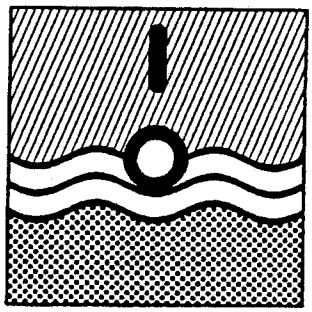
ΥΔΡΟΣΚΟΠΙΟ

ΠΡΟΓΡΑΜΜΑ STRIDE ΕΛΛΑΣ

ΔΗΜΙΟΥΡΓΙΑ ΕΘΝΙΚΗΣ ΤΡΑΠΕΖΑΣ
ΥΔΡΟΛΟΓΙΚΗΣ ΚΑΙ ΜΕΤΕΩΡΟΛΟΓΙΚΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΟΜΕΑΣ ΥΔΑΤΙΚΩΝ ΠΟΡΩΝ,
ΥΔΡΑΥΛΙΚΩΝ ΚΑΙ ΘΑΛΑΣΣΙΩΝ ΕΡΓΩΝ

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
DEPARTMENT OF WATER RESOURCES,
HYDRAULIC AND MARITIME ENGINEERING



ΠΡΟΔΙΑΓΡΑΦΕΣ ΕΛΕΓΧΟΥ ΛΟΓΙΣΜΙΚΟΥ

SOFTWARE TESTING SPECIFICATIONS

Γιάννης Α. Παραβάντης

John A. Paravantis

HYDROSCOPE

STRIDE HELLAS PROGRAMME

DEVELOPMENT OF A NATIONAL DATA
BANK FOR HYDROLOGICAL AND
METEOROLOGICAL INFORMATION

Αριθμός τεύχους **1/12**
Report number

ΑΘΗΝΑ - ΜΑΙΟΣ 1993
ATHENS - MAY 1993

ΠΕΡΙΛΗΨΗ

Στόχος αυτής της εργασίας είναι να βοηθήσει στην ανάπτυξη προδιαγραφών και σχεδίαση ελέγχων γιά το λογισμικό σύστημα του ΥΔΡΟΣΚΟΠΙΟΥ. Εξετάζεται αρχικά η διαδικασία ανάπτυξης λογισμικού, που εμπεριέχει τις φάσεις προγραμματισμού, αποσφαλμάτωσης και ελέγχου. Η προσέγγιση ενός χρήστη του συστήματος σχετίζεται με την ανάλυση των απαιτήσεων των χρηστών, την συγγραφή των προδιαγραφών του συστήματος και την σχεδίαση του ελέγχου παραλαβής. Η προσέγγιση του σχεδιαστή περιλαμβάνει τα καθήκοντα της σχεδίασης του συστήματος και των ελέγχων ενοποίησης και συστήματος. Τέλος, η προσέγγιση του προγραμματιστή περιλαμβάνει την συγγραφή προδιαγραφών, σχεδίαση και προγραμματισμό των υποενοτήτων του λογισμικού καθώς και τον έλεγχο των ενοτήτων. Οι έλεγχοι λογισμικού στοχεύουν στην εξασφάλιση πληρότητας, συνέπειας, συνεκτικότητας, ευκολίας χρήσεως, ταχύτητας εκτέλεσης, ασφάλειας και αξιοπιστίας του συστήματος, περιλαμβάνουν δε ελέγχους από την κορυφή προς τα κάτω ή από την βάση προς τα πάνω, στατικούς ή δυναμικούς και ανοικτής ή κλειστής δομής, η υλοποίηση των οποίων ολοκληρώνεται με τα στάδια ελέγχου α και β. Βασικά εργαλεία ελέγχου αποτελούν οι δοκιμαστικές εκτελέσεις, οι εποπτικοί έλεγχοι κώδικα, η στατική ανάλυση και οι κατάλογοι ελέγχου, προσχέδια των οποίων, κατάλληλα γιά το λογισμικό του ΥΔΡΟΣΚΟΠΙΟΥ που αναπτύσσεται σε Ingres & Windows4GL, παρουσιάζονται στην παρούσα εργασία.

ABSTRACT

The aim of this study is to assist in the development of specifications and the design of testing for the HYDROSCOPE software system. Initially, the software development process is examined, including programming, debugging and testing phases. The approach of a user of the system is related to analyzing user requirements, authoring system specifications and designing acceptance testing. The system designer approach is related to designing the system as well as integration and system testing. Finally, the approach of the programmer includes specifications development, design and programming of software modules and unit testing. Software testing attempts to achieve completeness, consistency, cohesion, ease of use, speed of execution, security and reliability of the system, including top-down or bottom-up, static or dynamic, as well as white-box or black-box testing, which are completed with the alpha and beta testing phases. Fundamental testing tools include dry runs, code inspection, static analyses and checklists, drafts of which, appropriate for the HYDROSCOPE software developed in Ingres and Windows4GL, are presented in this work.

ΠΕΡΙΕΧΟΜΕΝΑ

1 ΕΙΣΑΓΩΓΗ

Στόχος αυτής της εργασίας είναι να δώσει τα κατάλληλα εργαλεία στις ομάδες ανάπτυξης λογισμικού του ΥΔΡΟΣΚΟΠΙΟΥ, ώστε (α) να σχεδιάσουν ορθούς και πλήρεις ελέγχους του αναπτυσσόμενου λογισμικού συστήματος, και (β) να προγραμματίσουν την αναφορά των αποτελεσμάτων των ελέγχων και τυχόν διορθωτικών επεμβάσεων με τυποποιημένο και σαφή τρόπο. Η παρούσα εργασία αποτελεί ένα έγγραφο εργασίας, για συνεχή αναφορά από τους προγραμματιστές και τα λοιπά μέλη της ομάδας ανάπτυξης λογισμικού κατά την ανάπτυξη των προδιαγραφών και σχεδιασμό των διαφορετικών ελέγχων.

Η εργασία αυτή αρχικά κάνει μία κριτική παρουσίαση και συζήτηση επιλεγμένων θεωρητικών εννοιών σχετικά με τον έλεγχο ενός λογισμικού συστήματος (Κεφάλαιο 2). Ακολουθεί μία λεπτομερής παρουσίαση των διαφορετικών τύπων ελέγχου, με πρακτικές συμβουλές και οδηγίες για την εφαρμογή αυτών των ελέγχων στο σύστημα του ΥΔΡΟΣΚΟΠΙΟΥ (Κεφάλαιο 3). Η εργασία περιλαμβάνει και έναν κατάλογο δόκιμων Αγγλο-Ελληνικών όρων πληροφορικής (Παράρτημα), οι οποίοι θα υποβοηθήσουν την σωστή και κατανοητή σύνταξη των μετέπειτα αναφορών πάνω στους ελέγχους των διαφορετικών ομάδων ανάπτυξης λογισμικού. Στον κατάλογο αυτό περιλαμβάνονται και διάφορα ακρώνυμα του περιβάλλοντος Ingres (από Date, 1987).

2 ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΕΝΝΟΙΩΝ ΑΝΑΠΤΥΞΗΣ ΚΑΙ ΕΛΕΓΧΟΥ ΛΟΓΙΣΜΙΚΟΥ

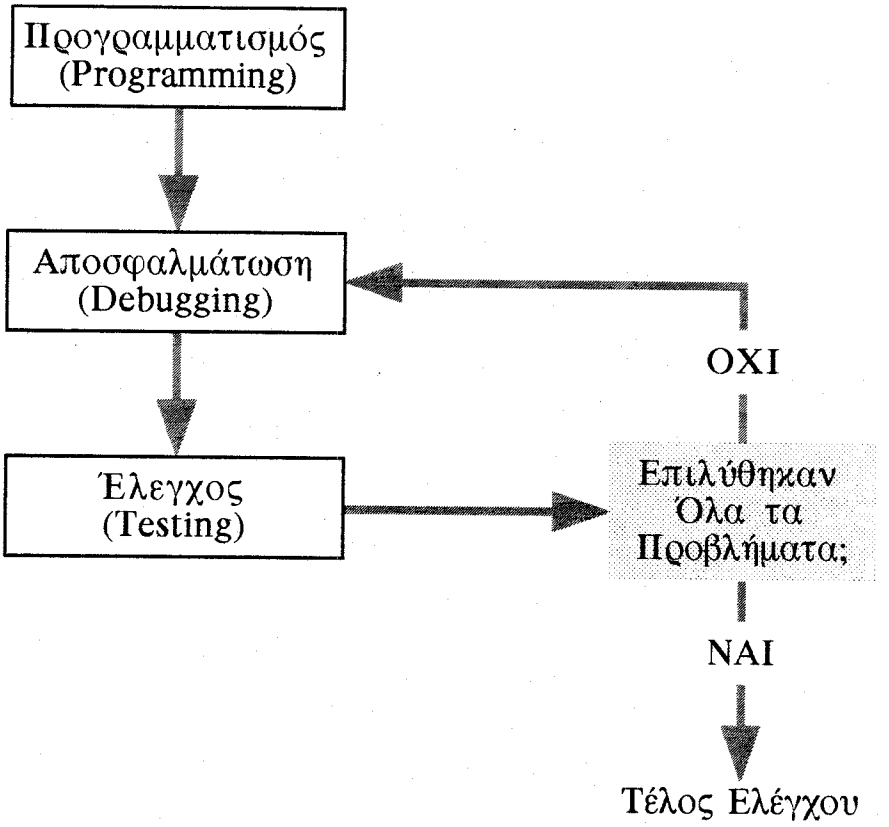
Το κεφάλαιο αυτό παρουσιάζει διάφορες θεωρητικές έννοιες σχετικές με τον σχεδιασμό, προγραμματισμό και αφαίρεση σφαλμάτων ενός λογισμικού συστήματος.

Οι ακόλουθες ενότητες δίνουν μία συνοπτική εικόνα της θεωρίας ελέγχου λογισμικού και παρουσιάζουν επιλεγμένες λεπτομέρειες που ενδιαφέρουν στα πλαίσια του λογισμικού του συγκεκριμένου έργου (ΥΔΡΟΣΚΟΠΙΟ). Αρχικά γίνεται μία γενική παρουσίαση των σταδίων και εργασιών που λαμβάνουν χώρα στην ανάπτυξη και έλεγχο ενός λογισμικού συστήματος. Ακολούθως, περιγράφονται οι ρόλοι και καθήκοντα των μελών μίας τυπικής ομάδας ανάπτυξης λογισμικού, με παρουσίαση μερικών λεπτομερειών για το ΥΔΡΟΣΚΟΠΙΟ. Τέλος, εξετάζονται μερικές τέτοιες μέθοδοι και εργαλεία ελέγχων.

2.1 Γενικά περί Ανάπτυξης Λογισμικού

Η ανάπτυξη και ο έλεγχος ενός λογισμικού συστήματος είναι στενά συνυφασμένα στο γενικό πλάνο της δημιουργίας μίας εφαρμογής. Το *Σχήμα 1* απεικονίζει την πορεία από τον προγραμματισμό μίας εφαρμογής (λογισμικού συστήματος) προς την αποσφαλμάτωση (αφαίρεση σφαλμάτων - debugging) και τον έλεγχο (testing) τού λογισμικού. Θα πρέπει να τονισθεί η διαφορά μεταξύ αφαίρεσης λαθών (debugging) και ελέγχου (testing) λογισμικού. Ο έλεγχος λογισμικού είναι μία διαδικασία που στοχεύει στην τεκμηρίωση της ύπαρξης λαθών, ενώ η αποσφαλμάτωση ασχολείται με την εύρεση αυτών των λαθών και την ακόλουθη αφαίρεση τους ώστε ο τελικός κώδικας να λειτουργεί ορθά.

Με το πέρας του προγραμματισμού της εφαρμογής, αρχίζει η ανεύρεση και διόρθωση λαθών και ελαττωμάτων με διάφορες μεθόδους ελέγχου, λεπτομέρειες των οποίων θα εξετασθούν σε επόμενες ενότητες. Ο έλεγχος αποκαλύπτει και άλλα λάθη τα οποία, με την σειρά τους, αφαιρούνται. Πολλές φορές, η αφαίρεση λαθών που γίνεται με αλλαγή υπάρχοντος κώδικα ίσως δε και με συγγραφή νέων τμημάτων κώδικα, γίνεται αφορμή εισαγωγής νέων λαθών. Ο κύκλος αυτός συνεχίζεται μέχρι να αφαιρεθούν κατά το δυνατόν όλα τα λάθη. Όταν δεν ανακαλύπτονται πλέον λάθη, ο έλεγχος του λογισμικού θεωρείται περατωθείς και το λογισμικό είναι ετοιμοπαράδοτο.



Σχήμα 1. Προγραμματισμός, αποσφαλμάτωση και έλεγχος λογισμικού.

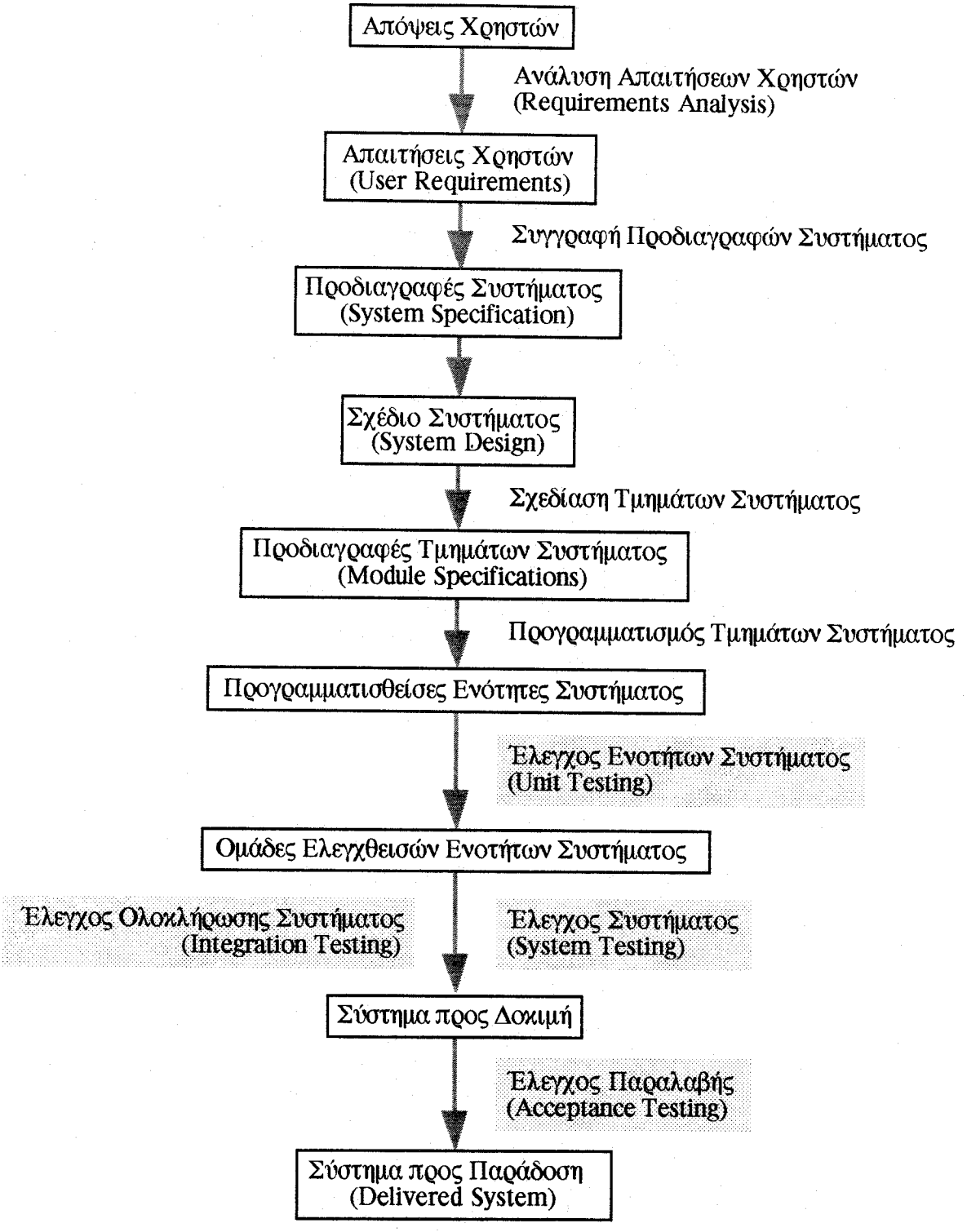
2.1.1 Στάδια Ανάπτυξης Λογισμικού. Οι διάφορες εργασίες και στάδια στην ανάπτυξη του λογισμικού ενός συστήματος (μίας πλήρους εφαρμογής) φαίνονται στο *Σχήμα 2* Το σημείο εκκίνησης για τον σχεδιασμό και την ανάπτυξη ενός λογισμικού συστήματος, είναι οι απόψεις των χρηστών. Η ανάλυση αυτών των απόψεων, που γίνεται με μεθόδους όπως συμπλήρωση ερωτηματολογίων, συνεντεύξεις αλλά και έρευνα γραφείου, καταλήγει στην συγγραφή του τεύχους των τυπικών απαιτήσεων των χρηστών (user requirements).

Στο σημείο αυτό, αρχίζει η ουσιαστική διαδικασία δημιουργίας του λογισμικού συστήματος, καθώς το τεύχος των τυπικών απαιτήσεων των χρηστών, χρησιμοποιείται για τον σχεδιασμό των προδιαγραφών του λογισμικού συστήματος (system specification), ώστε οι απαιτήσεις αυτές να πληρούνται. Ακολουθεί λεπτομερής σχεδιασμός του συστήματος (system design), με βάση αυτές τις προδιαγραφές.

Της ετοιμασίας ενός γενικού σχέδιου για το λογισμικό σύστημα, έπεται ο διαχωρισμός του συστήματος σε υποενότητες ή τμήματα (modules), των οποίων συντάσσονται οι προδιαγραφές (module specifications). Με την συμπλήρωση των προδιαγραφών αυτών, ολοκληρώνονται όλες εκείνες οι εργασίες που προηγούνται του προγραμματισμού του λογισμικού.

Ο προγραμματισμός του λογισμικού συνήθως αρχίζει από το μερικό προς το ολικό. Αρχικά προγραμματίζονται τα (ήδη σχεδιασθέντα) μερικά τμήματα (π.χ. υπορουτίνες και συναρτήσεις) του συστήματος, ακολουθεί δε ο κώδικας εκείνος που απαιτείται για την ενσωμάτωση όλων αυτών των τμημάτων, αρχικά σε ενότητες (units), και ακολούθως σε ενιαίο σύστημα. Τέλος, προγραμματίζεται η καλή επικοινωνία των τμημάτων και ενοτήτων μεταξύ τους.

Στο σημείο αυτό, αρχίζει ο έλεγχος του λογισμικού (software testing). Όπως και στην περίπτωση του προγραμματισμού, ο έλεγχος του λογισμικού αρχίζει από το μερικό προς το ολικό. Αρχικά, εκτελείται ο έλεγχος ενοτήτων του συστήματος (unit testing), ο οποίος περιλαμβάνει και τον έλεγχο των τμημάτων κώδικα (modules).



Σχήμα 2. Εργασίες και στάδια ανάπτυξης και ελέγχου λογισμικού.

Με το πέρας του ελέγχου των ενοτήτων, το λογισμικό σύστημα αποτελείται από ομάδες ελεγχθεισών ενοτήτων. Για την αφαίρεση σφαλμάτων που υπάρχουν στον κώδικα ο οποίος ολοκληρώνει το σύστημα και επιτρέπει στις ενότητες να επικοινωνούν μεταξύ τους, χρησιμοποιείται ο έλεγχος ολοκλήρωσης και ενοποίησης του συστήματος (integration testing) κατά τον οποίον ελέγχεται η σωστή ολοκλήρωση και ενοποίηση των διαφορετικών υποσυστημάτων του λογισμικού συστήματος. Με την συμπλήρωση του επιτυχούς ελέγχου όλων των υποσυστημάτων, ελέγχεται πλέον το ενιαίο λογισμικό σύστημα με τον έλεγχο συστήματος (system testing). Ο έλεγχος συστήματος μπορεί έτσι να θεωρηθεί σαν μία οριακή μορφή ελέγχου ολοκλήρωσης όπου αντί να εξετάζονται τα διάφορα υποσυστήματα χωριστά, εξετάζεται ολόκληρο το λογισμικό σύστημα σε ενιαία μορφή.

Με την ολοκλήρωση των ελέγχων ολοκλήρωσης και συστήματος, το λογισμικό σύστημα μπορεί πλέον να εκτελεσθεί ολοκληρωμένο, με σκοπό να δοκιμασθεί η επάρκειά του σε σχέση με τις αρχικές απαιτήσεις των πελατών. Ο σχετικός έλεγχος ονομάζεται έλεγχος παραλαβής (acceptance testing). Τελικό προϊόν του ελέγχου αυτού, είναι το ετοιμοπαράδοτο σύστημα.

Είναι ενδιαφέρουσα η παρατήρηση ότι η σχετική βιβλιογραφία εμφανίζει διαφορές ανάμεσα στους τρόπους καθορισμού των σταδίων ελέγχου. Η παρούσα εργασία επέλεξε τα στάδια που συζητούν οι Ould και Unwin (1989), ενώ ο Sommerville (1989) εκθέτει μικρές διαφορές στον καθορισμό των ελέγχων ολοκλήρωσης και συστήματος. Ακόμα, δημοσιεύσεις που επεξηγούν ελέγχους αντίστοιχων λογισμικών βάσεων δεδομένων, παρουσιάζουν αρκετές πρακτικές διαφοροποιήσεις και λεπτομέρειες πάνω στο γενικό θεωρητικό σχέδιο, με βάση τις ιδιαιτερότητες του συγκεκριμένου λογισμικού συστήματος (π.χ. Thornberg, 1991).

2.2 Γενικά περί Ελέγχου Λογισμικού

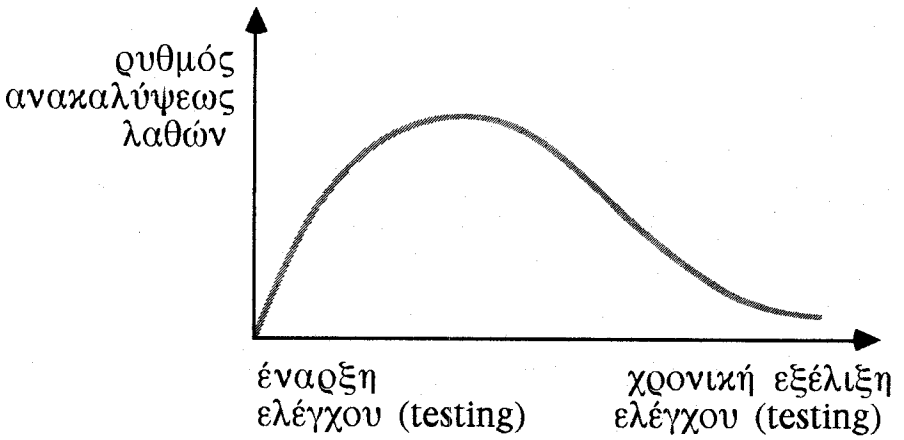
Ο έλεγχος λογισμικού σχετίζεται με την επαλήθευση (verification) και επικύρωση (validation) ενός λογισμικού συστήματος. Η διαδικασία επαλήθευσης έχει σαν στόχο να δείξει ότι ένα λογισμικό σύστημα έχει προγραμματισθεί σύμφωνα με τις προδιαγραφές του, ενώ η επικύρωση έχει σαν στόχο να δείξει ότι το σύστημα λειτουργεί σύμφωνα με τις απαιτήσεις και επιθυμίες των χρηστών του. Ο έλεγχος και η αφαίρεση σφαλμάτων λογισμικού, μειώνει το κόστος της επαλήθευσης και επικύρωσης ενός λογισμικού συστήματος.

Όπως αναφέρθηκε σε προηγούμενη ενότητα, ο έλεγχος λογισμικού αποκαλύπτει τυχόν λάθη που υπάρχουν σε ένα λογισμικό σύστημα - πρέπει να τονισθεί όμως ότι κανένας έλεγχος δεν μπορεί να αποδείξει την πλήρη απουσία λαθών. Επομένως, η αποτελεσματικότητα τού σωστού σχεδιασμού των ελέγχων είναι σημαντικός παράγοντας στην ανεύρεση όσο το δυνατόν περισσότερων σφαλμάτων. Κατά τον έλεγχο ενός μεγάλου συστήματος, είναι πιό σημαντικό να ελεγχθούν τα τμήματα εκείνα του συστήματος που χρησιμοποιούνται πιό συχνά (Sommerville, 1989).

Η σημασία των ελέγχων λογισμικού για την έγκαιρη αφαίρεση σφαλμάτων είναι μεγάλη. Η πιθανότητα αποτυχίας ενός λογισμικού συστήματος εξ αιτίας σφάλματος που θα αποκαλυφθεί μετά την παράδοση (το οποίο δηλαδή δεν ανευρέθηκε κατά τους ελέγχους) είναι αντιστρόφως ανάλογη του χρόνου που επενδύθηκε σε ελέγχους. Μάλιστα δε, όπως επισημαίνουν οι Ould και Unwin, όσο μακρύτερα από την γενεσιουργό αιτία του αποκαλυφθεί κάποιο σφάλμα λογισμικού, τόσο δυσκολότερη είναι η αφαίρεσή του (1989). Επίσης, όσο περισσότερο χρόνο αφήνονται σφάλματα μέσα στο σύστημα, τόσο περισσότερο κοστίζει η αφαίρεσή τους.

Οι Ould και Unwin αναφέρονται σε μελέτες που δείχνουν ότι οι έλεγχοι λογισμικού δεν γίνονται επαρκώς κατανοητοί, και μάλιστα εκτελούνται ανεπαρκώς (1989). Τονίζουν ότι οι διαδικασίες ελέγχου λογισμικού πρέπει να χαρακτηρίζονται από καλή οργάνωση, επαρκή προϋπολογισμό και μεθοδική προσέγγιση. Όσον δε αφορά την σημασία του σωστού σχεδιασμού και προδιαγραφών των ελέγχων λογισμικού, οι Ould και Unwin αναφέρουν ότι η ποιότητα των ελέγχων είναι απόλυτα συνδεδεμένη με τις προδιαγραφές στις οποίες είναι βασισμένοι (1989).

Το *Σχήμα 3* απεικονίζει την γενική καμπύλη ανακάλυψης λαθών κατά τον έλεγχο λογισμικού. Με την έναρξη του ελέγχου, ανακαλύπτονται όλο και περισσότερα λάθη, καθώς η αφαίρεση μερικών λαθών γίνεται αιτία εισαγωγής νέων λαθών στον κώδικα. Με την πρόοδο του ελέγχου, η εισαγωγή νέων λαθών ελαττώνεται, ώσπου ο ρυθμός ανακαλύψεως λαθών τείνει προς το μηδέν ("τείνει" ασυμπτωτικά προς το μηδέν παρά "παίρνει" την τιμή του μηδενός, διότι, γενικά, υπάρχουν πάντα κάποια λάθη που παραμένουν στον κώδικα).



Σχήμα 3. Ανακάλυψη λαθών κατά τον έλεγχο λογισμικού.

Όσον αφορά τα λάθη και ελαττώματα που μπορεί να αποκαλύψει ο έλεγχος λογισμικού, αυτά είναι διαφόρων μορφών (Lamb, 1988):

- σφάλματα λογικής,
- σφάλματα υπερχείλισης (overload ή overflow),
- προβλήματα συγχρονισμού,
- προβλήματα λειτουργίας και αποδόσεως, και
- αποτυχίες του λειτουργικού συστήματος (operating system) ή του μηχανικού υλικού (hardware).

2.2.1 Καθήκοντα και Τύποι Ελέγχου. Μία τυπική ομάδα ανάπτυξης και ελέγχου λογισμικού περιλαμβάνει καθήκοντα που σχετίζονται με (1) την γενική διαχείριση του λογισμικού συστήματος, (2) την εξασφάλιση λειτουργιών που είναι απαραίτητες στους μελλοντικούς χρήστες της εφαρμογής, (3) τον σχεδιασμό της δομής του συστήματος, και τέλος (4) τον προγραμματισμό του συστήματος.

Σε σχέση με τα ανωτέρω καθήκοντα και όσον αφορά τον έλεγχο του λογισμικού, η προσέγγιση του διαχειριστή (manager) σχετίζεται με την δόμηση και οργάνωση του λογισμικού και τον σχεδιασμό και ανάπτυξη των διαφόρων ελέγχων.

Η προσέγγιση του χρήστη (user) σχετίζεται με ελέγχους που επιτελούνται για να εξακριβωθεί ότι το λογισμικό ικανοποιεί τις απαιτήσεις των χρηστών. Η ανάλυση των απαιτήσεων των χρηστών (requirements analysis), οι προδιαγραφές του συστήματος (system specification) και ο έλεγχος παραλαβής (acceptance testing) είναι εργασίες που εντάσσονται στο πλαίσιο της προσέγγισης του χρήστη.

Πολύ σημαντικός είναι ο ρόλος του σχεδιαστή (designer) του συστήματος. Βασικό καθήκον του σχεδιαστή ενός λογισμικού συστήματος είναι η μετατροπή των διαφόρων απαιτήσεων των χρηστών και λοιπών επιθυμητών λειτουργιών, σε ένα συγκεκριμένο σχέδιο και δομή για το λογισμικό σύστημα. Η προσέγγιση αυτή εμπλέκεται σε διεργασίες όπως τις προδιαγραφές και τον σχεδιασμό του συστήματος (system design specifications και system design) καθώς και τούς έλεγχους ενοποίησης και συστήματος (integration και system testing).

Ο ρόλος του προγραμματιστή περιγράφει την τελευταία οπτική γωνία ελέγχου για μία ομάδας ανάπτυξης λογισμικού. Πέραν του προγραμματισμού κώδικα των διαφορετικών τμημάτων λογισμικού (modules), ο

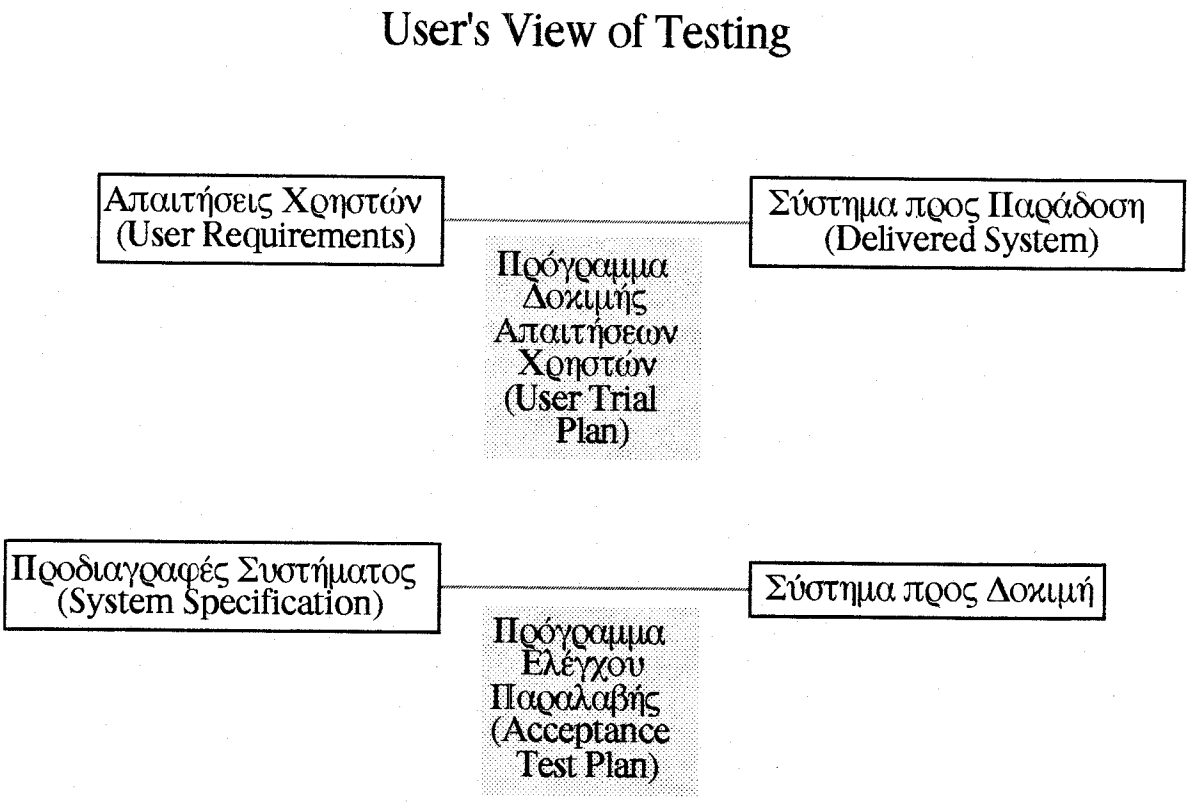
προγραμματιστής αναπτύσσει τις προδιαγραφές και σχεδιασμό των τμημάτων λογισμικού, προγραμματίζει τον κώδικα, και εκτελεί τον έλεγχο ενοτήτων (unit testing).

Μετά την γενική επισκόπηση των σταδίων που διακρίνονται στην ανάπτυξη λογισμικού καθώς και των εργασιών που εκτελούνται σε κάθε στάδιο (περιλαμβανομένων των ελέγχων λογισμικού) στρέφουμε την προσοχή μας σε περαιτέρω προσδιορισμό και αποσαφήνιση των διαφορετικών ελέγχων. Οι έλεγχοι που σχετίζονται με την οπτική γωνία του χρήστη, παρουσιάζονται στο *Σχήμα 4*. Στο άνω μέρος του σχήματος, εμφανίζεται η δοκιμή απαιτήσεων του χρήστη (user trial) η οποία σαν σκοπό έχει τον έλεγχο της ικανοποίησης των απαιτήσεως αυτών από το σύστημα που είναι έτοιμο προς παράδοση.

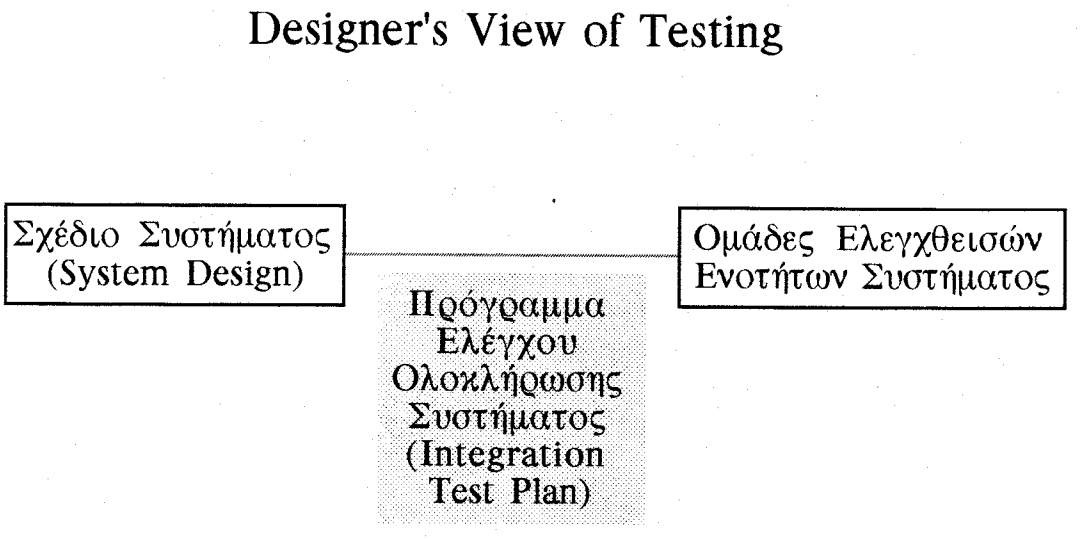
Στο κάτω μέρος του σχήματος, εμφανίζεται ο έλεγχος παραλαβής (acceptance test), το πρόγραμμα του οποίου προκύπτει από τον ρόλο αυτού του ελέγχου στον μεθοδικό έλεγχο τήρησης των (αρχικών) προδιαγραφών του συστήματος από το (τελικά υλοποιηθέν) σύστημα που είναι έτοιμο προς δοκιμή (ένα στάδιο προ του ετοιμοπαράδοτου συστήματος).

Το *Σχήμα 5* παρουσιάζει τους ελέγχους λογισμικού που σχετίζονται με την οπτική γωνία του σχεδιαστή (designer) ενός λογισμικού συστήματος. Ο έλεγχος ολοκλήρωσης του συστήματος (integration test) προκύπτει από το γενικό σχέδιο του λογισμικού συστήματος και την σχέση του με τις ελεγχθείσες ομάδες των ενοτήτων συστήματος, οι οποίες πρέπει να ανταποκρίνονται και να ικανοποιούν το γενικό σχέδιο. Το πρόγραμμα αυτού του ελέγχου περιλαμβάνει και μεθόδους για την τυχόν αφαίρεση σφαλμάτων.

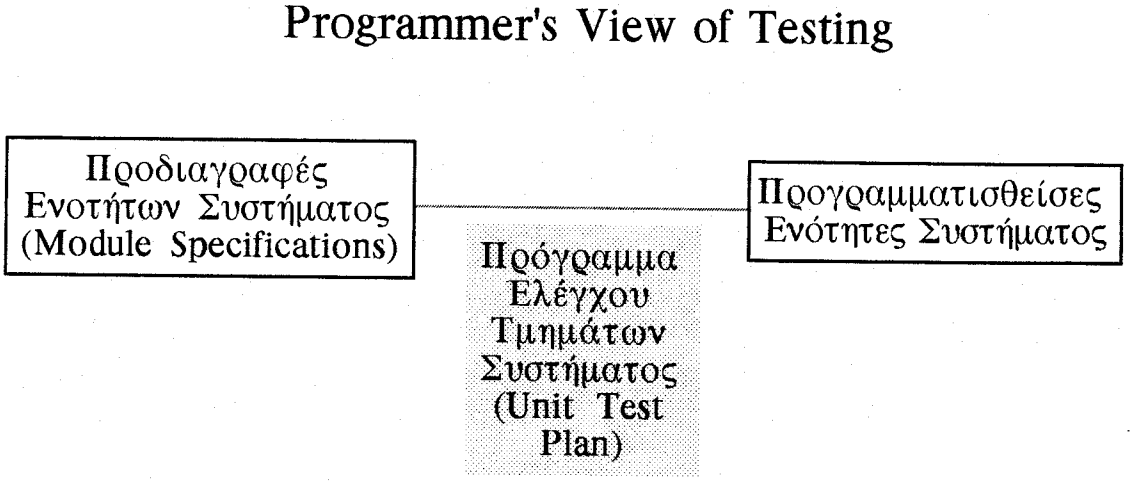
Οι έλεγχοι λογισμικού που έχουν σχέση με την οπτική γωνία που έχει ο προγραμματιστής του συστήματος (programmer) εμφανίζονται στο *Σχήμα 6*. Ο έλεγχος ενοτήτων του συστήματος (unit test) ελέγχει την τήρηση των προδιαγραφών των τμημάτων (modules) του λογισμικού συστήματος σε σχέση με τις προγραμματισθείσες (υλοποιηθείσες) ενότητες. Το πρόγραμμα ελέγχου μπορεί να περιλαμβάνει και μεθόδους βελτίωσης της τήρησης αυτών των προδιαγραφών.



Σχήμα 4. Έλεγχοι λογισμικού από την οπτική γωνία του χρήστη.



Σχήμα 5. Έλεγχοι λογισμικού από την οπτική γωνία του σχεδιαστή.



Σχήμα 6. Έλεγχοι λογισμικού από την οπτική γωνία του προγραμματιστή.

2.2.2 Μέθοδοι και Εργαλεία Ελέγχου. Η παρούσα ενότητα εξετάζει (α) διαφορετικές μεθόδους ελέγχου και (β) τα εργαλεία που χρησιμοποιούνται σε κάθε μέθοδο.

Πριν από την εφαρμογή του, ένας έλεγχος λογισμικού πρέπει να σχεδιασθεί προσεκτικά. Το γενικό αυτό σχέδιο του ελέγχου (test plan) στην ουσία πραγματεύεται την σύγκριση των πραγματικών αποτελεσμάτων του ελέγχου με τα αποτελέσματα που θα ήταν αναμενόμενα σε περίπτωση που το ελεγχόμενο λογισμικό δεν περιείχε λάθη ή άλλες ατέλειες. Μάλιστα, το ίδιο το λογισμικό σύστημα πρέπει να σχεδιάζεται έτσι ώστε να υποβάλλεται εύκολα σε όλους τους απαραίτητους ελέγχους (maximum testability) ενώ ταυτόχρονα να επιτρέπει την καλή τεκμηρίωση και παρακολούθηση των αποτελεσμάτων των ελέγχων (maximum traceability). Οι Ould και Unwin αναφέρουν ότι η εφαρμογή ανεπαρκούς θεωρίας και πρακτικής ελέγχων λογισμικού, επιτρέπει την δημιουργία τελικών λογισμικών συστημάτων των οποίων η ποιότητα και καλή λειτουργία δεν μπορούν να μετρηθούν και αποτιμηθούν αντικειμενικά (1989).

Ανάλογα με την προσέγγισή τους ως προς την δομή και ιεραρχία ενός λογισμικού συστήματος, οι έλεγχοι λογισμικού διακρίνονται σε "καθοδικούς" ελέγχους, που εξετάζουν το σύστημα από την κορυφή προς τα κάτω (top-down), και "ανοδικούς" ελέγχους που εξετάζουν το σύστημα από την βάση του προς τα πάνω (bottom-up). Στην περίπτωση ελέγχου από την κορυφή προς τα κάτω, διάφοροι μέθοδοι και εργαλεία επιστρατεύονται αρχικά για την εξέταση της γενικής δομής του λογισμικού συστήματος, έπεται δε ο έλεγχος των ιεραρχικά κατωτέρω τμημάτων - στην περίπτωση ελέγχου από την βάση προς τα πάνω, η φορά είναι αντίθετη. Γενικά, έχει εξακριβωθεί ότι η πλέον αποτελεσματική μέθοδος ελέγχου, συνίσταται σε μία κατάλληλη μίξη και των δύο μεθόδων (Sommerville, 1989).

Οι έλεγχοι λογισμικού διακρίνονται σε στατικούς (static) και δυναμικούς (dynamic), όπου οι στατικοί έλεγχοι ονομάζονται έτσι διότι, σε αντίθεση με τους δυναμικούς, δεν καταφεύγουν σε εκτέλεση κώδικα. Κατά τους στατικούς ελέγχους, εκτελείται (1) απλή επισκόπηση του κώδικα με σκοπό την ανακάλυψη λαθών και παραλείψεων, και (2) σύγκριση του προγράμματος με τις προδιαγραφές του με σκοπό να διαπιστωθεί η σύμφωνη με τις προδιαγραφές και σωστή λειτουργία των αντίστοιχων τμημάτων του συστήματος. Οι εικονικές εκτελέσεις προγραμμάτων με σκοπό την ανεύρεση λαθών αναφέρονται και σαν code walkthroughts (Ghezzi et al, 1991). Από την άλλη πλευρά, οι δυναμικοί έλεγχοι χρησιμοποιούνται για να βεβαιωθεί η

λειτουργία των τμημάτων κώδικα σύμφωνα με τις προδιαγραφές τους, όμως αυτή η επιβεβαίωση γίνεται με την εκτέλεση του κώδικα.

Η απλούστερη μορφή στατικού ελέγχου προγράμματος, είναι η απλή επισκόπηση του κώδικα, μέθοδος που θεωρείται αρκετά αποτελεσματική για την ανεύρεση προβλημάτων στον κώδικα (Sommerville, 1989). Συνήθως, η επισκόπηση γίνεται από μία ομάδα (που δύναται να περιλαμβάνει και μέλη της ομάδας ανάπτυξης και προγραμματισμού του εν λόγω λογισμικού συστήματος) η οποία ελέγχει τον κώδικα με σκοπό την ανακάλυψη λαθών. Να σημειωθεί ότι σε αυτή την περίπτωση, η εύρεση τέτοιων λαθών μπορεί να υποβοηθηθεί κατά πολύ από την ύπαρξη περιεκτικών καταλόγων ελέγχου (checklists) που περιέχουν ένα μεγάλο εύρος προβλημάτων που μπορεί να εμφανιστούν. Ο λεγόμενος δομικός έλεγχος (structural testing) είναι επίσης μία μορφή στατικού ελέγχου και συνίσταται στην ανάλυση ενός προγράμματος για την εξακρίβωση των δυνατών διαδρομών ροής εκτέλεσης με σκοπό τον καλύτερο σχεδιασμό ελέγχων και δοκιμαστικών δεδομένων για περαιτέρω δυναμική ανάλυση. Στην άλλη άκρη της απλής επισκόπησης, μπορεί να θεωρηθεί ότι υπάρχουν τα ειδικά λογισμικά εργαλεία που αποκαλούνται στατικοί αναλυτές (static analyzers), οι οποίοι αυτόματα "χτενίζουν" τον πηγαίο κώδικα, ψάχνοντας για διάφορα τυπικά προβλήματα όπως αυτά που συζητούνται στο επόμενο κεφάλαιο.

Οι δυναμικοί έλεγχοι χωρίζονται σε δύο τύπους: ελέγχους "μαύρου κουτιού" ή κλειστούς ελέγχους (black-box testing) και ελέγχους "άσπρου κουτιού" ή ανοικτούς ελέγχους (white-box testing). Στην παρούσα έκθεση, οι έλεγχοι άσπρου κουτιού θα αναφέρονται σαν έλεγχοι ανοικτής δομής ή ανοικτοί έλεγχοι ενώ οι έλεγχοι μαύρου κουτιού σαν έλεγχοι κλειστής δομής ή κλειστοί έλεγχοι.

Ο έλεγχος κλειστής δομής (black-box testing) γίνονται με βάση τις προδιαγραφές ενός τμήματος, χωρίς να ληφθούν υπ όψη οι λεπτομέρειες της εσωτερικής κατασκευής του λογισμικού που απαρτίζει αυτό το τμήμα. Οι είσοδοι (input) δεδομένων ελέγχονται, ενώ οι έξοδοι (output) δεδομένων συγκρίνονται με τα αναμενόμενα αποτελέσματα, χωρίς να χρησιμοποιείται καμμία πρόσβαση στην πηγαίο κώδικα (source code) της εφαρμογής. Οι έλεγχοι κλειστής δομής αναφέρονται σαν έλεγχοι μαύρου κουτιού διότι είναι σαν να ελέγχει κανείς ένα μαύρο κουτί στου οποίου το εσωτερικό δεν έχει πρόσβαση. Μία από τις καλύτερες (αν και παλαιότερες) αναφορές στους ελέγχους black-box περιλαμβάνεται στον Myers (1979).

Σε περίπτωση που ένας έλεγχος βασιστεί στις συγκεκριμένες

λεπτομέρειες της εσωτερικής δομής του εξεταζομένου τμήματος, ο έλεγχος αναφέρεται σαν έλεγχος ανοικτής δομής (του οποίου το περιεχόμενο είναι ορατό, εξ ου και η ονομασία white-box testing). Σε αντίθεση με τους ελέγχους κλειστής δομής, οι έλεγχοι ανοικτής δομής βασίζονται ουσιαστικά σε αυτή την εσωτερική δομή, για παράδειγμα, εξαναγκάζοντας διαφορετικά μονοπάτια του κώδικα να εκτελεσθούν, με ταυτόχρονη είσοδο διαφόρων δύστροπων δεδομένων. Με αυτή τους την πρόσβαση στον κώδικα της εφαρμογής, οι έλεγχοι white-box αναπτύσσουν πλέον κατάλληλα δοκιμαστικά δεδομένα για τον έλεγχο της σωστής λειτουργίας του προγράμματος μέσα από διάφορα μονοπάτια εκτέλεσης του κώδικα. Αν και εκφεύγει της εμβέλειας της παρούσας έκθεσης, δέον όπως αναφερθεί ότι η ανάπτυξη διαφόρων συνόλων και υποσυνόλων δοκιμαστικών δεδομένων δύναται να βασισθεί και σε ακριβείς μαθηματικές μεθόδους, π.χ. equivalence partitioning, όπως περιγράφεται στον Sommerville (1989).

Μετά την επισκόπηση διαφόρων μεθόδων ελέγχων, ας εξετάσουμε τώρα μερικά εργαλεία ελέγχων (Sommerville, 1989):

- γεννήτριες δοκιμαστικών δεδομένων (test data generators),
- δυναμικοί αναλυτές (dynamic analyzers),
- συγκριτές αρχείων (file comparators),
- προσομοιωτές (simulators),
- προγράμματα εκτύπωσης συμβολικών μεγεθών (symbolic dump programs),
- εργαλεία παρακολούθησης ροής προγράμματος (tracing tools), και
- εργαλεία για την αφαίρεση σφαλμάτων (debugging environments ή debuggers).

Όσον αφορά τυχόν πρόσθετο κώδικα που συχνά υπάρχει ανάγκη να γραφεί για τον έλεγχο και αποσφαλμάτωση μίας εφαρμογής, οι αποκαλούμενες <u>ρουτίνες οδήγησης</u> (test drivers) καλούν τις υπορουτίνες προς έλεγχο και εκτυπώνουν τυχόν ενδιάμεσα και τελικά αποτελέσματα για επιθεώρηση. Επίσης, οι <u>ρουτίνες υποκατάστασης</u> άλλων τμημάτων του λογισμικού (stubs) δύνανται να αποτελούνται από κώδικα που αντικαθιστά άλλες (πιό σύνθετες) ρουτίνες μόνο για τις ανάγκες του ελέγχου. Ενα τέτοιο παράδειγμα συνίσταται σε ρουτίνα ανεύρεσης πληροφοριών σε μία βάση δεδομένων η οποία μπορεί να αντικατασταθεί από ειδικό κώδικα ελέγχου ο οποίος να επιστρέφει σκόπιμα ωρισμένες μόνο πληροφορίες χωρίς καν να ψάχνει την βάση. Η αρχική ρουτίνα που κάλεσε τον κώδικα υποκατάστασης, "νομίζει" ότι οι επιστραφείσες

πληροφορίες ανευρέθηκαν στην βάση δεδομένων. Πολλές φορές η πλήρης αφαίρεση σφαλμάτων απαιτεί και συγγραφή πρόσθετων τμημάτων κώδικα διασύνδεσης (interface modules), ειδικά για τους ελέγχους - τμήματα δηλαδή που κανονικά δεν απαιτούνται από τις προδιαγραφές του λογισμικού συστήματος. Τέλος, υπάρχουν ακόμα και μετρικά στατιστικά μεγέθη (metrics), π.χ. για εκτίμηση της πολυπλοκότητας (complexity), αλληλοεξαρτήσεως (coupling) και συνεκτικότητας (cohesion) κώδικα.

Σαν τελευταίο στοιχείο προ της συμπλήρωσης της παρούσης ενότητας, θα αναφερθεί η ύπαρξη μαθηματικών μεθόδων για την επαλήθευση των προδιαγραφών ενός λογισμικού συστήματος με ακρίβεια και λιτότητα. Λόγω των εξειδικευμένων εφαρμογών τους, τέτοιες μέθοδοι δεν θα εξετασθούν περαιτέρω στα πλαίσια της παρούσας εργασίας.

3. ΣΧΕΔΙΑΣΗ ΚΑΙ ΕΚΤΕΛΕΣΗ ΕΛΕΓΧΩΝ ΛΟΓΙΣΜΙΚΟΥ

Στο κεφάλαιο αυτό επιχειρείται η συγκεκριμενοποίηση των εννοιών που παρουσιάστηκαν και συζητήθηκαν στο προηγούμενο κεφάλαιο, μέσω της σύνδεσης με το λογισμικό σύστημα του ΥΔΡΟΣΚΟΠΙΟΥ και με τα ειδικά χαρακτηριστικά της κατανεμημένης βάσης δεδομένων στο περιβάλλον ανάπτυξης INGRES και την γλώσσα τετάρτης γενεάς Windows4GL.

Μετά από μία εισαγωγική συζήτηση, παρουσιάζονται με αρκετές λεπτομέρειες οι διαφορετικοί τύποι ελέγχου που εφαρμόζονται στα πλαίσια ενός ολοκληρωμένου προγράμματος ελέγχου λογισμικού. Εξετάζονται πληρέστερα (α) οι έλεγχοι που επιτελεί ένας προγραμματιστής (εδώ περιλαμβάνεται ο έλεγχος ενοτήτων), (β) οι έλεγχοι που επιτελεί ο σχεδιαστής του συστήματος (περιλαμβανομένων των ελέγχων ενοποίησης και συστήματος), και (γ) οι έλεγχοι που έχουν σχέση με την οπτική γωνία του χρήστη (όπως ο έλεγχος παραλαβής), οι οποίοι χαρακτηρίζονται από μεγάλη σπουδαιότητα λόγω της έμφασης του όλου συστήματος του ΥΔΡΟΣΚΟΠΙΟΥ στις πολυάριθμες λειτουργίες διασύνδεσης με τον χρήστη.

3.1 Ανάπτυξη και Έλεγχος Λογισμικού στο ΥΔΡΟΣΚΟΠΙΟ

Αρχικά θα συζητήσουμε θέματα που σχετίζονται με τον έλεγχο μέρους του λογισμικού συστήματος που θα αναπτυχθεί στα πλαίσια του ΥΔΡΟΣΚΟΠΙΟΥ, συγκεκριμένα με τα διαλογικά (interactive) εκείνα τμήματα που υλοποιούν την σύνδεση της βάσης με τον χρήστη, μέσω περιβάλλοντος γραφικής διασύνδεσης (graphic user interface). Η παρούσα ενότητα θα βοηθήσει στην κατάρτιση ενός γενικού σχεδίου ελέγχου λογισμικού.

Η διασύνδεση της βάσης δεδομένων με τον τυπικό χρήστη του ΥΔΡΟΣΚΟΠΙΟΥ, υλοποιείται στο γραφικό περιβάλλον της Ingres (Interactive Graphics & Retrieval System) που ονομάζεται Windows4GL. Η Ingres είναι μία σχεσιακή βάση δεδομένων (relational data base), βασισμένη σε φόρμες (forms-based), εξ ού και η ονομασία των τμημάτων της Ingres, π.χ. Query-By-Forms (Husch και Husch, 1992). Το περιβάλλον αυτό είναι ένα τυπικό GUI (Graphical User Interface - γραφική διασύνδεση με τον χρήστη), ανάλογο με τα Microsoft Windows που λειτουργούν σε περιβάλλον MS-DOS καθώς και το γραφικό λειτουργικό σύστημα των υπολογιστών Apple Macintosh. Η γλώσσα Windows4GL είναι σχεδιασμένη για προγραμματισμό προσανατολισμένο σε αντικείμενα (object-oriented programming). Η ανάπτυξη του λογισμικού της διασύνδεσης της βάσης με τον χρήστη, διευκολύνεται πολύ από την ύπαρξη

ειδικού συντάκτη εφαρμογών (application editor), ο οποίος επιτρέπει την εύκολη δημιουργία και απεικόνιση μερών της εφαρμογής, όπως ενός πεδίου ή ενός ολόκληρου πίνακα αλλά και άλλων αντικειμένων, με την χρήση ποντικιού (mouse). Τα αντικείμενα αυτά δύνανται να συνοδεύονται από κώδικα (script) ο οποίος καθορίζει την λειτουργία τους και την επικοινωνία τους με άλλα αντικείμενα. Τέλος, τα αντικείμενα αυτά μαζί με τους κώδικές τους, τοποθετούνται σε φόρμες (forms) οι οποίες αντιπροσωπεύουν μία ενότητα του όλου λογισμικού διασύνδεσης και παρουσιάζονται σε παράθυρα (windows), τα οποία - κατά την ορολογία της Windows4GL - αποκαλούνται πλαίσια (frames).

Το λογισμικό της βάσης δεδομένων του ΥΔΡΟΣΚΟΠΙΟΥ αναπτύσσεται από διάφορες ομάδες (μεταξύ των οποίων η ομάδα του ΕΜΠ) κάθε μία των οποίων, οφείλει να ελέγξει εκείνες τις ενότητες του λογισμικού που ανέπτυξε. Η παρούσα έκθεση φιλοδοξεί να συντονίσει αυτές τις προσπάθειες, ειδικά όσον αφορά την κατάρτιση κατάλληλων προδιαγραφών για τους διαφορετικούς ελέγχους λογισμικού, και να παράσχει μία κοινή γλώσσα επικοινωνίας για την κοινοποίηση και σύγκριση των προσπαθείων και αποτελεσμάτων των διαφόρων μορφών ελέγχου λογισμικού.

Όπως τονίσθηκε στο προηγούμενο κεφάλαιο, είναι σημαντικό να ελεγχθούν τα μέρη εκείνα του ΥΔΡΟΣΚΟΠΙΟΥ που αναμένεται να χρησιμοποιούνται πιό συχνά. Δυστυχώς, συστήματα στα οποία η ασφάλεια είναι σημαντική, όπως το ΥΔΡΟΣΚΟΠΙΟ, είναι δύσκολο να ελεγχθούν με αποτελεσματικότητα, επειδή ένα πλήρες σχέδιο ελέγχων πρέπει να αναδημιουργήσει συνθήκες πολύ σπάνιων, μη ευνοϊκών για το σύστημα, γεγονότων. Για ένα σύστημα μάλιστα τόσο σύνθετο όπως το ΥΔΡΟΣΚΟΠΙΟ, πρέπει να εκτελούνται έλεγχοι καθ'όλη την διάρκεια ζωής του συστήματος (life-cycle testing), ήδη παράλληλα με την ανάπτυξη του συστήματος, ώστε τυχόν σφάλματα να ανακαλύπτονται νωρίς (Thornberg, 1991).

Ένα σημαντικό θέμα στην ανάπτυξη ενός ορθού και αποτελεσματικού σχεδίου ελέγχων, είναι ο καθορισμός των αρμοδιοτήτων κάθε μέλους μίας ομάδας ανάπτυξης λογισμικού, δηλαδή ο ορισμός του διαχειριστή (manager), σχεδιαστή (designer) και προγραμματιστή ή προγραμματιστών (programmer). Κάθε μέλος της ομάδας οφείλει να συμμετέχει στον σχεδιασμό διαφορετικών ελέγχων. Ένας προγραμματιστής για παράδειγμα, σχεδιάζει και υλοποιεί τον έλεγχο ενοτήτων (unit testing), ενώ ένα μέλος της ομάδας με αρμοδιότητες σχεδιαστή του συστήματος πρέπει να συνεργαστεί με τον προγραμματιστή κατά τους ελέγχους ολοκλήρωσης (integration testing) και συστήματος (system

testing) ώστε να βοηθήσει τον προγραμματιστή στην κατανόηση των γενικότερων προβλημάτων στον σχεδιασμό του συστήματος και στην μεταγλώττιση αυτών των προβλημάτων (υψηλού επιπέδου) σε (χαμηλού επιπέδου) προβλήματα ή βελτιώσεις κώδικα.

Θα πρέπει να τονισθεί ότι ανάμεσα στα τελικά προϊόντα των διεργασιών ανάπτυξης και υλοποίησης ελέγχων λογισμικού, θα πρέπει να περιλαμβάνεται και συλλογή ρουτινών που εκτελούν τους διαφορετικούς ελέγχους καθώς και τυχόν πληροφορίες για την εύκολη αναπαραγωγή των ελέγχων. Τέτοιες πληροφορίες θα είναι πολύτιμες σε περίπτωση μελλοντικής τροποποίησης του λογισμικού συστήματος, αφού με το πέρας της όποιας τροποποίησης θα πρέπει να εκτελεσθούν και πάλι πολλοί από τους ελέγχους που είχαν σχεδιασθεί και εκτελέσθηκαν με την αρχική συγγραφή του λογισμικού συστήματος.

Η πραγματοποίηση ελέγχων λογισμικού για ένα σύστημα όπως το ΥΔΡΟΣΚΟΠΙΟ πρέπει να νοείται ως καλύπτουσα και θέματα ποιότητας λογισμικού (software quality) τα οποία είναι εξαιρετικά σημαντικά ειδικά για ένα τέτοιο μεγάλο και σύνθετο σύστημα (Yorkes και Williams, 1990). Τα ακόλουθα χαρακτηριστικά που θεωρούνται επιθυμητά για ένα πραγματικό σύστημα σχεσιακής βάσεως δεδομένων,και για τα οποία έχει δε ήδη αποδειχθεί ότι εμπεριέχονται στην SQL (Hursh και Hursch, 1992), μπορούν να χρησιμοποιηθούν σαν ένας γενικός οδηγός για την ποιότητα του τελικού συστήματος του ΥΔΡΟΣΚΟΠΙΟΥ:

1. Οι πληροφορίες του σχεσιακού συστήματος δεδομένων να είναι ορισμένες σαφώς, σε λογικό επίπεδο, οι διαφορετικές δε καταστάσεις που τυχόν αντιπροσωπεύουν να αντιστοιχούν μονοσήμαντα σε (αριθμητικές ή λογικές) τιμές ενός πίνακα της βάσης (representation of information).
2. Όλα τα δεδομένα να είναι προσβάσιμα με λογική αναφορά στο κατάλληλο σημείο της βάσης (guaranteed logical accecibility).
3. Συστηματική καταγραφή όλων των δυνατών τιμών των δεδομένων, περιλαμβανομένων των ελλιπών (missing) ή κενών (blank) δεδομένων (systematic representation of missing information).
4. Ευκολία στην τοπική πρόσβαση των χρηστών που πρέπει να μπορούν να έχουν πρόσβαση στην βάση σαν να χειρίζονται απλά δεδομένα (dynamic online catalog).
5. Ύπαρξη πλήρους γλώσσας αλλαγής και έρευνας καταχωρημένων πληροφοριών όπως η SQL (comprehensive data sublanguage).
6. Χρήση φορμών οθόνης που δύνανται να ανανεωθούν από το σύστημα με

πλήρως υλοποιημένες όλες τις ικανότητες αλλαγών (updatable views).

7. Δυνατότητα για "έξυπνες" παρεμβολές νέων δεδομένων, αλλαγών στα υπάρχοντα δεδομένα και διαγραφές δεδομένων (high-level insert, update and delete).
8. Οι εσωτερικές (χαμηλού επιπέδου) λεπτομέρειες αποθήκευσης των δεδομένων να μην επηρεάζουν την λειτουργία της βάσης σε επίπεδο χρήστη (physical data independence).
9. Αλλαγές σε λογικό επίπεδο των πινάκων της βάσης να μην επηρεάζουν την λειτουργία της βάσης σε επίπεδο χρήστη (logical data independence).
10. Η κάθε "υποβάση" (πίνακας) να είναι ανεξάρτητη ως προς την ακεραιότητά της (integrity independence).
11. Η λειτουργία της (τοπικής) βάσης ως κατανεμημένης να μην επηρεάζει την γενική λειτουργία της βάσης (distribution independence).
12. Χαμηλού επιπέδου λειτουργίες να μην είναι δυνατόν να επηρεάσουν αρνητικά κανόνες ακεραιότητας και σχετικούς περιορισμούς στην λειτουργία της βάσης σε ψηλότερο επίπεδο (nonsubversion).

Μετά από αυτή την γενική συζήτηση γύρω από τον σχεδιασμό των ελέγχων λογισμικού του ΥΔΡΟΣΚΟΠΙΟΥ, στρέφουμε τώρα την προσοχή μας σε συγκεκριμένους ελέγχους.

3.2 Έλεγχοι Προγραμματιστή (Programmer)

Οι ακόλουθες ενότητες εξετάζουν σε λεπτομέρεια τους ελέγχους εκείνους που σχεδιάζονται και εκτελούνται από έναν προγραμματιστή. Επιγραμματικά, αυτοί οι έλεγχοι καλύπτουν την αποσφαλμάτωση των τμημάτων κώδικα (modules) και την δημιουργία ελεγχθεισών ενοτήτων του συστήματος (units).

Τα καθήκοντα του προγραμματιστή όσον αφορά την γενικότερη ανάπτυξη του συστήματος, περιλαμβανομένων των ελέγχων λογισμικού, είναι τα ακόλουθα:

- σύνταξη των προδιαγραφών των υποενοτήτων ή τμημάτων (module specification),
- προγραμματισμός των τμημάτων, και
- σχεδιασμό του ελέγχου ενοτήτων (unit testing).

Η ποιότητα των προδιαγραφών των τμημάτων του συστήματος πρέπει να ελεγχθεί για τα ακόλουθα χαρακτηριστικά:

- Συνεκτικότητα (cohesion), δηλαδή την καλή σύνδεση των διαφορετικών κομματιών μίας υποενότητας. Η συνεκτικότητα αυτή μπορεί να είναι λογικής ή άλλης λειτουργικής φύσεως (π.χ. κομμάτια ενός module που πρέπει να εκτελεσθούν σειριακά).
- Κρύψιμο των πρωτογενών δεδομένων που χειρίζεται ένα module - ο χρήστης πρέπει να μπορεί να επιτελεί όλες τις δυνατές χρήσεις των δεδομένων με προβλεπόμενες κλήσεις του module, παρά με απ' ευθείας πρόσβαση στα δεδομένα.
- Η αλληλοεξάρτηση των διαφόρων τμημάτων πρέπει να είναι ελάχιστη. Ένα module δεν πρέπει να χρησιμοποιεί (καλώντας) κώδικα ενός άλλου module, ούτε πρέπει να περνάει σύνθετες λογικές δομές σε άλλο module.
- Η πολυπλοκότητα του συστήματος να είναι ελάχιστη, σχεδιάζοντας σωστά ιεραρχημένα και ανεξάρτητα modules.
- Τα τμήματα (modules) να έχουν ένα μέσο μέγεθος που θα κριθεί κατάλληλο και στα πλαίσια της συγκεκριμένης εφαρμογή. Εάν τα τμήματα κώδικα είναι πολύ μικρά, το σύστημα θα περιέχει πολλά τμήματα, ενώ εάν είναι μεγάλα, θα είναι πολύπλοκα και δύσκολα στους ελέγχους.
- Τα modules να είναι σχεδιασμένα έτσι ώστε να είναι δεκτικά σε ελέγχους.

Επιθεωρήσεις των προδιαγραφών ενός τμήματος και λογικός έλεγχος διαφόρων (ακραίων) συνθηκών, είναι ένας από τους πιό απλούς τρόπος να ελεγχθούν οι προδιαγραφές αυτές (reviews, που περιλαμβάνουν walkthroughs ή inspections). Επ' αυτού, η τεκμηρίωση των προδιαγραφών του τμήματος είναι ένα σημαντικό βοήθημα, στο οποίο πρέπει να περιλαμβάνονται τα κάτωθι:

1. όνομα του τμήματος,
2. λειτουργία (σε λεπτομέρειες),
3. είσοδοι και έξοδοι δεδομένων (inputs/outputs), και
4. τυχόν εξωτερικές ενέργειες που επηρεάζουν την λειτουργία του τμήματος (π.χ. σε περίπτωση που ένα τμήμα διαβάζει την τιμή ενός αυτοματοποιημένου εργαστηριακού μετρητή, το γύρισμα του διακόπη για την έναρξη της λειτουργίας αυτού του μετρητή είναι απαραίτητο για την λειτουργία του τμήματος).

3.2.1 Έλεγχος Ενοτήτων Συστήματος (Unit Testing). Πριν από την έναρξη του ελέγχου ενοτήτων, είναι καλό να ελεγχθεί θεωρητικά (π.χ. με μολύβι και χαρτί) το προσχέδιο αυτού του ελέγχου, ώστε να ελεγχθεί η βασική πορεία των εργασιών ελέγχου και κατά πόσον το σχέδιο ελέγχου καλύπτει όλες τις πτυχές και λειτουργίες της ενότητας.

Ο έλεχος ενοτήτων ελέγχει την ποιότητα (καλή λειτουργία) των ενοτήτων του συστήματος. Για την επίτευξη αυτού του στόχου, χρησιμοποιούνται οι ακόλουθες μέθοδοι:

- Δοκιμαστικές (φανταστικές) εκτελέσεις κώδικα (dry running). Κατ' ουσίαν, ο προγραμματιστής ελέγχει την εκτέλεση του προγράμματος, παρακολουθώντας την ροή του κώδικα χειρωνακτικά (με χαρτί και μολύβι).
- Κατ' αναλογία με την προηγούμενη μέθοδο, μία ολόκληρη ομάδα ανθρώπων μπορεί να ελέγξει τον κώδικα με την καθοδήγηση του προγραμματιστή ο οποίος είναι τελικά ο υπεύθυνος για την διόρθωση τυχόν ανευρεθέντων λαθών (structured walkthroughs).
- Διάφοροι τυπικοί έλεγχοι, που σχετίζονται με αναμενόμενο χρόνο εκτέλεσης, σωστή μεταχείριση δεδομένων εισόδου, σωστή αναφορά λαθών, σωστή διαχείριση τυχόν αρχείων που δεν ευρίσκονται στο δίσκο (ή άλλο μέσο αποθήκευσης), σωστή ανταλλαγή παραμέτρων με άλλα τμήματα ή ενότητες, σωστή παραγωγή και εμφάνιση δεδομένων εξόδου.
- Η λεγόμενη στατική ανάλυση (static analysis) του κώδικα των τμημάτων των ενοτήτων, συνίσταται στον έλεγχο και σύγκριση των χαρακτηριστικών λειτουργίας όλων των τμημάτων και ενοτήτων, χωρίς να τρέχει το σύστημα. Η σύγκριση αυτή γίνεται για την αποκάλυψη τυχόν προβλημάτων στην κανονική ροή του προγράμματος, ή την διαχείριση των δεδομένων και άλλων πληροφοριών, όπως (Sommerville, 1989):
  - κώδικας που δεν εκτελείται ποτέ (unreachable code),
  - κατεύθυνση της ροής εκτέλεσης στην μέση ενός βρόγχου (branches into loops),
  - αδήλωτες μεταβλητές (undeclared variables),
  - ακατάλληλος τύπος ή αταίριαστος αριθμός παραμέτρων (parameter type/number mismatches),
  - συναρτήσεις ή υπορουτίνες που δεν καλούνται ποτέ (uncalled functions/procedures),
  - χρήση μεταβλητών των οποίων ο τύπος δεν έχει δηλωθεί

(uninitialized variables used),

- μη χρήση των αποτελεσμάτων που επεστράφησαν από μία συνάρτηση (non-usage of function results),
- υπέρβαση των διαστάσεων ενός πίνακα (array bound violation), και
- κακή χρήση ενός δείκτη (pointer misuse).

Για τον έλεγχο ενοτήτων των χαμηλού επιπέδου τμημάτων κώδικα απαιτούνται οδηγοί (test drives) για την παροχή ρουτινών εισόδου/εξόδου. Για τον έλεγχο ενοτήτων των υψηλού επιπέδου τμημάτων κώδικα, απαιτούνται κώδικες υποκατάστασης (stubs) για την παροχή των λειτουργιών τυχόν ελλειπόντων τμημάτων χαμηλού επιπέδου.

Τέλος, για την δημιουργία μίας αντικειμενικής έννοιας ποιότητας λογισμικού (software quality), απαιτείται η υιοθέτηση κριτηρίων διατυπωμένων με ακρίβεια. Μερικά τέτοια κριτήρια είναι:

- ευκολία χρήσης (ease of use) για έναν τυπικό χρήστη,
- ταχύτητα εκτέλεσης, με διαφορετικά συστήματα, και
- μεταφερσιμότητα της βάσης σε άλλα συστήματα (ικανότητα του κώδικα να μεταφέρεται εύκολα σε άλλο μηχανικό υλικό - portability).

Οι έλεγχοι προγραμματιστή στο ΥΔΡΟΣΚΟΠΙΟ μπορεί να αφορούν π.χ. τμήματα του συστήματος που ανήκουν σε ένα πλαίσιο, τον κώδικα του ίδιου του πλαίσιου και τυχόν υπορουτίνες και συναρτήσεις που συνοδεύουν το πλαίσιο. Η εμβέλεια αυτού του ελέγχου οριοθετείται από τον χαμηλότερου επιπέδου κώδικα σε γλώσσες τρίτης γενεάς, όπως C, και τον κώδικα Windows4GL εντός ενός πλαισίου, καλύπτει δε μέχρι και το επίπεδο του κώδικα 4GL.

3.3 Έλεγχοι Σχεδιαστή (Designer)

Ο έλεγχος ενοποίησης συστήματος (integration testing) σχετίζεται με την προσέγγιση του σχεδιαστή (designer) του συστήματος, του οποίου τα καθήκοντα έχουν ως ακολούθως:

- Χωρισμός του λογισμικού συστήματος σε λογικές ενότητες με βάση τις προδιαγραφές του Συστήματος.
- Προσδιορισμός των λειτουργιών και διασύνδεσης (interface) κάθε ενότητας.

- Καθορισμός της επικοινωνίας και συνεργασίας των ενοτήτων αυτών για την ολοκληρωμένη λειτουργία του συστήματος.
- Επιλογή μηχανικού υλικού (hardware) και γλώσσας προγραμματισμού για κάθε ενότητα του συστήματος, με βάση τις απαιτήσεις των χρηστών, τις προδιαγραφές του συστήματος και τις λεπτομέρειες λειτουργίας, διασύνδεσης και επικοινωνίας των ενοτήτων.

Ο επιτυχής σχεδιασμός του συστήματος είναι απαραίτητη προϋπόθεση της επιτυχημένης υλοποίησης ενός λογισμικού συστήματος (Ould και Unwin, 1989). Οι ακόλουθοι έλεγχοι σχεδιάζονται και εκτελούνται από τον σχεδιαστή του συστήματος:

1. Έλεγχος της ορθότητας των προδιαγραφών του συστήματος
2. Έλεγχος του σχεδίου του συστήματος σε σχέση με τις προδιαγραφές του.
3. Έλεγχος της ενοποίησης των ενοτήτων του συστήματος (integration testing)
4. Έλεγχος της ολοκλήρωσης των ενοτήτων του συστήματος (integration testing).

3.3.1 Έλεγχος Σχεδιασμού Συστήματος. Για τον έλεγχο των προδιαγραφών και του σχεδίου του συστήματος, εφαρμόζονται τα εξής κριτήρια και διαδικασίες:

- Πληρότητα (πλήρης και ορθή κάλυψη όλων των στοιχείων που περιλαμβάνονται στις προδιαγραφές του συστήματος). Ο έλεγχος πληρότητας αρχίζει με απλή επισκόπηση και επιθεώρηση των προδιαγραφών του συστήματος, ώστε να βεβαιωθεί ο σχεδιαστής ότι κάθε χαρακτηριστικό των προδιαγραφών έχει περιληφθεί στο σχέδιο του συστήματος. Ακολουθεί λεπτομερής έλεγχος της πληρότητας των εσωτερικών αλληλο-αναφορών στοιχείων του σχεδίου. Με την συμπλήρωση του ελέγχου πληρότητας του σχεδίου συστήματος, πρέπει να έχουν εξαληφθεί όλες οι ασάφειες, ανεπαρκείς εξηγήσεις ή και αναφορές σε ανύπαρκτα στοιχεία του σχεδίου.
- Συνέπεια (έλλειψη αντικρουόμενων στοιχείων). Η συνέπεια συνεπάγεται έλλειψη αντιθέσεων σε (α) χρήση των δεδομένων, (β) τυχόν λανθασμένες κλήσεις άλλων λειτουργιών, και (γ) προβλήματα συγχρονισμού.
- Σκοπιμότητα (υλοποιήσιμο σχέδιο συστήματος, επαρκείς δυνατότητες

επέκτασης και συντήρησης). Στους παράγοντες που μπορούν να δημιουργήσουν προβλήματα σκοπιμότητας για την υλοποίηση, περιλαμβάνονται τεχνικά θέματα, μηχανικό υλικό, ανθρώπινοι παράγοντες, θέματα απόδοσης, αξιοπιστία, πολυπλοκότητα, και μελλοντική επέκταση του συστήματος. Ο έλεγχος σκοπιμότητας είναι επιθυμητό να περιλαμβάνει και κοστολογικές εκτιμήσεις.

- Δεκτικότητα σε ελέγχους (εύκολα υλοποιήσιμοι έλεγχοι για όλα τα στοιχεία του σχεδίου). Σε σχέση με αυτό το κριτήριο, εξετάζονται παράγοντες που συντελούν ή εμποδίζουν την δεκτικότητα του λογισμικού συστήματος σε ελέγχους. Μερικοί τέτοιοι παράγοντες είναι η καλή τεκμηρίωση υπολογισμών και εύκολη πρόσβαση σε ενδιάμεσα αποτελέσματα, πρόβλεψη για εύκολη ανίχνευση και επανεκκίνηση διεργασιών, απλότητα λειτουργίας και πρόσβασης στον κώδικα, καλή τεκμηρίωση και ορατότητα των διεργασιών διασύνδεσης των διαφορετικών τμημάτων του συστήματος, πρόβλεψη καλού χειρισμού λαθών, καλή οργάνωση και ορθή κλήση/χρήση κοινών υπορουτινών και συναρτήσεων, καθώς και περιεκτική και ορθή τεκμηρίωση όλων των τμημάτων του κώδικα. Να τονισθεί ότι τυχόν κακή δεκτικότητα ενός λογισμικού συστήματος σε ελέγχους, τείνει να χειροτερεύει κατά την διάρκεια ανάπτυξης του συστήματος (Ould και Unwin, 1989).

3.3.2 Έλεγχος Ενοποίησης (Integration Testing). Στόχος του ελέγχου ενοποίησης (integration testing) είναι να επαληθεύσει την λειτουργικότητα, απόδοση και αξιοπιστία ενός λογισμικού συστήματος. Ο έλεγχος ενοποίησης στοχεύει στην αποκάλυψη λαθών και ασυνεπειών είτε μεταξύ ομάδων λογισμικού είτε μεταξύ λογισμικού και μηχανικού υλικού. Ο έλεγχος ενοποίησης παραλαμβάνει τμήματα κώδικα (modules) τα οποία έχουν ήδη ελεγχθεί κατά τον έλεγχο τμημάτων (module testing, μέρος του unit testing), τα ενοποιεί σε ομάδες και ελέγχει αυτές τις ομάδες με συγκεκριμένα τεστ που καθορίζονται στο γενικότερο σχέδιο του ελέγχου ενοποίησης. Να τονισθεί ότι κατά την διάρκεια ελέγχου ενοποίησης, χρησιμοποιούνται έλεγχοι κλειστής δομής (black-box testing) αφού η εσωτερική λειτουργία των τμημάτων έχει ήδη ελεγχθεί κατά την διάρκεια του ελέγχου τμημάτων (module testing), στα πλαίσια του ελέγχου ενοτήτων (unit testing). Τελικά, ο έλεγχος ενοποίησης παραδίδει ένα ολοκληρωμένο σύστημα το οποίο είναι έτοιμο για τον έλεγχο συστήματος (system testing) που εξετάζεται στην επόμενη ενότητα.

Η εμβέλεια του ελέγχου ενοποίησης εξαρτάται από τους ακόλουθους

παράγοντες (Ould και Unwin, 1989):

- Μέγεθος και πολυπλοκότητα του λογισμικού.
- Μηχανικό υλικό.
- Γενικότερη φιλοσοφία και εμβέλεια του ελέγχου τμημάτων (module testing).
- Διαθέσιμα λογισμικά εργαλεία.

Το σχέδιο του ελέγχου ενοποίησης πρέπει να περιλαμβάνει τα κάτωθι:

- Οργανωτικές λεπτομέρειες: διαδικασίες που θα ακολουθήσει η ομάδα ελέγχου ενοποίησης, μηχανικό υλικό και εργαλεία που θα χρησιμοποιηθούν.
- Στρατηγική: καθορισμός βημάτων, ενδιάμεσων και τελικών στόχων.
- Σχέδιο τμημάτων λογισμικού που θα αντιμετωπισθούν ως ενότητες, ειδικές συνθήκες και απαιτήσεις κάθε ενότητας, εξάρτηση ενοτήτων από προηγούμενες.

Το σχέδιο του ελέγχου ενοποίησης πρέπει να επιθεωρηθεί από όλες τις ομάδες ανάπτυξης λογισμικού, με στόχο την αποτίμηση της πληρότητας, σκοπιμότητας και συνέπειάς του. Οι συγκεκριμένοι έλεγχοι πρέπει να είναι πλήρως τεκμηριωμένοι όσον αφορά την προετοιμασία που απαιτείται, διαδικασία ελέγχου, και την διαδικασία που έπεται του ελέγχου ώστε να τερματισθούν ορθά τα λογισμικά τεστ, να καταγραφούν όλα τα αποτελέσματα και να επιστραφούν όλες οι ελεγχόμενες ενότητες λογισμικού στην κατάσταση που ήταν προ του ελέγχου. Σημαντικός είναι ο ρόλος της ομάδας ελέγχου ποιότητας, η οποία θα βεβαιώσει ότι το ολοκληρωμένο σύστημα ανταποκρίνεται στα πρότυπα του γενικώτερου έργου.

Κατά την διάρκεια του ελέγχου ενοποίησης, είναι σχεδόν αδύνατο να ελεγχθούν όλες οι δυνατές διαδρομές εκτέλεσης του κώδικα, κάτω από όλες τις δυνατές συνθήκες. Επομένως, πρέπει να υπάρχει κατάλληλη μεθοδολογία επιλογής και σχεδιασμού του ελέγχου κατά τέτοιο τρόπο ώστε να βελτιστοποιηθεί η έκταση και βάθος κάλυψης του ελέγχου ενοποίησης. Όπως προαναφέρθηκε, κατά την διάρκεια του ελέγχου ενοποίησης εκτελούνται (κατά κανόνα) κλειστοί έλεγχοι (black-box testing) αφού η εσωτερική λειτουργία των υποενοτήτων και τμημάτων κώδικα έχει ελεγχθεί σε προηγούμενους ελέγχους.

3.3.3 Έλεγχος Συστήματος (System Testing). Ο έλεγχος συστήματος (system testing) μπορεί να θεωρηθεί σαν μία οριακή μορφή ελέγχου ενοποίησης (integration testing) που έχει σαν σκοπό την ανακάλυψη και αφαίρεση λαθών εκτελώντας τεστ σε ολόκληρο το σύστημα. Το κύριο καθήκον κατά τον έλεγχο συστήματος συνίσταται σε ελέγχους κλειστής δομής (black-box) και αντιπαραβολή ολόκληρου του λογισμικού με τις προδιαγραφές του συστήματος. Με αυτή την έννοια, η επιτυχής συμπλήρωση του ελέγχου συστήματος οδηγεί στον έλεγχο αποδοχής (acceptance testing). Ο έλεγχος συστήματος είναι ιδιαίτερα σημαντικός στο ΥΔΡΟΣΚΟΠΙΟ για την εξασφάλιση της ορθής λειτουργίας της σχεδιαζόμενης βάσης σαν κατανεμημένη βάση σε δίκτυο.

Οι ακόλουθες διαδικασίες ελέγχων εντάσσονται στον έλεγχο συστήματος (Ould και Unwin, 1989), πρέπει επομένως να πραγματοποιηθούν στο σύνολο του λογισμικού συστήματος του ΥΔΡΟΣΚΟΠΙΟΥ:

- Λειτουργικός έλεγχος (functional testing), όπου έλεγχοι κλειστής δομής ελέγχουν την ορθή και σύμφωνη με τις προδιαγραφές εκτέλεση των λειτουργιών του συστήματος.
- Έλεγχος διασύνδεσης χρήστη (user interface testing), που μπορεί να αρχίσει νωρίς στον σχεδιασμό και υλοποίηση του λογισμικού ώστε τυχόν μη φιλικά χαρακτηριστικά να απορριφθούν έγκαιρα.
- Έλεγχος επίκλησης βοηθητικών οδηγιών (help information testing), οι οποίες πρέπει να είναι απλές στην πρόσβαση και εύκολες στην χρήση, ακόμα και για χρήστες με πολύ μικρή εμπειρία σε υπολογιστές.
- Έλεγχος ορίων του συστήματος (limit testing), που θα ελέγχει το σύστημα σε οριακές συνθήκες λειτουργίας.
- Έλεγχος υπό συνθήκες έντασης (stress testing), π.χ. πολλούς χρήστες ταυτόχρονα ή πολλές ταυτόχρονες προσβάσεις και αλλαγές δεδομένων.
- Έλεγχος όγκου δεδομένων (volume testing), όπου το σύστημα υποβάλλεται σε μεγάλους όγκους δεδομένων.
- Έλεγχος ασφάλειας (security testing), π.χ. δοκιμή πρόσβασης μίας βάσης δεδομένων από χρήστες που δεν έχουν κατάλληλη εξουσιοδότηση.
- Έλεγχος απόδοσης (performance testing), που περιλαμβάνει και ελέγχους ταχύτητας (ειδικά στην περίπτωση του ΥΔΡΟΣΚΟΠΙΟΥ, πρέπει να ελεγχθούν οι λειτουργίες πρόσβασης της βάσης από προσωπικό υπολογιστή - PC - αντί για σταθμό εργασίας).

- Έλεγχους συμβατότητας (compatibility testing) με παλαιότερες εκδόσεις (versions) του ίδιου συστήματος, ειδικά εξασφάλιση συμβατότητας προς τα άνω (upwards compatibility).
- Έλεγχος αξιοπιστίας (reliability testing) σε σχέση με τις προδιαγραφές του συστήματος, υπό συνθήκες που πρέπει να προσομοιάζουν τό πραγματικό περιβάλλον στο οποίο θα τρέχει το σύστημα.
- Έλεγχος εξόδου από λάθη (error exit testing).
- Έλεγχος ανάνηψης από λάθη (recovery testing), που είναι πολύ σημαντικός για συστήματα ευαίσθητα σε θέματα ασφαλείας.
- Έλεγχος δυνατότητας συντήρησης (maintenance potential testing).
- Σε περίπτωση λογισμικών συστημάτων που δύνανται να εγκατασταθούν σε διάφορετικά υπολογιστικά συστήματα, είναι σημαντικό να γίνει έλεγχος εγκατάστασης (installation testing) που μπορεί να ελέγξει ακραίες και μη ευνοϊκές συνθήκες εγκατάστασης (π.χ. εγκατάσταση τμημάτων της βάσης σε PC αντί για σταθμό εργασίας).
- Έλεγχος αποθήκευσης (storage testing), όπου ελέγχονται π.χ. προβλήματα σχετικά με την απόπειρα αποθήκευσης ή πρόσβασης ή αλλαγής εξαιρετικά μεγάλου όγκου δεδομένων.
- Έλεγχος χειρωνακτικών διεργασιών (manual testing), για συστήματα που απαιτούν εργασίες όπως χειρωνακτικό φόρτωμα ταινιών ή χειρωνακτική ενεργοποίηση κάποιας μακρυνής συσκευής.
- Έλεγχος πληροφοριών χρήστη (user information testing), όπου η τεκμηρίωση του λογισμικού (τα εγχειρίδια) ελέγχεται για πληρότητα, σαφήνεια, επεξήγηση μηνυμάτων λαθών κλπ.

Ο έλεγχος του συνόλου ενός λογισμικού συστήματος (system testing) διακρίνεται στα κατωτέρω επίπεδα:

- έλεγχος "α" (alpha testing)
- έλεγχος "β" (beta testing)

Ο έλεγχος "α" γίνεται από την ομάδα ή οργανισμό ανάπτυξης του λογισμικού. Κατά την διάρκεια του ελέγχου "α", είναι ενδεδειγμένο να σχεδιάζονται τεστ που εμπλέκουν, κατά το δυνατόν, τυπικούς χρήστες του μελλοντικού συστήματος. Η επιτυχής συμπλήρωση του ελέγχου "α" δεν σημαίνει ότι το σύστημα είναι πλήρως απαλλαγμένο από τυχόν λάθη, ούτε ότι το σύστημα συμφωνεί με τις προδιαγραφές του - απλώς ότι το σύστημα είναι

μίας ελάχιστα αποδεκτής ποιότητας ώστε να μπορούν να εκτελεσθούν πρόσθετοι έλεγχοι. Να σημειωθεί το εξής πρόβλημα ορολογίας: ο Sommerville αναφέρει τον έλεγχο "α" σαν "acceptance testing" (1989).

Ο έλεγχος "β" γίνεται από επιλεγμένους χρήστες του προγράμματος, οι οποίοι χρησιμοποιούν το λογισμικό σύστημα και αναφέρουν τυχόν προβλήματα στην ομάδα ανάπτυξης του συστήματος. Η χρήση του ελέγχου "β" συνίσταται στην αποκάλυψη τυχόν προβλημάτων του συστήματος σε συνθήκες πραγματικής χρήσης, επομένως είναι δυνατόν να υποβοηθήσει στην ανεύρεση προβλημάτων που δεν αναμένονται από τους σχεδιαστές και προγραμματιστές του συστήματος. Ένας τέτοιος έλεγχος είναι εξαιρετικά σημαντικός στην περίπτωση του ΥΔΡΟΣΚΟΠΙΟΥ λόγω της πολυπλοκότητας και όγκου του συστήματος.

Σε σχέση με την συγκεκριμένη εφαρμογή του ΥΔΡΟΣΚΟΠΙΟΥ, ο έλεγχος ολοκλήρωσης θα ασχολείται με υποενότητες της εφαρμογής που γενικά αποτελούνται από περισσότερα του ενός πλαίσια (multi-frame application modules). Από την άλλη πλευρά, ο έλεγχος συστήματος έχει την στενότερη εμβέλεια με την έννοια ότι εξετάζεται όλη η εφαρμογή ως μία ενότητα μόνο.

Με την συμπλήρωση των ελέγχων "α" και "β", το λογισμικό σύστημα είναι έτοιμο για έλεγχο παραλαβής, ο οποίος, αν και θεωρείται συνέχεια του ελέγχου συστήματος, εξετάζεται στην επόμενη ενότητα.

3.4 Έλεγχοι Σχετιζόμενοι με Χρήστες

Στην ενότητα αυτή εξετάζονται οι έλεγχοι που σχετίζονται με τα ακόλουθα τμήματα της ανάπτυξης λογισμικού που αφορούν τον χρήστη (Ould και Unwin, 1989):

- Τις απαιτήσεις των χρηστών (user requirements), οι οποίες εκφράζουν τις συγκεκριμένες απαιτήσεις και επιθυμίες των χρηστών για τις λειτουργίες του λογισμικού συστήματος.
- Τις προδιαγραφές του συστήματος (system specification), βασισμένες στις απαιτήσεις των χρηστών.
- Το σύστημα που είναι έτοιμο προς δοκιμή (system for trial), αναπτυχθέν με βάση τις προδιαγραφές του.
- Τελικό προϊόν των ελέγχων χρηστών είναι το σύστημα που είναι έτοιμο για παράδοση (delivered system) στους μελλοντικούς χρήστες.

Αρχικά ελέγχονται οι απαιτήσεις των χρηστών με βάση το εξής κριτήρια:

1. Πληρότητα (completeness). Το σχέδιο των απαιτήσεων των χρηστών στο οποίο βασίζεται η ανάπτυξη του λογισμικού συστήματος πρέπει να εκφράζει ορθά και με πληρότητα τις αρχικές απαιτήσεις και επιθυμίες των χρηστών. Κατ' αυτόν τον τρόπο, η εμβέλεια του λογισμικού συστήματος προκύπτει σαφώς και χωρίς κενά. Ειδική προσοχή απαιτείται στον ορισμό όλων των ειδών των πιθανών χρηστών του συστήματος.
2. Συνέπεια (consistency). Η έκφραση των απαιτήσεων των χρηστών πρέπει να χαρακτηρίζεται από συνέπεια (όσον αφορά εσωτερικές και εξωτερικές αναφορές) και να αποφεύγει την χρήση υπερβολικά τεχνικής ή άλλης ειδικής ορολογίας που πιθανόν να είναι κατανοητή σε ένα στενό μόνο κύκλο μηχανικών λογισμικού και τεχνικών προγραμματισμού.
3. Σκοπιμότητα και δυνατότητα πραγματοποίησης (feasibility). Οι απαιτήσεις πρέπει να καλούν την υλοποίηση τεχνικά ρεαλιστικών χαρακτηριστικών τα οποία να είναι δυνατόν να αναπτυχθούν σε ένα λογικό πλαίσιο χρονικής ανάπτυξης και να είναι συντηρήσιμα με μικρό κόστους.
4. Δεκτικότητα σε ελέγχους (testability). Προς τούτο, το κείμενο των απαιτήσεων των χρηστών πρέπει να είναι απλό και εύκολα κατανοητό.
5. Ευκολία παραπομπής (referability): οι απαιτήσεις να είναι συγκεκριμένες και να μην υπάρχει αλληλοκάλυψη (κάθε απαίτηση να αφορά θέματα που δεν έχουν καλυφθεί σε άλλη ενότητα, να είναι δε οι απαιτήσεις μονοσήμαντα ορισμένες)

Στις τεχνικές που μπορούν να χρησιμοποιηθούν για τον έλεγχο των απαιτήσεων των χρηστών, περιλαμβάνονται:

- απλός εποπτικός έλεγχος,
- συνεντεύξεις των χρηστών και ανάλυση των απαιτήσεών τους,
- κριτική ανασκόπηση (review) του κειμένου των απαιτήσεων και επιθυμιών των χρηστών από ομάδα στην οποία περιλαμβάνονται μέλη της ομάδας ανάπτυξης του λογισμικού αλλά και χρήστες, και
- κατάλογοι ελέγχου (checklists) επιθυμητών χαρακτηριστικών για τις απαιτήσεις των χρηστών - τέτοιοι κατάλογοι συνήθως συντάσσονται στα πλαίσια ανάπτυξης λογισμικών συστημάτων συγκεκριμένου είδους.

Ακολούθως ελέγχονται οι προδιαγραφές του συστήματος, οι οποίες μπορεί να είναι εκφρασμένες σε κείμενο, εικόνες ή και μαθηματούς/αλγεβρικούς συμβολισμούς:

1. Όσον αφορά την ορθότητα, συνέπεια, έλλειψη αμφιβόλου ερμηνείας και δεκτικότητα σε ελέγχους, το σύστημα πρέπει να ελεγχθεί ως προς τις κατωτέρω ιδιότητες (system qualification):
   - Συνέπεια (consistency). Στην περίπτωση αυτού του ελέγχου, οι προδιαγραφές του συστήματος διαβάζονται και ελέγχονται από περισσότερους από έναν (ανεξάρτητους) αναγνώστες, οι οποίοι προσπαθούν να εντοπίσουν ασυνέπειες στην χρήση όρων και συμβόλων. Επίσης, οι προδιαγραφές αυτές μπορούν να συγκριθούν με άλλες παρόμοιες προδιαγραφές συστημάτων.
   - Έλλειψη αντιφάσεων (no contradictions) όσον αφορά την εσωτερική κατασκευή των προδιαγραφών (εσωτερικές αντιφάσεις) αλλά και την διασύνδεση του συστήματος με άλλα συστήματα (εξωτερικές αντιφάσεις).
   - Έλλειψη ασαφειών (unambiguity). Τυχόν χρήση εξειδικευμένων όρων πρέπει να συνοδεύεται από κατάλληλη εξήγηση και διασαφήνιση, ειδικά σε περίπτωση που κοινοί όροι της καθομιλουμένης χρησιμοποιούνται με ειδική σημασία. Αυτό χρειάζεται ιδιαίτερη προσοχή στην περίπτωση της Ελληνικής γλώσσας, στην οποία εμφανίζεται να υπάρχει ένα σημαντικό ποσοστό πληροφορικών όρων που αποτελούν αδόκιμες μεταφράσεις των αντίστοιχων Αγγλικών λέξεων. (Το Παράρτημα Αγγλο-Ελληνικών όρων Πληροφορικής αποτελεί μία απόπειρα εξάλειψης αυτής της ασάφειας.) Επίσης, οι προδιαγραφές του συστήματος πρέπει να υιοθετούν και να ενσωματώνουν πλήρως την ευρύτερη ορολογία που χρησιμοποιείται στο επίπεδο της εταιρείας ή άλλου φορέα κατασκευής του συγκεκριμένου λογισμικού συστήματος.
   - Δεκτικότητα σε ελέγχους (testability). Κατά το δυνατόν, κάθε χαρακτηριστικό ή λειτουργία που αναπτύσσεται και παρουσιάζεται στις προδιαγραφές του συστήματος, πρέπει να είναι έτσι διατυπωμένο ώστε να επιδέχεται εύκολο έλεγχο με κάποια τεστ του λογισμικού συστήματος. Μάλιστα, οι προδιαγραφές μπορούν να προτείνουν και κάποιο προσχέδιο πιθανών τεστ που θα μπορούσαν να χρησιμοποιηθούν για τον έλεγχο του κάθε χαρακτηριστικού ή

λειτουργίας.

- Δυνατότητα και ευκολία εσωτερικής αλληλοαναφοράς (referability). Κάθε ενότητα που συζητά και αναπτύσσει συγκεκριμένα χαρακτηριστικά του συστήματος, πρέπει να προκύπτει με σαφήνεια από προηγούμενες συζητήσεις (περί απαιτήσεων των χρηστών). Επίσης, κάθε τέτοια ενότητα πρέπει να οδηγεί σε επόμενες ενότητες στις οποίες περιγράφεται με σαφήνεια η υλοποίηση του χαρακτηριστικού ή λειτουργίας.
- Σκοπιμότητα και δυνατότητα επίτευξης (feasibility). Το σύστημα που περιγράφεται στις προδιαγραφές πρέπει να είναι τεχνικά υλοποιήσιμο, στα πλαίσια των οικονομικών και χρονικών περιορισμών. Το προτεινόμενο σύστημα πρέπει να ανταποκρίνεται στα χαρακτηριστικά και επίπεδο γνώσεων των χρηστών. Ένα σύστημα που προορίζεται για χρήστες έμπειρους στην χρήση υπολογιστών (π.χ. ένα έμπειρο σύστημα για μηχανικούς) πρέπει να έχει διαφορετική διασύνδεση (interface) από ένα σύστημα για χρήστες με ελάχιστη εμπειρία σε υπολογιστές (π.χ. μία λογοτεχνική βάση για πεζογράφους και ποιητές). Επίσης, το προτεινόμενο σύστημα πρέπει να είναι οικονομικό στην λειτουργία του και να χαρακτηρίζεται από εύκολη συντήρηση (maintenance).

2. Όσον αφορά την ικανοποίηση των συγκεκριμένων απαιτήσεων των χρηστών (αναφερόμενοι στην έννοια της εγκυρότητας ή ορθότητας - validation - του συστήματος, δηλαδή τον έλεγχο του συστήματος σε σχέση με τις απαιτήσεις των χρηστών) το σύστημα πρέπει να ελεγχθεί ως προς τις κατωτέρω ιδιότητες:
   - Ιδιότητες (properties) του συστήματος, που έχουν σχέση με την απόδοσή του. Οι ιδιότητες αυτές μπορεί να είναι είτε ρητά και συγκεκριμένα διατυπωμένες (explicit properties), είτε να υπονοούνται ή να συνάγονται σαφώς από περιγραφές των προδιαγραφών (implicit properties). Ο έλεγχος για τις ιδιότητες του συστήματος μπορεί να γίνει με αναλυτικές (analytical), π.χ. αλγεβρικές, μεθόδους με κατασκευή μαθηματικών μοντέλων του συστήματος, ή εμπειρικές (empirical), όπου ο έλεγχος είναι δυνατός με κατασκευή και παρατήρηση της συμπεριφοράς πρωτότυπου μοντέλου του συστήματος (prototyping). Συχνά, ακόμα και γλώσσες προγραμματισμού τετάρτης γενεάς χρησιμοποιούνται για την

δημιουργία ενός τέτοιου μοντέλου.

- Λειτουργίες (functions) του συστήματος, που έχουν σχέση με τις αντιδράσεις του συστήματος σε εξωτερικές παρεμβάσεις, π.χ. η δημιουργία μίας γραφικής παραστάσεως στην οθόνη κατόπιν απαιτήσεως του χρήστη. Αυτές οι λειτουργίες του συστήματος, ελέγχονται για πληρότητα (completeness), συνάφεια και ουσιαστικότητα (relevance) καθώς και ορθότητα (correctness).

Πέραν των προδιαγραφών του, ελέγχεται το σύστημα στην φάση που είναι έτοιμο προς τελική δοκιμή, οπότε πρέπει:

1. Να ανταποκρίνεται στις απαιτήσεις των χρηστών σε όλη την διάρκεια ζωής του συστήματος. Αυτή η απαίτηση εκφεύγει των απλών ελέγχων που γίνονται στην τελική φάση της υλοποίησης ενός λογισμικού συστήματος, πρέπει δε να υπάρχει πρόβλεψη για περιοδικούς ελέγχους καθώς οι απαιτήσεις και επιθυμίες των χρηστών και τα χαρακτηριστικά του λογισμικού συστήματος αλλάζουν με τον χρόνο.
2. Να αποτελεί ορθή και ακριβή υλοποίηση των προδιαγραφών του, σε σχέση και με τον έλεγχο παραλαβής (acceptance test).

Στρέφουμε τώρα την προσοχή μας στον έλεγχο παραλαβής (acceptance testing), που αναφέρεται στα τεστ που γίνονται πάνω στο τελικό σύστημα, ώστε να αφαιρεθούν τυχόν προβλήματα προ της παραδόσεως στους χρήστες.

3.4.1 Έλεγχος Παραλαβής (Acceptance Testing). Ο έλεγχος παραλαβής έχει την έννοια ότι εάν ένα λογισμικό σύστημα περάσει αυτόν τον έλεγχο, οι μελλοντικοί χρήστες του συστήματος θα το αποδεχθούν (μάλιστα η ακριβής μετάφραση του αγγλικού όρου είναι έλεγχος αποδοχής, αν και ο δόκιμος ελληνικός όρος είναι έλεγχος παραλαβής). Να τονισθεί ότι ο έλεγχος παραλαβής αφορά όχι μόνο το λογισμικό αλλά και την σχετική τεκμηρίωση, δηλαδή τα εγχειρίδια χρήσεως (user manuals) και αναφοράς (reference manuals).

Με την συμπλήρωση του ελέγχου συστήματος, ο έλεγχος παραλαβής έχει σαν σκοπό την διαπίστωση ικανοιητικής πλήρωσης των απαιτήσεων των χρηστών (user requirements). Στην φάση αυτή, εμπλέκονται ο διαχειριστής και σχεδιαστής της εφαρμογής με σκοπό την σύνδεση των συγκεκριμένων δυνατοτήτων της εφαρμογής με τις τυπικές απαιτήσεις των χρηστών, όπως αρχικά διατυπώθηκαν.

Πολύ σημαντικό είναι να καθιερώσει κανείς δεδομένα κριτήρια αποδοχής, ώστε να είναι σαφής και μονοσήματα ωρισμένη η έκβαση του ελέγχου παραλαβής (Thornberg, 1991). Στην περίπτωση που οι προδιαγραφές του συστήματος είναι σαφείς και δεκτικές σε έλεγχο (μία από τις προϋποθέσεις που συζητήθηκε σε προηγούμενες παραγράφους), ο έλεγχος παραλαβής προκύπτει χωρίς δυσκολία. Τα τεστ του ελέγχου παραλαβής αφορούν τα εξής περιεχόμενα των προδιαγραφών του συστήματος:

- Λειτουργικότητα (functionality), για τον έλεγχο της οποίας απλώς σχεδιάζονται επαναληπτικά τεστ που παρακολουθούν την συμπεριφορά και αντίδραση του συστήματος σε διαφορετικές συνθήκες.
- Χαρακτηριστικά (attributes), όπως ρυθμός εξυπηρέτησης, χωρητικότητα, φορητότητα σε άλλες συνθήκες και μηχανικά συστήματα, ή ακρίβεια. Όπως και στην περίπτωση της λειτουργικότας του συστήματος, τα χαρακτηριστικά του ελέγχονται με συγκεκριμένα τεστ που δημιουργούν κατάλληλες συνθήκες για τον έλεγχο συγκεκριμένων μεγεθών.
- Περιορισμούς (constraints) για την σωστή λειτουργία του συστήματος, η τήρηση των οποίων ελέγχεται.

Για τυχόν αλλαγές, που θα είναι αναπόφευκτες στην περίπτωση ενός συστήματος του μεγέθους του ΥΔΡΟΣΚΟΠΙΟΥ, να σημειωθεί ότι ορισμένοι έλεγχοι πρέπει να ξαναγίνουν μετά την υλοποίηση των αλλαγών (regression testing; Thornberg, 1991).

ΒΙΒΛΙΟΓΡΑΦΙΑ

Chezzi, C., M. Jayazeri και D. Mandrioli. *Fundamentals of Software Engineering.* Prentice-Hall International, 1991.

Date, C.J. *A Guide to INGRES.* Addison-Wesley, 1987.

Hursch C.J. και J.L. Hursch. *Ingres SQL Developer's Guide.* Windcrest/McGraw-Hill, 1992.

Lamb, D.A. *Software Engineering: Planning for Change.* Prentice-Hall International, 1988.

Myers, G.J. *The Art of Software Testing.* Wiley, New York, 1979.

Ould, M.A. και C. Unwin (eds). *Testing in Software Development.* The British Computer Society Monographs in Informatics, P.A. Samet (ed), Cambridge University Press, 1989.

Sommerville, I. *Software Engineering.* Addison-Wesley, 3rd edition, 1989.

Thornberg, R. *Master Software Test Plan for the National Water Information System II.* U.S. Geological Survey, Reston Virginia, Draft, 11/26/1991.

Yorke, T.H. και O. Williams. *Design of a National Water Information System by the U.S. Geological Survey.* U.S. Geological Survey, Reston Virginia, Report, September 1990.

Παράρτημα
ΑΓΓΛΟ-ΕΛΛΗΝΙΚΟ ΛΕΞΙΚΟ
ΒΑΣΙΚΩΝ ΟΡΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ABF	*Application-By-Forms* (εργαλείο της Ingres για την δημιουργία εφαρμογών με φόρμες)
acceptance testing	έλεγχος παραλαβής, έλεγχος αποδοχής
AI	*artificial intelligence*
algorithm	αλγόριθμος
alpha testing	έλεγχος "α"
analog	αναλογικός
animation	(κινηματογραφική) κίνηση, κινούμενα σχέδια
ANSI	*American National Standards Institute*
application	εφαρμογή
artificial intelligence	τεχνητή νοημοσύνη ή ευφυία
ASCII	κώδικες αλφαριθμητικών χαρακτήρων *(American Standard Code for Information Interchange)*
assembler	συμβολομεταφραστής
BASIC	γλώσσα προγραμματισμού BASIC *(Beginner's All-Purpose Symbolic Instruction Code)*
beta testing	έλεγχος "β"
bit	(δυαδικό) ψηφίο
black box testing	κλειστός έλεγχος, έλεγχος (τύπου) μαύρου κουτιού
bottom-up	από κάτω προς τα πάνω, από την βάση προς την κορυφή
bridge	γέφυρα
byte	ψηφιολέξη
CAD	*computer-aided design*
CAM	*computer-aided manufacturing*
CD-ROM	ψηφιακός δίσκος *(Compact Disk)*
central unit	κεντρική μονάδα (υπολογιστή)
chip	ολοκληρωμένο κύκλωμα
client (module)	(τμήμα) πελάτης
clock device	ρολόι, ωρολογιακή συσκευή, χρονική συσκευή
command	διαταγή, προσταγή

compiler	μεταγλωττιστής
computer animation	(κινηματογραφική) κίνηση με τη βοήθεια υπολογιστή
computer network	δίκτυο υπολογιστών
computer-aided design	σχεδιασμός με τη βοήθεια υπολογιστή
computer-aided manufacturing	παραγωγή με τη βοήθεια υπολογιστή
coprocessor	συνεπεξεργαστής
databank	τράπεζα πληροφοριών (ή δεδομένων)
database	βάση δεδομένων
DBA	*database administrator* (επιβλέπων διαχειριστής βάσεως δεδομένων με εξουσίες ελέγχου της πρόσβασης άλλων χρηστών)
DBMS	*Data Base Management System* (Σύστημα Διαχείρισης Βάσεως Δεδομένων)
DCL	*Data Control Language* (κατηγορία εντολών SQL που ελέγχουν την πρόσβαση στα δεδομένα και την βάση)
DDL	*Data Definition Language* (κατηγορία εντολών SQL που ορίζουν αντικείμενα μίας βάσης δεδομένων)
DML	*Data Manipulation Language* (κατηγορία εντολών SQL που εκτελούν λειτουργίες ανευρέσεως και αλλαγής των δεδομένων μίας βάσης)
debugging	αφαίρεση (διόρθωση) σφαλμάτων
desktop publishing	συστήματα επιτραπέζιων εκδόσεων
digital	ψηφιακός
directory	κατάλογος (αρχείων)
disk	δίσκος
disk drive	μονάδα δίσκου
distributed	κατανεμημένος(-η)
dot matrix printer	εκτυπωτής ακίδων, κρουστικός εκτυπωτής
E-Mail	*electronic mail*
editor	συντάκτης
electronic mail	ηλεκτρονικό ταχυδρομείο
EQUEL	*embedded QUEL* (ενσωματομένη QUEL)
ESQL	*embedded SQL* (ενσωματομένη SQL)

expert system	σύστημα εμπειρογνωμοσύνης, έμπειρο σύστημα
FAX	τηλεομοιοτυπία, τηλεφωτοαντιγραφή
feature	χαρακτηριστικό
file	αρχείο
file name	όνομα αρχείου
file name extension	επέκταση ονόματος αρχείου
floating point	(αριθμός) κινητής υποδιαστολής, πραγματικός (αριθμός)
floppy disk	εύκαμπτος δίσκος, δισκέττα
folder	φάκελλος ή ντοσιέ (αρχείων)
FORTRAN	γλώσσα προγραμματισμού FORTRAN *(FORmula TRANslator)*
function	λειτουργία, συνάρτηση
GBF	*Graph-By-Forms* (τμήμα της Ingres)
hard disk	σκληρός δίσκος
hard drive	μονάδα σκληρού δίσκου
I/O	*input/output*
IC	*integrated circuit*
incremental	αυξητικός
index file	αρχείο δεικτών
INGRES	*Interactive Graphics and Retrieval System*
input/output	είσοδος/έξοδος (δεδομένων)
instruction	εντολή
integrated circuit	ολοκληρωμένο κύκλωμα
integration testing	έλεγχος ενοποίησης
interactive	διαλογικό, αλληλεπιδρών
interface	διασύνδεση (χρήστη), ενδιάμεσο (χρήστη-υπολογιστή)
interpreter	διερμηνευτής
IQUEL	*interactive QUEL* (διαλογική QUEL)
ISQL	*interactive SQL* (διαλογική SQL)
keyboard	πληκτρολόγιο
LAN	*Local Area Network*
laser printer	εκτυπωτής (ακτίνων) λέιζερ
line editor	συντάκτης (κειμένου με επεξεργασία) γραμμής
local area network	τοπικό δίκτυο (υπολογιστών)
logical	λογικός

machine language	γλώσσα μηχανής
maiframe	κεντρικός (υπολογιστής)
maintenance	συντήρηση
memory	μνήμη
menu driven	οδηγούμενο από μενού
micro	μικρουπολογιστής *(microcomputer)*
microchip	ολοκληρωμένο κύκλωμα
microprocessor	μικροεπεξεργαστής
mini	μινιυπολογιστής *(minicomputer)*
modem	διαμορφωτής/αποδιαμορφωτής *(MOdularor/DEModulator)*
module	τμήμα, υποενότητα (προγράμματος)
monitor	συσκευή οθόνης *(TV monitor)*
mouse	ποντίκι
multimedia	πολυμέσα
network	δίκτυο
neural networks	νευρωνικά δίκτυα
object code	αντικειμενικός ή τελικός κώδικας
operating system	λειτουργικό σύστημα
overload	υπερφόρτωση
performance	απόδοση, επίδοση
peripheral device	περιφερειακή συσκευή
physical	φυσικός (δηλ. πραγματικός)
pop up menus	αναδυόμενα μενού
printer	εκτυπωτής
program	πρόγραμμα
pull down menus	πτυσσόμενα ή συρόμενα ή κυλιόμενα μενού
QBF	*Query-By-Forms* (διαλογικό τμήμα της Ingres για την διαχείριση των δεδομένων μίας βάσης με την βοήθεια φορμών)
QUEL	*Query Language*
RAM	μνήμη άμεσης (ή τυχαίας) προσπέλασης *(Random Access Memory)*
random file	αρχείο άμεσης (ή τυχαίας) προσπέλασης
RBF	*Report-By-Forms* (τμήμα της Ingres)
regression testing	παλινδρομικός έλεγχος
repeater	αναμεταδότης

robot	ρομπότ
robotics	ρομποτική
ROM	μνήμη μόνο για ανάγνωση *(Read Only Memory)*
router	δρομολογητής
scanner	(οπτικός) σαρωτής
scientific notation	επιστημονική γραφή (αριθμών)
screen editor	συντάκτης (κειμένου με επεξεργασία) οθόνης
sequential file	σειριακό αρχείο, αρχείο σειριακής προσπέλασης
shell	φλοιός, κέλυφος
simulation	προσομοίωση
software	λογισμικό
source code	πηγαίος κώδικας, αρχικός κώδικας
spreadsheet	(ηλεκτρονικό) φύλλο εργασίας, ηλεκτρονικός πίνακας
SQL	*Structured Query Language* (βασική γλώσσα για την αποθήκευση και διαχείριση δεδομένων σε βάσεις)
string	συμβολοσειρά, σειρά (ή αλυσίδα) χαρακτήρων
stub	πρόγραμμα υποκατάστασης
subdirectory	υποκατάλογος (αρχείων)
system prompt	ένδειξη (ή γραμμή) αναμονής (ή ετοιμότητας) συστήματος
system testing	έλεγχος συστήματος
teletext	τηλεκειμενογραφία
telex	τηλέτυπο
test driver	πρόγραμμα οδήγησης ελέγχου
testing ή test	έλεγχος, δοκιμή
text editor	συντάκτης κειμένου
throughput	ρυθμός εξυπηρέτησης, ρυθμοαπόδοση
top-down	από πάνω προς τα κάτω, από την κορυφή προς την βάση
unit	ενότητα (προγράμματος)
user interface	διασύνδεση χρήστη, ενδιάμεσο χρήστη-υπολογιστή
utility program	βοηθητικό πρόγραμμα
validation	επικύρωση

verification	επαλήθευση
version	έκδοση
VIFRED	*Visual-Forms-Editor* (υποσύστημα της Ingres για σύνταξη φορμών)
VIGRAPH	*Visual-Graphics-Editor* (τμήμα της Ingres)
(code ή module) walkthrough	διέλευση (κώδικα ή τμήματος κώδικα)
WAN	*Wide Area Network*
white-box testing	ανοικτός έλεγχος, έλεγχος (τύπου) άσπρου κουτιού
wide area network	δίκτυο (υπολογιστών) ευρείας περιοχής
word processing	επεξεργασία κειμένου
word processor	επεξεργαστής κειμένου
workstation	σταθμός εργασίας
WYSIWYG	"παίρνετε ότι βλέπετε", με την έννοια ότι κάτι π.χ. θα εκτυπωθεί, όπως εμφανίζεται στην οθόνη *(What-You-See-Is-What-You-Get)*