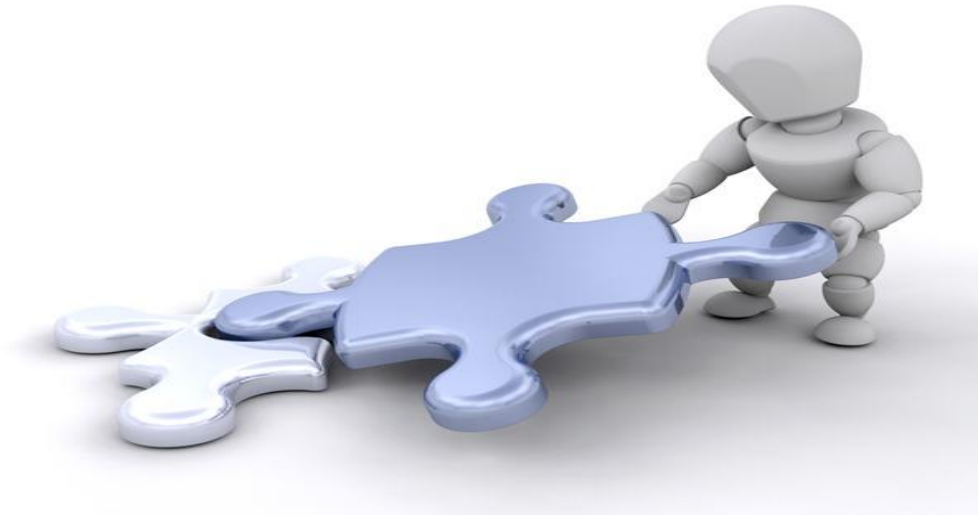
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη εργαλείου διαχείρισης έργων λογισμικού»



Του φοιτητή

Παρασκευά Κουκάρα

Αρ. Μητρώου: 06/3059

Επιβλέπων καθηγητής

Δεληγιάννης Ιγνάτιος

Θεσσαλονίκη 2013

ΠΡΟΛΟΓΟΣ

Η παρακάτω εργασία αποτελεί την ολοκλήρωση μιας προσπάθειας, που έγινε στα πλαίσια της ολοκλήρωσης των σπουδών μου, στο Τμήμα Πληροφορικής, του Αλεξάνδρειου Τεχνολογικού Εκπαιδευτικού Ιδρύματος.

Με την εργασία αυτή σε πρώτο στάδιο επιδιώκεται η κατανόηση και μελέτη εννοιών που αφορούν την ανάπτυξη λογισμικού και στη συνέχεια γίνεται πρακτική εφαρμογή αυτών των εννοιών μέσω της απόπειρας ανάπτυξης ενός εργαλείου διαχείρισης έργων λογισμικού.

Ολοκληρώνοντας τη συγγραφή αυτής της πτυχιακής εργασίας, θα ήθελα να ευχαριστήσω τον εποπτεύοντα διδάσκοντα κ. Δεληγιάννη Ιγνάτιο, για τις συμβουλές, τις παρατηρήσεις και την υποστήριξη του καθ' όλη τη διάρκεια εκπόνησης της εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους καθηγητές του Τμήματος Πληροφορικής , οι οποίοι μέσω των διδαχών τους κατά την παραμονή μου στο Τμήμα Πληροφορικής, υπήρξαν αρωγοί μου.

ΠΕΡΙΛΗΨΗ

Η εργασία αυτή έχει ως στόχο την απόκτηση των απαραίτητων γνώσεων που είναι σημαντικές ώστε να γίνει εφικτή η ανάπτυξη ενός εργαλείου το οποίο θα μπορεί να διαχειρίζεται έργα λογισμικού.

Γίνεται μελέτη και αναφορά σε έννοιες μηχανικής και μεθοδολογιών λογισμικού, όπως η εκτίμηση προσπάθειας, μετρικές κώδικα, περιπτώσεις χρήσης, πριν να γίνει η κωδικοποίησή τους από το εργαλείο που αναπτύχθηκε.

Εν κατακλείδι, το ίδιο το εργαλείο είναι ικανό να πραγματοποιεί την καταγραφή των εκπορευομένων αποτελεσμάτων.

ΛΕΞΕΙΣ-ΚΛΕΙΔΙΑ : εκτίμηση προσπάθειας, σημεία αστάθμητου βάρους στις περιπτώσεις χρήσης, παράγοντας τεχνικής πολυπλοκότητας, παράγοντας εξωγενούς πολυπλοκότητας, παράγοντας παραγωγικότητας, μετρικές, κυκλωματική ποπλυπλοκότητα, σταθμισμένες μέθοδοι ανά κλάση, έλλειψη συνοχής στις μεθόδους, βάθος δένδρου κληρονομικότητας, αριθμός απογόνων, ένωση μεταξύ αντικειμένων, Visual Studio, ckjm, components, περίπτωση χρήσης, εργαλείο σχεδίασης ανάπτυξης λογισμικού.

ABSTRACT

This work aims at the acquisition of the essential knowledge that is important so that it becomes feasible for someone to develop a tool which might be able to manage the development of software projects.

At first, it studies and reports, some of the most significant concepts of software methodologies and mechanics, like the effort estimation, the metrics of code and the use cases. Then, all these concepts that are discussed thoroughly, are encoded as parts of the developed tool.

In conlusion, the completed and finalised tool is capable of recording and saving the results.

KEY-WORDS : effort estmation, use case points, UUCP, TCF, ECF, PF, UCP, metrics, CC, WMC, LCOM, DIT, NOC, RFC, CBO, Visual Studio, ckjm, components, use case, software development tool.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

ΕΙΣΑΓΩΓΗ

Περιλαμβάνει τους στόχους και σκοπούς της πτυχιακής εργασίας καθώς και την περιγραφή των κεφαλαίων που ακολουθούν.

Αντικείμενο της πτυχιακής αυτής εργασίας είναι η δημιουργία ενός εργαλείου για την διαχείριση έργων λογισμικού. Συγκεκριμένα, με βάση τις περιπτώσεις χρήσης, το εργαλείο αυτό θα παρέχει τις ακόλουθες δυνατότητες:

- Θα διαχειρίζεται τον χρόνο ανάπτυξης ενός έργου λογισμικού.
- Θα παρακολουθείται η ανάπτυξη νέων λειτουργιών, αλλαγών, σφαλμάτων, ανά περίπτωση χρήσης, άτομο ή ομάδα.
- Θα υπάρχει η λειτουργικότητα εκτίμησης της προσπάθειας (effort estimation).
- Θα ενσωματώνει ένα διάθεσιμο εργαλείο μετρικών (metrics tool).

Η εργασία χωρίστηκε σε τρία (3) επιμέρους τμήματα κατά την σχεδίαση και συγγραφή της εφαρμογής:

- ✓ Interface της εφαρμογής.
- ✓ Εισαγωγή της εκτίμησης προσπάθειας (effort estimation).
- ✓ Ενσωμάτωση ενός έτοιμου εργαλείου μετρικών (metrics tool).

Για τα τρία (3) αυτά επιμέρους τμήματα, θα γίνει λεπτομερής αναφορά στα κεφάλαια που ακολουθούν.

Κεφάλαιο 1 : Γενικές πληροφορίες

Περιέχει πληροφορίες που αφορούν ζητήματα που μας απασχόλησαν σε πρώιμο στάδιο προσέγγισης του θέματος της εργασίας.

Κεφάλαιο 2: Εκτίμηση προσπάθειας (Effort Estimation)

Το συνολικό θεωρητικό υπόβαθρο που απαιτείται για να προσεγγίσουμε τις διάφορες έννοιες και ορισμούς για να κατανοήσουμε και να είμαστε σε θέση να εφαρμόσουμε την εκτίμηση προσπάθειας.

Κεφάλαιο 3: Μετρικές

Παρόμοια με το Κεφάλαιο 2, γίνεται περιγραφή και παρουσίαση των μετρικών που επιλέξαμε να χρησιμοποιήσουμε στην εφαρμογή μας.

Κεφάλαιο 4: Υλοποίηση εφαρμογής

Γίνεται χρήση στιγμιοτύπων από την ολοκληρωμένη πλέον εφαρμογή, ώστε να δείξουμε πως από την θεωρία των προηγούμενων κεφαλαίων προχωρήσαμε στην πράξη.

ΚΕΦΑΛΑΙΟ 1 : Γενικές πληροφορίες

Εισαγωγή

Στο πρώτο κεφάλαιο γίνεται εκτενέστερη περιγραφή των σκέψεων και στόχων σχετικά με το αντικείμενο της εργασίας, παρατίθενται στοιχεία και απόψεις που συλλέχθηκαν από διάφορες πηγές.

Λειτουργεί ως στάδιο που μας εισάγει στα θέματα που θα μας απασχολήσουν και θα αναλύσουμε λεπτομερώς στα επόμενα κεφάλαια.

Γίνεται αναφορά στην ανάπτυξη λογισμικού υπό την μορφή components και των πλεονεκτημάτων που προσφέρουν.

Αναφέρονται διαθέσιμα εργαλεία που υπάρχουν στην αγορά τα οποία επιτελούν μία από τις κύριες διεργασίες που καλούμαστε να αναπτύξουμε στην εφαρμογή που σχεδιάζουμε, τον υπολογισμό μετρικών.

Γίνεται μια σύντομη αναφορά στην έννοια των περιπτώσεων χρήσης που συναντάμε κατά την μελέτη μηχανικής λογισμικού.

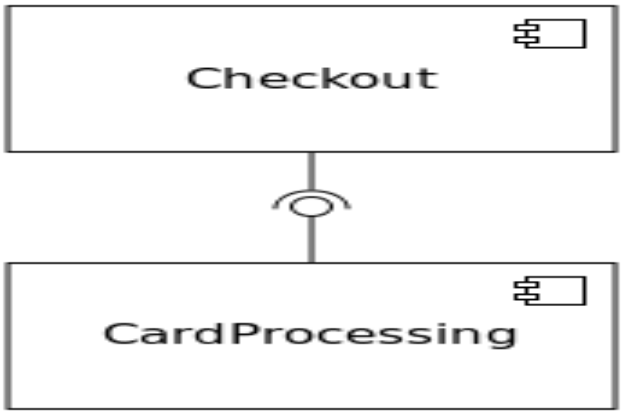
Τέλος, περιγράφεται το περιβάλλον που θα χρησιμοποιήσουμε (Visual Studio 2010) για να αναπτύξουμε την εφαρμογή μας, καθώς και κάποιες γενικές πληροφορίες σχετικά με αυτό.

1.1 Ανάπτυξη λογισμικού με την λογική των components

Ένα θέμα που μας απασχόλησε κατά τα αρχικά στάδια συγγραφής αυτής της εργασίας ήταν η ανάγκη για δημιουργία κώδικα με τέτοιο τρόπο, ώστε αργότερα να μπορεί να γίνει ικανότερη η επεκτασιμότητά του, αλλά και γενικότερα η εφαρμογή. Αυτό επιτεύχθηκε με τη μελέτη της έννοιας των components.

Ένα component είναι ένα αυτοτελές κομμάτι λογισμικού είτε είναι ένα πακέτο λογισμικού, είτε μια υπηρεσία, είτε ένα module που ενθυλακώνει ένα σύνολο από συσχετιζόμενες συναρτήσεις ή δεδομένα. Παραδείγματα components αποτελούν όλες οι διεργασίες συστημάτων καθώς είναι τοποθετημένες σε χωριστά «μέρη», ώστε μέσα σε κάθε «μέρος» να υπάρχουν δεδομένα που είναι σημασιολογικά αλληλένδετα μεταξύ τους.

Όσον αφορά τον ευρύτερο συντονισμό ενός συστήματος, τα components επικοινωνούν μεταξύ τους χρησιμοποιώντας τις διασυνδέσεις (interfaces). Όταν ένα component προσφέρει τις υπηρεσίες του στο σύστημα, υιοθετεί προσωρινά μία παρεχόμενη από το σύστημα διασύνδεση, η οποία οριοθετεί τις υπηρεσίες που άλλα components μπορούν να χρησιμοποιήσουν και με ποιον τρόπο. Αυτή η διασύνδεση μπορεί να φανεί ως υπογραφή του component, και αυτός που αιτείται να χρησιμοποιήσει τις υπηρεσίες του component δεν είναι αναγκαίο να έχει πρόσβαση στο εσωτερικό του κώδικά του. Γνωστή διαδικασία και ως ενθυλάκωση.

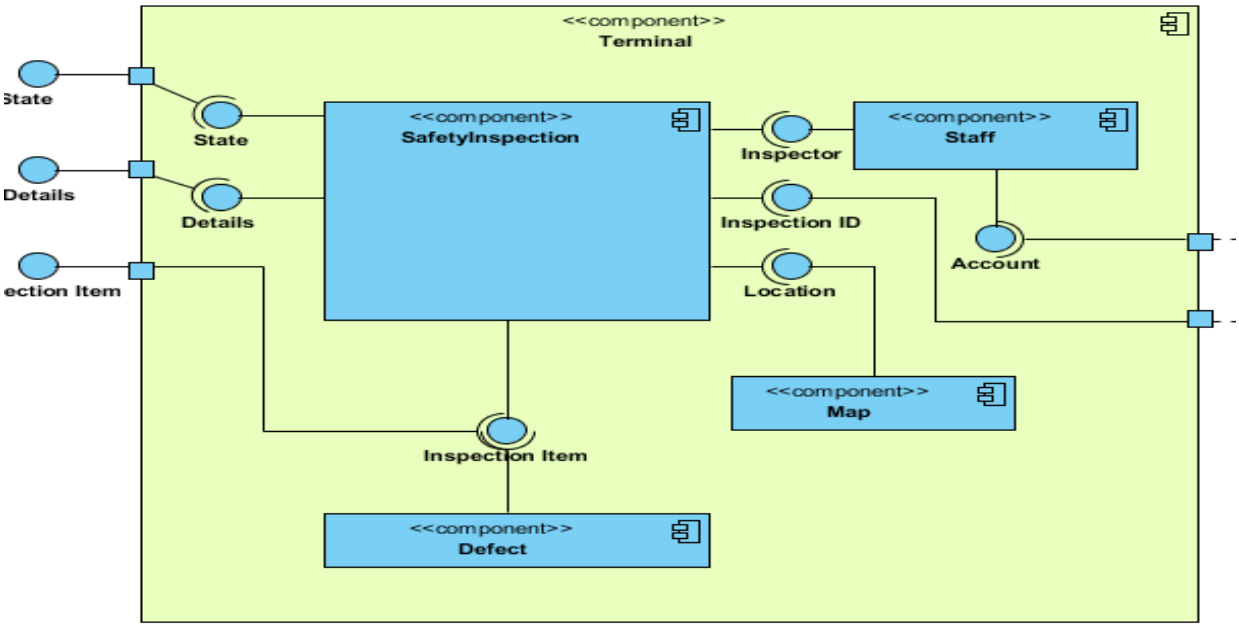


Σχήμα 1 : Παράδειγμα component

Σχετικά με την UML (Unified Modeling Language) που είναι και το αντικείμενο μελέτης μας, αυτές οι σχέσεις αναπαριστώνται διαγραμματικά (σχήμα 1, σχήμα 2).

Χαρακτηριστικό των components αποτελεί η ιδιότητα τους να λειτουργούν ως υποκατάστατα, καθώς μπορούμε πολύ εύκολα να αντικαταστήσουμε ένα component με ένα άλλο (είτε run-time, είτε design-time), εφόσον το νέο component πληροί τις προϋποθέσεις αντικατάστασης που έχει υποβάλει το παλιό component (γίνεται μέσω της οριοθέτησης του interface όπως είπαμε παραπάνω). Κάτι τέτοιο είναι πολύ χρήσιμο, καθώς μπορούμε να αντικαταστήσουμε ένα component με ένα νέο πιο ενημερωμένο, χωρίς να πάψει η λειτουργία ολόκληρου του συστήματος.

Άλλο σημαντικό χαρακτηριστικό αποτελεί η επαναχρησιμοποιησιμότητά τους, καθώς καθιστούν την ανάπτυξη λογισμικού κατά πολύ ταχύτερη (είναι έτοιμα, ελεγμένα κομμάτια κώδικα) και μπορούν να χρησιμοποιηθούν στην ανάπτυξη τελείως διαφορετικών προγραμμάτων [1].



Σχήμα 2 : Παράδειγμα component που εμπεριέχει άλλα components

Πλέον η χρήση των components ακολουθεί πλήρως την λογική της αντικειμενοστρέφειας, των θεωριών περί αντικειμένων, πλαισίων σχεδίασης (frameworks), και software design patterns και πιστεύεται πως μπορούν να γίνουν πλήρως αξιόπιστα και εσωτερικά τροποποιήσιμα.

1.2 Διαθέσιμα εργαλεία στην αγορά

Όπως έχει αναφερθεί και στα εισαγωγικά σχόλια αυτής της πτυχιακής εργασίας, ένας από τους στόχους της είναι η δημιουργία ενός συνολικού εργαλείου που θα είναι σε θέση να παρέχει ένα ικανοποιητικό GUI (graphics user interface), με δυνατότητα εξαγωγής effort estimation ανά περίπτωση χρήσης, καθώς και μετρικές από κομμάτια κώδικα (Java). Αυτή τη στιγμή δεν υπάρχει κάποιο τέτοιο εργαλείο freeware, οπότε καλούμαστε να αναπτύξουμε ένα πρώιμο στάδιο ενός τέτοιου ολοκληρωμένου εργαλείου. Στόχος μας είναι να παράξουμε μία αρχική σταθερή, γερά δομημένη version μιας τέτοιας εφαρμογής, που στην πορεία θα μπορεί να εμπλουτιστεί και με περισσότερες λειτουργίες ή και components.

1.2.1 freeware metric tools

Σε αυτό το υποκεφάλαιο παρατίθεται μία λίστα από προγράμματα-εφαρμογές που χρησιμοποιούνται για τον υπολογισμό μετρικών κώδικα. Η λίστα περιέχει τίτλους εφαρμογών και τις γενικότερες πληροφορίες σχετικά με αυτά. Στα πλαίσια αυτής της πτυχιακής εργασίας, δεν καλυπτόμαστε μόνο από τον υπολογισμό μετρικών κώδικα. Αυτή η λειτουργία όμως αποτελεί ένα βασικό κομμάτι της εφαρμογής που αναπτύχθηκε.

Πίνακας 1 : freeware metric tools

	Simple Licence	Input format	Distrib size	Source size	Written in	Metrics supported	Output	Comments
Free								
cloc	Free	source code	382kb	-	Perl		-	
CodeAnalyzer	Free	source code	411kb	-	Java		-	
CCCC	Free	source code	-	-	C/C++	cf report on tomcat	-	
Ohcount	Free	source code	307 kb	-	Ruby	loc	[txt] (default output)	Powers http://www.ohloh.net/. Detects licenses.
Metrics	Free	bytecode	1.9 Mb	13.51 KLOC (JavaNCSS)	Java	here		Needs an external xsl style sheet for formatting xml
JMT	Free	bytecode	485k		Java	here	[screenshot]	
DependencyFinder	Free	bytecode	3.5 Mb	17.41 KLOC (JavaNCSS)	Java	here	-	
JDepend	Free	bytecode	-	-	Java	here	-	
CKJM	Free	bytecode	1 Mb	0.33 KLOC (JavaNCSS)	Java	here	[txt]	Lighweight
Cyvis	Free	bytecode	3.6 Mb	2.5 KLOC (JavaNCSS)	Java	here	[screenshot]	Nice presentation
JavaNCSS	Free	source	1.1 Mb	37.49 KLOC (JavaNCSS)	Java	sloc	-	NCSS et CCN clearly defined
SLOCCount	Free	source	187k	3.5 KLOC	Perl/C	sloc	[txt]	
CodeCount	Free	source	-	3 KLOC	C	sloc	-	
LOCC (superseeds JavaCount)	Free	source	2.2 Mb	24.45 KLOC (JavaNCSS)	Java	sloc	-	

1.2.2 commercial metric tools

Παρόμοια με το προηγούμενο υποκεφάλαιο, παρατίθεται ένα πίνακας που περιέχει προγράμματα-εφαρμογές επί πληρωμή για τον υπολογισμό μετρικών, όπως και γενικές χρήσιμες πληροφορίες για αυτά:

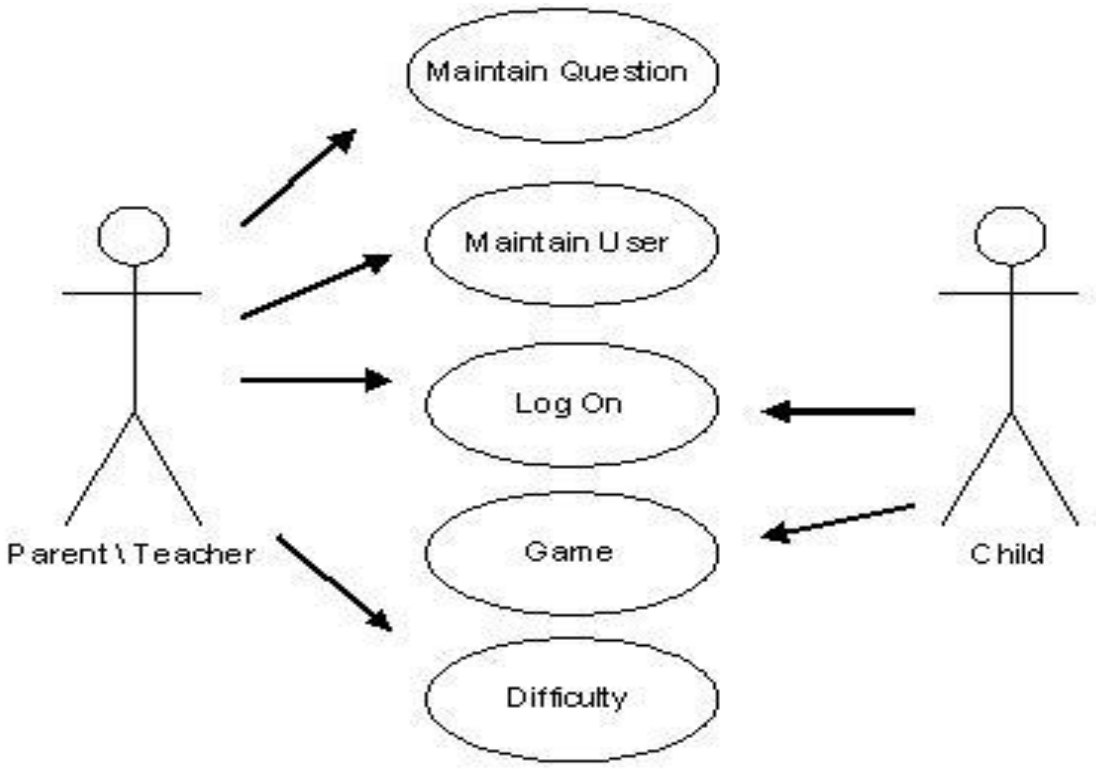
Πίνακας 2 : commercial metric tools

	Simple Licence	Input format	Distrib size	Source size	Written in	Metrics supported	Output	Comments
Logiscope/Kalimetrix	-	-	-	-	-		NA	Added March 2013, formerly supported by IBM
Squoring	-	-	108 MB win	-	Java	[html]	NA	Added March 2013
CodeReports	2000$	-	3.3Mb win	-	-	[html]	NA	
Ndepend	415$	-	6.3 Mb win	-	-	[html]	-	
JHawk	60$	-	936k eclipse	-	-	[html]	NA	
Understand for Java	495$	-	26.8 Mb linux/win	-	Perl	[html] [txt]	[sample]	
Resource Standard Metrics	200$	-	12.68Mb linux/win	-	-	[html]	[sample]	
Imagix 4D	2000$	-	4.5Mb linux/win	-	-	[html]	[sample]	
Essential Metrics	-	-	6Mb win	-	-	[pdf]	[sample]	
QSM SLIM-Metrics	-	-	-	-	-	-	-	
Semantic Designs: Java Source Code Metrics	250$	-	-	-	-	[html]	-	
Integrated Software Metrics / Predictive 3.0	-	-	unavailable 18/6/2008	-	-	-	-	

1.3 Περιπτώσεις Χρήσης

Γενικά στην μηχανική λογισμικού, περίπτωση χρήσης είναι ουσιαστικά μια λίστα βημάτων που περιγράφουν αλληλεπιδράσεις, μεταξύ ενός ρόλου που κατέχει κάποιο άτομο και του συστήματος που χρησιμοποιεί, προκειμένου να πετύχει κάποιο στόχο. Ως χρήστης ή αλλιώς actor συνήθως ορίζεται κάποιο φυσικό πρόσωπο, υπάρχουν όμως περιπτώσεις που χρήστης μπορεί να είναι και κάποιο εξωτερικό σύστημα.

Οι αλληλεπιδράσεις μεταξύ χρηστών και συστήματος μπορούν να αναπαρισταθούν με τη χρήση των διαγραμμάτων περιπτώσεων χρήσης.



Σχήμα 3 : διάγραμμα περίπτωσης χρήσης

Στα πλαίσια αυτής της εργασίας δεν θα μας απασχολήσει η γραφική απεικόνιση των περιπτώσεων χρήσης, αλλά η αναλυτική τους περιγραφή και καταγραφή σε κάποιο έγγραφο.

Ακολουθεί ένα παράδειγμα εγγράφου καταγραφής (πίνακας 3) μιας περίπτωσης χρήσης. Αυτό το έγγραφο αποτελεί μια απλή μορφή καταγραφής περίπτωσης χρήσης.

Πίνακας 3 : Έγγραφο καταγραφής περίπτωσης χρήσης

Όνομα Περίπτωσης Χρήσης	
Αριθμός Περίπτωσης Χρήσης	
Έκδοση	
Συγγραφείς (με χρονολογική σειρά)	
Προτεραιότητα (1-3)	
Χαρακτήρες που αφορά	
Προαπαιτούμενα	
Περιγραφή	
Εναλλακτική ροή Α	
Εναλλακτική ροή Β	
Μη λειτουργικές απαιτήσεις	

Αξίζει να αναφέρουμε πως δεν υπάρχει κάποιος συγκεκριμένος τύπος καταγραφής του περιεχομένου μιας περίπτωσης χρήσης, και απλώς χρησιμοποιούμε πιο απλούς τύπους καταγραφής ή πιο λεπτομερείς τύπους, ανάλογα με την κρίση μας και τις απαιτήσεις στο έργο που αναπτύσσουμε.

1.4 Visual Studio

Σχήμα 4 : Visual Studio Logo

Το περιβάλλον που επιλέχθηκε για να αναπτυχθεί το εργαλείο ανάπτυξης λογισμικού είναι το Visual Studio (έκδοση 2010). Επιλέχθηκε ανάμεσα από διάφορα άλλα γνωστά εργαλεία όπως το Eclipse, καθώς παρέχει δυνατότητες συγγραφής κώδικα σε διάφορες γλώσσες όπως C/C++ (μέσω της Visual C++), VB.NET (μέσω της Visual Basic .NET), C# (μέσω της Visual C#).

Συνεργάζεται πάρα πολύ καλά με τα διάφορα εργαλεία βάσεων δεδομένων, διαθέτει δυνατότητα δημιουργίας/διαχείρισης βάσης δεδομένων μέσα από το πρόγραμμα, database schema designer και αλλά και άλλα built-in tools όπως forms designer για GUI εφαρμογές, web designer και class designer.

Επίσης, μπορούμε πολύ εύκολα να εισάγουμε κώδικα διαφορετικών γλωσσών προγραμματισμού με τη βοήθεια των γλωσσικών υπηρεσιών (language services), οι οποίες επιτρέπουν στον code editor και τον debugger να υποστηρίξουν (σε διάφορα επίπεδα) σχεδόν οποιαδήποτε γλώσσα προγραμματισμού.

Τέλος, σημαντικό ρόλο στην επιλογή αυτού του περιβάλλοντος ανάπτυξης έπαιξε η ικανότητα του να δημιουργεί φόρμες και κλάσεις με τρόπο τέτοιο, που προάγει την συγγραφή κώδικα ως component και γραφικό περιβάλλον σε στυλ παραθύρων όπως τα windows, που είμαστε εξοικειωμένοι με αυτό.

Επίλογος

Στο πρώτο κεφάλαιο αυτής της εργασίας αναφερθήκαμε σε γενικές έννοιες που θα μας απασχολήσουν κατά τη διάρκεια συγγραφής αυτής της εργασίας καθώς και σε ζητήματα που μας απασχόλησαν κατά τα αρχικά στάδια συγγραφής της.

Το κεφάλαιο που ακολουθεί έχει πιο θεωρητικό περιεχόμενο και θεωρείται ένα από τα πιο σημαντικά σημεία που καλείται να καλύψει η εργασία αυτή.

ΚΕΦΑΛΑΙΟ 2 : Εκτίμηση προσπάθειας (Effort estimation)

Εισαγωγή

Σε αυτό το κεφάλαιο γίνεται αναφορά στους όρους της εκτίμησης προσπάθειας, στις επιμέρους έννοιες που την απαρτίζουν και που πρέπει να γνωρίζουμε και να κατανοήσουμε, ως θεωρητικό υπόβαθρο.

Στη συνέχεια, είμαστε έτοιμοι να προχωρήσουμε στην υλοποίηση αυτών, μέσω της εφαρμογής λογισμικού που αναπτύσσουμε στα πλαίσια αυτής της εργασίας.

Η εκτίμηση προσπάθειας που χρησιμοποιούμε είναι αυτή που αναπτύχθηκε από τον Gustav Karner το 1993. Η μέθοδος που την υλοποιεί, ονομάζεται «Σημεία Περιπτώσεων Χρήσης» (Use Case Points), βάσει της οποίας μπορεί να γίνει μια εκτίμηση της προσπάθειας που απαιτείται, πριν ξεκινήσει η ανάπτυξη οποιασδήποτε εφαρμογής.

2.1 Θεωρητική ανάπτυξη εκτίμησης προσπάθειας

Γενικότερα, projects με μεγάλες και πολύπλοκες περιπτώσεις χρήσης απαιτούν περισσότερο κόπο για να σχεδιαστούν και να υλοποιηθούν από ότι τα μικρότερα project με λιγότερο πολύπλοκες περιπτώσεις χρήσεις όπως αναφέρει ο Roy K. Clemmons στο [2]. Επιπλέον, ο χρόνος για να ολοκληρωθεί ένα έργο επηρεάζεται από τα ακόλουθα:

1. Τον αριθμό βημάτων για να ολοκληρωθούν οι περιπτώσεις χρήσης.
2. Τον αριθμό και την πολυπλοκότητα των χρηστών.
3. Τις τεχνικές απαιτήσεις των περιπτώσεων χρήσης, όπως οι παράλληλες εφαρμογές, η ασφάλεια και η επίδοση.
4. Διάφορους εξωτερικούς παράγοντες, όπως η εμπειρία και οι γνώσεις της ομάδας ανάπτυξης της εφαρμογής.

Μια μέθοδος εκτίμησης που λαμβάνει υπόψη τους παραπάνω παράγοντες νωρίς κατά την διάρκεια ανάπτυξης ενός έργου και που παράγει μια εκτίμηση (μέσα) στο 20% του πραγματικού χρόνου ολοκλήρωσης, πρόκειται να αποβεί πολύ χρήσιμη για τον σχεδιασμό ανάπτυξης του έργου, αλλά και του κόστους και αξιοποίησης των διαθέσιμων πόρων.

Όπως είπαμε προηγουμένως η μέθοδος που χρησιμοποιούμε είναι η Use Case Points, η οποία αναλύει τους συμμετέχοντες (actors) της περίπτωσης χρήσης, τα σενάρια, και ποικίλους τεχνικούς και περιβαλλοντολογικούς (εξωγενείς) παράγοντες που εμπεριέχονται όλα μαζί σε μία εξίσωση.

Την εξίσωση συνθέτουν τέσσερες (4) μεταβλητές:

1. Ο παράγοντας τεχνικής πολυπλοκότητας (TCF),
2. Ο παράγοντας εξωγενούς πολυπλοκότητας (ECF),
3. Σημεία αστάθμητου βάρους στις περιπτώσεις χρήσης (UUCP) και
4. Ο παράγοντας παραγωγικότητας (PF).

Κάθε μεταβλητή ορίζεται και υπολογίζεται ξεχωριστά από τις άλλες, χρησιμοποιώντας τιμές βαρύτητας, subjective values, και μεταβλητές περιορισμών. Οι τιμές βαρύτητας και οι μεταβλητές περιορισμών αρχικά χρησιμοποιήθηκαν με βάση τον ορισμό του Albrecht [3], αλλά στη συνέχεια εκ των πραγμάτων τροποποιήθηκαν από τους ανθρώπους των Objective Systems, σύμφωνα με την εμπειρία τους με την Objectory, μια μεθοδολογία που προτάθηκε από τον Ivar Jacobson για ανάπτυξη αντικειμενοστρεφών εφαρμογών [4]. Οι υποκειμενικές τιμές αποφασίζονται από την ομάδα ανάπτυξης, ανάλογα με την αντίληψή τους σχετικά με την τεχνική πολυπλοκότητα και αποδοτικότητα.

Όταν περιλαμβάνεται η παραγωγικότητα ως συντελεστής που εκφράζει το χρόνο, η εξίσωση μπορεί να χρησιμοποιηθεί για να υπολογίσει τον αριθμό ανθρωπο-ωρών που απαιτούνται για να ολοκληρωθεί ένα πρόγραμμα.

Η τυπική μορφή της εξίσωσης βάσει των παραπάνω είναι :

$$\mathbf{UCP = TCP * ECF * UUCP * PF} \qquad (2.1)$$

Τα απαραίτητα βήματα που ακολουθούμε για να υπολογίσουμε την εκτίμηση είναι:

1. Απόφαση και υπολογισμός των τεχνικών παραγόντων,
2. Απόφαση και υπολογισμός των εξωγενών παραγόντων,
3. Υπολογισμός του αστάθμητου βάρους σημείων των περιπτώσεων χρήσης,
4. Απόφαση για το ποιος είναι ο παράγοντας παραγωγικότητας και
5. Υπολογισμός τελικού αποτελέσματος-προϊόντος των μεταβλητών.

Παρακάτω ακολουθεί η αναλυτικότερη αναφορά στις επιμέρους μεταβλητές που συνθέτουν την εξίσωση της εκτίμησης προσπάθειας:

2.2 Σημεία αστάθμητου βάρους στις περιπτώσεις χρήσης (UUCP)

Τα UUCPs υπολογίζονται κάνοντας δυο (2) επιμέρους υπολογισμούς:

1. The Unadjusted Use Case Weight (UUCW) που βασίζεται στον συνολικό αριθμό δραστηριοτήτων (ή βημάτων) που εμπεριέχονται σε όλα τα σενάρια περιπτώσεων χρήσης.
2. The Unadjusted Actor Weight (UAW) που βασίζεται στην συνδυασμένη πολυπλοκότητα όλων των συμμετεχόντων (actors) σε όλες τις περιπτώσεις χρήσης.

2.2.1 Unadjusted Use Case Weight (UUCW)

Το UUCW προέρχεται από τον αριθμό των περιπτώσεων χρήσης και χωρίζονται σε τρεις κατηγορίες: simple, average, complex. Κάθε περίπτωση χρήσης κατηγοριοποιείται με βάση τον αριθμό βημάτων που το σενάριό της περιέχει, συμπεριλαμβανομένων και των εναλλακτικών ροών.

Σε αυτό το σημείο, αξίζει να σημειώσουμε πως ο αριθμός βημάτων σε ένα σενάριο επηρεάζει την εκτίμηση. Ένας μεγάλος αριθμός βημάτων σε ένα σενάριο περίπτωσης χρήσης θα προδιαθέσει το UUCW ως προς την πολυπλοκότητα και παράλληλα θα αυξήσει το UCP. Αντίθετα, ένας μικρός αριθμός βημάτων θα προδιαθέσουν το UUCW ως προς την απλότητα και θα μειώσουν το UCP. Παρόλα αυτά έχουν παρατηρηθεί περιπτώσεις που μειώνονται τα βήματα χωρίς να επηρεάζουν καθόλου τη συνολική διαδικασία.

Το UUCW υπολογίζεται αθροίζοντας τον αριθμό των περιπτώσεων χρήσης μιας κατηγορίας, πολλαπλασιάζοντας το κάθε σύνολο με τον προκαθορισμένο του παράγοντα βαρύτητας και στη συνέχεια προσθέτοντας τα αποτελέσματα.

Πίνακας 4 : Παράδειγμα UUCW

Use Case Type	Description	Weight
Simple	A simple user interface and touches only a single database entity; its success scenario has 3 steps	5

	or less; its implementation involves less than 5 classes.	
Average	More interface design and touches 2 or more database entities; between 4 to 7 steps; its implementation involves between 5 to 10 classes.	10
Complex	Involves a complex user interface or processing and touches 3 or more database entities; over seven steps; its implementation involves more than 10 classes.	15

Σημείωση: Όσον αφορά την εφαρμογή που αναπτύχθηκε δεν κατασκευάζουμε τον πίνακα που αθροίζει τα UUCW για όλες τις περιπτώσεις χρήσης (σε επιπεδο εφαρμογής), καθώς η μελέτη που κάνουμε γίνεται σε επίπεδο περίπτωσης χρήσης(δηλαδή εξετάζουμε μια-μια τις περιπτώσεις χρήσης).

2.2.2 Unadjusted Actor Weight (UAW)

Με παρόμοιο τρόπο οι τύποι των συμμετεχόντων ορίζονται ως simple, average, complex. Το UAW υπολογίζεται προσθέτοντας τον αριθμό των συμμετεχόντων σε κάθε κατηγορία, πολλαπλασιάζοντας αυτό το σύνολο με τον προκαθορισμένο παράγοντα βαρύτητάς του και τελικά προσθέτοντας τα παραγόμενα.

Τέλος, το UUCP υπολογίζεται προσθέτοντας την τελική τιμή του UUCW με το UAW.

UUCP = UUCW + UAW

Σημείωση: Το UUCP είναι ανεξάρτητο πλήρως, και δεν μετράει για τις άλλες μεταβλητές της εξίσωσης εκτίμησης προσπάθειας (γιαυτό και αποκαλείται unadjusted).

Πίνακας 5 : Παράδειγμα UAW

Actor Type	Description	Weight	Number of Actors	Result
Simple	The Actor represents another system with a defined API	1	2	2
Average	The Actor represents another system interacting through a protocol, like TCP/IP	2	0	0
Complex	The Actor is a person interacting via an interface.	3	1	3
			Total UAW	**5**

2.3 Παράγοντας τεχνικής πολυπλοκότητας (TCF)

Υπάρχουν δέκα τρεις (13) τεχνικοί παράγοντες για να υπολογίσουν τις επιπτώσεις που επιφέρουν στην παραγωγικότητα ενός έργου, τα διάφορα τεχνικά προβλήματα.

Κάθε παράγοντας παίρνει βαρύτητα ανάλογα με την σχετική του επίπτωση. Ανάλογα με το έργο οι τεχνικοί παράγοντες, ελέγχονται από την ομάδα ανάπτυξης η οποία και τους αποδίδει μία τιμή (perceived complexity) μεταξύ μηδέν (0) και πέντε (5). Η τιμή που αποδίδεται (0-5) κρίνεται με βάση την πρόβλεψη που μπορούν να κάνουν τα μέλη της ομάδας ανάπτυξης όσον αφορά την πολυπλοκότητα του έργου προς ανάπτυξη.

Για παράδειγμα εφαρμογές εκτενούς multitasking χρειάζονται περισσότερη γνώση και χρόνο από τους προγραμματιστές από ότι μια εφαρμογή single-task. Για τιμή perceived complexity 0 σημαίνει πως ο τεχνικός παράγοντας είναι ελάσσονος σημασίας για το project, για τιμή 3 είναι μέτριας και για 5 έχει ισχυρή επιρροή. Σε περίπτωση αμφιβολίας αποδίδεται τιμή 3.

Κάθε παράγοντας βαρύτητας πολλαπλασιάζεται από τον παράγοντα perceived complexity του, ώστε να παράγουν τον calculated παράγοντα. Αυτοί οι παράγοντες που προκύπτουν αθροίζονται για να δώσουν τον Technical Total παράγοντα. Επιπλέον, δύο (2) μεταβλητές περιορισμού υπολογίζονται για να δώσουν την τιμή του TCF. Αυτές οριοθετούν την επίδραση που έχει η TCF στην εξίσωση της UCP από 0.6 (όλες οι perceived complexities 0) το ελάχιστο έως 1.3 (όλες οι perceived complexities 5) το μέγιστο.

Οι τιμές TCF μικρότερες του 1 μειώνουν το UCP καθώς κάθε θετική τιμή που πολλαπλασιάζεται με μία τιμή μικρότερή της, μειώνει το τελικό μέγεθός της (π.χ 100 * 0.60 = 60 –μία μείωση της τάξεως του 40%). Αντίστοιχα, και για τιμή μεγαλύτερη του ευατού της, αυξάνεται το τελικό της μέγεθος (π.χ 100 * 1.30 = 130 –μία αύξηση της τάξεως του 30%). Λόγω των παραπάνω μπορούμε να πούμε πως η TCF εν τέλει, μπορεί να επηρεάσει την UCP απο -40% (0.6) έως +30% (1.3).

Τέλος, η μαθηματική διατύπωση της TCF είναι:

$$TCF = C_1 + C_2 \sum_{i=1}^{13} W_i * F_1 \qquad (2.2)$$

Όπου,

Constant 1 (C1) = 0.6
Constant 2 (C2) = .01
W = Weight
F = Perceived Complexity Factor

Και η γενική της, πιο εύχρηστη μορφή (αυτή θα χρησιμοποιούμε εμείς):

TCF = 0.6 + (.01 * Technical Total Factor) (2.3)

2.4 Παράγοντας εξωγενούς πολυπλοκότητας (ECF)

Η τιμή της ECF παράγει μία εκτίμηση για την εμπειρία της ομάδας ανάπτυξης. Πιο έμπειρες ομάδες παίζουν μεγαλύτερο ρόλο στη διαμόρφωση της τελικής τιμής της UCP από ότι ομάδες με λιγότερη εμπειρία.

Η κάθε ομάδα ανάπτυξης αποφασίζει την τιμή του κάθε παράγοντα perceived impact με βάση την αντίληψή της για το πόσο μπορεί να επηρεάσει στην επιτυχημένη ολοκλήρωση του έργου. Η τιμή 0 υποδεικνύει πως δεν επηρεάζει στην επιτυχία του έργου, η τιμή 1 υποδεικνύει πως ο παράγοντας έχει ισχυρή αρνητική επίπτωση στο έργο, η τιμή 3 μέτρια και η τιμή 5 πως έχει ισχυρή θετική επίπτωση.

Παραδείγματος χάριν, μέλη της ομάδας ανάπτυξης που έχουν μικρό ή καθόλου κίνητρο για το project τους, θα έχουν ισχυρή αρνητική επίπτωση (1), ενώ μέλη με πολλές γνώσεις αντικειμενοστρεφούς προγραμματισμού, έχουν ισχυρή θετική επίπτωση (5) στην επιτυχία του project.

Κάθε βάρος παράγοντα πολλαπλασιάζεται από το perceived impact του, ώστε να παράγει τον υπολογισμένο παράγοντα (calculated factor). Στη συνέχεια όλοι οι επιμέρους παράγοντες αθροίζονται δημιουργώντας τον συνολικό εξωγενή παράγοντα (Environmental Total Factor). Όσο μεγαλύτερες οι τιμές του, τόσο περισσότερο επηρεάζει την εξίσωση του UCP.

Παρόμοια με τον υπολογισμό του TCF, για να υπολογίσουμε το τελικό ECF χρησιμοποιούμε και δύο (2) μεταβλητές περιορισμών στην εξίσωση. Οι τιμές τους βασίζονται σε εμπειρικές τιμές που προκύπτουν από άτομα με πολύχρονη απασχόληση σε ανάπτυξη συστημάτων.

Βάσει αυτών, οι περιορισμοί κυμαίνονται μεταξύ, 0.425 (για προγραμματιστές μερικής απασχόλησης και δυσκολία γλώσσας προγραμματισμού μηδενική, όλες οι άλλες τιμές=5) και 1.4 (όλες οι τιμές perceived impact=0).
Συνοψίζοντας, η ECF μπορεί να μειώσει την τιμή της UCP έως και 57.5% ή αντίστοιχα να την αυξήσει κατά 40%. Συνεπώς μπορούμε να πούμε πως η ECF είναι σε θέση να επηρεάσει την UCP περισσότερο από την TCF.

Τέλος, η μαθηματική διατύπωση της ECF είναι:

$$ECF = C_1 + C_2 \sum_{i=1}^{8} W_i * F_1 \quad (2.4)$$

Όπου,

Constant 1 (C1) = 1.4

Constant 2 (C2) = -0.03

W = Weight

F = Perceived Impact

Και η γενική της, πιο εύχρηστη μορφή (αυτή θα χρησιμοποιούμε εμείς):

ECF = 1.4 + (-0.03 * Environmental Total Factor) (2.5)

Πίνακας 6 : Παράδειγμα ECF

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor (weight*perceived complexity)
E1	Familiarity with UML	1.5	3	4.5
E2	Application Experience	0.5	2	1
E3	Object Oriented Experience	1	2	2
E4	Lead analyst capability	0.5	2	1
E5	Motivation	1	2	2
E6	Stable Requirements	2	3	6
E7	Part-time workers	-1	3	-3

E8	Difficult Programming language	2	3	6
			Total Factor	**19.5**

2.5 Παράγοντας παραγωγικότητας (PF)

Ο παράγοντας παραγωγικότητας (PF) είναι μια αναλογία των ωρών εργασίας ανά περίπτωση χρήσης βασιζόμενη σε παλιότερες περιπτώσεις. Αν δεν υπάρχουν παλιότερα στοιχεία, δίνεται μια τιμή ανάμεσα στο 15 και στο 30, με συνήθως προτεινόμενη τιμή το 20.

Κανονικά ο συνολικός χρόνος υπολογισμού χρόνου ολοκλήρωσης ενός project υπολογίζεται πολλαπλασιάζοντας το UCP με το PF:

Συνολικός Χρόνος = UCP * PF (2.6)

Αλλά όπως αναφέραμε στην αρχή του κεφαλαίου θα χρησιμοποιήσουμε για τις ανάγκες της εργασίας την πιο άμεση μορφή εξίσωσης που υπολογίζει την τιμή της Use Case Points:
UCP = TCP * ECF * UUCP * PF

Όπου, ενσωματώνουμε την PF στον υπολογισμό της UCP.

Η τιμή που παράγεται αντιστοιχεί σε ώρες εργασίας που απαιτούνται ώστε να ολοκληρωθεί μια περίπτωση χρήσης. Παράδειγμα : UCP = 240.

Στην κυρίως εφαρμογή, δίνεται η δυνατότητα να μετατρέψουμε αυτή την τιμή σε ημέρες, εβδομάδες, μήνες, έτος , δημιουργώντας συναρτήσεις που κάνουν αυτή τη δουλειά. Παραδείγματος χάριν, διαιρώντας την παραπάνω τιμή δια του 40 (για μια εβδομάδα εργασίας ενός ανθρώπου) παίρνουμε την τιμή 6, δηλαδή 6 εβδομάδες εργασίας ή αλλιώς 1.5 μήνα.

Επίλογος

Σε αυτό το κεφάλαιο παράλληλα με την θεωρητική ανάλυση εννοιών, έγινε και η μαθηματική ερμηνεία των τύπων που χρησιμοποιούμε, ώστε να υπολογίσουμε την εκτίμηση προσπάθειας.

Η εκτίμηση προσπάθειας που θα χρησιμοποιήσουμε στα πλαίσια αυτής της πτυχιακής εργασίας γίνεται σε επίπεδο περίπτωσης χρήσης, γιαυτό όπως σχολιάστηκε και παραπάνω, λειτουργήσαμε αφαιρετικά όσον αφορά την χρήση κάποιων εννοιών.

Στο κεφάλαιο που ακολουθεί θα ασχοληθούμε με την έννοια των μετρικών με παρόμοια νοοτροποία ανάλυσης που ακολουθήσαμε σε αυτό.

ΚΕΦΑΛΑΙΟ 3 : Μετρικές

Εισαγωγή

Στο κεφάλαιο αυτό γίνεται αναφορά στις μετρικές και στις έννοιες περί αυτών που πρέπει να γνωρίζουμε ως θεωρητικό υπόβαθρο, προκειμένου να είμαστε έτοιμοι να προχωρήσουμε στην υλοποίησή τους, στην εφαρμογή λογισμικού που αναπτύσσουμε στα πλαίσια αυτής της εργασίας.

Οι μετρικές είναι αυτές που έχουν προταθεί αρχικά από τους S.R Chidamber και C.F Kemerer το 1991 [5].

Μετά την θεωρητική παρουσίαση ακολουθεί η παρουσίαση και αναφορά στο πρόγραμμα που χρησιμοποιήθηκε και ενσωματώθηκε στην κύρια εφαρμογή μας για να παράγει τα αποτελέσματα υπολογισμού των μετρικών.

3.1 Θεωρητική ανάπτυξη όρων μετρικών

3.1.1 Κυκλωματική πολυπλοκότητα (Cyclomatic Complexity-CC)

Η κυκλωματική πολυπλοκότητα προτάθηκε για πρώτη φορά ως μέτρηση μιας λογικής πολυπλοκότητας από τον T.J McCabe [6] το 1976. Σκοπός αυτής της μετρικής είναι να αξιολογεί την συντηρησιμότητα των μονάδων λογισμικού, γιαυτό και έχει χρησιμοποιηθεί ευρέως στους ερευνητικούς τομείς αυτού του αντικειμένου.

Στην πράξη η μετρική, χρησιμοποιείται για να υπολογίσει το χαμηλότερο όριο στον αριθμό μεθόδων που πρέπει να σχεδιαστούν και να εκτελεσθούν ώστε να είναι εγγυημένη η κάλυψη όλων των δηλώσεων σε ένα πρόγραμμα λογισμικού. Μια άλλη εφαρμογή της μετρικής είναι η χρήση της ως δείκτη αξιοπιστίας σε ένα σύστημα λογισμικού. Οι πειραματικές μελέτες δείχνουν έναν ισχυρό συσχετισμό μεταξύ της μετρικής του McCabe και του αριθμού λαθών που υπάρχουν στον πηγαίο κώδικα, καθώς επίσης και στο χρόνο που απαιτείται για να βρεί κάποιος και να διορθώσει τέτοια λάθη [7].

Είναι δύσκολο να βρεθεί μια πρακτική εφαρμογή αυτής της μετρικής που να αφορά την αντικειμενοστρέφεια, παρόλα αυτά, η μετρική μπορεί να χρησιμοποηθεί σαν ένας πολύ καλός δείκτης της πολυπλοκότητας μιας μεθόδου, η οποία μπορεί έμμεσα να χρησιμοποιηθεί για να υπολογίσει την πολυπλοκότητα μιας κλάσης. Δεδομένου ότι η μετρική CC είναι ένα μέτρο πολυπλοκότητας είναι επιθυμητό οι τιμές της να κρατηθούν όσο το δυνατόν χαμηλότερα.

Το ανώτερο όριο της CC έχει αποτελέσει ένα αντικείμενο συζήτησης για πολλά χρόνια και ο ίδιος ο McCabe προτείνει, βάσει εμπειρικών στοιχείων, την τιμή 10 ως πρακτικά το ανώτερο όριο. Υποστηρίζει ότι εάν μια μέθοδος υπερβαίνει αυτήν την τιμή γίνεται δύσκολο να εξεταστεί όπως πρέπει. Άλλες εμπειρικές μελέτες παράγουν παρόμοια αποτελέσματα και ορίζουν όρια μεταξύ 10 και 20 ως ανώτερα της CC [8].

Ορισμός:

$$v(G) = e - n + 2 \qquad (3.1)$$

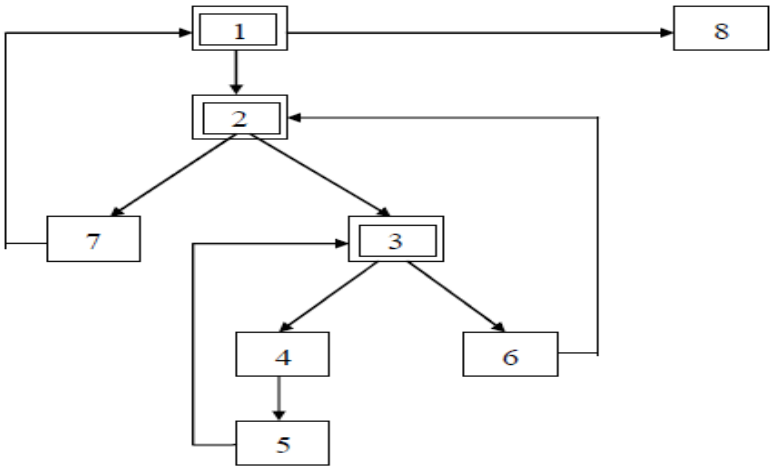
όπου v(G) η κυκλωματική πολυπλοκότητα του γραφήματος ροής G, της μεθόδου υπό εξέταση, όπου e ο αριθμός των ακμών στο G και n ο αριθμός των κόμβων στο G.

Παράδειγμα ψευδοκώδικα ώστε να γίνει κατανοητός ο τρόπος υπολογισμού της μετρικής:

```
   bubbleSort(array A)
1:          do while A not sorted
              set A as sorted
2:              for i=1 until i<A.size do
3:                if A.[i] < A.[i-1]
4:                  swap(A[i-1],A[i])
                    set A as unsorted
5:                end_if
6:              end_for
7:          end_while
8: end_bubbleSort
```

Σχήμα 5 : Παράδειγμα ψευδοκώδικα υπολογισμού CC

Απομένει να μετατρέψουμε αυτό το κομμάτι κώδικα σε ένα γράφημα ροής, έτσι ώστε να μπορέσουμε να υπολογίσουμε την κυκλωματική πολυπλοκότητα. Αυτό γίνεται χρησιμοποιώντας τα νούμερα που χαρακτηρίζουν τις γραμμές κώδικα πιο πάνω, ως κόμβους στο γράφημά μας.



Σχήμα 6 : Παράδειγμα γραφήματος ροής, κυκλωματικής πολυπλοκότητας

Ο αριθμός των ακμών είναι ίσος με δέκα (10) και ο αριθμός των κόμβων ίσος με οκτώ (8). Αυτά παράγουν την τιμή CC v(G) = 10 - 8 + 2 = 4.

Ένας ακόμη τρόπος να υπολογίσουμε το v(G) είναι:

$$v(G) = P+1 \qquad (3.2)$$

όπου P είναι ο αριθμός των κόμβων κατηγορήματος (predicate nodes)* που βρίσκονται στο G. Στην παραπάνω εικόνα οι κόμβοι με αριθμούς 1, 2 και 3 είναι κόμβοι κατηγορήματος, οπότε δίνουν τιμή v(G) = 3 + 1 = 4.

Σημείωση: Έγινε αναφορά σε αυτή την μετρική, καθώς αποτελεί προϋπόθεση της μετρικής που θα μελετήσουμε στη συνέχεια, την WMC. Η CC δεν υπολογίζεται εμφανώς, ούτε αποτελεί κομμάτι που υλοποιείται στο εργαλείο λογισμικού που αναπτύχθηκε.

**κόμβο κατηγορήματος (predicate node) ονομάζουμε τον κόμβο που παριστάνει μία Boolean (true/false) δήλωση στον κώδικα, γιαυτό και δημιουργεί δύο (2) εξερχόμενες ακμές αντί για μία.*

3.1.2 Σταθμισμένες μέθοδοι ανά κλάση (Weighted methods per class-WMC)

Οι S.R Chidamber και C.F Kemerer πρότειναν πρώτη φορά τη μετρική WMC το 1991 [5] που αφορά άμεσα στον καθορισμό της πολυπλοκότητας ενός αντικειμένου. Ο αριθμός μεθόδων και η πολυπλοκότητα των μεθόδων που περιλαμβάνονται σε μία κλάση δείχνουν πόσος χρόνος και προσπάθεια απαιτείται για να αναπτύξει και να διατηρήσει κάποιος ένα αντικείμενο.

Όσο μεγαλύτερος ο αριθμός μεθόδων σε αυτή την κλάση, τόσο μεγαλύτερος ο πιθανός αντίκτυπος στα παιδιά, καθώς τα παιδιά θα κληρονομήσουν όλες τις μεθόδους της. Ένας πολύ μεγάλος αριθμός μεθόδων μπορεί να οδηγήσει σε ένα πολύ συγκεκριμένο αντικείμενο εφαρμογής, περιορίζοντας κατά συνέπεια, τη δυνατότητα επαναχρησιμοποίησης [9].

Δεδομένου ότι η WMC μπορεί να περιγραφεί ως επέκταση της CC μετρικής, η συνιστώμενη αξία κατώτατων ορίων της, μπορεί να συγκριθεί με το ανώτερο όριο της CC μετρικής. Μπορούμε να πάρουμε την τιμή της WMC και να την διαιρέσουμε με τον αριθμό μεθόδων. Αυτή η τιμή που προκύπτει μπορεί να συγκριθεί με το ανώτερο όριο της αντίστοιχης CC. Ένα μειονέκτημα της χρησιμοποίησης της CC προκειμένου να μετρηθεί η πολυπλοκότητα αντικειμένων είναι ότι η τιμή της WMC δεν μπορεί να υπολογιστεί στα αρχικά στάδια σχεδίου, παραδείγματος χάριν, όταν καθοριστούν οι μέθοδοι σε μια κλάση, αλλά δεν έχουν ακόμη εφαρμοστεί.

Για να είναι σε θέση κάποιος να μετρήσει την WMC όσο το δυνατόν νωρίτερα κατά την συγγραφή κώδικα, θα πρέπει να χρησιμοποιήσει τον αριθμό μεθόδων ως αξία πολυπλοκότητας. Κάνοντας κάτι τέτοιο όμως, η WMC, παύει να αποτελεί μέτρο πολυπλοκότητας και γίνεται μέτρο μεγέθους, γνωστό ως η μετρική αριθμού μεθόδων (Number of Methods-NM) [10].

Ορισμός:

Ας θεωρήσουμε μία κλάση c1, με μεθόδους M1,M2,….Mn. Με c1,c2,…cn να είναι η στατική πολυπλοκότητα των μεθόδων. Τότε:

$$\mathrm{WMC} = \sum_{i}^{n} c_i \qquad (3.3)$$

Όπου,

n ο αριθμός των μεθόδων στην κλάση.

Αν όλες οι στατικές πολυπλοκότητες θεωρούνται μία ενότητα, η WMC ορίζεται ως ο αριθμός των μεθόδων (WMC = n).

Η στατική πολυπλοκότητα μιας μεθόδου μπορεί να μετρηθεί με πολλούς τρόπους (π.χ. CC) και οι δημιουργοί αυτής της μετρικής αφήνουν αυτή, να είναι μια επιλογή κατά την στιγμή υλοποίησης-συγγραφής κώδικα.

3.1.3 Έλλειψη συνοχής στις μεθόδους (Lack of Cohesion in Methods -LCOM)

Η συνοχή μιας κλάσης χαρακτηρίζεται από το πόσο καλά οι τοπικές μέθοδοι συσχετίζονται με τις τοπικές μεταβλητές δήλωσης σε μια κλάση . Οι S.R Chidamber και C.F Kemerer όρισαν τη μετρική LCOM το 1991 [5]. Από τότε, διάφοροι ορισμοί της LCOM έχουν προταθεί και συνεχίζουν ακόμη να γίνονται έρευνες σχετικά με αυτό. Η LCOM μετρική είναι μια αξία της ανομοιότητας των μεθόδων σε μια κλάση.

Μια υψηλή τιμή της LCOM σε μια κλάση υποδεικνύει ότι θα πρέπει να χωριστεί η κλάση σε δύο ή περισσότερες υποκλάσεις. Δεδομένου ότι μία κλάση μπορεί να έχει πολλές διαφορετικές εργασίες να εκτελέσει, είναι καλύτερο (design-wise) να χρησιμοποιηθούν πιο συγκεκριμένα αντικείμενα. Επειδή η LCOM είναι μια τιμή που προσδιορίζει την ανομοιότητα μεθόδων, βοηθά να εντοπιστούν οι ατέλειες στο σχεδιασμό τους [5].

Η συνεκτικότητα μεθόδων μέσα σε μια κατηγορία είναι επιθυμητή, δεδομένου ότι προωθεί την ενθυλάκωση και μειώνει την πολυπλοκότητα των αντικειμένων. Είναι δύσκολο να δοθούν οποιεσδήποτε ακριβείς τιμές κατωτάτων ορίων της

μετρικής LCOM, καθώς το αποτέλεσμα μπορεί να προσεγγιστεί από διαφορετικές οπτικές γωνίες, όπως η ικανότητα επαναχρησιμοποίησης, η πολυπλοκότητα και η συντηρησιμότητα μιας κλάσης.

Οι διάφορες μελέτες σχετικά με την LCOM, έχουν δείξει ότι οι υψηλές τιμές της συνδέθηκαν με χαμηλότερη παραγωγικότητα, μεγαλύτερη επανάληψη και μεγαλύτερη προσπάθεια σχεδίασης. Οι μελέτες δείχνουν επίσης πως η LCOM μπορεί να χρησιμοποιηθεί ως «προάγγελος» προσπάθειας συντήρησης [11].

Ορισμός LCOM κατά Chidamber-Kemerer

Θεωρείστε μία κλάση C_1 με n μεθόδους $M_1, M_2, \ldots, M_n$. Και $\{I_j\}$= ένα σετ μεταβλητών που χρησιμοποιούνται από τη μέθοδο M_i. Υπάρχουν n τέτοια σετ $\{I_1\}, \ldots, \{I_n\}$.

Ώστε να ισχύει $P = \{(I_i, I_j) \mid I_i \cap I_j = ø\}$ και $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq ø\}$.

Εάν όλα τα σετ $\{I_1\}, \ldots, \{I_n\}$ είναι ø τότε P = ø.

LCOM = |P| - |Q|, if |P| > |Q| (3.4)

Αλλιώς LCOM=0. (3.5)

Ο ορισμός της ανεπαρκούς διαύγειας περί ένωσης των σετ στην προσέγγιση των Chidamber-Kemerer, είναι κάπως διφορούμενος, γιαυτό και έγινε περαιτέρω προσπάθεια απόδοσης ορισμού μεταγενέστερα από άλλους, όπως ο Li και Henry [12]. Για τους σκοπούς αυτής της εργασίας θα αρκεστούμε στον αρχικό ορισμό (Chidamber-Kemerer) και δεν θα αναφερθούμε σε άλλες προσεγγίσεις.

3.1.4 Βάθος δένδρου κληρονομικότητας (Depth of Inheritance Tree-DIT)

Κληρονομικότητα έχουμε όταν μια κλάση μοιράζεται την ίδια δομή ή συμπεριφορά που ορίζεται σε μια άλλη κλάση. Όταν μια υποκλάση κληρονομεί από μία μόνο υπερκλάση, έχουμε ενιαία κληρονομικότητα (single inheritance) και όταν κληρονομεί από περισσότερες από μία υπερκλάσεις, έχουμε πολλαπλή κληρονομικότητα (multiple inheritance). Η κληρονομικότητα μεταξύ των κλάσεων αυξάνει την αποδοτικότητά τους και παράλληλα μειώνει τους πλεονασμούς.

Παρόλα αυτά, όσο βαθύτερη είναι η ιεραρχία κληρονομικότητας, τόσο μεγαλύτερη η πιθανότητα να γίνεται πιο πολύπλοκο και πιο δύσκολο να προβλέψουμε την συμπεριφορά των εμπλεκόμενων κλάσεων. Προκειμένου να ορίσουν ένα μέτρο βάθους στην ιεραρχία οι Shyam P. Chidamber και Chris F. Kemerer [5] (1991) πρότειναν την μετρική του βάθους δένδρου κληρονομικότητας.

Ο Joshua Bloch στο [13] επισημαίνει μερικούς κινδύνους με τη χρησιμοποίηση της κληρονομικότητας στα πακέτα (packages), δεδομένου ότι τα πακέτα συχνά γράφονται από διαφορετικά άτομα κατά την ανάπτυξη λογισμικού. Ένα πρόβλημα έγκειται στο ότι η υποκλάση εξαρτάται από την υπερκλάση της, και εάν γίνει μια αλλαγή στην δομή της υπερκλάσης, μπορεί να αναγκάσει την υποκλάση να καταρρεύσει.

Σύμφωνα με τον Bloch , η κληρονομικότητα πρέπει να χρησιμοποιείται μόνο όταν είναι εντελώς βέβαιο ότι μια κλάση Α που επεκτείνει μια κλάση Β είναι πραγματικά μια ειδίκευση της κλάσης Β. Εάν δεν είμαστε τελείως σίγουροι για τη σχέση αυτή, πρέπει να χρησιμοποιούμε σύνθεση .

Σύνθεση έχουμε όταν σε μια νέα κλάση δίνεται ένα ιδιωτικό πεδίο, το οποίο σχετίζεται με κάποια αναφορά υπάρχουσας κλάσης, και η υπάρχουσα κλάση γίνεται συστατικό της νέας. Όταν η επέκταση των κλάσεων σχεδιάζεται συγκεκριμένα με σκοπό και δυνατότητες επέκτασης ή οι κλάσεις βρίσκονται στο ίδιο πακέτο που αναπτύσσεται μονίμως από τους ίδιους προγραμματιστές, τότε η κληρονομικότητα είναι ένας ισχυρός τρόπος να πετύχουμε επαναχρησιμοποίηση κώδικα [13].

Έχουν υπάρξει διάφορες μελέτες σχετικά με τα κατώτατα επιτρεπτά όρια της DIT σε ένα σύστημα, αλλά δεν έχει διατυπώσει ακόμη κανείς με σαφήνεια, καθοριστικά όρια σχετικά με αυτό. Πάντως ισχύει πως για υψηλή τιμή του DIT υπάρχει καλή ικανότητα επαναχρησιμοποίησης, αλλά ο σχεδιασμός είναι πιο

σύνθετος και δυσνόητος. Είναι θέμα του προγραμματιστή να ορίσει ορθά, ένα κατάλληλο κατώτατο όριο, ανάλογα με τις τρέχουσες ιδιότητες και ανάγκες του συστήματος.

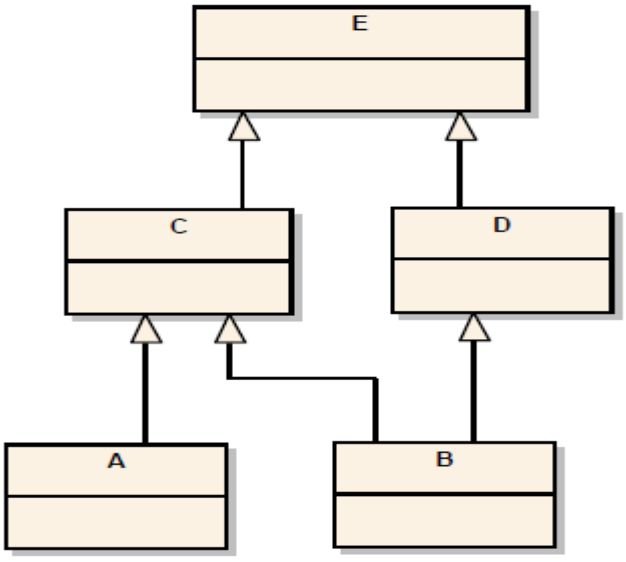
Ορισμός σύμφωνα με Chidamber-Kemerer:

Το βάθος της κληρονομικότητας μιας κλάσης είναι η τιμή της DIT της κλάσης. Σε περιπτώσεις που υπάρχουν πολλαπλές κληρονομικότητες, η τιμή της DIT είναι το μέγιστο μήκος από τον κόμβο στη ρίζα του δέντρου.

Στο [15] ο W. Li εντοπίζει δύο διφορούμενα προβλήματα αυτού του ορισμού:

1. Όταν έχουμε πολλαπλή κληρονομικότητα και πολλαπλές ρίζες συγχρόνως.
2. Υπάρχει σύγκρουση μεταξύ του ορισμού, της θεωρητικής βάσης και των απόψεων που δηλώνονται στο [16]. Και η θεωρητική βάση και οι απόψεις δείχνουν ότι η DIT πρέπει να μετρήσει τον αριθμό κλάσεων-προγόνων μιας κλάσης, αλλά ο ορισμός της DIT δηλώνει πως πρέπει να μετρηθεί το μήκος των βημάτων στην κληρονομικότητα.

Αυτή η διαφορά διευκρινίζεται στο παρακάτω παράδειγμα:



Σχήμα 7 : Παράδειγμα DIT

Σύμφωνα με τον ορισμό, η κλάση A και B έχουν το ίδιο μέγιστο μήκος από τη ρίζα του δέντρου στους κόμβους, κατά συνέπεια DIT(A) = DIT(B) = 2. Αλλά, η κλάση B κληρονομεί από περισσότερες κλάσεις από ότι η A, και σύμφωνα με τη θεωρητική βάση και τις απόψεις, οι κλάσεις A και B πρέπει να έχουν διαφορετικές τιμές DIT, DIT(A) = 2, και DIT(B) = 3 [15].

Η μελέτη για το πόσο αποδοτική είναι η τιμή της μετρικής, δεν θα εξεταστεί στα πλαίσια αυτής της εργασίας, καθώς απώτερος σκοπός μας είναι η εφαρμογή αυτών των μετρικών όπως έχουν ορισθεί ήδη. Παρόλα αυτά έχουν υπάρξει νέες προτάσεις για βελτιώσεις της συγκεκριμένης μετρικής, όπως αυτή του Wei Li που ορίζει μία εναλλακτική μετρική της DIT, τον αριθμό κλάσεων απογόνων (Number of Ancestor Classes-NAC).

3.1.5 Αριθμός απογόνων (Number of Children-NOC)

Αυτή η μετρική προτάθηκε από τους Shyam P. Chidamber και Chris F. Kemerer το 1991 [5] με στόχο να δείξει το επίπεδο επαναχρησιμοποίησης σε ένα σύστημα και σε επόμενο βήμα να δείξει το επίπεδο δοκιμής που απαιτείται. Όσο μεγαλύτερος ο αριθμός παιδιών σε μια ιεραρχία κληρονομικότητας τόσο μεγαλύτερη η επαναχρησιμοποίηση, δεδομένου πάντα, ότι η κληρονομικότητα είναι μια μορφή επαναχρησιμοποίησης. Αλλά εάν μια κλάση, ή ένα πακέτο, έχει μεγάλο αριθμό απογόνων μπορεί να αποτελεί μια περίπτωση κακής χρήσης της υπο-κατηγοριοποίησης, λόγω της πιθανότητας ανάρμοστης χρήσης αφαιρετικότητας (abstraction) της κλάσης γονέα.

Στην περίπτωση μιας κλάσης ή πακέτου, με μεγάλο αριθμό παιδιών, μπορεί να απαιτείται περαιτέρω δοκιμή των μεθόδων που χρησιμοποιούνται. Οι Chidamber και Kemerer πρότειναν ότι είναι καλύτερο να υπάρξει βάθος στην ιεραρχία κληρονομικότητας, δηλαδή υψηλό DIT και χαμηλό NOC. Ενώ οι Sheldon, Jerath και Chung [17] είχαν την άποψη:

Η αρχική υπόθεση υποστηρίζει ότι όσο βαθύτερη η ιεραρχία σε ένα δέντρο κληρονομικότητας, τόσο καλύτερη η ικανότητα επαναχρησιμοποίησης, αλλά καθίσταται δύσκολη η συντήρηση. Όσο πιο ρηχή η ιεραρχία, λιγότερη και η

αφαίρεση, αλλά είναι μεγαλύτερη η ικανότητα για κατανόηση και η δυνατότητα για τροποποίηση.

Επομένως συνιστάται, ένα βαθύ δέντρο κληρονομικότητας, να χωριστεί έτσι ώστε να δημιουργεί ένα ρηχό δέντρο. Τέλος, εξαρτάται από τον υπεύθυνο ανάπτυξης λογισμικού, να βρεθεί μια κατάλληλη τιμή κατωτάτων ορίων του τρέχοντος συστήματος, δεδομένου ότι υπάρχει εμπειρικά κάποιο ή έχει οριστεί θεωρητικά κατόπιν έρευνας.

Ορισμός:

NOC = ο αριθμός των άμεσων υποκλάσεων που υπάγονται σε μια κλάση στην ιεραρχία κλάσεων

Σύμφωνα με τον ορισμό της NOC μόνο οι άμεσες υποκλάσεις μετριούνται, αλλά μια κλάση μπορεί να επηρεάζει όλες τις υποκλάσεις της. Αυτό επισημαίνει ο W. Li [15] και προτείνει μια νέα μετρική, τον αριθμό κλάσεων απογόνων (Number of Descendent Classes -NDC) που έχει τις ίδιες ιδιότητες με την NOC, αλλά με λίγο διαφορετικό ορισμό, δηλαδή η μετρική αυτή ορίζεται ως ο συνολικός αριθμός των υποκλάσεων μια κλάσης.

3.1.6 Response For a Class-RFC

Εάν μια κλάση αποτελείται από μεγάλο αριθμό μεθόδων είναι πολύ πιθανό η πολυπλοκότητα της κλάσης αυτής να είναι υψηλή. Σε περίπτωση που είναι μεγάλος ο αριθμός μεθόδων που καλεί για να δώσει απάντηση σε μήνυμα που λαμβάνει από αντικείμενο συγκεκριμένης κλάσης, η συντήρηση και η δοκιμή του κώδικα γίνονται πολύπλοκα.

Οι Shyam P. Chidamber και Chris F. Kemerer (1991) [5] πρότειναν μία μετρική προκειμένου να μετρήσουν τον αριθμό τοπικών μεθόδων, αλλά και τον αριθμό των μεθόδων που κλήθηκαν από αυτές τις μεθόδους.

Δεν υπάρχει κάποια συγκεκριμένη τιμή κατωτάτων ορίων στη μετρική RFC. Σύμφωνα με τους Chidamber και Kemerer όσο μεγαλύτερη η τιμή της RFC, τόσο καλύτερο το επίπεδο κατανόησης που απαιτείται να υπάρχει, όσον αφορά αυτόν που ελέγχει τον κώδικα.

Ορισμός

RFC = |RS| όπου RS είναι το σετ αποκρίσεως για την κλάση, που δίνεται από το:

$$RS = \{M\} U_{all\ i} \{R_i\} \quad (3.6)$$

όπου $\{R_i\}$ = το σετ μεθόδων που καλείται απο τη μέθοδο i και

{M} = το σετ όλων των μεθόδων στην κλάση [14].

Προκειμένου να γίνει κατανοητός ο παραπάνω ορισμός παρατίθεται το παράδειγμα:

A::f1() καλεί B::f2()
A::f2() καλεί C::f1()
A::f3() καλεί A::f4()
A::f4() δεν καλεί κάποια μέθοδο
Τότε RS = {A::f1, A::f2, A::f3, A::f4} U {B::f2} U {C::f1} U {A::f4}
= {A::f1, A::f2, A::f3, A::f4, B::f2, C::f1} και RFC = 6

3.1.7 Ένωση μεταξύ αντικειμένων(Coupling Between Objects-CBO)

Μια κλάση συνδέεται με μία άλλη, εάν οι μέθοδοί της χρησιμοποιούν τις μεθόδους ή τις ιδιότητες της άλλης, ή αντίστροφα. Η ένωση μεταξύ αντικειμένων μιας κλάσης είναι ο αριθμός των άλλων κλάσεων με τις οποίες συνδέεται. Στο [18] ο P. Marinescu παραθέτει μερικές επιδράσεις υψηλής σύζευξης, στην ποιότητα που αποδίδουν σε ένα σύστημα.

- Η ικανότητα επαναχρησιμοποίησης κλάσεων ή/και υπο-συστημάτων είναι χαμηλή όταν οι συζεύξεις μεταξύ αυτών είναι υψηλές, καθώς μια γερή εξάρτηση οντότητας στο πλαίσιο όπου χρησιμοποιείται, καθιστά την οντότητα ισχυρή, ώστε να επαναχρησιμοποιηθεί σε διαφορετικό πλαίσιο.
- Κανονικά μια ενότητα (module) πρέπει να έχει χαμηλή σύζευξη στο υπόλοιπο των ενοτήτων (modules). Υψηλή σύζευξη μεταξύ των διαφορετικών μερών ενός συστήματος ασκεί αρνητική επίδραση στη διαμόρφωση του συστήματος

και είναι ένα σημάδι φτωχού σχεδιασμού, στο οποίο οι ευθύνες κάθε επιμέρους κομματιού δεν είναι σαφώς καθορισμένες.

- Η χαμηλή αυτάρκεια κλάσεων καθιστά ένα σύστημα δυσκολότερο να γίνει κατανοητό. Όταν ο έλεγχος ροής μιας κλάσης εξαρτάται από έναν μεγάλο αριθμό άλλων κλάσεων, είναι πολύ πιο δύσκολο να γίνει κατανοητή η λογική λειτουργίας της κλάσης, καθώς η κατανόησή της απαιτεί μια επαναλαμβανόμενη κατανόηση όλων των εξωτερικών κομματιών της λειτουργίας στα οποία στηρίζεται. Είναι επομένως προτιμότερο να υπάρχουν κλάσεις οι οποίες συνδέονται με μικρό αριθμό άλλων κλάσεων.

Υπάρχουν διάφοροι ορισμοί της σύζευξης, ανάλογα με το σκοπό που επιδιώκεται, π.χ. εσωτερικά δεδομένα, global δεδομένα, σύζευξη κληρονομικότητας, σύζευξη πακέτων και τα λοιπά.

Στα πλαίσια αυτής της εργασίας χρησιμοποιούμε τη βασική, ευρέως χρησιμοποιημένη μορφή της CBO που προτείνουν οι Shyam P. Chidamber και Chris F. Kemerer [5]. Σύμφωνα με αυτούς, η υπερβολική σύζευξη αποτρέπει την επαναχρησιμοποίηση, δεδομένου ότι όσο πιο ανεξάρτητη είναι μία κλάση, τόσο ευκολότερο είναι να επαναχρησιμοποιηθεί σε κάποια άλλη εφαρμογή.

Επίσης, υποστήριξαν ότι για να βελτιώσουν τη σύνθεση και να προωθήσουν την ενθυλάκωση, ο αριθμός αντικειμένων που εμπλέκονται και χρησιμοποιούνται μεταξύ δύο κλάσεων πρέπει να είναι μικρός. Κατά συνέπεια, όσο μεγαλύτερος είναι αυτός ο αριθμός, τόσο υψηλότερη η ευαισθησία σε αλλαγές στα άλλα μέρη της υλοποίησης, πράγμα που καθιστά την συντήρηση πιο δύσκολη.

Ορισμός:

CBO για μια κλάση είναι ο αριθμός των κλάσεων με τις οποίες συνδέεται.

3.2 Το πρόγραμμα ckjm

3.2.1 Γενικά στοιχεία

Στα πλαίσια ανάπτυξης λογισμικού αυτής της εργασίας χρησιμοποιείται το πρόγραμμα ckjm ως το μέσο που θα παράγει τα αποτελέσματα των μετρικών που επιθυμούμε. Το ckjm υπολογίζει τις αντικειμενοστρεφείς μετρικές Chidamber και Kemerer με την επεξεργασία του bytecode compiled Java αρχείων (.java).

Το πρόγραμμα υπολογίζει για κάθε κλάση που του δίνουμε ως είσοδο, τις ακόλουθες έξι (6) μετρικές που προτείνονται από τους Chidamber και Kemerer:

- WMC: Weighted methods per class
- DIT: Depth of Inheritance Tree
- NOC: Number of Children
- CBO: Coupling between object classes
- RFC: Response for a Class
- LCOM: Lack of cohesion in methods

Επιπλέον υπολογίζει τις μετρικές:

- Ca: Afferent couplings
- NPM: Number of public methods

οι οποίες δύο (2), απλώς αναφέρονται, καθώς δεν αποτελούν κομμάτι εφαρμογής και μελέτης στην παρούσα εργασία (παρόλα αυτά ο υπολογισμός τους πραγματοποιείται κανονικά στην εφαρμογή μας).

Το ckjm έχει γραφτεί απο τον Diomidis D. Spinellis [19] μετά από την παρατήρησή του, σχετικά με την έλλειψη αξιόπιστων προγραμμάτων για να υπολογιστούν οι αντικειμενοστρεφείς μετρικές των Chidamber και Kemerer [5].

Τα προγράμματα που υπήρχαν μέχρι τότε στο διαδίκτυο σε μορφή freeware ήταν είτε ελλιπή (υπολόγιζαν μόνο μερικές από τις μετρικές αυτές), είτε ήταν αναξιόπιστα (υπολόγιζαν λανθασμένα τις τιμές των μετρικών), είτε ήταν εξαιρετικά αργά (απαιτούσαν πολλά GB μνήμης RAM και ώρες επεξεργασίας).

Το ckjm είναι αυτοτελές (στέκεται μόνο του) και μικρό, σχεδιασμένο σύμφωνα με τη λογική των Unix και μπορεί να κάνει ένα πράγμα καλά. Δεν αναζητά καταλόγους αυτόματα ψάχνοντας αρχεία για να μετρήσει μετρικές από αυτά και δεν προσφέρει ένα graphics user interface. Παρόλα αυτά, εκτελεί τους υπολογισμούς λεπτομερώς και αποτελεσματικά όπως θα παρατηρήσουμε στην εφαρμογή που αναπτύξαμε.

3.2.2 Παράδειγμα εκτέλεσης

Για να τρέξουμε το πρόγραμμα απλά διευκρινίζουμε τα αρχεία κλάσεων (.java ή τα ζευγάρια των αρχείων jar/και κλάσεων) στη γραμμή εντολών του ή το τυπικό του input. Το πρόγραμμα θα παράξει μια γραμμή για κάθε κλάση, η οποία περιέχει το πλήρες όνομα αυτής της κλάσης και τις τιμές των μετρικών της. Αυτό το πρότυπο λειτουργίας επιτρέπει στο εργαλείο να επεκταθεί πολύ εύκολα χρησιμοποιώντας text pre- και post-processors.

Για να τρέξουμε το ckjm σε περιβάλλον windows πρέπει να τρέξουμε την java με – java flag, η οποία παρέχει ως argument το αρχείο ckjm.jar. Έπειτα, είμαστε σε θέση να διευκρινίσουμε ως arguments τα αρχεία κλάσεων java που θέλουμε να αναλύσουμε.

Παράδειγμα εκτέλεσης του ckjm μέσω γραμμής εντολών:

java -jar ckjm-1.9.jar C:/Users/Paris/Desktop/Metrics/.class > "C:\Users\Paris\Desktop\met.txt"*

Η έξοδος της εντολής θα είναι ένας κατάλογος ονομάτων κλάσεων, ακολουθούμενος από τις αντίστοιχες μετρικές για αυτή την κλάση: WMC, DIT, NOC, CBO, RFC, LCOM, Ca, και NPM.

TestBook 2 1 0 3 24 1 0 1

Book 1 1 0 0 2 0 2 0

UserInput 9 1 0 0 20 36 1 8

MyUtils 11 1 0 1 25 55 1 10

Σχήμα 8 : Παράδειγμα εκτέλεσης ckjm μέσω cmd

Όσον αφορά την εργασία μας, αυτό το πρόγραμμα έχει ενσωματωθεί στην κυρίως εφαρμογή μας, όπου παράγει τα αποτελέσματα των μετρικών όταν το ζητάμε, δίνοντας αποκλειστικά ως είσοδο φακέλους που περιέχουν αρχεία java (.java).

Τα αποτελέσματα αποθηκεύονται σε πίνακες στην database του προγράμματος και εμφανίζονται σε πίνακες μέσω ενός πιο οπτικά φιλικού interface, ορίζοντας ποιες τιμές είναι αποδεκτές σύμφωνα με το [20] όπως θα δούμε στο παρακάτω υποκεφάλαιο.

3.2.3 Χειρισμός μετρικών από το πρόγραμμα ckjm

Σύντομη παρουσίαση μετρικών και πως υπολογίζονται απο το ckjm.

WMC - Weighted methods per class

Η WMC είναι μία μετρική που υπολογίζει το σύνολο των πολυπλοκοτήτων των μεθόδων της. Σαν μέτρο πολυπλοκότητας μπορούμε να χρησιμοποιήσουμε την CC (έχει γίνει αναφορά σε προηγούμενο κεφάλαιο), ή μπορούμε να ορίσουμε αυθαίρετα τιμή πολυπλοκότητας 1 για κάθε μέθοδο. Το πρόγραμμα ckjm ορίζει τιμή πολυπλοκότητας 1 για κάθε μέθοδο και επομένως η τιμή της WMC είναι ίση με τον αριθμό μεθόδων στην κλάση.

DIT - Depth of Inheritance Tree

Αυτή η μετρική προβλέπει για κάθε κλάση μία μέτρηση επιπέδων κληρονομικότητας από την κορυφή ιεραρχίας αντικειμένου. Στην Java όπου όλες οι κλάσεις κληρονομούν την Object, η ελάχιστη αξία της DIT είναι 1.

NOC - Number of Children

Αυτή η μετρική μετρά απλά τον αριθμό των άμεσων απογόνων μιας κλάσης.

CBO - Coupling between object classes

Η μετρική αντιπροσωπεύει τον αριθμό κλάσεων που συνδέονται με μια δεδομένη κλάση (efferent συζεύξεις, CE). Αυτή η σύζευξη μπορεί να εμφανιστεί μέσω κλήσεων μεθόδου, πρόσβασης σε πεδία (fields), κληρονομικότητας, arguments, τύπων επιστροφής (return types), και εξαιρέσεων (exceptions).

RFC - Response for a Class

Η μετρική υπολογίζει τον αριθμό διαφορετικών μεθόδων που μπορούν να εκτελεσθούν όταν λαμβάνει ένα αντικείμενο της κλάσης ένα μήνυμα. Ιδανικά, θα θέλαμε να βρούμε για κάθε μέθοδο της κλάσης, τις μεθόδους που θα καλέσει η κλάση, και να επαναλαμβάνουμε αυτή τη διαδικασία για κάθε καλούμενη μέθοδο, υπολογίζοντας αυτό που ονομάζουμε «μεταβατική περάτωση της γραφικής παράστασης κλήσης της μεθόδου» (transitive closure of the method's call graph). Αυτή η διαδικασία μπορεί από τη μία να είναι ακριβής, αλλά από την άλλη αρκετά ανακριβής. Στο πρόγραμμα ckjm, υπολογίζεται μια απλή προσέγγιση στο response σετ, παρατηρώντας απλά την κλήση μεθόδων, μέσα στο σώμα των μεθόδων της κλάσης. Αυτή η απλοποίηση χρησιμοποιήθηκε επίσης το 1994 από τους Chidamber και Kemerer για την περιγραφή μετρικών.

LCOM - Lack of cohesion in methods

Μετράει τα σετ μεθόδων μέσα σε μια κλάση που δεν συσχετίζονται μέσω του διαμοιρασμού μερικών από τα πεδία της κλάσης. Ο αρχικός ορισμός αυτής της μετρικής (που είναι και αυτός που χρησιμοποιεί το ckjm) εξετάζει όλα τα ζεύγη των μεθόδων μιας κλάσης. Σε κάποια από αυτά τα ζεύγη και οι δύο μέθοδοι έχουν πρόσβαση τουλάχιστον σε ένα κοινό πεδίο της κλάσης, ενώ σε άλλα ζεύγη οι δύο μέθοδοι δεν έχουν πρόσβαση σε κανένα κοινό πεδίο. Η έλλειψη συνοχής σε αυτή την περίπτωση, υπολογίζεται αφαιρώντας από τον αριθμό ζευγών μεθόδων που δεν μοιράζονται κοινή πρόσβαση πεδίων, τον αριθμό των ζευγών που μοιράζονται.

Μεταγενέστεροι ορισμοί αυτής της μετρικής χρησιμοποιούν ως βάση μέτρησης, τον αριθμό κομματιών των γραφημάτων τμηματοποίησης των μεθόδων της κλάσης. Άλλοι τροποποίησαν τον ορισμό της συνεκτικότητας, ώστε να συμπεριλάβουν και τις κλήσεις μεταξύ των μεθόδων της κλάσης. Το πρόγραμμα ckjm ακολουθεί τον αρχικό καθορισμό (1994) από τους Chidamber και Kemerer [14].

Ca - Afferent couplings

Είναι μια μετρική που υπολογίζει πόσες άλλες κλάσεις χρησιμοποιούν τη συγκεκριμένη κλάση. Η Ca υπολογίζεται χρησιμοποιώντας τον ίδιο ορισμό με αυτόν που χρησιμοποιούμε για τον υπολογισμό της CBO (Ce).

NPM - Number of Public Methods

Αυτή η μετρική μετρά όλες τις μεθόδους σε μια κλάση που δηλώνονται ως δημόσιες (public). Μπορεί να χρησιμοποιηθεί για να μετρήσει το μέγεθος ενός API που παρέχεται από ένα πακέτο (package).

3.3 Καθορισμός αποδεκτών ορίων μετρικών

Τα όρια που προκύπτουν ως αποδεκτά για τις βασικές έξι (6) μετρικές που μελετάμε, προκύπτουν από τα συμπεράσματα που αναγράφονται στο paper των Linda H. Rosenberg [20], Ruth Stapko και Albert Gallo κατόπιν δοκιμών που έχουν γίνει τα τελευταία χρόνια πάνω σε χιλιάδες κλάσεις Java, σε projects που διεξάγονται στο Software Assurance Technology Center (SATC) του NASA Goddard Space Flight Center [20].

Σκοπός τους είναι να εντοπίσουν με όσο το δυνατόν μεγαλύτερη ακρίβεια τα πιο «προβληματικά» σημεία του κώδικα, ώστε να είναι πιθανό σε δεύτερο χρόνο να γίνουν βελτιώσεις ή και διορθώσεις για να είναι πιο αποδοτικός.

Σύμφωνα λοιπόν με αυτή την έρευνα, όταν άρχισαν να εφαρμόζονται οι μετρικές στον αντικειμενοστρεφή κώδικα, φάνηκε πως οι τιμές ήταν γενικότερα πολύ χαμηλότερες από αυτές που είμαστε εξοικειωμένοι να βλέπουμε σε λειτουργικό κώδικα.

Κρίνοντας από προγενέστερες θεσπίσεις κατωτάτων ορίων, ο αντικειμενοστρεφής κώδικας φάνηκε να είναι λιγότερο σύνθετος και πιο αρθρωτός (modular) από τον κώδικα κληρονομικοτήτων μη αντικειμενοστρέφειας. Αλλά λόγω του τελείως διαφορετικού τρόπου δόμησης ενός αντικειμενοστρεφούς συστήματος, οι χαμηλοί αριθμοί ήταν συχνά πολύ παραπλανητικοί, αγνοώντας τις αλληλεπιδράσεις μεταξύ των κλάσεων, και παραβλέποντας την πολυπλοκότητα, κυρίως λόγω χρήσης της κληρονομικότητας.

Οι ακόλουθες τιμές κατωτάτων ορίων για τις μεμονωμένες μετρικές προσδιορίστηκαν κατόπιν μελέτης των μετρικών που υπολογίστηκαν, χρησιμοποιώντας ως δείγμα τα διάφορα projects αντικειμενοστρέφειας σε Java και C++ του SATC .

- Number of methods (NOM) : ≤ 20 ιδανικό, ≤ 40 αποδεκτός ανά κλάση.
  Το εργαλείο μέτρησης συμπεριέλαβε δομητές (constructors) και αποδομητές (deconstructors) στις μετρήσεις των μεθόδων, οπότε αυτά τα κατώτατα όρια είναι σαφώς διογκωμένα. Παίρνοντας υπόψη αυτήν την παραδοχή, ο

συνιστώμενος αριθμός των πραγματικά εφαρμοζομένων μεθόδων, μεταφράζεται σε κάτω από 10 ανά κλάση.

- Weighted Methods per Class (WMC) : ≤ 25 ιδανικό, ≤ 40 αποδεκτό. Ο αριθμός μεθόδων και η πολυπλοκότητα αυτών καθορίζουν το πόσος χρόνος και προσπάθεια απαιτούνται για να αναπτύξουμε και στη συνέχεια να συντηρούμε μια κλάση. Παρά το ότι ο αριθμός των μεθόδων μπορεί να είναι διογκωμένος λόγω μέτρησης και των δομητών, η WMC μπορεί και παρέχει μια καλύτερη ρεαλιστική ιδέα της συνολικής πολυπλοκότητας μιας κλάσης.

- Response for Class (RFC) : ≤ 50.
  Γενικότερα έχουν παρατηρηθεί πολύ λίγες κλάσεις με RFC πάνω από 50. Εάν η τιμή της RFC είναι υψηλή, σημαίνει ότι η πολυπλοκότητα αυξάνεται και μειώνεται η κατανοησιμότητα. Όσο μεγαλύτερος ο αριθμός μεθόδων που μπορούν να κληθούν μέσα από μία κλάση με μηνύματα, τόσο μεγαλύτερη και η πολυπλοκότητα της κλάσης, καθιστώντας περίπλοκη την δοκιμή και την διόρθωσή της. Είναι πολύ δύσκολη η εφαρμογή αλλαγών σε μια κλάση με υψηλή RFC, λόγω της πιθανότητας για κατάρρευση του ήδη υπάρχοντος συστήματος αλληλεπιδράσεων.

- Coupling Between Objects (CBO) ≤ 5.
  Μια υψηλή τιμή CBO υποδεικνύει κλάσεις που είναι δύσκολο να γίνουν κατανοητές, να επαναχρησιμοποιηθούν ή να συντηρηθούν. Όσο μεγαλύτερη η CBO, τόσο υψηλότερη η ευαισθησία σε αλλαγές άλλων μερών του σχεδιασμού. Επομένως, η συντήρηση είναι πολύ πιο δύσκολη. Η χαμηλή σύζευξη καθιστά την κατανόηση της κλάσης ευκολότερη, λιγότερο επιρρεπή σε δημιουργία λαθών, και προωθεί την ενθυλάκωση και το modularity.

- Depth in Tree > 5.

  Σημαίνει ότι οι μετρικές για μια κλάση υποτιμούν πιθανώς την πολυπλοκότητά της. Για DIT 0 δείχνει μια «ρίζα»: Όσο υψηλότερη η τιμή της DIT, παραδείγματος χάριν 2 και 3 υποδεικνύουν έναν υψηλότερο βαθμό επαναχρησιμοποίησης. Μια πλειοψηφία ρηχών δέντρων (DIT < 2) μπορεί να αντιπροσωπεύει φτωχή εκμετάλλευση των πλεονεκτημάτων της αντικειμενοστρεφούς σχεδίασης και της κληρονομικότητας. Από την άλλη όμως, μια αφθονία βαθιάς κληρονομικότητας (DIT > 5) θα μπορούσε να αποδειχθεί υπερβολή, λαμβάνοντας τα πλεονεκτήματα της κληρονομικότητας, εις βάρος της πολυπλοκότητας. Όταν υπάρχει τέτοια ελευθερία στη χρήση κληρονομικότητας, οι προαναφερθείσες μετρικές κλάσεων υποτιμούν την πολυπλοκότητα του συστήματος.

- Number of Children (NOC)

  Όσο μεγαλύτερος ο αριθμός των παιδιών, τόσο μεγαλύτερη η πιθανότητα ανάρμοστης εφαρμογής αφαιρετικότητας του γονέα και ανάγκη για επιπρόσθετους ελέγχους, αλλά από την άλλη, τόσο μεγαλύτερη η δυνατότητα επαναχρησιμοποίησης, καθώς η κληρονομικότητα είναι μια μορφή επαναχρησιμοποίησης. Ενώ δεν υπάρχει κάποια «καλή» ή «κακή» τιμή για την NOC, η τιμή αποκτά βαρύτητα όταν μια κλάση βρίσκεται να έχει υψηλές τιμές για άλλες μετρικές. Η πολυπλοκότητα μιας κλάσης μεταφέρεται και σε όλες τις κλάσεις παιδιά και τελικά το συνολικό μας σύστημα φαίνεται να έχει μεγαλύτερη πολυπλοκότητα από ότι φάνηκε να έχει με την πρώτη ματιά.

Στην εφαρμογή που αναπτύξαμε γίνεται εφαρμογή κάποιων από τα παραπάνω (WMC,DIT,NOC,CBO,RFC) στα αποτελέσματα των μετρικών που λαμβάνονται. Όταν η τιμή του πεδίου που εμπεριέχει το αποτέλεσμα της μετρικής

βρίσκεται εκτός ορίων, το χρωματίζουμε με χρώμα «κόκκινο», ενώ αν η τιμή του είναι αποδεκτή, το χρωματίζουμε με χρώμα «πράσινο».

Σε περίπτωση που μία κλάση χαρακτηρίζεται από δύο (2) και πάνω μη αποδεκτές τιμές μετρικών, χαρακτηρίζεται ως «προβληματική» (πιθανότατα να μπορεί να βελτιωθεί ή και να διορθωθεί η υλοποίηση της, ώστε να μην έχει πιθανότητες να δημιουργήσει πρόβλημα στο project) και χρωματίζεται «κόκκινο» το πεδίο που περιέχει το όνομά της, αλλιώς η κλάση δεν αποτελεί αντικείμενο ενδιαφέροντος «προβληματικότητας» και χρωματίζεται «μπλε».

Σημείωση : Καθώς για την μετρική LCOM δεν έχει γίνει κάποια μελέτη που να ορίζει τα αποδεκτά της όρια, και δεδομένου ότι γνωρίζουμε πως για τιμή 0 προσδιορίζει μια συνεκτική κλάση, και για τιμή μεγαλύτερη του 0 υποδεικνύει γενικότερα μια κλάση που πρέπει ή θα μπορούσε να διασπαστεί σε μία ή περισσότερες κλάσεις, στην υλοποίηση της εφαρμογής θεωρούμε «προβληματική» τιμή LCOM > 2.

Επίλογος

Με το κλείσιμο αυτού του κεφαλαίου είμαστε πλέον έτοιμοι να ασχοληθούμε με την εφαρμογή όλων των εννοιών που έχουμε αναφέρει και αναλύσει έως τώρα, σε προγραμματιστικό επίπεδο.

Η εφαρμογή αναπτύσσεται στο περιβάλλον Visual Studio 2010 και γράφουμε σε γλώσσα Visual Basic .NET (VB.NET) που αποτελεί μια εξέλιξη της Visual Basic (VB).

Ακολουθεί το επόμενο κεφάλαιο που περιέχει την εφαρμογή των θεωριών που αναπτύξαμε, σε πράξη.

ΚΕΦΑΛΑΙΟ 4 : Υλοποίηση Εφαρμογής

Εισαγωγή

Σε αυτό το κεφάλαιο που αποτελεί και το τελευταίο της παρούσης εργασίας, γίνεται η παρουσίαση της υλοποίησης όλων των παραπάνω λειτουργιών που αναπτύξαμε στα προηγούμενα κεφάλαια μέσω της εξεταζόμενης εφαρμογής .

Για να γίνει πιο κατανοητή η περιγραφή, γίνεται χρήση στιγμιοτύπων από την ίδια την εφαρμογή κατά την διάρκεια εκτέλεσής της και έτσι θεωρούμε, πως το κεφάλαιο αυτό, λειτουργεί εν μέρει και ως οδηγός χρήσης λογισμικού.

Επιπλέον, γίνεται εκτενής παραπομπή στο παράρτημα της εργασίας ώστε να καλυφθεί όσο το δυνατόν καλύτερα, η ολοκληρωμένη παρουσίαση των λειτουργιών της εφαρμογής.

4.1 Interface εφαρμογής

Κάνοντας run την εφαρμογή μας, το πρώτο πράγμα που παρουσιάζεται σχετικά με το GUI είναι το prompt προς το χρήστη να κάνει login με username και password στο πρόγραμμα.

Στη συνέχεια, εφόσον γίνει επιτυχώς η εισαγωγή στο σύστημα, η κεντρική φόρμα που έχει εμφανιστεί στο παρασκήνιο (ως ανενεργή, πίσω από τη φόρμα login) γίνεται ενεργή και αποτελεί πλέον το κύριο interface αλληλεπίδρασης του χρήστη με την εφαρμογή.

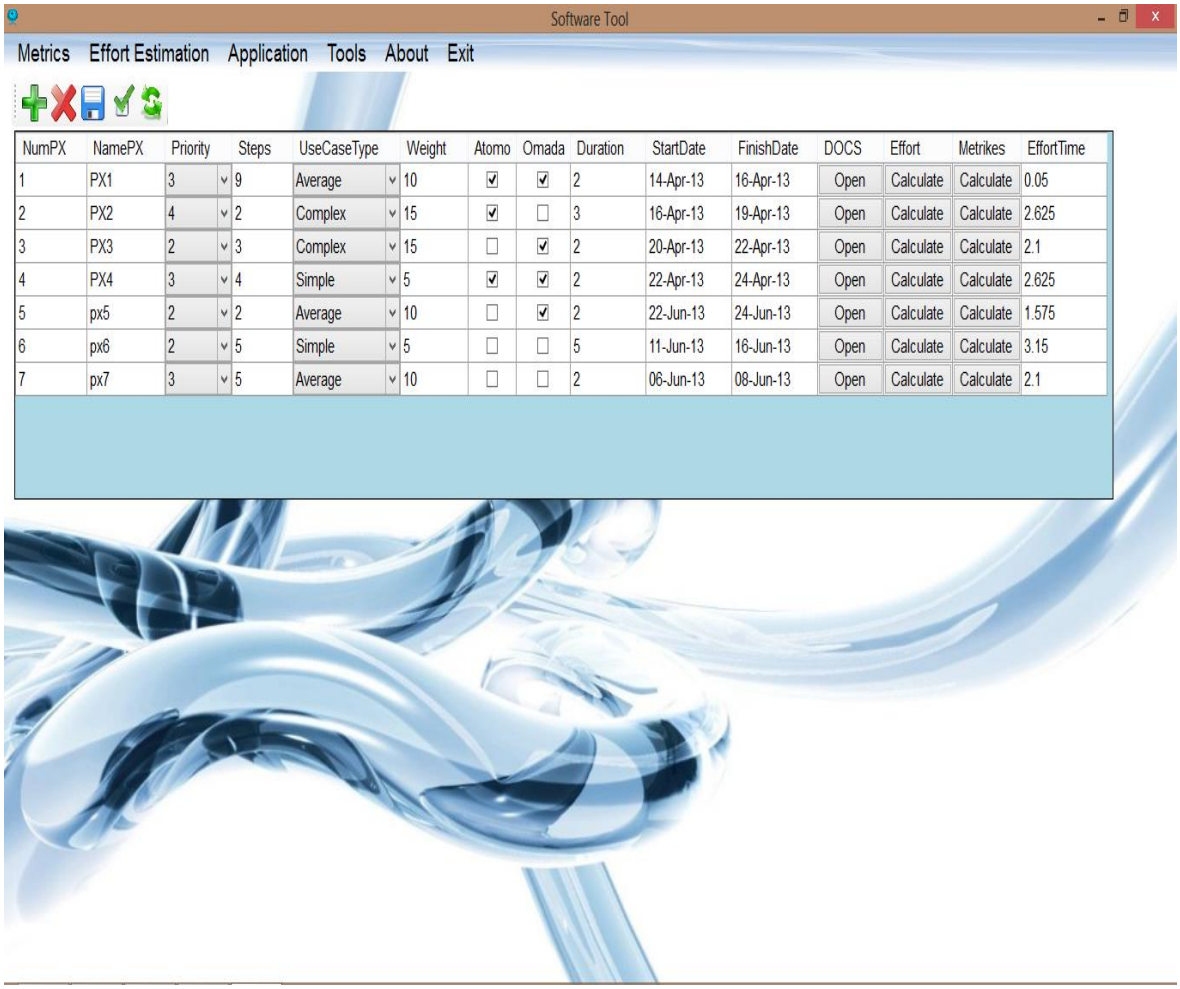
Σε αυτό το σημείο έχουμε αποκτήσει πρόσβαση στις διάφορες λειτουργίες της εφαρμογής ανάλογα με τα δικαιώματα που έχουμε ως χρήστης (για δικαιώματα γίνεται ανάλυση στο υποκεφάλαιο 4.2).

Γενικά ως (master-user) έχουμε τη δυνατότητα να εισάγουμε, να τροποποιήσουμε και να διαγράψουμε περιπτώσεις χρήσης, καθώς και να αναγράψουμε την αναλυτική της περιγραφή (σε έγγραφο κειμένου) , να υπολογίσουμε την εκτίμηση προσπάθειας για την συγκεκριμένη περίπτωση χρήσης και να υπολογίσουμε κάποιες βασικές μετρικές των κλάσεων (σε Java) που εμπλέκονται με αυτήν.

Τέλος, όλα τα αποτελέσματα που προκύπτουν από τις παραπάνω λειτουργίες, αποθηκεύονται οργανωμένα σε μια βάση δεδομένων που είναι ενσωματωμένη στην εφαρμογή. Είναι by default ορισμένο, μόνο ο master-user της εφαρμογής που είναι και ο προγραμματιστής της, να έχει πρόσβαση στους πίνακες αυτούς, ώστε σε περιπτώσεις επίβλεψης ή και διορθώσεων να μπορεί να προβεί στις απαραίτητες τροποποιήσεις.

Σημείωση : Οποιεσδήποτε αλλαγές κάνουμε στις λειτουργίες που παρέχονται για κάθε περίπτωση χρήσης, αποθηκεύονται στη βάση δεδομένων της εφαρμογής με τέτοιο τρόπο ώστε την επόμενη φορά που θα ανοίξουμε μια ήδη τροποποιημένη περίπτωση χρήσης, να έχουν κρατηθεί οι αλλαγές.

Ακολουθεί το στιγμιότυπο του Visual Studio 2010 που παρουσιάζει το κεντρικό interface της εφαρμογής:



NumPX	NamePX	Priority	Steps	UseCaseType	Weight	Atomo	Omada	Duration	StartDate	FinishDate	DOCS	Effort	Metrikes	EffortTime
1	PX1	3	9	Average	10	☑	☑	2	14-Apr-13	16-Apr-13	Open	Calculate	Calculate	0.05
2	PX2	4	2	Complex	15	☑	☐	3	16-Apr-13	19-Apr-13	Open	Calculate	Calculate	2.625
3	PX3	2	3	Complex	15	☐	☑	2	20-Apr-13	22-Apr-13	Open	Calculate	Calculate	2.1
4	PX4	3	4	Simple	5	☑	☑	2	22-Apr-13	24-Apr-13	Open	Calculate	Calculate	2.625
5	px5	2	2	Average	10	☐	☑	2	22-Jun-13	24-Jun-13	Open	Calculate	Calculate	1.575
6	px6	2	5	Simple	5	☐	☐	5	11-Jun-13	16-Jun-13	Open	Calculate	Calculate	3.15
7	px7	3	5	Average	10	☐	☐	2	06-Jun-13	08-Jun-13	Open	Calculate	Calculate	2.1

Εικόνα 1 : Κεντρικό Interface εφαρμογής

Όπως φαίνεται στην Εικόνα 1, κάθε σειρά (row) αυτού του πίνακα περιέχει μια περίπτωση χρήσης και τα βασικά χαρακτηριστικά της. Αριθμό, όνομα, βήματα, τύπο, βάρος, εμπλεκομένους, διάρκεια, ημερομηνίες εκκίνησης/λήξης, καθώς και την αναλυτική περιγραφή της, την εκτίμηση προσπάθειας και την δυνατότητα υπολογισμού μετρικών.

Σημείωση : Πατώντας το button "Open" στο row μιας περίπτωσης χρήσης, ανοίγει ένα έγγραφο κειμένου Word (.docx) το οποίο είναι ένα απλής μορφής έγγραφο καταγραφής περίπτωσης χρήσης (και αντιστοιχίζεται με την συγκεκριμένη), σύμφωνα με αυτά που αναφέρθηκαν στο υποκεφάλαιο 1.3.

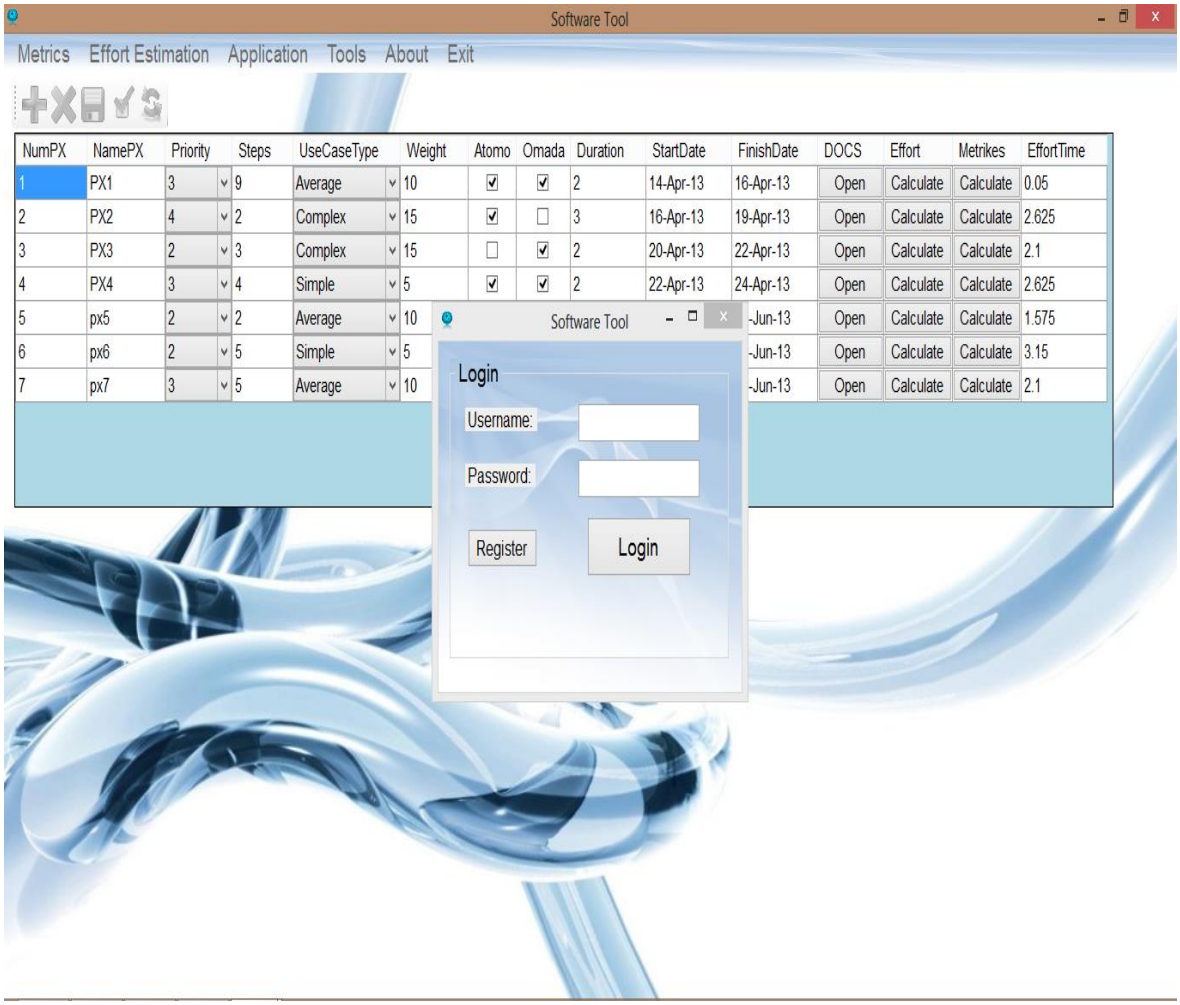
4.2 Το συστημα login

Το σύστημα login που έχει αναπτυχθεί έχει ως στόχο να παρέχει την δυνατότητα ιεραρχικής διάταξης των δικαιωμάτων διαφόρων χρηστών, αλλά και την ασφάλεια ώστε να μην μπορεί να έχει πρόσβαση σε δεδομένα του προγράμματος ο οποιοσδήποτε.

Κάθε φορά που δημιουργείται ένας χρήστης, δημιουργείται με default δικαιώματα αυτομάτως από το σύστημα, ώστε να έχει την ελάχιστη δυνατή προσβασιμότητα σε λειτουργίες της εφαρμογής (για τυπικούς λόγους ασφαλείας). Ο χρήστης δημιουργείται ενημερώνοντας κατάλληλα τα στοιχεία μιας νέας καταχώρησης στον πίνακα Users (βλέπε παράρτημα για πλήρη περιγραφή).

Το σύστημα login (εισόδου) αποτελείται από τέσσερες (4) ουσιαστικά φόρμες που η κάθε μία είναι υπεύθυνη για την υλοποίηση διαφορετικών λειτουργιών που αφορούν τους χρήστες (Users) ,αλλά και τα δικαιώματά τους στην εφαρμογή . Αυτές οι φόρμες είναι η Login ,Logout, PassChange, Registration (βλέπε παράρτημα για πλήρη παρουσίαση).

Ακολουθεί το στιγμιότυπο του Visual Studio 2010 που παρουσιάζει το Login της εφαρμογής:



Εικόνα 2 : Είσοδος στην εφαρμογή

Στην Εικόνα 2 φαίνεται το prompt της εφαρμογής για να εισάγουμε τα στοιχεία του χρήστη και εφόσον γίνουν αποδεκτά να δώσει πρόσβαση στην εφαρμογή. Επίσης παρέχεται η δυνατότητα δημιουργίας ενός νέου χρήστη μέσω του κουμπιού Register.

4.3 Υπολογισμός εκτίμησης προσπάθειας

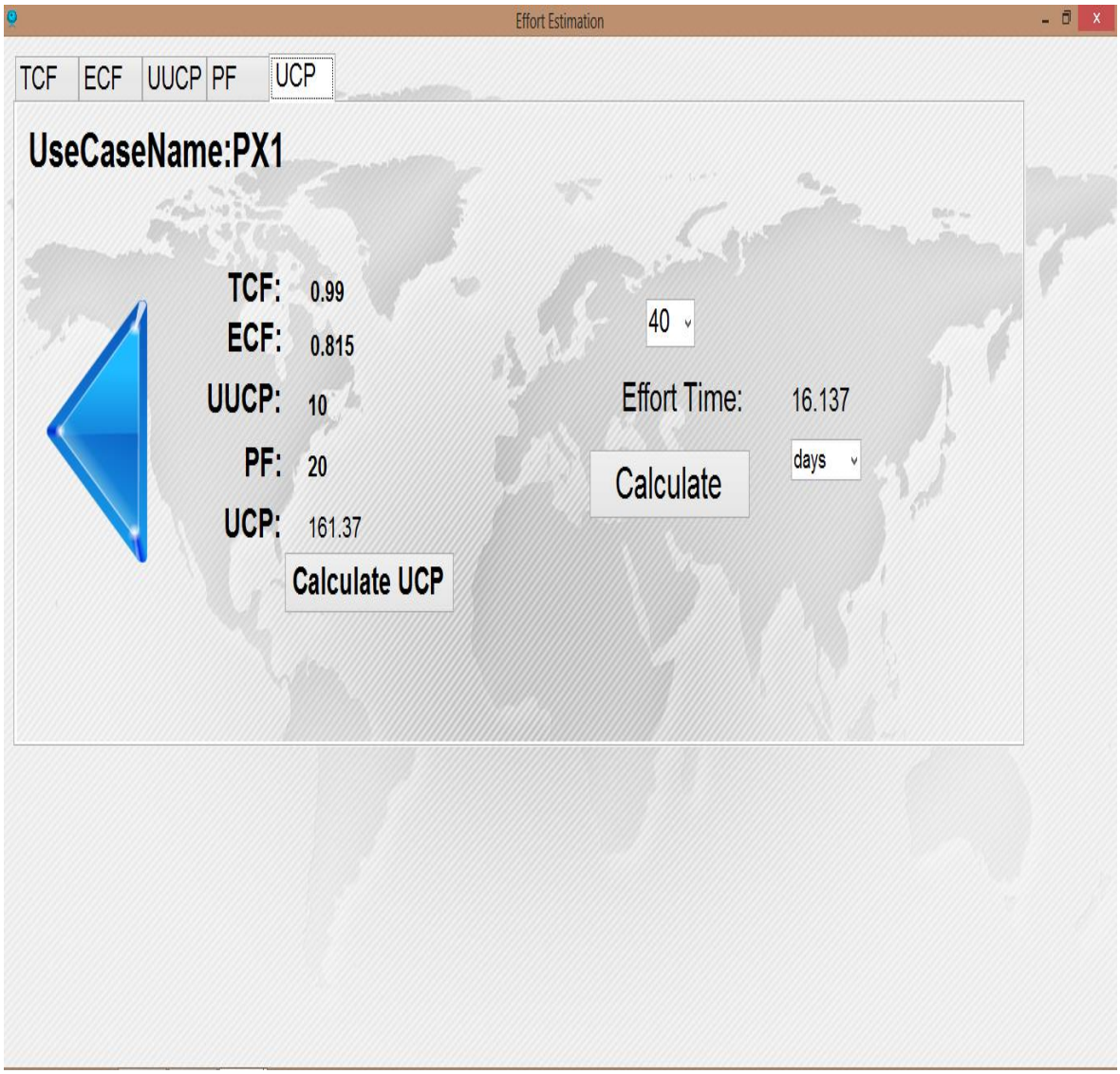
Αυτή η λειτουργία αποτελεί την μία εκ των δύο (2) βασικότερων λειτουργιών που καλούμαστε να υλοποιήσουμε στα πλαίσια αυτής της πτυχιακής και είναι η εκτίμηση προσπάθειας που απαιτείται για την ολοκλήρωση μιας μεμονωμένης περίπτωσης χρήσης.

Η διαδικασία που ακολουθούμε είναι αυτή που έχουμε περιγράψει αναλυτικά στο Κεφάλαιο 2 και η υλοποίηση της γίνεται σε μια φόρμα που την έχουμε ονομάσει Effort Estimation.

Εσωτερικά της φόρμας, στον κώδικά της, γίνεται η κωδικοποίηση των τύπων, εξισώσεων, σχέσεων και συναρτήσεων του Κεφαλαίου 2 και όλα μαζί αυτά τα στοιχεία με τη χρήση της βάσης δεδομένων (άντληση / εγγραφή στοιχείων) συνεργάζονται και αλληλεπιδρούν, ώστε να παράγουν τα επιθυμητά αποτελέσματα.

Για πλήρη παρουσίαση των πινάκων και tabs που εμπλέκονται στον υπολογισμό της εξίσωσης του Effort Time, ανατρέξτε στο παράρτημα.

Ακολουθεί το στιγμιότυπο του Visual Studio 2010 που παρουσιάζει το τελευταίο tab (UCP) της φόρμας Effort Estimation και περιέχει τα στοιχεία που προκύπτουν από τους υπολογισμούς της εκτίμησης προσπάθειας για μια περίπτωση χρήσης:



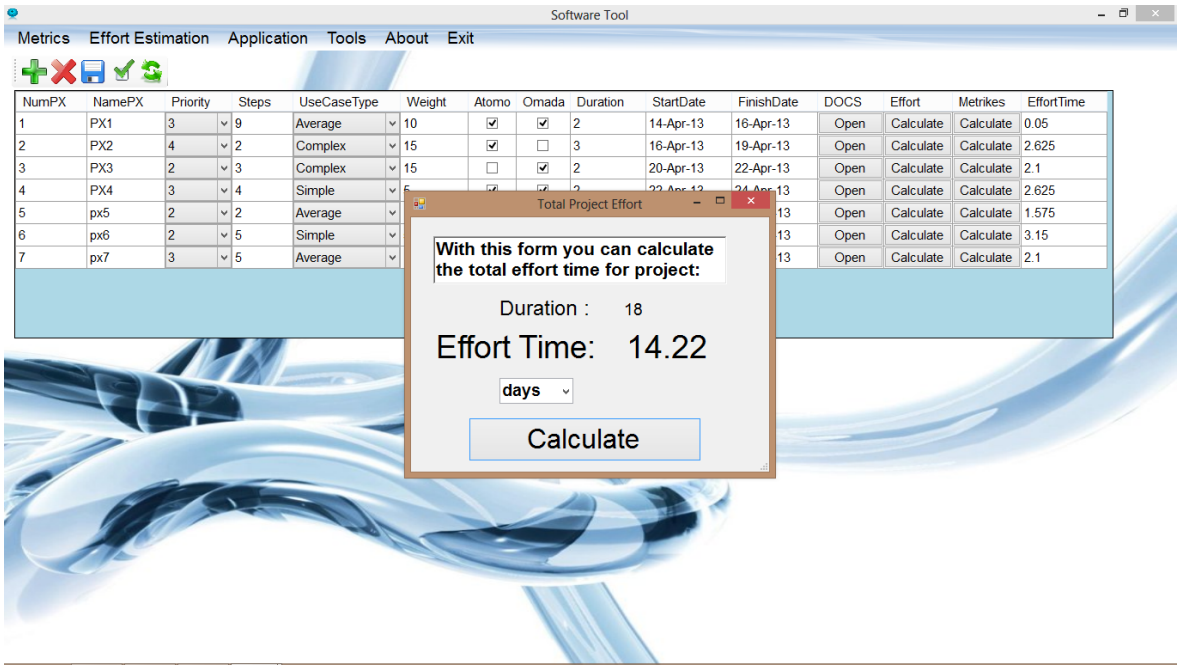
Εικόνα 3 : Φόρμα εκτίμησης προσπάθειας UCP tab

Όπως παρατηρούμε στην Εικόνα 3, στο tab UCP της φόρμας εκτίμησης προσπάθειας αναγράφεται το όνομα της περίπτωσης χρήσης που εξετάζουμε , οι τιμές των TCF, ECF, UUCP, PF, UCP και υπολογίζουμε το effort που απαιτείται, μεταφρασμένο σε ώρες, ημέρες, εβδομάδες, μήνες και χρόνια, ανάλογα με την επιλογή που έχουμε κάνει.

Το πρόγραμμα επιτρέπει να ορίσουμε το πόσες ώρες δουλεύει ο προγραμματιστής ανά εβδομάδα. Η default τιμή είναι ορισμένη σε σαράντα (40) ώρες ανά εβδομάδα.

Μπορούμε να το αλλάξουμε χειροκίνητα όμως (αλλάζοντας την τιμή στο dropbox), είτε να το ορίσει ο master-user από τα tools που είναι διαθέσιμα μόνο σε αυτόν.

Σε περίπτωση που ορίσει τιμή ο master-user από το tool, η τιμή εφαρμόζεται για όλες τις περιπτώσεις χρήσης.



Εικόνα 4 : Φόρμα συνολικής εκτίμησης προσπάθειας

Στην Εικόνα 4, παρατηρούμε την φόρμα που εμφανίζεται πατώντας από το κεντρικό μενού της εφαρμογής την επιλογή ‘Effort Estimation’. Αυτή η φόρμα μας δίνει την δυνατότητα να υπολογίσουμε το συνολικό Effort Time του project που αναπτύσσουμε.

Επίσης, επιτρέπει να μετατρέψουμε τα αποτελέσματά μας σε ημέρες, εβδομάδες, μήνες και χρόνια.

Τέλος, υπολογίζει και το συνολικό χρόνο που έχουμε ορίσει εμείς, πρόχειρα και προσεγγιστικά, για όλες τις περιπτώσεις χρήσης, ώστε να μπορούμε να έχουμε μια γενική εικόνα για την πιθανή απόκλιση στην τιμή της εκτίμησης προσπάθειας.

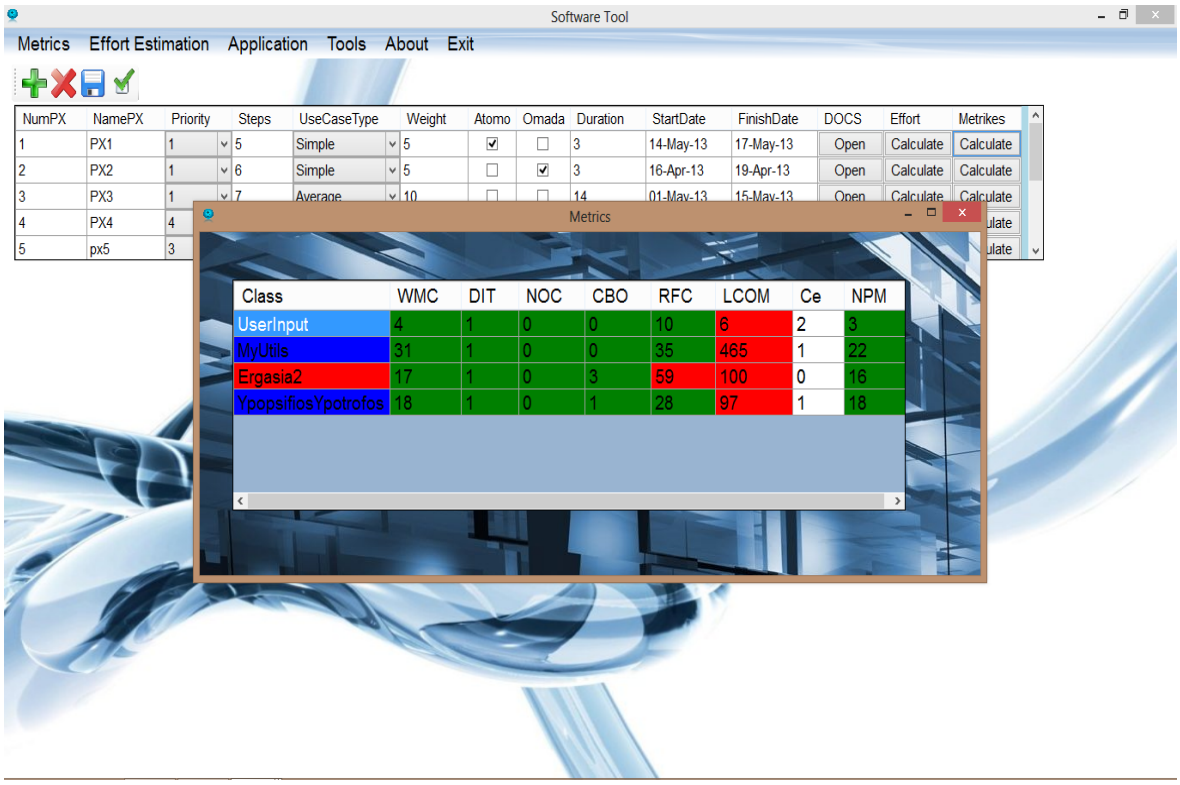
4.4 Υπολογισμός μετρικών

Αποτελεί την δεύτερη και σημαντικότερη λειτουργία που κληθήκαμε να υλοποιήσουμε στα πλαίσια συγγραφής αυτής της εφαρμογής. Οι μετρικές που εξετάζουμε είναι αυτές που μελετήσαμε στο κεφάλαιο 3, σύμφωνα με την πρώτη διατύπωσή τους από τους Chidamber και Kemerer [14].

Ως βοηθό εξαγωγής των συμπερασμάτων μας χρησιμοποιούμε το πρόγραμμα ckjm [19] (βλέπε κεφάλαιο 3) το οποίο παράγει τις τιμές μετρικών, κατόπιν εντολής που του δίνουμε να διαβάσει ένα φάκελο που περιέχει java αρχεία.

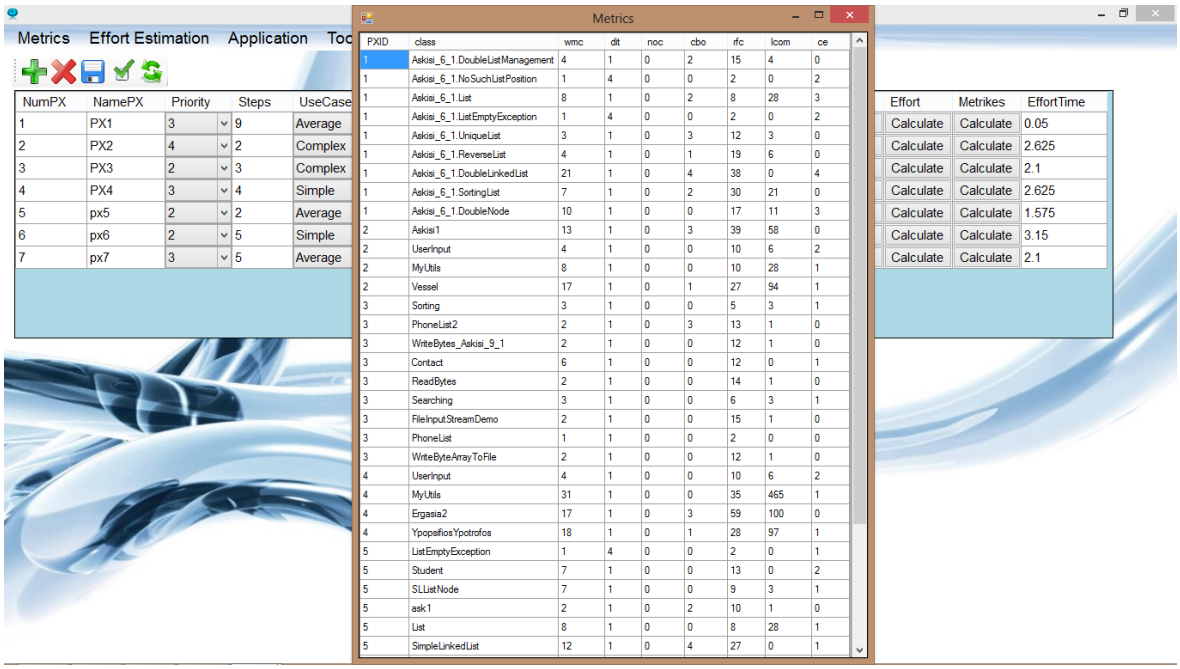
Αυτά τα αποτελέσματα τα τοποθετούμε σε ένα πίνακα όπου και αποθηκεύονται ανά περίπτωση χρήσης, και στη συνέχεια εφαρμόζουμε την εκτίμηση που έχει γίνει στο [20] σχετικά με το ποια όρια θεωρούμε αποδεκτά, ώστε να μην χαρακτηρίσουμε κάποια κλάση «προβληματική».

Ακολουθεί το στιγμιότυπο του Visual Studio 2010, που παρουσιάζει τα αποτελέσματα που εξάγουμε από τον υπολογισμό των μετρικών κλάσεων μιας περίπτωσης χρήσης:



Εικόνα 5 : Αποτέλεσμα μετρικών

Όπως φαίνεται στην Εικόνα 5, οι κλάσεις είναι «εντάξει», καθώς δεν παρουσιάζουν προβληματικές μετρικές πάνω από δύο (2) η καθεμιά, πέραν της κλάσης Ergasia2 όπου βάφεται κόκκινη καθώς είναι «προβληματική» σύμφωνα με αυτά που έχουμε αναφέρει στο κεφάλαιο 3.



Εικόνα 6 : Πίνακας όλων των μετρικών

Στην Εικόνα 6 παρατηρούμε τον πίνακα που έχουμε στη διάθεσή μας, επιλέγοντας από το κεντρικό μενού της εφαρμογής το ‘Metrics’. Αυτός ο πίνακας περιέχει όλες τις κλάσεις που έχουμε εισάγει για κάθε περίπτωση χρήσης, μαζί με τις τιμές των μετρικών τους.

Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκαν οι βασικότερες και πιο σημαντικές λειτουργίες που επιτελεί η εφαρμογή που αναπτύχθηκε.

Καθώς είναι αρκετά δύσκολο να γίνει πλήρης παρουσίαση αυτών των λειτουργιών, ο αναγνώστης καλό είναι να ελέγχει τις παραπομπές που γίνονται στο παράρτημα στο τέλος της εργασίας.

Πολύ σημαντικό ρόλο έπαιξε η σωστή οργάνωση της βάσης δεδομένων της εφαρμογής, καθώς και η ικανότητα συγγραφής κώδικα σε Visual Basic .NET. Η δομή της βάσης δεδομένων όπως και ορισμένα κομμάτια κώδικα υπάρχουν αναρτημένα στο παράρτημα της εργασίας.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Αυτή η εργασία ξεκίνησε στα πλαίσια μελέτης των εννοιών της ανάπτυξης έργων λογισμικού, εκτίμησης προσπάθειας και της εξαγωγής μετρικών από αντικειμενοστρεφή κώδικα. Αφού έγινε ενδελεχής μελέτη αυτών των εννοιών, χρησιμοποιώντας επιστημονικές αναφορές, μελέτες, δημοσιεύσεις και σχετικής βιβλιογραφίας, έγινε μια αναζήτηση για να διαπιστώσουμε εάν υπάρχει κάποιο διαθέσιμο εργαλείο που να είναι σε θέση να ενσωματώνει όλες αυτές τις έννοιες μαζί.

Καθώς τα αποτελέσματα της αναζήτησης αποδείχτηκαν αποθαρρυντικά, ο στόχος αυτής της πτυχιακής εργασίας αναμορφώθηκε στον σχεδιασμό και υλοποίηση ενός εργαλείου-εφαρμογής, που θα μπορεί να διαχειρίζεται την ανάπτυξη έργων λογισμικού.

Συχνά οι προσπάθειες ανάπτυξης λογισμικού μπορούν να αποβούν μη αποδοτικές. Και αυτό, κυρίως λόγω της ελλιπούς ανάδρασης μεταξύ αναλυτή και προγραμματιστή εφαρμογής, καθώς για την σωστή ανάπτυξη ενός ολοκληρωμένου έργου, απαιτείται μια αυτοτελής έμπειρη ομάδα ανάπτυξης λογισμικού, επαγγελματικού επιπέδου, όπου οι ρόλοι μέσα στην ομάδα είναι ξεκάθαρα ορισμένοι.

Η εφαρμογή που αναπτύχθηκε, έχει δημιουργήσει και αφήσει περιθώρια για μελλοντική προσθήκη επιπλέον λειτουργιών ή και βελτίωση των υπαρχόντων, όπως για παράδειγμα, την ενσωμάτωση διαγραμμάτων Gantt και την επέκταση της συμβατότητας με άλλες γλώσσες προγραμματισμού πέρα από τη Java, όσον αφορά το κομμάτι των μετρικών. Επιπλέον, μπορεί να γίνει και επέκταση, της διαθέσιμης λίστας μετρικών.

Το να μπορούμε να οργανώνουμε το χρόνο ανάπτυξης, βασισμένοι σε εμπειρικές μελέτες ειδικών, αλλά και σχετικών ερευνών, μπορεί να μας φανεί αρκετά ευεργετικό όσον αφορά το χρόνο που ξοδεύουμε στην ανάπτυξη, αλλά και στον τρόπο που γράφουμε τον κώδικά μας, ώστε να γίνει καλύτερος και πιο αποδοτικός.

Κλείνοντας, θεωρούμε πως το αντικείμενο μελέτης αυτής της εργασίας, ίσως είναι ένα από τα σημαντικότερα της μηχανικής λογισμικού, λαμβανομένου υπόψη ότι πλέον το λογισμικό, αναπτύσσεται με ταχύτατους ρυθμούς, σε οποιαδήποτε

προγραμματιστική γλώσσα και είναι απαραίτητα αναγκαίο να υπάρχει κάποιο εργαλείο, το οποίο θα μπορεί να επιβλέπει κάθε προσπάθειά μας για ανάπτυξη.

ΑΝΑΦΟΡΕΣ

[1] McIlroy, Malcolm Douglas (January 1969). *"Mass produced software components"*. Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968. Scientific Affairs Division

[2] Roy K. Clemmons, *Diversified Technical Services, Inc."Project Estimation With Use Case Points*"

[3] Albrecht, A.J. "*Measuring Application Development Productivity*". Proc. Of IBM Applications Development Symposium, Monterey, CA, 14-17 Oct. 1979

[4] Jacobson, I., G. Booch, and J. Rumbaugh. "*The Objectory Development Process*". Addison-Wesley, 1998.

[5] Chidamber S.R. & Kemerer, C.F., "*Towards a Metrics Suite for Object Oriented Design*" Proc. OOPSLA, 1991.

[6] McCabe, Thomas J., "A Complexity Measure*", IEEE Transactions on Software Engineering* SE-2, pp 308-320, 1976

[7] Roger S. Pressman, "*Software Engineering a Practitioner's Approach"*, Fourth Edition, McGraw-Hill, 1997

[8] Norman E. Fenton, Shari L. Pfleeger, "*Software Metrics A Rigorous & Practical Approach"*, Second Edition, PWS Publishing Company, 1997

[9] Shyam R. Chidamber, Chris F. Kemerer, "*A Metrics Suite For Object Oriented Design"*, M.I.T. Sloan School of Management, 1993

[10] David N. Card, Khaled El Emam, Betsy Scalzo, "*Measurement of Object Oriented Software Development Projects"*, Software Productivity Consortium, Herndon, Virgina, 2001.

[11] Wei Li, Sallie Henry, "*Maintenance Metrics for the Object Oriented Paradigm"*, Software Metrics Symposium, 21-22 May, pp. 52-60, 1993.

[12] Letha Etzkorn, Carl Davis, Wei Li, "*A Statistical Comparison of Various Definitions of the LCOM Metric"*, The University of Alabama, Huntsville, 1997

[13] Joshua Bloch, "*Effective Java"*, Addison-Wesley Pub. Co, First Edition, 2001.

[14] Shyam R. Chidamber, Chris F. Kemerer, "*A Metrics Suite For Object Oriented Design"*, IEEE Transactions on Software Engineering, Vol. 20, Issue 6, pp. 476-493, 1994.

[15] Wei Li, "*Another Metric Suite For Object Oriented Programming"*, The Journal of Systems and Software, Vol. 44, Issue 2, pp. 155-162, 1998.

[16] Aldo Liso, "*Software Maintainability Metrics Model: An Improvement in the Coleman-Oman Model"*, http://www.stsc.hill.af.mil/crosstalk/2001/08/liso.html, 2004-05-17.

[17] Frederick T. Sheldon, Kshamta Jerath, Hong Chung, "*Metrics for Maintainability of Class Inheritance Hierarchies"*, Journal of Software Maintenance and Evolution: Research and Practice, Vol. 14, Issue 3, pp. 147-160, 2002.

[18] Radu Marinescu, "*Measurement and Quality in Object Oriented Design"*, University of Timisoara, 2002.

[19] Diomidis D. Spinellis ,"*ckjm — Chidamber and Kemerer Java Metrics*" - http://www.spinellis.gr/sw/ckjm/

[20] Dr. Linda Rosenberg, Ruth Stapko, Al Gallo "*Applying Object Oriented Metrics*" Software Assurance Technology Center Goddard Space Flight Center – NASA http://satc.gsfc.nasa.gov

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Diomidis D. Spinellis ,"*ckjm — Chidamber and Kemerer Java Metrics*" - http://www.spinellis.gr/sw/ckjm/

[2] Evangelos Petroutsos, Mark Ridgeway "*Πλήρες εγχειρίδιο της Microsoft Visual Basic 2008*"

[3] Chidamber S.R. & Kemerer, C.F., "*Towards a Metrics Suite for Object Oriented Design*" Proc. OOPSLA, 1991

[4] Geoffrey K. Gill, Chris F. Kemerer, "*Cyclomatic Complexity Density and Software Maintenance Productivity*", IEEE Transactions on Software Engineering, 1991

[5] Karner, Gustav. "*Resource Estimation for Objectory Projects*" Objective Systems SF AB, 1993.

[6] M. Ochodek, J. Nawrocki, K.Kwarciak, "*Simplifying effort estimation based on Use Case Points*" Poznan University of Technology, Institute of Computing Science, ul. Piotrowo 2, 60-965 Poznan´, Poland

[7] Rosenberg, Linda, and Gallo, Albert, "*Implementing Metrics for Object Oriented testing*", Practical Software Measurement Symposium, 1999.

[8] Roy K. Clemmons, *Diversified Technical Services, Inc.*Project Estimation with Use Case Points"

[9] Visual Studio 2010, Microsoft msdn -http://msdn.microsoft.com/en-us/library/vstudio

ΠΑΡΑΡΤΗΜΑΤΑ

Ενδεικτικό κομμάτι κώδικα εφαρμογής

Αυτό το τμήμα κώδικα έχει ληφθεί από τη βιβλιογραφία [9] και έχει υποστεί κάποιες τροποποιήσεις ώστε να καλύψει τις ανάγκες μας στην εργασία. Πολύ συνοπτικά ορίζει μια κλάση, CalendarColumn ώστε να είμαστε σε θέση να αποθηκεύουμε ημερομηνίες ανά περίπτωση χρήσης, είτε εισάγοντας από το πληκτρολόγιο, είτε εμφανίζοντας και επιλέγοντας από ένα πινακάκι ημερομηνιών.

```
Imports System
Imports System.Windows.Forms


Public Class CalendarColumn
    Inherits DataGridViewColumn

    Public Sub New()
        MyBase.New(New CalendarCell())
    End Sub

    Public Overrides Property CellTemplate() As DataGridViewCell
        Get
            Return MyBase.CellTemplate
        End Get
        Set(ByVal value As DataGridViewCell)

            ' Ensure that the cell used for the template is a CalendarCell.
            If (value IsNot Nothing) AndAlso _
                Not value.GetType().IsAssignableFrom(GetType(CalendarCell)) _
                Then
                Throw New InvalidCastException("Must be a CalendarCell")
            End If
            MyBase.CellTemplate = value

        End Set
    End Property

End Class

Public Class CalendarCell
    Inherits DataGridViewTextBoxCell

    Public Sub New()
        ' Use the short date format.
        Me.Style.Format = "d"
    End Sub

    Public Overrides Sub InitializeEditingControl(ByVal rowIndex As Integer, _
        ByVal initialFormattedValue As Object, _
        ByVal dataGridViewCellStyle As DataGridViewCellStyle)
```

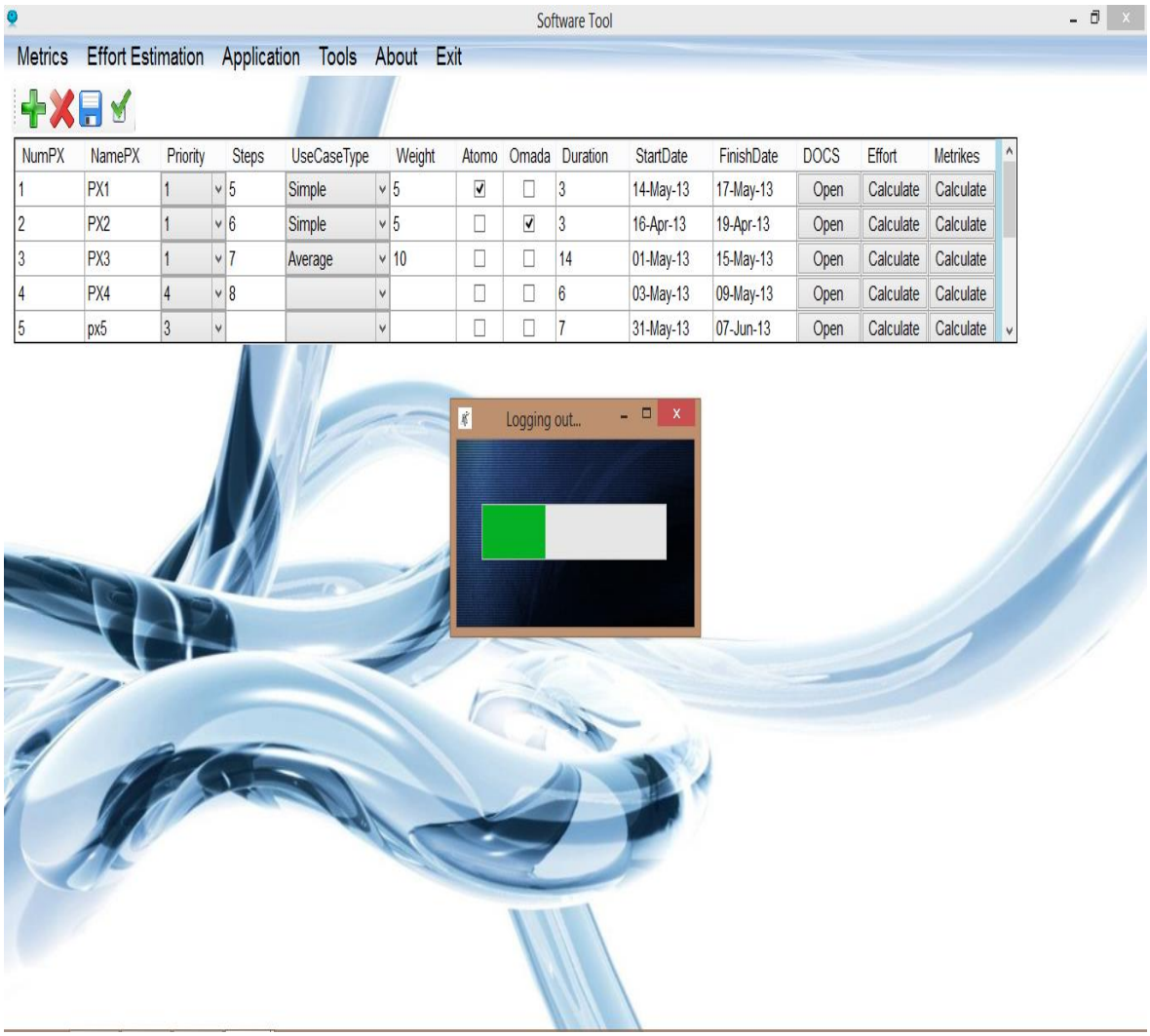
```
        ' Set the value of the editing control to the current cell value.
        MyBase.InitializeEditingControl(rowIndex, initialFormattedValue, _
            dataGridViewCellStyle)

        Dim ctl As CalendarEditingControl = _
            CType(DataGridView.EditingControl, CalendarEditingControl)

        ' Use the default row value when Value property is null.
        If (Me.Value Is Nothing) Then
            ctl.Value = CType(Me.DefaultNewRowValue, DateTime)
        Else
            'evala se comment auto to row gia na min vgazei error
            'ctl.Value = CType(Me.Value, DateTime)
        End If
    End Sub

    Public Overrides ReadOnly Property EditType() As Type
        Get
            ' Return the type of the editing control that CalendarCell uses.
            Return GetType(CalendarEditingControl)
        End Get
    End Property

    Public Overrides ReadOnly Property ValueType() As Type
        Get
            ' Return the type of the value that CalendarCell contains.
            Return GetType(DateTime)
        End Get
    End Property

    Public Overrides ReadOnly Property DefaultNewRowValue() As Object
        Get
            ' Use the current date and time as the default value.
            Return DateTime.Now
        End Get
    End Property

End Class

Class CalendarEditingControl
    Inherits DateTimePicker
    Implements IDataGridViewEditingControl

    Private dataGridViewControl As DataGridView
    Private valueIsChanged As Boolean = False
    Private rowIndexNum As Integer

    Public Sub New()
        Me.Format = DateTimePickerFormat.Short
    End Sub

    Public Property EditingControlFormattedValue() As Object _
        Implements IDataGridViewEditingControl.EditingControlFormattedValue

        Get
            Return Me.Value.ToShortDateString()
        End Get

        Set(ByVal value As Object)
            Try
                ' This will throw an exception of the string is
```

```
            ' null, empty, or not in the format of a date.
            Me.Value = DateTime.Parse(CStr(value))
        Catch
            ' In the case of an exception, just use the default
            ' value so we're not left with a null value.
            Me.Value = DateTime.Now
        End Try
    End Set

End Property

Public Function GetEditingControlFormattedValue(ByVal context _
    As DataGridViewDataErrorContexts) As Object _
    Implements IDataGridViewEditingControl.GetEditingControlFormattedValue

    Return Me.Value.ToShortDateString()

End Function

Public Sub ApplyCellStyleToEditingControl(ByVal dataGridViewCellStyle As  _
    DataGridViewCellStyle) _
    Implements IDataGridViewEditingControl.ApplyCellStyleToEditingControl

    Me.Font = dataGridViewCellStyle.Font
    Me.CalendarForeColor = dataGridViewCellStyle.ForeColor
    Me.CalendarMonthBackground = dataGridViewCellStyle.BackColor

End Sub

Public Property EditingControlRowIndex() As Integer _
    Implements IDataGridViewEditingControl.EditingControlRowIndex

    Get
        Return rowIndexNum
    End Get
    Set(ByVal value As Integer)
        rowIndexNum = value
    End Set

End Property

Public Function EditingControlWantsInputKey(ByVal key As Keys, _
    ByVal dataGridViewWantsInputKey As Boolean) As Boolean _
    Implements IDataGridViewEditingControl.EditingControlWantsInputKey

    ' Let the DateTimePicker handle the keys listed.
    Select Case key And Keys.KeyCode
        Case Keys.Left, Keys.Up, Keys.Down, Keys.Right, _
            Keys.Home, Keys.End, Keys.PageDown, Keys.PageUp

            Return True

        Case Else
            Return Not dataGridViewWantsInputKey
    End Select

End Function

Public Sub PrepareEditingControlForEdit(ByVal selectAll As Boolean) _
    Implements IDataGridViewEditingControl.PrepareEditingControlForEdit

    ' No preparation needs to be done.
```

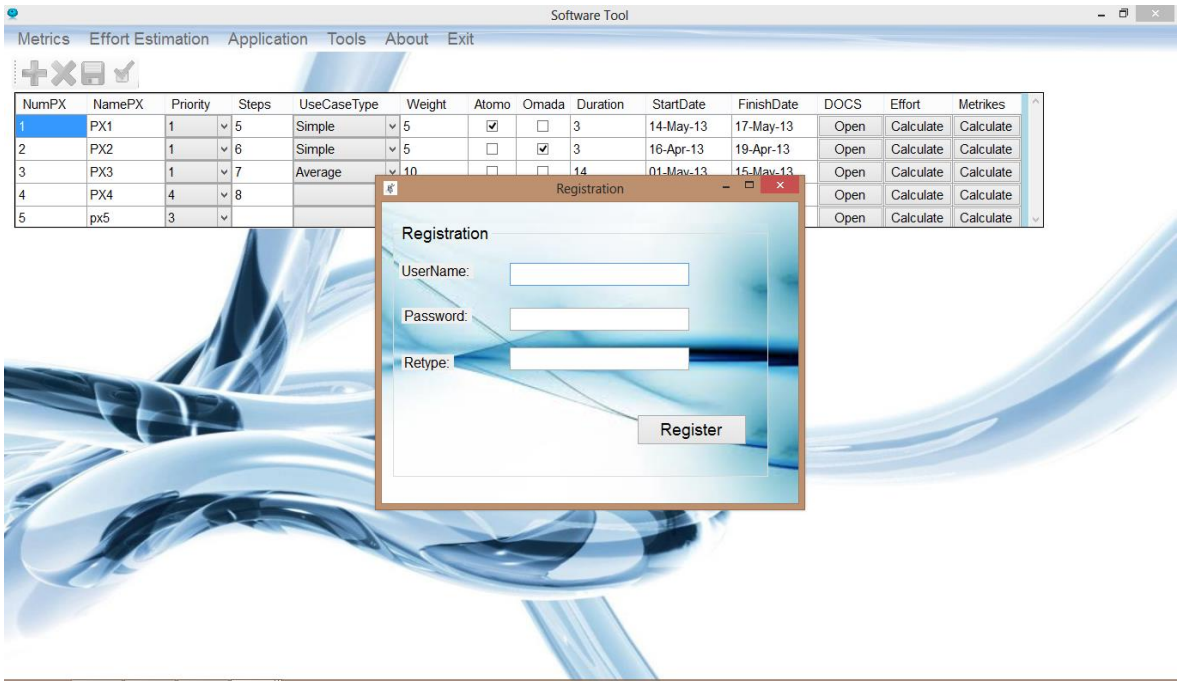
```
    End Sub

    Public ReadOnly Property RepositionEditingControlOnValueChange() _
        As Boolean Implements _
        IDataGridViewEditingControl.RepositionEditingControlOnValueChange

        Get
            Return False
        End Get

    End Property

    Public Property EditingControlDataGridView() As DataGridView _
        Implements IDataGridViewEditingControl.EditingControlDataGridView

        Get
            Return dataGridViewControl
        End Get
        Set(ByVal value As DataGridView)
            dataGridViewControl = value
        End Set

    End Property

    Public Property EditingControlValueChanged() As Boolean _
        Implements IDataGridViewEditingControl.EditingControlValueChanged

        Get
            Return valueIsChanged
        End Get
        Set(ByVal value As Boolean)
            valueIsChanged = value
        End Set

    End Property

    Public ReadOnly Property EditingControlCursor() As Cursor _
        Implements IDataGridViewEditingControl.EditingPanelCursor

        Get
            Return MyBase.Cursor
        End Get

    End Property

    Protected Overrides Sub OnValueChanged(ByVal eventargs As EventArgs)

        ' Notify the DataGridView that the contents of the cell have changed.
        valueIsChanged = True
        Me.EditingControlDataGridView.NotifyCurrentCellDirty(True)
        MyBase.OnValueChanged(eventargs)

    End Sub

End Class
```

Υπόλοιπες φόρμες συστήματος Login

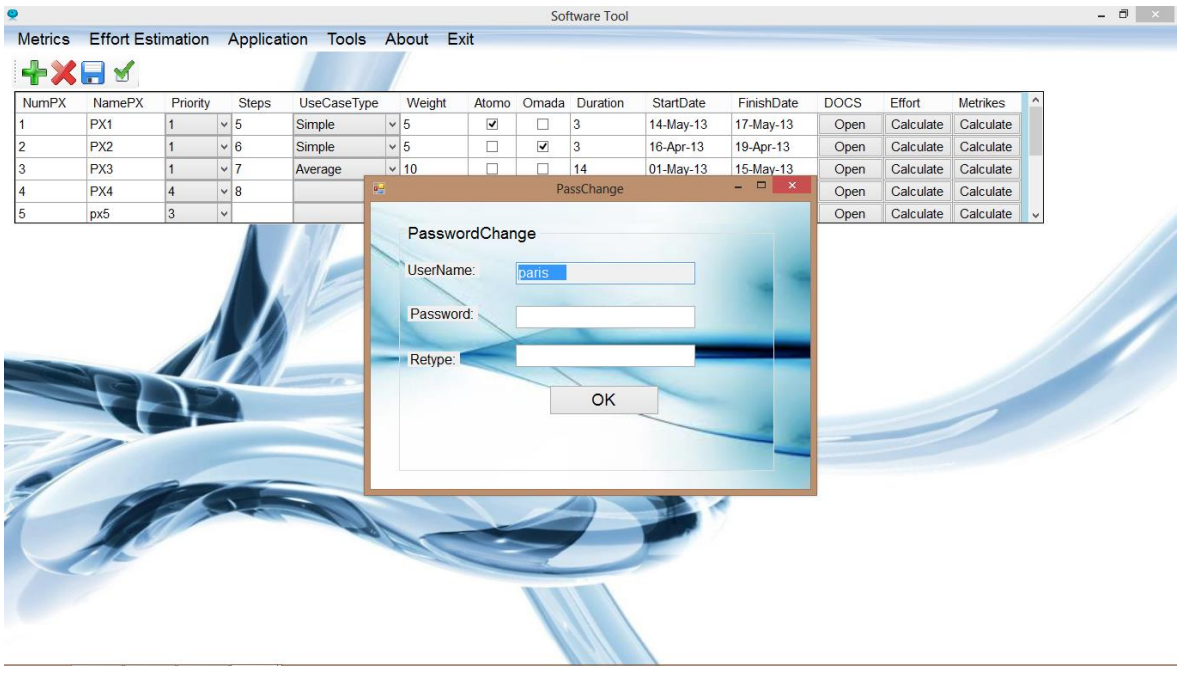
Σε αυτό το κομμάτι του παραρτήματος παρατίθενται όλες οι φόρμες που σχεδιάστηκαν και εμπλέκονται με το σύστημα login όπως παρουσιάστηκε στο κεφάλαιο 4.



Στην παραπάνω φόρμα φαίνεται η διαδικασία logout, όπου χρησιμοποιήθηκε ένας timer και ένα progress bar, για την ολοκλήρωση αποσύνδεσης από το σύστημα.



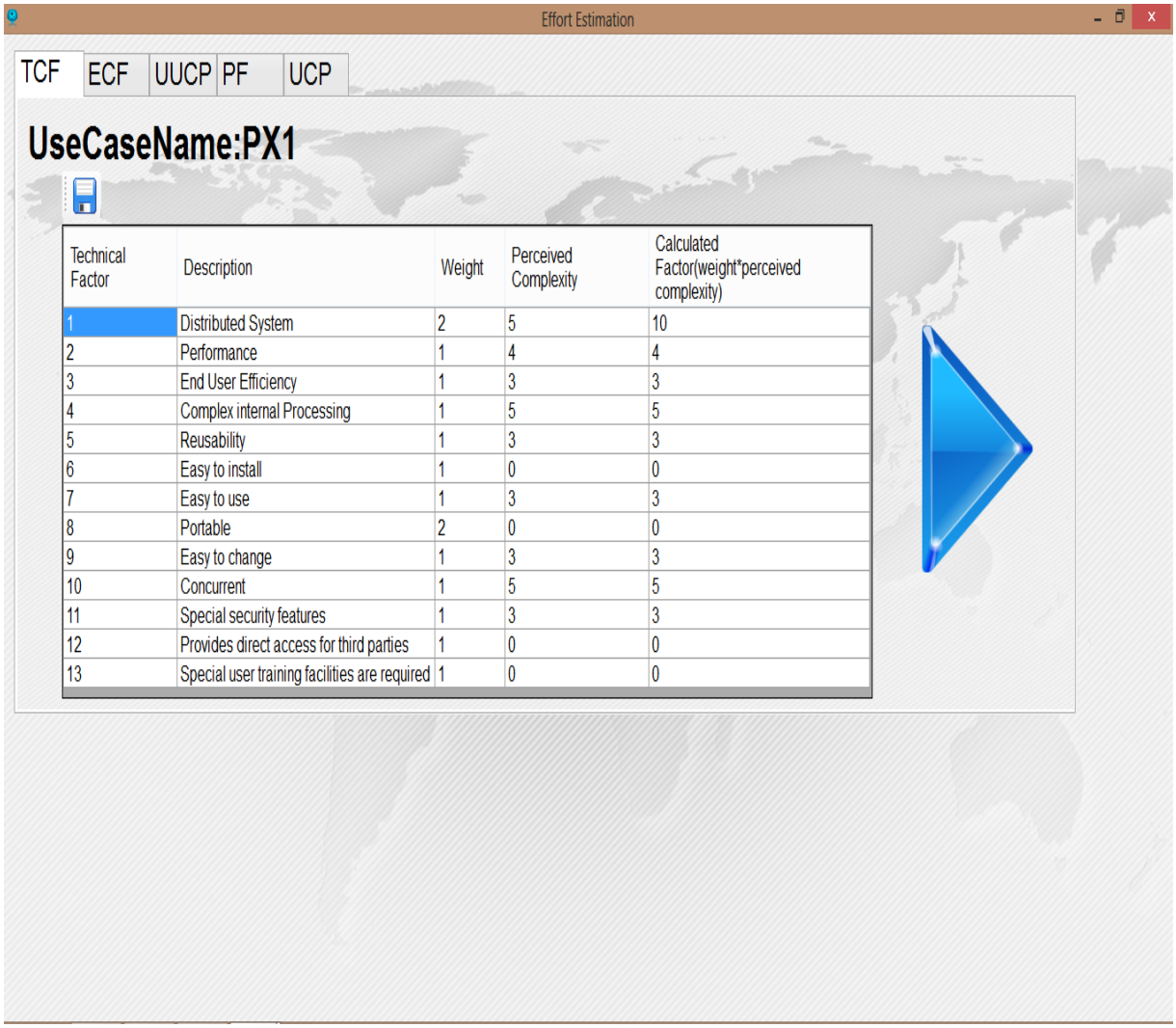
Στην παραπάνω φόρμα φαίνεται η διαδικασία του Registration, ενός χρήστη, οπού του ζητείται να εισαγάγει όνομα και κωδικό.



Εδώ παρουσιάζεται μία ακόμη λειτουργία που μας προσφέρει η εφαρμογή που σχεδιάσαμε και συγκεκριμένα μπορούμε εφόσον έχουμε εισέλθει επιτυχώς στο σύστημα, να αλλάξουμε τον κωδικό μας, εάν το επιθυμούμε.

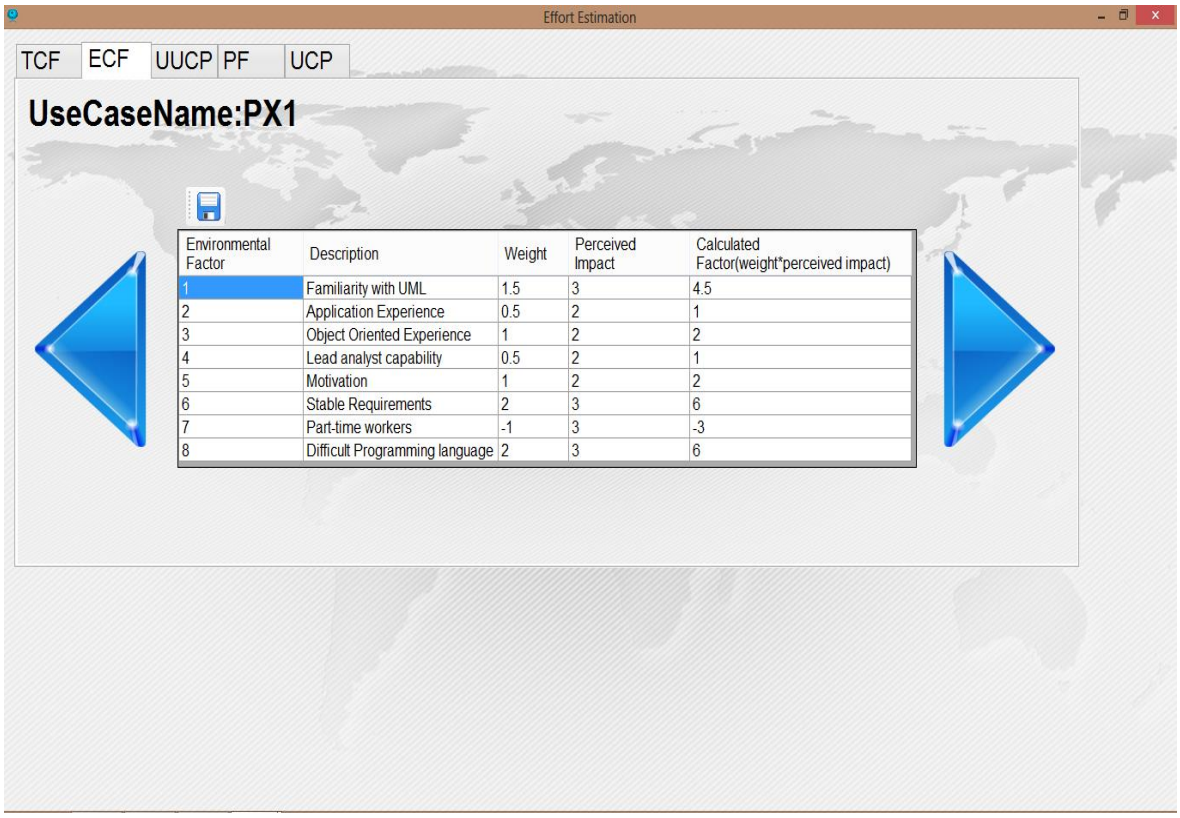
Υπόλοιπα tabs λειτουργίας εκτίμησης προσπάθειας

Στο κεφάλαιο 4 παρουσιάσαμε το στιγμιότυπο της φόρμας, που εμφανίζει τα αποτελέσματα που προκύπτουν από την λειτουργία της εκτίμησης προσπάθειας. Σε αυτό το κομμάτι του παραρτήματος δείχνουμε ένα-ένα τα tabs που επιτελούν τους επιμέρους υπολογισμούς για να καταλήξουμε στην τελική εκτίμηση.



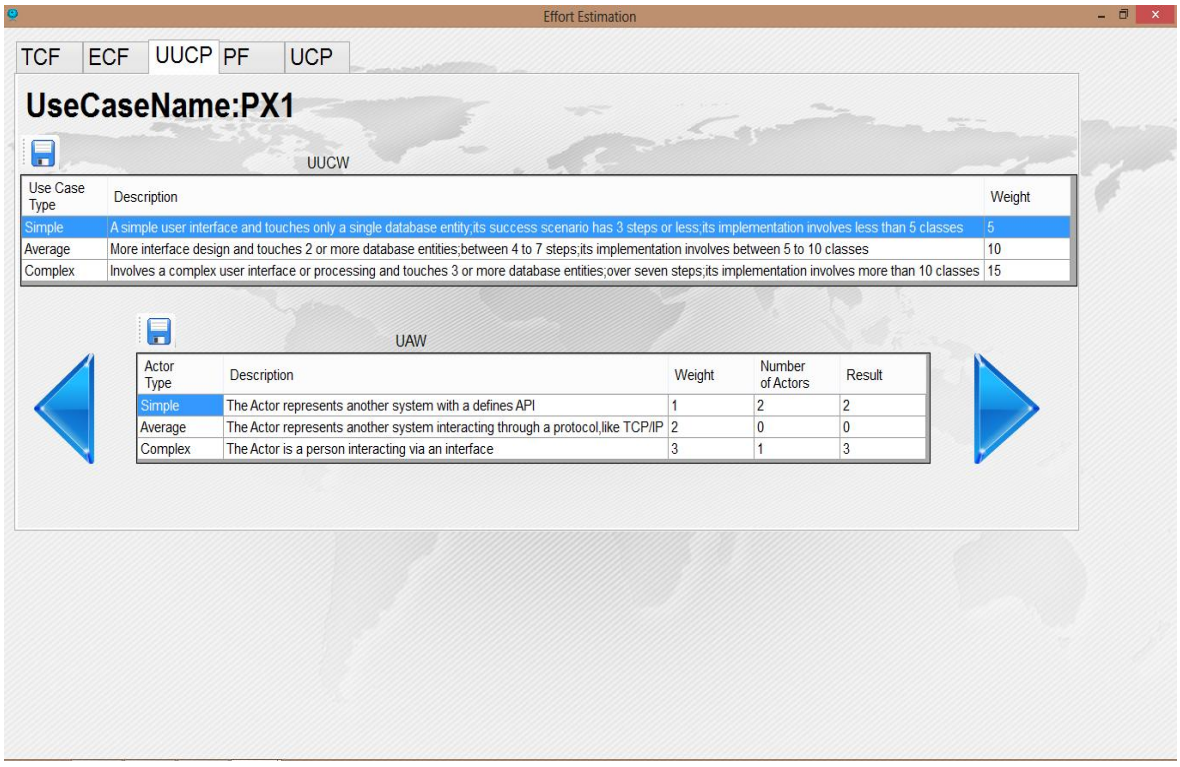
Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor(weight*perceived complexity)
1	Distributed System	2	5	10
2	Performance	1	4	4
3	End User Efficiency	1	3	3
4	Complex internal Processing	1	5	5
5	Reusability	1	3	3
6	Easy to install	1	0	0
7	Easy to use	1	3	3
8	Portable	2	0	0
9	Easy to change	1	3	3
10	Concurrent	1	5	5
11	Special security features	1	3	3
12	Provides direct access for third parties	1	0	0
13	Special user training facilities are required	1	0	0

Στο παραπάνω tab φαίνεται η διαδικασία υπολογισμού του TCF εφαρμόζοντας τις θεωρίες που παρουσιάσαμε κατά την συγγραφή της εργασίας, στην πράξη.



Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor(weight*perceived impact)
1	Familiarity with UML	1.5	3	4.5
2	Application Experience	0.5	2	1
3	Object Oriented Experience	1	2	2
4	Lead analyst capability	0.5	2	1
5	Motivation	1	2	2
6	Stable Requirements	2	3	6
7	Part-time workers	-1	3	-3
8	Difficult Programming language	2	3	6

Παρόμοια με το προηγούμενο tab εδώ υπολογίζουμε την τιμή του ECF, εισάγοντας έναν-έναν τους εξωγενείς παράγοντες.

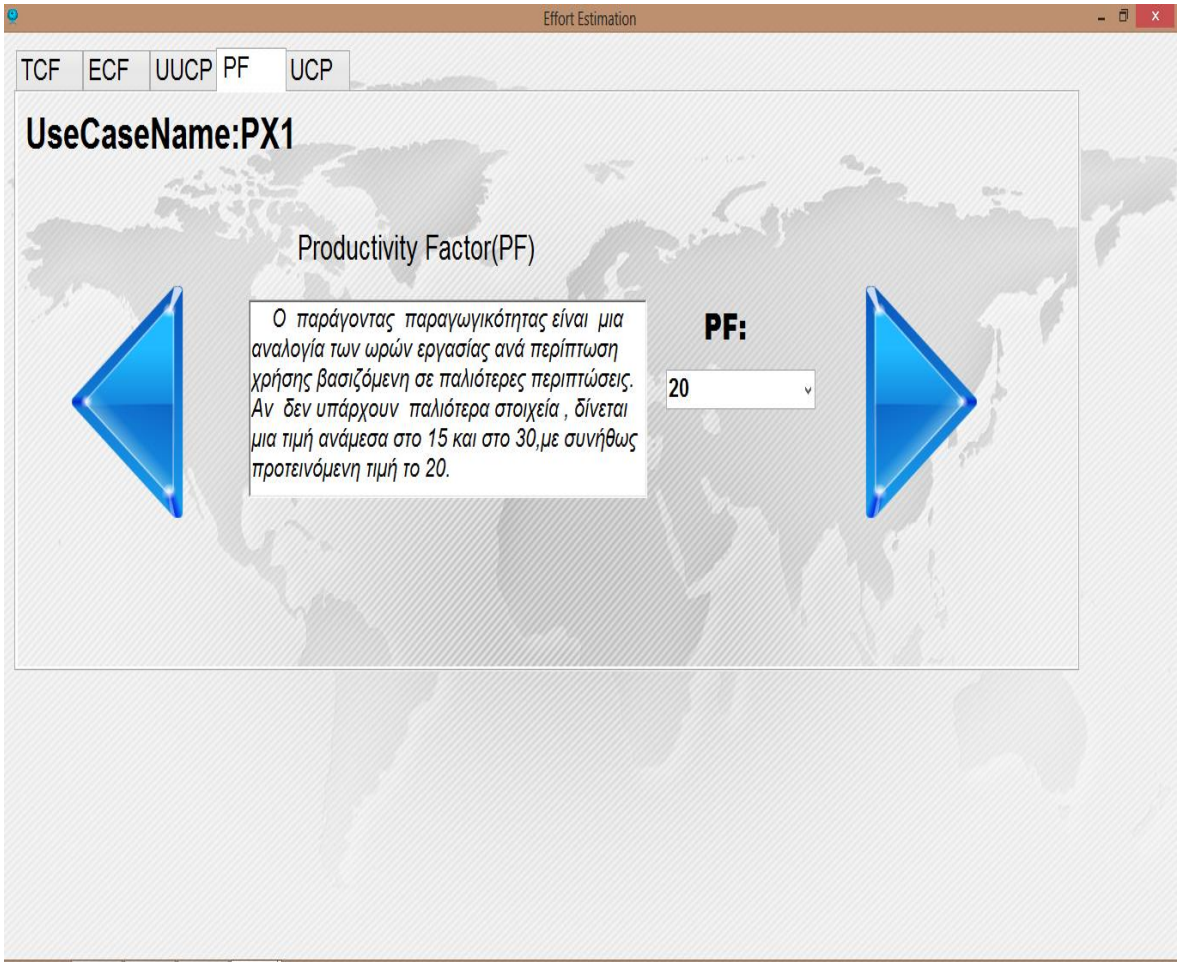


Use Case Type	Description	Weight
Simple	A simple user interface and touches only a single database entity;its success scenario has 3 steps or less;its implementation involves less than 5 classes	5
Average	More interface design and touches 2 or more database entities;between 4 to 7 steps;its implementation involves between 5 to 10 classes	10
Complex	Involves a complex user interface or processing and touches 3 or more database entities;over seven steps;its implementation involves more than 10 classes	15

UAW

Actor Type	Description	Weight	Number of Actors	Result
Simple	The Actor represents another system with a defines API	1	2	2
Average	The Actor represents another system interacting through a protocol,like TCP/IP	2	0	0
Complex	The Actor is a person interacting via an interface	3	1	3

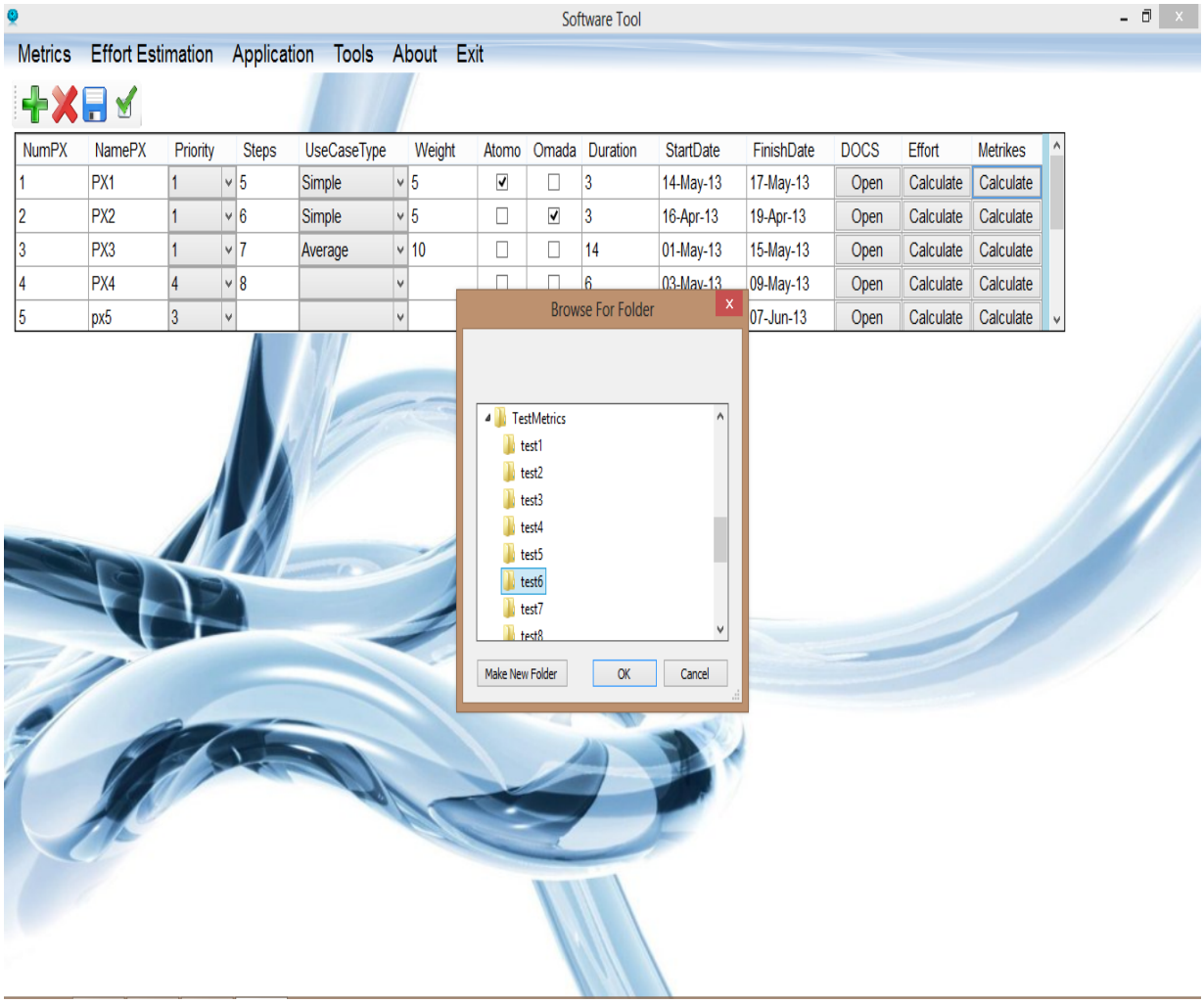
Παρόμοια με το προηγούμενο tab, υπολογίζουμε την τιμή του UUCP.



Σε αυτό το tab δίνεται μια πολύ σύντομη επεξήγηση του PF και επιλέγουμε την τιμή του από το dropbox.

Το prompt προς χρήστη να επιλέξει αρχεία εξαγωγής μετρικών

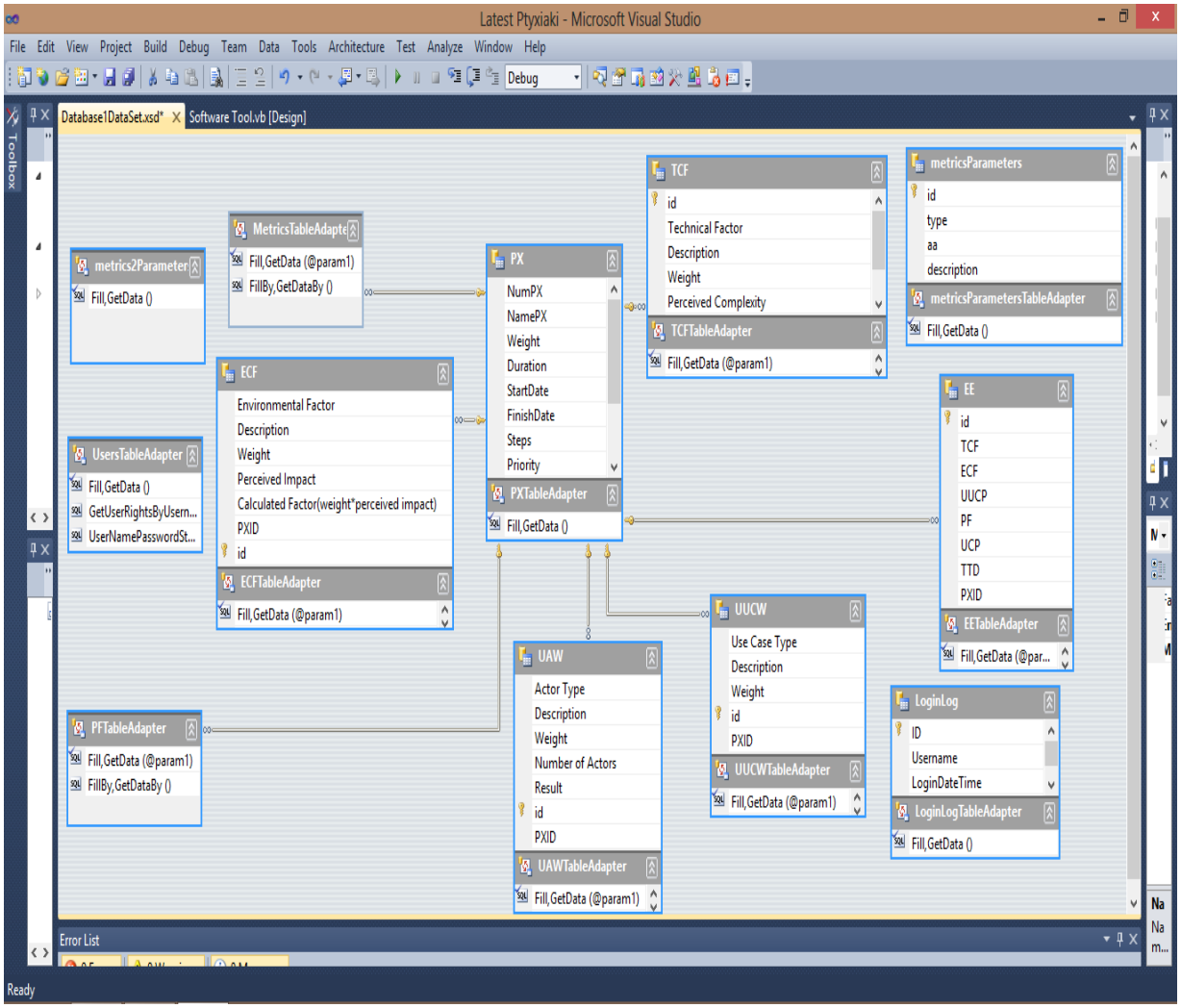
Στο κεφάλαιο 4 παρουσιάστηκε η φόρμα που περιέχει την εξαγωγή των αποτελεσμάτων των μετρικών για κάποια περίπτωση χρήσης. Σε αυτό το κομμάτι του παραρτήματος δείχνουμε ένα βήμα πίσω αυτή τη διαδικασία, δηλαδή το πως επιλέγουμε από ποια αρχεία θέλουμε να εξάγουμε τις μετρήσεις.



Στο παραπάνω σχήμα φαίνεται η προτροπή του συστήματος προς το χρήστη να επιλέξει τον φάκελο που περιέχει τα αρχεία (Java) κλάσεων που πρόκειται να εξετάσουμε.

Η δομή της βάσης δεδομένων της εφαρμογής, όπως την παρουσιάζει ο DataSet Designer

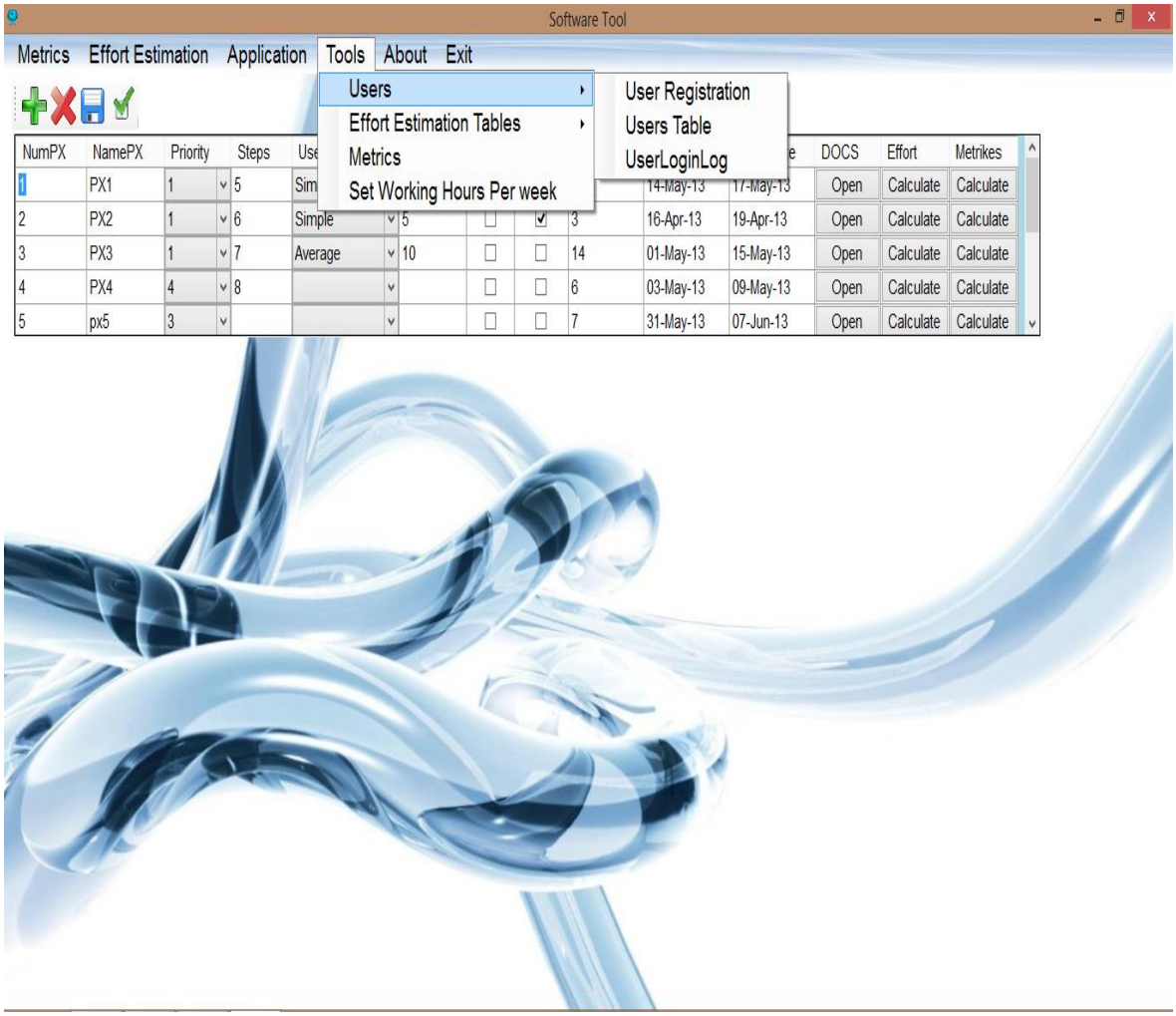
Όπως έχουμε αναφέρει αρκετές φορές κατά τη διάρκεια του κυρίως σώματος αυτής της εργασίας, πολύ σημαντικό ρόλο έπαιξε για την ολοκλήρωση αυτής της εφαρμογής, η δημιουργία μιας βάσης δεδομένων που θα διαχειρίζεται όλα αυτά τα δεδομένα και υπολογισμούς του συστήματος.



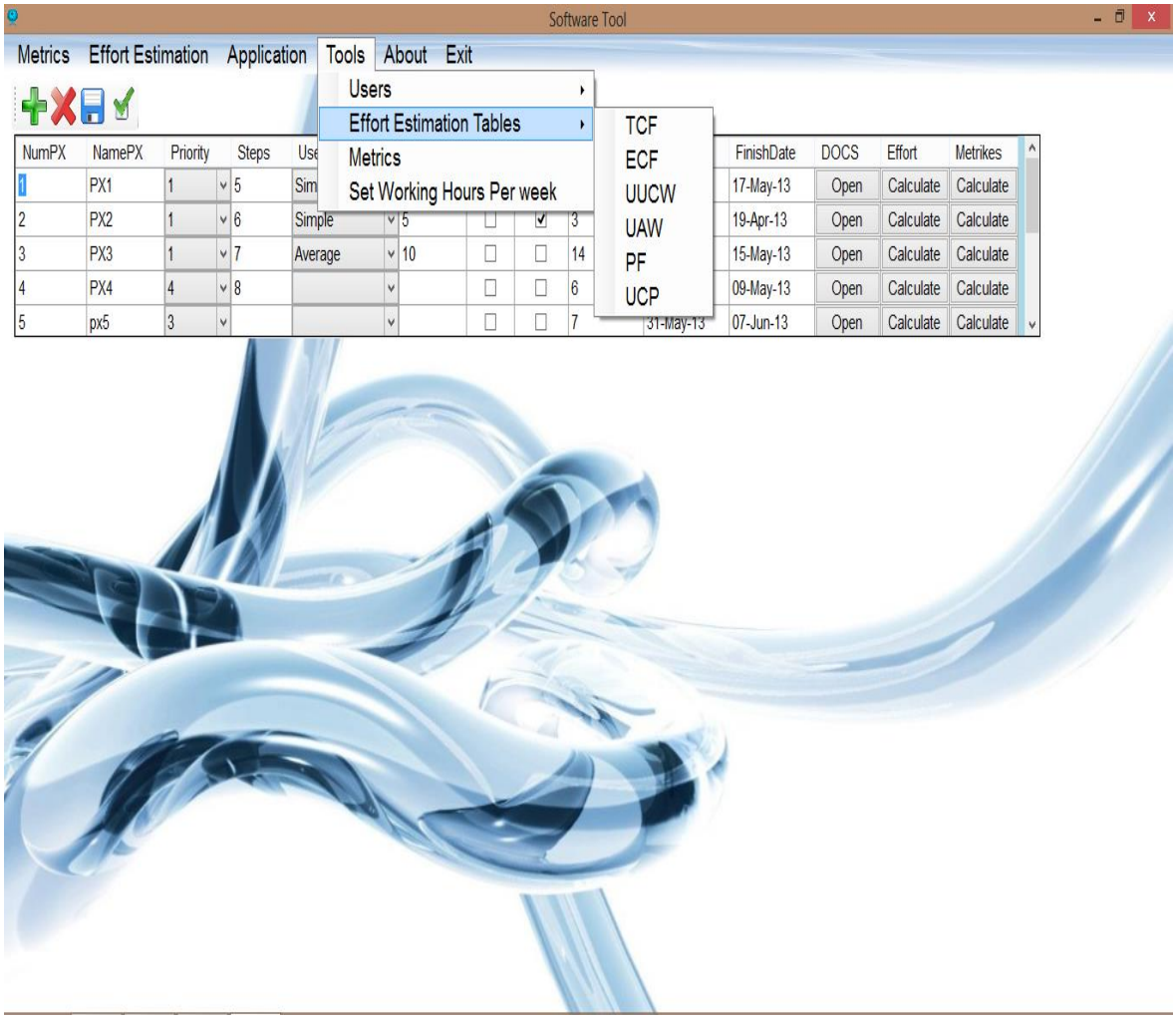
Στο παραπάνω σχήμα παρουσιάζεται το στιγμιότυπο του dataset designer του Visual Studio, όπου παρατηρούμε τους πίνακες που έπρεπε να δημιουργήσουμε καθώς και τις συσχετίσεις μεταξύ τους, ώστε να μπορούμε να διαχειριστούμε τα δεδομένα της εφαρμογής.

Τα tools που έχει πρόσβαση ο user

Εδώ παρουσιάζουμε τα εργαλεία στα οποία έχει πρόσβαση ο master-user (default), ώστε να έχει πλήρη έλεγχο σε όλες τις λειτουργίες της εφαρμογής ακόμη και σε χρόνο run-time. Όπως μπορούμε να παρατηρήσουμε στα δύο (2) επόμενα σχήματα, παρέχονται εργαλεία σχετικά με τους Users της εφαρμογής, την εκτίμηση προσπάθειας και τις μετρικές.



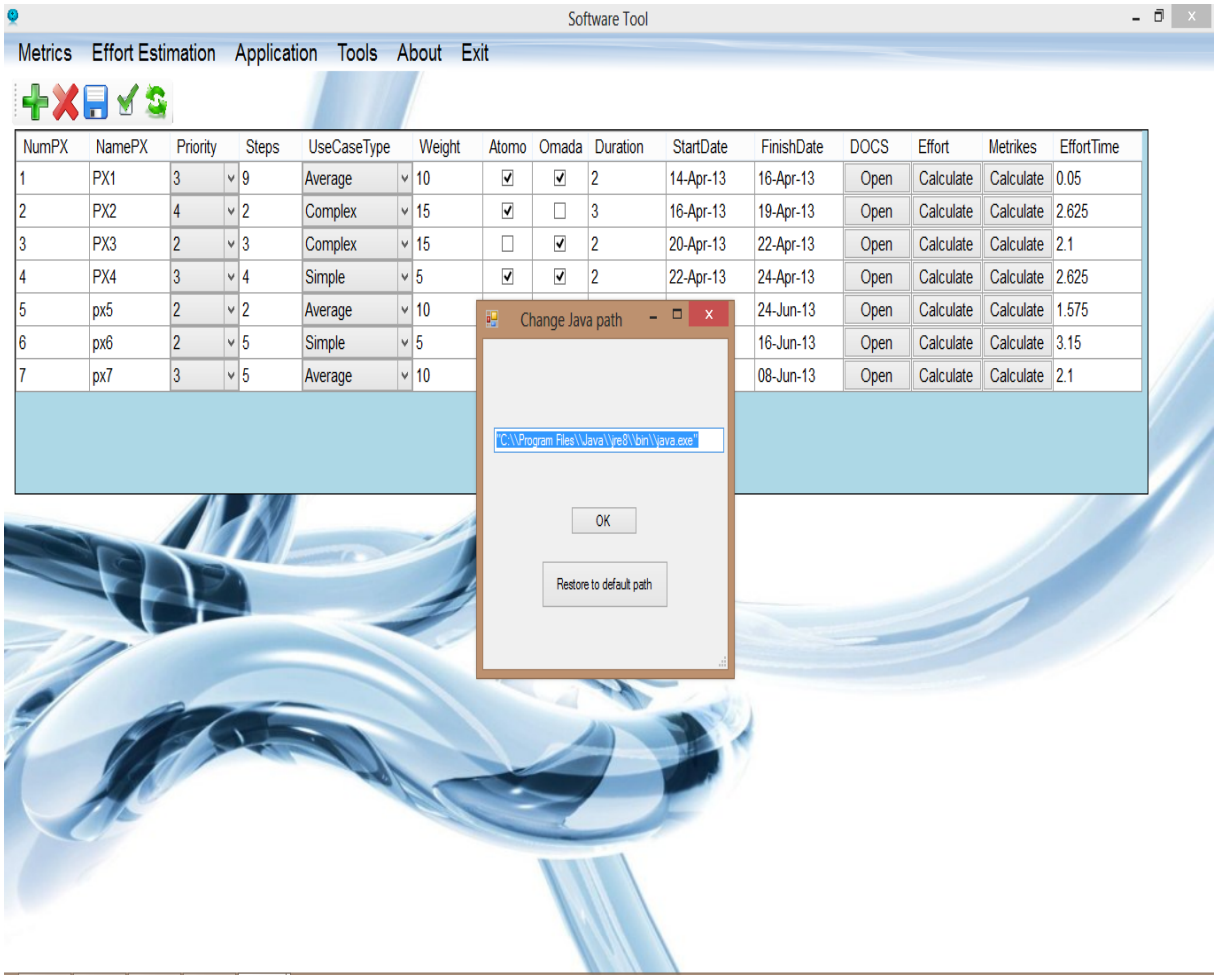
Στο παραπάνω στιγμιότυπο βλέπουμε πως μας δίνεται η δυνατότητα, να εισάγουμε έναν νέο χρήστη (on run-time), έχουμε πλήρη πρόσβαση στο πίνακα των χρηστών (μπορούμε να δούμε όνομα, κωδικό και δικαιώματα), αλλά και μπορούμε να εμφανίσουμε τον πίνακα LoginLog όπου αποθηκεύονται ποιος χρήστης εισήλθε στο σύστημα, πότε συνδέθηκε και πότε αποσυνδέθηκε.



Στο παραπάνω στιγμιότυπο παρουσιάζεται ο τρόπος που μπορούμε να δούμε τα στοιχεία που εμπεριέχονται στους βασικότερους πίνακες που εμπλέκονται στη λειτουργία των σημαντικότερων λειτουργιών της εφαρμογής.

Πολύ απλά με την επιλογή, παραδείγματος χάριν TCF ανοίγει μία φόρμα που στεγάζει αναλυτικά όλα τα στοιχεία του πίνακα TCF. Αντίστοιχα συμβαίνει για όλες τις άλλες επιλογές καθώς και για την επιλογή Metrics.

Όσον αφορά την επιλογή Set Working Hours Per week αφορά αυτά που αναλύσαμε στο υποκεφάλιο 4.3.



Στο τελευταίο στιγμιότυπο παρουσιάζεται η φόρμα από την οποία μπορούμε να αλλάξουμε το path της java στον υπολογιστή μας (εφόσον η εφαρμογή δεν την εντοπίζει), προκειμένου να μπορεί να τρέξει το πρόγραμμα ckjm για τον υπολογισμό μετρικών. Η πρόσβαση στη φόρμα γίνεται μέσω του κεντρικού μενού, Application→Change Java Path.