ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΞΙΟΛΟΓΗΣΗ ΠΟΙΟΤΗΤΑΣ ΚΩΔΙΚΑ ΣΕ ΛΟΓΙΣΜΙΚΟ SAP

Διπλωματική Εργασία

του

Σερέτη Κλεάνθη

Θεσσαλονίκη, Φεβρουάριος 2023

ΑΞΙΟΛΟΓΗΣΗ ΠΟΙΟΤΗΤΑΣ ΚΩΔΙΚΑ ΣΕ ΛΟΓΙΣΜΙΚΟ SAP

Σερέτης Κλεάνθης

Πτυχίο Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας 2022

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Αμπατζόγλου Απόστολος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 01/03/2023

Χατζηγεωργίου Αλέξανδρος	Ξυνόγαλος Στυλιανός	Αμπατζόγλου Απόστολος
.....	.....	.....

Σερέτης Κλεάνθης

.....

Περίληψη

Στην παρούσα διπλωματική εργασία, θα μελετηθεί η ποιότητα κώδικα σε συστήματα SAP ERP. Για τον σκοπό αυτό, αναπτύχθηκε λογισμικό για την ανάλυση και τον υπολογισμό μετρικών, σχετικά με την αξιολόγηση της ποιότητας του κώδικα.

Αρχικά, παρατίθεται η έννοια του SAP, αναλύεται η χρησιμότητά του καθώς και οι λύσεις που προσφέρει σε τυχόν προβλήματα που προκύπτουν στον επιχειρηματικό τομέα. Επιπρόσθετα, αναλύεται η βασική γλώσσα προγραμματισμού, με την οποία έχει αναπτυχθεί ο βασικός πυρήνας του λογισμικού SAP, η ABAP. Ενδεικτικά, θα μελετηθούν το συντακτικό, τα πλεονεκτήματα, η δομή, καθώς και ο τρόπος λειτουργίας της.

Συνεχίζοντας, περιγράφονται πληθώρα εργαλείων τα οποία παρέχονται από τον κατασκευαστή συνδυαστικά με το βασικό προϊόν και η χρήση τους απευθύνεται κυρίως σε προγραμματιστές, προκειμένου οι ίδιοι να τα αξιοποιήσουν στην βελτιστοποίηση και συντήρηση του κώδικα. Εντοπίζοντας και αναλύοντας τις ελλείψεις των παραπάνω εργαλείων, κυρίως στο πλήθος των μετρικών, αναπτύχθηκε λογισμικό με σκοπό την δημιουργία ενός ολοκληρωμένου εργαλείου ανάλυσης κώδικα. Σε δεύτερο χρόνο, περιγράφεται η μεθοδολογία που ακολουθήθηκε για την ανάπτυξη του συγκεκριμένου εργαλείου, οι σχεδιαστικές αποφάσεις που πάρθηκαν καθώς και η αρχιτεκτονική του. Επίσης, ερμηνεύεται η σημασία των μετρικών που χρησιμοποιήθηκαν ενώ παράλληλα περιγράφεται ο τρόπος με τον οποίο υπολογίζονται.

Τέλος, προβάλλονται τα αποτελέσματα, τα γραφήματα καθώς και κάποια συμπεράσματα, σχετικής ερευνάς που πραγματοποιήθηκε, όσον αφορά την σύγκριση των αποτελεσμάτων του λογισμικού που αναπτύχθηκε, με την αξιολόγηση συναδέλφων, σε ίδια τμήματα κώδικα.

Λέξεις Κλειδιά: SAP, συστήματα ERP, ποιότητα κώδικα, λογισμικό, μετρήσεις, ABAP, κλάση, μέθοδος, μετρικές, ανάλυση κώδικα.

Abstract

This thesis aims to study the quality of code in SAP ERP systems. To achieve this goal, a software tool was developed that analyzes and calculates metrics to evaluate code quality.

The thesis starts by introducing the concept of SAP, analyzing its usefulness, and exploring the solutions it provides to business problems. Additionally, the study analyzes the fundamental programming language used in the main core of SAP software, ABAP. This analysis includes an examination of its syntax, advantages, structure, and mode of operation.

Moving on, the thesis outlines various tools provided by the manufacturer to programmers, which are designed to optimize and maintain code in conjunction with the basic product. The study identifies and analyzes the limitations of these tools, particularly in terms of the number of metrics they provide. To overcome these limitations, the thesis presents the development of an integrated code analysis tool. The second year of the study is dedicated to describing the methodology used to develop the tool, the design decisions made, and its architecture. Additionally, the thesis explains the significance of the metrics used and describes how they are calculated.

The thesis concludes by presenting the results, graphs, and conclusions of a study that compares the evaluation of code sections by colleagues with the results obtained from the software developed. The study provides relevant research findings, highlighting the effectiveness of the software in analyzing code quality.

Keywords: SAP, ERP systems, code quality, software tool, metrics, ABAP, integrated code analysis, class, method, metrics, code quality.

Πρόλογος – Ευχαριστίες

Ξεκινώντας, θα ήθελα να ευχαριστήσω τους ανθρώπους που με βοήθησαν και στάθηκαν δίπλα μου στην διαδικασία της εκπόνησης της παρούσας διπλωματικής εργασίας. Ένα μεγάλό ευχαριστώ στον κ. Αμπατζόγλου, ο οποίος με καθοδήγησε σε κάθε βήμα μου, δίνοντας συμβουλές σε ζητήματα με τα οποία ερχόμουν πρώτη φορά σε επαφή. Επίσης, ένα ακόμα ευχαριστώ στους πολύ κοντινούς μου ανθρώπους, για την στήριξη τους στην προσπάθεια μου.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Τα έργα λογισμικού από το παρελθόν μέχρι και σήμερα μεγεθύνονται ραγδαία σε γραμμές κώδικα, καθώς προκύπτουν καθημερινά καινούργιες ανάγκες και απαιτήσεις. Η γρήγορη ανάπτυξη λογισμικού και παράδοση του, μπορεί να οδηγήσει σε προβλήματα όσον αφορά την μελλοντική επέκταση και συντήρηση του έργου. Οι σχεδιαστικές αποφάσεις που παίρνονται χωρίς μελέτη και ανάλυση του προβλήματος, μειώνουν τον συντελεστή της συντηρησιμότητας του έργου. Ο υπολογισμός μετρικών προσφέρει λύση σε αυτό το πρόβλημα των έργων λογισμικού. Αναλύει τον κώδικα και συνεισφέρει αισθητά στην βελτίωση της αποδοτικότητας, της ποιότητας, στην λήψη καλύτερων αποφάσεων ή την διαχείριση του έργου και το καθιστούν συντηρήσιμο.

1.2 Σκοπός – Στόχοι

Ο σκοπός αυτής της μελέτης είναι η δημιουργία ενός προγράμματος ανάλυσης κώδικα και υπολογισμού μετρικών, που θα συνεισφέρει στο έργο των προγραμματιστών του SAP. Αναλύοντας των κώδικα και βάση των αποτελεσμάτων, ο προγραμματιστής θα είναι σε θέση να προβεί στις αναγκαίες βελτιώσεις του έργου, καθιστώντας το συντηρήσιμο και ευανάγνωστο.

1.3 Συνεισφορά

Τα βήματα που ακολουθήθηκαν από την έναρξη της έρευνας είναι:

- Ερευνά και εντοπισμός εργαλείων αξιολόγησης ποιότητας κώδικα σε συστήματα SAP
- Μελέτη των αλγορίθμων για την υλοποίηση των μετρικών αξιολόγησης ποιότητας κώδικα
- Αρχιτεκτονικές και σχεδιαστικές αποφάσεις για την ανάπτυξη του προγράμματος

- Υλοποίηση του προγράμματος
- Έλεγχος και δοκιμές
- Εξαγωγή αποτελεσμάτων και δημιουργία ερωτηματολογίου για την εύρεση της μέσης τιμής της συντηρησιμότητας μεθόδων
- Ανάλυση και μελέτη των αποτελεσμάτων
- Διάρθρωση της διπλωματικής εργασίας

1.4 Βασική Ορολογία

ERP: Enterprise Resource Planning

SAP: Το πιο διαδεδομένο ERP σύστημα παγκόσμιος

ABAP: Η κύρια γλώσσα προγραμματισμού του SAP

1.5 Διάρθρωση της μελέτης

Στο δεύτερο κεφάλαιο της παρούσας διπλωματικής εργασίας περιγράφεται το τί είναι το SAP, η χρησιμότητα του και τα πλεονεκτήματα του. Επίσης αναλύεται η γλώσσα προγραμματισμού ABAP, το συντακτικό της καθώς και ο τρόπος χρήσης της. Στο τρίτο κεφάλαιο αναλύεται η μεθοδολογία που ακολουθήθηκε για την δημιουργία του προγράμματος υπολογισμού μετρικών, οι σχεδιαστικές αποφάσεις που πάρθηκαν καθώς και η αρχιτεκτονική του. Επιπρόσθετα, γίνεται αναφορά στην αξιοποίηση των ευρημάτων και στις χρήσιμες πληροφορίες που μπορούμε να λάβουμε από αυτά.

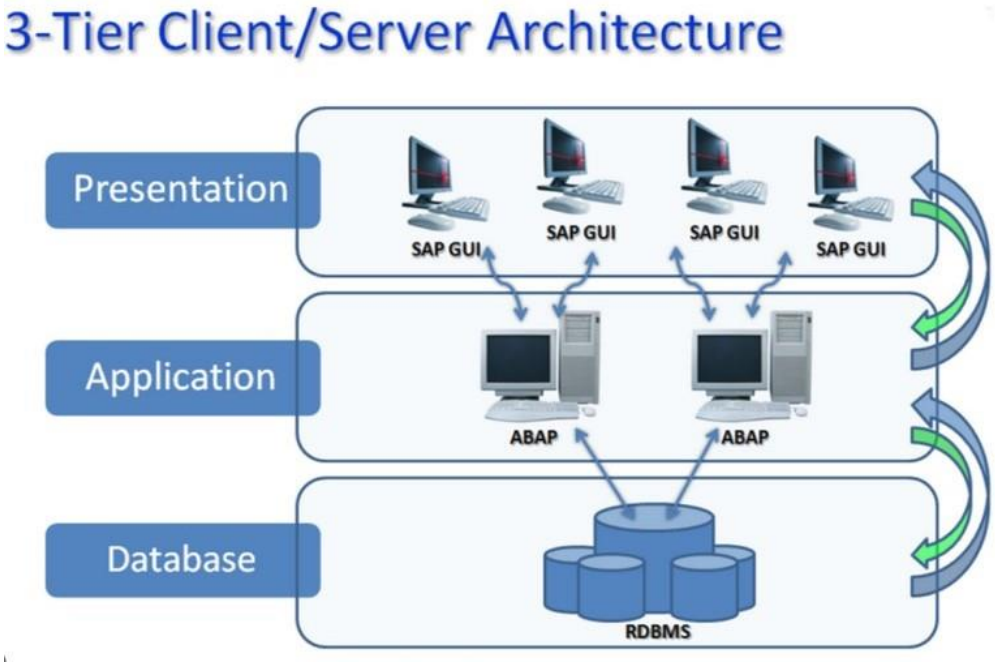
2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

2.1 SAP ERP

2.1.1 Τι είναι το SAP

Το SAP είναι ένα σύστημα προγραμματισμού πόρων για τις επιχειρήσεις (Enterprise Resource Planning - ERP) το οποίο δημιουργήθηκε το 1972, από πέντε πρώην υπαλλήλους της IBM, οι οποίοι είχαν το όραμα να δημιουργήσουν ένα λογισμικό που θα ανταποκρίνεται σε πραγματικά σενάρια επιχειρήσεων. Η εταιρία SAP κατοχύρωσε το όνομα της από τη φράση “Systemanalyse Programmentwicklung”, η οποία μεταφράζεται σε “System Analysis Program Development”. Σήμερα, η εταιρία έχει την επωνυμία SAP SE.

Μετά την ενημέρωση R/2 η SAP καθιέρωσε το παγκόσμιο πρότυπο ERP και με την R/3 καθιερώθηκε και το σύστημα τριών διαζωμάτων (***Εικόνα 2-1***). Πλέον η εταιρία έχει προχωρήσει σε τεχνολογίες Cloud χρησιμοποιώντας την δύναμη του in-memory, τεχνητή νοημοσύνη και μηχανική μάθηση.



Εικόνα 2-1: Αρχιτεκτονική τριών διαζωμάτων

Ένα σύστημα SAP μπορεί να προσφέρει λύσεις σε πολλούς τομείς μιας επιχείρησης, όπως πωλήσεις και διανομή, διαχείριση προϊόντων, σχεδιασμό παραγωγής, logistics, έλεγχο ποιότητας, οικονομικά και χρηματοοικονομικά, διαχείριση ανθρώπινου δυναμικού και πολλά άλλα. Είναι ευρέως γνωστό για την ευελιξία και επεκτασιμότητά του, καθώς διευκολύνει την διαχείριση περίπλοκων δεδομένων, μεγάλων εταιριών. Ωστόσο, λόγω του υψηλού κόστους του, το λογισμικό συνιστάται για μεγάλες και μόνο επιχειρήσεις.

Το SAP χωρίζεται σε τμήματα, τα οποία ονομάζονται **modules**. Το κάθε **module** έχει τις δικιές του αρμοδιότητες. Τα ποιο σημαντικά και γνωστά είναι τα εξής:

- Πωλήσεις & διανομή (Sales & Distribution - **SD**)
- Διαχείριση προϊόντων (Materials Management - **MM**)
- Σχεδιασμός παραγωγής (Production Planning – **PP**)
- Χρηματοοικονομική Λογιστική & διαχείριση (Financial Accounting & Controlling – **FICO**)
- Έλεγχος ποιότητας (Quality Management - **QM**)
- Διαχείριση μονάδας (Plant Maintenance – **PM**)
- Εκτέλεση Logistics (Logistics Execution – **LE**)
- Σύστημα Έργου (Project System – **PS**)
- Ανθρώπινο Δυναμικό (Human Resources – **HR**)

2.1.2 Πλεονεκτήματα και οφέλη

Τα συστήματα ERP καλύπτουν όλους τους βασικούς επιχειρηματικούς τομείς, όπως προμήθεια, παραγωγή, διαχείριση υλικών, πωλήσεις, μάρκετινγκ, οικονομικά και ανθρώπινους πόρους. Προσφέρουν μία ενιαία λύση, η οποία ενσωματώνει ψηφιακά όλα τα μέρη μιας επιχείρησης και περιέχει εφαρμογές που καλύπτουν όλες τις ανάγκες.
Μερικά από τα πλεονεκτήματα που μπορεί να προσφέρει ένα σύστημα ERP:

- Ένα ενιαίο, κεντρικό και ολοκληρωμένο σύστημα πληροφοριών και προγραμματισμού για όλες τις εργασίες της επιχείρησης.
- Βελτιωμένες διαδικασίες και ροές εργασίας.
- Μείωση όγκου και χρόνου επεξεργασίας και παραγωγής δεδομένων.
- Δημιουργία εύκολων reports για κάθε τμήμα της επιχείρησης.

- Καθιέρωση συνεπών διαδικασιών που βασίζονται στις ορθές πρακτικές κατασκευής.
- Επιτρέπει την εμφάνιση των πληροφοριών πωλήσεων, πελατών και προμηθευτών.
- Διαβάθμιση ασφαλείας.
- Βελτιώνει την πρόσβαση στα στοιχεία.
- Βελτιώνει την εργασία και την αποτελεσματικότητα.
- Βελτιώνει την ικανοποίηση των πελατών μέσω προγραμματισμού παράδοσης προϊόντων στην ώρα τους.
- Υπάρχει σωστά κατανεμημένη διαθέσιμη ποσότητα προϊόντος.
- Μειωμένα κόστη αποθεμάτων λόγω καλύτερου σχεδιασμού, παρακολούθησης και προβλέψεων.
- Καλύτερη ορατότητα και πλήρη ανάλυση στους λογαριασμούς και λιγότερα λάθη πληρωμών/παραδόσεων.
- Μειωμένο κόστος προμηθευτών με καλύτερη εκμετάλλευση οικονομιών κλίμακας και διαχείρισης προβλέψεων παραγωγής.
- Παρακολούθηση του πραγματικού κόστους των διαδικασιών και έλεγχος του κόστος κάθε διαδικασίας.
- Ενοποιημένη εικόνα των πωλήσεων, των αποθεμάτων και των απαιτήσεων.

2.2 ABAP

2.2.1 Περιγραφή

Η ABAP αρχικά πήρε το όνομα της από τις λέξεις “Allgemeiner Berichts-Aufbereitungs-Prozessor”, όπου στα γερμανικά σημαίνει “General reports preparation processor” και στη συνέχεια μετονομάστηκε σε “Advanced Business Application Programming”. Είναι πλέον μία αντικειμενοστραφής γλώσσα προγραμματισμού , τέταρτης γενιάς, που μοιάζει με την γλώσσα COBOL. Χρησιμοποιείται αποκλειστικά από τα συστήματα ERP SAP και πιο συγκεκριμένα στην πλατφόρμα SAP NetWeaver (Application Server). Δημιουργήθηκε το 1980, όπου και εφαρμόστηκε στο τότε σύστημα SAP R/2 που χρησιμοποιούσαν εταιρίες/πολυεθνικές για διαχείριση προϊόντων,

οικονομικών και πόρων. Το 1992, η τότε εταιρία SAP AG με την αναβάθμιση του λογισμικού της από την πλατφόρμα R/2 σε R/3, έκανε διαθέσιμη την γλώσσα προγραμματισμού σε εξωτερικούς συνεργάτες - προγραμματιστές και έδωσε την δυνατότητα στους πελάτες να επεκτείνουν διαθέσιμα προγράμματα που βρίσκονταν ενσωματωμένα στο SAP. Με αυτή την κίνηση, ολοένα και περισσότερα προγράμματα γράφτηκαν σε ABAP, με αποτέλεσμα μετά από μερικά χρόνια (1999) η εταιρία να ανακοινώσει και επίσημα την αντικειμενοστραφή επέκταση της γλώσσας (ABAP Objects).

Η πλατφόρμα SAP NetWeaver πλέον τρέχει σε συστήματα Linux, Microsoft Windows και Mac OS, ενώ χρησιμοποιεί βάσεις δεδομένων της Oracle, Microsoft και πλέον με την τελευταία ενημέρωση την βάση δεδομένων που δημιουργήθηκε από την SAP AG, S/4 HANA. Αξίζει να αναφερθεί ότι η ABAP, θεωρείται μία από τις πρώτες γλώσσες προγραμματισμού που ενσωμάτωσαν την έννοια των λογικών βάσεων δεδομένων (LDBs).

2.2.2 Περιβάλλον

Στις περισσότερες γλώσσες προγραμματισμού, κατά την δημιουργία ενός αρχείου, το αρχείο θα πάρει την επέκταση της αντίστοιχης γλώσσας. Για παράδειγμα, εάν χρησιμοποιούμε την γλώσσα προγραμματισμού Java, τότε όλα τα αρχεία του πηγαίου κώδικα έχουν την επέκταση “.java”, ενώ εάν γίνεται χρήση της C#, τότε η επέκταση θα είναι “.cs”.

Στο SAP, ο πηγαίος κώδικας αποθηκεύεται σε δύο μορφές, τον πηγαίο κώδικα που μπορεί να προβληθεί και να επεξεργαστεί από κάποιον editor (ABAP Workbench, Eclipse ADT) και ένα binary αρχείο παρόμοιο με Java Bytecode.

Όλα τα προγράμματα που έχουν υλοποιηθεί με ABAP, είτε από την ίδια την SAP είτε από εξωτερικούς προγραμματιστές, εκτελούνται εντός του πυρήνα του SAP, το οποίο είναι υπεύθυνο για την σωστή εκτέλεση των εντολών, τη συνοχή, των οθονών καθώς και των events τους.

Στο περιβάλλον του SAP είναι ενσωματωμένο ένα interface για την διαχείριση δεδομένων με την αντίστοιχη βάση δεδομένων. Μεταφράζει τις ABAP εντολές σε Native

SQL εντολές. Συνοπτικά, αναλαμβάνει όλη την επικοινωνία του προγράμματος με την βάση δεδομένων.

2.2.3 Landscapes

Όπως και στα περισσότερα έργα λογισμικού έτσι και στο SAP υπάρχει μια διαδικασία που απαιτείται για να διατεθεί ένα πρόγραμμα ή κάποια αλλαγή σε ένα ήδη υπάρχον πρόγραμμα, στο παραγωγικό σύστημα του εκάστοτε πελάτη. Η SAP προτείνει όπως και συνήθως γίνεται, την δημιουργία τριών περιβαλλόντων απόλυτα συνδεδεμένα μεταξύ τους.

Ξεκινώντας, το πρώτο σύστημα ονομάζεται **Development**, στο οποίο πραγματοποιούνται όλες οι διαδικασίες για την δημιουργία καινούργιων προγραμμάτων ή τυχόν αλλαγές σε παραγωγικά προγράμματα. Επίσης, πέρα από την δημιουργία προγραμμάτων, στο σύστημα Development, μπορούν να γίνουν και ρυθμίσεις τύπου configuration για τα standard προγράμματα που προσφέρει το SAP.

Το επόμενο σύστημα ονομάζεται **Quality Assurance**, στο οποίο μεταφέρονται τα προγράμματα από το Development, προκειμένου να πραγματοποιηθούν οι απαραίτητες δοκιμές (tests) και να ελεγχθεί η ποιότητα κώδικα.

Τέλος, το σύστημα **Production** είναι το παραγωγικό σύστημα του πελάτη. Συνήθως στο σύστημα Quality υπάρχουν test cases τα οποία ανταποκρίνονται στην πραγματικότητα. Δηλαδή τα δεδομένα είναι παρόμοια με αυτά που θα χρησιμοποιήσει ο πελάτης στο παραγωγικό σύστημα Production.

Βέβαια υπάρχουν και περιπτώσεις όπου ο πελάτης θα αιτηθεί παραπάνω από τρία συστήματα. Ένα σύστημα μπορεί να ενσωματωθεί ανάμεσα στο Development και Quality και να χρησιμοποιηθεί για Unit Testing. Επίσης, μία άλλη περίπτωση είναι να αιτηθεί ένα σύστημα Pre-Production για εκτεταμένο testing πρώτου πραγματοποιηθεί η μεταφορά των αλλαγών στο παραγωγικό σύστημα. Στην πλειοψηφία των περιπτώσεων που υπάρχουν πέντε συστήματα στο Landscape της εταιρίας, οι πιθανότητες να εντοπισθεί κάποιο σφάλμα (bug) σε πρόγραμμα είναι μικρότερες, διότι ο χρόνος που αφιερώνεται σε testing είναι σημαντικά μεγαλύτερος σε σχέση με ένα Landscape εταιρίας με τρία συστήματα.

2.2.4 Transactions

Κάθε εκτελέσιμο πρόγραμμα που υπάρχει διαθέσιμο στο SAP, είτε standard, είτε custom, θα τρέξει βάση του transaction κωδικού του. Για παράδειγμα, η μεταφορά των αλλαγών μεταξύ των συστημάτων Development – Quality – Production, διαχειρίζεται μέσα από το transaction "STMS".

Μερικά από τα πιο σημαντικά transactions όσο αναφορά την ανάπτυξη κώδικα:

- **SE38**: Ο βασικός editor για δημιουργία ABAP προγραμμάτων
- **SE11**: Το transaction για το Dictionary, υπάρχει δυνατότητα διαχείρισης πινάκων και τύπων.
- **SE37**: Ο editor για την δημιουργία Function Modules
- **SE24**: Ο editor για την δημιουργία κλάσεων και interfaces
- **SE80**: Ονομάζεται Object Navigator και συνδυάζει κάποια από τα παραπάνω transactions και αρκετές άλλες λειτουργίες.

Σε αυτό το σημείο να αναφερθεί ότι το όνομα κάθε custom αντικειμένου που δημιουργείται πρέπει να ξεκινάει με Z ή Y (Προτείνεται από το SAP).

2.2.5 Τύποι προγραμμάτων ABAP

Στο SAP υπάρχουν δύο κατηγορίες προγραμμάτων, τα εκτελέσιμα και τα μη εκτελέσιμα. Στην κατηγορία των εκτελέσιμων προγραμμάτων ανήκουν τα:

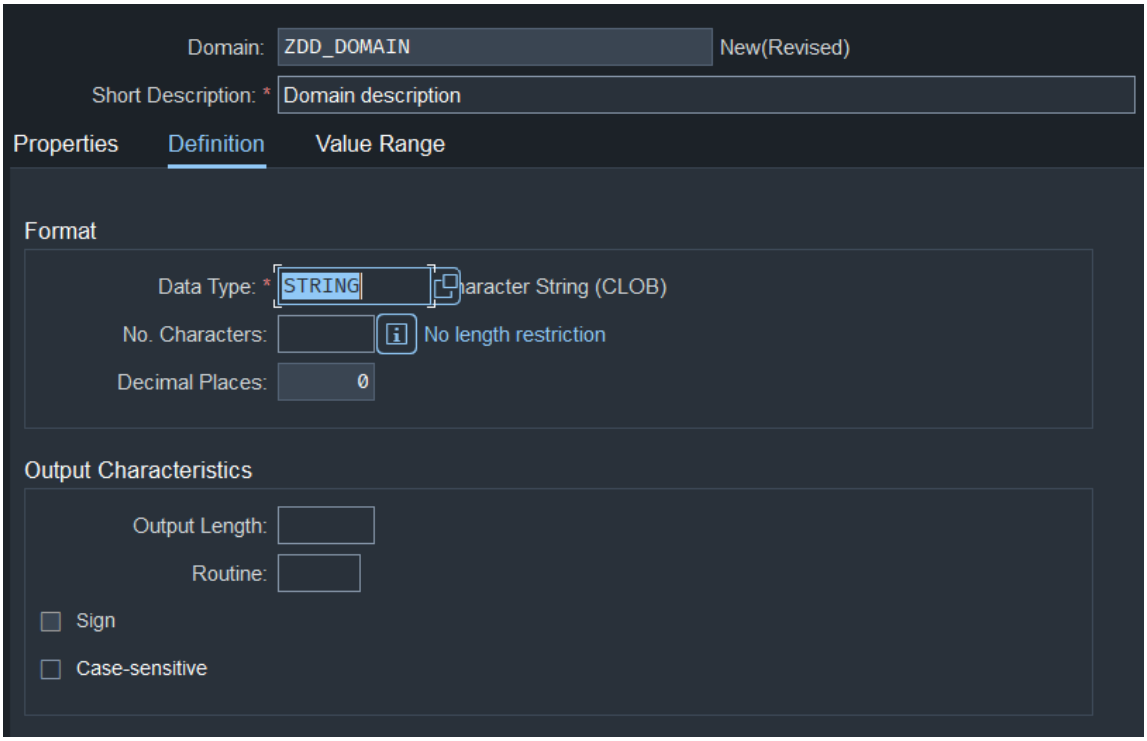
- **Reports**, τα οποία έχουν να κάνουν συνήθως με μία αρχική οθόνη εισαγωγής παραμέτρων του χρήστη που χρησιμοποιούνται από το πρόγραμμα και ως αποτέλεσμα εμφανίζεται μια λίστα με δεδομένα.
- **Module pools**, τα οποία αποτελούν μία πιο περίπλοκη μορφή προγραμμάτων. Αποτελούνται από οθόνες όπου η κάθε μία κρύβει μία λογική εκτέλεσης. Ορισμένες φορές τα προγράμματα “Module pools” αναφέρονται και ως dynpro.

Στην κατηγορία των μη εκτελέσιμων ανήκουν:

- **Includes**, χρησιμοποιούνται συνήθως για την διάσπαση μεγάλων προγραμμάτων. Συνήθως σε κάθε πρόγραμμα, υπάρχει ένα για την δήλωση μεταβλητών, ένα για την οθόνη του προγράμματος και ένα για τοπικές κλάσεις.
- **Subroutines**, αν και ξεπερασμένη τεχνολογία χρησιμοποιείται ακόμα. Περιέχει φόρμες (Forms) που μοιάζουν με μέθοδο κλάσης. Ο κώδικας των forms μπορεί να ξαναχρησιμοποιηθεί.
- **Function Modules**, επαναχρησιμοποιούμενος κώδικας διαθέσιμος σε όλο το σύστημα SAP. Μπορεί να δεχτεί παραμέτρους και να επιστρέψει αποτελέσματα.
- **Object classes**, βασισμένο στην αντικειμενοστρέφεια, όπως κάθε αντικειμενοστραφής γλώσσα προγραμματισμού.
- **Interfaces**, επίσης βασισμένο στην αντικειμενοστρέφεια. Κλάσεις με κενές μεθόδους.
- **Types pools**, είναι συλλογές τύπων και σταθερών μεταβλητών.

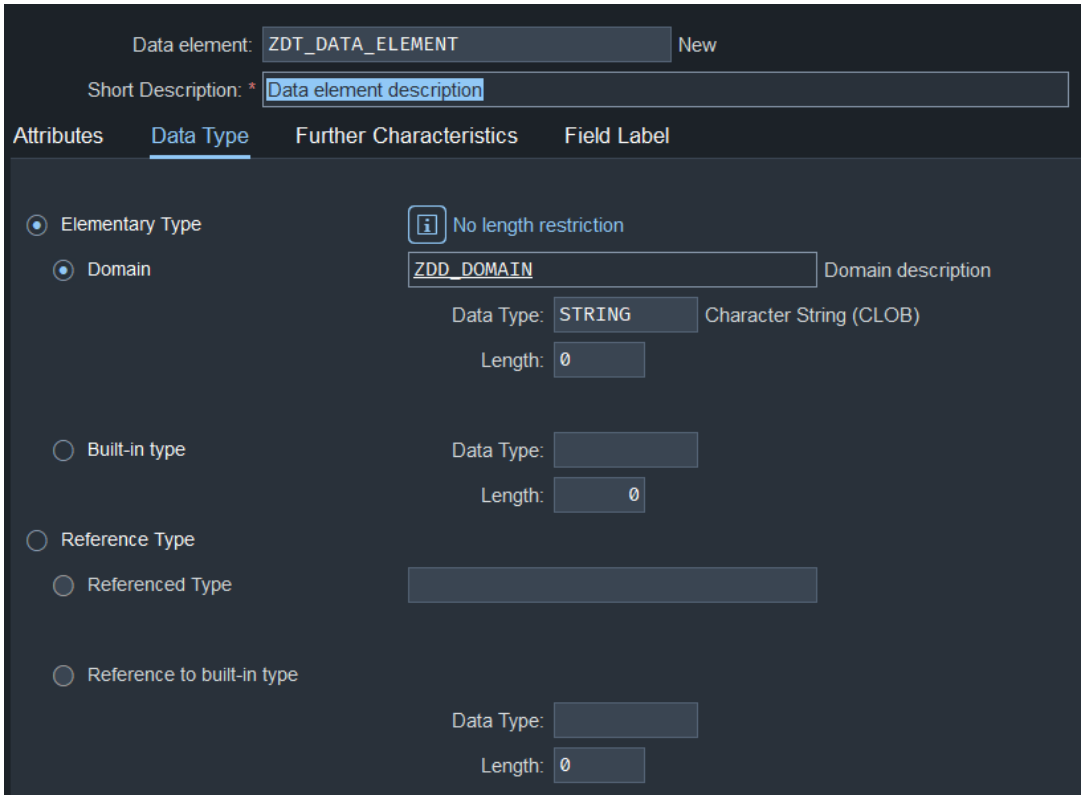
2.2.6 Dictionary SE11

Ένα από τα πιο σημαντικά transactions που πρέπει να γνωρίζει ένας προγραμματιστής είναι το **SE11.** Μέσα από αυτό μπορούμε να διαχειριστούμε πίνακες (tables), data-elements, domains και structures. Η βάση ξεκινάει από τα **domains**, τα οποία καθορίζουν τον τύπο του πεδίου ή μεταβλητής. Στην παρακάτω εικόνα φαίνεται η οθόνη κατά την δημιουργία ενός **domain**. Τα υποχρεωτικά πεδία είναι η περιγραφή και ο τύπος, ενώ υπάρχουν και δευτερεύοντα πεδία, όπως τα δεκαδικά ψηφία εάν είναι δεκαδικός αριθμός, case-sensitive εάν είναι κείμενο και φυσικά υπάρχει και η επιλογή διανύσματος. Οι υποστηριζόμενοι τύποι μεταβλητών είναι οι ακέραιοι (integer), δεκαδικοί (float), character (char), κείμενο (string), αλφαριθμητικοί (numc), ημερομηνία (dats), ώρα (tims), timestamp (utclong), boolean (abap_bool), byte (raw) και byte string (rawstring).



Εικόνα 2-2: Οθόνη δημιουργίας domain

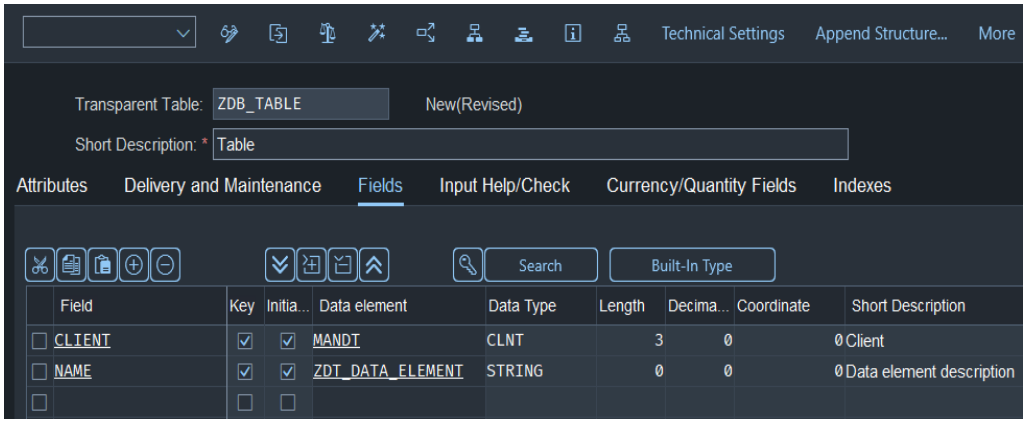
Επόμενα σε σειρά προτεραιότητας, έρχονται τα **data elements.** Το πρώτο που πρέπει να αναφερθεί είναι η δήλωση ενός **domain** στο **data element**. Ένα **data element** (***Εικόνα 2-3***) αντιπροσωπεύει τον πλήρες τύπο μιας μεταβλητής. Πέρα από το **domain** που μας δείχνει τι τιμές πρέπει να πάρει, συγκρατεί στοιχεία όπως την περιγραφή του πεδίου, τη βοήθεια αναζήτησης (search help) και την επικεφαλίδα του, δηλαδή τι θα δει ο χρήστης κατά την εκτύπωση (καρτέλα Field Label). Η περισσότερο συχνή τεχνική είναι η δημιουργία **data elements** και η χρήση έτοιμων **domains**, διότι το **domain** κρατάει τον τύπο και το **data element** την επικεφαλίδα και την περιγραφή. Επιπρόσθετα, εάν έχει υλοποιηθεί η βοήθεια αναζήτησης, μπορεί να βοηθήσει τον χρήστη στην εύρεση τιμών μέσα από ένα μενού αναζήτησης.



Εικόνα 2-3: Οθόνη δημιουργίας data element

Για να γίνει απολύτως κατανοητή η χρήση των προηγούμενων δύο, χρειάζεται η επεξήγηση των πινάκων. Δημιουργώντας ένα πίνακα είναι αναγκαίο να δηλώσουμε το όνομα του πεδίου και τον τύπο. Όπως φαίνεται στην ***Εικόνα 2-4***, στην στήλη "Data element" πρέπει να δηλώσουμε ένα ήδη υπάρχων **data element**. Πληκτρολογώντας το όνομα του και πατώντας enter εμφανίζονται όλες οι πληροφορίες που έχουμε ήδη καταχωρήσει, ο τύπος, το μέγεθος και η περιγραφή. Έτσι κατά την προσθήκη καινούργιων πεδίων μπορούμε να επαναχρησιμοποιήσουμε το ίδιο data element ή να δημιουργήσουμε καινούργιο. Όπως θα δούμε και παρακάτω, τα data elements χρησιμεύουν όχι μόνο στην δήλωση πεδίων σε πίνακες αλλά και σε μεταβλητές εντός των προγραμμάτων.

Σε αυτό το σημείο αξίζει να αναφερθεί, ότι όλοι οι πίνακες του SAP πρέπει να ξεκινούν με το πεδίο του πελάτη (client) το οποίο χρησιμοποιεί το data element "MANDT". Το πεδίο αυτό δεν αντιπροσωπεύει έναν πελάτη σαν εταιρία, αλλά ένα προφίλ για την εταιρία, το οποίο συγκρατεί ξεχωριστές πληροφορίες και δεδομένα. Με αυτόν τον τρόπο, μία εταιρία έχει τη δυνατότητα να χρησιμοποιεί παραπάνω από έναν clients.



Εικόνα 2-4: Οθόνη δημιουργίας πίνακα

Επίσης, η διαδικασία δημιουργίας **structure** είναι παρόμοια με αυτή που είδαμε παραπάνω, με την διαφορά ότι το structure αντιπροσωπεύει τον τύπο του πίνακα. Τέλος, τα structure χρησιμοποιούνται σε επίπεδο κώδικα και όχι στην βάση δεδομένων για την δημιουργία πίνακα.

2.2.7 Συντακτικό

Το συντακτικό της ABAP μοιάζει σε αυτό της COBOL, θεωρείται "proprietary language" δηλαδή μπορεί να χρησιμοποιηθεί μόνο εντός του συστήματος SAP. Όπως στις περισσότερες γλώσσες, χρησιμοποιούμε εντολές για να δηλώσουμε μεταβλητές και να δημιουργήσουμε αντικείμενα, με τον ίδιο τρόπο και στην περίπτωση της ABAP συντάσσουμε ένα block κώδικα. Προτού περάσουμε στην ανάλυση του συντακτικού, αξίζει να αναφερθούν μερικοί βασικοί κανόνες:

- Μία εντολή αποτελείται από λέξεις-κλειδιά, τελεστές, μεταβλητές και συμπληρωματικές λέξεις κλειδιά. Η πρώτη λέξη της εντολής είναι η λέξη κλειδί. Οι τελεστές μπορούν να χρησιμοποιηθούν για αριθμητικές πράξεις ή για λογικές.
- Για την ολοκλήρωση της εντολής χρησιμοποιείται η τελεία (**.**). Η εντολή μπορεί να μοιραστεί σε πολλές γραμμές.
- Κάθε λέξη στον κώδικα πρέπει να χωρίζεται με κενό.
- Ο κώδικας ABAP **δεν** είναι case-sensitive. Για παράδειγμα, οι μεταβλητές "**max_value**" και **"MAX_VALUE**" θεωρούνται ίδια μεταβλητή ακολουθώντας το πρότυπο ονομασίας "snake case".

- Κάθε εντολή μπορεί να περιέχει κώδικα στο εσωτερικό της, όπως μία “if” ή μία λούπα, τότε υπάρχει αντίστοιχο keyword που τερματίζει την εντολή. Η “if” τερματίζει με την “endif”, η “loop” με την “endloop”.

Προχωρώντας στην ανάλυση του συντακτικού, μερικές από τις παρακάτω εντολές, θα βοηθήσουν στην κατανόηση της γλώσσας.

2.2.7.1 Δηλώσεις

Για την δήλωση μεταβλητών είτε χρησιμοποιήσουμε το keyword “data” μεμονωμένα ή μία φορά με την χρήση άνω κάτω τελείας (:) και χωρίζοντας τις μεταβλητές με κόμμα (,). Σε κάθε μία από τις περιπτώσεις, η ABAP θα εκλάβει ακριβώς τις ίδιες μεταβλητές. Επίσης, παρατηρούμε ότι στην τρίτη γραμμή (***Εικόνα 2-5***) χρησιμοποιούμε το data element που δημιουργήσαμε προηγούμενος. Το συγκεκριμένο είναι τύπου string, επομένως η αντίστοιχη μεταβλητή μπορεί να λάβει μόνον τιμές κειμένου. Η διαφορά με την πρώτη γραμμή, στην οποία αναθέτουμε στην μεταβλητή απευθείας τύπο string είναι ότι, στην περίπτωση εκτύπωσης της μεταβλητής η “local_value”**,** δεν θα έχει επικεφαλίδα, ενώ η “local_data_element” θα πάρει την επικεφαλίδα που έχει δηλωθεί στο αντίστοιχο data element.

```
data local_value type string.
data local_int type i.
data local_data_element type zdt_data_element.
```

```
data: local_value        type string,
      dlocal_int         type i,
      local_data_element type zdt_data_element.
```

Εικόνα 2-5: Δήλωση μεταβλητών

Πρώτού περάσουμε στη δήλωση πινάκων, βασική προτεραιότητα είναι η κατανόηση της δημιουργίας τύπων (types). Όπως είδαμε και με τα data elements, στα οποία αναθέταμε ένα domain με τύπο, έτσι και μέσα στον κώδικα υπάρχει επιλογή να δημιουργήσουμε έναν τύπο και να τον αναθέσουμε σε μία μεταβλητή. Για την δημιουργία ενός τύπου, ξεκινάμε με το keyword “types” και όπως παρατηρούμε στην πρώτη γραμμή της ***Εικόνας 2-6***, δημιουργούμε ένα τύπο ο όποιος θα είναι δεκαδικός (το **p** προέρχεται από το packed number) με μέγεθος 5 και 2 δεκαδικούς χαρακτήρες. Αυτός ο τύπος μπορεί να χρησιμοποιηθεί οπουδήποτε εντός του κώδικα όπως και γίνεται στην ακριβώς επομένη γραμμή, όπου δημιουργούμε μια μεταβλητή βασισμένη στον τύπο που μόλις φτιάξαμε.

```
types type_var_with_decimals type p length 5 decimals 2.
data var_with_decimals type type_var_with_decimals.
```

Εικόνα 2-6: Δημιουργία τύπου και δήλωση

Παρόμοια είναι και η διαδικασία για την δημιουργία structure**.** Για την δημιουργία ενός structure, πρέπει να χρησιμοποιήσουμε το επιπρόσθετο keyword “begin of”, το οποίο θα κλείνει με το αντίστοιχο “end of” και στο ενδιάμεσο υπάρχουν τα πεδία που θέλουμε να προσθέσουμε. Στις πρώτες τέσσερις γραμμές της παρακάτω εικόνας (***Εικόνας 2-7***), φαίνεται η δημιουργία ενός structure με όνομα “struct_customer” και τα αντίστοιχα πεδία του, “id” και “name”. Η μεταβλητή “customer_line” θα κληρονομήσει τον τύπο του structure, δηλαδή θα περιέχει μέσα πεδίο “id” και “name”, ενώ η μεταβλητή “customer_internal_table” θα καταχωρηθεί ως πίνακας. Για την δήλωση πίνακα είναι απαραίτητη η χρήση της επιπρόσθετης λέξης κλειδιού “table of”, ενώ πριν, η δήλωση του είδους του πίνακα.

Τα είδη πινάκων στο SAP είναι τα: **standard** (χρησιμοποιείται στο 95% των περιπτώσεων), **sorted** για ταξινομημένους πίνακες και **hashed** για μεγάλους πίνακες με χρήση κλειδιού. Επομένως, όπως φαίνεται στην τελευταία γραμμή, δηλώνεται ένας πίνακας ο οποίος θα αποτελείται από τα πεδία που υπάρχουν στο structure “struct_customer”.

```
types: begin of struct_customer,
         id   type i,
         name type string,
       end of struct_customer.
data customer_line type struct_customer.
data customer_internal_table type standard table of struct_customer.

field-symbols: <customer> type struct_customer.
data customer_reference type ref to struct_customer.
```

Εικόνα 2-7: Δημιουργία structure, πίνακα και άλλων τύπων

Πέρα από τον κλασσικό τύπο μεταβλητων υπάρχουν και άλλοι δύο: το field-symbol και η αναφορά (type ref to). Το field-symbol, επιτρέπει την ανάθεση τιμών σε μεταβλητές δυναμικά και θυμίζει την χρήση των pointers άλλων γλωσσών προγραμματισμού. Ακολουθούν ένα ιδιαίτερο πρότυπο ονομασίας, χρησιμοποιόντας τους τελεστές “<” και “>” πριν και μετά το όνομα. Η δηλωση μεταβλητών με αναφορά δεν διαφέρει σε πολλά με τις προηγούμενες δηλώσεις και δίνει την δυνατότητα να

χρησιμοποιηθεί η μεταβλητή σαν αντικείμενο. Δηλαδή, αντι για την χρήση "-" για την πρόσβαση σε πεδίο, χρησιμοποιείται το συμβολο "→". Περισσότερα για τα αντικείμενα εξηγούνται παρακάτω.

2.2.7.2 Έλεγχος και επανάληψη

Πέρα από το κομμάτι των δηλώσεων, όπως σε όλες τις γλώσσες προγραμματισμού έτσι και στην ABAP υπάρχουν εντολές ελέγχου και επανάληψης. Οι εντολές "**if**", "**loop at**", "**do n times**" και "**while**" χρησιμοποιούνται αρκετά συχνά. Όπως προ-αναφέρθηκε, όποια εντολή δέχεται εμφωλέυμενες εντολές τότε υπάρχει και αντίστοιχο keyword τερματισμού. Δηλάδη, στην χρήση της "if" θα χρησιμοποιήσουμε στο τέλος και το keyword "endif", ενώ στην "loop at" το "endloop", όπως φαίνεται στην ***Εικόνα 2-8***. Σε άλλες γλώσσες όπως η java δεν συναντάμε keyword που τερματίζει το μπλοκ κώδικα αλλα ο εμφωλευμένος κώδικας αναπτύσεται εντώς αγκυλών.

```
"simple variable
loop at customer_internal_table into data(customer).
endloop.

"assigning to field-symbol
loop at customer_internal_table assigning field-symbol(<customer>).
endloop.

"pass by reference
loop at customer_internal_table reference into data(customer_reference).
endloop.
```

Εικόνα 2-8: Loops

Η χρήση μίας λούπας βοηθάει στην πρόσβαση των στοιχείων ενός πίνακα. Κάθε επανάληψη αποθηκεύει προσωρινά ένα στοιχείο του πίνακα σε μία τοπική μεταβλητή. Στην ABAP υπάρχουν τρεις διαφορετικοί τρόποι που μπορούμε να αναθέσουμε τιμή σε μία μεταβλητή κατά την χρήση λούπας. Ο πιο σύνηθες είναι ο πρώτος "…into data(customer)", όπου στην μεταβλητη "customer" αποθηκευεται η αντίστοιχη προσωρινή τιμή επανάληψης της λούπας. Ο δεύτερος τρόπος είναι η χρήση των field-symbol, που επιτρέπουν την ανάθεση τιμών σε μεταβλητές, με δυναμικό τρόπο. Ο τρίτος τρόπος περιλαμβάνει την ανάθεση τιμής με αναφορά "… reference into data(customer_reference)".

2.2.7.3 Χρήση SQL

Ένα μεγάλο πλεονέκτημα της ABAP είναι η ενσωματωμένη SQL. Όπως αναφέρθηκε πιο πάνω, υπάρχει εμφωλευμένο ένα interface που μεταφράζει τις εντολές SQL, σε native SQL εντολές. Αυτή η ενσωμάτωση καθιστά την εξαγωγή, ενημέρωση και διαγραφή δεδομένων από την βάση δεδομένων ευκολότερη και άμεση. Έχοντας μία βασική εξοικείωση με κάθε μορφής SQL ο παρακάτω κώδικας δεν θα είναι δύσκολος στην κατανόηση.

```
select flights~carrid, flights~connid, flights~fldate,
       bookings~bookid, bookings~customid
    from sflight as flights
    inner join sbook as bookings on bookings~carrid = flights~carrid
                                and bookings~connid = flights~connid
    into table @data(reservations)
    where flights~fldate < @sy-datum
      and flights~carrid = 'LH'.
```

Εικόνα 2-9: Χρήση ενσωματωμένης SQL

Οι διαφορές που ίσως να παρατηρούσαμε σε σχέση με την native SQL είναι, η χρήση περισπωμένης (~), αντί της τελείας (.), που είναι απολύτως λογικό να αντικατασταθεί διότι όπως αναφέρθηκε η τελεία είναι σύμβολο ολοκλήρωσης εντολής, και το παπάκι (@), το οποίο χρησιμοποιείται αποκλειστικά για την χρήση μεταβλητών εντώς μιας SQL εντολής. Η μεταβλητή “sy-datum” υπάρχει στο structure “sy”, όπου πήρε την ονομασία του από τη λέξη system, και είναι κατοχυρωμένο από το SAP περιέχοντας χρήσιμες πληροφορίες κατά την εκτέλεση του προγράμματος, όπως την ημερομηνία, όνομα χρήστη, ώρα, id πελάτη και πολλά άλλα. Το πεδίο “datum” αναφέρεται στην ημερομηνία.

Εν συντομία, η παραπάνω SQL συνθήκη θα μας επιστρέψει μερικά πεδία από τους πίνακες “sflight” και “sbook” (υπάρχουν ενσωματομένοι σε κάθε σύστημα SAP και χρησιμοποιούνται για σκοπούς εκμάθησης) όπου η ημερομηνία θα είναι μικρότερη από την σημερινή και το πεδίο “carrid” (κωδικός αεροπορικής εταιρίας) του πινακα “flights” να είναι ίσο με ‘*LH’* αποθηκευοντας τα αποτελεσματα στον πίνακα “**reservations**”.

2.2.7.4 Κλάσεις και αντικείμενα

Η δημιουργία μιας κλάσης ξεκινάει με το keyword “class” και τελείωνει με το “endclass”. Απαιτούνται δύο τμήματα κώδικα, ένα για τις δηλώσεις (definition) και ένα για την υλοποίηση (implementation). Δεξία από το definition όπως φαίνεται και στην ***Εικόνα 2-10***, με το keyword “create public/private” δηλώνουμε την ορατότητα του κατασκευαστή της κλάσης. Στο τμήμα των δηλώσεων, υποχρεούμαστε να δηλώσουμε τα attributes και τις μεθόδους. Στις μεθόδους πρέπει να αναφέρουμε και τις παραμέτρους καθώς και τον σκοπό τους. Οι τύποι παραμέτρων είναι οι:

- **Importing**, για την χρήση μίας μεταβλητής χωρίς την δυνατότητα επεξεργασίας
- **Changing**, χρήση μεταβλητής με δυνατότητα επεξεργασίας
- **Returning**, επιστροφή μεταβλητής μετά την ολοκλήρωση της μεθόδου

Αφού δηλώσουμε τις μεθόδους και τις παραμέτρους τους, στο αντίστοιχο τμήμα κώδικα για την υλοποίηση (implementation) πρέπει να ανάπτυξουμε το σώμα τους.

Ο κατεσκευαστής, εφόσον θέλουμε να προσθέσουμε σώμα, πρέπει να δηλωθεί στο τμήμα των δηλώσεων, αλλιώς μπορεί να παραληφθεί. Όπως και σε όλες τις γλώσσες προγραμαμτισμού, ο κατασκευαστής καλείται με την δημιουργία ενός αντικειμένου.

Στην ***Εικόνα 2-10*** και στην μέθοδο “set_last_name” φαίνεται να χρησιμοποιείται ένα keyword “me”. Το keyword αυτό έχει να κάνει με την αναφορά του αντικειμένου. Στην java υπάρχει το αντίστοιχο “this”.

Για την δημιουργία ενός αντικειμένου υπάρχουν δύο τρόποι. Ο πρώτος, και περισσότερο συνηθισμένος, είναι η δήλωση μιας μεταβλητής (***Εικόνα 2-11***) με παρόμοιο τρόπο όπως είδαμε και παραπάνω, με την μοναδική διαφορά, αντί της χρήσης του keyword “type”, έχουμε το “type ref to”. Δηλαδή γίνεται μία αναφορά στην αντίστοιχη κλάση. Δεν αρκεί όμως μόνο αυτό για την δημιουργία του αντικειμένου, απαιτείται η χρήση του “new #( )”. Ο δεύτερος τρόπος είναι πιο σύντομος και δημιουργεί απευθείας το αντικείμενο αναθέτοντάς το σε μία μεταβλητή. Στην εικόνα παρατηρείται ότι χρησιμοποιείται το keyword “**new** z_person**( )**”, αντί για το “#” υπάρχει το όνομα της κλάσης. Εφόσον δεν υπάρχει δήλωση της μεταβλητής, τότε είμαστε υποχρεωμένοι να αντικαταστήσουμε το “#” με το όνομα της κλάσης.

```
class z_person definition create public.

  public section.
    constants: class_name type string value 'z_person'.

    class-methods:
      print_class_name.

    methods:
      constructor
        importing name type string,
      set_last_name
        importing last_name type string,
      get_last_name
        returning value(return) type string.

  private section.
    data name type string.
    data last_name type string.

endclass.

class z_person implementation.

  method print_class_name.
    write: class_name.
  endmethod.

  method constructor.
    me->name = name.
  endmethod.

  method set_last_name.
    me->last_name = last_name.
  endmethod.

  method get_last_name.
    return = last_name.
  endmethod.

endclass.
```

Εικόνα 2-10: Ανάπτυξη κλάσης

Η κλήση των μεθόδων πραγματοποιείται με το σύμβολο "→", άμα η μέθοδος είναι στατική τότε χρησιμοποιείται το "=>". Για την αξιοποίηση της επιστρέψιμης τιμής (returning) της μεθόδου "get_last_name" (***Εικόνα 2-10***), μπορούμε αναθέσουμε την τιμή που επιστρέφεται απευθείας σε μια μεταβλητή.

```
"declaration way
data person1 type ref to z_person.
person1 = new #( 'John' ).

"in-line way
data(person2) = new z_person( 'George' ).

person2->set_last_name( 'Papadopoulos' ).
data(last_name) = person2->get_last_name( ).

z_person=>print_class_name( ).
```

Εικόνα 2-11: Δημιουργία αντικειμένων

2.3 Ανάλυση κώδικα ABAP

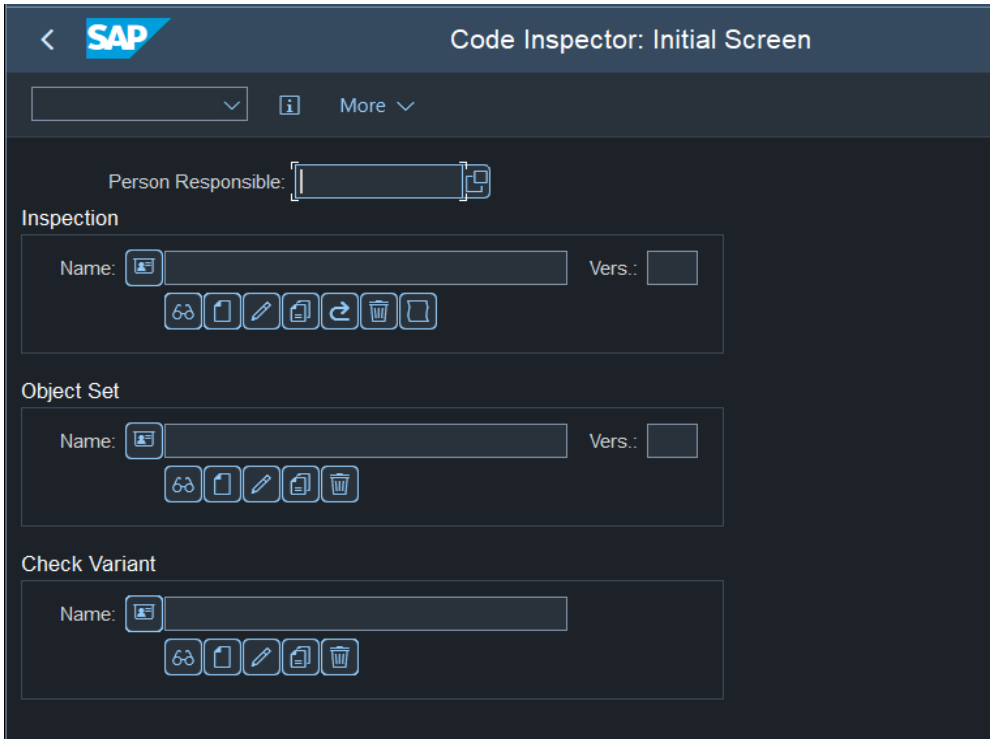
Το SAP πέρα από το να προσφέρει λύσεις στον πελάτη, έχει δημιουργήσει και εργαλεία που απευθύνονται σε προγραμματιστές. Πέρα από τον κλασσικό editor για την ανάπτυξη προγραμμάτων, κλάσεων και function modules, έχει δημιουργήσει και εργαλεία για ανάλυση κώδικα και βελτιστοποίηση. Στις επόμενες παραγράφους θα περιγράψουμε τρία από τα πιο σημαντικά προγράμματα για ανάλυση και αξιολόγηση κώδικα, τα οποία έπαιξαν και καθοριστικό ρόλο στην ολοκλήρωση της παρούσας μελέτης.

2.3.1 SAP Code Inspector (SCI)

Το "SAP Code Inspector", εν συντομία **SCI**, είναι ένα εργαλείο που έχει δημιουργηθεί από το SAP και προσφέρει τον έλεγχο των αντικειμένων (προγράμματα, κλάσεις, Function Modules), ως προς την απόδοση, την ασφάλεια, την χρήση σωστού συντακτικού καθώς και την χρήση προτύπων ονομασίας μεταβλητών. Για την χρήση του, εκτελούμε το transaction **SCI.**

Η αρχική του οθόνη μας προσφέρει τρείς επιλογές:

1) **Inspection**,
2) **Object Set**
3) **Check Variant** (***Εικόνα 2-12***).

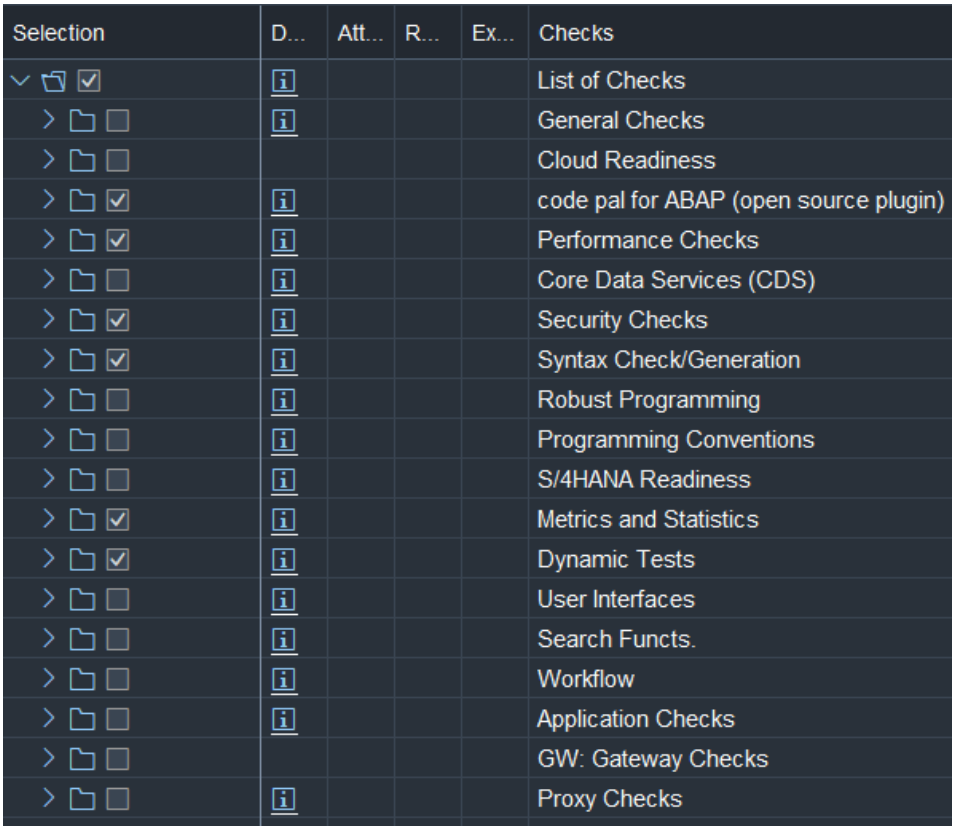


Εικόνα 2-12: Οθόνη εργαλείου "Code Inspector"

Η επιλογή **Check Variant** (***Εικόνα 2-13***), η πιο σημαντική λειτουργία σε αυτό το πρόγραμμα, έχει να κάνει με την προτίμηση ελέγχων (checks) που μπορούν να εφαρμοστούν στα αντικείμενα. Πιο αναλυτικά, το transaction SCI μας προσφέρει μία λίστα ελέγχων όπως, ελέγχους για:

- **Απόδοση**: περιλαμβάνει ελέγχους για εμφωλευμένες λούπες, πολλαπλές "if" μέσα σε λούπες, συχνά αιτήματα στη βάση δεδομένων, "select" χωρίς "where" συνθήκες, λάθος φιλτράρισμα σε πίνακες και πολλά άλλα.
- **Ασφάλεια**: ελέγχους για το εάν ο χρήστης έχει τους απαραίτητους ρόλους για να χρησιμοποιήσει το πρόγραμμα, ελέγχους για τροποποίηση, διαγραφή δεδομένων από πίνακα χωρίς ύπαρξη αυθεντικοποίησης του χρήστη.
- **Συντακτικό**: έλεγχος για ξεπερασμένες εντολές.
- **Ονομασία μεταβλητών**: ελέγχει εάν τα ονόματα μεταβλητών ακολουθούν τα επιλεγμένα πρότυπα.
- **Μετρικές**: εξαγωγή μετρικών όπως, πολυπλοκότητα, αριθμός σχολίων.
- **Unit Tests**: εκτέλεση (εάν υπάρχουν) unit tests

Βάσει των διαθέσιμων επιλογών, δημιουργούμε ένα variant, το οποίο θα μπορέσει να χρησιμοποιηθεί σε μελλοντικούς ελέγχους σε αντικείμενα.



Εικόνα 2-13: Οθόνη επιλογής ελέγχων

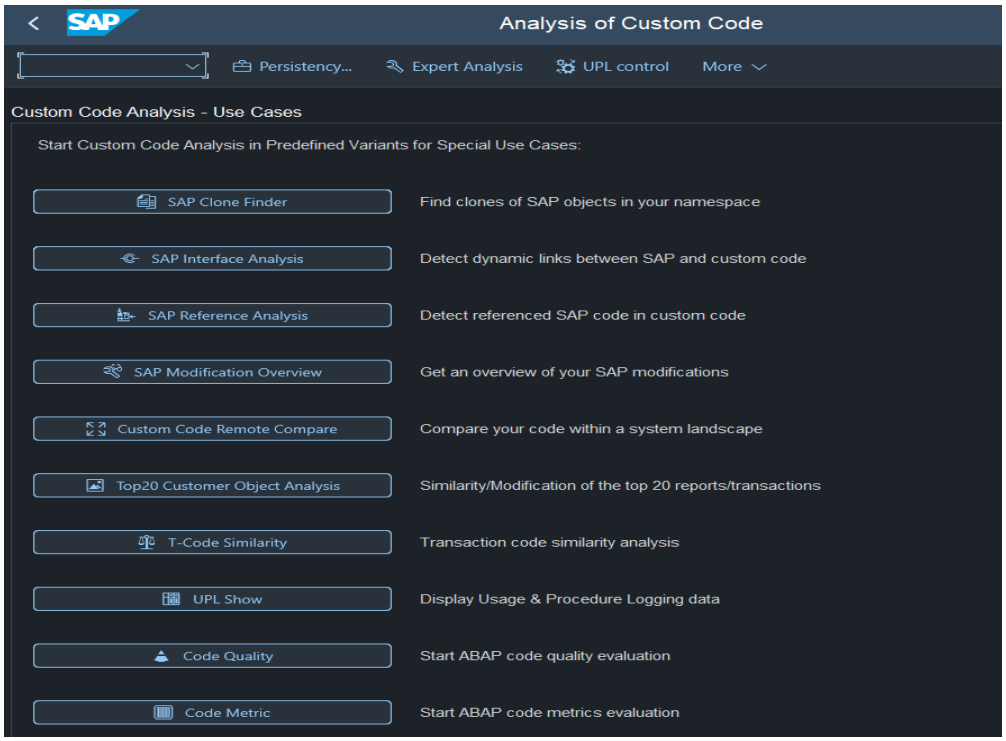
Όσον αφορά την επιλογή **Object Set,** μπορούμε να δημιουργήσουμε ένα γκρουπ με αντικείμενα, να προσθέσουμε πακέτα, κλάσεις, functions κτλ. Αντίθετα, με την επιλογή **Inspection** μπορούμε να συνδυάσουμε ένα **Object Set** και ένα **Variant** και να τρέξουμε τους ελέγχους. Στο τέλος λαμβάνουμε χρήσιμες πληροφορίες, τις οποίες πρέπει να λάβουμε υπόψιν προς διόρθωση. Κάθε φορά που εκτελούμε έλεγχο με την επιλογή **Inspection**, δημιουργούμε μία καινούργια έκδοση του ελέγχου, με αποτέλεσμα να χτίζουμε ένα ιστορικό. Το ιστορικό μπορεί να χρησιμεύσει στην ανίχνευση σφαλμάτων που διορθώθηκαν η παρέμειναν ως έχει.

2.3.2 "Analysis of Custom Code" tool

Το εργαλείο "**Analysis of Custom Code**" (transaction **/SDF/CD_CCA**) περιέχει πολλά μικρά εργαλεία για την ανάλυση, σύγκριση και έλεγχο του κώδικα. Μερικά από τα πιο χρήσιμα είναι, το "**SAP Clone Finder**" το οποίο εντοπίζει κλώνους αντικειμένων

στο σύστημα, αρκετά εύχρηστη λειτουργία, καθώς αρκετές φορές στο SAP μερικά reports για προβολή δεδομένων που δημιουργούνται από προγραμματιστές μοιάζουν μεταξύ τους. Το εργαλείο αυτό, μπορεί να βοηθήσει στον εντοπισμό τους για την δημιουργία ενός κοινού προγράμματος και την εξαγωγή δεδομένων.

Ένα παρόμοιο εργαλείο είναι το "**T-Code Similarity**", που εντοπίζει ομοιότητες όσον αφορά την γραφική απεικόνιση των προγραμμάτων. Το εργαλείο "**Code Quality**", βασίζεται στο εργαλείο "**SAP Code Inspector (SCI)**" της προηγούμενης παραγράφου. Περιέχει ιστορικό από ελέγχους που έλαβαν μέρος σε προγράμματα, προγραμματισμό καινούργιων ελέγχων σε συγκεκριμένη μέρα και ώρα ή ανά συγκεκριμένη χρονική περίοδο όπως επίσης και αποστολή των αποτελεσμάτων με e-mail. Το εργαλείο "**Code Metrics**" αναλύεται στην παρακάτω παράγραφο.



Εικόνα 2-14: Οθόνη εργαλείου "Analysis of Custom Code"

2.3.3 "Code Metrics" tool

Το "**SAP Code Metrics**" είναι ένα εργαλείο που μπορεί να χρησιμοποιηθεί για την ανάλυση του πηγαίου κώδικα και την παροχή πληροφοριών σχετικά με την ποιότητα και τη δυνατότητα συντήρησής του. Μπορεί να χρησιμοποιηθεί για τον έλεγχο της

συμμόρφωσης με τα πρότυπα κωδικοποίησης, τον εντοπισμό πιθανών σημείων συμφόρησης στην απόδοση και τον εντοπισμό επαναλαμβανόμενου κώδικα. Το εργαλείο μπορεί επίσης να χρησιμοποιηθεί για τη δημιουργία λεπτομερών αναφορών (reports), τα οποία με την σειρά τους, ενδέχεται να συμβάλλουν στην αξιολόγηση του κώδικα και την διαδικασία ανακατασκευής (refactoring) του.

Το πρόγραμμα, βασίζεται στο εργαλείο ανάλυσης ανοιχτού κώδικα, "**abapLint**", το οποίο είναι γραμμένο σε Python. Το εργαλείο χρησιμοποιεί ένα σύνολο προκαθορισμένων κανόνων και ελέγχει τον κώδικα βάσει αυτών. Μπορεί να ρυθμιστεί ώστε να ελέγχει συγκεκριμένα ζητήματα, όπως η χρήση ξεπερασμένων εντολών (obsolete) ή την παρουσία hard-coded τιμών (κείμενο που χρησιμοποιείται εντός εισαγωγικών, χωρίς την χρήση μεταβλητής για την στέγαση του). Η αναγνωσιμότητα, η δυνατότητα συντήρησης και η απόδοση είναι βασικοί παράγοντες στη ανάπτυξη λογισμικού και πρέπει να λαμβάνονται υπόψιν σε κάθε μορφής έργου.

Είναι σημαντικό να σημειωθεί ότι το εργαλείο δεν υποκαθιστά τους μη αυτόματους ελέγχους κώδικα, αλλά μπορεί να χρησιμοποιηθεί ως συμπλήρωμα και να βοηθήσει στον πιο αποτελεσματικό εντοπισμό πιθανών ζητημάτων. Επιπρόσθετα, μπορεί να βρεθεί στο transaction "**Analysis of Custom Code**" (**/SDF/CD_CCA**) ως "Code Metric", ενώ το δικό του transaction είναι "**/SDF/CD_CUSTOM_CODE_METRIC**".

Το εργαλείο έχει τη δυνατότητα να εξάγει μερικές βασικές μετρικές όπως επίσης και το σύνολο χρήσης των keywords. Οι μετρικές που εξάγει είναι οι:

- **Lines of Code (LoC) -** σύνολο των γραμμών κώδικα και των σχολίων,
- **Number of Statements (NoS) -** το σύνολο των δηλώσεων,
- **Number of Comments (NoC) -** το σύνολο των σχολίων,
- **Number of Pragmas (NoP) -** το σύνολο των pragmas (τα pragmas έχουν παρόμοια χρήση με τα annotations που συναντάμε σε άλλες γλώσσες προγραμματισμού),
- **Complexity of Conditions (COM) -** τη πολυπλοκότητα των συνθηκών (if, loops, while, case), μετράει τον αριθμό των λογικών τελεστών AND, OR κτλ.,
- **Number of Authors -** το σύνολο των προγραμματιστών που έχουν συμμετάσχει στην υλοποίηση του κώδικα,
- **Complexity Weighted by Decision Depth (DD) -** το βάθος των συνθηκών - πόσες συνθήκες εμφωλευμένες συναντάμε.

2.4 Enhancements και εξαγωγή στην μνήμη (cache)

Για την κατανόηση της μεθοδολογίας που ακολουθεί στο επόμενο κεφάλαιο, είναι απαραίτητο να αναφερθούμε στην έννοια των "enhancements" καθώς και στην εξαγωγή στην μνήμη.

Το SAP δεν επιτρέπει την τροποποίηση κώδικα στο έτοιμο λογισμικό που προσφέρει (πυρήνας), αλλά επιτρέπει την προσθήκη κώδικα συνήθως στην αρχή και στο τέλος κάθε μεθόδου, αυτό ονομάζεται "**enhancement**". Συνήθως χρησιμοποιούνται από τους προγραμματιστές για μικρές αλλαγές στην συμπεριφορά των προγραμμάτων ή για προσθήκη λογικής.

Όσον αφορά την εξαγωγή στην μνήμη, είναι αρκετά συχνό φαινόμενο κατά την εκτέλεση προγραμμάτων του SAP, ένα πρόγραμμα να εκτελεί ένα άλλο και το πρώτο στην σειρά να χρειάζεται πληροφορίες από το δεύτερο για να συνεχίσει την εκτέλεση του. Ο μόνος τρόπος για να επιτευχθεί το παραπάνω γεγονός, είναι η εξαγωγή αποτελεσμάτων στην μνήμη. Υπάρχει η δυνατότητα να εξάγουμε μεταβλητές, structures ή πίνακες στην μνήμη, τα οποία διαγράφονται με το που ολοκληρωθεί το session. Για να γίνει κατανοητή η παραπάνω λειτουργία, ας θέσουμε ένα παράδειγμα. Έστω το πρόγραμμα **A** που τυπώνει μερικές πληροφορίες στην οθόνη, και το πρόγραμμα **B** που σκανάρει στην βάση δεδομένων και τρέχει μερικούς υπολογισμούς. Το πρώτο πρόγραμμα χρειάζεται τα αποτελέσματα του προγράμματος **B** για να ολοκληρώσει την εκτέλεση του. Για να συμβεί αυτό, κατά την διάρκεια του προγράμματος **A**, θα καλεστεί το πρόγραμμα **B** (το πρόγραμμα **A** αναμένει την ολοκλήρωση του **B**) και αυτό προτού τερματιστεί θα εξάγει τα αποτελέσματα στην μνήμη με ένα στατικό ID. Με την ολοκλήρωση του **B**, το **A** θα εισάγει τα αποτελέσματα από την μνήμη με την χρήση του ίδιου στατικού ID και θα προχωρήσει στη τύπωση τον αποτελεσμάτων.

```
export object from variable to memory id memory_id.
"""""""""""""""""""""""""""""""""""""""""""""""""""
import object to other_variable from memory id memory_id.
```

Εικόνα 2-15: Εξαγωγή και εισαγωγή μεταβλητών στην μνήμη

Στην παραπάνω εικόνα φαίνονται οι δύο εντολές που χρησιμοποιούνται για εξαγωγή και εισαγωγή. Η "**export**" εξάγει στην μεταβλητή με όνομα "**object**" το περιεχόμενο της μεταβλητής "**variable**" με ID το περιεχόμεο της μεταβλητής

“**memory_id**”. Η “**import**”, εισάγει την μεταβλητεί “**object**” στην τοπική μεταβλητη “**other_variable**” από το την μνήμη με το ιδιο ID. Απαραίτητη είναι η χρήση του ίδιου ονόματος του αντικειμένου που εξάγεται, όπως το “**object**” του παραπάνω παραδείγματος.

3 Μεθοδολογία

Στο παρόν κεφάλαιο, περιγράφεται η αρχιτεκτονική του προγράμματος υπολογισμού μετρικών που υλοποιήθηκε στα πλαίσια της συγκεκριμένης έρευνας, καθώς και η μεθοδολογία που ακολουθήθηκε για την ανάπτυξή του σε σύστημα SAP. Τέλος παρουσιάζονται και σχολιάζονται τα ευρήματα από την εκτέλεση του προγράμματος σε συνδυασμό με κάποιες στατιστικές μελέτες.

3.1 Αρχιτεκτονική και σχεδιαστικές αποφάσεις

Για την κατανόηση της ροής της μεθοδολογίας, πρέπει πρώτα να επεξηγηθούν οι κλάσεις που δημιουργήθηκαν, ο τρόπος με τον οποίον δομήθηκαν, το περιεχόμενό τους καθώς και τα design patterns που έπαιξαν σημαντικό ρόλο στην σχεδίαση.

Το πρόγραμμα μαζί με όλα τα απαραίτητα αντικείμενα για την σωστή λειτουργία του, αποθηκεύτηκε σε ένα πακέτο με όνομα "**z_abap_metrics**". Το βασικό αυτό πακέτο, χωρίστηκε σε 4 υπό-πακέτα με το καθένα από αυτά να έχει τον δικό του ρόλο στην δομή το προγράμματος. Τα πακέτα, οι κλάσεις και τα interfaces που περιέχονται είναι τα παρακάτω:

- Πακέτο **z_program**:
    - Enhancement **zei_export_source_code**: υλοποιήθηκε για την εξαγωγή πληροφοριών από τον standard πρόγραμμα "Code Metric".
    - Πρόγραμμα **z_abap_metrics**: το κύριο πρόγραμμα για τον υπολογισμό των μετρικών και την εμφάνιση των αποτελεσμάτων.
    - Κλάση **z_class_manager:** διαχειρίζεται την εξαγωγή πληροφοριών από το standard πρόγραμμα και την εισαγωγή στο κύριο πρόγραμμα.
    - Κλάση **z_salv_output**: αναλαμβάνει την προβολή των τελικών αποτελεσμάτων.
    - Κλάση **z_popup_window**: δημιουργεί ένα popup παράθυρο στην περίπτωση εισαγωγής λάθος πληροφοριών στην αρχική οθόνη.
    - Κλάση **z_progress_indicator**: τυπώνει στο κάτω μέρος της οθόνης το ποσοστό ολοκλήρωσης κατά την εκτέλεση του προγράμματος. Βασίζεται στο πρότυπο "*Singleton*".

- Interface **zif_popup_window:** περιέχει πληροφορίες που κληρονομεί η κλάση **z_popup_window**.
- Data element **zdt_class_name**: για το όνομα της κλάσης.
- Data element **zdt_package**: για το όνομα του πακέτου
- Data element **zdt_method_name**: για τη μέθοδο.
- Data element **zdt_lines_of_code**: για την μέτρηση γραμμών κώδικα.
- Data element **zdt_number_of_comments**: για τα σχόλια.
- Data element **zdt_number_of_statements**: για τα statements.
- Data element **zdt_number_of_authors**: για τους συγγραφείς.
- Data element **zdt_complex_of_conditions**: για την περιπλοκότητα των συνθηκών.
- Data element **zdt_compex_weight**: για το βάθος των εμφωλευμένων συνθηκών.
- Data element **zdt_coupling_bt_objects**: για την σύζευξη των αντικειμένων.
- Data element **zdt_lack_of_cohesion**: για την έλλειψη συνοχή των μεθόδων.
- Structure **zst_abap_metrics**: το structure για την εμφάνιση των αποτελεσμάτων που περιέχει όλα τα παραπάνω data elements.

- Πακέτο **z_models**:
  - Κλάση **z_class**: αντιπροσωπεύει το μοντέλο για κάθε κλάση που ελέγχεται στο πρόγραμμα για τον υπολογισμό τον μετρικών.
  - Κλάση **z_methods**: η κλάση-μοντέλο για τη σωστή διαχείριση των μεθόδων κάθε κλάσης. Ένας πίνακας με αντικείμενα **z_methods** υπάρχει στην κλάση **z_class**.
  - Κλάση **z_metrics**: περιέχει όλες τις μετρικές που θα υπολογιστούν από το κύριο πρόγραμμα. Επίσης, κληρονομείται από την **z_methods**, καθώς όλοι οι υπολογισμοί γίνονται σε επίπεδο μεθόδου.
  - Κλάση **z_keywords**: σχεδιάστηκε για να συγκρατεί τα **keywords** της ABAP. Απαραίτητη για τον υπολογισμό της έλλειψης συνοχής.
  - Κλάση **z_variables:** σχεδιάστηκε για την διαχείριση των μεταβλητών κάθε κλάσης. Χρησιμοποιείται επίσης για τον υπολογισμό της έλλειψης συνοχής.

- Πακέτο **z_metrics**:
    - Κλάση **z_code_scanner_factory**: πρόκειται για μια κλάση βασισμένη στο design pattern "*factory*". Δέχεται ως παράμετρο τον πηγαίο κώδικα και μία σταθερά που λειτουργεί ως κλειδί για την ανάλυση του κώδικα. Σκανάρει τον πηγαίο κώδικα και τον τοποθετεί γραμμή προς γραμμή σε ένα πίνακα. Επίσης, αποθηκεύει τα statements και tokens σε ξεχωριστούς πίνακες. Ανάλογα με την σταθερά που δέχεται, θα συγκρατήσει τα σχόλια και τα pragmas, είτε δεν θα τα λάβει υπόψιν είτε θα συγκρατήσει και τα διαθέσιμα structures. Στο τέλος επιστρέφει όλα τα στοιχεία που ανέλυσε.
    - Κλάση **z_code_scanner**: μία super κλάση, για την ανάλυση του πηγαίου κώδικα. Συγκρατεί τα statements και keywords των μεθόδων καθώς και τον πηγαίο κώδικα. Στον κατασκευαστή της, καλεί την κλάση **z_code_scanner_factory** και ανάλογα από ποια κλάση κληρονομείται θα ληφθούν και τα αντίστοιχα αποτελέσματα. Οι περισσότερες από τις παρακάτω κλάσεις κληρονομούν την κλάση αυτή.
    - Κλάση **z_loc_calculator**: μετράει τις γραμμές κώδικα κάθε μεθόδου. Δεν λαμβάνει υπόψιν κενές γραμμές και σχόλια.
    - Κλάση **z_noc_calculator**: μετράει τα σχόλια εντός των μεθόδων.
    - Κλάση **z_nos_calculator**: μετράει τα διαθέσιμα statements των μεθόδων.
    - Κλάση **z_complex_calculator**: υπολογίζει την πολυπλοκότητα των συνθηκών. Για κάθε "and" και "or" σε συνθήκες ελέγχου, αυξάνεται κατά ένα.
    - Κλάση **z_decision_depth_calculator**: σχεδιάστηκε για τον υπολογισμό των εμφωλευμένων συνθηκών.
    - Κλάση **z_authors_calculator**: μετράει τους προγραμματιστές που έχουν προσθέσει-τροποποιήσει κώδικα σε κάθε μέθοδο.
    - Κλάση **z_cbo_calculator**: αναπτύχθηκε για να υπολογίζει την σύζευξη μεταξύ αντικειμένων.
    - Κλάση **z_cohesion_calculator**: δημιουργήθηκε για να υπολογίζει την έλλειψη συνοχής σε κάθε μέθοδο.

- Interface **zif_metrics**: περιέχει σταθερές που χρησιμοποιούνται σε όλο το πακέτο.
- Κλάση **z_calc_metrics_facade**: βασίζεται στο design pattern "*façade*", περιέχει την δημιουργία των αντικειμένων όλων των μετρικών και τον υπολογισμό των αποτελεσμάτων για την ευκολότερη διαχείριση.

- Πακέτο **z_exceptions**:
  - Κλάση **zcx_metrics_error**: μία κλάση exception που σχεδιάστηκε για την διαχείριση σφαλμάτων στις κλάσεις που υπολογίζουν μετρικές, δηλαδή του πακέτου **z_metrics**.
  - Κλάση **zcx_flow_issue**: μία κλάση exception για την διαχείριση σφαλμάτων κατά την ροή του προγράμματος, δηλαδή στο πακέτο **z_program**.
  - Interface **zif_exception_messages**: ένα interface που σχεδιάστηκε για να συγκρατεί τα exception μηνύματα που χρησιμοποιούν οι παραπάνω δύο κλάσεις.

3.2 Αξιοποίηση έτοιμων εργαλείων

Το πρώτο και πιο σημαντικό κομμάτι ήταν η εύρεση ενός σημείου στον ήδη υπάρχοντα κώδικα των εργαλείων του SAP, για τα οποία αναφορά έγινε σε προηγούμενο κεφάλαιο. Το εργαλείο "Code Metric" έπαιξε σημαντικό ρόλο στην εξαγωγή πληροφοριών. Μετά από μελέτη του κώδικα, εντοπίστηκε ένα σημείο εντός της τοπικής κλάσης "cl_calc_code_metrics" και πιο συγκεκριμένα στην μέθοδο "calc_code_metric_for_object", όπου φαίνεται να πραγματοποιείται ο υπολογισμός μερικών μετρικών. Με μια δεύτερη ματιά παρατηρήθηκε ότι ο τύπος μίας παραμέτρου της μέθοδού αυτής (i_object), περιείχε το όνομα της κλάσης που θα αναλυόταν. Λίγες γραμμές πιο κάτω, ένα αντικείμενο της κλάσης "cl_object_parser" δημιουργείται και περιέχει πληροφορίες όπως το πακέτο, όνομα κλάσης, όνομα μεθόδου, συγγραφέα και ένα πεδίο(source) με τον πηγαίο κώδικα της μεθόδου.

Με την τοποθέτηση ενός break-point στη μέθοδο αυτή και την εκτέλεση του προγράμματος, παρατηρήθηκε ότι, αναλύεται ο πηγαίος κώδικας της κάθε μεθόδου και

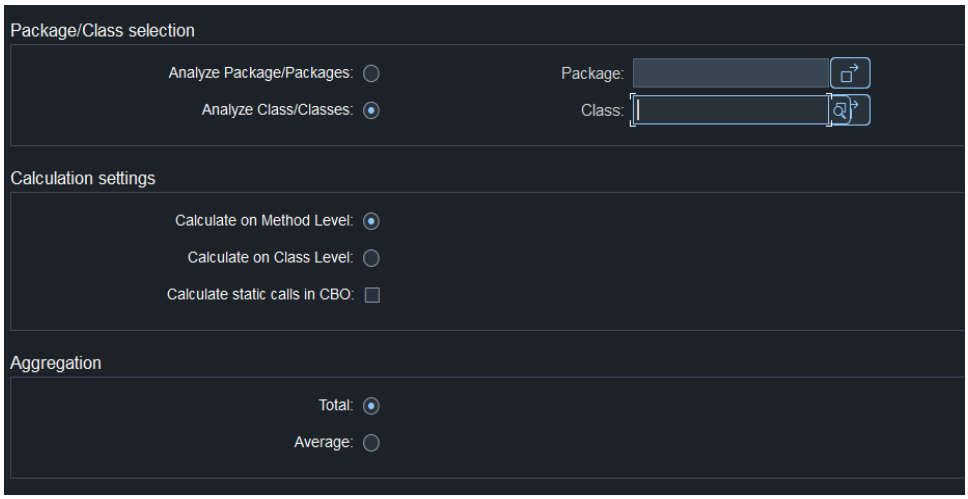
εκτελούνται υπολογισμοί για πληροφορίες όπως οι γραμμές κώδικα (LoC), αριθμός δηλώσεων (NoS), αριθμός σχολίων (NoC), αριθμός συγγραφέων (Authors), πολυπλοκότητα συνθηκών (COM) και πολυπλοκότητα σε βάθος (Depth). Η πιο σημαντική πληροφορία η οποία και χρειάστηκε παρακάτω για τον υπολογισμό των μετρικών, είναι ο πηγαίος κώδικας σε μορφή πίνακα.

Με βάση τις παραπάνω παρατηρήσεις, το σημείο αυτό θεωρήθηκε ιδανικό για εξαγωγή δεδομένων. Στο τέλος της μεθόδου, υλοποιήθηκε ένα **enhancement** το οποίο εξάγει ένα πίνακα με το όνομα του πακέτου, την κλάση, τις μεθόδους της κλάσης και τον πηγαίο κώδικα στην μνήμη. Αυτό το enhancement θα παίξει σημαντικό ρόλο στην λειτουργία του προγράμματος.

3.3 Ανάπτυξη προγράμματος

3.3.1 Αρχική οθόνη και εξαγωγή πηγαίου κώδικα

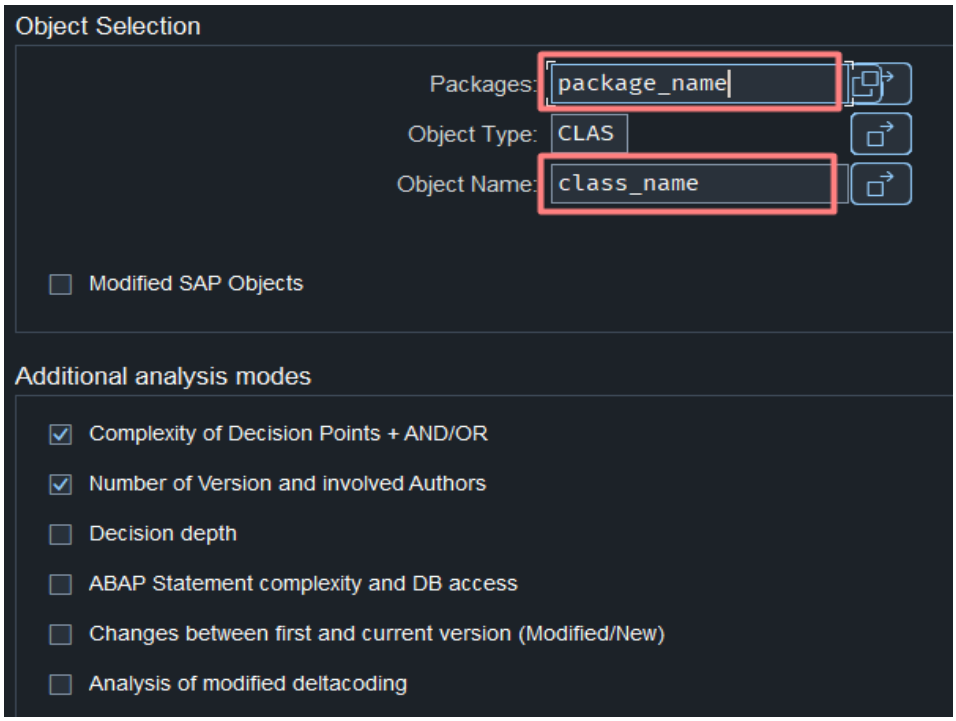
Ξεκινώντας την ανάπτυξη του προγράμματος, το πρώτο τμήμα που υλοποιήθηκε ήταν η αρχική οθόνη στην οποία εισέρχεται ο χρήστης κατά την εκτέλεσή του. Η οθόνη χωρίστηκε σε τρία κομμάτια. Στο πρώτο, ο χρήστης έχει τη δυνατότητα να επιλέξει εάν επιθυμεί την ανάλυση πακέτου ή κλάσης. Φυσικά, παρέχεται και η επιλογή ανάλυσης πολλαπλών πακέτων ή κλάσεων την φορά, με την εκμετάλλευση των λειτουργιών που προσφέρουν τα select-options (είναι μία μορφή παραμέτρων για την αρχική οθόνη που επιτρέπουν την εισαγωγή πολλαπλών τιμών, σε αντίθεση με την απλή παράμετρο που επιτρέπει την χρήση μόνο μίας τιμής). Στο δεύτερο μπλοκ, φαίνονται μερικές ρυθμίσεις που αφορούν τον υπολογισμό των μετρικών. Υπάρχει η επιλογή για ανάλυση κώδικα σε επίπεδο μεθόδου ή σε επίπεδο κλάσης καθώς και επιλογή κατά την οποία ο χρήστης επιθυμεί ή όχι τον υπολογισμό κλήσεων στατικών μεθόδων για την μετρική του CBO (Coupling Between Objects). Τέλος, παρέχεται επιλογή για τον τρόπο με τον οποίο θα εμφανίζονται τα αποτελέσματα ανά κλάση: σε σύνολο ή ο μέσος όρος.



Εικόνα 3-1: Αρχική οθόνη του κύριου προγράμματος

Μετά την σχεδίαση της αρχικής οθόνης, γίνεται ξεκάθαρο το πλάνο για την συνέχεια της υλοποίησης, Το επόμενο βήμα είναι να συλλέξουμε της πληροφορίες που εισάγει ο χρήστης και να καλέσουμε το εργαλείο του SAP “Code Metric” περνώντας τις

πληροφορίες ως παράμετρο. Πιο συγκεκριμένα, το “Code Metric” για να λειτουργήσει σωστά και να λάβουμε τα αποτελέσματα που επιθυμούμε, χρειάζεται να επιλέξουμε τα παρακάτω πεδία από το πρόγραμμα.



Εικόνα 3-2: Αρχική οθόνη του εργαλείου “Code Metric”

Φυσικά, επειδή το πρόγραμμα καλείται στο background, όλα αυτά τα πεδία θα περαστούν με την χρήση πίνακα παραμέτρων. Τα checkboxes αποθηκεύονται στατικά στον πίνακα, όπως και το πεδίο “Object Type”, ενώ τα πεδία “Packages” και “Object Name” αποθηκεύονται δυναμικά, ανάλογα με το τι έχει περάσει ο χρήστης στο κύριο πρόγραμμα και στα πεδία “Package” και “Class” (***Εικόνα 3-1***). Συνοπτικά, τρέχουμε το “Code Metric” εργαλείο κατά κύριο λόγο για την εξαγωγή του πηγαίου κώδικα των μεθόδων.

Με την ολοκλήρωση της δημιουργίας του πίνακα παραμέτρων καλείται το εργαλείο “Code Metric” με την χρήση της παρακάτω εντολής. Ο πίνακας “parameters” περιέχει τις παραμέτρους που περιγράψαμε πιο πάνω.

```
submit /sdf/cd_custom_code_metric exporting list to memory
    with selection-table parameters and return.
```

Εικόνα 3-3: Εντολή κλήσης τρίτου προγράμματος

Αφού εκτελεστεί και ολοκληρωθεί το “Code Metric”, τα αποτελέσματα αποθηκεύονται στην μνήμη τα οποία και πρέπει να εισάγουμε στο κύριο πρόγραμμα. Για κάθε κλάση ή για κάθε πακέτο προς ανάλυση, υπάρχει ένας πίνακας στην μνήμη που μοιάζει με αυτόν της ***Εικόνας 3-4.***

3.3.2 Διαχείριση αποτελεσμάτων Code Metric

DEVCLASS	OBJECT	OBJ_NAME	SUB_NAME	AUTHOR	CHANGER	SOURCE
$TMP	CLAS	ZCL_TEST_LCOM	TEST_SCAN_FACTORY	KSERETIS	KSERETIS	Standard Table[12x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_CONSTANTS_V2	KSERETIS	KSERETIS	Standard Table[9x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_FIELD_SYMBOL	KSERETIS	KSERETIS	Standard Table[11x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_CONSTANTS	KSERETIS	KSERETIS	Standard Table[9x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_THIS	KSERETIS	KSERETIS	Standard Table[8x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_OBJS	KSERETIS	KSERETIS	Standard Table[13x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_NESTED_KEYWORDS	KSERETIS	KSERETIS	Standard Table[17x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_METHOD_CALLS	KSERETIS	KSERETIS	Standard Table[10x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_LOCAL_DECLARATION	KSERETIS	KSERETIS	Standard Table[8x1(8)]
$TMP	CLAS	ZCL_TEST_LCOM	TEST_EXCEL_METHOD	KSERETIS	KSERETIS	Standard Table[17x1(8)]

Εικόνα 3-4: Πίνακας αποτελεσμάτων από το εργαλείο “Code Metric”

Παραπάνω, περιγράφεται η πρώτη φάση της υλοποιησης του προγράμματος που αποτελεί και τον πυρήνα της ανάπτυξης. Από τα αποτελέσματα της ***Εικόνας 3-4***, Η πρώτη στήλη “DEVCLASS” είναι το πακέτο, η στήλη “OBJECT” ο τύπος, η “OBJ_NAME” το ονομα της κλάσης, η “SUB_NAME” το ονομα της μεθόδου, στις επόμενες δύο στήλες υπάχει ο αρχικός συγγραφέας και αυτός που τροποποίησε τελευταίος τον κώδικα ενώ στην τελευταία στήλη “SOURCE” υπάρχει ένας εμφωλευμένος standard πίνακας με τον πηγαίο κώδικα της αντίστοιχης μεθόδου. Στην ***Εικόνα 3-5*** φαίνεται ο πηγαίος κώδικας της μεθόδου “test_field_symbol”.

TABLE_LINE [CString]

```
method test_field_symbol.
  loop at mara_tab assigning field-symbol(<fs_line>).
    if <fs_line>-matnr = 'AD'.
      select single *
          from mara
          into @data(line)
          where matnr = @<fs_line>-matnr.
      line-matkl = 'Product'.
    endif.
  endloop.
endmethod.
```

Εικόνα 3-5: Εμφωλευμένος πίνακας με τον πηγαίο κώδικα

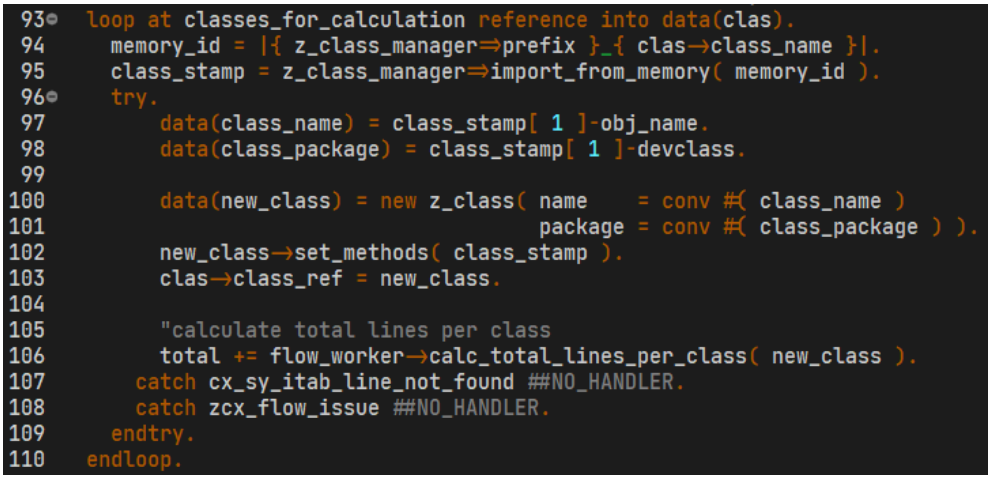
Η δεύτερη φάση του προγράμματος ξεκινάει με την επεξεργασία των παραπάνω δεδομένων. Το πρώτο βήμα είναι να συλεχθούν όλες οι κλάσεις προς ανάλυση. Εάν ο χρήστης έχει πληκτρολογήσει μία ή παραπάνω κλάσεις στο πεδίο “Class” της αρχικής οθόνης τότε απλά αυτές οι κλάσεις προστίθενται στον πίνακα “classes_for_calculation”. Αντιθέτως, άμα ο χρήστης έχει επιλέξει ανάλυση ολόκληρου πακέτου τότε υπάρχει μια διαδικασία παραπάνω. Το πρόγραμμα εντοπίζει τις διαθέσιμες κλάσεις στο πακέτο, καθώς και σε όλα τα υπο-πακέτα και τις προσθέτει στον πίνακα “classes_for_calculation”.

Ο πίνακας “classes_for_calculation”, αποτελεί έναν πίνακα απλού τύπου, με 2 μόνο πεδία, το όνομα της κλάσης και ένα αντικείμο της κλάσης “z_class” (***Εικόνα 3-6***). Χρησιμοποιείται για την προσωρινή αποθήκευση των κλάσεων που πρόκειται να αναλυθούν.

```
types: begin of classes_for_calc_struct,
         class_name type string,
         class_ref  type ref to z_class,
       end of classes_for_calc_struct.
data classes_for_calculation type standard table of classes_for_calc_struct.
```

Εικόνα 3-6: Structure πίνακα και πίνακας των κλάσεων προς ανάλυση

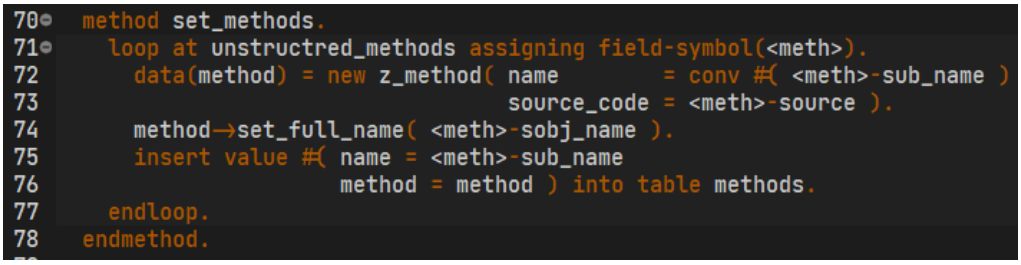
Πιο αναλύτικά (***Εικόνα 3-7***), ακολουθεί μία λούπα στην οποία ελέγχονται ένα προς ένα τα αντικείμενα που επιθυμεί ο χρήστης να αναλυθούν. Εάν υπάρχει αντίστοιχος πίνακας αποθηκευμένος στην μνήμη τότε αποθηκεύεται προσωρινά στην μεταβλητή “class_stamp”. Επίσης αποθηκεύεται το πακέτο και το όνομα της κλάσης στις μεταβλητές “class_package” και “class_name” αντίστοιχα. Στη συνέχεια, δημιουργείται ένα αντικείμενο της κλάσης “z_class” και περνιούνται οι μέθοδοι της αντίστοιχης κλάσης (γραμμή 100 - 102). Ακόμη, ανανεώνεται ο πίνακας “classes_for_calculation” μετά την αποθηκευση του καινούργιου αντικειμένου στην αντίστοιχη γραμμή (γραμμή 103). Τέλος, ανανεώνεται το συνολικό ποσό γραμμών κώδικα, το οποίο θα χρησιμοποιηθεί αργότερα από την κλάση “z_progress_indicator” για την προβολή του ποσοστού ολοκλήρωσης. Φυσικά, άμα οποιαδήποτε από τις παραπάνω εντολές αποτύχει, υπάρχουν δύο exceptions που θα απορροφήσουν το σφάλμα.



```
93  loop at classes_for_calculation reference into data(clas).
94    memory_id = |{ z_class_manager=>prefix }_{ clas->class_name }|.
95    class_stamp = z_class_manager=>import_from_memory( memory_id ).
96    try.
97        data(class_name) = class_stamp[ 1 ]-obj_name.
98        data(class_package) = class_stamp[ 1 ]-devclass.
99
100       data(new_class) = new z_class( name    = conv #( class_name )
101                                      package = conv #( class_package ) ).
102       new_class->set_methods( class_stamp ).
103       clas->class_ref = new_class.
104
105       "calculate total lines per class
106       total += flow_worker->calc_total_lines_per_class( new_class ).
107     catch cx_sy_itab_line_not_found ##NO_HANDLER.
108     catch zcx_flow_issue ##NO_HANDLER.
109   endtry.
110 endloop.
```

Εικόνα 3-7: Εισαγωγή δεδομένων από την μνήμη και δημιουργία αντικειμένων

Αναφορικά με την μέθοδο “set_methods” της κλάσης “z_class”, δέχεται ως παράμετρο τον πίνακα της ***Εικόνας 3-4***, στην τοπική μεταβλητή “unstructured_methods”. Η λούπα της ***Εικόνας 3-8***, διαβάζει γραμμή προς γραμμή τον πίνακα με τις μεθόδους και δημιουργεί ένα αντικείμενο τύπου “z_method”. Επίσης το εισάγει στον πίνακα “methods” με τις μεθόδους της κλάσης “z_class”.



```
70  method set_methods.
71    loop at unstructred_methods assigning field-symbol(<meth>).
72      data(method) = new z_method( name        = conv #( <meth>-sub_name )
73                                   source_code = <meth>-source ).
74      method->set_full_name( <meth>-sobj_name ).
75      insert value #( name = <meth>-sub_name
76                      method = method ) into table methods.
77    endloop.
78  endmethod.
```

Εικόνα 3-8: Μέθοδος “set_methods” της κλάσης “z_class”

Τελικά, αφού ολοκληρωθεί η λούπα της ***Εικόνας 3-7***, θα έχουμε ένα πίνακα με αντικείμενα “z_class” που έχουν δημιουργηθεί και ετοιμαστεί για την ανάλυση του πηγαίου κώδικα.

3.3.3 Υπολογισμός μετρικών

Μετά την διαχείριση των δεδομένων περνάμε στην φάση υπολογισμού των μετρικών σε επίπεδο μεθόδου. Στις γραμμές κώδικα που ακολουθούν (***Εικόνα 3-9***) στο κύριο πρόγραμμα, υπάρχει ακόμα μία λούπα η οποία αναλαμβάνει την δημιουργία ενός

αντικειμένου “z_calc_metrics_facade” για κάθε κλάση προς ανάλυση. Το συγκεκριμένο μπλοκ κώδικα αποτελεί το πιο χρονοβόρο κομμάτι του προγράμματος, διότι θα υπολογισθούν όλες οι μετρικές για κάθε μέθοδο, κάθε κλάσης που επιθυμεί ο χρήστης.

```
117  loop at classes_for_calculation reference into clas.
118    try.
119        data(metrics_facade) = new z_calc_metrics_facade( class_stamp         = clas→class_ref
120                                                          static_object_calls = cb_cbo ).
121        metrics_facade→calculate_metrics( ).
122        output→insert_methods_to_table( clas→class_ref ).
123      catch zcx_metrics_error ##NO_HANDLER.
124      catch zcx_flow_issue ##NO_HANDLER.
125    endtry.
126  endloop.
```

Εικόνα 3-9: Λούπα για τον υπολογισμό των μετρικών

Με την δημιουργία του αντικειμένου façade, το οποίο δέχεται ως παραμέτρους την κλάση-αντικείμενο που περιέχει όλες τις απαραίτητες πληροφορίες για την ανάλυση των μετρικών καθώς και την λογική μεταβλητή “cb_cbo” η οποία αντιστοιχεί στο checkbox της αρχικής οθόνης “Calculate static calls in CBO”, αναλαμβάνει αρκετό φόρτο εργασίας για την κάθε κλάση με την εκτέλεση της μεθόδου “calculate_metrics”. Όπως φαίνεται και στην ***Εικόνα 3-10***, για την κλάση την οποία δέχεται εκτελεί μία λούπα για όλες της μεθόδους της. Δημιουργεί ένα αντικείμενο για κάθε κλάση τύπου Calculator, ούτως ώστε να υπολογισθούν οι μετρικές. Όλες οι κλάσεις εκτός της “z_authors_calculator” βασίζονται στην super κλάση “z_code_scanner”, από την οποία κληρονομούν την μέθοδο “calculate” που έχει διαφορετική συμπεριφορά ανά κλάση, αλλά πάντα επιστρέφει το αποτέλεσμα του υπολογισμού της μετρικής. Η “z_authors_calculator” χρησιμοποιεί την μέθοδο “find_authors” για τον υπολογισμό της μετρικής “Number of Authors”.

Με τον υπολογισμό της κάθε μετρικής, το αποτέλεσμα αποθηκεύεται στην αντίστοιχη μεταβλητή του αντικειμένου “meth” με την χρήση των setters. Στο τέλος της μεθόδου “calculate_metrics”, θα έχει πραγματοποιηθεί ο υπολογισμός όλων των μετρικών κάθε μεθόδου της κλάσης.

Στις παρακάτω παραγράφους περιγράφεται η χρήση της factory κλάσης “z_code_scanner_factory”, η οποία αναλύει τον πηγαίο κώδικα, ανάλογα με τις παραμέτρους και η abstract μέθοδος “calculate” της κάθε κλάσης τύπου calculator.

```
29  method calculate_metrics.
30    try.
31        loop at class_stamp->get_methods( ) reference into data(meth).
32          "LoC is being calculated outside from the facade
33
34          "calculate NoC
35          data(noc_calculator) = new z_noc_calculator( meth->method->get_source_code( ) ).
36          meth->method->set_number_of_comments( noc_calculator->calculate( ) ).
37
38          "calculate NoP
39          data(nop_calculator) = new z_nop_calculator( meth->method->get_source_code( ) ).
40          meth->method->set_number_of_pragmas( nop_calculator->calculate( ) ).
41
42          "calculate NoS
43          data(nos_calculator) = new z_nos_calculator( meth->method->get_source_code( ) ).
44          meth->method->set_number_of_statements( nos_calculator->calculate( ) ).
45
46          "calculate complex
47          data(complex_calculator) = new z_complex_calculator( meth->method->get_source_code( ) ).
48          meth->method->set_complexity_of_conditions( complex_calculator->calculate( ) ).
49
50          "calculate complex weighted by decision
51          data(decision_depth) = new z_decision_depth_calculator( meth->method->get_source_code( ) ).
52          meth->method->set_complex_decision_depth( decision_depth->calculate( ) ).
53
54          "calculate coupling between objects
55          data(coupling) = new z_cbo_calculator( source_code         = meth->method->get_source_code( )
56                                                 static_object_calls = static_object_calls ).
57          meth->method->set_coupling_between_obj( coupling->calculate( ) ).
58
59          "number of authors
60          try.
61              data(authors) = new z_authors_calculator( meth->method->get_full_name( ) ).
62              meth->method->set_number_of_authors( authors->find_authors( ) ).
63
64 *             "calculate lack of cohesion in methods
65              data(lack_of_cohesion) = new z_cohesion_calculator( class_name  = conv #( class_stamp->get_name( ) )
66                                                                  method_name = meth->method->get_name( )
67                                                                  source_code = meth->method->get_source_code( ) ).
68              meth->method->set_lack_of_cohision( lack_of_cohesion->calculate( ) ).
69            catch zcx_metrics_error into data(ex).
70              ex->display_exception( ).
71          endtry.
72        endloop.
73      catch zcx_flow_issue ##NO_HANDLER.
74    endtry.
75  endmethod.
```

Εικόνα 3-10: Μέθοδος με την αρχικοποίηση αντικειμένων και υπολογισμό των μετρικών

3.3.3.1 Ανάλυση της Factory κλάσης - z_code_scanner_factory

Με την δημιουργία ενός αντικειμένου τύπου “z_code_scanner”, ο κατασκευαστής του (***Εικόνα 3-11***) καλεί τον super κατασκευαστή.

```
16  method constructor.
17    super->constructor( scan_type   = zif_metrics=>scan_type-simple
18                        source_code = source_code ).
19  endmethod.
```

Εικόνα 3-11: Κατασκευαστής κλάσεων-παιδιών της κλάσης “z_code_scanner”

Το structure “zif_metrics=>scan_type” που χρησιμοποιείται στην παράμετρο, αποτελεί μία σταθερά με τις τιμές της παρακάτω εικόνας. Ανάλογα με το πόσο βάθος

θέλουμε να δώσουμε στην ανάλυση του πηγαίου κώδικα, χρησιμοποιούμε και την αντίστοιχη σταθερά.

```
scan_type
  none                type string value 'NONE'
  simple              type string value 'SIMPLE'
  with_comments       type string value 'WITH_COMMENTS'
  with_pragmas        type string value 'WITH_PRAGMAS'
  with_keywords       type string value 'WITH_KEYWORDS'
  with_keywords_depth type string value 'WITH_KEYWORDS_DEPTH'
```

Εικόνα 3-12: Σταθερά "scan_type" του interface "zif_metrics"

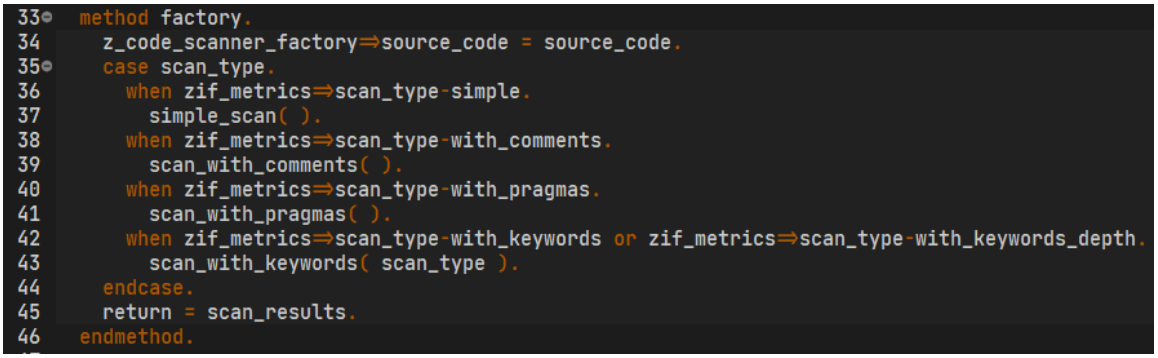
Με την χρήση της "none" δεν θα αναλυθεί καθόλου ο κώδικας, με την "simple" θα τον σκανάρει χωρίς παραμέτρους, οι "with_comments" και "with_pragmas" θα συνυπολογίσουν τις γραμμές σχολίων και pragmas, ενώ με την χρήση των "with_keywords" ή "with_keywords_depth" θα ληφθούν υπόψιν μόνο συγκεκριμένα keywords.

Ο κατασκευαστής της κλάσης "z_code_scanner" (***Εικόνα 3-13***), καλεί την στατική μέθοδο "factory" της κλάσης "z_code_scanner_factory", όπου επίσης χρησιμοποιεί την σταθερά "scan_type" και τον πηγαίο κώδικα.

```
49  method constructor.
50    me->source_code = source_code.
51    data(results) = z_code_scanner_factory=>factory( scan_type   = scan_type
52                                                     source_code = source_code ).
53    me->tokens = results-tokens.
54    me->statements = results-statements.
55  endmethod.
```

Εικόνα 3-13: Κατασκευαστής της κλάσης "z_code_scanner"

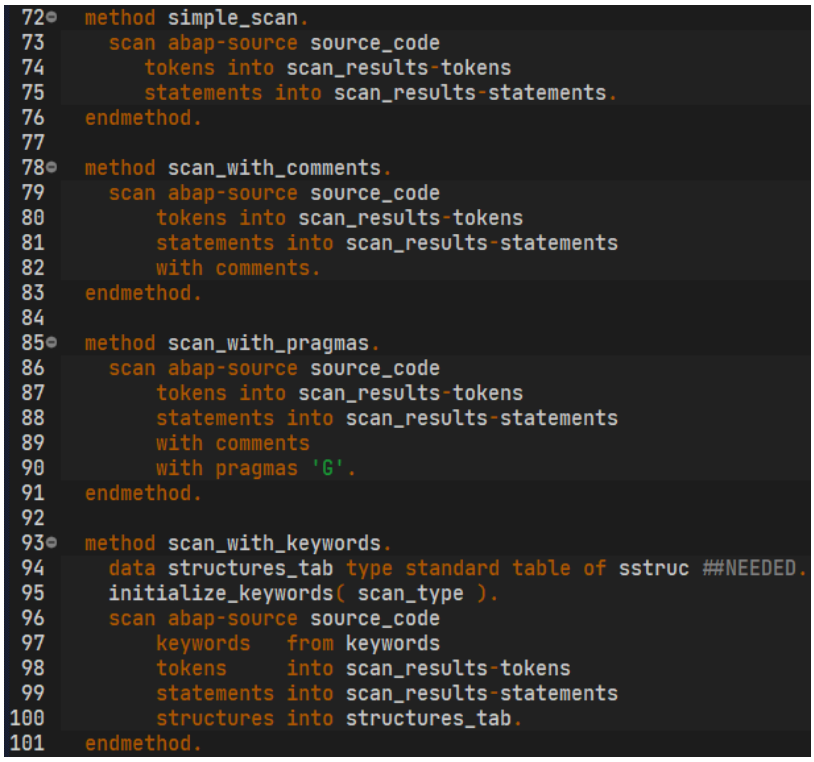
Η μέθοδος "factory" παίζει σημαντικό ρόλο στην σωστή λειτουργία του προγράμματος. Ανάλογα με την σταθερά που θα περαστεί στην μέθοδο "scan_type", θα αναλυθεί και ο πηγαίος κώδικας με βάση κάποια κριτήρια. Όπως φαίνεται και στην ***Εικόνα 3-14***, υπάρχει μία συνθήκη case η οποία δείχνει τον δρόμο προς την επιθυμητή ανάλυση.



```
33  method factory.
34    z_code_scanner_factory⇒source_code = source_code.
35    case scan_type.
36      when zif_metrics⇒scan_type-simple.
37        simple_scan( ).
38      when zif_metrics⇒scan_type-with_comments.
39        scan_with_comments( ).
40      when zif_metrics⇒scan_type-with_pragmas.
41        scan_with_pragmas( ).
42      when zif_metrics⇒scan_type-with_keywords or zif_metrics⇒scan_type-with_keywords_depth.
43        scan_with_keywords( scan_type ).
44    endcase.
45    return = scan_results.
46  endmethod.
```

Εικόνα 3-14: Μέθοδος factory της κλάσης "z_code_scanner_factory"

Οι παρακάτω μέθοδοι χρησιμοποιούνται εντός της case, αναλύουν τον πηγαίο κώδικα και αποθηκεύουν στα αποτελέσματα τα tokens και τα statements που βρέθηκαν με τη χρήση της εντολής "scan abap-source". Η εντολή αυτή, σκανάρει τον κώδικα και μπορεί να δεχτεί αρκετές παραμέτρους, όπως το συμπληρωματικό keyword "with comments" ή "with_pragmas" και αρκετά άλλα. Επιπρόσθετα, η μέθοδος factory επιστρέφει αυτά τα ευρήματα τα οποία και παίζουν καθοριστικό ρόλο στον υπολογισμό των μετρικών.



```
72  method simple_scan.
73    scan abap-source source_code
74       tokens into scan_results-tokens
75       statements into scan_results-statements.
76  endmethod.
77
78  method scan_with_comments.
79    scan abap-source source_code
80        tokens into scan_results-tokens
81        statements into scan_results-statements
82        with comments.
83  endmethod.
84
85  method scan_with_pragmas.
86    scan abap-source source_code
87        tokens into scan_results-tokens
88        statements into scan_results-statements
89        with comments
90        with pragmas 'G'.
91  endmethod.
92
93  method scan_with_keywords.
94    data structures_tab type standard table of sstruc ##NEEDED.
95    initialize_keywords( scan_type ).
96    scan abap-source source_code
97        keywords   from keywords
98        tokens     into scan_results-tokens
99        statements into scan_results-statements
100       structures into structures_tab.
101 endmethod.
```

Εικόνα 3-15: Μέθοδοι ανάλυσης κώδικα της κλάσης "z_code_scanner_factory"

3.3.3.2 Lines of Code (LoC) – z_loc_calculator

Ο υπολογισμός του συνόλου των γραμμών κώδικα δεν υπάρχει εντός της μεθόδου “calculate_metrics” της Εικόνας 3-10, διότι υπολογίζεται πιο νωρίς κατά την δημιουργία των κλάσεων-αντικειμένων, η οποία περιγράφηκε στην παράγραφο «***Διαχείριση αποτελεσμάτων Code Metric***» και πιο συγκεκριμένα, ο υπολογισμός γίνεται εντός της μεθόδου “calc_total_lines_per_class” (***Εικόνα*** 3-7, γραμμή 106). Ο λόγος που η μετρική υπολογίζεται σε αυτό το σημείο είναι διότι προτού ξεκινήσουμε την διαδικασία ανάλυσης, πρέπει να υπολογισθεί το σύνολο των γραμμών κώδικα για την προβολή του ποσοστού ολοκλήρωσης στο κάτω μέρος της οθόνης. Για τον υπολογισμό του συνόλου, χρειαζόμαστε τις γραμμές κάθε μεθόδου, οπότε για να μην επαναλαμβάνουμε την ίδια διαδικασία και εντός της μεθόδου “calculate_metrics”, ο υπολογισμός της μετρικής “Lines of Code” έλαβε μέρος εδώ.

```
202  method calc_total_lines_per_class.
203    loop at clas->get_methods( ) reference into data(meth).
204      "calculate LoC
205      data(loc_calculator) = new z_loc_calculator( meth->method->get_source_code( ) ).
206      meth->method->set_lines_of_code( loc_calculator->calculate( ) ).
207      "return the loc - the first and last line
208      return += meth->method->get_lines_of_code( ) - 2.
209    endloop.
210  endmethod.
```

Εικόνα 3-16: Μέθοδος “calc_total_lines_per_class”

3.3.3.3 Number of Comments (NoC) - z_noc_calculator

Ο υπολογισμός για την μετρική του συνόλου σχολίων (Number of Comments) ελέγχει τον πίνακα με τα tokens που έχει ήδη υπολογισθεί και ψάχνει για τον τύπο τους να είναι ίσος με ‘C’ ή ‘P’ (zif_metrics=>token_type-comment και zif_metrics=>token_type-pragma αντίστοιχα). Εφόσον ισχύει αυτή η συνθήκη, αυξάνει τον μετρητή των σχολίων κατά ένα.

```
22  method calculate.
23    loop at get_tokens( ) reference into data(token) ##NEEDED
24        where type = zif_metrics=>token_type-comment or type = zif_metrics=>token_type-pragma.
25      comments += 1.
26    endloop.
27    return = comments.
28  endmethod.
```

Εικόνα 3-17: Μέθοδος "calculate" της κλάσης "z_noc_calculator"

3.3.3.4 Number of Statements (NoS) – z_nos_calculator

Η μετρική "Number of Statements" (NoS) αφορά το σύνολο των δηλώσεων κάθε μεθόδου. Ο υπολογισμός είναι αρκετά εύκολος διότι, έχει ήδη αναλυθεί προηγουμένως στην μέθοδο "factory". Το μόνο που χρειάζεται είναι να επιστρέψουμε το σύνολο του πίνακα με τις δηλώσεις.

```
21  method calculate.
22    return = lines( get_statements( ) ).
23  endmethod.
```

Εικόνα 3-18: Μέθοδος "calculate" της κλάσης "z_nos_calculator"

3.3.3.5 Complexity of Conditions (COM) – z_complex_calculator

Ο υπολογισμός της περιπλοκότητας (COM) έχει να κάνει με την χρήση των λογικών τελεστών 'AND' και 'OR' σε συνθήκες ελέγχου. Με την ανάλυση του κώδικα, έχουν ήδη υπολογισθεί τα tokens, οπότε το μόνο που θα χρειαστεί να συμβεί είναι να ελεγχθεί ο πίνακας για το εάν περιέχει 'OR' ή 'AND' (zif_metrics=>tokens-or και str = zif_metrics=>tokens-and αντίστοιχα). Κάθε φορά που εντοπίζεται ένα από τα δύο, αυξάνεται η μετρική κατά ένα.

```
22  method calculate.
23    loop at get_tokens( ) reference into data(token) ##NEEDED
24        where str = zif_metrics=>tokens-or or str = zif_metrics=>tokens-and.
25      com_sum += 1.
26    endloop.
27    return = com_sum.
28  endmethod.
```

Εικόνα 3-19: Μέθοδος "calculate" της κλάσης "z_complex_calculator"

3.3.3.6 Complexity Wighted by Decision (Depth) – z_decision_depth_calculator

Η πολυπλοκότητα σε βάθος, διαβάζει με την σειρά όλα τα διαθέσιμα tokens της μεθόδου. Εάν αυτό που θα διαβάσει αντιπροσωπευει ένα token για έναρξη συνθήκης, με εξαίρεση το 'TRY', τότε αυξάνει το βάθος κατά ένα και το προσθέτει στο συνολικό βάθος. Εάν το token πρόκειται για ένα keyword που τερματίζει την συνθήκη, εκτός του 'ENDTRY', τότε μειώνει το βάθος κατά ένα. Όσο μεγαλύτερος είναι ο αριθμός του συνολικού βάθους, τόσο πιο πολλές εμφωλευμένες συνθήκες έχουμε.

```
23  method calculate.
24    data(keywords) = new z_keywords( ).
25    loop at get_tokens( ) reference into data(token).
26      if keywords->is_open_keyword( token->str ) and token->str <> 'TRY'.
27        deep += 1.
28        deep_total += deep.
29      endif.
30      if keywords->is_close_keyword( token->str ) and token->str <> 'ENDTRY'.
31        deep -= 1.
32      endif.
33    endloop.
34    return = deep_total.
35  endmethod.
```

Εικόνα 3-20: Μέθοδος "calculate" της κλάσης "z_decision_depth_calculator"

3.3.3.7 Coupling Between Objects (CBO) – z_cbo_calculator

Η σύζευξη μεταξύ αντικειμένων είναι μία από τις πιο βασικές μετρικές στα έργα λογισμικού. Ο υπολογισμός της είναι αρκετά απλός, μετρώντας τις κλήσεις άλλων μεθόδων εντός της μεθόδου που αναλύεται. Έτσι, ελέγχοντας ένα προς ένα τα tokens, εάν περιέχουν τα σύμβολα κλήσης μεθόδων '→' και '=>' (zif_metrics=>method_call-instance και zif_metrics=>method_call-static αντίστοιχα). Φυσικά, για τον συνυπολογισμό των στατικών κλήσεων παίζει ρόλο και η λογική μεταβλήτη "static_object_calls" που λαμβάνεται από την αρχική οθόνη.

```
25  method calculate.
26    loop at get_tokens( ) reference into data(token).
27      if token->str cs zif_metrics=>method_call-instance
28        or ( static_object_calls = abap_true and token->str cs zif_metrics=>method_call-static ).
29        coupling += 1.
30      endif.
31    endloop.
32    return = coupling.
33  endmethod.
```

Εικόνα 3-21: Μέθοδος "calculate" της κλάσης "z_cbo_calculator"

3.3.3.8 Number of Authors (Authors) – z_authors_calculator

Οπώς αναφέρθηκε πιο πάνω, η κλάση για την εύρεση των συγγραφέων δεν χρησιμοποιεί την μέθοδο “calculate” αλλά την “find_authors”. Για τον υπολογισμό των ατόμων που έχουν εμπλακεί στην προσθήκη/υλοποίηση της μεθόδου που αναλύεται, χρησιμοποιείται ένα standard function module του SAP (“SVRS_GET_VERSION_DIRECTORY_46”), το οποίο επιστρέφει μία λίστα με τους χρήστες που έχουν επεξεργαστεί τον κώδικα. Με βάση αυτή τη λίστα, υπολογίζεται το σύνολο των χρηστών.

```
28  method find_authors.
29    get_versions( ).
30    data last_author type string.
31    loop at version_list reference into data(version).
32      if sy-tabix = 1.
33        last_author = version→author.
34        authors += 1.
35        continue.
36      endif.
37
38      if last_author = version→author.
39        continue.
40      endif.
41
42      last_author = version→author.
43      authors += 1.
44    endloop.
45    return = authors.
46  endmethod.
```

Εικόνα 3-22: Μέθοδος “calculate” της κλάσης “z_authors_calculator”

3.3.3.9 Lack of Cohesion in Methods (LCOM2) – z_cohesion_calculator

Ο αλγόριθμος για τον υπολογισμό της έλλειψης συνοχής σε επίπεδο μεθόδου αποτελεί το πιο απαιτητικό και χρονοβόρο κομμάτι του προγράμματος και χωρίζεται σε τρία βήματα.

Το πρώτο βήμα είναι ο καθαρισμός του πηγαίου κώδικα από κενές γραμμές και σχόλια. Το δεύτερο είναι ο έλεγχος του κώδικα για δηλώσεις τοπικών μεταβλητών, σταθερών, structures και πινάκων. Αποθηκεύονται τοπικά για μελλοντική χρήση. Στο τρίτο και τελευταίο βήμα, γίνεται η ανάλυση του πηγαίου κώδικα γραμμή προς γραμμή. Κάθε γραμμή της μεθόδου συγκρίνεται με όλες τις υπόλοιπες γραμμές. Για παράδειγμα

για μία μέθοδο με 5 γραμμές, οι έλεγχοι που θα πραγματοποιηθούν είναι: γραμμή 1 – γραμμή 2, γραμμή 1 – γραμμή 3, .. ,γραμμή 1 - γραμμή 5, γραμμή 2 – γραμμή 3 κ.ο.κ..

```
128  method calculate.
129    clean_source_code( abap_true ).
130    set_local_vars_n_tokens_ids( ).
131    analyze_source_code( ).
132
133    return = cond #( when lack_of_cohesion-not_cohesive - lack_of_cohesion-cohesive < 0
134                     then 0 else lack_of_cohesion-not_cohesive - lack_of_cohesion-cohesive ).
135  endmethod.
```

Εικόνα 3-23: Μέθοδος "calculate" της κλάσης "z_cohesion_calculator"

Το πιο περίπλοκο τμήμα κώδικα και αυτό που πρέπει να αναλυθεί είναι το τρίτο, που βρίσκεται στην μέθοδο "analyze_source_code". Η διαδικασία που ακολουθείται είναι η εξής - εφαρμόζεται μία λούπα στον πίνακα του πηγαίου κώδικα και εντός αυτής χρειάζεται και άλλη μία μορφή επανάληψης, η οποία θα ελέγχει από την τωρινή γραμμή έως και την τελευταία γραμμή της μεθόδου. Η επανάληψη αυτή πραγματοποιείται με την εντολή while. Για παράδειγμα, άμα η μέθοδος μας έχει εφτά γραμμές κώδικα και η πρώτη λούπα ελέγχει την τρίτη γραμμή, τότε η while θα βοηθήσει στην πρόσβαση περιεχομένου από την γραμμή τέσσερα έως την τελευταία. Συνεπώς, δημιουργείται ένα ζευγάρι γραμμών, στο οποίο οι έλεγχοι που πραγματοποιούνται είναι, το εάν περιέχουν κάποια κοινή μεταβλητή, structure, πίνακα ή σταθερά, εάν η πρώτη γραμμή περιέχει keyword για ξεκίνημα εντολής και η επόμενη γραμμή περιέχει το αντίστοιχο keyword για την ολοκλήρωση της εντολής και τέλος εάν η παρένθεση από το κάλεσμα μίας μεθόδου κλείνει σε επόμενη γραμμή.

Σε οποιαδήποτε από τις παραπάνω περιπτώσεις το αποτέλεσμα είναι θετικό, τότε, το ζευγάρι γραμμών θεωρείται συνεκτικό, σε κάθε άλλη περίπτωση μη συνεκτικό. Με το πέρας της ανάλυσης, έχει ολοκληρωθεί η μέτρηση των συνεκτικών και μη, και προχωράμε στον υπολογισμό της μετρικής, η οποία και βγαίνει από τον τύπο:

$$LCOM2 = P - Q, \text{ εάν } P - Q \geq 0 \text{ / αλλιώς } LCOM2 = 0$$

όπου P τα μη συνεκτικά ζευγάρια γραμμών και Q τα συνεκτικά ζευγάρια γραμμών (***Εικόνα 3-23***).

3.3.4 Προβολή αποτελεσμάτων

Αφού ολοκληρωθεί η διαδικασία υπολογισμού, σειρά έχει η προβολή των αποτελεσμάτων. Οι μέθοδοι με τις υπολογισμένες μετρικές, μεταφέρονται σε έναν πίνακα με τις παρακάτω στήλες:

- Πακέτο
- Όνομα κλάσης
- Όνομα μεθόδου
- Γραμμές κώδικα (LoC)
- Αριθμός σχολίων (NoC)
- Αριθμός statements (NoS)
- Περιπλοκότητα (COM)
- Αριθμός συγγραφέων (Auths)
- Σύζευξη αντικειμένων (CBO)
- Έλλειψη συνοχής σε επίπεδο μεθόδου (LCOM2)

Τα πεδία πακέτο και κλάση ομαδοποιούνται πάντα, ενώ με βάση την επιλογή του χρήστη μπορεί να προβληθεί το άθροισμα των μετρικών ανά κλάση ή ο μέσος όρος. Όπως αναφέρθηκε παραπάνω, όλοι οι υπολογισμοί γίνονται σε επίπεδο μεθόδου, υπάρχει όμως και η επιλογή για προβολή των αποτελεσμάτων σε επίπεδο κλάσης. Στην ***Εικόνα 3-24***, φαίνεται η ολοκλήρωση του υπολογισμού μετρικών για μία κλάση.

Package	Class	Method	LoC	NoC	NoS	COM	Auths	Depth	CBO	LCOM2
Z001	ZCL_UTILITY	CONV_DATE_TO_SU01_FORMAT	3	0	3	0	1	0	0	0
		CONVERT_EXCEL_DATE_TO_YYYYMMDD	15	0	15	0	1	5	0	34
		CHAR_TO_TABLE	8	0	3	0	1	0	0	15
		COUNT_CHAR_LEADING_ZEROS	12	0	12	3	1	3	1	13
		MODIFY_MIDDLE_EAST_QUANTITIES	16	33	13	0	2	3	0	63
		REMOVE_DECIMAL_FROM_INTEGER	18	0	14	3	1	4	0	86
		SU01_DECIMAL_NOTATION	42	0	40	0	2	9	0	530
		TABLE_TO_STRING	24	1	25	0	2	9	1	163
		GET_GLOBAL_SET	31	0	11	0	3	3	0	390
		UPLOAD_CSV_TO_ITAB	108	9	81	13	1	28	24	4.929
		STRING_TO_NUMERIC	77	14	60	0	1	34	0	2.287
		GET_TVARVC_SELECT_OPTION	17	0	8	0	1	1	0	91
		GET_TVARVC_PARAMETER	9	0	4	0	2	0	0	19
		BUILD_RANGE	37	3	26	2	1	8	0	507
		UPLOAD_EXCEL_TO_ITAB	122	2	85	14	2	66	30	6.716
		DELETE_FILE_FROM_DIR	3	0	3	0	1	0	0	0
		COPY_FILE_TO_DIR	30	4	30	0	2	21	0	314
		MOVE_FILE_TO_DIR	5	0	4	0	1	0	2	0
	ZCL_UTILI...		577	66	437	35	3	194	58	16.157
Z001			577	66	437	35	3	194	58	16.157
			577	66	437	35	3	194	58	16.157

Εικόνα 3-24: Αποτελέσματα του προγράμματος

3.4 Ευρήματα

Για την αξιοποίηση των ευρημάτων επιλέχθηκαν 102 μέθοδοι από τρία έργα. Δημιουργήθηκε μία φόρμα Google με το περιεχόμενό τους και μία κλίμακα από το ένα έως το δέκα για την αξιολόγηση της αντίστοιχης μεθόδου ως συντηρήσιμη ή μη. Ρωτήθηκαν στο σύνολο 8 συνάδελφοι προγραμματιστές, που έχουν ασχοληθεί με τα συγκεκριμένα έργα.

Από τα αποτελέσματα του ερωτηματολογίου βγήκε ο μέσος όρος και σε συνδυασμό με τα αποτελέσματα που λήφθηκαν από την εκτέλεση του προγράμματος, προκύπτουν μερικά χρήσιμα και ενδιαφέροντα ευρήματα. Ο όρος “rate” που συναντάμε πιο κάτω αναφέρεται στον μέσο όρο των απαντήσεων των χρηστών.

3.4.1 Περιγραφικά στατιστικά

Ο παρακάτω πίνακας περιέχει στατιστικά από τις μετρικές που υπολογίστηκαν για 102 μεθόδους. Περιέχει τις ελάχιστες και μέγιστες τιμές που παίρνουν οι μετρικές, τον μέσο όρο (mean) και την απόκλιση. Η μικρή απόκλιση σημαίνει ότι τα αποτελέσματα βρίσκονται κοντά στον μέσο όρο.

Πίνακας 3-1: Περιγραφικά στατιστικά

	N	Minimum	Maximum	Mean	Std. Deviation
Rate	102	1,000	7,875	3,31985	1,606432
LoC	102	3	128	22,00	22,482
NoC	102	0	31	4,18	6,781
NoS	102	3	101	15,35	17,305
COM	102	0	14	0,62	2,106
Authors	102	1	4	1,41	0,813
Depth	102	0	66	4,31	9,616
CBO	102	0	30	1,70	4,985
LCOM2	97	0	7163	292,54	1012,614
Valid N (listwise)	97				

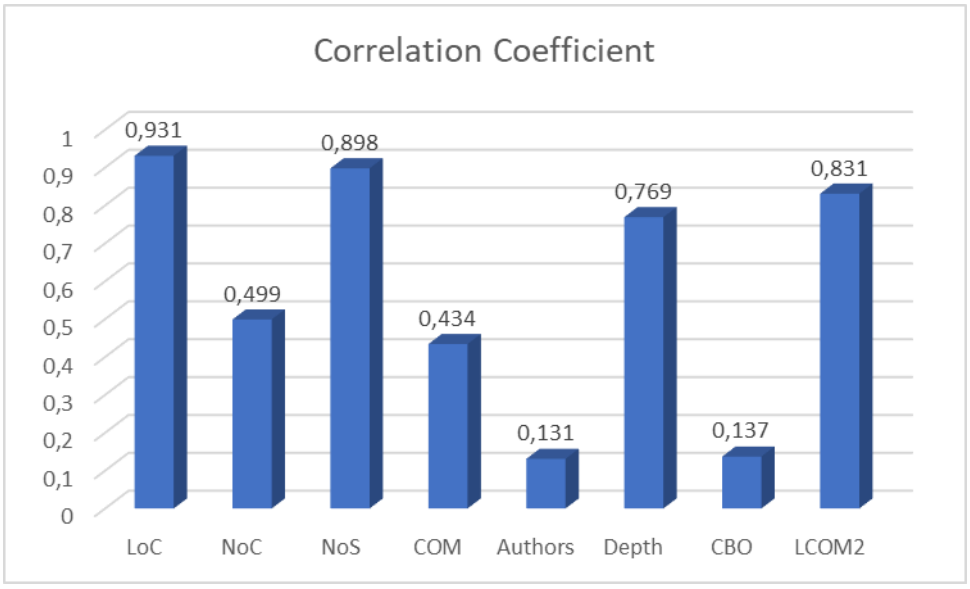
Στον ***Πίνακα 3-2*** παρουσιάζονται τα αποτελέσματα "Ανάλυση συσχέτισης Pearson", όπου δείχνουν τη συσχέτιση μεταξύ των μετρικών. Το εύρος τιμών του συντελεστή συσχέτισης (correlation coefficient) παίρνει τιμές από -1 έως 1 και αντικατοπτρίζει τη σύνδεση των μετρικών. Μία θετική τιμή σημαίνει ότι όταν, μία μετρική αυξάνεται τότε και η άλλη αυξάνεται, ενώ μία αρνητική σημαίνει ότι όταν μία αυξάνεται η άλλη μειώνεται.

Αξίζει να αναφερθεί ότι, οι υψηλότερες τιμές εμφανίζονται στις μετρικές LoC, NoS, Depth και LCOM2. Όσο αυξάνεται ο βαθμός συντηρησιμότητας (Rate) θα αυξάνονται οι παραπάνω μετρικές και αντιστρόφως. Δηλαδή, όσο μία μέθοδος μεγεθύνεται (αύξηση γραμμών - LoC) ή πολλαπλασιάζονται οι δηλώσεις (αύξηση NoS) ή μεγαλώνει το βάθος στις συνθήκες (Depth) και η έλλειψη συνοχής (LCOM2), θα αυξάνεται και ο συντελεστής "Rate". Αντιθέτως, οι μετρικές NoC, και COM έδειξαν χαμηλότερες τιμές. Βάσει αυτό, καταλαβαίνουμε ότι η αύξηση των τιμών, μεγαλώνει σε μικρότερο βαθμό όσο ανεβαίνει η τιμή "Rate". Οι χαμηλότερες τιμές προήλθαν από τις μετρικές Authors και CBO. Αποτελούν τις μοναδικές μετρικές που δεν είναι σημαντικά στατιστικά συσχετιζόμενες, διότι η τιμή της μεταβλητής "Sig. (2-tailed)" (αντιπροσωπεύει την μεταβλητή p), είναι μεγαλύτερη από 0.005.

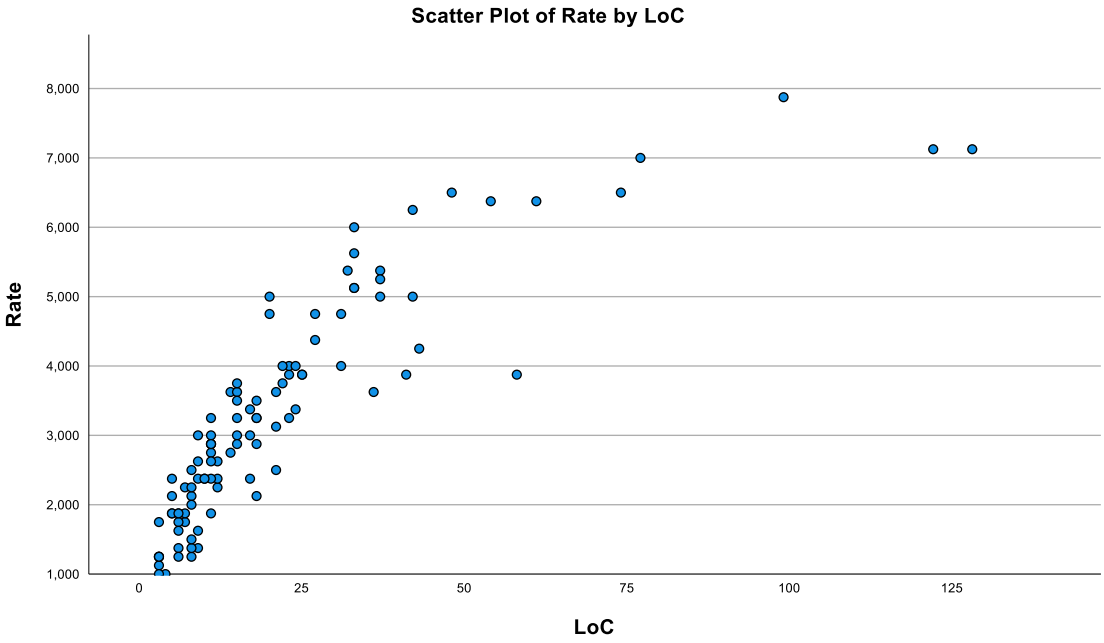
Πίνακας 3-2: Μη παραμετρικές συσχετίσεις - Συσχετισμοί

		LoC	NoC	NoS	COM	Authors	Depth	CBO	LCOM2
Rate	Correlation Coefficient	0,931	0,499	0,898	0,434	0,131	0,769	0,137	0,831
	Sig. (2-tailed)	<0,001	<0,001	<0,001	<0,001	0,191	<0,001	0,168	<0,001
	N	102	102	102	102	102	102	102	97

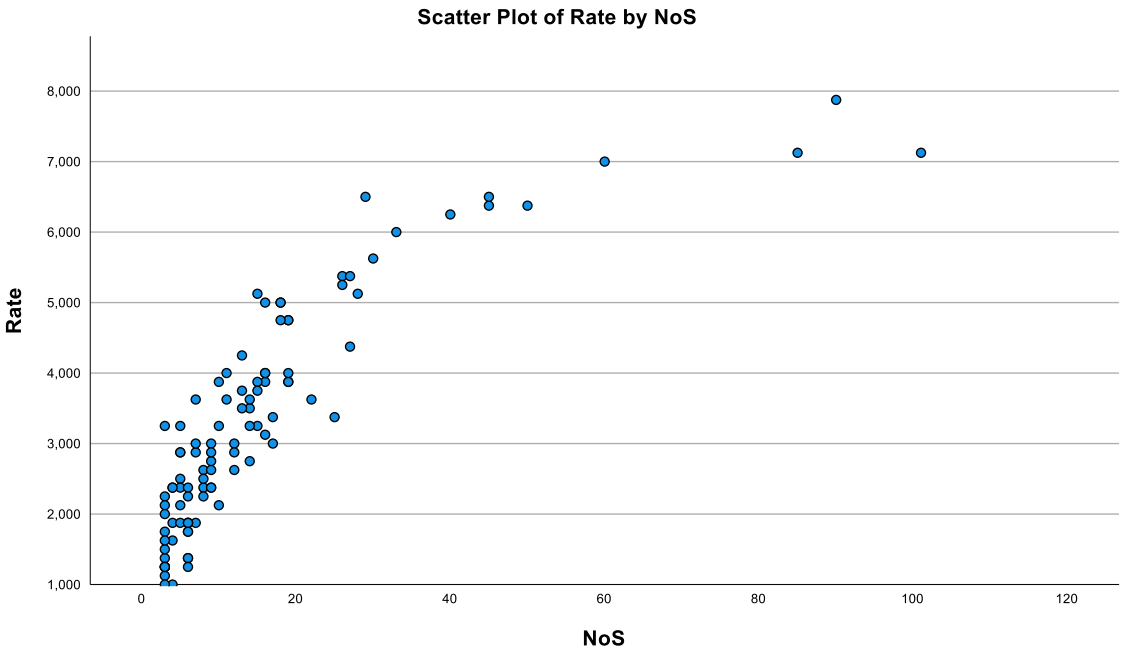
Στις εικόνες που ακολουθούν, παρατηρούνται μερικά γραφήματα που προκύπτουν από τους παραπάνω πίνακες.



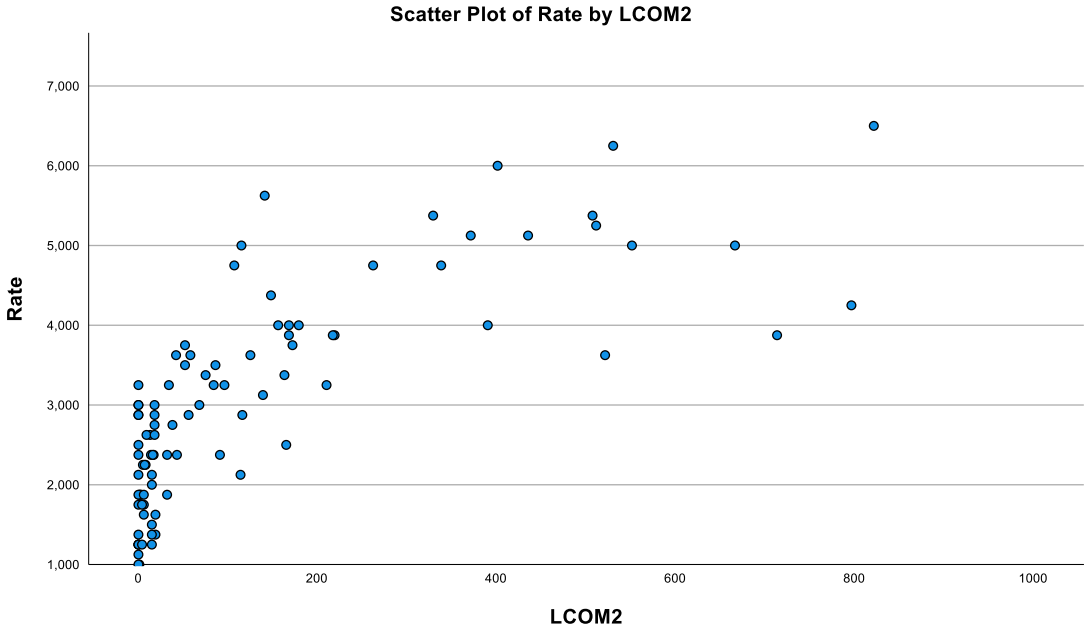
Εικόνα 3-25: Συντελεστής συσχέτισης ανά μετρική



Εικόνα 3-26: Αξιολόγηση προς γραμμές κώδικα



Εικόνα 3-27: Αξιολόγηση προς Αριθμό Δηλώσεων



Εικόνα 3-28: Αξιολόγηση προς Έλλειψη συνοχής

3.4.2 Συχνότητα

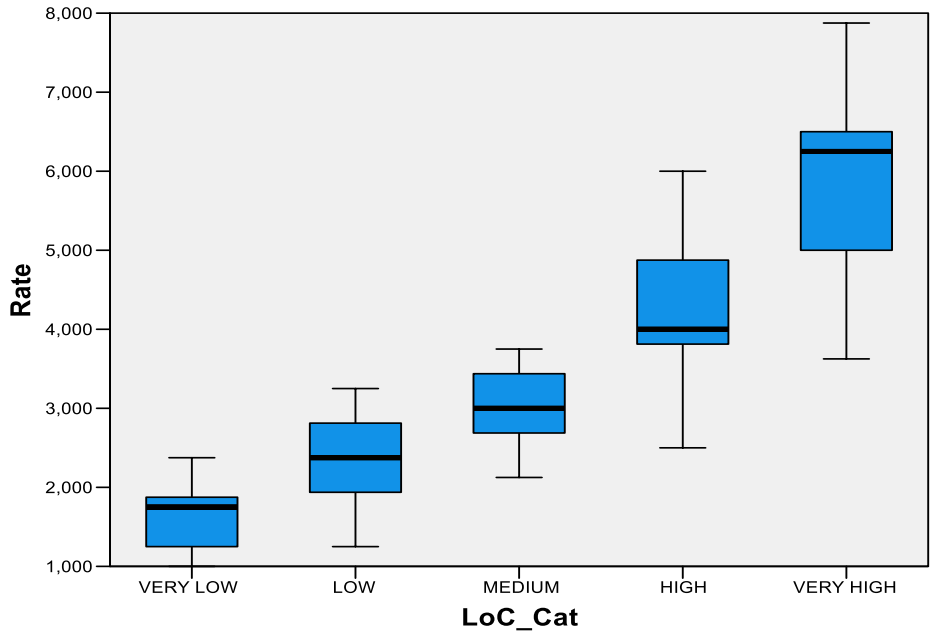
Στον Πίνακα 3-3, το σύνολο των αποτελεσμάτων των μεθόδων διαιρούνται με το 100, ούτως ώστε να μπορέσουμε να τα μοιράσουμε σε τμήματα, όπως, μικρότερα του 20%, 20-40%, 40-60%, 60-80% και μεγαλύτερα του 80%. Οι τιμές που μπορούν να πάρουν φαίνονται παρακάτω. Τα τμήματα αυτά ονομάζονται “Percentiles”. Για παράδειγμα, εάν μία μέθοδος έχει αξιολογηθεί με 3,1 τότε θα ανήκει στο δεύτερο τμήμα.

Πίνακας 3-3: Συχνότητα, Percentiles

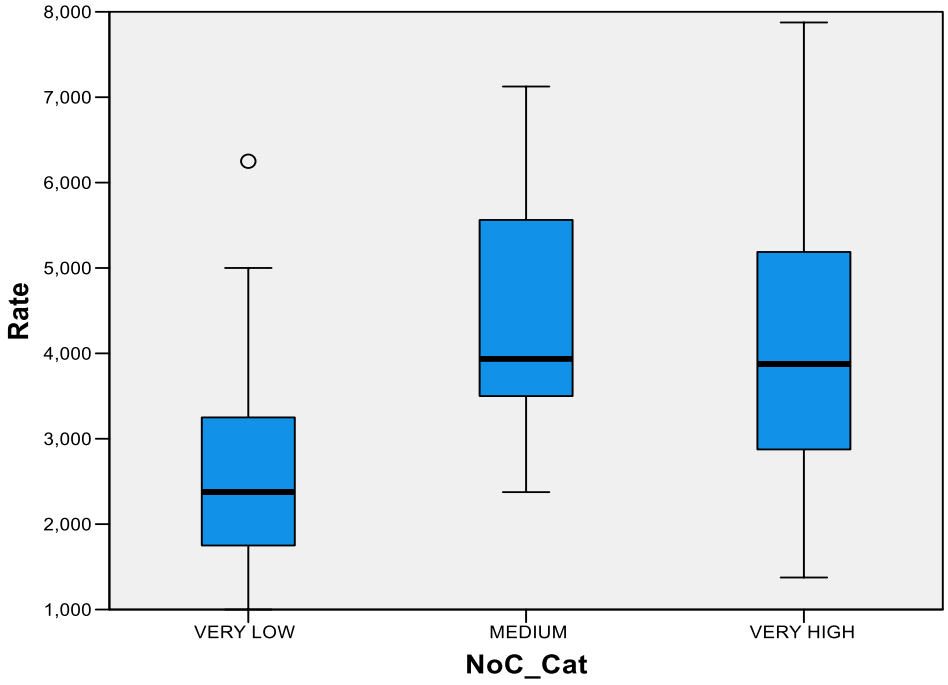
		Rate	LoC	NoC	NoS	COM	Authors	Depth	CBO	LCOM2
N	Valid	102	102	102	102	102	102	102	102	97
	Missing	0	0	0	0	0	0	0	0	5
Percentiles	20	1,875	7,60	0,00	5,00	0,00	1,00	0,00	0,00	2,60
	40	2,625	11,00	0,00	8,00	0,00	1,00	1,00	0,00	18,00
	60	3,475	18,00	2,00	13,80	0,00	1,00	2,00	0,80	90,00
	80	4,750	33,00	9,00	19,00	0,00	2,00	4,40	2,00	236,20

Τα θηκογράμματα που ακολουθούν παρουσιάζουν μία συνεχή άυξηση, όσο αυξάνεται η αξιολόγηση τόσο αυξάνονται και οι μετρικές. Εξαίρεση αποτελεί το θηκογράφημα για το πλήθος σχολίων (***Εικόνα 3-30***), στο οποίο φαίνεται ότι η μεγαλύτερη κατηγορία **very-high**, έχει περίπου ίδιο μέσο όρο με την μεσαία κατηγορία **medium**. Δηλαδή όσο αυξάνονται τα σχόλια, δεν αυξάνεται η αξιολόγηση των μεθόδων.

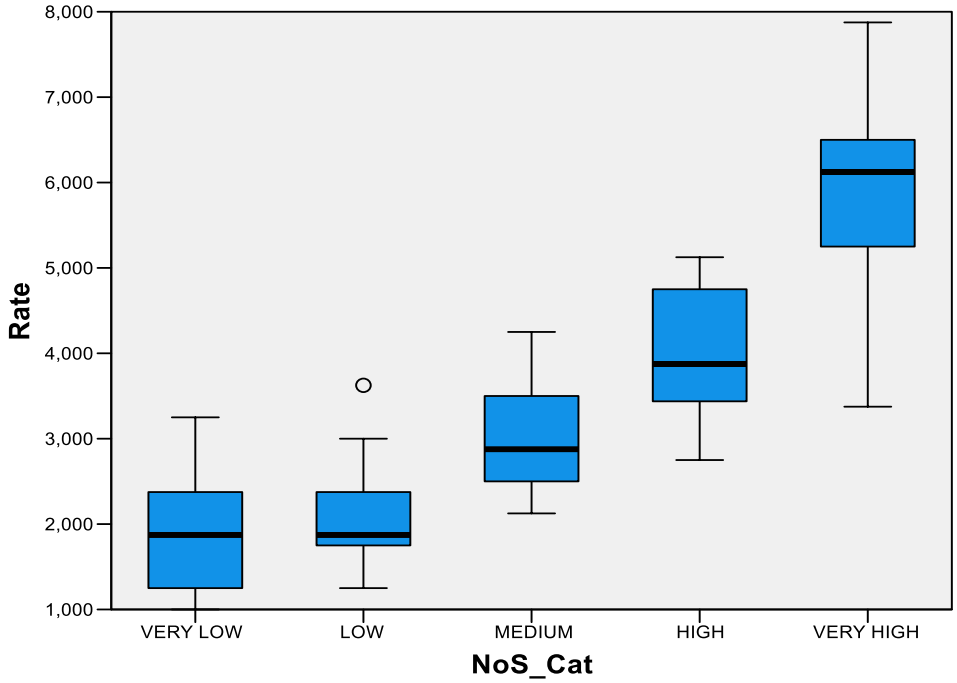
Ενδιαφέρον έχει η αντίθεση που παρατηρείται από τον συνδιασμό των αποτελεσμάτων στα γραφήματα της ***Εικόνας 3-29*** και της ***Εικόνας 3-30***, όπου στα χαμηλά τμήματα (**very low** και **low**) του πρώτου η αξιολόγηση είναι χαμηλή, όπως επίσης και στο χαμηλότερο του δεύτερου (**very low**). Δηλαδή, σε κώδικα με λίγες γραμμές, θα έχουμε λίγα σχόλια έως καθόλου, και στο σύνολό της η μέθοδος θα θεωρείται εύκολα συντηρίσιμη. Σε αντίθεση, στο τμήμα **very high** των δύο θηκογραμμάτων, όσο αυξάνονται οι γραμμές τόσο αυξάνεται και η αξιολογηση (***Εικόνα 3-29***), πράγμα που δεν συμβαίνει με τα σχόλια (***Εικόνα 3-30***).



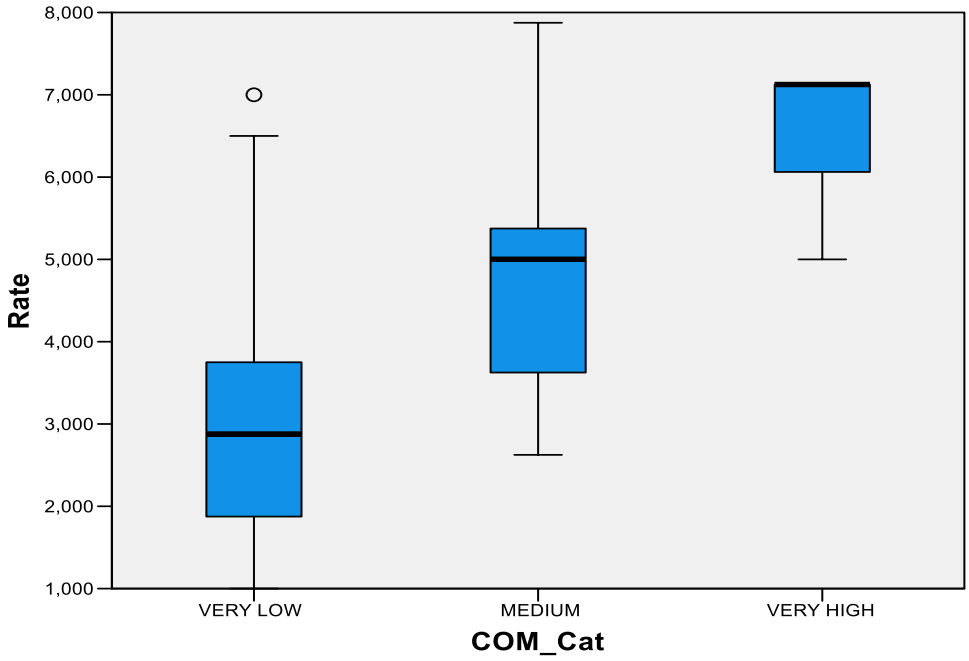
Εικόνα 3-29: Θηκόγραμμα (Box-plot) Αξιολόγηση προς Γραμμές κώδικα



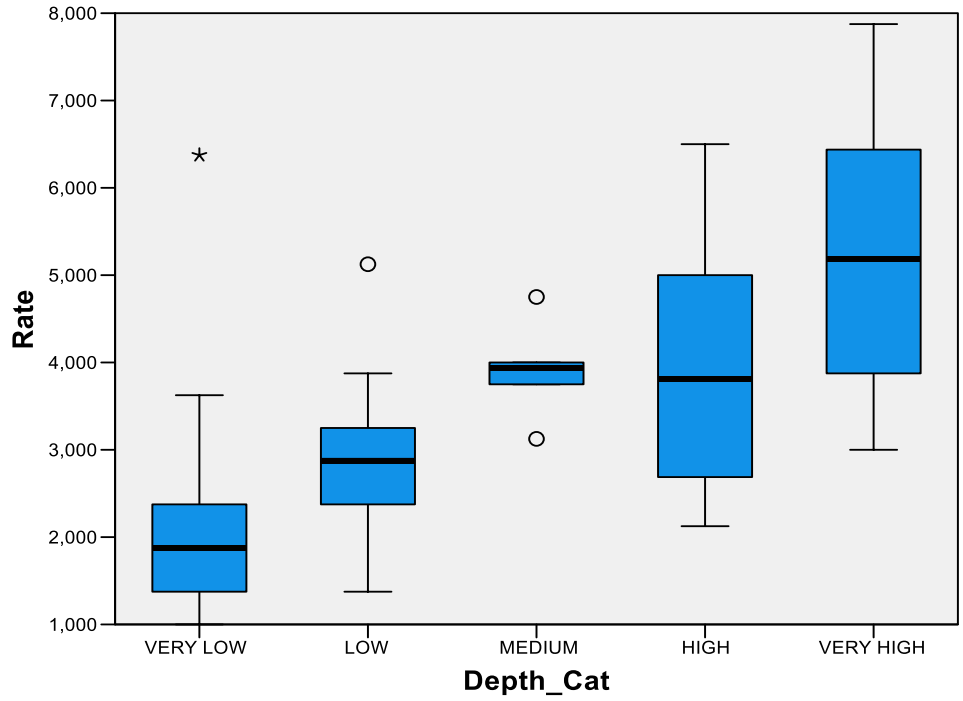
Εικόνα 3-30: Θηκόγραμμα (Box-plot) Αξιολόγηση προς Αριθμό σχολίων



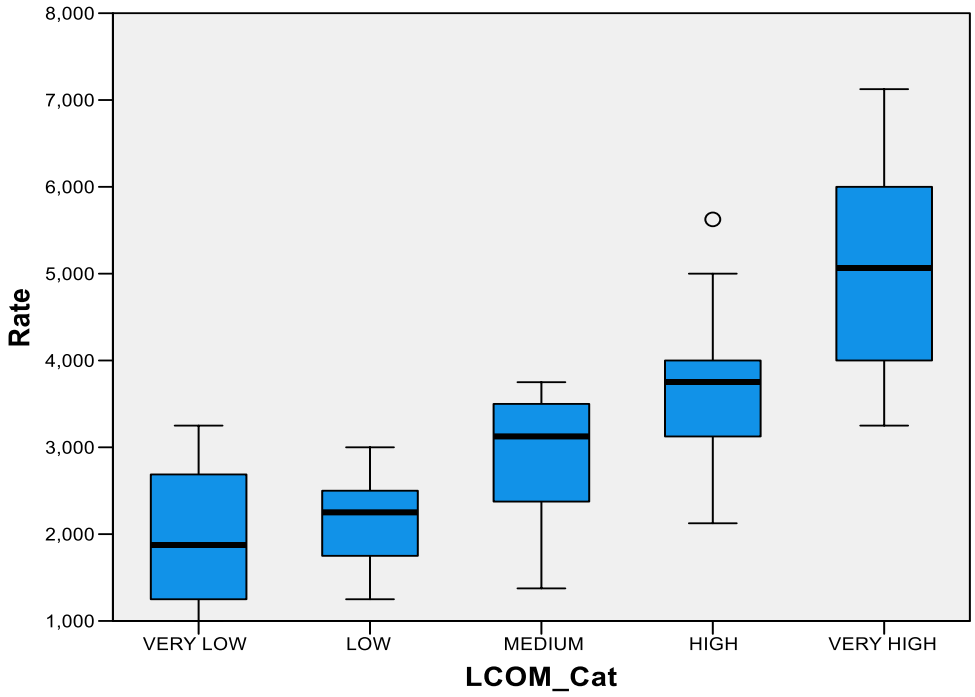
Εικόνα 3-31: Θηκόγραμμα(Box-plot) Αξιολόγηση προς Αριθμό Δηλώσεων



Εικόνα 3-32: Θηκόγραμμα(Box-plot) Αξιολόγηση προς Περιπλοκότητα



Εικόνα 3-33: Θηκόγραμμα(Box-plot) Αξιολόγηση προς Βάθος Πολυπλοκότητας



Εικόνα 3-34: Θηκόγραμμα(Box-plot) Αξιολόγηση προς Έλλειψη Συνοχής

3.4.3 Παλινδρόμηση

Οι παρακάτω δύο πίνακες, είναι τα αποτελέσματα από την ανάλυση παλινδρόμησης. Δημιουργήθηκαν τέσσερα μοντέλα. Το πρώτο εκτελέστηκε με την χρήση όλων των διαθέσιμων μετρικών (στήλη "Variables Entered") και στα επόμενα αφαιρούνταν μία μία (στήλη "Variables Removed"). Η ανάλυση σταμάτησε στο τέταρτο μοντέλο, το οποίο μας δίχνει ποσοστό επιτυχίας πρόβλεψης 91,1% (0,911) και φαίνεται στην στήλη "R Sqaure" του ***Πίνακα 3-5***.

Πίνακας 3-4: Ανάλυση Παλινδρόμησης, δημιουργία μοντέλων

Model	Variables Entered	Variables Removed	Method
1	LCOM2, Authors, NoC, CBO, Depth, COM, LoC, NoS	-	Enter
2	-	CBO	Backward (criterion: Probability of F-to-remove >= 0,100).
3	-	Depth	Backward (criterion: Probability of F-to-remove >= 0,100).
4	-	NoC	Backward (criterion: Probability of F-to-remove >= 0,100).

Πίνακας 3-5: Ανάλυση Παλινδρόμησης, ποσοστά

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	0,957	0,916	0,908	0,446163
2	0,957	0,915	0,908	0,446463
3	0,956	0,913	0,907	0,448853
4	0,954	0,911	0,906	0,452248

Ο ***Πίνακας 3-6*** περιέχει τιμές που μπορούμε να χρησιμοποιήσουμε για την πρόβλεψη της αξιολόγησης μίας μεθόδου. Πιο συγκεκριμένα, λαμβάνοντας υπόψιν μόνο το τέταρτο μοντελο και θέτωντας σε χρήση τον παρακάτω τύπο μπορούμε να προβλέψουμε την τιμή "rate" με ποσοστό επιτυχίας 91,1%.

Rate = Constant + (**0,091** * LoC) + (**0,059** * NoS) + (**0,122** * COM) + (**0,168** * Authors) + (**-0,002** * LCOM2),

όπου Constant η τιμή **0,794** του πίνακα και όπου οι μετρικές, τα αποτελέσματα που εξήχθηκαν με την εκτέλεση του προγράμματος της κάθε μεθόδου.

Πίνακας 3-6: Ανάλυση Παλινδρόμησης, τιμές πρόβλεψης

		Unstandardized Coefficients		Standardized Coefficients		
Model		B	Beta		t	Sig
1	(Constant)	0,821	0,121		6,784	<0,001
	LoC	0,093	0,009	1,255	10,858	<0,001
	NoC	0,017	0,008	0,072	1,973	0,052
	NoS	0,046	0,012	0,473	3,861	<0,001
	COM	0,121	0,046	0,175	2,653	0,009
	Authors	0,152	0,059	0,084	2,584	0,011
	Depth	0,02	0,013	0,128	1,542	0,127
	CBO	0,015	0,014	0,05	1,058	0,293
	LCOM2	-0,002	0	-1,307	-11,903	<0,001
2	(Constant)	0,829	0,121		6,865	<0,001
	LoC	0,092	0,009	1,241	10,801	<0,001
	NoC	0,016	0,008	0,069	1,912	0,059
	NoS	0,048	0,012	0,493	4,072	<0,001
	COM	0,128	0,045	0,185	2,833	0,006
	Authors	0,152	0,059	0,084	2,575	0,012
	Depth	0,018	0,013	0,115	1,402	0,164
	LCOM2	-0,002	0	-1,271	-12,149	<0,001
3	(Constant)	0,812	0,121		6,719	<0,001
	LoC	0,09	0,008	1,22	10,653	<0,001
	NoC	0,012	0,008	0,053	1,543	0,126
	NoS	0,057	0,01	0,585	5,738	<0,001
	COM	0,119	0,045	0,172	2,639	0,01
	Authors	0,147	0,059	0,081	2,482	0,015
	LCOM2	-0,002	0	-1,217	-12,443	<0,001
4	(Constant)	0,794	0,121		6,553	<0,001
	LoC	0,091	0,009	1,224	10,614	<0,001

	NoS	0,059	0,01	0,609	5,991	<0,001
	COM	0,122	0,045	0,176	2,68	0,009
	Authors	0,168	0,058	0,093	2,909	0,005
	LCOM2	-0,002	0	-1,228	-12,488	<0,001

4 Επίλογος

Εν κατακλείδι, η συνεχής βελτιστοποίηση και επέκταση των λογισμικών οδηγεί σε ενδεχόμενα προβλήματα, τα οποία με την σειρά τους στην αύξηση του τεχνικού χρέους. Το τεχνικό χρέος αναφέρεται στο σύνολο τεχνικών προκλήσεων και ζητημάτων σε ένα σύστημα λογισμικού που παραμένουν ανεπίλυτα. Αυτά τα ζητήματα μπορεί να προκύψουν από γρήγορες αποφάσεις που λαμβάνονται κατά τη διαδικασία της ανάπτυξης λογισμικού, τη χρήση ξεπερασμένης τεχνολογίας ή την ελλιπή εφαρμογή ελέγχων όσον αφορά την ποιότητα του κώδικα. Το τεχνικό χρέος μπορεί να οδηγήσει σε αναποτελεσματικότητα και δυσκολίες κατά την μελλοντική ανάπτυξη, συντήρηση και αναβάθμιση. Μπορεί επίσης να αυξήσει το κόστος των μελλοντικών εργασιών ανάπτυξης, καθώς η επίλυση αυτών των προκλήσεων και ζητημάτων θα απαιτήσει περισσότερο χρόνο και πόρους.

Η διαχείριση τεχνικού χρέους αποτελεί βασικό μέρος της ανάπτυξης και συντήρησης λογισμικού και περιλαμβάνει τη λήψη αποφάσεων σχετικά με το πότε θα δοθεί προτεραιότητα στη διόρθωση τεχνικών ζητημάτων και πότε θα δοθεί προτεραιότητα στις νέες απαιτήσεις. Ο στόχος είναι να εξισορροπηθούν τα βραχυπρόθεσμα οφέλη των νέων απαιτήσεων με το μακροπρόθεσμο κόστος των ανεπίλυτων τεχνικών ζητημάτων.

Ο υπολογισμός των μετρικών λογισμικού μπορεί να βοηθήσει τις εταιρίες να μειώσουν το τεχνικό τους χρέος. Η ανάπτυξη καθαρού κώδικα, η αποφυγή επαναλαμβανόμενου κώδικα και η χρήση σχολίων μπορούν να παίξουν σημαντικό ρόλο στην βελτίωση ποιότητας κώδικα, με συνέπεια την μείωση του τεχνικού χρέους.

4.1 Σύνοψη και συμπεράσματα

Μετά την έρευνα που πραγματοποιήθηκε για την ανάλυση και αξιοποίηση μεθόδων όσον αφορά την ποιότητα κώδικα, σε τρία διαφορετικά έργα λογισμικού, παρατηρήθηκε ότι η πλειοψηφία των μετρικών παραμένει σε φυσιολογικά επίπεδα. Οι μετρικές αυτές είναι οι: **Lines of Code** (LoC), **Number of Comments** (NoC), **Number of Statements** (NoS), **Complexity of Conditions** (COM), **Number of Authors** (Authors), **Complexity Weighted by Decision Depth** (Depth) καθώς και η αξιολόγηση από τους προγραμματιστές (**Rate**). Μία μέση μέθοδος συγκρατεί τα νούμερα των παραπάνω μετρικών σε τέτοιο επίπεδο, ούτως ώστε να θεωρείται εύκολα συντηρήσιμη και κατανοητή. Οι μετρικές που αποκλίνουν από τα φυσιολογικά επίπεδα είναι η **Coupling between Objects** (CBO) και η **Lack of Cohesion in Methods** (LCOM2). Η πρώτη, έχει αρκετά χαμηλό μέσο όρο, που υποδεικνύει ότι, υπάρχει έλλειψη στην χρήση αντικειμένων και στις κλήσεις μεθόδων μέσα από αυτά. Αντιθέτως, η **Lack of Cohesion in Methods**, σε σχετικά μικρές μεθόδους κρατάει το επίπεδό της αρκετά χαμηλά, ενώ αντίθετα σε μεθόδους μεγάλες και δύσκολες στην κατανόηση αυξάνεται ραγδαία.

Για την αποφυγή ανεπιθύμητων τιμών, συνιστάται η συχνή χρήση εργαλείων ανάλυσης και αξιολόγησης κώδικα. Πρόγραμμα όπως και αυτό που αναπτύχθηκε, απαιτούν χρόνο για την εκτέλεση τους, οπότε, για την ελαχιστοποίηση της διαδικασίας, προτείνεται η χρήση ανάλυσης σε καινούργια πακέτα και κλάσεις, καθώς και σε αυτά που έχουν τροποποιηθεί, και όχι σε ολόκληρο το έργο.

4.2 Μελλοντικές Επεκτάσεις

Για μελλοντική επέκταση της έρευνας, μερικές προσθήκες στο πλήθος των μετρικών θα μπορούσαν να φανούν χρήσιμες για κάθε ομάδα ανάπτυξης λογισμικού. Ορισμένες από αυτές είναι οι:

- **Code Duplication**, αφορά τμήματα κώδικα που έχουν υλοποιηθεί επανειλημμένα, ενώ θα μπορούσε να αναπτυχθεί μία κοινή μέθοδος για αυτόν τον σκοπό
- **Number of Bugs**, Υπολογισμός "bugs" στο σύστημα, όσο λιγότερα τόσο το καλύτερο
- **Mean Time to Repair**, εφόσον υπάρχουν σφάλματα στο σύστημα, να υπολογίζεται ένας μέσος χρόνος επισκευής
- **Number of Test Cases**, ο αριθμός των τεστ που έχουν διεξαχθεί, ένας υψηλός αριθμός υποδηλώνει πόσο διεξοδικά έχει ελεγχθεί το αντικείμενο

Τέλος, σε συνδυασμό με όλες τις υπάρχουσες μετρικές θα μπορούσαν να βγουν και τιμές για τους δείκτες "Readability", "Reusability" και "Extendibility", όπου αφορούν τις έννοιες ευκολία στην ανάγνωση και κατανόηση, επαναχρησιμοποίηση, και επεκτασιμότητα αντίστοιχα.

Βιβλιογραφία

Bandari, K., 2022. *Complete ABAP.* 3rd ed. SAP Press.

Gamma, E., Helm, R., Johnson, R. & John, V., 1994. *Design patterns : elements of reusable object-oriented.* 1st ed. Addison-Wesley Professional.

Garner, J., 2020. *SAP ABAP The Ultimate Begginer's Guide.* 1st ed. Independently published .

Haeuptle, K. et al., 2020. *Clean ABAP A Style Guide for Developers.* 1st ed. SAP Press.

McCue, I., 2022. *www.netsuite.com.* [Online] Available at: https://www.netsuite.com/portal/resource/articles/erp/erp-benefits.shtml [Accessed 10 January 2023].

SAP-Press, 2021. *blog.sap-press.com.* [Online] Available at: https://blog.sap-press.com/abap-syntax-an-overview [Accessed 10 January 2023].

TutorialsPoint, 2018. *tutorialspoint.* [Online] Available at: https://www.tutorialspoint.com/sap_abap/index.htm [Accessed 14 January 2023].

Wood, J. & Rupert, J., 2015. *Object-Oriented Programming with ABAP Objects.* 2nd ed. SAP Press.