ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη Εφαρμογών Διαχείρισης Διαδικασιών σε Περιβάλλον Java Spring

Αναστάσιος Γ. Σταθόπουλος
Α.Μ.: 3220

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων: Καθηγητής Χρήστος Ζαρολιάγκης
Συνεπιβλέπων: Δρ. Ιωάννης Χατζηγιαννάκης

Πάτρα
Ιούλιος 2009

Στην οικογένια μου και στην Αγγελική, για τη στήριξη και την υπομονή τους.

Περίληψη

Η παρούσα διπλωματική ασχολείται με την μελέτη μεθόδων και εργαλείων ανάπτυξης λογισμικού. Παρουσιάζονται μοντέλα διεργασιών λογισμικού, στα οποία γίνεται μοντελοποίηση των διαδικασιών που λαμβάνουν μέρος στην ανάπτυξη λογισμικού καθώς επίσης και μεθοδολογίες ταχείας ανάπτυξης λογισμικού, ο οποίες εισάγουν τεχνικές για τη συγγραφή αποδοτικού κώδικα γρήγορα και όσο το δυνατόν πιο κοντά στις απαιτήσεις του πελάτη. Επιπλέον, παρουσιάζονται σύγχρονες τεχνολογίες και εργαλεία ανάπτυξης λογισμικού, τα οποία χρησιμοποιούνται για την ανάπτυξη εφαρμογών τόσο κατανεμημένων όσο και κεντρικοποιημένων.

Η θεωρία που μελετάται γίνεται πράξη, όπου στα πλαίσια της διπλωματικής αναπτύσσεται ένα σύστημα διαχείρισης ερευνητικών έργων, το οποίο καλείται να εκπληρώσει τις προδιαγραφές που θέτει το *Ερευνητικό Ακαδημαϊκό Ινστιτούτο Τεχνολογίας Υπολογιστών* (Ε.Α.Ι.Τ.Υ.). Στην ανάπτυξη αυτού του συστήματος γίνεται χρήση των περισσότερων από τις μεθοδολογίες και τεχνολογίες που παρουσιάζονται στο θεωρητικό μέρος της διπλωματικής.

Η διπλωματική εργασία συνοδεύεται από ένα CD το οποίο περιέχει, τον *κώδικα της εφαρμογής*, την *τεκμηρίωση του κώδικα* που δημιουργήθηκε με τη χρήση του εργαλείου *javadoc*, καθώς και τον *παρόν κείμενο* σε ηλεκτρονική μορφή.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Καθηγητή του τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής, και επιβλέποντα καθηγητή μου, κ. Χρήστο Ζαρολιάγκη που μου έδωσε την ευκαιρία να ασχοληθώ με το αντικείμενο της παρούσας διπλωματικής και με τίμησε με την συνεργασία του.

Θερμές ευχαριστίες, επίσης απευθύνονται και στον Δρ. Ιωάννη Χατζηγιαννάκη, συνεπιβλέποντα της διπλωματικής μου, για την πολύτιμη βοήθεια και καθοδήγηση που μου προσέφερε καθ' όλη τη διάρκεια εκπόνησης της διπλωματικής εργασίας μου μέχρι και την ολοκλήρωσή της.

Επιπλέον, κατα τη διάρκεια της ανάπτυξης της εφαρμογής, είχα την χαρά να συνεργαστώ με την κ. Έφη Χήτα την οποία και ευχαριστώ πολύ για το χρόνο που μου αφιέρωσε.

Αναστάσιος Γ. Σταθόπουλος

Πάτρα, 2009.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Μέρος Ι

Εισαγωγή

ΚΕΦΑΛΑΙΟ **1**

Εισαγωγή

1.1 Κίνητρο και σημασία του θέματος

Η χρήση της πληροφορικής γίνεται όλο και πιο διαδεδομένη με τη πάροδο των χρόνων. Επιχειρήσεις και οργανισμοί στρέφονται σε λύσεις που προσφέρει η πληροφορική για την απλοποίηση των διαδικασιών τους αλλά και τη βελτίωση παροχής των υπηρεσιών τους. Επίσης, η αυξανόμενη χρήση του διαδικτύου έδωσε ώθηση για την ανάπτυξη όλο και πιο πολύπλοκων εφαρμογών οι οποίες απλοποιούσαν πολλές, μέχρι πρότινος, γραφειοκρατικές διαδικασίες, όπως η τραπεζικές συναλλαγές, η κατάθεση φορολογικής δήλωσης. Κάθε διαδικασία που μπορεί να πραγματοποιηθεί με τη χρήση της τεχνολογίας της πληροφορικής, αποτελεί και ένα διαφορετικό πρόβλημα το οποίο καλούνται να επιλύσουν οι μηχανικοί λογισμικού.

Σήμερα, σε όλα τα ανεπτυγμένα κράτη, εθνικοί οργανισμοί και κοινωφελείς επιχειρήσεις βασίζονται σε συστήματα υπολογιστών. Επιπλέον, τα περισσότερα ηλεκτρικά/ηλεκτρονικά προϊόντα περιλαμβάνουν κάποιου είδους υπολογιστή για τον έλεγχο και τη λειτουργία τους. Γι΄ αυτό το λόγο, η παραγωγή και συντήρηση λογισμικού οικονομικά αλλά ταυτόχρονα προσφέροντας τη καλύτερη δυνατή ποιότητα, είναι θεμελιώδης για τη λειτουργία σωστή λειτουργία της οικονομίας κάθε χώρας.

Η ανάπτυξη μιας εφαρμογής, ανάλογα με το είδος της, μπορεί να είναι μια πολύπλοκη διαδικασία κατά την οποία προκύπτουν πολλές προκλήσεις που πρέπει να ξεπεράσει η ομάδα ανάπτυξης. Ο στόχος της ανάπτυξης λογισμικού είναι το λογισμικό να πληροί τις προδιαγραφές που έχει θέσει ο πελάτης και παράλληλα αυτό να γίνει με τον πιο αποδοτικό και οικονομικό τρόπο.

Η ανάγκη μείωση της πολυπλοκότητας κατά την ανάπτυξη εφαρμογών οδήγησε στην μοντελοποίηση των διαδικασιών ανάπτυξης λογισμικού. Κατά καιρούς, έχουν προταθεί πολλά μοντέλα που βελτιώνουν σε σημαντικό βαθμό αυτή τη διαδικασία, κάποια εκ των οποίων, όπως το μοντέλο “καταρράκτη”, χρησιμοποιούνται ευρέος από πολλούς οργανισμούς για το σχεδιασμό της διαδικασίας ανάπτυξης λογισμικού. Η αυξανόμενη πολυπλοκότητα στο λογισμικού που αναπτύσσεται, καθώς και η βελτίωση στο σχεδιασμό και την αποδοτικότητα της ανάπτυξης που προσφέρουν αυτά τα μοντέλα, κάνουν επιτακτική τη μελέτη και υιοθέτησή τους.

Επιπρόσθετα, τα τελευταία χρόνια έχουν παρουσιαστεί μεθοδολογίες ανάπτυξης λογισμικού που χρησιμοποιούνται για την ταχύτερη ολοκλήρωση της διαδικασίας. Οι μεθοδολογίες αυτές, προσπαθούν να απλοποιήσουν την ανάπτυξη του λογισμικού αφαιρώντας περιττές και χρονοβόρες διαδικασίες οι οποίες υπήρχαν στις παλιότερες και λιγότερο ευέλικτες μεθόδους.

Παράλληλα, χρησιμοποιήθηκαν αρκετές γλώσσες προγραμματισμού για την ανάπτυξη εφαρμογών. Η πιο διαδεδομένη γλώσσα είναι η αντικειμενοστραφής γλώσσα Java. Λόγο της φύσης της (αντικειμενοστράφεια) έκανε πιο εύκολη την μοντελοποίηση των προβλημάτων. Επιπλέον, ήταν η πρώτη γλώσσα με την οποία μπορούσε κανείς να γράψει πολύπλοκες εφαρμογές χρησιμοποιώντας απλούστερα διακριτά μέρη. Όμως, αρχικά η Java δεν είχε τα κατάλληλα συστατικά για την ανάπτυξη πολύπλοκων εφαρμογών για επιχειρήσεις. Για αυτό το λόγο η Sun ανακοίνωσε το Μάρτιο του 1998 το πρότυπο Enterprise Java Beans (EJB), το οποίο προσέφερε τα κατάλληλα εργαλεία που χρειάζονταν οι προγραμματιστές. Το πρόβλημα των EJBs ήταν η πολυπλοκότητά τους, και είχαν ως αποτέλεσμα τη δημιουργία "βαριών" εφαρμογών.

Όσο περνούσε ο καιρός και η χρήση της Java γινόταν όλο και πιο διαδεδομένη, αναπτύχθηκαν καινούριες τεχνολογίες και περιβάλλοντα εργασίας, τα οποία προσφέρουν πολλά πλεονεκτήματα όπως ευελιξία, μεταφερσιμότητα, χαλαρό και κατανοητό κώδικα, δημιουργία ελαφριών εφαρμογών. Πλέον, η ανάπτυξη εφαρμογών σε πολλούς οργανισμούς και επιχειρήσεις βασίζεται σε αυτά τα περιβάλλοντα εργασίας και στις τεχνολογίες που παρέχουν, και η εξοικείωση μαζί τους, αποτελεί βασική προϋπόθεση για έναν επιτυχημένο μηχανικό λογισμικού.

1.2 Στόχοι της διπλωματικής εργασίας

Ένας από τους στόχους της παρούσας διπλωματικής ήταν η μελέτη των πιο γνωστών μοντέλων διεργασιών λογισμικού, που χρησιμοποιούνται ευρύτατα από πολλούς οργανισμούς και επιχειρήσεις ανάπτυξης λογισμικού. Επιπλέον, στόχος της διπλωματικής ήταν η μελέτη σύγχρονων μεθοδολογιών ταχείας ανάπτυξης λογισμικού καθώς και εργαλείων που διευκολύνουν την ανάπτυξη κώδικα και τη διαχείρισή του καθ' όλη τη διάρκεια ανάπτυξης μίας εφαρμογής. Ένας ακόμα στόχος, εξίσου σημαντικός με τους προηγούμενους, είναι η γνωριμία, και όσο το δυνατόν καλύτερη εξοικείωση, με σύγχρονες τεχνολογίες ανάπτυξης εφαρμογών που βασίζονται στη γλώσσα προγραμματισμού Java, και κυρίως του περιβάλλοντος εργασίας Spring.

Τέλος, ο σημαντικότερος στόχος ήταν η ανάπτυξη μιας κατανεμημένης εφαρμογής διαχείρισης ερευνητικών έργων. Οι περισσότερες τεχνολογίες που μελετούνται στα πλαίσια της διπλωματικής, χρησιμοποιούνται για την ανάπτυξη αυτής της εφαρμογής η οποία σκοπό έχει να πληροί της απαιτήσεις που θέτει το Ερευνητικό Ακαδημαϊκό Ινστιτούτο Τεχνολογίας Υπολογιστών (Ε.Α.Ι.Τ.Υ.). Η εγκατάσταση και μόνιμη χρήση της από τα μέλη του Ε.Α.Ι.Τ.Υ. ήταν ο υψηλότερος και δυσκολότερος στόχος της διπλωματικής.

1.3 Συνεισφορά της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία παρουσιάζει το σχεδιασμό και την υλοποίηση ενός κατανεμημένου συστήματος διαχείρισης έργων, βασισμένο σε σύγχρονες τεχνολογίες λογισμικού και παγκόσμιου ιστού. Επιπλέον, εξετάζει και αναλύει αρκετές πτυχές της διαδικασίας ανάπτυξης εφαρμογών και γενικά της τεχνολογίας λογισμικού. Εισάγεται η έννοια της *διεργασίας λογισμικού*, παρουσιάζονται τα τρία πιο γνωστά *μουτέλα διεργασίας λογισμικού* και αναλύονται τα πλεονεκτήματα και τα μειονεκτήματα που προκύπτουν απο τη χρήση τους. Επιπλέον, αναλύονται οι τρεις

πιο σημαντικές *μεθοδολογίες ταχείας ανάπτυξης λογισμικού*, τα πλεονεκτήματα και μειονεκτήματά τους, οι συνθήκες ανάπτυξης για τις οποίες είναι προτιμότερο να χρησιμοποιούνται.

Επιπρόσθετα, μελετούνται αναλυτικά διάφορα *εργαλεία ανάπτυξης λογισμικού*, τα οποία χρησιμοποιούνται σε διαφορετικές φάσης της ανάπτυξης. Αναλύοντας τα χαρακτηριστικά τους, μπορεί κανείς να καταλάβει πότε η χρήση κάθε εργαλείου μπορεί να βοηθήσει την ανάπτυξη μιας εφαρμογής και ποιο εργαλείο είναι κατάλληλο για το είδος του λογισμικού που αναπτύσσεται.

Στα πλαίσια της διπλωματικής περιγράφονται τα περιβάλλοντα εργασίας *Spring* και *Hibernate*. Η Hibernate είναι μία τεχνολογία που διευκολύνει σε μία εφαρμογή Java τη διαχείριση των δεδομένων προς αποθήκευση κάνοντας την αντιστοίχηση των αντικειμένων της Java σε μία σχεσιακή βάση δεδομένων. Η Spring είναι μία τεχνολογία που, μεταξύ άλλων, προσδίδει απλότητα στον κώδικα μιας εφαρμογής, αποτελεσματικό έλεγχο και χαλαρή διασύνδεση των αντικειμένων της.

Οι παραπάνω τεχνολογίες, χρησιμοποιούνται για την ανάπτυξη της εφαρμογής διαχείρισης ερευνητικών έργων. Κατά τη διάρκεια της ανάπτυξής της, εξετάστηκε εις βάθος το περιβάλλον εργασίας Spring, όπου χρησιμοποιήθηκαν προχωρημένες τεχνικές και δόθηκε η ευκαιρία να μελετηθεί η χρήση του στην ανάπτυξη μίας πραγματικής εφαρμογής. Επιπλέον, χρησιμοποιήθηκαν τα εργαλεία που αναλύονται στη διπλωματική εργασία, τα οποία διευκόλυναν την διαδικασία ανάπτυξης της εφαρμογής αλλά παράλληλα εξασφάλιζαν την ορθότητα και την αποδοτικότητα του παραγόμενου κώδικα.

Για τους παραπάνω λόγους, πιστεύουμε ότι η διπλωματική εργασία αποτελεί ένα καλό εγχειρίδιο για την ανάπτυξη εφαρμογών, στο οποίο πρώτα παρατίθεται το θεωρητικό υπόβαθρο για την ανάπτυξη εφαρμογών, και στη συνέχεια ακολουθεί η πρακτική εφαρμογή όσων μελετήθηκαν σε θεωρητικό επίπεδο με την ανάπτυξη της εφαρμογής διαχείρισης ερευνητικών έργων.

1.4 Δομή της διπλωματικής εργασίας

Σε αυτή την ενότητα παρουσιάζεται η δομή της παρούσας διπλωματικής εργασίας, η οποία χωρίζεται σε τέσσερις θεματικές ενότητες. Η πρώτη ενότητα περιλαμβάνει την εισαγωγή της διπλωματικής, όπου αναφέρονται η χρησιμότητα της διπλωματικής, το κίνητρο ενασχόλησης με το συγκεκριμένο θέμα καθώς και η σημασία του θέματος, οι στόχοι της διπλωματικής εργασίας και η συνεισφορά της, και τέλος η δομή της. Το δεύτερο μέρος περιλαμβάνει τρία κεφάλαια που έχουν σχέση με την ανάπτυξη εφαρμογών, τη μοντελοποίηση αυτής της διαδικασίας και εργαλεία που μπορούν να χρησιμοποιηθούν. Στο τρίτο μέρος, γίνεται η παρουσίαση τεχνολογιών ανάπτυξης εφαρμογών, και περιέχει δύο κεφάλαια, και αυτές είναι η *Hibernate* και η *Spring*. Στο τέταρτο μέρος ασχολείται με την εφαρμογή που αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας, όπου υπάρχουν τρία κεφάλαια που αφορούν την περιγραφή της εφαρμογής, το σχεδιασμό της και την υλοποίησή της. Στο τελευταίο μέρος καταγράφονται τα συμπεράσματα και ο απολογισμός της συγγραφής της διπλωματικής καθώς επίσης και η βιβλιογραφία που χρησιμοποιήθηκε.

Πιο αναλυτικά, στο **κεφάλαιο 2** ορίζεται η *διεργασία λογισμικού* και πως αυτή διαφοροποιείται ανάλογα με τη φύση της εφαρμογής που αναπτύσσεται. Στη συνέχεια, ορίζονται *μοντέλα διεργασιών λογισμικού* και περιγράφονται τρία από τα

πιο διαδεδομένα μοντέλα. Πρώτα, αναλύεται το μοντέλο του *''καταρράκτη''*, όπου παρουσιάζονται τα κύρια στάδια του μοντέλου καθώς και τα πλεονεκτήματα και μειονεκτήματά του. Το δεύτερο μοντέλο που περιγράφεται σε αυτό το κεφάλαιο είναι το *μουτέλο της εξελικτικής ανάπτυξης* όπου αναφέρονται οι δύο βασικοί τύποι αυτού του μοντέλου και τα προβλήματα που παρουσιάζονται όταν εφαρμόζεται. Το τελευταίο μοντέλου που παρουσιάζεται είναι η *τεχνολογία λογισμικού βασισμένη σε συνιστούντα μέρη*. Αναλύονται τα στάδια του μοντέλου καθώς και οι λόγοι για τους οποίους αξίζει να ακολουθηθεί.

Στο **κεφάλαιο 3** γίνεται ανάλυση *μεθοδολογιών ταχείας ανάπτυξης λογισμικού*. Το κεφάλαιο ξεκινάει με μια ιστορική αναδρομή για τις μεθόδους ανάπτυξης λογισμικού και συνεχίζει με τους λόγους που οδήγησαν στη δημιουργία πιο ευέλικτων μεθοδολογιών. Στη συνέχεια, περιγράφεται το *ευέλικτο μανιφέστο (agile manifesto)* και οι αρχές που το διέπουν, και το κεφάλαιο ολοκληρώνεται με την ανάλυση των δύο πιο γνωστών μεθοδολογιών ταχείας ανάπτυξης λογισμικού, του *ακραίου προγραμματισμού (extreme programming)* και *συνεχούς ενσωμάτωσης*.

Στο **κεφάλαιο 4** παρουσιάζονται διάφορα εργαλεία ανάπτυξης λογισμικού. Πρώτα παρουσιάζονται τα *ολοκληρωμένα περιβάλλουτα ανάπτυξης λογισμικού (IDEs)*, γίνεται αναφορά για το τι είναι τα IDEs, πως βοηθούν στην ανάπτυξη λογισμικού και περιγράφονται τα χαρακτηριστικά τριών από τα πιο γνωστά IDEs, το *Eclipse*, το *IntelliJ* και το *NetBeans IDE*. Στη συνέχεια του κεφαλαίου, γίνεται ανάλυση των *συστημάτων διαχείρισης εκδόσεων* και περιγράφονται τα χαρακτηριστικά των *CVS* και *Subversion*. Στα πλαίσια των εργαλείων ανάπτυξης λογισμικού, γίνεται ανάλυση των *συστημάτων παρακολούθησης προβλημάτων* και παρουσιάζονται δύο συστήματα αυτής της κατηγορίας, τα *Bugzilla* και *JIRA*. Επιπλέον, περιγράφονται τα χαρακτηριστικά τριών *συστημάτων διαχείρισης έργων λογισμικού*, των *SourceForge*, *GForge* και *Trac*. Το κεφάλαιο κλείνει ορίζοντας τις *μετρικές κώδικα* και περιγράφοντας το εργαλείο στατικής ανάλυσης κώδικα *PMD*.

Το **κεφάλαιο 5** αποτελεί το πρώτο κεφάλαιο του τρίτου μέρους της διπλωματικής, το οποίο πραγματεύεται με τεχνολογίες ανάπτυξης εφαρμογών. Σε αυτό το κεφάλαιο, περιγράφεται η τεχνολογία *Hibernate*. Αναφέρονται οι λόγοι που οδήγησαν στη δημιουργίας της, καθώς και οι δυνατότητες που προσφέρει. Επιπλέον, γίνεται επίδειξη του τρόπου λειτουργίας της και των βασικών χαρακτηριστικών της με τη βοήθεια ενός παραδείγματος.

Στο **κεφάλαιο 6**, γίνεται ανάλυση του περιβάλλοντος εργασίας *Spring*. Αρχικά αναφέρεται το τι είναι η τεχνολογία Spring και περιγράφονται οι δυνατότητες της, τα μέρη απο τα οποία αποτελείται, καθώς και τα χαρακτηριστικά που προσδίδονται σε μια εφαρμογή κατά τη χρήση της. Στη συνέχεια, αναπτύσσεται μια εφαρμογή για την εμφάνιση του γνωστού μηνύματος "Hello World" με τη χρήση της Spring για την παρουσίαση των βασικών της χαρακτηριστικών. Επιπλέον, περιγράφονται οι δύο βασικότερες αρχές που διέπουν τη Spring, το *dependency injection* και το *Aspect-Oriented Programming*, όπου χρησιμοποιούνται παραδείγματα για να γίνουν πιο κατανοητά.

Από το **κεφάλαιο 7** ξεκινάει το τέταρτο μέρος της διπλωματικής εργασίας το οποίο ασχολείται με την ανάπτυξη της εφαρμογής διαχείρισης ερευνητικών έργων. Το κεφάλαιο αυτό αποτελεί την *περιγραφή του συστήματος* που έχει δημιουργηθεί. Αρχικά, περιγράφεται η *δομή του οργανισμού* για τον οποίο αναπτύχθηκε η εφαρμογή και έπειτα αναφέρονται οι *προδιαγραφές της εφαρμογής*, και οι *απαιτήσεις* που θέτει ο οργανισμός.

Στο **κεφάλαιο 8** παρουσιάζεται αναλυτικά ο *σχεδιασμός της εφαρμογής*. Το κεφάλαιο ξεκινάει με τη *δομή του δικτύου* πάνω στο οποίο θα διατίθενται όλες οι υπηρεσίες της εφαρμογής. Στη συνέχεια, περιγράφονται τα *λογικά επίπεδα* από τα οποία αποτελείται η εφαρμογή και εξηγείται τι λειτουργίες περιλαμβάνει κάθε λογικό επίπεδο. Επιπλέον, σε αυτό το κεφάλαιο αναλύονται οι *οντότητες* της εφαρμογής μαζί με τα χαρακτηριστικά τους, οι οποίες, στο τέλος αυτής της ενότητας απεικονίζονται σε ένα διάγραμμα οντοτήτων συσχετίσεων. Σε αυτό το κεφάλαιο, γίνεται και αναφορά στα *είδη χρηστών* του συστήματος και τους ρόλους που μπορεί να έχουν. Το κεφάλαιο ολοκληρώνεται με *την περιγραφή των λειτουργιών* της εφαρμογής.

Το **κεφάλαιο 9**, το οποίο αποτελεί το τελευταίο κεφάλαιο του τέταρτου μέρους, παρουσιάζει τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Ξεκινά με τις *τεχνολογίες που χρησιμοποιήθηκαν* σε κάθε λογικό επίπεδο που όριζε ο σχεδιασμός της εφαρμογής, και συνεχίζει με *αποτελέσματα από ελέγχους* που έγιναν κατά την ανάπτυξή της. Στη συνέχεια αναφέρεται το *περιβάλλον ανάπτυξης* της εφαρμογής και τα διάφορα εργαλεία που χρησιμοποιήθηκαν μαζί με αυτό. Στο τέλος του κεφαλαίου παρουσιάζονται κάποιες *μετρικές κώδικα* της εφαρμογής και αναφέρονται τα μέσα που χρησιμοποιήθηκαν για να υπολογιστούν.

Τέλος, στο **κεφάλαιο 10** παρουσιάζονται κάποια *συμπεράσματα* το οποία προέκυψαν από τη συγγραφή της διπλωματικής εργασίας, και γίνεται ο απολογισμός όλων όσων επιτεύχθηκαν από αυτή την προσπάθεια.

Μέρος ΙΙ

Ανάπτυξη Εφαρμογών

ΚΕΦΑΛΑΙΟ **2**

Διεργασία λογισμικού (software process)

Η ανάπτυξη ενός προϊόντος λογισμικού αποτελείται από ένα σύνολο διαδικασιών και δραστηριοτήτων. Το σύνολο αυτών των δραστηριοτήτων ονομάζεται ***διεργασία λογισμικού (software process)***, πολλές φορές συναντάται και ως ***κύκλος ζωής λογισμικού (software life cycle)***.Υπάρχουν τέσσερις θεμελιώδεις δραστηριότητες οι οποίες είναι κοινές σε όλες τις διεργασίες λογισμικού:

1. *Ο προσδιορισμός των προδιαγραφών του λογισμικού* όπου οι μηχανικοί μαζί με τους πελάτες ορίζουν το πρόγραμμα που θα παραχθεί καθώς και τυχόν περιορισμούς κατά τη χρήση του.

2. *Η ανάπτυξη του λογισμικού* όπου το λογισμικό σχεδιάζεται και αναπτύσσεται.

3. *Η επικύρωση του λογισμικού* όπου το λογισμικό ελέγχεται ώστε να διασφαλιστεί ότι παρέχει αυτά που χρειάζεται ο πελάτης.

4. *Η εξέλιξη του λογισμικού* οπου το λογισμικό τροποποιείται ώστε να προσαρμοστεί στις νέες απαιτήσεις της αγοράς ή του πελάτη.

Διαφορετικά είδη λογισμικού αναπτύσσονται με διαφορετικό τρόπο και απαιτούν διαφορετικές δραστηριότητες για την παραγωγή τους. Για παράδειγμα, για την παραγωγή λογισμικού *πραγματικού χρόνου (real time)* για ένα αεροπλάνο, οι προδιαγραφές θα πρέπει να έχουν οριστεί πλήρως πριν αρχίσει η ανάπτυξή του, εξασφαλίζοντας ότι θα είναι αξιόπιστο και ασφαλές για τις πτήσεις του αεροπλάνου. Αντίθετα, για την παραγωγή συστημάτων *ηλεκτρονικού εμπορίου (e-commerce)* οι προδιαγραφες και το προγραμμα αναπτύσσονται παραλληλα, γιατί κατά τη διάρκεια ανάπτυξης, οι προδιαγραφές μεταβάλλονται συνεχώς, σύμφωνα με τις τάσεις του επιχειρηματικού κόσμου. Συνεπώς, αυτές οι παραπάνω γενικές δραστηριότητες υπάρχει περιπτωση να οργανώνονται διαφορετικά, και οι προδιαγραφές να περιγράφονται με λιγότερες ή περισσότερες λεπτομέρειες για διαφορετικά είδη λογισμικού.

Παρόλο που δεν υπάρχει μία ***ιδανική*** διεργασία λογισμικού, πάντα υπάρχει δυνατότητα βελτίωσης των διεργασιών λογισμικού που χρησιμοποιούνται από πολλούς οργανισμούς. Οι διεργασίες λογισμικού μπορεί να περιλαμβάνουν ξεπερασμένες τεχνολογίες ή να μην εκμεταλλεύονται τις καλές τεχνικές που υπάρχουν στην βιομηχανία λογισμικού. Όμως, οι διεργασίες μέσα σε έναν οργανισμό μπορούν να βελτιωθούν μέσω της προτυποποίησης τους, με αποτέλεσμα τη μείωση της διαφορετικότητας τους. Αυτό οδηγεί στη βελτίωση της επικοινωνίας μεταξύ των τμημάτων του οργανισμού και του χρόνου εκπαίδευσης,αλλά κάνει και πιο οικονομική την

υποστήριξη αυτόματων διαδικασιών. Επιπλέον, η προτυποποίηση των διαδικασιών λογισμικού ειναι μία καλή πρακτική στην μηχανική λογισμικού πολύ σημαντική γιατί ειναι το αρχικό βήμα για την εισαγωγή νέων μεθόδων και τεχνικών ανάπτυξης λογισμικού.

2.1 Μοντέλα Διεργασιών Λογισμικού

Ένα ***μοντέλο διεργασίας λογισμικού*** είναι η απλοποιημένη περιγραφή μιας διεργασίας λογισμικού στην οποία παρουσιάζεται μια επισκόπηση της διεργασίας. Τα μοντέλα διεργασίας μπορεί να περιλαμβάνουν δραστηριότητες που είναι μέρος της διεργασίας λογισμικού, προϊόντα λογισμικού και μηχανικούς λογισμικού. Ορισμένα μοντέλα αποτελούν *ορισμούς* του τρόπου με τον οποίο θα πρέπει να γίνεται η ανάπτυξη λογισμικού, ενώ κάποια άλλα είναι περιγραφές των μεθόδων με τις οποίες γίνεται στην πράξη η ανάπτυξη λογισμικού. Η δημιουργία ενός μοντέλου διεργασίας και η ανάλυση των υποδιεργασιών του βοηθά την ομάδα ανάπτυξης να κατανοήσει το χάσμα που υπάρχει μεταξύ αυτού που θα έπρεπε να γίνεται και αυτού που γίνεται.

Παρακάτω παρουσιάζονται μερικά παραδείγματα μοντέλων διεργασιών που μπορούν να δημιουργηθούν:

1. *Μουτέλο ροών εργασίας (Workflow model)*: Αυτό το μοντέλο δείχνει την ακολουθία των δραστηριοτήτων μιας διεργασίας λογισμικού, καθώς και τα δεδομένα τους, τα αποτελέσματα τους και τυχόν εξαρτήσεις μεταξύ των δραστηριοτήτων. Οι δραστηριότητες σε αυτό το μοντέλο αναπαρίστανται με ενέργειες που πρέπει να γίνουν από το ανθρώπινο δυναμικό.

2. *Μουτέλο ροών δεδομένων ή μοντέλο δραστηριοτήτων (dataflow or activity model)*: Το μοντέλο αυτό αναπαριστά την διεργασία ως ένα σύνολο από δραστηριότητες, κάθε μία εκ των οποίων εκτελεί κάποιο μετασχηματισμό δεδομένων. Δείχνει πως τα δεδομένα της διεργασίας, όπως κάποιες προδιαγραφές, μετασχηματίζονται στα παραγόμενα της διεργασίας, όπως κάποιος σχεδιασμός. Σε αυτό το μοντέλο, οι δραστηριότητες μπορεί να αναπαριστούν μετασχηματισμούς οι οποίοι εκτελούνται από ανθρώπους ή από υπολογιστές.

3. *Μουτέλο ρόλων/ενεργειών (role/action model)*: Παρουσιάζει τους ρόλους όσων εμπλέκονται στην διεργασία λογισμικού και τις αντίστοιχες δραστηριότητες για τις οποίες είναι υπεύθυνοι.

Όπως αναφέρθηκε και παραπάνω, ένα μοντέλο διεργασίας λογισμικού είναι μια γενική αναπαράσταση από συγκεκριμένη σκοπιά μιας διαδικασίας λογισμικού, με αποτέλεσμα να μην παρέχει λεπτομερείς πληροφορίες για την διαδικασία λογισμικού. Παρακάτω αναφέρονται τρία γενικά μοντέλα διεργασιών, τα οποία περιγράφονται από την σκοπιά της αρχιτεκτονικής. Αυτό σημαίνει ότι βλέπουμε το περιβάλλον εργασίας μιας διεργασίας αλλα όχι λεπτομέρειες συγκεκριμένων δραστηριοτήτων.

Αυτά τα γενικά μοντέλα δεν ειναί σαφείς περιγραφές των διεργασιών λογισμικού. Αντιθέτως, είναι μια αφαίρεση της διεργασίας που μπορεί να χρησιμοποιηθεί για την επεξήγηση διαφορετικών προσεγγίσεων στην ανάπτυξη λογισμικού. Μπορούμε να τα σκεφτούμε ως πλαίσια εργασίας των διεργασιών που μπορούν να επεκταθούν και να προσαρμοστούν με τέτοιο τρόπο ώστε να δημιουργήσουν πιο ειδικές διεργασίες ανάπτυξης λογισμικού.

Τα μοντέλα που θα περιγραφούν αναλυτικά είναι τα παρακάτω:

1. *Μοντέλο καταρράκτη (waterfall model)*: Αυτό το μοντέλο ξεκινά από τις θεμελιώδεις δραστηριότητες διεργασιών: τον ορισμό των προδιαγραφών, την ανάπτυξη, τον έλεγχο και την εξέλιξη του προϊόντος, και τις αναπαριστά ως ξεχωριστές φάσεις της διεργασίας λογισμικού όπως: συγγραφή των προδιαγραφών, σχεδιασμός του λογισμικού, υλοποίηση του λογισμικού, δοκιμή λογισμικού κτλ.

2. *Εξελικτική ανάπτυξη (Evolutionary development)*: Σε αυτή την προσέγγιση εκτελούνται ταυτόχρονα οι δραστηριότητες του ορισμού προδιαγραφών, της ανάπτυξης και του ελέγχου ποιότητας. Ένα αρχικό σύστημα αναπτύσσεται γρήγορα από αφηρημένες προδιαγραφές. Αυτό συνήθως βελτιώνεται με την βοήθεια του πελάτη για να δημιουργηθεί το τελικό σύστημα που να ικανοποιεί τις ανάγκες του.

3. *Τεχνολογία λογισμικού βασισμένη σε συνιστώντα μέρη (Component-based software engineering)*: Αυτή η προσέγγιση βασίζεται στην ύπαρξη ενός σημαντικού αριθμού επαναχρησιμοποιήσιμων συνιστωσών. Η διεργασία της ανάπτυξης του συστήματος εστιάζεται στην μίξη αυτών των συνιστωσών για τη δημιουργία αυτού του συστήματος παρά στην ανάπτυξή του από την αρχή.

Αυτά τα τρία γενικά μοντέλα διεργασίας χρησιμοποιούνται εκτενώς στη σύγχρονες πρακτικές της τεχνολογίας λογισμικού. Δεν είναι αμοιβαία αποκλειόμενα και σε πολλές περιπτώσεις χρησιμοποιούνται ταυτόχρονα, ειδικά στην ανάπτυξη μεγάλων συστημάτων. Υπάρχουν περιπτώσεις που τα υποσυστήματα μεγάλων συστημάτων αναπτύσσονται με διαφορετικές προσεγγίσεις.

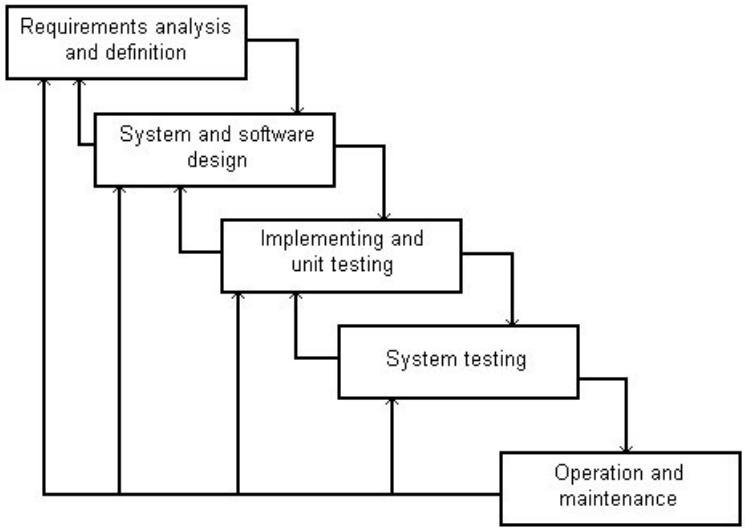
2.1.1 Μοντέλο "καταρράκτη" (The waterfall model)

Το πρώτο μοντέλο διεργασίας ανάπτυξης λογισμικού το οποίο προέρχεται απο πιο γενικές διεργασίες ανάπτυξης λογισμικού. Το μοντέλο αυτό παρουσιάζεται στην παρακάτω εικόνα. Λόγο της μορφής της γραμμικής ακολουθίας των φάσεων, το μοντέλο έγινε γνωστό ως το ***μοντέλο "καταρράκτη"***. Τα κύρια σταδια του μοντέλου αντιστοιχούν σε βασικές δραστηριότητες ανάπτυξης λογισμικού:

1. *Ανάλυση των απαιτήσεων και προσδιορισμός τους (Requirements analysis and definition)*: Οι υπηρεσίες του συστήματος, οι περιορισμοί και οι στόχοι καθορίζονται με την συμβολή των χρηστών του συστήματος. Καθορίζονται με λεπτομέρεια και αποτελούν τις προδιαγραφές του συστήματος.

2. *Σχεδιασμός συστήματος και λογισμικού (System and software design)*: Η διεργασία του σχεδιασμού του συστήματος διαχωρίζει τις απαιτήσεις σε συστήματα είτε λογισμικού είτε υλικού. Καθιερώνει μία συνολική αρχιτεκτονική για την ανάπτυξη συστήματος. Ο σχεδιασμός του λογισμικού περιλαμβάνει την αναγνώριση και την περιγραφή των αφαιρέσεων του συστήματος λογισμικού και τις σχέσεις μεταξύ τους.

3. *Υλοποίηση και έλεγχος μονάδας (Implementation and unit testing)*: Σε αυτό το στάδιο πραγματοποιείται ο σχεδιασμός του λογισμικού, ως ένα σύνολο προγραμμάτων ή μονάδων προγραμμάτων. Ο έλεγχος μονάδας περιλαμβάνει την επαλήθευση ότι κάθε μονάδα πληροί τις απαιτήσεις της.



Σχήμα 2.1: Μοντέλο καταρράκτη

4. *Συνένωση και έλεγχος συστήματος (Intergration and system testing)*: Κάθε ξεχωριστή μονάδα του προγράμματος ενοποιείται και ελέγχονται ως ένα συνολικό σύστημα για να διασφαλιστεί ότι πληρούνται οι απαιτήσεις του λογισμικού.

5. *Λειτουργία και συντήρηση (Operation and maintenance)*: Συνήθως, αν και δεν ειναι απαραίτητο, αυτή είναι η μεγαλύτερη φάση στο κύκλο ζωής του λογισμικού. Το σύστημα εγκαθίσταται και χρησιμοποιείται. Η συντήρηση περιλαμβάνει την διόρθωση λαθών, τα οποία δεν είχαν ανακαλυφθεί σε προηγούμενα στάδια του κύκλου ζωής. Επιπλέον, πραγματοποιείται η βελτίωση της υλοποίησης των μονάδων του συστήματος και ο εμπλουτισμός του με νέες λειτουργίες οι οποίες προέρχονται από τις νέες απαιτήσεις που ανακαλύπτονται κατά τη χρήση του συστήματος.

Θεωρητικά, το αποτέλεσμα κάθε φάσης είναι ένα ή περισσότερα έγγραφα, τα οποία πρέπει να εγκριθούν. Η επόμενη φάση δεν αρχίζει αν δεν έχει ολοκληρωθεί η προηγούμενη. Στην πραγματικότητα, αυτές οι φάσεις επικαλύπτονται και τροφοδοτούν με πληροφορίες η μία την άλλη. Κατά τη φάση του σχεδιασμού ανακαλύπτονται προβλήματα στις προδιαγραφές, κατά την φάση της συγγραφής κώδικα ανακαλύπτονται προβλήματα σχεδιασμού και ου το κάθε εξής. Η *διεργασία λογισμικού* δεν είναι ένα απλό γραμμικό μοντέλο, αλλά περιλαμβάνει μία ακολουθία επαναλήψεων των διαδικασιών της ανάπτυξης του λογισμικού.

Λόγο του κόστους της παραγωγής και έγκρισης των εγγράφων, οι επαναλήψεις είναι δαπανηρές και περιλαμβάνουν σημαντική επανάληψη εργασιών. Για αυτό το λόγο, μετά απο έναν μικρό αριθμό επαναλήψεων, είναι φυσικό να διακόπτονται μέρη της ανάπτυξης του λογισμικού, όπως ο καθορισμός των προδιαγραφών, και να συνεχίζει μαζί με επόμενα στάδια ανάπτυξης. Η λύση των προβλημάτων αναβάλλεται, αγνοείται ή επαναπρογραμματίζεται. Αυτή η πρώιμη παύση του προσδιορισμού των προδιαγραφών μπορεί να έχει ως αποτέλεσμα το σύστημα να μην πληροί τις απαιτήσεις του πελάτη. Επιπλέον, μπορεί να οδηγήσει στην δημιουργία συστημάτων με

κακή δομή, καθώς τα προβλήματα του σχεδιασμού έχουν παρακαμφθεί με διάφορα τεχνάσματα κατά τη διάρκεια της υλοποίησης.

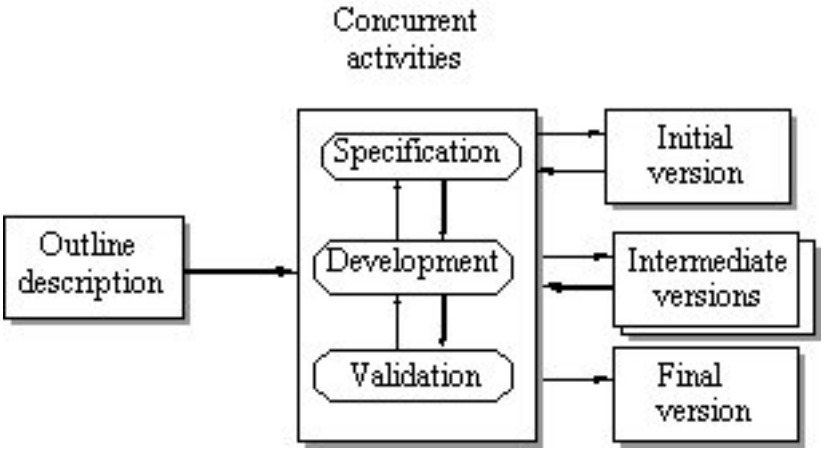
Κατά τη διάρκεια της τελευταίας φάσης του κύκλου ζωής (λειτουργία και συντήρηση), το λογισμικό τίθεται σε λειτουργία. Λάθη και παραλείψεις στις αρχικές απαιτήσεις του συστήματος ανακαλύπτονται. Προκύπτουν σχεδιαστικά και προγραμματιστικά λάθη καθώς και η ανάγκη για προσθήκη νέων λειτουργιών. Για αυτό το λόγο, το σύστημα πρέπει να εξελιχθεί για να παραμείνει χρήσιμο. Η πραγματοποίηση αυτών των αλλαγών (συντήρηση λογισμικού) μπορεί να περιλαμβάνει την επανάληψη προηγούμενων φάσεων της διαδικασίας λογισμικού.

Τα πλεονεκτήματα του μοντέλου "καταρράκτη" είναι η τεκμηρίωση που παράγεται σε κάθε φάση και το γεγονός ότι συνταιριάζει με άλλα μοντέλα διεργασίας τεχνολογίας λογισμικού. Το μεγαλύτερο του πρόβλημα είναι η μη ευέλικτη διαίρεση του έργου σε διακριτά στάδια. Δεσμεύσεις πρέπει να γίνουν από τα αρχικά στάδια, το οποίο έχει ως αποτέλεσμα την δυσκολία της ανταπόκρισης του μοντέλου στην αλλαγή των απαιτήσεων του πελάτη.

Επομένως, το μοντέλο "καταρράκτη" θα πρέπει να χρησιμοποιείται μόνο όταν οι απαιτήσεις του πελάτη έχουν κατανοηθεί πλήρως και δεν προβλέπεται να αλλάξουν αρκετά κατά την διάρκεια της ανάπτυξης του συστήματος.

2.1.2 Εξελικτική ανάπτυξη (Evolutionary development)

Η εξελικτική ανάπτυξη είναι βασισμένη στην εξής ιδέα: αρχικά πραγματοποιείται μια αρχική υλοποίηση, η οποία παρουσιάζεται στον μελλοντικό χρήστη ώστε να αναφέρει τις παρατηρήσεις του, έπειτα βελτιώνεται και περνάει από πολλές εκδόσεις μέχρι να αναπτυχθεί το σύστημα που να ικανοποιεί πλήρως τον πελάτη. Οι διαδικασίες της συγγραφής των προδιαγραφών, της ανάπτυξης και της επικύρωσης του συστήματος πραγματοποιούνται παράλληλα, προσφέρουν άμεση ανάδραση και δεν αποτελούν ξεχωριστές δραστηριότητες κατά την ανάπτυξη.



Σχήμα 2.2: Μοντέλο εξελικτικής ανάπτυξης

Υπάρχουν δύο θεμελιώδεις τύποι εξελικτικής ανάπτυξης:

1. *Διερευνητική ανάπτυξη (Exploratory development)* όπου ο στόχος της διεργασίας λογισμικού είναι μαζί με τον πελάτη να ανακαλυφθούν οι απαιτήσεις του και να παραδοθεί ένα τελικό σύστημα. Η ανάπτυξη ξεκινά με τα μέρη του

συστήματος στα οποία οι απαιτήσεις έχουν κατανοηθεί πλήρως. Το σύστημα εξελίσσεται προσθέτοντας νέες λειτουργίες οι οποίες προτείνονται από τον πελάτη.

2. *Ανάπτυξη με πρωτότυπο που θα πεταχτεί (Throwaway prototyping)* όπου ο στόχος της διεργασίας της εξελικτικής ανάπτυξης είναι η κατανόηση των απαιτήσεων του πελάτη με αποτέλεσμα την δημιουργία καλύτερου ορισμού των προδιαγραφών του συστήματος. Το πρωτότυπο εστιάζει στον πειραματισμό με τις απαιτήσεις του συστήματος που δεν έχουν κατανοηθεί πλήρως.

Η εξελικτική προσέγγιση στην ανάπτυξη λογισμικού συχνά είναι πιο αποτελεσματική από το μοντέλο "καταρράκτη" στην παραγωγή συστημάτων τα οποία πληρούν τις άμεσες απαιτήσεις ενός πελάτη. Το πλεονέκτημα μιας διεργασίας λογισμικού η οποία βασίζεται στην εξελικτική ανάπτυξη είναι ότι ο καθορισμός των προδιαγραφών μπορεί να σταδιακά. Όσο οι χρήστες του συστήματος κατανοούν περισσότερο τα προβλήματα που έχουν, αυτό απεικονίζεται στο σύστημα. Ωστόσο, από την σκοπιά της τεχνολογίας λογισμικού και της διαχείρισης, το μοντέλο της εξελικτικής ανάπτυξης παρουσιάζει δύο προβλήματα:

1. *Η πρόοδος της διεργασίας δεν είναι εμφανής*: οι υπεύθυνοι της ανάπτυξης χρειάζονται τακτικά κάποια αποτελέσματα ή παραδοτέα ώστε να υπολογίσουν τη πρόοδο της ανάπτυξης. Αν πολλά συστήματα αναπτύσσονται σε σύντομο χρονικό διάστημα, δεν είναι αποδοτικό οικονομικά να δημιουργείς έγγραφα που παρουσιάζουν κάθε έκδοση του συστήματος.

2. *Τα συστήματα συχνά δεν είναι καλά δομημένα*: Οι συνεχείς αλλαγές μπορεί να φθείρουν την δομή του λογισμικού. Η πραγματοποίηση αλλαγών στο λογισμικό γίνεται όλο και περισσότερο δύσκολη και δαπανηρή

Για συστήματα μικρού και μεσαίου μεγέθους (μέχρι 500,000 γραμμές κώδικα), σύμφωνα με το [22], η εξελικτική ανάπτυξη αποτελεί την καλύτερη προσέγγιση ανάπτυξης λογισμικού. Τα προβλήματα αυτής της προσέγγισης αρχίζουν να γίνονται έντονα για μεγάλα, πολύπλοκα και με μεγάλο κύκλο ζωής συστήματα, όπου ξεχωριστές ομάδες αναπτύσσουν διαφορετικά μέρη του συστήματος, γεγονός που κάνει δύσκολη την ενοποίηση των παραγώγων κάθε ομάδας.

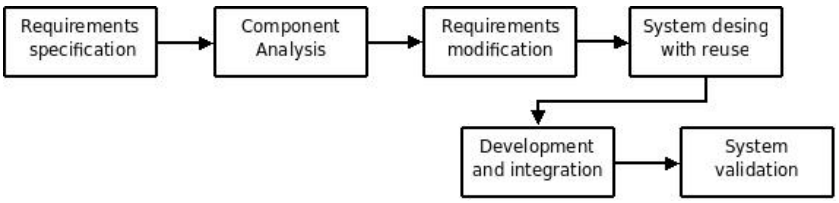
Για συστήματα μεγάλου μεγέθους, ο ίδιος συγγραφέας προτείνει μια μικτή διεργασία λογισμικού που να συνδυάζει τα πλεονεκτήματα των μοντέλων "καταρράκτη" και εξελικτικής ανάπτυξης. Αυτή η μικτή διεργασία μπορεί να περιλαμβάνει την ανάπτυξη ενός πρωτοτύπου που θα πεταχτεί χρησιμοποιώντας την εξελικτική προσέγγιση για να αντιμετωπίσουν τυχόν αβεβαιότητες στην προδιαγραφές του συστήματος. Έπειτα, το σύστημα θα μπορούσε να υλοποιηθεί ξανά χρησιμοποιώντας μια πιο δομημένη προσέγγιση. Στα μέρη του συστήματος που έχουν κατανοηθεί πλήρως, μπορούν να οριστούν οι προδιαγραφές και να αναπτυχθούν χρησιμοποιώντας κάποια διεργασία λογισμικού που βασίζεται στο μοντέλο του "καταρράκτη". Άλλα μέρη του συστήματος, όπως η διεπαφή χρήστη, που είναι δύσκολο να προσδιοριστούν εκ των προτέρων, θα πρέπει πάντα να αναπτύσσονται βάσει της προσέγγισης της διερευνητικής ανάπτυξης.

2.1.3 Τεχνολογία λογισμικού βασισμένη σε συνιστώντα μέρη (Component-based software engineering)

Στην πλειοψηφία των έργων λογισμικού, γίνεται επαναχρησιμοποίηση λογισμικού. Αυτό συνήθως γίνεται ανεπίσημα, όταν οι άνθρωποι που δουλεύουν σε ένα έργο έχουν εργαστεί σε ένα άλλο έργο με παρόμοιο σχεδιασμό και κώδικα. Ψάχνουν να βρουν τα κομμάτια κώδικα που μπορεί να μοιάζουν, τα τροποποιούν όπως πρέπει και τα ενσωματώνουν στο σύστημα που αναπτύσσουν. Στην εξελικτική προσέγγιση που περιγράφηκε παραπάνω, η επαναχρησιμοποίηση είναι βασική ώστε να επιτευχθεί η γρήγορη ανάπτυξη του συστήματος.

Αυτή η ανεπίσημη επαναχρησιμοποίηση κώδικα συμβαίνει ανεξάρτητα με το ποια διεργασία ανάπτυξης ακολουθείται. Ωστόσο, τα τελευταία χρόνια, μία νέα προσέγγιση ανάπτυξης λογισμικού εμφανίστηκε και χρησιμοποιείται όλο και περισσότερο. Ονομάζεται *τεχνολογία λογισμικού βασισμένη σε συνιστώντα μέρη (component-based software engineering - CBSE)* και βασίζεται στην επαναχρησιμοποίηση κώδικα.

Αυτή η προσέγγιση, βασίζεται σε ένα μεγάλο σύνολο επαναχρησιμοποιήσιμων τμημάτων λογισμικού και στα ολοκληρωμένα πλαίσια εργασίας αυτών των τμημάτων. Πολλές φορές, αυτά τα τμήματα είναι αυτόνομα συστήματα (*έτοιμα συστήματα (COTS — commercial off-the-shelf systems*) που προσφέρουν συγκεκριμένη λειτουργικότητα, όπως διαμόρφωση κειμένου ή αριθμητικούς υπολογισμούς. Το γενικό μοντέλο διεργασίας CBSE παρουσιάζεται στο παρακάτω σχήμα:



Σχήμα 2.3: Μοντέλο τεχνολογίας λογισμικού βασισμένη σε συνιστώντα μέρη

Ενώ τα στάδια του ορισμού των αρχικών προδιαγραφών και της επικύρωσης είναι παρόμοια με άλλα μοντέλα, σε μια διεργασία λογισμικού προσανατολισμένη στην επαναχρησιμοποίηση, τα ενδιάμεσα στάδια ειναι διαφορετικά. Αυτά τα στάδια είναι τα παρακάτω:

1. *Ανάλυση των συνιστωσών μερών (component analysis)*: Δοθείσης μίας περιγραφής των απαιτήσεων, αναζητούνται τα συνιστώντα μέρη που θα υλοποιήσουν τις απαιτήσεις της περιγραφής. Συνήθως, δεν υπάρχει ακριβής αντιστοιχία, και οι συνιστώσες που θα χρησιμοποιηθούν καλύπτουν εν μέρη την λειτουργικότητα που απαιτείται απο τον πελάτη.

2. *Τροποποίηση των απαιτήσεων (requirements modification)*: Κατά τη διάρκεια αυτής της φάσης, οι απαιτήσεις αναλύονται χρησιμοποιώντας πληροφορίες για τα συνιστώντα μέρη που έχουν βρεθεί από την προηγούμενη φάση. Μετά τροποποιούνται ώστε να παρουσιάζουν τα διαθέσιμα συνιστώντα μέρη. Όπου είναι δεν δυνατό να γίνουν τροποποιήσεις, επαναλαμβάνεται η φάση της ανάλυσης των συνιστωσών μερών για την αναζήτηση νέων λύσεων.

3. *Σχεδιασμός συστήματος με επαναχρησιμοποίησηsystem design with reuse*: Κατά τη διάρκεια αυτού του σταδίου, σχεδιάζεται το πλαίσιο εργασίας του συστήματος ή επαναχρησιμοποιείται κάποιο υπάρχον. Οι σχεδιαστές λαμβάνουν

υπόψιν τα μέρη που χρησιμοποιούνται ξανά και οργανώνουν το πλαίσιο εργασίας με τέτοιο τρόπο ώστε να τους εξυπηρετεί.

4. *Ανάπτυξη και συνένωση (development and integration)*: Το λογισμικό αναπτύσσεται, και τα συνιστώντα μέρη και τα έτοιμα συστήματα (COTS) συνδυάζονται για να δημιουργήσουν το νέο σύστημα. Η συνένωση του συστήματος, σε αυτό το μοντέλο, μπορεί να αποτελεί μέρος της διαδικασίας της ανάπτυξης και όχι μία ξεχωριστή διαδικασία.

Η τεχνολογία λογισμικού βασισμένη σε συνιστώντα μέρη έχει το πλεονέκτημα ότι το μέγεθος του λογισμικού που πρέπει να αναπτυχθεί είναι μειωμένο, με αποτέλεσμα την μείωση του κόστους και του κινδύνου. Συχνά, οδηγεί και σε ταχύτερη παράδοση του λογισμικού. Ωστόσο, υποχωρήσεις στις απαιτήσεις είναι αναπόφευκτες με αποτέλεσμα το παραδοθέν λογισμικό να μην πληροί τις πραγματικές απαιτήσεις του πελάτη. Επιπρόσθετα, χάνεται ο πλήρης έλεγχος της εξέλιξης του συστήματος καθώς νέες εκδόσεις των επαναχρησιμοποιημένων συνιστωσών δεν βρίσκονται κάτω από τον έλεγχο του οργανισμού που τα χρησιμοποιούν.

ΚΕΦΑΛΑΙΟ 3

Μεθοδολογίες ταχείας ανάπτυξης λογισμικού

Τη δεκαετία του 80′ μέχρι και τη δεκαετία του 90′, υπήρχε μια ευρύτατα διαδεδομένη άποψη ότι ο καλύτερος τρόπος για την δημιουργία καλού λογισμικού ήταν μέσω προσεκτικού σχεδιασμού του έργου, επίσημη διαβεβαίωση της ποιότητας, χρήση μεθόδων σχεδιασμού και ανάλυσης υποστηριζόμενα από CASE εργαλεία, έλεγχο και ανάπτυξη του έργο μέσω αυστηρών διεργασιών. Αυτή η άποψη άρχισε να προκαλεί ανησυχία στην κοινότητα των μηχανικών λογισμικού κατά την ανάπτυξη μεγάλης κλίμακας και κύκλου ζωής συστημάτων λογισμικού που δημιουργούνταν από έναν μεγάλο αριθμό ξεχωριστών προγραμμάτων.

Κάποια ή και όλα τα συστήματα συχνά ήταν κρίσιμα συστήματα. Το λογισμικό αναπτυσσόταν από μεγάλες ομάδες που κάποιες φορές βρίσκονταν και σε διαφορετικές εταιρίες,οι οποίες ήταν γεωγραφικά διασκορπισμένες, και εργάζονταν πάνω στο λογισμικό για μεγάλες χρονικές περιόδους. Ένα παράδειγμα τέτοιου είδους λογισμικού είναι τα συστήματα ελέγχου ενός σύγχρονου αεροσκάφους, όπου μεσολαβούν δέκα χρόνια περίπου από τον ορισμό των προδιαγραφών τους μέχρι την λειτουργία του συστήματος. Αυτές οι προσεγγίσεις περιλαμβάνουν σημαντικές επιβαρύνσεις τόσο οικονομικές όσο και σε προσπάθεια, που αφορούν τον σχεδιασμό ενός πλάνου για την ανάπτυξη του συστήματος, τον σχεδιασμό του ίδιου του συστήματος και την τεκμηρίωση του. Αυτές οι επιβαρύνσεις είναι δικαιολογημένες λόγο της συνεργασίας πολλών ομάδων ανάπτυξης, όταν το σύστημα είναι κρίσιμο και όταν πολλοί διαφορετικοί άνθρωποι είναι υπεύθυνοι για την συντήρηση του σε όλη την διάρκεια του κύκλου ζωής του λογισμικού.

Ωστόσο, όταν αυτή η επιβαρυμένη και βασισμένη σε πλάνο εργασίας προσέγγιση ανάπτυξης λογισμικού χρησιμοποιόταν σε μικρού ή μεσαίου μεγέθους συστήματα επιχειρήσεων, αυτή η επιβάρυνση ήταν τόσο μεγάλη που κυριαρχούσε κατά τη διεργασία της ανάπτυξης λογισμικού. Ο περισσότερος χρόνος ξοδευόταν στο πως έπρεπε να αναπτυχθεί το σύστημα παρά στην ανάπτυξη και στην δοκιμή του συστήματος. Όταν οι απαιτήσεις του συστήματος άλλαζαν, η αναθεώρηση του συστήματος ήταν βασική και ο ορισμός των απαιτήσεων καθώς και ο σχεδιασμός του συστήματος έπρεπε να τροποποιηθούν μαζί με το ίδιο το πρόγραμμα.

3.1 Ευέλικτο Μανιφέστο (Agile Manifesto)

Η απογοήτευση για τις επιβαρυμένες μεθόδους ανάπτυξης λογισμικού οδήγησε, κατά τη δεκαετία του 90′, ένα μεγάλο μέρος μηχανικών λογισμικού να προτείνουν νέες, πιο ευέλικτες μεθόδους. Αυτές οι μέθοδοι επέτρεπαν στην ομάδα ανάπτυξης να επικεντρώνεται στο λογισμικό παρά στον σχεδιασμό του και την τεκμηρίωσή του. Οι ***ευέλικτες μέθοδοι (agile methods)*** βασίζονται εξολοκλήρου σε μία προσέγγιση όπου επαναλαμβάνονται οι διαδικασίες του ορισμού προδιαγραφών του λογισμικών, την ανάπτυξης και της παράδοσής του, και αρχικά σχεδιάστηκαν για την υποστήριξη της ανάπτυξης επιχειρηματικού λογισμικού όπου οι απαιτήσεις του συστήματος άλλαζαν συχνά κατά τη διάρκεια της ανάπτυξης. Ο σκοπός τους είναι να παραδώσουν γρήγορα λειτουργικό λογισμικό στους πελάτες, οι οποίοι έπειτα μπορούν να ζητήσουν νέες ή αλλαγμένες λειτουργίες ενσωματωμένες στις επόμενες εκδόσεις του συστήματος.

Στις 11 έως 13 Φεβρουαρίου 2001, σε ένα χειμερινό θέρετρο της οροσειράς Wasatch της Utah συγκεντρώθηκαν εκπρόσωποι διάφορων νέων μεθοδολογιών ώστε να συζητήσουν την ανάγκη νέων ελαφρότερων μεθοδολογιών ως εναλλακτικές των παραδοσιακών επιβαρυμένων. Οι συζητήσεις κατέληξαν σε ένα *μανιφέστο* [1] βασικών αρχών οι οποίες αποτέλεσαν τα θεμέλια της ευέλικτης ανάπτυξης λογισμικού.

Πιθανότατα η πιο γνωστή ευέλικτη μέθοδος είναι ο ***“ακραίος προγραμματισμός” (extreme programming)*** ([7],[8]) ο οποίος περιγράφεται παρακάτω. Ωστόσο, υπάρχουν και άλλες μέθοδοι όπως: *Scrum ([26]), Crystal ([11]), Adaptive Software Development ([16]), DSDM ([31]) και Feature Driven Developement ([24])*. Η επιτυχία αυτών των μεθόδων οδήγησε στην ένταξή τους σε πιο παραδοσιακές μεθόδους ανάπτυξης λογισμικού που βασίζονται στην μοντελοποίηση του συστήματος.

Παρόλο που αυτές οι μέθοδοι βασίζονται στην ιδέα της αυξανόμενης ανάπτυξης και παράδοσης λογισμικού, προτείνουν διαφορετικές διεργασίες για να το πετύχουν. Ωστόσο, μοιράζονται ένα κοινό σύνολο αρχών και για αυτό έχουν πολλά κοινά σημεία. Αυτές οι αρχές φαίνονται σε παρακάτω πίνακα:

Στην πραγματικότητα, οι αρχές στις οποίες βασίζονται οι ευέλικτες μέθοδοι είναι πολλές φορές δύσκολο να πραγματοποιηθούν:

1. Παρόλο που η ιδέα της ανάμειξης του πελάτη στην διαδικασία ανάπτυξης του λογισμικού είναι ελκυστική, η επιτυχία της εξαρτάται από την επιθυμία και την δυνατότητα του πελάτη να διαθέσει χρόνο στην ομάδα ανάπτυξης. Συχνά, οι πελάτες έχουν άλλες υποχρεώσεις και δεν μπορούν να έχουν πλήρη συμμετοχή στην ανάπτυξη του λογισμικού.

2. Κάποια άτομα, μέλη μιας ομάδας, μπορεί να μην έχουν τον κατάλληλο χαρακτήρα και προσωπικότητα για συμμετέχουν εντατικά σε μία ομάδα, κάτι που είναι τυπικό στις ευέλικτες μεθόδους. Για αυτό το λόγο, μπορεί να μην συνεργάζονται σωστά με τα άλλα μέλη της ομάδας.

3. Το να θέτεις προτεραιότητες στις αλλαγές του συστήματος μπορεί να γίνει δύσκολο, ειδικά για συστήματα που υπάρχουν περισσότερα του ενός άτομα που

[1] *Οι 17 συγγραφεις του μανιφέστο ήταν οι: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas*

Αρχή	Περιγραφή
Ανάμειξη του πελάτη	Οι πελάτες θα πρέπει να είναι άμεσα αναμεμειγμένοι με την διεργασία της ανάπτυξης λογισμικού. Ο ρόλος τους είναι να παρέχουν νέες απαιτήσεις, να τους δίνουν προτεραιότητες και να αξιολογούν τις επαναληπτικές εκδόσεις του συστήματος.
Αυξανόμενη παράδοση	Το λογισμικό αναπτύσσεται σε μέρη και οι πελάτες ορίζουν τις απαιτήσεις που θα περιληφθούν σε κάθε μέρος.
Άνθρωποι, όχι διεργασίες	Οι ικανότητες της ομάδας ανάπτυξης θα πρέπει να αναγνωρίζονται και να εκμεταλλεύονται. Τα μέλη της ομάδας θα πρέπει να έχουν την δυνατότητα να εργάζονται με τον δικό τους τρόπο και όχι στα πλαίσια μιας ορισμένης διεργασίας.
Αποδοχή αλλαγών	Θα πρέπει να αναμένονται αλλαγές στις απαιτήσεις του συστήματος, για αυτό το λόγο ο σύστημα πρέπει να σχεδιαστεί ώστε να δέχεται τέτοιες αλλαγές.
Διατήρηση απλότητας	Θα πρέπει να γίνεται εστίαση στην απλότητα τόσο του λογισμικού που αναπτύσσεται όσο και της διαδικασίας της ανάπτυξης. Όποτε είναι δυνατό, εκτελούνται εργασίες για την εξάλειψη πολυπλοκότητας από το σύστημα.

Πίνακας 3.1: Οι βασικές αρχές των ευέλικτων μεθόδων

θέτουν τις προτεραιότητες για τις αλλαγές του συστήματος. Συχνά, κάθε άτομο θέτει διαφορετικές προτεραιότητες σε διαφορετικές αλλαγές.

4. Η διατήρηση της απλότητας απαιτεί περισσότερη εργασία. Υπό την πίεση των προθεσμιών παράδοσης του συστήματος, η ομάδα μπορεί να μην έχει το χρόνο να κάνει τις επιθυμητές απλοποιήσεις στο σύστημα.

Ένα άλλο πρόβλημα, μη τεχνικό, το οποίο είναι γενικό πρόβλημα των μεθόδων της αυξανόμενης ανάπτυξης και παράδοσης, προκύπτει όταν ο πελάτης χρησιμοποιεί έναν εξωτερικό οργανισμό για την ανάπτυξη του συστήματος. Το κείμενο των προδιαγραφών του συστήματος συνήθως αποτελεί μέρος του συμβολαίου μεταξύ του πελάτη και του οργανισμού που το αναπτύσσει. Λόγο του ότι ο αυξανόμενος ορισμός των απαιτήσεων αποτελεί μέρος των ευέλικτων μεθόδων, η συγγραφή συμβολαίων για αυτό τον τύπο ανάπτυξης μπορεί να είναι δύσκολη.

Συνεπώς, οι ευέλικτες μέθοδοι βασίζονται στα συμβόλαια, στα οποία ο πελάτης πληρώνει για τον χρόνο που απαιτείται για την ανάπτυξη του λογισμικό και όχι για ένα συγκεκριμένο σύνολο απαιτήσεων. Εφόσον όλα πάνε καλά, αυτό ωφελεί και τον πελάτη και τον δημιουργό του συστήματος. Ωστόσο, αν προκύψουν προβλήματα, θα προκύψουν διαφωνίες, που δύσκολα επιλύονται, για το ποιος ευθύνεται και ποιος πρέπει να πληρώσει για τον επιπλέον χρόνο και τους πόρους που χρειάστηκαν για να επιλυθούν τα προβλήματα.

3.2 Ακραίος Προγραμματισμός (Extreme programming)

Ο ***ακραίος προγραμματισμός (Extreme programming — XP)*** ειναι ίσως η πιο γνωστή και η πιο διαδεδομένη από τις ευέλικτες μεθόδους. Βαφτίστηκε έτσι από τον Beck ([8]) γιατί αυτή η μέθοδος δημιουργήθηκε χρησιμοποιώντας καλές και αναγνωρισμένες πρακτικές, όπως η επαναλαμβανόμενη ανάπτυξη, και την ανάμειξη του πελάτη σε *υπερβολικό (extreme)* βαθμό.

Στην μέθοδο XP, όλες οι απαιτήσεις εκφράζονται ως σενάρια, τα οποία υλοποιούνται άμεσα ως μία σειρά εργασιών. Οι προγραμματιστές εργάζονται ανά δύο και δοκιμάζουν κάθε εργασία πριν γράψουν τον κώδικα. Όλες οι δοκιμές πρέπει να ολοκληρωθούν με επιτυχία πριν ενσωματωθεί ο κώδικας τους στο σύστημα. Μεσολαβεί μικρό χρονικό διάστημα μεταξύ των διαφορετικών εκδόσεων του συστήματος.

Η μέθοδος XP περιλαμβάνει ένα αριθμό πρακτικών που αντιστοιχούν στις αρχές των ευέλικτων μεθόδων ανάπτυξης:

1. Η αυξανόμενη ανάπτυξη στηρίζεται στις μικρές και συχνές εκδόσεις του συστήματος, καθώς και σε μία προσέγγιση όπου η περιγραφή των απαιτήσεων βασίζεται στα σενάρια που παρέχει ο πελάτης και αποτελούν τη βάση για τον σχεδιασμό της διεργασίας.

2. Η ανάμειξη του πελάτη στηρίζεται στην πλήρη δέσμευση το πελάτη στην ομάδα ανάπτυξης. Ο εκπρόσωπος του πελάτη ή ο ίδιος ο πελάτης παίρνουν μέρος στην ανάπτυξη του συστήματος και είναι υπεύθυνος στο να καθορίζει αποδεκτές δοκιμές του συστήματος

3. Η άποψη "άνθρωποι, όχι διεργασίες" στηρίζεται στον προγραμματισμό ανά ζεύγη, τη συλλογική ιδιοκτησία του κώδικα του συστήματος, και τον υποφερτό ρυθμό ανάπτυξης που δεν περιλαμβάνει υπερβολικές ώρες εργασίας.

4. Η αλλαγή στηρίζεται σε συχνές εκδόσεις του συστήματος, την ανάπτυξη μετά τον έλεγχο και τη συνεχή συνένωση.

5. Η διατήρηση της απλότητας στηρίζεται στις συνεχείς αλλαγές του κώδικα χάριν της βελτίωσης της ποιότητάς του και στη χρήση απλών σχεδιασμών που δεν προσδοκούν μελλοντικές αλλαγές στο σύστημα.

Η μέθοδος XP αποτελεί μια "ακραία" προσεγγιση της επαναλαμβανόμενης ανάπτυξης. Νέες εκδόσεις του συστήματος μπορεί να δημιουργούνται πολλές φορές κάθε μέρα και πρόσθετες λειτουργίες παραδίδονται στον πελάτη κάθε δύο εβδομάδες. Όταν ο προγραμματιστής κατασκευάζει το σύστημα το οποίο προορίζεται για την κυκλοφορία μιας νέας έκδοσης, θα πρέπει να το ελέγξει χρησιμοποιώντας τις υπάρχουσες αυτοματοποιημένες δοκιμές καθώς και τις δοκιμές για τον έλεγχο των νέων λειτουργιών. Η νέα έκδοση του συστήματος είναι αποδεκτή μονο όταν όλες οι δοκιμές είναι επιτυχείς.

Μια βασική αρχή την παραδοσιακής τεχνολογίας λογισμικού είναι ότι θα πρέπει να σχεδιάζεις για αλλαγές. Αυτό σημαίνει ότι θα πρέπει να αναμένεις μελλοντικές αλλαγές στο λογισμικό και για αυτό θα πρέπει να σχεδιάζεις το λογισμικό έτσι ώστε αυτές οι αλλαγές να πραγματοποιούνται εύκολα. Ωστοσο, στην μέθοδο XP αυτή η αρχή έχει απορριφθεί με τη λογική ότι ο σχεδιασμός για αλλαγές είναι συχνά σπατάλη προσπάθειας. Οι αλλαγές που αναμένονται συνχά δεν πραγματοποιούνται ποτέ και οι αλλαγές που γίνονται ειναι εντελώς διαφορετικές απο αυτές που αναμένονταν.

Αρχή	Περιγραφή
Αυξανόμενος προγραμματισμός	Οι απαιτήσεις καταγράφονται σε Story cards και αποφασίζεται σε ποια έκδοση θα συμπεριληφθούν βάση του χρόνου που θα είναι διαθέσιμος ο ορισμός τους και της σχετικής τους προτεραιότητας. Οι προγραμματιστές μετατρέπουν αυτά τα σενάρια σε εργασίες.
Μικρές εκδόσεις	Το μικρότερο δυνατό σύνολο λειτουργιών είναι το πρώτο που αναπτύσσεται. Είναι συχνή η έκδοση επόμενων εκδόσεων καθώς και η αυξανόμενη προσθήκη λειτουργιών απο την πρώτη έκδοση του συστήματος.
Απλός σχεδιασμός	Ο σχεδιασμός που γίνεται είναι αρκετός ωστε να καλύπτει μόνο τις παρούσες απαιτήσεις.
Προγραμματισμός με δοκιμή πρώτα	Ένα αυτοματοποιημένο πλαίσιο εργασίας για έλεγχο μονάδας χρησιμοποιείται για την συγγραφή δοκιμών για κάθε νέο κομμάτι λειτουργικότητας, πριν αυτή η λειτουργικότητα υλοποιηθεί.
Αλλαγή κώδικα	Όλοι οι συγγραφείς κώδικα αναμένεται να αλλάζουν συνεχώς μέρη του κώδικα για να τον βελτιώσουν όσο μπορούν. Αυτό διατηρεί τον κώδικα απλό, και εύκολο προς συντήρηση.
Προγραμματισμός ανά ζεύγη	Οι προγραμματιστές δουλεύουν ανά ζεύγη, ελέγχοντας ο ενας την εργασία του αλλου και προσφέροντας βοήθεια για να πετύχουν το καλύτερο δυνατό αποτέλεσμα.
Συλλογική ιδιοκτησία	Τα ζεύγη των προγραμματιστών δουλεύουν σε όλα τα μέρη του συστήματος, ώστε να μην αναπτυχθεί μεμονομένη γνώση σε ένα κομμάτι του κώδικα και όλοι οι προγραμματιστές να γνωρίζουν καλά όλο τον κώδικα. Οποιοσδήποτε έχει την ικανότητα να αλλάζει οτιδήποτε.
Συνεχής συνένωση	Μόλις τελειώσει μια έργασία ενσωματώνεται σε όλο το σύστημα. Αφού ενσωματωθεί, πρέπει όλες οι δοκιμές μονάδων να είναι επιτυχείς.
Υποφερτός ρυθμός	Πολλές υπερωρίες δεν ειναι αποδεκτές καθώς το φαινόμενο του δικτύου συχνά μειώνει την ποιότητα του κώδικα και την αποδοτικότητα.
Διαθέσιμος πελάτης	Ένας αντιπρόσωπος των τελικών χρηστών θα πρέπει πάντα να είναι διαθέσιμος στην ομάδα του XP. Σε μία διεργασία του XP, ο πελάτης είναι μέλος την ομάδας ανάπτυξης και είναι υπεύθυνος να αποδίδει σε αυτή σωστά τις απαιτήσεις του συστήματος.

Πίνακας 3.2: Πρακτικές extreme programming

Το πρόβλημα με το σχεδιασμό χωρίς να λαμβάνουμε υπόψη τις αλλαγές σε λειτουργικότητα που μπορεί να ζητηθούν, είναι ότι όταν απαιτούνται αλλαγές, υποβαθμίζουν την δομή του κώδικα και η πραγματοποίηση των αλλαγών με τον καιρό γίνεται δυσκολότερη. Η μέθοδος XP παρακάμπτει αυτό το πρόβλημα προτείνοντας ότι ο κώδικας πρέπει να αλλάζει συνεχώς. Αυτό σημαίνει ότι η ομάδα προγραμματισμού συνεχώς ψάχνει για πιθανές βελτιώσεις του κώδικα και τις υλοποιεί αμέσως μόλις τις εντοπίσει. Για αυτό το λόγο, ο κώδικας πρέπει να είναι κατανοητός και να μπορούν να γίνουν εύκολα αλλαγές.

3.3 Συνεχής ενσωμάτωση (continuous integration)

Οι οργανισμοί σχεδόν πάντα έχουν την ανάγκη να χρησιμοποιούν επαναλαμβανόμενες και αξιόπιστες μεθόδους για να δημιουργούν τακτικές και δημόσιες εκδόσεις ενός προγράμματος. Η συνεχής ενσωμάτωση είναι η διεργασία κατά την οποία δημιουργείται μία νέα έκδοση κάθε φορά που ο προγραμματιστής στέλνει τον κώδικα του σε ένα εξυπηρετητή που ελέγχει τον κώδικα και στον οποίο υπάρχει η αποθήκη κώδικα.

Η πρακτική της συνεχούς ενσωμάτωσης αποτελεί μια θεμελιώδης αλλαγή της διεργασίας ανάπτυξης λογισμικού. Κάνει την ενσωμάτωση κώδικα, μια μέχρι πρότεινως επίπονη διαδικασία, απλή και βασικό μέρος των καθημερινών δραστηριοτήτων του προγραμματιστή. Η συνεχής ενσωμάτωση κάνει την ενσωμάτωση ένα κομμάτι του φυσιολογικού ρυθμού συγγραφής κώδικα, ένα αναπόσπαστο κομμάτι του κύκλου δοκιμής - συγγραφής - αλλαγής κώδικα. Η συνεχής ενσωμάτωση βοηθά την πρόοδο της διαδικασίας κάνοντας μικρά βήματα.

Η ενσωμάτωση θα πρέπει να συμβαίνει συνεχώς. Η συχνότητά της διαφέρει από έργο σε έργο, από προγραμματιστή σε προγραμματιστή, και από αλλαγή σε αλλαγή. Ωστόσο, ένας καλός κανόνας είναι ο προγραμματιστής πρέπει να ενσωματώνει τις αλλαγές κάθε λίγες ώρες και τουλάχιστον μια φορά την ημέρα.

Το να μάθει να ενσωματώνει κανείς τον κώδικά του τόσο συχνά απαιτεί εξάσκηση και πειθαρχία. Η ενσωμάτωση μπορεί να γίνεται κάθε φορά που μεταφράζεται ο κώδικας και όλες οι δοκιμές μονάδας εκτελούνται με επιτυχία. Η πρόκληση είναι να μάθει κανείς πώς να γράφει κώδικα, χωρίς να αποκλίνει από αυτόν την άποψη. Αν οι δοκιμές γίνονται στον κατάλληλο βαθμό λεπτομέρειας και οι αλλαγές γίνονται τακτικά, τότε σχεδόν πάντα θα είναι δυνατή η εύκολη ενσωμάτωση.

Η απόφαση για το πότε πρέπει να γίνεται η ενσωμάτωση αφορά τον έλεγχο του ρίσκου που παίρνουμε. Όταν γίνονται αλλαγές σε σημεία του κώδικα που ασχολούνται πολλά άτομα ταυτόχρονα, υπάρχει αυξημένος κίνδυνος να προκύψουν συγκρούσεις συγχώνευσης όταν γίνεται η αποστολή στην αποθήκη κώδικα. Όσο μεγαλύτερο είναι το διάστημα που μεσολαβεί και οι προγραμματιστές δεν ενσωματώνουν τον κώδικά τους, μεγαλώνουν είναι η πιθανότητες να προκύψουν συγκρούσεις καθώς και η προσπάθεια που θα χρειαστεί να επιλυθούν. Επειδή η προσπάθεια που απαιτείται για την ενσωμάτωση αυξάνεται εκθετικά σε σχέση με τα μέγεθος των χρονικών διαστημάτων ανάμεσα σε ενσωματώσεις, οι καλύτερες πρακτικές υπαγορεύουν ότι όταν γίνονται αλλαγές αυξημένου κινδύνου, ο προγραμματιστής πρέπει να ξεκινά από ένα καινούριο περιβάλλον εργασίας, να επικεντρώνεται μόνο στις απαιτούμενες αλλαγές, να προχωράει με τα μικρά και λογικά βήματα, και με την πρώτη ευκαιρία να ενσωματώνει τον κώδικά του.

Η επιτυχημένη ενσωμάτωση αποτελεί μέτρο προόδου. Παρέχει την ανάδραση ότι ο νέος κώδικας λειτουργεί κανονικά στο ενσωματωμένο περιβάλλον και συνεργάζεται επιτυχώς με τον υπόλοιπο κώδικα της εφαρμογής. Ο κώδικα στον περιβάλλον εργασίας ενός προγραμματιστή που δεν ενσωματώνεται, απλά δεν υπάρχει. Δεν αποτελεί μέρος του κώδικα της βασικής εφαρμογής και δεν μπορεί να προσπελαστεί από άλλους προγραμματιστές ή να δοκιμαστεί από τον πελάτη.

Η συνεχής ενσωμάτωση έχει πολλά πλεονεκτήματα:

- Όταν ένας έλεγχος μονάδας αποτυγχάνει ή ανακαλύπτεται κάποιο σφάλμα, οι προγραμματιστές μπορούν να επανέλθουν σε μία προηγούμενη κατάσταση χωρίς σφάλματα, και έτσι δεν χάνεται χρόνος στην αποσφαλμάτωση.
- Τα προβλήματα ενσωμάτωσης εντοπίζονται και διορθώνονται συνεχόμενα - δεν προκύπτουν κενά της τελευταίας στιγμής πριν τις ημέρες ανακοίνωσης νέας έκδοσης.
- Προειδοποιήσεις για χαλασμένο κώδικα και ασυμβατότητες.
- Προειδοποιήσεις για συγκρουόμενες αλλαγές.
- Άμεσες δοκιμές μονάδων για όλες τις αλλαγές.
- Συνεχής διαθεσιμότητα σύγχρονης έκδοσης για έλεγχο και δοκιμή.
- Η άμεση επίδραση του ελέγχου ημιτελούς ή καταστραμμένου κώδικα δρα ως κίνητρο να μάθουν να δουλεύουν αυξητικά με μικρότερους κύκλους ανάδρασης.

Όμως η συνεχής ενσωμάτωση έχει και κάποια μειονεκτήματα:

- Συχνά το κόστος συντήρησης αυξάνεται.
- Κάποιες ομάδες πιστεύουν ότι η πειθαρχία που απαιτείται για την συνεχή ενσωμάτωση μπορεί να προκαλέσει συμφορήσεις. Για να αντιμετωπιστεί αυτό, συχνά απαιτείται από τους προγραμματιστές να αλλάξουν την άποψή τους για αυτό το θέμα.
- Η άμεση σύγκρουση μίας αποστολής προκαλεί τη δημιουργία τοπικού αντίγραφου γιατί οι προγραμματιστές δεν μπορούν να καταχωρήσουν μη ολοκληρωμένο κώδικα.

ΚΕΦΑΛΑΙΟ 4

Εργαλεία ανάπτυξης λογισμικού

4.1 Ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού (IDEs)

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης είναι μία ολοκληρωμένη εφαρμογή που περιλαμβάνει όλα τα εργαλεία που χρειάζονται για την ανάπτυξη μιας εφαρμογής. Συνήθως ένα IDE αποτελείται από έναν κειμενογράφο για συγγραφή και επεξεργασία κώδικα, κάποιον μεταφραστή ή/και μεταγλωττιστή, εργαλεία για την αυτόματη εκτέλεση του κώδικα καθώς και εργαλεία για την αποσφαλμάτωσή του. Επιπλέον, συχνά περιλαμβάνεται κάποιο σύστημα διαχείρισης εκδόσεων, και εργαλεία που απλοποιούν την κατασκευή γραφικού περιβάλλοντος για την διεπαφή χρήστη (GUI). Πολλά σύγχρονα IDEs που χρησιμοποιούνται για αντικειμενοστραφή ανάπτυξη λογισμικού διαθέτουν φυλλομετρητή κλάσεων (class browser), εποπτεία αντικειμένων (object inspector) και ένα διάγραμμα ιεραρχίας κλάσεων (class hierarchy diagram).

Τα IDEs είναι σχεδιασμένα να μεγιστοποιούν την παραγωγικότητα του προγραμματιστή παρέχοντας ισχυρά συνδεδεμένα συνιστούντα μέρη με παρόμοια διεπαφή χρήστη. Αυτό σημαίνει πως ο προγραμματιστής, στην περίπτωση που χρησιμοποιούνται διαφορετικά προγράμματα ανάπτυξης, καταβάλλει λιγότερη προσπάθεια να μεταβεί από το ένα πρόγραμμα στο άλλο. Ωστόσο, επειδή ένα IDE είναι από τη φύση του πολύπλοκο, η παραγωγικότητα του προγραμματιστή αυξάνεται όσο αποκτά μεγαλύτερη γνώση και οικειότητα από την χρήση αυτού του εργαλείου.

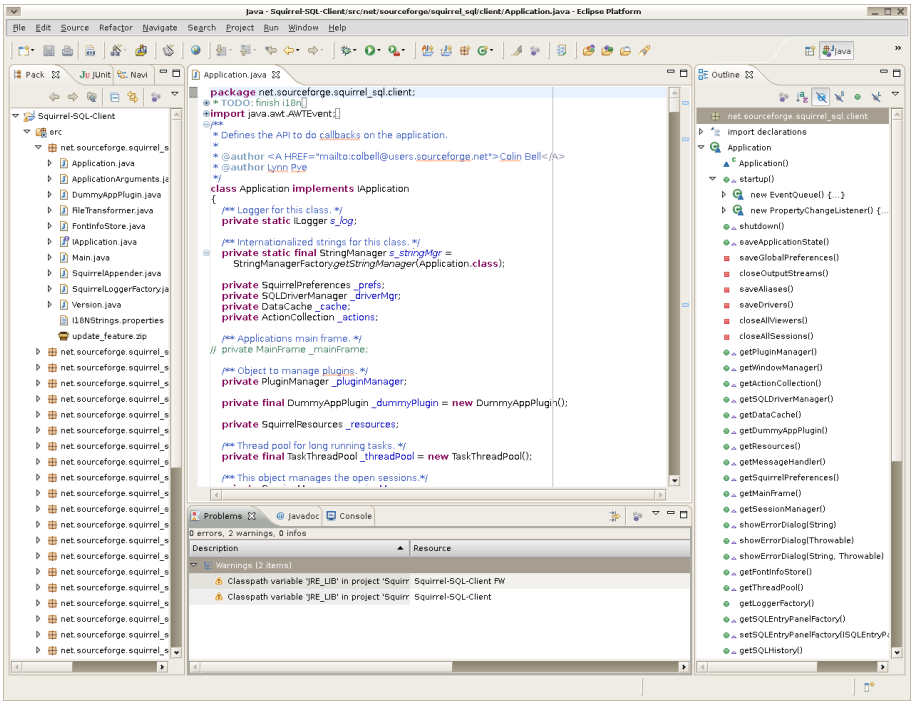
Συνήθως ένα IDE αφορά μία συγκεκριμένη γλώσσα προγραμματισμού, έτσι ώστε να παρέχει τα χαρακτηριστικά που ταιριάζουν καλύτερα σε αυτή την γλώσσα. Όμως, υπάρχουν και ολοκληρωμένα περιβάλλοντα που περιλαμβάνουν πολλές γλώσσες προγραμματισμού, όπως τα: *Eclipse*, πρόσφατες εκδώσεις του *NetBeans*, το *Microsoft Visual Studio*.

Τελευταία έχουν αναπτυχθεί αρκετά ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού. Τα δημοφιλέστερα είναι: το *Microsoft Visual Studio*, το *Eclipse*, το *Netbeanes* της Sun και το *IntelliJ Idea* της JetBrains. Κάποια από αυτά θα περιγραφούν αναλυτικότερα παρακάτω.

4.1.1 Eclipse

Το Eclipse είναι μία πλατφόρμα ανάπτυξης λογισμικού η οποία διατίθεται δωρεάν, είναι ανοιχτού κώδικα και αποτελείται από ένα IDE και ένα plug-in σύστημα

ώστε να είναι εύκολη η επέκτασή της. Υποστηρίζει πολλές γλώσσες προγραμματισμού κάποιες από τις οποίες είναι:C, C++, COBOL, Python, Perl, PHP. Η ανάπτυξή του έχει γίνει σε γλώσσα *Java* και χρησιμοποιείται για την ανάπτυξη εφαρμογών σε Java, αλλα με την χρήση διάφορων plug-ins, και σε άλλες γλώσσες όπως:C, C++, COBOL, Python, PHP.



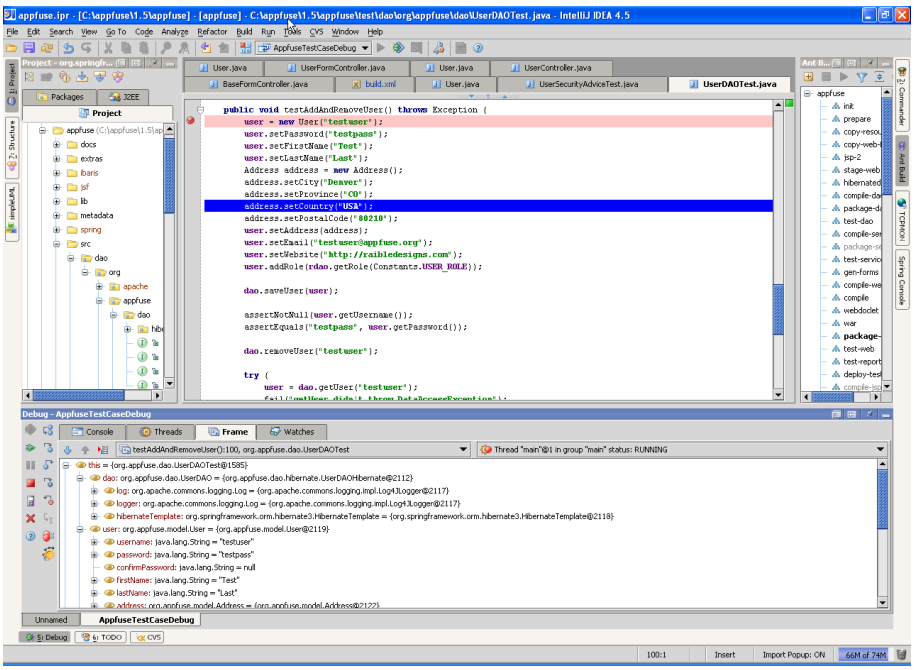
Σχήμα 4.1: Το περιβάλλον του Eclipse

Το Eclipse χρησιμοποιεί plug-ins για να παρέχει περισσότερες λειτουργίες πέρα από το σύστημα runtime, σε αντίθεση με άλλες εφαρμογές που η λειτουργικότητα τους είναι καθορισμένη. Ο μηχανισμός plug-in που χρησιμοποιεί είναι ένα ελαφρύ πλαίσιο εργασίας που αποτελείται από πολλές συνιστώσες. Εκτός από την δυνατότητα που δίνει στο Eclipse να επεκτείνει τη χρήση του και σε άλλες γλώσσες προγραμματισμού, επιτρέπει την εργασία με: στοιχειοθετικές γλώσσες όπως η LaTeX, διαδικτυακές εφαρμογές όπως η telnet, και συστήματα διαχείρισης βάσεων δεδομένων. Η αρχιτεκτονική του plug-in υποστηρίζει την συγγραφή αρχείων οποιαδήποτε κατάληξης στο περιβάλλον του Eclipse. Το Eclipse υποστηρίζει από την αρχή τη γλώσσα Java και το σύστημα διαχείρισης εκδόσεων CVS, με την υποστήριξη του Subversion να είναι διαθέσιμη από εξωτερικά plug-ins.

Το Eclipse SDK περιλαμβάνει τα Eclipse Development Tools, τα οποία παρέχουν ένα IDE με ενσωματωμένο μεταφραστή της Java και με ένα πλήρες μοντέλο των αρχείων πηγαίου κώδικα της Java. Αυτό επιτρέπει την χρήση προχωρημένων τεχνικών αλλαγής και ανάλυσης κώδικα. Επιπλέον, το IDE χρησιμοποιεί ένα workspace, το οποίο ειναι ένα σύνολο μεταδεδομένων για ένα καθορισμένο αριθμό αρχείων, επιτρέποντας τις τροποποιήσεις των αρχείων έξω από το IDE εφόσον το αντίστοιχο workspace ενημερωθεί αμέσως μετά τις αλλαγές.

4.1.2 IntelliJ

Πρόκειται για ένα εμπορικό ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού, κυρίως για Java εφαρμογές. Όμως υποστηρίζει και άλλες γλώσσες όπως: JSP, JavaScript/Flex, HTM/XHTML/CSS, XML/XSL, Ruby/JRuby, SQL. Επιπλέον, υποστηρίζονται πολλές σύγχρονες τεχνολογίες και πλαίσια εργασίας όπως Spring, Hibernate, Struts, AJAX.



Σχήμα 4.2: Το περιβάλλον του IntelliJ IDEA

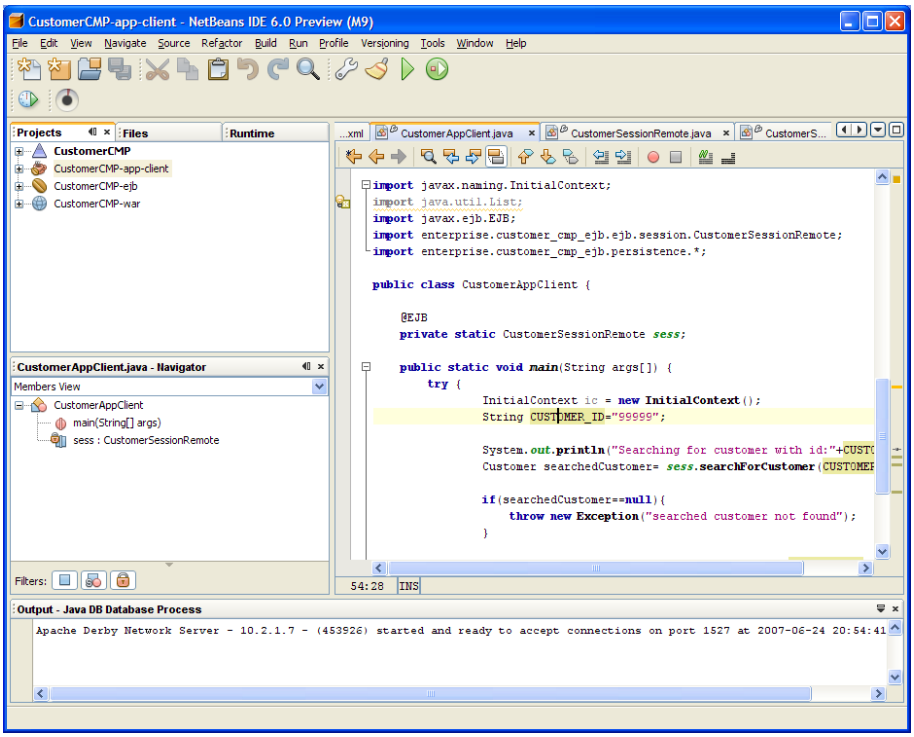
Το IntelliJ, όπως και τα περισσότερα IDEs, προσφέρει πολλά εργαλεία που διευκολύνουν τη συγγραφή κώδικα. Η αυτόματη παραγωγή κώδικα, όπως η δημιουργία μεθόδων δημιουργών σε κλάσεις, η ανάλυση και ολοκλήρωση κώδικα, ο έλεγχος λαθών κατά την διάρκεια της πληκτρολόγησης, ο αυτόματος σχολιασμός σύμφωνα με το πρότυπο της Java είναι λειτουργίες που βοηθούν τον προγραμματιστή να αυξήσει την παραγωγικότητα του και μειώνουν τον κίνδυνο λαθών.

Επιπλέον, υπάρχουν ενσωματωμένα περιβάλλοντα για το χτίσιμο της εφαρμογής (Ant/Maven), για τον έλεγχο της εφαρμογής μέσω δοκιμών μονάδας (Junit) και συστήματα διαχείρισης εκδόσεων.

Το συγκεκριμένο ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών χρησιμοποιείται και για την ανάπτυξη της εφαρμογής στα πλαίσια της διπλωματικής μου εργασίας. Η υποστήριξη των τεχνολογιών που χρησιμοποιώ και η ευκολία στην εκμάθηση του είναι οι βασικοί λόγοι επιλογής του συγκεκριμένου IDE.

4.1.3 NetBeans IDE

Το NetBeans IDE είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών που βασίζεται στην πλατφόρμα NetBeans. Η πλατφόρμα NetBeans επιτρέπει την ανάπτυξη της εφαρμογής NetBeans IDE από ένα σύνολο συστατικών μονάδων λογισμικού που ονομάζονται *modules*. Ένα module είναι ένα συμπιεσμένο αρχείο Java το οποίο περιέχει κλάσεις, που είναι προορισμένες να αλληλεπιδρούν με NetBeans



Σχήμα 4.3: Το περιβάλλον του NetBeans IDE

Open APIs, και ένα αρχείο που το χαρακτηρίζει ως module. Εφαρμογές που έχουν αναπτυχθεί πάνω σε modules μπορούν να επεκταθούν προσθέτωντας νέα modules.

Το NetBeans IDE διανέμεται δωρεάν και είναι λογισμικό ανοιχτού κώδικα. Υποστηρίζει την ανάπτυξη όλων των είδών εφαρμογών Java (Java SE, web, EJB και mobile εφαρμογές). Επιπλέον, υποστηρίζει την ανάπτυξη εφαρμογών με την χρήση του Ant, τον έλεγχο εκδόσεων, με την χρήση των CVS, Subversion, Mercurial και Clearcase, και την επεξεργασία κώδικα (refactoring).

Όλες οι λειτουργίες του IDE παρέχονται από τα modules. Κάθε module παρέχει μία απαιτούμενη λειτουργία όπως υποστήριξη ανάπτυξης εφαρμογών Java, επεξεργασία κώδικα, υποστήριξη συστημάτων διαχείρισης εκδόσεων CVS και SVN. Το NetBeans IDE παρέχει όλα τα απαιτούμενα modules για την ανάπτυξη εφαρμογών Java στο αρχικό πακέτο που κατεβάζει κανείς, το οποίο επιτρέπει στον χρήστη να αρχίσει άμεσα την ανάπτυξη μιας εφαρμογής. Τα modules επιτρέπουν την επέκταση του IDE. Νέες λειτουργίες, όπως η υποστήριξη επιπλέον γλωσσών προγραμματισμού μπορούν να προστεθούν μετά την αρχική εγκατάσταση.

4.2 Συστήματα Διαχείρισης Εκδόσεων (VCS)

Τα συστήματα διαχείρισης εκδόσεων χρησιμοποιούνται για την αποθήκευση πηγαίου κώδικα, εγγράφων τεκμηρίωσης της εφαρμογής, ακόμα και configuration files. Καθώς το λογισμικό σχεδιάζεται, αναπτύσσεται και εγκαθίσταται, είναι πολύ συνηθισμένο να υπάρχουν διαφορετικές εκδόσεις του λογισμικού σε διαφορετικά τοπικά αντίγραφα, και οι προγραμματιστές να αναπτύσσουν ταυτόχρονα αναβαθμίσεις τους. Σφάλματα και άλλα προβλήματα παρουσιάζονται σε συγκεκριμένες εκδόσεις (γιατί στην προσπάθεια αντιμετώπισης των προβλημάτων, δημιουργούνται νέα). Για

την αντιμετώπιση αυτών των προβλημάτων, είναι σημαντικό να υπάρχει η δυνατότητα να ανακτούνται διαφορετικές εκδόσεις του συστήματος οι οποίες δεν αντιμετωπίζουν τα προβλήματα που έχει η έκδοση με την οποία ασχολείται κάποιος προγραμματιστής. Επιπλέον, μερικές φορές είναι απαραίτητο να δημιουργούνται ταυτόχρονα δύο ή παραπάνω εκδόσεις του ίδιου συστήματος (όπως όταν προσπαθούν να λυθούν τα προβλήματα μιας έκδοσης αλλά δεν έχει υλοποιούνται νέες λειτουργίες, ενώ παράλληλα δημιουργούνται εκδόσεις του συστήματος με νέες λειτουργίες).

Οι παραπάνω λόγοι καθιστούν τη χρήση των συστημάτων διαχείρισης εκδόσεων απαραίτητη. Τα συστήματα VCS είναι συνήθως αυτόνομα προγράμματα τα οποία διαχειρίζονται πολλαπλές εκδόσεις ίδιων μονάδων πληροφορίας. Οι διαφορετικές εκδόσεις διαχωρίζονται μέσω κάποιου είδους αύξοντα αριθμό, συνοδεύονται από σχόλια που προσθέτει ο χρήστης και συνδέονται με τον χρήστη που πραγματοποίησε τις αλλαγές. Παρακάτω περιγράφονται τα δύο πιο γνωστά συστήματα διαχείρισης εκδόσεων, το CVS (Concurrent Versioning System) και το SVN (Subversion).

4.2.1 CVS (Concurrent Versioning System)

Πρόκειται για ένα λογισμικό ανοιχτού κώδικα που δημιουργήθηκε από τον Dick Grune τη δεκαετία του 1980. Το CVS χρησιμοποιεί την αρχιτεκτονική πελάτη - εξυπηρετητή : στον εξυπηρετητή αποθηκεύονται οι εκδόσεις του έργου και οι πελάτες συνδέονται για να λάβουν (check out) ένα πλήρες αντίγραφο του έργου, να εργαστούν πάνω σε αυτό το αντίγραφο και στο τέλος να στείλουν (chek in) τις αλλαγές τους. Συνήθως, οι πελάτες συνδέονται στον εξυπηρετητή μέσω ενός τοπικού δικτύου ή μέσω του διαδικτύου. Όμως, ο πελάτης και ο εξυπηρετητής μπορεί να βρίσκονται στο ίδιο μηχάνημα στο οποίο το CVS αποθηκεύει τις εκδόσεις ενός έργου αν οι προγραμματιστές είναι εργάζονται στο ίδιο σύστημα. Το πρόγραμμα του εξυπηρετητή "τρέχει" σε λειτουργικό Unix (αν και ο εξυπηρετητής CVSNT υποστηρίζει διάφορες εκδόσεις των Windows, της Microsoft, και Linux), ενώ οι CVS πελάτες μπορεί να τρέχουν σε οποιοδήποτε από τα πιο διαδεδομένα λειτουργικά συστήματα.

Πολλοί προγραμματιστές μπορεί να εργάζονται στο ίδιο έργο ταυτόχρονα, όπου ο καθένας επεξεργάζεται τα αρχεία του δικού του τοπικού αντίγραφου, και στέλνει τις αλλαγές στον εξυπηρετητή. Για να αποφευχθουν οι συγκρούσεις σε μέρη του κώδικα που δύο ή παραπάνω προγραμματιστές έχουν κάνει αλλαγές, ο εξυπηρετητής δέχεται τις αλλαγές μόνο από την πιο πρόσφατη έκδοση των αρχείων. Γι αυτό το λόγο οι προγραμματιστές πρέπει να κρατούν ενημερωμένο το τοπικό τους αντίγραφο ενσωματώνοντας συχνά τις αλλαγές των υπόλοιπων χρηστών. Αυτό γίνεται αυτόματα από το CVS και ζητείται η μεσολάβηση του χρήστη μόνο όταν δημιουργούνται συγκρούσεις με τον κώδικα του εξυπηρετητή που έχει αλλαχθεί από κάποιον άλλο χρήστη και το τοπικό αντίγραφο του προγραμματιστή που προσπαθεί να στείλει τις δικές του αλλαγές. Αν η αποστολή είναι επιτυχής, τότε ο αριθμός της έκδοσης όλων των αρχείων αυξάνεται , και ο CVS εξυπηρετητής αποθηκεύει σε ένα αρχείο καταγραφής την περιγραφή που έχει εισάγει ο χρήστης, την ημερομηνία αποστολής, και το όνομα του αποστολέα.

Οι προγραμματιστές μπορούν να συγκρίνουν εκδόσεις του συστήματος, να ζητήσουν ένα πλήρες ιστορικό των αλλαγών ή ένα συγκεκριμένο στιγμιότυπο του έργου βάση μιας ημερομηνίας ή ενός αριθμού έκδοσης. Επιπλέον, οι χρήστες μπορούν μέσω της εντολής *update* να ενημερώσουν το τοπικό τους αντίγραφο λαμβάνοντας την τελευταία έκδοση του έργου που υπάρχει στον εξυπηρετητή.

Το CVS χρησιμοποιεί την συμπίεση *delta (delta compression)* για την αποδοτική αποθήκευση διαφορετικών εκδόσεων του ίδιου αρχείου. Η υλοποίηση αυτή ευνοεί τα αρχεία με πολλές γραμμές (συνήθως αρχεία κειμένου) και σε ακραίες καταστάσεις μπορεί το σύστημα να αποθηκεύσει ξεχωριστά αντίγραφα για κάθε έκδοση ενός αρχείου.

4.2.2 SVN (Subversion)

Το SVN δημιουργήθηκε από την εταιρία CollabNet το 2000. Χρησιμοποιείται για την αποθήκευση εκδόσεων αρχείων πηγαίου κώδικα, ιστοσελίδων και εγγράφων τεκμηρίωσης. Ο στόχος της δημιουργίας του είναι να γίνει ο πιο συμβατός διάδοχος του ευρέως χρησιμοποιημένου CVS, έχοντας διορθώσει τα προβλήματα του CVS και έχοντας προσθέσει λειτουργίες που έλλειπαν από το CVS . Το SVN είναι πολύ γνωστό στην κοινότητα λογισμικού ανοιχτού κώδικα και χρησιμοποιείται σε πολλά έργα όπως: *Apache Software Foundation, KDE, Free Pascal, FreeBSD, GCC, Python, Django, Ruby, Mono, SourceForge.net, ExtJS* και *Tigris.org*. Το σύστημα SVN χρησιμοποιεί ως web server τον Apache HTTP Server και ως πρωτόκολλο το WebDAV/DeltaV. Επιπλέον, υπάρχει μία ανεξάρτητη διεργασία εξυπηρετητή που χρησιμοποιεί ένα προσαρμοσμένο στο TCP/IP πρωτόκολλο.

Οι αποστολές στο SVN είναι καθαρά ατομικές εργασίες. Όταν μια διαδικασία αποστολής διακοπεί, δεν προκαλεί την αστάθεια στην αποθήκη λογισμικού. Επιπλέον, τα αρχεία που αντιγράφηκαν, μετακινήθηκαν ή και διαγράφηκαν διατηρούν πλήρες ιστορικό των εκδόσεων τους. Επίσης, διατηρούνται οι εκδόσεις φακέλων, μετονομασιών και αρχείων μεταδεδομένων (χωρίς όμως να διατηρούνται ημερομηνίες για τις εκδόσεις τους). Ολόκληρες δενδρικές δομές φακέλων μπορούν να μετακινηθούν να αντιγραφούν πολύ εύκολα και γρήγορα και να διατηρηθεί το πλήρες ιστορικό των εκδόσεών τους. Άλλη μια δυνατότητα του SVN είναι η διατήρηση εκδόσεων για symbolic links.

Το κόστος της αποστολής έκδοσης ή της ενημέρωσης αυξάνεται ανάλογα με το μέγεθος των αλλαγών και όχι με το μέγεθος των δεδομένων. Αυτό οφείλεται στον τρόπο με τον οποίο αποθηκεύει τις εκδόσεις των αρχείων. Το SVN προσφέρει δύο τύπους αποθήκης δεδομένων: το σύστημα *FSFS (Fast Secure File System)* και το σύστημα *Berkeley DB*. Το FSFS αποδίδει γρηγορότερα σε φακέλους με μεγάλο αριθμό αρχείων και καταλαμβάνει λιγότερο χώρο στο δίσκο λόγω της μικρότερης καταγραφής των αλλαγών. Το SVN αντιμετωπίζει κάποιους περιορισμούς με την χρήση του Berkeley DB οδηγώντας την αποθήκη δεδομένων σε καταστάσεις απώλειας δεδομένων όταν οι προσπελάσεις στην βάση διακοπούν βίαια.

Όμως το σύστημα SVN έχει κάποιους περιορισμούς και προβλήματα. Ένα από αυτά είναι η υλοποίηση της διαδικασίας μετονομασίας φακέλων και αρχείων. Αυτή τη στιγμή υλοποιεί αυτή τη διαδικασία κάνοντας αντιγραφή στο νέο όνομα και διαγραφή του παλιού. Μόνο τα ονόματα αλλάζουν και το SVN εξακολουθεί να χρησιμοποιεί το παλιό όνομα σε παλαιότερες εκδόσεις. Ωστόσο, μπορεί να μπερδευτεί όταν αρχεία τροποποιούνται και μετακινούνται στην ίδια αποστολή. Επιπλεόν, το Subversion δεν έχει λειτουργίες της αποθήκης δεδομένων. Για παράδειγμα, μερικές φορές χρειάζεται να σβήσουμε οριστικά ιστορικές καταχωρήσεις για συγκεκριμένα δεδομένα, όμως δεν υπάρχει στο SVN τέτοια ενσωματωμένη λειτουργία ώστε να πραγματοποιείται εύκολα.

4.3 Συστήματα παρακολούθησης προβλημάτων (issue tracking systems)

Τα συστήματα παρακολούθησης προβλημάτων είναι πακέτα λογισμικού που διαχειρίζονται προβλήματα και τα διατηρούν σε μία λίστα, όπως απαιτείται από κάποιον οργανισμό. Τα συστήματα παρακολούθησης προβλημάτων χρησιμοποιούνται από οργανισμούς για να καταχωρούν, να ενημερώνουν και να επιλύουν προβλήματα που αναφέρονται από πελάτες ή ακόμα και προβλήματα που αναφέρονται από το προσωπικό της εταιρίας. Επιπλέον, ενα σύστημα παρακολούθησης προβλημάτων διατηρεί μία γνωσιακή βάση με πληροφορίες για τον κάθε πελάτη, λύσεις σε συχνά εμφανιζόμενα προβλήματα και άλλα παρόμοια δεδομένα.

Στην τεχνολογία λογισμικού, τα συστήματα παρακολούθησης προβλημάτων είναι σχεδιασμένα να συνεισφέρουν στην παροχή εγγύησης ποιότητας και για να βοηθήσουν του προγραμματιστές να αναφέρουν σφάλματα που μπορεί να υπάρχουν στον κώδικα στον οποίο εργάζονται. Το κύριο όφελος ενός συστήματος παρακολούθησης σφαλμάτων είναι να παρέχουν μια γενική εποπτεία των αιτήσεων που δημιουργούνται κατά την ανάπτυξη κώδικα (οι οποίες μπορεί να αφορούν λάθη ή βελτιώσεις του κώδικα)και την κατάστασή τους. Η λίστα με τις αιτήσεις ταξινομημένες κατά σειρά προτεραιότητας προσφέρει πολύ σημαντικά δεδομένα για την σχεδίαση των επόμενων βημάτων της εφαρμογής ή ακόμα και τις απαιτήσεις της επόμενης έκδοσης της.

Σε ένα επιχειρηματικό περιβάλλον, ένα σύστημα παρακολούθησης σφαλμάτων μπορεί να χρησιμοποιηθεί για την παραγωγή αναφορών για την παραγωγικότητα των προγραμματιστών όσο αναφορά τη διόρθωση σφαλμάτων. Ωστόσο, οι αναφορές αυτές μπορεί να οδηγήσουν σε εσφαλμένα συμπεράσματα γιατί διαφορετικά σφάλματα έχουν διαφορετικά επίπεδα σοβαρότητας και πολυπλοκότητας. Η σοβαρότητα ενός σφάλματος μπορεί να μην έχει άμεση σχέση με την πολυπλοκότητα του. Για αυτό μπορεί να υπάρχουν διαφορετικές απόψεις μεταξύ των managers και των σχεδιαστών.

Παρακάτω περιγράφονται δύο γνωστά συστήματα παρακολούθησης σφαλμάτων, το *Bugzilla* και το *JIRA*.

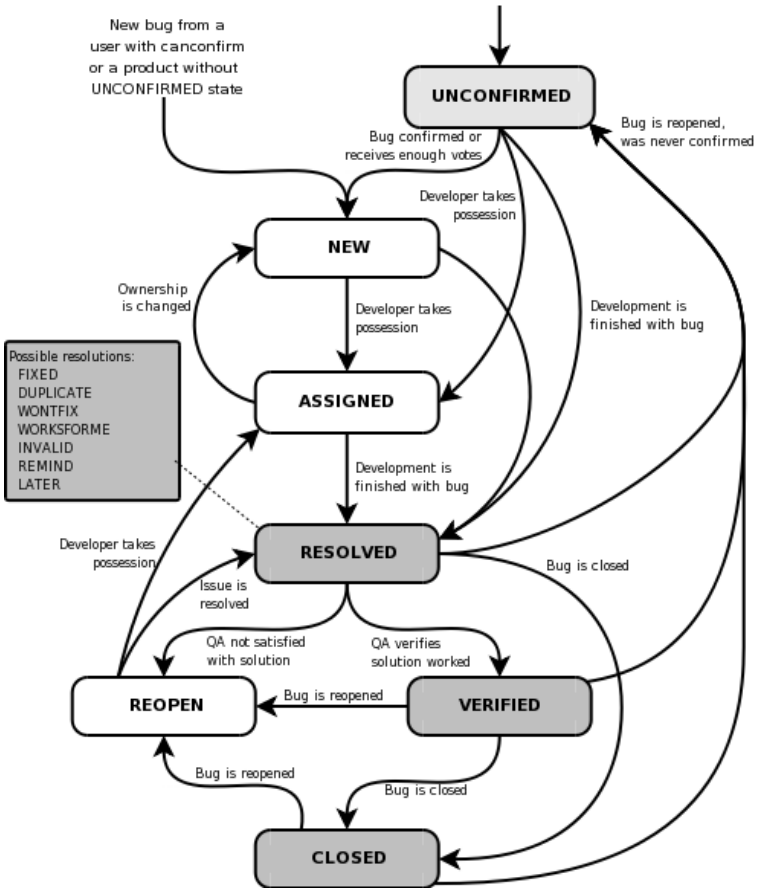
4.3.1 Bugzilla

Πρόκειται για ένα γενικού σκοπού, διαδικτυακό σύστημα παρακολούθησης σφαλμάτων και ένα εργαλείο ελέγχου το οποίο αρχικά αναπτύχθηκε και χρησιμοποιήθηκε από το έργο *Mozilla*, και κυκλοφόρησε υπό την άδεια *Mozilla Public*. Κυκλοφόρησε ως λογισμικό ανοιχτού κώδικα από την Netscape Communications το 1998, και υιοθετήθηκε από πολλούς οργανισμούς.

Το Bugzilla έχει κάποιες απαιτήσεις συστήματος για την εγκατάσταση του και αναφέρονται παρακάτω:

- Ένα συμβατό σύστημα διαχείρισης βάσης δεδομένων.
- Την κατάλληλη έκδοση του Perl 5.
- Μια συλλογή από Perl modules.
- Ένας συμβατός web server.
- Ένας συμβατός mail transfer agent, ή οποιοδήποτε SMTP server.

Επί του παρόντος, τα συστήματα βάσεων δεδομένων που είναι συμβατά είναι τα: MySQL, PostgreSQL, και Oracle. Το σύστημα Bugzilla συνήθως εγκαθίσταται σε λειτουργικό Linux και λειτουργεί χρησιμοποιώντας τον Apache HTTP Server, αλλά ο Microsoft Internet Information Services ή οποιοσδήποτε άλλος web server που υποστηρίζει CGI μπορεί να χρησιμοποιηθεί.



Σχήμα 4.4: Ο κύκλος ζωής μιας αίτησης σφάλματος

Παρόλο που ο κώδικας του Bugzilla έχει την προοπτική να το μετατρέψει σε ένα εργαλείο διαχείρισης έργων, ή σε εργαλείο διαχείρισης εργασιών, οι προγραμματιστές του Bugzilla επέλεξαν να επικεντρωθούν στον σχεδιασμό ενός συστήματος παρακολύθησης λαθών. Υπάρχουν κάποιες απαραίτητες σχεδιαστικές απαιτήσεις που περιλαμβάνουν τα εξής:

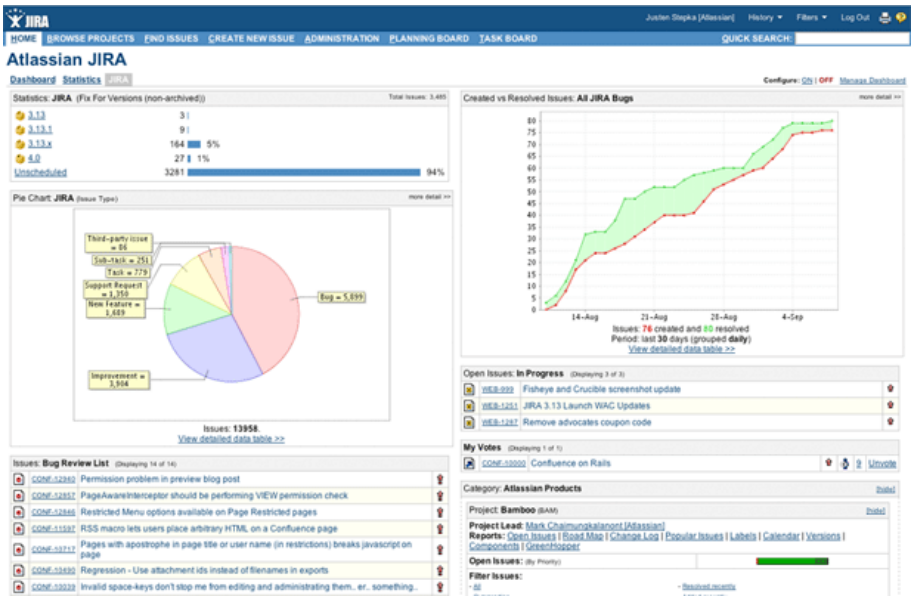
- τη δυνατότητα να τρέχει με εργαλεία δωρεάν διάθεσης και ανοιχτού κώδικα. Ενώ η ανάπτυξη του Bugzilla περιλαμβάνει την προσπάθεια υποστήριξης εμπορικών βάσεων δεδομένων, εργαλείων και λειτουργικών συστημάτων, δεν υπάρχει η πρόθεση αυτό να γίνει εις βάρος των εργαλείων ανοιχτού κώδικα.

- η διατήρηση της ταχύτητας και της αποτελεσματικότητας με οποιοδήποτε κόστος. Ένα από τα χαρακτηριστικά του Bugzilla που έλκει τους προγραμματιστές είναι η ταχύτητα του και η μη επιβαρυμένη υλοποίηση του, ελαχιστοποιώντας όσο είναι δυνατό τις κλήσεις προς τη βάση δεδομένων. Οι προσπελάσεις στην βάση προσπαθούν να είναι όσο το δυνατό λιγότερο απαιτητικές και αποφεύγεται η δημιουργία βαριάς HTML.

- η χρήση εισιτήριων για τα σφάλματα. Τα σφάλματα στον κώδικα μπορούν να σταλούν από οποιονδήποτε και να ανατεθούν σε έναν συγκεκριμένο προγραμματιστή. Διάφορες ενημερώσεις της κατάστασης του σφάλματος επιτρέπονται, μαζί με σημειώσεις από τον χρήστη που εντόπισε το σφάλμα καθώς και παραδείγματα στα οποία εμφανίζεται. Στην πράξη, τα περισσότερα έργα Bugzilla που επιτρέπουν την δημόσια καταχώρηση σφαλμάτων, αναθέτουν όλα τα σφάλματα σε κάποιον, ο οποίος έχει καθήκον την ανάθεση των σφαλμάτων σε προγραμματιστές και τον ορισμό του επιπέδου προτεραιότητας.

4.3.2 JIRA

Το JIRA είναι ένα εμπορικό πρόγραμμα για επιχειρήσεις, αναπτύχθηκε από την εταιρία Atlassian και συνήθως χρησιμοποιείται ως σύστημα παρακολούθησης σφαλμάτων κώδικα και ως σύστημα διαχείρισης έργων. Χρησιμοποιείται σε πολλά έργα λογισμικού ανοιχτού κώδικα με πάνω από 12.000 χρήστες σε περισσότερες από 100 χώρες. Το JIRA χρησιμοποιείται για την παρακολούθηση λαθών και σε πολλά μεγάλης κλίμακας ανοιχτού κώδικα, δημόσια έργα όπως το *Secondlife* της Linden Lab's, ένα παρόμοιο έργο που ονομάζεται *Opensimulator* και σε πολλά άλλα έργα ευρέως γνωστά στην κοινότητα των προγραμματιστών. Το σύστημα αυτό είναι πολύ χρήσιμο για μεγάλα ή/και δημόσια έργα.



Σχήμα 4.5: Το περιβάλλον του JIRA

Το JIRA επιτρέπει την απεικόνιση του κύκλου ζωής ενός προβλήματος στις επιχειρηματικές διεργασίες. Επιπλέον, προσφέρει αρκετούς τρόπους για την παροχή σε πραγματικό χρόνο πληροφοριών σχετικές με προβλήματα ή ροές εργασίας, στην κατάλληλη μορφή.

Το JIRA είναι γραμμένο σε Java και για απομακρυσμένη κλήση μεθόδων (Remote Procedure Calls) υποστηρίζει τα: SOAP, XML RPC και ένα Java API. Επιπλέον συνεργάζεται με αρκετά συστήματα διαχείρισης εκδόσεων όπως: Subversion, CVS, Clearcase, Visual SourceSafe, Mercurial, και Perforce.

Το JIRA έχει ένα σύστημα plugin το οποίο μέσω του JIRA API, δίνει τη δυνατότητα στους προγραμματιστές να ενσωματώνουν έναν μεγάλο αριθμό λειτουργιών που

δημιουργούνται είτε από την κοινότητα του JIRA είτε από τρίτους. Επιπλέον συνεργάζεται και με γνωστά IDEs όπως το Eclipse και το IntelliJ IDEA χρησιμοποιώντας το Atlassian IDE Connector ένα έργο ανοιχτού κώδικα από την Atlassian.

Πολλές ομάδες προγραμματιστών έχουν υιοθετήσει το JIRA στα έργα τους. Μερικές από αυτές είναι οι: JBoss, Spring Framework, OpenSymphony, Fedora Commons, και Codehaus XFire.

4.4 Συστήματα διαχείρισης έργων λογισμικού (project management systems)

4.4.1 SourceForge

Το SourceForce είναι μία web-based αποθήκη κώδικα. Είναι ένα κεντρικοποιημένο σύστημα στο οποίο οι προγραμματιστές μπορούν να ελέγχουν και να διαχειρίζονται την ανάπτυξη λογισμικού ανοιχτού κώδικα. Η εταιρία SourceForge Inc. χειρίζεται την ιστοσελίδα στην οποία τρέχει μία έκδοση του SourceForge Enterprise Edition. Το SourceForce φιλοξενεί πάνω από 180.000 έργα και έχει πάνω από 1,9 εκατομμύρια εγγεγραμμένους χρήστες, παρόλο που δεν είναι όλοι ενεργοί. Σύμφωνα με έρευνα της Compete.com το SourceForce.net δέχεται πάνω από 28 εκατομμύρια ετησίως.

Σχήμα 4.6: Η ιστοσελίδα του SourceForge

Οι προγραμματιστές έχουν πρόσβαση σε ένα κεντρικοποιημένο χώρο αποθήκευσης και σε εργαλεία διαχείρισης έργων. Όμως το SourceForce είναι ευρύτερα γνωστό για την παροχή συστημάτων διαχείρισης εκδόσεων όπως CVS, SVN, Git και Mercurial. Επιπλέον, προσφέρει wikis, ανάλυση και μετρικές κώδικα, πρόσβαση σε μία βάση δεδομένων MySQL και μοναδικές διευθύνσεις sub-domain όπως: http://project-name.sourceforge.net.

4.4.2 GForge

Το GForge είναι μία σύγχρονη και επεκτάσιμη πλατφόρμα η οποία διαθέτει μία εύχρηστη διεπαφή χρήστη και μπορεί να συνδέσει ένα μεγάλο σύνολο εργαλείων, τα οποία μπορεί να είναι εργαλεία για την διαχείριση πηγαίου κώδικα μέχρι και εξαιρετικά προσαρμόσιμοι trackers, εργαλεία διαχείρισης εργασιών, εργαλεία διαχείρισης εγγράφων, forums, mailing lists. Όλα αυτά τα εργαλεία ελέγχονται από ένα κεντρικοποιημένο σύστημα και συντηρούνται αυτόματα από το σύστημα αυτό.

Σχήμα 4.7: Λειτουργίες του GForge

Το GForge αποτελεί παρακλάδι του συστήματος SourceForge. Κυκλοφορεί υπό την άδεια GNU General Public License. Εκτός από τα εργαλεία που μπορούν να ενσωματωθούν στο GForge, προσφέρει hosting σε έργα, έλεγχο εκδόσεων μέσω CVS και Subversion, σύστημα παρακολούθησης σφαλμάτων στον κώδικα, και μετάδοση μηνυμάτων.

Το μόνο που χρειάζεται να κάνει κάποιος για να χρησιμοποιήσει το GForge είναι: αρχικά, να εγγραφεί στο σύστημα με τη χρήση συνθηματικού και κωδικού πρόσβασης, και μόλις επιβεβαιωθούν τα στοιχεία του να καταχωρίσει το έργο του. Όταν το έργο εγκριθεί για καταχώρηση, ο διαχειριστής το μόνο που κάνει είναι να επισκέπτεται την ιστοσελίδα και στην σελίδα διαχειριστή (admin) του παρέχεται ένας μεγάλος αριθμός υπηρεσιών για την διαχείριση του έργου του. Αυτές είναι οι παρακάτω:

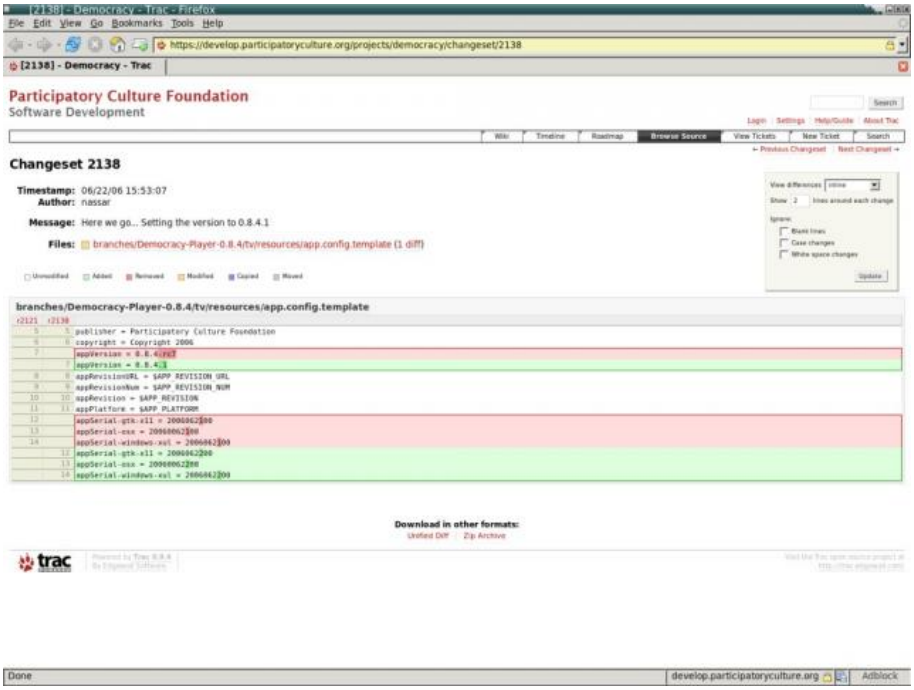
- Δημιουργία νέων tracker ή επεξεργασία παλιών.
- Αλλαγή ρυθμίσεων των επιλογών roles και Observer που παρέχουν άδεια σε μέρη του έργου και ρυθμίζουν το επίπεδο πρόσβασης σε αυτά.
- Προσθήκη μελών στο έργο.
- Δημιουργία νέων εργασιών για το έργο χρησιμοποιώντας τον διαχειριστή εργασιών του χρήστη και υπολογισμός του εκτιμώμενου χρόνου ολοκλήρωσής του με τη βοήθεια του Microsoft Project.
- Σύνδεση του κώδικα που στέλνει ο προγραμματιστής με κάποια συγκεκριμένη εργασία.

- Καταχώρηση ατελειών μετά από την κυκλοφορία κάποιας έκδοσης του συστήματος.
- Αποστολή αλλαγών στον κώδικα και σύνδεση τους με τις ατέλειες που έχουν καταχωρηθεί.

Το GForge μπορεί να ενσωματωθεί στο Eclipse, το Visual Studio και το Microsoft Project με την χρήση ενός plugin διατίθεται μαζί με το GForge.

4.4.3 Trac

Το TRAC είναι ένα web-based, ανοιχτού κώδικα σύστημα, με ενσωματωμένο wiki, το οποίο χρησιμοποιείται για την διαχείριση της ανάπτυξης ενός έργου, καθώς και την παρακολούθηση προβλημάτων. Ακολουθεί μία μινιμαλιστική προσέγγιση για web-based διαχείριση ανάπτυξης έργων.



Σχήμα 4.8: Το περιβάλλον του Trac

Η δημιουργία αυτού του συστήματος είναι εμπνευσμένη από το CVSTrac, και στην αρχή ονομάστηκε *svntrac* λόγο της ικανότητας του να αλληλεπιδρά με το Subversion. Το TRAC έχει γραφτεί στη γλώσσα προγραμματισμού Python. Μέχρι τα μέσα του 2005 κυκλοφορούσε υπό την άδεια GNU General Public License, αλλά από την έκδοση 0.9 κυκλοφορεί υπό την άδεια modified BSD license. Και οι δύο άδειες είναι ελεύθερης διάθεσης.

Το Trac επιτρέπει την διασύνδεση μεταξύ μιας βάσης δεδομένων σφαλμάτων, ενός συστήματος διαχείρισης εκδόσεων και του περιεχομένου ενός wiki. Επίσης, χρησιμοποιείται και ως διαδικτυακή διεπαφή συστημάτων διαχείρισης εκδόσεων όπως: Subversion, Git, Mercurial, Bazaar και Darcs. Περιλαμβάνει και άλλες λειτουργίες, μερικές από αυτές περιγράφονται παρακάτω:

- Υποστήριξη πολλαπλών έργων.

- Συστημα ticket (για την παρακολούθηση σφαλμάτων, τη διαχείριση λειτουργιών).
- Παραμετροποιημένες αναφορές.
- Χρονοδιάγραμμα όλων των πρόσφατων δραστηριοτήτων.
- RSS Feed
- Επεκτασιμότητα (μέσω των plugins της Python).

Είναι καταγεγραμμένοι πάνω από 450 οργανισμοί που χρησιμοποιούν το Trac. Ανάμεσα σε αυτούς είναι το εργαστήριο Jet Propulsion της NASA, που το χρησιμοποιεί για την διαχείριση διάφορων deep space/near space έργων, και το WebKit, ένα εργαλείο που χρησιμοποιείται στον φυλλομετρητή Safari της Apple.

4.5 Μετρικές κώδικα (code metrics)

Οι μετρικές κώδικα χρησιμοποιούνται για την αξιολόγηση του κώδικα μιας εφαρμογής οι οποίες συνήθως περιλαμβάνουν μετρήσεις για τα παρακάτω:

- Αριθμός γραμμών κώδικα, που χρησιμοποιείται για την μέτρηση του μεγέθους του λογισμικού που αναπτύσσεται.
- Κυκλωματική πολυπλοκότητα (Cyclomatic Complexity), που χρησιμοποιείται για την μέτρηση της πολυπλοκότητας ενός προγράμματος. Αυτό γίνεται υπολογίζοντας τον αριθμό των γραμμικά ανεξάρτητων μονοπατιών του πηγαίου κώδικα του προγράμματος.
- Σφάλματα ανά γραμμή κώδικα.
- Κάλυψη κώδικα, που χρησιμοποιείται για να υπολογιστεί το μέγεθος του κώδικα που έχει δοκιμαστεί.
- Ταίριασμα κλάσεων, όπου μετρούνται οι εξαρτήσεις μεταξύ κλάσεων και αυτό γίνεται μέσω των παραμέτρων μιας κλάσης, των κλήσεων των μεθόδων της και των υλοποιήσεων interface.

Έχουν αναπτυχθεί αρκετά εργαλεία τα οποία χρησιμοποιούνται για την **στατική ανάλυση κώδικα (static code analysis)**, τα οποία διαβάζουν και αναλύουν τον κώδικα ελέγχοντας για τυχόν λάθη, παραλήψεις ή μη αποδοτικές τεχνικές προγραμματισμού. Ένα από αυτά είναι το ***PMD*** το οποίο χρησιμοποιώ και για την ανάλυση του κώδικα της εφαρμογής που αναπτύσσω στα πλαίσια της διπλωματικής εργασίας μου.

4.5.1 PMD

Το PMD είναι ένα στατικός αναλυτής πηγαίου κώδικα γλώσσας προγραμματισμού Java. Εκτός από τον υπολογισμό κάποιων μετρικών κώδικα, βοηθάει στη συγγραφή κώδικα χρησιμοποιώντας κάποια πρότυπα. Αυτό βοηθάει στην διατήρηση της ομοιογένειας του κώδικα, κυρίως σε περιπτώσεις όπου μια εφαρμογή αναπτύσσεται από διαφορετικές ομάδες ατόμων.

Χρησιμοποιείται για την αναγνώριση προβλημάτων που μπορεί να υπάρχουν στον κώδικα όπως:

- Πιθανά σφάλματα - Κενά try/catch/finally/switch blocks.
- Νεκρός κώδικας - Μη χρησιμοποιούμενες τοπικές μεταβλητές, παράμετροι ή κρυφές μέθοδοι.
- Κενές if/while δηλώσεις.
- Πολύπλοκες εκφράσεις.
- Κλάσεις με υψηλές μετρήσεις κυκλωματικής πολυπλοκότητας.
- Πολλαπλά αντίγραφα του ίδιου κώδικα - Αντιγραμμένος κώδικα μπορεί να συνεπάγεται με αντιγραμμένα σφάλματα, καθώς επίσης και την μείωση της ευκολίας συντήρησης.

Συνήθως τα λάθη που ανιχνεύει το PMD δεν είναι πραγματικά λάθη, αλλά μη αποδοτικός κώδικας, και πολλές φορές η εφαρμογή εξακολουθεί να τρέχει ακόμα και αν δεν διορθωθούν.

Μέρος ΙΙΙ

Τεχνολογίες ανάπτυξης εφαρμογών

ΚΕΦΑΛΑΙΟ 5

Το περιβάλλον Hibernate

Στην ανάπτυξη εφαρμογών στις οποίες απαιτείται η αποθήκευση δεδομένων, η προσέγγιση που θα ακολουθηθεί για τη διαχείριση τους είναι πολύ καθοριστική για τον σχεδιασμό της εφαρμογής. Λόγο του γεγονότος ότι η αποθήκευση δεδομένων δεν είναι κάτι καινούριο για Java εφαρμογές, θα περίμενε κανείς ότι ο προγραμματιστής θα είχε την δυνατότητα να κάνει μια επιλογή ανάμεσα σε πολλές, καθιερωμένες λύσεις αποθήκευσης δεδομένων, κάτι που δυστυχώς δεν συμβαίνει.

Για πολλά χρόνια, η αποθήκευση δεδομένων (*persistance*) αποτελούσε ένα πολύ σημαντικό θέμα διαφωνίας στην κοινότητα των προγραμματιστών της Java. Πολλοί προγραμματιστές δεν συμφωνούσαν ούτε με το θέμα του προβλήματος. Είναι η αποθήκευση δεδομένων ένα πρόβλημα που έχει λυθεί από την σχεσιακή τεχνολογία (relational technology) και τις επεκτάσεις της, όπως οι αποθηκευμένες διαδικασίες, ή αποτελεί ένα πιο σημαντικό πρόβλημα το οποίο πρέπει να αντιμετωπιστεί με την χρήση ειδικών μοντέλων της, όπως τα Enterprise Java Beans (EJB); Πρέπει να γράφεται κώδικας ακόμα και για τις πιο βασικές ενέργειες της SQL όπως create, read, update, delete - (GRUD) ή πρέπει να γίνονται αυτόματα; Πως επιτυγχάνεται η μεταφερσιμότητα εάν κάθε σύστημα διαχείρισης βάσης δεδομένων χρησιμοποιεί τη δική του διάλεκτο; Πρέπει η SQL να εγκαταλειφθεί εντελώς και να υιοθετηθεί ένα διαφορετικό σύστημα διαχείρισης δεδομένων; Οι διαφωνίες συνεχίζονται ακόμα. Παρ' όλα αυτά, για την επίλυση του προβλήματος αναπτύχθηκε μια προγραμματιστική τεχνική με ευρεία αποδοχή που ονομάζεται ***object/reational mapping (ORM)***. Η ***Hibernate*** είναι μία ανοιχτού κώδικα υλοποίηση της τεχνικής ORM.

Για την διατήρηση δεδομένων (persistance) στην Java έχει αναπτυχθεί ένα πλαίσιο εργασίας για την αντιστοίχηση των οντοκεντρικών μοντέλων (object-oriented models) σε σχήματα παραδοσιακών σχεσιακών βάσεων δεδομένων. Αυτό το πλαίσιο εργασίας ονομάζεται *Java Persistence API (JPA)* και είναι μέρος της αναβαθμισμένης προδιαγραφής Enterprise Java Beans (EJB) 2.0. Η Hibernate υλοποιεί το JPA και υποστηρίζει όλες τις προτυποποιημένες αντιστοιχίσεις, ερωτήματα στην βάση δεδομένων, και APIs.

5.1 Τι είναι η Hibernate;

Η Hibernate είναι μια ολοκληρωμένη λύση στο πρόβλημα της διαχείρισης δεδομένων, μεσολαβώντας στις αλληλεπιδράσεις της εφαρμογής με την σχεσιακή βάση δεδομένων, επιτρέποντας στον προγραμματιστή να ασχοληθεί με μόνο με τις λειτουργίες της. Ο κύριο σκοπός της Hibernate είναι η δημιουργία μιας διεπαφής μεταξύ

των σχεσιακών βάσεων δεδομένων και του αντικειμενοστραφή προγραμματισμού. Για την επίτευξη αυτού του σκοπού, δημιουργούνται αντιστοιχίες μεταξύ των εννοιών του αντικειμενοστραφή προγραμματισμού, όπως οι συσχετίσεις, η κληρονομικότητα και ο πολυμορφισμός, και των πινάκων και σχέσεων μεταξύ αυτών μιας σχεσιακής βάσης. Με αυτόν τον τρόπο, ο προγραμματιστής βλέπει τελικά μια αντικειμενοστραφή βάση δεδομένων, ενώ στην πραγματικότητα χρησιμοποιεί μία σχεσιακή.

Έτσι, ο προγραμματιστής χρησιμοποιεί τα αντικείμενα της συγκεκριμένης εφαρμογής, τα τροποποιεί με τον τρόπο που ορίζει η εφαρμογή και τα αποθηκεύει (τροποποιεί, διαγράφει, και αναζητά) στην βάση δεδομένων ως αντικείμενα. Έτσι, η Ηιβερνατε, γνωρίζοντας την αντιστοιχία μεταξύ βάσης και λογικής της εφαρμογής, αναλαμβάνει να κατασκευάσει τον κατάλληλο κώδικα τον οποίο στέλνει τελικά στη βάση δεδομένων. Έπειτα, τα αποτελέσματα που επιστρέφονται από τη βάση δεδομένων στην Hibernate δίνονται στην εφαρμογή ως αντικείμενα. Πρόκειται, δηλαδή, για ένα ενδιάμεσο επίπεδο μεταξύ εφαρμογής και βάσης δεδομένων.

5.2 Πλεονεκτήματα της Hibernate

Υπάρχουν πολλοί λόγοι που η χρήση της Hibernate στην ανάπτυξη μίας εφαρμογής στην οποία απαιτείται αλληλεπίδραση με μία βάση δεδομένων αποτελεί μία πάρα πολύ καλή επιλογή. Οι κυριότεροι λόγοι περιγράφονται παρακάτω:

- ***Αύξηση της παραγωγικότητας***: Συνήθως, η συγγραφή κώδικα που σχετίζεται με την αποθήκευση δεδομένων είναι το ανιαρό κομμάτι στην ανάπτυξη μίας Java εφαρμογής. Η Hibernate εξαλείφοντας το μεγαλύτερο μέρος αυτής της εργασίας, επιτρέπει στον προγραμματιστή να ασχοληθεί με το λειτουργικό κομμάτι της εφαρμογής. Ο προγραμματιστής ενδιαφέρεται μόνο για τα αντικείμενα. Αυτά αποθηκεύονται στη βάση και ανακτώνται από αυτήν με ελάχιστο κόπο. Αυξάνει σε μεγάλο βαθμό την παραγωγικότητα του προγραμματιστή.

- ***Βελτίωση συντηρησιμότητας***: Λιγότερες γραμμές κώδικα κάνουν το σύστημα πιο ευκολονόητο, γιατί ο κώδικας είναι επικεντρωμένο στο λειτουργικό κομμάτι. Το πιο σημαντικό είναι ότι ένα σύστημα με λιγότερες γραμμές κώδικα είναι πιο εύκολο να τροποποιηθεί. Η αυτοματοποιημένη ORM αποθήκευση μειώνει σημαντικά τις γραμμές κώδικα. Όμως, ο αριθμός των γραμμών αποτελεί έναν αμφισβητήσιμο τρόπο μέτρησης της πολυπλοκότητας της εφαρμογής. Επιπλέον, σε συστήματα που ο κώδικας για την αποθήκευση δεδομένων έχει γραφτεί από τον προγραμματιστή (hand - coded), υπάρχει μία αναπόφευκτη σχέση μεταξύ της σχεσιακής αναπαράστασης και του μοντέλου αντικειμένου που αναπαριστά ένα δεδομένο του συστήματος. Αλλαγές στο μοντέλο έχει άμεση επιρροή στην αναπαράσταση στη βάση και το αντίστροφο και πολλές φορές, γίνονται συμβιβασμοί στο σχεδιασμό της μιας αναπαράστασης για την ύπαρξης της άλλης. (Στην πράξη, οι συμβιβασμοί γίνονται στο μοντέλο αντικειμένων.) Η Hibernate προσφέρει ένα ενδιάμεσο επίπεδο μεταξύ των δύο μοντέλων, επιτρέποντας τη χρήση της αντικειμενοστρέφιας με έναν πιο κομψό τρόπο και μεταφέροντας τα δεδομένα από το ένα μοντέλο στο άλλο με τις μικρότερες δυνατές αλλαγές.

- ***Βελτίωση απόδοσης συστήματος***: Η βελτίωση αυτή, δύσκολα θα επιτυγχανόταν αν ο κώδικας που εκτελούσε τις επιθυμητές λειτουργίες σχετικές με τη βάση

δεδομένων ήταν γραμμένος από τον προγραμματιστή (hand-coding). Η Hibernate δημιουργεί πολύ αποδοτικά ερωτήματα, πράγμα που διασφαλίζει την απόδοση σε πολλές περιπτώσεις. Επιπλέον, με τη χρήση μιας έξυπνης και αποδοτικής πολιτικής πρώτου και δεύτερου επιπέδου caching, επιτυγχάνεται μεγάλη κλιμακωσιμότητα. Δίνει τη δυνατότητα στον προγραμματιστή να επιλέξει το επίπεδο caching που επιθυμεί, όντας πολύ έξυπνη στην πολιτική των write-backs. Επίσης, μπορεί να συνδυαστεί πολύ καλά με κάποια από τα σημαντικότερα λογισμικά σασηινγ τόσο ελεύθερου λογισμικού όσο και εμπορικά πακέτα.

- ***Μεταφερσιμότητα***: Η Hibernate διαχωρίζει την εφαρμογή απ την υποκείμενη βάση δεδομένων και τη διάλεκτό της. Υποστηρίζει έναν αριθμό από διαφορετικές βάσεις δεδομένων και έτσι προσφέρει κάποιου επιπέδου μεταφερσιμότητα της εφαρμογής. Δεν θα πρέπει να περιμένει κανείς να ακολουθείται το *"write-once/run-anywhere"*, γιατί οι δυνατότητες κάθε βάσης δεδομένων διαφέρουν, και για να πετύχει κανείς πλήρη μεταφερσιμότητα θα πρέπει να θυσιάσει κάποιες δυνατότητες που χαρακτηρίζουν κάθε βάση δεδομένων ως ισχυρή και αποδοτική. Ένας ακόμα λόγος που η Hibernate προσφέρει μεταφερσιμότητα της εφαρμογής είναι το γεγονός ότι οι κλάσεις που αντιστοιχίζονται είναι απλές κλάσεις Java (Plain Old Java Objects - POJOs) με αποτέλεσμα να μη χρειάζεται να είναι απόγονοι μιας πολύπλοκης υποχρεωτικής δομής.

5.3 Ανάπτυξη εφαρμογών σε Hibernate

Για την ανάπτυξη μιας εφαρμογής χρησιμοποιώντας την Hibernate ο προγραμματιστείς πρέπει να ακολουθήσει τα παρακάτω βήματα:

1. Επιλογή της διαδικασίας ανάπτυξης της εφαρμογής
2. Δημιουργία της υποδομής της εφαρμογής
3. Ανάπτυξη του κώδικα της εφαρμογής και του κώδικα αντιστοίχισης (mapping)
4. Εκτέλεση εφαρμογής

5.3.1 Επιλέγοντας μία διαδικασία ανάπτυξης της εφαρμογής

Σε κάποια έργα, η ανάπτυξη μιας εφαρμογής καθοδηγείται από την ανάλυσης της επιχειρηματικής δομής σε αντικειμενοστραφείς όρους που γίνεται από τους προγραμματιστές. Δηλαδή, η ανάπτυξη της εφαρμογής επιρρεάζεται αρκετά από το υπάρχον σχεσιακό μοντέλο, το οποίο προέρχεται είτε από μια ήδη υπάρχουσα βάση δεδομένων, είτε από ένα καινούριο σχήμα το οποίο εχει σχεδιαστεί από έναν ειδικό σχεδιαστή. Έτσι ο προγραμματιστής πρέπει να πάρει κάποιες αποφάσεις σε σχέση με τον τρόπο ανάπτυξης της εφαρμογής. Οι πιο κοινές μέθοδοι ανάπτυξης είναι οι παρακάτω:

- *Από πάνω προς τα κάτω (Top down)* — Σε αυτόν τον τρόπο ανάπτυξης, ο προγραμματιστής ξεκινάει την υλοποίηση, με δεδομένο το μοντέλο της εφαρμογής (δηλαδή γνωρίζει τις κλάσεις του μοντέλου και τις μεταξύ τους συσχετίσεις), το

οποίο είναι ήδη υλοποιημένο και (ιδανικά) έχει πλήρη ελευθερία σε ό,τι αφορά τη δημιουργία του σχήματος της βάσης δεδομένων. Έτσι ο προγραμματιστής πρέπει να δημιουργήσει τα μεταδεδομένα αντιστοίχισης, είτε με την χρήση XML αρχείων είτε μέσω επιπρόσθετου κώδικα περιγραφής στον ήδη υπάρχοντα πηγαίο κώδικα (annotations), και ύστερα μπορεί να αφήσει την Hibernate να δημιουργήσει το σχήμα της βάσης με την χρήση του εργαλείου *hbm2ddl*. Όταν δεν υπάρχει κάποιο σχήμα βάσης, ο τρόπος που περιγράφηκε είναι ο πιο βολικός τρόπος δημιουργίας του.

- *Από κάτω προς τα πάνω (Bottom up)* — Αντίστροφα, στον bottom up τρόπο ανάπτυξης, το σχήμα της βάσης και έχει ήδη αποτυπωθεί και σε κάποια βάση δεδομένων. Σε αυτή την περίπτωση, ο ευκολότερος τρόπος να συνεχίσει ο προγραμματιστής είναι να χρησιμοποιήσει κάποιο εργαλείο αντίστροφης τεχνολογίας (ρεερσε-ενγινεερινγ) για να εξάγει τα δεδομένα από το σχήμα της βάσης. Αυτά τα μεταδεδομένα μπορούν να χρησιμοποιηθούν για την παραγωγή των XML αρχείων αντιστοίχισης, χρησιμοποιώντας το εργαλείο *hbm2hbmxml*. Με το εργαλείο *hbm2java*, τα μεταδεδομένα χρησιμοποιούνται για την παραγωγή κλάσεων και αντικειμένων για τον χειρισμό των δεδομένων. Ωστόσο, από τα μεταδεδομένα δεν μπορούν να εξαχθούν όλες οι συσχετίσεις των κλάσεων και άλλες Java μεταπληροφορίες, για αυτό ο προγραμματιστής θα πρέπει να εκτελέσει κάποιες εργασίες χειροκίνητα.

- *Από τη μέση προς τα έξω (Middle out)* — Τα μεταδεδομένα αντιστοίχισης της Hibernate που υπάρχουν σε αρχεία XML παρέχουν αρκετές πληροφορίες για τη δημιουργία του σχήματος βάσης και τη δημιουργία πηγαίου κώδικα της Java για το επίπεδο αποθήκευσης της εφαρμογής. Επιπλέον, τα αρχεία αντιστοίχισης XML είναι περιεκτικά, και για αυτό το λόγο κάποιοι σχεδιαστές και προγραμματιστές προτιμούν αυτή την τεχνική ανάπτυξης της εφαρμογής όπου από τα αρχεία αντιστοίχισης, και με κατάλληλα εργαλεία, παράγεται το σχήμα της βάσης, καθώς και ο αντίστοιχος Java κώδικα.

- *Συνάντηση στη μέση (Meet in the middle)* — Το πιο δύσκολο σενάριο είναι αυτό στο οποίο συνδυάζονται ήδη υπάρχουσες κλάσεις σε Java και σχήμα βάσης. Σε αυτή την περίπτωση, η βοήθεια που προσφέρουν τα εργαλεία της Hibernate είναι μικρή. Ο προγραμματιστής καλείται να γράψει ο ίδιος κώδικα αντιστοίχισης σε XML αρχεία και κάποιες φορές να αλλάξει τον πηγαίο κώδικα ή και το σχήμα βάσης έτσι ώστε να επιτευχθεί ο συνδυασμός τους.

Στη συνέχεια, περιγράφεται πιο αναλυτικά ο τρόπος δημιουργίας μιας εφαρμογής με τη χρήση της Hibernate. Στο παράδειγμα που ακολουθεί έχει γίνει η υπόθεση ότι η διαδικασία της ανάπτυξης που ακολουθείται είναι η *από πάνω προς τα κάτω*.

5.3.2 Στήνοντας την εφαρμογή

Αρχικά, υποθέτουμε έχουμε στην διάθεσή μας τη νεώτερη έκδοση της Hibernate, η οποία υπάρχει στον ιστότοπο της Hibernate στη διεύθυνση *www.hibernate.org*. Επίσης, πρέπει να υπάρχει εγκατεστημένο στο μηχάνημα που θα αναπτυχθεί η εφαρμογή το *Apache Ant*, καθώς και μια πρόσφατη έκδοση της *HSQLDB*, που θα είναι η βάση που θα χρησιμοποιήσουμε.

5.3.3 Δημιουργία του καταλόγου εργασίας

Δημιουργούμε ένα κατάλογο σε όποια τοποθεσία επιθυμούμε. Αυτός ο κατάλογος ονομάζεται ***κατάλογος εργασίας (working directory)***.Στον κατάλογο εργασίας τοποθετούμε τους φακέλους *lib*, στον οποίο τοποθετούμε όλες τις απαραίτητες βιβλιοθήκες για την εφαρμογή, και *src*. Η δομή του καταλόγου εργασίας πρέπει να είναι όπως φαίνεται παρακάτω:

```
WORKDIR
  +lib
    antlr.jar
    asm.jar
    asm-attrs.jar
    cglib.jar
    commons-collections.jar
    commons-logging.jar
    dom4.jar
    hibernate3.jar
    hsqldb.jar
    jta.jar
  +src
```

Οι παραπάνω βιβλιοθήκες υπάρχουν στην διανομή της Hibernate και οι περισσότερες από αυτές συνήθως απαιτούνται για την ανάπτυξη μιας τυπικής εφαρμογής με τη χρήση της Hibernate. Το αρχείο hsqldb.jar προέρχεται από τη διανομή της HSQLDB, και σε περίπτωση που γίνει χρήση άλλου συστήματος διαχείρισης βάσεων δεδομένων, θα πρέπει να αντικατασταθεί από τον αντίστοιχο οδηγό (driver της βάσης.

5.3.4 Δημιουργίας της κλάσης προς αποθήκευση

Οι εφαρμογές της Hibernate ορίζουν κλάσεις προς αποθήκευση (*peristant classes*) οι οποίες αντιστοιχίζονται στους πίνακες της βάσης δεδομένων. Παρακάτω παρουσιάζεται μια απλή κλάση που ορίζει ένα χρήστη ενός συστήματος.

```
1  package customer;
2
3  public class Customer {
4      private Long id;
5      private String name;
6      private String surname;
7
8      public Customer() {
9      }
10
11     public Long getId() {
12         return id;
13     }
14
15     private void setId(Long id) {
16         this.id = id;
17     }
18
19     public String getName() {
```

```
20          return name;
21      }
22
23      public void setName(String name) {
24          this.name = name;
25      }
26
27      public String getSurname() {
28          return surname;
29      }
30
31      public void setSurname(String surname) {
32          this.surname = surname;
33      }
34  }
```

Listing 5.1: Customer.java

Η κλάση Customer έχει τρείς μεταβλητές μέλη: *id* — έναν αναγνωριστικό αριθμό, *name* — το όνομα του χρήστη, *surname* — το επώνυμο του χρήστη. Ο αναγνωριστικός αριθμός θα αποτελέσει το πρωτεύον κλειδί, το οποίο θα επιτρέπει στην εφαρμογή να ξεχωρίζει τις διαφορετικές εγγραφές της κλάσης στην βάση δεδομένων. Δηλαδή αν δύο στιγμιότυπα της κλάσης Customer έχουν το ίδιο id, τότε αντιστοιχούν στην ίδια εγγραφή στη βάση δεδομένων.

Όλες οι μεταβλητές - μέλη της κλάσης, για να αντιστοιχηθούν σε κάποια στήλη του πίνακα της βάσης, πρέπει να έχουν μεθόδους τύπου JavaBeans μέσω των οποίων δίνεται πρόσβαση σε αυτές από άλλες κλάσεις. Επιπλέον, πρέπει να υπάρχει μία μέθοδος δημιουργός (constructor) η οποία δεν δέχεται κανένα όρισμα.

5.3.5 Αντιστοίχηση της κλάσης στο σχήμα βάσης

Η Hibernate χρειάζεται περισσότερες πληροφορίες για το πως θα πρέπει να αντιστοιχίσει την κλάση Customer στην βάση δεδομένων. Με άλλα λόγια, πρέπει να ξέρει πως τα στιγμιότυπα της κλάσης αποθηκεύονται και φορτώνονται στη βάση δεδομένων. Αυτά τα μεταδεδομένα μπορούν να γραφτούν σε ένα αρχείο αντιστοίχισης, γραμμένο σε XML, το οποίο, εκτός των άλλων, ορίζει πώς θα αντιστοιχηθούν οι μεταβλητές-μέλη της κλάσης *Customer* στις στήλες του πίνακα *CUSTOMERS* της βάσης δεδομένων. Το αρχείο αυτό πρέπει να έχει το όνομα της ακολουθούμενο από την κατάληξη *hbm.xml*, όπου στην περίπτωση του παραδείγματος θα είναι: *Customer.hbm.xml*.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE hibernate-mapping PUBLIC
3          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4          "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5  <hibernate-mapping>
6      <class name="customer.Customer" table="CUSTOMERS">
7          <id name="id" column="CUSTOMER_ID">
8              <generator class="increment"/>
9          </id>
10         <property name="name" column="CUSTOMER_NAME"/>
11         <property name="surname" column="CUSTOMER_SURNAME"/>
12     </class>
13 </hibernate-mapping>
```

Listing 5.2: Customer.hbm.xml

Το αρχείο αντιστοίχισης λέει στη Hibernate πώς η κλάση *Customer* θα αποθηκευθεί στον πίνακα *CUSTOMERS*, και πως το πρωτεύον κλειδί, το οποίο ορίζεται από το tag *id*, θα είναι η μεταβλητή *id* της κλάσης. Το tag *property* χρησιμοποιείται για τις υπόλοιπες μεταβλητές της κλάσης που αντιστοιχούν στις υπόλοιπες στήλες του πίνακα της βάσης και με την ιδιότητα *column* ορίζουμε ρητά σε ποιά στήλη αντιστοιχίζεται κάθε μεταβλητή, κάτι το οποίο, στις περισσότερες περιπτώσεις, θα μπορούσε να παραληφθεί και να αφήσουμε τη Hibernate να το επιλέξει αυτόματα.

Μετά τη δημιουργία των δύο παραπάνω αρχείων ο κατάλογος εργασίας πρέπει να έχει την παρακάτω δομή:

```
WORKDIR
  +lib
    <Hibernate and third-party libraries>
  +src
    +customer
      Customer.java
      Customer.hbm.xml
```

5.3.6 Αποθήκευση και ανάκτηση αντικειμένων

Η παρακάτω κλάση δείχνει τον τρόπο με τον οποίο αποθηκεύουμε ένα αντικείμενο στη βάση και έπειτα να το ανακτούμε από αυτή:

```
1  package customer;
2
3  import org.hibernate.*;
4
5  import java.util.*;
6
7  import util.HibernateUtil;
8
9  public class CustomerManager {
10
11     public static void main(String[] args) {
12
13         Customer customer = new Customer();
14         customer.setName("Joe");
15         customer.setSurname("Doe");
16
17         //Insert the customer into the database
18         Session session = HibernateUtil.getSessionFactory().getCurrentSession()
               ;
19         session.beginTransaction();
20         session.save(customer);
21         session.getTransaction().commit();
22         HibernateUtil.getSessionFactory().close();
23
24         //Retrieve customers from database
25         Session newsession = HibernateUtil.getSessionFactory().
               getCurrentSession();
26         newsession.beginTransaction();
27         Query query = newsession.createQuery("from Customer");
28         List customers = query.list();
29         newsession.getTransaction().commit();
```

```
30          HibernateUtil.getSessionFactory().close();
31      }
32  }
```

Listing 5.3: CustomerManager.java

Ο παραπάνω κώδικας βρίσκεται στο αρχείο *CustomerManager.java*. Είναι μία κλάση που περιέχει την κλασική μέθοδο *main*. Σε αυτή τη μέθοδο, δημιουργείται ένα αντικείμενο της κλάσης *Customer*, αποθηκεύεται σε μία HSQLDB βάση δεδομένων (το πρώτο μέρος του κώδικα της κλάσης), από την οποία, έπειτα ανακτούμε όλες τις εγγραφές τύπου *Customer* (το δεύτερο μέρος του κώδικα της κλάσης).

Στην κλάση, η Hibernate καλεί τις διεπαφές (interfaces) *Session, Transaction* και *Query* για την πρόσβαση στη βάση δεδομένων:

- ***Συνεδρία (Session)*** — Το Session της Hibernate είναι πολλά πράγματα σε ένα. Είναι ένα μονονηματικό, μη διαμοιραζόμενο αντικείμενο, το οποίο αντιπροσωπεύει ένα συγκεκριμένο κομμάτι εργασίας με τη βάση. Αποτελεί το API διαχείρισης της διασύνδεσης μεταξύ βάσης και Java κώδικα. Είναι αυτό που χρησιμοποιείται για την ανάκτηση και την αποθήκευση αντικειμένων.

- ***Συναλλαγή (Transaction)*** — Αυτό το API της Hibernate μπορεί να χρησιμοποιηθεί για να θέσει προγραμματιστικά περιορισμούς συναλλαγής, αλλά αυτό είναι προαιρετικό (οι περιορισμοί της συναλλαγής δεν είναι προαιρετικοί).

- ***Ερώτημα (Query)*** — Ένα ερώτημα προς τη βάση μπορεί να γραφτεί είτε σε απλή SQL είτε στην αντικειμενοστραφή γλώσσα που διαθέτει η Hibernate για ερωτήματα (HQL). Αυτή η διεπαφή επιτρέπει τη δημιουργία ερωτημάτων καθώς και την εκτέλεση τους με διάφορους τρόπους.

Αγνοούμε για λίγο τη γραμμή στην οποία καλείται η HibernateUtil.getSessionFactory() — επεξηγείται παρακάτω.

5.3.7 Ρύθμιση και χρήση της Hibernate

Ο συνηθισμένος τρόπος αρχικοποίησης της Hibernate, είναι η δημιουργία ενός αντικειμένου τύπου *SessionFactory* μέσω ενός αντικειμένου *Configuration*. Το *SessionFactory* χρειάζεται για τη δημιουργία νέων Sessions, ενώ το αντικείμενο τύπου Configuration αποτελεί ένα είδος αναπαράστασης ενός αρχείου ρυθμίσεων (configuration file) για τη Hibernate. Όλα αυτά συνδυάζονται στην κλάση *HibernateUtil*.

```
1  package util;
2
3  import org.hibernate.*;
4  import org.hibernate.cfg.*;
5
6  public class HibernateUtil {
7
8      private static final SessionFactory sessionFactory;
9
10     static {
11         try {
12             // Create the SessionFactory from hibernate.cfg.xml
```

```
13              sessionFactory = new Configuration().configure().
                    buildSessionFactory();
14          } catch (Throwable ex) {
15              // Make sure you log the exception, as it might be swallowed
16              System.err.println("Initial SessionFactory creation failed." + ex);
17              throw new ExceptionInInitializerError(ex);
18          }
19      }
20
21      public static SessionFactory getSessionFactory() {
22          return sessionFactory;
23      }
24
25  }
```

Listing 5.4: HibernateUtil.java

Με την παραπάνω κλάση γίνεται η χρήση του *SessionFactory* βολική. Ουσιαστικά δημιουργείται ένας τρόπος χειρισμού ενός μοναδικού *SessionFactory* έτσι ώστε όλα τα *Sessions* της συγκεκριμένης εφαρμογής να το χρησιμοποιούν. Για την αρχικοποιήση του sessionFactory η Hibernate θα ψάξει να βρει το default αρχείο ρυθμίσεων στον κατάλογο εργασίας. Το αρχείο αυτό θα πρέπει να έχει το όνομα *hibernate.cfg.xml*. Ωστόσο, θα μπορούσε να χρησιμοποιηθεί οποιοδήποτε άλλο όνομα, αρκεί να οριστεί ρητά γράφοντας:

```
sessionFactory = new Configuration()
                    .configure("/foo/myconfigurationfile.cfg.xml")
                    .buildSessionFactory();
```

Το αρχείο ρυθμίσεων της Hibernate είναι αυτό που ορίζει τον τρόπο σύνδεσης της εφαρμογής με το σύστημα διαχείρισης των βάσεων δεδομένων. Για την εφαρμογή που περιγράφουμε το αρχείο αυτό είναι το παρακάτω:

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5
6  <hibernate-configuration>
7      <session-factory>
8          <!-- Database connection settings -->
9          <property name="connection.driver_class">org.hsqldb.jdbcDriver</
               property>
10         <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
11         <property name="connection.username">sa</property>
12         <property name="connection.password"></property>
13
14         <!-- JDBC connection pool (use the built-in) -->
15         <property name="connection.pool_size">1</property>
16
17         <!-- SQL dialect -->
18         <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
19         <!-- Enable Hibernate's automatic session context management -->
20         <property name="current_session_context_class">thread</property>
21
22         <!-- Disable the second-level cache  -->
```

```
23          <property name="cache.provider_class">org.hibernate.cache.
                NoCacheProvider</property>
24
25          <!-- Echo all executed SQL to stdout -->
26          <property name="show_sql">true</property>
27
28          <!-- Drop and re-create the database schema on startup -->
29          <property name="hbm2ddl.auto">create</property>
30          <mapping resource="customer/Customer.hbm.xml"/>
31      </session-factory>
32  </hibernate-configuration>
```

Listing 5.5: hibernate.cfg.xml

Στο παραπάνω αρχείο ορίζουμε τις διάφορες βάσεις δεδομένων στις οποίες επιθυμούμε η εφαρμογή να έχει πρόσβαση καθώς και τη συμπεριφοράς της Hibernate σε κάθε περίπτωση. Για κάθε μία διαφορετική βάση που θέλουμε η Hibernate να επικοινωνήσει, αρκεί να ορίσουμε ένα διαφορετικό <*session-factory*>. Τα διάφορα <*session-factory*> μπορούν να τοποθετηθούν όλα στο ίδιο αρχείο, αλλά θεωρείται ευκολότερο προγραμματιστικά αυτά να είναι διαχωρισμένα σε διαφορετικά αρχεία.

Για κάθε διαφορετική διασύνδεση με τη βάση πρέπει να ορίσουμε συγκεκριμένες παραμέτρους έτσι ώστε να είναι δυνατή η αντιστοίχιση. Τέτοιες παράμετροι είναι το είδος της βάσης δεδομένων (πχ MySQL, Oracle κτλ), ο οδηγός (driver), που θα αναλάβει τις ενδιάμεσες ενέργειες, το όνομα και ο κωδικός του χρήστη που έχει πρόσβαση στη βάση, η διάλεκτος που χρησιμοποιεί η βάση, η τοποθεσία των αρχείων ρυθμίσεων για κάθε μια αντιστοιχισμένη κλάση. Βέβαια, εκτός αυτών των παραμέτρων που είναι απαραίτητες για τη λειτουργία της εφαρμογής, υπάρχουν και άλλες προαιρετικές οι οποίες ορίζουν την συμπεριφορά της Hibernate. Παράδειγμα τέτοιων παραμέτρων είναι το αν επιθυμούμε να χρησιμοποιούμε cache, ο μέγιστος αριθμός των αντικειμένων που θέλουμε να ανακτώνται κάθε φορά καν αν θέλουμε κάθε φορά να δημιουργείται η βάση από την αρχή. Αναλυτική περιγραφή όλων αυτών υπάρχει στην τεκμηρίωση της Hibernate.

Με αυτόν τον τρόπο, έχοντας κάνει τις ρυθμίσεις της Hibernate, έχουμε ολοκληρώσει όλες τις απαραίτητες ενέργειες για τη δημιουργία μιας απλής εφαρμογής.

ΚΕΦΑΛΑΙΟ 6

Το περιβάλλον Spring

Η Java είναι μία γλώσσα η οποία έχει τη δυνατότητα να δημιουργεί πολύπλοκες εφαρμογές οι οποίες αποτελούνται από διακριτά μέρη. Μία προσπάθεια να εκμεταλλευτεί ο προγραμματιστής αυτή την ιδιότητα της Java έγινε από τη Sun Microsystems όταν το 1996 ανακοίνωσε την προδιαγραφή JavaBeans 1.00-A. Αυτή η προδιαγραφή όριζε ένα σύνολο πολιτικών κώδικα που έδινε τη δυνατότητα σε απλά αντικείμενα Java να επαναχρησιμοποιούνται και να συνδυάζονται εύκολα για την δημιουργία πιο πολύπλοκων εφαρμογών. Όμως οι λειτουργίες που πρόσφερε η JavaBeans δεν ήταν αρκετές για να ικανοποιήσουν τους προγραμματιστές. Για αυτό το λόγο, η Sun δύο χρόνια αργότερα ανακοίνωσε την πρώτη έκδοση της *Enterprise JavaBeans (EJB)* προδιαγραφής. Παρόλο που η προδιαγραφή αυτή απλοποίησε αρκετά την υποδομή κάποιων πλευρών στην ανάπτυξη του κώδικα, περιέπλεξε την ανάπτυξη με την ανάθεση deployment descriptor και τη συγγραφή κώδικα για τη σύνδεση των interfaces. Για αυτό χρησιμοποιείται όλο και λιγότερο από τους προγραμματιστές τα τελευταία χρόνια.

Σήμερα, η ανάπτυξη εφαρμογών σε Java επιστρέφει στις ρίζες της. Νέες προγραμματιστικές τεχνικές, που περιλαμβάνουν aspect-oriented programming (AOP) και dependency injection (DI), επιστρέφουν στην JavaBeans την δύναμη που είχε κλέψει η EJB. Αυτές οι τεχνικές εμπλουτίζουν απλά αντικείμενα της Java — plain-old Java objects (POJOs) με δηλωτικό προγραμματισμό, που θυμίζει EJB, χωρίς όμως την πολυπλοκότητά της. Η *Spring* αποτελεί ένα από τα πιο ελαφριά διαδεδομένα περιβάλλοντα ανάπτυξης προγραμματισμού βασισμένα σε POJOs.

6.1 Τι είναι η Spring;

Η Spring ένα περιβάλλον εργασίας ανοιχτού κώδικα (open source framework), το οποίο δημιουργήθηκε από τον *Rob Johnson* και περιγράφηκε στο βιβλίο του [18]. Ο λόγος της δημιουργίας του ήταν η αντιμετώπιση της πολυπλοκότητας στην ανάπτυξη εφαρμογών *enterprise*. Η Spring κάνει δυνατή την χρήση απλών JavaBeans για την επίτευξη πραγμάτων που μέχρι πριν μπορούσαν να γίνουν μόνο μέσω EJBs. Ωστόσο, η χρησιμότητα της Spring δεν περιορίζεται μόνο στην ανάπτυξη server-side εφαρμογών. Οποιαδήποτε εφαρμογή Java μπορεί να επωφεληθεί από τη χρήση της Spring σε σχέση με την *απλότητα του κώδικα*, τον *αποτελεσματικό έλεγχο* και τη *χαλαρή διασύνδεσή (loose coupling)* της.

Η Spring κάνει πολλά πράγματα, αλλά αν την αναλύσει κάποιος στα βασικά της κομμάτια, θα δει ότι είναι ένας ελαφρύς (lightweight) aspect-oriented container ο ο-

ποίος εφαρμόζει dependency injection, ενώ παράλληλα αποτελεί και ένα περιβάλλον εργασίας. Για να γίνουν πιο κατανοητές αυτές οι έννοιες αναλύονται με περισσότερη λεπτομέρεια παρακάτω:

- *Lightweight* (Ελαφρύς) — Αυτή η ιδιότητα της Spring αφορά τόσο το μέγεθος όσο και το επιπλέον φορτία (overhead). Το κύριο μέρος του περιβάλλοντος εργασίας της Spring μπορεί να διανεμηθεί σε ένα μοναδικό αρχείο jar το οποίο δεν ξεπερνά τα 2.5 MB, ενώ η επιπρόσθετη εργασία που απαιτείται είναι αμελητέα.

- *Dependency Injection* — Η Spring προάγει τη χαλαρή διασύνδεση χρησιμοποιώντας μια τεχνική που είναι γνωστή ως *dependency injection* (DI). Όταν εφαρμόζεται η DI, τα αντικείμενα λαμβάνουν παθητικά τις εξαρτήσεις τους (dependencies) αντί να τις δημιουργούν ή να τις αναζητούν μόνα τους. Μπορεί κανείς να το σκεφτεί σαν το αντίστροφο του Java Naming and Directory Interface (JNDI) — αντί το αντικείμενο να αναζητά τις εξαρτήσεις του από έναν container, ο container να δίνει τις εξαρτήσεις στο αντικείμενο χωρίς να περιμένει πρώτα να ερωτηθεί.

- *Aspect-Oriented* — Η Spring παρέχει πλούσια υποστήριξη σε *Aspect Oriented Programming (AOP* και επιτρέπει τη δημιουργία εφαρμογών με συνοχή, το οποίο επιτυγχάνεται από το διαχωρισμό της εφαρμογής σε λειτουργικά επίπεδα. Έτσι, κάθε αντικείμενο της εφαρμογής κάνει αυτό για το οποίο είναι προορισμένο και τίποτα παραπάνω. Δεν είναι υπεύθυνο (ούτε καν γνωρίζει) για άλλες λειτουργίες της εφαρμογής.

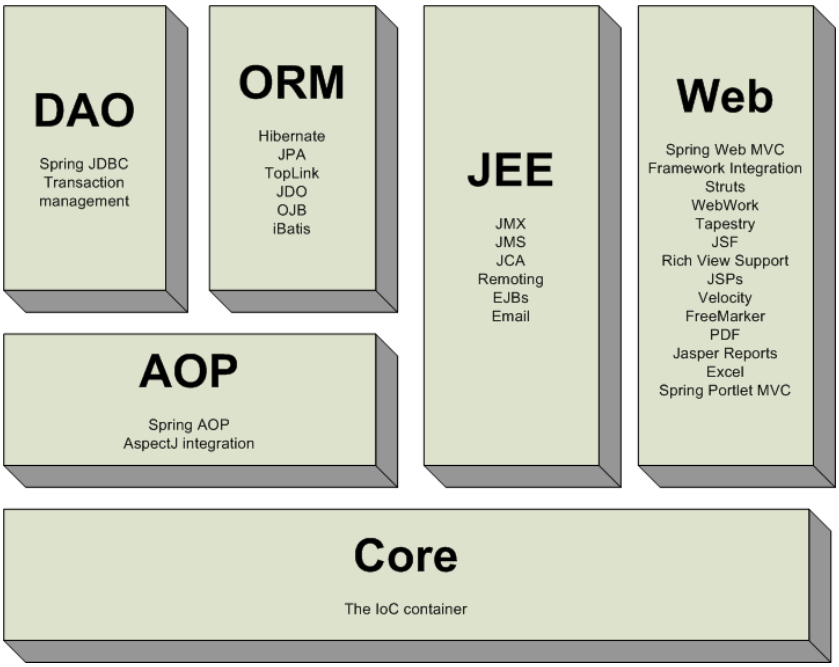
- *Container* — Η Spring αποτελεί έναν container, με την έννοια ότι κρατάει και διαχειρίζεται τον κύκλο ζωής και τη διαμόρφωση των αντικειμένων της εφαρμογής. Ο προγραμματιστής μπορεί να ορίσει πως θέλει να διαμορφώνονται και να δημιουργούνται τα αντικείμενα, καθώς και ποιες θέλει να είναι οι σχέσεις μεταξύ τους.

- *Framework* (Περιβάλλον εργασίας) — Η Spring παρέχει τη δυνατότητα να δημιουργηθούν πολύπλοκες εφαρμογές από το συνδυασμό άλλων, λιγότερο πολύπλοκων τμημάτων. Στη Spring τα αντικείμενα δημιουργούνται μέσω δηλώσεων σε απλά αρχεία XML. Επίσης, παρέχει πολλές δομικές λειτουργίες, όπως η διαχείριση συναλλαγών, η ενσωμάτωση ενός περιβάλλοντος εργασίας για την αποθήκευση δεδομένων κ.α., επιτρέποντας στον προγραμματιστή να ασχοληθεί περισσότερο με τη λογική της εφαρμογής.

Συνοψίζοντας, η Spring είναι ένα περιβάλλον που βοηθάει τον προγραμματιστή να δημιουργήσει εφαρμογές με χαλαρά διασυνδεδεμένο κώδικα. Ακόμη και αυτά ήταν όλα όσα μπορούσε να κάνει η Spring, τα οφέλη θα ήταν πάρα πολλά από άποψη συντηρισιμότητας και ελεγξιμότητας, και θα άξιζε τον κόπο η χρήση της. Όμως η Spring προσφέρει πολλά περισσότερα. Περιλαμβάνει κομμάτια που βασίζονται στα DI και AOP και έτσι συνθέτουν μια πλατφόρμα με πολλές λειτουργίες στην οποία μπορούν να αναπτυχθούν εφαρμογές.

6.2 Τα μέρη της Spring (Spring modules)

Το περιβάλλον εργασίας της Spring αποτελείται από αρκετά, καλά ορισμένα μέρη (modules), τα οποία ως σύνολο δίνουν στον προγραμματιστή ό, τι χρειάζεται για την ανάπτυξη εφαρμογών enterprise. Δεν χρειάζεται όλη η εφαρμογή να βασίζεται στη Spring. Ο προγραμματιστής είναι ελεύθερος να επιλέξει εκείνα τα μέρη που ταιριάζουν στην εφαρμογή του αλλά και να αναζητήσει άλλες επιλογές όταν η Spring δεν τον καλύπτει. Επίσης, παρέχει τρόπους διασύνδεσης με περιβάλλοντα εργασίας και βιβλιοθήκες, έτσι ώστε ο προγραμματιστής να μην χρειαστεί να τα αναπτύξει από την αρχή.



Σχήμα 6.1: Spring modules

Όπως φαίνεται στο παραπάνω σχήμα, όλα τα μέρη της Spring είναι χτισμένα πάνω στον container του πυρήνα της. Ο πυρήνας καθορίζει τον τρόπο που δημιουργούνται, αρχικοποιούνται και χειρίζονται τα beans — δηλαδή οι κύριες λειτουργίες της Spring. Όμως ο προγραμματιστής είναι πολύ πιθανό να ενδιαφέρεται για τα άλλα μέρη της Spring τα οποία χρησιμοποιούν τις υπηρεσίες που προσφέρει ο πυρήνας.

Παρακάτω περιγράφεται κάθε ένα από τα μέρη που παρουσιάζονται στο προηγούμενο σχήμα:

Ο Πυρήνας — core container

Ο πυρήνας παρέχει τη βασική λειτουργικότητα του περιβάλλοντος εργασίας της Spring. Σε αυτό το μέρος περιλαμβάνεται το BeanFactory, το οποίο είναι ο θεμελιώδης container τηςSpring και η βάση στην οποία στηρίζεται το DI της Spring.

Το AOP module

Η Spring παρέχει υποστήριξη για aspect-oriented programming μέσω του AOP module της. Όπως και το DI, το AOP υποστηρίζει τη χαλαρή διασύνδεση μεταξύ των

αντικειμένων κάθε εφαρμογής. Ωστόσο, με το AOP, σημαντικά θέματα που αφορούν μια εφαρμογή (όπως οι συναλλαγές και οι ασφάλεια) χωρίζονται από τα αντικείμενα στα οποία εφαρμόζονται.

Το ORM module

Για αυτούς που προτιμούν να χρησιμοποιούν ένα εργαλείο για object-relation mapping (ORM) αντί του JDBC, η Spring παρέχει το ORM module. Η υποστήριξη της Spring για ORM χτίζεται πάνω στο DAO παρέχοντας έναν αρκετά βολικό τρόπο για ανάπτυξη DAO για αρκετές λύσεις που προσφέρουν object-relational mapping. Η Spring δεν υλοποιεί κάποια λύση για ORM, υποστηρίζει αρκετά δημοφιλή περιβάλλοντα εργασίας, συμπεριλαμβανομένων των Hibernate, Java Persistence API, Java Data Objects και iBatis SQL Maps. Επιπλέον, το σύστημα διαχείρισης συναλλαγών της Spring υποστηρίζει και το JDBC.

Το DAO module

Η ανάπτυξη με JDBC συχνά περιλαμβάνει τη συγγραφή αρκετού κώδικα για τη σύνδεση στη βάση δεδομένων, τη δημιουργία κάποιας εντολής (statement), την επεξεργασία του αποτελέσματος και το κλείσιμο της σύνδεσης με τη βάση. Μέσω του Data Access Object (DAO) module, η Spring απλοποιεί τη διαδικασία που περιγράφηκε, διατηρεί τον κώδικα που αφορά τη βάση δεδομένων απλό και κατανοητό, και περιορίζει τα προβλήματα που μπορεί να προκύψουν. Επιπλέον, χτίζει ένα επίπεδο με κατανοητές εξαιρέσεις πάνω από τα μηνύματα σφάλματος που στέλνουν οι εξυπηρετητές των βάσεων δεδομένων.

Επιπρόσθετα, αυτό το module (χρησιμοποιώντας το AOP module) παρέχει υπηρεσίες διαχείρισης συναλλαγών για τα αντικείμενα των εφαρμογών που χρησιμοποιούν τη Spring.

Το Web module

Η Spring παρέχει μία πλούσια συλλογή κλάσεων για τη δημιουργία εφαρμογών που στηρίζονται στο διαδίκτυο (web-based applications) μέσω του web module της. Υποστηρίζεται η χρήση του προτύπου σχεδίασης εφαρμογών Model/View/Controller, ο χειρισμός ηλεκτρονικής αλληλογραφίας, καθώς και η χρήση απομακρυσμένων υπηρεσιών (remote support).

Όσον αφορά τη χρήση του προτύπου σχεδίασης εφαρμογών Model/View/Controller (MVC), η Spring υποστηρίζει πλήρως το πιο γνωστό MVC περιβάλλον εργασίας, το Apache Struts. Με αυτόν τον τρόπο δίνεται η δυνατότητα στον προγραμματιστή να χρησιμοποιήσει στις ήδη σχεδιασμένες κλάσεις του, τις αρχές του DI που προσφέρει η Java. Βέβαια, πέραν της υποστήριξης του Apache Struts, η Spring προσφέρει και τη δική της υλοποίηση για το πρότυπο MVC. Μέσω αυτού μπορεί να χρησιμοποιηθεί μια ευρεία γκάμα τεχνολογιών παρουσίασης, από σελίδες JSP και Apache Jakarta Velocity μέχρι Microsoft Excel και Adobe PDF.

Για την αποστολή μηνυμάτων ηλεκτρονικής αλληλογραφίας, η Σπρινγ παρέχει ένα αρκετά απλοποιημένο ΑΠΙ, το οποίο ταιριάζει στην φιλοσοφία του DI της Spring. Για το σκοπό αυτό παρέχονται δύο υλοποιήσεις, μία του JavaMail και μία της MailMessage κλάσης από το πακέτο *com.oreilly.servlet* του Jason Hunter. Μέσω αυτών δίνεται η δυνατότητα δημιουργίας ενός πρότυπου μηνύματος, το οποίο στη συνέχεια

μπορεί να χρησιμοποιηθεί σαν βάση για την αποστολή αλληλογραφίας μέσω της εφαρμογής.

Τέλος, για τη χρήση απομακρυσμένων υπηρεσιών, παρέχεται εκτεταμένη υποστήριξη μία μεγάλης συλλογής τεχνικών απομακρυσμένης πρόσβασης, για τη γρήγορη δημιουργία και πρόσβαση σε απομακρυσμένες υπηρεσίες. Τέτοιες τεχνικές είναι οι Java RMI, JAXRPC, Caucho Hessian και Caucho Burlap. Εκτός αυτών, η Spring παρέχει και το δικό της πρωτόκολλο, που χρησιμοποιεί το HTTP πρωτόκολλο επικοινωνίας και βασίζεται στο απλό Java serialization.

Java EE Connector API (JCA)

Αν και τα τελευταία χρόνια όλο και περισσότεροι προγραμματιστές τείνουν να χρησιμοποιούν τις λεγόμενες ελαφριές πλατφόρμες με τεχνολογίες όπως η Spring και ο Tomcat, υπάρχουν πολλές περιπτώσεις που η χρήση αρκετών παραδοσιακών J2EE APIs κρίνεται απαραίτητη. Η σύνδεση και επικοινωνία μεταξύ τέτοιου είδους εφαρμογών και εφαρμογών που χρησιμοποιούν άλλες τεχνολογίες μπορεί να είναι δύσκολη. Το JCA της Spring παρέχει ένα σταθερό και αξιόπιστο τρόπο για την επικοινωνία με μια σειρά από τέτοιου είδους enterprise συστήματα. Τα κυριότερα APIs που υποστηρίζοντια είναι το Java Naming and Directory Interface (JNDI), τα ΕΘΒ καθώς και η Java Message Service (JMS).

6.3 Ένα παράδειγμα

Το πιο βασικό πράγμα που κάνει η Java είναι dependency injection. Στη συνέχεια θα παρουσιαστεί μια απλή εφαρμογή, μια παραλλαγή του συνηθισμένου "Hello World", η οποία θα παρουσιάσει τα βασικά σημεία της Spring.

Η πρώτη κλάση που χρειάζεται η Hello World εφαρμογή είναι μία service class, η οποία θα τυπώνει το γνωστό χαιρετισμό. Παρακάτω φαίνεται το interface *GreetingService* το οποίο ορίζει τις συναρτήσεις της κλάσης.

```
1 public interface GreetingService {
2   public void sayGreeting();
3 }
```

Listing 6.1: GreetingService.java

Ακολούθως παρουσιάζεται η κλάση *GreetingServiceImpl* η οποία υλοποιεί το παραπάνω interface. Η χρήση interface για τον ορισμό των ενεργειών που μπορούν να γίνουν με τα διάφορα αντικείμενα δεν είναι απαραίτητη, εντούτοις η πολιτική αυτή συνιστάται.

```
1  public class GreetingServiceImpl implements GreetingService {
2    private String greeting;
3
4    public GreetingServiceImpl() {}
5
6    public GreetingServiceImpl(String greeting) {
7      this.greeting = greeting;
8    }
9
10   public void sayGreeting() {
11     System.out.println(greeting);
12   }
```

```
13
14   public void setGreeting(String greeting) {
15     this.greeting = greeting;
16   }
17 }
```

Listing 6.2: GreetingServiceImpl.java

Η κλάση GreetingServiceImpl έχει μία μόνο μεταβλητή μέλος, την greeting. Αυτή είναι ένα απλό String το οποίο θα κρατάει το χαιρετισμό που θα τυπωθεί όταν θα καλεστεί η μέθοδος *sayGreeting()*. Επιπλέον, έχει και δύο μεθόδους δημιουργούς, έναν κενό και έναν που παίρνει ως όρισμα το χαιρετισμό. Οποιοσδήποτε, από τους δύο αυτούς δημιουργούς μπορεί να κληθεί, ανάλογα με τις παραμέτρους που θα οριστούν. Παρακάτω φαίνεται το αρχείο ρυθμίσεων (configuration file) *hello.xml* το οποίο ορίζει στον πυρήνα της Spring πως ακριβώς πρέπει να εκτελέσει την υπηρεσία που δημιουργήθηκε.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
6
7   <bean id="greetingService"
8       class="com.springinaction.chapter01.hello.GreetingServiceImpl">
9     <property name="greeting" value="Buenos Dias!" />
10  </bean>
11 </beans>
```

Listing 6.3: hello.xml

Στο παραπάνω αρχείο δηλώνεται ένα στιγμιότυπο της GreetingServiceImpl στον container της Spring στο οποίο ορίζεται ότι η μεταβλητή μέλος (property) θα έχει την τιμή *"Buenos Dias!"*.

Αναλύοντας λίγο τον παραπάνω XML κώδικα βλέπουμε ότι το root στοιχείο είναι το <*beans*>, κάτι που ισχύει για οποιοδήποτε αρχείο ρυθμίσεων της Spring. Το στοιχείο <bean> λέει στον container για μία κλάση πώς αυτή πρέπει να αρχικοποιηθεί. Το χαρακτηριστικό (attribute) id του στοιχείου <bean> χρησιμοποιείται, για να ορίσει το όνομα του bean, ενώ το class χρησιμοποιείται για να οριστεί το ακριβές όνομα της κλάσης (classpath).

Μέσα στο στοιχείο <bean> ορίζεται ένα στοιχείο <property> το οποίο χρησιμοποιείται για να οριστεί μία μεταβλητή μέλος, σε αυτή την περίπτωση, η *greeting*. Το στοιχείο property λέει στον πυρήνα να καλέσει την μέθοδο *sayGreeting()* δίνοντας της την τιμή *"Buenos Dias!"* κατά την αρχικοποίηση του bean.

Παρακάτω φαίνεται τι είναι αυτό που κάνει ακριβώς ο πυρήνας κατά την αρχικοποίηση της υπηρεσίας σύμφωνα με το XML αρχείο ρυθμίσεων.

```
GreetingServiceImpl greetingService =
      new GreetingServiceImpl();
greetingService.setGreeting("Buenos Dias!");
```

Αυτό που μένει είναι η δημιουργία μίας κλάσης η οποία να φορτώνει τον πυρήνα της Spring και να τον χρησιμοποιεί για την εκτέλεση της υπηρεσίας.

```
1 import org.springframework.beans.factory.BeanFactory;
2 import org.springframework.beans.factory.xml.XmlBeanFactory;
```

```
3 import org.springframework.core.io.ClassPathResource;
4
5 public class HelloApp {
6   public static void main(String[] args) throws Exception {
7     BeanFactory factory =
8         new XmlBeanFactory(new ClassPathResource("hello.xml"));
9
10    GreetingService greetingService =
11        (GreetingService) factory.getBean("greetingService");
12
13    greetingService.sayGreeting();
14  }
15 }
```

Listing 6.4: HelloApp.java

Η κλάση *BeanFactory* που χρησιμοποιείται παραπάνω αποτελεί τον πυρήνα της Spring. Μετά τη φόρτωση του hello.xml αρχείου από τον πυρήνα η *main()* καλεί την μέθοδο *getBean()*, ώστε να ανακτήσει τις εξαρτήσεις της υπηρεσίας. Έχοντας αυτές τις εξαρτήσεις καλείται τελικά η μέθοδος *sayGreeting()*. Όταν εκτελεστεί η εφαρμογή, όπως αναμενόταν, τυπώνεται το:

```
Buenos Dias!
```

Η παραπάνω εφαρμογή αποτελεί το πιο απλό παράδειγμα χρήσης της Spring. Παρά την απλότητά του, παρουσιάζει το βασικό τρόπο ρύθμισης και χρήσης κλάσεων μέσω της Spring. Όμως, λόγω της απλότητάς του, δε δείχνει πως μπορεί να ρυθμιστεί κάποιο bean, ώστε να περαστεί η τιμή ενός String στο πεδίο (injection). Η πραγματική δύναμη της Spring έγκειται στο πως μπορούν τα διάφορα beans να συνδεθούν με άλλα beans κάνοντας χρήση του DI.

6.4 Κατανοώντας το dependency injection

Αν και η Spring έχει πολλές δυνατότητες, το DI αποτελεί την καρδιά του περιβάλλοντος εργασίας. Παρά το γεγονός ότι ο όρος αυτός ακούγεται σαν μια περίπλοκη προγραμματιστική τεχνική ή τρόπο σχεδιασμού (design pattern), στην πραγματικότητα δεν είναι τόσο πολύπλοκο όσο ακούγεται. Αντίθετα, η χρήση του, κάνει τον κώδικα πολύ πιο απλό, ευκολονόητο και εύκολα συντηρήσιμο.

Κάθε μη τετριμμένη εφαρμογή (δηλαδή οτιδήποτε πιο πολύπλοκο από το κλασικό παράδειγμα “Hello World”) αποτελείται από δύο ή περισσότερες κλάσεις οι οποίες συνεργάζονται, για να εκτελέσουν τις απαραίτητες ενέργειες. Παραδοσιακά κάθε αντικείμενο είναι υπεύθυνο να διατηρεί τις αναφορές του (reference) στα αντικείμενα με τα οποία συνεργάζεται (δηλαδή τις εξαρτήσεις του — dependencies). Κάτι τέτοιο οδηγεί σε έναν ισχυρά δεμένο και δύσκολο στον έλεγχο κώδικα.

Όταν γίνεται η χρήση του DI, τα αντικείμενα λαμβάνουν τις εξαρτήσεις τους κατά την ώρα της δημιουργίας τους από μία εξωτερική οντότητα η οποία συντονίζει κάθε αντικείμενο στο όλο σύστημα. Με άλλα λόγια, οι εξαρτήσεις δίνονται (injected) στα αντικείμενα. Έτσι, το DI αποτελεί ένα είδος αντιστροφής στην ευθύνη που αφορά τη διατήρηση των αναφορών των αντικειμένων. Η λειτουργικότητα αυτή παρέχεται από τον πυρήνα της Spring, ο οποίος είναι υπεύθυνος να δίνει στα αντικείμενα τις εξαρτήσεις τους. Ο πυρήνας γνωρίζει πως πρέπει να δώσει αυτές τις εξαρτήσεις στα κατάλληλα αντικείμενα διαβάζοντας το XML αρχείο ρυθμίσεων της Spring.

Το κλειδί της παραπάνω τεχνικής είναι η χαλαρή διασύνδεση. Αν ένα αντικείμενο γνωρίζει για τις εξαρτήσεις του μόνο μέσω μίας διεπαφής (όχι μέσω της υλοποίησή τους ή μέσω του τρόπου που δημιουργούνται), η εξάρτηση μπορεί να αλλαχτεί πολύ εύκολα από μια διαφορετική υλοποίηση χωρίς το αντικείμενο να γνωρίζει για τη διαφορά.

Το dependency injection στην πράξη

Παρακάτω παρουσιάζεται ένα παράδειγμα για να γίνει πιο κατανοητό το dependency injection. Για τις ανάγκες του παραδείγματος δημιουργούμε μία κλάση: την *KnightOfTheRoundTable*. Η κλάση αυτή αντιπροσωπεύει έναν ιππότη της στρογγυλής τραπέζης ο οποίος ξεκινά μία αποστολή.

```
1  public class KnightOfTheRoundTable {
2    private String name;
3    private HolyGrailQuest quest;
4
5    public KnightOfTheRoundTable(String name) {
6      this.name = name;
7      quest = new HolyGrailQuest();
8    }
9
10   public Object embarkOnQuest() throws QuestFailedException {
11     return quest.embark();
12   }
13 }
```

Listing 6.5: KnightOfTheRoundTable.java

Η παραπάνω κλάση αντιπροσωπεύει έναν ιππότη και δέχεται ως όρισμα στο δημιουργό της το όνομα του. Επιπλέον έχει μια μεταβλητή μέλος που αντιπροσωπεύει την αποστολή του ιππότη. Ο δημιουργός θέτει την αποστολή του ιππότη αρχικοποιώντας τη αντίστοιχη μεταβλητή ως μία HolyGrailQuest. Η υλοποίηση της παρουσιάζεται παρακάτω.

```
1  public class HolyGrailQuest {
2    public HolyGrailQuest() {}
3
4    public Object embark() throws GrailNotFoundException {
5      HolyGrail grail = null;
6      // Look for grail
7      ...
8      return grail;
9    }
10 }
```

Listing 6.6: HolyGrailQuest.java

Το αρνητικό που παρουσιάζεται σε αυτή την υλοποίηση είναι πως ο ιππότης της στρογγυλής τραπέζης λαμβάνει την αναζήτηση για το ιερό δισκοπότηρο. Ο ιππότης είναι υπεύθυνος να πάρει μόνος του μια αποστολή. Ένα από τα προβλήματα που δημιουργούνται είναι ότι η κλάση KnigthOfTheRoundTable δεν μπορεί να ελεγχθεί απομονωμένα. Πάντα θα ελέγχεται έμμεσα και η κλάση HolyGrailQuest. Με άλλα λόγια το πρόβλημα είναι το ταίριασμα. Σε αυτό το σημείο η κλάση KnigthOfTheRoundTable είναι στατικά συνδεδεμένη με την κλάση HolyGrailQuest. Ένας τρόπος να μειωθεί το σφιχτό ταίριασμα είναι να κρύβονται οι λεπτομέρειες τις υλοποίησης

πίσω από interfaces έτσι ώστε η πραγματική κλάση υλοποίησης να μπορεί να αλλαχτεί χωρίς να επηρεάζει την κλάση που επωφελείται από αυτή την υλοποίηση. Για παράδειγμα, δημιουργούμε το παρακάτω interface:

```
1 public interface Quest {
2   public Object embark() throws QuestFailedException;
3 }
```

Listing 6.7: Quest.java

Μετά, τροποποιούμε την κλάση HolyGrailQuest ώστε να υλοποιεί το interface:

```
1 public class HolyGrailQuest implements Quest {
2   public HolyGrailQuest() {}
3
4   public Object embark() throws GrailNotFoundException {
5     // do whatever it means to embark on a quest
6     return new HolyGrail();
7   }
8 }
```

Listing 6.8: HolyGrailQuest.java

Επιπλέον αλλάζουμε την ακόλουθη μέθοδο στην κλάση KnigthOfTheRoundTable ώστε να είναι συμβατή με τον τύπο των αποστολών:

```
private Quest quest;
...
public Object embarkOnQuest() throws QuestFailedException {
  return quest.embark();
}
```

Παρόμοια θα ήταν επιθυμητό και η κλάση KnigthOfTheRoundTable να υλοποιεί το παρακάτω interface:

```
1 public interface Knight {
2   public Object embarkOnQuest() throws QuestFailedException;
3   public String getName();
4 }
```

Listing 6.9: Knight.java

Το να κρύβει κανείς την υλοποίηση πίσω από ένα interface είναι ένα θετικό βήμα. Αλλά δεν αρκεί πάντα. Για παράδειγμα, η επόμενη κλάση KnigthOfTheRoundTable αναθέτει στον ιππότη μια αποστολή μέσω του interface Quest αλλά και πάλι ο ιππότης είναι υπεύθυνος για την αναζήτηση της αποστολή του:

```
1  public class KnightOfTheRoundTable implements Knight {
2    private String name;
3    private HolyGrailQuest quest;
4
5    public KnightOfTheRoundTable(String name) {
6      this.name = name;
7      quest = new HolyGrailQuest();
8    }
9
10   public Object embarkOnQuest() throws QuestFailedException {
11     return quest.embark();
12   }
13 }
```

Listing 6.10: KnightOfTheRoundTable.java

Η κρίσιμη ερώτηση είναι εάν ο ιππότης πρέπει να είναι υπεύθυνος για την ανάληψη μιας αποστολής ή αν θα πρέπει κάποιος άλλος να του την αναθέτει. Οι παρακάτω αλλαγές γίνονται στην κλάση KnigthOfTheRoundTable:

```
1  public class KnightOfTheRoundTable implements Knight {
2    private String name;
3    private Quest quest;
4
5    public KnightOfTheRoundTable(String name) {
6      this.name = name;
7    }
8
9    public Object embarkOnQuest() throws QuestFailedException {
10     return quest.embark();
11   }
12
13   public void setQuest(Quest quest) {
14     this.quest = quest;
15   }
16
17   public String getName() {
18     return name;
19   }
20 }
```

Listing 6.11: KnightOfTheRoundTable.java

Τώρα η αποστολή ανατίθεται στον ιππότη, και ο ιππότης δεν είναι πλέον υπεύθυνος να την ανακτά μόνος του. Αυτή είναι η αρμοδιότητα του DI, η ευθύνη για το συντονισμό της συνεργασίας μεταξύ εξαρτημένων αντικειμένων μεταφέρεται μακρυά από τα ίδια τα αντικείμενα.

Η δημιουργία συσχετίσεων μεταξύ αντικειμένων ονομάζεται *wiring*. Στη Spring υπάρχουν πολλοί τρόποι να συσχετίσεις αντικείμενα μεταξύ τους αλλά ο πιο συνηθισμένος είναι μέσω αρχείων XML. Το παρακάτω αρχείο παρουσιάζει ένα αρχείο XML στο οποίο δίνεται μία αποστολή σε έναν ιππότη της στρογγυλής τραπέζης:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://www.springframework.org/schema/beans
5               http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
6
7    <bean id="quest"
8        class="HolyGrailQuest"/>
9
10   <bean id="knight"
11       class="KnightOfTheRoundTable">
12     <constructor-arg value="Bedivere" />
13     <property name="quest" ref="quest" />
14   </bean>
15 </beans>
```

Listing 6.12: knight.xml

Τώρα που δημιουργήσαμε τις συσχετίσεις, ας τρέξουμε την εφαρμογή. Σε μία Spring εφαρμογή, ένα BeanFactoy φορτώνει τους ορισμούς των beans και κάνει τους συσχετισμούς. Όταν η εφαρμογή έχει μια αναφορά σε ένα αντικείμενο Knight, απλά καλεί τη μέθοδο embarkOnQuest() και ξεκινάει η περιπέτεια του ιππότη.

```
1  import org.springframework.beans.factory.BeanFactory;
2  import org.springframework.beans.factory.xml.XmlBeanFactory;
3  import org.springframework.core.io.FileSystemResource;
4
5  public class KnightApp {
6    public static void main(String[] args) throws Exception {
7      BeanFactory factory =
8        new XmlBeanFactory(new FileSystemResource(knight.xml"));
9
10     Knight knight =
11         (Knight) factory.getBean("knight");
12
13     knight.embarkOnQuest();
14   }
15 }
```

Listing 6.13: KnightApp.java

Όπως φαίνεται παραπάνω η κλάση δεν γνωρίζει τίποτα σχετικό με την αποστολή του ιππότη. Μόνο το QML αρχείο γνωρίζει τον τύπο της αποστολής που θα λάβει ο ιππότης.

6.5 Aspect-Oriented Programming

Όπως περιγράφηκε στην προηγούμενη παράγραφο, το DI καθιστά εφικτή τη χαλαρή διασύνδεση τμημάτων λογισμικού σε εφαρμογές. Αντίθετα, ο aspect-oriented προγραμματισμός επιτρέπει τη συγκέντρωση λειτουργιών οι οποίες χρησιμοποιούνται σε όλη την εφαρμογή, σε επαναχρησιμοποιήσιμα τμήματα.

Ο aspect-oriented προγραμματισμός συχνά ορίζεται ως μια τεχνική η οποία προωθεί τον διαχωρισμό ευθυνών μέσα στο σύστημα λογισμικού. Τα συστήματα αποτελούνται από πολλά ξεχωριστά τμήματα, το καθένα από τα οποία είναι υπεύθυνο για συγκεκριμένο κομμάτι λειτουργικότητας. Συχνά όμως αυτά τα κομμάτια έχουν επιπλέον ευθύνες πέρα από αυτές τις βασικές λειτουργίες για τις οποίες προορίζονται. Υπηρεσίες, όπως είναι η διατήρηση αρχείου, η διαχείριση συναλλαγών και η ασφάλεια συχνά υλοποιούνται μέσα σε τμήματα λογισμικού που η βασική τους ευθύνη είναι εντελώς διαφορετική.

Μοιράζοντας τέτοιου είδους ευθύνες σε πολλαπλά τμήματα κώδικα δημιουργούνται δύο επίπεδα πολυπλοκότητας στον κώδικα που αναπτύσσεται:

- Ο κώδικας, ο οποίος υλοποιεί λειτουργίες που αφορούν όλη την εφαρμογή, είναι διασκορπισμένος σε πολλά διαφορετικά τμήματα αντί να είναι συγκεντρωμένος. Αυτό σημαίνει ότι αν ο προγραμματιστής αποφασίσει να αλλάξει τον τρόπο που εκτελούνται αυτές οι λειτουργίες, πρέπει να ανατρέξει σε πολλά διαφορετικά τμήματα. Ακόμα και αν η διαφοροποίηση έγκειται μόνο στην αλλαγή του τρόπου κλήσης μιας και μόνο μεθόδου, η διόρθωση θα πρέπει να γίνει σε πολλά διαφορετικά τμήματα κώδικα.

- Τα διάφορα τμήματα του κώδικα είναι επιφορτισμένα με λειτουργίες οι οποίες δεν τα αφορούν, προσθέτοντάς τους επιπλέον κώδικα. Για παράδειγμα, αν η λειτουργία μιας μεθόδου είναι να προσθέσει μία εγγραφή σε έναν τηλεφωνικό κατάλογο, δεν πρέπει να την αφορά το αν αυτή η διαδικασία γίνει με ασφάλεια.

Ο aspect-oriented προγραμματισμός κάνει εφικτό το διαχωρισμό των διάφορων υπηρεσιών επιτρέποντας, όποτε κάτι τέτοιο κρίνεται επιθυμητό, να γίνεται χρήση τους από τμήματα κώδικα (components) που τις χρειάζονται. Αυτό οδηγεί στην ανάπτυξη πιο συνεκτικών κομματιών τα οποία είναι περισσότερο συγκεντρωμένα στο βασικό τους στόχο, αγνοώντας τελείως τις διάφορες άλλες υπηρεσίες του συστήματος. Με λίγα λόγια, η χρήση των aspects διασφαλίσει ότι τα POJOs θα παραμείνουν απλά (plain).

Ο AOP στην πράξη

Για την καλύτερη κατανόηση του aspect-oriented προγραμματισμού θα δανειστούμε και θα επεκτείνουμε το παράδειγμα που χρησιμοποιήσαμε παραπάνω για το dependency injection. Υποθέτουμε ότι μαζί με τον ιππότη στέλνουμε και έναν ραψωδό στις αποστολές για να τραγουδάει για τα κατορθώματα του ιππότη. Ακολουθεί η κλάση *Minstrel* για το ραψωδό:

```
1  import org.apache.log4j.Logger;
2
3  public class Minstrel {
4    private static final Logger SONG =
5        Logger.getLogger(Minstrel.class);
6
7    public Minstrel() {}
8
9    public void singBefore(Knight knight) {
10     SONG.info("Fa la la; Sir " + knight.getName() +
11         " is so brave!");
12   }
13
14   public void singAfter(Knight knight) {
15     SONG.info("Tee-hee-he; Sir " + knight.getName() +
16         " did embark on a quest!");
17   }
18 }
```

Listing 6.14: Minstrel.java

Συνεχίζοντας να σκεφτόμαστε με τρόπο DI αλλάζουμε την κλάση του ιππότη με τον εξής τρόπο:

```
public class KnightOfTheRoundTable implements Knight {
...
  private Minstrel minstrel;
  public void setMinstrel(Minstrel minstrel) {
    this.minstrel = minstrel;
  }
...
  public HolyGrail embarkOnQuest() throws QuestFailedException {
    minstrel.singBefore(this);
    HolyGrail grail = quest.embark();
    minstrel.singAfter(this);
    return grail;
  }
}
```

Με αυτό τον τρόπο υλοποίησης υπάρχει το εξής πρόβλημα. Ο ιππότης θα πρέπει να λέει στον ραψωδό πότε θα ξεκινήσει να τραγουδάει και πότε να σταματήσει. Το να θυμάται ο ιππότης να δίνει εντολή στο ραψωδό για το πότε θα τραγουδήσει, είναι κάτι που δυσκολεύει τις αποστολές του ιππότη. Το ιδανικό θα ήταν ο ραψωδός να ξεκινά μόνος του να τραγουδάει και να σταματάει όταν πρέπει. Ο ιππότης δεν θα έπρεπε να ξέρει (ούτε να το νοιάζει) ότι τα κατορθώματα του καταγράφονται σε τραγούδια. Με λίγα λόγια οι υπηρεσίες του ραψωδού υπερβαίνουν τα καθήκοντα του ιππότη.

Για αυτό το λόγο θα ήταν λογικό να μετατρέψουμε τον ραψωδό σε aspect το οποίο προσθέτει τις υπηρεσίες του στον ιππότη. Τότε οι υπηρεσίες του ραψωδού θα σκεπάζουν τη λειτουργικότητα του ιππότη χωρίς ο ίδιος να ότι υπάρχει ο ραψωδός. Μετατρέπουμε την κλάση Minstrel σε aspect χρησιμοποιώντας την υποστήριξη της Spring για AOP:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:aop="http://www.springframework.org/schema/aop"
5         xsi:schemaLocation="http://www.springframework.org/schema/beans
6               http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
7             http://www.springframework.org/schema/aop
8               http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
9
10   <bean id="quest"
11       class="com.springinaction.chapter01.knight.HolyGrailQuest"/>
12
13   <bean id="knight"
14       class="com.springinaction.chapter01.knight.KnightOfTheRoundTable">
15     <constructor-arg value="Bedivere" />
16     <property name="quest" ref="quest" />
17   </bean>
18
19   <bean id="minstrel"
20       class="com.springinaction.chapter01.knight.Minstrel"/>
21
22   <aop:config>
23     <aop:aspect ref="minstrel">
24       <aop:pointcut
25           id="questPointcut"
26           expression="execution(* *.embarkOnQuest(..)) and target(bean)" />
27
28       <aop:before
29           method="singBefore"
30           pointcut-ref="questPointcut"
31           arg-names="bean" />
32
33       <aop:after-returning
34           method="singAfter"
35           pointcut-ref="questPointcut"
36           arg-names="bean" />
37     </aop:aspect>
38   </aop:config>
39 </beans>
```

Listing 6.15: knight.xml

Ας δούμε πιο αναλυτικά τι συμβαίνει στο παραπάνω αρχείο:

- Αρχικά, βρίσκουμε μία δήλωση <bean>, που δημιουργεί ένα bean minstrel.

Αυτό είναι η κλάση Minstrel που αναφέρεται παραπάνω, η οποία είναι ανεξάρτητη.

- Το στοιχείο <aop:config> δείχνει ότι πρόκειται να γίνουν κάποια AOP πράγματα. Τα περισσότερα στοιχεία της διαμόρφωσης AOP της Spring πρέπει να βρίσκονται μέσα σε <aop:config>.

- Μέσα στο <aop:config> υπάρχει ένα στοιχείο <aop:aspect>. Αυτό το στοιχείο δείχνει ότι δηλώνουμε ένα aspect. Η λειτουργία του aspect ορίζεται στο bean στο οποίο αναφέρεται από το χαρακτηριστικό *ref*.

- Ένα aspect αποτελείται από pointcuts (τα μέρη στα οποία η λειτουργικότητα εφαρμόζεται) και advice (πως θα εφαρμοστεί η λειτουργικότητα). Το στοιχείο <aop:pointcut> ορίζει το pointcut το οποίο ενεργοποιείται από την εκτέλεση της μεθόδου embarkOnQuest().

- και τέλος, υπάρχουν δύο τμήματα του AOP advice. Το στοιχείο <aop:before> δηλώνει ότι μία μεθοδος singBefore() του Minstrel θα πρέπει να κληθεί πριν το pointcut ενώ το στοιχείο <aop:after> δηλώνει ότι μία μεθοδος singAfter() του Minstrel θα πρέπει να κληθεί μετά το pointcut. Το pointcut και στις δύο περιπτώσεις είναι μία αναφορά στο questPointcut, το οποίο είναι η εκτέλεση του embarkOnQuest().

Πλέον, ο ιππότης δεν χρειάζεται να λέει στο ραψωδό πότε να τραγουδήσει για τα κατορθώματα του. Ως aspect, ο ραψωδός το κάνει αυτόματα. Στην πραγματικότητα, ο ιππότης δεν θα πρέπει να γνωρίζει την ύπαρξη του ραψωδού, γι' αυτό η κλάση μπορεί να πάρει την μορφή που είχε πριν:

```
1  public class KnightOfTheRoundTable implements Knight {
2    private String name;
3    private Quest quest;
4
5    public KnightOfTheRoundTable(String name) {
6      this.name = name;
7    }
8
9    public Object embarkOnQuest() throws QuestFailedException {
10     return quest.embark();
11   }
12
13   public void setQuest(Quest quest) {
14     this.quest = quest;
15   }
16
17   public String getName() {
18     return name;
19   }
20 }
```

Listing 6.16: KnightOfTheRoundTable.java

Μέρος IV

Σύστημα διαχείρισης ερευνητικών έργων

ΚΕΦΑΛΑΙΟ **7**

Προδιαγραφές του συστήματος

Υπάρχουν πολλές εμπορικές ή ανοιχτού κώδικα εφαρμογές, όπως το *Microsoft Project* και το *OpenProj*, που προσφέρουν διάφορες υπηρεσίες για τη διαχείριση έργων. Κάποιες από τις υπηρεσίες που περιλαμβάνουν είναι ο χρονοσχεδιασμός του έργου, η διαχείριση του προϋπολογισμού του, η κατανομή των πόρων του οργανισμού που χρησιμοποιεί την εφαρμογή, και η διαχείριση των χρηστών της εφαρμογής.

Όμως κάθε οργανισμός είναι διαφορετικός και έχει διαφορετική δομή. Οι εσωτερικές διαδικασίες (για παράδειγμα η απόδοση μισθοδοσίας και πληρωμή δαπανών), καθώς και η ίδια η ανάπτυξη ενός έργου, γίνονται με διαφορετικό τρόπο από οργανισμό σε οργανισμό. Αυτό έχει ως αποτέλεσμα τα έτοιμα λογισμικά για τη διαχείριση έργων να μην καλύπτουν τις λειτουργικές ανάγκες του οργανισμού, και αυτός να στρέφεται προς την δημιουργία μιας εφαρμογής κατά παραγγελία.

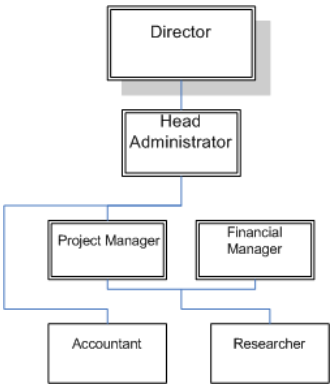
Στα πλαίσια της διπλωματικής εργασίας αναπτύσσεται ένα ***σύστημα διαχείρισης έργων***. Το σύστημα έχει ως σκοπό την εξυπηρέτηση των εργαζομένων του *Ερευνητικού Ακαδημαϊκού Ινστιτούτου Τεχνολογίας Υπολογιστών (Ε.Α.Ι.Τ.Υ.)* που απασχολούνται σε ερευνητικά έργα. Στο παρόν κεφάλαιο γίνεται η περιγραφή του συστήματος η οποία περιλαμβάνει τον ορισμό των προδιαγραφών και των απαιτήσεων του.

7.1 Δομή του οργανισμού

Σε αυτή την ενότητα παρουσιάζεται η δομή του Ερευνητικού Ακαδημαϊκού Ινστιτούτου Τεχνολογίας Υπολογιστών, βάση της οποίας ορίζονται οι προδιαγραφές του συστήματος.

- ***Εργαζόμενοι***: Στο Ε.Α.Ι.Τ.Υ., όπως και στους περισσότερους οργανισμούς υπάρχει μια εσωτερική δομή και ιεραρχία των εργαζομένων. Στα χαμηλότερα επίπεδα ιεραρχίας, βρίσκονται οι ερευνητές, οι οποίοι εργάζονται στα ερευνητικά έργα. Στο επόμενο σκαλί της ιεραρχίας βρίσκονται οι διαχειριστές (*project managers*) κάθε έργου στο οποίο εργάζονται οι ερευνητές. Οι project managers είναι υπεύθυνοι για τη διαχείριση και των καταμερισμό των ερευνητών στα πακέτα εργασίας ενός έργου. Εκτός από τον διαχειριστή του έργου, υπάρχει ο οικονομικός διαχειριστής (*financial manager*), ο οποίος είναι υπεύθυνος για τη σωστή διαχείριση του προϋπολογισμού (*budget*) κάθε έργου. Επιπλέον, υπάρχει ο επικεφαλής διαχειριστής (*head administrator*) του Ε.Α.Ι.Τ.Υ. που επιβλέπει όλα τα ερευνητικά έργα που αναλαμβάνει ο οργανισμός. Στην

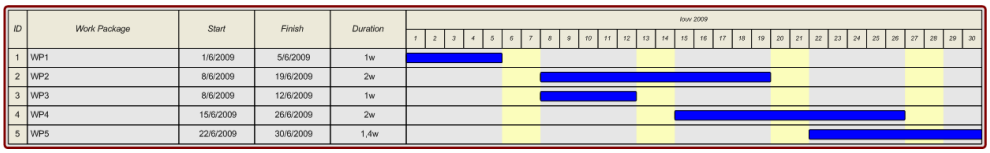
κορυφή της ιεραρχίας βρίσκεται ο διευθυντής (*director*) του οργανισμού. Το Ε.Α.Ι.Τ.Υ. διαθέτει και κάποιον λογιστή(accountant) ο οποίος φροντίζει όλες τις λογιστικές διαδικασίες που αφορούν τον οργανισμό.



Σχήμα 7.1: Ιεραρχία οργανισμού

- ***Ερευνητικά έργα (Projects)***: Το Ε.Α.Ι.Τ.Υ. αναλαμβάνει την ανάπτυξη πολλών ερευνητικών έργων. Κάθε ερευνητικό έργο έχει συγκεκριμένη διάρκεια που ορίζεται από το σχεδιασμό του, συγκεκριμένο προϋπολογισμό, συνολικό και ετήσιο, και χωρίζεται σε *πακέτα εργασίας (work packages)*.

- ***Πακέτα εργασίας (Work Packages)***: Αποτελούν υποσύνολα του έργου και ανατίθενται σε μια συγκεκριμένη ομάδα για την ανάπτυξή τους. Επιπλέον, κάθε πακέτο εργασίας έχει συγκεκριμένη διάρκεια η οποία είναι μικρότερη ή ίση με τη διάρκεια του έργου. Κάποια πακέτα εργασίας, για να ξεκινήσουν την ανάπτυξη, μπορεί να περιμένουν τα αποτελέσματα της ανάπτυξης προηγούμενων, ενώ κάποια άλλα μπορεί να αναπτύσσονται ταυτόχρονα. Στην παρακάτω εικόνα φαίνεται η οργάνωση των πακέτων εργασίας για ένα υποθετικό ερευνητικό έργο το οποίο έχει διάρκεια ενός μήνα και χωρίζεται σε πέντε πακέτα εργασίας:



ID	Work Package	Start	Finish	Duration
1	WP1	1/6/2009	5/6/2009	1w
2	WP2	8/6/2009	19/6/2009	2w
3	WP3	8/6/2009	12/6/2009	1w
4	WP4	15/6/2009	26/6/2009	2w
5	WP5	22/6/2009	30/6/2009	1,4w

Σχήμα 7.2: Παράδειγμα οργάνωσης ενός έργου σε πακέτα εργασίας

7.2 Προδιαγραφές του συστήματος

Πρόκειται για μία εφαρμογή *παγκόσμιου ιστού* όπου οι χρήστες, αφού κάνουν *login* στο σύστημα, έχουν στην διάθεση τους ένα σύνολο από λειτουργίες, οι οποίες διαφέρουν ανάλογα με την κατηγορία χρήστη. Οι λειτουργίες αυτές αφορούν συγκεκριμένες διαδικασίες του οργανισμού και εξετάζονται από διαφορετική σκοπιά για κάθε κατηγορία χρήστη. Παρακάτω παρουσιάζονται οι λειτουργίες για κάθε κατηγορία χρήστη:

Ερευνητές (Researchers)

Για κάθε ερευνητή υπάρχουν διαθέσιμες οι παρακάτω λειτουργίες:

- *Εμφάνιση όλων των έργων στα οποία εργάζεται.* Ο ερευνητής θα μπορεί να επιλέξει ένα από τα έργα στα οποία εργάζεται και να εκτελέσει κάποιες ενέργειες που περιγράφονται πιο κάτω.
- *Συμπλήρωση φόρμας timesheet.* Έχει τη δυνατότητα να συμπληρώσει ένα timesheet για ένα έργο κάποιο συγκεκριμένο μήνα.
- *Κατάθεση αίτησης δαπάνης.* Συμπληρώνει μια φόρμα και κάνει upload τα συνοδευτικά έγγραφα της δαπάνης, η οποία χρεώνεται σε ένα συγκεκριμένο έργο.
- *Αίτηση απόδοσης εξόδων ταξιδιού.* Συμπληρώνει μια φόρμα και κάνει upload τα συνοδευτικά έγγραφα των εξόδων του ταξιδιού, το οποίο χρεώνεται σε συγκεκριμένο έργο.
- *Έλεγχος κατάστασης αιτήσεων δαπάνης και απόδοσης ταξιδίου.* Ο ερευνητής έχει τη δυνατότητα να ελέγχει την κατάσταση της αίτησης του (όπως, από ποιον και πότε έχει εγκριθεί).

Υπεύθυνος έργου (project manager)

Ο υπεύθυνος έργου έχει τη δυνατότητα να χρησιμοποιήσει όλες τις λειτουργίες του ερευνητή. Όμως του παρέχονται κάποιες επιπλέον λειτουργίες που του επιτρέπουν την καλύτερη διαχείριση του έργου. Αυτές είναι οι παρακάτω:

- *Ανάθεση ενός ερευνητή σε ένα πακέτο εργασίας.* Ο υπεύθυνος έργου αναθέτει κάθε μήνα τους ερευνητές που είναι διαθέσιμοι σε πακέτα εργασίας του έργου.
- *Έγκριση αίτησης δαπάνης.* Ελέγχει την αίτηση δαπάνης του ερευνητή και την εγκρίνει.
- *Έγκριση αίτησης απόδοσης εξόδων ταξιδιού.* Ελέγχει την αίτηση απόδοσης ταξιδιών του ερευνητή και την εγκρίνει.
- *Παρακολούθηση αιτήσεων δαπάνης και απόδοσης ταξιδίου.* Ανά πάσα στιγμή μπορεί να παρακολουθήσει την κατάσταση όλων αιτήσεων που αφορούν το συγκεκριμένο έργο.
- *Παρακολούθηση δέσμευσης πόρων.* Ελέγχει τη μέχρι τώρα πορεία του έργου και αν οι διαθέσιμοι πόροι (τόσο οικονομικοί όσο και σε ανθρώπινο δυναμικό) που υπάρχουν έχουν κατανεμηθεί σωστά.

Οικονομικός διαχειριστής (financial manager)

Ο οικονομικός διαχειριστής έχει τη δυνατότητα να χρησιμοποιήσει όλες τις λειτουργίες του ερευνητή. Όμως του παρέχονται κάποιες επιπλέον λειτουργίες που του επιτρέπουν την καλύτερη διαχείριση του έργου. Αυτές είναι οι παρακάτω:

- *Έγκριση αίτησης δαπάνης.* Ελέγχει την αίτηση δαπάνης ενός ερευνητή και την εγκρίνει. Οι αιτήσεις αφορούν όλα τα έργα που έχει αναλάβει ο οργανισμός.
- *Έγκριση αίτησης απόδοσης εξόδων ταξιδιού.* Ελέγχει την αίτηση απόδοσης ταξιδιών ενός ερευνητή και την εγκρίνει. Οι αιτήσεις αφορούν όλα τα έργα που έχει αναλάβει ο οργανισμός.
- *Παρακολούθηση αιτήσεων δαπάνης και απόδοσης ταξιδίου.* Ανά πάσα στιγμή μπορεί να παρακολουθήσει την κατάσταση όλων αιτήσεων που αφορούν όλα τα έργα.

Επικεφαλής διαχειριστής (head administrator)

Ο επικεφαλής διαχειριστής έχει στη διάθεση του τις παρακάτω λειτουργίες:

- *Έγκριση αίτησης δαπάνης.* Ελέγχει την αίτηση δαπάνης ενός ερευνητή και την εγκρίνει. Οι αιτήσεις αφορούν όλα τα έργα που έχει αναλάβει ο οργανισμός.
- *Έγκριση αίτησης απόδοσης εξόδων ταξιδιού.* Ελέγχει την αίτηση απόδοσης ταξιδιών ενός ερευνητή και την εγκρίνει. Οι αιτήσεις αφορούν όλα τα έργα που έχει αναλάβει ο οργανισμός.

Διευθυντής (Director)

Ο διευθυντής έχει στη διάθεση του τις παρακάτω λειτουργίες:

- *Έγκριση αίτησης δαπάνης.* Ελέγχει την αίτηση δαπάνης ενός ερευνητή και την εγκρίνει. Οι αιτήσεις αφορούν όλα τα έργα που έχει αναλάβει ο οργανισμός.

Λογιστής (Accountant)

Ο λογιστής ενημερώνεται για τις εγκεκριμένες αιτήσεις και φροντίζει την διεκπεραίωση τους. Είναι υπεύθυνος ώστε να γίνεται με διαφάνεια η οικονομική διαχείριση του οργανισμού.

7.2.1 Απαιτήσεις συστήματος

Υπάρχουν κάποιες *απαιτήσεις – περιορισμοί* σε σχέση με τις λειτουργίες του συστήματος οι οποίες έχουν άμεση σχέση με την εσωτερική δομή και τον τρόπο λειτουργίας του Ε.Α.Ι.Τ.Υ.

Απαιτήσεις που αφορούν τους ερευνητές

Κάθε ερευνητής του οργανισμού μπορεί να συμμετέχει στην ανάπτυξη πολλών ερευνητικών έργων. Όμως η συνολική του απασχόληση στον οργανισμό, σε όλα τα έργα, δεν πρέπει να ξεπερνά το ποσό [1] που έχει συμφωνηθεί στο συμβόλαιό του. Επιπλέον, ένας διαχειριστής έργου ή ο οικονομικός διαχειριστής μπορεί να είναι ταυτόχρονα και ερευνητής.

[1] *Η απασχόληση ενός εργαζομένου στον οργανισμό μετριέται σε ανθρωπομήνες (**person-months - PM**). Ένας ανθρωπομήνας αντιστοιχεί στον χρόνο που αναλώνει ένα ερευνητής σε ένα έργο όταν εργάζεται με πλήρη απασχόληση.*

Απαιτήσεις που αφορούν τα ερευνητικά έργα

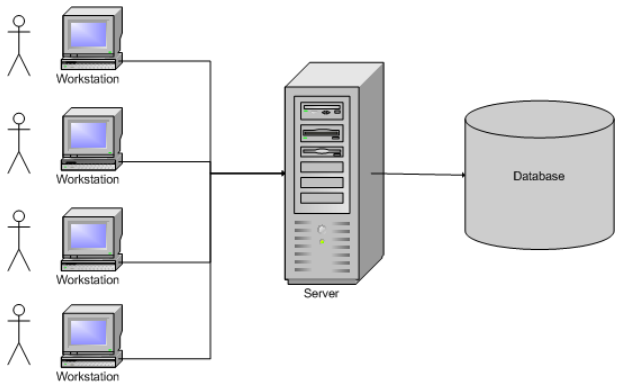
Όταν ο υπεύθυνος έργου πραγματοποιεί το καταμερισμό των ερευνητών σε πακέτα εργασίας στο έργο, θα πρέπει να προσέχει ώστε η συνολική ανάθεση εργασίας σε κάθε πακέτο εργασίας να μη ξεπερνά το συνολικό αριθμό ανθρωπομηνών που προβλέπονται από το συμβόλαιο του έργου για το συγκεκριμένο πακέτο εργασίας. Το ίδιο πρέπει να συμβαίνει και στο σύνολο ανθρωπομηνών όλου του έργου. Επίσης, θα πρέπει να γίνεται προσεκτική έγκριση απόδοσης δαπανών και εξόδων ταξιδίου ώστε να μην υπερβούν το ποσό του ετήσιου και συνολικού προϋπολογισμού.

ΚΕΦΑΛΑΙΟ 8

Σχεδιασμός συστήματος

Σε αυτό το κεφάλαιο παρατίθεται ο σχεδιασμός του συστήματος, ο οποίος αποτελεί μία πρόταση λύσης στις απαιτήσεις και στις προδιαγραφές που έχουν οριστεί (και περιγράφονται στο προηγούμενο κεφάλαιο). Στον σχεδιασμό, ορίζονται οι οντότητες του συστήματος, η δομή του, καθώς και λεπτομερής περιγραφή λειτουργιών. Στην ανάπτυξη λογισμικού, ο σχεδιασμός του συστήματος αποτελεί το συμβόλαιο για την ανάπτυξη του, ανάμεσα στον προγραμματιστή και τον πελάτη. Όταν ολοκληρωθεί το σύστημα, θα πρέπει να μπορεί να εκτελεί ***μόνο*** τις λειτουργίες που αναφέρονται στο σχεδιασμό του (τίποτα παραπάνω, τίποτα λιγότερο).

8.1 Δομή δικτύου



Σχήμα 8.1: Δομή δικτύου του συστήματος

Στο παραπάνω σχήμα παρουσιάζεται η δομή του δικτύου του συστήματος. Πρόκειται για ένα κατανεμημένο σύστημα το οποίο βασίζεται στο μοντέλο ***πελάτη-εξυπηρετητή (client-server model)***. Οι χρήστες συνδέονται μέσω των τερματικών τους στο server όπου τρέχει η εφαρμογή. Η εφαρμογή έχει πρόσβαση σε ένα σύστημα βάσης δεδομένων όπου αποθηκεύονται τα δεδομένα της. Η σύνδεση των χρηστών στην εφαρμογή (και κατά συνέπεια στο server) γίνεται μέσω του διαδικτύου, και δεν περιορίζεται μόνο σε ένα τοπικό δίκτυο.

8.2 Δομή του συστήματος

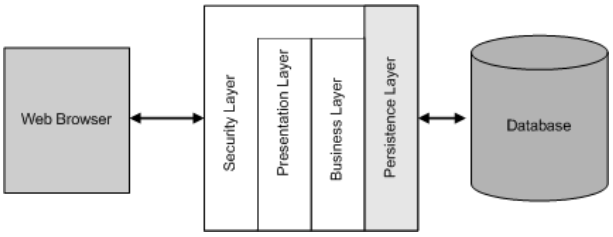
Η δομή του συστήματος αποτελείται από τέσσερα λογικά επίπεδα:

1. ***Το επίπεδο αποθήκευσης (persistence layer)***: Είναι το επίπεδο του συστήματος το οποίο επικοινωνεί με μία σχεσιακή βάση δεδομένων. Σε αυτό το επίπεδο εκτελούνται όλες οι λειτουργίες που έχουν σχέση με την αποθήκευση/ανάκτηση δεδομένων κατευθείαν απο τη βάση, καθώς και η μοντελοποίηση αυτών των δεδομένων. Μεταφέρει δεδομένα από τη βάση δεδομένων στα πιο πάνω επίπεδα και το αντίστροφο.

2. ***Το επίπεδο λειτουργιών (business layer)***: Σε αυτό το επίπεδο περιλαμβάνεται όλη η λειτουργικότητα του συστήματος. Ζητάει ή στέλνει προς αποθήκευση δεδομένα στο επίπεδο αποθήκευσης. Επίσης, παρέχει όλες τις λειτουργίες του συστήματος στο πιο πάνω επίπεδο που είναι το επίπεδο της παρουσίασης.

3. ***Το επίπεδο παρουσίασης (presentation layer)***: Είναι το επίπεδο το οποίο έχει άμεση επαφή και αλληλεπιδρά με τον χρήστη. Δέχεται δεδομένα από το χρήστη, το οποία προωθεί στα κατώτερα επίπεδα του συστήματος, καθώς επίσης εμφανίζει και δεδομένα τα οποία προέρχονται από αυτά. Είναι το επίπεδο μέσω του οποίου ο χρήστης αποκτά πρόσβαση στις λειτουργίες του συστήματος.

4. ***Το επίπεδο ασφάλειας (security layer)***: Επειδή πρόκειται για ένα σύστημα παγκόσμιου ιστού, θα πρέπει η πρόσβαση σε αυτό να είναι ελεγχόμενη. Σε αυτό το επίπεδο ελέγχεται αν κάποιος που προσπαθεί να προσπελάσει το σύστημα είναι εξουσιοδοτημένος, και αν ναι, ποιες λειτουργίες και πόρους του συστήματος έχει δικαίωμα να χρησιμοποιήσει.



Σχήμα 8.2: Δομή του συστήματος

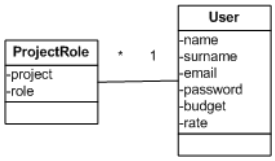
8.3 Οντότητες του συστήματος

Το σύστημα αποτελείται από ένα σύνολο οντοτήτων οι οποίες συσχετίζονται και αλληλεπιδρούν μεταξύ τους. Βρίσκονται στο λογικό επίπεδο της αποθήκευσης. Για κάθε οντότητα που παρουσιάζεται θα υπάρχει ένα σχήμα που θα δείχνει την οντότητα αλλά και τις συσχετίσεις που υπάρχουν με τις μέχρι εκείνη τη στιγμή ορισμένες οντότητες. Έτσι, όταν θα έχουν περιγραφεί όλες οι οντότητες θα έχει έχει χτιστεί το *διάγραμμα οντοτήτων συσχετίσεων (entity-relation diagram – E-R)*. Οι οντότητες αυτές περιγράφονται παρακάτω:

Χρήστες (Users)

Οι χρήστες του συστήματος έχουν τις παρακάτω ιδιότητες:

- *Όνομα*: Το όνομα του χρήστη αναπαριστάται με ένα αλφαριθμητικό μέγιστου μήκους 250 χαρακτήρων.
- *Επώνυμο*: Το επώνυμο του χρήστη αναπαριστάται με ένα αλφαριθμητικό μέγιστου μήκους 250 χαρακτήρων.
- *e-mail*: Η ηλεκτρονική διεύθυνση του χρήστη. Η ηλεκτρονική διεύθυνση του χρήστη χρησιμεύει και ως συνθηματικό (σε συνδυασμό με κάποιον κωδικό πρόσβασης) για την είσοδο του χρήστη στο σύστημα.
- *Κωδικός πρόσβασης (password)*: Χρησιμοποιείται σε συνδυασμό με την ηλεκτρονική διεύθυνση για την είσοδο του χρήστη στο σύστημα.
- *Λίστα από έργα/ρόλοι*: Για κάθε έργο στο οποίο συμμετέχει ο χρήστης εμφανίζεται το όνομα του έργου και ο ρόλος τον οποίο έχει σε αυτό το έργο.
- *Προϋπολογισμός (budget)*: Κάθε χρήστης έχει ένα προσωπικό budget το οποίο ορίζει πόσα χρήματα δικαιούται για αμοιβή, έξοδα ταξιδιών, δαπάνες για εξοπλισμό και λοιπά έξοδα.
- *Rate*: Το rate του χρήστη βάση του οποίου υπολογίζονται οι μηνιαίες αποδοχές του.



Σχήμα 8.3: Η οντότητα χρήστης και οι συσχετίσεις της

Ερευνητικά έργα (projects)

Τα ερευνητικά έργα έχουν τις παρακάτω ιδιότητες:

- *Όνομα*: Το όνομα του χρήστη αναπαριστάται με ένα αλφαριθμητικό μέγιστου μήκους 250 χαρακτήρων.
- *Σύντομο Όνομα (shortname)*: Το σύντομο όνομα του έργου αναπαριστάται με ένα αλφαριθμητικό μήκους 20 χαρακτήρων και συνήθως χρησιμοποιείται για να αναφερθεί κάποιος σε αυτό το έργο εν συντομία.
- *Κατηγορία*: Ένα έργο κατηγοριοποιείται σε ελληνικό ή ευρωπαϊκό έργο.
- *Ημερομηνία Έναρξης*: Η ημερομηνία είναι στη μορφή *"dd/MM/yyyy"*.
- *Ημερομηνία Λήξης*: Όπως και στην ημερομηνία έναρξης, η ημερομηνία είναι στη μορφή *"dd/MM/yyyy"*.

- *Τύπος έργου*: Ανάλογα με την κατηγορία του έργου, ορίζεται ο τύπος του έργου. Για παράδειγμα για Ευρωπαϊκά έργα ο τύπος μπορεί να είναι: STREP, IP, CA, ενώ για Ελληνικά: ΠΕΝΕΔ, ΤΣ.

- *Πρόγραμμα*: Ανάλογα με την κατηγορία του έργου, ορίζεται το πρόγραμμα στο οποίο ανήκει, για παράδειγμα για Ευρωπαϊκά έργα μπορεί να είναι: ICT, PEOPLE.

- *Έργο συντονιστής*: Ένα έργο μπορεί να είναι συντονιστής πολλών άλλων έργων.

- *website*: Στον ιστοσελίδα του έργου μπορεί υπάρχουν διαθέσιμες πληροφορίες για το έργο, όπως το θέμα του έργου, τα άτομα που εργάζονται σε αυτό, τα υποέργα του, κ.α.

- *Συνολικό budget*: Για κάθε έργο ορίζεται ένα συνολικό budget το οποίο αντιστοιχεί στο συνολικό κόστος του έργου, στο οποίο περιλαμβάνονται οι αμοιβές προσωπικού, οι δαπάνες για εξοπλισμό, οι δαπάνες για ταξίδια του προσωπικού, οι αμοιβές των διαχειριστών και λοιπά έξοδα.

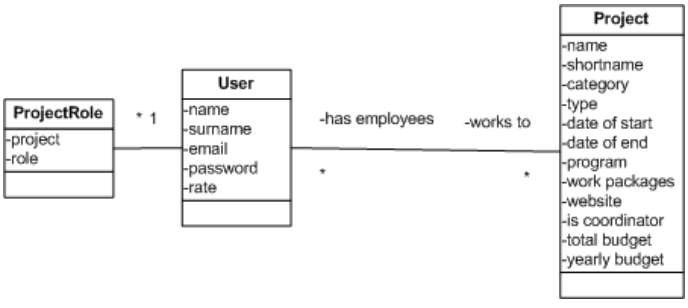
- *Ετήσιο budget*: Αποτελεί υποδιαίρεση του συνολικού budget. Το συνολικό budget μοιράζεται στα χρόνια που διαρκεί η ανάπτυξη του έργου και ανάλογα με την εργασία που εκτελείται κάθε χρόνο.

- *Υπεύθυνος έργου (project manager)*: Είναι υπεύθυνος για όλες τις διαδικασίες που έχουν να κάνουν με την ανάπτυξη του έργου όπως: διαχείριση προσωπικού, ανάθεση ερευνητών σε πακέτα εργασίας, έγκριση δαπανών.

- *Οικονομικός διαχειριστής (financial manager)*: Διαχειρίζεται τα οικονομικά ζητήματα του έργου, και φροντίζει ώστε το συνολικό κόστος του έργου να μην υπερβεί το budget που έχει συμφωνηθεί.

- *Ερευνητική ομάδα (research team)*: Απαρτίζεται από όλους τους ερευνητές που εργάζονται στο ερευνητικό έργο. Κάθε ερευνητής έχει τη δυνατότητα να συμμετέχει ταυτόχρονα στην ερευνητική ομάδα διαφορετικών έργων.

- *Πακέτα εργασίας (work packages)*: Ένα έργο αποτελείται από πολλά πακέτα εργασίας. Έτσι, το έργο χωρίζεται σε υποέργα, κάτι που κάνει πιο εύκολη την ανάπτυξή του έργου γιατί κατανέμεται η συνολική εργασία, και γίνεται πιο εύκολα η διαχείριση και η παρακολούθηση της πορείας του έργου.

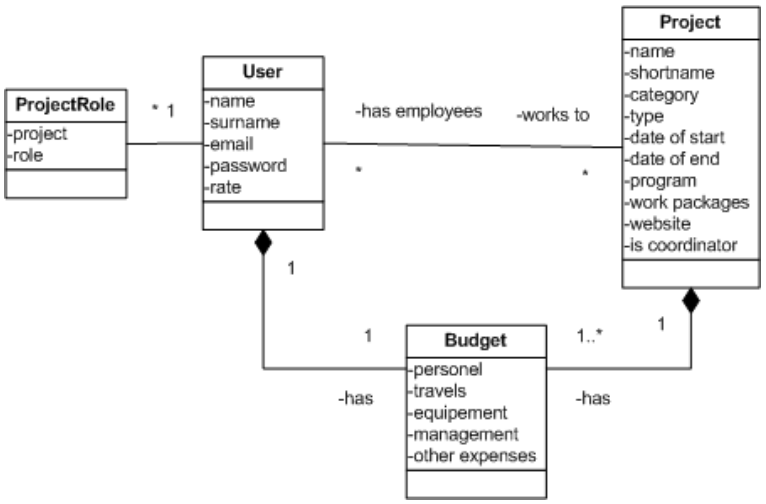


Σχήμα 8.4: Η οντότητα έργο και οι συσχετίσεις της

Προϋπολογισμός (budget)

Ένας προϋπολογισμός έχει τις παρακάτω ιδιότητες:

- *Αμοιβές προσωπικού*: Το ποσό του προϋπολογισμού που προορίζεται για την αμοιβή των ερευνητών του έργου (αν πρόκειται για προϋπολογισμός έργου) ή την αμοιβή του εργαζόμενου (αν πρόκειται για προϋπολογισμός εργαζόμενου).
- *Ταξίδια*: Το ποσό του προϋπολογισμού που προορίζεται για να καλύψει τα έξοδα ταξιδίων που γίνονται για τις ανάγκες της ανάπτυξης ενός έργου.
- *Εξοπλισμός*: Το ποσό του προϋπολογισμού που προορίζεται για την αγορά εξοπλισμού που χρειάζεται για την ανάπτυξη ενός έργου.
- *Management*: Το ποσό του προϋπολογισμού που προορίζεται για την αμοιβή των διαχειριστών του έργου.
- *Λοιπά έξοδα*: Το ποσό του προϋπολογισμού που προορίζεται για έξοδα τα οποία δεν εκπίπτουν σε καμία από τις παραπάνω κατηγορίες.



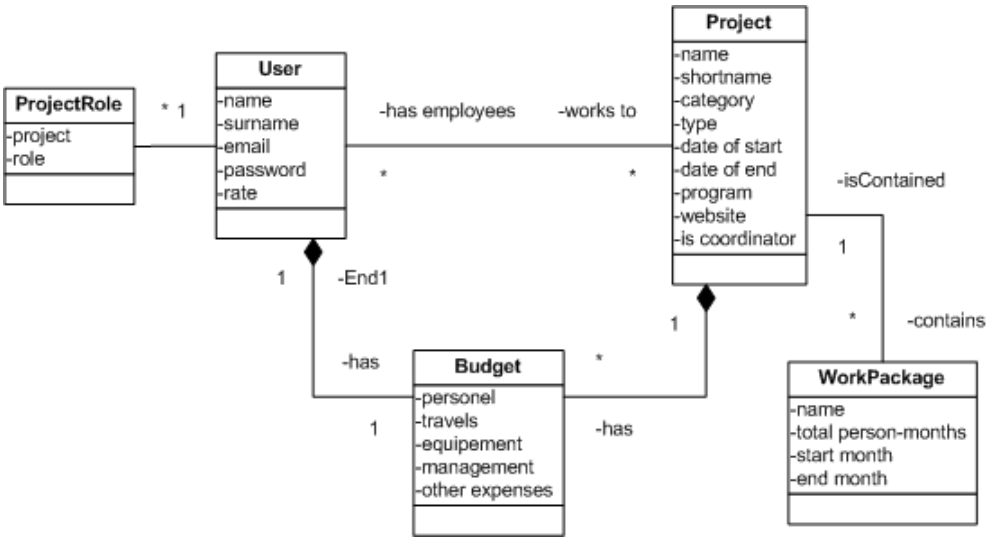
Σχήμα 8.5: Η οντότητα προϋπολογισμός και οι συσχετίσεις της

Πακέτα εργασίας (work packages)

Τα πακέτα εργασίας έχουν τις παρακάτω ιδιότητες:

- *Όνομα*: Το όνομα του πακέτου εργασίας ενός έργου. Πρέπει να εισάγεται μαζί με το πρόθεμα "WP#:" όπου "#" είναι ο αριθμός του πακέτου εργασίας.
- *Συνολικοί ανθρωπομήνες*: Το ποσό της εργασίας που πρέπει να δαπανηθεί για την ολοκλήρωσή του και δεν θα πρέπει σε καμία περίπτωση να γίνει υπέρβασή του κατά τη διαδικασία ανάπτυξης.
- *Μήνας έναρξης*: Είναι ένας αριθμός που αντιστοιχεί στον ***αριθμητικό*** (όχι ημερολογιακό) μήνα του έργου, στον οποίο ξεκινούν οι εργασίες του συγκεκριμένου πακέτου εργασίας.

- *Μήνας Λήξης*: Είναι ένας αριθμός που αντιστοιχεί στον ***αριθμητικό*** (όχι ημερολογιακό) μήνα του έργου, στον οποίο λήγουν οι εργασίες του συγκεκριμένου πακέτου εργασίας.



Σχήμα 8.6: Η οντότητα πακέτο εργασίας και οι συσχετίσεις της

Timesheets

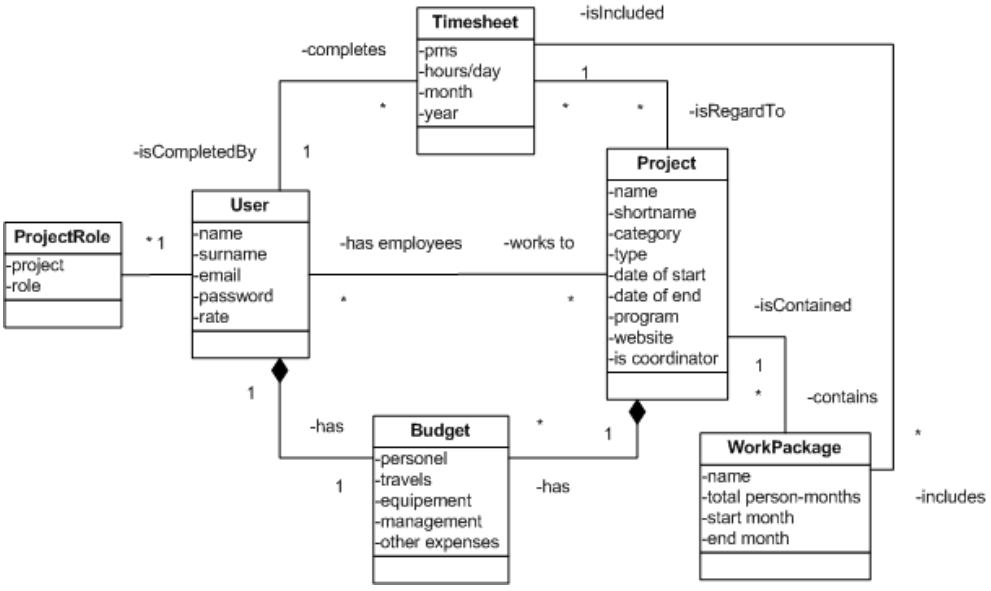
Τα timesheets έχουν τις παρακάτω ιδιότητες:

- *Χρήστης*: Ένας χρήστης καταθέτει το timesheet για τις ώρες που εργάστηκε σε κάποιο έργο.
- *Έργο*: Το έργο για το οποίο κατατίθεται το timesheet.
- *Ανθρωπομήνες ανά πακέτο εργασίας*: Ο διαχειριστής ορίζει για κάθε χρήστη τους ανθρωπομήνες που θα εργαστεί σε κάθε πακέτο εργασίας για το συγκεκριμένο μήνα.
- *Ώρες ανά ημέρα ανά πακέτο εργασίας*: Ο χρήστης εισάγει της ώρες που εργάζεται για κάθε μέρα του μήνα σε κάθε πακέτο εργασίας.
- *Χρονολογία*: Ο ημερολογιακός μήνας και το έτος που ο χρήστης καταθέτει το timesheet.

Αίτηση δαπάνης

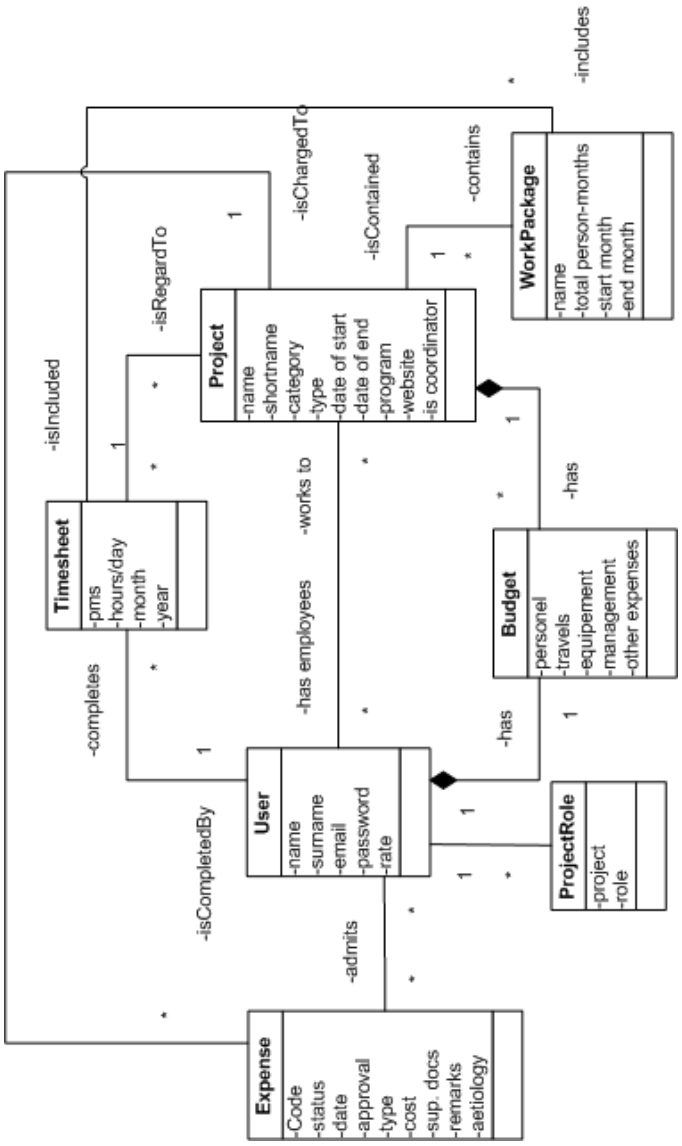
Η αίτηση δαπάνης έχει τις παρακάτω ιδιότητες:

- *Χειριστής*: Είναι ο χρήστης του συστήματος ο οποίος καταθέτει την αίτηση για τη δαπάνη.
- *Έργο*: Το έργο στο οποίο θα χρεωθεί η δαπάνη που κατατίθεται.



Σχήμα 8.7: Η οντότητα timesheet και οι συσχετίσεις της

- *Κωδικός δαπάνης*: Ένα μοναδικό αλφαριθμητικό το οποίο χρησιμοποιείται για την αναφορά στη συγκεκριμένη αίτηση δαπάνης.
- *Κατάσταση αίτησης δαπάνης*: Όταν κατατεθεί η αίτηση, θα πρέπει να εγκριθεί από έναν αριθμό ατόμων. Έτσι η αίτηση, μέχρι να εγκριθεί, περνάει από διάφορες φάσεις κατάστασης.
- *Είδος δαπάνης*: Αναφέρεται ακριβώς το αντικείμενο που αγοράστηκε.
- *Κόστος*: Το κόστος της δαπάνης για το είδος που περιγράφεται.
- *Παρατηρήσεις*: Ο χρήστης που καταθέτει την αίτηση δαπάνης μπορεί να γράψει κάποιες παρατηρήσεις που αφορούν τη δαπάνη.
- *Έγκριση*: Σημειώνεται κάθε χρήστης που εγκρίνει τη δαπάνη καθώς και την ημερομηνία που το κάνει αυτό.
- *Έγγραφα*: Ο χρήστης που καταθέτει τη δαπάνη πρέπει να συμπεριλάβει και τα συνοδευτικά της έγγραφα, όπως τιμολόγια και αποδείξεις.
- *Ημερομηνία*: Η ημερομηνία που καταθέτει ο χρήστης την αίτηση δαπάνης.
- *Αιτιολογία*: Αναφέρεται ο λόγος δημιουργίας αυτής της δαπάνης, όπως εξοπλισμός, φωτοτυπίες βιβλίων.
- *Χρήστης*: Αφορά τον πάγιο εξοπλισμό, και είναι το άτομο που το χρησιμοποιεί.
- *Δικαιούχος*: Ο δικαιούχος είναι ο άνθρωπος ο οποίος θα παραλάβει τα χρήματα που καλύπτουν τα έξοδα της δαπάνης.



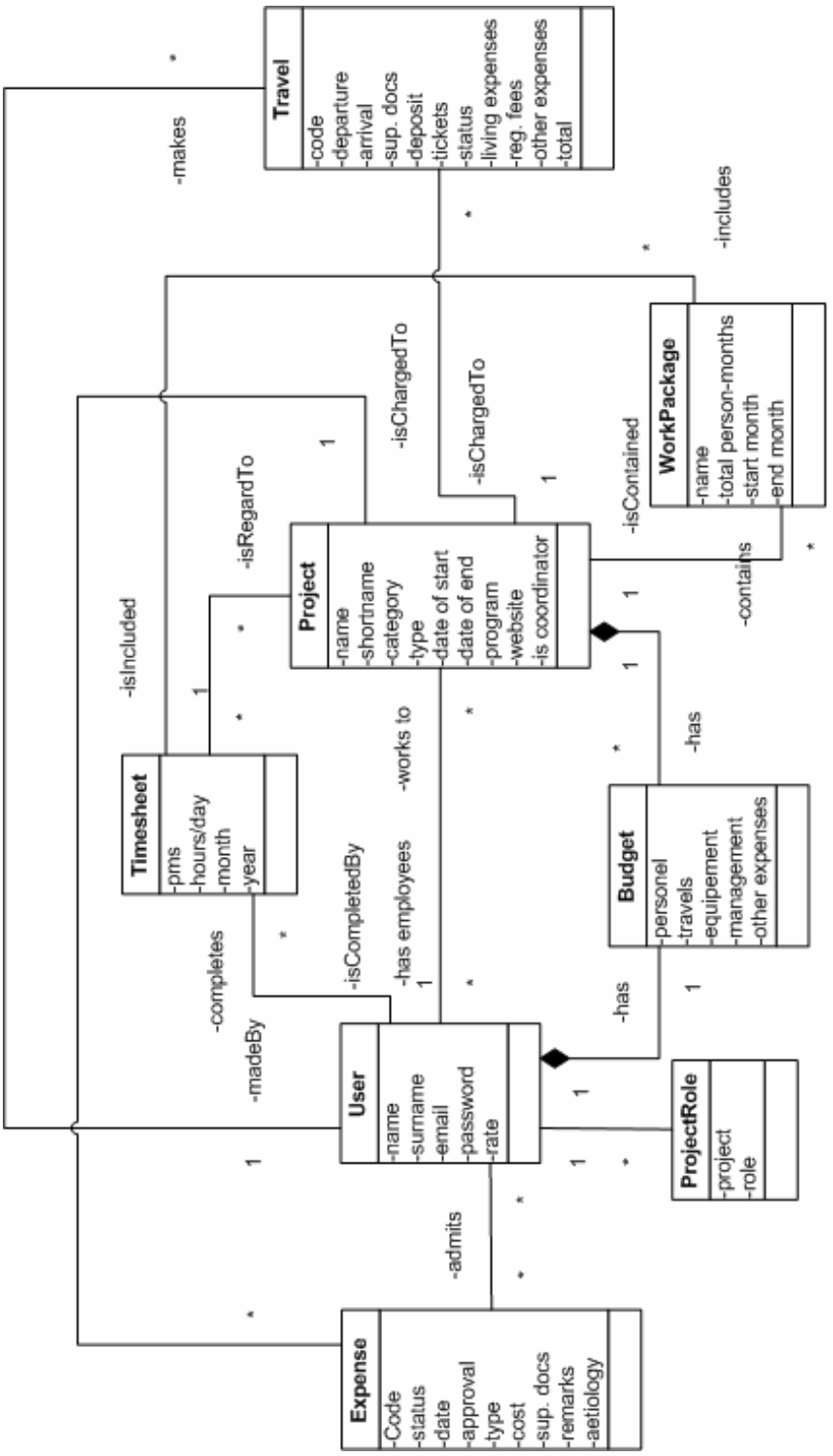
Σχήμα 8.8: Η οντότητα δαπάνη και οι συσχετίσεις της

Αίτηση απόδοσης εξόδων ταξιδιού

Η αίτηση απόδοσης εξόδων ταξιδιου είναι η τελευταία οντότητα του συστήματος. Με της περιγραφή της, θα έχουν οριστεί πλήρως όλες οι οντότητες του συστήματος, και οι ενσωμάτωσή της στο διάγραμμα οντοτήτων έχει ως αποτέλεσμα την ολοκλήρωση του διαγράμματος οντοτήτων συσχετίσεων του συστήματος. Η οντότητα έχει τις παρακάτω ιδιότητες:

- *Εντολέας*: Είναι ο άνθρωπος ο οποίος έδωσε την εντολή μετακίνησης.
- *Χρήστης*: Ο χρήστης ο οποίος πραγματοποίησε το ταξίδι για το οποίο κατατίθεται η αίτηση απόδοσης εξόδων.
- *Έργο*: Το έργο στο οποίο θα χρεωθούν τα έξοδα ταξιδιού.
- *Κωδικός εντολής*: Ένα μοναδικό αλφαριθμητικό το οποίο χρησιμοποιείται για την αναφορά στη συγκεκριμένη αίτηση απόδοσης εξόδων ταξιδιού.

- *Κατάσταση αίτησης*: Όταν κατατεθεί η αίτηση, θα πρέπει να εγκριθεί από έναν αριθμό ατόμων. Έτσι η αίτηση, μέχρι να εγκριθεί, περνάει από διάφορες φάσεις κατάστασης.
- *Ημερομηνία Αναχώρησης*: Η ημερομηνία είναι στη μορφή *"dd/MM/yyyy"*.
- *Ημερομηνία Επιστροφής*: Όπως και στην ημερομηνία αναχώρησης, η ημερομηνία είναι στη μορφή *"dd/MM/yyyy"*.
- *Προκαταβολή*: Το ποσό που έχει λάβει ως προκαταβολή ο χρήστης που πρόκειται να ταξιδέψει.
- *Εισιτήρια*: Το κόστος των εισιτηρίων όλου του ταξιδιού.
- *Έξοδα διαβίωσης*: Τα έξοδα διαβίωσης του χρήστη κατά τη διάρκεια του ταξιδιού του.
- *Δικαιολογητικά*: Ο χρήστης που καταθέτει την αίτηση πρέπει να συμπεριλάβει και τα συνοδευτικά της έγγραφα, όπως τιμολόγια και αποδείξεις.
- *Ημερομηνία*: Η ημερομηνία που καταθέτει ο χρήστης την αίτηση.
- *Registration Fees*: Αφορά τα έξοδα συμμετοχής σε εκδηλώσεις, όπως συνέδρια, στα οποία χρειαζόταν να παραβρεθεί στα πλαίσια της ανάπτυξης του έργου.
- *Διάφορα έξοδα*: Τα διάφορα έξοδα αφορούν έξοδα τα οποία δεν ανήκουν σε καμία άλλη απο τις παραπάνω κατηγορίες.
- *Σύνολο*: Το συνολικό κόστος του ταξιδιού.



Σχήμα 8.9: Διάγραμμα οντοτήτων συσχετίσεων του συστήματος

8.4 Χρήστες και ρόλοι

Σύμφωνα με τις προδιαγραφές και τις απαιτήσεις που έχουν τεθεί για το σύστημα, οι χρήστες του συστήματος έχουν διακεκριμένους ρόλους οι οποίοι δημιουργήθηκαν βάση της ιεραρχίας και της δομής του οργανισμού (που περιγράφεται στο προηγούμενο κεφάλαιο). Σε κάθε ρόλο αντιστοιχούν κάποιες λειτουργίες, και η αντιστοιχία αυτή έχει οριστεί στις προδιαγραφές του συστήματος. Στα USE CASES του σχεδιασμού σε κάθε λειτουργία αναφέρεται ποια κατηγορία χρηστών έχει πρόσβαση. Οι ρόλοι των χρηστών του συστήματος είναι οι παρακάτω:

- ***Ερευνητές (Researchers)***
- ***Υπεύθυνος έργου (Project manager)***
- ***Οικονομικός διαχειριστής (Financial manager)***
- ***Επικεφαλής διαχειριστής (Head administrator)***
- ***Διευθυντής (Director)***
- ***Λογιστής (Accountant)***

8.5 Περιγραφή Λειτουργιών (Use cases)

Παρακάτω περιγράφονται οι λειτουργίες του συστήματος με την χρήση των Use cases. Επιπλέον, παρουσιάζονται διαγράμματα ροής για τις διαδικασίες της αίτησης δαπάνης και της αίτησης απόδοσης εξόδων ταξιδιών.

Εισαγωγή ενός χρήστη

- ***Επιθυμητό αποτέλεσμα***: Εισαγωγή στο σύστημα ενός νέου χρήστη.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να προσθέσει ένα νέο χρήστη στο σύστημα.
- ***Κριτήριο τερματισμού***: Ο νέος χρήστης αποθηκεύεται στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την προσθήκη ενός νέου χρήστη στο σύστημα. Με την επιλογή αυτή εμφανίζεται μια φόρμα εισαγωγής νέου χρήστη, η οποία πρέπει να συμπληρωθεί από τον χρήστη. Τα στοιχεία της φόρμας είναι όλα απαραίτητα για την επιτυχημένη υποβολή της αίτησης. Με την υποβολή της φόρμας ένας νέος χρήστης αποθηκεύεται στο σύστημα.
- ***Απαραίτητες πληροφορίες***: Όνομα, επώνυμο, rate, email, κωδικός πρόσβασης, προσωπικό budget.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε Managers, head administrator και Director.

Επεξεργασία ενός χρήστη

- ***Επιθυμητό αποτέλεσμα***: Η αλλαγή των στοιχείων ενός χρήστη.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να αλλάξει τα στοιχεία ενός χρήστη.
- ***Κριτήριο τερματισμού***: Τα στοιχεία του χρήστη αποθηκεύονται στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την επεξεργασία των στοιχείων ενός χρήστη.. Με την επιλογή αυτή εμφανίζεται μια φόρμα επεξεργασία χρήστη, η οποία πρέπει να συμπληρωθεί από τον χρήστη. Τα στοιχεία της φόρμας είναι συμπληρωμένα απο τα υπάρχοντα στοιχεία του χρήστη και αλλάζονται μόνο αυτά που επιθυμεί ο χρήστης.
- ***Απαραίτητες πληροφορίες***: Οποιαδήποτε από τα παρακάτω στοιχεία που μπορεί αλλαχτούν : όνομα, επώνυμο, rate, email, κωδικός πρόσβασης, προσωπικό budget.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε Managers, head administrator και Director.

Εισαγωγή ενός έργου

- ***Επιθυμητό αποτέλεσμα***: Εισαγωγή στο σύστημα ενός νέου ερευνητικού έργου.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να προσθέσει ένα νέο ερευνητικό έργο στο σύστημα.
- ***Κριτήριο τερματισμού***: Το νέο ερευνητικό έργο αποθηκεύεται στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την προσθήκη ενός νέου ερευνητικού έργου. Με την επιλογή αυτή εμφανίζεται μια φόρμα εισαγωγής νέου έργου, η οποία πρέπει να συμπληρωθεί από τον χρήστη. Τα στοιχεία της φόρμας είναι όλα απαραίτητα για την επιτυχημένη υποβολή της αίτησης. Με την υποβολή της φόρμας το νέο έργο αποθηκεύεται στη βάση.
- ***Απαραίτητες πληροφορίες***: Όνομα έργου, σύντομο όνομα, κατηγορία έργου, ημερομηνία έναρξης, ημερομηνία λήξης, τύπος έργου, πρόγραμμα έργου, website, αν είναι έργο συντονιστής, συνολικό budget, πακέτα εργασίας, υπεύθυνος έργου, οικονομικός διαχειριστής, ερευνητική ομάδα.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε Managers, head administrator και Director.

Προσθήκη ερευνητών σε ένα έργο

- ***Επιθυμητό αποτέλεσμα***: Εισαγωγή νέων ερευνητών σε ένα έργο.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να προσθέσει ερευνητές σε ένα έργο.
- ***Κριτήριο τερματισμού***: Η ερευνητική ομάδα του έργου αποθηκεύεται στη βάση.

- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την προσθήκη ερευνητών σε ένα εργο. Με την επιλογή αυτή εμφανίζεται μια φόρμα εισαγωγής ερευνητή σε ένα έργο, η οποία περιέχει ένα drop down box με όλους τους διαθέσιμους χρήστες. Με την υποβολή της φόρμας η ερευνητική ομάδα του έργου αποθηκεύεται στη βάση.

- ***Απαραίτητες πληροφορίες***: Το ονοματεπώνυμο του ερευνητή που προστίθεται στο έργο.

- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη στους managers.

Προβολή συγκεκριμένου έργου

- ***Επιθυμητό αποτέλεσμα***: Προβολή στην οθόνη λεπτομερειών ενός συγκεκριμένου έργου.

- ***Σημείο εισαγωγής***: Ο χρήστης επιλέγει ένα συγκεκριμένο έργο για εμφάνιση, απο μία λίστα των διαθέσιμων για το χρήστη έργων.

- ***Κριτήριο τερματισμού***: Εμφάνιση λεπτομερειών για το έργο που επέλεξε ο χρήστης.

- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει από μία λίστα με ερευνητικά έργα, εκείνο για το οποίο θέλει να δει λεπτομέρειες και τις διαθέσιμες υπηρεσίες. Με την επιλογή αυτή εμφανίζονται λεπτομέρειες για το έργο, καθώς και οι ενέργειες που μπορεί να κάνει για αυτό το έργο.

- ***Απαραίτητες πληροφορίες***: Καμία.

- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε όλους.

Καθορισμός ετήσιου budget ενός έργου

- ***Επιθυμητό αποτέλεσμα***: Καθορισμός των ετήσιων budget ενός έργου και αποθήκευσή τους στη βάση.

- ***Σημείο εισαγωγής***: Ο χρήστης από την οθόνη ενός συγκεκριμένου έργου επιλέγει τον καθορισμό των ετήσιων budgets του.

- ***Κριτήριο τερματισμού***: Αποθήκευση των budgets ενός έργου στη βάση.

- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει από την οθόνη ενός συγκεκριμένου έργου, να καθορίσει τα ετήσια budgets του. Με την επιλογή αυτή εμφανίζεται μια φόρμα που περιέχει το συνολικό και τα ετήσια budgets, επεξεργάζεται τις τιμές τους και με την αποστολή της φόρμας τα στοιχεία αποθηκεύονται στη βάση.

- ***Απαραίτητες πληροφορίες***: Συνολικό και ετήσια budgets.

- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε όλους του χρήστες εκτός από τους researchers.

Καθορισμός person-months ερευνητή

- ***Επιθυμητό αποτέλεσμα***: Καθορισμός των person-months ενός ερευνητή για ένα συγκεκριμένο έργο για κάποιον μήνα του έργου.
- ***Σημείο εισαγωγής***: Ο χρήστης από την οθόνη ενός συγκεκριμένου έργου επιλέγει τον καθορισμό των person-months ερευνητών.
- ***Κριτήριο τερματισμού***: Αποθήκευση των person-months ενός χρήστη, για ένα έργο, κάποιο συγκεκριμένο μήνα, στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει από την οθόνη ενός συγκεκριμένου έργου, να καθορίσει τα person-months ενός ερευνητή που ανήκει στην ερευνητική ομάδα του έργου για ένα συγκεκριμένο μήνα. Με την επιλογή αυτή εμφανίζεται μια φόρμα με δύο drop-down boxes στα οποία επιλέγει ημερολογιακό μήνα και ερευνητή. Μετά από αυτή την επιλογή, εμφανίζεται μια φόρμα στην οποία πρέπει να συμπληρώσει, για κάθε πακέτο εργασίας που είναι διαθέσιμο το συγκεκριμένο μήνα, πόσα person-months θα εργαστεί ο ερευνητής. Με την αποστολή αυτή της φόρμας, τα person-months αποθηκεύονται στη βάση.
- ***Απαραίτητες πληροφορίες***: Τα person-months που θα εργαστεί ο ερευνητής, για κάθε πακέτο εργασίας .
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη στον project manager (υπεύθυνο έργου).

Κατάθεση timesheet

- ***Επιθυμητό αποτέλεσμα***: Αποθήκευση στη βάση ενός timesheet.
- ***Σημείο εισαγωγής***: Ο χρήστης από την οθόνη ενός συγκεκριμένου έργου επιλέγει την κατάθεση ενός timesheet.
- ***Κριτήριο τερματισμού***: Αποθήκευση του timesheet στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει από την οθόνη ενός συγκεκριμένου έργου, να καταθέσει ένα timesheet για ένα συγκεκριμένο μήνα. Με την επιλογή αυτή εμφανίζεται μια φόρμα με ένα drop-down box στα οποία επιλέγει τον ημερολογιακό μήνα. Μετά από αυτή την επιλογή, εμφανίζεται μια φόρμα στην οποία πρέπει να συμπληρώσει, για κάθε πακέτο εργασίας που είναι διαθέσιμο το συγκεκριμένο μήνα, πόσες ώρες την ημέρα εργάστηκε. Με την αποστολή αυτή της φόρμας, το timesheet αποθηκεύεται στη βάση.
- ***Απαραίτητες πληροφορίες***: Οι ώρες την ημέρα που εργάστηκε ο χρήστης, για κάθε πακέτο εργασίας .
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε όλους.

Παρακολούθηση πακέτων εργασίας ενός έργου

- ***Επιθυμητό αποτέλεσμα***: Εμφάνιση των στοιχείων όλων τον πακέτων εργασίας για κάθε μήνα και συνολικά για όλη τη διάρκεια του έργου.
- ***Σημείο εισαγωγής***: Ο χρήστης από την οθόνη ενός συγκεκριμένου έργου επιλέγει την παρακολούθηση των πακέτων εργασίας του έργου.
- ***Κριτήριο τερματισμού***: Εμφάνιση των στοιχείων των πακέτων εργασίας στην οθόνη.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει από την οθόνη ενός συγκεκριμένου έργου, να δει την συνολική εικόνα του έργου μέσω των πακέτων εργασίας του. Με την επιλογή αυτή εμφανίζονται πίνακες για τα πακέτα εργασίας που περιέχουν, για κάθε μήνα τους, τα προγραμματισμένα person-months, τα person-months που έχει ορίσει ο υπεύθυνος έργου για κάθε μήνα, και τα πραγματικά person-months που δήλωσαν οι χρήστες ό-τι εργάστηκαν. Επιπλέον, υπάρχει ένα συγκεντρωτικός πίνακας που έχει τα συνολικά person months για κάθε πακέτο εργασίας.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη στον project manager.

Παρακολούθηση του έργου κάποιο μήνα

- ***Επιθυμητό αποτέλεσμα***: Εμφάνιση των person-months όλων των μελών του έργου για ένα συγκεκριμένο μήνα.
- ***Σημείο εισαγωγής***: Ο χρήστης από την οθόνη της συνολικής παρακολούθησης ενός συγκεκριμένου έργου επιλέγει έναν συγκεκριμένο μήνα του έργου.
- ***Κριτήριο τερματισμού***: Εμφάνιση στην οθόνη των person-months, για ένα συγκεκριμένο μήνα όλων των μελών του έργου.
- ***Περιγραφή λειτουργίας***:Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει από την οθόνη συνολικής παρακολούθησης ενός συγκεκριμένου έργου ένα συγκεκριμένο μήνα για τον οποίο θέλει να δει τα συνολικά person-months όλων των μελών του έργου. Επιλέγοντας τον μήνα, εμφανίζεται στην οθόνη ένας πίνακας που περιέχει όλα τα μέλη που έχουν εργαστεί το συγκεκριμένο μήνα στο έργο καθώς και τα person-months τα οποία έχουν εργαστεί.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη στον project manager.

Προσθήκη αίτησης δαπάνης

- ***Επιθυμητό αποτέλεσμα***: Εισαγωγή στο σύστημα μιας νέας αίτησης για χρηματοδότηση δαπανών, η νέα αίτηση δημιουργείται με αρχική κατάσταση ***pending***. Ο υπεύθυνος έργου ενημερώνεται για τη νέα αίτηση, για να την εγκρίνει.

- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να προσθέσει μια νέα αίτηση δαπάνης στο σύστημα.
- ***Κριτήριο τερματισμού***: Η νέα αίτηση αποθηκεύεται στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την προσθήκη νέας αίτησης. Με την επιλογή αυτή εμφανίζεται μια φόρμα εισαγωγής δαπάνης, η οποία πρέπει να συμπληρωθεί από τον χρήστη. Τα στοιχεία της φόρμας είναι όλα απαραίτητα για την επιτυχημένη υποβολή της αίτησης. Με την υποβολή της αίτησης το νέο αίτημα έγκρισης δαπάνης αποθηκεύεται στη βάση.
- ***Απαραίτητες πληροφορίες***: Αιτιολογία δαπάνης, είδος και κόστος, συνοδευτικά έγγραφα, χρήστης εξοπλισμού, χειριστής εξοπλισμού, δικαιούχος.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε Researchers, Managers και Director.

Ολοκλήρωση αίτησης δαπάνης

- ***Επιθυμητό αποτέλεσμα***: Ολοκλήρωση της αίτησης μέσω επεξεργασία αυτής ως προς τα κόστη, μετατρέποντας την κατάσταση σε ***"approved"***. Ενημερώνεται ο χρήστης που δημιούργησε την αίτηση για την έγκρισή της.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να προχωρήσει σε ολοκλήρωση μιας αίτησης, που σημαίνει ότι προωθεί την ολοκληρωμένη αίτηση στον αντίστοιχο προϊστάμενό τους για περαιτέρω έγκριση.
- ***Κριτήριο τερματισμού***: Η επεξεργασμένη αίτηση αποθηκεύεται στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την επεξεργασία μιας αίτησης για την οποία έχει πάρει έγκριση από τα προηγούμενα επίπεδα και η συγκεκριμένη αίτηση έχει ολοκληρωθεί. Με την επιλογή αυτή εμφανίζεται μια φόρμα επεξεργασίας κόστους δαπάνης, η οποία πρέπει να συμπληρωθεί από το χρήστη. Τα κόστη που είχαν συμπληρωθεί αρχικά εμφανίζονται στα κατάλληλα πεδία της φόρμας. Τα στοιχεία της φόρμας είναι όλα απαραίτητα για την επιτυχημένη υποβολή της αίτησης. Με την υποβολή της αίτησης η δαπάνη αποθηκεύεται στη βάση.
- ***Απαραίτητες πληροφορίες***: Αιτιολογία δαπάνης, είδος και κόστος, συνοδευτικά έγγραφα, χρήστης εξοπλισμού, χειριστής εξοπλισμού, δικαιούχος.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε Researchers, Managers και Director.

Ενημέρωση κατάστασης αιτήσεων

- ***Επιθυμητό αποτέλεσμα***: Εμφάνιση στον πίνακα δαπανών των αιτήσεων όπως αυτές έχουν τη συγκεκριμένη χρονική στιγμή.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να ενημερωθεί για τις καταστάσεις στις οποίες βρίσκονται οι αιτήσεις.

- ***Κριτήριο τερματισμού***: Ενημέρωση του πίνακα δαπανών του χρήστη.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την ενημέρωση του πίνακα δαπανών. Η ενημέρωση πραγματοποιείται μέσω σύνδεσης με τη βάση και διαβάζοντας από αυτήν τις αιτήσεις ως έχουν εκείνη τη χρονική στιγμή.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη και στα έξι είδη χρηστών.

Έγκριση αίτησης δαπάνης

- ***Επιθυμητό αποτέλεσμα***: Έγκριση της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να εγκρίνει την αίτηση.
- ***Κριτήριο τερματισμού***: Αποθήκευση της εγκεκριμένης αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την έγκριση μιας από τις αιτήσεις που υπάρχουν στον πίνακα δαπανών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή έγκρισής της ("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε όλους εκτός από τους accountant και researchers.

Απόρριψη αίτησης δαπάνης

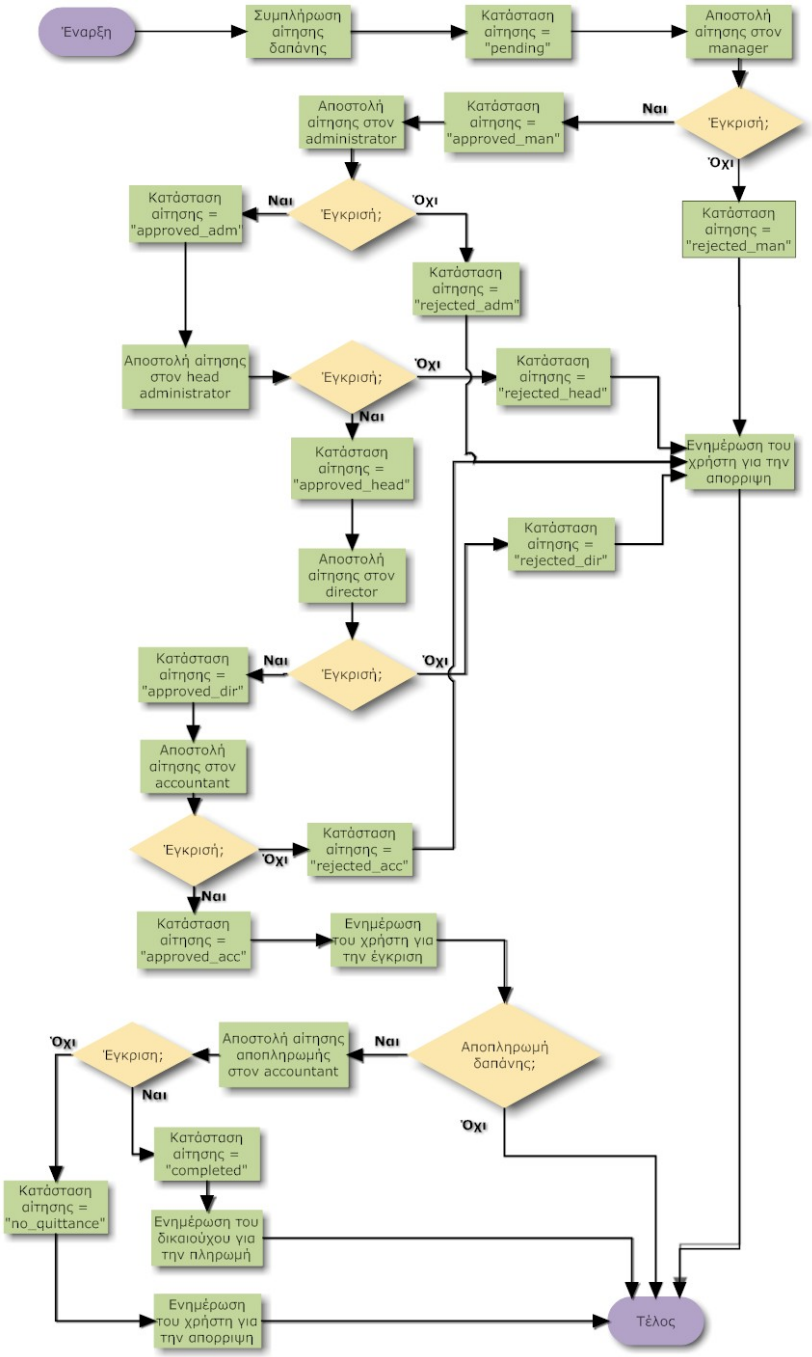
- ***Επιθυμητό αποτέλεσμα***: Απόρριψη της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να απορρίψει την αίτηση.
- ***Κριτήριο τερματισμού***: Αποθήκευση της απορριφθείσας αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την απόρριψη μιας από τις αιτήσεις που υπάρχουν στον πίνακα δαπανών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή απόρριψή της ("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε όλους εκτός από τους accountant και researchers.

Έγκριση αποπληρωμής δαπάνης

- ***Επιθυμητό αποτέλεσμα***: Έγκριση αποπληρωμής της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να εγκρίνει την πλήρη αποπληρωμή μιας αίτησης.
- ***Κριτήριο τερματισμού***: Αποθήκευση της εγκεκριμένης για αποπληρωμή αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την έγκριση αποπληρωμής μιας από τις αιτήσεις που υπάρχουν στον πίνακα δαπανών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή έγκρισής της("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη μόνο σε accountant. Η κατάστασή της αίτησης αλλάζει σε "completed" που σημαίνει ότι η αποπληρωμή της δαπάνης έχει εγκριθεί.

Απόρριψη αποπληρωμής δαπάνης

- ***Επιθυμητό αποτέλεσμα***: Απόρριψη αποπληρωμής της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να απορρίψει την αποπληρωμή μιας αίτησης.
- ***Κριτήριο τερματισμού***: Αποθήκευση της απορριφθείσας για αποπληρωμή αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την απόρριψη αποπληρωμής μιας από τις αιτήσεις που υπάρχουν στον πίνακα δαπανών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή έγκρισής της("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη μόνο σε accountant. Η κατάστασή της αίτησης αλλάζει σε "no_quittance" που σημαίνει ότι η αποπληρωμή της δαπάνης έχει απορριφθεί.



Σχήμα 8.10: Διάγραμμα ροής αίτησης δαπάνης

Προσθήκη αίτησης απόδοσης εξόδων ταξιδιού

- ***Επιθυμητό αποτέλεσμα***: Εισαγωγή στο σύστημα μιας νέας αίτησης χρηματοδότησης ταξιδιού, η νέα αίτηση δημιουργείται με αρχική κατάσταση ***pending***. Ο υπεύθυνος έργου ενημερώνεται για τη νέα αίτηση, για να την εγκρίνει.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να προσθέσει μια νέα αίτηση δαπάνης στο σύστημα.
- ***Κριτήριο τερματισμού***: Η νέα αίτηση αποθηκεύεται στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την προσθήκη νέας αίτησης. Με την επιλογή αυτή εμφανίζεται μια φόρμα εισαγωγής ταξιδιού, η οποία πρέπει να συμπληρωθεί από τον χρήστη. Τα στοιχεία της φόρμας είναι όλα απαραίτητα για την επιτυχημένη υποβολή της αίτησης. Με την υποβολή της αίτησης το νέο ταξίδι αποθηκεύεται στη βάση.
- ***Απαραίτητες πληροφορίες***: Αιτιολογία, προορισμός, ημερομηνία αναχώρησης/ημερομηνία επιστροφής, κόστος εισιτηρίων, κόστος διαμονής, κόστος εγγραφής, ημερήσιο κόστος και λοιπά έξοδα, καθώς και συνοδευόμενα έντυπα.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε Researchers, Managers και Director.

Ολοκλήρωση αίτησης απόδοσης εξόδων ταξιδιού

- ***Επιθυμητό αποτέλεσμα***: Ολοκλήρωση της αίτησης μέσω επεξεργασία αυτής ως προς τα κόστη, μετατρέποντας την κατάσταση σε ***"approved"***. Ενημερώνεται ο χρήστης που δημιούργησε την αίτηση για την έγκρισή της.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να προχωρήσει σε ολοκλήρωση μιας αίτησης, που σημαίνει ότι προωθεί την ολοκληρωμένη αίτηση στον αντίστοιχο προϊστάμενό τους για περαιτέρω έγκριση.
- ***Κριτήριο τερματισμού***: Η επεξεργασμένη αίτηση αποθηκεύεται στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την επεξεργασία μιας αίτησης για την οποία έχει πάρει έγκριση από τα προηγούμενα επίπεδα και η συγκεκριμένη αίτηση έχει ολοκληρωθεί. Με την επιλογή αυτή εμφανίζεται μια φόρμα επεξεργασίας κόστους ταξιδιού, η οποία πρέπει να συμπληρωθεί από το χρήστη. Τα κόστη που είχαν συμπληρωθεί αρχικά εμφανίζονται στα κατάλληλα πεδία της φόρμας. Τα στοιχεία της φόρμας είναι όλα απαραίτητα για την επιτυχημένη υποβολή της αίτησης. Με την υποβολή της αίτησης, το ταξίδι αποθηκεύεται στη βάση.
- ***Απαραίτητες πληροφορίες***: Κόστος εισιτηρίων, κόστος διαμονής, κόστος εγγραφής, ημερήσιο κόστος και λοιπά έξοδα.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε Researchers, Managers και Director.

Ενημέρωση κατάστασης αιτήσεων

- ***Επιθυμητό αποτέλεσμα***: Εμφάνιση στον πίνακα ταξιδιών των αιτήσεων όπως αυτές έχουν τη συγκεκριμένη χρονική στιγμή.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να ενημερωθεί για τις καταστάσεις στις οποίες βρίσκονται οι αιτήσεις.
- ***Κριτήριο τερματισμού***: Ενημέρωση του πίνακα ταξιδιών του χρήστη.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την ενημέρωση του πίνακα ταξιδιών. Η ενημέρωση πραγματοποιείται μέσω σύνδεσης με τη βάση και διαβάζοντας από αυτήν τις αιτήσεις ως έχουν εκείνη τη χρονική στιγμή.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη και στα έξι είδη χρηστών.

Έγκριση αίτησης ταξιδιού

- ***Επιθυμητό αποτέλεσμα***: Έγκριση της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να εγκρίνει την αίτηση.
- ***Κριτήριο τερματισμού***: Αποθήκευση της εγκεκριμένης αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την έγκριση μιας από τις αιτήσεις που υπάρχουν στον πίνακα ταξιδιών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή έγκρισής της ("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε όλους εκτός από τους accountant και researchers.

Απόρριψη αίτησης ταξιδιού

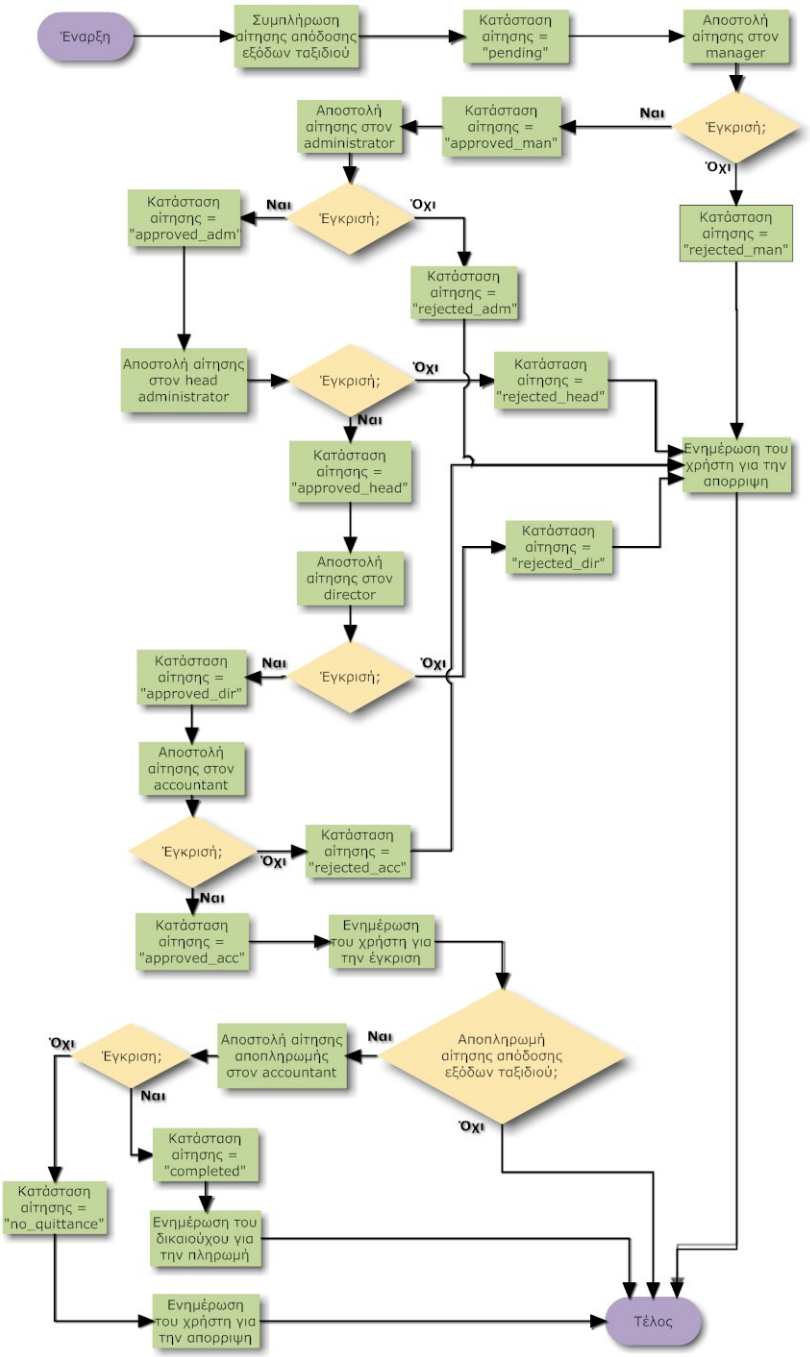
- ***Επιθυμητό αποτέλεσμα***: Απόρριψη της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να απορρίψει την αίτηση.
- ***Κριτήριο τερματισμού***: Αποθήκευση της απορριφθείσας αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την απόρριψη μιας από τις αιτήσεις που υπάρχουν στον πίνακα δαπανών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή απόρριψή της ("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη σε όλους εκτός από τους accountant και researchers.

Έγκριση αποπληρωμής εξόδων ταξιδιού

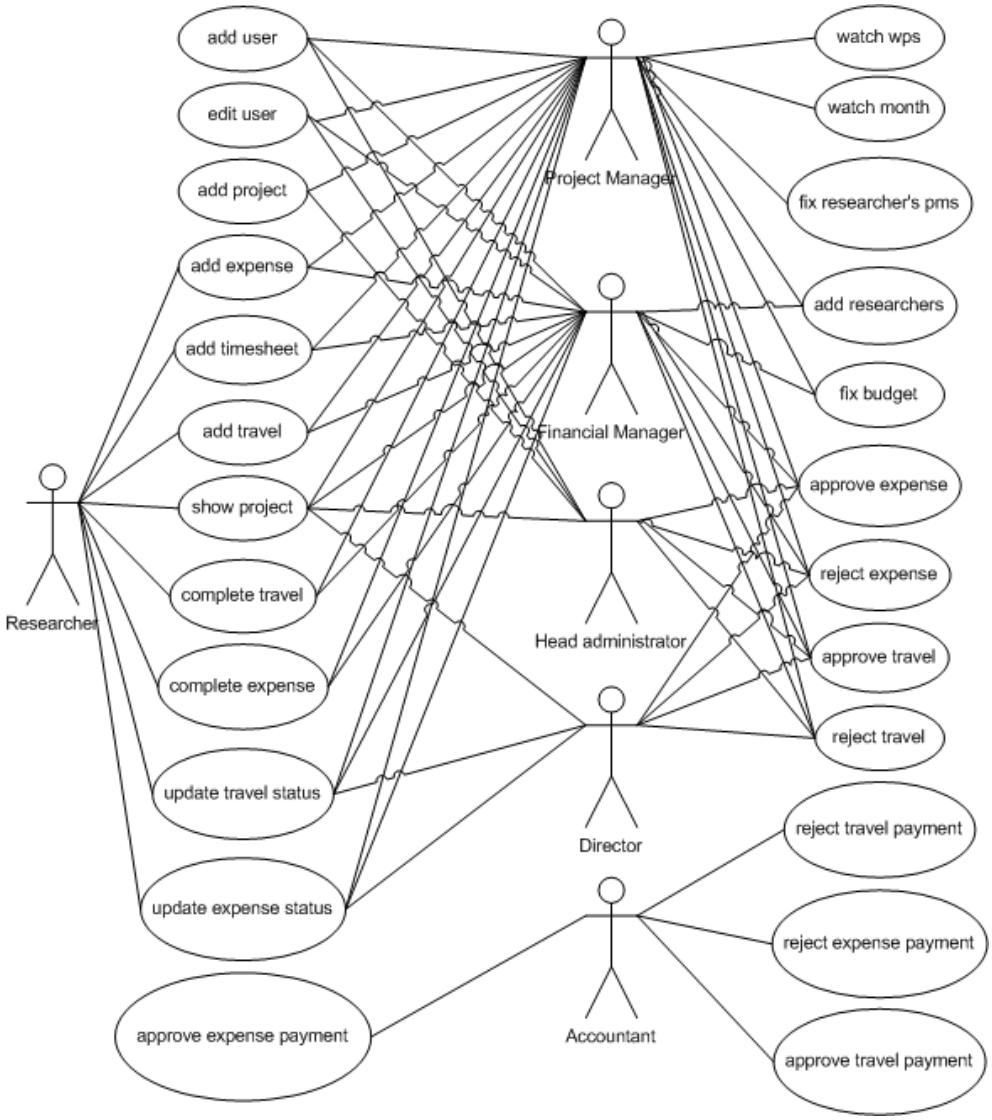
- ***Επιθυμητό αποτέλεσμα***: Έγκριση αποπληρωμής της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να εγκρίνει την πλήρη αποπληρωμή μιας αίτησης.
- ***Κριτήριο τερματισμού***: Αποθήκευση της εγκεκριμένης για αποπληρωμή αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την έγκριση αποπληρωμής μιας από τις αιτήσεις που υπάρχουν στον πίνακα ταξιδιών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή έγκρισής της("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη μόνο σε accountant. Η κατάστασή της αίτησης αλλάζει σε "completed" που σημαίνει ότι η αποπληρωμή της δαπάνης έχει εγκριθεί.

Απόρριψη αποπληρωμής εξόδων ταξιδιού

- ***Επιθυμητό αποτέλεσμα***: Απόρριψη αποπληρωμής της επιλεγμένης αίτησης.
- ***Σημείο εισαγωγής***: Ο χρήστης θέλει να απορρίψει την αποπληρωμή μιας αίτησης.
- ***Κριτήριο τερματισμού***: Αποθήκευση της απορριφθείσας για αποπληρωμή αίτησης στη βάση.
- ***Περιγραφή λειτουργίας***: Ο χρήστης, όταν αυτός το επιθυμεί, επιλέγει την απόρριψη αποπληρωμής μιας από τις αιτήσεις που υπάρχουν στον πίνακα ταξιδιών. Θα πρέπει η συγκεκριμένη αίτηση να βρίσκεται στην κατάλληλη κατάσταση πριν την επιλογή έγκρισής της("pending" ή "approved" ανάλογα με το επίπεδο ιεραρχίας), σύμφωνα με το workflow που ακολουθεί μια αίτηση.
- ***Απαραίτητες πληροφορίες***: Καμία.
- ***Άλλα σχόλια***: Η λειτουργία αυτή είναι διαθέσιμη μόνο σε accountant. Η κατάσταση της αίτησης αλλάζει σε "no_quittance" που σημαίνει ότι η αποπληρωμή της δαπάνης έχει απορριφθεί.



Σχήμα 8.11: Διάγραμμα ροής αίτησης απόδοσης εξόδων ταξιδιού



Σχήμα 8.12: Use case διάγραμμα

ΚΕΦΑΛΑΙΟ 9

Υλοποίηση του συστήματος

Στο παρόν κεφάλαιο περιγράφεται ο τρόπος υλοποίησης των διάφορων επιπέδων του συστήματος (όπως ορίστηκαν στο σχεδιασμό του), από τη σκοπιά των τεχνολογιών και των εργαλείων που χρησιμοποιήθηκαν. Για λεπτομέρειες της υλοποίησης του συστήματος, υπάρχει επαρκής τεκμηρίωση που έχει δημιουργηθεί με *Javadoc* και περιέχεται στο συνοδευτικό CD της διπλωματικής. Επιπλέον, παρουσιάζονται οι μετρικές κώδικα οι οποίες υπολογίστηκαν με τη βοήθεια κάποιων εργαλείων που χρησιμοποιήθηκαν σε όλη τη διαδικασία ανάπτυξης του συστήματος και θα αναφερθούν παρακάτω.

9.1 Τρόπος υλοποίησης λογικών επιπέδων

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την υλοποίηση του συστήματος είναι η αντικειμενοστραφής γλώσσα ***Java***. Επιπλέον, το σύστημα βασίζεται στο περιβάλλον εργασίας ***Spring***. Η έκδοση που χρησιμοποιήθηκε είναι η *2.5.4*. Με τη χρήση της Spring δημιουργήθηκαν οι συσχετίσεις μεταξύ όλων των κλάσεων της εφαρμογής (όπως για τις κλάσεις αποθήκευσης, τους controller) για τη δημιουργία χαλαρού κώδικα με τη βοήθεια του *dependency injection* που παρέχει η Spring.Για κάθε λογικό επίπεδο του συστήματος, ακολουθεί μια γενική περιγραφή της υλοποίησης και των εργαλείων που χρησιμοποιήθηκαν.

Σύστημα βάσης δεδομένων

Η αποθήκευση δεδομένων της εφαρμογής γίνεται με τη χρήση της ***HSQLDB (Hyper Structured Query Language Database)***. Είναι μία σχεσιακή *SQL* μηχανή σχεσιακής βάσης δεδομένων, η οποία είναι γραμμένη σε *Java*. Περιλαμβάνει έναν *JDBC* οδηγό και υποστηρίζει ένα πλούσιο υποσύνολο του προτύπου *ANSI-92 SQL*, και *SQL 99*, καθώς και προσθήκες που έγιναν το 2003. Προσφέρει μία μικρή (μικρότερη απο 100 KB σε μία έκδοση για applets), γρήγορη μηχανή βάσης δεδομένων, η οποία παρέχει in-memory και disk-based πίνακες, και υποστηρίζει embedded και server μεθόδους. Επιπλέον, περιλαμβάνει εργαλεία όπως, έναν όσο το δυνατό μικρότερο web server, in-memory ερωτήματα και εργαλεία διαχείρισης μαζί με έναν μεγάλο αριθμό παραδειγμάτων. Η έκδοση της HSQLDB που χρησιμοποιήθηκε είναι η *1.8.0.10*. Η διαχείριση της βάσης δεδομένων γίνεται από το λογικό *επίπεδο αποθήκευσης* που περιγράφεται αμέσως μετά.

Επίπεδο αποθήκευσης (persistence layer)

Για την υλοποίηση αυτού του επιπέδου χρησιμοποιήθηκε το περιβάλλον εργασίας ***Hibernate***, στην έκδοση *3.2.0.ga*. Σε αυτό το επίπεδο δημιουργήθηκαν οι κλάσεις αποθήκευσης (persistent classes) όπως καθοδηγεί ο σχεδιασμός. Επιπλέον, δημιουργήθηκαν αρχεία αντιστοίχησης για κάθε κλάση, που ορίζουν τον τρόπο που αποθηκεύεται κάθε κλάση στη σχεσιακή βάση δεδομένων του συστήματος (που είναι η HSQLDB). Η Hibernate ρυθμίστηκε έτσι ώστε να χρησιμοποιεί τον οδηγό της HSQLDB και να συνδέεται σωστά με τη βάση. Η ρύθμιση των *sessions* και *transactions* της Hibernate γίνεται από το περιβάλλον εργασίας Spring. Όπως περιγράφηκε και στο αντίστοιχο κεφάλαιο, με αυτόν τον τρόπο απλοποιείται πολύ ο κώδικας και διευκολύνει τον προγραμματιστή με τις λειτουργίες που αφορούν την πρόσβαση στη βάση δεδομένων.

Επίπεδο λειτουργιών

Στο επίπεδο αυτό υλοποιείται το κύριο μέρος των λειτουργιών που παρέχεται από την εφαρμογή (όπως απαιτείται από το σχεδιασμό) και γίνεται ευρεία χρήση της Spring. Επιπλέον, το επίπεδο αυτό παρέχει δεδομένα στο επίπεδο παρουσίασης από το επίπεδο αποθήκευσης και το αντίστροφο. Περιλαμβάνει τις κλάσεις "διαχειριστές" των δεδομένων, οι οποίες περιλαμβάνουν ερωτήματα προς τη βάση με τη βοήθεια του επιπέδου αποθήκευσης και της Spring.

Επίπεδο παρουσίασης

Για την υλοποίηση αυτού του επιπέδου χρησιμοποιείται ένα module της Spring, το ***Spring MVC*** σε συνεργασία με την τεχνολογία ***JavaServer Pages (JSP)***. Η λειτουργία αυτού του module βασίζεται στο πρότυπο Model/View/Controller. Αυτό που κάνει το Spring MVC είναι να δέχεται αιτήσεις σε ένα *dispatcher servlet*, το οποίο αφού συμβουλευτεί ένα ή περισσότερα *handler mappings*, στέλνει την αίτηση στον κατάλληλο *controller*. Ο controller, αφού επεξεργαστεί την αίτηση, στέλνει τα αποτελέσματα της επεξεργασίας (model data) μαζί με το όνομα της οθόνης (*view*, ένα JSP αρχείο) για την οποία προορίζονται, μέσω ενός *ModelAndView* αντικειμένου, στο dispatcher servlet. Αυτό με τη σειρά του ψάχνει να βρεί το view με το όνομα που περιέχει το ModelAndView αντικείμενο, και αφού βρεθεί, χρησιμοποιείται για την εμφάνιση των αποτελεσμάτων.

Επιπλέον, για την καλύτερη διάταξη των JSPs, χρησιμοποιείται το περιβάλλον εργασίας ***Tiles 2*** της *Apache*, στην έκδοση *2.0.5*.

Επίπεδο ασφάλειας

Η ασφάλεια της εφαρμογής υλοποιείται μέσω του module της Spring, ***Spring Security*** (γνωστό μέχρι πρότινος ως Acegi Security). Το module εκτελεί τις ακόλουθες λειτουργίες. Κατά την είσοδο ενός χρήστη στο σύστημα, αναζητά στη βάση αν υπάρχει αντιστοιχία των στοιχείων που εισάγει, και αν υπάρχει, ανασύρει τον/τους ρόλο/ους που έχει ο χρήστης στο σύστημα. Κάθε φορά που ο χρήστης προσπαθεί να προσπελάσει μια σελίδα, ελέγχεται, βάσει του ρόλου του, αν έχει τα απαιτούμενα δικαιώματα για αν δει αυτή τη σελίδα.

9.2 Έλεγχος του συστήματος

Όλες οι κλάσεις της εφαρμογής, εκτός από τους controllers του module Spring MVC, δοκιμάζονται χρησιμοποιώντας έλεγχο μονάδας. Για αυτό το λόγο χρησιμοποιείται το περιβάλλον εργασίας ***Junit***, στην έκδοση *4.4*. Με τη χρήση του Junit έγιναν ***139 επιτυχημένες δοκιμές*** στις κλάσεις της εφαρμογής.

Unit Test Results.

Designed for use with JUnit and Ant.

Summary

Tests	Failures	Errors	Success rate	Time
139	0	0	100.00%	65.006

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
prman.test.data	66	0	0	56.022	2009-06-27T18:53:33	tasos-laptop
prman.test.model	73	0	0	8.984	2009-06-27T18:54:39	tasos-laptop

Σχήμα 9.1: Συνολική αναφορά του Junit

Unit Test Results.

Designed for use with JUnit and Ant.

All Tests

Class	Name	Status	Type	Time(s)
ApprovalManagerTest	testAddNewApproval	Success		4.618
ApprovalManagerTest	testListApproval	Success		0.520
ApprovalManagerTest	testDeleteApproval	Success		0.494
ApprovalManagerTest	testGetByIdApproval	Success		0.422
ApprovalManagerTest	testUpdateApproval	Success		0.442
BudgetManagerTest	testAddNewBudget	Success		2.268
BudgetManagerTest	testListBudget	Success		0.476
BudgetManagerTest	testDeleteBudget	Success		0.450
BudgetManagerTest	testGetByIdBudget	Success		0.407
BudgetManagerTest	testUpdateBudget	Success		0.460
DayPmMonthManagerTest	testAddNewDayPmMonth	Success		2.312

Σχήμα 9.2: Μέρος αναλυτικής αναφοράς του Junit

9.3 Περιβάλλον ανάπτυξης συστήματος

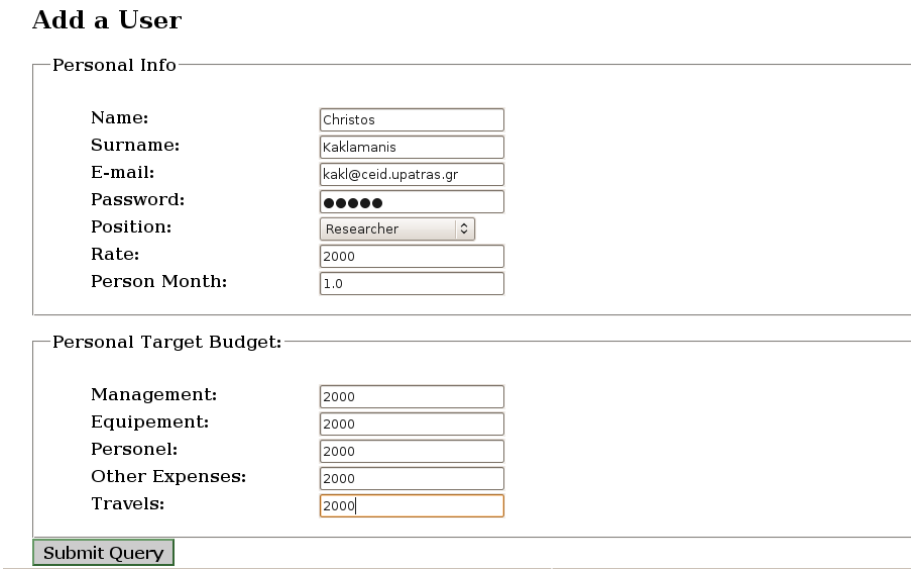
Η ανάπτυξη του συστήματος έγινε στο ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) ***IntelliJ IDEA 8***. Για την μετάφραση και το χτίσιμο της εφαρμογής χρησιμοποιήθηκε το εργαλείο ***Ant***, όπου όλες οι απαιτούμενες διαδικασίες για την ανάπτυξη ορίζονταν ως εργασίες του. Το Ant περιλαμβάνεται ως plugin από το IntelliJ IDEA 8. Ως μεταφραστής χρησιμοποιήθηκε το ***Java 6 software developement kit***. Επιπλέον, η web εφαρμογή γινόταν deploy στον web server ***Resin 3.1.5***

Για τη διαχείριση του έργου χρησιμοποιήθηκε το ολοκληρωμένο περιβάλλον διαχείρισης έργων ***Trac*** (το οποίο έχει περιγραφεί σε προηγούμενο κεφάλαιο). Η αποστολή κώδικα στο Trac γινόταν με τη χρήση του ***Subversion***, το οποίο περιλαμβάνεται στα διαθέσιμα plugins του IntelliJ IDEA 8.

9.4 Στιγμιότυπα (screenshots) του συστήματος

Σε αυτή την ενότητα ακολουθούν κάποιες επιλεγμένες οθόνες του συστήματος οι οποίες παρουσιάζουν κάποιες λειτουργίες του.

Εισαγωγή χρήστη



Σχήμα 9.3: Οθόνη εισαγωγής χρήστη

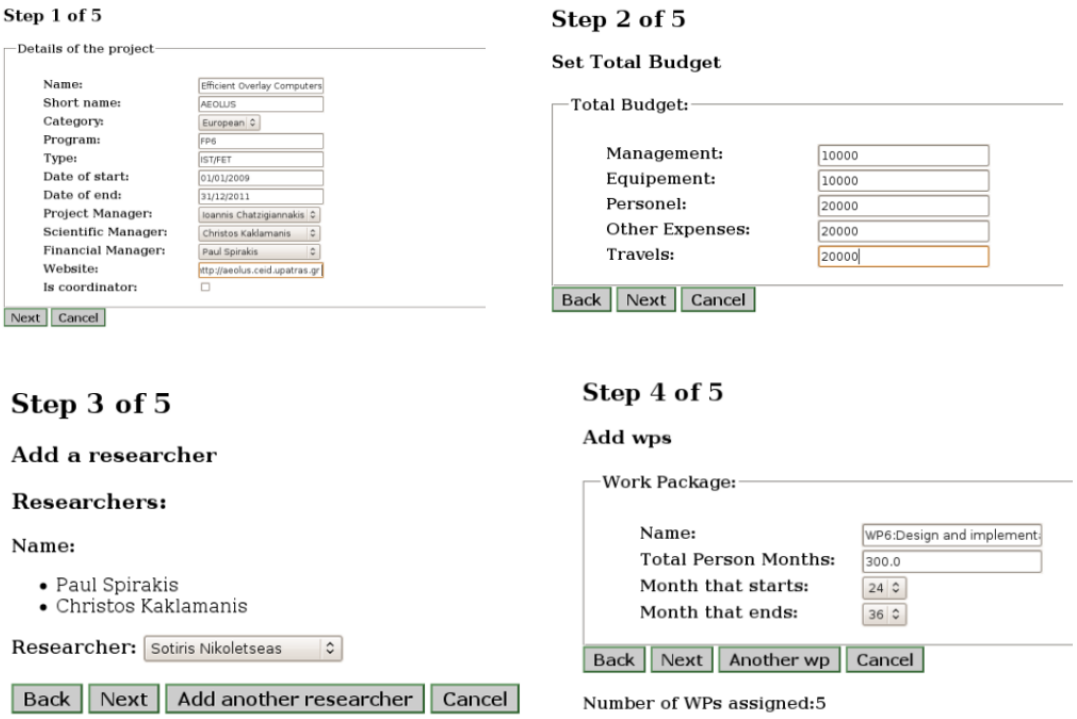
Η πρώτη οθόνη που παρουσιάζεται είναι η οθόνη *εισαγωγής χρήστη*, η οποία είναι διαθέσιμη σε όλους τους managers, στο head administrator και στο director. Σε αυτή την οθόνη οι χρήστες καλούνται να εισάγουν τα στοιχεία του χρήστη που είναι να εγγραφεί στο σύστημα και είναι τα εξής:

- *Όνομα*
- *Επώνυμο*
- *email*

- *Κωδικός πρόσβασης*
- *rate*, και
- *τα στοιχεία του προσωπικού του προϋπολογισμού*

Εισαγωγή ερευνητικού έργου

Η λειτουργία εισαγωγής ερευνητικού έργου στο σύστημα γίνεται σε πέντε βήματα και πραγματοποιείται από πέντε διαφορετικές οθόνες. Στο πρώτο βήμα εισάγονται τα στοιχεία του έργου. Στο δεύτερο βήμα εισάγεται ο συνολικός προϋπολογισμός του έργου, ακολουθεί η εισαγωγή της ερευνητικής ομάδας. Στο τέταρτο βήμα, εισάγονται ένα προς ένα τα πακέτα εργασίας, όπου για κάθε πακέτο εργασίας αποθηκεύονται το όνομά του, οι συνολικοί ανθρωπομήνες που θα σπαταληθούν για αυτό το πακέτο εργασίας, ο μήνας του έργου που ξεκινάει καθώς και ο μήνας του έργου που θα ολοκληρωθεί. Στο τελευταίο βήμα παρουσιάζεται μία σύνοψη όλων των στοιχείων που έχει εισάγει ο χρήστης για επιβεβαίωση της ορθότητάς τους. Αυτή η λειτουργία είναι διαθέσιμη σε όλους τους managers, στο head administrator και στο director.



Σχήμα 9.4: Οθόνη εισαγωγής έργου

Παρακολούθηση πακέτων εργασίας ενός έργου

Η λειτουργία παρακολούθησης πακέτων εργασίας ενός έργου παρουσιάζεται στην παρακάτω οθόνη και είναι διαθέσιμη μόνο στον project manager.

Research Academic Computer Technology Institute

Work Packages overview

Algorithmic Principles for Building Efficient Overlay Computers

Work Package: WP1:Paradigms and principles

Months	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Target Person Month	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33
Planned Person Month	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Difference PersonMonth	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.13	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33	83.33

Work Package: WP2:Resource management

Months	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Target Person Month	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33
Planned Person Month	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Difference PersonMonth	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	7.93	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33	8.33

Work Package: WP3:Sharing information and computation

Months	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Target Person Month	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
Planned Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Difference PersonMonth	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80

Work Package: WP4:Security and trust management

Months	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Target Person Month	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
Planned Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Difference PersonMonth	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12

Work Package: WP5:Extending global computing to wireless users

Months	24	25	26	27	28	29	30	31	32	33	34	35	36
Target Person Month	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69
Planned Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0
Difference PersonMonth	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69	7.69

Work Package: WP6:Design and implementation of components and applications for programmable overlay computers

Months	24	25	26	27	28	29	30	31	32	33	34	35	36
Target Person Month	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08
Planned Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual Person Month	0	0	0	0	0	0	0	0	0	0	0	0	0
Difference PersonMonth	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08

Summary

Work Packge	WP1:Paradigms and principles	WP2:Resource management	WP3:Sharing information and computation	WP4:Security and trust management	WP5:Extending global computing to wireless users	WP6:Design and implementation of components and applications for programmable overlay computers
Target Person Month	2,000	200	2,000	300	100	300
Actual Person Month	0.2	0.4	0	0	0	0
Users Person Month	0	0	0	0	0	0
Difference	1,999.8	199.6	2,000	300	100	300

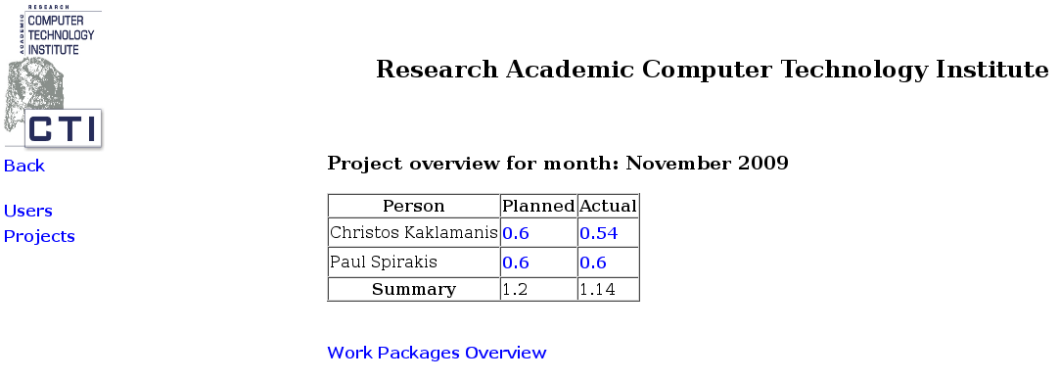
Print me!

Σχήμα 9.5: Οθόνη παρακολούθησης πακέτων εργασίας ενός έργου

Στην οθόνη αυτή εμφανίζονται πίνακες για τα πακέτα εργασίας ενός έργου που περιέχουν, για κάθε μήνα τους, τους προγραμματισμένους ανθρωπομήνες, τους ανθρωπομήνες που έχει ορίσει ο υπεύθυνος έργου για κάθε μήνα, και τους πραγματικούς ανθρωπομήνες που δήλωσαν οι χρήστες ότι εργάστηκαν. Επιπλέον, υπάρχει ένα συγκεντρωτικός πίνακας που έχει τους συνολικούς ανθρωπομήνες για κάθε πακέτο εργασίας του έργου.

Παρακολούθηση του έργου κάποιο μήνα

Η λειτουργία παρακολούθησης του έργου για κάποιο μήνα παρουσιάζεται στην παρακάτω οθόνη και είναι διαθέσιμη μόνο στον project manager.



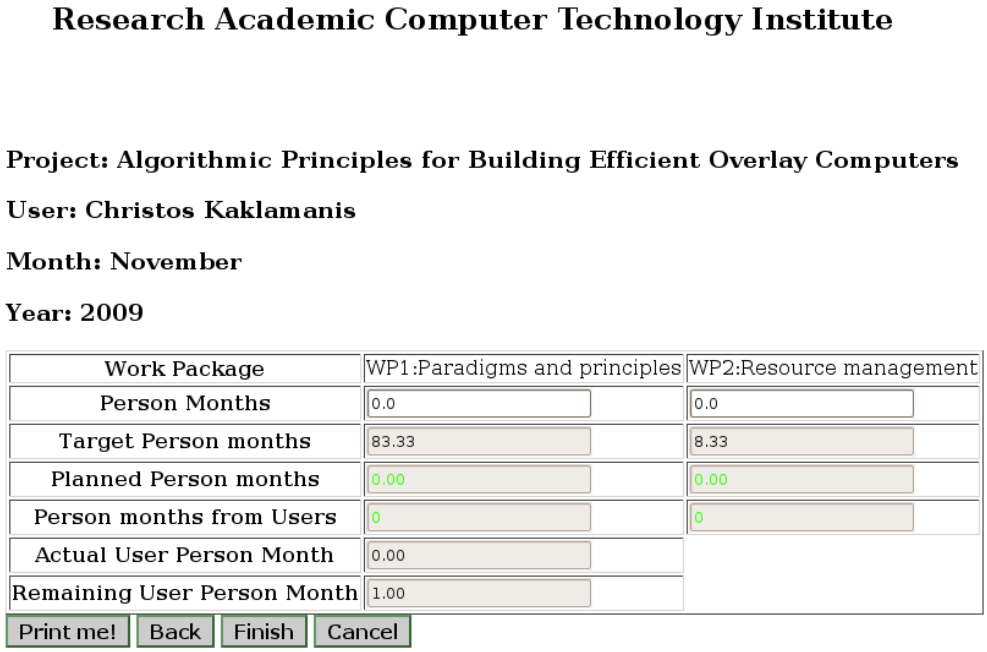
Σχήμα 9.6: Οθόνη παρακολούθησης του έργου κάποιο μήνα

Από την οθόνη συνολικής παρακολούθησης ενός συγκεκριμένου έργου επιλέγεται ένας συγκεκριμένος μήνας για τον οποίο ο project manager θέλει να δει τους συνολικούς ανθρωπομήνες όλων των ερευνητών του έργου για το συγκεκριμένο μήνα. Επιλέγοντας τον μήνα, εμφανίζεται στην οθόνη ένας πίνακας που περιέχει όλους τους ερευνητές που έχουν εργαστεί το συγκεκριμένο μήνα στο έργο καθώς και τους ανθρωπομήνες τους οποίους έχουν εργαστεί.

Καθορισμός ανθρωπομήνα ερευνητή

Η λειτουργία καθορισμού ανθρωπομήνα ενός ερευνητή παρουσιάζεται στην παρακάτω οθόνη και είναι διαθέσιμη μόνο στον project manager. Αυτή η οθόνη μαζί με τις δύο προηγούμενες που περιγράφηκαν παραπάνω, αποτελούν τις κύριες οθόνες που έχει στη διάθεση του ο project manager για τη διαχείριση και την επίβλεψη των ανθρωπομηνών του έργου.

Ο project manager αφού διαλέξει ερευνητή και το μήνα για τον οποίο θα καθορίσει τους ανθρωπομήνες του ερευνητής, εισάγει για κάθε πακέτο εργασίας που είναι διαθέσιμο τον συγκεκριμένο μήνα τους ανθρωπομήνες που θα εργαστεί ο ερευνητής για αυτά τα πακέτα εργασίας. Σε αυτή την οθόνη είναι διαθέσιμο το ποσό του ανθρωπομήνα που είναι διαθέσιμο για τον ερευνητή, οι συνολικοί ανθρωπομήνες που έχει ήδη ορίζει ο project manager, οι ανθρωπομήνες που έχουν ήδη εισάγει άλλοι ερευνητές, καθώς και οι ανθρωπομήνες που έχουν οριστεί στην εισαγωγή του έργου, οι οποίοι είναι διαθέσιμοι για το κάθε πακέτο εργασίας.



Σχήμα 9.7: Οθόνη καθορισμού ανθρωπομήνα ερευνητή

Κατάθεση timesheet

Στην οθόνη αυτή υπάρχει μία φόρμα την οποία πρέπει να συμπληρώσει ο χρήστης που καταθέτει το timesheet. Ο χρήστης συμπληρώνει για κάθε πακέτο εργασίας που είναι διαθέσιμο το συγκεκριμένο μήνα, πόσες ώρες την ημέρα εργάστηκε. Σε αυτή την οθόνη είναι διαθέσιμες οι ώρες που έχει ορίζει ο project manager για το συγκεκριμένο χρήστη ότι θα εργαστεί σε κάθε πακέτο εργασίας. Ο χρήστης δεν μπορεί να ξεπεράσει αυτές τις ώρες. Επιπλέον, ο χρήστης δεν μπορεί να εισάγει ώρες εργασίας τα σαββατοκύριακα. Για ευκολία του χρήστη, υπολογίζονται δυναμικά οι συνολικές ώρες που έχει εισάγει ο χρήστης, τόσο για κάθε πακέτο εργασίας όσο και οι συνολικές ώρες για όλα τα πακέτα εργασίας. Παρακάτω παρουσιάζεται η οθόνη στην οποία ένας χρήστης καταθέτει ένα timesheet. Η οθόνη αυτή είναι διαθέσιμη σε όλους τους χρήστες.

Research Academic Computer Technology Institute

Project: Algorithmic Principles for Building Efficient Overlay Computers

User: Christos Kaklamanis

Month: November

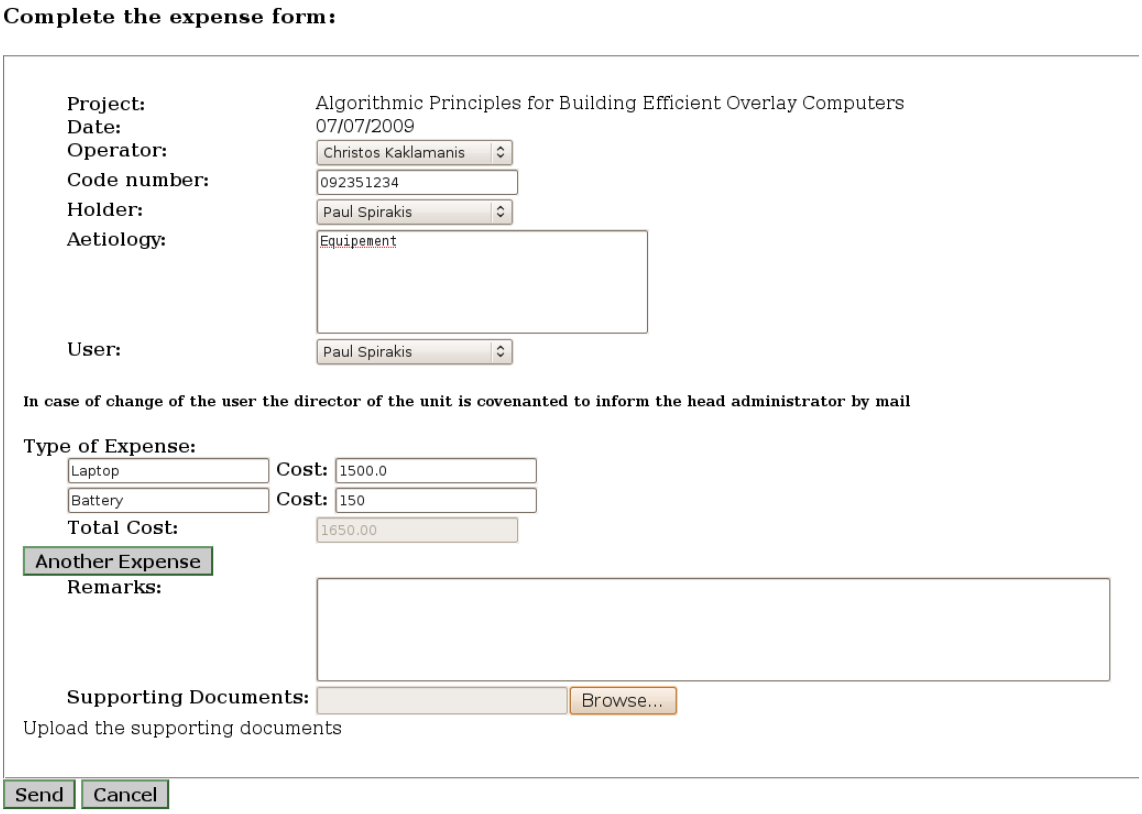
Year: 2009

Day	WP1:Paradigms and principles	WP2:Resource management
Sunday, 1	0.0	0.0
Monday, 2	0.0	0.0
Tuesday, 3	0.0	0.0
Wednesday, 4	0.0	0.0
Thursday, 5	0.0	0.0
Friday, 6	0.0	0.0
Saturday, 7	0.0	0.0
Sunday, 8	0.0	0.0
Monday, 9	0.0	0.0
Tuesday, 10	0.0	0.0
Wednesday, 11	0.0	0.0
Thursday, 12	0.0	0.0
Friday, 13	0.0	0.0
Saturday, 14	0.0	0.0
Sunday, 15	0.0	0.0
Monday, 16	0.0	0.0
Tuesday, 17	0.0	0.0
Wednesday, 18	0.0	0.0
Thursday, 19	0.0	0.0
Friday, 20	0.0	0.0
Saturday, 21	0.0	0.0
Sunday, 22	0.0	0.0
Monday, 23	0.0	0.0
Tuesday, 24	0.0	0.0
Wednesday, 25	0.0	0.0
Thursday, 26	0.0	0.0
Friday, 27	0.0	0.0
Saturday, 28	0.0	0.0
Sunday, 29	0.0	0.0
Monday, 30	0.0	0.0
Planned Per Work Package	26.25	52.5
Actual Per Work Package Total	0.00	0.00
Total Hours	0.00	

Print me! Back Finish Cancel

Σχήμα 9.8: Οθόνη κατάθεσης timesheet

Προσθήκη αίτησης δαπάνης



Σχήμα 9.9: Οθόνη προσθήκης αίτησης δαπάνης

Σε αυτή την οθόνη ο χρήστης καλείται να συμπληρώσει τα παρακάτω στοιχεία:

- *Χειριστής*: Ο χρήστης του συστήματος ο οποίος καταθέτει την αίτηση για τη δαπάνη.
- *Έργο*: Το έργο στο οποίο θα χρεωθεί η δαπάνη που κατατίθεται.
- *Κωδικός δαπάνης*: Ένα μοναδικό αλφαριθμητικό το οποίο χρησιμοποιείται για την αναφορά στη συγκεκριμένη αίτηση δαπάνης.
- *Είδος δαπάνης*: Αναφέρεται ακριβώς το αντικείμενο που αγοράστηκε.
- *Κόστος*: Το κόστος της δαπάνης για το είδος που περιγράφεται.
- *Παρατηρήσεις*: Ο χρήστης που καταθέτει την αίτηση δαπάνης μπορεί να γράψει κάποιες παρατηρήσεις που αφορούν τη δαπάνη.
- *Έγγραφα*: Ο χρήστης που καταθέτει τη δαπάνη πρέπει να συμπεριλάβει και τα συνοδευτικά της έγγραφα, όπως τιμολόγια και αποδείξεις.
- *Αιτιολογία*: Αναφέρεται ο λόγος δημιουργίας αυτής της δαπάνης, όπως εξοπλισμός, φωτοτυπίες βιβλίων.
- *Χρήστης*: Αφορά τον πάγιο εξοπλισμό, και είναι το άτομο που το χρησιμοποιεί.

- *Δικαιούχος*: Ο δικαιούχος είναι ο άνθρωπος ο οποίος θα παραλάβει τα χρήματα που καλύπτουν τα έξοδα της δαπάνης.

Η οθόνη αυτή είναι διαθέσιμη σε Researchers, Managers και Director.

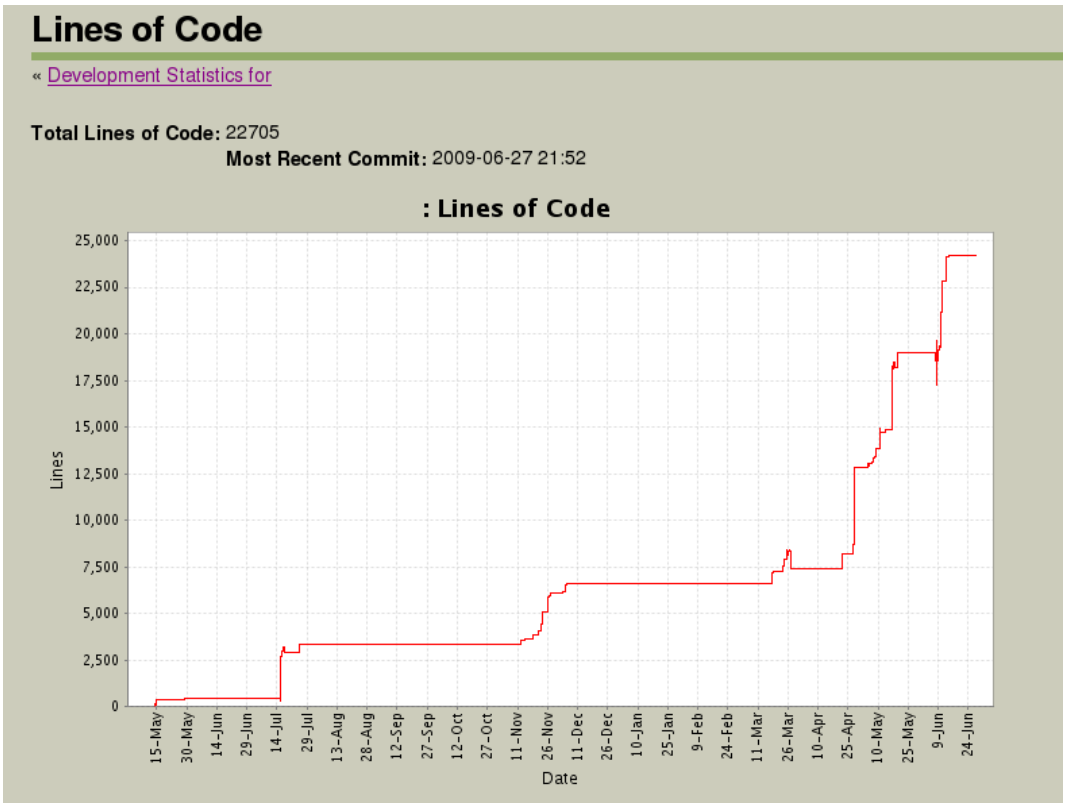
Προβολή όλων των χρηστών του συστήματος

Αυτή η οθόνη είναι διαθέσιμη σε όλους τους χρήστες εκτός από τους ερευνητές. Από τη συγκεκριμένη οθόνη ο χρήστης μπορεί να επιλέξει να δει λεπτομέρειες για κάποιον χρήστη, ώστε να τις επεξεργαστεί. Στην οθόνη φαίνεται το ονοματεπώνυμο κάθε χρήστη, συνοδευόμενο από το email του και από τη θέση του στο σύστημα. Πατώντας στο ονοματεπώνυμο του χρήστη εμφανίζονται λεπτομέρειες για το συγκεκριμένο χρήστη. Πατώντας στο email του χρήστη ανοίγει κάποιος email client για την αποστολή μηνύματος ηλεκτρονικού ταχυδρομείου.

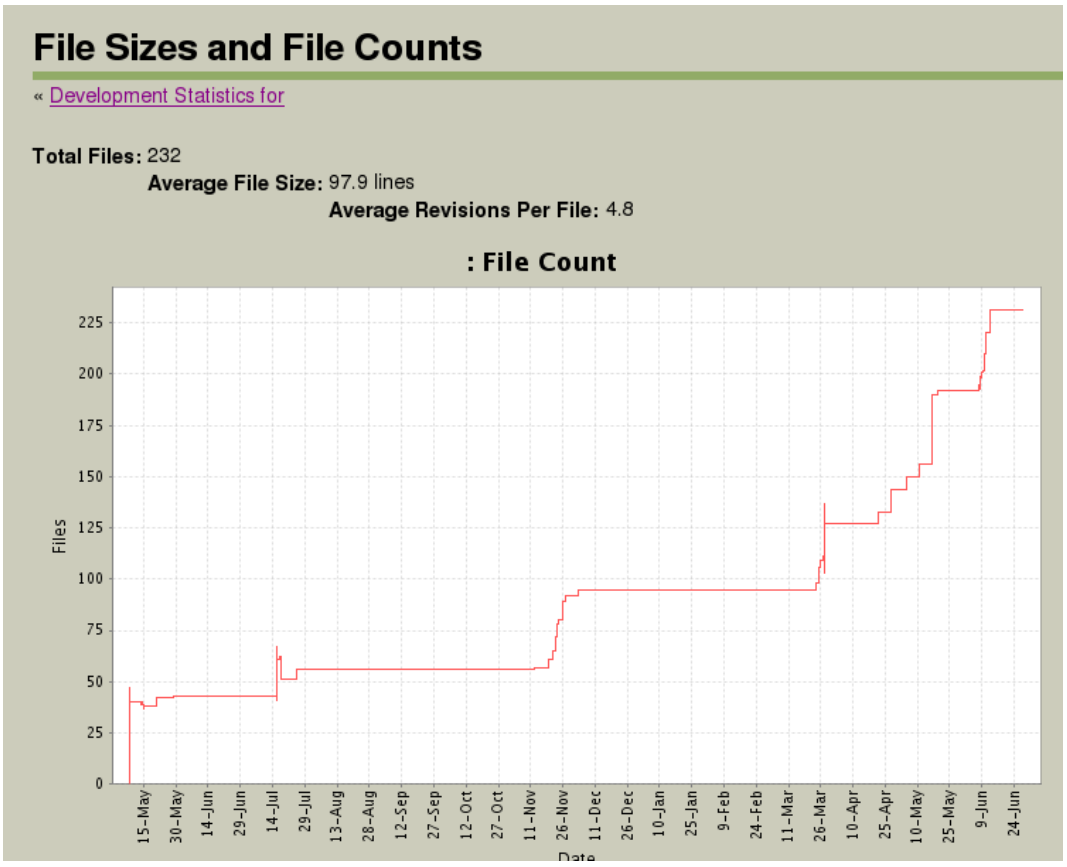
Σχήμα 9.10: Οθόνη προβολής όλων των χρηστών του συστήματος

9.5 Ποιότητα και μετρικές κώδικα

Μέσω των αποστολών με τη χρήση του Subversion, η χρήση του εργαλείου ***StatSVN***, έδινε τη δυνατότητα διατήρησης στατιστικών για την ανάπτυξη του κώδικα, όπως συχνότητα αποστολής κώδικα στο Trac, μέγεθος σε γραμμές κώδικα του συστήματος κ.α. Σύμφωνα με το StatSVN, η εφαρμογή έχει μέγεθος ***22705 γραμμές κώδικα*** σε ***232 αρχεία***. Στις παρακάτω εικόνες φαίνονται δύο από τις αναφορές του StatSVN.



Σχήμα 9.11: Μέγεθος του συστήματος σε γραμμές κώδικα



Σχήμα 9.12: Αριθμός αρχείων του συστήματος

Για την συγγραφή αποδοτικού και κατανοητού κώδικα, ακολουθώντας τα πρότυπα που έχουν οριστεί, χρησιμοποιήθηκαν τα εργαλεία ***PMD 4.2.1*** και ***Checkstyle 5.0***. Και τα δύο εργαλεία, σαρώνοντας τον κώδικα ελέγχουν για λάθη όπως άδειες try/catch/finally/switch δηλώσεις, αχρησιμοποίητες τοπικές μεταβλητές - παραμέτρους - μεθόδους κ.α. Στην εφαρμογή, ο αριθμός των λαθών που ανιχνεύτηκαν από το PMD, έχοντας ενεργοποιημένους όλους τους ελέγχους, είναι ***συνολικά 97***, σε ***15 αρχεία***, εκ των οποίων τα 24 είναι προτεραιότητας επιπέδου 3 και τα υπόλοιπα 73 είναι προτεραιότητας επιπέδου 5. Στην παρακάτω εικόνα φαίνεται η σύνοψη της αναφορά του PMD.

PMD 4.2.1 Report

Summary

Files	Total	Priority 1	Priority 2	Priority 3	Priority 4	Priority 5
15	97	0	0	24	0	73

Prio	File	Line	Description
5	.media.Hephaestus.Source.prman.trunk.prman.webapp.controllers.AddProjectFormController	116	Found 'DU'-anomaly for variable 'project' (lines '116'-'171').
5	.media.Hephaestus.Source.prman.trunk.prman.webapp.controllers.AddProjectFormController	139	Found 'DD'-anomaly for variable 'inUser' (lines '139'-'139').
5	.media.Hephaestus.Source.prman.trunk.prman.webapp.controllers.AddProjectFormController	139	Found 'DU'-anomaly for variable 'inUser' (lines '139'-'171').
5	.media.Hephaestus.Source.prman.trunk.prman.webapp.controllers.AddProjectFormController	145	Found 'DD'-anomaly for variable 'counter' (lines '145'-'147').
5	.media.Hephaestus.Source.prman.trunk.prman.webapp.controllers.AddProjectFormController	187	Found 'DU'-anomaly for variable 'project' (lines '187'-'207').

Σχήμα 9.13: Σύνοψη του PMD

Μέρος V

Συμπεράσματα – Βιβλιογραφία

ΚΕΦΑΛΑΙΟ **10**

Συμπεράσματα και προοπτικές

Όταν η όλη προσπάθεια συγγραφής της διπλωματικής εργασίας και της ανάπτυξης του συστήματος διαχείρισης ερευνητικών έργων έχει ολοκληρωθεί, είναι πλέον ασφαλές να εξαχθούν κάποια συμπεράσματα σχετικά με το αντικείμενο με το οποίο πραγματεύεται η διπλωματική. Επιπλέον, μπορεί να γίνει απολογισμός της όλης προσπάθειας ώστε να εξακριβωθεί ποιοι στόχοι εκπληρώθηκαν και σε τι ποσοστό.

Η ανάπτυξη μίας εφαρμογής είναι από τη φύση της μια πολύπλοκη και πολυδιάστατη διαδικασία. Όμως, πλέον υπάρχουν πολλές σύγχρονες μέθοδοι, οι οποίες έχουν ως σκοπό την απλοποίηση αυτής της διαδικασίας. Κάθε εφαρμογή προστάζει και διαφορετική προσέγγιση για την ανάπτυξή της, κάτι το οποίο εξαρτάται από το είδος της ίδιας της εφαρμογής. Δεν υπάρχει κάποια ιδανική διαδικασία ανάπτυξης λογισμικού, όμως πάντα υπάρχει τρόπος να βελτιωθούν οι διαδικασίες που ακολουθούνται κατά την ανάπτυξη. Υπάρχουν πλέον πολλές επιλογές σε σχέση με το πως θα μοντελοποιηθεί η διαδικασία της ανάπτυξης, με σκοπό την παραγωγή λογισμικού, οικονομικά και αποδοτικά. Αυτό που πρέπει να κάνει ο μηχανικός λογισμικού είναι να επιλέξει το κατάλληλο μοντέλο που θέλει να ακολουθήσει, το οποίο θα βελτιώνει όσο το δυνατόν περισσότερο τη διεργασία λογισμικού.

Πέρα απο τις επιλογές που υπάρχουν για την μοντελοποίηση της διεργασίας λογισμικού, σήμερα υπάρχουν και αρκετές τεχνολογίες που μπορούν να χρησιμοποιηθούν. Στην παρούσα διπλωματική περιγράφηκαν δύο τέτοιες τεχνολογίες (Spring και Hibernate), οι οποίες προσφέρουν ευκολία στην ανάπτυξη της εφαρμογής και βελτίωση της, κάθε μία σε διαφορετικά μέρη της εφαρμογής. Μπορούμε, με σιγουριά πλέον, να πούμε ότι το περιβάλλον εργασίας Hibernate σε συνδυασμό με το περιβάλλον εργασίας Spring αποτελεί μία πολύ καλή λύση για την αποθήκευση δεδομένων και διασύνδεση κλάσεων σε εφαρμογές που στηρίζονται σε Java. Ο συνδυασμός των δυνατοτήτων και η δυνατότητα διασύνδεσής τους δίνει τη δυνατότητα στον προγραμματιστή να ασχοληθεί μόνο με το λειτουργικό κομμάτι της εφαρμογής.

Επιπλέον, η πληθώρα εργαλείων που υπάρχουν για τα διάφορα μέρη της ανάπτυξης λογισμικού έχουν κάνει τη ζωή των προγραμματιστών πιο εύκολη. Όμως, η επιλογή των κατάλληλων εργαλείων αποτελεί κομβικό σημείο για την ανάπτυξη μίας εφαρμογής. Θα πρέπει να επιλέγονται εκείνα τα εργαλεία που θα βελτιώνουν την αποδοτικότητα ενός προγραμματιστή και όχι να αναλώνουν το χρόνο άσκοπα με τη χρήση τους. Πολλές φορές η επιλογή τους θα πρέπει να εξαρτάται από την εξοικείωση του προγραμματιστή με τα συγκεκριμένα εργαλεία.

Συμπερασματικά, το αποτέλεσμα της ανάπτυξης ενός λογισμικού εξαρτάται από

πολλές συνιστώσες. Για την επίτευξη ενός αποδοτικού λογισμικού γρήγορα και οικονομικά, πρέπει να γίνει μια σειρά από σωστές επιλογές οι οποίες θα διασφαλίζουν ότι το αποτέλεσμα μίας διεργασίας λογισμικού είναι το επιθυμητό, κάτι το οποίο ακολουθήθηκε στην περίπτωση της εφαρμογής που αναπτύχθηκε στα πλαίσια της διπλωματικής.

Για τις ανάγκες της ανάπτυξης του συστήματος, μελετήθηκαν πολλές από τις διαθέσιμες επιλογές που υπάρχουν, τόσο σε τεχνολογίες όσο και εργαλεία, αξιολογήθηκαν και η επιλογή τους έγινε με βάση τις προοπτικές που παρέχουν για όσο το δυνατόν καλύτερο αποτέλεσμα, για τη φύση του συγκεκριμένου συστήματος.

Όμως, οι ανάγκες του *Ερευνητικού Ακαδημαϊκού Ινστιτούτου Τεχνολογίας Υπολογιστών* μπορεί να αλλάξουν με τον καιρό. Ο τρόπος ανάπτυξης του συστήματος, επιτρέπει την εύκολη προσθήκη επιπλέον λειτουργιών χωρίς να επηρεάζει σε μεγάλο βαθμό τις ήδη υπάρχουσες καθώς επίσης υποστηρίζει και την τροποποίηση τους.

Επιπλέον, υπάρχει η προοπτική χρήσης του συστήματος και από άλλους ερευνητικούς οργανισμούς γιατί καλύπτει τις βασικές λειτουργίες που απαιτούνται, και με μικρές τροποποιήσεις μπορεί να προσαρμοστεί στις ανάγκες τους. Μπορεί κάποιος να χρησιμοποιήσει το σύστημα ως τη βάση στην οποία θα υλοποιηθούν επιπλέον λειτουργίες και διαφορετικές οντότητες ανάλογα με τη δομή του εκάστοτε οργανισμού. Επίσης, αν κάποιος επιθυμεί την επέκταση του, μπορεί να υλοποιήσει τη σύνδεση του με εργαλεία διαχείρισης έργων, όπως το TRAC, ή και με εργαλεία παρακολούθησης προβλημάτων, όπως το Bugzilla. Πρόκειται για ένα σύστημα, στο οποίο μπορούν να ενσωματωθούν πολλά εργαλεία ή να ενσωματωθεί εκείνο σε αυτά.

Κλείνοντας, θα ήθελα να αναφέρω πως ο τομέας της ανάπτυξης εφαρμογών, και γενικότερα της τεχνολογίας λογισμικού, είναι ένα τομέας με συνεχείς προκλήσεις οι οποίες αναδύονται από την ανάγκη για καλύτερο, πιο αποδοτικό αλλά και ταυτόχρονα προσιτό λογισμικό, το οποίο μπορεί να επιτευχθεί με την βελτίωση της διαδικασίας ανάπτυξης αλλά και των εργαλείων που μπορούν να χρησιμοποιηθούν.

Βιβλιογραφία

[1] Checkstyle 5.0, http://checkstyle.sourceforge.net/.

[2] Apache, "Apache ant", http://ant.apache.org/.

[3] Apache, "Tiles 2", http://tiles.apache.org/.

[4] APPENDIUM, "Statsvn", http://www.statsvn.org/.

[5] ATLASSIAN, "Jira", http://www.atlassian.com/software/jira/.

[6] Christian Bauer and Gavin King, *Java Persistence with Hibernate*, Manning Publications Co, 2007.

[7] Kent Beck, "Ieee computer", In *Embracing change with extreme programming*, 70–78, 1999.

[8] Kent Beck, "Extreme programming explained", Boston: Addison – Wesley, 2000.

[9] Caucho, "Resin", http://www.caucho.com/resin/.

[10] Jet Brains Co, "Intellij idea", http://www.jetbrains.com/idea/features/index.html, 2009.

[11] Alistair Cockburn, "Agile software development", Reading, MA: Addison – WesleyReading, 2001.

[12] CruiseControl.NET, "What is continuous integration", http://www.fermentas.com/techinfo/nucleicacids/maplambda.htm.

[13] Hibernate Reference Documentation, "Hibernate tutorial", http://docs.jboss.org/hibernate/stable/core/reference/en/html/tutorial-firstapp.html, 2004.

[14] Edgewall, "Trac", http://trac.edgewall.org/.

[15] GForge, http://gforge.org/gf/.

[16] Jim Highsmith, "Adaptive software developement: A collaborative approach to managing complex systems", New York: Dorset House, 2000.

[17] HSQLDB, http://hsqldb.org/.

[18] Rob Johnson, *Expert One-on-One: J2EE Design and Development*, Wiley, John & Sons, Incorporated, 2004.

[19] JUnit.org, "Junit", http://www.junit.org/.

[20] Jan Machacek, Aleksa Vukotic, Anirvan Chakraborty, and Jessica Ditt, *Pro Spring 2.5*, APRESS, 2008.

[21] Visual Studio Magazine, "Code metrics", http://visualstudiomagazine.com/articles/2008/10/21/code-metrics.aspx.

[22] Sun Microsystems, "Netbeans ide", http://www.netbeans.org/, 2009.

[23] Agile Modeling, "Introduction to uml 2 use case diagrams", http://www.agilemodeling.com/artifacts/useCaseDiagram.htm.

[24] S.R. Palmer and J.M. Felsing, "A practical guide to feature-driven development", Englewood Cliffs, NJ: Prentice Hall, 2002.

[25] Shari Lawrence Pfleeger, *Software Engineering: Theory and Practice*, Prentice Hall, 2nd ed., 2001.

[26] Ken Schwaber and Mike Beedle, "Agile software development with scrum", Englewood Cliffs, NJ: Prentice Hall, 2001.

[27] Khushboo Sharan, "Continuous integration using team foundation build", January 2006.

[28] Ian Sommerville, "Rapid software developement", In *Software Engineering*, Person Education Limited, 8th ed., 391–405, 2007.

[29] Ian Sommerville, "Software processes", In *Software Engineering*, Person Education Limited, 8th ed., 64–71, 2007.

[30] SourceForge.net, "Pmd", http://pmd.sourceforge.net/.

[31] J. Stapleton, "Dsdm dynamic systems development method", Harlow: Addison – Wesley, 1997.

[32] Βικιπαίδεια, "Hibernate framework", http://el.wikipedia.org/wiki/Hibernate_Framework.

[33] Java Touch, "Introduction to spring framework", http://javatouch.googlepages.com/javatouchguidetospringframework.

[34] Craig Walls, *Spring in action*, Manning Publications Co, 2nd ed., 2008.

[35] Wikipedia, "Apache ant", http://en.wikipedia.org/wiki/Apache_Ant.

[36] Wikipedia, "Bugzilla", http://en.wikipedia.org/wiki/Bugzilla.

[37] Wikipedia, "Checkstyle", http://en.wikipedia.org/wiki/Checkstyle.

[38] Wikipedia, "Concurrent versions system", http://en.wikipedia.org/wiki/Concurrent_Versions_System.

[39] Wikipedia, "Continuous integration", http://en.wikipedia.org/wiki/Continuous_Integration.

[40] Wikipedia, "Eclipse (software)", http://en.wikipedia.org/wiki/Eclipse_(software).

[41] Wikipedia, "Gforge", http://en.wikipedia.org/wiki/GForge.

[42] Wikipedia, "Hsqldb", http://en.wikipedia.org/wiki/HSQLDB.

[43] Wikipedia, "Integrated development environment", http://en.wikipedia.org/wiki/Integrated_development_environment.

[44] Wikipedia, "Issue tracking system", http://en.wikipedia.org/wiki/Issue_tracking_system.

[45] Wikipedia, "Java persistence api", http://en.wikipedia.org/wiki/Java_Persistence_API.

[46] Wikipedia, "Junit", http://en.wikipedia.org/wiki/JUnit.

[47] Wikipedia, "Netbeans", http://en.wikipedia.org/wiki/NetBeans.

[48] Wikipedia, "Object-relational mapping", http://en.wikipedia.org/wiki/Object-relational_mapping.

[49] Wikipedia, "Pmd (software)", http://en.wikipedia.org/wiki/PMD_(software).

[50] Wikipedia, "Software design", http://en.wikipedia.org/wiki/Software_design.

[51] Wikipedia, "Sourceforge", http://en.wikipedia.org/wiki/SourceForge.

[52] Wikipedia, "Subversion (software)", http://en.wikipedia.org/wiki/Subversion_(software).