος προγραμματισμός (Beck[2], 2000) προτείνουν να συλλέγονται οι απαιτήσεις χρήστη βηματικά και να γράφονται σε κάρτες. Έπειτα ο χρήστης θέτει σε σειρά προτεραιότητας τις απαιτήσεις που θα υλοποιηθούν στο επόμενο βήμα της ανάπτυξης του συστήματος.

Για επιχειρηματικά συστήματα με ασταθείς απαιτήσεις, νομίζω ότι αυτή η προσέγγιση είναι καλή. Θα αντιτείνω όμως ότι η σύνταξη ενός σύντομου εγγράφου υποστήριξης που ορίζει τις επιχειρηματικής απαιτήσεις και τις απαιτήσεις φερεγγυότητας για το σύστημα, εξακολουθεί να έχει τη χρησιμότητά της. Όταν εστιάζουμε την προσοχή μας στις λειτουργικές απαιτήσεις για την επόμενη έκδοση του συστήματος, είναι εύκολο να ξεχάσουμε τις απαιτήσεις που ισχύουν για το σύστημα ως σύνολο.

3.8 Μηχανική Απαιτήσεων

Στην ενότητα αυτή θα παρουσιαστεί η γενική διαδικασία ανάλυσης και προσδιορισμού απαιτήσεων από το λογισμικό, η οποία αναφέρεται και ως *μμηχανική απαιτήσεων* (requirements engineering)

Το πλήθος και η πολυπλοκότητα που χαρακτηρίζει πολλές από τις απαιτήσεις από το λογισμικό, ο σαφής προσδιορισμός και η παρακολούθηση των συσχετίσεων μεταξύ αυτών, καθώς και με απαιτήσεις από το σύστημα, τα διαφορετικά επίπεδα λεπτομέρειας στην περιγραφή τους είναι μερικά από τα σημαντικότερα προβλήματα που συναντά κανείς όταν καλείται να αντιμετωπίσει ένα πρόβλημα προσδιορισμού απαιτήσεων από το λογισμικό. Είναι ευνόητο ότι η επιτυχής αντιμετώπιση ενός τέτοιου προβλήματος δεν μπορεί να γίνει παρά μόνο με πειθαρχία, ακολουθώντας συγκεκριμένα βήματα και καταγράφοντας τα αποτελέσματα που παράγονται σε κάθε βήμα.

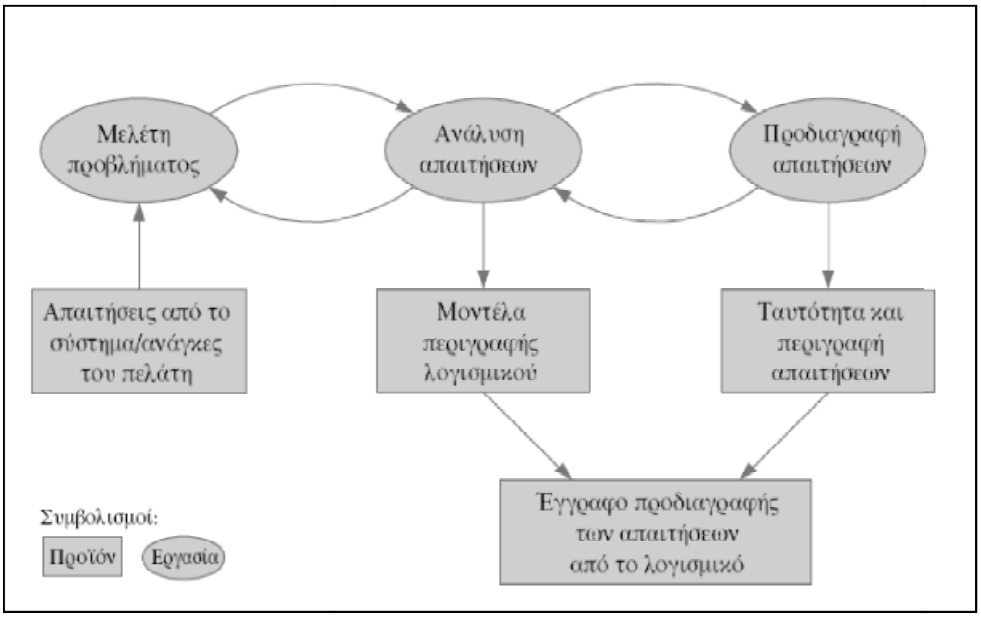
Υπάρχουν διάφορες προσεγγίσεις στο πρόβλημα αυτό, καθεμιά εκ των οποίων προτείνει τα δικά της βήματα ή τις δικές της λεπτομέρειες εκτέλεσης κάθε βήματος. Στην πράξη δεν υπάρχει μια «καλύτερη» από άλλες λύση και κάθε κατασκευαστής λογισμικού ακολουθεί τελικά μια δική του εκδοχή, που περιέχει στοιχεία μίας ή και περισσότερων προσεγγίσεων. Ο τρόπος επίλυσης του προβλήματος προσδιορισμού των απαιτήσεων που προτείνει καθεμία από αυτές αναφέρεται ως *μηχανική απαιτήσεων* (requirements engineering). Θα προσεγγίσουμε το θέμα της μηχανικής απαιτήσεων ως μια γενική αλληλουχία ενεργειών που πρέπει να γίνονται κατά τον προσδιορισμό των απαιτήσεων από το λογισμικό. Κατά τις ενέργειες αυτές παράγονται ορισμένα προϊόντα με τη μορφή εγγράφων και διαγραμμάτων.

Η προσέγγιση που θα ακολουθηθεί στο παρόν βιβλίο αναφέρεται ως *δομημένη ανάλυση* (structured analysis) και έχει γνωρίσει σημαντική διάδοση στο παρελθόν, όντας ένας αποτελεσματικός και πειθαρχημένος τρόπος για την κατασκευή λογισμικού. Τα τελευταία χρόνια έχει κερδίσει σημαντικό έδαφος η *αντικειμενοστρεφής ανάλυση και σχεδίαση* (object.oriented analysis and design), η οποία μπορεί να θεωρηθεί ως υπερσύνολο της δομημένης προσέγγισης.

3.8.1 Βήματα στον προσδιορισμό απαιτήσεων

Η πρώτη πηγή για τον καθορισμό των απαιτήσεων από μια εφαρμογή λογισμικού είναι, ασφαλώς, ο πελάτης, ο οποίος περιγράφει στον κατασκευαστή τις εργασίες που θεωρεί απαραίτητο να εκτελούνται από το λογισμικό. Η περιγραφή αυτή συνήθως γίνεται με τη μορφή μιας έκθεσης, η οποία ενίοτε δεν είναι πλήρης και περιέχει ασάφειες και διφορούμενα. Η έκθεση αυτή αποτελεί το πρώτο υλικό που έχει στη διάθεσή του ο κατασκευαστής προκειμένου να καθορίσει όλα τα στοιχεία που θα του επιτρέψουν να κατασκευάσει λογισμικό που ικανοποιεί τον πελάτη, καθώς και να καθορίσει το κόστος και να εκτιμήσει το χρόνο που θα απαιτηθεί. Εκτός από τις εργασίες που θα πρέπει να εκτελεί το λογισμικό, πρέπει να προσδιοριστούν και άλλα χαρακτηριστικά του, όπως, για παράδειγμα, το περιβάλλον λειτουργίας, ο τρόπος χρήσης και οι επιδόσεις.

Από την άλλη πλευρά, ο τρόπος με τον οποίο αντιλαμβάνεται ο πελάτης τις εργασίες που εκτελεί το λογισμικό δε βρίσκεται πάντα σε αντιστοιχία με τον τρόπο με τον οποίο αυτές μπορούν να ενσωματωθούν σε μια εφαρμογή λογισμικού. Σε πολλές περιπτώσεις αυτό που ο πελάτης αντιλαμβάνεται ως μια και μοναδική λειτουργία απαιτείται να αναλυθεί σε περισσότερες προκειμένου να υλοποιηθεί στο λογισμικό. Από τα παραπάνω συνάγεται ότι μια πρόσφορη για τον καθορισμό των απαιτήσεων από το λογισμικό διαδικασία περιγράφεται ως μια ακολουθία βημάτων, σε καθένα από τα οποία παράγεται μια και ολοένα και λεπτομερέστερη εκδοχή των απαιτήσεων από το λογισμικό. Τελικό προϊόν της διαδικασίας αυτής είναι το έγγραφο «Προδιαγραφές των απαιτήσεων από το λογισμικό», καθώς και ένα σύνολο από διαγράμματα τα οποία το συνοδεύουν. Η γενική μορφή της διαδικασίας φαίνεται στο Σχήμα 3.16.



ΣΧΗΜΑ 3.17 *Μηχανική απαιτήσεων: Η γενική μορφή της διαδικασίας προσδιορισμού των απαιτήσεων από το λογισμικό*

Η διαδικασία τροφοδοτείται με το έγγραφο των απαιτήσεων από το σύστημα ή, αν αυτό δεν είναι διαθέσιμο, με μια έκθεση αναγκών του πελάτη. Το πρώτο βήμα είναι η **μελέτη του εγγράφου απαιτήσεων από το σύστημα ή/και των αναγκών του πελάτη**, η οποία στοχεύει στην αρχική κατανόηση του πεδίου του προβλήματος για την επίλυση του οποίου καλείται να χρησιμοποιηθεί το λογισμικό που κατασκευάζεται. Η μελέτη αυτή συνήθως πραγματοποιείται από διοικητική και οργανωτική σκοπιά, προκειμένου να εκτιμηθεί η βιωσιμότητα, τα ρίσκα, ο προϋπολογισμός, το χρονοδιάγραμμα και άλλες διαχειριστικές παράμετροι της ανάπτυξης λογισμικού.

Ακολουθεί η **ανάλυση των απαιτήσεων**, η οποία στοχεύει στη δημιουργία μοντέλων που περιγράφουν διαφορετικές πλευρές του λογισμικού. Τα μοντέλα αυτά παριστάνονται με τη βοήθεια διαγραμμάτων *ροής δεδομένων*, *οντοτήτων*, *συσχετίσεων* και *μετάβασης καταστάσεων*, καθώς και με χρήση ενός πίνακα *λεξικού δεδομένων*, αναλυτική αναφορά στα οποία θα γίνει στη συνέχεια του κεφαλαίου.

Η **διάκριση και προδιαγραφή κάθε συγκεκριμένης απαίτησης** από το λογισμικό είναι το επόμενο βήμα, κατά το οποίο συμπληρώνεται το έγγραφο «Προδιαγραφές των απαιτήσεων από το λογισμικό», το οποίο είναι το επιθυμητό αποτέλεσμα της διαδικασίας. Το έγγραφο αυτό περιγράφει με λεπτομέρεια τις απαιτήσεις από το λογισμικό, τις ταξινομεί και τις ιεραρχεί και βρίσκεται σε πλήρη συμφωνία με τα διαγράμματα που έχουν παραχθεί στο προηγούμενο βήμα. Όπως φαίνεται στο Σχήμα 3.3, κατά τη διαδικασία αυτή μπορεί να πραγματοποιούνται πισωγυρίσματα, όταν κάτι τέτοιο κρίνεται απαραίτητο.

3.9 Ανάλυση απαιτήσεων

Κατά την *ανάλυση των απαιτήσεων* εντοπίζονται για πρώτη φορά οι απαιτήσεις από το λογισμικό και ακολουθούν έναν κύκλο ταξινόμησης, ιεράρχησης και επαλήθευσης, όπως φαίνεται στο Σχήμα 3.17. Αποτέλεσμα των εργασιών που εκτελούνται στη φάση αυτή είναι ένα σύνολο απαιτήσεων από το λογισμικό οι οποίες περιγράφονται με μορφή διαγραμμάτων. Η περιγραφή αυτή αποτελεί την είσοδο στο επόμενο βήμα, αυτό της διάκρισης και προδιαγραφής των απαιτήσεων από το λογισμικό.

ΣΧΗΜΑ 3.18 *Επιμέρους βήματα που εκτελούνται κατά τη φάση της ανάλυσης απαιτήσεων από το λογισμικό*

Κατά την **κατανόηση του προβλήματος**, ο αναλυτής υπεισέρχεται ο ίδιος στην ουσία του προβλήματος όσο περισσότερο γίνεται. Η έκθεση των αναγκών του πελάτη δεν είναι συνήθως επαρκής για την αντίληψη της ουσίας του αντικειμένου σχετικά με το οποίο αναπτύσσεται λογισμικό. Χωρίς την επίτευξη από πλευράς του κατασκευαστή κάποιας εξοικείωσης με την ουσία του προβλήματος στην επίλυση του οποίου συμβάλλει το λογισμικό που κατασκευάζει, δεν είναι δυνατή η επιτυχής ανάπτυξη λογισμικού που αφορά την επίλυση αυτού. Η εξοικείωση αυτή θα πρέπει να γίνει στο αρχικό στάδιο ανάπτυξης και θα σημαδέψει ακολούθως τη διαδικασία ανάπτυξης του λογισμικού. Πρόκειται για μια συναρπαστική, ιδιαίτερα δημιουργική και συνάμα δύσκολη εργασία, που καθιστά το επάγγελμα του μηχανικού λογισμικού από τα πιο ενδιαφέροντα και λιγότερο μονότονα.

Ακολούθως **συλλέγονται οι απαιτήσεις** των εμπλεκομένων με το λογισμικό και γίνεται μια αρχική καταγραφή τους σε λίστα. Η συλλογή αυτή γίνεται με τη βοήθεια συνεντεύξεων, ερωτηματολογίων, συζητήσεων με ειδικούς, ή με άλλους, κατά περίπτωση, πρόσφορους τρόπους. Η λίστα που δημιουργείται αρχικά περιέχει τις απαιτήσεις από το λογισμικό «ατάκτως ερριμμένες». Στη συνέχεια γίνεται μια πρώτη ταξινόμηση των απαιτήσεων σε ομάδες, ανάλογα με το υποσύνολο του προβλήματος που αφορούν, ή με άλλο, κατά περίπτωση, πρόσφορο τρόπο.

Στο σημείο αυτό ενδεχομένως να εντοπίζονται ασυνέπειες, δηλαδή δύο ή περισσότερες απαιτήσεις η ικανοποίηση των οποίων δεν μπορεί να γίνει ταυτόχρονα, οπότε είναι αναγκαία η **επίλυση συγκρούσεων**, η οποία μπορεί να επαναφέρει στο προσκήνιο τις επαφές με τον πελάτη, ή άλλη σχετική διαδικασία. Όταν έχει ολοκληρωθεί η επίλυση συγκρούσεων, οι απαιτήσεις τοποθετούνται σε μια **σειρά προτεραιότητας** ως προς τη σειρά ικανοποίησής τους. Η σειρά αυτή θα καθορίσει όχι μόνο τη χρονική αλληλουχία με την οποία ενσωματώνονται στο λογισμικό λειτουργίες που ικανοποιούν τις απαιτήσεις, αλλά και το ποιες από αυτές δε θα ικανοποιηθούν καθόλου, αν κάτι τέτοιο επιβληθεί από εξωτερικούς παράγοντες (λ.χ. κόστος).

Η διαδικασία ολοκληρώνεται με την **επαλήθευση των απαιτήσεων**, όπως έχουν διαμορφωθεί και ιεραρχηθεί. Για να γίνει η επαλήθευση, συνήθως απαιτείται νέα επαφή με τον πελάτη με τη μορφή συσκέψεων ή ανταλλαγής εγγράφων. Στην καλύτερη περίπτωση, οι απαιτήσεις ικανοποιούν τον πελάτη και μπορεί να ξεκινήσει η κατασκευή των μοντέλων περιγραφής λογισμικού που φαίνονται στο Σχήμα 3.17. Σε περίπτωση που οι απαιτήσεις δεν ικανοποιούν τον πελάτη, τότε λαμβάνουν χώρα τόσα πισωγυρίσματα όσα είναι αναγκαία προκειμένου αυτό να γίνει.

Δεν αποτελούν εξαίρεση οι περιπτώσεις που πρέπει κανείς να επανέλθει στη διαδικασία κατανόησης του προβλήματος και να ανακαλύψει νέες πλευρές ή/και νέες ερμηνείες.

3.9.1 Προδιαγραφή Απαιτήσεων

Όταν η παραπάνω διαδικασία έχει ολοκληρωθεί, έχουν κατασκευαστεί οι πρώτες εκδοχές των διαγραμμάτων ροής δεδομένων, οντοτήτων - συσχετίσεων και μετάβασης καταστάσεων. Είναι η στιγμή που μπορεί να πραγματοποιηθεί η αναλυτική προδιαγραφή των απαιτήσεων με τη σύνταξη του εγγράφου «Προδιαγραφές των απαιτήσεων από το λογισμικό».

_ Προδιαγραφή

Με την έννοια «προδιαγραφή» [7] *αναφερόμαστε στη δομημένη και λεπτομερή περιγραφή των απαιτήσεων από το λογισμικό, η οποία γίνεται με τη μορφή γραπτού λόγου και, όπου απαιτείται, διαγραμμάτων.*

Το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό είναι αναμφίβολα το σημαντικότερο από τα έγγραφα τεκμηρίωσης του λογισμικού. Οι ελλείψεις και οι αστοχίες όσων αναφέρονται σε αυτό θα μεταφερθούν σε όλη την υπόλοιπη διαδικασία κατασκευής του λογισμικού και ασφαλώς στο τελικό προϊόν, γεγονός που μπορεί να έχει ως αποτέλεσμα αυτό να είναι άχρηστο. Έχουν προταθεί αρκετοί εναλλακτικοί τρόποι δόμησης του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Είναι γενικά αποδεκτό ότι μερικά επιθυμητά χαρακτηριστικά του εγγράφου αυτού είναι τα ακόλουθα:

- Θα πρέπει να περιγράφει τη συμπεριφορά του λογισμικού προς το εξωτερικό του περιβάλλον (χρήστης, άλλες εφαρμογές λογισμικού) και όχι εσωτερικά του στοιχεία.
- Θα πρέπει να περιγράφει όλους τους περιορισμούς που αφορούν την ανάπτυξη του λογισμικού.
- Θα πρέπει να είναι εύκολο να αλλαχτεί.
- Θα πρέπει να είναι χρήσιμο στη συντήρηση του λογισμικού.
- Θα πρέπει να περιγράφει τη συμπεριφορά του λογισμικού σε ανεπιθύμητες καταστάσεις.

Στο Σχήμα 3.18 παρατίθεται μια δομή του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό βασισμένη σε διεθνές πρότυπο του IEEE (830.1993).

[7] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **69.**

1. Εισαγωγή
    1.1. Ταυτότητα του εγγράφου
    1.2. Σκοπός
    1.3. Εμβέλεια
    1.4. Ορισμοί, ακρωνύμια, συντομογραφίες
    1.5. Πηγές αναφορών
    1.6. Περίληψη
2. Γενική περιγραφή του λογισμικού
    2.1. Στίγμα
    2.2. Προοπτική
    2.3. Γενικές λειτουργίες του λογισμικού
    2.4. Χαρακτηριστικά χρηστών
    2.5. Περιορισμοί
    2.6. Παραδοχές και εξαρτήσεις
3. Ειδικές απαιτήσεις
    3.1. Απαιτήσεις εξωτερικών διαπροσωπειών
        3.1.1. Διαπροσωπείες χρήστη
        3.1.2. Διαπροσωπείες υλικού
        3.1.3. Διαπροσωπείες λογισμικού
        3.1.4. Διαπροσωπείες επικοινωνιών
    3.2. Λειτουργικές απαιτήσεις
        3.2.1. Τρόπος λειτουργίας 1
            3.2.1.1. Λειτουργική απαίτηση 1.1
                Περιγραφή, είσοδοι, επεξεργασία, έξοδοι
            3.2.1.2. Λειτουργική απαίτηση 1.2
                Περιγραφή, είσοδοι, επεξεργασία, έξοδοι

            ……...

        3.2.2. Τρόπος λειτουργίας 2
            3.2.2.1. Λειτουργική απαίτηση 2.1
                Περιγραφή, είσοδοι, επεξεργασία, έξοδοι
            3.2.2.2. Λειτουργική απαίτηση 2.2
                Περιγραφή, είσοδοι, επεξεργασία, έξοδοι

            ……..

3.2.N Τρόπος λειτουργίας N

3.2.N.1. Λειτουργική απαίτηση N.1

Περιγραφή, είσοδοι, επεξεργασία, έξοδοι

3.2.N.2. Λειτουργική απαίτηση N.2

Περιγραφή, είσοδοι, επεξεργασία, έξοδοι

……..

3.3. Απαιτήσεις επιδόσεων

3.4. Περιορισμοί σχεδίασης

3.4.1. Περιορισμοί από το υλικό

3.4.2. Συμμόρφωση με πρότυπα

3.5. Χαρακτηριστικά του λογισμικού

3.5.1. Αξιοπιστία

3.5.2. Διαθεσιμότητα

3.5.3. Ασφάλεια

3.5.4. Χαρακτηριστικά συντήρησης

3.5.5. Μεταφερσιμότητα

3.6. Άλλες απαιτήσεις

ΣΧΗΜΑ 3.19 *Μία προτεινόμενη από το IEEE δομή εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό*

Η παραπάνω περιγραφή αφορά τη δομή, δηλαδή τα κεφάλαια και τα περιεχόμενα καθενός εξ αυτών, τα οποία θα πρέπει να έχει το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό. Το μόνο σχετικά νέο στοιχείο του εγγράφου είναι ο χωρισμός του κεφαλαίου 3 ανάλογα με τον «τρόπο λειτουργίας». Σε μια απλή εφαρμογή λογισμικού υπάρχει μόνο ένας τρόπος λειτουργίας (mode). Δεν ισχύει το ίδιο για μεγάλες εφαρμογές με ιδιαίτερες απαιτήσεις, οι οποίες μπορεί να έχουν περισσότερους του ενός τρόπους λειτουργίας. Η γενική δομή του εγγράφου, η οποία οφείλει να καλύπτει τη γενική περίπτωση, πρέπει να το προβλέψει αυτό. Παράδειγμα αποτελεί μια εφαρμογή συλλογής και επεξεργασίας δεδομένων σε πραγματικό χρόνο. Η συμπεριφορά της εφαρμογής τη στιγμή της συλλογής των δεδομένων, όπου οι απαιτήσεις σε επιδόσεις μπορεί να είναι ιδιαίτερα υψηλές, είναι εντελώς διαφορετική από τη συμπεριφορά της τη στιγμή της στατιστικής επεξεργασίας ή της συντήρησης των δεδομένων αυτών, οπότε διακρίνουμε δύο τρόπους λειτουργίας.

Επανερχόμενοι, μελετώντας τη δομή του εγγράφου, διαπιστώνουμε ότι όλες οι απαιτήσεις από το λογισμικό μπορούν να ταξινομηθούν ώστε να περιγραφούν σε αυτό. Η ταξινόμηση αυτή δεν είναι πάντα εύκολη υπόθεση και συχνά δημιουργούνται συγχύσεις και ερωτήματα του τύπου «Τι πρέπει να γραφτεί σε αυτή την παράγραφο;». Η απάντηση δεν είναι εύκολη και απαιτείται αρκετή τριβή με το αντικείμενο μέχρις ότου να μπορεί κανείς να νιώθει εμπιστοσύνη στον τρόπο με τον οποίο αντιμετωπίζει το θέμα. Η εμπειρία δείχνει ότι σε επόμενο στάδιο της ανάπτυξης θα φανεί το πόσο επαρκές είναι το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό και θα δοθεί η ευκαιρία, ενδεχομένως με κάποια αναθεώρηση, αυτό να βελτιωθεί. Η ανάγκη για τέτοιες αναθεωρήσεις συνεχώς θα μειώνεται όσο αποκτάται εμπειρία.

ΚΕΦΑΛΑΙΟ 4 : Σχεδίαση

4.1 Σκοπός της Σχεδίασης

Στην ενότητα αυτή θα οριστεί το αντικείμενο της σχεδίασης του λογισμικού και θα οριοθετηθούν οι στόχοι τους οποίους καλείται να εκπληρώσει ο μηχανικός λογισμικού που ασχολείται με τη σχεδίαση.

Αυτό που ζητείται από τη σχεδίαση είναι ένας τρόπος περιγραφής της κατασκευής του λογισμικού έτσι ώστε αυτό να ικανοποιεί τις προδιαγραφές που έχουν τεθεί, δηλαδή να μπορεί να εκτελεί τις επιθυμητές λειτουργίες και να έχει τα επιθυμητά χαρακτηριστικά. Η ύπαρξη των προδιαγραφών είναι αναγκαία για να ξεκινήσει η σχεδίαση και επιβάλλεται από τις αρχές της Τεχνολογίας Λογισμικού. Η περιγραφή αυτή, που παράγεται κατά τη σχεδίαση, ονομάζεται «σχέδιο του λογισμικού».

4.1.1 Σχέδιο λογισμικού

Σχέδιο λογισμικού[8] *είναι η περιγραφή των μονάδων που αποτελούν το λογισμικό, των συσχετίσεων μεταξύ τους, της διάταξής τους, καθώς και της εσωτερικής τους λεπτομέρειας.*

Η παραγωγή του σχεδίου του λογισμικού είναι αναγκαία προκειμένου να γίνει δυνατή η κατασκευή του, όπως, άλλωστε, ισχύει για κάθε τεχνικό έργο. Η λύση στο πρόβλημα της σχεδίασης λογισμικού δεν είναι καθόλου εύκολη. Για κάθε προδιαγραφή δεν είναι παράλογο να θεωρούμε ότι μπορούμε να κατασκευάσουμε περισσότερα του ενός σχέδια, δηλαδή να θεωρούμε ότι μπορεί να υλοποιηθεί με περισσότερους του ενός τρόπους. Μερικές από τις σημαντικότερες πλευρές του προβλήματος της σχεδίασης είναι οι ακόλουθες:

- Με ποια στρατηγική πρέπει να αντιμετωπίσουμε τη μετάβαση από τις προδιαγραφές στη σχεδίαση έτσι ώστε η εργασία μας ναείναι αποτελεσματική;
- Ποιος από τους τρόπους που μπορούμε να σκεφτούμε για την υλοποίηση μιας προδιαγραφής είναι ο καλύτερος και πώς τεκμηριώνεται αυτό;
- Σε ποιο βαθμό δεσμεύεται η σχεδίαση από το περιβάλλον (γλώσσα προγραμματισμού, εργαλεία, λειτουργικό σύστημα) στο οποίο θα γίνει η κατασκευή του προγράμματος;

[8] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **105.**

- Ποια είναι η περισσότερο *επαρκής*, δηλαδή κατάλληλη, χωρίς ναείναι ούτε ελλιπής ούτε φλύαρη, περιγραφή του σχεδίου του λογισμικού;
- Πώς εξασφαλίζεται η ποιότητα του παραγόμενου λογισμικού μέσα από τις εργασίες που λαμβάνουν χώρα κατά τη σχεδίαση;

Σκοπός της σχεδίασης είναι να δώσει την καλύτερη δυνατή στις εκάστοτε συνθήκες απάντηση στα παραπάνω ερωτήματα. Ας σημειωθεί ότι δεν υπάρχει η «καλύτερη», κατ'απόλυτη έννοια, λύση, παρά μόνο η «καλύτερη δυνατή στις εκάστοτε συνθήκες». Η εποχή που επιδίωξη των μηχανικών λογισμικού ήταν η εύρεση της απόλυτα καλύτερης και γενικής λύσης στο πρόβλημα της σχεδίασης έχει δώσει τη θέση της στο ρεαλισμό της επιδίωξης της βέλτιστης λύσης μέσα σε κάθε συγκεκριμένο περιβάλλον κατασκευής λογισμικού.

Παρά τον υποκειμενισμό που, γενικά, διέπει το χαρακτηρισμό ενός σχεδίου λογισμικού, είναι χρήσιμο να συμφωνούνται κριτήρια ποιότητας, στα οποία να αποδίδεται η προσήκουσα κατά περίπτωση βαρύτητα. Τέσσερα τέτοια κριτήρια είναι τα ακόλουθα:

- Το σχέδιο πρέπει να ικανοποιεί όλες τις προδιαγραφές των απαιτήσεων από το λογισμικό (λειτουργικές και μη).
- Το σχέδιο πρέπει να περιγράφει πλήρως όλες τις πλευρές του λογισμικού: δεδομένα, λειτουργίες και συμπεριφορά, όπως αυτές θεωρούνται από την πλευρά του κατασκευαστή.
- Το σχέδιο πρέπει να είναι εύκολα κατανοητό από αυτούς που θα κληθούν να συγγράψουν τον πηγαίο κώδικα, δηλαδή τους προγραμματιστές.
- Το σχέδιο δεν πρέπει να περιέχει σφάλματα.

Τα παραπάνω, εκτός από κριτήρια, μπορούν να ακούγονται εξίσου εύκολα και ως ευχές. Μια πρακτική σχεδίασης πρέπει να στοχεύει στην καθοδήγηση του κατασκευαστή στην παραγωγή σχεδίου λογισμικού το οποίο να πληροί στο μέγιστο δυνατό βαθμό τα παραπάνω κριτήρια. Αυτό δεν είναι πάντα εύκολο, καθότι ενδέχεται να υπάρχουν και εσωτερικές αντιφάσεις μέσα στα κριτήρια (λ.χ. αντιφάσκουσες μεταξύ τους προδιαγραφές), αλλά και άλλοι πρακτικοί λόγοι (λ.χ. απαγορευτικό από τον προϋπολογισμό κόστος πλήρους λεπτομερούς περιγραφής του σχεδίου). Η Τεχνολογία Λογισμικού παρέχει μεθοδολογίες σχεδίασης οι οποίες αποτελούν κατευθυντήριες γραμμές για το σχεδιαστή. Συνδυάζοντας τη γνώση, την εμπειρία και τον αυτοσχεδιασμό, αλλά και με τη χρήση των κατάλληλων

εργαλείων ανάπτυξης λογισμικού, ο σχεδιαστής μπορεί να αντιμετωπίσει την πρόκληση, γιατί περί πρόκλησης πρόκειται κανοποίησης των κριτηρίων αυτών.

4.2 Τεχνοτροπίες σχεδίασης

Το πρόβλημα της σχεδίασης λογισμικού μπορεί να αντιμετωπιστεί με διάφορες στρατηγικές προσεγγίσεις. Οι διάφορες μεθοδολογίες που έχουν παρουσιαστεί μπορούν να ενταχθούν σε δύο μεγάλες κατηγορίες: τις *προσανατολισμένες στις διαδικασίες* (function oriented) και τις *προσανατολισμένες στα αντικείμενα* (object oriented). Μια σύντομη αναφορά στα χαρακτηριστικά των δύο κατηγοριών δίνεται στην ενότητα αυτή.

Η απάντηση στην ερώτηση «Τι θα κάνει το λογισμικό;» για μεγάλο χρονικό διάστημα δινόταν με τη μορφή μιας ιεραρχίας διαδικασιών, συναρτήσεων και άλλων ενεργών μονάδων λογισμικού, η οποία επιδρούσε σε ένα σύνολο ανεξάρτητων δεδομένων. Αν και από τη δεκαετία του 60 ακόμα είχαν παρουσιαστεί και άλλες απόψεις, η προσέγγιση αυτή επικράτησε και πάνω της γεννήθηκαν διάφορες μεθοδολογίες σχεδίασης λογισμικού, οι οποίες συγκρότησαν την οικογένεια μεθοδολογιών δομημένης σχεδίασης.

Η «άλλη άποψη» προσπαθούσε να απαντήσει στο ίδιο ερώτημα χωρίς να διακρίνει τα δεδομένα από τις διαδικασίες, προτείνοντας ένα σύνολο συνεργαζόμενων οντοτήτων που περιλαμβάνουν στο ίδιο κέλυφος και δεδομένα και διαδικασίες. Οι οντότητες αυτές ονομάστηκαν αντικείμενα (objects). Οι μεθοδολογίες που γεννήθηκαν πάνω σε αυτή την προσέγγιση ονομάστηκαν προσανατολισμένες στα αντικείμενα (object.oriented).

Και στις δύο περιπτώσεις, ο σχεδιαστής καταπιάνεται και με τις διεργασίες και με τα δεδομένα, αποδίδοντάς τους όμως διαφορετική βαρύτητα σε κάθε περίπτωση. Οι δύο οικογένειες φαίνονται στο Σχήμα 4.1.

ΣΧΗΜΑ 4.1 *Μια ταξινόμηση των τεχνοτροπιών σχεδίασης λογισμικού*

4.2.1 Δομημένη σχεδίαση

Οι μεθοδολογίες που ακολουθούν αυτή την προσέγγιση προτείνουν τρόπους αποσύνθεσης του συστήματος *από πάνω προς τα κάτω* (top.down) σε μια ιεραρχία διαδικασιών, συναρτήσεων και άλλων ενεργών μονάδων λογισμικού. Όσο κατεβαίνει κανείς στην ιεραρχία αυτή, τόσο μεγαλύτερη λεπτομέρεια συναντά, μέχρις ότου φτάσει στις απλές δομικές μονάδες, δηλαδή τις εντολές της γλώσσας προγραμματισμού. Η προσέγγιση αυτή παρουσιάζεται στο παρόν βιβλίο.

Γνωστές μεθοδολογίες που ανήκουν στην οικογένεια αυτή έχουν προταθεί από πολλούς συγγραφείς και για ενημέρωση μπορείτε να ανατρέξετε στη βιβλιογραφία. Οι περισσότερες των προσεγγίσεων αυτών επικεντρώνουν την προσοχή τους στις διαδικασίες πρώτα και μετά στα δεδομένα. Οι πιο σύγχρονες καθορίζουν τη δομή των διαδικασιών που επιδρούν πάνω στα δεδομένα με βάση τη δομή των δεδομένων αυτών και για το λόγο αυτό χαρακτηρίζονται ως «βασισμένες στα δεδομένα» και συγγενεύουν εν μέρει με τις «προσανατολισμένες στα αντικείμενα» τεχνοτροπίες.

4.2.2 Αντικεινενοστρεφής σχεδίαση

Η *αντικειμενοστρεφής προσέγγιση* (object.oriented) ακολουθεί ένα διαφορετικό δρόμο: αντί το σύστημα να θεωρείται ως μια ιεραρχία διαδικασιών, ανεξάρτητων από τα δεδομένα, θεωρείται ως μια συλλογή οντοτήτων, καθεμία εκ των οποίων περικλείει και διαδικασίες και δεδομένα. Η προσέγγιση βασίζεται στην ιδέα ότι στον πραγματικό κόσμο δεδομένα και

διαδικασίες μπορούν να ιδωθούν ενιαία με βάση το πεδίο ευθύνης κάποιων οντοτήτων που ονομάζονται «αντικείμενα».

Κάθε αντικείμενο παρέχει στο περιβάλλον του ένα σύνολο υπηρεσιών της ευθύνης του. Η συνεργασία του συνόλου των αντικειμένων του πεδίου μιας εφαρμογής λογισμικού παράγει το επιθυμητό αποτέλεσμα.

Μερικές από τις πιο γνωστές προσεγγίσεις που ανήκουν στην κατηγορία αυτή προτάθηκαν από τους Booch[6] (1994), Jacobsen[15] (1993), Για μεγάλο χρονικό διάστημα επικρατούσε μια σύγχυση σε επίπεδο ορολογίας και συμβολισμών στην οικογένεια της αντικειμενοστρεφούς ανάλυσης και σχεδίασης. Τα τελευταία χρόνια η σύγχυση αυτή φαίνεται να περιορίζεται με την εμφάνιση «συγχωνευμένων» μεθοδολογιών και ενοποιημένων συμβολισμών.

4.3 Αντικείμενο και αποτελέσματα της σχεδίασης

Στην ενότητα αυτή θα καθοριστούν τα αντικείμενα εργασίας κατά τη σχεδίαση λογισμικού, δηλαδή το ποια σχέδια πρέπει να κατασκευαστούν και το τι περιγράφει καθένα εξ αυτών. Επίσης, θα παρουσιαστεί ένας τρόπος παράστασης των αποτελεσμάτων της σχεδίασης με δομημένο κείμενο, ανάλογος με αυτόν του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Η αναφορά θα ακολουθήσει την προσέγγιση της δομημένης σχεδίασης.

Στον ορισμό που δόθηκε στην προηγούμενη ενότητα το σχέδιο λογισμικού φέρεται ως «η περιγραφή των μονάδων που αποτελούν το λογισμικό, των συσχετίσεων μεταξύ τους, της διάταξής τους, καθώς και της εσωτερικής τους λεπτομέρειας». Προκειμένου να γίνει δυνατή η περιγραφή αυτή, πρέπει να αντιμετωπίσουμε το πρόβλημα σε τέσσερα επίπεδα ως ακολούθως:

- **Αρχιτεκτονική σχεδίαση:** Αφορά τον καθορισμό τού ποιες είναι

οι μονάδες που συγκροτούν το σύστημα λογισμικού και πώς αυτές ανατίθενται για εκτέλεση (διατάσσονται) στις υπολογιστικές μονάδες που είναι διαθέσιμες.

- **Σχεδίαση διαπροσωπειών** (interfaces)**:** Αφορά τον καθορισμό της επικοινωνίας των μονάδων μεταξύ τους, με άλλα συστήματα λογισμικού, με άλλες συσκευές, καθώς και με τον άνθρωπο.

Λέγοντας «καθορισμός επικοινωνίας» εννοούμε την περιγραφή του ποια μονάδα επικοινωνεί με ποια, με ποιες παραμέτρους, καθώς και με όποιο άλλο στοιχείο απαιτείται για να περιγραφεί επαρκώς, κατά περίπτωση, η επικοινωνία.

- **Λεπτομερής σχεδίαση μονάδων:** Αφορά τον καθορισμό της εσωτερικής δομής κάθε μονάδας λογισμικού προκειμένου αυτή να ικανοποιεί, αφενός, τις λειτουργικές απαιτήσεις και αφετέρου, να συνεργάζεται με τις άλλες μονάδες όπως έχει καθοριστεί κατά την αρχιτεκτονική και τη σχεδίαση δεδομένων.
- **Σχεδίαση δεδομένων:** Πρόκειται για τη λεπτομερή σχεδίαση των δεδομένων, η τήρηση των οποίων αποτελεί απαίτηση από το λογισμικό, η οποία έχει τεθεί στη φάση προδιαγραφής των απαιτήσεων.

Είσοδο στην εργασία της σχεδίασης αποτελούν τα προϊόντα που έχουν παραχθεί κατά τη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό. Σε όλα της τα επίπεδα, η σχεδίαση λογισμικού είναι μια εργασία που προχωρά από τη μικρότερη στη μεγαλύτερη λεπτομέρεια, όπως φαίνεται στο Σχήμα 4.2.

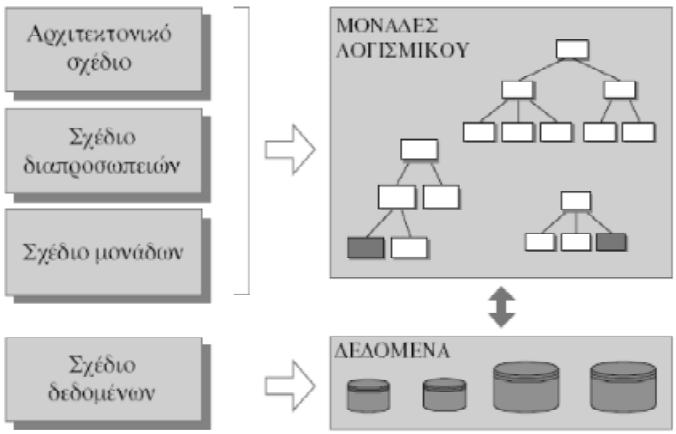
ΣΧΗΜΑ 4.2 *Εξέλιξη της σχεδίασης λογισμικού*

Στο Σχήμα 4.3 φαίνεται μια συσχέτιση μεταξύ των προϊόντων σχεδίασης, που προκύπτουν από την εκτέλεση των εργασιών στα τέσσερα επίπεδα που αναφέρθηκαν, με τα προϊόντα της φάσης προσδιορισμού των προδιαγραφών.

ΣΧΗΜΑ 4.3 *Από τα προϊόντα προδιαγραφών των απαιτήσεων στα προϊόντα σχεδίασης λογισμικού*

Όπως φαίνεται και στο Σχήμα 4.3, στη δομημένη ανάλυση και σχεδίαση μπορούμε να διακρίνουμε τη σχεδίαση των δεδομένων από τη σχεδίαση που αφορά τις μονάδες προγράμματος. Η σχεδίαση των δεδομένων φαίνεται να σχετίζεται σχεδόν αποκλειστικά με το διάγραμμα οντοτήτων - συσχετίσεων και με το λεξικό δεδομένων, ενώ η σχεδίαση μονάδων (αρχιτεκτονική, επικοινωνία, δομή) σχετίζεται με το σύνολο των υπόλοιπων αποτελεσμάτων των προδιαγραφών. Αποφεύγουμε τον ορισμό της έννοιας «μονάδα λογισμικού», διότι αυτός εξαρτάται από τη γλώσσα προγραμματισμού που χρησιμοποιείται. Οι *συναρτήσεις* (functions), οι *υπορουτίνες* (subroutines), οι *μονάδες* (units), οι *διαδικασίες* (procedures) είναι οι συνηθέστεροι όροι που χρησιμοποιούνται από τις γλώσσες προγραμματισμού για να περιγράψουν μονάδες λογισμικού.

Στο Σχήμα 4.4 φαίνεται η ανεξαρτησία των δεδομένων από τις μονάδες λογισμικού, η οποία, όπως τονίστηκε, είναι ουσιώδες χαρακτηριστικό της δομημένης ανάλυσης και σχεδίασης. Από το αρχιτεκτονικό σχέδιο, το σχέδιο διαπροσωπειών και το λεπτομερές σχέδιο μονάδων, τελικά θα κατασκευαστεί μια δομή ενεργών συστατικών (μονάδων) λογισμικού, τα οποία θα επιδρούν πάνω σε κάποια ανεξάρτητα δεδομένα.



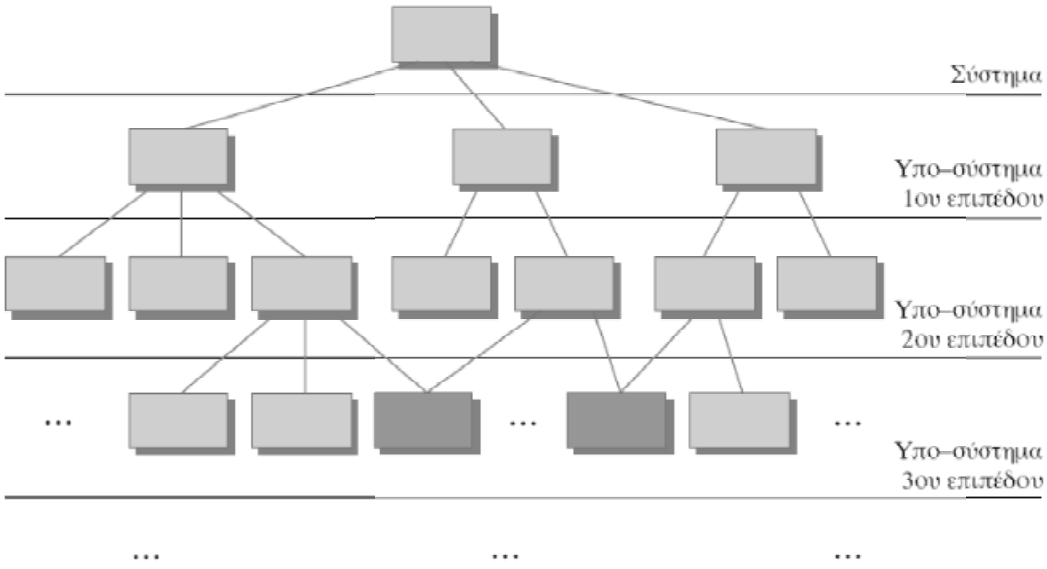
ΣΧΗΜΑ 4.4 *Δεδομένα και μονάδες λογισμικού στη δομημένη ανάλυση και σχεδίαση*

4.3.1 Αρχιτεκτονική σχεδίαση

Όπως αναφέρθηκε, κατά την *αρχιτεκτονική σχεδίαση* προσδιορίζεται το ποιες είναι οι μονάδες που αποτελούν το λογισμικό και πώς σχετίζονται μεταξύ τους. Ο προσδιορισμός αυτός αναφέρεται και ως *σχεδίαση συστήματος*. Στη δομημένη σχεδίαση η εργασία αυτή γίνεται λαμβάνοντας ως είσοδο τα διαγράμματα ροής δεδομένων. Σε αυτά εφαρμόζεται ένα σύνολο κανόνων και καθορίζονται οι μονάδες λογισμικού που είναι αναγκαίες για την υλοποίηση των μετασχηματισμών και των ροών δεδομένων.

Η αρχιτεκτονική δομή του λογισμικού μπορεί να ιδωθεί σε πολλά επίπεδα λεπτομέρειας. Στο Σχήμα 4.5 φαίνεται μια άποψη όπου το λογισμικό θεωρείται στο πρώτο επίπεδο ως ένα σύνθετο σύστημα, το οποίο αποτελείται από υποσυστήματα (δεύτερο επίπεδο), καθένα εκ των οποίων αποτελείται από υποσυστήματα 2ας τάξης (τρίτο επίπεδο) κοκ., μέχρις ότου φτάσουμε στις ατομικές μονάδες λογισμικού
(συναρτήσεις, υπορουτίνες, κτλ.).

Η αρχιτεκτονική λογισμικού καθορίζει ποιες θα είναι αυτές και στη συνέχεια το λεπτομερές σχέδιο μονάδων θα καθορίσει πώς ακριβώς θα υλοποιηθούν. Σε όσο χαμηλότερο επίπεδο βρισκόμαστε, τόσο πιθανότερο είναι να αναγνωρίζονται «κοινοί παράγοντες», δηλαδή μονάδες λογισμικού που ανήκουν σε περισσότερα του ενός υποσυστήματα του αλέσω παραπάνω επιπέδου. Το φαινόμενο εμφανίζεται κυρίως μετά από την εκτέλεση της *συναρτησιακής αποσύνθεσης* (functional decomposition), στην οποία θα αναφερθούμε κατά την περιγραφή της λεπτομερούς σχεδίασης μονάδων. Στο Σχήμα 4.5 τέτοιες μονάδες σημειώνονται με γκρι χρώμα.



ΣΧΗΜΑ 4.5 *Επίπεδα αρχιτεκτονικού σχεδίου λογισμικού*

Οι μονάδες που αποτελούν το λογισμικό μπορεί να ανατίθενται προς εκτέλεση σε πολλούς διατιθέμενους υπολογιστικούς πόρους (υπολογιστικά συστήματα, μονάδες επεξεργασίας). Στην απλούστερη περίπτωση, το λογισμικό είναι κατασκευασμένο ώστε να τρέχει εξ ολοκλήρου σε ένα μόνο υπολογιστικό σύστημα. Στη γενικότερη περίπτωση, τα υποσυστήματα του λογισμικού μπορούν να ανατίθενται σε διαφορετικούς υπολογιστικούς πόρους.

4.3.2 Σχεδίαση διαπροσωπειών

Καμία μονάδα λογισμικού δεν είναι ανεξάρτητη. Είτε χρησιμοποιεί άλλες μονάδες προκειμένου να επιτελέσει τη λειτουργία για την οποία προορίζεται είτε χρησιμοποιείται από άλλες μονάδες για τον ίδιο σκοπό. Η διατύπωση ότι μια μονάδα λογισμικού χρησιμοποιεί άλλες μονάδες δηλώνει την κλήση προγραμμάτων, υπορουτινών, συναρτήσεων και άλλων δομικών στοιχείων λογισμικού. Η κλήση μιας μονάδας Β από μια μονάδα Α σημαίνει:

- ενεργοποίηση της μονάδας Β, δηλαδή η ροή του προγράμματος μεταφέρεται στη μονάδα Β και εκτελούνται οι εντολές που περιέχονται σε αυτή, καθώς και
- επικοινωνία της μονάδας Α με τη Β μέσω παραμέτρων που περνά η Α στη Β και ενδεχομένως και αντίστροφα.

Κατά τη *σχεδίαση διαπροσωπειών* (interface design) καθορίζονται οι λεπτομέρειες του περάσματος των παραμέτρων αυτών σε όλα τα επίπεδα επικοινωνίας. Οι μονάδες λογισμικού δεν επικοινωνούν μόνο μεταξύ τους, αλλά και με εξωτερικά συστήματα ή συσκευές, καθώς και με τον άνθρωπο. Συγκεντρώνοντας το σύνολο των απαιτήσεων από τη σχεδίαση διαπροσωπειών, καταλήγουμε ότι κατά την εργασία αυτή θα πρέπει να καθοριστούν τα ακόλουθα:

- Ο αριθμός και ο τύπος των παραμέτρων κατά την κλήση μονάδων λογισμικού.
- Το είδος και οι λεπτομέρειες της επικοινωνίας με άλλα συστήματα λογισμικού.
- Το είδος και οι λεπτομέρειες της επικοινωνίας με εξωτερικές συσκευές.
- Η επικοινωνία του λογισμικού με τον άνθρωπο.

Οι παράμετροι κατά την επικοινωνία με άλλες μονάδες λογισμικού (στην ίδια εφαρμογή) καθορίζονται κατά την αρχιτεκτονική και τη λεπτομερή σχεδίαση και περιγράφονται στα διαγράμματα δομής προγράμματος και το λεπτομερές σχέδιο μονάδων, στα οποία θα αναφερθούμε σε επόμενες παραγράφους. Οι εξωτερικές διαπροσωπείες περιγράφονται αυτοτελώς με τον προσφορότερο κατά περίπτωση τρόπο, ανάλογα με τα εξωτερικά συστήματα λογισμικού ή τις εξωτερικές συσκευές. Η σχεδίαση της διαπροσωπείας με τον άνθρωπο είναι ένα ιδιαίτερα πολύπλοκο πρόβλημα, στο οποίο εκτός από μηχανικούς λογισμικού μπορεί να έχουν λόγο και άλλες ειδικότητες, όπως κοινωνιολόγοι και ψυχολόγοι, καθώς και κοινοί χρήστες.

4.3.3 Λεπτομερής σχεδίαση μονάδων

Το περισσότερο λεπτομερές επίπεδο της σχεδίασης είναι η *σχεδίαση μονάδων*. Πρόκειται για το Σητεία όπου πρέπει να καθοριστούν όλες οι κατασκευαστικές λεπτομέρειες που απαιτούνται για τη δημιουργία του πηγαίου κώδικα για όλες τις μονάδες λογισμικού. Η γνώση της ύπαρξης μιας μονάδας και των χαρακτηριστικών επικοινωνίας αυτής με άλλες μονάδες δεν επαρκεί, στη γενική περίπτωση, για την κατασκευή του πηγαίου κώδικα της ίδιας της μονάδας. Σε τετριμμένες περιπτώσεις, η λεπτομερής περιγραφή της εσωτερικής δομής δεν είναι αναγκαία για την κατασκευή της μονάδας. Στις περισσότερες όμως των περιπτώσεων αυτό δεν ισχύει και η *λεπτομερής σχεδίαση μονάδων* είναι αναγκαία.

Το υλικό που χρησιμοποιείται ως είσοδος στη φάση της λεπτομερούς σχεδίασης μονάδων είναι το «αρχιτεκτονικό σχέδιο», το «σχέδιο διαπροσωπειών», το «έγγραφο

προδιαγραφών των απαιτήσεων από το λογισμικό», καθώς και το «διάγραμμα μετάβασης καταστάσεων». Τα δύο πρώτα έχουν κατασκευαστεί κατά τη σχεδίαση, ενώ τα δύο τελευταία κατασκευάστηκαν κατά την προδιαγραφή των απαιτήσεων και περιέχουν περιγραφή της λειτουργικής συμπεριφοράς του λογισμικού με τη μορφή απαιτήσεων.

Κατά τη λεπτομερή σχεδίαση, για κάθε μονάδα θα δοθεί ένα περίγραμμα-ομοίωμα της δομής της, χρήσιμο για την κατασκευή της. Αποτέλεσμα της λεπτομερούς σχεδίασης είναι το *λεπτομερές σχέδιο μονάδων*, το οποίο στη συνέχεια θα δοθεί στους προγραμματιστές που θα συγγράψουν τον πηγαίο κώδικα.

4.3.4 Σχεδίαση δεδομένων

Η *σχεδίαση δεδομένων* θεωρείται ως μια από τις σημαντικότερες διαδικασίες στην ανάπτυξη λογισμικού. Η σύλληψη της δομής και των συσχετίσεων των δεδομένων μπορεί να καθορίζει πολλά από τα χαρακτηριστικά των λειτουργικών μονάδων λογισμικού. Στη δομημένη ανάλυση και σχεδίαση, ο προσδιορισμός των απαιτήσεων και η λεπτομερής σχεδίαση των δεδομένων, από τη μία, και η σχεδίαση των μονάδων λογισμικού, από την άλλη, είναι δύο διαφορετικές διακριτές εργασίες.

Προκειμένου να μπορεί να κατασκευαστεί με πληρότητα το λεπτομερές σχέδιο μονάδων λογισμικού, είναι απαραίτητο να διατίθεται το λεπτομερές σχέδιο δεδομένων. Αυτό ισχύει για τις περιπτώσεις λειτουργικών μονάδων λογισμικού που επιδρούν, με οποιονδήποτε τρόπο, πάνω σε δεδομένα. Έχοντας, λοιπόν, υπόψη το λεξικό δεδομένων και το διάγραμμα οντοτήτων - συσχετίσεων, ο σχεδιαστής λογισμικού:

- Πραγματοποιεί βελτιστοποιήσεις στο μοντέλο οντοτήτων-συσχετίσεων, με σκοπό τη βελτιστοποίηση των επιδόσεων, την πληροφοριακή πληρότητα και την εξάλειψη πλεονάζουσας (redundant) πληροφορίας.
- Καθορίζει τη δομή κάθε πίνακα (πεδία και τύποι αυτών) στο φυσικό επίπεδο.
- Καθορίζει τις δυνατές απόψεις των δεδομένων στο λογικό επίπεδο (views).

Με το αντικείμενο της λεπτομερούς σχεδίασης δεδομένων ασχολείται η γνωστική περιοχή των βάσεων δεδομένων. Το αποτέλεσμα της σχεδίασης δεδομένων είναι η λεπτομερής περιγραφή των οντοτήτων και των συσχετίσεων αυτών, σε επίπεδο που να είναι εφικτή η υλοποίηση της Βάσης Δεδομένων.

4.3.5 Το έγγραφο περιγραφής του σχεδίου του λογισμικού

Σε αναλογία με το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό, είναι χρήσιμο το σχέδιο του λογισμικού να αποτυπώνεται με χρήση ενός δομημένου εγγράφου. Το έγγραφο αυτό αποτελεί ένα ενιαίο δομικό κέλυφος στο οποίο ενσωματώνονται όλα τα επιμέρους προϊόντα της σχεδίασης με τη μορφή διαγραμμάτων, αλλά και με τη μορφή κειμένων ή άλλων τρόπων περιγραφής σχεδίου, τους οποίους θα αναφέρουμε στη συνέχεια. Μια γενική δομή του εγγράφου περιγραφής του σχεδίου του λογισμικού προτείνεται από το IEEE και παρατίθεται στο Σχήμα 4.6.

1. Εισαγωγή

- 1.1 Γενική περιγραφή
- 1.2 Εμβέλεια
- 1.3 Ορισμοί και ακρωνύμια
- 1.4 Αναφορές

2. Περιγραφή αποσύνθεσης του λογισμικού

- 2.1 Αποσύνθεση σε μονάδες
  - 2.1.1 Περιγραφή μονάδας 1
  - 2.1.2 Περιγραφή μονάδας 2
  - ...
- 2.2 Αποσύνθεση σε ταυτόχρονες διεργασίες
  - 2.2.1 Περιγραφή διεργασίας 1
  - 2.2.2 Περιγραφή διεργασίας 2
  - ...

2.3 Αποσύνθεση δεδομένων
2.3.1 Περιγραφή οντότητας δεδομένων 1
2.3.2 Περιγραφή οντότητας δεδομένων 2
...

3. Περιγραφή εξαρτήσεων
3.1 Εξαρτήσεις μεταξύ μονάδων
3.2 Εξαρτήσεις μεταξύ διεργασιών
3.3 Εξαρτήσεις μεταξύ δεδομένων

4. Περιγραφή διαπροσωπειών
4.1 Διαπροσωπείες μονάδων
4.2 Διαπροσωπείες διεργασιών
4.3 Διαπροσωπείες χρήστη
4.4 Εξωτερικές διαπροσωπείες
4.4.1 Συστήματα λογισμικού
4.4.2 Συσκευές

5. Λεπτομερές σχέδιο μονάδων
5.1 Λεπτομερές σχέδιο μονάδας 1
5.2 Λεπτομερές σχέδιο μονάδας 2
...

6. Λεπτομερές σχέδιο δεδομένων
6.1 Οντότητα δεδομένων 1
6.2 Οντότητα δεδομένων 2
...

ΣΧΗΜΑ 4.6 *Το έγγραφο περιγραφής του σχεδίου του λογισμικού (προτεινόμενη δομή σύμφωνα με το IEEE)*

4.4 Διατάξεις Λογισμικού

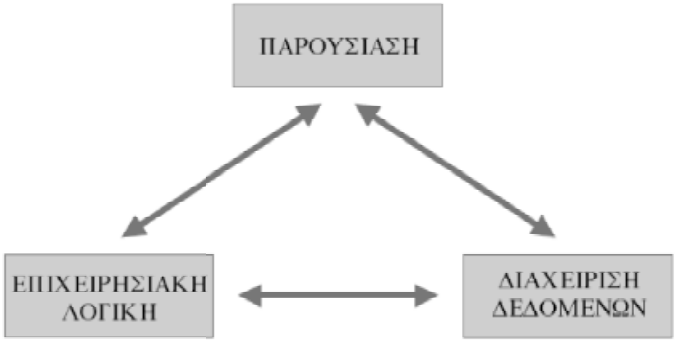
Πριν προχωρήσουμε στην αναφορά της προσέγγισης της «δομημένης σχεδίασης», είναι χρήσιμο να αναφερθούμε στις υπάρχουσες επιλογές διατάξεων λογισμικού. Θα παρουσιαστούν τέσσερα μοντέλα διατάξεων: το *μονολιθικό μοντέλο*, το *μοντέλο πελάτη - εξυπηρετητή* (client.server), το *τριμερές μοντέλο* (3.tier), καθώς και ένα γενικευμένο μοντέλο *πολλαπλής διάταξης* (multi.tier). Προκειμένου να αποφύγουμε σύγχυση και διφορούμενα, θα εισαγάγουμε την έννοια της «διάταξης λογισμικού».

4.4.1 Διάταξη λογισμικού

Διάταξη λογισμικού[9] *(software deployment) είναι η κατάτμηση μιας εφαρμογής σε ανεξάρτητα λειτουργικά τμήματα και η ανάθεση αυτών σε διατιθέμενους υπολογιστικούς πόρους (συστήματα, επεξεργαστές).*

Με βάση τον παραπάνω ορισμό, με τον όρο «διάταξη» αναφερόμαστε στη γενική αρχιτεκτονική του λογισμικού. Σε αρκετές περιπτώσεις χρησιμοποιείται σκέτος ο όρος «αρχιτεκτονική», ιδιαίτερα εκεί όπου το λογισμικό θεωρείται από εξωτερική σκοπιά και όχι από αυτή του κατασκευαστή. Ακολούθως θα αναφερόμαστε στη γενική αρχιτεκτονική με τον όρο «διάταξη» και στην εσωτερική αρχιτεκτονική, στην οποία ήδη αναφερθήκαμε, με τον όρο «αρχιτεκτονική».

Ο ορισμός των διατάξεων που θα ακολουθήσει θα βασιστεί στην κατηγοριοποίηση των εργασιών που μπορεί να κάνει μια εφαρμογή λογισμικού, όπως φαίνεται στο Σχήμα 4.7. Η κατηγοριοποίηση αυτή έχει γίνει γενικά αποδεκτή από κατασκευαστές και ερευνητές. Διακρίνονται τρία είδη εργασιών: οι εργασίες *παρουσίασης*, οι εργασίες *διαχείρισης δεδομένων* και οι εργασίες *επιχειρησιακής λογικής*.



ΣΧΗΜΑ 4.7 *Μια διάκριση των εργασιών που κάνει μια εφαρμογή λογισμικού*

Ως εργασίες **παρουσίασης** ορίζονται όλες οι εργασίες που σχετίζονται με την επικοινωνία του συστήματος με το χρήστη και με εξωτερικές συσκευές και συστήματα, δηλαδή εργασίες που υλοποιούν τις διαπροσωπείες του λογισμικού. Οι εργασίες **διαχείρισης δεδομένων** είναι εκείνες που ασχολούνται με την αποθήκευση και την ανάκτηση των δεδομένων. Τέλος, οι εργασίες **επιχειρησιακής λογικής** (businesslogic) είναι όλες οι

[9] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **118.**

υπόλοιπες εργασίες, δηλαδή εκείνες που υλοποιούν τις ιδιαίτερες λειτουργικές απαιτήσεις κάθε εφαρμογής λογισμικού.

Η διάκριση αυτή απαιτεί μια αυστηρότητα στον ορισμό των μονάδων λογισμικού. Μια μονάδα η οποία εκτελεί έναν υπολογισμό (εργασία επιχειρησιακής λογικής) δε θα πρέπει να τυπώνει η ίδια το αποτέλεσμά του στην οθόνη (εργασία παρουσίασης).

4.4.2 Η μονολιθική διάταξη

Η απλούστερη διάταξη λογισμικού είναι η *μονολιθική* (Σχήμα 4.8). Σε αυτήν ολόκληρη η εφαρμογή τρέχει σε ένα υπολογιστικό σύστημα. Η διάταξη αυτή είναι κατάλληλη για μικρές εφαρμογές με σχετικά περιορισμένες απαιτήσεις και λειτουργίες και, όπως είναι αναμενόμενο, υπήρξε η πρώτη διάταξη που για μεγάλο χρονικό διάστημα χρησιμοποιήθηκε.

ΣΧΗΜΑ 4.8 *Η μονολιθική διάταξη λογισμικού*

Ο συμβολισμός που χρησιμοποιείται στο Σχήμα 4.8 αναφέρεται ως *διάγραμμα διάταξης λογισμικού* (deployment diagram) και περιγράφει την ανάθεση τμημάτων της εφαρμογής σε υπολογιστικούς πόρους. Τα τμήματα αυτά συμβολίζονται με ένα τρισδιάστατο παραλληλεπίπεδο σκιασμένο όπως στο σχήμα, στην μπροστινή πλευρά του οποίου αναγράφεται η ονομασία κάθε τμήματος.

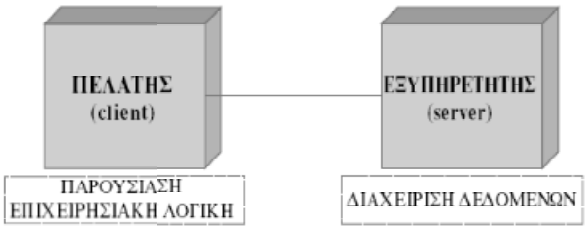
4.4.3 Η διάταξη πελάτη-εξυπηρετητή

Η αύξηση των απαιτήσεων από το λογισμικό, της πολυπλοκότητας, αλλά και του όγκου των δεδομένων που διαχειρίζεται μια εφαρμογή, κατέστησαν τη μονολιθική διάταξη ανεπαρκή για την ικανοποίηση πολλών απαιτήσεων. Παράλληλα, η ανάπτυξη των συστημάτων

διαχείρισης σχεσιακών βάσεων δεδομένων, αλλά και των δικτύων, επέτρεψαν την εξέλιξη της μονολιθικής διάταξης σε αυτή του *πελάτη - εξυπηρετητή* (client.server).

Στο Σχήμα 4.9 φαίνεται το διάγραμμα διάταξης πελάτη.εξυπηρετητή. Οι εργασίες που σχετίζονται με τη διαχείριση δεδομένων ανατίθενται σε ένα ξεχωριστό τμήμα της εφαρμογής, το οποίο τρέχει συνήθως σε ένα αφιερωμένο στη διαχείριση δεδομένων υπολογιστικό σύστημα. Οι εργασίες παρουσίασης και επιχειρησιακής λογικής τρέχουν σε ένα άλλο τμήμα, το οποίο επικοινωνεί μέσω δικτύου με τον εξυπηρετητή ζητώντας του την παροχή σχετικών με δεδομένα υπηρεσιών.

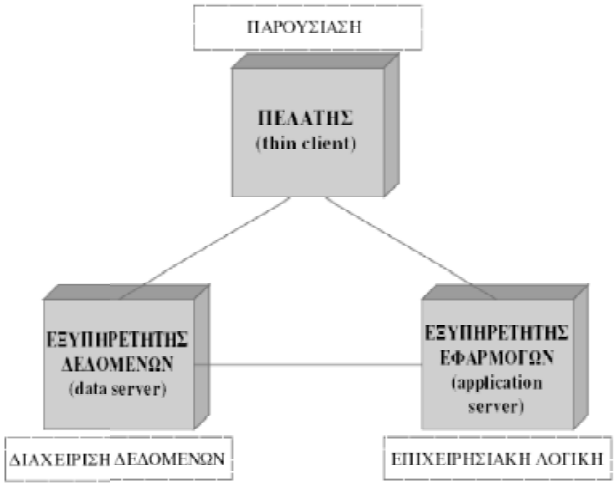
Η ιδέα, πρωτοποριακή για την εποχή της, έλυσε το πρόβλημα των ολοένα και μεγαλύτερων υπολογιστικών απαιτήσεων από τα γιγαντωμένα μονολιθικά συστήματα, οι οποίες μεγάλωναν μαζί με τον αριθμό των χρηστών αλλά και την πολυπλοκότητα των εργασιών που εκτελούσαν. Με τον καιρό, διάφορα προβλήματα της διάταξης αυτής αναδείχτηκαν. Το σημαντικότερο εντοπίζεται στην ανάγκη συντήρησης όλων των συστημάτων πελάτη (τα οποία μπορούσαν να είναι πολυάριθμα), καθώς συνέβαιναν οποιεσδήποτε μεταβολές στο επίπεδο της επιχειρησιακής λογικής. Επίσης, όσο μεγάλωνε η πολυπλοκότητα των λειτουργιών, τόσο περισσότερο τα συστήματα όπου έτρεχαν τα συστήματα πελάτη αποδεικνύονταν ανεπαρκή από πλευράς υπολογιστικής ισχύος.



ΣΧΗΜΑ 4.9 *Η διάταξη πελάτη.εξυπηρετητή*

4.4.4 Η τριμερής Διάταξη

Εμφανίστηκαν, λοιπόν, νέες εκδοχές της διάταξης πελάτη - εξυπηρετητή που διαχωρίζουν ακόμη περισσότερο τις «αρμοδιότητες» του λογισμικού. Ο πελάτης ελαφρύνεται και μένει μόνο με την ευθύνη της παρουσίασης, ενώ εμφανίζεται και ένας δεύτερος τύπος εξυπηρετητή, ο εξυπηρετητής εφαρμογών, ο οποίος κάνει τις εργασίες του επιπέδου της επιχειρησιακής λογικής. Η διάταξη αναφέρεται ως *τριμερής* (3.tier) και εικονίζεται στο Σχήμα 4.10.



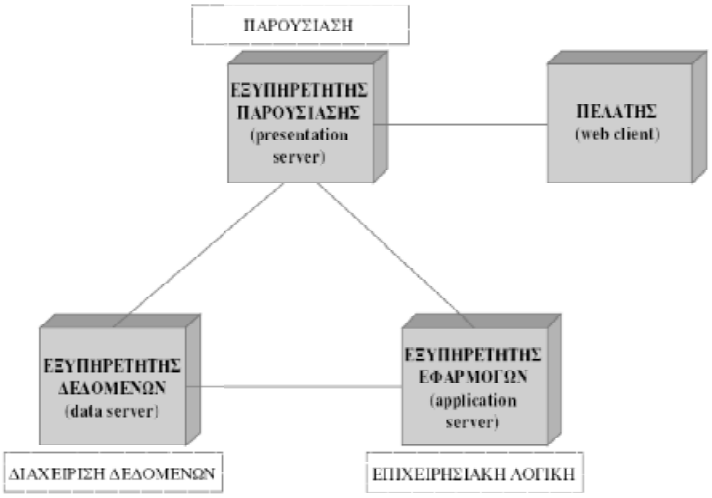
ΣΧΗΜΑ 4.10 *Η τριμερής διάταξη λογισμικού*

Στην περίπτωση αυτή, ο πελάτης χαρακτηρίζεται ως *ελαφρύς* (thin client), ακριβώς διότι κάνει λιγότερα πράγματα απ'ό,τι στην αρχική εκδοχή της διάταξης πελάτη - εξυπηρετητή. Τα συστήματα των εξυπηρετητών δεδομένων και εφαρμογών είναι συνήθως μεγάλα κεντρικά υπολογιστικά συστήματα. Τα προβλήματα της διάταξης πελάτη - εξυπηρετητή περιορίζονται, διότι οι απαιτήσεις συντήρησης των πελατών υφίστανται μόνο όταν συμβαίνουν μεταβολές στο επίπεδο της παρουσίασης. Πάντως, δεν παύουν να υπάρχουν.

4.4.5 Πολυμερής διάταξη

Το επόμενο αναμενόμενο βήμα είναι η αφαίρεση και της αρμοδιότητας της παρουσίασης από τον πελάτη και η ανάθεσή της σε έναν εξυπηρετητή παρουσίασης. Η ιδέα του εξυπηρετητή παρουσίασης γεννήθηκε με την εμφάνιση του *παγκόσμιου ιστού* (world wide web) και του Internet και έγινε δυνατή με την ανάπτυξη τεχνολογιών που επιτρέπουν την αλληλεπίδραση μεταξύ του ***web server*** και του συνδεδεμένου σε αυτόν *πελάτη* (browser). Τέτοιες τεχνολογίες είναι τα scripts, η Java, καθώς και το μοντέλο DCOM, και σήμερα είναι ήδη αρκετά διαδεδομένες. Η διάταξη ονομάστηκε *πολυμερής* (multi.tier) ή *βασισμένη στο* ***web*** (web based) και εμφανίζεται στο Σχήμα 4.11. Τα τμήματα της εφαρμογής λογισμικού διατάσσονται με τρόπο ώστε όλες οι εργασίες των τριών κατηγοριών (παρουσίασης, διαχείρισης δεδομένων και επιχειρησιακής λογικής) να εκτελούνται σε έναν ή περισσότερους για κάθε κατηγορία εξυπηρετητές. Ο εξυπηρετητής παρουσίασης δεν είναι παρά ένας εξυπηρετητής υπηρεσιών web (web server). Ο πελάτης δεν απαιτείται να διαθέτει κανένα

τμήμα της εφαρμογής, παρά μόνο τη δυνατότητα επικοινωνίας με τον web server, δηλαδή μια δικτυακή σύνδεση και ένα πρόγραμμα πλοήγησης στο web (browser).
Για το λόγο αυτό, ο πελάτης αναφέρεται και ως «web client». Τα προβλήματα ανάγκης συντήρησης των συστημάτων των πελατών εκμηδενίζονται, γεννιούνται όμως άλλα, αυτά της ταχύτητας και της ασφάλειας των δικτυακών συνδέσεων.

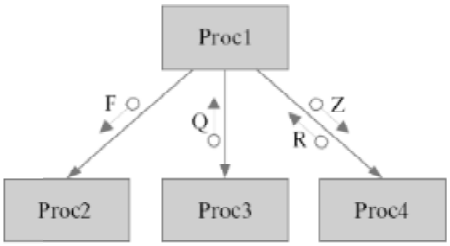


ΣΧΗΜΑ 4.11 *Μια πολυμερής διάταξη λογισμικού*

4.5 Αρχιτεκτονική σχεδίαση

Στην ενότητα αυτή θα περιγραφεί η διαδικασία κατασκευής του *αρχιτεκτονικού σχεδίου συστήματος* με βάση το διάγραμμα ροής δεδομένων, η οποία ακολουθείται στη δομημένη σχεδίαση.

Η μέθοδος της δομημένης σχεδίασης βασίζεται στα διαγράμματα ροής δεδομένων, από τα οποία, με κάποιο συστηματικό αλλά όχι αυτόματο τρόπο, παράγεται το αρχιτεκτονικό σχέδιο του λογισμικού. Το σχέδιο αυτό αποτυπώνεται με τη βοήθεια ενός διαγράμματος που ονομάζεται «*διάγραμμα δομής προγράμματος*», όπως αυτό που φαίνεται στο Σχήμα 4.12. Στο διάγραμμα δομής προγράμματος οι μονάδες λογισμικού παριστάνονται με ένα παραλληλόγραμμο. Η κλήση της μονάδας Β από τη μονάδα Α υποδηλώνεται με ένα βέλος που συνδέει την Α με τη Β. Το πέρασμα παραμέτρων προς αμφότερες υποδηλώνεται με ένα βέλος με έναν κύκλο στο άκρο της αρχής του, σημειωμένο σε διεύθυνση παράλληλη με τη διεύθυνση του βέλους που περιγράφει τη κλήση, το οποίο φέρει το όνομα της παραμέτρου.



ΣΧΗΜΑ 4.12 *Συμβολισμοί διαγραμμάτων δομής προγράμματος*

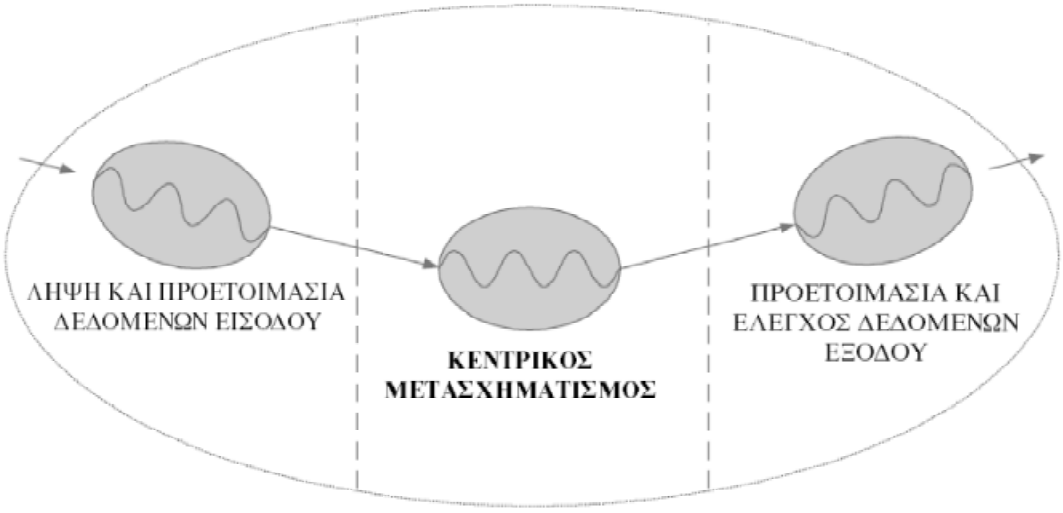
Η διαδικασία εκτελείται για καθένα από τα διαγράμματα ροής δεδομένων του συστήματος. Στα διαγράμματα αυτά εντοπίζονται δύο τύποι χαρακτηριστικών περιοχών, οι *κεντρικοί μετασχηματισμοί* και τα *κέντρα δοσοληψιών*. Όταν εντοπιστεί μια τέτοια περιοχή, δημιουργείται ή συμπληρώνεται ένα επίπεδο του διαγράμματος δομής προγράμματος και η διαδικασία επαναλαμβάνεται μέχρις ότου να εξαντληθεί το διάγραμμα ροής δεδομένων.

4.5.1 Κεντρικός μετασχηματισμός

Ως κεντρικός μετασχηματισμός[10] *ορίζεται μια περιοχή ενός διαγράμματος ροής δεδομένων, οι μετασχηματισμοί που ανήκουν στην οποία μπορεί να θεωρηθεί ότι μετατρέπουν δεδομένα εισόδου προκειμένου να δημιουργήσουν νέα δεδομένα εξόδου. Οι εργασίες που προηγούνται του κεντρικού μετασχηματισμού προσάγουν τα δεδομένα εισόδου σε αυτόν, ενώ οι εργασίες που έπονται αυτού καταναλώνουν τα δεδομένα εξόδου.*

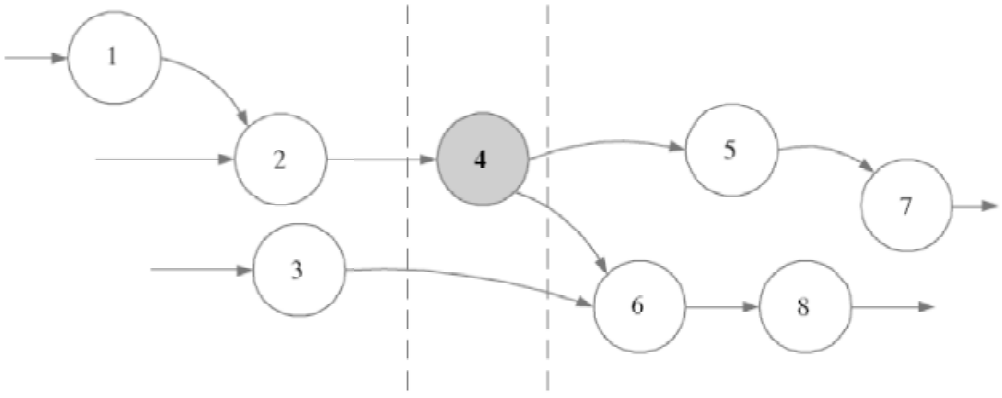
Στο Σχήμα 4.13 φαίνεται ένας κεντρικός μετασχηματισμός ως εντοπισμός χαρακτηριστικών περιοχών σε ένα υποσύνολο ενός διαγράμματος ροής δεδομένων. Στον κεντρικό μετασχηματισμό μπορεί να ανήκουν περισσότεροι του ενός μετασχηματισμοί και το ίδιο ισχύει για τα αριστερά και δεξιά τμήματα του διαγράμματος. Αυτό σημαίνει ότι ο κεντρικός μετασχηματισμός, η προετοιμασία δεδομένων εισόδου και η προετοιμασία δεδομένων εξόδου μπορούν να αντιστοιχούν σε σύνθετα τμήματα του διαγράμματος ροής δεδομένων και όχι σε απλούς μετασχηματισμούς.

[10] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **123.**



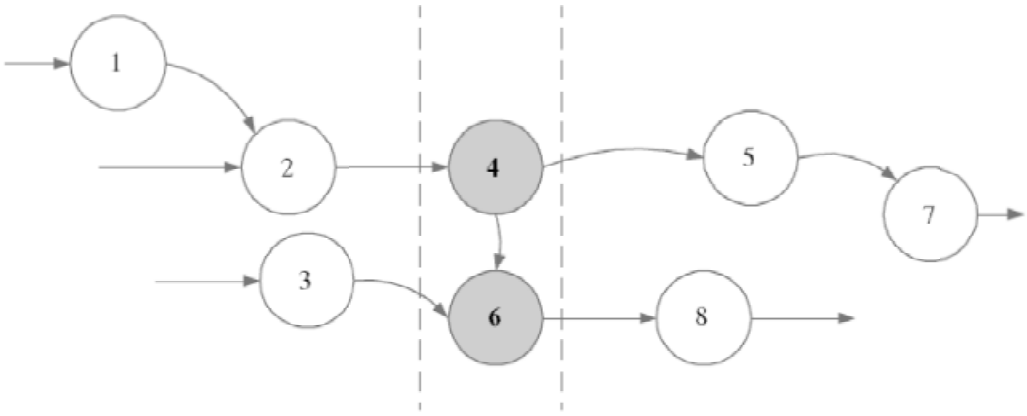
ΣΧΗΜΑ 4.13 *Η έννοια του κεντρικού μετασχηματισμού*

Δεν υπάρχει αυτόματος τρόπος για τον εντοπισμό των κεντρικών μετασχηματισμών. Η εμπειρία, ο αυτοσχεδιασμός και η διαίσθηση του μηχανικού λογισμικού έχουν και εδώ τον πρώτο λόγο. Επίσης, η επιλογή ενός κεντρικού μετασχηματισμού δεν είναι μοναδική. Αυτό σημαίνει ότι στο ίδιο διάγραμμα ροής δεδομένων δύο ή περισσότερες περιοχές μπορούν να χαρακτηριστούν ως κεντρικός μετασχηματισμός, χωρίς να είναι απαραίτητα λάθος ο ένας από τους δύο χαρακτηρισμούς. Η σύλληψη και η λεπτομέρεια του διαγράμματος ροής δεδομένων παίζουν, όπως είναι φανερό, καθοριστικό ρόλο στο Σητεία αυτό. Στο Σχήμα 4.14 φαίνεται ένα διάγραμμα ροής δεδομένων στο οποίο θεωρείται ότι οι μετασχηματισμοί 1, 2 και 3 λαμβάνουν και προετοιμάζουν τα δεδομένα εισόδου. Ο μετασχηματισμός 4 κάνει την κύρια δουλειά της δημιουργίας νέων δεδομένων και χαρακτηρίζεται ως κεντρικός μετασχηματισμός, ενώ οι μετασχηματισμοί 5, 6, 7 και 8 προετοιμάζουν τα δεδομένα εξόδου.



ΣΧΗΜΑ 4.14 *Ένα παράδειγμα εντοπισμού κεντρικού μετασχηματισμού*

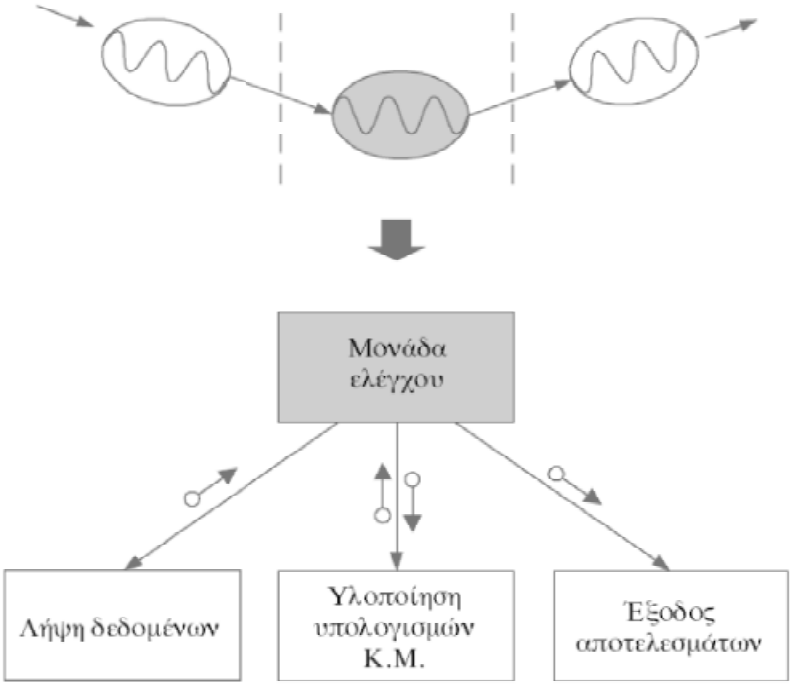
Μια διαφορετική εκδοχή επί του ιδίου διαγράμματος ροής δεδομένων φαίνεται στο Σχήμα 4.15. Η ιδέα είναι ότι και ο μετασχηματισμός 6 ανήκει στον κεντρικό μετασχηματισμό. Καμία από τις δύο εκδοχές δεν μπορεί να χαρακτηριστεί ως καλύτερη από την άλλη χωρίς να είναι γνωστό το συγκεκριμένο πρόβλημα και ο ορισμός εκάστου των μετασχηματισμών του διαγράμματος ροής δεδομένων.



ΣΧΗΜΑ 4.15 *Μια εναλλακτική επιλογή του κεντρικού μετασχηματισμού του προηγούμενου σχήματος*

Ένας κεντρικός μετασχηματισμός απεικονίζεται σε διάγραμμα δομής, όπως φαίνεται στο Σχήμα 4.16. Η μονάδα ελέγχου εκτέλεσης καλεί τις μονάδες που απαιτείται για να λάβει τα δεδομένα εισόδου, τις μονάδες που απαιτείται να εκτελέσουν τον κεντρικό μετασχηματισμό και τέλος, τις μονάδες στις οποίες θα διαθέσει τα δεδομένα εξόδου.

Στο διάγραμμα του Σχήματος 4.16 μπορούν να υπάρχουν περισσότερες από μια μονάδες λήψης και εξόδου αποτελεσμάτων, ανάλογα μετά πλήθος των εισερχόμενων και εξερχόμενων ροών στον κεντρικό μετασχηματισμό, αντίστοιχα. Το σχέδιο που αντιστοιχεί στον κεντρικό μετασχηματισμό περιέχει τόσες μονάδες όσοι και οι απλοί μετασχηματισμοί που περιλαμβάνονται στον κεντρικό μετασχηματισμό.



ΣΧΗΜΑ 4.16 *Απεικόνιση κεντρικού μετασχηματισμού σε διάγραμμα δομής*

Μια δεύτερη χαρακτηριστική περιοχή ενός διαγράμματος ροής δεδομένων είναι το *κέντρο δοσοληψιών*.

Κέντρο δοσοληψιών

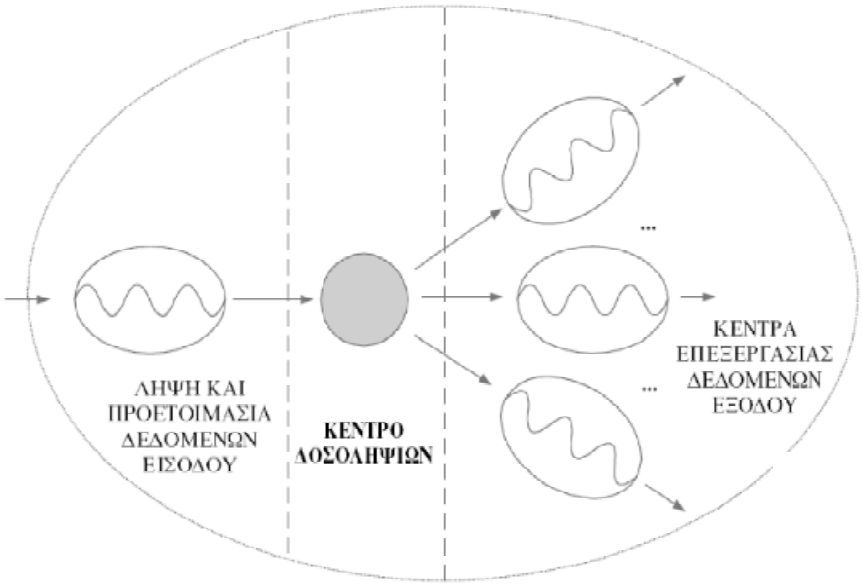
Ως κέντρο δοσοληψιών[11] *(transaction centre) ορίζεται ένας μετασχηματισμός του διαγράμματος ροής δεδομένων, ο οποίος δέχεται κάποια δεδομένα εισόδου και παράγει ένα σύνολο δεδομένων εξόδου ανάλογα με την τιμή των δεδομένων εισόδου.*

Αξίζει να σημειωθεί ότι, ενώ ένας κεντρικός μετασχηματισμός μπορεί να περιλαμβάνει περισσότερους του ενός απλούς μετασχηματισμούς του διαγράμματος ροής δεδομένων, ένα κέντρο δοσοληψιών περιλαμβάνει μόνο έναν. Ένα κέντρο δοσοληψιών φαίνεται στο Σχήμα 4.17.

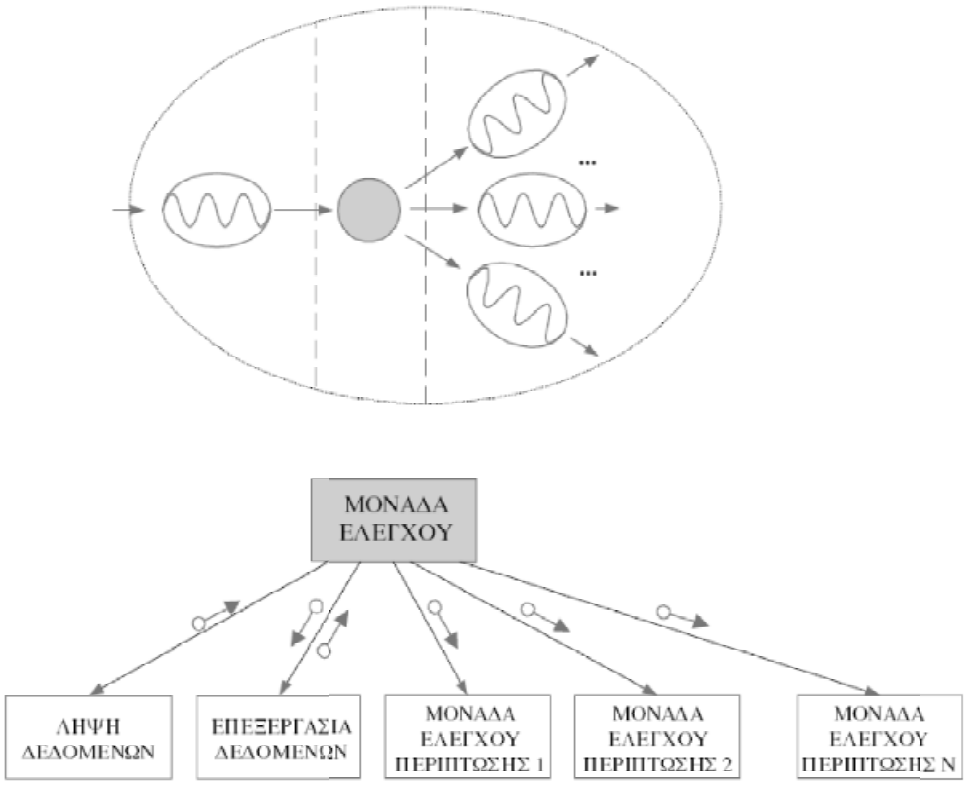
Χαρακτηριστική περίπτωση κέντρου δοσοληψιών είναι ένας μετασχηματισμός ελέγχου ροής προγράμματος, όπου, ανάλογα με την επιλογή του χρήστη, η ροή μεταφέρεται σε διαφορετικούς μετασχηματισμούς, όπως η περίπτωση ενός μενού. Ένα κέντρο δοσοληψιών απεικονίζεται σε διάγραμμα δομής, όπως στο Σχήμα 4.18. Η μονάδα ελέγχου του κέντρου δοσοληψιών καλεί δύο μονάδες για τη λήψη και επεξεργασία, αντίστοιχα, των δεδομένων

[11] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **126.**

εισόδου, καθώς και τόσες μονάδες όσες είναι οι περιπτώσεις των δεδομένων εξόδου για τη μεταφορά του ελέγχου.



ΣΧΗΜΑ 4.17 *Η έννοια του κέντρου δοσοληψιών*



ΣΧΗΜΑ 4.18 *Απεικόνιση κέντρου δοσοληψιών σε διάγραμμα δομής*

4.5.2 Βήματα κατασκευής διαγραμμάτων δομής

Η μετάβαση από το διάγραμμα ροής δεδομένων στο διάγραμμα δομής γίνεται με διαδοχική επανάληψη κάποιων βημάτων, μέχρι να έχουν προσδιοριστεί μονάδες για όλους τους μετασχηματισμούς που περιέχονται στα διαγράμματα ροής δεδομένων της εφαρμογής. Τα βήματα αυτά είναι:

1. **Εντοπισμός κεντρικού μετασχηματισμού.** Για κάθε τμήμα του διαγράμματος ροής δεδομένων εντοπίζεται ο κεντρικός μετασχηματισμός και διακρίνονται οι μετασχηματισμοί εισόδου και εξόδου σε αυτόν.

2. **Απεικόνιση του κεντρικού μετασχηματισμού σε διάγραμμα δομής**. Δημιουργείται ένα επίπεδο του διαγράμματος δομής που αντιστοιχεί στον κεντρικό μετασχηματισμό.

3. **Παραγοντοποίηση (factoring).** Για το αριστερό και το δεξί τμήμα του κεντρικού μετασχηματισμού (είσοδοι και έξοδοι) δημιουργούνται τα διαγράμματα δομής, που αντιστοιχούν στους μετασχηματισμούς που περιέχονται σε αυτά. Κάθε τέτοιος μετασχηματισμός απεικονίζεται σε μια μονάδα ελέγχου και σε δύο άλλες μονάδες. Από αυτές, η πρώτη λαμβάνει τα δεδομένα εισόδου, ενώ η δεύτερη πραγματοποιεί τη μετατροπή.

Η μονάδα ελέγχου διαθέτει τα δεδομένα στο παραπάνω επίπεδο. Κατά την παραγοντοποίηση ενδεχομένως να αναγνωριστούν και άλλοι κεντρικοί μετασχηματισμοί ή κέντρα δοσοληψιών, τα οποία αντιμετωπίζονται όπως περιγράφηκε. Η διαδικασία επαναλαμβάνεται μέχρις ότου να φτάσουμε στις πηγές και τους αποδέκτες των δεδομένων, δηλαδή το χρήστη, εξωτερικά συστήματα, συσκευές ή αρχεία.

4. **Συνένωση.** Η τελευταία εργασία που πρέπει να γίνει είναι αυτή της συνένωσης. Για τις περιπτώσεις όπου τα δεδομένα δε λαμβάνονται από εξωτερική πηγή ή δεν καταλήγουν σε εξωτερικό αποδέκτη, η μονάδα που τα εισάγει στον κεντρικό μετασχηματισμό αντικαθίσταται από τη μονάδα ελέγχου του μετασχηματισμού που τα παρέχει.

Κατά τη διαδικασία αυτή ενδεχομένως να διαπιστωθεί ότι είναι χρήσιμο να πραγματοποιηθούν τροποποιήσεις στα διαγράμματα ροής δεδομένων, γεγονός που συνιστά οπισθοδρόμηση στη διαδικασία ανάπτυξης λογισμικού, η οποία όμως είναι χρήσιμο να πραγματοποιηθεί.

4.6 Λεπτομερής σχεδίαση μονάδων

Έχοντας διαθέσιμο το αρχιτεκτονικό σχέδιο, είναι δυνατή η λεπτομερής σχεδίαση των μονάδων λογισμικού που περιλαμβάνονται σε αυτό. Κατά τη λεπτομερή σχεδίαση θα προσδιοριστεί η εσωτερική δομή κάθε μονάδας, δηλαδή θα δοθεί μια περιγραφή του πηγαίου κώδικα. Μέχρι το Σητεία αυτό είναι γνωστή μόνο η ονομασία κάθε μονάδας και οι παράμετροι εισόδου και εξόδου αυτής. Στοιχεία που σχετίζονται με την περιγραφή της συμπεριφοράς της είναι διαθέσιμα από τη φάση της προδιαγραφής των απαιτήσεων. Με το υλικό αυτό ο σχεδιαστής λογισμικού καλείται να κατασκευάσει το λεπτομερές σχέδιο.

Σε περίπτωση που το διάγραμμα δομής προγράμματος είναι επαρκώς λεπτομερές, καθεμία μονάδα του διαγράμματος αντιστοιχεί σε μια μονάδα κώδικα. Αν όμως αυτό δεν ισχύει, είναι ανάγκη κάθε μονάδα του διαγράμματος δομής να απεικονιστεί, ενδεχομένως, σε περισσότερες από μία μονάδες λογισμικού. Η εργασία αυτή γίνεται αναλύοντας κάθε πρόβλημα σχεδίασης σε διαδοχικά βήματα. Σε κάθε βήμα δημιουργούμε ένα σύνολο από μικρότερα προβλήματα σχεδίασης, καθένα εκ των οποίων επιλύουμε και πάλι αναλύοντάς το κοκ., μέχρις ότου φτάσουμε σε απλές δομικές μονάδες λογισμικού, όπως οι «διαδικασίες» και οι «συναρτήσεις». Η αντιμετώπιση αυτή ονομάζεται *εκλέπτυνση* σε διαδοχικά βή*ματα* (stepwise refinement) ή *συναρτησιακή αποσύνθεση* (functional decomposition).

Υπάρχουν αρκετοί τρόποι για να περιγράφονται τα αποτελέσματα της λεπτομερούς σχεδίασης. Ο επικρατέστερος είναι με χρήση μιας *γλώσσας σχεδίασης προγράμματος* (PDL:

program description language). Μια γλώσσα σχεδίασης προγράμματος μοιάζει με γλώσσα προγραμματισμού, χωρίς να έχει την αυστηρότητα στη σύνταξη και τη γραμματική που χαρακτηρίζει τις γλώσσες προγραμματισμού. Σκοπός της είναι να παρέχει μια εικόνα του λογισμικού η οποία να μπορεί να υλοποιηθεί εύκολα σε κάποια γλώσσα προγραμματισμού. Οι γλώσσες σχεδίασης προγράμματος περιέχουν τις βασικές δομές ελέγχου που απαντώνται στις γλώσσες προγραμματισμού. Μια απλή τέτοια γλώσσα που ομοιάζει με την Pascal και στην οποία θα βασιστούν τα παραδείγματα που θα ακολουθήσουν περιγράφεται στο Σχήμα 4.18. Οι γλώσσες σχεδίασης προγράμματος αναφέρονται συχνά και ως «ψευ-δοκώδικας».

Απλές εκφράσεις	Επαναληπτική εκτέλεση

/*σχόλιο*/
Μεταβλητή:-τιμή /*ανάθεση*/
Φραστική περιγραφή ενέργειας
+-*/^ /*μαθηματικές εκφράσεις*/
	FOR Μτβλ.FROM τιμή1 TO τιμή2 STEP τιμή3 DO
	(Ενέργειες)
	END FOR

Εκτέλεση με επιλογή περίπτωσης	**Εκτέλεση υπο συνθήκη**

CASE Έκφραση OF
(τιμή1) : (ενέργειες)
(τιμή 2) : (ενέργειες)
…
(τιμή N) : (ενέργειες)
OTERWISE
(Εντολές αν η έκφραση έχει άλλη τιμή)
END CASE
	IF Συνθήκη THEN
	(ενέργειες αν η συνθήκη είναι αληθής)
	ELISE
	(εντολή αν η συνθήκη είναι ψευδής)
	END IF

Επαναληπτική εκτέλεση με συνθήκη(1)	**Επαναληπτική εκτέλεση με συνθήκη(2)**

REPEAT
(ενέργειες)
UNTIL συνθήκη
	WHILE συνθήκη DO
	(ενέργειες)
	END WHILE

Ορισμός διαδικασιών	**Ορισμός συναρτήσεων**
PROCEDURE Όνομα(παράμετρος: IN/OUT,..) GLOBAL VAR Όνομα 1,όνομα 2,… LOCAL GLOBAL Όνομα1,όνομα 2,.. … (ενέργειες) … CALL όνομα διαδικασίας(παραμ1,παραμ2,..) … (ενέργειες) … END PROCEDURE	FUNCTION Όνομα συναρτησης(παράμετρος,..) GLOBAL VAR Όνομα 1,όνομα 2,… LOCAL GLOBAL Όνομα1,όνομα 2,.. … (ενέργειες) … όνομα συνάρτησης:=τιμή … (ενέργειες) … END FUNCTION

ΣΧΗΜΑ 4.19 *Μία απλή γλώσσα περιγραφής προγράμματος*

Προκειμένου να κατασκευαστεί το ομοίωμα αυτό (λεπτομερές σχέδιο), ο σχεδιαστής έρχεται αντιμέτωπος με αρκετά ζητήματα, όπως τα ακόλουθα:

- Ποιο είναι το καλύτερο από τα σχέδια τα οποία μπορεί να συλλάβει;

- Ποιος είναι ο προσφορότερος τρόπος για την περιγραφή του λεπτομερούς σχεδίου μονάδων;
- Πώς σχετίζεται το σχέδιο με τη γλώσσα προγραμματισμού και το περιβάλλον υλοποίησης γενικότερα;
- Πώς μπορεί ένα λεπτομερές σχέδιο μονάδων να διατηρείται επίκαιρο;

Οι απαντήσεις στα ερωτήματα αυτά δεν είναι εύκολες και εξαρτώνται από πλήθος παραμέτρων και υποκειμενικών προσεγγίσεων. Στο Κεφάλαιο 5 αντιμετωπίζεται το πρόβλημα της συγγραφής πηγαίου κώδικα και αναφέρονται οι διαφορετικές πλευρές του προβλήματος κατασκευής του «καλύτερου» πηγαίου κώδικα. Η χρησιμότητα της ύπαρξης λεπτομερούς σχεδίου με τη βοήθεια ψευδοκώδικα έχει αμφισβητηθεί. Το βασικότερο επιχείρημα της αμφισβήτησης είναι ότι ο προγραμματιστής πρέπει να έχει την ελευθερία να αυτοσχεδιάσει. Αντίλογος εδώ μπορεί να εντοπιστεί σε ζητήματα όπως η ποιότητα, η αναγνωσιμότητα και η ευκολία συντήρησης του κώδικα, τα οποία θα μας απασχολήσουν στο επόμενο κεφάλαιο.

Σε πολλές περιπτώσεις δεν μπορεί να υποστηριχτεί με αξιώσεις η ανεξαρτησία του λεπτομερούς σχεδίου μονάδων από τη γλώσσα προγραμματισμού. Στην πράξη, ο σχεδιαστής έχει πάντα στο μυαλό του τη γλώσσα προγραμματισμού με την οποία θα υλοποιηθεί το σχέδιό του, και αυτό δεν είναι κακό εφόσον διευκολύνει τη μεταφορά του σχεδίου σε πηγαίο κώδικα. Το θέμα της ανεξαρτησίας σχεδίου λογισμικού και γλώσσας προγραμματισμού έχει αποτελέσει επί μακρόν αντικείμενο συζητήσεων μεταξύ των ερευνητών και των κατασκευαστών λογισμικού. Η λύση που φαίνεται να επικρατεί είναι η υπό παραδοχές και περιορισμούς ανεξαρτησία του σχεδίου από τον κώδικα, με τη βοήθεια εργαλείων υποστήριξης της ανάπτυξης λογισμικού (CASE).

Στα εργαλεία CASE βρίσκεται και η περισσότερο προσγειωμένη απάντηση στο τελευταίο ερώτημα, αυτό της διατήρησης του σχεδίου του λογισμικού σε επίκαιρη μορφή. Όπως έχει ήδη αναφερθεί, το λογισμικό δεν παραμένει στατικό και υπόκειται σε πολλές αλλαγές κατά τον κύκλο ζωής του, είτε για τη διόρθωση σφαλμάτων είτε για τη μεταβολή των χαρακτηριστικών του. Αν και συνήθως οι αλλαγές πρέπει να γίνονται πρώτα στο σχέδιο και κατόπιν στον πηγαίο κώδικα, στην πράξη αυτό δε συμβαίνει και οι αλλαγές συχνά πραγματοποιούνται κατευθείαν στον κώδικα. Έτσι, ένα σχέδιο που έχει κατασκευαστεί με το χέρι συχνά καταλήγει άχρηστο εφόσον δεν ανταποκρίνεται στην πραγματικότητα. Κάτι τέτοιο είναι λιγότερο πιθανό να συμβεί, όταν ο κατασκευαστής χρησιμοποιεί κάποιο

εργαλείο, το οποίο είτε διευκολύνει είτε υποχρεώνει τον κατασκευαστή να διατηρεί ενημερωμένο σχέδιο.

4.7 Σχεδίαση δεδομένων

Σκοπός της σχεδίασης δεδομένων είναι ο προσδιορισμός των δομών δεδομένων, η ύπαρξη των οποίων εντοπίστηκε κατά τη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό. Ο προσδιορισμός αυτός πρέπει να είναι επαρκής για την απεικόνιση των δεδομένων σε ένα περιβάλλον υλοποίησης. Η εργασία της σχεδίασης δεδομένων είναι στενά συνυφασμένη με την εργασία κατάστρωσης των υπόλοιπων τμημάτων του σχεδίου του λογισμικού, στις οποίες αναφερθήκαμε. Συχνά, η εργασία σχεδίασης δεδομένων υποτιμάται από τους κατασκευαστές σε σχέση με την υπόλοιπη σχεδίαση.

Η σχεδίαση δεδομένων μπορεί να μελετηθεί αυτοτελώς σε σημαντικό βάθος και πλάτος, πράγμα που γίνεται στις Θ.Ε. «Βάσεις δεδομένων» και «Γλώσσες προγραμματισμού 2». Χωρίς κάτι τέτοιο να αποτελεί αντικείμενο του παρόντος βιβλίου, παραθέτουμε τις εργασίες που πρέπει οπωσδήποτε να εκτελούνται κατά τη λεπτομερή σχεδίαση δεδομένων:

- Τροποποίηση του μοντέλου οντοτήτων - συσχετίσεων, την αρχική μορφή του οποίου κατασκευάσαμε κατά τη συγγραφή των προδιαγραφών με σκοπό αυτό να περιέχει ακριβώς όση πληροφορία απαιτείται, κατανεμημένη σωστά μεταξύ των οντοτήτων. Η διαδικασία αυτή ονομάζεται «κανονικοποίηση».
- Καθορισμός των πεδίων που περιέχει κάθε πίνακας στο επίπεδο της φυσικής αποθήκευσης δεδομένων.
- Καθορισμός των εναλλακτικών μορφών εμφάνισης της οργάνωσης των δεδομένων πάνω από το φυσικό επίπεδο (ορισμό views).
- Βελτιστοποιήσεις με σκοπό τη βελτιστοποίηση των επιδόσεων και γενικά, την επίτευξη κριτηρίων επάρκειας.

Η εργασία απεικόνισης οντοτήτων του πεδίου της εφαρμογής με τη βοήθεια ενός σχήματος δεδομένων είναι από τις πιο ενδιαφέρουσες, δημιουργικές και σημαντικές εργασίες κατά την ανάπτυξη του λογισμικού. Μερικές αρχές που είναι χρήσιμο να ακολουθούνται κατά τη διαδικασία αυτή είναι οι ακόλουθες:

- Η σχεδίαση δεδομένων δεν πρέπει να υποτιμάται. Η απόδοση χρόνου και σημασίας σε αυτή και η πειθαρχημένη εφαρμογή αρχών σχεδίασης ανάλογων με αυτές που χρησιμοποιούνται για την κατασκευή του αρχιτεκτονικού σχεδίου του λογισμικού ανταποδίδουν πάντα το χρόνο που καταναλώνουν.
- Θα πρέπει να εντοπιστούν όλες οι δομές δεδομένων πάνω στα οποία επιδρούν οι μονάδες λογισμικού. Μετά από τη σχεδίαση δε θα πρέπει να υπάρχουν δεδομένα των οποίων ο προσδιορισμός να αφήνεται για το μέλλον.
- Το λεξικό δεδομένων, θα πρέπει να τηρείται ενημερωμένο.
- Οι αποφάσεις σχεδίασης δεδομένων που σχετίζονται με κατασκευαστικές λεπτομέρειες θα πρέπει να λαμβάνονται όσο αργότερα γίνεται. Η επιδίωξη αποφυγής της εξάρτησης των αποτελεσμάτων της σχεδίασης δεδομένων από το περιβάλλον υλοποίησης (όπως ένα σύστημα διαχείρισης βάσεων δεδομένων) συνήθως απαιτεί σημαντική προσπάθεια.
- Δεν είναι καλή ιδέα οποιαδήποτε μονάδα λογισμικού να μπορεί να διαβάσει και να μεταβάλει δεδομένα. Είναι σκόπιμο κατά τη σχεδίαση να καθορίζεται ποιες μονάδες λογισμικού επιτρέπεται να επιδρούν κατευθείαν στα δεδομένα. Με τον τρόπο αυτό μειώνονται οι πιθανότητες παρενεργειών στο λογισμικό καθώς πραγματοποιούνται μεταβολές στην εσωτερική δομή των δεδομένων.
- Θα πρέπει να διερευνάται η δυνατότητα χρήσης έτοιμων και δοκιμασμένων δομών δεδομένων από βιβλιοθήκες. Αν και η γενικευμένη χρήση βιβλιοθηκών συστατικών λογισμικού δεν έχει γίνει μέχρι σήμερα δυνατή, η εφαρμογή της ιδέας σε μικρή κλίμακα μέσα στην εμβέλεια ενός κατασκευαστή λογισμικού συχνά αποδίδει.

Το αποτέλεσμα της σχεδίασης δεδομένων είναι το τελικό διάγραμμα οντοτήτων συσχετίσεων.

ΚΕΦΑΛΑΙΟ 5 : Παραγωγή πηγαίου κώδικα

5.1 Από την σχεδίαση στην κωδικοποίηση

Η κατασκευή του πηγαίου κώδικα (κωδικοποίηση) είναι το τελευταίο μέρος της ανάπτυξης του λογισμικού το οποίο γίνεται, τουλάχιστον στο μεγαλύτερο μέρος του, από τον άνθρωπο. Με τη μετάβαση από τη σχεδίαση στην κωδικοποίηση ασχολείται η ενότητα αυτή.

5.1.1 Λογισμικό χωρίς σφάλματα

Η μετάβαση από τη λεπτομερή σχεδίαση στον πηγαίο κώδικα πρέπει να γίνεται με τρόπο που να παράγεται λογισμικό χωρίς σφάλματα. Η τελευταία έννοια γίνεται συχνά Σητεία αναφοράς από πολλούς εμπλεκόμενους στην ανάπτυξη του λογισμικού και είναι σκόπιμο να οριστεί ο τρόπος με τον οποίο θα τη χρησιμοποιούμε στη συνέχεια.

Λογισμικό χωρίς σφάλματα

Είναι το λογισμικό[12] *που πληροί τις προδιαγραφές βάσει των οποίων κατασκευάστηκε, χωρίς να παράγει σφάλματα κατά την εκτέλεσή του (runtime), δηλαδή το λογισμικό που κάνει ακριβώς αυτό για το οποίο προορίζεται και το κάνει σωστά.*

Όπως έχει ήδη αναφερθεί, οι προδιαγραφές αυτές μπορεί να είναι εσφαλμένες, ανεπίκαιρες και γενικά, να μην αντικατοπτρίζουν τις πραγματικές ανάγκες του χρήστη. Αυτό είναι ένα πρόβλημα που γεννιέται και οφείλει να αντιμετωπίζεται σε προηγούμενες φάσεις της ανάπτυξης και όχι στην κωδικοποίηση. Το λογισμικό χωρίς σφάλματα δε συμπεριφέρεται απαραίτητα κατά τον τρόπο που αναμένουν οι χρήστες, ακριβώς επειδή ενδέχεται να μην είναι ακριβείς οι προδιαγραφές.

Λόγοι για τους οποίους μπορεί να συμβαίνει αυτό έχουν αναφερθεί σε αρκετά σημεία του βιβλίου που κρατάτε. Οι προδιαγραφές μπορεί να μεταβληθούν κατά την ανάπτυξη του λογισμικού, και τη στιγμή που φτάνουμε στη συγγραφή του πηγαίου κώδικα να μην ισχύουν κάποιες από αυτές. Όπως χαρακτηριστικά αναφέρεται και στην Ενότητα 1.2, *«η ανάπτυξη του*

[12] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη, Τεχνολογία Λογισμικού Ι, σελ **147.**

λογισμικού ομοιάζει με σκόπευση κινούμενου στόχου, από κινούμενο έδαφος και με όπλο που συνεχώς αλλάζει τη συμπεριφορά του».

Είναι ο κόσμος που μεταβάλλεται και η διαλεκτική αλληλεπίδραση όλων των συστημάτων του, που δημιουργεί συνεχώς, ακόμα και πριν ολοκληρωθεί η ανάπτυξη μιας εφαρμογής, αιτίες τροποποιήσεων στο λογισμικό. Σε αυτό προστίθενται οι εγγενείς και πραγματικές δυσκολίες στην περιγραφή των απαιτήσεων από το λογισμικό και τελικά, μπορεί να καθίστανται ανακριβείς, εσφαλμένες ή, τέλος πάντων, ανεπίκαιρες κάποιες από τις προδιαγραφές του λογισμικού κατά την έναρξη της συγγραφής του πηγαίου κώδικα. Εδώ βρίσκεται και η πρόκληση που πρέπει να αντιμετωπίσει ο μηχανικός λογισμικού. Η Τεχνολογία Λογισμικού δε λύνει από μόνη της όλα τα προβλήματα, ούτε μπορεί να υποσχεθεί κάτι τέτοιο. Μπορεί όμως να αποτελέσει ένα χρήσιμο επιστημονικό εργαλείο για την αντιμετώπιση του προβλήματος. Σήμερα είναι το μόνο τέτοιο εργαλείο που διαθέτουμε.

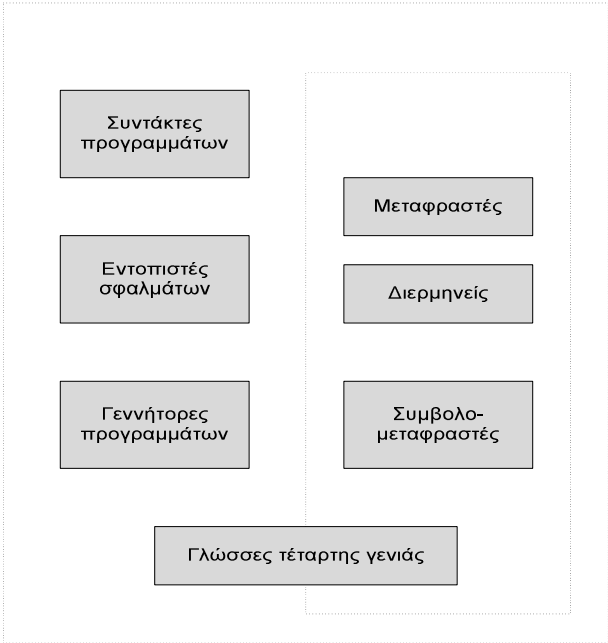
Η ανάπτυξη λογισμικού χωρίς σφάλματα είναι, λοιπόν, ιδιαίτερα δύσκολη και στην πράξη μάλλον ανέφικτη. Η εμφάνιση των σύγχρονων γλωσσών και περιβαλλόντων προγραμματισμού, που διευκολύνουν την ανάπτυξη σύνθετου και καλά δομημένου κώδικα, καθώς και η βελτίωση των χρησιμοποιούμενων προγραμματιστικών τεχνικών οδηγούν, γενικά, σε βελτίωση της ποιότητας του λογισμικού.

5.1.2 Εργαλεια κωδικοποίησης

Ένα μεγάλο πλήθος εργαλείων έχει κατασκευαστεί για τη διευκόλυνση των προγραμματιστών στη φάση της κωδικοποίησης. Τα πρωταρχικά εργαλεία στη φάση αυτή είναι, φυσικά, οι γλώσσες προγραμματισμού και γύρω από αυτές περιστρέφεται ένας μεγάλος αριθμός επιμέρους εργαλείων. Στο Σχήμα 5.1 γίνεται μια κατηγοριοποίηση των κυριότερων εργαλείων που είναι στη διάθεση των προγραμματιστών κατά τη φάση της κωδικοποίησης, οι οποίες κατηγορίες επεξηγούνται στη συνέχεια.

Συντάκτες προγραμμάτων (program editors): Είναι εξειδικευμένα προγράμματα επεξεργασίας κειμένου που αποσκοπούν στη διευκόλυνση της πρωτογενούς εργασίας συγγραφής κώδικα. Αρκετοί από τους σύγχρονους συντάκτες προγραμμάτων υποστηρίζουν έλεγχο της σύνταξης σε μία ή περισσότερες γλώσσες προγραμματισμού και λειτουργίες όπως χρωματική σύνταξη (syntax highlighting, δηλαδή χρωματισμό των δεσμευμένων λέξεων, των σχολίων, των μεταβλητών κτλ. με διαφορετικά χρώματα), αυτόματη στοίχιση, αρίθμηση γραμμών κ.ά.

Συστήματα υποστήριξης λογισμικού ολοκληρωμένα περιβάλλοντα προγραμματισμού



ΣΧΗΜΑ 5.1 *Εργαλεία κωδικοποίησης*

Μεταφραστικά περιβάλλοντα γλωσσών προγραμματισμού (programming language implementations) και παρεμφερή εργαλεία:

Συνήθως περιέχουν ένα ή περισσότερα από τα παρακάτω είδη προγραμμάτων, καθώς και άλλα βοηθητικά προγράμματα. Αξίζει να σημειωθεί ότι, σήμερα, τέτοια εργαλεία υπάρχουν ως ανεξάρτητα προγράμματα μόνο σε παλαιά ή ειδικής χρήσης υπολογιστικά συστήματα. Τα σύγχρονα μεταφραστικά περιβάλλοντα προγραμματισμού περιέχουν διάφορα εργαλεία κάτω από ένα ενιαίο κέλυφος, όπως αναφέρεται παρακάτω.

- *Μεταγλωττιστές* (compilers): Μεταφράζουν πηγαίο κώδικα γραμμένο σε μια γλώσσα προγραμματισμού υψηλού επιπέδου σε εκτελέσιμο κώδικα μηχανής.
- *Διερμηνείς* (interpreters): Εκτελούν γραμμή προς γραμμή πηγαίο κώδικα γραμμένο σε μια γλώσσα προγραμματισμού υψηλού επιπέδου.
- *Συμβολομεταφραστές* (assemblers): Μεταφράζουν πηγαίο κώδικα γραμμένο σε μια συμβολική γλώσσα μηχανής (assembly language) σε εκτελέσιμο κώδικα μηχανής.

Εντοπιστές σφαλμάτων (debuggers): Βοηθούν στον εντοπισμό σφαλμάτων κατά την εκτέλεση των προγραμμάτων. Υποστηρίζουν λειτουργίες όπως η εκτέλεση βήμα προς βήμα, η εμφάνιση τιμών μεταβλητών, η εμφάνιση της στοίβας εκτέλεσης και ο καθορισμός σημείων και συνθηκών διακοπής στην εκτέλεση του κώδικα.

Γεννήτορες προγραμμάτων (program generators): Πρόκειται για προγράμματα που δέχονται ως είσοδο τις προδιαγραφές του κώδικα γραμμένες σε κάποια γλώσσα υψηλού επιπέδου και παράγουν πηγαίο κώδικα γραμμένο σε κάποια γλώσσα προγραμματισμού που να πληροί τις δοθείσες προδιαγραφές. Οι γεννήτορες προγραμμάτων δεν είναι γενικοί και συνήθως εξειδικεύονται σε στενές θεματικές κατηγορίες προβλημάτων, όπως, για παράδειγμα, η κατασκευή προγραμμάτων που υλοποιούν τη διαπροσωπεία ανθρώπου - υπολογιστή, η κατασκευή λεκτικών και συντακτικών αναλυτών, προγραμμάτων προσομοίωσης κ.ά.

Συστήματα υποστήριξης λογισμικού (software support systems) και **ολοκληρωμένα περιβάλλοντα προγραμματισμού** (integrated programming environments): Αποτελούν προσπάθειες συγκέντρωσης και ενοποίησης διαφόρων εργαλείων προγραμματισμού που συνεργάζονται μεταξύ τους. Τα ολοκληρωμένα αυτά εργαλεία συνήθως περιλαμβάνουν ένα συνδυασμό όλων των χρήσιμων για τον προγραμματιστή εργαλείων, κοινός παρονομαστής των οποίων είναι συνήθως μια γλώσσα προγραμματισμού. Τα περισσότερα σύγχρονα περιβάλλοντα υλοποίησης γλωσσών προγραμματισμού ανήκουν σε αυτή την κατηγορία.

5.2 Επιθυμητά χαρακτηριστικά πηγαίου κώδικα

Στην ενότητα αυτή θα αναφερθούν τα χαρακτηριστικά του πηγαίου κώδικα που είναι επιθυμητά προκειμένου να εξασφαλίζεται η ποιότητα του λογισμικού. Κατά την κωδικοποίηση δεν αρκεί να κατασκευάζεται κάποιος οποιοσδήποτε πηγαίος κώδικας, αλλά είναι σημαντικό να τηρούνται τα απαραίτητα μέτρα, ώστε ο πηγαίος κώδικας να διαθέτει αυτά τα επιθυμητά χαρακτηριστικά.

5.2.1 Επάρκεια

Το βασικότερο επιθυμητό χαρακτηριστικό του πηγαίου κώδικα είναι η *επάρκεια* (efficiency). Ο κώδικας χαρακτηρίζεται επαρκής, όταν το σύστημα λογισμικού λειτουργεί σωστά και χωρίς σφάλματα, βάσει των απαιτήσεων που έχουν καθοριστεί, κάνει σωστή χρήση των διατιθέμενων πόρων και είναι διαθέσιμο όποτε απαιτείται. Η επάρκεια του πηγαίου κώδικα εξασφαλίζεται κυρίως κατά τη φάση ελέγχου του λογισμικού.

5.2.2 Επίδοση

Σε πολλά συστήματα λογισμικού καθοριστικό ρόλο κατέχει η *επίδοση* (performance). Στη μέτρηση της επίδοσης λαμβάνονται συνήθως υπόψη τα ακόλουθα χαρακτηριστικά:

- Η ταχύτητα εκτέλεσης συγκεκριμένων λειτουργιών του συστήματος λογισμικού.
- Οι απαιτήσεις του συστήματος σε πόρους, όπως μνήμη και χωρητικότητα δίσκου.

Όπως είναι φυσικό, διαφορετικά κομμάτια πηγαίου κώδικα που ικανοποιούν τις ίδιες λειτουργικές προδιαγραφές είναι δυνατό να διαφέρουν πολύ στην επίδοση, ακόμα κι αν υλοποιούν τον ίδιο αλγόριθμο.

Η επιλογή της κατάλληλης γλώσσας προγραμματισμού αλλά και η χρήση κατάλληλων προγραμματιστικών τεχνικών οδηγούν σε βελτίωση της επίδοσης του τελικού συστήματος λογισμικού.

5.2.3 Αναγνωσιμότητα

Στη διαδικασία ανάπτυξης λογισμικού συμβαίνει πολύ συχνά να χρειαστεί ένας προγραμματιστής να διαβάσει και να κατανοήσει τον πηγαίο κώδικα που έχει γράψει ένας άλλος προγραμματιστής. Συμβαίνει επίσης αρκετά συχνά να δυσκολεύεται ένας προγραμματιστής να καταλάβει κώδικα που έχει γράψει ο ίδιος, ύστερα από λίγο καιρό. Για τους δυο αυτούς λόγους, αναγκαίο χαρακτηριστικό του πηγαίου κώδικα είναι η *αναγνωσιμότητα* (readability), δηλαδή η ευκολία κατανόησης του πηγαίου κώδικα αποκλειστικά μέσω της ανάγνωσής του ως κειμένου. Ακολουθούν ορισμένες συμβουλές για τη συγγραφή περισσότερο αναγνώσιμου κώδικα, ανεξαρτήτως γλώσσας προγραμματισμού.

- **Απλότητα.** Είναι καλό οι προγραμματιστές να αποφεύγουν τη περιττή πολυπλοκότητα στη συγγραφή του πηγαίου κώδικα, όταν αυτό δεν είναι αναγκαίο για την εξασφάλιση των απαιτήσεων επίδοσης. Η αναζήτηση απλών και κομψών λύσεων στην κωδικοποίηση δεν είναι καθόλου ευκαταφρόνητο εγχείρημα. Επιπλέον, ο απλούστερος κώδικας, εκτός από περισσότερο ευανάγνωστος, είναι και λιγότερο επιρρεπής σε σφάλματα.
- **Πλεονασμός χάριν σαφήνειας.** Η χρήση κατάλληλων «σημείων στίξης» οδηγεί σε περισσότερο ευανάγνωστο κώδικα, όπως ακριβώς και στη φυσική γλώσσα. Τέτοια

σημεία στίξης είναι κυρίως οι παρενθέσεις και οι λέξεις-κλειδιά για την ομαδοποίηση εντολών (begin και end, { και } κ.ά.), που αν και ορισμένες φορές είναι περιττές σύμφωνα με τους συντακτικούς κανόνες της γλώσσας προγραμματισμού, είναι καλό να χρησιμοποιούνται για τη διευκόλυνση του αναγνώστη.

- **Επιλογή κατάλληλων ονομάτων.** Εξαιρετικής σημασίας για τη συγγραφή αναγνώσιμου κώδικα είναι η σωστή επιλογή ονομάτων για τα αναγνωριστικά του προγράμματος. Το ακόλουθο παράδειγμα, στο οποίο τα δυο κομμάτια κώδικα σε γλώσσα C είναι ισοδύναμα και διαφέρουν μόνο στα ονόματα των μεταβλητών και στη χρήση σταθερών, είναι ιδιαίτερα πειστικό.

```
if (x > 50000)
    y = (x – 50000)* 0.2;
Else
  y = 0;
```

```
const double NOT_TAXABLE = 50000;
const double TAX_RATE = 0.2;

if (income > NOT_TAXABLE)
        tax = (income – NOT_TAXABLE) * TAX_RATE;
else
        tax = 0;
```

- **Χρήση σχολίων.** Τα σχόλια αποτελούν αναπόσπαστο μέρος του πηγαίου κώδικα και ως εκ τούτου, πρέπει να γράφονται συγχρόνως με τον κώδικα και να ενημερώνονται ανελλιπώς σε κάθε μετέπειτα αλλαγή του. Τις περισσότερες φορές η έλλειψη σχολίων δυσχεραίνει ή καθιστά εντελώς αδύνατη την κατανόηση ενός προγράμματος. Για το λόγο αυτό, περιγραφικά σχόλια πρέπει να γράφονται τουλάχιστον στην αρχή των μονάδων του πηγαίου κώδικα, αλλά και όπου απαιτείται, στο εσωτερικό αυτών.
- **Στοίχιση.** Η αναγνωσιμότητα του πηγαίου κώδικα βελτιώνεται κατά πολύ αν χρησιμοποιούνται κανόνες στοίχισης του κώδικα και περιορίζεται ο αριθμός εντολών ανά γραμμή προγράμματος. Η μορφή της στοίχισης είναι υποκειμενική και πρέπει να αποφασίζεται από κοινού από την ομάδα κωδικοποίησης. Στη συνέχεια όμως οι κανόνες στοίχισης πρέπει να χρησιμοποιούνται πιστά από όλα τα μέλη της ομάδας.

5.2.4 Τεκμηρίωση

Οι τεχνικές για τη βελτίωση της αναγνωσιμότητας που αναφέρθηκαν παραπάνω δεν είναι πάντα αρκετές ως βοηθήματα για την κατανόηση του πηγαίου κώδικα. Είναι συχνά αναγκαίο να τεκμηριώνεται ο πηγαίος κώδικας με τη βοήθεια συνοδευτικών εγγράφων

γραμμένων σε κατάλληλη γλώσσα. Η *τεκμηρίωση* (documentation) του πηγαίου κώδικα ποικίλλει σε έκταση, μορφή και αυστηρότητα της γλώσσας που χρησιμοποιείται. Στην πράξη χρησιμοποιείται συνήθως φυσική γλώσσα, που αποσκοπεί στην επεξήγηση των πιο πολύπλοκων σημείων του πηγαίου κώδικα, όταν αυτό δεν είναι εύκολο να γίνει με σχόλια μέσα σε αυτόν.

5.2.5 Μεταφερσιμότητα

Με τον όρο *μεταφερσιμότητα* (portability) εννοούμε την ιδιότητα του πηγαίου κώδικα να μπορεί να εκτελεστεί χωρίς αλλαγές σε διαφορετικά περιβάλλοντα (λειτουργικά συστήματα, τύπος υπολογιστή) στα οποία διατίθενται εκδόσεις της γλώσσας προγραμματισμού που χρησιμοποιείται. Από την πλευρά του μηχανικού λογισμικού, είναι συνήθως πολύ σημαντικό να μπορεί να μεταφερθεί ο πηγαίος κώδικας ως έχει από μια υλοποίηση της γλώσσας προγραμματισμού στην οποία είναι γραμμένος σε μια άλλη. Για να επιτευχθεί αυτό, είναι τις περισσότερες φορές αρκετό:

- να μη χρησιμοποιούνται συστατικά της γλώσσας που υποστηρίζονται μόνο από συγκεκριμένες υλοποιήσεις και
- να μη γίνονται αυθαίρετες παραδοχές για τη συμπεριφορά των υλοποιήσεων, δηλαδή παραδοχές που δεν περιγράφονται ρητά στο πρότυπο (standard) της γλώσσας.

5.2.6 Δυνατότητα επαναχρησιμοποίησης

Η *δυνατότητα επαναχρησιμοποίησης* (reusability) είναι ιδιαίτερα σημαντικό χαρακτηριστικό του πηγαίου κώδικα, καθώς ένα αρκετά μεγάλο τμήμα του πηγαίου κώδικα ενός συστήματος λογισμικού μπορεί συνήθως να χρησιμοποιηθεί στην κωδικοποίηση άλλων συστημάτων λογισμικού. Προϋπόθεση για τη συγγραφή επαναχρησιμοποιήσιμου κώδικα είναι η κατάλληλη δόμησή του.

5.3 Τεχνικές συγγραφής πηγαίου κώδικα

Δεν υπάρχει η «χρυσή βίβλος» που περιέχει συμβουλές για τη συγγραφή αλάνθαστου πηγαίου κώδικα, διότι, πολύ απλά, δεν υπάρχει αλάνθαστος πηγαίος κώδικας. Μπορεί,

ωστόσο, κανείς να λάβει ορισμένα μέτρα προκειμένου να συγγράφει όσο το δυνατόν καλύτερο κώδικα. Η περιγραφή τέτοιων μέτρων με λεπτομέρεια δεν είναι δυνατή, εφόσον στο παρόν βιβλίο δεν αναφερόμαστε σε κάποια συγκεκριμένη γλώσσα προγραμματισμού. Είναι, πάραυτα, χρήσιμη η παράθεση μιας σειράς συμβουλών για τη συγγραφή πηγαίου κώδικα. Με το θέμα ασχολείται η παρούσα παράγραφος.

5.3.1 Τεχνικές αποφυγής σφαλμάτων

Η *αποφυγή σφαλμάτων* (fault avoidance) αποσκοπεί στη συγγραφή πηγαίου κώδικα με όσο το δυνατό λιγότερα σφάλματα. Η πιστή χρήση τεχνικών αποφυγής σφαλμάτων είναι ο ακρογωνιαίος λίθος στην ανάπτυξη αξιόπιστου λογισμικού. Δεδομένου του πολύ υψηλού κόστους εντοπισμού και διόρθωσης σφαλμάτων στον πηγαίο κώδικα, η διαδικασία ανάπτυξης λογισμικού πρέπει να είναι προσανατολισμένη κυρίως προς την αποφυγή των λαθών κατά τη συγγραφή του κώδικα. Κατάλληλα εργαλεία και τεχνικές είναι διαθέσιμα γι'αυτό το σκοπό. Γενικά, η συγγραφή κώδικα χωρίς σφάλματα διευκολύνεται από τη σωστή προδιαγραφή των απαιτήσεων, από την αξιοποίηση των χαρακτηριστικών της γλώσσας προγραμματισμού και ασφαλώς, από τον ίδιο τον ανθρώπινο παράγοντα.

5.3.2 Προδιαγραφή απαιτήσεων

Η ύπαρξη ενός αυστηρού τρόπου περιγραφής της προδιαγραφής των απαιτήσεων από το λογισμικό διευκολύνει εξαιρετικά την αποφυγή σφαλμάτων, καθώς αποφεύγονται παρερμηνείες στην επιθυμητή λειτουργία του πηγαίου κώδικα. Η πειθαρχημένη εφαρμογή των προβλεπόμενων μέχρι την κωδικοποίηση ενεργειών ανάπτυξης λογισμικού αποτελεί ισχυρό εργαλείο αποφυγής σφαλμάτων. Η σαφήνεια στην περιγραφή όλων των ενδιάμεσων προϊόντων (προδιαγραφές, σχεδίαση) και η αποφυγή «ευκόλως εννοουμένων» συνήθως αποδίδουν το χρόνο και τον κόπο που απαιτούν. Επιπλέον, έχουν αναπτυχθεί και *τυπικές μέθοδοι* (formal methods) για να γίνεται έλεγχος, χειρωνακτικά ή και ενίοτε αυτόματα, αν ο πηγαίος κώδικας που έχει γραφεί πληροί τις απαιτήσεις. Από την άλλη πλευρά όμως, το κόστος συγγραφής απαιτήσεων είναι συνήθως υψηλότερο, όταν η γλώσσα που χρησιμοποιείται είναι αυστηρότερη ή τυπική.

5.3.3 Αξιοποίηση της γλώσσας προγραμματισμού

Η πολιτική συγγραφής πηγαίου κώδικα πρέπει να αποβλέπει στην αξιοποίηση των *θετικών* χαρακτηριστικών μιας γλώσσας προγραμματισμού και στην αποφυγή χρήσης *αρνητικών* χαρακτηριστικών της. Για παράδειγμα, αν μια γλώσσα προγραμματισμού δεν είναι ισχυρή στους μαθηματικούς υπολογισμούς, τότε δεν είναι καλή ιδέα να χρησιμοποιηθεί για τη συγγραφή εφαρμογής στην οποία οι μαθηματικοί υπολογισμοί έχουν ιδιαίτερη βαρύτητα.

5.3.4 Επιδιωξη ποιοτητας

Η ομάδα κωδικοποίησης πρέπει να διαθέτει αυξημένη ευαισθησία για την εξασφάλιση της απαιτούμενης ποιότητας και την ανάπτυξη πηγαίου κώδικα που να διαθέτει τα επιθυμητά χαρακτηριστικά. Οι προγραμματιστές πρέπει να εφαρμόζουν αυστηρά τη συνολική πολιτική συγγραφής κώδικα και να επιδιώκουν εξαρχής την ανάπτυξη κώδικα χωρίς σφάλματα.

5.3.5 Ανοχή σε σφάλματα

Ανοχή σε σφάλματα (fault tolerance). Δεδομένης της πολυπλοκότητας των σύγχρονων συστημάτων λογισμικού, τις περισσότερες φορές δεν είναι εφικτή η αποφυγή ή ο εντοπισμός των σφαλμάτων. Ακόμα κι αν αυτά ήταν εφικτά, το κόστος για να επιτευχθούν είναι συνήθως απαγορευτικό. Για το λόγο αυτό, συστήματα λογισμικού με αυξημένες απαιτήσεις αξιοπιστίας οφείλουν να προβλέπουν την πιθανή εμφάνιση σφαλμάτων στη λειτουργία τους και να δείχνουν σχετική ανοχή. Ειδικές τεχνικές χρησιμοποιούνται για το σκοπό αυτό.

Συνήθως απαιτείται από τα συστήματα λογισμικού να συνεχίζουν τη λειτουργία τους μετά την εμφάνιση κάποιων σφαλμάτων ή άλλων απροσδόκητων καταστάσεων, με τις μικρότερες δυνατές απώλειες. Σε ορισμένες περιπτώσεις, π.χ. στο λογισμικό που χρησιμοποιείται στον υπολογιστή ενός αεροσκάφους εν πτήσει ή για τον έλεγχο εναέριας κυκλοφορίας, η συνολική αποτυχία ενός συστήματος λογισμικού θα είχε καταστροφικά αποτελέσματα και πρέπει να αποφευχθεί πάση θυσία. Για το λόγο αυτό και δεδομένης της αδυναμίας εξασφάλισης του αλάθητου των συστημάτων λογισμικού με τις τεχνικές αποφυγής και εντοπισμού σφαλμάτων, στην ανάπτυξη συστημάτων λογισμικού υψηλού κινδύνου γίνεται πρόβλεψη για την προφύλαξη από σφάλματα που ενδέχεται να παρουσιαστούν μετά την έναρξη λειτουργίας τους.

Για την ανάπτυξη συστημάτων λογισμικού ανεκτικών σε σφάλματα απαιτείται η προσθήκη επιπλέον κώδικα που αναλαμβάνει τις ακόλουθες λειτουργίες:

- **Δυναμικός εντοπισμός σφαλμάτων.** Το σύστημα λογισμικού πρέπει να είναι σε θέση να διαγνώσει ότι παρουσιάστηκε σφάλμα κατά τη λειτουργία του. Η διάγνωση τέτοιου είδους σφαλμάτων είναι απαραίτητο να γίνεται όσο το δυνατό νωρίτερα, ώστε να μη διευρύνονται και να μην επηρεάζουν ανεξάρτητα τμήματα του συστήματος που λειτουργούν σωστά.
- **Ανάνηψη από σφάλματα.** Σε περίπτωση εντοπισμού σφάλματος κατά τη λειτουργία του, το σύστημα λογισμικού πρέπει να επιδιορθώσει τη βλάβη και να αποκαταστήσει τη φυσιολογική του λειτουργία. Αυτό μπορεί να επιτευχθεί είτε με την οπισθοχώρηση σε μια προηγούμενη ασφαλή κατάσταση του συστήματος είτε με τη διόρθωση της βλάβης και τη μετάβαση σε μια νέα σωστή κατάσταση. Στην περίπτωση συστημάτων λογισμικού που διαχειρίζονται βάσεις δεδομένων χρησιμοποιείται συνήθως η τακτική οπισθοχώρησης και συγκεκριμένα η τεχνική των *συναλλαγών* (transactions). Αν

κάποια συναλλαγή δεν είναι δυνατό να ολοκληρωθεί λόγω της εμφάνισης σφάλματος, τότε ολόκληρη η συναλλαγή ακυρώνεται, προκειμένου να επανέλθει το σύστημα σε μια ασφαλή και συνεπή κατάσταση. Η τακτική μετάβασης σε νέα σωστή κατάσταση είναι συνήθως δυσκολότερη στην υλοποίηση.

- **Πρόληψη μελλοντικών σφαλμάτων.** Η ανάνηψη από ένα σφάλμα που παρουσιάστηκε κατά τη διάρκεια της λειτουργίας ενός συστήματος λογισμικού δεν είναι, φυσικά, αρκετή. Πρέπει να ληφθεί πρόνοια ώστε το σφάλμα να μην επαναληφθεί στο μέλλον. Όταν αυτό δεν είναι δυνατό να επιτευχθεί από το ίδιο το σύστημα λογισμικού, με την αυτόματη επιδιόρθωση του τμήματος που προκάλεσε το σφάλμα, απαιτείται η επέμβαση του κατασκευαστή. Στην περίπτωση αυτή το σύστημα οφείλει να γνωστοποιήσει την ύπαρξη του σφάλματος και να επιτρέπει τη συντήρησή του με το ελάχιστο δυνατό κόστος.

Διάφορες τεχνικές έχουν προταθεί για την υλοποίηση συστημάτων ανεκτικών σε σφάλματα. Δύο από αυτές περιγράφονται στη συνέχεια.

Προγραμματισμός πολλών εκδόσεων. Σύμφωνα με αυτή την τεχνική, το σύστημα λογισμικού υλοποιείται σε έναν αριθμό πλήρως λειτουργικών διαφορετικών εκδόσεων, που αναπτύσσονται από διαφορετικές ομάδες. Οι εκδόσεις αυτές εκτελούνται παράλληλα με τα ίδια δεδομένα εισόδου και τα αποτελέσματά τους συγκρίνονται μεταξύ τους. Στην περίπτωση ασυμφωνίας των αποτελεσμάτων, εφαρμόζεται ο κανόνας της πλειοψηφίας και οι εκδόσεις που οδήγησαν σε διαφορετικά αποτελέσματα απομονώνονται ως ελαττωματικές. Είναι φανερό ότι η τεχνική αυτή έχει σημαντικό κόστος, ωστόσο σε λογισμικόπου χρησιμοποιείται σε κρίσιμες εφαρμογές συνήθως δικαιολογείται το μεγάλο κόστος. Τουλάχιστον τρεις εκδόσεις του συστήματος πρέπει να είναι διαθέσιμες, όπως φαίνεται στο Σχήμα 5.4. Μια παραλλαγή είναι ή τεχνική να περιοριστεί μόνο στα επισφαλή τμήματα του λογισμικού, οπότε μειώνεται και το κόστος.

Αμυντικός προγραμματισμός (defensive programming). Σύμφωνα με αυτή την τεχνική, οι προγραμματιστές ενσωματώνουν στον πηγαίο κώδικα ελέγχους για τη συνέπεια των δεδομένων και την καλή λειτουργία, υιοθετώντας μια αμυντική στάση και προβλέποντας την περίπτωση ασυνεπειών. Αν κάποιος έλεγχος αποτύχει, τότε θεωρείται ότι εντοπίστηκε σφάλμα και εφαρμόζονται κατάλληλες τεχνικές ανάνηψης και μελλοντικής πρόληψης.

Εντοπισμός και διόρθωση σφαλμάτων

Ο εντοπισμός και η διόρθωση των σφαλμάτων που υπάρχουν στο λογισμικό αποσκοπεί στην εύρεση σφαλμάτων που υπάρχουν στον πηγαίο κώδικα πριν από την παράδοση αυτού σε τελική χρήση. Οι εργασίες αυτές γίνονται κατά τη φάση του ελέγχου του λογισμικού.

5.4 Επαναχρησιμοποίηση μονάδων προγράμματος

Επαναχρησιμοποίηση (reuse) είναι η διαδικασία της υλοποίησης νέου λογισμικού χρησιμοποιώντας συστατικά από ήδη κατασκευασμένο λογισμικό. Εκτός από τον πηγαίο κώδικα και τα εκτελέσιμα προγράμματα, επαναχρησιμοποιήσιμα συστατικά είναι επίσης τα αποτελέσματα της ανάλυσης και της σχεδίασης, η τεκμηρίωση, δεδομένα και σχέδια ελέγχου κτλ. Η επαναχρησιμοποίηση είναι βασική αρχή σε κάθε επιστήμη που σχετίζεται με την ανάπτυξη και την παραγωγή. Αποτελεί βάση για δραστική βελτίωση της ποιότητας και της αξιοπιστίας του λογισμικού, καθώς και για τη μακροπρόθεσμη μείωση του κόστους ανάπτυξής του.

Προκειμένου να είναι επαναχρησιμοποιήσιμος, ο πηγαίος κώδικας πρέπει να έχει γραφεί όχι μόνο για τη στενή του χρήση στο πλαίσιο κάποιας συγκεκριμένης εφαρμογής, αλλά με την πρόβλεψη να μπορεί να χρησιμοποιηθεί στο μέλλον σε παρόμοιες περιπτώσεις. Η γενικότητα εξασφαλίζεται με την παραμετροποίηση, όπου αυτό είναι δυνατό, σε βάρος συχνά του αρχικού κόστους ανάπτυξης του πηγαίου κώδικα. Επίσης, ο πηγαίος κώδικας πρέπει να είναι μεταφέρσιμος, να έχει τη μικρότερη δυνατή αλληλεπίδραση με τον υπόλοιπο κώδικα και, όπου αυτή η αλληλεπίδραση είναι αναπόφευκτη, να είναι καλά τεκμηριωμένη.

Η επαναχρησιμοποίηση έχει για πολύ καιρό αποτελέσει αντικείμενο συζητήσεων στην κοινότητα των κατασκευαστών και των ερευνητών. Παρά τις κατά καιρούς πομπώδεις δηλώσεις περί μεθοδολογιών ή εργαλείων που εξασφαλίζουν την επαναχρησιμοποίηση, κάτι τέτοιο δεν έχει μέχρι σήμερα γίνει δυνατό σε μεγάλη κλίμακα. Τέτοιες δηλώσεις συνόδεψαν επί μακρόν τις αντικειμενοστρεφείς μεθοδολογίες ανάπτυξης λογισμικού, ωστόσο το ενδιαφέρον είναι ότι μέχρι σήμερα η επαναχρησιμοποίηση είναι πραγματικότητα ίσως περισσότερο για γλώσσες προγραμματισμού όπως η FORTRAN, με τη μορφή βιβλιοθηκών συναρτήσεων, και λιγότερο για γενικής χρήσης λογισμικό κατά-σκευασμένο με σύγχρονες μεθόδους και γλώσσες προγραμματισμού. Η τάση είναι, ασφαλώς, προς την επικράτηση της επαναχρησιμοποίησης ως τεχνικής και μέσα στον ίδιο κατασκευαστή η επαναχρησιμοποίηση είναι ανάγκη, όχι απλώς επίτευγμα. Μεταξύ διαφορετικών κατασκευαστών, όμως, υπάρχει ακόμα αρκετή δουλειά να γίνει ώστε να μιλάμε για γενικευμένη πραγματικότητα.

Κατά τη διάρκεια και μετά το πέρας της διαδικασίες υλοποίησης, το πρόγραμμα που αναπτύσσεται πρέπει να ελεγχθεί ώστε να διασφαλιστεί ότι ικανοποιεί τις προδιαγραφές του και ότι παρέχει τη λειτουργικότητα που αναμένουν τα άτομα που πλήρωσαν για αυτό. Οι σχετικές διαδικασίες ελέγχου και ανάλυσης ονομάζονται επαλήθευση και επικύρωση (E&E). Δραστηριότητες επαλήθευσης και επικύρωσης λαμβάνουν χώρα σε κάθε στάδιο της διαδικασίες παραγωγής λογισμικού. Η E&E ξεκινάει με επισκοπήσεις των απαιτήσεων, συνεχίζεται με επισκοπήσεις του σχεδιασμού και επιθεωρήσεις του κώδικα, και ολοκληρώνεται με τη δοκιμή προϊόντος.

Επαλήθευση και επικύρωση δεν είναι το ίδιο πράγμα, αν και συχνά συγχέονται. Ο Boehm[4](Boehm, 1979) εξέφρασε τη διαφορά τους με λακωνικό τρόπο:

- «Επικύρωση (validation): Φτιάχνουμε το σωστό προϊόν;»
- «Επαλήθευση (verification): Φτιάχνουμε σωστά το προϊόν;»

Οι ανωτέρω ορισμοί μας λένε ότι ο ρόλος της επαλήθευσης αφορά τον έλεγχο για το αν το λογισμικό συμμορφώνεται με τις προδιαγραφές του. Αυτό που πρέπει να ελεγχθεί είναι αν το λογισμικό ικανοποιεί τις καθορισμένες λειτουργικές και μη λειτουργικές απαιτήσεις του. Η επικύρωση, όμως, είναι γενικότερη διαδικασία, της οποίας σκοπός είναι να εξασφαλίσει ότι το σύστημα λογισμικού ικανοποιεί τις προσδοκίες του πελάτη. Προχωράει πέρα από τον έλεγχο για το αν το σύστημα συμμορφώνεται με τις προδιαγραφές του, και προσπαθεί να αποδείξει ότι το λογισμικό αυτό που περιμένει ο πελάτης. Όπως είδαμε και στο κεφάλαιο 3, οι προδιαγραφές ενός συστήματος λογισμικού δεν αντικατοπτρίζουν πάντοτε τις πραγματικές επιθυμίες ή ανάγκες των χρηστών και των κατόχων του.

Ο τελικός στόχος της διαδικασίας επαλήθευσης και επικύρωσης είναι να δημιουργήσει για ένα επαρκές αισθητά εμπιστοσύνης ότι το σύστημα λογισμικού είναι κατάλληλο για τη συγκεκριμένη εργασία. Αυτό σημαίνει ότι το σύστημα πρέπει να είναι αρκετά καλό για τη χρήση που προορίζεται. Το επίπεδο της απαιτούμενης εμπιστοσύνης που πρέπει να εδραιωθεί εξαρτάται από το σκοπό του συστήματος, τις προσδοκίες των χρηστών του, και το τρέχον εμπορικό περιβάλλον:

1. Λειτουργία λογισμικού. Το επίπεδο της επιβεβλημένης εμπιστοσύνης εξαρτάται από το πόσο κρίσιμο είναι το λογισμικό για έναν οργανισμό. Λόγου χάρη, το επίπεδο της απαιτούμενης εμπιστοσύνης για λογισμικό το οποίο χρησιμοποιείται σε ένα σύστημα

κρίσιμο από άποψη ασφαλείας είναι κατά πολύ υψηλότερο από εκείνο που απαιτείται για κάποιο πρωτότυπο, το οποίο έχει αναπτυχθεί με σκοπό την παρουσίαση ορισμένων νέων ιδεών.

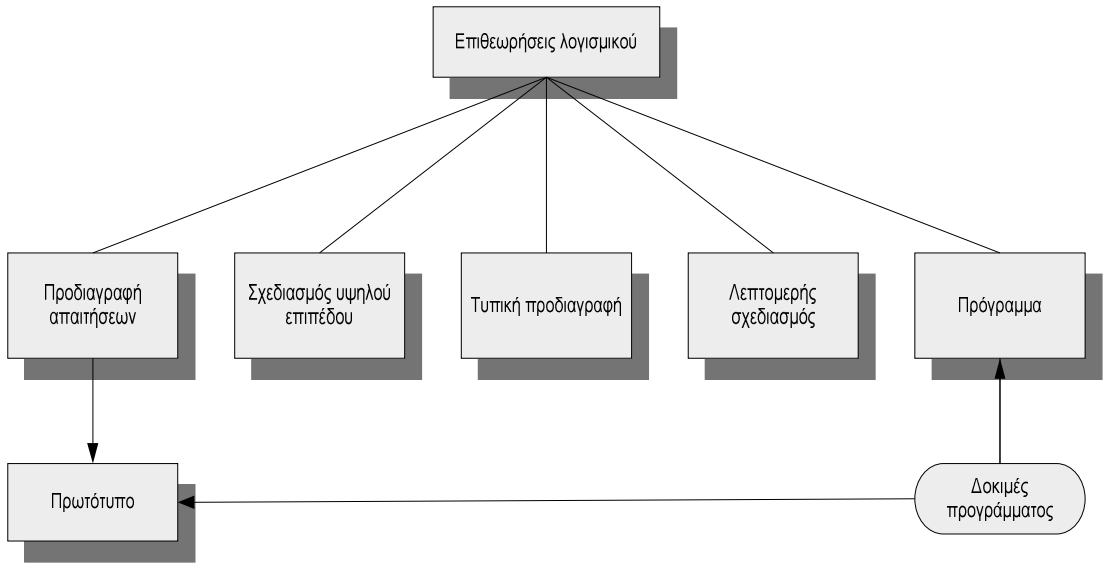
2. Προσδοκίες χρηστών. Είναι ιδιαίτερα λυπηρό για τη βιομηχανία λογισμικού το γεγονός ότι πολλοί χρήστες έχουν χαμηλές προσδοκίες από το λογισμικό τους και δεν εκπλήσσονται όταν αυτό παρουσιάζει προβλήματα. Είναι πρόθυμοι να αποδεκτούν αυτές τις αστοχίες, όταν τα οφέλη από τη χρήση ξεπερνούν τα μειονεκτήματα. Ωστόσο, η ανοχή των χρηστών στις αστοχίες συστημάτων είναι πλέον πολύ μικρότερη από ό,τι τη δεκατία του 1990. Σήμερα, η παράδοση αναξιόπιστων συστημάτων είναι λιγότερο αποδεκτή, συνεπώς οι εταιρείες λογισμικού πρέπει να αφιερώνουν μεγαλύτερη προσπάθεια στις διαδικασίες επαλήθευσης και επικύρωσης.
3. Εμπορικό περιβάλλον. Όταν ένα σύστημα πρόκειται να κυκλοφορήσει στην αγορά, οι κατασκευαστές του πρέπει να λαμβάνουν υπόψη τους ανταγωνιστικά προγράμματα, την τιμή που είναι πρόθυμοι να πληρώσουν οι πελάτες, και το απαιτούμενο χρονοδιάγραμμα παράδοσης του συστήματος. Σε περίπτωση που μια εταιρεία αντιμετωπίζει μικρό ανταγωνισμό, θα μπορούσε να αποφασίσει να κυκλοφορήσει ένα πρόγραμμα πριν αυτό δοκιμαστεί και αποσφαλματωθεί πλήρως, προκειμένου να προλάβει την αγορά. Οι πελάτες που δεν είναι πρόθυμοι να πληρώσουν υψηλές τιμές για λογισμικό, ίσως είναι έτοιμοι να αποδεχτούν περισσότερα προβλήματα από αυτό. Όλοι αυτοί οι παράγοντες πρέπει να εξετάζονται όταν λαμβάνεται η απόφαση για το πόση προσπάθεια πρέπει να δαπανηθεί στις διαδικασίες E&E.

Η διαδικασία E&E περιλαμβάνει δύο προσεγγίσεις για τον έλεγχο και την ανάλυση ενός συστήματος, οι οποίες συμπληρώνουν η μία την άλλη:

1. Οι επιθεωρήσεις λογισμικού(software inspections) ή αναθεωρήσεις από ομότιμους (peer reviews) αναλύουν και ελέγχουν αναπαραστάσεις ενός συστήματος όπως το έγγραφο των απαιτήσεων, τα σχεδιαστικά διαγράμματα, και ο πηγαίος κώδικας του προγράμματος. Επιθεωρήσεις μπορούν να χρησιμοποιηθούν σε όλα τα στάδια της διαδικασίας, και συμπληρώνονται από αυτόματες αναλύσεις του πηγαίου κώδικα ενός συστήματος ή των σχετικών εγγράφων. Οι επιθεωρήσεις λογισμικού και οι αυτοματοποιημένες αναλύσεις συνιστούν στατικές τεχνικές E&E, καθώς το λογισμικό δε χρειάζεται να εκτελείται σε κάποιον υπολογιστή.
2. Οι δοκιμές λογισμικού (software testing) περιλαμβάνουν την εκτέλεση μιας υλοποίησης του συστήματος με δοκιμαστικά δεδομένα. Εξετάζεται η έξοδος του

λογισμικού, καθώς και η λειτουργική του συμπεριφορά, ώστε να εξασφαλιστεί αν έχει την κατάλληλη απόδοση. Η εκτέλεση δοκιμών αποτελεί μια δυναμική τεχνική επαλήθευσης και επικύρωσης.

Η Εικόνα 6.1 δείχνει ότι οι επιθεωρήσεις και οι δοκιμές λογισμικού έχουν συμπληρωματικούς ρόλους στη διαδικασία της παραγωγής. Τα βέλη σημειώνουν τα στάδια της διαδικασίας κατά τα οποία μπορεί να χρησιμοποιηθεί κάθε τεχνική. Συνεπώς, επιθεωρήσεις λογισμικού μπορούν να χρησιμοποιηθούν σε όλα τα στάδια της διαδικασίας παραγωγής. Η αρχή γίνεται με το έγγραφο απαιτήσεων, και κατόπιν κάθε αναπαράσταση του λογισμικού σε γραπτή μορφή μπορεί να επιθεωρηθεί. Η βασική τεχνική που χρησιμοποιείται για τον εντοπισμό λαθών στις προδιαγραφές και το σχεδιασμό είναι οι επισκοπήσεις (reviews).



ΣΧΗΜΑ 6.1 *Στατική και δυναμική επαλήθευση και επικύρωση*

Η δοκιμή ενός συστήματος είναι δυνατή μόνο όταν υπάρχει διαθέσιμο κάποιο πρωτότυπο ή μια εκτελέσιμη έκδοση του προγράμματος. Ένα πλεονέκτημα της βαθμιαίας ανάπτυξης είναι ότι σχετικά νωρίς στη διαδικασία υπάρχει διαθέσιμη μια έκδοση του συστήματος η οποία μπορεί να δοκιμαστεί. Η νέα λειτουργικότητα δοκιμάζεται καθώς προστίθεται στο σύστημα, συνεπώς δεν είναι απαραίτητο να υπάρχει μια ολοκληρωμένη υλοποίηση για να ξεκινήσουν οι δοκιμές.

Στις τεχνικές επιθεώρησης περιλαμβάνονται οι επιθεωρήσεις προγραμμάτων, η αυτοματοποιημένη ανάλυση πηγαίου κώδικα, και η τυπική επαλήθευση. Ωστόσο, οι στατικές

τεχνικές μπορούν μόνο να ελέγξουν την αντιστοιχία μεταξύ προγράμματος και προδιαγραφών (επαλήθευση), και δεν είναι σε θέση να δείξουν ότι το λογισμικό είναι λειτουργικά χρήσιμο. Επίσης, οι στατικές τεχνικές δεν μπορούν να χρησιμοποιηθούν για τον έλεγχο ανακυπτουσών ιδιοτήτων του λογισμικού, όπως η απόδοση και η αξιοπιστία.

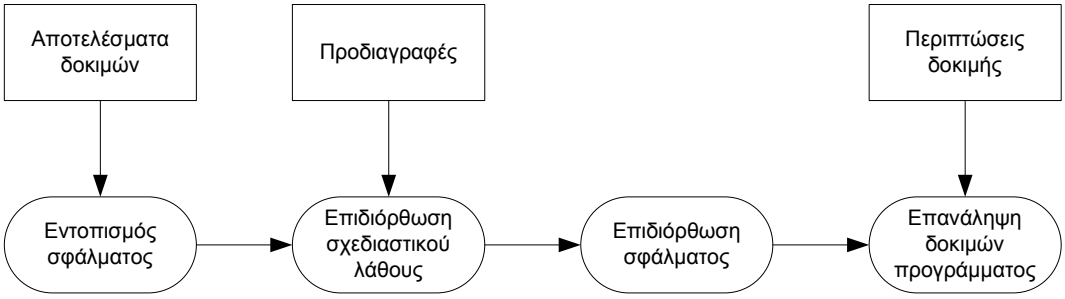
Αν και οι επιθεωρήσεις λογισμικού χρησιμοποιούνται πλέον ευρέως, η βασική τεχνική επαλήθευσης και επικύρωσης του λογισμικού θα είναι πάντοτε οι δοκιμές. Οι δοκιμές αφορούν τη λειτουργία του προγράμματος με τη χρήση δεδομένων τα οποία είναι παρόμοια με τα πραγματικά δεδομένα που επεξεργάζεται το σύστημα. Οι ατέλειες ή οι ανεπάρκειες ενός προγράμματος διαπιστώνονται με την εξέταση της εξόδου του προγράμματος και την αναζήτηση ανωμαλιών. Υπάρχουν δύο ξεχωριστοί τύποι δοκιμών οι οποίοι μπορούν να χρησιμοποιηθούν σε διαφορετικά στάδια της διαδικασίας παραγωγής λογισμικού:

1. Οι δοκιμές επικύρωσης (validation testing) προσπαθούν να δείξουν ότι το λογισμικό είναι αυτό που θέλει ο πελάτης-ότι δηλαδή το σύστημα ικανοποιεί τις απαιτήσεις του πελάτη. Μέρος των δοκιμών επικύρωσης θα μπορούσαν να αποτελούν στατιστικές δοκιμές, οι οποίες ελέγχουν την απόδοση και την αξιοπιστία του προγράμματος, καθώς και οι έλεγχοι συμπεριφοράς του συστήματος κάτω από συνθήκες λειτουργίας.
2. Οι δοκιμές για ατέλειες (defect testing) στοχεύουν στον εντοπισμό ατελειών του συστήματος και όχι στην προσομοίωση της λειτουργικής του χρήσης. Σκοπός των δοκιμών αυτών είναι ο εντοπισμός ασυνεπειών μεταξύ ενός προγράμματος και των προδιαγραφών του.

Φυσικά, μεταξύ των ανωτέρω προσεγγίσεων δεν υπάρχει κάποια αυστηρή διαχωριστική γραμμή. Οι δοκιμές επικύρωσης μπορεί να εντοπίσουν ατέλειες στο σύστημα, ενώ κάποιες από τις δοκιμές για ατέλειες ίσως επιβεβαιώσουν ότι το πρόγραμμα ικανοποιεί τις απαιτήσεις του.

Οι διαδικασίες E&E συνήθως συνδυάζονται με την αποσφαλμάτωση (debugging). Καθώς οι δοκιμές εντοπίζουν σφάλματα στο πρόγραμμα, αυτό πρέπει να τροποποιηθεί ώστε τα σφάλματα να διορθωθούν. Ωστόσο, οι δοκιμές (ή γενικότερα, οι διαδικασίες επαλήθευσης και επικύρωσης) και η αποσφαλμάτωση έχουν διαφορετικούς στόχους:

1. Οι διαδικασίες επαλήθευσης και επικύρωσης στοχεύουν στον εντοπισμό ατελειών σε ένα σύστημα λογισμικού.
2. Η αποσφαλμάτωση είναι μια διαδικασία που εντοπίζει και διορθώνει αυτές τις ατέλειες (Εικόνα 6.2)



ΣΧΗΜΑ 6.2 *Η διαδικασία αποσφαλμάτωσης*

Δεν υπάρχει κάποια απλή μέθοδος αποσφαλμάτωσης ενός προγράμματος. Οι ικανοί αποσφαλματωτές ελέγχουν τη δοκιμαστική έξοδο για μοτίβα τα οποία θα μπορούσαν να σημαίνουν ατέλειες, και εντοπίζουν το πρόβλημα χρησιμοποιώντας τις γνώσεις τους για τον τύπο της ατέλειας, το μοτίβο της εξόδου, τη γλώσσα προγραμματισμού, και την προγραμματιστική διαδικασία. Κατά την αποσφαλμάτωση, ο προγραμματιστής μπορεί να χρησιμοποιεί τις γνώσεις που έχει για συνηθισμένα λάθη προγραμματισμού (όπως η παράλειψη της αύξησης κάποιου μετρητή) και να τις συγκρίνει με τα παρατηρούμενα μοτίβα. Θα πρέπει επίσης να αναζητά χαρακτηριστικά λάθη που παρατηρούνται στη συγκεκριμένη γλώσσα προγραμματισμού, όπως λανθασμένη χρήση δεικτών στη C.

Ο εντοπισμός των σφαλμάτων ενός προγράμματος δεν είναι πάντοτε απλή διαδικασία, καθώς η ατέλεια μπορεί να μη βρίσκεται κοντά στο σημείο που αστόχησε το πρόγραμμα. Ο εντοπισμός ενός σφάλματος μπορεί να απαιτεί το σχεδιασμό επιπλέον δοκιμών, οι οποίες αναπαράγουν το αρχικό σφάλμα και εντοπίζουν τη θέση του μέσα στο πρόγραμμα. Ίσως να απαιτείται η παρακολούθηση της εκτέλεσης του προγράμματος γραμμή προς γραμμή (tracing) με μη αυτόματο τρόπο. Τέλος, χρήσιμα στον εντοπισμό της προέλευσης ενός προβλήματος μπορεί να φανούν διάφορα εργαλεία αποσφαλμάτωσης, τα οποία συλλέγουν πληροφορίες για την εκτέλεση του προγράμματος.

Τα περισσότερα συστήματα μεταγλώττισης διαθέτουν ενσωματωμένο ένα σύνολο εργαλείων υποστήριξης, μέρος του οποίου αποτελούν διάφορα αλληλεπιδραστικά εργαλεία αποσφαλμάτωσης. Τα εργαλεία αυτά παρέχουν ένα εξειδικευμένο περιβάλλον χρόνου εκτέλεσης για το πρόγραμμα, το οποίο επιτρέπει την πρόσβαση στον πίνακα συμβόλων του μεταγλωττιστή και μέσω αυτού, στις τιμές των μεταβλητών του προγράμματος. Ο προγραμματιστής έχει τη δυνατότητα να ελέγξει το πρόγραμμα εκτελώντας «βήμα προς

βήμα», δηλαδή εντολή προς εντολή. Μετά την εκτέλεση μιας εντολής, ο προγραμματιστής μπορεί να εξετάσει την τιμή κάθε μεταβλητής και να εντοπίσει έτσι τη θέση του σφάλματος.

Αφού εντοπιστεί κάποια ατέλεια, το πρόγραμμα πρέπει να διορθωθεί και να επικυρωθεί εκ νέου. Αυτό μπορεί να περιλαμβάνει νέες επιθεωρήσεις του προγράμματος ή δοκιμές παλινδρόμησης, κατά τις οποίες επαναλαμβάνονται όλες οι υπάρχουσες δοκιμές. Μια δοκιμή παλινδρόμησης (regression testing) έχει στόχο να εξασφαλίσει ότι οι μεταβολές που έγιναν σε ένα πρόγραμμα δεν εισήγαγαν νέα σφάλματα. Η πείρα έχει δείξει ότι ένα μεγάλο ποσοστό «επιδιορθώσεων» είτε είναι ημιτελείς είτε αποτελούν αιτία για νέα σφάλματα στο πρόγραμμα.

Θεωρητικά, μετά από κάθε επιδιόρθωση σφάλματος πρέπει να επαναλαμβάνονται όλες οι δοκιμές, στην πράξη όμως, κάτι τέτοιο έχει μεγάλο κόστος. Σε ένα τμήμα του σχεδίου δοκιμών θα πρέπει να προσδιορίζονται οι εξαρτήσεις μεταξύ των συστατικών στοιχείων, καθώς και οι δοκιμές που σχετίζονται με κάθε στοιχείο. Με άλλα λόγια, πρέπει να είναι γνωστό ποια συστατικά στοιχεία ελέγχει κάθε περίπτωση δοκιμής (μια ιδιότητα που ονομάζεται ανιχνευσιμότητα-traceability). Αν αυτές οι σχέσεις ανιχνευσιμότητας είναι τεκμηριωμένες, ο προγραμματιστής θα είναι σε θέση να εκτελέσει μόνο ένα υποσύνολο των περιπτώσεων δοκιμής του συστήματος, ώστε να ελέγξει το τροποποιημένο συστατικό στοιχείο και εκείνο που εξαρτούνται από αυτό.

6.1 Γενικό πλαίσιο ελέγχου

Ξεκινώντας την ενασχόλησή μας με το θέμα, είναι σκόπιμο να δώσουμε έναν ορισμό της έννοιας του «ελέγχου λογισμικού».

Έλεγχος λογισμικού

Έλεγχος[13] *είναι η διαδικασία κατά την οποία εξετάζεται το λογισμικό με χρήση ειδικά σχεδιασμένων τεχνικών και με σκοπό την εύρεση και διόρθωση σφαλμάτων στην υλοποίησή του.*

Τα σφάλματα που αναζητούνται σε αυτό το στάδιο ανάπτυξης κατηγοριοποιούνται σε δύο ομάδες. Στην πρώτη ομάδα διακρίνουμε τα σφάλματα που παρουσιάζονται λόγω αντίφασης των αποτελεσμάτων της λειτουργίας του λογισμικού με συγκεκριμένες απαιτήσεις και προδιαγραφές του συστήματος. Θεωρώντας ότι οι απαιτήσεις από το λογισμικό έχουν οριστεί σωστά, μπορούμε να πούμε ότι στην περίπτωση ύπαρξης των παραπάνω σφαλμάτων το λογισμικό δεν τις ικανοποιεί. Δηλαδή δίνει λάθος λύση σε ένα σωστό πρόβλημα. Στην περίπτωση αυτή έχουν γίνει λάθη κατά τη μετάβαση από τις προδιαγραφές στη σχεδίαση του λογισμικού. Ο τρόπος με τον οποίο ελέγχουμε το λογισμικό για τέτοιου είδους λάθη λέγεται *επικύρωση* (validation).

Στη δεύτερη ομάδα διακρίνουμε τα σφάλματα που παρουσιάζονται κατά την εκτέλεση συγκεκριμένων μονάδων του λογισμικού. Στην περίπτωση αυτή η σύλληψη της συγκεκριμένης μονάδας είναι σωστή, αλλά η υλοποίησή της παρουσιάζει λάθη. Ο τρόπος με τον οποίο ελέγχουμε το λογισμικό για τέτοιου είδους λάθη λέγεται *επαλήθευση* (verification).

Έτσι, υπάρχουν δύο διακριτοί τύποι ελέγχου:

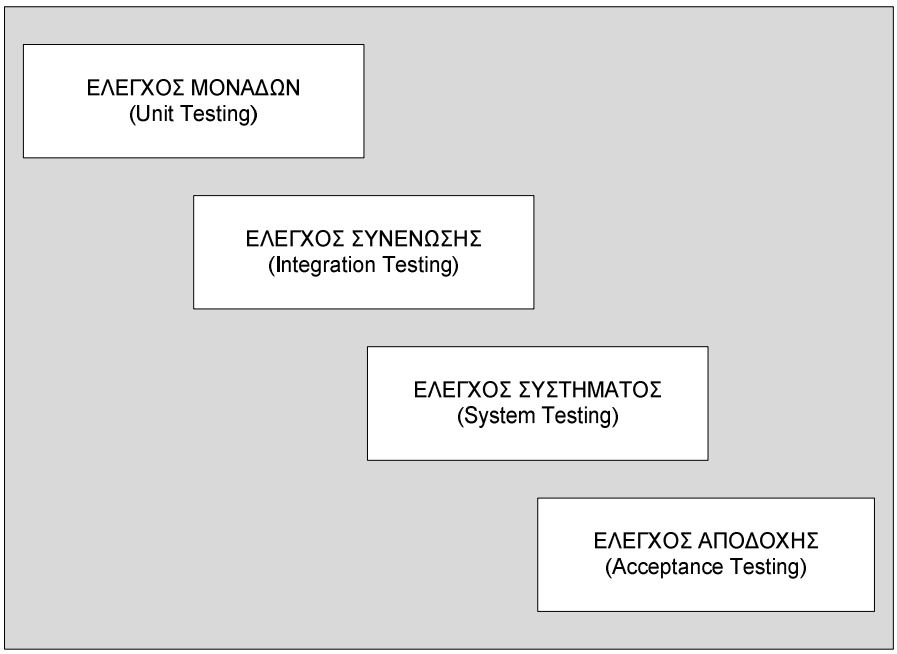
- Ο έλεγχος που βασίζεται στις απαιτήσεις από το σύστημα και κατά τον οποίο επαληθεύεται ότι το λογισμικό ανταποκρίνεται σε αυτές και
- Ο έλεγχος κατά τον οποίο επαληθεύεται ότι οι μονάδες του λογισμικού συστήματος έχουν υλοποιηθεί σωστά από προγραμματιστικής άποψης.

Έχοντας κατά νου ότι δεν υπάρχει λογισμικό χωρίς σφάλματα, μπορούμε να πούμε ότι ο έλεγχος κρίνεται επιτυχημένος όταν εντοπίσει σφάλματα και όχι στην αντίθετη περίπτωση.

[13] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **171.**

Στην περίπτωση που δεν εντοπιστούν σφάλματα, αυτό δε σημαίνει και ότι δεν υπάρχουν, αλλά μάλλον ότι ο έλεγχος που επιχειρήθηκε δεν ήταν ικανός να τα αποκαλύψει.

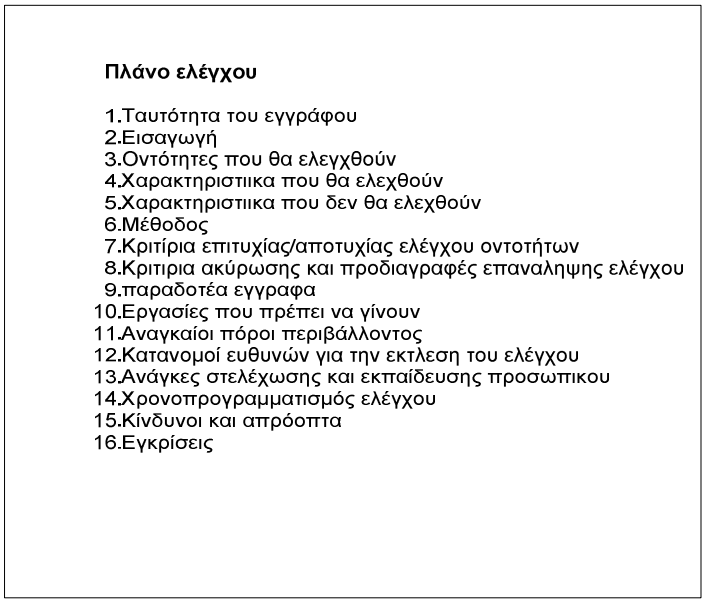
Ο έλεγχος εκτελείται σε τέσσερα στάδια Σχήμα 6.3, τα οποία αναλύονται στις επόμενες ενότητες, και τεκμηριώνεται με ένα πλάνο ελέγχου, το οποίο εξετάζεται στην επόμενη ενότητα.



ΣΧΗΜΑ 6.3 *Τα επίπεδα εκτέλεσης του ελέγχου*

6.1.1 Προγραμματισμός του ελέγχου

Για τη σωστή εκτέλεση του ελέγχου απαιτείται ο προγραμματισμός του, οι βάσεις για τον οποίο μπορούν να τίθενται οσοδήποτε νωρίς στην ανάπτυξη του λογισμικού. Ο προγραμματισμός αυτός καταγράφεται σε ένα έγγραφο, το οποίο ονομάζεται *πλάνο ελέγχου* (test plan), το οποίο περιέχει πληροφορίες για το σκοπό του ελέγχου που θα εκτελεστεί, τη στρατηγική που θα χρησιμοποιηθεί και τους αναγκαίους πόρους για τη διεκπεραίωση του. Ακολούθως, στο Σχήμα 6.4 παρατίθεται μια δομή του εγγράφου «πλάνο ελέγχου», βασισμένη σε πρότυπο του IEEE (*IEEE Standard for Software Test Documentation*, ANSI/IEEE, Std 829.1991).



ΣΧΗΜΑ 6.4 *Το πλάνο του ελέγχου λογισμικού κατά IEEE*

Το «πλάνο ελέγχου» δεν είναι απαραίτητο να κατασκευάζεται μετά την παραγωγή του πηγαίου κώδικα. Αντίθετα, όσο νωρίτερα είναι έτοιμο και διαθέσιμο σε όλους τους μετέχοντες στην ανάπτυξη του λογισμικού, τόσο περισσότερο διευκολύνεται η εκτέλεση του ελέγχου. Η διευκόλυνση αυτή μπορεί να εντοπιστεί σε δύο επίπεδα: πρώτον, στην ετοιμασία και διάθεση των αναγκαίων για τον έλεγχο εγγράφων και δεύτερον, στη δυνατότητα έναρξης της διαδικασίας του ελέγχου προτού να ολοκληρωθεί η παραγωγή του πηγαίου κώδικα, πράγμα που είναι και η καλύτερη μέθοδος που μπορεί να ακολουθήσει κανείς.

6.2 Τεχνικές ελέγχου

Ο έλεγχος μιας εφαρμογής λογισμικού στηρίζεται στην αρχή ότι εκτελείται ένα τμήμα αυτής με ένα σύνολο από δεδομένα εισόδου για τα οποία τα αποτελέσματα είναι γνωστά και αν τα αποτελέσματα που λαμβάνονται από την εκτέλεση δεν είναι ίδια με τα αναμενόμενα, τότε το τμήμα αυτό έχει σφάλματα.

Δοκιμή μονάδας

Η εκτέλεση μιας μονάδας προγράμματος με ένα σύνολο από δεδομένα για τα οποία τα αποτελέσματα είναι γνωστά λέγεται «δοκιμή» (test) της μονάδας. Δοκιμές[14] *που γίνονται με διαφορετικά δεδομένα θεωρούνται διαφορετικές.*

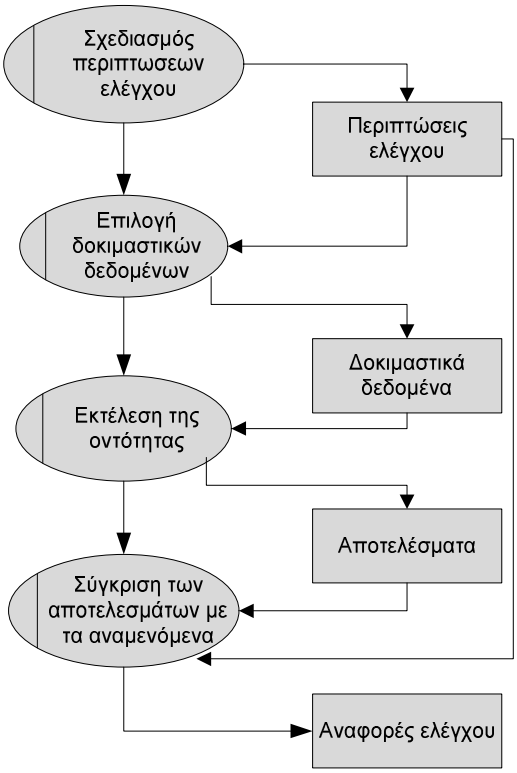
Τα δεδομένα εισόδου και εξόδου που χρησιμοποιούνται κατά τις δοκιμές λέγονται *δοκιμαστικά δεδομένα* (test data). Ακολούθως εισάγεται και ο όρος της «περίπτωσης ελέγχου».

Περίπτωση ελέγχου

Μια « περίπτωση ελέγχου[15]*» (test case) είναι το σύνολο των δοκιμαστικών δεδομένων, των συνθηκών εκτέλεσης και των αναμενόμενων αποτελεσμάτων που έχουν σχεδιαστεί με ένα συγκεκριμένο σκοπό, όπως το να καλύψουν ένα μονοπάτι εκτέλεσης του λογισμικού ή να επικυρώσουν τη συμφωνία με μια συγκεκριμένη απαίτηση από αυτό.*

Έχοντας κατά νου τους παραπάνω ορισμούς, στο Σχήμα 6.5 φαίνεται η γενική ροή ελέγχου μιας μονάδας λογισμικού. Αρχικά σχεδιάζονται οι περιπτώσεις ελέγχου, δηλαδή ορίζονται τα δοκιμαστικά δεδομένα, οι συνθήκες εκτέλεσης και τα αναμενόμενα αποτελέσματα για κάθε δοκιμή. Στη συνέχεια εκτελείται το ελεγχόμενο λογισμικό με τα δοκιμαστικά δεδομένα κάθε περίπτωσης ελέγχου και λαμβάνονται κάποια αποτελέσματα, τα οποία καταγράφονται και συγκρίνονται με τα αναμενόμενα αποτελέσματα των αντίστοιχων περιπτώσεων ελέγχου, ώστε να συνταχτούν οι απαραίτητες αναφορές ελέγχου.

[14] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **174.**
[15] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **174.**

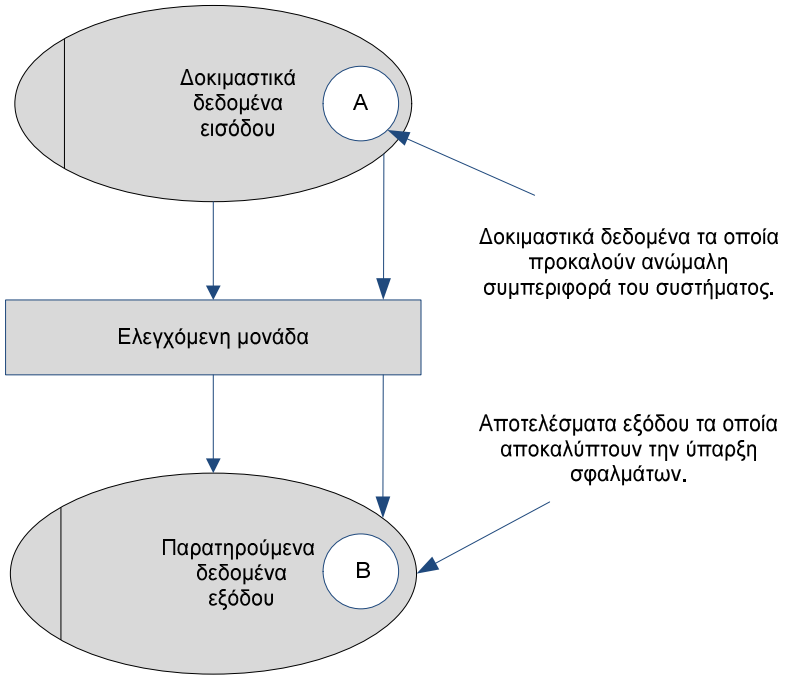


ΣΧΗΜΑ 6.5 *Γενική ροή ελέγχου μιας μονάδας λογισμικού*

Δεν είναι εφικτό να δοκιμαστούν όλες οι δυνατές περιπτώσεις ελέγχου, καθώς ο αριθμός τους είναι απαγορευτικά μεγάλος. Επομένως, θα πρέπει να γίνεται μια επιλογή ορισμένων μόνο απ' αυτές, η οποία θα πρέπει να είναι τέτοια ώστε να ικανοποιείται ο σκοπός του ελέγχου. Υπάρχουν δύο βασικές στρατηγικές που ακολουθούνται στην πράξη για τη λύση του προβλήματος αυτού, οι οποίες διακρίνονται από την άποψη από την οποία παρατηρούν το λογισμικό: η «στρατηγική του μαύρου κουτιού» και η «στρατηγική του γυάλινου κουτιού».

6.2.1 Στρατηγική του μαύρου κουτιού

Η *στρατηγική του μαύρου κουτιού* (black.box testing) βασίζεται στις προδιαγραφές της μονάδας λογισμικού που πρόκειται να ελεγχθεί, οι οποίες θεωρούνται γνωστές, ενώ ο τρόπος κατασκευής της θεωρείται άγνωστος. Αντιμετωπίζεται δηλαδή ως ένα «μαύρο κουτί» του οποίου η συμπεριφορά μπορεί να μελετηθεί μόνο παρατηρώντας τις εισόδους και τα αποτελέσματα που αντιστοιχούν σε αυτές (Σχήμα 6.6).



ΣΧΗΜΑ 6.6 *Έλεγχος λογισμικού με τη «στρατηγική του μαύρου κουτιού»*

Για την αποκάλυψη όλων των σφαλμάτων αρκεί να δοκιμαστούν τα δεδομένα που ανήκουν στο σύνολο Α, όπως φαίνεται στο παραπάνω σχήμα, δηλαδή τα δοκιμαστικά δεδομένα που προκαλούν ανώμαλη συμπεριφορά του συστήματος. Στην ουσία όμως, για να γίνει κάτι τέτοιο, θα πρέπει να δοκιμαστούν όλα τα δυνατά δεδομένα εισόδου, κάτι το οποίο είναι ανέφικτο. Θα πρέπει, λοιπόν, να γίνει επιλογή ενός συνόλου περιπτώσεων ελέγχου, οι οποίες έχουν μεγάλη πιθανότητα να αποκαλύψουν σφάλματα στην ελεγχόμενη οντότητα. Για το σκοπό αυτό έχουν αναπτυχθεί ορισμένες προσεγγίσεις, οι οποίες αναλύονται στη συνέχεια.

6.2.2 Προσέγγιση της ισοδύναμης διαμέρισης

Τα δεδομένα εισόδου σε ένα λογισμικό σύστημα μπορούν συνήθως να κατηγοριοποιηθούν σ'έναν αριθμό διαφορετικών κλάσεων. Οι κλάσεις αυτές έχουν κοινά χαρακτηριστικά (π.χ. θετικοί αριθμοί, αρνητικοί αριθμοί, συμβολοσειρές χωρίς κενά κτλ.). Μια μονάδα λογισμικού συνήθως συμπεριφέρεται με παρόμοιο τρόπο για όλα τα μέλη καθεμιάς απ'αυτές τις κλάσεις. Δηλαδή, αν μια τιμή αποκαλύψει ένα λάθος κατά τον έλεγχο, τότε το ίδιο λάθος θα αποκαλύψει και μια οποιαδήποτε από τις υπόλοιπες τιμές της ίδιας κλάσης. Επίσης, αν μια τιμή δεν αποκαλύψει κάποιο λάθος, το ίδιο θα κάνει και μια από τις

υπόλοιπες τιμές. Εξαιτίας αυτής της «ισοδύναμης» συμπεριφοράς, οι κλάσεις αυτές ονομάζονται *κλάσεις ισοδύναμων τιμών* (equivalence partitions).

Κλάση ισοδύναμων τιμών

Μια κλάση ισοδύναμων τιμών[16] *είναι ένα σύνολο με μέλη είτε δοκιμαστικά δεδομένα εισόδου είτε δεδομένα εξόδου, τα οποία προσδιορίζουν μια ισοδύναμη συμπεριφορά του συστήματος.*

Η προσέγγιση της «ισοδύναμης διαμέρισης» βασίζεται στον προσδιορισμό ενός συνόλου κλάσεων ισοδύναμων τιμών. Οι περιπτώσεις ελέγχου στη συνέχεια θα πρέπει να σχεδιάζονται έτσι ώστε τα δοκιμαστικά δεδομένα, αλλά και τα αναμενόμενα αποτελέσματα που τις χαρακτηρίζουν, να ανήκουν σε κάποια απ'αυτές τις κλάσεις. Με τον τρόπο αυτό μειώνεται το πλήθος των περιπτώσεων ελέγχου, καθώς πλέον το πολύ να είναι ίσο με το πλήθος των κλάσεων των ισοδύναμων τιμών, αφού πλέον δε χρειάζεται να ελέγξουμε παραπάνω από ένα δοκιμαστικό δεδομένο μιας κλάσης, καθώς θα πάρουμε το ίδιο αποτέλεσμα ως προς την ορθότητα.

Οι κλάσεις ισοδύναμων τιμών μπορούν να διακριθούν σε «κλάσεις ισοδύναμων τιμών εισόδου» και σε «κλάσεις ισοδύναμων τιμών εξόδου». Οι πρώτες έχουν μέλη δοκιμαστικά δεδομένα, ενώ οι δεύτερες έχουν μέλη δεδομένα εξόδου (αποτελέσματα εκτέλεσης). Επίσης, είναι δυνατή η διάκριση των κλάσεων ισοδύναμων τιμών εισόδου σε έγκυρες και άκυρες, ανάλογα με το αν προκαλούν ομαλή ή όχι λειτουργία της μονάδας που ελέγχεται.

Ο προσδιορισμός των κλάσεων ισοδύναμων τιμών εισόδου βασίζεται στους περιορισμούς που υπάρχουν στα δεδομένα εισόδου (συνθήκες εισόδου). Από τις συνθήκες εισόδου μπορούν να παραχθούν κλάσεις ισοδύναμων τιμών μ'έναν ευριστικό (δηλαδή αυθαίρετο και πρακτικά σωστό) τρόπο. Μερικές κατευθυντήριες γραμμές για τον προσδιορισμό των κλάσεων αυτών είναι οι εξής:

- Αν μια συνθήκη εισόδου προδιαγράφει ένα διάστημα τιμών, τότε υπάρχει μια έγκυρη κλάση ισοδύναμων τιμών (το δεδομένο εισόδου ανήκει στο συγκεκριμένο διάστημα) και μια ή δυο άκυρες κλάσεις (το δεδομένο εισόδου δεν ανήκει στο συγκεκριμένο διάστημα).

[16] Αναφορά στο βιβλίο του Βασίλειου Βεσκούκη,Τεχνολογία Λογισμικού Ι, σελ **176.**

- Αν μια συνθήκη εισόδου προδιαγράφει πλήθος αλλά και τη μέγιστη τιμή του, τότε υπάρχει μια έγκυρη κλάση ισοδύναμων τιμών (το δεδομένο εισόδου να είναι μεγαλύτερο του μηδενός και το πολύ ίσο με τη μέγιστη τιμή του) και δυο άκυρες κλάσεις (το δεδομένο εισόδου δεν ανήκει στο προηγούμενο διάστημα).
- Αν μια συνθήκη προδιαγράφει ένα σύνολο από τιμές για τις οποίες υπάρχει η υπόνοια ότι η προς έλεγχο οντότητα τις μεταχειρίζεται διαφορετικά, τότε υπάρχουν τόσες έγκυρες κλάσεις ισοδύναμων τιμών όσες είναι οι τιμές του συνόλου και μια άκυρη κλάση με μια μόνο τιμή τέτοια που να είναι διαφορετική από τις τιμές του συνόλου αυτού.

Αφού, λοιπόν, προσδιοριστούν οι κλάσεις ισοδύναμων τιμών, το επόμενο βήμα είναι η επιλογή των περιπτώσεων ελέγχου, η οποία μπορεί να γίνει σύμφωνα με τα παρακάτω βήματα:

- Μέχρι να καλυφθούν όλες οι έγκυρες κλάσεις από περιπτώσεις ελέγχου, ορίζεται μια καινούρια περίπτωση ελέγχου που να καλύπτει όσο γίνεται περισσότερες από τις ακάλυπτες έγκυρες κλάσεις.
- Μέχρι να καλυφθούν οι άκυρες κλάσεις από περιπτώσεις ελέγχου, ορίζεται μια καινούρια περίπτωση ελέγχου που να καλύπτει μια και μόνο μια από τις ακάλυπτες άκυρες κλάσεις.

6.2.2.1 Προσέγγιση συνοριακών τιμών

Συνοριακές τιμές (boundary values) σε μια κλάση ισοδύναμων τιμών είναι οι τιμές των άκρων της. Έχει παρατηρηθεί ότι κατά την ανάπτυξη μιας εφαρμογής λογισμικού οι σχεδιαστές και οι προγραμματιστές τείνουν να λαμβάνουν υπόψη τους τυπικές τιμές για τα δεδομένα εισόδου, ενώ συνήθως παραβλέπουν τις συνοριακές τιμές. Η αντιμετώπιση αυτή συμβάλλει στη γέννηση πολλών σφαλμάτων πάνω στις συνοριακές τιμές, καθώς και σε τιμές πριν ή μετά από αυτές. Είναι, λοιπόν, μια καλή στρατηγική ελέγχου να ελέγχονται μεθοδικά οι τιμές αυτές.

Κάτι τέτοιο ισχύει όχι μόνο για τις κλάσεις ισοδύναμων τιμών εισόδου, αλλά και για τις κλάσεις ισοδύναμων τιμών εξόδου. Συμπληρώνουμε, λοιπόν, τις περιπτώσεις ελέγχου που παράγονται με την προσέγγιση της «ισοδύναμης διαμέρισης» σύμφωνα με τις ακόλουθες κατευθυντήριες γραμμές:

- Αν μια συνθήκη εισόδου προδιαγράφει ένα διάστημα τιμών, ορίζονται έγκυρες περιπτώσεις ελέγχου για τα άκρα του διαστήματος και άκυρες για τις τιμές ακριβώς έξω από τα άκρα.
- Αν μια συνθήκη εισόδου προδιαγράφει ένα πλήθος από τιμές, ορίζονται περιπτώσεις ελέγχου με το μικρότερο και το μεγαλύτερο πλήθος τιμών, καθώς και με ένα μικρότερο ή ένα μεγαλύτερο, αντίστοιχα.
- Αντίστοιχα ορίζονται περιπτώσεις ελέγχου για κάθε συνθήκη εξόδου.
- Αν η είσοδος ή η έξοδος είναι ένα διατεταγμένο σύνολο (π.χ. ένα ακολουθιακό αρχείο, μια γραμμική λίστα), ορίζονται περιπτώσεις ελέγχου για το πρώτο και το τελευταίο στοιχείο του συνόλου.

6.2.2.2 Προσέγγιση αιτίου-Αποτέσματος

Η επιλογή των περιπτώσεων ελέγχου στην προσέγγιση *αιτίου - αποτελέσματος* (cause and effect graphing) γίνεται με τη βοήθεια ενός γράφου. Ο γράφος αυτός στους αρχικούς του κόμβους έχει αίτια, στους ενδιάμεσους περιορισμούς και στους τελικούς αποτελέσματα. Με τη βοήθεια του κόμβου αυτού κατασκευάζεται ένας *πίνακας απόφασης* (decision table) από τον οποίο μπορούν μηχανιστικά να παραχθούν περιπτώσεις ελέγχου.

Το πλεονέκτημα της προσέγγισης αυτής είναι ότι θεωρεί συνδυασμούς συνθηκών εισόδου και οι παραγόμενες περιπτώσεις ελέγχου είναι πολύ αποτελεσματικές. Στην πραγματικότητα, όμως, ο γράφος γίνεται πολύ μεγάλος και άβολος, με αποτέλεσμα η όλη διεργασία εξαγωγής περιπτώσεων ελέγχου να είναι πολύ κοπιαστική.

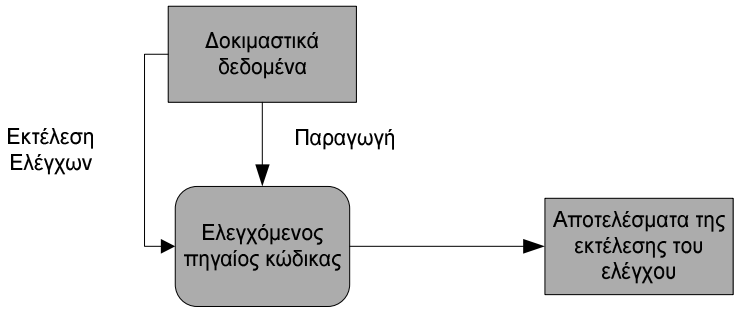
6.2.2.3 Προσέγγιση μαντέματος

Υπάρχουν πολλοί άνθρωποι οι οποίοι με τη διορατικότητα που τους διακρίνει επιλέγουν περιπτώσεις ελέγχου ικανές να αποκαλύπτουν σφάλματα στο λογισμικό. Σ’ αυτό ακριβώς το γεγονός βασίζεται η προσέγγιση του «μαντέματος». Βέβαια, στις περισσότερες περιπτώσεις η διορατικότητα που διακρίνει ένα άτομο είναι απόρροια των γνώσεων και της πείρας που διαθέτει. Αν και αυτή η μεθοδολογία είναι κάπως ανεπίσημη, δεν μπορούμε να την παραβλέψουμε, καθώς συχνά είναι ιδιαίτερα αποτελεσματική.

6.2.3 Στρατηγική του γυάλινου κουτιού

Στη στρατηγική αυτή το λογισμικό που ελέγχεται αντιμετωπίζεται ως ένα *άσπρο κουτί* (white box), ή καλύτερα ως ένα *γυάλινο κουτί* (glassbox). Αυτό, γιατί θεωρείται γνωστή (είναι ορατή) η δομή και ο τρόπος λειτουργίας του και δεν αποτελεί «μαύρο κουτί», όπως στην προηγούμενη περίπτωση. Η επιλογή των περιπτώσεων ελέγχου γίνεται μετά από μελέτη του κώδικα και της δομής τής υπό έλεγχο οντότητας. Απαραίτητα για την εργασία αυτή είναι το λεπτομερές σχέδιο της υπό έλεγχο μονάδας λογισμικού, καθώς και ο πηγαίος της κώδικας. Στο Σχήμα 6.7 δίνεται η ροή εργασιών κατά την εφαρμογή της «στρατηγικής του γυάλινου κουτιού».

Για την επιλογή περιπτώσεων ελέγχου, στη στρατηγική αυτή χρησιμοποιούνται διάφορα κριτήρια, τα οποία λέγονται «κριτήρια κάλυψης». Βέβαια, αυτονόητο είναι ότι όσο περισσότερες περιπτώσεις ελέγχου επιλέγονται για το κάθε κριτήριο, με τόσο περισσότερη ασφάλεια μπορεί να εγγυηθεί κάποιος ότι το έχει καλύψει. Μερικά χρήσιμα κριτήρια κάλυψης είναι τα ακόλουθα:



ΣΧΗΜΑ 6.7 *Στρατηγική του γυάλινου κουτιού*

- Όλα τα ανεξάρτητα μονοπάτια εκτέλεσης του ελεγχόμενου κώδικα πρέπει να εκτελούνται τουλάχιστον μια φορά. Δηλαδή δεν πρέπει να υπάρχουν γραμμές πηγαίου κώδικα των οποίων η εκτέλεση να μην ελέγχεται.
- Όλες οι διακλαδώσεις ροής με εντολές τύπου if . then . else πρέπει να εκτελούνται τουλάχιστον μια φορά, τόσο για την περίπτωση αληθούς συνθήκης όσο και ψευδούς.
- Όλοι οι βρόχοι επανάληψης θα πρέπει να εκτελούνται και για οριακές τιμές επαναλήψεων, αλλά και για αριθμό επαναλήψεων εντός των ορίων. Για παράδειγμα, ένας βρόγχος με μέγιστο αριθμό επαναλήψεων ν θα πρέπει να εκτελεστεί για 0, 1, 2, μ (όπου μ < ν), ν . 1, ν και ν + 1 επαναλήψεις.

Για να αποκαλυφθούν όλα τα σφάλματα της ελεγχόμενης οντότητας, πρέπει να δοκιμαστούν όλα τα δυνατά μονοπάτια εκτέλεσής της και όχι απλώς να εκτελεστεί μια φορά κάθε γραμμή προγράμματος. Εκτός τετριμμένων περιπτώσεων, κάτι τέτοιο είναι ανέφικτο, όπως συμβαίνει και με τη «στρατηγική του μαύρου κουτιού». Γενικά, η εμπειρία δείχνει ότι ο εξαντλητικός έλεγχος του «μαύρου κουτιού» είναι καλύτερος απ' αυτόν του «γυάλινου κουτιού».

Στην παραπάνω μονάδα προγράμματος οι μεταβλητές εισόδου είναι οι a, b και c. Έτσι, για κάθε περίπτωση ελέγχου θα πρέπει να επιλέξουμε μια τριάδα τιμών για τις μεταβλητές αυτές, οι οποίες και θα αποτελούν τα δοκιμαστικά δεδομένα εισόδου. Οι επιλογές αυτές θα πρέπει να είναι τέτοιες ώστε με το πέρας όλων των ελέγχων να έχουν εκτελεστεί τουλάχιστον μια φορά όλες οι εντολές της μονάδας, καθώς και όλες οι εντολές απόφασης, τόσο στην περίπτωση που οι συνθήκες έχουν τιμή αλήθειας όσο και στην περίπτωση που έχουν τιμή ψεύδους.

Για την επίδειξη της εφαρμογής της «στρατηγικής του γυάλινου κουτιού» δίνεται στη συνέχεια ένας πίνακας, η κάθε γραμμή του οποίου αντιστοιχεί στην εκτέλεση μιας περίπτωσης ελέγχου. Για κάθε τέτοια περίπτωση δίνονται το μονοπάτι εκτέλεσης εντολών (καταγράφονται οι γραμμές προγράμματος που εκτελούνται), καθώς και η κάλυψη συνθηκών, δηλαδή οι εντολές απόφασης που εκτελούνται και η τιμή της συνθήκης για καθεμιά απ'αυτές.

Δοκιμαστηκά δεδομένα A b c	Κάλυψη εντολών (μονοπάτι εκτέλεσης)	Κάλυψη συνθηκών (Εντολές απόφασης-τιμές συνθηκών)
0 1 2	1-2-3-6-12-18-19	3:F-T,6:F,12:F,18:T
1 3 1	1-2-3-4-6-7-8-9-10-11-19	3:F-F,4:T,6:T
4 -4 1	1-2-3-4-5-6-12-13-14-15-16-17-19	3:F-F,4:F,5-T,6-F,12-T

ΠΙΝΑΚΑΣ 6.8 *Εφαρμογή «στρατηγικής γυάλινου κουτιού»*

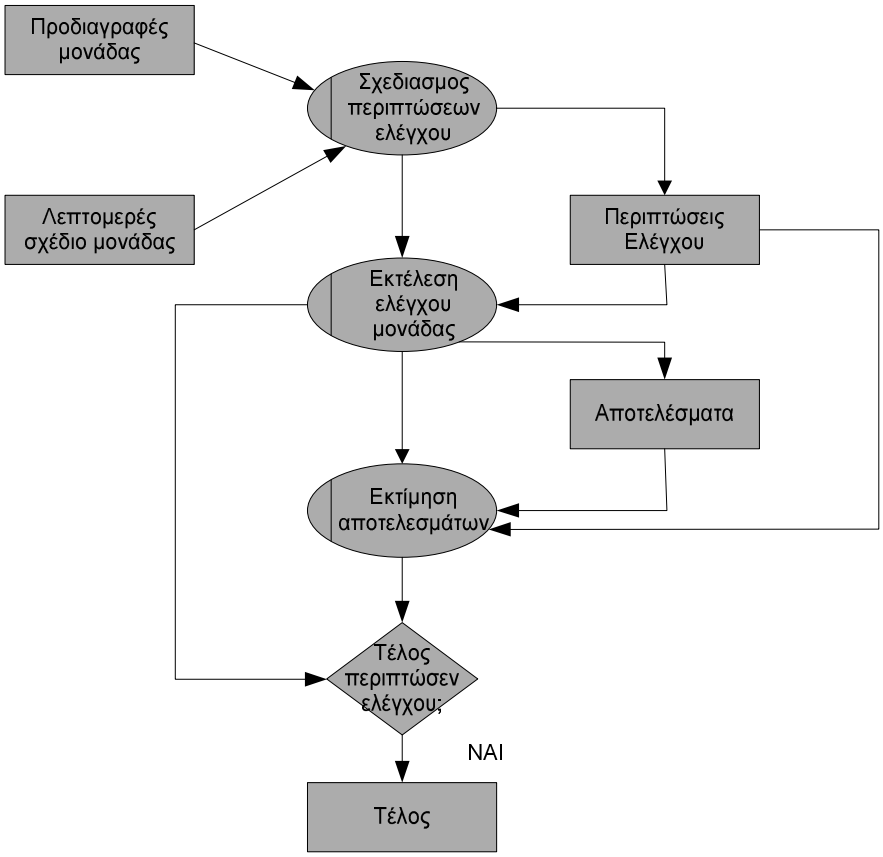
Όπως φαίνεται στον 6.8 πίνακα, οι εντολές της μονάδας που ελέγχεται εκτελούνται τουλάχιστον για μια φορά. Αντίθετα, οι συνθήκες σε ορισμένες από τις εντολές απόφασης δεν παίρνουν και τις δύο δυνατές τιμές τους. Μια ή και περισσότερες περιπτώσεις ελέγχου είναι απαραίτητες για να καλυφθεί και αυτό το κριτήριο και η επιλογή τους αφήνεται ως άσκηση για τον αναγνώστη.

6.3 Εκτέλεση ελέγχου

Ο έλεγχος μιας εφαρμογής λογισμικού πραγματοποιείται σε τέσσερα στάδια. Τα στάδια αυτά, από το κατώτερο επίπεδο στο ανώτερο, είναι ο *έλεγχος μονάδας* (unit testing), ο *έλεγχος συνένωσης* (integration testing), ο *έλεγχος συστήματος* (system testing) και ο *έλεγχος αποδοχής* (acceptance testing). Για τη συστηματική εκτέλεση του ελέγχου, προχωρούμε πάντα από το κατώτερο επίπεδο στο ανώτερο, καθώς η λογική που υιοθετείται είναι να ελέγχεται πρώτα το μέρος και μετά το όλο. Εξάλλου, αυτή είναι και η σειρά με την οποία κατασκευάζονται τα συστατικά του λογισμικού. Στο πρώτο επίπεδο ελέγχεται ο κώδικας μιας μονάδας του συστήματος, στο δεύτερο ο κώδικας περισσότερων της μιας μονάδας, στο τρίτο και τέταρτο ο κώδικας όλου του λογισμικού.

6.3.1 Έλεγχος μονάδας

Κατά τον *έλεγχο μονάδας* κάθε μονάδα του λογισμικού δοκιμάζεται μεμονωμένα με σκοπό να διαπιστωθεί αν πληροί τις προδιαγραφές της. Η διαδικασία αυτή περιγράφεται σχηματικά στο Σχήμα 6.9. Για τη διεκπεραίωση αυτού του επιπέδου ελέγχου μπορεί να χρησιμοποιηθεί οποιαδήποτε από τις στρατηγικές ελέγχου αναφέρθηκαν ή και οι δύο, προκειμένου να ελεγχθούν επιπλέον περιπτώσεις με σκοπό την εξασφάλιση της ορθής λειτουργίας της.



ΣΧΗΜΑ 6.9 *Ροή εργασιών κατά τον «έλεγχο μονάδας λογισμικού»*

Για την εκτέλεση του «ελέγχου μονάδας» χρησιμοποιούνται διάφορες τεχνικές. Οι τεχνικές αυτές διαφοροποιούνται ανάλογα με την επιλογή των περιπτώσεων ελέγχου, τα εργαλεία που μπορούν να χρησιμοποιηθούν, τη σειρά με την οποία κωδικοποιούνται και ελέγχονται οι μονάδες και με τις επιπτώσεις που έχουν στο κόστος παραγωγής των περιπτώσεων ελέγχου, καθώς και στο κόστος εντοπισμού και διόρθωσης των σφαλμάτων.

Η επιλογή της τεχνικής που θα χρησιμοποιηθεί έχει μεγάλη σημασία. Οι δύο κατηγορίες τέτοιων τεχνικών είναι η *αυξητική* (incremental) και η *μη αυξητική* (non.incremental). Ακολουθεί μια περιγραφή της μη αυξητικής τεχνικής, ενώ η περιγραφή της αυξητικής τεχνικής δίνεται μετά τον ορισμό του «ελέγχου συνένωσης», καθώς στην τεχνική αυτή ολοκληρώνονται και τα δύο πρώτα επίπεδα του ελέγχου.

Στη μη αυξητική τεχνική κάθε μονάδα ελέγχεται μεμονωμένα και εντελώς ανεξάρτητα από τις άλλες. Για τη διεκπεραίωση του «ελέγχου μονάδας» με αυτή την τεχνική, είναι απαραίτητη η δημιουργία καινούριων βοηθητικών μονάδων. Για κάθε μονάδα που ελέγχεται θα πρέπει να κατασκευάζεται μια βοηθητική μονάδα, η οποία θα την καλεί, θα της παρέχει τα δοκιμαστικά δεδομένα εισόδου και θα τυπώνει τα αποτελέσματα της εκτέλεσης σε

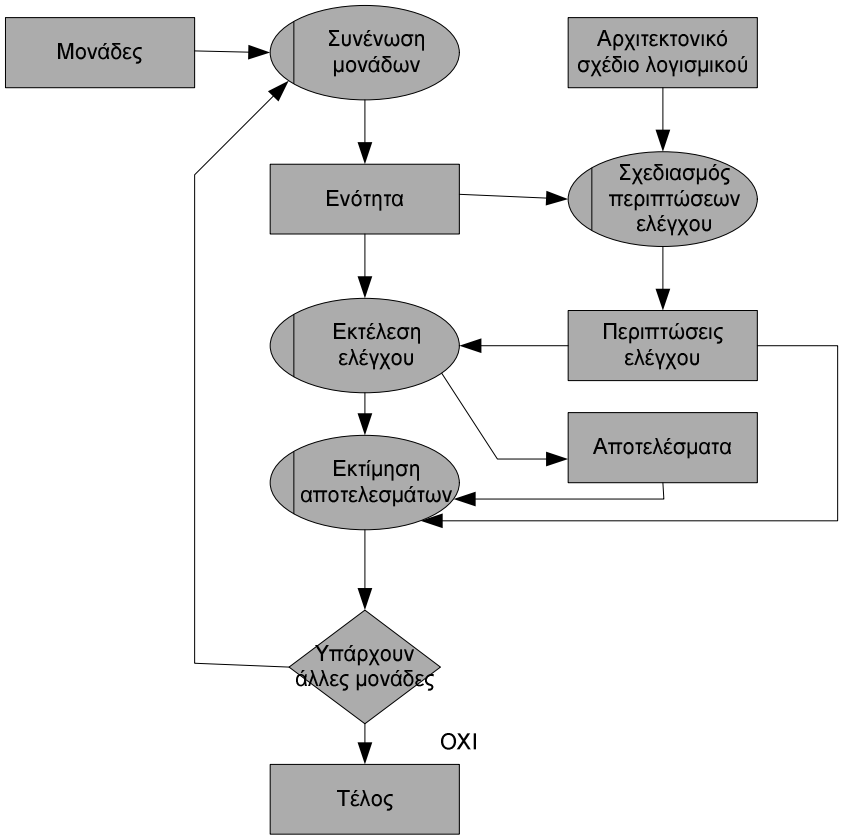
κάποια έξοδο. Αυτή η βοηθητική μονάδα ονομάζεται *οδηγός (*driver). Στην περίπτωση που η ελεγχόμενη μονάδα περιέχει κλήσεις σε άλλες μονάδες, τότε απαιτούνται και άλλες βοηθητικές μονάδες, οι οποίες θα πρέπει να κατασκευάζονται εκ των προτέρων, έτσι ώστε να κωδικοποιείται η επικεφαλίδα τους, να περνάνε τα ορίσματα και να επιστρέφουν κάποιες τυπικές τιμές. Αυτές οι καινούριες μονάδες ονομάζονται *στελέχη* (stubs).

Ο έλεγχος μονάδας είναι πολύ σημαντικός και δε θα πρέπει να παραλείπεται σε καμία περίπτωση. Μπορεί να αποκαλύψει σφάλματα τα οποία δεν είναι καθόλου εύκολο να εντοπιστούν στα επόμενα επίπεδα ελέγχου. Τέτοια μπορεί να είναι η χρησιμοποίηση κακών προγραμματιστικών τεχνικών (όπως η χρήση της εντολής goto), οι κακές επιδόσεις κ.ά.

6.3.1.1 Έλεγχος συνένωσης

Ο *έλεγχος συνένωσης* (integration testing) είναι η διαδικασία κατά την οποία ελέγχονται σύνθετα τμήματα, μέρη τού υπό ανάπτυξη λογισμικού. Ένα τέτοιο τμήμα περιλαμβάνει περισσότερες από μια μονάδες. Αυτό που ενδιαφέρει σ’αυτό το στάδιο του ελέγχου είναι να διαπιστωθεί αν οι μονάδες λογισμικού συνεργάζονται ομαλά κατά την ικανοποίηση κάποιας λειτουργικής απαίτησης.

Για τη σχεδίαση των περιπτώσεων ελέγχου στον «έλεγχο συνένωσης», χρησιμοποιείται και η «στρατηγική του άσπρου κουτιού», αλλά και η «στρατηγική του μαύρου κουτιού». Ο τρόπος με τον οποίο έχουν υλοποιηθεί οι διαπροσωπείες των μονάδων που ελέγχονται επηρεάζει τη διαδικασία αυτή. Η επιλογή των απαιτούμενων περιπτώσεων ελέγχου είναι απαραίτητο να εξασφαλίζει την εκτέλεση όλων των μονάδων της ελεγχόμενης ενότητας, καθώς και όλων των εντολών κλήσεων που περιέχονται στις μονάδες αυτές, τουλάχιστον μια φορά. Απαραίτητα, λοιπόν, έγγραφα για τη διεκπεραίωση του «ελέγχου συνένωσης» είναι το «αρχιτεκτονικό σχέδιο», καθώς και ο «κώδικας» και το «λεπτομερές σχέδιο των μονάδων». Μια σχηματική περιγραφή αυτού του επιπέδου του ελέγχου παρατίθεται στο Σχήμα 6.10.



ΣΧΗΜΑ 6.10 *Ροή εργασιών κατά τον «έλεγχο συνένωσης»*

Σε περίπτωση που ο έλεγχος μονάδας έχει γίνει σωστά με τη μη αυξητική τεχνική, όπως αυτή δόθηκε στην προηγούμενη παράγραφο, τότε δε χρειάζεται να γίνει «έλεγχος συνένωσης», καθώς ουσιαστικά θα έχει εκτελεστεί με τον «έλεγχο μονάδας». Σε αντίθετη περίπτωση, μπορούμε να διακρίνουμε πολλές τεχνικές για την εκτέλεση του «ελέγχου συνένωσης», η πιο αποτελεσματική από τις οποίες είναι η αυξητική τεχνική. Σύμφωνα με την αυξητική τεχνική, αφού κωδικοποιηθεί μια μονάδα, εκτελείται το πρώτο επίπεδο ελέγχου για αυτή (έλεγχος μονάδας). Στη συνέχεια ενώνεται με την ενότητα στην οποία ανήκει και εκτελείται ο «έλεγχος συνένωσης». Η διαδικασία επαναλαμβάνεται για κάθε καινούρια μονάδα που κωδικοποιείται, μέχρι να κατασκευαστεί ολόκληρο το λογισμικό.

Η αυξητική τεχνική μπορεί να εφαρμοστεί με δύο τρόπους: *από πάνω προς τα κάτω* (top.down incremental) ή *από κάτω προς τα πάνω* (bottom.up incremental), ανάλογα με τη σειρά με την οποία επιλέγεται να ελεγχθούν οι μονάδες του λογισμικού. Με αναφορά στο διάγραμμα δομής προγράμματος, στην περίπτωση που η κατασκευή του πηγαίου κώδικα των μονάδων αρχίσει από την κορυφή και συνεχίσει προς τα κάτω, έχουμε την προσέγγιση «από πάνω προς τα κάτω», ενώ στην αντίθετη περίπτωση έχουμε την προσέγγιση «από κάτω προς τα πάνω».

6.3.1.2 Πλεονεκτήματα και μειονεκτήματα

Ο τρόπος εκτέλεσης των δύο πρώτων επιπέδων του ελέγχου με την αυξητική «από πάνω προς τα κάτω» τεχνική έχει το μειονέκτημα ότι πρέπει να κατασκευάζονται στελέχη για τις μονάδες του λογισμικού που ακόμη δεν έχουν κατασκευαστεί. Απ'την άλλη όμως, έχει πολλά πλεονεκτήματα, όπως:

- Ο «έλεγχος συνένωσης» και ο «έλεγχος μονάδας» εκτελούνται με πολύ πιο βολικό και αξιόπιστο τρόπο. Για παράδειγμα, όταν μια μονάδα κωδικοποιηθεί και ελεγχθεί κατά τα δύο πρώτα επίπεδα ελέγχου, τότε υπάρχουν πολύ λίγα πιθανά σημεία εμφάνισης σφαλμάτων στις διαπροσωπείες μεταξύ των μονάδων. Εφόσον η ενό-τητα πριν από την ένωση της τελευταίας μονάδας έχει ελεγχθεί διεξοδικά, κάποιο σφάλμα που θα αποκαλυφθεί θα πρέπει να βρίσκεται σε κάποια από τις διαπροσωπείες της τελευταίας μονάδας.
- Είναι εφικτή η διαθεσιμότητα εκδόσεων του υπό ανάπτυξη λογισμικού κατά τη φάση του ελέγχου, στις οποίες είναι δυνατή η εκτέλεση μόνο ενός μέρους του συνόλου των λειτουργιών που προδιαγράφονται για το σύστημα. Αυτό είναι ένα σημαντικό γεγονός, και όντως πολλές εταιρείες παραδίδουν το λογισμικό τους με έναν αριθμό διαδοχικών τέτοιων εκδόσεων. Η λογική που ακολουθείται είναι στην πρώτη απ'αυτές τις εκδόσεις του συστήματος που παραδίδεται να έχουν υλοποιηθεί οι σημαντικότερες από τις λειτουργίες που πρέπει να εκτελεί το λογισμικό και σε καθεμιά από τις επόμενες εκδόσεις να προστίθενται όλο και περισσότερες.

Το βασικότερο πλεονέκτημα της προσέγγισης «από κάτω προς τα πάνω» είναι ότι τα σφάλματα στις μονάδες που βρίσκονται στα χαμηλότερα επίπεδα του γράφου κλήσης είναι εύκολο να αποκαλυφθούν. Απ'την άλλη, το βασικότερο μειονέκτημά της είναι ότι για την ορθή εκτέλεσή της είναι απαραίτητες μονάδες λογισμικού που εξομοιώνουν τις συνθήκες εκτέλεσης των ελεγχόμενων μονάδων, οι οποίες είναι σαφώς πιο πολύπλοκες από τους οδηγούς προγράμματος, στους οποίους αναφερθήκαμε παραπάνω. Αφού ολοκληρωθεί και ο «έλεγχος συνένωσης», περνάμε στο επόμενο επίπεδου του ελέγχου, το οποίο είναι ο «έλεγχος συστήματος».

6.3.1.3 Έλεγχος συστήματος

Στον «έλεγχο συστήματος» η οντότητα που ελέγχεται είναι πλέον ολόκληρη η εφαρμογή λογισμικού. Στο επίπεδο αυτό του ελέγχου εκτελούνται δοκιμές και γίνονται διάφορες επιδείξεις και αναλύσεις, με σκοπό να διαπιστωθεί αν το σύστημα που κατασκευάστηκε πληροί τις προδιαγραφές του. Δεν πρέπει να ξεχνάμε ότι πάντα ο στόχος σ'αυτή τη φάση ανάπτυξης του συστήματος είναι η αποκάλυψη σφαλμάτων.

Σ'αυτό το επίπεδο ελέγχου χρησιμοποιείται η «στρατηγική του μαύρου κουτιού» για την επιλογή των περιπτώσεων ελέγχου. Οι περιπτώσεις αυτές θα πρέπει να είναι τέτοιες ώστε να ελέγχονται όλες οι απαιτήσεις που περιγράφονται στο «έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό», δηλαδή όλες οι λειτουργικές και οι μη λειτουργικές απαιτήσεις. Για την ικανοποίηση ή όχι των λειτουργικών απαιτήσεων, οι έλεγχοι καθορίζονται από το χαρακτήρα των εκάστοτε απαιτήσεων. Για τις μη λειτουργικές απαιτήσεις χαρακτηριστικοί έλεγχοι που εκτελούνται είναι οι παρακάτω:

- **Έλεγχοι επίδοσης** (Performance testing). Στις δοκιμές αυτές το σύστημα δοκιμάζεται με ένα σταθερό φορτίο, με σκοπό το συντονισμό και τη βελτιστοποίηση των επιδόσεων του λογισμικού. Οι μετρήσεις που γίνονται συνήθως περιλαμβάνουν τον αριθμό των συναλλαγών, τον αριθμό των χρηστών, το μέγεθος των βάσεων δεδομένων στις οποίες γίνεται πρόσβαση κ.ά.
- **Έλεγχοι πίεσης** (Stress testing). Στις δοκιμές αυτές ελέγχονται διάφορες λειτουργίες του λογισμικού κατά την εμφάνιση ανώμαλων συνθηκών στην εκτέλεση του συστήματος. Μια τέτοια συνθήκη ορίζει η στιγμιαία εμφάνιση ενός υψηλού φορτίου, όπως, για παράδειγμα, η ταυτόχρονη εκδήλωση πολλών γεγονότων σ'ένα σύστημα αυτόματου ελέγχου, καθώς και η έλλειψη μνήμης ή διάφορων υλικών πόρων του συστήματος.
- **Έλεγχοι όγκου δεδομένων** (Volume testing). Σ'αυτές τις δοκιμές ελέγχεται η δυνατότητα του λογισμικού να δουλεύει με υψηλά φορτία τα οποία όμως δεν εμφανίζονται στιγμιαία. Αυτοί οι μεγάλοι όγκοι δεδομένων μπορεί να είναι είτε δεδομένα εισόδου είτε δεδομένα εξόδου είτε δεδομένα τα οποία είναι ήδη εγκατεστημένα σε κάποια βάση δεδομένων του λογισμικού. Ένα παράδειγμα ενός τέτοιου ελέγχου σε μια εφαρμογή λογισμικού τραπεζικών συναλλαγών είναι η πραγματοποίηση συναλλαγών από ολόκληρο τον πληθυσμό μιας χώρας.

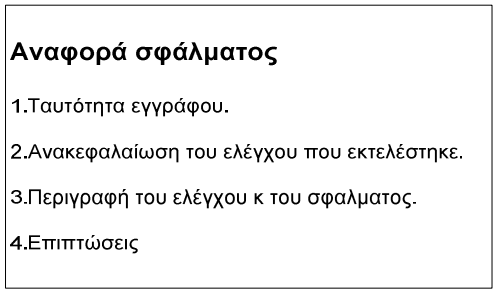
- **Έλεγχοι ασφάλειας** (Security testing). Οι δοκιμές αυτές ελέγχουν κατά πόσο οι πληροφορίες που περιέχει ένα λογισμικό σύστημα είναι προσβάσιμες από άτομα στα οποία δε θα πρέπει να παρέχεται δυνατότητα πρόσβασης στο σύστημα.
- **Έλεγχοι ανάκτησης** (Recovery testing). Στις δοκιμές αυτές ελέγχεται η δυνατότητα του λογισμικού να ανακτά τη λειτουργία του ύστερα από αιφνίδια σταματήματα. Για την εκτέλεση τέτοιων ελέγχων, το λογισμικό σύστημα εξαναγκάζεται σε αποτυχία με διαφόρους τρόπους και επιβεβαιώνεται κατά πόσο είναι ικανό να ανακτά την πλήρη λειτουργία του.

6.3.1.4 Έλεγχος αποδοχής

Ο «έλεγχος αποδοχής» είναι ένα υποσύνολο του «ελέγχου συστήματος». Δηλαδή ακολουθείται η ίδια λογική και τα απαραίτητα έγγραφα για τη διεκπεραίωσή του είναι επίσης οι προδιαγραφές των απαιτήσεων από το λογισμικό. Στην περίπτωση αυτή οι περιπτώσεις ελέγχος είναι ένα υποσύνολο των αντίστοιχων περιπτώσεων του «ελέγχου συστήματος» και για την επιλογή τους είναι αποκλειστικά υπεύθυνος ο πελάτης, στις εγκαταστάσεις του οποίου μπορεί και να πραγματοποιείται ο «έλεγχος αποδοχής». Ο στόχος αυτού του επιπέδου ελέγχου είναι να πειστεί ο πελάτης ότι το λογισμικό που κατασκευάστηκε για λογαριασμό του πληροί τις προδιαγραφές που τέθηκαν, η δε διάρκεια αυτού του ελέγχου μπορεί να είναι αρκετά μεγάλη, μέχρις ότου ικανοποιηθεί ο πελάτης. Στην πράξη δεν είναι λίγες οι περιπτώσεις όπου ο «έλεγχος συστήματος» ταυτίζεται με τον «έλεγχο αποδοχής». Τότε, ισχύει μία από τις τρεις ακόλουθες περιπτώσεις: (α) Ο «έλεγχος συστήματος» δεν είναι πλήρης, (β) ο «έλεγχος αποδοχής» είναι ιδιαίτερα επίπονος για τον πελάτη ή (γ) αυτό που εκτελείται βρίσκεται κάπου στο ενδιάμεσο και τα αποτελέσματά του δεν είναι επαρκή και αξιόπιστα.

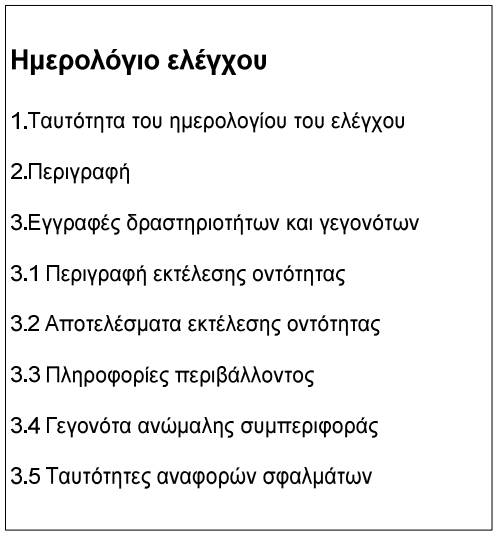
6.4 Αναφορές ελέγχου

Οι *αναφορές σφαλμάτων* είναι ένα σημαντικό τμήμα της τεκμηρίωσης του ελέγχου. Η σύνταξη αυτών των αναφορών αποτελεί το τελευταίο στάδιο της διαδικασίας του ελέγχου (Σχήμα 6.3). Ο σκοπός μιας τέτοιας αναφοράς είναι να καταγράψει με λεπτομέρεια ένα γεγονός που συμβαίνει κατά την εκτέλεση του ελέγχου, το οποίο υποδεικνύει την ύπαρξη ενός σφάλματος, ώστε να συμβάλλει στην αποτελεσματική διόρθωσή του. Μια τέτοια αναφορά συντάσσεται για κάθε σφάλμα το οποίο αποκαλύπτεται κατά την εκτέλεση ενός ελέγχου. Μια δομή της αναφοράς σφάλματος προτείνεται στο Σχήμα 6.11.



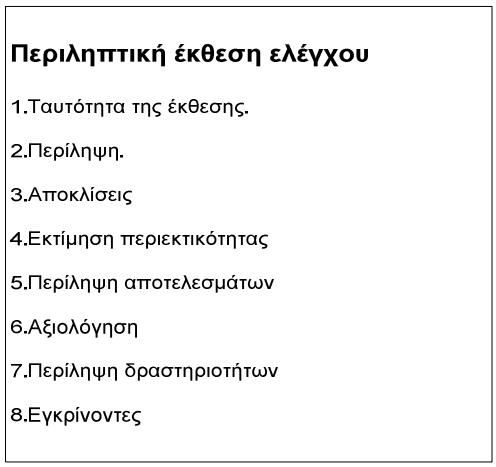
ΣΧΗΜΑ 6.11 *Υπόδειγμα δομής «αναφοράς σφάλματος»*

Δύο ακόμη σημαντικά έγγραφα για την ολοκληρωμένη τεκμηρίωση του ελέγχου είναι το *ημερολόγιο ελέγχου* (Test log) και η *περιληπτική έκθεση του ελέγχου* (Test.Summary report). Στο «ημερολόγιο ελέγχου» καταγράφονται πληροφορίες σχετικές με την εκτέλεση των ελέγχων και μπορεί να φανεί ιδιαίτερα χρήσιμο κατά τη σύνταξη των αναφορών σφαλμάτων, καθώς επίσης και κατά την εκτέλεση μελλοντικών ελέγχων. Το καταλληλότερο χρονικό πλαίσιο για τη σύνταξή του είναι παράλληλα με την εκτέλεση του ελέγχου. Ακολούθως, στο Σχήμα 6.12 παρουσιάζεται μια δομή του «ημερολογίου ελέγχου».



ΣΧΗΜΑ 6.12 *Το «ημερολόγιο ελέγχου»*

Τέλος, η «περιληπτική έκθεση του ελέγχου» συνοψίζει τις διαδικασίες που επιτελέστηκαν κατά την εκτέλεσή του και παρουσιάζει εκτιμήσεις για την περιεκτικότητά του. Επίσης, περιλαμβάνει και μια αξιολόγηση του όλου ελέγχου. Στο Σχήμα 6.13 παρουσιάζεται μια δομή της «περιληπτικής έκθεσης του ελέγχου».



Σχήμα 6.13 *Δομή «περιληπτικής έκθεσης ελέγχου»*

6.5 Διόρθωση σφαλμάτων

Μέχρι τώρα, στο παρόν κεφάλαιο ασχοληθήκαμε με τον έλεγχο, δηλαδή με τη διαδικασία αποκάλυψης σφαλμάτων στο λογισμικό. Η εργασία αυτή όμως απλώς επιβεβαιώνει την ύπαρξη τέτοιων σφαλμάτων, καθώς συνήθως δεν καθορίζει την ακριβή θέση ή την υφή τους. Το επόμενο βήμα για την ορθή ανάπτυξη του λογισμικού συστήματος

είναι η διόρθωση των σφαλμάτων που βρέθηκαν κατά τον έλεγχο. Για το σκοπό αυτό χρησιμοποιούνται *ειδικές τεχνικές* (debugging techniques).

Για την αποτελεσματική εφαρμογή των τεχνικών αυτών είναι απαραίτητη η συλλογή όσο το δυνατόν περισσότερων στοιχείων για το κάθε σφάλμα. Τέτοια στοιχεία μπορούν να συγκεντρωθούν με τη χρησιμοποίηση διαφόρων μεθόδων, όπως:

- Με την εισαγωγή εντολών για την εκτύπωση διαγνωστικών μηνυμάτων, στον κώδικα του λογισμικού. Τέτοια μηνύματα δίνουν τις τιμές των μεταβλητών σε ορισμένα κρίσιμα σημεία του προγράμματος και επιβεβαιώνουν ότι η εκτέλεσή του έχει περάσει από τα σημεία αυτά. Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, αυτή η μέθοδος κωδικοποίησης ονομάζεται «αμυντικός προγραμματισμός».
- Με τη μελέτη αντιγράφων της μνήμης σε δυαδική ή συμβολική μορφή. Τέτοια αντίγραφα παρέχουν πληροφορίες για την κατάσταση της εκτέλεσης του λογισμικού σε διάφορα κρίσιμα σημεία.
- Με τη χρησιμοποίηση *ιχνών* (traces). Τα «ίχνη» καταγράφουν τις εντολές όπως αυτές εκτελούνται, καθώς και τις τιμές προκαθορισμένων μεταβλητών ανά πάσα στιγμή.
- Με την τοποθέτηση *σημείων διακοπής* (breakpoints) του προγράμματος. Στα σημεία αυτά διακόπτεται η εκτέλεση του προγράμματος και ο έλεγχος δίνεται στο τερματικό, από όπου ο χρήστης μπορεί να δει τις τιμές των μεταβλητών, των καταχωριτών,να αλλάξει τις τιμές αυτές κτλ.

Ο έλεγχος και η διόρθωση των σφαλμάτων είναι δυο πολύ σημαντικά στάδια της ανάπτυξης ενός λογισμικού συστήματος. Σχετικές μελέτες έχουν αποδείξει ότι οι δύο αυτές εργασίες μπορεί να κοστίσουν μέχρι και το 50% του προϋπολογισμού για την κατασκευή μιας εφαρμογής λογισμικού. Η επιτυχημένη εκτέλεσή τους συμβάλλει ώστε ο εκάστοτε κατασκευαστής να κερδίζει τόσο σε χρήμα όσο και σε αξιοπιστία.

ΚΕΦΑΛΑΙΟ 7: Συντήρηση Συστήματος

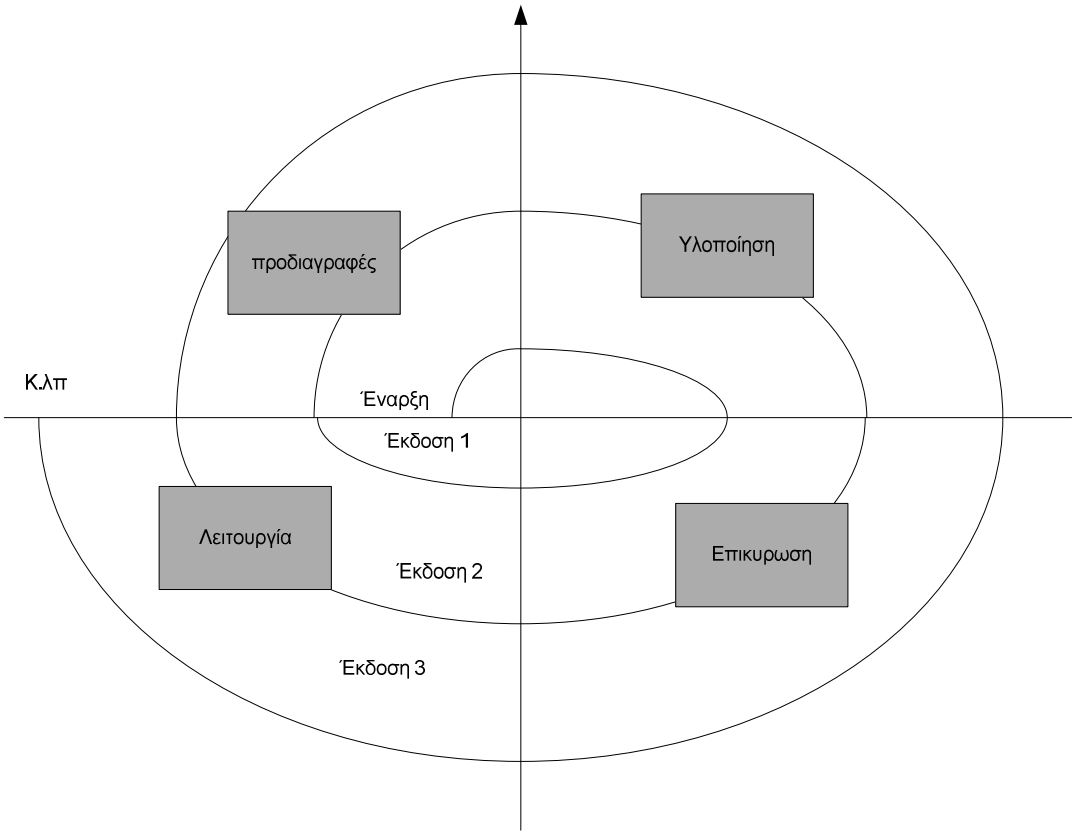
Μετά τη φάση της διανομής ενός συστήματος, αυτό αναπόφευκτα θα πρέπει να τροποποιηθεί αν πρόκειται να παραμείνει χρήσιμο. Από τη στιγμή που το λογισμικό αρχίσει να χρησιμοποιείται, εμφανίζονται νέες απαιτήσεις ενώ υπάρχουσες απαιτήσεις μεταβάλλονται. Επιχειρηματικές αλλαγές μπορεί να έχουν ως αποτέλεσμα νέες απαιτήσεις. Τμήματα του λογισμικού ίσως χρειαστούν τροποποίηση με σκοπό τη διόρθωση σφαλμάτων που εντοπίστηκαν κατά τη λειτουργία τους, για την προσαρμογή του συστήματος σε μια νέα πλατφόρμα, και για τη βελτίωση της απόδοσής του ή άλλων μη λειτουργικών χαρακτηριστικών. Συνεπώς, η ανάπτυξη του λογισμικού δε σταματάει με την παράδοση του συστήματος, αλλά συνεχίζεται για όλη τη διάρκεια ζωής του.

Η εξέλιξη λογισμικού (software evolution) αποτελεί σημαντικό τομέα, καθώς σήμερα οι οργανισμοί εξαρτώνται ολοκληρωτικά από τα συστήματα λογισμικού που διαθέτουν και έχουν επενδύσει σε αυτά εκατομμύρια δολάρια. Τα συστήματά τους αποτελούν κρίσιμα επιχειρηματικά περιουσιακά στοιχεία, και έτσι οι εταιρείες είναι υποχρεωμένες να επενδύσουν στη μεταβολή τους ώστε να διατηρήσουν την αξία τους. Ως αποτέλεσμα, το μεγαλύτερο μέρος του προϋπολογισμού που αφορά το λογισμικό σε μεγάλες εταιρείες είναι αφιερωμένο στη συντήρηση υφισταμένων συστημάτων, και δε θα πρέπει να μας εκπλήσσουν νούμερα όπως αυτά του Erlikh[11] (Erlikh, 2000) που υποδεικνύουν ότι το 90% του κόστους του λογισμικού είναι κόστος εξέλιξης. Ωστόσο, το συγκεκριμένο ποσοστό είναι αρκετά αβέβαιο, καθώς οι άνθρωποι εννοούν διαφορετικά πράγματα όταν αναφέρονται σε κόστος εξέλιξης ή συντήρησης.

Οι αλλαγές που πραγματοποιούνται μετά τη διανομή ενός συστήματος δεν αφορούν απλώς την επιδιόρθωση σφαλμάτων στο λογισμικό. Η πλειονότητα των αλλαγών είναι συνέπεια νέων απαιτήσεων, οι οποίες δημιουργούνται από επιχειρηματικές αλλαγές και μεταβολές των αναγκών των χρηστών. Επομένως, θα μπορούσαμε να φανταστούμε την τεχνολογία λογισμικού ως μία σπειροειδή διαδικασία με καθορισμό απαιτήσεων, σχεδιασμό, υλοποίηση, και δοκιμές, η οποία συνεχίζεται καθ’ όλη τη διάρκεια ζωής του συστήματος. Αυτό φαίνεται στην Εικόνα 7.1. Ξεκινάει με τη δημιουργία της Έκδοσης 1 του συστήματος, μετά την παράδοση της οποίας προτείνονται αλλαγές και αρχίζει σχεδόν αμέσως η ανάπτυξη της Έκδοσης 2. Για την ακρίβεια, η ανάγκη για εξέλιξη μπορεί να γίνει προφανής ακόμη και πριν τη διανομή του συστήματος, έτσι ώστε μετέπειτα εκδόσεις του λογισμικού να είναι υπό ανάπτυξη πριν καν κυκλοφορήσει η αρχική έκδοση.

Αυτό το μοντέλο εξέλιξης λογισμικού είναι εξιδανικευμένο, και μπορεί να εφαρμοστεί σε περιστάσεις όπου μία μόνο εταιρεία είναι υπεύθυνη τόσο για την αρχική ανάπτυξη του λογισμικού όσο και για την εξέλιξή του. Τα περισσότερα γενικά προϊόντα λογισμικού αναπτύσσονται με τη χρήση αυτής της προσέγγισης. Όταν όμως το λογισμικό κατασκευάζεται κατά παραγγελία, η ανάπτυξή του μπορεί να γίνει από κάποιον εξωτερικό συνεργάτη και η εξέλιξή του να αποτελεί ευθύνη του προσωπικού ανάπτυξης του πελάτη. Εναλλακτικά, ο χρήστης θα μπορούσε να συνάψει ξεχωριστή σύμβαση με μια εξωτερική εταιρεία για την υποστήριξη και την εξέλιξη του συστήματος.

Σε αυτές τις περιπτώσεις, η ομαλή ροή της σπειροειδούς διαδικασίας συχνά διακόπτεται. Ίσως τα έγγραφα απαιτήσεων και σχεδιασμού να μην έχουν μεταβιβαστεί από τη μία εταιρεία στην άλλη. Εταιρείες συγχωνεύονται ή αναδιοργανώνονται και κληρονομούν λογισμικό από άλλες εταιρείες, διαπιστώνοντας στη συνέχεια ότι αυτό χρειάζεται τροποποιήσεις. Όταν η μετάβαση από την ανάπτυξη στην εξέλιξη δεν είναι ομαλή, η διαδικασία μεταβολής του λογισμικού μετά την παράδοσή του συχνά ονομάζεται συντήρηση λογισμικού (software maintenance).



ΕΙΚΟΝΑ 7.1 *Σπειροειδές μοντέλο ανάπτυξης και εξέλιξης*

7.1 Δυναμική της εξέλιξης των προγραμμάτων

Ο όρος «δυναμική της εξέλιξης προγραμμάτων» αφορά τη μελέτη της μεταβολής ενός συστήματος. Το μεγαλύτερο μέρος του έργου σε αυτόν τον τομέα έχει πραγματοποιηθεί από τους Lehman και Beledy[18], αρχικά στις δεκαετίες του 1970 και 1980 (Lehman και Belady, 1985). Η εργασία συνεχίστηκε στη δεκαετία του 1990, καθώς ο Lehman και άλλοι ερεύνησαν τη σημασία της ανατροφοδότησης (feedback) στις διαδικασίες της εξέλιξης (Lehman 1996 [17],-Lehman[19],κ.α 1998-Lehman[20],κ.α. 2001).Αυτές οι μελέτες είχαν ως αποτέλεσμα ένα σύνολο νόμων (νόμοι του Lehman) οι οποίοι αφορούν τη μεταβολή των συστημάτων. Οι ερευνητές ισχυρίζονται ότι οι εν λόγω νόμοι (στην πραγματικότητα πρόκειται για υποθέσεις) είναι αμετάβλητοι και έχουν ευρεία εφαρμογή. Οι Lehman και Belady[18] εξέτασαν την ανάπτυξη και την εξέλιξη μιας σειράς μεγάλων συστημάτων λογισμικού. Οι προτεινόμενοι νόμοι, που φαίνονται στην εικόνα 7.2, ήταν αποτέλεσμα αυτών των μετρήσεων.

Ο πρώτος νόμος δηλώνει ότι η συντήρηση συστημάτων είναι μια αναπόφευκτη διαδικασία. Καθώς το περιβάλλον του συστήματος μεταβάλλεται, εμφανίζονται νέες απαιτήσεις και το σύστημα πρέπει να τροποποιηθεί. Όταν το τροποποιημένο σύστημα επανεισαχθεί στο περιβάλλον, προκαλεί επιπλέον περιβαλλοντικές αλλαγές, οπότε η διαδικασία της εξέλιξης ανακυκλώνεται.

Ο δεύτερος νόμος δηλώνει ότι καθώς ένα σύστημα αλλάζει, η δομή του υποβαθμίζεται. Ο μόνος τρόπος για να αποφευχθεί αυτό είναι η επένδυση στην προληπτική συντήρηση, όπου χρειάζεται να δαπανηθεί χρόνος για τη βελτίωση της δομής του λογισμικού χωρίς να προστεθεί νέα λειτουργικότητα. Προφανώς, κάτι τέτοιο σημαίνει επιπλέον κόστος, το οποίο προστίθεται στο κόστος για την υλοποίηση των επιβεβλημένων αλλαγών στο σύστημα.

Ο τρίτος νόμος είναι ίσως ο πιο ενδιαφέρων και ο πλέον επίμαχος των νόμων του Lehman. Δηλώνει ότι τα μεγάλα συστήματα διαθέτουν μια εγγενή δυναμική η οποία εδραιώνεται σε κάποιο αρχικό στάδιο της διαδικασίας ανάπτυξης. Η δυναμική αυτή προσδιορίζει τις γενικές τάσεις της διαδικασίας συντήρησης του συστήματος, και περιορίζει το πλήθος των πιθανών αλλαγών του. Οι Lehman και Belady[18] αναφέρουν ότι αυτός ο νόμος αποτελεί συνέπεια δομικών παραγόντων οι οποίοι επηρεάζουν και περιορίζουν τις αλλαγές των συστημάτων, καθώς και επιχειρησιακών παραγόντων που επηρεάζουν τη διαδικασία εξέλιξης.

Όταν ένα σύστημα υπερβεί κάποιο ελάχιστο μέγεθος, είναι πιο δύσκολο να τροποποιηθεί. Καθώς το σύστημα είναι μεγάλο και πολύπλοκο, είναι δύσκολο να κατανοηθεί,

και οι προγραμματιστές έχουν περισσότερες πιθανότητες να κάνουν λάθη εισάγοντας ελαττώματα στο σύστημα. Συνεπώς, ένας τρόπος για να αποφευχθεί η υποβάθμιση της αξιοπιστίας του συστήματος είναι η πραγματοποίηση μικρών αλλαγών. Μια μεγάλη αλλαγή θα μπορούσε να προκαλέσει πολλά νέα σφάλματα, τα οποία θα επισκιάσουν τις χρήσιμες αλλαγές που παραδίδονται στη νέα έκδοση του συστήματος.

Τα μεγάλα συστήματα συνήθως δημιουργούνται από μεγάλους οργανισμούς, με εσωτερικούς γραφειοκρατικούς μηχανισμούς οι οποίοι καθορίζουν τον προϋπολογισμό των αλλαγών για κάθε σύστημα και ελέγχουν τη διαδικασία λήψης αποφάσεων. Οι οργανισμοί πρέπει να λαμβάνουν αποφάσεις για τους κινδύνους και τα οφέλη των αλλαγών, καθώς και το κόστος που αυτές ενέχουν. Τέτοιες αποφάσεις χρειάζονται χρόνο να ληφθούν. Κατά τη διάρκεια αυτού του χρονικού διαστήματος, είναι πιθανό να προταθούν άλλες αλλαγές για το σύστημα, με υψηλότερη προτεραιότητα. Οι αρχικές αλλαγές ίσως χρειαστεί να αναβληθούν επ' αόριστον για το μέλλον. Έτσι, οι διαδικασίες λήψης αποφάσεων του οργανισμού διέπουν το ρυθμό των αλλαγών του συστήματος.

Νόμος	Περιγραφή
Συνεχής μεταβολή	Ένα πρόγραμμα το οποίο χρησιμοποιείται σε πραγματικό περιβάλλον λειτουργίας πρέπει αναγκαστικά να υποστεί αλλαγές, διαφορετικά θα γίνεται όλο και λιγότερο χρήσιμο στο συγκεκριμένο περιβάλλον.
Αυξανόμενη πολυπλοκότητα	Καθώς μεταβάλλεται ένα εξελισσόμενο πρόγραμμα, η δομή του τείνει να γίνεται πιο σύνθετη. Πρέπει λοιπόν να αφιερωθούν επιπλέον πόροι στη διατήρηση και την απλοποίηση της δομής.
Εξέλιξη μεγάλων προγραμμάτων	Η εξέλιξη ενός προγράμματος αποτελεί μια αυτορυθμιζόμενη διαδικασία. Ιδιότητες του συστήματος όπως το μέγεθος, το χρονικό διάστημα μεταξύ των εκδόσεων, και ο αριθμός των αναφερόμενων λαθών είναι κατά προσέγγιση αμετάβλητες σε κάθε έκδοση ενός συστήματος.
Επιχειρηματική σταθερότητα	Κατά τη διάρκεια ζωής ενός συστήματος, ο

	ρυθμός ανάπτυξής του είναι κατά προσέγγιση σταθερός και ανεξάρτητος από τους πόρους που είναι αφιερωμένοι στην ανάπτυξη του συστήματος.
Διατήρηση οικείων χαρακτηριστικών	Κατά τη διάρκεια ζωής ενός συστήματος, η βαθμιαία μεταβολή που λαμβάνει χώρα σε κάθε έκδοση είναι κατά προσέγγιση σταθερή.
Συνεχής αύξηση λειτουργικότητας	Η λειτουργικότητα που παρέχεται από ένα σύστημα πρέπει να αυξάνεται συνεχώς, ώστε οι χρήστες να παραμένουν ικανοποιημένοι.
Υποβάθμιση της ποιότητας	Η ποιότητα ενός συστήματος θα φαίνεται ότι υποβαθμίζεται, εκτός και αν το σύστημα αυτό προσαρμόζεται σε αλλαγές του λειτουργικού του περιβάλλοντος.
Σύστημα ανατροφοδότησης	Οι διαδικασίες εξέλιξης εμπεριέχουν πολυπρακτορικά (multi-agent) και πολυβροχικά (multi-loop) συστήματα ανατροφοδότησης, και να είναι σε θέση να επιτύχουν σημαντικές βελτιώσεις στα προϊόντα πρέπει να αντιμετωπίζονται ως συστήματα ανατροφοδότησης.

ΕΙΚΟΝΑ 7.2 *Οι νόμοι του Lehman*

Ο τέταρτος νόμος του Lehman δηλώνει ότι τα περισσότερα μεγάλα έργα προγραμματισμού λειτουργούν σε αυτό που εκείνος ονομάζει κορεσμένη κατάσταση. Δηλαδή, οι επιπτώσεις που έχουν στη μακροπρόθεσμη εξέλιξη του συστήματος αλλαγές στους πόρους ή το προσωπικό είναι ανεπαίσθητες. Αυτό συνάδει με τον τρίτο νόμο, που δηλώνει ότι η εξέλιξη ενός προγράμματος είναι σε μεγάλο βαθμό ανεξάρτητη από διαχειριστικές αποφάσεις. Ο νόμος επιβεβαιώνει ότι οι μεγάλες ομάδες ανάπτυξης λογισμικού είναι συχνά αντιπαραγωγικές λόγω των επικοινωνιακών επιβαρύνσεων που κυριαρχούν στην εργασία τους.

Ο πέμπτος νόμος του Lehman αφορά τις επαυξήσεις των μεταβολών σε κάθε έκδοση του συστήματος. Η προσθήκη νέας λειτουργικότητας σε ένα σύστημα αναπόφευκτα εισάγει

νέα σφάλματα. Όσο μεγαλύτερη λειτουργικότητα προστίθεται σε κάθε έκδοση, τόσο περισσότερα σφάλματα θα υπάρχουν. Έτσι, μια μεγάλη επαύξηση στη λειτουργικότητα κάποιας έκδοσης σημαίνει ότι αυτή η έκδοση θα πρέπει να ακολουθηθεί από μια περαιτέρω έκδοση, η οποία θα διορθώνει τα νέα σφάλματα που παρουσιάστηκαν στο σύστημα. Αυτή η νέα έκδοση θα περιέχει σχετικά μικρή ποσότητα νέας λειτουργικότητας. Ο νόμος σημαίνει ότι στο κόστος μιας μεγάλης επαύξησης στη λειτουργικότητα του συστήματος θα πρέπει πάντα να συνυπολογίζεται η ανάγκη για την επιδιόρθωση των σφαλμάτων.

Οι πέντε νόμοι συμπεριλαμβάνονταν στις αρχικές προτάσεις του Lehman, ενώ οι υπόλοιποι νόμοι προστέθηκαν μετά από περαιτέρω εργασία. Ο έκτος και ο έβδομος νόμος είναι παρόμοιοι, και ουσιαστικά λένε ότι οι χρήστες ενός συστήματος θα είναι όλο και πιο δυσαρεστημένοι με αυτό, εκτός και αν το σύστημα συντηρείται ή αποκτά νέα λειτουργικότητα. Ο τελευταίος νόμος αντικατοπτρίζει το πρόσφατο έργο επάνω στις διαδικασίες ανατροφοδότησης, αν και ακόμα δεν είναι σαφές πώς αυτό θα μπορούσε να εφαρμοστεί πρακτικά στην ανάπτυξη του λογισμικού.

Σε γενικές γραμμές, οι παρατηρήσεις του Lehman φαίνονται λογικές, και πρέπει να λαμβάνονται υπόψη κατά τον προγραμματισμό της διαδικασίας συντήρησης. Ωστόσο, επιχειρηματικοί λόγοι ίσως απαιτήσουν την παράκαμψή τους σε κάποιες περιπτώσεις. Για παράδειγμα, εμπορικοί λόγοι ίσως να απαιτούν ορισμένες μεγάλες αλλαγές σε κάποια έκδοση ενός συστήματος. Οι πιθανές συνέπειες των αλλαγών αυτών είναι ότι μπορεί να απαιτηθούν μία ή και περισσότερες εκδόσεις αφιερωμένες στην επιδιόρθωση των σφαλμάτων.

Σε κάποιες περιπτώσεις, οι ριζικές διαφορές που είναι προφανείς μεταξύ των εκδόσεων προϊόντων λογισμικού ίσως φαίνεται ότι παραβιάζουν τους νόμους του Lehman. Για παράδειγμα, το Microsoft Word έχει μετατραπεί από έναν απλό επεξεργαστή κειμένου ο οποίος λειτουργούσε με 256K μνήμης σε ένα γιγαντιαίο σύστημα με τεράστια πληθώρα λειτουργιών. Τώρα για να λειτουργήσει χρειάζεται πολλά megabyte μνήμης και γρήγορο επεξεργαστή. Η εξέλιξή του φαίνεται να έρχεται σε αντίθεση με τον τέταρτο και τον πέμπτο νόμο του Lehman. Ωστόσο, υποψιάζομαι ότι το συγκεκριμένο πρόγραμμα στην πραγματικότητα δεν αποτελεί μια σειρά βελτιώσεων ενός κοινού πυρήνα. Το όνομα διατηρήθηκε για εμπορικούς λόγους, αλλά το ίδιο το πρόγραμμα έχει ξαναγραφτεί και αναδομηθεί πολλές φορές μετά την αρχική του κυκλοφορία στην αγορά.

7.2 Συντήρηση λογισμικού

Συντήρηση λογισμικού ονομάζεται η γενική διαδικασία τροποποίησης ενός συστήματος μετά την παράδοσή του. Ο όρος συνήθως χρησιμοποιείται για λογισμικό κατά παραγγελία, όπου πριν και μετά την παράδοση εμπλέκονται διαφορετικές ομάδες ανάπτυξης. Οι αλλαγές που πραγματοποιούνται στο λογισμικό μπορεί να αφορούν απλώς τη διόρθωση λαθών στον κώδικα, να είναι πιο εκτεταμένες αλλαγές με σκοπό τη διόρθωση σχεδιαστικών σφαλμάτων, ή να αποτελούν σημαντικές βελτιώσεις οι οποίες διορθώνουν λάθη στις προδιαγραφές ή υποστηρίζουν νέες απαιτήσεις. Οι αλλαγές υλοποιούνται με την τροποποίηση υφιστάμενων συστατικών στοιχείων του συστήματος και, όταν είναι απαραίτητο, με την προσθήκη νέων συστατικών στοιχείων.

Υπάρχουν τρεις διαφορετικοί τύποι συντήρησης λογισμικού:

1. Συντήρηση με σκοπό την επιδιόρθωση σφαλμάτων λογισμικού. Η επιδιόρθωση σφαλμάτων του κώδικα έχει σχετικά μικρό κόστος, τα σχεδιαστικά λάθη έχουν μεγαλύτερο κόστος, καθώς μπορεί να απαιτήσουν την εκ νέου συγγραφή πολλών συστατικών στοιχείων ενός προγράμματος. Η επιδιόρθωση σφαλμάτων στις απαιτήσεις είναι η πλέον δαπανηρή, καθώς τέτοιου είδους λάθη είναι πιθανό να απαιτήσουν εκτεταμένη ανακατασκευή του συστήματος.
2. Συντήρηση με σκοπό την προσαρμογή του λογισμικού σε διαφορετικό λειτουργικό περιβάλλον. Αυτός ο τύπος συντήρησης είναι επιβεβλημένος όταν μεταβάλλεται κάποια πλευρά του περιβάλλοντος του συστήματος όπως το υλικό, η πλατφόρμα του λειτουργικού συστήματος, ή άλλο λογισμικό υποστήριξης. Το σύστημα πρέπει να τροποποιηθεί ώστε να προσαρμοστεί και να ανταπεξέλθει σε αυτές τις περιβαλλοντικές αλλαγές.
3. Συντήρηση με σκοπό την προσθήκη ή την τροποποίηση λειτουργικότητας του συστήματος. Ο συγκεκριμένος τύπος συντήρησης είναι απαραίτητος όταν μεταβάλλονται οι απαιτήσεις του συστήματος εξαιτίας οργανωτικών ή επιχειρηματικών αλλαγών. Η κλίμακα των απαραίτητων αλλαγών στο λογισμικό συχνά είναι πολύ μεγαλύτερη από άλλους τύπους συντήρησης.

Στην πράξη, δεν υπάρχει κάποια σαφής διάκριση μεταξύ των ανωτέρω τύπων συντήρησης. Όταν ένα σύστημα προσαρμόζεται σε νέο περιβάλλον, μπορεί να δεχτεί επιπλέον λειτουργικότητα ώστε να είναι σε θέση να εκμεταλλευτεί νέα περιβαλλοντικά χαρακτηριστικά. Συχνά παρουσιάζονται σφάλματα λογισμικού τα οποία οφείλονται στο ότι το σύστημα χρησιμοποιείται με απρόσμενους τρόπους από τους χρήστες. Ο καλύτερος τρόπος επιδιόρθωσης αυτών των σφαλμάτων είναι η μεταβολή του συστήματος ώστε να υποστηρίζει τον τρόπο εργασίας των εν λόγω χρηστών.

Οι ανωτέρω τύποι συντήρησης χαίρουν γενικής αναγνώρισης, ωστόσο ορισμένοι αναφέρονται σε αυτούς με διαφορετικά ονόματα. Για αναφορά στη συντήρηση με σκοπό την επιδιόρθωση σφαλμάτων, οι πάντες χρησιμοποιούν τον όρο διορθωτική συντήρηση (corrective maintenance). Ωστόσο, ο όρος προσαρμοστική συντήρηση (Adaptive maintenance) μερικές φορές σημαίνει την προσαρμογή σε νέο περιβάλλον και κάποιες άλλες φορές την προσαρμογή του λογισμικού σε νέες απαιτήσεις. Ο όρος τελειοποιητική συντήρηση (perfective maintenance) μπορεί να σημαίνει την τελειοποίηση του λογισμικού μέσω της υλοποίησης των νέων απαιτήσεων, σε άλλες περιπτώσεις, αναφέρεται στη βελτίωση της δομής και της απόδοσης ενός συστήματος χωρίς αλλαγές στη λειτουργικότητά του.

Μελέτες από τους Lientz και Swanson[21] (Lientz και Swanson, 1980) και από τους Nosek και Palvia[25] (Nosek και Palvia, 1990) υποδεικνύουν ότι το 65% περίπου της συντήρησης αφορά την υλοποίηση νέων απαιτήσεων, το 18% σχετίζεται με αλλαγές του συστήματος για την προσαρμογή του σε νέα λειτουργικά περιβάλλοντα, και το 17% αφορά τη διόρθωση σφαλμάτων (Εικόνα 7.3). Η κατανομή αυτή εξακολουθεί να είναι χονδρικά σωστή και για συστήματα κατά παραγγελία. Το σημαντικό εδώ δεν είναι τα συγκεκριμένα ποσοστά, αλλά το γεγονός ότι η επιδιόρθωση των σφαλμάτων ενός συστήματος δεν αποτελεί την πλέον δαπανηρή δραστηριότητα συντήρησης. Το μεγαλύτερο μέρος της συντήρησης αφιερώνεται στην εξέλιξη του συστήματος με σκοπό την υποστήριξη νέων περιβαλλόντων και νέων ή τροποποιημένων απαιτήσεων.

Το κόστος συντήρησης, ως ποσοστό του κόστους ανάπτυξης, ποικίλλει ανάλογα με τον τομέα εφαρμογής. Ο (Guimaraes[12], 1983) αναφέρει ότι το κόστος της συντήρησης επιχειρηματικών συστημάτων εφαρμογών γενικά είναι συγκρίσιμο με το κόστος ανάπτυξης. Για ένα ενσωματωμένο σύστημα πραγματικού χρόνου, το κόστος συντήρησης μπορεί να είναι μέχρι και τέσσερις φορές μεγαλύτερο από το κόστος ανάπτυξης. Οι υψηλές απαιτήσεις αξιοπιστίας και απόδοσης τέτοιου είδους συστημάτων συχνά απαιτούν τη στενή αλληλοσύνδεση των υπομονάδων, με αποτέλεσμα κάθε αλλαγή σε αυτές τις υπομονάδες να είναι δύσκολη.

Συνήθως, η επένδυση στο σχεδιασμό και την υλοποίηση ενός συστήματος με στόχο τη μείωση του κόστους συντήρησης αποτελεί μια οικονομικά συμφέρουσα πρακτική. Η προσθήκη νέας λειτουργικότητας μετά την παράδοση ενέχει μεγάλο κόστος, καθώς πρέπει να αφιερωθεί χρόνος για την κατανόηση του συστήματος και την ανάλυση των επιπτώσεων που θα έχουν οι προτεινόμενες αλλαγές. Έτσι, η εργασία που πραγματοποιείται κατά την

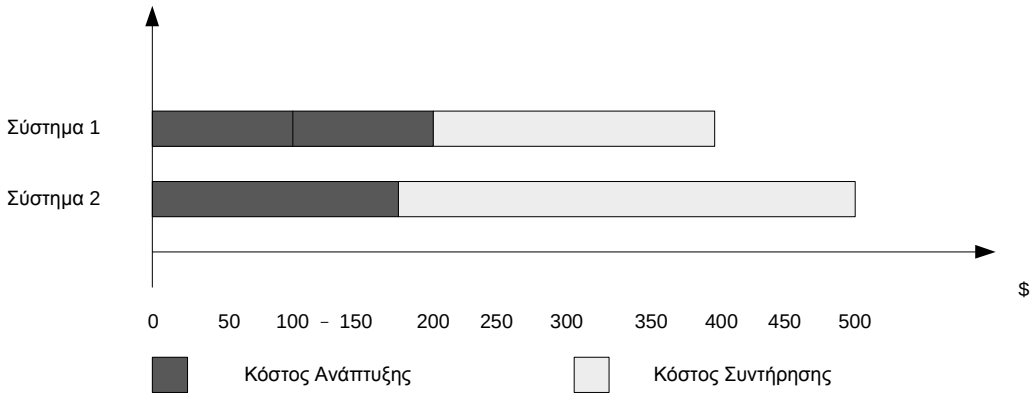
ανάπτυξη με στόχο την παράδοση εύκολα κατανοήσιμου και τροποποιήσιμου λογισμικού μπορεί να μειώσει το κόστος συντήρησης. Καλές τεχνικές τεχνολογίας λογισμικού, όπως επακριβείς προδιαγραφές, χρήση αντικειμενοστρεφούς ανάπτυξης, και διαχείριση διευθετήσεων, συνεισφέρουν στη μείωση του κόστους συντήρησης.

Η Εικόνα 7.4 παρουσιάζει τη μείωση που μπορεί να παρατηρηθεί στο συνολικό κόστος ενός συστήματος καθ' όλη τη διάρκεια ζωής του, καθώς αυξάνεται η προσπάθεια κατά την ανάπτυξη του συστήματος με στόχο τη συντηρησιμότητα. Λόγω της μείωσης που μπορεί να επέλθει στο κόστος της κατανόησης, της ανάλυσης, και των δοκιμών, η ανάπτυξη ενός συστήματος με στόχο τη συντηρησιμότητα μπορεί να έχει δραστική αντίδραση. Η επένδυση ενός επιπλέον κόστους ανάπτυξης 25.000 δολαρίων ώστε να γίνει το Σύστημα 1 πιο συντηρήσιμο, οδηγεί σε εξοικονόμηση 100.000 δολαρίων στο κόστος συντήρησης καθ' όλη τη διάρκεια ζωής του συστήματος. Αυτό σημαίνει ότι μια ποσοστιαία αύξηση του κόστους ανάπτυξης οδηγεί σε μια συγκρίσιμη ποσοστιαία μείωση του συνολικού κόστους του συστήματος.

Ένας σημαντικός λόγος για το μεγάλο ύψος του κόστους συντήρησης είναι ότι η προσθήκη λειτουργικότητας σε ένα σύστημα το οποίο βρίσκεται σε λειτουργία είναι πιο δαπανηρή από την υλοποίηση της ίδιας λειτουργικότητας κατά τη διάρκεια της ανάπτυξης. Οι βασικοί παράγοντες που διαφοροποιούν την ανάπτυξη από τη συντήρηση, και οι οποίοι οδηγούν σε μεγαλύτερο κόστος συντήρησης, είναι οι εξής:

1. Σταθερότητα της ομάδας. Μετά την παράδοση του συστήματος, συνήθως η ομάδα ανάπτυξης διαλύεται και οι προγραμματιστές εργάζονται σε νέα έργα. Η νέα ομάδα ή τα άτομα που αναλαμβάνουν τη συντήρηση του συστήματος δεν κατανοούν το σύστημα ή το σκεπτικό με το οποίο ελήφθησαν οι σχεδιαστικές αποφάσεις. Μεγάλο μέρος της προσπάθειας κατά τη διαδικασία της συντήρησης αφιερώνεται στην κατανόηση του υφιστάμενου συστήματος πριν την υλοποίηση αλλαγών.
2. Θέματα συμβολαίων. Το συμβόλαιο για τη συντήρηση ενός συστήματος συνήθως είναι ξεχωριστό από το συμβόλαιο ανάπτυξης. Το συμβόλαιο συντήρησης μπορεί να δοθεί σε κάποια εταιρεία διαφορετική από την εταιρεία ανάπτυξης του αρχικού συστήματος. Αυτός ο παράγοντας, μαζί με την έλλειψη σταθερότητας της ομάδας, σημαίνει ότι η ομάδα ανάπτυξης δεν έχει κάποιο κίνητρο να γράψει λογισμικό με τρόπο ο οποίος να διευκολύνει τις μεταβολές. Αν μια ομάδα μπορεί να ακολουθήσει συντομότερο δρόμο, ώστε να εξοικονομήσει κόπο κατά την ανάπτυξη, θα προτιμήσει να το κάνει ακόμη και αν αυτό σημαίνει αυξημένο κόστος συντήρησης.

3. Ικανότητες του προσωπικού. Το προσωπικό συντήρησης συχνά δεν έχει την απαραίτητη πείρα και εξοικείωση με τον τομέα της εφαρμογής. Η γνώμη των μηχανικών λογισμικών για τη συντήρηση δεν είναι καλή. Τη θεωρούν ως μια διαδικασία η οποία δεν απαιτεί τόσες ικανότητες όσο η ανάπτυξη συστημάτων, και έτσι συχνά η συντήρηση παραχωρείται σε νεότερο προσωπικό. Επιπλέον, παλιότερα συστήματα μπορεί να έχουν γραφτεί σε ξεπερασμένες γλώσσες προγραμματισμού. Το προσωπικό συντήρησης ίσως να μην έχει πείρα επάνω στην ανάπτυξη με αυτές τις γλώσσες, και χρειάζεται ειδική εκπαίδευση για να συντηρεί το σύστημα.
4. Ηλικία και δομή του προγράμματος. Με το πέρασμα του χρόνου, η δομή των προγραμμάτων τείνει να υποβαθμιστεί από τις μεταβολές, οπότε η κατανόηση και η τροποποίησή τους γίνεται όλο και δυσκολότερη. Κάποια συστήματα έχουν αναπτυχθεί χωρίς τη βοήθεια των σύγχρονων τεχνικών της τεχνολογίας λογισμικού. Ίσως η δομή τους να μην ήταν εξαρχής σωστή ή να είχαν βελτιστοποιηθεί με στόχο την αποδοτικότητα και όχι την κατανοησιμότητα. Η τεκμηρίωση μπορεί να έχει χαθεί ή να είναι συνεπής. Ένα παλιό σύστημα είναι πιθανό να μη χρησιμοποιεί διαχείριση διευθετήσεων, με αποτέλεσμα να σπαταλάται χρόνος για την εύρεση των σωστών εκδόσεων των συστατικών στοιχείων που πρέπει να μεταβληθούν.



ΕΙΚΟΝΑ 7.4 *Κόστος ανάπτυξης και συντήρησης*

Τα τρία πρώτα από αυτά τα προβλήματα προέρχονται από το γεγονός ότι πολλές εταιρείες εξακολουθούν να θεωρούν την ανάπτυξη και τη συντήρηση ξεχωριστές δραστηριότητες. Η συντήρηση αντιμετωπίζεται ως μια δραστηριότητα δεύτερης κατηγορίας, με αποτέλεσμα να μην υπάρχουν κίνητρα για την επένδυση χρημάτων κατά την ανάπτυξη με στόχο τη μείωση του κόστους μεταβολής του συστήματος. Η μόνη μακροπρόθεσμη λύση σε αυτό το πρόβλημα είναι να δεχτούμε ότι τα συστήματα σπανίως έχουν προκαθορισμένη διάρκεια ζωής και συνεχίζουν να χρησιμοποιούνται, σε κάποια μορφή, για απροσδιόριστο

χρονικό διάστημα. Πρέπει να θεωρούμε ότι ένα σύστημα εξελίσσεται συνεχώς καθ' όλη τη διάρκεια της ζωής του, μέσω μιας διαρκούς διαδικασίας ανάπτυξης.

Το τέταρτο ζήτημα, το πρόβλημα της υποβάθμισης της δομής των συστημάτων, είναι κατά κάποιον τρόπο το ευκολότερο να αντιμετωπίζει. Θα μπορούσαν να εφαρμοστούν τεχνικές ανακατασκευής λογισμικού για τη βελτίωση της δομής και της κατανοησιμότητας του συστήματος. Ένα σύστημα θα μπορούσε να προσαρμοστεί σε νέο υλικό μέσω αρχιτεκτονικών μετασχηματισμών. Τέλος, η υποστήριξη προληπτικών δραστηριοτήτων συντήρησης (ουσιαστικά η βαθμιαία ανακατασκευή) είναι σε θέση να επιφέρει βελτίωση ενός συστήματος και να διευκολύνει τη μεταβολή του.

7.3 Πρόβλεψη συντήρησης

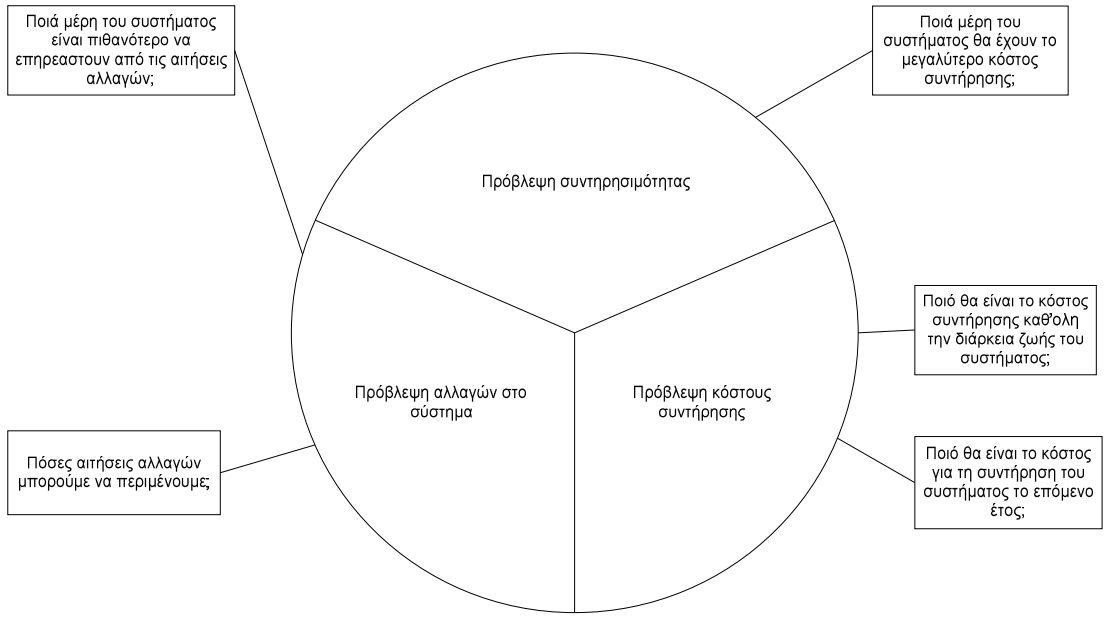
Οι διευθυντές μισούν τις εκπλήξεις, ιδιαίτερα αν αυτές οδηγούν σε υπερβολικά υψηλό κόστος. Επομένως, πρέπει να προσπαθείτε να προβλέπετε ποιες μεταβολές είναι πιθανόν να χρειαστούν σε ένα σύστημα, καθώς και ποια μέρη του συστήματος μπορεί να παρουσιάσουν τις μεγαλύτερες δυσκολίες κατά τη συντήρηση. Είναι επίσης σημαντικό να επιχειρήσετε μια εκτίμηση για το συνολικό κόστος συντήρησης ενός συστήματος ενός συστήματος, σε μια δεδομένη χρονική περίοδο. Η Εικόνα 7.5 παρουσιάζει αυτές τις προβλέψεις και τα αντίστοιχα ερωτήματα.

Οι ανωτέρω προβλέψεις προφανώς σχετίζονται στενά μεταξύ τους:

1. Το αν πρέπει να γίνει αποδεκτή μια μεταβολή συστήματος εξαρτάται, σε κάποιο βαθμό, από τη συντηρησιμότητα των συστατικών στοιχείων που επηρεάζονται από τη δεδομένη μεταβολή.
2. Η υλοποίηση αλλαγών σε ένα σύστημα τείνει να υποβαθμίζει τη δομή του, με αποτέλεσμα να μειώνεται η συντηρησιμότητα του συστήματος.
3. Το κόστος συντήρησης εξαρτάται από το πλήθος των αλλαγών, και το κόστος υλοποίησης των αλλαγών εξαρτάται από τη συντηρησιμότητα των συστατικών στοιχείων του συστήματος.

Η πρόβλεψη του πλήθους των αιτήσεων για αλλαγές σε ένα σύστημα απαιτεί μια αντίληψη της σχέσης ανάμεσα στο σύστημα και το εξωτερικό του περιβάλλον. Ορισμένα συστήματα διαθέτουν μια εξαιρετικά περίπλοκη σχέση με το εξωτερικό τους περιβάλλον, και αλλαγές σε αυτό το περιβάλλον αναπόφευκτα οδηγούν σε αλλαγές του συστήματος. Για να εκτιμήσετε τις σχέσεις ανάμεσα σε ένα σύστημα και το περιβάλλον του, πρέπει να αξιολογήσετε τους εξής παράγοντες:

1. Το πλήθος και την πολυπλοκότητα των διασυνδέσεων του συστήματος. Όσο μεγαλύτερος είναι ο αριθμός των διασυνδέσεων και όσο πιο περίπλοκες είναι αυτές οι διασυνδέσεις, τόσο μεγαλύτερη είναι η πιθανότητα απαιτήσεων για αλλαγές.
2. Το πλήθος των εγγενών ευμετάβλητων απαιτήσεων συστήματος. Οι απαιτήσεις που αντανακλούν πολιτικές και διαδικασίες της εταιρείας είναι συνήθως πιο ευμετάβλητες από απαιτήσεις οι οποίες βασίζονται σε σταθερά χαρακτηριστικά του τομέα.
3. Τις επιχειρηματικές διαδικασίες στις οποίες χρησιμοποιείται το σύστημα. Καθώς οι επιχειρηματικές διαδικασίες εξελίσσονται, παράγουν αιτήσεις για μεταβολές του συστήματος. Όσο πιο πολλές είναι οι επιχειρηματικές διαδικασίες που χρησιμοποιούν ένα σύστημα, τόσο περισσότερες είναι οι απαιτήσεις για αλλαγές στο σύστημα αυτό.



ΕΙΚΟΝΑ 7.5 *Προβλέψεις συντήρησης*

Για να είναι σε θέση κάποιος να προβλέψει τη συντηρησιμότητα ενός συστήματος, πρέπει να κατανοεί το πλήθος και το είδος των σχέσεων μεταξύ των συστατικών στοιχείων του, καθώς και το βαθμό της εγγενούς πολυπλοκότητας αυτών των συστατικών στοιχείων. Έχουν πραγματοποιηθεί διάφορες μελέτες για τους διάφορους τύπους πολυπλοκότητας σε ένα σύστημα (McCabe, 1976, Halstead, 1977), όπως και για τις σχέσεις μεταξύ πολυπλοκότητας και συντηρησιμότητας (Kafura και Reddy[16], 1987-Banker[1], κ.ά. 1993). Δεν προκαλεί έκπληξη ότι η κοινή διαπίστωση αυτών των μελετών είναι ότι όσο μεγαλύτερη είναι η πολυπλοκότητα ενός συστήματος ή συστατικού στοιχείου, τόσο μεγαλύτερο είναι και το κόστος συντήρησής του.

Έχει διαπιστωθεί ότι οι μετρήσεις πολυπλοκότητας συνιστούν ένα ιδιαίτερα χρήσιμο μέσο στον προσδιορισμό συγκεκριμένων συστατικών στοιχείων ενός προγράμματος των οποίων το κόστος συντήρησης θα μπορούσε να είναι ιδιαίτερα υψηλό. Οι Kafura και Reddy[16] (Kafura και Reddy, 1987) εξέτασαν μια σειρά από συστατικά στοιχεία συστημάτων και διαπίστωσαν ότι οι δραστηριότητες συντήρησης εστιάζονταν συνήθως σε ένα μικρό πλήθος πολύπλοκων συστατικών στοιχείων. Έτσι πρότειναν ότι η αντικατάσταση ιδιαίτερα πολύπλοκων συστατικών στοιχείων με απλούστερες εναλλακτικές λύσεις θα μπορούσε να μειώσει το κόστος συντήρησης.

Αφού τεθεί σε λειτουργία ένα σύστημα, η συντηρησιμότητά του μπορεί να προβλεφθεί με τη χρήση διαδικασιακών δεδομένων. Παραδείγματα διαδικασιακών μετρικών (process metrics) που μπορούν να χρησιμοποιηθούν για την αξιολόγηση της συντηρησιμότητας είναι τα εξής:

1. Το πλήθος των αιτήσεων για διορθωτική συντήρηση. Αύξηση του πλήθους των αναφορών για βλάβες μπορεί να σημαίνει ότι η κατά τη διαδικασία της συντήρησης εισάγονται στο πρόγραμμα περισσότερα σφάλματα από αυτά που διορθώνονται. Κάτι τέτοιο ίσως να αποτελεί ένδειξη υποβάθμισης της συντηρησιμότητας.
2. Ο μέσος χρόνος που απαιτείται για ανάλυση των επιπτώσεων. Αυτή η μετρική αντικατοπτρίζει το πλήθος των συστατικών στοιχείων ενός προγράμματος τα οποία επηρεάζονται από την αίτηση της αλλαγής. Αν ο χρόνος αυξάνεται, αυτό σημαίνει ότι επηρεάζονται όλο και περισσότερα συστατικά στοιχεία και υποβαθμίζεται η συντηρησιμότητα.
3. Ο μέσος χρόνος που απαιτείται για την υλοποίηση μιας αίτησης αλλαγής. Δεν είναι ίδιος με το χρόνο για την ανάλυση των επιπτώσεων, αν και μπορεί να σχετίζεται με αυτόν. Πρόκειται για το χρόνο που απαιτείται για την ίδια την τροποποίηση του συστήματος και της τεκμηρίωσής του, όταν έχουν πλέον προσδιοριστεί τα συστατικά στοιχεία που επηρεάζονται. Μια αύξηση του απαιτούμενου χρόνου για την υλοποίηση κάποιας αλλαγής μπορεί να σημαίνει υποβάθμιση της συντηρησιμότητας.
4. Το πλήθος των εκκρεμών αιτήσεων για αλλαγές. Αύξηση αυτού του αριθμού με το πέρασμα του χρόνου μπορεί να σημαίνει υποβάθμιση της συντηρησιμότητας.

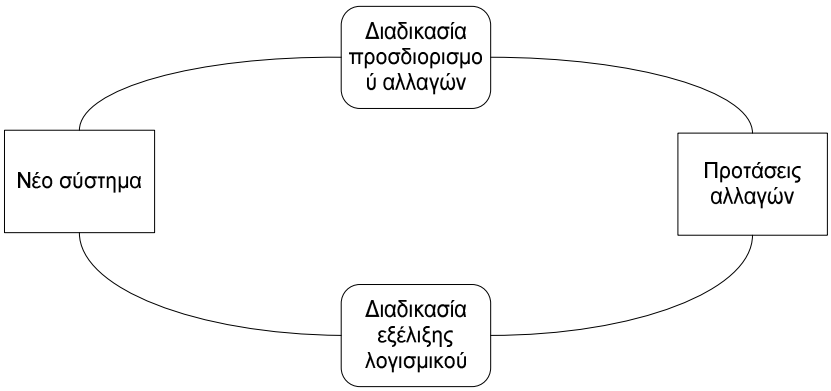
Οι πληροφορίες από τις προβλέψεις για τις αιτήσεις αλλαγών και τη συντηρησιμότητα του συστήματος χρησιμοποιούνται για την εκτίμηση του κόστους συντήρησης. Οι περισσότεροι διευθυντές εκτιμούν το κόστος συνδυάζοντας αυτά τα στοιχεία με την διαίσθηση και την πείρα τους. Το μοντέλο εκτίμησης κόστους COCOMO 2(Boehm[5], κ.ά. 2000), προτείνει ότι

μια εκτίμηση των δραστηριοτήτων συντήρησης λογισμικού θα μπορούσε να βασιστεί στην προσπάθεια κατανόησης του υφισταμένου κώδικα και την προσπάθεια για ανάπτυξη νέου κώδικα.

7.4 Διαδικασίες εξέλιξης

Οι διαδικασίες εξέλιξης λογισμικού (software evolution) ποικίλλουν σε μεγάλο βαθμό ανάλογα με τον τύπο του λογισμικού προς συντήρηση, τις διαδικασίες ανάπτυξης που χρησιμοποιούνται σε έναν οργανισμό, και τα άτομα τα οποία εμπλέκονται στη διαδικασία. Σε ορισμένες εταιρείες, η εξέλιξη μπορεί να αποτελεί άτυπη διαδικασία όπου οι αιτήσεις αλλαγών προέρχονται κυρίως από συζητήσεις μεταξύ χρηστών και δημιουργών του συστήματος. Σε άλλες εταιρείες η διαδικασία είναι τυποποιημένη και κάθε στάδιό της συνοδεύεται από δομημένη τεκμηρίωση.

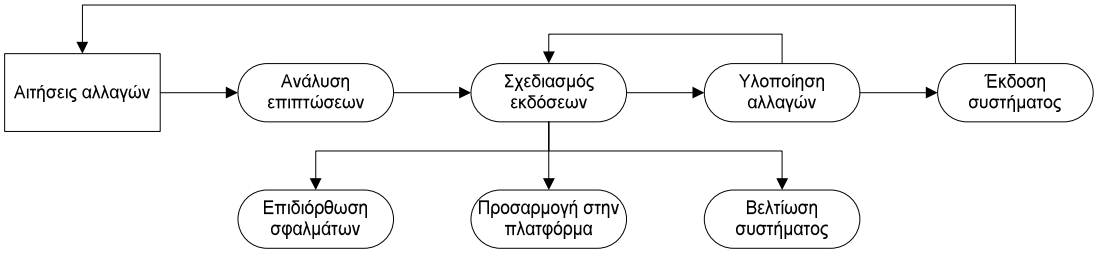
Σε όλους τους οργανισμούς, κινητήρια δύναμη για την εξέλιξη των συστημάτων είναι οι προτάσεις για αλλαγές. Αυτές οι προτάσεις μπορεί να αφορούν υπάρχουσες απαιτήσεις οι οποίες δεν έχουν υλοποιηθεί στο τελικό σύστημα, αιτήσεις των ενδιαφερομένων για νέες απαιτήσεις και επιδιορθώσεις σφαλμάτων, αλλά και νέες ιδέες και προτάσεις για βελτίωση του λογισμικού από την ομάδα ανάπτυξης του συστήματος. Όπως φαίνεται στην Εικόνα 7.6, οι διαδικασίες προσδιορισμού των αλλαγών και εξέλιξης ενός συστήματος είναι κυκλικές, και συνεχίζονται καθ' όλη τη διάρκεια ζωής του συστήματος αυτού.



ΕΙΚΟΝΑ 7.6 *Διαδικασίες προσδιορισμού μεταβολών και εξέλιξης*

Η διαδικασία εξέλιξης περιλαμβάνει τις θεμελιώδεις δραστηριότητες της ανάλυσης των μεταβολών, του σχεδιασμού των εκδόσεων, της υλοποίησης του συστήματος. Και της παράδοσης του συστήματος στον πελάτη. Καταρχήν αξιολογείται το κόστος και οι επιπτώσεις των αλλαγών, ώστε να προσδιοριστούν τα μέρη του συστήματος που

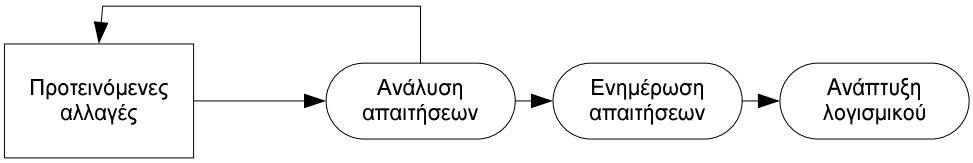
επηρεάζονται από αυτές και το κόστος που ενέχει η υλοποίησή τους. Αν οι προτεινόμενες αλλαγές γίνουν αποδεκτές, σχεδιάζεται μια νέα έκδοση του συστήματος. Κατά το σχεδιασμό εξετάζονται όλες οι προτεινόμενες μεταβολές (επιδιόρθωση σφαλμάτων, προσαρμογή, και νέα λειτουργικότητα). Στη συνέχεια αποφασίζεται το ποιες αλλαγές θα υλοποιηθούν στην επόμενη έκδοση του συστήματος. Οι αλλαγές υλοποιούνται και επικυρώνονται, και κυκλοφορεί μια νέα έκδοση του συστήματος. Κατόπιν η διαδικασία επαναλαμβάνεται με νέο σύνολο μεταβολών για την επόμενη έκδοση. Η Εικόνα 7.7, δανεισμένη από τον Arthur (Arthur, 1988), παρουσιάζει μια επισκόπηση της ανωτέρω διαδικασίας.



ΕΙΚΟΝΑ 7.7 *Η διαδικασία εξέλιξης ενός συστήματος*

Η διαδικασία της υλοποίησης μεταβολών είναι ουσιαστικά μια συνεχής επανάληψη της διαδικασίας ανάπτυξης, κατά την οποία σχεδιάζονται, υλοποιούνται, και δοκιμάζονται αναθεωρήσεις του συστήματος. Ωστόσο, μια κρίσιμη διαφορά αποτελεί το γεγονός ότι το αρχικό στάδιο της υλοποίησης των αλλαγών είναι η κατανόηση του προγράμματος. Κατά τη διάρκεια αυτής της φάσης, πρέπει να κατανοήσετε τη δομή του προγράμματος και τον τρόπο με τον οποίο παρέχει τη λειτουργικότητά του. Όταν υλοποιείται μια αλλαγή, χρησιμοποιούμε αυτή την γνώση για να διασφαλίσουμε ότι η αλλαγή αυτή δεν έχει ανεπιθύμητες παρενέργειες στο υφιστάμενο σύστημα.

Στην ιδανική περίπτωση, κατά το στάδιο υλοποίησης των αλλαγών αυτής της διαδικασίας θα πρέπει να τροποποιούνται οι προδιαγραφές, ο σχεδιασμός, και η υλοποίηση του συστήματος, ώστε να αντικατοπτρίζουν τις μεταβολές (Εικόνα 7.8). Προτείνονται, αναλύονται, και επικυρώνονται νέες απαιτήσεις, οι οποίες αντανακλούν τις μεταβολές που πρόκειται να γίνουν στο σύστημα. Κατόπιν σχεδιάζονται και υλοποιούνται τα απαραίτητα συστατικά στοιχεία, και το σύστημα δοκιμάζετε εκ νέου. Αν χρειάζεται, ένα μέρος της διαδικασίας ανάλυσης των μεταβολών αφιερώνεται στην κατασκευή πρωτοτύπων για τις προτεινόμενες αλλαγές.



ΕΙΚΟΝΑ 7.8 *Υλοποίηση αλλαγών*

Καθώς το λογισμικό τροποποιείται, αναπτύσσονται διαδοχικές τελικές εκδόσεις (releases) του συστήματος. Αυτές αποτελούνται από εκδόσεις συστατικών στοιχείων, οι οποίες πρέπει να παρακολουθούνται ώστε να διασφαλίζεται ότι κάθε τελική έκδοση του συστήματος χρησιμοποιεί τις σωστές εκδόσεις συστατικών στοιχείων.

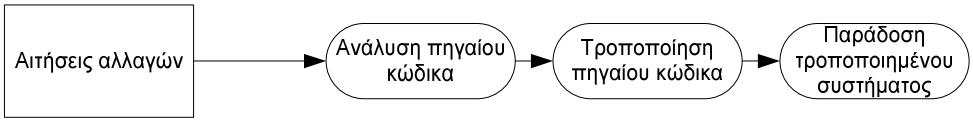
Κατά τη διαδικασία της εξέλιξης, οι απαιτήσεις αναλύονται λεπτομερώς και, συχνά, αυτό φέρνει στην επιφάνεια συνέπειες των αλλαγών οι οποίες δεν ήταν προφανείς κατά την προηγούμενη διαδικασία ανάλυσης. Έτσι, ίσως είναι απαραίτητο να τροποποιηθούν οι προτεινόμενες αλλαγές, ενώ η υλοποίησή τους μπορεί να απαιτήσει περαιτέρω συζητήσεις με τον πελάτη.

Μερικές φορές, οι αιτήσεις για αλλαγές σχετίζονται με προβλήματα του συστήματος τα οποία πρέπει να αντιμετωπιστούν επειγόντως. Αυτές οι επείγουσες αλλαγές μπορεί να προκύψουν για τρεις λόγους:

1. Έχει παρουσιαστεί κάποιο σοβαρό σφάλμα το οποίο πρέπει να διορθωθεί ώστε να είναι δυνατή η συνέχιση της ομαλής λειτουργίας του συστήματος.
2. Υπάρχουν αλλαγές στο λειτουργικό περιβάλλον του συστήματος οι οποίες έχουν απροσδόκητες επιπτώσεις που διαταράσσουν την ομαλή λειτουργία.
3. Η εταιρεία για την οποία εκτελείται το σύστημα αντιμετωπίζει απροσδόκητες αλλαγές, όπως την εμφάνιση νέων ανταγωνιστών ή την αλλαγή της νομοθεσίας.

Σε αυτές τις περιπτώσεις, η ανάγκη για γρήγορες μεταβολές μπορεί να σημαίνει ότι η εφαρμογή κάποιας τυπικής διαδικασίας ανάλυσης των αλλαγών είναι αδύνατη. Αντί να τροποποιηθούν οι απαιτήσεις και ο σχεδιασμός, το άμεσο πρόβλημα λύνεται μέσω μιας επείγουσας επιδιόρθωσης στο πρόγραμμα (Εικόνα 7.9). Η προσέγγιση αυτή ωστόσο ενέχει τον κίνδυνο οι απαιτήσεις, ο σχεδιασμός του λογισμικού, και ο κώδικας να αποκτήσουν σταδιακά ασυνέπεια. Την ώρα που σκοπεύουμε να τεκμηριώσουμε την αλλαγή στις απαιτήσεις και το σχεδιασμό, μπορεί να παρουσιαστεί η ανάγκη για κάποια επιπλέον επείγουσα επιδιόρθωση στο λογισμικό. Αυτή θα έχει προτεραιότητα σε σχέση με την

τεκμηρίωση. Στο τέλος, η αρχική αλλαγή ξεχνιέται, και η τεκμηρίωση του συστήματος δεν αποκτά ποτέ συνέπεια με τον κώδικα.



ΕΙΚΟΝΑ 7.9 *Η διαδικασία επείγουσας επιδιόρθωσης*

Ένα περαιτέρω πρόβλημα με τις επείγουσες επιδιορθώσεις ενός συστήματος είναι ότι πρέπει να ολοκληρωθούν όσο το δυνατόν ταχύτερα. Αντί λοιπόν να επιλεχθεί η καλύτερη λύση με βάση τη δομή του συστήματος, προτιμάται μια λύση η οποία μπορεί να εφαρμοστεί γρήγορα και να έχει άμεσο αποτέλεσμα. Κάτι τέτοιο επιταχύνει τη διαδικασία της «γήρανσης» του λογισμικού, δυσχεραίνοντας έτσι όλο και περισσότερο τυχόν μελλοντικές αλλαγές και αυξάνοντας το κόστος συντήρησης.

Όταν πραγματοποιούνται επείγουσες επιδιορθώσεις κώδικα, το ιδανικό είναι η αίτηση αλλαγής να παραμείνει σε εκκρεμότητα ακόμα και μετά το πέρας της επιδιόρθωσης των σφαλμάτων. Έτσι η μεταβολή θα μπορεί να υλοποιηθεί εκ νέου πιο προσεκτικά μετά από περαιτέρω ανάλυση, επαναχρησιμοποιώντας ίσως τον κώδικα της επιδιόρθωσης ως βάση. Καθώς οι προγραμματιστές θα έχουν περισσότερο χρόνο διαθέσιμο για την ανάλυση, μπορεί να εντοπίσουν κάποια εναλλακτική λύση η οποία αντιμετωπίζει καλύτερα το πρόβλημα. Στην πράξη, όμως, είναι σχεδόν αναπόφευκτο ότι τέτοιου είδους αλλαγές θα έχουν χαμηλή προτεραιότητα και έτσι, καθώς θα πραγματοποιούνται περαιτέρω μεταβολές στο σύστημα, η αναδημιουργία των επειγουσών επιδιορθώσεων δεν θα αποτελεί πλέον ρεαλιστική λύση.

7.5 Ανακατασκευή συστημάτων

Όπως αναφέρθηκε στην προηγούμενη ενότητα, η διαδικασία εξέλιξης ενός συστήματος περιλαμβάνει την κατανόηση του προγράμματος που πρέπει να τροποποιηθεί, και κατόπιν την υλοποίηση αυτών των αλλαγών. Ωστόσο, η κατανόηση και η τροποποίηση πολλών συστημάτων, ιδιαίτερα αν πρόκειται για παλαιότερα κληρονομημένα συστήματα, είναι δύσκολη υπόθεση. Τα προγράμματα μπορεί να έχουν αρχικά βελτιστοποιηθεί με στόχο την απόδοση ή την εξοικονόμηση χώρου εις βάρος της κατανοησιμότητας, ή, με την πάροδο του χρόνου, η αρχική δομή τους να έχει αλλοιωθεί από μια σειρά αλλαγών.

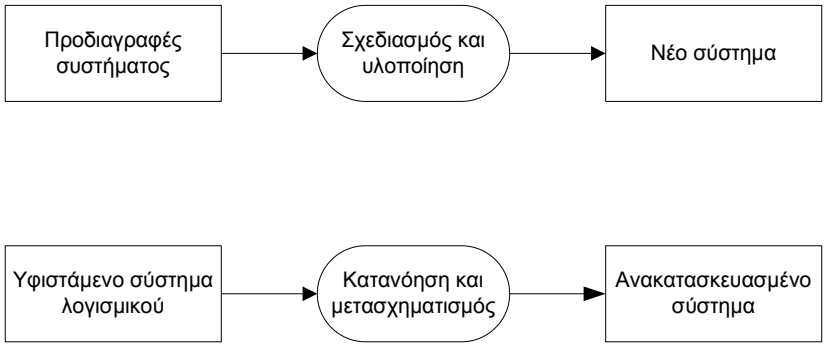
Στην προσπάθειά της να περιορίσει τα προβλήματα τροποποίησης κληρονομημένων συστημάτων, μια επιχείρηση μπορεί να αποφασίσει να ανακατασκευάσει αυτά τα συστήματα ώστε να βελτιώσει τη δομή και την κατανοησιμότητά τους. Η ανακατασκευή λογισμικού (software re-engineering) αφορά την εκ νέου υλοποίηση κληρονομημένων συστημάτων ώστε να γίνουν πιο συντηρήσιμα. Η ανακατασκευή μπορεί να περιλαμβάνει εκ νέου τεκμηρίωση του συστήματος, την οργάνωση και την αναδόμησή του, τη μετάφρασή του σε μια πιο σύγχρονη γλώσσα προγραμματισμού, καθώς και την τροποποίηση και ενημέρωση της δομής και των τιμών των δεδομένων του συστήματος. Η λειτουργικότητα του λογισμικού δεν μεταβάλλεται και, συνήθως, το ίδιο συμβαίνει με την αρχιτεκτονική του συστήματος.

Η ανακατασκευή ενός συστήματος λογισμικού διαθέτει δύο βασικά πλεονεκτήματα σε σχέση με πιο ριζικές προσεγγίσεις της εξέλιξης συστημάτων:

1. Μειωμένος κίνδυνος. Η εκ νέου ανάπτυξη λογισμικού, το οποίο είναι κρίσιμο από επιχειρηματική άποψη, ενέχει μεγάλους κίνδυνους. Οι προδιαγραφές του συστήματος μπορεί να περιέχουν λάθη, ή να παρουσιαστούν προβλήματα κατά την ανάπτυξη. Καθυστερήσεις στην κατασκευή του νέου λογισμικού μπορεί να σημαίνουν διαφυγόντα κέρδη και επιπλέον κόστος. Το 1999, για παράδειγμα, μια μεγάλη αμερικανική εταιρεία τροφίμων συνάντησε καθυστερήσεις στην κατασκευή ενός νέου συστήματος παραγγελιών, κάτι που με τη σειρά του καθυστέρησε την παράδοση αγαθών αξίας 100 εκατομμυρίων δολαρίων μέσα σε μια περίοδο αιχμής για τις πωλήσεις.
2. Μειωμένο κόστος. Το κόστος της ανακατασκευής είναι σημαντικά χαμηλότερο από το κόστος της ανάπτυξης νέου λογισμικού. Ο Ulrich[33] (Ulrich, 1990) παραθέτει το παράδειγμα ενός εμπορικού συστήματος για το οποίο το κόστος νέας υλοποίησης είχε υπολογιστεί στα 50 εκατομμύρια δολάρια. Το σύστημα ανακατασκευάστηκε επιτυχώς μόλις με 12 εκατομμύρια δολάρια. Υποψιάζομαι ότι το σχετικό κόστος της εκ νέου υλοποίησης θα είναι χαμηλότερο με τη βοήθεια της σύγχρονης τεχνολογίας λογισμικού, αλλά και πάλι σημαντικά υψηλότερο από το κόστος της ανακατασκευής.

Η βασική διαφορά μεταξύ ανακατασκευής και ανάπτυξης πρωτότυπου λογισμικού είναι το σημείο εκκίνησης στην ανάπτυξη. Αντί η διαδικασία να ξεκινάει με βάση κάποιες γραπτές προδιαγραφές, το παλιό σύστημα έχει το ρόλο των προδιαγραφών για το νέο σύστημα. Οι Chikofsky και Cross (Chikofsky και Cross, 1990[8]) ονομάζουν τη συμβατική ανάπτυξη ευθεία κατασκευή (forward engineering) ώστε να τη διακρίνουν από την ανακατασκευή

λογισμικού. Η διαφορά παρουσιάζεται στην Εικόνα 7.10. Η ευθεία κατασκευή ξεκινάει με τις προδιαγραφές συστήματος, και περιλαμβάνει το σχεδιασμό και την υλοποίηση ενός νέου συστήματος. Η ανακατασκευή ξεκινάει με ένα υφιστάμενο σύστημα, και η διαδικασία ανάπτυξης του νέου συστήματος βασίζεται στην κατανόηση και τον σχηματισμό του αρχικού.

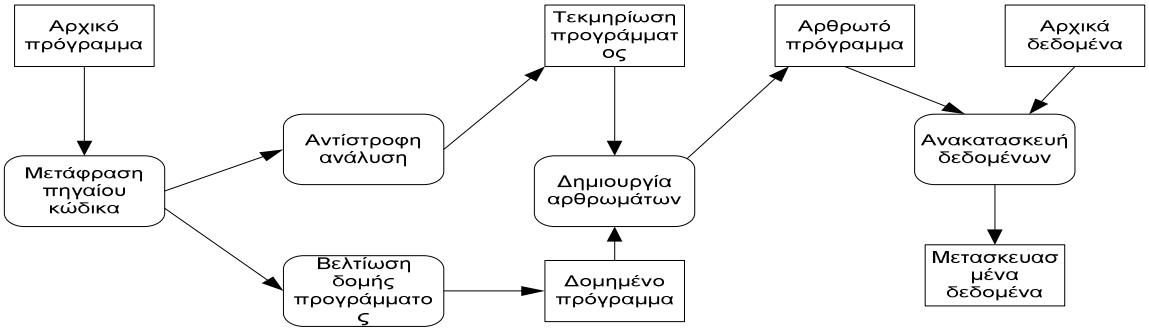


ΕΙΚΟΝΑ 7.10 *Ευθεία κατασκευή και ανακατασκευή*

Η διαδικασία της ανακατασκευής παρουσιάζεται στην Εικόνα 7.11. Η είσοδος της διαδικασίας είναι ένα κληρονομημένο πρόγραμμα, και η έξοδος είναι μια δομημένη αρθρωτή εκδοχή του ίδιου προγράμματος. Κατά την ανακατασκευή ενός προγράμματος, είναι επίσης πιθανό να μετασκευαστούν τα δεδομένα του συστήματος. Οι δραστηριότητες σε αυτή τη διαδικασία ανακατασκευής είναι οι εξής:

1. Μετάφραση πηγαίου κώδικα. Το πρόγραμμα μετατρέπεται από μια παλιά γλώσσα προγραμματισμού σε μια πιο σύγχρονη εκδοχή της ίδιας γλώσσας ή σε μια διαφορετική γλώσσα.
2. Αντίστροφη ανάλυση (reverse engineering). Το πρόγραμμα αναλύεται και εξάγονται πληροφορίες. Αυτό βοηθάει στην τεκμηρίωση της οργάνωσης και της λειτουργικότητάς του.
3. Βελτίωση δομής προγράμματος. Αναλύεται και τροποποιείται η δομή του προγράμματος, ώστε το πρόγραμμα να γίνει πιο ευανάγνωστο και εύληπτο.
4. Δημιουργία αρθρωμάτων. Ομαδοποιούνται μέρη του προγράμματος που σχετίζονται μεταξύ τους και, όπου είναι απαραίτητο, αφαιρούνται πλεονάζοντα στοιχεία. Σε ορισμένες περιπτώσεις, αυτό το στάδιο μπορεί να περιλαμβάνει αρχιτεκτονικό μετασχηματισμό, κατά τον οποίο ένα συγκεντρωτικό σύστημα, που προορίζεται για χρήση σε έναν μόνο σταθμό εργασίας, τροποποιείται ώστε να εκτελείται σε μια κατανεμημένη πλατφόρμα.

5. Μετασκευή δεδομένων. Τα δεδομένα που επεξεργάζεται το σύστημα τροποποιούνται ώστε να αντικατοπτρίζουν τις αλλαγές

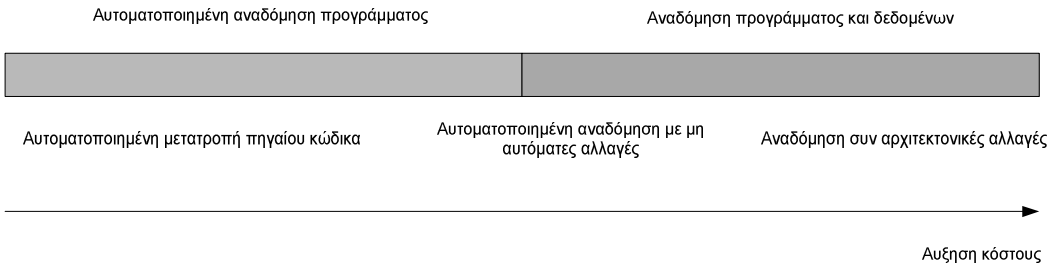


ΕΙΚΟΝΑ 7.11 *Η διαδικασία ανακατασκευής*

6. Η ανακατασκευή ενός συστήματος μπορεί να μην απαιτεί απαραιτήτως όλα τα βήματα της Εικόνας 7.11. Αν η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη του συστήματος εξακολουθεί να υποστηρίζεται από τον προμηθευτή του μεταγλωττιστή, η μετάφραση του πηγαίου κώδικα θα μπορούσε να παραληφθεί. Επίσης, αν η ανακατασκευή βασίζεται αποκλειστικά σε αυτοματοποιημένα εργαλεία, τότε ίσως να μην είναι αναγκαία η ανάκτηση της τεκμηρίωσης μέσω αντίστροφης ανάλυσης. Η μετασκευή των δεδομένων είναι απαραίτητη μόνο αν μεταβληθούν οι δομές δεδομένων του προγράμματος κατά την ανακατασκευή του προγράμματος. Ωστόσο, στην ανακατασκευή του λογισμικού περιλαμβάνεται πάντοτε κάποιος τύπος αναδόμησης του προγράμματος.

Η συνεργασία του ανακατασκευασμένου συστήματος με το νέο λογισμικό ίσως απαιτήσει την ανάπτυξη συστατικών στοιχείων προσαρμογής. Αυτά τα συστατικά στοιχεία κρύβουν τις αρχικές διασυνδέσεις οι οποίες μπορούν να χρησιμοποιηθούν από άλλα συστατικά στοιχεία. Αυτή η διαδικασία της «περιβολής» (wrapping) κληρονομημένων συστημάτων συνιστά μια σημαντική τεχνική στην ανάπτυξη επαναχρησιμοποιήσιμων συστατικών στοιχείων μεγάλης κλίμακας.

Το κόστος της ανακατασκευής εξαρτάται προφανώς από την έκταση της εργασίας που πραγματοποιείται. Η Εικόνα 7.12 παρουσιάζει ένα φάσμα δυνατών προσεγγίσεων στην ανακατασκευή. Το κόστος αυξάνεται από αριστερά προς τα δεξιά, οπότε η μετάφραση του πηγαίου κώδικα είναι η φθηνότερη επιλογή. Η ανακατασκευή ως μέρος αρχιτεκτονικών μεταβολών είναι η επιλογή με το υψηλότερο κόστος.



ΕΙΚΟΝΑ 7.12 *Προσεγγίσεις ανακατασκευής*

Εκτός από την έκταση της ανακατασκευής, οι βασικοί παράγοντες που επηρεάζουν το κόστος της διαδικασίας είναι οι εξής:

1. Η ποιότητα του λογισμικού που ανακατασκευάζεται. Όσο χαμηλότερη είναι η ποιότητα του λογισμικού και της τεκμηρίωσής του (αν υπάρχει), τόσο υψηλότερο είναι το κόστος της ανακατασκευής.
2. Τα διαθέσιμα εργαλεία για ανακατασκευή. Όταν δεν είναι δυνατή η χρήση εργαλείων CASE για την αυτοματοποίηση του μεγαλύτερου μέρους των αλλαγών στο πρόγραμμα, η ανακατασκευή ενός συστήματος λογισμικού είναι συνήθως ασύμφορη από οικονομική άποψη.
3. Η έκταση της μετατροπής που απαιτείται στα δεδομένα. Αν η ανακατασκευή απαιτεί τη μετατροπή μεγάλου όγκου δεδομένων, το κόστος της διαδικασίας αυξάνεται σημαντικά.
4. Η διαθεσιμότητα εξειδικευμένου προσωπικού. Αν το προσωπικό που είναι υπεύθυνο για τη συντήρηση του συστήματος δεν μπορεί να συμμετάσχει στη διαδικασία ανακατασκευής, το κόστος θα αυξηθεί, καθώς οι υπεύθυνοι της ανακατασκευής θα πρέπει να δαπανήσουν πολύ χρόνο για να κατανοήσουν το σύστημα.

Το κύριο μειονέκτημα της ανακατασκευής λογισμικού είναι ότι υπάρχουν κάποια πρακτικά όρια στο πόσο μπορεί να βελτιωθεί ένα σύστημα με τη χρήση αυτής της προσέγγισης. Για παράδειγμα, ένα σύστημα που έχει γραφτεί με βάση το συναρτησιακό μοντέλο δεν είναι δυνατόν να μετατραπεί σε αντικειμενοστρεφές. Δραστηριότητες όπως

μεγάλες αρχιτεκτονικές αλλαγές ή δραστική αναδιοργάνωση της διαχείρισης δεδομένων του συστήματος δεν μπορούν να πραγματοποιηθούν αυτόματα, οπότε επιφέρουν υψηλό επιπλέον κόστος. Αν και η ανακατασκευή μπορεί να βελτιώσει τη συντηρησιμότητα, το ανακατασκευασμένο σύστημα πιθανότατα δε θα είναι τόσο συντηρήσιμο όσο ένα νέο σύστημα το οποίο έχει αναπτυχθεί με τη χρήση σύγχρονων μεθόδων της τεχνολογίας λογισμικού.

ΚΕΦΑΛΑΙΟ 8:Ανάπτυξη Δικτυακής Εφαρμογής με Σύστημα Διαχείρισης Περιεχομένου

Για να αναπτυχθεί ένας δικτυακός τόπος , πρέπει πρώτα ο δημιουργός του να είναι σε θέση να γνωρίζει για ποιο λόγο δημιουργεί τον τόπο αυτό και ποια θα είναι η χρήση του. Εφόσον, περάσει από τη φάση αυτή, έπειτα περνάει στο στάδιο το οποίο χαρακτηρίζει το είδος του δικτυακού τόπου και ξεκινάει την ανάπτυξή του.

Τα είδη των δικτυακών τόπων είναι τρία: α) Δικτυακός Τόπος Περιεχομένου (πληροφοριακός), β) Εμπορικός δικτυακός τόπος (πωλήσεις) και γ) Επικοινωνιακός δικτυακός τόπος (αλληλεπίδραση με τους επισκέπτες).

Στην πτυχιακή αυτή εργασία θα ασχολειθούμε με το σχεδιασμό και την ανάπτυξη του δικτυακού τόπου του τμήματος Εφαρμογών Πληροφορικής στη Διοίκηση και Οικονομία του Παραρτήματος Αμαλιάδας του Ανώτατου Τεχνολογικού Εκπαιδευτικού Ιδρύματος Πατρών.

Ο λόγος για τον οποίο θα κατασκευάσουμε τη συγκεκριμένη ηλεκτρονική τοποθεσία είναι για την εξυπηρέτηση περαιτέρω αναγκών που δημιουργήθηκαν με το πέρασμα του χρόνου στο Ίδρυμα.Υλοποιήθηκε λοιπόν μια αρχική έκδοση του δικτυακού τόπου, πιο σύγχρονη και συνάμα πιο εύχρηστη και δυναμική.

Ο δικτυακός τόπος που αναπτύχθηκε χαρακτηρίζεται Περιεχομένου, διότι παρέχει πληροφορίες στους χρήστες του για θέματα που τους απασχολούν. Παρέχει κάθε είδους πληροφορία που θέλει να μάθει ο κάθε χρήστης που θα την επισκέπτεται έτσι ώστε να ενημερώνεται.

Κατά τη φάση της ανάπτυξης του δικτυακού τόπου καταφέραμε να καθορίσουμε τα ιδιαίτερα χαρακτηριστικά που θα έχει το τμήμα Εφαρμογές Πληροφορικής στη Διοίκηση και την Οικονομία τα οποία είναι τα ακόλουθα:

- Το περιεχόμενο σπουδών του τμήματος το οποίο καλύπτει τα γνωστικά αντικείμενα της Πληροφορικής, της Διοικητικής επιστήμης, των Οικονομικών και του marketing με εξειδίκευση στις εφαρμογές της πληροφορικής στη Διοίκηση και την Οικονομία στο πλαίσιο της οργάνωσης και λειτουργίας επιχειρήσεων και οργανισμών, τόσο του δημόσιου, όσο και του ιδιωτικού τομέα.
- Η δομή του προγράμματος σπουδών περιλαμβάνει τα γνωστικά αντικείμενα που στοχεύουν στη παροχή βασικών γνώσεων κατά τη διάρκεια των πρώτων εξαμήνων, ενώ στη συνέχεια και προς την ολοκλήρωσή του, τα γνωστικά αντικείμενα αφορούν πιο εξειδικευμένους επιστημονικά φορείς.
- Το Τμήμα ΕΠΔΟ έχει ως αποστολή την εκπαίδευση των φοιτητών ώστε να καταστούν επιστήμονες οι οποίοι με τη θεωρητική και εφαρμοσμένη κατάρτισή τους

να μεταφέρουν, να χρησιμοποιούν και να προάγουν σύγχρονες τεχνολογίες ΤΠΕ στον τομέα της διοίκησης επιχειρήσεων και οργανισμών.

- Στο Τμήμα ΕΠΔΟ εντάσσεται η δυνατότητα δημιουργίας ενός κλίματος ερευνητικών δραστηριοτήτων και επιστημονικής αναζήτησης με σκοπό την ανάπτυξη ακαδημαϊκής κουλτούρας και την προαγωγή του επιπέδου των σπουδών.
- Μετά την ολοκλήρωση των σπουδών του οι πτυχιούχοι αποκτούν τις απαραίτητες επιστημονικές και τεχνολογικές γνώσεις, οι οποίες τους επιτρέπουν να δραστηριοποιηθούν επαγγελματικά, υποστηρίζοντας με επιτυχία την αξιοποίηση των τεχνολογιών της Πληροφορικής και των Επικοινωνιών σε όλους τους τομείς της οικονομικής και επιχειρηματικής δραστηριότητας.
- Επιπλέον το Τμήμα ΕΠΔΟ παρέχει στους πτυχιούχους υψηλό επίπεδο γνώσεων στα γνωστικά επιστημονικά αντικείμενα της Πληροφορικής Επιστήμης, της Διοίκησης και Οργάνωσης Επιχειρήσεων, του Marketing, της Χρηματοδότησης και της Διαχείρισης Ανθρωπίνων Πόρων και δύναται να σχεδιάσουν, αναπτύξουν και εφαρμόσουν σύγχρονα Πληροφοριακά Συστήματα Διοίκησης.
- Ακόμα οι πτυχιούχοι του τμήματος απασχολούνται τόσο στο δημόσιο όσο και στον ιδιωτικό τομέα, είτε αυτοδύναμα, είτε σε συνεργασία με άλλους επαγγελματίες και επιστήμονες, σε όλους τους τομείς αξιοποίησης των Πληροφοριακών Συστημάτων Διοίκησης, σε θέματα που σχετίζονται με την μελέτη, την ανάπτυξη, τη διαχείριση, την εφαρμοσμένη έρευνα, την εκπαίδευση και την κατάρτιση.
- Τέλος, ένα ακόμη προνόμιο που παρέχει στους πτυχιούχους είναι η παρακολούθηση προγραμμάτων σπουδών για απόκτηση μεταπτυχιακού ή διδακτορικού διπλώματος σε πανεπιστημιακά ιδρύματα της χώρας μας ή του εξωτερικού για την εμβάθυνση των γνώσεών τους επάνω στο αντικείμενο που τους ενδιαφέρει.

8.1 ΕΠΙΛΟΓΗ ΣΥΣΤΗΜΑΤΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΠΕΡΙΕΧΟΜΕΝΟΥ (CMS)

Κατά την εκκίνηση εκπόνησης της πτυχιακής εργασίας τα επικρατέστερα open source CMSs ήταν τα "Joomla!", "WordPress" και "Drupal".

Ερευνώντας ποιο CMS αποτελούσε τη βέλτιστη επιλογή για την ανάπτυξη του web site, ανακτήθηκαν μέσω αναζήτησης στο διαδίκτυο ενδιαφέρουσες πληροφορίες, οι οποίες συνοψίζονται στον ακόλουθο πίνακα σύγκρισης:

Σύγκριση Drupal-Joomla-Wordpress			
Χαρακτηριστικό	**Drupal**	**Joomla**	**Wordpress**
			
Εγκατάσταση	Σχετικά απλή, συνήθως όμως απαιτείται κάποια τεχνική εξειδίκευση	Απλή	Απλή
Ευκολία Χρήσης	Προσανατολισμένο σε Developers και WebMasters	Απλό στη χρήση	Απλό στη χρήση
Περιβάλλον Διαχείρισης	Πληθώρα λειτουργιών	Γραφικό και διαισθητικό	Απλό
Δημοσίευση Περιεχομένου			
WYSIWYG επεξεργαστής κειμένου	Ναι	Ναι	Ναι
Δημιουργία & επεξεργασία κειμένου από Front και Back-End	Ναι	Ναι	Εξαρτάται από το εικαστικό πρότυπο
Εμφάνιση & Εικαστικά Πρότυπα			
Εικαστικά πρότυπα και συστατικά λογισμικού τρίτων κατασκευαστών	Αξιοπρεπής συλλογή (όχι πληθώρα δωρεάν)	Πλούσια συλλογή (και δωρεάν)	Πολύ πλούσια συλλογή (και δωρεάν)
Επεξεργασία προτύπων από το διαχειριστικό back-end	Όχι	Ναι	Ναι
Χρήση πολλαπλών προτύπων	Περιορισμένη	Ναι	Περιορισμένη
Αντιστοίχιση συστατικών λογισμικού σε σελίδες μέσω του διαχειριστικού back-end	Ναι	Ναι	Ναι
Διαχείριση μενού μέσω του διαχειριστικού back-end	Ναι	Ναι	Όχι
Πολυγλωσσική Υποστήριξη			
Υποστήριξη UTF-8	Ναι	Ναι	Ναι
Πακέτα γλωσσών για front και back-end	Ναι	Ναι	Ναι
Γενικές Λειτουργίες			
Ασφάλεια	Καλή	Καλή	Καλή
Υποστήριξη Blogs	Ναι	Ναι	Ναι
Ταχύτητα	Γρήγορο	Γρήγορο	Γρήγορο
Στατιστικά	Ναι	Ναι	Ναι

Από τη σχετική έρευνα προέκυψε ότι "Joomla!", ήταν το πιο πλούσιο CMS όσον αφορά τη διαθεσιμότητα δωρεάν συστατικών λογισμικού τρίτων κατασκευαστών (3rd party components, modules & plugins) και εικαστικών προτύπων (design templates), προσφέροντας τη δυνατότητα κατασκευής ακόμη και απαιτητικών δυναμικών ηλεκτρονικών τοποθεσιών, με σχετικά «βατό» τρόπο.
Το "WordPress" ήταν πολύ διαδεδομένο και δεν απαιτούσε ιδιαίτερες γνώσεις προγραμματισμού. Είχε κλίση κυρίως σε blog τύπου ιστοσελίδες, ενώ προσέφερε τη δυνατότητα κατασκευής και πιο περίπλοκων, παρέχοντας όμως πολύ λιγότερα δωρεάν συστατικά λογισμικού και εικαστικά πρότυπα και εμφανίζοντας κάποιους επιπλέον περιορισμούς (εμφανίζονται στον παραπάνω πίνακα).
Το "Drupal" τέλος, αποκλείστηκε λόγω της πιο αργής καμπύλης εκμάθησής του σε σχέση με αυτή του "Joomla!" και των απαιτήσεών του σε βαθύτερες τεχνικές γνώσεις. Ακόμη, ο αισθητά μικρότερος αριθμός δωρεάν συστατικών λογισμικού και εικαστικών προτύπων αποτέλεσε βασικό κατασταλτικό παράγοντα.

8.2 ΠΕΡΙΓΡΑΦΗ ΤΟΥ “JOOMLA!”

Το “Joomla!” είναι μία δωρεάν εφαρμογή ανοιχτού λογισμικού για τη δημιουργία δυναμικών ιστοσελίδων. Μπορεί να χρησιμοποιηθεί τόσο για προσωπικές ιστοσελίδες όσο και για επαγγελματικές.
Ανήκει στην κατηγορία των Συστημάτων Διαχείρισης Περιεχομένου (Content Management System CMS). Είναι γραμμένο σε γλώσσα PHP και τα δεδομένα αποθηκεύονται σε βάση δεδομένων MySQL.

8.2.1 Χαρακτηριστικά του “Joomla!”

Τα γενικά χαρακτηριστικά είναι τα εξής:

- Ανοιχτός κώδικας.
- Μεγάλη κοινότητα χρηστών στο www.joomla.org**[A]** και στο www.joomla.gr**[B]**.
- Μεγάλη ευελιξία στη δημοσίευση περιεχομένου.

- Διαχειριστής αρχείων για μεταφόρτωση και διαχείριση των αρχείων.
- Πολύ εύκολο στη χρήση του και από αρχάριους χρήστες Η/Υ.
- Δυνατότητες RSS.
- Κάδος ανακύκλωσης για τα αντικείμενα περιεχομένου.
- Ειδικός μηχανισμός για τις μηχανές αναζήτησης.
- Διαχείριση διαφημίσεων.
- Πολυγλωσσικότητα.
- Δεκάδες πρόσθετες εφαρμογές.
- Εύκολη εγκατάσταση εφαρμογών και πρόσθετων.
- Πολλά επίπεδα χρηστών.
- Στατιστικά.
- WYSIWYG επεξεργαστής κειμένου.
- Σύστημα ψηφοφοριών.
- Σύστημα αξιολόγησης άρθρων και πολλά άλλα.

8.2.2 Προδιαγραφές Εγκατάστασης

Οι προδιαγραφές εγκατάστασης είναι οι παρακάτω:

- Web server, Apache version 1.13.19 ή μεταγενέστερη ή Microsoft IIS.
- PHP έκδοση 4.3 ή μεταγενέστερη με υποστήριξη για MySQL.
- Η MySQL database system from version 3.23.x on or with Unicode character sets MySQL from 4.1.x on.
- Web Browser (Internet Explorer, Firefox), συνιστάται ο Firefox.

8.2.3 Η Δομή του joomla

Τα σημαντικά χαρακτηριστικά του joomla είναι τα παρακάτω:

- Δημόσιο τμήμα (Front End)

Το δημόσιο τμήμα είναι, στην ουσία, αυτό που βλέπει ο τελικός χρήστης. Μέσα στο δημόσιο τμήμα βρίσκονται τα άρθρα, τα μενού και γενικά όλα τα στοιχεία που θέλουμε να εμφανίζονται στην ιστοσελίδα.

- Περιοχή διαχείρισης (Back End)

Η περιοχή διαχείρισης είναι το ¨εργαστήριο¨ του joomla. Μέσα από την περιοχή διαχείρισης ο Διαχειριστής μπορεί να προσθέσει περιεχόμενο, να εμφανίζει ή να αποκρύπτει στοιχεία, να δημιουργεί χρήστες και γενικά να εκμεταλλεύεται όλες τις δυνατότητες του joomla.

- Μενού

Τα μενού είναι τα αντικείμενα με τα οποία ο χρήστης μπορεί να πλοηγείται στην ιστοσελίδα. Τα μενού μπορούν να είναι οριζόντια ή κατακόρυφα, δημιουργούνται δυναμικά και συνδέονται με τα αντικείμενα του joomla (ενότητες, κατηγορίες, άρθρα). Σε μια ιστοσελίδα joomla μπορούμε να έχουμε όσα μενού θέλουμε.

- Εφαρμογές (Components)

Οι εφαρμογές χρησιμοποιούνται για να μπορεί το joomla να επεκτείνεται. Άλλες είναι εμπορικές και άλλες ελεύθερης διανομής. Μερικές από αυτές είναι εφαρμογές για e-shop, gallery φωτογραφιών, e-learning.

- Ενθέματα (Modules)

Τα ενθέματα είναι τα ¨κουτιά¨ μέσα στα οποία εμφανίζονται τα περιεχόμενα, οι εφαρμογές, τα πρόσθετα και γενικά όλα τα αντικείμενα που εμφανίζονται στο Δημόσιο τμήμα (Front End).

- Πρόσθετα (plug-ins)

Τα πρόσθετα είναι κομμάτια κώδικα τα οποία εκτελούν κάποιες ειδικές λειτουργίες. Για παράδειγμα, ένα πρόσθετο είναι η μηχανή αναζήτησης που έχει το joomla για να μπορεί ο χρήστης να αναζητεί περιεχόμενο μέσα στην ιστοσελίδα.

- Πρότυπα (Templates)

Τα πρότυπα χρησιμεύουν για να διαχωριστεί το περιεχόμενο από την εμφάνιση. Στα πρότυπα ορίζονται τα χρώματα, η θέση των ενθεμάτων, και γενικά όλη η σχεδίαση της ιστοσελίδας.

8.2.4 PHP

Η PHP είναι μια γλώσσα προγραμματισμού που σχεδιάστηκε για τη δημιουργία δυναμικών ιστοσελίδων στο διαδίκτυο και είναι επισήμως γνωστή ως Hypertext preprocessor.

Είναι μια server-side scripting (εκτελείται στο διακομιστή) γλώσσα που γράφεται, συνήθως πλαισιωμένη από HTML, για μορφοποίηση των αποτελεσμάτων. Αντίθετα από μια συνηθισμένη HTML σελίδα, η σελίδα PHP δε στέλνεται άμεσα σε έναν πελάτη (client), αντ'αυτού πρώτα αναλύεται και μετά αποστέλλεται το παραγόμενο αποτέλεσμα. Τα στοιχεία HTML στον πηγαίο κώδικα μένουν ως έχουν, αλλά ο PHP κώδικας ερμηνεύεται και εκτελείται. Ο κώδικας PHP μπορεί να θέσει ερωτήματα σε βάσεις δεδομένων, να δημιουργήσει εικόνες, να διαβάσει και να γράψει αρχεία, να συνδεθεί με απομακρυσμένους υπολογιστές, Κ.Ο.

Γενικώς, οι δυνατότητες που μας δίνει είναι απεριόριστες.

Αρχικά η ονομασία της ήταν PHP/FI από το Forms Interpreter η οποία δημιουργήθηκε το 1995 από τον Rasmus Lerdorf ως μια συλλογή από Perl scripts που τα χρησιμοποιούσε στην προσωπική του σελίδα.

Δεν άργησε να τα εμπλουτίσει με λειτουργίες επεξεργασίας δεδομένων με SQL, αλλά τα σημαντικά βήματα που έφεραν και τη μεγάλη αποδοχή της PHP ήταν αρχικά η μετατροπή τους σε γλώσσα C και μετέπειτα η δωρεάν παροχή του πηγαίου κώδικα μέσω της σελίδας του, ώστε να επωφεληθούν όλοι από αυτό που είχε φτιάξει, αλλά και να τον βοηθήσουν στην περαιτέρω ανάπτυξή της.

8.2.5 MySQL

Η MySQL είναι βάση δεδομένων μέσα στην οποία μπορούμε να καταχωρούμε, να επεξεργαζόμαστε, να αναζητούμε και να ταξινομούμε δεδομένα. Παρέχει τη δυνατότητα λειτουργίας από πολλαπλούς χρήστες με ασφάλεια αφού μόνο οι κατοχυρωμένοι χρήστες μπορούν να έχουν πρόσβαση στα δεδομένα της.
Χρησιμοποιεί τη γλώσσα SQL που είναι η πιο διαδεδομένη γλώσσα για τις βάσεις δεδομένων.

8.2.6 CSS

Τα αρχικά CSS προέρχονται από το Cascading Style Sheets.
Τα CSS μας επιτρέπουν να διαχωρίσουμε το περιεχόμενο της ιστοσελίδας από το σχεδιαστικό κομμάτι.
Αυτό είναι πολύ σημαντικό αφού τα στοιχεία σχεδίασης της ιστοσελίδας θα είναι σε ένα ξεχωριστό αρχείο το οποίο θα τροφοδοτεί τις υπόλοιπες σελίδες.
Έτσι, εάν χρειαστεί να κάνουμε κάποια αλλαγή, όπως να αλλάξουμε το φόντο των σελίδων, το μόνο που έχουμε να κάνουμε είναι να επέμβουμε στο αρχείο CSS και αυτόματα οι αλλαγές θα επηρεάσουν όλα τα αρχεία τα οποία συνδέονται με αυτό.
Η HTML χρησιμοποιείται για να δομήσει το περιεχόμενο, ενώ, τα CSS για να το μορφοποιήσουν. Ας δούμε, για παράδειγμα, την ετικέτα που δηλώνει τις επικεφαλίδες επιπέδου ένα.
Στην HTML θα γράψουμε <h1>Επικεφαλίδα</h1> ενώ η μορφοποίησή της θα έρθει από το CSS: h1 {color:red} και που σημαίνει ότι το χρώμα της επικεφαλίδας θα είναι κόκκινο.
Σε αυτό το σημείο πρέπει να πούμε ότι τα CSS δεν χρησιμοποιούνται μόνο στις ιστοσελίδες αλλά και στο Microsoft Office Word βρίσκουμε τα styles, με τη διαφορά ότι αυτά τα styles μπορούν να μορφοποιούν μόνο κείμενο.
Μερικά από τα πλεονεκτήματα των CSS είναι:

- Διαχωρισμός του περιεχομένου από τη σχεδίαση.
- Ελαχιστοποίηση του χρόνου για τις αλλαγές στη σχεδίαση αφού όλα τα στοιχεία περιέχονται σε ένα αρχείο.
- Περισσότερο καθαρός κώδικας HTML.
- Προσβασιμότητα από όλους τους Web Browsers.

- Έχει πιστοποιηθεί από τον W3C, το μεγαλύτερο οργανισμό Web Standards'.
- Παρέχει αυξημένη ταχύτητα εμφάνισης της ιστοσελίδας.
- Μικρότερο μέγεθος αρχείων.
- Καλύτερη θέση στις μηχανές αναζήτησης λόγω καθαρότερου κώδικα.
- Ομοιόμορφη εμφάνιση όλων των ιστοσελίδων που συνδέονται με το CSS αρχείο αφού τα στοιχεία δεν αλλάζουν.

8.2.7 Σύστημα Διαχείρισης Περιεχομένου

Το Σύστημα Διαχείρισης Περιεχομένου (ΣΔΠ) είναι μία εφαρμογή που χρησιμοποιείται για να δημιουργήσει, να επεξεργαστεί, να διαχειριστεί και να δημοσιεύσει ιστοσελίδες στο διαδίκτυο. Τα ΣΔΠ μπορούν να χρησιμοποιηθούν για να δημιουργηθούν ιστοσελίδες όπως:

- Εταιρικές.
- Προσωπικές.
- Εκπαιδευτικές.
- Ηλεκτρονικά καταστήματα.
- Ενημερωτικές.

Γενικά, ιστοσελίδες μπορούν να καλύψουν όλη σχεδόν τη γκάμα των ενδιαφερομένων. Το περιεχόμενο που μπορεί να χρησιμοποιηθεί περιλαμβάνει κείμενα, εικόνες, ήχους, video, ηλεκτρονικά αρχεία και γενικώς οτιδήποτε μπορεί να διανεμηθεί ,έσω του διαδικτύου.

Ένα ΣΔΠ πρέπει να υποστηρίξει τις παρακάτω ενέργειες:

- Εύκολη διαχείριση περιεχομένου μέσω ενός Web Browser.
- Διαφορετικούς ρόλους και επίπεδα για τους χρήστες του.
- Δυνατότητα δημοσίευσης περιεχομένου από το χρήστη έπειτα από άδεια του διαχειριστή.
- Δυνατότητα κατηγοριοποίησης του περιεχομένου, ώστε να είναι ευκολότερη η διαχείρισή του.
- Διαχωρισμός περιεχομένου και εμφάνισης. Για παράδειγμα, οποιαδήποτε στιγμή να μπορούμε να αλλάξουμε το φόντο της ιστοσελίδας ή το στυλ της γραμματοσειράς μία φορά και να εφαρμοστεί σε όλες τις σελίδες.

CASE STUDY

Υλοποίηση του Δικτυακού Τόπου του Τμήματος ΕΠΔΟ

Η πτυχιακή αυτή εργασία ασχολείται με το σχεδιασμό και την ανάπτυξη του δικτυακού τόπου του τμήματος Εφαρμογών Πληροφορικής στη Διοίκηση και Οικονομία του Παραρτήματος Αμαλιάδας του Ανώτατου Τεχνολογικού Εκπαιδευτικού Ιδρύματος Πατρών. Ο προϋπάρχων δικτυακός τόπος είχε κατασκευασθεί πρόχειρα, για την κάλυψη ορισμένων άμεσων αναγκών που είχαν δημιουργηθεί με το πέρασμα του χρόνου στο Ίδρυμα. Υλοποιήθηκε λοιπόν μια αρχική έκδοση του δικτυακού τόπου, πιο σύγχρονη και συνάμα πιο εύχρηστη και δυναμική.

Ο λόγος για τον οποίο επιλέχθηκε η δημιουργία ενός web site οργανισμού και όχι εταιρείας (όπως περιγράφει ο τίτλος της πτυχιακής εργασίας) ήταν ότι υπήρχε μεγάλη ανάγκη για την αναβάθμιση της ηλεκτρονικής τοποθεσίας του τμήματος. Ωστόσο, η μεθοδολογία που ακολουθήθηκε κατά την κατασκευή του web site ήταν ακριβώς η ίδια με αυτή ενός οργανισμού.

Ύστερα από αρκετές περιηγήσεις στο δικτυακό τόπο του ΤΕΙ Αμαλιάδας, είτε για την εγγραφή των μαθημάτων, είτε για απλή ενημέρωση των νέων του τμήματος, δημιουργήθηκε η ανάγκη κατασκευής και ανανέωσης της ιστοσελίδας με ένα πιο σύγχρονο εργαλείο ανάπτυξης για δικτυακούς τόπους. Και αυτό λόγω του ότι το ήδη υπάρχον site ήταν απλό και λιτό χωρίς τα απαραίτητα στοιχεία/πληροφορίες τα οποία θα ήθελε να βρει ο οποιοσδήποτε, έτσι ώστε να μπορέσει να ενημερωθεί για το τμήμα.

Επίσης, η περιήγηση του δικτυακού τόπου ήταν αρκετά περίπλοκη όπου μετά από μικρό χρονικό διάστημα πλοήγησης μπερδευόσουν. Ακόμα η ανανέωση του δικτυακού τόπου ήταν χρονοβόρα και περίπλοκη για τον διαχειριστή της, λόγω του ότι ήταν στατική. Οπότε αυτό που χρειαζόταν ήταν η αναζήτηση ενός εύχρηστου εργαλείου ανάπτυξης δικτυακών τόπων όπου η ανανέωσή του θα ήταν εύκολη και ο διαχειριστής της θα χαιρόταν να την ανανεώνει χωρίς κόπο και χρόνο.

Ακόμα ένας στόχος ήταν η παροχή στους χρήστες/επισκέπτες για ευχρηστία κινήσεων και ανάγνωσης για τα δεδομένα όπου θα υπάρχουν στο δικτυακό τόπο. Επιπλέον, να παρέχει τη δυνατότητα εύκολης πλοήγησης ακόμα και σε επισκέπτες χωρίς ιδιαίτερες γνώσεις Ηλεκτρονικού Υπολογιστή, όπου θα έχουν την επιθυμία να πληροφορηθούν για οποιοδήποτε θέμα τους απασχολεί σχετικά με το τμήμα.

Η ανάγκη ύπαρξης του συγκεκριμένου δικτυακού τόπου είναι πραγματικά απαραίτητη γιατί θα προσφέρει στους ωφελούμενους όλες τις απαραίτητες πληροφορίες που αφορούν το τμήμα του ΤΕΙ Αμαλιάδας. Θα παρέχει πληροφορίες σχετικά με το κανονισμό σπουδών, τα μαθήματα των εξαμήνων, τα θέματα φοίτησης, τα ευρωπαϊκά προγράμματα, τις εκδηλώσεις

και ακόμα περισσότερα θέματα τα οποία αφορούν κυρίως του σπουδαστές του τμήματος. Επίσης, θα παρέχει πληροφορίες για το προσωπικό το οποίο περιέχεται στο τμήμα της Αμαλιάδας , ωρολόγιο πρόγραμμα για την εξεταστική περίοδο όπου αναφέρεται στο φοιτητικό κοινό του τμήματος. Επιπλέον, μία ακόμα ομάδα προσωπικού η οποία θα μπορεί να έχει ανάμειξη με το site του τμήματος και θα μπορέσει να μείνει ευχαριστημένο με τη δημιουργία του νέου ιστοτόπου, είναι το Διοικητικό Προσωπικό. Η ομάδα του συγκεκριμένου προσωπικού έχει ένα από τα σημαντικότερα κομμάτια που αναλαμβάνουν στο δικτυακό τόπο, το οποίο είναι τα οι Ανακοινώσεις, όπου θα πρέπει να είναι ενήμεροι για τα τελευταία γεγονότα του τμήματος, καθώς επίσης θα πρέπει να επικαιροποιούν τα γεγονότα τα οποία ανεβάζουν στο δικτυακό τόπο. Τέλος, το νέο δικτυακό τόπο, θα μπορούν να τον χρησιμοποιούν απλοί επισκέπτες, όπου θα ενημερώνονται για όλα τα σχετικά αρχεία του τμήματος εύκολα και γρήγορα.

Μεθοδολογία ανάπτυξης και Φάσεις Εφαρμογών Διαδικτύου για το site

Για την υλοποίηση του συγκεκριμένου έργου που αναπτύχθηκε ακολουθήθηκε το μοντέλο διαδικασίας ανάπτυξης λογισμικού ¨Το μοντέλο καταρράκτη¨. Η κεντρική ιδέα αυτού του μοντέλου είναι ότι το σύστημα λογισμικού αναπτύσσεται περνώντας ολόκληρο από διαδοχικές επιμέρους φάσεις, καθεμία από τις οποίες θεωρείται περατωμένη με την παραγωγή ορισμένων συστατικών λογισμικού. Κάθε επιμέρους φάση ολοκληρώνεται με μια εργασία επαλήθευσης/επικύρωσης των προϊόντων της, κατά την οποία αποφασίζεται η μετάβαση ή όχι στην επόμενη. Το λογισμικό εμφανίζεται πλήρες, δηλαδή με όλα τα λειτουργικά του χαρακτηριστικά, από την επιμέρους φάση της συνένωσης και μετά. Χαρακτηριστικό του μοντέλου του καταρράκτη είναι ότι, για να ξεκινήσει μια φάση πρέπει να έχει ολοκληρωθεί πλήρως η προηγούμενη. Η ανάπτυξη με τον τρόπο αυτό χαρακτηρίζεται ακολουθιακή, διότι οι επιμέρους φάσεις από τις οποίες διέρχεται είναι διακριτές και ακολουθούν η μία την άλλη.

- Αρχικά καθορίζονται οι απαιτήσεις από το σύστημα και το λογισμικό.

Προκειμένου να συλλεχτούν οι απαιτήσεις που θα έπρεπε να έχει ο δικτυακός τόπος πρωταρχικό ρόλο έπαιξε η παροχή γνώσεων που ήθελαν να αποκομούν οι φοιτητές μέσα από το συγκεκριμένο Δικτυακό τόπο.

Οπότε, έγινε συνάντηση με τον υπεύθυνο επικοινωνίας του τμήματος και μέσα από μια συνέντευξη καθορίστηκαν οι απαιτήσεις όπου υπήρχαν για τη δημιουργία του site. Επίσης,

συνεντεύξεις πάρθηκαν και από μια άλλη κατηγορία χρηστών που ήταν οι φοιτητές όπου έγινε συνέντευξη με αρκετούς από εκείνους προκειμένου να συλλεχθούν οι απαιτήσεις. Εκτός βέβαια από το παραπάνω κοινό το οποίο βοήθησε στην συλλογή των αναγκών που θα πρέπει να παρέχεται από την ιστοσελίδα, πήρα συνεντεύξεις και από εκπαιδευτικούς, από τον υπεύθυνο γραμματείας και από απλούς χρήστες που επισκεπτόντουσαν το site μόνο από περιέργεια. Οι απαιτήσεις οι οποίες καταγράφηκαν ήταν σχετικές με την αναζήτηση πληροφοριών της πτυχιακής εργασίας, τα εργαστηριακά μαθήματα, την εξεταστική περίοδο, το κανονισμό σπουδών και άλλα παρεμφερή που κάλυπταν το μενού της δικτυακής εφαρμογής. Επίσης έγινε έρευνα στους δικτυακούς τόπους άλλων ιδρυμάτων με την ίδια ειδικότητα του τμήματος όπως επίσης, καταγράφηκαν οι πιο σημαντικές ενότητες που έπρεπε να συμπεριληφθούν μέσα στο δικό μας δικτυακό τόπο.

Το αποτέλεσμα των συνεντεύξεων με τους χρήστες ήταν η σύνταξη του Εντύπου καταγραφής απαιτήσεων όπου αναγράφονται όλα όσα πρέπει να συμπεριλαμβάνονται στην κατασκευή της ιστοσελίδας και επισυνάπτεται στο Παράρτημα Ι.

Έτσι εφόσον καθορίστηκαν ακριβώς οι απαιτήσεις που είχε η δημιουργία αυτής της ιστοσελίδας πέρασα στο επόμενο στάδιο της διαδικασίας ανάπτυξης.

- Ανάλυση και σαφής καθορισμός απαιτήσεων

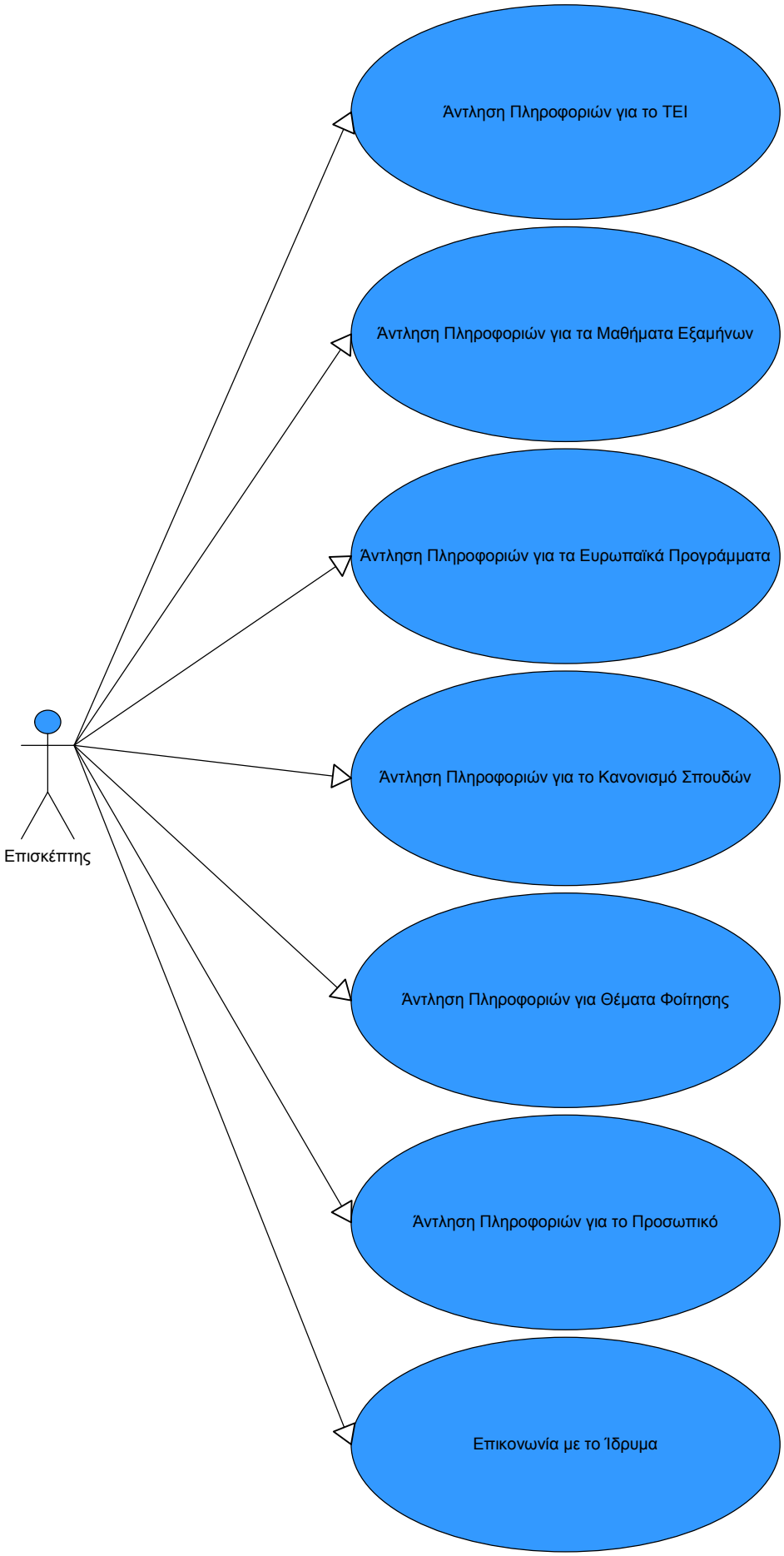
Σε αυτό εδώ το στάδιο του μοντέλου καταρράκτη καθορίστηκαν με σαφήνεια και αναλύθηκαν πιο λεπτομερειακά τις απαιτήσεις που προέκυψαν από την προηγούμενη φάση με τη βοήθεια του εργαλείου visual paradigm.Το συγκεκριμένο εργαλείο με βοήθησε να παρουσιάσω τους βασικούς χρήστες του δικτυακού μου τόπου καθώς επίσης, και τι υπηρεσίες μπορούν να έχουν από αυτόν. Αυτό θα το δούμε παρακάτω από τα σχεδιαγράμματα που ακολουθούν.

Επισκέπτης

Μία από τις υπηρεσίες που προσφέρει ο δικτυακός τόπος στον επισκέπτη είναι η παροχή πληροφοριών με περιεκτικό τρόπο που αφορούν το προπτυχιακό πρόγραμμα Σπουδών του τμήματος. Τα στοιχεία που εμπεριέχονται (μαθήματα, ύλη, πρόγραμμα διδασκαλίας, διδάσκοντες) αποσκοπούν στη διαμόρφωση εκ μέρους των σπουδαστών/τριών μιας σχετικά πλήρους εικόνας των σπουδών και των δυνατοτήτων που παρέχονται από το τμήμα. Επίσης, παρέχονται πληροφορίες που αφορούν τον τρόπο λειτουργίας του τμήματος, δυνατότητα

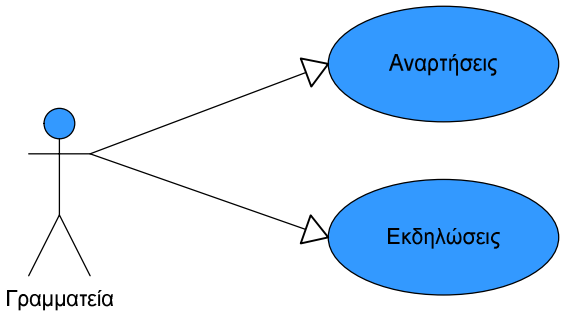
μετακίνησης των σπουδαστών σε ιδρύματα του εξωτερικού μέσω προγραμμάτων κινητικότητας, σε θέματα καθημερινής ζωής στο Παράρτημα Αμαλιάδας αλλά και στην πόλη της Αμαλιάδας γενικότερα.

Σημαντική πληροφορία η οποία παρέχεται από τον ιστότοπο είναι η φόρμα επικοινωνίας όπου δίνει τη δυνατότητα στον επισκέπτη/χρήστη να κάνει τις παρατηρήσεις του ή τις ερωτήσεις που τον απασχολούν.


Άντληση Πληροφοριών για το ΤΕΙ
Άντληση Πληροφοριών για τα Μαθήματα Εξαμήνων
Άντληση Πληροφοριών για τα Ευρωπαϊκά Προγράμματα
Άντληση Πληροφοριών για το Κανονισμό Σπουδών
Άντληση Πληροφοριών για Θέματα Φοίτησης
Άντληση Πληροφοριών για το Προσωπικό
Επικονωνία με το Ίδρυμα
Επισκέπτης

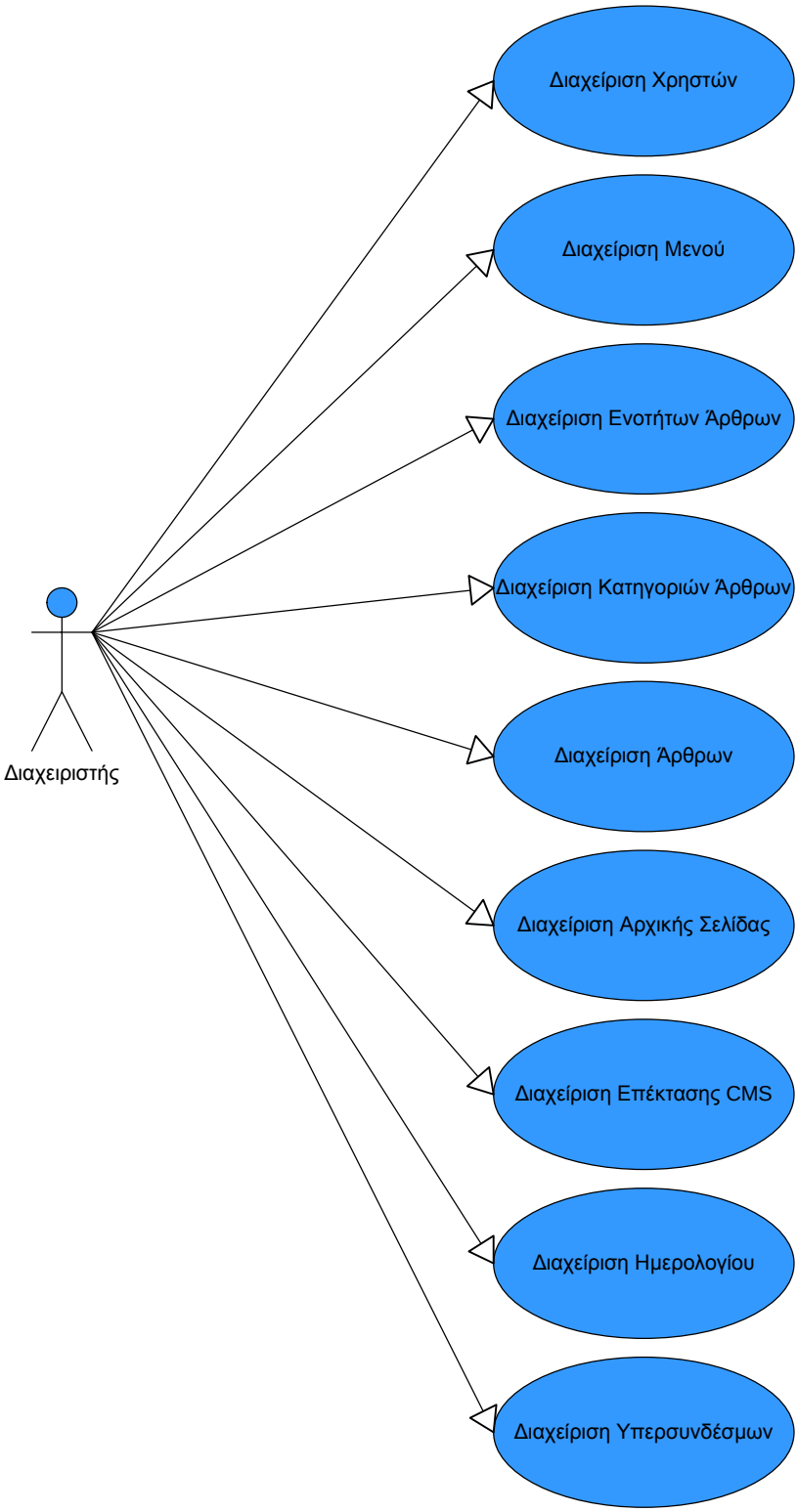
Γραμματεία

Αρμοδιότητα της Γραμματείας είναι η ανάρτηση ανακοινώσεων και εκδηλώσεων προς το κοινό και το διδακτικό προσωπικό, σχετικά με τις δραστηριότητες του ιδρύματος, τη διαμόρφωση του ωρολογίου προγράμματος των εξαμήνων και της εξεταστικής περιόδου, καθώς επίσης και των διαφόρων εκδηλώσεων που λαμβάνουν χώρα (π.χ. ορκωμοσίες κλπ).



Διαχειριστής

Ο Διαχειριστής είναι εκείνος ο οποίος έχει την μεγαλύτερη επίβλεψη του δικτυακού τόπου, όπως φαίνεται παρακάτω, με την έννοια πως όλο το διαχειριστικό κομμάτι το επεξεργάζεται ο ίδιος. Για οποιεσδήποτε αλλαγές πρόκυπτε να γίνουν στο δικτυακό τόπο είναι ο μόνος άνθρωπος που μπορεί να τις υλοποιήσει αλλά και να πάρει θέση για την καλύτερη τοποθέτηση της αλλαγής αυτής μέσα στο site.

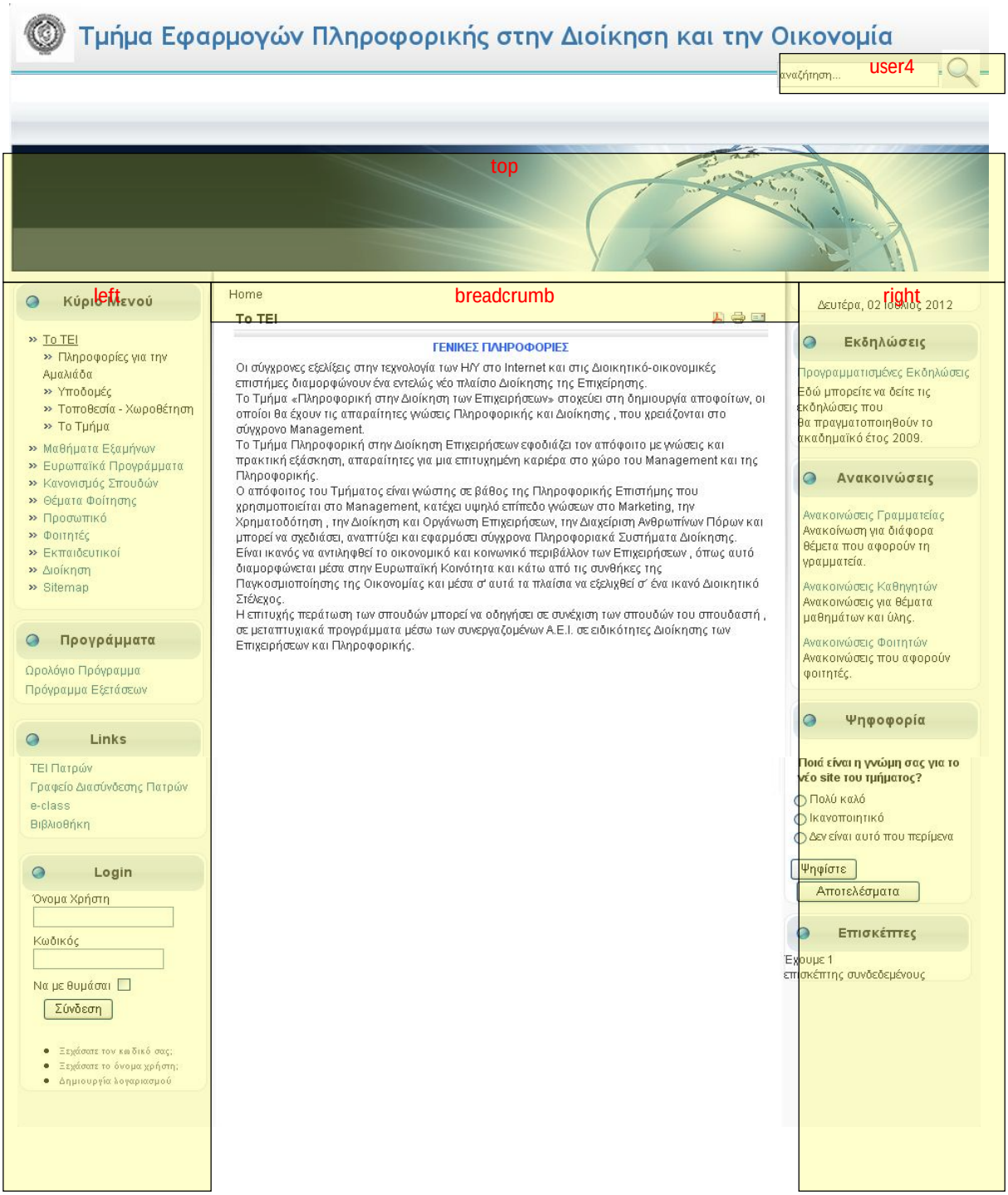

Διαχειριστής
Διαχείριση Χρηστών
Διαχείριση Μενού
Διαχείριση Ενοτήτων Άρθρων
Διαχείριση Κατηγοριών Άρθρων
Διαχείριση Άρθρων
Διαχείριση Αρχικής Σελίδας
Διαχείριση Επέκτασης CMS
Διαχείριση Ημερολογίου
Διαχείριση Υπερσυνδέσμων

- Σχεδιασμός συστήματος και λογισμικού.

Για τη σχεδίαση της ιστοσελίδας έπρεπε πρώτα απ'όλα η εφαρμογή να εγκατασταθεί σε έναν υπολογιστή ο οποίος θα χρησιμοποιούνταν για ανάπτυξη όλα τα προαπαιτούμενα για την φιλοξενία του διαδικτυακού τόπου του τμήματος ΕΠΔΟ. Συγκεκριμένα θα υποστήριζε την λειτουργία του apache Web Server την βάση δεδομένων MySQL, και την γλώσσα προγραμματισμού php. Στην συνέχεια έγινε εγκατάσταση του CMS joomla που παρέχεται από το διαδίκτυο δωρεάν. Όταν ολοκληρώθηκε η φάση της εγκατάστασης των προαπαιτούμενων και λογισμικού δημιουργίας ιστοσελίδας, έπρεπε να παρθούν αποφάσεις σχετικά με το τι πληροφορίες θα είχαν τα μενού τα οποία είχαν δημιουργηθεί. Οι πληροφορίες αυτές οι οποίες είναι σε θέση να εισαχθούν στα επιμέρους μενού που έχουν δημιουργηθεί στην προηγούμενη φάση, αφορούν κυρίως το αντικείμενο της ειδικότητας του τμήματος, το αναλυτικό πρόγραμμα φοίτησης όλων των εξαμήνων, με τις παροχές που έχει κάθε φοιτητής όταν σπουδάζει στο συγκεκριμένο τμήμα, το εκπαιδευτικό προσωπικό, το διοικητικό προσωπικό, τα ευρωπαϊκά προγράμματα που παρέχονται σε συνεργασία με τη σχολή και άλλα σχετικά τα οποία θα ολοκληρώσουν τα μενού. Επιπροσθέτως, θα πρέπει στο δικτυακό τόπο να υπάρχει μια αρχική σελίδα με γενική περιγραφή του τμήματος, όπως επίσης και πληροφορίες που αναζητά ένας πρωτοετής σπουδαστής σχετικά με την τοποθεσία της σχολής αλλά και το πώς θα επικοινωνήσει μαζί τους. Όλες αυτές οι πληροφορίες δημιουργήθηκαν από την φάση του καθορισμού απαιτήσεων. Μετά από αυτή τη καταγραφή των πληροφοριών επικράτησαν η κατηγορία:

- **ü** Κύριο μενού: είναι το μενού επιλογών όπου παρέχει πληροφορίες σχετικά με το τμήμα του ιδρύματος, τις σπουδές, το προσωπικό των καθηγητών, το κανονισμό σπουδών, τα θέματα φοίτησης, τη διοίκηση και τα ευρωπαϊκά προγράμματα.

Μετά την καταγραφή των πληροφοριών και τον διαχωρισμό των μενού, διαμορφώθηκε η πρώτη σελίδα όπως αυτή απεικονίζεται παρακάτω:



Στην κορυφή της ιστοσελίδας τοποθετήθηκε το μενού κορυφής με τις περισσότερο αναζητούμενες πληροφορίες των νεοεισαχθέντων φοιτητών, όπου τους βοηθά να πληροφορηθούν άμεσα (top).

Ακριβώς δίπλα από το μενού κορυφής τοποθετήθηκε η μηχανή αναζήτησης για γρήγορες αναζητήσεις μέσα στο δικτυακό τόπο(user 4). Το κύριο μενού τοποθετήθηκε στην αριστερή πλευρά της ιστοσελίδας λόγω ότι διακρίνεται περισσότερο απ'ότι στη δεξιά μεριά, μαζί με όλες τις χρήσιμες πληροφορίες που αναζητά ο κάθε επισκέπτης για το τμήμα (left). Επίσης, στην αριστερή πλευρά κάτω από το κύριο μενού τοποθετήθηκε ένα block με πληροφορίες σχετικά με τα προγράμματα σπουδών του τμήματος όπως π.χ. το Ωρολόγιο Πρόγραμμα και το Πρόγραμμα Εξετάσεων (left). Κάτω από αυτό το block εισήχθη ένας πίνακας με τις πιο χρήσιμες διευθύνσεις όπου θα μπορεί ο σπουδαστής καθ'όλη τη διάρκεια των σπουδών του να αναζητήσει πληροφορίες από άλλους δικτυακούς τόπους που τον παραπέμπουμε.Επίσης, κάτω από τα Links, προστέθηκε το login του δικτυακού τόπου όπου θα κάνουν μόνο όσοι έχουν πάρει δικαιώματα από τον διαχειριστή (left).

Στην πάνω δεξιά μεριά της σελίδας τοποθετήθηκε η μηχανή αναζήτησης όπου γίνεται αντιληπτή από οποιονδήποτε χρήστη που θα θελήσει να αναζητήσει την πληροφορία του γρηγορότερα (right).

Στην κάθετη δεξιά στήλη προσθέσαμε αρχικά την τρέχουσα ημερομηνία, όπου πληροφορεί τον χρήστη για την ημέρα και το μήνα που κάνει την αναζήτησή του (right). Ακόμα, κάτω από αυτό εμφανίζεται η περιοχή όπου μας ενημερώνει για τις προγραμματισμένες εκδηλώσεις που πραγματοποιούνται από τι τμήμα (right). Επίσης, κάτω ακριβώς από τις εκδηλώσεις προσθέσαμε μια περιοχή όπου θα αναγράφονται οι πιο Ανακοινώσεις του τμήματος και θα ενημερώνει το κοινό σχετικά με πρόσφτα θέματα που ανακοινώνει η γραμματεία, οι καθηγητές καθώς και οι φοιτητές (right). Επίσης, κάτω από τις ανακοινώσεις θεωρήθηκε ότι καλό είναι να παρουσιάζεται η περιοχή με την ψηφοφορία όπου κρατάει τα στατιστικά της σελίδας σχετικά με το σύνολο των ψήφων, την ημερομηνία της πρώτης και της τελευταίας ψήφου. Ακόμα, αμέως κάτω από την ψηφοφορία κρίθηκε καλό να προστεθεί η ενημερωτική περιοχή με την επισκεψιμότητα του site (right), πχ πόσους επισκέπτες έχουμε συνδεδεμένους στη σελίδα μας όταν εισερχόμαστε. Τέλος, ενδιάμεσα στην δεξιά και αριστερή στήλη διακρίνεται το μονοπάτι της ιστοσελίδας, όπου σε ενημερώνει κάθε φορά σε ποιο σημείο βρίσκεσαι (breadcrumb).

Μενού Επιλογών

Το ΤΕΙ
- Πληροφορίες για την Αμαλιάδα
- Υποδομές
- Τοποθεσία - Χωροθέτηση
- Το Τμήμα

Μαθήματα Εξαμήνων
- Εξάμηνο Α
- Εξάμηνο Β
- Εξάμηνο Γ
- Εξάμηνο Δ
- Εξάμηνο Ε
- Εξάμηνο ΣΤ
- Εξάμηνο Ζ
- Εξάμηνο Η
- Αλυσίδες Μαθημάτων

Ευρωπαϊκά Προγράμματα
- SOCRATES
- ERASMUS
- LEONARDO DA VINCI
- COMENIUS
- DRUNDTVING
- MINERVA
- LINGUA

Κανονισμός Σπουδών

Θέματα Φοίτησης
- Εγγραφές
- Μετεγγραφές
- Αιτήσεις - Πιστοποιητικά
- Εστία - Σίτιση

Προσωπικό
- Μόνιμο
- Έκτακτο
- Διοικητικό

Φοιτητές
- Επαγγελματικά Δικαιώματα
- Δικαιώματα Φοιτητών
- Πρακτική Άσκηση
- Πτυχιακή Εργασία

Εκπαιδευτικοί
- Δικαιώματα Εκπαιδευτικών
- Υποχρεώσεις Εκπαιδευτικών

- Συναντήσεις
- Δικτυακός Χώρος Εκπαιδευτικών

Διοίκηση

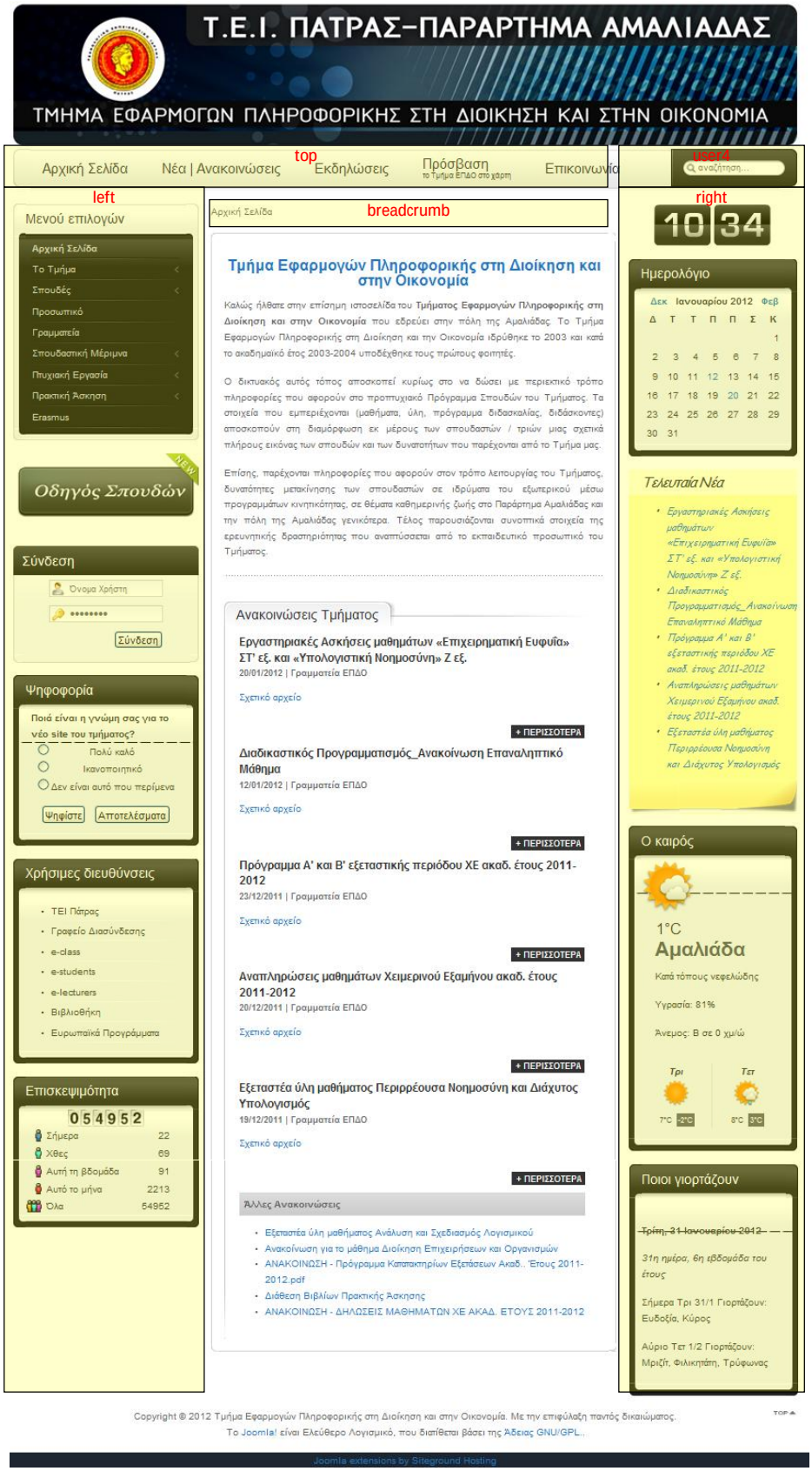
- Το Συμβούλιο
- Πρακτικά Συμβουλίου
- Οργανόγραμμα

Sitemap

Το παραπάνω παραδοτέο site υλοποιήθηκε ανάλογα με τις ανάγκες που υπήρχαν στο τμήμα ΕΠΔΟ του ΤΕΙ Αμαλιάδας την περίοδο όπου ξεκίνησα την εκπόνηση της πτυχιακής μου εργασίας. Η μορφή η οποία είχε όταν την παράδωσα στο ίδρυμα της Αμαλιάδας ήταν ακριβώς όπως στην παραπάνω εικόνα.

Κατόπιν παράδοσης του δικτυακού τόπου στο ΤΕΙ Αμαλιάδας της σχολής Εφαρμογών Πληροφορικής στη Διοίκηση και Οικονομία εντοπίστηκε η ανάγκη για την αναβάθμισή του από την ομάδα εργασίας του ΤΕΙ λόγω αισθητικού χαρακτήρα καθώς επίσης και ορισμένων επιπλέων blocks τα οποία θα βοηθούσαν το κοινό για την άντληση περισσότερων πληροφοριών.

Στο παρακάτω στιγμιότυπο του δικτυακού τόπου απεικονίζεται η μορφή η οποία εμφανίζει σήμερα η παραπάνω ηλεκτρονική τοποθεσία, καθώς επίσης, αναλύεται και η τοποθέτηση των προϋπαρχόντων/νέων άρθρων και κατηγοριών.


Τ.Ε.Ι. ΠΑΤΡΑΣ–ΠΑΡΑΡΤΗΜΑ ΑΜΑΛΙΑΔΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΓΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΣΤΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΣΤΗΝ ΟΙΚΟΝΟΜΙΑ
top
Αρχική Σελίδα
Νέα | Ανακοινώσεις
Εκδηλώσεις
Πρόσβαση
Επικοινωνία
user4
left
Μενού επιλογών
Αρχική Σελίδα
Το Τμήμα
Σπουδές
Προσωπικό
Γραμματεία
Σπουδαστική Μέριμνα
Πτυχιακή Εργασία
Πρακτική Άσκηση
Erasmus
Οδηγός Σπουδών
Σύνδεση
Όνομα Χρήστη
Σύνδεση
Ψηφοφορία
Ποιά είναι η γνώμη σας για το νέο site του τμήματος?
Πολύ καλό
Ικανοποιητικό
Δεν είναι αυτό που περίμενα
Ψηφίστε
Αποτελέσματα
Χρήσιμες διευθύνσεις
ΤΕΙ Πάτρας
Γραφείο Διασύνδεσης
e-class
e-students
e-lecturers
Βιβλιοθήκη
Ευρωπαϊκά Προγράμματα
Επισκεψιμότητα
054952
Σήμερα 22
Χθες 69
Αυτή τη βδομάδα 91
Αυτό το μήνα 2213
Όλα 54952
Αρχική Σελίδα
breadcrumb
Τμήμα Εφαρμογών Πληροφορικής στη Διοίκηση και στην Οικονομία
Καλώς ήλθατε στην επίσημη ιστοσελίδα του Τμήματος Εφαρμογών Πληροφορικής στη Διοίκηση και στην Οικονομία που εδρεύει στην πόλη της Αμαλιάδας. Το Τμήμα Εφαρμογών Πληροφορικής στη Διοίκηση και την Οικονομία ιδρύθηκε το 2003 και κατά το ακαδημαϊκό έτος 2003-2004 υποδέχθηκε τους πρώτους φοιτητές.
Ο δικτυακός αυτός τόπος αποσκοπεί κυρίως στο να δώσει με περιεκτικό τρόπο πληροφορίες που αφορούν στο προπτυχιακό Πρόγραμμα Σπουδών του Τμήματος. Τα στοιχεία που εμπεριέχονται (μαθήματα, ύλη, πρόγραμμα διδασκαλίας, διδάσκοντες) αποσκοπούν στη διαμόρφωση εκ μέρους των σπουδαστών / τριών μιας σχετικά πλήρους εικόνας των σπουδών και των δυνατοτήτων που παρέχονται από το Τμήμα μας.
Επίσης, παρέχονται πληροφορίες που αφορούν στον τρόπο λειτουργίας του Τμήματος, δυνατότητες μετακίνησης των σπουδαστών σε ιδρύματα του εξωτερικού μέσω προγραμμάτων κινητικότητας, σε θέματα καθημερινής ζωής στο Παράρτημα Αμαλιάδας και την πόλη της Αμαλιάδας γενικότερα. Τέλος παρουσιάζονται συνοπτικά στοιχεία της ερευνητικής δραστηριότητας που αναπτύσσεται από το εκπαιδευτικό προσωπικό του Τμήματος.
Ανακοινώσεις Τμήματος
Εργαστηριακές Ασκήσεις μαθημάτων «Επιχειρηματική Ευφυΐα» ΣΤ' εξ. και «Υπολογιστική Νοημοσύνη» Ζ εξ.
20/01/2012 | Γραμματεία ΕΠΔΟ
Σχετικό αρχείο
+ ΠΕΡΙΣΣΟΤΕΡΑ
Διαδικαστικός Προγραμματισμός_Ανακοίνωση Επαναληπτικό Μάθημα
12/01/2012 | Γραμματεία ΕΠΔΟ
Σχετικό αρχείο
+ ΠΕΡΙΣΣΟΤΕΡΑ
Πρόγραμμα Α' και Β' εξεταστικής περιόδου ΧΕ ακαδ. έτους 2011-2012
23/12/2011 | Γραμματεία ΕΠΔΟ
Σχετικό αρχείο
+ ΠΕΡΙΣΣΟΤΕΡΑ
Αναπληρώσεις μαθημάτων Χειμερινού Εξαμήνου ακαδ. έτους 2011-2012
20/12/2011 | Γραμματεία ΕΠΔΟ
Σχετικό αρχείο
+ ΠΕΡΙΣΣΟΤΕΡΑ
Εξεταστέα ύλη μαθήματος Περιρρέουσα Νοημοσύνη και Διάχυτος Υπολογισμός
19/12/2011 | Γραμματεία ΕΠΔΟ
Σχετικό αρχείο
+ ΠΕΡΙΣΣΟΤΕΡΑ
Άλλες Ανακοινώσεις
Εξεταστέα ύλη μαθήματος Ανάλυση και Σχεδιασμός Λογισμικού
Ανακοίνωση για το μάθημα Διοίκηση Επιχειρήσεων και Οργανισμών
ΑΝΑΚΟΙΝΩΣΗ - Πρόγραμμα Καπακτηρίων Εξετάσεων Ακαδ. Έτους 2011-2012.pdf
Διάθεση Βιβλίων Πρακτικής Άσκησης
ΑΝΑΚΟΙΝΩΣΗ - ΔΗΛΩΣΕΙΣ ΜΑΘΗΜΑΤΩΝ ΧΕ ΑΚΑΔ. ΕΤΟΥΣ 2011-2012
right
10 34
Ημερολόγιο
Δεκ Ιανουαρίου 2012 Φεβ
Δ Τ Τ Π Π Σ Κ
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
Τελευταία Νέα
Εργαστηριακές Ασκήσεις μαθημάτων «Επιχειρηματική Ευφυΐα» ΣΤ' εξ. και «Υπολογιστική Νοημοσύνη» Ζ εξ.
Διαδικαστικός Προγραμματισμός_Ανακοίνωση Επαναληπτικό Μάθημα
Πρόγραμμα Α' και Β' εξεταστικής περιόδου ΧΕ ακαδ. έτους 2011-2012
Αναπληρώσεις μαθημάτων Χειμερινού Εξαμήνου ακαδ. έτους 2011-2012
Εξεταστέα ύλη μαθήματος Περιρρέουσα Νοημοσύνη και Διάχυτος Υπολογισμός
Ο καιρός
1°C
Αμαλιάδα
Κατά τόπους νεφελώδης
Υγρασία: 81%
Άνεμος: Β σε 0 χμ/ώ
Τρι 7°C -2°C
Τετ 8°C 3°C
Ποιοι γιορτάζουν
Τρίτη, 31 Ιανουαρίου 2012
31η ημέρα, 6η εβδομάδα του έτους
Σήμερα Τρι 31/1 Γιορτάζουν: Ευδοξία, Κύρος
Αύριο Τετ 1/2 Γιορτάζουν: Μριζή, Φιλικητάτη, Τρύφωνας
Copyright © 2012 Τμήμα Εφαρμογών Πληροφορικής στη Διοίκηση και στην Οικονομία. Με την επιφύλαξη παντός δικαιώματος.
Το Joomla! είναι Ελεύθερο Λογισμικό, που διατίθεται βάσει της Άδειας GNU/GPL.
TOP
Joomla extensions by Siteground Hosting

Στην κορυφή της ιστοσελίδας τοποθετήθηκε το μενού κορυφής με τις περισσότερο αναζητούμενες πληροφορίες των νεοεισαχθέντων φοιτητών, όπου τους βοηθά να πληροφορηθούν άμεσα (top).

Ακριβώς δίπλα από το μενού κορυφής τοποθετήθηκε η μηχανή αναζήτησης για γρήγορες αναζητήσεις μέσα στο δικτυακό τόπο(user 4). Το κύριο μενού τοποθετήθηκε στην αριστερή πλευρά της ιστοσελίδας λόγω ότι διακρίνεται περισσότερο απ'ότι στη δεξιά μεριά, μαζί με όλες τις χρήσιμες πληροφορίες που αναζητά ο κάθε επισκέπτης για το τμήμα (left). Επίσης, στην αριστερή μεριά κάτω από το κύριο μενού τοποθετήθηκε ένα κουμπί με πληροφορίες σχετικά με τον οδηγό σπουδών του τμήματος (left). Κάτω από αυτό το κουμπί εισήχθη το login του δικτυακού τόπου όπου θα κάνουν μόνο όσοι έχουν πάρει δικαιώματα από τον διαχειριστή (left). Αμέσως μετά εμφανίζεται ένας πίνακας με τις χρήσιμες διευθύνσεις που θα χρησιμοποιήσει ο σπουδαστής καθ'όλη τη διάρκεια των σπουδών του (left). Και κάτω από αυτό εμφανίζεται το πινακάκι το οποίο μας πληροφορεί για την επισκεψιμότητα του δικτυακού τόπου (left).

Στην πάνω δεξιά μεριά και κάτω από τη μηχανή αναζήτησης τοποθετήθηκε ένα ρολόϊ (right). Αμέσως, κάτω από αυτό εμφανίζεται το ημερολόγιο όπου μας βοηθάει να αναζητήσουμε διάφορες ημερομηνίες που μας ενδιαφέρουν (right)(π.χ. ημέρα εξέτασης κάποιου μαθήματος). Τα τελευταία νέα του τμήματος φαίνονται κάτω από το ημερολόγιο όπου παρέχουν όλα τα τελευταία νέα της σχολής (right). Επίσης, κάτω από τα τελευταία νέα θεωρήθηκε ότι καλό είναι να φαίνεται η ημερήσια πρόβλεψη του καιρού της Αμαλιάδας όπως και τοποθετήθηκε (right).Στην τελευταία γωνία της δεξιάς μεριάς και κάτω από τον πίνακα με την πρόγνωση του καιρού τοποθετήθηκε ένας πίνακας που παρέχει πληροφορίες σχετικά με τις ημερήσιες εορτές (right).

Τέλος, στη μέση του δικτυακού τόπου παρέχεται χώρος έτσι ώστε να διακρίνονται οι ανακοινώσεις που βγάζει το τμήμα ανά πάσα στιγμή. Ακόμα κάτω από το μενού κορυφής υπάρχει το μονοπάτι της ιστοσελίδας, όπου σε ενημερώνει κάθε φορά σε ποιο σημείο βρίσκεσαι (breadcrumb).

- Υλοποίηση και Ενοποίηση δοκιμών του συστήματος.

Εφόσον ολοκληρώθηκε το στάδιο της σχεδίασης περνάμε στην επόμενη φάση του μοντέλου όπου γίνεται η ενοποίηση των εφαρμογών που έχουμε αναπτύξει μέχρι το προηγούμενο στάδιο και δοκιμάζουμε να δούμε αν η ενοποιημένη εφαρμογή λειτουργεί. Εφόσον είχαν

συλλεχθεί από το διαδίκτυο οι εφαρμογές οι οποίες είχαν ενδιαφέρον προστέθηκαν στο δικτυακό τόπο (ένα ρολόϊ, ένα ημερολόγιο, την εφαρμογή της εναλλαγής της γλώσσας) ενοποιήθηκαν με την υπάρχον εφαρμογή από την προηγούμενη φάση και τέλος έπρεπε να ερευνηθεί το Σύστημα Περιεχομένου του δικτυακού τόπου έτσι ώστε να μπορέσουμε να δούμε τι ακριβώς είχε δημιουργηθεί, τι δεν εμφανιζόταν από αυτά που είχαν εγκατασταθεί και τι ακόμα χρειαζόταν για να ολοκληρωθεί.

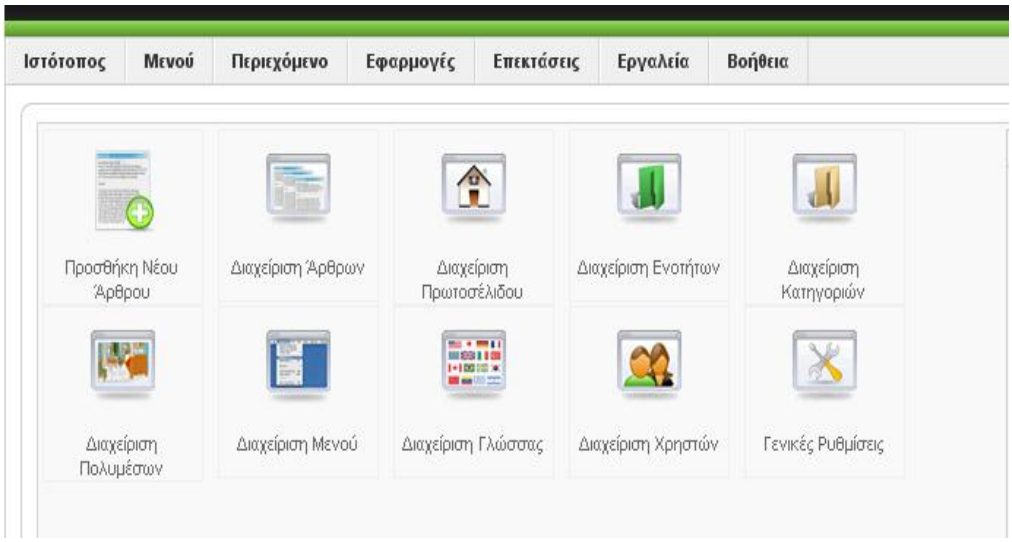
Αυτό που ελέχθηκε και που συνήθως παρουσιάζει προβλήματα ήταν η εναλλαγή της γλώσσας. Όντως κατά τη φάση του ελέγχου παρουσιάστηκε πρόβλημα όπου μετά από διορθώσεις κατάφερε να επιλυθεί.

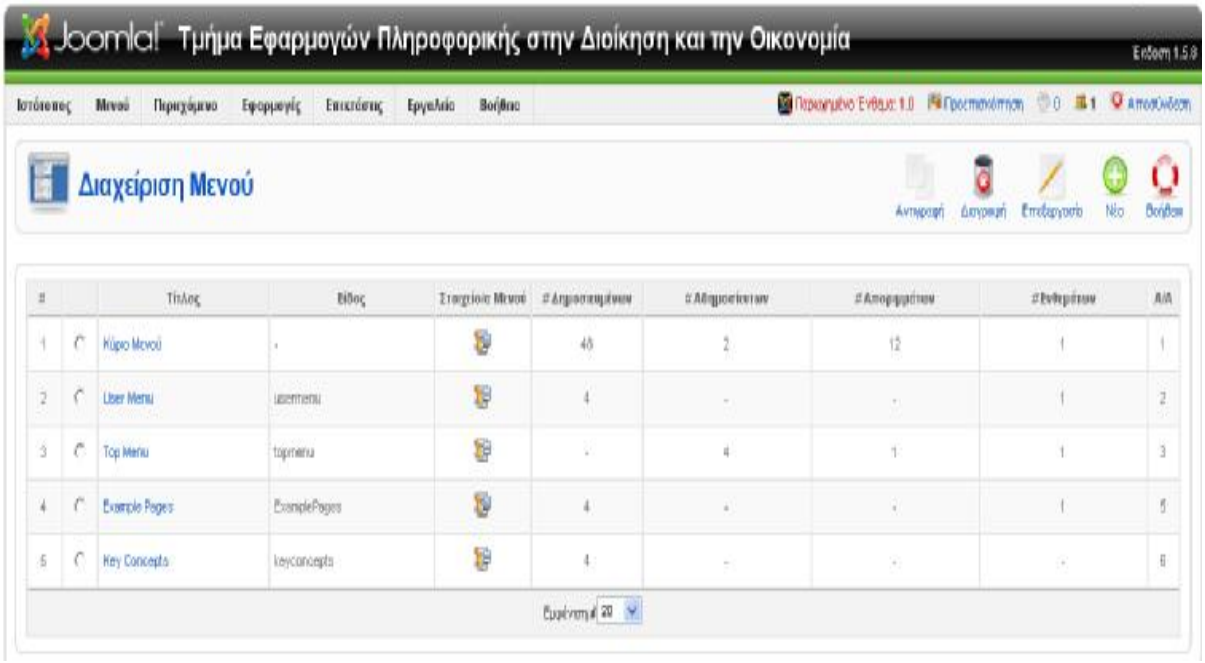
Επίσης, για την ολοκλήρωση της φάσης της υλοποίησης του δυναμικού δικτυακού τόπου που κατασκευάστηκε χρησιμοποιήθηκε ανοιχτός κώδικας (open source) με το Σύστημα Διαχείρισης Περιεχομένου (CMS-Content Management System) «joomla!» έκδοσης 1.5.8, με κατάλληλη παραμετροποίηση μέσω του διαχειριστικού περιβάλλοντος. Ο αναλυτικός οδηγός διαχείρισης του administrator back-end του CMS προσφέρεται στη διεύθυνση http://docs.joomla.org/Administrators**[D]**.

Λόγω μεγάλης έκτασης των εκτελεσμένων εργασιών παραμετροποίησης του έργου, ακολούθως παρατίθενται δειγματοληπτικά χαρακτηριστικά σημεία της διαδικασίας:

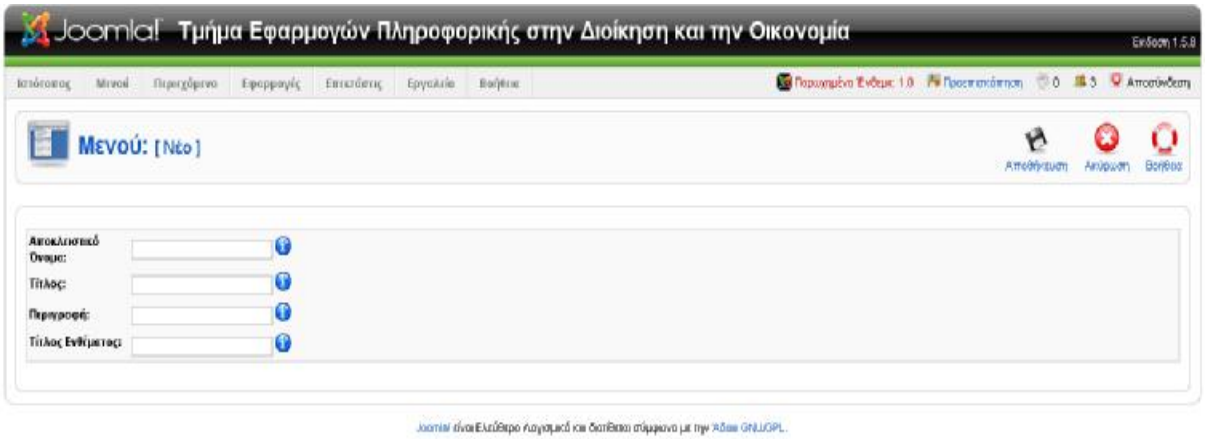
Για την δημιουργία νέου Μενού υλοποιούνται τα εξής βήματα:

- Εφόσον εισερχόμαστε στο Μενού -> Διαχείριση μενού, κάνουμε κλικ στο κουμπί Νέο που βρίσκεται στο επάνω δεξιά μέρος των εργαλείων. Εμφανίζεται μια οθόνη με κάποιες επιλογές. Τα πεδία με τα πρέπει να ασχοληθούμε αναφέρονται παρακάτω.



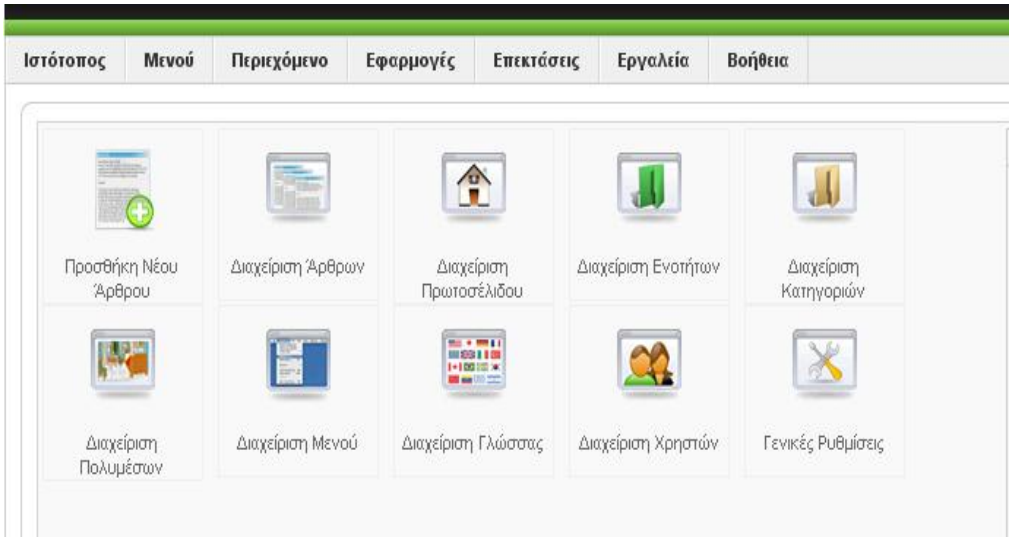


- Στο πεδίο τίτλος εισήχθη η ονομασία που θέλαμε να έχει το μενού μου στην περιοχή διαχείρισης των μενού (π.χ. Κύριο Μενού)
- Στο πεδίο Περιγραφή εισήχθη ακριβώς το ίδιο λεκτικό με τον τίτλο, δηλαδή Κύριο Μενού.
- Κάνουμε Κλικ στην αποθήκευση και μόλις κλείσει το παράθυρο η δημιουργία του μενού αποθηκεύεται.

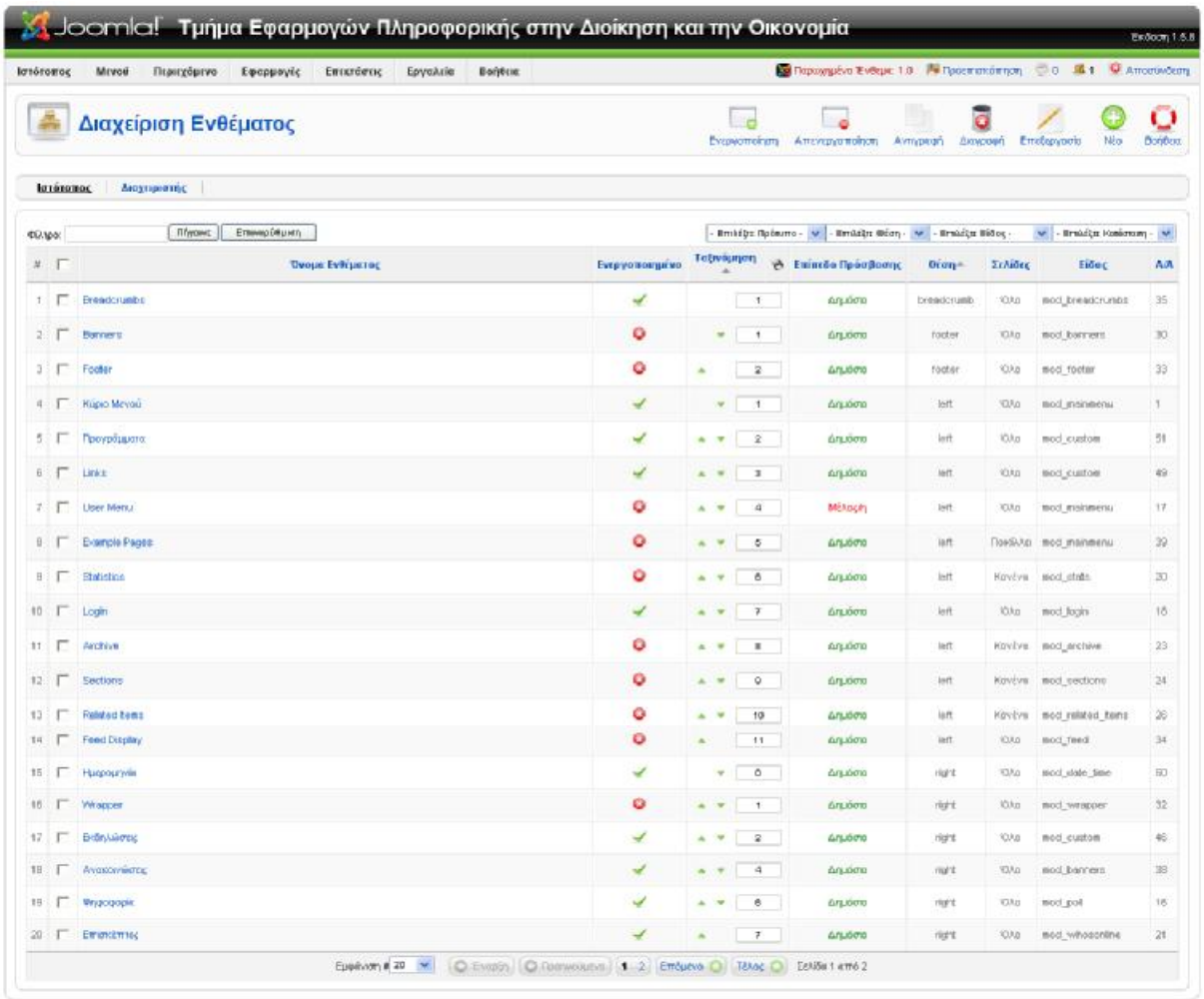


2)Για τον καθορισμό τοποθεσίας ενθεμάτων σε positions του design template ακολουθούμε την παρακάτω διαδικασία:

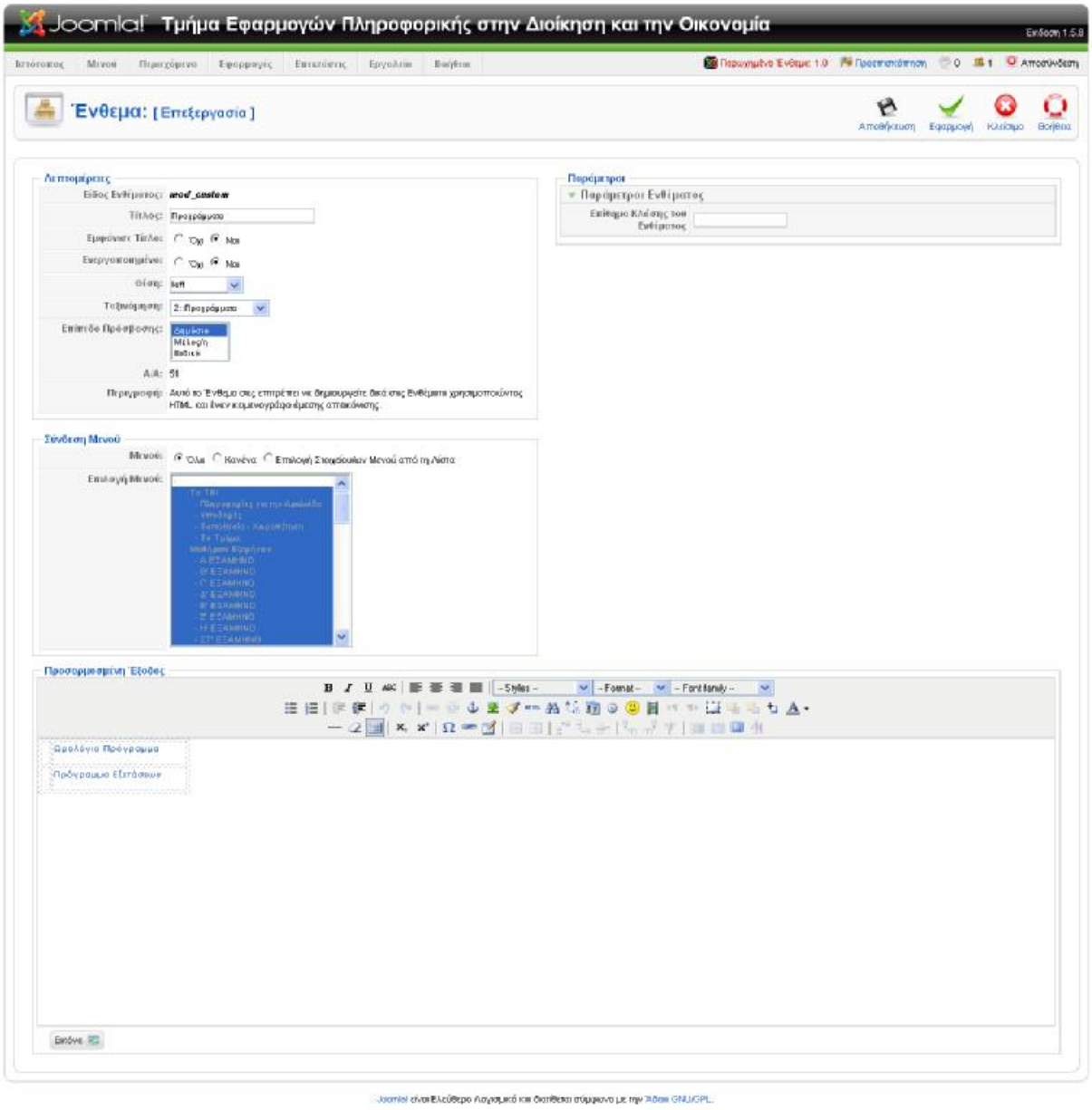
- Στο περιβάλλον διαχείρισης επιλέγουμε την εντολή Διαχείριση Ενθεμάτων από το μενού Επεκτάσεις.



- Κλικάρουμε το ένθεμα το οποίο θέλουμε να το τοποθετήσουμε σε κατάλληλη θέση (π.χ. το ένθεμα προγράμματα να το τοποθετήσουμε στην αριστερή μεριά του template).



- Εμφανίζεται ένα παράθυρο με διάφορες λεπτομέρειες και παραμέτρους που αφορούν το ένθεμα.



- Στις λεπτομέρειες υπάρχει μια επιλογή με το όνομα θέση, όπου μας δίνει τη δυνατότητα να τοποθετήσουμε το ένθεμά μας στη θέση που θέλουμε.
- Εφόσον επιλέξουμε την θέση του ενθέματος κάνουμε κλικ στο κουμπί Αποθήκευση έτσι ώστε να αποθηκευτεί η αλλαγή που πραγματοποιήσαμε.

Ακόμα, για την ανάπτυξη του δικτυακού τόπου διαμορφώθηκε κατάλληλο περιβάλλον που περιελάμβανε τα εξής συστατικά διακομιστή (web & database server, server site scripting language):

- Apache version 2.2.9
- My sql version 5.0.67 community -nt

- Ø Php version 5.2.6

- Λειτουργία και συντήρηση.

Τέλος, σε αυτή τη φάση εφόσον όλα ήταν έτοιμα έγινε η εγκατάσταση της διαδικτυακής εφαρμογής στους servers του ΤΕΙ και οι τελικές δοκιμές λειτουργίας πριν βγει στον αέρα.
Οι αλλαγές και οι διορθώσεις που θα χρειαστούν από αυτό το στάδιο και μετέπειτα το έχει ήδη αναλάβει άλλος διαχειριστής.

Βελτιώσεις / Επεκτάσεις Συστήματος

Ο δικτυακός τόπος δύναται να επεκταθεί περαιτέρω με την προσθήκη νέας λειτουργικότητας, η οποία θα ικανοποιεί μελλοντικές απαιτήσεις που θα προκύψουν.

Επιπλέον, η επιλογή του “joomla!” CMS για την υλοποίηση του ιστοτόπου συμβάλλει τα μέγιστα σε μια πιθανή επέκταση, λόγω των εκατοντάδων χιλιάδων πρόσθετων τμημάτων λογισμικού που διαθέτει, αλλά και της δυνατότητας που προσφέρει για δημιουργία κατά παραγγελία νέων πρόσθετων τμημάτων κώδικα, καλύπτοντας κάθε ανάγκη.

Συγκεκριμένα, βασικές/χρήσιμες δυνατότητες που θα μπορούσαν να προστεθούν είναι:

Δημιουργία πύλης εισόδου:

1. Φοιτητών: για την παρακολούθηση της προόδου τους, την άντληση εκπαιδευτικού υλικού, την ανάρτηση εργασιών τους κλπ.
2. Ακαδημαϊκού προσωπικού: για βαθμολόγηση φοιτητών, ανάρτηση εκπαιδευτικού υλικού κλπ.
3. Για διοικητική εξυπηρέτηση ακαδημαϊκού προσωπικού/φοιτητών (π.χ. κατάθεση αιτημάτων έκδοσης βεβαιώσεων φοίτησης/προϋπηρεσίας κλπ)

Επιπρόσθετα, μια βασική βελτίωση που θα μπορούσε να εφαρμοστεί στο δικτυακό τόπο, είναι η αναβάθμιση του παρεχόμενου επιπέδου ασφαλείας του όσον αφορά την front-end και back-end πρόσβαση των χρηστών, με χρήση κατάλληλου πρόσθετου λογισμικού.

Τέλος, είναι πολύ σημαντική η συνεχής αναβάθμιση της εγκατεστημένης έκδοσης του “joomla” (εφαρμογή διαφόρων patches ή εγκατάσταση νέων εκδόσεων), για την κάλυψη των συχνά εμφανιζόμενων κενών ασφαλείας, που ανακαλύπτονται διαρκώς από την τεράστια παγκόσμια κοινότητα υποστήριξής του.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Η εκπόνηση της παρούσας εργασίας ανέδειξε το πόσο σημαντική είναι η υιοθέτηση μιας δεδομένης ακολουθίας αυστηρά καθορισμένων βημάτων για την ανάπτυξη ενός έργου λογισμικού.

Τα στάδια τα οποία περιλαμβάνονται στο μοντέλο του καταρράκτη (το οποίο επιλέγη μεταξύ των διαθέσιμων μοντέλων κύκλου ζωής λογισμικού), διαγράφουν το περίγραμμα της πορείας που πρέπει να ακολουθηθεί, ώστε να επιτευχθεί το επιθυμητό αποτέλεσμα.

Η φάση σταθμός, η οποία χρήζει ιδιαίτερης σημασίας είναι η «Ανάλυση και ο καθορισμός απαιτήσεων». Η επιλογή του εν λόγω μοντέλου για την ανάπτυξη ενός έργου προϋποθέτει καταρχήν καλή γνώση των απαιτήσεων που καλείται να ικανοποιήσει το σύστημα.

Ο διαχωρισμένος σε διακριτά επίπεδα κύκλος ανάπτυξης που προσφέρει το μοντέλο του καταρράκτη επιβάλλει πειθαρχία και μειώνει το ρίσκο απόκλισης από το χρονοδιάγραμμα ή της μη ικανοποίησης των τεθέντων προδιαγραφών, διασφαλίζοντας μια σχετικά «βατή» οδό προς την ολοκλήρωση του υπό ανάπτυξη έργου.

Όσο νωρίτερα ολοκληρωθούν οι εργασίες καταγραφής απαιτήσεων και σχεδιασμού, τόσο βελτιώνεται η ποιότητα. Είναι πολύ πιο εύκολο να εντοπισθούν και να διορθωθούν πιθανές αποκλίσεις στο στάδιο σχεδίασης απ' ότι σε αυτό της δοκιμής, καθώς τότε όλα τα συστατικά λογισμικού έχουν ήδη ενοποιηθεί και η ανίχνευση συγκεκριμένων σφαλμάτων είναι ιδιαίτερα περίπλοκη.

Λόγω της ακολουθιακής φύσης των διακριτών εκτελούμενων φάσεων, η δυνατότητα επέκτασης ενός συστήματος καθίσταται δύσκολη, καθώς απαιτείται αυξημένο κόστος και χρόνος οπισθοχώρησης.

Τέλος σημειώνεται ότι, όταν το μοντέλο καταρράκτη εφαρμόζεται ορθά, το έργο ολοκληρώνεται πιο σύντομα, προκύπτει ένα άρτιο προϊόν και σημειώνεται σημαντικό κέρδος στο συνολικό κόστος και χρόνο ανάπτυξης.

Όσον αφορά τα υπόλοιπα αναφερόμενα στο Κεφ. 2 μοντέλα κύκλου ζωής λογισμικού, δεν επιλέχθηκαν διότι το μέγεθος του έργου ήταν μικρό και επρόκειτο να υλοποιηθεί από ένα άτομο, οπότε διάφορα άλλα χαρακτηριστικά όπως π.χ. η παράλληλη εργασία ατόμων / ομάδων σε επί μέρους τμήματα του έργου, η επαναλαμβανόμενη παραγωγή πρωτοτύπων, η διαχείριση κινδύνων, η εκτέλεση ελέγχων σε κάθε φάση, οι συχνές οπισθοδρομήσεις κλπ, χαρακτηριστικά άλλων μοντέλων που εξυπηρετούν καλύτερα την ανάπτυξη πιο περίπλοκων έργων, θα προκαλούσαν πρόσθετες καθυστερήσεις και θα καθιστούσαν τη συνολική διαδικασία πιο χρονοβόρα, κάτι που δεν ήταν επιθυμητό.

Αξιολογώντας όλα τα ανωτέρω, καθώς επίσης και το γεγονός ότι το σύνολο των απαιτήσεων / προδιαγραφών ήταν εξαρχής γνωστές, αποφασίστηκε να υιοθετηθεί το μοντέλο του καταρράκτη για την ανάπτυξη και ολοκλήρωση του έργου «Δημιουργία Ιστοσελίδας για τη προβολή μιας συγκεκριμένης εταιρείας-επιχείρησης».

ΠΑΡΑΡΤΗΜΑ

Έντυπο Καταγραφής Απαιτήσεων

για την Κατασκευή Ιστοσελίδας του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών

Πίνακας περιεχομένων

ΚΕΦΑΛΑΙΟ 1 ..... **206**

1 ΙΣΤΟΣΕΛΙΔΑ ΤΟΥ ΠΑΡΑΡΤΗΜΑΤΟΣ ΑΜΑΛΙΑΔΟΣ ΤΟΥ Τ.Ε.Ι. ΠΑΤΡΩΝ – ΣΥΝΟΠΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΚΑΙ ΔΙΑΡΘΡΩΣΗ **206**

1.1 ΓΕΝΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ **206**
1.2 ΔΙΕΠΑΦΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΜΕ ΤΟΥΣ ΧΡΗΣΤΕΣ **207**
1.3 ΧΡΗΣΤΕΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ **207**
1.4 ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΥΨΗΛΟΥ ΕΠΙΠΕΔΟΥ ΑΠΑΙΤΗΣΕΩΝ **208**
1.4.1 Μεθοδολογία καταγραφής απαιτήσεων ***208***

2 ΑΝΑΛΥΤΙΚΗ ΚΑΤΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ ΣΥΣΤΗΜΑΤΟΣ **209**

2.1 ΑΠΑΙΤΗΣΕΙΣ **209**
2.1.1 Γενικές απαιτήσεις ***209***
2.1.2 Εξειδίκευση απαιτήσεων ***211***

3 ΔΙΑΓΡΑΜΜΑΤΑ UML ..... ΣΦΑΛΜΑ! ΔΕΝ ΕΧΕΙ ΟΡΙΣΤΕΙ ΣΕΛΙΔΟΔΕΙΚΤΗΣ.

3.1 ΔΙΑΓΡΑΜΜΑΤΑ UML ΣΥΣΤΗΜΑΤΟΣ **215**

Περίληψη

Το συγκεκριμένο έγγραφο έχει σκοπό την παρουσίαση των απαιτήσεων για την ανάπτυξη της ηλεκτρονικής τοποθεσίας του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών, όπως αυτές καταγράφηκαν μέσω συνεντεύξεων με τους εμπλεκόμενους υπεύθυνους (διδακτικό προσωπικό, διοικητικό προσωπικό, φοιτητές).

1ο Κεφάλαιο: Περιγράφεται η γενική αρχιτεκτονική του συστήματος καθώς και οι χρήστες αυτού, οι ρόλοι των οποίων αναλύονται.

2ο Κεφάλαιο: Παρουσιάζονται αναλυτικά οι απαιτήσεις του συστήματος. Η καταγραφή παρουσιάζεται με την μορφή λειτουργικών και μη λειτουργικών απαιτήσεων για κάθε επιμέρους υποσύστημα που πρέπει να υλοποιηθεί, σε υψηλό επίπεδο, ώστε να δίδεται η γενική εικόνα της λειτουργικότητας του συστήματος. Οι χαμηλού επιπέδου λεπτομέρειες των απαιτήσεων θα καταγραφούν κατά την διαδικασία επαναξιολόγησης των απαιτήσεων κατά την υλοποίηση.

3ο Κεφάλαιο: Στην ενότητα αυτή παρουσιάζονται με μορφή διαγραμμάτων use case οι απαιτήσεις λειτουργικότητας του συστήματος. Τα διαγράμματα δείχνουν τις επιμέρους λειτουργίες που πρέπει να υλοποιεί το σύστημα, καθώς και τις σχέσεις εξάρτησης και επέκτασης των λειτουργιών για την επιτέλεση του σκοπού τους, αλλά και τις αλληλεπιδράσεις με τους χρήστες του συστήματος.

Το παρόν έγγραφο έχει τη μορφή μιας πρώτης καταγραφής απαιτήσεων, οι οποίες επικυρώθηκαν στο μέτρο του δυνατού από τους υπεύθυνους. Οι απαιτήσεις αυτές θα αναπτυχθούν λεπτομερέστερα καθώς οι λειτουργίες θα υλοποιούνται και θα αξιολογούνται στις επόμενες φάσεις του έργου, οδηγώντας σε μια σταδιακή ανανέωση, εμπλουτισμό ή τροποποίηση των αρχικών απαιτήσεων.

ΚΕΦΑΛΑΙΟ 1

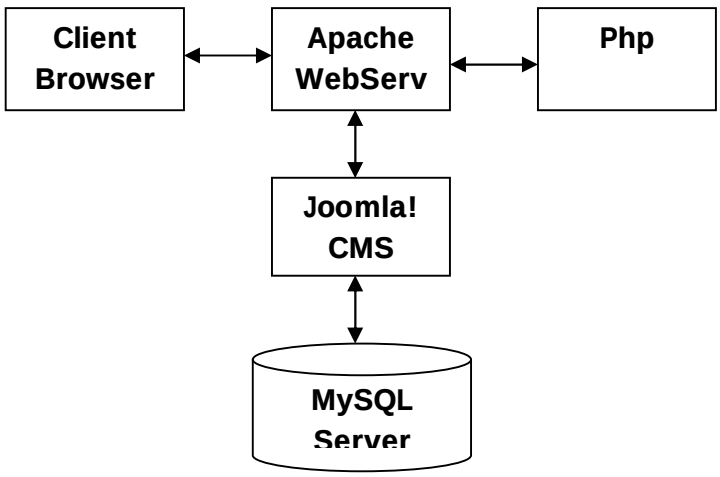
Ιστοσελίδα του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών – Συνοπτική περιγραφή και διάρθρωση

Η ενότητα αυτή περιέχει τις καταγεγραμμένες απαιτήσεις (λειτουργικές) της ηλεκτρονικής τοποθεσίας. Οι απαιτήσεις προέρχονται κατόπιν α)συνεντεύξεων με τους υπεύθυνους (διδακτικό προσωπικό, διοικητικό προσωπικό, φοιτητές) και β)κριτικής ανάλυσης και επεξεργασίας των συγκεντρωμένων στοιχείων.

Γενική αρχιτεκτονική του συστήματος

Η ηλεκτρονική τοποθεσία που θα αναπτυχθεί είναι ένα διαδικτυακό πληροφοριακό σύστημα με αρχιτεκτονική Πελάτη-Εξυπηρετητή. Διαμορφώνεται από το δημοφιλές Open Source Content Management System (Σύστημα Διαχείρισης Περιεχομένου Ανοικτού Κώδικα) “Joomla!”, το οποίο φιλοξενείται σε Apache Web Server εξοπλισμένο με PHP και χρησιμοποιεί μια database ενός MySQL Server για την αποθήκευση του περιεχομένου της ηλεκτρονικής τοποθεσίας.

Ο Apache web server είναι υπεύθυνος για την παραλαβή των αιτημάτων των client browsers (Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Opera, κλπ) και την εξυπηρέτησή τους, εκτελώντας τον κατάλληλο Server Side PHP κώδικα του Joomla! CMS, προσπελαύνοντας κατ’ ανάγκη τη MySQL database, παράγοντας τελικά τον κατάλληλο HTML κώδικα και επιστρέφοντάς τον πίσω στον client browser για την απεικόνιση της ιστοσελίδας.



Εικόνα 1. Γενική Αρχιτεκτονική συστήματος

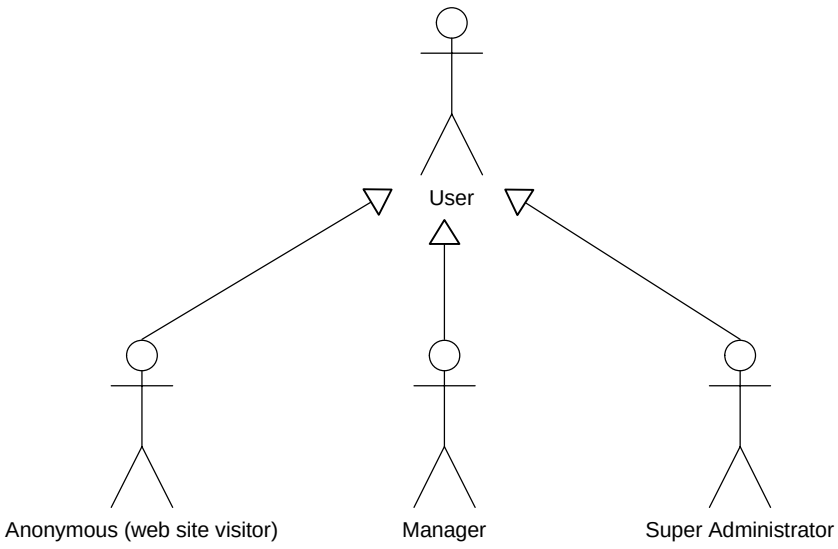
Διεπαφή του συστήματος με τους χρήστες

Το σύστημα υλοποιεί τη διεπαφή του με τους χρήστες (πιστοποιημένους ή ανώνυμους) μέσω κατάλληλων ιστοσελίδων, που επιτρέπουν την προβολή και διαχείριση του περιεχομένου.

Χρήστες του Συστήματος

Το επιλεγμένο CMS υποστηρίζει πολλαπλούς ρόλους χρηστών, με αντίστοιχα δικαιώματα πρόσβασης και διαχείρισης περιεχομένου (Public, Registered, Author, Editor, Publisher, Manager, Administrator και Super Administrator). Το υπό ανάπτυξη σύστημα εκτός από τους ανώνυμους χρήστες (επισκέπτες ιστοτόπου) απαιτεί τη δημιουργία χρηστών δύο εκ των διαθέσιμων τύπων του εν λόγω CMS, έναν για την ανάρτηση περιεχομένου σχετικού με τις διάφορες ανακοινώσεις, εκδηλώσεις και τελευταία νέα του ιδρύματος (ρόλου Manager) και έναν για τη συνολική διαχείριση & συντήρηση του ιστοτόπου (ρόλου Super Administrator).

Στο ακόλουθο σχήμα απεικονίζονται οι χρήστες του συστήματος:



Εικόνα 2. Ιεραρχία χρηστών

Συνοπτική παρουσίαση υψηλού επιπέδου απαιτήσεων

Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών παρουσιάζει μεγάλο πλήθος λειτουργικών και μη-λειτουργικών απαιτήσεων, οι οποίες δεν είναι δυνατό να καταγραφούν και να παρουσιαστούν με πλήρη λεπτομέρεια στο παρόν παραδοτέο, καθώς πολλές από αυτές τις απαιτήσεις θα ανακύψουν ή θα γίνουν εμφανείς κατά τη διαδικασία ανάπτυξης του συστήματος. Η παρούσα ενότητα συνοψίζει τη λειτουργικότητα του συστήματος υπό την μορφή απαιτήσεων, ενώ η ενότητα 2 αναπτύσσει τις καταγεγραμμένες απαιτήσεις περιγράφοντας σε υψηλό επίπεδο τη σκοπιμότητά τους και την πρόσβαση σε αυτές για τις διάφορες κατηγορίες χρηστών (high level requirements).

1.1.1 Μεθοδολογία καταγραφής απαιτήσεων

Οι απαιτήσεις που παρουσιάζονται στο παραδοτέο αυτό προέκυψαν κατόπιν συστηματικής μελέτης των προδιαγραφών του έργου ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών. Με βάση αυτά τα κείμενα και τη συμβολή των υπεύθυνων κατά τη διεξαγωγή συναντήσεων εργασίας (focus group meetings), συντάχθηκε προκαταρκτική λίστα απαιτήσεων. Εν συνεχεία, η λίστα αυτή τροποποιήθηκε και επικυρώθηκε με κατ’ ιδίαν συνεντεύξεις με τους υπεύθυνους.

Απαιτήσεις Υψηλού Επιπέδου

Το πληροφοριακό σύστημα ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να υποστηρίζει διαφορετικές κατηγορίες χρηστών με κατάλληλες μεθόδους πιστοποίησης. Με βάση την ταυτοποίησή του από το σύστημα, κάθε χρήστης λαμβάνει διαφορετικά δικαιώματα πρόσβασης και διαχείρισης του περιεχομένου.

Το πληροφοριακό σύστημα ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να προσφέρει στους πιστοποιημένους χρήστες του φιλικό περιβάλλον διαχείρισης περιεχομένου, προαπαιτώντας για τη διαχείρισή του βασικές μόνο γνώσεις επεξεργασίας κειμένου.

Το πληροφοριακό σύστημα ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να καταγράφει και να διατηρεί κατάλληλα στατιστικά και άλλα στοιχεία χρήσης ώστε να καταγράφονται οι επωφελούμενοι από το σύστημα και να μπορούν να συντάσσονται κατάλληλες αναφορές με βάση τα στοιχεία αυτά.

ΚΕΦΑΛΑΙΟ 2

Αναλυτική καταγραφή απαιτήσεων συστήματος

Στην ενότητα αυτή παρουσιάζεται μια αναλυτικότερη καταγραφή των υψηλού επιπέδου απαιτήσεων.

Απαιτήσεις

2.1.1 Γενικές απαιτήσεις

ΑΑ	**-**	**Τομέας**	Γενικές απαιτήσεις
Περιγραφή			
1. Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα βασίζεται σε σύστημα διαχείρισης περιεχομένου (Content Management System) για την αποθήκευση στατικού περιεχομένου και τη διάχυση πληροφορίας. 2. Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να περιέχει τις εξής κύριες ενότητες περιεχομένου (δεν αναφέρονται με σειρά παρουσίασης): 2.1. Νέα / Ανακοινώσεις 2.2. Εκδηλώσεις 2.3. Πρόσβαση (χάρτης) 2.4. Επικοινωνία 2.5. Το Τμήμα (Εφαρμογών Πληροφορικής στη Διοίκηση και στην Οικονομία) 2.6. Σπουδές 2.7. Προσωπικό 2.8. Γραμματεία 2.9. Σπουδαστική Μέριμνα 2.10. Πτυχιακή Εργασία 2.11. Πρακτική Άσκηση 2.12. Erasmus 2.13. Ημερολόγιο 2.14. Τελευταία Νέα 2.15. Ώρα 2.16. Πλαίσιο Αναζήτησης 2.17. Οδηγός Σπουδών 2.18. Ψηφοφορία			

2.19. Επισκεψιμότητα

2.20. Ο Καιρός

2.21. Εορτολόγιο

2.22. Χρήσιμες Διευθύνσεις

2.23. Login

2.1.2 Εξειδίκευση απαιτήσεων

ΑΑ	**1**	**Τομέας**	Αρχιτεκτονική Συστήματος
Περιγραφή			
1. Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα βασίζεται σε σύστημα διαχείρισης περιεχομένου (CMS). 1.1. Για την υλοποίηση προκρίνεται σκόπιμη η χρήση του συστήματος Joomla!". 1.2. Το σύστημα θα συνοδεύεται από κατάλληλη βάση δεδομένων σε MySQL. 1.3. Στο σύστημα αποθηκεύονται τα στατικά μέρη των ιστοτόπων. **1.4.** Το σύστημα είναι υπεύθυνο για την τήρηση στοιχείων λογαριασμού των χρηστών (κωδικοί και ονόματα χρήστη).			
Απαίτηση Πρόσβασης: Όλοι οι Χρήστες (Ρόλος Χρηστών User).			
Απαίτηση Ενημέρωσης Περιεχόμενου: Manager, Super Administrator.			

ΑΑ	4	Τομέας	Περιεχόμενο ιστοτόπου

Περιγραφή

2. Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να περιέχει την εξής ενότητες περιεχομένου:
   2.13. Ημερολόγιο
      2.13.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε ημερολόγιο του τρέχοντος μήνα και έτους.
   2.15. Ώρα
      2.15.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε ένδειξη της τρέχουσας ώρας.
   2.16. Πλαίσιο Αναζήτησης
      2.16.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε πλαίσιο αναζήτησης πληροφοριών του περιεχομένου της ιστοσελίδας.
   2.19. Επισκεψιμότητα
      2.19.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε πλαίσιο εμφάνισης στατιστικών μετρήσεων επισκεψιμότητας της ιστοσελίδας.
   2.20. Ο Καιρός
      2.20.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε πλαίσιο προβολής πληροφοριών των τρεχόντων καιρικών συνθηκών της Αμαλιάδος.
   2.21. Εορτολόγιο
      2.21.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε πλαίσιο προβολής εορταστικού ημερολογίου.

Απαίτηση Πρόσβασης: Όλοι οι Χρήστες (Ρόλος Χρηστών User).

2. Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να περιέχει την εξής ενότητες περιεχομένου:
   2.1. Νέα / Ανακοινώσεις
      2.1.1. Εμφανίζεται ως λίστα με πρόσφατα νέα και ανακοινώσεις του ιδρύματος.
      2.1.2. Ο Manager θα πρέπει να έχει πρόσβαση σε κατάλληλη φόρμα για την ανάρτηση νέων ανακοινώσεων.
      2.1.3. Ο Manager θα πρέπει να διαθέτει μέθοδο τροποποίησης αναρτηθέντων ανακοινώσεων ή διαγραφής αυτών.
   2.2. Εκδηλώσεις

2.2.1. Εμφανίζεται ως λίστα με πρόσφατες εκδηλώσεις του ιδρύματος.

2.2.2. Ο Manager θα πρέπει να έχει πρόσβαση σε κατάλληλη φόρμα για την ανάρτηση νέων εκδηλώσεων.

2.2.3. Ο Manager θα πρέπει να διαθέτει μέθοδο τροποποίησης αναρτηθέντων εκδηλώσεων ή διαγραφής αυτών.

2.14. Τελευταία Νέα

2.14.1. Εμφανίζεται ως λίστα με τα τελευταία νέα του ιδρύματος.

2.14.2. Ο Manager θα πρέπει να έχει πρόσβαση σε κατάλληλη φόρμα για την ανάρτηση των τελευταίων νέων.

2.14.3. Ο Manager θα πρέπει να διαθέτει μέθοδο τροποποίησης αναρτηθέντων τελευταίων νέων ή διαγραφής αυτών.

Απαίτηση Πρόσβασης: Όλοι οι Χρήστες (Ρόλος Χρηστών User).

Απαίτηση Ενημέρωσης Περιεχόμενου: Manager, Super Administrator.

2. Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να περιέχει την εξής ενότητες περιεχομένου:

2.3. Πρόσβαση (χάρτης)

2.3.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες πρόσβασης στο ίδρυμα.

2.4. Επικοινωνία

2.4.1. Ο χρήστης έχει πρόσβαση σε στοιχεία και φόρμα επικοινωνίας του ιδρύματος.

2.5. Το Τμήμα (Εφαρμογών Πληροφορικής στη Διοίκηση και στην Οικονομία)

2.5.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες του τμήματος Εφαρμογών Πληροφορικής στη Διοίκηση και στην Οικονομία του ιδρύματος.

2.6. Σπουδές

2.6.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες προσφερομένων προγραμματων σπουδών του ιδρύματος.

2.7. Προσωπικό

2.7.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες σχετικά με το διδακτικό και διοικητικό προσωπικό του ιδρύματος.

2.8. Γραμματεία

2.8.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες της Γραμματείας του ιδρύματος.

2.9. Σπουδαστική Μέριμνα

2.9.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες σπουδαστικής μέριμνας του ιδρύματος.

2.10. Πτυχιακή Εργασία

2.10.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες σχετικά με την πτυχιακή εργασία, απαραίτητη για απόκτηση πτυχίου.

2.11. Πρακτική Άσκηση

2.11.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες σχετικά με την πρακτική άσκηση φοιτητών του ιδρύματος.

2.12. Erasmus

2.12.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες σχετικά με το πρόγραμμα Erasmus.

2.17. Οδηγός Σπουδών

2.17.1. Ο χρήστης έχει πρόσβαση σε πληροφορίες σχετικά με τον οδηγό σπουδών του ιδρύματος.

2.18. Ψηφοφορία

2.18.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε πληροφορίες της τρέχουσας ψηφοφορίας της ιστοσελίδας.

2.22. Χρήσιμες Διευθύνσεις

2.22.1. Ο χρήστης θα πρέπει να έχει πρόσβαση σε χρήσιμες διευθύνσεις ιστοσελίδων.

Απαίτηση Πρόσβασης: Όλοι οι Χρήστες (Ρόλος Χρηστών User).

Απαίτηση Ενημέρωσης Περιεχόμενου: Super Administrator

2. Η ηλεκτρονική τοποθεσία του Παραρτήματος Αμαλιάδος του Τ.Ε.Ι. Πατρών θα πρέπει να περιέχει την εξής ενότητες περιεχομένου:

2.23. Login

2.23.1. Ο χρήστης θα πρέπει να διαθέτει σημείο καταχώρησης των διαπιστευτηρίων του για είσοδο στο σύστημα.

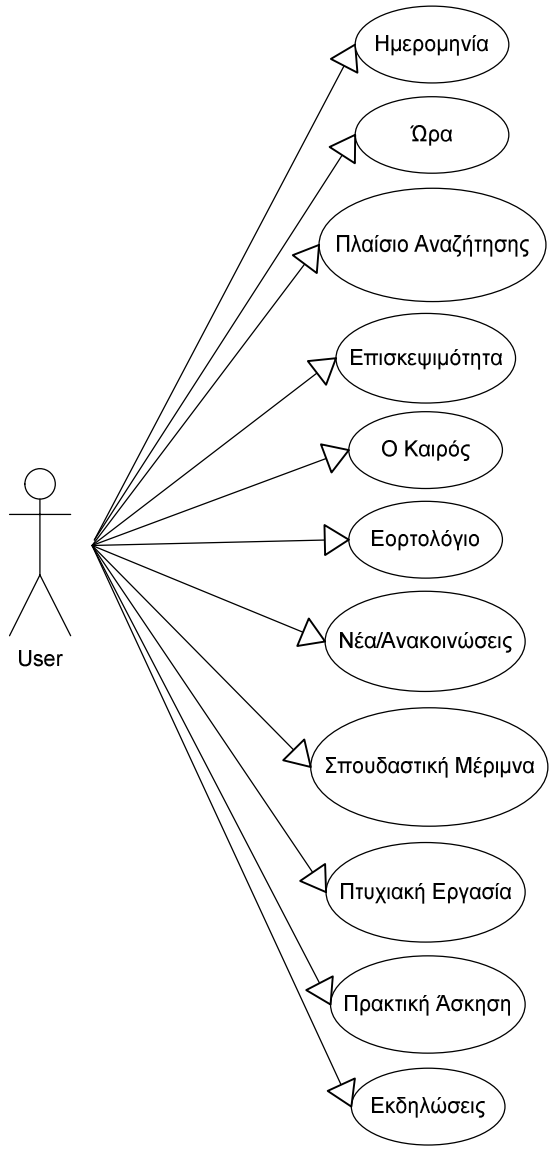
Απαίτηση Πρόσβασης: Manager, Super Administrator

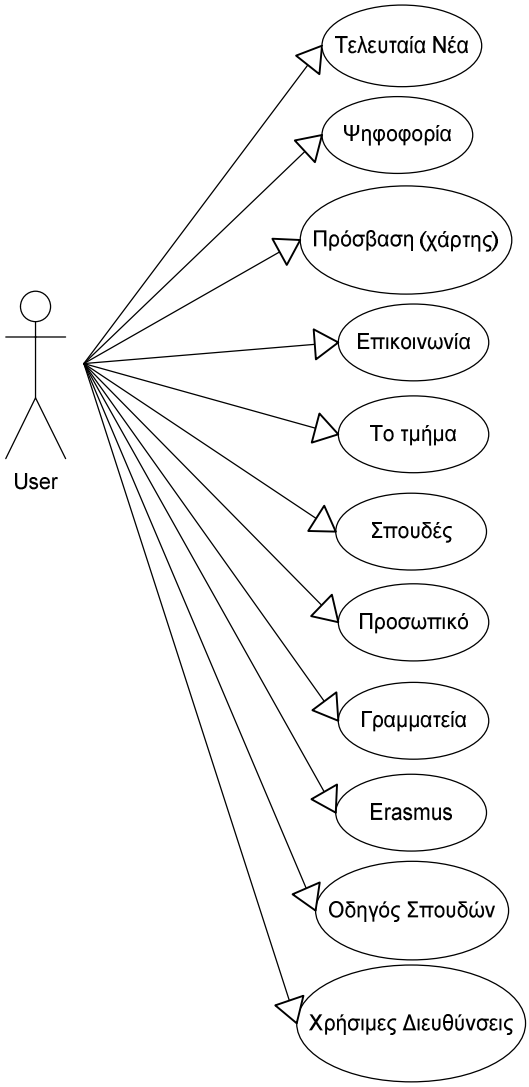
ΚΕΦΑΛΑΙΟ 3

Διαγράμματα UML Συστήματος

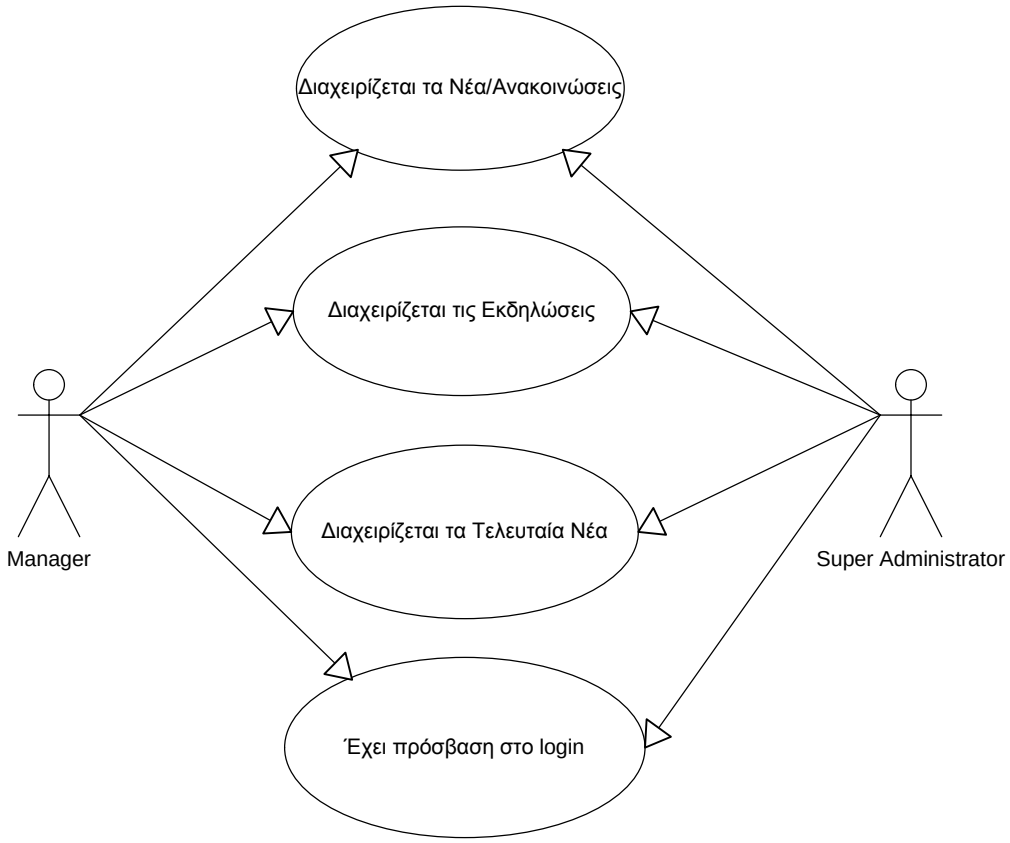
Στις παρακάτω δύο εικόνες απεικονίζεται ο χρήστης και οι οντότητες όπου μπορεί να έχει πρόσβαση.

Εικόνα 3.1: Use Case Diagram





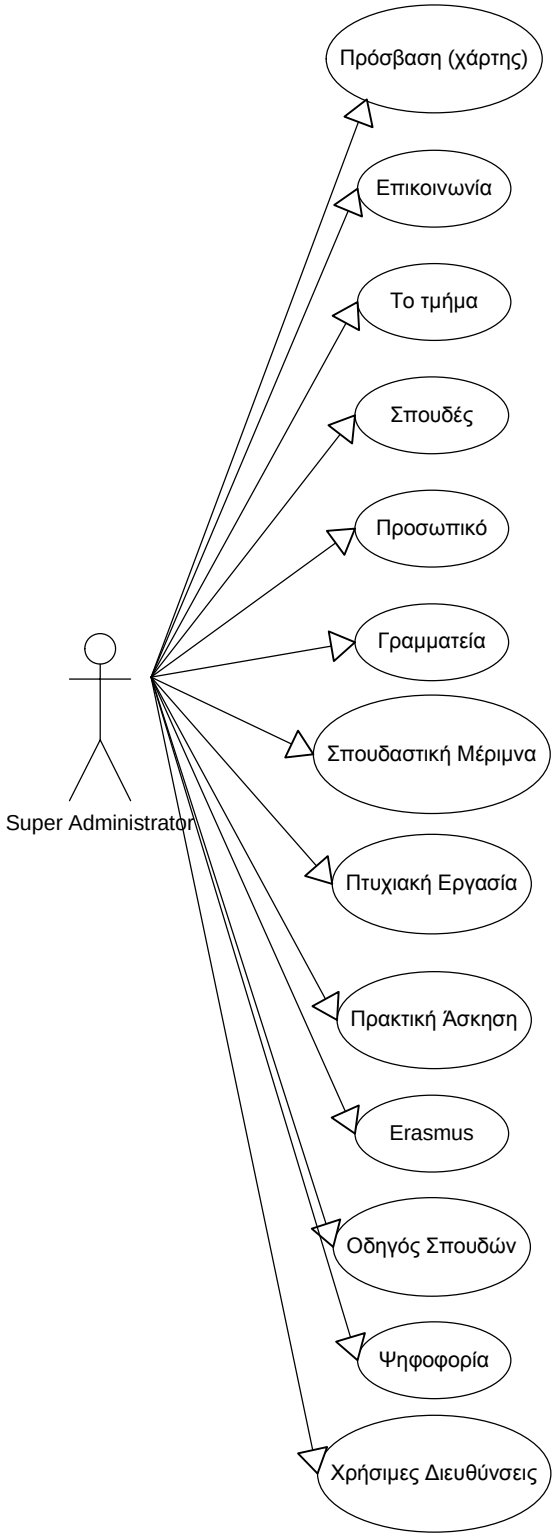
Εικόνα 3.2: Use Case Diagram



Εικόνα 3.3:Use Case Diagram

Στην παραπάνω εικόνα απεικονίζονται οι οντότητες οι οποίες μπορούν να διαχειρίζονται και να έχουν πρόσβαση ο Manager και ο Super Administrator.

Σε αυτό το σχήμα απεικονίζονται οι οντότητες όπου διαχειρίζεται μόνο ο Super Administrator.



Εικόνα 3.4:Use Case Diagram

Βιβλιογραφία

1. Banker, R. D., Datar, S. M., *κ.α.* (1993). Software complexity and maintenance costs. *Comm*. ACM, **36**(11), 81-94. (Κεφ. 21, 26)

2. Beck, K. (2000). *Extreme Programming Explained*. Boston: Addison-Wesley. (Κεφ. 4, 17, 25, 26)

3. Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, **21**(5), 61-72. (Κεφ. 4, 5)

4. Boehm, B. W. (1979). Software engineering; R & D trends and defense needs. *In Research Directions in Software Technology* (P. Wegner, ed.). Cambridge, MA:MIT Press, 1-9. (Κεφ. 22)

5. Boehm, B. W., Abts, C., *κ.ά.* (2000). *Software Cost Estimation with COCOMO II*. Uper Saddle River, NJ: Prentice Hall. (Κεφ. 19, 22, 26)

6. Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. Redwood City, CA: Benjamin-Cummimgs. (Κεφ. 1, 4, 8, 14)

7. Booch, G. Rumbaugh, J., κ.ά. (1999). *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley. (Κεφ. 1, 4, 8)

8. Chikofsky, E. J. and Cross, J. H. (1990). Reverse engineering and desing recovery: a taxonomy. *IEEE Software,***7**(1), 13-17. (Κεφ. 21)

9. Davis, A. M. (1993). *Software Requirements: Objects, Functions, & States*. Englewood Cliffs, NJ: Prentice Hall. (Κεφ. 6)

10. DeMarco, T. (1978). *Structured Analysis and Sysrem Specification.* New York: Yourdon Press. (Κεφ. 8)

11. Erlikh, L. (2000). Leveraging legacy system dollars for e-business. *IT Pro*, **May/June 2000**, 17-23. (Κεφ. 21)

12. Guimaraes, T. (1983). Managing application program maintenance expenditures. *Comm. ACM,* **26**(10), 739-46. (Κεφ. 21)

13. Heninger, K. L. (1980). Specifying software requirements for complex systems: new techniques and their applications. *IEEE Trans. on Software Engineering,* **SE-6**(1), 2-13.(Κεφ. 6)

14. Jackson, M. A.(1983). *System Development*. London: Prentice Hall. (Κεφ. 1, 8)

15. Jacobsen, I., Christenson, M., κ,ά. (1993). *Object-Oriented Software Engineering.* Wokingham: Addison-Wesley. (Κεφ. 6, 8, 15)

16. Kafura, D. and Reddy, G. R. (1987). The use of software complexity metrics in software maintenance. *IEEE Trans. on Software Engineering,* **SE-13**(3), 335-43. (Κεφ. 21)

17. Lehman M.M. (1996). Laws of software evolution revisited. *Proc. European Workshop on Software Process Technology (EWSPT'96),* Nancy, France: SpringerVerlag. (Κεφ. 21)

18. Lehman, M. M. and Belady, L. (1985). *Program Evolution: Processes of Software Change.* London: Academic Press. (Κεφ. 21)

19. Lehman, M. M., Perry, D. E., *κ.ά.* (1998). On evidence supporting the FEAST hypothesis and the laws of software evolution. *Proc. Metrics'98,* Bethesda, MD: IEEE Computer Society Press. (Κεφ. 21)

20. Lehman, M. M., Ramil, J. F., κ.ά. (2001). An approach to modeling long-term growth trends in software systems. *Proc. Int. Conf. on Software Maintenance,* Florence, Italy: IEEE Computer Society Press. (Κεφ. 21)

21. Lientz, B. P. and Swanson, E. B. (1980). *Software Maintenance Management.* Reading, MA: Addison-Wesley. (Κεφ. 21)

22. Linger, R. C. (1994). Cleanroom process model. *IEEE Software,* **11**(2), 50-8. (Κεφ. 4, 22)

23. Mills, H. D., Dyer, M., *κ.ά.* (1987). Cleanroom software engineering. *IEEE Software,* **4**(5), 19-25. (Κεφ. 3, 4, 22)

24. North, B. M. (2009). Οδηγός του joomla! Δημιουργία Επιτυχημένων Websites με joomla! Στο B. M. North, *Οδηγός του joomla! Δημιουργία Επιτυχημένων Websites με joomla!* Εκδόσεις:Παπασωτηρίου.

25. Nosek, J. T. and Palvia, P. (1990). Software maintenancemanagement:changes in the last decade. *Software Maintenance: Research and Practice,* **2**(3), 157-74.(Κεφ. 21)

26. Prowell, S. J., Trammell, C. J., *κ.ά.* (1999). *Cleanroom Software Engineering:Technology and Process.* Reading, MA: Addison-Wesley. (Κεφ. 4, 10, 22, 24)

27. Royce, W. W. (1970). Managing the development of large software systems: concepts and techniques. *Proc. IEEE WESTCON,* Los Angeles CA:IEEE Computer Society Press. (Κεφ. 4)

28. Rumbaugh, J., Blaha, M., *κ.ά.* (1991).*Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall. (Κεφ. 1, 4, 8)

29. Rumbaugh, J., Jacobson, I., *κ.ά.* (1999). *The Unified Modeling Language Reference Manual.* Reading, MA: Addison-Wesley. (Κεφ. 1, 4, 8, 14)

30. Rumbaugh, J., Jacobson, I., *κ.ά.* (1999). *The Unified Software Development Process.* Reading, MA: Addison-Wsley. (Κεφ. 1, 4, 8)

31. Selby, R. W., Basili, V. R., *κ.ά.* (1987). Cleanroom software development: an empirical evaluation. *IEEE Trans. on Software Engineering,* **SE-13**(9), 1027-37. (Κεφ. 4, 22)

32. Sommerville, I. (2009). Βασικές Αρχές Τεχνολογίας Λογισμικού. Στο I. Sommerville, *Βασικές Αρχές Τεχνολογίας Λογισμικού*. Κλειδάριθμος.

33. Ulrich, W. M. (1990).The evolutionary growth of software reengineering and the decade ahead.*American Programmer,* **3**(10), 14-20. (Κεφ. 21)

34. Wordsworth, J. (1996). *Software Engineering with B.* Wokingham:Addison-Wesley. (Κεφ. 4, 9, 10, 22)

35. Βασίλειος Βεσκούκης. (2000). Τεχνολογία Λογισμικού Ι. Στο Β. Βεσκούκης, *Τεχνολογία Λογισμικού Ι* (σ. 246). Πάτρα: Ελληνικό Ανοιχτό Πανεπιστήμιο.

36. Μανώλης Μαρκατσέλας - Ξαρχάκος Κώστας (2010) Εκδόσεις: Ξαρχάκος Κώστας.

Πηγές

A. **www.joomla.org (last visited on 11/04/2012)**

B. **www.joomla.gr (last visited on 11/04/2012)**

C. **http://www.itswtech.org/Lec/Rand%28SoftwareEng%29/Software%20Engineering%20lec%20%20%20%201.pdf (last visited on 11/04/2012)**

D. **http://docs.joomla.org/Administrators (last visited on 11/04/2012)**