



Διεθνές Πανεπιστήμιο της Ελλάδος

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**ΑΝΑΠΤΥΞΗ DASHBOARD ΓΙΑ ΤΗΝ
ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΤΗΣ “ΥΓΕΙΑΣ” ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ
(SOFTWARE HEALTH) ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ**



Φοιτήτρια:
Αντωνίου Σοφία
2020011

Επιβλέπων:
Αρβανίτου Ελβίρα-Μαρία

Θεσσαλονίκη, Σεπτέμβριος 2026

Τίτλος Δ.Ε. Ανάπτυξη dashboard για την παρακολούθηση της “υγείας” του λογισμικού (software health) σε πραγματικό χρόνο
Κωδικός Δ.Ε. 25374

Ονοματεπώνυμο φοιτήτριας Αντωνίου Σοφία
Ονοματεπώνυμο εισηγητή Αρβανίτου Ελβίρα-Μαρία
Ημερομηνία ανάληψης Δ.Ε. 05/02/2026
Ημερομηνία περάτωσης Δ.Ε. 06/09/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία της φοιτήτριας Αντωνίου Σοφίας που την εκπόνησε/αν. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραίτητα και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Η επιλογή του θέματος της παρούσας διπλωματικής εργασίας προέκυψε από το ενδιαφέρον μου για την ποιότητα λογισμικού και για τον τρόπο με τον οποίο μπορούν να αξιοποιηθούν τα σύγχρονα εργαλεία ανάλυσης κώδικα για την υποστήριξη της ανάπτυξης και της συντήρησης εφαρμογών. Κατά την ανάπτυξη ενός λογισμικού, η σταδιακή αύξηση της πολυπλοκότητας, η εμφάνιση προβλημάτων συντηρησιμότητας και το τεχνικό χρέος αποτελούν ζητήματα που μπορούν να επηρεάσουν σημαντικά τόσο την ποιότητα του κώδικα όσο και τη μελλοντική εξέλιξη ενός έργου. Για τον λόγο αυτό, θεώρησα ιδιαίτερα ενδιαφέρονσα τη μελέτη μεθόδων που επιτρέπουν την οργανωμένη συλλογή, επεξεργασία και παρουσίαση διαφορετικών μετρικών ποιότητας λογισμικού. Η δυνατότητα συνδυασμού δεδομένων από εργαλεία στατικής ανάλυσης, όπως το SonarQube και το AllinCode, με πληροφορίες από το ιστορικό ανάπτυξης ενός αποθετηρίου Git αποτέλεσε βασικό κίνητρο για την επιλογή του συγκεκριμένου θέματος. Παράλληλα, η εργασία μου έδωσε τη δυνατότητα να συνδυάσω θεωρητικές γνώσεις σχετικά με την ποιότητα λογισμικού και το τεχνικό χρέος, με την πρακτική ανάπτυξη ενός ολοκληρωμένου συστήματος. Στόχος ήταν η δημιουργία μιας εφαρμογής που δεν περιορίζεται στην απλή παρουσίαση μετρικών, αλλά διευκολύνει την κατανόηση της συνολικής κατάστασης ενός έργου και τον εντοπισμό περιοχών που ενδέχεται να απαιτούν μεγαλύτερη προσοχή.

Περίληψη

Η συνεχής εξέλιξη και συντήρηση ενός συστήματος λογισμικού οδηγεί συχνά σε συσσώρευση τεχνικού χρέους, αυξημένη πολυπλοκότητα και σταδιακή υποβάθμιση της ποιότητας του κώδικα. Καθώς ένα έργο μεγαλώνει, οι επαναλαμβανόμενες αλλαγές, οι βιαστικές σχεδιαστικές επιλογές και η προσθήκη νέων λειτουργιών μπορεί να επιβαρύνουν με την πάροδο του χρόνου ορισμένα τμήματα κώδικα. Ως αποτέλεσμα, τα τμήματα αυτά μπορεί να χρειάζονται περισσότερη προσπάθεια συντήρησης και να είναι πιο επιρρεπή σε προβλήματα. Για τον λόγο αυτό, η συστηματική μέτρηση και παρακολούθηση της κατάστασης ενός έργου λογισμικού είναι βασική προϋπόθεση για την έγκαιρη αναγνώριση και διαχείριση αυτής της επιβάρυνσης. Τα εργαλεία στατικής ανάλυσης δίνουν πλήθος μετρικών για την πολυπλοκότητα, τη συντηρησιμότητα και το τεχνικό χρέος. Η αξιολόγηση όμως μόνο με βάση στατικά χαρακτηριστικά δεν δείχνει πάντα τη συνολική εικόνα ενός έργου. Η συχνότητα των αλλαγών, η έκταση τους και ο αριθμός των προγραμματιστών που επεμβαίνουν σε ένα αρχείο μπορούν κι αυτά να αποτελέσουν ενδείξεις αυξημένου τεχνικού κινδύνου. Ακόμη, οι δομικές ιδιότητες των κλάσεων και οι σχέσεις ανάμεσα στα επιμέρους στοιχεία του συστήματος δίνουν επίσης χρήσιμη πληροφορία για τη συντηρησιμότητα του κώδικα.

Στην παρούσα διπλωματική εργασία αναπτύσσεται ένα ολοκληρωμένο σύστημα ανάλυσης και οπτικοποίησης της ποιότητας έργων λογισμικού. Για τον σκοπό αυτό, συνδυάζονται δεδομένα από το SonarQube, το AllinCode και το ιστορικό Git ενός αποθετηρίου. Οι διαφορετικές πηγές ενοποιούνται σε επίπεδο αρχείου και χρησιμοποιούνται για τον υπολογισμό επιμέρους δεικτών κινδύνου, καθώς και ενός συνολικού δείκτη υγείας του έργου. Τα αποτελέσματα αποθηκεύονται και παρουσιάζονται μέσα από ένα ενοποιημένο dashboard, που επιτρέπει τη συνολική αξιολόγηση ενός έργου αλλά και τον εντοπισμό αρχείων που χρειάζονται αυξημένη προσοχή. Σκοπός του συστήματος είναι να μετατραπεί ένα σύνολο ετερογενών τεχνικών δεδομένων σε πληροφορία χρήσιμη για τη συντήρηση και τη λήψη αποφάσεων.

« Development of a dashboard for monitoring software health in real time »

Antoniou Sophia

Abstract

The continuous evolution and maintenance of a software system often leads to the accumulation of technical debt, increased complexity and the gradual degradation of code quality. As a project grows, repeated changes, rushed design decisions, and the addition of new functionality can place an increasing burden on the codebase over time. As a result, certain parts of the system may require greater maintenance effort and become more prone to problems. For this reason, systematic measurement and monitoring of software health has become an important aspect of modern software development. Static analysis tools provide a wide range of metrics related to complexity, maintainability, and technical debt. However, evaluating a software project solely on the basis of static characteristics does not always provide a complete picture of its overall condition. The frequency of changes, their extent, and the number of developers modifying a file can also serve as indicators of increased technical risk. In addition, the structural properties of classes and the relationships among the individual components of a system provide further information about code maintainability.

This thesis presents the development of an integrated system for software quality analysis and visualization. To achieve this, data from SonarQube, AllinCode, and the Git history of a repository are combined. These different data sources are integrated at the file level and used to calculate individual risk indicators, as well as an overall project health score. The results are stored and presented through an interactive dashboard, which enables both the overall assessment of a project and the identification of files that require increased attention. The purpose of the system is to transform a set of heterogeneous technical data into information useful for maintenance and decision-making.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτριά μου για την καθοδήγηση και τις χρήσιμες συμβουλές της καθ' όλη τη διάρκεια εκπόνησης της εργασίας. Ακόμη, ευχαριστώ την οικογένεια και τους ανθρώπους μου για τη διαρκή υποστήριξη, την υπομονή και την ενθάρρυνσή τους, που αποτέλεσαν σημαντική βοήθεια καθ' όλη τη διάρκεια αυτής της προσπάθειας.

Περιεχόμενα

Πρόλογος.....	iii
Περίληψη	iv
Abstract	v
Ευχαριστίες	vi
Περιεχόμενα.....	vii
Κατάλογος Εικόνων	xi
Κατάλογος Αποσπασμάτων Κώδικα.....	xi
Κατάλογος Πινάκων	xii
Κεφάλαιο 1ο: Εισαγωγή	1
1.1 Κίνητρο Εργασίας.....	1
1.2 Ορολογία Τεχνικού Χρέους.....	2
1.3 Αντικείμενο και Στόχος	4
1.4 Δομή της Εργασίας	5
Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Σχετικές Εργασίες.....	6
2.1 Μέτρηση και διαχείριση Τεχνικού Χρέους.....	6
2.2 Ποιότητα και Συντηρησιμότητα Λογισμικού	7
2.2.1 Το Πρότυπο ISO/IEC 25010	7
2.2.2 Η Σημασία της Συντηρησιμότητας στον Κύκλο Ζωής του Λογισμικού.....	9
2.3 Στατική Ανάλυση Κώδικα	10
2.3.1 Αρχές Στατικής Ανάλυσης.....	10
2.3.2 Εντοπισμός Οσμών Κώδικα (Code Smells) και Παραβιάσεων Κανόνων	11
2.4 Συμπεριφορική Ανάλυση Κώδικα (Behavioral Code Analysis).....	12
2.4.1 Η Έννοια των Hotspots	12
2.4.2 Μεταβολή Κώδικα (Code Churn) και Μετρικές Git (Git Metrics): Πώς Η Δραστηριότητα των Προγραμματιστών Επηρεάζει την Υγεία του Κώδικα.....	13
2.5 Σχετικά Εργαλεία και Πλατφόρμες.....	13
2.5.1 Επαγγελματικά Εργαλεία	14
2.5.2 Ερευνητικές Πλατφόρμες.....	15
2.5.3 Σύγκριση και Ανάγκη για ένα Ενοποιημένο Dashboard.....	15
Κεφάλαιο 3ο: Μεθοδολογία και Σχεδιασμός Συστήματος.....	18
3.1 Αρχιτεκτονική Συστήματος	18
3.1.1 Σχεδιασμός του Back-end	19
3.1.2 Σχεδιασμός του Μηχανισμού Επεξεργασίας.....	19

3.1.3	Σχεδιασμός της Παρουσίασης.....	20
3.2	Διαδικασία Συλλογής Δεδομένων.....	20
3.2.1	Πηγές και δεδομένα.....	20
3.2.2	Ροή Εκτέλεσης μιας Ανάλυσης.....	20
3.3	Το Μοντέλο Αξιολόγησης	21
3.3.1	Κανονικοποίηση Δεδομένων και Βασικοί Κανόνες Υπολογισμού	21
3.3.1.1	Κριτήρια Επιλεξιμότητας και Πληθυσμοί Αρχείων	22
3.3.1.2	Κανονικοποίηση με Βάση το P95	22
3.3.1.3	Μετατροπή σε Κλίμακα 0–100	23
3.3.1.4	Συνάρτηση Περιορισμού Clamp	23
3.3.1.5	KLOC και Πυκνότητες.....	23
3.3.1.6	Επίπεδα Κινδύνου.....	24
3.3.1.7	Απουσία High-TD Πρόβλεψης.....	24
3.3.2	Υπολογισμός Επιμέρους Δεικτών σε Επίπεδο Αρχείου.....	24
3.3.2.1	Git Activity Score.....	24
3.3.2.2	Complexity Score	25
3.3.2.3	Hotspot Score	26
3.3.2.4	Structural Risk Score.....	26
3.3.2.5	Quality Risk Score.....	27
3.3.2.6	High-TD Risk Score	27
3.3.2.7	Refactoring Priority Score	28
3.3.2.8	Μη Αντισταθμιστικός Κανόνας Κινδύνου	28
3.3.3	Υπολογισμός Επιμέρους Δεικτών σε Επίπεδο Έργου.....	29
3.3.3.1	Technical Debt Project Risk.....	29
3.3.3.2	Complexity Project Risk.....	30
3.3.3.3	Quality Project Risk.....	30
3.3.3.4	Duplication Project Risk.....	31
3.3.3.5	Project Risk Count.....	31
3.3.3.6	Hotspot Project Risk.....	31
3.3.3.7	Structural Project Risk.....	32
3.3.3.8	High-TD Project Risk.....	32
3.3.4	Ορισμός του Global Health Score.....	33
Κεφάλαιο 4ο:	Υλοποίηση Dashboard.....	35
4.1	Τεχνολογίες Ανάπτυξης	35

4.1.1	Τοπικό Περιβάλλον Εκτέλεσης.....	35
4.1.2	Τεχνολογίες του Back-end	35
4.1.3	Python και Εξόρυξη Δεδομένων από το Git	35
4.1.4	Τεχνολογίες του Front-end.....	36
4.1.5	Οργάνωση του Πηγαίου Κώδικα	36
4.2	Ανάπτυξη της Μηχανισμού Επεξεργασίας (Analytics Script).....	37
4.2.1	Συλλογή και Προετοιμασία των Δεδομένων Εισόδου	37
4.2.2	Κανονικοποίηση των Διαδρομών.....	40
4.2.3	Ενοποίηση των Τριών Πηγών	42
4.2.4	Έλεγχος Επιλεξιμότητας	44
4.2.5	Κανονικοποίηση P95 και Υπολογισμός Δεικτών σε Επιπέδο Αρχείου	46
4.2.6	Υπολογισμός Δεικτών σε Επίπεδο Έργου	49
4.2.7	Μορφή Εξόδου της Ροής Επεξεργασίας	50
4.3	Σχεδιασμός δεδομένων και Διαχείριση Ιστορικότητας.....	50
4.3.1	Σχήμα της Βάσης Δεδομένων	50
4.3.1.1	Πίνακας Projects.....	50
4.3.1.2	Πίνακας Analysis Runs	50
4.3.1.3	Πίνακας Merged File Metrics.....	51
4.3.1.4	Πίνακας Project Trend History.....	51
4.3.2	Υλοποίηση και Σύνδεση της MariaDB	52
4.3.2.1	Διαχείριση Συνδέσεων με τη Βάση Δεδομένων (Connection Pool)	52
4.3.2.2	Αποθήκευση Δεδομένων	53
4.3.2.3	Αποθήκευση στον Πίνακα analysis_run.....	55
4.3.2.4	Αποθήκευση στον Πίνακα Merged File Metrics	55
4.3.3	Snapshots, Ιστορικό και Πολιτική Διατήρησης	55
4.4	Διεπαφή Χρήστη και Οπτικοποιήσεις.....	55
4.4.1	Σχεδιαστική Προσέγγιση και Δομή Πλοήγησης.....	55
4.4.2	Υλοποίηση του Front-end	56
4.4.3	Υλοποίηση των Οθονών του Dashboard.....	56
4.4.3.1	Οθόνη UI-01: Αρχική Σελίδα.....	56
4.4.3.2	Οθόνη UI-02 Overview	57
4.4.3.3	Οθόνη UI-03: Temporal Evolution	58
4.4.3.4	Οθόνη UI-03: Περιοχή High-TD Files.....	58
4.4.3.5	Οθόνη UI-04 Hotspots.....	59

4.4.3.6	Οθόνη UI-04: Πίνακας Refactoring Priority	60
4.4.3.7	Οθόνη UI-05 Quality & Structure	61
4.4.3.8	Οθόνη UI-05: Πίνακας Structure Issues per File	61
4.4.3.9	Οθόνη UI-06 All File Metrics	62
4.4.3.10	Οθόνη UI-06: Εφαρμογή Φίλτρων	63
Κεφάλαιο 5ο:	Μελέτη Περίπτωσης και Αποτελέσματα	64
5.1	Επιλογή και Περιγραφή των Έργων Βαθμονόμησης.....	64
5.2	Παρουσίαση Αποτελεσμάτων μέσω Dashboard	64
5.3	Ανάλυση Αποτελεσμάτων.....	65
5.4	Βαθμονόμηση και ανάλυση ευαισθησίας του μοντέλου.....	67
5.4.1	Αρχική Διαμόρφωση και Αρχικά Αποτελέσματα	67
5.4.2	Σύγκριση Εναλλακτικών Τύπων Hotspot Score	68
5.4.3	Βαθμονόμηση του Βάρους του Hotspot Risk	69
5.4.4	Βαθμονόμηση των Complexity Thresholds	70
5.4.5	Αναθεώρηση του Technical Debt Project Risk.....	71
5.4.6	Κατώφλια Ποιότητας	71
5.4.7	Κατώφλια Επανάληψης Κώδικα	72
5.4.8	Υπολογισμός του High-TD Project Risk	73
5.4.8.1	Έλεγχος του Κατωφλίου για το High-td Ratio και της Στάθμισης	74
5.4.9	Διόρθωση της Συγκέντρωσης των Project-Level Risks.....	76
5.4.9.1	Σύγκριση Εναλλακτικών Ποσοστών Εστίασης.....	76
5.4.10	Ανάλυση των Δομικών Βαρών	77
5.4.11	Τελική Διαμόρφωση και Αποτελέσματα	77
Κεφάλαιο 6ο:	Συμπεράσματα και Μελλοντικές επεκτάσεις	83
6.1	Γενική Συζήτηση και Ερμηνεία Ευρημάτων	83
6.2	Απειλές Εγκυρότητας.....	83
6.3	Προτάσεις για Μελλοντική Βελτίωση	84
6.3.1	Εξέλιξη της Μεθοδολογίας	84
6.3.2	Τεχνική Εξέλιξη της Εφαρμογής.....	85
BIBΛΙΟΓΡΑΦΙΑ		86
ΠΑΡΑΡΤΗΜΑ Α: ΟΔΗΓΙΕΣ ΕΓΚΑΤΑΣΤΑΣΗΣ ΚΑΙ ΕΚΤΕΛΕΣΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ.....		90

Κατάλογος Εικόνων

Εικόνα 2.1 - Διαγραμματική απεικόνιση ISO/IEC 25010.....	8
Εικόνα 2.2 - CodeScene Dashboard: Παράδειγμα ανάλυσης έργου.....	14
Εικόνα 3.1 - Συνολική αρχιτεκτονική του συστήματος.....	18
Εικόνα 3.2 - Ροή εκτέλεσης μιας ανάλυσης.....	21
Εικόνα 4.1 - Σχηματική απεικόνιση των πινάκων της βάσης δεδομένων.....	52
Εικόνα 4.2 - Οθόνη UI-01 Αρχική Σελίδα του dashboard.....	57
Εικόνα 4.3 - Οθόνη UI-02 Overview του έργου sling-whiteboard.....	57
Εικόνα 4.4 - Οθόνη UI-03 Temporal Evolution του έργου sling-whiteboard	58
Εικόνα 4.5 - Οθόνη UI-03: Περιοχή High-TD Risk Files για το έργο sling-whiteboard.....	59
Εικόνα 4.6 - Οθόνη UI-04 Hotspots του έργου sling-whiteboard.....	60
Εικόνα 4.7 - Οθόνη UI-04: Επιλογή αρχείου στο διάγραμμα φυσαλίδας για το έργο sling-whiteboard.....	60
Εικόνα 4.8 - Οθόνη UI-04: Πίνακας Refactoring Priority του έργου sling-whiteboard.....	61
Εικόνα 4.9 - Οθόνη UI-05 Quality & Structure του έργου sling-whiteboard.....	61
Εικόνα 4.10 - Οθόνη UI-05: Πίνακας Structure Issues per File του έργου sling-whiteboard.....	62
Εικόνα 4.11 - Οθόνη UI-06 All File Metrics του έργου sling-whiteboard.....	63
Εικόνα 4.12 - Οθόνη UI-06: Παράδειγμα εφαρμογής φίλτρων στα δεδομένα του έργου sling-whiteboard.....	63
Εικόνα 5.1 - Οθόνη Overview του έργου maven-archiver.....	65
Εικόνα 5.2 - Οθόνη Overview του έργου sling-whiteboard.....	65
Εικόνα 5.3 - Οθόνη Overview του έργου commons-csv.....	79
Εικόνα 5.4 - Οθόνη Overview του έργου velocity-tools.....	80

Κατάλογος Αποσπασμάτων Κώδικα

Απόσπασμα κώδικα 4.1: Συλλογή δεδομένων και δημιουργία της κοινής εισόδου του analytics pipeline(server.js).....	38-40
Απόσπασμα κώδικα 4.2: Εξαγωγή και κανονικοποίηση των paths από τις διαφορετικές πηγές δεδομένων(analytics_pipeline.py).....	40-42
Απόσπασμα κώδικα 4.3: Δημιουργία κοινής εγγραφής και συγχώνευση των AllinCode μετρικών ανά path(analytics_pipeline.py).....	42-44
Απόσπασμα κώδικα 4.4: Έλεγχος επιλεξιμότητας και σχηματισμός του κοινού πληθυσμού αρχείων (scoring.py).....	44-46
Απόσπασμα κώδικα 4.5: Υπολογισμός του Refactoring Priority Score και εφαρμογή του μη αντισταθμιστικού κανόνα(scoring.py).....	47-48

Απόσπασμα κώδικα 4.6: Υπολογισμός του Overall Project Risk και του Global Health Score(scoring.py).....	49-50
Απόσπασμα κώδικα 4.7: Αποθήκευση των αποτελεσμάτων ανάλυσης μέσα σε συναλλαγή βάσης δεδομένων(saveAnalysis.js).....	53-54

Κατάλογος Πινάκων

Πίνακας 4.1: Κύρια αρχεία και ευθύνες της εφαρμογής.....	37
Πίνακας 5.1: Οι επτά project-level κίνδυνοι, το Overall Project Risk και το Global Health Score για το Maven Archiver και το Sling Whiteboard	64
Πίνακας 5.2: Συνολικός κίνδυνος του έργου.....	68
Πίνακας 5.3: Σύγκριση Εναλλακτικών Τύπων Hotspot Score	69
Πίνακας 5.4: Σύγκριση Διαμόρφωσης	69
Πίνακας 5.5: Δοκιμή τιμών για την επιλογή Complexity / Cognitive Threshold.....	70
Πίνακας 5.6: Technical Debt Project Risk	71
Πίνακας 5.7: Technical Debt Project Risk με βάση την κλίμακα συντηρησιμότητας.....	71
Πίνακας 5.8: Κατώφλια Ποιότητας.....	72
Πίνακας 5.9: Δοκιμαστικές τιμές για την επιλογή κατωφλιού επανάληψης κώδικα	72
Πίνακας 5.10: Ανάλυση συνιστωσών High-TD Project Risk για τα 2 έργα	74
Πίνακας 5.11: High-TD Ratio Threshold και Στάθμισης.....	75
Πίνακας 5.12: High-TD Risk υπό διαφορετικά βάρη στάθμισης Ratio.....	75
Πίνακας 5.13: Εναλλακτικά Ποσοστά Εστίασης.....	77
Πίνακας 5.14: Υπολογισμός πίνακα με κοινό πληθυσμό και τελικό κανόνα συγκέντρωσης	77
Πίνακας 5.15: Πίνακας Βαρών Global Health Score.....	78
Πίνακας 5.16: Αρχικές και βαθμολογιμένες τιμές Global Health Score.....	79
Πίνακας 5.17: Αποτελέσματα ανάλυσης ευαισθησίας Global Health Score.....	80
Πίνακας 5.18: Αποτελέσματα ανάλυσης ευαισθησίας και σταθερότητας για τα έργα Velocity Tools και Commons CSV	81

Κεφάλαιο 1ο: Εισαγωγή

1.1 Κίνητρο Εργασίας

Η ανάπτυξη λογισμικού δεν ολοκληρώνεται με την πρώτη λειτουργική έκδοση ενός συστήματος. Τα περισσότερα συστήματα εξελίσσονται συνεχώς, καθώς προστίθενται νέες λειτουργίες, μεταβάλλονται οι απαιτήσεις, διορθώνονται σφάλματα και προσαρμόζεται ο κώδικας σε νέες ανάγκες. Οι ομάδες ανάπτυξης καλούνται να εξισορροπούν διαφορετικές προτεραιότητες, όπως η ποιότητα, ο διαθέσιμος χρόνος, το κόστος ανάπτυξης και η ανάγκη γρήγορης παράδοσης νέων λειτουργιών. Μια απόφαση που φαίνεται λογική μια συγκεκριμένη στιγμή, μπορεί αργότερα να γίνει πηγή περιορισμών ή επιπλέον κόπου. Η συσσώρευση τέτοιων τεχνικών επιβαρύνσεων συνδέεται με την έννοια του Τεχνικού Χρέους (Technical Debt). Η έννοια αυτή δεν δηλώνει απαραίτητα λανθασμένο ή μη λειτουργικό κώδικα. Ένα σύστημα μπορεί να ικανοποιεί τις λειτουργικές απαιτήσεις του και ταυτόχρονα να περιλαμβάνει εσωτερικές αδυναμίες. Οι αδυναμίες αυτές, κάνουν την κατανόηση, την τροποποίηση ή την επέκτασή του δυσκολότερη. Αυξημένη πολυπλοκότητα, ισχυρή σύζευξη, περιορισμένη συνοχή, επαναλαμβανόμενος κώδικας, οσμές κώδικα (code smells) ή άλλες αδυναμίες μπορούν να αυξήσουν σταδιακά την προσπάθεια που απαιτείται για τη συντήρηση[4]. Το πρόβλημα λοιπόν δεν βρίσκεται μόνο στην ύπαρξη μιας τεχνικής αδυναμίας, αλλά και στο πως αυτή επηρεάζει τη μελλοντική εξέλιξη του λογισμικού.

Η πλήρης εξάλειψη κάθε μορφής τεχνικού χρέους δεν αποτελεί πάντα ούτε εφικτό ούτε επιθυμητό στόχο. Σε πραγματικά έργα λογισμικού, οι διαθέσιμοι πόροι είναι περιορισμένοι. Έτσι, η ομάδα ανάπτυξης πρέπει να αποφασίζει πότε μια τεχνική παρέμβαση δικαιολογείται σε σχέση με άλλες ανάγκες του συστήματος. Μια προσωρινή τεχνική συντόμευση μπορεί να είναι αποδεκτή όταν επιτρέπει την έγκαιρη παράδοση μιας κρίσιμης λειτουργίας, αρκεί οι συνέπειές της να είναι κατανοητές και να παρακολουθούνται. Αντίστοιχα, η ύπαρξη ενός προβληματικού τμήματος κώδικα δεν σημαίνει ότι αυτό πρέπει υποχρεωτικά να αναδομηθεί άμεσα. Η πρακτική σημασία του εξαρτάται από το κόστος αποκατάστασης, τη σοβαρότητα της τεχνικής αδυναμίας και κυρίως από το κατά πόσο το συγκεκριμένο τμήμα κώδικα αναμένεται να συμμετέχει σε μελλοντικές αλλαγές [5]. Το γεγονός αυτό μετατρέπει τη διαχείριση του τεχνικού χρέους από πρόβλημα απλής αναγνώρισης σε πρόβλημα αξιολόγησης και προτεραιοποίησης. Σε ένα πραγματικό αποθετήριο (repository) μπορεί να εντοπιστεί μεγάλος αριθμός οσμών κώδικα, παραβιάσεων κανόνων, πολύπλοκων αρχείων, δομικών αδυναμιών και περιοχών αυξημένης δραστηριότητας [8]. Η ύπαρξη όλων αυτών των πληροφοριών όμως, δεν απαντά από μόνη της στο βασικό ερώτημα: *ποια από αυτά τα στοιχεία πρέπει να εξεταστούν πρώτα;* Απαιτείται επομένως ένας τρόπος εντοπισμού των περιοχών στις οποίες ο συνδυασμός τεχνικής επιβάρυνσης και πραγματικής δραστηριότητας καθιστά μια παρέμβαση πιο σημαντική.

Μία ακόμη δυσκολία είναι ότι η τεχνική κατάσταση ενός έργου λογισμικού δεν μπορεί να εκφραστεί επαρκώς μέσω μίας μόνο μετρικής. Ο πηγαίος κώδικας μπορεί να εξεταστεί ως προς την πολυπλοκότητα, τη διπλοτυπία, τις οσμές κώδικα, τα σφάλματα (bugs) και την εκτιμώμενη προσπάθεια αποκατάστασης. Επιπλέον, μια αντικειμενοστραφής δομή μπορεί να αξιολογηθεί ως προς τη σύζευξη, τη συνοχή, τον αριθμό μεθόδων και άλλα χαρακτηριστικά των κλάσεων. Παράλληλα, το ιστορικό ενός αποθετηρίου αποκαλύπτει ποια αρχεία μεταβάλλονται συχνότερα, πόσο έντονες είναι οι αλλαγές και πόσα άτομα έχουν συμμετάσχει στην εξέλιξή τους. Οι πληροφορίες αυτές δεν είναι εναλλάξιμες, αλλά περιγράφουν διαφορετικές διαστάσεις του ίδιου συστήματος. Συνεπώς, η ουσιαστική αποτίμηση του τεχνικού χρέους απαιτεί τη σύνθεση δεδομένων που αφορούν τόσο την τρέχουσα εσωτερική κατάσταση του λογισμικού όσο και τον τρόπο με τον οποίο αυτό εξελίσσεται [2]. Η ανάγκη συνδυαστικής

αξιολόγησης δημιουργεί όμως ένα δεύτερο πρόβλημα: όσο αυξάνεται ο αριθμός των μετρικών, τόσο δυσκολότερη γίνεται η ερμηνεία τους. Τα εργαλεία ανάλυσης μπορούν να παράγουν μεγάλο αριθμό τιμών, παραβιάσεων και επιμέρους αποτελεσμάτων. Η διαθεσιμότητα περισσότερων δεδομένων όμως, δεν συνεπάγεται αυτόματα και καλύτερη υποστήριξη αποφάσεων. Αντίθετως, όταν οι πληροφορίες παρουσιάζονται ως ένα μεγάλο σύνολο ασύνδετων αριθμών, ο χρήστης αναγκάζεται να πραγματοποιήσει ο ίδιος τη σύνθεση, να αποφασίσει ποια μέτρηση είναι σημαντικότερη και να μετακινηθεί μεταξύ διαφορετικών εργαλείων και επιπέδων ανάλυσης [6].

Η υποστήριξη της προτεραιοποίησης δεν σημαίνει βέβαια ότι η τελική απόφαση αυτοματοποιείται. Οι μετρικές δεν γνωρίζουν τη λειτουργική σημασία ενός αρχείου, τις επιχειρησιακές απαιτήσεις, το κόστος μιας παρέμβασης ή τους περιορισμούς που αντιμετωπίζει μια ομάδα ανάπτυξης. Μια αναδόμηση κώδικα (refactoring) που είναι επιθυμητή από άποψη εσωτερικής δομής, μπορεί να μην είναι η σωστή επιλογή όταν υπάρχουν πιο πιεστικές λειτουργικές ή χρονικές ανάγκες [9]. Γι' αυτό οι αριθμητικές αξιολογήσεις και οι κατατάξεις πρέπει να λειτουργούν ως ενδείξεις που οργανώνουν την πληροφορία και καθοδηγούν την ανθρώπινη διερεύνηση. Από αυτή την ανάγκη προκύπτει το βασικό κίνητρο της εργασίας. Διαφορετικά εργαλεία παρέχουν πληροφορίες για συγκεκριμένες πτυχές της τεχνικής κατάστασης ενός συστήματος, όμως τα αποτελέσματά τους παραμένουν καταναμεμημένα σε διαφορετικές πηγές. Για τον λόγο αυτό, χρειάζεται ένα ενιαίο περιβάλλον που θα συγκεντρώνει και θα οργανώνει τις διαθέσιμες πληροφορίες με τρόπο τέτοιο, ώστε να διευκολύνει την αξιολόγησή τους [7].

1.2 Ορολογία Τεχνικού Χρέους

Σύμφωνα με τον W. Cunningham, ο όρος Τεχνικό Χρέος χρησιμοποιείται ως μεταφορά για την περιγραφή των μακροπρόθεσμων συνεπειών τεχνικών αποφάσεων ή ανεπαρκειών που μπορούν να προσφέρουν ένα άμεσο όφελος, αυξάνοντας όμως τη μελλοντική προσπάθεια ανάπτυξης και συντήρησης [10]. Μία ομάδα ανάπτυξης λογισμικού μπορεί να επιλέξει μια γρηγορότερη ή απλούστερη υλοποίηση ώστε να ικανοποιήσει μια προθεσμία, παρόλο που γνωρίζει ότι η λύση αυτή δεν έχει την καλύτερη δυνατή εσωτερική ποιότητα. Μια τέτοια απόφαση δεν είναι απαραίτητα λανθασμένη. Το πρόβλημα εμφανίζεται όταν οι συνέπειες της δεν παρακολουθούνται. Σε αυτή την περίπτωση, η συσσωρευμένη τεχνική επιβάρυνση αρχίζει να δυσκολεύει τη μελλοντική εξέλιξη και συντήρηση του συστήματος.

Η οικονομική μεταφορά είναι χρήσιμη, καθώς αποτυπώνει τη σχέση μεταξύ άμεσου οφέλους και μελλοντικού κόστους. Όπως η ανάλυση οικονομικού χρέους επιτρέπει την αξιοποίηση ενός πόρου στο παρόν με αντάλλαγμα μια μελλοντική υποχρέωση, έτσι και μια τεχνική συντόμευση μπορεί να επιτρέψει ταχύτερη ανάπτυξη με αντάλλαγμα αυξημένη προσπάθεια σε επόμενες τροποποιήσεις. Η αναλογία αυτή βέβαια, δεν σημαίνει ότι κάθε τεχνική αδυναμία μπορεί να μετατραπεί με ακρίβεια σε χρηματική αξία [5]. Λειτουργεί κυρίως ως πλαίσιο κατανόησης και λήψης αποφάσεων, επιτρέποντας να εξεταστούν οι συνέπειες των τεχνικών επιλογών σε βάθος χρόνου. Μια βασική διάκριση αφορά το σκόπιμο (deliberate) και το ακούσιο (inadvertent) τεχνικό χρέος. Στην πρώτη περίπτωση, η ομάδα γνωρίζει ότι υιοθετεί μια τεχνικά υποδεέστερη λύση και αποδέχεται συνειδητά τις πιθανές συνέπειές της για να αποκτήσει ένα βραχυπρόθεσμο όφελος. Στην περίπτωση του ακούσιου χρέους αντίθετα, η επιβάρυνση δεν αποτελεί συνειδητή επιλογή. Μπορεί να προκύψει από ανεπαρκή γνώση, αλλαγή απαιτήσεων, εξέλιξη της τεχνολογίας ή μεταβολή του περιβάλλοντος στο οποίο λειτουργεί το σύστημα. Ένας σχεδιασμός που ήταν ικανοποιητικός στην αρχική ανάπτυξη, μπορεί να γίνει σταδιακά περιορισμός, χωρίς να έχει προηγηθεί καμία εσκεμμένη συντόμευση.

Κεντρική έννοια της οικονομικής μεταφοράς είναι και το κεφάλαιο (principal). Το κεφάλαιο αντιπροσωπεύει την προσπάθεια που απαιτείται για την αποκατάσταση μιας τεχνικής ανεπάρκειας. Ανάλογα την προσέγγιση, μπορεί να εκφράζεται ως χρόνος διόρθωσης, απαιτούμενη προσπάθεια,

αριθμός προβλημάτων ή οικονομικό κόστος. Στο επίπεδο του πηγαίου κώδικα, μια παραβίαση κανόνα ποιότητας συνδέεται συνήθως με την εκτιμώμενη προσπάθεια αποκατάστασης (remediation effort), που δίνει μια προσέγγιση του κόστους αποκατάστασης. Επιπλέον, το κεφάλαιο συμπληρώνεται από την έννοια του τόκου (interest). Ο τόκος αντιστοιχεί στην πρόσθετη προσπάθεια που προκαλείται όταν μια τεχνική ανεπάρκεια παραμένει στο σύστημα και επηρεάζει μεταγενέστερες εργασίες. Η διάκριση κεφαλαίου και τόκου είναι σημαντική, επειδή το κόστος διόρθωσης ενός προβλήματος δεν είναι ίδιο με το κόστος που δημιουργείται από τη διατήρησή του. Η παρατήρηση αυτή εισάγει την έννοια της πιθανότητας τόκου (interest probability), δηλαδή της πιθανότητας να προκύψει πράγματι πρόσθετο κόστος που συνδέεται με ένα στοιχείο τεχνικού χρέους. Ακόμη, σχετικό είναι και το επιτόκιο (interest rate), το οποίο εκφράζει τη σχέση του τόκου με το κεφάλαιο σε μια συγκεκριμένη χρονική περίοδο. Με τη διάκριση ανάμεσα στις τρεις αυτές έννοιες, το τεχνικό χρέος αντιμετωπίζεται ως δυναμικό φαινόμενο και όχι απλώς ως ένας συσσωρευμένος αριθμός προβλημάτων. Στη σχετική ορολογία συναντώνται ακόμη οι έννοιες του απλού τόκου (simple interest) και του ανατοκισμού (compound interest). Ο απλός τόκος αντιστοιχεί στο συνεχιζόμενο πρόσθετο κόστος από τη διατήρηση μιας τεχνικής αδυναμίας. Ο ανατοκισμός από την άλλη, εκφράζει την περίπτωση κατά την οποία το ίδιο το χρέος μπορεί να αυξάνεται με την πάροδο του χρόνου.

Για την πρακτική διαχείριση χρησιμοποιείται επίσης η έννοια του στοιχείου τεχνικού χρέους (TD item), το οποίο αποτελεί συγκεκριμένη και διαχειρίσιμη μονάδα τεχνικού χρέους. Μπορεί να αντιστοιχεί σε ένα προβληματικό τμήμα κώδικα, μία κλάση, μία σχεδιαστική ή αρχιτεκτονική απόφαση ή σε μια παραβίαση κανόνα ποιότητας. Η περιγραφή του συνήθως περιλαμβάνει τη θέση του προβλήματος, τον τύπο του, την εκτιμώμενη προσπάθεια αποκατάστασης, τις πιθανές συνέπειες και στοιχεία σχετικά με τη διαχείρισή του. Η αναπαράσταση του χρέους ως σύνολο διακριτών στοιχείων επιτρέπει να περάσει η αξιολόγηση από το γενικό επίπεδο του έργου, στο επίπεδο συγκεκριμένων τεχνικών παρεμβάσεων. Το τεχνικό χρέος συνδέεται έτσι και με την έννοια του κινδύνου. Μια τεχνική αδυναμία μπορεί να μην προκαλεί άμεση λειτουργική αστοχία, αλλά να αυξάνει την πιθανότητα μεγαλύτερης προσπάθειας, δυσκολίας ή ανεπιθύμητων παρενεργειών κατά τις επόμενες αλλαγές. Επομένως, η απόφαση σχετικά με το εάν ένα στοιχείο τεχνικού χρέους θα διατηρηθεί ή θα αντιμετωπιστεί μπορεί να βασιστεί στη σύγκριση του κόστους, του οφέλους και του σχετικού κινδύνου.

Το τεχνικό χρέος δεν είναι μια ενιαία κατηγορία προβλημάτων. Μπορεί να δημιουργηθεί σε διαφορετικά στάδια του κύκλου ζωής του λογισμικού και να αφορά διαφορετικά στοιχεία του συστήματος. Διακρίνονται δέκα κύριες κατηγορίες:

- Το **τεχνικό χρέος απαιτήσεων** (requirements debt) συνδέεται με προβλήματα ή εκκρεμότητες στις απαιτήσεις ενός συστήματος. Επειδή αφορά ένα πιο αφηρημένο επίπεδο από τον πηγαίο κώδικα, η αναγνώρισή του δεν μπορεί συνήθως να βασιστεί αποκλειστικά σε αυτοματοποιημένες μετρικές.
- Το **τεχνικό χρέος αρχιτεκτονικής** (architectural debt) αφορά αποφάσεις που επηρεάζουν τη συνολική οργάνωση του συστήματος. Ανεπιθύμητες εξαρτήσεις μεταξύ υποσυστημάτων, ανεπαρκής διαχωρισμός αρμοδιοτήτων ή αρχιτεκτονικές επιλογές που περιορίζουν την επέκταση μπορούν να αυξήσουν σημαντικά το κόστος μεταβολής.
- Το **τεχνικό χρέος σχεδιασμού** (design debt) αφορά περισσότερο την εσωτερική οργάνωση των στοιχείων που υλοποιούν την αρχιτεκτονική. Αυξημένη σύζευξη, χαμηλή συνοχή, προβληματικές σχέσεις μεταξύ κλάσεων ή πακέτων αποτελούν χαρακτηριστικές ενδείξεις. Σε αντικειμενοστραφή συστήματα μπορεί να προσεγγιστεί μέσω δομικών μετρικών που εξετάζουν πώς αλληλεπιδρούν οι κλάσεις και πώς οργανώνουν τις ευθύνες τους.
- Το **τεχνικό χρέος κώδικα** (code debt) έχει σχέση με αδυναμίες που εντοπίζονται στο επίπεδο της υλοποίησης, όπως σύνθετος ή δυσνόητος κώδικας, διπλοτυπίες, μεγάλες μέθοδοι, οσμές

κώδικα και παραβιάσεις κανόνων. Η κατηγορία αυτή έχει μελετηθεί ιδιαίτερα, καθώς ο πηγαίος κώδικας είναι άμεσα διαθέσιμος και επιτρέπει την εφαρμογή αυτοματοποιημένης στατικής ανάλυσης.

- Το **τεχνικό χρέος δοκιμών** (test debt) εμφανίζεται όταν οι δοκιμές δεν καλύπτουν επαρκώς τις ανάγκες του συστήματος. Περιορισμένη κάλυψη (test coverage), έλλειψη αυτοματοποιημένων δοκιμών ή αναβολή της δημιουργίας αναγκαίων περιπτώσεων δοκιμής αποτελούν ενδεικτικές περιπτώσεις.
- Το **τεχνικό χρέος χτισίματος** (build debt) και **υποδομής** (infrastructure debt) αφορούν αντίστοιχα τις διαδικασίες παραγωγής του λογισμικού και τις τεχνολογίες ή υποδομές που χρησιμοποιούνται για την ανάπτυξή του. Χειροκίνητες διαδικασίες, παρωχημένα εργαλεία και έλλειψη αυτοματοποίησης μπορούν να αυξήσουν την απαιτούμενη προσπάθεια χωρίς να εμφανίζονται ως άμεσο πρόβλημα στον ίδιο τον πηγαίο κώδικα [3].
- Το **τεχνικό χρέος τεκμηρίωσης** (documentation debt) αφορά ανεπαρκή ή παρωχημένη τεκμηρίωση, ενώ το **τεχνικό χρέος διαχείρισης εκδόσεων** (versioning debt) σχετίζεται με προβλήματα στη διαχείριση εκδόσεων και του πηγαίου κώδικα.
- Το **τεχνικό χρέος ελαττωμάτων** (defect debt) αποτελεί ιδιαίτερη περίπτωση, καθώς δεν υπάρχει πλήρης συμφωνία ως προς το αν ένα σφάλμα πρέπει από μόνο του να θεωρείται τεχνικό χρέος. Ορισμένες ταξινομήσεις το εντάσσουν στο τεχνικό χρέος, ενώ άλλες το αντιμετωπίζουν ως ανεξάρτητη διάσταση ποιότητας.

Οι στατικές μετρικές δεν μπορούν να αποτυπώσουν εξίσου καλά όλες τις μορφές τεχνικού χρέους. Μια μεθοδολογία που βασίζεται στον πηγαίο κώδικα, στις δομικές μετρικές και στο ιστορικό ενός αποθετηρίου παρέχει κυρίως ενδείξεις για το τεχνικό χρέος κώδικα και σχεδιασμού [4]. Παράλληλα, μπορεί να αναδείξει πλευρές της συντηρησιμότητας και της εξέλιξης του συστήματος. Τα αποτελέσματα επομένως δεν αποτελούν πλήρη αποτίμηση του συνολικού τεχνικού χρέους.

Η οικονομική ορολογία του τεχνικού χρέους γενικότερα, μπορεί να λειτουργεί ως ένα κοινό πλαίσιο επικοινωνίας ανάμεσα σε τεχνικούς και μη τεχνικούς ρόλους. Έννοιες όπως κεφάλαιο, τόκος, κόστος, όφελος και κίνδυνος επιτρέπουν να συζητούνται οι συνέπειες τεχνικών αποφάσεων με τρόπο που συνδέει την εσωτερική ποιότητα με την απαιτούμενη προσπάθεια [5].

1.3 Αντικείμενο και Στόχος

Αντικείμενο της παρούσας εργασίας είναι ο σχεδιασμός, η ανάπτυξη και η αξιολόγηση ενός ενοποιημένου περιβάλλοντος ανάλυσης έργων λογισμικού. Σκοπός του είναι η εκτίμηση της τεχνικής κατάστασης έργων λογισμικού και η υποστήριξη της λήψης αποφάσεων σχετικών με τη συντήρηση και την προτεραιοποίηση πιθανών ενεργειών αναδόμησης. Για τον σκοπό αυτό, αξιοποιούνται τρεις κύριες πηγές δεδομένων. Το *SonarQube* δίνει πληροφορίες στατικής ανάλυσης που αφορούν μεταξύ άλλων την πολυπλοκότητα, τη συντηρησιμότητα, τις οσμές κώδικα, τα σφάλματα, τον διπλότυπο κώδικα και το εκτιμώμενο τεχνικό χρέος. Το *AllinCode* προσθέτει δομικές μετρικές σε επίπεδο κλάσης και ενδείξεις αυξημένου κινδύνου τεχνικού χρέους. Τέλος, το ιστορικό *Git*, το οποίο παρέχει μετρικές ιστορικής εξέλιξης (process metrics) σχετικά με τη συχνότητα και την ένταση των αλλαγών, όπως το πλήθος των υποβολών αλλαγών, τη μεταβολή κώδικα και τον αριθμό των διαφορετικών προγραμματιστών.

Η εφαρμογή δεν έχει ως στόχο την κατασκευή ενός νέου εργαλείου στατικής ανάλυσης. Το κύριο ενδιαφέρον της βρίσκεται στην ενοποίηση, την επεξεργασία, την ερμηνεία και την οπτικοποίηση των αποτελεσμάτων. Με αυτόν τον τρόπο αποφεύγεται η επανάληψη λειτουργιών που ήδη παρέχονται από τα επιλεγμένα εργαλεία και η ανάπτυξη επικεντρώνεται στο πρόβλημα της πολυδιάστατης αξιολόγησης. Πάνω στα ενοποιημένα δεδομένα εφαρμόζεται η προτεινόμενη μεθοδολογία, η οποία υποστηρίζει τόσο

τη συνοπτική αποτίμηση της κατάστασης του έργου, όσο και την προτεραιοποίηση επιμέρους αρχείων. Ακόμη, αποθηκεύονται αποτελέσματα διαδοχικών αναλύσεων ώστε να είναι δυνατή η παρακολούθηση της τεχνικής εξέλιξης του έργου.

Στο σημείο αυτό είναι χρήσιμο να διευκρινιστεί και η έννοια του “πραγματικού χρόνου” στην παρούσα εργασία. Ο όρος αναφέρεται στην άμεση διαθεσιμότητα ενημερωμένων αποτελεσμάτων στον χρήστη μόλις ολοκληρωθεί μία ανάλυση, σε αντίθεση με προσεγγίσεις όπου η αποτίμηση τεχνικού χρέους γίνεται περιοδικά (π.χ. μηνιαία reports) ή απαιτεί χειροκίνητη σύνθεση αποτελεσμάτων μέσω πολλαπλών εργαλείων. Δεν αναφέρεται σε συνεχή (streaming) παρακολούθηση της κατάστασης του κώδικα κατά τη διάρκεια της ανάπτυξης, κάτι που συζητείται ως κατεύθυνση μελλοντικής επέκτασης στην Ενότητα 6.3.4. Όσον αφορά το πεδίο εφαρμογής, η παρούσα έκδοση λειτουργεί σε τοπικό περιβάλλον και επικεντρώνεται στην ανάλυση δημόσιων αποθετηρίων του GitHub. Η αξιολόγηση αφορά κυρίως τεχνικό χρέος κώδικα και σχεδιασμού, καθώς και πλευρές της συντηρησιμότητας και της ιστορικής εξέλιξης του έργου. Δεν επιχειρεί να καλύψει μορφές τεχνικού χρέους που δεν μπορούν να προκύψουν από τις διαθέσιμες πηγές δεδομένων.

Κατά συνέπεια, ο στόχος της εργασίας είναι διπλός. Σε τεχνικό επίπεδο είναι η δημιουργία μιας ολοκληρωμένης ροής από την εισαγωγή ενός αποθετηρίου έως τη συλλογή, ενοποίηση, αποθήκευση και οπτικοποίηση των αποτελεσμάτων. Σε επίπεδο μεθοδολογίας επιδιώκεται η οργάνωση ετερογενών μετρικών σε μια ερμηνεύσιμη διαδικασία αξιολόγησης, η οποία παρέχει τόσο μια συνοπτική εικόνα του έργου όσο και τη δυνατότητα εντοπισμού συγκεκριμένων περιοχών αυξημένης προτεραιότητας [6].

1.4 Δομή της Εργασίας

Η εργασία οργανώνεται σε έξι κεφάλαια, τα οποία ακολουθούν σταδιακά τη μετάβαση από το θεωρητικό πλαίσιο στη σχεδίαση, την υλοποίηση και την αξιολόγηση του συστήματος. Το πρώτο κεφάλαιο εισάγει το πρόβλημα που εξετάζεται, παρουσιάζει το κίνητρο της εργασίας, το αντικείμενο και τους επιμέρους στόχους της. Ακόμη, παρουσιάζονται βασικές έννοιες και ορολογία του τεχνικού χρέους. Στο δεύτερο κεφάλαιο αναπτύσσεται το θεωρητικό υπόβαθρο σχετικά με τη μέτρηση και τη διαχείριση του τεχνικού χρέους. Εξετάζονται επίσης οι έννοιες της ποιότητας και της συντηρησιμότητας λογισμικού, αλλά και η στατική και συμπεριφορική ανάλυση κώδικα. Επιπλέον, παρουσιάζονται σχετικά εργαλεία και πλατφόρμες. Το τρίτο κεφάλαιο περιγράφει τη μεθοδολογία και τον σχεδιασμό του προτεινόμενου συστήματος. Αναλύονται η συνολική αρχιτεκτονική, η διαδικασία συλλογής και ενοποίησης δεδομένων από SonarQube, AllinCode και Git. Παρουσιάζεται επίσης το μοντέλο αξιολόγησης με τους επιμέρους δείκτες κινδύνου και το Global Health Score. Το τέταρτο κεφάλαιο επικεντρώνεται στην υλοποίηση του dashboard και στις τεχνολογίες που χρησιμοποιήθηκαν. Ακόμη, περιγράφονται η ροή επεξεργασίας, η βάση δεδομένων, η διαχείριση ιστορικότητας και τέλος οι οθόνες της διεπαφής χρήστη. Στο πέμπτο κεφάλαιο παρουσιάζεται η μελέτη περίπτωσης και καταγράφονται τα αποτελέσματα εφαρμογής του συστήματος. Στη συνέχεια, ακολουθεί η διαδικασία βαθμονόμησης του μοντέλου, μαζί με την ανάλυση ευαισθησίας των βασικών παραμέτρων και βαρών. Τέλος, το έκτο κεφάλαιο συνοψίζει τα κύρια ευρήματα και συζητά τους περιορισμούς και τις απειλές εγκυρότητας.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Σχετικές Εργασίες

2.1 Μέτρηση και διαχείριση Τεχνικού Χρέους

Η μεταφορά του Τεχνικού Χρέους εμφανίστηκε για πρώτη φορά το 1992 στην εργασία του Ward Cunningham *The WyCash Portfolio Management System*. Ο Cunningham χρησιμοποίησε την οικονομική έννοια του χρέους για να περιγράψει τη σχέση μεταξύ της ταχύτερης ανάπτυξης στο παρόν και της πρόσθετης προσπάθειας που μπορεί να απαιτηθεί αργότερα εξαιτίας τεχνικών επιλογών που δεν αποτελούν την καταλληλότερη μακροπρόθεσμη λύση [10]. Η αξία της μεταφοράς βρίσκεται κυρίως στο ότι συνδέει τις τεχνικές επιλογές με τις συνέπειές τους στη μακροχρόνια ανάπτυξη και συντήρηση του συστήματος. Η έννοια διευρύνθηκε σημαντικά και πλέον το τεχνικό χρέος αντιμετωπίζεται ως ζήτημα που αφορά τόσο τεχνικές όσο και διαχειριστικές αποφάσεις. Η διαχείρισή του δεν περιορίζεται στον εντοπισμό τεχνικών αδυναμιών, αλλά οργανώνεται σε οκτώ βασικές δραστηριότητες [11]:

Αναγνώριση (identification). Η αναγνώριση προσπαθεί να προσδιορίσει την ύπαρξη και τη θέση του τεχνικού χρέους. Ανάλογα με τον τύπο του χρέους μπορούν να χρησιμοποιούνται διαφορετικές πηγές πληροφορίας. Για το τεχνικό χρέος κώδικα, η αυτοματοποιημένη στατική ανάλυση είναι μία από τις συχνότερες τεχνικές, καθώς μπορεί να εντοπίζει παραβιάσεις κανόνων, διπλοτυπίες (duplication), οσμές κώδικα (code smells) και ακραίες τιμές μετρικών. Για πιο αφηρημένες κατηγορίες, όπως το αρχιτεκτονικό χρέος ή το χρέος απαιτήσεων, χρειάζεται συνήθως επιπλέον πληροφορία που δεν είναι άμεσα διαθέσιμη στον πηγαίο κώδικα [4].

Μέτρηση (measurement). Η αναγνώριση από μόνη της δεν δείχνει ποια ευρήματα έχουν τη μεγαλύτερη πρακτική σημασία. Η μέτρηση προσπαθεί να ποσοτικοποιήσει την τεχνική επιβάρυνση και να επιτρέψει τη σύγκριση ανάμεσα σε διαφορετικές περιπτώσεις. Η μέτρηση του κεφαλαίου (principal) είναι σχετικά πιο άμεση, καθώς βασίζεται στην προσπάθεια που απαιτείται για την αποκατάσταση ενός ήδη γνωστού προβλήματος. Στα εργαλεία στατικής ανάλυσης αυτό υλοποιείται μέσω της προσπάθειας αποκατάστασης (remediation effort), όπου κάθε παραβίαση αντιστοιχίζεται σε εκτιμώμενο χρόνο επίλυσης. Έτσι, δύο αρχεία με το ίδιο πλήθος παραβιάσεων μπορεί να έχουν διαφορετική εκτιμώμενη προσπάθεια αποκατάστασης [1]. Η μέτρηση του τόκου (interest) είναι δυσκολότερη, καθώς αφορά προσπάθεια που δεν έχει ακόμη πραγματοποιηθεί. Στην πράξη, η πληροφορία που θα απαιτούνταν για ακριβή αποτίμηση δεν είναι διαθέσιμη, οπότε χρησιμοποιούνται προσεγγίσεις εκτίμησης [5]. Ο τόκος επίσης συνδέεται στενά με τη συντηρησιμότητα, αφού μια τεχνική αδυναμία αποκτά μεγαλύτερη σημασία όταν δυσκολεύει τις μελλοντικές αλλαγές. Τέλος, εδώ αποκτά σημασία και η πιθανότητα τόκου (interest probability). Αν ένα στοιχείο χαμηλής ποιότητας δεν χρειαστεί να αλλάξει, η επιβάρυνσή του παραμένει σε μεγάλο βαθμό ανενεργή, ενώ αν αλλάζει επανειλημμένα, αυξάνεται η πιθανότητα να μετατραπεί σε πραγματικό κόστος [2]. Το ιστορικό αλλαγών λειτουργεί έτσι ως σημαντικό συμπλήρωμα των στατικών μετρικών.

Προτεραιοποίηση (prioritization). Η προτεραιοποίηση αποτελεί κεντρική δραστηριότητα της διαχείρισης τεχνικού χρέους, καθώς σε πραγματικά έργα δεν είναι εφικτό να αποπληρωθούν όλα τα στοιχεία χρέους. Η λογική της διαλογής τεχνικού χρέους (Technical Debt triage) αφορά την αξιολόγηση και κατάταξη των στοιχείων, ώστε η διαθέσιμη προσπάθεια να κατευθύνεται σε εκείνα που θεωρούνται σημαντικότερα [8]. Το κεφάλαιο, ο πιθανός τόκος, η πιθανότητα αλλαγής και το επιχειρηματικό πλαίσιο μπορούν να επηρεάσουν την τελική απόφαση. Επομένως, ένα στοιχείο με υψηλό κόστος αποκατάστασης δεν πρέπει να αποκτά αυτόματα την υψηλότερη προτεραιότητα.

Πρόληψη (prevention). Η πρόληψη αφορά τη μείωση της δημιουργίας νέου τεχνικού χρέους. Σε αντίθεση με την αποπληρωμή, η οποία αντιμετωπίζει υπάρχοντα προβλήματα, η πρόληψη επιδιώκει να εμποδίσει τη συσσώρευση νέων αδυναμιών. Στις σύγχρονες ροές εργασίας αυτό μπορεί να γίνει μέσω επιθεώρησης του κώδικα, αυτοματοποιημένων ελέγχων, κριτηρίων ποιότητας (quality gates) και αξιολόγησης του νέου κώδικα πριν ενσωματωθεί στην κύρια έκδοση του συστήματος [2].

Παρακολούθηση (monitoring). Η παρακολούθηση προσθέτει χρονική διάσταση στη διαχείριση. Μια μεμονωμένη ανάλυση αποτελεί στιγμιότυπο της τρέχουσας κατάστασης και δεν δείχνει αν η ποιότητα βελτιώνεται, χειροτερεύει ή παραμένει σταθερή. Η αποθήκευση διαδοχικών μετρήσεων επιτρέπει να αναγνωριστούν τάσεις και να συσχετιστούν οι τεχνικές μεταβολές με σημαντικές αλλαγές του έργου.

Αποπληρωμή (repayment). Η αποπληρωμή περιλαμβάνει τις τεχνικές ενέργειες που μειώνουν το υπάρχον χρέος. Η παρέμβαση όμως δεν πρέπει να γίνεται μόνο επειδή ένα εργαλείο εντόπισε ένα πρόβλημα. Κάθε αναδόμηση κώδικα απαιτεί προσπάθεια και ενδέχεται να δημιουργήσει κίνδυνο παρενεργειών, ιδιαίτερα σε κεντρικά στοιχεία του συστήματος.

Εκπροσώπηση/τεκμηρίωση (representation/documentation). Αφορά τη συστηματική καταγραφή των στοιχείων του τεχνικού χρέους και των σχετικών αποφάσεων. Η δραστηριότητα είναι ιδιαίτερα σημαντική για μορφές χρέους που δεν ανακτώνται εύκολα από τον κώδικα. Μια αρχιτεκτονική παραχώρηση ή ένας γνωστός περιορισμός μιας υποδομής μπορεί να χαθεί όταν η πληροφορία παραμένει αποκλειστικά στη γνώση συγκεκριμένων μελών της ομάδας [4].

Επικοινωνία (communication). Η επικοινωνία αφορά τον τρόπο με τον οποίο οι πληροφορίες παρουσιάζονται στους διαφορετικούς ενδιαφερόμενους. Οι προγραμματιστές χρειάζονται λεπτομερή στοιχεία για συγκεκριμένα αρχεία, κλάσεις, προβλήματα και πιθανές παρεμβάσεις. Σε πιο διοικητικό επίπεδο, αντίθετα, χρειάζονται περισσότερο συνοπτικές πληροφορίες σχετικά με την κατάσταση, την τάση, το κόστος και τον κίνδυνο.

Συνολικά, φαίνεται ότι η μέτρηση και η διαχείριση αποτελούν διαφορετικά αλλά αλληλένδετα προβλήματα. Η στατική ανάλυση και οι μετρικές μπορούν να αναγνωρίσουν και να ποσοτικοποιήσουν τεχνικές αδυναμίες. Η ολοκληρωμένη διαχείριση όμως απαιτεί επιπλέον γνώση της εξέλιξης, της πιθανότητας μελλοντικής αλλαγής, του κόστους παρέμβασης και του περιβάλλοντος του έργου.

2.2 Ποιότητα και Συντηρησιμότητα Λογισμικού

Η σωστή εκτέλεση των λειτουργικών απαιτήσεων είναι απαραίτητη, αλλά δεν είναι η μοναδική προϋπόθεση ενός ποιοτικού συστήματος. Παράγοντες όπως η αξιοπιστία, η απόδοση, η ασφάλεια, η δυνατότητα αλληλεπίδρασης και η συντηρησιμότητα μπορούν να επηρεάσουν σημαντικά την αποτελεσματικότητα και τη μακροχρόνια βιωσιμότητα του προϊόντος. Ιδιαίτερο ενδιαφέρον για τη μελέτη του τεχνικού χρέους παρουσιάζει η διάκριση ανάμεσα σε εξωτερικές ιδιότητες, που γίνονται αντιληπτές κατά τη χρήση του συστήματος, και εσωτερικές ιδιότητες της δομής και της υλοποίησης. Ένας χρήστης μπορεί να μην αντιλαμβάνεται άμεσα ότι μια κλάση παρουσιάζει υψηλή σύζευξη ή ότι μια μέθοδος είναι ιδιαίτερα πολύπλοκη. Αυτές οι ιδιότητες όμως, μπορούν να επηρεάσουν σημαντικά την προσπάθεια που θα χρειαστεί όταν η ομάδα τροποποιήσει το αντίστοιχο τμήμα. Επομένως, η εσωτερική ποιότητα συνδέεται με τη συντηρησιμότητα. Ένα σύστημα που λειτουργεί σωστά σήμερα μπορεί να δημιουργήσει σημαντικές δυσκολίες στο μέλλον όταν η δομή του δεν επιτρέπει την εύκολη κατανόηση, απομόνωση και τροποποίηση των επιμέρους λειτουργιών.

2.2.1 Το Πρότυπο ISO/IEC 25010

Η συστηματική περιγραφή της ποιότητας υποστηρίζεται από διεθνή μοντέλα. Το παλαιότερο ISO/IEC 9126 αποτέλεσε για μεγάλο χρονικό διάστημα σημαντικό σημείο αναφοράς για την κατηγοριοποίηση

των χαρακτηριστικών ποιότητας. Σήμερα, έχει αντικατασταθεί από τα πρότυπα της οικογένειας SQuaRE (Systems and software Quality Requirements and Evaluation). Για την παρούσα εργασία χρησιμοποιείται το ISO/IEC 25010:2023, η νεότερη έκδοση του μοντέλου ποιότητας προϊόντος [12].



Εικόνα 2.1 - Διαγραμματική απεικόνιση ISO/IEC 25010

Το ISO/IEC 25010 παρέχει ένα κοινό εννοιολογικό πλαίσιο μέσω του οποίου οι απαιτήσεις ποιότητας μπορούν να οργανωθούν και να συνδεθούν με τις κατάλληλες μετρήσεις. Στην έκδοση του 2023, το μοντέλο ποιότητας προϊόντος οργανώνεται σε εννέα κύρια χαρακτηριστικά. Αρχικά, αναφέρεται η λειτουργική καταλληλότητα (Functional Suitability), η οποία αφορά την ορθότητα και την πληρότητα των λειτουργιών. Έπειτα, η αποδοτικότητα επιδόσεων (Performance Efficiency), που σχετίζεται με τη συμπεριφορά του συστήματος και τους πόρους που χρησιμοποιεί. Ακολουθεί η συμβατότητα (Compatibility), που σχετίζεται με τη συνεργασία με άλλα συστήματα. Κοντά σε αυτήν βρίσκεται και η ικανότητα αλληλεπίδρασης (Interaction Capability), δηλαδή η αποτελεσματική επικοινωνία με τους χρήστες. Η αξιοπιστία (Reliability) αφορά τη συνεπή λειτουργία του συστήματος. Η ασφάλεια (Security) έχει σχέση με την προστασία πληροφοριών και λειτουργιών, ενώ η ευελιξία (Flexibility) με την προσαρμογή σε μεταβαλλόμενες συνθήκες. Επίσης, η ασφάλεια έναντι κινδύνων (Safety), η οποία αφορά τον περιορισμό κινδύνων που οδηγούν σε μη αποδεκτές συνέπειες [12]. Μεγαλύτερη σημασία για την παρούσα εργασία έχει η συντηρησιμότητα (Maintainability), καθώς συνδέεται άμεσα με την προσπάθεια κατανόησης και μεταβολής του λογισμικού. Το ISO/IEC 25010 τη διακρίνει σε πέντε βασικά υποχαρακτηριστικά:

- Η **αρθρωτότητα (modularity)** αφορά τον βαθμό στον οποίο ένα σύστημα οργανώνεται σε διακριτά στοιχεία, έτσι ώστε μια αλλαγή σε ένα από αυτά να έχει περιορισμένες επιπτώσεις στα υπόλοιπα. Η έντονη σύζευξη μεταξύ διαφορετικών στοιχείων λειτουργεί αντίθετα σε αυτή την ιδιότητα. Όταν μία αλλαγή απαιτεί διαδοχικές τροποποιήσεις σε πολλά εξαρτώμενα στοιχεία, το κόστος συντήρησης αυξάνεται.
- Η **επαναχρησιμοποιησιμότητα (reusability)** έχει σχέση με το πόσο ένα στοιχείο μπορεί να χρησιμοποιηθεί σε περισσότερα από ένα συστήματα ή πλαίσια. Σαφείς αρμοδιότητες, περιορισμένες εξαρτήσεις και κατάλληλη ενθυλάκωση μπορούν να διευκολύνουν την

επαναχρησιμοποίηση, χωρίς όμως να σημαίνει ότι κάθε τμήμα κώδικα πρέπει να σχεδιάζεται ως γενικευμένο στοιχείο λογισμικού.

- Η **αναλυσιμότητα (analysability)** σχετίζεται με την αποτελεσματικότητα με την οποία ένας προγραμματιστής μπορεί να κατανοήσει τον κώδικα, να εντοπίσει την αιτία ενός προβλήματος και να εκτιμήσει τις πιθανές συνέπειες μιας αλλαγής. Αυξημένη πολυπλοκότητα, ισχυρή σύζευξη και ασαφής κατανομή ευθυνών μπορεί να αυξήσουν την προσπάθεια ανάλυσης.
- Η **τροποποιησιμότητα (modifiability)** αφορά τη δυνατότητα να γίνονται αλλαγές χωρίς υπερβολική προσπάθεια και χωρίς ανεπιθύμητες παρενέργειες. Το χαρακτηριστικό αυτό παρουσιάζει άμεση σύνδεση με τον τόκο: όταν η υπάρχουσα δομή κάνει κάθε νέα αλλαγή ακριβότερη, η τεχνική επιβάρυνση αρχίζει να δημιουργεί πραγματικό πρόσθετο κόστος.
- Η **ελεγχιμότητα (testability)** αφορά τη δυνατότητα καθορισμού και εκτέλεσης αποτελεσματικών δοκιμών. Ο κακός διαχωρισμός αρμοδιοτήτων, οι ισχυρές εξαρτήσεις και τα μεγάλα στοιχεία μπορεί να δυσκολεύουν την απομόνωση μιας λειτουργίας κατά τον έλεγχο. Κατά συνέπεια, η ελεγχιμότητα δεν εξαρτάται μόνο από τον αριθμό των διαθέσιμων δοκιμών, αλλά και από τη σχεδίαση του ίδιου του συστήματος.

Όλα τα παραπάνω δείχνουν πως η συντηρησιμότητα αποτελεί σύνθετο χαρακτηριστικό που επηρεάζεται από διαφορετικές ιδιότητες του κώδικα και της σχεδίασης. Επομένως, μετρικές όπως η πολυπλοκότητα, η σύζευξη, η συνοχή και το μέγεθος μπορούν να χρησιμοποιηθούν κατά βάση ως επιμέρους ενδείξεις [22]. Αντίστοιχα, οι δείκτες που παράγει το dashboard βασίζονται κυρίως σε παρατηρήσιμες διαστάσεις που σχετίζονται με τη συντηρησιμότητα, την αξιοπιστία, την πολυπλοκότητα, τη δομική ποιότητα και την ιστορική δραστηριότητα. Δεν επιχειρούν να καλύψουν το σύνολο των εννέα χαρακτηριστικών του ISO/IEC 25010, άρα δεν πρέπει και να ερμηνεύονται ως απόλυτη μέτρηση της συνολικής ποιότητας ενός προϊόντος.

2.2.2 Η Σημασία της Συντηρησιμότητας στον Κύκλο Ζωής του Λογισμικού

Μετά την πρώτη λειτουργική έκδοση, ένα σύστημα συνεχίζει να μεταβάλλεται ώστε να διορθώνονται προβλήματα, να προστίθενται νέες λειτουργίες και να προσαρμόζεται σε νέες απαιτήσεις και τεχνολογικές εξελίξεις. Αυτό σημαίνει πως η ποιότητα του κώδικα επηρεάζει άμεσα την προσπάθεια που απαιτούν αυτές οι μεταβολές, αφού τεχνικές αδυναμίες μπορούν να δυσκολέψουν τις επόμενες τροποποιήσεις. Για τον λόγο αυτό, η συντηρησιμότητα αποτελεί σημαντικό παράγοντα στη μακροχρόνια εξέλιξη του λογισμικού.

Ένα από τα κλασικά θεωρητικά πλαίσια που εξηγούν τη συγκεκριμένη δυναμική είναι το *Laws of Software Evolution* του Lehman. Η διατύπωση των νόμων εξελίχθηκε μέσα από πολυετή μελέτη πραγματικών συστημάτων και στην ώριμη μορφή της περιλαμβάνει οκτώ βασικές αρχές για την εξέλιξη των λεγόμενων E-type systems [13]. Ο νόμος της Συνεχούς Αλλαγής (Continuing Change) αναφέρει ότι ένα σύστημα που λειτουργεί μέσα σε πραγματικό και μεταβαλλόμενο περιβάλλον πρέπει να προσαρμόζεται συνεχώς, διαφορετικά γίνεται σταδιακά λιγότερο ικανοποιητικό. Ακόμη, ο νόμος της Αυξανόμενης Πολυπλοκότητας (Increasing Complexity) έχει ιδιαίτερη σημασία για τη μελέτη της τεχνικής κατάστασης. Καθώς ένα σύστημα εξελίσσεται, η πολυπλοκότητά του τείνει να αυξάνεται, εκτός αν πραγματοποιούνται ενέργειες για τον περιορισμό της [13]. Η αύξηση της πολυπλοκότητας δεν σημαίνει απαραίτητα ότι η ομάδα εργάζεται λανθασμένα, αλλά αποτελεί σε σημαντικό βαθμό φυσική συνέπεια ενός συστήματος που προσαρμόζεται συνεχώς. Οι υπόλοιπες αρχές του Lehman εξετάζουν πλευρές όπως η σταθερότητα του ρυθμού εξέλιξης, η διατήρηση της εξοικείωσης της ομάδας με το σύστημα, η ανάγκη συνεχούς ανάπτυξης και η αλληλεπίδραση μεταξύ του λογισμικού και του περιβάλλοντός του. Πιο συγκεκριμένα, η αρχή της Φθίνουσας Ποιότητας (Declining Quality) επισημαίνει ότι η ποιότητα ενός εξελισσόμενου συστήματος μπορεί να υποβαθμιστεί όταν δεν

πραγματοποιούνται επαρκείς ενέργειες συντήρησης και προσαρμογής [13]. Η ανάγκη περιορισμού αυτής της υποβάθμισης συνδέεται και με την αναδόμηση κώδικα, την οποία ο Fowler ορίζει ως μεταβολή της εσωτερικής δομής ενός συστήματος χωρίς αλλαγή της παρατηρήσιμης εξωτερικής συμπεριφοράς [14].

Οι αρχές αυτές παρέχουν θεωρητική αιτιολόγηση για την παρακολούθηση της κατάστασης του λογισμικού σε βάθος χρόνου. Εφόσον η αλλαγή είναι αναπόφευκτη και η πολυπλοκότητα μπορεί να αυξάνεται κατά την εξέλιξη, μια μεμονωμένη στατική ανάλυση δεν αρκεί για την κατανόηση της συνολικής πορείας του συστήματος. Για τον λόγο αυτό, οι επαναλαμβανόμενες αναλύσεις και τα ιστορικά δεδομένα μπορούν να προσφέρουν πληρέστερη εικόνα της εξέλιξής του.

2.3 Στατική Ανάλυση Κώδικα

2.3.1 Αρχές Στατικής Ανάλυσης

Η στατική ανάλυση κώδικα (Static Code Analysis) είναι μία από τις βασικές τεχνικές που χρησιμοποιούνται για την αυτοματοποιημένη αξιολόγηση της εσωτερικής ποιότητας. Κύριο χαρακτηριστικό της είναι ότι εξετάζει τον πηγαίο κώδικα χωρίς να χρειάζεται η εκτέλεση του προγράμματος. Η στατική ανάλυση διαφέρει από τον δυναμικό έλεγχο λογισμικού. Στις δυναμικές τεχνικές, το πρόγραμμα εκτελείται με συγκεκριμένες εισόδους και εξετάζεται η πραγματική του συμπεριφορά. Στη στατική ανάλυση, αντίθετα, αντικείμενο εξέτασης είναι η μορφή και η δομή του κώδικα. Τα εργαλεία στατικής ανάλυσης μπορούν να λειτουργούν σε διαφορετικά επίπεδα πολυπλοκότητας. Απλοί έλεγχοι βασίζονται σε συντακτικούς κανόνες, ενώ πιο προηγμένες τεχνικές μπορεί να αξιοποιούν το Δέντρο Αφηρημένης Σύνταξης, τη δομή ροής ελέγχου, τη ροή δεδομένων ή τις εξαρτήσεις μεταξύ διαφορετικών στοιχείων. Παράλληλα, μπορεί να υπολογίζονται αριθμητικές μετρικές που χαρακτηρίζουν διαφορετικές πτυχές της υλοποίησης.

Η δυνατότητα αυτοματοποίησης αποτελεί σημαντικό πλεονέκτημα. Σε ένα μεγάλο αποθετήριο, ο χειροκίνητος έλεγχος κάθε αρχείου δεν είναι πρακτικά εφικτός. Ένα εργαλείο στατικής ανάλυσης μπορεί να εφαρμόζει σταθερά το ίδιο σύνολο κανόνων, να παράγει συγκρίσιμα αποτελέσματα και να επαναλαμβάνει την ανάλυση μετά από κάθε σημαντική αλλαγή. Με αυτόν τον τρόπο, η στατική ανάλυση μπορεί να εφαρμοστεί επαναλαμβανόμενα σε μεγάλο αριθμό αρχείων και να ενσωματωθεί στην καθημερινή διαδικασία ανάπτυξης [4]. Τα αποτελέσματα μπορούν επίσης να παρέχονται σε διαφορετικά επίπεδα ανάλυσης. Ορισμένες τιμές αφορούν το σύνολο του έργου, ενώ άλλες μπορούν να αντιστοιχούν σε αρχείο, κλάση ή μέθοδο. Η ύπαρξη πληροφορίας σε επίπεδο αρχείου είναι ιδιαίτερα σημαντική για την υποστήριξη αποφάσεων συντήρησης, επειδή επιτρέπει τη μετατροπή μιας γενικής ένδειξης χαμηλής ποιότητας σε συγκεκριμένο τεχνικό σημείο, το οποίο μπορεί να διερευνήσει ο προγραμματιστής [1].

Τα εργαλεία στατικής ανάλυσης παρέχουν τόσο αριθμητικές μετρικές όσο και ενδείξεις για πιθανά προβλήματα του κώδικα. Στις βασικές μετρικές περιλαμβάνονται η Κυκλωματική και η Γνωστική Πολυπλοκότητα. Η Κυκλωματική Πολυπλοκότητα αποτυπώνει την πολυπλοκότητα που δημιουργείται από τη ροή ελέγχου. Όσο αυξάνεται ο αριθμός των ανεξάρτητων διαδρομών μέσα στον κώδικα, αυξάνεται συνήθως και η προσπάθεια που χρειάζεται για την πλήρη κατανόηση και τον έλεγχό του. Η Γνωστική Πολυπλοκότητα (Cognitive Complexity) προσπαθεί να προσεγγίσει πιο άμεσα τη γνωστική προσπάθεια που απαιτείται για την κατανόηση της λογικής. Διαδοχικές συνθήκες και πολύπλοκες δομές ελέγχου μπορεί να κάνουν τον κώδικα δύσκολο να ακολουθηθεί, ακόμη και όταν η λειτουργικότητά του είναι σωστή. Το μέγεθος αποτελεί επίσης βασικό χαρακτηριστικό. Ο αριθμός των γραμμών κώδικα, των δηλώσεων, των συναρτήσεων ή των κλάσεων μπορεί να χρησιμοποιηθεί ως περιγραφική μετρική.

Οι απόλυτες τιμές, όμως, πρέπει να ερμηνεύονται προσεκτικά. Ένα μεγάλο αρχείο είναι φυσιολογικό να περιέχει περισσότερα πιθανά προβλήματα από ένα μικρό, οπότε οι συγκρίσεις χρειάζονται συχνά πυκνότητες ή σχετικές τιμές. Τέλος, ο διπλότυπος κώδικας αποτελεί ακόμη μία σημαντική, διαφορετική διάσταση. Η ύπαρξη της ίδιας ή πολύ παρόμοιας λογικής σε πολλά σημεία αυξάνει τον αριθμό των θέσεων που ενδέχεται να χρειαστεί να τροποποιηθούν όταν αλλάξει η συγκεκριμένη συμπεριφορά. Το πρόβλημα δεν βρίσκεται στην ύπαρξη επιπλέον γραμμών κώδικα, αλλά στον κίνδυνο ασυνέπειας ανάμεσα στα διαφορετικά αντίγραφα.

Η στατική ανάλυση μπορεί επίσης να συνδεθεί με το τεχνικό χρέος μέσω της προσπάθειας αποκατάστασης. Οι διαφορετικές παραβιάσεις μπορούν να αντιστοιχίζονται σε εκτιμώμενη προσπάθεια αποκατάστασης, επιτρέποντας τη μετάβαση από το "πόσα προβλήματα υπάρχουν;" στο "πόση εκτιμώμενη προσπάθεια απαιτείται για την αντιμετώπισή τους;". Η προσέγγιση αυτή δεν πρέπει να συγχέεται με μια απόλυτη οικονομική μέτρηση. Η εκτίμηση της προσπάθειας που παρέχει ένα εργαλείο βασίζεται σε κανόνες και παραδοχές και δεν λαμβάνει υπόψη όλες τις ιδιαιτερότητες του έργου. Για τον λόγο αυτό, οι τιμές είναι καταλληλότερες για προσανατολισμό και σχετική σύγκριση [16]. Παράλληλα, η στατική ανάλυση μπορεί να λειτουργήσει ως μηχανισμός πρόληψης. Αντί να εξετάζεται η ποιότητα μόνο μετά από μεγάλη συσσώρευση αλλαγών, οι αναλύσεις μπορούν να ενσωματωθούν στη διαδικασία ανάπτυξης και συνεχούς ενσωμάτωσης, ώστε τα νέα προβλήματα να εντοπίζονται πριν γίνουν μόνιμο μέρος του codebase.

Παρά τα πλεονεκτήματά της, η στατική ανάλυση έχει σαφή όρια: δεν γνωρίζει πόσο συχνά χρησιμοποιείται ή τροποποιείται ένα αρχείο, ποια είναι η επιχειρησιακή του σημασία ή ποιο είναι το πραγματικό κόστος αλλαγής του για τη συγκεκριμένη ομάδα. Επομένως, η στατική ανάλυση αναδεικνύει τεχνικές αδυναμίες, χωρίς όμως να μπορεί να προβλέψει ποιες από αυτές θα απασχολήσουν πραγματικά την ομάδα στο μέλλον. Το κενό αυτό μπορεί να καλυφθεί με τον συνδυασμό της στατικής ανάλυσης με δομικές και ιστορικές πληροφορίες.

2.3.2 Εντοπισμός Οσμών Κώδικα (Code Smells) και Παραβιάσεων Κανόνων

Οι οσμές κώδικα (code smells) είναι ενδείξεις ότι η εσωτερική δομή ενός τμήματος κώδικα μπορεί να δυσκολεύει την κατανόηση, την τροποποίηση ή την επέκτασή του. Μια οσμή δεν συνιστά αυτόματα σφάλμα λειτουργίας. Ο κώδικας μπορεί να παράγει ακριβώς το σωστό αποτέλεσμα και ταυτόχρονα να έχει δομή που κάνει τη μελλοντική εξέλιξη πιο δύσκολη. Οι έννοιες σφάλμα (bug), οσμή κώδικα (code smell) και παράβαση κανόνα (rule violation) μπορεί να συνδέονται, αλλά δεν είναι ταυτόσημες. Ένα σφάλμα αφορά λανθασμένη συμπεριφορά ή αυξημένη πιθανότητα λειτουργικού προβλήματος. Μια οσμή κώδικα από την άλλη έχει σχέση πιο πολύ με την εσωτερική σχεδίαση και συντηρησιμότητα. Αντίστοιχα, μια παραβίαση κανόνα δηλώνει ότι ο κώδικας δεν ικανοποιεί συγκεκριμένο κανόνα του εργαλείου ανάλυσης.

Μια από τις πιο γνωστές οσμές είναι η μακριά μέθοδος (Long Method). Μια μακριά μέθοδος περιλαμβάνει υπερβολικά μεγάλο αριθμό ενεργειών και μπορεί να είναι δύσκολη στην κατανόηση, στον έλεγχο και στην επαναχρησιμοποίηση. Η διάσπασή της σε μικρότερες, σαφώς ονομασμένες λειτουργίες μπορεί σε κατάλληλες περιπτώσεις να περιορίσει το γνωστικό φορτίο [14]. Μια εξίσου γνωστή οσμή είναι η μεγάλη κλάση (Large Class), που εμφανίζεται όταν μία κλάση συγκεντρώνει υπερβολικά πολλές αρμοδιότητες. Η μεγάλη έκταση δεν είναι απόδειξη κακού σχεδιασμού, αλλά όταν συνοδεύεται από μεγάλο αριθμό διαφορετικών ευθυνών μπορεί να υποδεικνύει ότι η κλάση θα ήταν καλύτερο να χωριστεί σε μικρότερα και πιο συνεκτικά στοιχεία.

Η ύπαρξη μιας οσμής κώδικα δεν σημαίνει πως σίγουρα υπάρχει ανάγκη αναδόμησης. Ο ρόλος του στοιχείου, οι απαιτήσεις απόδοσης, το κόστος της παρέμβασης, οι εξαρτήσεις και η συχνότητα αλλαγής μπορούν να επηρεάσουν σημαντικά την τελική απόφαση. Η ίδια αρχή ισχύει και για τις παραβιάσεις κανόνων. Ένα εργαλείο στατικής ανάλυσης εφαρμόζει ένα συγκεκριμένο ποιοτικό προφίλ (quality profile), δηλαδή ένα σύνολο ενεργοποιημένων κανόνων. Η αλλαγή του προφίλ μπορεί να οδηγήσει σε διαφορετικό αριθμό προβλημάτων στον ίδιο ακριβώς πηγαίο κώδικα. Η κάθε τιμή επομένως πρέπει να ερμηνεύεται μαζί με το πλαίσιο της ανάλυσης.

Συνολικά, η στατική ανάλυση προσφέρει έναν ισχυρό μηχανισμό αναγνώρισης και ποσοτικοποίησης, αλλά αφήνει ανοιχτό το ζήτημα της πρακτικής σημασίας. Για να απαντηθεί το ερώτημα “ποια από τα προβληματικά αρχεία αξίζουν μεγαλύτερη προσοχή;”, είναι χρήσιμο να εξετάζεται παράλληλα ο τρόπος με τον οποίο τα αρχεία αυτά συμμετέχουν στην πραγματική εξέλιξη του έργου.

2.4 Συμπεριφορική Ανάλυση Κώδικα (Behavioral Code Analysis)

Η στατική ανάλυση περιγράφει σε μεγάλο βαθμό την κατάσταση του κώδικα σε μια συγκεκριμένη χρονική στιγμή. Η συμπεριφορική ανάλυση προσθέτει μια διαφορετική οπτική, αξιοποιώντας την ιστορία ανάπτυξης ώστε να εξετάσει πώς μεταβάλλεται το σύστημα στην πράξη. Ο όρος συμπεριφορική ανάλυση κώδικα δεν αναφέρεται στη συμπεριφορά της εφαρμογής κατά την εκτέλεσή της, αλλά στον τρόπο με τον οποίο ο κώδικας αλλάζει και εξελίσσεται κατά την ανάπτυξη. Η ανάλυση βασίζεται στο ιστορικό του ελέγχου εκδόσεων (version control) και εξετάζει στοιχεία όπως τις υποβολές αλλαγών, τη συχνότητα αλλαγών, τη μεταβολή κώδικα και τη συμμετοχή των προγραμματιστών. Έτσι, μπορεί να εντοπιστεί σε ποια σημεία του κώδικα συγκεντρώνεται η μεγαλύτερη δραστηριότητα με την πάροδο του χρόνου.

2.4.1 Η Έννοια των Hotspots

Τα hotspots είναι περιοχές του συστήματος όπου συνδυάζεται σημαντική δραστηριότητα αλλαγών με αυξημένη τεχνική δυσκολία ή χαμηλή εσωτερική ποιότητα. Η έννοια βασίζεται στην παρατήρηση ότι ένα προβληματικό τμήμα κώδικα αποκτά μεγαλύτερη πρακτική σημασία όταν είναι ταυτόχρονα συχνός στόχος ανάπτυξης. Ένα αρχείο που παρουσιάζει μεγάλη πολυπλοκότητα αλλά παραμένει αμετάβλητο επί χρόνια μπορεί να έχει περιορισμένη άμεση προτεραιότητα. Αντίθετα, ένα εξίσου ή ακόμη και λιγότερο σύνθετο αρχείο που τροποποιείται συνεχώς εκθέτει την ομάδα επανειλημμένα στην τεχνική του δυσκολία.

Η έννοια των hotspots συνδέεται άμεσα με την πιθανότητα τόκου (interest probability) του τεχνικού χρέους. Μία τεχνική αδυναμία δημιουργεί πραγματικό τόκο όταν το αντίστοιχο στοιχείο πρέπει να κατανοηθεί ή να τροποποιηθεί. Το ιστορικό δεν μπορεί να προβλέψει με βεβαιότητα τις μελλοντικές αλλαγές, αλλά μπορεί να παρέχει πληροφορία σχετικά με το ποια στοιχεία έχουν παρουσιάσει επαναλαμβανόμενη δραστηριότητα [2]. Η λογική αυτή αποτελεί κεντρικό χαρακτηριστικό του CodeScene, όπου η δραστηριότητα του κώδικα χρησιμοποιείται για την αναγνώριση hotspots και συνδυάζεται με την τεχνική κατάσταση του αντίστοιχου στοιχείου[19].

Η συχνή αλλαγή δεν αποδεικνύει από μόνη της πρόβλημα, καθώς μπορεί απλώς να αντιστοιχεί σε μια βασική λειτουργία που βρίσκεται ακόμη σε ενεργή ανάπτυξη. Ομοίως, η υψηλή πολυπλοκότητα μπορεί να είναι αποτέλεσμα εγγενώς σύνθετης επιχειρησιακής λογικής και όχι κακού σχεδιασμού. Ωστόσο, η ανάλυση hotspots προσφέρει έναν πρακτικό μηχανισμό περιορισμού του χώρου αναζήτησης. Σε μεγάλα αποθετήρια δεν είναι εφικτό να εξετάζονται με την ίδια ένταση όλα τα αρχεία, οπότε η ανάδειξη ενός μικρότερου συνόλου ενεργών και τεχνικά επιβαρυνμένων περιοχών επιτρέπει πιο στοχευμένη χρήση των διαθέσιμων πόρων.

2.4.2 Μεταβολή Κώδικα (Code Churn) και Μετρικές Git (Git Metrics): Πώς Η Δραστηριότητα των Προγραμματιστών Επηρεάζει την Υγεία του Κώδικα

Τα συστήματα ελέγχου εκδόσεων (version control systems) καταγράφουν λεπτομερώς την εξέλιξη ενός έργου και δημιουργούν μια πλούσια πηγή δεδομένων για ανάλυση λογισμικού. Το Git δεν περιγράφει μόνο ποια είναι η τελευταία μορφή του κώδικα, αλλά διατηρεί πληροφορίες σχετικά με το πότε, από ποιον και σε τι έκταση έγιναν οι αλλαγές. Μία από τις απλούστερες μετρικές είναι ο αριθμός των υποβολών αλλαγών (commits) στα οποία συμμετείχε ένα αρχείο. Μια υψηλή τιμή δείχνει ότι το αρχείο ήταν συχνό σημείο αλλαγών, δεν αποκαλύπτει όμως την έκταση κάθε αλλαγής. Για τον λόγο αυτό χρησιμοποιείται η μεταβολή κώδικα (churn), που προσπαθεί να αποτυπώσει τον συνολικό όγκο αλλαγών και μπορεί να υπολογιστεί από τις γραμμές που προστέθηκαν και διαγράφηκαν. Οι προστιθέμενες γραμμές (lines added) και οι διαγραμμένες γραμμές (lines deleted) δίνουν πιο αναλυτική πληροφορία. Μεγάλη τιμή στις προστιθέμενες γραμμές μπορεί να αντιστοιχεί σε σημαντική επέκταση της λειτουργικότητας, ενώ υψηλή τιμή στις διαγραμμένες γραμμές (deleted lines) μπορεί να είναι αποτέλεσμα αφαίρεσης παλαιού κώδικα ή αναδόμησης. Ο αριθμός των διαφορετικών προγραμματιστών (authors) αποτελεί μία ακόμη διάσταση, στην οποία ο μεγάλος αριθμός δεν είναι πάντα αρνητικό χαρακτηριστικό. Όταν όμως συνδυάζεται με υψηλή πολυπλοκότητα και μεγάλη συχνότητα αλλαγών, μπορεί να αυξάνει την προσπάθεια συντονισμού και κατανόησης. Η χρησιμότητα των ιστορικών μετρικών έχει επιβεβαιωθεί και εμπειρικά. Έχει διαπιστωθεί ότι μετρικές που προέρχονται από το ιστορικό αλλαγών μπορούν να προσφέρουν σημαντική πληροφορία για την πρόβλεψη ελαττωμάτων, συμπληρώνοντας τα στατικά χαρακτηριστικά του κώδικα [17].

Αντίστοιχα, η μεταβολή του κώδικα έχει συσχετιστεί με την εμφάνιση ελαττωμάτων και μπορεί να χρησιμοποιηθεί ως ένδειξη αυξημένης αναπτυξιακής δραστηριότητας και πιθανού κινδύνου [18]. Ωστόσο, υψηλή μεταβολή κώδικα δεν σημαίνει πάντα ότι ένα αρχείο είναι προβληματικό ή ελαττωματικό, γι' αυτό και η συγκεκριμένη μετρική είναι προτιμότερο να εξετάζεται σε συνδυασμό με άλλα χαρακτηριστικά του κώδικα. Τα δεδομένα Git επηρεάζονται επίσης από τις πρακτικές της ομάδας. Μία ομάδα μπορεί να δημιουργεί πολλές μικρές υποβολές αλλαγών, ενώ μια άλλη να συγκεντρώνει μεγάλο αριθμό αλλαγών σε λιγότερες υποβολές αλλαγών. Συνεπώς, η άμεση σύγκριση του αριθμού των υποβολών αλλαγών μεταξύ διαφορετικών αποθετηρίων πρέπει να πραγματοποιείται με προσοχή.

Παρόμοια, το μέγεθος ενός αρχείου επηρεάζει φυσιολογικά την πιθανότητα να συγκεντρώσει μεγαλύτερη μεταβολή κώδικα. Ένα πολύ μεγάλο αρχείο έχει περισσότερες γραμμές που μπορούν να αλλάξουν από ένα μικρό. Για τον λόγο αυτό, οι ακατέργαστες τιμές είναι πιο χρήσιμες όταν εξετάζονται μέσα στο πλαίσιο του ίδιου αποθετηρίου ή μαζί με άλλες μετρικές. Η ηλικία ενός αρχείου μπορεί επίσης να επηρεάζει τη συνολική δραστηριότητα. Ένα στοιχείο λογισμικού που υπάρχει επί δέκα χρόνια έχει περισσότερες ευκαιρίες να συγκεντρώσει υποβολές αλλαγών από ένα που δημιουργήθηκε πριν από μερικούς μήνες. Ανάλογα με τον στόχο της ανάλυσης μπορεί να χρησιμοποιηθεί ολόκληρο το ιστορικό ή ένα συγκεκριμένο χρονικό παράθυρο.

Παρά τους περιορισμούς, οι μετρικές Git προσφέρουν μια σημαντική πληροφορία που δεν είναι διαθέσιμη στον στατικό κώδικα: πού πραγματοποιήθηκε πραγματική αναπτυξιακή δραστηριότητα. Το γεγονός αυτό τις καθιστά ιδιαίτερα χρήσιμες για την προτεραιοποίηση.

2.5 Σχετικά Εργαλεία και Πλατφόρμες

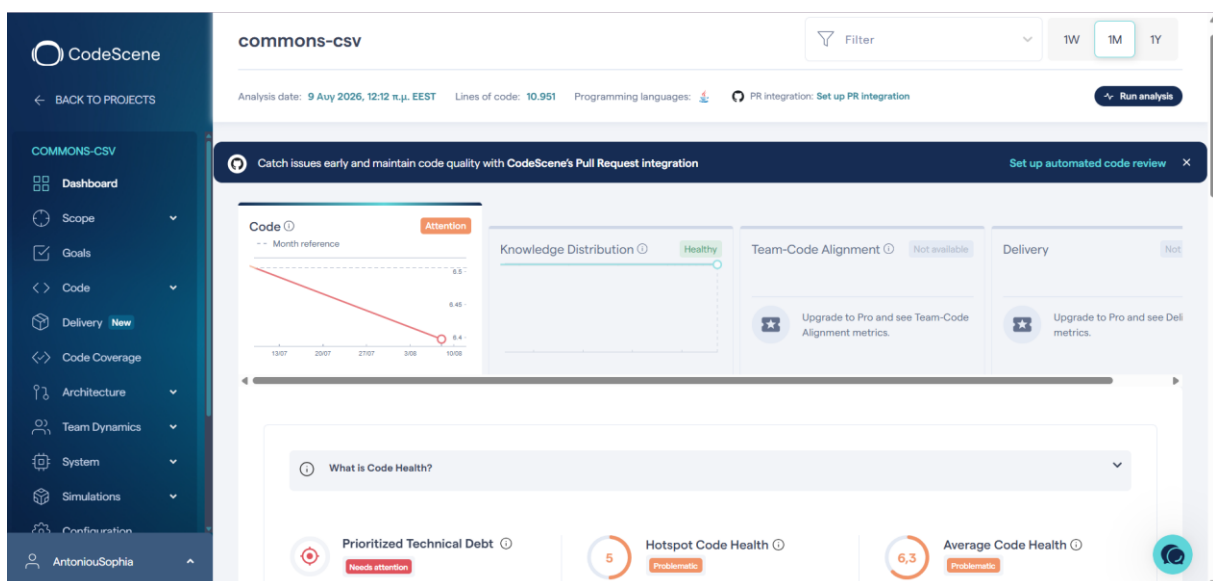
Η έρευνα γύρω από την ποιότητα και το τεχνικό χρέος έχει οδηγήσει σε μεγάλο αριθμό εργαλείων και πλατφορμών. Δεν ακολουθούν όλα την ίδια προσέγγιση, ούτε επιδιώκουν να μετρήσουν την ίδια έννοια. Ορισμένα επικεντρώνονται στις παραβιάσεις κανόνων και στη στατική ποιότητα (static quality), άλλα στη δομή των αντικειμενοστραφών συστημάτων (object-oriented systems) και άλλα στο ιστορικό και

στη συμπεριφορά της βάσης κώδικα (codebase). Η ετερογένεια αυτή είναι εμφανής και στα εργαλεία αξιολόγησης τεχνικού χρέους (Technical Debt assessment tools) [15]. Τα εργαλεία διαφέρουν ως προς τις μετρικές, τους κανόνες και τον τρόπο που μετατρέπουν τα τεχνικά ευρήματα σε ένδειξη χρέους. Το ότι υπάρχουν διαφορετικά αποτελέσματα δεν σημαίνει λοιπόν πως ένα εργαλείο είναι λανθασμένο. Συχνά σημαίνει ότι χρησιμοποιεί διαφορετική εννοιολογική κατασκευή και διαφορετικό μοντέλο αξιολόγησης (assessment model).

2.5.1 Επαγγελματικά Εργαλεία

Το CodeScene είναι χαρακτηριστικό παράδειγμα πλατφόρμας συμπεριφορικής ανάλυσης κώδικα (Behavioral Code Analysis). Βασική του διαφορά από τα κλασικά εργαλεία στατικής ανάλυσης είναι η αξιοποίηση του ιστορικού ανάπτυξης ως βασικής πηγής πληροφορίας. Η ανάλυση δεν εξετάζει μόνο την τεχνική κατάσταση του κώδικα, αλλά και το πού συγκεντρώνεται η πραγματική δραστηριότητα αλλαγών. Κεντρικό στοιχείο της προσέγγισης είναι τα hotspots, μέσω των οποίων η ιστορική δραστηριότητα συνδυάζεται με την τεχνική κατάσταση του κώδικα. Έτσι, δίνεται μεγαλύτερη έμφαση σε περιοχές που παρουσιάζουν τεχνικές αδυναμίες και παράλληλα συμμετέχουν ενεργά στην εξέλιξη του έργου [19].

Η υγεία του κώδικα (Code Health) χρησιμοποιείται για να αξιολογηθεί η τεχνική κατάσταση των περιοχών του κώδικα και λειτουργεί συμπληρωματικά προς τα hotspots [19]. Δεν πρέπει να εξισώνεται με την προσπάθεια αποκατάστασης (remediation effort) ενός εργαλείου στατικής ανάλυσης ή με έναν γενικό δείκτη τεχνικού χρέους, καθώς χρησιμοποιεί διαφορετικό μοντέλο αξιολόγησης. Επιπλέον, το CodeScene αξιοποιεί την ιστορική πληροφορία για να αναγνωρίσει σχέσεις που δεν είναι πάντα άμεσα εμφανείς από τη στατική δομή (static structure). Η συστηματική ταυτόχρονη μεταβολή διαφορετικών στοιχείων μπορεί να δείχνει λογική ή εξελικτική σύνδεση μεταξύ τους. Παράλληλα, καθώς το CodeScene χρησιμοποιεί το δικό του μοντέλο ανάλυσης, οι τελικές αξιολογήσεις του δεν είναι άμεσα ισοδύναμες με τα αποτελέσματα άλλων εργαλείων. Περισσότερη αξία έχει επομένως η εξέταση του τι μετρά κάθε εργαλείο και γιατί ένα συγκεκριμένο στοιχείο λογισμικού αποκτά υψηλή προτεραιότητα.



Εικόνα 2.2 - CodeScene Dashboard: Παράδειγμα ανάλυσης έργου

Για την παρούσα εργασία, το CodeScene αποτελεί κυρίως σημείο αναφοράς για την έννοια της προτεραιοποίησης με βάση τη συμπεριφορά του κώδικα (behavior-based prioritization). Η χρήση της

δραστηριότητας ως συμπληρωματικής διάστασης υποστηρίζει την επιλογή να μην εξετάζονται οι στατικές μετρικές απομονωμένα, αλλά να συνδέονται με το ιστορικό του αποθετηρίου.

2.5.2 Ερευνητικές Πλατφόρμες

Το AllinCode προσφέρει διαφορετική οπτική, με μεγαλύτερη έμφαση στη δομική ποιότητα των αντικειμενοστραφών συστημάτων και στην αναγνώριση κλάσεων που παρουσιάζουν αυξημένο σχετικό κίνδυνο τεχνικού χρέους. Η ανάλυση δεν οργανώνεται κυρίως γύρω από παραβιάσεις κανόνων, αλλά γύρω από χαρακτηριστικά όπως η πολυπλοκότητα, η σύζευξη, η συνοχή και το μέγεθος.

Οι δομικές μετρικές δεν περιγράφουν όλες την ίδια ιδιότητα. Η κυκλωματική πολυπλοκότητα σχετίζεται με την πολυπλοκότητα της ροής ελέγχου, ενώ η απόκριση για μια κλάση (RFC - Response For a Class) αποτυπώνει το εύρος της συμπεριφοράς που μπορεί να ενεργοποιηθεί ως απόκριση σε μήνυμα προς ένα αντικείμενο της κλάσης. Οι μετρικές σύζευξης εξετάζουν τις σχέσεις και εξαρτήσεις μεταξύ διαφορετικών στοιχείων. Αυξημένη σύζευξη μπορεί να σημαίνει ότι η κατανόηση ή η τροποποίηση μιας κλάσης χρειάζεται γνώση περισσότερων εξωτερικών στοιχείων και ότι μία αλλαγή έχει μεγαλύτερη πιθανότητα να επηρεάσει άλλα σημεία του συστήματος. Η συνοχή αντίστοιχα αφορά το κατά πόσο οι λειτουργίες μιας κλάσης συνδέονται με μια κοινή και σαφή ευθύνη. Χαμηλή συνοχή μπορεί να δείχνει ότι διαφορετικές αρμοδιότητες έχουν συγκεντρωθεί στο ίδιο στοιχείο. Οι μετρικές μεγέθους συμπληρώνουν την εικόνα, καθώς ένα πολύ μεγάλο στοιχείο λογισμικού μπορεί να απαιτεί αυξημένη προσπάθεια κατανόησης. Ωστόσο, καμία από τις παραπάνω τιμές δεν αρκεί απομονωμένα για να χαρακτηρίσει μια κλάση ως κακή ή ως τεχνικό χρέος.

Η αξία της πολυμετρικής προσέγγισης βρίσκεται ακριβώς στον συνδυασμό διαφορετικών χαρακτηριστικών. Μια μεγάλη κλάση μπορεί να παρουσιάζει υψηλή συνοχή και καθαρή αρμοδιότητα, ενώ μια μικρότερη κλάση μπορεί να εμφανίζει έντονες εξαρτήσεις και προβληματική εσωτερική οργάνωση. Η εξέταση περισσότερων χαρακτηριστικών μειώνει τον κίνδυνο μιας απόφασης βασισμένης αποκλειστικά σε μία ακραία τιμή. Στο AllinCode ιδιαίτερη σημασία έχει η αναγνώριση κλάσεων υψηλού κινδύνου τεχνικού χρέους (High-TD risk classes). Η ένδειξη αυτή δεν σημαίνει ότι έχει αποδειχθεί η ύπαρξη ενός συγκεκριμένου στοιχείου τεχνικού χρέους, ούτε ότι πρέπει αυτόματα να γίνει αναδόμηση. Εκφράζει αυξημένο κίνδυνο σύμφωνα με τα χαρακτηριστικά και το μοντέλο που χρησιμοποιείται. Η χρήση προβλεπτικών μοντέλων δημιουργεί παράλληλα ανάγκη ερμηνευσιμότητας. Μία απλή ένδειξη “Υψηλός Κίνδυνος” έχει περιορισμένη χρησιμότητα όταν δεν είναι γνωστό ποιοι παράγοντες την προκάλεσαν. Για τον λόγο αυτό, σημαντική είναι η δυνατότητα εξήγησης της συμβολής των διαφορετικών χαρακτηριστικών. Οι εξηγήσεις που βασίζονται στη μέθοδο SHAP (SHapley Additive exPlanations) επιτρέπουν να εξεταστεί η συμβολή κάθε χαρακτηριστικού σε μία συγκεκριμένη πρόβλεψη. Έτσι, αντί ο χρήστης να βλέπει μόνο ότι μια κλάση έχει υψηλό κίνδυνο, μπορεί να διερευνήσει αν η πρόβλεψη επηρεάζεται περισσότερο από την πολυπλοκότητα, τη σύζευξη, τη συνοχή ή το μέγεθος. Πέρα από την ερμηνευσιμότητα των προβλέψεων όμως, υπάρχει και ένας πιο πρακτικός περιορισμός: η μονάδα ανάλυσης (unit of analysis) που χρησιμοποιεί το εργαλείο. Το AllinCode λειτουργεί κυρίως σε επίπεδο κλάσης, ενώ άλλες πηγές μπορούν να επιστρέφουν δεδομένα σε επίπεδο αρχείου. Επομένως, η ενοποίηση απαιτεί διαδικασία αντιστοίχισης (mapping), ιδιαίτερα όταν ένα αρχείο περιέχει περισσότερες από μία κλάσεις.

2.5.3 Σύγκριση και Ανάγκη για ένα Ενοποιημένο Dashboard

Τα τρία εργαλεία, όπως φάνηκε παραπάνω, απαντούν σε συμπληρωματικά ερωτήματα. Ένα εργαλείο στατικής ανάλυσης εξετάζει κυρίως την τρέχουσα μορφή του κώδικα, μια δομική πλατφόρμα εστιάζει

στις σχέσεις και στην οργάνωση των κλάσεων, ενώ η συμπεριφορική ανάλυση εξετάζει τον τρόπο με τον οποίο η πραγματική δραστηριότητα ανάπτυξης κατανέμεται μέσα στη βάση κώδικα. Η στατική ανάλυση έχει το πλεονέκτημα της αυτοματοποίησης και της λεπτομερούς αναγνώρισης συγκεκριμένων παραβιάσεων. Μπορεί να παρέχει πληροφορίες σχετικά με τεχνικές μετρικές και παραβιάσεις. Ωστόσο, η τρέχουσα μορφή του κώδικα δεν αποκαλύπτει από μόνη της πόσο συχνά ένα στοιχείο λογισμικού συμμετέχει στην πραγματική συντήρηση. Η συμπεριφορική προσέγγιση συμπληρώνει αυτή την πληροφορία μέσω του ιστορικού των αλλαγών, αναγνωρίζοντας τα στοιχεία στα οποία συγκεντρώνεται πραγματική δραστηριότητα. Από μόνη της όμως, η μεγάλη συχνότητα αλλαγών δεν αποτελεί ένδειξη χαμηλής ποιότητας. Η δομική ανάλυση προσθέτει μια ακόμη διάσταση. Μετρικές όπως η σύζευξη, η συνοχή, το μέγεθος και η πολυπλοκότητα μπορούν να αποκαλύψουν αδυναμίες της αντικειμενοστραφούς σχεδίασης που δεν αποτυπώνονται πλήρως μόνο από το πλήθος των παραβιάσεων κανόνων. Παράλληλα, δεν παρέχουν άμεσα πληροφορία σχετικά με τη συχνότητα πραγματικών αλλαγών.

Η πρακτική αξία των τριών πηγών προκύπτει επομένως από τον συνδυασμό τους. Ένα αρχείο μπορεί, για παράδειγμα, να εμφανίζει αυξημένες οσμές κώδικα και προσπάθεια αποκατάστασης, υψηλή σύζευξη, χαμηλή συνοχή και παράλληλα μεγάλη μεταβολή κώδικα στο Git. Η συνύπαρξη αυτών των ενδείξεων δημιουργεί ισχυρότερη βάση για περαιτέρω διερεύνηση από οποιαδήποτε μεμονωμένη μέτρηση. Αντίθετα, ένα αρχείο μπορεί να εμφανίζει υψηλή πολυπλοκότητα αλλά ελάχιστη δραστηριότητα αλλαγών. Η πληροφορία παραμένει σημαντική, όμως η προτεραιότητά του μπορεί να είναι διαφορετική. Έτσι, η ενσωμάτωση του ιστορικού επιτρέπει μια πιο πρακτική αξιολόγηση του κινδύνου συντήρησης. Η ενοποίηση των δεδομένων ωστόσο, δεν περιορίζεται σε μια απλή ένωση αριθμητικών τιμών. Κάθε πηγή χρησιμοποιεί διαφορετικό επίπεδο ανάλυσης, διαφορετικές μετρικές και διαφορετικό τρόπο οργάνωσης των αποτελεσμάτων. Το SonarQube παρέχει μεγάλο μέρος της πληροφορίας σε επίπεδο αρχείου και ζητήματος, το AllinCode επικεντρώνεται κυρίως σε κλάσεις και δομικές μετρικές, ενώ το Git καταγράφει την ιστορική δραστηριότητα ανά διαδρομή αρχείου. Για τον σωστό συνδυασμό τους απαιτούνται καθαρισμός (cleaning), τυποποίηση (standardization) και αντιστοίχιση (mapping) των επιμέρους εγγραφών.

Επιπλέον, δύο μετρικές με παρόμοιο όνομα δεν είναι αναγκαστικά αριθμητικά ισοδύναμες. Διαφορετικοί αναλυτές (parsers), επίπεδα ανάλυσης και παραδοχές μπορούν να οδηγήσουν σε αποκλίσεις. Επομένως, στόχος δεν είναι να εξαναγκαστούν οι πηγές να συμφωνούν αριθμητικά, αλλά να αξιοποιηθεί η καθεμία για τη διάσταση που μπορεί να περιγράψει. Ιδιαίτερη σημασία έχει και η διαχείριση ελλিপών δεδομένων. Η απουσία μιας μέτρησης δεν πρέπει να μετατρέπεται αυθαίρετα σε μηδενική τιμή. Ένα αρχείο που δεν αναλύθηκε από μια πηγή δεν είναι ισοδύναμο με ένα αρχείο που αναλύθηκε και παρουσίασε μηδενικό κίνδυνο. Η διαφάνεια σχετικά με τη διαθεσιμότητα των πηγών είναι επομένως απαραίτητη για μια αξιόπιστη ενοποίηση.

Ένα ενοποιημένο dashboard πρέπει ακόμη να επιτρέπει τη μετάβαση σε διαφορετικά επίπεδα πληροφορίας. Μια συνοπτική ένδειξη είναι χρήσιμη για την αρχική εικόνα του έργου, αλλά ο χρήστης πρέπει να μπορεί να διερευνήσει ποια διάσταση δημιούργησε τον κίνδυνο και ποια συγκεκριμένα αρχεία και μετρικές βρίσκονται πίσω από την αντίστοιχη τιμή [6]. Η παρουσίαση μεγάλου αριθμού μετρικών μπορεί να οδηγήσει σε υπερφόρτωση πληροφορίας, επομένως η σύνοψη πρέπει να συνοδεύεται από δυνατότητα αναλυτικής εμβάθυνσης (drill-down). Η χρονική διάσταση αποτελεί επιπλέον πλεονέκτημα ενός ενοποιημένου περιβάλλοντος. Η αποθήκευση διαδοχικών αναλύσεων επιτρέπει τη σύγκριση της κατάστασης του έργου σε διαφορετικές χρονικές στιγμές και την παρακολούθηση της εξέλιξής του. Έτσι, η αξιολόγηση δεν περιορίζεται σε ένα μεμονωμένο στιγμιότυπο, αλλά αποτυπώνει και τη μεταβολή της κατάστασης με την πάροδο του χρόνου. Σημαντική είναι επίσης η ερμηνευσιμότητα

(interpretability) των σύνθετων δεικτών. Μελέτες δείχνουν ότι η αυτοματοποίηση είναι επιθυμητή, χωρίς όμως να αφαιρεί από τον χρήστη τον έλεγχο των αποτελεσμάτων [9]. Η ενοποίηση πολλών μετρικών μπορεί να διευκολύνει τη σύνοψη, αλλά δημιουργεί τον κίνδυνο ενός δείκτη (score) του οποίου η προέλευση δεν είναι εμφανής. Για τον λόγο αυτό πρέπει να διατηρούνται οι ακατέργαστες μετρικές και οι ενδιάμεσες συνιστώσες.

Στην προτεινόμενη προσέγγιση, το SonarQube παρέχει πληροφορίες για τη στατική ποιότητα, το AllinCode συμπληρώνει την ανάλυση με δομικές μετρικές και ενδείξεις υψηλού κινδύνου τεχνικού χρέους (High-TD), ενώ το Git προσθέτει την ιστορική διάσταση μέσω στοιχείων όπως οι υποβολές αλλαγών, οι διαφορετικοί προγραμματιστές και η μεταβολή κώδικα. Οι πληροφορίες κάθε πηγής παραμένουν διακριτές, ενώ διατηρείται και η προέλευση κάθε μέτρησης. Με αυτή την οργάνωση, ο χρήστης μπορεί να ξεκινά από τη συνολική εικόνα του έργου, να εξετάζει τις διαφορετικές διαστάσεις κινδύνου, να εντοπίζει hotspots και στη συνέχεια να προχωρά στις επιμέρους μετρικές ενός συγκεκριμένου αρχείου ή κλάσης. Η συνοπτική παρουσίαση και η αναλυτική διερεύνηση λειτουργούν έτσι ως διαφορετικά επίπεδα της ίδιας διαδικασίας αξιολόγησης. Στόχος δεν είναι η παραγωγή ενός ακόμη αριθμητικού δείκτη, αλλά η δημιουργία ενός περιβάλλοντος στο οποίο διαφορετικές μορφές τεχνικής πληροφορίας μπορούν να συνδυάζονται και να αξιοποιούνται για την υποστήριξη αποφάσεων συντήρησης και προτεραιοποίησης. Πάνω σε αυτή τη λογική βασίζεται και η μεθοδολογία που ακολουθείται στην εργασία.

Κεφάλαιο 3ο: Μεθοδολογία και Σχεδιασμός Συστήματος

3.1 Αρχιτεκτονική Συστήματος

Η εφαρμογή ακολουθεί αρθρωτή αρχιτεκτονική, στην οποία η συλλογή δεδομένων, η επεξεργασία, η αποθήκευση και η παρουσίαση αποτελούν ξεχωριστά υποσυστήματα με ορισμένες ευθύνες. Η συνολική ροή συνοψίζεται ως εξής:

GitHub Repository → Node.js Back-end → SonarQube, AllinCode και Git

→ Python Processing → MariaDB → REST API → React Dashboard

Το σημείο εκκίνησης είναι ένα δημόσιο αποθετήριο του GitHub. Ο χρήστης καταχωρεί το URL του έργου ή ζητά επανάληψη ανάλυσης ενός ήδη αποθηκευμένου έργου. Στη συνέχεια, το Node.js back-end αναλαμβάνει τον συντονισμό των επιμέρους σταδίων, λειτουργώντας ως το μοναδικό σημείο επικοινωνίας του front-end με τις πηγές δεδομένων. Το SonarQube εκτελεί στατική ανάλυση του πηγαίου κώδικα και το back-end παίρνει τις απαιτούμενες μετρικές μέσω του Web API. Μετά, το Git επεξεργάζεται το ιστορικό του αποθετηρίου και παράγει τις μετρικές δραστηριότητας ανά αρχείο.

Τα δεδομένα των τριών πηγών διαφέρουν ως προς τη μορφή, την ονομασία των πεδίων και τον αριθμό των αρχείων που καλύπτουν. Για τον λόγο αυτό, δεν χρησιμοποιούνται απευθείας από το front-end. Αρχικά, επεξεργάζονται από ένα ανεξάρτητο αρχείο python (python script), το οποίο κανονικοποιεί τις διαδρομές, ενοποιεί τις εγγραφές και εφαρμόζει τους τύπους της μεθοδολογίας. Μετά την επιτυχή του ολοκλήρωση, το back-end αποθηκεύει το νέο στιγμιότυπο στη MariaDB. Τέλος, το front-end λαμβάνει το αποθηκευμένο αποτέλεσμα μέσω του REST API, για να το παρουσιάσει στις επιμέρους οθόνες του dashboard. Το React front-end δεν επικοινωνεί ποτέ απευθείας με το SonarQube, το AllinCode ή τη βάση δεδομένων, παραμένοντας έτσι ανεξάρτητο από τις λεπτομέρειες συλλογής και επεξεργασίας. Η αρχιτεκτονική αυτή ακολουθεί τον διαχωρισμό συλλογής, ανάλυσης και παρουσίασης έχουν και άλλα εργαλεία παρακολούθησης τεχνικού χρέους [23].



Εικόνα 3.1 - Συνολική αρχιτεκτονική του συστήματος

Το διάγραμμα παρουσιάζει τα κύρια υποσυστήματα και τις μεταξύ τους συνδέσεις, χωρίς να περιλαμβάνει κάθε αρχείο ή συνάρτηση της εφαρμογής. Δείχνει καθαρά τις πηγές δεδομένων, το επίπεδο συντονισμού, το υποσύστημα επεξεργασίας, τη βάση και τη διαδρομή μέσω της οποίας τα αποτελέσματα καταλήγουν στο dashboard.

3.1.1 Σχεδιασμός του Back-end

Το back-end είναι το κεντρικό σημείο συντονισμού της εφαρμογής. Δέχεται αιτήματα από το front-end, επικοινωνεί με τα εξωτερικά εργαλεία, ενεργοποιεί τον μηχανισμό επεξεργασίας και αποθηκεύει ή ανακτά δεδομένα από τη MariaDB. Η εσωτερική οργάνωση βασίζεται στον διαχωρισμό των διαδρομών του API από τις υπηρεσίες που αναλαμβάνουν συγκεκριμένες λειτουργίες:

- εκτέλεση και ανάκτηση δεδομένων του SonarQube
- ανάλυση του Git ιστορικού
- αυθεντικοποίηση και επικοινωνία με το AllinCode
- εκτέλεση της επεξεργασίας και του υπολογισμού των δεικτών σε Python
- πρόσβαση στη βάση δεδομένων

Αυτός ο διαχωρισμός διαδρομών και υπηρεσιών μειώνει τη σύζευξη. Μια αλλαγή σε διαδρομή του AllinCode για παράδειγμα, μπορεί να μείνει μέσα στην αντίστοιχη υπηρεσία χωρίς να αλλάξει όλο το back-end ή το front-end.

Η βασική διαδρομή ανάλυσης δέχεται το URL του αποθετηρίου, εκτελεί διαδοχικά τα στάδια συλλογής δεδομένων και δημιουργεί την είσοδο για το αρχείο επεξεργασίας σε Python. Μετά την επιτυχή ολοκλήρωση της επεξεργασίας, καλεί την υπηρεσία αποθήκευσης και επιστρέφει στο front-end τη νέα ανάλυση. Για την ανάκτηση των αποθηκευμένων δεδομένων χρησιμοποιούνται ξεχωριστές διαδρομές, οι οποίες επιστρέφουν τη λίστα των έργων μαζί με την τελευταία τους ανάλυση.

3.1.2 Σχεδιασμός του Μηχανισμού Επεξεργασίας

Ο μηχανισμός επεξεργασίας έχει σχεδιαστεί ως ανεξάρτητο υποσύστημα σε Python, το οποίο μετατρέπει τα δεδομένα των τριών πηγών σε ενιαίο σύνολο αποτελεσμάτων εφαρμόζοντας το μοντέλο αξιολόγησης της ενότητας 3.3. Το πρώτο στάδιο αφορά την προετοιμασία των διαδρομών. Μπορεί να επιστρέφονται με διαφορετική μορφή από το SonarQube, το AllinCode και το Git, γι' αυτό μετατρέπονται αρχικά σε κοινή μορφή. Στη συνέχεια δημιουργείται μία ενοποιημένη εγγραφή ανά διαδρομή, με τις διαθέσιμες Git, SonarQube και AllinCode μετρικές και τους αντίστοιχους δείκτες παρουσίας (flags) κάθε πηγής. Μετά την ενοποίηση εφαρμόζονται οι κανόνες επιλεξιμότητας και σχηματίζονται οι πληθυσμοί G, S και A, καθώς και ο κοινός πληθυσμός:

$$C = G \cap S \cap A$$

όπου G τα επιλέξιμα αρχεία με διαθέσιμα δεδομένα Git, το S όσα έχουν δεδομένα SonarQube, το A όσα έχουν δεδομένα AllinCode. Υπολογίζονται οι τιμές αναφοράς P95 και κανονικοποιούνται οι ακατέργαστες μετρικές για τον υπολογισμό των δεικτών σε επίπεδο αρχείου. Στη συνέχεια, υπολογίζονται οι επιμέρους κίνδυνοι σε επίπεδο έργου. Οι κίνδυνοι αυτοί συνδυάζονται στο Overall Project Risk, από το οποίο προκύπτει το Global Health Score. Ο μηχανισμός διατηρεί επίσης τα ενδιάμεσα στοιχεία που απαιτούνται για την επαλήθευση των υπολογισμών. Όταν ένας δείκτης δεν μπορεί να υπολογιστεί λόγω ελλিপών δεδομένων, επιστρέφεται ως μη διαθέσιμος. Έτσι, η απουσία μιας πηγής δεν παράγει τεχνητά χαμηλό αποτέλεσμα κινδύνου.

3.1.3 Σχεδιασμός της Παρουσίασης

Το dashboard οργανώνεται σε πέντε οθόνες, οι οποίες ακολουθούν τη μετάβαση από το γενικό προς το ειδικό. Η οθόνη Overview παρουσιάζει τη συνολική εικόνα της τελευταίας αποθηκευμένης ανάλυσης. Περιλαμβάνει το Global Health Score, το Project Risk Breakdown, κάποιες βασικές μετρικές σε επίπεδο έργου και επιλεγμένα προειδοποιητικά μηνύματα. Η Temporal Evolution συνδυάζει τη χρονική εξέλιξη επιλεγμένων τιμών τεχνικού χρέους με τα αποτελέσματα High-TD της ανάλυσης. Η Hotspots περιλαμβάνει το γράφημα των hotspots, τις λεπτομέρειες του επιλεγμένου αρχείου και τον πίνακα Refactoring Priority. Η Quality & Structure δείχνει τα προβλήματα ποιότητας και τις αντικειμενοστραφείς δομικές μετρικές. Τέλος, η οθόνη All File Metrics δίνει πρόσβαση στο σύνολο των αποθηκευμένων εγγραφών σε επίπεδο αρχείου. Υπάρχει δυνατότητα αναζήτησης βάσει διαδρομής, φίλτρα, ταξινόμηση, σελιδοποίηση και εξαγωγή σε CSV για χρήση εκτός του dashboard.

3.2 Διαδικασία Συλλογής Δεδομένων

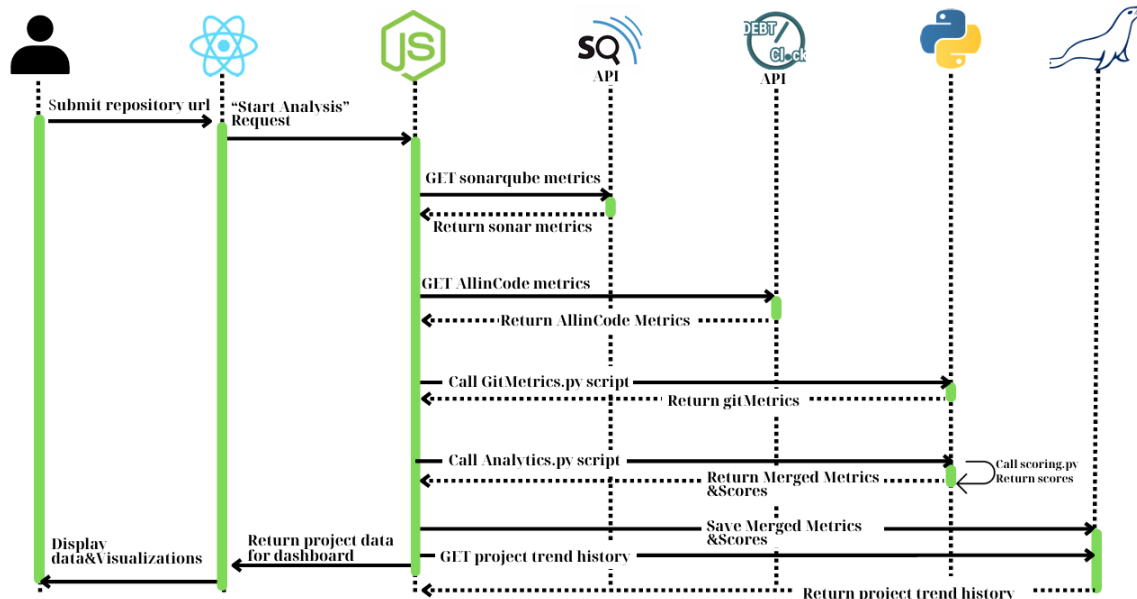
3.2.1 Πηγές και δεδομένα

Από το SonarQube ανακτώνται συνολικές και ανά αρχείο μετρικές που αφορούν την πολυπλοκότητα, την ποιότητα του κώδικα, την επανάληψη και το τεχνικό χρέος. Τα δεδομένα αυτά είναι η βασική πηγή για τους δείκτες ποιότητας, πολυπλοκότητας, διπλότυπου κώδικα και τεχνικού χρέους. Η επικοινωνία με το AllinCode περιλαμβάνει αυθεντικοποίηση, έλεγχο ύπαρξης του έργου και έναρξη ανάλυσης. Αν το AllinCode δεν είναι διαθέσιμο ή κάποια διαδικασία αποτύχει, η εφαρμογή συνεχίζει με τα δεδομένα του SonarQube και του Git, ενώ οι αντίστοιχες AllinCode συνιστώσες παραμένουν μη διαθέσιμες αντί να μετατρέπονται σε μηδενικές τιμές. Τέλος, από το Git εξάγονται μετρικές ανά αρχείο με βάση το ιστορικό του αποθετηρίου. Σε αυτές περιλαμβάνονται ο αριθμός των υποβολών αλλαγών, η μεταβολή κώδικα, οι γραμμές που προστέθηκαν ή διαγράφηκαν και ο αριθμός των διαφορετικών προγραμματιστών που συμμετείχαν στις αλλαγές.

3.2.2 Ροή Εκτέλεσης μιας Ανάλυσης

Η διαδικασία ξεκινά όταν ο χρήστης εισάγει το URL ενός αποθετηρίου ή ζητά επανάληψη της ανάλυσης για ένα αποθηκευμένο έργο. Το front-end στέλνει το URL στο back-end και εμφανίζει κατάσταση φόρτωσης μέχρι να ολοκληρωθεί η διαδικασία. Αρχικά εκτελείται η ανάλυση του έργου στο SonarQube. Το back-end προετοιμάζει το αποθετήριο, εκτελεί το SonarScanner και ανακτά τις συνολικές και ανά αρχείο μετρικές. Στη συνέχεια, εξάγονται από το Git οι μετρικές ιστορικής δραστηριότητας για κάθε αρχείο. Ακολουθεί η ροή του AllinCode. Το back-end πραγματοποιεί την απαιτούμενη αυθεντικοποίηση και ελέγχει αν το έργο είναι ήδη καταχωρισμένο στην πλατφόρμα. Εάν ναι ξεκινά απλώς νέα ανάλυση, ενώ για νέο αποθετήριο προηγείται η δημιουργία του και μετά ακολουθεί η πρώτη ανάλυση. Αν υπάρχει ήδη διαδικασία σε εξέλιξη, δεν ξεκινά δεύτερη, αλλά συνεχίζεται η αναμονή της ενεργής εκτέλεσης. Η κατάσταση ελέγχεται περιοδικά (polling) μέχρι να γίνουν διαθέσιμα τα νέα αποτελέσματα. Μετά την ολοκλήρωση της βασικής ανάλυσης ανανεώνονται και τα High-TD αποτελέσματα. Αν η ροή αυτή αποτύχει, το back-end καταγράφει το σφάλμα και συνεχίζει με τα διαθέσιμα SonarQube και Git δεδομένα, με τις AllinCode συνιστώσες να σημειώνονται ως μη διαθέσιμες. Όταν ολοκληρωθεί η συλλογή, τα δεδομένα μετατρέπονται σε κοινή δομή εισόδου και περνούν στο υποσύστημα επεξεργασίας σε Python. Εκεί πραγματοποιούνται η κανονικοποίηση των διαδρομών, η ενοποίηση των δεδομένων, ο έλεγχος επιλεξιμότητας και υπολογίζονται οι δείκτες σε επίπεδο αρχείου και έργου.

Η αποθήκευση στη MariaDB πραγματοποιείται μετά την επιτυχή ολοκλήρωση της επεξεργασίας. Δημιουργείται ή ενημερώνεται η εγγραφή του έργου, εισάγεται νέα ανάλυση στο `analysis_run`, αποθηκεύονται οι αντίστοιχες μετρικές στο `merged_file_metrics` και προστίθεται νέα εγγραφή στο `project_trend_history`. Στο τέλος διαγράφονται οι πλήρεις αναλύσεις που είναι παλαιότερες από τις δύο πιο πρόσφατες του συγκεκριμένου έργου. Το back-end επιστρέφει το id της νέας ανάλυσης, το front-end το ανακτά μέσω του REST API και εμφανίζει το dashboard. Αν αποτύχει κάποιο κρίσιμο στάδιο πριν από την αποθήκευση, το αίτημα ολοκληρώνεται με μήνυμα σφάλματος και δεν δημιουργείται νέα εγγραφή στο `analysis_run`.



Εικόνα 3.2 - Ροή εκτέλεσης μιας ανάλυσης

3.3 Το Μοντέλο Αξιολόγησης

Οι επιμέρους μετρικές κανονικοποιούνται και συνδυάζονται ώστε να παραχθούν σύνθετοι δείκτες κινδύνου και ποιότητας. Η διαδικασία αυτή ακολουθεί τις αρχές της πολυκριτηριακής αξιολόγησης. Σύμφωνα με αυτές, μετρικές διαφορετικών κλιμάκων μετατρέπονται σε συγκρίσιμες τιμές πριν συνδυαστούν μέσω συντελεστών βαρύτητας [24]. Με τον τρόπο αυτό, το dashboard υποστηρίζει την ιεράρχηση των αρχείων που απαιτούν έλεγχο ή αναδόμηση και την παρακολούθηση της εξέλιξης του τεχνικού χρέους στον χρόνο [25].

Οι συντελεστές βαρύτητας αποτελούν παραμέτρους του προτεινόμενου μοντέλου. Η επιλογή τους έγινε με βάση τα εμπειρικά ευρήματα της βιβλιογραφίας, τους στόχους της προσέγγισης και τη διαδικασία σχεδιαστικής αξιολόγησης κατά την υλοποίηση. Δεδομένου ότι η προτεραιοποίηση του τεχνικού χρέους εξαρτάται από τα χαρακτηριστικά και τις απαιτήσεις κάθε έργου, δεν υπάρχει ένα καθολικά αποδεκτό σύνολο βαρών ή παραγόντων αξιολόγησης [26].

3.3.1 Κανονικοποίηση Δεδομένων και Βασικοί Κανόνες Υπολογισμού

Πριν υπολογιστούν οι επιμέρους δείκτες, εφαρμόζονται κάποιοι κανόνες επιλεξιμότητας και βοηθητικοί μετασχηματισμοί. Αυτό είναι απαραίτητο επειδή οι τρεις πηγές δεδομένων δεν καλύπτουν ακριβώς τα ίδια αρχεία, και επειδή οι μετρικές έχουν διαφορετικές μονάδες και κλίμακες.

3.3.1.1 Κριτήρια Επιλεξιμότητας και Πληθυσμοί Αρχείων

Όταν ενοποιούνται τα αποτελέσματα των τριών πηγών, προκύπτει ένα σύνολο εγγραφών σε επίπεδο αρχείου. Δεν είναι όμως όλες οι εγγραφές κατάλληλες για κάθε υπολογισμό. Ένα αρχείο μπορεί να υπάρχει μόνο στο ιστορικό του Git, να έχει αναλυθεί από το SonarQube αλλά όχι από το AllinCode ή να διαθέτει δομικές μετρικές χωρίς αντίστοιχη πρόβλεψη High-TD. Γι' αυτό η επιλεξιμότητα καθορίζεται κάθε φορά από τις πηγές που χρειάζεται ο συγκεκριμένος δείκτης. Ως βασικά επιλέξιμο θεωρείται ένα τρέχον αρχείο πηγαίου κώδικα που δεν είναι αρχείο δοκιμών (test file) και δεν ανήκει σε γνωστό generated, build ή dependency path. Αποκλείονται για παράδειγμα αρχεία κάτω από καταλόγους όπως: target, build, dist, node_modules, vendor και generated. Ακόμη, αποκλείονται παλιές διαδρομές Git (Git-only paths) που δεν αντιστοιχούν πια σε ενεργό αρχείο του έργου. Ορίζονται τέσσερα σύνολα αρχείων:

- το G περιλαμβάνει τα επιλέξιμα αρχεία με διαθέσιμα δεδομένα Git
- το S όσα έχουν δεδομένα SonarQube
- το A όσα έχουν δεδομένα AllinCode
- το H όσα αρχεία επέστρεψε η διαδρομή High-TD του AllinCode API μαζί με διαθέσιμη πιθανότητα.

Επειδή η συγκεκριμένη διαδρομή (endpoint) επιστρέφει μόνο τα αρχεία που έχουν χαρακτηριστεί ως High-TD, το H δεν εκφράζει το πλήρες σύνολο των αρχείων που αξιολογήθηκαν από το μοντέλο. Για τους πιο σύνθετους υπολογισμούς, που συνδυάζουν και τις τρεις πηγές, χρησιμοποιείται:

$$C = G \cap S \cap A$$

Η χρήση αυτού του κοινού πληθυσμού αποτρέπει τη σύγκριση τιμών που έχουν υπολογιστεί πάνω σε διαφορετικά υποσύνολα αρχείων. Έτσι, ένα αρχείο χωρίς δεδομένα AllinCode για παράδειγμα, δεν παίρνει τεχνητά structural_risk_score = 0 και δεν φαίνεται λανθασμένα πιο υγιές από ένα αρχείο που έχει αναλυθεί πλήρως. Κάθε δείκτης χρειάζεται τη δική του κάλυψη δεδομένων:

- το Git Activity Score υπολογίζεται στο G
- το Complexity Score και το Quality Risk Score στο S
- το Structural Risk Score στο A
- το Hotspot Score στην τομή $G \cap S$
- το Refactoring Priority Score στον κοινό πληθυσμό $C = G \cap S \cap A$
- το High-TD Risk Score στο H

Οι ανεξάρτητες προβολές του dashboard μπορούν να χρησιμοποιούν το ευρύτερο έγκυρο σύνολο της κάθε πηγής. Ο πίνακας δομικών μετρικών μπορεί να δείχνει όλα τα επιλέξιμα αρχεία του A. Αντίθετα, το Refactoring Priority και οι δείκτες κινδύνου σε επίπεδο αρχείου, χρησιμοποιούν πάντα τον κοινό πληθυσμό C. Η απουσία δεδομένων δεν αντιμετωπίζεται ποτέ ως πραγματική τιμή μηδέν. Το μηδέν σημαίνει ότι η μέτρηση ήταν διαθέσιμη και δεν βρήκε δραστηριότητα ή πρόβλημα. Όταν η απαιτούμενη πηγή λείπει, ο δείκτης χαρακτηρίζεται μη διαθέσιμος, αποθηκεύεται ως NULL/None και εμφανίζεται ως N/A ή Not flagged, ανάλογα την περίπτωση.

3.3.1.2 Κανονικοποίηση με Βάση το P95

Οι μετρικές σε επίπεδο αρχείου κανονικοποιούνται ως προς το 95ο εκατοστημόριο της κατανομής τους μέσα στο ίδιο project:

$$Normalized\ value = \min\left(\frac{value}{P_{95value}}, 1\right)$$

Η επιλογή του P95 αντί της απόλυτης μέγιστης τιμής βασίζεται στο γεγονός ότι οι κατανομές πολλών μετρικών λογισμικού είναι έντονα ασύμμετρες. Οι περισσότερες κλάσεις εμφανίζουν χαμηλές τιμές, ενώ ένα μικρό ποσοστό παρουσιάζει πολύ υψηλές τιμές. Αυτό έχει παρατηρηθεί και στις αντικειμενοστραφείς μετρικές, όπου λίγες κλάσεις εμφανίζουν πολύ υψηλή πολυπλοκότητα και αυξημένο κίνδυνο [27]. Αν χρησιμοποιούνταν η απόλυτη μέγιστη τιμή, ένα και μόνο ακραίο αρχείο θα μπορούσε να τραβήξει υπερβολικά το σημείο αναφοράς και να συμπίσει προς το μηδέν τις τιμές (scores) όλων των υπόλοιπων αρχείων. Με το P95, οι τιμές πάνω από το 95ο εκατοστημόριο απλώς περιορίζονται στο 1, χωρίς να αλλάζει η κλίμακα για το μεγαλύτερο μέρος των αρχείων.

Για να παραμένουν συγκρίσιμα τα επιμέρους στοιχεία του Refactoring Priority, οι τιμές αναφοράς P95 υπολογίζονται κατά προτεραιότητα πάνω στον κοινό πληθυσμό C. Αν μια μετρική δεν έχει καμία θετική τιμή μέσα σε αυτόν τον πληθυσμό, χρησιμοποιείται σαν εναλλακτική λύση το αντίστοιχο ευρύτερο σύνολο G, S ή A. Οι πραγματικές μηδενικές τιμές δεν συμμετέχουν στον υπολογισμό του σημείου αναφοράς, αλλά διατηρούνται ως έγκυρες μηδενικές μετρήσεις κατά τον υπολογισμό του score του αντίστοιχου αρχείου. Αν το P95_value είναι μηδέν, τότε και η κανονικοποιημένη τιμή ορίζεται ως μηδέν, ώστε να αποφεύγεται η διαίρεση με το μηδέν. Επειδή ο υπολογισμός αυτός γίνεται ξεχωριστά για κάθε αποθετήριο, οι δείκτες σε επίπεδο αρχείου λειτουργούν κυρίως ως σχετικές τιμές εσωτερικής προτεραιοποίησης. Ένα complexity_score = 80, σημαίνει ότι το αρχείο βρίσκεται ψηλά στην κατανομή πολυπλοκότητας του συγκεκριμένου έργου. Δεν σημαίνει όμως, ότι έχει ακριβώς το ίδιο απόλυτο επίπεδο πολυπλοκότητας με ένα αρχείο που έχει score 80 σε άλλο έργο.

3.3.1.3 Μετατροπή σε Κλίμακα 0–100

Μετά την κανονικοποίηση, κάθε τιμή μετατρέπεται σε score από 0 έως 100:

$$score = normalized\ value \cdot 100$$

Η κλίμακα 0–100 είναι εύκολη στην ερμηνεία και μπορεί να χρησιμοποιηθεί απευθείας σε διαγράμματα με progress bars, gauges, heatmaps και risk badges.

3.3.1.4 Συνάρτηση Περιορισμού Clamp

Όταν ένας υπολογισμός μπορεί να ξεπεράσει την επιθυμητή κλίμακα, χρησιμοποιείται η συνάρτηση:

$$clamp(x) = \min(\max(x, 0), 100)$$

Η συνάρτηση αυτή εξασφαλίζει ότι όλα οι δείκτες κινδύνου παραμένουν πάντα μέσα στο διάστημα 0–100.

3.3.1.5 KLOC και Πυκνότητες

Για τις συγκρίσεις σε επίπεδο project, οι απόλυτες τιμές μετατρέπονται σε πυκνότητες ανά χίλιες γραμμές κώδικα:

$$metric\ per\ kloc = \frac{metric}{KLOC}, \text{ όπου } KLOC = \frac{socar\ nloc}{1000}$$

Για παράδειγμα:

$$\text{Code Smells per kloc} = \frac{\text{sonar code smells}}{\text{KLOC}}$$

Αυτή η μετατροπή περιορίζει την επίδραση του μεγέθους. Ένα μεγάλο project δεν πρέπει να θεωρείται αυτόματα πιο προβληματικό μόνο επειδή έχει περισσότερες οσμές κώδικα συνολικά ή μεγαλύτερη συνολική πολυπλοκότητα. Οι δείκτες σε επίπεδο έργου είναι έτσι πιο κατάλληλοι για σύγκριση μεταξύ έργων απ' ό,τι οι δείκτες σε επίπεδο αρχείου, αν και η σύγκριση εξακολουθεί να εξαρτάται από τα κατώφλια (thresholds) που χρησιμοποιούνται. Όσα κατώφλια δεν προέρχονται από τεκμηριωμένο εργαλείο ή καθιερωμένο πρότυπο αντιμετωπίζονται ως σχεδιαστικές επιλογές της μεθοδολογίας και ελέγχθηκαν σε διαφορετικά repositories κατά τη βαθμονόμηση.

3.3.1.6 Επίπεδα Κινδύνου

Τα scores 0–100 αντιστοιχίζονται σε τέσσερα επίπεδα κινδύνου:

Low για $0 \leq \text{score} < 40$,
Medium για $40 \leq \text{score} < 60$,
High για $60 \leq \text{score} < 80$ και
Critical για $80 \leq \text{score} \leq 100$.

Οι κατηγορίες αυτές χρησιμεύουν απλώς για πιο εύκολη παρουσίαση στο dashboard και δεν αποτελούν εμπειρικά αποδεδειγμένα κατώφλια για κάθε μετρική. Γι' αυτό, μαζί με την κατηγορία εμφανίζεται και η αντίστοιχη αριθμητική τιμή του δείκτη. Όταν ένας δείκτης δεν είναι διαθέσιμος, δεν του αποδίδεται αντίστοιχο επίπεδο κινδύνου.

3.3.1.7 Απουσία High-TD Πρόβλεψης

Σε επίπεδο αρχείου, όταν το AllinCode δεν επιστρέφει High-TD αποτέλεσμα για ένα αρχείο, το high_td_risk_score δεν ορίζεται σε μηδέν. Αποθηκεύεται ως μη διαθέσιμο και στο dashboard εμφανίζεται η ένδειξη Not flagged. Έτσι ξεχωρίζει η απουσία θετικού High-TD αποτελέσματος από μια πραγματική πρόβλεψη πολύ χαμηλού κινδύνου. Στον υπολογισμό του Refactoring Priority, το High-TD λειτουργεί ως προαιρετικό στοιχείο. Όταν λείπει, το βάρος του αφαιρείται και τα βάρη των υπόλοιπων διαθέσιμων συνιστωσών κανονικοποιούνται ανάλογα. Έτσι η απουσία High-TD αποτελέσματος δεν μειώνει τεχνητά το Refactoring Priority Score. Σε επίπεδο έργου, γίνεται επιπλέον διάκριση ανάμεσα στην επιτυχή κλήση χωρίς αποτελέσματα και στη μη διαθεσιμότητα της πηγής. Αν η διαδρομή High-TD του API εκτελεστεί επιτυχώς αλλά δεν επιστρέψει κανένα αρχείο χαρακτηρισμένο ως High-TD, ο αντίστοιχος δείκτης κινδύνου θεωρείται ίσος με μηδέν. Αντίθετα, αν η διαδρομή δεν είναι διαθέσιμη ή η κλήση αποτύχει, ο δείκτης χαρακτηρίζεται ως μη διαθέσιμος και αφαιρείται από τον υπολογισμό του Overall Project Risk.

3.3.2 Υπολογισμός Επιμέρους Δεικτών σε Επίπεδο Αρχείου

3.3.2.1 Git Activity Score

Το Git Activity Score δείχνει πόσο ενεργό είναι ένα αρχείο μέσα στο ιστορικό του έργου. Υπολογίζεται από τον αριθμό των υποβολών αλλαγών που επηρέασαν το αρχείο, τη συνολική μεταβολή κώδικα και

τον αριθμό των διαφορετικών προγραμματιστών που το τροποποίησαν. Ο δείκτης υπολογίζεται μόνο για αρχεία με διαθέσιμα δεδομένα Git:

$$\text{Git Activity Score} = (0.55 \cdot \text{normalized commit count} + 0.35 \cdot \text{normalized churn} + 0.10 \cdot \text{normalized authors count}) \cdot 100$$

Το ιστορικό αλλαγών δίνει πληροφορίες που δεν φαίνονται από την τρέχουσα μορφή του κώδικα. Ένα αρχείο που αλλάζει συχνά εκτίθεται επανειλημμένα σε νέες τροποποιήσεις και έχει περισσότερες πιθανότητες να δεχτεί μια λανθασμένη αλλαγή. Σε τρεις εκδόσεις του Eclipse, τα μοντέλα που βασίστηκαν σε μετρικές ιστορικής εξέλιξης ταξινομήσαν τα προβληματικά αρχεία πιο αποτελεσματικά από τα μοντέλα που χρησιμοποίησαν μόνο στατικές μετρικές κώδικα [17]. Παρόμοια, μια ανάλυση 85 εκδόσεων από 12 έργων ανοικτού κώδικα έδειξε ότι οι μετρικές ιστορικής εξέλιξης ήταν γενικά πιο χρήσιμες για την πρόβλεψη σφαλμάτων, ενώ οι μετρικές κώδικα άλλαζαν λίγο από έκδοση σε έκδοση και οδηγούσαν συχνά στα ίδια αρχεία [28].

Το `commit_count` παίρνει το μεγαλύτερο βάρος (0.55), γιατί δείχνει πιο άμεσα πόσο συχνά ένα αρχείο βρίσκεται στο επίκεντρο της ανάπτυξης ή της συντήρησης. Η συχνότητα αλλαγών είναι βασικό στοιχείο και στην ανάλυση hotspots. Τα hotspots εντοπίζουν τα σημεία του συστήματος όπου συγκεντρώνεται έντονη δραστηριότητα και η τεχνική τους υγεία χρησιμοποιείται για να κριθεί αν χρειάζονται προτεραιότητα [19]. Η τιμή 0.55 δεν προέρχεται αυτούσια από κάποια μελέτη, αλλά επιλέχθηκε ώστε η συχνότητα αλλαγών να είναι ο κύριος παράγοντας, χωρίς όμως να ακυρώνεται η επίδραση της μεταβολής κώδικα. Αντίστοιχα, το `churn` παίρνει βάρος 0.35 και εκφράζει την έκταση των αλλαγών. Δύο αρχεία μπορεί να έχουν τον ίδιο αριθμό υποβολών αλλαγών, αλλά στο ένα να αλλάζουν λίγες γραμμές και στο άλλο να ξαναγράφονται ολόκληρα κομμάτια κώδικα. Σε σχετική εμπειρική μελέτη, οι απόλυτες τιμές μεταβολής κώδικα από μόνες τους δεν ήταν ιδιαίτερα αποτελεσματικές. Αντίθετα, οι σχετικές μετρικές, που συνέδεαν τις αλλαγές με το μέγεθος του στοιχείου λογισμικού ή με το χρονικό διάστημα μέσα στο οποίο πραγματοποιήθηκαν, προέβλεψαν καλύτερα την πυκνότητα σφαλμάτων [18]. Τέλος, το `authors_count` παίρνει το μικρότερο βάρος (0.10). Δείχνει πόσο διασκορπισμένη είναι η ευθύνη για ένα αρχείο και μπορεί να συνδέεται με μεγαλύτερη ανάγκη συντονισμού. Ο μεγάλος αριθμός διαφορετικών προγραμματιστών δεν σημαίνει απαραίτητα χαμηλή ποιότητα. Μπορεί απλώς να δείχνουν φυσιολογική συνεργασία πάνω σε ένα κεντρικό τμήμα του συστήματος. Επιπλέον, ορισμένες μετρικές ιδιοκτησίας του κώδικα και εμπειρίας παρουσιάζουν μεγαλύτερη αδράνεια σε σχέση με πιο άμεσες μετρικές αλλαγών, όπως οι γραμμές που προστέθηκαν ή διαγράφηκαν [28]. Γι' αυτό χρησιμοποιείται ως συμπληρωματικός παράγοντας στον υπολογισμό του Git Activity Score.

3.3.2.2 Complexity Score

Το Complexity Score υπολογίζεται μόνο για αρχεία με διαθέσιμα δεδομένα SonarQube και συνδυάζει δύο μετρικές:

$$\text{Complexity Score} = (0.60 \cdot \text{normalized sonar complexity} + 0.40 \cdot \text{normalized sonar cognitive complexity}) \cdot 100$$

Όπως αναφέρθηκε στο Κεφάλαιο 2, η κυκλωματική και η γνωστική πολυπλοκότητα αποτυπώνουν διαφορετικές διαστάσεις της δυσκολίας του κώδικα. Ο υπολογισμός της κυκλωματικής πολυπλοκότητας ξεκινά από τιμή 1 για κάθε συνάρτηση και αυξάνεται με κάθε νέο σημείο διακλάδωσης [29]. Η γνωστική πολυπλοκότητα, αντίθετα, δεν μετρά μόνο τις διακλαδώσεις. Αυξάνεται περισσότερο όταν υπάρχουν εμφωλευμένες δομές ή διακοπές της γραμμικής ροής, χωρίς όμως να επιβαρύνει συντακτικές συντομεύσεις που κάνουν τον κώδικα πιο συμπαγή. Τέτοιες συντομεύσεις δεν δυσκολεύουν την ανάγνωση, οπότε δεν αντιμετωπίζονται ως πρόσθετη πολυπλοκότητα [30].

Η κυκλωματική πολυπλοκότητα παίρνει το μεγαλύτερο βάρος (0.60), καθώς είναι πιο καθιερωμένη και έχει άμεση ερμηνεία σε σχέση με τις διαδρομές ελέγχου και την προσπάθεια που απαιτείται για τις δοκιμές. Ένα πολύπλοκο αρχείο μπορεί να μην έχει σφάλματα, αλλά οι αλλαγές σε αυτό απαιτούν περισσότερη προσοχή. Η γνωστική πολυπλοκότητα παίρνει βάρος 0.40. Παρότι η λογική της είναι χρήσιμη, δεν έχει αποδειχθεί ότι προβλέπει σαφώς καλύτερα την ανθρώπινη κατανόηση σε σχέση με τις παλαιότερες μετρικές. Σε εμπειρική αξιολόγηση παρουσίασε παρόμοια σχέση με την ευκολία κατανόησης του κώδικα με εκείνη της κυκλωματικής πολυπλοκότητας, των λογικών γραμμών κώδικα (LLOC) και άλλων στατικών μετρικών. Επιπλέον, μοντέλα που τη συνδυάζαν με άλλες μετρικές παρουσίασαν σχεδόν ίδια ακρίβεια με μοντέλα που δεν τη χρησιμοποιούσαν [31].

3.3.2.3 Hotspot Score

Το Hotspot Score υπολογίζεται μόνο για αρχεία που διαθέτουν ταυτόχρονα δεδομένα Git και SonarQube:

$$\text{Hotspot Score} = \sqrt{\text{Git Activity Score} \cdot \text{Complexity Score}}$$

Η γεωμετρική μορφή του τύπου διατηρεί την κλίμακα 0–100 και απαιτεί να είναι διαθέσιμες και οι δύο διαστάσεις. Αν κάποιο στοιχείο είναι πραγματικά μηδενικό, το hotspot_score γίνεται μηδέν. Από την άλλη, αν κάποιο στοιχείο δεν είναι διαθέσιμο, το hotspot_score παραμένει N/A. Η επιλογή αυτή αποτρέπει το ενδεχόμενο μια πολύ υψηλή τιμή activity να αντισταθμίσει σχεδόν πλήρως μια πολύ χαμηλή τιμή complexity, όπως θα μπορούσε να συμβεί με μια απλή προσθετική μορφή.

3.3.2.4 Structural Risk Score

Το Structural Risk Score υπολογίζεται μόνο για αρχεία με διαθέσιμες δομικές μετρικές AllinCode:

$$\text{Structural Risk Score} = (0.25 \cdot \text{normalized wmc} + 0.25 \cdot \text{normalized rfc} + 0.20 \cdot \text{normalized lcom} + 0.15 \cdot \text{normalized mpc} + 0.15 \cdot \text{normalized cbo}) \cdot 100$$

Οι μετρικές WMC, RFC, CBO και LCOM προέρχονται από τη γνωστή οικογένεια μετρικών CK (CK metrics suite). Η WMC (Weighted Methods per Class) περιγράφει την πολυπλοκότητα ή τον αριθμό των μεθόδων μιας κλάσης και η RFC (Response For a Class) το σύνολο των μεθόδων που μπορεί να εκτελεστούν ως απόκριση σε μια κλήση. Η CBO (Coupling Between Object classes) περιγράφει αντίστοιχα τη σύζευξη με άλλες κλάσεις, ενώ η LCOM (Lack of Cohesion in Methods) την έλλειψη συνοχής μεταξύ των μεθόδων [32]. Οι μετρικές αυτές έχουν χρησιμοποιηθεί για τον εντοπισμό κλάσεων με αυξημένη πιθανότητα εμφάνισης σφαλμάτων, αλλά η σχετική ισχύς κάθε μετρικής δεν παραμένει ίδια σε όλα τα εμπειρικά σύνολα δεδομένων [33].

Η WMC παίρνει βάρος 0.25. Όσο περισσότερη συμπεριφορά συγκεντρώνεται σε μία κλάση, τόσο περισσότερος κώδικας πρέπει να γίνει κατανοητός, να ελεγχθεί και να συντηρηθεί. Παράλληλα, η WMC έχει παρουσιάσει σταθερά θετική σχέση με την πιθανότητα εμφάνισης σφαλμάτων σε πολλές εμπειρικές μελέτες. Η RFC παίρνει επίσης 0.25, καθώς ένα μεγαλύτερο σύνολο απόκρισης σημαίνει περισσότερες πιθανές αλληλεπιδράσεις που πρέπει να ληφθούν υπόψη κατά την αλλαγή ή τις δοκιμές. Έχει κι αυτή εμφανίσει αρκετά συνεπή θετική σχέση με την πιθανότητα εμφάνισης σφαλμάτων [27]. Η LCOM παίρνει βάρος 0.20, επειδή η υψηλή έλλειψη συνοχής μπορεί να δείχνει ότι μια κλάση συγκεντρώνει λειτουργίες που δεν σχετίζονται επαρκώς μεταξύ τους και ενδέχεται να χρειάζεται καλύτερο διαχωρισμό ευθυνών. Τα εμπειρικά αποτελέσματα όμως σχετικά με τη συγκεκριμένη μετρική δεν είναι απόλυτα συνεπή. Σε ορισμένες μελέτες εμφανίζεται ως σημαντικός παράγοντας, ενώ σε άλλες η

συμβολή της είναι πιο περιορισμένη [33]. Για τον λόγο αυτό διατηρεί σημαντική συμμετοχή στον δείκτη, αλλά με ελαφρώς μικρότερο βάρος. Τέλος, η MPC και η CBO παίρνουν βάρος 0.15 η καθεμία, καθώς και οι δύο αποτυπώνουν πλευρές της σύζευξης και των αλληλεπιδράσεων μεταξύ κλάσεων. Η απόδοση πολύ υψηλού βάρους και στις δύο θα μπορούσε να οδηγήσει σε υπερβολική εκπροσώπηση της διάστασης της σύζευξης στον συνολικό δείκτη. Το χαμηλότερο βάρος της CBO δεν υποδηλώνει μικρότερη σημασία της μετρικής, καθώς έχει παρουσιάσει ισχυρή ικανότητα πρόβλεψης όταν εξετάζεται ανεξάρτητα [34]. Η συγκεκριμένη επιλογή οφείλεται κυρίως στην επικάλυψη της πληροφορίας που παρέχει με την MPC.

3.3.2.5 Quality Risk Score

Το Quality Risk Score υπολογίζεται μόνο για αρχεία με διαθέσιμα δεδομένα SonarQube και συνδυάζει οσμές κώδικα και σφάλματα:

$$Quality\ Risk\ Score = (0.60 \cdot normalized\ code\ smells + 0.40 \cdot normalized\ bugs) \cdot 100$$

Τα δύο αυτά στοιχεία αξιολογούν διαφορετικές πλευρές της ποιότητας του έργου λογισμικού. Τα σφάλματα συνδέονται κυρίως με την αξιοπιστία, ενώ οι οσμές κώδικα με τη συντηρησιμότητα [29]. Οι οσμές κώδικα παίρνουν το μεγαλύτερο βάρος 0.60, καθώς το dashboard εστιάζει στο τεχνικό χρέος, τη συντηρησιμότητα και τον ορισμό προτεραιοτήτων για αναδόμηση. Ωστόσο, οι ορισμοί, τα κατώφλια και τα αποτελέσματα των εργαλείων ανίχνευσης δεν είναι πάντα συνεπή μεταξύ τους. Επιπλέον, η αναδόμηση δεν βελτιώνει απαραίτητα κάθε χαρακτηριστικό ποιότητας [35]. Σε μεγάλη εμπειρική μελέτη με 395 εκδόσεις από 30 έργα ανοικτού κώδικα, οι κλάσεις με οσμές είχαν γενικά μεγαλύτερη πιθανότητα αλλαγών και εμφάνισης σφαλμάτων από τις κλάσεις χωρίς αυτά [36]. Αυτό βέβαια δεν σημαίνει ότι κάθε οσμή είναι απόδειξη σοβαρού προβλήματος. Η επίδραση διαφέρει ανά τύπο οσμής και μπορεί να επηρεάζεται από άλλους παράγοντες, όπως το μέγεθος ή ο ρόλος της κλάσης. Τα σφάλματα παίρνουν βάρος 0.40, καθώς αποτελούν πιο άμεση ένδειξη κινδύνου για την αξιοπιστία. Δεν λαμβάνουν όμως το μεγαλύτερο βάρος, επειδή ο δείκτης στοχεύει κυρίως στον εντοπισμό αρχείων που έχουν μεγαλύτερη ανάγκη για αναδόμηση και αυξημένο τεχνικό χρέος.

3.3.2.6 High-TD Risk Score

Το High-TD Risk Score προκύπτει απευθείας από την πιθανότητα που επιστρέφει το AllinCode. Η πιθανότητα εκφράζεται ως δεκαδική τιμή στο διάστημα από 0 έως 1 και μετατρέπεται στην κοινή κλίμακα 0–100:

$$HighTD\ Risk\ Score = HighTD\ probability \cdot 100$$

Για παράδειγμα, πιθανότητα High-TD ίση με 0,84 αντιστοιχεί σε `high_td_risk_score = 84`. Το αποτέλεσμα αποτελεί έξοδο του προγνωστικού μοντέλου του AllinCode και δεν προκύπτει από κανονικοποίηση σε σχέση με τα υπόλοιπα αρχεία του έργου. Ο δείκτης δεν αποτελεί σταθμισμένο συνδυασμό ακατέργαστων μετρικών, αλλά χρησιμοποιείται ως συμπληρωματικό προγνωστικό σήμα. Όταν η διαδρομή High-TD του API δεν επιστρέφει θετικό αποτέλεσμα για ένα αρχείο, το `high_td_risk_score` παραμένει μη διαθέσιμο και στο dashboard εμφανίζεται η ένδειξη Not flagged.

3.3.2.7 Refactoring Priority Score

Το Refactoring Priority Score είναι ο βασικός σύνθετος δείκτης και υποδεικνύει ποιο αρχείο πρέπει να εξεταστεί πρώτο για πιθανή αναδόμηση. Υπολογίζεται αποκλειστικά για τον κοινό πληθυσμό C, ώστε κάθε υποψήφιο αρχείο να διαθέτει δεδομένα Git, SonarQube και AllinCode.

Όταν υπάρχει πρόβλεψη High-TD, ο τύπος είναι:

$$\text{Refactoring Priority Score} = 0.24 \cdot \text{Git Activity Score} + 0.16 \cdot \text{Complexity Score} + 0.30 \cdot \text{Structural Risk Score} + 0.20 \cdot \text{Quality Risk Score} + 0.10 \cdot \text{HighTD Risk Score}$$

Όταν δεν υπάρχει πρόβλεψη High-TD, το προαιρετικό βάρος 0.10 αφαιρείται και το άθροισμα των υποχρεωτικών στοιχείων διαιρείται με το 0.90:

$$\text{Refactoring Priority Score} = \frac{(0.24 \cdot \text{Git Activity Score} + \dots + 0.20 \cdot \text{Quality Risk Score})}{0.9}$$

Η απόφαση σχετικά με το ποια στοιχεία του τεχνικού χρέους πρέπει να αντιμετωπιστούν πρώτα βασίζεται σε πολλαπλά κριτήρια. Οι υπάρχουσες προσεγγίσεις χρησιμοποιούν διαφορετικούς συνδυασμούς παραγόντων, όπως το κεφάλαιο, ο τόκος, το κόστος, η επίδραση και η συντηρησιμότητα. Ωστόσο, δεν υπάρχει συναίνεση για ένα μοναδικό σύνολο μετρικών ή για τον τρόπο μέτρησής τους [26]. Γι' αυτό ο δείκτης παρουσιάζεται εδώ ως μοντέλο κατάταξης για το συγκεκριμένο πλαίσιο, όχι ως καθολική μέτρηση.

Τα βάρη 0.24 και 0.16 προέκυψαν από την αναδιανομή της αρχικής συνεισφοράς του Hotspot. Έτσι, το Git Activity και το Complexity συμμετέχουν άμεσα στον τελικό τύπο, χωρίς να εισάγεται ολόκληρο το hotspot score ως ξεχωριστό, εμφωλευμένο στοιχείο. Η χρήση ανεξάρτητων στοιχείων κάνει το τελικό αποτέλεσμα πιο διαφανές και περιορίζει την κρυφή επανάληψη της πολυπλοκότητας. Το Structural Risk Score παίρνει βάρος 0.30, γιατί η αναδόμηση αφορά κυρίως τον τρόπο οργάνωσης και αλληλεπίδρασης του κώδικα. Η συνοχή, η σύζευξη και η πολυπλοκότητα είναι από τις εσωτερικές ιδιότητες που εξετάζονται συχνότερα κατά τη μελέτη της επίδρασης της αναδόμησης στην ποιότητα έργων λογισμικού [35]. Το Quality Risk Score παίρνει βάρος 0.20, καθώς οι οσμές κώδικα και τα σφάλματα είναι άμεσες ενδείξεις πιθανών προβλημάτων, χωρίς όμως να βαραίνουν όσο οι δομικές μετρικές. Εμπειρικά ευρήματα δείχνουν ότι το μέγεθος των αρχείων και ο αριθμός των αλλαγών που δέχονται, εξηγούν το μεγαλύτερο μέρος της προσπάθειας συντήρησης. Αντίθετα, η ανεξάρτητη επίδραση των οσμών κώδικα είναι σχετικά περιορισμένη [37]. Τέλος, το High-TD Risk Score παίρνει βάρος 0.10 και λειτουργεί σαν δεύτερη γνώμη από το προγνωστικό μοντέλο. Επειδή είναι ήδη επεξεργασμένη έξοδος και όχι άμεσα παρατηρήσιμη ακατέργαστη μετρική, ενισχύει την κατάταξη όταν συμφωνεί με τις υπόλοιπες ενδείξεις, χωρίς να μπορεί να την καθορίσει μόνο του. Όταν δεν είναι διαθέσιμο, η κανονικοποίηση των βαρών εξασφαλίζει ότι δεν επιβάλλεται καμία τεχνητή ποινή στο αρχείο.

3.3.2.8 Μη Αντισταθμιστικός Κανόνας Κινδύνου

Το Refactoring Priority Score είναι ένα σταθμισμένο άθροισμα, πράγμα που σημαίνει ότι μια πολύ υψηλή τιμή σε μία μετρική μπορεί να κρυφτεί πίσω από χαμηλές τιμές στις υπόλοιπες. Για να μην παραβλεφθεί κάποιος υψηλός τεχνικός κίνδυνος, εφαρμόζεται επιπλέον ένας μη αντισταθμιστικός κανόνας:

$$\text{Final Risk Level} = \max(\text{risk level}(\text{Refactoring Priority Score}), \text{risk level}(\text{Actual Risk max}))$$

Όπου:

$$Actual Risk max = \max(Complexity Score, Structural Risk Score, Quality Risk Score, HighTD Risk Score) , \text{ με highTD risk score προαιρετικό}$$

Ο κανόνας αυτός συμπληρώνει το σταθμισμένο μοντέλο. Ο συνδυασμός αντισταθμιστικών και μη αντισταθμιστικών τεχνικών είναι χρήσιμος, όταν δεν θέλουμε μια πολύ καλή τιμή σε ένα κριτήριο να κρύψει ένα ακραίο πρόβλημα σε άλλο [24]. Το Git Activity Score δεν συμμετέχει σε αυτόν τον κανόνα, γιατί υψηλή δραστηριότητα από μόνη της δεν αποδεικνύει χαμηλή ποιότητα. Αντίθετα, μια ακραία τιμή σε έναν επιμέρους δείκτη κινδύνου μπορεί να οδηγήσει σε μια κρίσιμη ένδειξη, ακόμη κι όταν η συνολική σταθμισμένη προτεραιότητα είναι χαμηλότερη. Τα μη διαθέσιμα προαιρετικά στοιχεία αγνοούνται και δεν μετατρέπονται σε μηδενικές τιμές. Οι δύο τελικές ενδείξεις εξυπηρετούν διαφορετικό σκοπό: το Refactoring Priority Level δείχνει τη σειρά προτεραιότητας, ενώ το Final Risk Level λειτουργεί σαν προειδοποίηση ακραίου κινδύνου.

3.3.3 Υπολογισμός Επιμέρους Δεικτών σε Επίπεδο Έργου

Σε επίπεδο έργου, οι μετρικές του SonarQube βασίζονται σε πυκνότητες ανά KLOC και σε σταθερά κατώφλια. Έτσι είναι καταλληλότερες για σύγκριση μεταξύ διαφορετικών έργων. Αντίθετα, οι δείκτες κινδύνου έργου που βασίζονται σε αρχεία (Hotspot, Structural και High-TD Project Risk), υπολογίζονται με βάση τον κοινό πληθυσμό C, ώστε να αναφέρονται στο ίδιο σύνολο αρχείων.

3.3.3.1 Technical Debt Project Risk

Το SonarQube Technical Debt Ratio εκφράζει τον λόγο μεταξύ του εκτιμώμενου κόστους αποκατάστασης των προβλημάτων συντηρησιμότητας και του εκτιμώμενου κόστους ανάπτυξης του κώδικα. Η προεπιλεγμένη κλίμακα χρησιμοποιεί τα όρια 5%, 10%, 20% και 50% για τη μετάβαση μεταξύ των αξιολογήσεων A-E. Τα όρια αυτά, όπως και το εκτιμώμενο κόστος ανάπτυξης ανά γραμμή κώδικα, μπορούν να τροποποιηθούν από τον διαχειριστή του SonarQube [29].

Στην παρούσα μεθοδολογία, τα όρια της κλίμακας A–E αντιστοιχίζονται σε διαδοχικές ζώνες της κλίμακας κινδύνου 0–100. Αν r είναι το Technical Debt Ratio εκφρασμένο σε ποσοστιαίες μονάδες, ο δείκτης υπολογίζεται ως εξής:

$$\text{Για } 0 \leq r < 5:$$

$$Technical Debt Project Risk = 4 * r$$

$$\text{Για } 5 \leq r < 10:$$

$$Technical Debt Project Risk = 20 + 4 * (r - 5)$$

$$\text{Για } 10 \leq r < 20:$$

$$Technical Debt Project Risk = 40 + 2 * (r - 10)$$

$$\text{Για } 20 \leq r < 50:$$

$$Technical Debt Project Risk = 60 + (2 / 3) * (r - 20)$$

$$\text{Για } 50 \leq r < 100:$$

$$Technical Debt Project Risk = 80 + 0,4 * (r - 50)$$

$$\text{Για } r \geq 100:$$

$$Technical Debt Project Risk = 100$$

Η αντιστοίχιση είναι συνεχής και διατηρεί τη διαφορά μεταξύ των βαθμίδων συντηρησιμότητας. Έτσι, οι τιμές που βρίσκονται στις κατηγορίες B, C, D και E δεν καταλήγουν όλες αυτόματα στο μέγιστο risk. Για παράδειγμα, Technical Debt Ratio 0,9% αντιστοιχεί σε δείκτη κινδύνου 3,6. Αντίστοιχα, τιμές 5%,

10%, 20% και 50% αντιστοιχούν σε δείκτες κινδύνου 20, 40, 60 και 80. Η κατηγορία E του SonarQube δεν διαθέτει ανώτατο όριο. Για τις ανάγκες της κοινής κλίμακας 0–100, η παρούσα μέθοδος ορίζει ως σχεδιαστικό σημείο κορεσμού το Technical Debt Ratio 100%. Η επιλογή αυτή αποτελεί παράμετρο του προτεινόμενου μοντέλου και όχι επίσημο όριο του SonarQube.

3.3.3.2 Complexity Project Risk

Αρχικά υπολογίζονται οι πυκνότητες:

$$\text{Complexity per kloc} = \frac{\text{sonar complexity}}{\text{KLOC}}$$

$$\text{Cognitive Complexity per kloc} = \frac{\text{sonar cognitive complexity}}{\text{KLOC}}$$

Στη συνέχεια μετατρέπονται σε risk scores με τα σημεία κορεσμού 180 και 120 ανά KLOC αντίστοιχα:

$$\text{Complexity Density Risk} = \text{clamp} \left(\frac{\text{Complexity per kloc}}{180} \cdot 100 \right)$$

$$\text{Cognitive Complexity Density Risk} = \text{clamp} \left(\frac{\text{Cognitive Complexity per kloc}}{120} \cdot 100 \right)$$

Ο τελικός τύπος είναι:

$$\text{Complexity Project Risk} = 0.60 \cdot \text{Complexity Density Risk} + 0.40 \cdot \text{Cognitive Complexity Density Risk}$$

Η στάθμιση 0.60/0.40 ακολουθεί την ίδια λογική με το Complexity Score στο επίπεδο αρχείου. Η κυκλωματική πολυπλοκότητα παραμένει η βασική μετρική, ενώ η γνωστική πολυπλοκότητα λειτουργεί συμπληρωματικά. Η επιλογή των σημείων κορεσμού 180 και 120 ανά KLOC εξετάζεται στην Ενότητα 5.4.

3.3.3.3 Quality Project Risk

Οι πυκνότητες

$$\text{Code Smells per kloc} = \frac{\text{sonar code smells}}{\text{KLOC}} \text{ και}$$

$$\text{Bugs per kloc} = \frac{\text{sonar bugs}}{\text{KLOC}}$$

μετατρέπονται σε:

$$\text{Code Smells Risk} = \text{clamp} \left(\frac{\text{Code Smells per kloc}}{50} \cdot 100 \right)$$

$$\text{Bugs Risk} = \text{clamp} \left(\frac{\text{Bugs per kloc}}{5} \cdot 100 \right)$$

Ο τελικός τύπος είναι:

$$Quality\ Project\ Risk = 0.60 \cdot Code\ Smells\ Risk + 0.40 \cdot Bugs\ Risk$$

Οι οσμές κώδικα παίρνουν μεγαλύτερο βάρος γιατί ο στόχος παραμένει η αξιολόγηση της συντηρησιμότητας και του τεχνικού χρέους, ενώ τα σφάλματα εκφράζουν περισσότερο την αξιοπιστία. Η επιλογή των σημείων κορεσμού 50 smells/KLOC και 5 bugs/KLOC εξετάζεται στην Ενότητα 5.4.

3.3.3.4 Duplication Project Risk

Ο δείκτης υπολογίζεται ως:

$$Duplication\ Project\ Risk = clamp\left(\frac{sonar\ duplicated\ lines\ density}{15} \cdot 100\right)$$

Η μετρική Duplicated Lines Density του SonarQube είναι το ποσοστό των γραμμών που συμμετέχουν σε διπλότυπο κώδικα [29]. Η τιμή 15% είναι επιλογή της μεθοδολογίας και πρέπει να συγκριθεί με εναλλακτικά κατώφλια κατά τη βαθμονόμηση.

3.3.3.5 Project Risk Count

Για το Hotspot και το Structural Project Risk δεν χρησιμοποιείται ο μέσος όρος όλων των αρχείων. Αντίθετα, η αξιολόγηση εστιάζει στα αρχεία με τις υψηλότερες τιμές, ώστε οι περιπτώσεις αυξημένου κινδύνου να μην εξομαλύνονται από το μεγαλύτερο πλήθος χαμηλότερων τιμών. Ο αριθμός των αρχείων που εξετάζονται υπολογίζεται πάνω στον κοινό πληθυσμό C:

$$Project\ Risk\ Count = \min(N, \max(5, \text{ceil}(0.30 \cdot N))),$$

όπου N είναι το πλήθος των αρχείων του C για τα οποία είναι διαθέσιμος ο δείκτης που εξετάζεται.

Αν για τον εξεταζόμενο δείκτη υπάρχουν έως πέντε διαθέσιμες τιμές στον κοινό πληθυσμό C, χρησιμοποιούνται όλες. Δεν εφαρμόζεται κάποιο ανώτατο όριο αρχείων στον μαθηματικό υπολογισμό, γιατί ένα τέτοιο όριο θα έκανε μεγάλα και μικρά έργα να αξιολογούνται με διαφορετική αναλογία του πληθυσμού τους. Ένα ανώτατο όριο μπορεί να χρησιμοποιείται μόνο στην οπτικοποίηση, για λόγους αναγνωσιμότητας. Το ποσοστό εστίασης 30% επιλέχθηκε έπειτα από τη σύγκριση των εναλλακτικών διαμορφώσεων που παρουσιάζεται στην Ενότητα 5.4.

3.3.3.6 Hotspot Project Risk

Ο δείκτης υπολογίζεται ως ο μέσος όρος των υψηλότερων διαθέσιμων hotspot scores του κοινού πληθυσμού:

$$Hotspot\ Project\ Risk = \frac{1}{k} \sum_{i \in H_k} Hotspot\ Score_i,$$

όπου k προκύπτει από τον παραπάνω τύπο Project risk count.

Ο μέσος όρος όλων των αρχείων θα μπορούσε να κρύψει λίγα σοβαρά hotspots μέσα σε πολλά αρχεία χαμηλού ενδιαφέροντος. Η εστίαση στα ενεργά και τεχνικά δύσκολα σημεία είναι συμβατή με τη λογική προτεραιοποίησης των hotspot όπου προτεραιότητα δίνεται στον κώδικα που αλλάζει συχνά και έχει χαμηλή τεχνική υγεία [19]. Αν ο κοινός πληθυσμός είναι κενός ή δεν υπάρχουν διαθέσιμα hotspot scores, ο δείκτης παραμένει N/A και δεν εισάγεται ως μηδέν στο Global Health Score.

3.3.3.7 Structural Project Risk

Αντίστοιχα:

$$Structural\ Project\ Risk = \frac{1}{k} \sum_{i \in T_k} Structural\ Risk\ Score_i ,$$

όπου k προκύπτει από τον παραπάνω τύπο Project risk count.

Η χρήση των κορυφαίων υποψηφίων εμποδίζει την εξομάλυνση σοβαρών προβλημάτων κλάσεων μέσα σε έναν γενικό μέσο όρο. Ο δείκτης υπολογίζεται μόνο πάνω στον κοινό πληθυσμό έγκυρων υποψηφίων για αναδόμηση και παραμένει μη διαθέσιμος όταν δεν υπάρχουν κατάλληλα δεδομένα για τον υπολογισμό του.

3.3.3.8 High-TD Project Risk

Το High-TD Project Risk παραμένει προαιρετικό και συνδυάζει την έκταση και την ένταση του προβλήματος. Η αναλογία των High-TD αρχείων υπολογίζεται ως προς τον κοινό πληθυσμό C :

$$Source\ HighTD\ Ratio = \frac{highTD\ flagged\ items}{|C|}$$

Ο λόγος αυτός εκφράζει το ποσοστό των αρχείων του κοινού πληθυσμού που έχουν αναγνωριστεί ως αρχεία υψηλού τεχνικού χρέους. Η αντίστοιχη συνιστώσα κινδύνου υπολογίζεται με κατώφλι ίσο με 20%:

$$HighTD\ Ratio\ Risk = clamp\left(\frac{Source\ HighTD\ Ratio}{0.20} \cdot 100\right)$$

Όταν το 20% ή περισσότερο των αρχείων του κοινού πληθυσμού έχει χαρακτηριστεί ως High-TD, η συγκεκριμένη συνιστώσα παίρνει τη μέγιστη τιμή 100. Για μικρότερα ποσοστά, η τιμή αυξάνεται αναλογικά. Εκτός από την έκταση του προβλήματος, λαμβάνεται υπόψη και η ένταση των ισχυρότερων διαθέσιμων High-TD ενδείξεων. Η αποκλειστική χρήση της μέγιστης πιθανότητας θα καθιστούσε το συνολικό αποτέλεσμα σε επίπεδο έργου εξαρτημένο από ένα μόνο αρχείο, το οποίο ενδέχεται να αποτελεί μεμονωμένη ακραία περίπτωση. Για τον περιορισμό της ευαισθησίας αυτής, χρησιμοποιείται ο μέσος όρος του υψηλότερου 20% των αρχείων που έχουν χαρακτηριστεί ως High-TD και ανήκουν στον κοινό πληθυσμό C . Η γενική λογική της προσέγγισης παρουσιάζει ομοιότητες με μεθόδους που βασίζονται στον μέσο όρο των υψηλότερων τιμών (tail-average aggregation), αντί αποκλειστικά στη χειρότερη τιμή [38]. Η επιλογή αυτή είναι επίσης συνεπής με έρευνες στον χώρο των μετρικών λογισμικού, οι οποίες δείχνουν ότι ο τρόπος συγκέντρωσης των επιμέρους τιμών μπορεί να επηρεάσει ουσιαστικά τα τελικά αποτελέσματα και πρέπει να εξετάζεται πειραματικά [39].

Αν M είναι το πλήθος των High-TD αρχείων του κοινού πληθυσμού, το πλήθος των πιθανοτήτων που συμμετέχουν στον υπολογισμό ορίζεται ως:

$$HighTD\ Probability\ Count = \min(M, \max(3, \text{ceil}(0.20 \cdot M)))$$

Με τον εξωτερικό περιορισμό $\min(M, \dots)$, όταν υπάρχουν λιγότερα από τρία High-TD αρχεία χρησιμοποιούνται όλες οι διαθέσιμες πιθανότητες, ενώ διαφορετικά λαμβάνονται υπόψη τουλάχιστον

οι τρεις υψηλότερες. Έτσι, η συνιστώσα δεν καθορίζεται από μία μόνο παρατήρηση, αλλά εξακολουθεί να εστιάζει στις ισχυρότερες High-TD ενδείξεις. Αν $p_1, p_2, \dots, p_k$ είναι οι k υψηλότερες πιθανότητες σε φθίνουσα σειρά, η συνιστώσα υπολογίζεται ως:

$$\text{HighTD Probability Risk} = \left(\frac{p_1 + p_2 + \dots + p_k}{k} \right) \cdot 100, \text{ όπου} \\ k = \text{HighTD Probability Count}$$

Ο συνολικός δείκτης εξακολουθεί να συνδυάζει την έκταση και την ένταση του προβλήματος:

$$\text{HighTD Project Risk} = 0.60 \cdot \text{HighTD Ratio Risk} + 0.40 \cdot \text{HighTD Probability Risk}$$

Το High-TD Ratio Risk λαμβάνει βάρος 0,60, επειδή σε επίπεδο έργου ενδιαφέρει κυρίως πόσο εκτεταμένη είναι η παρουσία High-TD αρχείων. Το High-TD Probability Risk λαμβάνει βάρος 0,40 και λειτουργεί συμπληρωματικά, αποτυπώνοντας τη μέση ένταση της ισχυρότερης ομάδας προβλέψεων. Η μέγιστη πιθανότητα εξακολουθεί να αποθηκεύεται και μπορεί να εμφανίζεται ξεχωριστά στο dashboard ως ένδειξη για το σοβαρότερο μεμονωμένο αρχείο.

Η κάλυψη του μοντέλου High-TD δεν υπολογίζεται ως ξεχωριστός αριθμητικός δείκτης, καθώς η αντίστοιχη διαδρομή API δεν επιστρέφει το πλήρες σύνολο των αρχείων που αξιολογήθηκαν. Αποθηκεύεται μόνο η πληροφορία σχετικά με το αν ήταν διαθέσιμη και αν εκτελέστηκε επιτυχώς. Αν η κλήση ολοκληρωθεί κανονικά αλλά δεν επιστρέφει κανένα High-TD αρχείο, το High-TD Project Risk θεωρείται μηδέν. Αν η διαδρομή δεν είναι διαθέσιμη ή η κλήση αποτύχει, ο δείκτης θεωρείται μη διαθέσιμος και το βάρος του αφαιρείται από τον υπολογισμό του συνολικού project risk. Το κατώφλι κορεσμού του High-TD Ratio ίσο με 20%, το ποσοστό εστίασης των πιθανοτήτων ίσο με 20%, το ελάχιστο πλήθος των τριών αρχείων και η στάθμιση 0,60/0,40 αποτελούν σχεδιαστικές επιλογές της παρούσας μεθόδου.

3.3.4 Ορισμός του Global Health Score

Το Global Health Score είναι ένας συνοπτικός σύνθετος δείκτης της τεχνικής κατάστασης του έργου. Αρχικά, η στάθμιση του Overall Risk Score είναι:

$$\text{Overall Project Risk} = 0.20 \cdot \text{Technical Debt Risk} + 0.20 \cdot \text{Complexity Risk} + \\ 0.15 \cdot \text{Quality Risk} + 0.10 \cdot \text{Duplication Risk} + 0.15 \cdot \text{Hotspot Risk} + \\ 0.10 \cdot \text{Structural Risk} + 0.10 \cdot \text{HighTD Risk}$$

Οι επιμέρους δείκτες παίρνουν τα εξής βάρη:

- **Technical Debt και Complexity:** Το Technical Debt Project Risk και το Complexity Project Risk παίρνουν από 0.20. Το τεχνικό χρέος του SonarQube έχει άμεση ερμηνεία ως εκτιμώμενη προσπάθεια διόρθωσης των προβλημάτων συντήρησης, ενώ το ποσοστό του τεχνικού χρέους (debt ratio) συγκρίνει αυτή την προσπάθεια με το εκτιμώμενο κόστος ανάπτυξης [29]. Η πολυπλοκότητα παίρνει ισότιμο βάρος, γιατί επηρεάζει την κατανόηση, τις δοκιμές και τη δυσκολία των αλλαγών. Η γνωστική πολυπλοκότητα προσθέτει μια διαφορετική οπτική, αλλά δεν έχει αποδειχθεί σταθερά ανώτερο από την κλασική πολυπλοκότητα. Για τον λόγο αυτό, δεν κυριαρχεί στον δείκτη [31].
- **Quality Project Risk:** Το Quality Project Risk παίρνει βάρος 0.15, γιατί οι οσμές κώδικα και τα σφάλματα είναι σημαντικές ενδείξεις για τη συντηρησιμότητα και την αξιοπιστία. Το βάρος είναι λίγο μικρότερο από αυτό της πολυπλοκότητας, καθώς οι μετρήσεις των προβλημάτων

εξαρτώνται από τους ενεργούς κανόνες, τη διαμόρφωση των εργαλείων ανάλυσης και τον τρόπο ανίχνευσης του εργαλείου.

- **Hotspot Project Risk:** Το Hotspot Project Risk λαμβάνει βάρος 0.15, υψηλότερο από εκείνο των υπόλοιπων δεικτών που προκύπτουν από δεδομένα σε επίπεδο αρχείου, επειδή συνδυάζει δύο διαφορετικά στοιχεία: τη συχνότητα αλλαγών και την πολυπλοκότητα του κώδικα. Με αυτόν τον τρόπο δίνεται μεγαλύτερη έμφαση στις περιοχές όπου συγκεντρώνεται η δραστηριότητα ανάπτυξης και ενδέχεται να απαιτείται αυξημένη προσπάθεια συντήρησης [19].
- **Duplication, Structural και High-TD:** Οι δείκτες Duplication, Structural Risk και High-TD Risk λαμβάνουν βάρος από 0.10 ο καθένας. Ο πρώτος αποτυπώνει την έκταση της επανάληψης κώδικα και την πιθανότητα να απαιτούνται οι ίδιες αλλαγές σε πολλά σημεία. Ο δεύτερος εκφράζει αδυναμίες στην αντικειμενοστραφή σχεδίαση, ενώ ο τρίτος προσθέτει την εκτίμηση του προγνωστικού μοντέλου του AllinCode.

Όταν όλοι οι δείκτες είναι διαθέσιμοι, το άθροισμα των βαρών είναι 1.00. Σε περίπτωση που κάποια από τις συνιστώσες δεν είναι διαθέσιμη, γίνεται αναδιανομή των βαρών αναλόγως, όπως φαίνεται στο απόσπασμα κώδικα 4.6. Τέλος, ο τύπος του Global Health Score είναι ο εξής:

$$Global\ Health\ Score = 100 - Overall\ Project\ Risk$$

Όσο μεγαλύτερο είναι το Global Health Score, τόσο χαμηλότερος είναι ο συνολικός δείκτης κινδύνου του έργου.

Κεφάλαιο 4ο: Υλοποίηση Dashboard

Η ανάπτυξη της εφαρμογής επικεντρώθηκε στη συλλογή, ενοποίηση, επεξεργασία και οπτικοποίηση δεδομένων για την ποιότητα λογισμικού και το τεχνικό χρέος. Η διαδικασία ξεκινά με την εισαγωγή ενός αποθετηρίου GitHub και αξιοποιεί δεδομένα από το SonarQube, το AllinCode και το ιστορικό του Git. Έπειτα, παράγει συγκεντρωτικούς δείκτες κινδύνου σε επίπεδο έργου και αρχείου. Τα αποτελέσματα αποθηκεύονται στη βάση δεδομένων και παρουσιάζονται μέσω του dashboard, παρέχοντας μια ολοκληρωμένη εικόνα της κατάστασης του εξεταζόμενου έργου λογισμικού.

4.1 Τεχνολογίες Ανάπτυξης

4.1.1 Τοπικό Περιβάλλον Εκτέλεσης

Η εφαρμογή αναπτύχθηκε και εκτελείται σε τοπικό περιβάλλον Windows. Η επιλογή αυτή επέτρεψε την εγκατάσταση του SonarQube Server, του SonarScanner, της MariaDB και των εργαλείων ανάπτυξης στον ίδιο υπολογιστή, χωρίς να απαιτείται εξωτερικός διακομιστής ή υπηρεσία νέφους. Το back-end λειτουργεί στη θύρα 3000, ενώ το SonarQube είναι διαθέσιμο τοπικά στη θύρα 9000. Η διεπαφή χρήστη, είναι υλοποιημένη σε React και εκτελείται μέσω του Vite. Παράλληλα, η MariaDB εκτελείται μέσω του XAMPP, ενώ το HeidiSQL χρησιμοποιήθηκε κατά την ανάπτυξη για τη δημιουργία, τη διαχείριση και την επιθεώρηση των πινάκων. Το τελικό σχήμα της βάσης εξάγεται στο αρχείο database/schema.sql και ορίζει τους πίνακες projects, analysis_runs, merged_file_metrics και project_trend_history. Περιλαμβάνει επίσης τα ευρετήρια (indexes), τα ξένα κλειδιά (foreign keys) και τις στήλες που χρησιμοποιούνται από το επίπεδο αποθήκευσης δεδομένων. Έτσι η δομή της βάσης μπορεί να αναπαραχθεί σε νέο περιβάλλον χωρίς χειροκίνητη δημιουργία των πινάκων.

4.1.2 Τεχνολογίες του Back-end

Το back-end υλοποιήθηκε σε Node.js, με χρήση του πλαισίου (framework) Express. Το Express διαχειρίζεται τις διαδρομές του API και συνδέει κάθε HTTP αίτημα με την κατάλληλη συνάρτηση επεξεργασίας [40]. Επιπλέον, χρησιμοποιείται ενδιάμεσο λογισμικό (middleware) για την αποδοχή δεδομένων JSON και την ενεργοποίηση του CORS, ώστε το front-end να μπορεί να επικοινωνεί με το back-end από διαφορετική τοπική θύρα. Για την επικοινωνία με το SonarQube χρησιμοποιείται το Axios, ενώ για το AllinCode το ενσωματωμένο Fetch API του Node.js. Η σύνδεση με τη MariaDB πραγματοποιείται μέσω του mysql2/promise, που υποστηρίζει ασύγχρονες λειτουργίες με async και await. Το back-end εκτελεί επίσης εξωτερικές εντολές και αρχεία κώδικα σε Python μέσω της μονάδας (module) child_process. Για την εκτέλεση του analytics_pipeline.py χρησιμοποιείται η execFile, με το αρχείο κώδικα και τα ορίσματα (arguments) να μεταβιβάζονται χωριστά χωρίς ενδιάμεσο περιβάλλον γραμμής εντολών (shell). Αντίθετα, οι εντολές Git, Maven, Gradle και SonarScanner, καθώς και η εκτέλεση του git_metrics.py, πραγματοποιούνται μέσω βοηθητικών συναρτήσεων που χρησιμοποιούν exec και εκτελούνται μέσω του περιβάλλοντος γραμμής εντολών.

4.1.3 Python και Εξόρυξη Δεδομένων από το Git

Η Python χρησιμοποιείται σε δύο στάδια της επεξεργασίας. Το git_metrics.py αναλύει το ιστορικό του αποθετηρίου και εξάγει μετρικές ιστορικής εξέλιξης για κάθε αρχείο. Στη συνέχεια, το analytics_pipeline.py, μαζί με το scoring.py, συνδυάζει τα δεδομένα από το Git, το SonarQube και το AllinCode και υπολογίζει τους τελικούς δείκτες. Για την ανάλυση του ιστορικού Git χρησιμοποιείται το PyDriller, ένα πλαίσιο ανάπτυξης σχεδιασμένο για την εξόρυξη δεδομένων από αποθετήρια [41].

Μέσω της κλάσης Repository και της μεθόδου `traverse_commits()` είναι δυνατή η επεξεργασία των υποβολών αλλαγών. Το PyDriller υποστηρίζει επίσης μετρικές ιστορικής εξέλιξης, όπως τη μεταβολή κώδικα, τον αριθμό υποβολών αλλαγών και τον αριθμό των προγραμματιστών που συμμετείχαν στις αλλαγές [42]. Στην παρούσα υλοποίηση, εξετάζονται οι αλλαγές κάθε υποβολής και από αυτές υπολογίζονται οι μετρικές για κάθε αρχείο. Παράλληλα, εφαρμόζονται οι απαραίτητοι κανόνες φιλτραρίσματος.

4.1.4 Τεχνολογίες του Front-end

Το front-end αναπτύχθηκε με React 18.2.0 και οργανώθηκε ως Vite project. Το Vite παρέχει το τοπικό περιβάλλον ανάπτυξης και δημιουργεί τα στατικά αρχεία που χρειάζονται για την τελική έκδοση της εφαρμογής [43]. Η πλοήγηση μεταξύ της αρχικής οθόνης και της οθόνης του έργου πραγματοποιείται με το React Router 6.30.3. Οι επιμέρους προβολές του dashboard ελέγχονται από την κατάσταση του ProjectDetails. Για τη δημιουργία των διαγραμμάτων χρησιμοποιούνται το Chart.js 4.5.1 και το react-chartjs-2 5.3.1. Το Chart.js εμφανίζει τα γραφήματα σε στοιχεία canvas και υποστηρίζει γραμμικά διαγράμματα, ραβδογράμματα, δακτυλιοειδή διαγράμματα και διαγράμματα φυσαλίδων [44]. Η χρήση της ίδιας βιβλιοθήκης σε όλες τις προβολές εξασφαλίζει κοινή εμφάνιση, αναδυόμενες ετικέτες πληροφοριών (tooltips) και αυτόματη προσαρμογή στο διαθέσιμο μέγεθος της οθόνης.

4.1.5 Οργάνωση του Πηγαίου Κώδικα

Ο πηγαίος κώδικας χωρίζεται σε τρεις κύριους φακέλους: back-end, processing και front-end. Ο διαχωρισμός ακολουθεί τα τρία βασικά επίπεδα ευθύνης της εφαρμογής. Πιο συγκεκριμένα, το back-end χειρίζεται τα αιτήματα και τον κύκλο ζωής της ανάλυσης. Το processing εκτελεί την επεξεργασία και το front-end παρουσιάζει τα αποθηκευμένα αποτελέσματα.

Στον φάκελο back-end βρίσκεται το αρχείο `server.js`, το οποίο περιέχει τις διαδρομές του API και την κεντρική ακολουθία εκτέλεσης. Οι επιμέρους λειτουργίες έχουν μεταφερθεί στον υποφάκελο `services`. Το `sonarService.js` αναλαμβάνει την τοπική ανάλυση SonarQube, ενώ το `gitMetricsService.js` συνδέει τη Node.js με το αρχείο Python που εξάγει τις μετρικές Git. Αντίστοιχα, τα `allincodeAuthService.js` και `allincodeService.js` χειρίζονται την αυθεντικοποίηση και τις αναλύσεις του AllinCode. Το `analyticsPipelineService.js` εκτελεί την τελική ροή επεξεργασίας σε Python. Ως προς την αποθήκευση των δεδομένων, η πρόσβαση στη βάση συγκεντρώνεται στα `db.js`, `saveAnalysis.js`, `getProjects.js` και `getAnalysisResult.js`. Από πλευρά επεξεργασίας, ο φάκελος processing περιλαμβάνει τα τρία ενεργά αρχεία Python: `git_metrics.py`, `analytics_pipeline.py` και `scoring.py`. Τέλος, στον φάκελο front-end/src βρίσκονται οι οθόνες `Home.jsx` και `ProjectDetails.jsx`, καθώς και τα στοιχεία που υλοποιούν τις επιμέρους προβολές και τα γραφήματα. Η οθόνη ProjectDetails λειτουργεί ως κεντρικό στοιχείο συντονισμού (container). Ειδικότερα, φορτώνει την αποθηκευμένη ανάλυση και διατηρεί την ενεργή κατηγορία του πλευρικού μενού. Παράλληλα, δίνει τα κατάλληλα δεδομένα στα OverviewSection, TemporalEvolutionSection, HotspotsSection, QualityStructureSection και AllFileMetricsSection.

Πίνακας 4.1 - Κύρια αρχεία και ευθύνες της εφαρμογής

Αρχείο ή φάκελος	Κύριος ρόλος
backend/server.js	Δήλωση διαδρομών API και κεντρικός συντονισμός της ανάλυσης
backend/services/sonarService.js	Κλωνοποίηση αποθετηρίου, εκτέλεση SonarScanner και ανάκτηση μετρικών SonarQube
backend/services/gitMetricsService.js	Πλήρης κλωνοποίηση του αποθετηρίου Git και εκτέλεση του git_metrics.py
backend/services/allincodeService.js	Δημιουργία έργου, επανάληψη ανάλυσης, έλεγχος κατάστασης και ανάκτηση αποτελεσμάτων High-TD
backend/services/analyticsPipelineService.js	Δημιουργία αρχείου JSON εισόδου και εκτέλεση της ροής επεξεργασίας σε Python
backend/services/saveAnalysis.js	Αποθήκευση ανάλυσης και εφαρμογή της πολιτικής διατήρησης των αναλύσεων
processing/analytics_pipeline.py	Κανονικοποίηση διαδρομών και ενοποίηση δεδομένων από τις διαφορετικές πηγές
processing/scoring.py	Έλεγχος επιλεξιμότητας, κανονικοποίηση με βάση το P95 και υπολογισμός των δεικτών
frontend/src/pages/Home.jsx	Εισαγωγή URL, εμφάνιση λίστας έργων και επανάληψη ανάλυσης
frontend/src/pages/ProjectDetails.jsx	Φόρτωση αποθηκευμένης ανάλυσης και σύνθεση του dashboard
frontend/src/components/dashboard	Υλοποίηση των πέντε αναλυτικών οθονών του dashboard
frontend/src/components/charts	Υλοποίηση των γραφημάτων με τη βιβλιοθήκη Chart.js

4.2 Ανάπτυξη της Μηχανισμού Επεξεργασίας (Analytics Script)

4.2.1 Συλλογή και Προετοιμασία των Δεδομένων Εισόδου

Η διαδικασία ενεργοποιείται με την υποβολή του URL ενός δημόσιου αποθετηρίου GitHub. Το back-end αρχικά ελέγχει το URL και προσδιορίζει τον ιδιοκτήτη και το όνομα του έργου. Στη συνέχεια, ξεκινά τη συλλογή δεδομένων από το SonarQube, το ιστορικό του Git και το AllinCode. Για την ανάλυση του τρέχοντος στιγμιότυπου του κώδικα, το αποθετήριο κλωνοποιείται προσωρινά και εκτελείται τοπικά ο SonarScanner. Μετά την ολοκλήρωση της ανάλυσης, το back-end ανακτά μέσω του Web API του SonarQube τις συνολικές μετρικές του έργου και τις μετρικές ανά αρχείο. Η ανάλυση του ιστορικού Git πραγματοποιείται ξεχωριστά μέσω του git_metrics.py και του PyDriller. Από αυτή τη διαδικασία υπολογίζονται για κάθε αρχείο ο αριθμός των υποβολών αλλαγών, ο αριθμός των προγραμματιστών που συμμετείχαν στις αλλαγές, οι γραμμές που προστέθηκαν και διαγράφηκαν και η συνολική μεταβολή κώδικα. Τέλος, η επικοινωνία με το AllinCode πραγματοποιείται μέσω του διαθέσιμου API. Το back-end ελέγχει αν το έργο είναι ήδη καταχωρισμένο, ζητά νέα ανάλυση και ανακτά τις δομικές μετρικές και τα διαθέσιμα αποτελέσματα High-TD. Επειδή το AllinCode αποτελεί εξωτερική υπηρεσία, η μη διαθεσιμότητά του δεν διακόπτει απαραίτητα ολόκληρη τη διαδικασία. Η

επεξεργασία μπορεί να συνεχιστεί με τα διαθέσιμα δεδομένα του SonarQube και του Git, ενώ η απουσία δεδομένων από το AllinCode καταγράφεται ρητά.

Μετά την ολοκλήρωση της συλλογής, τα αποτελέσματα των τριών πηγών οργανώνονται σε ένα ενιαίο αντικείμενο JSON, το οποίο αποτελεί την είσοδο για το αρχείο επεξεργασίας σε Python. Περιλαμβάνει τις μετρικές σε επίπεδο έργου και αρχείου, τα στοιχεία του αποθετηρίου και τις ενδείξεις διαθεσιμότητας κάθε πηγής. Έτσι, το back-end αναλαμβάνει τον συντονισμό της διαδικασίας συλλογής, ενώ η ενοποίηση των δεδομένων και ο υπολογισμός των δεικτών πραγματοποιούνται στο υποσύστημα επεξεργασίας.

```
//...
// SonarQube
console.log("Running SonarQube analysis...");
const sonarResult = await main(url);

// Git metrics
console.log("Running Git metrics analysis...");
const gitResult = await analyzeGitMetrics(url, {
  onlySource: true,
  excludeTests: false
});

// AllinCode
console.log("Preparing a new AllinCode analysis...");
let allincodeProject = null;
let allincodeFiles = [];
let allincodeHighRisk = [];
let allincodeHighTD = [];
let allincodeHighTDAvailable = false;
let allincodeLifecycle = {
  available: false,
  project: null,
  highRisk: null,
  error: null
};

try {

  const projectData = await ensureAllinCodeProjectData({
```

```

        owner,
        name,
        repositoryUrl: url
    });

    allincodeProject = projectData.project;
    allincodeFiles = projectData.files;
    allincodeLifecycle.available = true;
    allincodeLifecycle.project = projectData.lifecycle;

    const highRiskData = await ensureAllinCodeHighRiskData({
        owner,
        name
    });

    allincodeHighRisk = highRiskData.all;
    allincodeHighTD = highRiskData.highTD;
    allincodeHighTDAvailable = highRiskData.highTDAvailable;
    allincodeLifecycle.highRisk = highRiskData.lifecycle;
} catch (error) {
    allincodeLifecycle.error = error.message;
    console.warn(
        "AllinCode project/reanalysis flow unavailable:",
        error.message
    );
}

// Input Python
const analyticsInput = {
    repository: url,
    owner,
    name,

    sonar_project: sonarResult,

    sonar_files: extractArray(sonarResult?.sonar_files),

    git_files: extractArray(

```

```

gitResult?.gitMetrics?.files ||

gitResult?.files
),

allincode_project: allincodeProject,
allincode_files: extractArray(allincodeFiles),

allincode_high_risk: extractArray(allincodeHighRisk),
allincode_high_td: extractArray(allincodeHighTD),
allincode_high_td_available: allincodeHighTDAvailable
};
// ...

// Python merge
const analyticsOutput = await runAnalyticsPipeline(analyticsInput);

const savedToDatabase = await saveAnalysisResult({
  repoUrl: url,
  result: analyticsOutput.result
});

```

Απόσπασμα κώδικα 4.1: Συλλογή δεδομένων και δημιουργία της κοινής εισόδου του analytics pipeline(server.js).

Το αντικείμενο `analyticsInput` συγκεντρώνει τα στοιχεία του αποθετηρίου, τις μετρικές επιπέδου έργου και αρχείων των τριών πηγών, τα High-TD αποτελέσματα και την ένδειξη διαθεσιμότητας της αντίστοιχης διαδρομής. Η `runAnalyticsPipeline()` αποθηκεύει την είσοδο σε αρχείο JSON με κωδικοποίηση UTF-8, εκτελεί το `analytics_pipeline.py` και διαβάζει το παραγόμενο αποτέλεσμα. Η διατήρηση των αρχείων εισόδου και εξόδου διευκολύνει τον έλεγχο και την επανάληψη της επεξεργασίας Python, χωρίς νέα κλωνοποίηση του αποθετηρίου και νέα επικοινωνία με τα εξωτερικά εργαλεία.

4.2.2 Κανονικοποίηση των Διαδρομών

Το πρώτο πρόβλημα που αντιμετωπίζει το `analytics_pipeline.py` είναι η διαφορετική αναπαράσταση των διαδρομών. Η `normalize_path()` μετατρέπει τα διαχωριστικά διαδρομών των Windows σε / και αφαιρεί πιθανά προθέματα μονάδας δίσκου (drive prefixes). Ακόμη, αφαιρεί αρχικά ./ και περιττές αρχικές καθέτους και συμπύσσει διαδοχικά διαχωριστικά. Κατά την εξαγωγή της διαδρομής από μία εγγραφή εξετάζονται διαφορετικά πιθανά πεδία, ενώ για κλειδιά του SonarQube της μορφής `projectKey:path` χρησιμοποιείται το τμήμα μετά το “:”. Η αντιστοίχιση βασίζεται στην πλήρη κανονικοποιημένη σχετική διαδρομή και όχι μόνο στο όνομα του αρχείου. Η επιλογή αυτή είναι ιδιαίτερα σημαντική σε έργα με πολλαπλές υπομονάδες (multi-module projects), όπου το ίδιο όνομα αρχείου μπορεί να εμφανίζεται σε διαφορετικούς φακέλους. Η χρήση μόνο του βασικού ονόματος θα μπορούσε να οδηγήσει σε λανθασμένη συγχώνευση.

```

# ...
def normalize_path(path):
    if path is None:
        return None

    path = str(path).strip()

    if not path:
        return None

    path = path.replace("\\", "/")

    # Remove Windows drive prefix if it ever appears,
    # e.g. C:/project/src/A.java
    path = re.sub(r"^[A-Za-z]:/", "", path)

    while path.startswith("./"):
        path = path[2:]

    path = path.lstrip("/")
    path = re.sub(r"/+", "/", path)

    return path if path else None

# ... Παραλείπονται βοηθητικές συναρτήσεις ανάκτησης και προετοιμασίας τιμών ...

def get_path_from_item(item):
    if not isinstance(item, dict):
        return None

    path = get_value(
        item,
        [
            "path",
            "class_path",
            "file",
            "file_path",
            "component.path",
            "component.name"
        ]
    )

```

```

)

if path:
    return normalize_path(path)

# Sonar key sometimes looks like
# projectKey:src/main/java/A.java
key = item.get("key")

if isinstance(key, str) and ":" in key:
    possible_path = key.split(":", 1)[1]

    if "/" in possible_path or "\\" in possible_path:
        return normalize_path(possible_path)

fallback = get_value(
    item,
    ["name", "class_name"]
)

return normalize_path(fallback)

# ...

```

Απόσπασμα κώδικα 4.2: Εξαγωγή και κανονικοποίηση των paths από τις διαφορετικές πηγές δεδομένων(analytics_pipeline.py).

4.2.3 Ενοποίηση των Τριών Πηγών

Μετά την κανονικοποίηση δημιουργείται μία βασική εγγραφή για κάθε μοναδική διαδρομή. Οι λίστες δεδομένων Git, SonarQube και AllinCode προσθέτουν τα δικά τους πεδία στην ίδια εγγραφή. Παράλληλα, ενημερώνονται οι ενδείξεις: `has_git_data`, `has_sonar_data`, `has_allincode_data` και `has_high_td_data`.

Ο τρόπος συγχώνευσης λαμβάνει υπόψη τη διαφορετική λεπτομέρεια των πηγών. Το Git και το SonarQube επιστρέφουν συνήθως μία εγγραφή ανά αρχείο. Όταν εμφανίζονται πολλαπλές εγγραφές, διατηρούνται οι μεγαλύτερες διαθέσιμες τιμές ώστε να μη χάνονται δεδομένα από επαναλήψεις της ίδιας πηγής. Το AllinCode από την άλλη, μπορεί να επιστρέφει περισσότερες από μία κλάσεις για το ίδιο αρχείο. Στην περίπτωση αυτή, οι τιμές τεχνικού χρέους αθροίζονται, ενώ για τις δομικές μετρικές διατηρείται η μεγαλύτερη τιμή μεταξύ των κλάσεων. Η επιλογή αυτή αντανακλά το γεγονός ότι το συνολικό χρέος μπορεί να προκύπτει από περισσότερες κλάσεις του ίδιου αρχείου. Αντίθετα, στις μετρικές κινδύνου μεγαλύτερη σημασία έχει η ισχυρότερη διαθέσιμη ένδειξη μέσα στο αρχείο.

```

# ...

def ensure_record(merged, path):
    if path not in merged:
        merged[path] = create_base_record(path)

```

```

    return merged[path]

# ... Παραλείπονται οι συναρτήσεις συγχώνευσης των δεδομένων Git και SonarQube
...

def merge_allincode_files(merged, allincode_files):
    for item in allincode_files:
        path = get_path_from_item(item)

        if not path:
            continue

        record = ensure_record(merged, path)

        record["has_allincode_data"] = True

        # If multiple class records map to the same file,
        # debt values are summed.
        record["allincode_total_debt"] += to_number(
            get_value(item, ["totalDebt", "total_debt"])
        )
        record["allincode_debt"] += to_number(
            get_value(item, ["debt"])
        )
        record["allincode_interest_rate"] = max(
            record["allincode_interest_rate"],
            to_number(
                get_value(
                    item,
                    ["interestRate", "interest_rate"]
                )
            )
        )

    )

# ... Παραλείπεται η ανάκτηση των LOC και complexity μετρικών ...

for metric in [
    "wmc",
    "rfc",

```

```

        "lcom",
        "mpc",
        "cbo",
        "dac",
        "nom"
    ]:
        record[metric] = max(
            record[metric],
            to_number(
                get_value(
                    item,
                    [
                        metric,
                        f"fileMetrics.{metric}",

                        f"file_metrics.{metric}",
                        f"metrics.{metric}"
                    ]
                )
            )
        )
    )
)

```

Απόσπασμα κώδικα 4.3: Δημιουργία κοινής εγγραφής και συγχώνευση των AllinCode μετρικών ανά `path(analytics_pipeline.py)`

Τα High-TD δεδομένα συγχωνεύονται επίσης με βάση τη διαδρομή. Για κάθε αρχείο διατηρούνται η υψηλότερη πιθανότητα, η ένδειξη χαρακτηρισμού ως High-TD και τα διαθέσιμα στοιχεία σε επίπεδο κλάσης.

4.2.4 Έλεγχος Επιλεξιμότητας

Το `scoring.py` δεν υπολογίζει όλους τους δείκτες για κάθε ενοποιημένη εγγραφή. Αρχικά ελέγχει αν η διαδρομή αντιστοιχεί σε υποστηριζόμενο αρχείο πηγαίου κώδικα. Αποκλείονται τα αρχεία δοκιμών, τα αρχεία που βρίσκονται σε φακέλους όπως `target`, `build`, `dist`, `node_modules`, `vendor`, `generated` και `.git`, καθώς και παλιές διαδρομές Git που δεν εμφανίζονται στις τρέχουσες πηγές ανάλυσης. Στη συνέχεια δημιουργούνται ξεχωριστές ενδείξεις σχετικά με τις απαιτήσεις υπολογισμού του κάθε δείκτη. Το Git Activity απαιτεί δεδομένα Git, τα Complexity και Quality Risk δεδομένα SonarQube, ενώ το Structural Risk δεδομένα AllinCode. Για το Hotspot απαιτούνται δεδομένα Git και SonarQube. Ο κοινός πληθυσμός για τους δείκτες σε επίπεδο έργου περιλαμβάνει μόνο αρχεία που υπάρχουν και στις τρεις βασικές πηγές.

```

# ...

def is_base_scoring_item(item):
    """

```

```

Basic eligibility for file-level scoring.
"""
path = item.get("path")

return (
    not is_test_item(item)
    and has_current_analysis_data(item)
    and is_code_file(path)
    and not is_excluded_scoring_path(path)
)

def eligible_for_git_activity(item):
    return (
        is_base_scoring_item(item)
        and safe_bool(item.get("has_git_data"))
    )

def eligible_for_sonar_scores(item):

    return (
        is_base_scoring_item(item)
        and safe_bool(item.get("has_sonar_data"))
    )

def eligible_for_structural_score(item):
    return (
        is_base_scoring_item(item)
        and safe_bool(item.get("has_allincode_data"))
    )

def eligible_for_hotspot_score(item):
    return (
        is_base_scoring_item(item)
        and safe_bool(item.get("has_git_data"))
        and safe_bool(item.get("has_sonar_data"))
    )

```

```

def is_common_scoring_item(item):
    """
    Population used by every calculation that combines
    Git, SonarQube and AllinCode dimensions.
    """
    return (
        is_base_scoring_item(item)
        and safe_bool(item.get("has_git_data"))
        and safe_bool(item.get("has_sonar_data"))
        and safe_bool(item.get("has_allincode_data"))
    )

def is_refactoring_candidate(item):
    """
    Refactoring candidates are exactly the same with the scoring
    population. This prevents missing source data from being
    interpreted as zero.
    """
    return is_common_scoring_item(item)

#...

```

Απόσπασμα κώδικα 4.4: Έλεγχος επιλεξιμότητας και σχηματισμός του κοινού πληθυσμού αρχείων (scoring.py).

Οι ενδείξεις αυτές αποθηκεύονται μαζί με τις μετρικές κάθε αρχείου. Έτσι, το front-end μπορεί να αποκλείει αρχεία που δεν είναι κατάλληλα για αξιολόγηση αναδόμησης, χωρίς να τα αφαιρεί από το σύνολο δεδομένων. Αυτό είναι χρήσιμο για παλιές διαδρομές Git, αρχεία ρυθμίσεων και εγγραφές για τις οποίες υπάρχουν μόνο μερικά δεδομένα. Οι αναλυτικοί ορισμοί των πληθυσμών δεδομένων και οι προϋποθέσεις υπολογισμού κάθε δείκτη παρουσιάζονται στην ενότητα 3.3.

4.2.5 Κανονικοποίηση P95 και Υπολογισμός Δεικτών σε Επίπεδο Αρχείου

Για κάθε ακατέργαστη μετρική που συμμετέχει σε κανονικοποίηση υπολογίζεται η αντίστοιχη τιμή P95. Ως κύρια βάση αναφοράς χρησιμοποιείται ο κοινός πληθυσμός αρχείων C. Αν για κάποια μετρική δεν υπάρχουν θετικές τιμές στον κοινό πληθυσμό, χρησιμοποιείται ως εναλλακτική η τιμή P95 του ευρύτερου επιλέξιμου πληθυσμού της αντίστοιχης πηγής (Git, SonarQube ή AllinCode). Με αυτόν τον τρόπο αποφεύγεται η απώλεια μιας χρήσιμης τιμής αναφοράς όταν η συγκεκριμένη μετρική δεν εμφανίζεται στον κοινό πληθυσμό. Οι κανονικοποιημένες τιμές περιορίζονται στο διάστημα 0 έως 1 και χρησιμοποιούνται στη συνέχεια για τον υπολογισμό των δεικτών σε επίπεδο αρχείου στην κλίμακα 0 έως 100. Όταν ένα αρχείο δεν είναι επιλέξιμο για έναν δείκτη ή δεν διαθέτει την απαιτούμενη πηγή δεδομένων, η αντίστοιχη τιμή παραμένει None αντί να θεωρείται μηδενική. Οι επιμέρους δείκτες στρογγυλοποιούνται πριν από τον υπολογισμό του Refactoring Priority Score. Έτσι, ο τελικός συνδυασμός βασίζεται στις ίδιες τιμές που αποθηκεύονται στη βάση και επιστρέφονται στο front-end. Για κάθε επιλέξιμο αρχείο αποθηκεύονται επίσης τα επιμέρους επίπεδα κινδύνου, το max_component_score και το final_risk_level. Το max_component_score προκύπτει από την

υψηλότερη διαθέσιμη τιμή μεταξύ των δεικτών πολυπλοκότητας, δομικού κινδύνου, ποιότητας και High-TD. Το τελικό επίπεδο προτεραιότητας είναι το αυστηρότερο μεταξύ του επιπέδου που προκύπτει από το Refactoring Priority Score και του επιπέδου της ισχυρότερης επιμέρους ένδειξης. Κατά συνέπεια, μια ιδιαίτερα σοβαρή διάσταση κινδύνου δεν εξομαλύνεται πλήρως από τον σταθμισμένο συνδυασμό.

```
def calculate_refactoring_priority_score(item):
    if not is_common_scoring_item(item):
        return None

    weighted_components = [
        (item.get("git_activity_score"), 0.24),
        (item.get("complexity_score"), 0.16),
        (item.get("structural_risk_score"), 0.30),
        (item.get("quality_risk_score"), 0.20),
    ]

    if item.get("high_td_risk_score") is not None:
        weighted_components.append(
            (item.get("high_td_risk_score"), 0.10)
        )

    available_weight = sum(
        weight
        for value, weight in weighted_components
        if value is not None
    )

    if available_weight <= 0:
        return None

    weighted_sum = sum(
        safe_number(value) * weight
        for value, weight in weighted_components
        if value is not None
    )

    return clamp(
        weighted_sum / available_weight
    )
```

```

def calculate_final_risk_level(item):
    priority_score = item.get(
        "refactoring_priority_score"
    )

    if priority_score is None:
        return None

    priority_level = risk_level(priority_score)

    component_scores = [
        item.get("complexity_score"),
        item.get("structural_risk_score"),
        item.get("quality_risk_score"),
        item.get("high_td_risk_score"),
    ]

    available_scores = [
        safe_number(value)
        for value in component_scores
        if value is not None
    ]

    if not available_scores:
        return priority_level

    component_level = risk_level(
        max(available_scores)
    )

    return max_risk_level(
        priority_level,
        component_level
    )

```

Απόσπασμα κώδικα 4.5: Υπολογισμός του Refactoring Priority Score και εφαρμογή του μη αντισταθμιστικού κανόνα(scoring.py).

4.2.6 Υπολογισμός Δεικτών σε Επίπεδο Έργου

Μετά τον υπολογισμό των δεικτών σε επίπεδο αρχείου σχηματίζεται ο κοινός πληθυσμός που χρησιμοποιείται για τον υπολογισμό των κινδύνων σε επίπεδο έργου και εφαρμόζεται ο κανόνας επιλογής του ανώτερου τμήματός του. Από αυτό προκύπτουν τα Hotspot και Structural Project Risks. Παράλληλα, οι συνολικές μετρικές του SonarQube χρησιμοποιούνται για τα Technical Debt, Complexity, Quality και Duplication Project Risks. Για το High-TD Project Risk, η υλοποίηση χρησιμοποιεί μόνο τα αρχεία που έχουν χαρακτηριστεί ως High-TD και ανήκουν στον κοινό πληθυσμό. Αποθηκεύονται το ποσοστό High-TD αρχείων, η μέγιστη πιθανότητα ως πρόσθετη πληροφορία, ο μέσος όρος του επιλεγμένου ανώτερου τμήματος των πιθανοτήτων και το πλήθος των τιμών που χρησιμοποιήθηκαν. Αν η αντίστοιχη διαδρομή του API είναι διαθέσιμη αλλά δεν επιστρέφει κανένα High-TD αρχείο, ο δείκτης θεωρείται μηδενικός. Αντιθέτως, αν η υπηρεσία δεν είναι διαθέσιμη ή ο κοινός πληθυσμός είναι κενός, η τιμή παραμένει None. Έπειτα, το Overall Project Risk υπολογίζεται μόνο από τις διαθέσιμες συνιστώσες. Το άθροισμα των ενεργών βαρών χρησιμοποιείται ως παρονομαστής, ώστε η απουσία μιας προαιρετικής συνιστώσας να μη οδηγεί σε τεχνητή βελτίωση της κατάστασης. Τέλος, το Global Health Score προκύπτει ως συμπλήρωμα του συνολικού κινδύνου ως προς το 100.

```
weighted_components = [
    (technical_debt_project_risk, 0.20),
    (complexity_project_risk, 0.20),
    (quality_project_risk, 0.15),
    (duplication_project_risk, 0.10),
    (hotspot_project_risk, 0.15),
    (structural_project_risk, 0.10),
    (high_td_project_risk, 0.10),
]

available_components = [
    (value, weight)
    for value, weight in weighted_components
    if value is not None
]

available_weight = sum(
    weight
    for _, weight in available_components
)

if available_weight > 0:
    overall_project_risk = sum(
        safe_number(value) * weight
        for value, weight in available_components
    )
```

```

) / available_weight

global_health_score = (
    100 - overall_project_risk
)
else:
    overall_project_risk = None
    global_health_score = None

```

Απόσπασμα κώδικα 4.6: Υπολογισμός του Overall Project Risk και του Global Health Score(scoring.py).

4.2.7 Μορφή Εξόδου της Ροής Επεξεργασίας

Η έξοδος της ροής επεξεργασίας περιλαμβάνει τρεις κύριες ενότητες: `summary`, `project_metrics` και `merged_file_metrics`. Το `summary` περιέχει πλήθη κάλυψης και βασικές πληροφορίες ενοποίησης. Το `project_metrics` περιλαμβάνει τις συνολικές ακατέργαστες μετρικές, τις πυκνότητες, τους δείκτες κινδύνου σε επίπεδο έργου και τα στοιχεία του υπολογισμού High-TD. Το `merged_file_metrics` περιλαμβάνει για κάθε διαδρομή τα ακατέργαστα δεδομένα, τις ενδείξεις διαθεσιμότητας και επιλεξιμότητας, τους επιμέρους δείκτες και τα επίπεδα κινδύνου. Το αποτέλεσμα γράφεται σε νέο αρχείο JSON και επιστρέφεται στο Node.js. Η ύπαρξη τόσο της εισόδου όσο και της εξόδου διευκολύνει τον έλεγχο των υπολογισμών. Σε περίπτωση προβλήματος στην επεξεργασία, το υποσύστημα Python μπορεί να εκτελεστεί ξανά χωρίς νέα επικοινωνία με τα εξωτερικά εργαλεία.

4.3 Σχεδιασμός δεδομένων και Διαχείριση Ιστορικότητας

Η βάση δεδομένων χρησιμοποιείται για τη μόνιμη αποθήκευση των έργων και των αποτελεσμάτων των αναλύσεων, ώστε να είναι δυνατή η άμεση παρουσίαση τους στο dashboard χωρίς επανάληψη της διαδικασίας συλλογής και υπολογισμού. Υλοποιείται σε MariaDB και εκτελείται τοπικά μέσω του XAMPP. Η σχεδίαση περιλαμβάνει τέσσερις βασικούς πίνακες: `projects`, `analysis_runs`, `merged_file_metrics` και `project_trend_history`.

4.3.1 Σχήμα της Βάσης Δεδομένων

4.3.1.1 Πίνακας Projects

Ο πίνακας `projects` περιλαμβάνει μία εγγραφή για κάθε αποθηκευμένο αποθετήριο, με το URL, τον ιδιοκτήτη, το όνομα, την ημερομηνία δημιουργίας της εγγραφής και την ημερομηνία της τελευταίας επιτυχούς ανάλυσης. Η μοναδικότητα του URL του αποθετηρίου αποτρέπει τη δημιουργία πολλών εγγραφών για το ίδιο έργο. Ένα project όμως μπορεί να συνδέεται με περισσότερα `analysis_runs`, ώστε τα βασικά στοιχεία του αποθετηρίου να αποθηκεύονται μία φορά και κάθε νέα επιτυχής εκτέλεση να καταγράφεται ξεχωριστά.

4.3.1.2 Πίνακας Analysis Runs

Ο πίνακας `analysis_runs` αποθηκεύει το αναλυτικό στιγμιότυπο κάθε επιτυχούς εκτέλεσης: ημερομηνία, status καθώς και τις συνολικές μετρικές που επιστράφηκαν από το SonarQube και το AllinCode. Στον ίδιο πίνακα αποθηκεύονται οι δείκτες επιπέδου αρχείου, το Overall Project Risk και το Global Health Score. Για την ερμηνεία των αποτελεσμάτων διατηρούνται ακόμη:

- τα πλήθη των ενοποιημένων και επιλέξιμων αρχείων
- οι πυκνότητες ανά KLOC
- οι τιμές που χρησιμοποιήθηκαν στους υπολογισμούς σε επίπεδο έργου
- το ποσοστό και ο αριθμός των αρχείων που έχουν χαρακτηριστεί ως High-TD
- η μέγιστη πιθανότητα
- ο μέσος όρος της επιλεγμένης ανώτερης ομάδας
- το πλήθος των πιθανοτήτων που συμμετείχαν στον υπολογισμό

Κάθε εκτέλεση ανάλυσης αποτελεί πλήρες στιγμιότυπο της κατάστασης του έργου σε συγκεκριμένη χρονική στιγμή. Για τον περιορισμό του όγκου της βάσης διατηρούνται μόνο η τελευταία και η αμέσως προηγούμενη πλήρης εκτέλεση ανάλυσης για κάθε έργο.

4.3.1.3 Πίνακας Merged File Metrics

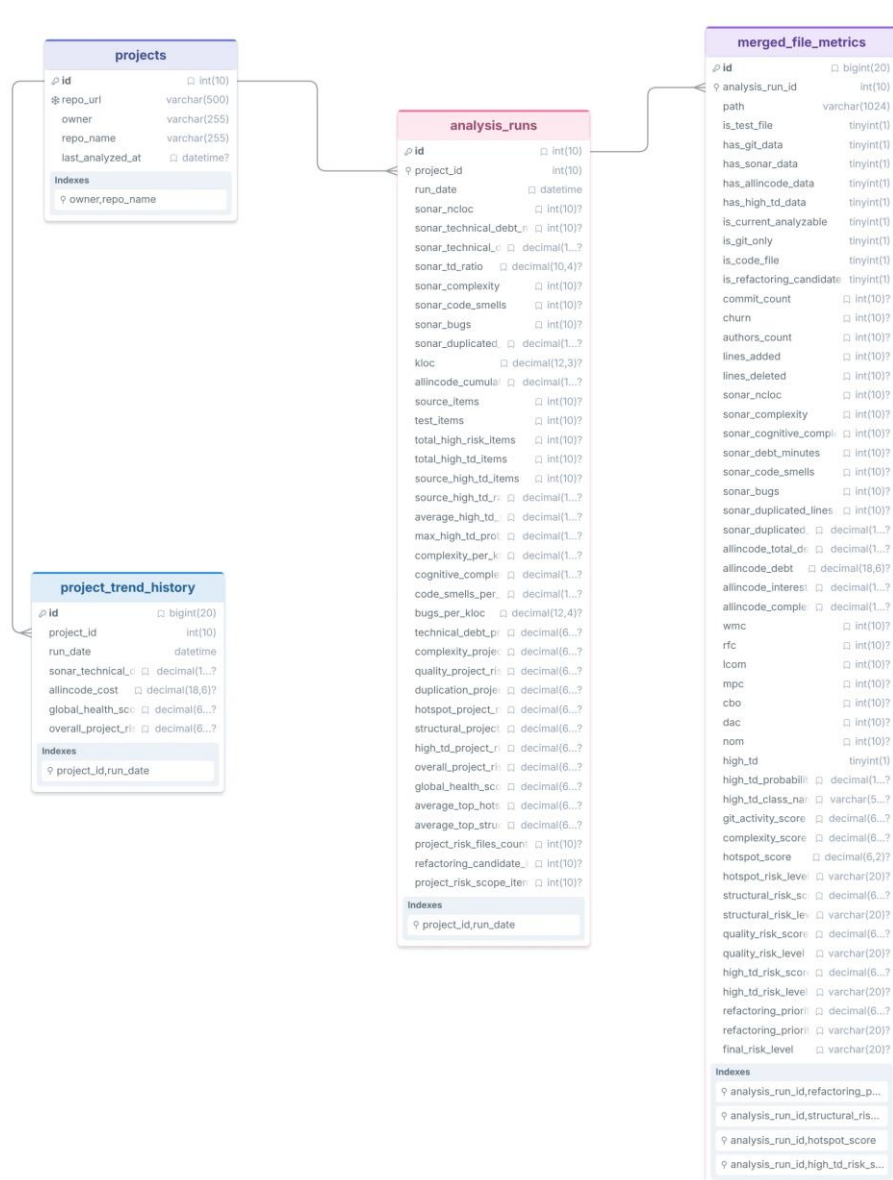
Ο πίνακας `merged_file_metrics` αποθηκεύει τις ενοποιημένες εγγραφές σε επίπεδο αρχείου. Κάθε εγγραφή συνδέεται με συγκεκριμένο `analysis_run` και περιλαμβάνει τη διαδρομή, τις διαθέσιμες ακατέργαστες τιμές, τους υπολογισμένους δείκτες και τις πληροφορίες επιλεξιμότητας. Η αποθήκευση των ακατέργαστων μετρικών μαζί με τους υπολογισμένους δείκτες είναι σημαντική για την ερμηνεία των αποτελεσμάτων. Αν διατηρούνταν μόνο το τελικό Refactoring Priority Score, δεν θα ήταν εύκολο να εξεταστεί ποια μετρική ή ποιο στοιχείο επηρέασε την τελική κατάταξη. Ο πίνακας περιλαμβάνει επίσης τις ενδείξεις που δείχνουν:

- αν υπάρχουν δεδομένα από κάθε πηγή
- αν το αρχείο είναι αρχείο δοκιμών
- αν αντιστοιχεί σε στοιχείο που μπορεί να αναλυθεί
- αν πρόκειται για διαδρομή που εμφανίζεται μόνο στο Git
- αν ανήκει σε αναγνωρισμένο τύπο κώδικα
- αν αποτελεί υποψήφιο αρχείο για αναδόμηση

Οι εγγραφές συνδέονται με το `analysis_runs` μέσω ξένου κλειδιού με `ON DELETE CASCADE` [47]. Έτσι, όταν διαγράφεται μια παλαιότερη πλήρης εκτέλεση ανάλυσης, αφαιρούνται αυτόματα και οι αντίστοιχες εγγραφές σε επίπεδο αρχείου.

4.3.1.4 Πίνακας Project Trend History

Ο πίνακας `project_trend_history` αποτελεί το ελαφρύτερο ιστορικό της εφαρμογής. Περιλαμβάνει μόνο τις τιμές που απαιτούνται για τη χρονική εξέλιξη: ημερομηνία, ημέρες τεχνικού χρέους, κόστος του AllinCode και Global Health Score. Κάθε εγγραφή συνδέεται απευθείας με το `project` και όχι με το `analysis_run`, ώστε να διατηρεί τις ιστορικές τιμές όλων των επιτυχών αναλύσεων, ανεξάρτητα από τον περιορισμό των δύο πιο πρόσφατων αναλύσεων.



Εικόνα 4.1 - Σχηματική απεικόνιση των πινάκων της βάσης δεδομένων

Οι βασικές σχέσεις είναι ένα-προς-πολλά. Ένα project μπορεί να συνδέεται με πολλά analysis_runs. Κάθε analysis_run μπορεί να περιλαμβάνει πολλές εγγραφές merged_file_metrics. Ακόμη, το ίδιο project μπορεί να έχει πολλές εγγραφές project_trend_history. Τα ξένα κλειδιά συνδέουν κάθε αποτέλεσμα με το αντίστοιχο project και analysis_run, διατηρώντας την αναφορική ακεραιότητα των δεδομένων.

4.3.2 Υλοποίηση και Σύνδεση της MariaDB

4.3.2.1 Διαχείριση Συνδέσεων με τη Βάση Δεδομένων (Connection Pool)

Η σύνδεση με τη MariaDB υλοποιείται στο db.js μέσω του mysql2/promise. Δημιουργείται ένα σύνολο επαναχρησιμοποιούμενων συνδέσεων (connection pool) με προεπιλεγμένα στοιχεία τοπικής εγκατάστασης και δυνατότητα αντικατάστασης μέσω των μεταβλητών DB_HOST, DB_PORT, DB_USER, DB_PASSWORD και DB_NAME. Ο μέγιστος αριθμός ταυτόχρονων συνδέσεων έχει

οριστεί σε 10. Η χρήση του pool επαναχρησιμοποιεί ανοικτές συνδέσεις και αποφεύγει το κόστος δημιουργίας νέας σύνδεσης για κάθε ερώτημα. Η επίσημη τεκμηρίωση του MySQL2 προτείνει τη χρήση pools με promises και τονίζει ότι κάθε σύνδεση πρέπει να απελευθερώνεται μετά την ολοκλήρωση της λειτουργίας της [45]. Στην εφαρμογή το saveAnalysis.js αποκτά σύνδεση, εκτελεί τη συναλλαγή και την απελευθερώνει στο finally.

4.3.2.2 Αποθήκευση Δεδομένων

Η saveAnalysisResult() εκτελεί όλες τις λειτουργίες αποθήκευσης μέσα σε μια συναλλαγή. Αρχικά καλείται beginTransaction(). Αν ολοκληρωθούν επιτυχώς οι εγγραφές σε όλους τους πίνακες, καθώς και ο καθαρισμός των παλαιότερων εκτελέσεων, καλείται η commit(). Αν οποιοδήποτε βήμα αποτύχει, εκτελείται rollback() και το σφάλμα επιστρέφεται στην κεντρική διαδρομή του API. Το COMMIT οριστικοποιεί τις αλλαγές μιας συναλλαγής και τις καθιστά ορατές στις επόμενες συναλλαγές [46].

```
async function saveAnalysisResult({
  repoUrl,
  result,
  durationSeconds = null
}) {
  const projectMetrics =
    result.project_metrics || {};

  const mergedFileMetrics =
    result.merged_file_metrics || [];

  const connection =
    await pool.getConnection();

  try {
    await connection.beginTransaction();

    const projectId = await upsertProject(
      connection,
      projectMetrics,
      repoUrl
    );

    const analysisRunId =
      await insertAnalysisRun(
        connection,
        projectId,
        projectMetrics,
```

```

        durationSeconds
    );

    const insertedFileMetrics =
        await insertMergedFileMetrics(
            connection,
            analysisRunId,
            mergedFileMetrics
        );

    await insertProjectTrendHistory(
        connection,
        projectId,
        projectMetrics
    );

    await cleanupOldAnalysisRuns(
        connection,
        projectId
    );

    await connection.commit();

    return {
        project_id: projectId,
        analysis_run_id: analysisRunId,
        inserted_file_metrics:
            insertedFileMetrics
    };
} catch (error) {
    await connection.rollback();
    throw error;
} finally {
    connection.release();
}
}

```

Απόσπασμα κώδικα 4.7: Αποθήκευση των αποτελεσμάτων ανάλυσης μέσα σε συναλλαγή βάσης δεδομένων(saveAnalysis.js).

Η εγγραφή του έργου γίνεται με την εντολή ON DUPLICATE KEY UPDATE. Επειδή το repo_url έχει περιορισμό UNIQUE, μια νέα εκτέλεση ενημερώνει την ημερομηνία της τελευταίας ανάλυσης χωρίς να δημιουργεί δεύτερη εγγραφή για το ίδιο έργο. Η χρήση της LAST_INSERT_ID(id) επιτρέπει την ανάκτηση του ίδιου αναγνωριστικού έργου, τόσο κατά τη δημιουργία νέας εγγραφής όσο και κατά την ενημέρωση μιας υπάρχουσας.

4.3.2.3 Αποθήκευση στον Πίνακα analysis_run

Η insertAnalysisRun() δημιουργεί δυναμικά τη λίστα των στηλών και των θέσεων παραμέτρων (placeholders). Πριν από την εισαγωγή εφαρμόζονται βοηθητικές συναρτήσεις που μετατρέπουν κείμενα με ποσοστά, νομισματικά σύμβολα ή δεκαδικά κόμματα σε αριθμητικές τιμές. Έτσι η βάση λαμβάνει ομοιόμορφους αριθμητικούς τύπους ακόμη και όταν μια εξωτερική πηγή επιστρέφει μορφοποιημένη συμβολοσειρά (formatted string).

4.3.2.4 Αποθήκευση στον Πίνακα Merged File Metrics

Οι εγγραφές των αρχείων μπορεί να είναι εκατοντάδες. Για τον λόγο αυτό, το insertMergedFileMetrics() δεν εκτελεί ξεχωριστό INSERT για κάθε γραμμή. Δημιουργεί ένα πίνακα τιμών και πραγματοποιεί μαζική εισαγωγή. Με αυτόν τον τρόπο, περιορίζονται ο αριθμός των αιτημάτων προς τη βάση και όλες οι εγγραφές παραμένουν μέσα στην ίδια συναλλαγή. Κάθε εγγραφή περιλαμβάνει: την κανονικοποιημένη διαδρομή, τις ενδείξεις διαθεσιμότητας των πηγών, τις ακατέργαστες μετρικές, τα πεδία High-TD, τις ενδείξεις επιλεξιμότητας και τους δείκτες σε επίπεδο αρχείου. Οι τιμές None της Python μετατρέπονται σε NULL στην SQL, ώστε η απουσία δεδομένων να είναι διακριτή από το μηδέν.

4.3.3 Snapshots, Ιστορικό και Πολιτική Διατήρησης

Στην παρούσα υλοποίηση διατηρούνται πλήρως μόνο τα δύο πιο πρόσφατα analysis_runs κάθε έργου. Ο περιορισμός αυτός αποτελεί πολιτική διατήρησης και όχι περιορισμό του σχήματος της βάσης, το οποίο επιτρέπει τη σύνδεση ενός project με περισσότερα analysis_runs. Μετά την εισαγωγή μιας νέας ανάλυσης, η cleanupOldAnalysisRuns() ταξινομεί τα analysis_runs του συγκεκριμένου project κατά φθίνουσα ημερομηνία και id, χρησιμοποιώντας τη συνάρτηση ROW_NUMBER(). Όσες εγγραφές έχουν σειρά μεγαλύτερη από δύο διαγράφονται. Λόγω του κανόνα ON DELETE CASCADE, αφαιρούνται αυτόματα και οι αντίστοιχες εγγραφές από τον πίνακα merged_file_metrics. Αντίθετα, οι ανεξάρτητες εγγραφές του project_trend_history παραμένουν, ώστε να διατηρείται το μακροχρόνιο ιστορικό των βασικών δεικτών. Τα δεδομένα αυτά αξιοποιούνται στην προβολή Temporal Evolution για την παρουσίαση της ιστορικής εξέλιξης του έργου, στο αντίστοιχο διάγραμμα. Η επιλογή αυτή επιτυγχάνει ισορροπία ανάμεσα στη δυνατότητα πρόσβασης στα πιο πρόσφατα πλήρη στιγμιότυπα και στον περιορισμό του όγκου της βάσης. Η διεπαφή χρησιμοποιεί σήμερα μόνο την τελευταία ανάλυση, αλλά η διατήρηση του αμέσως προηγούμενου επιτρέπει τη μελλοντική προσθήκη άμεσης σύγκρισης χωρίς αλλαγή της πολιτικής αποθήκευσης.

4.4 Διεπαφή Χρήστη και Οπτικοποιήσεις

4.4.1 Σχεδιαστική Προσέγγιση και Δομή Πλοήγησης

Πριν από την υλοποίηση του front-end σχεδιάστηκαν έξι προσχέδια (mockups). Στόχος ήταν να καθοριστούν η βασική δομή της εφαρμογής, η θέση των σημαντικότερων πληροφοριών και η πορεία που θα ακολουθεί ο χρήστης. Δεν αποτελούν ακριβή απεικόνιση της τελικής εφαρμογής, αλλά μια την αρχική σχεδιαστική προσέγγιση πάνω στην οποία βασίστηκε η ανάπτυξη του dashboard. Η πλοήγηση

οργανώνεται γύρω από ένα σταθερό πλευρικό μενού με τις πέντε βασικές οθόνες του Dashboard, με επιλογή επιστροφής στην αρχική οθόνη των έργων στο επάνω μέρος.

4.4.2 Υλοποίηση του Front-end

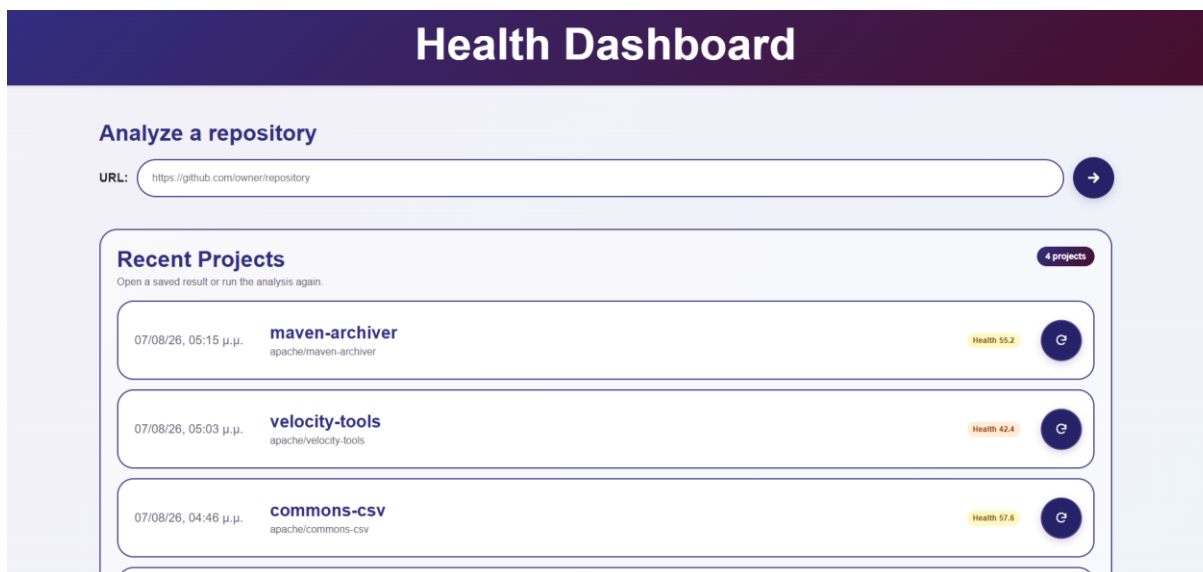
Η πλοήγηση στην εφαρμογή υλοποιείται με το React Router. Για τον σκοπό αυτό χρησιμοποιούνται τα BrowserRouter, Routes και Route, τα οποία αντιστοιχίζουν κάθε διαδρομή της εφαρμογής στην κατάλληλη οθόνη. Η αρχική διαδρομή / οδηγεί στην αρχική οθόνη Home, ενώ η διαδρομή /project/:projectId χρησιμοποιείται για την προβολή και την ανάλυση ενός έργου. Με αυτόν τον τρόπο, η μετάβαση μεταξύ των οθονών πραγματοποιείται χωρίς πλήρη επαναφόρτωση της εφαρμογής [48]. Κατά την έναρξη μιας νέας ανάλυσης, το URL του αποθετηρίου κωδικοποιείται και ενσωματώνεται στη διαδρομή /project/:projectId. Παράλληλα, μεταφέρεται η πληροφορία που δηλώνει ότι πρόκειται για νέα ανάλυση. Αντίστοιχα, όταν ο χρήστης επιλέγει ένα ήδη αποθηκευμένο έργο, μεταφέρεται το αναγνωριστικό της τελευταίας αποθηκευμένης ανάλυσης. Το ProjectDetails χρησιμοποιεί αυτή την πληροφορία για να ανακτήσει τα αντίστοιχα αποτελέσματα από τη MariaDB μέσω του REST API, χωρίς να ξεκινήσει νέα ανάλυση.

Οι πέντε οθόνες του dashboard δεν διαθέτουν ξεχωριστές διαδρομές URL. Η ενεργή προβολή διατηρείται στο ProjectDetails μέσω της μεταβλητής κατάστασης activeSection. Ανάλογα με την επιλογή του χρήστη, εμφανίζεται το αντίστοιχο τμήμα της διεπαφής. Η αρχική τιμή της μεταβλητής είναι η προβολή Overview, επομένως μετά από ανανέωση της οθόνης η επιλεγμένη εσωτερική προβολή επανέρχεται σε αυτή. Μελλοντικά, οι επιμέρους προβολές θα μπορούσαν να αποκτήσουν δικές τους διαδρομές ή παραμέτρους URL, ώστε να είναι δυνατή η απευθείας πρόσβαση σε συγκεκριμένη ενότητα του dashboard.

4.4.3 Υλοποίηση των Οθονών του Dashboard

4.4.3.1 Οθόνη UI-01: Αρχική Σελίδα

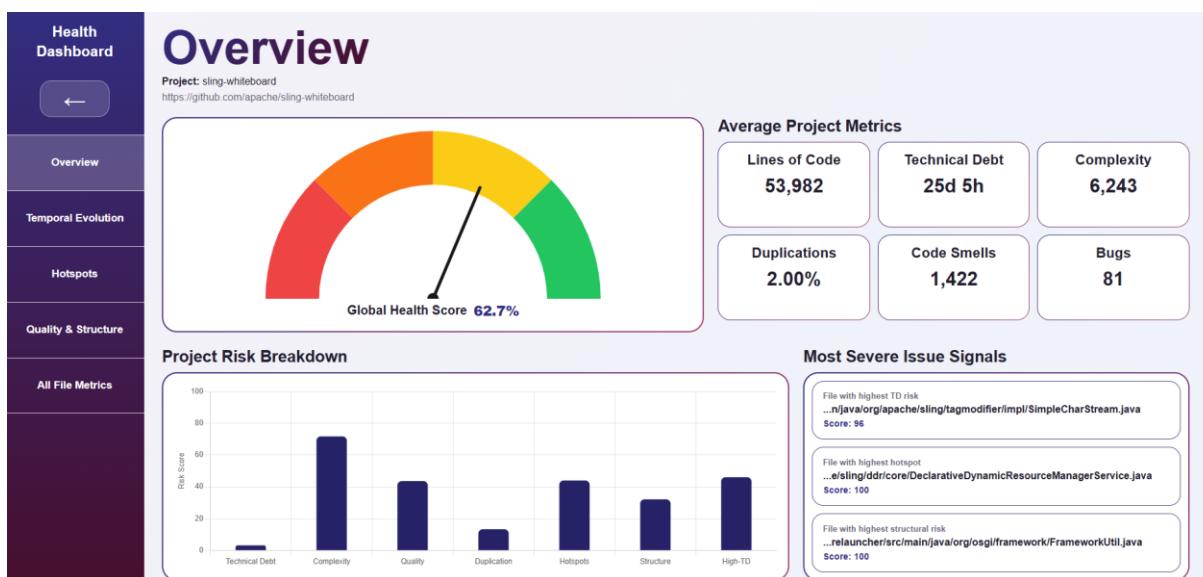
Στην εικόνα 4.2 παρουσιάζεται η αρχική σελίδα του dashboard. Ο χρήστης μπορεί να εισαγάγει το URL ενός αποθετηρίου του GitHub και να εκτελέσει μια νέα ανάλυση. Στη συνέχεια, το URL ελέγχεται ως προς τη βασική εγκυρότητά του και ξεκινά η νέα ανάλυση. Παράλληλα, εμφανίζονται οι τα ονόματα των έργων αποθετηρίων που έχουν ήδη αναλυθεί, μαζί με την ημερομηνία της τελευταίας ανάλυσης και το Global Health Score. Για κάθε έργο ο χρήστης μπορεί να εμφανίσει την τελευταία ανάλυσή του ή να ξεκινήσει νέα πατώντας το αντίστοιχο κουμπί στα δεξιά.



Εικόνα 4.2 - Οθόνη UI-01 Αρχική Σελίδα του dashboard

4.4.3.2 Οθόνη UI-02 Overview

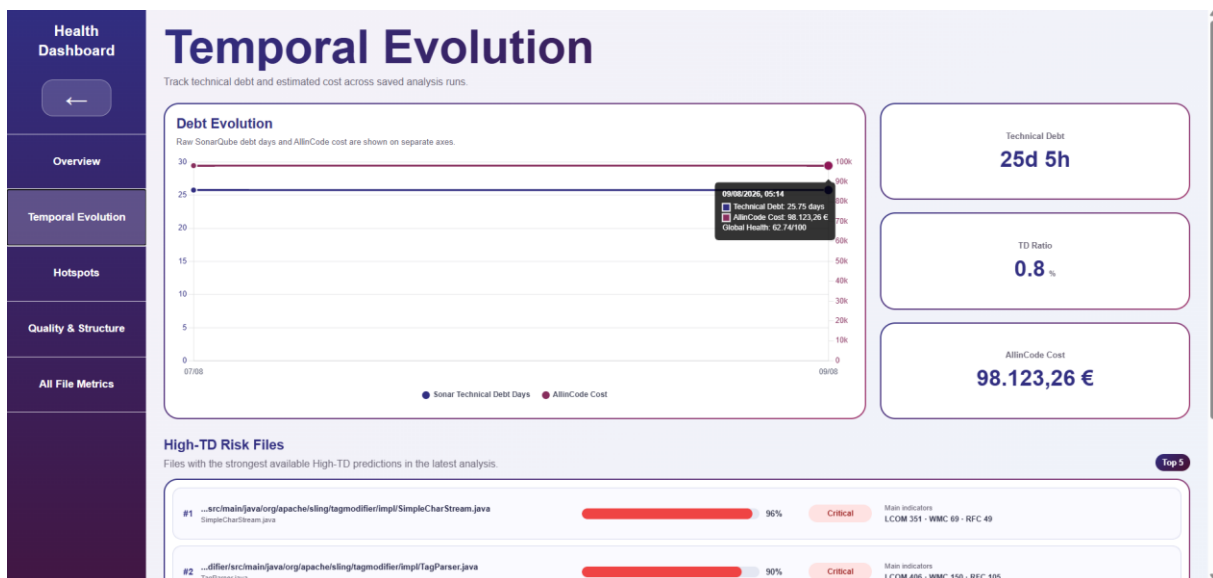
Επιλέγοντας ένα ήδη αποθηκευμένο έργο ή με την ολοκλήρωση μιας νέας ανάλυσης, ο χρήστης οδηγείται στην οθόνη Overview (Εικόνα 4.3). Στη συγκεκριμένη περίπτωση, παρουσιάζονται τα αποτελέσματα του έργου sling-whitboard. Το Global Health Score εμφανίζεται σε μορφή μετρητή σε κλίμακα 0-100 αριστερά της οθόνης, ενώ δεξιά του υπάρχουν κάρτες που παρουσιάζουν βασικές μετρικές του έργου όπως το τεχνικό χρέος, η πολυπλοκότητα και οι οσμές κώδικα. Επιπλέον, στο κάτω μέρος της οθόνης βρίσκεται το ραβδόγραμμα Project Risk Breakdown, το οποίο απεικονίζει τους διαθέσιμους επιμέρους δείκτες κινδύνου του έργου. Με αυτόν τον τρόπο, ο χρήστης μπορεί να διακρίνει άμεσα ποιες διαστάσεις επιβαρύνουν περισσότερο το έργο. Ακόμη, εμφανίζεται η περιοχή Signals, όπου επισημαίνονται το αρχείο με την υψηλότερη πιθανότητα High-TD, την υψηλότερη βαθμολογία hotspot και τον υψηλότερο δομικό κίνδυνο. Τέλος, στην αριστερή πλευρά της οθόνης βρίσκεται το πλευρικό μενού πλοήγησης, από το οποίο ο χρήστης μπορεί να μεταβεί στις υπόλοιπες προβολές του dashboard. Στο επάνω μέρος του μενού υπάρχει επίσης η επιλογή επιστροφής στην αρχική οθόνη, πατώντας το αντίστοιχο κουμπί.



Εικόνα 4.3 - Οθόνη UI-02 Overview του έργου sling-whitboard

4.4.3.3 Οθόνη UI-03: Temporal Evolution

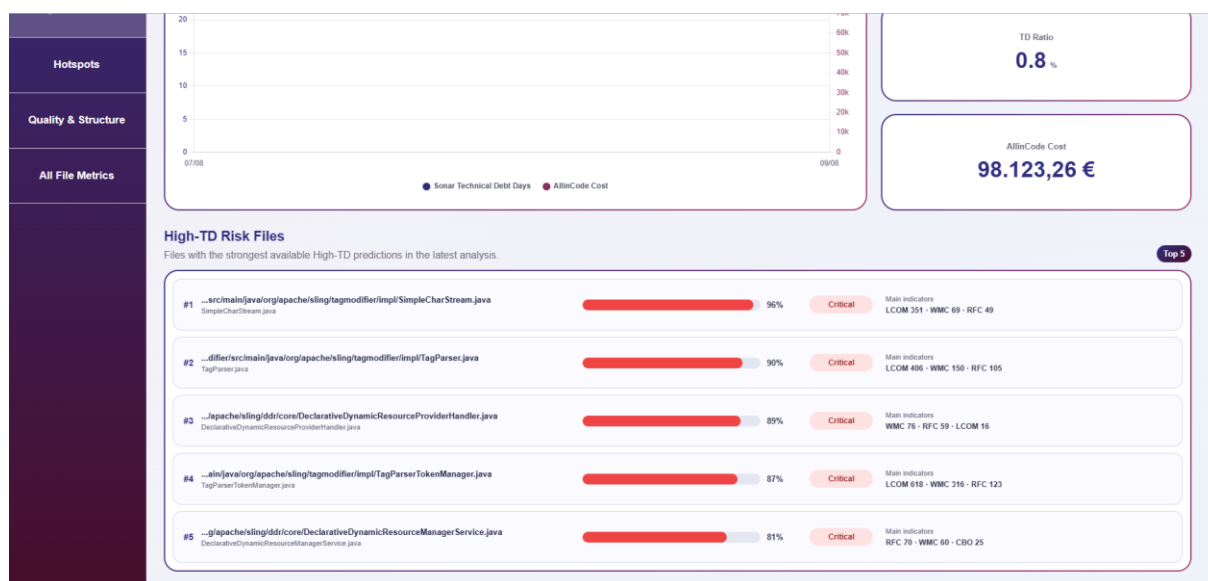
Επιλέγοντας το Temporal Evolution από το πλευρικό μενού, ο χρήστης μεταφέρεται στην αντίστοιχη οθόνη, όπως φαίνεται στην Εικόνα 4.4. Στο πάνω μέρος της οθόνης, εμφανίζεται ένα γραμμικό διάγραμμα με δύο κατακόρυφους άξονες. Η πρώτη γραμμή αποτυπώνει το σωρευτικό κόστος που σχετίζεται με το τεχνικό χρέος του έργου, ενώ η δεύτερη παρουσιάζει το τεχνικό χρέος σε ημέρες. Οι δύο γραμμές χρησιμοποιούν διαφορετικούς κατακόρυφους άξονες, καθώς εκφράζονται σε διαφορετικές μονάδες μέτρησης. Ο χρήστης μέσω του διαγράμματος μπορεί να παρακολουθεί τη χρονική εξέλιξη του τεχνικού χρέους του έργου. Με την τοποθέτηση του δείκτη πάνω σε κάποιο σημείο των γραμμών, εμφανίζονται αναδυόμενες πληροφορίες σχετικές με τη συγκεκριμένη ανάλυση. Επιπλέον, στα δεξιά του διαγράμματος παρουσιάζονται σε ξεχωριστές κάρτες το τεχνικό χρέος, το ποσοστό τεχνικού χρέους και το σωρευτικό κόστος της τελευταίας ανάλυσης.



Εικόνα 4.4 - Οθόνη UI-03 Temporal Evolution του έργου sling-whiteboard

4.4.3.4 Οθόνη UI-03: Περιοχή High-TD Files

Στο κάτω μέρος της οθόνης Temporal Evolution παρουσιάζεται η περιοχή High-TD Risk Files, όπως φαίνεται στην Εικόνα 4.5. Πρόκειται για μια λίστα η οποία εμφανίζει έως πέντε αρχεία της τελευταίας ανάλυσης με τις ισχυρότερες ενδείξεις High-TD, σε φθίνουσα σειρά. Για κάθε αρχείο, η πιθανότητα High-TD παρουσιάζεται ως ποσοστό και οπτικοποιείται μέσω γραμμής προόδου. Εμφανίζονται επίσης το επίπεδο κινδύνου και οι βασικοί δομικοί δείκτες που σχετίζονται με τον χαρακτηρισμό του συγκεκριμένου αρχείου ως High-TD.



Εικόνα 4.5 - Οθόνη UI-03: Περιοχή High-TD Risk Files για το έργο sling-whiteboard

4.4.3.5 Οθόνη UI-04 Hotspots

Στην Εικόνα 4.6 παρουσιάζεται η οθόνη Hotspots. Στο επάνω μέρος της οθόνης παρατηρείται ένα διάγραμμα φυσαλίδων που προσφέρει πληροφορίες σχετικά με τα hotspots του έργου. Ο οριζόντιος άξονας εκφράζει την πολυπλοκότητα και ο κατακόρυφος τον αριθμό των υποβολών αλλαγών Git. Το μέγεθος κάθε φυσαλίδας αντιστοιχεί στη μεταβολή κώδικα του αρχείου, ενώ ο χρωματισμός της συνδέεται με το επίπεδο κινδύνου. Ο χρήστης έχει τη δυνατότητα να περιορίσει το διάγραμμα στα 25, 40 ή 60 σημαντικότερα αρχεία ή να εμφανίσει όλα τα επιλέξιμα αρχεία, μέσω του αναπτυσσόμενου μενού που βρίσκεται στο επάνω μέρος του διαγράμματος. Με την τοποθέτηση του δείκτη πάνω σε μια φυσαλίδα, εμφανίζονται συνοπτικές αναδυόμενες πληροφορίες για το αντίστοιχο αρχείο. Όταν δεν έχει επιλεγεί κάποιο αρχείο, η περιοχή Project Details στα δεξιά του διαγράμματος εμφανίζει συνοπτικές πληροφορίες για τον συνολικό πληθυσμό. Με την επιλογή μιας φυσαλίδας, ενημερώνεται η περιοχή και ο τίτλος της αλλάζει σε File Details. Σε αυτήν ο χρήστης μπορεί να δει αναλυτικότερες πληροφορίες για το συγκεκριμένο αρχείο, όπως φαίνεται στην Εικόνα 4.7. Τέλος, αν το επιλεγμένο αρχείο ανήκει στον πίνακα Refactoring Priority που βρίσκεται στο κάτω μέρος της οθόνης, η αντίστοιχη σειρά του πίνακα επισημαίνεται. Έτσι, ο χρήστης μπορεί να εντοπίσει επιπλέον πληροφορίες σχετικά με την προτεραιότητα αναδόμησης του αρχείου.



Εικόνα 4.6 - Οθόνη UI-04 Hotspots του έργου sling-whiteboard



Εικόνα 4.7 - Οθόνη UI-04: Επιλογή αρχείου στο διάγραμμα φυσαλίδας για το έργο sling-whiteboard

4.4.3.6 Οθόνη UI-04: Πίνακας Refactoring Priority

Στην οθόνη Hotspots εμφανίζεται και ο πίνακας Refactoring Priority, όπως φαίνεται στην Εικόνα 4.8. Ο πίνακας παρουσιάζει τα 15 αρχεία του έργου με την υψηλότερη προτεραιότητα αναδόμησης, ταξινομημένα κατά φθίνουσα τιμή του Refactoring Priority Score. Για κάθε αρχείο περιλαμβάνονται οι επιμέρους δείκτες που συμβάλλουν στη διαμόρφωση της τελικής προτεραιότητας, καθώς και το αντίστοιχο επίπεδο κινδύνου.

File	Hotspot	Structure	Quality	High-TD	Refactoring Score	Level
#1 ...e/sling/ddr/core/DeclarativeDynamicResourceManagerService.java	100	64.06	100	81	87.32	Critical
#2 ...src/main/java/org/apache/sling/tagmodifier/impl/TagParser.java	88.32	97.29	60	90	84.91	Critical
#3 ...a/org/apache/sling/tagmodifier/impl/TagParserTokenManager.java	88.32	90.06	60	87	82.44	Critical
#4 ...relauncher/src/main/java/org/osgi/framework/FrameworkUtil.java	81.85	100	73.33	52	81.96	Critical
#5 ...sling/ddr/core/DeclarativeDynamicResourceProviderHandler.java	100	59.29	40	89	74.69	High
#6 ...gn/src/main/java/org/apache/tomcat/buildutil/SignCodeMojo.java	100	68.73	30	74	74.02	High
#7 ...g/feature/launcher/extensions/connect/impl/PosiSRLauncher.java	93.11	57.72	56.67	-	73.56	High
#8 ...n/java/org/apache/sling/tagmodifier/impl/SimpleCharStream.java	72.55	71.75	60	96	71.61	High
#9 ...sling/feature/launcher/atomos/config/AtomosConfigLauncher.java	74.53	80.35	63.33	54	71.52	High
#10 ...a/org/apache/sling/microscripting/helpers/EspReader.java	74.83	62.37	60	-	66.83	High
#11 ...rc/main/java/org/apache/sling/offline/impl/OfflineContext.java	57.97	83.48	60	-	66.76	High
#12 ...g/apache/sling/models/persister/impl/ReflectionUtils.java	83.16	75.4	16.67	-	65.23	High
#13 ...sling/remote/resourceprovider/impl/RemoteResourceProvider.java	74.83	70.93	20	81	62.82	High
#14 ...main/java/org/apache/sling/upgrade/impl/UpgradeWebConsole.java	57.61	80.68	30	-	61.92	High
#15 ...sling/microscripting/helpers/servlets/SlingSafeMethodsServlet.java	60.94	80.89	36.67	-	61.84	High

Εικόνα 4.8 - Οθόνη UI-04: Πίνακας Refactoring Priority του έργου sling-whiteboard

4.4.3.7 Οθόνη UI-05 Quality & Structure

Στην Εικόνα 4.9 παρουσιάζεται η οθόνη Quality & Structure, η οποία συγκεντρώνει πληροφορίες σχετικά με την ποιότητα και τη δομή του κώδικα σε επίπεδο αρχείου. Το πρώτο διάγραμμα παρουσιάζει τα αρχεία με το υψηλότερο Quality Risk. Τοποθετώντας τον δείκτη επάνω σε κάποιο αρχείο του διαγράμματος, εμφανίζονται αναδυόμενες πληροφορίες σχετικά με τα ποιοτικά χαρακτηριστικά του. Στη δεξιά πλευρά της οθόνης, ένα δεύτερο διάγραμμα αποτυπώνει την κατανομή των αρχείων ανά επίπεδο δομικού κινδύνου.



Εικόνα 4.9 - Οθόνη UI-05 Quality & Structure του έργου sling-whiteboard

4.4.3.8 Οθόνη UI-05: Πίνακας Structure Issues per File

Στο κάτω μέρος της οθόνης Quality & Structure περιλαμβάνεται επίσης ο πίνακας Structure Issues per File, στον οποίο παρουσιάζονται τα αρχεία με τον υψηλότερο δομικό κίνδυνο. Περιέχει τη διαδρομή του κάθε αρχείου, τις αντίστοιχες δομικές μετρικές και το επίπεδο δομικού κινδύνου, όπως φαίνεται στην Εικόνα 4.10. Ο πίνακας επιτρέπει στον χρήστη να εξετάσει το κάθε αρχείο με μεγαλύτερη λεπτομέρεια, ενώ τα κελιά χρησιμοποιούν χρωματική κωδικοποίηση για την ευκολότερη διάκριση των

υψηλότερων τιμών των επιμέρους μετρικών. Η χρωματική κωδικοποίηση βασίζεται σε εσωτερική συγκριτική αξιολόγηση μεταξύ των αρχείων του ίδιου έργου, οπότε δεν είναι ιδανική για άμεση σύγκριση μεταξύ διαφορετικών έργων.

Path	WMC	CBO	RFC	MPC	NOM	DAC	LCOM	Structure Risk
#1 ...her/src/main/java/org/osgi/framework/FrameworkUtil.java	59	71	487	428	59	0	514	100
#2 .../apache/sling/feature/launcher/atomos/AtomosRunner.java	40	39	162	133	19	0	103	100
#3 ...n/java/org/apache/sling/tagmodifier/impl/TagParser.java	150	17	105	76	29	9	406	97.29
#4 .../apache/sling/tagmodifier/impl/TagParserTokenManager.java	316	7	123	88	35	1	618	90.06
#5 .../apache/sling/commons/log/json/impl/OsgiConfigurator.java	14	21	85	71	14	2	91	84.01
#6 ...java/org/apache/sling/impl/OfflineContext.java	54	26	64	10	54	0	1,431	83.48
#7 ...sourceprovider/dropbox/impl/DropboxStorageProvider.java	43	33	67	55	12	2	39	82.44
#8 ...icrosling/helpers/servlets/SlingSafeMethodsServlet.java	18	28	111	93	18	1	64	80.89
#9 ...va/org/apache/sling/upgrade/impl/UpgradeWebConsole.java	13	30	88	75	13	1	78	80.68
#10 ...eature/launcher/atomos/config/AtomosConfigLauncher.java	38	31	107	0	0	0	70	80.35
#11 ...che/sling/microsling/servlet/MicroslingMainServlet.java	12	25	91	79	12	4	66	77.39
#12 ...e/sling/models/persistor/impl/ReflectionUtils.java	12	18	87	75	12	0	66	75.4
#13 ...org/apache/sling/tagmodifier/impl/SimpleCharStream.java	69	8	49	22	27	0	351	71.75
#14 ...emote/resourceprovider/impl/RemoteResourceProvider.java	99	27	86	0	0	0	27	70.93
#15 ...java/org/apache/sling/tagmodifier/impl/DBObjectBuilder.java	19	10	64	45	19	0	171	70.49
#16 ...main/java/org/apache/tomcat/buildutil/SignCodeMojo.java	39	22	53	37	3	0	15	68.73
#17 ...ddr/core/DeclarativeDynamicResourceManagerService.java	60	25	70	0	0	0	3	64.06

Εικόνα 4.10 - Οθόνη UI-05: Πίνακας Structure Issues per File του έργου sling-whiteboard

4.4.3.9 Οθόνη UI-06 All File Metrics

Στην Εικόνα 4.11 παρουσιάζεται η οθόνη All File Metrics, η οποία παρέχει τη δυνατότητα αναλυτικής εξέτασης των αποθηκευμένων μετρικών για κάθε αρχείο του έργου. Λόγω του μεγάλου αριθμού διαθέσιμων μετρικών, οι στήλες του πίνακα οργανώνονται σε τρεις κατηγορίες: Risk Scores, Git & Quality και Structure & Debt. Ο χρήστης μπορεί να εναλλάσσεται μεταξύ των κατηγοριών πατώντας την αντίστοιχη επιλογή στο επάνω μέρος του πίνακα. Η κατηγορία Risk Scores συγκεντρώνει τους υπολογισμένους δείκτες κινδύνου και προτεραιότητας του dashboard. Η κατηγορία Git & Quality περιλαμβάνει μετρικές που σχετίζονται με το ιστορικό αλλαγών και την ποιότητα του κώδικα, όπως ο αριθμός υποβολών αλλαγών, η μεταβολή κώδικα, η πολυπλοκότητα, οι οσμές κώδικα και τα σφάλματα. Τέλος, η κατηγορία Structure & Debt περιλαμβάνει τις δομικές μετρικές WMC, CBO, RFC, MPC, NOM, DAC και LCOM, καθώς και πληροφορίες που σχετίζονται με το τεχνικό χρέος. Στον πίνακα εφαρμόζεται χρωματική κωδικοποίηση στις τιμές που εκφράζουν δείκτες ή επίπεδα κινδύνου, ώστε να διευκολύνεται η οπτική διάκριση της σοβαρότητάς τους. Στην κατηγορία Risk Scores η κωδικοποίηση καλύπτει όλους τους επιμέρους δείκτες. Στις κατηγορίες Git & Quality και Structure & Debt ωστόσο, η κωδικοποίηση περιορίζεται στις στήλες που σχετίζονται με την αξιολόγηση κινδύνου.

Health Dashboard

←

Overview

Temporal Evolution

Hotspots

Quality & Structure

All File Metrics

All File Metrics

Search, filter and compare all stored file-level metrics.

Search by path

Files

Risk Level

Clear

Export CSV

Search file or path...

Current source files

All levels

Risk Scores

Git & Quality

Structure & Debt

Showing 1 – 25 of 426

Rows per page: 25

Path	Git Activity	Complexity Score	Hotspot	Structure	Quality	High-TD	Refactoring	Priority Level
org.apache.sling.drolicore/src/main/java/org/apache/sling/drolicore/DeclarativeDyn...	100	100	100	64.06	100	81	87.32	Critical
tag-modifier/src/main/java/org/apache/sling/tag/modifier/impl/TagParser.java	78	100	88.32	87.29	60	90	84.91	Critical
tag-modifier/src/main/java/org/apache/sling/tag/modifier/impl/TagParserTokenMana...	78	100	88.32	90.06	60	87	82.44	Critical
connectfeaturelauncher/src/main/java/org/apache/sling/feature/launcher/extension...	67	100	81.85	100	73.33	52	81.95	Critical
org.apache.sling.drolicore/src/main/java/org/apache/sling/drolicore/DeclarativeDyn...	100	100	100	59.29	40	89	74.69	Critical
codeign/src/main/java/org/apache/forcodeign/build/impl/CodeMojo.java	100	100	100	68.73	30	74	74.02	Critical
connectfeaturelauncher/src/main/java/org/apache/sling/feature/launcher/extension...	96.69	89.67	93.11	57.72	56.87	-	73.56	Critical
tag-modifier/src/main/java/org/apache/sling/tag/modifier/impl/SimpleCharStream.java	56	94	72.55	71.75	60	96	71.61	Critical
atomosfeaturelauncherconfig/src/main/java/org/apache/sling/feature/launcher/atom...	67	82.9	74.53	80.35	63.33	54	71.52	Critical
history/microsling/src/main/java/org/apache/sling/microsling/scripting/ScriptFile...	56	100	74.83	62.37	60	-	68.83	Critical
org.apache.sling.offline/src/main/java/org/apache/sling/offline/impl/OfflineContext.j...	56	60	57.97	83.48	60	-	66.76	Critical
SlingModelPersister/src/main/java/org/apache/sling/models/persister/impl/Util/Reflect...	75.06	92.13	83.16	75.4	16.67	-	65.23	Critical
it-is-cloudy-here/org-apache-sling-remote-resource-provider/src/main/java/org/apac...	56	100	74.83	70.93	20	81	62.82	Critical
upgrade/src/main/java/org/apache/sling/upgrade/impl/UpgradeWebConsole.java	78	42.55	57.61	80.68	30	-	61.92	Critical

Εικόνα 4.11 - Οθόνη UI-06 All File Metrics του έργου sling-whiteboard

4.4.3.10 Οθόνη UI-06: Εφαρμογή Φίλτρων

Στο επάνω μέρος της οθόνης παρέχεται επίσης η δυνατότητα αναζήτησης με βάση τη διαδρομή του αρχείου, καθώς και φίλτρα για τον περιορισμό των εμφανιζόμενων εγγραφών. Μέσω του αναπτυσσόμενου μενού που αφορά το σύνολο των αρχείων, ο χρήστης μπορεί να περιορίσει την προβολή σε συγκεκριμένες κατηγορίες αρχείων, όπως τα τρέχοντα αρχεία πηγαίου κώδικα ή τα αρχεία που είναι υποψήφια για αναδόμηση. Επιπλέον, επιλέγοντας μέσα από το αντίστοιχο αναπτυσσόμενο μενού μπορεί να περιορίσει τα αποτελέσματα με βάση το επίπεδο κινδύνου, όπως φαίνεται στην εικόνα 4.12. Ο χρήστης μπορεί ακόμη να ταξινομή τις εγγραφές με βάση τη στήλη του πίνακα που επιθυμεί, πατώντας επάνω στην αντίστοιχη κεφαλίδα. Το κουμπί Clear στα επιτρέπει την άμεση αφαίρεση των εφαρμοσμένων φίλτρων και την επαναφορά της προεπιλεγμένης προβολής. Τα αποτελέσματα μπορούν επίσης να εξαχθούν σε αρχείο CSV, το οποίο περιλαμβάνει το σύνολο των εγγραφών που προκύπτουν από τα ενεργά φίλτρα και την εφαρμοζόμενη ταξινόμηση, πατώντας αντίστοιχο κουμπί στη δεξιά πλευρά της οθόνης. Τέλος, για τη διευκόλυνση της εξέτασης μεγάλου αριθμού αρχείων, υπάρχει σελιδοποίηση με δυνατότητα εμφάνισης 25, 50 ή 100 εγγραφών ανά οθόνη. Η επιλογή αυτή γίνεται επίσης μέσω του αναπτυσσόμενου μενού, που παρατηρείται στην επάνω δεξιά πλευρά του πίνακα.

Health Dashboard

←

Overview

Temporal Evolution

Hotspots

Quality & Structure

All File Metrics

All File Metrics

Search, filter and compare all stored file-level metrics.

Search by path

Files

Risk Level

Clear

Export CSV

Search file or path...

Refactoring candidates

Medium

Risk Scores

Git & Quality

Structure & Debt

Showing 1 – 6 of 6

Rows per page: 50

Path	Git Activity	Complexity Score	Hotspot	Structure	Quality	High-TD	Refactoring	Priority Level
history/microsling/src/main/java/org/apache/sling/microsling/scripting/ScriptFiles...	53.98	66.97	60.13	58.95	23.33	-	51.55	Medium
mdresourceprovider/src/main/java/org/apache/sling/mdresource/impl/MarkdownFiles...	79.81	58.56	68.36	37.17	16.67	-	47.79	Medium
history/microsling/src/main/java/org/apache/sling/microsling/service/MicroslingMainS...	44.72	46.56	45.63	77.39	6.67	-	47.48	Medium
org.apache.sling.offline/src/main/java/org/apache/sling/offline/impl/Starter.java	93.4	53.41	70.63	0	56.67	-	47	Medium
upgrade/src/main/java/org/apache/sling/upgrade/BundleEntry.java	79.21	58.72	68.2	36.37	0	-	43.69	Medium
json-exporter/src/main/java/org/apache/sling/whiteboard/json-exporter/impl/JsonExport...	66.81	37.49	50.95	49.3	10	-	43.14	Medium

Εικόνα 4.12 - Οθόνη UI-06: Παράδειγμα εφαρμογής φίλτρων στα δεδομένα του έργου sling-whiteboard

Κεφάλαιο 5ο: Μελέτη Περίπτωσης και Αποτελέσματα

5.1 Επιλογή και Περιγραφή των Έργων Βαθμονόμησης

Η ανάπτυξη των σύνθετων δεικτών απαιτεί τη λήψη αποφάσεων σχετικά με τους μαθηματικούς τύπους, τα σημεία κορεσμού, τα βάρη και τον τρόπο συγκέντρωσης των αποτελεσμάτων σε επίπεδο αρχείου. Για τον λόγο αυτό πραγματοποιήθηκε η διαδικασία βαθμονόμησης, ώστε να ελεγχθεί η συμπεριφορά του μοντέλου και να εντοπιστούν διαμορφώσεις που οδηγούσαν σε υπερβολικά αυστηρά, επιεική ή ασταθή αποτελέσματα.

Η βαθμονόμηση πραγματοποιήθηκε σε δύο έργα διαφορετικό μέγεθος. Ως μικρό έργο επιλέχθηκε το apache/maven-archiver, με 2.327 KLOC και 7 αρχεία που πληρούσαν τα τελικά κριτήρια επιλεξιμότητας. Αντίστοιχα, ως μεγαλύτερο έργο επιλέχθηκε το apache/sling-whiteboard, με 53.976 KLOC και 385 εγγραφές που ανήκαν στον τελικό κοινό πληθυσμό C. Όσον αφορά τα χαρακτηριστικά των δύο έργων, το maven-archiver ανήκει στο οικοσύστημα Apache Maven και παρέχει βοηθητικές λειτουργίες για τη δημιουργία αρχείων JAR και άλλων πακέτων. Η περιορισμένη έκταση και ο μικρός αριθμός επιλέξιμων αρχείων του το καθιστούν κατάλληλο για τον έλεγχο της συμπεριφοράς του μοντέλου σε μικρά έργα. Από την άλλη, το sling-whiteboard αποτελεί τον πειραματικό χώρο του Apache Sling. Η μεγαλύτερη έκταση, η πολυμορφία των αρχείων και ο σημαντικά μεγαλύτερος αριθμός τους επιτρέπουν τον έλεγχο του μοντέλου σε ένα μεγαλύτερο και πιο σύνθετο αποθετήριο. Η διαφορά μεγέθους μεταξύ των δύο έργων επέτρεψε να εξεταστεί τόσο η συμπεριφορά των τύπων όσο και η επίδραση του πλήθους των αρχείων στον υπολογισμό των συγκεντρωτικών δεικτών από τις τιμές σε επίπεδο αρχείου.

5.2 Παρουσίαση Αποτελεσμάτων μέσω Dashboard

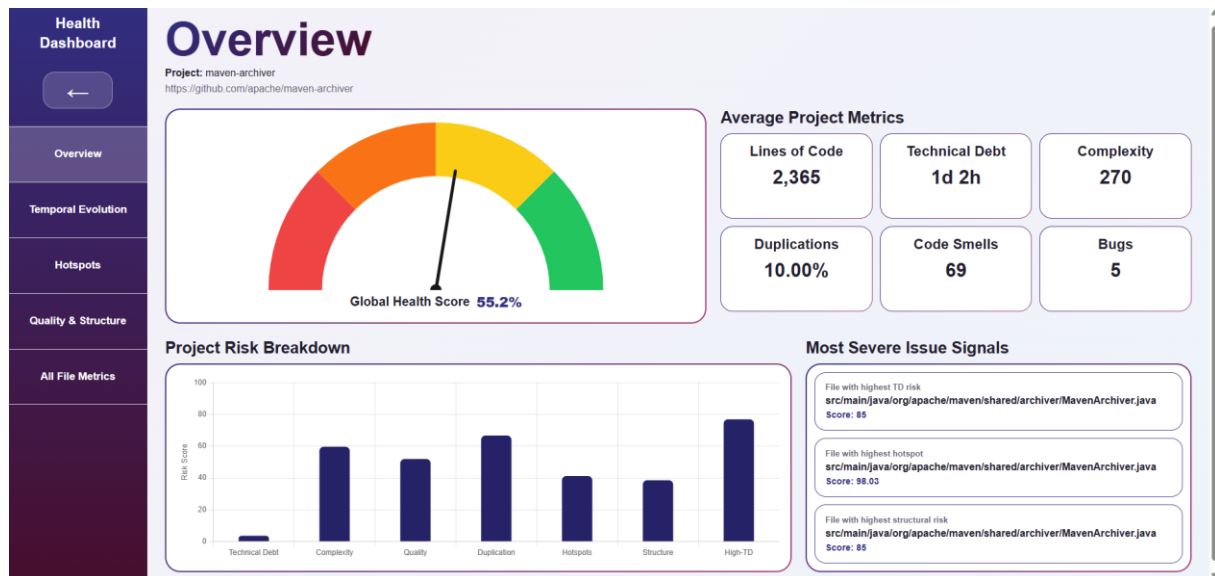
Η παρουσίαση της μελέτης περίπτωσης βασίζεται στα αποτελέσματα μετά την ολοκλήρωση της βαθμονόμησης. Οι ενδιάμεσες διαμορφώσεις χρησιμοποιήθηκαν για την επιλογή των τελικών τύπων, των κατωφλίων και των βαρών και εξετάζονται στην Ενότητα 5.4. Οι τιμές που ακολουθούν αντιστοιχούν στην τελική έκδοση του μοντέλου και αποτελούν τα αποτελέσματα που αποθηκεύονται και παρουσιάζονται από το dashboard.

Πίνακας 5.1 - Οι επτά project-level κίνδυνοι, το Overall Project Risk και το Global Health Score για το Maven Archiver και το Sling Whiteboard.

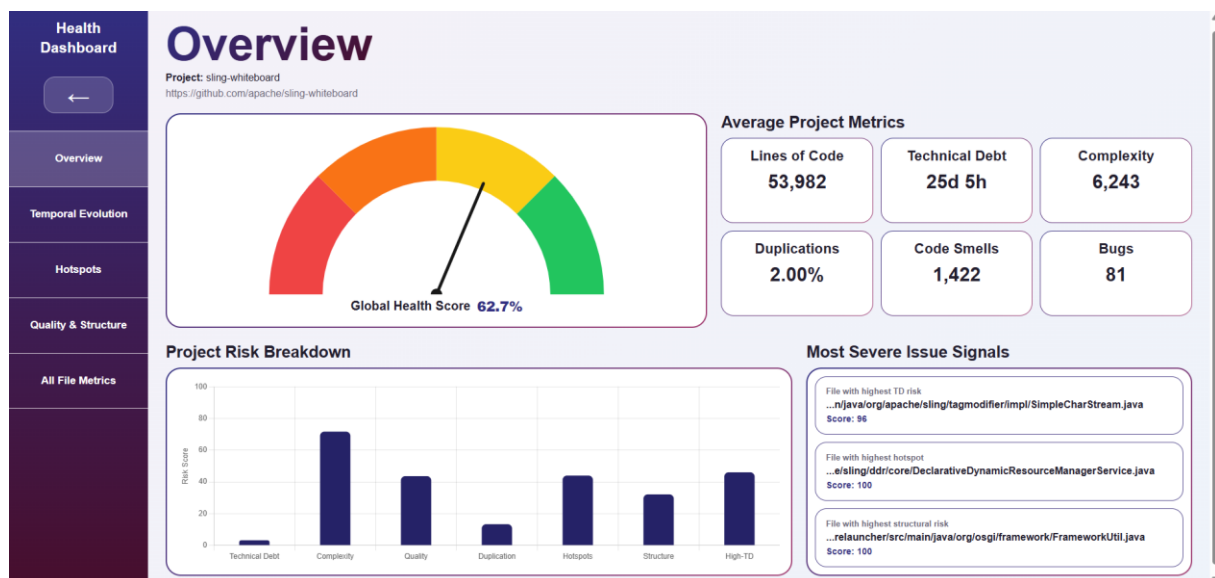
Project-level δείκτης	Maven Archiver	Sling Whiteboard
Technical Debt Project Risk	3,60	3,20
Complexity Project Risk	59,46	71,67
Quality Project Risk	52,03	43,62
Duplication Project Risk	66,67	13,33
Hotspot Project Risk	41,27	43,96
Structural Project Risk	39,15	32,12
High-TD Project Risk	76,46	46,02
Overall Project Risk	44,84	37,26
Global Health Score	55,16	62,74

Οι επιμέρους κίνδυνοι και το Overall Project Risk αυξάνονται προς το 100, οπότε μια υψηλότερη τιμή σημαίνει μεγαλύτερη επιβάρυνση. Αντίθετα, το Global Health Score αυξάνεται προς την καλύτερη

συνολική εικόνα. Με βάση την τελική διαμόρφωση, το Maven Archiver λαμβάνει Global Health Score 55,16, ενώ το Sling Whiteboard 62,74.



Εικόνα 5.1 - Οθόνη Overview του έργου maven-archiver



Εικόνα 5.2 - Οθόνη Overview του έργου sling-whiteboard

5.3 Ανάλυση Αποτελεσμάτων

Η διαφορά των 7,58 μονάδων στο Global Health Score δείχνει ότι το Sling Whiteboard παρουσιάζει συνολικά καλύτερη εικόνα από το Maven Archiver, χωρίς όμως αυτό να σημαίνει πως δεν έχει προβλήματα. Η διαφορά αυτή δεν οφείλεται σε μία μόνο μετρική, αλλά στη διαφορετική κατανομή των επιμέρους κινδύνων. Η αποσύνθεση του Global Health Score είναι επομένως απαραίτητη, καθώς δύο έργα μπορούν να λάβουν σχετικά κοντινές συνολικές βαθμολογίες για διαφορετικούς λόγους.

Το Maven Archiver παίρνει Global Health Score 55,16 και παρουσιάζει υψηλότερη επιβάρυνση στις διαστάσεις του High-TD, της επανάληψης κώδικα, της ποιότητας και του δομικού κινδύνου. Η

σημαντικότερη ένδειξη είναι το High-TD Project Risk, το οποίο φτάνει το 76,46. Ένα από τα επτά αρχεία του κοινού πληθυσμού έχει χαρακτηριστεί ως High-TD, με πιθανότητα 0,84. Παρόλο που είναι ένα μόνο αρχείο, αντιστοιχεί στο 14,29% του μικρού κοινού πληθυσμού και αυξάνει σημαντικά τη σχετική συνιστώσα του High-TD κινδύνου. Η συγκεκριμένη εγγραφή αποτελεί επομένως την πρώτη περιοχή που πρέπει να εξεταστεί σε επίπεδο αρχείου. Η δεύτερη σημαντικότερη επιβάρυνση του, προέρχεται από την επανάληψη κώδικα, με Duplication Project Risk 66,67. Ακολουθούν το Complexity Risk με 59,46 και το Quality Risk με 52,03. Η εικόνα αυτή συμφωνεί με την αυξημένη πυκνότητα προβλημάτων για το μέγεθος του έργου, καθώς καταγράφονται 30,08 οσμές κώδικα και 2,15 σφάλματα ανά KLOC. Επομένως, το μικρότερο μέγεθος του Maven Archiver δεν συνεπάγεται αυτόματα και χαμηλότερη επιβάρυνση. Παράλληλα, το Hotspot Project Risk είναι 41,27 και υπολογίζεται από τα πέντε αρχεία με τις υψηλότερες τιμές, επειδή ο κοινός πληθυσμός περιλαμβάνει μόνο επτά αρχεία και εφαρμόζεται το ελάχιστο όριο των πέντε. Το Structural Project Risk είναι 39,15. Οι δύο αυτές τιμές δείχνουν ότι υπάρχουν αρχεία στα οποία συνδυάζονται δραστηριότητα αλλαγών, πολυπλοκότητα και δομικές ενδείξεις, χωρίς όμως αυτές οι διαστάσεις να αποτελούν τις κυριότερες πηγές επιβάρυνσης του έργου. Το Technical Debt Project Risk παραμένει αρκετά χαμηλό, με τιμή 3,60, επειδή το Technical Debt Ratio είναι περίπου 0,9%. Αυτή η χαμηλή τιμή δεν έρχεται σε αντίθεση με το μέτριο Global Health Score. Δείχνει απλώς ότι η εκτιμώμενη προσπάθεια αποκατάστασης είναι μικρή σε σχέση με το μέγεθος του έργου. Αντίθετα, άλλες διαστάσεις όπως η επανάληψη κώδικα, τα High-TD αποτελέσματα, η πολυπλοκότητα και η πυκνότητα προβλημάτων ποιότητας, εξακολουθούν να επιβαρύνουν τη συνολική εικόνα.

Με βάση τα παραπάνω, φαίνεται πως η διερεύνηση του Maven Archiver θα έπρεπε να ξεκινήσει από το High-TD αρχείο και τις περιοχές με αυξημένη επανάληψη κώδικα. Σε δεύτερο επίπεδο μπορούν να εξεταστούν τα αρχεία στα οποία η πολυπλοκότητα συνδυάζεται με οσμές κώδικα και σφάλματα ή συχνές αλλαγές. Η σειρά αυτή δεν αποτελεί αυτόματη εντολή διόρθωσης, αλλά προτεραιοποίηση των περιοχών στις οποίες πρέπει να κατευθυνθεί αρχικά η ανθρώπινη αξιολόγηση.

Το Sling Whiteboard παίρνει υψηλότερο Global Health Score, ίσο με 62,74. Η βασικότερη πηγή επιβάρυνσης είναι η πολυπλοκότητα, καθώς το Complexity Project Risk φτάνει το 71,67. Η τιμή αυτή είναι συμβατή με την αυξημένη πυκνότητα γνωστικής πολυπλοκότητας, η οποία προσεγγίζει τις 99,36 μονάδες ανά KLOC. Επομένως, το μέγεθος και η μεγαλύτερη λειτουργική έκταση του έργου δημιουργούν σημαντικό φορτίο κατανόησης και συντήρησης, ακόμη και όταν οι υπόλοιπες διαστάσεις παρουσιάζουν καλύτερη εικόνα. Το Hotspot Project Risk είναι 43,96 και υπολογίζεται από τα 116 αρχεία με τις υψηλότερες τιμές στον κοινό πληθυσμό των 385 αρχείων. Το αποτέλεσμα δείχνει ότι ένα υποσύνολο του έργου συνδυάζει πολυπλοκότητα με αυξημένη δραστηριότητα αλλαγών. Τα αρχεία που εμφανίζουν ταυτόχρονα υψηλό Complexity Score και Hotspot Score αποτελούν καταλληλότερους στόχους διερεύνησης από αρχεία που είναι σύνθετα αλλά παραμένουν σταθερά και αλλάζουν σπάνια. Επιπλέον, στον κοινό πληθυσμό του Sling Whiteboard εντοπίζονται 12 High-TD αρχεία, τα οποία αντιστοιχούν στο 3,12% των 385 αρχείων. Οι τρεις υψηλότερες πιθανότητες είναι 0,96, 0,90 και 0,89, με μέσο όρο 0,9167. Οι υψηλές αυτές πιθανότητες αυξάνουν τη συνιστώσα έντασης του High-TD κινδύνου, όμως η μικρή αναλογία αρχείων που έχουν χαρακτηριστεί ως High-TD περιορίζει το συνολικό High-TD Project Risk στο 46,02. Τα συγκεκριμένα αρχεία παραμένουν σημαντικοί υποψήφιοι για έλεγχο, χωρίς όμως να υποδεικνύουν ότι μεγάλο μέρος του έργου χαρακτηρίζεται από υψηλό τεχνικό χρέος. Εμφανίζει επίσης Quality Risk 43,62 και Structural Risk 32,12. Αντίθετα, το Duplication Risk είναι αρκετά χαμηλότερο, με τιμή 13,33, ενώ το Technical Debt Risk περιορίζεται στο 3,20, καθώς το Technical Debt Ratio βρίσκεται περίπου στο 0,8%. Οι θετικές αυτές ενδείξεις αντισταθμίζουν σε σημαντικό βαθμό την αυξημένη πολυπλοκότητα και εξηγούν γιατί το Sling Whiteboard καταλήγει σε υψηλότερο Global Health Score.

Σύμφωνα λοιπόν με τα αποτελέσματα, η προτεραιοποίηση για το Sling Whiteboard καλό θα ήταν να εστιάσει αρχικά στα σύνθετα αρχεία που εμφανίζονται παράλληλα ως hotspots ή έχουν λάβει ισχυρή High-TD ένδειξη. Σε επόμενη φάση μπορούν να εξεταστούν οι περιοχές με προβλήματα ποιότητας ή αυξημένο δομικό φορτίο. Η επανάληψη κώδικα και ο συνολικός λόγος τεχνικού χρέους δεν αποτελούν τις κυριότερες πηγές επιβάρυνσης του συγκεκριμένου έργου.

Κατά τη σύγκριση των δύο έργων, χρειάζεται επίσης να ληφθεί υπόψη ο τρόπος με τον οποίο υπολογίζονται οι επιμέρους δείκτες. Ιδιαίτερα οι τιμές των Hotspot και Structural Project Risks πρέπει να ερμηνεύονται με προσοχή κατά τη σύγκριση των δύο έργων. Οι αντίστοιχοι δείκτες σε επίπεδο αρχείου κανονικοποιούνται ως προς το P95 κάθε αποθετηρίου, επομένως εκφράζουν κυρίως τη σχετική συγκέντρωση του κινδύνου μέσα στο εξεταζόμενο έργο. Η τιμή 43,96 του Sling και η τιμή 41,27 του Maven δεν αποδεικνύουν ότι τα δύο έργα έχουν το ίδιο απόλυτο επίπεδο δραστηριότητας και πολυπλοκότητας.

Συνολικά, τα τελικά αποτελέσματα διακρίνουν δύο διαφορετικά προφίλ τεχνικής επιβάρυνσης. Το Maven Archiver επηρεάζεται περισσότερο από την επανάληψη κώδικα, τα High-TD αποτελέσματα και την πυκνότητα προβλημάτων ποιότητας. Από την άλλη, το Sling Whiteboard επιβαρύνεται κυρίως από την πολυπλοκότητα και τη συγκέντρωση σύνθετων αρχείων με αυξημένη δραστηριότητα. Η προτεραιοποίηση λειτουργεί ως κατεύθυνση της διερεύνησης και όχι ως αυτόματη απόφαση αναδόμησης, καθώς η τελική επιλογή πρέπει να λαμβάνει υπόψη τον λειτουργικό ρόλο, την αρχιτεκτονική σημασία, τις εξαρτήσεις και το πραγματικό κόστος κάθε αλλαγής.

5.4 Βαθμονόμηση και ανάλυση ευαισθησίας του μοντέλου

Μετά την παρουσίαση των τελικών αποτελεσμάτων, εξετάζεται η διαδικασία που οδήγησε στην οριστική διαμόρφωση του μοντέλου. Η βαθμονόμηση πραγματοποιήθηκε πριν από την τελική ανάλυση και περιλάμβανε τη σύγκριση εναλλακτικών τύπων, κατωφλίων, ποσοστών συγκέντρωσης και βαρών. Η παρουσίασή της σε αυτό το σημείο αξιολογεί κατά πόσο οι τελικές επιλογές είναι αιτιολογημένες και αν το μοντέλο παραμένει σταθερό απέναντι σε περιορισμένες μεταβολές των παραμέτρων του. Οι πίνακες που ακολουθούν στα παρακάτω υποκεφάλαια δείχνουν τα αποτελέσματα που καταγράφηκαν στο αντίστοιχο στάδιο της βαθμονόμησης. Επειδή οι αλλαγές εφαρμόστηκαν διαδοχικά και ακολούθησε νέα τελική ανάλυση, ορισμένες ενδιαμέσες τιμές παρουσιάζουν μικρές αποκλίσεις από τα τελικά αποτελέσματα της Ενότητας 5.2.

5.4.1 Αρχική Διαμόρφωση και Αρχικά Αποτελέσματα

Στην αρχική διαμόρφωση, ο συνολικός κίνδυνος του έργου υπολογιζόταν ως εξής:

$$\begin{aligned} \text{Overall Project Risk} = & 0.25 \cdot \text{Technical Debt Risk} + 0.20 \cdot \text{Complexity Risk} + \\ & 0.15 \cdot \text{Quality Risk} + 0.10 \cdot \text{Duplication Risk} + 0.10 \cdot \text{Hotspot Risk} + \\ & 0.10 \cdot \text{Structural Risk} + 0.10 \cdot \text{HighTD Risk} \end{aligned}$$

Και το Global Health Score προέκυπτε από τον τύπο:

$$\text{Global Health Score} = 100 - \text{Overall Project Risk}$$

Τα αρχικά αποτελέσματα ήταν τα εξής:

Πίνακας 5.2 συνολικός κίνδυνος του έργου

Sling Whiteboard	Maven Archiver	Project-level δείκτης
16,00	18,00	Technical Debt Risk
86,01	72,37	Complexity Risk
43,62	53,29	Quality Risk
13,33	68,00	Duplication Risk
59,50	19,38	Hotspot Risk
55,13	39,15	Structural Risk
48,91	76,46	High-TD Risk
45,43	47,27	Overall Project Risk
54,57	52,73	Global Health Score

Για να φανεί ποιοι δείκτες επηρέαζαν περισσότερο το τελικό αποτέλεσμα, υπολογίστηκε η πραγματική συνεισφορά κάθε διάστασης:

$$Contribution_i = Weight_i \cdot Risk_i$$

Η πολυπλοκότητα ήταν ο σημαντικότερος επιβαρυντικός παράγοντας και στα δύο έργα, προσθέτοντας περίπου 14,47 μονάδες στο συνολικό risk του Maven και περίπου 17,20 μονάδες σε αυτό του Sling. Αντίθετα, παρότι το Technical Debt Risk είχε το μεγαλύτερο βάρος (25%), η πραγματική συνεισφορά του ήταν μόνο 4,50 μονάδες στο Maven και 4,00 στο Sling, επειδή και τα δύο έργα είχαν χαμηλό technical debt ratio. Αυτό έδειξε πως η τελική αξιολόγηση δεν εξαρτάται μόνο από τα βάρη, αλλά επηρεάζεται εξίσου από τα κατώφλια που μετατρέπουν τις ακατέργαστες τιμές στην κοινή κλίμακα κινδύνου 0–100.

5.4.2 Σύγκριση Εναλλακτικών Τύπων Hotspot Score

Η αρχική μορφή του Hotspot Score ήταν προσθετική:

$$Hotspot Score_{additive} = 0.60 \cdot Git Activity Score + 0.40 \cdot Complexity Score$$

Η μορφή αυτή είναι εύκολη στην ερμηνεία, επιτρέπει όμως σε μια πολύ υψηλή τιμή δραστηριότητας να αντισταθμίσει μια πολύ χαμηλή τιμή πολυπλοκότητας. Για παράδειγμα, αν το Git Activity Score ήταν ίσο με 100 και το Complexity Score ίσο με 0, ο προσθετικός τύπος θα έδινε Hotspot Score ίσο με 60. Έτσι, ένα αρχείο μπορούσε να λάβει σχετικά υψηλό Hotspot Score μόνο επειδή άλλαζε συχνά, ακόμη και αν ήταν τεχνικά απλό. Για την αντιμετώπιση αυτού του προβλήματος εξετάστηκαν δύο πολλαπλασιαστικές μορφές:

$$Hotspot Score_{product} = \frac{(Git Activity Score \cdot Complexity Score)}{100} \text{ και}$$

$$Hotspot Score_{geometric} = \sqrt{Git Activity Score \cdot Complexity Score}$$

Πίνακας 5.3 Σύγκριση Εναλλακτικών Τύπων Hotspot Score

Τύπος Hotspot Score	Maven Archiver	Sling Whiteboard
Προσθετικός τύπος	19,38	59,50
Γεωμετρικός μέσος	17,54	56,06
Απλό γινόμενο	4,91	34,57

Το απλό γινόμενο αποδείχθηκε υπερβολικά αυστηρό και συμπίεζε σημαντικά την κλίμακα. Για παράδειγμα, όταν οι δύο επιμέρους βαθμολογίες είναι ίσες με 60:

$$Hotspot Score_{product} = \frac{(60 \cdot 60)}{100} = 36$$

Η τιμή 36 ανήκει στην κατηγορία Low, παρότι και οι δύο επιμέρους διαστάσεις έχουν τιμή 60. Ο γεωμετρικός μέσος διατήρησε την απαίτηση να συνυπάρχουν υψηλή δραστηριότητα και πολυπλοκότητα, χωρίς να μειώνει υπερβολικά το αποτέλεσμα. Όταν οι δύο βαθμολογίες είναι ίσες με 60:

$$Hotspot Score_{geometric} = \sqrt{60 \cdot 60} = 60$$

Παράλληλα, όταν μία από τις δύο βαθμολογίες είναι μηδενική, το τελικό Hotspot Score γίνεται επίσης μηδέν. Για τον λόγο αυτό επιλέχθηκε ο γεωμετρικός τύπος, καθώς η μορφή αυτή συμφωνεί καλύτερα με την έννοια του hotspot ως σημείου στο οποίο συνυπάρχουν αυξημένη δραστηριότητα αλλαγών και τεχνική δυσκολία.

5.4.3 Βαθμονόμηση του Βάρους του Hotspot Risk

Μετά την επιλογή του γεωμετρικού τύπου εξετάστηκε αν το αρχικό βάρος του Hotspot Risk, ίσο με 10%, αποτύπωνε επαρκώς τη σημασία των hotspots. Δοκιμάστηκαν τρεις διαμορφώσεις:

- Technical Debt Risk 25% και Hotspot Risk 10%
- Technical Debt Risk 20% και Hotspot Risk 15%
- Technical Debt Risk 15% και Hotspot Risk 20%

Τα υπόλοιπα βάρη παρέμειναν σταθερά.

Πίνακας 5.4 Σύγκριση Διαμόρφωσης

Διαμόρφωση	Maven Health	Sling Health
TD 25%, Hotspot 10%	52,92	54,91
TD 20%, Hotspot 15%	52,94	52,91
TD 15%, Hotspot 20%	52,96	50,91

Στο Maven Archiver η αλλαγή είχε μικρή επίδραση, επειδή το Technical Debt Risk και το γεωμετρικό Hotspot Risk είχαν σχετικά κοντινές τιμές στο συγκεκριμένο στάδιο. Στο Sling Whiteboard η αύξηση του βάρους των hotspots μείωσε περισσότερο το Global Health Score, επειδή το Hotspot Risk ήταν σημαντικά υψηλότερο από το Technical Debt Risk. Η διαφοροποίηση αυτή σχετίζεται και με τον τρόπο συμμετοχής της πολυπλοκότητας στον συνολικό δείκτη, καθώς συμμετέχει αυτόνομα στο Global Health Score με βάρος 20% και ταυτόχρονα αποτελεί μέρος του Hotspot Score. Επομένως, ένα πολύ υψηλό βάρος στα hotspots θα αύξανε έμμεσα και τη συνολική επίδραση της πολυπλοκότητας. Για τον λόγο

αυτό επιλέχθηκε η ενδιάμεση διαμόρφωση με Technical Debt Risk = 20% και Hotspot Risk = 15%. Το Hotspot Risk αποκτά έτσι μεγαλύτερη σημασία από την αρχική τιμή του 10%, χωρίς όμως να γίνεται εξίσου κυρίαρχο με την αυτόνομη διάσταση της πολυπλοκότητας.

5.4.4 Βαθμονόμηση των Complexity Thresholds

Το Complexity Project Risk προκύπτει από την πυκνότητα της κυκλωματικής πολυπλοκότητας και της γνωστικής πολυπλοκότητας ανά KLOC. Οι δύο επιμέρους κίνδυνοι υπολογίζονται ως:

$$Complexity Density Risk = \min \left(\left(\frac{Complexity per KLOC}{Complexity Threshold} \right) \cdot 100, 100 \right)$$

$$Cognitive Complexity Density Risk = \min \left(\left(\frac{Cognitive Complexity per KLOC}{Cognitive Complexity Threshold} \right) \cdot 100, 100 \right)$$

Ο συνολικός δείκτης υπολογίζεται ως:

$$Complexity Project Risk = 0.60 \cdot Complexity Density Risk + 0.40 \cdot Cognitive Complexity Density Risk$$

Στην αρχική διαμόρφωση χρησιμοποιούνταν τα σημεία κορεσμού:

$$Complexity Threshold = \frac{150}{KLOC}$$

$$Cognitive Complexity Threshold = \frac{100}{KLOC}$$

Με τις τιμές αυτές, το Complexity Risk ήταν 72,37 για το Maven Archiver και 86,01 για το Sling Whiteboard. Στο Sling, το cognitive complexity ήταν περίπου 99,36/KLOC και βρισκόταν σχεδόν στο σημείο κορεσμού των 100/KLOC. Για τη βαθμονόμηση εξετάστηκαν τέσσερα ζεύγη κατωφλίων:

Πίνακας 5.5 Δοκιμή τιμών για την επιλογή Complexity / Cognitive Threshold

Complexity / Cognitive Threshold	Maven Risk	Sling Risk
120 / 80	90,46	97,83
150 / 100	72,37	86,01
180 / 120	60,31	71,67
200 / 125	55,60	66,49

Τα κατώφλια 120/80 κρίθηκαν υπερβολικά αυστηρά, επειδή οδηγούσαν και τα δύο έργα κοντά στον κορεσμό. Αντίθετα, τα κατώφλια 200/125 οδηγούσαν σε αρκετά επιεική αξιολόγηση. Το ενδιάμεσο ζεύγος 180/120 διατήρησε το Sling ως το έργο με μεγαλύτερο κίνδυνο πολυπλοκότητας, χωρίς όμως να το κατατάσσει σχεδόν αυτόματα στην κατηγορία Critical. Για τον λόγο αυτό επιλέχθηκαν:

$$Complexity Threshold = \frac{180}{KLOC}$$

$$Cognitive Complexity Threshold = \frac{120}{KLOC}$$

Οι τιμές αυτές δεν αποτελούν καθολικά όρια καλού ή κακού λογισμικού. Χρησιμοποιούνται ως σημεία κορεσμού για τη μετατροπή των πυκνοτήτων στην κοινή κλίμακα κινδύνου 0–100.

5.4.5 Αναθεώρηση του Technical Debt Project Risk

Στην αρχική έκδοση του μοντέλου, το Technical Debt Project Risk υπολογιζόταν με γραμμικό τύπο και ένα σταθερό σημείο κορεσμού:

$$Technical\ Debt\ Risk = \min \left(\left(\frac{Technical\ Debt\ Ratio}{Threshold} \right) \cdot 100, 100 \right)$$

Κατά την αρχική βαθμονόμηση εξετάστηκαν κατώφλια ίσα με 3%, 5% και 10%.

Πίνακας 5.6 Technical Debt Project Risk

Μέθοδος υπολογισμού	Maven TD Risk	Sling TD Risk
Γραμμικός κορεσμός στο 3%	30,00	26,67
Γραμμικός κορεσμός στο 5%	18,00	16,00
Γραμμικός κορεσμός στο 10%	9,00	8,00
Τμηματική κλίμακα A–E	3,60	3,20

Αρχικά επιλέχθηκε το κατώφλι 5%, επειδή αντιστοιχεί στο προεπιλεγμένο όριο ανάμεσα στις αξιολογήσεις A και B του SonarQube. Ωστόσο, ο γραμμικός κορεσμός στο 5% παρουσίαζε ένα σημαντικό μειονέκτημα. Κάθε έργο με Technical Debt Ratio ίσο ή μεγαλύτερο από 5% λάμβανε Technical Debt Risk ίσο με 100. Επομένως, δεν διατηρούνταν καμία διαφορά ανάμεσα στις κατηγορίες B, C, D και E. Για την τελική έκδοση επιλέχθηκε η τμηματική αντιστοίχιση της κλίμακας συντηρησιμότητας A–E στην κλίμακα κινδύνου 0–100:

Πίνακας 5.7 Technical Debt Project Risk με βάση την κλίμακα συντηρησιμότητας

SonarQube Rating	Technical Debt Ratio	Περιοχή Risk
A	0% έως μικρότερο από 5%	0–20
B	5% έως μικρότερο από 10%	20–40
C	10% έως μικρότερο από 20%	40–60
D	20% έως μικρότερο από 50%	60–80
E	50% και άνω	80–100

Οι τιμές μεταξύ των ορίων υπολογίζονται με γραμμική παρεμβολή. Για Technical Debt Ratio 0,9%, το Maven Archiver λαμβάνει Technical Debt Project Risk 3,60. Αντίστοιχα, για Technical Debt Ratio 0,8% το Sling Whiteboard λαμβάνει Technical Debt Project Risk 3,20. Η τελική υλοποίηση χρησιμοποιεί τα σημεία 0%, 5%, 10%, 20%, 50% και 100% και υπολογίζει τις ενδιάμεσες τιμές με γραμμική παρεμβολή.

5.4.6 Κατώφλια Ποιότητας

Το Quality Project Risk βασίζεται στις πυκνότητες των οσμών κώδικα και των σφαλμάτων ανά KLOC:

$$Code\ Smells\ Risk = \min\left(\left(\frac{Code\ Smells\ per\ KLOC}{50}\right) \cdot 100, 100\right)$$

$$Bugs\ Risk = \min\left(\left(\frac{Bugs\ per\ KLOC}{5}\right) \cdot 100, 100\right) \quad \text{και}$$

$$Quality\ Project\ Risk = 0.60 \cdot Code\ Smells\ Risk + 0.40 \cdot Bugs\ Risk$$

Τα παραπάνω κατώφλια διατηρήθηκαν, επειδή έδωσαν αποτελέσματα που συμφωνούσαν με τα διαφορετικά προφίλ των δύο έργων.

Πίνακας 5.8 Κατώφλια Ποιότητας

Μετρική	Maven Archiver	Sling Whiteboard
Code Smells/KLOC	30,08	26,35
Bugs/KLOC	2,15	1,50
Quality Project Risk	53,29	43,62

Παρότι το Sling είχε περισσότερα προβλήματα σε απόλυτους αριθμούς, το Maven είχε μεγαλύτερη πυκνότητα προβλημάτων ως προς το μέγεθός του και επομένως μεγαλύτερο Quality Risk.

5.4.7 Κατώφλια Επανάληψης Κώδικα

Για τη μετατροπή του ποσοστού διπλότυπων γραμμών σε κλίμακα κινδύνου 0–100 εξετάστηκαν τρία εναλλακτικά σημεία κορεσμού, ίσα με 10%, 15% και 20%.

$$Duplication\ Project\ Risk = \min\left(\left(\frac{Duplicated\ Lines\ Density}{Duplication\ Threshold}\right) \cdot 100, 100\right)$$

Η επιλογή των συγκεκριμένων τιμών βασίστηκε τόσο στη σχετική βιβλιογραφία όσο και στη συμπεριφορά του δείκτη στα δύο έργα βαθμονόμησης. Στο πρακτικό μοντέλο συντηρησιμότητας των Heitlager, Kuipers και Visser, ποσοστά δυπλοτυπίας μεταξύ 10% και 20% εντάσσονται σε περιοχή χαμηλής συντηρησιμότητας, ενώ τιμές πάνω από 20% αντιστοιχούν στη δυσμενέστερη κατηγορία [49].

Πίνακας 5.9 Δοκιμαστικές τιμές για την επιλογή κατωφλιού επανάληψης κώδικα

Duplication Threshold	Maven Archiver	Sling Whiteboard
10%	100,00	20,00
15%	68,00	13,33
20%	51,00	10,00

Με κατώφλι 10%, το Maven Archiver οδηγείται άμεσα στη μέγιστη τιμή κινδύνου, καθώς η πυκνότητα διπλότυπου κώδικα είναι 10,2%. Η επιλογή αυτή κρίθηκε υπερβολικά αυστηρή, επειδή δεν διατηρεί διάκριση μεταξύ έργων που βρίσκονται λίγο πάνω από το 10% και έργων με πολύ υψηλότερη επανάληψη κώδικα. Αντίθετα, το κατώφλι 20% αποδίδει περισσότερο επιεική τιμή και περιορίζει τη συνεισφορά της διπλοτυπίας στον συνολικό κίνδυνο. Τελικά, το κατώφλι 15% επιλέχθηκε ως ενδιάμεση διαμόρφωση. Βρίσκεται μέσα στην προβληματική περιοχή 10–20% που αναφέρεται στη βιβλιογραφία και αποδίδει στο Maven υψηλή, αλλά όχι κορεσμένη, τιμή κινδύνου ίση με 68,00. Παράλληλα, το

χαμηλό ποσοστό διπλότυπων γραμμών του Sling Whiteboard οδηγεί σε αντίστοιχα χαμηλό δείκτη κινδύνου, ίσο με 13,33.

5.4.8 Υπολογισμός του High-TD Project Risk

Για τη βαθμονόμηση του High-TD Project Risk χρησιμοποιήθηκαν οι ορισμοί των συνόλων C και H της Ενότητας 3.3. Ειδικότερα, η αναλογία των αρχείων που έχουν χαρακτηριστεί ως High-TD στον κοινό πληθυσμό υπολογίστηκε ως:

$$Source\ HighTD\ Ratio = \frac{|H \cap C|}{|C|}$$

Η αναλογία αυτή δείχνει πόσο συχνά εμφανίζονται αρχεία που έχουν χαρακτηριστεί ως High-TD μέσα στον κοινό πληθυσμό και δεν αποτελεί μέτρο κάλυψης του προγνωστικού μοντέλου. Η αντίστοιχη συνιστώσα κινδύνου υπολογίζεται ως:

$$HighTD\ Ratio\ Risk = \min\left(\left(\frac{Source\ HighTD\ Ratio}{0.20}\right) \cdot 100, 100\right)$$

Στην αρχική διαμόρφωση του δείκτη, η έκταση του προβλήματος αποτυπωνόταν μέσω του Source High-TD Ratio, ενώ η ένταση εκφραζόταν μέσω της μέγιστης πιθανότητας που είχε επιστραφεί για κάποιο αρχείο του κοινού πληθυσμού που είχε χαρακτηριστεί ως High-TD. Η αρχική συνιστώσα πιθανότητας υπολογιζόταν ως:

$$HighTD\ Probability\ Risk = Max\ HighTD\ Probability \cdot 100$$

Η επιλογή της μέγιστης πιθανότητας είχε ως στόχο να διατηρεί την επίδραση της σοβαρότερης μεμονωμένης High-TD ένδειξης, ακόμη και όταν το πλήθος των High-TD αρχείων ήταν περιορισμένο. Η αποκλειστική χρήση της όμως, καθιστούσε ολόκληρη τη συνιστώσα έντασης εξαρτημένη από ένα μόνο αρχείο. Έτσι, μία μεμονωμένη πολύ υψηλή πιθανότητα μπορούσε να αυξήσει το αποτέλεσμα σε επίπεδο έργου, ακόμη και όταν οι υπόλοιπες High-TD πιθανότητες ήταν αισθητά χαμηλότερες. Για το λόγο αυτό, κρίθηκε ότι δεν ήταν πλήρως αντιπροσωπευτική για έναν δείκτη σε επίπεδο έργου. Παράλληλα, η χρήση του μέσου όρου όλων των πιθανοτήτων των High-TD αρχείων θα μπορούσε να οδηγήσει στο αντίθετο πρόβλημα, εξομαλύνοντας υπερβολικά τις ισχυρότερες προβλέψεις μέσω των χαμηλότερων τιμών. Για τον περιορισμό αυτής της ευαισθησίας, εξετάστηκε η χρήση συγκεντρωτικής τιμής από την ανώτερη περιοχή των πιθανοτήτων. Στην τελική διαμόρφωση επιλέχθηκε ο μέσος όρος του υψηλότερου 20% των πιθανοτήτων των High-TD αρχείων του κοινού πληθυσμού. Για τον υπολογισμό της συνιστώσας εφαρμόζεται ελάχιστο πλήθος τριών αρχείων, όταν υπάρχουν τουλάχιστον τρεις διαθέσιμες προβλέψεις. Αν M είναι το πλήθος των High-TD αρχείων του κοινού πληθυσμού:

$$M = |H \cap C|$$

Το πλήθος των πιθανοτήτων που συμμετέχουν στον υπολογισμό ορίζεται ως:

$$HighTD\ Probability\ Count = \min(M, \max(3, \text{ceil}(0.20 \cdot M)))$$

Αν k είναι το High-TD Probability Count, η συνιστώσα πιθανότητας υπολογίζεται ως:

$$HighTD \text{ Probability Risk} = \left(\frac{(p_1 + p_2 + \dots + p_k)}{k} \right) \cdot 100$$

Ο τελικός δείκτης είναι:

$$HighTD \text{ Project Risk} = 0.60 \cdot HighTD \text{ Ratio Risk} + 0.40 \cdot HighTD \text{ Probability Risk}$$

Πίνακας 5.10 Ανάλυση συνιστωσών High-TD Project Risk για τα 2 έργα

Μετρική	Maven Archiver	Sling Whiteboard
Μέγεθος κοινού πληθυσμού C	7	385
High-TD αρχεία στο $H \cap C$	1	12
Source High-TD Ratio	14,29%	3,12%
High-TD Ratio Risk	71,43	15,58
Μέγιστη High-TD πιθανότητα	0,84	0,96
High-TD Probability Count	1	3
Μέσος όρος υψηλότερου 20%	0,84	0,9167
High-TD Probability Risk	84,00	91,67
High-TD Project Risk	76,46	46,02

Μεγαλύτερο βάρος δίνεται στην αναλογία των High-TD αρχείων, επειδή αυτή αποτυπώνει πόσο εκτεταμένο είναι το πρόβλημα στον κοινό πληθυσμό. Η δεύτερη συνιστώσα λειτουργεί συμπληρωματικά και εκφράζει τη μέση ένταση της ισχυρότερης ομάδας High-TD προβλέψεων. Με αυτόν τον τρόπο, ένα μεμονωμένο αρχείο με πολύ υψηλή πιθανότητα εξακολουθεί να εμφανίζεται στο dashboard, αλλά δεν καθορίζει από μόνο του τη συνολική τιμή. Η εφαρμογή της προσέγγισης αυτής στα δύο έργα έδωσε διαφορετικά αποτελέσματα. Στο Maven Archiver υπήρχε μόνο ένα High-TD αρχείο στον κοινό πληθυσμό. Συνεπώς, χρησιμοποιήθηκε η μοναδική διαθέσιμη πιθανότητα 0,84 και δεν προέκυψε μεταβολή σε σχέση με τη χρήση της μέγιστης τιμής. Αντίθετα, στο Sling Whiteboard υπήρχαν 12 High-TD αρχεία. Το υψηλότερο 20% με ελάχιστο τρία αρχεία, οδήγησε στη χρήση των τριών υψηλότερων πιθανοτήτων, 0,96, 0,90 και 0,89, των οποίων ο μέσος όρος ήταν 0,9167. Το High-TD Probability Risk διαμορφώθηκε επομένως σε 91,67 και το High-TD Project Risk σε 46,02.

5.4.8.1 Έλεγχος του Κατωφλίου για το High-TD Ratio και της Στάθμισης

Για τον έλεγχο της συμπεριφοράς του κατωφλίου, εξετάστηκαν τα ποσοστά 10%, 20% και 30%. Για κάθε κατώφλι t , η συνιστώσα της αναλογίας υπολογίστηκε ως:

$$HighTD \text{ Ratio Risk}_t = \min \left(\left(\frac{Source \text{ HighTD Ratio}}{t} \right) \cdot 100, 100 \right)$$

Η τελική τιμή υπολογίστηκε αρχικά με την επιλεγμένη στάθμιση 0,60/0,40:

$$HighTD \text{ Project Risk} = 0.60 \cdot HighTD \text{ Ratio Risk} + 0.40 \cdot HighTD \text{ Probability Risk}$$

Πίνακας 5.11 High-TD Ratio Threshold και Στάθμισης

Ratio Threshold	Maven High-TD Ratio Risk	Maven High-TD Project Risk	Sling High-TD Ratio Risk	Sling High-TD Project Risk
10%	100,00	93,60	31,17	55,37
20%	71,43	76,46	15,58	46,02
30%	47,62	62,17	10,39	42,90

Με κατώφλι 10%, το Maven Archiver οδηγείται σε πλήρη κορεσμό της συνιστώσας αναλογίας, καθώς το 14,29% των αρχείων του κοινού πληθυσμού έχει χαρακτηριστεί ως High-TD. Η επιλογή αυτή αποδίδει High-TD Project Risk ίσο με 93,60 και κρίθηκε ιδιαίτερα αυστηρή, ειδικά για μικρά έργα στα οποία ένα μόνο High-TD αρχείο μπορεί να αντιπροσωπεύει μεγάλο ποσοστό του πληθυσμού. Αντίθετα, με κατώφλι 30%, το Maven λαμβάνει High-TD Project Risk ίσο με 62,17. Η επιλογή αυτή περιορίζει σημαντικά την επίδραση της αναλογίας και δίνει μεγαλύτερη σχετική σημασία στη συνιστώσα πιθανότητας, η οποία βασίζεται στον μέσο όρο του υψηλότερου 20% των High-TD πιθανοτήτων. Στο Sling Whiteboard η μεταβολή είναι μικρότερη, επειδή το ποσοστό των High-TD αρχείων είναι μόνο 3,12%.

Τελικά, επιλέχθηκε το κατώφλι 20% ως ενδιάμεση διαμόρφωση. Αποφεύγει τον άμεσο κορεσμό του Maven, ενώ εξακολουθεί να αποδίδει υψηλό κίνδυνο όταν ένα σημαντικό μέρος του κοινού πληθυσμού έχει χαρακτηριστεί ως High-TD. Παράλληλα, στο Sling διατηρεί χαμηλότερη τη συνιστώσα της αναλογίας, αντανakλώντας το περιορισμένο ποσοστό High-TD αρχείων. Εξετάστηκε επίσης η επίδραση της στάθμισης μεταξύ του High-TD Ratio Risk και του High-TD Probability Risk, διατηρώντας σταθερό το κατώφλι κορεσμού της αναλογίας στο 20%.

Πίνακας 5.12 High-TD Risk υπό διαφορετικά βάρη στάθμισης Ratio

Βάρη Ratio / Probability	Maven High-TD Project Risk	Sling High-TD Project Risk
0,50 / 0,50	77,71	53,63
0,60 / 0,40	76,46	46,02
0,70 / 0,30	75,20	38,41

Στο Maven Archiver οι διαφορές μεταξύ των τριών σταθμίσεων είναι περιορισμένες, επειδή τόσο το Ratio Risk όσο και το Probability Risk έχουν σχετικά υψηλές τιμές. Στο Sling Whiteboard, αντίθετα, η στάθμιση επηρεάζει περισσότερο το αποτέλεσμα. Το High-TD Probability Risk, το οποίο προκύπτει από τον μέσο όρο του υψηλότερου 20% των πιθανοτήτων, είναι επίσης υψηλό και ισούται με 91,67. Η ισότιμη στάθμιση 0,50/0,50 αυξάνει το High-TD Project Risk του Sling σε 53,63, επειδή δίνει μεγαλύτερη επιρροή στη συνιστώσα πιθανότητας. Αντίθετα, η στάθμιση 0,70/0,30 μειώνει το αποτέλεσμα σε 38,41 και δίνει σαφώς μεγαλύτερη προτεραιότητα στην έκταση του προβλήματος. Έτσι, η στάθμιση 0,60/0,40 διατηρήθηκε ως ενδιάμεση επιλογή. Η αναλογία λαμβάνει το μεγαλύτερο βάρος, επειδή σε επίπεδο έργου ενδιαφέρει κυρίως πόσο διαδεδομένο είναι το πρόβλημα, ενώ η συνιστώσα πιθανότητας εξακολουθεί να συμμετέχει ουσιαστικά, ώστε να αποτυπώνεται η μέση ένταση της ισχυρότερης ομάδας High-TD προβλέψεων. Η συγκεκριμένη σύγκριση δεν αποδεικνύει ότι το κατώφλι 20% και η στάθμιση 0,60/0,40 είναι βέλτιστα για κάθε έργο λογισμικού. Δείχνει όμως ότι αποτελούν ισορροπημένες επιλογές για τα δύο έργα βαθμονόμησης. Δεν οδηγούν σε υπερβολικά γρήγορο κορεσμό της συνιστώσας αναλογίας, ούτε σε υπερβολική εξομάλυνση των ισχυρότερων High-TD προβλέψεων. Οι κανόνες για τη διαχείριση μιας κενής επιτυχούς απόκρισης και της μη διαθεσιμότητας της διαδρομής High-TD του API παραμένουν ίδιοι με εκείνους που περιγράφονται στην Ενότητα 3.3 και δεν μεταβλήθηκαν κατά τη βαθμονόμηση. Η τελική υλοποίηση υπολογίζει τόσο το Source High-TD Ratio,

όσο και τη συνιστώσα πιθανότητας αποκλειστικά από τα High-TD αρχεία που ανήκουν στον κοινό πληθυσμό C.

5.4.9 Διόρθωση της Συγκέντρωσης των Project-Level Risks

Στην αρχική προσέγγιση, το Hotspot Project Risk και το Structural Project Risk υπολογίζονταν από τα αρχεία με τις υψηλότερες αντίστοιχες τιμές. Ο αριθμός τους υπολογιζόταν ως:

$$Risk\ Count = \min(50, \max(5, \text{ceil}(0.30 \cdot N)))$$

Το ανώτατο όριο των 50 αρχείων δημιούργησε πρόβλημα στα μεγαλύτερα έργα. Στο Maven χρησιμοποιούνταν πέντε από τα επτά αρχεία, δηλαδή περίπου το 71,43% του κοινού πληθυσμού. Στο Sling χρησιμοποιούνταν μόνο 50 από τα 385 αρχεία, δηλαδή περίπου το 12,99%. Το μεγάλο έργο αξιολογούνταν επομένως από πολύ μικρότερο ποσοστό του πληθυσμού του. Το όριο των 50 ήταν χρήσιμο για την παρουσίαση στο dashboard, αλλά όχι για τον μαθηματικό υπολογισμό του project-level risk. Στην τελική μέθοδο αφαιρέθηκε το ανώτατο όριο:

$$Project\ Risk\ Count = \min(N, \max(5, \text{ceil}(0.30 \cdot N)))$$

Αν ο κοινός πληθυσμός περιλαμβάνει έως πέντε αρχεία, χρησιμοποιούνται όλα. Για μεγαλύτερους πληθυσμούς χρησιμοποιείται το υψηλότερο 30%, με ελάχιστο πέντε αρχεία. Το ελάχιστο των πέντε εφαρμόζεται ώστε οι δείκτες μικρών έργων να μη βασίζονται σε πολύ περιορισμένο αριθμό παρατηρήσεων και να μην επηρεάζονται υπερβολικά από μία μόνο ακραία τιμή. Επομένως, στα μικρά έργα το ποσοστό μπορεί να είναι μεγαλύτερο από 30%, ενώ στα μεγαλύτερα έργα προσεγγίζει το επιλεγμένο ποσοστό. Στο Maven δεν προέκυψε μεταβολή, επειδή το ελάχιστο των πέντε εφαρμόζονταν και στις δύο περιπτώσεις. Στο Sling, αντίθετα, η χρήση μόνο των 50 υψηλότερων τιμών υπερεκτιμούσε τον κίνδυνο, καθώς εστίαζε αποκλειστικά σε ένα πολύ μικρό τμήμα του έργου. Με τη χρήση 116 αρχείων, οι δύο δείκτες αντιπροσωπεύουν καλύτερα το επιλεγμένο ποσοστό του κοινού πληθυσμού.

5.4.9.1 Σύγκριση Εναλλακτικών Ποσοστών Εστίασης

Μετά την αφαίρεση του ανώτατου ορίου των 50 αρχείων, εξετάστηκε η επίδραση του ποσοστού εστίασης στα Hotspot και Structural Project Risks. Συγκρίθηκαν ποσοστά 20%, 25%, 30% και 40%. Για κάθε ποσοστό p , ο αριθμός των επιλεγμένων αρχείων υπολογίστηκε ως:

$$k_p = \min(N, \max(5, \text{ceil}(p \cdot N))),$$

όπου $N = |C|$ είναι το πλήθος των αρχείων του κοινού πληθυσμού.

Αφού οι τιμές ταξινομηθούν σε φθίνουσα σειρά, οι δύο δείκτες υπολογίζονται ως:

$$Hotspot\ Project\ Risk_p = \left(\frac{1}{k_p}\right) \cdot \sum_{i=1}^{k_p} Hotspot\ Score_i$$

$$Structural\ Project\ Risk_p = \left(\frac{1}{k_p}\right) \cdot \sum_{i=1}^{k_p} Structural\ Risk\ Score_i$$

Πίνακας 5.13 Εναλλακτικά Ποσοστά Εστίασης

Ποσοστό εστίασης	Αρχεία Maven	Hotspot Risk Maven	Structural Risk Maven	Αρχεία Sling	Hotspot Risk Sling	Structural Risk Sling
20%	5	41,27	39,15	77	53,05	41,57
25%	5	41,27	39,15	97	47,87	36,16
30%	5	41,27	39,15	116	43,96	32,12
40%	5	41,27	39,15	154	38,01	26,18

Στο Maven Archiver δεν παρατηρήθηκε μεταβολή, επειδή ο κοινός πληθυσμός περιλάμβανε μόνο επτά αρχεία και σε όλες τις περιπτώσεις εφαρμόστηκε το ελάχιστο όριο των πέντε. Το όριο αυτό αποτρέπει τον υπολογισμό του κινδύνου από μία ή δύο μόνο ακραίες τιμές, αλλά στα πολύ μικρά έργα επηρεάζει περισσότερο το αποτέλεσμα από το επιλεγμένο ποσοστό. Στο Sling Whiteboard, η αύξηση του ποσοστού πρόσθεσε αρχεία με χαμηλότερους δείκτες και μείωσε τον αντίστοιχο μέσο όρο. Το 20% έδωσε μεγαλύτερη έμφαση στα αρχεία με τις υψηλότερες τιμές, ενώ το 40% κάλυψε μεγαλύτερο μέρος του έργου και εξομάλυνε περισσότερο τα αποτελέσματα. Τελικά επιλέχθηκε το 30% ως ενδιαμέση λύση. Στο Sling αντιστοιχεί σε 116 από τα 385 αρχεία και δίνει Hotspot Project Risk 43,96 και Structural Project Risk 32,12. Η επιλογή διατηρεί την εστίαση στις σημαντικότερες περιοχές κινδύνου, χωρίς να περιορίζει τον υπολογισμό σε πολύ μικρό μέρος του έργου.

5.4.10 Ανάλυση των Δομικών Βαρών

Το Structural Risk Score υπολογίζεται από τις κανονικοποιημένες μετρικές WMC, RFC, LCOM, MPC και CBO:

$$\text{Structural Risk Score} = 0.25 \cdot WMC + 0.25 \cdot RFC + 0.20 \cdot LCOM + 0.15 \cdot MPC + 0.15 \cdot CBO$$

Εξετάστηκαν δύο εναλλακτικές διαμορφώσεις. Στην πρώτη, το βάρος του CBO αυξήθηκε από 15% σε 20% και το βάρος του MPC μειώθηκε από 15% σε 10%. Στη δεύτερη, το βάρος του RFC αυξήθηκε από 25% σε 30% και το βάρος του MPC μειώθηκε από 15% σε 10%. Τα αποτελέσματα υπολογίστηκαν ξανά με τον τελικό κοινό πληθυσμό και τον τελικό κανόνα συγκέντρωσης, δηλαδή πέντε αρχεία για το Maven και 116 για το Sling.

Πίνακας 5.14 Υπολογισμός πίνακα με κοινό πληθυσμό και τελικό κανόνα συγκέντρωσης

Structural Scenario	Maven Risk	Structural Sling Structural Risk
Αρχική στάθμιση	39,15	32,12
Αυξημένο βάρος CBO	38,70	32,89
Αυξημένο βάρος RFC	38,72	32,67

Οι διαφορές ήταν περιορισμένες και δεν άλλαξαν ουσιαστικά την αξιολόγηση των δύο έργων. Δεν προέκυψε επομένως επαρκής λόγος αλλαγής των αρχικών βαρών. Η διατήρηση μιας σχετικά ισορροπημένης κατανομής κρίθηκε ασφαλέστερη από την υπερβολική ενίσχυση μίας μόνο μετρικής. Η σχετική προγνωστική ισχύς των αντικειμενοστραφών μετρικών μπορεί να διαφέρει ανά σύνολο δεδομένων.

5.4.11 Τελική Διαμόρφωση και Αποτελέσματα

Με βάση τις προηγούμενες δοκιμές, η τελική διαμόρφωση του μοντέλου ορίστηκε ως εξής:

- $Hotspot\ Score = \sqrt{Git\ Activity\ Score \cdot Complexity\ Score}$
- $Complexity\ Threshold = \frac{180}{KLOC}$
- $Cognitive\ Complexity\ Threshold = \frac{120}{KLOC}$
- $Quality\ Thresholds = \frac{50\ Code\ Smells}{KLOC}$ και $\frac{5\ Bugs}{KLOC}$
- $Duplication\ Threshold = 15$
- $HighTD\ Ratio\ Threshold = 20$
- $HighTD\ Probability\ Count = \min(M, \max(3, \text{ceil}(0,20 \cdot M)))$
- $Project\ Risk\ Count = \min(N, \max(5, \text{ceil}(0,30 \cdot N)))$, όπου M είναι το πλήθος των High-TD αρχείων του κοινού πληθυσμού και N το συνολικό πλήθος των αρχείων του κοινού πληθυσμού C.

Το Technical Debt Project Risk υπολογίζεται με την τμηματική αντιστοίχιση της κλίμακας Α–Ε. Τα τελικά βάρη του Global Health Score παρουσιάζονται στον ακόλουθο πίνακα.

Πίνακας 5.15 Πίνακας Βαρών Global Health Score

Project-level διάσταση	Βάρος
Technical Debt Project Risk	20%
Complexity Project Risk	20%
Quality Project Risk	15%
Duplication Project Risk	10%
Hotspot Project Risk	15%
Structural Project Risk	10%
High-TD Project Risk	10%
Σύνολο	100%

Ο συνολικός κίνδυνος υπολογίζεται ως:

$$Overall\ Project\ Risk = 0.20 \cdot Technical\ Debt\ Risk + 0.20 \cdot Complexity\ Risk + 0.15 \cdot Quality\ Risk + 0.10 \cdot Duplication\ Risk + 0.15 \cdot Hotspot\ Risk + 0.10 \cdot Structural\ Risk + 0.10 \cdot HighTD\ Risk$$

και

$$Global\ Health\ Score = 100 - Overall\ Project\ Risk$$

Όταν κάποια προαιρετική συνιστώσα δεν είναι διαθέσιμη, δεν αντικαθίσταται από μηδενικό κίνδυνο. Ο συνολικός κίνδυνος υπολογίζεται μόνο από τις διαθέσιμες συνιστώσες:

$$Overall\ Project\ Risk = \frac{\sum(Weight_i \cdot Risk_i)}{\sum(Weight_i)}$$

Με αυτόν τον τρόπο, η απουσία δεδομένων δεν βελτιώνει τεχνητά το Global Health Score. Η τελική υλοποίηση χρησιμοποιεί τα βάρη 20%, 20%, 15%, 10%, 15%, 10% και 10% και προσαρμόζει τα διαθέσιμα βάρη όταν κάποια συνιστώσα απουσιάζει. Οι αρχικές τιμές του Global Health Score και οι τιμές μετά τη βαθμονόμηση είναι οι εξής:

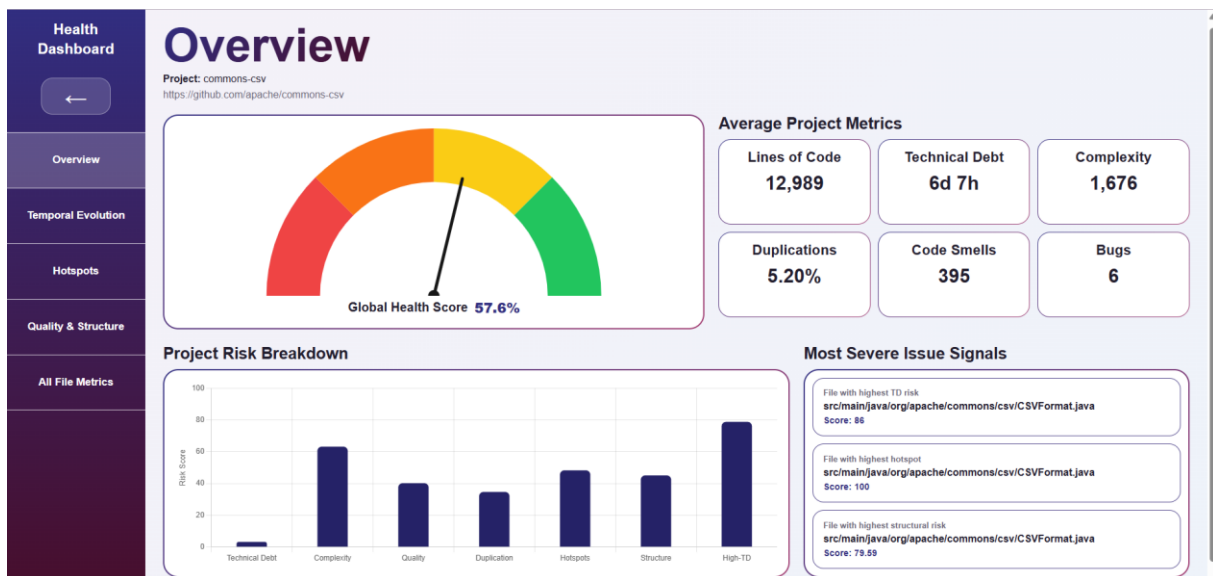
Πίνακας 5.16 Αρχικές και βαθμολογμένες τιμές Global Health Score

Project	Αρχικό Health	Τελικό Health	Μεταβολή
Maven Archiver	52,73	55,16	+2,43
Sling Whiteboard	54,57	62,74	+8,17

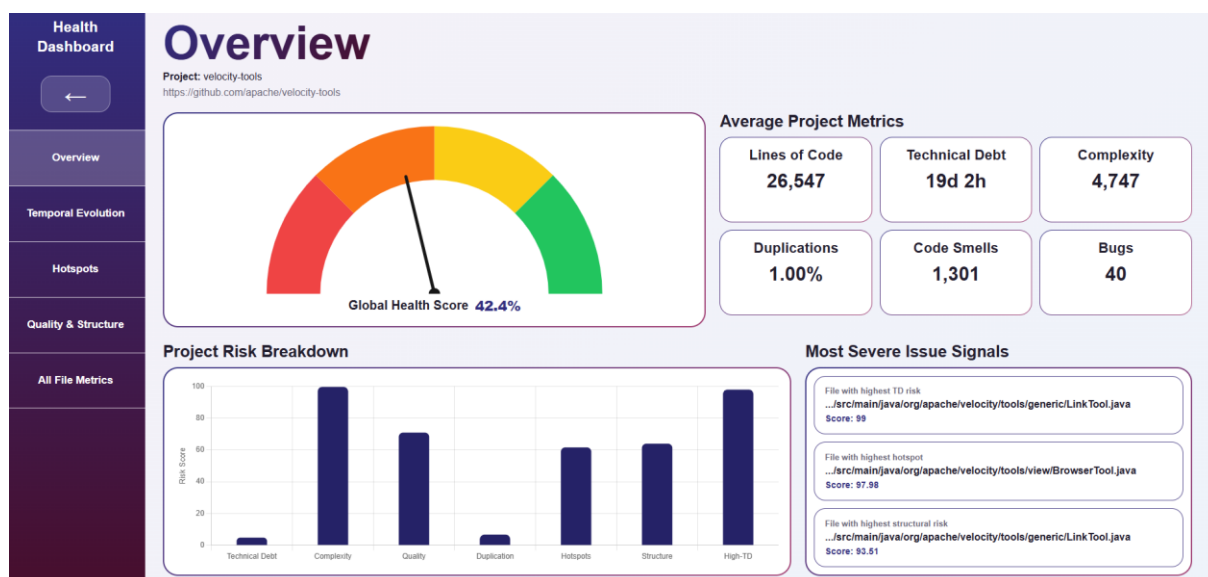
5.4.12 Ανάλυση Ευαισθησίας των Βαρών σε Επίπεδο Έργου

Για την αξιολόγηση της τελικής διαμόρφωσης επιλέχθηκαν δύο νέα αποθετήρια, το Velocity Tools και το Commons CSV, ώστε η εφαρμογή της μεθοδολογίας να εξεταστεί σε ανεξάρτητα έργα. Και τα δύο αποτελούν έργα ανοικτού κώδικα της Apache Software Foundation, γραμμένα κυρίως σε Java, αλλά με διαφορετικό μέγεθος και λειτουργικό αντικείμενο. Το Velocity Tools αποτελεί συλλογή εργαλείων και υποδομών για την ενσωμάτωση του Apache Velocity σε διαδικτυακές και μη εφαρμογές. Αντίθετα, το Commons CSV είναι μια μικρότερη και περισσότερο εστιασμένη βιβλιοθήκη για την ανάγνωση και εγγραφή αρχείων CSV. Η επιλογή τους επιτρέπει την εφαρμογή της ίδιας μεθοδολογίας σε έργα διαφορετικής κλίμακας και δομής, με κοινό τεχνολογικό υπόβαθρο, εκτός του συνόλου βαθμονόμησης. Έτσι, μπορεί να εξεταστεί αν τα αποτελέσματα και η συμπεριφορά των δεικτών παραμένουν συνεπή σε νέα δεδομένα.

Αρχικά, παρουσιάζονται τα συνοπτικά αποτελέσματα της ανάλυσης κάθε έργου με βάση την τελική διαμόρφωση του μοντέλου, όπως αυτά εμφανίζονται στο dashboard, στις Εικόνες 5.3 και 5.4 αντίστοιχα. Στη συνέχεια, εξετάζεται η ευαισθησία των αποτελεσμάτων σε μεταβολές των επιλεγμένων βαρών.



Εικόνα 5.3 - Οθόνη Overview του έργου commons-csv



Εικόνα 5.4 - Οθόνη Overview του έργου velocity-tools

Χρησιμοποιήθηκε η μέθοδος μεταβολής μίας παραμέτρου κάθε φορά. Κάθε βάρος αυξήθηκε και μειώθηκε κατά 10% και 20%, ενώ τα υπόλοιπα βάρη αναπροσαρμόστηκαν αναλογικά, ώστε να ισχύει:

$$\Sigma w_i = 1$$

Με επτά συνιστώσες και τέσσερις μεταβολές ανά συνιστώσα δημιουργήθηκαν 28 εναλλακτικά σενάρια για κάθε έργο, πέρα από το βασικό σενάριο αναφοράς.

Πίνακας 5.17 Αποτελέσματα ανάλυσης ευαισθησίας Global Health Score

Project	Baseline Global Health	Ελάχιστη τιμή	Μέγιστη τιμή	Μέγιστη απόκλιση
Velocity Tools	42,42	39,78	45,06	2,64
Commons CSV	57,60	55,64	59,56	1,96

Στο Velocity Tools η μεγαλύτερη μεταβολή προέκυψε από το βάρος του Technical Debt Project Risk. Ο συγκεκριμένος κίνδυνος είχε τιμή μόνο 4,80, σημαντικά χαμηλότερη από τις περισσότερες υπόλοιπες συνιστώσες. Η μείωση του βάρους του Technical Debt Risk κατά 20% μείωσε το Global Health Score σε 39,78, επειδή αυξήθηκε αναλογικά η επίδραση των υψηλότερων κινδύνων. Αντίθετα, η αύξηση του βάρους του κατά 20% αύξησε το Global Health Score σε 45,06. Αισθητή ήταν επίσης η επίδραση του Complexity Project Risk του οποίου η τιμή ήταν 99,60, δηλαδή ιδιαίτερα υψηλή. Όσον αφορά το Commons CSV, η μεγαλύτερη επίδραση προέκυψε επίσης από το βάρος του Technical Debt Project Risk. Η τιμή του συγκεκριμένου κινδύνου ήταν μόλις 3,20. Η μείωση του βάρους του κατά 20% μείωσε το Global Health Score σε 55,64, ενώ η αύξησή του κατά 20% αύξησε το Global Health Score σε 59,56. Η δεύτερη μεγαλύτερη επίδραση προέκυψε από το Complexity Project Risk, χωρίς όμως η μεταβολή του να ξεπεράσει περίπου τη μία μονάδα από το βασικό αποτέλεσμα.

Σε όλα τα εξεταζόμενα σενάρια, το Velocity Tools διατήρησε χαμηλότερο Global Health Score από το Commons CSV. Το εύρος των αποτελεσμάτων του Velocity Tools ήταν 39,78-45,06, ενώ του Commons CSV 55,64-59,56. Συνεπώς, μικρές και μέτριες μεταβολές των βαρών δεν άλλαξαν τη βασική εικόνα της μεταξύ τους σύγκρισης. Η μέγιστη απόκλιση ήταν 2,64 μονάδες στο Velocity Tools και 1,96 μονάδες στο Commons CSV. Το αποτέλεσμα δείχνει ότι το Global Health Score δεν παρουσιάζει ακραία αστάθεια απέναντι στις εξεταζόμενες αλλαγές των βαρών.

5.4.13 Ανάλυση Ευαισθησίας των Βαρών σε Επίπεδο Αρχείου

Για τον έλεγχο της σταθερότητας του Refactoring Priority Score εξετάστηκαν τα βάρη των διαστάσεων:

- Git Activity
- Complexity
- Structural Risk
- Quality Risk
- High-TD Risk

Κάθε βάρος αυξήθηκε και μειώθηκε κατά 10% και 20%. Σε κάθε σενάριο άλλαζε μόνο ένα βάρος, ενώ τα υπόλοιπα αναπροσαρμόζονταν αναλογικά ώστε το άθροισμά τους να παραμένει ίσο με 1. Δημιουργήθηκαν 20 εναλλακτικά σενάρια ανά έργο. Η ανάλυση εφαρμόστηκε στα 95 επιλέξιμα αρχεία του Velocity Tools και στα 12 επιλέξιμα αρχεία του Commons CSV. Η σταθερότητα αξιολογήθηκε μέσω:

- του συντελεστή συσχέτισης κατάταξης Spearman
- του ποσοστού κοινών αρχείων στις πρώτες πέντε θέσεις
- του ποσοστού κοινών αρχείων στις πρώτες δέκα θέσεις
- του αριθμού των αλλαγών στο τελικό επίπεδο κινδύνου

Πίνακας 5.18 Αποτελέσματα ανάλυσης ευαισθησίας και σταθερότητας για τα έργα Velocity Tools και Commons CSV

Project	Αρχεία	Σενάρια	Spearman	Top-5 overlap	Top-10 overlap	Μέγιστες αλλαγές risk level
Velocity Tools	95	20	0,9975	60%	90%	0
Commons CSV	12	20	0,9930	100%	100%	0

Στο Velocity Tools η χαμηλότερη τιμή Spearman ήταν 0,9975 και προέκυψε όταν το βάρος του Git Activity αυξήθηκε κατά 20%. Η τιμή αυτή δείχνει ότι η συνολική σειρά των 95 αρχείων παρέμεινε πολύ κοντά στη βασική κατάταξη. Στο ίδιο σενάριο, 4 από τα 5 πρώτα αρχεία παρέμειναν ίδια. Η μεγαλύτερη μεταβολή στην πρώτη πεντάδα εμφανίστηκε σε διαφορετικό σενάριο, όταν το βάρος του Quality Risk μειώθηκε κατά 20%. Τότε το Top-5 overlap μειώθηκε στο 60%, δηλαδή 3 από τα 5 αρχικά κορυφαία αρχεία παρέμειναν στην πρώτη πεντάδα. Το Top-10 overlap δεν έπεσε κάτω από 90% σε κανένα από τα 20 σενάρια. Αντίστοιχα, στο Commons CSV η χαμηλότερη τιμή Spearman ήταν 0,9930. Παρότι εμφανίστηκαν μικρές μεταβολές στη μεταξύ τους σειρά, τα ίδια 5 αρχεία παρέμειναν στις πρώτες 5 θέσεις και τα ίδια 10 στις πρώτες 10 θέσεις σε όλα τα σενάρια. Επομένως, τόσο το Top-5 όσο και το Top-10 overlap παρέμειναν στο 100%. Η ελαφρώς χαμηλότερη ελάχιστη τιμή του Spearman σε σύγκριση με το Velocity Tools πρέπει να εξετάζεται και σε συνδυασμό με το μικρό πλήθος των 12 αρχείων, καθώς ακόμη και λίγες αλλαγές στη σειρά τους έχουν μεγαλύτερη επίδραση στον συντελεστή. Συμπληρωματικά, στο Velocity Tools οι μεταβολές του Git Activity είχαν τη μεγαλύτερη συνολική επίδραση στις αριθμητικές τιμές του Refactoring Priority Score, ενώ αισθητή ήταν επίσης η επίδραση του Quality Risk και του Structural Risk. Στην περίπτωση του Commons CSV, οι μεγαλύτερες αριθμητικές μεταβολές εμφανίστηκαν κυρίως κατά τη μεταβολή του Quality Risk και του Git Activity, χωρίς όμως να αλλάζει η σύνθεση των πρώτων 5 ή 10 αρχείων. Η μεταβολή του High-TD Risk είχε μικρότερη επίδραση και στα δύο έργα, καθώς θετική πρόβλεψη High-TD υπήρχε μόνο για μέρος του

κοινού πληθυσμού. Σε κανένα από τα εξεταζόμενα σενάρια δεν προέκυψε αλλαγή στο τελικό επίπεδο κινδύνου των αρχείων.

Συνολικά, η ανάλυση δείχνει ότι η συνολική κατάταξη παραμένει σε μεγάλο βαθμό σταθερή απέναντι σε μικρές και μέτριες αλλαγές των βαρών, χωρίς αυτό να σημαίνει ότι οι επιμέρους θέσεις παραμένουν απόλυτα αμετάβλητες. Η μεγαλύτερη ευαισθησία εμφανίζεται στις πιο υψηλές θέσεις του Velocity Tools, όπου η μεταβολή του Quality Risk μπορεί να αλλάξει τη σύνθεση της πρώτης πεντάδας.

Κεφάλαιο 6ο: Συμπεράσματα και Μελλοντικές επεκτάσεις

6.1 Γενική Συζήτηση και Ερμηνεία Ευρημάτων

Η παρούσα εργασία έδειξε ότι η ενοποίηση στατικών, δομικών και ιστορικών δεδομένων μπορεί να οδηγήσει σε μια συνεκτική και ερμηνεύσιμη αποτίμηση της τεχνικής κατάστασης ενός έργου λογισμικού. Η εφαρμογή παρέχει τόσο μια συνοπτική εικόνα μέσω του Global Health Score και των επιμέρους δεικτών, όσο και μια κατάταξη των αρχείων μέσω του Refactoring Priority Score, επιτρέποντας στον χρήστη να κινηθεί από τη συνολική εικόνα προς τα συγκεκριμένα αρχεία που χρειάζονται διερεύνηση, διατηρώντας παράλληλα τη σύνδεση με τις αρχικές μετρικές. Πρόκειται όμως για ένα μέσο υποστήριξης και όχι για μηχανισμό αυτόματης λήψης αποφάσεων. Οι τελικές αποφάσεις απαιτούν γνώση της λειτουργίας του συστήματος, των επιχειρησιακών απαιτήσεων και του κόστους κάθε παρέμβασης.

Ένα σημαντικό συμπέρασμα αφορά τον τρόπο ερμηνείας των σύνθετων δεικτών. Η συνολική τιμή του Global Health Score δεν αρκεί από μόνη της για την κατανόηση της τεχνικής κατάστασης ενός έργου. Για τον λόγο αυτό, πρέπει να εξετάζεται μαζί με τις επιμέρους συνιστώσες του. Η εφαρμογή της τελικής διαμόρφωσης στα δύο έργα βαθμονόμησης ανέδειξε αυτή τη διαφοροποίηση. Το Maven Archiver επηρεάστηκε περισσότερο από το High-TD Risk, την επανάληψη κώδικα και τα προβλήματα ποιότητας, ενώ στο Sling Whiteboard κυριότερη πηγή επιβάρυνσης ήταν η πολυπλοκότητα. Επομένως, συμπεραίνεται πως παρόμοιες συνολικές βαθμολογίες μπορούν να προκύπτουν από διαφορετικά προφίλ τεχνικής επιβάρυνσης.

Τέλος, η ανάλυση ευαισθησίας έδειξε ότι η βασική εικόνα των αποτελεσμάτων παραμένει σε μεγάλο βαθμό σταθερή απέναντι σε μικρές και μέτριες μεταβολές των βαρών, τόσο σε επίπεδο έργου όσο και σε επίπεδο αρχείων. Το αποτέλεσμα αυτό υποστηρίζει τη σταθερότητα της μεθοδολογίας, χωρίς όμως να αποδεικνύει ότι τα συγκεκριμένα βάρη και κατώφλια αποτελούν τη μοναδική ή τη βέλτιστη επιλογή για κάθε έργο λογισμικού.

6.2 Απειλές Εγκυρότητας

Ένας βασικός περιορισμός είναι ο μικρός αριθμός έργων που χρησιμοποιήθηκαν κατά την αξιολόγηση. Η εφαρμογή της μεθοδολογίας σε δύο έργα δεν επιτρέπει στατιστική γενίκευση των αποτελεσμάτων ούτε επιβεβαιώνει ότι τα επιλεγμένα κατώφλια και βάρη θα λειτουργούν με τον ίδιο τρόπο σε έργα διαφορετικού μεγέθους, γλώσσας ή οργανωτικού περιβάλλοντος. Για τον λόγο αυτό, οι συγκεκριμένες παράμετροι πρέπει να αντιμετωπίζονται ως ενδεικτικές και όχι ως καθολικές επιλογές για κάθε έργο λογισμικού.

Πέρα από το μέγεθος του δείγματος, τίθεται και το ζήτημα της εγκυρότητας του ίδιου του μοντέλου αξιολόγησης. Οι σύνθετοι δείκτες που χρησιμοποιούνται στην εργασία δεν αποτελούν άμεσες μετρήσεις της συνολικής ποιότητας ή του πραγματικού κόστους συντήρησης, αλλά προκύπτουν από την κανονικοποίηση και την ενοποίηση των επιλεγμένων μετρικών. Κατά συνέπεια, επηρεάζονται από τις επιλογές των μαθηματικών τύπων, των κατωφλίων, των βαρών και των πληθυσμών που συμμετέχουν στους υπολογισμούς. Ειδικότερα, οι δείκτες επιπέδου αρχείου βασίζονται σε κανονικοποίηση P95 που υπολογίζεται ξεχωριστά για κάθε αποθετήριο, οπότε είναι κατάλληλοι κυρίως για την εσωτερική κατάταξη των αρχείων του ίδιου έργου. Η ίδια τιμή Refactoring Priority Score ή Structural Risk Score σε δύο διαφορετικά έργα για παράδειγμα, δεν σημαίνει απαραίτητα πως έχουν την ίδια τεχνική επιβάρυνση. Παρομοίως, το Global Health Score έχει υβριδικό χαρακτήρα. Ο λόγος είναι πως οι διαστάσεις Technical Debt, Complexity, Quality και Duplication βασίζονται σε σταθερά κατώφλια ή πυκνότητες, αλλά τα Hotspot και Structural Project Risks προέρχονται από δείκτες που είναι σχετικοί

ως προς το εκάστοτε αποθετήριο. Συνεπώς, η σύγκριση διαφορετικών έργων είναι ενδεικτική και πρέπει να συνοδεύεται από την εξέταση των επιμέρους κινδύνων και των αρχικών μετρικών.

Περιορισμός υπάρχει επίσης στις μετρικές ιστορικού αλλαγών. Το Git Activity Score χρησιμοποιεί το συνολικό `commit_count` και το συνολικό `churn` από το διαθέσιμο ιστορικό του αποθετηρίου. Το γεγονός αυτό σημαίνει πως μεγαλύτερα ή παλαιότερα αρχεία μπορεί να έχουν συγκεντρώσει περισσότερες αλλαγές λόγω μεγέθους ή ηλικίας και όχι απαραίτητα επειδή παρουσιάζουν δυσανάλογα αυξημένο τεχνικό κίνδυνο. Η πληροφορία αυτή δεν ακυρώνει τη χρησιμότητα της δραστηριότητας αλλαγών για την προτεραιοποίηση, αλλά πρέπει να λαμβάνεται υπόψη κατά την ερμηνεία του αποτελέσματος.

Μια ακόμη πηγή αβεβαιότητας αφορά την κάλυψη και την ποιότητα των δεδομένων. Η αξιοπιστία των αποτελεσμάτων εξαρτάται από την κάλυψη και τη σωστή αντιστοίχιση των δεδομένων που προέρχονται από το SonarQube, το Git και το AllinCode. Ο κοινός πληθυσμός $C = G \cap S \cap A$ εξασφαλίζει ότι τα αρχεία που συμμετέχουν στους ολοκληρωμένους υπολογισμούς διαθέτουν πληροφορίες και από τις τρεις βασικές πηγές. Η επιλογή αυτή αποτρέπει τη δημιουργία παραπλανητικών αποτελεσμάτων λόγω ελλιπών δεδομένων, μπορεί όμως να περιορίσει τον αριθμό των αρχείων που τελικά αξιολογούνται. Η κάλυψη επηρεάζεται επίσης από τις γλώσσες που υποστηρίζει κάθε εργαλείο, τη διαθεσιμότητα συγκεκριμένων μετρικών και τη συνέπεια των μονοπατιών αρχείων μεταξύ των πηγών. Για τον λόγο αυτό, η εφαρμογή διατηρεί ξεχωριστές ενδείξεις διαθεσιμότητας και επιλεξιμότητας και δεν μετατρέπει τις απύουσες τιμές σε τεχνητά μηδενικά αποτελέσματα.

Τέλος, ζήτημα αποτελεί και η αναπαραγωγικότητα των αποτελεσμάτων, καθώς τα αποθετήρια και τα εξωτερικά εργαλεία μπορεί να μεταβάλλονται με την πάροδο του χρόνου. Η τρέχουσα υλοποίηση δεν αποθηκεύει όλα τα στοιχεία που απαιτούνται για την ακριβή επανάληψη μιας ανάλυσης. Ως αποτέλεσμα, μια μελλοντική εκτέλεση στο ίδιο αποθετήριο μπορεί να αφορά διαφορετικό στιγμιότυπο του κώδικα ή διαφορετική διαμόρφωση των εργαλείων.

6.3 Προτάσεις για Μελλοντική Βελτίωση

6.3.1 Εξέλιξη της Μεθοδολογίας

Η σημαντικότερη μελλοντική κατεύθυνση είναι η αξιολόγηση της μεθοδολογίας με βάση περισσότερα και διαφορετικά έργα. Ένα μεγαλύτερο δείγμα, με διαφορετικές γλώσσες, ηλικίες, μεγέθη, επίπεδα δραστηριότητας και οργανωτικά περιβάλλοντα, θα επέτρεπε να εξεταστεί καλύτερα η δυνατότητα γενίκευσης της μεθοδολογίας. Η ισχυρότερη μορφή επικύρωσης όμως θα προέκυπτε από τη σύγκριση των δεικτών με ένα ανεξάρτητο σύνολο πραγματικών δεδομένων αναφοράς. Για παράδειγμα, μια διαχρονική μελέτη θα επέτρεπε να εξετάσει αν τα αρχεία που κατατάσσονται σήμερα σε υψηλή προτεραιότητα, είναι πράγματι εκείνα που παρουσιάζουν στο μέλλον περισσότερες διορθώσεις, αυξημένο κόστος συντήρησης ή συχνότερες παρεμβάσεις. Επιπλέον, μια επόμενη έκδοση της μεθοδολογίας μπορεί να εξετάσει πιο αναλυτικά την επίδραση του μεγέθους των αρχείων. Η χρήση μετρικών που λαμβάνουν υπόψη το μέγεθος θα βοηθούσε στην καλύτερη ερμηνεία της κατάταξης, αφού τότε θα φαινόταν αν η υψηλή θέση ενός αρχείου οφείλεται κυρίως στο μέγεθός του ή σε άλλους παράγοντες. Παράλληλα, η συνολική μεταβολή κώδικα θα μπορεί να συμπληρωθεί με τη σχετική μεταβολή κώδικα (*relative churn*), δηλαδή τη μεταβολή σε σχέση με το μέγεθος του αρχείου. Θα μπορούσε ακόμη να υπάρχει η επιλογή η δραστηριότητα του αρχείου να εξετάζεται για ένα συγκεκριμένο χρονικό διάστημα, ώστε η ηλικία του να μην επηρεάζει υπερβολικά την κατάταξη.

Επίσης, χρειάζεται πιο συστηματικός έλεγχος του μη αντισταθμιστικού κανόνα. Μια καλή μελλοντική προσθήκη θα ήταν να καταγράφονται οι περιπτώσεις στις οποίες μια ακραία τιμή αλλάζει την τελική

κατηγορία κινδύνου. Με αυτόν τον τρόπο θα εξεταζόταν εάν ο κανόνας αποτρέπει πράγματι την απόκρυψη σημαντικού κινδύνου ή αν γίνεται υπερβολικά αυστηρός. Τέλος, για την πιο αξιόπιστη σύγκριση μεταξύ διαφορετικών έργων, χρειάζεται βαθμονόμηση των βαρών σε μεγαλύτερο και πιο ετερογενές σύνολο έργων. Αντίθετα, για τη χρονική σύγκριση του ίδιου έργου, θα ήταν προτιμότερο να διατηρείται μια σταθερή βάση αναφοράς ανά αποθετήριο. Έτσι, οι μεταβολές των Hotspot και των δομικών δεικτών θα συγκρίνονται με την ίδια αρχική κατάσταση.

Τέλος, μια ακόμη κατεύθυνση αφορά την αναπαραγωγιμότητα και την κάλυψη των δεδομένων. Κάθε μελλοντικό `analysis_run` θα ήταν χρήσιμο να αποθηκεύει το `commit SHA`, τις εκδόσεις των εργαλείων που χρησιμοποιούνται, καθώς και τις βασικές ρυθμίσεις που επηρεάζουν την ανάλυση. Με αυτό τον τρόπο, τα αποτελέσματα θα συνδέονται με συγκεκριμένο στιγμιότυπο του κώδικα και θα μπορούν να επαναληφθούν με μεγαλύτερη ακρίβεια.

6.3.2 Τεχνική Εξέλιξη της Εφαρμογής

Η τρέχουσα υλοποίηση ακολουθεί μοντέλο ανάλυσης κατόπιν αιτήματος (on-demand), όπου η ροή SonarQube-Git-AllinCode-Python ενεργοποιείται από τον χρήστη και τα αποτελέσματα αποθηκεύονται ως στιγμιότυπο. Μια μελλοντική επέκταση θα μπορούσε να αξιοποιεί μηχανισμούς αυτόματης ειδοποίησης GitHub (webhooks), ώστε η ανάλυση να ενεργοποιείται μετά από κάθε αποστολή νέας υποβολής αλλαγών (push). Παράλληλα, η χρήση WebSocket, δηλαδή μηχανισμού άμεσης αμφίδρομης επικοινωνίας, θα μπορούσε να επιτρέπει την αυτόματη ενημέρωση του dashboard χωρίς νέα ενέργεια του χρήστη. Η εκτέλεση των αναλύσεων θα ήταν επίσης καλό να μεταφερθεί σε εργασίες παρασκήνιου (background jobs), με την κατάστασή τους να αποθηκεύεται στη βάση δεδομένων. Έτσι, ο χρήστης θα έχει τη δυνατότητα να απομακρύνεται από τη σελίδα και να επιστρέφει αργότερα χωρίς να χάνεται η κατάσταση της εκτέλεσης.

Επιπλέον, η αξιοποίηση των διαδοχικών αναλύσεων μπορεί να υποστηρίξει τη σύγκριση διαφορετικών εκδόσεων του έργου. Το dashboard θα μπορούσε να παρουσιάζει μεταβολές ανά στοιχείο λογισμικού, νέα ή επιλυμένα hotspots και αλλαγές στην κατάταξη των αρχείων. Οι ακατέργαστες χρονικές τιμές που αποθηκεύονται ήδη, όπως οι ημέρες τεχνικού χρέους και το κόστος του AllinCode, παρέχουν μια βάση για αυτή την επέκταση. Μελλοντικά, τα αποτελέσματα θα μπορούσαν επίσης να συνδεθούν με υποβολές αλλαγών (commits), αιτήματα ενσωμάτωσης αλλαγών (pull requests) ή καταγεγραμμένα ζητήματα (issues). Με αυτή την προσθήκη, ο χρήστης θα έχει την επιλογή να εξετάζει εκτός από τη μεταβολή ενός δείκτη και τις αλλαγές που σχετίζονται με αυτή.

Όσον αφορά την ασφάλεια, η εφαρμογή δέχεται αυθαίρετο URL δημόσιου αποθετηρίου GitHub από το οποίο προσδιορίζει το όνομα του αποθετηρίου και τον ιδιοκτήτη, που στη συνέχεια χρησιμοποιούνται σε εξωτερικές εντολές κελύφους (shell). Η απευθείας χρήση μη ελεγμένων τιμών μπορεί να δημιουργήσει κίνδυνο έγχυσης εντολών (command injection). Για τον λόγο αυτό, μια μελλοντική έκδοση θα πρέπει να εφαρμόζει αυστηρότερη επικύρωση των URL και ασφαλέστερη διαχείριση των παραμέτρων που χρησιμοποιούνται στις εξωτερικές εντολές.

Συνολικά, η παρούσα εργασία καταλήγει σε μια λειτουργική και ερμηνεύσιμη πρώτη έκδοση μιας εφαρμογής με σκοπό την αξιολόγηση της τεχνικής κατάστασης ενός έργου και την προτεραιοποίηση των αρχείων του. Τα αποτελέσματα της βαθμονόμησης και της ανάλυσης ευαισθησίας δείχνουν ότι η μεθοδολογία παρουσιάζει αρκετά σταθερή συμπεριφορά, χωρίς όμως να είναι από μόνη της μια λύση για οριστικές αποφάσεις. Η αξιολόγηση με βάση περισσότερα έργα και η χρήση πραγματικών δεδομένων συντήρησης αποτελούν βασικά επόμενα βήματα για την περαιτέρω εξέλιξη της εφαρμογής.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] E. M. Arvanitou, A. Ampatzoglou, S. Bibi, A. Chatzigeorgiou, and I. Stamelos, “Monitoring Technical Debt in an Industrial Setting,” 23rd International Conference on the Evaluation and Assessment in Software Engineering (EASE' 19), ACM, Copenhagen, Denmark, 14-17 April 2019.
- [2] A. Ampatzoglou, A. Chatzigeorgiou, E. M. Arvanitou and S. Bibi, “SDK4ED: A Platform for Technical Debt Management,” *Software: Practice and Experience*, vol. 52, no. 8, pp. 1879–1902, 2022.
- [3] Ar. Ampatzoglou, Ap. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun and K. Systs, “The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study”, 8th International Workshop on Managing Technical Debt (MTD' 16), IEEE, USA, Oct. 2016.
- [4] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management,” *Journal of Systems and Software*, vol. 101, pp. 193-220, March 2015.
- [5] Ar. Ampatzoglou, Ap. Ampatzoglou, A. Chatzigeorgiou and P. Avgeriou, “The financial aspect of managing technical debt: A systematic literature review”, *Information and Software Technology*, Elsevier, vol. 64, pp. 52–73, Aug. 2015.
- [6] R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics," 2012 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, 2012, pp. 987-996
- [7] D. Zhang, S. Han, Y. Dang, J. -G. Lou, H. Zhang and T. Xie, "Software Analytics in Practice," in *IEEE Software*, vol. 30, no. 5, pp. 30-37, Sept.-Oct. 2013
- [8] T. Besker, A. Martini and J. Bosch, "Technical Debt Triage in Backlog Management," 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), Montreal, QC, Canada, 2019, pp. 13-22
- [9] J. P. Biazotto, D. Feitosa, P. Avgeriou and E. Y. Nakagawa, “Understanding Practitioners’ Reasoning and Requirements for Efficient Tool Support in Technical Debt Management,” *Empirical Software Engineering*, vol. 30, 2025.
- [10] W. Cunningham, “The WyCash Portfolio Management System,” *Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, New York, NY, USA, 1992, pp. 29–30.
- [11] P. Kruchten and I. Ozkaya, *Managing Technical Debt: Reducing Friction in Software Development*. Addison-Wesley Professional, 2019.
- [12] ISO/IEC 25010:2023, *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model*, 2nd ed., 2023.
- [13] M. M. Lehman, “Laws of Software Evolution Revisited,” *European Workshop on Software Process Technology*, 1996, pp. 108–124.
- [14] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.
- [15] P. C. Avgeriou et al., "An Overview and Comparison of Technical Debt Measurement Tools," in *IEEE Software*, vol. 38, no. 3, pp. 61-71, May-June 2021

- [16] M. T. Baldassarre, V. Lenarduzzi, S. Romano and N. Saarimäki, "On the Diffuseness of Technical Debt Items and Accuracy of Remediation Time When Using SonarQube," *Information and Software Technology*, vol. 128, 2020.
- [17] R. Moser, W. Pedrycz and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," 2008 ACM/IEEE 30th International Conference on Software Engineering, Leipzig, Germany, 2008, pp. 181-190
- [18] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," *Proceedings. 27th International Conference on Software Engineering*, 2005. ICSE 2005., St. Louis, MO, USA, 2005, pp. 284-292
- [19] CodeScene, "Behavioral Code Analysis in Practice," *CodeScene*. [Online]. Available: https://codescene.com/hubfs/web_docs/Behavioral-code-analysis-in-practice.pdf
- [20] M. V. Kosti, A. Ampatzoglou, A. Chatzigeorgiou, G. Pallas, I. Stamelos and L. Angelis, "Technical Debt Principal Assessment Through Structural Metrics," 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, Austria, 2017, pp. 329-333
- [21] D. Tsoukalas, N. Mittas, E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou and D. Kehagias, "Local and Global Explainability for Technical Debt Identification," in *IEEE Transactions on Software Engineering*, vol. 50, no. 8, pp. 2110-2123, Aug. 2024
- [22] M. Riaz, E. Mendes and E. Tempero, "A systematic review of software maintainability prediction and metrics," 2009 3rd International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, FL, USA, 2009, pp. 367-377
- [23] Y. Li, M. Soliman, P. Avgeriou and M. Van Ittersum, "DebtViz: A Tool for Identifying, Measuring, Visualizing, and Monitoring Self-Admitted Technical Debt," 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bogotá, Colombia, 2023, pp. 558-562
- [24] A. Trendowicz, M. Kläs, C. Lampasona, J. Münch, C. Körner and M. Saft, "Model-based Product Quality Evaluation with Multi-Criteria Decision Analysis," *International Conference on Software Process and Product Measurement (IWSM/MetriKon/Mensura 2010)*, Stuttgart, Germany, 2010, pp. 3–20.
- [25] F. Basciani, D. Di Pompeo, J. Di Rocco and A. Pierantonio, "A customizable approach to assess software quality through Multi-Criteria Decision Making," 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), L'Aquila, Italy, 2023, pp. 264-271
- [26] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini and F. Arcelli Fontana, "A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools," *Journal of Systems and Software*, vol. 171, 2021.
- [27] R. Shatnawi, "A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems," in *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 216-225, March-April 2010
- [28] F. Rahman and P. Devanbu, "How, and why, process metrics are better," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 2013, pp. 432-441

- [29] SonarSource, “Understanding measures and metrics,” *SonarQube Server 2026.1 LTA Documentation*, 2026. [Online]. Available: <https://docs.sonarsource.com/sonarqube-server/2026.1/user-guide/code-metrics/metrics-definition>.
- [30] G. A. Campbell, “Cognitive Complexity: A new way of measuring understandability,” Version 1.7, SonarSource, 2023. [Online]. Available: <https://www.sonarsource.com/docs/CognitiveComplexity.pdf>
- [31] L. Lavazza, A. Z. Abualkishik, G. Liu and S. Morasca, “An empirical evaluation of the ‘Cognitive Complexity’ measure as a predictor of code understandability,” *Journal of Systems and Software*, vol. 197, 2023.
- [32] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," in *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, June 1994
- [33] V. R. Basili, L. C. Briand and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," in *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-761, Oct. 1996
- [34] T. Gyimothy, R. Ferenc and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," in *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897-910, Oct. 2005
- [35] G. Lacerda, F. Petrillo, M. Pimenta and Y.-G. Guéhéneuc, “Code Smells and Refactoring: A Tertiary Systematic Review of Challenges and Observations,” *Journal of Systems and Software*, vol. 167, Art. no. 110610, 2020.
- [36] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto and A. De Lucia, "[Journal First] On the Diffuseness and the Impact on Maintainability of Code Smells: A Large Scale Empirical Investigation," 2018 *IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, Gothenburg, Sweden, 2018, pp. 482-482
- [37] D. I. K. Sjøberg, A. Yamashita, B. C. D. Anda, A. Mockus and T. Dybå, "Quantifying the Effect of Code Smells on Maintenance Effort," in *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1144-1156, Aug. 2013
- [38] C. Acerbi and D. Tasche, “Expected Shortfall: A Natural Coherent Alternative to Value at Risk,” *Economic Notes*, 2002.
- [39] F. Zhang, A. E. Hassan, S. McIntosh and Y. Zou, "The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models," in *IEEE Transactions on Software Engineering*, vol. 43, no. 5, pp. 476-491, 1 May 2017
- [40] OpenJS Foundation, “Express.js Routing,” *Express.js Documentation*. [Online]. Available: <https://expressjs.com/en/guide/routing.html>
- [41] D. Spadini, M. Aniche and A. Bacchelli, “PyDriller: Python Framework for Mining Software Repositories,” in *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 908–911, 2018
- [42] PyDriller, “Process Metrics,” *PyDriller Documentation*. [Online]. Available: <https://pydriller.readthedocs.io/en/latest/processmetrics.html>.
- [43] Vite Contributors, “Getting Started,” *Vite Documentation*. [Online]. Available: <https://vite.dev/guide/>.

- [44] Chart.js Contributors, “Chart.js,” *Charts.js Documentation* 2025. [Online]. Available: <https://www.chartjs.org/docs/latest/>.
- [45] MYSQL2, “Quickstart,” *MYSQL2 Documentation*. [Online]. Available: <https://sidorares.github.io/node-mysql2/docs>.
- [46] MariaDB, “COMMIT,” *MariaDB Server Documentation*. [Online]. Available: <https://mariadb.com/docs/server/reference/sql-statements/transactions/commit>.
- [47] MariaDB, “Foreign Key Constraints,” *MariaDB Server Documentation*. [Online]. Available: <https://mariadb.com/docs/server/architecture/server-constraints/foreign-key-constraints>.
- [48] React router, “Routing,” *React Router Documentation*. [Online]. Available: <https://reactrouter.com/start/declarative/routing>.
- [49] I. Heitlager, T. Kuipers and J. Visser, "A Practical Model for Measuring Maintainability," 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), Lisbon, Portugal, 2007, pp. 30-39

ΠΑΡΑΡΤΗΜΑ Α: ΟΔΗΓΙΕΣ ΕΓΚΑΤΑΣΤΑΣΗΣ ΚΑΙ ΕΚΤΕΛΕΣΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

1. Απαραίτητο λογισμικό

Πριν από την εκτέλεση της εφαρμογής πρέπει να εγκατασταθούν:

- Git για τη λήψη των αποθετηρίων από το GitHub
- Node.js 22.12 ή νεότερο, μαζί με το npm
- Python 3 για την εκτέλεση των αρχείων ανάλυσης
- XAMPP για την τοπική λειτουργία του MariaDB/MySQL server
- HeidiSQL για τη δημιουργία, εισαγωγή και διαχείριση της βάσης δεδομένων (προτείνεται)
- Java Development Kit συμβατό με την έκδοση του SonarQube
- SonarQube Community Build το οποίο εκτελείται τοπικά
- SonarScanner CLI για την αποστολή του κώδικα στο SonarQube
- Maven ή Gradle όταν αναλύονται έργα Java που χρησιμοποιούν αντίστοιχο σύστημα build
- Πρόσβαση στην υπηρεσία AllinCode και έγκυρα στοιχεία σύνδεσης
- Σύνδεση στο διαδίκτυο κατά την εγκατάσταση και την εκτέλεση της εφαρμογής

Το Git, η Python, το Node.js, το npm και η Java πρέπει να είναι διαθέσιμα από τη γραμμή εντολών. Το Maven ή το Gradle πρέπει επίσης να είναι διαθέσιμο, όταν απαιτείται από το Java project που πρόκειται να αναλυθεί.

2. Αποσυμπίεση του project

Ο χρήστης κατεβάζει το αρχείο ZIP της εφαρμογής. Στη συνέχεια το αποσυμπίεζει σε έναν τοπικό φάκελο. Η βασική δομή των φακέλων είναι:

- project/ back-end/
- project/ front-end/
- Project/processing/

3. Ρύθμιση της βάσης δεδομένων με XAMPP και HeidiSQL

Με την προϋπόθεση ότι το XAMPP και το HeidiSQL (ή αντίστοιχη πλατφόρμα) είναι ήδη εγκατεστημένα, ο χρήστης ακολουθεί τα παρακάτω βήματα:

- Ανοίγει το XAMPP Control Panel
- Ενεργοποιείται η MySQL πατώντας Start
- Ανοίγει το HeidiSQL
- Δημιουργείται ή επιλέγεται σύνδεση τύπου MariaDB or MySQL

Συμπληρώνονται τα στοιχεία σύνδεσης:

- Hostname: 127.0.0.1
- User: root
- Password: ο κωδικός που έχει οριστεί στο XAMPP ή κενό πεδίο
- Port: 3306

Μετά τη σύνδεση δημιουργείται νέα βάση δεδομένων με όνομα `analyzed_projects`. Η βάση μπορεί να δημιουργηθεί από το γραφικό περιβάλλον του HeidiSQL ή με την εντολή:

```
CREATE DATABASE analyzed_projects
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;
```

Επιλέγεται η βάση `analyzed_projects`. Στη συνέχεια, ο χρήστης ανοίγει στο HeidiSQL το αρχείο SQL που περιλαμβάνεται στο ZIP του project. Τέλος, εκτελείται το αρχείο, ώστε να δημιουργηθούν οι απαραίτητοι πίνακες:

- `projects`
- `analysis_runs`
- `merged_file_metrics`
- `project_trend_history`

Μετά την ολοκλήρωση της ρύθμισης, το HeidiSQL μπορεί να κλείσει. Το XAMPP πρέπει να παραμένει ενεργό όσο εκτελείται η εφαρμογή.

4. Ρύθμιση του SonarQube

Το SonarQube πρέπει να ξεκινήσει πριν από το back-end. Σε περιβάλλον Windows μπορεί να εκτελεστεί το αρχείο `StartSonar.bat` από τον φάκελο εγκατάστασης του, μέσω της γραμμής εντολών. Μετά την εκκίνησή του, πρέπει να είναι διαθέσιμο στη διεύθυνση `http://localhost:9000`. Στη συνέχεια ο χρήστης συνδέεται στο SonarQube και μεταβαίνει στις ρυθμίσεις του λογαριασμού του ώστε να δημιουργήσει ένα προσωπικό token με τον εξής τρόπο:

My Account → Security → Generate Tokens

Το token αυτό θα καταχωριθεί στο αρχείο `.env`. Κάθε χρήστης πρέπει να χρησιμοποιεί το δικό του token. Επίσης, ο χρήστης που δημιουργεί το token πρέπει να έχει δικαίωμα εκτέλεσης αναλύσεων και δημιουργίας νέων SonarQube projects.

5. Δημιουργία του αρχείου `.env`

Μέσα στον φάκελο back-end δημιουργείται ένα αρχείο με όνομα `.env`. Το αρχείο περιλαμβάνει τις ρυθμίσεις της MariaDB, του SonarQube και του AllinCode:

- `DB_HOST=127.0.0.1`
- `DB_PORT=3306`
- `DB_USER=root`
- `DB_PASSWORD=`
- `DB_NAME=analyzed_projects`
- `SONAR_URL=http://localhost:9000`
- `SONAR_TOKEN=your_personal_sonarqube_token`
- `SONAR_SCANNER_HOME=C:\path\to\sonar-scanner`
- `ALLINCODE_BASE_URL=https://se.uom.gr/back-end-debtclock/api`
- `ALLINCODE_EMAIL=your_email`
- `ALLINCODE_PASSWORD=your_password`

Προαιρετικά μπορούν να οριστούν χρονικά όρια για το AllinCode:

- ALLINCODE_POLL_INTERVAL_MS=10000
- ALLINCODE_ANALYSIS_TIMEOUT_MS=1800000

Αν ο χρήστης root διαθέτει κωδικό, συμπληρώνεται στο DB_PASSWORD. Το SONAR_SCANNER_HOME πρέπει να δείχνει στον βασικό φάκελο του SonarScanner και όχι στον υποφάκελο bin. Το .env δεν πρέπει να περιλαμβάνεται σε δημόσιο αποθετήριο ή στο ZIP με πραγματικούς κωδικούς και tokens. Αντί γι' αυτό, στο ZIP συμπεριλαμβάνεται ένα .env.example με κενές τιμές.

6. Εγκατάσταση των Python βιβλιοθηκών

Η βιβλιοθήκη PyDriller είναι απαραίτητη για την επεξεργασία του ιστορικού Git. Η εγκατάστασή της πραγματοποιείται από τη γραμμή εντολών με την εντολή `python -m pip install pydriller`. Η PyDriller χρησιμοποιείται από το `git_metrics.py`, ενώ τα `analytics_pipeline.py` και `scoring.py` βασίζονται στις βιβλιοθήκες της βασικής εγκατάστασης της Python.

7. Εγκατάσταση των εξαρτήσεων του back-end και του front-end

Η εγκατάσταση των εξαρτήσεων πραγματοποιείται μέσω της γραμμής εντολών. Αρχικά, ο χρήστης μεταβαίνει στον φάκελο back-end και εκτελεί την `npm install`. Αντίστοιχα, μεταβαίνει στον φάκελο front-end και εκτελεί την ίδια εντολή.

Οι εντολές διαβάζουν το αντίστοιχο αρχείο `package.json` κάθε φακέλου. Στο back-end εγκαθίστανται το Express, το mysql2, το Axios και οι υπόλοιπες απαραίτητες βιβλιοθήκες. Στο front-end εγκαθίστανται το React, το Vite, το React Router, το Chart.js και οι υπόλοιπες εξαρτήσεις της διεπαφής χρήστη. Η διαδικασία απαιτείται μόνο κατά την πρώτη εγκατάσταση ή όταν αλλάζουν οι εξαρτήσεις του project.

8. Εκτέλεση της εφαρμογής

Πριν ξεκινήσει η εφαρμογή πρέπει να λειτουργούν ήδη το XAMPP και το SonarQube στη θύρα 9000. Στον φάκελο back-end μέσω της γραμμής εντολών εκτελείται η εντολή `node server.js`. Αν η εκκίνηση ολοκληρωθεί σωστά, εμφανίζεται “Server running on `http://localhost:3000`”. Το τερματικό αυτό πρέπει να παραμείνει ανοιχτό.

Αντίστοιχα, στον φάκελο front-end μέσω της γραμμής εντολών εκτελείται η εντολή `npm run dev`. Το Vite εμφανίζει τη διεύθυνση της εφαρμογής, η οποία είναι συνήθως `http://localhost:5173`. Το τερματικό του front-end πρέπει επίσης να παραμείνει ανοιχτό. Η εφαρμογή λειτουργεί μέσω φυλλομετρητή στη διεύθυνση που εμφανίζει το Vite.

Ο πηγαίος κώδικας είναι διαθέσιμος στον παρακάτω σύνδεσμο:

https://github.com/AntoniouSophia/Code_Health_Dashboard