ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

CONTINUOUS INTEGRATION

(ΣΥΝΕΧΗΣ ΕΝΣΩΜΑΤΩΣΗ)

Του φοιτητή:

Ρέντα Δημήτρη

Αρ. Μητρώου: 032235

Επιβλέπων Καθηγητής:

Ψαρράς Νικόλαος

Θεσσαλονίκη 2014

Περιεχόμενα

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

Πρόλογος

Η παρούσα πτυχιακή εργασία πραγματεύεται το θέμα της συνεχής ενσωμάτωσης στην ανάπτυξη λογισμικού.

Πρόκειται για μία ανάλυση της κλασσίκης μεθόδου του καταρράκτη που ευδοκίμασε τα προηγούμενα χρόνια, εστιάζοντας στους λόγους σταδιακής αποξένωσης της στον σημερινό τρόπο ανάπτυξης.

Εκτενής ανάλυση γίνεται στο κομμάτι της δοκιμής λογισμικού, testing, παρουσιάζοντας τις διαφορετικές τεχνικές που χρησιμοποιούνται συνήθως. Το στάδιο αυτό θεωρείται από τα πιο κρίσιμα στις μέρες μας.

Επίσης. αναλύονται οι νέες ευέλικτες μέθοδοι Agile, με έμφαση στις eXtreme Programming και SCRUM, παρουσιάζοντας τον λόγο που όλο και περισσότερες ομάδες ανάπτξης λογισμικού τις επιλέγουνε.

Ακολουθεί η εισαγωγή στην συνεχή ενσωμάτωση αναλύοντας τα οφέλη της και τις πρακτικές της. Σε αυτό το κεφάλαιο παρουσιάζεται και το εργαλείο Jenkins, το οποίο χρησιμοποιείται παγκοσμίος στην ανάπτυξη λογισμικού.

Καταλήγοντας, αναπτύζεται μία εφαρμογή χρησιμοποιώντας τις τεχνικές της συνεχούς ενσωμάτωσης, η οποία αποτελεί και οδηγό εγκατάστασης και χρήσης του εργαλείου.

Θα ήθελα να ευχαριστήσω τον καθηγητή μου Ψαρρά Νικόλαο για την πολύτιμη βοήθεια του και συμπαράσταση όλους αυτούς τους μήνες.

Κεφάλαιο 1 Εισαγωγή

Στις μέρες μας το λογισμικό κατέχει πολύ σημαντικό ρόλο στη ζωή μας και άμεσα ή έμμεσα επηρεάζει την καθημερινότητα μας. Ολοένα και περισσότερες διαδικασίες που εκτελούμε κάθημερινά τείνουν να αυτοματοποιούνται. Σε αυτή την διαδικασία συμβάλλει η ταχεία ανάπτυξη του λογισμικού. Για παράδειγμα, προγράμματα που ελέγχουνε το δίκτυο ηλεκτροδότησης, τις πληρωμές μας, τις μεταφορές μας είναι μερικά που αποτελούνε αναπόστατο κομμάτι της ζωής μας. Μπορεί κάποιος να ισχυριστεί ότι το λογισμικό στην εποχή μας είναι πιο σημαντικό από ότι ποτέ. Οπότε η ορθή και μελετημένη ανάπτυξη του, χρησιμοποιώντας αξιόπιστες τεχνικές, κρίνεται απαραίτητη ώστε να συμβάλει στην θετική εμπειρία χρήσης του.

Υπάρχει πληθώρα ειδών λογισμικού που σημαίνει πως υπάρχουνε και διαφορετικοί τρόποι ανάπτυξης αυτών. Σε αυτό το κεφάλαιο θα αναλυθεί ο βασικότερος τρόπος που ονομάζεται το μοντελό καταρράκτη, ενώ πρώταθα γίνει αναφορά στον κύκλο ανάπτυξης του λογισμικού και τις διαφορετικές φάσεις που απαιτούνται.

1.1 Κύκλος Ζωής Ανάπτυξης Λογισμικού

Η ανάπτυξη του λογισμικού περιλαμβάνει τον πελάτη, τον τελικό χρήστη και την ομάδα ανάπτυξης (προγραμματιστές, αναλυτές, δοκιμαστές, διάφοροι ειδικοί).

Το αρχικό βήμα στην όλη διαδικασία είναι η πρώτη συνάντηση με τον πελάτη ώστε να καθοριστούνε οι απαιτήσεις του, σχετικά με το τί θέλει να πετύχει με την δημιουργία ενός καινούριου λογισμικού ή σύστηματος.

Μόλις καθοριστούνε αυτές οι απαιτήσεις, δημιουργείται ένα πρώιμο σχέδιο του συστήματος το οποίο τις καλύπτει. Σκοπός αυτού του βήματος είναι να γίνει η αρχική εκτίμηση στο πώς θα μοιάζει το τελικό προιόν, βάση της προοπτικής του πελάτη. Για παράδειγμα, παρουσίαση στον πελάτη το τί θα παράγει το λογισμικό, σε τί μορφή, δημιουργία αρχικών σχεδιών, ανάλυση ιδεών και γενικά το πώς θα αλληλεπηδρά ο χρήστης με το σύστημα.

Το πρώιμο σχέδιο αναλύεται από τον πελάτη και αφού πάρει την έγκριση του, θα αποτελέσει την βάση για τον πραγματικό σχεδιασμό του λογισμικού. Μέχρι

στιγμής δεν έχει γίνει αναφορά για το τί γλώσσες προγραμματισμού θα χρησιμοποιηθούνε αφού είναι πολυ νωρίς. Πρώτα πρέπει να αποφασίζεται η λειτουργία και μετά το πώς θα αναπτυχθεί αυτή. Ακολουθεί η καθολική επικυροποιήση αυτού του σχεδίου και τότε αρχίζει η συζήτηση μεταξύ της ομάδας για το πώς θα αναπτυχθεί το λογισμικό.

Με την ανάπτυξη του ανάλογου πρόγραμματος, περνάμε στην διαδικασία ελέγχου και δοκιμών, η οποία αποτελείται από διαφορετικά στάδια. Σκοπός είναι να παραχθεί και να παραδοθεί στο τέλος ένα πλήρως λειτουργικό σύστημα με όσο το δυνατόν λιγότερα λάθη. Στο τελικό στάδιο των δοκιμών το σύστημα συγκρίνεται με τις αρχικές προδιαγραφές και το κατά πόσο τις πληρεί.

Το τελευταίο βήμα είναι η παράδοση του συστήματος στον πελάτη, η εγκατάσταση του και η χρήση από τους πραγματικούς χρήστες του, ώστε να απαλειφθούνε τυχόν λάθη. Την ολοκήρωση του σταδίου ακολουθεί μια μακρά περίοδος συντήρησης του συστήματος.

Ο παραπάνω τρόπος είναι ο πλέον παραδοσιακός και εφαρμόζεται σε μεσαία και μεγάλα έργα. Αποτελείται από μία δομημένη σειρά διαδοχικών διαδικασίων, για την σύλληψη των απαιτήσεων, το σχεδιασμό, την ανάπτυξη και την λειτουργία του λογισμικού. (Shari Lawrence Pfleeger, 2010)

Συνοψίζοντας τα βήματα είναι :

- Ανάλυση Απαιτήσεων
- Σχεδιασμός Συστήματος
- Σχεδιασμός Λογισμικού
- Συγγραφή Προγραμμάτων
- Έλεγχος Προγραμμάτος
    - Μονάδας
    - Συγώνευσης
    - Συστήματος
- Παράδοση Συστήματος
- Συντήρηση Συστήματος

Στο ιδανικό σενάριο, αυτές οι δραστηριότητες εκτελούνται σειριακά μία μία και όταν φτάσουμε στην συντήρηση του συστήματος έχουμε ένα πλήρως λειτουργικό λογισμικό. Στην πραγματικότητα αρκετά από αυτά τα βήματα είναι επαναλαμβανόμενα. Για παράδειγμα στον σχεδιασμό του συστήματος μπορεί να προσθεθούνε επιπλέον απαιτήσεις από τον πελάτη ή ακόμα χειρότερα στο στάδιο του ελέγχου να διαπυστωθεί πως το σύστημα δεν καλύπτει κάποιες βασικές απαιτήσεις. Για αυτό τον λόγο έχουνε αναπτυχθεί μέθοδοι οι οποίες αναλαμβάνουνε να καθοδηγήσουνε αποτελεσματικά την διαδικασία ανάπτυξης λογισμικού ώστε να μειωθούνε αυτά τα προβλήματα.

Σκοπός αυτών των μεθόδων είναι η παραγωγή υψηλής ποιότητας λογισμικού μέσω μίας επαναλαμβανόμενης διαδικασίας για τον καλύτερο και ορθότερο έλεγχο καθόλη την διάρκεια της ανάπτυξης.

Κάθε ένα από τα παραπάνω βήματα είναι απο μόνο του μία διεργασία η οποία αποτελείται από επιμέρους δραστηριότητες. Και κάθε επιμέρους δραστηριότητα περιλαμβάνει περιορισμούς, παραγώμενα αποτελέσματα και απαιτούμενους πόρους. Για παράδειγμα η ανάλυση των απαιτήσεων απαιτεί την εισαγωγή προτάσεων σχετικά με την λειτουργία του λογισμικού, και θα παράγει έγγραφα που θα βοηθήσουνε τους προγραμματιστές της ομάδας.

1.2 Ανάλυση Επιμέρους Φάσεων

Σε αυτό το σημείο καλό είναι να γίνει μία ανάλυση στα επιμέρους βήματα της ανάπτυξης λογισμικού.

1.2.1 Ανάλυση Απαιτήσεων

Στην φάση αυτή αναλύεται το ήδη υπάρχον σύστημα (εφόσον υπάρχει) και αναλύονται οι απαιτήσεις και ανάγκες του πελάτη. Συνήθως οι απαιτήσεις χωρίζονται σε :

- Λειτουργικές. Περιγράφουνε τις απαραίτητες απαιτήσεις, που χωρίς αυτές δεν υπάρχει παραδοτέο σύστημα. Για παράδειγμα τί τιμές να δέχεται ένα πεδίο, σε τί μορφή να είναι το αποτέλεσμα, τί λειτουργία να έχει ένα πεδίο.

- Μη Λειτουργικές. Αυτές περιλαμβάνουνε κάποια ποιοτικά χαρακτηριστικά του συστήματος, όπως χρόνο απόκρισης ή σχεδιαστικά χαρακτηριστικά, όπως χρώμα του φόντου μιας φόρμας. (Sommerville, 2011)

Πρέπει να σημειωθεί, πως η διαδικασία αυτή είναι κάτι περισσότερο από απλή καταγραφή των θέλω και των απαιτήσεων του πελάτη. Πρέπει να βρεθούνε οι απαιτήσεις στις οποίες θα συμφωνήσει και η ομάδα ανάπτυξης και σχεδιασμού, για την εφικτή δημιουργία τους. Σύμφωνα με έρευνες αυτό το βήμα είναι και το πιο κρίσιμο σε όλη την διαδικασία ανάπτυξης λογισμικού.

Απαίτηση είναι η υλοποιήση μιας συγκεκριμένης συμπεριφοράς από το σύστηματος (αυτοματοποιήση μιας χειροκίνητης διεργασίας ή δημιουργία μιας καινούριας λειτουργίας). Έχει να κάνει με αντικείμενα ή οντότητες, την κατάτασταση την οποία βρίσκονται, τις προυποθέσεις που χρειάζονται για να αλλάξει η κατάσταση τους καθώς και τις συναρτήσεις που θα αναλάβουνε να διεκπερώσουνε αυτή την αλλαγή.

Σε αυτό το στάδιο, δεν αναφερόμαστε σε υλοποιήση του συστήματος ακόμα, αλλά μιλάμε για μια πιο αφαιρετική έννοια, αποκρύπτοντας τις λεπτομέρειες. Σκοπός αυτής της φάσης είναι η συλλογή των θέλω του πελάτη και για αυτό συγκεντρωνόμαστε στον πελάτη και το πρόβλημα παρά στην λύση. Μας ενδιαφέρει το τί θέλουμε να κάνει το σύστημα και όχι το πώς θα γίνει αυτό.

Αυτή την διαδικασία την αναλαμβάνουμε ειδικοί αναλυτές, μέλη της ομάδας και αποτελείται από τις εξής διεργασίες :

i. Εκμαίευση Απαιτήσεων. Είναι το πιο σημαντικό μέρος της όλης διαδικασίας. Αρκετές τεχνικές χρησιμοποιούνται ώστε να βοηθήσει η ομάδα των πελάτη να περιγράψει το τί ακριβώς θέλει από το καινούριο σύστημα. Εκτός από τον πελάτη, οι τελικοί χρήστες, ειδικοί, δικηγόροι, μηχανικοί λογισμικού μπορεί να λάβουν μέρος σε αυτή την διεργασία. Κάθε ένας από τους παραπάνω έχει και διαφορετική εικόνα για το τελικο σύστημα και για αυτό μπορεί να προκύψουνε αντικρουόμενες απαιτήσεις, οι οποίες επιλύνονται από τους αναλυτές της ομάδας. Το κυριότερο εργαλείο για αυτή την διεργασία είναι οι συνεντεύξεις με τα αναφερόμενα άτομα, ενώ σημαντικά είναι και η ανάλυση της οποιαδήποτε τεκμηρίωσης

(documentation) υπάρχει ήδη και η παρατήρηση και συλλογή πληροφοριών από το υπάρχον σύστημα.

ii. Ανάλυση των απαιτήσεων. Σε αυτή την διεργασία, οι αναλυτές μοντελοποιούνε τις απαιτήσεις του χρήστη χρησιμοποιώντας ειδικά εργαλεία, όπως τα μονέλτα ER ή γλώσσες όπως η UML, ώστε να γίνει κατανοητή η λειτουργία που ζήτησε ο πελάτης καθώς και να δημιουργηθούνε σχήματα και μοντέλα που θα βοηθήσουνε στην επικοινωνία μεταξύ των αναλυτών και του πελάτη.

iii. Προδιαγραφές Απαιτήσεων. Σε αυτή την διεργασία, που είναι περίπου ίδια με την προηγούμενη, οι προδιαγραφές μοντελοποιούνται υπό την οπτική των προγραμματιστών. Αντικείμενα, διεπαφές τιμές εισόδου είναι μερικές έννοιες από αυτές που αναλύονται σε αυτό το στάδιο καθώς και το τί πρέπει να κάνουνε οι προγραμματιστές.

iv. Επικύρωση και Επαλήθευση. Στο τελικό στάδιο σε συνεργασία με τον πελάτη επικυρώνονται οι απαιτήσεις που τεθήκανε και διασταυρώνονται οι απαιτήσεις με τις τελικές προδιαγραφές. Αυτό δημιουργεί και μια δέσμευση για το μέλλον.

Το παραδοτέο αυτής της διαδικασίας είναι το έγγραφο Software Requirements Specification (SRS), στο οποίο θα βασιστεί η ομάδα των προγραμματιστών για να αναπτύξει το λογισμικό.

1.2.2 Σχεδιασμός Συστήματος

Σε αυτή την φάση η ομάδα έχει πολύ καλή κατανόηση του προβλήματος του πελάτη. Τα αποτελέσματα της προηγούμενης διαδικασίας χρησιμοποιούνται για να γίνει ο σχεδιασμός του συστήματος δηλαδή η δημιουργική διεργασία της αναζήτησης λύσεων για το πώς να υλοποιηθεί το λογισμικό.

Στο στάδιο αυτό, περιγράφεται ο τρόπος που θα υλοποιηθεί η κάθε λειτουργία του συστήματος. Επίσης, καθορίζονταιτο ποιά εργαλεία και τεχνολογίες θα χρησιμοποιηθούνε για την ανάπτυξη. Αυτή η περιγραφή πρέπει να είναι άκρως αναλυτική ώστε να τροφοδοτήσει την ομάδα των προγραμματιστών με τα κατάλληλα εφόδια για την δημιουργία του κώδικα.

Ο σχεδιασμός ενός συστήματος είναι επαναλαμβανόμενη διαδικασία, στην οποία οι σχεδιαστές αναλύουνε τις απαιτήσεις, προτείνουνε λύσεις στους πελάτες, προσαρμόζουνε ξανά τις απαιτήσεις και παίρνουνε πληροφορίες από τους προγραμματιστές για το αν είναι εφικτές οι λύσεις.

Αποτελείται από τις εξής διεργασίες :

i. Δημιουργία σχεδιαστικών προτύπων. Προτείνονται αρχικά πρότυπα για το πώς θα σχεδιαστεί το σύστημα
ii. Ανάλυση αυτών των προτύπων. Αναλύεται το κατά πόσο αυτά τα πρότυπα αρχιτεκτονικής ανταποκρίνονται στις απαιτήσεις.
iii. Βελτίωση των προτύπων. Βάση της προηγούμενης αναφοράς.

Κάθε στάδιο της δημιουργίας προτύπων αρχιτεκτονικής συνοδεύεται από έντυπα και τεκμηρίωση για την κάθε επιλογή.

Το αποτέλεσμα αυτού του βήματος είναι το έγγραφο Software Architecture Document (SAD). Το έγγραφο αυτό θα καθοδηγήσει την ομάδα προγραμματιστών στην υλοποιήση του προγραμμάτος, αφού αποτελεί μία υψηλού επιπέδου ανάλυση για το πώς θα γίνει η ανάπτυξη.

Καλό είναι να αναφερθεί πως υπάρχουνε διαφορετικές προοπτικές σχεδίασης σε αυτό το στάδιο όπως:

- Λογική προοπτική. Αναγνώριση στοιχείων του συστήματος και των σχέσεων μεταξύ τους.
- Φυσική προοπτική. Μετάφραση των υποσυστημάτων σε συγκεκριμένο τεχνικό σχέδιο για το νέο σύστημα.
- Προοπτική εφαρμογής. Ταυτοποίηση μονάδων κώδικα όπως αντικείμενα σε αρχεία πηγαίου κώδικα.
- Προοπτική Εξάρτησης. Το πώς αλληλοσυνδέονται μεταξύ τους τα επιμέρους υποσυστήματα.

1.2.3 Σχεδιασμός και συγγραφή προγραμμάτων

Σε αυτό το στάδιο, η ομάδα προγραμματιστών έχει τα απαιτούμενα έγγραφα ώστε να προχωρήσει στην ανάπτυξη του λογισμικού.

Το Software Architecture Document, περιλαμβάνει το ποιές τεχνολογίες θα χρησιμοποιηθούνε και είναι στα χέρια των προγραμματιστών με ποιόν τρόπο θα χρησιμοποιηθούνε.

Αναλύεται πλέον το ποιά γλώσσα προγραμματισμού θα χρησιμοποιηθεί, αν γίνει χρήση επιπλέον βιλβιοθηκών, αν χρειάζεται να γίνει έρευνα για δημιουργία καινούριων αλγορίθμων κ.ά.

Πολλά εργαλεία υπάρχουνε για την διευκόλυνση των προγραμματιστών, η χρήση της UML (διαγράμματα κλάσσης), design patterns για την ορθή συγγραφή κώδικα είναι μερικά από αυτά. (Sommerville, 2011)

1.2.4 Έλεγχος λογισμικού

Σε αυτό το στάδιο, το λογισμικό έχει αναπτυχθεί και πρέπει να βρεθούνε τυχόν αστοχίες και λάθη.

Ασυμφωνία ανάμεσα στις απαιτήσεις και στο τελικό προϊόν μπορεί να εντωπιστούνε σε αυτό το στάδιο.

Αρκετές τεχνικές υπάρχουνε για τον ενδελεχή έλεγχο και δοκιμή του συστήματος και θα αναλυθούνε στο επόμενο κεφάλαιο.

1.2.5 Παράδοση και Συντήρηση

Υπό κανονικές συνθήκες το λογισμικό μετά την φάση του ελέγχου λειτουργεί όπως πρέπει και σύμφωνα με τις αρχικές απαιτήσεις. Συνήθως όμως, με την καθημερινή του χρήση ανκαλύπτονται καινούρια προβλήματα τα οποία απαιτούν συνεχή αντιμετώπιση, δημιουργώντας καινούριες εκδόσεις ή αναβαθμίσεις.

Οπότε καταλαβαίνει κανείς, ότι η ανάπτυξη του λογισμικού δεν σταματάει με την παράδοση και εγκατάσταση του, αλλά συνεχίζεται. Καινούρια κενά ασφαλείας μπορεί να χρήζουνε αντιμετώπισης, νέες λειτουργίες μπορεί να προσθεθούν στην πορεία κ.ά

Τέλος, με την παράδοση στον πελάτη, παραδίδεται αναλυτική τεκμηρίωση η οποία θα βοηθήσει στην αντιμετώπιση μετέπειτα προβλημάτων ενώ γίνεται και παράδοση εγχειριδιού χρήσης.

1.3 Μοντελοποίηση της Ανάπτυξης Λογισμικού

Δημιουργώντας ένα μοντέλο που περιγράφει ακριβώς τα βήματα, την σειρά και την διαδικασία που πρέπει να ακολουθήθει για την ανάπτυξη του λογισμικού οφελεί την ομάδα για τους εξής λόγους:

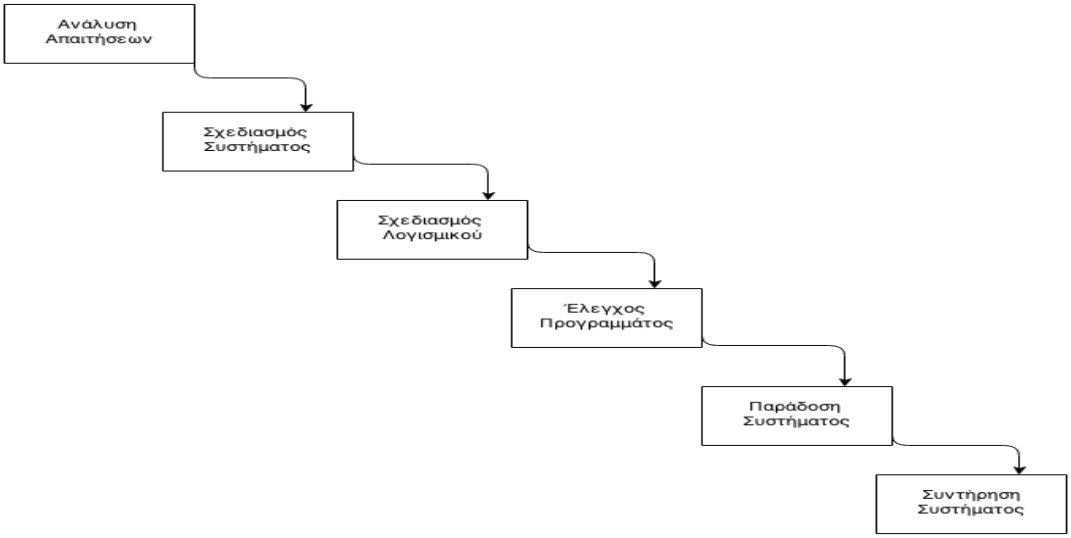
- Βοηθάει την ομάδα να βρει τυχόν ασυνέπειες, πλεονασμούς και περιορισμούς στην διαδικασία και στα επιμέρους βήματα της. Καθώς αυτά τα προβλήματα καταγράφονται και διορθώνονται, η διαδικασία γίνεται πιο αποδοτική.
- Βοηθάει την ομάδα να έχει αντίληψη του έργου εξαρχής, αφού γνωρίζει ποια είναι τα βήματα, την σειρά τους και τις διαδικασίες που χρειάζονται.
- Βοηθάει στην αυξημένη παραγωγικότητα των επιμέρους ομάδων.
- Οδηγεί την ομάδα στο με ποιά σειρά να εκτελεστούνε τα βήματα δίνοντας έτσι περισσότερο έλεγχο και εξαρχής χρονικό σχεδιασμό.

Κάθε μοντέλο λαμβάνει σαν είσοδο τις απαιτήσεις και του συστήματος και παράγει το τελικό λογισμικό.

1.4 Μοντέλο Καταρράκτη

Πολλά μοντέλα έχουνε δημιουργηθεί όλα αυτά τα χρόνια, αλλά το πιο βασικό είναι αυτό του καταρράκτη. Είναι από τα πρώτα μοντέλα που αναπτυχθήκανε με σκοπό την καθοδήγηση της ομάδας στην ανάπτυξη λογισμικού.

Τα στάδια που αποτελούνε το μοντέλο παρουσιάζονται στο σχήμα 1.1 .



Σχήμα 1-1 Μοντέλο Καταρράκτη

1.3.1 Χαρακτηριστικά

Είναι το παλαιότερο που υπάρχει αφού χρονολογείται από το 1970.

Το μοντέλο αποτυπώνεται με φθίνουσα κλήση, υποδηλώνοντας πως κάθε φάση έχει σαν είσοδο της το αποτέλεσμα της προηγούμενης, και για να ξεκινήσει μία φάση πρώτα πρέπει να έχει ολοκληρωθεί η προηγούμενη της.

Στο τέλος κάθε φάσης παράγονται αρκετά έντυπα και έγγραφα τεκμηρίωσης.

1.3.2 Πλεονεκτήματα

Το συγκεκριμένο μοντέλο έχει τα εξής πλεονεκτήματα :

- Είναι απλό, διαχειρίσιμο και εύκολα κατανοητό, ακόμα και για κάποιον που ασχολείται πρώτη φορά με την ανάπτυξη λογισμικού.
- Παράγει αρκετά έγγραφα με την ολοκλήρωση της κάθε φάσης.
- Είναι κατάλληλο για μικρά έργα (μερικών εβδομάδων).
- Είναι κατάλληλο για έργα, των οποίων οι απαιτήσεις είναι σίγουρο πως δεν θα αλλάξουνε κατά την διάρκεια.
- Εύκολο να υλοποιηθεί.

- Βοηθάει στην εύκολη κατανομή της εργασίας ανάμεσα στα μέλη της ομάδας.
- Κάθε φάση παράγει ένα σαφώς καθορισμένο παραδοτέο.

1.3.3 Μειονεκτήματα

Παρά τα όσα πλεονεκτήματα το μοντέλο αυτό έχει κατηγορηθεί ότι έχει αρκετά σημαντικά μειονεκτήματα :

- Αλληλοεπικάλυψη φάσεων.
- Αλλαγή στις απαιτήσεις αργά στην ανάπτυξη του λογισμικού, αποβαίνει μοιραίο, αφού πρέπει να ξεκινήσουμε σχεδόν από την αρχή.
- Συνήθως οι απαιτήσεις όντως αλλάζουνε, οπότε είναι δύσκολη εξαρχής δέσμευση και δημιουργία του εγγράφου SRS.
- Δύσκολο να γίνει εκτίμηση χρόνου και κόστους.
- Η διαχείριση ρίσκων δεν είναι μέρος του μοντέλου.
- Για να φτάσει το λογισμικό στην φάση της δοκιμής, πρέπει να είναι σχεδόν έτοιμο. Η δοκιμή μπορεί να ανακαλύψει μεγάλα σφαλμάτων, κάτι το οποίο είναι χρονοβόρο να διορθωθεί τόσο αργά.
- Η γραμμικότητα του μοντέλου σπάνια συναντάται σε πραγματικά έργα.
- Ο πελάτης αργεί να πάρει μια λειτουργική εικόνα του συστήματος.

1.3.4. Σύνοψη του Μοντέλου

Παρά τα όσα θετικά μας δίνει η χρήση του, τα αρνητικά το καθιστούνε πλέον ακατάλληλο για διαχείριση ανάπτυξης λογισμικού στις μέρες μας.

Η δοκιμή του λογισμικού στο τελευταίο σχεδόν στάδιο είναι από τα πιο μεγάλα μειονεκτήματα, αφού είναι σίγουρο πως αρκετά σφάλματα θα παρουσιαστούνε σε αυτό το στάδιο, και η επιδιόρθωση τους θα καθυστερήσει την τελική παράδοση. Ενώ η προσθήκη απαιτήσεων μετά την πρώτη φάση, είναι κάτι πολύπλοκο.

1.4 Επίλογος

Σε αυτό το κεφάλαιο έγινε μία εισαγωγή στην ανάπτυξη του λογισμικού και παρουσιάστηκε το πιο βασικό μοντέλο, του καταρράκτη.

Υπάρχουνε και άλλα μοντέλα, βελτίωσης του καταρράκτη, όπως το μοντέλο V, το Spiral μοντέλο, το Prototype μοντέλο, τα οποία δεν έχουνε αναλυθεί.

Παρατηρήσαμε ότι η δοκιμή του λογισμικού παίζει πολύ σημαντικό ρόλο, και δεν πρέπει να γίνεται στο τέλος της ανάπτυξης του λογισμικού. Υπάρχουνε αρκετές τεχνικές σχετικά με την δοκιμη οι οποίες θα αναλυθούνε αργότερα.

Τέλος, να σημειωθεί πως έχουνε δημιουργηθεί καινούριες τεχνικές ανάπτυξης λογισμικού με στόχο την ταχεία ανάπτυξη του και την εισαγωγή της δοκιμής σε κάθε φάση. Αυτές οι τεχνικές θα συζητηθούνε στο κεφάλαιο 3.

Κεφάλαιο 2 Έλεγχος Λογισμικού

Ο έλεγχος (testing) του λογισμικού είναι αναπόσπαστη διαδικασία της ολικής ανάπτυξης του. Υπάρχουνε αρκετές τεχνικές και μέθοδοι για τον αποτελεσματικό έλεγχο, ο οποίος βοηθάει στην εξάλειψη όσο το δυνατόν περισσοτέρων σφαλμάτων.

2.1 Ορισμοί

“ Ο έλεγχος του λογισμικού είναι η διαδικασία της ανάλυσης και παρακολούθησης του λογισμικού ή μέρους αυτού υπό συγκεκριμένες συνθήκες και τιμές εισόδου, με σκοπό την συλλογή των αποτελεσμάτων εξόδου και σύγκριση αυτών με τις αρχικές απαιτήσεις (requirements)” IEEE definition

Στον πυρήνα του ο έλεγχος είναι η διαδικασία αξιολόγησης του τί παράγει το λογισμικό τροφοδοτούμενο με συγκεκριμένες τιμές εισόδου εν συγκρίση με το τί πρέπει να παράγει. Αυτό προϋποθέτει την γνώση των αρχικών απαιτήσεων και την δυνατότητα παρακολούθησης και αξιολόγησης των αποτελεσμάτων που παράγονται.

Προτού προχωρήσουμε καλό είναι να αναφερθούνε και να διαχωριστούνε οι έννοιες σφάλμα ή ελάτωμμα (fault), λάθος (error), βλάβη (failure).

- Λάθος. Είναι ανθρώπινο αποτέλεσμα. Όταν ο προγραμματιστής κάνει ένα λάθος στον κώδικα, το ονομάζουμε bug.Τα λάθη έχουν την συνήθεια να μεγεθύνονται κατά τη διάρκεια ανάπτυξης του λογισμικού. Ένα λάθος στις αρχικές απαιτήσεις θα εμφανιστεί πολύ μεγαλύτερο και πιο δύσκολα διαχειρίσιμο στην φάση του προγραμματισμού.
- Σφάλμα. Το σφάλμα είναι το αποτέλεσμα του λάθους ή η αναπαράσταση του λάθους στο σύστημα. Υπάρχουν δύο είδη σφαλμάτων:
    - Παράλειψης : Εμφανίζεται όταν αποτυγχάνουμε ή δεν δίνεται η δυνατότητα να βάλουμε τις σωστές τιμές εισόδου, με αποτέλεσμα να ενεργοποιείται το λάθος.
    - Εσκεμμένα : Εμφανίζεται όταν εσκεμμένα βάζουμε λάθος τιμές εισόδου.

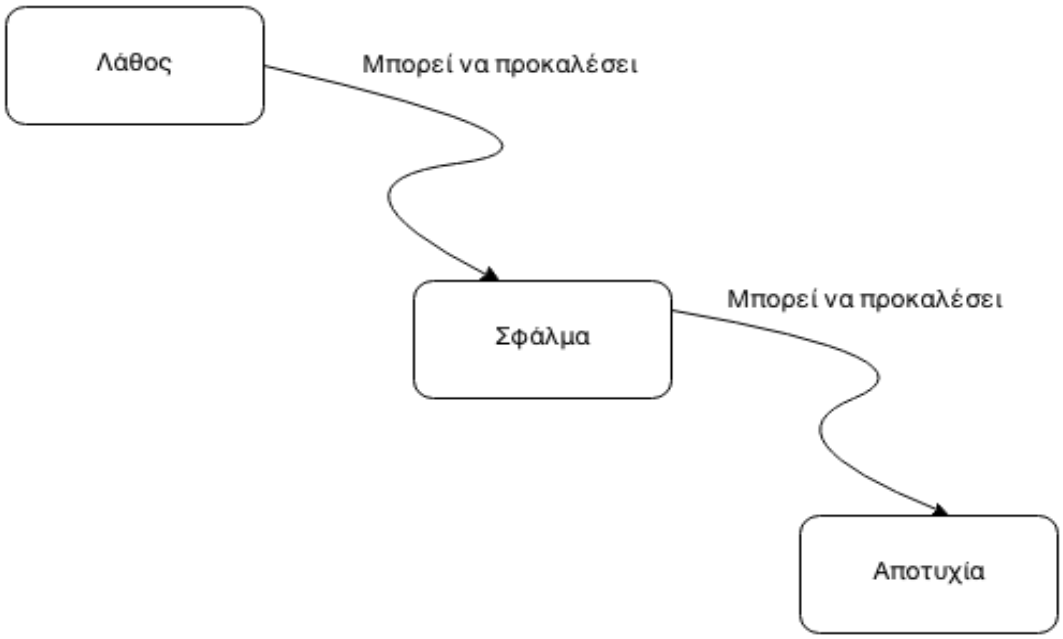
  Από τα δύο είδη σφαλμάτων τα παράλειψης είναι πιο δύσκολα να ανιχνευτούν.

- Αποτυχία. Η αποτυχία εμφανίζεται όταν εκτελείται ένα σφάλμα. Μπορεί να είναι αποτέλεσμα ελλιπών ή λάθος αρχικών απαιτήσεων, απαιτήσεων οι οποίες δεν μπορούνε να υλοποιηθούνε ενώ και το ίδιο το σύστημα στο οποίο αναπτύσσεται το λογισμικό μπορεί να περιέχει λάθη που θα οδηγήσουνε σε αποτυχία.

Ανεξαρτήτως της δυνατότητας συγγραφής ορθών προγραμμάτων, είναι φανερό από την πληθώρα διαφορετικών πηγών σφαλμάτων, πως είναι απαραίτητο να εξασφαλίστει ότι τα επιμέρους υποσυστήματα του λογισμικού μας είναι σωστά προγραμματισμένα χωρίς λάθη.

Σύμφωνα με τον Dijkstra, ο έλεγχος μας δίνει μόνο την παρουσία κάποιων λαθών και όχι την απουσία αυτών.

Παρακάτω παρουσιάζεετα η σχέση μεταξύ των 3 τους.



Σχήμα 2-1 Σχέση μεταξύ Λάθους-Σφάλματος-Αποτυχίας

Υπάρχουνε διαφορετικά είδη λαθών τα οποία πρέπει να αντιμετωπίσει η ομάδα, μερικά από αυτά είναι :

- Αλγοριθμικά λάθη. Συμβαίνουνε όταν ο σχεδιασμός του αλγορίθμου που χρειάζεται να εκτελείται για την παραγωγή αποτελέσματος, εμπεριέχει λάθη. Είναι εύκολα να εντοπιστούν, τροφοδοτώντας το πρόγραμμα με προκαθορισμένες τιμές εισόδου και γνωστές τιμές εξόδου.
- Λάθη ακρίβειας και υπολογισμών. Συμβαίνουνε, όταν ένας μαθηματικός τύπος δεν έχει υλοποιηθεί σωστά από τον προγραμματιστή ή όταν έχουμε μείωση ακρίβειας λόγω μη σωστής χρήσης τύπων(int αντί για float, απευθείας αφαίρεση ή σύγκριση δύο float μεταβλητών).
- Λάθη συγχρονισμού. Ιδίως στις εφαρμογές πραγματικού χρόνου είναι σημαντικό τα επιμέρους γεγονότα να ακολουθούνε πιστά τον προκαθορισμένο χρόνο εκτέλεσης.

Τέλος τα σφάλματα χωρίζονται σε δύο κατηγορίες, τί είδος είναι και το πόσο σοβαρό ή καταστροφικό είναι το αποτέλεσμα της παραγωγής του.

2.2 Περιπτώσεις Ελέγχου Χρήσης (Test cases)

Ο έλεγχος έχει να κάνει με την εύρεση όλων των παραπάνω προβλημάτων στο λογισμικό. Το πιο σημαντικό στοιχείο του είναι η δημιουργία περιπτώσεων χρήσης ελέγχου (test cases). (Copeland, 2004)

Γίνεται κατανοητό πως είναι αδύνατον να ελέγξουμε το λογισμικό μας για κάθε τιμή εισόδου, για αυτό σχεδιάζονται οι περιπτώσεις ελέγχου, ώστε να περιορίσουμε και να ομαδοποιήσουμε διάφορα εύρη τιμών.

Κάθε περίπτωση ελέγχου έχει ξεχωριστή ταυτότητα και είναι συνδεδεμένη με μια προκαθορισμένη συμπεριφορά του λογισμικού.

Για να είναι αποτελεσματικές οι περιπτώσεις ελέγχου, πρέπει να μελετηθούνε και να σχεδιαστούνε με προσοχή. Για αρκετούς αυτό το κομμάτι είναι το ίδιο δύσκολο και σημαντικό όσο του σχεδιασμού του αρχικού λογισμικού.

Κάθε μία περίπτωση αποτελείται από :

- Τιμές εισόδου. Οι πιο συνηθισμένες είναι οι τιμές που εισάγει ο χρήστης από το πληκτρολόγιο, παρόλα αυτά υπάρχουνε και άλλες πηγές, όπως μία μέθοδος που στέλνει δεδομένα σε μία άλλη.
- Αναμενόμενες τιμές εξόδου. Συνήθως θεωρούνται τα αποτελέσματα που εμφανίζονται στην οθόνη μετά την εκτέλεση του προγραμμάτος. Όπως είδαμε όμως μπορεί να υπάρχουνε και άλλες τέτοιες τιμές.
- Ροή εκτέλεσης. Υπάρχουνε δύο διαφορετικά είδη που αφορούνε τον τρόπο ή ροή της εκτέλεσης:

    - Αλληλοεξαρτώμενες περιπτώσεις. Κάθε μία περίπτωση, μετά την εκτέλεση της, αφήνει το σύστημα σε κατάσταση τέτοια ώστε να δεχτεί την επόμενη περίπτωση (πχ. βάσεις δεδομένων, εισαγωγή στοιχείων, διάβασμα στοιχείων, ανανέωση..). Το θετικό είναι πως κάθε περίπτωση είναι πιο μικρή και πιο απλή. Ενώ το αρνητικό, πως σε περίπτωση που ένας αρχικός έλεγχος αποτύχει αυτόματα αποτυγχάνουμε και οι υπόλοιποι.

- Ανεξάρτητες περιπτώσεις. Κάθε περίπτωση είναι εντελώς ανεξάρτητη από τις υπόλοιπες. Το θετικό είναι ότι πληθώρα περιπτώσεων μπορεί να εκτελεστεί με οποιαδήποτε σειρά. Το αρνητικό είναι, ότι αυτές οι περιπτώσεις είναι πιο μεγάλες και πολύπλοκες.

Οι τιμές εισόδου και οι αναμενόμενες τιμές εξόδου είναι τα πιο σημαντικά μέρη της περίπτωσης χρήσης. Στον σχεδιασμό ολοκληρωμένων προγραμμάτων ελέγχου, οι τιμές εξόδου συνήθως δημιουργούνται από ένα πρόγραμμα ή διαδικασία που ονομάζεται oracle. Υπάρχουνε τέσσερα είδη τέτοιων συστημάτων :

- Kiddie Oracles. Είναι το πρώτο στάδιο, εκτελείται το πρόγραμμα και παρατηρούμε τις τιμές εξόδου. Σε περίπτωση που φαίνονται σωστά τα αποτελέσματα σημαίνει πως είναι κιόλας.
- Περιπτώσεις χρήσης οπισθοδρόμησης (regression test). Σύγκριση των αποτελεσμάτων με αυτά της προηγούμενης έκδοσης του λογισμικού.
- Επικυρωμένα Δεδομένα. Εκτέλεση του προγράμματος και σύγκριση αποτελεσμάτων με ήδη γνωστά και επικυρωμένα αποτελέσματα, μαθηματικές φόρμουλες, τύποι.
- Υπάρχον πρόγραμμα. Σύγκριση των αποτελεσμάτων με αυτά της μιας άλλης έκδοσης του λογισμικού. (Copeland, 2004)

2.3 Προσεγγίσεις Ελέγχου

Υπάρχουνε 2 διαφορετικές προσεγγίσεις στον έλεγχο του λογιμισικού. Η δυναμική και η στατική. Η διαφορά έγγειται στην εκτέλεση ή όχι του λογισμικού. Πρώτα αναλύεται η δυναμική (εκτελείται ο κώδικας) και ακολουθεί η στατική προσέγγιση (δεν εκτελείται ο κώδικας).

Στην δυναμική προσσέγγιση έχουνε επικρατήσει 2 στρατηγικές ελέγχου οι οποίες διαχωρίζονται βάση της πρόσβασης ή όχι στον πηγαίο κώδικα.

2.3.1 Στρατηγική Μαύρου Κουτιού (Black Box Testing)

Αυτό το είδος ελέγχου διεξάγει τα απαραίτητα βήματα ελέγχου βάση μόνο των αρχικών απαιτήσεων και προδιαγραφών. Δεν υπάρχει πρόσβαση στον πηγαίο κώδικα και δεν απαιτείται γνώση των εσωτερικών μονοπατιών εκτέλεσης και των διαφορετικών εσωτερικών δομών του λογισμικού. (Shari Lawrence Pfleeger, 2010)

Οι περιπτώσεις χρήσης και οι τιμές εισόδου σχεδιάζονται βάση των αρχικών προδιαγραφών. Η υλοποίηση των μεθόδων του πηγαίου κώδικα δεν είναι γνωστή.

Το λογισμικό υπό εξέταση θεωρείται πως είναι ένα μαύρο κουτί, που απλά δέχεται τιμές εισόδου και παράγει τιμές εξόδου, τις οποίες πρέπει να αναλύσει ο δοκιμαστής και να διαπιστώσει άμα είναι σύμφωνες με τις αρχικές απαιτήσεις. Ακολουθεί το σχήμα αυτής της μέθοδο.

Σχήμα 2-0-2 Στρατηγική Μαύρου Κουτιού

Η διαδικασία που ακολουθείται για την υλοποίηση αυτής της στρατηγικής είναι η εξής :

I. Οι αρχικές απαιτήσεις ή προδιαγραφές αναλύονται.
II. Έγκυρες τιμές εισόδου επιλέγονται, βάση των απαιτήσεων, ώστε να αποδειχτεί πως το λογισμικό ή μέρους αυτού που ελέγχεται παράγει τα σωστά αποτελέσματα.
III. Τα αναμενόμενα αποτελέσματα καθορίζονται.
IV. Περιπτώσεις ελέγχου χρήσης δημιουργούνται βάση των επιλεγμένων τιμών εισόδου και εκτελούνται.
V. Τα πραγματικά αποτελέσματα συγκρίνονται με τα αναμενόμενα.
VI. Αποφασίζεται αν το λογισμικό μας ανταποκρίνεται στις αρχικές απαιτήσεις.

Σχήμα 2-2 - Στρατηγική Μαύρου Κουτιού

Η στρατηγική αυτή εφαρμόζεται σε όλα τα επίπεδα ελέγχου του συστήματος, τα οποία θα αναλυθούνε παρακάτω.

Τα θετικά της συγκεκριμένης στρατηγικής είναι ότι καθοδηγεί τον δοκιμαστή (tester), να σχεδιάσει περιπτώσεις χρήσης οι οποίες είναι αποτελεσματικές και αποδοτικές στην εύρεση σφαλμάτων.

Ο δοκιμαστής δεν χρειάζεται να έχει ιδιαίτερες γνώσεις προγραμματισμού, αφού ο σχεδιασμός των περιπτώσεων χρήσης βασίζεται εξολοκλήρου στις αρχικές απαιτήσεις.

Η στρατηγική αυτή βοηθάει στο να ξεκαθαρίσουνε οποιεσδήποτε ασάφειες στις προδιαγραφές.

Υπάρχουνε και αρνητικά της χρήσης αυτής της. Ο δοκιμαστής δεν μπορεί ποτέ να είναι σίγουρος σε τί ποσοστό έχει ελέγξει το λογισμικό. Όσο αποτελεσματικές και να είναι οι περιπτώσεις χρήσης, κάποια εσωτερικά μονοπάτια εκτέλεσης είναι εξαιρετικά δύσκολο να εξεταστούνε. Για παράδειγμα το κομμάτι κώδικα 1 που ακολουθεί είναι σχεδόν αδύνατο να ελεγχθεί με την παραπάνω στρατηγική.

```
if (name == "Dimitris" && employeeId == 1245 && currentSalary == 1400 && dayOfWeek == "Saturday")
{
    sendCheckToEmployee(1245, 3000);
}
```

Κώδικας 1 - Μειονέκτημα Μαύρου Κουτιού

Για την δημιουργία των περιπτώσεων χρήσης πρέπει να είναι υπάρχουν σαφή αρχικές προδιαγραφές του λογισμικού.

2.3.1.1 Τεχνικές Ελέγχου Μαύρου Κουτιού

Η στρατηγική μαύρου κουτιού μπορεί να υλοποιηθεί χρησιμοποιώντας διαφορετικές τεχνικές. Σκοπός αυτών των τεχνικών είναι να περιοριστεί ο αριθμός των περιπτώσεων ελέγχου χρήσης που χρειάζεται να δημιουργηθούνε.

Μία πρώτη και πρόχειρη προσέγγιση θα ήτανε να ελεγχθεί το λογισμικό με κάθε πιθανό συνδυασμό τιμών εισόδου. Αυτό καταλαβαίνουμε πως δεν είναι εφικτό, αφού θα έπαιρνε πάρα πολύ χρόνο για να ολοκληρωθεί η διαδικασία. Αυτή η

τεχνική ονομάζεται εξοντωτικός έλεγχος (exhaustive testing) και δεν χρησιμοποιείται.

Το επόμενο βήμα είναι η κατηγοριοποίηση και ομαδοποίηση των τιμών εισόδου. Σκοπός είναι να μειωθούνε οι περιπτώσεις χρήσης ενώ παράλληλα να έχουμε έναν ικανοποιητικό βαθμό ελέγχου. Αυτή η τεχνική ονομάζεται ισοδύναμη κλάσση.

Κάθε ισοδύναμη κλάσση αποτελείται από ένα εύρος τιμών εισόδου, οι οποίες παράγουνε το ίδιο αποτέλεσμα όταν εκτελούνται από το λογισμικό.Κάθε συνθήκη εισόδου, από τις απαιτήσεις, αναλύεται και διαχωρίζεται σε δύο ή περισσότερες κλάσεις.Για παράδειγμα η αρχική απαίτηση ορίζει το εξής :

- Ηλικία πρόσληψης :
-
    - 0 – 16. Όχι πρόσληψη
    - 16 – 18. Πρόσληψη με μειωμένο ωράριο
    - 18 – 45. Κανονική πρόσληψη

Παρατηρούμε ότι οι τιμές εισόδου είναι κατηγοροποιημένες. Για κάθε μία συνθήκη μπορύμε να δημιουργήσουμε ξεχωριστές ισοδύναμες κλάσεις. Αντί δηλαδή να δοκιμάσουμε κάθε τιμή από 0 – 16, μπορούμε να επιλέξουμε μία μόνο τιμή σε αυτό το εύρος και να αναμένουμε το ίδιο αποτέλεσμα και για τις υπόλοιπες τιμές του εύρους αυτού.

Τα βήματα δηλαδή είναι. Πρώτα αναγνωρίζουμε την κλάσση ισοδυναμίας. Έπειτα για κάθε κλάσση δημιουργούμε μία περίπτωση χρήσης. Εκτελούμε τις περιπτώσεις χρήσης και καταγράφουμε το αποτέλεσμα.

Τα όρια και το εύρος των τιμών εξαρτώνται από τις αρχικές απαιτήσεις. Μπορεί να είναι διακριτές τιμές (είναι ανήλικος, δεν είναι ανήλικος), εύρος τιμών και συνδυασμός των δύο. Όλα εξαρτώνται από τις εκάστοτε απαιτήσεις.

Αυτή η τεχνική χρησιμοποιείται σε λογισμικά των οποίων οι τιμές εισόδου είναι σε μορφή εύρους και κάθε τιμή σε αυτό το εύρος επεξεργάζεται το ίδιο από το λογισμικό.

Αν και η τεχνική της ισοδύναμης κλάσης είναι από τις πιο βασικές, εμπεριέχει προβλήματα τα οποία μπορεί να οδηγήσουνε σε εσφαλμένα αποτελέσματα. Για αυτόν τον λόγο η τεχνική, ελέγχου οριακών τιμών δημιουργήθηκε.

Όπως μπορούμε να παρατηρήσουμε στο παραπάνω παράδειγμα, η τιμή 16 και η τιμή 18 ανήκουνε σε δύο διαφορετικές κλάσεις ισοδυναμίας. Είναι ευρέως γνωστό, ότι πολλά σφάλματα κρύβονται στις οριακές τιμές, σε κάθε λογισμικό,και για αυτό πρέπει να ελέγχονται διεξοδικά.

Η τεχνική της ισοδύναμης κλάσης μας οδηγεί στην τεχνική ελέγχου οριακών τιμών. Για κάθε ισοδύναμη κλάση αναγνωρίζονται οι οριακές τιμές, και για κάθε οριακή τιμή δημιουργούνται τρεις περιπτώσεις χρήσης, μία με τιμή εισόδου την οριακή τιμή, μία με τιμή εισόδου την οριακή τιμή + 1, και μία με τιμή εισόδου την οριακή τιμή -1. Το αποτέλεσμα είναι να ελέγχονται οι τιμές, γύρω από την οριακή, οι οποίες είναι αυτές που προκαλούνε τα περισσότερα λάθη.

Αυτή τη τεχνική εφαρμόζεται μετά την τεχνική ισοδύναμης κλάσης.

Υπάρχουνε αρκετές ακόμα τεχνικές μαύρου κουτιού, οι δύο προηγούμενες είναι οι πιο διάσημες και πιο χρησιμοποιημένες. Μερικές από τις υπόλοιπες είναι :

- Πίνακες Αποφάσεων. Βοηθάει στην καταγραφή και έλεγχο πολύπλοκων αρχικών απαιτήσεων και κανόνων.
- Έλεγχος ανά ζεύγη. Χρησιμοποιείται όταν ο αριθμός των συνδυασμών που πρέπει να ελεγχθούνε είναι πολύ μεγάλος (έλεγχος του ίδιου λογισμικού σε διαφορετικούς browsers, με τον καθένα να πρέπει να δοκιμαστεί σε διαφορετικές εκδόσεις..) Αντί να δοκιμάζονται όλοι οι πιθανοί συνδυασμοί, χρησιμοποιείται αυτή η τεχνική.
- Έλεγχος μεταβατικής κατάστασης. Βοηθάει στο να ελεγχθούνε απαιτήσεις οι οποίες περιγράφουνε διαφορετικές καταστάσεις (states) του λογισμικού.

2.3.2 Στρατηγική Άσπρου Κουτιού (White Box Testing)

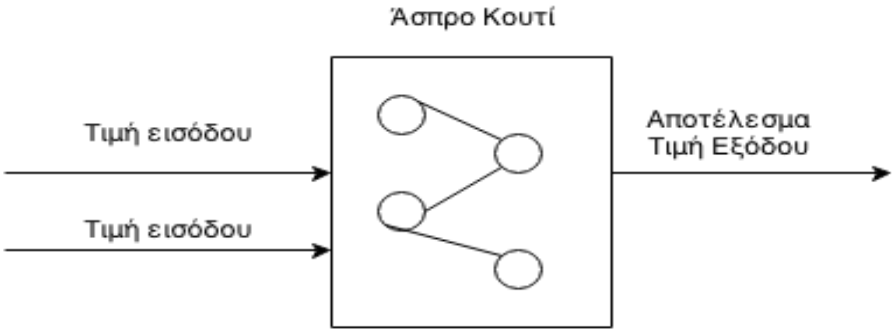
Αυτό το είδος ελέγχου διεξάγει τα απαραίτητα βήματα στηριζόμενο στα εσωτερικά μονοπάτια εκτέλεσης, την εσωτερική δομή και του πηγαίου κώδικα του λογισμικού. Απαιτεί περισσότερες προγραμματιστικές γνώσεις από τον δοκιμαστή, σε αντίθεση με την στρατηγική του μαύρου κουτιού και όπως

αναφέρθηκε η μεγάλη διαφορά των δύο είναι ότι έχει πρόσβαση στον πηγαίο κώδικα.

Όλα οι περιπτώσεις χρήσης ελέγχου σχεδιάζονται μετά την ανάλυση του πηγαίου κώδικα.

Το λογισμικό υπό εξέταση είναι ανοιχτό και τα εσωτερικά μονοπάτια εκτέλεσης

διαθέσιμα στον δοκιμαστή, όπως φαίνεται στο σχήμα



Σχήμα 2-3 Στρατηγική Άσπρου Κουτιού

Η διαδικασία που ακολουθείται για αυτή την στρατηγική είναι η εξής :

I. Η υλοποίηση του υπό εξέταση λογισμικού αναλύεται.
II. Ανακαλύπτονται τα διαφορετικά μονοπάτια εκτέλεσης.
III. Οι τιμές εισόδου διαλέγονται έτσι ώστε να εκτελεστούνε όσο το δυνατόν περισσότερα μονοπάτια εκτέλεσης. Επίσης, καθορίζονται τα αναμενόμενα αποτελέσματα.
IV. Οι έλεγχοι εκτελούνται.
V. Τα πραγματικά αποτελέσματα συγκρίνονται με τα αναμενόμενα
VI. Αποφασίζεται αν το λογισμικό λειτουργεί όπως πρέπει.

Όπως και ο έλεγχος μαύρου κουτιού, μπορεί να εφαρμοστεί σε όλα τα επίπεδα ελέγχου του συστήματος.

Τα θετικά αυτής της στρατηγικής είναι ότι δίνει την σιγουριά στον δοκιμαστή πως έχουνε αναγνωριστεί, εκτελεστεί και ελεγχθεί όλα τα διαφορετικά μονοπάτια εκτέλεσης του λογισμικού.

Επίσης, με αυτή την μέθοδο αναγνωρίζεται πιθανώς νεκρός κώδικας (κώδικας ο οποίος δεν εκτελείται ποτέ στο λογισμικό).

Παρά τα όποια θετικά έχει τέσσερα διακριτά αρνητικά. Πρώτον, τα διαφορετικά μονοπάτια εκτέλεσης μπορεί να είναι εκατοντάδες χιλιάδες ώστε να καθιστά αδύνατο να ελεγχθούνε όλα.

Δεύτερον, και ίσως το πιο σημαντικό, δεν εντοπίζει λάθη λογικής, μπορεί να ξεφύγει για παράδειγμα λάθος διαίρεσης με το 0. Το μονοπάτι εκτέλεσης δηλαδή, να ελεγχθεί αλλά για διαφορετικές τιμές και να περάσει τον έλεγχο.

Τρίτον, η μέθοδος αυτή θεωρεί ότι η ροή ελέγχου είναι σωστή εξαρχής, άρα δεν υπάρχει τρόπος να ελεγχθούνε μονοπάτια τα οποία λείπουνε.

Τέταρτον, όπως έχει αναφερθεί, ο δοκιμαστής χρειάζεται να έχει προηγμένες γνώσεις προγραμματισμού για να την αξιοποιήσει.

2.3.2.1 Τεχνικές Ελέγχου Άσπρου Κουτιού

Κατά κύριο λόγω δύο είναι οι τεχνικές που χρησιμοποιούνται για αυτή την στρατηγική. (Copeland, 2004)

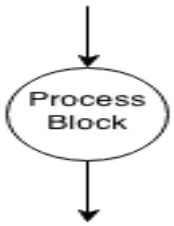
I. Ροή Ελέγχου (Control Flow Testing). Αυτή η τεχνική αναγνωρίζει τα μονοπάτια εκτέλεσης του λογισμικού ή μέρους αυτού και στη συνέχεια δημιουργούνται και εκτελούνται ελεγχόμενες περιπτώσεις χρήσης, με στόχο την κάλυψη αυτών των μονοπατιών.

   Μονοπάτι νοείται ως μια ακολουθία εντολών υπό εκτέλεση η οποία έχει αρχή και τέλος. Συμπεραίνουμε, ότι είναι αδύνατο να ελεγχθούνε όλα τα πιθανά μονοπάτια του λογισμικού.

   Το βασικό συστατικό ελέγχου της ροής είναι οι γράφοι ελέγχου ροής (Control Flow Graphs). Αυτοί οι γράφοι αποτυπώνουνε την εσωτερική δομή του λογισμικού. Κάθε ακολουθία εντολών μετατρέπεται σε γράφο. Οι κυριότεροι είναι :

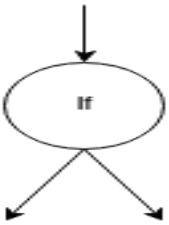
   - Process Blocks. Ο αναφερόμενος γράφος είναι μία αλληλουχία εντολών οι οποίες εκτελούνται σεριακά από την αρχή μέχρι το τέλος.

Μόνο ένα σημείο εισόδου και εξόδου επιτρέπονται. Με την αρχικοποίηση του block κάθε εντολή του εκτελείται. Ο γράφος παρουσιάζεται στο παρακάτω σχήμα.



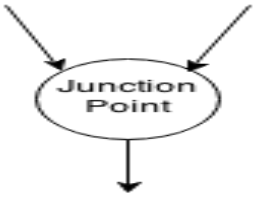
Σχήμα 2-4 Process Blocks Graph

- Decision Point. Ο αναφερόμενος γράφος αποτυπώνει την αλλαγή της ροής εκτέλεσης, λόγω μιας συνθήκης ελέγχου (if, switch). Ο γράφος αποτυπώνεται στο σχήμα.



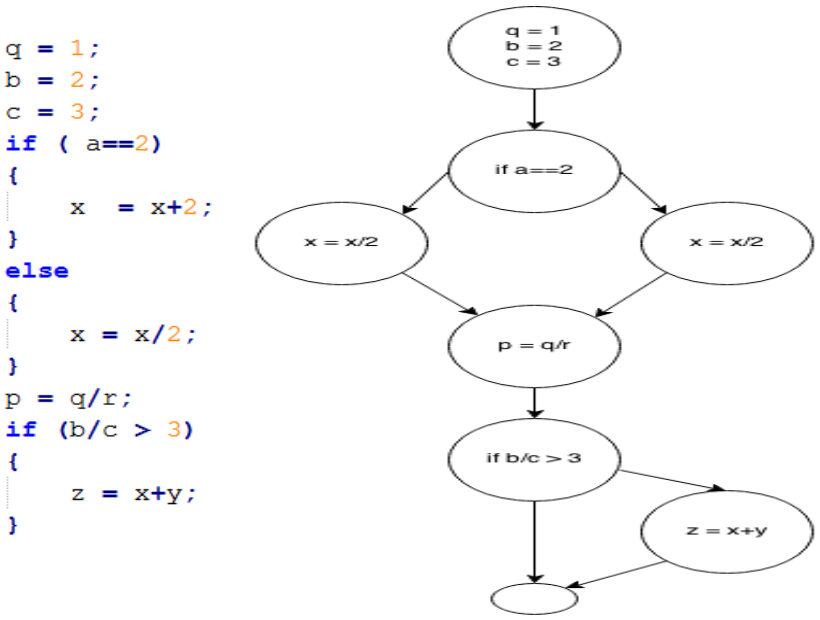
Σχήμα 2-5 Decision Point Graph

- Junction Point. Ο αναφερόμενος γράφος, είναι το σημείο ένωσης δύο διαφορετικών ροών εκτέλεσης. Ο γράφος παρουσιάζεται στο σχήμα.



Σχήμα 2-6- Junction Point Graph

Οι παραπάνω γράφοι μπορεί να χρησιμοποιηθούνε για να αναπαραστήσουν την ροή ελέγχου του λογισμικού μας. Αυτό αποτυπώνεται στο σχήμα.



Σχήμα 2-7 - Συνδυασμός Γράφων

Ο γράφος ελέγχου του ανωτέρου σχήματος, μας δείχνει τα διαφορετικά μονοπάτια εκτέλεσης του προγράμματος. Όλες οι περιπτώσεις χρήσης προς έλεγχο σχεδιάζονται βάση αυτού του γραφήματος.

Στην τεχνική ελέγχου ροής υπάρχει η δυνατότητα να επιλεγεί το ποσοστό του κώδικα το οποίο θα ελεγχθεί. Υπάρχουν αρκετές μετρικές κάλυψης οι οποίες διαχωρίζονται βάση του ποσοστού κάλυψης που παρέχουνε. Οι πιο βασικές είναι :

- Κάλυψη Εντολής (Statement Coverage). Αυτή η μετρική μας δίνει το αν κάθε εντολή στον κώδικα εκτελείται τουλάχιστον μία φορά.Είναι η πιο βασική μετρική. Δεν γίνεται να βασιστεί ολόκληρος ο έλεγχος μόνο σε αυτή όμως, γιατί μπορεί να παραβλέψει αρκετά σφάλματα (συνθήκες λογικών πράξεων && δεν αξιολογούνται πλήρως). Είναι το αρχικό βήμα για αυτή την τεχνική.
- Κάλυψη Απόφασης (Condition Coverage). Αυτή η μετρική, εξασφαλίζει πως κάθε συνθήκη ελέγχου αξιολογείται και για τις δύο

εκβάσεις της, αληθές, ψευδής. Το πρόβλημα της είναι, ότι δεν ελέγχει σωστά συνθήκες που περιέχουν άλλες συνθήκες.

- Κάλυψη Συνθήκης (Condition Coverage). Αυτή η μετρική, λύνει το πρόβλημα της προηγούμενης. Τα πιθανά πολλαπλά μέρη μιας συνθήκης αξιολογούνται μεμονωμένα και για τις δύο εκβάσεις τους.
- Κάλυψη Πολλαπλής Συνθήκης (Multiple Condition Coverage). Αυτή η μετρική, προσφέρει εκτενέστερη κάλυψη από την προηγούμενη, αφού οι συνθήκες αξιολογούνται συνδυαστικά.
- Κάλυψη Μονοπατιού (Path Coverage). Αυτή η μετρική, εξασφαλίζει τον έλεγχο των πιθανών μονοπατιών εκτέλεσης. Το αρνητικό της είναι ότι δεν διαχειρίζεται σωστά τις επαναλήψεις.

Προτού την έναρξη της τεχνικής ελέγχου ροής πρέπει να αποφασιστεί το ποσοστό κάλυψης των επιμέρους μετρικών. Τα βήματα για την εφαρμογή αυτής της τεχνικής είναι τα εξής :

a. Δημιουργία του γράφου ροής ελέγχου, αναλύεται παρακάτω.
b. Υπολογισμός του Cyclomatic Complexity C (άκρες – κόμβοι + 2). Αυτό ο υπολογισμός μας δίνει τον ελάχιστο αριθμό των ανεξαρτήτων, μη επαναλαμβανόμενων μονοπατιών)
c. Για κάθε C δημιουργία περίπτωσης ελέγχου χρήσης.
d. Οι έλεγχοι εκτελούνται.

II. Ροή Δεδομένων (Data Flow Testing). Αυτή η τεχνική βοηθάει στην εύρεση σφαλμάτων στα δεδομένα του λογισμικού (έλεγχος συνθήκης με μη αρχικοποιημένες μεταβλητές, εκτύπωση μη αρχικοποιημένων μεταβλητών, αναφορά σε μεταβλητή που έχει διαγραφεί..). Κάθε μεταβλητή στο πρόγραμμα έχει και έναν κύκλο ζωής, πρώτα δημιουργούνται, μετά χρησιμοποιούνται και τέλος καταστρέφονται.

Η τεχνική αυτή, είναι σχεδόν ίδια με αυτή της ροής ελέγχου, προσθέτοντας επιπλέον πληροφορίες για τις επιμέρους μεταβλητές του προγράμματος.

Δύο μέθοδοι χρησιμοποιούνται σε αυτή την τεχνική:

- Στατική Ανάλυση Δεδομένων. Δημιουργείται ξανά ο γράφος ροής, και επιπρόσθετα τοποθετείται ο κύκλος ζωής της κάθε μεταβλητής. Με αυτή την τεχνική ο δοκιμαστής μπορεί να εξαλείψει σφάλματα όπως τα προηγούμενα.
- Δυναμική Ανάλυση Δεδομένων. Για κάθε μία μεταβλητή δημιουργείται και εκτελείται ένας γράφος ροής.

Ένα από τα συχνά λάθη των προγραμματιστών είναι η χρησιμοποίηση μη αρχικοποιημένων μεταβλητών ή μεταβλητών που έχουνε διαγραφεί, με την χρήση αυτής της τεχνικής τα εν λόγω σφάλματα εξαλείφονται.

Συνοψίζοντας, ο συνδυασμός των δύο στρατηγικών ελέγχου, παρέχει επαρκή κάλυψη στο λογισμικό υπό εξέταση. Οπότε, καλό είναι να χρησιμοποιούνται και οι δύο, με τις επιμέρους τεχνικές τους, για σωστή κάλυψη και δοκιμή του προγράμματος.

Οι παραπάνω στρατηγικές συνοψίζουνε την δυναμική προσέγγιση ελέγχου του συστήματος.

Όπως, έχει αναφερθεί υπάρχει και στατική προσέγγιση. Η στατική προσέγγιση δεν απαιτεί την εκτέλεση του κώδικα και αποτελείται από 2 στρατηγικές, την επιθεώρηση και την στατική ανάλυση.

Η στατική προσέγγιση περιλαμβάνει την εξακρίβωση της ορθότητας όχι μόνο του κώδικα, αλλά και των αρχικών απαιτήσεων ή προδιαγραφών.

2.3.3 Στατική Επιθεώρηση

Η επιθεώρηση είναι ένας συστηματικός έλεγχος των εγγράφων που έχουνε παραχθεί, από ένα ή περισσότερα άτομα με κύριο σκοπό την εύρεση και την αφαίρεση οποιονδήποτε λαθών. Το διάβασμα των εγγράφων είναι η πιο απλή μορφή.

Η τεχνική αυτή, μπορεί να χρησιμοποιηθεί για την σωστή επαλήθευση όλων των εγγράφων, από τις αρχικές απαιτήσεις μέχρι τις περιπτώσεις ελέγχου χρήσης. Από πολλούς θεωρείται το πρώτο βήμα στην διαδικασία ελέγχου του λογισμικού. Η επιθεώρηση των εγγράφων, βοηθάει στην εξάλειψη των λαθών προτού η

ομάδα φτάσει στην φάση ανάπτυξης (development), κάνοντας την αφαίρεση αυτών των λαθών ευκολότερη και λιγότερο χρονοβόρα. Γενικά ισχύει, πως όσο πιο νωρίς στην διαδικασία ανάπτυξης του λογισμικού ανακαλυφθεί ένα λάθος τόσο πιο εύκολα αντιμετωπίζεται. Η επιθεώρηση του κώδικα με γνώμονα γνωστές πρακτικές ανάπτυξης και ορθής συγγραφής μπορεί να εξαλείψει επιπλέον λάθη.

Τα κυριότερα λάθη τα οποία εξαλείφονται με αυτή την μέθοδο είναι :

- Διαφορές υλοποίησης σε σχέση με ήδη γνωστές πρακτικές.
- Προβλήματα στις αρχικές απαιτήσεις.
- Προβλήματα στον σχεδιασμό του λογισμικού(διαφορά από τις απαιτήσεις).
- Πολύπλοκος κώδικας ο οποίος οδηγεί στην έλλειψη μεταγενέστερης συντήρησης.

2.3.4 Στατική Ανάλυση

Η διαφορά αυτής τη τεχνικής με την προηγούμενη είναι ότι διεξάγεται αφού έχει αναπτυχθεί ο κώδικας. Γίνεται ανάλυση του πηγαίου κώδικα, χωρίς να εκτελείται με αποτέλεσμα να βρίσκονται λάθη όσο πιο νωρίς γίνεται, και πρν ξεκινήσει η δυναμική διαδικασία ελέγχου. Μερικά από τα λάθη που μπορεί να ανακαλύψει είναι :

- Μη σωστή διασύνδεση μεταξύ των υποσυστημάτων (διαφορετικά interfaces)
- Μεταβλητές οι οποίες δεν χρησιμοποιούνται.
- Παραβίαση κοινών κανόνων συγγραφής κώδικα.
- Σφάλματα ασφάλειας.
- Συντακτικές παραβιάσεις.

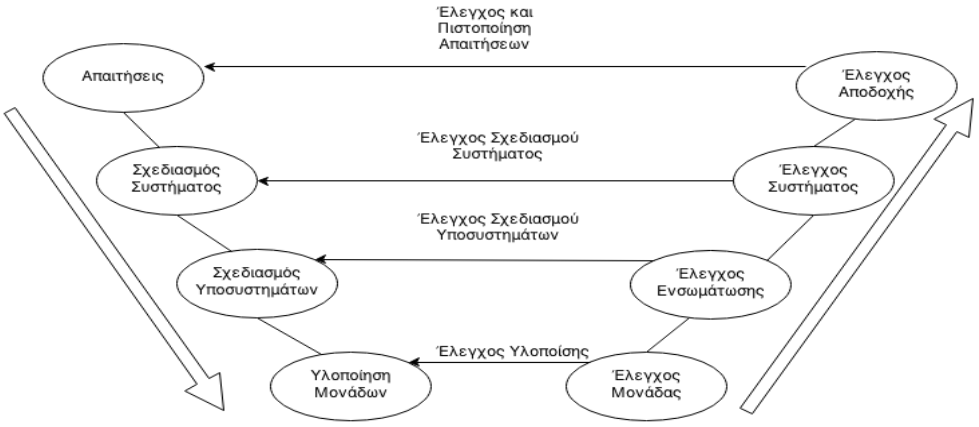
Και οι δύο τεχνικές στατικής προσέγγισης βοηθάνε στη εύρεση και αντιμετώπιση των σφαλμάτων όσο πιο νωρίς γίνεται στην διαδικασία ανάπτυξης του λογισμικού. Και η δυναμιή και η στατική προσέγγιση, προσφέρουνε εξίσου στην εύρεση των σφαλμάτων και πρέπει να χρησιμοποιούνται μαζί.

2.4 Επίπεδα Ελέγχου

Ανάλογα με το επίπεδο του συστήματος το οποίο ελέγχεται, ο έλεγχος χωρίζεται σε : (Sommerville, 2011)

- Έλεγχος Μονάδας (unit testing)
- Έλεγχος Ενσωμάτωσης (integration testing)
- Έλεγχος Συστήματος (system testing)
- Έλεγχος Αποδοχής (acceptance testing)

Η αντιστοίχηση των επιπέδων αποτυπώνεται στο σχήμα.



Σχήμα 2-8 Επίπεδα Ελέγχου

2.4.1 Έλεγχος Μονάδας

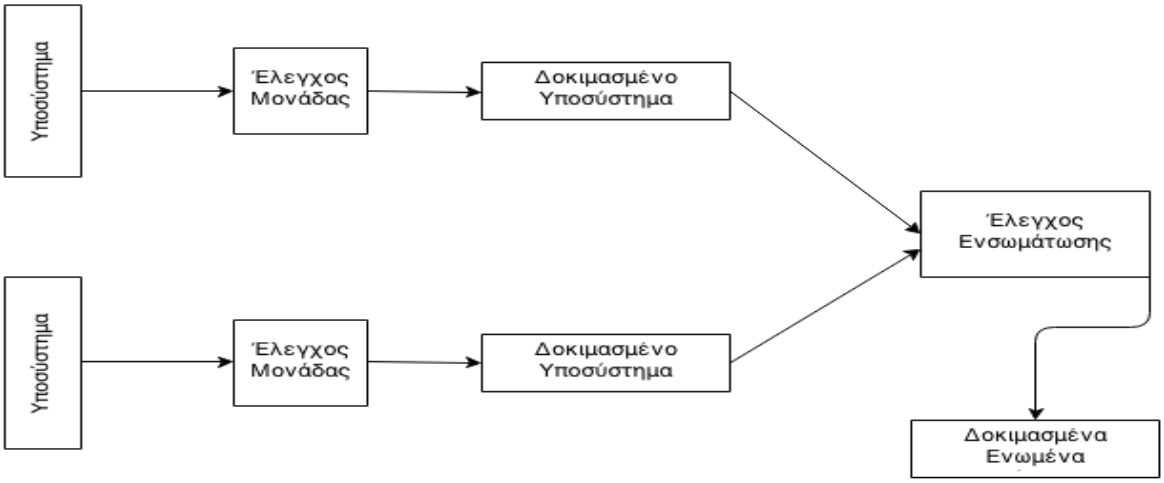
Ο έλεγχος μονάδας αποτελεί το βασικότερο στάδιο του συνολικού ελέγχου. Ο έλεγχος σε αυτό το επίπεδο επικεντρώνεται ξεχωριστά στα μικρότερα κομμάτια κώδικα. Τα ξεχωριστά κομμάτια μπορεί να είναι μία μέθοδος, μία διαδικασία ή μία κλάση ολόκληρη, στον αντικειμενοστραφή προγραμματισμό.

Οι έλεγχοι σε αυτό το επίπεδο συνήθως διεξάγονται από τον προγραμματιστή ο οποίος γράφει την μονάδα, διότι αυτός έχει καλύτερη γνώση του πώς είναι υλοποιημένη. Σκοπός είναι να βρεθούνε προβλήματα στα μικρότερα ανεξάρτητα κομμάτια που απαρτίζουνε το λογισμικό. Το αποτέλεσμα αυτού του επιπέδου είναι, μονάδες οι οποίες έχουν ελεγχθεί και είναι σύμφωνες με τις προδιαγραφές και απαιτήσεις τους.

Όλες οι παραπάνω τεχνικές και στρατηγικές χρησιμοποιούνται σε αυτό το επίπεδο ελέγχου, με κυριότερες τις τεχνικές του Άσπρου Κουτιού, αφού ο προγραμματιστής έχει πρόσβαση στον πηγαίο κώδικα.

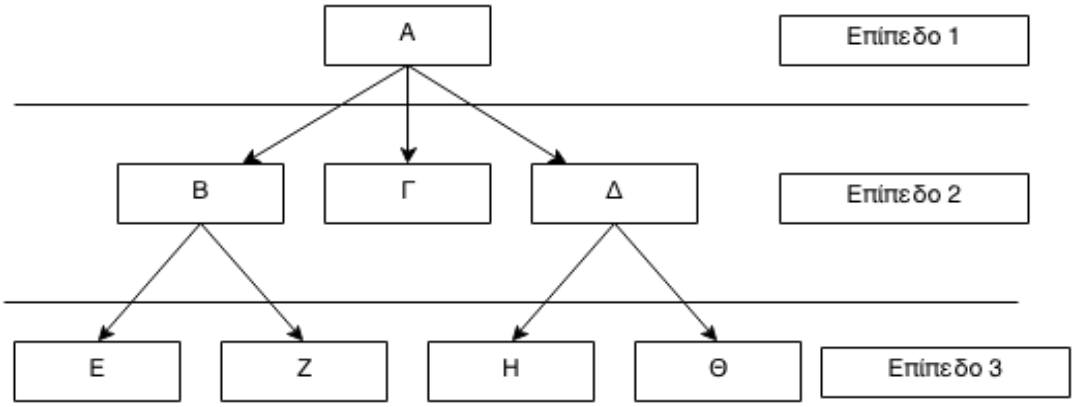
2.4.2 Έλεγχος Ενσωμάτωσης

Αυτό το επίπεδο ελέγχου ακολουθεί την ολοκλήρωση του ελέγχου μονάδας. Όλα τα επιμέρους υποσυστήματα ή μονάδες που έχουνε ελεγχθεί, χρειάζεται να ενωθούνε μεταξύ τους ώστε να δημιουργήσουνε ένα μεγαλύτερο σύνολο. Σκοπός αυτού του ελέγχου είναι η εύρεση προβλημάτων συνεργασίας μεταξύ των μονάδων. Συνήθως, και σε αυτό το επίπεδο ο προγραμματιστής εκτελεί χρέη και δοκιμαστή. Ακλουθεί η σχέση μεταξύ των 2 επιπέδων.



Σχήμα 2-9 - Έλεγχος Ενσωμάτωσης

Η διαδικασία περιλαμβάνει δημιουργία σεναρίων ελέγχου χρήσης, βάση των αρχικών προδιαγραφών, με σκοπό να εντοπιστούν οι διασυνδέσεις μεταξύ των μονάδων. Υπάρχουνε διαφορετικές τεχνικές για την ολοκλήρωση του ελέγχου οι οποίες διαχωρίζονται με βάση την σειρά την οποία ενσωματώνονται τα επιμέρους υποσυστήματα μεταξύ τους. Το σχήμα αναπαριστά τις διασυνδέσεις μεταξύ των υποσυστημάτων σε μορφή δέντρου.



Σχήμα 2-10 Έλεγχος Ενσωμάτωσης Υποσυστήματα

Οι πιο διαδεδομένες τεχνικές είναι:

I. Από Πάνω προς τα Κάτω Ενσωμάτωση. Σε αυτή την τεχνική η ενσωμάτωση ξεκινάει από το πρώτο, βασικό υποσύστημα (Α) και σε κάθε επανάληψη ενσωματώνεται και ένα υποσύστημα.Πρέπει να δημιουργηθούνε “κενά” υποσυστήματα (stubs), τα οποία παίζουνε τον ρόλο των υποσυστημάτων που καλούνται από τα υπό έλεγχο υποσυστήματα, τα οποία ακόμα δεν έχουνε ελεγχθεί ή υλοποιηθεί ακόμα. Στο παράδειγμά μας, η διαδικασία θα ήτανε κάπως έτσι, έλεγχος του Α – Β, Α – Γ, Α –Δ. Β – stub Ε, Β – stub Ζ...

   Τα θετικά αυτής τεχνικής είναι ότι οι περιπτώσεις ελέγχου δημιουργούνται βάση των απαιτήσεων, τα σφάλματα μπορούν να βρεθούν νωρίς, δεν χρειάζεται δημιουργία οδηγών, όπως επόμενη τεχνική.

   Τα προβλήματα της είναι ότι πολλές τεχνικές λεπτομέρειες αναβάλλονται και καθυστερεί ο έλεγχος τους, μπορεί να χρειάζονται αρκετά “κενά” υποσυστήματα, τα οποία αναλόγως την υλοποίηση μπορεί να είναι δύσκολο να δημιουργηθούν.

II. Από Κάτω προς τα Πάνω Ενσωμάτωση. Σε αυτή την τεχνική η ενσωμάτωση ξεκινάει ανάποδα, από τα χαμηλά υποσυστήματα προς τα ανώτερα. Τα “κενά” υποσυστήματα αντικαθίστανται από υποσυστήματα οδηγούς (drivers), τα οποία αναλαμβάνουνε την προσομοίωση των

ανωτέρων κόμβων. Στο παράδειγμά μας η διαδικασία θα ήτανε κάπως έτσι, έλεγχος του Ε – Ζ – Β, Η – Θ – Δ, όλα μαζί.

Τα θετικά της είναι ότι διαχειρίζεται καλύτερα την περίπτωση την οποία οι βασικές λειτουργίες είναι πολύπλοκες.

Τα αρνητικά της είναι ότι, το γραφικό περιβάλλον και η διεπαφή με τον χρήστη αφήνεται για έλεγχο στο τέλος και ότι είναι πιο δύσκολη η συγγραφή των οδηγών.

III. Σάντουιτς Ενσωμάτωση. Είναι συνδυασμός των δύο προηγούμενων τεχνικών. Η ενσωμάτωση γίνεται μία από πάνω και μία από κάτω.
IV. Big Bang Ενσωμάτωση. Δεν υπάρχει σαφής οδηγία ενσωμάτωσης. Αφού έχουνε σχεδιαστεί και υλοποιηθεί όλα τα υποσυστήματα, ενώνονται μεταξύ τους. Αυτή η μέθοδος δεν χρησιμοποιείται ποτέ.

Όλες οι παραπάνω τεχνικές και στρατηγικές χρησιμοποιούνται σε αυτό το επίπεδο ελέγχου.

2.4.3 Έλεγχος Συστήματος

Αυτό το επίπεδο ελέγχου ακολουθεί το επίπεδο ενσωμάτωσης. Χρησιμοποιώντας την στρατηγική του μαύρου κουτιού, οι δοκιμαστές πρώτα εξετάζουνε τις λειτουργικές απαιτήσεις που έχουνε τεθεί από τον πελάτη, και βάση αυτών σχεδιάζουν περιπτώσεις ελέγχου χρήσης για να βεβαιώσουνε ότι το λογισμικό ανταποκρίνεται σε αυτές τις απαιτήσεις. Το λογισμικό πλέον εξετάζεται ως σύνολο μαζί με το περιβάλλον στο οποίο θα εγκατασταθεί, οπότε νέοι παράμετροι πρέπει να ληφθούν υπόψη. Συνήθως σε αυτό το επίπεδο το λογισμικό εξετάζεται από μία ανεξάρτητη ομάδα. Διαφορετικά είδη ελέγχων πραγματοποιούνται σε αυτό το επίπεδο δίνοντας έμφαση στις μη λειτουργικές απαιτήσεις :

- Εξαντλητικοί έλεγχοι, ώστε να μετρηθεί το κατά πόσο το σύστημα μπορεί να λειτουργήσει κάτω από μεγάλο φόρτο εργασίας.
- Συμπεριφορά σε ακραίες συνθήκες.
- Έλεγχος ασφάλειας και πρόσβασης.
- Έλεγχος αποκατάστασης.

- Έλεγχος ποιότητας.
- Έλεγχοι συμβατότητας
- Έλεγχος αξιοπιστίας.
- Έλεγχος απόδοσης κ.ά.

Η στρατηγική του μαύρου κουτιού χρησιμοποιείται μόνο.

2.4.4 Έλεγχος Αποδοχής

Είναι το τελευταίο βήμα ελέγχου του λογισμικού. Το σύστημα πλέον αξιολογείται και δοκιμάζεται από τους πραγματικούς του χρήστες. Ελέγχεται το κατά πόσο το σύστημα ανταποκρίνεται στις αρχικές απαιτήσεις.

Μόνο η στρατηγική μαύρου κουτιού εφαρμόζεται σε αυτό το στάδιο.

2.5 Επίλογος

Σε αυτό το κεφάλαιο έγινε αναφορά στις στρατηγικές και μεθόδους που χρησιμοποιούνται για τον έλεγχο του λογισμικού.

Η διαδικασία που αναλύθηκε εφαρμόζεται στο τελευταίο στάδιο του μοντέλου του καταρράκτη. Παρατηρήσαμε ότι είναι μια χρονοβόρα διαδικασία και η εύρεση σφαλμάτων τόσο αργά στην ανάπτυξη του λογισμικού αποβαίνει μοιραία. Αυτός είναι ο λόγος που καινούριες μέθοδοι ανάπτυξης λογισμικού έχουνε αναπτυχθεί, με γνώμονα τον συνεχή έλεγχο του λογισμικού σε όλη την διάρκεια ανάπτυξης του.

Καλό είναι να αναφερθεί πως παρουσιάστηκαν οι βασικές έννοιες του ελέγχου λογισμικού. Υπάρχουν ακόμα πολλές τεχνικές και μέθοδοι που χρησιμοποιούνται (smoke test, regression test, test automation tools walkthrough, beta testing…).

Κεφάλαιο 3 Ευέλικτες Μεθοδολογίες Ανάπτυξης Λογισμικού

Στα προηγούμενα κεφάλαια αναλύσαμε την κλασσική μέθοδο ανάπτυξης λογισμικού καθώς και τις μεθοδολογίες και τεχνικές που υπάρχουν στην δοκιμή του.

Καταλήξαμε στην σημαντικότητα των δοκιμών για παραγωγή ποιοτικού κώδικα, καθώς και στο μειονέκτημα της πλήρους δοκιμής στα τελευταία της ανάπτυξης, αφού η αντιμετώπιση σφαλμάτων τόσο αργά είναι χρονοβόρα και πολυέξοδη διαδικασία.

Επιπλέον, αναφέρθηκε ότι η εξαρχής δέσμευση στις απαιτήσεις είναι κάτι το οποίο δεν ανταποκρίνεται στην πραγματικότητα, καθώς οι απαιτήσεις του πελάτη διαμορφώνονται και αλλάζουνε κατά την διάρκεια της ανάπτυξης λογισμικού.

Το μοντέλο του καταρράκτη δεν μπορεί να δώσει λύση στα προηγούμενα προβλήματα, αφού είναι ανελαστικό και καθόλου ευέλικτο ώστε να ανταποκριθεί σε καινούριες απαιτήσεις ενώ αφήνει την δοκιμή του λογισμικού στα τελευταία βήματα. Για αυτό τον λόγο έχουνε αναπτυχθεί καινούριες μεθοδολογίες. Αυτές οι μεθοδολογίες ονομάζονται Ευέλικτες (Agile). (Shore, 2007)

3.1 Agile Manifest

Οι μεθοδολογίες αυτές βασίζονται στην προσέγγιση στην οποία επαναλαμβάνονται οι διαδικασίες συλλογής απαιτήσεων. Επιτρέπουνε στην ομάδα να επικεντρώνεται στο λογισμικό παρά στον σχεδιασμό και την τεκμηρίωση του, ενώ σκοπός τους είναι να παραδώσουν γρήγορα λειτουργικό λογισμικό στον πελάτη ο οποίος με βάση αυτό να ζητήσει νέες λειτουργίες ή να θέσει νέες απαιτήσεις.

Το 2001, 17 προγραμματιστές δημιουργήσανε μία σειρά από 12 αρχές που ονομάστηκε Μανιφέστο των Ευέλικτων Μεθόδων, και αποτελεί την βάση για όλες τις ευέλικτες μεθοδολογίες. Οι 12 αρχές είναι : (Shore, 2007)

I. Πρώτη προτεραιότητα είναι η ικανοποίηση του πελάτη μέσω της έγκαιρης και συνεχούς παράδοσης λειτουργικού προϊόντος.

II. Οι αλλαγές στις απαιτήσεις είναι ευπρόσδεκτες, ακόμα και σε προχωρημένα στάδια της ανάπτυξης.
III. Παράδοση συχνά λογισμικού που λειτουργεί, σε διαστήματα μερικών εβδομάδων, μηνών, με προτίμηση στη συντομότερη χρονική κλίμακα.
IV. Οι προγραμματιστές και οι ειδικοί της αγοράς πρέπει να συνεργάζονται καθημερινά καθ'όλη την διάρκεια της ανάπτυξης.
V. Θεμελίωση των έργων γύρω από άτομα με πάθος και ενδιαφέρον. Το περιβάλλον διαμορφώνεται κατάλληλα, ενώ παρέχεται η αναγκαία υποστήριξη. Υπάρχει εμπιστοσύνη στις ικανότητες των μελώ της ομάδας.
VI. Το λογισμικό που λειτουργεί είναι το κυριότερο μέτρο προόδου.
VII. Οι ευέλικτες μεθοδολογίες προάγουν την αειφόρο ανάπτυξη. Όλα τα συμβαλλόμενα μέλη θα πρέπει να είναι σε θέση να διατηρούν ένα σταθερό ρυθμό επ'αόριστον.
VIII. Η διαρκής έμφαση στην τεχνική αρτιότητα και στην εύρυθμη σχεδίαση ενισχύουνε την ευελιξία.
IX. Η απλότητα είναι ουσιώδης.
X. Οι καλύτερες αρχιτεκτονικές, απαιτήσεις και σχέδια προκύπτουν από ομάδες που οργανώνονται μόνες τους.
XI. Σε τακτά χρονικά διαστήματα, η ομάδα συλλογίζεται για το πώς θα γίνει πιο αποτελεσματική.

Οι παραπάνω ιδέες ακολουθούνε όλες τις μεθοδολογίες.

Επιπλέον έχουνε θεσπιστεί και αξίες που πρέπει να τηρούνται. Οι δηλώσεις στα αριστερά πρέπει να προτιμούνται από αυτές στα δεξιά.

- Τα άτομα και τις αλληλεπιδράσεις πάνω από τις διαδικασίες και τα εργαλεία. Η αξία αυτή τονίζει το πόσο σημαντικός είναι ο ανθρώπινος παράγοντας στην ανάπτυξη του λογισμικού. Τα μέλη της ομάδας πρέπει να ενθαρρύνονται ώστε να αποδίδουνε το μέγιστο των ικανοτήτων τους, χωρίς να είναι περιορισμένοι από εργαλεία που πρέπει να χρησιμοποιήσουνε.
- Το λογισμικό που λειτουργεί πάνω από την εκτενή τεκμηρίωση. Η αξία αυτή αποτυπώνει τον βασικό στόχο της ανάπτυξης λογισμικού, την παράδοση δηλαδή λογισμικού ποιοτικού χωρίς σφάλματα. Υποστηρίζεται

ότι σε άλλες μεθοδολογίες αυτός ο στόχος έχει χαθεί και δίνεται μεγαλύτερη έμφαση στην παραγωγή εγγράφων τεκμηρίωσης.

- Την συνεργασία με τον πελάτη πάνε από συμβατικές διαπραγματεύσεις. Στις κλασσικές μεθοδολογίες, μετά την δέσμευση του εγγράφου SRS, ο πελάτης στην ουσία δεν αποτελούσε μέρος της διαδικασίας ανάπτυξης, παρά μόνο στα τελευταία στάδια ξανά, όπου του παρουσιαζότανε το τελικό προϊόν. Μεγιστοποιώντας την συνεργασία με τον πελάτη κερδίζουνε και τα μέλη της ομάδας και ο ίδιος, αφού οι μεν δεν χρειάζεται να κάνουνε υποθέσεις σχετικά με τις απαιτήσεις του πελάτη, ενώ ο ίδιος στο τέλος έχει ένα λογισμικό που καλύπτει τις απαιτήσεις του σε μεγαλύτερο βαθμό.
- Την ανταπόκριση στην αλλαγή πάνε από την τήρηση ενός προδιαγεγραμμένου σχεδίου. Οι αλλαγές στις απαιτήσεις κατά την διάρκεια της ανάπτυξης του λογισμικού είναι συχνό φαινόμενο. Οι ευέλικτες μεθοδολογίες είναι προετοιμασμένες για αυτές τις αλλαγές.

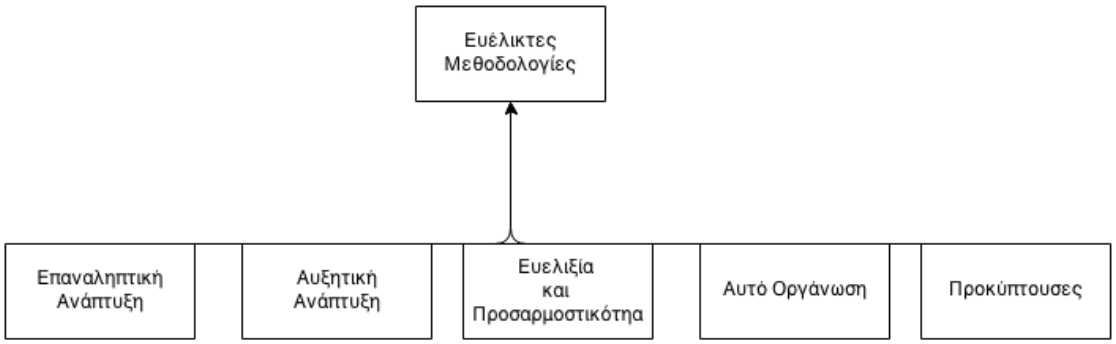
Συνοψίζοντας, αυτές οι μεθοδολογίες τοποθετούνε τον ανθρώπινο παράγοντα στο κέντρο του ενδιαφέροντος. Κύριος στόχο είναι η παραγωγή συνεχώς λειτουργικού λογισμικού, και επάνω σε αυτό να κάνει τις όποιες αλλαγές ο πελάτης.

3.2 Χαρακτηριστικά και στόχος

Τα κύρια χαρακτηριστικά όλων των μεθόδων είναι :

- Επαναληπτική ανάπτυξη. Εκδόσεις λειτουργικού λογισμικού αρκετά συχνά.
- Αυξητική ανάπτυξη. Κάθε υποσύστημα αναπτύσσεται με τέτοιο τρόπο ώστε να επιτρέπει την προσθήκη νέων απαιτήσεων. Αυτό πετυχαίνεται, με τον διαχωρισμό του συστήματος σε μικρότερα υποσυστήματα, βάση των λειτουργιών του , ώστε να προστίθενται νέες λειτουργίες κάθε φορά.
- Ευελιξία και Προσαρμοστικότητα. Οι αλλαγές είναι μέρος της διαδικασίας.
- Αυτό οργάνωση. Η ομάδα έχει την αυτονομία να οργανώνεται έτσι, ώστε να επιτύχει την βέλτιστη απόδοση της.
- Προκύπτουσες. Οι απαιτήσεις και τα εργαλεία που θα χρησιμοποιηθούνε προκύπτουν στην πορεία της ανάπτυξης.

Τα παραπάνω αποτυπώνονται στο σχήμα 1.



Σχήμα 3-1 Ευέλικτες Μέθοδοι Χαρακτηριστικά

Ο στόχος των ευέλικτων μεθοδολογιών είναι η ικανοποίηση του πελάτη με την γρήγορη και συνεχή παράδοση λογισμικού. Πολλοί πελάτες έχουνε ανάγκες οι οποίες αλλάζουνε στην διάρκεια του χρόνου. Για αυτό, οι απαιτήσεις του λογισμικού αλλάζουνε συνέχεια και οι μεθοδολογίες πρέπει να είναι φτιαγμένες για αυτές τις αλλαγές.

3.3 Πλεονεκτήματα

- Ταχύτερη ανάπτυξη του λογισμικού.
- Τελικό λογισμικό πιο κοντά στις πραγματικές απαιτήσεις του πελάτη. Από την στιγμή που είναι μέρος της όλης διαδικασίας, και η ίδια η ανάπτυξη καθοδηγείται από τις δικιές του απαιτήσεις.
- Καλύτερη Ποιότητα. Όλες οι μεθοδολογίες δίνουν έμφαση στην παραγωγή υψηλής ποιότητας λογισμικού.
- Αυξημένη Ευελιξία. Η ανάπτυξη είναι προσαρμόσιμη σε τυχόν αλλαγές των απαιτήσεων.
- Διαχείριση Κινδύνων. Οι μικρές αυξητικές εκδόσεις του λογισμικού, καθιστούνε ευκολότερο την εύρεση τυχόν προβλημάτων, νωρίς στην ανάπτυξη.
- Ικανοποίηση Πελάτη.

3.4 Μειονεκτήματα.

Παρά τα όσα πλεονεκτήματα έχουνε αναφερθεί οι μεθοδολογίες έχουνε και μερικά μειονεκτήματα :

- Δεν είναι το ίδιο δομημένες όσο το μοντέλο καταρράκτη. Χωρίς ένα εξαρχής πλάνο είναι δύσκολο να υπολογιστεί ο χρόνος και το κόστος.
- Η τεκμηρίωση δεν είναι το ίδιο πλήρης με αυτή στο μοντέλο καταρράκτη.
- Απαραίτητη είναι η εμπειρία σε όλη την ομάδα. Είναι πιο δύσκολο για κάποιον που να ξεκινήσει με αυτές τις μεθοδολογίες.

3.5 Μεθοδολογίες

Τα μειονεκτήματα βέβαια δεν αποτελούνε ανασταλτικό παράγοντα, αφού στις μέρες μας οι ευέλικτες μεθοδολογίες χρησιμοποιούνται κατά κόρον. Οι πιο σημαντικές και αυτές που θα αναλυθούνε είναι :

- eXtremeProgramming
- SCRUM

Μικρή αναφορά θα γίνει στην Test Driven Development διαδικασία.

3.6 eXtremeProgramming (Ακραίος Προγραμματισμός)

Η μεθοδολογία eXtremeProgramming είναι από τις πρώτες που ενσωματώνουνε όλες τις έννοιες και ιδέες των ευέλικτων μεθοδολογιών. Παρουσιάστηκε το 1999 από τον Beck, ενώ θεωρείται η πιο διαδεδομένη. (Kent Beck, 2004)

Είναι μία επαυξητική διαδικασία που υποστηρίζει την ταχύτερη ανάπτυξη του έργου, χρησιμοποιώντας μικρούς επαναληπτικούς κύκλους. Αυτός ο τρόπος δίνει την ευελιξία στην ομάδα να ανταποκρίνεται στις αλλαγές των απαιτήσεων του πελάτη.

3.6.1 Αρχές

Η φιλοσοφία της μεθοδολογίας αποτυπώνεται στις πέντε βασικές αρχές της :

I. Επικοινωνία. Συχνή επικοινωνία μεταξύ του πελάτη και των προγραμματιστών όσο και μεταξύ των μελών της ομάδας.
II. Απλότητα. Δίνει την ευχέρεια στην ομάδα προγραμματιστών να επιλέγουν τον απλούστερο τρόπο ανάπτυξης (σε σχεδιαστικό και προγραμματιστικό

επίπεδο), για την λύση του προβλήματος του πελάτη. Ο κώδικας βελτιώνεται στην πορεία, στις διαφορετικές εκδόσεις του. Η απλότητα του κώδικα τον καθιστά πιο κατανοητό, ενώ περιορίζονται και τα λάθη.

III. Ανατροφοδότηση. Με την τακτική και γρήγορη παράδοση εκδόσεων λειτουργικού λογισμικού στον πελάτη, προωθεί την ανατροφοδότηση από τον πελάτη βοηθώντας στον εντοπισμό τυχόν αποκλίσεων από τις απαιτήσεις. Συχνή ανατροφοδότηση λαμβάνει η ομάδα και από το ίδιο το σύστημα με την συνεχή δοκιμή του.

IV. Θάρρος. Ένα από τα πιο σημαντικά στοιχεία, είναι η σωστή ψυχολογία του προγραμματιστή. Σε αυτή την μεθοδολογία, όλα τα μέλη της ομάδας έχουνε την πλήρη βοήθεια ώστε να ολοκληρώσουνε το έργο, χωρίς να φοβούνται τις αλλαγές στον κώδικα. Αυτό επιτυγχάνεται με την χρήση εργαλείων, τα οποία διαχειρίζονται τον κώδικα και σε περίπτωση λαθών είναι ευκολότερο να γυρίσει η ομάδα σε παλαιότερη σταθερή έκδοση.

V. Σεβασμός. Απαραίτητος είναι ο σεβασμός ανάμεσα στα μέλη της ομάδας και στην δουλειά τους. Οι προγραμματιστές δεν προωθούνε αλλαγές οι οποίες δεν περνάνε τις δοκιμές.

Η δημιουργία λογισμικού βασίζεται στην συχνή επικοινωνία μεταξύ του πελάτη και των προγραμματιστών καθόλη την διάρκεια ανάπτυξης. Στο κλασσικό μοντέλο, αυτή η διαδικασία ολοκληρώνεται με την δημιουργία εγγράφων. Στον ακραίο προγραμματισμό πιο σημαντικό είναι η ταχεία ανάπτυξη του λογισμικού παρά των εγγράφων γύρω από αυτό.

Η μεθοδολογία αυτή ενθαρρύνει τους προγραμματιστές, να αρχίσουνε τον προγραμματισμό με την πιο απλή λύση και σε κάθε επανάληψη να προσθέτουνε καινούρια στοιχεία. Σε αντίθεση με την κλασσική μεθοδολογία, όπου πρώτα πρέπει να δημιουργηθούνε όλα τα απαραίτητα έγγραφα και μετά να ξεκινήσει η ουσιαστική ανάπτυξη.

3.6.2 Ρόλοι

Στον ακραίο προγραμματισμό υπάρχουνε διάφοροι ρόλοι, όπου ο καθένας έχει τα δικά του δικαιώματα και υπευθυνότητες. Όπως αναφέρθηκε πολύ σημαντική είναι η επικοινωνία μεταξύ του πελάτη και των προγραμματιστών και αυτό

επιτυγχάνεται με τον διαμοιρασμό εργασιών μεταξύ τους. Οι προγραμματιστές αναλαμβάνουνε τις τεχνικές αποφάσεις ενώ την όλη διαδικασία την καθοδηγεί ο πελάτης.

Πιο αναλυτικά οι ρόλοι είναι οι εξής:

- Πελάτης. Βάση των απαιτήσεων του αναπτύσσεται το λογισμικό, όσο πιο συχνή είναι η ανάμειξη του και η επικοινωνία με την ομάδα των προγραμματιστών τόσο πιο κοντά στις απαιτήσεις του θα είναι το τελικό λογισμικό. Οι απαιτήσεις του καταγράφονται ως ιστορίες από την ομάδα και σύμφωνα με αυτές τις ιστορίες προχωράει το έργο. Επίσης, αποφασίζει πότε μία απαίτηση έχει ικανοποιηθεί, με το να δημιουργεί ελέγχους αποδοχής σε συνεργασία με τον δοκιμαστή.
- Προγραμματιστής. Αναλαμβάνει να γράψει τον κώδικα και τις δοκιμές τις οποίες πρέπει να περάσει κάθε κομμάτι κώδικα. Μετατρέπει τις ιστορίες που λαμβάνει από τον πελάτη σε λογισμικό. Είναι απαραίτητη η σωστή συνεργασία με τον πελάτη. Με την ανάλυση της κάθε ιστορίας μπορεί να κάνει και μια χρονική πρόβλεψη για το πότε θα έχει ολοκληρωθεί.
- Δοκιμαστής. Ο ρόλος του είναι να βοηθάει τον πελάτη στην δημιουργία ελέγχων αποδοχής, ενώ βοηθάει και τους προγραμματιστές στην δημιουργία ελέγχων μονάδας. Τέλος, δοκιμάζει ανά τακτά διαστήματα το σύστημα.
- Καταγραφέας. Κρατάει το πρόγραμμα της όλης διαδικασίας. Ένα από τα πιο σημαντικά σημεία της μεθοδολογίας είναι η εκτίμηση του χρόνου παράδοσης και η εκτίμηση των πόρων που χρειάζονται.Αυτή την δουλειά αναλαμβάνει ο καταγραφέας, ενώ καταγράφει και την πρόοδο υλοποίησης κάθε επανάληψης.
- Εκπαιδευτής. Είναι ο υπεύθυνος για την ορθή υλοποίηση και τήρηση της μεθοδολογίας. Καθοδηγεί την ομάδα για την σωστή εφαρμογή της μεθοδολογίας.
- Σύμβουλος. Συνήθως εξωτερικός συνεργάτης ειδικός σε έναν συγκεκριμένο τομέα. Δίνει συμβουλές στην ομάδα.

3.6.3 Διαδικασία Ανάπτυξης

Ο κύκλος ζωής ανάπτυξης ενός λογισμικού χρησιμοποιώντας αυτή την μεθοδολογία διαμορφώνεται ως εξής :

i. Φάση Διερεύνησης. Στη αρχική φάση, μοντελοποιούνται οι απαιτήσεις του χρήστη καθώς και η αρχιτεκτονική του συστήματος. Οι απαιτήσεις του χρήστη παρουσιάζονται ως Ιστορίες Χρήσης στους προγραμματιστές. Αυτές οι ιστορίες γράφονται από τον πελάτη και στην ουσία αποτυπώνουνε το τί λειτουργικότητες θέλει να περιλαμβάνει το λογισμικό. Παρά το ότι η μεθοδολογία δεν έχει ξεκάθαρη φάση εντοπισμού απαιτήσεων, αυτό επιτυγχάνεται με τις ιστορίες.
Επιπλέον, καθορίζεται από την ομάδα τα εργαλεία και οι τεχνολογίες που θα χρησιμοποιηθούνε.

ii. Φάση Προγραμματισμού. Σε αυτή την φάση οι ιστορίες του χρήστη χωρίζονται σε επιμέρους διεργασίες. Οι διεργασίες ταξινομούνται ως προς τον βαθμό σημαντικότητας, ώστε να καθοριστεί από ποιες θα ξεκινήσει η ομάδα. Κάθε διεργασία, θα αποτελέσει και έναν κύκλο επανάληψης, μέσα στον οποίο θα αναπτυχθεί μία λειτουργία του συστήματος. Η ομάδα εκτιμά τον χρόνο που απαιτείται για την υλοποιήση της κάθε διεργασίας και το ποιά χαρακτηριστικά θα αναπτυχθούνε κατά την διάρκεια της κάθε επανάληψης.

iii. Φάση Σχεδιασμού. Κάθε επανάληψη ξεκινάει με τον σχεδιασμό της. Όπως έχει αναφερθεί η απλότητα είναι σημαντική, οπότε σε επιλέγεται να απαπτυχθούνε μόνο τα βασικά κομμάτια μιας ιστορίας, και στην συνέχεια να προστίθενται και υπόλοιπα. Σε αυτή την φάση, επιλέγεται και το πρότυπο γραφής του κώδικα (το πώς θα ονοματίζονται οι κλάσεις, οι μεταβλητές..). Σημαντικό είναι ότι σε αυτή την φάση καθορίζεται ο αριθμός των επαναλήψεων καθώς και το ποιους λειτουργικούς έλεγχους πρέπει να περάσει το σύστημα για να θεωρείται ολοκληρωμένη κάθε ιστορία.

iv. Φάση Ανάπτυξης Κώδικα. Στην μεθοδολογία του Ακραίου Προγραμματισμού, αυτή είναι η πιο σημαντική φάση. Ο πυρήνας της

μεθοδολογίας είναι ο προγραμματισμός. Η προτεραιότητα δίνεται σε αυτήν την φάση, σε σχέση με την τεκμηρίωση για παράδειγμα. Υπάρχουνε κάποιοι κανόνες που πρέπει να τηρούνται :

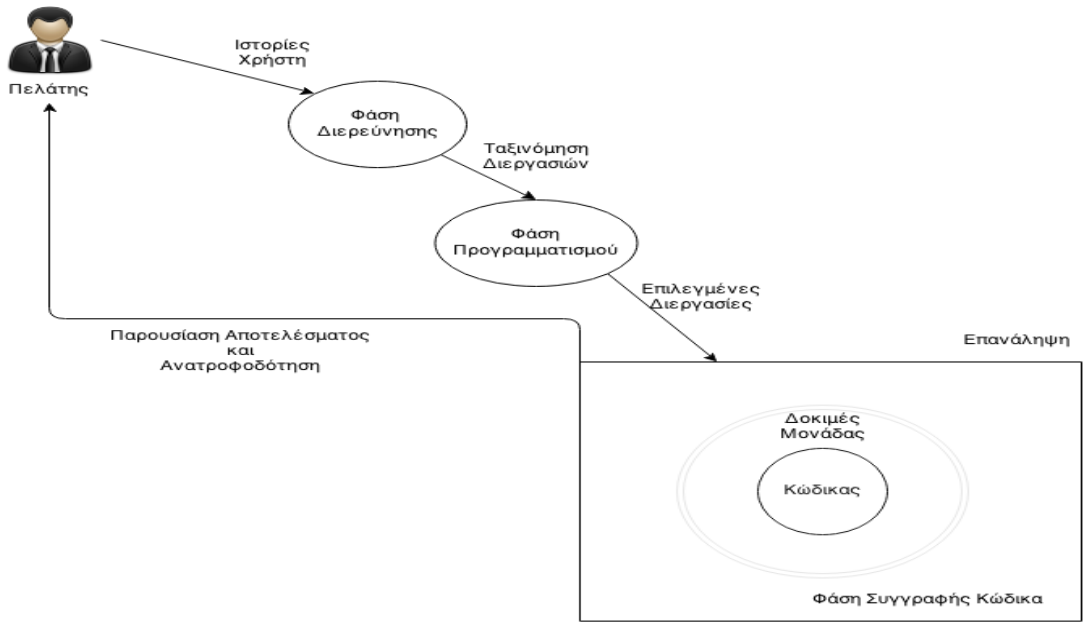
- Ανάπτυξη του κώδικα βάση του πρότυπου ραφής που αποφασίστηκε στην προηγούμενη φάση.
- Ο κώδικας αναπτύσσεται από δύο προγραμματιστές ώστε να παράγεται υψηλότερης ποιότητας αποτέλεσμα.
- Το πολύ 40 ώρες εργασίας την εβδομάδα, για κάθε προγραμματιστή.
- Συχνές ενσωμάτωσης του κώδικα στην κεντρική αποθήκη, ώστε να εκτελούνται και οι ανάλογοι έλεγχοι, και ενσωμάτωση μόνο όταν έχει περάσει με επιτυχία ο κώδικας τους ελέγχους μονάδας.

v. Φάση Ελέγχων. Ο έλεγχος του κώδικα είναι αναπόσπαστο κομμάτι της διαδικασίας και δεν γίνεται μόνο στο τελευταίο στάδιο σε σύγκριση με το μοντέλο του καταρράκτη.
Οι έλεγχοι είναι ενσωματωμένοι στην φάση συγγραφής κώδικα. Κάθε υπομονάδα που αναπτύσσεται έχει τους δικούς της ελέγχους μονάδας, που πρέπει να περάσει με επιτυχία προτού ο κώδικας ενσωματωθεί στην κεντρική αποθήκη.
Επιπλέον, ο πελάτης σε συνεργασία με τον δοκιμαστή αναπτύσουνε ελέγχους αποδοχής οι οποίοι κρίνουνε το κατά πόσο είναι σωστά υλοποιημένη μία ιστορία.

vi. Φάση Ανατροφοδότησης. Η βάση της μεθοδολογίας είναι η συνεχής ενημέρωση του πελάτη για το λογισμικό και η ανατροφοδότηση του καθόλη την διάρκεια της ανάπτυξης. Το κύριο εργαλείο για αυτή την ανατροφοδότηση είναι οι έλεγχοι αποδοχής που έχουνε αναπτυχθεί. Σε κάθε καινούρια επίδειξη του λογισμικού ο πελάτης μπορεί να προσθέσει καινούρια χαρακτηριστικά, τα οποία θα αποτελέσουνε τις ιστορίες για τον επόμενο κύκλο ανάπτυξης.

Σε όλη την διαδικασία ανάπτυξης η ομάδα έχει καθημερινές συναντήσεις, στις οποίες αναλύεται το τί έχει αναπτυχθεί μέχρι στιγμής και ποιο είναι το πλάνο της ημέρας. Επίσης, στο τέλος κάθε επανάληψης, υπάρχει συνάντηση με τον πελάτη όπου και παρουσιάζεται το λογισμικό μέχρι στιγμής.

Το σχήμα 2 παρουσιάζει αυτές τις φάσεις.



Σχήμα 3-2 Φάσεις Ακραίου Προγραμματισμού

Στην ανατροφοδότηση, σημαντικό ρόλο παίζουνε και οι δοκιμές αποδοχής που αναπτυχθήκανε σε συνεργασία με τον πελάτη. Παρατηρούμε ότι η δοκιμή του κώδικα είναι πολύ συχνή και απαραίτητη σε αυτή την μεθοδολογία.

3.6.4 Πρακτικές

Η ιδέα του Ακραίου Προγραμματισμού μέχρι στιγμής είναι ότι ο πελάτης παίρνει τις αποφάσεις που αφορούν τις λειτουργικές απαιτήσεις του συστήματος ενώ η η ομάδα αποφασίζει για τα τεχνικά ζητήματα.

Την διαδικασία ανάπτυξης συμπληρώνουνε οι εξής 12 πρακτικές :

- Ιστορίες Χρήσης. Θεωρείται το αντίστοιχο των περιπτώσεων χρήσης της UML. Στην ουσία, ο πελάτης ορίζει τις προδιαγραφές που θέλει να έχει το

λογισμικό. Οι ιστορίες αυτές είναι όσο το δυνατόν πιο αναλυτικές, ώστε η ομάδα να κατανοήσει πλήρως τις ανάγκες του πελάτη.

- Σύντομες Εκδόσεις. Η μεθοδολογία δίνει έμφαση στις συνεχόμενες εκδόσεις σύντομες του λογισμικού, ώστε να μπορεί να πάρει ανατροφοδότηση από τον πελάτη και να ενσωματώσει τυχόν αλλαγές, όπως ορίζουν οι κανόνες των Ευέλικτων Μεθόδων.
- Συμβολισμός. Περιγραφή του συστήματος με όρους κατανοητούς και από τον πελάτη και από την ομάδα.
- Συλλογική Κυριότητα. Ο κώδικας ανήκει σε όλη την ομάδα. Οπότε ο καθένας από την ομάδα μπορεί να αλλάξει από οτιδήποτε.
- Χρήση Προγραμματιστικών Προτύπων. Είναι απαραίτητη η τήρηση των προτύπων καθόλη την διάρκεια ανάπτυξης, ώστε να υπάρχει συμβατότητα μεταξύ των ομάδων.
- Απλός Σχεδιασμός. Σημαντικός είναι ο σχεδιασμός της απλούστερης δυνατής λύσης. Η μη αναγκαία πολυπλοκότητα πρέπει να αποφεύγεται.
- Επανακατασκευή κώδικα. Το λογισμικό πρέπει συνέχεια να βελτιώνεται, με σκοπό να αφαιρούνται προβλήματα.
- Δοκιμές. Η μεθοδολογία ορίζει, πρώτα την συγγραφή δοκιμών μονάδας και μετά του πραγματικού κώδικα, ο οποίος πρέπει να περνάει με επιτυχία αυτές τις δοκιμές.
- Προγραμματισμός σε ζεύγη. Οι προγραμματιστές εργάζονται σε ζεύγη. Με τον τρόπο αυτόν, ο κώδικας ανασκοπείται την ώρα δημιουργίας του. Ο ένας προγραμματιστής γράφει την κώδικα, ενώ ο άλλος παρατηρεί και γράφει τις δοκιμές μονάδας.
- Συνεχής Ενσωμάτωση. Όλες οι αλλαγές στον κώδικα δοκιμάζονται και ενσωματώνονται συνεχώς στην αποθήκη. Θα γίνει εκτενέστερη αναφορά σε αυτό το κομμάτι στα επόμενα κεφάλαια.
- 40 ώρες Εργασία. Στόχος είναι να μην ξεπερνάει τις 40 ώρες την εβδομάδα η ομάδα. Σε περίπτωση που δεν τηρείται αυτό, σημαίνει πως γίνεται κάτι λάθος.
- Συμμετοχή του πελάτη. Ο πελάτης πρέπει να θεωρείται μέλος της ομάδας. Πρέπει να είναι διαθέσιμος καθόλη την διαδικασία ανάπτυξης

ώστε να απαντήσει τυχόν ερωτήσεις με σκοπό να μην υλοποιηθεί κάτι περιττό ή μη σωστό.

3.6.5 Ανάπτυξη Καθοδηγούμενη από τις Δοκιμές

Στο κεφάλαιο του Ακραίου Προγραμματισμού πρέπει να γίνει και μια μικρή αναφορά στην ανάπτυξη η οποία είναι καθοδηγούμενη από τις δοκιμές (Test Driven Development), η οποία αναπτύχθηκε από τον δημιουργό της μεθοδολογίας που εξετάζουμε. (Astels, 2003)

Όπως έχουμε παρατηρήσει μέχρι στιγμής οι δοκιμές παίζουνε πολύ σημαντικό ρόλο στην μεθοδολογία. Στη TDD, πρώτα δημιουργούνται οι δοκιμές μονάδας για κάθε απαίτηση και μετά γράφεται επαρκής κώδικας ώστε να περάσουν με επιτυχία μία οι δοκιμές.

Για την ανάπτυξη των δοκιμών ο προγραμματιστής πρέπει να γνωρίζει αρκετά καλά τις απαιτήσεις του πελάτη. Συνήθως, έπειτα από κάθε δοκιμή που περνάει, ο κώδικας ανακατασκευάζεται ώστε να γίνει καλύτερος ποιοτικά. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να περάσουνε με επιτυχία όλες οι δοκιμές μονάδας.

Τέλος, όσο περισσότερο αναπτύσσεται το λογισμικό τόσα περισσότερες δοκιμές χρησιμοποιούνται, ενώ στην πορεία αναπτύσσονται δοκιμές ενσωμάτωσης.

Τα πλεονεκτήματα αυτού του τρόπου ανάπτυξης είναι :

- Κώδικας καθαρός από λάθη, συνεπώς και καλύτερος ποιοτικά
- Δυνατότητα εύκολης ανακατασκευής.
- Γρηγορότερος εντωπισμός μη σωστών απαιτήσεων.
- Εύκολη ανταπόκριση σε νέες απαιτήσεις και επεκτάσεις.

Από την άλλη τα μειονεκτήματα είναι :

- Προσθέτει επιπλέον πολυπλοκότητα. Μερικές φορές είναι πιο δύσκολο να αναπτυχθεί η δοκιμή από ότι ο πραγματικός κώδικας.
- Είναι πιο χρονοβόρο.

3.6.6 Κριτική του Ακραίου Προγραμματισμού

Παρά τα όσα πλεονεκτήματα μας προσφέρει αυτή η μεθοδολογίας υπάρχουνε και κάποια μειονεκτήματα επιπροσθέτως των ευέλικτων μεθόδων :

- Είναι δύσκολο να τηρηθεί κατά γράμμα.
- Πολλοί προγραμματιστές δεν είναι συνηθισμένοι στην ιδέα να δουλεύουνε μαζί με κάποιον άλλον.
- Μερικές φορές, ο πελάτης μπορεί να δώσει περισσότερη έμφαση στην δημιουργία δοκιμών αποδοχής παρά στις πραγματικές απαιτήσεις.

3.7 SCRUM

Ο όρος SCRUM προέρχεται από το rugby, όταν η η ομάδα συνεργάζεται για την επαναφορά της μπάλας. Αυτή η μεθοδολογία πρωτοεμφανίστηκε το 1986 σε ένα ιαπωνέζικο άρθρο, σχετικά με διαδικασία ανάπτυξης προϊόντος. Η μεθοδολογία δεν έχει εφαρμογή μόνο στην ανάπτυξη λογισμικού, αλλά χρησιμοποιείται και στην ανάλυση πολύπλοκων έργων εκτός πληροφορικής. Είναι επαυξητική και επαναληπτική μεθοδολογία, που δεν ορίζει κάποια συγκεκριμένη τεχνική, αλλά δίνει έμφαση στο πως η ομάδα θα πρέπει να λειτουργήσει ώστε να καταφέρει να παράγει το τελικό αποτέλεσμα σε ένα συνεχώς μεταβαλλόμενο περιβάλλον. Συγκεκριμένα, στόχος της είναι όσο το δυνατόν μικρότερο χρονικό διάστημα κύκλου ανάπτυξης και παράδοσης τμημάτων κώδικα, ενώ καθημερινή είναι η επικοινωνία των μελών της κάθε ομάδας. (Schwaber, 2001) (Shore, 2007)

3.7.1 Διαδικασία Ανάπτυξης

Η μεθοδολογία χωρίζει την όλη διαδικασία ανάπτυξης σε κύκλους, που ονομάζονται Sprint. Κάθε κύκλος συνήθως έχει διάρκεια από 1 ως 4 εβδομάδες. Κάθε κύκλος έχει εξαρχής προκαθορισμένη διάρκεια και ολοκληρώνεται σε συγκεκριμένη ημερομηνία ακόμα και αν δεν έχει ολοκληρωθεί η διεργασία που ήτανε να γίνει.

Στην αρχή κάθε κύκλου, κάθε μία από τις ομάδες ανάπτυξης επιλέγει διεργασίες από μία Λίστα απαιτήσεων, που περιλαμβάνει όλες τις απαιτήσεις που ζητήθηκαν από τον πελάτη, ταξινομημένες βάση προτεραιότητας και δεσμεύεται να τις

ολοκληρώσει μέχρι το πέρας του κύκλου. Κατά την διάρκεια του κύκλου αυτές οι διεργασίες και τα απαιτούμενα αποτελέσματα δεν αλλάζουνε.

Κάθε μέρα, η ομάδα συγκεντρώνεται και συζητάνε σχετικά με το τί επιτέχθη την προηγούμενη ημέρα, ενώ θέτουνε τους στόχους της ημέρας. Οι στόχοι αυτοί συνήθως αποτυπώνονται σε έναν πίνακα διεργασιών.

Στο τέλος κάθε κύκλου η ομάδα παρουσιάζει τί έχει καταφέρει, άμα ολοκλήρωσε όλες τις διεργασίες που είχε, ονομάζεται “done”. Σε αυτό το στάδιο λαμβάνει ανατροφοδότηση από τον πελάτη. Οι όποιες αλλαγές ζητηθούνε ενσωματώνονται στον επόμενο κύκλο.

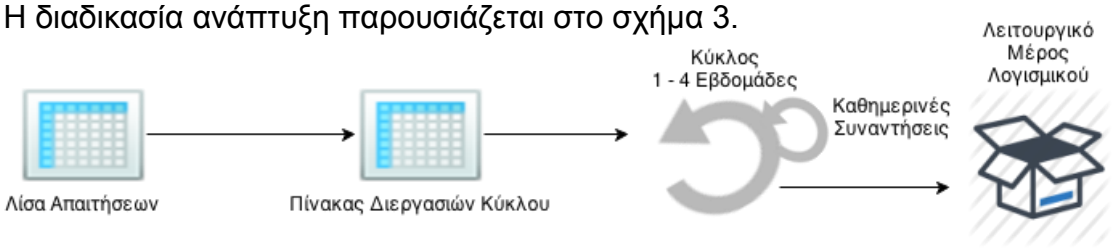
Αναλυτικά οι φάσεις είναι :

i. Αρχική Διερεύνηση. Αυτή η φάση χωρίζεται σε :
    - Ανάλυση. Κύριος στόχος αυτής της φάσης είναι ο καθορισμός των απαιτήσεων και του επιδιωκόμενου στόχου. Σε αυτή την φάση δημιουργείται η λίστα απαιτήσεων, σε συμφωνία με τον πελάτη και την ομάδα, ενώ γίνεται και μια εκτίμηση για το πόσο χρόνο θα χρειαστεί κάθε απαίτηση μέχρι να υλοποιηθεί. Η λίστα αυτή ενημερώνεται σε κάθε κύκλο. Επίσης, γίνεται και ανάλυση πιθανών ρίσκων.
    - Αρχιτεκτονικής. Η ομάδα μελετά την λίστα απαιτήσεων, γίνεται ο σχεδιασμός του συστήματος ενώ δημιουργείται και ένα αρχικό πλάνο του τι θα περιλαμβάνει η τελική έκδοση του συστήματος.

ii. Ανάπτυξης. Η φάση αυτή είναι η καρδιά της μεθοδολογίας, ενώ πλέον εφαρμόζονται οι τεχνικές των ευέλικτων μεθόδων. Σε αυτό το σημείο είναι που ξεκινάει ο κάθε κύκλος. Η ομάδα επιλέγει το ποιά απαίτηση, βάση προτεραιότητας, θα αναπτύξει σε αυτόν τον κύκλο και δημιουργεί τον πίνακα διεργασιών κύκλου.

iii. Ολοκλήρωσης. Σε αυτή την φάση, έχουνε ολοκληρωθεί όλοι οι απαιτούμενοι κύκλοι. Σε συμφωνία με τον πελάτη επιβεβαιώνεται η ανάπτυξη όλων των αρχικών απαιτήσεων ώστε να ξεκινήσει η διαδικασία παράδοσης του λογισμικού.

Η διαδικασία ανάπτυξη παρουσιάζεται στο σχήμα 3.



Σχήμα 3-3 Φάσεις SCRUM

3.7.2 Ρόλοι

Στην SCRUM μεθοδολογία υπάρχουν οι εξής ρόλοι :

- SCRUM Master. Ο οποίος είναι κάποιος ειδικός επάνω στην μεθοδολογία. Σκοπός του είναι να διασφαλίσει ότι το έργο υλοποιείται σύμφωνα με τους κανόνες της μεθοδολογίας. Συνεργάζεται με την ομάδα ανάπτυξης και προσπαθεί να μειώσει τα όποια εξωτερικά προβλήματα μπορεί να την απασχολούνε. Επίσης, είναι ο συνδετικός κρίκος ανάμεσα στην ομάδα τον πελάτη και τη διοίκηση.
- Ιδιοκτήτης του Έργου. Είναι ο υπεύθυνος για την διαχείριση του έργου. Βοηθάει στην λήψη αποφάσεων σχετικά με την Λίστα Απαιτήσεων.
- Πελάτης. Λαμβάνει μέρος στην δημιουργία της Λίστας Απαιτήσεων. Στο τέλος κάθε κύκλου προσθέτει λειτουργίες ή ζητάει αλλαγές.
- Ομάδα SCRUM. Είναι υπεύθυνη για τις αποφάσεις που πρέπει να λάβουνε χώρα με σκοπό την επίτευξη των στόχων κάθε κύκλου. Είναι σύνηθες, περισσότερες από μία ομάδες να εργάζονται παράλληλα κατά την διάρκεια ανάπτυξης.

3.7.3 Κριτική της SCRUM

Παρά τα πλεονεκτήματα που είδαμε πως έχει η μεθοδολογία, υπάρχουνε και μερικά μειονεκτήματα :

- Σε περίπτωση που μία απαίτηση ή διεργασία δεν είναι πλήρως ορισμένη, είναι δύσκολο να γίνει εκτίμηση χρόνου.
- Για την ορθή λειτουργία οι ομάδες πρέπει να στελεχώνονται με πολύ έμπειρα άτομα.

- Σε περίπτωση που τα μέλη της ομάδας δεν είναι πλήρως αφοσιωμένα το έργο κατά πάσα πιθανότητα θα αποτύχει.

3.8 Επίλογος

Σε αυτό το κεφάλαιο αναλυθήκανε οι αρχές των ευέλικτων μεθοδολογιών ανάπτυξης λογισμικού. Αναφέρθηκαν οι αρχές και οι ιδέες πίσω από αυτές τις μεθοδολογίες και γα ποιον λόγο προτιμούνται τις μέρες μας σε σχέση με το κλασσικό μοντέλο του καταρράκτη.

Παρουσιαστήκανε οι δύο πιο δημοφιλείς ευέλικτες μεθοδολογίες Extreme Programming και SCRUM. Για κάθε μία είδαμε πως υλοποιούνε τις ευέλικτες τεχνικές και ιδέες, ενώ μοντελοποιήθηκε η διαδικασία ανάπτυξης λογισμικού και για τις δύο.

Κεφάλαιο 4 Continuous Integration και εργαλείο Jenkins

Η τεχνική continuous integration (συνεχής ενσωμάτωση), έχει βοηθήσει στην αποτελεσματικότερη ανάπτυξη λογισμικού, ιδιαίτερα μετά την εμφάνιση των Agile μεθόδων.

Όταν ένα λογισμικό αποτελείται από εκατοντάδες ή χιλιάδες διαφορετικά υποσυστήματα (components), με πολύπλοκη εξάρτηση μεταξύ τους, η παραμικρή αλλαγή σε ένα από αυτά τα υποσυστήματα επηρεάζει την συμπεριφορά των υπολοίπων. Η σύνδεση όλων αυτών των υποσυστημάτων στο τέλικο βήμα της ανάπτυξης του λογισμικού προκαλεί έντονα προβλήματα, μερικά από αυτά είναι :

- Τα οποιαδήποτε σφάλματα διασύνδεσης εμφανίζονται στο τελευταίο στάδιο της ανάπτυξης λογισμικού, άρα είναι πιο δύσκολο να διορθωθούνε και πιο χρονοβόρο.
- Η απομόνωση του σφάλματος είναι πολύ πιο δύσκολη σε σχέση με τον εντοπισμό του στις προηγούμενες φάσεις.
- Είναι πιθανή η εμφάνιση σφαλμάτων που δεν είχανε προβλεφθεί από την αρχή.

Για την αποφυγή τέτοιων προβλημάτων η μεθοδολογία της συνεχόμενης ενσωμάτωσης έχει αναπτυχθεί. (Shore, 2007) (Sommerville, 2011)

Η τεχνική του continuous integration (συνεχής ενσωμάτωση) εφαρμόζεται καθόλη την διάρκεια του κύκλου ανάπτυξης λογισμικού, στην οποία κάθε μέλος της ομάδας ανάπτυξης ενσωματώνει την δουλειά του με τον κεντρικό κώδικα τουλάχιστον μία φορά κάθε μέρα. Αυτό μας οδηγεί σε πολλαπλές ενσωματώσεις κάθε μέρα.

Κάθε καινούρια ενσωμάτωση πιστοποιείται από μία αυτόματη διαδικασία, η οποία περιλαμβάνει και αρκετά τεστ, με σκοπό να αναγνωριστούνε πιθανά λάθη όσο πιο γρήγορα γίνεται. Αυτή η διαδικασία οδηγεί σε μείωση των λαθών ενσωματώσης, με αποτέλεσμα η ομάδα να παράγει πιο συνεκτικό λογισμικό και πιο γρήγορα, αφού δεν περιμένει την τελευταία φάση της ανάπτυξης για την αναγνώριση των σφαλμάτων.

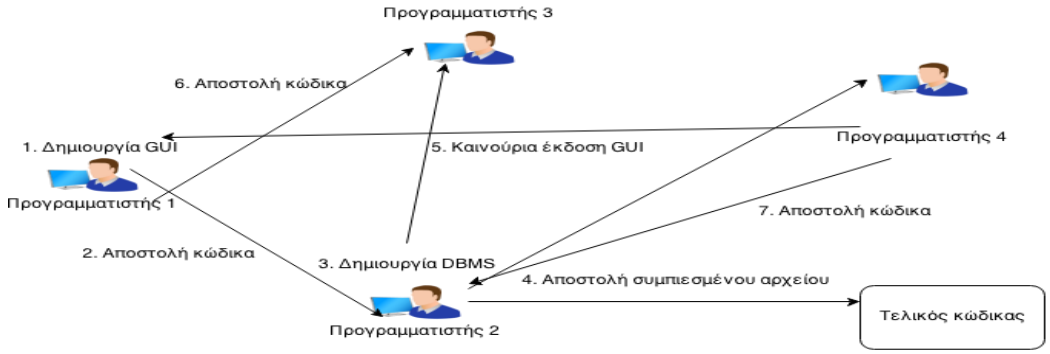
4.1 Κεντρική Διαχείριση και Έλεγχος Έκδοσης

Για να είναι εφικτή η διαχείριση λογισμικού τέτοιου μεγέθους χρησιμοποιείται μια αποθήκη (repository), στην οποία αποθηκεύεται το κεντρικό λογισμικό ή ο κύριος κώδικας. Κάθε αλλαγή στα επιμέρους υποσυστήματα πρέπει να ενσωματωθεί σε αυτήν την αποθήκη. Στις μέρες μας υπάρχουνε αρκετοί πάροχοι τέτοιων αποθηκών, με τους ποιο γνωστούς το github και το sourceforge.

Η τεχνική αυτή επιτρέπει την δημιουργία διαφορετικών εκδόσεων του ίδιου προγραμμάτος. Κάθε φορά που γίνεται μία αλλαγή στον κεντρικό κώδικα, η έκδοση του αυξάνεται. Ένα από τα πιο διαδεδομένα εργαλεία διαχείρισης εκδόσεων και αποθήκης είναι το SVN, το οποίο επιτρέπει την κεντρική διαχείριση του λογισμικού.

4.1.1 SVN

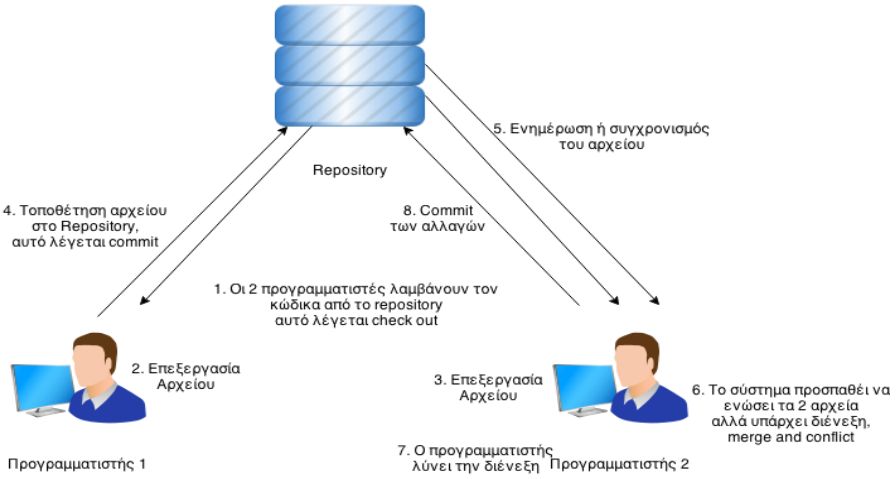
Το subversion είναι ένα λογισμικό ανοιχτού κώδικα το οποίο επιτρέπει την διαχείριση των διαφορετικών εκδόσεων του λογισμικού. Δηλαδή, διαχειρίζεται τις αλλαγές που γίνονται στα αρχεία και στους καταλόγους του λογισμικού. Αυτή η διαχείριση, δίνει την δυνατότητα ανάκτησης παλαιότερης έκδοσης, στην περίπτωση που κάποια αλλαγή στον κώδικα προκαλέσει πρόβλημα.



Σχήμα 4-1 Διαχείρηση Χωρίς Κεντρικό Έλεγχο

Το λογισμικό είναι διαχειρίσιμο μέσω διαδικτύου και μπορεί να χρησιμοποιηθεί από μέλη της ομάδας τα οποία δεν είναι τοποθετημένα στην ίδια περιοχή. Στο σχήμα I παρουσιάζεται η διαχείριση του λογισμικού χωρίς την χρήση κεντρικής αποθήκης, ενώ στο σχήμα II το ίδιο λογισμικό αναπτύσσεται με την χρήση και του SVN.

Παρατηρούμε πως αυτή η ανταλλαγή αρχείων προς όλους τους προγραμματιστές, σε κάθε αλλαγή είναι πολύπλοκη και δεν βοηθάει στο να έχει ο καθένας την πιο πρόσφατη έκδοση του κώδικα.



Σχήμα 4-2 - Διαχείρηση με SVN

Στην εικόνα 2, η διαχείρηση είναι κεντρική. Ο κεντρικός κώδικος είναι αποθηκευμένος στην αποθήκη. Κάθε φορά που ένας προγραμματιστής θέλει να κάνει μία αλλαγή, λαμβάνει (check out) τον κεντρικό κώδικα από την αποθήκη. Κάθε φορά που ο προγραμματιστής θέλει να τοποθετήσει αρχείο στην αποθήκη πρώτα κάνει ενημέρωση, ώστε να εξασφαλίσει ότι διαθέτει την τελευταία έκδοση. Σε περίπτωση, που η αλλαγή του συμπίπτει με αλλαγή στον ίδιο κώδικα από άλλον προγραμματιστή υπάρχει διένεξη (conflict). Είναι στο χέρι του να επιλύσει αυτή την διένεξη και να ενημερώσει τον κεντρικό κώδικα.

4.1.2 Προβλήματα με το SVN

Το κύριο πρόβλημα που χρειάζεται να επιλυθεί με την κεντρική διαχείριση του κώδικα είναι η περίπτωση που ένας προγραμματιστής διαγράψει κατά λάθος τις αλλαγές στον κώδικα ενός άλλου.

Για παράδειγμα το ακόλουθο σενάριο μπορεί να προκύψει :

1. Δύο προγραμματιστές αποφασίζουνε να επεξεργαστούνε το ίδιο αρχείο από την ίδια αποθήκη.
2. Ο προγραμματιστής 1, αποθηκεύει τις αλλαγές του στην αποθήκη πρώτος.
3. Λίγο αργότερα, ο προγραμματιστής 2, διαγράφει τις αλλαγές του πρώτου με την δική του έκδοση του αρχείου.
4. Η έκδοση του πρώτου προγραμματιστή 1 δεν θα χαθεί, αφού το σύστημα κρατάει ιστορικό, αλλά κάθε αλλαγή σε αυτήν την έκδοση δεν είναι εμφανής στην έκδοση του προγραμματιστή 2, μιας και δεν είδε ποτέ τις αλλαγές του πρώτου.
5. Το αποτέλεσμα είναι, όποιες αλλαγές κάνει ο προγραμματιστής 1, στην ουσία θα χαθούνε αφού δεν είναι μέρος της τελευταίας έκδοσης του αρχείου.

Το παραπάνω σενάριο είναι εφικτό να συμβεί, άμα δεν παρθούνε επαρκή μέτρα αντιμετώπισης. Δύο είναι τα πιο συνηθισμένα:

- Κλείδωμα-επεξεργασία-ξεκλείδωμα. Το σύστημα επιτρέπει μόνο σε έναν προγραμματιστή κάθε φορά να επεξεργάζεται το αρχείο, χρησιμοποιώντας κλειδαριές. Λύνεται το πρόβλημα της ταυτόχρονης επεξεργασίας με αυτόν

τον τρόπο. Ωστόσο, αυτή η τεχνική δημιουργεί επιπλέον προβλήματα (ο προγραμματιστής ξέχασε πως έχει κλειδωμένο το αρχείο, δεν γίνεται ο ένας να επεξεργαστεί την αρχή και ο άλλος το τέλος του αρχείου)

- Αντιγραφή-επεξεργασία-συγχώνευση. Κάθε προγραμματιστής δημιουργεί μία τοπική αντιγραφή (εικόνα), του κεντρικού κώδικα από την αποθήκη. Έπειτα, επεξεργάζονται ανεξάρτητα με τον κώδικα των άλλων την δική τους τοπική αντιγραφή. Τέλος, οι τοπικές αντιγραφές ενώνονται σε μία κεντρική, βάση της έκδοσης τους, όπως φαίνεται στην εικόνα 2. Αυτό το μοντέλο εφαρμόζεται στο SVN.

4.2 Αυτόματη Διαδικασία Κτισίματος (Build Automation)

Προτού αναλύσουμε την συνεχή ενσωμάτωση, καλό είναι να γίνει αναφορά στην αυτόματη διαδικασία κτισίματος, αφού αποτελεί ένα από τα βασικά σημεία της και αποτελεί μία τεχνική συνεχόμενης βελτίωσης της ποιότητας του λογισμικού σε συνδυασμό με την διαδικασία ελέγχου έκδοσης.

Ένα από τα προβλήματα της ανάπτυξης λογισμικού είναι ότι η τελική ένωση όλων των επιμέρους συστατικών μπορεί να είναι χρονοβόρα. Αυτό μπορεί να λυθεί ακολουθώντας την διαδικασία αυτόματου κτισίματος. Η διαδικασία αυτή όχι μόνο βοηθάει στο να παρουσιάζει την πρόοδο της ανάπτυξης κάθε στιγμή, αλλά βοηθάει και στην εξάλειψη αρκετών σφαλμάτων αρκετά νωρίς στην όλη διαδικασία ανάπτυξης.

Είναι η διαδικασία με την οποία επιταχύνεται πιο αποτελεσματικό και αποδοτικό κτίσιμο του λογισμικού με σκοπό την ευκολία και ταχύτητα παραγωγής του τελικού αποτελέσματος. Αυτή η διαδικασία δεν αφορά μόνο στον κώδικα που γράφει ο προγραμματιστής αλλά εφαρμόζεται και σε άλλες διαδικασίες όπως :

- Αυτόματο Compiling του κώδικα σε binaries
- Αυτόματη δημιουργία εγκαταστάσημων συστατικων (MSI files, JAR files) από τα binaries
- Αυτόματη δημιουργία και εκτέλεση των διάφορων τεστ
- Αυτόματη ανάπτυξη των παραπάνω έτοιμα για χρήση

Το αποτέλεσμα του άμα είναι λειτουργικό ή όχι το λογισμικό μας φαίνεται μόνο στο τέλος αφού συνδεθούν όλα του τα επιμέρους υποσυστήματα μεταξύ τους, κάτι το οποίο είναι πολύπλοκο, χρονοβόρο και ενδέχεται να δημιουργήσει αρκετά προβλήματα.

Το σημερινό λογισμικό αποτελείται από αρκετά υποσυστήματα, και χωρίς μια αυτόματη διαδικασία ένωσης όλων αυτών, θα μιλούσαμε για μια ιδιαίτερα επίπονη διαδικασία. Η έλλειψη μιας τέτοιας διαδικασίας, θα εμφανιζότανε και στην περίπτωση που θέλαμε να κάνουμε αλλαγές στο τελικό ενωμένο προϊόν, αφού θα χρειαζότανε η εξαρχής διαδικασία κτισίματος. Καταλαβαίνουμε, ότι μία αυτόματη μέθοδος είναι απαραίτητη.

Συνήθως, την όλη διαδικασία αναλαμβάνει ένας build server, ο οποίος είναι υπεύθυνος να εκτελεί τα παραπάνω βήματα. Ο server, κοιτάει ανά τακτά χρονικά διαστήματα, άμα υπάρχει αλλαγή στην τωρινή έκδοση του λογισμικού ώστε να τρέξει το κτίσιμο. Το αποτέλεσμα είναι ότι έχουμε γρήγορη πληροφόρηση για τυχόν λάθη.

Το λογισμικό που τρέχει σε αυτούς τους servers είναι scripts, ένα απλό παράδειγμα είναι τα Makefiles στις εκδόσεις Linux και το Apache Ant στο οποίο θα γίνει εκτενέστερη αναφορά αργότερα. Οι εντολές που περιέχονται σε αυτά τα αρχεία πρέπει να τρέχουνε περιοδικά από τον server ή κάθε φορά που παρατηρείται κάποια αλλαγή, ώστε να βεβαιωθούμε ότι πάντα υπάρχει η τελευταία λειτουργική έκδοση στην αποθήκη. Για να γίνει αυτό χρειάζεται ένας server ιδιαίτερα έξυπνος, και υπάρχουνε αρκετά λογισμικά ειδικά για αυτή την δουλειά. Αργότερα θα αναλυθεί, ένα από τα πιο διαδεδομένα το Jenkins.

4.3 Πρόβλημα Ενσωμάτωσης

Στις μέρες μας η ανάπτυξη ενός λογισμικού είναι πιθανόν να μην γίνεται από μία μόνο ομάδα η οποία είναι τοποθετημένη στην ίδια περιοχή. Διαφορετικά συστατικά του λογισμικού μπορεί να αναπτύσσονται ανεξάρτητα το ένα από το άλλο από ομάδες οι οποίες δεν είναι τοποθετημένες στην ίδια γεωγραφική περιοχή. Επίσης, μέρη του λογισμικού αναπτύσσονται από την ομάδα ως κλειστού κώδικα ενώ άλλα αναπτύσσονται από κοινού σαν λογισμικό ανοιχτού

κώδικα. Οι διαφορετικές διανομές Linux είναι ένα είδος πολύπλοκου λογισμικού ανοιχτού κώδικα, οι οποίες αποτελούνται από χιλιάδες διαφορετικά μέρη, τα οποία είναι αλληλένδετα μεταξύ τους.

Σε περιπτώσεις όπως οι παραπάνω, η παραμικρή αλλαγή σε ένα από τα υποσυστήματα του λογισμικού μπορεί να προκαλέσει αλυσιδωτές αντιδράσεις στο κεντρικό λογισμικό. Αυτό σημαίνει, πως η κάθε αλλαγή πρέπει να ελέγχεται και να δοκιμάζεται διεξοδικά προτού ενσωματωθεί στο κεντρικό λογισμικό.

Στον παραδοσιακό τρόπο ανάπτυξης λογισμικού η ενσωμάτωση και ο διεξοδικός έλεγχος για σφάλματα ήτανε είναι μέρος της τελευταίας φάσης, προκαλώντας πολλαπλά προβλήματα ενσωμάτωσης των διαφορετικών υποσυστημάτων μεταξύ τους.

4.4 Συνεχής Ενσωμάτωση

Τα παραπάνω προβλήματα λύνονται με την συνεχόμενη ενσωμάτωση. Ο όρος ενσωμάτωση σημαίνει την ένωση των διαφορετικών υποσυστημάτων σε μία κεντρική και ελεγχόμενη σύνθεση (configuration) και πρωτοεμφανίστηκε ως διαδικασία του XtremeProgramming. Η συνεχής ενσωμάτωση είναι άρρηκτα συνδεδεμένη με την ποιότητα του λογισμικού, το οποίο μεταφράζεται σε πολλούς και διεξοδικούς ελέγχους για σφάλματα. Όπως έχει ήδη αναφερθεί, οι καινούριες μέθοδοι ανάπτυξης λογισμικού προάγουν τον έλεγχο του κώδικα όσο πιο νωρίς γίνεται, ενώ σε μερικές (Test Driven Development), η δημιουργία των τεστ και ο έλεγχος γίνεται παράλληλα με την ανάπτυξη του λογισμικού.

Ο απώτερος σκοπός της συνεχόμενης ενσωμάτωσης είναι να συνδυάζει τα διαφορετικά μέρη του λογισμικού, σε ένα σταθερό και σωστό αποτέλεσμα, έπειτα από κάθε αλλαγή ενός ή περισσοτέρων υποσυστημάτων. Αυτό σημαίνει, την αυτόματη δημιουργία μιας σταθερής κατάστασης στην οποία έχουνε γίνει όλοι οι απαραίτητοι έλεγχοι ώστε να καλύπτονται τα κριτήρια ποιότητας και να καλύπτει επιτυχώς τα τεστ που έχουνε αναπτυχθεί. Το αποτέλεσμα είναι η κατασκευή ποιοτικού λογισμικού το οποίο πριν δημοσιευθεί έχει περάσει όλα τα επιμέρους τεστ. Από την στιγμή που εγκατασταθεί και αναπτυχθεί αυτή η μεθοδολογία, παίζει πολύ σημαντικό ρόλο στην τελική ποιότητα του λογισμικού, και δίνοντας

την δυνατότητα καταγραφής μετρικών και σταστικών κάθε ενσωμάτωσης προωθεί την μακροπρόθεσμη ποιοτική ανάπτυξη.

Ο συνδυασμός των εκδόσεων των υποσυστημάτων τα οποία έχουνε παραχθεί με την ενσωμάτωση ονομάζεται σύνθεση. Μπορεί να γίνει διαχωρισμός μεταξύ των συνθέσεων οι οποίες καλύπτουνε τις μη λειτουργικές απαιτήσεις (non functional) και σε αυτές που δεν τις καλύπτουνε.

Ένα βασικό μέρος της ενσωμάτωσης είναι η διαδικασία compile και linking των επιμέρους υποσυστημάτων και βιβλιοθηκών για την παραγωγή ενός εκτελέσιμου αρχείου. Κάθε φορά που ένας προγραμματιστής προωθεί τις αλλαγές στην αποθήκη αυτά είναι κάποια από τα βήματα που εκτελούνται αυτόματα.

4.4.1 Διαδικασία Συνεχούς Ενσωμάτωσης

Πρώτα από όλα, ο προγραμματιστής λαμβάνει τον κεντρικό κώδικα από την αποθήκη και δημιουργεί ένα τοπικό αντίγραφο του. Έχοντας στην κατοχή του αυτή την αντιγραφή, μπορεί να επεξεργαστεί τον κώδικα, να προσθέσει καινούρια στοιχεία, να βελτιώσει κάποια κομμάτια του ή ό,τι άλλο είναι απαραίτητο. Επιπλέον, μπορεί να προσθέσει καινούρια τεστ, για να αξιολογήσει την σταθερότητα του υπάρχοντα κώδικα.

Αφού έχει τελειώσει με τις όποιες αλλαγές, εκτελεί ένα αυτόματο κτίσιμο(build) του κώδικα, το οποίο κάνει compile και linking στο τοπικό του αντίγραφο. Αυτή η διαδικασία τρέχει και όλα τα τεστ που έχουνε γραφτεί για το κομμάτι του κώδικα που έχει στην κατοχή του. Για να θεωρηθεί επιτυχές ένα κτίσιμο πρέπει αν περάσει όλα τα τεστ χωρίς λάθη.

Το επόμενο βήμα είναι, και αφού ο προγραμματιστής είναι σίγουρος πλέον πως το αντίγραφο του δεν περιέχει σφάλματα, να προωθήσει τις αλλαγές στην αποθήκη και να τις ενσωματώσει στο κεντρικό κώδικα. Όπως έχει ήδη αναφερθεί, πρώτα συγχρονίζει τον κώδικα του με τον κεντρικό και λύνει τις οποιαδήποτε διενέξεις (conflicts) και ξαναχτίζει τον τοπικό κώδικα. Σε περίπτωση που λόγω των διενέξεων δεν υπάρχει επιτυχές κτίσιμο, είναι ευθύνη του να διορθώσει τα προβλήματα.

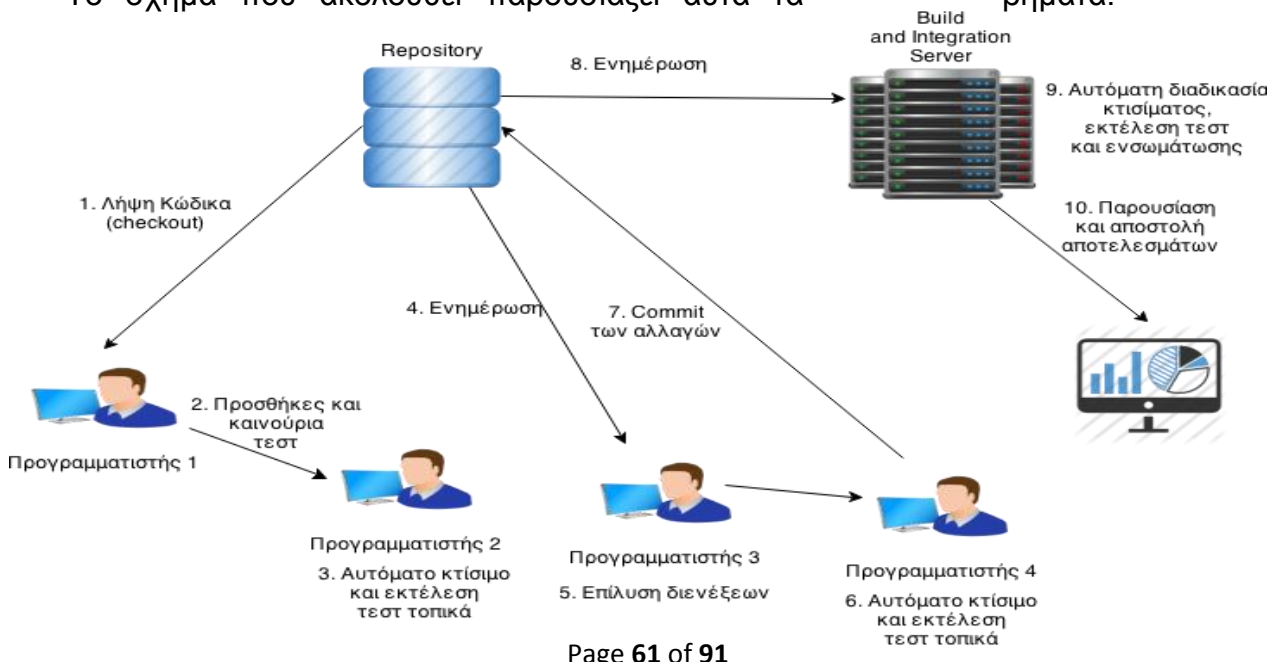
Μόλις, έχει ένα επιτυχές και συγχρονισμένο με τον κεντρικό κώδικα κτίσιμο, προωθεί τις αλλαγές στην αποθήκη.

Αυτό είναι το σημείο που λαμβάνει χώρα η συνεχής ενσωμάτωση. Υπάρχει ένας διακομιστής ο οποίος είναι υπεύθυνος για την ενσωμάτωση (Build and Integration Server). Ο διακομιστής αυτός, αναλαμβάνει πρώτα να τρέξει την αυτόματη διαδικασία κτισίματος και μετά μία σειρά από τεστ (unit και integration tests), και μόνο αφού περάσουνε με επιτυχία ενσωματώνονται οι αλλαγές στον κεντρικό κώδικα. Σε διαφορετική περίπτωση, ενημερώνονται οι προγραμματιστές ώστε να διορθώσουνε τα λάθη. Αυτή είναι η κύρια λειτουργία της συνεχόμενης ενσωμάτωσης. Σε κάθε αλλαγή του κώδικα, να εξασφαλίζεται πως ο κεντρικός κώδικας είναι σε σταθερή και λειτουργική κατάσταση, καθαρός από οποιαδήποτε σφάλματα.

Από ότι καταλαβαίνουμε, τα τεστ που θα τρέξουνε στον διακομιστή πρέπει να είναι ήδη γραμμενά από τους δοκιμαστές. Η αντιμετώπιση των σφαλμάτων όσο πιο νωρίς στην διαδικασία ανάπτυξης λογισμικού γίνεται τόσο μικρότερες παρενέργειες έχει στην συνέχεια. Θα ήτανε πολύ πιο δύσκολα να διορθωθούνε λάθη κατά την διάρκεια της τελευταίας φάσης ανάπτυξης, καθώς θα σήμαινε περισσότερο χρόνο δουλειάς, περισσότερα έξοδα καθώς και πιο δύσκολα αντιμετωπίσιμα σφάλματα.

Το κυριότερο στοιχείο επιτυχίας της συνεχόμενης ενσωμάτωσης είναι ότι προλαμβάνονται σφάλματα και προβλήματα προτού αυτά εκδηλωθούν στο τέλος της ανάπτυξης.

Το σχήμα που ακολουθεί παρουσιάζει αυτά τα βήματα.



Σχήμα 4-3 - Συνεχής Ενσωμάτωση

Το τελικό αποτέλεσμα, είναι κεντρικός κώδικας, πλήρως ελεγχμένος για λάθη. Σε περίπτωση κάποιου σφάλματος, δημιουργείται μία αναφορά και αποστέλεται στον προγραμματιστή. Υπάρχει, ταχεία ενημέρωση για οποιοδήποτε σφάλμα, κάνοντας πιο εύκολη την διαδικασία επίλυσης του.

4.4.2 Μειονεκτήματα της συνεχής ενσωμάτωσης

Παρά το ότι αυτή η διαδικασία έχει αρκετά πλεονεκτήματα υπάρχουνε μερικα αρνητικά στοιχεία που αποτρέπουνε τις ομάδες να ακολουθήσουνε αυτόν τον τρόπο ανάπτυξης :

- Είναι πολύπλοκο να στηθεί η όλη διαδικασία. Χρειάζεται αρκετή ώρα και τεχνικές γνώσεις για να εγκατασταθεί και να λειτουργεί πλήρως.
- Για να τελειοποιηθεί η διαδικασία, το κτίσιμο, η εκτέλεση των τεστ και οι αναφορές να γίνονται αυτόματα, αυτό προϋποθέτει την αρκετά καλή γνώση μιας scripting γλώσσας.
- Χρειάζεται την στήριξη της ίδιας της ομάδας, αλλάζοντας την φιλοσοφία της, ώστε να εκτελεί συχνά αυτή την διαδικασία.
- Απαιτούμενο είναι η δημιουργία αρκετών τεστ, τα οποία δοκιμάζουνε σε ικανοποιητικό βαθμό τον κώδικα.

Παρατηρούμε πως το μεγαλύτερο αρνητικό είναι η αρχική ρύθμιση όλων των παραμέτρων. Με την λύση αυτού του προβλήματος τα θετικά είναι πολλά περισσότερα όπως είδαμε και επιπλέον έχουμε καλύτερη διαχείριση ολόκληρου του project ενώ είναι ευκολότερο να πετύχουμε υψηλότερη ποιότητα (περισσότερα τεστ, μετρικές). Το συμπέρασμα είναι, πως αξίζει να αφιερώσει κάποιος χρόνο για την εγκατάσταση και σωστή λειτουργία της διαδικασίας.

4.5 Εργαλείο Jenkins Παρουσίαση

Για την εκτέλεση των παραπάνω βημάτων υπάρχουνε αρκετά εργαλεία. Ένα από τα πιο διαδεδομένα και πολυ χρησιμοποιημένα είναι το Jenkins. Είναι ένα Java based ανοιχτού κώδικα εργαλείο σχεδιασμένο για να τρέχει επάνω σε έναν διακομιστή. Το συγκεκριμένο εργαλείο συναντάτε σχεδόν σε όλα τα μεγάλα projectσ για τον καλύτερο έλεγχο και για το εντυπωσιακό πλήθος αποτελεσμάτων που παράγει.

4.5.1 Ιστορία

Πριν το εργαλείο Jenkins υπήρχε το εργαλείο ονόματι Hudson. Το Hudson αναπτύχθηκε από την Sun Microsystems και πρωτοκυκλοφόρησε το 2005. Στα επόμενα χρόνια κέρδισε αρκετά βραβεία ανάπτυξης λογισμικού. (Smart, 2011)

Την χρονιά 2010, η κοινότητα που χρησιμοποιούσε το Hudson, άρχισε να έχει αμφιβολίες για το πώς το ανέπτυξε πλέον η Oracle. Η σύγκρουση αυτή, γέννησε το Jenkins. Από τότε τα 2 εργαλεία χρησιμοποιούνται κατά κόρων και αναπτύσσονται παράλληλα.

4.5.2 Χαρακτηριστικά

Παρουσιάζονται τα βασικότερα χαρακτηριστικά του:

- Εύκολο στην εγκατάσταση.
- Εύκολο στην ρύθμιση του. Παρέχει αναλυτική διεπαφή χρήστη για τις ρυθμίσεις του.
- Επαρκές υποστήριξη και ενεργή κοινότητα.
- Πλήθος plugins για συλλογή αρκετών μετρικών και στατιστικών του κώδικα.
- Αποστολή email, μετά από κάθε ενσωμάτωση.
- Παράλληλο κτίσιμο και εκτέλεση. Μπορεί να ρυθμιστεί, ώστε κάθε φορά τα τεστ, και το κτίσιμο να εκτελούνται σε πολλούς υπολογιστές, ώστε να βελτιωθεί η απόδοση και να έχουμε το λογισμικό μας να δοκιμάζεται σε διαφορετικά λειτουργικά και ρυθμίσεις.
- Υποστήριξη με αρκετά λογισμικά και πλατφόρμες.
- Υποστήριξη διαφορετικών μεθόδων ελέγχου κώδικα (git και svn).

4.5.3 Προαπαιτούμενα

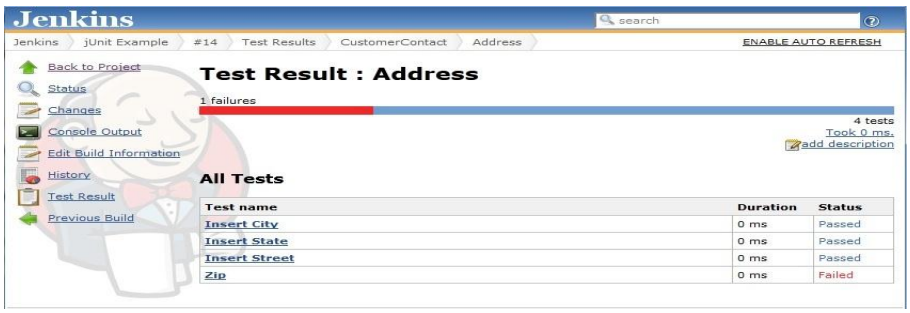
Για την ορθή λειτουργία του εργαλείου χρειάζεται να υπάρχει ένας διακομιστής (TomCat, Apache), ένα εργαλείο κτισίματος και γνώση μίας scripting γλώσσας (Maven, Ant) και ένα εργαλείο διαχείρισης εκδόσεων (Git, SVN).

4.5.4 Αποτελέσματα και Ασφάλεια

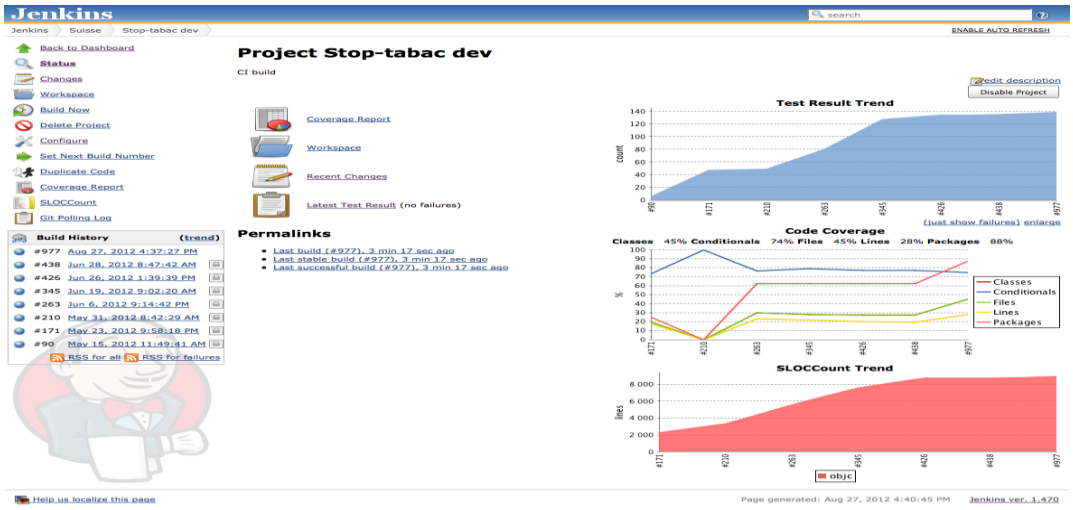
Με την εκτέλεση ενός πλήρους κύκλου συνεχούς ενσωμάτωσης το εργαλείο μπορεί να παράξει αρκετά αποτελέσματα, σχετικά με το άμα ο κώδικας πέρασε τα τεστ, αποτελέσματα σχετικά με την πολυπλοκότητα του κώδικα (γραμμές κώδικα, στατική ανάλυση κώδικα), κάλυψη του κώδικα, μετρικές σχετικά με την ποιότητα του κώδικα (στυλ γραφής, σχόλια, διπλός κώδικας), ιστορικό αλλαγών. Η λίστα των αποτελεσμάτων και των γραφημάτων είναι αρκετά μεγάλη και γίνεται μεγαλύτερη με την χρήση των plugins. Στο σχήμα 4 βλέπουμε ένα ενδεικτικό αποτέλεσμα, παρατηρούμε πως από τα τέσσερα τεστ που εκτελέστηκαν, ο κώδικας πέρασε τα τρία, το οποίο σημαίνει ότι πρέπει να υπάρξουνε διορθώσεις. Στο σχήμα 5 παρουσιάζονται πιο αναλυτικά αποτελέσματα, όπως ποσοστό κάλυψης, γραμμές κώδικα.

Το εργαλείο επίσης παρέχει ρυθμίσεις ασφαλείας, για το ποιοι χρήστες έχουνε πρόσβαση στα αποτελέσματα, ποιοι μπορούνε να αλλάξουνε τις παραμέτρους του.

Με την χρήση των αποτελεσμάτων, η ομάδα ανάπτυξης μπορεί να καταλήξει σε συμπέρασμα σχετικά με την ποιότητα του λογισμικού μας, και σε περίπτωση στην οποία δεν εκπληρώνονται οι αρχικές προδιαγραφές ποιότητας, σημαίνει πως απαιτούνται αλλαγές.



Σχήμα 4-4 Jenkins Πρώτο Ενδεικτικό Αποτέλεσμα



Σχήμα 4-5 Jenkins Αναλυτικά Αποτελέσματα

Το παραπάνω κεφάλαιο ανέλυσε την διαδικασία της συνεχούς ενσωμάτωσης, δίνοντας έμφαση στα επιμέρους στοιχεία της διαδικασίας. Αναφορά έγινε στον έλεγχο έκδοσης και στην αυτόματη διαδικασία κτισίματος, τα οποία είναι απαραίτητα για την συνεχή ενσωμάτωση. Αναφέρθηκαν, τα προβλήματα τα οποία επικαλείται να λύσει αυτή η μεθολογία ενώ παρουσιάστηκαν και κάποια αρνητικά που υπάρχουνε.

Τέλος, έγινε παρουσίαση του εργαλείου Jenkins, το οποίο είναι αυτό το οποίο χρησιμοποιείται περισσότερο για αυτή την διαδικασία.

Κεφάλαιο 5 Οδηγός εργαλείου Jenkins

Στο προηγούμενο κεφάλαιο έγινε μία εισαγωγή στο πρόγραμμα Jenkins, το οποίο είναι αυτό που χρησιμοποιείται κατά κόρον για την συνεχή ενσωμάτωση. Σε αυτό το κεφάλαιο θα γίνει μία βήμα προς βήμα συνεχής ενσωμάτωση ενός λογισμικού. Θα αναλυθούνε τα βήματα που χρειάζονται να ακολουθήσει κάποιος ώστε να επωφεληθεί από τα οφέλη της συνεχής ενσωμάτωσης.

Προτού προχωρήσουμε στον οδηγό καλό είναι να αναφερθεί πως η συνεχής ενσωμάτωση δεν αποτελείται από μία μόνο φάση. Συνήθως μέχρι τον

ολοκληρωτικά αυτοματοποιημένο τρόπο ανάπτυξης υπάρχουνε τα εξής στάδια : (Smart, 2011)

1. Έλλειψη διακομιστή ενσωμάτωσης

   Αρχικά, η ομάδα ανάπτυξης δεν βασίζεται σε κάποιον κεντρικό διακομιστή ο οποίος εκτελεί το αυτόματο κτίσιμο (build). Το λογισμικό αναπτύζεται τοπικά από τον προγραμματιστή. Κάποιο ant script μπορεί να χρησιμοποιείται για την διευκόλυνση του, ενώ μπορεί να υπάρχει κεντρική αποθήκη κώδικα, αλλά οι αλλαγές να μην προωθούνται τακτά ή βάση κάποιου προγράμματος.

2. Νυχτερινά Κτισίματα (Nightly Builds)

   Σε αυτό το στάδιο, υπάρχει ένας διακομιστής που αναλαμβάνει το αυτόματο κτίσιμο, κάτι το οποίο συνήθως γίνεται το βράδυ, εξού και η ονομασία. Σε αυτό το στάδιο, δεν υπάρχουν κάποια τεστ να τρέχουνε μαζί με το κτίσιμο ή και να υπάρχουνε δεν είναι απαραίτητο να εκτελούνται κάθε φορά. Ωστόσο, οι προγραμματιστές ενσωματώνουνε βάση προγράμματος τον κώδικα τους στην κεντρική αποθήκη. Σε περίπτωση κάποιου λάθους ή κάποιου conflict οι προγραμματιστές ειδοποιούνται με email. Σε αυτή την φάση, τα μέλη της ομάδας δεν θεωρούνε υποχρεωτικό να διορθώσουνε άμεσα ένα σφάλμα.

3. Νυχτερινά Κτισίματα με Αυτόματα Τεστ

   Σε αυτό το στάδιο αρχίζει να διαμορφώνεται το αίσθημα της συνεχής ενσωμάτωσης στην ομάδα. Ο διακομιστής είναι ρυθμισμένος να εκτελεί το όποιο script έχει κάθε φορά που παρατηρείται αλλαγή στον κώδικα της κεντρικής αποθήκης. Επίσης, αρκετά τεστ ενσωμάτωσης εκτελούνται σε αυτό το στάδιο κάθε φορά και τα μέλη της ομάδας είναι πιο συνειδητοποιημένα ώστε να διορθώσουνε άμεσα τα οποιαδήποτε σφάλματα.

4. Μετρικές

Αυτό το στάδιο παράγει αποτελέσματα σχετικά με την ποιότητα του κώδικα, χρησιμοποιώντας διάφορες μετρικές (γραμμές κώδικα, διπλός – νεκρός κώδικας). Είναι πιθανό να δημιουργείται και η αντίστοιχη τεκμηρίωση (documentation) του κώδικα. Αυτά τα βήματα προσφέρουν την δυνατότητα στην ομάδα να βελτιώσει τον κώδικα του λογισμικού.

5. Περισσότερα Τεστ

   Σε αυτό το στάδιο, η διαδικασία δημιουργίας τεστ είναι πιο σοβαρή. Η ομάδα πλέον μπορεί να ακολουθεί την Test Driven Development διαδικασία. Ο κώδικας πλέον, πρέπει να περνάει όλα τα τεστ ώστε να ενσωματώνεται στην κεντρική αποθήκη.

6. Αυτοματοποιημένη Ανάπτυξη

   Πλέον δημιουργούνται τεστ στο επίπεδο συστήματος ή αποδοχής και ανάλογες τεχνικές ανάπτυξης χρησιμοποιούνται από την ομάδα.

7. Συνεχόμενη Ανάπτυξη

   Στο τελευταίο στάδιο όλα τα τεστ εκτελούνται και η ανάπτυξη του λογισμικού βασίζεται στα αποτελέσματα του διακομιστή ενσωμάτωσης. Είναι εύκολο για την ομάδα να γυρίσει σε προηγούμενη έκδοση του λογισμικού, χωρίς ιδιαίτερη προσπάθεια.

5.1 Προαπαιτούμενα Εργαλεία

Σε αυτό το κεφάλαιο θα γίνει αναφορά στα εργαλεία που χρειάζονται για την συνεχή ενσωμάτωση. (Bergman, 2011)

Πρώτα από όλα το περιβάλλον που θα χρησιμοποιηθεί είναι η έκδοση Lubuntu 12.04 με αριθμό kernel 3.2.14, η οποία τρέχει σε virtual περιβάλλον χρησιμοποιώντας το πρόγραμμα VmWare player. Με την εγκατάσταση του Lubuntu εγκαθιστούμε και τα build essentials tools.

Η εφαρμογή στην οποία θα βασιστεί αυτός ο οδηγός αποτελείται από κώδικα γραμμένο σε PHP 5.3 χρησιμοποιώντας το default template του framework Symfony2 έκδοση 2.15. Αυτό το framework δημιουργήθηκε το 2005 και πλεόν

θεωρείται από τα καλύτερα, πιο γρήγορο και πιο ευέλικτο framework για ανάπτυξη εφαρμογών διαδικτύου. Ο editor που θα χρησιμοποιηθεί είναι το εργαλείο NetBeans έκδοση 8 ενώ απαραίτητη είναι η έκδοση Java SDK και Java jre 7.

Ο διακομιστής που θα τρέχει την εφαρμογή είναι ο γνωστός Apache2.

Το εργαλείο που θα χρησιμοποιήσουμε για να δημιουργήσουμε το αυτόματο κτίσιμο είναι το Ant. Στην ουσία το script που θα αναπτύξουμε είναι ο συνδετικός κρίκος ανάμεσα στα διαφορετικά κομμάτια της συνεχής ενσωμάτωσης. Αυτό το script, αναφέρει βήμα βήμα το ποιες εντολές και προγράμματα θα εκτελεστούνε, με ποιες παραμέτρους και το που θα αποθηκευτούνε τα αποτελέσματα τους. Είναι το script που θα εκτελεί το Jenkins.

Το εργαλείο για την δημιουργία κεντρικής αποθήκης και διαχείρισης κώδικα είναι το git έκδοση 1.7.9.5. Είναι από τα πιο διαδεδομένα εργαλεία που προσφέρει άριστη συνεργασία ανάμεσα στα υπόλοιπα εργαλεία.

Τα εργαλεία που θα μας βοηθήσουνε για την εκτέλεση των test και για την παραγωγή των μετρικών είναι τα εξής :

- PhpUnit, έκδοση 4.2. Αυτό το εργαλείο είναι το κυρίαρχο για την δημιουργία τεστ στο επίπεδο μονάδας για εφαρμογές PHP, ενώ παρέχει και αρκετές μετρικές όπως κάλυψη κώδικα, για την οποία θα χρειαστούμε το εργαλείο xdebug.so
- PHP_CodeSniffer, έκδοση 1.5.4. Το συγκεκριμένο εργαλείο επικυρώνει την ορθότητα του κώδικα μας βάση κάποιων προτύπων (στην περίπτωση μας πρότυπο του Symfony2 CodeStandards).
- PHPLOC, έκδοση 2.0.6. Αυτό το εργαλείο μετράει το μέγεθος του κώδικα και την αναλύει την γενική δομή του.
- PHP_Depend (pdepend), έκδοση 1.1.4. Αυτό το εργαλείο πραγματοποιεί στατική ανάλυση στον κώδικα παράγοντας διάφορες μετρικές σχετικά με την πολυπλοκότητα του.
- PHP Mess Detector (PHPMD), έκδοση 1.5.0. Αυτό το εργαλείο αναλύει τον πηγαίο κώδικα ψάχνοντας για διάφορα σφάλματα, όπως μη επιστρεφόμενη τιμή σε συνάρτηση που κανονικά θα έπρεπε, μη βελτιστοποιημένος κώδικας, νεκρός κώδικας.

- PHP Copy/Paste Detector (PHPCPD), έκδοση 2.0.1. Αυτό το εργαλείο μας δίνει το ποσοστό κώδικα ο οποίος είναι αντιγραφή από άλλο σημείο του.
- PHP_CodeBrowser (PHPCB), έκδοση 1.1.1. Αυτό το εργαλείο αναλύει τον κώδικα και δημιουργεί μία ιστοσελίδα για την ευκολότερη εξερεύνηση του κώδικα.

Κάθε ένα από τα παραπάνω εργαλεία παράγει ένα αποτέλεσμα, είτε αρχείο xml, είτε ιστοσελίδα, είτε εικόνες. Σκοπός του Jenkins είναι, μετά την ενορχηστρωμένη εκτέλεση (ant), αυτών των προγραμμάτων να αναλύσει και να μας παρουσιάσει τα αποτελέσματα τους. Σε περίπτωση λάθους ή μη ικανοποιητικών αποτελεσμάτων στις μετρικές οι προγραμματιστές ενημερώνονται.
Για την ορθή αξιοποίηση των αποτελεσμάτων αυτών, υπάρχουνε τα αντίστοιχα plugins του Jenkins, τα οποία σαν τιμή εισόδου έχουνε τα αποτελέσματα των παραπάνω προγραμμάτων. Αυτά τα plugins παρουσιάζονται στον πίνακα 1

Πίνακας 1 - Plugins

Πρόγραμα PHP	Jenkins Plugin
PhpUnit	xUnit
PhpUnit Κάλυψη Κώδικα	Clover PHP
PHP_Depend	JDepend
PHP_CodeSniffer	CheckStyle
PHP Copy/Paste Detector	DRY
PHPLOC	Plot
PHP Mess Detector	PMD
PHP_CodeBrowser	HTML Publisher

Εργαλείο Jenkins έκδοση 1.5.74. Αυτό είναι το κύριο εργαλείο για την συνεχόμενη ενσωμάτωση. Τα παραπάνω plugins εγκαθίστανται μέσα από το περιβάλλον του, όπως θα αναλυθεί πιο κάτω.

5.2 Δημιουργία Project και Κεντρικής Αποθήκης

Σε αυτό το κεφάλαιο θα παρουσιαστεί ένας βήμα προς βήμα οδηγός για την εγκατάσταση των παραπάνω εργαλείων.

Θεωρείτε πως υπάρχει ήδη ένα μηχάνημα με την έκδοση Lubuntu εγκατεστημένη μαζί με τα εργαλεία build-essentials, curl και την έκδοση Java 7 (SDK και jre). Τα προγράμματα αυτά τα εγκαθιστούμε με την χρήση της εντολής :

```
sudo apt-get install [όνομα προγράμματος].
```

Το επόμενο βήμα είναι η εγκατάσταση του διακομιστή μας, που όπως αναφέρθηκε είναι ο apache, μαζί με τα πακέτα PHP και MySql. Η απαιτούμενη εντολή είναι :

```
sudo apt-get install lamp-server^ .
```

Για την ευκολία μας, αλλάζουμε τα δικαιώματα στον φάκελο /var/www με την εντολή sudo chmod 777 /var/www. Σε αυτόν τον φάκελο θα αναπτύξουμε το project μας.
Επίσης εγκαθιστούμε τα πακέτα php5-intl και php5-xsl, τα οποία είναι απαιτούμενα για την συλλογή κάποιων μετρικών, χρησιμοποιώντας την ίδια διαδικασία.
Για το εργαλείο ant, το οποίο θα χρησιμοποιήσουμε για να ενορχηστρώσουμε την αυτόματη διαδικασία χτισίματος, εκτελούμε την εντολή :

```
sudo apt-get install ant.
```

Το ίδιο ισχύει και για το πρόγραμμα git, το οποίο θα μας βοηθήσει στην δημιουργία και την επικοινωνία με την κεντρική αποθήκη.
Για την προσθήκη του προγράμματος NetBeans, κατεβάζουμε το πρόγραμμα (έκδοση 8, PHP, HTML development) το κάνουμε εκτελέσιμο με την εντολή chmod +x [όνομα προγράμματος] και από την γραμμή εντολών τρέχουμε το αρχείο που κατεβάσαμε με ./[όνομα αρχείου].
Το επόμενο βήμα είναι η εγκατάσταση του framework Symfony2. Οι εξής διαδικασίες απαιτούνται :

I. `curl -sS https://getcomposer.org/installer | php`. Για την όλη διαδικασία θα χρησιμοποιήσουμε το εργαλείο composer, το οποίο

αναλαμβάνει την διαχείριση των επιμέρους υποσυστημάτων που είναι απαραίτητα για το Symfony2.

II. Για την εκτέλεση του από οποιαδήποτε τοποθεσία εκτελούμε : `sudo mv composer.phar /usr/local/bin/composer`

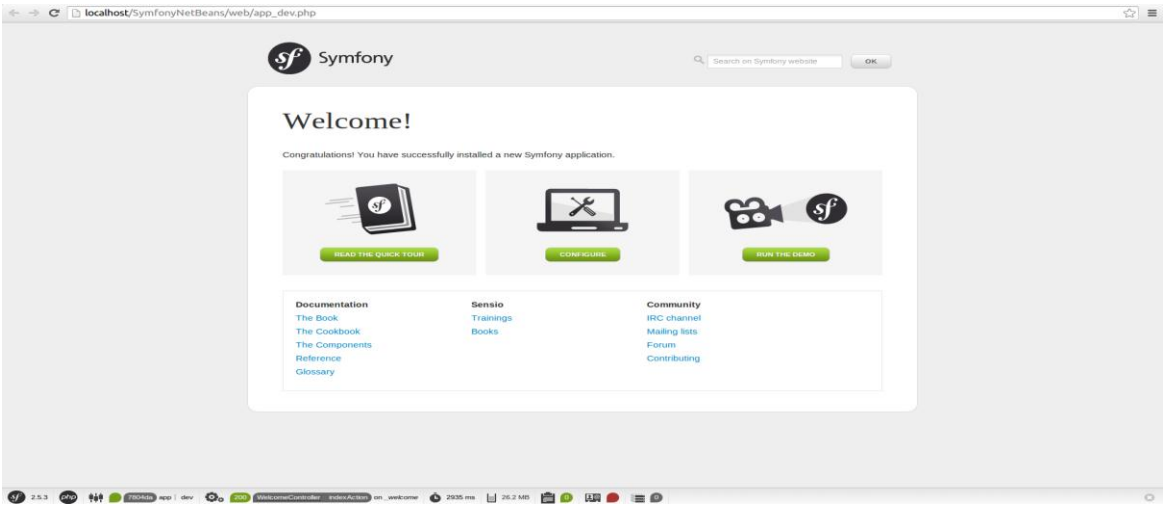
III. Κατεβάζουμε το πακέτο symfony2 από : http://symfony.com/download?v=Symfony_Standard_Vendors_2.5.3.zip

Πλέον έχουμε στον υπολογιστή μας το Symfony2.

Για την δημιουργία του πρώτου μας project, ανοίγουμε το NetBeans, επιλέγουμε new project, PHP, βάζουμε ένα όνομα στο Project Name και στο πεδίο Sources Folder βάζουμε τον φάκελο `/var/www`, στο πεδίο Run As επιλέγουμε το Local Web Site και σαν starting url επιπλέον προσθέτουμε το `web/app_dev.php`.

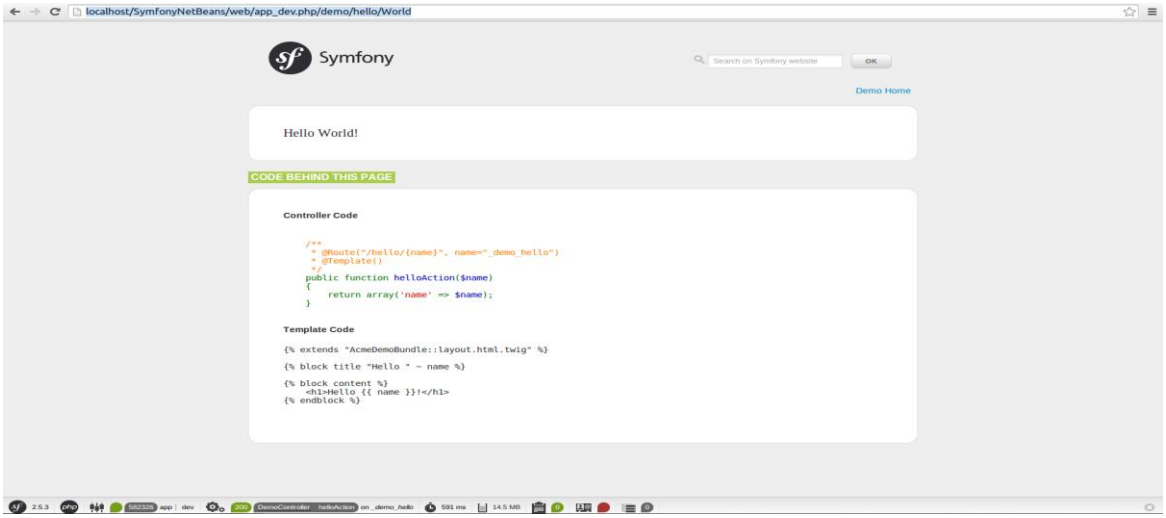
Στην επόμενη καρτέλα επιλέγουμε Symfony2 framework, options και στο πεδίο Symfony βάζουμε το αρχείο που κατεβάσαμε στο προηγούμενο βήμα III. Στη τελευταία καρτέλα στην επιλογή options και Composer βάζουμε το /usr/local/composer, πατάμε Finish και περιμένουμε.

Αν όλα πήγανε όπως πρέπει όταν εκτελέσουμε το project πρέπει να ανοίξει ένας browser όπως φαίνεται στην εικόνα 1.



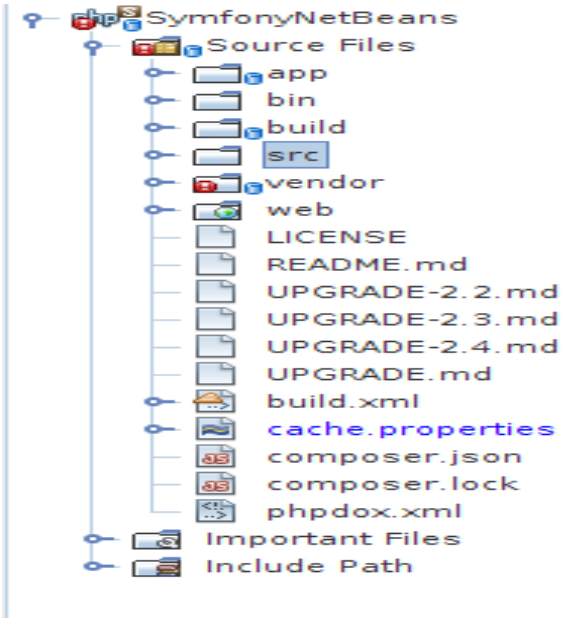
Εικόνα 1 Symfony Πρώτη Οθόνη

Πηγαίνοντας στην διεύθυνση `localhost/[όνομαproject]/web/app_dev.php/demo/hello/World`, η εικόνα2 εμφανίζεται.



Εικόνα 2 – Οθόνη Hello World

Στην εικόνα3 παρουσιάζεται η δομή του project μας μέχρι στιγμής.



Εικόνα 3 – Δομή Project

Η εικόνα αποτυπώνει και το πού θα τοποθετήσουμε το αρχείο build.xml, το οποίο θα εκτελείται από το εργαλείο ant.

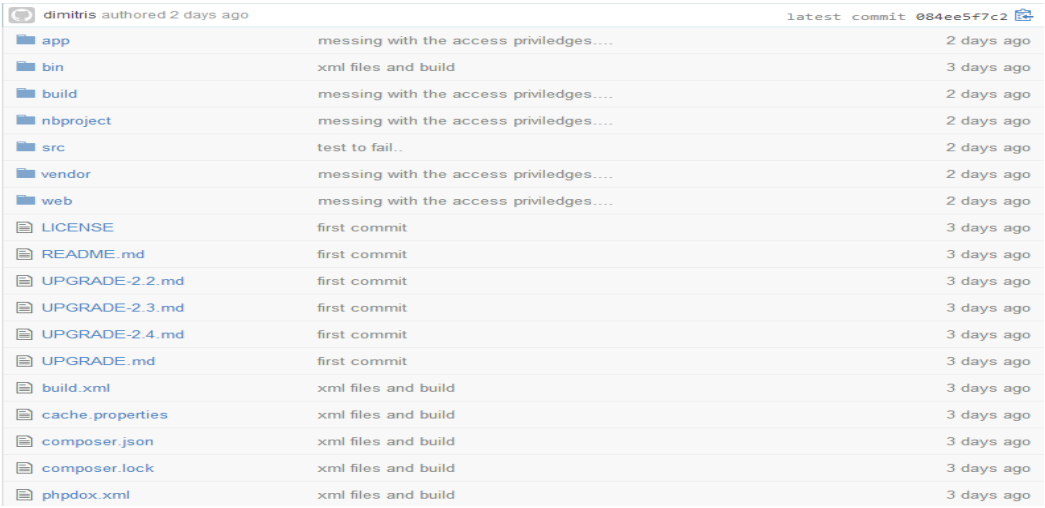
Το επόμενο στάδιο είναι η δημιουργία κεντρικής αποθήκης με το πρόγραμμα git. Μέσω του NetBeans θα γίνεται η διαχείριση της. Πρώτα από όλα πρέπει να δημιουργήσουμε μία τοπική, και μετά μία η οποία θα είναι στο διαδίκτυο.

Για το πρώτο βήμα επιλέγουμε το Team και την επιλογή git init. Με αυτή την διαδικασία δημιουργήσαμε μία τοπική αποθήκη για την διαχείρηση του κώδικα μας. Με δεξί κλικ στο project μας επιλέγουμε git και commit, ώστε να προωθήσουμε τον κώδικα μας στην αποθήκη αυτή. Μας δίνετε η επιλογή το ποια αρχεία να διαχειρίζεται το πρόγραμμα git, καλό είναι να τα επιλέξουμε όλα. Σε αυτή την καρτέλα κάθε φορά που θα κάνουνε commit θα εμφανίζονται τα αρχεία και οι φάκελοι που αλλάξανε από την τελευταία φορά. Βάζουμε και ένα σχόλιο, μιας και είναι απαραίτητο.

Για το δεύτερο βήμα θα χρησιμοποιήσουμε το site github. Δημιουργούμε λογαριασμό και επιλέγουμε να δημιουργήσουμε ένα καινούριο repository. Κρατάμε το όνομα που θα του δώσουμε (στην περίπτωση μας JenkinsTest). Επίσης στην διαχείριση λογαριασμού βάζουμε και έναν κωδικό.

Πλέον έχουμε δημιουργήσει μία κεντρική αποθήκη στο διαδίκτυο, την οποία θα χρησιμοποιεί το εργαλείο Jenkins για να ελέγχει τις αλλαγές στον κώδικα μας.

Στο NetBeans επιλέγουμε ξανά το project και το git, μόνο που τώρα επιλέγουμε remote και push. Στο πεδίο specify Git repository location βάζουμε την διεύθυνση που μας έδωσε το github, βάση του ονόματος που βάλαμε, στην περίπτωση μας https://github.com/drentas/JenkinsTest.git. User και password βάζουμε αυτά που βάλαμε και στην δημιουργία λογαριασμού στο github. Στην επόμενη καρτέλα επιλέγουμε το master και το ίδιο στην επόμενη. Πλέον το project μας είναι αποθηκεύμενο στο διαδίκτυο, όπως φαίνεται στην εικόνα 4.



dimitris authored 2 days ago		latest commit 084ee5f7c2
app	messing with the access priviledges....	2 days ago
bin	xml files and build	3 days ago
build	messing with the access priviledges....	2 days ago
nbproject	messing with the access priviledges....	2 days ago
src	test to fail..	2 days ago
vendor	messing with the access priviledges....	2 days ago
web	messing with the access priviledges....	2 days ago
LICENSE	first commit	3 days ago
README.md	first commit	3 days ago
UPGRADE-2.2.md	first commit	3 days ago
UPGRADE-2.3.md	first commit	3 days ago
UPGRADE-2.4.md	first commit	3 days ago
UPGRADE.md	first commit	3 days ago
build.xml	xml files and build	3 days ago
cache.properties	xml files and build	3 days ago
composer.json	xml files and build	3 days ago
composer.lock	xml files and build	3 days ago
phpdox.xml	xml files and build	3 days ago

Εικόνα 4 – Project στο git

5.4 Εγκατάσταση εργαλείων PHP

Όπως έχει ήδη αναφερθεί, για την συλλογή των μετρικών και των αποτελεσμάτων των δοκιμών διάφορα εργαλεία χρειάζονται.

Για την ευκολότερη λήψη μερικών εξ αυτών, εγκαθιστούμε το πρόγραμμα php-pear, με την εντολή:

```
sudo apt-get install php-pear
```

το οποίο είναι μια κεντρική αποθήκη διαχείρησης εργαλείων για την γλώσσα php.

Ο τρόπος εγκατάστασης για το κάθε ένα ξεχωριστά είναι :

- PhpUnit
  I. `wget https://phar.phpunit.de/phpunit.phar`
  II. `chmod +x phpunit.phar`
  III. `sudo mv phpunit.phar /usr/local/bin/phpunit` (για την εκτέλεση του προγράμματος από οποιοδήποτε σημείο).
- Για την δημιουργία Code Coverage
  I. `sudo apt-get install php5-xdebug php5-dev`
- CodeSniffer και ενεργοποίηση standards για το Symfony2
  I. `sudo pear install PHP_CodeSniffer`

II. `pear config-show | grep php_dir`
III. `cd /usr/share/php (ή σε όποιο path είναι το php_dir)`
IV. `cd PHP/CodeSniffer/Standards`
V. `sudo git clone git://github.com/opensky/Symfony2-coding-standard.git Symfony2`
VI. `sudo phpcs –config-set default_standar Symfony2`

- phpLoc
  I. `wget https://phar.phpunit.de/phploc.phar`
  II. `chmod +x phploc.phar`
  III. `sudo mv phploc.phar /usr/local/bin`
- phpDepend
  I. `sudo pear channel-discover pear.pdepend.org`
  II. `pear remote-list –c pdepend`
  III. `sudo pear install pdepend/PHP_Depend`
- phpmd
  I. `sudo pear channel-discover pear.phpmd.org`
  II. `pear remote-list –c phpmd`
  III. `sudo pear install phpmd/PHP_PMD`
- phpcpd
  I. `wget https://phar.phpunit.de/phpcpd.phar`
  II. `chmod +x phpcpd.phar`
  III. `mv phpcpd.phar /usr/local/bin/phpcpd`
- phpcb
  I. `https://github.com/Mayflower/PHP_CodeBrowser/releases/download/1.1.1/phpcb-1.1.1.phar`
  II. `Chmod +x phpcb-1.1.1.phar`
  III. `mv phpcb.phar /usr/local/phpcb`

Τα παραπάνω εργαλεία είναι αυτά των οποίων τα αποτελέσματα θα τροφοδοτήσουνε τα plugins του Jenkins.

Αρκετά από τα παραπάνω εργαλεία χρειάζονται και ένα αρχείο xml, από το οποίο θα διαβάσουνε το κατάλληλο configuration (σε ποιον φάκελο είναι τα τεστ που πρέπει να εκτελεστούνε, πού να αποθηκευτούνε τα αποτελέσματα...).

Για το εργαλείο PhpUnit δημιουργούμε στον φάκελο app του project μας, ένα αρχείο xml με ονομασία phpunit.xml.dist, σε περίπτωση που υπάρχει κάποιο αρχείο με όνομα phpunit.xml ήδη το σβήνουμε. Κάθε φορά που εκτελούμε την εντολή phpunit, από αυτό το αρχείο θα λαμβάνει τις απαραίτητες παραμέτρους.

Στην περίπτωση μας το αρχείο αυτό φαίνεται στην εικόνα 5

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

<!-- http://www.phpunit.de/manual/current/en/appendixes.configuration.html -->
<phpunit
  backupGlobals               = "false"
  backupStaticAttributes      = "false"
  colors                      = "true"
  convertErrorsToExceptions   = "true"
  convertNoticesToExceptions  = "true"
  convertWarningsToExceptions = "true"
  processIsolation            = "false"
  stopOnFailure               = "false"
  syntaxCheck                 = "false"
  bootstrap                   = "bootstrap.php.cache" >

    <testsuites>
        <testsuite name="Project Test Suite">
            <directory>../src/*/*Bundle/Tests</directory>
            <directory>../src/*/Bundle/*Bundle/Tests</directory>
        </testsuite>
    </testsuites>

    <logging>
        <log type="coverage-html" target="build/coverage" title="BankAccount" charset="UTF-8"
             yui="true" highlight="true"
       lowUpperBound="35" highLowerBound="70"/>
        <log type="coverage-clover" target="build/logs/clover.xml"/>
        <log type="junit" target="build/logs/junit.xml" logIncompleteSkipped="false"/>
        <log type="coverage-crap4j" target="build/logs/crap4j.xml"/>
    </logging>


    <filter>
        <whitelist>
            <directory>../src</directory>
            <exclude>
                <directory>../src/*/*Bundle/Resources</directory>
                <directory>../src/*/*Bundle/Tests</directory>
                <directory>../src/*/Bundle/*Bundle/Resources</directory>
                <directory>../src/*/Bundle/*Bundle/Tests</directory>
            </exclude>
        </whitelist>
    </filter>

</phpunit>
```

Εικόνα 5 – phpunit.xml

Το αρχείο bootstrap.php.cache, δημιουργείται αυτόματα από το framework (υπάρχει πιθανότητα το bootstrap αρχείο που δημιουργείται να μην έχει την κατάληξη .cache, οπότε την προσθέτουμε εμείς).

Παρατηρούμε ότι το αρχείο phpunit.xml.dist περιέχει πληροφορίες σχετικά με το ποιους φακέλους να ψάξει για να βρει τα τεστ, καθώς και πληροφορίες για το πού να δημιουργήσει τα αποτελέσματα του (τμήμα logging).

Εκτελώντας την εντολή `phpunit -c app/`, ενώ βρισκόμαστε στην ρίζα του project μας παίρνουμε το αποτέλεσμα που φαίνεται στην εικόνα 6

```
dimitris@dimitris-virtual-machine:/var/www/SymfonyNetBeans$ phpunit -c app/
PHPUnit 4.2.0 by Sebastian Bergmann.

Configuration read from /var/www/SymfonyNetBeans/app/phpunit.xml.dist

..

Time: 5.15 seconds, Memory: 26.75Mb

OK (2 tests, 3 assertions)

Generating code coverage report in Clover XML format ... done

Generating Crap4J report XML file ... done

Generating code coverage report in HTML format ... done
```

Εικόνα 6 – Αποτέλεσμα PhpUnit

Μας ενημερώνει ότι τα τεστ που έχουμε περάσανε με επιτυχία. Ενώ δημιουργηθήκανε διάφορα αρχεία, τα οποία θα αναλάβει να τα ερμηνεύσει το εργαλείο Jenkins.

Στην εικόνα 7 παρουσιάζεται το περιεχόμενο του αρχείου phpmd.xml, που τοποθετούμε στο ίδιο σημείο με το προηγούμενο.

```
<?xml version="1.0"?>
<ruleset name="Symfony2 ruleset" xmlns="http://pmd.sf.net/ruleset/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://pmd.sf.net/ruleset/1.0.0 http://pmd.sf.net/ruleset_xml_schema.xsd"
         xsi:noNamespaceSchemaLocation="http://pmd.sf.net/ruleset_xml_schema.xsd">
    <description>
        Custom ruleset.
    </description>

    <rule ref="rulesets/design.xml" />
    <rule ref="rulesets/unusedcode.xml" />
    <rule ref="rulesets/codesize.xml" />
    <rule ref="rulesets/naming.xml" />

</ruleset>
```

Εικόνα 7 – phpmd.xml

Αυτό το αρχείο περιλαμβάνει οδηγίες σχετικά με το ποια coding standards πρέπει να τηρούνται στο project μας.

5.5 Δημιουργία αρχείου κτισίματος

Με την παραπάνω μικρή εισαγωγή, φαίνεται το πόσο χρονοβόρο και πολύπλοκο είναι να εκτελεί κανείς όλα αυτά τα εργαλεία ένα ένα.

Αυτό το πρόβλημα αναλαμβάνει να επιλύσει το εργαλείο ant και το αρχείο build.xml που θα δημιουργήσουμε.

Στο τέλος αυτής της ενότητας θα έχουμε ένα πλήρως αυτοματοποιημένο περιβάλλον εκτέλεσης των τεστ και παραγωγής των αποτελεσμάτων, με την χρήση μίας μόνο εντολής.

Πρώτα από όλα το αρχείο build.xml τοποθετείται στον κεντρικό κορμό του project μας όπως φαίνεται στην εικόνα 3.

Το αρχείο χωρίζεται σε διάφορα τμήματα (targets) βάση του ποιου πρόγραμμα θα εκτελεστεί, ενώ στην αρχή έχει εντολές για το πού θα δημιουργηθούνε οι φάκελοι επανατοποθέτησης των αποτελεσμάτων, καθώς και μεταβλητές που περιέχουνε τα διαφορετικά paths μέσα στο project μας.

Η εικόνα 8 δείχνει την αρχικοποίηση του αρχείου, της μεταβλητές μας καθώς και το ποια επιμέρους τμήματα θα εκτελεστούνε.

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

<project name="JenkinsProject" default="build">
  <property name="workspace" value="${basedir}" />
  <property name="sourcedir" value="${basedir}/src" />
  <property name="builddir" value="${workspace}/app/build" />

  <target name="build"
  depends="prepare,vendors,parameters,lint,phploc,pdepend,phpcpd,phpmd-ci,phpcs-ci,phpdox,phpunit,phpcb"/>

  <target name="build-parallel" depends="prepare,lint,tools-parallel,phpunit,phpcb"/>

  <target name="tools-parallel" description="Run tools in parallel">
    <parallel threadCount="2">
      <sequential>
        <antcall target="pdepend"/>
        <antcall target="phpmd-ci"/>
      </sequential>
      <antcall target="phpcpd"/>
      <antcall target="phpcs-ci"/>
      <antcall target="phploc"/>
      <antcall target="phpdox"/>
    </parallel>
  </target>

  <target name="clean" description="Cleanup build artifacts">
    <delete dir="${builddir}/api"/>
    <delete dir="${builddir}/code-browser"/>
    <delete dir="${builddir}/coverage"/>
    <delete dir="${builddir}/logs"/>
    <delete dir="${builddir}/pdepend"/>
    <delete dir="${builddir}/docs/*"/>
  </target>

  <target name="prepare" depends="clean" description="Prepare for build">
    <mkdir dir="${builddir}/api"/>
    <mkdir dir="${builddir}/code-browser"/>
    <mkdir dir="${builddir}/coverage"/>
    <mkdir dir="${builddir}/logs"/>
    <mkdir dir="${builddir}/pdepend"/>
  </target>
```

Εικόνα 8 – build.xml Αρχικοποιήση

Το αρχείο είναι άκρως κατανοητό στο τί κάνει μέχρι στιγμής. Καθαρίζει τα προηγούμενα αποτελέσματα στο target clean, δημιουργεί καινούριους φακέλους στο target prepare, στο build μας δίνει όλα τα targets που θα εκτελεστούνε ενώ το tools-parallel μας δίνει την δυνατότητα να εκτελούμε διαφορετικά εργαλεία ταυτόχρονα.

Για κάθε εργαλείο, πρέπει να δημιουργήσουμε το αντίστοιχο target. Ο πίνακας 2 δείχνει τα κομμάτια για κάθε εργαλείο.

Πίνακας 2 - Αντιστοίχηση Plugin - Target

PhpUnit	`<target name="phpunit" description="Run unit tests with PHPUnit">`
	`  <exec executable="phpunit" failonerror="true">`
	`  <arg value="-c" />`
	`  <arg path="${basedir}/app/phpunit.xml.dist" />`
	`  </exec>`
	`</target>`

phploc	`<target name="phploc" description="Measure project size using PHPLOC">`
	`  <exec executable="phploc">`
	`  <arg value="--log-csv" />`
	`  <arg value="${builddir}/logs/phploc.csv" />`
	`  <arg path="${sourcedir}" />`
	`  </exec>`
	`</target>`

Lint	`<target name="lint" description="Perform syntax check of sourcecode files">`
	`  <apply executable="php" failonerror="true">`
	`  <arg value="-l" />`
	`  <fileset dir="${sourcedir}">`
	`  <include name="**/*.php" />`
	`  <modified />`
	`  </fileset>`
	`  <fileset dir="${basedir}/src/">`
	`  <include name="**/*Test.php" />`
	`  <modified />`
	`  </fileset>`
	`  </apply>`
	`</target>`

Pdepend	`<target name="pdepend" description="Calculate software metrics using PHP_Depend">`
	`  <exec executable="pdepend">`
	`  <arg value="--jdepend-xml=${builddir}/logs/jdepend.xml" />`
	`  <arg value="--jdepend-chart=${builddir}/pdepend/dependencies.svg" />`
	`  <arg value="--overview-pyramid=${builddir}/pdepend/overview-pyramid.svg" />`
	`  <arg path="${sourcedir}" />`
	`  </exec>`
	`</target>`

phpmd

```
<target name="phpmd" description="Perform project mess detection using PHPMD and print human readable output.
        Intended for usage on the command line before committing.">
  <exec executable="phpmd">
    <arg path="${basedir}/src" />
    <arg value="text" />
    <arg value="${workspace}/app/phpmd.xml" />
  </exec>
</target>
```

Phpmd-ci

```
<target name="phpmd-ci" description="Perform project mess detection
        using PHPMD creating a log file for the continuous integration server">
  <exec executable="phpmd">
    <arg path="${sourcedir}" />
    <arg value="xml" />
    <arg value="${workspace}/app/phpmd.xml" />
    <arg value="--reportfile" />
    <arg value="${builddir}/logs/pmd.xml" />
  </exec>
</target>
```

Phpcs

```
<target name="phpcs" description="Find coding standard violations
  using PHP_CodeSniffer and print human readable output.
  Intended for usage on the command line before committing.">
  <exec executable="phpcs">
    <arg value="--standard=Symfony2" />
    <arg path="${sourcedir}" />
  </exec>
</target>
```

Phpcs-ci

```
<target name="phpcs-ci" description="Find coding standard violations using PHP_CodeSniffer
    creating a log file for the continuous integration server">
  <exec executable="phpcs" output="/dev/null">
    <arg value="--report=checkstyle" />
    <arg value="--report-file=${builddir}/logs/checkstyle.xml" />
    <arg value="--standard=Symfony2" />
    <arg path="${sourcedir}" />
  </exec>
</target>
```

Phpcpd

```
<target name="phpcpd" description="Find duplicate code using PHPCPD">
  <exec executable="phpcpd">
    <arg value="--log-pmd" />
    <arg value="${builddir}/logs/pmd-cpd.xml" />
    <arg path="${sourcedir}" />
  </exec>
</target>
```

Phpcb

```
<target name="phpcb" description="Aggregate tool output with PHP_CodeBrowser">
  <exec executable="phpcb">
    <arg value="--log" />
    <arg path="${builddir}/logs" />
    <arg value="--source" />
    <arg path="${sourcedir}" />
    <arg value="--output" />
    <arg path="${builddir}/code-browser" />
  </exec>
</target>
```

Επιπλέον, για να έχει το εργαλείο Jenkins το κατάλληλο configuration τοποθετούμε δύο targets σχετικά με το Symfony2 τα οποία παρουσιάζονται στον πίνακα 3.

Πίνακας 3 - Symfony2 - build.xml

vendors	`<target name="vendors" description="Update vendors">`
	`  <exec executable="composer" failonerror="true">`
	`  <arg value="update" />`
	`  <arg value="--no-scripts"/>`
	`  </exec>`
	`</target>`

Parameters	`  <target name="parameters" description="Copy parameters">`
	`  <exec executable="cp" failonerror="true">`
	`  <arg path="app/config/parameters.yml.dist" />`
	`  <arg path="app/config/parameters.yml" />`
	`  </exec>`
	`  </target>`
	
	`</project>`

Εκτελώντας πλέον την εντολή ant στον κορμό του project μας, θα πάρουμε μια μακρά λίστα, με τα αποτελέσματα των παραπάνω εργαλείων. Αυτά τα αποτελέσματα θα ερμηνεύσει το εργαλείο Jenkins.

5.6 Εγκατάσταση Jenkins

Αφού έχουμε εκτελέσει τα παραπάνω βήματα είμαστε σε θέση να εγκαταστήσουμε το Jenkins ακολουθώντας τις παρακάτω εντολές :

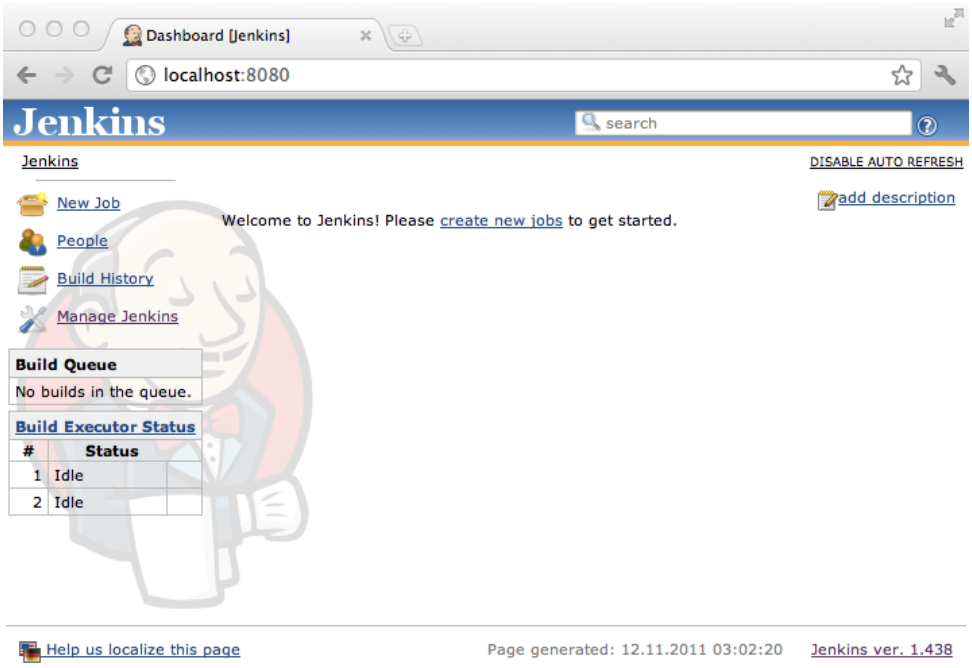
I. `wget -q -O http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -`
II. `sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/Jenkins.list'`
III. `sudo apt-get update`
IV. `sudo apt-get install Jenkins`

Με την παραπάνω διαδικασία έχουμε την τελευταία έκδοση του προγράμματος πάντα, αφού προσθέτουμε στο σύστημα μας την κετρνική αποθήκη κώδικα του

διαχειρίζεται τις εκδόσεις του Jenkins. Θα παίρνουμε και ειδοποιήσεις για τυχόν ενημερώσεις πλέον.

Κάθε φορά που θα ξεκινάμε τον υπολογιστή μας, η διεργασία Jenkins πρέπει να ξεκινάει αυτόματα και αυτή. Για να το πετύχουμε αυτό προσθέτουμε στο αρχείο /etc/rc.local την γραμμή `/etc/init.d/Jenkins start`.

Αν όλα πήγανε όπως πρέπει, ανοίγοντας τον browser μας στην διεύθυνση localhost:8080 θα δούμε την εικόνα 9

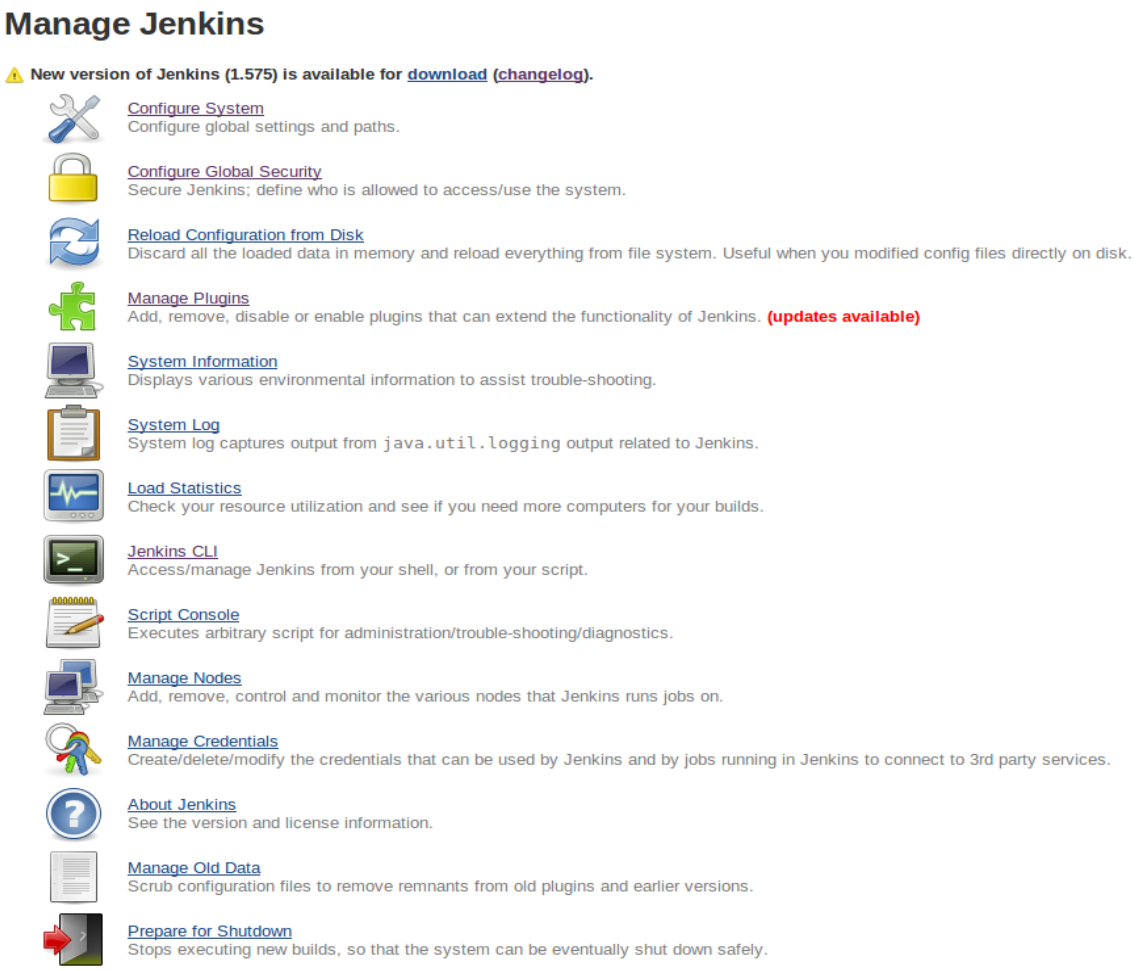


Εικόνα 9 – Αρχική Οθόνη Jenkins

Εδώ βλέπουμε την κεντρική οθόνη του εργαλείου. Με την επιλογή New Job προσθέτουμε καινούρια εργασία, στην επιλογή people βλέπουμε το ποια άτομα έχουνε πρόσβαση και τί είδους πρόσβαση, το build history μας δείχνει για κάθε εργασία το κάθε κτίσιμο και το αποτέλεσμα του ενώ τέλος η καρτέλα Manage Jenkins μας δίνει την δυνατότητα παραμετροποίησης του εργαλείου.

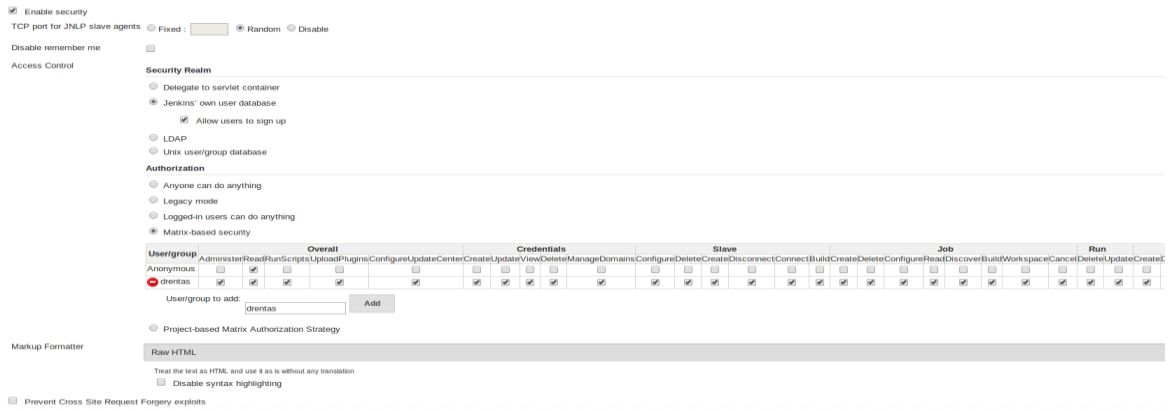
Για αρχή καλό είναι να περιορίσουμε την πρόσβαση για λόγους ασφαλείας. Μα δίνονται πάρα πολλές επιλογές για τον τρόπο διαχείρισης των χρηστών και για το ποιοί μπορούνε να έχουνε πρόσβαση και τί είδους πρόσβαση στο εργαλείο. Θα χρησιμοποιήσουμε, την βάση του Jenkins σε αυτό το παράδειγμα, ενώ μπορούμε να χρησιμοποιήσουμε Access Lists.

Πηγαίνοντας στην καρτέλα Manage Jenkins συναντάμε τις επιλογές της εικόνας 10



Εικόνα 10 – Επιλογές Jenkins

Επιλέγουμε την καρτέλα Configure Global Security και βάζουμε τις τιμές όπως φαίνονται στην εικόνα 11



Εικόνα 11 – Επιλογές Ασφάλειας

Με την αποθήκευση αυτών των αλλαγών, θα χρειαστεί να δημιουργήσουμε λογαριασμό με τον όνομα χρήστη που βάλαμε, και κάθε φορά να συνδεόμαστε με αυτόν.

Σαν εργαλείο έχει πάρα πολλές επιλογές το οποίο είναι λογικό να μην τις καλύψουμε όλες, παρά μόνο τα απαραίτητα.

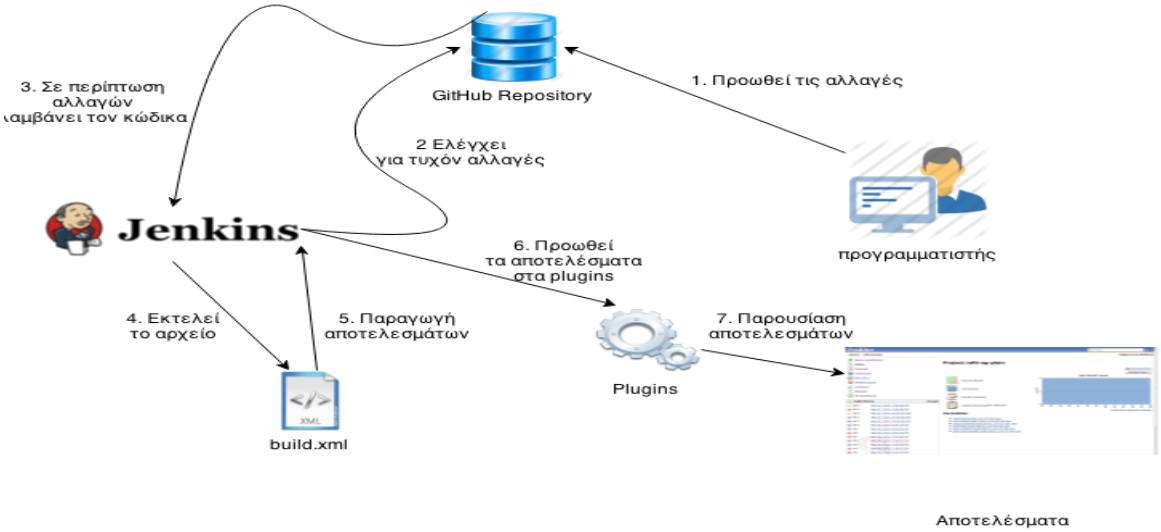
Το επόμενο βήμα είναι να εγκαταστήσουμε τα plugins τα οποία θα αναλάβουνε την επεξεργασία των αποτελεσμάτων των εργαλείων μετά την εκτέλεση του build.xml αρχείου.

Ενώ είμαστε στην καρτέλα Manage Jenkins, επιλέγουμε το Plugin Manager, μετά πηγαίνουμε στο Advanced και πατάμε το check now. Με αυτή την επιλογή ενημερώνεται το πρόγραμμα με τις τελευταίες εκδόσεις των plugins. Έπειτα, πηγαίνουνε στην καρτέλα Available και εγκαθιστούμε τα plugins που αναφέραμε στον πίνακα 1.

Πλέον έχουμε έτοιμο ρυθμισμένο το πρόγραμμα, το επόμενο βήμα είναι η δημιουργία εργασιών.

5.7 Δημιουργία Εργασιών.

Αυτό που θέλουμε να καταφέρουμε σε αυτή την ενότητα είναι η λήψη του κώδικα από την κεντρική μας αποθήκη, η εκτέλεση του αρχείου build.xml, η επεξεργασία και τέλος η προβολή των αποτελεσμάτων. Αυτό αποτυπώνεται στην εικόνα 12.



Σχήμα 5-1 Ροή εργαςίας Jenkins

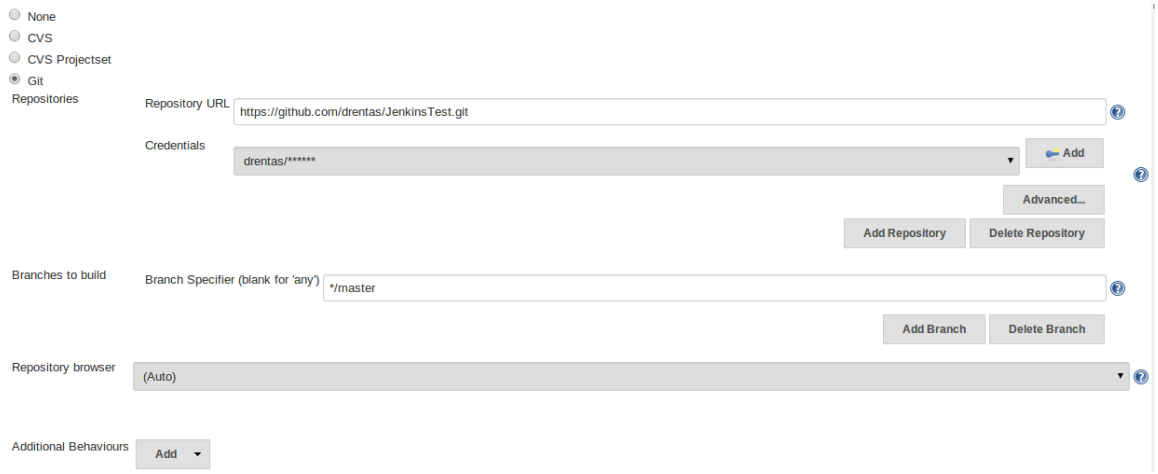
Πρώτα από όλα επιλέγουμε το μενού New Item από το κεντρικό menu του Jenkins.

Στην επόμενη καρτέλα βάζουμε το όνομα που θέλουμε να έχει αυτή η εργασία και επιλέγουμε Build free style software project.

Όπως είδαμε το εργαλείο χρειάζεται να γνωρίζει 4 πράγματα, τα οποία αναλύονται ως εξής :

1. Κεντρική αποθήκη

   Στην εργασία που δημιουργήσαμε επιλέγουμε το configure και στην επιλογή Source Code Management βάζουμε την κεντρική αποθήκη που δημιουργήσαμε όπως φαίνεται στην εικόνα 13

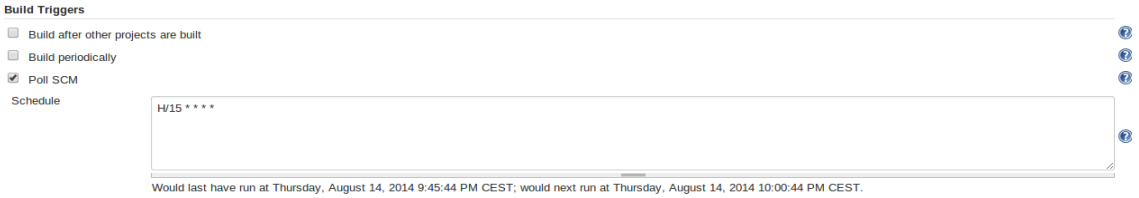


Εικόνα 12 – Εισαγωγή Κεντρικής Αποθήκης

   Μπορούμε να προσθέσουμε το username και password που είχαμε βάλει στην δημιουργία της κεντρικής αποθήκης.

2. Κάθε πότε να ελέγχει για αλλαγές στην κετρική αποθήκη το Jenkins

   Στην επιλογή Build Triggers επιλέγουμε Poll SCM και στο πεδίο από κάτω βάζουμε το κάθε πότε θέλουμε να ελέγχει την κεντρική αποθήκη, για παράδειγμα ανά 15 λεπτά όπως φαίνεται στην εικόνα 14.



Εικόνα 13 – Έλεγχος για Αλλαγές στην Αποθήκη

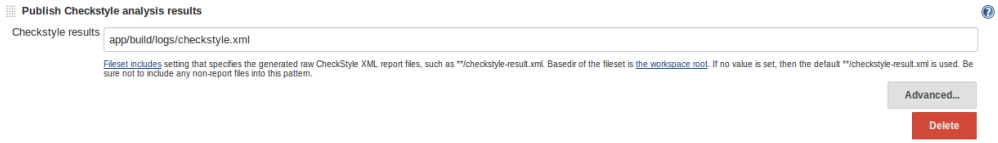
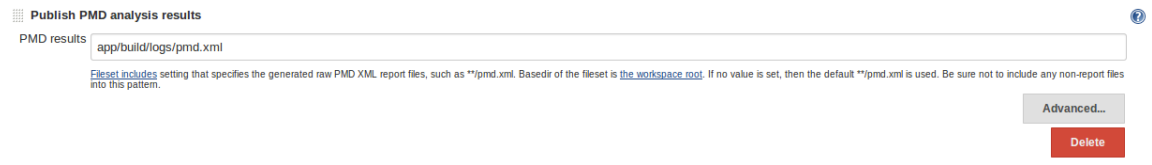
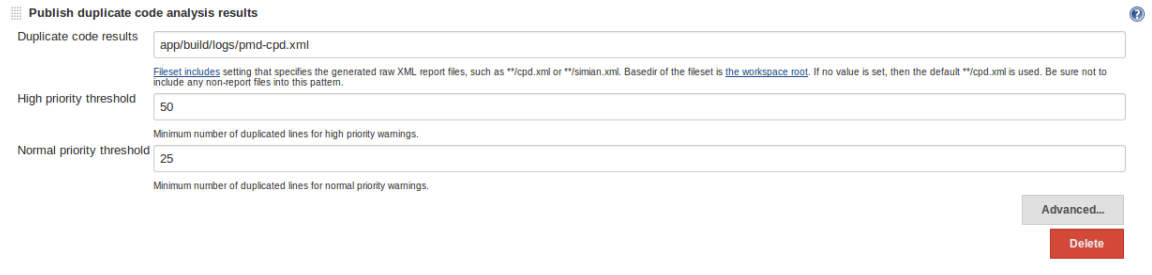
3. Ποιό εργαλείο αυτόματου κτισίματος χρησιμοποιείται

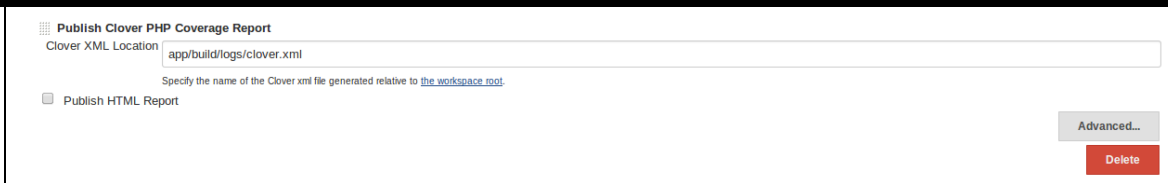
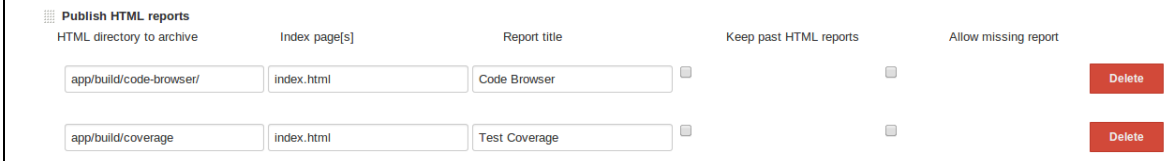
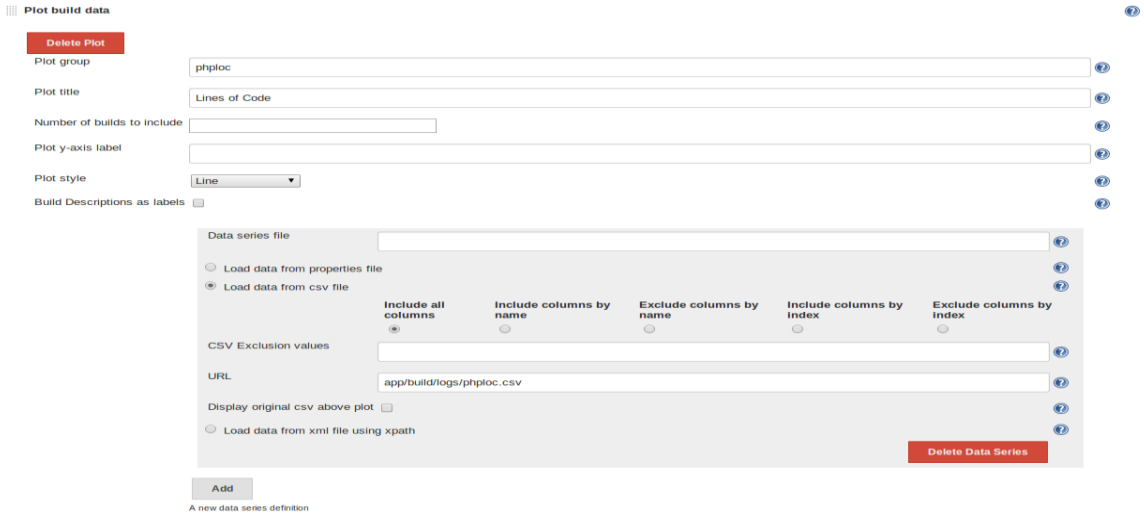
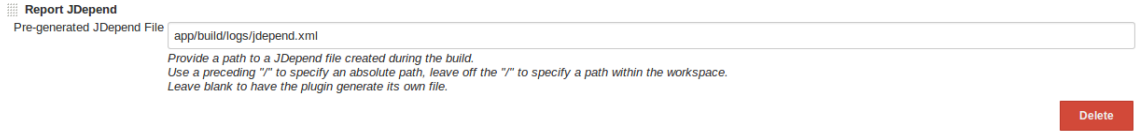
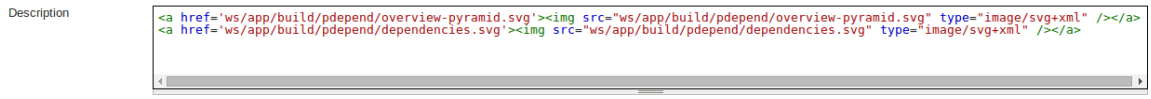
   Σε αυτή την επιλογή πρέπει να ενημερώσουμε το Jenkins ότι χρησιμοποιήσαμε ένα script ant για το αυτόματο κτίσιμο. Στην επιλογή Add build step επιλέγουμε invoke ant.

4. Το πώς θα επεξεργαστούμε τα αποτελέσματα.

   Επιλέγοντας το Add post-build action, προσθέτουμε τα plugins που έχουμε εγκαταστήσει. Ο πίνακας 4, παρουσιάζει αναλυτικά τα βήματα αυτά

Πίνακας 4 - Αντιστοιχία Plugin - Αποτελέσματα

CheckStyle	
PMD	
Duplicate	

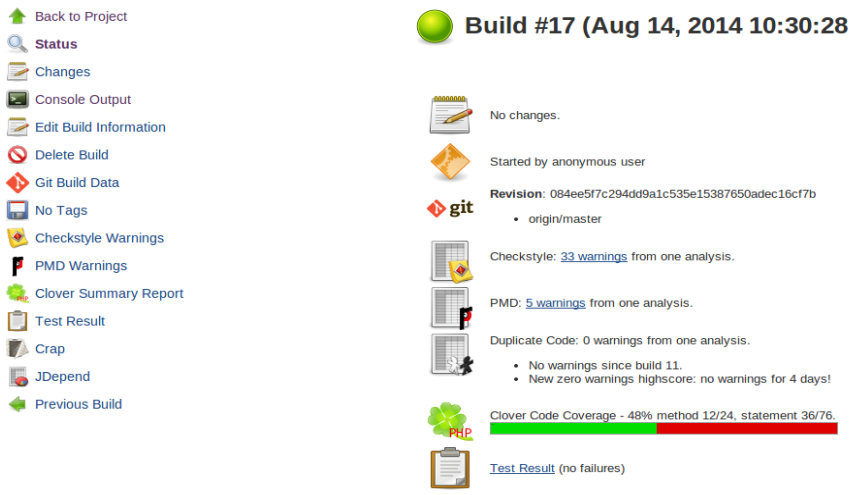
Clover	
Browser, Coverage	
LoC	
PHPUnit	
JDepend	
Pdepend	

Πίνακας 5

Το εργαλείο Pdepend παράγει 2 εικόνες. Δεν υπάρχει κάποιο plugin για την αναπαράσταση αυτών των εικόνων και για αυτό βάζουμε τον κώδικα HTML, στο description της εργασίας μας.

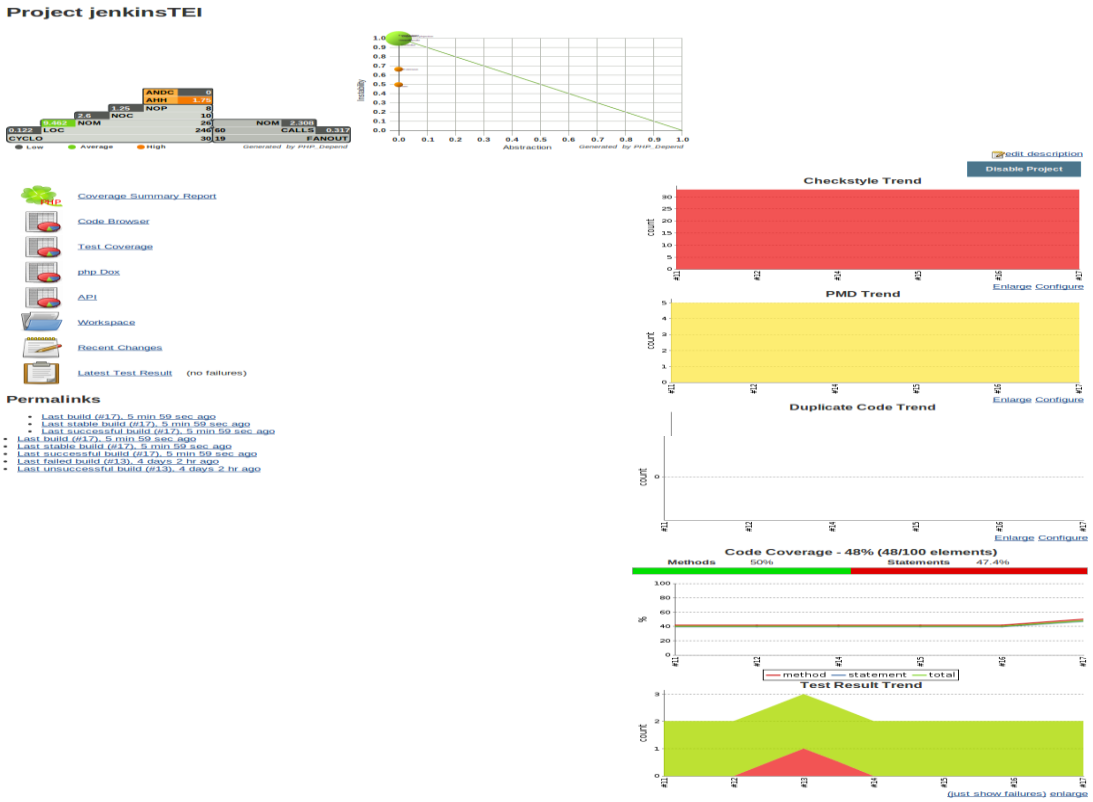
5.8 Εκτέλεση και εμφάνιση αποτελεσμάτων

Με τις παραπάνω ενέργειες έχουμε στήσει το εργαλείο και είναι έτοιμο για την εκτέλεση της εργασίας. Πατάμε το Build Now και η διαδικασία ξεκινάει. Επιλέγοντας την εργασία που εκτελείται και μετά console output βλέπουμε αναλυτικά τα βήματα που εκτελούνται. Με το τέλος της εκτέλεσης θα πάρουμε όλα τα αποτελέσματα. Η εικόνα της συγκεκριμένης εργασίας μας πλέον θα είναι όπως την εικόνα 15



Εικόνα 14 – Αποτελέσματα Εργασίας

Στην εικόνα 16 βλέπουμε όλα τα αποτελέσματα μαζεμένα, στην κεντρική οθόνη της των εργασιών μας. Την κάλυψη του κώδικα, τα αποτελέσματα των τεστ, διάφορες μετρικές σχετικά με την πολυπλοκότητα του project μας.



Εικόνα 16 – Αποτελέσματα Project

Κάθε ένα από τα παραπάνω αποτελέσματα, έχει και επιπλέον επιλογές, ενώ επιλέγοντας το όνομα του project μας δίνονται και άλλες επιλογές όπως Code Browsing, Κάλυψη κώδικα η οποία παρουσιάζεται στην εικόνα 17.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		47.37%	36 / 76		50.00%	12 / 24		25.00%	2 / 8
Command		0.00%	0 / 16		0.00%	0 / 2		0.00%	0 / 1
Controller		38.10%	8 / 21		33.33%	3 / 9		0.00%	0 / 3
DependencyInjection		100.00%	4 / 4		100.00%	2 / 2		100.00%	1 / 1
EventListener		100.00%	6 / 6		100.00%	2 / 2		100.00%	1 / 1
Form		0.00%	0 / 7		0.00%	0 / 2		0.00%	0 / 1
Twig		81.82%	18 / 22		71.43%	5 / 7		0.00%	0 / 1

Εικόνα 17 – Code Browser

Ο χρήστης, μπορεί να έχει μία καλη εικόνα για το project του αναλύοντας τις παραπάνω μετρικές.

Σε περίπτωση που τα τεστ μας δεν περάσουνε το εργαλείο μας δίνει αναφορά για το τί απέτυχε.

Επιπλέον μπορούμε να παραμετροποιήσουμε το πότε θα θεωρείτο μία εργασία ανεπιτυχής, καθορίζοντας όρια τιμών των αποτελεσμάτων (κάλυψη κώδικα μικρότερο του 60% για παράδειγμα).

5.9 Τυχόν προβλήματα

Τα περισσότερα από τα προβλήματα, προέρχονται από τα δικαιώματα πρόσβασης στους διάφορους φακέλους από το Jenkins. Αυτά λύνονται με τις παρακάτω εντολές :

i. `Sudo usermode –G www-data –a 'jenkins'`
ii. `Sudo chmod 777 –R app/cache`
iii. `Διαγραφή των περιεχομένων του app/cache φακέλου`
iv. `Στην αρχή των αρχείων web/app.php και b/app_dev.php, προσθέτουμε το umask(000)`
v. `Σε περίπτωση εμφάνισης Fatal Error Class Sensio not found, στο αρχείο compose.json μετακινούμε το "sensio/generator-bundle":"~2.3 από το σημείο "require-dev" στο σημείο "require". Προσοχή στο "," .`

5.10 Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκε το εργαλείο Jenkins, και έγινε ένας βήμα προς βήμα οδηγός δημιουργίας περιβάλλοντος συνεχής ενσωμάτωσης για ένα PHP project.

Παρατηρούμε, ότι το αρχικό στήσιμο και παραμετροποίηση του συστήματος μπορεί να είναι πολύπλοκη και χρονοβόρα, αφού απαιτεί αρκετά εργαλεία.

Παρά αυτό το αρνητικό, τα αποτελέσματα βελτιώνουνε την ποιότητα του project μας και αφού στηθεί το εργαλείο μετά δεν χρειάζεται ιδιαίτερη συντήρηση.

6

Βιβλιογραφία

[1]Astels, D. (2003). *Test-Driven Development: A Practical Guide.* USA: Prentice Hall.

[2]Bergman, S. (2011). *PHP Projects with Jenkins.* USA: O'Reilly.

[3]Copeland, L. (2004). *A Practitioner's Guide to Software Test Design.* Chicago: Artech House.

[4]Kent Beck, C. A. (2004). *Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series) .* USA: Addison-Wesley.

[5]Schwaber, K. (2001). *Agile Software Development with Scrum.* USA: Prentice Hall.

[6]Shari Lawrence Pfleeger, J. M. (2010). *Software Engineering Theory and Practice 4th Edition.* Seattle: Pearson.

[7]Shore, J. (2007). *The Art of Agile Development Pragmatic guide to agile software development.* USA: O'Reilly.

[8]Smart, J. F. (2011). *Jenkins The Definitive Guide 1st Edition.* USA: O'Reilly.

[9]Sommerville, I. (2011). *Software Engineering 9th Edition.* Massachusetts: Addison-Wesley.