ΔΙΕΡΓΑΣΙΑ ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗΣ ΣΥΣΤΑΤΙΚΩΝ ΛΟΓΙΣΜΙΚΟΥ ΑΝΟΙΚΤΟΥ ΚΩΔΙΚΑ

του

ΚΡΗΤΙΚΟΥ ΑΠΟΣΤΟΛΟΥ

Μεταπτυχιακή Διατριβή Ειδίκευσης
που εκπονήθηκε για λογαριασμό του Τμήματος Πληροφορικής
στα πλαίσια των απαιτήσεων για την απόκτηση

Μεταπτυχιακού Διπλώματος Ειδίκευσης
στα «Πληροφοριακά Συστήματα»

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ, ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Φεβρουάριος, 2010

ΕΥΧΑΡΙΣΤΙΕΣ

«Ο κατασκευαστής δεν μπορεί να είναι ούτε ο μόνος,
ούτε ο καλύτερος κριτής του έργου του»

~ Αριστοτέλης

Με την παρούσα εργασία, την μεταπτυχιακή μου διατριβή, κλείνει ακόμη ένα κεφάλαιο για να επιστρέψω στον καθορισμό νέων απαιτήσεων, όπως προτείνει το μοντέλο ανάπτυξης λογισμικού με τη μέθοδο του καταρράκτη[1]. Πριν όμως συμβεί αυτό θα ήθελα να αφιερώσω μερικές γραμμές σε όσους βοήθησαν ώστε η προσπάθεια που αποτυπώνεται στην παρούσα εργασία να εξελιχθεί ομαλά και τελικά να ακολουθήσει τον, κατά το δυνατό, βέλτιστο δρόμο.

Πρώτα από όλα, θα ήθελα να ευχαριστήσω τα μέλη της τριμελούς εξεταστικής επιτροπής μου, τους κυρίους Αγγελή Ελευθέριο και Κατσαρό Παναγιώτη για την άψογη συνεργασία, τις γνώσεις και τα κίνητρα για έρευνα που μου μετέδωσαν καθ' όλη τη διάρκεια του προγράμματος μεταπτυχιακών σπουδών στο Τμήμα Πληροφορικής. Ευχαριστώ ιδιαίτερα τον κ. Σταμέλο Ιωάννη, επιβλέποντα καθηγητή της μεταπτυχιακής μου διατριβής, για την εξαιρετική καθοδήγηση και τη συνεχή και ουσιώδη επίβλεψη του έργου μου. Κυρίως όμως τον ευχαριστώ για την εμπιστοσύνη του στο πρόσωπο μου και την υποστήριξη των στόχων μου για περαιτέρω δραστηριοποίηση στον τομέα της έρευνας.

[1] http://en.wikipedia.org/wiki/Waterfall_model

Θερμές ευχαριστίες στον κ. Κακαρόντζα Γεώργιο για την πολύτιμη βοήθειά του καθ' όλη τη διάρκεια εκπόνησης της μεταπτυχιακής μου διατριβής. Η εξαιρετική γνώση του πεδίου έρευνας σε συνδυασμό με την άριστη κατάρτισή του σε θέματα ανάπτυξης λογισμικού τον καθιστούν πολύτιμο συνεργάτη. Χωρίς την ουσιαστική του παρέμβαση, η παρούσα εργασία θα υστερούσε σαφώς, ποιοτικά. Τον ευχαριστώ επίσης για την πολύτιμη συνεισφορά του στα πλαίσια προετοιμασίας της ερευνητικής εργασίας «*A semi-automated process model for Open Source code reuse*». Η καθοδήγησή του σε συνδυασμό με αυτήν του κ. Σταμέλου κατέστησε το περιεχόμενο της παρούσας διατριβής δημοσιεύσιμο μεταδίδοντάς μου ταυτόχρονα εμπειρία στη συγγραφή ερευνητικών άρθρων.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου που, όπως πάντα, αγκάλιασε την προσπάθειά μου, με στήριξε και πάνω από όλα, ανέχτηκε για ενάμιση περίπου χρόνο την παρουσία της *μυστήριας, στωική φιγούρας πίσω από την οθόνη*, εντός του σπιτιού.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ & ΠΙΝΑΚΩΝ

ΕΙΚΟΝΕΣ

ΠΙΝΑΚΕΣ

ΠΕΡΙΛΗΨΗ

Έχει καταστεί σαφές πως το Ελεύθερο Λογισμικό / Λογισμικό Ανοικτού Κώδικα (ΕΛ/ΛΑΚ) ή Free Libre / Open Source Software (FLOSS)[2] όπως συναντάται στην διεθνή βιβλιογραφία, έχει εμφανίσει, τα τελευταία χρόνια, μια θεαματική και συνεχώς ανοδική πορεία. Ως εκ τούτου πλειάδα γραμμών πηγαίου κώδικα διατέθηκαν και διατίθενται καθημερινά, ελεύθερα, στο Διαδίκτυο. Δεν είναι μάλιστα λίγες οι περιπτώσεις όπου ο πηγαίος αυτός κώδικας είναι προσεκτικά σχεδιασμένος, υλοποιημένος, έχει υποστεί ελέγχους για σφάλματα και, ως εκ τούτου, προσφέρεται για επαναχρησιμοποίηση. Τελευταία, ο ανοικτός κώδικας έχει αρχίσει να έχει έντονη απήχηση σε μικρομεσαίες εταιρίες ανάπτυξης κώδικα (Small and Medium Enterprises - SMEs), οι οποίες τον επαναχρησιμοποιούν για να επιτύχουν ταχύτερη ανάπτυξη και να μειώσο ν κατά το δυνατό το ν χρό ν ελέγχου το υ κώδικά το ς. Η ύπαρξη των αποθετηρίων έργων ΕΛ/ΛΑΚ (source code forges), όπως SourceForge[3], Google Code[4], Koders[5], κλπ., που αποτελούν ουσιαστικά δεξαμενές κώδικα ανοικτού λογισμικού, διευκολύνει τη διαδικασία αναζήτησης επαναχρησιμοποιήσιμου κώδικα παρέχοντας μηχανισμούς αναζήτησης με βάση την γλώσσα προγραμματισμού, την αδειοδότηση, κ.ο.κ.

Η παρούσα Μεταπτυχιακή Διατριβή στοχεύει σε μια αναλυτική καταγραφή της διαδικασίας της επαναχρησιμοποίησης συστατικών ανοικτού κώδικα (open source software components) που υπάρχουν διαθέσιμα στο διαδίκτυο. Αρχικά γίνεται μία

[2] Το FLOSS συναντάται επίσης με τις ονομασίες: *Free / Open Source Software (F/OSS), FOSS*
[3] http://www.sourgeforge.net
[4] http://http://code.google.com
[5] http://www.koders.com/

επισκόπηση του χώρου όπου αφενός καλύπτονται σφαιρικά οι βασικές έννοιες τις οποίες ο αναγνώστης συναντά στη συνέχεια της διατριβής και αφετέρου γίνεται αναφορά σε μια σειρά προβληματισμών που αποτέλεσαν εφαλτήριο για την διεξαγωγή του παρόντος ερευνητικού έργου. Ακολουθεί μια σειρά περιπτώσεων χρήσης που τεκμαίρει την αποτελεσματικότητα της επαναχρησιμοποίησης κώδικα ως κομμάτι της γενικότερης διαδικασίας ανάπτυξης έργων λογισμικού. Στη συνέχεια όλη η πρότερη γνώση οργανώνεται σε ένα ημι-αυτόματο μοντέλο διαδικασίας επαναχρησιμοποίησης ανοικτού κώδικα και προτείνεται (υπό μορφή σχεδίου) μιας διαδικτυακή υπηρεσία που θα υλοποιεί τη λειτουργικότητα του μοντέλου αυτού. Η διατριβή ολοκληρώνεται με αναφορές σε σχετικό έργο, συμπεράσματα και σκέψεις για μελλοντικό ερευνητικό έργο.

Η μελέτη που διεξήχθη κατά την εκπόνηση της παρούσας μεταπτυχιακής διατριβής κατέληξε στην συγγραφή επιστημονικής εργασίας με τίτλο *«A semi-automated process model for Open Source code reuse»* και εστάλη για δημοσίευση στην συνάντηση εργασίας με τίτλο *«Emerging Trends in FLOSS Research and Development»* που έλαβε χώρα στα πλαίσια του συνεδρίου International Conference on Software Engineeging 2010 (ICSE 2010).

Κεφάλαιο 1

ΕΙΣΑΓΩΓΗ

Η επαναχρησιμοποίηση κώδικα δεν είναι καινούριο φαινόμενο. Όποιος ασχολείται με τον προγραμματισμό, επαγγελματικά ή ερασιτεχνικά, γρήγορα αντιλαμβάνεται ότι ορισμένα κομμάτια κώδικα υλοποιούν κλασικές απαιτήσεις που συναντώνται σε πλειάδα έργων λογισμικού, εμπορικά και μη. Σύνδεση και διαχείριση μιας βάσης δεδομένων, ανάγνωση από ή εγγραφή σε ένα αρχείο συγκεκριμένου τύπου (ASCII, XML, JSON, κλπ.), είσοδος, έξοδος ή εγγραφή ενός επισκέπτη σε μια διαδικτυακή υπηρεσία είναι μερικά τυπικά συστατικά λογισμικού (software components) που μια εταιρία ή κάποιος ιδιώτης που ασχολείται με την ανάπτυξη λογισμικού μπορεί να κληθεί να αντιμετωπίσει.

Πολύ συχνή είναι επίσης η περίπτωση όπου κώδικας που έχει αναπτυχθεί για τις ανάγκες ενός έργου μπορεί να αποτελέσει βάση, ή να επαναχρησιμοποιηθεί αυτούσιος για την υλοποίηση μίας ή περισσότερων απαιτήσεων κάποιου μελλοντικού έργου λογισμικού. Αυτό το φαινόμενο, είναι αρκετά σύνηθες σε εταιρίες ανάπτυξης λογισμικού που δραστηριοποιούνται σε συγκεκριμένο (ή συγκεκριμένα πεδία) και αναφέρεται ως κληρονομημένος κώδικας (legacy code) [Etzkorn97].

Με την ραγδαία εξέλιξη και υιοθέτηση του Ελεύθερου Λογισμικού / Λογισμικού Ανοικτού Κώδικα (ΕΛ/ΛΑΚ) έγινε γνωστό και το μοντέλο συνεργατικής ανάπτυξης λογισμικού. Ο πυρήνας μιας εφαρμογής υλοποιείται συνήθως από έναν προγραμματιστή (ή μια μικρή ομάδα προγραμματιστών) και στη συνέχεια διατίθεται ελεύθερα στο

διαδίκτυο. Στη συνέχεια, ένας ή περισσότεροι προγραμματιστές που, είτε βρίσκουν την εφαρμογή ενδιαφέρουσα είτε χρειάζονται μέρος ή το σύνολό της για κάποια άλλη εφαρμογή που αναπτύσσουν οι ίδιοι, την μεταφορτώνουν, την επεκτείνουν με αλλαγές ή διορθώσεις και στη συνέχεια επιστρέφουν το αποτέλεσμα ως παρακαταθήκη στο αρχικό έργο. Αυτή η νέα «κουλτούρα» ανάπτυξης λογισμικού οδήγησε στην δημοσίευση χιλιάδων γραμμών κώδικα στο διαδίκτυο μετατρέποντάς το, θα μπορούσαμε να πούμε, σε μία μεγάλη δεξαμενή επαναχρησιμοποιήσιμου κώδικα.

1.1 Επαναχρησιμοποίηση κώδικα

Ο όρος επαναχρησιμοποίηση κώδικα συναντάται στη βιβλιογραφία με πολλές παραλλαγές. Ας δούμε μερικές[6].

Επαναχρησιμοποίηση Κώδικα είναι η λογική ότι το σύνολο ή ένα μέρος ενός ολοκληρωμένου έργου λογισμικού που έχει υλοποιηθεί κάποια στιγμή, μπορεί, θα έπρεπε ή χρησιμοποιείται σε μια άλλη εφαρμογή που υλοποιήθηκε χρονικά αργότερα.

~Wikipedia

Επαναχρησιμοποιήσιμος κώδικας είναι ο κώδικας που μπορεί να χρησιμοποιηθεί, χωρίς αλλαγές, για να επιτελέσει συγκεκριμένο έργο, ανεξάρτητα από την εφαρμογή που χρησιμοποιεί τον κώδικα.

~MSDN Library

[6] Το πρωτότυπο, αγγλικό κείμενο παρατίθεται αυτούσιο στο Παράρτημα Α

Επαναχρησιμοποίηση κώδικα είναι η χρήση υπάρχοντος κεφαλαίου γνώσης που αφορά το λογισμικό ή υπαρχόντων αρχετύπων για την κατασκευή νέων αρχετύπων

~W. B. Frakes & C. J. Fox

Θα πρέπει σε αυτό το σημείο να γίνει ξεκάθαρο ότι οι παραπάνω ορισμοί παρατίθενται όχι για να δομήσουν έναν τελικό ορισμό της επαναχρησιμοποίησης κώδικα αλλά για να δώσουν μία σφαιρική εικόνα του πως αντιλαμβάνονται τον όρο διαφορετικές ομάδες ενδιαφέροντος. Η Wikipedia[7] αποτελεί μία online εγκυκλοπαίδεια στην οποία μπορεί να συνεισφέρει κείμενο ο οποιοσδήποτε. Επομένως εκφράζει την χαλαρή γνώμη του ευρύτερου κοινού. Το Miscosoft Developer Network (MSDN)[8] αποτελεί κομμάτι μιας από τις μεγαλύτερες εταιρίες λογισμικού παγκοσμίως. Θα μπορούσαμε λοιπόν να θεωρήσουμε ότι μας προσφέρει την εταιρική σκοπιά ως προς την επαναχρησιμοποίηση λογισμικού. Τέλος, ο ορισμός που εμπεριέχεται στην εργασία των Frakes και συνεργατών με τίτλο «Sixteen questions about software reuse» [Frakes95] εκφράζει την οπτική του ερευνητή. Ενδιαφέρον είναι το γεγονός ότι παρά την διαφορά στη φύση αλλά και στα κίνητρα των τριών αυτών φορέων, οι ορισμοί τους εμφανίζουν πολλές ομοιότητες. Αυτό επιβεβαιώνει ακριβώς την παρατήρηση που κάναμε και προηγουμένως, ότι δηλαδή η έννοια της επαναχρησιμοποίησης κώδικα, αν και καινούρια στη βιβλιογραφία, είναι διαισθητικά οικεία σε όποιον έχει ασχοληθεί με την ανάπτυξη λογισμικού.

[7] http://en.wikipedia.org
[8] http://msdn.microsoft.com

1.2 Ο ρόλος του «μηχανικού επαναχρησιμοποίησης» (Reuse Engineer)

Στην παρούσα εργασία θα δείτε πολλές φορές να αναφερόμαστε στον όρο *«μηχανικός επαναχρησιμοποίησης»*. Ο όρος αυτός περιγράφει τον ρόλο μιας οντότητας (συνήθως φυσικού προσώπου) η οποία επιχειρεί να επαναχρησιμοποιήσει κώδικα είτε προσαρμόζοντας των επαναχρησιμοποιήσιμο κώδικα στο σύστημα υπό ανάπτυξη (system under development), είτε το σύστημα υπό ανάπτυξη στον επαναχρησιμοποιήσιμο κώδικα, ή και τα δύο ταυτόχρονα. Τον ρόλο του μηχανικού λογισμικού μπορεί να αναλάβει κάποιος προγραμματιστής, ιδιαίτερα όταν υπάρχει απουσία κάποιας συστηματικής διαδικασίας επαναχρησιμοποίησης κώδικα, φαινόμενο που είναι πολύ συχνό σε περιβάλλοντα μικρομεσαίων επιχειρήσεων ανάπτυξης λογισμικού. Στον αντίποδα, μηχανικός λογισμικού θα μπορούσε να είναι όντως κάποιος μηχανικός στον οποίο έχει ανατεθεί το έργο της εύρεσης και προσαρμογής επαναχρησιμοποιήσιμων συστατικών λογισμικού σε περιβάλλοντα με περισσότερο συστηματικές προσεγγίσεις.

1.3 Πηγές επαναχρησιμοποιήσιμου κώδικα

Η έννοια του επαναχρησιμοποιήσιμου κώδικα είναι συγκεχυμένη. Λόγω της εξάπλωσης του Ανοικτού Κώδικα πολύ συχνά υπάρχει η παρανόηση ότι επαναχρησιμοποιήσιμος κώδικας είναι μόνον ο ανοικτός κώδικας. Στην πραγματικότητα, ένας τέτοιος ισχυρισμός είναι λάθος αφού επαναχρησιμοποιήσιμος κώδικας μπορεί να θεωρηθεί και ο κληρονομημένος κώδικας, ο οποίος δεν διατίθεται απαραίτητα ελεύθερα στο διαδίκτυο.

Εν γένει, επαναχρησιμοποιήσιμος κώδικας μπορεί να αντληθεί:

- Από αποθετήρια κώδικα ΕΛ/ΛΑΚ (source code forges): Ένα αποθετήριο κώδικα ΕΛ/ΛΑΚ αποτελεί μια δεξαμενή έργων ανοικτού λογισμικού των οποίων ο κώδικας είναι ελεύθερος για χρήση κάτω από τους όρους μιας ειδικής άδειας (GPL, LGPL, APACHE license, κλπ.).

Εικόνα 1: Η κεντρική ιστοσελίδα του SourceForge.com

- Από βιβλιοθήκες (libraries): Πρόκειται για πακέτα αρχείων πηγαίου κώδικα τα οποία επιτελούν συγκεκριμένο έργο (π.χ. διαχείριση λογαριασμού e-mail μέσω POP3 πρωτοκόλλου, διαχείριση ετικετών mp3 αρχείων, σχεδιασμών γραφημάτων, κ.α.). Οι βιβλιοθήκες αυτές δεν αποτελούν (συνήθως) αυτοτελή έργα και ο προγραμματιστής συνήθως έχει πρόσβαση στις δυνατότητές τους μέσω κάποιας Διεπαφής Προγραμματισμού Εφαρμογής (Application Programming Interface (API)).

- Κώδικας από εξειδικευμένες μηχανές αναζήτησης (koders[9], krugle[10], κ.α.): Επιτρέπουν την αναζήτηση σε επίπεδο ολόκληρου έργου (project) ή επίπεδο κλάσης (class). Στην πραγματικότητα μοιάζουν πολύ με τα αποθετήρια κώδικα ΕΛ/ΛΑΚ με μία σημαντική διαφορά. Τα forges εμπεριέχουν το στοιχείο της συνεργατικότητας αφού σε κάθε εφαρμογή υπάρχουν εγγεγραμμένοι προγραμματιστές που την αναπτύσσουν, επεκτείνουν, τεστάρουν, κλπ. Στις μηχανές αναζήτησης πηγαίου κώδικα ο στόχος είναι η αναζήτηση επαναχρησιμοποιήσιμου κώδικα χωρίς ο χρήστης να έχει τη δυνατότητα να επέμβει, αλλάζοντας τον τρέχοντα κώδικα.

Εικόνα 2: Η κεντρική ιστοσελίδα του Koders.com

[9] http://www.koders.com/
[10] http://www.krugle.com/

- Κώδικας από στοχευμένα φόρα ή ιστολόγια, πανεπιστημιακά μαθήματα προσανατολισμένα στον προγραμματισμό, κλπ.: Αναπτύσσοντας ένα έργο λογισμικού υπάρχει μεγάλη πιθανότητα να συναντήσουμε μια σειρά από τετριμμένες απαιτήσεις. Ανάγνωση από / εγγραφή σε κάποιο αρχείο, σύνδεση με μία βάση δεδομένων, ταξινόμηση στοιχείων ενός πίνακα, είναι μερικές κλασικές περιπτώσεις τετριμμένων απαιτήσεων. Λόγω της απλότητας των περιπτώσεων αυτών, επαναχρησιμοποιήσιμος κώδικας είναι δυνατό να βρεθεί σε φόρα ή ιστολόγια στοχευμένα σε θέματα προγραμματισμού. Παρόμοια πληροφορία μπορεί να περιέχεται σε ιστοσελίδες πανεπιστημιακών μαθημάτων που έχουν σχέση με γλώσσες προγραμματισμού. Το MIT Open Courseware, πλατφόρμα που προσφέρει τίτλους ακαδημαϊκών μαθημάτων του πανεπιστημίου του MIT δωρεάν μέσω του διαδικτύου αποτελεί χαρακτηριστικό παράδειγμα.

Εικόνα 3: MIT Open Courseware

- Κώδικας από υπηρεσίες κοινωνικής δικτύωσης: Οι υπηρεσίες κοινωνικής δικτύωσης (Social Networking Services ή απλά Social Networks)[11] αποτελούν αδιαμφισβήτητα την πιο πρόσφατη επανάσταση στον παγκόσμιο ιστό. Εκτός των υπηρεσιών νέων, εικόνας, ήχου, βίντεο, κλπ. που έκαναν την εμφάνισή τους σχεδόν άμεσα, έχουν δημιουργηθεί επίσης υπηρεσίες κοινωνικής δικτύωσης με θεματολογία τον πηγαίο κώδικα. Οι υπηρεσίες αυτές εφαρμόζουν το στοιχείο της κοινωνικής δικτύωσης με προσανατολισμό στον πηγαίο κώδικα. Οι χρήστες συνήθως αποστέλλουν αρχεία πηγαίου κώδικα τάξεως κλάσης. Επιτρέπουν επίσης την αναζήτηση, συνήθως με βάση την γλώσσα προγραμματισμού. Παράδειγμα τέτοιας υπηρεσίας αποτελεί το Snipt[12]

1.4 Θέματα αδειοδότησης

Όταν μελετούμε ζητήματα επαναχρησιμοποιήσιμου κώδικα, πάντοτε ανακύπτει το θέμα της αδειοδότησης. Εκτός από την περίπτωση που έχουμε να κάνουμε με κληρονομημένο κώδικα (legacy code), σε όλες τις υπόλοιπες περιπτώσεις η ύπαρξη άδειας κρίνεται απαραίτητη για να μπορεί να γίνει επαναχρησιμοποίηση κώδικα.

Στην περίπτωση επαναχρησιμοποίησης βιβλιοθηκών ή κώδικα που αντλήθηκε από αποθετήρια κώδικα ΕΛ/ΛΑΚ ή μηχανές αναζήτησης, η ύπαρξη άδειας αναφέρεται ρητά και συνήθως αντίγραφό της συνοδεύει το κώδικα που επιστρέφεται μετά από κάποια αναζήτηση.

[11] Πολύ συχνά στη διεθνή βιβλιογραφία οι υπηρεσίες αυτές αναφέρονται και ως Web 2.0 Services
[12] http://snipt.org/

Στις περιπτώσεις άντλησης κώδικα από μη οργανωμένες πηγές, όπως στοχευμένα φόρα, ιστολόγια, κλπ. η ύπαρξη άδειας δεν είναι δεδομένη. Για την ακρίβεια της περισσότερες φορές δεν γίνεται καμία αναφορά σε θέματα αδειοδότησης. Αυτό συμβαίνει κυρίως γιατί ο κώδικας που φιλοξενείται σε τέτοιους ιστοχώρους είναι μικρός σε έκταση και επιτελεί «τετριμμένη» λειτουργικότητα. Πρόκειται για ζητήματα υλοποίησης που έχουν αντιμετωπιστεί πολλάκις και ως εκ τούτου δεν τίθενται καν θέματα κατοχύρωσης πνευματικών δικαιωμάτων από τους δημιουργούς τους.

Στις υπηρεσίες κοινωνικής δικτύωσης δεν φαίνεται να ακολουθείται κάποια πεπατημένη. Ο πηγαίος κώδικας αποστέλλεται από τους χρήστες χωρίς να συμβαίνει κάποιου είδους έλεγχος (αυτοματοποιημένος ή μη). Ως εκ τούτου, η ύπαρξη η όχι άδειας εξαρτάται αποκλειστικά από το αν ο συγγραφέας του κώδικα την έχει επισυνάψει στο αρχείο που δημοσίευσε στην εκάστοτε υπηρεσία. Δεδομένου ωστόσο, ότι τέτοιου είδους κοινωνικά δίκτυα αφενός δημιουργήθηκαν για να υποβοηθούν την διακίνηση πληροφορίας στο διαδίκτυο και αφετέρου παρέχουν πρόσβαση στις υπηρεσίες τους μέσω Application Programming Interfaces (APIs), είναι ασφαλές να θεωρήσουμε ότι ο κώδικας που φιλοξενείται μπορεί να επαναχρησιμοποιηθεί χωρίς να επιβάλλεται η ύπαρξη κάποιας σχετικής άδειας. Παρόμοια υπόθεση ισχύει και για τον κώδικα που αποτελεί μέρος σημειώσεων διαλέξεων κάποιου ακαδημαϊκού μαθήματος που προσφέρεται ελεύθερα στο διαδίκτυο. Παρ' όλο που σε μια τέτοια περίπτωση, αποκλείεται κάθε κομμάτι κώδικα να συνοδεύεται από κάποια άδεια χρήσης είναι θεμιτό ο επισκέπτης να θεωρήσει αυτονόητη την δυνατότητα πειραματισμού και άρα επαναχρησιμοποίησής του. Μία καλή πρακτική για τις περιπτώσεις που δεν είναι ξεκάθαρο αν ο κώδικας διατίθεται κάτω από

άδειας ανοικτού λογισμικού ή όχι, είναι η επικοινωνία με τον δημιουργό και αίτηση γραπτής άδειας (υπό μορφή e-mail), επαναχρησιμοποίησής του.

1.5 Κριτήρια επιλογής επαναχρησιμοποιήσιμου κώδικα

Όπως τονίσθηκε στην εισαγωγική παράγραφο, είναι αδιαμφισβήτητη πλέον η ύπαρξη μεγάλης ποσότητας ανοικτού, και άρα επαναχρησιμοποιήσιμου, κώδικα στο διαδίκτυο. Στις προηγούμενες δύο παραγράφους μελετήσαμε τις πηγές άντλησης επαναχρησιμοποιήσιμου κώδικα καθώς και τα θέματα αδειοδότησης. Ένα ακόμη σημαντικό κομμάτι στην διαδικασία επαναχρησιμοποίησης κώδικα είναι τα κριτήρια με βάση τα οποία ο μηχανικός επαναχρησιμοποίησης θα προτιμήσει κάποια εναλλακτική κώδικα έναντι κάποιας άλλης. Στη λίστα που ακολουθεί αναφέρονται συγκεντρωμένα τα βασικότερα από αυτά:

- Αναγνωσιμότητα: Ο καλογραμμένος κώδικας αποτελεί γενικότερα ζητούμενο στην τεχνολογία λογισμικού. Όταν μιλάμε για επαναχρησιμοποίηση κώδικα η ανάγκη για ευανάγνωστο κώδικα είναι ακόμη μεγαλύτερη αφού ο ενδιαφερόμενος καλείται συνήθως να επιλέξει γρήγορα κάποια από τις εναλλακτικές. Τον ενδιαφέρει λοιπόν ο κώδικας που θα επιλέξει να είναι κατά το δυνατό πιο ευανάγνωστος ώστε η διαδικασία προσαρμογής στο γενικότερο σύστημα να τελειώσει γρήγορα και χωρίς δυσάρεστες καταστάσεις.

- Ευκολία στην κατανόηση: Αυτό το χαρακτηριστικό εμπεριέχεται, εν μέρει, στην αναγνωσιμότητα ωστόσο το εξετάζουμε ως χωριστό για να τονίσουμε κάποια ιδιαίτερα σημεία που περιλαμβάνει. Η αναγνωσιμότητα έχει να κάνει περισσότερο με την μορφή του πηγαίου κώδικα αυτού κάθε αυτό. Περιλαμβάνει ζητήματα όπως η τήρηση εσοχών, καλή παρουσίαση των διακλαδώσεων αποφάσεων (if… else if.. else…), κλπ. Θέματα μορφοποίησης εν γένει. Το πόσο κατανοητό είναι ένα κομμάτι πηγαίου κώδικα, εξαρτάται, αν μιλάμε για αντικειμενοστραφή ανάπτυξη για παράδειγμα, από την σωστή επιλογή ονομάτων για τοπικές μεταβλητές, χαρακτηριστικά και μεθόδους για τις κλάσεις, κ.ο.κ.

- Ύπαρξη τεκμηρίωσης: Η επαναχρησιμοποίηση κώδικα βοηθά, εν γένει, στο να αυξηθεί η ταχύτητα της ανάπτυξης του έργου μειώνοντας παράλληλα την ανάγκη για εκτενή έλεγχο του κώδικα (θεωρώντας ότι ο επαναχρησιμοποιήσιμος κώδικας έχει ήδη ελεγχθεί μία τουλάχιστον φορά για σφάλματα πριν δημοσιευθεί στο διαδίκτυο). Η ύπαρξη τεκμηρίωσης σε συνδυασμό με το κριτήριο της ευκολίας στην κατανόηση, που είδαμε προηγουμένως, επιταχύνουν τη διαδικασία επιλογής μεταξύ συστατικών λογισμικού με παρόμοια λειτουργικότητα από τον μηχανικό επαναχρησιμοποίησης. Μπορούμε εύκολα να αντιληφθούμε ότι κώδικας με ελλιπή ή καθόλου τεκμηρίωση μπορεί να καθυστερήσει τη διαδικασία επιλογής σε τέτοιο βαθμό, που η υλοποίηση από το μηδέν να είναι περισσότερο συμφέρουσα.
- Αυτονομία: Όσο πιο αυτόνομο είναι ένα συστατικό λογισμικού, τόσο πιο εύκολη γίνεται η επαναχρησιμοποίησή του. Συστατικά λογισμικού με πολλές εξαρτήσεις

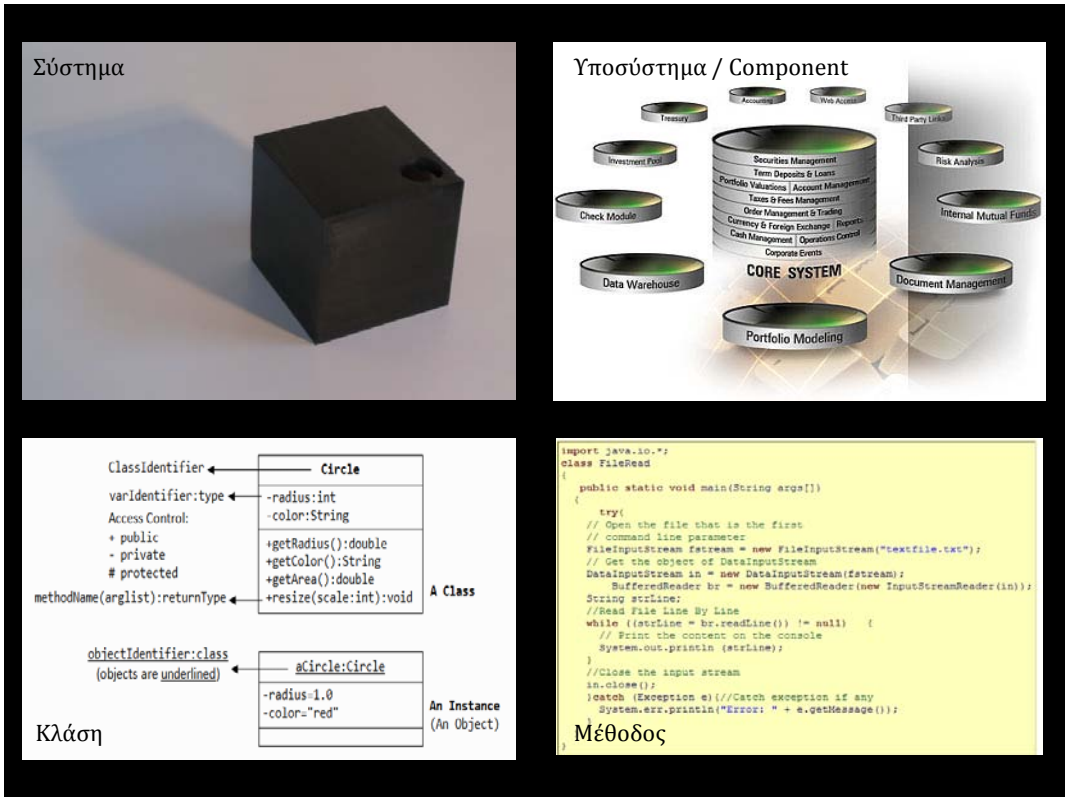
εμφανίζουν προβλήματα κατά τη διαδικασία ενσωμάτωσής τους στο προς ανάπτυξη σύστημα.

- Διάθεση κάτω από κατάλληλη άδεια: Θα πρέπει ο κώδικας που πρόκειται να επαναχρησιμοποιηθεί να διατίθεται «ελεύθερα» κάτω από κατάλληλη άδεια. Τα έργα ΕΛ/ΛΑΚ ακολουθούν στο σύνολό τους τέτοιες άδειες για αυτό και κρίνονται τα πλέον κατάλληλα για επαναχρησιμοποίηση.

1.6 Διαδικασία αναζήτησης

Κατά τη διαδικασία ανάπτυξης ενός έργου λογισμικού οι ανάγκες για επαναχρησιμοποιήσιμο κώδικα μπορεί να διαφέρουν. Αρχικά, γίνεται προσπάθεια εύρεσης συστατικών μεγάλου μεγέθους για την κάλυψη κατά το δυνατών μεγαλύτερων απαιτήσεων. Σε δεύτερο χρόνο, αναζητούνται συστατικά μικρότερης κλίμακας ή ακόμη και μέρη συστατικών για να καλύψουν κενά που πιθανώς εκκρεμούν στη μέχρι τώρα υλοποίηση. Τέλος, κατά τη διαδικασία ανάπτυξης κώδικα από το μηδέν, πολύ συχνά αναζητείται επαναχρησιμοποιήσιμος κώδικας για να αντιμετωπιστούν συγκεκριμένες, τετριμμένες λειτουργίες.

Από τα παραπάνω γίνεται φανερό ότι η αναζήτηση επαναχρησιμοποιήσιμου κώδικα μπορεί να αφορά πηγαίο κώδικα το μέγεθος του οποίου να κυμαίνεται από το μέγεθος ενός μεγάλου συστατικού λογισμικού, ως εκείνο μιας μεθόδου (βλ. Εικόνα 4).



Εικόνα 4: Αναζήτηση επαναχρησιμοποιήσιμου κώδικα με βάση το μέγεθος

1.6.1 Κριτήριο αναζήτησης

Από τα παραπάνω γίνεται εμφανές ότι βασικό κριτήριο αναζήτησης αποτελεί το μέγεθος του συστήματος ή της απαίτησης προς υλοποίηση. Όσο μεγαλώνει το μέγεθος του ζητούμενου, τόσο αυξάνεται και η πολυπλοκότητά του. Με βάση το μέγεθος / πολυπλοκότητα οι πηγές επαναχρησιμοποιήσιμου κώδικα που αναλύθηκαν στην ενότητα 1.3 κατηγοριοποιούνται όπως δείχνει ο Πίνακας 1 (βλ. επόμενη σελίδα).

Μεγάλο μέγεθος / Υψηλή πολυπλοκότητα	**Μικρό μέγεθος / Χαμηλή πολυπλοκότητα**
Βιβλιοθήκες (jars, κλπ.)	Πηγαίος κώδικας από υπηρεσίες κοινωνικής δικτύωσης (snipt, κλπ.)
Πηγαίος κώδικας από αποθετήρια κώδικα ΕΛ/ΛΑΚ (source code forges)	Πηγαίος κώδικας από στοχευμένα φόρα, ιστολόγια, κλπ.
Πηγαίος κώδικας από ειδικές μηχανές αναζήτησης (sourceforge, koders, κλπ.)	Πηγαίος κώδικας από ειδικές μηχανές αναζήτησης (sourceforge, koders, κλπ.)
Κληρονομημένος κώδικας (legacy code)	Κληρονομημένος κώδικας (legacy code)

Πίνακας 1: Διαχωρισμός πηγών επαναχρησιμοποιήσιμου κώδικα με βάση το μέγεθος

Όταν το ζητούμενο εμφανίζει αυξημένο μέγεθος ή πολυπλοκότητα είναι φυσικό να στραφούμε σε πιο οργανωμένες πηγές που φιλοξενούν κυρίως συστατικά λογισμικού (και όχι τόσο μεμονωμένα αρχεία πηγαίου κώδικα). Η χρήση έτοιμων πακέτων κώδικα (υπό τη μορφή βιβλιοθηκών) ή αποθετηρίων κώδικα ΕΛ/ΛΑΚ αποτελούν τις πρωταρχικές επιλογές του μηχανικού επαναχρησιμοποίησης σε αυτήν την κατηγορία αναζήτηση. Ειδικές μηχανές αναζήτησης κώδικα μπορούν να εξυπηρετήσουν επίσης αλλά σπάνια θα δώσουν την ποσότητα των εναλλακτικών που θα λάβουμε από τις δύο προαναφερθείσες πηγές. Τέλος και ο κληρονομημένος κώδικας, εφόσον υπάρχει, είναι δυνατό να δώσει λύση. Στην περίπτωση αυτή, το κέρδος είναι μεγαλύτερο αφού ο κώδικας προς επαναχρησιμοποίηση αναπτύχθηκε από την ίδια την εταιρία, πράγμα που συνεπάγεται, σχεδόν βέβαιη συνοχή με το υπόλοιπο προγραμματιστικό μοτίβο καθώς

και εγγύηση ότι πέρασε από τη διαδικασία ελέγχου που προβλέπει το εκάστοτε ενδοεταιρικό ή προσωπικό, αν πρόκειται για ελεύθερο επαγγελματία, μοτίβο.

Για μικρότερης κλίμακας ζητούμενα η αναζήτηση λύσεων σε ολόκληρα συστατικά λογισμικού είναι μάλλον μία κακή επιλογή. Όταν η αναζήτηση γίνεται σε επίπεδο κλάσης ή μεθόδου, βοηθά πολύ περισσότερο ένας μηχανισμός που να επιστρέφει γρήγορα πλειάδα αποτελεσμάτων. Σε αυτήν την κατηγορία αναζητήσεων, ο επαναχρησιμοποιήσιμος κώδικας είναι δεδομένο ότι θα αλλάξει, έστω και ελάχιστα, πριν ενσωματωθεί στο προς ανάπτυξη σύστημα. Επίσης, ο έλεγχος για σφάλματα είναι αναπόφευκτός. Υπηρεσίες κοινωνικής δικτύωσης, φόρα, ιστολόγια και ακαδημαϊκά μαθήματα στοχευμένα στην ανάπτυξη λογισμικού ή τις γλώσσες προγραμματισμού αποτελούν τις καλύτερες πηγές αναζήτησης κώδικα που εμπίπτει σε αυτήν την κατηγορία. Επίσης, λύση μπορεί να δοθεί και εδώ από ειδικές μηχανές αναζήτησης αφού έχουν τη δυνατότητα να αναζητούν τόσο σε επίπεδο συστατικού λογισμικού όσο και σε επίπεδο αρχείων πηγαίου κώδικα. Τέλος, είναι φανερό ότι σε περίπτωση ύπαρξης κληρονομημένου κώδικα υπάρχει μεγάλη πιθανότητα κάποια από τα ζητούμενα αυτού του τύπου να έχουν ήδη υλοποιηθεί.

1.6.2 Τύποι αναζήτησης

Με βάση τον διαχωρισμό των πηγών που είδαμε στην προηγούμενη παράγραφο, διακρίνουμε δύο ειδών αναζητήσεις:

- Αναζήτηση κώδικα σε αποθετήρια έργων ΕΛ/ΛΑΚ: Η αναζήτηση αυτή περιλαμβάνει ουσιαστικά την αναζήτηση σε όλες τις πιθανές πηγές που φιλοξενούν επαναχρησιμοποιήσιμο κώδικα υπό τη μορφή συστατικών λογισμικού. Στόχος είναι η κάλυψη βασικών λειτουργικών απαιτήσεων για τις οποίες απαιτείται η συγγραφή πολλών γραμμών κώδικα ενώ εμφανίζουν υψηλή πολυπλοκότητα. Αξίζει να σημειωθεί ότι η ονομασία *«Αναζήτηση κώδικα σε αποθετήρια έργων ΕΛ/ΛΑΚ»* αντικατοπτρίζει μόνον τη φύση της αναζήτησης. Στην πραγματικότητα περιλαμβάνει όλες τις πηγές που αναφέρονται στην αριστερή στήλη του Πίνακα 1.

- Αναζήτηση περιπτώσεων τετριμμένου κώδικα: Αφορά στην αναζήτηση μικρού σε έκταση και απλού σχετικά κώδικα. Σε αυτού του είδους την αναζήτηση ο χρήστης στοχεύει περισσότερο σε ποσοτικά πολλά παρά ποιοτικά αποτελέσματα. Συνήθως αναζητά πηγαίο κώδικα της τάξεως ενός αρχείου κλάσης ή μιας μεθόδου. Συνήθως η χρήση μιας απλής μηχανής αναζήτησης (π.χ. Google[13], Yahoo![14], Bing[15]) επιστρέφει πλειάδα σχετικών αποτελεσμάτων. Ωστόσο, ως εν δυνάμει εναλλακτικές πηγές αναζήτησης μπορεί να γίνει χρήση των πηγών που αναφέρονται τη δεξιά στήλη του Πίνακα 1.

Στους πίνακες 2 και 3 μπορείτε να βρείτε τα βασικά πλεονεκτήματα και μειονεκτήματα της «Αναζήτησης κώδικα σε αποθετήρια έργων ΕΛ/ΛΑΚ» και της «Αναζήτησης περιπτώσεων τετριμμένου κώδικα αντίστοιχα.

[13] http://www.google.com
[14] http://www.yahoo.com
[15] http://www.bing.com

Αναζήτηση κώδικα σε αποθετήρια έργων ΕΛ/ΛΑΚ	
Πλεονεκτήματα	**Μειονεκτήματα**
Καλύπτει μεγάλες ανάγκες υλοποίησης	Επίπονη και αργή διαδικασία
Ο κώδικας είναι συνήθως καλογραμμένος, και τεκμηριωμένος	Απαιτείται χρόνος για εξοικείωση με τον κώδικα που πρόκειται να χρησιμοποιήσουμε
Αυξημένες πιθανότητες για ποιοτικό κώδικα	Μικρή απόκριση σε πολύ εξεζητημένα θέματα
Συνήθως συνοδεύεται από άδεια	

Πίνακας 2. Πλεονεκτήματα / Μειονεκτήματα αναζήτησης κώδικα σε αποθετήρια έργων ΕΛ/ΛΑΚ

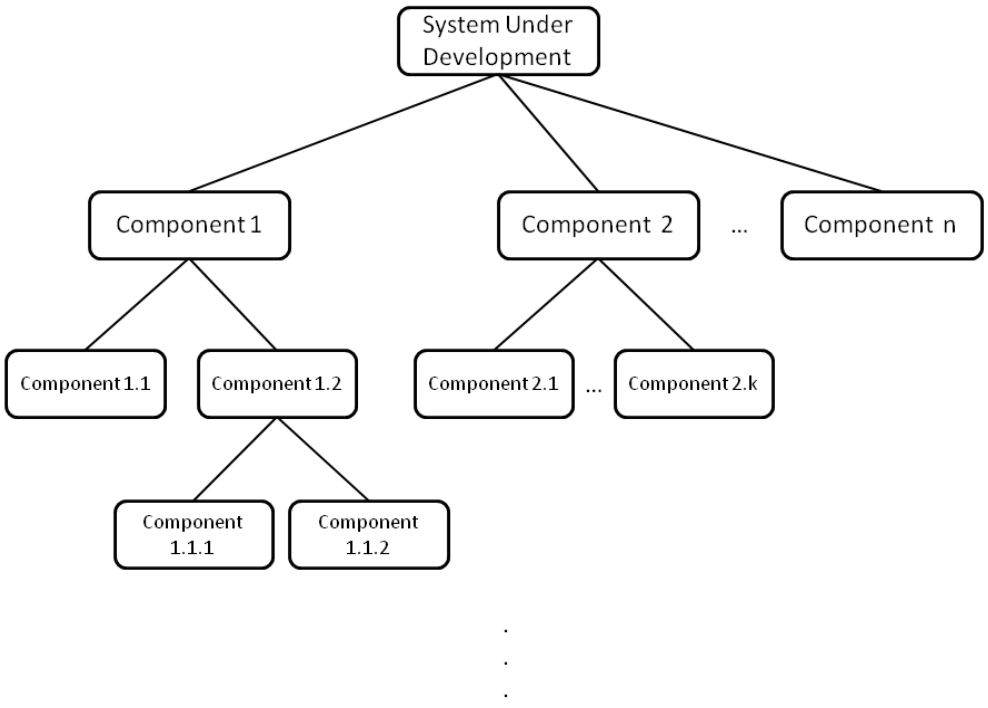
Αναζήτηση περιπτώσεων τετριμμένου κώδικα	
Πλεονεκτήματα	**Μειονεκτήματα**
Γρήγορη απόκριση	Χαμηλής ποιότητας κώδικας (συνήθως κακογραμμένος)
Μαζική συμμετοχή (π.χ. Forums, Social Networks)	Εν δυνάμει θέματα πνευματικών δικαιωμάτων
Δυνατότητα μέτρησης αξιοπιστίας (π.χ. Mailing Lists)	Η αναζήτηση αυτού του τύπου λειτουργεί καλά για πολύ μικρά κομμάτια κώδικα μόνο
Πλειάδα υλικού (οι βασικότερες των ενεργειών φιλοξενούνται σε sites ακαδημαϊκών χώρων)	

Πίνακας 3. Πλεονεκτήματα / Μειονεκτήματα αναζήτησης περιπτώσεων τετριμμένου κώδικα

Κεφάλαιο 2

ΜΕΛΕΤΕΣ ΠΕΡΙΠΤΩΣΗΣ ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΙΜΟΥ ΚΩΔΙΚΑ

Η επαναχρησιμοποίηση λογισμικού βασίζεται στη λογική αντιμετώπισης των απαιτήσεων του υπό ανάπτυξη συστήματος υλοποιώντας το τμηματικά. Πριν προχωρήσουμε παρουσιάζοντας λεπτομερέστερα την λογική της τμηματικής ανάπτυξης όμως, θα πρέπει να ορίσουμε την έννοια «τμήμα λογισμικού». Το μεγαλύτερο ποσοστό επαναχρησιμοποιήσιμου κώδικα συναντάται σε αποθετήρια κώδικα έργων ΕΛ/ΛΑΚ. Επιπρόσθετα, είναι κοινή πρακτική για όσους αναπτύσσουν εφαρμογές ανοικτού λογισμικού, να οργανώνουν τον κώδικά τους σε συστατικά λογισμικού (software components) άλλοτε μικρότερα και άλλοτε μεγαλύτερα.



Εικόνα 5: Σύστημα υπό ανάπτυξη αποδομημένο σε συστατικά λογισμικού

Έχοντας αυτό στο μυαλό μας μπορούμε να επιστρέψουμε στο έγγραφο απαιτήσεων του συστήματος που μας ζητήθηκε να υλοποιήσουμε και να τις οργανώσουμε σε πιθανά συστατικά λογισμικού ακολουθώντας μια διαδικασία παρόμοια με αυτήν που απεικονίζεται στην Εικόνα 5.

Αρχικά, θεωρούμε κάθε μία από τις λειτουργικές απαιτήσεις ως ξεχωριστό συστατικό λογισμικού. Στη συνέχεια, ανάλογα με το πόσο πολύπλοκη λειτουργικότητα εμφανίζει κάθε ένα από αυτά τα συστατικά, μπορούμε να τα αποδομήσουμε σε απλούστερα, αν κριθεί απαραίτητο ή να τα αφήσουμε ως έχουν.

Ακολουθώντας αυτή τη διαδικασία καταλήγουμε με μία δενδρική δομή η οποία έχει σαν ρίζα το υπό ανάπτυξη σύστημα και σαν φύλλα, τα συστατικά λογισμικού που θα πρέπει να υλοποιηθούν ώστε να ολοκληρωθεί η ανάπτυξη του ζητούμενου συστήματος. Προφανώς οι ενδιάμεσοι κόμβοι δεν είναι τίποτε περισσότερο παρά ομάδες συστατικών λογισμικού. Δεν έχουν κάποια ουσιαστική αξία για την μέθοδό μας εκτός από το να παραπέμπουν στις αρχικές περισσότερο *αφηρημένες* απαιτήσεις.

Εφόσον αυτά τα συστατικά λογισμικού έχουν ορισθεί, είμαστε έτοιμοι να αρχίσουμε την αναζήτησή τους στα αποθετήρια έργων ΕΛ/ΛΑΚ. Κατά τη διάρκεια αυτού του ιδιότυπου «σαφάρι» συστατικών μπορεί να έρθουμε αντιμέτωποι με μία από τις παρακάτω περιπτώσεις:

- Το συστατικό λογισμικού που μας ενδιαφέρει είναι διαθέσιμο: Σε αυτή την περίπτωση το μόνο που έχουμε να κάνουμε είναι να τροποποιήσουμε το συστατικό με βάση των αναγκών μας και να το ενσωματώσουμε στην υπόλοιπη υλοποίησή μας.

- Το συστατικό λογισμικού που αναζητούμε δεν υπάρχει αλλά υποσύνολα αυτού είναι διαθέσιμα: Σε αυτήν την περίπτωση θα πρέπει να ανατρέξουμε στην δενδρική αναπαράσταση του υπό ανάπτυξη συστήματός μας και να επεκτείνουμε τον κόμβο που απεικονίζει το συστατικό λογισμικού το οποίο αναζητούμε, αποδομώντας το σε απλούστερα, κατά το δοκούν.

- Το συστατικό λογισμικού που αναζητούμε δεν υπάρχει και η διαδικασία αποδόμησής του σε απλούστερα δείχνει πολύ πιο χρονοβόρα από το να το υλοποιήσουμε από το μηδέν: Σε αυτήν την περίπτωση προχωρούμε στην υλοποίηση του συστατικού από το μηδέν. Δεδομένου μάλιστα ότι σε αυτήν την περίπτωση το προς υλοποίηση συστατικό λογισμικού αποτελεί συνήθως ένα κομμάτι κώδικα τετριμμένης λειτουργικότητας μπορούμε να απευθυνθούμε σε στοχευμένα σε γλώσσες προγραμματισμού φόρα, ιστολόγια ή ανοικτά στο κοινό ακαδημαϊκά μαθήματα για να επαναχρησιμοποιήσουμε κομμάτια κώδικα που επιτελούν βασική λειτουργικότητα (π.χ. ανάγνωση από / εγγραφή σε αρχείο). Συνήθως μια απλή αναζήτηση κάνοντας χρήση κάποιας κλασικής μηχανής αναζήτησης είναι αρκετή.

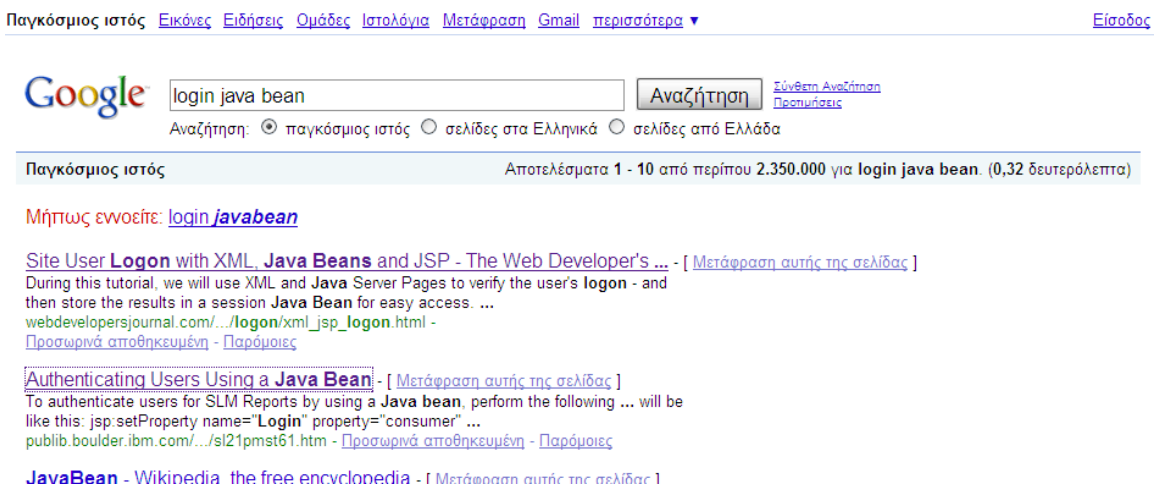
Θέλοντας να ελέγξουμε την αποτελεσματικότητα της επαναχρησιμοποίησης κώδικα σε πραγματικές συνθήκες ανάπτυξης λογισμικού πειραματιστήκαμε με δύο διαφορετικές μελέτες περίπτωσης, τα σενάρια των οποίων ακολουθούν.

2.1 Μελέτη περίπτωσης #1 :: Αναζήτηση συστατικού λογισμικού που να υλοποιεί την διαδικασία εισόδου (login) κάνοντας χρήση Java Beans.

Στόχος: Η εύρεση επαναχρησιμοποιήσιμου συστατικού λογισμικού που να υλοποιεί τη λειτουργικότητα εισόδου σε μια διαδικτυακή υπηρεσία και να έχει τη μορφή ενός Java Bean.

Η διαδικασία επαναχρησιμοποίησης που ακολουθήθηκε είναι η ακόλουθη:

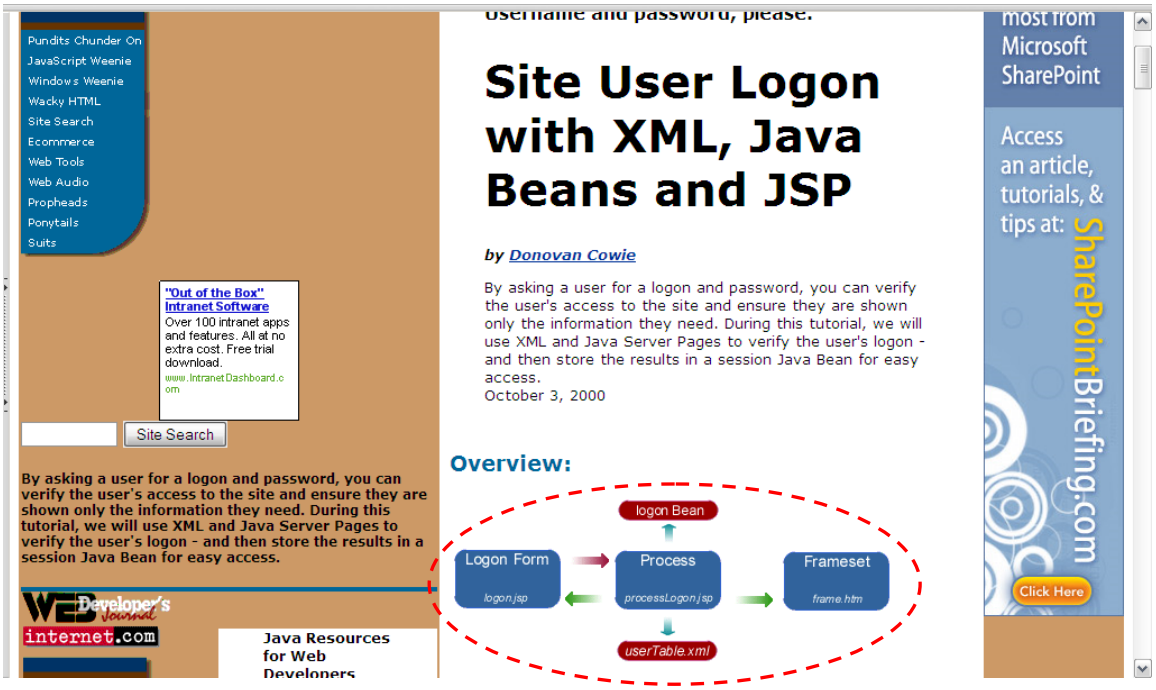
1. Αναζητούμε γρήγορη λύση (ως εκ τούτου χρησιμοποιούμε μια από τις γνωστές μηχανές αναζήτησης – στην προκειμένη περίπτωση την Google –). Δοκιμάζουμε τις εξής λέξεις κλειδιά “login java bean”.



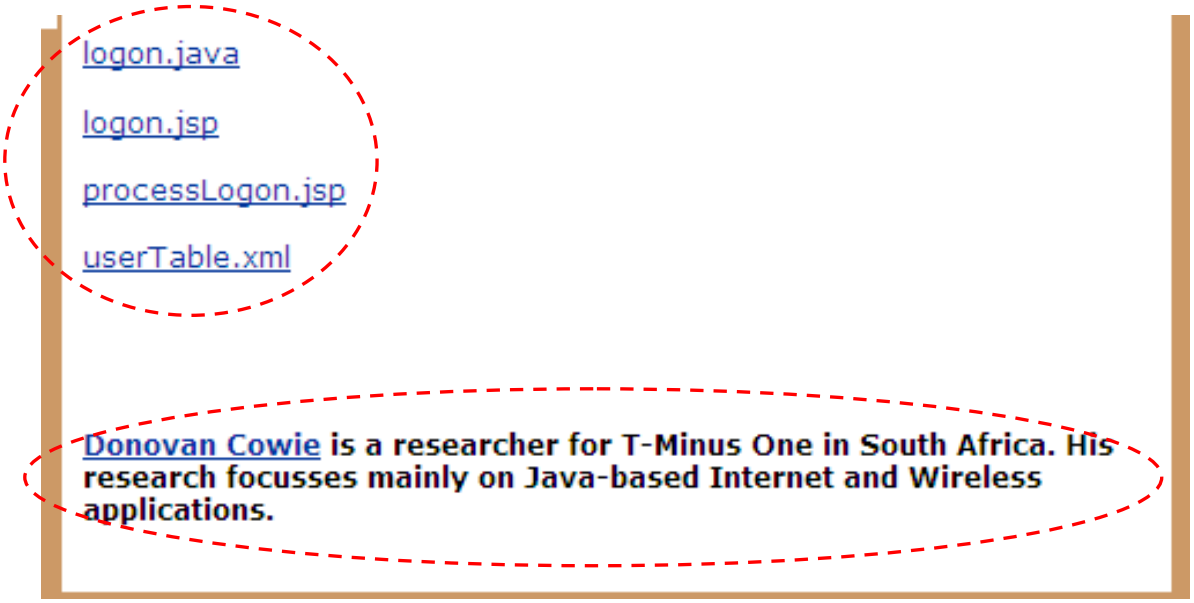
Εικόνα 6: Αναζήτηση συστατικού εισόδου μέσω Google

2. Το δεύτερο κατά σειρά αποτέλεσμα που τιτλοφορείται «Authenticating Users Using A Java Bean» φαίνεται πολλά υποσχόμενο (βλ. Εικόνα 6).

3. Επισκεπτόμενοι τον παραπάνω σύνδεσμο ανακαλύπτουμε ότι περιλαμβάνει πηγαίο κώδικα (υλοποιημένο με τη μορφή Java Bean) καθώς και αναλυτική περιγραφή της αρχιτεκτονικής του συγκεκριμένου λογισμικού. Επίσης υπάρχουν αναλυτικές πληροφορίες για τον δημιουργό που αναφέρουν ότι κατέχει τόσο την ακαδημαϊκή ιδιότητα όσο και αυτή του προγραμματιστή σε μια εταιρία λογισμικού.



Εικόνα 7: Η ιστοσελίδα που περιέχει το ζητούμενο συστατικό



Εικόνα 8: Πηγαίος κώδικας και πληροφορίες για τον συγγραφέα

4. Δεν αναφέρεται ρητά ότι ο παρόν κώδικας διατίθεται ελεύθερα. Ωστόσο ούτε υπάρχει αναφορά πνευματικής ιδιοκτησίας και απαγόρευσης επαναχρησιμοποίησης του κώδικα.

5. Το πιθανότερο είναι ότι μετά από ενημέρωση του συγγραφέα του κώδικα για την πρόθεσή μας να τον επαναχρησιμοποιήσουμε, θα πάρουμε την άδειά χωρίς πρόβλημα (με μεγάλη πιθανότητα βέβαια να υπάρξει η απαίτηση να αναφέρεται ρητά το όνομά του στον πηγαίο κώδικα).

Κατά τη διάρκεια της παραπάνω διαδικασίας:

- Δαπανήθηκαν περίπου 30'' (δευτερόλεπτα) για την αναζήτηση μέσω Google.
- Δαπανήθηκαν περίπου 5' (λεπτά) για να βεβαιωθούμε για την ποιότητα της ιστοσελίδας που ανακαλύψαμε.

Συνολικά χρειάστηκαν 5,5' (λεπτά) για να αποκτήσουμε ακριβώς το επαναχρησιμοποιήσιμο συστατικό λογισμικού που αναζητούσαμε.

2.2 Μελέτη περίπτωσης #2 :: Διπλή επαναχρησιμοποίηση κώδικα για μετανάστευση από μία Java βιβλιοθήκη διαχείρισης της υπηρεσίας WordNet σε μία άλλη, λόγω διένεξης.

Η μελέτη περίπτωσης είναι αληθινή. Έλαβε χώρα στα πλαίσια απασχόλησης του γράφοντος σε ανάπτυξη δικτυακής πλατφόρμας (για λογαριασμό ενός ερευνητικού κέντρου) στα πλαίσια έργου χρηματοδοτούμενο από την ευρωπαϊκή ένωση. Η

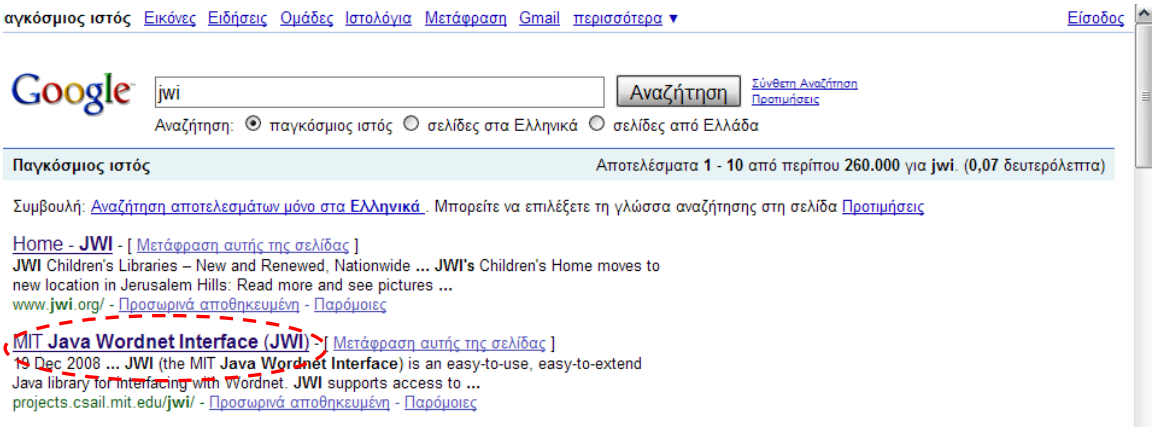
λεπτομερέστερη αναφορά στο έργο, τους στόχους του ή τον φορέα υλοποίησης παραλείπεται σκοπίμως αφενός γιατί δεν έχει να συνεισφέρει τίποτε στην μελέτη περίπτωσης αυτή κάθε αυτή και αφετέρου ο γράφων δεσμεύεται από τη σύμβαση έργου.

Έχει υλοποιηθεί μία κλάση Java που διαχειρίζεται την λειτουργικότητα της υπηρεσίας WordNet[16] στα πλαίσια του προαναφερθέντος έργου. Σε αυτή την υλοποίηση χρησιμοποιήθηκε η βιβλιοθήκη Java WordNet Library (JWNL) σαν μέσο για να επιτευχθεί η σύνδεση με της εφαρμογής που αναπτύσσεται[17] με το WordNet και να μπορεί να γίνεται χρήση των υπηρεσιών του.

Αφότου η υλοποίηση της προαναφερθείσας κλάσης έχει τελειώσει και ενώ γίνεται προσπάθεια να εξαχθεί η εφαρμογή, το προς ανάπτυξη σύστημά δηλαδή, ως Java OSGI bundle, μία διένεξη με την JWNL βιβλιοθήκη κάνει επιτακτική την ανάγκη να χρησιμοποιηθεί κάποια άλλη βιβλιοθήκη διαχείρισης του WordNet μέσω Java. Κάτι τέτοιο καθιστά εξαιρετικά πιθανή την περίπτωση να πρέπει να ξαναγραφεί ο κώδικας της Java κλάσης που δημιουργήσαμε ώστε να διαχειριζόμαστε το WordNet.

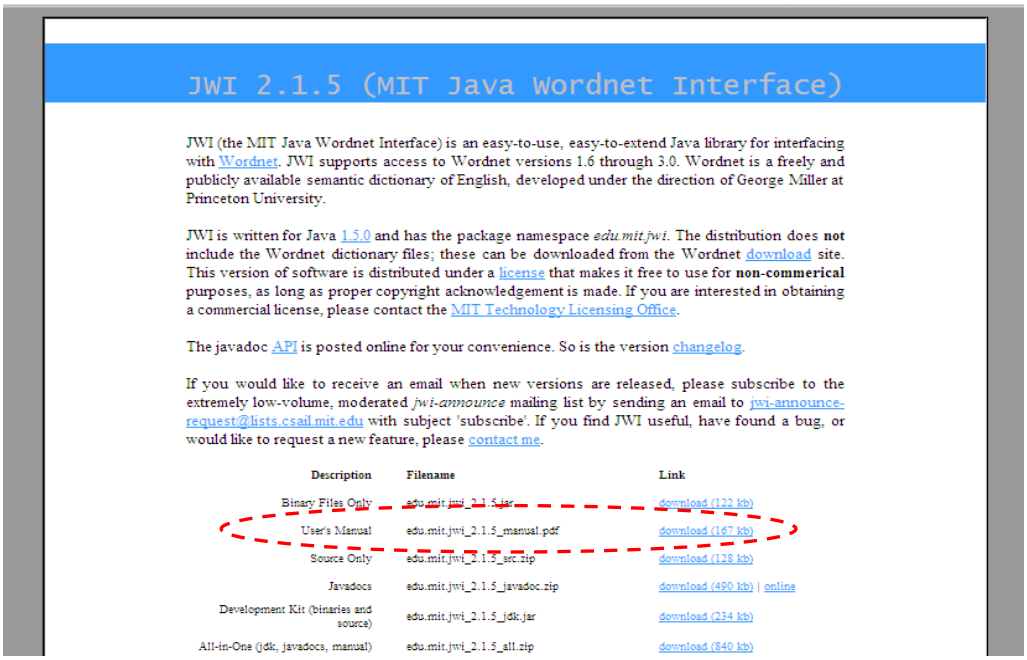
Ας υποθέσουμε τώρα ότι η βιβλιοθήκη – αντικαταστάτης έχει εντοπιστεί. Πρόκειται για την Java WordNet Interface (JWI) (βλ. Εικόνα 9).

[16] Το WordNet (http://wordnet.princeton.edu/) αποτελεί έναν διαδικτυακό θησαυρό της Αγγλικής γλώσσας. Έχει σχεδιαστεί και υλοποιηθεί από το πανεπιστημίο του Princeton.
[17] Η ανάλυση του γενικότερου υπό ανάπτυξη συστήματος παραλείπεται για λόγους καλύτερης παρουσίασης της περίπτωσης χρήσης.



Εικόνα 9: Αναζήτηση εναλλακτικής βιβλιοθήκης διαχείρισης του WordNet

Θα βόλευε αν μπορούσε να εντοπιστεί επαναχρησιμοποιήσιμος κώδικας για να δημιουργηθεί ταχύτερα η καινούρια κλάση – διαχειριστής του WordNet για σύστημά υπό ανάπτυξη. Μια μελέτη του εγχειριδίου χρήσης (manual) της βιβλιοθήκης (βλ. Εικόνα 10), στην επίσημη ιστοσελίδα της βιβλιοθήκης, αποκαλύπτει μία κλάση – υπόδειγμα που όμως φαίνεται να υλοποιεί το μεγαλύτερο μέρος της επιθυμητής λειτουργικότητας (βλ. Εικόνα 11).



JWI 2.1.5 (MIT Java Wordnet Interface)

JWI (the MIT Java Wordnet Interface) is an easy-to-use, easy-to-extend Java library for interfacing with Wordnet. JWI supports access to Wordnet versions 1.6 through 3.0. Wordnet is a freely and publicly available semantic dictionary of English, developed under the direction of George Miller at Princeton University.

JWI is written for Java 1.5.0 and has the package namespace *edu.mit.jwi*. The distribution does **not** include the Wordnet dictionary files; these can be downloaded from the Wordnet download site. This version of software is distributed under a license that makes it free to use for **non-commerical** purposes, as long as proper copyright acknowledgement is made. If you are interested in obtaining a commercial license, please contact the MIT Technology Licensing Office.

The javadoc API is posted online for your convenience. So is the version changelog.

If you would like to receive an email when new versions are released, please subscribe to the extremely low-volume, moderated *jwi-announce* mailing list by sending an email to jwi-announce-request@lists.csail.mit.edu with subject 'subscribe'. If you find JWI useful, have found a bug, or would like to request a new feature, please contact me.

Description	Filename	Link
Binary Files Only	edu.mit.jwi_2.1.5.jar	download (122 kb)
User's Manual	edu.mit.jwi_2.1.5_manual.pdf	download (167 kb)
Source Only	edu.mit.jwi_2.1.5_src.zip	download (128 kb)
Javadocs	edu.mit.jwi_2.1.5_javadoc.zip	download (490 kb) | online
Development Kit (binaries and source)	edu.mit.jwi_2.1.5_jdk.jar	download (234 kb)
All-in-One (jdk, javadocs, manual)	edu.mit.jwi_2.1.5_all.zip	download (840 kb)

Εικόνα 10: Το εγχειρίδιο χρήσης για την JWI

```
1  public void testDictionary() throws IOException {
2
3     // construct the URL to the Wordnet dictionary directory
4     String wnhome = System.getenv("WNHOME");
5     String path = wnhome + File.separator + "dict";
6     URL url = new URL("file", null, path);
7
8     // construct the dictionary object and open it
9     IDictionary dict = new Dictionary(url);
10    dict.open();
11
12    // look up first sense of the word "dog"
13    IIndexWord idxWord = dict.getIndexWord("dog", POS.NOUN);
14    IWordID wordID = idxWord.getWordIDs().get(0);
15    IWord word = dict.getWord(wordID);
16    System.out.println("Id = " + wordID);
17    System.out.println("Lemma = " + word.getLemma());
18    System.out.println("Gloss = " + word.getSynset().getGloss());
19
20 }
```

Figure 1: Sample Dictionary code

Εικόνα 11(α): Παράδειγμα κλάσης της βιβλιοθήκης JWI

```
1  Id = WID-2084071-n-?-dog
2  Lemma = dog
3  Gloss = a member of the genus Canis (probably descended from the
   common wolf) that has been domesticated by man since prehistoric
   times; occurs in many breeds; "the dog barked all night"
```

Figure 2: Sample Dictionary Code Output (for Wordnet 3.0)

Εικόνα 11(β): Έξοδος του παραδείγματος κλάσης JWI

Αντί για την προσαρμογή του επαναχρησιμοποιήσιμο κώδικα που ανακαλύφθηκε, εκ νέου, στις ανάγκες του έργου, αποφασίστηκε να διεξαχθεί ένα πείραμα. Να χρησιμοποιηθεί ο κώδικας που ανακαλύφθηκε για να προσαρμοστεί η πρώτη κλάση – διαχειριστής του WordNet που λειτουργούσε με την JWNL βιβλιοθήκη. Το γεγονός ότι το WordNet παρέχει συγκεκριμένα δεδομένα κάνει τις υλοποιήσεις των διαφόρων βιβλιοθηκών παρόμοιες πράγμα το οποίο, όπως θα περίμενε κανείς, κληρονομούν και οι διεπαφές προγραμματισμού εφαρμογών (APIs), επίσης.

Συνδυάζοντας αυτό το τέχνασμα με τον επαναχρησιμοποιήσιμο κώδικα που εντοπίστηκε, η καινούρια κλάση διαχείρισης του WordNet είχε υλοποιηθεί σε λιγότερο από μία ώρα (>1h). Πιο συγκεκριμένα:

- Δαπανήθηκαν 10’ για αναζήτηση της νέας βιβλιοθήκης (JWI).
- Δαπανήθηκαν 5’ για να εξοικειωθούμε με την κλάση – υπόδειγμα.
- Δαπανήθηκαν 30’ για να αλλάξουμε την πρώτη υλοποίηση της κλάσης διαχειριστή του WordNet που είχε φτιαχθεί ώστε να λειτουργεί με τη νέα βιβλιοθήκη.

- Δαπανήθηκαν 10’ για να ελέγξουμε την νέα μας υλοποίηση ως προς την λειτουργικότητά της.

Συνολικά δαπανήθηκαν περίπου 55’ < 1 ώρας.

Για λόγους πληρότητας θεωρήθηκε σκόπιμο να παρατεθούν οι χρόνοι ανάπτυξης της πρώτης υλοποίησης (που ήταν επίσης αποτέλεσμα επαναχρησιμοποίησης κώδικα).

Αναλυτικά:

- Δαπανήθηκε 1,5 ώρα για να εντοπιστεί επαναχρησιμοποιήσιμος κώδικας.
- Δαπανήθηκαν 4 ώρες για να προσαρμοστεί ο κώδικας που ανακαλύφθηκε στις ανάγκες του υπό ανάπτυξη συστήματος.
- Δαπανήθηκαν 2 ώρες για να ελεγχθεί η υλοποίηση για σφάλματα.

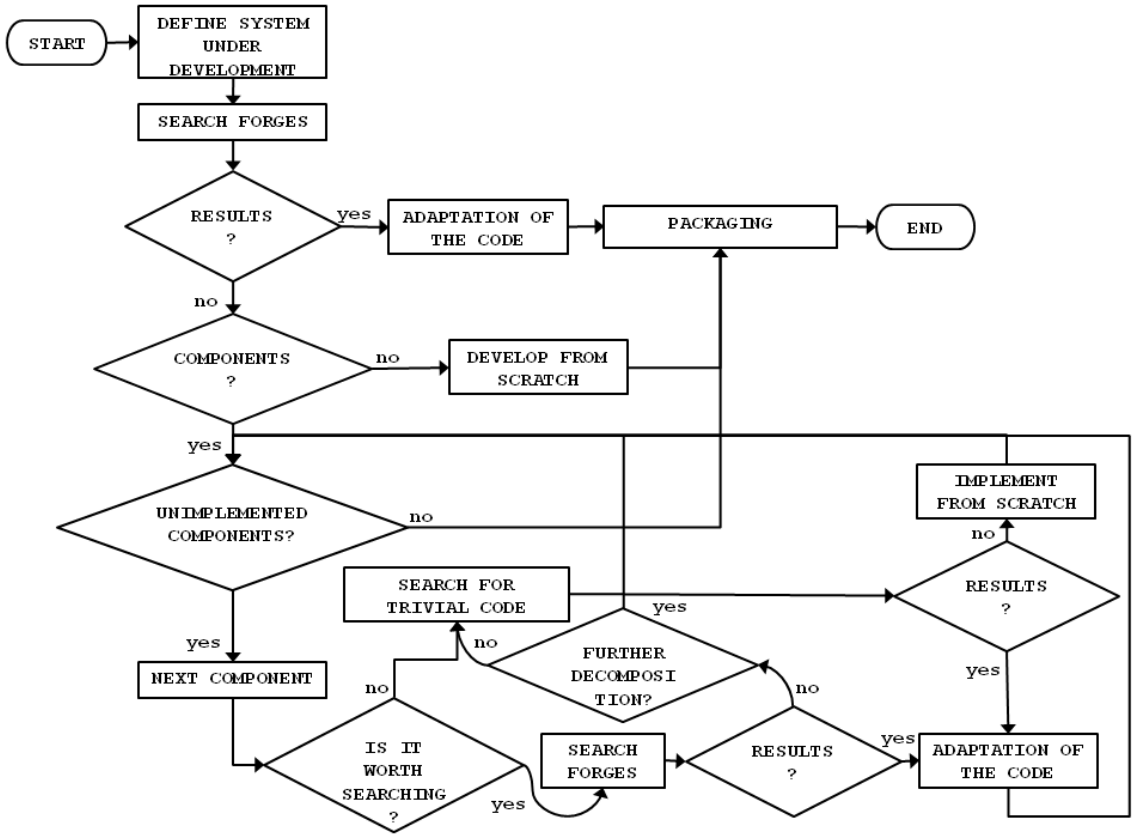
Συνολικά δαπανήθηκαν 7,5 ώρες για να φθάσει η πρώτη μας υλοποίηση στην τελική της μορφή.

Οι παραπάνω μελέτες περίπτωσης είχαν ως αποτέλεσμα τον σχεδιασμό ενός ημί-αυτόματου μοντέλου της διεργασίας επαναχρησιμοποίησης ΕΛ/ΛΑΚ το οποίο και παρουσιάζουμε αναλυτικά στο επόμενο κεφάλαιο.

Κεφάλαιο 3

ΕΝΑ ΗΜΙ-ΑΥΤΟΜΑΤΟ ΜΟΝΤΕΛΟ ΓΙΑ ΤΗ ΔΙΑΔΙΚΑΣΙΑ ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗΣ ΚΩΔΙΚΑ ΕΛ/ΛΑΚ

Σε αυτό το κεφάλαιο προσπαθούμε να οργανώσουμε τη γνώση που αποκομίσαμε από τις μελέτες περιπτώσεων που παρουσιάσαμε στο προηγούμενο κεφάλαιο σε μια πιο τυπική, σχηματική αναπαράσταση. Ένα μοντέλο.



Εικόνα 12: Μοντέλο επαναχρησιμοποίησης κώδικα ΕΛ/ΛΑΚ

Πρόκειται για ένα μοντέλο επαναχρησιμοποίησης κώδικα ΕΛ/ΛΑΚ (βλ. Εικόνα 12). Αν και μοιάζει δαιδαλώδες, εκ πρώτης όψεως, όταν επεξηγηθεί ο αναγνώστης ανακαλύπτει ότι είναι πολύ εύκολο να γίνει κατανοητό και να εφαρμοστεί.

Κατά την εκκίνηση ορίζεται το σύστημα που πρέπει να υλοποιηθεί (από εδώ και στο εξής θα αναφερόμαστε σε αυτό ως *σύστημα υπό υλοποίηση*). Ολόκληρο το σύστημα θα μπορούσε να θεωρηθεί συστατικό λογισμικού. Είναι πιθανό, επομένως να βρίσκεται ήδη διαθέσιμο για επαναχρησιμοποίηση σε κάποιο αποθετήριο κώδικα ΕΛ/ΛΑΚ. Ο μηχανικός επαναχρησιμοποίησης αναζητά το συστατικό αυτό και, αν η έρευνα είναι επιτυχής ένα ή περισσότερα αποτελέσματα επιστρέφονται. Στη συνέχεια, μετά από κατάλληλη προσαρμογή, η παράγωγη δουλειά πακετάρεται και είναι πλέον έτοιμη να παραδοθεί στον πελάτη. Σε αυτό το σημείο ο προσεκτικός αναγνώστης παρατηρεί ότι δεν έχει προταθεί κάποια πολιτική επιλογής ενός συστατικού μεταξύ των εναλλακτικών επιλογών (στην περίπτωση φυσικά που έχουν επιστραφεί περισσότερα από ένα κατάλληλα συστατικά λογισμικού). Η παραπάνω παρατήρηση, αν και σωστή δεν αποτελεί σε καμία περίπτωση παράλειψη. Σε αυτή τη φάση επιλέξαμε να προτείνουμε το μοντέλο στη βασική του μορφή επικεντρώνοντας αντίστοιχα στην βασική του λειτουργικότητα. Η αξιολόγηση των συστατικών λογισμικού παραλήφθηκε σκόπιμα, ώστε να αποτελέσει ερευνητικό αντικείμενο στο μέλλον.

Στην περίπτωση που ο μηχανικός λογισμικού έχει εξαλείψει την πιθανότητα να υπάρχει κάποιο συστατικό λογισμικού που να συγκεντρώνει την επιθυμητή λειτουργικότητα, προχωράει στην αποδόμηση του συστήματος υπό υλοποίηση σε επί μέρους συστατικά.

Σε αυτό το σημείο έχει ήδη αρχίσει να κατασκευάζει, ασυναίσθητα την δενδρική δομή που παρουσιάστηκε στο Κεφάλαιο 2.

Υπάρχει μια μικρή πιθανότητα η λειτουργικότητα του συστήματος υπό υλοποίηση να είναι υπερβολικά απλή για να διασπασθεί σε επί μέρους λειτουργίες, δηλαδή επί μέρους συστατικά. Σε αυτήν την περίπτωση το μοντέλο μας προτείνει ότι το σύστημα θα πρέπει να υλοποιηθεί από το μηδέν. Ένα ακόμη σενάριο που θα μπορούσε να οδηγήσει στην επιλογή υλοποίησης από το μηδέν θα μπορούσε να είναι η διαπίστωση ότι η διάσπαση του συστήματος υπό υλοποίηση και η έρευνα για επαναχρησιμοποιήσιμα συστατικά χρειάζεται περισσότερη δαπάνη χρόνου, από ότι η υλοποίηση του συστήματος από το μηδέν. Σε κάθε περίπτωση, μόλις η απόφαση για υλοποίηση από το μηδέν παρθεί, το υπό υλοποίηση σύστημα υλοποιείται, πακετάρεται και είναι έτοιμο να δοθεί στον πελάτη.

Τις περισσότερες φορές, το μέσο σύστημα υπό ανάπτυξη μπορεί να αποδομηθεί σε επί μέρους συστατικά. Σε αυτήν την περίπτωση ο μηχανικός επαναχρησιμοποίησης θα πρέπει να αναζητήσει τα συστατικά λογισμικού, ένα προς ένα. Στο μοντέλο διεργασίας μας, αυτή η ενέργεια αντικατοπτρίζεται στο σχήμα επιλογής με τίτλο «UNIMPLEMENTED COMPONENTS?». Ο ρόλος του συγκεκριμένου κόμβου επιλογής είναι δυικός. Αφενός εκκινεί τη διαδικασίας προσπάθειας να ανακαλυφθούν επαναχρησιμοποιήσιμα συστατικά λογισμικού για τις απαιτήσεις που δεν έχουν ακόμη υλοποιηθεί. Αφετέρου, αποτελεί τη συνθήκη τερματισμού για τον προαναφερθέντα βρόγχο και ουσιαστικά για όλη τη διαδικασία, αφού η λειτουργία του περιορίζεται στο να ελέγχει αν υπάρχουν συστατικά που δεν έχουν υλοποιηθεί ακόμη. Όταν όλη η

επιθυμητή λειτουργικότητα έχει υλοποιηθεί, το σύστημα υπό υλοποίηση θεωρείται έτοιμο για πακετάρισμα και παράδοση στον πελάτη.

Για κάθε συστατικό που δεν έχει υλοποιηθεί, μια υπό-διαδικασία ξεκινά με στόχο να ελεγχθεί αν μπορεί να βρεθεί επαναχρησιμοποιήσιμος κώδικας η θα πρέπει η εν λόγω απαίτηση να υλοποιηθεί από το μηδέν.

Όπως αναφέρθηκε και στο Κεφάλαιο 2, όταν αποδομούμε συστατικά σε απλούστερα, ερχόμαστε αντιμέτωποι με τον κίνδυνο να χαθούμε στη διαδικασία και εν τέλει να δαπανήσουμε περισσότερο χρόνο στο να ανακαλύψουμε επαναχρησιμοποιήσιμο κώδικα για ένα συστατικό από ότι θα χρειαζόμασταν ουσιαστικά για να το υλοποιήσουμε από το μηδέν. Με στόχο την αποφυγή τέτοιων φαινομένων, το μοντέλο της διεργασίας μας επιβάλει στον μηχανικό επαναχρησιμοποίησης, για κάθε ένα από τα συστατικά λογισμικού, να αναρωτηθεί πρώτα αν πραγματικά έχει νόημα να αντιμετωπιστεί με επαναχρησιμοποιήσιμο κώδικα. Εν γένει, υπάρχουν δύο σενάρια στα οποία, η έρευνα για επαναχρησιμοποιήσιμο κώδικα θα πρέπει να αποφεύγεται.

- Το συστατικό που θέλουμε να υλοποιήσουμε είναι πάρα πολύ συγκεκριμένο, επομένως μεγάλη ποσότητα χρόνου θα πρέπει να δαπανηθεί στην αναζήτηση με μεγάλη πιθανότητα να μην αποφέρει κανένα αποτέλεσμα.

- Η υλοποίηση του συστατικού είναι τετριμμένη· επομένως θα χρειαστεί λιγότερος χρόνος για να υλοποιηθεί από έναν έμπειρο προγραμματιστή, από ότι για έναν

μηχανικό επαναχρησιμοποίησης να εντοπίσει κατάλληλο κώδικα ώστε να καλύψει της ανάγκες της εν λόγω απαίτησης.

Εφόσον δε κριθεί σκόπιμο για το συγκεκριμένο συστατικό να γίνει αναζήτηση επαναχρησιμοποιήσιμων λύσεων, μένει μόνον μια λύση. Να υλοποιηθεί από το μηδέν. Αν προσέξουμε καλά τη σχηματική απεικόνιση του μοντέλου (βλ. Εικόνα 12) ωστόσο, θα διαπιστώσουμε ότι υπάρχει ένα διαφορετικό βήμα από την υλοποίηση από το μηδέν. Αναφέρεται ως «SEARCH FOR TRIVIAL CODE». Η επαναχρησιμοποίηση κώδικα συνέβαινε πολύ πριν εμφανιστούν τα οργανωμένα αποθετήρια κώδικα ΕΛ/ΛΑΚ. Η πρωταρχική ανάγκη για επαναχρησιμοποίηση κώδικα προέκυψε από την εμπειρική παρατήρηση ότι ορισμένα τμήματα κώδικα συναντώνται στις περισσότερες εφαρμογές λογισμικού. Η ανάγνωση από ή εγγραφή σε κάποιο αρχείο, η σύνδεση με μια βάση δεδομένων, η δημιουργία ενός Java Comparator, είναι απαιτήσεις που ένας προγραμματιστής συναντά σχεδόν σε κάθε έργο λογισμικού στο οποίο συμμετέχει. Η εμφάνιση αυτών των απαιτήσεων είναι μάλιστα τόσο συχνή ώστε δεν θα ήταν υπερβολή να χαρακτηρίσουμε την υλοποίησή τους ως *κώδικα βιβλιογραφίας*. Εφόσον αναζητούμε τέτοιου είδους κώδικα και με βάση πάντοτε το μοντέλο διεργασίας μας, ο μηχανικός επαναχρησιμοποίησης θα πρέπει να εκτελέσει αυτό που στο διάγραμμα ροής του μοντέλου μας αναφέρεται ως «SEARCH FOR TRIVIAL CODE». Με αυτόν τον τρόπο, ακόμη και όταν η επαναχρησιμοποίηση κώδικα καταλήξει σε αδιέξοδο, ο μηχανικός επαναχρησιμοποίησης μπορεί να είναι σίγουρος ότι έχει εξαλείψει κάθε δυνατό τρόπο επαναχρησιμοποίησης κώδικα. Αφότου όλα τα πιθανά κομμάτια επαναχρησιμοποιήσιμου «τετριμμένου» κώδικα έχουν συγκεντρωθεί, ο ωφέλιμος

κώδικας προσαρμόζεται στις ανάγκες του συστατικού προς υλοποίηση, το συστατικό θεωρείται υλοποιημένο και ο μηχανικός επαναχρησιμοποίησης μπορεί να προχωρήσει στην ανάπτυξη του επόμενο συστατικού. Φυσικά, αν όλες οι κατηγορίες αναζήτησης αποτύχουν, η λύση της υλοποίησης από το μηδέν είναι αναπόφευκτή.

Σαν τελευταία περίπτωση θα εξετάσουμε το σενάριο όπου το συστατικό λογισμικού υπό υλοποίηση δεν μπορεί να εντοπιστεί σε κάποιο αποθετήριο έργων ΕΛ/ΛΑΚ και ο μηχανικός επαναχρησιμοποίησης αποφασίζει να το αποδομήσει σε απλούστερα μικρότερης λειτουργικότητας. Άσχετα με το πόσο σύνθετο είναι το συστατικό λογισμικού το οποίο προσπαθούμε να κατασκευάσουμε, ο μηχανικός επαναχρησιμοποίησης πρέπει να σιγουρευτεί ότι δεν υπάρχει διαθέσιμο σε κάποιο αποθετήριο κώδικα ΕΛ/ΛΑΚ. Ως εκ τούτου, όπως φαίνεται και από το μοντέλο μας, θα πρέπει να εκτελέσει μια αναζήτηση για σχετικά επαναχρησιμοποιήσιμα συστατικά λογισμικού. Αφότου σιγουρευτεί ότι δεν υπάρχει κάτι έτοιμο, θα πρέπει να εξετάσει αν το προς υλοποίηση συστατικό μπορεί να διασπαστεί, ή όχι, σε απλούστερα συστατικά. Αν κάτι τέτοιο δεν είναι εφικτό, τότε συμπεραίνουμε ότι το συστατικό προς υλοποίηση είναι μέτριας ή μικρής πολυπλοκότητας και ως τέτοιο θα πρέπει να αντιμετωπιστεί με την λογική που παρουσιάσαμε λίγο πιο πάνω όταν μιλήσαμε για την «SEARCH FOR TRIVIAL CODE» διαδικασία αναζήτησης. Στην περίπτωση που το προς υλοποίηση συστατικό είναι αρκετά μεγάλο ώστε να μπορεί να διασπαστεί σε απλούστερα, τα νέα υπό-συστατικά θα πρέπει να αντιμετωπιστούν ως νέα, προς υλοποίηση συστατικά και η διαδικασία θα συνεχίσει κανονικά.

Το μοντέλο που παρουσιάζουμε σε αυτό το κεφάλαιο χαρακτηρίζεται ως ημι-αυτόματο γιατί, όπως ο αναγνώστης εύκολα καταλαβαίνει από την παραπάνω ανάλυση, απαιτεί τη διαρκή παρουσία του μηχανικού επαναχρησιμοποίησης. Υπενθυμίζουμε πως μηχανικός αναζήτησης μπορεί να θεωρηθεί ένας ειδικός στην τεχνολογία λογισμικού ο οποίος έχει εκπαιδευτεί στην ανάπτυξη λογισμικού με χρήση επαναχρησιμοποιήσιμου κώδικα. Απλοί προγραμματιστές μπορούν, κάτω από ειδικές περιπτώσεις, επίσης να επιφορτιστούν με αυτόν τον ρόλο. Σε πιο οργανωμένα περιβάλλοντα ο ρόλος του μηχανικού λογισμικού μπορεί να ανατεθεί σε άτομα, ως αυτοτελής αρμοδιότητα, στα πλαίσια της ομάδας ανάπτυξης.

Κεφάλαιο 4

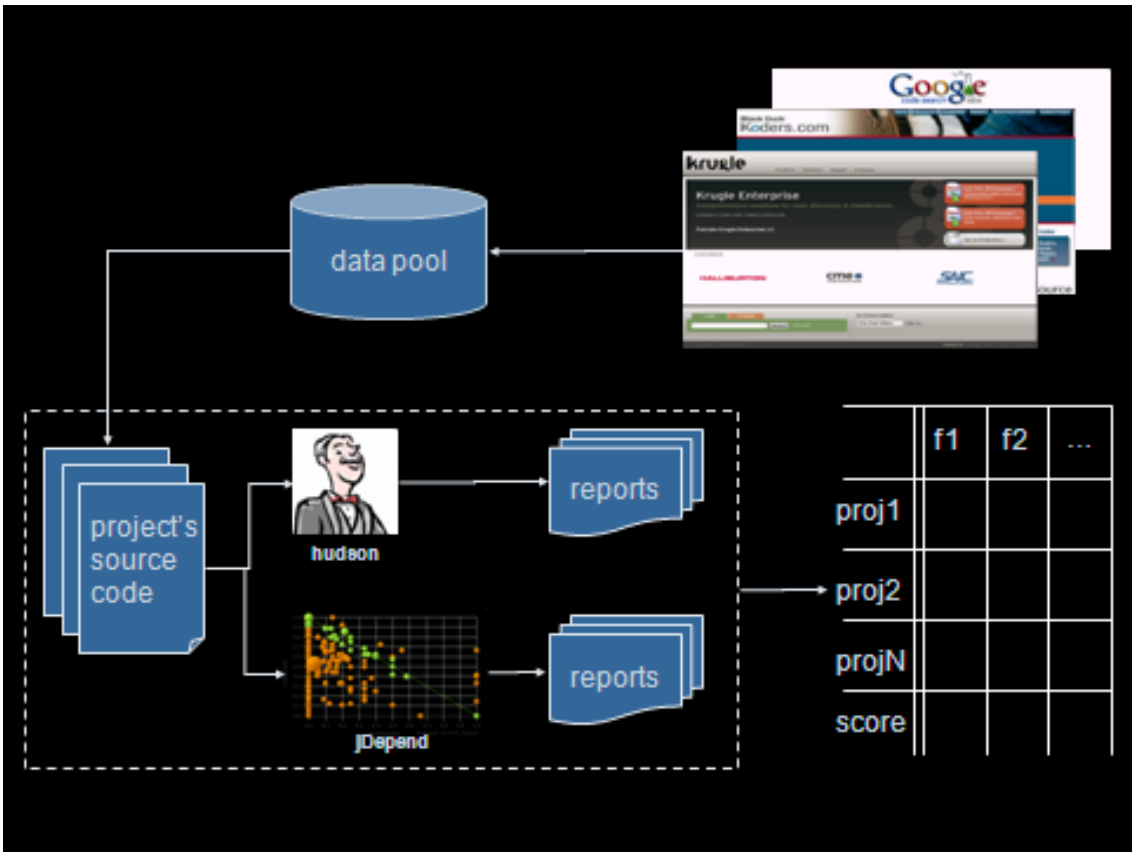
ΣΧΕΔΙΑΣΜΟΣ ΠΛΗΡΟΦΟΡΙΑΚΟΥ ΣΥΣΤΗΜΑΤΟΣ ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗΣ ΚΩΔΙΚΑ ΕΛ/ΛΑΚ ΣΥΝΔΥΑΖΟΝΤΑΣ ΣΧΕΤΙΚΑ ΕΡΓΑΛΕΙΑ

Στο κεφάλαιο αυτό γίνεται μια προσπάθεια σχεδιασμού ενός πληροφοριακού συστήματος το οποίο θα υλοποιήσει, στο βαθμό του δυνατού, τη λειτουργικότητα του μοντέλου διεργασίας επαναχρησιμοποίησης κώδικα ΕΛ/ΛΑΚ που παρουσιάστηκε αναλυτικά στο Κεφάλαιο 3.

Όπως θα διαπιστώσετε, ο σχεδιασμός του λογισμικού κάνει χρήση ήδη υπαρχόντων πληροφοριακών συστημάτων και τεχνολογιών αναλαμβάνοντας κατά κάποιον τρόπο τον ρόλο της ενορχήστρωσής τους ώστε να αυτοματοποιήσει, στο βαθμό που αυτό είναι εφικτό, την διαδικασία επαναχρησιμοποίησης κώδικα, περιορίζοντας τον ρόλο του μηχανικού επαναχρησιμοποίησης στην προσαρμογή, σύνδεση και έλεγχο των επί μέρους συστατικών λογισμικού και το πακετάρισμα του τελικού προϊόντος ώστε να παραδοθεί στον πελάτη.

Θα πρέπει να τονίσουμε σε αυτό το σημείο ότι το σύστημα αυτό κάθε αυτό δεν έχει υλοποιηθεί (λόγω περιορισμένου χρόνου στα πλαίσια της μεταπτυχιακής διατριβής). Με τον σχεδιασμό, ωστόσο, που παραθέτουμε παρακάτω, έχουν τεθεί οι βάσεις ώστε η υλοποίησή του να επιχειρηθεί στα πλαίσια μελλοντικού ερευνητικού έργου. Η υλοποίηση ενός τέτοιου συστήματος θα μπορούσε να αποτελέσει πλατφόρμα

πειραματισμού για θέματα τεχνολογίας λογισμικού, επαναχρησιμοποίηση κώδικα αλλά και ΕΛ/ΛΑΚ έργων λογισμικού.



Εικόνα 13: Αρχιτεκτονική πληροφοριακού συστήματος επαναχρησιμοποίησης κώδικα ΕΛ/ΛΑΚ

Όπως εύκολα μπορεί να αντιληφθεί κανείς κοιτάζοντας την προτεινόμενη αρχιτεκτονική του συστήματος που παρουσιάζεται στην Εικόνα 13. δεν είναι τίποτε άλλο παρά μια ενορχήστρωση ήδη υπαρχόντων συστημάτων και τεχνολογιών.

Πιο αναλυτικά, ξεκινώντας από επάνω δεξιά και ακολουθώντας τη φορά των βελών, ξεκινούμε με ένα σύνολο αποθετηρίων έργων ΕΛ/ΛΑΚ, μηχανών αναζήτησης κώδικα, κλπ. Όλες αυτές οι πηγές τροφοδοτούν μία βάση έργων επαναχρησιμοποιήσιμου κώδικα

από τον οποίο το σύστημά μας θα προσπαθήσει να εξάγει και στη συνέχεια να προτείνει κατάλληλα συστατικά λογισμικού με βάσει τις εκάστοτε απαιτήσεις.

Στο περιγεγραμμένο με διακεκομμένη γραμμή πλαίσιο που φαίνεται παρακάτω, περιλαμβάνεται ο μηχανισμός αξιολόγησης των συστατικών λογισμικού που ανακαλύφθηκαν. Κάθε υποψήφιο συστατικό λογισμικού ελέγχεται από δύο συστήματα, τα Hudson και JDepend (βλ. παρακάτω για αναλυτική περιγραφή των συστημάτων αυτών). Το Hudson αποτελεί μια πλατφόρμα συνεργατικής ανάπτυξης κώδικα που αναλαμβάνει τον έλεγχο της ορθής λειτουργίας του συστήματος, ανακάλυψη πιθανών σφαλμάτων ή κακών πρακτικών προγραμματισμού, unit testing, κλπ. Το JDepend, αν και υποστηρίζει μόνον εφαρμογές γραμμένες σε Java αποτελεί ένα πολύ καλό εργαλείο μέτρησης της ποιότητας, σε επίπεδο σχεδιασμού του συστήματος υπό ανάπτυξη. Κάθε ένα από τα δύο αυτά συστήματα παράγει από μία αναφορά για κάθε ένα από τα υποψήφια συστατικά. Αφότου η αξιολόγηση όλων των υποψηφίων συστατικών για μία απαίτηση έχει ολοκληρωθεί, μία ενιαία αναφορά υπό μορφή πίνακα δημιουργείται ώστε να μπορέσει ο μηχανικός επαναχρησιμοποίησης να επιλέξει εύκολα και γρήγορα το συστατικό λογισμικού που τελικά θα χρησιμοποιηθεί στο τελικό σύστημα.

Όπως είπαμε η λογική του συστήματος είναι εξαιρετικά απλή και βασίζεται στην οργάνωση ήδη υπαρχόντων συστημάτων προσανατολισμένων στην αναζήτηση και διαχείριση κώδικα ανοικτού λογισμικού. Παρακάτω μπορείτε να βρείτε μια αναλυτική περιγραφή των συστημάτων που κρίνουμε ότι πρέπει να χρησιμοποιηθούν για να μπορέσει η παραπάνω αρχιτεκτονική να υλοποιηθεί.

Σχετικά εργαλεία και υπηρεσίες

Σε αυτόν τον τομέα μπορείτε να βρείτε μια αναλυτική περιγραφή των εργαλείων που θα πρέπει να χρησιμοποιηθούν για την υλοποίηση του συστήματος που περιγράφηκε στην αρχή του τρέχοντος κεφαλαίου.

Αναζήτηση επαναχρησιμοποιήσιμου κώδικα:

Τα εργαλεία αυτής της κατηγορίας χωρίζονται σε τρεις κατηγορίες:

Α. Αποθετήρια έργων ΕΛ/ΛΑΚ

Οι υπηρεσίες αυτού του τύπου φιλοξενούν ολόκληρα έργα ΕΛ/ΛΑΚ και επιτρέπουν την αναζήτηση με βάση την γλώσσα προγραμματισμού, τον τύπο άδειας, κλπ. Τις περισσότερες φορές παρέχεται δυνατότητα χρήσης των υπηρεσιών τους από άλλες εφαρμογές μέσω Application Programming Interfaces (APIs). Ας δούμε μερικές από τις αντιπροσωπευτικότερες:

Sourceforge: Ίσως το πιο γνωστό αποθετήριο έργων ΕΛ/ΛΑΚ. Φιλοξενεί χιλιάδες έργα ΕΛ/ΛΑΚ και συνεχώς ανανεώνεται με νεότερα. Επίσης αποτελεί μια τεράστια κοινότητα προγραμματιστών οι οποίοι απασχολούνται στα διάφορα έργα με διάφορους ρόλους όπως αυτόν του προγραμματιστή, ελεγκτή σφαλμάτων, κλπ.

Google code: Αποτελεί αποθετήριο κώδικα και μηχανή αναζήτησης κώδικα ταυτόχρονα. Όντας η επίσημη υπηρεσία της Google προσανατολισμένη σε κώδικα, μεταφέρει μαζί της μία τεράστια κοινότητα προγραμματιστών και έργων ΕΛ/ΛΑΚ.

Εικόνα 14: Η κεντρική οθόνη της υπηρεσίας Google Code

Β. Μηχανές αναζήτησης πηγαίου κώδικα

Οι μηχανές αναζήτησης πηγαίου κώδικα είναι προσανατολισμένες στην εύρεση κώδικα μικρής κλίμακας, τάξεως κλάσης. Συνήθως επιτρέπουν την αναζήτηση με βάση τη γλώσσα προγραμματισμού, την αδειοδότηση αλλά και την εμβέλεια κώδικα (κλάση, μέθοδο, σχόλια, ακόμη και documentation). Ας δούμε μερικές από τις αντιπροσωπευτικότερες:

Krugle.org:

Το Krugle ακολουθεί τη λογική του Google αλλά για αναζήτηση πηγαίου κώδικα. Επιτρέπει αναζήτηση σε επίπεδο έργου, κλάσης, γλώσσας προγραμματισμού και εμβέλειας. Ιδιαίτερα για την εμβέλεια παρέχει μια σειρά από ενδιαφέρουσες επιμέρους δυνατότητες:

- Αναζήτηση σε σχόλια
- Αναζήτηση σε πηγαίο κώδικα
- Αναζήτηση σε λοιπά έγγραφα (που περιλαμβάνονται στο έργο)
- Αναζήτηση σε δηλώσεις κλάσεων
- Αναζήτηση σε δηλώσεις μεθόδων / συναρτήσεων
- Αναζήτηση σε κλήσεις μεθόδων / συναρτήσεων

Εικόνα 15: Η κεντρική οθόνη της υπηρεσίας Krugle

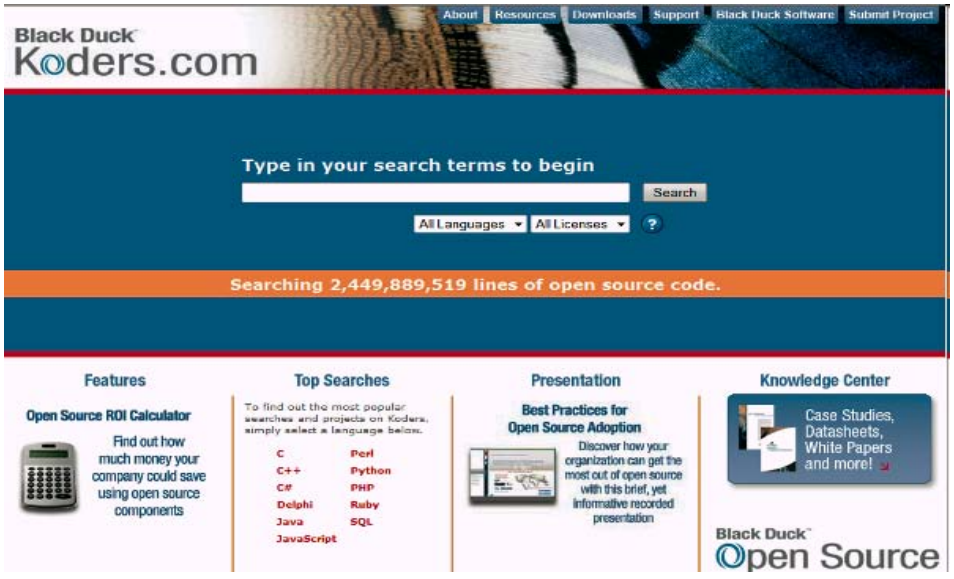
Το Krugle υποστηρίζει συνεργασία με όλους τους γνωστούς browsers αλλά και το eclipse με ειδικά plug-ins ενώ προσφέρει δυνατότητα σύνδεσης μέσω διεπαφής προγραμματισμού εφαρμογής (API), αλλά δυστυχώς επί πληρωμή.

Koders.com:

Επιτρέπει αναζήτηση με βάση τη γλώσσα προγραμματισμού, την αδειοδότηση (περιλαμβάνει περίπου 30 διαφορετικές άδειες). Ενδιαφέρον παρουσιάζει το γεγονός ότι εμφανίζει στατιστικά μαζί για κάθε αναζήτηση. Για παράδειγμα:

Η αναζήτηση με λέξη κλειδί apache επέστρεψε:

- 11370 Interface Definitions
- 74267 Class Definitions
- 183 Method Definitions

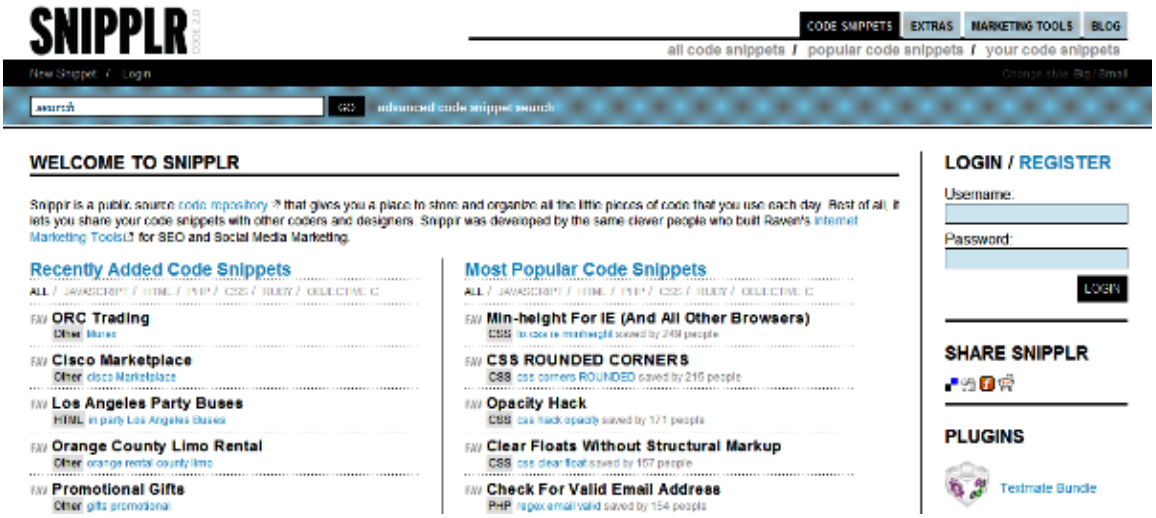


Εικόνα 16: Η κεντρική οθόνη της υπηρεσίας Koders

Γ. Υπηρεσίες κοινωνικής δικτύωσης προσανατολισμένες σε κώδικα

Snipplr.com

Πρόκειται για μια υπηρεσία κοινωνικής δικτύωσης προσανατολισμένη σε πηγαίο κώδικα. Φιλοξενεί κομμάτια κώδικα από διαφορετικές γλώσσες προγραμματισμού που επιτελούν πολύ συγκεκριμένες διαδικασίες (πολλές φορές τετριμμένες). Η πρόσβαση στις υπηρεσίες του snipplr μπορεί να γίνει και μέσω διεπαφής προγραμματισμού εφαρμογής (API).



Εικόνα 17: Η κεντρική οθόνη της υπηρεσίας Snipplr.com

Έλεγχος επαναχρησιμοποιήσιμου κώδικα:

Εικόνα 18: Η επίσημη ιστοσελίδα του Hudson

Hudson:

Πρόκειται για ένα σύστημα συνεργατικής ανάπτυξης κώδικα και σύστημα ελέγχου των έργων που φιλοξενεί. Περιλαμβάνει μια σειρά από πολύ ενδιαφέροντα χαρακτηριστικά:

- Βασικοί άξονες λειτουργίας του Hudson
  - Ανάπτυξη / δοκιμή έργου συνεχώς και παράλληλα
  - Παρακολούθηση / επικοινωνία σχετικά με εκτελέσεις διεργασιών εκτός λογισμικού

        - Cron jobs / procmail jobs για αποστολή ενημερωτικών μηνυμάτων προόδου
        - Πολλαπλοί τρόποι ειδοποίησης σε περίπτωση αποτυχίας

- Υποστήριξη ποικίλων:
    - Περιβαλλόντων ανάπτυξης
    - JDKs (εκδόσεων Java)
    - Βάσεων δεδομένων
    - Λειτουργικών συστημάτων

- Αναφορές δοκιμών:
    - JUnit
    - TestNG

Το Hudson επίσης ακολουθείται από μια τεράστια κοινότητα προγραμματιστών ενώ υποστηρίζεται από εκατοντάδες πρόσθετα που επεκτείνουν τις δυνατότητές του.

Έλεγχος ποιότητας:

JDepend:

Το JDepend αποτελεί ένα σύστημα ποιοτικού ελέγχου του πηγαίου κώδικα. Εξετάζει την ποιότητα του κώδικα σε θέματα επεκτασιμότητας, επαναχρησιμοποίησης και συντηρησιμότητας. Η λειτουργία του βασίζεται στον έλεγχο των συνδέσεων μεταξύ των κλάσεων του έργου.

Κεφάλαιο 5

ΣΧΕΤΙΚΟ ΕΡΕΥΝΗΤΙΚΟ ΕΡΓΟ

Στο [Crnkovic06] παρουσιάζεται μια παραλλαγή του μοντέλου καταρράκτη ώστε να λαμβάνει υπόψη την επαναχρησιμοποίηση συστατικών λογισμικού. Παρουσιάζονται δύο ειδών διεργασίες. Η μία αφορά στην ανάπτυξη συστατικών επαναχρησιμοποιήσιμου κώδικα και η άλλη στην ανάπτυξη συστημάτων κάνοντας χρήση επαναχρησιμοποιήσιμων συστατικών λογισμικού. Οι συγγραφείς αναλύουν λεπτομερώς τις αλλαγές που πρέπει να συμβούν στις διαδικασίες του μοντέλου καταρράκτη ώστε να καταστεί εφικτή η ανάπτυξη συστημάτων με επαναχρησιμοποίηση συστατικών λογισμικού. Για να μπορέσει να υπάρξει συνοχή μεταξύ των δυο διεργασιών που αναφέρθηκαν παραπάνω οι συγγραφείς εισάγουν μία νέα διεργασία, αυτήν του «Αποτίμησης Συστατικών λογισμικού» η οποία θα πρέπει να διενεργείται κατά το δυνατόν ανεξάρτητα από τη διαδικασία ανάπτυξης του συστήματος. Εν γένει μια διαδικασία αποτίμησης περιλαμβάνει τα ακόλουθα:

- Ανακάλυψη των συστατικών λογισμικού
- Επιλογή των συστατικών λογισμικού με βάση την καταλληλότητά τους ως προς τρέχοντα ή μελλοντικά προϊόντα
- Επαλήθευση συστατικών λογισμικού
- Αποθήκευση των συστατικών λογισμικού και μετα-δεδομένων μελλοντική αναφορά

Το μοντέλο που προτείνουμε στην παρούσα μεταπτυχιακή διατριβή θα μπορούσε να χρησιμοποιηθεί από μηχανικούς λογισμικού για να υποβοηθήσει την ανακάλυψη και επιλογή συστατικών λογισμικού με έναν πιο συστηματικό τρόπο όταν πρόκειται για επαναχρησιμοποίηση κώδικα από έργα ΕΛ/ΛΑΚ.

Στο [Hummel07] προτείνεται μια διεργασία με ονομασία *«Extreme Harvesting»* που χρησιμοποιεί unit tests που έχουν υλοποιηθεί στα πλαίσια μια διαδικασίας ανάπτυξης λογισμικού με ευέλικτες μεθοδολογίες (π.χ. Ακραίος Προγραμματισμός) σαν κριτήριο αναζήτησης για επαναχρησιμοποιήσιμα συστατικά λογισμικού. Υπάρχουν δύο λογικές στη διαδικασία Extreme Harvesting. Η definitive harvesting και η speculative harvesting. Στην περίπτωση της speculative harvesting τα επαναχρησιμοποιήσιμα συστατικά λογισμικού πλησιάζουν πολύ αυτό που αναζητούσε ο προγραμματιστής αλλά δεν ταιριάζουν απόλυτα στις προσδοκίες του. Οι προγραμματιστές επομένως χρειάζεται να δαπανήσουν σημαντικό χρόνος την προσαρμογή και χρήση των επαναχρησιμοποιήσιμων συστατικών λογισμικού που έχουν ανακαλύψει. Η διαδικασία επομένως δεν είναι απόλυτα αυτοματοποιημένη. Μπορεί ωστόσο να υποστηριχθεί, από ένα ειδικά δημιουργημένο πρόσθετο για το Eclipse.

Ένα ακόμη εργαλείο το οποίο προτείνεται στο [McCarey05] είναι το *Rascal,* ένας ευφυής πράκτορας, που επιθεωρεί την υλοποίηση κώδικα από το μηδέν και χρησιμοποιεί τεχνικές τεχνητή νοημοσύνη για να ταιριάζει επαναχρησιμοποιήσιμο κώδικα που βρίσκεται σε αποθετήρια έργων ΕΛ/ΛΑΚ με ήδη υλοποιημένο κώδικα. Αυτό το εργαλείο στοχεύει σε μια πιο αυτοματοποιημένη λύση από ότι η πρόταση του

[Hummel07] αφού η διαδικασία αναζήτησης πυροδοτείται από τον ευφυή πράκτορα και τα συστατικά λογισμικού που ανακαλύπτονται παρουσιάζονται σε αυτόν που διενεργεί επαναχρησιμοποίηση κώδικα χωρίς ο ίδιος να εκτελέσει κάποια ενέργεια αναζήτησης. Τελικά, βέβαια, ο χρήστης είναι επιφορτισμένος με την επιλογή του / των κατάλληλου / κατάλληλων συστατικών λογισμικού από τις εναλλακτικές που του επέστρεψε ο ευφυής πράκτορας και τον σωστό συνδυασμό τους ώστε να υλοποιηθεί το τελικό σύστημα.

Σε σχέση με τις δουλειές των [Hummel07] και [McCarey05] η δική μας στοχεύει στην κατανόηση της διαδικασίας της επαναχρησιμοποίησης κώδικα σε επίπεδο ανθρώπινης αντίληψης σε πρώτη φάση και στην συνέχεια στην δημιουργία κατάλληλων εργαλείων για την υποστήριξη μιας τέτοιας διαδικασίας. Αν και κανείς δεν μπορεί να αμφισβητήσει την χρησιμότητα εργαλείων όπως τα δύο που περιγράψαμε προηγουμένως, η δική μας προσέγγιση επικεντρώνεται στην διαδικασία επαναχρησιμοποίησης αυτή κάθε αυτή, με την ελπίδα ότι θα καταφέρουμε να κατανοήσουμε καλύτερα τη φύση των προβλημάτων της εν λόγω ερευνητικής περιοχής. Πιστεύουμε πως το να κατανοήσουμε σε μεγαλύτερο βαθμό τέτοιου είδους προβλήματα αποτελεί προαπαιτούμενο για την δημιουργία αποτελεσματικότερων εργαλείων υποστήριξης του χώρου.

Εκτός της αναζήτησης και ανάκτησης επαναχρησιμοποιήσιμων συστατικών λογισμικού, που αποτελεί το βασικό πεδίο της ερευνητικής μας δραστηριότητας, υπάρχουν και άλλα πολύ σημαντικά ζητήματα στην επαναχρησιμοποίηση κώδικα ως κομμάτι της τεχνολογίας λογισμικού γενικά αλλά και στην επαναχρησιμοποίηση συστατικών λογισμικού ΕΛ/ΛΑΚ, συγκεκριμένα.

Τέτοια ζητήματα περιλαμβάνουν θέματα αδειοδότησης και ποιότητας. Έχει παρατηρηθεί μάλιστα ιδιαίτερη δραστηριότητα ως προς αυτά τα ζητήματα του χώρου επαναχρησιμοποίησης κώδικα. Για παράδειγμα το έργο FOSSology [Gobeille08] θεωρείται το καλύτερο αυτή τη στιγμή στην αναζήτηση αδειοδότησης για έργα ΕΛ/ΛΑΚ, παράγοντας πολύ σημαντικός ιδιαίτερα στις εταιρίες που ενδιαφέρονται να εκμεταλλευτούν εμπορικά τον ανοικτό κώδικα [Madanmohan04]. Έργα όπως το SQO-OSS [Samoladas08] στοχεύουν στο να προσφέρουν πληροφορία για επαναχρησιμοποιήσιμα έργα λογισμικού που αφορά σε θέματα ποιότητας.

Κεφάλαιο 6

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΟ ΕΡΓΟ

Στην παρούσα μεταπτυχιακή διατριβή μελετήσαμε τον ρόλο της επαναχρησιμοποίησης κώδικα ως τμήμα της ανάπτυξης ενός έργου λογισμικού. Προσπαθήσαμε να παρουσιάσουμε, υπό μορφή περίπτωσης χρήσης την προσέγγιση του μηχανικού επαναχρησιμοποίησης στην ανάπτυξη λογισμικού κάνοντας χρήση εννοιών προσανατολισμένων σε υπολογιστικά συστήματα. Στη συνέχεια προσπαθήσαμε να μοντελοποιήσουμε την παραπάνω συμπεριφορά και καταλήξαμε στο να προτείνουμε ένα ημι-αυτόματο μοντέλο επαναχρησιμοποίησης έργων ΕΛ/ΛΑΚ. Το μοντέλο αυτό στοχεύει στο να παρουσιάσει την κουλτούρα ενός μηχανικού επαναχρησιμοποίησης, την διάθεσή του δηλαδή να χρησιμοποιήσει όσο περισσότερο έτοιμο κώδικα μπορεί και να υλοποιήσει τον ελάχιστο δυνατό. Τέλος βασισμένοι στη γνώση που αποκτήσαμε από όλη τη μελέτη, προτείνουμε τον σχεδιασμό ενός εργαλείου αναζήτησης και αξιολόγησης συστατικών λογισμικού σε αποθετήρια έργων ΕΛ/ΛΑΚ το οποίο θα μπορούσε να αποτελέσει εργαλείο πειραματισμού για το μελλοντικό ερευνητικό μας έργο.

Όπως αναφέραμε και πριν, το μοντέλο που προτείνουμε στην παρούσα εργασία αποτελεί μία πρώτη προσπάθεια να καταγραφεί μία καλά ορισμένη διαδικασία επαναχρησιμοποίησης κώδικα. Προς το παρόν η παρουσία ενός ειδικού είναι επιτακτική για να λειτουργήσει το μοντέλο διαδικασίας. Ο ειδικός αυτός καλείται να πάρει καίριες αποφάσεις που αφορούν στην επιλογή του καλύτερου συστατικού λογισμικού ανάμεσα σε διαφορετικές εναλλακτικές, στο αν κάποια απαίτηση θα πρέπει να αποδομηθεί σε

απλούστερες ή τι είδους προσαρμογή χρειάζονται τα επιλεγμένα συστατικά λογισμικού ώστε να ενσωματωθούν χωρίς πρόβλημα στο υπό ανάπτυξη σύστημά μας.

Σαν μελλοντικό ερευνητικό έργο θα μας ενδιέφερε να μελετήσουμε τη δυνατότητα να προτείνουμε ένα τελείως αυτοματοποιημένο μοντέλο διαδικασίας επαναχρησιμοποίησης κώδικα το οποίο θα μπορεί να μπορεί να λαμβάνει τετριμμένες ωστόσο πολύ σημαντικές αποφάσεις όπως ποιο από τα προτεινόμενα συστατικά λογισμικού είναι το καλύτερο βάσει καλά ορισμένων μέτρων. Μια άλλη ενδιαφέρουσα πρόκληση θα ήταν να προσπαθήσουμε να μετρήσουμε την *συμβατότητα (fitness)* ενός συστατικού λογισμικού με το προς υλοποίηση σύστημα. Με τον όρο συμβατότητα αναφερόμαστε στην ομοιότητα που μπορεί να εμφανίζει ένα συστατικό όσον αφορά το στυλ προγραμματισμού, τα πρότυπα ανάπτυξης, την ποιότητα κλπ.

ΑΝΑΦΟΡΕΣ

[Etzkorn97]
Etzkorn, L. H. and Davis, C. G. 1997. Automatically Identifying Reusable OO Legacy Code. Computer 30, 10 (Oct. 1997), 66-71. DOI= http://dx.doi.org/10.1109/2.625311.

[Frakes95]
Frakes, W. B. and Fox, C. J. 1995. Sixteen questions about software reuse. *Commun. ACM* 38, 6 (Jun. 1995), 75-ff. DOI= http://doi.acm.org/10.1145/203241.203260

[Crnkovic06]
Crnkovic, I., Chaudron, M., and Larsson, S. 2006. Component-Based Development Process and Component Lifecycle. In *Proceedings of the international Conference on Software Engineering Advances* (October 29 - November 03, 2006). ICSEA. IEEE Computer Society, Washington, DC, 44. DOI= http://dx.doi.org/10.1109/ICSEA.2006.28

[Hummel07]
Oliver Hummel and Colin Atkinson: “Supporting Agile Reuse Through Extreme Harvesting”, in proc. of the 8th International XP Conference, pp. 28-37, Springer, 2007

[McCarey05]
Frank McCarey, Mel Ó Cinnéide and Nicholas Kushmerick: “Rascal: A Recommender Agent for Agile Reuse", Artificial Intelligence Review, vol. 24, no. 3-4, pp. 253-276, Springer, November 2005

[Gobeille08]
R. Gobeille: “The FOSSology project”, In Proceedings of the 2008 international Working Conference on Mining Software Repositories (MSR '08), pp. 47-50, ACM, 2008

[Madanmohan04]
T.R. Madanmohan and R. De’, “Open Source Reuse in Commercial Firms”, IEEE Software, vol. 21, Dec. 2004, pp. 62-69

[Samoladas08]
I. Samoladas, G. Gousios, D. Spinellis and I. Stamelos: “The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation”, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, pp. 237-248, Springer, 2008

[Bassett95]
Bassett P.G. (1995). To make or to buy? There is a third alternative. American Programmer. November.

[Chen94]
Chen D.J. and Chen D.T.K. (1994). An experimental study of using reusable software design frameworks to achieve software reuse. *Journal of Object-Oriented Programming, 7(2),* 56-66

[Davis94]
Davis T. (1994b). Adopting a policy of reuse. *IEEE Spectrum,* June, pp 44-8

[Swreuse]
Jacobson I., Griss M., Jonsson P. (1997). Software Reuse. Architecture, process and Organization for Bussiness Success. ISBN 0-201-92476-5.

ΠΑΡΑΡΤΗΜΑ Α

Code Reuse Definitions

- Reusable code is code that can be used, without modification, to perform a specific service regardless of what application uses the code.

  ~MSDN Library

- Code reuse is the idea that a partial or complete computer program written at one time can be, should be, or is being used in another program written at a later time

  ~Wikipedia

- Software reuse is the use of existing software knowledge or artifacts to build new software artifacts

  ~W. B. Frakes & C. J. Fox **[Frakes95]**

ΠΑΡΑΡΤΗΜΑ Β

Μελέτη Περίπτωσης #1

(Πόροι)

Παρακάτω παρατίθενται το πρωτότυπο άρθρο, όπως εμφανίζεται στην ιστοσελίδα που μας παρέπεμψε η Google στο δεύτερο βήμα της μελέτης περίπτωσης #1 (βλ. ενότητα 2.1). Η επισύναψη του άρθρου έχει σαν στόχο να τεκμηριώσει την αποτελεσματικότητα της διαδικασίας επαναχρησιμοποίησης κώδικα αποδεικνύοντας ότι οδήγησε σε μία λύση που:

- Συνάδει απόλυτα με τις τεχνικές απαιτήσεις (συστατικό εισόδου υλοποιημένο σε μορφή Java Bean).
- Είναι τεκμηριωμένη.
- Είναι ποιοτική.

Στη συνέχεια του παραρτήματος μπορείτε να βρείτε τα σημεία ενδιαφέροντος σημασμένα με κίτρινο.

Site User Logon with XML, Java Beans and JSP

by Donovan Cowie

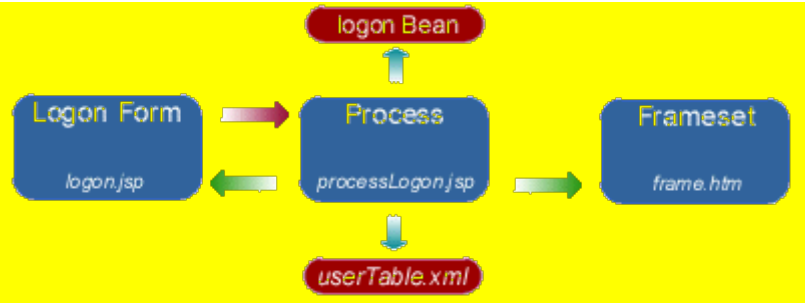
October 3, 2000

By asking a user for a logon and password, you can verify the user's access to the site and ensure they are shown only the information they need. During this tutorial, we will use

XML and Java Server Pages to verify the user's logon - and then store the results in a session Java Bean for easy access.

Overview:

- The visitor enters his or her userID and password into an HTML form.
- processLogon.jsp opens userTable.xml and searches for the visitor's details. If a match is found, the visitor is allowed through to the rest of the site. Their userID is stored in a session Bean to be accessed by other pages. The Bean also keeps track of the fact that the user has been through a security check successfully. If no match is found, the visitor is routed back to the logon form.



Environment:

The application discussed in this tutorial was tested on Windows NT, using Apache's Tomcat server. It has also been tested on Java Web Server 2.0 running on Windows 98. You can download both - Tomcat from the Apache site - and an evaluation version of

Java Web Server from the Sun site.

The XML parser I used was the Java API for XML Parsing (JAXP). If you're new to XML parsing, I recommend you download JAXP from the Sun site. JAXP has quite a nifty install - even adding the classes to your classpath during the install. I also recommend browsing through the documentation that comes bundled with JAXP. The JAXP tutorial will provide you with quite a thorough background for your projects in XML.

Logon Form:

The Logon page is the entrance to your site, or a part of your site that you only want a special type of user to view. Examples of this can be seen on e-commerce sites where visitors access their personal accounts, developer areas (like the Sun or Nokia sites), or application sites with personalized user application areas.

```
<form method="POST" action="processLogon.jsp" name="logon">

<table border="0" cellpadding="0" cellspacing="0" width="200">
<tr>
    <td width="50%">Name</td>
    <td width="50"><input type="text" name="userID"></td>
```

```
</tr>
<tr>
    <td width="50%">Password</td>
    <td width="50%"><input type="password" name="pwd"></td>
</tr>
<tr>
    <td width="50%" colspan="2" align="center">
    <input type="submit" value="submit">
    </td>
</tr>
</table>
</center>
</form>
```

This is pretty straight HTML stuff. The page contains a form, which in turn contains the input fields for the userID and pwd parameters.

I included the logon form in a JSP page, so that Tomcat knows to compile and cache the page. The page will work just fine as an HTML page as well.

When the visitor hits submit on the form, the userID and pwd parameters are submitted to processLogon.jsp for further processing.

userTable.xml:

Before going on to **processLogon.jsp**, let's take a quick look at the contents of the XML file that we will be reading from:

```
<?xml version="1.0" encoding="UTF-8"?>
<userTable>
<user userID="Joe" pwd="Soap" profession="researcher" level="user"> Joe </user>
<user userID="Dude" pwd="Dude" profession="developer" level="user"> Some Dude
</user>
<user userID="James" pwd="sunny skies" profession="sales" level="sales admin">
McCarthy </user>
</userTable>
```

The document makes use of a quick-and-easy schema for storing data. The root element is userTable and the only child elements off it are user.

processLogon.jsp:

processLogon is the page responsible for doing most of this part of the application's thinking. It accepts the userID and pwd parameters and then reads through an XML file for the user's details. If the user's record is not found, they are redirected back to the logon page. If the user is found, their security settings are set in the session Bean and they are forwarded to the next page.

Page Directives

The imports I've used cover the tasks of opening and parsing the XML document.

```
<%@ page language="java" %>
<%@ page import="javax.xml.parsers.*" %>
<%@ page import="org.w3c.dom.Document" %>
<%@ page import="org.w3c.dom.Element" %>
<%@ page import="org.w3c.dom.DOMException" %>
<%@ page import="java.net.URL" %>
<%@ page import="java.io.InputStream" %>
<%@ page import="java.io.IOException" %>
```

Page Variables

The first set of variables keep track of constants about the server environment, such as the file root for the application and URLs for the data. Note that on a site where security is a consideration, you might want to keep these elsewhere, possibly in a Bean, or an included file.

```
<%! String fileRoot = "e:/www/testpro/";%>
```

<%! String root = "http://fugazi/testpro/";%>

<%! String dataRoot = "http://fugazi/testpro/data/";%>

<%! String fileName= dataRoot + "userTable.xml";%>

The logon Bean is instantiated at this point. This Bean will be responsible for keeping track of information about, and for, the visitor.

<jsp:useBean id="logon" scope="session" class="logonBean.logon" />

<%! String msg="Parse Successful!";%>

<%! String nodeValue="No Value";%>

<%! String userID="";%>

<%! String pwd = "";%>

<%! String routeURL="";%>

<%! int listLength;%>

HTML Header

<html>

<head>

<title>processLogon</title>

</head>

<body>

Get Page Parameters

The first bit of business we need to attend to is the retrieval of page parameters that were sent from the logon form. Note that I check each of the parameters to make sure that 'userID' and 'pwd' are not assigned null values if no values are sent from the user form.

```
<%
//Get page parameters:
if(request.getParameter("userID") != null)
{
    userID = request.getParameter("userID");
}
if(request.getParameter("pwd") != null)
{
    pwd = request.getParameter("pwd");
}
```

Declare and Initialize Variables

Some of our most important variables are initialized at this point. The document object is declared, and a DocumentBuilderFactory is instantiated. The DocumentBuilderFactory

will later be used to create a DocumentBuilder, which in turn will parse the XML file and build a Document object, which we will store in the **document** object.

```
//Declare variables
Document document;
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

The msg variable is used for debugging. routeURL is used at the end of the page in a JavaScript rerouting script. The default is 'logon.jsp'. The visitor will be rerouted to the logon page if he or she is not found.

If the user is found, routeURL will be changed to contain 'frame.htm' - or in fact any secure page that you would like the user to view after having logged on.

```
//Initialize variables:
msg="Parse Successful!";
routeURL = "logon.jsp";
```

Read the XML file

The next section of code handles the opening of the file, the parsing of the XML document, and the generation of a set of user nodes. The NodeList can be accessed in your JSP page pretty much like a record set object. Later we will be looping through it,

searching for the current user's record.

```
try {
//Open the file for reading:
URL u = new URL(fileName);
InputStream inputXML = u.openStream();

//Build document:
DocumentBuilder builder = factory.newDocumentBuilder();
document = builder.parse(inputXML);

//Generate the NodeList;
org.w3c.dom.NodeList nodeList = document.getElementsByTagName("user");
```

Search for User's Record

I've used a loop to get hold of the user record. Each time the loop iterates, a new Node object is created, which contains all the user data. This Node object is cast to an Element. The Element object curElm is queried, using the getAttribute() method, to supply the userID and pwd attributes of the current user in the loop.

The Node's userID and pwd attributes are compared to the userID and pwd parameters

sent from the logon page. If the user is found, the routeURL variable is changed point to frame.htm - the secure frameset.

```
outer:
for (int i=0; i<nodeList.getLength(); i++)
{
org.w3c.dom.Node curNode = nodeList.item(i);

//Get userID attribute:
Element curElm = (Element)nodeList.item(i);
String curUserID = curElm.getAttribute("userID");

//Get pwd attribute:
String curPwd = curElm.getAttribute("pwd");

if (curUserID.equals(userID) && curPwd.equals(pwd))
{
routeURL = "frame.htm"
logon.setUserID(userID);
logon.setSecure();
break outer;
} //end if
```

```
}//end for
```

Exception Handling

I use quite a simple mechanism for debugging. A catch is declared for all possible exceptions. Should an exception occur during the run of my JSP page, it will be caught - and the exception's value is converted to String and stored in the msg variable.

The msg variable's contents can be displayed in the HTML part of the page, using <%=msg%>. Once the page has been tested, you may want to comment out the <%=msg%> tag.

```
}catch(Exception e)
{
    msg = msg + e.toString();
}

%>
<%=msg%>
```

Page Routing

The simple JavaScript below will reroute the visitor to the desired page.

<script language="javascript">document.location='<%=routeURL%>'</script>

</body>

</html>

Logon.java:

This application's logon session Bean is an illustration of how simple and useful a Bean can be. By making use of a session Bean, we are able to check on any of the pages on our site if the visitor has access to the secure information.

The Bean keeps track of the user's userID. In the other application pages I use the userID from this bean to unlock information specific to the user.

Getting to the userID is easy. Simply declare the Bean and then use the getUserID() method to retrieve the String. Similarly, the boolean secure can be accessed via the getSecure() method.

The setUserID() and setSecure() methods should be called once the user's identity has been validated. These are the methods we called in **processLogon.jsp**.

```
package logonBean;

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class logon

{

// Variables

String userID = "";

boolean secure = false;


public String getUserID()

{

return userID;

}

public void setUserID(String userID)

{

this.userID = userID;

}

public void setSecure()

{

secure = true;

}
```

```
public boolean getSecure()
{
return secure;
}
}
```

Conclusion:

This tutorial acts an introductory overview of quite a few new technologies. JSP pages provide the server page processing, HTML framework and manipulation. The XML userTable file provides a simple example of the use of an XML file to store your application's data.

Finally, we use a session Java Bean to store information about the user. The functionality of the Java Bean can be extended quite a lot. It would be possible to use JDBC in the Bean to connect to a database, or add methods to handle the processing of XML data.

By asking the user for a logon, and using a session Java Bean, it will be possible to keep track of all sorts of information about the user - such as their banking or credit-card information and their email address. By keeping track of this information, it is possible to offer visitors some great functionality, like shopping carts, email address books or to store the visitor's personal preferences for easy access.

You can download files associated with this tutorial using Save Link/Target As on the following links:

logon.java

logon.jsp

processLogon.jsp

userTable.xml

ΠΑΡΑΡΤΗΜΑ Γ

Μελέτη Περίπτωσης #2

(Πόροι)

Παρακάτω παραθέτουμε τον κώδικα των δύο εκδόσεων της Java κλάσης – διαχειριστή της υπηρεσίας WordNet.

Έκδοση 1.0 (χρήση της βιβλιοθήκης JWNL)

```
package eu.projectName.workPackageName.serviceName.lexicalAnalyzer;
// … various imports
import net.didion.jwnl.JWNL;
/**
 * @author Apostolos Kritikos
 */
public class WordnetHandler {
   // Dictionary object
   public static Dictionary wordnet;
   // Initialize the database!
   static {
      String propertiesFile = ConfigLogMerger.getConfig().getWordnetPropertiesFile();
      try {
         JWNL.initialize(new FileInputStream(propertiesFile));
```

```
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (JWNLException e) {
            e.printStackTrace();
        }
    }
    public WordnetHandler( ) {
    }
    //get senses for all the passed words for all possible POS
    public String getSense(String word) {
        ArrayList<Synset> wordContainer = new ArrayList<Synset>();
   //Indexwordset returns all the POS (as a noun,verb,adj and adverb) that this word could have.
        //the word "run" would return "run" as noun, running as adj and run as verb
        IndexWordSet set = null;
        try {
            set = Dictionary.getInstance().lookupAllIndexWords(word);
        } catch (JWNLException ex) {
            Logger.getLogger(WordnetHandler.class.getName()).log(Level.SEVERE,     null, ex);
        }
        if (set.size() != 0) {
            //get all the possible POS index words
```

```
            IndexWord[] indexWords = set.getIndexWordArray();
            for (int j = 0; j < indexWords.length; j++) {
                wordContainer.addAll(getSenses(indexWords[j]));
            }
            return indexWords[0].getLemma();
        } else {
            return "";
        }
    }
    public ArrayList<Synset> getSenses(IndexWord word) {
        //Returns an arraylist of all passed Indexword's senses
        IndexWord wordInd = word;
        Synset[] senses = null;
        try {
            senses = wordInd.getSenses();
        } catch (JWNLException ex) {
            Logger.getLogger(WordnetHandler.class.getName()).log(Level.SEVERE,     null,
ex);
        }
        ArrayList<Synset> sensesContainer = new ArrayList<Synset>();
        for (int i = 0; i < senses.length; i++) {
            sensesContainer.add(senses[i]);
        }
```

```
        if (sensesContainer != null) {
            return sensesContainer;
        }
        return null;
    }
}
```

Έκδοση 2.0 (χρήση της βιβλιοθήκης JWI)

```
package eu.projectName.workPackageName.serviceName.lexicalAnalyzer;
// … various imports
import edu.mit.jwi.IDictionary;
//… other JWI imports
/**
 * @author Apostolos Kritikos
 */
public class WordnetHandler2 {
        // Dictionary object
        public static IDictionary wordnet;
        // Initialize the database!
        static {
                String path = " /dict";
                URL url = null;
```

```
		try {
			url = new URL("file", null, path);
		} catch (MalformedURLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		// construct the dictionary object and open it
		wordnet = new edu.mit.jwi.Dictionary(url);
		wordnet.open();
	}
	public WordnetHandler2() {
	}
	public String getSense(String text, String pos) {
		IIndexWord idxWord = null;
		if (pos.equals("NOUN")) {
			idxWord = wordnet.getIndexWord(text, POS.NOUN);
		} else if (pos.equals("VERB")) {
			idxWord = wordnet.getIndexWord(text, POS.VERB);
		}
		IWordID wordID = null;
		try {
		wordID = idxWord.getWordIDs().get(0);
		}
```

```
                catch (NullPointerException e) {
                        return "";
                }
                IWord word = wordnet.getWord(wordID);
                return word.getLemma();
        }
}
```