ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Reverse Engineering Plug In για το εργαλείο Eclipse

Των φοιτητών

Μοσχόπουλου Σωτήριου

και Μπαλατσού Ιωάννη

Αρ. Μητρώου: 123888 και 123885

Επιβλέπων καθηγητής

κ. Σφέτσος Παναγιώτης

Θεσσαλονίκη 2017

ΠΡΟΛΟΓΟΣ

Ένα χαρακτηριστικό που μας κάνει ανθρώπους είναι η προσπάθεια να καταλάβουμε τον κόσμο γύρω μας. Είναι ο λόγος που χτίζουμε επιταχυντές σωματιδίων, ταξιδεύουμε στο διάστημα, και κάνουμε επιστημονικές έρευνες γενικά. Μας αρέσει να διαλύουμε πράγματα μέχρι το τελευταίο τους κομμάτι και να το ξανασυναρμολογούμε. Αντίστροφη μηχανική (Reverse Engineering) σημαίνει να δουλεύεις “ανάποδα” απο κάτι που είναι ανθρώπινο δημιούργημα, ώστε να αποκτήσεις γνώση για τις εσωτερικές λειτουργίες του ή τον υποκειμενικό του σχεδιασμό. Ο προγραμματισμός χρησιμοποιώντας το testing (Test-driven Development TDD) είναι μια προσέγκιση που χρησιμοποιεί ένα αναπτυσσόμενο αριθμό από τεστ (tests) τα οποία οδηγούν και υποστηρίζουν την παραγωγή κώδικα. Στην δικιά μας περίπτωση, με αφορμή του paper Towards a framework based Test-driven Reverse Engineering Process: A Case Study των κ. Σφετσου, κ.Ανγγελή και κ. Σταμέλου [1], προσπαθούμε να ενώσουμε αυτές τις δυο μεθόδους και να την εφαρμόσουμε στο εργαλείο Eclipse σαν plug in. Ουσιαστικά σε αυτή την μελέτη που έγινε, εμπειρικά, γίνεται η προσπάθεια διερεύνησης της προσαρμογής του TDD στο περιβάλλον της Αντίστροφης Μηχανικής Λογισμικου (Software Reverse Engineering SRE).

ΠΕΡΙΛΗΨΗ

Καθώς η αντίστροφη μηχανική ξεκίνησε με την ανάλυση του υλικού μέρους των υπολογιστών (hardware), σήμερα παίζει σημαντικό ρόλο στον κόσμο του λογισμικού. Η αντίστροφη μηχανική λογισμικού είναι η διαδικασία ανάλυσης ενός λογισμικού συστήματος, είτε εν μέρη είτε ολόκληρο, για την εξαγωγή σχεδίασης και εκτέλεσης της πληροφορίας. Ένα τυπικό σενάριο SRE θα περιείχε μια λειτουργία λογισμικού που λειτουργεί χρόνια και “κουβαλάει” αρκετούς κανόνες μιας επιχείρησης στις γραμμές του κωδικά του. Όπως αναφέρθηκε και προηγουμένως, σκοπός της εργασίας αυτής είναι η εφαρμογή της παραπάνω μελέτης στο εργαλείο Eclipse. Αρχικά δίνεται πρόλογος που περιγράφει τις δυο βασικές έννοιες που χρησιμοποιούμε στην εργασία και γίνεται μια σύντομη περιγραφή της μελέτης που έγινε [1]. Ακολουθούν τέσσερα κεφάλαια όπου στο κεφάλαιο της εισαγωγής αναφέρουμε το περιεχόμενο τους.

Λέξεις Κλειδιά – Προγραμματισμός οδηγούμενος απο το testing; Αντίστροφη Μηχανική; Eclipse; Αντίστροφη Μηχανική οδηγούμενη από testing;

ABSTRACT

While reverse engineering probably started with the analysis of hardware, today it plays a significant role in the software world. Software Reverse Engineering (SRE) is the practice of analyzing a software system, either in whole or in part, to extract design and implementation information. A typical SRE scenario would involve a software module that has for years and carries several rules of business in its lines of code. As we mentioned earlier, the purpose of this work is to apply the above study as an Eclipse plug-in. First, there is a preface describing the two basic concepts we use in the work and a brief description of the study made [1]. Here are four chapters where we mention their content in the chapter of the introduction.

Keywords – Test-driven Development; Reverse Engineering; Eclipse; Test-driven Reverse Engineering;

ΕΥΧΑΡΙΣΤΙΕΣ (προαιρετικά)

Κατάλογος περιεχομένων

Ευρετήριο πινάκων

Ευρετήριο σχημάτων

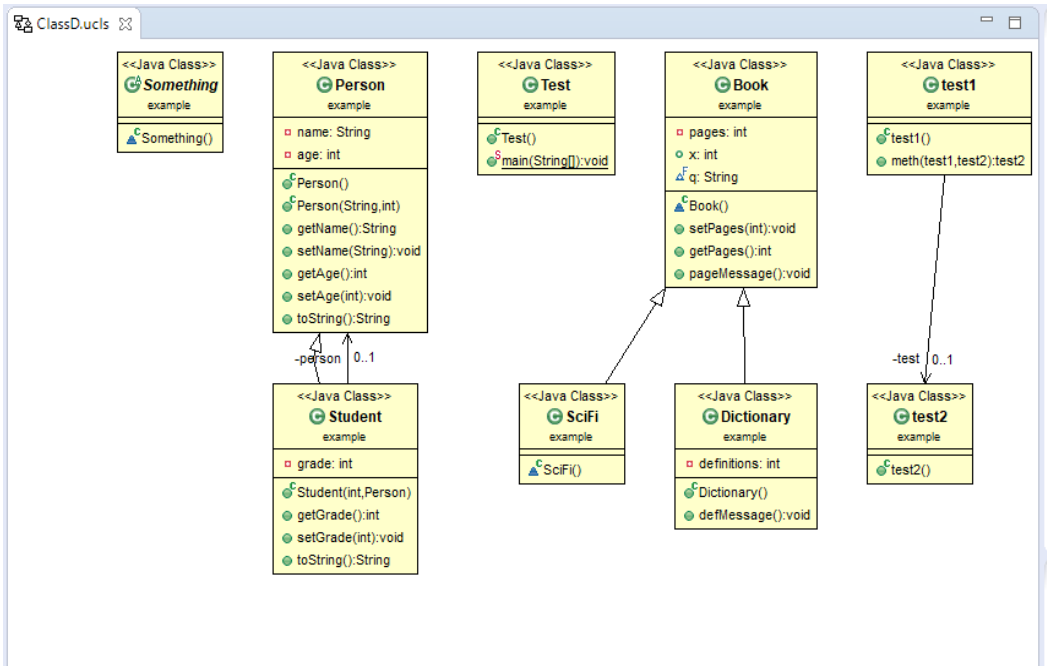
ΕΙΣΑΓΩΓΗ

Στόχος αυτής της Πτυχιακής Εργασίας είναι η υλοποίηση του παραπάνω framework στο εργαλείο Eclipse. Σκοπός αυτού του framework είναι η βοήθεια διάφορων δραστηριοτήτων Αντίστροφης Μηχανικής (RE), όπως η κατανόηση κώδικα και σχεδίασης, η ανάκτηση σχεδίου, και η εκ νέου σχεδίαση για τον εφαρμοσμένο κώδικα.

Το συγκεκριμένο framework προτείνει την ιεράρχηση των εφαρμοσμένων λειτουργιών και ως ακολούθως την κατά προτεραιότητα εξέταση μονάδων (uni-testing). Στα κεφάλαια που θα ακολουθήσουν θα κάνουμε μια ιστορική αναδρομή όσο αναφόρα το Reverse Engineering και το Test-driven Development, κάποιες εφαρμογές του RE, αναφορά στο εργαλείο Eclipse, τα αποτελέσματα που θα προκύψουν απο την εφαρμογή του framework, κώδικα καθώς και οδηγίες χρήσης του λογισμικού.

Αρχικά , έγινε μια έρευνα αγοράς για να δούμε εάν υπάρχει κάπου υλοποιημένη αυτή η φιλοσοφία του συγκεκριμένου framework. Ένα εργαλείο που βρήκαμε στην αρχή ήταν το Eclipse UML Generators του Eclipse, το οποίο, ουσιαστικά έκανε παραγωγή διαγραμμάτων UML και συγκεκριμένα Class Diagrams χωρίς όμως να χρησιμοποιείται το Testing. Μια γρήγορη περιγραφή του εργαλείου αυτού είναι: Το Eclipse UML Generators παρέχει στοιχεία που γεφυρώνουν αυτόματα το χάσμα μεταξύ μοντέλων UML και πηγαίου κώδικα. Αυτό επιτυγχάνεται, είτε με την εξαγωγή δεδομένων από μοντέλα UML (και προφίλ ή μοντέλα δακόσμησης) για την παραγωγή πηγαίου κώδικα ή με κώδικα πηγής Αντίστροφης Μηχανικής για την παραγωγή μοντέλων UML. Επίσης, χρησιμοποιεί το Acceleo, το οποίο είναι πραγματιστική εφαρμογή του προτύπου Object Management Group (OMG) MOF σε γλώσσα κειμένου (MTL). Είναι βαθμιαία και μπορεί να επεκταθεί χάρη στο πρωταρχικό του σύστημα. Η κατανάλωση ή παραγωγή μοντέλων UML βασίζεται στην εφαρμογή MDT UML2. Η UML (Unified Modeling Language) είναι η πιο χρησιμοποιημένη προδιαγραφή της OMG.

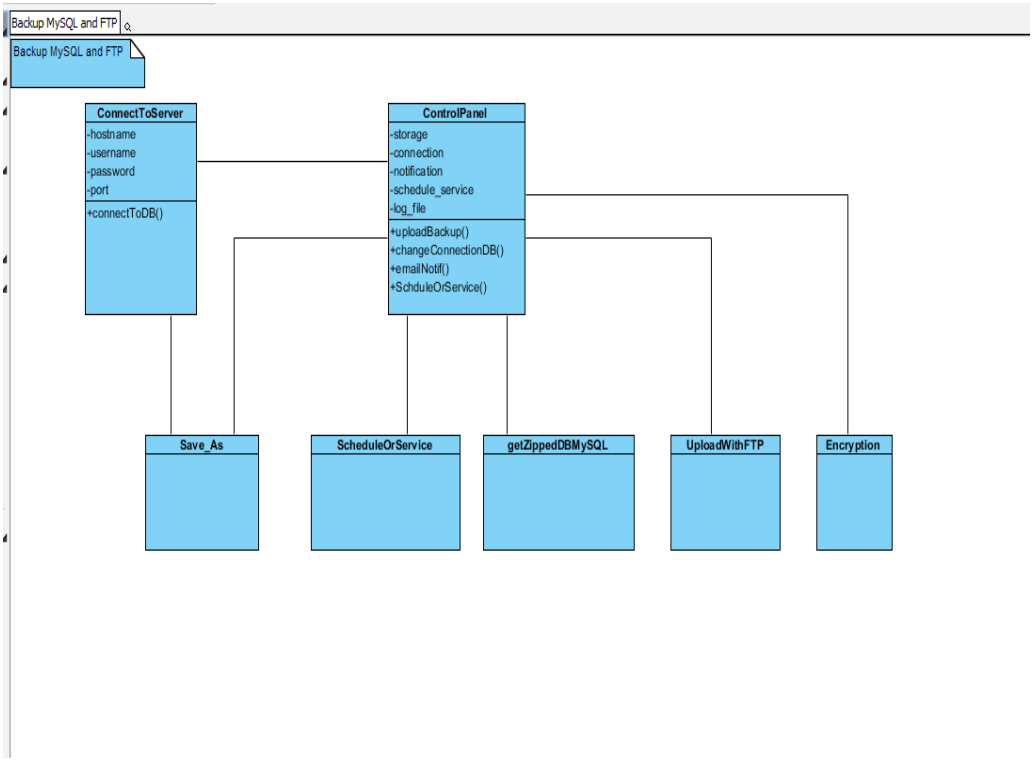
Ένα παράδειγμα για το πως φαίνεται ένα διάγραμμα κλάσης με το συγκεκριμένο εργαλείο φαίνεται στην παρακάτω εικόνα:



Σχήμα 1: «Class Diagram με το εργαλείο Eclipse UML Generators.»

Ένα άλλο εργαλείο που συναντήσαμε είναι της Visual Paradigm που ονομάζεται Round-trip Engineering. Το Visual Paradigm υποστηρίζει τη δημιουργία πηγαίου κώδικα Java από UML Class Diagrams και τη συγχρονισμένη κωδικοποίηση του πηγαίου κώδικα και του μοντέλου UML, μέσω της τεχνολογίας Java Round-trip Engineering. Το Round-trip Engineering παρέχεται μέσω του πακέτου Visual Paradigm Standard και είναι επι πληρωμής. Επίσης, η ίδια διαδικασία είναι διαθέσιμη και για τη γλώσσα προγραμματισμού C++. Αυτό που δεν μπορέσαμε να βρούμε κατά την έρευνα αγοράς μας, ήταν το πως λειτουργεί αυτό το πακέτο που προσφέρει το Visual Paradigm, δηλάδη να δούμε αν χρησιμοποιείται παρόμοια φιλοσοφία με αυτή που υλοποιούμε εμείς.

Ένα παράδειγμα για το πως φαίνεται ένα διάγραμμα κλάσης με το συγκεκριμένο εργαλείο φαίνεται στην παρακάτω εικόνα:



Σχήμα 2: «Class Diagram με το εργαλείο του Visual Paradigm.»

Άλλα παρόμοια εργαλεία που συναντήσαμε είναι η ArgoUML, το Microsoft Visual Studio, το NetBeans, το PlantUML. Όμως τα δυο εργαλεία που αναφέραμε στην αρχή είναι τα πιο ευρέως διαδεδομένα και τα πιο εύχρηστα.

Μια από τις δυσκολίες που συναντήσαμε στην αρχή είναι το πως θα συλλέγαμε τις πληροφορίες μιας κλάσης, ώστε μέσα από αυτές τις πληροφορίες να εμφανίσουμε αυτό που θέλουμε.

Αρχικά, κάναμε μια αναζήτηση για το αν υπάρχουν βιβλιοθήκες της Java οι οποίες κάνουν ακριβώς αυτή την εργασία. Αναζητήσαμε για βιβλιοθήκες οι οποίες κάνουν parse τα αρχεία με την κατάληξη .java. Όντως, ανακαλύψαμε μια τέτοια βιβλιοθήκη, η οποία ονομάζεται JavaParser. Όμως, ενώ λειτουργούσε αποτελεσματικά ως βιβλιοθήκη, ήταν δύσκολη στο χειρισμό της και απαιτούσε πολλές γραμμές κώδικα για να την συλλογή μιας πληροφορίας, όπως για παράδειγμα την ανάλυση μιας κλάσης για να δούμε τις μεταβλητές που έχει. Έτσι, είπαμε να συνεχίσουμε την αναζήτηση μας μήπως και βρούμε κάτι πιο εύχρηστο. Άλλο ένα εργαλείο που βρήκαμε ήταν το Reflection API, το οποίο χρησιμοποιείται συνήθως από προγράμματα που απαιτούν τη δυνατότητα εξέτασης ή

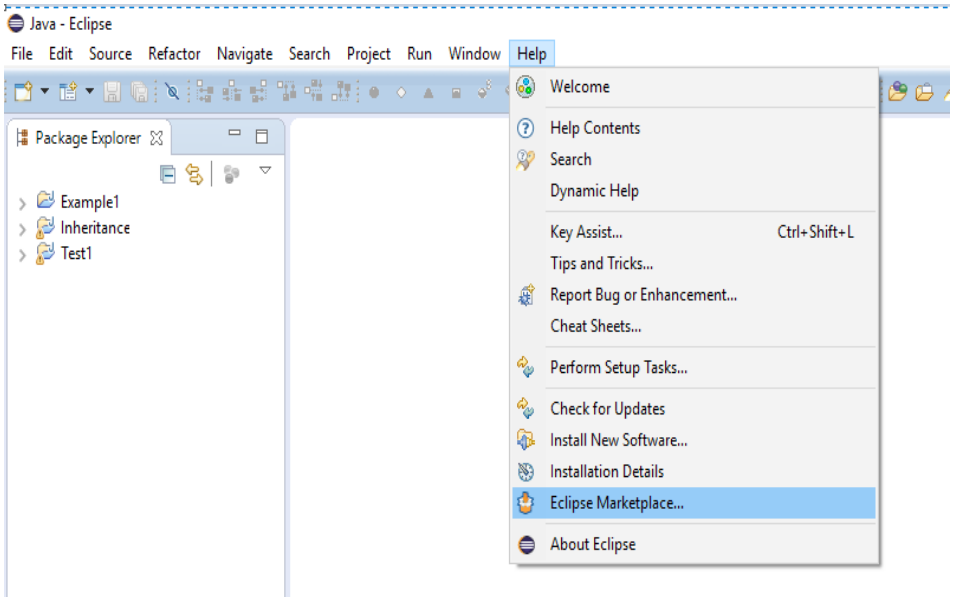
τροποποίησης της συμπεριφοράς κατά την εκτέλεση εφαρμογών που εκτελούνται σε εικονικό μηχάνημα της Java. Όμως αυτό που βρήκαμε για το συγκεκριμένο εργαλείο, ήταν ότι απαιτεί τη φόρτωση και την εκτέλεση του .class αρχείου. Επομένως, ήταν μια βιβλιοθήκη η οποία ήταν αδύνατον να χρησιμοποιηθεί, για τα δικά μας δεδομένα.

Έν τέλει, βρήκαμε τη βιβλιοθήκη ClassParser της Apache, η οποία με λίγες αυτή τη φορά εντολές, μας έδινε τις πληροφορίες που θέλαμε. Συγκεκριμένα, είναι μια Wrapper Class που αναλύει ένα αρχείο Java .class. Η μέθοδος ανάλυσης επιστρέφει ένα αντικείμενο JavaClass. Όταν εμφανιστεί ένα σφάλμα εισόδου/εξόδου ή μια ασυνέπεια, μια κατάλληλη εξαίρεση μεταδίδεται πίσω σε αυτόν που την έχει καλέσει. Η δομή και τα ονόματα συμμορφώνονται, εκτός από μερικές ανέσεις, ακριβώς με την προδιαγραφή JVM 1.0. Εκτενέστερη ανάλυση της συγκεκριμένης βιβλιοθήκης και των περιεχομένων της θα γίνει στα επόμενα κεφάλαια.

Όσο αναφορά την υλοποίηση του plug-in, ήταν ξεκάθαρο ότι θα χρησιμοποιούσαμε εργαλείο του Eclipse από την στιγμή που το plug-in είναι για το Eclipse.

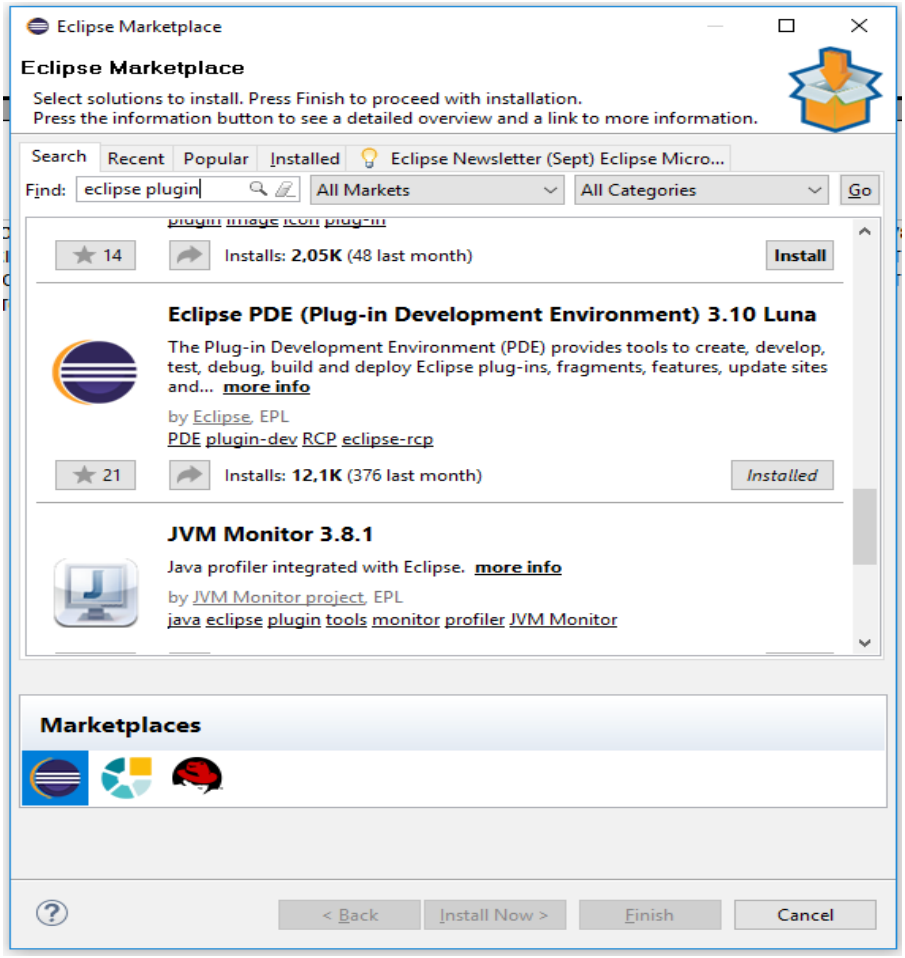
Αναζητώντας στο Eclipse Marketplace βρήκαμε το Eclipse PDE (Plug-in Development Environment), το οποίο περιγράφουμε παρακάτω στα επόμενα κεφάλαια. Παρακάτω δείχνουμε τη διαδικασία με την οποία κατεβάζει κανείς το εργαλείο αυτό και το πως γίνεται η εγκατάσταση του.

Αρχικά στο μενού επάνω στην επιλογή **Help**, επιλέγουμε το **Eclipse Marketplace...** όπως φαίνεται στην παρακάτω εικόνα:



Σχήμα 3: «Επιλογή του Eclipse Marketplace.»

Στη συνέχεια εμφανίζεται το παράθυρο Eclipse Marketplace, στο οποίο μπορεί κανείς να κάνει αναζήτηση διάφορα plug-in, εφαρμογές και extensions για το Eclipse. Στην αναζήτηση που κάναμε, όπως αναφέραμε, μας εμφάνισε το PDE όπως φαίνεται στην παρακάτω εικόνα:



Σχήμα 4: «Το PDE στο Eclipse Marketplace.»

Για να γίνει η εγκατάσταση του συγκεκριμένου plug-in, αυτό που πρέπει να κάνει κάποιος είναι απλά να πατήσει το button **Install Now >**.

Είναι χρήσιμο να τονιστεί επίσης, ότι σε κάποια φάση της εκπόνησης της Πτυχιακής Εργασίας μας, χρειάστηκε να δουλέψουμε από δύο διαφορετικές πόλεις. Εκεί μας βοήθησε πολύ το εργαλείο GitHub. Το GitHub είναι ένα web-based Git ή ένα αποθετήριο ελέγχου έκδοσης (Version Control Repository) και μια υπηρεσία φιλοξενίας μέσω διαδικτύου. Χρησιμοποιείται κυρίως για κώδικα. Προσφέρει όλες τις λειτουργίες του Git για τον έλεγχο κατανεμημένης έκδοσης (Distributed Version Control) και τη διαχείριση πηγαίου κώδικα (Source Code Management), καθώς και για την προσθήκη των δικών του χαρακτηριστικών.

Επίσης, παρέχει έλεγχο και πολλές λειτουργίες συνεργασίας όπως παρακολούθηση σφαλμάτων, αιτήματα χαρακτηριστικών, διαχείριση εργασιών και wiki για κάθε έργο. Περισσότερη ανάλυση, βέβαια, θα γίνει σε ξεχωριστό κεφάλαιο όπου περιγράφουμε πως λειτουργεί συγκεκριμένα αυτό το εύχρηστο εργαλείο.

Τέλος, λίγα λόγια σχετικά με τον τρόπο βαθμολόγησης των κλάσεων και των μεθόδων τις οποίες θα εμφανίσει το εργαλείο στο τέλος. Οι κλάσεις έχουν ως παράμετρο μόνο τα αντικείμενα άλλων κλάσεων καθώς και τις μεταβλητές που προσθέτει η Java. Σκοπός πίσω από αυτή τη σκέψη είναι η σωστή και ορθή δόμηση μιας κλάσης που θα βασίζεται στην αλληλεπίδραση, την χρήση και θα εξαρτάται από άλλες κλάσεις. Με λίγα λόγια το να δομήσει κάποιος μια κλάση απλά που θα περιέχει μεθόδους δημόσιες ή στατικές ή ακομα και πολλές ιδιωτικές και να τις έχει σε ένα σημείο και από να τις καλεί στον υπόλοιπο κώδικα του με τη χρήση αντικειμένου ή απλά με την κλάση. Πιστεύουμε ότι αυτή δεν είναι σωστή δόμηση οπότε και δεν λαμβάνουμε υπόψη μας τον αριθμό των μεθόδων που έχει μια κλάση, καθώς σωστή και πλήρης κλάση θεωρούμε αυτή που περιέχει και αντικείμενα άλλων κλάσεων και μέσω αυτών γίνεται κλάση μεθόδων που βοηθάνε στην υλοποίηση και εξαγωγή των σωστών αποτελεσμάτων. Το εργαλείο έχει την δυνατότητα να εντοπίσει όλα τα στοιχεία που περιέχει μια κλάση που σημαίνει ότι θα μπορούσε να είναι πολύ πιο ευέλικτο και να προσθέτει πολύ περισσότερες δυνατότητες και να εντοπίζει με μεγαλύτερη ακρίβεια τις σημαντικές κλάσεις και μεθόδους τους αλλά έτσι θα χανόταν η ουσία της δόμησης σωστών και ολοκλήρομένων κλάσεων.

Στις μεθόδους από την άλλη ελέγχουν τον τύπο προσβασιμότητας και το πόσα ορίσματα δέχεται, αν δέχεται, τι επιστρέφει ή αν είναι τύπου void και έτσι βαθμολογούμε τις μεθόδους. Θα μπορούσε να προστεθεί εδώ και η έκταση των γραμμών που καταλαμβάνει μια μέθοδος αλλά αυτό πιστεύουμε είναι ανούσιο απο την στιγμή που δεν είναι σίγουρο ότι ο προγραμματιστής θα γράψει όλη την λογική και τον κώδικα μέσα στο σώμα μιας μεθόδου ή θα επιλέξει να σπάσει τον κώδικα με επιμέρους πολλές ή λίγες βοηθητικές ώστε να έχει την δυνατότητα να τις χρησιμοποιήσει και σε άλλα σημεία του κωδικά του. Έτσι οι μέθοδοι όσο μεγάλες ή μικρές και αν είναι δεν αποτελούν παράγοντα βαθμολόγισης.

Το τελικό αποτέλεσμα που εμφανίζεται στον χρήστη είναι όλες οι κλάσεις με τα πεδία τους και τις μεθόδους τους κατα βαθμολογική σειρά απο την πιο σημαντική προς την λιγότερο σημαντική.

ΚΕΦΑΛΑΙΟ 1

Ιστορική Αναδρομή

ΕΙΣΑΓΩΓΗ

Ο όρος Αντίστροφη Μηχανική προέρχεται από την ανάλυση του υλικού μέρους του υπολογιστή (hardware) – όπου η μέθοδος της αποκωδικοποίησης σχεδίων από τελικά προϊόντα είναι κάτι κοινό.

Η Αντίστροφη Μηχανική, όπως αναφέρθηκε και προηγουμένως, συνήθως εφαρμόζεται για την βελτίωση των προϊόντων, καθώς και στην ανάλυση των ανταγωνιστικών προϊόντων ή σε κατάσταση στρατιωτικής ή εθνικής ασφάλειας.

Γεγονότα στην πρόσφατη ιστορία έχουν κάνει την αντίστροφη μηχανική λογισμικού να είναι ένα ενεργό πεδίο ως προς την έρευνα.

ΥΠΟΚΕΦΑΛΑΙΟ 1.1

Η Αντίστροφη Μηχανική στον Β' Παγκόσμιο Πόλεμο και στον Ψυχρό Πόλεμο

Η Αντίστροφη Μηχανική χρησιμοποιείται συχνά από τον στρατό με σκοπό να αντιγράψουν τεχνολογίες από άλλα έθνη, συσκευές ή πληροφορίες, ή μέρη των οποίων, έχουν ληφθεί από κανονικά στρατεύματα ή από επιχειρήσεις πληροφοριών. Χρησιμοποιούνταν συχνά κατά την διάρκεια του Β' Παγκοσμίου Πολέμου και του Ψυχρού Πολέμου. Τα πιο γνωστά παραδείγματα από τον Β' Παγκόσμιο και αργότερα συμπεριλαμβάνουν :

- Την περίπτωση Jerry can (σημαίνει μπιτόνι βενζίνης) : Οι Βρετανικές και Αμερικανικές δυνάμεις παρατήρησαν ότι οι Γερμανικές δυνάμεις είχαν βαρέλια με βενζίνη που είχαν εξαιρετικό σχεδιασμό. Αυτό που έκαναν είναι να εφαρμόζουν αντίστροφη μηχανική και να δημιουργήσουν αντίγραφα από τα μπιτόνια των Γερμανών. Τα μπιτόνια αυτά έμειναν γνωστά ως “Jerry cans”.

- Την περίπτωση Tupolev Tu-4 : Τρία αμερικάνικα βομβαρδιστικά τύπου B-29 σε αποστολή πάνω από την Ιαπωνία αναγκάστηκαν να προσγειωθούν σε εδάφη της Σοβιετικής Ένωσης. Οι Σοβιετικοί, οι οποίοι δεν είχαν παρόμοια βομβαρδιστικά, αποφάσησαν να αντιγράψουν τα βομβαρδιστικά

τύπου B-29. Μέσα σε λίγα χρόνια είχαν αναπτύξει τα Tu-4, ένα σχεδόν τέλειο αντίγραφο.

- Την περίπτωση V2 Rocket : Τεχνικά έγγραφα για το V2 και τις σχετικές τεχνολογίες καταγράφηκαν από τους Δυτικούς Συμμάχους στο τέλος του πολέμου. Οι Σοβιετικοί και αιχμαλωτισμένοι Γερμανοί μηχανικοί αναγκάστηκαν να αναπαράγουν τεχνικά έγγραφα και σχέδια, που εργάζονταν από το υλικό που είχαν συλλέξει, για να κάνουν τον κλώνο του πυραύλου, τον R-1, ο οποίος ξεκίνησε το μεταπολεμικό σοβιετικό πρόγραμμα πυραύλων που οδήγησε στο R-7 και στην αρχή του αγώνα για την κατάκτηση του διαστήματος.

- Τον πύραυλο K-13/R3S (όνομα αναφοράς από το NATO AA-2 Atoll) : Ένα σοβιετικό αντίγραφο από το AIM-9 Sidewinder που χρησιμοποιήθηκε αντίστροφη μηχανική πάνω του, που κατέστει δυνατό μετά την απόκτηση ενός ταϊβανέζικου AIM-9B που χτύπησε ένα κινέζικο MiG-17 χωρίς να εκραγεί. Όλως τυχαίως, ο πύραυλος σφήνωσε στην άτρακτο του αεροπλάνου, και ο πιλότος επέστρεψε στην βάση με αυτόν.

- Τον πύραυλο BGM-71_TOW : Τον Μάϊο του 1975, μετά από διαπραγματεύσεις μεταξύ του Ιράν και των πυραυλικών συστημάτων Hughes για την συνεργασία για την παραγωγή των πυραύλων TOW και Maverick, οι οποίες σταμάτησαν λόγω διαφωνίων που αφορούσαν την τιμολόγηση. Η επακόλουθη επανάσταση του 1979 τερμάτισε όλα τα σχέδια για μία τέτοια συνεργασία. Το Ιράν αργότερα κατάφερε να εφαρμόσει αντίστροφη μηχανική στον συγκεκριμένο πύραυλο, και να δημιουργήσει το δικό του αντίγραφο το The Toophan.

ΥΠΟΚΕΦΑΛΑΙΟ 1.2

Νομιμότητα

Στις Ηνωμένες Πολιτείες και σε πολλά άλλα κράτη, ακόμη και αν ένα τεχνούργημα ή μια διαδικασία προστατεύεται από εμπορικά μυστικά, στην αντίστροφη μηχανική το τεχνούργημα ή η διαδικασία είναι συχνά νόμιμη, εφόσον λαμβάνεται νόμιμα. Οι “πατέντες” από την άλλη μεριά, χρειάζονται μια δημόσια αποκάλυψη μιας πιθανής εφεύρεσης, και συνεπώς, οι κατοχυρωμένες “πατέντες” δεν χρειάζεται να ανασχηματιστούν ώστε να μελετηθούν. Ένα κοινό κίνητρο της

αντίστροφης μηχανικής είναι να προσδιορίσει αν το προϊόν ενός ανταγωνιστή περιέχει παραβιάσει ευρεσιτεχνιών ή παραβιάσεις πνευματικών δικαιωμάτων.

Η αντίστροφη μηχανική υλικών ή λογισμικών συστημάτων η οποία γίνεται για λόγους διαλειτουργικότητας (για παράδειγμα, για την υποστήριξη μορφών αρχείων χωρίς έγγραφα ή μη πιστοποιημένων περιφερειακών συσκευών), θεωρείται ως επι το πλείστον νόμιμο, αν και οι κάτοχοι των “πατεντών” συχνά το αμφισβητούν και προσπαθούν να καταπνίξουν την αντίστροφη μηχανικη των προϊόντων τους για οποιοδήποτε λόγο.

Η αντίστροφη μηχανική λογισμικού στις Ηνωμένες Πολιτείες είναι γενικά παράνομη επειδή οι περισσότεροι EULA (End User License Agreement) την απαγορεύουν και τα δικαστήρια έχουν βρεί συμβατικές απαγορεύσεις που να υπερισχύουν του νόμου περί πνευματικών δικαιωμάτων.

ΥΠΟΚΕΦΑΛΑΙΟ 1.3

Το πρόβλημα Y2K

Στις αρχές του '90, το πρόβλημα Y2K δημιούργησε την ανάγκη για την ανάπτυξη εργαλείων που θα μπορούσαν να διαβάσουν μεγάλες ποσότητες πηγαίου ή δυαδικού κώδικα για την ευπάθεια αυτή που προέκυψε.

Λίγο μετά την προετοιμασία για το πρόβλημα Y2K, στα μέσα προς τέλη του '90, η υιοθέτηση του διαδικτύου από διάφορες επιχειρήσεις και οργανισμούς δημιουργήθηκε η ανάγκη για την κατανόηση των εσωτερικών συστημάτων ώστε η πληροφορία που ήταν αποθηκευμένη σε αυτά να είναι διαθέσιμη στο διαδίκτυο.

Η επιθυμία των επιχειρήσεων να επεκταθούν στην αγορά του διαδικτύου και οι επίλυση τέτοιων προβλημάτων όπως του Y2K δημιούργησε επίσης και την ανάγκη για την αντίστροφη μηχανική και στο λογισμικό.

ΕΠΙΛΟΓΟΣ

Βλέπουμε ότι η Αντίστροφη Μηχανική χρησιμοποιείται εδώ και πολλά χρόνια, αν και βέβαια σε επίπεδο hardware και περισσότερο για στρατιωτικούς σκοπούς. Μετά την ανάπτυξη της τεχνολογίας και των ηλεκτρονικών υπολογιστών η Αντίστροφη Μηχανική χρησιμοποιήθηκε και στα λογισμικά. Παρακάτω θα δούμε κάποιες χρήσεις της.

ΚΕΦΑΛΑΙΟ 2

Χρήσεις Αντίστροφης Μηχανικής

ΕΙΣΑΓΩΓΗ

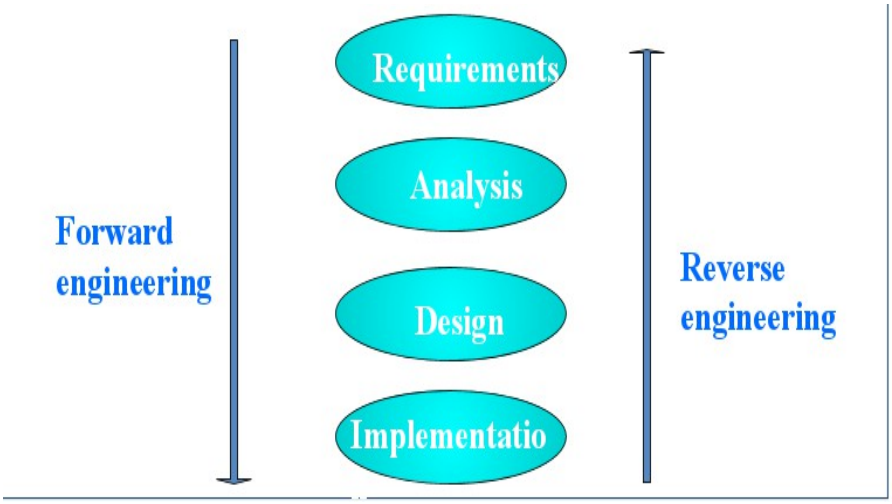
Με την πρώτη ματιά, φαίνεται ότι η ανάγκη για αντίστροφη μηχανική λογισμικού μπορεί να μειωθεί απλά διατηρώντας καλή τεκμηρίωση για όλο το λογισμικό που είνια γραμμένο. Καθώς η παρουσία αυτής της ιδέας θα ήταν ιδανική, στην πραγματικότητα δεν συμβαίνει κάτι τέτοιο. Για παράδειγμα, ακόμη και μία εταιρία που έχει βγάλει λογισμικό στην αγορά, δεν μπορεί πλέον να το καταλάβει, επειδή οι αρχικοί σχεδιαστές και προγραμματιστές του λογισμικού μπορεί να έχουν φύγει από την εταιρία ή το λογισμικό μπορεί να έχει αποκτηθεί από έναν πωλητή που δεν βρίσκεται πλέον στην εταιρία.

Προχωρώντας, το όραμα είναι είναι να συμπεριληφθεί η αντίστροφη μηχανική λογισμικού σταδιακά, ως μέρους της φυσιολογικής ανάπτυξης ή του “forward engineering” των συστημάτων. Σε τακτικά σημεία κατά τη διάρκεια του κύκλου ανάπτυξης, ο κώδικας θα αντιστραφεί για να ανακαλυφθεί εκ νέου ο σχεδιασμός του, έτσι ώστε η τεκμηρίωση (το documentation δηλαδή) να ενημερωθεί. Αυτό θα βοηθήσει στην αποφυγή της τυπικής κάταστασης όπου οι λεπτομερείς πληροφορίες σχετικά με ένα σύστημα λογισμικού όπως η αρχιτεκτονική του, οι περιορισμοί σχεδίασης του και οι πωλήσεις βρίσκονται μόνο στην μνήμη του δημιουργού του.

ΥΠΟΚΕΦΑΛΑΙΟ 2.1

Η Αντίστροφη Μηχανική στην Ανάπτυξη Λογισμικού

Ο όρος Αντίστροφη Μηχανική όσο αναφορά την εφαρμογή του σε λογισμικό σημαίνει διαφορετικά πράγματα σε διαφορετικούς ανθρώπους, προτρέποντας τον Chikofsky και τον Cross να γράψουν ένα paper το οποίο μελετά τις διάφορες χρήσεις και σημασίες ενός taxonomy [2]. Σύμφωνα με αυτούς, Αντίστροφη Μηχανική είναι η διαδικασία ανάλυσης ενός υποκείμενου συστήματος για την δημιουργία μίας αναπαράστασης του συστήματος σε υψηλότερα αφαιρετικά επίπεδα (Σχήμα 1). Μπορεί, επίσης, να το δεί κανείς από την μεριά πηγαίνοντας αντίθετα του κύκλου ανάπτυξης.



Σχήμα 5: «Επιστροφή στην αρχή κύκλου ανάπτυξης ενός συστήματος.»

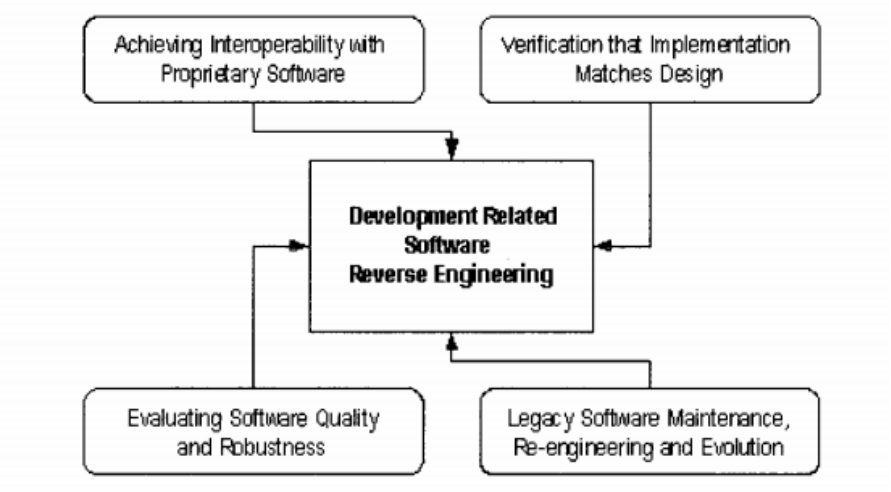
Σε αυτό το μοντέλο, τα αποτελέσματα αυτής της φάσης υλοποίησης (σε μορφή πηγαίου κώδικα) μετατρέπεται από την αντίστροφη μηχανική, προς την φάση της ανάλυσης, σε μια “αντιστροφή” του παραδοσιακού μοντέλου του καταρράκτη. Η αντίστροφη μηχανική είναι μια διαδικασία ελέγχου μόνο: το υπό εξέταση σύστημα λογισμικού δεν τροποποιείται (πράγμα που θα καθιστούσε τον ανασχεδιασμό). Η τεχνολογία αντικατάστασης λογισμικού χρησιμοποιείται για τη αποτροπή τόσο της αντίστροφης μηχανικής όσο και της αναδιοργάνωσης του ιδιόκτητου λογισμικού και των συστημάτων που λειτουργούν με λογισμικό.

Στην πράξη, εμφανίζονται δυο κύριοι τύποι αντίστροφης μηχανικής. Στην πρώτη περίπτωση, ο πηγαίος κώδικας είναι ήδη διαθέσιμος για το λογισμικό, αλλά ανακαλύπτονται πτυχές υψηλότερου επιπέδου του προγράμματος, που ίσως είναι ανεπαρκώς τεκμηριωμένες ή είναι τεκιμιριωμένες αλλά όχι πλέον έγκυρες. Στην δεύτερη περίπτωση, δεν υπάρχει διαθέσιμος πηγαίος κώδικας για το λογισμικό και όλες οι προσπάθειες για την ανακάλυψη ενός πιθανού πηγαίου κώδικα για το λογισμικό θεωρούνται αντίστροφη μηχανική. Η δεύτερη χρήση του όρου είναι και αυτή με την οποία είναι και οι περισσότεροι εξοικειωμένοι. Η αντίστροφη μηχανική του λογισμικού μπορεί να κάνει χρήση της τεχνικής σχεδιασμού του “καθαρού δωματίου” (clean room design technique) για να αποφευχθεί η παραβίαση των δικαιωμάτων πνευματικής ιδιοκτησίας.

Σε σχετική σημείωση, το testing μαυρου κουτιού (black box testing) στην τεχνολογία λογισμικού και πολλά κοινά με την αντίστροφη μηχανική. Ο δοκιμαστής (tester) έχει συνήθως το API, αλλά οι στόχοι του είναι συνήθως να βρεθούν σφάλματα και μη κατορθωμένα χαρακτηριστικά με την εκτόξευση του προϊόντος από το εξωτερικό.

Άλλοι στόχοι της αντίστροφης μηχανικής είναι ο έλεγχος ασφαλείας, όπως θα αναφερθεί στο επόμενο υποκεφάλαιο, η αφαίρεση της προστασίας απο την αντιγραφή (“cracking”), η καταστρατήγηση των περιορισμών πρόσβασης που συχνά υπάρχουν στα ηλεκτρονικά είδη ευρείας κατανάλωσης, η προσαρμογή των ενσωματωμένων συστημάτων (όπως τα συστήματα διαχείρισης του κινητήρα), οι εσωτερικές επισκευές ή μετασκευές.

Σύμφωνα με τον E. Eliam [3], υπάρχουν τέσσερα σενάρια σχετικά με την ανάπτυξη λογισμικού με την αντίστροφη μηχανική. Τα σενάρια αυτά καλύπτουν ένα ευρύ φάσμα δραστηριοτήτων που περιλαμβάνουν συντήρηση, επαναχρησιμοποίηση, ανασχεδιασμό, εξέλιξη, διαλειτουργικότητα και δοκιμή λογισμικού. Το σχήμα 1 τα συνοψίζει.



Σχήμα 6: «Σενάρια σχετικά με την ανάπτυξη Αντίστροφης Μηχανικής Λογισμικού.»

Τα παρακάτω είναι λειτουργίες που μπορούν να εκτελεστούν σε κάθε ένα από τα σενάρια [3]:

- Επίτευξη διαλειτουργικότητας με ιδιόκτητο λογισμικό: Ανάπτυξη εφαρμογών ή προγράμματα οδήγησης συσκευών (device drivers) που διαλειτουργούν (χρησιμοποιούν) αποκλειστικές βιβλιοθήκες σε λειτουργικά συστήματα ή εφαρμογές.
- Επαλήθευση ότι η υλοποίηση συμπίπτει με το σχέδιο: Επαλήθευση ότι ο παραγόμενος κώδικας κατά τη διάρκεια της ανάπτυξης με την μέθοδο “forward engineering” ταιριάζει με το σχεδιασμό του ανεστραμμένου κώδικα πίσω σε ένα αφηρημένο σχέδιο.
- Αξιολόγηση της ποιότητας και της ανθεκτικότητας του λογισμικού: Εξασφάλιση της ποιότητας του λογισμικού πριν την αγορά του, πραγματοποιώντας ευρετική ανάλυση των δυαδικών ψηφίων για έλεγχο ορισμένων ακολουθιών εντολών που εμφανίζονται σε κώδικα κακής ποιότητας.
- Συντήρηση παλαιού λογισμικού, ανασχεδιασμός και εξέλιξη: Ανάκτηση σχεδίων ενοτήτων παλαιού τύπου λογισμικού, όταν η πηγή δεν είναι διαθέσιμη για να καταστεί δυνατή η συντήρηση, η εξέλιξη και η επαναχρησιμοποίηση των ενοτήτων.

ΥΠΟΚΕΦΑΛΑΙΟ 2.2

Η Αντίστροφη Μηχανική στην Ασφάλεια Λογισμικού

Από την πλευρά μιας εταιρίας λογισμικού, είναι επιθυμητό τα προϊόντα της να μην είναι επιρρεπή στην πειρατεία και στην αντίστροφη μηχανική. Το να κάνεις λογισμικό που είναι δύσκολο να είναι επιρρεπές σε αντίστροφη μηχανική έρχεται σε σύγκρουση με την ιδέα να είναι σε θέση να ανακτηθεί αργότερα ο σχεδιασμός του λογισμικού για συντήρη και εξέλιξη. Ως εκ τούτου, οι κατασκευαστές λογισμικού συνήθως δεν εφαρμόζουν τέτοιους μηχανισμούς, που να αποτρέπουν την αντίστροφη μηχανική, έως ότου το προϊόν είναι έτοιμο να αποσταλεί στον πελάτη. Συνήθως, οι κατασκευαστές του λογισμικού θα επενδύσουν χρόνο στο να είναι το προϊόν δύσκολα επιρρεπές στην αντίστροφη μηχανική μόνο αν υπάρχουν αλγόριθμοι που κεντρίζουν το ενδιαφέρον και κάνουν το προϊόν να ξεχωρίζει από τον ανταγωνισμό.

Το να κάνεις το λογισμικό να είναι δύσκολο επιρρεπές στην πειρατεία και στην αντίστροφη μηχανική είναι συχνά κινούμενος στόχος και απαιτεί ειδικές δεξιότητες και κατανόηση από την μεριά του προγραμματιστή. Οι προγραμματιστές λογισμικού που έχουν την ευκαιρία να δοκιμάσουν τέτοιες τεχνικές μπορεί να είνια

σε καλύτερη θέση να βοηθήσουν τους εργοδότες τους, ή και τους ίδιους, ώστε να προστατέψουν την πνευματική ιδιοκτησία του προϊόντος. Όπως αναφέρει και ο M. R. Ali [4], “για να νικήσει έναν απατεώνα πρέπει να σκέφετσαι σαν έναν”. Το να εφαρμόζει κανείς αντίστροφη μηχανική σε ιούς ή άλλα κακόβουλα λογισμικά, οι προγραμματιστές θα πρέπει να μάθουν την εσωτερική τους λειτουγία και να γίνουν μάρτυρες από πρώτο χέρι πώς οι ευπάθειες βρίσκουν τρόπο να βλάπτουν τα προγράμματα υπολογιστών. Η αντίστροφη μηχανική σε λογισμικό που έχει μολυνθεί από ιό, είναι μια τεχνική, που χρησιμοποιείται από προγραμματιστές που δημιούργησαν τα λεγόμενα anti-virus, για τον εντοπισμό και την εξουδετέρωση νέων ιών ή ακόμα και την κατανόηση της συμπεριφοράς των κακόβουλων αυτών προγραμμάτων.

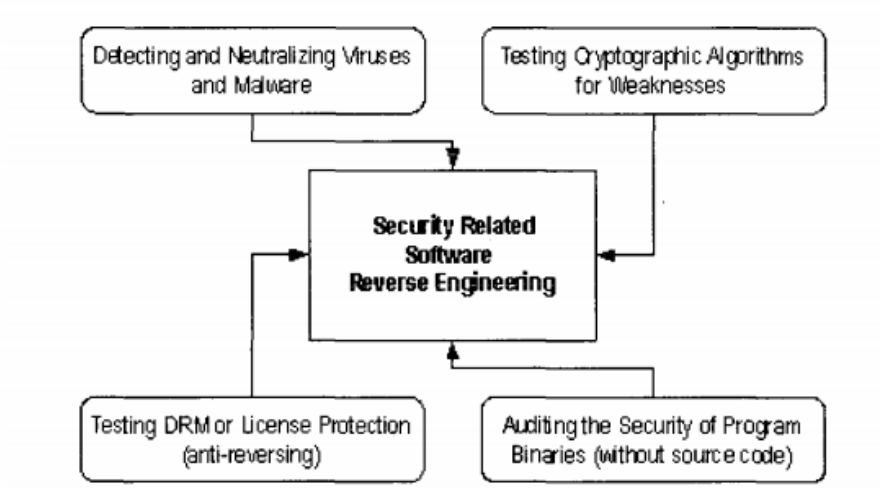
Γλώσσες προγραμματισμού όπως η Java, οι οποίες δεν απαιτούν από τους προραμματιστές την διαχείρηση λεπτομερειών χαμηλού επιπέδου, χρησιμοποιούνται σχεδόν παντού. Ως αποτέλεσμα, οι προγραμματιστές χάνουν όλο και περισσότερο την επαφή με αυτό που συμβαίνει στο υπολογιστικό σύστημα κατά την διάρκεια εκτέλεσης των προγραμμάτων. Ο M. R. Ali [4] δηλώνει ότι οι προγραμματιστές μπορούν να αποκτήσουν καλύτερη και βαθύτερη κατανόηση του λογισμικού και του υλικού μέσω της εκμάθηση ιδεών αντίστροφης μηχανικής. Οι χάκερ έχουν αποδείξει ότι έχουν μια βαθύτερη κατανόηση των λεπτομερειών του χαμηλού επιπέδου από ό,τι οι επαγγελματίες [4].

Σύμφωνα με τον E. Eliam [3], υπάρχουν τέσσερα σενάρια σχετικά με την αντίστροφη μηχανική πάνω στην ασφάλεια του λογισμικού. Παρόμοια με τα προηγούμενα σενάρια που αναφέρθηκαν, καλύπτουν ένα ευρύ φάσμα δραστηριοτήτων: διασφαλίζουν ότι το λογισμικό είνια ασφαλές για ανάπτυξη και χρήση, προστατεύουν έξυπνους αλγόριθμους ή επιχειρηματικές διαδικασίες, αποτρέποντας έτσι την πειρατεία λογισμικού και ψηφιακών μέσων όπως η μουσική, οι ταινίες και βιβλία, και βεβαιώνουν ότι οι αλγόριθμοι κρυπτογράφησης δεν είναι ευάλωτοι σε επιθέσεις. Το σχήμα 2 τα συνοψίζει. Τα παρακάτω είναι διαδικασίες μπορούν να εκτελεσθούν σε κάθε ένα από αυτά τα σενάρια:

- Ανίχνευση και εξουδετέρωση ιών και κακόβουλων προγραμμάτων: Ανίχνευση, ανάλυση ή εξουδετέρωση κακόβουλου λογισμικού, ιών, spyware και adware.

- Δοκιμή αλγορίθμων κρυπτογράφησης για αδυναμίες: Έλεγχος του επιπέδου ασφαλείας των δεδομένων που παρέχεται από έναν συγκεκριμένο αλγόριθμο κρυπτογράφησης, αναλύοντάς το για τυχόν αδυναμίες.

- Δοκιμή προστασίας DRM ή αδειών χρήσης: Προστασία του λογισμικού και των μέσων εκτύπωσης ψηφιακών διακαιωμάτων μέσω της εφαρμογής και δοκιμών τεχνικών αντίστοφης μηχανικής.
- Έλεγχος της ασφάλειας των προγραμμάτων: Έλεγχος ενός προγράμματος για την ασφάλεια του από ευπάθειες, χωρίς πρόσβαση στον πηγαίο κώδικα, με την σάρωση ακολουθιών για πιθανές εκμεταλλεύσεις.



Σχήμα 7: «Σενάρια σχετικά με την ανάπτυξη Αντίστροφης Μηχανικής στην Ασφάλεια Λογισμικών.»

ΕΠΙΛΟΓΟΣ

Στο επόμενο κεφάλαιο θα δούμε την έννοια του Test-driven Development, καθώς και μια ιστορική αναδρομή και κάποιες χρήσες του εργαλείου Eclipse.

ΚΕΦΑΛΑΙΟ 3

Test-driven Development και το εργαλείο Eclipse

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο θα αναφερθούμε κυρίως στο TDD, αλλά θα αναφέρουμε και κάποια πράγματα για το εργαλείο Eclipse.

Το Test-driven Development (TDD), είναι μια εξελικτική προσέγγιση για την ανάπτυξη, η οποία συνδυάζει τη μέθοδο test-first development όπου πρώτα γράφει κάποιος το test και μετά αρκετό κώδικα ώστε να εκπληρώσει αυτή το test και το refactoring. Ποιος είναι ο πρωταρχικός στόχος του TDD; Από μια άποψη, στόχος του είναι η προδιαγραφή και όχι η επικύρωση. Με άλλα λόγια, είναι ένας τρόπος να σκεφτεί κανείς τις απαιτήσεις ή το σχεδιασμό πριν ξεκινήσει να γράφει λειτουργικό κώδικα. Μια άλλη άποψη λέει ότι είναι ουσιαστικά μια τεχνική προγραμματισμού. Όπως έλεγε και ο Ron Jeffries, στόχος του TDD είναι να γραφτεί καθαρός κώδικας.

ΥΠΟΚΕΦΑΛΑΙΟ 3.1

Test-driven Development

Το Test-driven Development (TDD), μια από τις καλύτερες πρακτικές ακραίου προγαμματισμού (Extreme Programming XP), υιοθετείται σήμερα ευρέως ως μια ανεξάρτητη πρακτική.

Πρόκειται για μια επαναληπτική και βαθμιαία προσέγγιση στην ανάπτυξη λογισμικού, όπου οι προγραμματιστές γράφουν εκτελέσιμες δοκιμές (περιπτώσεις δοκιμής ή test cases) πριν γράψουν τον κώδικα που δοκιμάζουν [5]. Σε αυτή τη διαδιακασία, το λογισμικό είναι ενσωματωμένο και βελτιώνεται ανάλογα με το χαρακτηριστικό γνώρισμα. Οι προγραμματιστές αναπτύσσουν τα συστήματα μέσω κύκλων δοκιμής, ανάπτυξης και refactoring, κάνουν λεπτομερή σχεδιασμό και σκέφτονται νέες λειτουργίες.

Πιστεύουμε ότι η αξιοποίηση των πρακτικών TDD σε διαδικασία RE και η διαμόρφωση ενός αποτελεσματικού πλαισίου επεξεργασίας, οι αναπτυξιακές ομάδες θα επιτύχουν στις δραστηριότητες της RE, αντιμετωπίζοντας τα εμπόδια που προκύπτουν από τον ήδη εφαρμοσμένο και μη δοκιμασμένο κώδικα.

ΥΠΟΚΕΦΑΛΑΙΟ 3.2

Το εργαλείο Eclipse

Το Eclipse είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) που χρησιμοποιείται στον προγραμματισμό και είναι το πιο διαδεδομένο Java IDE. Περιέχει ένα βασικό χώρο εργασίας (workspace) και ένα επεκτάσιμο σύστημα plug-in για την προσαρμογή του περιβάλλοντος. Το Eclipse είνα γραμμένο κυρίως στην Java και η κύρια χρήση του είναι η ανάπτυξη εφαρμογών Java, αλλά μπορεί επίσης να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών σε άλλες γλώσσες προγραμματισμού μέσω plug-ins, συμπεριλαμβανομένου τις Ada, ABAP, C, C++, COBOL, D, FORTRAN, Haskell, JavaScript, Julia, Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby, Rust, Scala κλπ.

Ο αρχικός κώδικας προέκυψε από την IBM VisualAge. Το κιτ ανάπτυξης λογισμικού Eclipse (SDK), το οποίο περιλαμβάνει τα εργαλεία ανάπτυξης Java, προορίζεται για προγραμματιστές Java. Οι χρήστες μπορούν να επεκτείνουν τις δυνατότητες τους εγκαθιστώντας plug-ins γραμμένα για την πλατφόρμα Eclipse, όπως εργαλεία ανάπτυξης για άλλες γλώσσες προγραμματισμού και μπορούν να γράψουν και να συνεισφέρουν τα δικά τους plug-in.

Το κιτ ανάπτυξης λογισμικού Eclipse (SDK) είναι δωρεάν και είναι λογισμικό ανοιχτού κώδικα, το οποίο κυκλοφορεί σύμφωνα με τους όρους της δημόσιας άδειας Eclipse, αν και δεν είναι συμβατό με την Γενική Άδεια Δημόσιας Χρήσης GNU. Είναι ένας από τους πρώτους IDE που τρέχουν κάτω από το GNU Classpath και τρέχει χωρίς προβλήματα στο IcedTea.

Το Eclipse ήταν εμπνευσμένο από το SmallTalk της VisualAge που ανήκε στην οικογένεια των IDE προϊόντων. Παρόλο που ήταν αρκετά επιτυχής, ένα σημαντικό μειονέκτημα των προϊόντων της VisualAge ήταν ότι ο κώδικας δεν ανήκε σε component-based λογισμικό. Αντίθετα, ο κώδικας για ένα project πραγματοποιούνταν σε ένα συμπιεσμένο κομμάτι (κάπως σαν ένα αρχείο zip αλλά σε ιδιόκτητο σχήμα που ονομάζεται .dat). Η πρόσβαση σε μεμονωμένες κλάσεις ήταν δύσκολη, πόσο μάλλον έξω απο το εργαλείο. Μια όμαδα που ανήκε κυρίως στο εργαστήριο της IBM Cary NC ανέπτυξε ένα νέο προϊόν ως αντικατάσταση με βάση την Java.

Τον Νοέμβριο του 2001, δημιουργήθηκε μια κοινοπραξία με ένα διοικητικό συμβούλιο για την προώθηση της ανάπτυξης του Eclipse σαν λογισμικό ανοιχτού κώδικα. Εκτιμάται ότι η IBM είχε επενδύσει ήδη περίπου 40 εκατομμύρια δολάρια την εποχή εκείνη. Τα αρχικά μέλη ήταν η Borland, η IBM, η Merant, η QNX

Λογισμικά Συστήματα, η Rational Software, η Red Hat, η SuSE, η TogetherSoft, και η WebGain. Ο αριθμός τωμ αγωνοδίκων αυξήθηκε σε πάνω από 80 μέχρι το τέλος του 2003. Τον Ιανουάριο τοου 2004 δημιουργήθηκε το ίδρυμα Eclipse.

ΕΠΙΛΟΓΟΣ

Αφού έχουμε αναφέρει κάποια πράγματα για το Test-driven development και έχοντας κάνει μια ιστορική αναφορά για το εργαλείο Eclipse, στο επόμενο κεφάλαιο γίνεται αναλυτική περιγραφή του πως λειτουργεί και πως δημιουργείται ένα plug-in.

ΚΕΦΑΛΑΙΟ 4

Το περιβάλλον του Plug-in

ΕΙΣΑΓΩΓΗ

Το περιβάλλον ανάπτυξης του Plug-in (Plug-in Development Environment PDE) παρέχει εργαλεία δημιουργίας, ανάπτυξης, δοκιμής, αποσφαλμάτωσης, δημιουργίας και ανάπτυξης πρόσθετων στοιχείων του Eclipse, λειτουργιών, site ενημέρωσης και προϊόντων RCP. Το PDE παρέχει επίσης εκτενή εργαλεία OSGi, τα οποία το καθιστούν ένα ιδανικό περιβάλλον για τον προγραμματισμό διάφορων component, και όχι μόνο την ανάπτυξη ενός plug-in στο Eclipse.

Το PDE είναι χωρισμένο σε τρία βασικά component:

- UI – Ένα πλούσιο σύνολο μοντέλων, εργαλείων και editors για την ανάπτυξη plug-in και πακέτων OSGi.
- API Tools – Εργαλειομηχανή για την υποστήριξη τεκμηρίωσης και συντήρησης API.
- Build – Ant based εργαλεία και script για την αυτοματοποίηση των διαδικασιών κατασκευής.

PDE UI

Το PDE UI παρέχει editors, οδηγούς (γνωστοί ως wizards), launchers, προβολές (views) και άλλα εργαλεία για τη δημιουργία ενός πλήρως λειτουργικού

περιβάλλοντος για την ανάπτυξη πρόσθετων στοιχείων Eclipse, λειτουργιών, site ενημέρωσης, προϊόντων RCP και δεσμών (bundles) OSGi.

Μερικά από αυτά τα εργαλεία περιέχουν:

- **Form-Based Manifest Editors**: Editors πολλαπλών σελίδων που διαχειρίζονται κεντρικά όλα τα αρχεία manifest ενός plug-in ή μιάς λειτουργίας.
- **RCP Tools**: Οδηγοί εγκατάστασης (Wizards) και έναν μορφοποιημένο editor που επιτρέπει τον καθορισμό, το μαρκάρισμα, τη δοκιμή (test) και την εξαγωγή προϊόντων σε πολλαπλές πλατφόρμες.
- **New Project Creation Wizards**: Δημιουργία νέου Plug-in και fragment, νέας λειτουργίας και ενημερωμένης έκδοσης κώδικα και site ενημέρωσης.
- **Import Wizards**: Εισαγωγή ενός plug-in και λειτουργιών από το σύστημα αρχείων.
- **Export Wizards**: Οδηγοί (Wizards) που δημιουργούν, εξάγουν plug-ins, fragments και προϊόντα με ένα κλικ.
- **Launchers**: Έλεγχος και εντοπισμός σφαλμάτων στις εφαρμογές Eclipse και στα OSGi bundles.
- **Views**: Το PDE παρέχει προβολές (Views) που βοηθούν τους προγραμματιστές να επιθεωρούν διάφορες πτυχές του περιβάλλοντος ανάπτυξης τους.
- **Miscellaneous Tools**: Οδηγοί για την εκκαθάριση αρχείων manifest.
- **Conversion Tools**: Οδηγός για τη μετατροπή ενός απλού project Java ή plain JAR σε ένα project plug-in.
- **Integration with JDT**: Τα αρχεία manifest ενός plug-in συμμετέχουν στην αναζήτηση Java και στο refactoring.

API Tools

Τα API Tools βοηθούν στην τεκμηρίωση και την συντήρηση των API που παρέχονται από τα plug-in και τα OSGi bundles.

Μερικά από τα χαρακτηριστικά περιλαμβάνουν:

- **Compatibility Analysis**: Προσδιορισμός προβλημάτων δυαδικής συμβατότητας σε σχέση με προηγούμενες εκδόσεις ενός plug-in.
- **API Restriction Tags**: Οι ετικέτες Javadoc παρέχονται για να καθορίσουν ρητά τους περιορισμούς που σχετίζονται με τους τύπους και τα μέλη.
- **Version Number Validation**: Προσδιορισμός μη έγκυρων αριθμών εκδόσεων plug-in σε σχέση με προηγούμενες εκδόσεις ενός pug-in.
- **Javadoc @since Tag Validation**: Προσδιορισμός ελλιπών και μη έγκυρων @since ετικετών σε τύπους και μέλη.
- **API Leak Analysis**: Προσδιορισμός τύπων και μεθόδων API που διαρρέουν non-API τύπους.
- **Quick Fixes**: Γρήγορες διορθώσεις παρέχονται για την κατάλληλη προσαρμογή των εκδόσεων plug-in και των ετικετών @since.

PDE Build

Το PDE Build διευκολύνει την αυτοματοποίηση των διαδικασιών δημιουργίας Plug-in. Επίσης παράγει script ενεργειών τύπου Ant σύμφωνα με πληροφορίες σχετικά με την ανάπτυξη που παρέχονται, για παράδειγμα, από τα αρχεία plugin.xml και build.properties. Τα παραγώμενα Ant script μπορούν να αντλήσουν τα σχετικά project από ένα CVS repository, build jars, το Javadoc, τα source zip, επίσης μπορούν να βάλουν τα πάντα σε μορφή έτοιμη για αποστολή και να τα στείλει σε μια απομακρυσμένη τοποθεσία (π.χ. τοπικό δίκτυο ή Server λήψης).

ΥΠΟΚΕΦΑΛΑΙΟ 4.1

Δημιουργία ενός Plug-in Project

Για να χρησιμοποιήσει κανείς οποιοδήποτε σημείο επέκτασης Eclipse (Eclipse extension point), συμπεριλαμβανομένων εκείνων που ορίζονται από το Remote System Explorer, πρέπει πρώτα να δημιουργήσει ένα project plug-in χρησιμοποιώντας το περιβάλλον ανάπτυξης plug-in (PDE). Στην απλούστερη περίπτωση, ένα project plug-in απαιτεί ένα αρχείο MANIFEST.MF που περιγράφει το plug-in και τις εξαρτήσεις του κι αν επεκτείνει τον πίνακα εργασίας. Ένα αρχείο plugin.xml που αναγνωρίζει τα σημεία επέκτασης που εφαρμόζονται και ένα σύνολο κλάσεων Java που υλοποιούν αυτά τα σημεία επέκτασης. Συνήθως υπάρχει ένα αρχείο κλάσης plug-in που χρησιμοποιείται ως ο συνολικός

διαχειριστής του project και σημείο ολοκλήρωσης που μπορούν να βασιστούν σε άλλες κατηγορίες.

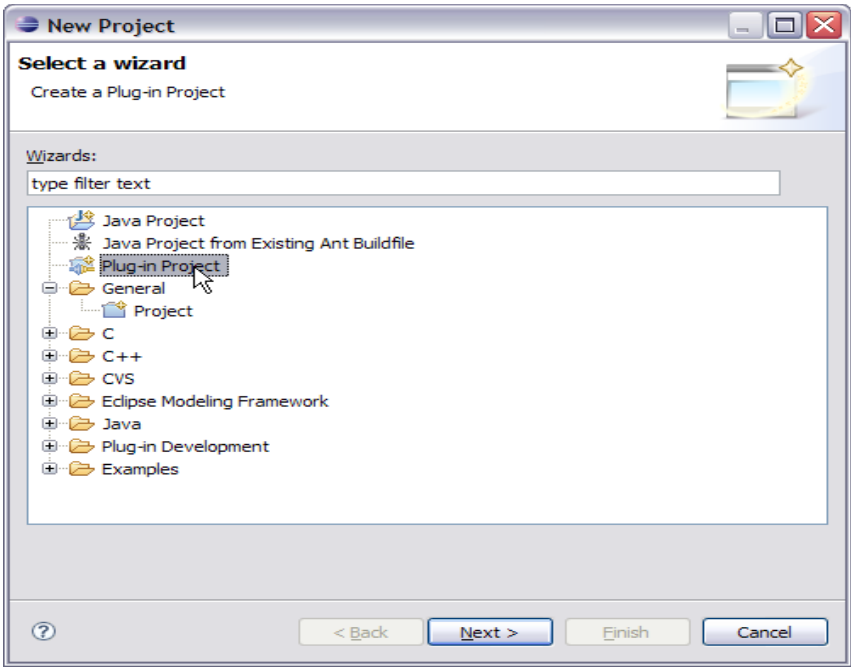
Αν το project plug-in υπάρχει ήδη, θα χρειαστεί να το ενημερώσει κανείς ελαφρώς για να κάνει την επέκταση του SystemBasePlugin και να προσθέσει τις λίγες μεθόδους που απαιτεί. Το περιβάλλον Eclipse θα κρατήσει συγχρονισμένα τις συνδέσεις του classpath και του plug-in.

Το Eclipse παρέχει ένα πλήθος από πρότυπα project plug-in, τα οποία παράγουν έναν αριθμό αρχείων του project που απεικονίζουν παραδείγματα διάφορων σημείων επέκτασης Eclipse (Eclipse extension points). Ενώ είναι ελεύθερος κανείς να διαλέξει ένα από αυτά τα project ή να αρχίσει με κάποιο υπάρχον πρόγραμμα plug-in, αν έχει ένα, στα RSE tutorials τα πάντα δημιουργούνται με το χέρι, έτσι ώστε να διατηρούνται εστιασμένα στις κλάσεις και τα αρχεία που απαιτούνται από το RSE.

Βήμα προς βήμα: Η δημιουργία ενός Plug-in Project

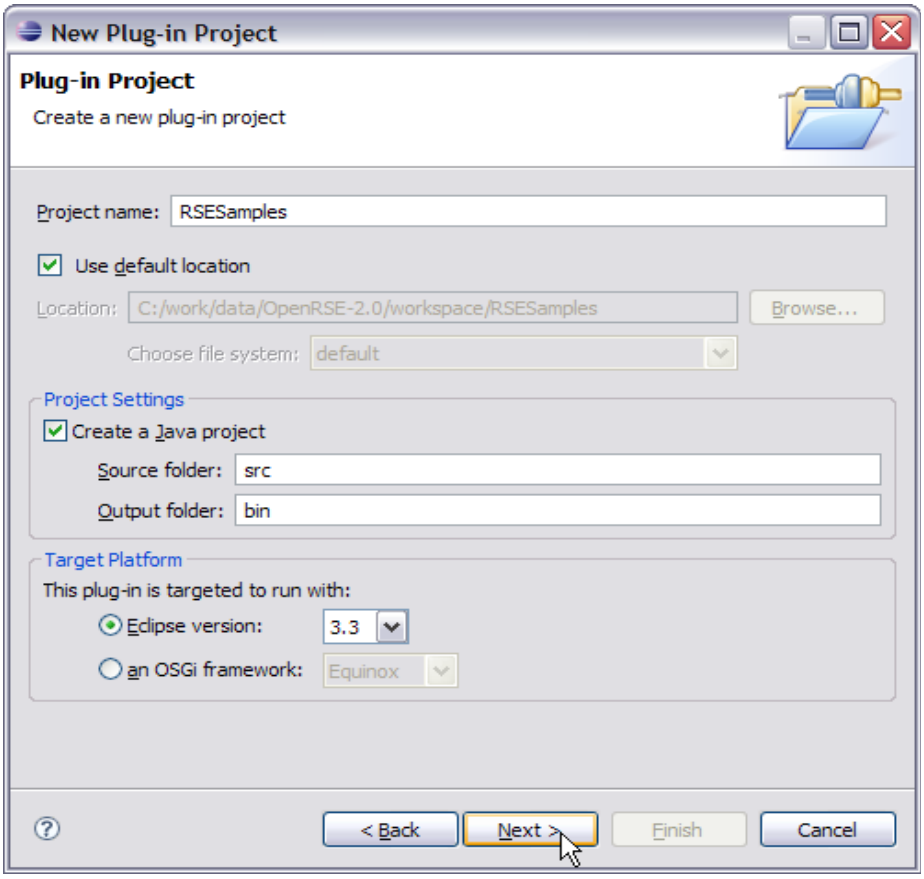
1. Επιλέξτε **File->New->Project**.

2. Στο dialog box που θα εμφανιστεί επιλέξτε τον Plug-in Project οδηγό (wizard). Πατήστε **Next >**.



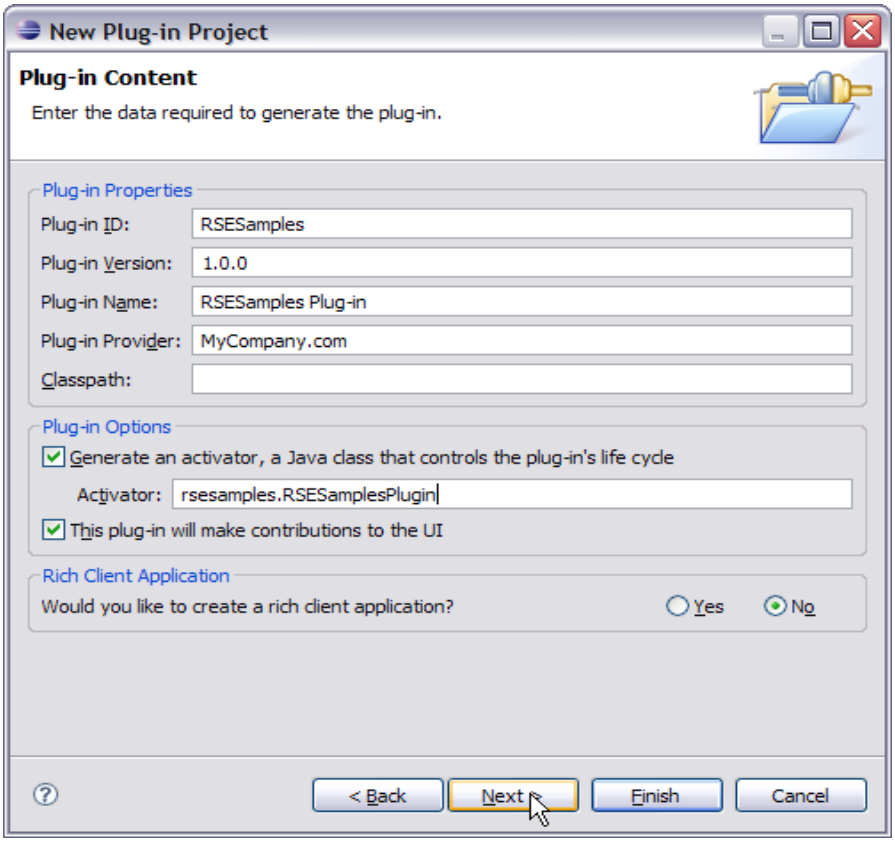
Σχήμα 8: «Επιλογή Plug-in Project στο Wizard»

3. Στην πρώτη σελίδα του οδηγού εισάγετε το όνομα του project. Πατήστε **Next >**.



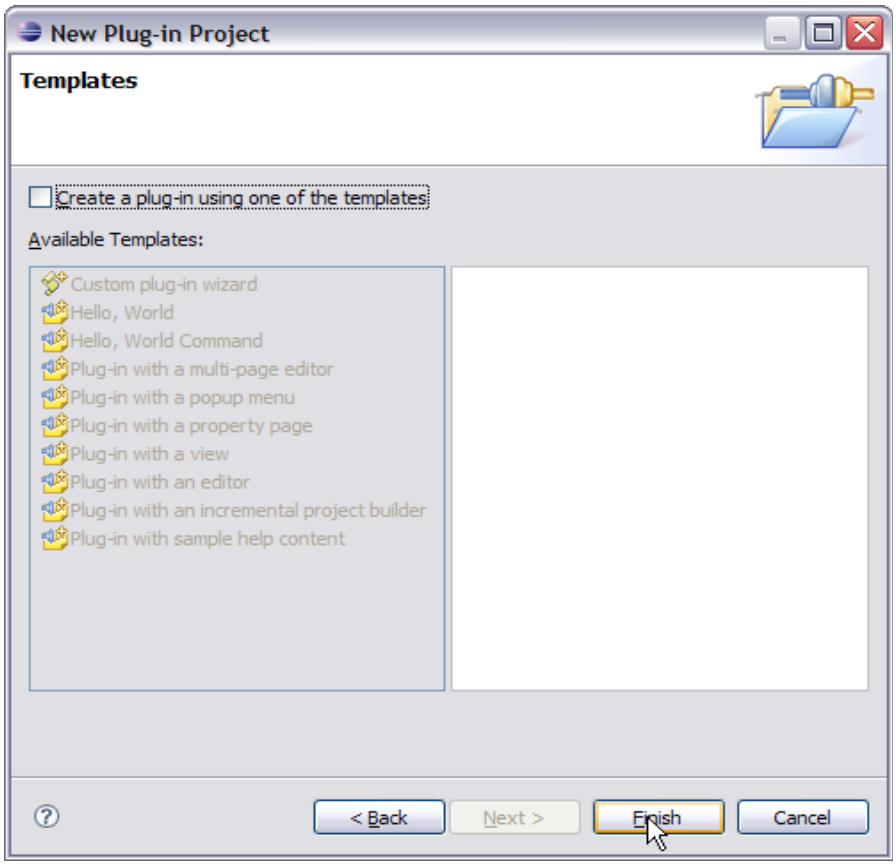
Σχήμα 9: «Εισαγωγή ονόματος στο νέο project.»

4. Στη δεύτερη σελίδα του οδηγού, συμπληρώστε στον **Activator** το όνομα που θέλετε όπως φαίνεται στο παράδειγμα.



Σχήμα 10: «Εισαγωγή ονόματος στον Activator.»

5. Στην τρίτη σελίδα του οδηγού αποεπιλέξτε το checkbox “Create a plug-in using one of the templates” και πατήστε **Finish**.



Σχήμα 11: «Τέλος του Οδηγού New Plug-in Project.»

6. Το νέο plug-in project δημιουργήθηκε και εμφανίζεται στον Package Explorer του Plug-in Development perspective. Οι νέες ιδιότητες του plug-in είναι επίσης ανοιχτές στον plug-in editor.

7. Μεταβείτε στην καρτέλα των εξαρτήσεων (dependencies) του προγράμματος περιήγησης plug-in και προσθέστε τα ακόλουθα πρόσθετα στη λίστα:

- org.eclipse.core.resources

- org.eclipse.rse.ui
- org.eclipse.rse.services
- org.eclipse.rse.files.ui
- org.eclipse.rse.shells.ui
- org.eclipse.rse.subsystems.files.core
- org.eclipse.rse.subsystems.shells.core

8. Τώρα πηγαίνετε στην καρτέλα MANIFEST.MF των ιδιοτήτων του plug-in. Αυτό δείχνει την προέλευση του αρχείου MNIFEST.MF που σχετίζεται με αυτό το plug-in. Αλλάξτε τη γραμμή Bundle-SymbolicName προσθέτοντας ένα singleton: = true directive.

```
Bundle-SymbolicName:RSESamples;singleton:=true
```

Αυτό μας επιτρέπει να προσθέσουμε επεκτάσεις στο plug-in σε μεταγενέστερο σημείο. Αποθηκεύστε τις ιδιότητες του plug-in και κλείστε τον editor.

9. Δεξί κλικ στο project και δημιουργήστε ένα αρχείο plugin.xml. Κανονικά, αυτό θα δημιουργηθεί εάν χρησιμοποιήσατε ένα template για να δημιουργήσετε το plug-in σας. Θα χρησιμοποιήσουμε αυτό το αρχείο για να προσθέσουμε επεκτάσεις στο RSE, αλλά προς τι παρόν θα είναι απλά ένας σκελετός με τα ακόλουθα περιεχόμενα:

```
<plugin>

</plugin>
```

Προσθέστε τις παραπάνω γραμμές στο κενό αρχείο plugin.xml και αποθηκεύστε το.

10. Επεκτείνετε το φάκελο **src**, στη συνέχεια το φάκελο πακέτων με το όνομα που έχετε δώσει στο project και κάντε διπλό κλικ στο **.java** αρχείο, το οποίο έχει επίσης το όνομα που έχετε δώσει στο project, για να επεξεργαστούμε αυτή την κλάση. Αλλάξτε όπως περιγράφεται παρακάτω:

- Επεκτείνετε το SystemBasePlugin αντί του AbstractUIPlugin
- Προσθέστε τη δήλωση για το resourceBundle
- Προσθέστε τη δήλωση για το messageFile
- Επικαλέστε (invoke) τον constructor της υπερκλάσης από αυτόν τον constructor
- Προσθέστε τη μέθοδο initializeImageRegistry()
- Προσθέστε τη μέθοδο getMessageFile()
- Προσθέστε τη μέθοδο getResourceBundle()
- Προσθέστε τη στατική μέθοδο getWorkspace()
- Προσθέστε τη στατική μέθοδο getResourceString(String)
- Προσθέστε τη στατική μέθοδο getPluginMessageFile()
- Προσθέστε τη στατική μέθοδο getPluginMessage(String)

11. Δημιουργήστε το resource αρχείο του project για translatable strings: δεξί κλικ στο project και επιλέξτε **New->File** για να ανοίξετε τον οδηγό **New File**. Βάλτε για όνομα αρχείου το εξής: **“όνομα project”Resources.properties**, όπως καθορίστηκε στην κλήση της loadResourceBundle στον constructor της κλάσης του plug-in. Πατήστε **Finish** για τη δημιουργία του αρχείου.

12. Δημιουργήστε τα μηνύματα του RSE-style του project για translatable μηνύματα: δεξί κλικ στο project και επιλέξτε **New->File** για να ανοίξετε τον οδηγό **New File**. Βάλτε για όνομα αρχείου το εξής: **“όνομα project”Messages.xml**, όπως καθορίστηκε στην κλήση της loadMessageFile στον constructor της κλάσης του plug-in. Πατήστε **Finish** για τη δημιουργία του αρχείου. Θα δείτε ανοιχτό τον XML editor για το νέο αρχείο. Πατήστε την καρτέλα **Source** στο τέλος του editor, και προσθέστε τις παρακάτω γραμμές κώδικα:

```
<?xml version="1.0" encoding='UTF-8'?>

<!DOCTYPE MessageFile SYSTEM "../org.eclipse.rse.ui/messageFile.dtd">
<!-- This is a message file used by SystemMessage and SystemMessageDialog
-->
<MessageFile Version="1.0">
	<Component Name="RSE Samples" Abbr="RSS">
		<Subcomponent Name="General" Abbr="G">
			<MessageList>
				<Message ID="1001" Indicator="E">
					<LevelOne>Sample message</LevelOne>
					<LevelTwo>This is a sample with one
					substitution variable: %1</LevelTwo>
				</Message>
			</MessageList>
		</Subcomponent>
	</Component>
</MessageFile>
```

Αποθηκεύστε και κλείστε το αρχείο.

13. Το plug-in έχει δημιουργηθεί και είναι έτοιμο. Το μόνο που χρειάζεται τώρα είναι να προσθέσετε τον κώδικα για την υλοποίηση των extension points.

ΥΠΟΚΕΦΑΛΑΙΟ 4.2

Eclipse Application Launcher

Το PDE παρέχει ένα πρόγραμμα εκτέλεσης εφαρμογών Eclipse (Eclipse Application Launcher) το οποίο επιτρέπει την εκτέλεση και την διόρθωση (debug) του plug-in που έχει δημιουργηθεί, ξεκινώντας μια ξεχωριστή εφαρμογή Eclipse. Όπως συμβαίνει με όλους τους άλλους launchers στο Eclipse (για παράδειγμα Java Application και Java Applet launchers κτλπ), ο Application Launcher του Eclipse μπορεί να χρησιμοποιηθεί μέσω συντόμευσης και η εκτέλεση του διαχειρίζεται κεντρικά απο τον **Launch Configuration Dialog**.

Εκτέλεση μέσω συντόμευσης

Ένας γρήγορος τρόπος για τη δοκιμή ενός plug-in είναι ξεκινώντας μια ξεχωριστή εφαρμογή μέσω της συντόμευσης εκτέλεσης της εφαρμογής Eclipse, που είναι διαθέσιμη ακολουθώντας τα παρακάτω βήματα:

- τα hot links στην ενότητα **Testing** της σελίδας **Overview** του προγράμματος προβολής του plug-in (plug-in manifest editor)

- ενέργειες στο μενού περιεχομένων των plug-in project στις επιλογές **Run As >** και **Debug As >**

Οι συντομεύσεις εκκίνησης (Launch shortcuts) είναι context-sensitive. Εάν, για παράδειγμα, ο επιλεγμένος πόρος είναι ένα plug-in project ή ένας manifest editor που δηλώνει μια εφαρμογή (μία επέκταση του τύπου org.eclipse.core.runtime.applications.extension) ή ένα προϊόν (μία επέκταση του τύπου org.eclipse.core.runtime.products), τότε το PDE εκκινεί αυτή την εφαρμογή ή το προϊόν. Το σύνολο των PDE plug-in που ξεκινούν με αυτό είναι ελάχιστο και αποτελείται από το επιλεξμένο plug-in και όλες τις προϋποθέσεις που συνεπάγουν με αυτό. Εάν το επιλεγμένο plug-in περιέχει τόσο μια επέκταση προϊόντος όσο και μια επέκταση εφαρμογής, η επέκταση του προϊόντος έχει προτεραιότητα.

Εάν το επιλεγμένο plug-in δεν περιέχει ούτε επέκταση προϊόντος ούτε επέκταση εφαρμογής, το PDE εκκινεί το προεπιλεγμένο προϊόν όπως ορίζεται στο κλειδί eclipse.product του αρχείου congi.ini που βρίσκεται στον κατάλογο ${target_home}/configuration. Το ${target_home} αναφέρεται στη θέση της πλατφόρμας στόχου όπως καθορίζεται στη σελίδα προτιμήσεων του **Windows > Preferences... > Plug-in Development > Target Platform**. Εάν ξεκινήσει το προεπιλεγμένο προϊόν, το σύνολο των plug-in που χρησιμοποιούνται στην εκκίνηση είναι όλα plug-in χώρου εργασίας (workspace) και όλα τα plug-in ελέγχονται στην σελίδα προτιμήσεων της πλατφόρμας στόχευσης. Τα plug-in στόχοι, των οποίων το ID έρχεται σε σύγκρουση με το ID ενός plug-in χώρου εργασίας, δεν περιλαμβάνεται στο configuration της εκκίνησης.

Το PDE δημιουργεί μια νέα διαμόρφωση εκκίνησης της εφαρμογής Eclipse και τη ρυθμίζει με λογικές προεπιλογές. Αυτή η διαδικασία γίνεται μόνο όταν το απαιτεί η περίσταση και όχι κάθε φορά που γίνεται κλήση της εκκίνησης της εφαρμογής Eclipse. Το PDE αναζητά πρώτα τις υπάρχουσες διαμορφώσεις εκκίνησης εφαρμογών Eclipse και επαναχρησιμοποιεί μία που έχει ήδη συνδεθεί με το προϊόν ή την εφαρμογή που έχει ξεκινήσει. Στην περίπτωση που ύπαρχουν πολλές υπάρχουσες διαμορφώσεις εκκίνησης που σχετίζονται με το προϊόν ή την εφαρμογλη που εκτελείται, το PDE εμφανίζει ένα παράθυρο διαλόγου που περιέχει όλες τις αντίστοιχες ρυθμίσεις εκκίνησης και επιτρέπει στον χρήστη να αποφασίσει.

Προσαρμογή ενός Launch Configuration

Στην περίπτωση που χρειαστεί κανείς πλήρη έλεγχο του τρόπου εκκίνησης της εφαρμογής Eclipse, μπορεί να δημιουργήσει και προσαρμώσει μια διαμόρφωση

εκκίνησης (launch configuration) στο παράθυρο διαλόγου **Launch Configuration Dialog**.

Το παράθυρο διαλόγου Launch Configuration Dialog μπορεί να επικαλεστεί μεσω των επιλογών **Run > Run...** ή **Debug > Debug...** από το επάνω μενού. Ένα νέο *Eclipse Application* launch configuration μπορεί να δημιουργηθεί κάνοντας διπλό κλικ στον κόμβο **Eclipse Application** στο πρόγραμμα προβολής δέντρων στα αριστερά.

ΕΠΙΛΟΓΟΣ

Αφού έχουμε δει σε μικρό βαθμό το πως λειτουργεί το plug-in και πως δημιουργούμε ένα, ας δούμε στο επόμενο κεφάλαιο πως λειτουργεί και το εργαλείο GitHub.

ΚΕΦΑΛΑΙΟ 5

Το εργαλείο GitHub

ΕΙΣΑΓΩΓΗ

Η χρήση του εργαλείου GitHub για την εκπόνηση της πτυχιακής μας εργασίας έπαιξε σημαντικό ρόλο διότι μας βοήθησε να έχουμε τον κώδικα μας ανεβασμένο σε ένα σημείο το οποίο ήταν εύκολα προσβάσιμο και για τους δυο από οποιοδήποτε σημείο και ταυτόχρονα μας έδινε τη δυνατότητα να κάνουμε αλλαγές στον κώδικα και να τις μοιραζόμαστε μεταξύ μας χωρίς να επηρεάζεται το σύνολο του κώδικα που είχαμε γράψει.

Πιο συγκεκριμένα, το GitHub είναι ένα VCS (version control system) το οποίο σου δίνει τη δυνατότητα να ελέγχεις τις εκδόσεις του κώδικα που γράφεις. Δίνει τη δυνατότητα στον προγραμματιστή να μπορεί να έχει πολλές εκδοχές του κώδικά του κρατώντας μια σταθερή έκδοση (Master). Δημιουργώντας πολλά κλαδιά (Branches) δίνεται η δυνατότητα στον προγραμματιστή ή στην ομάδα να μπορεί να αναπτύξει παραπάνω από μια εκδόσεις έχοντας τον ίδιο αρχικό κώδικα.

ΥΠΟΚΕΦΑΛΑΙΟ 5.1

Περιγραφή του εργαλείου

Ξεκινώντας, ο χρήστης δημιουργεί το repository στο οποίο μέσα θα στεγαστεί ο κώδικας που θα αναπτύσσει η ομάδα ή ο προγραμματιστής. Μαζί με τη δημιουργία του repository ο χρήστης μπορεί να επιλέξει αν αυτό θα είναι δημόσιο, δηλαδή θα μπορεί να το βλέπει όποιος επισκέπτεται το προφίλ του, ή να είναι

ιδιωτικό και να έχουν πρόσβαση μόνο όσοι έχουν οριστεί από τον χρήστη ως συνεργάτες (collaborators) του. Μπορεί επίσης να προσθέσει και ένα αρχείο που ονομάζεται gitignore το οποίο αφήνει έξω από το repository οτιδήποτε ορίσουν οι χρήστες που δεν θα ήθελαν να υπάρχει εκεί. Το αρχείο αυτό αγνοεί τα αρχεία τα οποία ορίζουν οι χρήστες τόσο όταν ψάχνει τις διαφορές μεταξύ των αρχείων στο master ή σε κάποιο άλλο branch όσο και κατά την στιγμή που κάποιος από την ομάδα ή ο προγραμματιστής κάνει commit τον κώδικα του.

Μετά την δημιουργία του repostiroty, η ομάδα ή ο προγραμματιστής θα πρέπει να κάνει το πρώτο του commit δηλαδή να ανεβάσει τον κώδικα που θα στεγάσει το repository του. Μπορεί να κάνει το πρώτο commit στο master branch και να έχει τη βάση του κώδικα πάνω στην οποία θα μπορεί να φτιάξει πολλές άλλες εκδόσεις ή να φτιάξει ένα branch με το οποίο θα έχει την έκδοση του κώδικα που επιθυμεί πριν την κάνει merge με το master branch.

Ο χρήστης ανεβάζει κατά την δική του κρίση τον κώδικα του στο master ή σε κάποιο άλλο branch και εν συνεχεία ξεκινάει να γράφει κώδικα για να επεκτείνει τις λειτουργίες που περιέχει το λογισμικό του ή να φτιάξει νέες. Όταν αποφασίσει πως είναι ώρα να κάνει commit το κομμάτι κώδικα που έχει γράψει μπορεί πάλι να το κάνει με δύο τρόπους όπως έκανε και στο πρώτο commit του. Αν αποφασίσει να κάνει commit στο master τότε ο κώδικας στο τοπικό του repository και αυτού στο GitHub θα ενωθούν με τη διαδικασία του merge και θα προκύψει το τελικό αποτέλεσμα αφαιρώντας και προσθέτοντας γραμμές. Ο τρόπος αυτός εμπεριέχει πολλούς κινδύνους και υπάρχει περίπτωση να χαθεί κώδικας, καθώς η απευθείας ένωση με το master branch θα προκαλέσει αλλαγές στη βάση του κώδικα. Ακόμα, υπάρχει περίπτωση κάποιος άλλος χρήστης κάνοντας sync από το master branch να χάσει τη δουλεία του ή να δημιουργηθούν conflicts μεταξύ των αρχείων που ανέβασε ο προηγούμενος χρήστης και αυτών που θέλει να ανεβάσει ο επόμενος χρήστης, οποίος δεν είδε ότι τα αρχεία αυτά άλλαξαν και ότι θα πρέπει να αποθηκεύσει τις αλλαγές του, έπειτα να κάνει sync και μετά να τις ξαναπροσθέσει μέσα στον κώδικα.

Ο δεύτερος τρόπος, που είναι πιο ασφαλής και δεν έχει σχεδόν κανέναν κίνδυνο, είναι ο χρήστης να δημιουργήσει ένα νέο branch να κάνει όσα commit επιθυμεί σε αυτό ανάλογα με το πόσες αλλαγές θέλει να κάνει να ανοίξει ένα Pull Request. Κατά την περίοδο που το Pull Request είναι ανοιχτό μπορούν να γίνονται κανονικά commits στο branch Και αυτά εμφανίζονται στη λίστα με τα αρχεία που θα γίνουν merge με αυτό το request. Οπότε οι χρήστες μπορούν να κάνουν αλλαγές μέχρι να είναι έτοιμες οι λειτουργίες ή αλλαγές που θέλουν να κάνουν στον κώδικα.

Στη συνέχεια ο κώδικας μπορεί να περάσει από κάποιο review ή συζήτηση με την ομάδα σχετικά με τις λειτουργίες που έχει προσθέσει ή έχει αλλάξει ή και τα δυο και αφού δεν υπάρχουν προβλήματα μπορεί να κάνει merge το branch αυτό στο master branch χωρίς να υπάρχει περίπτωση να χαλάσει τη δουλεία κάποιου άλλου. Μπορεί να γίνει merge μεταξύ δυο branch και το ένα να μην είναι το master.

ΥΠΟΚΕΦΑΛΑΙΟ 5.2

Επιπλέον δυνατότητες

Το GitHub δεν προσφέρει μόνο αυτές τις λειτουργίες καθώς έχει πολλές περισσότερες και είναι ένα εργαλείο με το οποίο μπορεί να αναπτυχθεί ένα λογισμικό μαζί με τις διάφορες εκδόσεις που μπορεί να έχει.

Επίσης η πλατφόρμα του GitHub προσφέρει τη δυνατότητα σε χρήστες που δεν είναι συνεργάτες στο repository και αφού αυτό είναι δημόσιο να μπορούν να το κάνουν clone τοπικά στον υπολογιστή τους και αφού κάνουν κάποιες αλλαγές τροποποιήσεις να μπορούν να ανοίγουν ένα branch και μέσω αυτού να προτείνουν τις αλλαγές ή τις διορθώσεις τους.

Όταν ένας χρήστης επιθυμεί να κατεβάσει ένα repository τοπικά στον υπολογιστή του προκειμένου να εργαστεί πάνω στον κώδικα που περιέχει αυτό τότε το κάνει clone. Με αυτή την λειτουργία ουσιαστικά γίνεται κατέβασμα όλου του repository τοπικά και αυτό συνδέεται με τον αντίστοιχο που υπάρχει στον GitHub server. Έπειτα, ο χρήστης αν επιθυμεί μπορεί να κάνει sync σε κάποιο από τα υπάρχοντα branch ή να δημιουργήσει το δικό του και μετά να το ανεβάσει και να παρουσιάσει έτσι τη δουλεία του.

Κατέβασμα του κώδικα μπορεί να γίνει και με άλλο τρόπο οποίος είναι να γίνει όλο το repository είναι .zip αρχείο και ο χρήστης να το κάνει λήψη κανονικά στον υπολογιστή του αλλά αυτό δεν του δίνει τη δυνατότητα της σύνδεσης του τοπικού repository με αυτό που είναι στον GitHub server.

ΕΠΙΛΟΓΟΣ

Το Github ως εργαλείο αποτελεί ένα πολύ καλό μέσο για να μπορεί ο προγραμματιστής ή ομάδα να ελέγχει τον κώδικα που γράφει, να είναι εύκολα προσβάσιμος, να μπορούν να κάνουν αλλαγές χωρίς να επηρεάζουν ο ένας τον άλλον, να έχουν πολλές εκδοχές του ίδιου κώδικα που θα αποτελούν διαφορετικές εκδόσεις και να έχουν όλοι την ίδια βάση και αρχή. Αφού έχουμε δει πως λειτουρεί

και το GitHub, στο επόμενο κεφάλαιο γίνεται αναλυτική περιγραφή του TDRE framework.

ΚΕΦΑΛΑΙΟ 6

Ανάλυση του framework TDRE

ΕΙΣΑΓΩΓΗ

Η έλλειψη μιας καθιερωμένης διαδικασίας RE οδήγησε στην συγγραφή του paper και στο να προτείνουν [1] το πλαίσιο (framework) διαδικασιών TDRE που συνδυάζει τις πρακτικές TDD και τις δραστηριότητες RE. Το framework αυτό προτείνει την ιεραρχική δοκιμή μονάδων προτεραιότητας των ήδη υλοποιημένων λειτουργιών, οι οποίες έχουν επίσης προτεραιότητα μέσω μιας τεχνικής ιεράρχησης προτεραιοτήτων και χρησιμοποιώντας συγκεκριμένους παράγοντες προτεραιότητας.

Το προτεινόμενο πλαίσιο επεξεργασίας, σε τέσσερις ξεχωριστές φάσεις, βοηθά τη διαδικασία RE, από τη φάση ανάλυσης μέχρι την ανάκτηση και την εξαγωγή των διαγραμμάτων UML. Για να αξιολογηθεί εμπειρικά η χρησιμότητα και η αποτελεσματικότητα του προτεινόμενου framework διαδικασιών TDRE και η χρήση της πρακτικής TDD στο περιβάλλον RE, έγινε η διεξαγωγή μιας μελέτης περίπτωσης (case study) [1] στον ακαδημαϊκό κόσμο.

Κύριο ερώτημα στην μελέτη αυτή είναι: *'Βοηθά η πρακτική TDD την RE στην κατανόηση του κώδικα και του σχεδιασμού και εν τέλη στη διαδικασία επανασχεδιασμού;*' και ακόμα πιο συγκεκριμένα '*Μήπως η κατά προτεραιότητα δοκιμή μονάδας των εφαρμοζόμενων λειτουργιών βελτιώνει την κατανόηση του κώδικα και του σχεδιασμού στη διαδικασία RE;*'.

ΥΠΟΚΕΦΑΛΑΙΟ 6.1

Φάσεις του framework διαδικασίας TDRE

Α. Φάση Ι: Ανάλυση συστήματος, κατανόηση κώδικα και αποσύνθεση

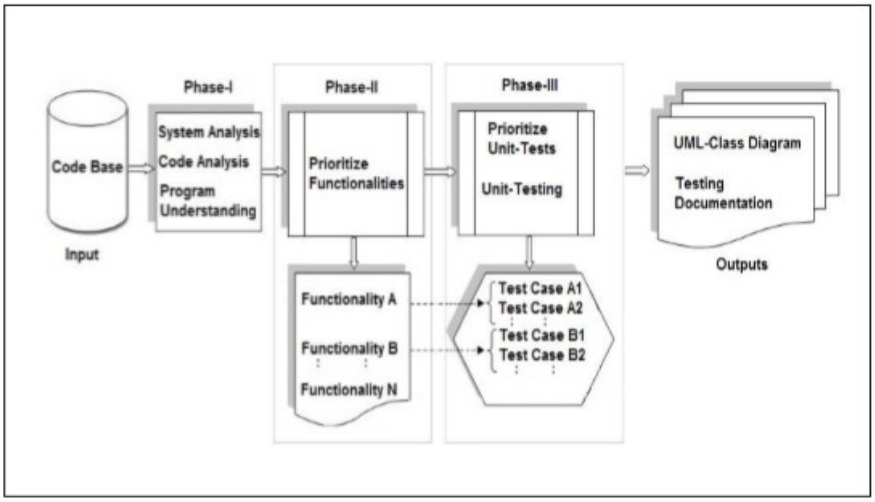
Σε αυτή την αρχική φάση εντοπίζονται, αξιολογούνται και προσδιορίζονται τα πιο σημαντικά χαρακτηριστικά του συστήματος (Σχήμα 3). Η φλάση περιλαμβάνει τον προσδιορισμό, την οργάνωση, και την αναπαράσταση της δομής, της σύνθεσης και των αλληλεπιδράσεων των στοιχείων στον τομέα. Οι περισσότερες από τις δραστηριότητες, σε αυτή τη φάση, υλοποιούνται παράλληλα. Οι δραστηριότητες ανάλυσης του συστήματος και τα δημιουργημένα αντικείμενα περιλαμβάνουν:

1) Αναγνώριση πακέτων, κλάσεων, μεθόδων, κατασκευαστές (constructors) για ΟΟ (Object Oriented) κώδικα (αρχική έρευνα του κώδικα, δημιουργία λιστών χαρτογράφησης).

2) Προσδιορισμός των αλληλεπιδράσεων και της ροής εργασίας (δημιουργία γραφηματών ροής κ.λπ.).

3) Προδιαγραφές εισόδων και εξόδων του συστήματος (δημιουργία των αντίστοιχων αντικειμένων).

4) Δημιουργία λιστών εφαρμογής λειτουργικότητας (απλές λίστες περιγραφής για τις ήδη εφαρμοζόμενες λειτουργίες – λίστες μεθόδων/λειτουργιών).



Σχήμα 12: «Η διαδικασία του framework TDRE.»

Β. Φάση ΙΙ: Κατανόηση του προγράμματος - Προτεραιότητα των λειτουργιών

Η κατανόηση του προγράμματος είναι μια αντικειμενική και όχι μια σαφώς καθορισμένη διαδικασία [6], αντιστρέφοντας τη διαδικασία αντιστοίχησης τομέα με την ανάκτηση των χαμένων πληροφοριών και την ρητή εμφάνιση των σιωπηρών πληροφοριών [7]. Δυο κοινές προσεγγίσεις για την κατανόηση του κώδικα είναι η λειτουργική προσέγγιση (bottom-up) η οποία δίνει έμφαση στην γνώση από το τι κάνει το σύστημα και στη συμπεριφορική προσέγγιση (top-down) η οποία δίνει έμφαση στο πώς λειτουργεί το σύστημα [7].

Η προσέγγιση “bottom-up” ανασυντάσσει τον σχεδιασμό υψηλού επιπέδου ενός συστήματος, ξεκινώντας με στοιχεία βάσης του πηγαίου κώδικα που συνδέονται μεταξύ τους για να σχηματίσουν μεγαλύτερα υποσυστήματα και η διαδικασία αυτή επαναλαμβάνεται σε πολλά επίπεδα μέχρι να σχηματιστεί ένα πλήρες σύστημα ανώτατου επιπέδου. Η προσέγγιση “bottom-up”, γνωστή και ως προσέγγισης σύνθεσης, ταιριάζει καλύτερα στον Object Oriented κώδικα.

Η προσέγγιση “top-down” είναι ουσιαστικά η διάσπαση ενός συστήματος για να αποκτήσει κανείς γνώσεις για τα σύνθετα υποσυστήματα του. Η προσέγγιση αυτή, γνωστή και ως προσέγγιση αποσύνθεσης, ταιριάζει καλύτερα με τον διαδικασιακό κώδικα.

Οι testers και οι μηχανικοί συντήρησης χρησιμοποιούν και τις δύο προσεγγίσει για να κατανοήσουν και να ορίσουν το σύστημα αντίστροφης μηχανικής (ευκαιριακή προσέγγιση) [7]. Ανεξαρτήτως της προσέγγισης που χρησιμοποιείται ακόμα και αν χρησιμοποιούνται και οι δύο, δημιουργούμε τον κατάλογο προτεραιοτήτων των εφαρμοζόμενων λειτουργιών λαμβάνοντας υπόψη ορισμένα σημαντικά χαρακτηριστικά (αναφορά στο επόμενο υποκεφάλαιο).

Γ. Φάση III: Διαχωρισμός κώδικα και ιεράρχηση Uni-Testing

Οι προϋποθέσεις του προτεινόμενου framework είναι η έλλειψη μονάδων δοκιμών (unit tests) και η έλλειψη άλλων δεδομένων ιστορικού ανάπτυξης. Έτσι, η βάση κώδικα είναι ο μόνος διαθέσιμος πόρος και αυτή είναι η προεπιλεγμένη κατάσταση στα περισσότερα περιβάλλοντα RE.

Λαμβάνοντας υπόψη αυτές τις προϋποθέσεις και βοηθώντας με την ήδη ταυτοποίηση των εξαρτημάτων λογισμικού στη Φάση I, εφαρμόζουμε την διάσπαση του κώδικα [Weiser, 1984] όπου χρειάζεται, για τη γραφή unit testing σύμφωνα με τον κατάλογο προτεραιοτήτων λειτουργιών. Η ομαδοποίηση των test cases και των σουιτών (suites), βάση αυτού του καταλόγου, χαρτογραφεί ακριβώς τα unit tests στις εφαρμοζόμενες λειτουργίες σύμφωνα με την ιεράρχηση τους. Η περίπτωση αυτή εφαρμόζεται και στο forward-engineering, όπου οι ιστορίες των χρηστών και τα unit tests υλοποιούνται σύμφωνα με την προτεραιότητα τους ως προς τη σημασία τους.

Σε μεγάλες βάσεις κώδικα με κλιμακούμενες λειτουργίες, μπορούν να εφαρμοστούν άμεσα προκαθορισμένες περιπτώσεις δοκιμών (test cases) και σουίτες και η διαδικασία αυτή επαναλαμβάνεται μέχρι το τέλος της λίστας προτεραιοτήτων. Σε αυτές τις περιπτώσεις, εφαρμόζονται ταυτόχρονα η Φάση III και η Φάση IV.

Δ. Φάση IV: Εφαρμογή Unit-Testing και εκπροσώπηση σχεδιασμού

Σε αυτή τη φάση, υλοποιείται η διαδικασία TD για τις λειτουργίες προτεραιότητας. Η διαδικασία του testing ξεκινά με τα πρώτα test cases της λίστας και συνεχίζει επαναληπτικά και σταδιακά μέχρι το τέλος της λίστας προτεραιότητας. Μέσα από αυτή την ιεραρχική διαδικασία TD, οι σημαντικότερες λειτουργίες του συστήματος ελέγχονται από την αρχή.

Τα Unit-Tests διευκρινίζουν τον κρυφό σχεδιασμό και επικυρώνουν το εύρος, τη συμπεριφορά και τα χαρακτηριστικά όλων των λειτουργιών, συμβάλλοντας στην κατανόηση του κώδικακαι επαναδραστηριοποίηση, ειδικά σε περιπτώσεις όπου οι εφαρμοζόμενες λειτουργίες είναι μεγάλες και σύθετες.

Στο τέλος της διαδικασίας, τα Unit-Tests περιλαμβάνουν αποθετήριο συσσωρευμένων πληροφοριών για το σχεδιασμό κώδικα, βοηθώντας του μηχανικούς λογισμικού να επανασχεδιάσουν τα διαγράμματα UML.

Δομή και παρουσίαση: Τα αντικείμενα που δημιουργήθηκαν απεικονίζουν το γενικό υποκείμενο μοντέλο τομέα. Το μοντέλο τομέα είναι μια εννοιολογική “παράσταση που συλλαμβάνει τη δομή και τη σύνθεση των αντικειμένων σε μια περιοχή προβλήματος” [8]. Στο προτεινόμενο μοντέλο [1] ένα διάγραμμα κλάσης UML χρησιμοποιείται για να αντιπροσωπεύει αυτό το μοντέλο τομέα.

ΥΠΟΚΕΦΑΛΑΙΟ 6.2

Προτεραιότητα στις λειτουργίες

Τεχνικές ιεράρχησης έχουν χρησιμοποιηθεί σε διαφορετικές περίπλοκες καταστάσεις λήψης αποφάσεων στον τομέα της μηχανικής λογισμικού, όπως οι απαιτήσεις [9], [10], και το Unit-Testing [11], [12]. Η ιεράρχηση αποδίδει τιμές σε αντικείμενα που έχουν συγκριθεί μεταξύ τους για την καθιέρωση μιας σχετικής σειράς μεταξύ τους. Κάθε τεχνική ιεράρχησης περιλαμβάνει διαφορετικές πτυχές που πρέπει να ληφθούν υπόψη.

Μια πτυχή είναι μια ιδιότητα ή χαρακτηριστικό του project που μπορεί να χρησιμοποιηθεί για τη δραστηριότητα ιεράρχησης προτεραιοτήτων. Διαφορετικές κλίμακες μέτρησης μπορούν να χρησιμοποιηθούν για τη μέτρηση αυτών των πτυχών, όπως οι κανονικές κλίμακες και οι κλίμακες αναλογίας. Η κανονική κλίμακα δεν μπορεί να χρησιμοποιηθεί στην δικιά μας περίπτωση, επειδή ακόμα κια να μπορεί δώσει προτεραιότητα στις λειτουργίες σε αύξουσα ή φθίνουσα σειρά, δεν καθορίζει πόση σημαντική είναι μια λειτουργία σε σχέση με άλλες. Αντίθετα, η κλίμακα αναλογίας είναι πιο ισχυρή επιτρέποντας συγκρίσεις και μας δίνει την δυαντότητα να ποσοτικοποιήσουμε πόσο πιο σημαντική είναι μια λειτουργία σε σύγκριση με άλλες. Η κλίμακα αναλογίας που έχει επιλεχθεί, κυμαίνεται από 1-100 τοις εκατό.

Μια συγκριτική ανάλυση των πλέον κατάλληλων παραγόντων προτεραιότητας που έχουν ήδη χρησιμοποιηθεί σε σχετικούς τομείς, όπως οι απαιτήσεις και η

προτεραιότητα των Unit-Test, βοηθούν στο να αποφασιστούν οι προτεινόμενοι παράγοντες προτεραιότητας. Μεταξύ των διάφορων τεχνικών ιεράρχησης, επιλέχθηκε η Αθροιστική Ψηφοφορία (Cumulative Voting) ή 100-dollar Test [13], η οποία είναι μια ευρέως χρησιμοποιούμενη τεχνική για την ιεράρχηση των απαιτήσεων. Ωστόσο, τόσο η τεχνική ιεράρχησης όσο και οι προτεινόμενοι συντελεστές ιεράρχησης προσαρμόζονται ώστε να ταριάζουν στην δικιά μας περίπτωση. Οι παρακάτω παράγοντες χρησιμοποιούνται ως είσοδοι (inputs) για την ιεράρχηση των λειτουργιών:

1) Σημαντικότητα (Importance, σταθμισμένη: επείγουσα ανάγκη υλοποίησης, σημασία για την αρχιτεκτονική και τη δομή του προϊόντος)

2) Πολυπλοκότητα (Complexity, σταθμισμένη: δομή κώδικα, λειτουργική πολυπλοκότητα)

3) Εξάρτηση (Dependency, σταθμισμένη: εξάρτηση ανταλλαγής δεδομένων, εξάρτηση αλληλεπιδράσεων)

Η Σημαντικότητα (Importance) είναι μια πολύπλευρη ιδέα, καθώς εξαρτάται από την οπτική που λαμβάνουμε υπόψη κατά τη διαδικασία ταξινόμησης. Ως εκ τούτου, αξιολογείται ως ο σημαντικότερος από του τρείς αυτούς παράγοντες προτεραιότητας και χαρακτηρίζεται με περισσότερους βαθμούς από τους άλλους δύο [1]. Ο αρχικός κατάλογος των αποτιμιμένων λειτουργιών ταξινομείται και γίνεται ο τελικός κατάλογος προτεραιοτήτων των εφαρμοζόμενων λειτουργιών. Η προτεινόμενη προσέγγιση μπορεί να εφαρμοστεί εξίσου είτε σε απλή μέθοδο/συνάρτηση είτε σε κλάση ή σε πακέτο.

ΕΠΙΛΟΓΟΣ

Οι τεχνικές κατανόησης κώδικα και οι προσεγγίσεις προτεραιότητας έχουνε χρησιμοποιηθεί για την αποκάλυψη χαμένων πληροφοριών και υποκείμενων σχεδίων με αντικειμενική και αποτελεσματική αναπαράσταση του συστήματος σε υψηλότερο επίπεδο αφαίρεσης (διαγράμματα κλάσης UML). Έτσι, χρησιμοποιήσαμε εμπειρικά κάποια points (πόντοι), τα οποία points μας βοηθούν στο να δούμε ποια από τις τρεις λειτουργίες (Σημαντικότητα, Πολυπλοκότητα και Εξάρτηση) είναι πιο σημαντική.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτό το σημείο προτείνουμε ένα σύστημα πόντων, το οποίο θα βαθμολογεί τις τρείς παραπάνω λειτουργίες: Σημαντικότητα, Πολυπλοκότητα και Εξάρτηση.

Στον παρακάτω πίνακα παρουσιάζουμε τις μεταβλητές που χρησιμοποίησαμε για να βαθμολογήσουμε την κάθε λειτουργία που επιτελεί και κάνουμε μια μικρή επεξήγηση για το τι αντιπροσωπεύει η κάθε μεταβλητή.

Όνομα	Επεξήγηση	Βαθμοί
eachStandardClassObject	Τα αντικείμενα που προέρχονται από standard Κλάσεις της Java (πχ Object, Date κτλ)	0.3
eachCustomClassObject	Τα αντικείμενα που προέρχονται από Κλάσεις που έχει δημιουργήσει ο χρήστης	0.5
eachStandardAttribute	Οι μεταβλητές που προέρχονται από standard τύπους της Java (πχ String, int, double κτλ) και είναι παράμετροι σε μεθόδους	0.1
eachCustomAttribute	Οι μεταβλητές που προέρχονται από τύπους που δημιούργησε ο χρήστης και είναι παράμετροι σε μεθόδους	0.4
hasAttribute	Μια μέθοδος έχει μια ή περισσότερες παραμέτρους	0.3
voidMethod	Όταν η μέθοδος είναι void	0.1
standardReturnType	Όταν η μέθοδος επιστρέφει τύπους της Java (πχ String, int, double κτλ)	0.2
customReturnType	Όταν η μέθοδος επιστρέφει τύπους που έχει δημιουργήσει ο χρήστης	0.3
eachMethod	Κάθε μέθοδος που έχει μια Κλάση	0.1

Πίνακας 1: «Πίνακας Πόντων»

ΑΝΑΦΟΡΕΣ

[1] Panagiotis Sfetsos, Lefteris Angelis and Ioannis Stamelos, Towards a framework based Test-driven Reverse Engineering Process: A Case Study.

[2] E.J. Chikofsky and J.H. Cross, Reverse Engineering and Design Recovery: A Taxonomy in IEEE Software, 1990.

[3] E. Eliam, Secrets of Reverse Engineering, Indianapolis, IN: Wiley, 2005.

[4] M. R. Ali, “Why teach reverse engineering?” ACM SIGSOFT SEN, v.30, n.4, pp.1-4, Jul 2005.

[5] K. Beck, Test-driven development: By example, Boston: Addison-Wesley, 2003.

[6] A. O'Hare, and E. Troan. RE-Analyzer: From source code to structured analysis. IBM Systems Journal, 33(1): 110-130, 1994.

[7] S. Tilley, D. Smith. Towards a Framework fro Program Understanding, International Workshop on Program Comprehension – 1996, pp. 19, DOI: 10.119/WPC.1996.501117, 1996.

[8] R. Warden. Software Reuse and Reverse Engineering in Practice. London, England, 1992.

[9] M. Pergher and B. Rossi. Requirements prioritization in software engineering: A systematic mapping study. Proceedings of EmpiRE'13, DOI: 10.1109 / EmpiRE.2013.6615215, pp. 40-44, 2013.

[10] B. Regnell, M. Host, J. Natt och Dag, P. Beremark and T. Hjelm. An Industrial Case Study on Distributed Prioritization in Market-Driven Requirements Engineering for Packaged Software, Requirements Engineering, 6(1), pp. 51-62, 2001.

[11] H. Do, G. Rothermel , and A. Kinneer. Empirical studies of test case prioritization in a junit testing environment. Proceedings of the 15th International Symposium on Software Reliability Engineering, 2004.

[12] Kamal, A.W. A Hierarchical Approach To Software Testing, Master Thesis, MSE-2006-07, pp. 65. Blekinge Institute of Technology, 2006.

[13] D. Leffingwell and D. Widrig. Managing Software Requirements – A Unified Approach, Addison-Wesley, Upper Saddle River, NJ, 2000.

ΒΙΒΛΙΟΓΡΑΦΙΑ

P. Antonini, P. Benedusi, G. Cantone and A. Cimitile, (1987), Maintenance and reverse engineering: low-level design documents production and improvement, Austin, Texas.

R.S. Arnold, (1993), Software Reengineering. IEEE Computer Society Press.

K. Beck, (2003), Test-Driven development: By example, Boston.

G. Canfora and M. Di Penta , (2007), New Frontiers of Reverse Engineering, in Proc, Future of Software Engineering, Minneapolis, MN, pp. 326-341.

E.J. Chikofsky and J.H. Cross, (1990), Reverse Engineering and Design Recovery: A Taxonomy in IEEE Software.

R. N. Charette, (1986), Software Engineering Environments: Concepts and Technology, Intertext Publications, Int, New York.

H. Do, G. Rothermel, and A. Kinneer., (2004), Empirical studies of test case prioritization in a junit testing environment.

V.H.S. Durelli, R.A.D. Penteado, S. Borges, (2010), An Iterative Reengineering Process Applying Test-Driven Development and Reverse Engineering Patterns.

E. Eliam, (2005), Secrets of Reverse Engineering, Indianapolis.

Kamal, A.W., (2006), A Hierarchical Approach To Software Testing.

Mamta Garg and Manoj Kumar Jindal, (2009), Reverse Engineering – Roadmap to Effective Software Design, International Journal of Recent Trends in Engineering, Vol1, No. 2.

H.A. Müller, J.H. Jahnke, D.B. Smith, M. Storey, S.R Tilley and K. Wong, (2000), Reverse Engineering: a roadmap, in Proc. Conf. Future of Software Engineering, Limerick, Ireland.

M. L. Nelson, (1996), A Survey of Reverse Engineering and Program Comprehension.

Panagiotis Sfetsos, Lefteris Angelis and Ioannis Stamelos (), Towards a framework based Test-driven Reverse Engineering Process: A Case Study, Thessaloniki, Greece.

M. Pergher and B. Rossi, (2013), Requirements prioritization in software engineering: A systematic mapping study, pp. 40-44.

P. Runeson, M. Host, A. Rainer and B. Regnell, (2012), Case Study Research in Software Engineering: Guidelines and Examples.

E. Shihab, Z.M Jiang, B. Adams, A. Hassan, R. Bowerman, (2010), Prioritizing Unit Test Creation for Test-Driven Maintenance of Legacy Systems.

E. Shihab, Z.M Jiang, B. Adams, A. Hassan, R. Bowerman, (2011), Prioritizing the creation of unit tests in legacy software systems.

I. Somerville, (1989), Software Engineering, 3rd edn, Addison Wesley.

P. Tonella and A. Potrich, (2005), Reverse Engineering of Object Oriented Code, Monographs in Computer Science, Springer.

R. Warden, (1992), Software Reuse and Reverse Engineering in Practice, London, England.

B. W. Weide, W. D. Heym, J.E. Hollingsworth, (1995), Reverse engineering of legacy code exposed, Seattle, Washington, WA.

M. Weiser, (1984), Program slicing. IEEE Transactions on Software Engineering, Volume 10, Issue 4, pages 352-357, IEEE Computer Society Press.

J. Willey and Sons, (2005), Reversing: secrets of reverse engineering.

C. Wohlin, P. Runeson, M. Höst, M. Ohlson, B. Regnell, A. Wesslén, (2000), Experimentation in software engineering: an introduction.

ΠΑΡΑΡΤΗΜΑΤΑ

Ανάλυση κώδικα

Σε αυτό το κεφάλαιο θα γίνει η ανάλυση κάποιων σημαντικών Κλάσεων και θα συμπεριλάβουμε κάποια κομμάτια κώδικα για κάποιες συγκεκριμένες λειτουργίες του plug-in.

Η Κλάση CalculatePoints.java

Περιέχει μια Public μέθοδο η οποία δέχεται σαν παράμετρο μια λίστα τύπου ClassForClass. Η μέθοδος ξεκινάει και για κάθε αντικείμενο τις λίστας κοιτάει αν είναι class ή abstract ή αλλιώς interface ο τύπος της class και ανάλογα συνεχίζει ο κώδικας για να βαθμολογήσει την κλάση και τις μεθόδους αυτής. Αν η κλάση δεν είναι interface τότε ελέγχουμε τα attributes αν έχει η κλάση και ανάλογα με τον τύπο τους δίνουμε πόντους στην κλάση. Αν κάποιο αντικείμενο προέρχεται από κάποια κλάση που έχει φτιάξει ο χρήστης τότε δίνονται στην κλάση 0.5 πόντοι ενώ αν κάποιο αντικείμενο προέρχεται από κάποια βασική βιβλιοθήκη της Java τότε η κλάση παίρνει 0.3 πόντους.

Μετά το πέρας της επανάληψης για όλα τα attribute που έχει η κλάση η μέθοδος επιστρέφει το σύνολο των πόντων και αυτοί εισάγονται στη μέθοδο μέσω της set μεθόδου για να ξέρει το κάθε αντικείμενο πόσους πόντους έχει ως κλάση. Μετά η μέθοδος για κάθε μέθοδο που μπορεί να έχει ένα αντικείμενο ClassForClass ελέγχει τι είναι. Αν η μέθοδος είναι τύπου void τότε δίνονται στην μέθοδο 0.1 πόντοι. Αν όμως η μέθοδος έχει κάποιο βασικό τύπο της Java τότε της δίνονται 0.2 πόντοι. Αν η μέθοδος επιστρέφει κάποιον τύπο που έχει ορίσει ο χρήστης τότε η μέδοθος παίρνει 0.3 πόντους.

Μετά η μέθοδος συνεχίζει κοιτώντας τις παραμέτρους για κάθε μια από αυτές. Αν έχει παραμέτρους τότε της δίνονται 0.3 πόντοι. Κάθε παράμετρος που δέχεται η μέθοδος ελέγχεται να είναι κάποιος βασικός τύπος της Java και αν είναι η μέθοδος παίρνει 0.1 πόντους ενώ αν είναι κάποιος τύπος που έχει ορίσει ο χρήστης τότε η μέθοδος παίρνει 0.4 πόντους. Οι πόντοι μέσω set μεθόδου περνούν στο αντικείμενο και έτσι κάθε αντικείμενο ξέρει πόσους πόντους έχει η κάθε μέθοδος του. Αν η μέθοδος δεν περιέχει καμία παράμετρο τότε η μέθοδος κοιτάει να δει το τύπο της και της δίνει πόντους ανάλογα όπως έχει αναφερθεί πιο πάνω. Αν η κλάση είναι interface τότε η μέθοδος κοιτάει πόσους κλάσεις το υλοποιούν και για κάθε μια παίρνει 1 πόντο.

Η Κλάση ClassForAttributes.java

Το αρχείο αναπαριστά τα αντικείμενα μια κλάσης. Έχει 3 πεδία ένα για τον τύπο πρόσβασης που είναι το αντικείμενο (public, protected, private), έναν τύπο για τον τύπο του αντικειμένου και ένα τύπο για το όνομα του αντικειμένου. Έχει έναν δομητή που παίρνει τα παραπάνω πεδία ως ορίσματα κατά την δημιουργία του αντικειμένου της κλάσης αυτής. Επίσης έχει μεθόδους set και get για όλα τα παραπάνω πεδία καθώς και μια toString μέθοδο για την εμφάνισή τους.

Η Κλάση ClassForClass.java

Το αρχείο είναι η αναπαράσταση μια κλάσης στη Java. Περιέχει ένα πεδίο για το όνομα της κλάσης, μετά έχει 3 πεδία αληθείας το οποίο ανάλογα με το πιο είναι αληθές ορίζουν αν είναι απλή κλάση, αν είναι abstract κλάση, ή αν είναι interface, επίσης έχει 2 ακόμα πεδία αληθείας τα οποία αν είναι αληθής κάποιο από τα δυο ή και τα δυο η κλάση επεκτείνει κάποια πιο γενική κλάση και υλοποιεί ένα ή περισσότερα interfaces.

Το επόμενο πεδίο είναι μια λίστα η οποία περιέχει την κλάση που μπορεί να επεκτείνει η κλάση. Στη συνέχεια υπάρχει ακόμα μια λίστα η οποία περιέχει όλα τα interfaces ή το interface τα οποία υλοποιεί το αντικείμενο της κλάσης. Το επόμενο πεδίο είναι η λίστα μέσα στην οποία υπάρχουν όλες οι μέθοδοι που περιέχει μια κλάση και είναι τύπου ClassForMethods. Μετά υπάρχει ακόμα μια λίστα η οποία περιέχει όλα τα attributes της κλάσης και είναι τύπου ClassForAttributes. Το επόμενο πεδίο είναι για όνομα του πακέτου στο οποίο ανήκει η κλάση. Η επόμενη λίστα είναι από τις πιο σημαντικές καθώς περιέχει όλες εκείνες τις κλάσεις που έχει αντικείμενο της εν λόγω κλάσης. Τα επόμενα δυο πεδία είναι για τους πόντους.

Πιο συγκεκριμένα το πρώτο είναι για τους πόντους όταν το αντικείμενο κλάση είναι class ή abstract class και το τελευταίο είναι για να τους πόντους όταν το αντικείμενο κλάση είναι interface. Η κλάση έχει 4 δομητές. Ο πρώτος, είναι ο default κενός δομητής. Ο επόμενος δομητής χρησιμοποιείται για την αρχικοποίηση του αντικειμένου όταν αυτό είναι class και έχει σαν παραμέτρους τρεις μεταβλητές αληθείας που η πρώτη δηλώνει πως το εν λόγω αντικείμενο είναι class και οι δυο επόμενες αν επεκτείνει κάποια κλάση και/ή υλοποιεί κάποιο interface. Οι δυο επόμενοι δομητές υπάρχουν για να αρχικοποιούν αντικείμενα κλάσεις όταν αυτά είναι είτε interface είτε abstract class. Στον τελευταίο δομητή υπάρχει ένα πεδίο μόνο και μόνο για να μπορεί να υπάρξει ο δομητής λόγω ότι υπάρχει ήδη ένας δομητής με ένα όρισμα.

Η μέθοδος findTheTruth είναι τύπου void και αν η κλάση είτε επεκτείνει κάποια άλλη είτε υλοποιεί κάποιο Interface τότε η μέθοδος αυτή προσθέτει στη σωστή λίστα την κλάση ή το interface που δέχεται ως παραμέτρους. Η επόμενες τρεις μέθοδοι με ονόματα ForClass, ForInterface και ForAbstract είναι η μέθοδοι για την εμφάνιση του αντικειμένου ανάλογα τι είναι όπως δηλώνουν και τα ονόματα των μεθόδων. Υπάρχουν μέθοδοι set και get για όλα τα πεδία της κλάσης. Οι μέθοδοι με ονόματα show, showAttri και showMethods δέχονται ως παράμετρο μια λίστα και επιστρέφουν το περιεχόμενο της για εμφάνιση.

```
public void findTheTruth(String motherClass, String InterfaceImplementation)
    if(one.equals("yes")) {
        if (doesExtention) {
            whatExtends.add(motherClass);
        }
        if (doesImplementation) {
            whichInterface.add(InterfaceImplementation);
        }
    }
}
```

Σχήμα 13: «Η μέθοδος findTheTruth.»

Η Κλάση ClassForMethods.java

Το αρχείο αναπαριστά μια μέθοδο γραμμένη σε κώδικα Java. Το πρώτο πεδίο του είναι η προσβασιμότητα που έχει η μέθοδος τόσο από άλλες κλάσεις όσο και από αυτή που βρίσκεται. Επόμενο πεδίο είναι ο τύπος της μεθόδου που μπορεί να είναι void ή να επιστρέφει έναν τύπου της Java ή κάποιον custom που έχει ορίσει ο χρήστης. Το επόμενο πεδίο είναι ένας πίνακας που περιέχει όλα τα ορίσματα που έχετε η μέθοδος μαζί με το τύπο και το όνομα τους. Το προτελευταίο όρισμα δηλώνει σε ποια κλάση ανήκει η μέθοδος αυτή.

Τελευταίο όρισμα οι πόντοι της μεθόδου που θα της δοθούν κατά την αξιολόγηση όπως έχει αναφερθεί πιο πάνω. Η κλάση έχει ένα δομητή ο οποίος

δέχεται πέντε ορίσματα τα οποία είναι όλα τα πεδία της κλάσης εκτός από τους πόντους και έτσι αρχικοποιείται ένα αντικείμενο. Ακολουθούν οι μέθοδοι set και get για όλα τα πεδία και μετά η toString μέθοδος για την εμφάνιση. Η τελευταία μέθοδος με όνομα showAttributes παίρνει σαν παράμετρο έναν πίνακα που περιέχει όλες τις παραμέτρους που δέχεται η μέθοδος και τις επιστρέφει για να εμφανιστούν μέσω της toString.

Η Κλάση GlobalMenuHandler.java

Το βασικό αρχείο μέσα στο οποίο υπάρχει η μέθοδος run μέσα από την οποία ξεκινάει το plug in.

Ξεκινώντας έχουν οριστεί κάποια αντικείμενα για να μπορούμε να έχουμε πρόσβαση στις κλάσεις με τις οποίες θα αποδομήσουμε τον κώδικα που επιθυμεί ο χρήστης να πάρουμε όλα τα απαραίτητα στοιχεία να τα εισάγουμε στα αντικείμενα μας και μετά να τα αξιολογήσουμε και στο τέλος να τα παρουσιάσουμε στην χρήστη. Το πρώτο αντικείμενο που ορίζουμε είναι της κλάσης getJavaFiles και έχει όνομα getfiles. Το δεύτερο είναι μια λίστα τύπου String με όνομα getJavaFilesPath και θα περιέχει όλα τα μονοπάτια προς τα .class αρχεία για να μπορέσει το Plugin για κάθε αρχείο να το βρεί και να το αποδομήσει. Το επόμενο αντικείμενο της κλάσης ParsingClassFiles με όνομα pcf είναι το πιο σημαντικό αντικείμενο καθώς με αυτό θα κληθεί η μέθοδος που θα αποδομήσει ένα ένα όλα τα .class αρχεία που υπάρχουν στην λίστα getJavaFilesPath. Το τρίτο αντικείμενο είναι της κλάσης UMLClassDrawer με όνομα classDraw το οποίο μετά την αποδόμηση και την δημιουργία όλου του μοντέλου το οποίο και θα του δοθεί μέσω παραμέτρου θα κληθεί να το απεικονίσει για να έχει οπτική επαφή ο χρήστης για το ποιές κλάσεις και μέθοδοι του είναι αρκετά σημαντικές και χρήζουν προσοχής και testing.

Στη συνέχεια, έχουμε μια λίστα τύπου ClassForClass με όνομα classes μέσα στην οποία θα προστίθεται κάθε ένα αρχείο που θα αποδομείται και θα δημιουργείται το μοντέλο το οποίο θα εμφανιστεί στον τελικό χρήστη. Μετά έχουμε ένα αντικείμενο της κλάσης CalculatePoints με την βοήθεια του οποίου θα αξιολογηθούν όλες οι κλάσεις και οι μέθοδοι του μοντέλου αφού έχει τελειώσει με την αποδόμηση και την πλήρης απεικόνιση του και το οποίο θα υπάρχει μέσα στη λίστα classes.

Ξεκινώντας η μέθοδος run αρχικοποιεί το αντικείμενο classDraw της κλάσης UMLClassDrawer με τον κενό δομητή της κλάσης. Στη συνέχεια ορίζεται ένα αντικείμενο τύπου File το οποίο και θα μας βοηθήσει να εντοπίσουμε την αρχή από την οποία θα ξεκινήσει να ψάχνει ο κώδικας για να βρει τα .class αρχεία μέσα

στους υποφακέλους. Αφού ανοίξει το παράθυρο επιλογής φακέλου και ο χρήστης επιλέξει φάκελο τότε σε μια μεταβλητή με το όνομα path ο κώδικας κρατάει το path του φακέλου που θα αποτελεί την αρχή και δίνεται σαν παράμετρος μαζί με την την λίστα getJavaFilesPath η οποία θα γεμίσει με όλα τα paths από τα .class αρχεία που υπάρχουν από τον επιλεγμένο φάκελο και μέσα σε όλους τις υποφακέλους που υπάρχουν από κάτω του. Μετά για κάθε ένα path της getJavaFilesPath που μόλις γέμισε η μέθοδος getAllJavaFiles περνάτε σαν παράμετρος στην μέθοδο ParsingListwithClassFiles η οποία θα επιστρέψει ένα αντικείμενο τύπου ClassForClass με το οποίο θα αναπαριστά την κλάση που περιέχει το .class αρχείο του path.

Το αντικείμενο προστίθεται στη λίστα classes. Όταν τελειώσει η επανάληψη και έχει γίνει αποδόμηση όλων των αρχείων και δόμηση όλων των αντικειμένων τότε καλείται η μέθοδος calculatePointsForMethods η οποία παίρνει ως παράμετρο όλη την λίστα classes και αξιολογεί κάθε ένα αντικείμενο της όπως έχει αναφερθεί πιο πάνω. Πριν γίνει η εμφανίσει τον αποτελεσμάτων μετά την βαμθολόγιση όλες οι μέθοδοι κάθε κλάσης περνάνε μέσα απο ταξινόμιση ώστε να εμφανιστούν κατά αύξουσα σειρα στον χρήστη με σκοπό να γνωρίζει ποια είναι η πιο σημαντική, αρχίζοντας απο αυτήν την εξακρίβωση της σωστής τους λειτουργίας. Τέλος καλείται η μέθοδος buildTable που δέχεται και αυτή την λίστα classes ως παράμετρο και βάση της αξιολόγησης που έχει γίνει σε κάθε αντικείμενο της λίστας θα εμφανίσει αυτά που έχουν μαζέψει πόντους ίσα και πάνω από το ένα όριο.

Η Κλάση ParsingClassFiles.java

Το αρχείο είναι ο πυρήνας του plug in καθώς περιέχει όλη την λογική για την δόμηση των αντικειμένων τα οποία θα αναπαριστούν της κλάσεις με τις μεθόδους και τα πεδία τους του κώδικα που έχει δώσει ο χρήστης προς επεξεργασία

Η κλάση ξεκινάει και ορίζει μια λίστα τύπου ClassForAttributes στην οποία θα αποθηκευτούν όλα τα attribute κάθε κλάσης. Στη συνέχεια ορίζει μια ακόμα λίστα τύπου ClassForMethods στην οποία θα αποθηκευτούν όλες οι μέθοδοι που θα περιέχει μια κλάση. Έπειτα, ορίζει ένα αντικείμενο από κάθε μια κλάση από τις ClassForClass, ClassForMethods ,ClassForAttributes, SearchingForAttributes και τρία πεδία δυο για το όνομα της κλάσης ένα που θα περιέχει το όνομα μαζί με το με το πακέτο στο οποίο ανήκει η κλάση και ένα για το όνομα της κανονικά καθώς και μια μεταβλητή αληθείας για γνωρίζει ο κώδικας αν η κλάση είναι μέρος ενός πακέτου ή όχι.

Η κλάση έχει μια μέθοδος η οποία δέχεται ένα όρισμα. Η μέθοδος έχει όνομα ParsingListwithClassFiles και είναι υπεύθυνη για να αποδομήσει ένα .class αρχείο και να δομήσει τα κατάλληλα αντικείμενα έτσι ώστε να γίνει σωστά η απεικόνιση του αρχείου μέσα από τα αντικείμενα. Η μέθοδος ξεκινάει και ορίζει ένα αντικείμενο της κλάσης ClassParser με παράμετρο το path του .class αρχείου και έτσι ελέγχουμε αν η κλάση είναι απλή class ή abstract class ή interface. Αν η κλάση είναι απλή τότε το δημιουργείται το αντικείμενο της κλάσης ClassForClass που είχαμε ορίσει πιο πάνω με όνομα classobj και ο δομητής του παίρνει σαν ορίσματα τρεις τιμές αληθείας από τις οποίες η πρώτη είναι αληθής που σημαίνει ότι η κλάση είναι απλή και οι άλλες δυο είναι ψευδής.

Στο παρακάτω κομμάτι του κώδικα βλέπουμε πώς κάνουμε parse ένα αρχείο τύπου .class, ώστε να πάρουμε τις πληροφορίες που θέλουμε (στο συγκερκριμένο σημείο παίρνουμε πληροφορίες για μια Κλάση και για μια Υπέρκλαση).

```
//Check If the Class on path is a Class
if(cp.parse().isClass()) {
    classobj = new ClassForClass(true, false, false);
    cp = new ClassParser(path);
    className = cp.parse().getClassName();
    simpleClassName = className.substring(className.lastIndexOf('.') + 1);
    classobj.setClassName(simpleClassName);
    cp = new ClassParser(path);
    classobj.setPackageName(cp.parse().getPackageName());
    if(!(classobj.getPackageName().isEmpty())){
        hasPackage = true;
    }
    cp = new ClassParser(path);
    //Check if the class on the path has a SuperClass
    String superclassname = cp.parse().getSuperclassName();
    if(!(superclassname.equals("java.lang.Object"))) {
        String simpleSuperClassName = superclassname.substring(superclassname.lastIndexOf('.') + 1);
        classobj.setWhatExtends(simpleSuperClassName);
        if(!(superclassname.isEmpty())){
            classobj.setDoesExtention(true);
        }
    }
```

Σχήμα 14: «Έλεγχος αν είναι κλάση.»

Στη συνέχεια του κώδικα ξαναορίζουμε το αντικείμενο της κλάσης ClassParser με το ίδιο path και παίρνουμε το όνομα της κλάσης μαζί με το πακέτο στο πακέτο και αν συνέχεια παίρνουμε μόνο το όνομα της κλάσης και το ορίζουμε μέσα από την setClassName στο classobj αντικείμενο. Ξαναορίζουμε το cp (ClassParser αντικείμενο) και παίρνουμε το πακέτο αν ανήκει μια κλάση και το ορίζουμε μέσω της setPackageName στο classobj. Αν η κλάση ανήκει σε κάποιο πακέτο τότε η μεταβλητή hasPackage γίνεται αληθής. Ορίζουμε και πάλι το αντικείμενο cp και κοιτάμε αν η κλάση κάνει extend κάποια κλάση και σε περίπτωση που κάνει κάποια, πέρα από την Object, τότε την ορίζουμε στο αντικείμενο classobj με την μέθοδο setWhatExtends και επίσης ορίζουμε σε αληθής την μεταβλητή doesExtention του αντικειμένου classobj με την μέθοδο setDoesExtention σε αληθής. Ορίζουμε και πάλι το cp και τώρα κοιτάμε για πιθανά Interface που μπορεί να υλοποιεί η κλάση. Τα βάζουμε σε ένα πίνακα τύπου String και αν συνέχεια αφού τους αφαιρέσουμε το όνομα του πακέτου και αφήσουμε κρατήσουμε μόνο το όνομα του interface το περνάμε σε μια λίστα τύπου string και μόλις πάρουμε όλα τα interface ορίζουμε την λίστα που τα περιέχει στο classobj με την μέθοδο setWhichInterface και αν συνέχεια η λίστα δεν είναι άδεια ορίζουμε και την μεταβλητή doesImplementation σε αληθής με την μέθοδο setDoesImplementation.

Στη συνέχεια και αφού ορίσουμε το cp αντικείμενο παίρνουμε τα attribute της κλάσης στον πίνακα fields που έχει οριστεί. Μετά παίρνουμε με την βοήθεια της μεθόδου split την προσβασιμότητα τον τύπο και το όνομα του attribute και αρχικοποιούμε το αντικείμενο attriobj το αποθηκεύουμε στη λίστα attriList και προσθέτουμε και στη λίστα conectsWith του αντικειμένου classobj τον τύπο του αντικειμένου ώστε να γνωρίζουμε με ποιές άλλες κλάσεις συνδέεται η εν λόγω κλάση. Η διαδικασία συνεχίζεται για όλα τα attribute της κλάσης. Και τέλος ορίζουμε το αντικείμενο cp και παίρνουμε όλες τις μεθόδους που έχει η κλάση και τις αποθηκεύουμε στον πίνακα meth που έχουμε ορίσει. Μετά με την βοήθεια της μεθόδου split παίρνουμε την προσβασιμότητα, τον τύπο και το όνομα μαζί με τις παραμέτρους που μπορεί να δέχεται η μέθοδος και καλώντας την μέθοδο findAttributes περνώντας της σαν παράμετρο την θέση i του πίνακα meth και μια μεταβλητή αληθείας που για να γνωρίζει η μέθοδος αν η κλάση είναι μέρος ενός πακέτου και η μέθοδος μας επιστρέφει ένα πίνακα με όλα τα ορίσματα που δέχεται η μέθοδος με τον τύπο και το όνομα τους. Μετά αρχικοποιούμε το αντικείμενο methodobj και αποθηκεύουμε στη λίστα methodList.

Αν η κλάση είναι interface τότε αρχικοποιούμε το αντικείμενο classobj με μια μεταβλητή αληθείας που δηλώνει ότι είναι interface. Έπειτα παίρνουμε το όνομα του και το θέτουμε στο αντικείμενο classobj με την μέθοδο setClassName και το όνομα του πακέτου που ανήκει και το θέτουμε και αυτό με την μέθοδο

setPackageName. Μετά παίρνουμε όλες τις μεθόδους που υλοποιεί το interface και ακριβώς με τον ίδιο τρόπο που περιγράφτηκε πιο πάνω αρχικοποιούμε το αντικείμενο methodobj και το αποθηκεύουμε στη λίστα methodList. Αν η κλάση είναι abstract ακολουθείτε η ίδια διαδικασία που περιγράφηκε όταν η κλάση είναι απλή class.

Στο τέλος οι λίστες attriList και methodList αποθηκεύονται στο αντικείμενο classobj με τις μεθόδους setAttris και setMethods και έτσι έχουμε μια κλάση με όλα τις τα χαρακτηριστικά αποθηκευμένα και η κλάση επιστρέφει το αντικείμενο classobj. Η διαδικασία επαναλαμβάνεται για όλα τα .class αρχεία που έχουν βρεθεί.

Στο παρακάτω σημείο κώδικα, παίρνουμε τις μεταβλητές μιας Κλάσης.

```
//The attributes of each class
Field fields[] = new Field[1000];
cp = new ClassParser(path);
fields = cp.parse().getFields();
for(int i = 0; i < fields.length; i++) {
    String string2 = fields[i].toString();
    String[] parts2 = string2.split(" ");
    if(!(classobj.getPackageName().equals(""))){
        String[] attriType = parts2[1].split("\\.");
        attriobj = new ClassForAttributes(parts2[0],attriType[attriType.length-1],parts2[2]);
    }else{
        attriobj = new ClassForAttributes(parts2[0],parts2[1],parts2[2]);
    }
    attriList.add(attriobj);
    classobj.getCoonectsWith().add(attriobj.getType());
}
```

Σχήμα 15: «Μεταβλητές μιας κλάσης.»

Η Κλάση SearchingForAttributes.java

Η κλάση SearchingForAttributes έχει τρεις μεθόδους συνολικά, μια Plublic για να μπορούν να έχουν πρόσβαση στις επόμενες δυο που διαθέτει οι οποίες είναι private.

Η public μέθοδος με όνομα findAttributes δέχεται σαν παραμέτρους ένα string και μια μεταβλητή αληθείας. Ορίζει ένας πίνακα τύπου String μέσα στον οποίο κάθε θέση του θα είναι οι παράμετροι κάθε μεθόδου με τον τύπο και όνομα τους το κάθε ένα. Ανάλογα με το αν είναι αληθής η μεταβλητή αυτή, επιλέγεται ποια από τις δυο μεθόδους θα κληθεί για να πάρει τις παραμέτρους τις μεθόδου και οι δύο μέθοδοι ακολουθούν την ίδια ακριβώς λογική και διαφέρουν σε πολύ μικρές λεπτομέρειες που αφορούν καθαρά αν η κλάση ανήκει σε κάποιο πακέτο ή όχι. Οι δυο μέθοδοι αφαιρούν με την μέθοδο split πρώτα την αριστερή και μετά την δεξιά παρένθεση και στη συνέχεια τα κόμματα και έτσι έχουμε κάθε μια μεταβλητή που δέχεται η μέθοδος με την τύπο και το όνομα της και επιστρέφεται ο πίνακας.

Η κλάση GetJavaFiles.java

Το αρχείο έχει μια public μέθοδο που δέχεται δυο ορίσματα το πρώτο είναι το Path που έχει δώσει ο χρήστης από το οποίο θα αρχίσει να ψάχνει ο κώδικας ώστε να εντοπίσει όλα τα .class αρχεία και να κρατήσει τα paths όλων μέσα στη λίστα που δέχεται σαν δεύτερο όρισμα και την οποία στο τέλος θα επιστρέψει. Ο κώδικας ελέγχει αν το αρχείο που βλέπει είναι φάκελος, και αν είναι, τότε καλεί και πάλι την μέθοδο για να συνεχίσει όταν βρει αρχεία ελέγχει αν έχουν κατάληξη .class και τα προσθέτει στην λίστα fileList την οποία και μετά το πέρας των επαναλήψεων επιστρέφει.

Η Κλάση UMLClassDrawer.java

Η κλάση UMLClassDrawer είναι υπεύθυνη για την σχεδίαση των κλάσεων και τον μεθόδων τους που έχουν ξεπεράσει ένα κατώτατο όριο. Δίνεται στη μέθοδο buildTable όλη μια λίστα τύπου ClassForClass σαν παράμετρος που περιέχει όλες τις κλάσεις του κώδικα που έχει δώσει ο χρήστης με τους πόντους και την πλήρης καταγραφή κάθε μιας κλάσης. Ο κώδικας ελέγχει τους πόντους κάθε αντικειμένου και παρουσιάζει τις καλύτερες.

Στο επόμενο σημείο κώδικα βλέπουμε πως “ζωγραφίζουμε”/εμφανίζουμε τις πληροφορίες για την κάθε Κλάση στο Class Diagram.

```
//if class is CLass
if(clas.isClass() && (clas.getPoints() > 0.1)){
    classLabel = new Label(clas.getClassName(),
            new Image(d,
            UMLClassFigure.class.getResourceAsStream("/resources/class/class_obj.png")));
    classfont = new Font(null, "Arial", 12, SWT.BOLD);
    if(!(clas.getPackageName().equals(""))){
        packageFont = new Font(null, "Arial", 8, 0);
        packageLabel = new Label(clas.getPackageName(),
                new Image(d,
                UMLClassFigure.class.getResourceAsStream("/resources/class/package_obj.png")));
        packageLabel.setFont(packageFont);
```

Σχήμα 16: «Εμφάνιση ενός Class Diagram.»

Τέλος, σημαντικό είναι αναφέρουμε το πως ανοίγει ένα παράθυρο ώστε να εμφανίζουμε σε αυτό ένα Class Diagram.

```
public void startDrawer(){

    d.syncExec(new Runnable(){

        public void run(){

            //shell.setSize(400, 400);
            shell.scroll(100, 100, 100, 100, 100, 100, true);
            shell.setText("Class Diagram");

        }
    });
    openShell();
    stopDrawer();
}

public void openShell(){

    shell.open();
}

public void stopDrawer(){
    while (!shell.isDisposed())
        while (!d.readAndDispatch())
            d.sleep();
}

public void kill(){
    if(shell.isDisposed()){
        shell.close();
    }
}
```

Σχήμα 17: «Άνοιγμα ενός shell.»

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Σε αυτό το κεφάλαιο παρουσιάζεται ένας σύντομος οδηγός για το πως λειτουργεί το plug-in.

Γενικά Χαρακτηριστικά του Plug-in

Plug-in for Reverse Engineering

Το συγκεκριμένο plug-in είναι ένα λογισμικό το οποίο δεν είναι ήδη εγκατεστημένο στο Eclipse, αλλά πρεπεί να γίνει εγκατάσταση μέσω του Eclipse Marketplace. Η λειτουργία του προγράμματος σας επιτρέπει την δημιουργία Class Diagrams που προέρχεται από κώδικα Java.

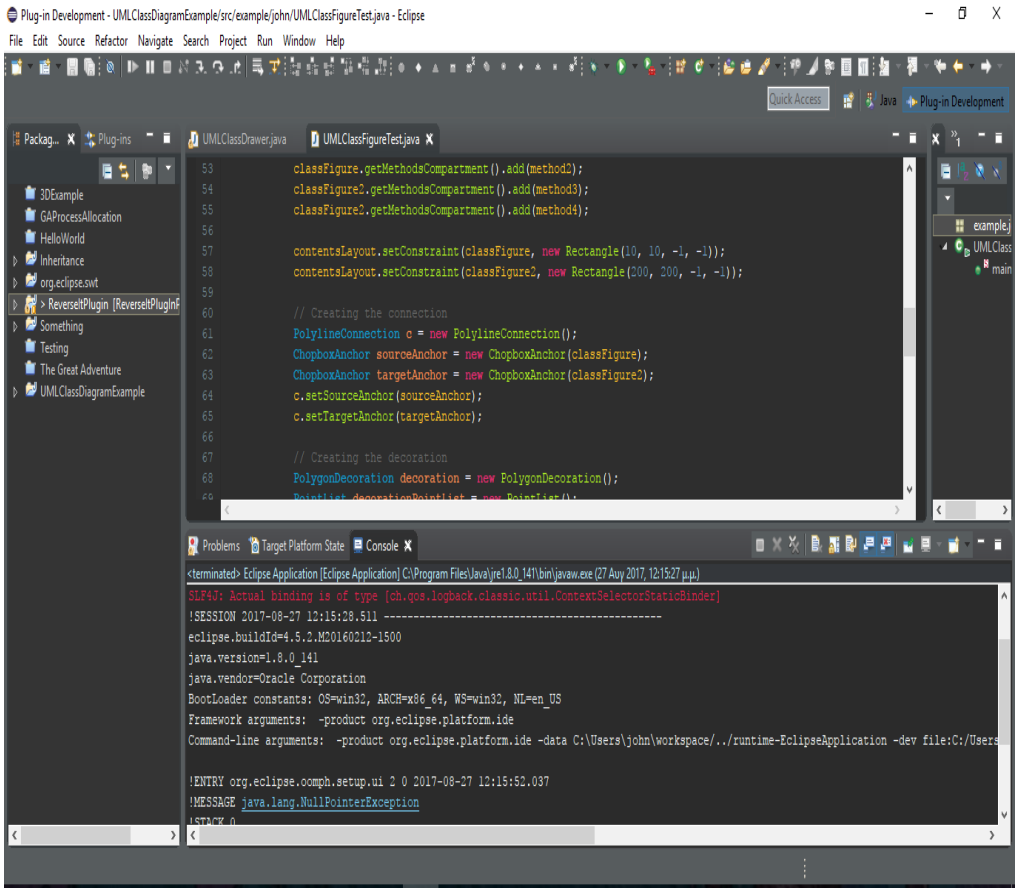
Τύποι αρχείων που υποστηρίζονται από το λογισμικό

- .java
- .class

Περιγραφή επιφάνειας εργασίας

Όσοι έχουν δουλέψει με το πρόγραμμα Eclipse και είναι οικείοι με το εργαλείο αυτό θα γνωρίζουν ότι το περιβάλλον εργασίας είναι απλό και εύχρηστο. Για όσους δεν το έχουν ξαναδουλέψει, το περιβάλλον χωρίζεται σε 4 βασικά μέρη (σχήμα 18) :

- Λειτουργίες (περιλαμβάνει το menu: File, Edit, Source κτλ)
- Project Explorer (εδώ φορτώνονται/δημιουργούνται τα project)
- Editor (το μέρος όπου γράφει κανείς το κώδικα)
- Console (εδώ γίνεται το compile του project)

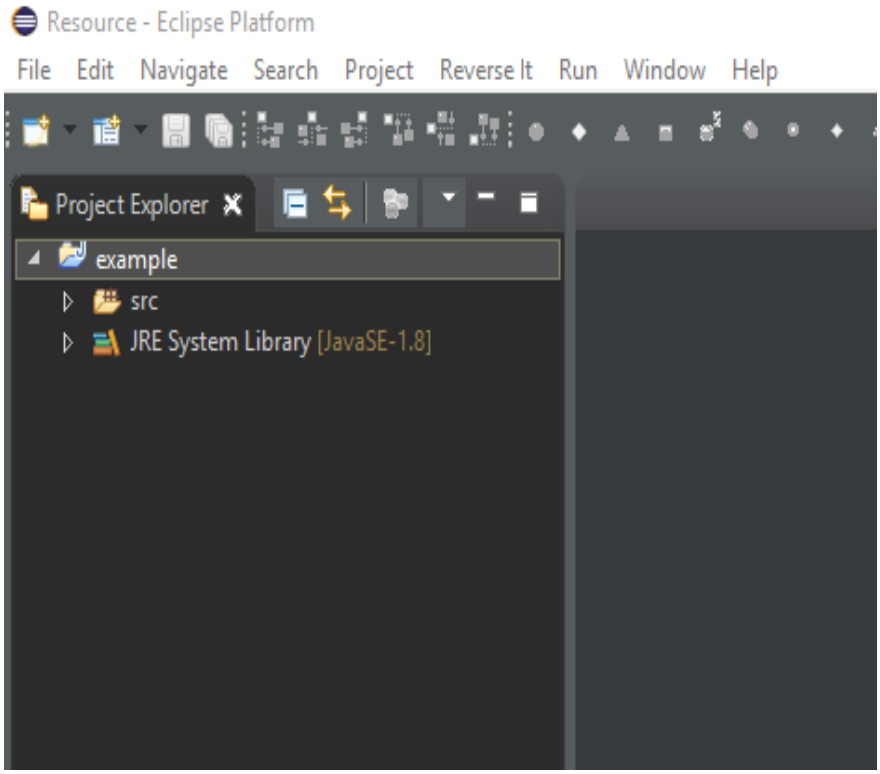


Σχήμα 18: «Παράδειγμα επιφάνειας εργασίας του Eclipse.»

Διαδικασία παραγωγής ενός Class Diagram

1. Επιλογή project

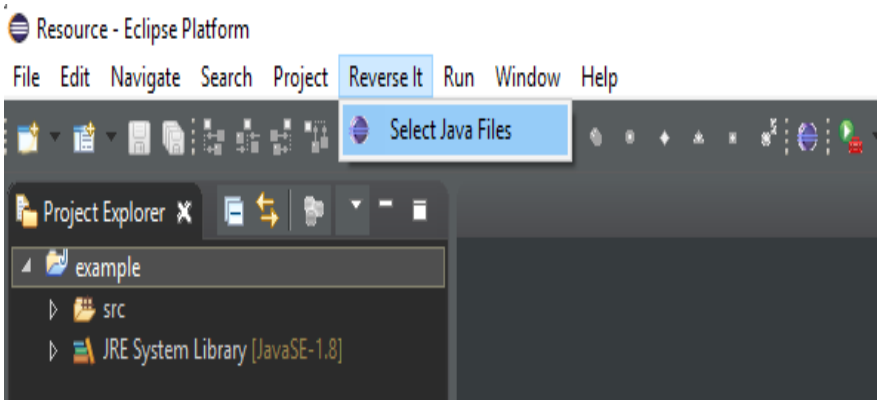
Αρχικά, επιλέξτε το project από το οποίο θέλετε να παράξετε το διάγραμμα όπως φαίνεται στο σχήμα 19:



Σχήμα 19: «Επιλογή ενός project στον project explorer.»

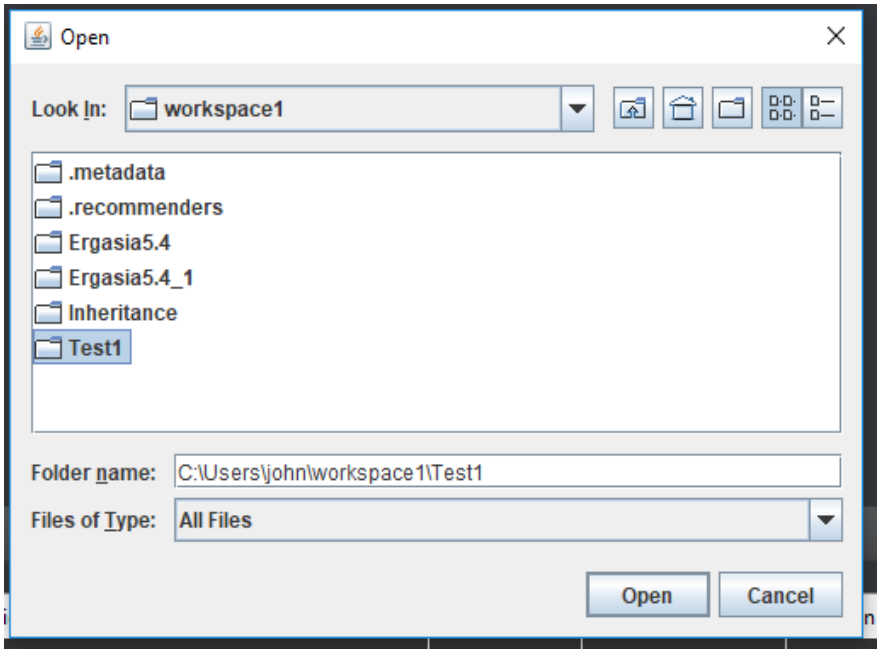
2. Εκκίνηση διαδικασίας παραγωγής Class Diagram

Στη συνέχεια στο menu, υπάρχει μια επιλογή που ονομάζεται “Reverse It”. Επιλέγοντας το “Reverse It” δίνεται μια δεύτερη επιλογή η οποία λέει “Select Java Files” (σχήμα 20).



Σχήμα 20: «Επιλογή από το menu.»

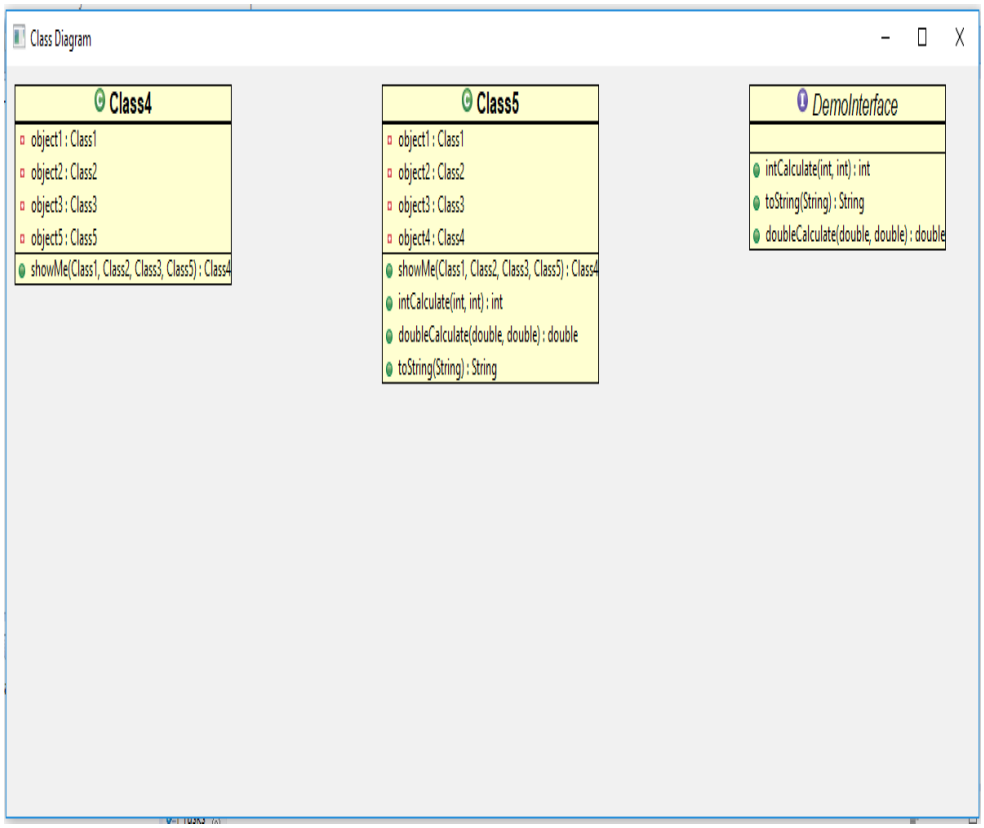
Αφού επιλέξετε “Select Java Files”, θα εμφανιστεί ένα dialog (σχήμα 21) το οποίο σας επιτρέπει να επιλέξετε το project από την θέση που βρίσκεται το project στον υπολογιστή σας.



Σχήμα 21: «Επιλογή αρχείων.»

3. Εμφάνιση του Class Diagram

Τέλος, αφού επιλέξετε τον φάκελο ο οποίος περιέχει τα αρχεία τα οποία θέλετε να εμφανίσετε σε Class Diagram, το αποτέλεσμα (σχήμα 22) φαίνεται κάπως έτσι:



Σχήμα 22: «Αποτέλεσμα διαδικασίας. Οι μέθοδοι που εμφανίζονται έχουν ταξινομηθεί με αύξουσα σειρά.»