

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τους ανθρώπους που συνέβαλλαν στην επιτυχή ολοκλήρωση αυτής της διπλωματικής εργασίας.

- Τον καθηγητή μου, κ. Ανδρέα Συμεωνίδη, για την εμπιστοσύνη με την ανάθεσή της, την αρωγή του κατά την εκπόνησή της και τις συμβουλές του στις πιο κρίσιμες στιγμές
- Την οικογένειά μου για την απεριόριστη υποστήριξη και εμπιστοσύνη
- Τους φίλους μου για τη συμπαράσταση και τις συνεχείς παροτρύνσεις τους

Περίληψη

Βιώνουμε μία εποχή συνταρακτικής αύξησης της ποσότητας της πληροφορίας σε κάθε τομέα της ανθρώπινης ζωής. Ο τομέας της Τεχνολογίας Λογισμικού, κεντρικός πυλώνας πλέον της σύγχρονης ανθρώπινης δραστηριότητας, ήταν αδύνατον να μείνει ανεπηρέαστος. Μοιραία, τα προϊόντα λογισμικού που εξυπηρετούν ανάγκες μεγάλωσαν, εξελίχθηκαν, πολλαπλασιάστηκαν. Μαζί τους πολλαπλασιάστηκαν και τα δεδομένα, τα οποία ακολουθώντας το ρεύμα της εποχής, δε θα μπορούσε/έπρεπε να μείνουν ανεκμετάλλευτα. Η συσσωρευμένη σε αυτά γνώση απλά περιμένει να ανακαλυφθεί. Έτσι γεννήθηκε ο χώρος της Εξόρυξης Δεδομένων από Αποθήκες Λογισμικού.

Οι μέχρι τώρα σχετικές προσεγγίσεις στην Εξόρυξη Δεδομένων από Αποθήκες Λογισμικού επικεντρώνονται στη μελέτη των αλλαγών στον πηγαίο κώδικα των προγραμμάτων. Εντούτοις, άλλες Αποθήκες Λογισμικού παραμένουν ανεκμετάλλευτες σε μεγάλο βαθμό.

Στην παρούσα διπλωματική εργασία στόχος είναι η ενασχόληση με τα Συστήματα Καταγραφής Σφαλμάτων των προγραμμάτων, και συγκεκριμένα η επίδειξη μερικών από τις δυνατότητες για την εξαγωγή υφής δεδομένων που βρίσκεται σε αυτά. Επιλέξαμε να δημιουργήσουμε και να μελετήσουμε σημασιολογικά ορισμένα χαρακτηριστικά του συνόλου δεδομένων, ώστε να εντοπίσουμε τη λανθάνουσα πληροφορία που κρύβεται στη φυσική περιγραφή στις Αναφορές των Σφαλμάτων. Με αυτό κατά νου, κάναμε χρήση τεχνικών Επεξεργασίας Φυσικής Γλώσσας και Σημασιολογικής Ανάλυσης και Συσχέτισης.

Το τελικό αποτέλεσμα της διπλωματικής εργασίας είναι ένας πολυπαραμετρικός μηχανισμός σημασιολογικά ενήμερης πληροφορίας που προέρχεται από Αναφορές Σφαλμάτων, όπως αυτές ορίζονται σε μεγάλα σε έκταση έργα λογισμικού.

Abstract

A mechanism for extracting semantically-aware knowledge in Software Repositories

We humans are confronted with an unprecedented increase in the amount of information we intake, in virtually every part of our digital life. The field of Software Engineering, being a central foothold of the modern human activity, has been inevitably affected. As a direct consequence, software products that meet our needs have grown, evolved, multiplied. Along, software data have grown too, so much that, following the trend of the time, could not be ignored anymore. Thus, the field of Mining Software Repositories has emerged...

Related literature up to this point have been studying mainly changes in program source code, leaving out information residing in other kinds of Software Repositories, which have remained relatively unexploited.

Current diploma thesis attempts to bridge the gap and exploit data coming from Software Issue Tracking Systems. Focus is given on extracting knowledge on one of the most critical facets of Software development, that of identifying bugs. We have developed a mechanism that semantically analyzes certain features of the available data, in order to detect the latent information inside the natural description coming with the bug reports. To do so, we have exploited primitives from the fields of Natural Language Processing, Semantic Analysis and Semantic Relatedness.

The final product of the diploma thesis is a multi-variate mechanism for extracting semantically-aware knowledge in Software Repositories of well-known and fairly developed projects.

Nikolaos Stasinopoulos, November 2010

Περιεχόμενα

Ευχαριστίες	5
Περίληψη	7
Abstract.....	8
Περιεχόμενα	9
Λίστα Σχημάτων – Εικόνων	12
Λίστα Πινάκων	13
Κεφάλαιο 1: Εισαγωγή.....	15
1.1 Αντικείμενο της Διπλωματικής Εργασίας	15
1.2 Στόχοι της Διπλωματικής Εργασίας	16
1.3 Δομή της Διπλωματικής Εργασίας.....	16
Κεφάλαιο 2: Εξάγοντας Γνώση από Αποθήκες Λογισμικού	19
2.1 Οι Αποθήκες Λογισμικού (Software Repositories)	19
2.1.1 Δεδομένα Λογισμικού στο Σύστημα Διαχείρισης Εκδόσεων (Version Control System)	20
2.1.2 Άλλα Δεδομένα Λογισμικού – Μεταδεδομένα	20
2.2 Προεπεξεργασία των Δεδομένων (Data preprocessing)	21
2.2.1 Η Εξαγωγή των Δεδομένων	21
2.2.2 Ανάκτηση Συναλλαγών	22
2.2.3 Αντιστοίχιση Αλλαγών σε Οντότητες.....	24
2.2.4 Καθαρισμός Δεδομένων	24
2.3 Δημοφιλείς στρατηγικές Εξόρυξης Δεδομένων σε Αποθήκες Λογισμικού	25
2.3.1 Τα ερωτήματα της MSR διαδικασίας.....	26
2.3.2 Μετρικές αξιολόγησης της MSR διαδικασίας	26
2.4 Επισκόπηση βιβλιογραφίας.....	27
2.4.1 Προσεγγίσεις ανάλυσης μεταδεδομένων	27
2.4.2 Προσεγγίσεις ανάλυσης διαφορών ανάμεσα στις εκδόσεις λογισμικού	30

2.4.3	Προσεγγίσεις ανίχνευσης συχνών μοτίβων (Frequent-pattern mining)	32
2.5	Το υποκείμενο του πειράματος	32
Κεφάλαιο 3: Μεθοδολογία I Επεξεργασία Δεδομένων – Φυσικής Γλώσσας		35
3.1	Περιγραφή της Μεθοδολογίας	35
3.1.1	Τα βασικά στάδια της μεθοδολογίας	35
3.1.2	Τεχνικές που υιοθετήθηκαν	36
3.2	Εξαγωγή Δεδομένων	38
3.2.1	Δομή ενός ticket	38
3.2.2	Το πρότυπο JSON	40
3.2.3	Βιβλιοθήκες Συντακτικής Ανάλυσης JSON (JSON Parsing)	42
3.2.4	Επιλογή Βιβλιοθηκών JSON Parsing	42
3.2.4.1	Ο Συντακτικός Αναλυτής Jackson JSON Processor	43
3.2.4.2	Ο Συντακτικός Αναλυτής org.json	43
3.2.5	Υλοποίηση Χαρτογράφησης και Αποθήκευσης των Δεδομένων της RoR	43
3.3	Προεπεξεργασία Δεδομένων	45
3.3.1	Καθαρισμός ασύμβατης πληροφορίας	46
3.3.2	Διαχείριση Μερικώς Δομημένων Δεδομένων Ιστού	46
3.3.2.1	Η Βιβλιοθήκη HtmlCleaner και η γλώσσα ερωτημάτων XPath	46
3.4	Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing)	48
3.4.1	Γενικές Αρχές	48
3.4.2	Η Στατιστική Επεξεργασία Φυσικής Γλώσσας (Statistical NLP)	51
3.4.3	Ο Επεξεργαστής Φυσικής Γλώσσας Stanford Parser	52
3.4.4	Εφαρμογή του Stanford Parser στα δεδομένα της RoR	53
Κεφάλαιο 4: Μεθοδολογία II: Σημασιολογική Ανάλυση και Συσχέτιση των Δεδομένων		57
4.1	Σημασιολογική Ανάλυση	57
4.1.1	Πιθανοτικό Μοντέλο Κύριου Θέματος (Probabilistic Topic Model)	57
4.1.2	Η Λανθάνουσα κατά Dirichlet Κατάταξη (LDA)	59
4.1.3	Η Δειγματοληψία Gibbs	60
4.1.4	Η βιβλιοθήκη JGibbLDA	61
4.1.5	Εφαρμογή της Λανθάνουσας κατά Dirichlet Κατάταξης και Εκπαίδευση του Μοντέλου στα Δεδομένα tickets	62

4.2	Σημασιολογική Συσχέτιση.....	64
4.2.1	Μετρικές Σημασιολογικής Συσχέτισης	64
4.2.2	Η Κανονικοποιημένη Google Απόσταση.....	65
4.2.3	Εφαρμογή της Μετρικής NGD στα δεδομένα του προβλήματος.....	66
4.3	Προετοιμασία τελικού Συνόλου Δεδομένων.....	68
Κεφάλαιο 5:	Επίδειξη και Τεκμηρίωση.....	71
5.1	Υλοποίηση πειραμάτων	71
5.1.1	Πρώτο Πείραμα: Επιλογή Βέλτιστου Αλγορίθμου.....	72
5.1.2	Πείραμα 2ο: Σύγκριση Κατασκευασμένων Συνόλων Δεδομένων	73
5.2	Αναγνώριση συσχετίσεων ανάμεσα στα εγγενή και τα επίκτητα χαρακτηριστικά του σετ	74
Κεφάλαιο 6:	Συμπεράσματα – Μελλοντική Εργασία	77
6.1	Σύνοψη - Συμπεράσματα.....	77
6.2	Μελλοντική Εργασία.....	77
A.1	Το σύστημα καταγραφής σφαλμάτων Lighthouse	79
A.2	Απεικόνιση του ticket #1800 σε JSON format	80
Βιβλιογραφία		87

Λίστα Σχημάτων – Εικόνων

Σχήμα 2.2-1 Παράδειγμα Μεταγραφής Δεδομένων από CVS σε εσωτερική Βάση Δεδομένων	22
Σχήμα 2.2-2 Διαδικασία Ομαδοποίησης Commits σε Transactions	23
Σχήμα 2.2-3 Αναπαράσταση Αλγορίθμων Σταθερού και Κινητού Χρονικού Πλαισίου για το Σχηματισμό Transactions	23
Σχήμα 2.2-4 Μεθοδολογία Αντιστοίχισης Αλλαγών στον Πηγαίο Κώδικα σε Οντότητες Κώδικα	24
Σχήμα 3.1-1 Τα βασικά στάδια της μεθοδολογίας	35
Σχήμα 3.2-1- Παράδειγμα απεικόνισης ενός ticket σε μορφή XML	40
Σχήμα 3.2-2α - Οι τύποι δεδομένων που αναγνωρίζει το JSON format	41
Σχήμα 3.2-3β - Απεικόνιση Αντικειμένου (object) στο JSON format	41
Σχήμα 3.2-4γ - Απεικόνιση Συλλογής στο JSON Format	41
Σχήμα 3.2-5 Πρώτο Στάδιο: Εξαγωγή Δεδομένων από ticket.json	44
Σχήμα 3.2-6 Στάδια Δημιουργίας Πρώιμου Dataset	45
Σχήμα 3.3-1 Δεύτερο Στάδιο: Προεπεξεργασία και Καθαρισμός Μερικώς δομημένων Δεδομένων	45
Σχήμα 3.3-2 Λειτουργίες της HtmlCleaner	47
Σχήμα 3.4-1 Τρίτο Στάδιο: Εξαγωγή Φυσικής Γλώσσας (Natural Language Processing)	48
Σχήμα 3.4-2 Συντακτική Ανάλυση της πρότασης "Bills on ports and immigration were submitted by Senator Brownback , Republican of Kansas." μέσω δομών φράσεων	50
Σχήμα 3.4-3 Συντακτική Ανάλυση της πρότασης "Bills on ports and immigration were submitted by Senator Brownback , Republican of Kansas." μέσω Γραμματικών Εξαρτήσεων	51
Σχήμα 3.4-4 Επαναληπτική Διαδικασία Συντακτικής Ανάλυσης και Εξαγωγής Διανύσματος Λέξεων	53
Σχήμα 4.1-1 Τέταρτο Στάδιο: Σημασιολογική Ανάλυση	57
Σχήμα 4.1-2 Παραγωγή Εγγράφων από Θέματα	58
Σχήμα 4.1-3 Στατιστική Αναγωγή για την ανακάλυψη Θεμάτων από Έγγραφα	58
Σχήμα 4.1-4 Γραφική αναπαράσταση του LDA Topic Model	59
Σχήμα 4.2-1 Πέμπτο Στάδιο: Σημασιολογική Συσχέτιση (Semantic Relatedness)	64
Εικόνα 3.2-1 Γραφική Αναπαράσταση ticket σε μορφή JSON	38
Εικόνα 3.2-2 Η δομή του καταλόγου Data που περιέχει τα tickets	42
Εικόνα 4.2-1 Αρχική Σελίδα Lighthouseapp.com	79
Εικόνα 4.2-2 Ενημέρωση του Lighthouseapp μέσω commit στο Σύστημα Διαχείρισης Εκδόσεων git	80

Λίστα Πινάκων

Πίνακας 3.3-2 Διαχωρισμός προτάσεων με markup parsing	48
Πίνακας 3.3-3 Διαχωρισμός προτάσεων με markup parsing μετά την αφαίρεση ειδικών χαρακτήρων .	48
Πίνακας 4.1-1 Παράδειγμα αρχείου εισόδου για τη βιβλιοθήκη JGibbLDA	61
Πίνακας 4.1-2 Ορίσματα παραμετροποίησης JGibbLDA	61
Πίνακας 4.1-3 Επιστρεφόμενα αρχεία μετά την εφαρμογή JGibbLDA	62
Πίνακας 4.1-4 Οθόνη Τερματικού μετά την Εφαρμογή LDA σε Πραγματικά Δεδομένα	62
Πίνακας 4.1-6 Επιστρεφόμενο αρχείο model-final.twords . Οι λέξεις σε bold επιλέγονται ως οι πλέον αντιπροσωπευτικές του κάθε θέματος	63
Πίνακας 4.1-7 Επιστρεφόμενο αρχείο model-final.theta Κατάταξη tickets/εγγράφων ανά topic	63
Πίνακας 4.2-1 Συνοπτική Λίστα Σφαλμάτων Λογισμικού	66
Πίνακας 4.2-2 Λίστα Χαρακτηριστικών Ποιότητας Λογισμικού κατά το Πρότυπο ISO/IEC 9126-1:2001..	67
Πίνακας 4.2-3 Τα θέματα που έχουν προκύψει από την περιγραφή του ticket.....	67
Πίνακας 4.2-4 Αντιστοίχιση Θεμάτων tickets σε Bugs και SQ Metrics (Εμφάνιση Τερματικού)	68
Πίνακας 4.3-1 Φίλτρα που εφαρμόστηκαν στις Ιδιότητες του Συνόλου Δεδομένων	69
Πίνακας 4.3-2 Παράδειγμα Εφαρμογής φίλτρου StringToWordVector με χρήση μετασχηματισμού Fourier και εντοπισμό ετυμολογικής ρίζας	69
Πίνακας 5.1-1 Αποτελέσματα Πρώτου Πειράματος για επιλογή Αλγορίθμου	72
Πίνακας 5.1-2 Αποτελέσματα Δεύτερου Πειράματος για Σύγκριση των Datasets	73

Κεφάλαιο 1: Εισαγωγή

1.1 Αντικείμενο της Διπλωματικής Εργασίας

Η τεχνολογία της Εξόρυξης Δεδομένων (Data Mining) έχει τα τελευταία χρόνια αναδειχτεί ως χρήσιμο εργαλείο ανακάλυψης γνώσης σε πολλούς, ενίοτε απρόσμενους, τομείς ενδιαφέροντος. Η Εξόρυξη Δεδομένων άνθησε και ωθήθηκε από την επανάσταση που έφερε η μαζική παραγωγή, διακίνηση και αποθήκευση πληροφορίας από ένα συνεχώς αυξανόμενο αριθμό πηγών (Βάσεις Δεδομένων, Αποθήκες Δεδομένων, Web 1.0, Web 2.0 κλπ).

Με πεδίο εφαρμογής που αυξάνει το εύρος του όσο αυξάνονται τα δεδομένα, η διαδικασία της ανακάλυψης μοτίβων (pattern discovery) καλύπτει σήμερα τομείς όπως το εμπόριο και γενικά το «επιχειρείν», την Επιστήμη και Τεχνολογία (με παραδείγματα τη Βιοπληροφορική, τη Γενετική, τη Διανομή Ηλεκτρικής Ενέργειας) ή την Εκπαίδευση. Στον ευρύτερο χώρο της Τεχνολογίας Λογισμικού (Software Engineering), η δραματική αύξηση των διαθέσιμων δεδομένων επήλθε, σε μεγάλο βαθμό, ως αποτέλεσμα της χρήσης υποβοηθητικών εργαλείων ανάπτυξης λογισμικού.

Οι συγκεκριμένες τεχνολογίες, ενσωματώνοντας Συστήματα Διαχείρισης Εκδόσεων (Version Control Systems) και Συστήματα Καταγραφής Σφαλμάτων (Bug/Issue Tracking Systems), διευκολύνουν πλέον την παραγωγή κώδικα από ομάδες ανάπτυξης στο επίπεδο του καταμερισμού εργασίας, του ελέγχου και της διόρθωσης λαθών, αλλά και στην επικοινωνία ανάμεσα στα μέλη της ομάδας. Σε δεύτερο επίπεδο, και με την εξάπλωση της πρόσβασης στο διαδίκτυο για τους προγραμματιστές, κατέστη δυνατή και η απομακρυσμένη πρόσβαση στον πηγαίο κώδικα του υπό ανάπτυξη λογισμικού, ελαχιστοποιώντας δραστικά τη σημασία της γεωγραφικής απόστασης.

Οι παραπάνω εξελίξεις οδήγησαν αναπόφευκτα σε μία έκρηξη παραγωγής νέου λογισμικού. Από τη φάση ανάπτυξης των πολυάριθμων έργων λογισμικού (software projects) υπάρχει πλέον άφθονη πληροφορία σε διάφορες μορφές αποθηκευμένη στις λεγόμενες **Αποθήκες Λογισμικού (Software Repositories)**. Τα δεδομένα που εντοπίζονται στις Αποθήκες Λογισμικού, ιδίως όταν προέρχονται από προϊόντα που διαρκώς αναπτύσσονται, αντικατοπτρίζουν διαφορετικές προσεγγίσεις, αλλαγές, σφάλματα και αποφάσεις στην πορεία ανάπτυξης. Αποτελούν, δηλαδή, το αποτύπωμα του **εξελικτικού μονοπατιού** (evolution path) ενός πακέτου λογισμικού.

Από την άλλη, μια από τις σημαντικότερες προκλήσεις της *Τεχνολογίας Λογισμικού* (Software Engineering) είναι η ανακάλυψη τάσεων, τεχνικών, επαναλαμβανόμενων πρακτικών, μη εμφανών (εκ πρώτης όψεως) διασυνδέσεων ανάμεσα στα τμήματα του λογισμικού και ξεχωριστά χαρακτηριστικά του εκάστοτε εξεταζόμενου έργου ή μίας ομάδας έργων λογισμικού που εξετάζονται μαζί.

Είναι προφανές λοιπόν, ότι οι προσεγγίσεις των ερευνητών στην ανάκτηση της πληροφορίας δε θα μπορούσαν παρά να εκμεταλλεύονται (χωρίς εντούτοις να περιορίζονται σε αυτές) και τεχνικές από το ακμάζον πεδίο της *Εξόρυξης Δεδομένων* (Data Mining) και, ευρύτερα, της *Ανακάλυψης Γνώσης* (Knowledge Discovery). Αυτή ακριβώς η προσέγγιση στην εξαγωγή γνώσης από τα δεδομένα που προκύπτουν κατά την ανάπτυξη λογισμικού καλείται **Εξόρυξη Δεδομένων από Αποθήκες Λογισμικού (Mining Software Repositories – MSR)**.

1.2 Στόχοι της Διπλωματικής Εργασίας

Ο Σκοπός της παρούσας Διπλωματικής Εργασίας είναι η εξαγωγή πληροφορίας από τις αναφορές σφαλμάτων που καταγράφονται κατά την παραγωγή ενός έργου λογισμικού. Ως τελικό προϊόν παρέχεται ένας μηχανισμός για τη δημιουργία ενός σημασιολογικά ενήμερου συνόλου δεδομένων στο οποίο θα είναι δυνατόν να εφαρμοστούν τεχνικές Εξόρυξης Δεδομένων. Οι επιμέρους στόχοι που θα οδηγήσουν στο τελικό αποτέλεσμα αναφέρονται παρακάτω.

Αρχική πρόκληση ήταν η μελέτη της υπάρχουσας βιβλιογραφίας ώστε να εντοπιστούν τεχνικές που έχουν ήδη χρησιμοποιηθεί στο χώρο της Εξόρυξης Δεδομένων Λογισμικού, ενώ παράλληλα αποσαφηνίστηκε η σχετική ορολογία. Στη συνέχεια, αναζητήθηκε το Σύστημα Καταγραφής Σφαλμάτων στο οποίο θα εφαρμόζονταν οι τεχνικές εξόρυξης. Σε δεύτερο χρόνο, στόχος ήταν η εξαγωγή των δεδομένων των αναφορών των σφαλμάτων από το σύστημα καταγραφής και η μεταγραφή τους σε προγραμματιστικά διαχειρίσιμη μορφή. Στο τέλος της φάσης αυτής είχαμε μία πρώτη αντίληψη των δεδομένων που είχαμε να επεξεργαστούμε.

Ο κύριος στόχος ήταν η αναζήτηση των μεθόδων που θα οδηγούσαν στην καλύτερη κατανόηση της πληροφορίας και συνακόλουθα η δημιουργία ενός συνόλου δεδομένων με όσο το δυνατόν πληρέστερη και ακριβέστερη πληροφορία. Διαπιστώνοντας ότι κάθε σφάλμα συνοδεύεται από μία λεπτομερή περιγραφή σε φυσική γλώσσα από τον χρήστη που το υπέβαλλε, αποφασίστηκε να αναλυθεί αυτή η περιγραφή με τεχνικές Επεξεργασίας Φυσική Γλώσσας και αναζητηθεί η σημασιολογική υπόσταση της περιγραφής. Τελικά, το σετ δεδομένων που προέκυψε υποβλήθηκε σε ενδεικτικά πειράματα, ώστε να αποδειχθεί ότι εκπληρώνει τις προσδοκίες μας.

1.3 Δομή της Διπλωματικής Εργασίας

Στο Πρώτο Κεφάλαιο εισάγεται ο αναγνώστης στο αντικείμενο της διπλωματικής εργασίας και στη βασική της διάρθρωση.

Στο Δεύτερο Κεφάλαιο, παρατίθενται ουσιαστά σημεία της σχετικής βιβλιογραφίας. Διευκρινίζονται οι έννοιες της Αποθήκης Λογισμικού και των Δεδομένων που περιέχει, και περιγράφονται οι ήδη υπάρχουσες τεχνικές. Στη συνέχεια, εντοπίζεται η χωροθέτηση της παρούσας εργασίας στο χώρο της Εξόρυξης Δεδομένων από Αποθήκες Λογισμικού και επιλέγεται το υποκείμενο των πειραμάτων που ακολουθούν.

Στο Τρίτο Κεφάλαιο, αναπτύσσεται η πρώτη μεθοδολογία η οποία συνίσταται στην εξαγωγή των δεδομένων από την Αποθήκη Λογισμικού, στη διαχείριση μερικών δομημένων δεδομένων ιστού και στην επεξεργασία της φυσικής γλώσσας. Περιγράφονται οι τεχνικές των παραπάνω λειτουργιών και η χρήση των σχετικών εργαλείων.

Στο Τέταρτο Κεφάλαιο, επιχειρείται η σημασιολογική ανάλυση του προβλήματος. Παρουσιάζεται το θεωρητικό υπόβαθρο πίσω από τη στρατηγική αυτή και εφαρμόζονται τα δύο εργαλεία ανάλυσης και συσχέτισης. Το τελικό αποτέλεσμα προκύπτει με την εφαρμογή ενός επιπλέον βήματος προεπεξεργασίας.

Στο Πέμπτο Κεφάλαιο, επιδεικνύεται η χρήση του Συνόλου Δεδομένων που δημιουργήθηκε σε πραγματικές συνθήκες και η αποτελεσματικότητα των μεθόδων που χρησιμοποιήθηκαν. Επιλέγεται ένας αποτελεσματικό αλγόριθμος κατάταξης και εφαρμόζεται στα δεδομένα.

Στο Έκτο και τελευταίο κεφάλαιο, παρατίθενται οι τελικές σκέψεις γύρω από την εργασία και οι πιθανές μελλοντικές επεκτάσεις ή χρήσεις.

Στο Παράρτημα Α παρουσιάζεται το σύστημα διαχείρισης σφαλμάτων το οποίο επεξεργαστήκαμε και ένα δείγμα αναφοράς σφάλματος.

Κεφάλαιο 2: Εξάγοντας Γνώση από Αποθήκες Λογισμικού

2.1 Οι Αποθήκες Λογισμικού (Software Repositories)

Η πραγματική άνθηση στο πεδίο της *Εξόρυξης Δεδομένων από Αποθήκες Λογισμικού* πραγματοποιήθηκε με την επέκταση του ερευνητικού ενδιαφέροντος για την *εξέλιξη του λογισμικού* (software evolution) από τα συστήματα λογισμικού που είχαν παραχθεί για εμπορική χρήση (proprietary software) και στο χώρο του *λογισμικού ανοιχτού κώδικα* (open-source software development). Κύρια αιτία της στροφής αυτής ήταν η, χωρίς σημαντικούς περιορισμούς, πρόσβαση σε αποθήκες υψηλής ποιότητας λογισμικού.

Οι **Αποθήκες Λογισμικού (Software Repositories)** περιέχουν τον πηγαίο κώδικα (source code), όπως αυτός έχει αποτυπωθεί μέσω κάποιου υποβοηθητικού *συστήματος διαχείρισης εκδόσεων* (Version Control System), τις καταγεγραμμένες *απαιτήσεις* (requirements), τα διαπιστωθέντα προγραμματιστικά σφάλματα (bugs) και τις προτάσεις βελτίωσης μέσω των Συστημάτων Καταγραφής Σφαλμάτων (Bug Tracking System), καθώς και μη-πηγαία τμήματα του λογισμικού, όπως τα *αρχεία επικοινωνίας* (πχ ηλεκτρονική λίστες αλληλογραφίας) της ομάδας ανάπτυξης του προϊόντος.

Αν και όλα αυτά τα συστήματα έχουν ως κοινό σκοπό την αυτοματοποίηση διαδικασιών που αφορούν την εξέλιξη του λογισμικού, διαφέρουν ως προς το είδος της πληροφορίας που χειρίζονται, τον τρόπο που το κάνουν και τους formalismούς που χρησιμοποιούν για την αποθήκευσή της. Επιπλέον, καθιστώντας τη συνολική μελέτη πιο επίπονη, τα *δεδομένα λογισμικού* δε διαθέτουν κάποιου είδους *σαφή και ορισμένη μεταξύ τους διασύνδεση*.

Τα βασικά αντικείμενα της Εξόρυξης Δεδομένων Τεχνολογίας Λογισμικού είναι:

- Οι εκδόσεις λογισμικού και το κάθε είδους περιεχόμενό του (πχ κώδικας, οντότητες κώδικα, απαιτήσεις, bugs, τεκμηρίωση κ.ά.) και οι διαφορές που εντοπίζονται ανάμεσά τους
- Τα μεταδεδομένα που συνοδεύουν κάθε αλλαγή στο λογισμικό

2.1.1 Δεδομένα Λογισμικού στο Σύστημα Διαχείρισης Εκδόσεων (Version Control System)

Πιο συγκεκριμένα, η σχετική ερευνητική δραστηριότητα έχει επικεντρωθεί στη μελέτη αποθηκών λογισμικού που διαχειρίζονται τον πηγαίο κώδικα με ένα Σύστημα Διαχείρισης Εκδόσεων. Ιδιαίτερα το πακέτο λογισμικού **CVS** (*Concurrent Versions System*) [(Free Software Foundation)].

Το CVS διαχειρίζεται τις αλλαγές ανάμεσα στις διαφορετικές εκδόσεις λογισμικού, συγκεντρώνοντας ταυτόχρονα δεδομένα που αφορούν τις συνθήκες κάτω από τις οποίες έγιναν οι αλλαγές (commits), καθώς και από ποιον προγραμματιστή αυτές πραγματοποιήθηκαν.

Παρότι ιδιαίτερα χρήσιμο, το CVS πραγματοποιεί μόνο μία απλή καταγραφή των αλλαγών σε επίπεδο αρχείων πηγαίου κώδικα (*source file*) και αριθμού γραμμής (*line of code*), χωρίς να μπαίνει στη διαδικασία να εντοπίσει φυσικές, συντακτικές ή σημασιολογικές οντότητες εντός του κώδικα. Ακόμη, στα μειονεκτήματά του CVS πρέπει να προστεθεί η αδυναμία του να διαχειριστεί πολλαπλές αλλαγές ταυτόχρονα σε διαφορετικά αρχεία (μία διαδικασία που αναφέρεται ως *delta commit* στη σχετική ορολογία) ως μία *συναλλαγή* (*transaction*). Επίσης, στο CVS δεν υπάρχει πρόβλεψη για τη διαχείριση σημείων στα οποία πραγματοποιείται διακλάδωση (*branch*) ή συγχώνευση (*merge*) στην ανάπτυξη του έργου λογισμικού.

Εξελιγμένα συστήματα διαχείρισης εκδόσεων (*versioning*), όπως το *git* (διανεμημένος έλεγχος έκδοσης) (Linus Torvalds) και το *Subversion* (Apache SubVersion), μπορούν να αντιμετωπίσουν πιο αποτελεσματικά κάποιες από τις αδυναμίες του CVS που προαναφέρθηκαν.

2.1.2 Άλλα Δεδομένα Λογισμικού – Μεταδεδομένα

Πέρα από τα δεδομένα που αποτελούν μέρος του πηγαίου κώδικα του λογισμικού, υπάρχουν και δεδομένα που σχετίζονται άμεσα με τον κύκλο ζωής του προγράμματος (*product lifecycle*). Τα δεδομένα αυτά, εφόσον αναλυθούν αυτόνομα ή σε συνδυασμό με τον πηγαίο κώδικα, μπορούν να οδηγήσουν σε σημαντικά συμπεράσματα. Τα δεδομένα αυτά καλούνται **Μεταδεδομένα (Metadata)**, εφόσον σχετίζονται έμμεσα με τον πηγαίο κώδικα.

Μεταδεδομένα περιέχονται σε ένα **Σύστημα Καταγραφής Σφαλμάτων (Bug Tracking System)**. Ιδιαίτερα διαδεδομένη υλοποίηση τέτοιου συστήματος είναι ο **Bugzilla** (*manage software development*). Στον Bugzilla, κάθε bug καταγράφεται (*ticket* ή *bug report*) και συνοδεύεται από επιπλέον μεταδεδομένα σχετικά με το πότε αναφέρθηκε, σε ποιον ανατέθηκε η επίλυσή του, το βαθμό σημαντικότητάς του κ.ά.

Η σημασία των μεταδεδομένων σφαλμάτων (δηλ. των αναφορών σφαλμάτων) είναι μεγάλη, καθώς αποτελούν σημαντικό δείκτη της διαδικασίας εξέλιξης μέσω επίλυσης προβλημάτων στον πηγαίο κώδικα του λογισμικού ή στη λογική λειτουργίας τους. Είναι, λοιπόν, δηλωτικά του μονοπατιού εξέλιξης.

Άλλοι τύποι μεταδεδομένων περιέχονται στα αρχεία επικοινωνίας των μελών της ομάδας ανάπτυξης, όπως emails ή συνεδρίες συνομιλίας (chat sessions). Τα δεδομένα που ανταλλάσσονται περιέχουν πληροφορία που μπορεί να εξηγήσει επιλογές στον κώδικα ή να συσχετίσει μοτίβα συμπεριφοράς με συγκεκριμένους χρήστες. Μία επιλογή για τα δεδομένα επικοινωνίας είναι να μελετηθούν παράλληλα με κοινωνικούς γράφους (social graphs) που ορίζονται στο σύνολο των ατόμων της ομάδας ανάπτυξης (Huang & Liu, 2005)

2.2 Προεπεξεργασία των Δεδομένων (Data preprocessing)

Πέρα από τις υπόλοιπες εγγενείς αδυναμίες του παραδοσιακού Συστήματος Διαχείρισης Εκδόσεων CVS, τα ιστορικά δεδομένα που βρίσκονται αποθηκευμένα στις Αποθήκες Λογισμικού πρέπει να υποστούν επεξεργασία πριν να καταστούν διαθέσιμα για τα υπόλοιπα στάδια της MSR διαδικασίας.

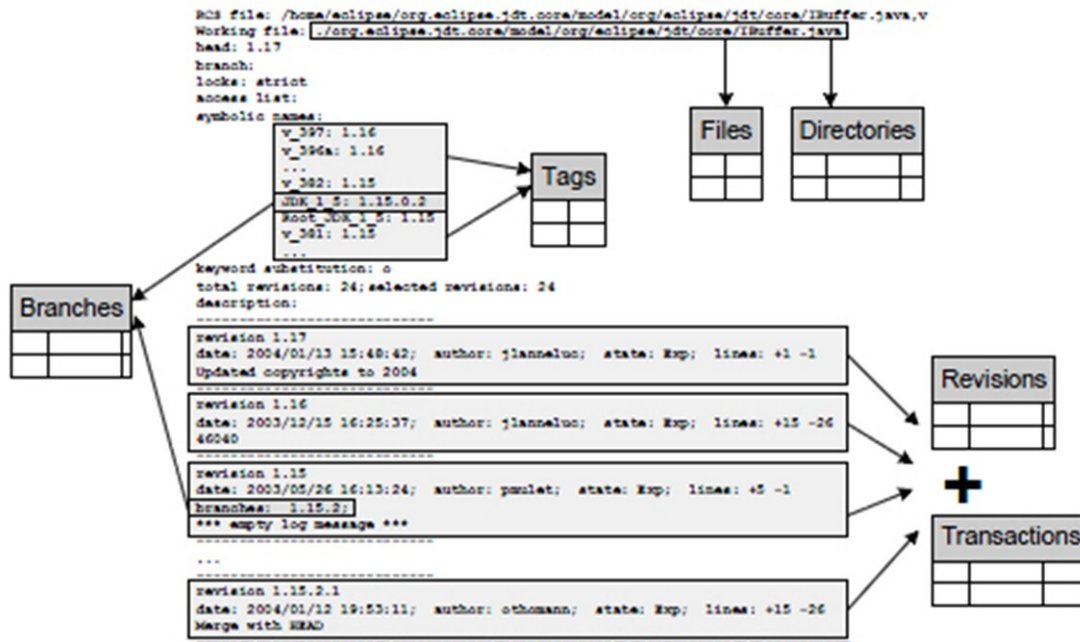
Τα στάδια αυτά περιγράφονται από τους (Zimmermann & Weißgerber, 2004) και αφορούν κυρίως αναλύσεις εξειδίκευσης πάνω στα δεδομένα (fine-grained analyses). Αυτές οι τεχνικές φιλοδοξούν να μειώσουν τον υπολογιστικό φόρτο, την ταχύτητα πρόσβασης στα δεδομένα των Αποθηκών Λογισμικού και το θόρυβο που ενδεχομένως να προκύψει κατά την Εξόρυξη των Δεδομένων. Είναι προφανές πως, ανάλογα με το στόχο της ανάλυσης, υπάρχουν και κάποιες διαφοροποιήσεις στον τρόπο προεπεξεργασίας.

Οι λειτουργίες που πρέπει να επιτελεστούν κατά την προεπεξεργασία των δεδομένων είναι οι εξής:

- *Εξαγωγή Δεδομένων (Data Extraction)*, αν και συνήθως αντιγράφονται αυτούσια από το CVS σε μία βάση δεδομένων
- *Ανάκτηση Συναλλαγών (Restoring Transactions)*, είναι το πρόβλημα του εντοπισμού και ομαδοποίησης των αλλαγών στον κώδικα που συμβαίνουν ταυτόχρονα
- *Αντιστοίχιση Αλλαγών σε Οντότητες (Mapping Changes to Entities)*, δηλαδή ο εντοπισμός των συσχετίσεων σε επίπεδο μεθόδου, μεταβλητής κτλ., αντί απλά σε μία γραμμή κώδικα (LoC – Line of Code)
- *Καθαρισμός δεδομένων (Cleansing Data)*, ως ένα κατεχοχήν στάδιο της Εξόρυξης Δεδομένων εν γένει, εφαρμόζεται στο διαχωρισμό συναλλαγών που προκαλούν περισσότερο θόρυβο (noise) στα δεδομένα απ' ότι συμβολή στην εξαγωγή γνώσης.

2.2.1 Η Εξαγωγή των Δεδομένων

Για την εξαγωγή των δεδομένων από το CVS χρησιμοποιείται η εντολή **log** που επιστρέφει το σύνολο των αρχείων που έχουν αποθηκευθεί στην Αποθήκη Λογισμικού διαχρονικά. Το εξαγόμενο σύνολο αναλύεται συντακτικά (parsing) και η πληροφορία τοποθετείται στους πίνακες μίας βάσης δεδομένων. Ενδεικτικά, αποθηκεύονται τα αρχεία και οι τοποθεσίες (directories), οι αναθεωρήσεις (revisions), οι συναλλαγές (transactions), οι διακλαδώσεις (branches) στην εξέλιξη του λογισμικού κ.ά. (Apache Software Foundation)



Σχήμα 2.2-1 Παράδειγμα Μεταγραφής Δεδομένων από CVS σε εσωτερική Βάση Δεδομένων

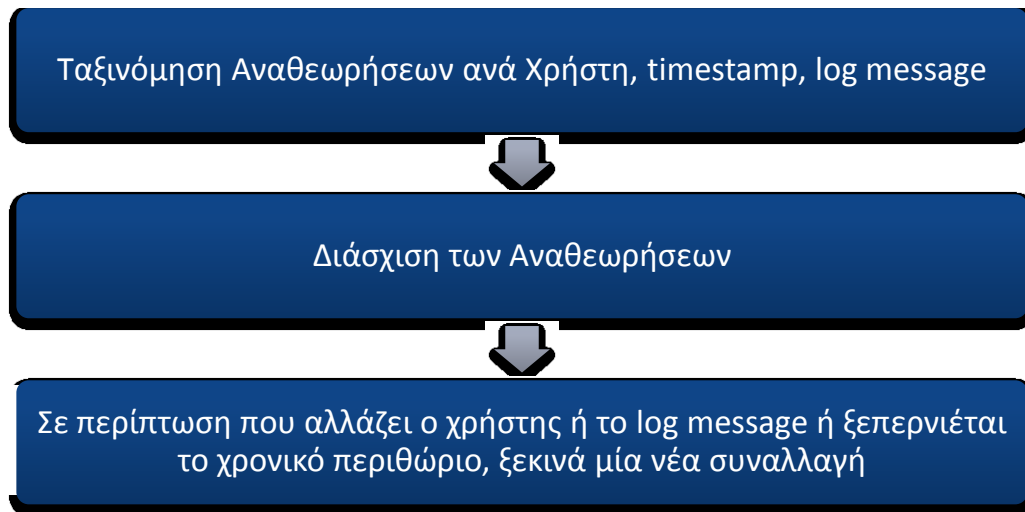
2.2.2 Ανάκτηση Συναλλαγών

Όπως έχει ήδη αναφερθεί, το CVS δεν έχει τη δυνατότητα να ομαδοποιήσει τις αλλαγές (commits) που συνέβησαν σε πολλαπλά αρχεία και είναι συσχετισμένες μεταξύ τους σε μία συναλλαγή (transaction) από την πλευρά του προγραμματιστή. Αυτό οδηγεί σε σαφή μείωση της πληροφορίας από πλευράς ανάλυσης, ιδίως σε σημασιολογικό επίπεδο συσχέτισης των προγραμματιστικών οντοτήτων εντός του πηγαίου κώδικα.

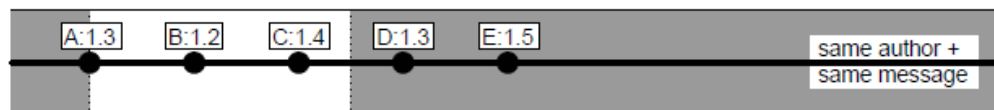
Μία προφανής λύση στο ζήτημα είναι θεωρηθούν ως μία συναλλαγή όλες οι *αναθεωρήσεις* (revisions ή αλλαγές) που προέρχονται από τον ίδιο προγραμματιστή εντός ενός *εύλογου* χρονικού διαστήματος. Οι δύο προσεγγίσεις για το χρονικό περιθώριο είναι οι εξής:

- Σταθερό Χρονικό Πλαίσιο (fixed time window), σε αυτήν την περίπτωση η μέγιστη διάρκεια μιας συναλλαγής είναι επακριβώς ορισμένη
- Κινητό Χρονικό Πλαίσιο (sliding time window), ο μέγιστος χρόνος ανάμεσα σε δύο διαδοχικές αναθεωρήσεις είναι ορισμένος

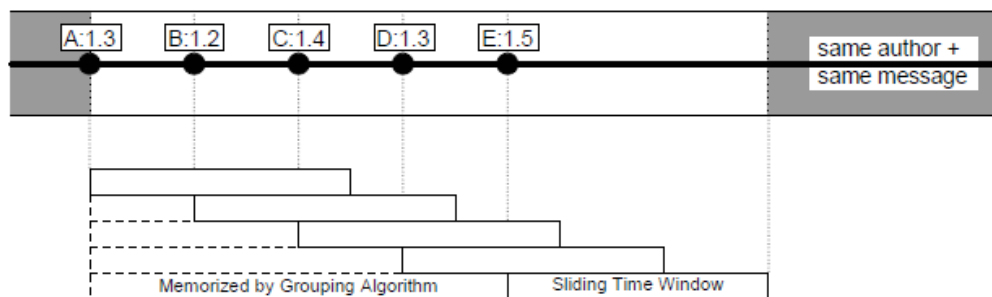
Στα Σχήματα Σχήμα 2.2-2 και Σχήμα 2.2-3 φαίνεται η αλγοριθμική ομαδοποίηση αλλαγών στον πηγαίο κώδικα σε Συναλλαγές.



Σχήμα 2.2-2 Διαδικασία Ομαδοποίησης Commits σε Transactions



(a) Fixed Time Window



(b) Sliding Time Window

Σχήμα 2.2-3 Αναπαράσταση Αλγορίθμων Σταθερού και Κινητού Χρονικού Πλαισίου για το Σχηματισμό Transactions

Η επιλογή ανάμεσα σε Σταθερό ή Κινητό Πλαίσιο εξαρτάται από το αντικείμενο της ανάλυσης, με το τελευταίο να υπερτερεί σε μεγαλύτερα ή αόριστα χρονικά διαστήματα.

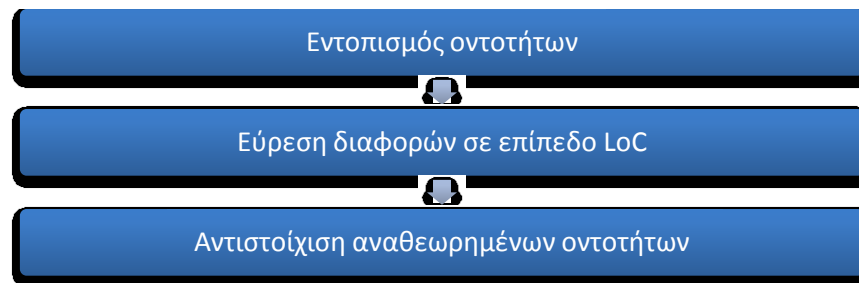
Για πιο ακριβείς ομαδοποιήσεις σε συναλλαγές μπορούν να χρησιμοποιηθούν αναφορές όπως τα *commit mails*, που διαμοιράζονται στις ομάδες ανάπτυξης και περιέχουν το σύνολο των αρχείων που επηρεάστηκαν από τις αλλαγές, καθώς και άλλα σχετικά στοιχεία.

2.2.3 Αντιστοίχιση Αλλαγών σε Οντότητες

Η αδυναμία του CVS να διαχειριστεί τις αλλαγές σε επίπεδο οντοτήτων (συντακτικών, πολύ περισσότερο σημασιολογικών) οδηγεί στην ανάγκη χρήσης επιπλέον εργαλείων και μεθοδολογιών. Έχουν προταθεί διαφορετικές προσεγγίσεις που διαφοροποιούνται στα επιμέρους σημεία. Σε γενικές γραμμές, η ανάλυση των αλλαγών σε χαμηλότερο επίπεδο από αυτό του αρχείου (file level) μπορεί να γίνει με ένα εργαλείο που εντοπίζει διαφορές ανάμεσα στις εκδόσεις (*diff tool*) και με περαιτέρω συντακτική επεξεργασία. Εξάλλου, με αυτόν τον τρόπο μπορεί να γίνει χειρισμός όχι μόνον των ιστορικών δεδομένων (historical data), αλλά και πηγαίου κώδικα, τεκμηρίωσης, αρχείων XML, ακόμη και UML διαγραμμάτων. Ενδεικτικά παρουσιάζουμε τη μεθοδολογία των Zimmermann & Weißgerber (2004).

Μεθοδολογία Αντιστοίχισης.

Για κάθε έκδοση του κώδικα εντοπίζονται οι οντότητες (πχ μέθοδοι, μεταβλητές) με ευριστικές μεθόδους. Στη συνέχεια συγκρίνονται μεταξύ τους οι διαδοχικές εκδόσεις (revisions) του κώδικα μέσω ενός *diff* αλγορίθμου (τυπικά *ldiff*) και εντοπίζονται οι διαφοροποιήσεις στο επίπεδο γραμμής κώδικα (Line of Code – LoC). Οι οντότητες που βρίσκονται μέσα στις γραμμές κώδικα στις οποίες εντοπίστηκαν οι διαφοροποιήσεις θεωρούνται ως τροποποιημένες. (Σχήμα 2.2-4 **Error! Reference source not found.**)



Σχήμα 2.2-4 Μεθοδολογία Αντιστοίχισης Αλλαγών στον Πηγαίο Κώδικα σε Οντότητες Κώδικα

Η μέθοδος αυτή έχει εγγενή προβλήματα που οφείλονται στη φύση του *diff* αλγορίθμου, ο οποίος επενεργεί σε γραμμές πηγαίου κώδικα και όχι σε μεμονωμένες οντότητες. Εξαιτίας αυτού, έχουν προταθεί διαφοροποιημένοι *diff* αλγόριθμοι που υπολογίζουν διαφορές σε οντότητες. Οι αλγόριθμοι αυτοί είναι πιο δαπανηροί υπολογιστικά, αλλά αποφέρουν καλύτερα αποτελέσματα.

Εναλλακτικά, η ανάλυση προχωρά σε ανάπτυξη του προγράμματος σε συντακτικό επίπεδο με τη χρήση *Abstract Syntax Trees* – ASTs, ή σε σημασιολογικό, με *Abstract Semantic Graphs*.

2.2.4 Καθαρισμός Δεδομένων

Όπως και σε κάθε πεδίο εφαρμογής των τεχνικών της Εξόρυξης Δεδομένων, έτσι και στην Εξόρυξη Δεδομένων από Αποθήκες Λογισμικού ο καθαρισμός των δεδομένων (Data Preprocessing) που θα χρησιμοποιηθούν έχει μεγάλη σημασία για να μειωθεί ο υπολογιστικός φόρτος, για την αποφυγή αλλοίωσης των συμπερασμάτων.

Πέρα από τα κλασικά θέματα στον καθαρισμό δεδομένων, δύο είναι τα σημαντικά ζητήματα που πρέπει να αντιμετωπιστούν:

- Το ενδεχόμενο μίας πολύ μεγάλης συναλλαγής με πολλές ταυτόχρονες αλλαγές στον κώδικα. Μία τέτοια συναλλαγή πιθανώς υποδηλώνει συνολική αλλαγή στη δομή του συστήματος και όχι μία αλλαγή που διαδίδεται σε άλλες αλλαγές (change coupling/propagation). Αυτή η περίπτωση αντιμετωπίζεται με την απόρριψη του χαρακτηρισμού ως απλών συναλλαγών (transactions) των ομαδοποιήσεων που υπερβαίνουν ένα (συνήθως αυθαίρετο) άνω όριο στον αριθμό commits/transaction.
- Μια συναλλαγή που έχει προκύψει από την συγχώνευση δύο κλάδων ανάπτυξης ενός προγράμματος (merging branches). Κάτι τέτοιο έχει ως αποτέλεσμα να επιστραφεί πλήθος διαφορών οντοτήτων που ταυτίζονται ανάμεσα στους κλάδους ανάπτυξης, χωρίς αυτές οι διαφορές να αντικατοπτρίζουν την πραγματική εξελικτική πορεία του λογισμικού στο χρόνο. Επειδή το CVS δεν προσδιορίζει ποιες αλλαγές προέκυψαν από συγχωνευμένους κλάδους, συνήθως καταφεύγουμε σε ευριστικές λύσεις.

2.3 Δημοφιλείς στρατηγικές Εξόρυξης Δεδομένων σε Αποθήκες Λογισμικού

Η Εξόρυξη Δεδομένων Λογισμικού εστιάζει σε δύο γενικές στρατηγικές όσον αφορά τη χρήση των Αποθηκών Λογισμικού:

1. Χρήσης της λειτουργικότητας των ίδιων των συστημάτων διαχείρισης των Αποθηκών Λογισμικού, δηλαδή εντολές εξαγωγής δεδομένων από αυτές. Για παράδειγμα, χρησιμοποιώντας εντολές του CVS εξάγεται μία συγκεκριμένη έκδοση (version) του πηγαίου κώδικα και τα ανάλογα μεταδεδομένα. Χρησιμοποιώντας διαφορετικές εξαγόμενες εκδόσεις προκύπτουν συμπεράσματα που αφορούν τις ιδιότητες ενός συστήματος λογισμικού.
2. Προσέγγιση της εξελικτικής πορείας του λογισμικού ως το σύνολο των αλλαγών που είναι καταχωρημένες στο Σύστημα Διαχείρισης Εκδόσεων. Μελετώντας τις διαφοροποιήσεις από έκδοση σε έκδοση μπορούμε να προσεγγίσουμε καλύτερα αλλαγές που συμβαίνουν, όχι μόνο σε αρχεία, αλλά και οντότητες εντός του κώδικα (σημασιολογικές, φυσικές, συντακτικές, τεκμηρίωσης κ.ά.). Αυτό ακριβώς αποτελεί και το πλεονέκτημα αυτής της μεθοδολογίας: μπορεί να επιτευχθεί διαφορετικός βαθμός «εξειδίκευσης/εστίασης» στον κώδικα (**granularity**), ανάλογα με τα ερωτήματα που τίθενται στην MSR διαδικασία.

Γενικότερα, μπορεί να ειπωθεί ότι στόχος είναι η δημιουργία όσο το δυνατόν πιο εξεζητημένων εργαλείων που έχουν τη δυνατότητα να εντοπίσουν και να εξορύξουν γνώση σε ειδικότερες οντότητες (fine-grained entities). Τα εργαλεία αυτά θα εξορύξουν γνώση η οποία αντλείται, είτε από τις ιδιότητες ενός λογισμικού (software properties), είτε από τη δυναμική των αλλαγών στο λογισμικού (software changes).

2.3.1 Τα ερωτήματα της MSR διαδικασίας

Όμοια με κάθε εφαρμογή εξόρυξης δεδομένων και η MSR διαδικασία θέτει ποιοτικά ερωτήματα που αφορούν δεδομένα τύπου «καλαθιού αγοράς» (*Market Basket Questions - MBQ*). Τα MB ερωτήματα δομούνται ως εξής:

“Εάν το γεγονός A συμβεί τι άλλο είναι πιθανό συμβεί;”

Η απάντηση είναι μία ομάδα κανόνων (*set of rules*) που περιγράφουν την υπό συνθήκη πιθανότητα ανάμεσα σε δύο γεγονότα. (Association Rule Extraction)

Μία δεύτερη κατηγορία ερωτημάτων που έχουν ιδιαίτερη σημασία στην ποιοτική ανάλυση του λογισμικού είναι τα ερωτήματα κυριαρχίας (*Prevalence Questions - PQ*). Τυπικά, οι PQ απαντούν σε ποσοτικά ερωτήματα ή ερωτήματα τύπου «ναι ή όχι» (Boolean). Η σημαντικότητα αυτών των ερωτημάτων φαίνεται στον υπολογισμό μετρικών (*complexity metrics*) του λογισμικού ή απαντήσεων σε λογικές ερωτήσεις, όπως:

«Έγινε κάποια αλλαγή στην 47^η γραμμή κώδικα;»

2.3.2 Μετρικές αξιολόγησης της MSR διαδικασίας

Αν και το εύρος των διαθέσιμων αποθηκών λογισμικού (*software repositories*) είναι εξαιρετικά μεγάλο, λόγω και της πληθώρας προϊόντων λογισμικού που έχουν αναπτυχθεί με βάση τη φιλοσοφία και το μοντέλο του ανοικτού κώδικα, στην πράξη οι περισσότερες ερευνητικές προσπάθειες επικεντρώνονται γύρω από συγκεκριμένα projects μεγάλης απήχησης. Αυτή η τάση έχει ωφελήσει τις συγκρίσεις ανάμεσα στα αποτελέσματα των εργαλείων που εκμεταλλεύονται την MSR διαδικασία. Για αυτά τα εργαλεία είναι δυνατόν να χρησιμοποιηθούν καλώς ορισμένες κλασικές **μετρικές** απόδοσης (*metrics*), όπως η **ακρίβεια** (*precision*) και η **ανάκληση** (*recall*).

Η ακρίβεια () περιγράφει ποιο κλάσμα των επιστρεφόμενων από το εργαλείο αποτελεσμάτων είναι αποδεκτό.

(2.1)

Η ανάκληση () ισούται με το πόσα σχετικά αποτελέσματα επεστράφησαν σε σχέση με όσα αναμένονταν.

(2.2)

Τα δύο αυτά μέτρα είναι επιθυμητό να προσεγγίζουν τη μονάδα. Στην πράξη, εντούτοις, είναι αδύνατο να επιτευχθούν τόσο υψηλά αποτελέσματα για την ακρίβεια και για την ανάκληση ταυτόχρονα, συνεπώς ο στόχος προσδιορίζεται στο να βρεθεί μία βέλτιστη συμπεριφορά σε ένα συνδυασμό των δύο.

Οι (Zimmermann, Weißgerber, Diehl, & Zeller, 2004) προσπάθησαν επιπροσθέτως να συγκεντρώσουν τα αποτελέσματα που έβγαλαν για κάθε **κανόνα συσχέτισης (association rule)** ανάμεσα σε επιστρεφόμενο και αναμενόμενο αποτέλεσμα σε ποσοτικούς δείκτες.

Λαμβάνοντας τη μέση τιμή όλων των ζευγών ακρίβειας και ανάκλησης για όλα τα ιστορικά δεδομένα των αποθηκών λογισμικού (precision – recall pairs) αποκτάται μία άποψη, *προσανατολισμένη στο χρήστη (user-oriented)* ή, αλλιώς, μακρο-αξιολόγηση (macro-evaluation) της MSR διαδικασίας.

$$\text{---} \quad , \quad \text{---} \quad (2.3)$$

Αντιθέτως, εάν μετρηθούν οι μετρικές αυτές με αναφορά στο σύνολο των επιστρεφόμενων ή αποδεκτών αποτελεσμάτων και όχι ξεχωριστά για κάθε ζεύγος, δηλαδή ισχύουν οι ακόλουθοι ορισμοί:

$$\text{---} \quad (2.4)$$

$$\text{---} \quad (2.5)$$

Η αξιολόγηση αυτή καλείται μικρο-εκτίμηση (micro-evaluation) ή προσέγγιση *προσανατολισμένη στο σύστημα (system-oriented approach)*. Αυτή η θεώρηση επιτρέπει ερωτήματα κυριαρχίας (PQ) του τύπου «κάθε ν-οστή πρόταση είναι σωστή ή λάθος».

Εναλλακτικά, στην περίπτωση που η MSR διαδικασία *μοντελοποιηθεί* σε θεωρητικό επίπεδο –μία σαφώς πιο επίπονη επιλογή– το αποτέλεσμά της αξιολογείται συγκρίνοντας την κατανομή των θεωρητικά υπολογισμένων με την πραγματική κατανομή των υπαρκτών ιστορικών αποτελεσμάτων. Σε αυτήν την περίπτωση η σύγκριση πραγματοποιείται με τη χρήση *μετρικών εντροπίας (entropy measurements)*.

2.4 Επισκόπηση βιβλιογραφίας

2.4.1 Προσεγγίσεις ανάλυσης μεταδεδομένων

Όπως έχει ήδη αναφερθεί, η έρευνα στο πεδίο της Εξέλιξης του Λογισμικού (με τη χρήση τεχνικών εξόρυξης δεδομένων) δεν επικεντρώνεται απαραίτητα στα ίδια τα δεδομένα που είναι αποθηκευμένα στις αποθήκες λογισμικού πηγαίου κώδικα και στις διαφορές ανάμεσα στις εκδόσεις ενός προγράμματος. Αντίθετα, πολύ συχνά εστιάζει στη μελέτη **μεταδεδομένων (metadata)**, δηλαδή δεδομένων που περιγράφουν τα αρχικά δεδομένα (δηλαδή τον πηγαίο κώδικα), καθιστώντας εφικτή την εξαγωγή λογικών συσχετίσεων. Για παράδειγμα, ο χρόνος στον οποίο έγινε μία συγκεκριμένη αλλαγή στον κώδικα (timestamp) και η ταυτότητα χρήστη που την έκανε μπορούν να συνδυαστούν χρονικά με ένα συγκεκριμένο bug report και αποτελούν μεταδεδομένα.

Το πλεονέκτημα της χρήσης μεταδεδομένων είναι ότι αποφεύγεται η πρόσβαση απευθείας στον πηγαίο κώδικα και μειώνεται η ανάγκη για την προετοιμασία των δεδομένων για εξόρυξη. Τυπικά, τα εργαλεία που χρησιμοποιούνται είναι σχετικά απλά στη σύλληψη (κανονικές εκφράσεις – *regular expressions*, *heuristics*, *έυρεση κοινών υποακολουθιών*).

Η κατηγορία αυτή συστημάτων είναι και η πιο διαδεδομένη. Χαρακτηριστικά παραδείγματα τέτοιων προσεγγίσεων αποτελούν τα παρακάτω:

1. Στην εργασία τους οι (Gall, Jayayeri, & Krajewski, 2003) εξάγουν λογικές και κρυμμένες σημασιολογικές εξαρτήσεις ανάμεσα σε κλάσεις (στο παράδειγμα του αντικειμενοστρεφούς προγραμματισμού) από το ιστορικό των εκδόσεων του λογισμικού (*version history*). Σκοπός είναι να βρεθούν ποιες κλάσεις αλλάζουν μαζί και πόσες αλλαγές συμπαράσύρονται σε ολόκληρο το σύστημα με την τροποποίηση μίας κλάσης.

Καταγράφοντας όλες τις αλλαγές στις κλάσεις ανά έκδοση του προϊόντος εντοπίζονται μοτίβα αλλαγών (*change patterns*) με βάση το αναγνωριστικό του προγραμματιστή που έκανε την τροποποίηση σε ένα συγκεκριμένο χρόνο (*timestamp*) με πληροφορίες από το ίδιο το CVS (*CVS annotations*). Αλλαγές που συνέβησαν εντός συγκεκριμένου χρονικού παραθύρου (4 min) από τον ίδιο προγραμματιστή θεωρούνται ως λογικά εξαρτημένες (λογική σταθερού χρονικού πλαισίου – Βλέπε Ενότητα 2.2.2). Η μεθοδολογία αυτή εφαρμόστηκε σε εμπορικό σύστημα (*proprietary software*) με ικανοποιητικά αποτελέσματα στην τιμή της πρόβλεψης (*recall*).

Η ανάλυση επεκτάθηκε από τους (Gall, Pinzger, & Fischer, 2003) με την ενσωμάτωση των περιεχόμενων του αρχείου καταγραφής (*log file*) του CVS και αναφορών *bugs* από τον Bugzilla σε μία σχεσιακή βάση δεδομένων. Με τη βοήθεια ευριστικών τεχνικών βασισμένων σε κανονικές εκφράσεις (*regular expressions*) συνδέθηκαν λογικά τα δύο παραπάνω συστήματα. Ιδιαίτερη πρόνοια είχε ληφθεί για τα σημεία συγχώνευσης κλάδων (*branch-merge points*) στην πορεία ανάπτυξης του λογισμικού. Η τεχνική αυτή εφαρμόστηκε στο Mozilla Project και τα αποτελέσματα έδειξαν συσχετίσεις ανάμεσα σε αναφορές *bug* και αλλαγές στον πηγαίο κώδικα σε *επίπεδο αρχείου*, καθώς και στατιστικές παρατηρήσεις σχετικές με τη γραμμική μεγέθυνση του Mozilla Project.

Χρησιμοποιώντας για τρίτη φορά την ίδια μεθοδολογία, οι (Gall, Ratzinger, Oberleitner, & Fischer, 2005) εξήγαν γενικά συμπεράσματα σχετικά με τις αλληλεπιδράσεις αλλαγών (*change dependencies*) ανάμεσα στα μέλη μιας *οικογένειας προϊόντων λογισμικού* (συγκεκριμένα τις διανομές των λειτουργικών συστημάτων *xBSD*: *BSD Unix*, *OpenBSD*, *FreeBSD*, *NetBSD*).

2. Οι (Silwerski, Zimmermann, & Zeller, 2005) μελέτησαν τις συνθήκες υπό τις οποίες αλλαγές που γίνονται στον κώδικα για να διορθωθούν προηγούμενα λάθη ή για άλλους βελτιωτικούς λόγους μπορεί να προκαλέσουν με τη σειρά τους *bugs* που απαιτούν εκ νέου επιδιόρθωση (***bug-fixing changes***). Λαμβάνοντας υπόψη τα *log files* του CVS και του Bugzilla για τα έργα λογισμικού Eclipse και Mozilla, έγινε προσπάθεια να εντοπιστούν τα χαρακτηριστικά των αλλαγών αυτών που οδηγούν σε περαιτέρω σφάλματα. Τέτοια χαρακτηριστικά είναι το ποιοι και πότε έκαναν τις αλλαγές ή ποιο είδος προγραμματιστικών κλάσεων αφορούσαν αυτές. Για τον σκοπό αυτό ομαδοποιήθηκαν τα *commits* σε συναλλαγές (*transactions*) με τη μέθοδο του Κινητού Πλαισίου

(sliding-window approach – Ενότητα 2.2.2). Με εκτεταμένη χρήση κανονικών εκφράσεων ως ευριστική τεχνική στα μηνύματα (commit messages) που συνόδευαν τα commits και τα bug reports στον Bugzilla, εντοπίστηκαν ποιες αλλαγές σχετίζονται με ένα συγκεκριμένο bug. Το ποιες αλλαγές πραγματοποιήθηκαν προσδιορίστηκε μέσω του CVS diff αλγόριθμου και βελτιώθηκε με επιπλέον ευριστικές τεχνικές.

Τα συμπεράσματα στα οποία οδηγήθηκαν επιβεβαίωσαν ότι διορθώσεις (fixes) μπορεί να απαιτήσουν εκ νέου διορθώσεις και αφορούν τις ιδιότητες του λογισμικού (software properties).

Παρατηρήθηκε ότι συναλλαγές που περιέχουν fixes, έχουν τριπλάσια πιθανότητα να προκαλέσουν νέα σφάλματα σε σχέση με κοινές συναλλαγές. Επίσης, προέκυψαν ποιοτικά συμπεράσματα σε σχέση με το πώς επηρεάζεται η ποιότητα κώδικα από το χρόνο συγγραφής του. Χαρακτηριστικά οι ερευνητές εφιστούν την προσοχή με τη φράση: «*Μην προγραμματίζετε την Παρασκευή*» για το Mozilla Project, ίσως την πιο γνωστή ρήση που έχει προκύψει στο επιστημονικό οικοσύστημα της MSR.

3. Για την εκμετάλλευση των μεταδεδομένων στην MSR διαδικασία προτάθηκε και εξετάστηκε από τους (German & Hindle, 2005) μία γλώσσα ερωτημάτων (query language) με όνομα SCQL. Εξάγοντας μεταδεδομένα από τα log files του CVS, μπορούν να συνταχθούν και να απαντηθούν δομημένα ερωτήματα με τρόπο όμοιο με αυτόν που γίνεται σε μία βάση δεδομένων.
4. Μία ξεχωριστή κατηγορία αλλαγών που έχει τύχει ιδιαίτερης μελέτης είναι οι αλλαγές μικρής κλίμακας, ειδικά οι αλλαγές μίας γραμμής κώδικα (one line change). Το αντικείμενο της μελέτης των (Purushoththaman & Perry, 2004) ήταν η επίδραση τέτοιων αλλαγών σε πιθανά σφάλματα του λογισμικού, οι σχέσεις ανάμεσα σε αυτές τις αλλαγές και ο λόγος της πραγματοποίησής τους. Για να το πετύχουν αυτό διαχώρισαν τις επεμβάσεις στον κώδικα σε τρεις επιμέρους κατηγορίες: διόρθωση, προσαρμογή και τελειοποίηση. Τα πειραματικά δεδομένα αντλήθηκαν από ένα εμπορικό τηλεπικοινωνιακό σύστημα switching και οδήγησαν τελικά σε στατιστικού τύπου παρατηρήσεις στα ερωτήματα που τέθηκαν.
5. Η εξέλιξη μιας διανομής λογισμικού (δηλαδή ενός πακέτου προϊόντων λογισμικού) αναλύθηκε από τους (Robles, Gonzalez-Barahona, Michlmayr, & Amor, 2006) δίνοντας έμφαση στα ποσοτικά στοιχεία, όπως τον αριθμό των πακέτων, των γραμμών κώδικα και τον όγκο των αρχείων σε μία σειρά διαδοχικών εκδόσεων της διανομής του διαδεδομένου λειτουργικού συστήματος Debian Linux. Για τη διανομή αυτή μελετήθηκαν 5 σταθερές εκδόσεις (από 2.0 έως 3.1) σε χρονικό διάστημα 7 ετών. Το αρχείο Sources.gz ήταν το αντικείμενο επεξεργασίας για μία σειρά μετρικών. Την εργασία αυτή ανέλαβε το πρόγραμμα SLOCCount. Τα αποτελέσματα είναι ιδιαίτερα ενδιαφέροντα καθώς, πέρα από στατιστικές παρατηρήσεις που αφορούν το Debian αυτό καθαυτό (λ.χ. διπλασιασμός LOC ανά δύο χρόνια και κυριαρχία των μικρών πακέτων στη διανομή), προέκυψαν και γενικού ενδιαφέροντος συμπεράσματα, όπως το ότι το μέγεθος των προγραμμάτων που είναι γραμμένα σε διαδικαστικές γλώσσες (procedural) είναι μεγαλύτερο από αυτό των αντίστοιχων σε αντικειμενοστρεφείς γλώσσες.
6. Με ανάλογο τρόπο οι Bieman & Dinh-Trong (2005) εργάστηκαν για την επαλήθευση 5 υποθέσεων σχετικών με την επιτυχή ανάπτυξη λογισμικού κώδικα που εισήγαγαν με μορφή ερωτημάτων οι

(Mockus, Fielding, & Herbsleb, 2002). Οι τελευταίες εμπειρικές υποθέσεις (βασισμένες στην ανάπτυξη των Apache και Mozilla) ελέγχθηκαν στο CVS του λειτουργικού συστήματος FreeBSD για να διαπιστωθεί το κατά πόσον αποτελούν γενικές τάσεις στη διαχείριση projects ανοικτού κώδικα. Επιπρόσθετοι στόχοι ήταν οι εύρεση ομοιοτήτων ανάμεσα στις διαδικασίες (processes) των παραπάνω προϊόντων και μία αρχική ματιά στη μελέτη ποιότητας λογισμικού (SQA – Software Quality Assurance). Τα μεταδεδομένα αντλήθηκαν από το ιστορικό αλλαγών (CVS log file), τη βάση δεδομένων των bugs (GNATS) και τα αρχεία επικοινωνίας με emails. Από τα ανωτέρω εκτιμήθηκαν μόνο οι αναφορές που σχετίζονταν με προβλήματα (problem reports – PRs). Αναζητήθηκαν οι συγγραφείς τους, ο αριθμός των commits/χρήστη, και το σύνολο των αλλαγών που έγιναν με εργαλεία που ανέπτυξαν οι ερευνητές. Τα αποτελέσματα επαληθεύουν τις 3 από τις 5 υποθέσεις του πειράματος.

Παρατηρείται λοιπόν, ότι το πειραματικό ενδιαφέρον μπορεί να οδηγήσει σε επιβεβαίωση θεωρητικών υποθέσεων και με τη βοήθεια της MSR διαδικασίας.

2.4.2 Προσεγγίσεις ανάλυσης διαφορών ανάμεσα στις εκδόσεις λογισμικού

Ο τρόπος με τον οποίο μεταβαίνει ένα προϊόν λογισμικού από τη μία έκδοση στην επόμενη, όπως έχει ήδη αναφερθεί, καταγράφεται στο Σύστημα Διαχείρισης Εκδόσεων του (πολύ συχνά το CVS). Στο σύστημα CVS αποθηκεύονται οι διαφοροποιήσεις από έκδοση σε έκδοση, όπως αυτές έχουν προκύψει από τα commits του κάθε προγραμματιστή. Οι αλλαγές αυτές αποτυπώνονται σε επίπεδο αρχείου και γραμμής κώδικα. Η αποτύπωση αυτή δεν είναι επιθυμητή από τον ερευνητή που θέλει να εξάγει συμπεράσματα για το λογισμικό που μελετά σε ένα ενδιάμεσο επίπεδο: αυτό των κλάσεων και των μεθόδων που χρησιμοποιούνται από τους προγραμματιστές.

Έχουν προταθεί, λοιπόν, μέθοδοι ώστε οι αλλαγές να κατατάσσονται συντακτικά (δηλαδή με βάση τη θέση της αλλαγής στον κώδικα) και σημασιολογικά (δηλ. με βάση της λειτουργικής τοποθέτησης του αντικειμένου της αλλαγής ανάμεσα στα υπόλοιπα αντικείμενα του προγράμματος). Με αυτόν τον τρόπο επιτυγχάνεται η εποπτεία τόσο σε χαμηλό όσο και υψηλό επίπεδο σημασιολογικά, το λεγόμενο *granularity*. Χαρακτηριστικά παραδείγματα αυτής της κατηγορίας προσεγγίσεων αποτελούν τα παρακάτω:

1. Για τη σημασιολογική ανάλυση ενός ιστορικού εκδόσεων (version history) αναπτύχθηκε το εργαλείο DEX από τους (Raghavan, Rohana, Leon, Podgurski, & Augustin, 2004). Κάθε έκδοση αναπαρίσταται με έναν Αφηρημένο Σημασιολογικό Γράφο (*Abstract Semantic Graph*). Στη συνέχεια εφαρμόζεται ένας ευριστικός αλγόριθμος που συγκρίνει ανά δύο τους γράφους στους κόμβους τους. Καταγράφει διαγραφές, προσθήκες, ανακατατάξεις ή τροποποιήσεις σε scripts, τα οποία περιέχουν όλη την πληροφορία των αλλαγών, και πάνω σε αυτά τίθενται όλα τα ερωτήματα, και τα οποία μπορεί να αφορούν οντότητες του κώδικα. Ο αλγόριθμος διαφοροποίησης τρέχει σε χρόνο πολυωνυμικό ως προς τον αριθμό των κόμβων του γράφου και εφαρμόστηκε στο ιστορικό εκδόσεων μεγάλων προγραμμάτων ανοικτού κώδικα, του GNU / gcc και του Apache server.

2. Εναλλακτικά, αναλύοντας συντακτικά τον κώδικα, οι αλλαγές που συμβαίνουν σε ένα πρόγραμμα μπορούν να καταγραφούν εφόσον χρησιμοποιηθεί ένα *Αφηρημένο Συντακτικό Δένδρο* (Abstract Syntax Tree – AST). Με τη σύνταξη των δένδρων αυτών για κάθε έκδοση του προγράμματος, ξεκινά μία διαδικασία αντιστοίχισης ώστε να αναγνωριστούν οι αντίστοιχες οντότητες (μεταβλητές, τύποι και μέθοδοι) των δύο υπό σύγκριση ASTs. Με βάση το ορισμένο σετ κανόνων, από έκδοση σε έκδοση οι οντότητες χαρακτηρίζονται ως καινούριες, διαγραφμένες ή τροποποιημένες. Σκοπός αυτής της μελέτης ήταν να υποστηριχθεί η αναβάθμιση λογισμικού χωρίς να σταματήσει η εκτέλεσή του. Απαντώντας σε ερωτήματα βάσει της ιστορίας των προγραμμάτων διαπιστώθηκε πως ο Apache είναι καλύτερος υποψήφιος για δυναμική αναβάθμιση σε σχέση με το OpenSSH.
3. Ενώ είναι δυνατή η σημασιολογική και η συντακτική ανάλυση ξεχωριστά, οι τεχνικές μπορούν και να συνδυαστούν. Οι (Dig & Johnson, 2006) και (Dig, Comertoglu, Marinov, & Johnson, 2006) ανέλυσαν αρχικά συντακτικά τον κώδικα για να εξάγουν οντότητες πηγαίου κώδικα και στη συνέχεια κάνοντας χρήση της *κωδικοποίησης Shingles* εντόπισαν ζεύγη οντοτήτων ανάμεσα σε μια παλιά και μία νεότερη έκδοση που μπορεί να προέκυψαν από βελτιώσεις σχεδίασης (refactorings). Ως Refactorings ορίζονται οι αλλαγές σε ένα τμήμα κώδικα που, ενώ επηρεάζουν την εσωτερική του δομή, δεν αλλάζουν το πώς συμπεριφέρεται και το πώς γίνεται ορατός από το υπόλοιπο του προγράμματος. Τα υποψήφια αυτά ζεύγη φιλτράρονται συγκρίνοντας το πώς καλούνται από το υπόλοιπο πρόγραμμα και στις δύο εκδόσεις του. Αυτή η μεθοδολογία υλοποιήθηκε σε plugins του Eclipse IDE για επτά διαφορετικούς τύπους refactoring. Τα αποτελέσματα ήταν ιδιαίτερα ενθαρρυντικά, συγκεκριμένα τόσο η ακρίβεια (precision), όσο και η ανάκληση (recall) ήταν πάνω από 0.85 (ή 85%).
Σε μία ελαφρώς διαφορετική έρευνα (Dig & Johnson, 2006) μελετήθηκε το κατά πόσον μία βελτίωση σχεδίασης σε κάποιο API μπορεί να αλλάξει τη συμπεριφορά στο επίπεδο του πελάτη (client) του API. Εφόσον μία αλλαγή καταστρέφει ή όχι την προς-τα-πίσω συμβατότητα (backwards compatibility) θεωρείται breaking ή non-breaking. Η ερευνητική προσπάθεια εκτός από τις διαδοχικές εκδόσεις πηγαίου κώδικα, συνέκρινε και την τεκμηρίωση, τις σημειώσεις έκδοσης (release notes), ενώ περιέλαβε και μη-αυτόματη εξέταση των διαφοροποιήσεων. Η εφαρμογή έγινε στα Eclipse framework, struts, jHotDraw, log4j, mortgage. Τα αναμενόμενα αποτελέσματα έδειξαν υψηλή πιθανότητα για breaking refactorings.
4. Τέλος, την ομοιότητα Java κλάσεων χρησιμοποιώντας αλγόριθμους που χειρίζονται δομές δένδρων εξέτασαν οι (Sager, Bernstein, Pinzger, & Kiefer, 2006). Συγκεκριμένα, και πάλι αναπτύχθηκαν διαδοχικές εκδόσεις του πηγαίου κώδικα σε Abstract Syntax Trees και εφαρμόστηκαν πάνω τους μια σειρά από αλγόριθμους (bottom-up maximum common subtree isomorphism, top-down maximum common subtree isomorphism και tree-edit distance). Ο τελευταίος αποδείχθηκε πιο αποτελεσματικός και χρησιμοποιήθηκε στις εκδόσεις 3.0 και 3.1 του Eclipse compare plugin.

2.4.3 Προσεγγίσεις ανίχνευσης συχνών μοτίβων (Frequent-pattern mining)

Από τα δεδομένα που βρίσκονται σε αποθήκες λογισμικού με τεχνικές εξόρυξης δεδομένων σκοπός είναι να αναγνωρισθούν μοτίβα, τάσεις και κανόνες που δεν είναι ορατά με απλή παρατήρηση του συνόλου των δεδομένων. Η εφαρμογή κλασικών τεχνικών της εξόρυξης κανόνων συσχέτισης (association rule mining) μπορεί να οδηγήσει σε αναγνώριση οντοτήτων λογισμικού που συνδέονται μεταξύ τους. Δηλαδή, περιπτώσεις στις οποίες αλλαγή σε μία οντότητα επιφέρει αλλαγή σε μία ή περισσότερες άλλες.

Στην περίπτωση αυτή, η εξόρυξη συσχέτισης γίνεται σε δεδομένα που έχουν προκύψει από συναλλαγές (transactional), όπως κατεχοχήν είναι τα δεδομένα πηγαίου κώδικα, τα δεδομένα από Συστήματα Διαχείρισης Εκδόσεων (CVS), τα δεδομένα σε συστήματα καταγραφής bugs και τα μεταδεδομένα αυτών.

Ένα χαρακτηριστικό παράδειγμα εξόρυξης μοτίβων στην MSR είναι η προσπάθεια να εντοπιστούν αλλαγές που επιφέρουν αλλαγές (change couplings). Αυτές οι προσεγγίσεις κυριαρχούν από την αρχή της ερευνητικής προσπάθειας στο αντικείμενο της MSR. Ενδεικτικά, οι (Zimmermann, Weißgerber, Diehl, & Zeller, 2004) παρουσίασαν το εργαλείο ROSE (Eclipse IDE plugin), το οποίο διατρέπει τον πηγαίο κώδικα και αναγνωρίζει οντότητες ανά γραμμή. Στη συνέχεια εντοπίζει στην αποθήκη λογισμικού του CVS αλλαγές που έχουν πραγματοποιηθεί εντός συγκεκριμένου χρονικού πλαισίου (sliding window) και τις ομαδοποιεί ως συναλλαγές (transactions). Η προσέγγιση αυτή εισάγει και τον κατάλληλο φορμαλισμό για τον χειρισμό των αλλαγών και των συναλλαγών

2.5 Το υποκείμενο του πειράματος

Έχει γίνει πλέον σαφές ότι ο χώρος Εξόρυξης Γνώσης σε Αποθήκες Λογισμικού είναι ιδιαίτερα ευρύς και επιτρέπει μία πλειάδα προσεγγίσεων. Από τη μελέτη της σχετικής βιβλιογραφίας προκύπτει ότι η κύρια πηγή δεδομένων στην οποία έχει εστιάσει η επιστημονική προσοχή είναι τα δεδομένα που αφορούν έμμεσα ή άμεσα τον πηγαίο κώδικα. Δηλαδή, ο κύριος όγκος της ερευνητικής προσπάθειας συγκεντρώνεται στα δεδομένα που περιέχονται σε Συστήματα Διαχείρισης Εκδόσεων Λογισμικού, όπως το CVS. Η εκμετάλλευση δεδομένων που αφορούν την Καταγραφή των Προγραμματιστικών Σφαλμάτων (Bugs) είναι συγκριτικά πολύ μικρότερη και σχεδόν ποτέ αυτόνομη. Στη συντριπτική πλειοψηφία των περιπτώσεων, τα δεδομένα που προέρχονται από αναφορές σφαλμάτων σε Συστήματα Καταγραφής (Bug Tracking Systems), όπως το Bugzilla, χρησιμοποιούνται επικουρικά για τα κύρια δεδομένα που αφορούν πηγαίο κώδικα.

Με αυτό ως δεδομένο, η παρούσα εργασία επιχειρεί να προσεγγίσει το πρόβλημα από άλλη οπτική. Ξεκινώντας από τις αναφορές σφαλμάτων που βρίσκονται στη Βάση Δεδομένων ενός ΣΚΣ, η παρούσα εργασία στοχεύει στον εντοπισμό και την κατηγοριοποίηση συχνά εμφανιζόμενων σφαλμάτων (bugs) κατά τη διάρκεια της ανάπτυξης ενός προϊόντος λογισμικού. Η χρησιμότητα της αναγνώρισης τέτοιων κατηγοριών bugs σε ένα ΣΚΣ είναι προφανής:

- αναδεικνύονται τα επαναλαμβανόμενα προγραμματιστικά λάθη ή οι αδυναμίες στη διαδικασία ανάπτυξης του λογισμικού σε μεγάλο χρονικό διάστημα (ενδεχομένως και ολόκληρο το χρόνο ζωής του)
- οι προγραμματιστές εκπαιδεύονται, αποκτώντας επίγνωση των λαθών κατά τη συγγραφή πηγαίου κώδικα και αναγνωρίζοντας στη συνέχεια μεθόδους επίλυσής τους με αναφορά το Σύστημα Διαχείρισης Εκδόσεων
- ο Υπεύθυνος του Έργου (team leader) αναγνωρίζει έγκαιρα τα προβλήματα, εφιστά την προσοχή και βελτιώνει τη συλλογική λειτουργία της προγραμματιστικής ομάδας, ενώ
- ένα Περιβάλλον Ολοκληρωμένης Ανάπτυξης (IDE) λογισμικού μπορεί να σημειώνει πιθανά bugs ή/και να προτείνει λύσεις, ιδανικά σε πραγματικό χρόνο.

Τα παραπάνω παραδείγματα επιδεικνύουν τις ευεργετικές επιπτώσεις που μπορεί να έχει η συγκεκριμένη λειτουργία στη μεθοδολογία ανάπτυξης και ελέγχου ποιότητας (software development, software quality assurance) και κατά συνέπεια στο συνολικό κύκλο ζωής (lifecycle) του λογισμικού.

Για να επιτευχθεί ο παραπάνω στόχος επιλέχθηκε ως αντικείμενο ανάλυσης ένα προϊόν λογισμικού με τα εξής χαρακτηριστικά:

- Σημαντικό χρονικό διάστημα ανάπτυξης και αρκετές εκδόσεις του λογισμικού
- Σχετικά σταθερή βάση προγραμματιστών του project (regular contributors)
- Εύχρηστο, ενημερωμένο και ολοκληρωμένο σύστημα καταγραφής σφαλμάτων και εκδόσεων (bugtracking and version control system, αντίστοιχα)

Επιπλέον, για αποφευχθούν ζητήματα ιδιοκτησίας και διαθεσιμότητας των δεδομένων προκρίθηκε η επιλογή λογισμικού ανοικτού κώδικα (ΛΑΚ – open source software) και όχι εμπορικού (proprietary software). Άλλωστε, είναι κοινός τόπος ότι τα πιο επιτυχημένα παραδείγματα συνεργατικού λογισμικού αφορούν προϊόντα ΛΑΚ που διαθέτουν ισχυρή κοινότητα προγραμματιστών που τα υποστηρίζουν διαρκώς, χωρίς να ικανοποιείται κανένα συνηθισμένο κριτήριο γεωγραφικής εγγύτητας, γλωσσικής ή εργασιακής συνάφειας.

Από το πλήθος των διαθέσιμων επιλογών και λαμβάνοντας υπόψη τα παραπάνω κριτήρια, επιλέχθηκε να μελετηθεί η Ruby On Rails (Ruby on Rails), μία πλατφόρμα γρήγορης ανάπτυξης προγραμμάτων για το διαδίκτυο (rapid development web framework), η οποία βασίζεται στη δημοφιλή γλώσσα διαδικτυακού προγραμματισμού Ruby.

Η ανάπτυξη του Ruby On Rails (RoR) project γίνεται συλλογικά, χρησιμοποιώντας το σύστημα Διαχείρισης εκδόσεων Git και το σύστημα καταγραφής σφαλμάτων Lighthouse (Παράρτημα Α).

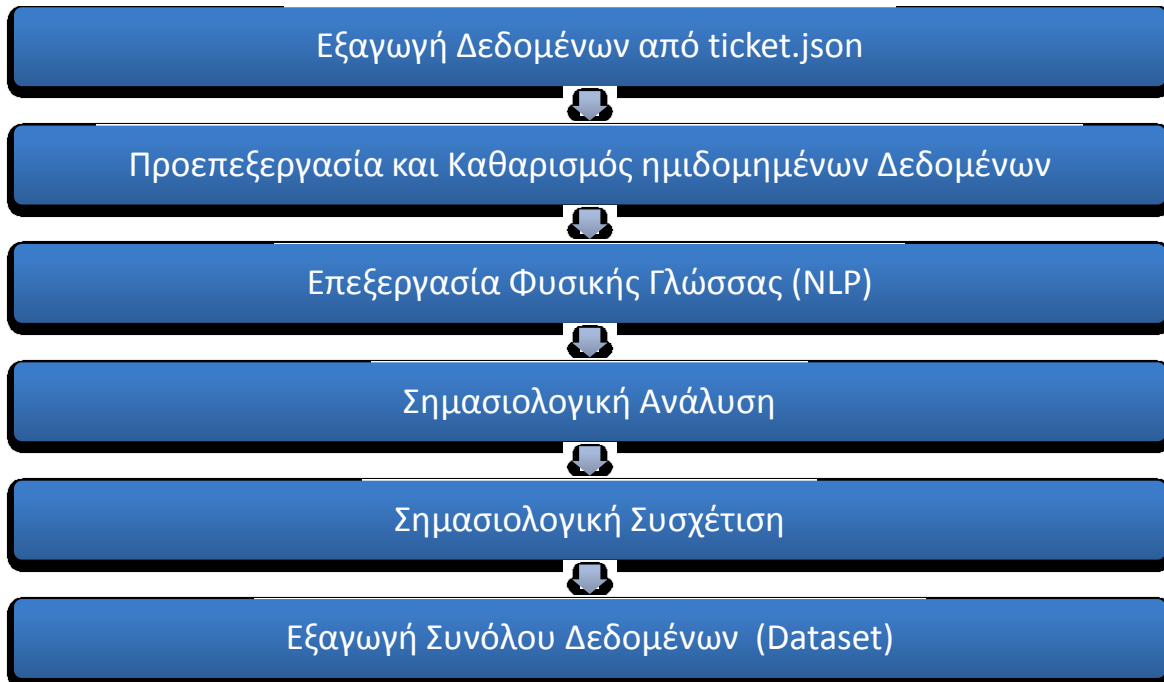
Κεφάλαιο 3: Μεθοδολογία I

Επεξεργασία Δεδομένων – Φυσικής Γλώσσας

3.1 Περιγραφή της Μεθοδολογίας

3.1.1 Τα βασικά στάδια της μεθοδολογίας

Η μεθοδολογία που αναπτύχθηκε με στόχο την αναγνώριση και κατηγοριοποίηση σφαλμάτων σε αποθήκες λογισμικού αποτελείται από 6 στάδια. Ξεκινώντας από δεδομένα (tickets) διαμορφωμένα σε αρχεία τύπου .json (ή .xml), όπως αυτά παρέχονται πρωτογενώς από την αποθήκη λογισμικού και το σύστημα καταγραφής σφαλμάτων, επεξεργαζόμαστε την πληροφορία και τα μεταδεδομένα αυτής ώστε να δημιουργηθεί ένα αρχείο έτοιμο για εφαρμογή τεχνικών εξόρυξης δεδομένων.



Σχήμα 3.1-1 Τα βασικά στάδια της μεθοδολογίας

Συνοπτικά, η διαδικασία, όπως φαίνεται και στο Σχήμα 3.1-1, αναλύεται στα εξής:

- Την **Εξαγωγή των δεδομένων από κάθε ticket** και την αντιστοίχιση των τιμών που λαμβάνουν οι ιδιότητές του σε ξεχωριστές οντότητες. Η διαδικασία αυτή περιγράφεται αναλυτικά στην παράγραφο 3.2 - Εξαγωγή Δεδομένων.
- Την **Προεπεξεργασία και τον καθαρισμό δεδομένων που είναι μερικώς δομημένα** (semi-structured). Τέτοια δεδομένα αποτελούν Δεδομένα του Παγκόσμιου Ιστού που έχουν σημειωθεί σύμφωνα με το HTML ή άλλο πρότυπο. Η διαδικασία του καθαρισμού (data cleansing) περιλαμβάνει την απόρριψη της περιττής πληροφορίας και τη μείωση του χώρου των δεδομένων. Η διαδικασία αυτή περιγράφεται αναλυτικά στην παράγραφο 0 - O org.json χρησιμοποιήθηκε για την ανάκτηση των πληροφοριών συγκεκριμένων πεδίων ιδιοτήτων του *ticket.json*, αντί του *Jackson JSON Processor*, ο οποίος χρησιμοποιήθηκε για την ανάκτηση ολόκληρων των *.json tickets*.
-
- Την **Επεξεργασία σε επίπεδο φυσικής γλώσσας (Natural Language Processing - NLP)**. Η διαδικασία αυτή έχει ως σκοπό την αναγνώριση της συντακτικής συσχέτισης και της γραμματικής θέσης ανάμεσα σε μέλη (λέξεις) ενός σώματος κειμένου. Το αποτέλεσμα της είναι διανύσματα λέξεων (word vectors) που εμπεριέχουν την ουσία της πληροφορίας του κειμένου. Η διαδικασία αυτή περιγράφεται αναλυτικά στην παράγραφο 3.4 - Επεξεργασία Φυσικής Γλώσσας (*Natural Language Processing*).
- Τη **Σημασιολογική ανάλυση** (*semantic analysis*) του χώρου των διανυσμάτων λέξεων και στοχεύει στην ομαδοποίησή τους σε συλλογές κοινού θέματος (*topic*) με τη χρήση στατιστικών σημασιολογικών τεχνικών μοντέλου θέματος (*topic modeling*). Περιγράφεται αναλυτικά στην παράγραφο 4.1 - 4.1 Σημασιολογική Ανάλυση 3.4.4
- Τη **Σημασιολογική συσχέτιση** (*semantic association*) ανάμεσα στα θέματα του προηγούμενου βήματος και γνωστών από την Τεχνολογία Λογισμικού Σφαλμάτων (*bugs*) και Μετρικών Ποιότητας Λογισμικού (*Software Quality Metrics*). Περιγράφεται αναλυτικά στην παράγραφο 4.2 - Σημασιολογική Συσχέτιση.
- Την **Εξαγωγή του Συνόλου των Δεδομένων** (*Dataset Extraction*) μετά από τα παραπάνω στάδια σε εκμεταλλεύσιμη μορφή από το γνωστό πρόγραμμα Εξόρυξης Δεδομένων *Weka*. Περιγράφεται στην Παράγραφο 4.3 - Προετοιμασία τελικού Συνόλου Δεδομένων.

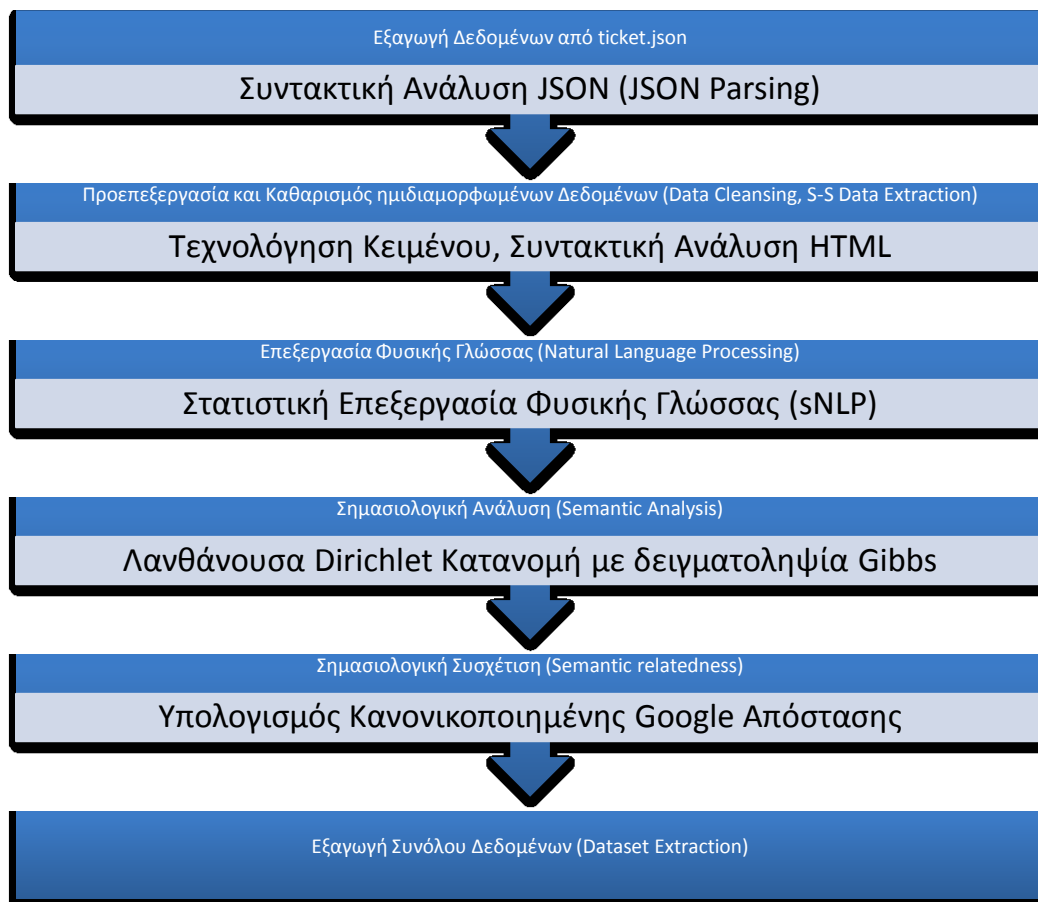
3.1.2 Τεχνικές που υιοθετήθηκαν

Σε κάθε στάδιο της διαδικασίας χρησιμοποιούνται καλά δοκιμασμένες τεχνικές, οι οποίες προέρχονται από τις περιοχές της Ανάκτησης Πληροφορίας (*Information Retrieval*), της Υπολογιστικής Γλωσσολογίας (*Computational Linguistics*), της Εξόρυξης Δεδομένων από Κείμενο (*Text Mining*), της Μηχανικής Εκμάθησης (*Machine Learning*) και της Στατιστικής Σημασιολογικής Ανάλυσης (*Probabilistic latent semantic analysis*). Πιο συγκεκριμένα:

- Για την εξαγωγή δεδομένων από *.json* αρχεία χρησιμοποιούνται συντακτικοί αναλυτές (*JSON parsers*) που αναλαμβάνουν την ανάγνωση ιδιοτήτων και την αντιστοίχιση σε *Java Objects*. Από τις διαθέσιμες εναλλακτικές επιλέχθηκε η εξειδικευμένη βιβλιοθήκη συναρτήσεων συντακτικών αναλυτών *Jackson JSON Processor* (<http://jackson.codehaus.org/>).

- Για την προεπεξεργασία των δεδομένων (Data Preprocessing) και, πιο συγκεκριμένα, για δεδομένα σε μορφή HTML χρησιμοποιούνται τεχνικές καθαρισμού του κειμένου μέσω συντακτικών αναλυτών που εργάζονται πάνω σε HTML δεδομένα (HTML Parsers) (HtmlCleaner - (<http://htmlcleaner.sourceforge.net/>))
- Για την επεξεργασία φυσικής γλώσσας (NLP) χρησιμοποιήθηκε το στατιστικό και στοχαστικό μοντέλο ανάλυσης το οποίο αντιμετωπίζει επιτυχώς τη δυσκολία ανάλυσης μία μακράς πρότασης κειμένου.
- Για τη σημασιολογική ανάλυση και την ομαδοποίηση των διανυσμάτων λέξεων επιλέχθηκε η Λανθάνουσα κατά Dirichlet Κατανομή (LDA) στην τροποποιημένη της μορφή, στην οποία τα δεδομένα υπόκεινται σε δειγματοληψία Gibbs. Η εκτίμηση γίνεται χρησιμοποιώντας μετρική σημασιολογικής ομοιότητας (Semantic similarity).
- Τέλος, ως μετρική σημασιολογικής ομοιότητας και, κατά συνέπεια, αναγνώρισης επιλέχθηκε η επίδοση στην Κανονικοποιημένη Google Απόσταση (Normalized Google Distance – NGD) (Cilibrasi & Vitanyi, 2005).

Η αντιστοίχιση τεχνικών και εργαλείων με τα στάδια της μεθοδολογίας φαίνεται στο Σχήμα 3.1-2 .



Σχήμα 3.1-2- Τεχνικές που χρησιμοποιούνται σε κάθε βήμα της ανάλυσης

3.2 Εξαγωγή Δεδομένων

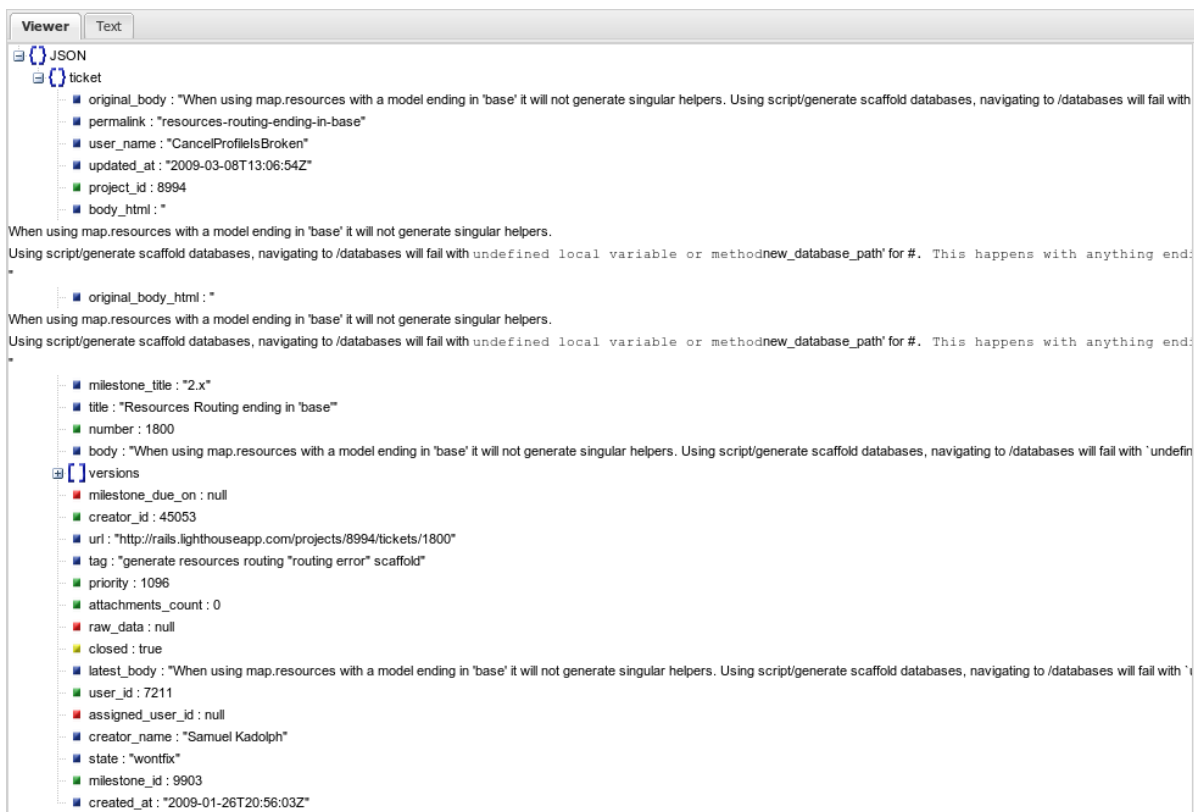
Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, το προϊόν λογισμικού που επελέγη για να μελετηθεί είναι το **Ruby On Rails**, μία πλατφόρμα γρήγορης ανάπτυξης προγραμμάτων για το web (rapid development web framework) που βασίζεται την ανερχόμενη γλώσσα προγραμματισμού Ruby.

Η ανάπτυξη του Ruby On Rails (RoR) γίνεται συλλογικά εντός ενός οικοσυστήματος χρηστών που χρησιμοποιούν την υπηρεσία διαχείρισης εκδόσεων github.com και την υπηρεσία καταγραφής σφαλμάτων Lighthouse. Κάθε ενέργεια/σφάλμα καταγράφεται σε ένα ticket, τη δομή του οποίου συζητούμε παρακάτω.

3.2.1 Δομή ενός ticket

Ένα ticket, είτε αναπαρίσταται σε XML μορφή είτε σε JSON, αποτελεί μια δομή δεδομένων (data structure) που εμπεριέχει στοιχεία σχετικά με το τι αφορά, πού ακριβώς απαντάται στον κώδικα, ποιοι είναι οι εμπλεκόμενοι χρήστες κ.ά.

Η αντιμετώπιση ενός σφάλματος στον πηγαίο κώδικα δεν αποτελεί μία στατική διαδικασία ούτε γενικά πανάκεια, καθώς ενδέχεται να προκύψουν νέα σφάλματα ή η επίλυση να μην είναι πλήρης. Στο αρχείο του ticket, περιέχονται διαφορετικές εκδόσεις όπως αυτές προκύπτουν από μεταγενέστερες καταγραφές, που εμπλουτίζουν τις πληροφορίες που διατίθενται.



Εικόνα 3.2-1 Γραφική Αναπαράσταση ticket σε μορφή JSON

Τα πεδία (fields) ή ιδιότητες (attributes) που περιγράφουν ένα ticket μπορούν να χωριστούν ως εξής:

- **Στα βασικά πεδία, που ορίζουν κάθε διαφορετικό ticket.** Ο αύξων αριθμός (<number>), ο τίτλος του (<title>), σύντομη μνημονική διεύθυνση (<permalink>) και η διεύθυνσή του στο περιβάλλον του Lighthouse (<url>). Ακόμη, περιέχονται στοιχεία που αφορούν το project (<project-id>) και το σε ποιο σκέλος ανάπτυξης του project διαπιστώθηκε το σφάλμα (<milestone-due-on>, <milestone-title>, <milestone-id>)
- **Στα αναφερόμενα στους χρήστες πεδία, που εμπλέκονται στην καταγραφή και την επίλυση των σφαλμάτων.** Το αναγνωριστικό του χρήστη που ανέφερε το ticket (<creator-id>), αυτού στον οποίον ανατέθηκε (<assigned-user-id>) και αυτού που ενημερώνει το ticket (<user-id>, <username>)
- **Στις ιδιότητες που το περιγράφουν.** Την προτεραιότητά του (<priority>), την κατάστασή του ως προς το επίπεδο επίλυσης (<state>), μια περιγραφή μέσω ετικετών (<tag>), αν είναι κλειστό ή όχι (<closed>), μια αναλυτική περιγραφή σε απλό κείμενο και κείμενο μορφοποιημένο σε HTML (<original-body>, <original-body-html>, αντίστοιχα), τις ημερομηνίες δημιουργίας του ticket (<created-at>), καθώς και το πότε ενημερώθηκε τελευταία φορά (<updated-at>).
- **Στις πολλαπλές εκδόσεις που εμπεριέχονται στο object array,** με έκδοση <versions> και συνημμένα πεδία (<attachments-count>)

Για το κάθε πεδίο που παρουσιάζεται διαφέρει και ο τύπος των δεδομένων που αυτό μπορεί να δεχθεί. Για παράδειγμα, υπάρχουν πεδία, όπως τα <number> και <priority>, που δέχονται αριθμητικά δεδομένα (ακεραίους), άλλα που αφορούν ημερομηνίες και δέχονται ημερομηνίες σύμφωνα με το πρότυπο ISO 8601 (πχ 2010-02-26T22:43:47+00:00) και άλλα που δέχονται αλφαριθμητικές αναπαραστάσεις (string) που μπορεί να είναι σε μορφή απλού κειμένου (πχ <original-body>) ή σε μία δομημένη μορφή πλαισιωμένη από HTML tags (πχ <original-body-html>).

Στην παρούσα εργασία, η προσπάθεια που γίνεται για την εξαγωγή χρήσιμης πληροφορίας επικεντρώνεται κυρίως στην περιγραφή του ticket από το χρήστη που το αναφέρει. Συνεπώς, ένα από τα πεδία στα οποία δίνεται ιδιαίτερο βάρος είναι το **<original-body-html>¹** το οποίο περιέχει **μερικώς δομημένο κείμενο (semi-structured (web) data)**. Το πεδίο αυτό, όπως είναι προφανές, δεν είναι δυνατόν να γίνει κατανοητό χωρίς αποδιαμόρφωση, η οποία γίνεται με τη βοήθεια τεχνικών **Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing)**. Οι τεχνικές αυτές θα συζητηθούν εκτεταμένα στην παράγραφο 3.4 - Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing).

¹ Η σημασία της ύπαρξης μερικώς δομημένων δεδομένων (html formatted) θα φανεί στο στάδιο της προεπεξεργασίας πριν προχωρήσουμε στην ανάλυση κειμένου με εργαλεία Επεξεργασίας Φυσικής Γλώσσας (Ενότητα 3.4)

```

<?XML version="1.0" encoding="UTF-8"?>
<tickets type="array">
  <ticket>
    <assigned-user-id type="integer">83</assigned-user-id>
    <attachments-count type="integer">7</attachments-count>
    <closed type="boolean">false</closed>
    <created-at type="datetime">2010-01-21T15:16:13+00:00</created-at>
    <creator-id type="integer">65576</creator-id>
    <milestone-due-on type="datetime" nil="true"></milestone-due-on>
    <milestone-id type="integer">27004</milestone-id>
    <number type="integer">3765</number>
    <permalink>missing-shallow-routes-in-new-router-dsl</permalink>
    <priority type="integer">2</priority>
    <project-id type="integer">8994</project-id>
    <raw-data type="binary" nil="true" encoding="base64"></raw-data>
    <state>open</state>
    <tag>shallow_routes patch rails3 router</tag>
    <title>Missing shallow routes in new router DSL</title>
    <updated-at type="datetime">2010-02-26T22:43:47+00:00</updated-at>
    <user-id type="integer">83</user-id>
    <user-name>DHH</user-name>
    <creator-name>stevestmartin</creator-name>
    <assigned-user-name>DHH</assigned-user-name>
    <url>http://rails.Lighthouseapp.com/projects/8994/tickets/3765</url>
    <milestone-title>3.0</milestone-title>
    <original-body>It seems that shallow routes have been removed from the new router dsl. I
    attempted to dig through the tests source, and havent seen anything leading me to believe its
    implemented. Is this an oversight, or planned removal?</original-body>
    <latest-body>It seems that shallow routes have been removed from the new router dsl. I
    attempted to dig through the tests source, and havent seen anything leading me to believe its
    implemented. Is this an oversight, or planned removal?</latest-body>
    <original-body-html>&lt;div&gt;&lt;p&gt;It seems that shallow routes have been removed
    from the new router dsl. I attempted to dig through the tests source, and haven't seen
    anything leading me to believe its implemented.&lt;/p&gt;&lt;p&gt;Is this an oversight, or
    planned removal?&lt;/p&gt;&lt;/div&gt;</original-body-html>
  </ticket>

```

Σχήμα 3.2-1- Παράδειγμα απεικόνισης ενός ticket σε μορφή XML

3.2.2 Το πρότυπο JSON

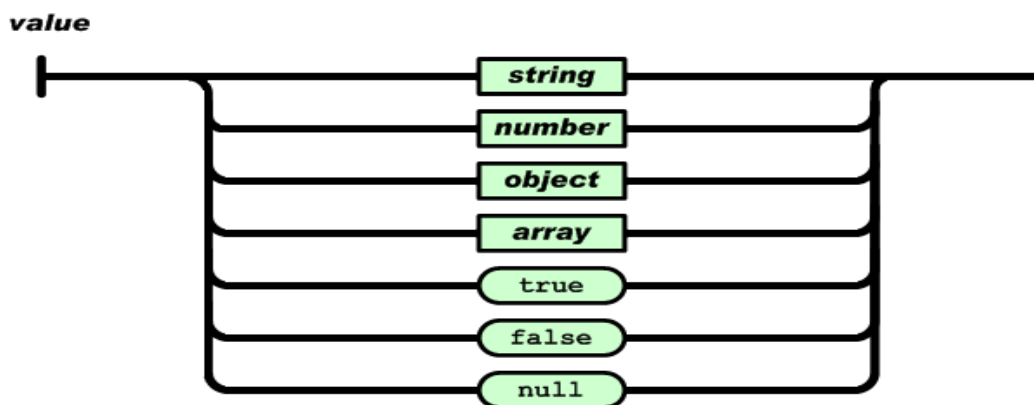
Όπως έχει ήδη αναφερθεί, το Lighthouse API δίνει τη δυνατότητα πρόσβασης στα δεδομένα τόσο σε μορφή XML (Extensible Markup Language), όσο και σε JSON. Για τις ανάγκες τις παρούσας εργασίας, επιλέχθηκε να ανακτηθούν τα tickets το καθένα σε ξεχωριστό αρχείο ticket.json για λόγους ευκολίας στην πρόσβαση και μεγέθους (η αποθηκευμένη πληροφορία καταλαμβάνει κωδικοποιημένη σε JSON, εν γένει, λιγότερο χώρο από ότι σε XML).

Το JSON (JavaScript Object Notation), είναι ένα ανοικτό πρότυπο ανταλλαγής πληροφορίας που προέκυψε ως επέκτασης της γλώσσας προγραμματισμού JavaScript για την αναπαράσταση σε σειριακή μορφή απλών δομών δεδομένων και συλλογών που καλούνται Αντικείμενα (objects).

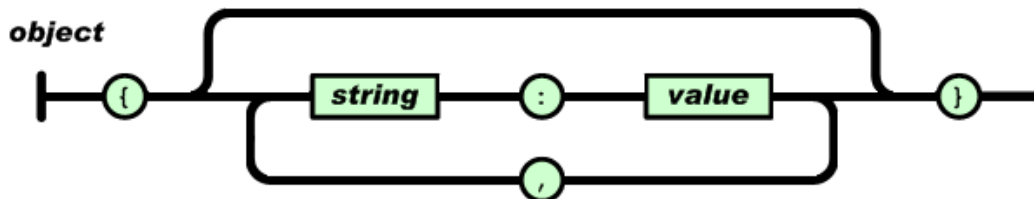
Αρχεία με κατάληξη .json που συμμορφώνονται στο πρότυπο (RFC 4627) συνηθέστατα μεταφέρουν σε δικτυακό περιβάλλον δεδομένα από τον διακομιστή στον πελάτη (server to client), ως εναλλακτική στη γλώσσα σήμανσης XML.

Το πρότυπο αναγνωρίζει βασικούς τύπους δεδομένων:

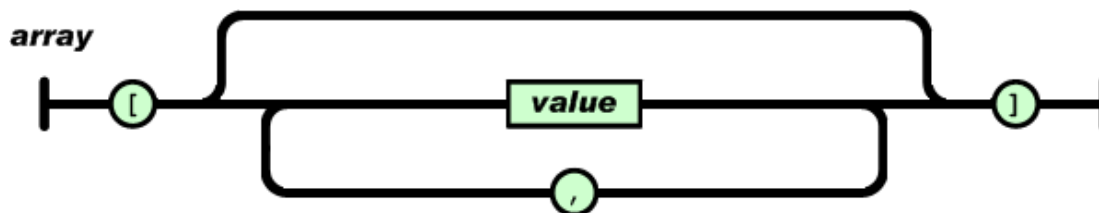
- ακεραίους και πραγματικούς *Αριθμούς*
- Αλφαριθμητικές ακολουθίες (*strings*) σε Unicode κωδικοποίηση
- Δυαδική επιλογή (*Boolean*),
- Ταξινομημένες Συλλογές (*Arrays*) με κόμμα διακεκριμένων τιμών που περικλείονται από αγκύλες, και
- Αντικείμενα (*Objects*), που είναι συλλογές από ζεύγη ονομάτων/τιμών, εκ των οποίων η μία είναι ένα αλφαριθμητικό κλειδί και η άλλη η τιμή που του ανατίθεται (*key/value pairs*).



Σχήμα 3.2-2α - Οι τύποι δεδομένων που αναγνωρίζει το JSON format



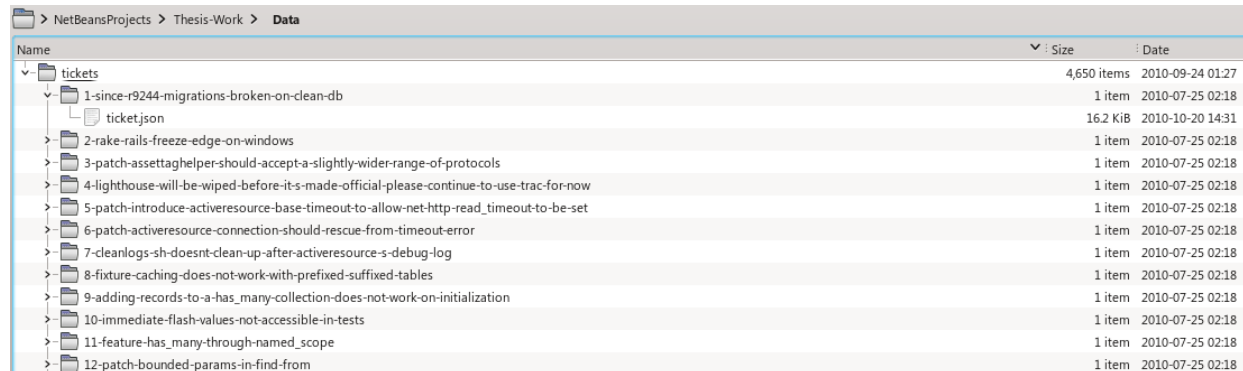
Σχήμα 3.2-3β - Απεικόνιση Αντικειμένου (object) στο JSON format



Σχήμα 3.2-4γ - Απεικόνιση Συλλογής στο JSON Format

Πρέπει να σημειωθεί ότι η απεικόνιση ενός ticket, εναλλακτικά, σε JSON ή XML πρότυπο δε διαφοροποιεί τις καθορισμένες ιδιότητές του, καθώς οι διαφορές εντοπίζονται μόνο στο περίβλημά του (container).

Το σύνολο των tickets ανακτάται σε ένα σύνολο υποκαταλόγων (subdirectories) κάθε ένας εκ των οποίων περιέχει ένα μοναδικό ticket.json (Εικόνα 3.2-2 **Error! Reference source not found.**)



Name	Size	Date
tickets	4,650 items	2010-09-24 01:27
1-since-r9244-migrations-broken-on-clean-db	1 item	2010-07-25 02:18
2-rake-rails-freeze-edge-on-windows	1 item	2010-07-25 02:18
3-patch-assettaghelper-should-accept-a-slightly-wider-range-of-protocols	1 item	2010-07-25 02:18
4-lighthouse-will-be-wiped-before-it-s-made-official-please-continue-to-use-trac-for-now	1 item	2010-07-25 02:18
5-patch-introduce-activeresource-base-timeout-to-allow-net-http-read_timeout-to-be-set	1 item	2010-07-25 02:18
6-patch-activeresource-connection-should-rescue-from-timeout-error	1 item	2010-07-25 02:18
7-cleanlogs-sh-doesnt-clean-up-after-activeresource-s-debug-log	1 item	2010-07-25 02:18
8-fixture-caching-does-not-work-with-prefixed-suffixed-tables	1 item	2010-07-25 02:18
9-adding-records-to-a-has_many-collection-does-not-work-on-initialization	1 item	2010-07-25 02:18
10-immediate-flash-values-not-accessible-in-tests	1 item	2010-07-25 02:18
11-feature-has_many-through-named_scope	1 item	2010-07-25 02:18
12-patch-bounded-params-in-find-from	1 item	2010-07-25 02:18

Εικόνα 3.2-2 Η δομή του καταλόγου Data που περιέχει τα tickets

Παράδειγμα ticket που αναπαρίσταται σε μορφή JSON μπορεί να βρεθεί στο **Error! Reference source not found.**

3.2.3 Βιβλιοθήκες Συντακτικής Ανάλυσης JSON (JSON Parsing)

Για την εξαγωγή των τιμών που λαμβάνουν οι ιδιότητες του κάθε ticket -εφόσον αυτό είναι κωδικοποιημένο σε JSON format- χρησιμοποιείται εξειδικευμένο λογισμικό Συντακτικής Ανάλυσης (Syntactic Analysis, Parsing Software). Τα συγκεκριμένα προγράμματα συνήθως διατίθενται ως βιβλιοθήκες συναρτήσεων που αναλαμβάνουν να εντοπίσουν την πληροφορία στο .json αρχείο και να την εξάγουν σε μορφή κατανοητή από άλλες συναρτήσεις.

- Τη *διευρυμένη λειτουργικότητά* τους. Πολλοί συντακτικοί αναλυτές (parser) επιδιώκουν να προσφέρουν υπηρεσίες βέλτιστης διαχείρισης ροής δεδομένων (streaming), χαρτογράφησης (mapping) σε αντικείμενα κλάσεων αντικειμενοστραφούς προγραμματισμού (object data mapping/binding) και αυτόματη δημιουργία δενδρικών δομών δεδομένων (tree model). Ζητούμενο, επίσης, είναι η μετατροπή ενός .json αρχείου στο ισοδύναμο στο επίπεδο της πληροφορίας XML (JSON to XML Mapping).
- Την *ταχύτητα* και την *ακρίβεια*, καθώς σε κρίσιμες στο πεδίο του χρόνου (time-critical) ή της αξιοπιστίας (reliable) εφαρμογές απαιτείται η μέγιστη ακρίβεια στον ελάχιστο χρόνο με το ελάχιστο υπολογιστικό κόστος.
- Την παράλληλη δυνατότητα εξαγωγής (parsing) και παραγωγής (writing) .json αρχείων.
- Τη *συμμόρφωση* (compliance) με το πρότυπο JSON.

3.2.4 Επιλογή Βιβλιοθηκών JSON Parsing

Στην παρούσα εργασία εκμεταλλευόμαστε τις δυνατότητες δύο τέτοιων Συντακτικών Αναλυτών που είναι γραμμένοι στη γλώσσα Java:

- Τον **Jackson JSON Processor** (<http://jackson.codehaus.org/>)

- και τη βασική βιβλιοθήκη **org.json** (<http://www.json.org/java/index.html>)

3.2.4.1 Ο Συντακτικός Αναλυτής Jackson JSON Processor

Ο Jackson υποστηρίζει πολλά από τα απαιτούμενα χαρακτηριστικά ενός Συντακτικού Αναλυτή, όπως περιγράφονται στην προηγούμενη παράγραφο. Επιπλέον, θεωρείται ένας από τους ταχύτερους αναλυτές σε περιβάλλον Java, υλοποιεί πλήρως την αντιστοίχιση δεδομένων σε Αντικείμενα Κλάσεων της Java (Java Beans, Plain Java Objects), είναι λογισμικό ανοικτού κώδικα υπό την άδεια χρήσης GPL και δεν εξαρτάται από άλλες βιβλιοθήκες συναρτήσεων. Οι μέθοδοι επεξεργασίας ενός *.json* αρχείου είναι τρεις:

- **Λειτουργία σε συνθήκες ροής (streaming)**, στην οποία αναλύει και δημιουργεί *.json* αρχεία αυξητικά (incremental processing) σε ροές συμβόλων (token streams). Πλεονέκτημά της είναι ότι χρησιμοποιείται σε χρονικά κρίσιμες λειτουργίες και ότι είναι εξαιρετικά γρήγορη.
- **Λειτουργία μοντελοποίησης σε Δέντρο (tree modeling)**, στην οποία χειρίζεται τα δεδομένα ως κόμβους ενός αποθηκευμένου στη μνήμη δένδρου βασισμένος, εν πολλοίς, στη λειτουργικότητα του XML DOM (Document Object Model). Πρόκειται για την πιο εύχρηστη λειτουργία.
- **Λειτουργία αντιστοίχισης δεδομένων (Data Binding)**, κατά την οποία μετατρέπει JSON-μορφοποιημένα δεδομένα σε Αντικείμενα Java (Plain Old Java Objects- POJOs) και αντίστροφα. Η τελευταία λειτουργία είναι η πιο ευέλικτη και χρησιμοποιήθηκε για το πρώτο στάδιο της εξαγωγής των δεδομένων του κάθε ticket από το περιέχον ticket.json.

3.2.4.2 Ο Συντακτικός Αναλυτής org.json

Ο Συντακτικός Αναλυτής **org.json** αποτελεί μία συλλογή βασικών συναρτήσεων διαχείρισης JSON αρχείων που δε διαθέτει το πλήθος λειτουργιών άλλων parsers, αλλά έχει αποτελέσει τη βάση δημιουργίας άλλων περισσότερο εκτεταμένων πακέτων, όπως το json-simple.

Το **πλεονέκτημά** του είναι η μειωμένη κατανάλωση υπολογιστικών πόρων (μνήμης) και η δυνατότητά του να διαβάζει απευθείας από συγκεκριμένα γνωστά πεδία ενός *.json* χωρίς να χρειάζεται η χαρτογράφηση ολόκληρου του *.json* αρχείου.

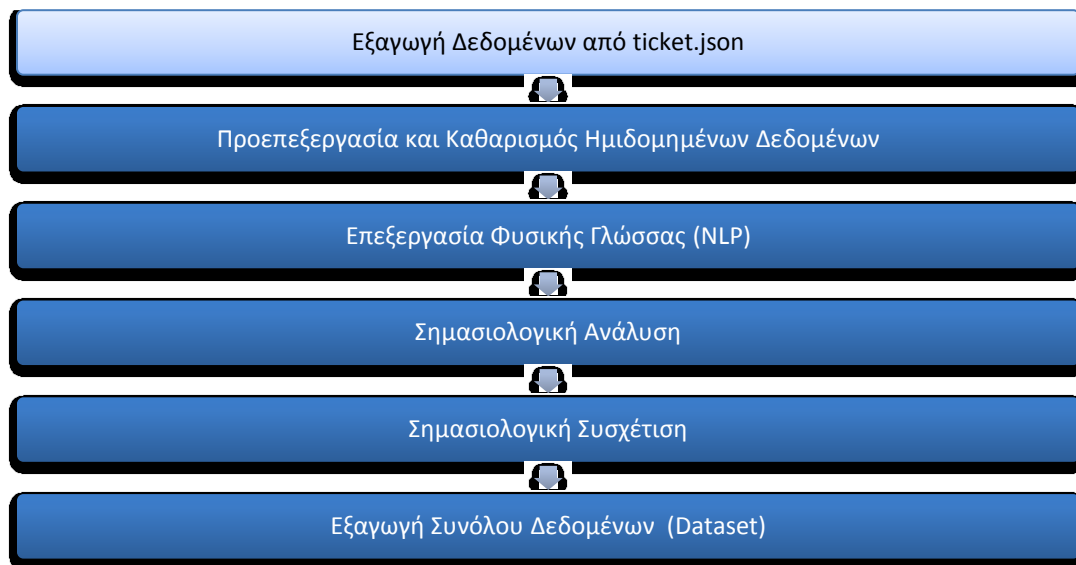
Ο org.json χρησιμοποιήθηκε για την ανάκτηση των πληροφοριών συγκεκριμένων πεδίων ιδιοτήτων του ticket.json, αντί του Jackson JSON Processor, ο οποίος χρησιμοποιήθηκε για την ανάκτηση ολόκληρων των .json tickets.

3.2.5 Υλοποίηση Χαρτογράφησης και Αποθήκευσης των Δεδομένων της RoR

Οι τεχνολογίες και τα εργαλεία που αναφέρθηκαν παραπάνω χρησιμοποιήθηκαν για την ανάγνωση των ticket.json αρχείων. Ο Jackson JSON Processor χρησιμοποιήθηκε για να χαρτογραφήσει κάθε ticket.json σε ένα Java Object. Για τον σκοπό αυτό δημιουργήσαμε μία κλάση αντιστοιχίας (binding class) που περιέχει μεταβλητές με αναγνωριστικά ίδια με τα πεδία του ticket (Πίνακας 3.2-1). Μέσω του

εργαλείου Jackson.mapper() η αντιστοίχιση πραγματοποιείται λαμβάνοντας υπόψη και τον τύπο των δεδομένων που μεταγράφονται (int, String, ArrayList κτλ).

Δημιουργήθηκε έτσι στη μνήμη ένα αντικείμενο για κάθε ticket. Στη συνέχεια, μπορούμε να έχουμε πρόσβαση στο ticket μέσω του αντικειμένου τύπου *ticket* και στις επιμέρους τιμές των μεταβλητών που αντιστοιχούν στα αρχικά πεδία του *ticket*.



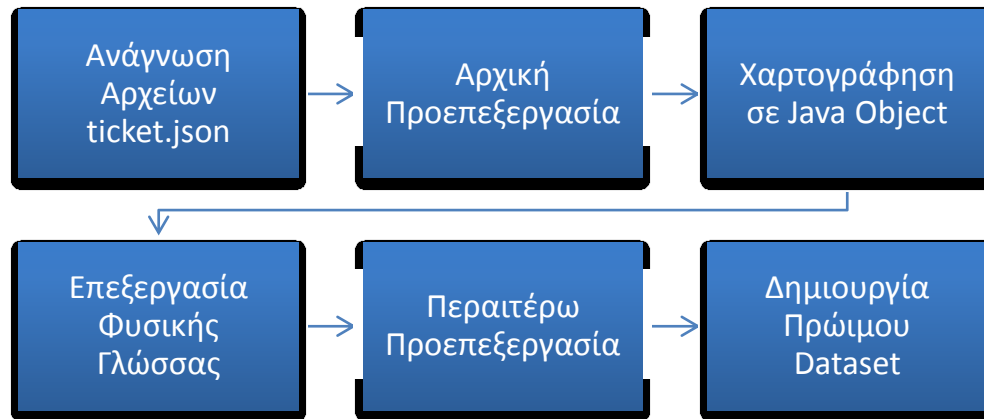
Σχήμα 3.2-5 Πρώτο Στάδιο: Εξαγωγή Δεδομένων από ticket.json

Η αναδρομική εφαρμογή αυτής της διαδικασίας στον κατάλογο με όλα τα *ticket.json* (Εικόνα 3.2-2) που αντλήθηκαν από το Lighthouse ΣΔΣ, σε συνδυασμό με την προεπεξεργασία των δεδομένων και την επεξεργασία φυσικής γλώσσας που περιγράφονται στις επόμενες παραγράφους (3.3 και 3.4, αντίστοιχα) έχουν ως αποτέλεσμα τη δημιουργία ενός πρώιμου συνόλου δεδομένων (*dataset*) το οποίο θα αποτελέσει τη βάση για την παραγωγή του τελικού συνόλου δεδομένων, που είναι και ο σκοπός της διπλωματικής εργασίας.

Το πρώιμο αυτό υποσύνολο αποθηκεύεται σε ένα αρχείο *.arff*, το οποίο είναι άμεσα επεξεργάσιμο από τη σουίτα εξόρυξης δεδομένων Weka (Waikato Environment for Knowledge Analysis – (Weka 3: Data Mining Software in Java) για μια πρώτη μελέτη των δεδομένων.

ArrayList <attachments> attachments	String creator_name	String permalink	int user_id;
int assigned_user_id	String latest_body	int priority	String user_name;
String assigned_user_name	String milestone_due_on	int project_id	String url;
String body	String milestone_title	String raw_data	int attachments_count;
String body_html	int milestone_id	String state	ArrayList <Versions> versions
boolean closed	int number	String tag	int user_id;
String created_at	String original_body	String title	String user_name;
int creator_id	String original_body_html	String updated_at	String url;

Πίνακας 3.2-1 Λίστα μεταβλητών της κλάσης αντιστοιχίας *ticket.class*



Σχήμα 3.2-6 Στάδια Δημιουργίας Πρώιμου Dataset

3.3 Προεπεξεργασία Δεδομένων

Τα δεδομένα που εξάγονται από ένα αρχείο ticket.json μέσω του συντακτικού αναλυτή JSON, υπόκεινται σε ένα πρώτο στάδιο επεξεργασίας. Η *Προεπεξεργασία Δεδομένων* (Data Preprocessing), πρωταρχικό στάδιο της Εξόρυξης Δεδομένων, είναι απαραίτητη για την απόρριψη άχρηστης, ακατανόητης, επουσιώδους και επαναλαμβανόμενης πληροφορίας.



Σχήμα 3.3-1 Δεύτερο Στάδιο: Προεπεξεργασία και Καθαρισμός Μερικώς δομημένων Δεδομένων

Στην περίπτωση των tickets περιττή πληροφορία μπορεί να βρίσκεται:

- Σε οποιοδήποτε πεδίο (ιδιότητα) και να είναι ακατανόητη ή ασύμβατη με το αναμενόμενο είδος πληροφορίας (πχ αριθμητικά δεδομένα σε πεδίο που επιτρέπει μόνο αλφαριθμητικά (strings)).
- Στο πεδίο **<original-body-html>** που, όπως έχει αναφερθεί, είναι ιδιαίτερου ενδιαφέροντος. Για το πεδίο αυτό αναμένονται, εκτός από φυσικό κείμενο, και προτάσεις συγκεκριμένες ακολουθίες αναγνωριστικών (tags), όπως αυτά ορίζονται από το πρότυπο σήμανσης που ακολουθείται (δηλ. HTML).

3.3.1 Καθαρισμός ασύμβατης πληροφορίας

Για την πρώτη περίπτωση, ο καθαρισμός γίνεται με απλή αντικατάσταση ειδικών χαρακτήρων που δεν αναγνωρίζονται σε επόμενα στάδια της διαδικασίας από την κενή συμβολοσειρά (" ") μέσω κανονικών εκφράσεων (regular expressions). Ο Καθαρισμός αυτός συμβαίνει αμέσως πριν την χαρτογράφηση από το ticket.json στο Java Object μέσω του Jackson API.

Τέτοιοι χαρακτήρες είναι:

Χαρακτήρες	Αντικατάσταση με
Ακολουθίες διαφυγής παραγράφου \r\n, \r, \n, \n\r	Χαρακτήρα κενού " "
Ενδιάμεσα κόμματα ",",	Χαρακτήρα κενού " "
"@", "\$"	Χαρακτήρα κενού " "
Backslash "\" (εντοπίζεται με την κανονική έκφραση "\\")	Χαρακτήρα κενού " "
"&", "<", ">"	&, <, >
"""	Χαρακτήρα κενού " "
Ψηφία "&#\d+;"	

3.3.2 Διαχείριση Μερικώς Δομημένων Δεδομένων Ιστού

Τα Μερικώς Δομημένα Δεδομένα είναι μια διαμόρφωση δεδομένων που δεν συμμορφώνονται με την αυστηρή δομή των πινάκων και των μοντέλων δεδομένων που απαντώνται σε Βάσεις Δεδομένων, αλλά εντούτοις περιέχουν αναγνωριστικά (tags) ή άλλες σημάνσεις για να διαχωρίσουν σηματολογικά στοιχεία και ιεραρχίες εγγραφών και πεδίων εντός των δεδομένων. Ως τέτοιες διαμορφώσεις χαρακτηρίζονται οι XML, HTML και γενικά οι γλώσσες σήμανσης (markup languages).

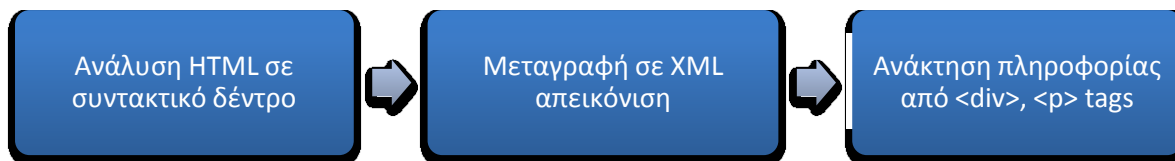
Στα πλαίσια της παρούσας διπλωματικής εργασίας έχει ήδη δοθεί ιδιαίτερη έμφαση στο πεδίο **<original-body-html>** που υπάρχει στα .json tickets. Αξίζει να σημειωθεί ότι η πληροφορία που φέρει το πεδίο **<original-body-html>** δε διαφέρει από αυτή του **<original-body>**. Η διαφορά εντοπίζεται στο ότι στην πρώτη περίπτωση έχουμε ένα σώμα κειμένου διαμορφωμένο σε HTML, ενώ στη δεύτερη το ίδιο περιεχόμενο βρίσκεται σε μορφή απλού κειμένου (plain text).

Αν και αυτή η διαφοροποίηση φαινομενικά περιπλέκει την ανάγνωση του κειμένου από τον χρήστη - άνθρωπο, στην πραγματικότητα διευκολύνει την ανάγνωση εκ μέρους της μηχανής (machine). Αυτό συμβαίνει διότι κάθε παράγραφος είναι σαφώς ορισμένη εντός ενός div tag (<div></div>) και κάθε πρόταση μέσα σε αυτή εντός του p tag (<p></p>). Επομένως, η αναγνώριση του πότε αρχίζει και πότε τελειώνει μία πρόταση δεν εναπόκειται πλέον στην αναζήτηση των «ανθρώπινων» σημείων στίξης (τελεία ή άνω τελεία, αλλά των «μηχανικών» (<div> και <p> tags).

3.3.2.1 Η Βιβλιοθήκη HtmlCleaner και η γλώσσα ερωτημάτων XPath

Η βιβλιοθήκη HtmlCleaner (<http://htmlcleaner.sourceforge.net/>) ενσωματώνει τις λειτουργίες ενός συντακτικού αναλυτή HTML που επεξεργάζεται HTML κείμενο μέσω συντακτικών δένδρων και μεταγράφει την πληροφορία σε XML διαμόρφωση σύμφωνα με τα standards του W3C. Στη συνέχεια, η πληροφορία είναι προσβάσιμη μέσω ενός υποσυνόλου της XML γλώσσας ερωτημάτων XPath. Η XPath

αναλύει το XML αρχείο που δημιουργείται και εξάγει τις πληροφορίες από συγκεκριμένο XML tag (στην περίπτωση του **<original-body-html>** από τα **<div>** και **<p>**).



Σχήμα 3.3-2 Λειτουργίες της HtmlCleaner

Στα πλαίσια αυτής της εργασίας η εξαγωγή των προτάσεων από το **<original-body-html>** γίνεται σε δύο στάδια (βλ. Σχήμα 3.3-2)

Αρχικά, καλείται η συνάρτηση *clean()* της βιβλιοθήκης HtmlCleaner η οποία δημιουργεί ένα συντακτικό δέντρο του HTML κειμένου, προσπελάσιμο από τη ρίζα (root node) του. Στη συνέχεια, καλείται στη ρίζα η συνάρτηση εκτίμησης *evaluateXPath()* με όρισμα **"//div/p"** (κανονική έκφραση σύμφωνα με το XPath Πρότυπο), αναζητώντας δηλαδή νέες προτάσεις εντός του τμήματος κειμένου. Κάθε πρόταση που επιστρέφεται τοποθετείται σε νέα γραμμή μίας νέας συμβολοσειράς.

Αν συγκρίνουμε, για παράδειγμα, τα εν λόγω πεδία στο υπ' αριθμόν #3453 έχουμε :

"original_body":

```
"Trying to use PostgreSQL as DB backend for rail v2.3.4 (gentoo).\r\n\r\nrake
db:create:sessions uses rails string type for session_id field. Unfortunately
postgresql_adapter translates this type to postgres type: character varying with limit 255
(set explicitly in the adapter code) which is too short for session_id generated by
rails.\r\n\r\nI simply changed the limit value for the string type to 512 and it seems
working, however I don't know how long this field should be to handle every session_id
generated by rails.\r\n\r\nThe adapter was installed on my system together with activerecord-
2.3.4"
```

"original_body_html":

```
"<div><p>Trying to use PostgreSQL as DB backend for rail v2.3.4\n(gentoo).</p>\n<p>rake
db:create:sessions uses rails string type for session_id\nfield. Unfortunately
postgresql_adapter translates this type to\npostgres type: character varying with limit 255
(set explicitly in\nthe adapter code) which is too short for session_id generated
by\nrails.</p>\n<p>I simply changed the limit value for the string type to 512 and\nit seems
working, however I don't know how long this field should\nbe to handle every session_id
generated by rails.</p>\n<p>The adapter was installed on my system together
with\nactiverecord-2.3.4</p></div>"
```

Πίνακας 3.3-1 - Σύγκριση ανάμεσα σε **<original-body>** και **<original-body-html>**

Παρατηρούμε πως στη δεύτερη περίπτωση απουσιάζουν χαρακτήρες νέας γραμμής (**\r\n\r** κτλ) ενώ αντίθετα υπάρχουν **div** και **p** HTML tags.

Αν τώρα εφαρμοστεί η τεχνική που αναφέρθηκε το αποτέλεσμα θα είναι δύο «προτάσεις»

```
Trying to use PostgreSQL as DB backend for rail v2.3.4\n(gentoo).
rake db:create:sessions uses rails string type for session_id\nfield. Unfortunately
postgresql_adapter translates this type to\npostgres type: character varying with limit 255
(set explicitly in\nthe adapter code) which is too short for session_id generated by\nrails.
I simply changed the limit value for the string type to 512 and\nit seems working, however I
don't know how long this field should\nbe to handle every session_id generated by rails.
```


The adapter was installed on my system together with \nactive-record-2.3.4

Πίνακας 3.3-2 Διαχωρισμός προτάσεων με markup parsing

Και μετά την αφαίρεση των ανεπιθύμητων χαρακτήρων κατά τα προηγούμενα:

Trying to use PostgreSQL as DB backend for rail v2.3.4 (gentoo).

rake db:create:sessions uses rails string type for session_id field. Unfortunately postgresql_adapter translates this type to postgres type: character varying with limit 255 (set explicitly in the adapter code) which is too short for session_id generated by rails.

I simply changed the limit value for the string type to 512 and it seems working, however I don't know how long this field should be to handle every session_id generated by rails.

The adapter was installed on my system together with activerecord-2.3.4

Πίνακας 3.3-3 Διαχωρισμός προτάσεων με markup parsing μετά την αφαίρεση ειδικών χαρακτήρων

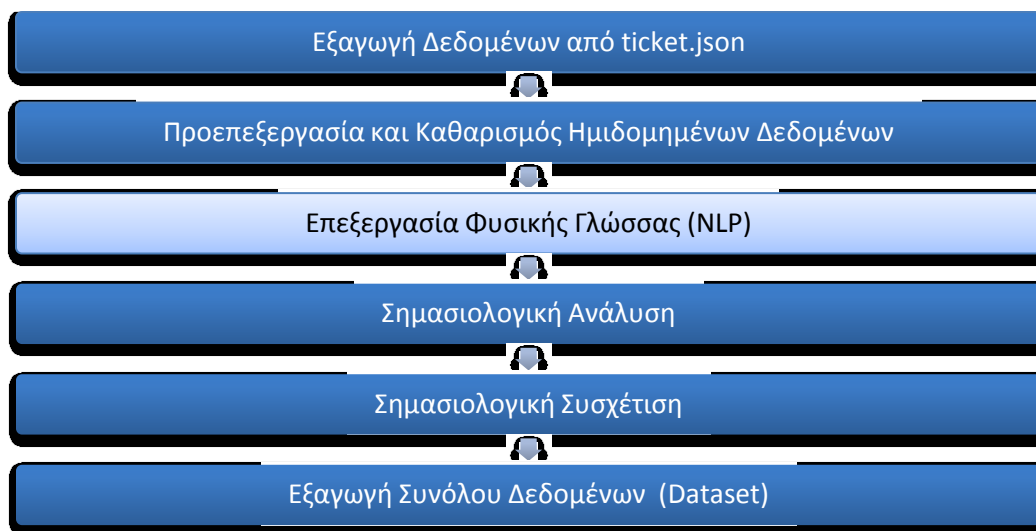
που είναι μία απολύτως κατανοητή αλληλουχία προτάσεων, ακόμη και για τον ανθρώπινο παρατηρητή.

Διαπιστώνουμε λοιπόν συγκρίνοντας τα αποτελέσματα της τεχνικής αυτής με το απλό κείμενο στον Πίνακα 3.3-1 (α) ότι με ελάχιστο υπολογιστικό κόστος μπορούμε να έχουμε καλύτερα αποτελέσματα.

3.4 Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing)

3.4.1 Γενικές Αρχές

Η Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing - NLP) είναι το κοινό αντικείμενο της Επιστήμης Υπολογιστών και της Γλωσσολογίας που ασχολείται με την αλληλεπίδραση μηχανής και φυσικών ανθρώπινων γλωσσών (natural languages).



Σχήμα 3.4-1 Τρίτο Στάδιο: Εξαγωγή Φυσικής Γλώσσας (Natural Language Processing)

Καθώς οι μοντέρνοι αλγόριθμοι της NLP στηρίζονται στη μηχανική εκμάθηση (machine learning) και ιδιαίτερα στη στατιστική μηχανική εκμάθηση, η NLP εντάσσεται στον τομέα της Τεχνητής Νοημοσύνης και οι τεχνικές της κατηγοριοποιούνται στην Υπολογιστική Γλωσσολογία. Το θεωρητικό και μαθηματικό

υπόβαθρό της εδράζεται πέρα στη στατιστική (δίνοντας έμφαση στη Bayesian στατιστική), στη Γραμμική Άλγεβρα και τη Θεωρία Βελτιστοποίησης.

Ως *Φυσική Γλώσσα* ορίζεται οποιαδήποτε γλώσσα προκύπτει με αβίαστο τρόπο ως άμεσο αποτέλεσμα της έμφυτης ικανότητας του ανθρώπινου νου. Διαφέρει από τις Δομημένες (structured) και τις Κανονικές (formal) γλώσσες (γλώσσες προγραμματισμού, μαθηματικές γλώσσες/γλώσσες λογικής), καθώς οι γραμματικοί κανόνες που διέπουν μία Φυσική Γλώσσα μελετώνται εκ των υστέρων, ενώ στην περίπτωση των προαναφερθέντων γλωσσών κατασκευάζονται ώστε να τις σχηματίσουν.

Μία Φυσική Γλώσσα (όπως και γενικά κάθε Γλώσσα) χαρακτηρίζεται από μία Γραμματική. Γραμματική ονομάζεται το σύνολο των Γραμματικών Κανόνων (Grammar Rules), δηλαδή δομικών κανόνων που κυβερνούν τη σύνθεση προτάσεων, φράσεων και λέξεων. Η Επεξεργασία Φυσικής Γλώσσας καλείται να ανακαλύψει αυτούς τους κανόνες και να αναλύσει με αυτούς φυσικό κείμενο (plain text).

Η γενική αποστολή της NLP, δηλ. η μελέτη κειμένου σε φυσική γλώσσα, μπορεί να αναλυθεί σε πολλά επιμέρους προβλήματα που απασχολούν τους ερευνητές. Τέτοια είναι η αυτόματη δημιουργία περίληψης, η αναζήτηση αναφορών ανάμεσα σε μέλη του κειμένου, η ανάλυση ομιλιών και συνομιλιών, η μετάφραση από μία γλώσσα σε άλλη, η γραμματική και συντακτική ανάλυση, η αναγνώριση ομιλίας, η αναγνώριση θεματολογίας κ.ά.

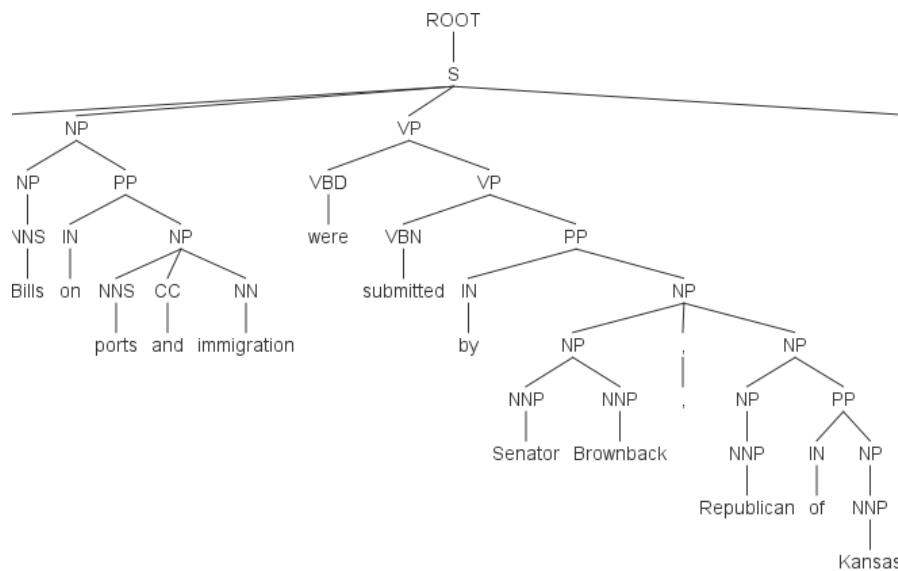
Σε αδρές γραμμές, η βασική λογική της NLP απαιτεί την ανάλυση με τεχνικές Μηχανικής Εκμάθησης για την εξαγωγή κανόνων (rules extraction). Ενώ αρχικά η έρευνα στράφηκε προς τη δημιουργία αυστηρών κανόνων (τύπου if-then) μέσω δένδρων αποφάσεων (decision trees) που προσομοίαζαν σε κανόνες ορισμένους από τον άνθρωπο, στη συνέχεια το ενδιαφέρον μετατέθηκε στα στατιστικά μοντέλα. Τα τελευταία παράγουν πιο ευέλικτους κανόνες που δίνουν καλύτερα αποτελέσματα σε άγνωστο και ακαθάριστο κείμενο.

Η εφαρμογή των αλγόριθμων συνήθως γίνεται σε συλλογές πραγματικών εγγράφων (corpora)². Τα κείμενα αυτά θεωρούνται *δεδομένα εκμάθησης* (training data) για τα οποία έχουν εξαχθεί εκ των προτέρων χειροκίνητα οι σωστές τιμές. Οι αλγόριθμοι εκπαιδεύουν τα βάρη τους (weight training) ώστε να προσεγγίσουν αυτές ακριβώς τις τιμές. Ο στόχος είναι όχι μόνο να εκπαιδευτούν έτσι ώστε να δίνουν ικανοποιητικά αποτελέσματα, αλλά να το κάνουν αποφεύγοντας να εξειδικεύσουν στα δεδομένα εκμάθησης. Έτσι, περιορίζεται το φαινόμενο της υπερ-εκπαίδευσης (overfitting), λόγω της υπερβολικής εξάρτησης από τα δεδομένα εκπαίδευσης. Το μοντέλο που εξάγεται, εφαρμόζονται πάνω σε διαφορετικό *κείμενο ελέγχου* (testing data), ώστε να εκτιμηθεί η ικανότητά του να εξάγει σωστούς κανόνες.

Για να παράξει ένα καλό μοντέλο πρόβλεψης, η NLP διαδικασία αναλύει κάθε Πρόταση (βασική μονάδα ανάλυσης) δημιουργώντας ένα **Συνεκτικό Συντακτικό Δένδρο (Parse Tree)** που αναπαριστά τη συντακτική δομή της. Σε ένα Συντακτικό Δένδρο οι εσωτερικοί κόμβοι αναπαριστούν μη-τερματικά

²Ένα τυπικό και δοκιμασμένο παράδειγμα corpus (πληθ. corpora), το οποίο και θα χρησιμοποιηθεί στην παρούσα εργασία, είναι η Τράπεζα Δένδρων Penn του Πανεπιστημίου της Pennsylvania. Περιέχει 500 έγγραφα του Brown Corpus και 2,500 πραγματικά άρθρα της εφημερίδας Wall Street Journal.

στοιχεία και τα φύλλα του δένδρου τερματικά. Για μία απλή πρόταση που αποτελείται από υποκείμενο – ρήμα – αντικείμενο συμβολίζουμε με S την πρόταση που τοποθετείται στη ρίζα του δένδρου, με NP (noun phrase) μία φράση ουσιαστικού και VP (verb phrase) μία ρηματική φράση. Για κάθε μέρος του λόγου υπάρχουν οι αντίστοιχοι συμβολισμοί που τοποθετούνται στο Συντακτικό Δένδρο, όπως φαίνεται στο Σχήμα 3.4-2 για την Πρόταση “Bills on ports and immigration were submitted by Senator Brownback , Republican of Kansas.”. Αυτή είναι η μία προσέγγιση της ανάλυσης μίας πρότασης που βασίζεται ουσιαστικά στη **χρήση Δομών Φράσεων (Phrase Structures)**.



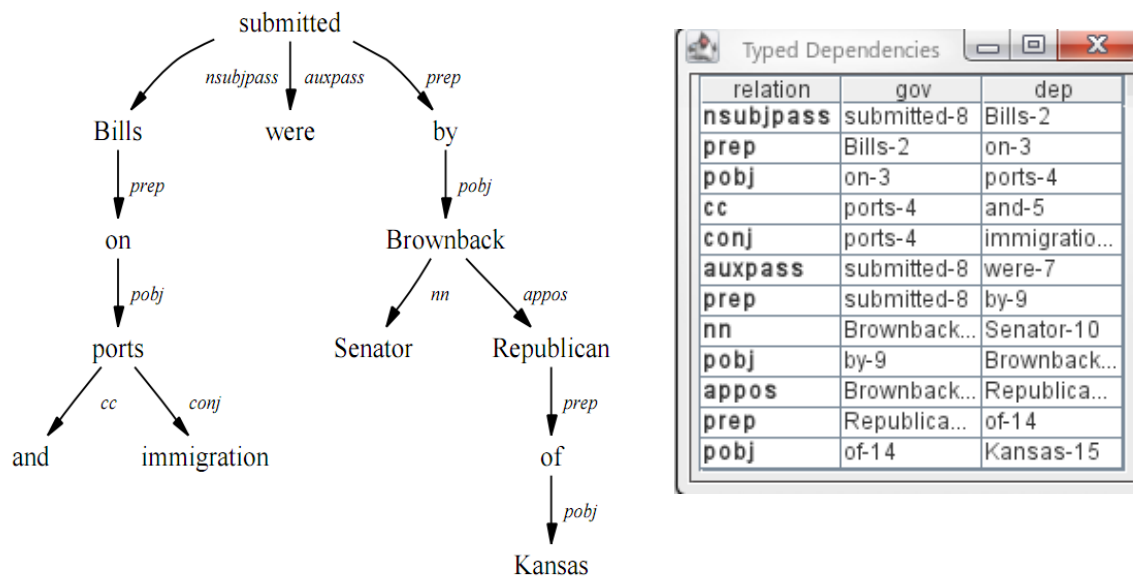
Σχήμα 3.4-2 Συντακτική Ανάλυση της πρότασης "Bills on ports and immigration were submitted by Senator Brownback , Republican of Kansas." μέσω δομών φράσεων

Ένας διαφορετικός τρόπος αναπαράστασης μίας πρότασης είναι μέσω της αναγνώρισης της *Σχέσης (Dependency)* ανάμεσα σε δύο λέξεις. Αν επιπλέον τοποθετηθεί και ένα αναγνωριστικό (tag) στον τύπο της Σχέσης, έχουμε την περίπτωση μίας **Γραμματικής Σχέσης (Typed Dependency)**. Το αναγνωριστικό αυτό δεν είναι άλλο από το είδος της Γραμματικής Σχέσης που διέπει τα μέλη της Σχέσης, πχ *object* για μία σχέση αντικειμένου ανάμεσα σε ρήμα και ουσιαστικό.

Τότε, το Συντακτικό Δένδρο διαμορφώνεται σε μία εντελώς διαφορετική ιεράρχηση. Στη ρίζα του δένδρου (root node) τοποθετείται γενικά το κυρίαρχο ρήμα της πρότασης και στη συνέχεια φύονται οι κλάδοι, που χαρακτηρίζονται από τον τύπο της Σχέσης και οδηγούν στις λέξεις – κόμβους του δένδρου. Στο Σχήμα 3.4-3 φαίνεται το Συντακτικό Δένδρο της ίδιας πρότασης (“Bills on ports and immigration were submitted by Senator Brownback , Republican of Kansas.”).

Μία σχέση Σχέσης είναι της μορφής:

όπου το πρώτο μέλος είναι το κυβερνόν (gov - governor) στη γραμματική σχέση, και το δεύτερο το εξαρτώμενο (dep - dependent). Έτσι, για παράδειγμα, ανάμεσα στις λέξεις *Bills* και *submitted* υπάρχει μία σχέση με αναγνωριστικό *nsubjpass* (passive nominal subject – παθητικό ονομαστικό υποκείμενο) η οποία συμβολίζεται -εκτός δένδρου- με το διατεταγμένο ζεύγος:



Σχήμα 3.4-3 Συντακτική Ανάλυση της πρότασης "Bills on ports and immigration were submitted by Senator Brownback , Republican of Kansas." μέσω Γραμματικών Εξαρτήσεων

Πίνακας 3.4-1 Πίνακας Γραμματικών Εξαρτήσεων της πρότασης

3.4.2 Η Στατιστική Επεξεργασία Φυσικής Γλώσσας (Statistical NLP)

Οι κανόνες παραγωγής μίας φυσικής γλώσσας δεν μπορούν να εντοπιστούν σαφώς από τους γλωσσολόγους, διότι για την πλήρη κατανόηση της γραμματικής της απαιτείται πλήρης επίγνωση του περιβάλλοντος στο οποίο εμφανίζεται. Έτσι, το πρόβλημα της Επεξεργασίας Φυσικής Γλώσσας κατατάσσεται στην κατηγορία του TN-Πλήρες (AI-Complete). Ένα TN-Πλήρες πρόβλημα ανάγεται στην επίλυση του κεντρικού προβλήματος της Τεχνητής Νοημοσύνης που απαιτεί τη δημιουργία μίας υπολογιστικής μηχανικής ίσης ή ανώτερης ικανότητας με τον ανθρώπινο νου (σημείο αυτοσυνείδησης). Με άλλα λόγια ένα πρόβλημα όπως η NLP ή η Υπολογιστική Όραση υπονοείται ότι δεν μπορούν να λυθούν με απλούς αλγόριθμους.

Η **Στατιστική Επεξεργασία Φυσικής Γλώσσας (Statistical NLP - sNLP)** λειτουργεί αντιστοιχίζοντας σε γραμματικούς κανόνες (grammar rules) πιθανότητες. Μπορούμε να ορίσουμε ως *Ερμηνεία* (parse) της πρότασης το άθροισμα των γραμματικών κανόνων για τους όρους που η πρόταση περιέχει. Συνήθως, η Ερμηνεία έχει τη μορφή Συντακτικού Δένδρου (Parse Tree). Συνεπώς, για τις διαφορετικές πιθανότητες κάθε γραμματικού κανόνα προκύπτουν και Συντακτικά Δένδρα με διαφορετική πιθανότητα αληθοφάνειας.

Στόχος της sNLP είναι να αναζητήσει (search) ποια Ερμηνεία της Πρότασης ανταποκρίνεται στην πραγματική γραμματική ανάλυσή της, αν αυτή γινόταν από τον άνθρωπο. Για να επιτευχθεί αυτός ο στόχος χρησιμοποιούνται οι μέθοδοι της Μηχανικής Εκμάθησης, ώστε βελτιώσουν υπολογιστικά την εξαντλητική αναζήτηση ανάμεσα στο σύνολο των πιθανών ερμηνειών της Πρότασης. Έτσι, για

παράδειγμα, περιορίζουν την έρευνα στις πιο πιθανοφανείς Ερμηνείες, βελτιώνουν την πιθανότητα της αναζήτησης (αλγόριθμος Baum-Welch) και απορρίπτουν Ερμηνείες που προσομοιάζουν (αλγόριθμος Viterbi).

Από τις μεθόδους που έχουν εφαρμοσθεί στην NLP, η Στατιστική Επεξεργασία Φυσικής Γλώσσας (Statistical NLP - sNLP) πλεονεκτεί σε σχέση με τη χρησιμοποίηση αλγορίθμων δένδρων απόφασης:

- Η sNLP δημιουργεί πιο ευπροσάρμοστα μοντέλα πιθανοτήτων που μπορούν να προσφέρουν την σχετική πιθανότητα διαφορετικών απαντήσεων αντί μόνο μίας των Δένδρων Αποφάσεων. Αυτού του τύπου η πληροφορία μπορεί να παρέχει πιο ενδιαφέροντα αποτελέσματα, ιδίως όταν το μοντέλο που παράγεται ενσωματώνεται σε ένα μεγαλύτερο σύστημα που μπορεί να αξιολογήσει τις σχετικές πιθανότητες.
- Σε περίπτωση που το κείμενο που θα υποστεί επεξεργασία περιέχει σφάλματα και προέρχεται από πραγματικά και όχι εργαστηριακά δοκιμασμένα δεδομένα. Σε αυτή την περίπτωση η sNLP αποδεικνύεται περισσότερο αξιόπιστη (Manning & Schütze, 1999).

Παραγωγή Γραμματικών Εξαρτήσεων με Στατιστική Επεξεργασία Φυσικής Γλώσσας

Για την παραγωγή Γραμματικών Εξαρτήσεων (Typed Dependencies) από μία πρόταση ακολουθούνται δύο στάδια: Αρχικά, εξάγεται η σχέση ανάμεσα σε δύο λέξεις και στη συνέχεια, αναγνωρίζεται ο τύπος της σχέσης.

1. Για το πρώτο βήμα, αναλύεται η πρόταση για την εύρεση Δομών Φράσεων που την απαρτίζουν. Στη συνέχεια σε κάθε φράση αναζητείται το επικεφαλής (head) συστατικό μέρος, δηλαδή η μία λέξη που επικρατεί. Η αναζήτηση δε γίνεται με συντακτικά κριτήρια, αλλά με σημασιολογικά κριτήρια: αναζητώντας ουσιαστικά τις πιο σημαντικές λέξεις της πρότασης για να ξεκινήσεις από αυτές η γραμματική ανάλυση.
2. Εν συνεχεία, για να αναγνωριστεί ο τύπος της σχέσης (πχ αντικείμενο, υποκείμενο, βοηθητική πρόταση, προσδιορισμοί) κάθε δυνατός συνδυασμός τύπων (48 στην περίπτωση του Stanford Parser) εφαρμόζεται στο συντακτικό δένδρο των δομών φράσεων που έχει παραχθεί στο πρώτο βήμα και επιλέγεται ο πιο πιθανοφανής.

3.4.3 Ο Επεξεργαστής Φυσικής Γλώσσας Stanford Parser

Ο Επεξεργαστής Φυσικής Γλώσσας Stanford (Stanford Parser) (Stanford Uni.) αποτελεί ένα πακέτο λογισμικού που, χρησιμοποιώντας τις τεχνικές που παρουσιάστηκαν προηγουμένως, κατορθώνει να αναλύει γραμματικά και συντακτικά προτάσεις. Έχοντας τη δυνατότητα να παράγει συντακτικά δένδρα Δομών Εκφράσεων (Phase Structures) και να αναγνωρίζει τις Γραμματικές Σχέσεις (Typed Dependencies - 48 στην περίπτωση του Stanford Parser) πάνω σε αυτά αποτελεί ένα ολοκληρωμένο λογισμικό Στατιστικής Επεξεργασίας Φυσικής Γλώσσας.

Για τη λειτουργία του ο Stanford Parser χρησιμοποιεί ένα Επεξεργαστή εκπαιδευμένο στην Τράπεζα Penn (βλ. υποσημείωση 2, παρ. **Error! Reference source not found.** σελ.**Error! Bookmark not defined.**). Δέχεται ως είσοδο ένα σώμα κειμένου το οποίο και αναλύει σε μία σειρά προτάσεων που χωρίζονται

μεταξύ τους με τον χαρακτήρα νέας γραμμής. Στη συνέχεια, διακρίνει τις λέξεις κάθε πρότασης εμπλέκοντας έναν διαχωριστή (tokenizer) και εφαρμόζει την ανάλυσή του αναζητώντας τις Γραμματικές Σχέσεις ανάμεσά τους.

Στην παρούσα εργασία εφαρμόζουμε τον Επεξεργαστή στο κείμενο που έχει προκύψει μετά τη συντακτική ανάλυση στο HTML-διαμορφωμένο περιεχόμενο της ιδιότητας `<original-body-html>` του κάθε ticket (**Error! Reference source not found.**).

3.4.4 Εφαρμογή του Stanford Parser στα δεδομένα της RoR

Σκοπός μας είναι να εξάγουμε σημαντικές λέξεις-κλειδιά (keywords) από το κείμενο που θα ενσωματωθούν στο τελικό Σύνολο Δεδομένων (Dataset) που θέλουμε να δημιουργήσουμε.



Σχήμα 3.4-4 Επαναληπτική Διαδικασία Συντακτικής Ανάλυσης και Εξαγωγής Διανύσματος Λέξεων

Επιλογή Γραμματικών Σχέσεων

Σε κάθε πρόταση του κειμένου αναζητούμε Γραμματικές Σχέσεις υποκειμένου ή αντικειμένου που συνδέουν, προφανώς, ουσιαστικά με ρήματα. Αυτά τα δύο είδη των σχέσεων είναι, συνήθως, τα πιο σημαντικά σε μία πρόταση και περιέχουν την απαραίτητη πληροφορία για την περιγραφή του ticket. Ο Επεξεργαστής επιστρέφει το είδος της σχέσης και τα μέλη της στη γνωστή μορφή:

Typed Dependency (Governor, Dependent)

όπου *Typed Dependency* παίρνει τιμές από τον Πίνακα Πίνακας 3.4-2 και φυσικά τα *Governor*, *Dependent* ενέχουν θέση ρήματος ή ουσιαστικού.

subj – subject	obj – object
<ul style="list-style-type: none"> • nsubj - nominal subject • nsubjpass - passive nominal subject • csubj - clausal subject 	<ul style="list-style-type: none"> • dobj - direct object • iobj - indirect object • pobj – object of proposition

Πίνακας 3.4-2 Υποκατηγορίες Γραμματικών Σχέσεων Υποκειμένου – Αντικειμένου

Stemming

Στη συνέχεια εφαρμόζουμε **Stemming** σε κάθε λέξη, δηλαδή αναζητούμε ετυμολογικά τη ρίζα της αφαιρώντας τυχόν καταλήξεις ή κλίσεις εφόσον πρόκειται για ρήματα. Η λειτουργία αυτή περιέχεται στη βιβλιοθήκη συναρτήσεων του Stanford Parser και βασίζεται σε ένα Μετατροπέα Πεπερασμένων Καταστάσεων (Finite-State Transducer) που εντοπίζει τις καταλήξεις (Minnen G., 2001). Ο Μετατροπέας, δέχεται ως είσοδο τη λέξη και το μέρος του λόγου που αποτελεί (πχ VBG για γερούνδιο) ανιχνεύει την κατάληξη και την αφαιρεί. Δυστυχώς, η τεχνική αυτή δεν μπορεί να εφαρμοστεί σε παράγωγες λέξεις ή συγκριτικούς βαθμούς επιθέτων.

Το Stemming είναι πολύ σημαντικό στην Προεπεξεργασία των Δεδομένων. Στην περίπτωση της παρούσας εργασίας αποτελεί το *δεύτερο βήμα Προεπεξεργασίας*, μετά την αποδιαμόρφωση του HTML περιεχομένου της `<original-body-html>` ιδιότητας.

Καταγραφή των λέξεων-κλειδών

Ακολουθως, τα μέλη των Σχέσεων που επιλέχθηκαν και υπέστησαν αναγωγή στην ετυμολογική βάση τοποθετούνται σε ένα **Διάνυσμα Λέξεων (word vector)**, το οποίο θα αποτελέσει μία νέα –παράγωγη– ιδιότητα του ticket και θα τοποθετηθεί στο Dataset.

Κάθε λέξη που τοποθετείται στο Διάνυσμα Λέξεων ελέγχεται για το εάν είναι μοναδική σε αυτό και δίνονται κάποια όρια για το μέγεθός της – δεν μπορεί να έχει μήκος μικρότερο των 3 ή μεγαλύτερο των 20 χαρακτήρων. Με αυτόν τον τρόπο, απορρίπτονται διπλές εγγραφές και λέξεις μικρές ή μεγάλες που δεν προσφέρουν σε πληροφορία, αλλά αντίθετα αυξάνουν το υπολογιστικό κόστος και προκαλούν στατιστικές αλλοιώσεις στο dataset.

Τα Διανύσματα Λέξεων όλων των tickets εγγράφονται σε εξωτερικό αρχείο (keywords.txt) για αυτόνομη πρόσβαση και για να χρησιμοποιηθούν στη συνέχεια.

Επίδειξη Χρήσης του Stanford Parser

Ας χρησιμοποιήσουμε ως παράδειγμα την τιμή της `<original-body-html>` ιδιότητας του ticket #1800. Υπενθυμίζουμε ότι μετά την πρώτη επεξεργασία αφαίρεσης των HTML Tags το κείμενο αυτό έχει ως εξής:

```
Trying to use PostgreSQL as DB backend for rail v2.3.4 (gentoo).
rake db:create:sessions uses rails string type for session_id field. Unfortunately
postgresql_adapter translates this type to postgres type: character varying with limit 255
(set explicitly in the adapter code) which is too short for session_id generated by rails.
I simply changed the limit value for the string type to 512 and it seems working, however I
don't know how long this field should be to handle every session_id generated by rails.
The adapter was installed on my system together with activerecord-2.3.4
```

Για την τελευταία πρόταση:

“The adapter was installed on my system together with activerecord-2.3.4”

ο Stanford Parser επιστρέφει τις σχέσεις:

```
determiner(adapter/NN, The/DT)
nominal passive subject(installed/VBN, adapter/NN)
passive auxiliary(installed/VBN, was/VBD)
possession modifier(system/NN, my/PRP$)
prep_collapsed(installed/VBN, system/NN)
adjectival modifier(2.3.4/CD, activerecord/JJ)
prep_collapsed(installed/VBN, 2.3.4/CD)
```

Επιλέγονται οι σχέσεις υποκειμένου/αντικειμένου και από αυτές ως λέξεις-κλειδιά οι [adapter installed], οι οποίες μετά το stemming γίνονται [adapter, install]. Ακολούθως, αυτές προστίθενται στο Διάνυσμα Λέξεων που αποθηκεύεται εξωτερικά. Το τελικό Διάνυσμα λέξεων για το συγκεκριμένο ticket γίνεται:

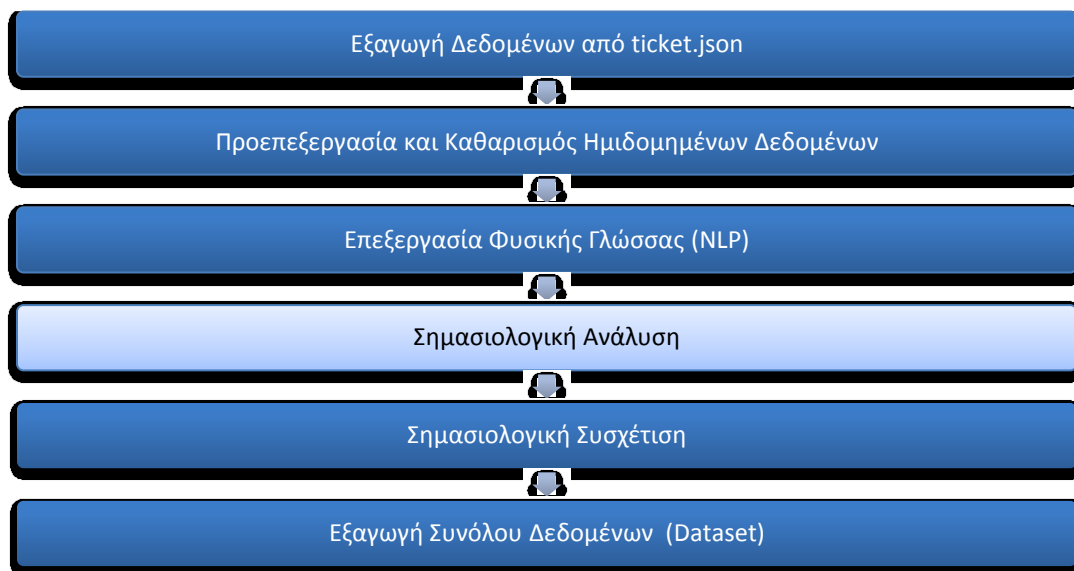
[postgresql use try session type translate character change
value seem don know field handle adapter install].

Κεφάλαιο 4: Μεθοδολογία II: Σημασιολογική Ανάλυση και Συσχέτιση των Δεδομένων

4.1 Σημασιολογική Ανάλυση

Στα προηγούμενα βήματα της μεθοδολογίας, έγινε εξαγωγή των δεδομένων και καθαρισμός τους, ενώ εντοπίστηκαν οι κρίσιμες λέξεις-κλειδιά στην περιγραφή κάθε ticket και τοποθετήθηκαν στο ανάλογο Διάνυσμα Λέξεων. Στόχος του επόμενου βήματος είναι η Σημασιολογική Ανάλυση των Διανυσμάτων Λέξεων ώστε να ομαδοποιηθούν κατάλληλα.

Στη συνέχεια ορίζεται το μοντέλο της ανάλυσης και οι τεχνικές που θα χρησιμοποιηθούν.



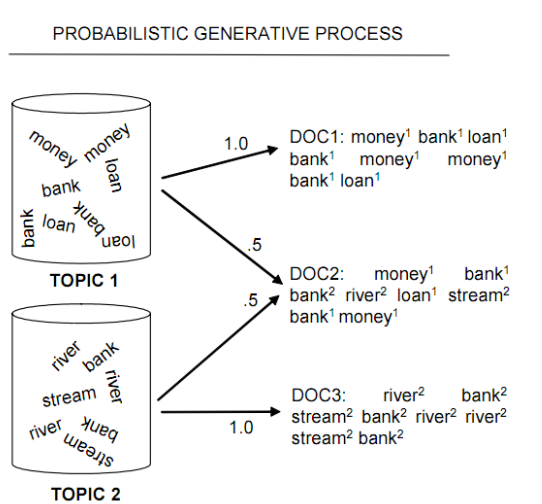
Σχήμα 4.1-1 Τέταρτο Στάδιο: Σημασιολογική Ανάλυση

4.1.1 Πιθανοτικό Μοντέλο Κύριου Θέματος (Probabilistic Topic Model)

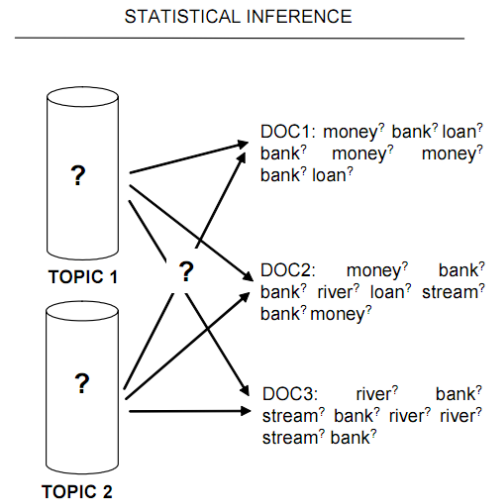
Το Πιθανοτικό Μοντέλο Κύριου Θέματος (ΠΜΚΘ –Topic Model) είναι ένα στατιστικό μοντέλο που χρησιμοποιείται στο χώρο της Επεξεργασίας Φυσικής Γλώσσας και γενικότερα της Ανάκτησης

Πληροφορίας (Information Retrieval - IR) για την ανακάλυψη Αφηρημένων Θεμάτων (Abstract Topics)³ σε συλλογές εγγράφων (documents).

Η ιδέα που κρύβεται πίσω από το ΠΜΚΘ είναι ότι κάθε έγγραφο προκύπτει από την ανάμιξη Θεμάτων, ενώ κάθε Θέμα είναι μία κατανομή πιθανότητας απλών λέξεων (words) (Steyvers). Εξαιτίας αυτής της δυνατότητας παραγωγής εγγράφων, το ΠΜΚΘ αποτελεί ένα Παραγωγικό Μοντέλο (Generative Model).



Σχήμα 4.1-2 Παραγωγή Εγγράφων από Θέματα



Σχήμα 4.1-3 Στατιστική Αναγωγή για την ανακάλυψη Θεμάτων από Έγγραφα

Ακόμη περισσότερο, η ανάλυση δεν προϋποθέτει οι λέξεις να είναι σε σειρά εμφάνισης στα έγγραφα, αλλά κάνει χρήση μόνο της συχνότητας εμφάνισής τους.⁴ Προφανώς, είναι δυνατόν να προκύψουν μικτά έγγραφα με συνδυασμό θεμάτων.

Η αντίστροφη διαδικασία είναι η εύρεση των λέξεων που απαρτίζουν κάθε Θέμα έχοντας ως δεδομένα τα έγγραφα. Αυτή η διαδικασία καλείται Στατιστική Αναγωγή (Statistical Inference), φαίνεται στο Σχήμα 4.1-3 και είναι αυτό που θα επιχειρηθεί στη συνέχεια πάνω στα δεδομένα του πειράματος.

Έχει προταθεί μία σειρά από ΠΜΚΘ για την ανάλυση εγγράφων και την αναζήτηση θεμάτων (Blei, Ng, & Jordan, 2003) (Steyvers & Griffiths) (Hofmann). Όλα αυτά τα μοντέλα αντιμετωπίζουν το έγγραφο ως ανάμειξη θεμάτων, αλλά διαφοροποιούνται ως προς τη στατιστική αντιμετώπιση του μοντέλου.

Στοιχεία Μαθηματικής Τεκμηρίωσης του Πιθανοτικού Μοντέλου Κύριου Θέματος

Θέτουμε την κατανομή των θεμάτων (topics) σε κάθε έγγραφο και την a priori κατανομή των λέξεων (words) για δοθέν θέμα. Αν η λέξη που ανήκει σε ένα έγγραφο, τότε η πιθανότητα το j-στό θέμα να έχει επιλεχθεί για τη i-στή λέξη και η πιθανότητα της λέξης στο j-στο θέμα.

³ Η έννοια της αφαίρεσης (abstraction) στο Θέμα εξηγείται από το ότι το Θέμα δεν αναπαρίσταται μονοσήμαντα, αλλά ως το σύνολο των πιο πιθανών λέξεων που το συνθέτουν.

⁴ Από εδώ προκύπτει και η ιδέα του «σακουλιού λέξεων» (bag-of-words).

Το Μοντέλο ορίζει την ακόλουθη κατανομή των λέξεων μέσα στο έγγραφο.

(4.6)

όπου T το πλήθος των Θεμάτων. Για να απλοποιηθούν οι συμβολισμοί ορίζουμε με την κατανομή των λέξεων για το j -οστό θέμα και την κατανομή των θεμάτων στο έγγραφο d . Επιπλέον, θεωρούμε ότι το πλήθος των εγγράφων είναι D και κάθε έγγραφο αποτελείται από λέξεις. Συνεπώς, είναι ο συνολικός αριθμός των λέξεων. Οι παράμετροι φ και θ δηλώνουν ποιες λέξεις είναι σημαντικές για κάθε θέμα και έγγραφο αντίστοιχα.

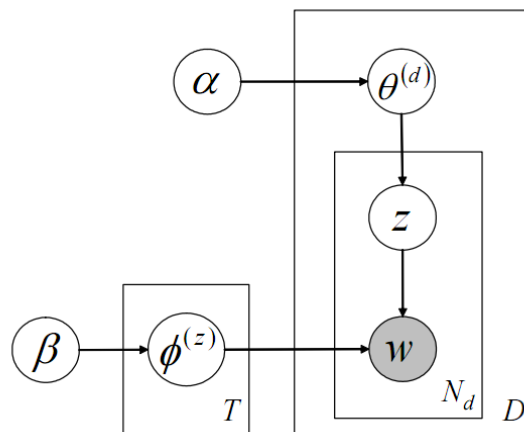
Ο Hoffmann (Hofmann) (Hofmann) εισήγαγε πρώτος τη χρήση του μοντέλου και τη μέθοδο της Πιθανοτικής Λανθάνουσας Σημασιολογικής Καταγραφής (Probabilistic Latent Semantic Indexing – pLSI). Το μοντέλο του δεν έκανε κάποια υπόθεση για την κατανομή πιθανότητας .

4.1.2 Η Λανθάνουσα κατά Dirichlet Κατάταξη (LDA)

Το κενό αυτό καλύφθηκε με την επέκταση του μοντέλου από τους (Blei, Ng, & Jordan, 2003) (Blei, Ng, & Jordan, 2003) (Blei, Ng, & Jordan, 2003) με την εισαγωγή της Dirichlet κατανομής ως εκ των προτέρων στη . Το νέο μοντέλο παραγωγής εγγράφων ονομάστηκε για αυτόν το λόγο Λανθάνουσα κατά Dirichlet Κατάταξη (Latent Dirichlet Allocation – LDA). Η επιλογή της Dirichlet κατανομής, εκτός από προφανής⁵, είναι και πολύ βολική καθώς απλοποιεί σημαντικά τη στατιστική αναγωγή.

Η συνάρτηση πυκνότητας πιθανότητας της κατανομής για διαστάσεις και την πολυωνυμική κατανομή είναι:

(4.7)



Σχήμα 4.1-4 Γραφική αναπαράσταση του LDA Topic Model

⁵ Η Κατανομή Dirichlet είναι η Εκ των Προτέρων Συζυγής της Πολυωνυμικής Κατανομής στην Bayesian Στατιστική.

Οι παράμετροι της Dirichlet κατανομής (4.7) είναι οι α_i . Συνήθως, οι παράμετροι λαμβάνονται ίσες με 1. Με όμοιο τρόπο είναι δυνατόν να προστεθεί και μία παράμετρος β η οποία προκύπτει από την εισαγωγή μία συμμετρικής Dirichlet κατανομής στην α .

Καλές επιλογές για τις παραμέτρους α και β θεωρούνται οι $\alpha = 1$ — και $\beta = 1$.

4.1.3 Η Δειγματοληψία Gibbs

Οι μεταβλητές ενδιαφέροντος της LDA είναι η κατανομή θέματος – λέξεων και η κατανομή θεμάτων – εγγράφων. Όμως, ο υπολογισμός αυτών των μεταβλητών είναι δύσκολος και υποφέρει από προβλήματα ευστάθειας. Για το λόγο αυτό, αντί να υπολογιστούν άμεσα οι θ και ϕ για κάθε έγγραφο, επιλέγεται να εκτιμηθεί η εκ των υστέρων πιθανότητα τοποθέτησης λέξεων σε θέματα, με εξουδετέρωση των θ και ϕ . Σε κάθε λέξη αντιστοιχίζεται ένας ακέραιος αριθμός z που σημειώνει το θέμα στο οποίο αυτή ανήκει.

Για τον παραπάνω υπολογισμό, χρησιμοποιείται ο αλγόριθμος δειγματοληψίας Gibbs που δειγματοληπτεί από μία πολυδιάστατη κατανομή χρησιμοποιώντας ένα μικρότερο υποσύνολο τιμών.

Αλγόριθμος Δειγματοληψίας Gibbs

Αναπαριστούμε το σύνολο των εγγράφων (corpus of documents) με τους δείκτες θέσης των λέξεων (word indices) w και θέσης των εγγράφων (document indices) d . Για κάθε λέξη w στο corpus υπολογίζεται η πιθανότητα να συνδεθεί η τρέχουσα λέξη με κάθε θέμα, υπό τη συνθήκη ανάθεσης στο ίδιο θέμα όλων των άλλων λέξεων. Από αυτή την υπό συνθήκη κατανομή πιθανότητας, δειγματοληπτείται ένα θέμα και αποθηκεύεται ως το θέμα στο οποίο ανατίθεται η λέξη.

Αυτή η υπό συνθήκη κατανομή με συνάρτηση πιθανότητας $P(w|z, \phi_z)$, όπου ϕ_z η ανάθεση στο θέμα της λέξης, ϕ_z όλων των λέξεων πλην της μίας που εξετάζεται, ϕ_z κατά τα γνωστά και αναφέρεται στις υπόλοιπες θέσεις ϕ_z και στις παραμέτρους ϕ_z και β .

Οι (Griffith04) απέδειξαν ότι, εξουδετερώνοντας τις θ μεταβλητές ότι:

$$(4.8)$$

Όπου ϕ_z και β είναι πίνακες των μετρήσεων, οι ϕ_z και β περιέχουν τον πλήθος των αναθέσεων της λέξης w στο j -οστό θέμα, μη συμπεριλαμβανομένης της εμφάνισης w , και το πλήθος των αναθέσεων του j -οστού θέματος σε μία λέξη του εγγράφου d .

Από την εξίσωση (4.8) μπορούμε να αντιληφθούμε πως λειτουργεί η δειγματοληψία Gibbs: Στο δεξί σκέλος της, ο αριστερός παράγοντας υπολογίζει την πιθανότητα μίας λέξης w να ανήκει στο θέμα z , ενώ ο δεξιός την πιθανότητα να ανήκει το θέμα z στο έγγραφο d . Καθώς πολλές εμφανίσεις της ίδιας λέξης στο corpus έχουν ανατεθεί στο θέμα z , αυξάνεται η πιθανότητα να γίνει ανάθεση της λέξης στο θέμα z . Και όταν ένα θέμα z έχει εμφανιστεί πολλές φορές σε ένα έγγραφο d , αυξάνεται η πιθανότητα κάθε λέξη που ανήκει στο έγγραφο αυτό να ανατεθεί στο θέμα z .

Συμπερασματικά, οι λέξεις συνδέονται με θέματα ανάλογα με το πόσο συχνά εμφανίζονται σε αυτό, όπως επίσης ανάλογα με το πόσο κυριαρχεί το θέμα στο έγγραφο.

Για την εκτίμηση των μεταβλητών ενδιαφέροντος και , η οποία γίνεται a posteriori, ισχύουν οι σχέσεις εκτίμησής τους:

(4.9)

4.1.4 Η βιβλιοθήκη JGibbLDA

Για την εφαρμογή της θεωρητικής ανάλυσης που προηγήθηκε χρησιμοποιήθηκε το πακέτο λογισμικού JGibbLDA (<http://jgibbllda.sourceforge.net/>) που αποτελεί μία υλοποίηση της LDA με δειγματοληψία Gibbs γραμμένη σε Java.

Χρησιμοποιώντας την ίδια ορολογία, το λογισμικό πραγματοποιεί Σημασιολογική Ανάλυση σε ένα αρχείο κειμένου που σε κάθε σειρά του έχει ένα έγγραφο (document) που αποτελείται από λέξεις (words) που διαχωρίζονται με κενό διάστημα. (Πίνακας 4.1-1)

Πίνακας 4.1-1 Παράδειγμα αρχείου εισόδου για τη βιβλιοθήκη JGibbLDA

```
10
problem glob param
activerecord run base set migration line fail place logger seem switch
action rail check version have ruby1 meet requirement seem command require gem don want policy
organisational package manage dependency rack
encounter bug fix realize bunch patch have test feature
filename have image cause throw create error alt attribute tag view\helpers\asset -lcb- option file
object convert field seconds take machine patch simplify code operation avoid result
need return error allow user fix method way name distinguish patch add option break set code continue work
company example -lcb-
work ignore believe have body -rrb- line pass findelement seem traverse dom element find
activemodel remove space lead trail filter add class call option attribute clean patch rebase
session use test patch move line validate try protection get message include behavior see file fix
```

Καλώντας τη συνάρτηση δημιουργίας του μοντέλου τοποθετούμε ως ορίσματα, πέρα από το αρχείο δεδομένων, τις παραμέτρους και , τον αριθμό των επαναλήψεων μέχρι να προσεγγιστεί, τη συχνότητα αποθήκευσης των μοντέλων και των αριθμό των πιο πιθανών λέξεων για κάθε θέμα που επιθυμούμε να διατηρήσουμε.

Παράμετρος	Συμβολισμός	Προκαθορισμένη Τιμή
Εξωτερικό αρχείο Δεδομένων Εκπαίδευσης	dfile <string>	-
Πλήθος επιθυμητών θεμάτων	ntopics <int>	-
Παράμετρος	alpha <double>	50/ntopics
Παράμετρος	beta <double>	0.1
Πλήθος Λέξεων ανά Θέμα	twords<int>	0 (20)
Πλήθος Επαναλήψεων Εκπαίδευσης	niters <int>	2000

Πίνακας 4.1-2 Ορίσματα παραμετροποίησης JGibbLDA

Ως έξοδος επιστρέφεται ένας αριθμός αρχείων που περιγράφουν το μοντέλο θέματος που εκπαιδεύτηκε, ενώ εκτυπώνονται στην οθόνη τερματικού πληροφορίες για τις επιλογές με τις οποίες εκτελείται (configuration).

Αρχείο	Περιγραφή
<model_name>.others	Καταγραφή των τιμών των παραμέτρων πχ <code>alpha</code> , αριθμός επαναλήψεων
<model_name>.phi	Περιέχει τις κατανομές λέξεων σε θέματα δηλ. την πιθανότητα
<model_name>.theta	Περιέχει τις κατανομές θεμάτων σε έγγραφα, δηλ.
<model_name>.tassign	Περιέχει την κατανομή των λέξεων σε θέματα, χωρισμένων ανά θέμα
<model_name>.twords	Περιέχει κάθε θέμα με τις λέξεις που περιέχει
wordmap.txt	Καταγράφει την αντιστοιχία λέξεων με δείκτες (words - word indices)

Πίνακας 4.1-3 Επιστρεφόμενα αρχεία μετά την εφαρμογή JGibbLDA

4.1.5 Εφαρμογή της Λανθάνουσας κατά Dirichlet Κατάταξης και Εκπαίδευση του Μοντέλου στα Δεδομένα tickets

Εφαρμόζουμε όσα προαναφέρθηκαν στη λίστα διανυσμάτων που αναφέρθηκε στην προηγούμενη παράγραφο (ΟΕπίδειξη Χρήσης του Stanford Parser), η οποία επαναλαμβάνεται για ευκολία στον Πίνακα 4.1-1) με τιμές παραμέτρων όπως αυτές επιστρέφονται στο τερματικό:

Performing Latent Dirichlet Allocation on data

```
Starting LDA for discovering 3 topics in /Data/keywords.txt
Gibbs LDA Parameters:
alpha:      16.0
beta:       0.1
Topics:      3
Iterations:  500
savestep:    500
Topic Words: 20
dfile:       keywords.txt
Estimate the LDA model from scratch
Sampling 500 iteration!
Iteration 1 ...
Iteration 2 ...
Iteration 3 ...
[...]
Iteration 499 ...
Gibbs sampling completed!
Saving the final model!
```

Πίνακας 4.1-4 Οθόνη Τερματικού μετά την Εφαρμογή LDA σε Πραγματικά Δεδομένα

Το αποτέλεσμα της παραγωγής θεμάτων αποθηκεύεται στο αρχείο **model-final.twords** :

Topic 0th:	Topic 1th:	Topic 2th:
have 0.07735	patch 0.09272	option 0.052542
seem 0.05849	line 0.05636	test 0.03559
add 0.03962	set 0.038181	error 0. 03559
glob 0.020754	fix 0. 038181	-lcb- 0. 03559
run 0.020754	file 0. 038181	code 0. 03559
base 0. 020754	problem 0.02	work 0. 03559
fail 0. 020754	param 0.02	activeresord 0.01864

switch 0. 020754 rail 0. 020754 require 0. 020754 package 0. 020754 fix 0. 020754 feature 0. 020754 filename 0. 020754 throw 0. 020754 attribute 0. 020754 object 0. 020754 take 0. 020754 machine 0. 020754 operation 0. 020754	migration 0.02 action 0.02 check 0.02 ruby1 0.02 meet 0.02 command 0.02 want 0.02 manage 0.02 rack 0.02 bunch 0.02 image 0.02 cause 0.02 create 0.02	place 0. 01864 logger 0. 01864 version 0. 01864 requirement 0. 01864 gem 0. 01864 don 0. 01864 policy 0. 01864 organisational 0. 01864 dependency 0. 01864 encounter 0. 01864 bug 0. 01864 realize 0. 01864 alt 0. 01864
--	--	--

Πίνακας 4.1-5 Επιστρεφόμενο αρχείο model-final.twords . Οι λέξεις σε **bold επιλέγονται ως οι πλέον αντιπροσωπευτικές του κάθε θέματος**

Παρατηρούμε ότι δημιουργούνται τρία θέματα που περιέχουν τις παραπάνω λέξεις (20 ανά θέμα) και τις αντίστοιχες πιθανότητες κάθε λέξη να υπάρχει στο θέμα. Από τις 20 λέξεις επιλέγονται οι πιο πιθανές είτε ως συγκεκριμένος αριθμός (πχ οι 5 πρώτες) είτε θέτοντας ένα κάτω όριο στην πιθανότητα (πχ 0.25). Στο παράδειγμά μας επιλέγουμε τις 5 πιο πιθανές λέξεις ανά θέμα και ορίζουμε αυτό το «σακούλι λέξεων» ως το πιο αντιπροσωπευτικό του θέματος. Άρα πλέον κάθε θέμα (topic) έχει ως εξής:

Εξάλλου, στο αρχείο **model-final.theta** περιέχονται οι πιθανότητες κάθε ticket (που ενέχει τη θέση του εγγράφου στην LDA) να ανήκει σε ένα θέμα.

Topic 0th:	Topic 1th:	Topic 2th:
0.3333333333333333	0.35294117647058826	0.3137254901960784
0.3559322033898305	0.3220338983050847	0.3220338983050847
0.3088235294117647	0.35294117647058826	0.3382352941176471
0.3157894736842105	0.3333333333333333	0.3508771929824561
0.3064516129032258	0.3225806451612903	0.3709677419354839
0.3666666666666666	0.3333333333333333	0.3
0.34782608695652173	0.3188405797101449	0.3333333333333333
0.3225806451612903	0.27419354838709675	0.4032258064516129
0.3709677419354839	0.3387096774193548	0.2903225806451613
0.265625	0.375	0.359375

Πίνακας 4.1-6 Επιστρεφόμενο αρχείο model-final.theta Κατάταξη tickets/εγγράφων ανά topic

Συγκρίνοντας μεταξύ τους τις πιθανότητες μπορούμε να κατατάξουμε ένα ticket σε ένα θέμα από τα τρία θέματα που προέκυψαν από την LDA.

Στο σημείο αυτό έχουν παραχθεί θέματα (topics) ως συλλογές λέξεων και κάθε ticket κατατάσσεται σε ένα θέμα. Παρατηρούμε, όμως, ότι κάθε θέμα που δημιουργήθηκε δεν έχει αναγνωριστικό (δηλ. όνομα) παρά είναι απλά μία συλλογή λέξεων. Αυτό φυσικά δε βοηθά στην αποκάλυψη της φυσικής σημασίας που έχει η κατάταξη των tickets σε θέματα. Στο επόμενο μέρος, θα κληθούμε να ανακαλύψουμε μία ονομασία για κάθε θέμα.

4.2 Σημασιολογική Συσχέτιση

Η Σημασιολογική Συσχέτιση (Semantic Relatedness) είναι η έννοια του πόσο σχετίζονται δύο ή περισσότερες έννοιες σε σημασιολογικό επίπεδο (δηλ. με βάση το νοηματικό τους περιεχόμενο). Για την ποσοτικοποίηση της έννοιας της Σημασιολογικής Συσχέτισης χρησιμοποιούνται διάφορες μετρικές οι οποίες τοποθετούν όμοιες σημασιολογικά λέξεις μαζί και «απομακρύνουν» τις ανόμοιες.

Στην περίπτωση της σημασιολογικής ανάλυσης κειμένου οι μετρικές αυτές μετρούν ουσιαστικά την συνύπαρξη (co-occurrence) λέξεων θεωρώντας ότι αυτή δε συμβαίνει τυχαία, αλλά παρατηρείται μόνο σε περιπτώσεις θεματικής σύμπτωσης.

Συνεπώς, μετρώντας τη σημασιολογική συσχέτιση (ή εγγύτητα) των θεμάτων (ως μη ταξινομημένο άθροισμα λέξεων) που έχουμε εξάγει μαζί με πιθανές περιγραφές τους, προσπαθούμε να τα αναγνωρίσουμε. Οι περιγραφές αυτές ουσιαστικά αποτελούν λέξεις που περιέχονται σε ένα λεξιλόγιο (vocabulary) που ορίζεται στη βάση του κάθε προβλήματος ή είναι εκ των προτέρων ορισμένο.



Σχήμα 4.2-1 Πέμπτο Στάδιο: Σημασιολογική Συσχέτιση (Semantic Relatedness)

4.2.1 Μετρικές Σημασιολογικής Συσχέτισης

Μία, μη εξαντλητική, λίστα των Μετρικών Σημασιολογικής Συσχέτισης (ΣΣ) είναι η εξής:

- Η Λανθάνουσα Σημασιολογική Ανάλυση (Latent semantic analysis - LSA) και οι βελτιώσεις της.
- Η Σημειακή Εκτίμηση Αμοιβαίας Πληροφορίας (Pointwise Mutual Information – PMI) και οι βελτιώσεις της.
- Η Κανονικοποιημένη Συμπιεσμένη Απόσταση (Normalized Compression Distance – NCD). Βελτίωση της NCD είναι η Κανονικοποιημένη Google Απόσταση (Normalized Google Distance – NGD) που θα αναλυθεί στη συνέχεια.

- Το WordNet, που είναι μία βάση δεδομένων λεξικογραφικού χαρακτήρα της Αγγλικής Γλώσσας, κατασκευασμένη μη-αυτοματοποιημένα.
- Ο Βαθμός Συγγένειας της Διαδικτυακής Εγκυκλοπαίδειας Wikipedia (n° of Wikipedia – noW), που κατασκευάζεται από το σύνολο των άρθρων της εγκυκλοπαίδειας ως γράφος. Η μετρική αυτή προκύπτει με αλγόριθμους επί του γράφου που υπολογίζουν το ελάχιστο μονοπάτι ανάμεσα στους όρους-κόμβους.

Στα πλαίσια της παρούσας διπλωματικής υιοθετούμε την Κανονικοποιημένη Google Απόσταση (Normalized Google Distance – NGD) για να μελετήσουμε τη λίστα θεμάτων που προέκυψαν από τη μελέτη της περιγραφής των tickets στις προηγούμενες παραγράφους.

4.2.2 Η Κανονικοποιημένη Google Απόσταση

Ας θεωρήσουμε δύο αλφαριθμητικά (strings) και για τα οποία αναζητούμε το ελάχιστο δυαδικό πρόγραμμα στο υπολογιστικό σύστημα αναφοράς που μπορεί να υπολογίσει το από το και αντίστροφα. Αυτό ορίζεται ως *απόσταση πληροφορίας* και συμβολίζεται ως . Αποδεικνύεται (Bennett, Li, Vitanyi, & Zurek, 1998) ότι η απόσταση πληροφορίας πληροί τα κριτήρια για να είναι μετρική. Από τη βασική αυτή μετρική προκύπτει η *κανονικοποιημένη απόσταση πληροφορίας* (*normalized information distance – NID*) που παίρνει τιμές ανάμεσα σε 0 και 1 ως εξής (Li, Chen, Li, Ma, & Vitanyi, 2004):

(4.10)

Με την εφαρμογή συνάρτησης συμπίεσης σε κάθε αλφαριθμητικό προκύπτει η συμπίεσμένη απόσταση πληροφορίας:

(4.11)

Η Google κατανομή (Google distribution)

Εισάγουμε ως *όρο αναζήτησης* (*search term*) στη μηχανή αναζήτησης Google το κάθε αλφαριθμητικό ή . Τότε το σύνολο των μονοσύνολων όρων είναι . Η έρευνα και για τα δύο αλφαριθμητικά στο Google δεν είναι διατεταγμένο ζεύγος, δηλ. οι όροι είναι δυνατόν να αντιστραφούν.

Το σύνολο των σελίδων που έχει δεικτοδοτήσει το Google είναι . Η τάξη μεγέθους του είναι και υπολογίζεται στα αποτελέσματα. Θεωρούμε ότι το ενδεχόμενο να επιστραφεί μία τυχαία ιστοσελίδα από τη μηχανή αναζήτησης είναι ισοπίθανο με κάθε άλλη και ίσο με .

Συμβολίζουμε με το σύνολο των σελίδων που επιστρέφονται για όρο αναζήτησης και το ονομάζουμε **γεγονός Google (Google event)**. Συνεπώς, η συχνότητα εμφάνισης — Όμοια για την παράθεση των όρων και (, επιστρέφονται τα αποτελέσματα .

Google Semantics

Το γεγονός Google , αποτελώντας το σύνολο όλων των σελίδων που περιέχουν μία ή περισσότερες εμφανίσεις του αλφαριθμητικού όρου αναζήτησης , επομένως ενσωματώνοντας, με κάθε πιθανή έννοια, όλα τα άμεσα συμφραζόμενα με τα οποία το εμφανίζεται στο Παγκόσμιο Ιστό. Αυτό συνιστά τα **Google semantics** του όρου (Cilbrasy, 2007).

Μαθηματική Φόρμουλα της Μετρικής NGD

Παρόμοια με τον μαθηματικό ορισμό (4.10), προκύπτει η Κανονικοποιημένη Google Απόσταση:

(12)

Όπου ο αριθμός των σελίδων που περιέχουν τον όρο και ο αριθμός αυτών που περιέχουν ταυτόχρονα του και .

Η επιλογή του M, καθώς η Google από το 2007 δεν ανακοινώνει τον αριθμό των σελίδων που βρίσκονται στο ευρετήριό της, είναι, όπως αναφέραμε της τάξης του , χωρίς αυτό να περιορίζει την επιλογή του αριθμού. Μοναδικό κάτω όριο είναι η ώστε το αποτέλεσμα της (12) να είναι πάντα θετικό.

4.2.3 Εφαρμογή της Μετρικής NGD στα δεδομένα του προβλήματος

Από το αποτέλεσμα της εφαρμογής της LDA στο προηγούμενο βήμα έχουμε ανακτήσει τα θέματα των tickets ως «σακούλια λέξεων» και στόχος μας είναι να τα συσχετίσουμε με γνωστές *κατηγορίες bugs* και με χαρακτηριστικά που αφορούν την *Ποιότητα Λογισμικού (Software Quality)*.

Προγραμματιστικά Σφάλματα και Χαρακτηριστικά Ποιότητας Λογισμικού

Υπάρχουν διάφορες συμβάσεις σχετικά με την κατηγοριοποίηση των προγραμματιστικών σφαλμάτων (programming bugs) στη βιβλιογραφία.

Μία συνοπτική λίστα με σφάλματα λογισμικού :

* Divide by zero	* Handle leak
* Infinite loops	* Stack overflow
* Arithmetic overflow	* Stack underflow
* Arithmetic underflow	* Buffer overflow
* Exceeding array bounds	* Deadlock
* Uninitialized variable	* Off by one error
* Access violation	* Race hazard
* Memory leak	* Loss of precision in type conversion

Πίνακας 4.2-1 Συνοπτική Λίστα Σφαλμάτων Λογισμικού

Σε ό,τι αφορά τη συσχέτιση με την Ποιότητα Λογισμικού επιλέχθηκε το Διεθνές Πρότυπο για την Εκτίμηση της Ποιότητας Λογισμικού **ISO/IEC 9126** (ISO/IEC, 2001) το οποίο ορίζει τα εξής χαρακτηριστικά:

Functionality - Λειτουργικότητα. <ul style="list-style-type: none"> • Suitability • Accuracy • Interoperability • Security • Functionality Compliance 	Efficiency - Αποδοτικότητα <ul style="list-style-type: none"> • Time Behaviour • Resource Utilisation • Efficiency Compliance
Reliability - Αξιοπιστία <ul style="list-style-type: none"> • Maturity • Fault Tolerance • Recoverability • Reliability Compliance 	Maintainability - Συντηρησιμότητα <ul style="list-style-type: none"> • Analyzability • Changeability • Stability • Testability • Maintainability Compliance
Usability - Χρηστικότητα. <ul style="list-style-type: none"> • Understandability • Learnability • Operability • Attractiveness • Usability Compliance 	Portability – Φορητότητα <ul style="list-style-type: none"> • Adaptability • Installability • Co-Existence • Replaceability • Portability Compliance

Πίνακας 4.2-2 Λίστα Χαρακτηριστικών Ποιότητας Λογισμικού κατά το Πρότυπο ISO/IEC 9126-1:2001

Η Σημασιολογική Συσχέτιση των θεμάτων με τις παραπάνω λίστες γίνεται, όπως προαναγγέλθηκε, με τον υπολογισμό της τιμής της μετρικής NGD για αλφαριθμητικά και ή .

Πίνακας 4.2-3 Τα θέματα που έχουν προκύψει από την περιγραφή του ticket

Η συνάρτηση που αναπτύχθηκε έχει ως εξής:

- Σχηματίζονται διαδοχικά οι όροι αναζήτησης , και η παράθεσή τους. Στη συνέχεια, κάθε όρος τοποθετείται ως όρισμα στην αναζήτηση Google κάνοντας χρήση του Google Ajax API (Google Inc.) (Google Inc.) σε διαμόρφωση (wrapping) για τη γλώσσα Java.
- Το αποτέλεσμα που επιστρέφεται από το διακομιστή της μηχανής αναζήτησης είναι σε μορφή JSON. Το .json αρχείο περιέχει τα πρώτα 8 αποτελέσματα της αναζήτησης μαζί με άλλα στατιστικά στοιχεία. Το πεδίο του .json που ενδιαφέρει είναι το **estimatedResultCount** που επιστρέφει το πλήθος των σελίδων που περιέχουν τον όρο αναζήτησης. Η ανάκτηση γίνεται χρησιμοποιώντας το Συντακτικό Αναλυτή JSON **org.json (3.2.4.2)**.
- Στη συνέχεια, υπολογίζεται το μέτρο της NGD μέσω του τύπου (12), και τέλος,
- Συγκρίνονται για κάθε τα μέτρα NGD και επιλέγεται να αντιστοιχιστεί το πιο σχετικό στο κάθε .
- Το σύνολο των αντιστοιχίσεων καταγράφεται εξωτερικά στο αρχείο `bugsperticket.txt`

Η διαδικασία επαναλαμβάνεται για τα χαρακτηριστικά Ποιότητας Λογισμικού (Πίνακας 4.2-2) με τις αντιστοιχίσεις να εγγράφονται στο αρχείο `SQMAssigned.txt`

Στο τέλος, όλες οι αντιστοιχήσεις ενσωματώνονται στο Dataset (`data2.arff`), ενώ στην οθόνη τερματικού εμφανίζονται συνοπτικές πληροφορίες για την αντιστοίχιση που προέκυψε μαζί με την τιμή της NGD που υπερτέρησε:

```
run:
```

Calculating Normalized Google Distance and Assigning Tags

```
have seem add glob run ruby
Top Result      Handle leak      0.019081373341026536
patch line set fix file ruby
Top Result      Loss of precision in type conversion      0.03403547253332332
option test error -lcb- code ruby
Top Result      Race hazard      0.1733250205196864
have seem add glob run ruby
Top Result      Functionality Compliance      0.052251059664542895
patch line set fix file ruby
Top Result      Recoverability      0.0982691177863115
option test error -lcb- code ruby
Top Result      Recoverability      0.1286234298962544
```

```
BUILD SUCCESSFUL (total time: 52 seconds)
```

Πίνακας 4.2-4 Αντιστοίχιση Θεμάτων tickets σε Bugs και SQ Metrics (Εμφάνιση Τερματικού)

Μετά από αυτό το βήμα έχει ουσιαστικά δομηθεί το Σύνολο των Δεδομένων που αφορούν τα tickets του Συστήματος Καταγραφής Σφαλμάτων του λογισμικού Ruby On Rails. Πέρα από τα βασικά attributes που είναι διαθέσιμα, έχουν εξαχθεί και τοποθετεί στο Dataset και πληροφορίες που περιέχονταν στην περιγραφή κάθε ticket του ΣΚΣ.

Το τελευταίο βήμα της διαδικασίας είναι να αποφασιστεί ποιες από τις ιδιότητες των tickets περιέχουν πραγματικά χρήσιμη πληροφορία, να αφαιρεθούν τα τετριμμένα και να μετατραπούν οι τύποι των δεδομένων σε αξιοποιήσιμη μορφή από το πρόγραμμα που θα κληθεί να εξορύξει πληροφορία από το Dataset, το Weka.

4.3 Προετοιμασία τελικού Συνόλου Δεδομένων

Το Σύνολο Δεδομένων που έχει προκύψει μετά την εφαρμογή των τεχνικών σηματολογικής ανάλυσης και συσχέτισης που επιλέξαμε διαθέτει μία πληθώρα χαρακτηριστικών, πρωτογενών και παραγώγων τα οποία πρέπει να επεξεργαστούμε πριν καταλήξουμε στο Τελικό Σύνολο Δεδομένων.

Συνεπώς, χρησιμοποιήσαμε συγκεκριμένα φίλτρα που υλοποιεί το Weka, τα φίλτρα μετατροπής τύπων ιδιοτήτων, τα φίλτρα αφαίρεσης και το φίλτρο ανακατάταξης ιδιοτήτων.

Τα φίλτρα μετατροπής ιδιοτήτων εφαρμόστηκαν για να μετατρέψουν:

- Αριθμητικές Ιδιότητες σε Ονομαστικές (Numeric to Nominal)
- Αλφαριθμητικές σε Ονομαστικές (String to Nominal)
- Αλφαριθμητικές σε Διανύσματα Λέξεων (String to WordVector)

ώστε να είναι εκμεταλλεύσιμες από τους αλγόριθμους κατάταξης, ενώ το φίλτρο αφαίρεσης εφαρμόστηκε για να αφαιρεθεί πλεονάζουσα πληροφορία. Οι ιδιότητες που αφαιρέθηκαν φαίνονται στον Πίνακα 4.3-1.

Πίνακας 4.3-1 Φίλτρα που εφαρμόστηκαν στις ιδιότητες του Συνόλου Δεδομένων

weka.filters.unsupervised.attribute.*	Attributes
Remove	number, project_id,raw_data,creator_name, version_id,milestone_due_on,priority,permalink,url, user_name
NumericToNominal	assigned_user_id, creator_id, user_id, milestone_id
StringToNominal	milestone_title,state,bugs,metrics
StringToWordVector	tag,title,wordvector
RemoveByName	Στα διανύσματα λέξεων που προέκυψαν από το προηγούμενο φίλτρο, για μικρές λέξεις
Reorder	Τοποθέτηση ιδιότητας ως Κλάση του Συνόλου Δεδομένων

Πίνακας 4.3-2 Παράδειγμα Εφαρμογής φίλτρου StringToWordVector με χρήση μετασχηματισμού Fourier και εντοπισμό ετυμολογικής ρίζας

StringToWordVector	-R9 -P tag_ -W10 -T -N 0 -S -stemmerweka.core.stemmers.LovinsStemmer -M1
--------------------	--

Κεφάλαιο 5: Επίδειξη και Τεκμηρίωση

5.1 Υλοποίηση πειραμάτων

Στα δύο προηγούμενα κεφάλαια περιγράφηκαν οι προκλήσεις που ενέχει η εξαγωγή δεδομένων σφαλμάτων από Αποθήκες Λογισμικού και υλοποιήθηκαν λύσεις για την αντιμετώπισή τους. Έγινε λόγος για το πώς είναι δυνατόν να ανακτηθούν οι αναφορές των σφαλμάτων (bug tickets) ως οντότητες, ώστε να εξαχθεί η άμεση πληροφορία που αυτές φέρουν. Στη συνέχεια εξετάστηκαν οι περιγραφές σε φυσική γλώσσα των αναφορών από τους ίδιους τους χρήστες που τις υπέβαλλαν. Εντοπίστηκε το εκμεταλλεύσιμο τμήμα των περιγραφών με Επεξεργασία Φυσικής Γλώσσας. Σε εκείνο το σημείο επιλέχθηκε να γίνει Σημασιολογική Ανάλυση ώστε να αναγνωριστεί η ουσία της περιγραφής ενός σφάλματος.

Σκοπός ήταν να δημιουργηθεί ένα ισχυρό σύνολο δεδομένων το οποίο να μη βασίζεται μόνο σε τύπους δεδομένων συνηθισμένων σε μία τεχνική αναφορά, αλλά να έχει ενισχυθεί από τη μετα-πληροφορία που προέκυψε από την εκμετάλλευση της φυσικής περιγραφής.

Για να ελεγχθεί η υπόθεση ότι η γνώση μας για ένα σφάλμα ενισχύεται με τον παραπάνω τρόπο, διεξάγαμε κάποια ενδεικτικά πειράματα εξόρυξης δεδομένων πάνω στο Σύνολο Δεδομένων που δημιουργήσαμε. Ο στόχος της πραγματοποίησης αυτών των δοκιμών είναι πολλαπλός:

- Να μετρηθεί ποιοτικά η διαφοροποίηση στη δυνατότητα εξαγωγής ασφαλών συμπερασμάτων για τα σφάλματα Συνόλου Δεδομένων με την προσθήκη της μετα-πληροφορίας.
- Να επιβεβαιωθεί ότι το Σύνολο Δεδομένων που εξάχθηκε είναι αποδεκτό ως υποκείμενο τεχνικών εξόρυξης δεδομένων.
- Να διερευνηθούν πιθανές χρήσεις του Συνόλου Δεδομένων στην Εξόρυξη Δεδομένων Λογισμικού και να εξετασθεί υπό ποιες συνθήκες είναι δυνατόν να βελτιωθεί.

Οι τεχνικές των προηγούμενων κεφαλαίων επιτρέπουν να κατασκευαστούν διαφορετικές εκφάνσεις του Συνόλου Δεδομένων, ανάλογα με το ποια επιθυμούμε να είναι η ενσωμάτωση της μετα-πληροφορίας σε αυτό. Υπενθυμίζεται ότι στις 12 βασικές ιδιότητες του ticket που περιέχουν αρχική πληροφορία, έχουν προστεθεί άλλες 3:

- Ένα διάνυσμα λέξεων-κλειδιών για κάθε ticket (ιδιότητα wordvector)
- Ένας πιθανός τύπος σφάλματος (ιδιότητα bugs)
- Ένα πιθανό χαρακτηριστικό αξιολόγησης Ποιότητας Λογισμικού (ιδιότητα metrics).

Χρησιμοποιώντας κάποιες από αυτές τις ιδιότητες ως μεταβλητές ελέγχου, θα προσπαθήσουμε να κατατάξουμε τα tickets ως προς τον πιθανό τύπο του bug και την κατάσταση ως προς την επίλυσή του (state).

Στη διάθεσή μας έχουμε μία σειρά από διαφορετικές εκφάνσεις του Συνόλου Δεδομένων στις οποίες διαφέρει ο βαθμός ενσωμάτωσης της μεταπληροφορίας. Κάθε σύνολο από tickets έχει κατασκευαστεί με ένα αριθμό από θέματα (5,10,20,25) που προέκυψαν από την LDA ανάλυση με ή χωρίς την επιπλέον λέξη “ruby” στο διάνυσμα των λέξεων-κλειδιών. Επομένως μία ακόμη παράμετρος που πρέπει να εξετάσουμε είναι η συμπεριφορά των αλγορίθμων κατάταξης σε σχέση με το πλήθος των topics.

5.1.1 Πρώτο Πείραμα: Επιλογή Βέλτιστου Αλγορίθμου

Σε πρώτη φάση, θα επιλέξουμε τον αλγόριθμο κατάταξης που είναι πιο αποτελεσματικός επί των δεδομένων που διαθέτουμε. Χρησιμοποιούμε τα Datasets που διαθέτουν 5,10 και 20 topics για τους αλγόριθμους C5.0, RandomTree και RandomForest. Ιδιότητα Ελέγχου είναι η Metrics και η κατάταξη γίνεται στην ιδιότητα Bugs. Τα αποτελέσματα παρουσιάζονται στον Πίνακα 5.1-1.

Πίνακας 5.1-1 Αποτελέσματα Πρώτου Πειράματος για επιλογή Αλγορίθμου

Αλγόριθμοι		C5.0	C5.0	C5.0	RandomTree	RandomTree	RandomForest	RandomForest
Παράμετροι		-C 0.25- M 2 -J	-C 0.45- M 2 -J	-C 0.35-M 2 -J	-M 1.0 -S 1	-K 0 -M 1.0 -S 1 -depth 2	-I 10 -K 0 -S 1	-I 10 -K 0 -S 1 - depth 2
Dataset	Topics No							
Παράθεση "ruby"	5 + metrics attr.	44.52	43.19	43.30	37.44	38.13	42.10	39.38
	5 - metrics attr.	37.24	33.73	33.80	33.84	36.30	35.95	37.09
	Μεταβολή (%)	19.55	28.05	28.11	10.64	5.04	17.11	6.17
Χωρίς Παράθεση "ruby"	5 + metrics attr.	81.18	81.19	81.22	56.76	51.50	77.55	72.81
	5 - metrics attr.	37.24	33.73	33.80	34.52	36.20	36.18	37.12
	Μεταβολή (%)	117.99	140.71	140.30	64.43	42.27	114.34	96.15
Παράθεση "ruby"	10 + metrics attr.	67.56	67.56	67.55	56.44	57.29	59.98	59.88
	10 - metrics attr.	55.93	56.05	55.98	51.76	54.81	53.13	54.29
	Μεταβολή (%)	20.79	20.54	20.67	9.04	4.52	12.89	10.30
Χωρίς Παράθεση "ruby"	10 + metrics attr.	82.12	82.14	82.11	72.63	74.41	77.10	75.13
	10 - metrics attr.	74.03	74.03	74.03	67.24	71.66	72.41	73.13
	Μεταβολή (%)	10.93	10.96	10.91	8.02	3.84	6.48	2.73

Παράθεση "ruby"	20 + metrics attr.	57.93	57.88	57.90	42.36	44.37	48.24	47.12
	20 - metrics attr.	42.02	36.94	42.02	36.79	39.92	38.58	41.02
	Μεταβολή (%)	37.86	56.69	37.79	15.14	11.15	25.04	14.87

Από τα αποτελέσματα συμπεραίνουμε σε γενικές γραμμές οι αλγόριθμοι συμπεριφέρονται όμοια όταν χρησιμοποιείται για την κατάταξή τους (με βάση την ιδιότητα Bugs) η επιπλέον πληροφορία των metrics, δηλαδή όταν χρησιμοποιούμε μόνο τις εγγενείς ιδιότητες των συνόλων. Όταν όμως λαμβάνεται υπόψη στην κατάταξη και η ιδιότητα των metrics ο C5.0 υπερτερεί σε σχέση με τους άλλους αλγόριθμους (RandomTree και RandomForest). *Λόγω της καλής του απόδοσης ο αλγόριθμος κατάταξη C5.0 θα αποτελέσει τον βασικό αλγόριθμο των πειραμάτων μας.*

Σε κάθε περίπτωση όμως έχουμε μία πρώτη ένδειξη ότι όντως υπάρχει βελτίωση στην κατάταξη των tickets με τη χρήση της μεταπληροφορίας.

5.1.2 Πείραμα 2ο: Σύγκριση Κατασκευασμένων Συνόλων Δεδομένων

Στο πείραμα αυτό εξετάζεται εάν και πόσο επηρεάζεται η απόδοση των αλγόριθμων κατάταξης από τον αριθμό των topics που χρησιμοποιήθηκαν και ανάλογα με τη χρήση ή όχι της επιπρόσθετης λέξης "ruby". Τα αποτελέσματα των πειραμάτων φαίνονται στον Πίνακας 5.1-2.

Πίνακας 5.1-2 Αποτελέσματα Δεύτερου Πειράματος για Σύγκριση των Datasets

Dataset	Αλγόριθμοι	C5.0	C5.0	C5.0	C5.0
	Παράμετροι	-C 0.55-M 2 -J	-C 0.45-M 2 -J	-C 0.35-M 2 -J	-C 0.25-M 2 -J
Παράθεση "ruby"	5 Topics + metrics attr.	43.14	43.19	43.30	44.52
	5 Topics - metrics	33.38	33.73	33.80	37.24
	Μεταβολή (%)	29.24	28.05	28.11	19.55
Χωρίς Παράθεση "ruby"	5 + metrics attr.	82.38	81.19	81.22	81.18
	5 - metrics attr.	33.38	33.73	33.80	37.24
	Μεταβολή (%)	146.79	140.71	140.30	117.99
Παράθεση "ruby"	10 + metrics attr.	64.58	67.56	67.55	67.56
	10 - metrics attr.	51.14	56.05	55.98	55.93
	Μεταβολή (%)	26.28	20.54	20.67	20.79
Χωρίς Παράθεση "ruby"	10 + metrics attr.	81.20	82.14	82.11	82.12
	10 - metrics attr.	69.13	74.03	74.03	74.03

	Μεταβολή (%)	17.46	10.96	10.91	10.93
Παράθεση "ruby"	20 Topics + metrics attr.	52.99	57.88	57.90	57.93
	20 Topics – metrics attr.	36.76	36.94	42.02	42.02
	Μεταβολή (%)	44.15	56.69	37.79	37.86
Χωρίς Παράθεση "ruby"	20 + metrics attr.	63.88	66.32	66.29	66.49
	20 – metrics attr.	53.51	63.23	63.23	63.23
	Μεταβολή (%)	19.38	4.89	4.84	5.16
Παράθεση "ruby"	25 Topics + metrics attr.	42.53	43.55	43.47	44.76
	25 Topics – metrics attr.	29.03	29.03	29.05	32.27
	Μεταβολή (%)	46.50	50.02	49.64	38.70
Χωρίς Παράθεση "ruby"	25 + metrics attr.	60.22	61.32	61.87	61.93
	25 – metrics attr.	35.36	35.36	43.18	43.16
	Μεταβολή (%)	70.31	73.42	43.28	43.49

Από τα αποτελέσματα του αλγόριθμου κατάταξης για τις διάφορες τιμές των παραμέτρων του συμπεραίνουμε αρχικά ότι σε κάθε περίπτωση τα αποτελέσματα κατάταξης είναι ενθαρρυντικά. Παρατηρούμε με τη χρήση της μετα-πληροφορίας που περιέχεται στην ιδιότητα Metrics, η βελτίωση της κατάταξης είναι ιδιαίτερα σημαντική σε κάθε εκδοχή του πειράματος. Αξίζει να σημειωθεί ότι ο αλγόριθμος κατάταξης συμπεριφέρεται ακόμη καλύτερα όταν δε χρησιμοποιείται η λέξη "ruby". Το γεγονός αυτός είναι δείγμα της αξιοπιστίας του Συνόλου Δεδομένου και της ανθεκτικότητάς του. Δεν μπορεί να εξαχθεί κάποιο άμεσο συμπέρασμα για την επίδραση του αριθμού των topics και χρειάζεται περισσότερη πειραματική μελέτη.

5.2 Αναγνώριση συσχετίσεων ανάμεσα στα εγγενή και τα επίκτητα χαρακτηριστικά του σετ

Η προσπάθεια αναγνώρισης της κατάστασης ενός σφάλματος, εν προκειμένω της ιδιότητας state (open, close, κλπ) , είναι ένας σημαντικός στόχος. Τυχόν επιτυχής αντιμετώπιση αυτού του ζητήματος θα έχει άμεση επίδραση στην εφαρμοσμένη Τεχνολογία Λογισμικού και όχι μόνο.

Στην παρούσα εργασία μελετήθηκε η δυνατότητα να εξαχθούν συμπεράσματα για την κατάσταση του σφάλματος με το χειρισμό των ιδιοτήτων bugs και metrics. Εντούτοις, τα αποτελέσματα δεν έχουν στατιστική σημασία.

Το ζήτημα απαιτεί περαιτέρω μελέτη και πιθανόν μία επέκταση του πεδίο λήψεως δεδομένων στα δεδομένα διαχείρισης εκδόσεων, και ιδιαίτερα στα commit headers, βελτώνει την αντίληψή μας για το ζήτημα.

Κεφάλαιο 6: Συμπεράσματα – Μελλοντική Εργασία

6.1 Σύνοψη - Συμπεράσματα

Η διπλωματική αυτή ξεκίνησε ως μία προσπάθεια να μελετηθεί η δυνατότητα ανακάλυψης γνώσης από δεδομένα που είναι αποθηκευμένα σε αποθήκες λογισμικού. Η σχετική βιβλιογραφία αποκάλυψε ότι υπάρχει κενό στη μελέτη των προγραμματιστικών σφαλμάτων, όπως αυτά καταγράφονται κατά την ανάπτυξης ενός προϊόντος λογισμικού. Επιλέχθηκε να μελετηθούν οι αναφορές σφαλμάτων του λογισμικού Ruby On Rails. Σκοπός μας ήταν να δημιουργήσουμε ένα αξιόλογο σύνολο δεδομένων από αυτά τα σφάλματα. Ξεκινώντας από ακατέργαστα δεδομένα έγινε εξαγωγή και διήθηση πληροφορίας, και στη συνέχεια έγινε ανάλυση ως προς το νόημα της περιγραφής των σφαλμάτων σε φυσική γλώσσα. Εφαρμόστηκαν τεχνικές Επεξεργασίας Φυσικής Γλώσσας και αναλύθηκε σημασιολογικά το περιεχόμενο της περιγραφής. Το αποτέλεσμα της ανάλυσης ήταν η δημιουργία νέων, -παράγωγων- ιδιοτήτων ως προς την αναφορά σφάλματος. Για να ελέγχουμε τα αποτελέσματα της εργασίας, εκτελέσαμε πειράματα κατάταξης των αναφορών χρησιμοποιώντας την παράγωγη πληροφορία.

Η όλη ανάλυση που προηγήθηκε κατέδειξε πως είναι δυνατόν πληροφορία καθαρά τεχνικού χαρακτήρα να μελετηθεί χρησιμοποιώντας ένα πλήθος διαθέσιμων τεχνικών της Ανάκτησης Πληροφορίας. Οι δυνατότητες της υπάρχουσας τεχνολογίας εξόρυξης δεδομένων με τις κατάλληλες προσαρμογές μπορούν να υιοθετηθούν στο πεδίο των μερικώς δομημένων δεδομένων, όπως τα δεδομένα λογισμικού.

6.2 Μελλοντική Εργασία

Είναι σαφές ότι σκοπός της παρούσας εργασίας δεν ήταν εξάγει άμεσα συμπεράσματα για τα προγραμματιστικά σφάλματα του λογισμικού με το οποίο ασχολήθηκε. Βασικό μέλημα ήταν να ανοίξει το δρόμο για την περαιτέρω μελέτη δεδομένων από τέτοιες αποθήκες λογισμικού δείχνοντας ότι ένα στιβαρό σύνολο δεδομένων από προγραμματιστικά σφάλματα είναι εφικτό. Ιδιαίτερα, η εργασία που έγινε δεν περιορίζει ουσιαστικά την εφαρμογή της σε αναφορές σφαλμάτων άλλου λογισμικού, καθώς δεν έγινε καμία παραδοχή βασισμένη στα συγκεκριμένα δεδομένα στα οποία εφαρμόστηκε που να υποσκάπτει τη θεωρητική της βάση.

Αυτό σημαίνει ότι μελλοντικά είναι δυνατόν να εφαρμοστεί, με τις κατάλληλες προσαρμογές, και σε άλλα έργα λογισμικού, και όχι μόνο στο αντικείμενο των προγραμματιστικών σφαλμάτων. Επίσης, μιας και στόχος της παρούσας εργασίας ήταν να επιδείξει μία οδό σημασιολογικής αναγνώρισης των δεδομένων, ο μελλοντικός χρήστης μπορεί να επισκεφτεί ξανά κάθε στάδιο της μεθοδολογίας, ώστε να δοκιμάσει τις επιλογές που στην παρούσα εργασία δεν έγιναν ή να προσπαθήσει να πειραματιστεί βελτιστοποιώντας τις παραμέτρους των εργαλείων που χρησιμοποιήθηκαν.

Ολοκληρώνοντας το σκοπό της, η εργασία προσφέρει έναν παραμετροποιήσιμο μηχανισμό με το οποίο μπορεί ο καθένας να πειραματιστεί. Αυτό μπορεί να γίνει είτε δοκιμάζοντας να εξάγει συσχετίσεις ανάμεσα στα δεδομένα σφαλμάτων, είτε συνδυάζοντάς τα με δεδομένα που προέρχονται από το σύστημα διαχείρισης εκδόσεων του Ruby On Rails. Μπορεί ακόμα και να συγκρίνει τα σφάλματα αυτού του προϊόντος λογισμικού με δεδομένα σφαλμάτων άλλων προϊόντων και να περάσει στο στάδιο της σύγκρισης μεθοδολογιών της Τεχνολογίας Λογισμικού.

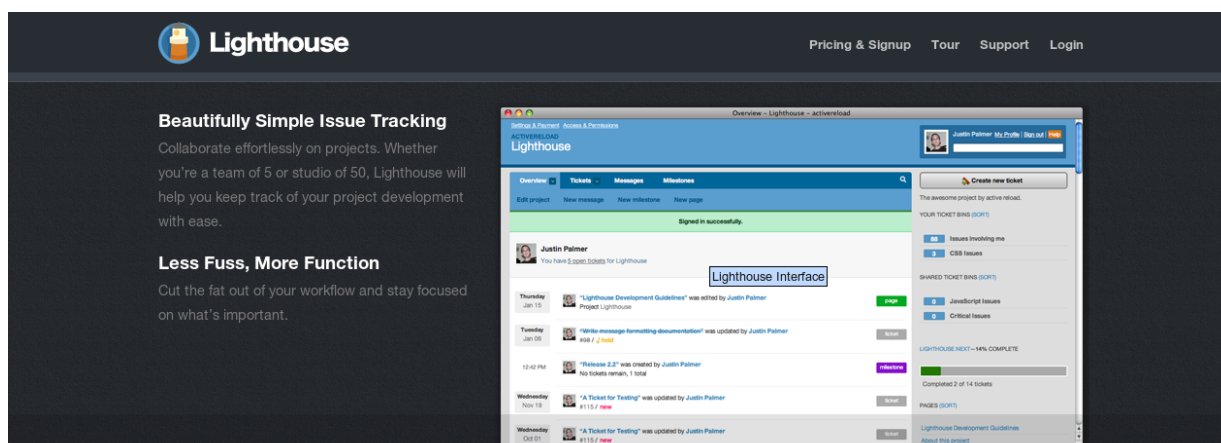
Όλες αυτές οι δυνατότητες που προκύπτουν είναι χαρακτηριστικό του χώρου της Εξόρυξης Δεδομένων από Αποθήκες Λογισμικού που έχοντας στη διάθεσή του έναν τεράστιο όγκο ανεκμετάλλευτων δεδομένων καλείται να τα διαχειριστεί με στόχο τη βελτίωση του παραγόμενου λογισμικού.

Παράρτημα Α

A.1 Το σύστημα καταγραφής σφαλμάτων Lighthouse

Το **Lighthouse app** (<http://Lighthouseapp.com>) είναι μία online υπηρεσία καταγραφής σφαλμάτων (bug ticketing and tracking) από την εταιρεία entp (<http://entp.com/>) που παρέχεται δωρεάν σε κοινότητες που αναπτύσσουν λογισμικού ανοικτού κώδικα, όπως αυτή του Ruby On Rails, και επί πληρωμή σε επιχειρήσεις που παράγουν εμπορικό λογισμικό.

Το Lighthouse παρέχει οργάνωση των tickets που ανοίγονται από τους προγραμματιστές, όταν αυτοί εντοπίζουν ένα bug ή προτείνουν μία αλλαγή στον πηγαίο κώδικα. Επίσης, προσφέρει ορισμένα εργαλεία παρακολούθησης της πορείας του project στο χρόνο ή της δραστηριότητας των χρηστών του μαζί με πλήρη στατιστικά στοιχεία για κάθε ticket που προστίθεται στη βάση δεδομένων που υποστηρίζει το έργο λογισμικού.



The screenshot displays the Lighthouse app interface. On the left, there's a dark sidebar with the Lighthouse logo and navigation links: Pricing & Signup, Tour, Support, and Login. The main content area shows a project overview for 'Lighthouse' with tabs for Overview, Tickets, Messages, and Milestones. A list of tickets is visible, including 'Lighthouse Development Guidelines' and 'Release 2.2'. A sidebar on the right contains a 'Create new ticket' button and filters for 'Issues involving me', 'Assigned issues', and 'Critical issues'. Below the screenshot, there's a promotional banner with the text 'Lighthouse will simplify your workflow so you can do the job you were hired to do. Try it for free. No credit card required.' and a 'Try Lighthouse for free' button. Below the banner, there are six feature highlights: 'Automatically Organize Tasks', 'Make Better Use Of Your Email', 'Consolidate Your Projects', 'Unify With Customer Support', 'Set And Meet Goals', and 'Share Documents & Images'.

Automatically Organize Tasks
As you create and tag issues they can be categorized behind the scenes automatically.

Make Better Use Of Your Email
Stay in your happy place. Create and reply to tickets directly from your inbox.

Consolidate Your Projects
You can see the status of all your projects in a simple overview and follow along with feeds.

Unify With Customer Support
Your staff can create new Lighthouse tickets directly from **Tender**, our customer support service.

Set And Meet Goals
Use milestones to help you plan features and establish release dates.

Share Documents & Images
Attach documents or images directly to tickets so anyone on your team can find them.

Εικόνα 6.2-1 Αρχική Σελίδα Lighthouseapp.com

Το γραφικό περιβάλλον διεπαφής του Lighthouse παρέχει μία περιορισμένη εποπτεία του πηγαίου κώδικα. Για την εμβριθέστερη μελέτη ενός project είναι δυνατή η χρήση του API (Application Programming Interface - Διασύνδεση Προγραμματισμού Εφαρμογών) του Lighthouse app, μιας βιβλιοθήκης εντολών που επιτρέπει την πρόσβαση στα δεδομένα των tickets που είναι αποθηκευμένα. Με τη χρήση του Lighthouse API είναι εφικτή η πρόσβαση στα δεδομένα των tickets συγκεντρωτικά

χρησιμοποιώντας XML αναπαράσταση ή, ticket προς ticket, υιοθετώντας το πρότυπο JSON για την σειριοποίηση (serializing) των tickets ως δομή δεδομένων. Το πρότυπο JSON και το πώς χρησιμοποιείται στην περίπτωση του Lighthouse ticket αναλύθηκε στην υποπαράγραφο (0).

Ένα ακόμη χαρακτηριστικό του Lighthouse είναι ότι προσφέρει διαλειτουργικότητα (Interoperability) με υπηρεσίες που υλοποιούν Συστήματα Διαχείρισης Εκδόσεων (όπως το git) και τις ανάλογες διαδικτυακές υπηρεσίες, (αντίστοιχα, github.com). Η συγκεκριμένη δυνατότητα προσφέρει άμεση σύνδεση ανάμεσα στην καταγραφή του σφάλματος σε ένα ticket και την αντιστοίχιση στο τμήμα πραγματικού πηγαίου κώδικα όπου διορθώνεται το σφάλμα. Με αυτόν τον τρόπο, καθίσταται ευκολότερη η εποπτεία στην ανάπτυξη του λογισμικού και ταχύτερη η υλοποίηση των tickets.

```
git commit -m "fixed the bug [#15 #16 state:committed]"
git commit -m "fixed the bug [#15 state:committed] [#16 state:committed milestone:next]"
```

Εικόνα 6.2-2 Ενημέρωση του Lighthouseapp μέσω commit στο Σύστημα Διαχείρισης Εκδόσεων git

A.2 Απεικόνιση του ticket #1800 σε JSON format

```
{
  "ticket": {
    "original_body": "When using map.resources with a model ending in 'base' it will not generate singular helpers.\r\n\r\nUsing script/generate scaffold databases, navigating to /databases will fail with `undefined local variable or method `new_database_path' for #<ActionView::Base:0xXXXXXX>`. This happens with anything ending in base, even something like `plant_base`.",
    "permalink": "resources-routing-ending-in-base",
    "user_name": "CancelProfileIsBroken",
    "updated_at": "2009-03-08T13:06:54Z",
    "project_id": 8994,
    "body_html": "<div><p>When using map.resources with a model ending in 'base' it will\nnot generate singular helpers.</p>\n<p>Using script/generate scaffold databases, navigating to\n/databases will fail with\n<code>undefined local variable or\nmethod</code>new_database_path' for #<code>. This happens with\nanything ending in base, even something\nlike</code>plant_base<code>.</code></p></div>",
    "original_body_html": "<div><p>When using map.resources with a model ending in 'base' it will\nnot generate singular helpers.</p>\n<p>Using script/generate scaffold databases, navigating to\n/databases will fail with\n<code>undefined local variable or\nmethod</code>new_database_path' for #<code>. This happens with\nanything ending in base, even something\nlike</code>plant_base<code>.</code></p></div>",
    "milestone_title": "2.x",
    "title": "Resources Routing ending in 'base'",
    "number": 1800,
    "body": "When using map.resources with a model ending in 'base' it will not generate singular helpers.\r\n\r\nUsing script/generate scaffold databases, navigating to /databases will fail with `undefined local variable or method `new_database_path' for #<ActionView::Base:0xXXXXXX>`. This happens with anything ending in base, even something like `plant_base`.",
    "versions": [
      {
        "permalink": "resources-routing-ending-in-base",
        "user_name": "Samuel Kadolph",
        "updated_at": "2009-01-26T20:56:08Z",
        "project_id": 8994,
```



```
"body_html": "<div><p>When using map.resources with a model ending in 'base' it will\nnot generate singular helpers.</p>\n<p>Using script/generate scaffold databases, navigating to\n/databases will fail with\n<code>undefined local variable or\nmethod</code>new_database_path'\nfor\n#&lt;ActionView::Base:0xXXXXXX&gt;</code>. This happens with\nanything ending in base, even something\nlike</code>plant_base<code>.</code></p></div>",
```

```
"milestone_title": "2.x",
```

```
"title": "Resources Routing ending in 'base'",
```

```
"number": 1800,
```

```
"body": "When using map.resources with a model ending in 'base' it will not generate singular helpers.\r\n\r\nUsing script/generate scaffold databases, navigating to /databases will fail with `undefined local variable or method `new_database_path' for #<ActionView::Base:0xXXXXXX>`. This happens with anything ending in base, even something like `plant_base`.",
```

```
"creator_id": 45053,
```

```
"url": "http://rails.lighthouseapp.com/projects/8994/tickets/1800",
```

```
"tag": "generate resources routing \"routing error\" scaffold",
```

```
"priority": 0,
```

```
"attachments_count": 0,
```

```
"diffable_attributes": {
```

```
},
```

```
"closed": false,
```

```
"user_id": 45053,
```

```
"assigned_user_id": null,
```

```
"creator_name": "Samuel Kadolph",
```

```
"state": "new",
```

```
"milestone_id": 9903,
```

```
"created_at": "2009-01-26T20:56:03Z"
```

```
},
```

```
"permalink": "resources-routing-ending-in-base",
```

```
"user_name": "CancelProfileIsBroken",
```

```
"updated_at": "2009-03-08T13:06:54Z",
```

```
"project_id": 8994,
```

```
"body_html": "<div><p>We should handle \"database\" in particular correctly since\n<a href=\"/projects/8994/tickets/1942\" title=\"Ticket #1942\">#1942</a> was committed.</p>\n<p>The more general case is because the pluralization doesn't\nround-trip:</p>\n<p>\"base\".pluralize.singularize =&gt;\n\"basis\"</p>\n<p>You can workaround by setting up your own inflections. <a href=\"/projects/8994/tickets/2046\" title=\"Ticket #2046\">#2046</a> is the\nsame issue.</p></div>",
```

```
"milestone_title": "2.x",
```

```
"title": "Resources Routing ending in 'base'",
```

```
"number": 1800,
```

```
"body": "We should handle \"database\" in particular correctly since #1942 was committed. \r\n\r\nThe more general case is because the pluralization doesn't round-trip:\r\n\r\n\"base\".pluralize.singularize =>\n\"basis\"\r\n\r\nYou can workaround by setting up your own inflections. #2046 is the same issue.",
```

```
"creator_id": 45053,
```

```
"url": "http://rails.lighthouseapp.com/projects/8994/tickets/1800",
```

```
"tag": "generate resources routing \"routing error\" scaffold",
```

```
"priority": 0,
```

```
"attachments_count": 0,
```

```
"diffable_attributes": {
```

```
"state": "new"
```

```
},
```

```
"closed": true,
```

```

    "user_id": 7211,
    "assigned_user_id": null,
    "creator_name": "Samuel Kadolph",
    "state": "wontfix",
    "milestone_id": 9903,
    "created_at": "2009-03-08T13:06:49Z"
  }
],
"milestone_due_on": null,
"creator_id": 45053,
"url": "http://rails.lighthouseapp.com/projects/8994/tickets/1800",
"tag": "generate resources routing \"routing error\" scaffold",
"priority": 1096,
"attachments_count": 0,
"raw_data": null,
"closed": true,
"latest_body": "When using map.resources with a model ending in 'base' it will not generate singular
helpers.\r\n\r\nUsing script/generate scaffold databases, navigating to /databases will fail with `undefined local
variable or method `new_database_path' for #<ActionView::Base:0xxxxxxx>`. This happens with anything ending
in base, even something like `plant_base`.",
  "user_id": 7211,
  "assigned_user_id": null,
  "creator_name": "Samuel Kadolph",
  "state": "wontfix",
  "milestone_id": 9903,
  "created_at": "2009-01-26T20:56:03Z"
}
}
"user_id": 7211,
"assigned_user_id": null,
"creator_name": "Samuel Kadolph",
"state": "wontfix",
"milestone_id": 9903,
"created_at": "2009-01-26T20:56:03Z"
}
}

```

Viewer Text

JSON

ticket

- original_body : "When using map.resources with a model ending in 'base' it will not generate singular helpers. Using script/generate scaffold databases, navigating to /databases will fail with 'undefined local variable or method 'new_database"
- permalink : "resources-routing-ending-in-base"
- user_name : "CancelProfilesBroken"
- updated_at : "2009-03-08T13:06:54Z"
- project_id : 8994
- body_html : "

When using map.resources with a model ending in 'base' it will not generate singular helpers.
Using script/generate scaffold databases, navigating to /databases will fail with undefined local variable or methodnew_database_path' for #. This happens with anything ending in base, even something likepla

"

original_body_html : "

When using map.resources with a model ending in 'base' it will not generate singular helpers.
Using script/generate scaffold databases, navigating to /databases will fail with undefined local variable or methodnew_database_path' for #. This happens with anything ending in base, even something likepla

"

- milestone_title : "2.x"
- title : "Resources Routing ending in 'base'"
- number : 1800
- body : "When using map.resources with a model ending in 'base' it will not generate singular helpers. Using script/generate scaffold databases, navigating to /databases will fail with 'undefined local variable or method 'new_database_path"

versions

0

- permalink : "resources-routing-ending-in-base"
- user_name : "Samuel Kadolph"
- updated_at : "2009-01-26T20:56:08Z"
- project_id : 8994
- body_html : "

When using map.resources with a model ending in 'base' it will not generate singular helpers.
Using script/generate scaffold databases, navigating to /databases will fail with undefined local variable or methodnew_database_path' for #<ActionView::Base:0xXXXXXX>. This happens with anything ending in base,

"

- milestone_title : "2.x"
- title : "Resources Routing ending in 'base'"
- number : 1800
- body : "When using map.resources with a model ending in 'base' it will not generate singular helpers. Using script/generate scaffold databases, navigating to /databases will fail with 'undefined local variable or method 'new_databas
- creator_id : 45053
- url : "http://rails.lighthouseapp.com/projects/8994/tickets/1800"
- tag : "generate resources routing 'routing error' scaffold"
- priority : 0
- attachments_count : 0
- diffable_attributes
- closed : false
- user_id : 45053

jsonviewer.stack.hu/# GO! Next Previous

Viewer Text

```

    "permissions": "Resources Routing ending in 'base'",
    "user_name": "Samuel Kadolph",
    "updated_at": "2009-01-26T20:56:08Z",
    "project_id": 8994,
    "body_html": ""
  },
  {
    "milestone_title": "2.x",
    "title": "Resources Routing ending in 'base'",
    "number": 1800,
    "body": "When using map.resources with a model ending in 'base' it will not generate singular helpers. Using script/generate scaffold databases, navigating to /databases will fail with 'undefined local variable or method 'new_databases'. This happens with anything ending in base, ",
    "creator_id": 45053,
    "url": "http://rails.lighthouseapp.com/projects/8994/tickets/1800",
    "tag": "generate resources routing 'routing error' scaffold",
    "priority": 0,
    "attachments_count": 0,
    "diffable_attributes": {
      "closed": false
    },
    "user_id": 45053,
    "assigned_user_id": null,
    "creator_name": "Samuel Kadolph",
    "state": "new",
    "milestone_id": 9903,
    "created_at": "2009-01-26T20:56:03Z"
  }
]
1
  "milestone_due_on": null,
  "creator_id": 45053,
  "url": "http://rails.lighthouseapp.com/projects/8994/tickets/1800",
  "tag": "generate resources routing 'routing error' scaffold",
  "priority": 1096,
  "attachments_count": 0,
  "raw_data": null,
  "closed": true,
  "latest_body": "When using map.resources with a model ending in 'base' it will not generate singular helpers. Using script/generate scaffold databases, navigating to /databases will fail with 'undefined local variable or method 'new_databases'. This happens with anything ending in base, ",
  "user_id": 7211,
  "assigned_user_id": null,
  "creator_name": "Samuel Kadolph",
  "state": "wontfix",
  "milestone_id": 9903,
  "created_at": "2009-01-26T20:56:03Z"
}

```

jsonviewer.stack.hu/#

GO! Next Previous

Βιβλιογραφία

Free Software Foundation. (n.d.). *CVS - Concurrent Versions System*. Retrieved 10 30, 2010, from <http://www.nongnu.org/cvs/>

Apache Software Foundation. (n.d.). *log--Print out log information for files*. Retrieved 11 02, 2010, from <http://www.cvsnt.org/manual/html/log.html>

Apache SubVersion. (n.d.). Retrieved 11 02, 2010, from <http://subversion.apache.org/>

Bennett, C. H., Li, M., Vitanyi, P., & Zurek, W. (1998). Information Distance. *IEEE Transactions in Informational Theory* , 1407-1423.

Bieman, J., & Dinh-Trong, T. (2005). The FreeBSD project: A replication case study of open source. *IEE Transactions on Software Engineering* (31(6):481-494).

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research* , 993-1022.

Cilbrasy, V. (2007). The Google Similarity Distance. *IEEE Transactions in Knowledge and Data Engineering* , 370-383.

Cilibrasi, R. L., & Vitanyi, P. M. (2005). The Google Similarity Distance. *IEEE ITSOC*. Rotorua, NZ.

codehaus. (n.d.). <http://jackson.codehaus.org/>. Retrieved 11 02, 2010, from <http://jackson.codehaus.org/>

Dig, D., & Johnson, R. (2006). How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution:Research and Practice* , 83-107.

Dig, D., & Johnson, R. (2006). How do APIs evolve? A story of refactoring.

Dig, D., Comertoglu, C., Marinov, D., & Johnson, R. (2006). Automated detection of refactorings in evolving components . *EEOOP*.

Gall, H., Pinzger, M., & Fischer, M. (2003). Populating a release history database from version control and bug tracking systems. *Proceedings of 19th IEE Intrnational Conference on Software Maintenance* , 23-32.

Gall, H., Ratzinger, J., Oberleitner, J., & Fischer, M. (2005). Mining evolution data of a product family. *Proceedings 3rd International Workshop on Mining Software Repositories*.

Gall, Jayayeri, & Krajewski. (2003). CVS release history data for detecting logical couplings. *Proceeding of the 6th International Workshop on Principles of Software Evolution (IWPSE '03)*. IEEE Computer Society.

German, D., & Hindle, A. (2005). SCQL: A formal model and a query language for source control repositories. *Proceeding 2nd International Conference on Mining Software Repositories*.

Google Inc. (n.d.). *Google Custom Search API*. Retrieved 11 02, 2010, from <http://code.google.com/intl/el-GR/apis/customsearch/v1/overview.html>

Hofmann, T. Probabilistic Latent Semantic Indexing. *22nd Intn'l SIGIR Conference on Research and Development in Information Retrieval*.

<http://htmlcleaner.sourceforge.net/>. (n.d.). Retrieved 11 02, 2010, from <http://htmlcleaner.sourceforge.net/>

Huang, S.-K., & Liu, K.-M. (2005). Mining versions histories to verify the learning process of legitimate peripheral participants. *2nd International Workshop on MSR* (pp. 78-84). New York: ACM Press.

ISO/IEC. (2001). *Software engineering -- Product quality -- Part 1: Quality model*. Retrieved from <http://www.iso.org/>:
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749

Li, M., Chen, X., Li, X., Ma, B., & Vitanyi, P. (2004). The similarity metric. *IEEE Transactions Information Theory* , 3250-3264.

Linus Torvalds. (n.d.). *Git Version Control*. Retrieved 11 01, 2010, from <http://git-scm.com/>

Machine Learning Group at University of Waikato. (n.d.). Retrieved 11 01, 2010, from Weka 3: Data Mining Software in Java: <http://www.cs.waikato.ac.nz/ml/weka/>

manage software development. (n.d.). Retrieved 11 02, 2010, from <http://www.bugzilla.org/>

Manning, C., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge: MIT Press.

Minnen G., C. J. (2001, 7(3)). Applied morphological processing of English. *Natural Language Engineering* , pp. 207-223.

Mockus, A., Fielding, T., & Herbsleb, D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering* , 309-346.

- Purushotthaman, R., & Perry, D. (2004). Towards understanding the rhetoric of small changes. *1st International Conference on Mining Software Repositories*.
- Raghavan, S., Rohana, R., Leon, D., Podgurski, A., & Augustin, V. D. (2004). Dex: A semantic-graph differencing tool for studying changes in large code bases. *20th IEEE International Conference on Software Maintenance*.
- Robles, G., Gonzalez-Barahona, J., Michmlayr, M., & Amor, J. (2006). Mining large software compilations over time: Another perspective on software evolution. *3rd International Workshop on Mining Software Repositories*. ACM Press.
- Ruby on Rails . (n.d.). *Ruby on Rails Homepage*. Retrieved 11 02, 2010, from Ruby on Rails : <http://rubyonrails.org/>
- Sager, T., Bernstein, A., Pinzger, M., & Kiefer, C. (2006). Detecting similar java classes using tree algorithms. *3rd Workshop on MSR*.
- Semantic similarity*. (n.d.). Retrieved 11 02, 2010, from http://en.wikipedia.org/wiki/Semantic_similarity
- Silwerski, J., Zimmermann, T., & Zeller, A. (2005). When do changes induce fixes? *2nd International Workshop on MSR* (pp. 24-28). ACM Press.
- Stanford Uni. (n.d.). *The Stanford Parser: A statistical parser*. Retrieved 11 01, 2010, from <http://nlp.stanford.edu/software/lex-parser.shtml>
- Steyvers, M., & Griffiths, T. Probabilistic Topic Models. In L. Ernbaum, *Latent Semantic Analysis: A Road to Meaning*.
- Wikipedia. (n.d.). *LDA* . Retrieved 10 25, 2010, from http://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
- Zimmermann, T., & Weißgerber, P. (2004). *Preprocessing CVS Data for Fine-Grained Analysis*.
- Zimmermann, T., Weißgerber, P., Diehl, S., & Zeller, A. (2004). *Mining Version Histories to Guide Software Changes*.