Πανεπιστήμιο Πειραιώς
Τμήμα Ψηφιακών Συστημάτων

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ

«Διδακτικής της Τεχνολογίας & Ψηφιακών Συστημάτων»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ :

«Πλαίσιο Ελέγχου Λογισμικού»

Γεωργία Μπαρνασά –ΑΜ : ΜΕ09065

Επιβλέπουσα : Επικ, Καθ. Α.Πρέντζα

Πειραιάς 2012

Πανεπιστήμιο Πειραιώς
Τμήμα Ψηφιακών Συστημάτων

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ

«Ψηφιακών Συστημάτων»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ :

«Πλαίσιο Ελέγχου Λογισμικού»

Εγκρίθηκε από την εξεταστική επιτροπή την Ιουνίου 2012.

Αθήνα, Ιούνιος 2012

....

Αφιερώνεται στην Οικογένειά μου

για την αγάπη τους και την υπομονή

τους όλα αυτά τα χρόνια και στους φίλους μου

για τη συμπαράστασή τους…

Ευχαριστώ ιδιαιτέρως την καθηγήτρια μου Κ. Ανδριάνα Πρέντζα η οποία με εμπιστεύτηκε αναθέτοντας μου την παρούσα πτυχιακή εργασία. Καθώς επίσης και για την πολύτιμη βοήθεια και καθοδήγησή της, καθ' όλη την διάρκεια εκπόνησης της πτυχιακής μου εργασίας , η οποία με τις παρατηρήσεις και συμβουλές της βοήθησε στη βελτίωση της υλοποίησης.

Ευχαριστώ θερμά την εταιρία στην οποία εργάζομαι (Unisystems) η οποία μου επέτρεψε την χρήση του εργαλείου ελέγχου, τον έλεγχο του συστήματος καθώς και την πρόσβαση σε δεδομένα της εταιρίας.

Γεωργία Π. Μπαρνασά

Software Testing

Γεωργία Μπαρνασά

Έλεγχος Λογισμικού

ΠΕΡΙΛΗΨΗ

Ο έλεγχος των προγραμμάτων στις μέρες μας είναι μια αναπόφευκτη διαδικασία και ένα σημαντικό, απαραίτητο στοιχείο της ανάπτυξης λογισμικού. Αρκετοί επιστήμονες ασχολήθηκαν με τεχνικές ελέγχου οι οποίες θα προσφέρουν αυτοματοποιημένα ποιοτικά και αξιόπιστα αποτελέσματα σε σύντομο χρονικά διάστημα, χωρίς την επιβάρυνση του ανθρώπου[1]. Στην παρακάτω εργασία παρουσιάζονται και προτείνονται κάποια αυτοματοποιημένα εργαλεία ελέγχου λογισμικού με σκοπό την βελτίωση των τεχνικών ελέγχου και της διαδικασίας ελέγχου λογισμικού γενικότερα. Έχοντας αποδεχθεί το γεγονός αυτό και εφόσον ο έλεγχος αποτελεί ούτως ή άλλως μια καθοριστική φάση της ανάπτυξης του λογισμικού, η παρούσα εργασία, επιχειρεί να εστιαστεί στη φάση του ελέγχου και να διερευνήσει τη συσχέτισή της με τα κατάλληλα εργαλεία ελέγχου λογισμικού.

Η διαδικασία ελέγχου λογισμικού δεν θεωρείται καθόλου εύκολη λειτουργία. Αντιθέτως είναι μια επίπονη και δαπανηρή εργασία μιας και καταλαμβάνει το 25% - 50% του συνολικού κόστους στη διαδικασία παραγωγής ενός λογισμικού. Ακόμη ο χρόνος που χρειάζεται για τη διεκπεραίωση της μπορεί να είναι και μεγαλύτερος από αυτόν του χρόνου κατασκευής του λογισμικού. Είναι εμφανές λοιπόν το πόσο επιτακτική είναι η ανάγκη εύρεσης ενός αποτελεσματικού αυτοματοποιημένου συστήματος ελέγχου. Ο αναγνώστης εδώ θα πρέπει να σημειώσει ότι μπορεί να μην είναι δυνατόν να βρεθεί μια αυτοματοποιημένη λύση από έναν και μόνο οργανισμό ο οποίος θα καλύψει πλήρως και επαρκώς τις ανάγκες ελέγχου της επιχείρησης αλλά θα δει πως μπορούν να υπάρξουν και συνδυασμοί τεχνικών ώστε να δοθεί η βέλτιστη και αποτελεσματικότερη λύση στις ανάγκες ελέγχου ενός οργανισμού. Στην μελέτη περίπτωσης ελέγχου ηλεκτρονικού συστήματος μεταφοράς χρημάτων που παρουσιάζεται επιλέγεται το καταλληλότερο εργαλείο ελέγχου, για το σύστημα WinRunner, το οποίο δίνει την δυνατότητα για ένα εύκολο, γρήγορο και αποδοτικό τρόπο ελέγχου της απόδοσης και αντοχής του συστήματος κάτω από συνθήκες πίεσης του συστήματος. Τέλος δίνεται μια μελλοντική πρόβλεψη για τον αυτοματοποιημένο έλεγχο μιας και η δημοτικότητα της αυτοματοποίησης ελέγχου έχει αυξηθεί τα τελευταία έτη επειδή οι εταιρίες λογισμικού δεν έχουν το χρόνο ή τα χρήματα για να επενδύσουν σε μεγάλες ομάδες ελέγχου ώστε να εξασφαλίσουν ότι οι εφαρμογές λειτουργούν σύμφωνα με τις καθορισμένες προδιαγραφές.

Περιεχόμενα

Περιεχόμενα Εικόνων

Περιεχόμενα Διαγραμμάτων

Περιεχόμενα Πινάκων

Περιεχόμενα Σχημάτων

1. Έλεγχος Λογισμικού

1.1. Εισαγωγή

1.1.1. Ορισμός προβλήματος

Είναι γνωστό πως στις μέρες μας η ευρεία χρήση γραφικού περιβάλλοντος στα σύγχρονα λογισμικά συστήματα έχει προκαλέσει την δημιουργία ακόμη πιο πολύπλοκων γραφικών περιβαλλόντων. Λόγω της αύξησης της πολυπλοκότητας αυτής δημιουργήθηκαν προβλήματα στον έλεγχο της ορθότητας αυτών των λογισμικών συστημάτων, αφού όσο διευκολύνουν την αλληλεπίδραση του χρήστη, τόσο δυσκολεύουν στην διαδικασία παραγωγής τους τον προγραμματιστή. Έτσι, δεδομένου της σημαντικής θέσης που έχει το γραφικό περιβάλλον στα σύγχρονα λογισμικά συστήματα, ο **έλεγχος** της **ορθότητας** του επηρεάζει την **ευρωστία**, την **ασφάλεια** και τη **δυνατότητα** χρησιμοποίησης του. *Πιο συγκεκριμένα* ***ο έλεγχος λογισμικού είναι μια τεχνική για την επαλήθευση της ποιότητας των προϊόντων και επίσης για την έμμεση βελτίωση τους, με τον προσδιορισμό των ατελειών και των προβλημάτων του λογισμικού.***

Ο έλεγχος αυτών των συστημάτων γραφικού περιβάλλοντος είναι η διαδικασία με την οποία ένα πρόγραμμα υπόκειται σε έλεγχο κατά πόσο τηρεί τις καταγραμμένες προδιαγραφές που τέθηκαν κατά το δεύτερο στάδιο κύκλου ζωής λογισμικού, όπως αυτό ορίζεται από την Τεχνολογία Λογισμικού. *Συνεπώς ο έλεγχος λογισμικού αποτελεί αναπόσπαστο κομμάτι τις διαδικασίας παραγωγής λογισμικού.* Για να γίνει ο έλεγχος αυτός, απαιτείται τεράστια προσπάθεια από την πλευρά του προγραμματιστή σε χρόνο, κόπο και χρήμα για να δημιουργήσει τέτοια σενάρια ελέγχου ώστε να είναι σίγουρος ότι το λογισμικό δουλεύει σωστά.

Είναι κατανοητό λοιπόν πως ο έλεγχος της ορθότητας ενός συστήματος με γραφικό περιβάλλον είναι δύσκολος για διάφορους λόγους: Καταρχήν ο χώρος των δυνατών αλληλεπιδράσεων σε ένα γραφικό περιβάλλον είναι τεράστιος. Κάθε διαφορετική

ακολουθία αλληλεπίδρασης με το πρόγραμμα θα προκαλέσει το σύστημα λογισμικού να μεταβεί σε διαφορετική κατάσταση κάτι θα οδηγήσει να δημιουργηθούν διαφορετικά αποτέλεσμα.

Με βάση αυτό, θα πρέπει ο ελεγκτής να δημιουργήσει τα απαραίτητα σενάρια ελέγχου, ώστε να καλύψει όλες τις δυνατές περιπτώσεις. Τα σενάρια αυτά μέχρι τώρα δημιουργούνταν στο χέρι και στη συνέχεια ο ελεγκτής θα έπρεπε να τα εκτελέσει και πάλι χειροκίνητα και να καταγράψει τα αποτελέσματα. Αυτό όμως δημιουργεί το εξής πρόβλημα. Ο ελεγκτής ή ο οποιοσδήποτε άλλος δημιουργός αυτών των σεναρίων, λόγω της εμπειρίας και της γνώσης που κατέχει, είναι δυνατόν να αγνοήσει περιπτώσεις - σενάρια τα οποία γνωρίζει πως δεν έχουν κάποια λογική συνοχή και που ο εξειδικευμένος χρήστης του υπό έλεγχο λογισμικού συστήματος δεν θα χρησιμοποιήσει. Αυτά τα σενάρια όμως ενδέχεται να περιέχουν λάθη και κάποιος ανειδίκευτος χρήστης πιθανόν να τα εκτελέσει. Σε αυτή την περίπτωση, το σύστημα λογισμικού θα παράγει λάθος αποτέλεσμα κατά την εκτέλεση του.

Συμπεραίνουμε λοιπόν πως για το λόγο αυτό, επιβάλλεται η ύπαρξη κάποιου συστήματος αυτόματου ελέγχου λογισμικού συστήματος με γραφικό περιβάλλον στο οποίο θα πρέπει να παράγονται εντελώς τυχαία σενάρια ελέγχου χωρίς να υπάρχει εκ των προτέρων καμία γνώση για το είδος και την λειτουργία του υπό έλεγχο λογισμικού συστήματος.

1.1.2.Σκοπός της διπλωματικής εργασίας

Πρωταρχικός στόχος της συγκεκριμένης εργασίας είναι η μελέτη και η κατανόηση των βασικών αρχών του ελέγχου λογισμικού και κατ' επέκταση των εργαλείων αυτών που εκτελούν τον έλεγχο λογισμικού. Στην συνέχεια η έρευνα επεκτείνεται στην ανάλυση του αναφερόμενου θέματος και προτείνεται μια νέα προσέγγιση στον αυτόματο έλεγχο λογισμικού συστήματος με γραφικό περιβάλλον με αυτόματη παραγωγή και εκτέλεση τυχαίων σεναρίων ελέγχου. Πιο συγκεκριμένα παρουσιάζεται η μ*ελέτη περίπτωσης ελέγχου ηλεκτρονικού συστήματος μεταφοράς χρημάτων όπου παρουσιάζεται το καταλληλότερο εργαλείο ελέγχου για το*

συγκεκριμένο σύστημα το WinRunner το οποίο χρησιμοποιείται για τον έλεγχο της διεπαφής χρήστη δίνοντας την δυνατότητα για ένα εύκολο, γρήγορο και αποδοτικό τρόπο ελέγχου της αναμενόμενης συμπεριφοράς των επιλεγμένων ενοτήτων τους συστήματος. Το WinRunner το οποίο αυτοματοποιεί ολόκληρη την διαδικασία του έλεγχου λογισμικών συστημάτων δέχεται ως είσοδο το υπό έλεγχο λογισμικό σύστημα, εξαγάγει όλα τα στοιχεία του γραφικού του περιβάλλοντος και τα παρουσιάζει. Στην συνέχεια, δέχεται ως δεύτερη είσοδο τις προδιαγραφές και με βάση αυτές, παράγει γενικά ή ειδικά, ανάλογα με το πόσο συγκεκριμένα θέλει να κάνει έλεγχο ο χρήστης, τα σενάρια ελέγχου. Τέλος εκτελεί τα σενάρια ελέγχου αυτόματα, και παρουσιάζει τα αποτελέσματα της εκτέλεσης στον χρήστη.

1.1.3.Δομή Εργασίας

Κεφάλαιο 1 : Στο κεφάλαιο αυτό αναφέρονται οι στόχοι της διπλωματική εργασίας και ποια είναι η συνεισφορά της διπλωματικής στον τομέα ελέγχου λογισμικού. Επίσης πιο κάτω γίνεται ανάλυση εννοιών και όρων του υπό μελέτη θέματος.

Κεφάλαιο 2 : Στο κεφάλαιο αυτό παρουσιάζονται οι μελλοντικοί στόχοι στον τομέα ελέγχου λογισμικού, τα όνειρα, οι προκλήσεις που αντιμετωπίζουμε σήμερα στην έρευνα ελέγχου λογισμικού. Τέλος αναλύεται και η μεθοδολογία ελέγχου που χρησιμοποιήθηκε για να πραγματοποιηθεί αυτή η έρευνα.

Κεφάλαιο 3 : Στο κεφάλαιο αυτό παρουσιάζονται συνοπτικά οι επικρατέστερες τεχνικές ελέγχου λογισμικού.

Κεφάλαιο 4 : Στο κεφάλαιο αυτό παρουσιάζονται τα επίπεδα ελέγχου λογισμικού. Αν ο έλεγχος λογισμικού αναφέρεται στην ολοκλήρωση του συστήματος στον έλεγχο μονάδας ή στον έλεγχο εγκατάστασης κτλ.

Κεφάλαιο 5 : Στο κεφάλαιο αυτό αναφέρονται τα σύγχρονα εργαλεία ελέγχου και παρουσιάζεται ο αυτοματοποιημένος έλεγχος, τα οφέλη του και τα μειονεκτήματα του.

Κεφάλαιο 6 : Στο κεφάλαιο αυτό γίνεται σύγκριση των επικρατέστερων εργαλείων ελέγχου και επιλέγεται το καλύτερο για τον έλεγχο της μελέτης περίπτωσης που έχει επιλεχτεί να εφαρμοστεί ο έλεγχος λογισμικού.

Κεφάλαιο 7 : Στο κεφάλαιο αυτό παρουσιάζεται η μελέτη περίπτωσης του ηλεκτρονικού συστήματος μεταφοράς χρημάτων (EFT). Πιο συγκεκριμένα αναλύεται τι είναι ένα κεντρικό τραπεζικό σύστημα και τι συστήματα περιέχει ενώ αναφέρεται και ο λόγος που επιλέχθηκε για να εξεταστεί.

Κεφάλαιο8 : Στο κεφάλαιο αυτό παρουσιάζεται μια επισκόπηση της έρευνας ελέγχου λογισμικού και προτείνονται κάποιες μελλοντικές εργασίες.

Κεφάλαιο 9 : Στο Κεφάλαιο αυτό αναφέρεται η βιβλιογραφία της εργασίας.

2. Μελλοντικοί στόχοι και προκλήσεις στην έρευνα ελέγχου λογισμικού

2.1. Τι είναι ο έλεγχος λογισμικού;

Οι επενδύσεις στην τεχνολογία πληροφοριών (ΤΠ) συνεχίζουν να αυξάνονται με έναν επιταχύνοντας ρυθμό. Αυτό είναι κατά ένα μέρος συνδεδεμένο από τη συνεχώς αυξανόμενη εξάρτηση από την Τεχνολογία της πληροφορίας. Συνεπώς, το ζήτημα του ελέγχου ενός πληροφοριακού συστήματος γίνεται ολοένα και σημαντικότερο.

Πιο συγκεκριμένα η διαδικασία ελέγχου λογισμικού είναι μια επίπονη και δαπανηρή εργασία μιας και καταλαμβάνει το 25% - 50% του συνολικού κόστους στη διαδικασία παραγωγής ενός λογισμικού. *Ακόμη ο χρόνος που χρειάζεται για τη διεκπεραίωση της μπορεί να είναι και μεγαλύτερος από αυτόν του χρόνου κατασκευής του λογισμικού.* Πριν από την παράδοση οποιουδήποτε προϊόντος ή εφαρμογής, είναι απαραίτητο να διεξάγεται απαραίτητα έλεγχος του λογισμικού, όπου αφενός μεν να ελέγχεται η ορθότητα των προϊόντων κάθε φάσης του κύκλου ζωής ανάπτυξής του σύμφωνα με την διαδικασία της επαλήθευσης (verification) και αφετέρου να εκτιμάται το κατά πόσο το λογισμικό ικανοποιεί τις απαιτήσεις (requirements) που έχουν τεθεί, δηλαδή να πραγματοποιείται η λεγόμενη διαδικασία της επικύρωσης (validation). Βέβαια πρέπει να επισημανθεί πως ο έλεγχος αποσκοπεί στην εύρεση παρουσίας λαθών και όχι στην υπόδειξη έλλειψης λαθών. Πιο συγκεκριμένα, ο έλεγχος μπορεί να αποδείξει την ύπαρξη αδυναμιών του λογισμικού με την εύρεση κάποιων σφαλμάτων, αλλά δεν μπορεί σε καμία περίπτωση να αποδείξει την τελειότητα του λογισμικού, εάν δεν ανεβρεθούν σφάλματα. Σε μια τέτοια περίπτωση το πιο πιθανό είναι να μην πραγματοποιήθηκε κατάλληλος και επαρκής έλεγχος.

Λόγω της δυσκολίας αυτής της διεργασίας, αρκετοί επιστήμονες ασχολήθηκαν με τεχνικές ελέγχου οι οποίες θα προσφέρουν αυτοματοποιημένα ποιοτικά και αξιόπιστα αποτελέσματα σε σύντομο χρονικά διάστημα, χωρίς την επιβάρυνση του ανθρώπου[2]. Όταν γίνεται αναφορά σε ποιοτικά και αξιόπιστα αποτελέσματα εννοείται η εύρεση πιθανών λαθών στον πηγαίο κώδικα. Εάν αυτό γίνει εφικτό τότε αυτή η διαδικασία θα πάψει να είναι τόσο δύσκολη και επίπονη, καθώς επίσης θα μειωθεί και το κόστος ανάπτυξης λογισμικού. Η σωστή τοποθέτηση ως προς την ποιότητα είναι η πρόληψη. Αφού, είναι σαφέστατα πολύ καλύτερο να αποφεύγονται τα προβλήματα, παρά να διορθώνονται. Είναι ίσως προφανές, αλλά

άξιο προσοχής, ότι ακόμα και μετά από έναν επιτυχώς ολοκληρωμένο έλεγχο σε όλο το σύστημα, το λογισμικό θα μπορούσε ακόμα να περιέχει λάθη.

Αλλά ακόμα και η αναγνώριση σφαλμάτων, τις περισσότερες φορές, οδηγεί σε αλλαγές του κώδικα, πράγμα που μπορεί να καταλήξει σε βελτίωση της κατάστασης, αλλά μπορεί και να εισάγει επιπρόσθετα προβλήματα, ιδιαίτερα στην περίπτωση που πρόκειται για αλλαγές σε έναν κώδικα μεγάλου μεγέθους και υψηλής πολυπλοκότητας. Αν λοιπόν θεωρηθεί ότι ο απώτερος σκοπός για κάθε προϊόν, είναι να πραγματοποιείται ο πλέον αποδοτικός ως προς το κόστος έλεγχος, που να διαβεβαιώνει ότι είναι αρκετά αξιόπιστο, αρκετά ασφαλές και ικανοποιεί τις απαιτήσεις του χρήστη/πελάτη, διαπιστώνεται ότι κάτι τέτοιο είναι εξαιρετικά δύσκολο, αν όχι ακατόρθωτο να επιτευχθεί, από τη στιγμή που δεν υπάρχει ποτέ αρκετός χρόνος για να ελεγχθούν τα πάντα ολοκληρωτικά. Γίνεται λοιπόν φανερό, ότι προκειμένου να πραγματοποιηθεί σωστός έλεγχος, απαιτείται κατανόηση των κινδύνων που σχετίζονται με την ύπαρξη σφαλμάτων στο λογισμικό, κίνδυνοι για τον χρήστη ή τον πελάτη, τον υπεύθυνο ανάπτυξης ή τον προμηθευτή, ή ακόμα και τους συντηρητές.
Με την πάροδο των χρόνων, η άποψη του Ελέγχου λογισμικού έχει εξελιχθεί προς μια εποικοδομητικότερη τοποθέτηση. Ο έλεγχος δεν φαίνεται πλέον ως δραστηριότητα που αρχίζει μόνο όταν είναι πλήρης η φάση κωδικοποίησης, με μοναδικό σκοπό την εύρεση αποτυχιών του συστήματος. Ο έλεγχος λογισμικού φαίνεται σήμερα ως δραστηριότητα που πρέπει να καλύψει ολόκληρη την αναπτυξιακή διαδικασία, και είναι το ίδιο σημαντικό σαν μέρος της πραγματικής κατασκευής του προϊόντος. Ο προγραμματισμός για τον έλεγχο πρέπει να αρχίσει από τα πρώτα στάδια ανάλυσης της απαίτησης, και τα σενάρια ελέγχου και οι διαδικασίες πρέπει να είναι συστηματικές και συνεχώς καθορισμένες καθώς η ανάπτυξη προχωρά. Αυτές οι δραστηριότητες του προγραμματισμού και του σχεδιασμού των ελέγχων αποτελούν μια χρήσιμη εισαγωγή για τους σχεδιαστές ώστε να δώσουν έμφαση στις πιθανές αδυναμίες (όπως, π.χ., παραλείψεις ή αντιφάσεις σχεδίου, και παραλείψεις ή ασάφειες στην τεκμηρίωση).

Ο έλεγχος λογισμικού είναι ένας ευρύς όρος που καλύπτει ένα μεγάλο φάσμα διαφορετικών δραστηριοτήτων, από τον έλεγχο που γίνεται σε ένα μικρό μέρους του κώδικα από τον προγραμματιστή (Unit Testing), στην επικύρωση πελατών ενός μεγάλου συστήματος πληροφοριών (acceptance testing) και στην παρακολούθηση του χρόνου εκτέλεσης μιας δίκτυο-κεντρικής εφαρμογής προσανατολισμένης στις υπηρεσίες. Στα διάφορα στάδια, οι περιπτώσεις ελέγχου θα μπορούσαν να κατηγοριοποιηθούν σε διαφορετικούς στόχους, όπως η έκθεση των αποκλίσεων από τις απαιτήσεις του χρήστη, ή η αξιολόγηση της προσαρμογής σε μια τυποποιημένη προδιαγραφή, ή η αξιολόγηση της ευρωστίας στους απαιτητικούς όρους φορτίων ή στις κακόβουλες εισαγωγές, ή η μέτρηση των δεδομένων

χαρακτηριστικών, όπως η απόδοση ή η δυνατότητα χρησιμοποίησης, ή ο υπολογισμός της λειτουργικής αξιοπιστίας, και ούτω καθεξής. Εκτός αυτού, ο έλεγχος θα μπορούσε να συνεχιστεί σύμφωνα με μια επίσημη ελεγχόμενη διαδικασία, που απαιτεί τον αυστηρό προγραμματισμό και την τεκμηρίωση, ή καλύτερα ανεπίσημα και κατά περίπτωση (ad hoc) διερευνητικό έλεγχο (exploratory testing).

Τέλος το μερίδιο της επένδυσης που λαμβάνει ένα μεγάλο ποσοστό των οργανωτικών εξόδων ενώ η αιτιολόγηση τέτοιων επενδύσεων αφού έχει γίνει ένα σημαντικό θέμα για τους ιθύνοντες. Η δικαιολόγηση των δαπανών στην ΤΠ είναι ένα από μακρο υφιστάμενο πρόβλημα, και οι διευθυντές για τις προηγούμενες λίγες δεκαετίες έχουν εκφράσει τις ανησυχίες τους για την αξία που παίρνουν από τις επενδύσεις της ΤΠ.

Βασικές αρχές Ελέγχου Λογισμικού

- ✓ Κάθε έλεγχος πρέπει να αφορά μια συγκεκριμένη απαίτηση
- ✓ Η προετοιμασία των ελέγχων αρχίζει πολύ πριν την εκτέλεσή τους
- ✓ Αρχή του Pareto: 80% των λαθών θα αφορούν 20% των τμημάτων του λογισμικού
- ✓ Ο έλεγχος πρέπει να αρχίζει από τις μικρές δομικές μονάδες και να καταλήγει στο ολοκληρωμένο σύστημα
- ✓ Δεν είναι δυνατό να ελεγχθεί ένα σύστημα πλήρως
- ✓ Ο αποτελεσματικός έλεγχος εκτελείται από τρίτους

Οι έλεγχοι λογισμικού περιλαμβάνουν τα εξής:

- Όσο καλύτερα δουλεύει η εφαρμογή τόσο πιο αποτελεσματικά μπορεί να ελεγχθεί.
- Όσο καλύτερα μπορεί το λογισμικό να ελεγχθεί τόσο καλύτερα μπορεί ο έλεγχος να αυτοματοποιηθεί και να βελτιστοποιηθεί.
- Ένας επιτυχής έλεγχος είναι αυτός που ανακαλύπτει ένα λάθος που ήταν πολύ δύσκολο να εντοπιστεί. Ο έλεγχος είναι μια διαδικασία για να προσδιοριστεί η ακρίβεια και η πληρότητα του λογισμικού.

Έλεγχος Λογισμικού

- Ο γενικός στόχος του ελέγχου λογισμικού είναι να βεβαιωθεί η ποιότητα του συστήματος λογισμικού ασκώντας έλεγχο συστηματικά στο λογισμικό μέσα σε προσεκτικά ελεγχόμενες καταστάσεις.

Ταξινομώντας με βάση το σκοπό ο έλεγχος λογισμικού μπορεί να διαιρεθεί σε

1. Έλεγχο ακρίβειας
2. Έλεγχο απόδοσης
3. Έλεγχο αξιοπιστίας
4. Έλεγχο ασφάλειας.

Αυτοί οι έλεγχοι συνήθως πραγματοποιούνται από τρίτα άτομα για να διαπιστώσουν οι χρήστες και οι ιδιοκτήτες της εταιρείας για την οποία παράγεται το λογισμικό σύστημα αν το προϊόν που παράγγειλαν είναι αυτό που θα τους παραδοθεί.

Συνεπεία αυτής της ποικιλίας των στόχων και του σκοπού, είναι η δημιουργία μιας πολλαπλότητας εννοιών για τον όρο «έλεγχος λογισμικού», το οποίο έχει παράγει πολλές και ιδιαίτερες ερευνητικές προκλήσεις. Στο υπόλοιπο κείμενο γίνεται μια προσπάθεια για ταξινόμηση των πολλών εννοιών του ελέγχου λογισμικού.

Ο έλεγχος αποτελείται πάντα από την παρατήρηση ενός δείγματος εκτελέσεων, και την παραγωγή μιας τελικής απόφασης για αυτές. Αρχίζοντας από αυτήν την πολύ γενική άποψη, μπορούν έπειτα να συγκεκριμενοποιηθούν οι διαφορετικές περιπτώσεις, με τη διάκριση συγκεκριμένων πτυχών που μπορούν να χαρακτηρίσουν το δείγμα των παρατηρήσεων:

ΓΙΑΤΙ: το γιατί γίνεται σε αυτό το αντικείμενο έλεγχος; Αυτή η ερώτηση αφορά το αντικείμενο ελέγχου, π.χ.: γίνεται έλεγχος για ελαττώματα; ή, πρέπει να αποφασίσουν οι χρήστες για το εάν το προϊόν μπορεί να προωθηθεί; ή μήπως πρέπει να αξιολογηθεί η δυνατότητα χρησιμοποίησης της πλατφόρμας του χρήστη (User Interface);

ΠΩΣ: ποιο δείγμα παρατηρείται, και πώς επιλέχθηκε; Αυτό είναι το πρόβλημα της επιλογής ελέγχου, το οποίο μπορεί να γίνει adhoc, τυχαία, ή με συστηματικό τρόπο με την εφαρμογή κάποιας αλγοριθμικής ή στατιστικής τεχνικής. Έχει εμπνεύσει πολύ έρευνα, σχετικά με το πώς επιλέγονται οι υποθέσεις - το κριτήριο ελέγχου επηρεάζει πολύ την αποτελεσματικότητα ελέγχου.

ΠΟΣΟ: πόσο μεγάλο είναι το δείγμα; Μαζί με την ερώτηση του πώς επιλέγονται οι παρατηρήσεις δειγμάτων (επιλογή ελέγχου), είναι πόσες από αυτές επιλέγονται

(επάρκεια ελέγχου *test adequacy*, ή stopping rule). Η αξιοπιστία κάλυψης ή η ανάλυση μετρήσεων αποτελούν δύο «κλασσικές» προσεγγίσεις που απαντούν μια τέτοια ερώτηση.

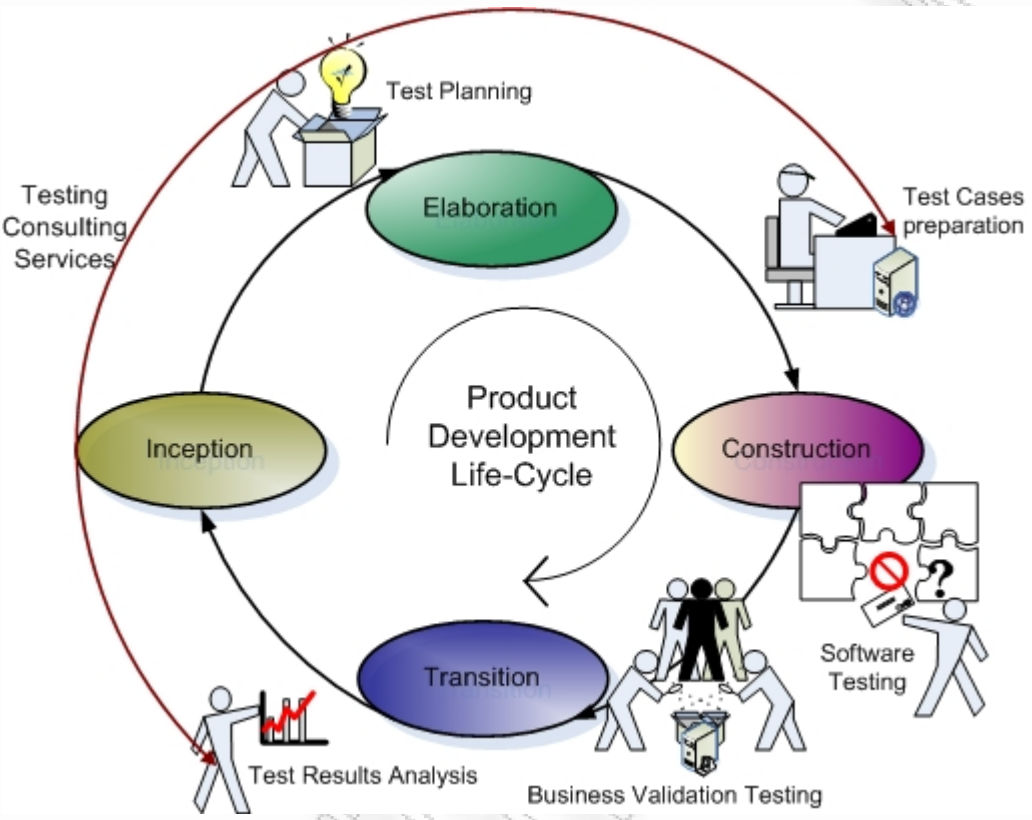
ΤΙ: Ποιο είναι το αντικείμενο που εκτελείται; Λαμβάνοντας υπόψη (πιθανή σύνδεση possibly composite) το σύστημα υπό έλεγχος, παρατηρείται η εκτέλεσή του είτε συνολικά, είτε εστιάζοντας μόνο στο α μέρος του, που μπορεί να είναι λίγο ή πολύ μεγάλο (έλεγχος τμημάτων unit test, έλεγχος συστατικών υποσυστημάτων component/ subsystem test, έλεγχος ολοκλήρωσης integration test), λίγο ή πιο πολύ καθορισμένο: αυτή η πτυχή αναδεικνύεται σε διάφορα επίπεδα ελέγχου, και αποτελεί προϋπόθεση για τα πρώτα επίπεδα που βοηθούν στο να επιτραπεί η εκτέλεση ελέγχου μέρους ενός μεγαλύτερου συστήματος.

Στις περισσότερες των περιπτώσεων, *ο καθορισμός του τι πρέπει να ελεγχθεί ή το πόσο έλεγχος πρέπει να διεξαχθεί, δεν βασίζεται στους εμπλεκόμενους κινδύνους*. Αυτή ακριβώς η αντιμετώπιση έχει ως συνέπεια οι περισσότεροι οργανισμοί να πραγματοποιούν εξαντλητικούς ελέγχους, πιστεύοντας ότι με αυτό τον τρόπο θα σιγουρευτούν ότι δεν υπάρχουν λάθη στο σύστημα, αντί να προσπαθούν να επιτύχουν ένα επίπεδο εμπιστοσύνης ανάλογο προς τους κίνδυνους που εμπλέκονται. Φυσικά αποτυγχάνουν, διότι όπως προαναφέρθηκε, ο εξαντλητικός έλεγχος είναι πάντα αδύνατος. Όμως, είναι πολύ πιθανό να μην έχουν επίγνωση αυτή τους της αποτυχίας. Μπορεί να πιστεύουν ότι επειδή πραγματοποίησαν έναν διεξοδικό έλεγχο, το σύστημα είναι απαλλαγμένο από λάθη. Αυτή είναι και η πιο επικίνδυνη κατάσταση: όχι μόνο ανυπαρξία κριτηρίων βασιζόμενων στους κινδύνους, αλλά και μη αναγνώριση του γεγονότος ότι παραμένει κίνδυνος και μετά τον έλεγχο του συστήματος λογισμικού.

ΠΟΥ: πού εκτελείται η παρατήρηση; Αφορά αυστηρά αυτά που εκτελούνται, είναι η ερώτηση εάν αυτό γίνεται στο εσωτερικό, σε ένα μιμούμενο περιβάλλον ή στο τελικό στοχευόμενο περιεχόμενο. Αυτή η ερώτηση αφορά την υψηλή σχετικότητα όσον αφορά τον έλεγχο ενσωματωμένων συστημάτων.

ΟΤΑΝ: πότε εκτελούνται οι έλεγχοι στον κύκλο ζωής των προϊόντων; Το συμβατικό επιχείρημα είναι το πιο πρόωρο, το καταλληλότερο, δεδομένου ότι το κόστος της αφαίρεσης των ελαττωμάτων / λαθών αυξάνονται καθώς ο κύκλος ζωής προχωρά. Αλλά, μερικές παρατηρήσεις, συνήθως εκείνες που εξαρτώνται από το περιβάλουν πλαισίου, δεν μπορούν πάντα να παρατηρηθούν στο εργαστήριο, και δεν μπορεί να παρατηρηθεί και οτιδήποτε περαιτέρω σημαντικό μέχρι το σύστημα να εγκατασταθεί και να τεθεί σε λειτουργία. Αυτές οι ερωτήσεις παρέχουν ένα πολύ απλό και διαισθητικό σχήμα χαρακτηρισμού των δραστηριοτήτων της διαδικασίας

ελέγχου λογισμικού, οι οποίες μπορούν να βοηθήσουν στην οργάνωση μιας χαρτογράφησης (roadmap) για μελλοντικές ερευνητικές προκλήσεις.



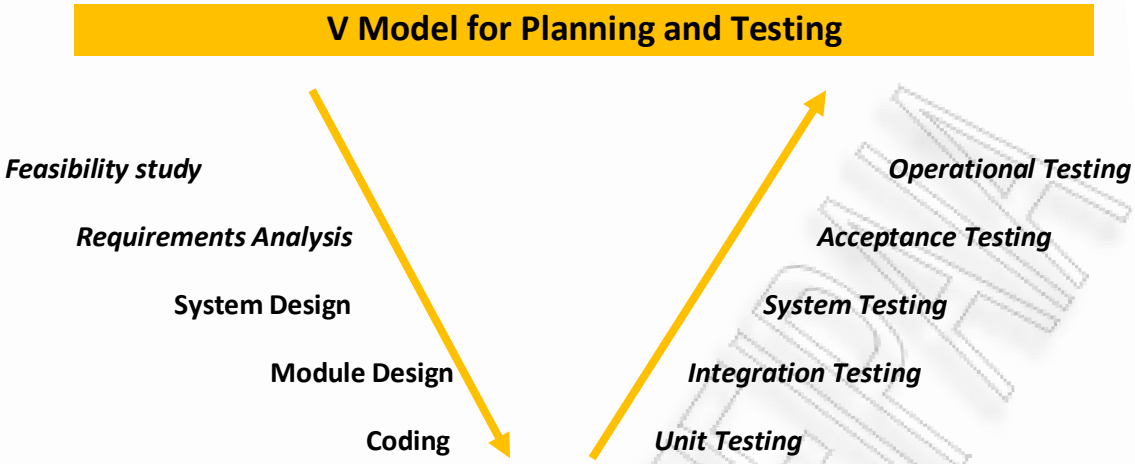
Εικόνα 1 «Διαδικασία ελέγχου λογισμικού».
Πηγή: «Mohd. Ehmer Khan, *Different Forms of Software Testing Techniques for Finding Errors*».

2.2. Το Κλασικό Μοντέλο Ελέγχου

Το κλασικό μοντέλο σχεδίασης ελέγχου σπάει την διαδικασία ελέγχου σε τρείς φάσεις:

1. Έλεγχος μονάδας και συστατικού - *Unit, module and component testing*
2. Έλεγχος ολοκλήρωσης- *Integration testing*
3. Έλεγχος συστήματος - *System testing*
4. Έλεγχος αποδοχής- *Acceptance testing*

Ο έλεγχος παλινδρόμησης είναι ειδικός αφού δεν αποτελεί φάση ελέγχου αλλά μια συνεχή δραστηριότητα που συμβαίνει από την φάση 1 μέχρι την 3



Σχήμα 1 Κλασικό μοντέλο σχεδίασης ελέγχου

Το κλασικό μοντέλο σχεδίασης ελέγχου απεικονίζεται συχνά ως "V" για να επεξηγήσει πιο σαφώς τη σχέση μεταξύ της ανάπτυξης και του ελέγχου λογισμικού. Σύμφωνα με αυτό το πρότυπο, οι περιπτώσεις ελέγχου που εκτελούνται κατά τη διάρκεια των φάσεων ελέγχου στα δεξιά προγραμματίζονται κατά τη διάρκεια των φάσεων ανάπτυξης στα αριστερά. Η μελέτη σκοπιμότητας εφοδιάζει τις περιπτώσεις ελέγχου για τον λειτουργικό έλεγχο, η ανάλυση απαιτήσεων παρέχει τις περιπτώσεις ελέγχου για τον έλεγχο αποδοχής, και ο έλεγχος συστημάτων επηρεάζεται από το σχέδιο του συστήματος. Το σχέδιο ενότητας διευκρινίζει τις διεπαφές που περιλαμβάνονται στον έλεγχο ολοκλήρωσης.

Αφότου η κωδικοποίηση είναι εν εξελίξει, οι περιπτώσεις ελέγχου μπορούν να έχουν προτεραιότητα για τον έλεγχο μονάδων βασισμένο στο σύνολο χαρακτηριστικών γνωρισμάτων και τη δομή συγκεκριμένων μονάδων (κατηγορίες, μέθοδοι, λειτουργίες, κ.λπ.). Αυτό το διάγραμμα δεν προορίζεται να υπονοήσει ένα "Μεγάλο Bang" ανάπτυξης έργου. Λαμβάνοντας υπόψη τις τρέχουσες πρακτικές ανάπτυξης, η ανάπτυξη και οι φάσεις ελέγχου που παρουσιάζονται εδώ πιθανότερο να αντιπροσωπεύουν τη ροή μιας ενιαίας επανάληψης.

Σύμφωνα με τα όσα έως τώρα αναφέρθηκαν, προκύπτει μια συνολική εικόνα της όλης διαδικασίας διαχείρισης του λογισμικού σε συσχέτιση πάντα με την φάση του ελέγχου του. Έτσι, πρώτα απ' όλα και προκειμένου να παρθούν οι διάφορες αποφάσεις ελέγχου, πρέπει απαραιτήτως να έχει προηγηθεί καθορισμός μιας συγκεκριμένης πολιτικής σχετικά με την εκτίμηση του χρόνου και των πόρων που διατίθενται για τη διεξαγωγή του ελέγχου. Στη συνέχεια, σύμφωνα με τα δεδομένα που υπάρχουν, γίνονται οι διάφορες επιλογές από την ομάδα ελέγχου, ή και από άλλα υπεύθυνα για το έργο άτομα και πραγματοποιείται ο έλεγχος του λογισμικού. Έχοντας αποδεχθεί τα όσα προαναφέρθηκαν, η παρούσα εργασία, αφού πρώτα

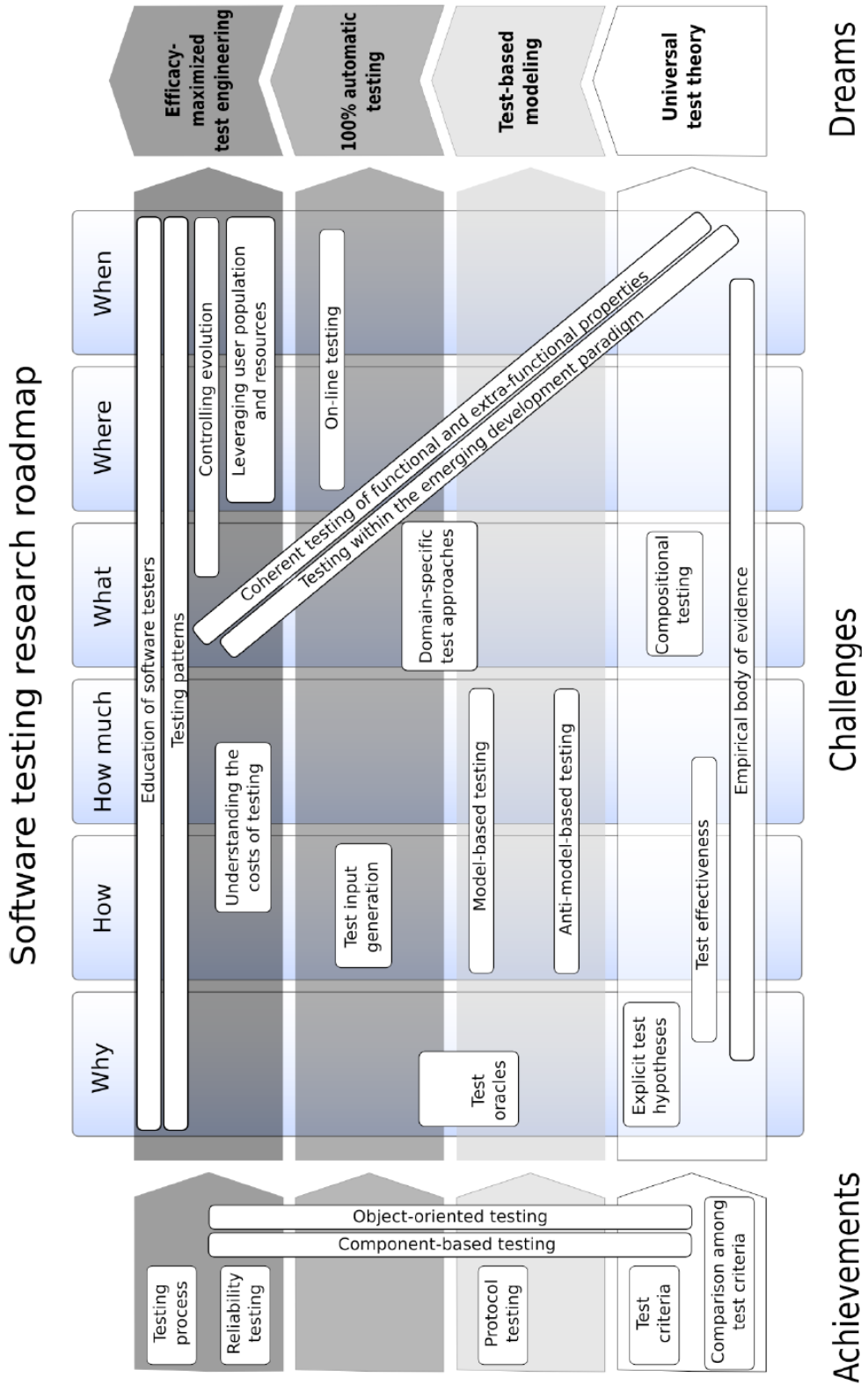
παρουσιάσει τη γενικότερη ιδέα των εννοιών, μέσω διαφόρων προσεγγίσεων και θεωρήσεων που έχουν κατά καιρούς αναπτυχθεί, επιχειρεί να εστιαστεί στη φάση του ελέγχου και να παρουσιάσει μια μελέτη αντίστοιχης περίπτωσης ελέγχου.

2.3. Το Roadmap της έρευνας ελέγχου λογισμικού

Ένα roadmap[3] παρέχει τις κατευθύνσεις για να φθάσουμε σε ένα επιθυμητό προορισμό. Η έρευνα ελέγχου λογισμικού roadmap οργανώνεται ως εξής:

Ο επιθυμητός προορισμός απεικονίζεται από την μορφή ενός συνόλου (τεσσάρων) ονείρων: χρησιμοποιείται αυτός ο όρος για να δηλώσουμε ότι αυτοί είναι ασυμπτωτικοί στόχοι στο τέλος τεσσάρων προσδιορισμένων διαδρομών για την πρόοδο της έρευνας.

Στη μέση είναι οι προκλήσεις που αντιμετωπίζονται από την τρέχουσα και μελλοντική έρευνα ελέγχου λογισμικού, σε νωρίτερο ή πιο ώριμο στάδιο, και με λιγότερες ή περισσότερες πιθανότητες για επιτυχία. Αυτές οι προκλήσεις αποτελούν τις κατευθύνσεις που ακολουθούνται από το ταξίδι προς το όνειρο, και ως έτσι είναι τα κύρια, πιο σημαντικά μέρη του roadmap.



Σχήμα 2 *Roadmap*

Πηγή : « Antonia Bertolino,IEEE 2007 Software Testing Research: Achievements, Challenges, Dreams»

Στο roadmap που παρουσιάζεται στο σχήμα 2, έχει τοποθετηθεί η αναδυόμενη έρευνα και οι τρέχουσες ερευνητικές κατευθύνσεις στο κέντρο, με πιο σημαντικά

θέματα - τα επιτεύγματα από τη μια και οι τελικοί στόχοι - τα όνειρα(μελλοντικοί στόχοι) από την άλλη. Οι τέσσερις οριζόντιες λουρίδες απεικονίζουν τις προσδιορισμένες ερευνητικές διαδρομές προς τα όνειρα, δηλαδή:

1. Παγκόσμια θεωρία Ελέγχου
2. Μοντέλο βασισμένο στον έλεγχο
3. 100% Αυτόματος έλεγχος
4. Δυναμικός μηχανικός έλεγχος

Οι διαδρομές είναι διαταγμένες από κάτω προς τα επάνω σε σχέση με την προοδευτική χρησιμότητα: η θεωρία είναι στη βάση των υιοθετημένων προτύπων, τα οποία απαιτούνται στη συνέχεια για την αυτοματοποίηση, η οποία συμβάλλει στην οικονομική αποδοτική εφαρμογή της μηχανικής ελέγχου. Η έκταση των προκλήσεων είναι οριζόντια άνω των έξι κάθετων λουρίδων που αντιστοιχούν στο ΓΙΑΤΙ, ΠΩΣ, ΠΟΣΟ, ΤΙ, ΠΟΥ, και ΠΟΤΕ ερωτήματα που χαρακτηρίζουν τον έλεγχο λογισμικού (χωρίς συγκεκριμένη σειρά).

Οι ερευνητικές προκλήσεις ελέγχου λογισμικού βρίσκουν τη θέση τους μέσα στο σχέδιο, κάθετα ανάλογα με το μακροπρόθεσμο όνειρο ή όνειρα, κυρίως προς εκείνα που τείνουν, και οριζόντια ανάλογα με το ποια ερώτηση, ή ποιες ερωτήσεις, του προτεινόμενου χαρακτηριστικού του ελέγχου λογισμικού στρέφονται κυρίως. Στο υπόλοιπο αυτού του εγγράφου, θα συζητηθούν τα στοιχεία (επιτεύγματα, προκλήσεις, όνειρα) αυτού του roadmap. Θα συγκριθεί συχνά αυτό το roadmap με προκάτοχό του το 2000 από τον Harrold[4], το οποίο θα αναφέρεται εφεξής ως FOSE2000.

2.4. Ιστορική αναφορά στον έλεγχο Λογισμικού

Πριν περιγράφουν οι μελλοντικές διαδρομές της έρευνας ελέγχου λογισμικού, γίνεται μια προσπάθεια εδώ για ένα στιγμιότυπο μερικών θεμάτων που αποτελούν «το σώμα» της γνώσης στον έλεγχο λογισμικού και έχουν καθιερωθεί ως σημαντικά ερευνητικά επιτεύγματα. Στο roadmap του σχήματος 2, αυτά αντιπροσωπεύονται στην αριστερή πλευρά. Η προέλευση της βιβλιογραφίας του ελέγχου λογισμικού είναι από την αρχή της δεκαετίας του '70 (αν και οποιοσδήποτε μπορεί να φανταστεί ότι η ίδια η έννοια γεννήθηκε ταυτόχρονα με τις πρώτες εμπειρίες στον προγραμματισμό). Ο Hetzel[5] **There are no sources in the current document.** χρονολογεί την πρώτη διάσκεψη αφιερωμένη στον έλεγχο κώδικα το 1972. *Ο έλεγχος συλλήφθηκε όπως μια τέχνη, και εξηγήθηκε ως η*

«καταστρεπτική» μέθοδος εκτέλεσης ενός προγράμματος με την πρόθεση εύρεσης λαθών, που αντιτάσσεται στη σχεδίαση που αποτέλεσε «το εποικοδομητικό» συμβαλλόμενο μέρος.

Είναι αυτά τα χρόνια που γίνεται ο κορυφαίος αναφερόμενος αφορισμός του Dijkstra για τον έλεγχο λογισμικού, που μπορεί μόνο να παρουσιάσει την παρουσία ελαττωμάτων, αλλά ποτέ την απουσία τους . Η δεκαετία του '80 είδε την υπόθεση του ελέγχου λογισμικού από μια πιο μηχανική αρχή, και από μια άποψη αλλαγής του στόχου από μια μόνο ανακάλυψη λάθους σε μια πιο περιεκτική και θετική άποψη, αυτή της πρόληψης των λαθών. Ο έλεγχος λογισμικού χαρακτηρίζεται πλέον ως μια ευρεία και συνεχής δραστηριότητα σε όλη τη διαδικασία ανάπτυξης λογισμικού, της οποίας ο στόχος είναι η μέτρηση και η αξιολόγηση των ιδιοτήτων και των ικανοτήτων του λογισμικού, και ο Beizer[6] δηλώνει: Πέρα από την διαδικασία ελέγχου λογισμικού , η διαδικασία του σχεδιασμού των ελέγχων είναι ένας από τους καλύτερους και πιο γνωστούς τρόπους για την αποφυγή σφαλμάτων. Πράγματι πολύ έρευνα έχει ωριμάσει από τα πρώτα χρόνια μέσα από τεχνικές και εργαλεία που βοηθούν στο να γίνει μια τέτοια «σκέψη ελέγχου λογισμικού» πιο συστηματική και να ενσωματωθεί μέσα στη διαδικασία ανάπτυξης.

Έχουν προταθεί διάφορα μοντέλα διαδικασίας ελέγχου για βιομηχανική έγκριση, μεταξύ των οποίων το δημοφιλέστερο πιθανόν είναι το “Vmodel” (βλέπε *σχήμα 1*). Πολλά από τα συστατικά του μοιράζονται τη διάκριση επιπέδων ελέγχου όπως είναι ο έλεγχος ολοκλήρωσης ,ο έλεγχος της μονάδας και ο έλεγχος του συστήματος. Πρόσφατα υποστηρίχτηκε από μερικούς ως ανεπαρκής η επίπτωση του V μοντέλου μιας συγχρονισμένης και τυπικά τεκμηριωμένης διαδικασίας ελέγχου, κάτι που οδήγησε να υποστηριχτούν σε αντίθεση πιο ευέλικτες διαδικασίες. Σχετικά με τον έλεγχο πιο συγκεκριμένα, ένα διαφορετικό πρότυπο που κερδίζει την προσοχή είναι η test-driven ανάπτυξη , μια από τις πιο κύριες πρακτικές ακραίου προγραμματισμού. Η καθιέρωση μιας κατάλληλης διαδικασίας για έλεγχο λογισμικού ήταν απαριθμημένη στο FOSE2000 μεταξύ των θεμελιωδών ερευνητικών θεμάτων..

2.5. Προβλήματα κατά τον έλεγχο Λογισμικού

Ο έλεγχος λογισμικού δεν έχει την δυνατότητα να αναγνωρίσει όλες τις ατέλειες μέσα σε ένα λογισμικό. Αντίθετα απλά συγκρίνει την κατάσταση και την συμπεριφορά του ίδιου του προϊόντος με διάφορους κανόνες ή μηχανισμούς με τους οποίους μπορεί κάποιος να αναγνωρίσει το πρόβλημα. Αυτοί οι κανόνες μπορούν να αφορούν τις προδιαγραφές, όμοια προϊόντα, παλαιότερες εκδόσεις του ίδιου του προϊόντος, συμπεράσματα όσο αφορά αναμενόμενα ή ηθελημένα αποτελέσματα, προσδοκίες των πελατών ή χρηστών, διάφορα standards που

αφορούν το προϊόν, νόμους που εφαρμόζονται κατά την χρήση του και άλλα κριτήρια. Υπό αυτή την προοπτική τίθεται και το κρίσιμο ζήτημα **καθορισμού των κριτηρίων** παύσης ελέγχου, διότι προκειμένου να πραγματοποιηθεί ο έλεγχος του λογισμικού, πρέπει να λαμβάνονται υπόψη σημαντικοί παράγοντες που επηρεάζουν την όλη διαδικασία, όπως για παράδειγμα οι πόροι που διατίθενται σε κάθε περίπτωση, εννοώντας κυρίως τα χρονικά και οικονομικά περιθώρια που δίνονται και τα οποία πρέπει να τηρούνται από την ομάδα ελέγχου. Γιατί βέβαια, είναι δυνατό να σχεδιάζονται πολύ περισσότεροι έλεγχοι από ότι θα ήταν ποτέ εφικτό να πραγματοποιηθούν και είναι γνωστό, πως ο εξαντλητικός έλεγχος θα σήμαινε ότι πολλά προϊόντα δεν θα έφταναν ποτέ στην αγορά, γιατί ο έλεγχος δεν θα ολοκληρωνόταν ποτέ, αλλά και αν ακόμα έφταναν η τιμή τους θα κυμαινόταν σε υπερβολικά υψηλά επίπεδα.

2.5.1. Κριτήρια ελέγχου.

Εξαιρετικά πλούσιο είναι το σύνολο κριτηρίων ελέγχου που επινοούνται από παρελθοντικές έρευνες για να βοηθήσουν το συστηματικό προσδιορισμό στις διάφορες περιπτώσεις ελέγχου. Παραδοσιακά αυτοί οι έλεγχοι έχουν διακριθεί μεταξύ του άσπρου κουτιού (white-box) και του μαύρου κουτιού (black-box), ανάλογα με το εάν ο πηγαίος κώδικας αξιοποιείται ή όχι κατά τη διάρκεια ελέγχου. Μια πιο καθαρισμένη ταξινόμηση μπορεί να γίνει σύμφωνα με τον πηγαίο κώδικα από τον οποίο προέρχονται οι περιπτώσεις ελέγχου, υπάρχουν πολλά εγχειρίδια και άρθρα ερευνών που παρέχουν περιεκτικές περιγραφές των υπαρχόντων κριτηρίων. Πράγματι, σήμερα υπάρχουν πολλά κριτήρια μεταξύ του τι να επιλεχθεί, έτσι ώστε η πραγματική πρόκληση να είναι η ικανότητα να γίνει μια δίκαιη επιλογή, ή μάλλον να γίνει κατανοητό πώς μπορούν να συνδυαστούν τα κριτήρια αποτελεσματικότερα. *Σήμερα έχει δοθεί περισσότερη προσοχή στον έλεγχο βασισμένο στο πρότυπο*. Είναι επίσης σημαντικό για την ορθή διεξαγωγή του ελέγχου, να λαμβάνεται υπόψη και το γεγονός ότι εκτός από ορισμένα κομμάτια του λογισμικού που είναι πιθανό να εμπεριέχουν περισσότερα λάθη από κάποια άλλα, υπάρχουν και τμήματα του λογισμικού με μεγαλύτερη συχνότητα χρήσης από άλλα, με αποτέλεσμα να αυξάνεται η πιθανότητα αποτυχίας του συστήματος λογισμικού εάν αυτά εμπεριέχουν κάποιο λάθος.

Έλεγχος Λογισμικού

Αν λοιπόν ο *απώτερος σκοπός για κάθε προϊόν*, είναι να πραγματοποιείται ***ο πλέον αποδοτικός ως προς το κόστος έλεγχος,*** που να ***διαβεβαιώνει*** ότι είναι αρκετά ***αξιόπιστο***, αρκετά ***ασφαλές*** και ***ικανοποιεί*** τις ***απαιτήσεις*** του χρήστη/πελάτη, διαπιστώνεται ότι κάτι τέτοιο είναι εξαιρετικά δύσκολο, αν όχι ακατόρθωτο να επιτευχθεί, από τη στιγμή που δεν υπάρχει ποτέ αρκετός χρόνος για να ελεγχθούν τα πάντα πλήρως. Το γεγονός αυτό συνηγορεί υπέρ της απόψεως ότι οι αποφάσεις ελέγχου, όπως για παράδειγμα σε ποιες περιοχές πρέπει να εστιαστεί ο έλεγχος, ποια μέθοδος θα ακολουθηθεί ή το πότε πρέπει να σταματήσει, να λαμβάνονται μεταξύ των άλλων και βάσει των εμπλεκομένων κινδύνων.

Για να γίνει λοιπόν ο έλεγχος ενός λογισμικού πρέπει να επικεντρωθεί κανείς σε κάποια κριτήρια. Τα κριτήρια αυτά αναφέρονται σε διαφορετικούς τομείς του προγράμματος. Στόχος των κριτηρίων αυτών είναι να ανακαλύψει του τομείς του προγράμματος που δεν θα εκτελεστούν από ένα σύνολο περιπτώσεων ελέγχου. Μετά τον εντοπισμό των κομματιών αυτών δημιουργούνται πρόσθετες περιπτώσεις ελέγχου για να αυξηθεί η κάλυψη του κώδικα και εμμέσως η ποιότητα του ελέγχου που γίνεται. Τα κριτήρια ελέγχου μπορούν να χωριστούν στις ακόλουθες ομάδες:

Τα κριτήρια ελέγχου[7] μπορούν να χωριστούν στις ακόλουθες ομάδες:

• Structural testing

Τα κριτήρια ελέγχου καθορίζονται με βάση την κάλυψη συγκεκριμένων ομάδων στοιχείων της δομής του προγράμματος. Για παράδειγμα στο path testing όπου ελέγχεται ένα-ένα μονοπάτι ξεχωριστά.

• Fault-base testing

Τα κριτήρια ελέγχου καθορίζονται έτσι ώστε να επικεντρώνονται στην ανίχνευση λαθών στο λογισμικό.

- Error base testing

Τα κριτήρια ελέγχου επικεντρώνονται σε διάφορες περιπτώσεις ελέγχου οι οποίες επιλέχθηκαν έτσι ώστε να εκτελούν συγκεκριμένες λειτουργίες του προγράμματος για τον έλεγχο συγκεκριμένων λαθών.

Εκτός όμως από τα παραπάνω, εξίσου σημαντική παράμετρο αποτελεί και η κρισιμότητα του λογισμικού η οποία μπορεί να καθορίζεται από τον σκοπό για τον οποίο πρόκειται να χρησιμοποιηθεί το προϊόν, το ποιος πρόκειται να το χρησιμοποιήσει και τι απαιτήσεις έχει από αυτό, ή το ποιες θα είναι οι πιθανές

συνέπειες εάν τελικά δεν λειτουργήσει κατά το αναμενόμενο. Γιατί βέβαια, δεν είναι σπάνιες οι περιπτώσεις όπου σπαταλούνται άδικα πολύτιμοι πόροι οργανισμών σε ελέγχους λογισμικού που είτε δεν παρουσιάζει ιδιαίτερη κρισιμότητα, είτε και αν ακόμη παρουσιάζει, δεν υπάρχει αναγνώριση ορισμένων περιοχών του λογισμικού που είναι περισσότερο «κρίσιμες» για την ορθή λειτουργία του, έτσι ώστε ο έλεγχος να εστιαστεί σε αυτές.

2.5.2. Σύγκριση μεταξύ των κριτηρίων ελέγχου.

Παράλληλα με την έρευνα για τα κριτήρια επιλογής ελέγχου και για την επάρκεια ελέγχου, έχει γίνει πολύ έρευνα για την αξιολόγηση της σχετικής αποτελεσματικότητας των διάφορων κριτηρίων ελέγχου, και ειδικά των παραγόντων που καθιστούν μια τεχνική καλύτερη από μια άλλη στην εύρεση σφαλμάτων[8]. Οι προηγούμενες μελέτες έχουν συμπεριλάβει διάφορες αναλυτικές συγκρίσεις μεταξύ διαφορετικών τεχνικών. Αυτές οι μελέτες έχουν επιτρέψει να καθιερωθεί μια ένταξη ιεραρχίας της σχετικής πληρότητας μεταξύ συγκρίσιμων κριτήριων, και κατανόηση των παραγόντων που επηρεάζουν την πιθανότητα εύρεσης λαθών, που εστιάζει περισσότερο στη σύγκριση του τμήματος (δηλ., συστηματικού) σε σχέση με τον τυχαίο έλεγχο. «Καταδεικνύοντας την αποτελεσματικότητα των τεχνικών ελέγχου» που προσδιορίστηκε στην πραγματικότητα ως η βασική έρευνα στο FOSE2000, απαιτεί ακόμα και σήμερα ο στόχος αυτός περαιτέρω έρευνα, μόνο που η έμφαση δίνεται τώρα στην εμπειρική αξιολόγηση.

2.6. Στόχος : Καθολική θεωρία ελέγχου

Με τον όρο μιας «καθολικής» θεωρίας ελέγχου εννοείται ένα συνεπές και αυστηρό πλαίσιο στο οποίο οι ελεγκτές μπορούν να αναφερθούν για να καταλάβουν τις σχετικές δυνάμεις και τους περιορισμούς των υπαρχουσών τεχνικών ελέγχου, και να καθοδηγηθούν στο να επιλέξουν την επαρκέστερη, ή το μίγμα αυτών, λαμβάνοντας υπόψη τους παρόντες συντελεστές. Η δημιουργική εργασία στην θεωρία ελέγχου λογισμικού χρονολογείται από την πρόσφατη δεκαετία του '70, όταν εισήχθησαν αρχικά οι σχετικές έννοιες μιας «αξιόπιστης» ή «ιδανικής» ακολουθίας ελέγχου[9]. Χάρις σε αυτήν την πρωτοποριακή εργασία, υπάρχουν τα λογικά επιχειρήματα για να επιβεβαιωθεί το αρκετά προφανές γεγονός ότι ο έλεγχος δεν μπορεί ποτέ να είναι ακριβής . Αλλά αυτή καθ' εαυτή η γνώση, εκτός από την προειδοποίηση ότι παρόλο που έγιναν πολλοί έλεγχοι το λογισμικό μπορεί ακόμα να είναι ελαττωματικό, παρέχει λίγες οδηγίες σχετικά με το τι είναι αυτό που μπορούμε να συμπεράνουμε για το ελεγμένο λογισμικό έχοντας εφαρμόσει μια επιλεγμένη

τεχνική, ή πηγαίνοντας ακόμα παραπέρα σχετικά με το πώς μπορούμε δυναμικά να συντονίσουμε την στρατηγική ελέγχου με συσσωρευμένα αποτελέσματα λαμβάνοντας υπόψη το τι παρατηρείται.

2.6.1. Πρόκληση: Ρητές υποθέσεις ελέγχου

Δεδομένου ότι ο έλεγχος είναι απαραίτητος, αυτή η καθολική θεωρία πρέπει επίσης να γίνει σαφής για κάθε τεχνική της οποίας είναι ελλοχεύουσες οι υποθέσεις της:

Αρχικά η επισημοποίηση, της έννοιας υπόθεσης ελέγχου δικαιολογεί την κοινή και διαισθητική πρακτική ελέγχου πίσω από την επιλογή του κάθε συνόλου πεπερασμένων ελέγχων, με την οποία λαμβάνεται αντιπροσωπευτικό δείγμα για διάφορες πιθανές εκτελέσεις. Με την εξαίρεση κάποιων επίσημων προσεγγίσεων ελέγχου, οι υποθέσεις ελέγχου μένουν συνήθως αόριστες, ενώ θα ήταν πολύ σημαντικό να καταστούν σαφής.

Μια σύνοψη των υποθέσεων ελέγχου πίσω από τις πιο κοινές προσεγγίσεις ελέγχων δίνεται, για παράδειγμα, από τον Gaudel[10], που αναφέρει, μεταξύ άλλων, την *Uniformity Hypothesis* (υπόθεση ομοιομορφίας) για τα κριτήρια μερών του blackbox (το λογισμικό θεωρείται πως συμπεριφέρεται ομοιόμορφα σε κάθε έλεγχο), και η *Regularity Hypothesis* (υπόθεση κανονικότητας), που χρησιμοποιεί μια λειτουργία μεγέθους κατά τη διάρκεια των ελέγχων. Η έρευνα αυτή πρέπει να επεκταθεί για να καλύψει άλλα κριτήρια και προσεγγίσεις. Οι υποθέσεις ελέγχου πρέπει να χωριστούν σε ενότητες με βάση το αντικείμενο ελέγχου: διάφορες θεωρίες/υποθέσεις θα ήταν αναγκαίες κατά τον έλεγχο της αξιοπιστίας τους, κατά τον έλεγχο για εντοπισμό σφαλμάτων, κ.ο.κ

Κάνοντας σαφής τις υποθέσεις, αυτή η πρόκληση εξηγεί το ΓΙΑΤΙ παρατηρούνται μερικές εκτελέσεις.

2.6.2. Πρόκληση: Αποτελεσματικότητα ελέγχου

Για να καθιερωθεί μια χρήσιμη θεωρία για τον έλεγχο, πρέπει να αξιολογηθεί η αποτελεσματικότητα των υφιστάμενων και νέων κριτήριων ελέγχων. Αν και όπως έχει ειπωθεί ανάμεσα στα επιτεύγματα, έχουν πραγματοποιηθεί αρκετές συγκριτικές μελέτες για αυτό το σκοπό, ο Fose2000 έχει ήδη επισημάνει ότι χρειάζονται συμπληρωματικές έρευνες για να παραχθούν αναλυτικά, στατιστικά, ή εμπειρικά αποδεικτικά της αποτελεσματικότητας των κριτηρίων επιλογής-ελέγχου στην αποκάλυψη σφαλμάτων. Οι προκλήσεις αυτές ισχύουν ακόμα. Ειδικότερα, σήμερα είναι **γενικά αποδεκτό** ότι είναι πάντα περισσότερο αποτελεσματικό να

χρησιμοποιηθεί ένας **συνδυασμός τεχνικών**, αντί να εφαρμοστεί μόνο μια, ακόμη και αν κριθεί η πιο ισχυρή, επειδή κάθε τεχνική μπορεί να στοχεύει σε διαφορετικούς τύπους σφαλμάτων.

Αρκετά έργα έχουν συμβάλει στην καλύτερη κατανόηση εγγενών περιορισμών ελέγχου διαφορετικών προσεγγίσεων, ξεκινώντας από τον Άμλετ και τον Τέιλορ[11] και την ερευνά τους που αναφέρεται στον έλεγχο μερών και τις βασικές υποθέσεις. Απαιτείται ακόμη περαιτέρω εργασία, ιδίως για να καταστούν σαφείς οι συγκρίσεις σε σχέση με την πολυπλοκότητα που υπάρχει στον πραγματικό κόσμο των ελέγχων (για παράδειγμα, ο Zhu και ο He[12] που αναλύουν την επάρκεια των ελέγχων σε ταυτόχρονα συστήματα), καθώς και την βελτίωση υποθέσεων στις βάσεις τέτοιων συγκρίσεων, προκειμένου να ληφθούν υπόψη οι εξελίξεις στην αυτοματοποίηση του ελέγχου. Για παράδειγμα, ακόμη και η συμβατική διαμάχη σχετικά με τα συσχετιζόμενα πλεονεκτήματα των συστηματικών εναντίον τυχαίων τεχνικών έχει σήμερα ανανεωθεί από τις επερχόμενες εξελιγμένες μεθόδους για την αυτοματοποίηση της γενιάς τυχαίου ελέγχου (τα οποία αναλύονται στην ενότητα 2.2). Αυτή η πρόκληση αντιμετωπίζει το ΓΙΑΤΙ, το ΠΩΣ και το ΠΟΣΟ ΠΟΛΥ των ελέγχων, από την άποψη των λαθών (σε ποια και πόσα πολλά) που στοχεύουμε.

2.6.3. Πρόκληση: Σύνθεση ελέγχου

Η ολοένα αυξανόμενη πολυπλοκότητα του λογισμικού καθιστά δύσκολο τον έλεγχο του και εμποδίζει την πρόοδο προς την κατεύθυνση μιας ονειρικής έρευνας, συμπεριλαμβανομένης και της θεωρίας ελέγχου. Παραδοσιακά, η πολυπλοκότητα του ελέγχου λογισμικού αντιμετωπιζόταν με βάση την αρχαία στρατηγική διαίρει και βασίλευε, δηλαδή, ο έλεγχος ενός μεγάλου πολύπλοκου συστήματος αποσυντίθεται σε ξεχωριστούς ελέγχους των αποτελούμενων «μερών» του. Μεγάλο μέρος παρελθοντικής έρευνας ασχολήθηκε, με τεχνικές και εργαλεία που βοηθούν στοιχειώδης στρατηγικές ελέγχων στην οργάνωση και εκτέλεση διαφορετικών προοδευτικά ομαδοποιήσεων στοιχείων. Το πρόβλημα έχει γίνει ιδιαίτερα σχετικό σήμερα με την εμφάνιση του πρότυπου ανάπτυξης CB, όπως έχει συζητηθεί στον FOSE2000, και ακόμη περισσότερο με την αυξανόμενη υιοθέτηση των συνθέσεων ενός δυναμικού συστήματος. Έτσι, χρειάζεται ένα κεφάλαιο της θεωρίας των ελέγχων που θα βοηθήσει στην σύνθεση ελέγχου: πρέπει να γίνει κατανοητό πώς μπορούν να επαναχρησιμοποιηθούν τα αποτελέσματα των ελέγχων που παρατηρήθηκαν στον ξεχωριστό έλεγχο των μεμονωμένων μερών (όπως μονάδων ή κατασκευαστικών στοιχείων ή υποσυστημάτων), ιδίως ό, τι συμπεράσματα μπορούν να συναχθούν για το σύστημα, και ποιές επιπλέον περιπτώσεις ελέγχου πρέπει να εκτελεστούν με την συνένωση.

Για παράδειγμα, ο Άμλετ έχει προτείνει μια απλή θεμελιακή θεωρία για την αξιοπιστία λογισμικού βασισμένη στο περιεχόμενο, πρόσφατα επεκτάθηκε με την έννοια της «κατάστασης», αλλά χρειάζεται αρκετή δουλειά ώστε να εφαρμοστεί γενικά. Ο Blundell και οι συνεργάτες[13] του ερευνούν αντί αυτού την εφαρμογή στον έλεγχο της εγγυημένης υπόθεσης μιας τεχνικής επαλήθευσης που χρησιμοποιείται για να συναχθούν καθολικές ιδιότητες του συστήματος από τον μεμονωμένο έλεγχο των επιμέρους συστατικών. Για να είναι σε θέση να ελεγχθεί ένα συστατικό μεμονωμένα, θα πρέπει να γίνουν υποθέσεις σχετικά με το περιεχόμενό του, η εγγύηση υπόθεσης ελέγχει αν ένα συστατικό εγγυάται μια ιδιότητα υποθέτοντας ότι το πλαίσιο συμπεριφέρεται σωστά, και στη συνέχεια ελέγχεται συμμετρικά υποθέτοντας ότι η συνιστώσα είναι σωστή. Η υπόσχεση του ελέγχου εγγύησης – υπόθεσης θα είναι ότι με την παρατήρηση των ιχνών ελέγχου επιμέρους συστατικών θα μπορούσε κανείς να συμπεράνει γενικές συμπεριφορές.

Η κοινότητα ελέγχου πρωτόκολλου είναι επίσης ενεργή στην έρευνα σύνθεσης ελέγχου. Για παράδειγμα, ο Van der Bijl και οι συνεργάτες του [14] έχουν αναλύσει επισήμως την παράλληλη σύνθεση δύο στοιχείων επικοινωνίας, με βάση τη θεωρία ioco-ελέγχου[15], που λειτουργεί σε συστήματα μεταβάσεων με ετικέτες. Ειδικότερα, εάν δύο στοιχεία έχουν ελεγχθεί ξεχωριστά και έχει αποδειχθεί ότι είναι σωστό το ioco τους, είναι σωστό και το ioco μετά την ένωση τους; Οι συντάκτες δείχνουν ότι, σε γενικές γραμμές αυτό δεν μπορεί να συναφθεί, αλλά η απάντηση μπορεί να είναι καταφατική για τα στοιχεία των οποίων οι εισροές είναι πλήρως καθορισμένες. Οι Gotzhein και Khendek[16] αντίθετα θεώρησαν τον κωδικό «glue» για την ενσωμάτωση των στοιχείων επικοινωνίας, έχοντας φτιάξει ένα μοντέλο σφάλματος για το οποίο ανέπτυξαν μια διαδικασία ώστε να βρεθούν οι περιπτώσεις ελέγχου με τον κωδικό glue.

Αυτή η πρόκληση σχετίζεται σαφώς, με το τι μπορεί να ελεγχθεί.

2.6.4. *Πρόκληση: Το εμπειρικό σύνολο των τεκμηρίων ελέγχου*

Σήμερα η σημασία του πειραματισμού με σκοπό να προάγει την ωριμότητα της αρχής της τεχνολογίας λογισμικού δεν χρειάζεται σίγουρα να τονιστεί (ο Sioberg και οι συνεργάτες του συζητούν σε βάθος τις προκλήσεις της έρευνας που προέρχονται από τις εμπειρικές μεθόδους)[17]. Σε κάθε θέμα της έρευνας τεχνολογίας λογισμικού, οι εμπειρικές μελέτες είναι ουσιώδεις για την αξιολόγηση προτεινόμενων τεχνικών και πρακτικών, για να γίνει κατανοητό το πώς και πότε λειτουργούν και να

βελτιωθούν πάνω σε αυτές. Αυτό είναι προφανώς αλήθεια και για τον έλεγχο, στον οποίο ο ελεγχόμενος πειραματισμός είναι μια απαραίτητη έρευνα μεθοδολογίας[18] .

Στο FOSE2000, ο Harrold προσδιόρισε από αυτή την άποψη τις εξής ανάγκες: ελεγχόμενα πειράματα όπου θα παρουσιαστούν οι τεχνικές. Η συλλογή και η δημόσια διαθεσιμότητα θέτει πειραματικά θέματα και βιομηχανικό πειραματισμό. Όλες αυτές οι ανάγκες μπορούν να επιβεβαιωθούν σήμερα, και μια πιο πρόσφατη επισκόπηση των πειραμάτων τεχνικής ελέγχου που κατέληξε δυστυχώς στο συμπέρασμα ότι πάνω από το ήμισυ της υπάρχουσας γνώσης (τεχνική ελέγχου) βασίζεται, στις εντυπώσεις και αντιλήψεις και, ως εκ τούτου, είναι άνευ οποιασδήποτε επίσημης βάσης. Πράγματι, από τον πειραματισμό, θα πρέπει να στοχεύει κανείς στην παραγωγή μιας εμπειρικής γνώσης, η οποία αποτελεί τη βάση για τη δημιουργία και την εξέλιξη της θεωρίας για τον έλεγχο. Και για αυτό το λόγο θα πρέπει να υπάρχουν πειράματα με νόημα, από την άποψη της κλίμακας, από τα θέματα που χρησιμοποιούνται, καθώς και του περιεχομένου, που δεν είναι πάντα ρεαλιστικό. Συνήθως, και για τις τρεις πτυχές, το εμπόδιο είναι το κόστος: οι προσεκτικές εμπειρικές μελέτες προϊόντων μεγάλης κλίμακας, μέσα σε περιβάλλοντα πραγματικού κόσμου, και ενδεχομένως η αναπαραγωγή από πολλούς επαγγελματίες ελεγκτές έτσι ώστε να επιτευχθούν γενικά έγκυρα αποτελέσματα είναι φυσικά απαγορευτικά ακριβό. Ένας πιθανός τρόπος να ξεπεραστούν τα εν λόγω φράγματα θα μπορούσε να είναι η ένωση των δυνάμεων σε αρκετές ερευνητικές ομάδες και η διανομή ενός ευρέως αναπαραγόμενου πειράματος. Η ιδέα θα είναι περίπου αυτή της λειτουργίας της ταξινόμησης αρχικά των "Ανοιχτών πειραμάτων", παρόμοια με τον τρόπο που πολλά έργα ανοικτού κώδικα έχουν πραγματοποιηθεί με επιτυχία.

Σήμερα όλο και μεγαλώνει η συνειδητοποίηση της ανάγκης να ενωθούν οι δυνάμεις και λαμβάνονται ήδη ορισμένες προσπάθειες προς τη δημιουργία αποθήκευσης διαμοιραζόμενων δεδομένων, όπως είναι οι κατανεμημένοι πειραματικοί έλεγχοι, όπως η συλλογή PlanetLab[19] με περισσότερα από 700 συνδεδεμένα μηχανήματα με όλο τον κόσμο.

Αυτή είναι μια θεμελιώδης πρόκληση που εκτείνεται στα έξι χαρακτηριστικά ερωτήματα του roadmap (βλ Σχήμα 2).

2.6.5. Αντικειμενοστραφής έλεγχος (Object-oriented testing)

Στη δεκαετία του '90 η εστίαση ήταν στον έλεγχο του αντικειμενοστρεφούς λογισμικού (Object Oriented,OO). Απορρίπτοντας τον μύθο ότι η ενισχυμένη συναρμολογησιμότητα και η επαναχρησιμοποίηση που τέθηκαν προς συζήτηση με

τον Object Oriented προγραμματισμό θα μπορούσαν ακόμη και να αποτρέψουν την ανάγκη για έλεγχο, οι ερευνητές σύντομα συνειδητοποίησαν ότι όχι μόνο όλα όσα ήδη έμαθαν για τον έλεγχο λογισμικού εφαρμόστηκαν επίσης και για τον κώδικα ΟΟ, αλλά και η ανάπτυξη ΟΟ εισήγαγε νέους κινδύνους και δυσκολίες, αυξάνοντας ως εκ τούτου την ανάγκη και την πολυπλοκότητα του ελέγχου. Συγκεκριμένα, μεταξύ των κύριων μηχανισμών της ΟΟ ανάπτυξης, η ενθυλάκωση μπορεί να βοηθήσει να μείνουν κρυφά λάθη και να καταστήσουν τον έλεγχο πιο απαιτητικό, η κληρονομιά απαιτεί τον εκτενή επανέλεγχο του κληρονομημένου κώδικα και ο πολυμορφισμός και η δυναμική κλήση συνδέσεων νέα πρότυπα κάλυψης. Εκτός αυτού απαιτούνται, κατάλληλες στρατηγικές για τον αποτελεσματικό επ'αυξητικό έλεγχο ολοκλήρωσης και για να χειριστούν το σύνθετο φάσμα των πιθανών στατικών και δυναμικών εξαρτήσεων μεταξύ των κατηγοριών.

2.6.6. *Έλεγχος βασισμένος στα συστατικά*

Προς το τέλος της δεκαετίας του '90, η ανάπτυξη βασισμένη στα συστατικά (CB) προέκυψε ως τελευταία προσέγγιση που θα παρήγαγε τη γρήγορη ανάπτυξη λογισμικού με λιγότερους πόρους. Ο έλεγχος σε αυτό το παράδειγμα εισήγαγε νέες προκλήσεις, οι οποίες διακρίνονται μεταξύ τεχνικού είδους και θεωρητικού είδους. Στην τεχνική πλευρά, τα συστατικά πρέπει να είναι αρκετά όσον αφορά γενικά την ανάπτυξη σε διαφορετικές πλατφόρμες και πλαίσια, επομένως ο χρήστης εξαρτημάτων πρέπει να επανελέγξει το συστατικό στο συγκεντρωμένο σύστημα όπου είναι εγκατεστημένο. Αλλά το κρίσιμο πρόβλημα εδώ είναι να αντιμετωπιστεί η έλλειψη πληροφοριών για την ανάλυση και τον έλεγχο των εξωτερικά αναπτυγμένων τμημάτων. Στην πραγματικότητα, ενώ οι διεπαφές συστατικών έχουν περιγραφεί σύμφωνα με συγκεκριμένα συστατικά μοντέλων, αυτά δεν παρέχουν αρκετές πληροφορίες για τον λειτουργικό έλεγχο. Επομένως η έρευνα έχει υποστηρίξει ότι οι σωστές πληροφορίες, ή ακόμα και οι ίδιες περιπτώσεις ελέγχου (όπως στον ενσωματωμένο έλεγχο, *Built-InTesting*), ενσωματώνονται μαζί με το συστατικό για facilitating testing από τον χρήστη, και επίσης ότι η «σύμβαση» που τα συστατικά τηρούν πρέπει να είναι ρητή, για να επιτρέπεται η επαλήθευση.

Ο έλεγχος των συστημάτων βασισμένων στα συστατικά παρατέθηκε επίσης σαν θεμελιώδη πρόκληση στο FOSE2000[20]. Αυτό που απομένει ένα ανοικτό αειθαλές πρόβλημα είναι το θεωρητικό μέρος του CB ελέγχου: πώς μπορεί κανείς να συμπεράνει τις ενδιαφέρουσες ιδιότητες ενός συγκεντρωμένου συστήματος, ξεκινώντας από τα αποτελέσματα της εξέτασης των συστατικών μεμονωμένα; *Τα θεωρητικά θεμέλια του συνθετικού ελέγχου παραμένουν ακόμα και σήμερα μια σημαντική έρευνα πρόκληση.*

2.6.7. Έλεγχος πρωτοκόλλου

Ωθούμενη από την πίεση της διευκόλυνσης της επικοινωνίας, η έρευνα στον έλεγχο πρωτόκολλου έχει προχωρήσει κατά μήκος μιας χωριστής σειράς σε σχέση με τον έλεγχο λογισμικού. Στην πραγματικότητα, χάρις στην ύπαρξη ακριβών προδιαγραφών επιθυμητής συμπεριφοράς, η έρευνα θα μπορούσε από πολύ νωρίς να αναπτύξει προηγμένες μεθόδους και εργαλεία για την προσαρμογή του ελέγχου στις προκαθορισμένες προδιαγραφές. Δεδομένου ότι αυτά τα αποτελέσματα συλλήφθηκαν για έναν τομέα εφαρμογής καθορισμένο με σαφήνεια, δεν ισχύουν εύκολα για τον γενικό έλεγχο λογισμικού. Εντούτοις, το ίδιο αρχικό πρόβλημα με την εξασφάλιση σωστής αλληλεπίδρασης μεταξύ μακρινών συστατικών και υπηρεσιών προκύπτει σήμερα σε μια ευρύτερη κλίμακα για οποιοδήποτε σύγχρονο λογισμικό. Επομένως η έρευνα ελέγχου λογισμικού θα μπορούσε να μάθει από το πρωτόκολλο ελέγχου τη συνήθεια της υιοθέτησης τυποποιημένων επίσημων προδιαγραφών, που είναι η τάση σε σύγχρονες εφαρμογές που είναι προσανατολισμένες προς τις υπηρεσίες. Αντίστροφα, ενώ τα πρώτα πρωτόκολλα ήταν απλά και εύκολα ανιχνεύσιμα, σήμερα η εστίαση μετατοπίζεται σε πιο υψηλά επίπεδα πρωτοκόλλων επικοινωνίας, και ως εκ τούτου το θέμα της πολυπλοκότητας το πιο σύνηθες θέμα ελέγχου λογισμικού, ξεκινά να γίνεται «πιεστικό». Επομένως, ο εννοιολογικός διαχωρισμός μεταξύ του ελέγχου πρωτοκόλλου και των γενικών προβλημάτων ελέγχου λογισμικού σταδιακά εξαφανίζεται.

Τα πρωτόκολλα είναι οι κανόνες που κυβερνούν την επικοινωνία μεταξύ των συστατικών ενός διανεμημένου συστήματος, και αυτά πρέπει να διευκρινιστούν πλήρως προκειμένου να διευκολυνθεί η διαλειτουργικότητα. Ο έλεγχος πρωτοκόλλου στοχεύει στην επαλήθευση της προσαρμογής των εφαρμογών πρωτοκόλλου ενάντια στις προδιαγραφές τους. Τα τελευταία απελευθερώνονται από συγκεκριμένους οργανισμούς, ή από κοινοπραξίες των επιχειρήσεων. Σε ορισμένες περιπτώσεις, εφαρμόζεται επίσης μια τυποποιημένη ακολουθία ελέγχου προσαρμογής.

Η προσαρμογή σε έναν έλεγχο πρωτοκόλλου προσδιορίζεται από την άποψη του ρόλου του πελάτη και του ρόλου του εξυπηρετητή. Μια εφαρμογή μπορεί να απαιτήσει την προσαρμογή σε έναν ή και σε δύο ρόλους. Μια εφαρμογή που προσαρμόζεται στο ρόλο *του πελάτη* πρέπει να τηρείται από τα πρωτόκολλα για την έναρξη μιας σύνδεσης σε ένα άλλο ανοικτό σύστημα, το οποίο πρέπει να δημιουργήσει σωστά τα αιτήματα των υπηρεσιών πρωτοκόλλου και για τις άλλες λειτουργικές μονάδες που ισχυρίζεται πως υποστηρίζει. Μια εφαρμογή που προσαρμόζεται σε έναν ρόλο *εξυπηρετητή* πρέπει να τηρήσει τα πρωτόκολλα για το άνοιγμα μιας σύνδεσης που ζητείται από ένα άλλο ανοικτό σύστημα, όπου πρέπει

να ανταποκριθεί σωστά στα αιτήματα υπηρεσιών πρωτοκόλλου για τις λειτουργικές μονάδες που ισχυρίζεται ότι υποστηρίζει.

2.6.8. *Έλεγχος αξιοπιστίας.*

Ο έλεγχος αξιοπιστίας αναγνωρίζει ότι δεν μπορεί κανείς να ανακαλύπτει και την πιο μικρή αποτυχία συνεχώς, και ως εκ τούτου, χρησιμοποιείται το λειτουργικό σχεδιάγραμμα για να καθοδηγήσει τον έλεγχο, το οποίο προσπαθεί να εξαφανίσει εκείνες τις αποτυχίες που θα φανερώνονταν πιο συχνά: διαισθητικά ο ελεγκτής μιμείται το πώς οι χρήστες θα χρησιμοποιήσουν το σύστημα. Η αξιοπιστία λογισμικού είναι συνήθως βασισμένη στα *πρότυπα αξιοπιστίας*: άλλα είναι τα πρότυπα που πρέπει να χρησιμοποιηθούν, εάν τα ανιχνευμένα ελαττώματα έχουν αποσυρθεί, οπότε σ' αυτή την περίπτωση η αξιοπιστία αυξάνεται και άλλα στην αντίθετη περίπτωση. Η έρευνα στην αξιοπιστία λογισμικού έχει διασταυρωθεί με την έρευνα ελέγχου λογισμικού με πολλούς καρποφόρους τρόπους. Τα πρότυπα για την αξιοπιστία λογισμικού έχουν μελετηθεί ενεργά τις δεκαετίες 80 και 90[21]. Αυτά τα πρότυπα είναι τώρα έτοιμα και μπορούν να εμπλακούν στη διαδικασία ελέγχου παρέχοντας ποσοτικές οδηγίες για το πώς και πόσο να εξετασθεί. Για παράδειγμα, αυτό έγινε από τον Musa στο δικό του έλεγχο αξιοπιστίας λογισμικού (Software-Reliability-Engineered Testing SRET) και υποστηρίζεται επίσης στη αναπτυξιακή διαδικασία του Cleanroom, η οποία ακολουθεί την μέθοδο στατιστικών προσεγγίσεων ελέγχου για να παράγει επικυρωμένα μέτρα αξιοπιστίας[22].

Δυστυχώς, η πρακτική ελέγχου αξιοπιστίας δεν έχει προχωρήσει με την ίδια ταχύτητα όπως αυτή των θεωρητικών προόδων στην αξιοπιστία λογισμικού, πιθανώς επειδή είναι (αντιληπτή ως) μια πολύπλοκη και ακριβή δραστηριότητα, αλλά και για την έμφυτη δυσκολία της αναγνώρισης του απαραίτητου λειτουργικού σχεδιαγράμματος . Μέχρι και σήμερα η απαίτηση για αξιοπιστία αυξάνεται και ως εκ τούτου προκύπτει η ανάγκη για πρακτικές προσεγγίσεις που θα εξετάσουν με συνοχή τη λειτουργική και την extra λειτουργική συμπεριφορά των σύγχρονων συστημάτων λογισμικού, όπως παρουσιάζεται παρακάτω. Για μελλοντικές προκλήσεις στην αξιοπιστία ελέγχου γίνεται αναφορά στον Lyu roadmap[23].

2.6.9. *Στόχος : Έλεγχος βασισμένος στο πρότυπο*

Ένα μεγάλο μέρος της έρευνας επικεντρώνεται στις μέρες μας στον έλεγχο βασισμένο στο πρότυπο (modelbased). Η κύρια ιδέα είναι να χρησιμοποιηθούν πρότυπα/μοντέλα που καθορίζονται στην κατασκευή του λογισμικού για να οδηγήσουν τη διαδικασία ελέγχου, και συγκεκριμένα να δημιουργήσουν αυτόματα περιπτώσεις ελέγχου. Η πραγματιστική προσέγγιση που απαιτείται για

ερευνητικούς ελέγχους είναι αυτή που ακολουθεί την τρέχουσα τάση στη μοντελοποίηση: ανάλογα με το ποια είναι η γλώσσα που χρησιμοποιείται, π.χ. UML ή η Z, θα πρέπει να προσαρμοστεί μια τεχνική ελέγχου όσο πιο αποτελεσματικά γίνεται [24]. Αλλά το όνειρο από την άποψη του ελεγκτή θα ήταν να ανατραπεί αυτή η προσέγγιση σε σχέση με το τι έρχεται πρώτο και το τι έρχεται μετά: αντί να πάρει κανείς ένα έτοιμο μοντέλο και να δει με ποιον τρόπο μπορεί να το εκμεταλλευθεί καλύτερα για τον έλεγχο, ας εξετάσει καλύτερα πώς θα πρέπει να φτιάξει το μοντέλο στην ιδανική περίπτωση έτσι ώστε να μπορεί να ελεγχθεί αποτελεσματικά το λογισμικό. Δεν θα ήταν ωραίο αν οι προγραμματιστές- γνώριζαν καλά τη σημασία και τη δυσκολία του ελέγχου και δημιουργούσαν κατάλληλα μοντέλα για τον έλεγχο; Αυτό είναι το κίνητρο εδώ να ανατραπεί ο όρος «έλεγχος βασισμένος στο μοντέλο» και να μετατραπεί στον όρο «μοντελοποίηση ελέγχου» .

Ασφαλώς, αυτό είναι μόνο ένα νέος όρος για μια παλιά ιδέα, καθώς πράγματι μπορούν να βρεθούν ήδη αρκετές ερευνητικές κατευθύνσεις που άλλες περισσότερο ή άλλες λιγότερο κατευθύνονται προς το όνειρο. Στη μία πλευρά, η έννοια αυτή της μοντελοποίησης που βασίζεται στον έλεγχο είναι στενά συνδεδεμένη με, έναν παράγοντα, της παλιάς ιδέας του "Σχεδιασμού ικανότητας ελέγχου", που ασχολείται κυρίως με τη σχεδίαση λογισμικού για ενίσχυση του ελέγχου (των συντελεστών παραγωγής)[25] και της παρατήρησης (των αποτελεσμάτων)[26]. Αλλά επίσης σχετικές μπορεί να φαίνονται οι πρώην προσεγγίσεις σε ελέγχους με βάση τους ισχυρισμούς και τα πιο πρόσφατα περιστατικά που βασίζονται σε συμβάσεις. Συγκεκριμένα οι ισχυρισμοί έχουν αρχικά αναγνωριστεί ως ένα χρήσιμο εργαλείο για την ενίσχυση του ελέγχου, δεδομένου ότι μπορούν να επαληθεύουν κατά το χρόνο εκτέλεσης την εσωτερική κατάσταση ενός προγράμματος.

Μια σύμβαση καθιερώνει μια "νομική" συμφωνία μεταξύ δύο αλληλεπιδρώντων μερών, το οποίο εκφράζεται μέσα από τρεις διαφορετικούς τύπους ισχυρισμών: προϋποθέσεις, post-conditions και σταθερές. Το βήμα προς τη χρήση αυτών των συμβάσεων ως σημείο αναφοράς για έλεγχο είναι σύντομο.

2.6.10. Πρόκληση: Έλεγχος με βάση το πρότυπο

Οι συχνές αναφερθείσες τάσεις στο παρόν έγγραφο της αύξησης των επιπέδων πολυπλοκότητας και των αναγκών για υψηλή ποιότητα οδηγούν στην αύξηση του κόστους ελέγχου, στο σημείο όπου οι παραδοσιακές πρακτικές ελέγχου γίνονται οικονομικά δαπανηρές, αλλά, ευτυχώς, από την άλλη πλευρά, η αυξανόμενη χρήση μοντέλων στην ανάπτυξη λογισμικού καταργεί προοπτικές αφαίρεσης του κύριου

εμπόδιου για την θέσπιση μοντέλου βασισμένο στον έλεγχο, η οποία είναι (επίσημη) η μοντελοποίηση δεξιοτήτων.

Ο έλεγχος με βάση το μοντέλο είναι στην πραγματικότητα ένα είδος ταινίας «πίσω στο μέλλον» για τον έλεγχο λογισμικού. Πράγματι, η ιδέα των δοκιμών modelbased είναι εδώ και δεκαετίες (η Moore άρχισε την έρευνα στη γενιά του ελέγχου, το 1956!)[27], αλλά τα τελευταία χρόνια έχουμε δει συντριπτικό ενδιαφέρον όσον αφορά την εφαρμογή του σε πραγματικές εφαρμογές (για μια εισαγωγή στις διαφορετικές προσεγγίσεις και εργαλεία σε modelbased, έλεγχο). Ωστόσο, η βιομηχανική θέσπιση που βασίζεται στο μοντέλο ελέγχου παραμένει χαμηλή και τα σημάδια των επιτευγμάτων της αναμενόμενης έρευνας είναι αδύναμα. Επομένως, πέρα από τις θεωρητικές προκλήσεις, οι ερευνητές επικεντρώνονται σήμερα στο πώς να νικηθούν οι φραγμοί στην ευρεία υιοθέτηση. Υπάρχουν σημαντικά θέματα τεχνικής και θέματα που σχετίζονται με την διαδικασία που είναι εν αναμονή. Ένα ευρέως αναγνωρισμένο ζήτημα είναι πώς μπορούν να συνδυαστούν διάφορα στυλ της μοντελοποίησης (όπως το σύστημα που βασίζεται στην μετάβαση, το pre/post που βασίζεται στην κατάσταση, και το σύστημα που βασίζεται στο σενάριο). Για παράδειγμα, θα πρέπει να βρεθούν αποτελεσματικοί τρόποι για να συνθέσουν προσεγγίσεις βασισμένες στις καταστάσεις και στο σενάριο.

Στην περίπτωση του ελέγχου βάσει μοντέλου τα σενάρια ελέγχου εξάγονται από ένα ολόκληρο ή ένα κομμάτι ενός μοντέλου που περιγράφει τις λειτουργικές απαιτήσεις του συστήματος που ελέγχεται. Συνήθως το μοντέλο είναι η αφαιρετική, μερική συμπεριφορά του συστήματος.

Στη Microsoft, όπου ο έλεγχος βάσει των μοντέλων υποστηρίζεται χρόνια τώρα, αλλά με περιορισμένη παρακολούθηση, επιδιώκεται μια multi-paradigmatic προσέγγιση[28] για να προτιμηθεί μια ευρύτερη υιοθέτηση. Η ιδέα είναι ότι τα μοντέλα που απορρέουν από διάφορα παραδείγματα και εκφράζονται σε οποιαδήποτε μορφή μπορεί να χρησιμοποιηθούν άψογα μέσα σε ένα ολοκληρωμένο περιβάλλον. Το μάθημα, στην πραγματικότητα είναι ότι αναγκάζοντας τους χρήστες να χρησιμοποιούν μια νέα «σημείωση» (notation) δεν λειτουργεί, αντίθετα πρέπει ο πυρήνας μιας προσέγγισης που βασίζεται στον έλεγχο προτύπου να είναι άγνωστος και να επιτρέπει στους προγραμματιστές να χρησιμοποιούν τις υπάρχουσες σημειώσεις και τα υπάρχοντα περιβάλλοντα προγραμματισμού. Χρειάζονται επίσης τρόποι για να συνδυαστούν κριτήρια με βάση το μοντέλο με άλλες προσεγγίσεις, για παράδειγμα μια πολλά υποσχόμενη ιδέα είναι να χρησιμοποιηθεί ο έλεγχος σε προσομοιώσεις για να βελτιστοποιήσουν τη σουίτα δοκιμών και να δώσουν ώθηση στον έλεγχο.

Τα ζητήματα σχετικά με τη διαδικασία αφορούν την ανάγκη να ενσωματώσουν την πρακτική έλεγχου βασισμένη στο πρότυπο στις τρέχουσες διαδικασίες λογισμικού: ίσως τα κρίσιμα ζητήματα εδώ είναι οι δύο συσχετιζόμενες ανάγκες για τη διαχείριση ελέγχου της παραγωγής των μοντέλων ελέγχου όσο πιο επίσημα γίνεται, ενώ διατηρείται η δυνατότητα να παραχθούν εκτελέσιμοι έλεγχοι από τη μια πλευρά και η τήρηση της ανιχνευσιμότητας από απαιτήσεις σε ελέγχους καθ' όλη τη διάρκεια της αναπτυξιακής διαδικασίας, από την άλλη. Στο τέλος επίσης χρειάζονται εργαλεία μεγάλης βιομηχανικής-δύναμης για τη δημιουργία και τη διαδραστική διαμόρφωση, τα οποία μπορούν να βοηθήσουν στο να μειώσουν την ανεπαρκή εκπαίδευση των τωρινών ελεγκτών (ή ίσως και τις υπερβολικές απαιτήσεις πείρας των προτεινόμενων τεχνικών). Μια πρόσθετη περίπτωση ελέγχου βασισμένου στο πρότυπο είναι ο *προσαρμοστικός έλεγχος*, δηλ., *ο έλεγχος για το εάν το σύστημα υπό έλεγχος συμμορφώνεται με τις προκαθορισμένες προδιαγραφές του.*

Αρχίζοντας από τη δεκαετία του '70 έχουν προταθεί πολλοί αλγόριθμοι, μια πρόσφατη εκτενής επισκόπηση των τωρινών ανοικτών προκλήσεων δίνεται στο διδακτικό έγγραφο του Broy[29] που αναφέρει τον έλεγχο βασισμένο στο πρότυπο για τα αντιδραστικά συστήματα. Τα αποτελέσματα που έχουν επιτευχτεί μέχρι τώρα είναι εντυπωσιακά για θεωρητικούς λόγους, αλλά πολλές από τις προτεινόμενες μεθόδους ισχύουν μετά βίας στα ρεαλιστικά συστήματα, παρόλα αυτά έχουν παραχθεί διάφορα εργαλεία και μερικά από αυτά εφαρμόζονται σε εξειδικευμένες περιοχές. Μια καλή επισκόπηση των εργαλείων για τον έλεγχο βασισμένο στο πρότυπο που στηρίζεται σε μια σωστή θεωρία παρέχεται από τον Belinfante, ο οποίος δίνει έμφαση στην ανάγκη της βελτίωσης και διευκόλυνσης της εφαρμογής της θεωρίας.

Αυτή η πρόκληση αναφέρεται στο ΠΩΣ επιλέγονται ποιές εκτελέσεις ελέγχου να παρατηρηθούν, και εν μέρει το ΠΟΣΟ να παρατηρηθούν.

2.6.11. Πρόκληση: Έλεγχος μη βασισμένος στο πρότυπο

Παράλληλα με τον έλεγχο που βασίζεται στο πρότυπο, διάφορες προσπάθειες έχουν αφιερωθεί προς τις νέες μορφές ελέγχου που εκτείνονται κυρίως προς την ανάλυση των εκτελέσεων του προγράμματος, παρά από ένα πρότυπο a priori. Αντί της υιοθέτησης ενός προτύπου, μπορεί κανείς να επινοήσει από αυτό ένα σχέδιο ελέγχου, και ως εκ τούτου να συγκρίνει τα αποτελέσματα του ελέγχου σε σχέση με το πρότυπο, αυτές οι διαφορετικές προσεγγίσεις συλλέγουν πληροφορίες από την

εκτέλεση του προγράμματος, είτε μετά από ενεργή ζήτηση από κάποια εκτέλεση, είτε παθητικά κατά τη λειτουργία, και προσπαθούν να συνθέσουν από αυτές μερικές σχετικές ιδιότητες των στοιχείων ή των συμπεριφορών. Μπορούν να υπάρξουν περιπτώσεις στις οποίες τα πρότυπα απλά δεν υπάρχουν ή δεν είναι προσιτά και άλλες περιπτώσεις στις οποίες η γενική αρχιτεκτονική συστημάτων δεν είναι αποφασισμένη a priori, αλλά δημιουργείται και εξελίσσεται δυναμικά κατά τη διάρκεια της ζωής ενός συστήματος ή, ένα πρότυπο είναι αρχικά δημιουργημένο, αλλά κατά τη διάρκεια της ανάπτυξης γίνεται σταδιακά λιγότερο χρήσιμο μιας και η ανταπόκριση του με την εφαρμογή δεν επιβάλλεται και χάνεται.

Ως εκ τούτου, συμμετρικά με τον έλεγχο βασισμένο στο πρότυπο, ισχύει ότι (ρητά ή σιωπηρά) ένα πρότυπο παράγεται a posteriori μέσω του ελέγχου, ο οποίος αναφέρεται ως έλεγχος μη βασισμένος στο πρότυπο. Υπό αυτόν τον όρο αναφέρονται σε όλες τις διαφορετικές προσεγγίσεις που με την έννοια του ελέγχου, κατασκευάζουν αντίστροφα ένα πρότυπο, υπό μορφή σταθεράς πέρα από τις μεταβλητές του προγράμματος, ή υπό μορφή ενός statemachine, ή ενός επονομαζόμενου συστήματος μετάβασης (Labelled Transition System), ή ενός διαγράμματος ακολουθίας, και ούτω καθεξής, και έπειτα ελέγχουν ένα τέτοιο πρότυπο για να ανιχνεύσουν εάν το πρόγραμμα συμπεριφέρεται σωστά ή όχι.

Ο έλεγχος μη βασισμένος στο πρότυπο μπορεί να στηριχθεί στις μεγάλες προόδους της δυναμικής ανάλυσης προγράμματος, η οποία είναι μια πολύ ενεργή ερευνητική αρχή σήμερα, όπως θεωρείται από τον Canfora και τον Di Penta . Πρέπει να είναι κανείς σε θέση να συμπεράνει τις ιδιότητες του συστήματος σε ένα περιορισμένο σύνολο παρατηρηθέντων αποτελεσμάτων, ή ακόμα και μερών «ιχνών», δεδομένου ότι μπορούν να παρατηρηθούν τα συστατικά που διαμορφώνουν το σύστημα. Σε μια πρόσφατη εργασία, οι Mariani και Pezze προτείνουν την τεχνική BCT για να παραχθούν τα πρότυπα συμπεριφοράς για τα ελεγχόμενα τμήματα COTS. Στην προσέγγισή τους τα συμπεριφοριστικά πρότυπα αποτελούνται και από τα δύο I/O πρότυπα, που λαμβάνονται με τη βοήθεια του γνωστού ανιχνευτή Daikon, και τα πρότυπα αλληλεπίδρασης, υπό τη μορφή Finite State. Κατά έναν ενδιαφέρον τρόπο, αυτά τα παραγόμενα πρότυπα μπορούν κατόπιν να χρησιμοποιηθούν για τον έλεγχο βασισμένο στο πρότυπο εάν και όταν αντικαθίστανται τα συστατικά από νέα. Μια σχετική πρόκληση είναι να διατηρηθούν τα δυναμικά παραγόμενα πρότυπα ενημερωμένα: ανάλογα με το τύπο βελτίωσης στο σύστημα, επίσης όπως παρατηρούν και οι Mariani και Pezze, περιγράφοντας μερικές πιθανές στρατηγικές το πρότυπο μπορεί να χρειάζεται να καθοριστεί.

Αυτή η πρόκληση αναφέρεται επίσης στο ΠΩΣ και στο ΠΟΣΟ πολύ παρατηρούνται οι εκτελέσεις ελέγχου.

2.6.12. Έλεγχος Βασιζόμενος στις Απαιτήσεις

Εάν από τη μία θεωρηθεί ότι οι απαιτήσεις αποτελούν ένα σύνολο ιδεών το οποίο ορίζει την ποιότητα ενός συγκεκριμένου προϊόντος και από την άλλη ότι ο έλεγχος αποτελεί τη διαδικασία ανάπτυξης και εκτίμησης της ποιότητας του προϊόντος, γίνεται φανερή η άμεση συσχέτιση που υπάρχει ανάμεσα τους.

Έχει διατυπωθεί η άποψη ότι χωρίς να είναι δηλωμένες οι απαιτήσεις δεν είναι δυνατός ο έλεγχος και ότι ένα προϊόν λογισμικού πρέπει να ικανοποιεί τις απαιτήσεις που έχουν τεθεί. Εάν λοιπόν είναι αρκετά σημαντικό να ικανοποιείται μια απαίτηση και είναι δουλειά του υπεύθυνου για τον έλεγχο να εκτιμήσει το προϊόν σε σχέση με την απαίτηση αυτή, τότε η παραπάνω άποψη είναι βασικά ορθή.

Όμως, η βαθύτερη αλήθεια είναι ότι οι απαιτήσεις οι οποίες έχουν καταγραφεί δεν είναι οι μόνες απαιτήσεις. Εξαιτίας της ατέλειας και της ασάφειας, ο έλεγχος δεν πρέπει να θεωρείται απλώς ως μία διαδικασία εκτίμησης. Είναι επιπλέον και μία διαδικασία διερεύνησης της σημασίας και των συνεπειών των απαιτήσεων. Γι' αυτό, ο έλεγχος όχι μόνο είναι δυνατός χωρίς την ύπαρξη καταγεγραμμένων απαιτήσεων, αλλά και ιδιαίτερα χρήσιμος σε μια τέτοια περίπτωση. Μία καλή ομάδα ελέγχου θα πρέπει να βρίσκεται σε επιφυλακή για την αναγνώριση κενών στις απαιτήσεις και να προσπαθεί να τα αντιμετωπίσει στο βαθμό που δικαιολογείται από τους κινδύνους της κατάστασης. Η ιδέα ότι το προϊόν λογισμικού πρέπει να ικανοποιεί τις απαιτήσεις που έχουν τεθεί και ότι αυτό χαρακτηρίζει την ποιότητά του είναι αλήθεια, εφόσον κάθε απαίτηση είναι ορθή και επιτεύξιμη. Αλλά αυτό εξαρτάται από την ύπαρξη ενός καθαρού και ολοκληρωμένου συνόλου απαιτήσεων. Διαφορετικά δεν ανταποκρίνεται στην πραγματικότητα η ιδέα της ποιότητας που προαναφέρθηκε. Πρακτικά, αυτό που ισχύει είναι ότι ενώ η ποιότητα καθορίζεται από τις απαιτήσεις, δεν καθορίζεται απλά από το άθροισμα των "ικανοποιημένων" απαιτήσεων που έχουν καταγραφεί. Υπάρχουν πολλοί τρόποι για να ικανοποιήσεις ή να παραβιάσεις τις απαιτήσεις. Οι απαιτήσεις δεν είναι όλες ίσες σε σημασία και συχνά έρχονται ακόμα και σε σύγκρουση μεταξύ τους.

2.6.13. Πρόκληση: Χρησμοί ελέγχου (Oracle)

Πολύ σχετικό με τον προγραμματισμό ελέγχου, και συγκεκριμένα με το πρόβλημα για το πώς να παράγει τις περιπτώσεις ελέγχου, είναι το ζήτημα της απόφασης εάν ένα αποτέλεσμα ελέγχου είναι αποδεκτό ή όχι. Αυτό αντιστοιχεί στον αποκαλούμενο «χρησμό», μια ιδανικά μαγική μέθοδος που παρέχει τα αναμενόμενα αποτελέσματα για κάθε δοθέν σενάριο ελέγχου πιο ρεαλιστικά, μια

μηχανή που μπορεί να εκπέμψει μια απόφαση επιτυχίας /αποτυχίας όσον αφορά τα αποτελέσματα ελέγχου.

Αν και είναι προφανές ότι μια εκτέλεση ελέγχου την οποία δεν είναι κάποιος ικανός να διακρίνει αν είναι επιτυχής ή έχει αποτύχει είναι ένας άχρηστος έλεγχος, και αν και η κριτική διάθεση αυτού του προβλήματος έχει από νωρίς εμφανισθεί στην βιβλιογραφία του ελέγχου, στο πρόβλημα χρησμού έχει δοθεί ελάχιστη προσοχή από την έρευνα και στην πράξη υπάρχουν λίγες ακόμα εναλλακτικές λύσεις. Αλλά δεδομένης της παρούσας κατάστασης που είναι ήδη σήμερα μη ικανοποιητική, με την αυξανόμενη πολυπλοκότητα και την κριτική διάθεση του λογισμικού οι εφαρμογές προορίζονται να αποτελέσουν ένα αξιόπιστο εμπόδιο που βασίζεται στην αυτοματοποίηση ελέγχου (στην πραγματικότητα, η πρόκληση χρησμών ελέγχου επικαλύπτει επίσης τη διαδρομή προς την αυτοματοποίηση ελέγχου). Πράγματι, η ακρίβεια και η αποδοτικότητα των χρησμών έχουν πολλές επιπτώσεις στο κόστος ελέγχου/αποτελεσματικότητας: δεν είναι επιθυμητό οι αποτυχίες ελέγχου να περνούν και να μην ανιχνεύονται, αλλά από την άλλη πλευρά δεν είναι επιθυμητό ούτε να ενημερώνονται οι χρήστες για τα λάθη, τα οποία σπαταλούν σημαντικούς πόρους τους συστήματος. Πρέπει να βρεθούν πιο αποδοτικές μέθοδοι για την κατανόηση και αυτοματοποίηση των χρησμών, φιλτράροντας τη πληροφορία που είναι διαθέσιμη.

Αυτή η πρόκληση αναφέρεται στην ερώτηση ΓΙΑΤΙ, από την άποψη του τι εξετάζεται.

2.7. Στόχος: 100% Αυτόματος έλεγχος

Η εκτεταμένη αυτοματοποίηση είναι ένας από τους τρόπους για να επιτευχθεί η ποιοτική ανάλυση και ο έλεγχος σε συνάρτηση με την αυξανόμενη ποσότητα και την πολυπλοκότητα του λογισμικού. Η έρευνα τεχνολογίας λογισμικού δίνει μεγάλη έμφαση στην αυτοματοποίηση της παραγωγής του λογισμικού, με έναν όγκο παραγωγής σύγχρονων εργαλείων ανάπτυξης σε ακόμα μεγαλύτερες και πιο σύνθετες ποσότητες κώδικα και με λιγότερη προσπάθεια. Η άλλη πλευρά του νομίσματος είναι ο μεγάλος κίνδυνος ότι οι μέθοδοι που αξιολογούν την ποιότητα του παραχθέντος λογισμικού με αυτόν τον τρόπο, και πιο συγκεκριμένα οι εξεταστικές μέθοδοι, δεν μπορούν να συμβαδίσουν με τέτοιες μεθόδους κατασκευής λογισμικού.

Ένα μεγάλο μέρος των τρεχουσών ερευνών ελέγχου στοχεύει στη βελτίωση του βαθμού εφικτής αυτοματοποίησης, είτε με την ανάπτυξη προηγμένων τεχνικών για την γενίκευση των εισαγωγών ελέγχου (αυτή η πρόκληση επεκτείνεται παρακάτω), ή, πέρα από την παραγωγή ελέγχου, με την εύρεση καινοτόμων διαδικασιών υποστήριξης με σκοπό την αυτοματοποίηση της διαδικασίας ελέγχου. Το όνειρο θα

ήταν ένα ισχυρό ενσωματωμένο περιβάλλον δοκιμών το όποιο ολοκληρώνεται και επεκτείνεται από μόνο του, ως κομμάτι του λογισμικού , ενώ μπορεί αυτόματα να φροντίσει την πιθανή ενορχήστρωση του και να γενικεύσει ή να ανακτήσει τον απαιτούμενο κώδικα κλιμάκωσης (οδηγοί, στέλεχος, προσομοιωτές), παράγοντας τις περισσότερες κατάλληλες περιπτώσεις ελέγχου, ενώ τέλος τις εκτελεί και εκδίδει μια αναφορά ελέγχου. Αυτή η ιδέα, αν και ουτοπική, έχει προσελκύσει οπαδούς, για παράδειγμα την πρόσφατη πρωτοβουλία DARPA για τον διαρκή έλεγχο (που αναφέρεται επίσης στο FOSE2000 Perpetual Test) και πιο πρόσφατα στη συνεχή προσέγγιση ελέγχου Saff και Ernst, η οποία στοχεύει ακριβώς στο να εκτελούνται έλεγχοι στο υπόβαθρο του μηχανήματος του προγραμματιστή ενώ αυτός προγραμματίζει.

Αρκετά ελπιδοφόρα βήματα έχουν γίνει πρόσφατα προς την πραγματοποίηση αυτού του ονείρου για τον έλεγχο μονάδων, η οποία αναγνωρίζεται ευρέως ως ουσιαστική φάση για την εξασφάλιση της ποιότητας λογισμικού, επειδή με τη διερεύνηση μεμονωμένων μονάδων ξεχωριστά μπορεί από νωρίς να ανιχνεύσει επιφανειακά αλλά ακόμα και βαθειά-κρυμμένα ελαττώματα που θα βρίσκονταν μετά βίας κατά τον έλεγχο συστημάτων. Δυστυχώς, ο έλεγχος μονάδων συχνά εκτελείται επιφανειακά ή παραλείπεται συνολικά επειδή έχει μεγάλο κόστος. Χρειάζονται προσεγγίσεις για να γίνει αυτό εφικτό στην διαδικασία της βιομηχανικής ανάπτυξης. Ένα σημαντικό συστατικό του ελέγχου μονάδας με υψηλό κόστος είναι η τεράστια ποσότητα πρόσθετης κωδικοποίησης που είναι απαραίτητη για την προσομοίωση του περιβάλλοντος όπου θα εκτελεστεί ο έλεγχος μονάδας, και για την εκτέλεση του αναγκαίου λειτουργικού ελέγχου των αποτελεσμάτων των μονάδων. Για να ικανοποιηθούν αυτοί οι στόχοι, ιδιαίτερα - επιτυχή για την ανάπτυξη ήταν τα πλαίσια που ανήκουν στην οικογένεια XUnit.

Μεταξύ αυτών, το πιο επιτυχές είναι το JUnit, που επιτρέπει την αυτοματοποίηση της κωδικοποίησης των περιπτώσεων ελέγχου της Java και της διαχείρισης τους και έχει ευνοήσει τη διάδοση της ήδη αναφερθείσας ανάπτυξης του ελέγχου test driven. Εντούτοις τέτοια πλαίσια δεν βοηθούν στην παραγωγή ελέγχου και την προσομοίωση του περιβάλλοντος. Θα ήταν επιθυμητό να ωθηθεί η αυτοματοποίηση περαιτέρω, όπως παραδείγματος χάριν στην κατευθυνόμενη αυτοματοποιημένη τυχαία προσέγγιση ελέγχου (DART Directed Automated Random Tests), η οποία αυτοματοποιεί πλήρως τον έλεγχο μονάδας από: την αυτοματοποιημένη εξαγωγή διεπαφών μιας στατικής ανάλυσης πηγαίου-κώδικα την αυτοματοποιημένη παραγωγή ενός τυχαίου οδηγού ελέγχου για την διεπαφή και την δυναμική ανάλυση της συμπεριφοράς του προγράμματος κατά τη διάρκεια εκτέλεσης τυχαίων σεναρίων ελέγχου, που στοχεύουν στο να παράγουν αυτόματες νέες εισαγωγές ελέγχου που μπορούν να κατευθύνουν την εκτέλεση των εναλλακτικών πορειών του προγράμματος.

Ένα άλλο παράδειγμα παρέχεται από την έννοια της «αναταραχής λογισμικού» (Software agitation), μια αυτόματη τεχνική μονάδα ελέγχου που υποστηρίζεται από το εμπορικό εργαλείο Agitator, όπου συνδυάζει τις διαφορετικές αναλύσεις, όπως η συμβολική εκτέλεση, η επίλυση περιορισμού, και η τυχαία κατευθυνόμενη παραγωγή εισαγωγής για την παραγωγή των δεδομένων εισόδου, μαζί με το σύστημα Daikon. Μια ακόμα προσέγγιση αποτελείται από τον παραμετροποιημένο έλεγχο μονάδας της Microsoft (PUT), δηλ., κωδικοποιημένες μονάδες ελέγχου που δεν καθορίζονται (όπως συμβαίνει με αυτές που προγραμματίζονται στα πλαίσια του XUnit), αλλά εξαρτώνται από μερικές παραμέτρους εισαγωγής.

Τα τρία αναφερθέντα παραδείγματα δεν είναι βεβαίως τα μοναδικά από το αρκετά ενεργό και καρποφόρο στάδιο που η αυτοματοποίηση ελέγχου κατέχει αυτήν την περίοδο. Η κοινή ελλοχεύουσα τάση που προκύπτει είναι η προσπάθεια να συνδυάσει και να εφαρμόσει αποτελεσματικά μηχανισμούς που προέρχονται από διαφορετικούς τύπους ανάλυσης, και αυτό, μαζί με την εκθετική αύξηση των διαθέσιμων υπολογιστικών πόρων, θα μπορούσε να είναι η κατεύθυνση μιας πραγματικής επιτυχίας προς το όνειρο αυτοματοποίησης ελέγχου κατά 100%.

2.7.1. Πρόκληση: Παραγωγή εισόδων ελέγχου

Η έρευνα στην αυτόματη παραγωγή των εισόδων ελέγχου απασχολούσε πάντα σε μεγάλο βαθμό και αυτήν την περίοδο είναι υπό έρευνα τόσες πολλές προηγμένες τεχνολογίες που ακόμη και αφιερώνοντας ολόκληρο το έγγραφο μόνο σε αυτό το θέμα δεν θα έφτανε για να τις καλύψει επαρκώς. Αυτό που τρομάζει είναι ότι μέχρι σήμερα όλη η προσπάθεια έχει παράγει περιορισμένο αντίκτυπο στη βιομηχανία, όπου η δραστηριότητα παραγωγής ελέγχου παραμένει κατά ένα μεγάλο μέρος χειροκίνητη[30]. Αλλά τελικά ο συνδυασμός θεωρητικής προόδου στις ελλοχεύουσες τεχνολογίες, όπως η συμβολική εκτέλεση, ο έλεγχος προτύπου, η παρουσίαση αποδείξεων θεωρήματος, στατικές και δυναμικές αναλύσεις, με τις προόδους τεχνολογίας στη διαμόρφωση προτύπων βιομηχανικής δύναμης και με διαθέσιμη υπολογιστική δύναμη φαίνεται πως φέρνουν πιο κοντά αυτό το αντικείμενο και έχουν αναζωογονήσει την πίστη των ερευνητών.

Τα πιο ελπιδοφόρα αποτελέσματα ακούγεται πως προέρχονται από τρεις κατευθύνσεις, και ειδικά από την αμοιβαία σύγκλισή τους: η ήδη ευρέως συζητημένη προσέγγιση βασισμένη στο πρότυπο, οι «έξυπνοι» τρόποι για να εφαρμοστεί η τυχαία παραγωγή, και μια πλούσια ποικιλία των τεχνικών βασισμένων στην αναζήτηση που χρησιμοποιούνται για την παραγωγή ελέγχου άσπρου και μαύρου κουτιού.

Σχετικά με την παραγωγή ελέγχου βασισμένου στο πρότυπο, η έρευνα έχει μεγάλες προσδοκίες από αυτήν την κατεύθυνση δίνοντας έμφαση στη χρησιμοποίηση (των επίσημων) προτύπων που οδηγούν τον έλεγχο που προέρχεται κυρίως από την αυτοματοποιημένη παραγωγή περιπτώσεων ελέγχου. Πολλά από τα υπάρχοντα εργαλεία είναι βασισμένα στην κατάσταση και δεν ασχολούνται με τα δεδομένα εισόδου.

Συγκριμένα χρονολογώντας πίσω στη μέση της δεκαετίας του '70, η συμβολική εκτέλεση μπορεί να θεωρηθεί η πιο παραδοσιακή προσέγγιση στην αυτοματοποιημένη παραγωγή στοιχείων ελέγχου. Μια τέτοια προσέγγιση, η οποία είχε τεθεί σε παύση για κάποιο διάστημα, λόγω πολλών ελλοχευουσών δυσκολιών, αναζωογονείται σήμερα από την αναβίωση των strongly-typed γλωσσών, ενώ ο Lee και οι συνεργάτες του ερευνούν τις πιο ελπιδοφόρες εξελίξεις.

Όσον αφορά την τυχαία παραγωγή ελέγχου, αυτό συνήθιζε να θεωρείται μια ρηχή προσέγγιση, σε σχέση με τις συστηματικές τεχνικές, που κρίθηκαν να είναι πιο περιεκτικές και ικανές στο να βρουν τις σημαντικές περιπτώσεις αιχμής (corner cases) που θα ήταν πιθανό να αγνοηθούν από τις τυχαίες τεχνικές. Εντούτοις, προηγούμενες μελέτες σύγκριναν κυρίως τις εφαρμογές τυχαίου ελέγχου με τις περίπλοκες εφαρμογές των συστηματικών τεχνικών. Σήμερα, διάφοροι ερευνητές προτείνουν έξυπνες εφαρμογές τυχαίου ελέγχου που φαίνεται να ξεπερνούν τη συστηματική παραγωγή ελέγχου. Η ελλοχεύουσα ιδέα τέτοιων προσεγγίσεων είναι ότι η τυχαία παραγωγή βελτιώνεται δυναμικά, με την εκμετάλλευση της ανατροφοδότησης των πληροφοριών που συλλέγονται καθώς εκτελούνται οι έλεγχοι. Για παράδειγμα, ο Sen και οι συνεργάτες του έχουν χτίσει πάνω από την αναφερμένη προσέγγιση DART, μια έννοια του «concolic ελέγχου», η οποία είναι ο συνδυασμός συμπαγή (τυχαίου) ελέγχου με συμβολική εκτέλεση. Οι συμπαγής και συμβολικές εκτελέσεις εκτελούνται παράλληλα και «βοηθούν» η μια την άλλην. Αντ' αυτού, ο Pacheco και οι συνεργάτες του επιλέγουν τυχαία μια περίπτωση ελέγχου, την εκτελούν και την ελέγχουν σε σχέση με ένα σύνολο καταστάσεων και φίλτρων.

Ύστερα η πιο ελπιδοφόρα κατεύθυνση είναι να βρεθούν ικανοί τρόποι να συνδυαστούν οι αντίστοιχες δυνάμεις του συστηματικού (που βασίζεται στο πρότυπο) και τυχαίου ελέγχου. Τέλος, σχετικά με την παραγωγή ελέγχου βασισμένου στην αναζήτηση, αυτό αποτελείται από την εξερεύνηση της κατάστασης των λύσεων (τις επιδιωκόμενες περιπτώσεις ελέγχου) για ένα επιλεγμένο κριτήριο ελέγχου, με τη χρησιμοποίηση των μεταευριστικών τεχνικών που κατευθύνουν την αναζήτηση προς τους ενδεχομένως περισσότερο ελπιδοφόρους τομείς του χώρου εισαγωγής. Το ελκυστικό χαρακτηριστικό

γνώρισμα είναι ότι αυτή η προσέγγιση εμφανίζεται να ισχύει αρκετά αποδοτικά σε μια απεριόριστη σειρά προβλημάτων σε μια πρόσφατη έρευνα που παρέχεται από τον McMinn . Η παραγωγή δεδομένων ελέγχου που βασίζεται στην αναζήτηση είναι μόνο μια πιθανή εφαρμογή της μηχανικής ελέγχου που βασίζεται στην αναζήτηση.

Αυτή η πρόκληση εξετάζει το ΠΩΣ παράγονται οι παρατηρήσεις.

2.7.2. Πρόκληση: Προσεγγίσεις εξειδικευμένου ελέγχου

Ο έλεγχος μπορεί επίσης να ωφεληθεί από τον περιορισμό του πεδίου εφαρμογής στις ανάγκες ενός συγκεκριμένου πεδίου. Η έρευνα πρέπει να εξετάσει το πώς η γνώση πεδίου μπορεί να βελτιώσει την διαδικασία ελέγχου. Πρέπει να επεκταθούν οι προσεγγίσεις που βασίζονται στο πεδίο στο στάδιο ελέγχου, και πιο συγκεκριμένα στην εύρεση των μεθόδων που βασίζονται σε συγκεκριμένα πεδία και εργαλεία για να ωθήσουν την αυτοματοποίηση ελέγχου. Ο έλεγχος που βασίζεται στο πεδίο θα μπορούσε να χρησιμοποιήσει εξειδικευμένα είδη προσεγγίσεων, διαδικασίες και εργαλεία.

Οι τεχνικές ελέγχου έχουν ερευνηθεί για συγκεκριμένα πεδία, για παράδειγμα για τις βάσεις δεδομένων, για τη δυνατότητα χρησιμοποίησης GUI, για τις εφαρμογές Ιστού, για τα συστήματα τηλεπικοινωνιών αλλά υπάρχει λίγη ενασχόληση σχετικά με την ανάπτυξη των μεθοδολογιών για την εκμετάλλευση της γνώσης του πεδίου. Μια ενδιαφέρουσα πρωτοποριακή εργασία οφείλεται στον Reyes και στον Richardson, που ανέπτυξαν από νωρίς ένα πλαίσιο, αποκαλούμενο Siddhartha, για την ανάπτυξη οδηγών ελέγχου σε συγκεκριμένο πεδίο. Το Siddhartha εφάρμοσε μια μέθοδο βασισμένη στο παράδειγμα, επαναληπτική για την ανάπτυξη μεταφραστών σε συγκεκριμένο πεδίο από τις προδιαγραφές ελέγχου σε έναν οδηγό βασισμένο στο πεδίο. Πιο πρόσφατα, η Sinha και ο Smidts ανέπτυξαν ι την πιο ενδιαφέρουσα τεχνική, η οποία αναφέρεται σε μια εξειδικευμένη γλώσσα και διαμορφώνει το σύστημα υπό έλεγχο και καταδεικνύει πώς αυτό επιτρέπει την αυτόματη εξαγωγή και πως ενσωματώνει εξειδικευμένες απαιτήσεις στα πρότυπα ελέγχου.

Αυτή η πρόκληση αναφέρεται στο είδος της εφαρμογής που παρατηρείται δηλαδή απαντά στην ερώτηση ΤΙ.

2.7.3. Πρόκληση: On-line έλεγχος

Παράλληλα με την παραδοσιακή άποψη του ελέγχου ως δραστηριότητα που εκτελείται πριν από την κανονική λειτουργία του συστήματος ελέγχεται και εάν ένα πρόγραμμα θα συμπεριφερθεί όπως αναμένεται, μια νέα έννοια ελέγχου που

προκύπτει γύρω από την ιδέα της παρακολούθησης μιας συμπεριφοράς συστήματος σε πραγματική λειτουργία, που χρησιμοποιεί τις δυναμικές τεχνικές ανάλυσης και self-test.

Ο έλεγχος πραγματικού χρόνου εκτέλεσης είναι σε λειτουργία πάνω από 30 έτη, αλλά έχει προκύψει ένα ανανεωμένο ενδιαφέρον λόγω της αυξανόμενης πολυπλοκότητας και της πανταχού παρούσας φύσης των συστημάτων λογισμικού. Η ορολογία δεν είναι ομοιόμορφη και χρησιμοποιούνται διαφορετικοί όροι όπως η παρακολούθηση, ο έλεγχος χρόνου εκτέλεσης και το on-line testing στην λογοτεχνία[31] ελέγχου. Όλες οι προσεγγίσεις έχουν σαν στόχο να παρατηρήσουν τη συμπεριφορά του λογισμικού στον τομέα αυτό, σχετικά με το αν συμμορφώνεται με τους στόχους του ,με την προοριζόμενη συμπεριφορά του, την ανίχνευση δυσλειτουργιών ή επίσης και με τα προβλήματα απόδοσης. Σε μερικές περιπτώσεις γίνεται προσπάθεια μιας αποκατάστασης σε απευθείας σύνδεση, ενώ σε άλλες περιπτώσεις η ανάλυση διευθύνεται εκτός σύνδεσης προκειμένου να παραχθεί ένα σχεδιάγραμμα ή για να παρθούν άλλα αξιόπιστα σχήματα.

Ένα ιδιαίτερο χαρακτηριστικό γνώρισμα του on-line testing είναι ότι δεν χρειάζεται να επινοηθεί μια ακολουθία ελέγχου για να υποκινηθεί το σύστημα υπό έλεγχο, δεδομένου ότι υπάρχει περιορισμός στο να παρατηρηθεί παθητικά τι συμβαίνει. Στην πραγματικότητα, στο πρωτόκολλο επικοινωνίας οι προσεγγίσεις ελέγχου καλούνται παθητικός έλεγχος.

Έλεγχος κατά την λειτουργία (on-line testing) όπου τα σήματα διέγερσης παρέχονται από patterns που λαμβάνονται κατά την λειτουργία του συστήματος. Αυτό που ενδιαφέρει περισσότερο είναι οι ιδιότητες της απόκρισης του συστήματος και όχι τόσο η απόκριση.

Σε γενικές γραμμές, η έμφυτη «παθητικότητα» των προσεγγίσεων του on-line testing τις καθιστά λιγότερο ισχυρές από τις δυναμικές προσεγγίσεις. Όλες οι προσεγγίσεις μπορούν να επαναδιευθυνθούν στην επαλήθευση των παρατηρηθέντων ιχνών εκτέλεσης ενάντια στους ισχυρισμούς που κάνουν σαφείς επιθυμητές ιδιότητες, ή ενάντια στις σταθερές προδιαγραφές. Για παράδειγμα, ο Bayse και οι συνεργάτες του έχουν αναπτύξει ένα εργαλείο που υποστηρίζει τον παθητικό έλεγχο ενάντια στις σταθερές που προέρχονται από το FSMs και διακρίνονται μεταξύ των απλών σταθερών και των υποχρεωτικών σταθερών. Πιο γενικά, η αποτελεσματικότητα του on-linetesting θα εξαρτηθεί από τον προσδιορισμό των ισχυρισμών αναφοράς. Επίσης, η συλλογή των ιχνών θα μπορούσε να υποβιβάσει την απόδοση συστημάτων. Πρέπει να γίνει κατανοητό ποια είναι τα καλά μέρη και ο σωστός συγχρονισμός για την εξέταση του συστήματος.

Ένα άλλο θέμα αφορά τη δυνατότητα να συνεχιστεί ο εξειδικευμένος έλεγχος, ειδικά για τις ενσωματωμένες εφαρμογές που πρέπει να επεκταθούν σε ένα περιορισμένο σε πόρους περιβάλλον, όπου τα έξοδα που απαιτήθηκαν απο την οργάνωση του ελέγχου δεν μπόρεσαν να είναι εφικτά. Μια ενδιαφέρουσα νέα ερευνητική κατεύθυνση έχει ληφθεί από τον Kapfhammer και τους συνεργάτες του, οι οποίοι αναπτύσσουν το εργαλείο Juggernaut για τον έλεγχο των εφαρμογών της Java μέσα σε ένα περιορισμένο περιβάλλον. Η αρχική ιδέα τους είναι να εκμεταλλευτούν τις πληροφορίες εκτέλεσης, που χρησιμοποιούνται μέχρι τώρα για να συντονίσουν την ακολουθία ελέγχου, καθώς επίσης και την προσαρμογή του περιβάλλοντος ελέγχου (χρησιμοποιούν συγκεκριμένα την προσαρμοστική εκφόρτωση κώδικα για να μειώσουν τις απαιτήσεις μνήμης).
Μια τέτοια ιδέα είναι πολύ ελκυστική και μπορεί να βρει βεβαίως πολλές άλλες χρήσιμες εφαρμογές.

Αυτή η πρόκληση αφορά κυρίως το ΠΟΥ και ΠΟΤΕ να παρατηρηθούν οι εκτελέσεις ελέγχου, δίνοντας ιδιαίτερη προσοχή στα δυναμικά εξελισσόμενα συστήματα.

2.8. Στόχος: Μέγιστη αποτελεσματικότητα στον έλεγχο λογισμικού

Ο τελικός σκοπός της έρευνας ελέγχου λογισμικού σήμερα, όπως ήταν και στο FOSE2000, παραμένει το να είναι «ακριβής οι πρακτικές μέθοδοι ελέγχου, τα εργαλεία και οι διαδικασίες για την ανάπτυξη μιας υψηλής ποιότητας λογισμικού». Όλα τα θεωρητικά, τεχνικά και οργανωτικά ζητήματα που ερευνούνται μέχρι τώρα πρέπει να συμφιλιωθούν σε μια βιώσιμη διαδικασία ελέγχου που θα παράγει τη μέγιστη αποδοτικότητα και αποτελεσματικότητα. Εκτός αυτού, οι έμφυτες τεχνικές λεπτομέρειες και η εκλέπτυνση των προηγμένων λύσεων που προτείνονται από τους ερευνητές πρέπει να μείνουν κρυφές πίσω από εύκολα στη χρήση ολοκληρωμένα περιβάλλοντα. Αυτό το όραμα κάνει μια τέτοια προκλητική προσπάθεια να θεωρείται η κορύφωση του ονείρου της έρευνας ελέγχου λογισμικού.

Το κύριο εμπόδιο σε ένα τέτοιο όνειρο, που υπονομεύει όλες τις αναφερθείσες ερευνητικές προκλήσεις μέχρι τώρα, είναι η αυξανόμενη πολυπλοκότητα των σύγχρονων συστημάτων. Αυτή η αύξηση πολυπλοκότητας έχει επιπτώσεις όχι μόνο στο ίδιο το σύστημα, αλλά και στα περιβάλλοντα συστήματα που επεκτείνονται, και χαρακτηρίζονται έντονα από τη μεταβλητότητα και την δυναμικότητα. Οι στρατηγικές για να ευθυγραμμίσουν την αναπτυξιακή διαδικασία για να μεγιστοποιήσουν έτσι την αποτελεσματικότητα ελέγχου ανήκουν στο σχέδιο για τη

δυνατότητα ελέγχου. Έχει αναφερθεί ήδη η δυνατότητα ελέγχου στην ομιλία για τα πρότυπα και τα χειροποίητα αντικείμενα που μπορούν να ενισχυθούν έτσι ώστε να διευκολύνουν τον έλεγχο. Εντούτοις, η δυνατότητα ελέγχου είναι μια ευρύτερη έννοια από το πώς ακριβώς το σύστημα διαμορφώνεται, περιλαμβάνει επίσης τα χαρακτηριστικά της εφαρμογής, καθώς επίσης και της ίδιας της τεχνικής ελέγχου και του περιβάλλοντος υποστήριξής του. Πράγματι, το σχέδιο για τη δυνατότητα ελέγχου έχει αποδοθεί από τους επαγγελματίες να είναι ο κύριος οδηγός εξόδων στον έλεγχο.

2.8.1. Πρόκληση: Ελέγχοντας την εξέλιξη

Οι περισσότερες δραστηριότητες ελέγχου που συνεχίζουν στη βιομηχανία περιλαμβάνουν τον επανέλεγχο του ήδη ελεγμένου κώδικα για να εξακριβώσουν ότι οι αλλαγές είτε στο πρόγραμμα είτε στο πλαίσιο δεν είχαν επιπτώσεις στην ορθότητα των συστημάτων. Όπως επισημαίνεται στο FOSE2000, λόγω του υψηλού κόστους του ελέγχου οπισθοδρόμησης, χρειάζονται αποτελεσματικές τεχνικές για να δοθεί προτεραιότητα στις περιπτώσεις ελέγχου οπισθοδρόμησης και για να αυτοματοποιηθεί η επανεκτέλεση των περιπτώσεων ελέγχου.

Μια συσχετιζόμενη ιδέα είναι η παραγοντοποίηση ελέγχου (*test factoring*), η οποία αποτελείται από τη μετατροπή ενός μακροπρόθεσμου έλεγχου συστημάτων σε μια συλλογή πολλών μικρών μονάδων ελέγχου. Αυτές οι μονάδες ελέγχου μπορούν να ασκήσουν ένα μικρό μέρος του συστήματος με τον ίδιο ακριβώς τρόπο που έκανε ο έλεγχος συστημάτων, αλλά παραμένοντας πιο συγκεντρωμένες μπορούν να επισημάνουν τις αποτυχίες σε ειδικά επιλεγμένα μέρη του συστήματος. Το *test factoring* ερευνάται σήμερα ενεργά δεδομένου ότι υπόσχεται μια σειρά βελτιώσεων μεγέθους στην εκτέλεση των ελέγχων παλινδρόμησης. Μια κοινή βασική υπόθεση πολλών υπαρχουσών προσεγγίσεων στην παλινδρόμηση είναι ότι ο έλεγχος είναι ένα χειροποίητο αντικείμενο λογισμικού που μπορεί να θεωρηθεί ως αναφορά, για παράδειγμα για την προδιαγραφή των απαιτήσεων, ή την αρχιτεκτονική λογισμικού.

Ως εκ τούτου, ένα κρίσιμο ζήτημα αφορά το πώς να διατηρηθεί ο έλεγχος ποιότητας του λογισμικού που εξελίσσεται δυναμικά στον τομέα. Πρέπει να γίνει κατανοητό ποια είναι η έννοια του ελέγχου παλινδρόμησης σε ένα τέτοιο εξελικτικό πλαίσιο, και πώς μπορεί να τροποποιηθεί και να επεκταθεί η βασική ιδέα του επιλεκτικού ελέγχου, δηλ., πόσο συχνά πρέπει να ελεγχθούν τα ίχνη εκτέλεσης; Πώς μπορούν να συγκριθούν τα ίχνη που λαμβάνονται σε διαφορετικά χρονικά διαστήματα και να παρατηρηθεί εάν η εξέλιξη έφερε οποιεσδήποτε δυσλειτουργίες;

Αυτή η πρόκληση αφορά ευρέως το ΤΙ, ΠΟΥ και ΠΟΤΕ επαναλαμβάνονται μερικές εκτελέσεις μετά την εξέλιξη του λογισμικού.

2.8.2. Πρόκληση: Αύξηση του πληθυσμού των χρηστών και των πόρων.

Έχει ήδη γίνει αναφορά στην αναδυόμενη τάση συνεχούς επικύρωσης μετά από την εγκατάσταση του λογισμικού, με τη βοήθεια των προσεγγίσεων ελέγχου on-line (δείτε την παράγραφο 2.2), όταν παραδοσιακές τεχνικές ελέγχου offline γίνονται μη αποτελεσματικές. Μιας και το λογισμικό εντατικών συστημάτων μπορεί να συμπεριφερθεί πολύ διαφορετικά σε ποικίλα περιβάλλοντα και διαμορφώσεις, χρειάζονται πρακτικοί τρόποι να μεγεθύνουν σε κλίμακα τον on-line έλεγχο για να καλύψει το ευρύ φάσμα πιθανών συμπεριφορών. Μια προσέγγιση που προκύπτει για να κατευθύνει και να εξετάσει αυτήν την πρόκληση είναι να αυξηθούν οι εσωτερικές δραστηριότητες εξασφάλισης ποιότητας με τη χρησιμοποίηση των στοιχείων που συλλέγονται δυναμικά από το πεδίο.

Αυτό είναι δέσμευση δεδομένου ότι μπορεί να βοηθήσει να αποκαλυφθούν πραγματικά φάσματα χρήσης και να εκτεθούν τα πραγματικά προβλήματα πάνω στα οποία πρέπει να εστιάσουν οι δραστηριότητες ελέγχου. Για παράδειγμα, με το να δίνεται σε κάθε χρήστη μια διαφορετική προεπιλεγμένη διαμόρφωση, η βάση των χρηστών μπορεί να εκθέσει γρηγορότερα τις συγκρούσεις διαμορφώσεων ή τα προβλήματα. Αν και μερικές εμπορικές πρωτοβουλίες έχουν αρχίσει να εμφανίζονται, όπως είναι η εμπειρία πελατών της Microsoft στο πρόγραμμα βελτίωσης, αυτές οι προσπάθειες είναι ακόμα στο αρχικό τους στάδιο, και μια σημαντική ερευνητική πρόκληση που αφήνεται ανοικτή είναι πώς να καθοριστούν αποδοτικές και αποτελεσματικές οι τεχνικές που θα εξαπολύουν τη δυνατότητα που αντιπροσωπεύεται από έναν μεγάλο αριθμό χρηστών, που εκτελούν παρόμοιες εφαρμογές, στις διασυνδεμένες μηχανές. Αυτή η υψηλού επιπέδου πρόκληση περιλαμβάνει αρκετές ακόμα συγκεκριμένες προκλήσεις, μεταξύ των όποιων είναι:

- Πώς μπορεί να γίνει συλλογή των δεδομένων χρόνου εκτέλεσης από τα προγράμματα που τρέχουν χωρίς να προκαλέσει επιβάρυνση στην απόδοση του συστήματος;
- Πώς μπορεί να γίνει αποθήκευση και διαχείριση του συλλεγμένου (ενδεχομένως μεγάλο σε ποσότητα) ποσού ακατέργαστων δεδομένων για να γίνει έτσι αποτελεσματικά η εξαγωγή της σχετικής πληροφορίας;

- Πώς μπορούν να χρησιμοποιηθούν αποτελεσματικά τα συλλεχθέντα δεδομένα για την αύξηση και τη βελτίωση του εσωτερικού ελέγχου και των δραστηριοτήτων συντήρησης;

Αυτή η πρόκληση προτείνει ότι οι χρήστες ορίζουν το ΠΟΥ και το ΠΟΤΕ για να διερευνήσουν το λογισμικό που χρησιμοποιείται.

2.8.3. Πρόκληση: Σχέδια ελέγχου

Η δυνατότητα ελέγχου είναι ένα ποιοτικό χαρακτηριστικό λογισμικού που εκθέτει το βαθμό στον οποίο ένα κατασκεύασμα ελέγχου λογισμικού διευκολύνει την διαδικασία ελέγχου.

Έχει ήδη αναφερθεί ότι πέρα από το όνειρο μιας χρήσιμης θεωρίας ελέγχου, πρέπει να γίνει κατανοητή η σχετική αποτελεσματικότητα των τεχνικών ελέγχου στους τύπους λαθών που στοχεύουν. Για να εφαρμοστεί η μηχανική στη διαδικασία ελέγχου, πρέπει να συλλεχθούν τα στοιχεία για τέτοιες πληροφορίες για να βρεθεί το αποτελεσματικότερο σχέδιο για τον έλεγχο ενός συστήματος. Αυτό γίνεται συνήθως, όταν συνδυάζεται για παράδειγμα ο λειτουργικός έλεγχος βασισμένος στις απαιτήσεις με τα μέτρα επαρκής κάλυψης του κώδικα. Μια άλλη επαναλαμβανόμενη σύσταση είναι ο συνδυασμός λειτουργικού ελέγχου με συγκεκριμένη επαλήθευση ειδικών περιπτώσεων εισαγωγής. Εντούτοις, τέτοιες πρακτικές πρέπει να υποστηριχτούν από μια συστηματική προσπάθεια να εξαχθούν και να οργανωθούν οι επαναλαμβανόμενες και αποδεδειγμένες αποτελεσματικές λύσεις στον έλεγχο των προβλημάτων σε έναν κατάλογο σχεδίων ελέγχου, ομοίως με αυτό που είναι τώρα ένα καθιερωμένο σχέδιο για τις προσεγγίσεις σχεδίου.

Η δυνατότητα ελέγχου συστατικού είναι ένα σημαντικό ποιοτικό χαρακτηριστικό που αξιολογεί το βαθμό στον οποίο ένα αντικείμενο λογισμικού διευκολύνει την διαδικασία ελέγχου. Ένας χαμηλότερος βαθμός δυνατότητας ελέγχου οδηγεί σε αυξανόμενη προσπάθεια ελέγχου. Κατά συνέπεια, είναι σημαντικό να προσδιοριστούν τα σχέδια που θα βελτιώσουν τις ικανότητες λογισμικού. Η ελεγξιμότητα και η παρατηρητικότητα είναι τα σημεία κλειδιά στη δυνατότητα ελέγχου. Για να εξεταστεί ένα συστατικό είναι απαραίτητο να ελεγχθούν οι εισαγωγές (ελεγξιμότητα) και να παρατηρηθούν τα αποτελέσματα (παρατηρητικότητα). Χωρίς αυτά τα σημεία κλειδιά είναι δύσκολο να βελτιωθεί η δυνατότητα ελέγχου συστημάτων.

Τα σχέδια προσφέρουν καλές αποδεδειγμένες λύσεις στα επαναλαμβανόμενα προβλήματα, ή, με άλλα λόγια, συγκεκριμενοποιούν ή ειδικεύονται στην συγγραφή

επίλυσης προβλημάτων. Καθώς ο έλεγχος αναγνωρίζεται ως ακριβής και επιρρεπής προσπάθεια, κάνοντας σαφής ποιές επιτυχής διαδικασίες είναι ιδιαίτερα επιθυμητές. Μια σχετική προσπάθεια είναι το χαρακτηριστικό σχήμα των Vegas και των συνεργατών του για το πώς επιλέγονται οι τεχνικές ελέγχου. Αυτοί ερεύνησαν τον τύπο γνώσης που οι επαγγελματίες χρησιμοποιούν για να επιλέξουν τις τεχνικές ελέγχου για ένα πρόγραμμα λογισμικού και έχουν παράγει έναν τυποποιημένο κατάλογο των σχετικών παραμέτρων. Εντούτοις, οι οργανώσεις που χρησιμοποιούν το προτεινόμενο σχήμα ίσως να μην αποκαλύπτουν όλες τις απαραίτητες πληροφορίες, ως εκ τούτου ερευνούν πιο πρόσφατα επίσης τις πηγές πληροφορίας, και πώς αυτές οι πηγές πρόκειται να χρησιμοποιηθούν. Παρόμοιες μελέτες απαιτούνται για να τυποποιηθούν και να τεκμηριωθούν οι επιτυχείς πρακτικές για οποιαδήποτε άλλη σχετική δραστηριότητα ελέγχου.

Τα αφηρημένα σχέδια δυνατότητας (Abstract testability patterns) ελέγχου αντιστοιχούν σε αρχιτεκτονικούς μηχανισμούς, και όχι σε αρχές δυνατότητας ελέγχου. Ένας αρχιτέκτονας λογισμικού χρησιμοποιεί μια ευδιάκριτη συλλογή σχεδίων για να σχεδιάσει ένα σύστημα. Τα αρχιτεκτονικά σχέδια παρέχουν ένα προκαθορισμένο σύνολο δομών, ευθυνών, κανόνων και οδηγιών για να οργανώσουν τη σχέση μεταξύ των συστατικών του συστήματος. Ένα σχέδιο εφαρμόζει μια ακολουθία αποφάσεων σχεδίου για να ρυθμιστούν ορισμένα ποιοτικά χαρακτηριστικά συστημάτων. Η δυνατότητα ελέγχου αρχιτεκτονικής παρουσιάστηκε από την Nancy Eickelmann και την Debra Richardson. Οι συνεργάτες τους προτείνουν ότι οι αρχιτεκτονικές αποφάσεις πρέπει να ευθυγραμμιστούν με τις στρατηγικές ελέγχου. Κατά αυτόν τον τρόπο, η δυνατότητα ελέγχου της αρχιτεκτονικής είναι ο συνδυασμός μεταξύ των αρχιτεκτονικών σχεδίων και της στρατηγικής ελέγχου.

Στην πραγματικότητα αυτή η πρόκληση εκτείνεται και στα έξι ερωτήματα του roadmap (βλ Σχήμα 2).

2.8.4. Πρόκληση: Κατανόηση του κόστους ελέγχου

Μιας και ο έλεγχος δεν πραγματοποιείται αποσπασματικά, αλλά ολοκληρωμένα με κινδύνους και περιορισμούς σε ασφάλεια καθώς επίσης και σε οικονομία, θα πρέπει τελικά να συνδεθεί η διαδικασία ελέγχου με τις τεχνικές και το κόστος τους.

Κάθε ερευνητικό άρθρο σχετικά με τον έλεγχο λογισμικού αρχίζει από τον ισχυρισμό ότι ο έλεγχος είναι μια πολύ ακριβή δραστηριότητα, αλλά στερείται των ενημερωμένων και αξιόπιστων αναφορών. Είναι κάπως ανησυχητικό ότι ακόμα και σήμερα οι αναφορές σχετικά με το υψηλό κόστος του ελέγχου συνδέονται με

εγχειρίδια που χρονολογούνται περισσότερα από είκοσι χρόνια πριν. Εν τούτοις, για να μεταφερθούν σωστά οι ερευνητικές πρόοδοι στην πράξη πρέπει να είναι σε θέση κανείς να ποσοτικοποιήσει τις άμεσες και έμμεσες δαπάνες των τεχνικών ελέγχου λογισμικού.

Δυστυχώς, η περισσότερη έρευνα στον έλεγχο λογισμικού υποστηρίζει μια θέση με ουδέτερη αξία, μιας και κάθε ελάττωμα που βρίσκεται είναι εξίσου σημαντικό ή έχει το ίδιο κόστος, κάτι που φυσικά δεν ισχύει, χρειάζονται τρόποι για να ενσωματωθεί η οικονομική αξία στην διαδικασία ελέγχου, να ενημερωθούν οι διευθυντές ελέγχου και να εφαρμόσουν την κρίση τους ώστε να επιλέξουν τις πιο κατάλληλες προσεγγίσεις. Ο Boehm και οι συνεργάτες του έχουν εισαγάγει το παράδειγμα που βασίζεται στην αξία της τεχνολογίας λογισμικού (VBSE)[32], στο οποίο επιδιώκονται ποσοτικά πλαίσια για να υποστηρίξουν τις αποφάσεις διευθυντών λογισμικού που ενισχύουν την αξία των παραδοθέντων συστημάτων λογισμικού. Πιο συγκεκριμένα, διάφορες πτυχές της εξασφάλισης ποιότητας λογισμικού έχουν ερευνηθεί συμπεριλαμβανομένης της εξασφάλισης ποιότητας βασισμένης στην αξία και τον έλεγχο βασισμένο στον κίνδυνο (VBSE)[33]. Το VBSE αφορά κυρίως τη διαχείριση των διαδικασιών, για παράδειγμα, τον έλεγχο W.R.T όπου εξετάζονται διαφορετικοί τύποι συναρτήσεων υπολογισμού της χρησιμότητας κεφαλαίου στο χρόνο ανταλλαγής της παράδοσης σε σχέση με την εμπορική του τιμή. Θα πρέπει επίσης να είναι κανείς σε θέση να ενσωματώσει τις λειτουργίες εκτίμησης της οικονομικής αναλογίας της αποτελεσματικότητας των διαθέσιμων τεχνικών ελέγχου. Η ερώτηση κλειδί είναι: λαμβάνοντας υπόψη έναν σταθερό προϋπολογισμό ελέγχου, πώς αυτός θα έπρεπε να υιοθετηθεί αποτελεσματικότερα;

Αυτή η πρόκληση εξετάζει σαφώς κυρίως το ΠΩΣ και το ΠΟΣΟ του ελέγχου.

2.8.5. Πρόκληση: Εκπαίδευση των ελεγκτών λογισμικού

Τέλος, για το λογισμικό που εξετάζεται όπως και για οποιαδήποτε άλλη δραστηριότητα τεχνολογίας λογισμικού, ένας κρίσιμος πόρος παραμένει ο ανθρώπινος παράγοντας. Πέρα από τη διαθεσιμότητα των προηγμένων τεχνικών, εργαλείων και αποτελεσματικών διαδικασιών, η ικανότητα, η υποχρέωση και το κίνητρο των ελεγκτών μπορούν να κάνουν μια μεγάλη διαφορά μεταξύ μιας επιτυχούς διαδικασίας ελέγχου και μιας ατελέσφορης. Η έρευνα κάθε πλευράς πρέπει να προσπαθήσει για την παραγωγή κατασκευασμένων αποτελεσματικών λύσεων που ενσωματώνονται εύκολα στην ανάπτυξη και δεν απαιτούν βαθιά τεχνική ειδίκευση.

Για να προγραμματίσουν και να εκτελέσουν τους ελέγχους, οι ελεγκτές λογισμικού πρέπει να λάβουν υπόψη το λογισμικό, την συνάρτηση που υπολογίζει, τα δεδομένα εισαγωγής, το πώς μπορούν πρέπει να συνδυαστούν και το περιβάλλον στο οποίο το λογισμικό θα λειτουργήσει τελικά. Αυτή η δύσκολη, χρονοβόρα διαδικασία απαιτεί την τεχνική εκλέπτυνση και τον κατάλληλο προγραμματισμό. Οι ελεγκτές δεν πρέπει να έχουν μόνο ανεπτυγμένες δυνατότητες προγραμματισμού -ο έλεγχος απαιτεί συχνά πολύ κωδικοποίηση- αλλά επίσης και την γνώση πολλών γλωσσών προγραμματισμού, τη θεωρία γραφικών παραστάσεων και τους αλγορίθμους. Για να γίνει πιο σαφές ποια είναι μερικά από τα προβλήματα λογισμικού αναφέρονται οι παρακάτω τέσσερις φάσεις:

- Μοντελοποίηση του περιβάλλοντος λογισμικού
- Επιλογή σεναρίων ελέγχου
- Εκτέλεση και επαλήθευση σεναρίων ελέγχου
- Μέτρηση προόδου ελέγχου

Τέλος, οι ελεγκτές πρέπει να εκπαιδευτούν για να καταλάβουν τις βασικές έννοιες του ελέγχου, τους περιορισμούς και τις δυνατότητες που προσφέρονται από τις διαθέσιμες τεχνικές. Η εκπαίδευση αυτή πρέπει, να συμβαδίζει με τις προόδους στον έλεγχο της τεχνολογίας ενώ είναι εμφανές ότι η εκπαίδευση πρέπει να καλύψει όλες τις χαρακτηριστικές πτυχές ελέγχου.

2.9. Εγκάρσιες προκλήσεις

Η ιστορία της έρευνας τεχνολογίας λογισμικού συγχρονίζεται με την επόμενη ανάδυση των νέων παραδειγμάτων ανάπτυξης, τα οποία υπόσχονται να απελευθερώσουν την υψηλότερη ποιότητα και το λιγότερο ακριβό λογισμικό. Σήμερα, η μόδα είναι προσανατολισμένη στις υπηρεσίες υπολογιστών και προκύπτουν πολλές ενδιαφέρουσες προκλήσεις για τον έλεγχο των εφαρμογών που είναι προσανατολισμένες προς τις υπηρεσίες.

Υπάρχουν διάφορες ομοιότητες με τα συστήματα CB, και όπως και στο CB έλεγχο, οι υπηρεσίες μπορούν να εξεταστούν από διαφορετικές απόψεις, ανάλογα με το ποιος είναι ο εμπλεκόμενος συμμέτοχος[34]. Ο υπεύθυνος για την ανάπτυξη υπηρεσιών, που εφαρμόζει μια υπηρεσία, ο φορέας παροχής υπηρεσιών, ο οποίος την επεκτείνει και την καθιστά διαθέσιμη, και ο ολοκληρωτής υπηρεσιών, ο οποίος συνθέτει τις υπηρεσίες που παρέχονται ενδεχομένως από άλλους, έχει πρόσβαση σε διαφορετικά είδη πληροφοριών και έχει διαφορετικές ανάγκες ελέγχου. Εκτός από τον υπεύθυνο για την ανάπτυξη υπηρεσιών, πρέπει να εφαρμοστούν και οι

τεχνικές ελέγχου μαύρων κουτιών, επειδή οι λεπτομέρειες σχεδίου και εκτέλεσης υπηρεσιών δεν είναι διαθέσιμες.

Μια ιδιαίτερη πτυχή υπηρεσιών είναι ότι αναγκάζονται να παρέχουν μια τυποποιημένη περιγραφή με το επεξεργάσιμο σχήμα υπολογιστών για να επιτρέψουν την αναζήτηση και την ανακάλυψη. Έτσι, δεδομένου ότι είναι συχνά η μόνη διαθέσιμη πληροφορία για την ανάλυση, η έρευνα ερευνά πώς να εκμεταλλευτεί αυτήν την υποχρεωτική προδιαγραφή για λόγους ελέγχου. Αυτήν την περίοδο, αυτή η περιγραφή περιλαμβάνει μόνο τη διεπαφή υπηρεσιών από την άποψη της υπογραφής των παρεχόμενων μεθόδων (για παράδειγμα ο καθορισμός WSDL για τις υπηρεσίες Ιστού). Υπογραφές με σαφής μεθόδους παρέχουν τη φτωχή εκφραστικότητα για λόγους ελέγχου, και στην πραγματικότητα οι ερευνητές στοχεύουν στον εμπλουτισμό μιας τέτοιας περιγραφής για να επιτρέψουν έναν πιο σημαντικό έλεγχο.

Προς την προώθηση της δια-λειτουργικότητας, μια πρώτη ανησυχία είναι να εξασφαλιστεί ότι οι υπηρεσίες συμμορφώνονται με τα καθιερωμένα τυποποιημένα πρωτόκολλα για την ανταλλαγή μηνυμάτων. Για παράδειγμα, οι οδηγίες έχουν εκδοθεί από τη (διαλειτουργικότητα υπηρεσιών Ιστού) οργάνωση WS-ι, μαζί με τον έλεγχο των εργαλείων για να παρατηρήσουν και να ελέγξουν ότι το μήνυμα που ανταλλάσσεται συμμορφώνεται με τις οδηγίες. Μια τέτοια προσέγγιση είναι βεβαίως απαραίτητη για να βεβαιώσει τη διαλειτουργικότητα, αλλά μη ικανοποιητική, και πιο συγκεκριμένα αυτό δεν εξετάζει τη δυναμική συμπεριφορά και δεν ενδιαφέρεται για την επαλήθευση των πρόσθετων λειτουργικών ιδιοτήτων.

Εννοιολογικά, μπορεί να διακριθεί ο off-line και ο on-line έλεγχος των υπηρεσιών. Όσον αφορά τον off-line έλεγχο, οι δύο προσεγγίσεις που προκύπτουν είναι βασισμένες στο πρότυπο και την μεταβολή. Για τους ελέγχους των υπηρεσιών βασισμένων στο πρότυπο, η γενική ιδέα είναι να υποτεθεί ότι οι υπεύθυνοι για την ανάπτυξη ή/και οι ολοκληρωτές υπηρεσιών παρέχουν τα κατάλληλα πρότυπα για την αυτόματη παραγωγή των περιπτώσεων ελέγχου.

Ο On-line έλεγχος , υποθέτει μια πρόσθετη σημασία για τον έλεγχο των υπηρεσιών, μιας και ο έλεγχος της πραγματικής εκτέλεσης είναι ο μόνος τρόπος να παρατηρηθεί η συμπεριφορά της εφαρμογής. Μια προσανατολισμένη προς τις υπηρεσίες εφαρμογή προκύπτει γενικά από την ολοκλήρωση διάφορων υπηρεσιών, που ελέγχεται και ανήκει σε διαφορετικούς οργανισμούς. Κατά συνέπεια, διανέμεται ο έλεγχος μιας εφαρμογής, και οι υπηρεσίες που το συνθέτουν ανακαλύπτουν η μια την άλλη σε χρόνο εκτέλεσης και μπορούν να αλλάξουν χωρίς προγενέστερη ειδοποίηση, έτσι καμία δεν μπορεί να προβλέψει το δέκτη ή τον πάροχο μιας δεδομένης υπηρεσίας.

Ο έλεγχος των υπηρεσιών εισάγει πολλά λεπτά προβλήματα, σχετικά με την υποβάθμιση της απόδοσης, την παραγωγή ανεπιθύμητων παρενεργειών και το κόστος. Πρέπει να γίνει κατανοητό από τη μια πλευρά πώς μπορεί να παρατηρηθεί η εκτέλεση σε ένα διανεμημένο δίκτυο χωρίς (επίσης) να προκαλούνται δυσμενείς επιπτώσεις στην απόδοση των συστημάτων, από την άλλη πλευρά χρειάζονται τα μέσα για να δικαιολογηθεί σε αφηρημένο επίπεδο σύνθεσης υπηρεσιών και γίνει κατανοητό τι και πότε πρέπει να ελεγχθεί.

2.9.1. Πρόκληση: Έλεγχος κατανόησης των λειτουργικών ιδιοτήτων

Ο όγκος της λογοτεχνίας ελέγχου λογισμικού κατευθύνει κατά πολύ τον έλεγχο λειτουργίας, δηλαδή, ελέγχει ότι η συμπεριφορά που έχει παρατηρηθεί συμμορφώνεται με τη λογική των προδιαγραφών. Αλλά αυτό δεν είναι αρκετό για να εγγυηθεί την πραγματική χρησιμότητα και την επάρκεια στο σκοπό του λογισμικού υπό έλεγχο: σημαντικό είναι, το λογισμικό που πρέπει να συμπεριφέρεται σωστά να εκπληρώσει τις πρόσθετες λειτουργικές ιδιότητες, ανάλογα με τη συγκεκριμένη περιοχή εφαρμογής. Ειδικότερα, ενώ η συμβατική λειτουργία ελέγχου δεν παρέχει για οποιαδήποτε έννοια του χρόνου, πολλά χαρακτηριστικά γνωρίσματα της εκτεθειμένης συμπεριφοράς ενός μέρους του λογισμικού μπορεί να εξαρτηθεί από το πότε παράγονται τα αποτελέσματα, ή πόσο καιρό παίρνει για να παραχθούν. Ομοίως, ενώ ο έλεγχος λειτουργίας δεν αντιμετωπίζει τη χρήση και τους φόρτους εργασίας των πόρων, στις συγκεκριμένες περιοχές, όπως οι τηλεπικοινωνίες, το θέμα της απόδοσης υπολογίζεται για μια σημαντική κατηγορία ελαττωμάτων [35].

Θα ήταν επιθυμητό οι προσεγγίσεις ελέγχου να εφαρμόζονται στο χρόνο ανάπτυξης έτσι θα μπορούσε να παρέχει ανατροφοδότηση όσο το δυνατόν νωρίτερα. Η τρέχουσα έρευνα που μπορεί να προετοιμάσει το έδαφος δεν είναι μεγάλη και οι υιοθετημένες προσεγγίσεις μπορούν να ταξινομηθούν σε αυτές που βασίζονται στο πρότυπο και σε αυτές που βασίζονται στο γενετικό. Μεταξύ των προηγούμενων, χρειάζονται αποτελεσματικοί τρόποι για να ενισχυθούν τα πρότυπα με τους επιθυμητούς πρόσθετους λειτουργικούς περιορισμούς. Σε αυτήν την κατεύθυνση, οι ερευνητές από το πανεπιστήμιο του Aalborg[36] ερευνούν από καιρό την επέκταση της υπάρχουσας θεωρίας ελέγχου προσαρμογής στη χρονομετρημένη ρύθμιση, παράγοντας ένα εργαλείο που μπορεί να παράγει τις περιπτώσεις ελέγχου από τις χρονομετρημένες ρυθμίσεις και τις εκτελεί παρακολουθώντας τα αποτελέσματα.

Οι προσεγγίσεις βασισμένες στο πρότυπο είναι βεβαίως ένα σημαντικό όργανο καθώς επίσης και τα ενσωματωμένα συστήματα σε πραγματικό χρόνο. Επίσης καλό

θα ήταν να δοθεί σημασία και στις καινοτόμες προσεγγίσεις: για παράδειγμα, ο Wegener και ο Grochtmann [37] έχουν προτείνει τη χρήση των εξελικτικών αλγορίθμων (evolutionary Algorithms) οι οποίοι μειώνουν τον έλεγχο που γίνεται σε πραγματικό χρόνο στην βελτιστοποίηση του προβλήματος εύρεσης της καλύτερης περίπτωσης και της χειρότερης περίπτωσης χρόνου εκτέλεσης. Μια τέτοια ιδέα θα μπορούσε να επεκταθεί σε άλλες πρόσθετες λειτουργικές ιδιότητες, ερμηνεύοντας κατάλληλα τον περιορισμό/εμπόδιο στη βελτιστοποίηση ενός προβλήματος.

3. Μεθοδολογία που χρησιμοποιήθηκε

Το λογισμικό, ένα πολύπλοκο πνευματικό προϊόν, αναπόφευκτα προκύπτει από την ανάπτυξή του με ανεπιθύμητα ελαττώματα (defects), τα οποία είναι πιθανό να οφείλονται σε ενδογενείς και εξωγενείς αιτίες και των οποίων η μη ανακάλυψη και κατάλληλη αντιμετώπιση μπορεί να προκαλέσει συνέπειες που μπορούν να ποικίλουν από απλά ενοχλητικές έως και καταστροφικές. Τεράστιες ποσότητες πόρων ξοδεύονται σε όλες τις σχετικές με τη τεχνολογία λογισμικού βιομηχανίες με σκοπό την αποφυγή αποτυχιών λογισμικού. Παρόλα αυτά, τελικά είναι αδύνατο να υπάρξει η εγγύηση ότι το λογισμικό είναι τέλειο , όπως είναι επίσης αδύνατο να προβλεφθεί και να εξαλειφθεί κάθε τι από τον εξωτερικό κόσμο που πιθανόν να απειλήσει το λογισμικό όσο αυτό εκτελείται.

Αυτό που όμως είναι δυνατό να επιτευχθεί, είναι η μείωση της πιθανότητας αποτυχίας του έργου λογισμικού, η οποία θα επιφέρει και ελάττωση της αβεβαιότητας που ενυπάρχει σε αυτό. Προϋπόθεση για την επίτευξη αυτής της ελάττωσης αποτελεί η εφαρμογή μιας κατάλληλης τεχνικής ελέγχου σε όλη τη διάρκεια της ανάπτυξης του λογισμικού ώστε να επιτευχθεί επαρκής αναγνώριση και αποτελεσματική αντιμετώπιση των διαφόρων κινδύνων που απειλούν το σύστημα λογισμικού.

Στην παρούσα εργασία έγινε έρευνα λοιπόν, σχετικά με τις αποτελεσματικότερες τεχνικές ελέγχου λογισμικού. Πιο αναλυτικά μελετήθηκε γενικά ο όρος έλεγχος λογισμικού και επισημάνθηκαν οι καλύτερες τεχνικές ελέγχου και οι πιο διαδεδομένες στο σύγχρονο κόσμο ελέγχου λογισμικού.

3.1. Το είδος της έρευνας

Το είδος της έρευνας που διεξήχθη είναι εξερευνητικό αφού εξερευνήσαμε και αναλύσαμε την ευρεία έννοια του ελέγχου λογισμικού και τι αυτή περιλαμβάνει. Διεξήχθη ποσοτική και ποιοτική ανάλυση των στοιχείων ελέγχου λογισμικού καθώς και των εργαλείων με τα οποία γίνεται αποδοτικός και αποτελεσματικός έλεγχος λογισμικού.

Πιο συγκεκριμένα στη παρούσα εργασία μελετήθηκαν έγγραφα, αναφορές, άρθρα και σχεδόν όλη η σύγχρονη βιβλιογραφία στον έλεγχο λογισμικού με στόχο να παρουσιαστεί απλά και περιγραφικά το τι ακριβώς είναι και τι περιλαμβάνει ο όρος αυτός. Στην συνέχεια επιλέχθηκαν από συγκριτικούς πίνακες και κείμενα από όλη την σχετική βιβλιογραφία που μελετήθηκε (είτε από βιβλία είτε από άρθρα είτε από έγκυρες ιστοσελίδες και ηλεκτρονικά βιβλία του διαδικτύου) τα δημοφιλέστερα αποδοτικότερα και αποτελεσματικότερα εργαλεία ελέγχου λογισμικού και παρουσιάστηκαν αναλυτικά εστιάζοντας στα πλεονεκτήματα και τα μειονεκτήματα

του κάθε εργαλείου με σκοπό την επιλογή του καταλληλότερου εργαλείο ελέγχου για την δική μας μελέτη περίπτωσης.

3.2. Τι αξιολογείται

Στη συγκεκριμένη μεθοδολογία αξιολογείται η αξιοπιστία ενός συστατικού λογισμικού, αποκαλύπτοντας μία μετρική. Η μετρική της αξιοπιστίας βασίζεται σε δύο διαφορετικά κομμάτια πληροφορίας: (1) ο αριθμός των σεναρίων δοκιμών που έχουν χρησιμοποιηθεί (πιο συγκεκριμένα, ο αριθμός διαδοχικών σεναρίων δοκιμών που δεν έχουν οδηγήσει σε μια αποτυχία του συστατικού από την τελευταία τροποποίηση του λογισμικού) και (2) μια πρόβλεψη του πόσο ικανό είναι το συστατικό να κρύψει τις ατέλειες από τα σενάρια δοκιμών.

3.3. Γιατί επιλέχτηκε η συγκεκριμένη μελέτη περίπτωσης

Η επιλογή της συγκεκριμένης μελέτης περίπτωσης ελέγχου πραγματοποιήθηκε λόγω της καθοριστικής σημασίας της λειτουργίας της εφαρμογής ηλεκτρονικής μεταφοράς χρημάτων σε ένα τραπεζικό σύστημα όπως το bMaster. Το bMaster όπως αναφέρεται και στην ενότητα 7.1.1 είναι ένα ολοκληρωμένο κεντρικό τραπεζικό σύστημα το οποίο εφαρμόζεται και υλοποιείται από τράπεζες της Κύπρου(Eurobank-Alpha Bank) και της Ελλάδος(Alpha Bank). Πιο συγκεκριμένα η ηλεκτρονική μεταφορά χρημάτων αποτελεί των πυρήνα του τραπεζικού συστήματος αφού επιτελεί τις περισσότερες και σημαντικότερες καθημερινές ενέργειες που κάνει ένας χρήστης του συστήματος. Ο έλεγχος λοιπόν της παραπάνω εφαρμογής είναι ζωτικής σημασίας αφού περιλαμβάνει το 50-60% των καθημερινών ενεργειών των χρηστών του συστήματος στις μέρες μας. Είναι κατανοητό λοιπόν πως ο κορμός των συναλλαγών του τραπεζικού συστήματος θα πρέπει να εξασφαλιστεί πως λειτουργεί σωστά και αποδοτικά.

Σχετικά με την επιλογή του εργαλείου ελέγχου WinRunner είναι το μοναδικό εργαλείο που παρέχει τόση ευχρηστία και ευκολία στην χρήση του που δικαίως έχει προκριθεί παγκοσμίως ως ένα από τα καλύτερα εργαλεία ελέγχου λογισμικού. Το WinRunner είναι ένα μοντέλο αξιολόγησης της αξιοπιστίας τύπου «μαύρου κουτιού» για την επιβεβαίωση της επάρκειας των δοκιμών, για την επισήμανση λαθών μικρού μεγέθους που μπορούν να υπάρχουν και να μην αποκαλύπτονται στο πρόγραμμα.

Πιο συγκεκριμένα η μεθοδολογία ελέγχου που ακολουθείται με το εργαλείο έλεγχου WinRunner είναι η εξής :

1. *Δημιουργία του GUI Map File*
2. *Δημιουργία Script Ελέγχου*
3. *Έλεγχος αποσφαλμάτωσης*

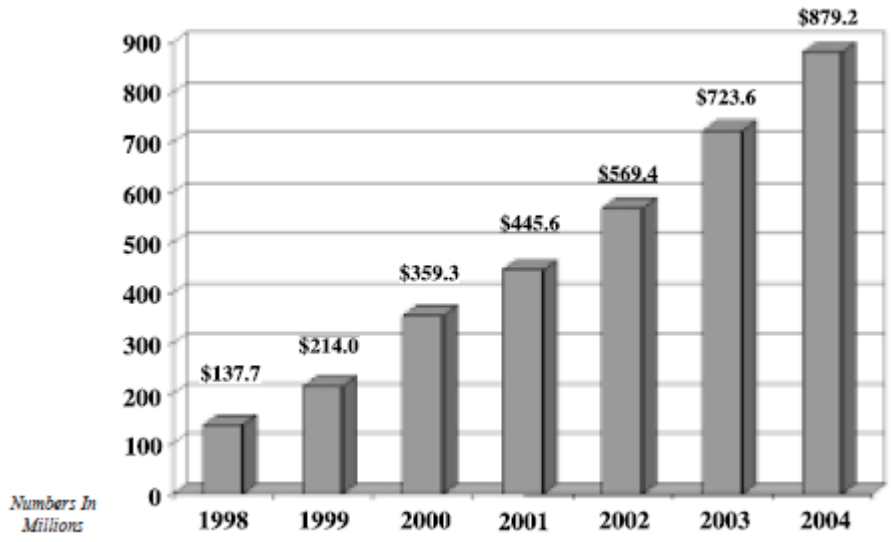
4. *Εκτέλεση των ελέγχων*
5. *Παρουσίαση αποτελεσμάτων*
6. *Αναφορά λαθών*

Στην περίπτωση που οι δοκιμές δεν αποκαλύψουν κανένα λάθος, το συγκεκριμένο μοντέλο θεωρεί ότι το υπό έλεγχο λογισμικό δεν έχει λάθη παρέχοντας μία αναφορά που βασίζεται στις επαναλαμβανόμενες δοκιμές για την ορθότητα του λογισμικού.

Πιο αναλυτικά προσφέρει:

1) Εύκολη δημιουργία σεναρίων ελέγχου.
2) Επιθεώρηση δεδομένων.
3) Ανάλυση δεδομένων
4) Παρουσίαση δεδομένων

Παρακάτω αναφέρονται τα όνειρα της έρευνας ελέγχου λογισμικού και πως εξελίσσεται η ίδια η έρευνα ελέγχου λογισμικού όλα αυτά τα χρόνια. [38] Περιγράφεται η πορεία της έρευνας ελέγχου λογισμικού όλα αυτά τα χρόνια ενώ για κάθε ένα από τα όνειρα εξετάζονται μερικές σχετικές προκλήσεις ώστε να προωθηθεί η κατάσταση προόδου πιο κοντά στο ίδιο το όνειρο.



Διάγραμμα 1. Παγκόσμια ανάπτυξη της αγοράς ελέγχου λογισμικού (1998-2004)

Πηγή: «Newport Group, Inc. 2001, *IT Trends Research and Reporting*»

4. Τεχνικές και επίπεδα ελέγχου λογισμικού

Ο έλεγχος λογισμικού είναι η διαδικασία επικύρωσης και επαλήθευσης του λογισμικού συστήματος που πραγματοποιείται πάνω σε ένα ολοκληρωμένο διαδραστικό σύστημα *α) για να αξιολογήσει κατά πόσο οι λειτουργίες που εκτελεί είναι ταυτόσημες και ικανοποιούν τις επιχειρηματικές και τεχνικές προδιαγραφές που τέθηκαν, κατά την διάρκεια του σταδίου ορισμού των προδιαγραφών, που αποτελούν νομικό έγγραφο και καθοδηγούν ολόκληρη την διαδικασία σχεδιασμού και υλοποίησης* και β) *ότι το υπό έλεγχο λογισμικό σύστημα εκτελεί τις λειτουργίες του όπως αναμενόταν*. Η ικανοποίηση των προδιαγραφών θα καθορίσει το ποσοστό αξιοπιστίας του υπό έλεγχο λογισμικού συστήματος.

4.1. Κατηγορίες Ελέγχου

Μόλις παραχθεί ο πηγαίος κώδικας, το λογισμικό πρέπει να εξεταστεί ώστε να μπορούν τα λάθη να προσδιοριστούν και να αφαιρεθούν πριν την τελική παράδοση στον πελάτη. Ενώ είναι πρακτικά αδύνατο να αφαιρεθεί το κάθε λάθος από ένα μεγάλο πακέτο λογισμικού, ο στόχος του μηχανικού λογισμικού είναι να αφαιρεθούν όσο το δυνατόν περισσότερα, αρχικά, στον κύκλο ανάπτυξης του λογισμικού.

Υπάρχουν δύο βασικές τεχνικές ελέγχου του τυπικού λογισμικού, έλεγχος της εφαρμογής τι παίρνει ως είσοδο και τι ως έξοδο (μαύρο κουτί) και η εξέταση της εσωτερικής λογικής των τμημάτων λογισμικού (άσπρο-κουτί). Η διαδικασία ελέγχου των συστημάτων αρχίζουν με μια αναθεώρηση της ανάλυσης και των σχεδιαστικών μοντέλων. Χρησιμοποιούνται περιπτώσεις χρήσης από το πρότυπο ανάλυσης για να αποκαλύψουν λάθη επικύρωσης λογισμικού. Ο έλεγχος πρέπει να προγραμματιστεί και να σχεδιαστεί.

Ένας διαχωρισμός των τεχνικών ελέγχου γίνεται βάσει του αν κατά την διάρκεια του ελέγχου εκτελείται το πρόγραμμα, οπότε διεξάγεται δυναμικός έλεγχος *(dynamic testing)*, ή δεν εκτελείται, οπότε πραγματοποιείται η λεγόμενη στατική ανάλυση *(static analysis)* στην οποία περιλαμβάνονται τεχνικές όπως η απόδειξη προγράμματος *(program proving)*, η συμβολική εκτέλεση *(symbolic execution)* και η ανάλυση ανωμαλιών *(anomaly analysis).*

Ως στατικός έλεγχος μπορούν να χαρακτηριστούν τα reviews, τα walkthroughs ή τα inspections ενώ η εκτέλεση του κώδικα με βάση ένα σύνολο από σενάρια ελέγχου

(Test Cases (TC)) συγκαταλέγεται στον δυναμικό έλεγχο. Παρατηρείται όμως συχνά ότι ο στατικός έλεγχος αποφεύγεται και αυτό οφείλεται στην δυνατότητα του δυναμικού ελέγχου να ανιχνεύει το μεγαλύτερο φάσμα των ανιχνεύσιμων λαθών. Ο δυναμικός έλεγχος μπορεί να ανεξαρτητοποιηθεί με την διαδικασία ολοκλήρωσης του λογισμικού εφόσον μπορούν να ελεγχθούν ξεχωριστά διάφορες ενότητες ή απλά διαδικασίες του προγράμματος. Ο έλεγχος λογισμικού αποτελείται από τη δυναμική επαλήθευση της συμπεριφοράς ενός προγράμματος για ένα πεπερασμένο σύνολο περιπτώσεων ελέγχου, που επιλέγεται κατάλληλα από τη συνήθως απεριόριστη περιοχή εκτελέσεων, ενάντια στη διευκρινισμένη αναμενόμενη συμπεριφορά.

4.2. Στατικός έλεγχος - Δομικός έλεγχος

Ένας στατικός έλεγχος είναι ένας έλεγχος που αξιολογεί την ποιότητα του συστήματος χωρίς το σύστημα να εκτελείται πραγματικά. Ένας στατικός έλεγχος εξετάζει το σύστημα-ή τα μέρη από τα χειροποίητα αντικείμενα συσχετιζόμενα με το σύστημα-για να δει εάν μπορεί να βρει αρχικά τα προβλήματα. Όταν οι προγραμματιστές διαβάζουν το κώδικα τους αφού τον γράψουν, αυτό ονομάζεται «desk-checking», το οποίο είναι ένα είδος στατικού ελέγχου. Ένα άλλο είδος στατικού ελέγχου, σε αυτήν την περίπτωση, είναι η συνεδρίαση της αναθεώρησης που αξιολογεί τις απαιτήσεις, το σχέδιο ή τον κώδικα. Το όφελος είναι σαφές μόλις σκεφτεί κανείς το εξής: Εάν βρεθεί ένα πρόβλημα στις απαιτήσεις που αφορούν το σύστημα προτού αυτό μετατραπεί σε πρόβλημα στο σύστημα, θα κερδηθεί χρόνος και χρήμα. Ακόμα κι αν βρεθεί το πρόβλημα με την αναθεώρηση του κώδικα, αυτό βοηθά στα γενικά προβλήματα που ίσως αντιμετωπίσει κανείς στον έλεγχο του συστήματος, εγκαθιστώντας, και εκτελώντας το σύστημα για να βρεθεί το λάθος. Δεν περιλαμβάνει όλος ο στατικός έλεγχος τους ανθρώπους που κάθονται σε έναν πίνακα και εξετάζουν ένα έγγραφο.

Για να αξιοποιηθεί σωστά ο στατικός έλεγχος, πρέπει να ξεκινήσει στη σωστή στιγμή. Παραδείγματος χάριν, επανεξετάζοντας τις απαιτήσεις μετά τους προγραμματιστές που έχουν τελειώσει την κωδικοποίηση, το σύστημα μπορεί να βοηθήσει τους ελεγκτές να σχεδιάσουν τα σενάρια ελέγχου. Εντούτοις, τα σημαντικά αποτελέσματα στην επένδυση του στατικού ελέγχου δεν είναι πλέον διαθέσιμα, δεδομένου ότι οι ελεγκτές δεν μπορούν να αποτρέψουν τα λάθη στον κώδικα που έχει ήδη γραφτεί. Για να υπάρχουν βέλτιστα αποτελέσματα, ένας στατικός έλεγχος πρέπει να γίνει το συντομότερο δυνατόν αφότου έχει δημιουργηθεί το στοιχείο που εξετάζεται, ενώ οι υποθέσεις και οι εμπνεύσεις παραμένουν φρέσκες στο μυαλό του δημιουργού και κανένα από τα λάθη στο πρόγραμμα δεν έχει προκαλέσει αρνητικές συνέπειες στις προς τα κάτω

διαδικασίες. Επίσης, ο αριθμός εκτελέσεων που μπορεί να παρατηρηθεί ρεαλιστικά στον έλεγχο πρέπει προφανώς να είναι πεπερασμένος. Σαφώς, "αρκετός" έλεγχος πρέπει να εκτελεσθεί για να παρασχεθεί η απαραίτητη επιβεβαίωση. Πράγματι, ο έλεγχος υπονοεί πάντα μια ανταλλαγή μεταξύ των περιορισμένων φυσικών πόρων και των προγραμμάτων, και εγγενώς απεριόριστες απαιτήσεις ελέγχων: αυτή η σύγκρουση μαρτυρά γνωστά προβλήματα ελέγχων, που περιέχουν και προβλήματα τεχνικής φύσης (κριτήρια για την απόφαση επάρκειας του ελέγχου) και διοικητικής φύσης (υπολογισμός της προσπάθειας να υποβληθεί σε έλεγχο).

4.2.1. Έλεγχος White Box

Ο στατικός έλεγχος είναι ένας ισχυρός έλεγχος αλλά όχι μια καθορισμένη τεχνική ελέγχου. Σε κάποιο σημείο πρέπει πραγματικά να εκτελεστεί το σύστημα και να γίνει εύρεση των λαθών σε αυτό [39]. Οι **έλεγχοι που περιλαμβάνουν την εκτέλεση του συστήματος υπό έλεγχο, καλούνται δυναμικοί έλεγχοι. Οι δομικοί έλεγχοι είναι ένα σημαντικό παράδειγμα των δυναμικών δοκιμών.** Οι δομικοί έλεγχου είναι βασισμένοι στον τρόπο με τον οποίο το σύστημα χτίζεται. Μερικοί άνθρωποι αναφέρονται στους δομικούς ελέγχους ως «white box» ή ίσως πιο σωστά ως «glass box», επειδή ο tester κοιτάζει μέσα στο «κιβώτιο» (το σύστημα υπό έλεγχο) και αντλεί τους ελέγχους από αυτήν την περιοχή. Οι δομικοί έλεγχοι εφαρμόζονται χαρακτηριστικά στα επιμέρους συστατικά και τις διεπαφές, που είναι ιδιαίτερα αποτελεσματικές στην ανακάλυψη των εντοπισμένων λαθών στις ροές ελέγχου και στοιχείων.

Κατά τον δυναμικό έλεγχο υπάρχει η δυνατότητα να ακολουθηθούν δύο μέθοδοι παραγωγής και ελέγχου σεναρίων ελέγχου. Υπάρχει το Black Box κατά την οποία ο έλεγχος γίνεται χωρίς να έχει γνώση της εσωτερικής υλοποίησης του προγράμματος. ***Μέθοδοι οι οποίες χρησιμοποιούν το Black Box Testing είναι η Specification-based testing, model-based testing, boundary value analysis και πολλές άλλες.***

Μια μορφή δομικού ελέγχου περιλαμβάνει τη δημιουργία συνηθισμένων δεδομένων ελέγχου. Οι ελεγκτές που δημιουργούν τέτοια δεδομένα πρέπει συχνά να καταλάβουν την ελλοχεύουσα βάση δεδομένων σχημάτων ή σχεδίων, ειδικά κατά την εργασία σε προσεκτικά επεξεργασμένα σύνολα δεδομένων που εξετάζουν τα όρια της ισχύος δεδομένων με έναν ή περισσότερους τρόπους. Εκτός από τον έλεγχο των επιμέρους συστατικών και των ροών στοιχείων, με τον προσεκτικό σχεδιασμό μπορούν να επαναχρησιμοποιηθούν τέτοια στοιχεία ελέγχου για άλλα είδη ελέγχων.

Μπορεί για παράδειγμα να υπάρχει ένας πίνακας φόρων που να δείχνει την αναμενόμενη έξοδο για συγκεκριμένες εισόδους, όμως να μην γνωρίζουν οι χρήστες γενικά πως υπολογίζεται ο φόρος. Ο αλγόριθμος υπολογισμού του φόρου εξαρτάται από τις φορολογικές κατηγορίες και όσο τα όρια των φορολογικών ορίων όσο και τα αντίστοιχα ποσοστά είναι μέρος της εσωτερικής επεξεργασίας που επιτελεί το συστατικό. Εξετάζοντας το συστατικό σαν ένα κλειστό, δεν μπορούν να επιλεχθούν αντιπροσωπευτικές περιπτώσεις ελέγχου επειδή είναι αρκετά γνωστή η εσωτερική επεξεργασία που εκτελείται.

Για να αντιμετωπιστεί αυτό το πρόβλημα, μπορεί κανείς αντίθετα να δει το αντικείμενο ελέγχου σαν ένα ανοικτό κουτί (open box) (μερικές φορές καλείται και διαφανές κουτί (clear box) ή λευκό κουτί (white box)). Για παράδειγμα μπορούν να επινοηθούν περιπτώσεις ελέγχου που να εκτελούν όλες τις εντολές ή όλα τα μονοπάτια ελέγχου μέσα στα συστατικά ώστε να εξασφαλιστεί ότι το αντικείμενο που θα ελεγχθεί δουλεύει σωστά.

Όταν επιλέγεται ο τρόπος ελέγχου, δε χρειάζεται να επιλεχθεί αποκλειστικά έλεγχος ανοικτού ή κλειστού κουτιού. Γενικά η επιλογή φιλοσοφίας ελέγχου εξαρτάται από πολλού παράγοντες, στους οποίους συμπεριλαμβάνονται οι ακόλουθοι:

- Ο αριθμός των πιθανών λογικών μονοπατιών
- Η φύση των δεδομένων εισόδου
- Το μέγεθος των υπολογισμών που εμπλέκεται
- Η πολυπλοκότητα των αλγορίθμων.

Οι δοκιμές ελέγχου παράγονται προσεκτικά βάση ενός ή συνδυασμού περισσότερων κριτηρίων κάλυψης. Για μια τιμή εισόδου που δίνεται στο σύστημα ελέγχεται όχι μόνο το αν δημιουργείται το σωστό αποτέλεσμα, αλλά και το πώς προκύπτει. Πιο κάτω γίνεται αναφορά στα κύρια πλεονεκτήματα καθώς και μειονεκτήματα της τεχνικής αυτής.

Πλεονεκτήματα:

Αυτή η μέθοδος αναγνωρίζει με σχετική ευκολία το φαινόμενο της συμπτωματικής ακρίβειας (coincidental correctness). Το φαινόμενο αυτό αναφέρεται στις περιπτώσεις όπου παρόλο το αποτέλεσμα προκύπτει ορθό, ο τρόπος που υπολογίζεται είναι λανθασμένος. Για τα συγκεκριμένα δεδομένα ελέγχου δεν είναι και τόσο σημαντικό αλλά κατά πάσα πιθανότητα για άλλα να μην είναι ορθό το αποτέλεσμα.

• Επίσης με την μέθοδο αυτή μπορούν να ελεγχθούν όλοι οι πιθανοί τρόποι λειτουργίας του μιας και ο έλεγχος γίνεται βάση του πηγαίου κώδικα.

• Με την μέθοδο αυτή μπορούν να ανακαλυφθούν περιπτώσεις νεκρού κώδικα.

Δηλαδή περιπτώσεις κώδικα που δεν εκτελούνται ποτέ και δεν μπορούν να ανακαλυφθούν με την τεχνική του black box testing.

Μειονεκτήματα:

• Με την μέθοδο αυτή είναι πολύ δύσκολο να εντοπισθούν λάθη λογικής.

• Είναι σχεδόν αδύνατο σε περιπτώσεις μεγάλων συστημάτων να εξεταστούν όλα τα μονοπάτια του κώδικα έτσι ώστε να ανακαλυφθούν όλα τα λάθη που υπάρχουν. Αυτό είναι ένα πρόβλημα το οποίο περιορίζει αρκετά την τεχνική αυτή και γίνονται προσπάθειες επίλυσής του.

• Δεν υπάρχει τρόπος να εντοπισθούν τα μονοπάτια τα οποία έχουν παραληφθεί.

4.3. Δυναμικός - Συμπεριφοριστικός έλεγχος

Ένας δυναμικός έλεγχος υπονοεί πάντα την εκτέλεση του προγράμματος για (άξιες) εισαγωγές. Για να υπάρχει ακρίβεια, μόνο η αξία εισαγωγής δεν είναι πάντα ικανοποιητική για να καθορίσει έναν έλεγχο, ως σύνθετο, ένα μη ντετερμινιστικό σύστημα μπορεί να αντιδράσει διαφορετικά σε μια συγκεκριμένη εισαγωγή, ανάλογα με την κατάσταση του συστήματος. Διαφορετικά από τον έλεγχο, και συμπληρωματικά με αυτό, είναι οι στατικές τεχνικές ανάλυσης, όπως η αξιολόγηση από ομότιμους και η επιθεώρηση (όπου μερικές φορές αναφέρονται εσφαλμένα σαν "στατικό testing") αυτά δεν εξετάζονται ως τμήμα αυτής της γνωστικής περιοχής (ούτε είναι η εκτέλεση του προγράμματος σε συμβολικές εισαγωγές, ή συμβολική αξιολόγηση). Ο συμπεριφοριστικός έλεγχος εστιάζει πιο χαρακτηριστικά στα παγκόσμια ζητήματα των ροών εργασίας, σε διαμορφώσεις, απόδοση, και ούτω καθεξής. Λόγω της εστίασης στον τρόπο με τον οποίο το σύστημα δουλεύει, οι συμπεριφοριστικοί έλεγχοι μπορούν να καλύψουν πολλά από τα σημαντικά σχεδιαγράμματα χρήσης και από τους ποιοτικούς κινδύνους. Ενώ μπορεί να εκτελείται συμπεριφοριστικός έλεγχος συγκεκριμένων συστατικών -και μερικές μεθοδολογίες προγραμματισμού το προωθούν ώστε να γίνεται έτσι-οι περισσότεροι συμπεριφοριστικοί έλεγχοι εστιάζουν στις μεγάλες μερίδες του συστήματος ή των πλήρως ολοκληρωμένων συστημάτων. Καθώς το σύστημα ενώνεται, οι συμπεριφορές του συστήματος αρχίζουν να γίνονται πιο ορατές. Οι Testers μπορούν έπειτα να αποκτήσουν την επίγνωση στην εμπειρία του χρήστη σε θέματα ποιότητας.

Ο χειρωνακτικός συμπεριφοριστικός έλεγχος περιλαμβάνει γενικά τον εσωτερικό προγραμματισμό, το προσεκτικό σχεδιασμό, και τον προσεκτικό έλεγχο του αποτελέσματος. Ένας καλός χειρωνακτικός ελεγκτής κρίνει αμέσως τα αποτελέσματα που έχουν παρατηρηθεί κάτι που ένα αυτοματοποιημένο εργαλείο δεν μπορεί. Οι έμπειροι χειρωνακτικοί ελεγκτές ξέρουν πώς να ακολουθήσουν μια αλυσίδα από λάθη - ή να φωτίσουν μια νέα αλυσίδα από λάθη καθώς προχωρά ο έλεγχος.

4.3.1. Έλεγχος Black Box

Μια μορφή δυναμικού ελέγχου είναι ίσως ο πιο γνωστός για να ελέγξει τους επαγγελματίες, ο συμπεριφοριστικός έλεγχος «Black Box». Οι συμπεριφοριστικοί έλεγχοι, όπως το όνομα «Black Box» υποδηλώνει, **εστιάζει σε αυτό** που το **σύστημα κάνει**, όχι στο πώς το κάνει. Κάποιοι επαγγελματίες ελεγκτές θεωρούν πως η εσωτερική γνώση για το πώς δουλεύει το σύστημα μπορεί πραγματικά να καταστήσει έναν συμπεριφοριστικό tester λιγότερο αποτελεσματικό, επειδή αυτή η κατανόηση μπορεί να παρεμποδίσει τη δυνατότητα να εξεταστεί το σύστημα με τους τρόπους που ο χρήστης θα το χρησιμοποιούσε πραγματικά. Εντούτοις, έχει διαπιστωθεί ότι μπορεί να ελαχιστοποιηθεί αυτό το πρόβλημα μέσω του προσεκτικά σχεδιασμένου ελέγχου και εκτέλεσης, και η εσωτερική ματιά στο πώς το σύστημα δουλεύει μπορεί να βοηθήσει τους συμπεριφοριστικούς testers να βρουν περισσότερα λάθη και να γράψουν περισσότερο παρατηρητικές εκθέσεις λαθών.

Ένας άλλος σημαντικός τύπος χειρωνακτικού συμπεριφοριστικού ελέγχου, είναι αυτός που καλείται και **ζωντανός έλεγχος.** Στον ζωντανό έλεγχο οι ελεγκτές ή χρήστες δοκιμάζουν το σύστημα με πραγματικά δεδομένα, τις ροές εργασίας χρηστών, τις διαμορφώσεις πεδίων, και άλλες περιπτώσεις πραγματικής χρήσης. Ο έλεγχος αποδοχής είναι μια μορφή ζωντανού ελέγχου συνήθη σε μελέτες ανάπτυξης της Τεχνολογίας Πληροφοριών. Για τα προσανατολισμένα στην αγορά συστήματα, ο έλεγχος beta είναι μια συνηθισμένη ζωντανή τεχνική ελέγχου. Ένα μέρος κλειδί του συμπεριφοριστικού ελέγχου είναι αυτό που καλούν «έλεγχος φιλοξενουμένων (guest testing) », όπου οι πελάτες στέλνουν τους ικανότερους χρήστες του συστήματος στα γραφεία του πελάτη για να συμμετέχει στις τελικές φάσεις του συμπεριφοριστικού ελέγχου.

Ο καλός συμπεριφοριστικός έλεγχος, απαιτεί κάποιο ποσό κατανόησης της κατάλληλης συμπεριφοράς του συστήματος. Με άλλα λόγια, η ομάδα ελέγχου χρειάζεται έναν καλό βαθμό διείσδυσης στην επιχειρησιακή περιοχή ή την περιοχή εφαρμογής του συστήματος. Αυτό ισχύει ιδιαίτερα στα πολυσύνθετα συστήματα

όπως είναι τα ιατρικά συστήματα και τα τραπεζικά. Εντούτοις, άνθρωποι που έχουν υπόβαθρο μόνο στην περιοχή της εφαρμογής -και δεν έχουν καμία κατανόηση του ελέγχου-συχνά αποτυγχάνουν να καταλάβουν πόσο γρήγορα ένας έμπειρος ελεγκτής μπορεί να εμφανιστεί για να πετύχει στην περιοχή εφαρμογής.

Για παράδειγμα σε ένα τραπεζικό σύστημα που έγιναν έλεγχοι από άτομα που δεν είχαν κανένα οικονομικό υπόβαθρο ,οι χρήστες έμειναν έκπληκτοι στο πόσο αποτελεσματικοί ήταν στην εύρεση των λαθών στο σύστημα.

Ο Black-box έλεγχος ή concrete box είναι μια κατηγορία, όπου οι συνθήκες ελέγχου για το πρόγραμμα, δημιουργούνται βάση των λειτουργιών και των προδιαγραφών του συστήματος. Ο προγραμματιστής το μόνο που χρειάζεται είναι πληροφορίες για τις τιμές εισόδου. Στη συνέχεια γνωρίζοντας αφαιρετικά την λειτουργία του συστήματος, αγνοώντας τον κώδικα (για αυτό λέγεται μαύρο κουτί), παρατηρεί τα αποτελέσματα και τα συγκρίνει με τα επιθυμητά. Η τεχνική αυτή βασίζεται στις προδιαγραφές και τις λειτουργίες του προγράμματος. Λόγο του ότι δεν χρειάζεται γνώση στον κώδικα του λογισμικού, δεν είναι αναγκαίο ο ελεγκτής να είναι ο ίδιος ο προγραμματιστής. Πιο κάτω γίνεται αναφορά στα κύρια πλεονεκτήματα καθώς και μειονεκτήματα της τεχνικής αυτής.

Πλεονεκτήματα:

- Είναι αποτελεσματικότερος σε μεγάλα κομμάτια κώδικα σε σχέση με το white box testing
- Ο ελεγκτής δεν χρειάζεται προγραμματιστικές γνώσεις, αρκεί να γνωρίζει για συγκεκριμένες εισόδους τις επιθυμητές εξόδους.
- Ο προγραμματιστής και ο ελεγκτής είναι ανεξάρτητοι ο ένας απ' τον άλλο.
- Οι έλεγχοι γίνονται από την πλευρά άποψης του χρήστη.
- Βοηθά στο να ανακαλυφθούν οποιεσδήποτε ασάφειες ή ασυνέπειες στις προδιαγραφές.

Μειονεκτήματα:

- Μόνο ένας μικρός αριθμός πιθανών inputs μπορεί πραγματικά να ελεγχθεί μιας και είναι αδύνατο να ελεγχθούν όλοι οι δυνατοί συνδυασμοί. Αποτέλεσμα αυτού είναι ο μη ικανοποιητικός έλεγχος μιας και δεν ελέγχονται όλα τα πιθανά μονοπάτια.
- Για να δημιουργηθούν οι περιπτώσεις ελέγχου πρέπει να έχουμε σαφή προδιαγραφές (μη διφορούμενες, συγκεκριμένες).
- Μπορεί να υπάρξει περιττή επανάληψη των test inputs, αν ο ελεγκτής δεν ενημερώνεται για τα test cases του προγραμματιστή.
- Δεν μπορεί να κατευθυνθεί προς συγκεκριμένα τμήματα του κώδικα που μπορούν να είναι πολύ σύνθετα και επομένως επιτρέπονται περισσότερα λάθη.
- Οι περισσότερες έρευνες που αφορούν τον έλεγχο έχουν κατευθυνθεί προς την τεχνική του white box testing

4.4. Η μέθοδος ελέγχου record / playback

Στην αγορά υπάρχουν αρκετές εφαρμογές λογισμικού των οποίων σκοπός είναι ο έλεγχος του γραφικού περιβάλλοντος κάποιου λογισμικού. Μια από τις τεχνικές που χρησιμοποιείται ευρέως σήμερα σε διάφορα τέτοια λογισμικά συστήματα είναι η τεχνική Capture / Playback·

Με μια απλή περιδιάβαση σε ένα τέτοιο λογισμικό σύστημα ελέγχου παρατήρησα πως η τεχνική αυτή χρησιμοποιεί ένα εργαλείο που ηχογραφεί χειρονακτικές αλληλεπιδράσεις του χρήστη με το υπό έλεγχο λογισμικό σύστημα. Κατά την διάρκεια αυτών των αλληλεπιδράσεων το σύστημα αυτό ζητούσε συνεχώς στοιχεία επαλήθευσης των κινήσεων αυτών.

Μετά το πέρας της ηχογράφησης των αλληλεπιδράσεων με το υπό έλεγχο λογισμικό σύστημα το σύστημα δίνει την επιλογή να γίνει εκτέλεση των ηχογραφημένων αλληλεπιδράσεων. Αν το λογισμικό που έχει τεθεί υπό έλεγχο αντιδράσει διαφορετικά σε κάποια από τις εκτελέσεις, τότε το σύστημα που κάνει τον έλεγχο παρουσιάζει την αντίδραση σαν ελάττωμα στο υπό έλεγχο λογισμικό σύστημα.

4.5. Έλεγχος Gray Box

Πρόκειται[40] για τον συνδυασμό των πιο πάνω μεθόδων ελέγχου, του black box και του white box testing. Συγκεκριμένα στην περίπτωση αυτή το άτομο που θα κάνει τον έλεγχο λογισμικού γνωρίζει από πριν τον κώδικα του συστήματος που εξετάζεται. Έτσι, κατά την δημιουργία σεναρίων ελέγχου, αφιερώνει στον κώδικα ένα περιορισμένο αριθμό σεναρίων ελέγχου για να πραγματοποιήσει τον έλεγχο. Στα υπόλοιπα σενάρια ελέγχου ο προγραμματιστής χρησιμοποιεί μια black box προσέγγιση, με σενάρια ελέγχου τα οποία, δεδομένου κάποιων εισόδων, ελέγχουν το αποτέλεσμα ή έξοδο του λογισμικού συστήματος.

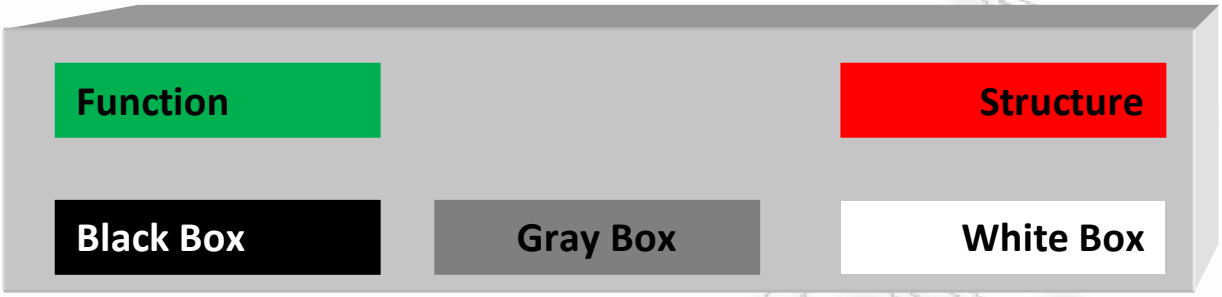
Όπως και στο black box testing, η προσέγγιση που χρησιμοποιείται βασίζεται στη διεπαφή ή γραφικό περιβάλλον του λογισμικού συστήματος με μόνη διαφορά ότι το άτομο που διενεργεί τον έλεγχο γνωρίζει τον κώδικα ή μέρος του κώδικα του λογισμικού συστήματος. Γι' αυτό τον λόγο παράγονται καλύτερα σενάρια ελέγχου που δίνουν καλύτερα αποτελέσματα από τις προηγούμενες δύο μεθόδους ελέγχου αφού υπάρχει περισσότερη γνώση σε ότι αφορά το υπό έλεγχο λογισμικό σύστημα.

Ο βασικός σκοπός χρήσης του grey box testing είναι η επαναχρησιμοποίηση των σεναρίων ελέγχου αφού γίνουν αλλαγές και / ή προσθήκες στον κώδικα του λογισμικού συστήματος και η ανίχνευση λαθών που σχετίζονται με κακό σχεδιασμό και κακή υλοποίηση του υπό έλεγχο συστήματος λογισμικού.

Η μέθοδος αυτή είναι ένας συνδυασμός των δύο πιο πάνω τεχνικών. Δηλαδή συνδυάζει τόσο τον έλεγχο βάση της εσωτερικής δομής όσο και τον έλεγχο βάση των προδιαγραφών του λογισμικού. Σκοπός της δημιουργίας της πιο πάνω τεχνικής είναι η όσο το δυνατόν καλύτερη αξιοποίηση των πλεονεκτημάτων των δυο προηγούμενων τεχνικών, με αποτέλεσμα τον καλύτερο έλεγχο.

Οι τεχνικές ελέγχου γκρίζου κουτιού συνδυάζουν την μεθοδολογία ελέγχου του άσπρου και του μαύρου κουτιού. Οι τεχνικές ελέγχου γκρι κουτιού χρησιμοποιούνται για τον έλεγχο ενός τμήματος του λογισμικού ενάντια στις προδιαγραφές του αλλά χρησιμοποιώντας κάποια γνώση της εσωτερικής δομής εργασίας του κώδικα του συστήματος επίσης. Ο έλεγχος γκρι κουτιού μπορεί επίσης να περιλαμβάνει την αντίστροφη εφαρμοσμένη μηχανική για να καθορίσει, για παράδειγμα, τις τιμές ορίου ή τα μηνύματα λάθους. Είναι ακόμα μια διαδικασία που περιλαμβάνει το λογισμικό ελέγχου έχοντας ήδη κάποια γνώση της λογικής του κώδικα του προγράμματος. Η κατανόηση των εσωτερικών λειτουργιών του

προγράμματος στον έλεγχο του γκρι κουτιού είναι περισσότερη από τον έλεγχο του μαύρου κουτιού, αλλά λιγότερο από τον σαφή έλεγχο του άσπρου κουτιού.



Εικόνα 2 «Παρουσίαση τεχνικών ελέγχου»

4.6. Η σημασία επιλογής της σωστής τεχνικής

Η επιλογή των σωστών τεχνικών είναι κρίσιμη για την επίτευξη ενός καλού αποτελέσματος από την επένδυση ελέγχου λογισμικού. Μερικές δοκιμές περιλαμβάνουν την ανάλυση δομής του συστήματος, ενώ άλλες περιλαμβάνουν την ανάλυση της συμπεριφοράς του συστήματος. Κάθε τεχνική μπορεί να περιλαμβάνει πρόσθετες δεξιότητες και συγκεκριμένους συμμετέχοντες, και να συνεπάγεται η κατάλληλη χρήση εργαλείου-ή όχι.

Το αποτέλεσμα της επένδυσης δεν επηρεάζεται μόνο από τον σωστό έλεγχο , αλλά και από τον σωστό τρόπο που έχει επιλεχτεί για να γίνει ο έλεγχος. Οι διάφορες τεχνικές υπάρχουν για το σχέδιο και την εκτέλεση ελέγχου. Υπάρχουν πολλές τεχνικές που αφορούν τη σχεδίαση ελέγχου και την εκτέλεση. Μερικοί έλεγχοι δεν απαιτούν το σύστημα να είναι σε λειτουργία ενώ άλλοι ναι. Μερικοί έλεγχοι απαιτούν εσωτερική γνώση για το πώς το σύστημα λειτουργεί, ενώ άλλοι απαιτούν μια κατανόηση σχετικά με το τι το σύστημα κάνει. Μερικές δοκιμές εκτελούνται σταδιακά με το χέρι, ενώ σε άλλες περιπτώσεις ένας υπολογιστής κάνει τη εκτέλεση έλεγχου, αν και οι ειδικευμένοι μηχανικοί ελέγχου είναι αυτοί που πρέπει να εκτελέσουν την ερμηνεία σχεδίου και αποτελέσματος ελέγχου. Ο ειδικευμένος επαγγελματίας ελέγχου επιτυγχάνει τη βέλτιστη επιστροφή στην επένδυση ελέγχου με το να επιλέξει σοφά τις τεχνικές ελέγχου.

Στατικές τεχνικές (δεν απαιτούν την εκτέλεση του προγράμματος)

- Στατική ανάλυση (π.χ. Java bytecode verification)
- επιθεώρηση (review)
- περιήγηση (walkthrough)
- επισκόπηση (inspection)

Δυναμικές τεχνικές (βασίζονται στην εκτέλεση του προγράμματος)

- Συμβολική εκτέλεση
- Δυναμικός έλεγχος
- έλεγχος μονάδας (unit testing)
- έλεγχος συνένωσης (integration testing)
- έλεγχος συστήματος (system testing)
- έλεγχος αποδοχής (acceptance testing)

Διάφοροι τύποι White Box Testing είναι το Application Programming Interface (API) Testing όπου γίνεται έλεγχος του λογισμικού με την χρήση APIs, το code coverage όπου δημιουργούνται σενάρια ελέγχου τα οποία καλύπτουν συγκεκριμένα κριτήρια για κάλυψη του κώδικα, π.χ. δημιουργία σεναρίων ελέγχου για την κάλυψη όλων των statements στο πρόγραμμα τουλάχιστον μία φορά, Fault Injection όπου εισάγονται συγκεκριμένα λανθασμένα δεδομένα για να ελεγχτεί η ορθή λειτουργία των statements και στατικός έλεγχος ο οποίος έχει την δυνατότητα να ελέγχει όλα τα είδη του στατικού ελέγχου. [41]

Ακόμα με το White Box Testing μπορεί να ελεγχθεί η κάλυψη του κώδικα με ποικίλους τρόπους έτσι ώστε ο έλεγχος να γίνεται στις ενότητες ή σε ολόκληρο τον κώδικα ανάλογα με τις ανάγκες του ελέγχου. Εκτός από το statement coverage που αναφέρθηκε πιο πάνω υπάρχει και το path coverage το οποίο ελέγχει την κάλυψη όλων των δυνατών μονοπατιών από την αρχή του κώδικα, το function coverage που καταγράφει την κάλυψη των συναρτήσεων του κώδικα, το branch coverage το οποίο ελέγχει την κάλυψη των διακλαδώσεων του κώδικα στα σημεία που διαχωρίζονται τα μονοπάτια όπως τα if statements κ.α.[42] Ενώ η τεχνική path coverage προϋποθέτει πως ο ελεγκτής πρέπει να γνωρίζει τις εσωτερικές λειτουργίες του προγράμματος, δηλαδή να γνωρίζει τη λειτουργία του καθώς και τον πηγαίο κώδικα. Οι δοκιμές ελέγχου σχεδιάζονται βάση της δομής του

προγράμματος. Λόγω του ότι τα δεδομένα ελέγχου όπως αναφέρθηκε και πιο πάνω λαμβάνονται βάση του πηγαίου κώδικα, υπάρχει μεγαλύτερη πιθανότητα εντοπισμού λαθών.

Οι τεχνικές black-box δεν ασχολούνται με την εσωτερική δομή του προγράμματος, αλλά ελέγχουν μόνο τη συμπεριφορά του και το αν οι υπηρεσίες που παρέχει είναι σύμφωνες με τις προδιαγραφές που έχουν τεθεί. Σε αυτή τη κατηγορία ανήκουν :

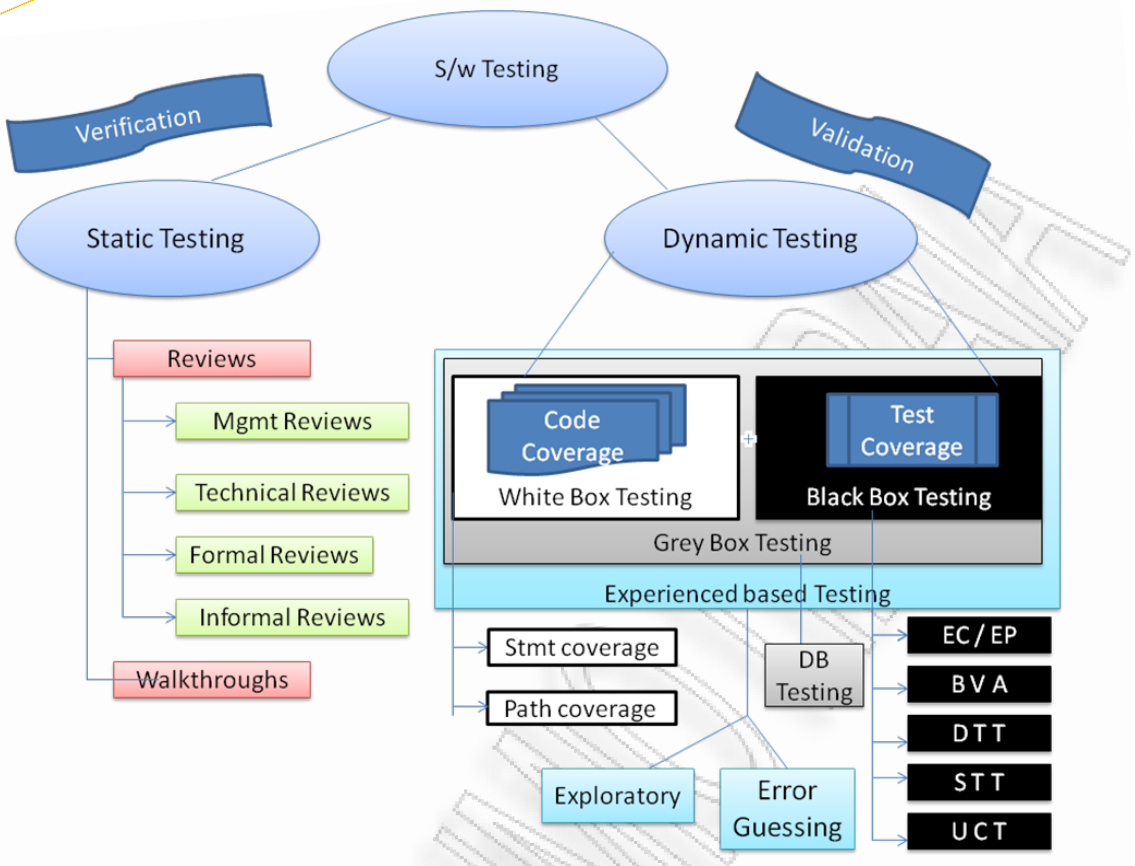
***Functional testing*:** όπου γίνεται έλεγχος σωστής λειτουργίας για κάθε function του προγράμματος, για το αν δηλαδή δίνει τα αναμενόμενα αποτελέσματα σύμφωνα με τις προδιαγραφές.

Έλεγχος ακραίων τιμών *(External Values Testing)*: όπου ελέγχονται όλες οι ακραίες τιμές που λαμβάνουν τα δεδομένα, όπου και είναι πιθανότερη η εμφάνιση σφαλμάτων.

Τυχαίος Έλεγχος *(Random Testing)* : όπου το πρόγραμμα εκτελείται με δεδομένα προερχόμενα από τυχαίες (ή ψευδοτυχαίες) διαδικασίες. Οι τεχνικές white-box λαμβάνουν υπόψη την εσωτερική δομή του προγράμματος σε αντίθεση με την προηγούμενη κατηγορία τεχνικών πράγμα που τις κατατάσσει κάτω από τη γενικότερη κατηγορία του δομημένου ελέγχου (structural testing).

Οι τεχνικές αυτού του είδους περιλαμβάνουν :

Ανάλυση ανωμαλιών (Anomaly Analysis): όπου το λογισμικό εξετάζεται για τυχόν ανωμαλίες, όπως αχρησιμοποίητες μεταβλητές (unused variables) ή τμήματα κώδικα που δεν εκτελούνται ποτέ (unreachable code). Έλεγχο κατευθυνόμενης ροής εκτέλεσης *(Control-Flow Driven Testing*): όπου τα δεδομένα εισόδου επιλέγονται έτσι ώστε να εκτελεστεί ένα συγκεκριμένο μονοπάτι του προγράμματος. Με αυτό τον τρόπο μπορεί να επιτευχθεί πλήρης κάλυψη εντολών ή αύξηση οποιουδήποτε μέτρου κάλυψης κώδικα επιθυμείται και στο βαθμό που επιθυμείται.



Σχήμα 3 «Τεχνικές σχεδιασμού ελέγχου.»

(Πηγή: Aditya ,Engineering College *Software Testing Methodologies*, http://www.mcr.org.in/sureshmudunuri/stm/)

Το παραπάνω διάγραμμα παρουσιάζει τις διάφορες τεχνικές σχεδιασμού ελέγχου που ακολουθούνται και την διάρκεια του ελέγχου λογισμικού.

4.7. Ομάδα Ελέγχου

Οι προγραμματιστές του λογισμικού έχουν ως κύριες ευθύνες τον έλεγχο της λειτουργίας και επίδοσης του συστήματος ενώ ο πελάτης παίζει ένα σημαντικό ρόλο στους ελέγχους αποδοχής και εγκατάστασης. Ωστόσο, η ομάδα που συμμετέχει σε όλους τους ελέγχους σχηματίζεται και από τις δυο κατηγορίες. Συχνά δεν συμμετέχουν στον έλεγχο προγραμματιστές που αναπτύσσουν το σύστημα είναι πολύ εξοικειωμένοι με τη δομή και το σκοπό της υλοποίησης και μπορεί να δυσκολεύονται να αναγνωρίσουν τις διαφορές μεταξύ υλοποίησης και απαιτούμενων λειτουργιών ή επίδοση του συστήματος.

Συνεπώς η ομάδα ελέγχου[43] συνήθως είναι ανεξάρτητη από το προσωπικό υλοποίησης. Στην ιδανική περίπτωση, μερικά μέλη της ομάδας ελέγχου έχουν ήδη εμπειρία ως ελεγκτές συστημάτων. Συνήθως αυτοί οι «επαγγελματίες ελεγκτές» έχουν εργαστεί ως αναλυτές, προγραμματιστές και σχεδιαστές, ενώ τώρα αφιερώνουν όλο το χρόνο τους στον έλεγχο των συστημάτων. Οι ελεγκτές είναι εξοικειωμένοι όχι μόνο με τις προδιαγραφές του συστήματος αλλά και με μεθόδους και εργαλεία ελέγχου.

Οι επαγγελματίες ελεγκτές (professional testers) οργανώνουν και εκτελούν τους ελεγκτές. Συμμετέχουν από την αρχή, σχεδιάζοντας πλανά ελέγχου και περιπτώσεις ελέγχου καθώς το έργο εξελίσσεται. Οι επαγγελματίες ελεγκτές εργάζονται σε συνεργασία με την ομάδα διαχείρισης σχηματισμών για να παρέχουν έγγραφα τεκμηρίωσης και άλλους μηχανισμούς ώστε να αντιστοιχίζουν τους ελέγχους στις απαιτήσεις του συστήματος , στα συστατικά του σχεδίου και στον κώδικα.

Οι επαγγελματίες ελεγκτές εστιάζουν στην ανάπτυξη των ελέγχων , και των μεθόδων και διαδικασιών ελέγχου. Επειδή οι ελεγκτές μπορεί να μην είναι τόσο πεπειραμένοι στις λεπτομέρειες των απαιτήσεων όσο είναι αυτοί που τις έγραψαν , η ομάδα ελέγχου περιλαμβάνει επιπρόσθετα, άτομα τα οποία είναι εξοικειωμένα με τις απαιτήσεις. Οι αναλυτές (analysts) που συμμετείχαν στον καθορισμό των αρχικών απαιτήσεων και προδιαγραφών είναι χρήσιμοι στον έλεγχο επειδή καταλαβαίνουν το πρόβλημα όπως καθορίζεται από τον πελάτη. Μεγάλο μέρος της διαδικασίας ελέγχου του συστήματος περιλαμβάνει τη σύγκριση του νέου συστήματος με τις αρχικές απαιτήσεις, και οι αναλυτές κατέχουν καλή γνώση των αναγκών και των στόχων του πελάτη. Εφόσον έχουν εργαστεί μαζί με τους σχεδιαστές για να σχηματίσουν μια λύση, οι αναλυτές έχουν κάποια ιδέα του τρόπου με τον οποίο θα πρέπει να δουλεύει το σύστημα για την επίλυση του προβλήματος.

Οι αποτελεσματικές αναθεωρήσεις περιλαμβάνουν τους σωστούς ανθρώπους. Οι εμπειρογνώμονες επιχειρησιακών περιοχών πρέπει να παρευρεθούν στις αναθεωρήσεις απαιτήσεων, οι αρχιτέκτονες συστημάτων πρέπει να παρευρεθούν στις αναθεωρήσεις σχεδίου, και οι ειδικοί προγραμματιστές πρέπει να παρευρεθούν στις αναθεωρήσεις κώδικα. Οι ελεγκτές, είναι επίσης χρήσιμοι συμμετέχοντες, επειδή είναι καλοί στην επισήμανση των ασυνεπειών, της ασάφειας, των λεπτομερειών που είναι ελλιπείς, και άλλων. Εντούτοις, από τους ελεγκτές που παρευρίσκονται σε συναντήσεις αναθεώρησης χρειάζεται να παρουσιαστεί η απαραίτητη γνώση της επιχειρησιακής περιοχής, της αρχιτεκτονικής συστημάτων, και προγραμματισμού σε κάθε αναθεώρηση. Και ο καθένας που παρευρίσκεται σε μια αναθεώρηση, πέρασμα, επιθεώρηση, ή όπως καλείται μέσα την οργάνωσή σας πρέπει να καταλάβει τους βασικούς κανόνες τέτοιων εκδηλώσεων.

Έλεγχος Λογισμικού

Επειδή οι έλεγχοι και οι περιπτώσεις ελέγχων είναι στενά συνδεδεμένοι με τις απαιτήσεις και το σχέδιο, στην ομάδα ελέγχου υπάρχει ένα αντιπρόσωπος διαχείρισης σχηματισμών (configuration management representative). Καθώς το σύστημα αποτυγχάνει και απαιτούνται αλλαγές, ο ειδικός διαχείρισης σχηματισμών κανονίζει τις αλλαγές, οι οποίες θα πρέπει να αντικατοπτρίζονται στα έγγραφα τεκμηρίωσης , στις απαιτήσεις, στο σχέδιο, στον κώδικα ή σε άλλο προϊόν της ανάπτυξης του λογισμικού. Στην πράξη οι αλλαγές για τη διόρθωση ενός σφάλματος μπορεί να καταλήξουν σε τροποποιήσεις άλλων περιπτώσεων ελέγχου ή ενός μεγάλου μέρους του πλάνου ελέγχου. Ο ειδικός διαχείρισης σχηματισμών υλοποιεί αυτές τις αλλαγές και συντονίζει την αναθεώρηση των ελέγχων.

Οι σχεδιαστές συστήματος (system designers) διασαφηνίζουν στην ομάδα ελέγχου την προοπτική και το σκοπό του σχεδίου. Οι σχεδιαστές καταλαβαίνουν την προτεινόμενη λύση, καθώς επίσης και τους περιορισμούς που παρουσιάζει αυτή η λύση. Επίσης γνωρίζουν τον τρόπο με τον οποίο το σύστημα χωρίζεται σε λειτουργικά υποσυστήματα ή υποσυστήματα που σχετίζονται με διαφορετικά δεδομένα , καθώς επίσης και πως αναμένεται να λειτουργεί το σύστημα. Καθώς σχεδιάζει περιπτώσεις και διασφαλίζει την κάλυψη των ελέγχων , η ομάδα ελέγχου καλεί τους σχεδιαστές να τη βοηθήσουν να καταγράψει όλες τις δυνατές περιπτώσεις που πρέπει να αντιμετωπισθούν.

Τέλος, η ομάδα ελέγχου περιλαμβάνει χρήστες (users). Οι χρήστες έχουν τα καλύτερα προσόντα για να εκτιμήσουν ορισμένα θέματα , όπως πχ εάν το λογισμικό είναι κατάλληλο για τους χρήστες του εύκολο στη χρήση και άλλους ανθρώπινους παράγοντες. Μερικές φορές οι χρήστες δεν συμμετέχουν αρκετά στα πρώτα στάδια ανάπτυξης λογισμικού. Οι αντιπρόσωποι των πελατών που συμμετέχουν κατά τη διάρκεια της ανάλυσης απαιτήσεων μπορεί να μην σκοπεύουν να χρησιμοποιήσουν οι ίδιοι αυτό το σύστημα, αλλά να έχουν καθήκοντα που να σχετίζονται με αυτά των ατόμων που θα χρησιμοποιήσουν το λογισμικό. Για παράδειγμα οι αντιπρόσωποι μπορεί να είναι διευθυντικά στελέχη αυτών που θα χρησιμοποιήσουν το σύστημα ή να είναι τεχνικοί αντιπρόσωποι , οι οποίοι σχετίζεται έμμεσα με τη δουλειά τους. Εντούτοις, αυτοί οι αντιπρόσωποι μπορεί να είναι τόσο απομακρυσμένοι από το πραγματικό πρόβλημα, ώστε η περιγραφή των απαιτήσεων να είναι ανακριβής ή ελλιπής. Ο πελάτης μπορεί να μην είναι ενήμερος για την ανάγκη επαναπροσδιορισμού ή προσθήκης απαιτήσεων.

Για το λόγο αυτό , οι χρήστες του συστήματος είναι απαραίτητοι, ιδιαίτερα μάλιστα αν δεν ήταν παρόντες όταν καθορίστηκαν για πρώτη φόρα οι απαιτήσεις του συστήματος. Ένας χρήστης είναι πιθανό να είναι εξοικειωμένος με το πρόβλημα επειδή το αντιμετωπίζει καθημερινά ή έστω συχνά, οπότε αυτός ο χρήστης είναι

πολύτιμος στην αποτίμηση του συστήματος και στην επαλήθευση ότι το παραγόμενο σύστημα επιλύει το πρόβλημα.

Ακόμα και όταν έχει γίνει ανάπτυξη ενός συστήματος με μη εγωιστική προσέγγιση, μερικές φορές υπάρχει δυσκολία στο να απομακρύνουν οι χρήστες τα προσωπικά τους αισθήματα από τη διαδικασία ελέγχου. Συνεπώς, χρησιμοποιείται συχνά μια ανεξάρτητη ομάδα ελέγχου για να εξετάσει το σύστημα. Καταυτό τον τρόπο αποφεύγεται η σύγκρουση ανάμεσα στις προσωπικές ευθύνες για τα σφάλματα και την ανάγκη να ανακαλυφθούν όσο το δυνατόν περισσότερα σφάλματα. Για μερικά αντικείμενα υπό έλεγχο, είναι αδύνατο για την ομάδα ελέγχου να δημιουργήσει ένα σύνολο αντιπροσωπευτικών περιπτώσεων ελέγχου που να επιδεικνύουν τη σωστή λειτουργία του συστήματος για όλες τις δυνατές περιπτώσεις.

Με άλλα λόγια, όσο μεγαλύτερη εμπειρία διαθέτουν τα άτομα που είναι επιφορτισμένα με την ευθύνη διεξαγωγής του ελέγχου του λογισμικού, ιδιαίτερα μάλιστα σε παρόμοια με το υπό έλεγχο λογισμικά, τόσο υψηλότερη είναι και η πιθανότητα εύρεσης μεγαλύτερου ποσοστού σφαλμάτων, πράγμα που συνεπάγεται τη μείωση του ποσοστού της εναπομείνασας επικινδυνότητας. Η εμπειρία της ομάδας ελέγχου, μπορεί και αυτή να καταταχθεί σε διάφορες βαθμίδες, με μονάδα μέτρησης το χρόνο ως εξής :

Κατάταξη	Πολύ χαμηλή	Χαμηλή	Ονομαστική	Υψηλή	Πολύ υψηλή
Εμπειρία	≤ 4 Μήνες	> 4 μήνες και ≤ 1 Χρόνο	> 1 και ≤ 3 χρόνια	> 3 και ≤ 6 Χρόνια	> 6 χρόνια

Πίνακας 1 Κατάταξη του παράγοντα "Εμπειρία Ομάδας Ελέγχου"

Βέβαια, πρέπει να σημειωθεί ότι η εμπειρία δεν συνεπάγεται απαραιτήτως και ικανότητα, αλλά όπως και να έχει πληροφορίες που αφορούν την εμπειρία ή ακόμα και το εκπαιδευτικό επίπεδο της ομάδας ελέγχου, αποτελούν συχνά ένα σημαντικό στοιχείο ένδειξης του κατά πόσο τα άτομα που είναι επιφορτισμένα με την ευθύνη του ελέγχου, μπορούν να φέρουν εις πέρας την αποστολή τους με επιτυχία.

Οι ελεγκτές είναι «εύκαμπτοι» κυρίως επειδή οι επαγγελματίες ελεγκτές είναι συνήθως μη ειδικευμένοι σε κάτι και έχουν μια γενική εικόνα. Στον έλεγχο επιτρέπεται να δει κανείς ολόκληρη την εικόνα και αυτός είναι ο λόγος για τον οποίο μερικοί άνθρωποι επιλέγουν να είναι ελεγκτές παρά προγραμματιστές. Επιπλέον, πέρα από τις δεξιότητες επιχειρησιακών περιοχών, οι πραγματικοί επαγγελματίες ελέγχου έχουν και δεξιότητες ελέγχου. Ξέρουν πώς να

χρησιμοποιήσουν τα αυτοματοποιημένα εργαλεία και πώς να εφαρμόσουν τον διερευνητικό έλεγχο. Ξέρουν πώς να αναλύσουν ισοδύναμα τα μέρη, τις μεταβολές στην κατάσταση ,τις κινήσεις και στη συνέχεια τη μετάφραση της ανάλυσης σε αποτελεσματικές και αποδοτικές αυτοματοποιημένες και μη περιπτώσεις ελέγχου. Μπορούν επίσης να «ξεστήσουν» ένα σύστημα και να υπολογίσουν τα λάθη που μπορεί να υπάρχουν. (Αυτές είναι δεξιότητες που ένας τυπικός χρήστης δεν έχει, αυτός είναι ο λόγος για τον οποίο οι συμπεριφοριστικές ομάδες ελέγχου που αποτελούνται μόνο από τους χρήστες, χωρίς οποιουσδήποτε έμπειρους και εκπαιδευμένους ελεγκτές, τείνουν να κάνουν μια σχετικά φτωχή εργασία στην εύρεση λαθών).

Επιπρόσθετα, πολλοί άλλοι παράγοντες δικαιολογούν την ύπαρξη ανεξάρτητης ομάδας ελέγχου. Πρώτον, μπορεί κατά λάθος να εισάγονται σφάλματα κατά τη διάρκεια ερμηνείας του σχεδίου ,όταν καθορίζεται η λογική του προγράμματος κατά τη διάρκεια συγγραφής της αναλυτικής τεκμηρίωσης ,ή όταν υλοποιούνται οι αλγόριθμοι. Μια ανεξάρτητη ομάδα ελεγκτών μπορεί να συμμετάσχει στην αναθεώρηση των συστατικών κατά τη διάρκεια της ανάπτυξης. Η ομάδα μπορεί να αποτελεί μέρος της αναθεώρησης των απαιτήσεων και σχεδίασης, μπορεί να ελέγξει τα συστατικά κώδικα ένα-ένα και μπορεί να ελέγξει το σύστημα καθώς ολοκληρώνεται και παρουσιάζεται στους πελάτες για αποδοχή. Καταυτό τον τρόπο ο έλεγχος μπορεί να προχωρά ταυτόχρονα με τη συγγραφή κώδικα η ομάδα ελέγχων μπορεί να ελέγχει τα συστατικά καθώς ολοκληρώνονται και να ξεκινήσει να τα ενώνει καθώς η ομάδα προγραμματισμού συνεχίζει να φτιάχνει τον κώδικα άλλων συστατικών του συστήματος.

5. Επίπεδα ελέγχου λογισμικού

Όταν ολοκληρωμένα συστήματα ή πρόσθετες ενότητες επεκτείνονται, εκτελείται ο έλεγχος των συστημάτων. Ο έλεγχος συστημάτων εξετάζει θέματα όπως η πληρότητα, η απόδοση του συστήματος στην περίπτωση υπερφόρτωσης εφαρμογών, η ασφάλεια, η ολοκλήρωση στο περιβάλλον του συστήματος και ούτω καθεξής. Όταν πρόκειται να εκτελεστούν οι έλεγχοι συστημάτων, γίνεται συχνά η ανάθεση σε μια ειδική ομάδα ελέγχου όπως προαναφέρθηκε και πιο πάνω.

Οι λίστες ελέγχου συστημάτων περιλαμβάνουν τις εξής ερωτήσεις:

- Λειτουργική πληρότητα του συστήματος ή της πρόσθετης ενότητας - Συμπεριφορά χρόνου εκτέλεσης στα διάφορα λειτουργικά συστήματα ή στις διαφορετικές διαμορφώσεις υλικού.
- Ικανότητα εγκατάστασης (Installability) και διαμόρφωσης (configurability) στα διάφορα συστήματα
- Όρια δυναμικότητας (μέγιστο μέγεθος αρχείων, αριθμός αρχείων, μέγιστος αριθμός ταυτόχρονων χρηστών, κ.λπ.)
- Συμπεριφορά απόκρισης σε προβλήματα του περιβάλλοντος προγραμματισμού (συντριβή συστήματος, μη διαθέσιμο δίκτυο, πλήρης σκληρός δίσκος, μη έτοιμος εκτυπωτής)
- Προστασία ενάντια σε αναρμόδια πρόσβαση σε δεδομένα και προγράμματα.

Αρχικά, κάθε συστατικό του προγράμματος ελέγχεται ξεχωριστά, απομονωμένο από τα άλλα συστατικά του συστήματος. Αυτό το είδος ελέγχου που είναι γνωστή ως έλεγχος υπομονάδων (module testing), έλεγχος συστατικών (component testing), ή έλεγχος μονάδων (unit testing)επαληθεύει ότι το συστατικό λειτουργεί σωστά όταν εισάγονται οι αναμενόμενοι τύποι εισόδων (σύμφωνα με αυτό που αναμένεται από το σχέδιο του συστατικού). **Ο έλεγχος μονάδων** εκτελείται σε ένα ελεγχόμενο περιβάλλον όταν είναι δυνατόν, ώστε η μονάδα ελέγχου να μπορεί να εισάγει ένα προκαθορισμένο σύνολο δεδομένων στο συστατικό που ελέγχεται και να παρακολουθήσει τις ενέργειες και τα δεδομένα εξόδου που παράγονται. Επιπρόσθετα, η ομάδα ελέγχου εξετάζει τις εσωτερικές δομές δεδομένων, τη λογική και τις συνοριακές συνθήκες για τα δεδομένα εισόδου και εξόδου. (από βιβλίο)

Όταν οι ομάδες συστατικών περάσουν από έλεγχο μονάδων, το επόμενο βήμα είναι να εξασφαλιστεί ότι οι διασυνδέσεις ανάμεσα στα συστατικά ορίζονται και η διαχείρισή τους γίνεται κατάλληλα. **Ο έλεγχος ολοκλήρωσης** (integration testing) είναι η διαδικασία κατά την οποία επαληθεύεται ότι τα συστατικά λειτουργούν μαζί όπως περιγράφεται στις προδιαγραφές σχεδίου συστήματος και σχεδίου προγράμματος.

Έλεγχος Λογισμικού

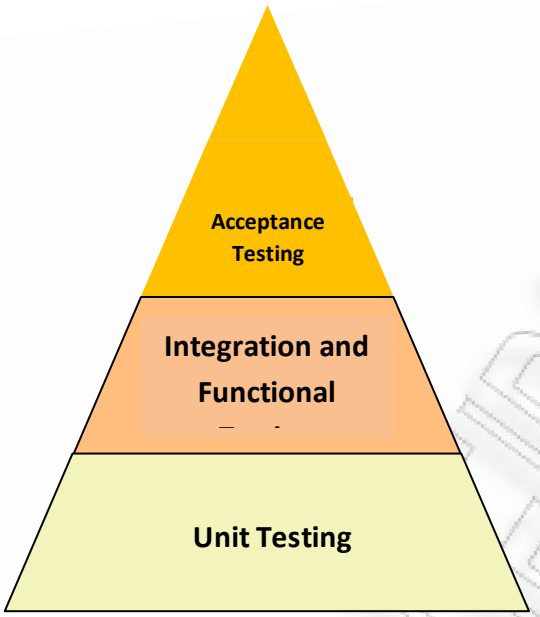
Μόλις βεβαιωθούμε ότι η πληροφορία μεταφέρεται μεταξύ των συστατικών σύμφωνα με το σχέδιο, ελέγχουμε το σύστημα για να επαληθεύσουμε ότι παρουσιάζει την επιθυμητή λειτουργία. **Ο έλεγχος λειτουργιών** (function test) εφαρμόζεται για την αξιολόγηση του συστήματος ώστε να καθοριστεί κατά πόσον οι λειτουργίες περιγράφονται στις προδιαγραφές των απαιτήσεων εκτελούνται σωστά από το ολοκληρωμένο σύστημα. Ως αποτέλεσμα λαμβάνουμε ένα σύστημα που θεωρούμε ότι λειτουργεί κανονικά.

Οι απαιτήσεις τεκμηριώνονται με δύο τρόπους: πρώτα στην ορολογία του πελάτη και έπειτα σαν σύνολο απαιτήσεων λογισμικού και υλικού που μπορούν να χρησιμοποιούν οι τεχνολόγοι λογισμικού. Ό έλεγχος λειτουργιών συγκρίνει το σύστημα που κατασκευάζεται με τις λειτουργίες που περιγράφονται στις προδιαγραφές απαιτήσεων του κατασκευαστή λογισμικού.

Στη συνέχεια **ο έλεγχος επίδοσης** (performance testing) συγκρίνει το σύστημα με τις υπόλοιπες απαιτήσεις λογισμικού και υλικού. Όταν ο έλεγχος εκτελείται με επιτυχία στο πραγματικό περιβάλλον εργασίας ενός πελάτη ,παράγει ένα επικυρωμένο σύστημα (validated system).

Όταν ολοκληρωθεί ο έλεγχος επίδοσης, οι κατασκευαστές είναι σίγουροι ότι λειτουργεί σύμφωνα με την κατανόηση της περιγραφής του συστήματος που έχει γίνει. Το επόμενο βήμα είναι η συνεργασία με τον πελάτη ώστε να επαληθεύσουμε ότι το σύστημα λειτουργεί και σύμφωνα με τις δικές του προσδοκίες.

Συνεργαζόμαστε με τον πελάτη για να εκτελέσουμε τον **έλεγχο αποδοχής** (acceptance testing), όπου το σύστημα εξετάζεται έναντι της περιγραφής απαιτήσεων που έχει δώσει ο πελάτης. Με την ολοκλήρωση του ελέγχου αποδοχής, το σύστημα που έγινε αποδεκτό εγκαθίσταται στο περιβάλλον στο οποίο θα λειτουργήσει πραγματικά ο τελικός **έλεγχος εγκατάστασης** (installation test) εκτελείται για να βεβαιωθούμε ότι το σύστημα εξακολουθεί να λειτουργεί όπως αναμένεται.



Διάγραμμα 2 Επίπεδα ελέγχου λογισμικού

5.1. Έλεγχος μονάδων

Η έλεγχος μονάδων πραγματοποιείται πρώτιστα από τους ίδιους τους προγραμματιστές και αφορά τις διορθώσεις της λειτουργικότητας του συστήματος και την πληρότητα των μεμονωμένων μονάδων του, π.χ. των λειτουργιών, των διαδικασιών ή των μεθόδων κλάσεων, και των κλάσεων ή των τμημάτων του συστήματος. Οι μονάδες μπορούν να είναι οποιουδήποτε μεγέθους. Παράγονται συνεχώς κατά τη διάρκεια ενός έργου, που σημαίνει ότι τα ολοκληρωμένα ξεχωριστά τμήματα του προγράμματος θα είναι συνήθως διαθέσιμα για τον έλεγχο μονάδων. Στο τμήμα της Πληροφορικής μερικών οργανισμών, ένα ολοκληρωμένο πρόγραμμα είναι επίσης μια μονάδα. Η έλεγχος μονάδων δεν είναι μια φάση του προγράμματος. Είναι μια προοπτική από την οποία αξιολογούνται και ελέγχονται τα τμήματα λογισμικού. Ο έλεγχος μονάδων ενδιαφέρεται για τις απαιτήσεις μονάδων. Δεν εξετάζει το πώς οι μονάδες αλληλεπιδρούν, αυτό αποτελεί ολόκληρο ένα άλλο θέμα ελέγχου ούτε ελέγχει το πώς η μονάδα συναρμόζεται μέσα στο περιβάλλον συστημάτων ή το πώς αλληλεπιδρά με το σύστημα, αυτό είναι η επαρχία του ελέγχου των συστημάτων.

Αρχικά, εξετάζεται ο κώδικας διαβάζοντας τον, γίνεται προσπάθεια να εντοπιστούν αλγοριθμικά σφάλματα, σφάλματα στα δεδομένα , ή συντακτικά σφάλματα. Υπάρχει δυνατότητα ακόμα και να συγκριθεί ο κώδικας με τις προδιαγραφές και με το σχέδιο για να εξασφαλιστεί ότι έχουν ληφθεί υπόψη όλες οι σχετικές περιπτώσεις. Στη συνέχεια, ο κώδικας μεταγλωττίζεται και εξαλείφονται τα συντακτικά σφάλματα που έχουν απομείνει. Τέλος, δημιουργούνται περιπτώσεις

ελέγχου (test cases) για να φανεί ότι η είσοδος μετατρέπεται κατάλληλα στην επιθυμητή έξοδο.

5.1.1. Επιλογή περιπτώσεων ελέγχου

Για τον έλεγχο ενός συστατικού επιλέγονται δεδομένα εισόδου και συνθήκες, το συστατικό διαχειρίζεται τα δεδομένα, και παρατηρείται η έξοδος. Ένα σημείο ελέγχου (test point) ή περίπτωση ελέγχου (test case) είναι μια συγκεκριμένη επιλογή δεδομένων εισόδου που θα χρησιμοποιηθούν για τον έλεγχο ενός προγράμματος.

Πώς επιλέγονται οι περιπτώσεις ελέγχου και καθορίζονται οι έλεγχοι ;

Αρχικά καθορίζονται οι στόχοι του ελέγχου. Στη συνέχεια, επιλέγονται οι περιπτώσεις ελέγχου και καθορίζεται ένας έλεγχος που θα ικανοποιεί κάποιο συγκεκριμένο σκοπό. Ένας σκοπός θα μπορούσε να είναι η επίδειξη ότι όλες οι εντολές εκτελούνται χωρίς προβλήματα. Ένας άλλος σκοπός θα μπορούσε να είναι η επίδειξη ότι κάθε λειτουργία εκτελείται σωστά από τον κώδικα. Οι στόχοι καθορίζουν τον τρόπο με τον οποίο επιλέγεται η είσοδος ώστε να επιλεχθούν οι περιπτώσεις ελέγχου.

Ο κώδικας μπορεί να εξεταστεί είτε ως κλειστό είτε ως ανοικτό κουτί, αναλόγως με τους στόχους του ελέγχου. *Αν είναι κλειστό κουτί*, τροφοδοτείται με όλες τις δυνατές εισόδους και συγκρίνεται η έξοδος με εκείνη που αναμένεται σύμφωνα με τις απαιτήσεις.

Ο έλεγχος κλειστού κουτιού υποφέρει από την αβεβαιότητα σχετικά με το πόσο οι περιπτώσεις ελέγχου που έχουν επιλεγεί θα οδηγήσουν στον εντοπισμό κάποιου συγκεκριμένου σφάλματος. Απο την άλλη μεριά ο έλεγχος ανοικτού κουτιού εμπεριέχει πάντοτε τον κίνδυνο να δοθεί μεγάλη προσοχή στις εσωτερικές διεργασίες του κώδικα. Δηλαδή να καταλήξει ο χρήστης να ελέγχει τι κάνει το πρόγραμμα αντί να ελέγχει τι θα έπρεπε να κάνει.

Μπορούμε να συνδυαστούν οι *έλεγχοι ανοικτού και κλειστού κουτιού* για την παραγωγή δεδομένων ελέγχου. Πρώτον, θεωρώντας ότι το πρόγραμμα είναι ένα κλειστό κουτί, μπορούν να χρησιμοποιηθούν οι εξωτερικές προδιαγραφές του προγράμματος για την παραγωγή των αρχικών περιπτώσεων ελέγχου. Αυτές οι περιπτώσεις θα πρέπει να περιλαμβάνουν όχι μόνο τα αναμενόμενα δεδομένα εισόδου, αλλά και τις οριακές συνθήκες για τις εισόδους και τις εξόδους καθώς επίσης και πολλές περιπτώσεις μη έγκυρων δεδομένων.

5.2. Λειτουργικός έλεγχος

Οι περιπτώσεις ελέγχου μονάδων καλύπτουν τις λειτουργικές διορθώσεις και την πληρότητα της μονάδας καθώς επίσης και του κατάλληλου χειρισμού του λάθους, του ελέγχου των τιμών εισαγωγής (εισαγωγή παραμέτρων), της ακρίβειας των δεδομένων εξόδου (επιστρεφόμενες τιμές) και την απόδοση του συστήματος[44]. Αυτοί οι τύποι ελέγχου εμπίπτουν στην κατηγορία του ελέγχου του γκρι κουτιού, που σημαίνει ότι καλύπτουν τη λειτουργία μονάδων (η «μαύρη» πτυχή). Εντούτοις, προκειμένου να βρεθούν και να αξιολογηθούν οι περιπτώσεις ελέγχου, μπορεί να προσεγγιστεί ο πηγαίος κώδικας (η «άσπρη» πτυχή).

Οι συναρτήσεις επιστρέφουν μια τιμή. Επομένως, είναι ευκολότερο να ελεγχθούν από την μονάδα ελέγχου. Οι περιπτώσεις ελέγχου προέρχονται από το συνδυασμό όλων των παραμέτρων εισαγωγής. Οι παράμετροι εισαγωγής περιλαμβάνουν οτιδήποτε ασκεί επίδραση στην παραγωγή μιας λειτουργίας. Αυτό σημαίνει τις παραμέτρους που περνούν όταν η λειτουργία καλείται, αλλά και παράγοντες όπως οι ρυθμίσεις συστημάτων (μορφή ημερομηνίας, δεκαδικός διαχωριστής, δικαιώματα χρηστών, κ.λπ.). Κατά την επινόηση χρήσιμων περιπτώσεων ελέγχου, γίνεται η ανακάλυψη όλων των παραγόντων που ασκούν επιρροή. Σε γενικές γραμμές, θα μπορούσε να υπολογιστεί το αναμενόμενο αποτέλεσμα για οποιοδήποτε συνδυασμό παραμέτρων εισαγωγής. Η επιστρεφόμενη τιμή μιας λειτουργίας με έναν δεδομένο συνδυασμό παραμέτρων εισαγωγής είναι το πραγματικό αποτέλεσμα λειτουργικότητας και αυτό θα πρέπει να ταιριάξει με το αναμενόμενο αποτέλεσμα.

Οποιοσδήποτε γρήγορος έλεγχος στην αρχή της λειτουργίας ενός συστήματος θα ήταν σίγουρα καλό. Η ανακάλυψη των ελάχιστων ή των μέγιστων τιμών ορισμένων παραμέτρων εισαγωγής θα πρέπει να οδηγήσουν το χρήστη να αναρωτηθεί αν θα μπορούσαν να οδηγήσουν σε προβλήματα. Εάν προκύψει μια κρίσιμη κατάσταση κατά τον έλεγχο της εισαγωγής της αντίστοιχης ελάχιστης ή μέγιστης τιμής ελέγχου θα πρέπει να ελεγχθεί. Αυτό δεν χρειάζεται απαραίτητα να πάρει διαστάσεις πλήρους ελέγχου. Μια ματιά στον κώδικα αρκεί πολλές φορές για να καταλάβει κανείς που είναι το πρόβλημα. Το σημαντικότερο είναι να σκεφτεί κανείς τον έλεγχο του πηγαίου κώδικα έχοντας στο μυαλό του την συγκεκριμένη ερώτηση. Η σκέψη από την άποψη των ελάχιστων και μέγιστων περιπτώσεων βοηθάει. Επίσης πρέπει να σιγουρευτεί ο χρήστης ότι η λειτουργικότητα του συστήματος αποκρίνεται με σταθερό τρόπο στις περιπτώσεις λάθους. Η εμπειροτεχνία μέθοδος λέει πως: Εάν κάτι δεν είναι προφανές, αξίζει τον έλεγχο. Το «Προφανές» θα πρέπει να σημαίνει ακριβώς αυτό. Εάν μια ιδιαίτερη συμπεριφορά λειτουργίας δεν είναι αμέσως σαφής

από τον πηγαίο κώδικα και θα πρέπει να βασανίσει το μυαλό το χρήστη για το πώς να συμπεριφερθεί μια λειτουργία σε μια αρκετά πιο σύνθετη κατάσταση, είναι πάντα ασφαλέστερο να εξεταστεί αυτή η συμπεριφορά από το να ονειρευτεί κανείς μια λύση. Το καλύτερο εργαλείο για αυτό είναι ένας διορθωτής πηγαίου κώδικα. Εάν παρατηρήσει ο χρήστης το πώς ο υπολογιστής προχωρά σταδιακά κατά τον έλεγχο μιας λειτουργίας, το πώς ελέγχει περιοδικά τις τιμές για τις τοπικές μεταβλητές και το πως εκτελεί τα κομμάτια του κώδικα βαθμιαία, θα το δει από μια διαφορετική οπτική γωνία όσον αφορά τη λειτουργία από ότι εάν εξεταστεί μόνο από τον χρήστη με ένα απλό πρόγραμμα σύνταξης κειμένου.

5.3. Έλεγχος ολοκλήρωσης

Όταν τέλος τα διάφορα συστατικά λειτουργούν σωστά και ανταποκρίνονται στους στόχους που έχουν τεθεί για το σύστημα, ενοποιούνται σε ένα και μόνο σύστημα. Αυτή η ολοκλήρωση σχεδιάζεται και συντονίζεται έτσι ώστε όταν παρουσιαστεί κάποια δυσλειτουργία ,να υπάρχει κάποια ιδέα του τι προκάλεσε. Επιπρόσθετα, η σειρά με την οποία ελέγχονται τα συστατικά επηρεάζει την επιλογή των περιπτώσεων ελέγχου καθώς και των εργαλείων ελέγχου. Για μεγάλα συστήματα μερικά συστατικά μπορεί να βρίσκονται στη φάση συγγραφής κώδικα, άλλα μπορεί να είναι στη φάση ελέγχου μονάδων ,και ακόμα μερικές άλλες ομάδες συστατικών να ελέγχονται μαζί. Η στρατηγική ελέγχου που ακολουθούμε εξηγεί γιατί και πως τα συστατικά συνδυάζονται για τον έλεγχο του ολοκληρωμένου συστήματος. Αυτή η στρατηγική επηρεάζει όχι μόνο το χρόνο ολοκλήρωσης και τη σειρά συγγραφής του κώδικα ,αλλά και το κόστος καθώς και την πληρότητα του ελέγχου.

Το σύστημα εξετάζεται ξανά σαν μια ιεραρχία συστατικών, όπου κάθε συστατικό ανήκει σε κάποιο επίπεδο του σχεδίου. Μπορεί να ξεκινήσει κανείς από την κορυφή και να πάει προς τα κάτω , να πάει από κάτω προς τα πάνω ή να χρησιμοποιήσει ένα συνδυασμό των δυο αυτών προσεγγίσεων καθώς το σύστημα ελέγχεται.

5.3.1. Αρχές του ελέγχου συστήματος

Στόχος του ελέγχου μονάδων και του ελέγχου ολοκλήρωσης είναι να εξασφαλιστεί ότι ο κώδικας υλοποιεί σωστά το σχέδιο του λογισμικού αυτό σημαίνει ότι οι προγραμματιστές έγραψαν τον κώδικα για να υλοποιούσουν αυτό που οι σχεδιαστές είχαν σκοπό να πετύχουν. Στον έλεγχο συστήματος υπάρχει ένας διαφορετικό στόχο: να εξασφαλιστεί ότι το σύστημα κάνει αυτό που θέλει να κάνει ο πελάτης. Για να γίνει κατανοητό πως μπορεί να επιτευχθεί αυτός ο στόχος, πρέπει πρώτα να γίνει κατανοητό από πού προέρχονται τα σφάλματα ενός συστήματος.

5.3.2. Διαδικασία Ελέγχου Συστήματος

Υπάρχουν αρκετά βήματα κατά τον έλεγχο ενός συστήματος:

1.Έλεγχος λειτουργιών

2.Έλεγχος επίδοσης

3.Έλεγχος αποδοχής

Εάν διάφορα τμήματα του προγράμματος ομαδοποιηθούν σε ένα ενιαίο τμήμα μπορούν να προκύψουν κίνδυνοι σχετικοί με την ίδια τη διευθέτηση. Αυτά τα ζητήματα εξετάζονται στον έλεγχο ολοκλήρωσης. Ο έλεγχος ολοκλήρωσης είναι διαφορετικός από τον έλεγχο μονάδων της επόμενης μεγαλύτερης μονάδας του προγράμματος. Ο έλεγχος ολοκλήρωσης ενδιαφέρεται περισσότερο για το εάν τα ξεχωριστά τμήματα λειτουργούν αποτελεσματικά μαζί από ότι το τι κάνει το νέο ενιαίο τμήμα που έχει προκύψει. Η αλληλεπίδραση των μεμονωμένων μερών μιας μονάδας ή το υποσύστημα θέτει μεγάλους κινδύνους σχετικά με το εάν τα μέρη είναι φυσικά χωρισμένα, παραδείγματος χάριν σε διαφορετικά .exe και .dll αρχεία. Εάν τα μέρη ενός συστήματος συνδυάζονται σε ένα εκτελέσιμο αρχείο, οι ασυμβατότητες των διεπαφών, τα χαμένα αρχεία και άλλα προβλήματα θα εμφανιστούν αμέσως. Εάν τα μέρη του προγράμματος είναι φυσικά χωρισμένα, ο πρώτος έλεγχος ενός συστατικού θα γίνει μόνο όταν καλείται. Ελέγχεται η σωστή έκδοση του συστήματος, καθώς επίσης και η αντιστοιχία μεταξύ της διεπαφής και της κλήσης της λειτουργίας, και ούτω καθεξής.

5.4. Έλεγχος εγκατάστασης

Αρχικά ελέγχονται οι λειτουργίες που εκτελούνται από το σύστημα. Ξεκινούν με ένα σύνολο συστατικών τα οποία είχαν ελεγχθεί ξεχωριστά και έπειτα όλα μαζί. Ένας έλεγχος (function test) πραγματοποιείται για να επαληθεύσει ότι το ολοκληρωμένο σύστημα εκτελεί τις λειτουργίες του όπως είναι καθορισμένες στις απαιτήσεις του συστήματος.

Για παράδειγμα, ο έλεγχος λειτουργίας ενός πακέτου τραπεζικού λογαριασμού επαληθεύει ότι αυτό ενημερώνει σωστά το υπόλοιπο μετά από μια κατάθεση, δίνει τη δυνατότητα απόσυρσης των χρημάτων, υπολογίζει το επιτόκιο παρουσιάζει το υπόλοιπο του λογαριασμού και άλλες τραπεζικές λειτουργίες.

Μόλις η ομάδα ελέγχου πειστεί ότι οι λειτουργίες δουλεύουν όπως είναι καθορισμένο να δουλεύουν , ο έλεγχος επίδοσης (performance test) συγκρίνει τα ολοκληρωμένα συστήματα με τις μη λειτουργικές απαιτήσεις του συστήματος. Αυτές οι απαιτήσεις, στις οποίες συμπεριλαμβάνονται η ασφάλεια , η ακρίβεια , η ταχύτητα και η αξιοπιστία, περιορίζουν τον τρόπο που εκτελούνται οι λειτουργίες του συστήματος.

Για παράδειγμα, ο έλεγχος επίδοσης ενός πακέτου τραπεζικού λογαριασμού εκτιμά την ταχύτητα με την οποία γίνονται οι υπολογισμοί , την ακρίβεια των υπολογισμών, τα μέτρα ασφαλείας που απαιτούνται και τον χρόνο απόκρισης του συστήματος στα ερωτήματα του χρήστη.

Σε αυτό το σημείο, το σύστημα λειτουργεί με τον τρόπο κατά τον οποίο είχε σχεδιαστεί να λειτουργεί. Επομένως, αυτό το σύστημα ονομάζεται επαληθευμένο σύστημα (verified system). Το σύστημα αυτό εκφράζει την ερμηνεία των προδιαγεγραμμένων απαιτήσεων από το σχεδιαστή. Έπειτα συγκρίνεται το σύστημα με τις προσδοκίες του πελάτη εξετάζοντας τα έγγραφα καθορισμού απαιτήσεων. Εάν μείνουν οι χρήστες ικανοποιημένοι με το σύστημα που κατασκευάστηκε θεωρώντας ότι ανταποκρίνεται στις απαιτήσεις , τότε υπάρχει ένα επικυρωμένο σύστημα (validated system) αυτό σημαίνει ότι έχει επαληθευθεί ότι το σύστημα ικανοποιεί τις προκαθορισμένες απαιτήσεις.

Μέχρι τώρα όλοι οι έλεγχοι έχουν εκτελεστεί από τους κατασκευαστές του συστήματος και βασίζονται στην κατανόηση που έχουν οι κατασκευαστές σχετικά με το σύστημα και τους στόχους του. Επιπλέον , το σύστημα ελέγχεται και από τους πελάτες ,ώστε να βεβαιωθούν ότι ανταποκρίνεται στις ανάγκες και τις απαιτήσεις τους, οι οποίες μπορεί να είναι διαφορετικές από την άποψη που έχουν σχηματίσει

για αυτές οι κατασκευαστές του συστήματος. Ο έλεγχος αυτός, που καλείται έλεγχος αποδοχής (acceptance test), διαβεβαιώνει τους πελάτες ότι το σύστημα που τελικά κατασκευάστηκε είναι κατάλληλο. Ο έλεγχος αποδοχής μερικές φορές εκτελείται στην πραγματική πλατφόρμα λειτουργίας, όμως τις περισσότερες φορές εκτελείται σε πλατφόρμες λειτουργίας διαφορετικές από την πραγματική. Για το λόγο αυτό μπορεί να εκτελεστεί ένας τελικός έλεγχος εγκατάστασης (installation test) ώστε να δοθεί η δυνατότητα στους χρήστες να ελέγξουν τις λειτουργίες του συστήματος και να καταγράψουν επιπρόσθετα προβλήματα που θα εμφανιστούν (τα οποία προέρχονται από το πραγματικό περιβάλλον λειτουργίας).

Για παράδειγμα , ένα σύστημα για χρήση πάνω σε πλοίο μπορεί να σχεδιαστεί, να κατασκευαστεί και να ελεγχθεί στον τόπο κατασκευής του με τις συνθήκες διαμορφωμένες σαν συνθήκες πλοίου, οι οποίες βεβαίως δεν είναι οι πραγματικές συνθήκες λειτουργίας , απλά μια προσομοίωση. Μόλις ολοκληρωθούν οι έλεγχοι στο χώρο του κατασκευαστή , θα πρέπει να πραγματοποιηθεί ένα επιπλέον σύνολο ελέγχων εγκατάστασης με το σύστημα πάνω σε ένα πραγματικό πλοίο (για το οποίο κατασκευάστηκε το σύστημα).

5.5. Έλεγχος βασισμένος στις προδιαγραφές

Ο έλεγχος βάσει των προδιαγραφών έχει ως στόχο τον έλεγχο της λειτουργίας του λογισμικού σύμφωνα με τα τις καταχωρημένες προδιαγραφές. Έτσι ο έλεγχος γίνεται στα δεδομένα εισόδου και τα αποτελέσματα που εξάγονται από το αντικείμενο ελέγχου. Κατά των έλεγχο βάσει των προδιαγραφών συνήθως απαιτείται μεγάλος αριθμός από σενάρια ελέγχου τα όποια μπορούν να παράγουν αποτελέσματα με τα οποία θα γίνει η επαλήθευση και η σύγκριση με τα αναμενόμενα. Ο συγκεκριμένος έλεγχος είναι απαραίτητος αν και σε μερικές περιπτώσεις δεν παρέχει κάλυψη για συγκεκριμένους κινδύνους.

5.6. Έλεγχος ασφάλειας

Ο έλεγχος ασφάλειας λογισμικού επαληθεύει ότι μόνο το εξουσιοδοτημένο προσωπικό μπορεί να έχει πρόσβαση στο πρόγραμμα και μόνο το εξουσιοδοτημένο προσωπικό μπορεί να έχει πρόσβαση σε διαθέσιμες λειτουργίες στο επίπεδο ασφάλειάς τους. Ο έλεγχος ασφάλειας οποιουδήποτε αναπτυγμένου συστήματος (σύστημα υπό ανάπτυξη) είναι το να βρεθούν οι σημαντικότερες «τρύπες» και αδυναμίες του συστήματος που μπορούν να προκαλέσουν σημαντική ζημιά στο σύστημα από εξουσιοδοτημένους χρήστες. Ο έλεγχος ασφάλειας είναι πολύ χρήσιμος για τον ελεγκτή για την εύρεση και τον καθορισμό των προβλημάτων. Εξασφαλίζει ότι το σύστημα θα λειτουργεί για αρκετό χρονικό διάστημα χωρίς

οποιοδήποτε σοβαρό πρόβλημα. Επίσης εξασφαλίζει ότι τα συστήματα που χρησιμοποιούνται από έναν οργανισμό είναι ασφαλή από οποιαδήποτε αναρμόδια επίθεση. Κατά αυτόν τον τρόπο, ο έλεγχος ασφάλειας είναι ωφέλιμος για τον οργανισμό από όπου και αν το δει κανείς.

5.7. Έλεγχος παλινδρόμησης

Ο έλεγχος παλινδρόμησης δεν αποτελεί μια ενιαία φάση εξέτασης. Αντ' αυτού, περιλαμβάνει την επανάληψη των ελέγχων από τις πρώτες τρεις φάσεις για να εξετάσει συγκεκριμένα χαρακτηριστικά γνωρίσματα ή υπογνωρίσματα. Εάν οποιοδήποτε μέρος του προγράμματος τροποποιείται ή ενισχύεται, θα πρέπει να υποβληθεί σε έναν νέο έλεγχο μονάδων από τον προγραμματιστή. Το να περιοριστεί ο έλεγχος στη νέα λειτουργία δεν είναι αρκετό. Ο προγραμματιστής θα πρέπει επίσης να εξασφαλίσει ότι η τροποποίηση δεν παρεμποδίζει την υπάρχουσα λειτουργία. Αυτό μπορεί να επιτευχθεί καλύτερα μέσω της χρήσης των ενσωματωμένων ελέγχων σε σχέση με τα πλαίσια ελέγχου. Κάθε μέρος του προγράμματος που υποτίθεται πως δουλεύει με την τροποποιημένη ή ενισχυμένη μονάδα πρέπει να υποβληθεί σε έλεγχο ολοκλήρωσης. Ο έλεγχος συστημάτων πρέπει επίσης να επαναληφθεί πριν από την παράδοση νέων εκδόσεων λογισμικού. Λόγω του υψηλού κόστους τους, βασικοί έλεγχοι ολοκλήρωσης και παλινδρόμησης συστημάτων γίνονται ως automated smoke tests. Ο εκτενής έλεγχος παλινδρόμησης πραγματοποιείται μόνο στην διαδικασία της προετοιμασίας για μια νέα παράδοση λογισμικού.

5.7.1. Εκδόσεις λογισμικού (Versions)

Καθώς το λογισμικό ελέγχεται και δοκιμάζεται, εντοπίζονται σφάλματα τα οποία πρέπει να διορθωθούν ή να γίνουν μικρές βελτιώσεις στην αρχική λειτουργικότητα του λογισμικού αυτού. Μια νέα έκδοση (release) του λογισμικού είναι ένα βελτιωμένο σύστημα το οποίο έχει σκοπό να αντικαταστάσει το παλιό.

Μια διαμόρφωση για ένα συγκεκριμένο σύστημα μερικές φορές καλείται εκδοχή (version). Συνεπώς η αρχική διανομή ενός πακέτου λογισμικού μπορεί να αποτελείται από διαφορετικές τέτοιες εκδοχές μια για κάθε πλατφόρμα ή συσκευή στην οποία θα χρησιμοποιηθεί το λογισμικό αυτό.

Η ομάδα διαχείρισης σχηματισμών είναι υπεύθυνη να εξασφαλίσει ότι κάθε εκδοχή ή έκδοση είναι σωστή και σταθερή πριν δοθεί στους πελάτες για χρήση, καθώς επίσης να διασφαλίσει ότι οι αλλαγές έγιναν γρήγορα και με ακρίβεια. Η ακρίβεια

είναι ένας πολύ κρίσιμος παράγοντας , επειδή είναι επιθυμητό να αποφεύγονται νέα σφάλματα κατά τη διάρκεια διόρθωσης των παλαιότερων.

Εφόσον διασφαλιστεί το γεγονός ότι ικανοποιούνται και οι υπόλοιπες παράμετροι ποιότητας (όπως για παράδειγμα: λειτουργικότητα, αξιοπιστία, συντηρησιμότητα, αποδοτικότητα, κ.α.), έπεται ότι το σύστημα λογισμικού μπορεί να εγγυηθεί την ικανοποίηση του πελάτη (user satisfaction) και κατά συνέπεια είναι έτοιμο για παράδοση.

6. Σύγχρονες τεχνολογίες ελέγχου και εργαλεία

Ανάλογα με το **πότε** μέσα στον κύκλο ζωής λογισμικού μια ομάδα ανάπτυξης θα επιθυμούσε να εισάγει ένα αυτοματοποιημένο εργαλείο ελέγχου, οι τύποι των εργαλείων μπορούν να διαιρεθούν σε εργαλεία που αφορούν την ανάλυση και τις προδιαγραφές του συστήματος, άλλα για τη φάση προσδιορισμού των απαιτήσεων, τη φάση σχεδίου, τη φάση εφαρμογής, την εξασφάλιση ποιότητας και τέλος τη φάση διόρθωσης σφαλμάτων του συστήματος. Υπάρχουν επίσης μετρήσεις και εργαλεία ελέγχου του οργανισμού που βοηθούν στο στάδιο της προσπάθειας διαχείρισης του ελέγχου. Μερικά εργαλεία στοχεύουν σε συγκεκριμένους στόχους όπως γεννήτριες στοιχείων ελέγχου, προσομοιωτές και εργαλεία διαμόρφωσης πρωτοτύπου[45].

6.1. Σύγχρονα εργαλεία ελέγχου

Υπάρχουν δεκάδες αυτοματοποιημένα εργαλεία ελέγχου λογισμικού [46]στην αγορά σήμερα. Τα εργαλεία ποικίλουν, όσον αφορά τη λειτουργικότητα τους μεταξύ του λειτουργικού ελέγχου-μαύρου κουτιού, και τα εργαλεία ανάλυσης πηγαίου κώδικα. Πολλά διαθέσιμα εργαλεία καταγράφουν και αναπαράγουν σενάρια ελέγχου τα οποία εισάγουν οι ελεγκτές, ελέγχοντας τα σενάρια ελέγχου ή εξερευνώντας τα (ένας καλός κατάλογος με εργαλεία ελέγχου βρίσκεται στο Aptest). Ο έλεγχος παλινδρόμησης είναι η πιο γνωστή τεχνική αυτόματου ελέγχου.

Μερικά από τα εργαλεία στοχεύουν στην ανίχνευση των λαθών χρόνου εκτέλεσης και διαρροών μνήμης. Μεταξύ αυτών των εργαλείων είναι το Purify από το Rational Software, το C Verifier από το PolySpace, το SWAT από το Coverity, το Insure++ από το Parasoft και το GlowCode από το Electric Software. Άλλα εργαλεία, είναι τα εργαλεία ελέγχου σεναρίων του οργανισμού, που βοηθήσουν στην οργάνωση της διαδικασίας ελέγχου. Οι γεννήτριες στοιχείων ελέγχου παράγουν δεδομένα που μιμούνται τα πραγματικά στοιχεία των λειτουργικών συστημάτων με σκοπό τον έλεγχο. Οι προσομοιωτές των περιβαλλόντων ελέγχου είναι επίσης διαθέσιμοι καθώς επίσης και τα αυτοματοποιημένα εργαλεία ελέγχου πίεσης και φορτίου. Τα ακόλουθα τμήματα περιλαμβάνουν μια πιο λεπτομερή ματιά σε μερικά από τα χαρακτηριστικά γνωρίσματα και την νέα τεχνολογία σε αρκετά από τα πιο ευρέως χρησιμοποιημένα εργαλεία.

6.2. Βελτιωμένος έλεγχος απόδοσης

Τα στοιχεία απόδοσης δεν μετριούνται ούτε καταγράφονται πλέον με χρονόμετρα. Μόλις το 1998, σε μια τυχαία επιχείρηση, ο έλεγχος απόδοσης διευθύνθηκε καθώς ένας μηχανικός ελέγχου κάθισε με ένα χρονόμετρο, και χρονομετρούσε τη λειτουργικότητα του συστήματος ενώ ένας άλλος μηχανικός ελέγχου εκτελούσε χειροκίνητα σενάρια ελέγχου. Αυτή η μέθοδος σύλληψης μέτρησης απόδοσης είναι μια μέθοδος εντατική , ιδιαίτερα επιρρεπής σε λάθη και δεν επιτρέπει την αυτόματη επανάληψη.

Σήμερα πολλά εργαλεία ελέγχου φορτίου που είναι στην αγορά επιτρέπουν στο μηχανικό ελέγχου να εκτελέσει αυτόματους ελέγχους της λειτουργίας του συστήματος, παράγοντας χρόνους και γραφικές παραστάσεις και επισημαίνοντας τις δυσκολίες και τα κατώτατα όρια του συστήματος. Πλέον δεν χρειάζεται ένας μηχανικός ελέγχου να καθίσει με ένα χρονόμετρο στο χέρι. Αντ' αυτού, εκτελεί ένα script ελέγχου για να συλλάβει τις στατιστικές απόδοσης αυτόματα, δίνοντας την ευχέρεια στον μηχανικό λογισμικού να κάνει μια πιο δημιουργική εργασία ελέγχου.

Στο παρελθόν, απαιτούνταν ένας αριθμός από διάφορους υπολογιστές και ανθρώπους για να εκτελέσουν ένα πλήθος ελέγχων ξανά και ξανά ώστε να παραχθούν επανειλημμένως έγκυροι στατιστικά αριθμοί απόδοσης. Τα νέα αυτοματοποιημένα εργαλεία ελέγχου απόδοσης επιτρέπουν στον μηχανικό ελέγχου να εκμεταλλευτεί τα προγράμματα που διαβάζουν δεδομένα από ένα αρχείο ή έναν πίνακα ή που χρησιμοποιούν εργαλεία παραγωγής δεδομένων, άσχετα με το εάν οι πληροφορίες αποτελούνται από μια γραμμή ή εκατοντάδες γραμμές δεδομένων. Επίσης μπορούν να αναπτυχθούν και άλλα προγράμματα ή να επαναχρησιμοποιηθούν από τις βιβλιοθήκες προγράμματος ελέγχου για να υποστηρίξουν βρόγχους επανάληψης και υποθετικές καταστάσεις.

Η νέα γενιά των εργαλείων ελέγχου επιτρέπει στον μηχανικό ελέγχου να εκτελέσει ελέγχους απόδοσης αφύλακτα, επειδή επιτρέπουν να οριστεί εκ των προτέρων ο χρόνος εκτέλεσης ελέγχου, στην συνέχεια ενεργοποιείται αυτόματα ένα script ελέγχου, χωρίς οποιαδήποτε χειροκίνητη εμπλοκή (μεσολάβηση). Πολλά αυτοματοποιημένα εργαλεία ελέγχου απόδοσης επιτρέπουν τον εικονικό έλεγχο χρηστών, στον οποίο ο μηχανικός ελέγχου μπορεί να μιμηθεί τους δεκάδες, τους εκατοντάδες, ή ακόμα και τους χιλιάδες χρήστες που εκτελούν τα διάφορα scripts ελέγχου.

Ο στόχος του ελέγχου απόδοσης είναι να παρουσιάσει ότι ένα σύστημα λειτουργεί σύμφωνα με τις προδιαγραφές απαιτήσεων της απόδοσής του όσον αφορά τους αποδεκτούς χρόνους απόκρισης, καθώς επεξεργάζεται τους απαραίτητους όγκους συναλλαγών στην βάση δεδομένων παραγωγής. Κατά τη διάρκεια του ελέγχου απόδοσης, χρησιμοποιούνται τα φορτία παραγωγής για να προβλέψουν τη συμπεριφορά και χρησιμοποιείται ένα ελεγχόμενο και μετρημένο φορτίο για να μετρηθεί ο χρόνος απόκρισης. Η ανάλυση των αποτελεσμάτων του ελέγχου απόδοσης βοηθά στο να υποστηριχθεί ο συντονισμός απόδοσης.

6.3. Προϋπολογισμός κόστους εργαλείων

Μετά την περιγραφή των οφελών των προτεινόμενων αυτοματοποιημένων εργαλείων ελέγχου, είναι απαραίτητο να υπολογιστεί το κόστος τους. ίσως να είναι απαραίτητο να περιγραφεί μια συγχρονισμένη εφαρμογή των εργαλείων ελέγχου έτσι ώστε οι δαπάνες να μπορούν να διανεμηθούν στην περίοδο του χρόνου. Μια καλή πηγή όσον αφορά τη λήψη των χαρακτηριστικών γνωρισμάτων των εργαλείων και για πληροφορίες κόστους είναι το Διαδίκτυο.

Μόλις προσδιορίσει η ομάδα ελέγχου το ενδεχόμενο κόστος για την αρχική αγορά ενός αυτοματοποιημένου εργαλείου ελέγχου, είναι πολύ σημαντικό να γίνει παράλληλα και μια γρήγορη αξιολόγηση σχετικά με το εάν αυτό το κόστος συμφωνεί με τις προσδοκίες της διοίκησης. Επιπρόσθετα, μπορεί να χρειαστεί ένα σχέδιο για τη συγχρονισμένη εφαρμογή των εργαλείων ελέγχου ώστε να τροποποιηθεί για να ευθυγραμμίσει τη βραχυπρόθεσμη στρατηγική εφαρμογής με το πραγματικό διαθέσιμο κεφάλαιο. Μια άλλη επιλογή είναι να δοθούν επιχειρήματα αύξησης του κεφαλαίου ώστε να ευθυγραμμίσει το πραγματικά διαθέσιμο κεφάλαιο με τις απαιτήσεις απόδοσης ελέγχου του οργανισμού. Τέλος πρέπει να διευθυνθεί μια ανάλυση κόστους-κέρδους, με την ομάδα ελέγχου εξασφαλίζοντας ότι το διαθέσιμο κεφάλαιο μπορεί να υποστηρίξει αυτήν την ανάλυση.

Υπό τον όρο ότι πληρούνται οι διοικητικές προσδοκίες, οι προϋπολογισμοί αγοράς εργαλείων, και οι δαπάνες για τα εργαλεία ελέγχου, είναι στην συνέχεια καλό να οργανωθεί μια παρουσίαση του προτεινόμενου εργαλείου ελέγχου από τον προμηθευτή για τη διαχείριση του, στην οποία επαναλαμβάνονται τα οφέλη της χρήσης του εργαλείου στον οργανισμό. Η παρουσίαση μπορεί επίσης να επαναλάβει το κόστος του εργαλείου. Οι δαπάνες που συνδέονται με την εφαρμογή του εργαλείου ελέγχου μπορούν να περιλαμβάνουν τις απαραίτητες δαπάνες αναβάθμισης του λογισμικού για να καλυφθούν οι όποιες απαιτήσεις απόδοσης, τις

οποιεσδήποτε απαραίτητες συμφωνίες συντήρησης λογισμικού, την άμεση υποστήριξη, και τις απαιτήσεις για την κατάρτιση εργαλείων. Εάν, στο στάδιο προτάσεων των εργαλείων ελέγχου, τα μέλη ομάδας ελέγχου δεν είναι σίγουρα σχετικά με το ποιό συγκεκριμένο εργαλείο προτιμούν, μπορεί να είναι απαραίτητο να υπολογιστούν οι δαπάνες εργαλείων παρέχοντας μια κλίμακα κόστους για κάθε ενδιαφέρον είδος αυτοματοποιημένου εργαλείου ελέγχου. Εάν η ομάδα ελέγχου έχει προσδιορίσει ένα πολύ ικανό εργαλείο ελέγχου που καλύπτει τις απαιτήσεις του οργανισμού αλλά είναι πολύ πιο ακριβό από αυτό που επιτρέπει ο προγραμματισμένος προϋπολογισμός, υπάρχουν διάφορες άλλες διαθέσιμες επιλογές.

Κατ' αρχάς, η ομάδα ελέγχου θα μπορούσε να επιλέξει ένα εργαλείο όχι τόσο ακριβό που να υποστηρίζει επαρκώς τις απαιτήσεις ελέγχου. Δεύτερον, θα μπορούσε να περιγράψει την εξοικονόμηση χρημάτων ή την απόδοση ενισχύοντας τα οφέλη με τέτοιο τρόπο ώστε να πείσει τη διαχείριση ότι το εργαλείο ελέγχου άξιζε την αρχική επένδυση. Τρίτον, θα μπορούσε να ακυρώσει την εφαρμογή του εργαλείου ελέγχου και να προγραμματίσει μια επιπρόσθετη εφαρμογή κατά τη διάρκεια της επόμενης περιόδου του προϋπολογισμού.

Κατά την εκτέλεση μια αναβάθμισης ενός οργανισμού ή μιας ανάλυσης αναγκών, είναι σημαντικό να είναι εξοικειωμένος κανείς με τους διαφορετικούς τύπους εργαλείων που είναι διαθέσιμοι στην αγορά. Αυτό το τμήμα του κειμένου παρέχει μια επισκόπηση των εργαλείων που υποστηρίζουν τις διάφορες φάσεις ζωής του προϊόντος. Αυτό το τμήμα του εργαλείου δεν προορίζεται να είναι περιεκτικό, αλλά αντ' αυτού παρέχει ένα σύνολο δειγμάτων εργαλείων που βελτιώνουν τον κύκλο ζωής ελέγχου. Εκτός από τα εργαλεία ελέγχου, συμπεριλαμβάνονται και άλλα εργαλεία στον πίνακα επειδή υποστηρίζουν την παραγωγή ενός συστήματος ελέγχου. Ακόμα κι αν μερικά εργαλεία χρησιμοποιούνται σε διάφορες φάσεις (όπως η σάρωση λαθών και τα εργαλεία διαχείρισης παραμετροποίησης), ο πίνακας απαριθμεί μόνο τα εργαλεία που χρησιμοποιούνται στην πρώτη φάση. Το παράρτημα Β παρέχει τα παραδείγματα και τις λεπτομέρειες των τύπων των εργαλείων που απαριθμούνται εδώ, και δίνει παραδείγματα και άλλων πηγών με πληροφορίες εργαλείων.

Ο οργανισμός πρέπει να καθορίσει ποια εργαλεία μπορούν να είναι πιο ωφέλιμα όσον αφορά την βελτίωση της αναπτυξιακής διαδικασίας των συστημάτων. Αυτή η αξιολόγηση γίνεται με την σύγκριση της τρέχουσας διαδικασίας με μια στοχευόμενη διαδικασία, επαληθεύοντας τον δείκτη βελτίωσης και πραγματοποιώντας μια ανάλυση κόστους-κέρδους. Πριν την αγορά ενός εργαλείου που να υποστηρίζει μια δραστηριότητα τεχνολογίας λογισμικού, πρέπει να εκτελεσθεί μια αξιολόγηση

εργαλείων για το εργαλείο, παρόμοια με την αυτοματοποιημένη διαδικασία αξιολόγησης εργαλείων ελέγχου που περιγράφεται παρακάτω.

6.4. Σχεδιασμός και οπτικά εργαλεία διαμόρφωσης

Το σχέδιο και τα οπτικά εργαλεία διαμόρφωσης χρησιμοποιούνται κυρίως στις φάσεις ανάλυσης και σχεδίου. Τα εργαλεία αυτά βοηθούν την ομάδα ανάπτυξης καθώς επίσης και τους τελικούς χρήστες, ώστε να κατανοήσουν τη λειτουργία του συστήματος που βρίσκεται υπό κατασκευή. Μερικά εκ των διαθέσιμων εργαλείων παράγουν επίσης τη «ραχοκοκαλιά» των συστημάτων από το σχέδιο που παράγεται από τα οπτικά εργαλεία διαμόρφωσης, συμπεριλαμβανομένων των πινάκων βάσεων δεδομένων, των κλάσεων, των ενοτήτων, κ.λπ. Οι προγραμματιστές μπορούν έπειτα να γράψουν τον κώδικα στα προσδιορισμένα πεδία. Το Rational Rose και το Oracle Designer είναι δύο από τα πιο γνωστά οπτικά εργαλεία σχεδιασμού και διαμόρφωσης διαθέσιμα στην αγορά.

Το **Rational Rose** παρέχει ένα κανάλι για να επικοινωνήσει η αρχιτεκτονική του συστήματος με τα διάφορα συμβαλλόμενα μέρη της ομάδας ανάπτυξης, όπως από τους αναλυτές και τους σχεδιαστές στους υπεύθυνους για την ανάπτυξη και σε άλλα μέλη ομάδας. Το Rational Rose επιτρέπει στους σχεδιαστές να χαρτογραφήσουν οπτικά την αρχιτεκτονική συστημάτων τα πρότυπα και τα διαγράμματα χρησιμοποιώντας πρότυπα παρόμοια με την UML. Τα διαγράμματα οδηγούν προς ένα πιο λεπτομερές επίπεδό μόνο για να ορίσουν την αλληλεπίδραση μεταξύ των διαφορετικών μερών του λογισμικού. Μπορείτε να γράψετε τον κώδικα στην περιοχή ορισμού των υπορουτίνων, και έτσι όταν επιλέγετε να παραγάγετε τον κώδικα από το Rational Rose , ο κώδικας που γράψατε θα υπάρχει εκεί που πρέπει στην αίτησή σας. Εντούτοις, το Rational Rose δεν οδηγεί στις λεπτομέρειες του σχεδίου ή δεν ερμηνεύει τη λειτουργία του λογισμικού όπως κάνει με το σχέδιο υψηλότερου επιπέδου. Συνεπώς, τα εργαλεία ελέγχου που είναι τώρα διαθέσιμα και παράγονται από το Rational, δεν βασίζονται σε συγκεκριμένη λειτουργία, αλλά στη «ραχοκοκαλιά», και τα πραγματικά σενάρια των χρηστών της εφαρμογής.

6.4.1. Εργαλεία της φάσης επιχειρησιακής ανάλυσης

Υπάρχουν πολυάριθμα εργαλεία που είναι στην αγορά και υποστηρίζουν τη φάση επιχειρησιακής ανάλυσης. Μερικά εργαλεία[47] υποστηρίζουν διάφορες μεθοδολογίες, όπως είναι η ενοποιημένη γλώσσα διαμόρφωσης (UML). Τα εργαλεία ανάλυσης άλλων επιχειρήσεων έχουν τη δυνατότητα να καταγράψουν τις

ευκαιρίες βελτίωσης της διαδικασίας και να παρέχουν τις ικανότητες οργάνωσης στοιχείων, βελτιώνοντας έτσι τον κύκλο ζωής του ελέγχου.

6.4.2. Εργαλεία επιχειρησιακής διαμόρφωσης

Τα εργαλεία επιχειρησιακής διαμόρφωσης υποστηρίζουν τη δημιουργία προτύπων διαδικασίας, προτύπων οργανισμού, και προτύπων δεδομένων. Επιτρέπουν τους ορισμούς καταγραφής των αναγκών των χρηστών και την αυτοματοποίηση της γρήγορης κατασκευής ευέλικτων, γραφικών πελατοκεντρικών εφαρμογών. Μερικά εργαλεία επιχειρησιακής διαμόρφωσης ενσωματώνονται με άλλες φάσεις του συστήματος/του κύκλου ζωής ελέγχου, όπως η φάση διαμόρφωσης δεδομένων, η φάση σχεδίου, η φάση προγραμματισμού, και η φάση διαχείρισης ελέγχου και παραμετροποίησης. Αυτά τα εργαλεία μπορούν να είναι απαραίτητα στην υποστήριξη της προσπάθειας ελέγχου. Η χρησιμοποίηση αυτών των εργαλείων σωστά και αποτελεσματικά θα ενισχύσει τη διαδικασία διαμόρφωσης σε όλο τον κύκλο ζωής των συστημάτων, ενώ ταυτόχρονα ενισχύει την παραγωγή συστημάτων ελέγχου.

6.4.3. Εργαλεία διαχείρισης διαμόρφωσης

Τα εργαλεία διαχείρισης διαμόρφωσης πρέπει να αναπτυχθούν νωρίς στον κύκλο ζωής, έτσι ώστε να ρυθμιστεί η αλλαγή και να καθιερωθεί μια επαναλαμβανόμενη διαδικασία. Αν και αυτή η κατηγορία εργαλείου αποτελεί τμήμα της φάσης επιχειρησιακής ανάλυσης τα εργαλεία διαχείρισης διαμόρφωσης είναι που χρησιμοποιούνται ουσιαστικά σε όλο τον κύκλο ζωής. Τα τελικά αποτελέσματα της κάθε φάσης κύκλου ζωής των συστημάτων πρέπει να βασισμένα σε ένα εργαλείο διαχείρισης διαμόρφωσης.

6.4.4. Εργαλεία ανίχνευσης λαθών

Ακριβώς όπως είναι σημαντικό να χρησιμοποιηθούν τα εργαλεία διαχείρισης διαμόρφωσης καθ' όλη τη διάρκεια ελέγχου, έτσι είναι επίσης σημαντικό να χρησιμοποιηθούν εργαλεία ανίχνευσης λαθών από την αρχή και σε όλο τον κύκλο ζωής του ελέγχου. Όλα τα λάθη ή οι αναφορές των προβλημάτων λογισμικού που εμφανίζονται σε όλη τη διάρκεια ζωής του συστήματος πρέπει να τεκμηριωθούν και

να διαχειριστούν κατά το κλείσιμο. Ο προσδιορισμός των λαθών είναι ο αρχικός στόχος των δραστηριοτήτων ελέγχου και εξασφάλισης ποιότητας. Όσον αφορά τα λάθη που αφαιρούνται επιτυχώς από την εφαρμογή, πρέπει να αναγνωρισθούν και να παρακολουθούνται μέχρι την εξουδετέρωση τους.

6.5. Τεχνική επιθεώρηση των εργαλείων διαχείρισης

Μια από τις καλύτερες στρατηγικές ανίχνευσης λαθών στηρίζεται στις τεχνικές αναθεωρήσεις ή τις επιθεωρήσεις, επειδή τέτοιες αναθεωρήσεις επιτρέπουν στα λάθη να γίνουν αντιληπτά νωρίς στον κύκλο ζωής. Οι αναθεωρήσεις και οι επιθεωρήσεις αντιπροσωπεύουν μια επίσημη τεχνική αξιολόγησης εφαρμόσιμη στις απαιτήσεις λογισμικού, το σχέδιο, τον κώδικα, και άλλα προϊόντα εργασίας λογισμικού. Συνεπάγουν μια λεπτομερή εξέταση από ένα πρόσωπο ή μια ομάδα εκτός από το συντάκτη. Τα τεχνικά εργαλεία διαχείρισης αναθεώρησης επιτρέπουν την αυτοματοποίηση της επιθεώρησης διαδικασίας, διευκολύνοντας την επικοινωνία. Υποστηρίζουν επίσης την αυτοματοποιημένη συλλογή των βασικών μετρήσεων, όπως τα στοιχεία δράσης που ανακαλύπτονται κατά τη διάρκεια μιας αναθεώρησης.

6.5.1. Εργαλεία φάσης καθορισμού των απαιτήσεων

Το λογισμικό πρέπει να αξιολογηθεί σχετικά με την κατανόηση αυτού που το λογισμικό προορίζεται να κάνει. Αυτή η κατανόηση είναι συνδεδεμένη με τις προδιαγραφές απαιτήσεων ή τους ορισμούς των περιπτώσεων χρήσης. Η ποιότητα των καθορισμένων απαιτήσεων μπορεί να καταστήσει την προσπάθεια ελέγχου (και, φυσικά, ανάπτυξης) σχετικά ανώδυνη ή εξαιρετικά δύσκολη. Όταν μια προδιαγραφή απαιτήσεων περιέχει όλες τις πληροφορίες που απαιτούνται από έναν μηχανικό ελέγχου σε μια μορφή χρησιμοποιήσιμη, οι απαιτήσεις θεωρούνται έτοιμες προς έλεγχο. Οι απαιτήσεις ελέγχου ελαχιστοποιούν την προσπάθεια και το κόστος. Εάν οι απαιτήσεις δεν είναι έτοιμες για να ελεγχθούν ή ελέγξιμες, οι μηχανικοί ελέγχου πρέπει να ψάξουν για την χαμένη πληροφορία- μια μακριά, κουραστική διαδικασία που μπορεί να παρατείνει σημαντικά την προσπάθεια ελέγχου. Δείτε το παράρτημα Α για περισσότερες πληροφορίες για τις απαιτήσεις ελέγχου.

6.5.2. Εργαλεία διαχείρισης απαιτήσεων

Τα εργαλεία διαχείρισης απαιτήσεων επιτρέπουν να ικανοποιούνται οι απαιτήσεις γρήγορα και αποτελεσματικά. Οι απαιτήσεις μπορούν να καταγραφούν σε μια

φυσική γλώσσα, όπως τα αγγλικά, χρησιμοποιώντας έναν συντάκτη κειμένων μέσα σε ένα εργαλείο διαχείρισης απαιτήσεων. Μπορούν επίσης να γραφτούν σε μια επίσημη γλώσσα όπως η LOTOS ή η Z 48 που χρησιμοποιεί έναν syntax-directed συντάκτη. Επιπλέον, οι απαιτήσεις μπορούν να διαμορφωθούν γραφικά, χρησιμοποιώντας εργαλεία όπως το Validator/ Req. Μια μέθοδος που μοντελοποιεί τις απαιτήσεις περιλαμβάνει τις περιπτώσεις χρήσης. Η κατασκευή της περίπτωσης χρήσης καθορίζει τη συμπεριφορά ενός συστήματος ή άλλης σημασιολογικής οντότητας χωρίς την αποκάλυψη της εσωτερικής δομής της οντότητας. Κάθε περίπτωση χρήσης διευκρινίζει μια ακολουθία ενεργειών, συμπεριλαμβανομένων των παραλλαγών, που η οντότητα μπορεί να εκτελέσει αλληλεπιδρώντας με τους δράστες της οντότητας. Πολλά εργαλεία διαχείρισης απαιτήσεων υποστηρίζουν την ανιχνευσιμότητα πληροφοριών.

Η ανιχνευσιμότητα περιλαμβάνει πολλά περισσότερα πέρα από την ανιχνευσιμότητα των απαιτήσεων. Συνδέοντας όλες τις πληροφορίες μαζί-όπως συνδέσεις που δημιουργούνται μεταξύ των απαιτήσεων/των περιπτώσεων χρήσης και σχεδίου, εφαρμογής, ή διαδικασιών ελέγχου-είναι ένας κρίσιμος παράγοντας στην επίδειξη της συμμόρφωσης και της πληρότητας ενός προγράμματος. Παραδείγματος χάριν, ο σχεδιαστής πρέπει να ανιχνεύσει τα τμήματα σχεδίου στις λεπτομερείς απαιτήσεις συστημάτων, ενώ ο υπεύθυνος για την ανάπτυξη πρέπει να ανιχνεύσει τα τμήματα κώδικα στα τμήματα σχεδίου. Ο μηχανικός ελέγχου πρέπει να επισημάνει τις απαιτήσεις συστημάτων στις διαδικασίες ελέγχου, μετρώντας με αυτόν τον τρόπο το ποσό της δημιουργίας της διαδικασίας ελέγχου. Τα εργαλεία διαχείρισης απαιτήσεων μπορούν επίσης αυτόματα να καθορίσουν ποιες διαδικασίες ελέγχου επηρεάζονται όταν τροποποιείται μια απαίτηση. Επιπλέον, τα εργαλεία διαχείρισης απαιτήσεων υποστηρίζουν τη διαχείριση πληροφοριών. Ανεξάρτητα από εάν το πρόγραμμα περιλαμβάνει το λογισμικό, το υλικό, το firmware, ή μια διαδικασία, τα δεδομένα πρέπει να διαχειριστούν και να ανιχνευθούν για να εξασφαλίσουν συμμόρφωση με τις απαιτήσεις μέσω όλων των φάσεων ανάπτυξης.

6.5.3. Εργαλεία φάσης προγραμματισμού

Κατά τη διάρκεια των φάσεων επιχειρησιακής ανάλυσης και σχεδίου, τα χρησιμοποιούμενα εργαλεία διαμόρφωσης επιτρέπουν συχνά την παραγωγή κώδικα από τα πρότυπα που δημιουργήθηκαν σε προηγούμενες φάσεις. Εάν αυτές οι δραστηριότητες οργάνωσης και προετοιμασιών έχουν πραγματοποιηθεί σωστά, ο προγραμματισμός μπορεί να απλοποιηθεί. Κατά συνέπεια οι προγραμματιστές πρέπει να ακολουθήσουν τα πρότυπα και να μεταφέρουν ένα μεγάλο μέρος αυτών

που ένα σύστημα μπορεί να κάνει μέσω της συνετής επιλογής των ονομάτων για μεθόδους, λειτουργίες, κατηγορίες, και άλλες δομές.

Επίσης, οι προγραμματιστές πρέπει να συμπεριλάβουν εκτενής προλόγους ή σχόλια στον κώδικά τους που να περιγράφουν και να τεκμηριώνουν το σκοπό και την οργάνωση του προγράμματος. Επιπλέον, οι υπεύθυνοι για την ανάπτυξη εφαρμογής πρέπει να δημιουργήσουν τη λογική του προγράμματος και τους αλγορίθμους που υποστηρίζουν την εκτέλεση του κώδικα. Οι πρόλογοι, οι αλγόριθμοι, και ο κώδικας του προγράμματος μπορούν να χρησιμεύσουν ως εισαγωγές στα εργαλεία ελέγχου κατά τη διάρκεια της αυτοματοποίησης ελέγχου της φάσης ανάπτυξης.

Αυτές οι εισαγωγές διευκολύνουν έναν μηχανικό ελέγχου να σχεδιάσει έναν έλεγχο. Οι πρόλογοι μπορούν να προβλεφθούν ως περιγραφές απαιτήσεων για τις μικρές μονάδες του κώδικα λογισμικού που ο προγραμματιστής θα αναπτύξει. Τα εργαλεία ελέγχου που χρησιμοποιούνται κατά τη διάρκεια της φάσης απαιτήσεων μπορούν να επαναχρησιμοποιηθούν για να εξετάσουν αυτές τις μονάδες του κώδικα λογισμικού. Τα εργαλεία όπως ο metrics reporter, ο ελεγκτής κώδικα, και ο instrumentor μπορούν επίσης να υποστηρίξουν τον έλεγχο κατά τη διάρκεια της φάσης προγραμματισμού. Μερικές φορές αυτά τα εργαλεία είναι ταξινομημένα ως στατικά εργαλεία ανάλυσης, επειδή ελέγχουν τον κώδικα ενώ δεν εκτελείται και είναι σε μια στατική φάση.

6.5.4. Εργαλεία διορθωτές σύνταξης

Οι ελεγκτές και οι διορθωτές σύνταξης συσσωρεύονται συνήθως μέσα σε έναν μεταγλωττιστή γλώσσας υψηλού επιπέδου. Αυτά τα εργαλεία υποστηρίζουν τη δυνατότητα ελέγχου του λογισμικού και επομένως είναι σημαντικά στη βελτίωση της δυνατότητας ελέγχου του λογισμικού κατά τη διάρκεια της φάσης προγραμματισμού. Η διόρθωση μπορεί να περιλαμβάνει την τοποθέτηση σημείων παύσης στον κώδικα που επιτρέπουν στο πρόγραμμα εκτέλεσης να παύσει για διόρθωση, εισάγοντας μια περίπτωση παύσης, επιτρέποντας να είναι ορατός ο πηγαίος κώδικας κατά τη διάρκεια της διόρθωσης, και επιτρέποντας στις μεταβλητές του προγράμματος να φαίνονται και να τροποποιούνται.

6.5.5. Εργαλεία διαρροής μνήμης και εργαλεία ανίχνευσης λάθους

Τα Εργαλεία διαρροής μνήμης και τα εργαλεία ανίχνευσης λάθους μπορούν να ελέγξουν για διαρροές της μνήμης, καθώς παρουσιάζουν που η μνήμη έχει διατεθεί αλλά δεν υπάρχει κανένας δείκτης, τέτοια μνήμη δεν μπορεί ποτέ να χρησιμοποιηθεί ή να ελευθερωθεί. Τα λάθη χρόνου εκτέλεσης μπορούν να

ανιχνευθούν στις βιβλιοθήκες τρίτων, στις κοινές βιβλιοθήκες, και σε άλλο κώδικα. Τα εργαλεία διαρροής μνήμης και εργαλεία ανίχνευσης λάθους μπορούν να προσδιορίσουν τα προβλήματα όπως οι τοπικές μεταβλητές, τα λάθη υπερχείλισης σωρών, και τα στατικά λάθη πρόσβασης στη μνήμη. Αυτά τα εργαλεία είναι πολύ ωφέλιμες προσθήκες στον κύκλο ζωής ελέγχου.

Ένα πρόωρο εργαλείο ελέγχου κώδικα, ονομαζόμενο LINT, είχε παρασχεθεί στους υπεύθυνους εφαρμογών ως τμήμα του λειτουργικού συστήματος της UNIX. Το LINT είναι ακόμα διαθέσιμο στα σημερινά συστήματα της UNIX. Πλέον προσφέρονται πολλοί άλλοι ελεγκτές κώδικα που υποστηρίζουν και άλλα λειτουργικά συστήματα. Το όνομα «LINT» επιλέχτηκε κατάλληλα, επειδή ο ελεγκτής κώδικα πηγαίνει κατευθείαν στον κώδικα και διαλέγει όλο το «fuzz» που καθιστά τα προγράμματα ακατάστατα και επιρρεπή σε λάθη. Αυτό ο τύπος εργαλείου ψάχνει για τοποθετημένους σε λάθος μέρος δείκτες, μεταβλητές, και αποκλίσεις από τα πρότυπα. Οι ομάδες ανάπτυξης εφαρμογών που χρησιμοποιούν τις επιθεωρήσεις λογισμικού ως στοιχείο του στατικού ελέγχου μπορούν να ελαχιστοποιήσουν τη στατική προσπάθεια ελέγχου με την επίκληση ενός κώδικα ελέγχου για να βοηθήσει να προσδιοριστούν τα μικρά προβλήματα πριν από κάθε επιθεώρηση[49].Ελεγκτές κώδικα όπως ο Abraxas Software's Codecheck μετρούν την συντηρησιμότητα την φορητότητα, την πολυπλοκότητα, και την συμμόρφωση προτύπων του πηγαίου κώδικα C και C++ .

6.5.6. Στατικές και δυναμικές συσκευές ανάλυσης

Μερικά εργαλεία επιτρέπουν τη στατική και δυναμική ανάλυση του κώδικα. Τα εργαλεία όπως το LDRA εκτελούν τη στατική ανάλυση, αξιολογώντας τον κώδικα από την πλευρά των προτύπων του προγραμματισμού, τις μετρικές πολυπλοκότητας, τον απρόσιτο κώδικα, και πολλά άλλα. Αυτά τα εργαλεία υποστηρίζουν επίσης τη δυναμική ανάλυση, η οποία περιλαμβάνει την εκτέλεση του κώδικα χρησιμοποιώντας τα στοιχεία ελέγχου για να ανιχνεύσουν τα λάθη στο χρόνο εκτέλεσης, καθώς επίσης και την ανίχνευση του μη ελεγμένου κώδικα, την ανάλυση της κατάστασης της εκτέλεσης και πολλά άλλα. Αυτοί οι τύποι προϊόντων αναλύουν τον πηγαίο κώδικα, συντάσσοντας εκθέσεις και αναφορές γραπτώς και με γραφική μορφή.

6.6. Εργαλεία μετρήσεων

Τα εργαλεία μετρήσεων, που χρησιμοποιούνται κατά τη διάρκεια των φάσεων ελέγχου μονάδων και ολοκλήρωσης, μπορούν να προσδιορίσουν τον μη ελεγμένο κώδικα και να υποστηρίξουν τον δυναμικό έλεγχο. Αυτοί οι τύποι εργαλείων παρέχουν την ανάλυση κάλυψης για να ελέγξουν ότι ο κώδικας εξετάζεται με όσο το δυνατόν περισσότερες λεπτομέρειες.

Το εργαλείο Metrics Reporter υπάρχει εδώ και χρόνια και παραμένει πολύτιμο. Αυτό το εργαλείο διαβάζει τον πηγαίο κώδικα και παρουσιάζει πληροφορίες των μετρήσεων, συχνά με γραφικό σχήμα. Εκθέτει τις μετρικές πολυπλοκότητας από την άποψη της ροής δεδομένων, της δομής δεδομένων, και της ροής ελέγχου. Παρέχει επίσης τις μετρήσεις για το μέγεθος του κώδικα από την άποψη των ενοτήτων, των τελεστών, των χειριστών, και των γραμμών κώδικα. Ένα τέτοιο εργαλείο μπορεί να βοηθήσει τον προγραμματιστή να διορθώσει τον κώδικα και να βοηθήσει τον μηχανικό ελέγχου να καθορίσει ποια μέρη του κώδικα λογισμικού απαιτούν περισσότερη προσοχή σε θέματα ελέγχου.

6.6.1. Γεννήτριες στοιχείων ελέγχου

Πολλά εργαλεία που βρίσκονται αυτή την περίοδο στην αγορά υποστηρίζουν την παραγωγή των στοιχείων ελέγχου. Τέτοια στοιχεία ελέγχου μπορούν να χρησιμοποιηθούν για όλες τις φάσεις ελέγχου, ειδικά κατά τη διάρκεια του ελέγχου απόδοσης και πίεσης, απλοποιώντας την διαδικασία ελέγχου.

6.6.2. Τα Εργαλεία προσομοίωσης

Τα εργαλεία διαμόρφωσης προσομοίωσης μπορούν να μιμηθούν τη συμπεριφορά των προτύπων της εφαρμογής που είναι υπο έλεγχο, χρησιμοποιώντας διάφορες τροποποιήσεις του συγκεκριμένου περιβάλλοντος εφαρμογής ως τμήμα των «τι-εάν» σεναρίων. Αυτά τα εργαλεία παρέχουν μια εσωτερική ματιά όσον αφορά την απόδοση και τη συμπεριφορά των υπαρχόντων ή προτεινόμενων δικτύων, των συστημάτων, και των διαδικασιών.

6.6.3. Εργαλεία διαχείρισης ελέγχου

Τα εργαλεία διαχείρισης ελέγχου υποστηρίζουν τον κύκλο ζωής ελέγχου λαμβάνοντας υπ' όψη τον προγραμματισμό, τη διαχείριση, και την ανάλυση όλων των πτυχών του. Μερικά εργαλεία, όπως το Rational's TestStudio και το WinRunner, είναι ενσωματωμένα με τα εργαλεία διαχείρισης απαίτησης και διαμόρφωσης, απλοποιώντας με αυτόν τον τρόπο ολόκληρη την διαδικασία του κύκλου ζωής του ελέγχου.

6.6.4. Εργαλεία ελέγχου δικτύων

Η εμφάνιση των εφαρμογών που λειτουργούν σε ένα πελάτη-εξυπηρετητή, σ' ένα πολυεπίπεδο, ή διαδικτυακό περιβάλλον, έχει εισαγάγει νέα πολυπλοκότητα στην προσπάθεια ελέγχου. Ο μηχανικός ελέγχου δεν δουλεύει πλέον μια ενιαία κλειστή εφαρμογή που λειτουργεί σε ένα ενιαίο σύστημα, όπως στο παρελθόν. Αντ' αυτού, η αρχιτεκτονική πελάτη-εξυπηρετητή περιλαμβάνει τρία συστατικά: τον κεντρικό υπολογιστή, τον πελάτη, και το δίκτυο. Η διαπλανητική συνδεσιμότητα αυξάνει επίσης τη δυνατότητα για λάθη. Κατά συνέπεια, ο έλεγχος πρέπει να εστιάσει στην απόδοση του κεντρικού υπολογιστή και του δικτύου, καθώς επίσης και στην γενική απόδοση και τη λειτουργία των συστημάτων στα τρία συστατικά. Πολλά εργαλεία ελέγχου δικτύων επιτρέπουν στο μηχανικό ελέγχου να παρατηρήσει, να μετρήσει, να εξετάσει, και να εντοπίσει την απόδοση μέσω ολόκληρου του δικτύου.

6.6.5. Εργαλεία ελέγχου εφαρμογής GUI

Υπάρχουν πολλά αυτοματοποιημένα εργαλεία ελέγχου GUI στην αγορά. Αυτά τα εργαλεία περιλαμβάνουν συνήθως μια καταγραφή και μια λειτουργία αναπαραγωγής, που επιτρέπει στο μηχανικό ελέγχου να δημιουργήσει (αρχείο), να τροποποιήσει, και να εκτελέσει (αναπαραγωγή) αυτοματοποιημένους ελέγχους σε πολλά περιβάλλοντα. Εργαλεία που καταγράφουν τα συστατικά του GUI στο επίπεδο widget (και όχι στο επίπεδο bitmap) είναι τα πιο χρήσιμα. Η δραστηριότητα καταγραφής συλλαμβάνει τις πληκτρολογήσεις που εισάγονται από το μηχανικό ελέγχου, δημιουργώντας αυτόματα ένα script σε μια γλώσσα υψηλού επιπέδου στο υπόβαθρο. Αυτή η καταγραφή είναι ένα πρόγραμμα υπολογιστή, το οποίο αναφέρεται σε αυτό ως «script» ελέγχου. Η χρήση μόνο των λειτουργιών σύλληψης/αναπαραγωγής εκμεταλλεύεται συνήθως το 10% της ικανότητας ελέγχου ενός εργαλείου. Ο μηχανικός ελέγχου θα πρέπει να τροποποιήσει το script για να δημιουργήσει μια επαναχρησιμοποιήσιμη και συντηρήσιμη διαδικασία ελέγχου. Αυτό το script θα γίνει ο βασικός έλεγχος και μπορεί αργότερα να εφαρμοστεί ξανά σε ένα νέο λογισμικό που δημιουργείται για τη σύγκριση. Τα εργαλεία ελέγχου που

παρέχουν μια ικανότητα καταγραφής συσσωρεύονται συνήθως με έναν συγκριτή (comparator), ο οποίος συγκρίνει αυτόματα τα πραγματικά αποτελέσματα με τα αναμενόμενα και καταγράφει τα αποτελέσματα. Τα αποτελέσματα μπορούν να συγκρίνονται ανά pixel, χαρακτήρα- χαρακτήρα, δεδομένου ότι το εργαλείο επισημαίνει αυτόματα την αποτυχία μεταξύ του αναμενόμενου και του πραγματικού αποτελέσματος. Στην περίπτωση του εργαλείου ελέγχου WinRunner, ένα θετικό αποτέλεσμα καταγράφεται στον Tests Results Viewer ως πέρασμα και απεικονίζεται στην οθόνη χρησιμοποιώντας μαύρο χρώμα, ενώ μια αποτυχία αντιπροσωπεύεται χρησιμοποιώντας κόκκινο χρώμα.

6.6.5.1. Έτοιμα συστήματα ελέγχου GUI λογισμικών συστημάτων

CompuWare TestPartner

Τα σύστημα αυτό παρέχει λειτουργίες για έλεγχο στοιχείων του γραφικού περιβάλλοντος λογισμικών συστημάτων μέσω test scripts. Τα test scripts αυτά μπορούν να γραφούν σε γλώσσα προγραμματισμού Visual Basic εάν ο χρήστης του συστήματος αυτού είναι έμπειρος προγραμματιστής. Στην περίπτωση όμως που ο χρήστης δεν είναι γνώστης προγραμματιστικών μεθόδων, το σύστημα πάλι μπορεί να χρησιμοποιηθεί μέσω του Visual Navigator, μιας λειτουργίας που παρέχεται από το σύστημα και μέσω αυτής να δημιουργήσει και να εκτελέσει τα test scripts. Παρόλα αυτά τα test scripts στην δεύτερη περίπτωση δεν θα είναι το ίδιο αποτελεσματικά.

IBM Rational Test Tools

Η εταιρεία Rational Software, θυγατρική εταιρεία της IBM, παρέχει συστήματα ελέγχου λογισμικών συστημάτων για διάφορα στάδια υλοποίησης. Για τον σκοπό ελέγχου του γραφικού περιβάλλοντος λογισμικών συστημάτων η εταιρεία αυτή διαθέτει δύο εφαρμογές.

Rational Robot

Το σύστημα αυτό συνεργάζεται με το Visual Studio 6 της Microsoft και χρησιμοποιεί λειτουργία capture / playback για να δημιουργήσει τα test scripts. Τα test scripts δημιουργούνται μέσω μιας διαδικασίας μετάφρασης των αλληλεπιδράσεων του χρήστη με το υπό έλεγχο λογισμικό σύστημα και καταγράφονται σε γλώσσα SQA Basic η οποία είναι παρόμοια με την Visual Basic. Στο σύστημα αυτό απουσιάζουν λειτουργίες για αποδοτική αναγνώριση και επαλήθευση των στοιχείων του γραφικού περιβάλλοντος.

Rational Visual Test

Το σύστημα αυτό αρχικά δημιουργήθηκε από την Microsoft και στην συνέχεια αγοράστηκε από την Rational Software. Παρέχει στους χρήστες ένα API (Application Programming Interface) μέσω του οποίου οι χρήστες μπορούν να δημιουργήσουν εφαρμογές ελέγχου του γραφικού περιβάλλοντος άλλων λογισμικών συστημάτων στο χέρι, ή με χρήση λειτουργίας capture / playback η οποία καταγράφει το test script σε γλώσσα Test Basic. Οι εφαρμογές που δημιουργούνται χρειάζονται συνεχή επίβλεψη από τους χρήστες, ώστε να τους παρέχουν δεδομένα ελέγχου.

Mercury Interactive Tools

Η εταιρεία Mercury Interactive διαθέτει διάφορα εργαλεία και εφαρμογές για έλεγχο τόσο εφαρμογών λογισμικού τόσο και διαδικτυακών εφαρμογών. Οι πιο διαδεδομένες από αυτές περιγράφονται πιο κάτω.

WinRunner

Το σύστημα αυτό μπορεί να χρησιμοποιηθεί για έλεγχο σε λογισμικά συστήματα προορισμένα για Windows αλλά και σε διαδικτυακές εφαρμογές. Η χρήση του βασίζεται και πάλι σε λειτουργίες capture / playback. Οι χρήστες μπορούν να προσθέσουν ή να αφαιρέσουν στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος μέσω κάποιου είδους χαρτογράφησης του γραφικού περιβάλλοντος, μπορούν να αλληλεπιδράσουν με τις φόρμες του συστήματος και να προσθέσουν δεδομένα ελέγχου. Τα δεδομένα αυτά όμως πρέπει να δημιουργηθούν χειροκίνητα από τους χρήστες του συστήματος. Ακόμη κάποιες από τις λειτουργίες του είναι ειδικές και δεν μπορούν να γενικευτούν για χρήση σε οποιαδήποτε εφαρμογή.

LoadRunner

Το σύστημα αυτό χρησιμοποιεί test scripts που δημιουργούνται από άλλες εφαρμογές της Mercury Interactive για εκτέλεση ελέγχου. Στη συνέχεια το σύστημα εκτελεί load testing για να ελέγξει την απόδοση του συστήματος.

Abbot

Το σύστημα Abbot χρησιμοποιείται σε εφαρμογές Java. Χρησιμοποιεί λειτουργίες για να αναπαραστήσει αλληλεπιδράσεις χρηστών με το υπό έλεγχο λογισμικό σύστημα. Ο χρήστης και σε αυτό το σύστημα πρέπει να γράψει στο χέρι τα test scripts με χρήση γλώσσας Java.

Guitar

Αυτό το σύστημα αποτελεί μια ολοκληρωμένη λύση για έλεγχο λογισμικών συστημάτων. Περιλαμβάνει λειτουργία παραγωγής σεναρίων ελέγχου και λειτουργία αναπαραγωγής για να εκτελέσει τον έλεγχο στην υπό έλεγχο εφαρμογή.

6.6.6. Εργαλεία ελέγχου φορτίων/απόδοσης/πίεσης

Τα εργαλεία ελέγχου απόδοσης, όπως το Rational's Performance Studio, επιτρέπουν τον έλεγχο φορτίων, όπου το εργαλείο μπορεί να προγραμματιστεί για να εκτελείται σε διάφορα τερματικά ταυτόχρονα να φορτώνει το σύστημα πελάτη/εξυπηρετητή και να μετρά το χρόνο απόκρισης. Ο έλεγχος φορτίων περιλαμβάνει χαρακτηριστικά διάφορα σενάρια για να αναλύσει πώς το σύστημα πελάτη/εξυπηρετητή ανταποκρίνεται κάτω από διάφορα φορτία και πιέσεις. Η έλεγχος πίεσης περιλαμβάνει τη διαδικασία χρήσης των υπολογιστών των χρηστών σε σενάρια υψηλής-πίεσης που βλέπουν πότε και εάν καταρρέουν.

Δεν υπάρχει κανένα εργαλείο που να είναι το ιδανικό. Όλα τα εργαλεία έχουν θετικά και αρνητικά ποιο είναι το πιο κατάλληλο εξαρτάται από το περιβάλλον του συστήματος και από άλλες απαιτήσεις και κριτήρια του οργανισμού.[50]

6.7. Μειονεκτήματα χρήσης αυτοματοποιημένων εργαλείων ελέγχου

Στην παραπάνω ενότητα, περιγράφηκε η λειτουργία μερικών από τα πιο κοινά εργαλεία ελέγχου λογισμικού. Εντούτοις, κανένα από τα έξυπνα αυτόματα εργαλεία ελέγχου δεν :

Αντικαθιστούν την ομάδα ελέγχου.

Αυτά τα εργαλεία ελέγχου δεν είναι μηχανήματα δεν μπορεί ο χρήστης να παρέχει το πρόγραμμα και να περιμένει πως τα εργαλεία ελέγχου θα παράγουν ένα πλήρως ελεγμένο πρόγραμμα με μια αναλυτική αναφορά σχετικά με το τι ελέγχθηκε και πως. Η αξιολόγηση του κινδύνου και ο προγραμματισμός ελέγχου, η προετοιμασία και ο έλεγχος των scripts που διευκρινίζουν το πώς ο έλεγχος θα διευθετηθεί και τέλος η ερμηνεία των καταγραμμένων αποτελεσμάτων είναι όλες αυτές οι

δραστηριότητες που κανένα εργαλείο δεν μπορεί να κάνει από μόνο του για τον χρήστη.[51]

Μπορούν να ελεγχθούν τα πάντα, ακόμη και με την αυτοματοποίηση

Σαν ελεγκτής ή διαχειριστής έργου, θα πρέπει κανείς να αποδεχτεί το γεγονός ότι δεν μπορούν όλα να εξεταστούν. Ακόμα και χρησιμοποιώντας εργαλεία που αυτοματοποιούν τον έλεγχο. Κατά την προγραμματισμό ενός αυτοματοποιημένου ελέγχου, γίνεται εστίαση στους κινδύνους που συνδέονται με την ενότητα του προγράμματος που εξετάζεται, ακριβώς όπως και σε ένα χειρωνακτικό έλεγχο. Θα πρέπει να γίνει εστίαση στους τεχνικούς κινδύνους και θα πρέπει να δοθεί λιγότερη έμφαση στους κινδύνους αποδοχής. Οι τεχνικοί κίνδυνοι που συνδέονται με το περιβάλλον του συστήματος παρέχουν το πιο γόνιμο έδαφος για τον αυτοματοποιημένο έλεγχο. Στην περίπτωση που πρέπει να εξεταστεί το λογισμικό καθώς και όλες οι τροποποιήσεις που απαιτούνται σε διάφορες πλατφόρμες, ο καλά προγραμματισμένος αυτοματοποιημένος έλεγχος μπορεί να μειώσει κατά πολύ την εργασία που θα χρειαζόταν να κάνει ο χρήστης. Σε αντίθεση, αυτοματοποιημένοι έλεγχοι που ελέγχουν επανειλημμένα το ίδιο χαρακτηριστικό γνώρισμα του προγράμματος στο ίδιο πάντα περιβάλλον σπάνια θα ανακαλύψουν νέα λάθη. Κατά γενικό κανόνα, οι έλεγχοι που έχουν σχεδιαστεί με σκοπό να αποδείξουν ότι ένα πρόγραμμα δουλεύει-αναφέρονται ως θετικοί έλεγχοι προσαρμόζονται πιο εύκολα στην αυτοματοποίηση από ότι οι έλεγχοι που έχουν ως σκοπό να βρουν προηγούμενα μη καταγεγραμμένα λάθη.

Ενημερώνονται αυτόματα οι έλεγχοι

Η διαδικασία δημιουργίας ενός αυτοματοποιημένου ελέγχου αποκαλύπτει συχνά περισσότερα λάθη από όσα παρουσιάζουν αργότερα τα αυτόματα εκτελεσμένα scripts ελέγχου. Ενώ στην περίπτωση που τα scripts ελέγχου έχουν τελειώσει την εκτέλεση τους δεν σημαίνει ότι η διαδικασία ελέγχου έχει τελειώσει. Εάν ένας αυτοματοποιημένος έλεγχος αποτύχει, θα πρέπει να ερευνηθούν οι αιτίες αποτυχίας. Σε πολλές περιπτώσεις, το script ελέγχου δεν ταιριάζει πλέον με το λογισμικό στην περίπτωση που έχουν γίνει τροποποιήσεις ή διάφοροι εμπλουτισμοί. Ένας αυτοματοποιημένος έλεγχος δεν διακρίνει τη διαφορά μεταξύ μιας αλλαγής του προγράμματος και ενός λάθους. Όσον αφορά στον έλεγχο, οτιδήποτε είναι διαφορετικό από την τελευταία φορά που ελέγχθηκε θα είναι λάθος με τον νέο έλεγχο. Εάν η αλλαγή δεν είναι πράγματι ένα λάθος, θα πρέπει το script ελέγχου να ενημερωθεί αντίστοιχα. Σε οποιαδήποτε περίπτωση αλλαγής του λογισμικού, θα πρέπει ο χρήστης να ενημερώνει και τα αντίστοιχα scripts ελέγχου .

Έλεγχος Λογισμικού

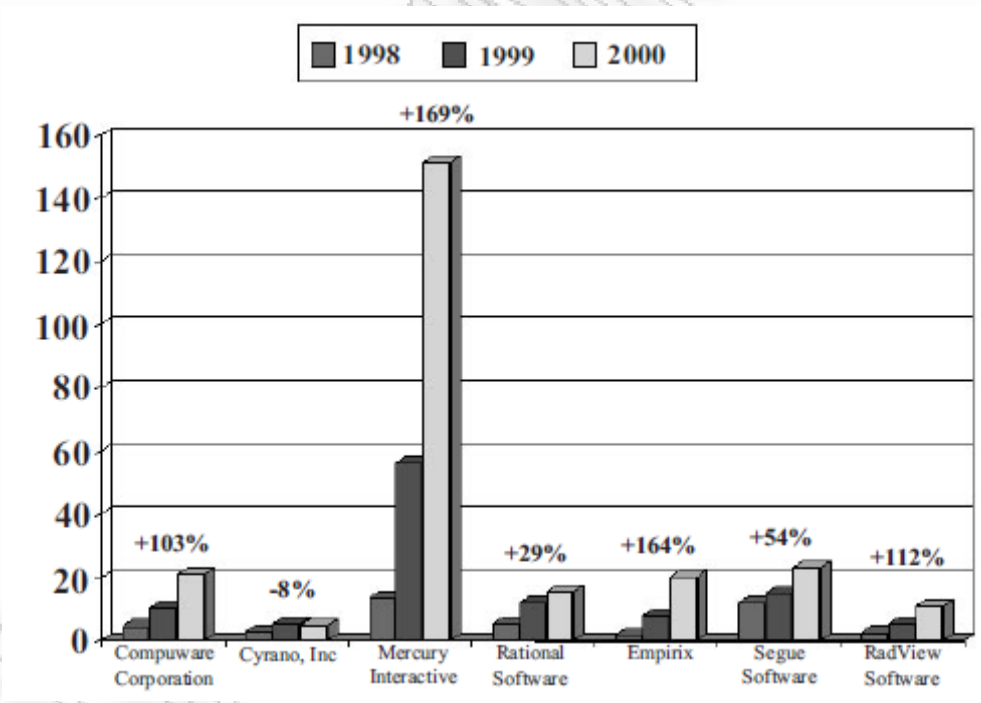
Η συνεχής εστίαση στον έλεγχο και οι συνεχείς ανανεώσεις στις δραστηριότητες ελέγχου δημιουργούν σαφώς ένα λογισμικό καλύτερης ποιότητας, αλλά φυσικά με κόστος.

Επίσης κανένα εργαλείο δεν δημιουργεί σενάρια ελέγχου βασισμένα στην κατανόηση της κατασκευής του λογισμικού και του τι προορίζεται να κάνει. Μερικά από τα διαθέσιμα αυτά εργαλεία καταγράφουν και αναπαράγουν τα σενάρια ελέγχου που παράγονται από την περιήγηση των ελεγκτών στο προϊόν κατά τον έλεγχο του. Σε αυτήν την περίπτωση, η κάλυψη βασίζεται στο πόσο καλά ο ελεγκτής ή ο χρήστης περνά από τα διαφορετικά και ενδεχομένως πολυάριθμα σενάρια συστημάτων. Οι ανταποκρίσεις του συστήματος πρέπει να κριθούν από τον ελεγκτή που καθορίζει εάν το σύστημα πέρασε τον έλεγχο ή όχι, βασισμένος στην πλήρη κατανόηση και γνώση των προδιαγραφών. Ο ελεγκτής λοιπόν είναι αυτός που θα αποφασίσει αν το σύστημα πέρασε το συγκεκριμένο έλεγχο ή όχι.

7. Επιλογή εργαλείου ελέγχου

7.1. Αυτοματοποιημένα εργαλεία ελέγχου

Η λέξη κλειδί εδώ είναι το αυτοματοποιημένα. Ένα αυτοματοποιημένο σύστημα ελέγχου λογισμικού[52] είναι ένα σύστημα το οποίο θα παράγει δεδομένα εισόδου βάση του κώδικα του προγράμματος, με τα οποία θα ελέγχει διάφορα κομμάτια (μονοπάτια) του κώδικα. Έτσι με όπλο την υπολογιστική δύναμη των σημερινών υπολογιστών η διαδικασία αυτή θα φθάσει σε υψηλά επίπεδα απόδοσης. Αποτέλεσμα αυτού είναι ο έλεγχος περισσότερων κομματιών του κώδικα σε πολύ λιγότερο χρόνο σε σχέση με τον έλεγχο που μπορεί να κάνει από μόνος του ο άνθρωπος. Συνεπώς η πιθανότητα εύρεσης λαθών θα είναι κατά πολύ μεγαλύτερη. Τα συστήματα αυτά είναι αναγκαία πλέον μετά την δημιουργία πολύπλοκων και πολύ μεγάλων σε έκταση κώδικα λογισμικών.



Διάγραμμα 3 Η ανάπτυξη της αγοράς οργανισμών ελέγχου λογισμικού.

Πηγή: «Newport Group, Inc.2001, IT Trends Research and Reporting»

Ήταν ουσιαστικό να ερευνηθούν τα τρέχοντα αυτοματοποιημένα εργαλεία ελέγχου στην σημερινή αγορά, προκειμένου να προσδιοριστεί που μπορούν να γίνουν βελτιώσεις. Τα προϊόντα των δημοφιλέστερων παρόχων αυτοματοποιημένων

εργαλείων ελέγχου εμπεριέχονται παρακάτω, συμπεριλαμβανομένου του Mercury Interactive, του Rational Rose, του Segue και του Compuware. Στο Ηνωμένο Βασίλειο, το Mercury Interactive κρατά τουλάχιστον το 50% του μεριδίου αγοράς στα εργαλεία ελέγχου λογισμικού. Έχουν έναν αριθμό απο αυτοματοποιημένα εργαλεία ελέγχου, συμπεριλαμβανομένου των **WinRunner-TestDirector [53], του QuickTest [54] και του LoadRunner [55].**

Το **Rational Software**[56] προσφέρει ένα πλήρες σύνολο κύκλου ζωής εργαλείων, συμπεριλαμβανομένου του ελέγχου για την πλατφόρμα των windows. Είναι οι πρωτοπόροι στην αγορά στα αντικειμενοστρεφή εργαλεία ανάπτυξης. Τα κύρια εργαλεία ελέγχου που είναι διαθέσιμα από το Rational Software περιλαμβάνουν το Rational Robot, το Functional Tester, τον Performance Tester και το Rational Purify. Το Rational Robot είναι ένα εργαλείο διαχείρισης ελέγχου που προσφέρει σενάρια ελέγχου ως αντικείμενα ελέγχου. Όπως περιγράφεται από το Rational είναι ένα " Γενικού σκοπού αυτοματοποιημένο εργαλείο για πελατοκεντρικές εφαρμογές.

Ο **Functional Tester**, από την άλλη μεριά είναι ένας αυτοματοποιημένος λειτουργικός και παλινδρομικός έλεγχος της Java, του Web[57] και των VS.NET εφαρμογών που βασίζονται στο winForm , όπως περιγράφεται από το Rational. Στοχεύει στη διαδικασία της παραγωγής test script και της διαχείρισης τους. Ο Performance Tester πιστοποιεί τον χρόνο απόκρισης και κλιμάκωσης της εφαρμογής ενώ στοχεύει στο να αναπαραστήσει ένα περιβάλλον πραγματικού - χρόνου βασισμένο στο WEB όπου πολλοί από τους χρήστες χρησιμοποιούν ταυτόχρονα το λογισμικό. Το Rational Purify έχει σαν στόχο την ανίχνευση διαρροής της μνήμης κα γενικότερα προβλημάτων της μνήμης για Linux και UNIX.

Το πιο σημαντικό προϊόν της Compuware είναι το **QARun.** Παρόμοιο με άλλα αυτοματοποιημένα εργαλεία ελέγχου, το QARun βασίζεται στον έλεγχο σεναρίου, καταγράφοντας τις ενέργειες του χρήστη και τις αντίστοιχες αποκρίσεις των συστημάτων σε scripts ελέγχου για να ελέγξει μερικές από τις λειτουργίες των εφαρμογών. Το Compuware έχει επίσης ένα εργαλείο ελέγχου φόρτωσης. Το Segue είναι ένας άλλος οργανισμός που παρέχει κάποια συγκριτικά εργαλεία ελέγχου λογισμικού των εργαλείων Mercury Interactive and του Rational Software. Μεταξύ των προϊόντων που προσφέρει είναι το SilkCentral Test Manager, το SilkTest 5, και ο SilkPerformer.

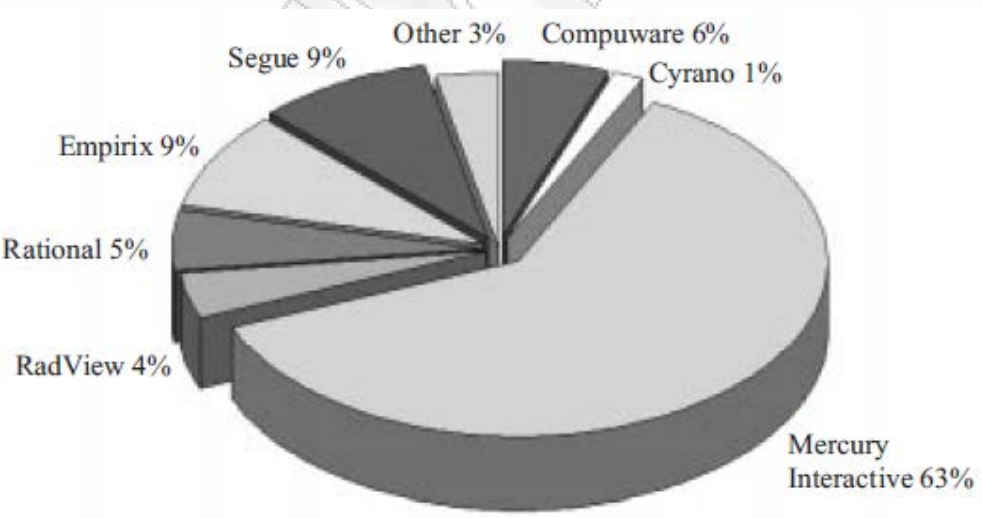
7.2. Κριτήρια του για την επιλογή του σωστού εργαλείου

Η απόδοση της επένδυσης (ROI) που αποκομίζεται με τη χρησιμοποίηση ενός αυτοματοποιημένου εργαλείου ελέγχου εξαρτάται κατά ένα μεγάλο μέρος από την κατάλληλη επιλογή ενός εργαλείου ελέγχου. Τα αυτοματοποιημένα εργαλεία ελέγχου μπορούν να ποικίλουν ανάμεσα σε αυτά που έχουν απλή λειτουργικότητα

και σε εκείνα που προσφέρουν μια πιο σύνθετη λειτουργικότητα, και η απόδοσή τους αντίστοιχα μπορεί να ποικίλει από μέτρια σε άριστη. Ένα εργαλείο ελέγχου με την ελάχιστη λειτουργικότητα θα κοστίσει συχνά λιγότερο από ένα εργαλείο με μια πιο εκτενή λειτουργικότητα. Η πρόκληση για τον ελεγκτή είναι να επιλέξει το καλύτερο εργαλείο ελέγχου για τον οργανισμό και να καταλάβει ποιοι τύποι εργαλείων είναι διαθέσιμοι και θα μπορούσαν να ικανοποιήσουν αυτές τις ανάγκες. Παραδείγματος χάριν, ένα εργαλείο διαχείρισης απαιτήσεων όπως το DOORS μπορεί επίσης να υιοθετηθεί και ως εργαλείο διαχείρισης ελέγχου, ακόμα κι αν το DOORS δεν διαφημίζεται για αυτόν τον σκοπό. Όσον αφορά την επιλογή ενός εργαλείου ελέγχου, ο μηχανικός ελέγχου πρέπει να περιγράψει τα σημαντικότερα κριτήρια για την επιλογή εργαλείων. Είναι σημαντικό να επισημανθεί στην πρόταση ότι ο καθορισμός των απαιτήσεων των εργαλείων και η έρευνα και η αξιολόγηση των διάφορων εργαλείων θα απαιτήσουν προσωπικούς πόρους, χρόνο, και χρήματα.

7.3. Σύγκριση εργαλείων

Συγκριτικά με το Rational Robot, το SilkCentral είναι ένα εργαλείο διαχείρισης και οργάνωσης ελέγχου[58]. Το SilkTest είναι συγκρίσιμο με το QuickTest, και τον Rational Functional Tester και είναι ένα εργαλείο ελέγχου παλινδρόμησης που καταγράφει και αναπαράγει τις περιπτώσεις ελέγχου [59]. Το SilkPerformer είναι ένα αυτοματοποιημένο εργαλείο ελέγχου φορτίων, πίεσης και απόδοσης, συγκρίσιμο με τον ελεγκτή απόδοσης του Rational και του Mercury LoadRunner.



Διάγραμμα 4 Μερίδιο αγοράς για τους οργανισμούς ελέγχου λογισμικού το 2000.

Πηγή: «Newport Group, Inc.2001, IT Trends Research and Reporting»

Και οι τέσσερις κύριοι προμηθευτές αυτοματοποιημένων εργαλείων ελέγχου έχουν παρόμοια προϊόντα όσον αφορά τη λειτουργία, με τα προϊόντα των ανταγωνιστών τους. Τα διαθέσιμα εργαλεία είναι κυρίως ταξινομημένα σε τέσσερις σημαντικούς τύπους δηλαδή, στη διαχείριση ελέγχου, στη σύλληψη και την επανάληψη, λειτουργικός και έλεγχος παλινδρόμησης και απόδοσης ή εργαλεία ελέγχου πίεσης (stress testing tools). Ο πίνακας 2 συνοψίζει τη λειτουργικότητα των διαφορετικών εργαλείων ελέγχου. Εργαλεία διαχείρισης ελέγχου όπως ο Mercury' s Test Director, το Rational ,το WinRunner, τον Test Manager, το SilkCentral από το Segue και το QACenter από το Compuware που οργανώνουν όλα την διαδικασία ελέγχου.

7.3.1. Πίνακες Σύγκρισης

Τα εργαλεία διαχείρισης ελέγχου όπως περιγράφονται από το Mercury, παρέχουν μια συνεπή, επαναλαμβανόμενη διαδικασία για την συλλογή των απαιτήσεων, προγραμματίζουν και σχεδιάζουν ελέγχους, αναλύουν τα αποτελέσματα, και διαχειρίζονται τα σφάλματα και τα θέματα που προκύπτουν[60]. Οι ελεγκτές μπορούν να δημιουργήσουν τις περιπτώσεις ελέγχου και να τις καταγράψουν σε εργαλεία διαχείρισης ελέγχου για να οργανώσουν την διαδικασία ελέγχου.

Organization	***Scenario Management***	***Capture and Replay***	***Functional & Regression Testing***	***Performance Testing***
Mercury Interactive	TestDirector	WinRunner	QuickTest	LoadRunner
Rational Software	TestManager	RationalRobot	FunctionalTester	PerformanceTester,Purify
Segue	SilkCentral	SilkTest	SilkTest5	SilkPerformer
Compuware	QACenter	QARun	TestPartner	QACenter

Πίνακας 2 «Παρουσίαση της λειτουργικότητας των εργαλείων ελέγχου»

Τα εργαλεία σύλληψης και επανάληψης συνήθως παράγουν αυτόματα Script ελέγχου από τα σενάρια ελέγχου που συλλαμβάνονται κατά τη διάρκεια της περιήγησης ενός ελεγκτή στο λογισμικό. Παραδείγματα από τα πιο συνηθισμένα εργαλεία αναπαραγωγής είναι τα Mercury's WinRunner, Rational Robot, SilkTest από το Segue, και το QARun από το Compuware.

Έλεγχος Λογισμικού

Τα εργαλεία λειτουργικότητας και παλινδρόμησης είναι επίσης κοινά εργαλεία στην αυτοματοποίηση λογισμικού. Τα εργαλεία ελέγχου παλινδρόμησης εκτελούν scripts ελέγχου εφόσον έχουν επιλυθεί τα προβλήματα, για να εξετάσουν αν δημιουργήθηκαν άλλα προβλήματα σε άλλα μέρη του λογισμικού που εκτελούνταν προηγουμένως σωστά. Παραδείγματα εργαλείων παλινδρόμησης και λειτουργικότητας είναι τα Mercury's QuickTest, Rational Functional Tester, Segue's SilkTest5, και Compuware's TestPartner.

Ακολουθεί μια σύγκριση εργαλείου με εργαλείο ώστε να καταλάβει ο χρήστης την λειτουργικότητα του κάθε εργαλείου.

	Record&Playback	Web Testing	Database Tests	Data Functions	Object Mapping	Image Testing	Test/Error recovery	Object Name Map	Object Identity Tool	Extensible Language	Environment Support	Integration	Cost	Ease of Use	Support	Object Tables
WinRunner	2	1	1	2	1	1	2	1	2	2	1	1	3	2	1	1
QARun	1	2	1	2	1	1	2	2	1	2	2	1	2	2	2	1
Silk Test	1	2	1	2	1	1	1	1	2	1	2	3	3	3	2	1
Visual Test	3	3	4	3	2	2	2	4	1	2	3	2	1	3	2	2
Robot	1	2	1	1	1	1	2	4	1	1	2	1	2	1	2	1

Πίνακας 3 Σύγκριση λειτουργικότητας κάθε εργαλείου

Ο παραπάνω πίνακας παρέχεται για γρήγορη και εύκολη αναφορά στις δυνατότητες του κάθε εργαλείου ελέγχου. Σε κάθε κατηγορία του πίνακα δίνεται ένα ποσοστό από 1 έως 5.

1. Άριστη υποστήριξη για την λειτουργικότητα
2. Καλή υποστήριξη αλλά κάπου υστερεί κάποιο άλλο εργαλείο παρέχει πιο αποτελεσματική υποστήριξη
3. Βασική υποστήριξη
4. Δεν υποστηρίζεται, ή υποστηρίζεται μέσω ενός άλλου επιπρόσθετου συστήματος, γενικά κάτω του μέσου όρου
5. Καμία υποστήριξη

Όπως είναι λογικό από τα παραπάνω όσο πιο χαμηλό είναι το άθροισμα τόσο πιο καλό το εργαλείο.

Αποτελέσματα Σύγκρισης

• **WinRunner = 24**

• QARun = 25

• SilkTest = 24

• Visual Test = 39

• Robot = 24

Αν τα εργαλεία ήταν τόσο φθηνά όσο είναι το Visual Test είχαν τόσο καλή λειτουργικότητα στο θέμα διαχείρισης δεδομένων όπως το Robot ήταν τόσο εξυπηρετικά εύκολα στη χρήση τους και φιλικά ως προς το χρήστη όπως το WinRunner είχαν μια εύκαμπτη και δυνατή γλώσσα προγραμματισμού όπως το SilkTest και μια κύρια αποθήκη και λειτουργικότητες κειμένου όπως το QARun τότε θα υπήρχε το τέλειο εργαλείο για έλεγχο το VisualRoWinSilkQA!!!

Άλλα θα σας παρουσιάσω συνοπτικά με ποια κριτήρια επιλέχθηκαν και χωρίστηκαν σε κατηγορίες τα προϊόντα που βρίσκονται ήδη στην αγορά.

- Εάν το θέμα είναι η **τιμή** τότε η λύση είναι το Visual Test Robot και το QARun τα οποία τείνουν να είναι τα πιο οικονομικά .
- Εάν το θέμα δεν είναι η τιμή και θέλει ο χρήστης να έχει ένα **ολοκληρωμένο πακέτο** ελέγχου το οποίο μπορεί να προϋποθέτει και σύνδεση με εργαλεία ανάπτυξης κώδικα. Τότε τα επικρατέστερα εδώ είναι τα Compuware και το Rational.
- Για έλεγχο εφαρμογών **web** τα καλύτερα είναι τα WinRunner, SilkTest και το Robot.
- Ένα είναι επιθυμητό να δημιουργηθούν τεράστια πακέτα ελέγχου παλινδρόμησης για διάφορα έργα πιθανόν σε διάφορες πλατφόρμες. Θα πρότεινα τα SilkTest, WinRunner και QARun σε αυτή την περίπτωση. Λόγω του ότι αυτά τα εργαλεία έχουν φανταστικές δυνατότητες για διαχείριση scripts είτε μέσω χαρτογράφησης GUI ή παρόμοιες δυνατότητες καθώς επίσης και στην βάση ανάκτησης συστημάτων.
- Εάν χρειάζεται κάποιος πολύ επαγγελματική βοήθεια και υποστήριξη τότε οι καλύτερες εταιρίες είναι το Mercury και το Compuware.

7.3.2. Επιλογή εργαλείου ελέγχου WinRunner

Είναι το δημοφιλέστερο εργαλείο ελέγχου, με αφθονία λειτουργιών, πολύ καλή υποστήριξη, πολύ καλή online επικοινωνία και καλή υποστήριξη Cross browser. Πιο αναλυτικά :

- ✓ **Αυξάνει σημαντικά τη δύναμη και την ευελιξία των δοκιμών χωρίς οποιοδήποτε προγραμματισμό:** Η γεννήτρια λειτουργίας παρουσιάζει έναν γρήγορο και χωρίς λάθη τρόπο για να σχεδιαστούν οι έλεγχοι και να ενισχυθούν τα scripts χωρίς οποιαδήποτε γνώση προγραμματισμού. Οι ελεγκτές μπορούν απλά να δείξουν ένα αντικείμενο GUI, και ο WinRunner θα το ελέγξει, θα εξετάσει την κλάση του και θα προτείνει να χρησιμοποιηθεί μια κατάλληλη λειτουργικότητα.

- ✓ **Χρησιμοποιούνται πολλαπλοί τύποι επαλήθευσης χρήσης για να εξασφαλιστεί η σωστή λειτουργικότητα:** Το WinRunner παρέχει τα σημεία ελέγχου (checkpoints)για το κείμενο, το GUI, τα bitmaps, τις συνδέσεις URL και τη βάση δεδομένων, επιτρέποντας στους ελεγκτές να ελέγξουν τις αναμενόμενες και τα πραγματικά αποτελέσματα και να προσδιορίσει τα πιθανά προβλήματα με τα πολυάριθμα αντικείμενα GUI και τη λειτουργικότητα τους.

- ✓ **Ελέγχει την ακεραιότητα των στοιχείων στη βάση δεδομένων:** Η ενσωματωμένη επαλήθευση βάσεων δεδομένων επιβεβαιώνει τις τιμές που αποθηκεύονται στη βάση δεδομένων και εξασφαλίζει την ακρίβεια της συναλλαγής και την ακεραιότητα των στοιχείων των αρχείων που έχουν ενημερωθεί, έχουν διαγραφεί και έχουν προστεθεί.

- ✓ **Είναι δυνατή η προβολή, η αποθήκευση και ο έλεγχος με μια ματιά κάθε ιδιότητας των αντικειμένων που είναι υπο έλεγχο:** Ο WinRunner GUI κατάσκοπος προσδιορίζει αυτόματα, καταγράφει και επιδεικνύει τις ιδιότητες των τυποποιημένων αντικειμένων GUI, των ελέγχων ActiveX, καθώς επίσης και των αντικειμένων και των μεθόδων της Java. Αυτό εξασφαλίζει ότι κάθε αντικείμενο στο ενδιάμεσο με τον χρήστη αναγνωρίζεται από το script και μπορεί να εξεταστεί.

- ✓ **Διατήρηση των έλεγχων και η δημιουργία επαναχρησιμοποιήσιμων script:** Ο χάρτης GUI παρέχει μια συγκεντρωμένη αποθήκευση αντικειμένων, επιτρέποντας στους ελεγκτές να ελέγξουν και να τροποποιήσουν οποιοδήποτε αντικείμενο ελέγχου. Αυτές οι αλλαγές έπειτα διαδίδονται

αυτόματα σε όλα τα κατάλληλα scripts, εξαλείφοντας την ανάγκη να χτιστούν νέα scripts κάθε φορά που τροποποιείται η εφαρμογή.

- ✓ **Ο έλεγχος πολλαπλών περιβαλλόντων με μια μόνο εφαρμογή:** Το WinRunner υποστηρίζει περισσότερα από 30 περιβάλλοντα, συμπεριλαμβανομένων των Web, Java, Visual Basic,κ.λπ. Επιπλέον, παρέχει στοχοθετημένες λύσεις για εφαρμογές ERP/CRM όπως είναι τα SAP, Siebel, PeopleSoft και διάφορα άλλα.

- ✓ **Απλοποίηση της δημιουργίας των scripts ελέγχου:** Ο WinRunner's DataDriver Wizard απλοποιεί πολύ τη διαδικασία της δημιουργίας δεδομένων και script ελέγχων . Αυτό επιτρέπει τη βέλτιστη χρήση των πόρων Ανάλυσης Ποιότητας και οδηγεί σε έναν πιο λεπτομερή έλεγχο.

- ✓ **Προσδιορίζει αυτόματα τις αποκλίσεις στα στοιχεία:** Το WinRunner εξετάζει και συγκρίνει τα αναμενόμενα και τα πραγματικά αποτελέσματα χρησιμοποιώντας πολλαπλές επαληθεύσεις για το κείμενο, το GUI, τα bitmaps, τα URLs, και τις βάσεις δεδομένων. Αυτό εξασφαλίζει τη σταθερή λειτουργικότητα και την εκτέλεση των επιχειρησιακών συναλλαγών πριν τεθεί η εφαρμογή στην παραγωγή.

- ✓ **Επικυρώνει τις εφαρμογές στις μηχανές αναζήτησης:** Το WinRunner επιτρέπει να χρησιμοποιηθεί ο ίδιος έλεγχος για να επικυρώσει τις εφαρμογές διαδικτυακά. Αυτό βοηθά στο να κερδίσει χρόνο ελέγχου και μειώνει τον αριθμό των script που πρέπει να δημιουργηθούν και να διαχειριστούν.

- ✓ **Ανακτά αυτόματα τις εφαρμογές ελέγχου από μια καταστροφή:** Απροσδόκητα γεγονότα, λάθη, και συντριβές εφαρμογής κατά τη διάρκεια μιας λειτουργίας ελέγχου μπορούν να προκαλέσουν προβλήματα στην διαδικασία ελέγχου και να διαστρεβλώσουν τα αποτελέσματα. Ο WinRunner's Recovery Manager επιτρέπει την αποκατάσταση και παρέχει έναν μάγο που καθοδηγεί τη διαδικασία εύρεσης ενός σεναρίου αποκατάστασης.

- ✓ **Δυνατότητα σύνδεσης και με άλλα προϊόντα ελέγχου:** Το WinRunner ενσωματώνεται πλήρως και με άλλες λύσεις ελέγχου, συμπεριλαμβανομένου του LoadRunner για τον έλεγχο φορτίων και τον TestDirector για τη σφαιρική διαχείριση ελέγχου. Επιπλέον, οι οργανισμοί μπορούν να επαναχρησιμοποιήσουν τα scripts ελέγχου του WinRunner με τον επαγγελματία QuickTest.

8. Μελέτη Περίπτωσης – Εφαρμογή ελέγχου σε τραπεζικό σύστημα

Οι διασκελισμοί στον τομέα της τεχνολογίας έχουν επαναπροσδιορίσει το ρόλο και τη δομή ενός πληροφοριακού τμήματος σε μια τράπεζα.. Η νέα εποχή των τραπεζικών εργασιών απαιτεί φρέσκιες τεχνολογικές προσεγγίσεις, τη στιγμή που οι τράπεζες οφείλουν να απαλλαγούν απο το βάρος άκαμπτων λύσεων, που έχουν σχεδιαστεί να εξυπηρετούν ανάγκες που ίσχυαν όταν ο οικονομικός κόσμος ήταν πολύ διαφορετικός. Το γεγονός ότι χρησιμοποιώντας καλύτερη τεχνολογία και συστήματα, οι τράπεζες μπορούν να συγκεντρώσουν περισσότερους πελάτες και να διατηρήσουν τους υπάρχοντες έχει αναγκάσει το επιχειρησιακό τμήμα να εξετάσει τώρα το πληροφοριακό τμήμα ως αποτελεσματικό εργαλείο μάρκετινγκ. Στη λειτουργική πλευρά, η δύναμη της τεχνολογίας πληροφοριών οδηγεί στη μείωση δαπανών συναλλαγής, την παροχή καλύτερης εξυπηρέτησης πελατών και την προσφορά μιας γενικότερης ευκολίας για τους πελάτες το οποίο έχει βασικά δημιουργήσει αυτήν την αμοιβαία ωφέλιμη κατάσταση και για τις τράπεζες αλλά και για τους πελάτες της. Αυτά είναι η κύρια αιτία που έχει λάβει τόσο μεγάλη σημασία η τεχνολογία πληροφορικής στον τομέα των τραπεζών τον τελευταίο καιρό. Ο κορμός της τεχνολογίας στο τμήμα πληροφορικής μιας τράπεζας είναι το «κεντρικό τραπεζικό σύστημα».

8.1. Τι είναι ένα κεντρικό τραπεζικό σύστημα;

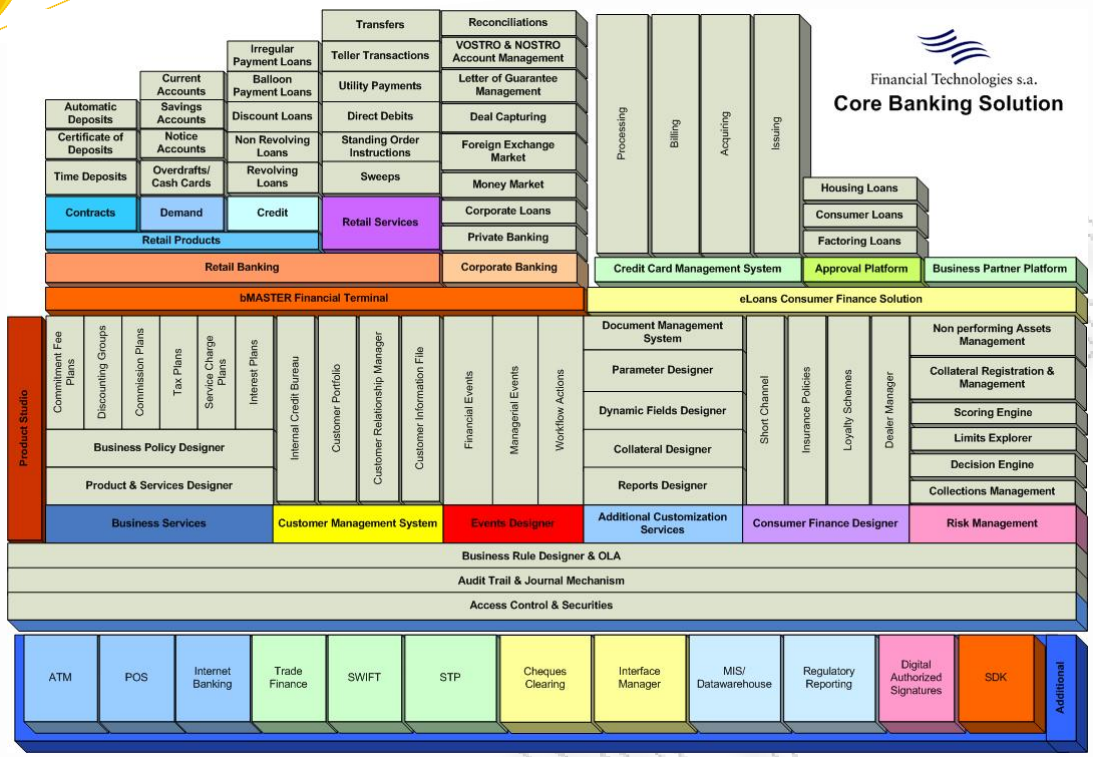
Τα κεντρικά τραπεζικά συστήματα είναι βασικά η καρδιά όλων των συστημάτων που τρέχουν σε μια τράπεζα και διαμορφώνουν τον πυρήνα της πλατφόρμας πληροφορικής της τράπεζας[61]. Μεταξύ άλλων λειτουργιών, παρέχει τη διαχείριση πληροφορίας του πελάτη, την κεντρική λογιστική και τις λειτουργίες συναλλαγής-επεξεργασίας, οι οποίες είναι οι πιο θεμελιώδεις διαδικασίες σε μια τράπεζα. Με την πρόοδο στην τεχνολογία και με το πέρασμα του χρόνου, τα κεντρικά συστήματα σήμερα τείνουν να καλύψουν όλο και περισσότερη λειτουργικότητα παρέχοντας στην τράπεζα μια ενσωματωμένη λύση για τις περισσότερες από τις διαδικασίες της σε διαφορετικούς επιχειρησιακούς τομείς. Παράλληλα, παρέχει επίσης μια κεντρική λειτουργική βάση δεδομένων που περιλαμβάνει τα κεφάλαια των πελατών και τις εγγυήσεις που δίνουν τη δυνατότητα να παραχθεί μια πλήρης εικόνα του βαθμού της σχέσης του πελάτη με την τράπεζα, η οποία είναι και θεμελιώδης για την στρατηγική CRM της τράπεζας.

8.1.1. To bMaster

Στην παρακάτω μελέτη περίπτωσης παρουσιάζεται το εργαλείο ελέγχου που χρησιμοποιήθηκε για να ελεγχθεί το τραπεζικό σύστημα **bMASTER**.

Το bMASTER Enterprise Banking System, αποτελεί ένα πλήρες, τεχνολογικά προηγμένο και ευέλικτο πακέτο Core Banking εφαρμογών, αποκλειστικά ανεπτυγμένο από τη Uni Systems. Είναι μια πλήρης, τεχνολογικά προηγμένη λύση που μπορεί να λειτουργήσει σε χρηματοπιστωτικά ιδρύματα όλων των μεγεθών και όλων των χωρών, συγκεκριμένα έχει υιοθετηθεί πλήρως αυτήν την περίοδο από την Eurobank και Alpha Bank Κύπρου. Καλύπτει ένα ευρύ φάσμα επιχειρησιακών διαδικασιών που πραγματοποιούνται μέσα σε μια τράπεζα, όπως λιανική τραπεζική, τραπεζική επιχειρήσεων και multi-channel τραπεζικές εργασίες. Καλύπτει μεγάλο εύρος τραπεζικών λειτουργιών και ροών εργασίας, όπως τραπεζική επιχειρήσεων, λιανική τραπεζική και τραπεζική μέσω πολλαπλών εναλλακτικών καναλιών διανομής. Ο ευέλικτος αρχιτεκτονικός σχεδιασμός του bMASTER, επιτρέπει στο Management της τράπεζας να αντιστοιχίζει την στρατηγική του οργανισμού με το τραπεζικό πακέτο και μέσω των εργαλείων για λήψη διοικητικών αποφάσεων που διαθέτει το πακέτο, να παρακολουθεί την εκτέλεση του. Το bMASTER Enterprise Banking System χρησιμοποιεί τις πιο σύγχρονες τεχνολογίες για να επιτύχει την ευελιξία και δυναμική που απαιτείται για την λειτουργία των σύγχρονων χρηματοπιστωτικών ιδρυμάτων.

Το **bMASTER Enterprise Banking System**[62] διαθέτει ένα μεγάλο εύρος υποσυστημάτων που προσφέρουν ολοκληρωμένη λειτουργικότητα για πολλαπλά είδη τραπεζικών εργασιών[63], όπως Διαχείριση Πελατών, Διαχείριση Προϊόντων, Καταθέσεις, Εισπράξεις-Πληρωμές, Πάγιες Εντολές, Συμβάσεις / Όρια Πιστοδοτικών Προϊόντων, Καλύμματα, Χορηγήσεις, Κίνηση Κεφαλαίων, Αγοραπωλησία Συναλλάγματος, Ενέγγυες Πιστώσεις, Εγγυητικές Επιστολές, Γενική Λογιστική, Υποστήριξη Teller/Καταστήματος, Διαχείριση Παραμέτρων, Διοικητική Πληροφόρηση (MIS), κ.α. Στην παρακάτω εικόνα παρουσιάζεται όλη η λειτουργικότητα του συστήματος bMASTER.



<u>Εικόνα 3</u> «Η αρχιτεκτονική του συστήματος bMASTER»

Πηγή : «Uni.systems s.a. 2008, *bMaster Enterprise Banking System - Solution Overview*»

Στο συγκεκριμένο παράδειγμα ελέγχου γίνεται χρήση της εφαρμογής ηλεκτρονική μεταφορά χρημάτων (Electronic Funds Transfer – EFT) που ανήκει στο σύστημα[64] *αυτό.*

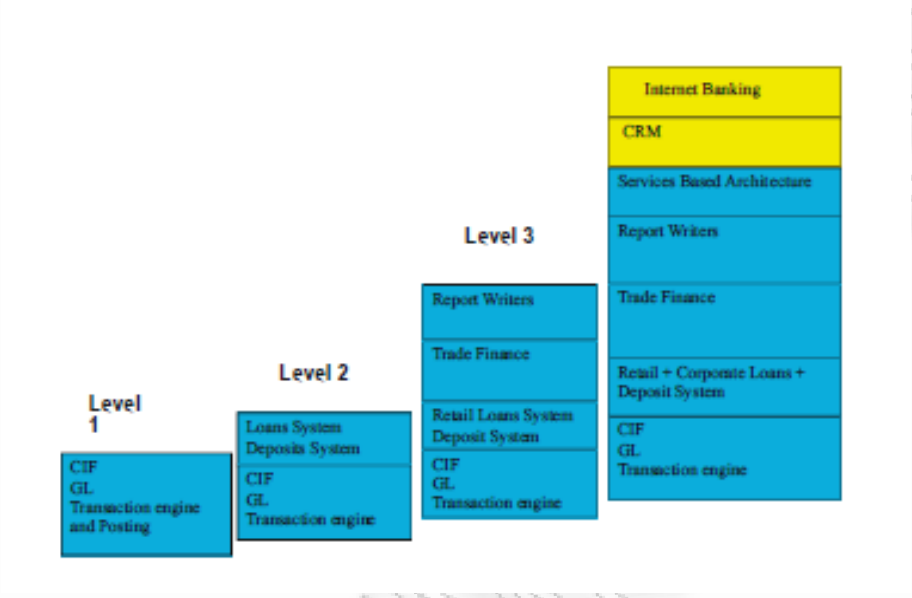
8.2. Τα στοιχεία του συστήματος

Σε γενικές γραμμές ένα χαρακτηριστικό παραδοσιακό κεντρικό τραπεζικό σύστημα θα αποτελούταν από πέντε συστατικά:

1. Αρχείο πληροφορίας πελάτη
2. Γενικό καθολικό σύστημα
3. Επεξεργασία συναλλαγής
4. Σαφή σχέδια δανείων και καταθέσεων
5. Βασική έκθεση αναφοράς για MIS

Με βάση τα συστήματα [65]που είναι διαθέσιμα στην αγορά, έχει γίνει ταξινόμηση σε πέντε επίπεδα, πρώτιστα βάσει της λειτουργικότητας που προσφέρεται με το επίπεδο ένα που είναι το παραδοσιακότερο σύστημα που προσφέρει τη βασική

λειτουργικότητα και το επίπεδο 5 που είναι τα πιο πρόσφατα συστήματα κατάστασης προόδου με προηγμένα χαρακτηριστικά γνωρίσματα[66].



Σχήμα 4 «Ταξινόμηση συστημάτων με βάση την λειτουργικότητα»

Πηγή : «Uni.systems s.a. 2008, *bMaster Enterprise Banking System - Solution Overview*»

Τα συστήματα έχουν ταξινομηθεί βάσει της διαθέσιμης λειτουργικότητας για την κάλυψη των αναγκών των διάφορων επιχειρησιακών γραμμών. Τα περισσότερα συστήματα από το επίπεδο 3 και μετά μπορούν να ταξινομηθούν ως ώριμα συστήματα, τα οποία προσφέρουν μια μέτρια διάδοση της λειτουργικότητας που απαιτείται για να τρέξει αποτελεσματικά μια μεσαίου μεγέθους τράπεζα. Τα συστήματα που εμπίπτουν στο επίπεδο 4 και πιο πέρα μπορούν να ταξινομηθούν ως πιο προηγμένα συστήματα - που στοχεύουν πρώτιστα στη λειτουργία μεγάλων τραπεζών σε πολλές επιχειρησιακές γραμμές και προσφέρουν μια μεγάλη κλίμακα προϊόντων και υπηρεσιών στους πελάτες τους. Αυτές οι τράπεζες θα ήταν χαρακτηριστικά μεταξύ των μεγαλύτερων τραπεζών στη χώρα ή στον τομέα που λειτουργούν.

8.3. Σύστημα EFT

Το ηλεκτρονικό σύστημα μεταφοράς χρημάτων αναφέρεται σε μια κλάση από συστήματα που παρέχουν λειτουργίες για online συναλλαγές όπως η μεταφορά χρημάτων μεταξύ των τραπεζικών λογαριασμών, των ηλεκτρονικών πληρωμών, των

χρηματαποστολών, των ελέγχων διαθέσιμου υπόλοιπου, της διαχείρισης επιταγών, των εγγυητικών επιστολών και γενικότερα τραπεζικών εμβασμάτων.

Το bMASTER ενσωματώνει μέσω του EFT την ηλεκτρονική κίνηση κεφαλαίων που μπορεί να υποστηρίξει μεταφορές κεφαλαίων από οποιοδήποτε τύπο πηγής σε οποιοδήποτε τύπο προορισμού με οποιοδήποτε πιθανό συνδυασμό. Όσον αφορά στην κίνηση κεφαλαίων εξωτερικού το bMASTER προσφέρει μια κεντροποιημένη διαδικασία επεξεργασίας όλων των εισερχόμενων και εξερχόμενων εντολών πληρωμής. Τα συστήματα κίνησης κεφαλαίων που καλύπτονται είναι τα εξής:

TARGET II

EBA/EURO1

EBA/STEP1

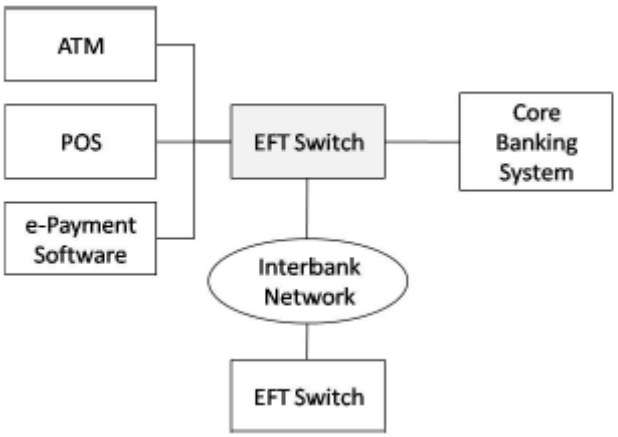
EBA/STEP2

DIAS

SEPA

Από τη μια μεριά το EFT περιλαμβάνει τερματικά (όπως ATMs, POS συσκευές) και τις εφαρμογές ηλεκτρονικών-πληρωμών. Αυτά τα συστατικά παρέχουν τη λειτουργικότητα EFT στους τελικούς χρήστες. Από την άλλη μεριά, το EFT περιλαμβάνει τα κεντρικά τραπεζικά συστήματα που διαχειρίζονται τους τραπεζικούς λογαριασμούς στις συναλλαγές. Η αλληλεπίδραση μεταξύ αυτών των δύο πλευρών γίνεται συνήθως μέσω ενός συστήματος λογισμικού που ονομάζεται EFT. Όταν δύο διαφορετικά όργανα συμμετέχουν σε μια συναλλαγή, επικοινωνούν μέσω ενός περιφερειακού ή εθνικού διατραπεζικού δικτύου που ενεργεί ως μεσολαβητής μεταξύ των δύο συστημάτων των οργανισμών. Το σχήμα 1 παρουσιάζει τα βασικά χαρακτηριστικά σε ένα ηλεκτρονικό σύστημα EFT.

Υπάρχουν επίσης λειτουργικότητες όπως η παραμετροποίηση συναλλαγών, πιο συγκεκριμένα: Ο σχεδιασμός του bMASTER προσφέρει στην Τράπεζα έτοιμους παραμετρικούς κανόνες για τον ορισμό νέων επιχειρησιακών συναλλαγών μεγιστοποιώντας την επαναχρησιμοποίηση έτοιμου λογισμικού και ελαχιστοποιώντας τον χρόνο και την άνθρωπο-προσπάθεια υλοποίησης που αφορούν την διαχείριση των συναλλαγών αυτών, κάτι με το οποίο δε θα ασχοληθούμε στην παρούσα εργασία.



Σχήμα 5 Σύστημα EFT

Πηγή : «Uni.systems s.a. 2008, *bMaster Enterprise Banking System - EFT Overview*»

Ο καλύτερος τρόπος για να ελεγχθεί η λογική της διεπαφής και η αποδοτικότητα της ροής εργασίας είναι να περιγραφούν όλες οι εισαγωγές, όλα τα κλίκ του ποντικιού, και όλες οι αντιδράσεις του προγράμματος λεπτομερώς. Η καλύτερη θέση για μια τέτοια λεπτομερή περιγραφή του πρωτοτύπου GUI είναι η έγγραφη περιγραφή της περίπτωσης χρήσης, δεδομένου ότι είναι ευρέως γνωστό πως ο «ύποπτος» βρίσκεται στις λεπτομέρειες.

8.4. Έλεγχος συνένωσης, ολοκλήρωσης στην μελέτη περίπτωσης

Η ολοκλήρωση συστημάτων είναι μια συστηματική προσέγγιση που δημιουργεί την πλήρη δομή λογισμικού μαζί με τις διεπαφές όπως διευκρινίζεται και στο σχέδιο των ελεγμένων τμημάτων των εφαρμογών. Κατά τη διάρκεια του ελέγχου ολοκλήρωσης των συστημάτων, οι έλεγχοι διευθύνονται για να βρουν τα λάθη που σχετίζονται με την διεπαφή. Ο σκοπός εδώ είναι να εξασφαλιστεί, από όλες τις ενότητες που μπορούν να προσαρμοστούν σύμφωνα με την εργασιακή απαίτηση της τράπεζας, μια πλήρη ομοφωνία σε ένα ενσωματωμένο σενάριο. Το περιβάλλον ελέγχου συστημάτων χρειάζεται το κατάλληλο υλικό, το λογισμικό συστημάτων και οποιοδήποτε άλλο λογισμικό για να υποστηρίξει την ολοκλήρωση σε ένα «ζωντανό» περιβάλλον παραγωγής. Ο στόχος είναι να γίνει ο έλεγχος συστημάτων σε ένα περιβάλλον που είναι όσο το δυνατόν πιο κοντά στο περιβάλλον παραγωγής.

8.4.1. Έλεγχος UAT της εφαρμογής EFT

Ο έλεγχος αποδοχής χρηστών είναι ένα έλεγχος λειτουργίας (για να επικυρώσει το προϊόν λογισμικού σε σχέση με την προδιαγραφή των απαιτήσεων και τον έλεγχο ολόκληρου του συστήματος. Προσπαθεί να παρουσιάσει αποκλίσεις μεταξύ της ιδιότητας του προϊόντος και της απαίτησης). Γενικά η κεντρική ομάδα από την πλευρά της τράπεζας αναλαμβάνει τον έλεγχο αποδοχής χρηστών.

8.5. Προκλήσεις εφαρμογής

Η διαδικασία εφαρμογής κατά έναν μέσο όρο μπορεί να πάρει απο 6 μήνες μέχρι ένα έτος αφού εξαρτάται από το βαθμό προσαρμογής που απαιτείται. Εάν ο προμηθευτής έχει σε ισχύ καλές διαδικασίες εφαρμογής, ο χρόνος εφαρμογής μπορεί να μειωθεί. Γενικά επιλέγονται μερικά καταστήματα που δικτυώνονται στο πλαίσιο του νέου συστήματος, και μόλις εγκατασταθούν όλα τα στοιχεία, θα επεκταθεί αργά και σε άλλα καταστήματα της τράπεζας. Αυτή η διαδικασία καλείται ‘going live’ που σημαίνει ότι το σύστημα τέθηκε σε παραγωγή.

Μια άλλη πρόκληση είναι η εισαγωγή των κληρονομημένων στοιχείων στο νέο σύστημα. Δεδομένου ότι τα δεδομένα είναι πολύ σημαντικά και απόρρητα σε μια τράπεζα είναι πολύ σημαντικό τα στοιχεία να μεταφερθούν από το τρέχον σύστημα επιτυχώς στο νέο σύστημα. Πρέπει να δοθεί προσοχή κατά την εισαγωγή. Πριν από την εφαρμογή η τράπεζα μπορεί να χρειαστεί να μοιραστεί μερικά από τα δεδομένα της με τον συνεργάτη εφαρμογής για λόγους ελέγχου. Η καλύτερη δυνατή διαχείριση είναι μια άλλη κρίσιμη πτυχή για μια επιτυχή εφαρμογή. Η κύρια διαχείριση πρέπει να είναι απόλυτα δεκτική και θετική με το έργο. Πρέπει να τοποθετήσει τους καλύτερους υπαλλήλους του από το τμήμα Πληροφορικής και το επιχειρησιακό τμήμα στο έργο και πρέπει να παραμείνουν μέχρις ότου ολοκληρωθεί. Μια άλλη σημαντική πτυχή από μια ινδική προοπτική είναι η διαχείριση που πρέπει επίσης να καθησυχάσει οποιουσδήποτε φόβους των υπαλλήλων για περικοπή των δαπανών. Όλοι οι ενδιαφερόμενοι υπάλληλοι πρέπει να ενημερωθούν και να εκπαιδευτούν κατάλληλα ώστε να δεχτούν το νέο σύστημα.

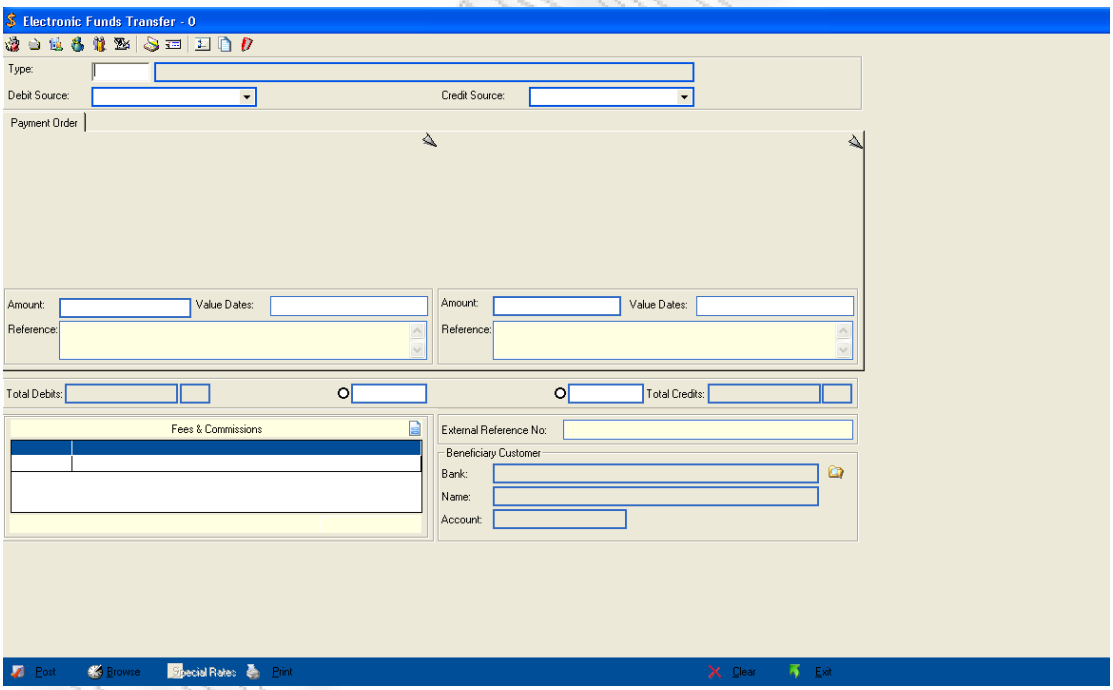
8.6. Σενάριο ελέγχου συστήματος EFT

Κατά την διαδικασία ελέγχου λογισμικών συστημάτων, επιβάλλεται να υπάρχει μια συλλογή από σενάρια ελέγχου (test cases) σκοπός της οποίας είναι για να χρησιμοποιηθεί και να δοκιμάσει το υπό έλεγχο λογισμικό σύστημα έτσι ώστε να αποδειχτεί ότι το σύστημα αυτό συμπεριφέρεται όπως αναμενόταν. Τα σενάρια

αυτά περιέχουν λεπτομερείς οδηγίες για την λειτουργία του λογισμικού συστήματος για το οποίο καταγράφονται.

Υπάρχουν δύο είδη σεναρίων ελέγχου α) τα τυπικά σενάρια ελέγχου και β) τα άτυπα σενάρια ελέγχου. Η διαφορά αυτών των δύο τρόπων δημιουργίας και καταγραφής σεναρίων ελέγχου βρίσκεται στο γεγονός ότι στα τυπικά σενάρια ελέγχου είναι γνωστή η είσοδος και το αναμενόμενο αποτέλεσμα. Αυτά τα χαρακτηριστικά αναγράφονται στα σενάρια ελέγχου και συμβολίζονται με τα αναγνωριστικά pre-conditions και post-conditions ανάλογα.

Τα σενάρια όπως αναφέρθηκε και πιο πάνω δημιουργούνται αυτόματα με την εκτέλεση των προδιαγραφών. Για την δημιουργία του απαιτείται κάποια μικρή μετατροπή στο κώδικα των προδιαγραφών. Η μετατροπή αυτή γίνεται εύκολα αφού χρειάζεται μόνο μια μικρή προσθήκη κάποιων γραμμών κώδικα σε κάθε αναπαράσταση στοιχείου γραφικού περιβάλλοντος που βρίσκεται στις προδιαγραφές.



Εικόνα 4«Αρχική οθόνη του συστήματος»

Κατά την διάρκεια του σεναρίου, η παροχή της λεπτομέρειας στις σημαντικότερες ή στις δύσκολες περιπτώσεις χρήσης είναι μια πιθανή μορφή ελέγχου μέσα στην αναπτυξιακή διαδικασία. Αυτός ο τύπος ελέγχου μπορεί να εφαρμοσθεί πολύ πριν τεθεί ουσιαστικά υπο δοκιμή η εφαρμογή. Μπορούν να χρησιμοποιηθούν μερικά από τα πρότυπα ελέγχου κατά την περιγραφή των πιθανών εισαγωγών και των

παραγόμενων αντιδράσεων. Παραδείγματος χάριν, μπορείτε να εστιάσετε στις χαμηλότερες και πιθανότερες εισαγωγές, στους απαιτούμενους τομείς, στις προαιρετικές εισαγωγές, και ούτω καθεξής. Με αυτόν τον τρόπο, παρέχεται μια πολύ σαφέστερη εικόνα των απαραίτητων λειτουργιών του προγράμματος από ότι θα παρεχόταν από μια ανάλυση περίπτωσης χρήσης.

Type
Σε αυτό το πεδίο εισάγεται ο κωδικός της κίνησης που θέλει ο χρήστης να κάνει. Στην περίπτωση της απλής μεταφοράς χρημάτων εισάγεται ο κωδικός «300». Όταν επιλεχθεί ένας τύπος κίνησης, εμφανίζονται τα απαραίτητα στοιχεία της κίνησης που πρέπει να συμπληρωθούν στη φόρμα τα οποία είναι τα παρακάτω:

Debit Source
Σε αυτό το πλαίσιο εμφανίζεται το είδος του λογαριασμού που θα γίνει η κίνηση , δηλαδή αν είναι καταθετικός απλός ή δάνειο.

Payment Order, Balance
Εδώ εισάγονται τα στοιχεία του πελάτη και επιλέγεται ο λογαριασμός που θέλει ο πελάτης να χρεώσει προκειμένου να κάνει την μεταφορά των χρημάτων.
Πιο συγκεκριμένα τα πεδία που εμφανίζονται εδώ περιγράφονται παρακάτω:

Ιδιοκτήτης. Ο AccOwner του λογαριασμού είναι ο χρήστης που θα χρεωθεί το ποσό της μεταφοράς στο λογαριασμό του.

Λογαριασμός. Είναι ο λογαριασμός του μέρους χρεώσεων που σημαίνει ότι θα χρεωθεί όταν εκτελεστεί η συναλλαγή. Το Combo-box παρουσιάζει όλους τους διαθέσιμους λογαριασμούς του AccOwner που έχει επιλεχτεί ενεργώντας ως φίλτρο για τους διαθέσιμους λογαριασμούς προς επιλογή.

Νόμισμα. Είναι το νόμισμα του λογαριασμού. Αυτό το πεδίο είναι εκτός λειτουργίας και είναι το νόμισμα με το οποίο ανοίχτηκε ο λογαριασμός.

Υπόλοιπο. Είναι το τρεχούμενο υπόλοιπο του λογαριασμού.

Διαθέσιμο. Είναι το διαθέσιμη υπόλοιπο του λογαριασμού. Αυτή η ισορροπία παρουσιάζεται επίσης στην εφαρμογή διευθυντών απολογισμού

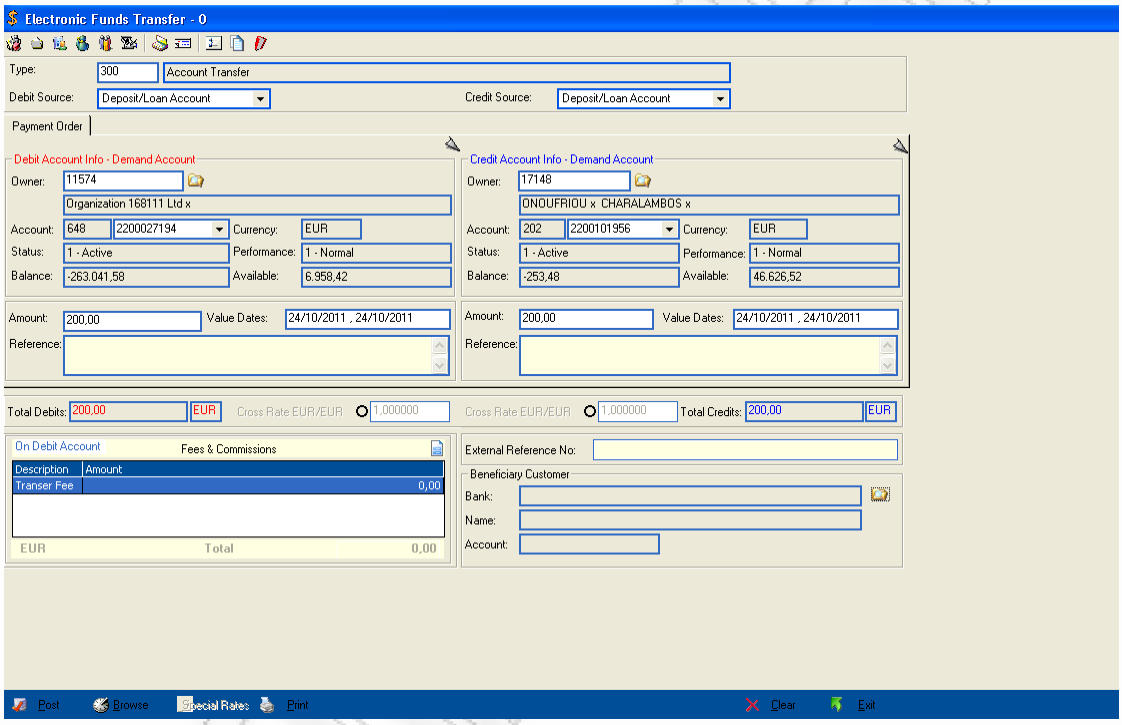
Ποσό. Είναι το ποσό που μεταφέρεται στο νόμισμα του λογαριασμού. Αυτό δεν είναι απαραιτήτως το ποσό με το οποίο ο λογαριασμός θα χρεωθεί. Σε περίπτωση που υπάρχουν προμήθειες, το ποσό που μεταφέρεται είναι άλλο.

Value Dates. Είναι οι ημερομηνίες που δείχνουν πότε θα είναι διαθέσιμα τα χρήματα στον πιστωτικό λογαριασμό.

Αναφορά. Είναι ένα πεδίο κειμένου όπου ο χρήστης μπορεί να εισάγει σχόλια σχετικά με πληροφορίες της συναλλαγής.

Σημείωση: Ο χρήστης πρέπει να συμπληρώσει επίσης το πιστωτικό μέρος δεξιά στο πλαίσιο της εφαρμογής έτσι ώστε να γίνει ο υπολογισμός των προμηθειών και να μπορεί να υπολογιστεί το τελικό μεταφερόμενο ποσό. Οι προμήθειες είναι βασισμένες στο συνδυασμό των μερών των χρεώσεων και των πιστώσεων. Οι συναλλαγές του EFT είναι συνδυασμένες συναλλαγές που συντίθενται από 2 άλλες περιπτώσεις στις οποίες η μια θα χρεώσει το μέρος των χρεώσεων και η άλλη θα πιστώσει το πιστωτικό μέρος των (λογαριασμοί ή μετρητά). Για να συναχθεί αυτό το συνδυασμένο γεγονός το σύστημα πρέπει να ξέρει εξαρχής τα χρεωστικά και πιστωτικά μέρη.

Ένα παράδειγμα φαίνεται παρακάτω:



Εικόνα 5«Παράδειγμα λειτουργίας του συστήματος»

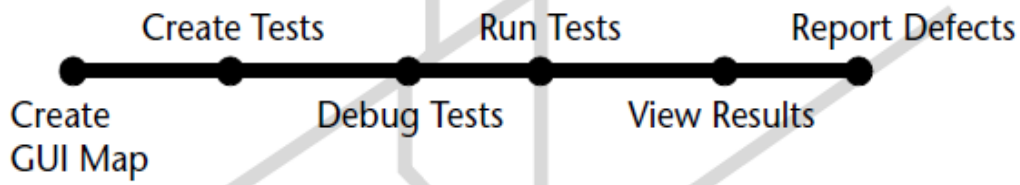
Στο παραπάνω παράδειγμα ο χρήστης εισάγει το ποσό 200 EUR. Μόνο αφού συμπληρώσει το δεξί μέρος (Credit Account Info – Demand Account) η κίνηση ορίζεται πλήρως και το σύστημα υπολογίζει το ποσό που πρέπει να μεταφερθεί, στην συγκεκριμένη περίπτωση το ποσό της προμήθειας είναι 0.

Παρακάτω παρουσιάζεται το εργαλείο ελέγχου που χρησιμοποιήθηκε για να ελεγχτεί το παραπάνω σύστημα.

8.7. Διαδικασία ελέγχου Win Runner

Το WinRunner[67] παρέχει μια ολοκληρωμένη λύση για όλα τα στάδια ελέγχου : σχεδιασμός ελέγχου, ανάπτυξη ελέγχου, έλεγχος GUI και απόδοσης συστήματος, ανίχνευση λαθών και έλεγχος απόδοσης στη περίπτωσης πολλαπλής χρήσης του συστήματος.

Η διαδικασία ελέγχου με το εργαλείο WinRunner περιλαμβάνεις τα εξής στάδια:



Εικόνα 6 «Ροή διαδικασίας ελέγχου με το εργαλείο WinRunner»

Πηγή: «Mercury Interactive Corporation, 2003 *WinRunner User's Guide, Version 7.6*»

Δημιουργία του GUI Map File: Με τη δημιουργία του χάρτη GUI ο WinRunner μπορεί να προσδιορίσει τα αντικείμενα GUI στην εφαρμογή που πρόκειται να εξεταστεί.
Δημιουργία Script Ελέγχου: Αυτή η διαδικασία περιλαμβάνει την καταγραφή, τον προγραμματισμό ή και τα δύο. Κατά τη διάρκεια της διαδικασίας καταγραφής των ελέγχων, θα πρέπει να εισαχθούν σημεία ελέγχου (checkpoints) όπου η αντίδραση της εφαρμογής θα πρέπει να εξεταστεί.

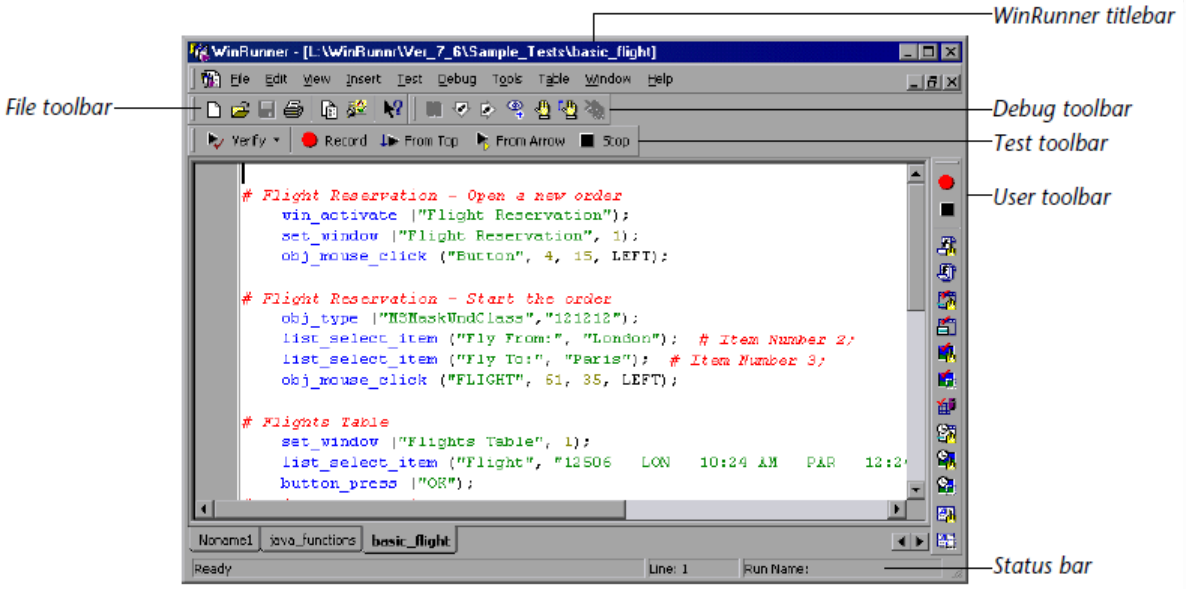
Έλεγχος αποσφαλμάτωσης: Εκτέλεση των ελέγχων σε κατάσταση Debug για να σιγουρευτεί ο χρήστης ότι εκτελούνται ομαλά.

Εκτέλεση των ελέγχων: εκτέλεση ελέγχων σε κατάσταση επιβεβαίωσης για να ελεγχθεί η εφαρμογή.

Παρουσίαση αποτελεσμάτων: Αυτό καθορίζει την επιτυχία ή την αποτυχία των δοκιμών.

Αναφορά λαθών: Εάν η εκτέλεση ενός συγκεκριμένου έλεγχου αποτύχει λόγω λάθους στην εφαρμογή που ελέγχεται, τα λάθη μπορούν να αναφερθούν άμεσα μέσω του Test Results window.

Η αρχική εικόνα είναι



Εικόνα 7 «Ανάλυση της διεπαφής του WinRunner»

8.7.1. Πώς αναγνωρίζει τα αντικείμενα ο Win Runner

Οι εφαρμογές GUI φτιάχνονται από αντικείμενα GUI όπως είναι τα παράθυρα, τα κουμπιά, οι λίστες- κατάλογοι και τα μενού-επιλογές. Ο WinRunner's Rapid Test Script Wizard μαθαίνει τις περιγραφές όλων των αντικειμένων GUI και σώζει την περιγραφή του GUI αντικειμένου στο GUI Map file.gui, που είναι ο κορμός του Win Runner. Όταν εκτελούνται οι έλεγχοι, ο WinRunner χρησιμοποιεί αυτό το αρχείο για να προσδιορίσει και να εντοπίσει τα αντικείμενα.

8.7.2. Δημιουργία του αρχείου χαρτών GUI και η φόρτωση του

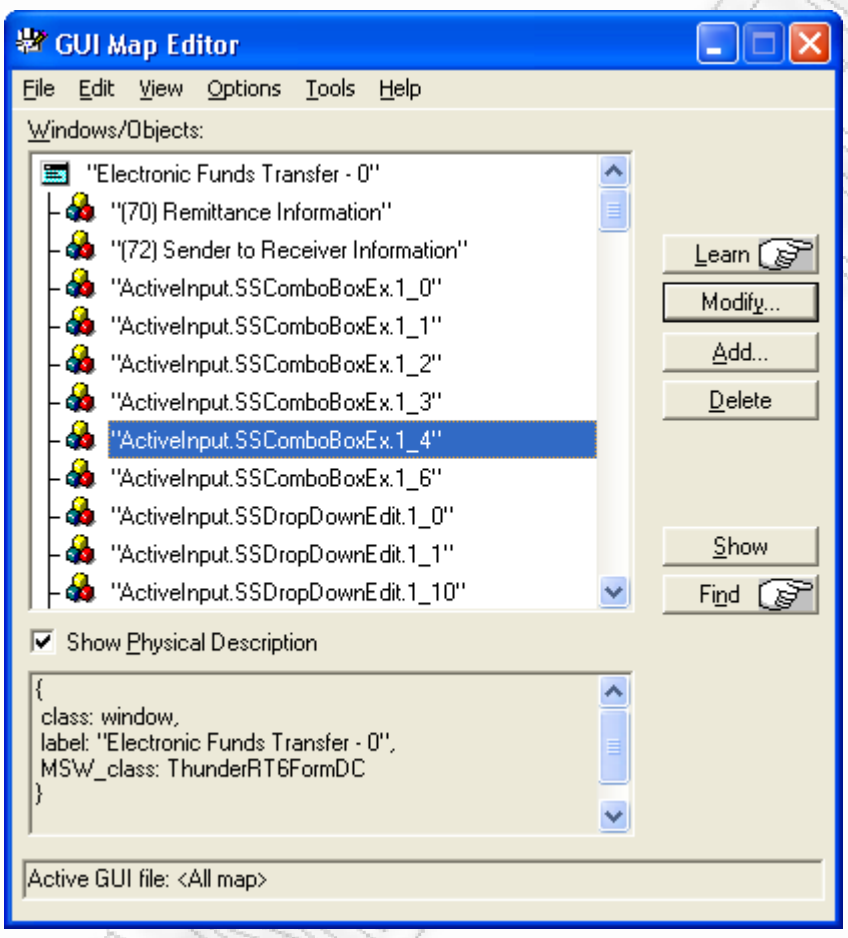
(είναι ένα αρχείο που αποθηκεύονται οι ιδιότητες του αντικειμένου που ελέγχεται)

Ο σκοπός ύπαρξης ενός εργαλείου mapping είναι να μειώσει στο ελάχιστο την χειρωνακτική εργασία που πρέπει να κάνει ο προγραμματιστής κατά την διάρκεια

του ελέγχου. Στο σύστημα που υλοποιήθηκε έγινε εξάλειψη της οποιασδήποτε συμμετοχής χρήστη κατά την διαδικασία εκτέλεσης των σεναρίων ελέγχου.

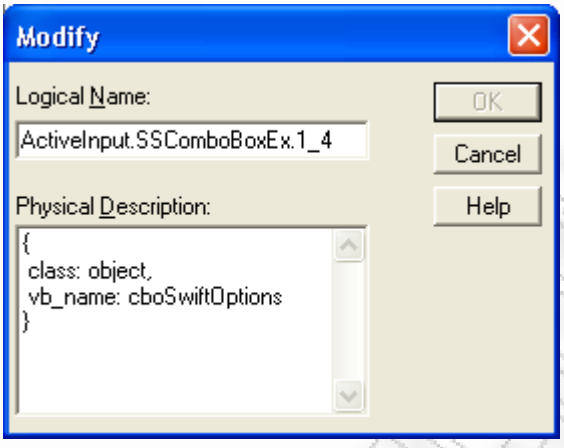
Υπάρχουν τρεις τρόποι δημιουργίας του αρχείου χαρτών GUI (GUI Map file)
Rapid Test Script Wizard: ανοίγει συστηματικά τα παράθυρα στην εφαρμογή σας και μαθαίνει μια περιγραφή του κάθε αντικειμένου του GUI.

- ✓ Χρησιμοποιείται για να μάθει ολόκληρη την εφαρμογή
- ✓ Έλεγχος της διεπαφής του χρήστη.



Η καταγραφή: προσθέτει παράθυρα και αντικείμενα στον χάρτη του GUI όπως αυτά ανιχνεύονται από το χρήστη.

Ο συντάκτης χαρτών GUI (**GUI Map Editor**) χρησιμοποιείται για να αποθηκεύσει όλες τις πληροφορίες για τα στοιχεία GUI που βρίσκονται στην εφαρμογή. Στην περίπτωση μας η εφαρμογή αυτή είναι το EFT όπου μπορούν να τροποποιηθούν οι φυσικές περιγραφές των αντικειμένων. Το εργαλείο σύνταξης χαρτών GUI μπορεί να χρησιμοποιηθεί για να τροποποιήσει εύκολα τις πληροφορίες στο αρχείο χαρτών. Το αρχείο χαρτών GUI μπορεί να φορτωθεί κατευθείαν από τον :
GUI Map editor / GUI_load("filename.gui'')



8.8. Καταγραφή ελέγχου

Με την καταγραφή, μπορούμε να δημιουργήσουμε γρήγορα τα αυτοματοποιημένα scripts ελέγχου, επιλέγοντας τα αντικείμενα με το ποντίκι, και με την εισαγωγή δεδομένων από το πληκτρολόγιο.
Η καταγραφή παράγει δηλώσεις στην TSL (Test Script Language), στην Mercury's interactive Test Script Language και είναι Case sensitive.

Τα σενάρια ελέγχου που παράγονται μπορούν να είναι είτε τυχαία είτε συγκεκριμένα, και δημιουργούνται με βάση ενός test string το οποίο δημιουργείται αυτόματα, είτε εισάγεται από τον χρήστη του συστήματος. Στην περίπτωση της αυτόματης και τυχαίας δημιουργίας του test string λαμβάνονται υπόψη δυο παράμετροι. Η πρώτη παράμετρος αφορά μια λίστα αντικειμένων στην οποία αποθηκεύονται τα στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος τα οποία ο χρήστης θέλει να συμπεριλάβει στον έλεγχο. Η δεύτερη παράμετρος αφορά το μήκος του σεναρίου ελέγχου που θέλει να δημιουργήσει. Με την λέξη μήκος εννοείται ο αριθμός των στοιχείων του γραφικού περιβάλλοντος που θέλει ο χρήστης να χειριστεί κατά την διάρκεια του έλεγχου. Για παράδειγμα στην συγκεκριμένη εφαρμογή που θα ελεγχθεί, εάν ο χρήστης δώσει παράμετρο 10,58 KB , τότε στην εκτέλεση του σεναρίου θα αλλάξουν οι διαστάσεις του παραθύρου της εφαρμογής.

Για παράδειγμα :

```
set_window ("WorkerW_1", 2);
      list_select_item ("SysListView32", "bMASTER;1,58 KB");
      list_activate_item ("SysListView32", "bMASTER;1,58 KB");
```

Όπου ορίζονται οι διαστάσεις του παραθύρου της εφαρμογής EFT.

8.8.1. Επιλογή του τρόπου καταγραφής

Προτού αρχίσετε την καταγραφή ενός ελέγχου, πρέπει να επιλέξετε τον κατάλληλο τρόπο καταγραφής.
Υπάρχουν δύο διαθέσιμοι τρόποι καταγραφής.

- Ο τρόπος που βασίζεται στο περιεχόμενο – που καταγράφει την εκτέλεση που κάνει ο χρήστης από την άποψη των αντικειμένων GUI.

  π.χ. button_press («Ok»)

- Αναλογικός τρόπος - τα αρχεία συντονίζουν τις εισαγωγές από τα ποντίκια και τα πληκτρολόγια π.χ Mtype (“<kleft>+”).

8.9. Εκτελώντας τον έλεγχο διεπαφής

Το WinRunner παρέχει τρεις τρόπους και τους οποίους χρησιμοποιήσαμε για την εκτέλεση ελέγχου:

- **Verify mode :** κατά την εκτέλεση ενός ελέγχου ελέγχεται η συμπεριφορά της εφαρμογής και όταν ο χρήστης θελήσει αποθηκεύει το αποτέλεσμα του ελέγχου.

- **Debug Mode:** ο χρήστης ελέγχει ότι το script ελέγχου εκτελείται ομαλά χωρίς λάθη στη σύνταξη. Η «κατάσταση αποσφαλμάτωσης» δεν θα παράγει τα αποτελέσματα του ελέγχου.

- **Update mode:** ο χρήστης δημιουργεί νέα προσδοκώμενα αποτελέσματα για ένα GUI check point ή ένα bitmap check point.

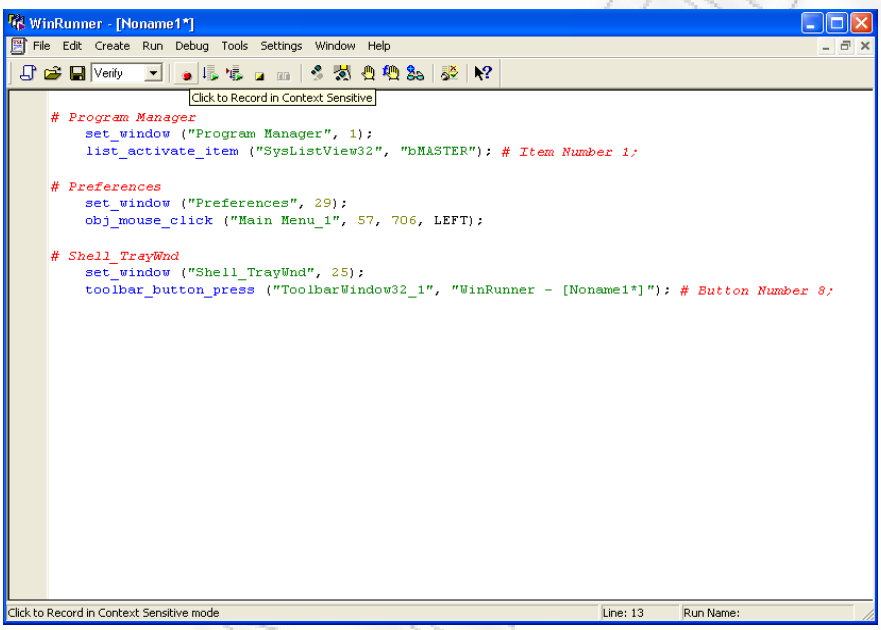
8.10. Καταγραφή ελέγχου EFT

Στο παράδειγμα μας αρχικά «δείχνουμε» στο εργαλείο τι ακριβώς θέλουμε να κάνει. Δηλαδή όπως προανέφερα και στην ενότητα (3.4 ελέγχου record / playback) με μια απλή περιδιάβαση το εργαλείο ηχογραφεί τις χειρονακτικές αλληλεπιδράσεις του χρήστη με το υπό έλεγχο λογισμικό σύστημα.

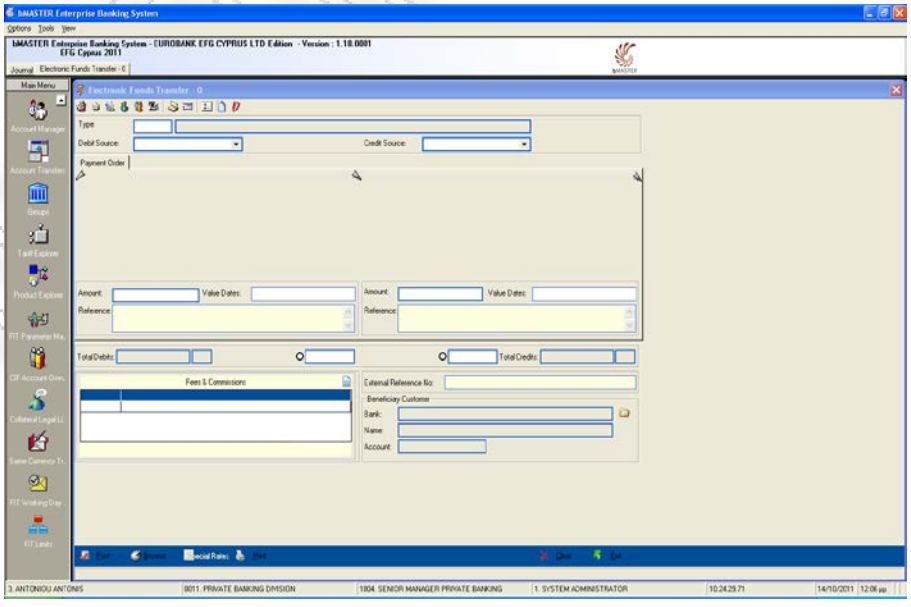
Έλεγχος Λογισμικού

Μετά το πέρας της ηχογράφησης των αλληλεπιδράσεων με το υπό έλεγχο λογισμικό σύστημα το σύστημα δίνει την επιλογή να γίνει εκτέλεση των ηχογραφημένων αλληλεπιδράσεων.

Σαν πρώτο βήμα, λοιπόν θα πρέπει να ανοίξουμε την πλατφόρμα του bMaster και να επιλέξουμε την εφαρμογή που μας ενδιαφέρει , Account Transfers.



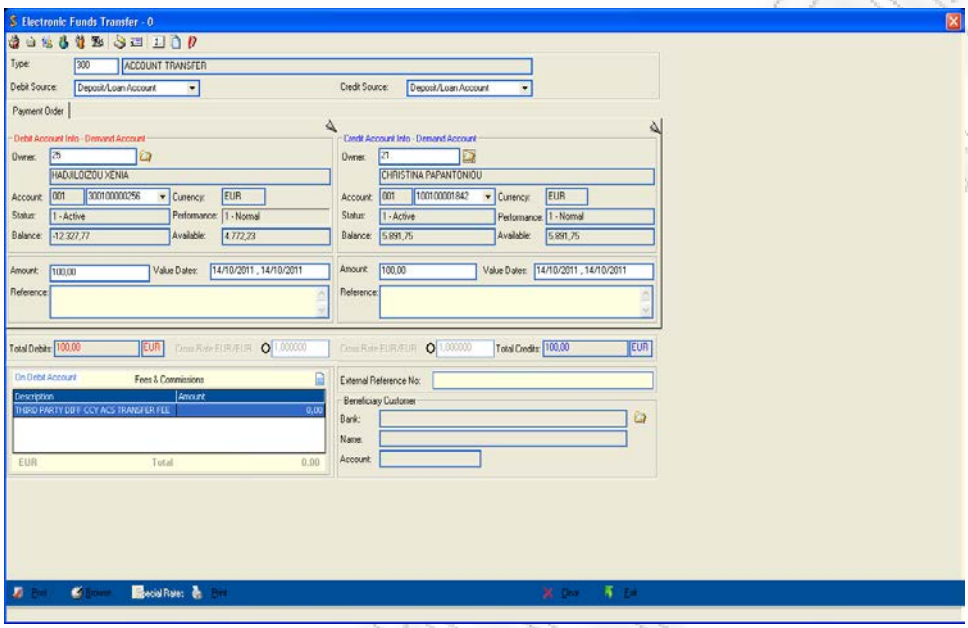
Εδώ παρουσιάζεται η εικόνα που εμφανίζεται όταν επιλέξουμε την επιλογή εκτέλεση του test στην μπάρα επιλογών του Win Runner.



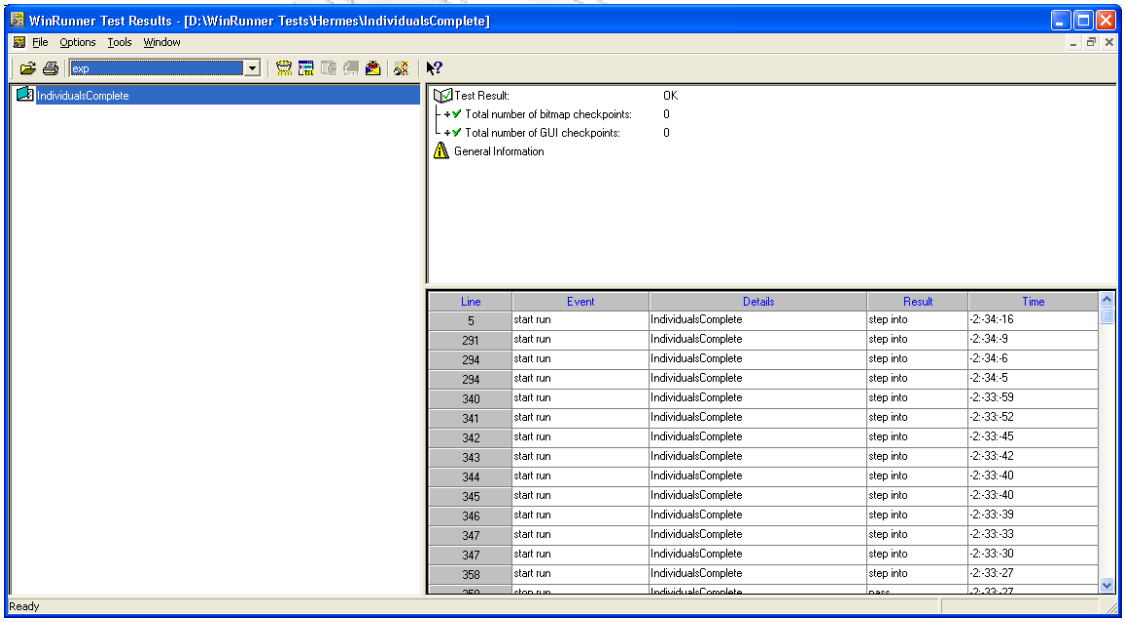
Έλεγχος Λογισμικού

Στη συνέχεια ο χρήστης «δείχνει» στο εργαλείο ποια είναι τα απαραίτητα πεδία που θα πρέπει να συμπληρωθούν προκειμένου να κάνει μια απλή μεταφορά μεταξύ λογαριασμών (βλέπε ενότητα 7.3 με πεδία). Αφού λοιπόν το σύστημα καταγράψει ποια πεδία πρέπει να συμπληρωθούν και πως (excel αρχείο εισαγωγή δεδομένων) εκτελεί τον έλεγχο αυτόματα όταν ο χρήστης επιλέξει εκτέλεση ελέγχου.

Και παρουσιάζεται η παρακάτω εικόνα στην εφαρμογή μας

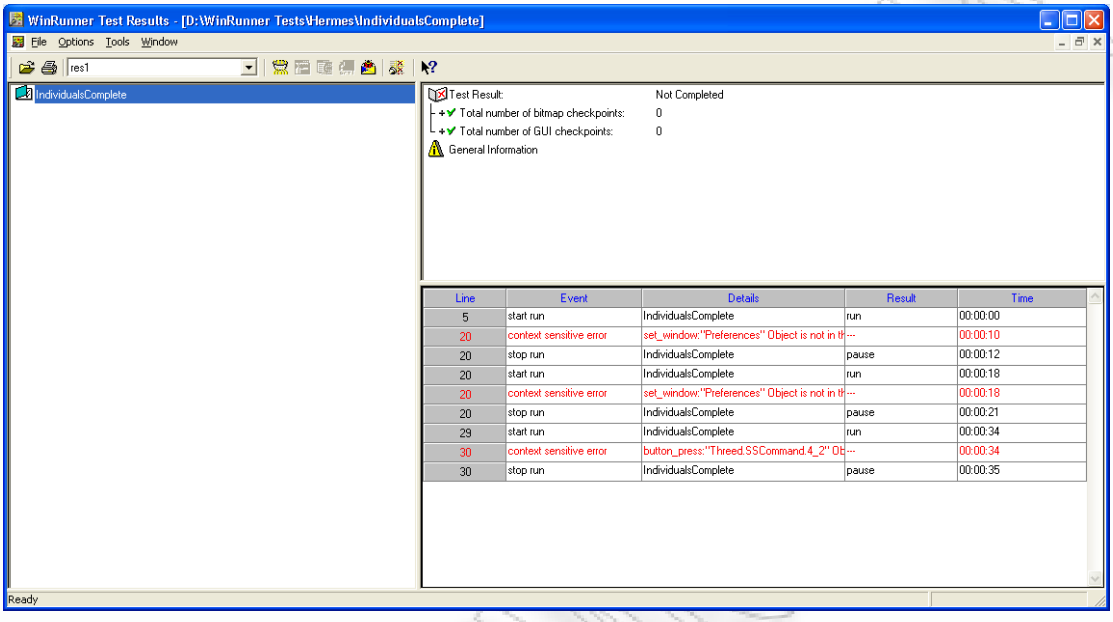


Όταν το σύστημα ολοκληρώσει επιτυχώς τον έλεγχο εμφανίζονται τα παρακάτω:

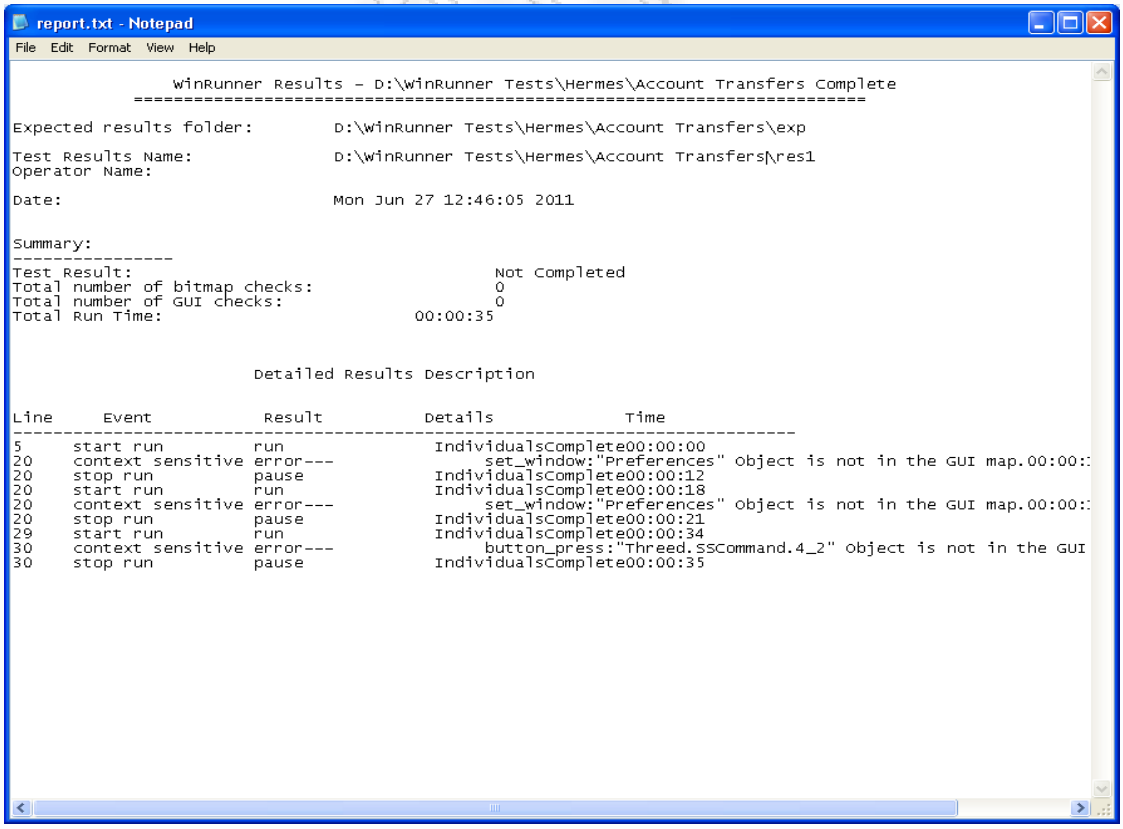


<u>Εικόνα 8</u> «Αποτελέσματα ελέγχου του συστήματος».

Ενώ στην περίπτωση που εμφανιστούν λάθη παρουσιάζονται αναλυτικά ως εξής:



<u>Εικόνα 9</u> «Αποτελέσματα ελέγχου του συστήματος στην περίπτωση ανίχνευσης λαθών».

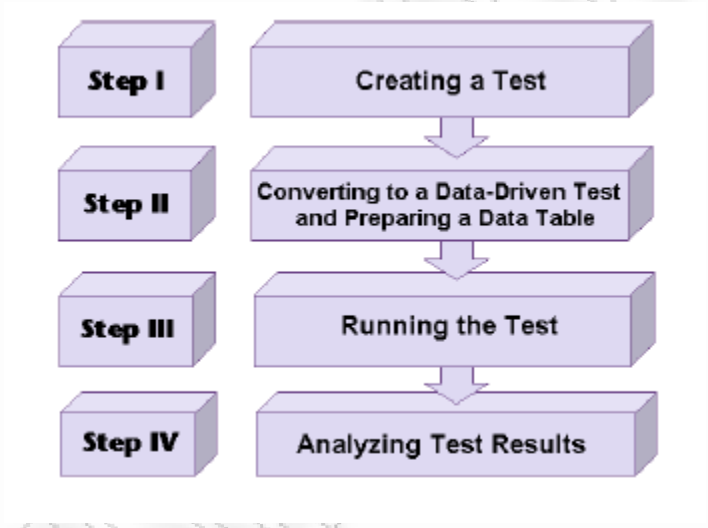


<u>Εικόνα 10</u> «Αρχείο log καταγραφής λαθών».

8.11. Έλεγχος συστήματος EFT με την μέθοδο data driven.

Κατά τον έλεγχο της εφαρμογής, ίσως να χρειαστεί να εξεταστεί το πώς εκτελεί η εφαρμογή την ίδια λειτουργία με πολυάριθμα σύνολα δεδομένων. Σε αυτήν την περίπτωση χρησιμοποιείται ο έλεγχος που βασίζεται στα δεδομένα (Data Driven Test). Με την αντικατάσταση των σταθερών τιμών στον έλεγχο με τιμές που αποθηκεύονται σε έναν πίνακα δεδομένων (ένα εξωτερικό αρχείο), είναι δυνατό να παραχθούν πολλά σενάρια ελέγχου χρησιμοποιώντας τον ίδιο έλεγχο.

Το παρακάτω διάγραμμα παρουσιάζει τα στάδια της διαδικασίας ελέγχου data-driven στο WinRunner:



Διάγραμμα 5 Στάδια διαδικασίας ελέγχου data-driven

Πηγή: «Mercury Interactive Corporation, 2003 *WinRunner User's Guide, Version 7.6*»

Υπάρχουν δύο τρόποι ελέγχου προσανατολισμένου προς τα δεδομένα:

- ✓ Data driven Wizard
- ✓ Τροποποίηση Script το ελέγχου με το χέρι.

Στην περίπτωση μας :

```
set_window ("Account_Transfers", 100);
      obj_mouse_click ("ActiveInput.SSComboBoxEx.1_3", 63, 11, LEFT);
      obj_type
("SSMultiCombo","<kDown_E><kDown_E><kDown_E><kReturn>");
      wait(1);
#-------------------------------------------------------------------
#
```

```
        tblNames = "D:\\WinRunner
Tests\\Hermes\\IndividualsComplete\\Names.xls";
              ddt_open(tblNames,DDT_MODE_READWRITE);
                    x=int(rand(1)*15);
```

Φόρτωση του xls αρχείου (**Names**.xls). Σε αυτό το αρχείο περιέχονται όλα τα ονόματα και τα στοιχεία των πελατών που θέλουμε το σύστημα να χρησιμοποιήσει κατά τον έλεγχο ώστε να δούμε την συμπεριφορά του συστήματος σε αυτή την περίπτωση.

8.12. Διαδικασία Data Driven ελέγχου EFT

- Δημιουργία ελέγχου
- Μετατροπή ελέγχου σε έναν έλεγχο που οδηγείται από τα δεδομένα
- Προετοιμασία ενός πίνακα δεδομένων
- Εκτέλεση ελέγχου
- Ανάλυση αποτελεσμάτων

Στην περίπτωση ελέγχου του συστήματος EFT.Αρχικά ορίζουμε το αρχείο με τα δεδομένα που θέλουμε να εισάγει το σύστημα με σκοπό να εκτελέσει τον έλεγχο.

Στην συνέχεια ο wizard μας καθοδηγεί στα επόμενα βήματα ώστε να ολοκληρωθεί η διαδικασία ελέγχου.

Στην παρακάτω εικόνα παρουσιάζονται αναλυτικά τα δεδομένα του αρχείου που έγινε εισαγωγή στο σύστημα.

Data Table - D:\WinRunner Tests\Hermes\IndividualsComplete\Names.xls

File Edit Data Format Help

A1 Papadopoulos

	SurNameF	NameF	MiddleNameF	FatherNameF	MotherNameF	SurNameGr	NameGr	MiddleNameG	FatherN
1	Papadopoulos	Nick	Kostas	Andrew	Maria	Παπαδόπουλος	Νίκος	Κώστας	Ανδρέα
2	Papadopoulos	Manos	Panagiotis	George	Afrodite	Παπαδόπουλος	Μάνος	Παναγιώτης	Γεώργι
3	Papadopoulos	Alex	Jim	Akis	Marigo	Παπαδόπουλος	Αλέξης	Δημήτριος	Άκης
4	Papadopoulos	George	Nick	Jim	Anastasia	Παπαδόπουλος	Γεώργιος	Νίκος	Δημήτρ
5	Papadopoulos	Markos	Alex	Manos	Dimitra	Παπαδόπουλος	Μάρκος	Αλέξης	Μάνος
6	Papadopoulos	Hercules	Dias	Kronos	Europe	Παπαδόπουλος	Ηρακλής	Δίας	Κρόνος
7	Papadopoulos	Omiros	Thiseas	Aigeas	Evridiki	Παπαδόπουλος	Όμηρος	Θησέας	Αιγέας
8	Papadopoulos	Odisseas	Ainstain	Adam	Eva	Παπαδόπουλος	Οδυσσέας	Αϊνστάιν	Αδάμ
9	Papadopoulos	Roben	William	John	Katerina	Παπαδόπουλος	Ρομπέν	Γουίλιαμ	Γιάννης
10	Papadopoulos	Adolfos	Rudolf	Hans	Aglaia	Παπαδόπουλος	Αδόλφος	Ρουντολφ	Χανς
11	Papadopoulos	Romeos	Marios	Koulis	Melpomeni	Παπαδόπουλος	Ρωμέος	Μάριος	Κούλης
12	Papadopoulos	Arnold	Rocky	Balboa	Frosini	Παπαδόπουλος	Άρνολντ	Ρόκυ	Μπαλμ
13	Papadopoulos	Asterix	Obelix	Panoramix	Persefoni	Παπαδόπουλος	Αστερίξ	Οβελίξ	Πανορα
14	Papadopoulos	Miky	Donald	Goofy	Miny	Παπαδόπουλος	Μίκυ	Ντόναλντ	Γκούφη
15									
16									
17									
18									
19									
20									
21									
22									
23									

Ready

Εικόνα 11 «Αρχείο με δεδομένα (data driven test)».

8.13. Συγχρονισμός δεδομένων

Ο συγχρονισμός χρησιμοποιήθηκε για να υπάρξει ομοιομορφία μεταξύ της εφαρμογής και των script ελέγχου. Επιτρέπει στον χρήστη να λύσει τα προβλήματα συγχρονισμού μεταξύ του ελέγχου και της εφαρμογής που γίνεται ό έλεγχος. Παραδείγματος χάριν, εάν δημιουργείται ένας έλεγχος που ανοίγει μια εφαρμογή βάσεων δεδομένων, μπορεί να προστεθεί ένα σημείο συγχρονισμού που αναγκάζει τον έλεγχο να περιμένει έως ότου φορτωθούν τα αρχεία της βάσης δεδομένων στην οθόνη.

8.14. Batch Test

Ένα Batch Test είναι ένα script ελέγχου που περιέχει όλες τις καταστάσεις κλήσεων άλλων ελέγχων. Ανοίγει και εκτελεί κάθε έλεγχο και σώζει τα αποτελέσματα ελέγχου. Εμποδίζει να εμφανιστεί μήνυμα λάθους κατά τη διάρκεια εκτέλεσης του script ελέγχου. Ένας έλεγχος μετατρέπεται σε batch test όταν επιλέγεις την εκτέλεση του ελέγχου σε κατάσταση batch mode.

Π.Χ

GUI_load(“a1.gui”);

call “a1”()

Κουτιά διαλόγου

Μπορεί ο χρήστης να δημιουργήσει κουτιά διαλόγου που να εμφανίζονται κατά τη διάρκεια της εκτέλεσης ενός διαδραστικού ελέγχου. Θα προτείνει στο χρήστη να εκτελέσει κάποια λειτουργία όπως το να γράψει ένα κείμενο ή να επιλέξει ένα αντικείμενο από τη λίστα.

Τύποι διαλογικών κουτιών:

Κουτιά Διαλόγου εισαγωγής
Κουτιά Διαλόγου λίστας
Κουτιά Διαλόγου κωδικού

Στην περίπτωση που τρέχει ο έλεγχος και δεν βρεθεί κάποιο αντικείμενο εμφανίζεται το ακόλουθο κουτί διαλόγου, ζητώντας από τον χρήστη να υποδείξει που βρίσκεται το αντικείμενο ώστε να συνεχιστεί ο έλεγχος.

Συναρτήσεις:

- Εσωτερικές συναρτήσεις
- Εισαγωγή συνάρτησης- Αντικείμενο/παράθυρο

Η γεννήτρια συναρτήσεων- είναι ένα οπτικό εργαλείο που παρουσιάζει ένα γρήγορο και χωρίς λάθη τρόπο για να προγραμματίσει ο χρήστης τους ελέγχους του.
Μπορεί να προσθέσει δηλώσεις της TSL στον έλεγχο χρησιμοποιώντας την γεννήτρια συναρτήσεων με δυο τρόπους:

- ✓ Δείχνοντας ένα αντικείμενο GUI
- ✓ Ή επιλέγοντας μια συνάρτηση από τη λίστα.

Οι συναρτήσεις που ορίζονται από τον χρήστη είναι οι :

Compiled modules - Μια διαμορφωμένη εφαρμογή είναι ένα script που περιέχει μια βιβλιοθήκη της συνάρτησης που είναι καθορισμένη από το χρήστη. Όταν φορτώνονται οι διαμορφωμένες εφαρμογές σε ένα script, η λειτουργία του συντάσσεται αυτόματα και παραμένει στη μνήμη.

Οι διαμορφωμένες εφαρμογές μπορούν να βελτιώσουν την απόδοση των ελέγχων. Δεδομένου ότι γίνεται διόρθωση λαθών στην διαμορφωμένη εφαρμογή πριν την χρήση της, ο έλεγχος θα απαιτήσει λιγότερο έλεγχο λαθών.

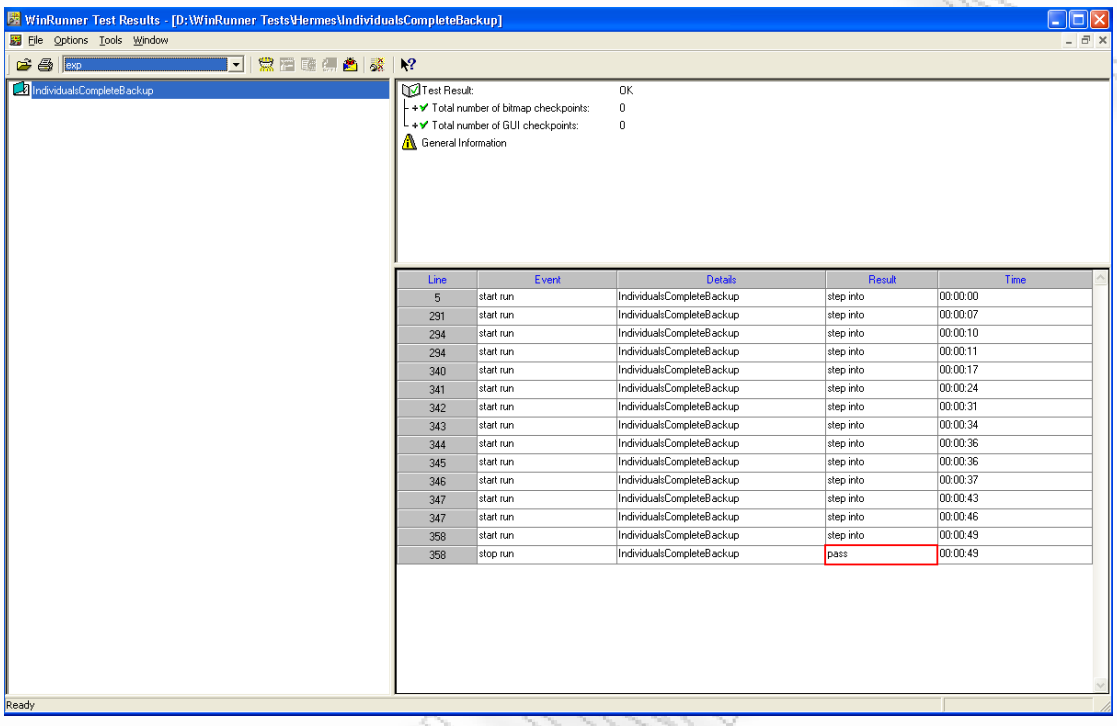
Break points- Σημεία Πάυσης

Θέτοντας σημεία παύσης μπορεί ήμασταν σε θέση να σταματήσουμε την εκτέλεση ελέγχου σε ένα συγκεκριμένο σημείο μέσα στο script ελέγχου.

Πχ:

```
# bMASTER Enterprise Banking System
    set_window ("bMASTER Enterprise Banking System", 5);
    menu_select_item ("View;Choices");
```

8.15. Τα αποτελέσματα ελέγχου του συστήματος EFT:



Όπως φαίνεται το σύστημα δεν ανίχνευσε καμία διαφορά μεταξύ προδιαγραφών και υλοποίησης ολοκληρώνοντας έτσι τον έλεγχο επιτυχώς.

Στην παραπάνω λοιπόν, περίπτωση ελέγχου της εφαρμογής EFT μαζί με τη βοήθεια του εργαλείου WinRunner καταφέραμε να εκτελέσουμε τους ελέγχους που αναφέρθηκαν και να δούμε πως το σύστημα ανταποκρίνεται κάτω από διαφορετικές συνθήκες. Η επιτυχία εξέτασης του συστήματος ανάγεται σε 2 επίπεδα:

• επίπεδο οργανισμού

Στο επίπεδο αυτό το σύστημα κατάφερε να ικανοποιήσει τις αρχικές απαιτήσεις σχεδιασμού καθώς οι χρόνοι απόκρισης του παραμείνανε στα πλαίσια που είχαν αρχικά οριστεί. Δεν υπήρξε καθυστέρηση στην διαχείριση των στοιχείων αλλά ούτε και στην εκτέλεση των κινήσεων.

• επίπεδο διεργασιών ή λειτουργιών

Έλεγχος Λογισμικού

Στο επίπεδο των διεργασιών το σύστημα ολοκλήρωσε επιτυχώς όλες τις απαιτούμενες διεργασίες μέσα στον απαιτούμενο χρόνο και με την προσθήκη πολλών δεδομένων ταυτόχρονα, αύξηση δηλαδή του όγκου πληροφορίας.

Στην εξεταζόμενη περίπτωση λοιπόν ελέγχθηκε το λειτουργικό σύστημα ηλεκτρονικών μεταφορών, όπου παρατηρήθηκε, όπως προαναφέρθηκε, πως το σύστημα ανταποκρίθηκε επιτυχώς στον έλεγχο πίεσης ενώ, οι κινήσεις μεταφοράς χρημάτων εκτελέστηκαν ομαλώς. Ταυτόχρονα παρατηρήθηκε πως οι αποδόσεις του συστήματος σε αυτήν την περίπτωση ήταν ικανοποιητικές.

Κρίνεται εξίσου απαραίτητο επίσης, τα σενάρια ελέγχου να είναι ενδελεχή, ούτως ώστε το άτομο που θα αξιολογήσει τα αποτελέσματα του ελέγχου να μπορεί εύκολα να επαληθεύσει ότι η έξοδος, το αποτέλεσμα δηλαδή του λογισμικού συστήματος είναι το ίδιο με αυτό που δόθηκε στο σενάριο ελέγχου ενώ στο τέλος παρουσιάζονται τα αποτελέσματα της εκτέλεσης στον χρήστη.

Επιπλέον, τυχόν λάθη που παρουσιάζονται διορθώνονται και οι έλεγχοι συνεχίζονται μέχρι το πρόγραμμα να απαλλαγεί από αυτά. Ελέγχονται τα αποτελέσματα προκειμένου να έχουμε την οριστική επικύρωση του συστήματος ενώ αν παρουσιάζεται απόκλιση από τις προδιαγραφές, θα πρέπει να επιστρέψουμε στην φάση της ανάλυσης και σχεδίασης προκειμένου να γίνουν οι κατάλληλες διορθώσεις και στο τέλος της φάσης των ελέγχων αυτών, έχουμε ένα έτοιμο πρόγραμμα που μπορούμε να παραδώσουμε στον πελάτη (φάση παράδοσης συστήματος) για χρήση.

8.15.1. Αποτελέσματα ελέγχου απόδοσης της εφαρμογής EFT

Ο έλεγχος απόδοσης έχει ως σκοπό να μετρήσει το πόσο γρήγορα το πρόγραμμα ολοκληρώνει έναν δεδομένο στόχο. Ο αρχικός στόχος είναι να καθοριστεί εάν η ταχύτητα επεξεργασίας είναι αποδεκτή σε όλα τα μέρη του προγράμματος. Εάν οι ρητές απαιτήσεις διευκρινίζουν την εκτέλεση προγράμματος, κατόπιν οι έλεγχοι απόδοσης είναι συχνά διενεργηθείς ως έλεγχοι αποδοχής. Στην δικής μας περίπτωση έγινε έλεγχος απόδοσης του συστήματος κάτω από συγκεκριμένες συνθήκες. Επίσης λόγω του ότι οι έλεγχοι απόδοσης είναι εύκολο να αυτοματοποιηθούν ο έλεγχος έγινε με βάση το εργαλείο WinRunner όπου έγινε εισαγωγή πλήθους δεδομένων στο σύστημα ώστε να μετρηθούν οι αποδόσεις του συστήματος. Αυτό είναι σημαντικό στην περίπτωση που θέλει ο χρήστης να κάνει μια σύγκριση απόδοσης των διαφορετικών καταστάσεων συστημάτων καθώς χρησιμοποιεί την διεπαφή χρήστη. Οι έλεγχοι απόδοσης είναι επίσης κατάλληλες για τον έλεγχο παλινδρόμησης. Η σύλληψη και η αυτόματη επανάληψη των ενεργειών των χρηστών κατά τη διάρκεια του ελέγχου παράγουν ένα σταθερό χρόνο απόκρισης του συστήματος .

Στην δική μας περίπτωση χρησιμοποιήθηκε η παραπάνω μορφή ελέγχου ώστε να ελεγχθεί η απόδοση του συστήματος χωρίς χρονοκαθυστερήσεις εισάγοντας δεδομένα αυτόματα στο σύστημα. Παρακολουθώντας αναλυτικά τα αποτελέσματα του ελέγχου είναι σε θέση ο χρήστης να συνάγει έγκυρα συμπεράσματα έχοντας παράλληλα εκτελέσει πολλαπλά σενάρια σε πολύ λιγότερο χρόνο σε σύγκριση με τον χειροκίνητο έλεγχο.

8.15.2. Αποτελέσματα ελέγχου πίεσης της εφαρμογής EFT

Ο έλεγχος πίεσης είναι μια μορφή ελέγχου απόδοσης που μετρά την ταχύτητα επεξεργασίας του προγράμματος καθώς αυξάνεται το φορτίο των δεδομένων του συστήματος. Το φορτίο του συστήματος χαρακτηριστικά σημαίνει τον αριθμό ταυτόχρονων χρηστών που μπορεί να εξυπηρετήσει το σύστημα (πελάτες). Μπορεί επίσης ο όρος αυτός να χρησιμοποιηθεί και για τον όγκο των δεδομένων που υποβάλλονται σε επεξεργασία, ο αριθμός των δεδομένων που καταγράφονται σε μια βάση δεδομένων, η συχνότητα των εισερχόμενων μηνυμάτων ή παρόμοιων παραγόντων. Στην δική μας περίπτωση πραγματοποιήθηκε ο έλεγχος εισάγοντας ένα αρχείο excel με μια λίστα δεδομένων από πλήθος πελατών οι οποίοι εισήχθησαν στο σύστημα ενώ παράλληλα ο χρήστης παρατηρούσε το χρόνο απόκρισης του συστήματος και αν το σύστημα ανταποκρινόταν σωστά ή αντιμετώπιζε κάποια δυσλειτουργία.

Στην περίπτωση αυτή ο χρήστης σημείωνε την συνθήκη κάτω από την οποία το σύστημα δεν λειτουργούσε σωστά και επαναλάμβανε τον έλεγχο εκ νέου ώστε να επιβεβαιωθεί ότι το σύστημα θέλει διόρθωση στο συγκεκριμένο σημείου του κώδικα.

8.16. Συμπεράσματα χρήσης του WinRunner

Με το συγκεκριμένο εργαλείο καταφέραμε να υλοποιήσουμε έλεγχο στο σύστημα ηλεκτρονικών μεταφορών EFT σε περιβάλλον Windows.

Η χρήση του βασίζεται σε λειτουργίες capture / playback. Παρέχει στους χρήστες του εργαλείου την δυνατότητα να προσθέσουν ή να αφαιρέσουν στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος μέσω κάποιου είδους χαρτογράφησης, ενώ μπορούν ακόμα να αλληλεπιδράσουν με τις φόρμες του συστήματος και να προσθέσουν δεδομένα ελέγχου.

Τα αποτελέσματα που προέκυψαν μπορεί ο ελεγκτής να τα αξιοποιήσει σαν δείγματα της μέτρησης της ποιότητας αντοχής και ανεκτικότητας του συγκεκριμένου συστήματος. Ενώ οι τιμές των αποτελεσμάτων μπορούν εύκολα να συγκεντρωθούν και να μελετηθούν σχετικά με το αν είναι τιμές εντός των αποδεκτών ορίων σε σχέση με τις συμφωνημένες απαιτήσεις.

Συγκεντρωτικά μερικά από τα πλεονεκτήματα του συγκεκριμένου προϊόντος είναι τα ακόλουθα:

Πλεονεκτήματα:

1. Απλότητα. Ο χειρισμός του συστήματος Winrunner είναι όσο το δυνατό πιο απλός. Ο χειριστής αν είναι εξοικειωμένος με συστήματα πληροφορικής δεν θα δυσκολευτεί να χειριστεί το σύστημα.

2. Ευρωστία. Το σύστημα μπορεί να συνεχίζει την λειτουργία του ακόμη και στην περίπτωση που ανίχνευσε λάθη στην υλοποίηση του υπό έλεγχο λογισμικού συστήματος και μπορεί να εκτελέσει το σύνολο των σεναρίων ελέγχου που καθορίστηκαν για την συγκεκριμένη εκτέλεση, ούτως ώστε να εξαχθούν τα αποτελέσματα.

3. Αυτοματοποίηση. Το σύστημα αυτοματοποιεί όσο το δυνατό τις περισσότερες λειτουργίες στον έλεγχο λογισμικών συστημάτων ούτως ώστε να μπορούν στην συνέχεια χωρίς κόπο οι χρήστες του να παίρνουν τα αποτελέσματα της εκτέλεσης και να κάνουν τις αναγκαίες αλλαγές στο υπό έλεγχο λογισμικό σύστημα. Οι αυτοματοποιημένες λειτουργίες περιλαμβάνουν την εξαγωγή των στοιχείων του γραφικού περιβάλλοντος, την παραγωγή σεναρίων ελέγχου και την εκτέλεση των σεναρίων αυτών.

4. Απόδοση. Το σύστημα, καθώς εκτελεί τα σενάρια ελέγχου στο υπό έλεγχο λογισμικό σύστημα, αποθηκεύει σε αρχείο όλες τις πληροφορίες και τα αποτελέσματα σχετικά με την κάθε λειτουργία που εκτελεί. Αξίζει να σημειωθεί ότι τα αποτελέσματα της εκτέλεσης συγκρίνονται αυτόματα με τα αποτελέσματα του σεναρίου ελέγχου και μπορούν να παρουσιάσουν στον χρήστη οπτικά αποτελέσματα. Τα αποτελέσματα μπορούν να αποθηκεύονται σε μορφή απλού αρχείου κειμένου (simple text file).

Μειονεκτήματα

- Ο κώδικας γίνεται σε μια ιδιόκτητη γλώσσα (TSL).Αυτού του είδους οι γλώσσες προγραμματισμού μπορεί να είναι πολύ περιοριστικές και να έχουν λίγες διαθέσιμες πηγές ενημέρωσης. Λίγες πηγές είναι διαθέσιμες στον TSL προγραμματισμό (βασίζεται στην γλώσσα προγραμματισμού C αλλά δεν είναι C).Για ελεγκτές οι οποίο δεν είναι εξοικειωμένοι με τεχνική γνώση και γενικά δεν έχουν τεχνικό υπόβαθρο δεν είναι εύκολο να παρακολουθούν τον κώδικα.
- Τέλος θα πρέπει να αναφερθεί πως δεν παρέχει καμία υποστήριξη σε Java και σε .Net ενώ δεν μπορεί να λειτουργήσει σε περιβάλλοντα ανεπτυγμένα με αυτές τις γλώσσες προγραμματισμού.

9. Επισκόπηση

Συμπεραίνουμε λοιπόν ότι ο έλεγχος λογισμικού είναι μία αρκετά σημαντική λειτουργία, για την δημιουργία ορθών και αξιόπιστων λογισμικών, δεδομένου δε ότι τα λογισμικά είναι πλέον αρκετά πολύπλοκα και άρα πρέπει να δημιουργηθούν κατάλληλες μέθοδοι ελέγχου έτσι ώστε να εκτελούν επαρκή έλεγχο. Το να καλυφθούν βέβαια σε μια εργασία όλες οι τρέχουσες και οι προβλεπόμενες ερευνητικές κατευθύνσεις στον έλεγχο λογισμικού είναι σαφώς αδύνατο, ενώ η συμβολή αυτού του εγγράφου πρέπει να θεωρηθεί ότι τείνει στο να απεικονίσει περιεκτικά το τι είναι ο έλεγχος λογισμικού και να καταδείξει το πόσο σημαντικός είναι για την ανάπτυξη ενός πληροφοριακού συστήματος.

Είναι προφανές ότι οι στόχοι που έχουν παρουσιαστεί στο roadmap και έχουν δηλωθεί ως όνειρα προορίζονται να παραμείνουν έτσι. Εντούτοις, σε μια έρευνα roadmap η πραγματικότητα δεν είναι η ταμπέλα στο τέρμα, αλλά τα μονοπάτια κατά μήκος των επισημασμένων διαδρομών. Έτσι, το τι είναι πραγματικά σημαντικό στο οποίο στρέφονται οι ερευνητές για να υπογραφεί η πρόοδος είναι οι αποκαλούμενες προκλήσεις, και βεβαίως το roadmap τις παρέχει σε αφθονία, κάποιες σε ένα πιο ώριμο στάδιο, ενώ άλλες μόλις αρχίζουν να εμφανίζονται. Αυτό που βεβαιώνεται είναι ότι ο έλεγχος λογισμικού είναι και θα συνεχίσει να είναι μια θεμελιώδης δραστηριότητα της τεχνολογίας λογισμικού, το λογισμικό θα πρέπει πάντα να δοκιμαστεί τελικά και να ελεγχθεί. Και όπως έχει εκτενώς συζητηθεί σε αυτό το έγγραφο, θα πρέπει να καταστήσουμε στα σίγουρα πιο αποτελεσματική τη διαδικασία, πιο προβλέψιμη και πιο εύκολη (που συμπίπτει και με τα τέσσερα όνειρα ελέγχου που παρουσιάστηκαν).

9.1. Μελλοντική εργασία

Το θέμα της Διπλωματικής αυτής Εργασίας επιλέχθηκε γιατί πιστεύω πως είναι επίκαιρο και σημαντικό για τον κλάδο της Πληροφορικής και προορίζεται να συνεχιστεί ώστε να εξαλειφθούν όλοι οι περιορισμοί και οι ελλείψεις του. Λόγω λοιπόν του ότι η έρευνα δεν σταματά ποτέ υπάρχουν πολλά πράγματα ακόμη που μπορούν να ελεγχτούν όπως η υποστήριξη περισσοτέρων γλωσσών προγραμματισμού. Μετά την ενασχόλησή μου με την παρούσα εργασία παρατήρησα μια άλλη επέκταση του συστήματος αυτοματοποίησης ελέγχου που θα πρέπει να υλοποιηθεί, είναι η αυτοματοποίηση της διαδικασίας καταγραφής των προδιαγραφών σε κάποια γλώσσα. Στην περίπτωση που η καταγραφή των προδιαγραφών γίνει σε άλλη γλώσσα μοντελοποίησης από του συστήματος, το σύστημα θα πρέπει να μπορεί να μεταφράσει τις προδιαγραφές στην γλώσσα μοντελοποίησης που αναγνωρίζεται από το σύστημα. Ακόμη πιστεύω πως η πρόσθεση δυνατότητας ελέγχου και σε άλλες γλώσσες προγραμματισμού θα βοηθήσει πολύ τα σύγχρονα συστήματα, δηλαδή θα ήταν καλύτερο να βελτιωθούν τα εργαλεία αυτόματου ελέγχου και να μπορούν να δέχονται οποιαδήποτε γλώσσα ανάπτυξης του συστήματος. Υπάρχουν πολλοί βρόγχοι επανάληψης και άλλες δομές σε άλλες γλώσσες στις οποίες μπορούν να εφαρμοστούν διάφορες τεχνικές ώστε να γίνουν πιο αποδοτικές και να επιστρέφουν αποτελέσματα σε μικρότερο χρόνο. Επίσης θα μπορούσαν να δοκιμαστούν νέες fitness functions για να βρεθεί η αποδοτικότερη, δηλαδή αυτή που καλύπτει τα περισσότερα μονοπάτια. Σημαντικό επίσης είναι να προστεθούν γενετικοί αλγόριθμοι ή νευρωνικά δίκτυα στην υλοποίηση παραγωγής σεναρίων ελέγχου. Αυτό θα επιτρέψει την σημαντική μείωση του αριθμού των σεναρίων που παράγονται από το σύστημα έχοντας πάντα υπόψη ότι η χρήση τέτοιων αλγόριθμων και ιδίως των γενετικών, δίνουν τις περισσότερες φορές πολύ καλά αποτελέσματα. Η γνώμη μου λοιπόν είναι πως ο ρόλος της γλώσσας είναι πολύ σημαντικός στην σημερινή εποχή και βλέπουμε συνεχώς την δημιουργία εφαρμογών σε νέες γλώσσες και τα συστήματα αναπτύσσονται με νέες απαιτήσεις.

Θεωρώ ότι τέτοιες ερευνητικές προσπάθειες παρουσιάζουν ελπιδοφόρα αποτελέσματα στην επίδειξη των βελτιώσεων αποδοτικότητας και αποτελεσματικότητας του ελέγχου λογισμικού. Άλλωστε ο έλεγχος του λογισμικού, αποτελεί ούτως ή άλλως μια εξαιρετικά σημαντική διαδικασία, για την οποία δαπανάται το μεγαλύτερο ποσοστό των διαθέσιμων για την ανάπτυξη του λογισμικού πόρων και από την οποία εξαρτάται σε μεγάλο βαθμό η μετέπειτα συμπεριφορά του συστήματος. Κατά συνέπεια, πρέπει σε έναν οργανισμό

ανεξαρτήτου μεγέθους να δαπανώνται χρήματα και να θεωρείται απαραίτητος ο έλεγχος λογισμικού διατηρώντας παράλληλα το κόστος ελέγχου σε ένα σωστό επίπεδο και αξιοποιώντας οποιοδήποτε ανθρώπινο παράγοντα ή μηχάνημα αποτελεσματικά. Οι έλεγχοι λογισμικού δημιουργούν ευκαιρίες που οδηγούν σε καρποφόρες σχέσεις ανάμεσα σε διάφορους ερευνητικούς τομείς.

Μια απαραίτητη τελική παρατήρηση αφορά τις πολλές καρποφόρες σχέσεις μεταξύ των ελέγχων λογισμικού και άλλων ερευνητικών τομέων. Με την εστίαση στα συγκεκριμένα προβλήματα του ελέγχου λογισμικού, έχουμε αγνοήσει στην πραγματικότητα πολλές ενδιαφέρουσες ευκαιρίες που προκύπτουν συνοριακά μεταξύ του ελέγχου και άλλων αρχών. Μερικά αναφέρθηκαν σε αυτό το έγγραφο, όπως για παράδειγμα οι τεχνικές ελέγχου με βάση το πρότυπο (π.χ., η κατεύθυνση προς τον έλεγχο που βασίζεται στο πρότυπο), ή η χρήση προσεγγίσεων που βασίζονται στην έρευνα , για την παραγωγή εισόδου ελέγχου, ή την εφαρμογή των τεχνικών ελέγχου για να αξιολογηθούν οι ιδιότητες απόδοσης. Πιστεύω ότι πραγματικά οι δημιουργικές ιδέες που μπορούν να προκύψουν από μια πιο ολιστική προσέγγιση στην έρευνα ελέγχου λογισμικού είναι αρκετές. Στην παρούσα εργασία μπορεί ο αναγνώστης να βρει και να εκτιμήσει νέες ενδιαφέρουσες συμπράξεις που εκτείνονται στις ερευνητικές αρχές της τεχνολογίας λογισμικού. Αυτές είναι μερικές μόνο από τις σκέψεις που μπορούν να υλοποιηθούν ως μελλοντικές εργασίες.

10. Βιβλιογραφία

[1] Mohd. Ehmer Khan, Different Forms of Software Testing Techniques for Finding Errors, Department of Information Technology Al Musanna College of Technology, Sultanate of Oman.

[2] M. J. Harrold. Testing: a roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, pages 61–72. IEEE Computer Society, 2000. In conjunction with ICSE2000.

[3] Software Testing Research: Achievements, Challenges, Dreams, Antonia Bertolino,IEEE 2007.

[4] M. J. Harrold. Testing: a roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, pages 61–72. IEEE Computer Society, 2000. In conjunction with ICSE2000.

[5] W. Hetzel. The Complete Guide to Software Testing, 2nd Edition. QED Inf. Sc., Inc., 1988.

[6] B. Beizer. Software Testing Techniques (2nd ed.). Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[7] Galileo Computing Software Testing and Internationalization © 2003 Lemoine International and the Localization Industry Standards Association (LISA)

[8] Antonia Bertolino,Software Testing, © IEEE – Trial (Version 0.95) – May 2001

[9] W. Howden. Reliability of the path analysis testing strategy. *IEEE Trans. Softw. Eng.*, SE-2(3):208–215, 1976.

[10] M.-C. Gaudel. Formal methods and testing: Hypotheses,and correctness approximations. In *Proc. FM 2005, LNCS 3582*, pages 2–8. Springer-Verlag, 2005.

[11] D. Hamlet and R. Taylor. Partition testing does not inspire confidence. IEEE Trans. Softw. Eng., 16(12):1402–1411, 1990.

[12] H. Zhu and X. He. A theory of behaviour observation in software testing. Technical Report CMS-TR-99-05, 24, 1999

[13] C. Blundell, D. Giannakopoulou, and C. S. Pasareanu. Assume-guarantee testing. In *Proc. SAVCBS '05*, pages 7– 14. ACM Press, 2005.

[14] M. van der Bijl, A. Rensink, and J. Tretmans. Compositional testing with ioco. In *Proc. FATES 2003, LNCS 2931*, 2003.

[15] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17:103– 120, 1996.

[16] R. Gotzhein and F. Khendek. Compositional testing of communication systems. In *Proc. IFIP TestCom 2006, LNCS 3964*, pages 227–244. Springer Verlag, May 2006.

[17] D. Sjøberg, T. Dyb°a, and M. Jørgensen. The future of empirical methods in software engineering research.

[18] H. Do, S. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Softw. Eng.*, 10(4):405– 435, 2005.

[19] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[20] M. J. Rehman, F. Jabeen, A. Bertolino, and A. Polini. Testing software components for integration: a survey of issues and techniques. *Software Testing, Verification and Reliability*.

[21] M. Lyu (ed.). *Handbook of Software Reliability Engineering*. McGraw-Hill, New York, and IEEE CS Press, Los Alamitos, 1996.

[22] J. Poore, H. Mills, and D. Mutchler. Planning and certifying software system reliability. *IEEE Software*, pages 87–99, Jan. 1993.

[23] M. Lyu. Software reliability engineering: A roadmap. In *[19]*.

[24] Elfriede Dustin, Effective Software Testing ,Copyright © 2003 by Pearson Education, Inc.

[25] Y. Le Traon, B. Baudry, and J.-M. J´ez´equel. Design by contract to improve software vigilance. *IEEE Trans. Softw. Eng.*, 32(8):571–586, 2006.

[26] L. C. Briand, Y. Labiche, and M. M. S´owka. Automated, contract-based user testing of commercial-off-theshelf components. In *Proc. 28th Int. Conf. on Sw. Eng.*, pages 92–101. ACM Press, 2006.

[27] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, pages 129–153, 1956.

[28] W. Grieskamp. Multi-paradigmatic model-based testing. In *Proc. FATES/RV*, pages 1–19. LNCS 4262, August 15-16, 2006.

[29] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of* Reactive Systems - Advanced Lectures, LNCS 3472. Springer Verlag, 2005.

[31] N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Trans. Softw. Eng.*, 30(12):859–872, 2004.

[32] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Gruenbacher,editors. Value-Based Software Engineering. Springer-Verlag, Heidelberg, Germany, 2006.

[33] S. Biffl, R. Ramler, and P. Gruenbacher. Value-based management of software testing.

[34] G. Canfora and M. Di Penta. Testing services and servicecentric systems: Challenges and opportunities. IT Professional, 8(2):10–17, March/April 2006.

[35] E. Weyuker and F. Vokolos. Experience with performance testing of software systems: Issues, an approach, and case study. IEEE Trans. Soft. Eng., 26(12):1147–1156, 2000.

[36] K. G. Larsen, M. Mikucionis, B. Nielsen, and A. Skou. Testing real-time embedded software using UPPAAL-TRON: an industrial case study. In Proc. 5th ACM Int. Conf. on Embedded Softw., pages 299. ACM Press, 2005.

[37]J. Wegener and M. Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. Real-Time Syst., 15(3):275–298, 1998.

[38] Source: Newport Group, Inc. ©2001, IT Trends Research and Reporting

[39] Rex Black ,Investing in Software Testing: The Importance of the Right Technique, Copyright © 2002 Rex Black, All Rights Reserved

[40] What is gray box testing, 2009, http://www.robdavispe.com/firee2/software-qa-testing-test-tester-2210.html

[41] Gilbert Thomas Laycock, “The theory and practice of specification based testing”,1993

[42]Nilesh Parekh, “Software Testing – White Box Testing Strategy”, 2005

[43] Hans Schaefer, What a Tester Should Know, even After Midnight http://home.c2i.net/schaefer/testing , html.hans.schaefer@ieee.org

[44] James A. Whittaker, Exploratory Software Testing, Copyright © 2010 Pearson Education, Inc.

[45] Chays, David, Test Data Generation for Relational Database Applications. Ph.D. thesis, Polytechnic University, January 2004.

[46] Ray Robinson , Automation Test Tools, Copyright by Ray Robinson 2001

[47] David Burns, Selenium 1.0 Testing Tools Beginner's Guide, Birmingham – Mumbai, Copyright © 2010 Packt Publishing

[48] Thomas Kuhn, The Structure of Scientific Revolution

[49] Whittaker, James, What is Software Testing? And Why is it So Hard?, IEEE Software, January/February 2000.

[50] Elfriede Dustin et al., Automated Software Testing (Reading, Mass.: Addison-Wesley, 1999).

[51] Adapted from Elfriede Dustin et al., Automated Software Testing (Reading, Mass.: Addison-Wesley, 1999, Chapter 2.)

[52] Rana Farid Mikhail, Donald Berndt & Abraham Kandel, University of South Florida, USA Automated database Applications testing, Specification Representation for Automated Reasoning , Copyright © 2010 by World Scientific Publishing Co. Pte. Ltd.

[53] Mercury Software Products: Mercury WinRunner. http://www.mercury.com/us/products/quality-center/functional-testing/winrunner

[54]MercurySoftware Products: Mercury QuickTest. http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional

[55] Mercury Software Products: Mercury LoadRunner. http://www.mercury.com/us/products/performance-center/loadrunner

[56]RationalSoftware:RationalFunctionalTester. http://www306.ibm.com/software/awdtools/tester/functional/index.html

[57] Jeff McWherter,Ben Hall,Testing ASP.NET Web Applications, Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana

[58] Segue, SilkCentral Test Manager, www.segue.com

[59]Segue,SilkTest. http://www.segue.com/products/functional-regressional-testing/silktest.asp

[60] Mercury Software Products: www.merc-int.com .

[61] http://www.vietnamesetestingboard.org/zbxe/?document_srl=538476–Testing Banking Applications

[62] Bmaster Core Banking System http://unisystems.gr/el/solutions-services-inside/bank-solutions-xrimatooikonomikou-sector/solutions-core-banking-branch-automation/bmaster.html

[63] Current Software Testing Trends in Banking and Financial Services Industry - January 31st, 2011 www.testingtrendz.com

[64] Uni.systems s.a. 2008, *bMaster Enterprise Banking System - Solution Overview*

[65] Software testing process for shared banking services (sbs) system, Norakmar Binti Arbain Sulaiman)

[66] Uni.systems s.a. 2008, *bMaster Enterprise Banking System - EFT Overview*

[67] WinRunner User's Guide, Version 7.6 © 2003 Mercury Interactive Corporation, All rights reserved.