ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΡΓΑΛΕΙΩΝ ΕΝΤΟΠΙΣΜΟΥ ΤΕΧΝΙΚΟΥ ΧΡΕΟΥΣ / EXPERIMENTAL ANALYSIS OF TECHNICAL DEBT TRACKING TOOLS

Διπλωματική Εργασία
της
Μόσχου Αθανασίας

Θεσσαλονίκη Ιούνιος 2019

ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΡΓΑΛΕΙΩΝ ΕΝΤΟΠΙΣΜΟΥ ΤΕΧΝΙΚΟΥ ΧΡΕΟΥΣ / EXPERIMENTAL ANALYSIS OF TECHNICAL DEPT TRACKING TOOLS

Μόσχου Αθανασία
Πτυχίο Τμήματος Οικονομικών Επιστημών, Πανεπιστήμιο Μακεδονίας, 2011
Πτυχίο Τμήματος Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας, 2017

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Χατζηγεωργίου Αλέξανδρος

Εγκρίθηκε από την τριμελή επιτροπή

Χατζηγεωργίου Αλέξανδρος	Γεωργιάδης Χρήστος	Ξυνόγαλος Στυλιανός

Μόσχου Αθανασία

Περίληψη

Η μελέτη και η διαχείριση του τεχνικού χρέους κερδίζουν διαρκώς έδαφος στις σύγχρονες επιχειρήσεις λογισμικού. Οι μετρικές και οι μεθοδολογίες αποτίμησής του ποικίλουν και υπάρχουν αρκετά εργαλεία που υλοποιούν διαφορετικές προσεγγίσεις. Στα πλαίσια αυτής της εργασίας πραγματοποιείται αρχικά μία έρευνα στους τρόπους μέτρησης του τεχνικού χρέους καθώς και στα εργαλεία που έχουν άμεση αναφορά σε αυτό.

Στο πρακτικό κομμάτι της εργασίας επιλέχθηκαν συνολικά δέκα έργα λογισμικού τα οποία διαφέρουν σε μέγεθος και στην φύση των προβλημάτων που διαχειρίζονται, προκειμένου να αναλυθούν από τρία εργαλεία το SonarQube, το Squore και το CAST. Αφού συγκεντρώθηκαν τα αποτελέσματα της ανάλυσης του κώδικα η σκυτάλη περνάει στην στατιστική ανάλυση, η οποία σκοπό έχει να εξετάσει τον βαθμό στον οποίο συμφωνούν τα εργαλεία μεταξύ τους ως προς το ποιες είναι οι πιο προβληματικές κλάσεις σε κάθε έργο, αναφορικά με το τεχνικό χρέος.

Για την αξιολόγηση του βαθμού συμφωνίας των εργαλείων χρησιμοποιήθηκε ο δείκτης W του Kendall. Έπειτα εφαρμόστηκε η μεθοδολογία αρχετύπων προκειμένου να αναγνωριστούν και να μελετηθούν συγκεκριμένα μοτίβα τα οποία υπάρχουν στο σύνολο των προβληματικών κλάσεων. Στο τέλος της ανάλυσης εξετάζεται κατά πόσο αυτή η μελέτη και τα συμπεράσματα που προκύπτουν, μπορούν να γενικευτεί και σε άλλα έργα λογισμικού.

Γενικός στόχος της εργασίας είναι να εξετάσει το αν τα διαφορετικά εργαλεία αξιολόγησης κώδικα που κυκλοφορούν στην αγορά συμφωνούν μεταξύ τους και αν μπορεί το κάθε ένα να σταθεί μόνο του ή είναι πιο αποτελεσματικό να χρησιμοποιηθούν σε συνδυασμό. Τελικός σκοπός της εργασίας είναι η δημιουργία ενός benchmark με βάση την τομή των τριών εργαλείων.

Λέξεις Κλειδιά: Τεχνικό χρέος, benchmark, ποιότητα κώδικα, ανάλυση αρχετύπων

Abstract

The study and management of technical debt continually gains ground in modern software companies. There are many metrics and methodologies used to calculate it and several tools have been developed which approach it from different perspectives. The first part of this thesis examines the different methods used to calculate Technical Debt as long as the different tools mentioned in bibliography which serve this purpose.

For the second and more practical section of this thesis, ten open source projects have been selected to be analyzed with three different tools, SonarQube, Squore and CAST. These projects vary in size and the kind of problem they try to solve. After analyzing these projects and normalizing the results a statistical analysis takes place. Using it we try to find out firstly if these tools agree on the most problematic classes as far as Technical Debt is concerned.

In order to specify the degree of agreement between the tools, Kendall's W index was used. Then using the archetypal analysis, eight archetypes were produced. Grouping the classes depending on their similarity with each archetype a golden set of the common most problematic classes in all tools was formed. The last step was to check the generalization of this case study then build a benchmark with them.

The general purpose of this thesis is to examine if the different tools of calculating technical debt agree with each other. This is an effort to answer the question if each one of them can fulfill the needs of a company alone or a combination of two or more tools is needed for accurate results. Finally we want to build a benchmark to be used as a reference for other tools to compare their results with it, and produce a set which can be used in any neural network as training set, in order to identify problematic classes without the previous analysis.

Keywords: Technical Debt, Benchmark, Code quality, archetypal analysis

Πρόλογος – Ευχαριστίες

Είναι πολύ σημαντικό κίνητρο στην ζωή να θέτεις στόχους και να μην σταματάς να αγωνίζεσαι ούτε στιγμή για να τους πετύχεις. Κάνοντας αυτή την διαδρομή χρειάζεσαι άτομα δίπλα σου τα οποία να σε στηρίζουν και να σου συμπαραστέκονται. Θα ήθελα να πω ένα μεγάλο λοιπόν ευχαριστώ στην οικογένειά μου και στους γονείς μου Ασημάκη Μόσχο και Σοφία Πασχαλίδου που όλα αυτά τα χρόνια στάθηκαν δίπλα μου και στήριξαν κάθε απόφασή μου με όλες τους τις δυνάμεις. Ευχαριστώ για όλες τις πολύτιμες συμβουλές και για όλες εκείνες τις στιγμές που δώσανε το έναυσμα να πάω παρακάτω.

Ένα ευχαριστώ αξίζει και σε αρκετούς καθηγητές του Πανεπιστημίου Μακεδονίας τόσο του τμήματος Οικονομικών Επιστημών όσο και του τμήματος Εφαρμοσμένης Πληροφορικής που με βοήθησαν να αγαπήσω την γνώση. Ιδιαίτερο όμως ευχαριστώ οφείλω στον καθηγητή κύριο Αλέξανδρο Χατζηγεωργίου που από πολύ νωρίς πίστεψε στις δυνατότητές μου και με όρεξη και μεράκι συνέβαλε τα μέγιστα στην δουλειά που παρουσιάζεται σε αυτή την εργασία

Τέλος αυτή η εργασία δεν θα είχε ολοκληρωθεί εάν δεν υπήρχε η σημαντική βοήθεια και η καθοδήγηση μίας ομάδας ατόμων οι οποίοι με την ίδια όρεξη δούλεψαν ώστε να έχουμε στα χέρια μας αυτό το αποτέλεσμα. Αυτή η ομάδα αποτελείται από τους κυρίους Αμπατζόγλου Απόστολο, Μίττα Νικόλαο, Αμανατίδη Θεόδωρο και Αγγελή Ελευθέριο.

Όσο η εμπειρία μου μεγαλώνει στον τομέα της παραγωγής λογισμικού συνειδητοποιώ ολοένα και περισσότερο ότι δεν αρκεί μόνο να γνωρίζει να γράφεις κώδικα. Πρέπει να μπορείς να γράφεις καλό κώδικα. Είναι ένα καθημερινό στοίχημα που σε βοηθάει να βελτιώνεσαι διαρκώς και να κάνεις την μελλοντική δουλειά σου πιο εύκολη. Είναι λοιπόν εξαιρετικά ενδιαφέρον να γνωρίζεις εκ των προτέρων τι κάνει έναν κώδικα καλό και πως μπορείς να αναγνωρίζεις έγκαιρα τα προβλήματα πριν διογκωθούν. Αυτός είναι και ο λόγος για τον οποίο η εργασία αυτή θέλησα να κινηθεί εντός αυτού του πλαισίου.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Πίνακας πινάκων

Εισαγωγή

Πρόβλημα - Σημαντικότητα του θέματος

Τα έργα λογισμικού έχουν αρκετές ιδιαιτερότητες σε σχέση με τα απλά προϊόντα της αγοράς. Μία εξ αυτών είναι ότι καλούνται να διαχειριστούν αρκετά πολύπλοκα προβλήματα. Όσο η χρήση τους εξαπλώνεται τόσο τα προβλήματα γίνονται πιο σύνθετα και ταυτόχρονα ο χρόνος παράδοσης που απαιτείται από τους πελάτες περιορίζεται. Αυτό έχει ως αποτέλεσμα τις περισσότερες φορές να θυσιάζεται η ποιότητα προκειμένου να προλάβει η ομάδα ανάπτυξης τις προθεσμίες. Μπορεί λοιπόν ένα κομμάτι κώδικα να είναι λειτουργικό αλλά αν δεν έχει την απαραίτητη ποιότητα δεν μπορεί να είναι συντηρήσιμο και να υποβάλλεται εύκολα σε ελέγχους. Ως εκ τούτου είναι επικίνδυνο και εν δυνάμει μπορεί να προκαλέσει σημαντικά προβλήματα και να χαθούν τόσο χρηματικά κεφάλαια αλλά και ανθρώπινες ζωές.

Όταν ένα τμήμα κώδικα δεν έχει την απαραίτητη ποιότητα που απαιτείται αυτό δημιουργεί ένα χρέος για την εφαρμογή. Κάποια στιγμή αυτή η λανθασμένη υλοποίηση πρέπει να διορθωθεί, διαδικασία που απαιτεί επιπλέον χρόνο. Ειδικά εάν πάνω σε αυτή την υλοποίηση έχουν βασιστεί και άλλα νεότερα κομμάτια κώδικα τότε πρέπει να γίνει η αλλαγή με τέτοιο τρόπο ώστε να μην επηρεάσει αυτά τα κομμάτια. Εναλλακτικά θα πρέπει να μεταβληθούν και αυτά τα τμήματα κώδικα που επηρεάζονται αντίστοιχα, γεγονός που προσθέτει επιπλέον εργατοώρες. Έχουν γίνει λοιπόν προσπάθειες ώστε να γεφυρωθεί το χάσμα μεταξύ των μηχανικών λογισμικού που μπορούν να κατανοήσουν αυτή την αναγκαιότητα της αυξημένης ποιότητας και των διοικητικών στελεχών που για καιρό θεωρούσαν την ποιότητα κάτι ακριβό και αντιπαραγωγικό. Στην προσπάθεια αυτή το 1992 εμφανίστηκε ο όρος του τεχνικού χρέους ο οποίος μετέφερε όλες αυτές τις ανησυχίες από τον κόσμο της τεχνολογίας λογισμικού σε οικονομικούς όρους πολύ πιο οικείους και κατανοητούς από τις διοικήσεις.

Το τεχνικό χρέος είναι μία προσπάθεια ποσοτικοποίησης ενός ποιοτικού δείκτη. Αυτό σημαίνει πως το ειδικό βάρος που αντιστοιχίζεται σε κάθε ποιοτικό χαρακτηριστικό μπορεί να διαφέρει για τον καθένα που πραγματοποιεί την μέτρηση και ως εκ τούτου να διαφέρει και το τελικό αποτέλεσμα.

Στην προσπάθεια αποτίμησης του τεχνικού χρέους έχουν συμβάλει πολλές εταιρείες οι οποίες έχουν αναπτύξει εργαλεία για να αξιολογούν την ποιότητα του κώδικα και να μετράνε το τεχνικό χρέος που προκύπτει από τις περιπτώσεις που δεν ακολουθούνται οι ενδεδειγμένες τεχνικές. Το κύριο ερώτημα που αιωρείται λοιπόν είναι το ποιο από όλα τα εργαλεία είναι πιο αξιόπιστο και θα έπρεπε να προτιμάται από τους ερευνητές και από τις επιχειρήσεις. Η απάντηση δεν μπορεί να δοθεί κατ' απόλυτο τρόπο καθώς δεν υπάρχει κάποια συγκεκριμένη βάση σύγκρισης ώστε να δούμε πιο εργαλείο έχει καλύτερα αποτελέσματα σε σχέση με αυτή την βάση. Αυτό είναι και το πρόβλημα το οποίο εντοπίστηκε και εξετάζεται σε αυτή την διπλωματική, προσπαθώντας να δοθεί μία πρώτη λύση δημιουργώντας μία βάση σύγκρισης ως σημείο μελλοντικής αναφοράς.

Σκοπός και Στόχοι

Η μελέτη αυτή στοχεύει να βοηθήσει στην κατεύθυνση της απάντηση του ερωτήματος "Ποιο εργαλείο πρέπει να προτιμάται και γιατί". Πραγματοποιείται η συλλογή ορισμένων γνωστών και δημοφιλών έργων ανοιχτού κώδικα τα οποία θα αναλυθούν από ένα σύνολο εργαλείων αποτίμησης ποιότητας λογισμικού. Αφού ολοκληρωθεί η ανάλυση και ληφθούν τα αποτελέσματα κάθε εργαλείου θα γίνει η σύγκρισή τους.

Είναι σημαντικό να δούμε εάν μία προβληματική κλάση για ένα εργαλείο αρχικά είναι προβληματική και για τα υπόλοιπα. Να εξεταστεί δηλαδή η τομή των εργαλείων τόσο μεταξύ του συνόλου τους όσο και των επιμέρους υποσυνόλων. Έχει ενδιαφέρον να μελετηθεί εάν δύο από τα εργαλεία συμφωνούν περισσότερο μεταξύ τους, και αν συμβαίνει αυτό να εντοπιστούν οι αιτίες τους. Σημαντική είναι επίσης και η κατάταξη των προβληματικών κλάσεων για κάθε εργαλείο και ο βαθμός στον οποίο συμφωνούν οι κατατάξεις μεταξύ τους.

Σκοπός της μελέτης είναι η δημιουργία ενός benchmark, μιας βάσης από το σύνολο των κοινών προβληματικών κλάσεων των εργαλείων. Αυτή η βάση μπορεί να αξιοποιηθεί και να έχει διπλό όφελος. Εάν μία κλάση είναι προβληματική και για τα τρία εργαλεία τότε οποιαδήποτε άλλη κλάση με παρόμοια χαρακτηριστικά και μετρικές επίσης θα χαρακτηριστεί ως προβληματική σε πιθανή της ανάλυση. Επιπλέον οποιοδήποτε άλλο εργαλείο, είτε από τα

ήδη υπάρχοντα που για κάποιον λόγο δεν συμμετέχουν στην μελέτη, είτε μελλοντικό θα μπορεί αναλύοντας τα ίδια έργα λογισμικού να συγκρίνει και αυτό τα αποτελέσματά του με τα εργαλεία που εξετάστηκαν.

Ερωτήματα και Υποθέσεις

Τα κυριότερα ερωτήματα που θα γίνει προσπάθεια να απαντηθούν κατά την διάρκεια της μελέτης είναι τα εξής:

Ε1. Υπάρχει τομή μεταξύ των προβληματικών κλάσεων που εντοπίζουν διαφορετικά εργαλεία ανάλυσης κώδικα;

Ε2. Σε τι βαθμό συμφωνούν τα εργαλεία αναφορικά με την κατάταξη των πιο προβληματικών κλάσεων;

Ε3. Μπορεί να δημιουργηθεί ένα benchmark με τα δεδομένα των εργαλείων που έχουν προκύψει;

Οι υποθέσεις που έχουν γίνει για την διεξαγωγή της μελέτης ήταν οι ελάχιστες δυνατές προκειμένου αφενός το περιβάλλον να είναι ελεγχόμενο χωρίς ωστόσο αυτό να περιορίζει την γενίκευση του μοντέλου. Η βασική λοιπόν υπόθεση είναι πως τα έργα ανοιχτού κώδικα που χρησιμοποιήθηκαν είναι αντιπροσωπευτικά των έργων που υπάρχουν στην αγορά και αυτό γιατί προέρχονται από μία αξιόπιστη πηγή και έχουν χρησιμοποιηθεί από πολλούς χρήστες και προγραμματιστές ανά την υφήλιο. Για να υπάρχει μάλιστα ποικιλία προτιμήθηκαν έργα διαφορετικά σε μέγεθος αλλά και σε φύση του προβλήματος το οποίο επιλύουν. Θεωρήθηκε πως μία αρχική βάση 10 έργων είναι αρκετή καθώς η ανάλυση δεν γίνεται σε επίπεδο έργου αλλά σε επίπεδο κλάσεων ο αριθμός των οποίον είναι σημαντικά μεγαλύτερος. Πρακτικά λοιπόν το σύνολο ορισμού της μελέτης περιέχει μερικές χιλιάδες δεδομένα.

Μία ακόμα σημαντική υπόθεση είναι πως οι γλώσσες προγραμματισμού αντιμετωπίζουν κοινά προβλήματα. Τα έργα τα οποία αναλύονται στην μελέτη αποτελούνται τουλάχιστον κατά 97% από Java κώδικα. Αφενός λοιπόν υποθέτουμε πως το υπόλοιπο 3% περίπου δεν έχει μεγάλη

συμβολή στο τεχνικό χρέος, αφετέρου γίνεται σιωπηρά η υπόθεση πως παρόμοια συμπεριφορά έχουν και τα έργα που είναι γραμμένα σε άλλες γλώσσες προγραμματισμού.

Τα εργαλεία τα οποία αποφασίστηκε να χρησιμοποιηθούν στην μελέτη είναι αυτά τα οποία ορίζουν ξεκάθαρα το μέγεθος του τεχνικού χρέους. Τα εργαλεία θέλαμε να υποστηρίζουν διαφορετικές γλώσσες προγραμματισμού για ανάλυση ακόμα και αν τα έργα που προτιμήθηκαν στα πλαίσια αυτής της μελέτης ήταν μόνο γραμμένα σε Java. Έτσι διασφαλίζεται κατά ένα ποσοστό η κοινή αντιμετώπιση και των υπολοίπων γλωσσών προγραμματισμού αναφορικά με τον υπολογισμό του τεχνικού χρέους από τα ίδια εργαλεία. Από αυτά τώρα τα εργαλεία που αποτελούν το σύνολο που περιγράφηκε παραπάνω, χρησιμοποιήθηκαν όσα είχαν ελεύθερη άδεια αλλά και όσα μας παραχώρησαν άδεια για ακαδημαϊκή χρήση.

Συνεισφορά

Μέχρι αυτή την στιγμή αντίστοιχο benchmark σχετικό με την μέτρηση του Τεχνικού Χρέους που να βασίζεται στην τομή εργαλείων ανάλυσης κώδικα δεν έχει γίνει γνωστό στο ευρύ κοινό. Πρόκειται λοιπόν για μία προσπάθεια η οποία είναι πρότυπη και η οποία μπορεί να αποδειχθεί ιδιαίτερα χρήσιμη τόσο για τους ερευνητές όσο και για τις επιχειρήσεις.

Τα περισσότερα εργαλεία που είναι διαθέσιμα στην αγορά απαιτούν την εξασφάλιση κάποιας άδειας. Αποτελούν συνεπώς ένα έξοδο για την επιχείρηση τα οποία όπως είναι φυσικό στοχεύει να περιορίσει στο δυνατό ελάχιστο. Άρα μία επιχείρηση θα διαθέσει χρήματα για την απόκτηση ενός και μόνο εργαλείου εάν δεν προτιμήσει την λύση του ανοιχτού κώδικα. Εκεί λοιπόν πρέπει να παρθεί η απόφαση για το ποιο εργαλείο είναι το πλέον κατάλληλο κατά περίπτωση. Εάν κάποιο εργαλείο εντοπίζει περισσότερες ευπάθειες από τα υπόλοιπα ίσως θα πρέπει να προτιμηθεί. Αντίθετα αν δύο εργαλεία εντοπίζουν τα ίδια προβλήματα δεν υπάρχει λόγος να αποκτηθούν και τα δύο καθώς η ανάγκη καλύπτεται από το ένα. Υπάρχει σαφώς και η περίπτωση που η βέλτιστη λύση είναι ο συνδυασμός εργαλείων, όταν αυτά δεν εστιάζουν στα ίδια προβλήματα αλλά συνδυαστικά μπορούν να εντοπίσουν το σύνολο των ευπαθειών.

Από την μεριά των ερευνητών το benchmark μπορεί να αποτελέσει σημείο αναφοράς για την ερευνά τους. Στην περίπτωση που μελετάται μία νέα προσέγγιση στην μέτρηση του τεχνικού χρέους ή η δημιουργία ενός νέου εργαλείου, τότε τα αποτελέσματά του μπορούν να συγκριθούν με τα ήδη υπάρχοντα στην βάση. Έτσι μπορεί κάθε εργαλείο να γνωρίζει κατά πόσο συμφωνεί ή αποκλίνει από τα αυτά που χρησιμοποιήθηκαν στην παρούσα μελέτη.

Από την στιγμή λοιπόν που θα προκύψει το benchmark το κέρδος δύναται να είναι διπλό για διάφορους επιστημονικούς τομείς. Το δε πείραμα και η μελέτη των δεδομένων μπορεί να επαναληφθεί ακολουθώντας την ίδια μεθοδολογία και να προσαρμοστεί στις ανάγκες του κάθε ερευνητή. Σε αυτή την μελέτη για παράδειγμα χρησιμοποιήθηκαν τρία εργαλεία. Κατά τον ίδιο τρόπο μπορούν να προστεθούν περισσότερα ή να αλλάξουν κάποια από αυτά που συμμετέχουν ήδη.

Βασική Ορολογία

Τεχνικό Χρέος

Πρόκειται για μία μεταφορά η οποία αναφέρθηκε από τον Cunningham το 1992 πρώτη φορά. Ήθελε με αυτόν τον τρόπο να δείξει πως πολλές φορές προκειμένου να ανταποκριθεί η ομάδα ανάπτυξης στους χρονικούς περιορισμούς που βάζει ο πελάτης παραδίδει έναν κώδικα λιγότερο ποιοτικό από αυτόν που θα έπρεπε. Αυτή η έκπτωση ποιότητας λοιπόν είναι ένα χρέος το οποίο όσο συσσωρεύεται τόσο πιο δύσκολη γίνεται η αποπληρωμή του. [1]

Code smells

Πρόκειται για μία ένδειξη στον κώδικα ότι κάτι δεν είναι όπως θα έπρεπε. Μπορεί να μην προκαλεί σφάλμα την δεδομένη στιγμή στην εκτέλεση του προγράμματος, είναι όμως μία ένδειξη πως το σημείο αυτό είναι δύσκολο στην συντήρηση και πιθανότατα θα προκαλέσει προβλήματα λειτουργικά και λογικά στο μέλλον.

Design Patterns - Πρότυπα σχεδίασης.

Τα πρότυπα σχεδίασης είναι έτοιμες τυποποιημένες και δοκιμασμένες λύσεις σε συνηθισμένα προβλήματα που μπορεί να αντιμετωπίσει ο προγραμματιστής κατά την συγγραφή κώδικα. [2]

Grime Build Up - Βρώμικος Κώδικας

Κατά την ανάπτυξη ενός λογισμικού οι προγραμματιστές πρέπει να το αναπτύσσουν λαμβάνοντας υπόψη τα πρότυπα σχεδίασης που υπάρχουν. Ωστόσο ακόμα και αν στην αρχική σχεδίαση και υλοποίηση ενός έργου εφαρμόστηκαν αυτά τα πρότυπα, οι συνεχείς προσθήκες και μεταβολές μπορεί να οδηγήσουν σε παρέκκλιση από αυτά. Έτσι λοιπόν όχι μόνο γράφεται νέος κώδικας ο οποίος δεν ακολουθεί σχεδιαστικά πρότυπα αλλά κινδυνεύουν να χαλάσουν και αυτά που χρησιμοποιήθηκαν στο προηγούμενο στάδιο της ανάπτυξης.[3]

Modularity Violations - Παραβιάσεις τυποποίησης

Κάθε μεγάλο έργο πρέπει να διασπάται σε μικρότερα τμήματα προκειμένου κάθε ένα να έχει ξεχωριστή λειτουργικότητα και να μπορεί να αναπτύσσεται ανεξάρτητα των υπολοίπων. Αυτό κάνει κάθε τμήμα πιο εύκολα συντηρήσιμο και επεκτάσιμο. Ωστόσο πολλές είναι οι φορές που υπάρχουν ισχυρές αλληλεξαρτήσεις μεταξύ των τμημάτων. Αυτό έχει ως αποτέλεσμα μία αλλαγή σε ένα από αυτά να επηρεάζει σε μεγάλο βαθμό και τα υπόλοιπα.[4]

Διάρθρωση Μελέτης

Η μελέτη αυτή χωρίζεται σε δύο βασικές ενότητες την βιβλιογραφική επισκόπηση και την πρακτική εφαρμογή.

Στην βιβλιογραφική επισκόπηση αρχικά γίνεται η ανάλυση ορισμένων βασικών εννοιών που έχουν ιδιαίτερη σημασία για τον χώρο που εξετάζουμε. Γίνεται αναφορά σε όρους όπως η ποιότητα του κώδικα, η συντηρισιμότητά του και το τεχνικό χρέος. Έπειτα ακολουθεί μία έρευνα για τις μεθόδους καταγραφής και μέτρησης του τεχνικού χρέους και το πως αυτές μπορούν να αποδοθούν ως ένα benchmark το οποίο θα είναι σημείο αναφοράς και σύγκρισης της ποιότητας διαφόρων εφαρμογών.

Το δεύτερο κομμάτι είναι η παρουσίαση της έρευνας που πραγματοποιήθηκε. Αρχικά αναλύεται ο τρόπος με τον οποίο επιλέχθηκα τα έργα προς ανάλυση και έπειτα γίνεται μία σύντομη αναφορά σε αυτά τα έργα. Στην συνέχεια ακολουθεί μία παρουσίαση των τριών εργαλείων που χρησιμοποιήθηκαν για την ανάλυση του κώδικα στα πλαίσια της μελέτης αυτής. Έπειτα υπάρχει μία πιο λεπτομερής αναφορά στην φύση των δεδομένων που συλλέχθηκαν και αποτέλεσαν την βάση της μελέτης, πως αυτά προσφέρονται από το κάθε εργαλείο και τι διαδικασία τροποποίησης ακολουθήθηκε προκειμένου να μπορούν να είναι επεξεργάσιμα. Στο τέλος ακολουθεί μία σύντομη περιγραφή της στατιστικής ανάλυσης η οποία ακολούθησε την αρχική επεξεργασία των δεδομένων.

Βιβλιογραφική Επισκόπηση

Ποιότητα Κώδικα

Όπως έχει ειπωθεί από τον Martin Fowler μόλις το 2008 "*Any fool can write code that a computer can understand. Good programmers write code that humans can understand*." (Κάθε ανόητος μπορεί να γράψει κώδικα που είναι κατανοητός από τον υπολογιστή. Ο καλός προγραμματιστής μπορεί να γράψει κώδικα που είναι κατανοητός από τους ανθρώπους). Παρά το υπερβολικό στοιχείο της κατάταξης όλων των προγραμματιστών στην κατηγορία του "ανόητου" η βασική ιδέα εδώ είναι σωστή. Ο κώδικας που παραδίδει κάποιος προγραμματιστής στην παραγωγή πρέπει να είναι τόσο καθαρός ώστε να μπορεί να γίνει κατανοητός και συντηρήσιμος από οποιοδήποτε άλλο μέλος της ομάδας ανάπτυξης. Το πόσο καλός είναι αυτός ο κώδικας αντικατοπτρίζεται στην λεγόμενη ποιότητά του. Είναι ένα μέγεθος το οποίο μεταβάλλεται μέσα στον χρόνο και είναι σημαντικό για όλα τα αξιόλογα έργα να διατηρείται σε υψηλά επίπεδα.

Η διατήρηση της ποιότητας διαχρονικά θεωρείται ακριβή και αντιπαραγωγική διαδικασία από τους διοικούντες, παρόλο που στο παρελθόν έχει στοιχίσει μεγάλα οικονομικά ποσά αλλά και ανθρώπινες ζωές. Πλέον όμως όλο και περισσότερη σημασία δίνεται και αυτό είναι φανερό από την κατάρτιση ήδη δύο προτύπων ποιότητας. Το πρότυπο Sqale είναι μία γενική μέθοδος ανεξάρτητη από την γλώσσα προγραμματισμού και το εργαλείο κώδικα που χρησιμοποιείται. Βασίζεται στην καταμέτρηση του τεχνικού και του σχεδιαστικού χρέους. Το δεύτερο πρότυπο είναι το SQuARE το οποίο βρίσκεται ακόμα υπό ανάπτυξη και θα αποτελέσει την εξέλιξη του ISO 9126. Την ίδια στιγμή για συγκεκριμένους τομείς όπως οι αυτοκινητοβιομηχανία υπάρχουν πιο εξειδικευμένα πρότυπα τα οποία πρέπει να ακολουθούνται[5].

Η δομική ποιότητα (structural quality) αναφέρεται στον βαθμό στον οποίο η αρχιτεκτονική της εφαρμογής και ο κώδικάς της αποφεύγουν γνωστά και συχνά λάθη που εντοπίζονται στο λογισμικό. Ταυτόχρονα θα πρέπει να τηρούνται οι αρχές της τεχνολογίας λογισμικού και οι τεχνικές σωστής υλοποίησης. Μπορεί να μετρηθεί σε επίπεδο εφαρμογής ή κώδικα ανάλογα με το επίπεδο λεπτομέρειας που απαιτείται. Έχει μελετηθεί και παρατηρηθεί πως όσο

χαμηλότερη είναι η δομική ποιότητα σε ένα έργο, τόσο μεγαλύτερο είναι το ρίσκο που περιέχει. Μπορεί εν δυνάμει να δημιουργήσει σοβαρά προβλήματα στην επιχείρηση που χρησιμοποιεί το λογισμικό και κατ' επέκταση στην φήμη της εταιρείας που το παράγει. [6]

Συντηρισημότητα Έργου (Maintainability)

Τα έργα λογισμικού στην πλειοψηφία τους έχουν κάποια χαρακτηριστικά τα οποία δεν είναι κοινά με τα υπόλοιπα προϊόντα στην αγορά. Όταν ένα πρόγραμμα παραδίδεται από τον δημιουργό στον πελάτη η ανάμειξη του δημιουργού δεν ολοκληρώνεται εκεί. Το πρόγραμμα είναι ένας "ζωντανός οργανισμός" ο οποίος χρήζει διορθώσεων, τροποποιήσεων και επεκτάσεων. Όλα αυτά μάλιστα πρέπει να γίνουν ενώ η προηγούμενη έκδοση του προϊόντος παραμένει πλήρως λειτουργική μέχρι την στιγμή της αναβάθμισής της. Από την άλλη μεριά η νέα έκδοση δεν πρέπει να στερείται κάποια προηγούμενη λειτουργικότητα. Είναι λοιπόν πολύ σημαντικό αυτή η αλληλεπίδραση που ακολουθεί την παράδοση να γίνεται με τον πλέον ομαλό και εύκολο τρόπο.

Δείκτης αυτής της ευκολίας είναι η συντηρισιμότητά του έργου, ή αλλιώς ο δείκτης maintainability ο οποίος είναι από τους βασικούς δείκτες που μετράει κάθε εργαλείο ανάλυσης ποιότητας κώδικα. Υπολογίζεται πως το 40% ως και το 70%, για κάποιους όπως η CAST ίσως και ακόμα μεγαλύτερο ποσοστό [7], του κόστους ενός έργου συνδέεται με την συντηρισμότητά του. Αναδεικνύοντας έτσι το πόσο σημαντικό και κερδοφόρο μπορεί να είναι εάν ένα έργο σχεδιαστεί και υλοποιηθεί με τις σωστές προδιαγραφές.

Η σημασία το δείκτη φαίνεται άλλωστε και από την ύπαρξη ειδικού δείκτη ποιότητας του ISO 25010. Ο δείκτης αυτός χρησιμοποιώντας το μοντέλο μέτρησης ποιότητας SIG (Software Improvement Group) εξετάζει την συντηρισιμότητά του έργου και έπειτα την σταθμίζει με βάση ένα προηγούμενο benchmark που έχει προκύψει από αντίστοιχες μετρήσεις σε άλλα έργα.

Η συντηρισιμότητά του έργου δεν είναι ένα στατικό μέγεθος. Αναπτύσσεται και έχει τις διακυμάνσεις της όπως πραγματοποιούνται αλλαγές και προσθήκες από μία γενιά σε άλλη.

Είναι λοιπόν ενδιαφέρον να μελετάται η πορεία της ανάμεσα στις γενιές. Προκύπτει άλλωστε πως ο όγκος του κώδικα ο οποίος τείνει να αυξάνει από γενιά σε γενιά σχετίζεται άμεσα με το ποσοστό διατηρησιμότητας (maintainability rate) του ίδιου του έργου. Το μοντέλο SIG μάλιστα σταθμίζει τα βάρη στους διάφορους παράγοντες που λαμβάνει υπόψη του κατά τις μετρήσεις, σε άμεση συσχέτιση με τον όγκο του κώδικα. Ένας ακόμα λόγος για την παρακολούθηση της εξέλιξης της συντηρησιμότητας είναι ότι δεν παρουσιάζει την ίδια συμπεριφορά στις αλλαγές. Το αντίκτυπο που έχει μία αλλαγή η οποία χειροτερεύει την ποιότητα του κώδικα είναι πολύ μεγαλύτερο ποσοστιαία, από ότι μία αντίστοιχη αλλαγή η οποία βελτιώνει την ποιότητα. [8].

Τεχνικό Χρέος

Ο όρος του Τεχνικού Χρέους εισάγεται από τον Ward Cunningham το 1992 στο συνέδριο OOPSLA (Object-Oriented Programming, Systems, Languages & Applications). Εκεί χρησιμοποιεί το χρέος ως μεταφορά για την παράδοση του κώδικα στον πελάτη. Για λόγους χρόνου λοιπόν μπορεί να παραδοθεί στον πελάτη κώδικας ο οποίος είναι λειτουργικός αλλά δεν έχει την απαραίτητη ποιότητα. Η παραδοχή αυτή ταυτίζεται με τον όρο του χρέους. Εάν αυτό το χρέος είναι μικρό και αποπληρωθεί γρήγορα η ζημιά είναι από μικρή ως αμελητέα καθώς το κέρδος που προκύπτει από την έγκαιρη παράδοση του έργου είναι μεγαλύτερο. Αυτό βέβαια σημαίνει πως μετά την παράδοση θα αφιερωθεί χρόνος ώστε να ξαναγραφτούν κομμάτια κώδικα με τον σωστό πλέον τρόπο. Εάν όμως αυτό δεν συμβεί τότε το έργο κινδυνεύει καθώς γίνεται συνεχώς και πιο δύσκολη η συντήρησή και η επέκτασή του.

Το κόστος που προκύπτει από το τεχνικό χρέος και πρέπει να αποπληρωθεί χωρίζεται σε δύο βασικές κατηγορίες. Η πρώτη είναι η αποπληρωμή του ίδιου του χρέους. Ο χρόνος δηλαδή που απαιτείται για να διορθωθεί ο προβληματικός κώδικας αυτός καθ’ αυτός και αυτό αναφέρεται ως κεφάλαιο του χρέους (principal). Η δεύτερη κατηγορία είναι η αποπληρωμή των τόκων. Εάν ο αρχικός προβληματικός κώδικας αποτελεί βάση πάνω στην οποία αναπτύχθηκαν και άλλα κομμάτια του έργου, τότε πρέπει να αφιερωθεί χρόνος ώστε να διορθωθούν και αυτά.[9] Ο τόκος με την σειρά του αποτελείται και αυτός από δύο επιμέρους ποσά, το ποσό του τόκου που εκφράζει την επιπλέον προσπάθεια που θα χρειαστεί (interest

amount) και την πιθανότητα αυτό να προκαλέσει αλυσιδωτά προβλήματα στον υπόλοιπο κώδικα (interest probability)[10].

Ο παραλληλισμός αυτός ήταν αναγκαίος καθώς αποτελεί την γέφυρα μεταξύ των μηχανικών λογισμικού και των διευθυντικών στελεχών και οικονομολόγων στους οποίους οι οικονομικοί όροι είναι πιο οικείοι και έτσι μπορούν να αντιληφθούν την σημασία του. Άλλωστε το αποτέλεσμα του τεχνικού χρέους σε ένα έργο δεν είναι εμφανές με την πρώτη ματιά καθώς είναι ένα ποιοτικό μέγεθος το οποίο μπορεί να προκαλέσει προβλήματα μακροπρόθεσμα ακόμα και αν την δεδομένη στιγμή φαίνεται πως όλα λειτουργούν όπως πρέπει. Η βιαστική συγγραφή του κώδικα μπορεί να αποφέρει βραχυπρόθεσμο κέρδος. Μακροχρόνια όμως επιβραδύνει την παραγωγική διαδικασία.[11] Κομμάτια κώδικα που έχουν γραφτεί προκειμένου να παραδοθούν εγκαίρως πιθανότατα να χρειαστεί να ξαναγραφτούν από την αρχή ώστε τελικά να καλύπτουν όλες τις απαιτούμενες λειτουργίες, να είναι ασφαλή από σφάλματα κατά την εκτέλεση, συντηρίσμα και επεκτάσιμα. Αρκετές μάλιστα είναι οι φορές που το να γραφτεί ένα μέρος του κώδικα από την αρχή είναι πιο αποτελεσματικό και γρήγορο από το να γίνει προσπάθεια διόρθωσης υφιστάμενου κώδικα.

Κατά την ανάπτυξη λογισμικού ο στόχος του μηδενικού χρέους δεν είναι ρεαλιστικός.[11] Και αυτό καθώς εξαρτάται από πολλούς παράγοντες. Ακόμα και γίνει η παραδοχή πως οι προγραμματιστές έχουν την απαραίτητη γνώση και εμπειρία για να γράφουν ποιοτικό κώδικα, οι προδιαγραφές ενός έργου είναι δυναμικές και μία μεταγενέστερη προδιαγραφή ή η μεταβολή κάποιας ήδη υπάρχουσας μπορεί να δημιουργήσει την ανάγκη μεταβολής του τρόπου αντιμετώπισης του προβλήματος σε επίπεδο κώδικα. Για παράδειγμα μπορεί η εισαγωγή μιας νέας υποκατηγορίας δεδομένων να οδηγήσει στην ανάγκη κληρονομικότητας.

Είναι σημαντικό λοιπόν να καθοριστεί το μέγεθος Τεχνικού Χρέους το οποίο είναι αποδεκτό σε κάθε περίπτωση και πιο ποσοστό από αυτό αξίζει να αποπληρωθεί. Τα χαρακτηριστικά τα οποία καθορίζουν τα παραπάνω καθώς και την προτεραιότητα στο ποιο μέρος του τεχνικού χρέους πρέπει να διευθετηθεί πρώτα, διαφέρουν κατά περίπτωση. Στην περίπτωση για παράδειγμα της Agile μεθοδολογίας ο αυτοματισμός των ελέγχων και η μείωση της πολυπλοκότητας είναι σημαντικά ζητήματα που είναι στην κορυφή της λίστας. Ρόλο στην

επιλογή παίζει η κρισιμότητα του συνόλου του προγράμματος αλλά και κάθε επιμέρους μονάδας. Όσο πιο κρίσιμη είναι η λειτουργία τόσο μεγαλύτερη προτεραιότητα έχει η επιδιόρθωσή της. Από την άλλη μεριά οι εφαρμογές του ιστού δίνουν μεγάλη σημασία σε χαρακτηριστικά που αφορούν την ασφάλεια. Το σημείο του κύκλου ζωής του έργου αλλά και η τωρινή κατάστασή του επίσης αποτελούν κριτήρια. Ενώ τέλος οι ίδιες οι απαιτήσεις του έργου αλλά και η προγραμματισμένη επαναχρησιμοποίηση συγκεκριμένου μέρους του μπορούν να θέσουν ορισμένα κομμάτια πρώτα στην λίστα απαλοιφής του τεχνικού χρέους[11].

Κατηγοριοποίηση Τεχνικού Χρέους

Υπάρχουν διαφορετικοί τρόποι κατηγοριοποίησης του τεχνικού χρέους οι οποίοι βασίζονται στις διαφορετικές σκοπιές από τις οποίες μπορεί να το εξετάσει κανείς. Η κατηγοριοποίηση μπορεί να αφορά το τι επηρεάζει το χρέος, από ποιον προήλθε ή την αιτία για την οποία δημιουργήθηκε.

Αρχικά το τεχνικό χρέος μπορεί να προκύψει ηθελημένα ή μη. Χωρίζεται λοιπόν σε ακούσιο και εκούσιο. Υπάρχουν περιπτώσεις όπου οι ίδιοι οι προγραμματιστές δεν έχουν την απαραίτητη γνώση και οι τεχνικές που χρησιμοποιούν δεν είναι οι ενδεδειγμένες. Αυτό είναι το ακούσιο χρέος. Υπάρχουν όμως και περιπτώσεις όπου το χρέος δημιουργείται εκούσια, εν γνώση των προγραμματιστών και είναι η θυσία που πραγματοποιείται ελέω χρονικών περιορισμών[9].

Μία άλλη ταξινόμηση αναφέρεται στο βιβλίο “Advances in Computers” στο κεφάλαιο “Measuring and Monitoring Technical Debt”[10]. Εκεί οι συγγραφείς παρουσιάζουν τις εξής τέσσερις κατηγορίες.

- Χρέος ελέγχου (testing debt): Όταν ένα κομμάτι κώδικα βγαίνει στην παραγωγή χωρίς να γίνει επιθεώρησή του και χωρίς να υπάρχουν δομές όπως JUnit tests για παράδειγμα που να εγγυώνται την σωστή λειτουργία του, τότε δημιουργείται χρέος.

- Χρέος ελαττωμάτων (defect debt): Υπάρχουν αρκετές περιπτώσεις όπου μπορεί να εντοπιστεί ένα ελάττωμα στην λειτουργία του προγράμματος και παρόλα αυτά να αποφασιστεί να μην διορθωθεί την δεδομένη στιγμή.

- Χρέος τεκμηρίωσης (documentation debt): Κάθε κομμάτι κώδικα πρέπει να τεκμηριώνεται επαρκώς ώστε να μπορεί να το επεξεργαστεί και να το τροποποιήσει οποιοσδήποτε προγραμματιστής και όχι μόνο ο αρχικός δημιουργός του. Αυτό το κομμάτι του χρέους είναι αρκετά σύνηθες στα μεγάλα έργα λογισμικού. Η τεκμηρίωση συνήθως περιορίζεται μόνο στα ιδιαίτερα πολύπλοκα μέρη του έργου. Πολλές φορές ακόμα και αυτή είναι ανεπαρκής.

- Σχεδιαστικό χρέος (design debt): Αυτό το κομμάτι του χρέους αφορά παραβιάσεις που γίνονται στην χρήση των ενδεδειγμένων προγραμματιστικών τεχνικών αλλά και του αρχιτεκτονικού μοντέλου του έργου.

Στο άρθρο “Identification and management of technical debt: A systematic mapping study”[12] οι συγγραφείς προχωρούν σε μία πιο αναλυτική αναπαράσταση κατηγοριών. Αρχικά λοιπόν αναγνωρίζουν πως υπάρχει το χρέος ελέγχου, το χρέος τεκμηρίωσης καθώς και το χρέος ελαττωμάτων όπως και στον προηγούμενο άρθρο. Η κατηγορία του σχεδιαστικού χρέους αναλύεται περισσότερο ενώ τέλος εισάγονται οι έννοιες νέων κατηγοριών.

Το σχεδιαστικό χρέος αναλύεται στις εξής τρεις υποκατηγορίες:

- Σχεδιαστικό χρέος (design debt): Προβλήματα που αφορούν τις αποφάσεις κατά την σχεδίαση του λογισμικού.

- Χρέος κώδικα (code debt): Προβλήματα που αφορούν καθαρά το κομμάτι του κώδικα όπως για παράδειγμα η μεγάλη πολυπλοκότητα και δυσχεραίνουν την συντήρησή του.

- Αρχιτεκτονικό χρέος (architectural debt): Προβλήματα που αφορούν την δομή του έργου όπως οι παραβιάσεις τυποποίησης (modularity violations).

Πέρα από την αναγνώριση του χρέους ελέγχου εδώ εντοπίζεται και ένα δεύτερο σχετικό μέγεθος:

- Χρέος αυτοματοποιημένου ελέγχου (automated test debt): Περιλαμβάνει τον χρόνο που απαιτείται προκειμένου να υλοποιηθούν οι αυτόματοι έλεγχοι που θα διασφαλίζουν την σωστή λειτουργία του λογισμικού.

Οι νέες κατηγορίες χρέους που εισάγονται αναφέρονται παρακάτω:

- Χρέος προδιαγραφών (requirements debt): Υπάρχουν περιπτώσεις όπου παρόλο που η σωστή υλοποίηση είναι γνωστή και εφικτή αναγκαστικά παραβιάζονται κάποιες αρχές προκειμένου το τελικό προϊόν να ανταποκρίνεται στις ανάγκες του πελάτη. Συχνό θύμα τέτοιων αναγκών είναι οι προδιαγραφές ασφάλειας.

- Χρέος υποδομών (infrastructure debt): Όταν οι διαθέσιμες υποδομές και το υλικό δεν είναι το απαραίτητο τότε η διαδικασία ανάπτυξης μπορεί να επιβραδυνθεί σημαντικά.

- Χρέος ανθρώπων (person debt): Υπάρχουν φορές όπου το ανθρώπινο δυναμικό δεν επαρκεί για να υλοποιήσει όλο το έργο στον απαιτούμενο χρόνο. Ακόμη όμως και η πρόσληψη νέου προσωπικού δεν σημαίνει ότι θα επιταχύνει την διαδικασία. Αντιθέτως βραχυπρόθεσμα θα την επιβραδύνει καθώς αυτά τα νέα άτομα πρέπει να εκπαιδευτούν από τους ήδη πιο έμπειρους συναδέλφους τους.

- Χρέος διαδικασιών (process debt): Κάθε εταιρεία ακολουθεί συγκεκριμένες διαδικασίες. Αυτό δεν σημαίνει ότι όλες οι διαδικασίες είναι αποτελεσματικές για όλα τα έργα ενός οργανισμού.

- Χρέος οικοδόμησης έργου (build debt): Αφορά την διαδικασία που ακολουθείται για να “χτιστεί” το έργο. Η οποία μπορεί να είναι σημαντικά πιο αργή εάν ο κώδικας δεν είναι σωστά δομημένος και αν υπάρχουν μέσα σε αυτόν χαρακτηριστικά που δεν επιθυμεί ο πελάτης.

- Χρέος υπηρεσιών (service debt): Αυτό το είδος χρέους εμφανίζεται όταν γίνεται λανθασμένη επιλογή web services κατά την υλοποίηση με αποτέλεσμα να υπάρχουν ασυμφωνίες μεταξύ της πληροφορίας που παρέχεται από αυτές και αυτής που χρειάζεται τελικά το σύστημα για να λειτουργήσει.

- Χρέος χρηστικότητας (usability debt): Αποφάσεις χρηστικότητας οι οποίες δεν είναι οι κατάλληλες και τελικά θα χρειαστεί να επαναπροσαρμοστούν.

- Χρέος εκδόσεων (versioning debt): Προβλήματα μεταξύ των διαφορετικών εκδόσεων του κώδικα.

Τρόποι μέτρησης Τεχνικού Χρέους

Η αποτίμηση του Τεχνικού Χρέους είναι η προσπάθεια μετατροπής ενός ποιοτικού σε έναν ποσοτικό δείκτη. Κάθε τέτοια προσπάθεια εγκυμονεί τον κίνδυνο απόδοσης διαφορετικού βάρους σε κάθε ποιοτικό χαρακτηριστικό και άρα διαφοροποίησης στον τελικό ποσοτικό δείκτη. Αυτός ο τομέας προβληματισμού έχει τραβήξει το ενδιαφέρον αρκετών ερευνητών.

Μία γενική προσέγγιση αξιολόγησης του χρέους παρουσιάζεται στο βιβλίο “Advances in Computers”. Το πρώτο βήμα είναι η αναγνώριση των προβλημάτων σε ένα έργο και η κατάρτιση μιας λίστας με αυτά. Έπειτα γίνεται μία πρώτη πρόχειρη εκτίμηση για το κάθε αντικείμενο της λίστας. Σε κάθε πρόβλημα αντιστοιχίζεται ένα από τα τρία επίπεδα σημαντικότητας, υψηλό, μεσαίο και χαμηλό. Κλείνει έτσι ο πρώτος κύκλος αξιολόγησης. Ο δεύτερος κύκλος δίνει έμφαση στην λεπτομερή εξέταση του κάθε προβλήματος. Η αξιολόγησή του και η εκτίμηση για το χρέος που αυτό μπορεί να προκαλέσει βασίζεται κυρίως σε ιστορικά δεδομένα της επιχείρησης και αποτελείται από τον συνδυασμό των τριών επιμέρους μεγεθών του κεφαλαίου, του τόκου και της πιθανότητας του τόκου.

Το κεφάλαιο μπορεί να υπολογιστεί κατευθείαν από την στατιστική εξέταση των ιστορικών στοιχείων της επιχείρησης. Πιθανότατα η επιχείρηση έχει αντιμετωπίσει και στο παρελθόν αντίστοιχα προβλήματα άρα μπορεί να εκτιμήσει το μέγεθός τους καθώς και τον χρόνο που μπορεί να απαιτήσει η επιδιόρθωσή τους. Ακόμα όμως και να μην είναι διαθέσιμα τα ιστορικά

στοιχεία σε αυτό το επίπεδο αρκεί και η γνώμη ενός ειδικού για να καλύψει την εκτίμηση. Αντίστοιχα και η πιθανότητα του τόκου στηρίζεται επίσης σε ιστορικά δεδομένα και στην εμπειρία που έχει η επιχείρηση. Η διαδικασία εκτίμησης του ποσού του τόκου ωστόσο είναι λίγο πιο πολύπλοκη και αυτό γιατί θα πρέπει να εξεταστεί το αντίκτυπο που είχαν αρκετές από τις τελευταίες αλλαγές ώστε να προκύψει ένας μέσος όρος ο οποίος έπειτα πρέπει να σταθμιστεί ανάλογα με την σημαντικότητα του προβλήματος για να προκύψει το ποσό του τόκου[10].

Στο άρθρο “Comparing four approaches for TD Identification” [13]παρουσιάζονται οι εξής τέσσερις διαφορετικές προσεγγίσεις για το πως μπορούμε να αποτιμήσουμε το τεχνικό χρέος.

- Οσμές κώδικα (code smells)
- Αυτόματη στατική ανάλυση
- “Βρώμικος” (grime) κώδικας (χωρίς σχεδιαστικά πρότυπα - design patterns)
- Παραβιάσεις τυποποίησης (modularity violations)

Στην μελέτη του άρθρου πραγματοποιείται η σύγκριση των αποτελεσμάτων αυτών των τεσσάρων μεθόδων και εξετάζεται το ποσοστό στο οποίο συμπίπτουν. Το άρθρο καταλήγει πως υπάρχει μικρή συσχέτιση των αποτελεσμάτων καθώς κάθε μία από τις τεχνικές αναδεικνύει διαφορετικά προβλήματα στον κώδικα. Μάλιστα το ίδιο άρθρο αναφέρει πως το CAST χρησιμοποιεί την μέθοδο αυτόματης στατικής ανάλυσης ενώ το SonarQube χρησιμοποιεί στατικές μετρήσεις γεγονός που έχει ενδιαφέρον και για την μελέτη της συγκεκριμένης εργασίας καθώς στην ερμηνεία των αποτελεσμάτων θα πρέπει να ληφθεί αυτή η διαφοροποίηση υπόψη.

Η τιμή των δεικτών Τεχνικού Χρέους φαίνεται να αυξάνει αναλογικά με το μέγεθος του έργου που εξετάζεται καθώς αυξάνονται και οι οσμές κώδικα και οι παραβιάσεις τυποποίησης, ενώ αντίθετα η μέση τιμή των προβλημάτων ανά κλάση δεν εμφανίζει την ίδια μονότονη αύξηση. [13]

Επειδή όπως αναφέρθηκε το μηδενικό Τεχνικό Χρέος είναι μη ρεαλιστικός στόχος πρέπει να τεθεί ένα όριο το οποίο να προσδιορίζει το αποδεκτό ποσό του. Για να γίνει αυτό πρέπει πρώτα να σταθμιστούν οι διαφορετικοί παράγοντες του. Σύμφωνα με τον αλγόριθμο του Israel Gat η μεθοδολογία προσέγγισης αυτής της μέτρησης ξεκινάει με τα εξής τέσσερα βήματα

- Χρήση εργαλείων για την στατική και δυναμική ανάλυση του κώδικα
- Καθορισμός του κόστους διόρθωσης των προβλημάτων που εντοπίστηκαν βάσει στοιχείων της βιομηχανίας
- Υπολογισμός του συνολικού κόστους ως άθροισμα των επιμέρους
- Μετατροπή τους σε χρηματικές μονάδες

Αφού γίνουν οι μετρήσεις πρέπει να προσδιοριστούν τα όρια (thresholds) που χωρίζουν το αποδεκτό χρέος, από τις περιοχές που χρειάζονται βελτίωση και από τις επικίνδυνες περιοχές οι οποίες αποτελούν κίνδυνο για την συντήρηση. Τα όρια μπορούν να διαφέρουν για τα διάφορα υποσυστήματα ενός μεγάλου έργου. Καθώς επίσης διαφορετικά όρια μπορούν να οριστούν για τον παλιό και για τον νέο κώδικα γιατί είναι λιγότερο κοστοβόρο να προληφθεί το Τεχνικό Χρέος παρά να το διορθωθεί. Αν τα όρια που τεθούν για τον νέο κώδικα είναι μικρότερα από αυτά του παλιού, με την προϋπόθεση να τηρηθούν, τελικά μακροπρόθεσμα μειώνουν τον μέσο όρο τεχνικού χρέους σε όλο το έργο. Τα όποια όρια θα τεθούν πρέπει να είναι μεταβλητά και να προσαρμόζονται κάθε φορά ανάλογα με τις καινούριες μετρήσεις κώδικα[11].

Ο καθορισμός των ορίων είναι πολύ σημαντικός καθώς είναι αυτός που τελικά προσδιορίζει το αποτέλεσμα. Τα όρια αποφασίζονται ανάλογα με τις μετρήσεις που γίνονται πάνω σε συγκεκριμένες μονάδες λογισμικού κάθε φορά το σύνολο των οποίων ονομάζεται “benchmarking base”. Όσο μεγαλύτερη είναι αυτή η βάση των μετρήσεων τόσο μικρότερες είναι οι μελλοντικές αποκλίσεις που παρατηρούνται. Παρότι το πλήθος των συστημάτων είναι καθοριστικό για τις μετρήσεις, το μέγεθος των έργων που αποτελούν την βάση δεν φαίνεται να έχει την ίδια σημασία.

Στο άρθρο «Deriving Metric Thresholds from Benchmark Data» [14] παρουσιάζεται από τους συγγραφείς ένας τρόπος εξαγωγής των ορίων των μετρικών με τέτοιο τρόπο ώστε οι τιμές που προκύπτουν να έχουν νόημα εντός του πεδίου στο οποίο ορίζονται και να μην είναι αυθαίρετες. Η μέθοδος αυτή καθοδηγείται από τα δεδομένα και όχι από την γνώμη των ειδικών, σέβεται και ακολουθεί τα στατιστικά χαρακτηριστικά των μετρικών και μπορεί να επαναληφθεί για διαφορετικά σύνολα δεδομένων. Αφού λοιπόν ολοκληρωθεί η συλλογή των μετρικών και κανονικοποιηθούν σε επίπεδο οντότητας και έργου πρέπει να ληφθεί η απόφαση για το ποιο ποσοστό του έργου πρέπει να αντιπροσωπεύεται στο benchmark. Η κανονικοποίηση πραγματοποιείται σταθμίζοντας τα μεγέθη ανάλογα με τον αριθμό των γραμμών του έργου. Ανάλογα με το που θα τεθεί αυτό το όριο σχετικά με την αντιπροσώπευση του έργου, προκύπτει και η τιμή της μετρικής που αποτελεί το επιτρεπτό μέγεθος σε επίπεδο έργου.

Επειδή διαφορετικές benchmarking βάσεις έργων μπορεί να παράξουν διαφορετικά αποτελέσματα πρέπει η τελική βάση που θα οριστεί να έχει τέτοια χαρακτηριστικά ώστε να της επιτρέψουν να είναι γενική και επαναχρησιμοποιήσιμη. Υπάρχουν δύο κύριες βασικές προσεγγίσεις για την ποιοτική μέτρηση της βάσης. Η μέθοδος Squale και η μέθοδος Quamoco. Η μέθοδος Squale καθορίζει τις τιμές των ορίων όπως αυτές προκύπτουν από την γνωμοδότηση ειδικών του κλάδου. Είναι δηλαδή περισσότερο εμπειρικός δείκτης. Το Quamoco είναι μία διαφορετική προσέγγιση η οποία μπορεί να αναγνωρίσει τα συστήματα λογισμικού με διαφορετικά επίπεδα ποιότητας αλλά και την διαφοροποίηση ποιότητας μεταξύ των γενεών, ενώ τελικά παράγει δείκτες που έρχονται σε συμφωνία και με την γνώμη των ειδικών.

Τελικά για να προκύψει μία βάση η φύση της οποίας δεν καθορίζει τα αποτελέσματα πρέπει το ίδιο πλήθος έργων ακόμα και αν αυτά είναι διαφορετικά να καταλήγει στα ίδια μεγέθη. Το πλήθος όμως των έργων επηρεάζει την ταξινόμηση των προβλημάτων που εμφανίζονται και τα αποτελέσματα μέσης απόκλισης. Τέλος εάν για βάση χρησιμοποιηθούν μεγαλύτερα συστήματα τα αποτελέσματα του benchmark που θα προκύψουν μπορούν να χρησιμοποιηθούν με μεγαλύτερη ακρίβεια και για την αξιολόγηση μικρότερων έργων.[15]

Στο άρθρο «A Benchmarking-based Model for Technical Debt Calculation» [9] γίνεται μία προσπάθεια αυτόματης αποτίμησης του τεχνικού χρέους. Η αποτίμηση αυτή γίνεται στο επίπεδο ενός benchmark το οποίο έχει προκύψει από μία βάση έργων με γνωστό δεδομένο επίπεδο ποιότητας. Οι συγγραφείς αρχικά μελέτησαν και παρουσίασαν τρεις διαφορετικές μεθοδολογίες αποτίμησης του χρέους. Την μεθοδολογία του SIG όπως αναφέρθηκε και προηγούμενα σε αυτό το κεφάλαιο, και τις μεθοδολογίες της CAST και το SQUALE για στις οποίες θα γίνει αναφορά στην συνέχεια. Τα δεδομένα που χρησιμοποιήθηκαν προέκυψαν από προηγούμενες μελέτες και αξιόλογες πηγές δημοφιλείς σε μεγάλο πλήθος χρηστών και έπειτα κανονικοποιήθηκαν. Η μελέτη που βασίζεται λοιπόν στο benchmark με προηγούμενη γνώση φαίνεται να μπορεί να εκτιμήσει τα έξοδα αποκατάστασης του αρχικού ποσού του χρέους επιτυχώς.

Δείκτης SQALE

Ο δείκτης αυτός (Software Quality Assessment based on Lifecycle Expectations) εμφανίστηκε το 2010 και από τότε χρησιμοποιείται σε αρκετά εργαλεία ανάλυσης. Η μέθοδος αυτή χρησιμοποιείται από μία σειρά εργαλείων όπως το SonarQube και το SQuORE τα οποία χρησιμοποιήθηκαν και σε αυτή την μελέτη αλλά και άλλα γνωστά εργαλεία όπως NDepend, Mia-Quality και Security Reviewer Suite. [16]

Για να μπορέσει να αξιοποιηθεί ο δείκτης πρέπει πρώτα να οριστούν από την εταιρεία οι επιθυμητές τεχνικές που πρέπει να ακολουθούνται από τους προγραμματιστές κατά την διαδικασία της ανάπτυξης. Υπάρχουν ήδη έτοιμοι κατάλογοι με αυτές τις τεχνικές όπως είναι ο κατάλογος της Agile Alliance Debt Analysis Model (A2DAM). Κάθε εταιρεία ωστόσο μπορεί να δημιουργήσει την δική της λίστα με αυτά.

Έπειτα εξετάζεται κατά πόσο ο κώδικας συμφωνεί με τις τεχνικές που ορίστηκαν. Ανάλογα με το πόσο αποκλίνει ο κώδικας από τις προδιαγραφές η μεθοδολογία SQALE δίνει μία εκτίμηση του κεφαλαίου και του τόκου του χρέους. Στο πρώτο μοντέλο εκτίμησης από τα δύο της μεθοδολογίας υπολογίζεται ο χρόνος που χρειάζεται να διορθωθεί ο κώδικας που περιέχει το σφάλμα. Το σύνολο των χρόνων που προκύπτουν είναι το βασικό τεχνικό χρέος (remediation

cost). Το δεύτερο μοντέλο εκτίμησης είναι αυτό το οποίο υπολογίζει όλα τα παράπλευρα κόστη που μπορεί να προκύψουν εξαιτίας του αρχικού τεχνικού χρέους και χρειάζονται επιπλέον δουλειά για να επιλυθούν (Non remediation cost). Αυτό το δεύτερο μέγεθος αποτυπώνεται στον δείκτη “Business Impact” γιατί μπορεί να μην επηρεάζει απευθείας τον κώδικα αλλά και άλλες λειτουργίες της επιχείρησης.

Τελικά ο δείκτης που ενδιαφέρει την επιχείρηση είναι ο “SQALE Debt Ratio”. Πρόκειται για τον λόγο του τεχνικού χρέους προς τον αρχικό προϋπολογισμό του έργου. Όπως σε κάθε έργο τελικά αυτός θα δείξει εάν το τρέχον χρέος είναι κερδοφόρο ή ζημιογόνο. Ο δείκτης αυτός έχει μία πενταβάθμια κλίμακα η οποία ξεκινάει από το Α όπου είναι και μικρότερος ο λόγος και καταλήξει στο Ε όπου ο λόγος είναι μεγαλύτερο. Δεν αρκεί όμως μόνο να γνωρίζουμε το μέγεθος του λόγου. Προκειμένου να μειωθεί τελικά αυτός ο δείκτης θα πρέπει να μπορούμε να γνωρίζουμε πως αυτός συνδέεται και σε τι ποσοστό με τα ποιοτικά χαρακτηριστικά του έργου. Έτσι εντοπίζονται οι αιτίες του χρέους και είτε διορθώνονται είτε προλαμβάνονται.

Οι διορθώσεις που θα γίνουν στο έργο δεν έχουν αυστηρά προκαθορισμένη σειρά. Όπως αναφέρθηκε και προηγούμενα η προτεραιότητα εξαρτάται από πολλούς παράγοντες και την ίδια την φύση του έργου. Πρακτικά όμως η προτεραιότητα υπαγορεύεται από έναν αυστηρό παράγοντα και αυτός δεν είναι άλλος από τον χρόνο παράδοσης του έργου στον πελάτη. Εάν λοιπόν υπάρχει χρόνος αρκετός τότε πρέπει να φροντίσει η ομάδα ανάπτυξης να λύσει ότι δομικά προβλήματα υπάρχουν πριν προχωρήσει στην επιδιόρθωση προβλημάτων τεχνικού χρέους και αυτό γιατί είναι πιθανό λύνοντας τα δομικά προβλήματα να μειωθεί ταυτόχρονα και το τεχνικό χρέος. Στην περίπτωση όπου χρόνος υπάρχει αλλά παρόλα αυτά είναι περιορισμένος πρέπει να διορθωθούν τα προβλήματα που συμβάλλουν περισσότερο στον δείκτη “Business Impact” και επηρεάζουν σε μεγαλύτερο βαθμό την λειτουργία της επιχείρησης. Στην χειρότερη ωστόσο περίπτωση όπου ο χρόνος δεν υπάρχει τότε προτεραιότητα έχουν τα σημεία εκείνα που μπορούν να αποπληρώσουν περισσότερο ποσοστό από το αρχικό κεφάλαιο του χρέους. [17]

Η μετρήσεις που αφορούν το χρέος πρέπει να είναι διαρκείς. Πρέπει η παρακολούθηση της ποιότητας του κώδικα να είναι μέρος μιας αυτοματοποιημένης διαδικασίας όπως για

παράδειγμα αυτής του Jenkins [18]. Έτσι η εικόνα της πορεία του χρέους είναι ξεκάθαρη στην ομάδα ανάπτυξης και οι διορθωτικές ενέργειες μπορούν να είναι πιο άμεσες.

Στον δείκτη SQALE υπολογίζεται η συνολική προσπάθεια επιδιόρθωσης των σφαλμάτων κώδικα, προς τις 8 ανθρωποώρες που είναι διαθέσιμες ανά ημέρα. Η συνολική προσπάθεια μπορεί να αποτιμηθεί με δύο τρόπους. Στην πρώτη περίπτωση θεωρείται σταθερή η προσπάθεια για την επίλυση κάθε σφάλματος. Στην δεύτερη περίπτωση υπολογίζεται το γινόμενο του αριθμού των σφαλμάτων επί τον χρόνο που χρειάζεται το κάθε ένα. Στο τέλος γίνεται και η προσθήκη ενός μεγέθους χρόνου (offset) όπου εκεί υπολογίζονται όλοι οι παράγοντες που μπορεί να προκαλέσουν καθυστερήσεις στην διαδικασία, όπως η κατανόηση του σφάλματος ή η μετάβαση από το ένα σφάλμα σε κάποιο άλλο[9].

Μέθοδος CAST

Το εργαλείο της CAST εξετάζει την δομική ποιότητα του έργου. Βασίζεται στην αξιολόγηση των παρακάτω πέντε παραγόντων προκειμένου να αξιολογήσει τον κώδικα.

- Ασφάλεια (security): Εντοπίζει παραβιάσεις πρακτικών που κάνουν τον κώδικα ανθεκτικό σε απειλές ασφάλειας, όπως την εξασφάλιση πρόσβασης μόνο σε εξουσιοδοτημένα άτομα, την εμπιστευτικότητα ή την κλοπή δεδομένων.

- Ευρωστία (robustness): Εξετάζει πόσο ανθεκτικό είναι το πρόγραμμα σε σφάλματα. Πόσο γρήγορα και αποτελεσματικά μπορεί να επανέλθει σε περίπτωση δυσλειτουργίας, αλλά και το ποια είναι η πιθανότητα να καταστραφούν δεδομένα εξαιτίας κακών πρακτικών σε επίπεδο κώδικα.

- Αποδοτικότητα (efficiency): Μελετά την καλή απόδοση του συστήματος ως προς την επίτευξη του σκοπού του, αλλά και ως προς την αξιοποίηση των διαθέσιμων πόρων.

- Μεταβλητότητα (changeability): Εκφράζει την δυσκολία τροποποίησης του κώδικα καθώς και την δυνατότητα προσθήκης νέων χαρακτηριστικών.

- Μεταφερσιμότητα (transferability): Το μέγεθος αυτό αποτυπώνει το πόσο εύκολα μπορεί να μεταφερθεί η ουσία του κώδικα σε κάποιον προγραμματιστή, έτσι ώστε αυτός να το καταλάβει και να μπορεί να γίνει παραγωγικός με αυτό.

Η CAST διαθέτει ένα σύνολο από έργα τα οποία χρησιμοποιεί ως benchmarking base όπως αναφέρθηκε προηγούμενα. Αυτά σύμφωνα με τα δεδομένα του 2015 είναι 1850 έργα λογισμικού από 329 οργανισμούς που είναι γραμμένα σε διαφορετικές γλώσσες προγραμματισμού. Με βάση αυτά τα έργα υπολογίζεται το μέσο κόστος επιδιόρθωσης κάθε γραμμής κώδικα. Αυτό το κόστος είναι μεταβλητό στον χρόνο καθώς μεταβάλλονται και τα έργα τα οποία αποτελούν την βάση.

Στην CAST γίνεται η υπόθεση πως μόνο ένα μέρος των προβλημάτων θα επιδιορθωθούν σε ένα έργο. Έτσι λοιπόν το τεχνικό χρέος είναι αποτέλεσμα της στάθμισης αυτού του ποσοστού.

Τα προβλήματα αφού εντοπιστούν χωρίζονται σε τρεις βασικές κατηγορίες ανάλογα με την κρισιμότητά τους. Από αυτά θεωρείται ότι θα επιλυθεί το 50% όσων έχουν χαρακτηριστεί υψηλής κρισιμότητας, το 25% των προβλημάτων μεσαίας κρισιμότητας και μόλις το 10% όσων ανήκουν στο χαμηλότερο επίπεδο. Επίσης γίνεται η υπόθεση πως για κάθε πρόβλημα απαιτείται μία ανθρωποώρα ώστε να διορθωθεί. Τέλος καθορίζεται το κόστος ανά ανθρωποώρα με τα τελευταία δεδομένα να το θέτουν στα 75$ ανά ώρα. Έτσι λοιπόν προκύπτει η φόρμουλα υπολογισμού του τεχνικού χρέους ως εξής:

TD = ((50% * high_severity_violations) + (25% * moderate_severity_violations) + (10% * low_severity_violations)) * hour_to_fix_per_issue * cost_per_hour

Λοιπές μεθοδολογίες

Στο άρθρο “Tracking TD - An exploratory case study” [19] του 2011 γίνεται μία σύγκριση σχετικά με το κόστος που θα είχε η διόρθωση ενός προβλήματος εάν αυτό λυνόταν την στιγμή που εντοπίστηκε, σε σχέση με το πραγματικό κόστος που είχε τελικά την στιγμή που διορθώθηκε. Στο παράδειγμά τους εξετάζεται η περίπτωση όπου για ένα πρόβλημα, παρόλο που επισημάνθηκε νωρίς, πάρθηκε η στρατηγική απόφαση να καθυστερήσει η διόρθωσή του

προκειμένου να παραδοθεί στον πελάτη. Πέρα από την αρχική αποδοχή του προβλήματος ακολούθησε και μία ενημέρωση του συστήματος για βελτίωση της ποιότητάς του η οποία υλοποιήθηκε πάνω στον αρχικό προβληματικό κώδικα. Μετά την παράδοση του προγράμματος στον πελάτη εξαιτίας των αναγκών της αγοράς η επιδιόρθωση ήταν μονόδρομος. Η διαφορά ήταν πως τώρα εκτός από τον αρχικό κώδικα έπρεπε να διορθωθεί και ο κώδικας που είχε προστεθεί λόγω της ενημέρωσης.

Στο παραπάνω παράδειγμα προκύπτουν τρία σημεία κλειδιά στην διαδικασία. Η στιγμή που εντοπίστηκε το πρόβλημα, η στιγμή που έγινε η ενημέρωση και η στιγμή που λύθηκε το πρόβλημα. Για κάθε μία από αυτές τις στιγμές στην μεθοδολογία που παρουσιάζεται πρέπει να υπολογιστεί το ποσό του τόκου. Το ποσό αυτό είναι η διαφορά του κόστους που θα είχε η επίλυση του προβλήματος την στιγμή που εντοπίστηκε, με το πραγματικό κόστος που είχε αν διορθωνόταν σε οποιοδήποτε από τα τρία σημεία κλειδιά. Πέρα όμως από το κεφάλαιο πρέπει να υπολογιστεί και η πιθανότητα εμφάνισης και επιδιόρθωσης του τόκου. Η συνολική πιθανότητα είναι το γινόμενο των πιθανοτήτων εμφάνισης σε κάθε ένα από τα τρία σημεία κλειδιά.

Η μελέτη καταλήγει πως το τελικό κόστος που προκύπτει από την αναβολή της επιδιόρθωσης είναι περίπου τριπλάσιο από αυτό που θα υπήρχε εάν το πρόβλημα αντιμετωπιζόταν την στιγμή που εντοπίστηκε.

Εργαλεία Μέτρησης Τεχνικού Χρέους

SonarQube

Το SonarQube εμφανίστηκε στην αγορά το 2007 και αποτελεί ένα από τα πλέον διαδεδομένα εργαλεία αξιολόγησης κώδικα και διαρκούς ανάλυσης της ποιότητά του. Χαρακτηριστικό είναι ότι ακόμα και σε βιβλιογραφικό επίπεδο η χρήση του αναφέρεται σε πληθώρα εργασιών και άρθρων σχετικά με την ποιότητα του κώδικα και το τεχνικό χρέος. Είναι εργαλείο ανοιχτού κώδικα και ο κώδικάς του είναι διαθέσιμος στο GitHub [20]. Το έργο είναι γραμμένο σε Java και είναι ένα εργαλείο στατικής ανάλυσης το οποίο υποστηρίζει πολλές και διαφορετικές γλώσσες προγραμματισμού.[21]

Η μέτρηση του τεχνικού χρέους είναι μία βασική λειτουργία του εργαλείου. Για αυτό και προσφέρεται μία σειρά σχετικών μετρικών. Ιδιαίτερο όμως ενδιαφέρον παρουσιάζουν οι μετρικές του Technical Debt και του Technical Debt Ratio οι οποίες μάλιστα χωρίζονται σε αυτές που αφορούν τον συνολικό κώδικα αλλά και αυτές που αφορούν τον κώδικα ο οποίος προστέθηκε μετά την τελευταία ανάλυση.[22] Η δυνατότητα μέτρησης του χρέους είναι διαθέσιμη ως πρόσθετο (plug in) για το εργαλείο. Το χρέος μετράται και σε ανθρωπομέρες αλλά μετατρέπεται και σε δολάρια. Αφού πραγματοποιηθεί η στατική ανάλυση και εντοπιστούν τα προβλήματα και ο αριθμός αυτών, τότε υπολογίζεται το γινόμενο του πολλαπλασιασμού αυτού του αριθμού με τον μέσο όρο χρόνου που έχει εκτιμηθεί ότι ο κάθε τύπος προβλήματος χρειάζεται για να επιλυθεί. Το γινόμενο αυτό αποτελεί το τεχνικό χρέος. Το τεχνικό χρέος όμως υπολογίζεται και ως ποσοστό δίνοντας τον δείκτη Technical Debt Ratio. Ο δείκτης μας δείχνει την σχέση του τρέχοντος τεχνικού χρέους με τον χρόνο που θα απαιτούσαν τα προβληματικά κομμάτια κώδικα να ξαναγραφτούν από την αρχή. [23]

Για τον υπολογισμό του τεχνικού χρέους λαμβάνονται υπόψη πολλά διαφορετικά προβλήματα που μπορούν να εντοπιστούν στον κώδικα. Αυτά χωρίζονται σε 6 βασικές κατηγορίες, διπλότυπος κώδικας, παραβιάσεις αρχών, έλλειψη σχολίων, πολυπλοκότητα, έλλειψη ελέγχων και σχεδιαστικά σφάλματα. Η ανάλυση είναι πλήρως αυτοματοποιημένη. Μάλιστα φαίνεται πως η μέθοδος υπολογισμού που χρησιμοποιείται έχει αποτελέσματα με ικανοποιητικό βαθμό συσχέτισης και με την επαναχρησιμοποίηση του κώδικα και με το πόσο εύκολα κατανοητός γίνεται και τέλος με το πόσο αποτελεσματικός και λειτουργικός είναι. [24]

Η μέτρηση του χρέους από το SonarQube πραγματοποιείται με την μέθοδο SQALE. Υπάρχουν προκαθορισμένες πέντε βαθμίδες με τα αντίστοιχα κατώφλια. Ανάλογα λοιπόν με τον δείκτη του τεχνικού χρέους και την σύγκρισή του με τα κατώφλια αυτά, γίνεται και η εκτίμηση της τιμής “Maintainability rating”. Ταυτόχρονα δημιουργείται και η πυραμίδα του τεχνικού χρέους όπως επιτάσσει η μέθοδος SQALE. Σε αυτή την πυραμίδα γίνεται αρχικά η κατανομή του χρέους ανάλογα με το πόσο μακροπρόθεσμο αντίκτυπο αναμένεται να έχει αλλά και την κρισιμότητα του προβλήματος. Σύμφωνα λοιπόν με αυτή την πυραμίδα και την στρατηγική

αποπληρωμής που ακολουθεί ο οργανισμός μπορεί να γίνει η ανάθεση και η διαχείριση των δραστηριοτήτων που αφορούν το τεχνικό χρέος.[25]

SQuORE

Το εργαλείο SQuORE αναπτύχθηκε από την εταιρεία Squoring Technologies η οποία ιδρύθηκε στην Γαλλία το 2010. Η εταιρεία εξαγοράστηκε τον Ιούνιο του 2018 από την Γερμανική Vector Informatik. Πρόκειται για ένα εργαλείο το οποίο λειτουργεί για διαφορετικές πλατφόρμες και γλώσσες κώδικα. Για την χρήση του απαιτείται άδεια η οποία εκχωρήθηκε δωρεάν για τα πλαίσια της τρέχουσας έρευνας.

Στόχος του εργαλείου είναι να επιτρέπει σε όλους τους εμπλεκόμενους να έχουν μία κοινή εικόνα του έργου που αναπτύσσουν, έτσι ώστε να γίνει αντιληπτή η αναγκαιότητα της διατήρησης της ποιότητας σε υψηλά επίπεδα. Για να συμβεί αυτό απαιτείται η συλλογή πολλών μετρικών και η ομαδοποίησή τους σε μία ενιαία εικόνα για να καλυφθεί η ποιοτική αξιολόγηση του έργου από διαφορετικές σκοπιές.

Το SQuORE αποτελείται ουσιαστικά από τρία μικρότερα εργαλεία. Το πρώτο από αυτά είναι υπεύθυνο για την συλλογή δεδομένων. Σκοπός είναι να μπορεί το εργαλείο να δέχεται εύκολα και γρήγορα δεδομένα από διαφορετικές πηγές όπως το internet, αρχεία xml ή csv ακόμα και binary αρχεία. Αυτά τα δεδομένα τα αναλύει προκειμένου να καταλήξει στο ιεραρχικό δέντρο της δομής του έργου προς ανάλυση. Έπειτα αυτό το δέντρο περνάει μέσα από το δεύτερο εργαλείο το οποίο είναι υπεύθυνο για τους υπολογισμούς. Αφού έχουν πραγματοποιηθεί οι αρχικές μετρήσεις ο μηχανισμός προχωρά στις επιμέρους μετρήσεις κάθε κόμβου της ιεραρχίας. Αυτή η διαδικασία γίνεται με βάση τα μοντέλα ποιότητα τα οποία μας δείχνουν πως γίνεται η κατανομή των δεδομένων στο ιεραρχικό δέντρο. Τα μοντέλα αυτά δεν είναι πανάκια και πρέπει κάθε φορά να προσαρμόζονται στις ανάγκες του κάθε οργανισμού. Η μέτρηση όμως των δεδομένων δεν είναι παρά η ποσοτική έκφρασή τους χωρίς την ποιοτική τους αποτίμηση. Για να μετατραπούν οι μετρήσεις σε ποιοτικές απαιτούνται οι δείκτες οι οποίοι ουσιαστικά συγκρίνουν την μέτρηση με βάση κάποια κλίμακα η οποία ορίζει όρια στις μετρικές και καθορίζει την κατάταξή τους. Για κάθε μετρική λοιπόν ορίζονται τα όρια κλίμακας

της. Ανάλογα με το μέτρο της κάθε μετρικής γίνεται η αντιστοίχιση στο ανάλογο κλιμάκιο. Σε κάθε κλιμάκιο αντιστοιχίζεται ένα βάρος με βάση το οποίο θα σταθμιστεί το τεχνικό χρέος. Τρίτο και εξίσου σημαντικό εργαλείο είναι ο πίνακας ελέγχου το λεγόμενο “Dashboard”. Εδώ πλέον υπάρχει η οπτικοποίηση των αποτελεσμάτων με τέτοιο τρόπο ώστε να είναι ευανάγνωστα στον μέσο χρήστη και να μπορεί εύκολα και χωρίς περιττή πληροφορίας να πάρει μία ιδέα για το τι γίνεται γενικά στο έργο. Μάλιστα για αυτό τον λόγο το SQuORE έχει μία κλίμακα κατάταξης επτά επιπέδων προκειμένου να μπορεί εύκολα να γίνει αντιληπτό σε τι επίπεδο είναι κάθε δομική μονάδα την δεδομένη στιγμή και πόσο χρειάζεται να βελτιωθεί ακόμα.

Για το SQuORE τεχνικό χρέος είναι η απόσταση που έχει η τρέχουσα ποιότητα του έργου από την επιθυμητή. Αφού λοιπόν έχουν εντοπιστεί τα προβλήματα θεωρείται ο προγραμματιστής είναι σε θέση να γνωρίζει τι πρέπει να διορθωθεί ώστε να βελτιωθεί ο κώδικάς του. Υπάρχουν όμως περιπτώσεις όπου το ίδιο κομμάτι του κώδικα εμφανίζει περισσότερες από μία ευπάθειες. Για να σημειωθεί βελτίωση αυτού του κομματιού δεν αρκεί να εξαλειφθεί μια μόνο από αυτές. Πρέπει να διορθωθούν ως ένα ποσοστό όλες ή συγκεκριμένος συνδυασμός από αυτές. Το εργαλείο λοιπόν δίνει την δυνατότητα στον χρήστη να ορίσει τέτοιους συνδυαστικούς κανόνες προκειμένου η παρακολούθηση της ποιότητα και του τεχνικού χρέους να γίνεται ορθά και ρεαλιστικά.

Όλη αυτή η διαδικασία αποτελεί το πρώτο βήμα όπου ο χρήστης πρέπει να μπορεί να αναγνωρίσει το σημείο στο οποίο βρίσκεται το έργο του. Το επόμενο βήμα είναι να μπορεί να παρατηρεί την πορεία του στον χρόνο από εδώ και πέρα. Το SQuORE μπορεί να γίνει μέρος της αυτοματοποιημένης διαδικασίας του Jenkins ώστε σε ημερήσια βάση να παράγει νέες αναφορές για την ποιότητα του κώδικα αλλά και ταυτόχρονα να ελέγχει και να αναλύει ποια αρχεία είναι τα πιο επικίνδυνα και χρειάζονται και συντήρηση αλλά και Unit Tests προκειμένου να αυξήσουν την παραγωγικότητα και την απόδοση του έργου.[5]

CAST

Η εταιρεία ιδρύθηκε το 1990 με έδρα την Γαλλία και κυκλοφόρησε το πρώτο της προϊόν το 1995. Η πλατφόρμα ανάλυσης ποιότητας κώδικα το CAST AIP υποστηρίζει πάνω από 50 γλώσσες προγραμματισμού και 12 είδη βάσεων δεδομένων. Η χρήση του απαιτεί άδεια την οποία χορήγησε η εταιρεία στα πλαίσια αυτής της μελέτης.

Η πλατφόρμα αποτελείται από επιμέρους εργαλεία όπως:

- Health Dashboard: Μία σύνοψη της ασφάλειας, της παραγωγικότητας, της μεταφερσιμότητας, της ποιότητας και της μεταβλητότητας του έργου.
- Engineering Dashboard: Αξιολόγηση του κώδικα και εντοπισμός σφαλμάτων και σε υψηλό επίπεδο αλλά και σε μεγαλύτερη λεπτομέρεια.
- Security Dashboard: Εξειδίκευση στα σφάλματα που αφορούν την ασφάλεια
- CAST Enlighten: Οπτικοποίηση της δομής του έργου καθώς και τον εξαρτήσεων μεταξύ των τμημάτων του
- CAST Appmark: Μία benchmarking βάση πάνω στην οποία μπορούν να γίνουν συγκρίσεις του προϊόντος με άλλα προϊόντα στην αγορά.
- Imaging System: Οπτικοποίηση όλου του συστήματος

Ο παραδοσιακός τρόπος που γίνεται ο έλεγχος λειτουργικότητας των εφαρμογών εντοπίζει τα όποια προβλήματα υπάρχουν σε προχωρημένα στάδια της ζωής του προϊόντος. Αυτό έχει ως αποτέλεσμα το κόστος τους να είναι μεγάλο, η διαδικασία επίλυσής τους χρονοβόρα και το αντίκτυπό του σε άλλα σημεία του κώδικα αλλά και άλλες διαδικασίες της επιχείρησης σημαντικό. Όσες εταιρείες το έχουν συνειδητοποιήσει αυτό στρέφονται στις λεγόμενες τεχνικές “Shift Left”, μετατόπιση αριστερά αν μπορούμε να το μεταφράσουμε στα Ελληνικά. Αυτές οι τεχνικές προϋποθέτουν την παρακολούθηση της ποιότητας από την πρώτη μέρα ζωής του έργου λογισμικού και σε διαρκή βάση. Με αυτόν τον τρόπο τα προβλήματα εντοπίζονται αλλά και διορθώνονται νωρίς περιορίζοντας έτσι το κόστος τους. Το CAST λοιπόν ενθαρρύνει την στροφή προς αυτή την διαρκή παρακολούθηση.

Ταυτόχρονα το CAST φροντίζει και για την μείωση και άλλων πτυχών του τεχνικού χρέους, όπως για παράδειγμα αυτή της τεκμηρίωσης. Το εργαλείο παράγει αυτόματα διάφορα έγγραφα σχετικά με το έργο όπως σχεσιακά διαγράμματα ή έγγραφα απεικόνισης της αρχιτεκτονικής του. Αυτό έχει σαν αποτέλεσμα αφενός τα μέλη της ομάδας να μην χρειάζεται να δαπανήσουν χρόνο δημιουργώντας τα έγγραφα, αφετέρου όταν ένα νέο άτομο προστεθεί στην ομάδα να έχει στην διάθεσή του όλα τα απαραίτητα στοιχεία που χρειάζεται ώστε να κατανοήσει γρήγορα το έργο και να μπορέσει να ενταχθεί το συντομότερο δυνατόν στην παραγωγική διαδικασία.

Κύρια όμως λειτουργία του εργαλείου είναι να μπορεί να εντοπίζει τα πολύπλοκα και επικίνδυνα στοιχεία της εφαρμογής. Έτσι μπορεί να γίνει ο απαραίτητος προγραμματισμός και να αφιερωθεί η προσοχή που χρειάζεται αλλά και η στοχευμένη εκπαίδευση εάν αυτό κρίνεται αναγκαίο.[6]

Το εργαλείο αρχικά συγκεντρώνει ένα σύνολο προβλημάτων κώδικα που σχετίζονται με την ποιότητα, την ασφάλεια και την απόδοση. Εξετάζει το πόσο εύκολα μπορούν να τροποποιηθούν αυτά τα προβλήματα και το πόσο επιρρεπή είναι στο να μεταφερθούν από ένα σημείο στο άλλο. Στην συνέχεια αυτά τα προβλήματα σταθμίζονται σε τρεις κατηγορίες κρισιμότητας, χαμηλή, μεσαία και υψηλή.

Το CAST θεωρεί πως δεν θα διορθωθούν όλα τα προβλήματα αλλά ένα ποσοστό από αυτά μόνο. Έτσι λοιπόν σταθμίζει κάθε επίπεδο κρισιμότητας με το αντίστοιχο ποσοστό καταλήγοντας τελικά σε ένα άθροισμα συνολικών σφαλμάτων που πρέπει να διορθωθούν. Έπειτα ορίζει τον χρόνο ο οποίος απαιτείται για κάθε σφάλμα προκειμένου να διορθωθεί. Και αφού προκύψουν οι ανθρωποώρες γίνεται ο τελικός πολλαπλασιασμός με το κόστος ανά ανθρωποώρα ώστε να υπολογιστεί το μέγεθος του τεχνικού χρέους.[9]

TD-Tracker

Παρουσιάστηκε στην μελέτη [26] «Supporting Technical Debt Cataloging with TD-Tracker tool» το 2015. Το πρόβλημα που εντόπισαν οι δημιουργοί του και προσπάθησαν να επιλύσουν αφορά την κατάρτιση των καταλόγων με τα προβληματικά στοιχεία τα οποία αποτελούν

τελικά το τεχνικό χρέος. Θέλησαν λοιπόν να δημιουργήσουν ένα εργαλείο το οποίο μπορεί να συλλέγει προβληματικά σημεία κώδικα, ευπάθειες, αδύναμα σημεία ελέγχου και τεκμηρίωσης, ώστε να μπορεί να υπάρχει μία γενική άποψη για το χρέος καθ' όλη την διάρκεια ζωής του.

Τελικά ο κατάλογος που δημιουργείται έχει την δομή η οποία υπαγορεύεται από το Technical Debt Managing Framework και περιέχει τα παρακάτω στοιχεία

- Περιγραφή
- Τύπος τεχνικού χρέους
- Σημείο του έργου στο οποίο εντοπίζεται
- Η αναγκαιότητα ολοκλήρωσης της εργασίας επιδιόρθωσής του
- Εκτίμηση του κεφαλαίου του χρέους
- Εκτίμηση της πιθανότητας εμφάνισης τόκου
- Εκτίμηση του ποσού του τόκου

Η μεθοδολογία μέτρησης που προτείνεται από τους δημιουργούς χωρίζεται σε τρία στάδια. Το πρώτο είναι αυτό της αναγνώρισης. Εντοπίζονται τα προβλήματα και δημιουργείται μία λίστα με πιθανούς "υπόπτους" δημιουργίας τεχνικού χρέους. Το δεύτερο στάδιο είναι χειροκίνητη διαδικασία ή ημιαυτόματη και δέχεται ως είσοδο την λίστα που προέκυψε στο προηγούμενο στάδιο. Αυτή η λίστα φιλτράρεται και τελικά επιβεβαιώνεται από έναν ειδικό ο οποίος μπορεί να μεταβάλλει ή να διαγράψει στοιχεία από την λίστα. Τα στοιχεία πλέον που έχουν μείνει στον κατάλογο είναι και αυτά που αθροίζονται για να υπολογίσουν το τεχνικό χρέος. Τρίτο και τελευταίο στάδιο είναι η διαχείριση. Πρέπει να υπάρχει ένα έμπειρο άτομο το οποίο να επωμιστεί την ευθύνη να διανέμει τις εργασίες που αφορούν το τεχνικό χρέος με βάση κάποια συγκεκριμένη στρατηγική, και έπειτα να μπορεί να επιβλέπει την πρόοδο των εργασιών αυτών.

Το TD-Tracker λοιπόν είναι μία εφαρμογή ιστού η οποία υλοποιεί το παραπάνω πρωτόκολλο. Η συλλογή δεδομένων μπορεί να πραγματοποιηθεί με δύο τρόπους. Ο πρώτος είναι η απευθείας σύνδεση ενός εργαλείου εκτίμησης κώδικα, όπως το Sonar, με την βάση. Αυτή η διαδικασία προϋποθέτει αντιστοίχιση των πεδίων που προσφέρει το εξωτερικό εργαλείο με τα πεδία που υπάρχουν στην βάση της εφαρμογής. Η δεύτερη εναλλακτική είναι η διασύνδεση της εφαρμογής απευθείας με τα αποθετήρια του GitHub. Μόλις ολοκληρωθεί η συλλογή των δεδομένων, το εργαλείο υποστηρίζει και την χειροκίνητη και την ημιαυτόματη δημιουργία καταλόγου. Για να γίνει αυτό πρέπει να οριστεί και το ανάλογο άτομο που είναι υπεύθυνο για την μετέπειτα διαχείριση των εργασιών για το τεχνικό χρέος.

Το εργαλείο δεν χρησιμοποιεί κάποιον συγκεκριμένο αλγόριθμο προκειμένου να δοθεί προτεραιότητα σε συγκεκριμένες εργασίες καθώς αυτό εξαρτάται από την στρατηγική κάθε εταιρείας που το χρησιμοποιεί. Μπορούν ωστόσο να οριστούν φίλτρα τα οποία να αποτελούνται από οποιονδήποτε συνδυασμό των μετρικών που παρέχονται προκειμένου να προκύψει η ταξινομημένη λίστα και να δοθεί η προτεραιότητα από τον υπεύθυνο.[26]

TEDMA

Το TEDMA πρόκειται για ένα εργαλείο που παρουσιάστηκε στο συνέδριο “43rd Euromicro Conference on Software Engineering and Advanced Applications” το 2017. Παρακολουθεί την εξέλιξη διαφόρων δεικτών που αφορούν το τεχνικό χρέος καθώς το έργο εξελίσσεται από γενιά σε γενιά.

Η ιδέα του εργαλείου προέκυψε από την ανάγκη διαχείρισης των πολλών διαφορετικών τεχνικών μέτρησης τεχνικού χρέους και των στρατηγικών αντιμετώπισής τους. Επειδή υπάρχουν πολλές διαφορετικές προσεγγίσεις χωρίς να υπάρχουν ξεκάθαρα στοιχεία για το ποια είναι η πιο ορθή, πρέπει ένα εργαλείο να είναι ευέλικτο και να μπορεί να προσαρμοστεί σε διαφορετικές ανάγκες. Μάλιστα υπάρχει η επιθυμία πολλές φορές να μπορεί να γίνει συνδυασμός στρατηγικών προκειμένου να πάρει η εταιρεία τις μετρήσεις και τα αποτελέσματα που χρειάζεται.

Το TEDMA είναι εργαλείο που είναι ανοιχτό στο να συνδυάζεται και να συνεργάζεται με άλλα εργαλεία ανάλυσης. Λαμβάνει υπόψη ιστορικά δεδομένα του έργου και αποθηκεύει τα δεδομένα του σε βάσεις δεδομένων ώστε να μπορούν να αξιοποιηθούν από άλλα εργαλεία στην συνέχεια. Από την στιγμή που θα ξεκινήσει η ανάλυση ενός έργου τότε κάθε αλλαγή που πραγματοποιείται στον κώδικά του μπορεί να αναλυθεί και αυτή. Υπάρχει μάλιστα διαθέσιμος μηχανισμός ο οποίος εξασφαλίζει πως κάθε αλλαγή και το αντίκτυπό της θα ληφθούν μόνο μία φορά υπόψη κατά την ανάλυση αποφεύγοντας τις επαναλήψεις και τα ψευδή δεδομένα.

Η ανάλυση πραγματοποιείται σε επίπεδο αρχείου. Ωστόσο είναι δυνατόν με την κατάλληλη παραμετροποίηση να υλοποιηθεί η ανάλυση και για άλλα επίπεδα αφαιρετικότητας όπως οι μέθοδοι. Πληροφορίες και μετρικές αποθηκεύονται για κάθε αρχείο και για κάθε γενιά του αρχείου στην βάση που δημιουργεί το εργαλείο. Μέχρι αυτή την στιγμή η ανάλυση υποστηρίζεται μόνο για έργα Java.

Το εργαλείο αποτελείται από τρία επίπεδα. Το πρώτο επίπεδο είναι αυτό της συλλογής δεδομένων. Αυτή την στιγμή διαθέσιμες πηγές για άντληση δεδομένων είναι τα αποθετήρια του Git από όπου μπορεί να ληφθούν πληροφορίες και για τις διάφορες γενιές του έργου. Τα δεδομένα αφού ληφθούν αποθηκεύονται στην βάση του εργαλείου. Επόμενο επίπεδο είναι το επίπεδο υπηρεσιών όπου κατά κύριο λόγο προσφέρονται τρεις υπηρεσίες. Η πρώτη αφορά το φόρτωμα των δεδομένων. Την μετατροπή τους δηλαδή σε τέτοια μορφή ώστε να είναι επεξεργάσιμα από την εφαρμογή. Αυτό το στάδιο το ακολουθεί η ανάλυση από τους δύο διαθέσιμους αναλυτές το PMD όπου εντοπίζονται οσμές κώδικα και το FindBug όπου εντοπίζονται προβλήματα. Επόμενη υπηρεσία είναι αυτή της στατιστικής ανάλυσης η οποία υλοποιείται με την χρήση της γλώσσας R. Τέλος υπάρχει η υπηρεσία μοντέλου διαχείρισης του τεχνικού χρέους η οποία βοηθά στην λήψη αποφάσεων στηριζόμενη σε μοντέλα αποφάσεων που είναι υλοποιημένα σε Java ή R. Το τελευταίο από τα τρία επίπεδα είναι αυτό της παρουσίασης το οποίο ωστόσο είναι υπό ανάπτυξη αλλά αναμένεται να περιέχει μία υπηρεσία για την αυτοματοποιημένη παραγωγή αναφορών και μία υπηρεσία οπτικοποίησης ώστε να παρουσιάζεται στον χρήστη ένας ολοκληρωμένος πίνακας με την τρέχουσα εικόνα του έργου. [27]

AnaConDebt

Ένα σημαντικό μέρος του τεχνικού χρέους είναι το αρχιτεκτονικό χρέος το οποίο προκύπτει από την παρέκκλιση του έργου από τις βασικές αρχές καλής αρχιτεκτονικής. Η αλλαγές σε τέτοιο επίπεδο μπορούν να είναι αρκετά ακριβές και απαιτούν σημαντικό χρόνο. Όσο προχωράει ο κύκλος ζωής ενός έργου τόσο μεγαλύτερο κόστος και αντίκτυπο έχουν τέτοιου είδους δραστικές παρεμβάσεις. Είναι σημαντικό λοιπόν η ανάγκη για τέτοιες παρεμβάσεις να εντοπίζεται και να πραγματοποιείται αν και όταν οι συνθήκες είναι κατάλληλες ώστε να ελαχιστοποιηθεί το κόστος που θα έχουν.

Το AnaConDebt εστιάζει στην απόφαση αποπληρωμής του αρχιτεκτονικού χρέους. Βοηθάει τα διοικητικά στελέχη ώστε να αποφασίσουν πρώτα αν θα πρέπει να αποπληρωθεί ή όχι το χρέος, και έπειτα να εκτιμήσουν τον χρόνο που ενδείκνυται για την αποπληρωμή.

Για το πρώτο σκέλος της απόφασης η γενική αίσθηση είναι πως το κεφάλαιο πρέπει να πληρωθεί μόνο αν είναι μικρότερο του συνολικού επιτοκίου που θα προκύψει κατά την διάρκεια ζωής του έργου. Το εργαλείο εδώ υπολογίζει τον λόγο του τρέχοντος κεφαλαίου προς τον συνολικό τόκο. Αυτό βέβαια προϋποθέτει πως ο κύκλος ζωής του έργου είναι γνωστός εκ των προτέρων. Επειδή κάτι τέτοιο δεν συμβαίνει τις περισσότερες φορές το συνολικό επιτόκιο μπορεί να υπολογιστεί με σημείο αναφοράς το πιο μακροπρόθεσμο μελλοντικό σημείο του έργου που είναι γνωστό. Εάν ο λόγος έχει αποτέλεσμα μεγαλύτερο της μονάδας σημαίνει πως το τρέχον κεφάλαιο είναι μεγαλύτερο του συνολικού επιτοκίου και άρα δεν είναι συμφέρον να αποπληρωθεί. Η εταιρεία θα πρέπει να μάθει να ζει με αυτό το χρέος.

Στην περίπτωση που το χρέος πρέπει να αποπληρωθεί θα πρέπει να αποφασιστεί ποια είναι η κατάλληλη στιγμή για να γίνει αυτή η εξόφληση. Είναι το δεύτερο σκέλος της απόφασης που αφορά το “πότε”. Εδώ αρχικά πρέπει να οριστεί μία μελλοντική περίοδος ως σημείο αναφοράς. Έπειτα πρέπει να υπολογιστούν δύο μελλοντικά ποσά. Το πρώτο είναι το μελλοντικό κεφάλαιο το οποίο είναι το άθροισμα του τρέχοντος κεφαλαίου και του επιπλέον κεφαλαίου που έχει προκύψει εξαιτίας του επιτοκίου. Έπειτα υπολογίζεται το υπολειπόμενο επιτόκιο ως η διαφορά του συνολικού επιτοκίου και του επιτοκίου στο μελλοντικό σημείο

αναφοράς. Παίρνοντας τον λόγο του μελλοντικού κεφαλαίου με το υπολειπόμενο επιτόκιο και αφαιρώντας τον λόγο του τρέχοντος κεφαλαίου με το συνολικό επιτόκιο προκύπτει το μέγεθος που καθορίζει πότε πρέπει να υλοποιηθεί η αλλαγή στο έργο. Εάν αυτός ο αριθμός είναι κοντά στο μηδέν σημαίνει ότι δεν είναι άμεση ανάγκη καθώς η μελλοντική προσαύξηση του χρέους θα είναι ελάχιστη. Εάν όμως ο αριθμός είναι μεγάλος σημαίνει πως όσο περισσότερο καθυστερεί η αλλαγή το κόστος θα είναι τελικά πολλαπλάσιο του αρχικού.

Η πρόκληση σε αυτό το εργαλείο είναι η μεθοδολογία που χρειάζεται προκειμένου να γίνει όσο το δυνατόν πιο ακριβής εκτίμηση του μελλοντικού κεφαλαίου και του επιτοκίου. Για να καταλήξουν στους παράγοντες που μπορούν να επηρεάσουν αυτά τα μεγέθη πραγματοποιήθηκε μία μελέτη σε διαφορετικούς οργανισμούς. Χρειάστηκαν 11 επαναλήψεις της μεθοδολογίας προκειμένου να καταλήξουν στο σύνολο των παραγόντων εσωτερικών και εξωτερικών. Οι εσωτερικοί παράγοντες είναι αυτοί οι οποίοι έχουν αντίκτυπο στην ανάπτυξη και την συντήρηση του έργου. Εξωτερικοί είναι οι παράγοντες που έχουν αντίκτυπο στον πελάτη που χρησιμοποιεί το λογισμικό. Για την αξιολόγηση της τρέχουσας κατάστασης κάθε φορά μπορούσαν να αξιοποιηθούν δεδομένα από εργαλεία ανάλυσης αλλά και ο ίδιος ο πηγαίος κώδικας. Οι μελλοντικές εκτιμήσεις βασίστηκαν στις γνώμες ειδικών. [28]

NDepend

Το NDepend είναι ένα εργαλείο στατικής ανάλυσης το οποίο υποστηρίζει έργα που είναι γραμμένα σε .NET. Είναι διαθέσιμο μέσω του Visual Studio Marketplace και συμβατό με όλες τις εκδόσεις του Visual Studio από το 2010 και μετά. Εμφανίστηκε πρώτη φορά το 2004. Η πρώτη επίσημη όμως έκδοσή του βγήκε στην αγορά τον Απρίλιο του 2007. Θυγατρικό του εργαλείο είναι το JArchitect το οποίο εμφανίστηκε το 2010 και υποστηρίζει την ανάλυση των έργων Java ακολουθώντας την ίδια μεθοδολογία με το NDepend. Το JArchitect [29] είναι συμβατό με τους δημοφιλέστερους IDE της Java. Για την χρήση και των δύο εργαλείων απαιτείται εμπορική άδεια.

Η φιλοσοφία του εργαλείου είναι να αντιμετωπίζει τον πηγαίο κώδικα σαν μία μορφή βάσης δεδομένων. Για να αντλήσει ο χρήστης τα αποτελέσματα που θέλει πρέπει να κάνει

ερωτήματα σε αυτή την βάση. Υπάρχουν ήδη κάποια ερωτήματα έτοιμα προς χρήση που αποτελούν το βασικό σύνολο κανόνων ανάλυσης. Ο χρήστης μπορεί να παρέμβει και να τροποποιήσει αυτούς τους κανόνες καθώς ο κώδικας που τους υλοποιεί είναι διαθέσιμος ή ακόμα και να προσθέσει καινούργιους κανόνες ανάλογα με τις ανάγκες κάθε οργανισμού αλλά και κάθε ξεχωριστού έργου. Οι κανόνες υλοποιούνται με την χρήση της LINQ. [30]

Το εργαλείο διαθέτει ειδικό API το οποίο είναι επιφορτισμένο με τον υπολογισμό του τεχνικού χρέους και του ετήσιου επιτοκίου που προκύπτει από αυτό. Και τα δύο αυτά μεγέθη μετριούνται σε ανθρωποώρες. Για κάθε πρόβλημα που εντοπίζεται στον κώδικα υπάρχει μία μέθοδος που υπολογίζει το κόστος της επίλυσής του αλλά και τον μέσο όρο του επιπλέον μελλοντικού κόστους σε περίπτωση που δεν διορθωθεί το πρόβλημα. Ο χρόνος που απαιτείται για την διόρθωση ενός τμήματος του κώδικα, μπορεί να επηρεάζεται από διάφορους παράγοντες εκτός της ίδιας της φύσης του προβλήματος. Τέτοιοι παράγοντες είναι για παράδειγμα η παρουσία δομών ελέγχου. Μία μέθοδος που διαθέτει τους απαραίτητους μηχανισμούς ελέγχου μπορεί να τροποποιηθεί και συνεπώς να διορθωθεί πολύ πιο γρήγορα από μία αντίστοιχη η οποία δεν ελέγχεται από πουθενά. Και αυτό γιατί στην πρώτη περίπτωση μπορούν εύκολα να ελεγχθούν οι συνέπειες της τροποποίησης και να διασφαλιστεί η ομαλή λειτουργία του υπόλοιπου συστήματος τουλάχιστον στα επίπεδα πριν την παρέμβαση στον κώδικα. Ο χρόνος που απαιτείται για την επίλυση καθώς και το σύνολο των παράπλευρων παραγόντων που μπορούν να οδηγήσουν σε επιπλέον καθυστέρηση είναι και αυτά παραμετροποιήσιμα από τον χρήστη καθώς διατίθενται σε μορφή scripts.

Και σε αυτό το εργαλείο τα προβλήματα που εντοπίζονται χωρίζονται σε πέντε κατηγορίες σημαντικότητας οι οποίες από την λιγότερο στην περισσότερο σοβαρή, τιτλοφορούνται ως εξής: Info, Minor, Major, Critical, Blocking. Και οι κανόνες όμως έχουν δύο κατηγορίες οι “Κρίσιμοι” (Critical) και οι μη. Το επίπεδο κρισιμότητας των κανόνων δεν σχετίζεται άμεσα με το επίπεδο σοβαρότητας του προβλήματος. Πρόκειται για έναν δείκτη ο οποίος είναι παραμετροποιήσιμος από τον χρήστη και μπορεί με αυτόν τον τρόπο να θέσει όρια ως προς το ποιοι κανόνες είναι απαραίτητο να ικανοποιούνται προκειμένου το έργο να περάσει σε επόμενο επίπεδο ποιότητας (quality gate).

Ένα μέρος της παραμετροποίησης του εργαλείου απαιτεί γνώσεις κώδικα. Υπάρχει όμως και ένα γραφικό περιβάλλον προκειμένου να οριστούν οι βασικές παράμετροι που αποτελούν την βάση των υπολογισμών. Τέτοιες είναι για παράδειγμα ο αριθμός των εργάσιμων ωρών εντός της ημέρας και των εργάσιμων ημερών ενός έτους, καθώς επίσης και το κόστος της κάθε ανθρωποώρας.

Το εργαλείο αποτιμά το τεχνικό χρέος με την μέθοδο SQALE. Πιο συγκεκριμένα χρησιμοποιεί τους δείκτες “Debt Ratio” (ρυθμός χρέους) και “Debt Rating” (κατάταξη χρέους) που αποτελούν μέρος του δείκτη SQALE. Ο ρυθμός χρέους εκφράζεται ως το ποσοστό του εκτιμώμενου τεχνικού χρέους ως προς την εκτιμώμενη προσπάθεια που θα χρειαζόταν να ξαναγραφτεί ο κώδικας από την αρχή. Η ποσοτικοποίηση της προσπάθειας βασίζεται κυρίως σε εμπειρικά δεδομένα του οργανισμού και είναι και αυτή μία από τις παραμέτρους που ορίζει ο χρήστης στο σύστημα. Η κατάταξη χρέους είναι μία κλίμακα με πέντε στάδια. Σε κάθε στάδιο ο χρήστης μπορεί να ορίσει ένα ποσοστό χρέους ως κατώφλι. Έτσι η προηγούμενη μέτρηση οδηγεί στην αντίστοιχη κατάταξη του δείκτη “Debt Ranking”.[31]

CodeScene

Το CodeScene της Empear εμφανίστηκε πρώτη φορά στην αγορά μόλις το 2016. Στο άρθρο [32] «Assessing Technical Debt in Automated Tests with CodeScene» το εργαλείο παρουσιάζεται ως ένα μέσο να καλύψει την αναγκαιότητα ελέγχου και διατήρησης της ποιότητας στους αυτοματοποιημένους ελέγχους ενός έργου. Η προσέγγιση αυτή δεν διαφέρει καθόλου από το να συμμετέχει στην ανάλυση ο κώδικας του κορμού του έργου και αυτό γιατί και οι αυτοματοποιημένοι έλεγχοι αποτελούνται από γραμμές κώδικα. Είναι και αυτά από μόνα τους προγράμματα με μεθόδους και όλα τα χαρακτηριστικά που έχει ένα έργο λογισμικού. Υπόκεινται σε αλλαγές και τροποποιήσεις και αντιμετωπίζουν τους ίδιους κινδύνους εάν η ποιότητά του κώδικα που τους αποτελεί δεν είναι η πρέπουσα. Μάλιστα υπάρχουν περιπτώσεις όπου ο κώδικας που υπάρχει στους αυτοματοποιημένους ελέγχους ενός έργου είναι περισσότερος από αυτόν που αποτελεί το ίδιο το έργο.

Οι αυτοματοποιημένοι έλεγχοι κερδίζουν διαρκώς έδαφος και εφαρμόζονται όλο και περισσότερο. Αυτό συμβαίνει καθώς τα έργα λογισμικού που υλοποιούνται είναι όλο και μεγαλύτερα. Τα προβλήματα που καλούνται να αντιμετωπίσουν είναι περισσότερο πολύπλοκα και συνεχώς μεγαλύτερες ομάδες συμμετέχουν στην ανάπτυξή τους. Οι αυτοματοποιημένοι έλεγχοι μπορούν να διασφαλίσουν την ορθή λειτουργία του έργου ανεξάρτητα από τις μεταβολές που δέχεται και να το κάνουν ανεκτικό σε αλλαγές.

Το CodeScene λοιπόν είναι ένα εργαλείο το οποίο συνδυάζει μία σειρά από μεθόδους μελέτης και ανάλυσης δεδομένων προκειμένου να δομήσει μία διαδικασία λήψης αποφάσεων ως προς την σειρά που θα πρέπει να εκτελεστούν οι εργασίες μεταβολών σε ένα έργο επιτυγχάνοντας τον υψηλότερο δείκτη απόδοσης. Συνδυάζει την άντληση δεδομένων από αποθετήρια κώδικα, με την στατική ανάλυση και την μηχανική μάθηση.

Ξεκινώντας από την στατική ανάλυση συλλέγονται δεδομένα που αφορούν τα προβλήματα που εντοπίζονται σε ένα έργο και οι μετρικές του. Το πρόβλημα εδώ προκύπτει από το γεγονός πως στην στατική ανάλυση όλος ο κώδικας αντιμετωπίζεται ως ίσης σημασίας. Όπως έχει αναφερθεί προηγούμενα ωστόσο η κρισιμότητα ενός προβλήματος εξαρτάται από το σημείο στο οποίο εμφανίζεται και το κατά πόσο αυτό το σημείο χρησιμοποιείται στο έργο και επηρεάζει και άλλα σημεία του ίδιου του έργου ή της διαδικασίας της επιχείρησης. Τέτοια δεδομένα συμπεριφοράς και οι εξωτερικοί παράγοντες που μπορούν να επηρεάζουν την σημασία ενός προβλήματος δεν μπορούν να μελετηθούν με την στατική ανάλυση. Μπορούν όμως να αντληθούν από το ιστορικό και τα δεδομένα που παρέχονται στα αποθετήρια κώδικα και αυτό καθιστά αναγκαία την εξέτασή τους.

Αφού ολοκληρωθεί η συλλογή των μετρικών αυτές πρέπει να αξιολογηθούν και να τους ανατεθεί το αντίστοιχο βάρος που έχουν στο τεχνικό χρέος. Το βάρος και η σημασία τους διαφέρει μεταξύ των έργων. Είναι τα χαρακτηριστικά ενός έργου όπως το μέγεθος και η κρισιμότητα του προβλήματος το οποίο διαχειρίζεται, αυτά που καθορίζουν την σημασία που έχει η κάθε μετρική για αυτό. Το CodeScene εδώ αξιοποιεί τις τεχνικές μηχανικής μάθησης. Προσπαθεί να ανακαλύψει μοτίβα τέτοιων χαρακτηριστικών στα έργα λογισμικού και με βάση αυτά να κατατάξει τις μετρικές.

Τελικά αφού ολοκληρωθεί και αυτή η αξιολόγηση το τελικό αποτέλεσμα που προκύπτει είναι μία αναφορά όπου τα αρχεία του έργου παρουσιάζονται σε κατάταξη ανάλογα με το συνολικό αντίκτυπο που έχουν στις διαδικασίες της επιχείρησης. Έτσι λοιπόν τα διοικητικά στελέχη μπορούν να έχουν μία εκτίμηση για το ποιες εργασίες πρέπει να δρομολογηθούν πρώτες ώστε να υπάρχει το μεγαλύτερο όφελος. Η διαδικασία είναι πλήρως αυτοματοποιημένη και το μόνο που χρειάζεται να δοθεί ως είσοδος στο εργαλείο είναι το αποθετήριο όπου βρίσκεται ο πηγαίος κώδικας

DebtFlag

Το εργαλείο DebtFlag αναπτύχθηκε για να εντοπίζει, να παρακολουθεί και να βοηθάει στην επίλυση προβλημάτων που σχετίζονται με το τεχνικό χρέος σε έργα λογισμικού γραμμένα σε Java. Το εργαλείο αυτό αποτελείται από δύο μέρη. Το πρώτο μέρος είναι ένα πρόσθετο το οποίο μπορεί να χρησιμοποιηθεί σε συνεργασία με τον Eclipse IDE. Το δεύτερο μέρος είναι μία διαδικτυακή εφαρμογή.

Το πρόσθετο του Eclipse είναι υπεύθυνο για την ανάλυση του κώδικα και τον εντοπισμό των προβλημάτων. Πρέπει να εντοπίσει το σημείο στο οποίο παρουσιάζεται το πρόβλημα αλλά και τις διασυνδέσεις που έχει αυτό το σημείο με άλλα τμήματα του έργου μέσω των εξαρτήσεων. Μόλις εντοπιστούν τα προβλήματα δημιουργούνται τα αντικείμενα (TDI: Technical Debt Items) τα οποία θα ληφθούν υπόψη κατά τον υπολογισμό του χρέους. Αυτά τα αντικείμενα είναι οργανωμένα στην λίστα TDL: (Technical Debt List) και μορφολογικά ακολουθούν την δομή που προτείνεται από το TDMF (Technical Debt Management Framework). Η δομή αυτή όπως συνέβη και σε προηγούμενο εργαλείο τροποποιήθηκε προκειμένου να καλύπτει τις ανάγκες του DebtFlag. Έτσι λοιπόν κάθε TDI αντικείμενο αποτελείται από μία λίστα με DebtFlag αντικείμενα τα οποία αποτελούν την σύνδεση του προβλήματος με το σημείο του κώδικα το οποίο επηρεάζει. Αυτά με την σειρά τους περιέχουν πληροφορίες σχετικά με το σημείο που εντοπίζεται το πρόβλημα, την χρονική στιγμή που εντοπίστηκε αλλά και ποιος είναι ο συγγραφέας του. Όλα αυτά τα δεδομένα μπορεί να τα αντλήσει από τα μεταδεδομένα που προσφέρει το περιβάλλον ανάπτυξης του Eclipse. Κάθε DebtFlag αντικείμενο μπορεί να έχει έναν ή και περισσότερους τύπους τεχνικού χρέους. Όλοι αυτοί απαριθμούνται σε μία λίστα

που βρίσκεται μέσα στην δομή του. Ο λόγος για τον οποίο για κάθε TDI αντικείμενο μπορούν να υπάρχουν πολλά DebtFlag αντικείμενα είναι γιατί ένα προβληματικό κομμάτι κώδικα δεν προκαλεί θέμα μόνο σε ένα σημείο. Μέσω των κλήσεων και των εξαρτήσεων που μπορεί να υπάρχουν στο έργο το πρόβλημα μπορεί να μεταδίδεται μεταξύ των οντοτήτων. Για αυτό είναι σημαντικό για το εργαλείο να μπορεί να επεξεργάζεται και να γνωρίζει το δέντρο των εξαρτήσεων του έργου.

Αφού αναγνωριστούν τα αντικείμενα με τον ίδιο δομημένο τρόπο γράφονται σε μία βάση δεδομένων. Αυτή η βάση είναι ο ενδιάμεσος συνδετικός κρίκος μεταξύ των δύο εφαρμογών του πρόσθετου και της διαδικτυακής εφαρμογής. Η διαδικτυακή εφαρμογή έπειτα διαβάζει τα δεδομένα από την βάση και πλέον είναι υπεύθυνη για να οπτικοποιήσει τα αποτελέσματα και να παρουσιάσει σχετικές αναφορές.

Με τον τρόπο που αποθηκεύει το εργαλείο τα δεδομένα επιτρέπει να γίνουν δύο είδη διαχείρισης του τεχνικού χρέους. Η πρώτη μορφή είναι αυτή που πραγματοποιείται σε επίπεδο έργου και επιχείρησης και είναι πιο κεντρική και γενική. Η δεύτερη είναι πιο λεπτομερής και γίνεται σε επίπεδο υλοποίησης. Στην δεύτερη περίπτωση είναι δυνατή η εφαρμογή μεθόδων μικροδιαχείρισης (micromanagement).[33]

VisminerTD

Το VisminerTD είναι ένα εργαλείο ανοιχτού κώδικα (ο κώδικας των σχετικών εργαλείων είναι διαθέσιμος στα παρακάτω αποθετήρια [34] hb.com/visminer) το οποίο στηρίζεται στο διαδίκτυο και ασχολείται με την αναγνώριση την επίβλεψη και την διαχείριση του τεχνικού χρέους. Ιδιαίτερη σημασία δίνεται από το εργαλείο στην σωστή παρουσίαση των αποτελεσμάτων με τρόπο τέτοιο ώστε να είναι εύκολα κατανοητά. Ο χρήστης πρέπει να μπορεί με μία γρήγορη ματιά να έχει την γενική εικόνα του προϊόντος. Το εργαλείο παρακολουθεί το έργο καθώς εξελίσσεται από γενιά σε γενιά και συνδυάζει ένα σύνολο μετρικών με την ανάλυση των σχολίων του κώδικα προκειμένου να εξάγει τα αποτελέσματα. Μόλις λοιπόν εντοπιστεί κάποιο πρόβλημα στον κώδικα αυτό καταγράφεται και έπειτα παρακολουθείται. Έτσι οι διαχειριστές μπορούν να γνωρίζουν πότε και πόσο αυξάνεται το

κόστος που προκαλεί αυτό το πρόβλημα ή την χρονική στιγμή που το χρέος του αποπληρώθηκε και άρα σταμάτησε να είναι επιζήμιο για το σύστημά.

Πυρήνας του εργαλείου είναι το Repository Miner, επίσης ανοιχτού κώδικα διαθέσιμου στο ίδιο αποθετήριο. Πρόκειται για ένα Java Framework το οποίο εξάγει δεδομένα διαβάζοντας τον κώδικα που είναι διαθέσιμος σε αποθετήρια. Συλλέγει ένα σύνολο δεδομένων που αφορούν τις συνεισφορές σε κώδικα (commits), τους προγραμματιστές, τις τροποποιήσεις που γίνονται αλλά και μετρικές που αφορούν το έργο. Συνολικά το εργαλείο μπορεί να εξάγει μία λίστα 19 μετρικών και να εντοπίσει 7 τύπους οσμών κώδικα. Ως προβληματικός χαρακτηρίζεται ο κώδικας που είναι διπλότυπος επίσης, ενώ προβληματικά σημεία μπορούν να προκύψουν και από την ανάλυση των σχολίων που σχετίζονται με το τεχνικό χρέος. Το εργαλείο αναγνωρίζει συνολικά 9 τύπους χρέους οι οποίοι περιλαμβάνουν το αρχιτεκτονικό, το χρέος χτισίματος, το χρέος στον κώδικα, χρέος σχεδιασμού, ελαττωμάτων, τεκμηρίωσης, απαιτήσεων, ανθρώπων και ελέγχου. Τα αποτελέσματα αυτά έπειτα αποθηκεύονται σε μία βάση δεδομένων MongoDB προκειμένου να είναι διαθέσιμα προς χρήστη στην περαιτέρω ανάλυση.

Το εργαλείο ανάλυσης του VisminerTD τρέχει πάνω από το Repository Miner και αποτελείται από δύο κυρίως μέρη. Το VisminerTD Client και το VisminerTD Service. Το VisminerTD Service είναι ένα REST Service το οποίο είναι υπεύθυνο να επικοινωνεί με την βάση. Το VisminerTD Client είναι το ενδιάμεσο πρόγραμμα το οποίο χρησιμοποιείται προκειμένου να γίνει η αναζήτηση συγκεκριμένων αποτελεσμάτων. Ο client ρωτάει το service δίνοντάς του τις απαραίτητες παραμέτρους και μόλις λάβει την απάντηση επεξεργάζεται τα δεδομένα και τα εμφανίζει στον χρήστη.

Μεγάλη σημασία δίνεται από το εργαλείο όπως αναφέρθηκε στην οπτικοποίηση των αποτελεσμάτων. Υπάρχει μία σειρά οθονών οι οποίες χρησιμοποιούνται προκειμένου να δοθεί και η γενική αλλά και η αναλυτική κατά περίπτωση εικόνα του έργου. Για την οπτικοποίηση χρησιμοποιούνται τα HighCharts. Το εργαλείο TDAnalyzerView είναι υπεύθυνο για την εμφάνιση των προβληματικών αντικειμένων που εντοπίστηκαν. Από αυτή την λίστα που προκύπτει ο χρήστης μπορεί να επιβεβαιώσει πως πρόκειται για προβληματικό αντικείμενο

που δημιουργεί χρέος ή να το απορρίψει. Έπειτα υπάρχει η οθόνη TD form και η οθόνη TD Management View μέσω των οποίων πραγματοποιείται η διαχείριση του χρέους. Η οθόνη TD Timeline δείχνει την εξέλιξη του χρέους στις διαφορετικές γενιές του προϊόντος μαζί με την οθόνη TD Evaluation view η οποία επιτρέπει την σύγκριση μεταξύ γενεών. Τέλος υπάρχουν οθόνες που παρέχουν γραφική αναπαράσταση των μετρικών και των αποτελεσμάτων της ανάλυσης.[35]

DebtGrep

Η περίπτωση του DebtGrep διαφέρει από τα εργαλεία που παρουσιάστηκαν ως τώρα. Δεν πρόκειται για ένα εργαλείο που μετράει το τεχνικό χρέος, αλλά για ένα εργαλείο που έχει ως σκοπό να προλάβει το τεχνικό χρέος πριν αυτό δημιουργηθεί. “Κάλλιον του θεραπεύειν το προλαμβάνειν” και στην προκειμένη περίπτωση και πιο φθηνό.

Το λογισμικό αναπτύχθηκε από την Ericsson 4G 5G Baseband και χρησιμοποιήθηκε από την εταιρεία το 2015. Το λογισμικό τηλεπικοινωνιών που καλείται να συντηρήσει η εταιρεία πέρα από το μέγεθός του είναι απαιτητικό και ως προς την διαχείρισή του γιατί στην ομάδα ανάπτυξης συμμετέχουν πολλοί προγραμματιστές γράφοντας κώδικα σε κομμάτια που πρέπει να συνδυαστούν εν τέλη μεταξύ τους. Οι προγραμματιστές σε τέτοια έργα είναι πιθανό να μην βρίσκονται στον ίδιο χώρο, ούτε καν στην ίδια χώρα. Γεγονός που καθιστά την επικοινωνία μεταξύ τους για κοινές προγραμματιστικές τεχνικές κάτι παραπάνω από αναγκαία. Το DebtGrep προσπαθεί να επιβάλει μία πειθαρχία ως προς τις τεχνικές και τα εργαλεία που χρησιμοποιούνται κατά την ανάπτυξη.

Η καρδιά του εργαλείου είναι ένα αρχείο στο οποίο γράφονται οι κανόνες χρησιμοποιώντας regex (Regular Expressions). Το αρχείο αυτό περιέχει όλες τις λέξεις κλειδιά που είναι απαγορευμένες και δεν πρέπει να εμφανίζονται στον κώδικα. Έτσι λοιπόν αν μία μέθοδος έχει καταργηθεί και θεωρείται ξεπερασμένη μπορεί η υπογραφή της να συμπεριληφθεί στο συγκεκριμένο αρχείο και να προειδοποιείται ο προγραμματιστής κάθε φορά που προσπαθεί να την χρησιμοποιήσει. Το αρχείο αυτό μπορεί να περιλαμβάνει και εξαιρέσεις κανόνων καθώς και συγκεκριμένους φακέλους με αρχεία στα οποία πρέπει να εφαρμόζεται κάθε

κανόνας. Με αυτόν τον τρόπο μπορούν να περιοριστούν οι κανόνες ως απαγορευτικοί μόνο στον νέο κώδικα, ώστε το απαραίτητο refactor που απαιτεί ο παλιός κώδικας να προγραμματιστεί σε μελλοντική ημερομηνία χωρίς να αποτελεί εμπόδιο στην προσθήκη νέου. Το γεγονός ότι οι κανόνες περιγράφονται με κανονικές εκφράσεις τους δίνει την δυνατότητα να είναι ανεξάρτητοι της γλώσσας προγραμματισμού πράγμα ιδιαίτερα χρήσιμο σε περιπτώσεις όπου στο ίδιο έργο λογισμικού διαφορετικά κομμάτια υλοποιούνται σε διαφορετικές γλώσσες και τεχνολογίες[36].

Μεθοδολογία

Για την πραγματοποίηση αυτής της μελέτης επιλέχθηκαν δέκα έργα της Apache Foundation. Τα έργα είναι ανοιχτού κώδικα με πρόσβαση σε αυτόν μέσω των αποθετηρίων του GitHub. Επιλέχθηκε η μελέτη του τεχνικού χρέους να γίνει σε επίπεδο κλάσεων, καθώς εκεί βρίσκεται η ρίζα του προβλήματος η οποία αθροιστικά μας δίνει τελικά την τιμή του τεχνικού χρέους για όλο το έργο. Οι κλάσεις λοιπόν που προέκυψαν ως προβληματικές μετά από την ανάλυση είναι περίπου 12.000.

Χρησιμοποιήθηκαν τρία εργαλεία ανάλυσης κώδικα τα οποία παρέχουν μετρικές αποτίμησης τεχνικού χρέους. Ταυτόχρονα για να μπορούν να διαβαστούν και να αξιοποιηθούν τα δεδομένα για περαιτέρω έρευνα χρειάστηκε να αναπτυχθεί ένα πρόγραμμα σε JAVA το οποίο επιφορτίστηκε με την συλλογή και την κατάλληλη μορφοποίησή τους.

Στην συνέχεια πραγματοποιήθηκε η στατιστική μελέτη των αποτελεσμάτων. Αρχικά χρησιμοποιήθηκε η μέθοδος του Kendall’s W coefficient of concordance προκειμένου να ελεγχθεί εάν τα εργαλεία ανάλυσης κώδικα συμφωνούν μεταξύ τους. Στην συνέχεια εφαρμόστηκε η ανάλυση αρχετύπων ώστε να εντοπιστούν τα αρχέτυπα προφίλ για τα σύνολα των αποτελεσμάτων και αν γίνει η επιλογή του αρχέτυπου του οποίου η γειτονιά θα χρησιμοποιηθεί προκειμένου να προκύψει το benchmark. Στην τρέχουσα περίπτωση η γειτονιά που εμφανίζει το μεγαλύτερο ενδιαφέρον είναι η γειτονιά με τις πιο προβληματικές κλάσεις. Η μελέτη των χαρακτηριστικών αυτών των κλάσεων αποτελεί το benchmark. Με αυτά ως δεδομένα πλέον κάθε κλάση θα μπορεί να συγκριθεί και να χαρακτηριστεί ως πολύ προβληματική ή μη, χωρίς να προηγηθεί ολόκληρη η ανάλυση ξανά.

Επιλογή των projects προς ανάλυση

Σε μία μελέτη όπως αυτή η επιλογή των δεδομένων προς ανάλυση είναι ιδιαίτερα σημαντική και πρέπει να πραγματοποιηθεί με τέτοιο τρόπο ώστε τα αποτελέσματα που θα προκύψουν να είναι αντικειμενικά και όσο το δυνατόν πιο αντιπροσωπευτικά του μέσου όρου των έργων λογισμικού που υπάρχουν. Προτιμήθηκαν έργα τα οποία έχουν ικανοποιητικό αριθμό

γραμμών κώδικα και τα προβλήματα που επιλύουν είναι αρκετά πολύπλοκα ώστε να ταιριάζουν στις προδιαγραφές των περισσοτέρων εμπορικών λογισμικών και να έχουν περίπου τις ίδιες πιθανότητες εμφάνισης τεχνικού χρέους.

Τα open source έργα της Apache Software Foundation [37] αποτελούν μία καλή αρχή έρευνας γιατί ο κώδικάς τους υπάρχει διαθέσιμος σε δημόσια αποθετήρια και υπάρχει πρόσβαση στις διάφορες γενιές λογισμικού. Έτσι μπορεί να μελετηθεί και η πρόοδος του τεχνικού χρέους από γενιά σε γενιά. Ταυτόχρονα σε αυτά τα έργα συμμετέχουν αρκετοί μηχανικοί λογισμικού με διαφορετικά υπόβαθρα, εμπειρίες και τεχνικές γραφής. Παρά την μεγάλη διαφοροποίηση ωστόσο που υπάρχει τα έργα αυτά είναι λειτουργικά και αναγνωρισμένα σε παγκόσμια κλίμακα. Είναι λοιπόν αρκετά ενδιαφέρον πως η σύνθεση αυτή μπορεί να οδηγήσει σε ένα τόσο ολοκληρωμένο αποτέλεσμα και το τι κοστίζει αυτή η διαφορετικότητα σε τεχνικό χρέος στο έργο.

Τα διαθέσιμα έργα είναι αρκετά και η μελέτη δεν μπορούσε να επεκταθεί σε όλα αυτά. Συνεπώς έπρεπε να τεθούν κάποια πιο συγκεκριμένα κριτήρια για την επιλογή τους. Αποφασίστηκε η αναζήτηση να επικεντρωθεί σε έργα της JAVA για λόγους ομοιομορφίας. Τα έργα της Apache περιλαμβάνουν διαφορετικές γλώσσες πολλές φορές ακόμα και μέσα στο ίδιο project. Τα έργα λοιπόν που αποτελούνται 100% από κώδικα JAVA είναι περιορισμένα και δεν θα ήταν αρκετά για τις ανάγκες της μελέτης. Για αυτό τον λόγο επιλέχθηκε ως threshold το ποσοστό του κώδικα της JAVA σε κάθε έργο να μην είναι μικρότερο του 97%. Το ποσοστό αυτό υπολογίστηκε με βάση τα στατιστικά στοιχεία που προσφέρει το github [38] για κάθε project το οποίο φιλοξενεί, την δεδομένη στιγμή της επιλογής του κάθε έργου (Οκτώβριος 2018).

Παραμένοντας στο κριτήριο της ομοιομορφίας τα έργα της JAVA έπρεπε να περιοριστούν ακόμα περισσότερο. Πλέον ως επιπλέον κριτήριο εισάγεται η αρχιτεκτονική του project. Η ίδια η Apache έχει εισάγει ένα project και ταυτόχρονα ένα πρότυπο ανάπτυξης λογισμικού το Maven [39]. Το ίδιο το project τελικά επιλέχθηκε να συμμετάσχει στην ανάλυση, αλλά και όλα τα υπόλοιπα που επιλέχθηκαν ακολουθούν την ίδια αρχιτεκτονική.

Η σημασία της επιλογής του Maven ως αρχιτεκτονική προκύπτει από την ίδια την ανάγκη που γέννησε το project. Σύμφωνα με την ίδια την Apache υπήρχε η επιθυμία για μία κεντρική δομή μέσω της οποία θα μπορούσε να γίνει η διαχείριση ενός έργου λογισμικού με τρόπο ομογενοποιημένο ανεξάρτητα από την φύση του προβλήματος που αυτό διαχειρίζεται. [40] Το Maven επιτρέπει, και κατά μία έννοια οδηγεί, στην δημιουργία μικρότερων τμημάτων κώδικα τα λεγόμενα Modules τα οποία μπορούν να αποτελέσουν ανεξάρτητους τομείς, αλλά σε συνδυασμό δομούν το καθολικό σύστημα. Η συντηρησιμότητα αυτών των μικρότερων τμημάτων είναι πιο εύκολη, ενώ ταυτόχρονα η διαχείριση των εξαρτήσεων μεταξύ τους έχει επίσης προβλεφθεί και πραγματοποιείται σε υψηλό επίπεδο χωρίς ο προγραμματιστής να μπαίνει σε λεπτομέρειες που αφορούν την δομή του συστήματος. Κρύβονται έτσι αρκετές πολυπλοκότητες της αρχιτεκτονικής και τα έργα αποκτούν συνοχή, ενώ η αναπαραγωγή συγκεκριμένων καταστάσεων γίνεται πιο εύκολη και γρήγορη.[41] Αποδεσμεύεται με αυτόν τον τρόπο ο προγραμματιστής από ένα μεγάλο φορτίο, κερδίζοντας χρόνο ο οποίος ιδανικά θα μπορούσε να επενδυθεί στην ποιότητα του κώδικα διορθώνοντας και μελλοντικά μειώνοντας το τεχνικό χρέος του έργου.

Επόμενο κριτήριο επιλογής ήταν η ημερομηνία της τελευταίας δημοσιευμένης έκδοσης του λογισμικού. Η πληροφορία αυτή επίσης είναι διαθέσιμη στα αποθετήρια. Κάθε καινούρια έκδοση λογισμικού εκτός από την εισαγωγή νέων χαρακτηριστικών περιέχει και βελτιώσεις. Είναι σημαντικό λοιπόν ο κώδικας που επιλέχθηκε προς ανάλυση να μην είναι παλιός αλλά να είναι ενεργός με τις όποιες αλλαγές αλλά και βελτιώσεις επιφέρουν οι συνεχείς προσθήκες των προγραμματιστών. Έτσι λοιπόν όλα τα έργα που επιλέχθηκαν είχαν τουλάχιστον μία καινούρια δημοσιευμένη έκδοση μέσα στον τελευταίο χρόνο από την στιγμή της επιλογής τους.

Έχοντας πλέον περιορίσει αρκετά το πεδίο ορισμού του συνόλου των έργων λογισμικού, για την τελική επιλογή λήφθηκε υπόψη ο συνολικός αριθμός γραμμών του κάθε έργου. Τα έργα πρέπει να είναι διαφόρων μεγεθών. Αφ’ ενός με αυτόν τον τρόπο αυξάνεται το εύρος των έργων με παρόμοια χαρακτηριστικά τα οποία αντιπροσωπεύονται στην μελέτη. Αφετέρου όσο περισσότερος κώδικας υπάρχει σε ένα πρόγραμμα τόσο περισσότερες είναι οι πιθανές

εμφανίσεις χρέους. Συνεπώς δεν έπρεπε να υπάρξει περιορισμός ούτε για τα μικρότερα έργα τα οποία ίσως δεν έχουν τόσο μεγάλο τεχνικό χρέος, ούτε όμως και για τα μεγάλα έργα τα οποία έχουν, κατ' απόλυτη τιμή, περισσότερα προβληματικά σημεία.

Με γνώμονα λοιπόν όλα τα προηγούμενα κριτήρια τα έργα τα οποία επιλέχθηκαν προς μελέτη είναι τα εξής:

Table 1 – Τα έργα της Apache που αναλύθηκαν

Project	LOC	Java Dens.
mina	34549	99.4%
deltaspike	145662	97.3%
opennlp	93208	99.5%
maven	106221	99.5%
wss4j	135742	100%
pdfbox	213436	99.2%
fop	292091	97.9%
cayenne	348242	99.7%
jclouds	482330	99.4%
openjpa	571702	99.2%

Παρουσίαση των έργων

MINA

Το MINA είναι το ακρωνύμιο του "Multipurpose Infrastructure of Network Application". Είναι ένα framework το οποίο χρησιμοποιείται από δικτυακές εφαρμογές. Προσφέρει βιβλιοθήκες οι οποίες υλοποιούν διάφορα πρωτόκολλα επικοινωνίας όπως το TCP και το UDP. Το MINA προήλθε από τον συνδυασμό των χαρακτηριστικών δύο προγενέστερων frameworks του Netty2 και του SEDA. Εμφανίστηκε στην οικογένεια της Apache το 2004. [42]

Το έργο αποτελείται από 34.549 γραμμές κώδικα, το 99.4% από τις οποίες είναι γραμμένες σε JAVA. Στο αποθετήριό του εμφανίζονται 16 προγραμματιστές να έχουν συμβάλει στην ανάπτυξή του. Η τελευταία δημοσιευμένη έκδοση του που είναι και αυτή που αναλύθηκε εδώ, είναι η 2.0.19 με ημερομηνία 07 Ιουλίου 2018. [41]

Deltaspike

Το έργο Deltaspike είναι ένας διαχειριστής των CDI. Προσπαθεί να αποδεσμεύσει τον προγραμματιστή από τις ιδιαιτερότητες του περιβάλλοντος ανάπτυξης, έτσι ώστε να μπορεί να χρησιμοποιηθεί ο ίδιος κώδικας σε διαφορετικές υλοποιήσεις CDI.

Εν συντομία CDI είναι η συντομογραφία του “Context and Dependency Injection”. Είναι μία έννοια την οποία συναντάμε στην Java EE (Java Enterprise Edition) από την έκδοση 6 και μετά. Ορίζεται η εξάρτηση μιας οντότητας Α από μία οντότητα Β, εφόσον η Β γίνει εισαγωγή (“Inject”) στην Α. Ο προγραμματιστής δηλώνει τον τύπο και την σημασία της οντότητας Β. Η οντότητα Α δεν χρειάζεται να γνωρίζει τον κύκλο ζωής της οντότητας Β ούτε τις υπόλοιπες λεπτομέρειες υλοποίησής της. Μάλιστα στην περίπτωση που μια οντότητα γίνει “Inject” δεν απαιτείται να υπάρχει κατασκευαστής της οντότητας σε αυτή την κλάση προκειμένου να κληθούν μέθοδοι επί αυτού του αντικειμένου. Σχεδόν όλες οι POJO (Plain Old Java Object) κλάσεις μπορούν να γίνουν “Inject” σε κάποια άλλη. Το κέρδος που απορρέει από εδώ είναι το ευκολότερο refactor αλλά και ο έλεγχος σε επίπεδο μεταγλώττισης (compile), πριν το στάδιο της ανάπτυξης (deploy) της εφαρμογής στον διακομιστή εφαρμογών (application server). [43]

Το Deltaspike αποτελείται από 145.662 γραμμές κώδικα. Είναι το έργο με το χαμηλότερο ποσοστό σε γραμμές Java καθώς το ποσοστό τους είναι 97.3%. Το υπόλοιπο αποτελείται κυρίως από HTML. Εδώ οι προγραμματιστές που φαίνεται ότι έχουν συνεισφορά είναι 44. Η τελευταία του δημοσίευση που είναι και μέρος αυτής της μελέτης έχει ημερομηνία 15 Μαΐου 2018 και είναι η 1.8.2.[44]

OpenNLP

Με τα αρχικά NLP αναφερόμαστε στο “Natural Language Processing” δηλαδή στην επεξεργασία της φυσικής γλώσσας από υπολογιστικά συστήματα. Η αναγνώριση των λέξεων μία προς μία ήταν ένα πρόβλημα που λύθηκε νωρίς. Όμως αυτό δεν είναι αρκετό καθώς έπρεπε να προχωρήσει η κατανόηση του νοήματος της γλώσσας. Η διαδικασία αυτή είχε και έχει ακόμα πάρα πολλές δυσκολίες. Ακόμα και σε γλώσσες που έχουν δουλευτεί σε μεγάλο βαθμό προς αυτή την κατεύθυνση εμπόδια όπως οι διαφορετικές διάλεκτοι της γλώσσας ή λέξεις οι οποίες είναι ομόηχες αλλά παρόλα αυτά γράφονται διαφορετικά και έχουν ξεχωριστό νόημα μένει να ξεπεραστούν. Πλέον τα συστήματα του NLP στηρίζονται στην στατιστική και την μηχανική μάθηση (machine learning) ώστε να μπορούν να μαθαίνουν και να προσαρμόζονται αφού η φυσική γλώσσα δεν μπορεί να μπει σε αυστηρά καλούπια όπως μία γλώσσα προγραμματισμού για παράδειγμα [45].

Το έργο OpenNLP είναι μία βιβλιοθήκη η οποία περιέχει εργαλεία για τις υλοποίηση των βασικών μεθόδων ανάλυσης φυσικής γλώσσας. Χρησιμοποιείται για να δομήσει πιο σύνθετες μηχανές ανάλυσης και περιέχει στοιχεία μηχανικής μάθησης. [46]

Περιέχει 93.208 γραμμές κώδικα εκ των οποίων το 99.5% είναι JAVA κώδικας. Οι προγραμματιστές που έχουν συμβάλλει με κώδικα στο έργο είναι 20. Η έκδοση που αναλύθηκε από τα εργαλεία ήταν η 1.9.0 - rc2, από τότε όμως και μέχρι την στιγμή που γράφεται το κείμενο έχουν υπάρξει άλλες δύο εκδόσεις η 1.9.1 - rc1 και η 1.9.1 - rc2, γεγονός που δείχνει ότι το έργο είναι ενεργό και ανανεώνεται διαρκώς. [46]

Maven

Το τι είναι το Maven και ποια η σημασία του για την αρχιτεκτονική ενός java έργου αναφέρθηκε και προηγούμενα. Είναι η προσπάθεια λοιπόν να γίνει το χτίσιμο του έργου εύκολα και ομοιόμορφα. Έτσι υπάρχει κέρδος χρόνου αλλά και ποιότητας στο σύστημα. Το Maven χρησιμοποιείται ευρέως πλέον από την κοινότητα των java προγραμματιστών και αποτελείται από μία δραστήρια κοινότητα.[39]

Το Maven αποτελείται από 106.221 γραμμές κώδικα. Ο JAVA κώδικας αποτελεί το 99.5% του συνολικού κώδικα και σε αυτό το έργο. Συνολικά οι προγραμματιστές που έχουν συνεισφορά

είναι 75 ενώ τελευταία φορά ανέβηκε κώδικας μόλις 2 μέρες πριν από την στιγμή που γράφεται το κείμενο δείχνοντας ότι πρόκειται για ένα ζωντανό έργο. Στην ανάλυση συμπεριλήφθηκε η έκδοση 3.5.4 η οποία υπάρχει από τις 17 Ιουνίου 2018. Στο μεταξύ και μέχρι την σημερινή ημερομηνία έχει υπάρξει και νέα έκδοση η 3.6.0 η οποία δημοσιεύθηκε στις 24 Οκτωβρίου 2018.[47]

WSS4J

Το WSS4J είναι τα αρχικά για το “Web Services Security For Java”. Πρόκειται για μία βιβλιοθήκη με ένα σύνολο υλοποιημένων μεθόδων που αποσκοπούν στην ασφάλεια των υπηρεσιών διαδικτύου (web services). Τα πακέτα ασφαλείας που υλοποιούνται είναι τα εξής [48]

- SOAP Message Security 1.1
- Username Token Profile 1.1
- X.509 Certificate Token Profile 1.1
- SAML Token Profile 1.1
- Kerberos Token Profile 1.1
- SOAP Messages with Attachments Profile 1.1
- Basic Security Profile 1.1 [48]

Στο έργο υπάρχουν 135.742 γραμμές κώδικα. Ολόκληρο το έργο είναι γραμμένο σε JAVA. Είναι το έργο στο οποίο συμμετέχουν οι λιγότεροι προγραμματιστές, μόλις 5 στο σύνολο, ωστόσο παραμένει ένα ενεργό έργο με νέο κώδικα να ανεβαίνει διαρκώς. Στις 12 Ιουνίου 2018 έχει καταγραφεί η τελευταία δημοσίευση έκδοσης του έργου. Αυτή είναι και η έκδοση η οποία έχει συμπεριληφθεί στην μελέτη.

PDFbox

Όπως προδίδει και το όνομα του έργου πρόκειται για ένα εργαλείο το οποίο ασχολείται με τα αρχεία της μορφής PDF. Τα Portable Document Format αρχεία ξεκίνησαν από την Adobe. Πλέον αποτελούν διεθνές πρότυπο αναγνωρισμένο από το ISO (International Organization for Standardization). Τα αρχεία με αυτή την προέκταση έχουν την ίδια μορφή ανεξάρτητα από το περιβάλλον και το πρόγραμμα στο οποίο θα τα ανοίξει ο χρήστης. Έτσι λοιπόν ο δημιουργός τους μπορεί να είναι σίγουρος πως η εικόνα που βλέπει την στιγμή που αποθηκεύεται το αρχείο του είναι και αυτή που θα δει ο τελικός χρήστης. [49]

Το έργο PDFbox της Apache είναι μία εργαλειοθήκη για την δημιουργία και επεξεργασία αυτού του είδους των αρχείων. [50] Αποτελείται από 213.436 γραμμές κώδικα συνολικά και το 99.2% είναι JAVA. Με μόλις 6 προγραμματιστές να γράφουν ενεργά κώδικα στο έργο, η έκδοση που χρησιμοποιήθηκε στην μελέτη είναι η 2.0.9 με ημερομηνία 26 Ιουνίου 2018. Ενώ πλέον η πιο σύγχρονη έκδοση είναι η 2.0.13 η οποία δημοσιεύτηκε στις 28 Νοεμβρίου 2018. [51](

FOP

Το FOP είναι η συντομογραφία για το “Formatting Object Processor”. Διαβάζει έγγραφα τα οποία είναι δομημένα σε XSL και τα αποδίδει σε μία εκτυπώσιμη μορφή όπως για παράδειγμα το PDF που προαναφέρθηκε.[52]

Ένα έγγραφο που είναι δομημένο σε XSL πρέπει να περιέχει όλες τις πληροφορίες σχετικά με την μορφοποίησή του. Τέτοιες πληροφορίες αφορούν το μέγεθος και τα περιθώρια της σελίδας αλλά και μεγέθη γραμματοσειρών και χρώματα. Είναι δηλαδή ένα έγγραφο το οποίο πέρα από το κείμενο το οποίο πρέπει να εκτυπωθεί, περιέχει και ετικέτες (tags) τα οποία περιγράφουν την δομή του. [53]

Το 97.9% από τις συνολικά 292.091 γραμμές κώδικα του έργου, είναι γραμμένες σε JAVA. Οι υπόλοιπες γλώσσες που το απαρτίζουν είναι η XSLT, HTML, JavaScript, Python και Shell. Οι προγραμματιστές που συμμετέχουν με κώδικα είναι και εδώ 7. Η τελευταία έκδοση η οποία είναι και μέρος της μελέτης είναι η 2.3 η οποία δημοσιεύτηκε στις 24 Μαΐου 2018. [54]

Cayenne

Πρόκειται για ένα framework το οποίο ασχολείται με την αντιστοίχιση ενός σχεσιακού μοντέλου με ένα αντικειμενοστραφές, γνωστό και ως ORM (object - relational mapping). Η πρώτη εμφάνιση του έργου ήταν το 2002 ενώ στην οικογένεια της Apache ανήκει από το 2006. [55]

Στον σχεδιασμό μιας εφαρμογής η οποία περιλαμβάνει αποθήκευση δεδομένων σε μία βάση δεδομένων υπάρχουν δύο κυρίαρχες στρατηγικές οι οποίες ακολουθούνται. Η πρώτη είναι το σχεσιακό μοντέλο και η δεύτερη το αντικειμενοστραφές. Αυτά τα μοντέλα έχουν σημαντικές διαφορές μεταξύ τους. Ωστόσο υπάρχουν περιπτώσεις που φαίνεται ότι πρέπει να συνδυαστούν στην ίδια εφαρμογή. Αυτό σημαίνει πως αυτές οι διαφορές πρέπει να επιλυθούν και να γεφυρωθεί το χάσμα μεταξύ των δύο μοντέλων, ώστε η εφαρμογή να λειτουργεί ομαλά και χωρίς ο χρήστης να καταλαβαίνει διαφορά και το πότε γίνεται η μετάβαση. Η γεφύρωση αυτή των διαφορών είναι αυτό που ονομάζεται ORM[56].

Στο cayenne υπάρχουν 348.242 γραμμές κώδικα και το 99.7% από αυτές είναι γραμμένες σε JAVA. Συνολικά 29 προγραμματιστές έχουν συνεισφορά στο αποθετήριο του έργου. Η έκδοση η οποία έχει αναλυθεί είναι η 3.1.2. Πλέον έχει φτάσει στην έκδοση 4.0.1 η οποία δημοσιεύτηκε στις 20 Δεκεμβρίου 2018. [57]

Jclouds

Το jcoulds είναι μία βιβλιοθήκη ανοιχτού κώδικα η οποία αφορά cloud εφαρμογές γραμμένες σε java. Ο όρος cloud γενικά αναφέρεται στην δυνατότητα που έχει ο χρήστης να αξιοποιήσει πόρους είτε σε επίπεδο λογισμικού, είτε σε επίπεδο υλικού τους οποίους δεν διαθέτει άμεσα αυτός αλλά υπάρχουν διαθέσιμοι στο δίκτυο στο οποίο ανήκει. Έτσι λοιπόν cloud μπορεί να είναι ο αποθηκευτικός χώρος με υπηρεσίες όπως το Google Drive, το SharePoint, το DropBox και άλλα. Μπορεί όμως cloud να είναι και μία εφαρμογή η οποία δεν είναι εγκατεστημένη στον υπολογιστή του χρήστη και είναι προσβάσιμη μέσω του διαδικτύου. Ή ακόμα και να βρίσκεται η εφαρμογή ως εκτελέσιμο πρόγραμμα στον εν λόγω υπολογιστή, υπάρχει η δυνατότητα να αξιοποιεί υπηρεσίες για την ανάγνωση ή την αποθήκευση δεδομένων που υπάρχουν στο δίκτυο[58].

Σκοπός του jclouds λοιπόν είναι να μπορεί μία εφαρμογή java να συνδεθεί με τους πιο διαδεδομένους προμηθευτές υπηρεσιών cloud. Η εφαρμογή έτσι φιλοξενείται στους cloud servers αλλά και χρησιμοποιεί τους πόρους αποθήκευσης δεδομένων τους. Επίσης διαθέτει μηχανισμούς κατανομής φόρτου εργασίας μεταξύ των κόμβων του cloud (load balance).[59]

Το jclouds είναι το μεγαλύτερο από τα έργα που επιλέχθηκαν προς ανάλυση και φτάνει τις 482330 γραμμές κώδικα. Το 99,4% από αυτές πρόκειται για κώδικα σε JAVA. Ταυτόχρονα έχει μακράν τους περισσότερους προγραμματιστές που συμμετέχουν στο αποθετήριό του, φτάνοντας τους 214. Η τελευταία έκδοση του λογισμικού έχει ημερομηνία στις 03/02/2019 ωστόσο η έκδοση η οποία χρησιμοποιήθηκε είναι η 2.1.0 από τον Φεβρουάριο του 2018.[60]

OpenJPA

Το openJPA είναι ακόμα ένα project το οποίο βοηθάει τον προγραμματιστή να ορίσει τις συσχετίσεις μεταξύ ενός σχεσιακού μοντέλου και της αντικειμενοστραφούς απεικόνισης του σε POJO κλάσεις JAVA [61]. Το JPA (Java Persistence API) περιέχει μηχανισμούς αντιστοίχισης των πινάκων με τις μεταβλητές μέσα σε μία κλάση, ενώ ταυτόχρονα μπορεί να προσδιορίσει διαφορετικές σχέσεις όπως αυτές του πρωτεύοντος ή του ξένου κλειδιού, σχέσεις ένα προς πολλά, ένα προς ένα και πολλά προς πολλά. Επίσης διαχειρίζεται τις συναλλαγές με την βάση καθώς και την χρήση της cache κατά την πραγματοποίησή τους[43].

Το openJPA είναι μία υλοποίηση για το JPA ανοιχτού κώδικα. Αποτελείται από 571.702 γραμμές κώδικα, το 99.2% εκ των οποίων είναι γραμμένες σε JAVA. Σύμφωνα με το αποθετήριό του στο GitHub μόλις 7 είναι οι προγραμματιστές που έχουν συμμετάσχει στην συγγραφή του. Ενώ η τελευταία του έκδοση με αριθμό 3.0.0 είναι διαθέσιμη από τις 14 Ιουνίου 2018.[61]

Επιλογή των εργαλείων ανάλυσης κώδικα

Με μία σύντομη έρευνα στο διαδίκτυο κανείς μπορεί να καταλάβει πως υπάρχει μία ποικιλία εργαλείων τα οποία αναλύουν την ποιότητα του κώδικα ενός έργου και προσφέρουν μετρικές όπως το μέγεθος του έργου, η πολυπλοκότητα ή τα code smells. Κάποια από αυτά τα εργαλεία

στοχεύουν σε έργα που είναι γραμμένα σε συγκεκριμένη γλώσσα προγραμματισμού όπως JAVA, Python ή C#, ενώ άλλα έχουν την ικανότητα να αναλύουν έργα υποστηρίζοντας πολλές διαφορετικές γλώσσες ταυτόχρονα. Οι διαθέσιμες γλώσσες προς ανάλυση λοιπόν μπορούν να αποτελέσουν το πρώτο κριτήριο επιλογής ενός εργαλείου. Στην περίπτωση της συγκεκριμένης μελέτης το κάθε εργαλείο πρέπει αδιαμφισβήτητα να μπορεί να αναλύει έργα γραμμένα σε JAVA καθώς όπως αναφέρθηκε και προηγούμενα τα έργα που επιλέχθηκαν αποτελούνται από JAVA σε ποσοστό τουλάχιστον 97%.

Η επόμενη κατηγοριοποίηση η οποία μπορεί να γίνει στα εργαλεία αφορά την διαθεσιμότητά τους. Κάποια από αυτά είναι ανοιχτού κώδικα και δωρεάν. Κάποια άλλα είναι εμπορικά και πρέπει να αγοραστεί άδεια για την χρήση τους. Στα πλαίσια της μελέτης αυτής λοιπόν τα εργαλεία που μπορούν να αξιοποιηθούν είναι αρχικά αυτά του ανοιχτού κώδικα στα οποία υπάρχει δωρεάν πρόσβαση. Από τα υπόλοιπα εμπορικά προϊόντα μπορούν να συμπεριληφθούν αυτά για τα οποία οι εταιρείες θα παραχωρήσουν την σχετική άδεια για ακαδημαϊκή χρήση.

Τέλος από τα σημαντικότερα κριτήρια αποτελούν οι μετρικές που δίνει κάθε εργαλείο ως αποτέλεσμα της ανάλυσης. Η μελέτη αυτή επικεντρώνεται στην μέτρηση του τεχνικού χρέους. Τα εργαλεία λοιπόν που θα χρησιμοποιηθούν πρέπει να αποδίδουν με σαφήνεια την μετρική αποτίμησής του. Τα περισσότερα εργαλεία μετράνε το σύνολο των προβλημάτων που εντοπίζουν σε ένα έργο ή σε μία κλάση. Μπορούνε να παρουσιάσουν για παράδειγμα το πόσες φορές εμφανίζεται επανάληψη κώδικα και απαιτείται εξαγωγή μεθόδου, ή το ποιες κλάσεις έχουν μεγάλη πολυπλοκότητα. Μία στάθμιση αυτών των προβλημάτων θα μπορούσε να οδηγήσει στην αποτίμηση του τεχνικού χρέους κάθε μονάδας από κάθε εργαλείο. Στα πλαίσια όμως αυτής της μελέτης το ενδιαφέρον στρέφεται στο πως αυτή η στάθμιση γίνεται από το ίδιο το εργαλείο χωρίς να χρειαστεί εξωτερική παρέμβαση. Άρα λοιπόν η λίστα των εργαλείων περιορίζεται μόνο σε όσα από αυτά προσφέρουν ρητά αυτή την στάθμιση και τον υπολογισμό της μετρικής μετά την ανάλυσή τους.

Λαμβάνοντας λοιπόν τα παραπάνω κριτήρια η επιλογή των εργαλείων που χρησιμοποιήθηκαν για την ανάλυση περιλαμβάνει τα παρακάτω.

Table 2 – Εργαλεία ανάλυσης κώδικα που χρησιμοποιήθηκαν

Όνομα Εργαλείου	Εταιρεία	Είδος Άδειας
SonarQube	SonarSource	Ανοιχτού κώδικα
Squore	Squoring Technologies	Ακαδημαϊκή χρήση
CAST	CAST	Ακαδημαϊκή χρήση

Επεξεργασία δεδομένων

Κάθε έργο λογισμικού αναλύεται τρεις φορές. Μία για κάθε διαθέσιμο εργαλείο. Σαν είσοδος σε όλα τα εργαλεία χρησιμοποιείται ο πηγαίος κώδικας του έργου που ήταν διαθέσιμος στα αποθετήριά του. Στα εργαλεία χρησιμοποιήθηκαν οι κανόνες που είναι διαθέσιμοι με την βασική εγκατάστασή τους. Δεν τροποποιήθηκαν οι κανόνες με κανέναν τρόπο ακόμα και στα εργαλεία που δίνουν τέτοια δυνατότητα. Αφού ολοκληρώθηκε η ανάλυση προέκυψε μία νέα σειρά προβλημάτων τα οποία έπρεπε να επιλυθούν πριν συνεχίσει η διαδικασία της επεξεργασίας.

Όγκος Δεδομένων

Αρχικά ο όγκος των δεδομένων ήταν τέτοιος ο οποίος δεν επέτρεπε την χειροκίνητη συλλογή τους. Έπρεπε η διαδικασία να αυτοματοποιηθεί. Για αυτό τον λόγο δημιουργήθηκε ένα εργαλείο γραμμένο σε JAVA το οποίο παίζει τον ρόλο του διαμεσολαβητή προκειμένου να συγκεντρώσει τα στοιχεία από τις διαφορετικές πηγές και να κάνει μία πρώτη επεξεργασία ώστε να τα φέρει σε μορφή κατάλληλη για να συνεχίσει η ανάλυσή τους. Αφού έχει περάσει από αρκετά διαφορετικά στάδια πλέον το εργαλείο αυτό μπορεί να διαβάσει τα σύνολα δεδομένων κάθε εργαλείου, να ξεχωρίσει και να υπολογίσει όπου χρειάζεται τις μετρικές που ενδιαφέρουν την έρευνα και να αποδώσει το dataset των αποτελεσμάτων σε κοινή μορφή είτε για κάθε εργαλείο ξεχωριστά, είτε για το σύνολο των εργαλείων σε κοινό αρχείο.

Μορφή Δεδομένων

Ο τρόπος που κάθε εργαλείο εξάγει τα δεδομένα του είναι διαφορετικός. Αυτό αποτέλεσε ακόμα ένα πρόβλημα το οποίο έπρεπε να λυθεί προκειμένου εν τέλη τα δεδομένα να έχουν κοινή μορφή όταν θα αποτελούσαν είσοδο για την στατιστική ανάλυση.

SonarQube

Το SonarQube προσφέρει ένα WEB API [62] στον χρήστη. Με το κατάλληλο query μπορούν να συγκεντρωθούν τα αποτελέσματα σε μορφή JSON. Τα αποτελέσματα των μετρικών δίνονται ομαδοποιημένα ανά αρχείο. Η προσέγγιση αυτή ταιριάζει με την προσέγγιση της έρευνας στην οποία αποφασίστηκε να έχουμε ως μονάδες προς ανάλυση τα αρχεία κλάσεων κάθε έργου. Από το πλήθος των διαθέσιμων μετρικών αυτές που βρέθηκαν στο επίκεντρο ήταν το πλήθος των προβλημάτων (issues) που εντοπίζονται σε κάθε κλάση καθώς και ο δείκτης SQALE. Τα προβλήματα στο SonarQube χωρίζονται σε πέντε κατηγορίες.

- Blocker: Το μέγιστο επίπεδο σοβαρότητας. Περιλαμβάνει bugs τα οποία μπορεί να επηρεάσουν σοβαρά την λειτουργία του λογισμικού κατά την χρήση του.

- Critical: Σφάλματα με μικρότερη πιθανότητα εμφάνισης από αυτά της προηγούμενης κατηγορίας ή ευπάθειες (vulnerabilities).

- Major: Εδώ πλέον κατά κύριο λόγο αναφέρονται τα code smells. Αυτά με την σειρά τους έχουν μία κλίμακα σοβαρότητας, οπότε εδώ περιέχονται όσα θεωρούνται ότι έχουν το μεγαλύτερο αντίκτυπο.

- Minor: Στοιχεία τα οποία έχουν μικρότερη επιρροή στον χρόνο που χρειάζεται ο προγραμματιστής.

- Info: Περισσότερο πληροφοριακού χαρακτήρα. Έχουν από ελάχιστη ως μηδενική συνεισφορά στο τεχνικό χρέος.[63]

Παίρνοντας το άθροισμα λοιπόν αυτών των μετρικών υπολογίστηκε ο συνολικός αριθμός προβλημάτων σε κάθε κλάση (total issues).

Σχετικά με την συντηρησιμότητα (maintainability) του κώδικα προσφέρονται συνολικά επτά μετρικές.

- Code smells
- New code smells
- Maintainability rating
- Technical debt
- Technical debt on new code
- Technical debt ratio
- Technical debt ratio on new code

Από αυτές λοιπόν επιλέχθηκε το Technical Debt ή αλλιώς SQALE Index ως η καταλληλότερη μετρική καθώς είναι αυτή η οποία μετράει την προσπάθεια που χρειάζεται προκειμένου να διορθωθούν όλα τα κομμάτια κώδικα που προκαλούν το τεχνικό χρέος. Η μονάδα μέτρησης είναι τα λεπτά.[64]

Ιδιαίτερα χρήσιμη φάνηκε η δυνατότητα του API να μπορεί να ξεχωρίζει τα αρχεία σε αυτά που περιέχουν τον πραγματικό κορμό του έργου από αυτά που είναι αρχεία για test ή αρχεία properties. Καθώς είναι και το μοναδικό εργαλείο από τα τρία το οποίο περιλαμβάνει τέτοια λειτουργικότητα χρησιμοποιήθηκε ως σημείο αναφοράς προκειμένου να εξαλειφθούν αυτού του είδους τα αρχεία και από τα σύνολα αποτελεσμάτων των άλλων δύο εργαλείων.

Squore

Το Squore δίνει μεγαλύτερη έμφαση στην διαγραμματική απεικόνιση των αποτελεσμάτων. Παρόλα αυτά και σε αυτό το εργαλείο υπήρχε η δυνατότητα τα αποτελέσματα να ομαδοποιηθούν ανά αρχείο. Το εξαγόμενο αρχείο είναι της μορφής csv. Συνεπώς εδώ έπρεπε να αναπτυχθεί άλλος μηχανισμός τον οποίο θα υποστήριζε το εργαλείο προκειμένου να

διαβαστούν τα δεδομένα από το csv αρχείο και να παραχθούν τα αντίστοιχα XML. Όπως και πριν και εδώ περιλαμβάνουν το path του αρχείου και το σύνολο των προβλημάτων που εμφανίζονται σε αυτά συμπεριλαμβανομένων όλων των επιπέδων σημαντικότητας που προσφέρει το Squore (Blocker, Critical, Major και Minor). [65]

Αναφορικά με το τεχνικό χρέος το εργαλείο προσφέρει την μετρική “Technical Debt”. Εδώ το χρέος εκφράζεται σε ανθρωποώρες (man hours). Κάθε 8 ώρες θεωρείται ότι συμπληρώνεται μία ανθρωπομέρα. Για να μπορέσει το αποτέλεσμα εδώ να αξιοποιηθεί σε σύγκριση με το προηγούμενο έπρεπε να αναφέρεται στην ίδια μονάδα μέτρησης. Για αυτό λοιπόν έγινε κανονικοποίηση του μεγέθους και έτσι το τεχνικό χρέος πλέον εκφράστηκε και σε αυτή την περίπτωση σε λεπτά.

Εφ’ όσον εδώ δεν προκύπτει εμφανώς η δυνατότητα να εξαιρεθούν οι κλάσεις που αφορούν test και properties έπρεπε να γίνει εκ των υστέρων φιλτράρισμα των αρχείων που συμπεριλαμβάνονται στα αποτελέσματα. Το φιλτράρισμα αυτό έγινε χρησιμοποιώντας ως σημείο αναφοράς την λίστα των αρχείων που είχε προκύψει από το αντίστοιχο ερώτημα στο API του SonarQube. Εάν κάποιο αρχείο δεν συμπεριλαμβάνεται σε αυτή την αρχική λίστα θα έπρεπε να εξαιρεθεί και από την λίστα αποτελεσμάτων του Squore. Στο σημείο αυτό πρέπει να τονιστεί πως το φιλτράρισμα που αναφέρεται μέσω του API αφορά μόνο το είδος του αρχείου και όχι την πιθανότητα εμφάνισης προβληματικών μετρικών. Έτσι ακόμα και κλάσεις οι οποίες ανήκουν στον βασικό κορμό του κώδικα αλλά στο SonarQube εμφανίζουν μηδενικά προβλήματα συμμετέχουν στο σύνολο ώστε να μπορούν να τις αποδοθούν τιμές και από τα υπόλοιπα δύο εργαλεία της ανάλυσης.

CAST

Το CAST ήταν το πιο απαιτητικό εργαλείο από τα τρία προκειμένου να ευθυγραμμιστούν τα δεδομένα με τα υπόλοιπα. Φυσικά όμως δεν μπορεί να κατηγορηθεί για αυτό, μιας και οφείλεται στην διαφορετική φιλοσοφία του εργαλείου και στην αντιμετώπιση του τεχνικού χρέους σε επίπεδο έργου και όχι σε επίπεδο κλάσης.

Με την βοήθεια και την συνεργασία της ομάδας της CAST ωστόσο κατέστη δυνατό να ανακτηθούν τα αποτελέσματα όπως έπρεπε προκειμένου να υπάρχει ομοιομορφία. Το εργαλείο αυτό χρησιμοποιεί μία ενδιάμεση τοπική βάση στην οποία αποθηκεύει τα αποτελέσματα της ανάλυσης κώδικα που κάνει. Μπορεί λοιπόν στους πίνακες ελέγχου του να εμφανίζει συνολικές μετρικές σε επίπεδο έργου ή module, ωστόσο στην βάση τα αποτελέσματα είναι κρατημένα τόσο ανά αρχείο όσο και ανά είδος προβλήματος. Η ομάδα της CAST λοιπόν παρείχε το κατάλληλο SQL ερώτημα που έπρεπε να γίνει στην βάση ώστε για τις ανάγκες της έρευνας να εξάγουμε το τεχνικό χρέος για κάθε πρόβλημα που εμφανίζεται σε κάθε κλάση. Αυτά τα αποτελέσματα αποθηκεύτηκαν σε csv αρχεία και πλέον το εργαλείο που αναπτύχθηκε, με έναν παρόμοιο μηχανισμό με πριν, διαβάζει τα δεδομένα από τα csv αρχεία και αφού τα ομαδοποιήσει ανά κλάση παράγει και εδώ τα αντίστοιχα XML αρχεία.

Η μετρική του συνολικού πλήθους των προβλημάτων υπάρχει και στο CAST. Σε αυτή την περίπτωση ωστόσο το ερώτημα που έγινε στην βάση δεδομένων επιστρέφει τον συνολικό αριθμό χωρίς να ξεχωρίζει επίπεδα σημαντικότητας. Αυτό δεν δημιουργεί ωστόσο πρόβλημα στην μελέτη καθώς δεν εξετάζεται σε αυτό το στάδιο η κατανομή των σφαλμάτων αλλά μόνο το τεχνικό χρέος που αυτά δημιουργούν.

Όπως αναφέρθηκε και πριν στην περίπτωση του Squore, και εδώ δεν προκύπτει σαφώς η διάκριση του τύπου των αρχείων ώστε να μην ληφθούν υπόψη τα αρχεία ιδιοτήτων και τα αρχεία ελέγχου. Οπότε ακολουθήθηκε πάλι η διαδικασία φιλτραρίσματος των αρχείων χρησιμοποιώντας το σύνολο των αρχείων του SonarQube.

Σύγκριση αποτελεσμάτων εργαλείων

Πλέον αφού υπάρχουν τα αποτελέσματα και των τριών εργαλείων εκφρασμένα σε κοινή μορφή μπορεί να πραγματοποιηθεί και η σύγκριση μεταξύ τους. Πριν προχωρήσουμε στην στατιστική ανάλυση των δεδομένων έπρεπε να γίνει μία πρώτη διερεύνηση για να δούμε εάν όντως υπάρχει κάποια σύγκλιση των αποτελεσμάτων των εργαλείων η οποία άξιζε να της αφιερωθεί περισσότερος χρόνος.

Αρχικά επιλέχθηκαν τέσσερις βαθμίδες σύγκρισης. Για κάθε εργαλείο επιλέχθηκε κάθε φορά το 10%, 20%, 30% και 40% των πιο προβληματικών κλάσεων. Από τα σύνολα τα οποία προέκυψαν εξετάστηκε ποιες ήταν οι κλάσεις οι οποίες εμφανίζονταν και στα τρία (ένα για κάθε εργαλείο ανά ποσοστό). Σε αυτό το στάδιο δεν εξετάζεται το ranking των κλάσεων μέσα στα σύνολα παρά μόνο η ύπαρξή τους μέσα σε αυτά. Τα αποτελέσματα εμφανίζονται στον παρακάτω πίνακα.

Table 3 – Αριθμός κοινών κλάσεων με τα περισσότερα προβλήματα και το υψηλότερο τεχνικό χρέος σε διαφορετικά κατώφλια

	10%		20%		30%		40%	
	mina							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	19	22	38	44	57	66	76	88
Squore Classes	6	28	13	56	20	84	27	112
Cast Classes	15	45	31	90	46	136	62	181
Total Classes	0	8	0	21	3	33	10	44
	deltaspike							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	24	28	49	56	73	84	98	112
Squore Classes	9	39	19	78	28	117	38	156
Cast Classes	13	81	26	162	39	244	52	325
Total Classes	0	13	0	34	0	49	1	71
	opennlp							
	Total	TD	Total	TD	Total	TD	Total	TD

	Issues		Issues		Issues		Issues	
Sonar Classes	30	34	60	69	91	103	121	138
Squore Classes	7	42	15	85	22	128	30	171
Cast Classes	26	68	53	136	79	204	106	272
Total Classes	0	24	8	48	10	64	14	96
	maven							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	29	35	59	71	89	107	119	143
Squore Classes	7	30	14	60	21	90	29	120
Cast Classes	15	64	30	128	46	192	61	256
Total Classes	2	18	2	33	2	57	8	78
	wss4j							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	25	35	51	70	77	105	102	141
Squore Classes	13	34	26	68	39	102	52	136
Cast Classes	6	50	13	100	20	150	27	200
Total Classes	1	17	5	41	5	74	6	108
	pdfbox							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	38	43	76	87	114	131	152	175

Squore Classes	15	74	30	148	45	222	61	296
Cast Classes	33	100	66	201	99	302	133	402
Total Classes	0	17	4	42	6	78	9	121
	fop							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	74	80	149	161	223	242	298	322
Squore Classes	19	112	39	225	58	338	78	451
Cast Classes	64	156	128	313	192	470	257	627
Total Classes	0	47	3	98	5	168	9	226
	cayenne							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	66	83	132	166	199	249	265	332
Squore Classes	31	109	62	218	93	328	124	437
Cast Classes	50	162	100	324	151	487	201	649
Total Classes	1	22	3	78	13	141	29	221
	jclouds							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	128	147	256	294	384	442	512	589
Squore Classes	27	191	54	382	81	573	109	764
Cast Classes	89	305	178	611	267	916	356	1222

Total Classes	1	55	6	195	11	299	22	415
	openjpa							
	Total Issues	TD	Total Issues	TD	Total Issues	TD	Total Issues	TD
Sonar Classes	78	96	157	193	235	290	314	386
Squore Classes	32	102	64	205	97	307	129	410
Cast Classes	69	137	139	275	208	413	278	550
Total Classes	3	50	13	112	32	187	47	274

Επειδή το κάθε εργαλείο ακολουθεί την δική του μεθοδολογία αποτίμησης έπρεπε να διασφαλιστεί πως τα μεγέθη τα οποία θα λάμβαναν μέρος στην σύγκριση θα είναι όμοια προκειμένου η διαδικασία να έχει νόημα. Έτσι η σύγκριση έγινε σε δύο άξονες. Ο πρώτος ήταν το πλήθος των σφαλμάτων που εντοπίζονται σε κάθε κλάση. Είναι ενδιαφέρον αρχικά το γεγονός ότι κάθε εργαλείο εντοπίζει διαφορετικό αριθμό σφαλμάτων, ενδεικτικό του ότι χρησιμοποιείται διαφορετικό σύνολο κανόνων σε κάθε υλοποίηση. Ωστόσο η συμφωνία των εργαλείων στην τομή τους εάν ληφθεί υπόψη αυτή η παράμετρος είναι πολύ μικρή και πολλές φορές δεν υπάρχει κυρίως όσο το ποσοστό μειώνεται. Σε αρκετές περιπτώσεις όταν η σύγκριση γίνεται για το 10% των πιο προβληματικών κλάσεων η τομή είναι μηδενική. Στο έργο fop για παράδειγμα οι προβληματικές κλάσεις του SonarQube είναι 74, του Squore μόλις 19, ενώ του CAST 64. Η τομή όμως των εργαλείων είναι μηδέν. Αυτό σημαίνει πως συνολικά τα εργαλεία εντόπισαν προβλήματα σε 157 κλάσεις που καμία δεν είναι ίδια με την άλλη. Υπάρχουν μάλιστα περιπτώσεις έργων όπως το mina και το deltaspike τα οποία ούτε στο 20% της σύγκρισης εμφανίζουν συμφωνία στην τομή.

Ο δεύτερος άξονας με βάση τον οποίο έγινε η διερεύνηση είναι πιο υποσχόμενος. Ουσιαστικά πλέον εδώ έχει μετατραπεί το τεχνικό χρέος σε ανθρωποώρες με βάση την εκτίμηση του κάθε εργαλείου. Η κατάταξη των κλάσεων ως πως το πόσο προβληματικές είναι ανάλογη του χρόνου επιδιόρθωσης που απαιτείται. Σταθερά και σε όλα τα επίπεδα ποσοστών συμφωνίας

που εξετάστηκαν εμφανίζονται περισσότερα κοινά σημεία από ότι στην πρώτη μεθοδολογία. Δεν εμφανίζεται μηδενική τομή σε καμία περίπτωση.

Το πλήθος των προβληματικών κλάσεων οι οποίες συμμετέχουν στην τομή έχει την δική του σημασία. Όλες αυτές οι κλάσεις θα αποτελέσουν την βάση benchmarking. Όσο λοιπόν περισσότερες είναι τόσο μεγαλύτερη ακρίβεια θα έχει και το τελικό benchmark. Και σε επόμενο βήμα αυτές οι κλάσεις μπορούν να αποτελέσουν τα δεδομένα εισόδου σε ένα στατιστικό μοντέλο ή σε ένα μοντέλο μηχανικής μάθησης.

Πέρα από το μέγεθος όμως της τομής ενδιαφέρον έχει και η κατάταξη των κλάσεων για κάθε εργαλείο. Από αυτή την σύγκριση μπορεί να προκύψουν ερωτήματα του γιατί ένα εργαλείο θεωρεί πως μία κλάση θέλει περισσότερο χρόνο να διορθωθεί από ότι κάποιο άλλο. Αν η διαφοροποίηση έχει να κάνει με το σύνολο κανόνων που χρησιμοποιούνται τότε πριν την απόφαση για το ποιο εργαλείο θα πρέπει να χρησιμοποιήσει κανείς θα πρέπει να ερευνά τους κανόνες που αυτό υλοποιεί. Εναλλακτικά και εφόσον δίνεται η δυνατότητα από το εργαλείο θα πρέπει ο χρήστης να ορίζει το κατάλληλο σύνολο κανόνων που αντιπροσωπεύουν καλύτερα την φύση του έργου που αναλύεται.

Στον παρακάτω πίνακα φαίνεται ενδεικτικά η πιο προβληματική κλάση κάθε έργου, όπως αυτή εντοπίζεται από κάθε εργαλείο.

Table 4 – Η κλάση με το υψηλότερο τεχνικό χρέος σε κάθε έργο

	Sonar	Squore	Cast
mina	ObjectMBean.java	ObjectMBean.java	AbstractPollingIoProcessor.java
deltaspike	CdiTestRunner.java	Annotateds.java	CdiTestRunner.java
opennlp	turkishStemmer.java	turkishStemmer.java	ParserCrossValidator.java
maven	ModelMerger.java	ModelMerger.java	ModelMerger.java
wss4j	Merlin.java	WSHandler.java	SP12Constants.java

pdfbox	COSParser.java	COSParser.java	COSParser.java
fop	GlyphPositioningTable.java	GlyphPositioningTable.java	OTFAdvancedTypographicTableReader.java
cayenne	ExpressionParser.java	ExpressionParserTokenManager.java	SimpleNode.java
jclouds	EC2HardwareBuilder.java	ShellToken.java	Server.java
openjpa	JavaParser.java	DBDictionary.java	JavaParser.java

Στατιστική ανάλυση

Γενική μεθοδολογία

Kippendorff's Alpha - Reliability

Ο συντελεστής Alpha (α) του Kippendorff [66] είναι ένας συντελεστής αξιοπιστίας που μπορεί να αξιολογήσει την συμφωνία μεταξύ των παρατηρητών. Στα πλαίσια της τρέχουσας ανάλυσης ως παρατηρητές ορίζονται τα τρία εργαλεία ανάλυσης κώδικα που χρησιμοποιήθηκαν, των οποίων την τομή επιθυμούμε να εξετάσουμε προκειμένου να βρεθεί ο βαθμός συμφωνίας των αποτελεσμάτων τους.

Ο γενικός τύπος του συντελεστή α είναι ο εξής

$$\alpha = 1 - \frac{D_o}{D_e}$$

Όπου Do είναι η ασυμφωνία μεταξύ των τιμών των παρατηρήσεων και De είναι η αναμενόμενη ασυμφωνία όταν μεταβληθεί η κωδικοποίηση των παρατηρήσεων. Όταν έχουμε πλήρη συμφωνία των παρατηρήσεων το Do είναι ίσο με μηδέν και συνεπώς ο συντελεστής α ισούται με την μονάδα. Αν όμως οι παρατηρητές θεωρούν πως τα αποτελέσματα ήταν θέμα τύχης και δεν μπορούν να ξεχωρίσουν αν αυτό που εξετάζουν είναι πραγματικά δεδομένα ή τιμές που προέρχονται από επιλογή ενός τυχαίου συνόλου τότε ισχύει ότι

$$D_o = D_e \; και \; συνεπώς \; \alpha = 0$$

Η μέθοδος αυτή επιλέχθηκε γιατί μπορεί να αξιολογήσει διαφορετικά είδη δεδομένων με βάση τα ίδια αξιώματα αξιοπιστίας. Δεν επηρεάζεται από τον αριθμό των παρατηρητών, το πλήθος των παρατηρήσεων, την έλλειψη τιμών και τις μονάδες μέτρησης. Αυτά τα δεδομένα είναι πολύ σημαντικά για την τρέχουσα μελέτη για πολλούς λόγους. Αρχικά η μη δέσμευση στον αριθμό των παρατηρητών που εδώ είναι τα προγράμματα μέτρησης τεχνικού χρέους δίνει την δυνατότητα επανάληψης της μελέτης συμπεριλαμβάνοντας νέα και περισσότερα εργαλεία για μελλοντική ενίσχυση των αποτελεσμάτων. Επίσης η αποδέσμευση από το πλήθος των παρατηρήσεων επιτρέπει να εφαρμοστεί η ίδια ανάλυση και σε διαφορετικά έργα λογισμικού με διαφορετικό αριθμό κλάσεων, μεγαλύτερο ή μικρότερο των όσων χρησιμοποιήθηκαν εδώ. Ιδιαίτερα σημαντικό είναι το ότι η μέθοδος μπορεί να εφαρμοστεί ακόμα και αν κάποιες από τις παρατηρήσεις δεν έχουν τιμή. Από την φύση των δεδομένων προς ανάλυση, προκύπτει ότι μία κλάση που μπορεί να είναι προβληματική για ένα εργαλείο και να εμφανίζει ένα ποσό τεχνικού χρέους για κάποιο άλλο υπάρχει η πιθανότητα να μην θεωρηθεί προβληματική και άρα δεν θα υπάρχει ανάλογη εκτίμηση. Για το εργαλείο στο οποίο η κλάση λοιπόν δεν εμφανίζεται θα υπάρχει έλλειψη στην μέτρηση του τεχνικού χρέους.

Περιγράφοντας γενικά την μεθοδολογία αρκεί να γίνει αναφορά στην περίπτωση που υπάρχουν πολλοί παρατηρητές (περισσότεροι από δύο), γίνεται ανάλυση οποιασδήποτε μετρικής και υπάρχουν ελλιπή δεδομένα στην ανάλυση. Υπάρχουν τέσσερα βασικά βήματα που πρέπει να ακολουθήσει ο ερευνητής για να υπολογίζει τον συντελεστή α.

Στο πρώτο βήμα πρέπει να δημιουργηθεί η μήτρα αξιοπιστίας (reliability data matrix). Κάθε γραμμή αντιπροσωπεύει έναν παρατηρητή ενώ κάθε στήλη μία μετρική. Σε κάθε κελί της μήτρας εμφανίζεται η τιμή που έχει αυτή η μετρική για τον κάθε παρατηρητή. Για κάθε μετρική πρέπει να υπολογιστεί και η τιμή mu που αντιστοιχεί στο πλήθος των παρατηρήσεων που υπάρχουν για την συγκεκριμένη μετρική. Εάν όλοι ο παρατηρητές έχουν αποτυπώσει τιμή για την μετρική τότε το mu ισούται με το πλήθος των παρατηρητών. Εάν για κάποιον παρατηρητή λείπει η τιμή τότε αντίστοιχα μικραίνει και η τιμή του mu.

Στο δεύτερο βήμα δημιουργείται η μήτρα συμπτώσεων (coincidence matrix). Εδώ και στις γραμμές και στις στήλες μπαίνουν οι διαφορετικές τιμές των παρατηρήσεων που εμφανίζονται στα κελιά της προηγούμενης μήτρας. Στην συνέχεια για κάθε συνδυασμό γραμμής και στήλης πρέπει να εντοπιστεί το πλήθος των εμφανίσεων του στην μήτρα αξιοπιστίας. Η συχνότητα εμφάνισης κάθε ζεύγους τιμών c και k υπολογίζεται από τον παρακάτω τύπο

$$O_{ck} = \sum \frac{Number\ of\ c-k\ pairs}{m_u - 1}$$

Αρχικά πρέπει να υπολογιστεί πόσα ζευγάρια μπορεί να δώσει κάθε στήλη σε σχέση με το πόσες τιμές έχει και πόσες της λείπουν. Το πλήθος αυτό υπολογίζεται από τον τύπου

$$m_u * (m_u - 1)$$

Έπειτα υπολογίζεται για κάθε ζεύγος τιμών c και k το πλήθος που εμφανίζεται το συγκεκριμένο ζευγάρι στην κάθε στήλη δίνοντας το ock της κάθε στήλης. Τέλος τα ock που έχουν προκύψει από την εξέταση όλων των στηλών αθροίζονται δίνοντας τις τιμές που μπαίνουν σε κάθε κελί της μήτρας συμπτώσεων. Αθροίζοντας κάθε γραμμή και κάθε στήλη της μήτρας υπολογίζονται τα αντίστοιχα n1, n2, … , nc, …, nk, ...

Το επόμενο βήμα της ανάλυσης είναι η αξιοποίηση της κατάλληλης μεθόδου για τον υπολογισμό των διαφορών δ. Το ποια συνάρτηση θα χρησιμοποιηθεί εδώ για να παραχθεί το αποτέλεσμα εξαρτάται από την φύση των δεδομένων που αναλύονται. Εάν τα δεδομένα είναι δυαδικά η διαδικασία είναι απλή καθώς οι παρατηρητές ή θα συμφωνούν ή θα διαφωνούν. Οι περιπτώσεις όμως που τα δεδομένα διαφέρουν αλγεβρικά ή χρησιμοποιούνται για δεδομένα διαστημάτων και κατάταξης είναι πιο πολύπλοκες και απαιτούν πιο σύνθετες συναρτήσεις ώστε να υπολογιστεί το λεγόμενο metricδ^2_{ck}.

Το τέταρτο και τελευταίο βήμα περιλαμβάνει τον υπολογισμό του συντελεστή α. Αξιοποιώντας τον τύπο

$$a = 1 - \frac{D_o}{D_1} = 1 - (n-1)\frac{\sum_c \sum_{k>c} o_{ck\ metric\ \delta^2_{ck}}}{\sum_c \sum_{k>c} n_c n_{k\ metric}\ \delta^2_{ck}}$$

Ένας εναλλακτικός τρόπος υπολογισμού της μεθόδους παραλείπει το βήμα 2 και στην θέση της μήτρας συμπτώσεων εμφανίζεται η μήτρα μετρικών και τιμών. Η γραμμή κάθε μήτρας περιλαμβάνει τις πιθανές τιμές με τις οποίες εμφανίζεται κάθε μετρική. Η στήλη της μήτρας πάλι εμφανίζει τις ίδιες τις μετρικές. Τα κελιά της μήτρας περιέχουν το πλήθος των εμφανίσεων κάθε τιμής σε κάθε μετρική. Εάν κάποια τιμή δεν εμφανίζεται καμία φορά για κάποια μετρική τότε την θέση της παίρνει το μηδέν.

Η μορφή του τελικού τύπου υπολογισμού αλλάζει και είναι η εξής:

$$a = 1 - \frac{D_o}{D_1} = 1 - (n-1)\ \frac{\sum_u \frac{1}{n_u - 1}\ \sum_c \sum_{k>c} n_{uc} n_{uk\ metric}\ \delta_{ck}^2}{\sum_c \sum_{k>c} n_c n_{k\ metric}\ \delta_{ck}^2}$$

Kendall's W coefficient of concordance

Η μέθοδος αυτή παρουσιάστηκε από τον Kendall το 1948. Χρησιμοποιείται για να μετρήσει την συμφωνία μεταξύ ποσοτικών ή ημιποσοτικών μετρικών ενός συνόλου αντικειμένων. Ο δείκτης που προκύπτει συμβολίζεται με το γράμμα W και οι τιμές του κυμαίνονται από το μηδέν που σημαίνει πλήρης ασυμφωνία, ως την μονάδα που δηλώνει πλήρης συμφωνία. [67]

Οι τύποι που χρησιμοποιούνται για τον υπολογισμό του δείκτη αποτελούνται από 2 διαφορετικά ζεύγη παρουσιάζονται παρακάτω.

Ζεύγος 1.

$$S = \sum_{i=1}^{n} (R_i - \bar{R})^2$$

$$W = \frac{12S}{m^2(n^3 - n) - mT}$$

Ζεύγος 2

$$S' = \sum_{i=1}^{n} R_i^2 = SSR$$

$$W' = \frac{12S' - 3m^2n(n+1)^2}{m^2(n^3 - n) - mT}$$

Όπου ισχύουν τα εξής

R είναι ο μέσος όρος των τιμών των γραμμών Ri

Ri είναι η τιμή κάθε γραμμής

S είναι το άθροισμα των τετραγώνων των διαφορών

n είναι ο αριθμός των αντικειμένων

m είναι ο αριθμός των τιμών

T είναι ο συντελεστής διόρθωσης

Ο δείκτης αυτός τελικά υπολογίζει τον λόγο της υπάρχουσας συνολικής διακύμανσης ως προς την μέγιστη δυνατή διακύμανση για το συγκεκριμένο σύνολο δεδομένων. Ο έλεγχος σημαντικότητας που ακολουθεί την μεθοδολογία είναι ο x^2 με (n-1) βαθμούς ελευθερίας και ο τύπος που εφαρμόζεται είναι ο

$$x^2 = m(n-1)W$$

Ανάλυση Αρχετύπων (Archetypal Analysis)

Το όνομα της μεθοδολογίας αυτής της ανάλυσης προέρχεται από την Ελληνική λέξη «Αρχέτυπος» το οποίο σημαίνει αυτός που αποτελεί πρότυπο και έχει τις ρίζες της στα γραπτά

που τυπώθηκαν την πρώτη περίοδο της τυπογραφίας. Η ανάλυση αυτή προτάθηκε από τους Adele Cutler και Leo Breiman το 1994 στο ομώνυμο άρθρο τους «Archetypal Analysis» [68].

Η μεθοδολογία εστιάζει στην ανάλυση πολυπαραγοντικών δεδομένων, δεδομένα δηλαδή των οποίων η ανάλυση βασίζεται σε ένα διάνυσμα μεταβλητών για την κάθε παρατήρηση. Μελετώντας αυτόν τον χώρο πολλαπλών διαστάσεων των μετρήσεων γίνεται προσπάθεια αποτύπωσης σημείων αναφοράς τα οποία ονομάζονται αρχέτυπα. Τα αρχέτυπα που παράγονται αντιπροσωπεύουν τα αποκλίνοντα προφίλ του συνόλου των δεδομένων. Δομούνται με τέτοιο τρόπο ώστε κάθε άλλο μέλος του συνόλου να μπορεί να προσεγγιστεί ικανοποιητικά χρησιμοποιώντας ένα μείγμα αυτών των αρχετύπων. Το σύνολο των σημείων αναφοράς που αποτελούν τα αρχέτυπα βρίσκονται στα όρια του συνόλου των δεδομένων. Ενώ η μεθοδολογία προσφέρει επίσης και έναν μηχανισμό για την αξιολόγηση των συντελεστών ομοιότητας που καθορίζουν το πόσο μοιάζει κάθε σημείο των αρχετύπων με τα υπόλοιπα.

Σημαντικό ρόλο για την σωστή απόδοση των αποτελεσμάτων της μεθοδολογίας παίζει η απόφαση σχετικά με τον συνολικό αριθμό αρχετύπων που θα επιλεγούν. Γενικά ισχύει πως όσο μεγαλύτερος είναι ο αριθμός των αρχετύπων τόσο καλύτερα φαίνεται η διαφορετικότητα που υπάρχει μεταξύ των υποομάδων ενός συνόλου δεδομένων. Εάν όμως τα αρχέτυπα είναι παραπάνω από όσα πρέπει τότε δυσχεραίνεται η χρήση τους προκειμένου να αποτελέσουν σημείο αναφοράς ενός benchmark. Πρέπει λοιπόν να βρεθεί η χρυσή τομή μεταξύ του αριθμού των σημείων που είναι αρκετά ώστε να αποδώσουν την διαφορετικότητα του συνόλου στον μέγιστο βαθμό, χωρίς ωστόσο να ξεπεραστεί το όριο όπου ένα νέο αρχέτυπο όχι μόνο δεν προσφέρει τίποτα στην ποιότητα της ανάλυσης αλλά αποτελεί τροχοπέδη στον τελικό σκοπό της.

Αποτελέσματα

Δημιουργία μήτρας κατάταξης

Αρχικά για να προκύψουν τα αποτελέσματα πρέπει τα δεδομένα να δομηθούν στην σωστή τους μορφή. Όπως έχει ήδη αναφερθεί έγινε μία προσπάθεια τα δεδομένα να εκφραστούν σε ίδιες μονάδες προκειμένου η σύγκριση να είναι εφικτή. Για αυτό τον λόγο επιλέχθηκαν ως

μέτρο αξιολόγησης οι μετρικές οι οποίες εκφράζουν το τεχνικού χρέος σε ανθρωποώρες. Ωστόσο αυτό φαίνεται πως για την στατιστική ανάλυση δεν είναι αρκετό καθώς τα εργαλεία ακολουθούν διαφορετικές προσεγγίσεις αξιολόγησης του χρέους που οδηγούν σε διαφορετικές τιμές και διακυμάνσεις των αποτελεσμάτων. Οι διαφορές των εργαλείων αρχικά αφορούν το σύνολο των κανόνων που ελέγχει το κάθε ένα. Αν ένα εργαλείο ελέγχει περισσότερους κανόνες από ένα άλλο, το τελικό TD είναι πιθανότατα μεγαλύτερο. Επίσης κάθε εργαλείο χρησιμοποιεί διαφορετική μέθοδο μέτρησης του χρέους η οποία μπορεί να οδηγήσει σε διαφορετική τιμή. Τέλος η εκτίμηση κάθε εργαλείου σχετικά με τον χρόνο που χρειάζεται η διόρθωση κάθε προβλήματος εξαρτάται από τις μελέτες και τις μετρήσεις που έχει κάνει κάθε ομάδα ανάπτυξης. Υπάρχει όμως ένα μέγεθος το οποίο είναι ανεξάρτητο από τις ιδιαιτερότητες του κάθε εργαλείου και αυτό είναι η κατάταξη (ranking) κάθε κλάσης μέσα στο σύνολο των προβληματικών κλάσεων κάθε εργαλείου. Για κάθε ένα εργαλείο όλες οι κλάσεις αξιολογούνται και κατατάσσονται με τα ίδια κριτήρια. Έτσι η κατάταξη είναι ένα μέγεθος που κατά απόλυτη τιμή μπορεί να αξιοποιηθεί στην ανάλυση απαλλαγμένο από τις όποιες επιρροές μπορούν να έχουν οι διαφοροποιήσεις των εργαλείων.

Έτσι λοιπόν για κάθε έργο λογισμικού δημιουργείται μία μήτρα, όπως η παρακάτω, η οποία έχει σε κάθε γραμμή μία κλάση του έργου και κάθε στήλη αφορά ένα εργαλείο ανάλυσης. Οι τιμές στα κελιά της μήτρας είναι η θέση που έχει η κάθε κλάση στην κατάταξη για το κάθε εργαλείο.

Table 5 - Παράδειγμα μήτρας κατάταξης κλάσεων για κάθε εργαλείο

Όνομα Κλάσης	SonarQube	CAST	Squore
Class1	Rank 11	Rank 12	Rank 13
Class2	Rank 21	Rank 22	Rank 23
Class3	Rank 31	Rank 32	Rank 33

Δημιουργία διαγραμμάτων διασποράς

Αφού πλέον υπάρχουν οι μήτρες μπορούν να κατασκευαστούν τα διαγράμματα διασποράς γνωστά και ως στικτά διαγράμματα. Αυτά τα διαγράμματα βοηθούν στο να γίνει κατανοητή ακόμα και οπτικά η σχέση μεταξύ δύο μεταβλητών. Πέρα από το ένα υπάρχει σχέση η όχι, ο ερευνητής μπορεί να δει και ως έναν βαθμό την ένταση αυτής της σχέσης. Στην περίπτωση της συγκεκριμένης μελέτης οι άξονες του διαγράμματος είναι τα εργαλεία ανάλυσης κώδικα που χρησιμοποιήθηκαν. Κάθε σημείο που εμφανίζεται αντιπροσωπεύει κάθε κλάση και έχει ως συντεταγμένες την τιμή της κατάταξης που έχει κάθε κλάση σε κάθε ένα από τα εργαλεία.

Ένα παράδειγμα τέτοιου διαγράμματος φαίνεται παρακάτω στις περίπτωση των 2 αλλά και των τριών διαστάσεων καθώς στην περίπτωσή μας οι παρατηρητές είναι τρεις.

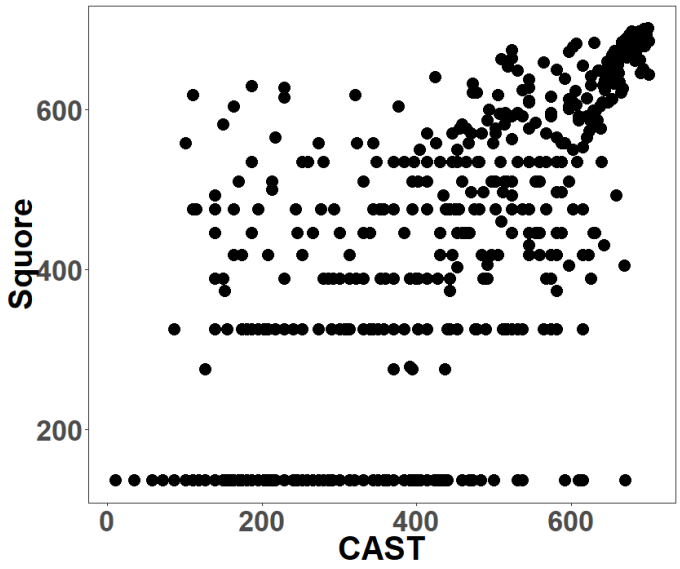

Figure 1- Διάγραμμα διασποράς κλάσεων για τα εργαλεία Squore και CAST στο OpenNLP

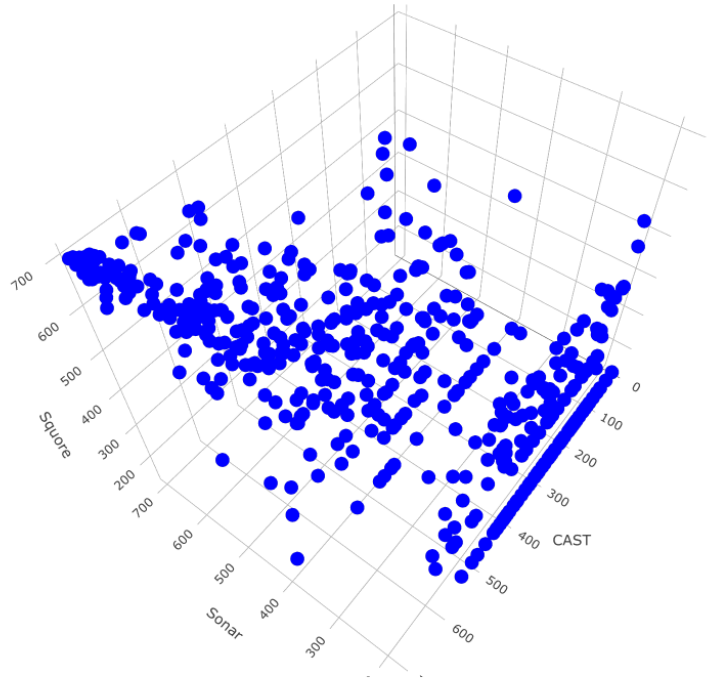

Figure 2 - Διάγραμμα διασποράς τριών διαστάσεων για το OpenNLP και όλα τα εργαλεία

Αρχικά αναφερόμενοι στην εικόνα των δύο διαστάσεων όπου η απεικόνιση των σημείων είναι πιο σαφής, εμφανίζονται στους άξονες τα εργαλεία Squore και CAST. Παρατηρώντας την κατανομή των σημείων πάνω στο διάγραμμα αρχικά παρατηρείται στην επάνω δεξιά γωνία μία πυκνή συγκέντρωση σημείων. Αυτό σημαίνει ότι για ένα αρκετά μεγάλο σύνολο κλάσεων και τα δύο εργαλεία συμφωνούν ότι είναι αρκετά προβληματικές. Αντίθετα παρατηρώντας τις δύο σχεδόν παράλληλες του άξονα Χ ευθείες που σχηματίζονται προς την μέση και αρκετά χαμηλά στο διάγραμμα, καταλήγουμε στο συμπέρασμα πως υπάρχει ένα σύνολο κλάσεων που για το Squore δεν θεωρούνται πολύ προβληματικές σε αντίθεση με το CAST που εντοπίζει πιο έντονο τεχνικό χρέος σε αυτές. Άρα αυτό είναι ένα σημείο ασυμφωνίας των δύο εργαλείων. Συνεπώς έχουμε δύο γενικές κατηγορίες οι οποίες μπορούμε να πούμε ότι αναγνωρίζονται εδώ. Η πρώτη είναι αυτή που θα ονομάσουμε **Ruler** και είναι τα υποσύνολα στα οποία υπάρχει συμφωνία των εργαλείων, και η δεύτερη είναι αυτή που θα ονομάσουμε **Rebel** όπου υπάρχει διαφωνία μεταξύ των εργαλείων για την κατάταξη των στοιχείων.

Αν αναλύσουμε το γράφημα στις τρεις του διαστάσεις αρχικά παρατηρούμε ότι πλέον στον τρίτο άξονα προστίθεται και το SonarQube. Θα διαπιστώσουμε επιπλέον πως υπάρχει και μία ακόμα κατηγορία προφίλ η οποία θα προκύψει και θα την ονομάσουμε **Partner**. Σε αυτή την κατηγορία 2 από τα 3 εργαλεία συμφωνούν ότι μία κλάση έχει υψηλό τεχνικό χρέος ενώ στο τρίτο εργαλείο η κλάση το ποσό του εμφανίζεται σαφώς μικρότερο.

Υπολογισμός του Kendall's W coefficient of concordance

Ο υπολογισμός του Kendall's W coefficient of concordance δείκτη πραγματοποιήθηκε για κάθε ένα από τα έργα λογισμικού που αναλύθηκαν στα τρία εργαλεία. Το p – value, η πιθανότητα δηλαδή να λάβουμε μία ακραία τιμή όταν η υπόθεση Ho, η υπόθεση της συμφωνίας των εργαλείων, είναι αληθινή, σε κάθε περίπτωση ορίζεται στο 0.001.

Table 6 - Kendall's W δείκτης για κάθε έργο

Έργο	Kendall's W coefficient of concordance
Cayenne	0.755
Deltaspike	0.718
FOP	0.722
jClouds	0.687
Maven	0.713
Mina	0.703
OpenJPA	0.834
Opennlp	0.793
PdfBox	0.730
Wss4j	0.781

Σύμφωνα με τον εμπειρικό κανόνα τον οποίο προτείνει ο Roy C. Schmidt στο άρθρο του «Managing Delphi Surveys Using Nonparametric Statistical Techniques» [69] το 1997 εάν ο δείκτης Kendall's W ξεπερνάει το 0.7 τότε μπορεί να θεωρηθεί πως η συμφωνία μεταξύ των παρατηρητών είναι αρκετά ισχυρή. Στα δέκα έργα τα οποία συμμετείχαν στην ανάλυση, η χαμηλότερη τιμή που παρατηρείται είναι στο jClouds το 0.687, το οποίο ωστόσο είναι αρκετά κοντά στο όριο του 0.7. Όλα τα υπόλοιπα το ξεπερνάνε, με την μεγαλύτερη τιμή το 0.834 στο Open JPA. Άρα λοιπόν και απαντώντας στο ερώτημα που τέθηκε αρχικά για τον βαθμό συμφωνίας μεταξύ των εργαλείων, όπως προκύπτει από την παραπάνω ανάλυση, τα εργαλεία συμφωνούν σε έναν μεγάλο βαθμό ως προς το ποιες είναι οι προβληματικές κλάσεις αλλά και στην κατάταξη των πιο προβληματικών από αυτές.

Ανάλυση Αρχετύπων

Η ίδια μήτρα που περιλαμβάνει τη κατάταξη των κλάσεων στα εργαλεία και παρουσιάστηκε στον πίνακα **5** αποτελεί και την είσοδο για την Ανάλυση Αρχετύπων. Προκειμένου να εντοπιστούν ποιες κλάσεις από το σύνολο των προβληματικών εμφανίζουν παρόμοια χαρακτηριστικά προκειμένου να μπορέσουν να αποτελέσουν ένα αρχέτυπο, ακολουθούνται τα παρακάτω τρία βήματα.

- Αναγνώριση των αρχετύπων μελετώντας το σύνολο των διαφορετικών μετρήσεων τεχνικού χρέους όπως αυτές προκύπτουν για κάθε ένα από τα εργαλεία ανάλυσης κώδικα που χρησιμοποιήθηκαν.
- Εφαρμογή της λύσης αρχετύπων στο πλαίσιο της διαχείρισης του τεχνικού χρέους ώστε να προσδιοριστούν τα χαρακτηριστικά του καθενός.
- Αναγνώριση των κλάσεων οι οποίες ανήκουν στα αρχέτυπα και εμφανίζουν είτε υψηλό, είτε χαμηλό χρέος σε όλα τα εργαλεία και άρα μπορούν να αποτελέσουν το «χρυσό σύνολο».

Αρχικά λοιπόν αναγνωρίστηκαν 8 συνολικά αρχέτυπα με τα παρακάτω χαρακτηριστικά:

- **The Max Ruler** : Περιλαμβάνει τις κλάσεις με το υψηλότερο Τεχνικό Χρέος στις οποίες συμφωνούν όλα τα εργαλεία

- **The Min Ruler** : Περιλαμβάνει τις κλάσεις με το χαμηλότερο Τεχνικό Χρέος στις οποίες συμφωνούν όλα τα εργαλεία
- **The Rebel 1** : Περιλαμβάνει τις κλάσεις που εμφανίζουν υψηλό Τεχνικό Χρέος στο εργαλείο SonarQube αλλά χαμηλό στα εργαλεία Squore και Cast
- **The Rebel 2** : Περιλαμβάνει τις κλάσεις που εμφανίζουν υψηλό Τεχνικό Χρέος στο εργαλείο Cast αλλά χαμηλό στα εργαλεία Squore και SonarQube
- **The Rebel 3** : Περιλαμβάνει τις κλάσεις που εμφανίζουν υψηλό Τεχνικό Χρέος στο εργαλείο Squore αλλά χαμηλό στα εργαλεία SonarQube και Cast
- **The Partner 1** : Περιλαμβάνει τις κλάσεις που εμφανίζουν υψηλό Τεχνικό Χρέος στα εργαλεία SonarQube και Squore αλλά χαμηλό στο εργαλείο Cast
- **The Partner 2** : Περιλαμβάνει τις κλάσεις που εμφανίζουν υψηλό Τεχνικό Χρέος στα εργαλεία SonarQube και Cast αλλά χαμηλό στο εργαλείο Squore
- **The Partner 3** : Περιλαμβάνει τις κλάσεις που εμφανίζουν υψηλό Τεχνικό Χρέος στα εργαλεία Squore και Cast αλλά χαμηλό στο εργαλείο SonarQube

Εδώ γίνεται εμφανές πως ο αριθμός των αρχέτυπων επηρεάζεται σαφώς από τον αριθμό των εργαλείων που χρησιμοποιούνται για την ανάλυση του κώδικα. Στην περίπτωσή της τρέχουσας μελέτης υπάρχουν 3 εργαλεία. Εάν ο αριθμός των εργαλείων αυξηθεί στα 4 για παράδειγμα τότε σίγουρα θα προστεθεί από ένα ακόμα αρχέτυπο στις κατηγορίες των Rebel και των Partner, και εκτός αυτού πιθανότατα θα πρέπει να εξεταστούν και οι περιπτώσεις όπου υπάρχει συμφωνία κατά ζεύγη.

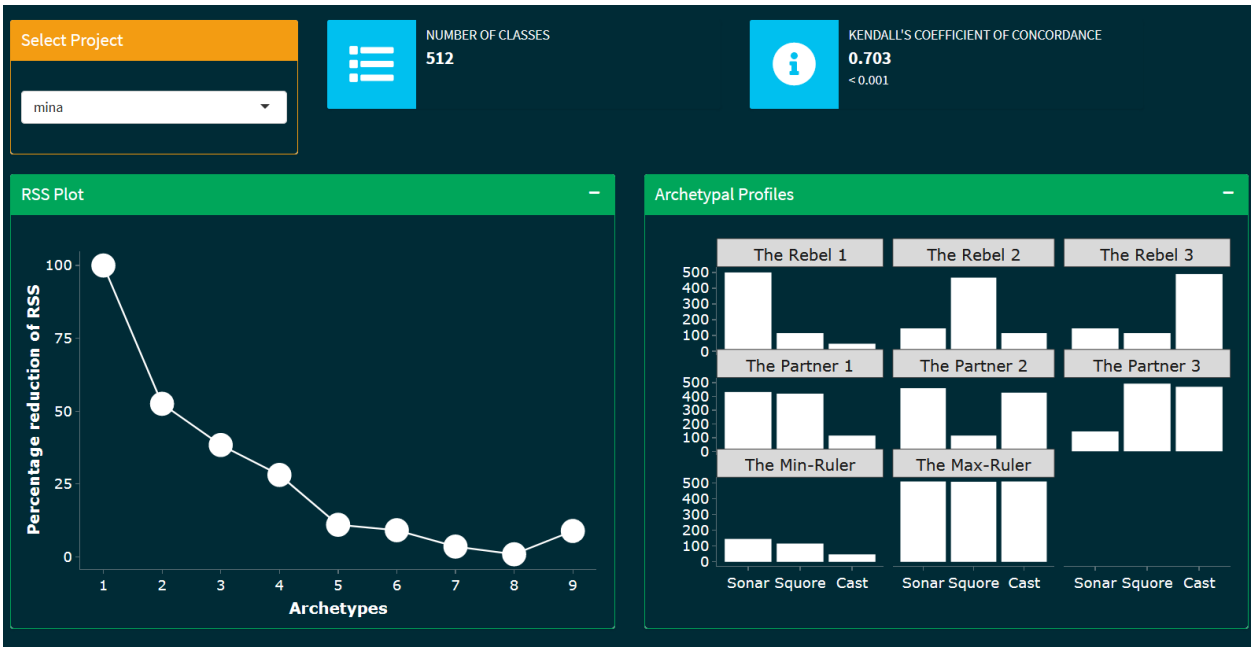

Figure 3- Διαγράμματα αρχετύπων του TD Benchmarker

Επόμενο στάδιο της μεθοδολογίας είναι να εξετάσουμε το πόσες κλάσεις βρίσκονται κοντά στην γειτονιά του Max Ruler. Πόσες δηλαδή εμφανίζονται προβληματικές με μεγάλο ποσό Τεχνικού Χρέους στο σύνολο των εργαλείων. Για να γίνει αυτός ο έλεγχος πρέπει πρώτα να αποφασιστεί το κατώφλι το οποίο ορίζει την συμφωνία. Ο ορισμός αυτός όμως δεν είναι κάτι το απόλυτο καθώς πρέπει να εξυπηρετεί τις ανάγκες κάθε ερευνητή. Εξαιτίας αυτού λοιπόν προτιμήθηκε να γίνει μία διερεύνηση σχετικά με το πώς συμπεριφέρεται το ποσοστό της ομοιομορφίας καθώς το κατώφλι αυτό μεταβάλλεται.

Ως ποσοστό ομοιομορφίας ορίζεται ο λόγος του συνολικού αριθμού κλάσεων ενός έργου, ως προς τον αριθμό των κλάσεων που συμμετέχουν στην τομή των τριών εργαλείων με δεδομένο κατώφλι. Για παράδειγμα στο έργο mina οι συνολικές κλάσεις είναι 512 όπως φαίνεται στην εικόνα 4. Αν ορίσουμε το κατώφλι ομοιότητας στο 0.8 τότε βλέπουμε πως στην τομή συμμετέχουν 33 κλάσεις όπως φαίνεται στην εικόνα 5. Άρα αυτό μας δίνει έναν λόγο 33/512=0.064

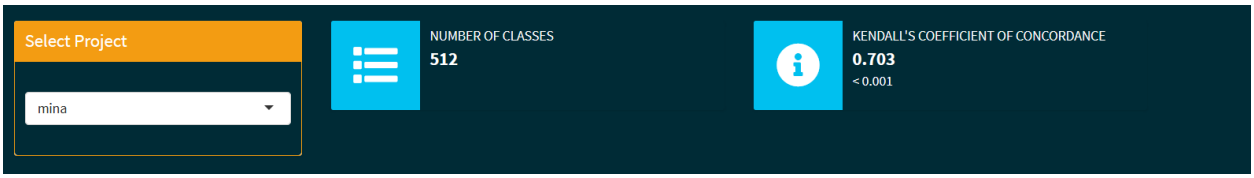

Figure 4 - Σύνολο κλάσεων και δείκτης του Kendall του mina

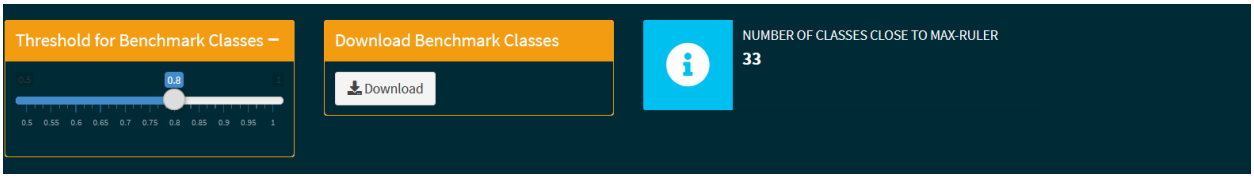

Figure 5 - Ορισμός κατωφλίου και σύνολο κλάσεων τομής

Αξίζει ενδεικτικά να δούμε πως συμπεριφέρεται αυτός ο δείκτης για το mina όταν το κατώφλι ομοιομορφίας μεταβάλλεται. Οι διαφορετικές τιμές φαίνονται στον παρακάτω πίνακα

Table 7 - Ποσοστό κλάσεων που συμμετέχουν στην τομή για διαφορετικά κατώφλια στο έργο mina

Κατώφλι	Αριθμός Κλάσεων	Ποσοστό
0.60	73	14,26%
0.65	63	12,3%
0.70	49	9,57%
0.75	39	7,62%
0.80	33	6,45%
0.85	22	4,3%
0.90	14	2,73%

Η συμπεριφορά που παρατηρείται εδώ και είναι παρόμοια με αυτή την οποία βλέπουμε σε όλα τα έργα που αναλύθηκαν είναι πως όσο το κατώφλι μεγαλώνει και άρα γίνονται πιο αυστηρά τα κριτήρια τόσο το ποσοστό των κλάσεων της τομής μικραίνει.

Επόμενο βήμα της ανάλυσης είναι η εξέταση του κατά πόσο τα αποτελέσματα αυτά μπορούν να γενικευτούν. Ένα benchmark θα πρέπει να χρησιμοποιηθεί για οποιοδήποτε έργο λογισμικού και όχι μόνο για τα συγκεκριμένα δέκα που συμμετέχουν στην μελέτη άρα είναι πολύ σημαντικό τα αποτελέσματα να είναι ανεξάρτητα της βάσης που χρησιμοποιήθηκε για την παραγωγή του.

Για την εξέταση της γενίκευση χρησιμοποιήθηκε το μοντέλο «Linear Mixed Effects». Τα μοντέλα αυτά χρησιμοποιούνται για να περιγράψουν τις σχέσεις μεταξύ μίας μεταβλητής απόκρισης και των μεταβλητών των δεδομένων τα οποία ομαδοποιούνται με βάση ένα ή περισσότερα κριτήρια ταξινόμησης [70]. Η μέθοδος αυτή μπορεί να εκτιμήσει και τις στατικές και σταθερές επιπτώσεις του των δεδομένων αλλά και τυχαία συμβάντα που μπορούν να εντοπιστούν και να επηρεάσουν τα αποτελέσματα.

Από αυτή την ανάλυση προκύπτει πως ο ορισμός του κατωφλίου (threshold) είναι σημαντικός παράγοντας ο οποίος επηρεάζει σε μεγάλο βαθμό την μεταβλητή που απαντάει στο ερώτημα ποιες είναι οι κλάσεις με το υψηλότερο τεχνικό χρέος. Επίσης εφαρμόζοντας την μέθοδο Tukey's HSD test προκειμένου να συγκριθούν τα αποτελέσματα για τα διαφορετικά κατώφλια, προκύπτει ότι υπάρχουν σημαντικές διαφορές μεταξύ τους.

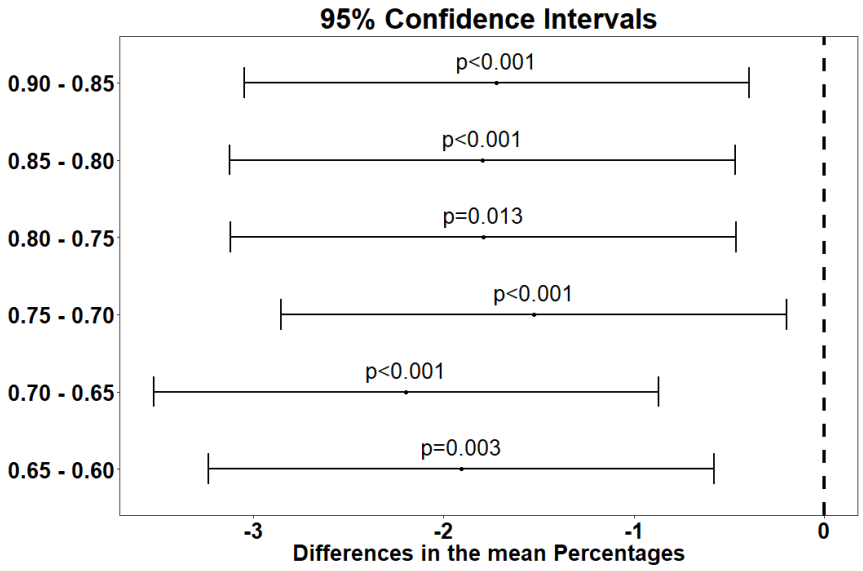

Figure 6 - Post hoc ανάλυση για το LME μοντέλο

Πλέον υπάρχει η δυνατότητα αξιολόγησης των χαρακτηριστικών των κλάσεων του συνόλου των εξεταζόμενων έργων προκειμένου να εντοπιστεί το ποσοστό ή ακόμα και ο αριθμός αυτών οι οποίες μοιάζουν με τις κλάσεις που αποτελούν το σύνολο όσων ακολουθούν το αρχέτυπο του Max Ruler. Ο υπολογισμός μπορεί να γίνει για διαφορετικά κατώφλια. Οι κλάσεις οι οποίες πλέον βρίσκονται σε αυτό το σετ που αξιολογήθηκε αποτελούν και το benchmark το οποίο μπορεί να χρησιμοποιηθεί ως βάση για επέκταση της μελέτης. Για παράδειγμα αυτές οι κλάσεις μπορούν να δοθούν ως είσοδο σε ένα νευρωνικό δίκτυο προκειμένου να εκπαιδευτεί και να αναγνωρίζει κλάσεις με παρόμοια χαρακτηριστικά σε ένα οποιοδήποτε έργο λογισμικού. Αυτόματα θα μπορεί να αποφανθεί με σχετική βεβαιότητα ότι η κλάση θα εμφανίζει μεγάλο τεχνικό χρέος χωρίς να προηγηθεί ανάλυση με οποιοδήποτε εργαλείο πόσο μάλλον η συγκριτική εξέτασή της από περισσότερα του ενός.

TD Benchmarker

Στα πλαίσια της ανάλυσης των αποτελεσμάτων από το σύνολο των συμμετεχόντων στην έρευνα αποφασίστηκε και υλοποιήθηκε η διαδικτυακή εφαρμογή TD Benchmarker η οποία παρουσιάζει γραφικά και με διαδραστικό τρόπο τα σημαντικότερα των σταδίων όλης της διαδικασίας. Η εφαρμογή είναι διαθέσιμη στον παρακάτω σύνδεσμό

http://se.uom.gr:3838/

Η εφαρμογή χωρίζεται σε δύο βασικές οθόνες. Στην πρώτη οθόνη ο χρήστης μπορεί να επιλέξει ένα από τα δέκα έργα τα οποία χρησιμοποιήθηκαν στην μελέτη και αναλύθηκαν από τα εργαλεία ποιότητας κώδικα αλλά και συμμετείχαν στην στατιστική ανάλυση. Όπως φαίνεται και στην εικόνα 7 για κάθε έργο εμφανίζεται το σύνολο των κλάσεων που διέθετε στην έκδοση που συμμετέχει στην έρευνα. Στην συνέχεια εμφανίζεται η τιμή του Kendall’s W coefficient of concordance δείκτη. Τα δύο διαγράμματα που ακολουθούν δείχνουν την κατανομή των αρχετύπων καθώς και τα ραβδογράμματα για κάθε ένα από τα οχτώ αρχέτυπα προφίλ που εντοπίστηκαν.

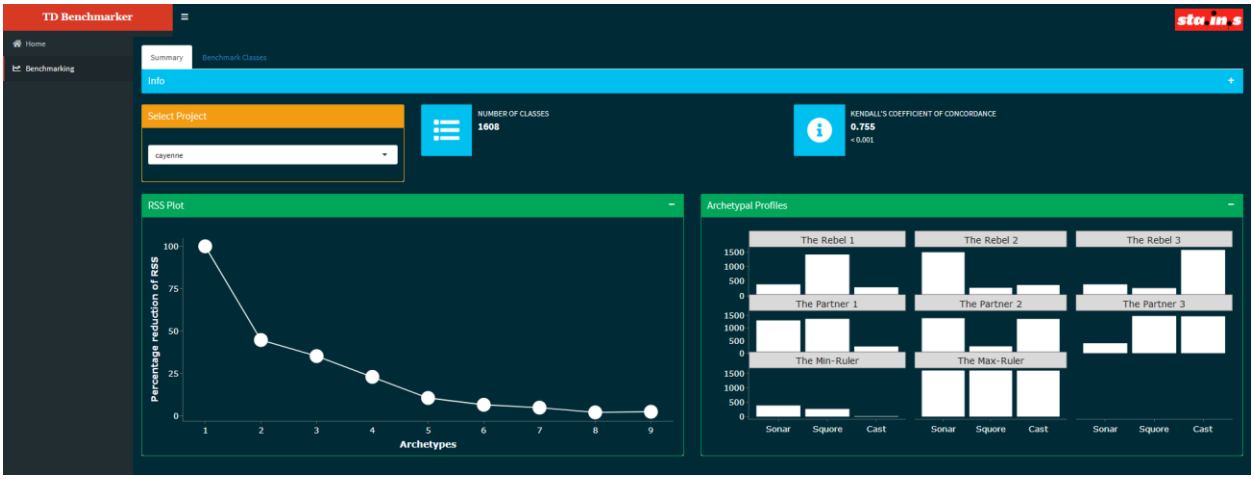

Figure 7 - Οθόνη διαγραμμάτων του TD Benchmarker

Στην δεύτερη οθόνη που εμφανίζεται στην εικόνα 8 πλέον εμφανίζονται περισσότερες λεπτομέρειες που αφορούν τις μονάδες που χρησιμοποιήθηκαν, τις ίδιες τις κλάσεις. Ο χρήστης εδώ μπορεί να προσαρμόσει την τιμή του κατωφλίου και να δει την επίπτωση που έχει αυτή στον αριθμό των κλάσεων οι οποίες μπορούν και το περνάνε κάθε στιγμή. Βλέπει λοιπόν ζωντανά το συμπέρασμα που αναφέρθηκε στην επόμενη ενότητα πως όσο το κατώφλι μεγαλώνει σε απόλυτη τιμή τόσο λιγότερες είναι οι κλάσεις που συμμετέχουν στην τομή. Από τον πίνακα που εμφανίζεται στην σελίδα ο χρήστης μπορεί να αντλήσει πληροφορίες για το ποιες είναι οι κλάσεις αυτές που θεωρούνται πιο προβληματικές και από τα τρία εργαλεία και ακολουθούν το αρχέτυπο προφίλ του Max Ruler. Τέλος ο χρήστης μπορεί να κατεβάσει τα δεδομένα αυτά σε μορφή .csv αρχείου μαζί με τις μετρήσεις που προσφέρονται για κάθε εργαλείο σε αυτή την οθόνη αλλά και τον δείκτη για το Max Ruler προφίλ προκειμένου να μπορέσει να τα επεξεργαστεί περεταίρω.

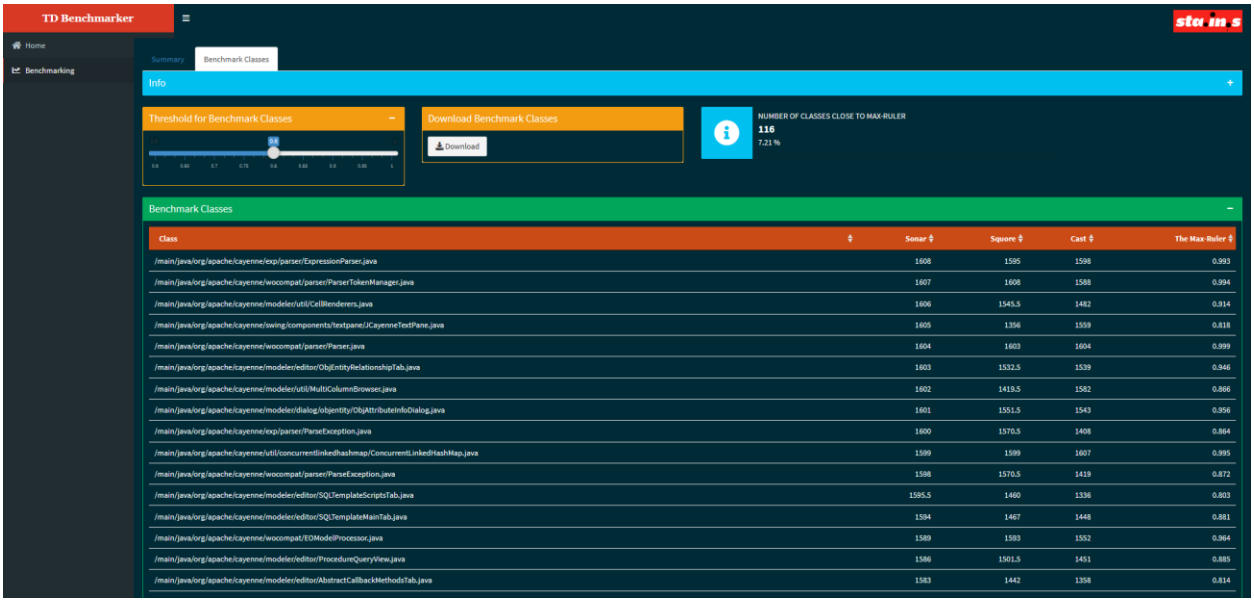

Figure 8 - Κλάσεις που συμμετέχουν στην τομή για συγκεκριμένο κατώφλι

Επίλογος

Σύνοψη και συμπεράσματα

Καθώς όλο και περισσότερες ανάγκες της κοινωνίας και της καθημερινότητας αναζητούν την λύση τους και την αυτοματοποίησή τους μέσα από έργα λογισμικού, αυτά γίνονται όλο και μεγαλύτερα και πιο απαιτητικά. Είναι σημαντικό λοιπόν και λόγω του μεγέθους τους και λόγω της κρισιμότητας των προβλημάτων που καλούνται να διαχειριστούν αυτά τα έργα να μπορούν εύκολα να ελέγχονται αλλά και να συντηρούνται ώστε να διορθώνονται οι όποιες προβληματικές λειτουργίες τους, και ταυτόχρονα να μπορούν να επεκταθούν. Η ποιότητα του κώδικα παίζει σημαντικότατο ρόλο σε αυτό καθώς είναι αυτή η οποία καθορίζει το πόσο εύκολα μπορούν να πραγματοποιηθούν οι προαναφερθείσες ενέργειες ή ακόμα και να τις καταστήσουν απαγορευτικές.

Υπάρχει ένα κλίμα στην βιβλιογραφία αλλά και στην βιομηχανία παραγωγής λογισμικού που στρέφεται στην ανάδειξη της αναγκαιότητας για συνεχή παρακολούθηση της ποιότητας και της ένταξης των εργασιών που αφορούν την βελτίωσή της στην κανονική ροή της παραγωγικής διαδικασίας. Σε αυτό το πνεύμα έχουν κινηθεί αρκετές εταιρείες παραγωγής λογισμικού αλλά και ομάδες που προσφέρουν εργαλεία ανοιχτού κώδικα, έτσι ώστε να δημιουργήσουν πλατφόρμες ανάλυσης των έργων οι οποίες μπορούν να αναδείξουν το συντομότερο δυνατόν τα προβλήματα που υπάρχουν προκειμένου να διορθωθούν και να μην διαιωνιστούν επηρεάζοντας και άλλα υγιή κομμάτια κώδικα.

Στην προσπάθεια να γεφυρωθεί το χάσμα επικοινωνίας μεταξύ των μηχανικών λογισμικού και των διοικητικών στελεχών ώστε να κατανοήσουν όλοι την σημασία της ποιότητας όσο χρονοβόρα και να είναι αυτή η διαδικασία, χρησιμοποιήθηκε ο όρος του τεχνικού χρέους. Τον όρο αυτόν τον υιοθέτησαν γρήγορα τα εργαλεία ανάλυσης και πολλά από αυτά αναπτύχθηκαν αποκλειστικά για αυτόν τον σκοπό. Το πρόβλημα που προκύπτει είναι πως ενώ η ποιότητα του κώδικα μετράται ως αναμενόμενο σε ποιοτικά χαρακτηριστικά, αυτά πρέπει να ποσοτικοποιηθούν με κάποιον τρόπο ώστε να μεταφραστούν σε ώρες ή χρηματικές ομάδες και να δείξουν το χρέος. Η διαφοροποίηση λοιπόν των εργαλείων έγκειται αρχικά στο τι

θεωρείται πρόβλημα σε έναν κώδικα και έπειτα στο πόσο κρίσιμο είναι αυτό το πρόβλημα και πόσο στοιχίζει στην κάθε εταιρεία.

Επειδή η χρήση περισσότερων του ενός εργαλείου είναι και χρονοβόρα αλλά προσθέτει και επιπλέον κόστος στην επιχείρηση υπάρχει η ανάγκη να ξέρει κανείς εκ των προτέρων ποιο εργαλείο καλύπτει τις ανάγκες του και αν είναι αρκετό ένα ή περισσότερα εργαλεία αν τρέξουν σε συνδυασμό.

Σε αυτή την εργασία λοιπόν έγινε μία προσπάθεια να συγκεντρωθούν τα εργαλεία που χρησιμοποιούνται από το μεγαλύτερο μέρος της αγοράς για την διασφάλιση της ποιότητας του κώδικα και να πραγματοποιηθεί μία σύγκριση του βαθμού συμφωνίας των αποτελεσμάτων τους. Χρησιμοποιήθηκαν λοιπόν τα εργαλεία SonarQube, SQuORE και CAST με τα οποία αναλύθηκαν 10 γνωστά και ενεργά έργα της Apache Foundation. Συλλέχθηκαν τα δεδομένα που αφορούν το τεχνικό χρέος με τρόπο τέτοιο ώστε να εκφράζονται σε κοινές μονάδες (ανθρωποώρες) προκειμένου να μπορούν στην συνέχεια να αναλυθούν μέσω της στατιστικής και να εξεταστεί ο βαθμός συμφωνίας τους.

Από την στατιστική ανάλυση η οποία εφαρμόστηκε στα αποτελέσματα των εργαλείων προκύπτει πως τα τρία εργαλεία που εξετάστηκαν συμφωνούν σε αρκετά μεγάλο βαθμό μεταξύ τους. Αυτό δείχνει ο δείκτης W του Kendall ο οποίος για όλα τα έργα λογισμικού που αναλύθηκαν σε αυτή την μελέτη περίπτωσης ξεπερνάει το κατώφλι του εμπειρικού κανόνα που ορίζεται στο 0,7. Υπάρχει ένα αρκετά μεγάλο ποσοστό κλάσεων οι οποίες θεωρούνται προβληματικές και μάλιστα εμφανίζουν υψηλές τιμές τεχνικού χρέους και στις τρεις μετρήσεις. Αντίστοιχα υπάρχει ένα ποσοστό κλάσεων που σε όλα τα εργαλεία εμφανίζουν μικρό έως και καθόλου τεχνικό χρέος.

Η συμφωνία βέβαια δεν είναι απόλυτη. Και στα τρία εργαλεία παρατηρούνται αποκλίσεις δύο ειδών. Στην πρώτη περίπτωση υπάρχουν κλάσεις που για το συγκεκριμένο εργαλείο δεν θεωρούνται ιδιαίτερα προβληματικές ενώ αντίθετα για τα άλλα δύο εμφανίζουν μεγάλο Τεχνικού Χρέος. Αντίστοιχα στην δεύτερη περίπτωση υπάρχουν κλάσεις που εμφανίζονται ιδιαίτερα προβληματικές για ένα εργαλείο όχι όμως και για τα άλλα δύο της μελέτης.

Έτσι λοιπόν προκύπτουν τα 8 αρχέτυπα, οι 8 κατηγορίες οι οποίες μπορούν μόνες τους ή σε συνδυασμό να περιγράψουν το σύνολο των κλάσεων που συναντώνται στα έργα λογισμικού. Από τα αρχέτυπα αυτά εξετάζουμε αυτό που αφορά το σύνολο των πιο προβληματικών κλάσεων στις οποίες συμφωνούν και τα τρία εργαλεία. Το ποσοστό των κλάσεων που ανήκουν στην γειτονιά αυτού του αρχέτυπου εξαρτάται από το επίπεδο της ομοιότητας που επιθυμεί να εξετάσει ο κάθε ερευνητής. Σε κάθε περίπτωση ωστόσο γεγονός αποτελεί πως όσο αυτό το επίπεδο ομοιότητας αυξάνεται τόσο το ποσοστό των κλάσεων που συμμετέχουν στην τομή μειώνεται καθώς τα κριτήρια γίνονται ολοένα και πιο αυστηρά.

Από την ανάλυση αρχετύπων προκύπτει πως υπάρχει ένα σύνολο κλάσεων οι οποίες εμφανίζουν υψηλό ποσό τεχνικού χρέους και στα τρία εργαλεία. Αυτές οι κλάσεις έχουν κάποια κοινά χαρακτηριστικά. Με δεδομένα λοιπόν τα χαρακτηριστικά αυτά μπορεί να δομηθεί ένα benchmark σύμφωνα με το οποίο μόλις αναγνωρίζονται αυτά μέσα σε μία κλάση τότε μπορούμε να αποφανθούμε πως αυτή η κλάση θα εμφανίσει μεγάλο τεχνικό χρέος και για τα τρία αυτά εργαλεία, χωρίς να χρειαστεί να περάσει από όλη την διαδικασία της ανάλυσης.

Όρια και περιορισμοί της έρευνας

Η έρευνα περιορίστηκε στην ανάλυση 10 εργαλείων ανοιχτού κώδικα. Ο ανοιχτός κώδικας έχει κάποια ιδιαίτερα χαρακτηριστικά τα οποία δεν συμφωνούν πλήρως με αυτά των εμπορικών λογισμικών. Ωστόσο τα έργα που επιλέχθηκαν μπορούν να θεωρηθούν αξιόπιστα καθώς χρησιμοποιούνται και από εμπορικές εταιρείες για την ανάπτυξη των δικών τους λογισμικών. Συνεπώς κατά μία έννοια είναι “εμπορικά” λογισμικά.

Επίσης η έρευνα περιορίστηκε στην χρήση τριών εργαλείων εκτίμησης τεχνικού χρέους. Τα εργαλεία έπρεπε να μπορούν να αναλύσουν έργα γραμμένα σε Java, να έχουν σαφή τεκμηρίωση και να είναι διαθέσιμα είτε δωρεάν είτε να παρέχει η εταιρεία ειδική άδεια για ακαδημαϊκή χρήση. Τα εργαλεία επίσης θα έπρεπε ρητά να προσφέρουν μέτρηση του τεχνικού χρέους ώστε η στάθμιση των παραγόντων να γίνεται με τα κριτήρια που έχουν τεθεί από κάθε

εργαλείο χωρίς την δική μας παρέμβαση. Τα τρία εργαλεία που χρησιμοποιήθηκαν ωστόσο είναι τα πλέον διαδεδομένα στην αγορά και καλύπτουν το μεγαλύτερο μέρος της. Οπότε ακόμα και με αυτόν τον μικρό αριθμό εργαλείων μπορούμε να πούμε ότι αξιολογήθηκαν οι μέθοδοι που χρησιμοποιούνται αυτή την στιγμή από τις περισσότερες επιχειρήσεις.

Οι κανόνες αξιολόγησης του κώδικα καθώς και το ειδικό βάρος του καθενός που χρησιμοποιήθηκαν ήταν αυτά που παρέχει κάθε εργαλείο εξ αρχής. Παρόλο που υπάρχει η δυνατότητα να παραμετροποιηθούν αυτά τα μεγέθη σκοπός δεν ήταν να γίνει η κανονικοποίηση με την δική μας παρέμβαση αλλά να γίνει η αξιολόγηση όπως αρχικά εκτιμάται από την κάθε ομάδα ανάπτυξης των εργαλείων.

Μελλοντικές επεκτάσεις

Όπως έχει αναφερθεί και σε προηγούμενα κεφάλαια για να είναι ένα benchmark αξιοποιήσιμο θα πρέπει να μην εξαρτάται από το ποια έργα αποτελούν την βάση εκτίμησής του. Κάτι λοιπόν που αξίζει να διερευνηθεί είναι η πραγματοποίηση του ίδιου πειράματος με ένα διαφορετικό σύνολο έργων, και κατά πόσο τα αποτελέσματα που θα προκύψουν ταυτίζονται με αυτά που προέκυψαν και παρουσιάστηκαν σε αυτή την εργασία. Ιδιαίτερο ενδιαφέρον μάλιστα θα έχει αν τα έργα που θα συμμετέχουν στην επανάληψη του πειράματος είναι σε διαφορετικές γλώσσες από αυτή της Java ώστε να επιβεβαιωθεί και η παραδοχή που έγινε στην αρχή πως όλα τα έργα λογισμικού ανεξάρτητα από την γλώσσα στην οποία υλοποιήθηκαν εμφανίζουν γενικά τις ίδιες ευπάθειες.

Από την στιγμή που υπάρχουν τα δεδομένα για τα τρία αυτά εργαλεία διαθέσιμα κάθε καινούριο εργαλείο ή κάποιο εργαλείο που για οποιονδήποτε λόγο από όσους αναφέρθηκαν προηγουμένως δεν συμμετείχε στην εργασία, μπορεί να συγκριθεί μαζί τους. Αναλύοντας τα ίδια έργα λογισμικού μπορεί κανείς εύκολα και χονδρικά να δει τον βαθμό συμφωνίας με ένα απλό εργαλείο σύγκρισης όπως αυτό που αναπτύχθηκε και χρησιμοποιήθηκε για την αρχική διερεύνηση των αποτελεσμάτων αυτής της εργασίας προκειμένου να αξιολογηθεί εάν θα συνέχιζε με την στατιστική ανάλυση ή όχι. Για επιπλέον λεπτομέρεια στην ανάλυση θα πρέπει

να επαναληφθεί η στατιστική ανάλυση συμπεριλαμβανομένου των αποτελεσμάτων του νέου εργαλείου.

Σε συνέχεια αυτής της μελέτης ένα πιθανό επόμενο βήμα είναι η αξιοποίηση των αποτελεσμάτων στην μηχανική μάθηση. Καθώς το σύνολο των μονάδων ανάλυσης το αποτελούν οι κλάσεις και όχι τα έργα, η βάση δεδομένων προς ανάλυση είναι αρκετές χιλιάδες δείγματα. Για όλες αυτές τις κλάσεις έχουν προκύψει οι μετρικές και τα χαρακτηριστικά της κάθε μίας μέσω της ανάλυσης των εργαλείων. Για τις κλάσεις που ανήκουν στην τομή λοιπόν των εργαλείων μπορούν να διερευνηθούν τυχόν μοτίβα που προκύπτουν και οδηγούν στο συμπέρασμα ότι μία κλάση εμφανίζει υψηλό τεχνικό χρέος. Τροφοδοτώντας λοιπόν ένα νευρωνικό δίκτυο με τις κλάσεις αυτές μπορεί να μάθει να αναγνωρίζει αν μία κλάση θα εμφανιζόταν ως προβληματική και από τα 3 εργαλεία ή όχι. Έτσι λοιπόν αρκεί ένα εργαλείο και το νευρωνικό δίκτυο για να αποφανθούμε για το αν θα ήταν προβληματική η κλάση και για τα άλλα δύο εργαλεία ή όχι. Η ανάλυση καινούριων έργων λογισμικού γίνεται και πιο γρήγορη και πιο φθηνή.

Βιβλιογραφία

[1] "The WyCash Portfolio Management System." [Online]. Available: http://c2.com/doc/oopsla92.html. [Accessed: 10-Jun-2019].

[2] Α. Χατζηγεωργίου, *Αντικειμενοστρεφής Σχεδίαση*. 2005.

[3] Clemente Izurieta and James Bieman, "Testing Consequences of Grime Buildup in Object Oriented Design Patterns," in *Proc. Int. Conf. Software Testing, Verification, and Reliability*, 2018.

[4] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, Waikiki, Honolulu, HI, USA, 2011, p. 411.

[5] B. Baldassari, "SQuORE: a new approach to software project assessment," presented at the International Conference on Software & Systems Engineering and their Applications, 2013.

[6] CAST and ATOS, "Structural quality an Atos quality approach for the digital age." .

[7] "Software Maintainability | | FREE Demo | | Video Explanation." [Online]. Available: https://www.castsoftware.com/glossary/software-maintainability. [Accessed: 27-May-2019].

[8] T. Dohmen, M. Bruntink, D. Ceolin, and J. Visser, "Towards a Benchmark for the Maintainability Evolution of Industrial Software Systems," in *2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, Berlin, Germany, 2016, pp. 11–21.

[9] A. Mayr, R. Plosch, and C. Korner, "A Benchmarking-Based Model for Technical Debt Calculation," in *2014 14th International Conference on Quality Software*, Allen, TX, USA, 2014, pp. 305–314.

[10] C. Seaman and Y. Guo, "Measuring and Monitoring Technical Debt," in *Advances in Computers*, vol. 82, Elsevier, 2011, pp. 25–46.

[11] R. J. Eisenberg, "A threshold based approach to technical debt," *ACM SIGSOFT Softw. Eng. Notes*, vol. 37, no. 2, p. 1, Apr. 2012.

[12] N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," *Inf. Softw. Technol.*, vol. 70, pp. 100–121, Feb. 2016.

[13] N. Zazworka *et al.*, "Comparing four approaches for technical debt identification," *Softw. Qual. J.*, vol. 22, no. 3, pp. 403–426, Sep. 2014.

[14] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *2010 IEEE International Conference on Software Maintenance*, Timi oara, Romania, 2010, pp. 1–10.

[15] K. Lochmann, "A benchmarking-inspired approach to determine threshold values for metrics," *ACM SIGSOFT Softw. Eng. Notes*, vol. 37, no. 6, p. 1, Nov. 2012.

[16] "SQALE | Software Quality Assessment based on Lifecycle Expectations." [Online]. Available: http://www.sqale.org/. [Accessed: 27-May-2019].

[17] J.-L. Letouzey and M. Ilkiewicz, "Managing Technical Debt with the SQALE Method," *IEEE Softw.*, vol. 29, no. 6, pp. 44–51, Nov. 2012.

[18] "Jenkins." [Online]. Available: https://jenkins.io/. [Accessed: 27-May-2019].

[19] Y. Guo *et al.*, "Tracking technical debt — An exploratory case study," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Williamsburg, VA, USA, 2011, pp. 528–531.

[20] "GitHub - SonarSource/sonarqube: Continuous Inspection." [Online]. Available: https://github.com/SonarSource/sonarqube. [Accessed: 27-May-2019].

[21] “Continuous Inspection | SonarQube.” [Online]. Available: https://www.sonarqube.org/. [Accessed: 27-May-2019].
[22] “Metric Definitions | SonarQube Docs.” [Online]. Available: https://docs.sonarqube.org/latest/user-guide/metric-definitions/. [Accessed: 27-May-2019].
[23] G. A. Campbell and P. P. Papapetrou, *SonarQube in action*. Shelter Island, New York: Manning, 2014.
[24] A. B. Sandberg, M. Staron, and V. Antinyan, “Towards proactive management of technical debt by software metrics,” in *Proceedings of the 14th Symposium on Programming Languages and Software Tools*, 2015.
[25] “The SQALE Pyramid: A powerful indicator | SQALE.” [Online]. Available: http://www.sqale.org/blog/the-sqale-pyramid-a-powerful-indicator. [Accessed: 27-May-2019].
[26] L. B. Foganholi, R. Eduardo Garcia, D. Medeiros Eler, R. Celso Messias Correia, and C. Olivete Junior, “Supporting Technical Debt Cataloging with TD-Tracker Tool,” *Adv. Softw. Eng.*, vol. 2015, pp. 1–12, 2015.
[27] C. Fernandez-Sanchez, H. Humanes, J. Garbajosa, and J. Diaz, “An Open Tool for Assisting in Technical Debt Management,” in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Vienna, Austria, 2017, pp. 400–403.
[28] A. Martini and J. Bosch, “An empirically developed method to aid decisions on architectural technical debt refactoring: AnaConDebt,” in *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE ’16*, Austin, Texas, 2016, pp. 31–40.
[29] “JArchitect :: Java Static Analysis and Code Quality Tool.” [Online]. Available: https://www.jarchitect.com/. [Accessed: 04-May-2019].
[30] K. Chopra and M. Sachdeva, “EVALUATION OF SOFTWARE METRICS FOR SOFTWARE PROJECTS,” *Int. J. Comput. Technol.*, vol. 14, no. 6, pp. 5845–5853, Jun. 2015.
[31] “Compute and Manage the Technical Debt with NDepend.” [Online]. Available: https://www.ndepend.com/docs/technical-debt. [Accessed: 04-May-2019].
[32] A. Tornhill, “Assessing Technical Debt in Automated Tests with CodeScene,” in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Vasteras, 2018, pp. 122–125.
[33] J. Holvitie and V. Leppanen, “DebtFlag: Technical debt management with a development environment integrated tool,” in *2013 4th International Workshop on Managing Technical Debt (MTD)*, San Francisco, CA, USA, 2013, pp. 20–27.
[34] “visminer · GitHub.” [Online]. Available: https://github.com/visminer/. [Accessed: 27-May-2019].
[35] T. S. Mendes, F. G. S. Gomes, D. P. Gonçalves, M. G. Mendonça, R. L. Novais, and R. O. Spínola, “VisminerTD: a tool for automatic identification and interactive monitoring of the evolution of technical debt items,” *J. Braz. Comput. Soc.*, vol. 25, no. 1, Dec. 2019.
[36] S. Arvedahl, “Introducing debtgrep, a tool for fighting technical debt in base station software,” in *Proceedings of the 2018 International Conference on Technical Debt - TechDebt ’18*, Gothenburg, Sweden, 2018, pp. 51–52.
[37] “The Apache Software Foundation.” [Online]. Available: https://www.apache.org/index.html. [Accessed: 27-May-2019].
[38] “The Apache Software Foundation · GitHub.” [Online]. Available: https://github.com/apache/. [Accessed: 27-May-2019].
[39] “Maven - Apache Maven Web Page.” [Online]. Available: https://maven.apache.org/. [Accessed: 27-May-2019].
[40] “Maven - Project Documentation.” The Apache Foundation, 2008.
[41] V. Massol, T. O’Brien, and M. K. Loukides, *Maven: a developer’s notebook*, 1st ed. Beijing ; Sebastopol, CA: O’Reilly, 2005.

[42] “Apache MINA.” [Online]. Available: https://mina.apache.org/. [Accessed: 27-May-2019].
[43] A. Gupta, *Java EE 7 essentials*, 1st ed. Sebastopol, CA: O’Reilly & Associates Inc, 2013.
[44] “GitHub - Apache Deltaspike.” [Online]. Available: https://github.com/apache/deltaspike. [Accessed: 27-May-2019].
[45] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, “Natural language processing: An introduction,” *J. Am. Med. Inform. Assoc.*, vol. 18, no. 5, pp. 544–551, Sep. 2011.
[46] “Apache OpenNLP Developer Documentation.” [Online]. Available: https://opennlp.apache.org/docs/1.9.1/manual/opennlp.html. [Accessed: 27-May-2019].
[47] “GitHub - Apache Maven.” [Online]. Available: https://github.com/apache/maven. [Accessed: 27-May-2019].
[48] “Apache WSS4J.” [Online]. Available: https://ws.apache.org/wss4j/. [Accessed: 27-May-2019].
[49] “What is PDF? Adobe Portable Document Format | Adobe Acrobat DC.” [Online]. Available: https://acrobat.adobe.com/us/en/acrobat/about-adobe-pdf.html. [Accessed: 27-May-2019].
[50] “Apache PDFBox | A Java PDF Library.” [Online]. Available: https://pdfbox.apache.org/. [Accessed: 27-May-2019].
[51] “GitHub - Apache PDFBox.” [Online]. Available: https://github.com/apache/pdfbox. [Accessed: 27-May-2019].
[52] “Apache(tm) FOP.” [Online]. Available: https://xmlgraphics.apache.org/fop/. [Accessed: 08-Jun-2019].
[53] D. Tidwell, “XSL Formatting Objects (XSL-FO), Part 1:Get thebasics of XSL-FO techniquesConvert HTML documents into formatting objects and then intoPDF files.” IBM, 04-Feb-2003.
[54] “GitHub - Apache FOP.” [Online]. Available: https://github.com/apache/fop. [Accessed: 08-Jun-2019].
[55] “Apache Cayenne.” [Online]. Available: https://cayenne.apache.org/. [Accessed: 08-Jun-2019].
[56] C. Ireland, M. Newton, D. Bowers, and K. Waugh, “Understanding object-relational mapping: A framework based approach,” *Int. J. Adv. Softw.*, vol. 2, pp. 202–216, Jan. 2019.
[57] “GitHub - Apache cayenne.” [Online]. Available: https://github.com/apache/cayenne. [Accessed: 08-Jun-2019].
[58] Mohd Zamri Murah, “Teaching and Learning Cloud Computing,” *Procedia - Soc. Behav. Sci.*, vol. 59, pp. 157–163, Oct. 2012.
[59] “Apache jclouds.” [Online]. Available: https://jclouds.apache.org/. [Accessed: 08-Jun-2019].
[60] “GitHub - Apache jclouds.” [Online]. Available: https://github.com/apache/jclouds. [Accessed: 08-Jun-2019].
[61] “Apache OpenJPA.” [Online]. Available: http://openjpa.apache.org/. [Accessed: 08-Jun-2019].
[62] “Web API - Extend - Doc SonarQube.” [Online]. Available: https://docs.sonarqube.org/display/DEV/Web+API. [Accessed: 08-Jun-2019].
[63] “Issues | SonarQube Docs.” [Online]. Available: https://docs.sonarqube.org/latest/user-guide/issues/. [Accessed: 08-Jun-2019].
[64] “Metric Definitions | SonarQube Docs.” [Online]. Available: https://docs.sonarqube.org/latest/user-guide/metric-definitions/. [Accessed: 08-Jun-2019].
[65] “SQUORING Technologies | Squore Software Analytics.” [Online]. Available: https://www.squoring.com/en/produits/squore-software-analytics/. [Accessed: 08-Jun-2019].
[66] K. Krippendorff, “Computing Krippendorff ’s Alpha-Reliability.” 2011.
[67] N. J. Salkind, Ed., *Encyclopedia of research design*. Thousand Oaks, Calif: SAGE Publications, 2010.
[68] A. Cutler and L. Breiman, “Archetypal Analysis,” *Technometrics*, vol. 36, no. 4, pp. 338–347, Nov. 1994.
[69] R. C. Schmidt, “Managing Delphi Surveys Using Nonparametric Statistical Techniques,” *Decis. Sci.*, vol. 28, no. 3, pp. 763–774, Jul. 1997.

[70] J. Pinheiro and D. Bates, “Linear Mixed-Effects Models: Basic Concepts and Examples,” in *Mixed-Effects Models in S and S-PLUS*, New York: Springer-Verlag, 2000, pp. 3–56.