ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Έλεγχος Λογισμικού και Διαχείριση Επικινδυνότητας»

Ταγκαλάκη Βασιλική

Μ3980004

Επιβλέπων Καθηγητής: Νικόλαος Μαλεύρης

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΘΗΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2000

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΕΧΟΜΕΝΑ *1*

EXECUTIVE SUMMARY *3*

ΠΕΡΙΛΗΨΗ *9*

ΕΙΣΑΓΩΓΗ *18*

ΚΕΦΑΛΑΙΟ 1 *20*

1.1 Εισαγωγικά **20**

1.2 Επικινδυνότητα Λογισμικού (Software Risk) **20**
1.2.1 Ορισμός Επικινδυνότητας Λογισμικού 20
1.2.2 Τύποι Κινδύνων Έργων Λογισμικού 22

1.3 Σχέση Ασφάλειας και Επικινδυνότητας Λογισμικού **25**

1.4 Διαχείριση Επικινδυνότητας Λογισμικού (Software Risk Management) **25**
1.4.1 Η Έννοια της Διαχείρισης Επικινδυνότητας 25
1.4.2 Επίπεδα Διαχείρισης Επικινδυνότητας 27
1.4.3 Κύκλος Ζωής Διαχείρισης Επικινδυνότητας 28
1.4.4 Διαχείριση Επικινδυνότητας στον Κύκλο Ζωής Ανάπτυξης Λογισμικού 33

1.5 Ευριστική Ανάλυση Επικινδυνότητας (Heuristic Risk Analysis) **36**
1.5.1 Inside-Out προσέγγιση 36
1.5.2 Outside-In προσέγγιση 37

1.6 Διαχείριση Επικινδυνότητας και Διαχείριση Ανάπτυξης Συστήματος Λογισμικού 40

ΚΕΦΑΛΑΙΟ 2 *42*

2.1 Εισαγωγικά **42**

2.2 Σχέση Ελέγχου Λογισμικού και Διαχείρισης Επικινδυνότητας **42**

2.3 Έλεγχος Βασιζόμενος στην Επικινδυνότητα και τις Απαιτήσεις **46**

ΚΕΦΑΛΑΙΟ 3 *48*

3.1 Εισαγωγικά **48**

3.2 Παράγοντες Επικινδυνότητας Λογισμικού **48**
3.2.1 Κρισιμότητα Λογισμικού (CRIT) 49
3.2.2 Μέγεθος Λογισμικού (SIZE) 51
3.2.3 Μέθοδος Ελέγχου (TESM) 53
3.2.4 Εμπειρία Ομάδας Ελέγχου (TEXP) 57
3.2.5 Πολυπλοκότητα Λογισμικού (CPLX) 58
3.2.6 Συχνότητα Χρήσης (FREQ) 60
3.2.7 Χρόνος Ελέγχου (TIME) 62
3.2.8 Διαθέσιμοι Πόροι (RESO) 63

3.3 Έλεγχος Αντικειμενοστραφούς Λογισμικού **64**
3.3.1 Βασική μονάδα για έλεγχο (Basic unit for testing) 64
3.3.2 Κληρονομικότητα (Inheritance) 64
3.3.3 Ενθυλάκωση (Encapsulation) 64
3.3.4 Πολυμορφισμός (Polymorphism) 65
3.3.5 Έλεγχος White-Box 65
3.3.6 Έλεγχος Black-Box 65
3.3.7 State-Based Έλεγχος 65

3.3.8 Επάρκεια και Κάλυψη (Adequacy and Coverage) 66
3.3.9 Στρατηγικές Ενοποίησης (Integration Strategies) 66
3.3.10 Στρατηγική Διαδικασίας Ελέγχου (Test Process Strategy) 66
3.3.11 Συμπεράσματα για τον Έλεγχο και την Επικινδυνότητα των Αντικειμενοστραφών Συστημάτων Λογισμικού 66

ΚΕΦΑΛΑΙΟ 4 68

4.1 Εισαγωγικά 68

4.2 Καθορισμός Πολιτικής σε σχέση με το Επιτρεπτό Επίπεδο Επικινδυνότητας 68

4.3 Isorisk Chart 69

4.4 Συνολική Διαδικασία Διαχείρισης Επικινδυνότητας σε σχέση με τον Έλεγχο Λογισμικού 70

ΚΕΦΑΛΑΙΟ 5 73

5.1 Συμπεράσματα 73

5.2 Μελλοντική Έρευνα 74

ΒΑΣΙΚΟΙ ΟΡΙΣΜΟΙ 76

ΒΙΒΛΙΟΓΡΑΦΙΑ – ΑΝΑΦΟΡΕΣ 78

EXECUTIVE SUMMARY

1. Introduction

Software failure is the nightmare of the Information Age. Vast software engineering resources are ubiquitously spent throughout all industries to avert this nightmare. However, in the end it is impossible to guarantee that software is perfect, nor is it possible to predict and eliminate every possible gliche that could attack software from the outside world as it executes.

Although, much of the failure could be avoided by dealing with various risk factors than waiting for problems to occur and then trying to react. In other words, an adequate and systematic risk management could provide insight to potential problem areas and identify, address and eliminate them before they derail the project.

Risk management has long been recognized as an important part of project management in all the traditional engineering disciplines. Software risk management has received increasing attention over the last decade, as new opportunities have opened and worldwide competition for software business has become keener.

Most of the approaches that have been developed suggest that risk management is a process that must be applied throughout all the stages of the software development life cycle, in order to track, manage and control effectively the several risks that threaten the software.

Even though the afore mentioned view has turned out to be true, it is important to highlight the fact that, while in all the other phases of the software life cycle there is a great chance of risk elimination, in the testing phase there is always some risk remaining after the end of the phase.

On top of that, software testing, whose principal objective is to gain confidence in the software, is one of the most important activities in the development process, where a great amount of the available resources is spent. Therefore, it is essential for the success of the project; to take into account the risks involved, in order to make effective testing decisions.

Having in mind the afore mentioned observations, the present work focuses on the software testing phase and inquire into its relationship with the risk management process. It also introduces a number of factors that affect the total risk remaining after

the completion of testing, and it proposes an overall risk management process in relation with the testing that taking place.

2. Software Risk

According to David Cluch, a risk is a combination of an abnormal event or failure and the consequences of that event or failure to a system's operators, users, or environment.

Like any other engineering project, software development is a risky business. However, the risks involved in software engineering is much more complicated due to mainly three broad reasons: (i) software is an extremely complex end creative artifact and software development could not be rigorously defined as engineering process yet, (ii) it is much more human-oriented, and (iii) sound risk analysis and reliability testing methods are in lacking.

3. Software Risk Management

If we accept that risk includes technical and managerial factors, which threaten the success of the project, risk management is the process of identifying these threats, analyzing them, quantifying their effects, and implementing plans that counteract their negative effects.

In other words, risk management asks, "What exactly could go wrong, and what do we do when it does?" The first sub question looks for anything that could fail and trigger a string of failures; the second one produces alternative courses of action if a problem occur.

The practice of risk management involves the following four main steps:

1) Risk Analysis which consisting of:
   - ⇨ Risk Identification
   - ⇨ Risk Estimation
   - ⇨ Risk Prioritization
2) Risk Management Planning
3) Risk Resolution
4) Risk Monitoring

4. Correlation between Risk Management and Software Testing

Software testing is often associated with two basic terms: *verification*, which refers to ensuring correctness from phase to phase of the software development cycle, and *validation*, which involves checking the software against the requirements.

Even though testing can prove the presence of vulnerabilities in software by finding faults in it, it is not able to guarantee its perfection by demonstrating that there are no faults in it. Moreover, during the testing phase, as the errors are identified, they may result in changes to the code, which can lead to additional problems and ripple effect errors, especially in code that is high risk due to size and complexity.

If we consider that the main objective for a testing team is to do the most cost effective testing which can ensure that the product is reliable enough, safe enough and satisfies the user/customer requirements, we establish the fact that something like that is extremely difficult, if not impossible, to accomplish, since there is never enough time to test everything completely. What is required, is understanding of the risks that relate with software faults, risks for the user or the customer, the developer or the supplier, or even the responsible for the maintenance of the software. In other words, testing is nothing more than risk management, whose aim is to achieve confidence in the software.

5. Software Risk Factors

After a close examination, we found that the risk remaining after the end of the testing phase is determined by the following set of factors:

1) Software Criticality (CRIT)
Software criticality can be defined as the function of the losses caused dy unsatisfactory outcomes {CRIT = f(L(UO) (1)} who can range from catastrophic (loss of life or permanent disability) to negligible (no damage, no injury). Criticality characterises the software and categorises it to safety-critical, semi-critical and non-critical.

2) Software Size (SIZE)
Three sizing quantities are used, so as to comprise both conventional and object-oriented software: (i) *Object Points*, including screens, reports and 3GL modules, (ii) *Function Points* which measure a software project by quantifying the information

processing functionality associated with major external data or control input, output, or file types, and (iii) *Source Lines of Code*.

3) Testing Method (TESM)

There are two prominent strategy dimensions in testing techniques: function (black-box)/structured (white-box) and static/dynamic. Depending on the testing method used, different levels of coverage are achieved and consequently different level of risk remains after the completion of the phase.

4) Experience of the Testing Team (TEXP)

The effectiveness of software testing depends on the capability and the experience of the testing team; the more experienced the team is, particularly in similar with the one under development projects, the higher the probability of uncovering faults in the software. One way to categorize this factor is on the basis of time, even though it is not easy to set strict borders between the categories.

5) Software Complexity (CPLX)

The complexity of the software, like its size, is something that affects the testing process and its categorization is based on the Constructive Cost Model (COCOMO), which was developed for software cost estimation.

6) Frequency of Use (FREQ)

Frequency of use is a factor that must not be ignored, as the more a software application is used, the higher the probability of an unsatisfactory outcome to occur. Frequency of use can vary even between the various areas of the same software, as there may be, for example, some basic blocks that are being executed more often than others.

7) Time for Testing (TIME)

For every software project there are certain time limitations and deadlines in product delivery. Sometimes this fact is one of the reasons for a non-effective testing. But time shouldn't be regarded as a critical factor by itself; for instance, there is a great chance to have enough time for testing but choose an inappropriate method.

8) Available Resources (RESO)

Except the time limitations, in every case, there are certain resources available for the software development, meaning basically the financial resources and the budget that must not be exceeded.

The last seven factors affect the probability of an unsatisfactory outcome to occur, so we have the following relation:

$$P(UO) = f(SIZE,TEXP,TESM,CPLX,FREQ,TIME,RESO) \quad (2)$$

Where: P(UO) is the probability of an unsatisfactory outcome

If we define risk as the product of the probability of an unsatisfactory outcome and the loss experienced if the outcome occurs (which from relation (1) is criticality) we have:

$$R = P(UO) \times CRIT \quad (3)$$

6. Risk Management Procedure in relation with Software Testing

In order to make various testing decisions; a policy must be determined from the Development Company about the tolerable risk level of the software. This determination will be based on the available time and resources, and of course on the criticality of the project.

After making the appropriate decisions, testing is being executed, and after that an assessment of the risk level remaining is taking place, based on the abone mentioned factors.

If the risk level satisfies the requirements of the company, other quality measures are used in order to decide whether the software system is ready for delivery or not. If the risk level exceeds the tolerable one, new decisions must be made in order to do more testing. Of course, that presupposes that there is time and resources left for more testing, because if that is not the case, the company must request them from the customer and then decide the next step.

7. Conclusions

Risk Management is a critical part of any software-intensive project, and is particularly critical in large projects, projects where there is high uncertainty, or projects which are on the leading age of technology.

Our claim is that although risk must be recognized and managed throughout all the stages of the software development life cycle, it is necessary to pay great attention to the risks involved in the testing phase. And that is because during the testing phase, we are not able to eliminate the risks but only minimize them.

Attempting to determine the total risk remainimg after the end of testing, we found a set of factors that affect the probability of an unwanted outcome and together with the criticality of the software, which represents the loss experienced if the outcome occurs, we managed to define this risk.

Of course, the set of factors mentioned is not closed; there may be more factors involved, which we did not manage to find.

8. Future Work

The present work, showing that there is always some risk remaining in the software after the testing phase, sets a solid base for further research in that field. The remaining risk can be estimated by quantifying the factors that were introduced and by defining a proper mathematical relation.

Furthermore, on the basis of the present work, a model for risk management, in relation of course with software testing can be developed.

Moreover, a risk management tool can be developed whose use would definitely help to reduce the uncertainty involved in any software project. A tool like that would be extremely useful for insurance companies, whose job is to indemnify or guarantee a software user against loss caused by a software failure (software insurability).

Finally, an interesting target would be the combination of risk with other software quality criteria, like security and reliability.

ΠΕΡΙΛΗΨΗ

1. Εισαγωγή

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 διαχείρισης επικινδυνότητας καθόλη τη διάρκεια της ανάπτυξης του λογισμικού ώστε να επιτευχθεί επαρκής αναγνώριση και αποτελεσματική αντιμετώπιση των διαφόρων κινδύνων που απειλούν το σύστημα λογισμικού.

Παρόλο που οι διάφορες προσεγγίσεις που έχουν κατά καιρούς αναπτυχθεί και εφαρμόζονται, εξετάζουν το θέμα της διαχείρισης της επικινδυνότητας λογισμικού, κατά τη διάρκεια όλων των φάσεων του κύκλου ζωής ανάπτυξης λογισμικού, πρέπει να γίνει κατανοητό ότι ενώ σε όλες τις υπόλοιπες φάσεις του κύκλου ζωής ανάπτυξης λογισμικού είναι υπαρκτή η πιθανότητα όχι μόνο σημαντικής μείωσης, αλλά και εξάλειψης του μεγέθους του κινδύνου που απομένει μετά το πέρας τους, στη φάση του ελέγχου δεν ισχύει το ίδιο. Είναι δηλαδή σίγουρο ότι μετά την ολοκλήρωση του ελέγχου θα απομείνει οπωσδήποτε κάποιο ποσοστό επικινδυνότητας.

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 τη διαχείριση επικινδυνότητας. Μέσα από αυτή τη προσπάθεια, διαφαίνεται η αξία της διαχείρισης επικινδυνότητας για τον έλεγχο του λογισμικού, ενώ παράλληλα προκύπτουν ορισμένοι σημαντικοί παράγοντες που επηρεάζουν άμεσα το ποσοστό κινδύνου που απομένει μετά την ολοκλήρωση της φάσης, καθώς και μια συνολική διαδικασία διαχείρισης επικινδυνότητας του λογισμικού σε σχέση πάντα με τον έλεγχο που διεξάγεται σε αυτό.

2. Επικινδυνότητα Λογισμικού

Σύμφωνα με τον David Gluch, η επικινδυνότητα είναι ένας συνδυασμός ενός μη επιθυμητού γεγονότος ή μιας αποτυχίας και των συνεπειών αυτού του γεγονότος ή της αποτυχίας στους χειριστές, στους χρήστες ή το περιβάλλον ενός συστήματος.

Όπως και ένα οποιοδήποτε άλλο έργο, έτσι και ένα έργο λογισμικού εμπεριέχει πολλούς και ποικίλους κινδύνους. Μόνο που οι κίνδυνοι που εμπλέκονται στη τεχνολογία λογισμικού είναι πολύ περισσότερο περίπλοκοι λόγω τριών κυρίως λόγων: (i) το λογισμικό είναι ένα εξαιρετικά πολύπλοκο πνευματικό δημιούργημα και η ανάπτυξή του δεν μπορεί να ορισθεί αυστηρά ως μία μηχανική διαδικασία, (ii) είναι πολύ περισσότερο προσανατολισμένο στον άνθρωπο (human-oriented) και (iii) υπάρχει έλλειψη καλών και αποτελεσματικών μεθόδων ανάλυσης επικινδυνότητας και ελέγχου αξιοπιστίας.

3. Διαχείριση Επικινδυνότητας Λογισμικού

Αν η επικινδυνότητα θεωρηθεί ως το γενικότερο πλαίσιο που αναφέρεται σε πολιτικούς, τεχνικούς και διαχειριστικούς παράγοντες που απειλούν την επιτυχία των έργων λογισμικού, η διαχείρισή της αποτελεί τη διαδικασία αναγνώρισης και ανάλυσης αυτών των απειλών, ποσοτικοποίησης των επιπτώσεών τους και εφαρμογής σχεδίων που θα ελαττώσουν ή θα εξουδετερώσουν τις αρνητικές τους συνέπειες.

Με άλλα λόγια, δεδομένου ότι τα προβλήματα είναι αναπόφευκτα, η διαχείριση επικινδυνότητας θέτει τα ερωτήματα "Τι ακριβώς μπορεί να πάει στραβά; Και αν συμβεί κάτι τέτοιο τι μπορεί να γίνει για να αντιμετωπιστεί;" Το πρώτο ερώτημα στοχεύει στο να βρει οτιδήποτε μπορεί να αποτύχει και να προκαλέσει μια σειρά αποτυχιών. Το δεύτερο ερώτημα έχει ως στόχο την παραγωγή εναλλακτικών τρόπων δράσης σε περιπτώσεις εμφανίσεως προβλημάτων.

Όσον αφορά το κύκλο ζωής της, η διαχείριση επικινδυνότητας αντιμετωπίζεται σαν το γενικότερο πλαίσιο που περιλαμβάνει τις ακόλουθες φάσεις :

1. Ανάλυση Επικινδυνότητας (Risk Analysis) η οποία με τη σειρά της αποτελείται από :
   - ⇨ Αναγνώριση ή Προσδιορισμό Επικινδυνότητας (Risk Identification)
   - ⇨ Εκτίμηση Επικινδυνότητας (Risk Estimation or Risk Assessment)
   - ⇨ Καθορισμός Προτεραιότητας Κινδύνων (Risk Prioritization)
2. Σχεδιασμός Διαχείρισης Επικινδυνότητας (Risk Management Planning)
3. Επίλυση Επικινδυνότητας (Risk Resolution)
4. Επίβλεψη Επικινδυνότητας (Risk Monitoring)

4. Σχέση Ελέγχου Λογισμικού και Διαχείρισης Επικινδυνότητας

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 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.

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 θεωρήσουμε 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, διαπιστώνουμε 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. Ο έλεγχος δηλαδή αποτελεί στην πραγματικότητα μια διαδικασία διαχείρισης επικινδυνότητας, που στοχεύει στην εξασφάλιση εμπιστοσύνης στο λογισμικό. Άλλωστε εάν δεν υπήρχαν κίνδυνοι που να απειλούσαν το λογισμικό δεν θα υπήρχε και λόγος διεξαγωγής του ελέγχου.

5. Παράγοντες Επικινδυνότητας Λογισμικού

Σε μία προσπάθεια καθορισμού της επικινδυνότητας του λογισμικού, σε σχέση πάντα με τον έλεγχο που πραγματοποιείται σε αυτό, προκύπτει ένα σύνολο παραγόντων που την επηρεάζουν άμεσα και το οποίο παρουσιάζεται παρακάτω:

1) Κρισιμότητα Λογισμικού (CRIT)

Η κρισιμότητα που χαρακτηρίζει ένα λογισμικό μπορεί να καθοριστεί συναρτήσει της σοβαρότητας των επιπτώσεων που πιθανό να υπάρξουν, εξαιτίας της εμφάνισης κάποιων μη επιθυμητών αποτελεσμάτων. Κατά συνέπεια, μπορεί να θεωρηθεί ότι ισχύει η σχέση :

$$CRIT = f(L(UO)) \quad (1)$$

όπου : CRIT: η κρισιμότητα του λογισμικού

L(UO): η απώλεια που θα επέλθει λόγω του μη επιθυμητού αποτελέσματος (Loss if the outcome is unsatisfactory)

2) Μέγεθος Λογισμικού (SIZE)

Ο έλεγχος που διεξάγεται σε ένα λογισμικό και κατ' επέκταση και η επικινδυνότητα που απομένει μετά την ολοκλήρωσή του, εξαρτάται άμεσα από το μέγεθός που αυτό έχει. Διότι όσο μεγαλύτερο είναι το μέγεθος του λογισμικού, τόσο μεγαλύτερη είναι και η πιθανότητα εμφάνισης σφαλμάτων και κατά συνέπεια και η πιθανότητα μη εντοπισμού ενός μέρους αυτών.

Χρησιμοποιούνται τρία διαφορετικά μεγέθη μέτρησης: (i) *Object Points*, που θεωρούνται οι οθόνες, τα reports και τα συστατικά μέρη που είναι εκφρασμένα σε γλώσσες τρίτης γενεάς (3GL Components), (ii) *Unadjusted Function Points*, που μετρούν το μέγεθος ενός έργου λογισμικού, με το να ποσοτικοποιούν την λειτουργικότητα της επεξεργασίας πληροφοριών που σχετίζεται με σημαντικά εξωτερικά δεδομένα ή εισόδους ελέγχου, εξόδους ή τύπους αρχείων και (iii) *Γραμμές πηγαίου κώδικα (Source Lines of Code)*.

3) Μέθοδος Ελέγχου (TESM)

Ανάλογα με τη μέθοδο ελέγχου ή τον συνδυασμό μεθόδων που επιλέγεται να ακολουθηθεί σε κάθε περίπτωση επηρεάζεται και η αποτελεσματικότητα του ελέγχου ως προς την εύρεση σφαλμάτων στο λογισμικό και κατά συνέπεια και η επικινδυνότητα που απομένει μετά την ολοκλήρωσή του.

Ένας πρώτος διαχωρισμός των μεθόδων 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)* Ένας δεύτερος διαχωρισμός βασίζεται στο αν υπάρχει ενδιαφέρον για την εσωτερική δομή του προγράμματος οδηγώντας σε δύο βασικές κατηγορίες: *White-box testing* και *Black-box testing*.

4) Εμπειρία Ομάδας Ελέγχου (TEXP)

Η αποδοτικότητα του ελέγχου και κατ' επέκταση και το επίπεδο επικινδυνότητας που απομένει μετά την ολοκλήρωσή του, εξαρτάται και από την ικανότητα και εμπειρία

της ομάδας που διεξάγει τον έλεγχο. Η εμπειρία της ομάδας ελέγχου, μπορεί να καταταχθεί σε διάφορες βαθμίδες, με μονάδα μέτρησης το χρόνο, αν και δεν είναι εύκολο να τεθούν αυστηρά όρια μεταξύ των κατηγοριών.

5) Πολυπλοκότητα Λογισμικού (CPLX)

Η πολυπλοκότητα που έχει το λογισμικό αποτελεί έναν ακόμα παράγοντα που επηρεάζει την επικινδυνότητα που το χαρακτηρίζει. Γιατί βέβαια, όσο υψηλότερη είναι η πολυπλοκότητα του λογισμικού, τόσο μεγαλύτερη είναι και η πιθανότητα εμφάνισης μιας αποτυχίας, λόγω μη εύρεσης κάποιων σφαλμάτων κατά τη διάρκεια του ελέγχου.

Το επίπεδο πολυπλοκότητας του λογισμικού μπορεί να καθοριστεί με βάση ορισμένες κατηγορίες λειτουργιών του και πιο συγκεκριμένα βάσει των λειτουργιών ελέγχου, υπολογισμού, εξαρτώμενων από συσκευές, διαχείρισης δεδομένων και διαχείρισης διεπαφής χρήστη, βάσει της κατάταξης που ακολουθεί το Constructive Cost Model (COCOMO), το οποίο χρησιμοποιείται για την εκτίμηση κόστους ενός έργου λογισμικού.

6) Συχνότητα Χρήσης (FREQ)

Η συχνότητα χρήσης αποτελεί έναν ακόμα σημαντικό παράγοντα που επηρεάζει την επικινδυνότητα του λογισμικού. Και αυτό διότι, όσο πιο συχνά χρησιμοποιείται μια εφαρμογή ή ένα προϊόν, τόσο περισσότερο αυξάνεται και η πιθανότητα να αναδειχθεί ένα σφάλμα του λογισμικού και κατ' επέκταση να προκύψει ένα μη επιθυμητό αποτέλεσμα.

Όμως δεν υπάρχει διαφοροποίηση ως προς τη συχνότητα χρήσης μόνο ανάμεσα σε διαφορετικά συστήματα λογισμικού, αλλά και μεταξύ των τμημάτων και του ίδιου του λογισμικού. Δηλαδή, υπάρχουν περιοχές οι οποίες είναι πιθανό να χρησιμοποιούνται πιο συχνά από, όπως υπάρχουν και στον κώδικα μονοπάτια των οποίων η προσπέλαση είναι συχνότερη από κάποιων άλλων.

7) Χρόνος Ελέγχου (TIME)

Ως γνωστόν, για κάθε σύστημα λογισμικού θέτονται κάποια χρονικά όρια μέσα στα οποία πρέπει να ολοκληρωθεί η ανάπτυξή του, ώστε να παραδοθεί έγκαιρα το όποιο προϊόν ή εφαρμογή. Αυτοί ακριβώς οι περιορισμοί ως προς τα χρονικά περιθώρια επηρεάζουν φυσικά και την αποδοτικότητα του ελέγχου, και κατά συνέπεια και την επικινδυνότητα που απομένει μετά το πέρας του.

Όμως, ο χρόνος που διαρκεί ο έλεγχος δεν μπορεί να αποτελέσει απόδειξη για το πόσο πλήρης αυτός ήταν, μιας και υπάρχει η πιθανότητα διεξαγωγής εκτενούς ελέγχου, χωρίς να επιτυγχάνεται τελικά ικανοποιητική κάλυψη, λόγω κάποιων λανθασμένων επιλογών που έχουν παρθεί.

8) Διαθέσιμοι Πόροι (RESO)

Ένας άλλος, εξίσου σημαντικός με τον χρόνο παράγοντας είναι και οι πόροι που διατίθενται κάθε φορά προκειμένου να πραγματοποιηθεί η ανάπτυξη ενός συστήματος λογισμικού. έτσι, εκτός των χρονικών περιορισμών, υφίστανται και οικονομικοί περιορισμοί, οι οποίοι πολλές φορές δεν επιτρέπουν στην ομάδα ελέγχου να ενεργήσει όπως θα επιθυμούσε, με αποτέλεσμα ο έλεγχος που διεξάγεται να μην είναι ο πλέον αποτελεσματικός.

Οι τελευταίοι επτά παράγοντες που παρουσιάστηκαν πιο πάνω, μπορούμε να θεωρήσουμε ότι επηρεάζουν την πιθανότητα του να λάβει χώρα ένα ανεπιθύμητο αποτέλεσμα, δηλαδή μια αποτυχία του συστήματος λογισμικού. Υπό αυτή την προοπτική προκύπτει η σχέση:

$$P(UO) = f(SIZE, TEXP, TESM, CPLX, FREQ, TIME, RESO) \quad (2)$$

όπου: P(UO) : η πιθανότητα να συμβεί ένα μη επιθυμητό αποτέλεσμα
(probability of an unsatisfactory outcome)

Βάσει των παραγόντων που προαναφέρθηκαν, για την επικινδυνότητα που απομένει μετά την ολοκλήρωση του ελέγχου του λογισμικού θα ισχύει η παρακάτω σχέση:

$$R = P(UO) * CRIT \quad (3)$$

όπου: R : η επικινδυνότητα του λογισμικού
CRIT όπως ορίσθηκε από τη σχέση (1)
P(UO) όπως ορίσθηκε από τη σχέση (2)

6. Συνολική Διαδικασία Διαχείρισης Επικινδυνότητας σε σχέση με τον Έλεγχο Λογισμικού

Σύμφωνα με τα όσα έως τώρα αναφέρθηκαν, προκύπτει μια συνολική εικόνα της όλης διαδικασίας διαχείρισης επικινδυνότητας του λογισμικού σε συσχέτιση πάντα με την φάση του ελέγχου του.

Έτσι, πρώτ' απ' όλα και προκειμένου να παρθούν οι διάφορες αποφάσεις ελέγχου, πρέπει απαραιτήτως να έχει προηγηθεί καθορισμός μιας συγκεκριμένης πολιτικής σχετικά με το ανεκτό επίπεδο επικινδυνότητας, σύμφωνα με όσα προαναφέρθηκαν, περιλαμβάνοντας φυσικά και 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.

Κατόπιν ακολουθεί εκτίμηση του επιπέδου επικινδυνότητας που απομένει μετά την ολοκλήρωση του ελέγχου, συνυπολογίζοντας όλους τους παράγοντες που το επηρεάζουν και οι οποίοι παρουσιάστηκαν προηγουμένως. Αν το επίπεδο αυτό ικανοποιεί τις όποιες απαιτήσεις που έχουν τεθεί σύμφωνα με την πολιτική που ακολουθείται, τότε ο έλεγχος έχει ικανοποιήσει έναν από τις παραμέτρους που χαρακτηρίζουν την ποιότητα του λογισμικού, δηλαδή την επίτευξη ενός ικανοποιητικού και αποδεκτού ποσοστού επικινδυνότητας.

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.

Εάν όμως η επικινδυνότητα που απομένει ξεπερνά τα όρια που έχουν τεθεί, πρέπει να παρθούν νέες αποφάσεις ώστε να διεξαχθεί περαιτέρω έλεγχος έως ότου τελικά αυτά επιτευχθούν, υπό την προϋπόθεση βέβαια ότι υπάρχουν τα χρονικά και οικονομικά περιθώρια που θα επιτρέψουν κάτι τέτοιο. Σε περίπτωση όμως που δεν υπάρχουν είτε τα χρονικά περιθώρια είτε οι απαιτούμενοι πόροι για τη συνέχιση του ελέγχου, οι υπεύθυνοι για το έργο αιτούνται στον παραλήπτη του προϊόντος για να τους δοθεί παράταση ή να τους παραχωρηθούν επιπλέον πόροι.

Εφόσον η αίτηση αυτή γίνει δεκτή τότε η ομάδα ελέγχου μπορεί να προχωρήσει σε περαιτέρω έλεγχο, διαφορετικά το προϊόν παραδίδεται όπως έχει, εις γνώση όμως των πελατών σχετικά με το επίπεδο επικινδυνότητας που έχει απομείνει.

7. Συμπεράσματα

Η διαχείριση επικινδυνότητας αποτελεί αδιαμφισβήτητα ένα πολύ σημαντικό κομμάτι οποιουδήποτε έργου λογισμικού και είναι ιδιαίτερα κρίσιμη για μεγάλα έργα, για έργα που χαρακτηρίζονται από υψηλή αβεβαιότητα ή για έργα των οποίων μια πιθανή δυσλειτουργία θα μπορούσε να προκαλέσει μη αναστρέψιμες καταστροφές.

Αυτό όμως που πρέπει να γίνει κατανοητό είναι ότι ακόμα και αν έχουν αντιμετωπιστεί κατάλληλα οι κίνδυνοι στις προηγούμενες φάσεις, δεν υπάρχει ποτέ πιθανότητα να εκτελεστεί ένας τόσο εξαντλητικός έλεγχος ο οποίος να μπορέσει να διασφαλίσει ότι όλα θα λειτουργήσουν κατά το προσδοκώμενο και ότι τίποτα δεν θα μπορέσει να προκαλέσει μια αποτυχία του λογισμικού.

Στα πλαίσια μιας προσπάθειας προσδιορισμού του επιπέδου της μετά τον έλεγχο εναπομείνασας επικινδυνότητας, η εργασία παρουσιάζει και αναλύει ένα σύνολο

παραγόντων οι οποίοι επηρεάζουν την πιθανότητα εμφάνισης ενός μη επιθυμητού αποτελέσματος και την απώλεια που θα έχει αυτή η εμφάνιση, επηρεάζοντας κατά συνέπεια και την επικινδυνότητα που απομένει μετά την ολοκλήρωση του ελέγχου, σύμφωνα με τη σχέση που ορίζεται.

Δεν πρέπει βέβαια σε καμία περίπτωση, να θεωρηθούν δεσμευτικοί οι συγκεκριμένοι παράγοντες. Ενώ δηλαδή θεωρείται ότι είναι απαραίτητο να ληφθούν υπόψη προκειμένου να προκύψει ένα ορθό αποτέλεσμα, δεν αποκλείεται η ύπαρξη και κάποιων άλλων επιπρόσθετων παραγόντων, οι οποίοι να παίζουν έναν εξίσου σημαντικό ρόλο στον προσδιορισμό της υπολειπόμενης επικινδυνότητας.

Πρέπει επίσης να σημειωθεί ότι οι παράγοντες που προέκυψαν στην πραγματικότητα αποτελούν ένα προστάδιο για την εύρεση ενός κατάλληλου μοντέλου επικινδυνότητας σε σχέση με τον έλεγχο λογισμικού, πράγμα που αν και αποτελούσε επιδίωξη της παρούσας εργασίας, τελικά δεν επιτεύχθει.

Αντί αυτού, προτείνεται μία διαδικασία η οποία συνδυάζει τη διαχείριση επικινδυνότητας με τον έλεγχο που διεξάγεται στο λογισμικό, η εφαρμογή της οποίας προϋποθέτει την ύπαρξη συγκεκριμένης πολιτικής από την πλευρά των υπευθύνων για την ανάπτυξη του λογισμικού, έτσι ώστε να παρέχεται μια βάση για τη λήψη αποφάσεων σχετικών με το επιτρεπτό επίπεδο της μετά τον έλεγχο εναπομείνασας επικινδυνότητας.

8. Μελλοντική Έρευνα

Η παρούσα εργασία, αποδεικνύοντας το γεγονός ότι πάντα υπάρχει ένα ποσοστό επικινδυνότητας που απομένει μετά την ολοκλήρωση του ελέγχου του λογισμικού, παρέχει μια καλή βάση πάνω στην οποία μπορεί να στηριχθεί περαιτέρω έρευνα του πεδίου που προκύπτει από τη συσχέτιση των εννοιών της διαχείρισης επικινδυνότητας λογισμικού και του ελέγχου που διεξάγεται σε αυτό.

Πιο συγκεκριμένα, λαμβάνοντας κανείς τους παράγοντες που επηρεάζουν το ποσοστό της εναπομείνασας επικινδυνότητας, έτσι όπως παραθέτονται και αναλύονται στα πλαίσια της εργασίας, θα μπορούσε να κάνει μια προσπάθεια ποσοτικοποίησης τους και ανεύρεσης κατάλληλων μετρικών, ώστε να δοθεί τελικά η δυνατότητα υπολογισμού αυτού του ποσοστού μέσω μαθηματικού τύπου, ο οποίος θα είναι σύμφωνος με τη σχέση που έχει ήδη παρουσιασθεί.

Επίσης, θα μπορούσε να πραγματοποιηθεί μοντελλοποίηση της επικινδυνότητας του λογισμικού σε σχέση πάντα με τον έλεγχο, ενώ παράλληλα ανοίγονται και προοπτικές δημιουργίας ενός αυτοματοποιημένου εργαλείου, το οποίο στηριζόμενο και πάλι

στους ίδιους παράγοντες, αλλά πιθανόν και σε κάποιους επιπλέον, να παρέχει εκτίμηση του ποσοστού επικινδυνότητας που υπολείπεται μετά το πέρας του ελέγχου.

Η δημιουργία ενός τέτοιου εργαλείου, σίγουρα θα προκαλούσε γενικότερο ενδιαφέρον, και ιδιαίτερα αυτό των διαφόρων επιχειρήσεων και οργανισμών που ασχολούνται με την ανάπτυξη λογισμικού, αφού η χρήση του θα μπορούσε να αποτελέσει ένα μέσο μείωσης της αβεβαιότητας που αναπόφευκτα ενυπάρχει σε ένα οποιοδήποτε έργο λογισμικού.

Μια άλλη περιοχή στην οποία θα μπορούσε να έχει εφαρμογή με επιτυχία κάτι τέτοιο, είναι αυτή των εταιριών που παρέχουν ασφάλιση που σχετίζεται με λογισμικό, μιας και σίγουρα θα ελάττωνε σημαντικά το ποσοστό του ρίσκου που διατίθεται να πάρει ο ασφαλιστής προκειμένου να εξασφαλίσει ένα χρήστη λογισμικού από απώλειες που πιθανόν να συμβούν εξαιτίας μιας αποτυχίας του συστήματος (software insurability).

Τέλος, αυτό που είναι σημαντικό να επιτευχθεί είναι ένας κατάλληλος ποιοτικός, αλλά κυρίως ποσοτικός συνδυασμός της επικινδυνότητας του λογισμικού με τα διάφορα κριτήρια ποιότητας που το χαρακτηρίζουν, όπως είναι για παράδειγμα η αξιοπιστία ή η ασφάλεια, ώστε να παρέχεται στον ενδιαφερόμενο μια πληρέστερη και ακριβέστερη εικόνα του.

ΕΙΣΑΓΩΓΗ

"Life without adventure is likely to be unsatisfying, but a life which adventure is allowed to take whatever form it will, is likely to be short"

Bertrand Russell

Οι αποτυχίες λογισμικού είναι ο εφιάλτης της εποχής της πληροφορίας. Τεράστιες ποσότητες πόρων ξοδεύονται σε όλες τις σχετικές με τη τεχνολογία λογισμικού βιομηχανίες με σκοπό την καταπολέμηση αυτού του εφιάλτη. 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 διαχείρισης επικινδυνότητας καθόλη τη διάρκεια της ανάπτυξης του λογισμικού.

Άλλωστε η επικινδυνότητα και η διαχείριση αυτής, έχουν αναδειχθεί σε θέματα μεγάλου ενδιαφέροντος, ιδιαίτερα μάλιστα κατά τη διάρκεια των τελευταίων ετών, όπου έχουν προκύψει νέες προοπτικές και ευκαιρίες, ενώ παράλληλα έχει οξυνθεί και ο ανταγωνισμός στη βιομηχανία λογισμικού.

Οι διάφορες προσεγγίσεις που έχουν κατά καιρούς αναπτυχθεί και εφαρμόζονται, εξετάζουν το θέμα της διαχείρισης της επικινδυνότητας λογισμικού, κατά τη διάρκεια όλων των φάσεων του κύκλου ζωής ανάπτυξης λογισμικού, χωρίς καμία ιδιαίτερη εστίαση σε κάποια συγκεκριμένη φάση.

Αυτή η άποψη περί εφαρμογής της διαχείρισης επικινδυνότητας από τη στιγμή που ξεκινά να αναπτύσσεται το λογισμικό, έως τη στιγμή που παραδίδεται και συντηρείται, είναι απόλυτα ορθή και δικαιολογημένη. Διότι σε διαφορετική περίπτωση δεν θα μπορούσε να επιτευχθεί επαρκής αναγνώριση και αποτελεσματική αντιμετώπιση των διαφόρων κινδύνων που απειλούν το σύστημα λογισμικού.

Αυτό όμως που πρέπει να γίνει κατανοητό είναι το γεγονός ότι ενώ σε όλες τις υπόλοιπες φάσεις του κύκλου ζωής ανάπτυξης λογισμικού είναι υπαρκτή η πιθανότητα όχι μόνο σημαντικής μείωσης, αλλά και εξάλειψης του μεγέθους του

κινδύνου που απομένει μετά το πέρας τους, στη φάση του ελέγχου δεν ισχύει το ίδιο. Διότι είναι σίγουρο ότι μετά την ολοκλήρωση του ελέγχου θα απομείνει οπωσδήποτε κάποιο ποσοστό επικινδυνότητας.

Άλλωστε ο έλεγχος του λογισμικού, αποτελεί ούτως ή άλλως μια εξαιρετικά σημαντική διαδικασία, για την οποία σπαταλάται 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 απαραιτήτως οι όποιες αποφάσεις ελέγχου να λαμβάνονται και βάσει των εμπλεκομένων κινδύνων.

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 διερευνήσει τη συσχέτισή της με τη διαχείριση επικινδυνότητας.

Μέσα από αυτή τη προσπάθεια, διαφαίνεται η αξία της διαχείρισης επικινδυνότητας για τον έλεγχο του λογισμικού, ενώ παράλληλα προκύπτουν ορισμένοι σημαντικοί παράγοντες που επηρεάζουν άμεσα το ποσοστό κινδύνου που απομένει μετά την ολοκλήρωση της φάσης, καθώς και μια συνολική διαδικασία διαχείρισης επικινδυνότητας του λογισμικού σε σχέση πάντα με τον έλεγχο που διεξάγεται σε αυτό.

ΚΕΦΑΛΑΙΟ 1

"If you don't attack the risks,
they will actively attack you"

Tom Gitb,1998

1.1 Εισαγωγικά

Το πρώτο αυτό κεφάλαιο έχει έναν κυρίως εισαγωγικό ρόλο, πραγματοποιώντας μια συνοπτική, αλλά απαραίτητη, παρουσίαση των εννοιών της επικινδυνότητας και της διαχείρισης αυτής, υπό μια γενικότερη προοπτική, αλλά κυρίως σε σχέση με το λογισμικό.

Πιο συγκεκριμένα, παρατίθενται επιλεκτικά κάποιοι ορισμοί των εννοιών, όπως αυτοί υπάρχουν στη βιβλιογραφία και γίνεται μια προσπάθεια εξέτασης ορισμένων θεμάτων που σχετίζονται άμεσα με αυτές, όπως είναι η αναγνώριση διαφόρων τύπων κινδύνων, η σχέση της επικινδυνότητας με την ασφάλεια του λογισμικού, αλλά και η ύπαρξη ποικίλων προσεγγίσεων που αφορούν την εφαρμογή της διαχείρισης επικινδυνότητας στο κύκλο ζωής ανάπτυξης του λογισμικού.

1.2 Επικινδυνότητα Λογισμικού (Software Risk)

1.2.1 Ορισμός Επικινδυνότητας Λογισμικού

Ο πλέον διαδεδομένος ορισμός της επικινδυνότητας (risk) είναι αυτός που την θεωρεί ως την πιθανότητα να συμβούν ανεπιθύμητα γεγονότα.

Σύμφωνα με τον David Gluch, η επικινδυνότητα είναι ένας συνδυασμός ενός μη επιθυμητού γεγονότος ή μιας αποτυχίας και των συνεπειών αυτού του γεγονότος ή της αποτυχίας στους χειριστές, στους χρήστες ή το περιβάλλον ενός συστήματος.

Κατά τον D. Phillips, αποτελεί ένα πρόβλημα το οποίο μπορεί να μη συμβεί ποτέ, αλλά αν συμβεί μπορεί να βλάψει ή και να καταστρέψει ένα σύστημα ή γενικότερα ένα έργο [Phil98]. Η επικινδυνότητα δηλαδή περιλαμβάνει παράγοντες, τόσο τεχνικούς όσο και διαχειριστικούς, οι οποίοι μπορούν να αποτελέσουν απειλή για την επιτυχία του έργου [Bala98].

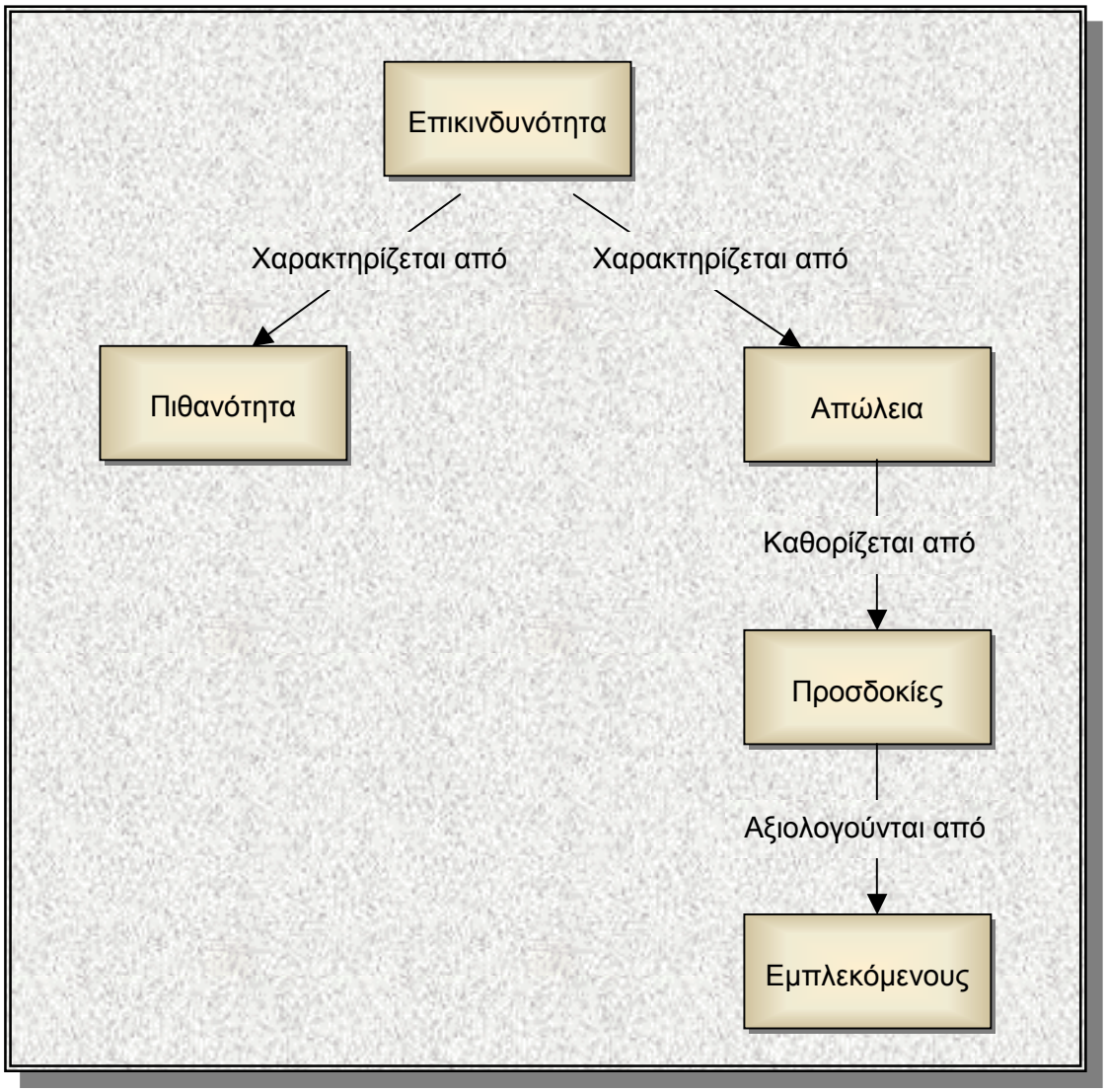
Εάν αναφερόμαστε πιο συγκεκριμένα σε κάποιο σύστημα λογισμικού, έχει ορθώς επικρατήσει η θεώρηση ότι η επικινδυνότητα μπορεί ακριβέστερα να εκφραστεί ως συνάρτηση της φύσης και του βαθμού των ευπαθειών (vulnerabilities) που πιθανόν να έχει το λογισμικό, της φύσης και της πιθανότητας εμφάνισης απειλών (threats) κατά του συστήματος λογισμικού και της φύσης και έντασης των επιπτώσεων (impacts) που αυτές θα έχουν εάν τελικά πραγματοποιηθούν.

Αν τα παραπάνω εκφραστούν με έναν πιο τυπικό τρόπο, προκύπτει η σχέση:

R = f (T, V, I)
όπου: { R = Risk και T = Threats, V = Vulnerabilities, I = Impacts}

Η σχέση αυτή υποδηλώνει την ύπαρξη τόσο εξωτερικών, όσο και εσωτερικών κινδύνων στο σύστημα λογισμικού, με την εμφάνιση των μεν να οφείλεται πολλές φορές στην ύπαρξη των δε. Γιατί βέβαια κάποιες από τις απειλές δεν θα μπορούσαν ποτέ να λάβουν χώρα εάν δεν υπήρχαν αδυναμίες του συστήματος που να το επέτρεπαν.

Το κοινό στοιχείο όλων των παραπάνω ορισμών της επικινδυνότητας είναι ότι περιλαμβάνουν δύο βασικά χαρακτηριστικά: την πιθανότητα και την απώλεια. Το ίδιο συμβαίνει και με τον ορισμό που υποστηρίζει η μέθοδος Riskit, σύμφωνα με την οποία ο ορισμός της απώλειας εξαρτάται από τις εκάστοτε προσδοκίες, οι οποίες με τη σειρά τους εξαρτώνται από τους εμπλεκόμενους στο έργο [Kont97]. Αυτή ακριβώς η προσέγγιση διαφαίνεται στο σχήμα 1.1.



Σχήμα 1.1: *Ορισμός της Επικινδυνότητας στη μέθοδο Riskit*

1.2.2 Τύποι Κινδύνων Έργων Λογισμικού

Όπως και ένα οποιοδήποτε άλλο έργο, έτσι και ένα έργο λογισμικού εμπεριέχει πολλούς και ποικίλους κινδύνους. Μόνο που οι κίνδυνοι που εμπλέκονται στη τεχνολογία λογισμικού είναι πολύ περισσότερο περίπλοκοι λόγω τριών κυρίως λόγων: (i) το λογισμικό είναι ένα εξαιρετικά πολύπλοκο πνευματικό δημιούργημα και η ανάπτυξή του δεν μπορεί να ορισθεί αυστηρά ως μία μηχανική διαδικασία, (ii) είναι πολύ περισσότερο προσανατολισμένο στον άνθρωπο (human-oriented) και (iii) υπάρχει έλλειψη καλών και αποτελεσματικών μεθόδων ανάλυσης επικινδυνότητας και ελέγχου αξιοπιστίας [Ashr97].

Είναι λοιπόν πολύ σημαντικό να υπάρχει δυνατότητα μιας κάποιας εκ των προτέρων γνώσης του είδους των κινδύνων που πιθανό να απειλήσουν την ανάπτυξη ενός έργου

λογισμικού, έτσι ώστε να μπορέσει να υπάρξει ταχύτερη και ευκολότερη αναγνώρισή τους, αλλά και αποτελεσματικότερη αντιμετώπισή τους, εάν τελικά εμφανιστούν.

Από διάφορες έρευνες που έχουν διεξαχθεί, έχουν αναγνωρισθεί και καταγραφεί κάποιοι τύποι κινδύνων οι οποίοι συναντώνται αρκετά συχνά σε έργα λογισμικού. Πιο συγκεκριμένα, οι πιο συνηθισμένοι κίνδυνοι είναι οι ακόλουθοι [Bala98], [Wieg98] :

- ✓ Ανεπάρκεια προσωπικού
- ✓ Ακατάλληλη εκπαίδευση
- ✓ Ανεπαρκής κατανόηση μεθόδων, εργαλείων και τεχνικών
- ✓ Μη ρεαλιστικά σχέδια και προϋπολογισμοί
- ✓ Σταθερότητα εξωτερικού ή έτοιμου λογισμικού
- ✓ Ανεπαρκείς ή λανθασμένες απαιτήσεις
- ✓ Μη ικανοποιητικές διεπαφές χρήστη
- ✓ Αρχιτεκτονική, επίδοση, ποιότητα
- ✓ Αλλαγές απαιτήσεων
- ✓ Legacy software
- ✓ Εξωτερικά εκτελούμενες εργασίες

Κάποιοι άλλοι συχνά εμφανιζόμενοι σε έργα λογισμικού κίνδυνοι είναι :

- ✓ Περιορισμοί πλατφόρμας ανάπτυξης
- ✓ Νέες τεχνολογίες ή μεθοδολογίες ανάπτυξης
- ✓ Περιορισμένοι πόροι
- ✓ Προβλήματα γλώσσας/επικοινωνίας
- ✓ Συγκρούσεις ανάμεσα στο προσωπικό
- ✓ Απασχόληση ομάδας ανάπτυξης και σε άλλες δραστηριότητες
- ✓ Μη ικανοποίηση προτύπων

Σύμφωνα με μία άλλη προσέγγιση, η επικινδυνότητα λογισμικού μπορεί να διαχωριστεί σε κάποιες κατηγορίες ανάλογα με την περιοχή που επηρεάζεται όταν το λογισμικό δεν λειτουργεί κατά το αναμενόμενο, έτσι ώστε να υπάρξει και καλύτερη κατανόηση της φύσης των κινδύνων [Hall98].

Πιο συγκεκριμένα προκύπτουν τρεις βασικές κατηγορίες :

1) Software project risk:

Επηρεάζει λειτουργικές και οργανωτικές κυρίως παραμέτρους της ανάπτυξης λογισμικού. Σχετίζεται με περιορισμούς πόρων, εξωτερικές διεπαφές, σχέσεις

πελατών ή περιορισμούς συμβολαίων, ενώ η πιο σημαντική παράμετρος που συνήθως αναφέρεται από διάφορες περιπτώσεις έργων, είναι η χρηματοδότηση.

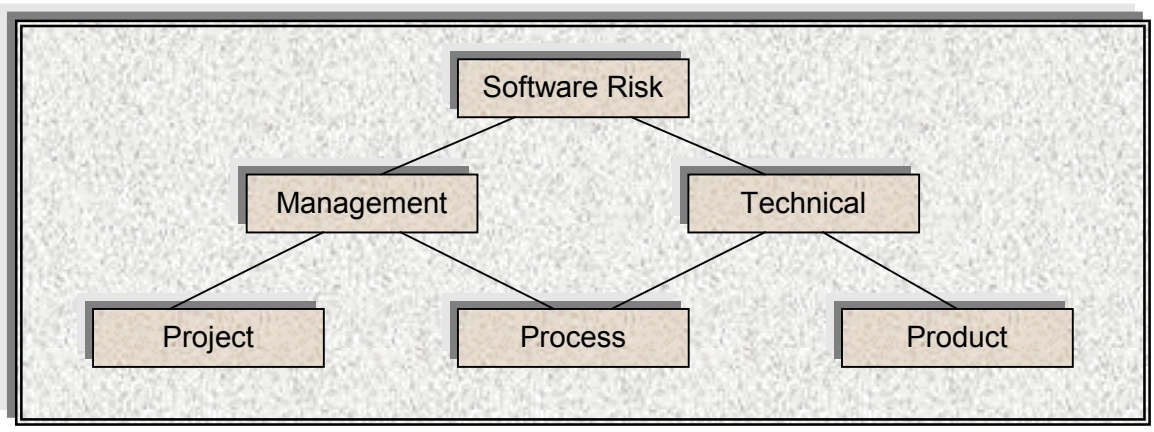
2) Software process risk:

Επηρεάζει τεχνικές και διοικητικές διαδικασίες. Στις διοικητικές διαδικασίες, η επικινδυνότητα μπορεί να εντοπισθεί σε δραστηριότητες όπως ο σχεδιασμός, η παρακολούθηση της πορείας του έργου, η εξασφάλιση ποιότητας και το configuration management. Στις τεχνικές διαδικασίες μπορεί να εντοπιστεί σε δραστηριότητες όπως η ανάλυση απαιτήσεων, ο σχεδιασμός, η κωδικοποίηση και ο έλεγχος.

3) Software product risk:

Αυτή η κατηγορία επηρεάζει χαρακτηριστικά ενδιάμεσων και τελικών προϊόντων. Εδώ ο κίνδυνος μπορεί να σχετίζεται με τη σταθερότητα των απαιτήσεων, την εκτέλεση του σχεδίου, τη πολυπλοκότητα του κώδικα και τις προδιαγραφές ελέγχου.

Τα παραπάνω μπορούν να παρουσιαστούν σχηματικά ως ακολούθως :



Σχήμα 1.2 : *Κατηγοριοποίηση Επικινδυνότητας Λογισμικού*

Βέβαια, δεν πρέπει να λησμονείται το γεγονός ότι κάθε σύστημα λογισμικού είναι μοναδικό και χαρακτηρίζεται από το δικό του ιδιαίτερο σύνολο κινδύνων, με αποτέλεσμα η αναγνώριση των κινδύνων και πολύ περισσότερο η κατηγοριοποίησή τους, να αποτελεί ένα αρκετά δύσκολο έργο.

1.3 Σχέση Ασφάλειας και Επικινδυνότητας Λογισμικού

Προβλήματα ασφάλειας (safety) σίγουρα υπήρχαν από πολύ παλιά ιδιαίτερα σε τομείς όπως η αεροναυπηγική και η άμυνα, αλλά παρουσιάστηκαν ακόμα πιο έντονα από τη στιγμή που χρησιμοποιήθηκαν συστήματα λογισμικού σε κρίσιμες (safety-critical) εφαρμογές πολλών και διαφορετικών μεταξύ τους περιοχών, όπως η ιατρική, η μεταφορές, η βιομηχανία ενέργειας, η βιομηχανία κατασκευών, κ.α.

Έτσι μαζί με τα πλεονεκτήματα που προσφέρονταν, όπως η πολυμέρεια, η δύναμη, η επίδοση και η αποδοτικότητα προστέθηκαν και διάφοροι κίνδυνοι που απέρρεαν από την ανικανότητα προσφοράς λογισμικού απαλλαγμένου από λάθη. Κατά συνέπεια το πρόβλημα είναι να αντιπαρατίθενται τα οφέλη που προσφέρονται, με τους κινδύνους που εισάγονται. Ακόμα και όπου οι τα υπολογιστικά συστήματα μπορούν να βελτιώσουν την ασφάλεια, δεν είναι απόλυτα σίγουρο ότι θα το κάνουν, μιας και οι τεχνολογικές βελτιώσεις συχνά επιτρέπουν την εμφάνιση μεγαλύτερων κινδύνων.

Η ασφάλεια του λογισμικού, που ως γνωστόν, αποτελεί ένα από τα κριτήρια ποιότητάς του, μπορεί να ορισθεί ως "απελευθέρωση από εκείνες τις συνθήκες οι οποίες μπορούν να προκαλέσουν θάνατο, καταστροφή, επαγγελματική ζημία, ή οποιαδήποτε άλλη βλάβη ή απώλεια εξοπλισμού ή περιουσίας" [Leves87].

Από τον παραπάνω ορισμό γίνεται φανερό ότι η έννοια της ασφάλειας, είναι άμεσα συνδεόμενη με την έννοια της επικινδυνότητας που χαρακτηρίζει το λογισμικό, συμπέρασμα το οποίο προκύπτει και από το γεγονός ότι η ασφάλεια είναι κάτι πολύ σχετικό και είναι σίγουρο πως τίποτα δεν μπορεί να είναι εντελώς ασφαλές κάτω από οποιεσδήποτε συνθήκες. Έτσι, στην πραγματικότητα αυτό που πραγματοποιείται είναι μια προσπάθεια παροχής ενός αποδεκτού επιπέδου επικινδυνότητας.

1.4 Διαχείριση Επικινδυνότητας Λογισμικού (Software Risk Management)

1.4.1 Η Έννοια της Διαχείρισης Επικινδυνότητας

Είναι πλέον κοινώς αποδεκτό το γεγονός ότι η αβεβαιότητα αποτελεί ένα φυσιολογικό και αναπόφευκτο χαρακτηριστικό των περισσότερων έργων λογισμικού. Γι' αυτόν ακριβώς το λόγο η ικανότητα διαχείρισης της αποτελεί κρίσιμο παράγοντα για την επιτυχία τους. Ιδιαίτερα μάλιστα σε μια εποχή που πρέπει να αντιμετωπιστούν ελλείψεις σε πόρους, μεγάλα τεχνολογικά άλματα και αυξημένες απαιτήσεις για πολύπλοκα συστήματα σε ένα ταχέως εξελισσόμενο περιβάλλον.

Δεν είναι άλλωστε λίγες οι περιπτώσεις όπου αποτυχίες λογισμικού, έχουν πλήξει, αν όχι καταστρέψει, πολλά φιλόδοξα προγράμματα και έργα. Έχουν επίσης επηρεάσει πολυάριθμες κοινωνικές και επιστημονικές εφαρμογές και έχουν προκαλέσει σοβαρές αρνητικές συνέπειες σε πολλές επιχειρήσεις και οργανισμούς.

Πολλές από αυτές τις αποτυχίες θα μπορούσαν να έχουν αποφευχθεί, ή έστω σημαντικά μειωθεί, εάν κατά την διάρκεια του κύκλου ζωής ανάπτυξης του λογισμικού είχε πραγματοποιηθεί σωστή διαχείριση επικινδυνότητας, ώστε να εντοπιστούν και να χειριστούν ανάλογα περιοχές του λογισμικού οι οποίες παρουσίαζαν πραγματικά υψηλή επικινδυνότητα.

Αν λοιπόν η επικινδυνότητα θεωρηθεί ως το γενικότερο πλαίσιο που αναφέρεται σε πολιτικούς, τεχνικούς και διαχειριστικούς παράγοντες που απειλούν την επιτυχία των έργων λογισμικού, η διαχείρισή της αποτελεί τη διαδικασία αναγνώρισης και ανάλυσης αυτών των απειλών, ποσοτικοποίησης των επιπτώσεών τους και εφαρμογής σχεδίων που θα ελαττώσουν ή θα εξουδετερώσουν τις αρνητικές τους συνέπειες.

Με άλλα λόγια, δεδομένου ότι τα προβλήματα είναι αναπόφευκτα, η διαχείριση επικινδυνότητας θέτει τα ερωτήματα "Τι ακριβώς μπορεί να πάει στραβά; Και αν συμβεί κάτι τέτοιο τι μπορεί να γίνει για να αντιμετωπιστεί;" Το πρώτο ερώτημα στοχεύει στο να βρει οτιδήποτε μπορεί να αποτύχει και να προκαλέσει μια σειρά αποτυχιών. Το δεύτερο ερώτημα έχει ως στόχο την παραγωγή εναλλακτικών τρόπων δράσης σε περιπτώσεις εμφανίσεως προβλημάτων.

Είναι επίσης σημαντικό να αναφερθεί και το γεγονός ότι η έννοια της διαχείρισης επικινδυνότητας εμπεριέχει αναμφισβήτητα τα λεγόμενα τρία P (3Ps), δηλαδή ανθρώπους, διαδικασία και προϊόν (people, process and product), καθότι η πιθανότητα εμφάνισης προβλημάτων εξαρτάται άμεσα από τις σχέσεις που υπάρχουν ανάμεσα στα τρία αυτά στοιχεία. Αν για παράδειγμα, κάποιοι άνθρωποι που δεν γνωρίζουν το προϊόν περιορίζονται από μία απλή διαδικασία, τότε είναι πολύ πιθανό να λάβει χώρα κάποιο πρόβλημα.

Αποτελεί τέλος και μία διαδικασία μετάφρασης ποιοτικών ιδεών σε ποσοτικούς όρους, αφού για παράδειγμα μετατρέπει μία δήλωση όπως : "Με απασχολεί η διεπαφή μεταξύ του δικού μας λογισμικού και του πακέτου που αγοράζουμε" σε "Υπάρχει μια πιθανότητα 30% να αποτύχει η διεπαφή και σε κλίμακα από 1 έως 10, η ζημιά για το έργο να είναι 8, με το 10 να αντιστοιχεί στην πιο καταστροφική επίπτωση".

1.4.2 Επίπεδα Διαχείρισης Επικινδυνότητας

Υπάρχουν διαφορετικοί τρόποι με τους οποίους μπορεί να διεξαχθεί η διαδικασία της διαχείρισης επικινδυνότητας ενός έργου λογισμικού, οι οποίοι διαφοροποιούνται ανάλογα με το πότε και το πώς πραγματοποιείται η αναγνώριση και αντιμετώπιση των κινδύνων.

Αυτοί ακριβώς οι τρόποι χαρακτηρίζονται ως επίπεδα διαχείρισης επικινδυνότητας και έχουν ως εξής [Bala98] :

- **Διαχείριση Κρίσης (Crisis Management) :** Αυτός ο τρόπος διαχείρισης μπορεί να χαρακτηρισθεί ως "μάχη με τη φωτιά", καθότι η προσπάθεια αντιμετώπισης των κινδύνων γίνεται αφού πρώτα έχουν γίνει προβλήματα.

- **Αντιμετώπιση κατά την Αποτυχία (Fix on Failure) :** Αυτή η περίπτωση περιλαμβάνει ανίχνευση και άμεση αντίδραση, αφότου όμως έχουν λάβει χώρα οι κίνδυνοι.

- **Μετριασμός Κινδύνων (Risk Mitigation) :** Εδώ σχεδιάζεται η κάλυψη των κινδύνων, αλλά δεν γίνεται καμία προσπάθεια να προληφθούν.

- **Πρόληψη (Prevention) :** Σύμφωνα με αυτή την προσέγγιση σχεδιάζεται η αναγνώριση και πρόληψη των κινδύνων, ώστε να αποτραπεί κατά το δυνατόν η μετατροπή τους σε προβλήματα που θα επιφέρουν αρνητικά αποτελέσματα στο έργο λογισμικού.

- **Εξάλειψη Κύριων Αιτιών (Elimination of Root Causes) :** Αποτελεί τον πλέον αποτελεσματικό, αλλά συνάμα και δύσκολο τρόπο διαχείρισης της επικινδυνότητας, μιας και γίνεται προσπάθεια αναγνώρισης και εξάλειψης των παραγόντων που επιτρέπουν στους κινδύνους να συμβούν.

Τα παραπάνω επίπεδα, τα οποία όπως είναι φυσικό διαφέρουν ως προς την αποτελεσματικότητά τους, εφαρμόζονται κάθε φορά βάσει επιλογών οι οποίες λαμβάνονται από τους εμπλεκόμενους στο έργο ανάπτυξης του λογισμικού και οι οποίες με την σειρά τους εξαρτώνται από διάφορους παράγοντες που το επηρεάζουν.

1.4.3 Κύκλος Ζωής Διαχείρισης Επικινδυνότητας

Είναι σίγουρο πως δεν αποτελεί καθόλου εύκολη υπόθεση ούτε ο ακριβής προσδιορισμός της έννοιας της διαχείρισης επικινδυνότητας, αλλά ούτε και των σταδίων που αυτή περιλαμβάνει, καθότι τα διάφορα μοντέλα που έχουν κατά καιρούς αναπτυχθεί και χρησιμοποιούνται παρουσιάζουν διαφορετικές προσεγγίσεις ως προς τα βήματα και τις τεχνικές που πρέπει να ακολουθούνται, προκειμένου να διεξάγεται με επιτυχία η όλη διαδικασία.

Επίσης στην βιβλιογραφία υπάρχει μία σύγχυση ανάμεσα στις έννοιες της ανάλυσης και της διαχείρισης της επικινδυνότητας, αλλά επικρατεί η άποψη που θεωρεί την ανάλυση επικινδυνότητας ως ένα κομμάτι της διαχείρισης.

Πιο συγκεκριμένα η διαχείριση επικινδυνότητας αντιμετωπίζεται σαν το γενικότερο πλαίσιο που περιλαμβάνει τις ακόλουθες φάσεις :

2. Ανάλυση Επικινδυνότητας (Risk Analysis) η οποία με τη σειρά της αποτελείται από :

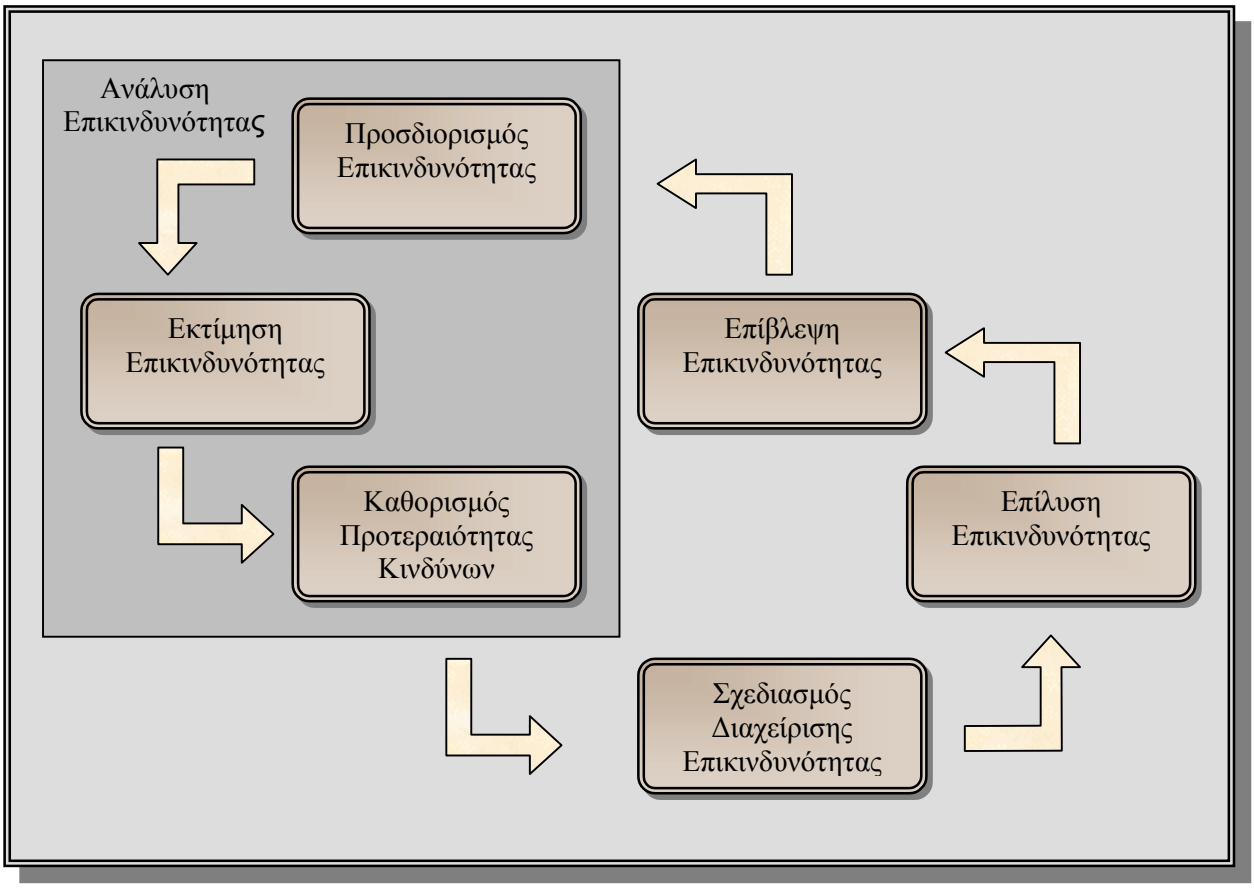
   - ⇨ Αναγνώριση ή Προσδιορισμό Επικινδυνότητας (Risk Identification)
   - ⇨ Εκτίμηση Επικινδυνότητας (Risk Estimation or Risk Assessment)
   - ⇨ Καθορισμός Προτεραιότητας Κινδύνων (Risk Prioritization)

5. Σχεδιασμός Διαχείρισης Επικινδυνότητας (Risk Management Planning)

6. Επίλυση Επικινδυνότητας (Risk Resolution)

7. Επίβλεψη Επικινδυνότητας (Risk Monitoring)

Σύμφωνα με την παραπάνω θεώρηση ο κύκλος ζωής της διαχείρισης επικινδυνότητας, μπορεί να αναπαρασταθεί σχηματικά ως εξής :



Σχήμα 1.3 : *Ο Κύκλος Ζωής Διαχείρισης Επικινδυνότητας*

Αναλυτικότερα οι φάσεις που προαναφέρθηκαν έχουν ως ακολούθως :

1.4.3.1 Ανάλυση Επικινδυνότητας

Η ανάλυση επικινδυνότητας αποτελεί την πρώτη φάση της όλης διαδικασίας διαχείρισης και αποτελείται, όπως αναφέρθηκε και προηγουμένως, από τρία στάδια: τον προσδιορισμό, την εκτίμηση και την αποτίμηση της επικινδυνότητας. Πιο συγκεκριμένα :

- *Προσδιορισμός Επικινδυνότητας*

Προκειμένου να διαχειριστούν οι κίνδυνοι που απειλούν ένα έργο λογισμικού, πρέπει πρώτ' απ' όλα να αναγνωρισθούν. Σε αυτή τη φάση πραγματοποιείται μια προσπάθεια ανεύρεσης πιθανών κινδύνων χρησιμοποιώντας ποικίλες μεθόδους, όπως καταλόγους κινδύνων, ερωτηματολόγια, έχοντας ως βάση βέβαια και προηγούμενη εμπειρία. Ορισμένοι από τους κινδύνους είναι πολύ πιθανό να είναι κοινοί σε πολλά έργα λογισμικού ενώ κάποιοι άλλοι να προκύπτουν μόνο σε μεμονωμένες περιπτώσεις.

Η αναγνώριση των κινδύνων περιλαμβάνει εύρεση της περιοχής την οποία επηρεάζουν, αλλά και των αιτιών που τους προκαλούν. Για την ορθή διεξαγωγή αυτής ακριβώς της διαδικασίας απαιτείται να ληφθούν υπόψη στοιχεία σχετικά τόσο με το προϊόν, όσο και με τον παράγοντα άνθρωπο αλλά και την διαδικασία.

Όπως είναι φυσικό, ο προσδιορισμός των κινδύνων αποτελεί μια υποκειμενική διαδικασία και τα αποτελέσματά της εξαρτώνται από την προοπτική και την εμπειρία αυτού που την εκτελεί. Γι' αυτόν ακριβώς το λόγο, και προκειμένου να πραγματοποιηθεί αποδοτικότερη διεξαγωγή της διαδικασίας, να υπάρχει συνεργασία ατόμων που εμπλέκονται γενικότερα στο έργο ανάπτυξης, έτσι ώστε να θέτονται προς συζήτηση περισσότερες και διαφορετικές ανησυχίες και ερωτήσεις. Είναι μάλιστα συνετό, κάποιες φορές να ζητείται και η βοήθεια εξωτερικών συμβούλων και γενικότερα ανθρώπων που διαθέτουν εμπειρία σε παρόμοιες καταστάσεις [Phil98].

- *Εκτίμηση Επικινδυνότητας*

Αφού έχουν αναγνωρισθεί οι κίνδυνοι που είναι πιθανό να απειλήσουν το έργο, πρέπει να πραγματοποιηθεί μία εκτίμηση αυτών, ώστε να προσδιορισθεί η πιθανότητα εμφάνισής τους, καθώς και η σοβαρότητα των επιπτώσεων που θα έχουν εάν τελικά λάβουν χώρα.

Ο προσδιορισμός της πιθανότητας εμφάνισης των κινδύνων, είναι σίγουρο πως δεν αποτελεί καθόλου εύκολη υπόθεση και είναι σχεδόν ακατόρθωτο να δοθούν ακριβή ποσοστά. Γι' αυτό το λόγο, συνηθίζεται να χρησιμοποιείται κάποιο είδος κλίμακας που να κατατάσσει για παράδειγμα την πιθανότητα σε χαμηλή/μέτρια/υψηλή.

Ταυτόχρονα με την εκτίμηση της πιθανότητας, κρίνεται και η σοβαρότητα των επιπτώσεων των κινδύνων, επιπτώσεις στον προϋπολογισμό, το σχέδιο ανάπτυξης, την επίδοση, την ικανοποίηση του πελάτη και σε πολλές άλλες περιοχές. Μια τέτοια εκτίμηση βασίζεται στη φύση των κινδύνων, αλλά και στο πότε αυτοί μπορεί να συμβούν. Γιατί βέβαια το να εμφανιστεί ένας κίνδυνος νωρίς στον κύκλο ζωής ανάπτυξης του λογισμικού, έχει διαφορετική επίπτωση για το έργο από ότι θα είχε αν συνέβαινε αργότερα.

Από τον συνδυασμό της πιθανότητας εμφάνισης των κινδύνων και το μέγεθος της απώλειας που αυτοί θα έχουν εάν εμφανιστούν, μπορεί να προκύψει η συνολική έκθεση του έργου στο κίνδυνο (risk exposure).

- *Καθορισμός Προτεραιότητας Κινδύνων*

Ύστερα από την εκτίμηση των κινδύνων, γίνεται προσπάθεια να τους δοθεί ένας είδος προτεραιότητας, βάσει της πιθανότητας εμφάνισής τους καθώς και των ενδεχόμενων μη επιθυμητών αποτελεσμάτων τους.

Έτσι για παράδειγμα αναγνωρίζονται ως πιο σημαντικοί κίνδυνοι όσοι μπορούν να προκαλέσουν σοβαρές αρνητικές επιπτώσεις έχοντας μέτρια έως και υψηλή πιθανότητα να συμβούν, καθώς και όσοι μπορούν να προκαλέσουν μέτριας ή και χαμηλής κλίμακας επιπτώσεις έχοντας όμως υψηλή πιθανότητα εμφάνισης. Πιο συγκεκριμένα, μπορεί να δημιουργηθεί ένας πίνακας που να φανερώνει τις προτεραιότητες αυτές και ο οποίος παρουσιάζεται σε επόμενη παράγραφο στα πλαίσια της SRE μεθόδου (Πίνακας 1.1).

Κατά αυτό το τρόπο πραγματοποιείται ένα είδος κατηγοριοποίησης των κινδύνων, ώστε να υπάρξει εστίαση κατ' αρχάς στους σημαντικότερους εξ αυτών και να ακολουθηθεί στη συνέχεια η καταλληλότερη δυνατή στρατηγική αντιμετώπισής τους.

1.4.3.2 Σχεδιασμός Διαχείρισης Επικινδυνότητας

Μετά την ολοκλήρωση της φάσης της ανάλυσης και αφού έχουν δοθεί οι όποιες προτεραιότητες στους κινδύνους που έχουν εντοπιστεί και εκτιμηθεί, ακολουθεί η φάση του σχεδιασμού της διαχείρισης.

Στόχος εδώ είναι η εύρεση λύσεων και στρατηγικών που χρειάζεται να ακολουθηθούν προκειμένου να αντιμετωπιστεί κάθε ένας από τους κινδύνους, με σεβασμό βέβαια στη σειρά προτεραιότητας που αυτοί διαθέτουν.

Επίσης πραγματοποιείται ανάλυση του κόστους των προτεινόμενων λύσεων, για να εκτιμηθεί το κατά πόσο αυτές μπορούν να δικαιολογηθούν από τον υπάρχοντα προϋπολογισμό. Γιατί βέβαια, δεν αρκεί να είναι ικανοποιητικές και αποτελεσματικές οι λύσεις που ανευρίσκονται, αλλά και να βρίσκονται μέσα στα επιτρεπτά οικονομικά πλαίσια.

Τέλος, σχεδιάζεται το σημείο του έργου στο οποίο θα εφαρμοστεί η όποια λύση ή στρατηγική και καθορίζεται ο τρόπος με τον οποίο θα μπορέσει να γίνει η απαιτουμένη ένταξη ή αλλαγή με ομαλό τρόπο.

1.4.3.3 Επίλυση Επικινδυνότητας

Η επίλυση επικινδυνότητας είναι η διαδικασία που πραγματοποιείται όταν οι κίνδυνοι λαμβάνουν χώρα, δηλαδή όταν μετατρέπονται σε πραγματικά προβλήματα.

Ουσιαστικά η επίλυση αποτελεί την υλοποίηση της προηγούμενης φάσης, μιας και στηρίζεται στο σχέδιο αντιμετώπισης των κινδύνων που έχει δημιουργηθεί.

Τρόποι επίλυσης της επικινδυνότητας υπάρχουν πολλοί και ποικίλοι, αλλά οι σημαντικότεροι εξ αυτών είναι αυτοί που ήδη έχουν παρουσιαστεί σε προηγούμενη παράγραφο ως επίπεδα διαχείρισης επικινδυνότητας (Παρ. 1.2.2).

1.4.3.4 Επίβλεψη Επικινδυνότητας

Η τελευταία φάση της διαδικασίας διαχείρισης επικινδυνότητας, είναι η επίβλεψη. Είναι αυτή που παρέχει τη διαβεβαίωση ότι ακολουθείται το σχέδιο που έχει τεθεί καθώς και την αποτελεσματικότητα αυτού.

Πιο συγκεκριμένα, σε αυτό το σημείο παρακολουθείται η εξέλιξη κάθε κινδύνου και η πορεία της επίλυσής του, ενώ όπου κρίνεται απαραίτητο, πραγματοποιούνται κατάλληλες διορθωτικές κινήσεις.

Τέλος περιοδικά, όπως άλλωστε είναι και αναμενόμενο, αναγνωρίζονται νέοι κίνδυνοι, οι οποίοι αρχικά εκτιμούνται, κατόπιν τους παρέχεται κάποια προτεραιότητα και στη συνέχεια βρίσκονται και εφαρμόζονται τρόποι αντιμετώπισής τους.

Από τα παραπάνω γίνεται φανερό ότι η επίβλεψη της επικινδυνότητας αποτελεί ένα εξαιρετικά σημαντικό κομμάτι της γενικότερης διαδικασίας διαχείρισης, αφού χωρίς αυτή θα αυξανόταν σημαντικά η πιθανότητα αποτυχίας ως προς την καταπολέμηση των κινδύνων, καθότι είναι σχεδόν σίγουρο ότι ποτέ δεν θα λειτουργήσουν όλα σύμφωνα με το τρόπο που σχεδιάστηκαν. Αλλά και αν ακόμα λειτουργούσαν κανείς δεν μπορεί να εγγυηθεί ότι στην πορεία δε θα προκύψουν νέοι κίνδυνοι οι οποίοι δεν έχουν προβλεφθεί.

1.4.4 Διαχείριση Επικινδυνότητας στον Κύκλο Ζωής Ανάπτυξης Λογισμικού

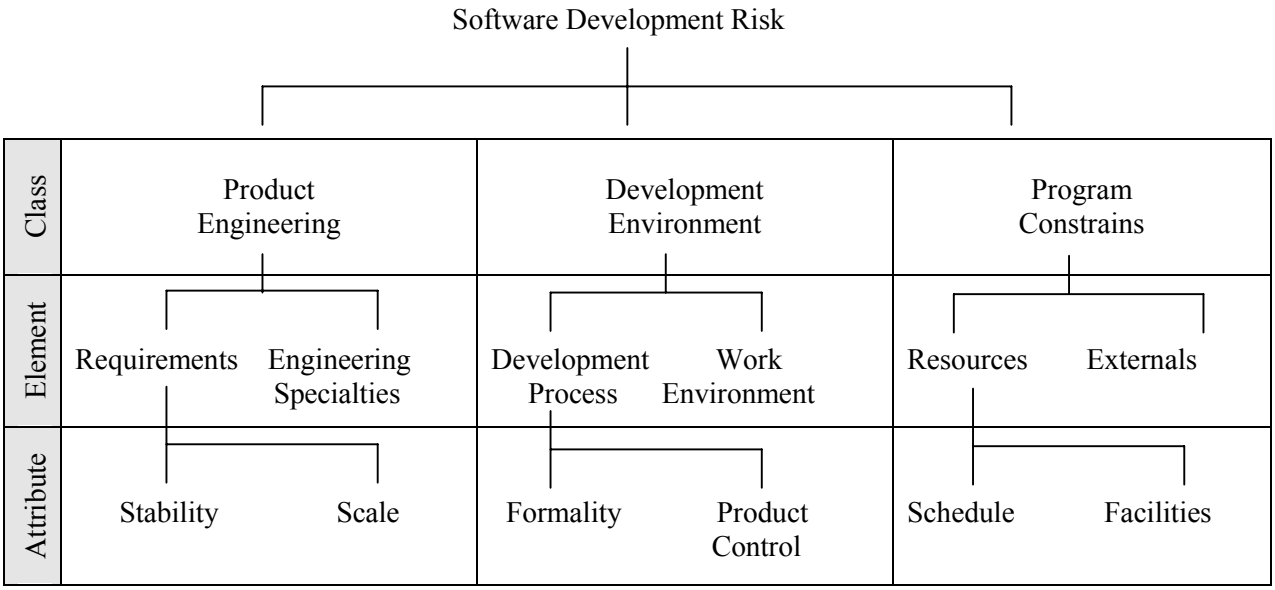
Προκειμένου να δοθεί μια ικανοποιητική εικόνα της εφαρμογής της διαδικασίας της διαχείρισης επικινδυνότητας στον κύκλο ζωής του λογισμικού, από την πληθώρα των προσεγγίσεων που υπάρχουν επιλέχθηκε να παρουσιαστούν δύο, οι οποίες θεωρείται ότι είναι αρκετά αντιπροσωπευτικές ως προς τα βήματα που ακολουθούνται.

1.4.4.1 Software Risk Evaluation (SRE)

Το SRE μοντέλο, αναπτύχθηκε από το Software Engineering Institute (SEI) του Carnegie Melon University και στοχεύει στην αναγνώριση, ανάλυση και μετριασμό των τεχνικών κινδύνων που σχετίζονται με την ανάπτυξη συστημάτων λογισμικού. Το μοντέλο περιλαμβάνει το παράδειγμα διαχείρισης επικινδυνότητας (Risk Management Paradigm), τη ταξινόμηση της επικινδυνότητας ανάπτυξης λογισμικού (Software Development Risk Taxonomy) και βασιζόμενες στη ταξινόμηση μεθόδους ανάλυσης σε μορφή ερωτηματολογίων (Taxonomy-Based Questionnaire methods of analysis).

Το Risk Management Paradigm εμπεριέχει τέσσερις βασικές διαδικασίες: ανίχνευση (detection), προσδιορισμό (specification), εκτίμηση (assessment) και συνένωση (consolidation). Η ανίχνευση βρίσκει τους τεχνικούς κινδύνους που απειλούν το λογισμικό, ο προσδιορισμός καταγράφει σημαντικά θέματα σχετικά με τους κινδύνους που έχουν αναγνωρισθεί, η εκτίμηση καθορίζει τη σοβαρότητα κάθε κινδύνου και η συνένωση συγχωνεύει, συνδυάζει και συνοψίζει δεδομένα που αφορούν τους κινδύνους σε μία περιεκτική μορφή.

Η Software Development Risk Taxonomy (SDRT), που απεικονίζεται στο σχήμα 1.2, παρέχει μια βάση για την οργάνωση και τη μελέτη του πεδίου των θεμάτων ανάπτυξης λογισμικού. Μπορεί να αποτελέσει ένα συστηματικό τρόπο εκμαίευσης και οργάνωσης κινδύνων και παρέχει ένα σημαντικό σκελετό για τη μέθοδο διαχείρισης επικινδυνότητας και την τεχνική ανάπτυξη.



Σχήμα 1.4: *Software Development Risk Taxonomy*

Το βασιζόμενο στη ταξινόμηση ερωτηματολόγιο (TBQ) είναι ένα εργαλείο που χρησιμοποιείται για την αναγνώριση των κινδύνων που υπεισέρχονται στην ανάπτυξη του λογισμικού. Οι περισσότερες αποτυχίες λογισμικού θα μπορούσαν να έχουν αποφευχθεί ή έστω σημαντικά μειωθεί, εάν υπήρχε από νωρίς σαφές ενδιαφέρον για την αναγνώριση και την αντιμετώπιση των τμημάτων υψηλού κινδύνου του έργου λογισμικού. Το TQB παρέχει τη διαβεβαίωση ότι καλύπτονται όλες οι ενδεχόμενες περιοχές κινδύνου και περιλαμβάνει συγκεκριμένες ιδέες και ερωτήσεις που επιτρέπουν στο άτομο που διαχειρίζεται το ερωτηματολόγιο να ερευνά για κινδύνους.

Η SRE μέθοδος αποτελείται από τέσσερις πρωτεύουσες διαδικασίες, οι οποίες και προαναφέρθηκαν και τρεις υποστηρικτικές. Οι υποστηρικτικές SRE διαδικασίες του Risk Management Paradigm είναι ο σχεδιασμός (planning) και ο συντονισμός (coordination), η επαλήθευση (verification) και η επικύρωση (validation) και τέλος η εκπαίδευση (training) και η επικοινωνία (communication). Η μέθοδος χρησιμοποιεί την SDRT και το TQB για ανίχνευση νωρίς στον κύκλο ζωής ανάπτυξης, ώστε να εντοπίζει τις τεχνικά επικίνδυνες περιοχές του λογισμικού. Κάθε κίνδυνος καταγράφεται μαζί με την κατάστασή του, τις συνέπειές του και την πηγή του. Κατόπιν κατηγοριοποιείται σύμφωνα με το αν ανήκει σε κλάση (class), στοιχείο (element) ή ιδιοχαρακτηριστικό (attribute) της ταξινόμησης. Το μέγεθος και η σοβαρότητα κάθε επιμέρους τεχνικού κίνδυνου, καθορίζεται από το γινόμενο της πιθανότητας εμφάνισης του κινδύνου επί τη σοβαρότητα της επίπτωσης που θα έχει. Με αυτό τον τρόπο προκύπτει ένας πίνακας επιπέδων σοβαρότητας κινδύνων (πίνακας 1.1). Τέλος, βάσει ειδικευμένης γνώμης, τα δεδομένα του κινδύνου συγχωνεύονται και συνδυάζονται σε μία περιεκτική φόρμα λήψης αποφάσεων.

Πιθανότητα / Σοβαρότητα	Πολύ πιθανό	Πιθανό	Απίθανο
Καταστροφική/ Κρίσιμη	Υψηλό		
Μέτρια		Μέτριο	
Αμελητέα			Χαμηλό

Πίνακας 1.1 : *Πίνακας Επιπέδων Σοβαρότητας Κινδύνων*

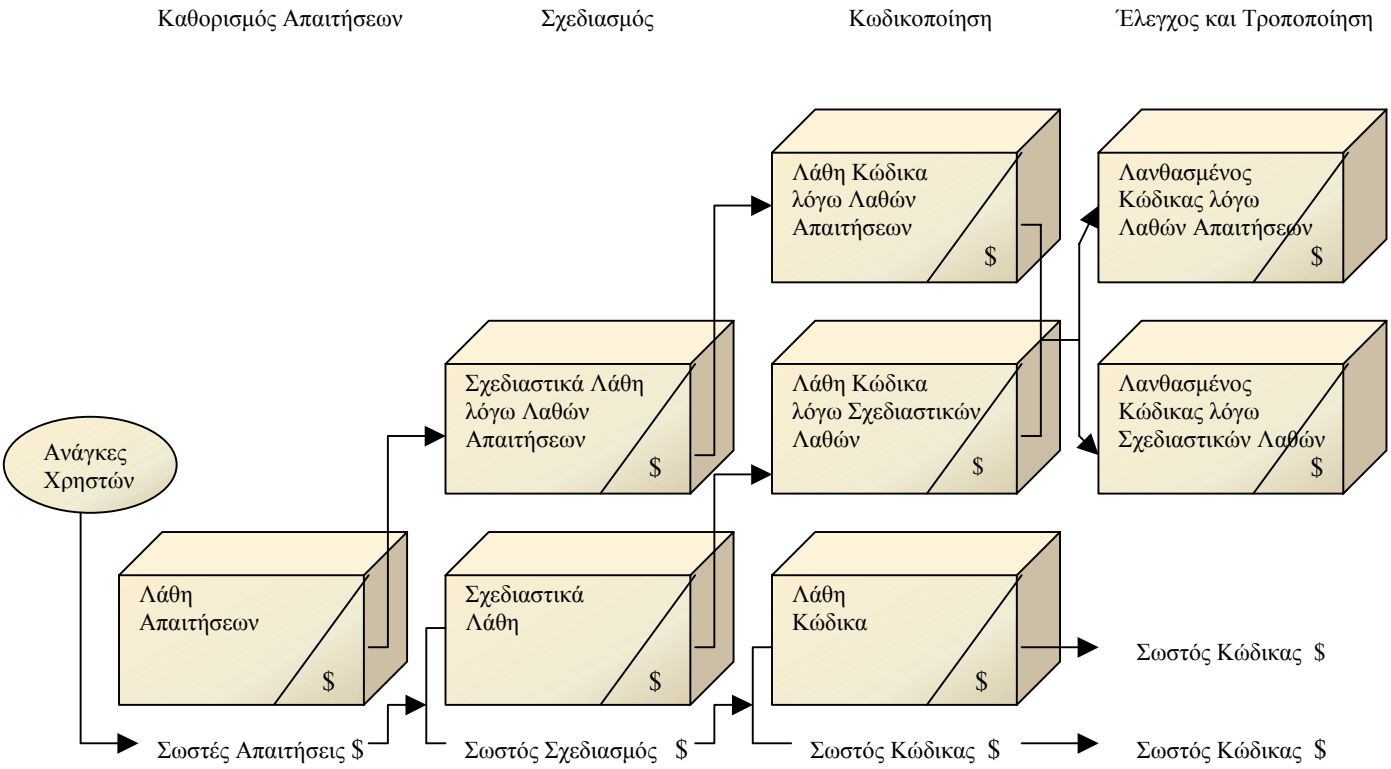
1.4.4.2 The Cleanroom Approach

Το μοντέλο που παρουσιάστηκε πιο πάνω, έχει εφαρμογή σε οποιαδήποτε από τις φάσεις του κύκλου ζωής ανάπτυξης λογισμικού, όπως συμβαίνει και με τα περισσότερα από τα υπόλοιπα μοντέλα που έχουν αναπτυχθεί. Όμως στις περισσότερες των περιπτώσεων, αν και είναι πιθανό να αναγνωρίζονται επικίνδυνες περιοχές νωρίς στο κύκλο ζωής, στην πραγματικότητα διαχειρίζονται αρκετά αργότερα.

Το γεγονός αυτό θεωρείται από πολλούς υπεύθυνο για το ότι δεν μπορεί να υπάρξει κατάλληλη και αποτελεσματική αντιμετώπιση των διαφόρων κινδύνων που απειλούν το λογισμικό. Με άλλα λόγια είναι σημαντικό, η προσπάθεια για αποφυγή των κινδύνων, να ξεκινά από το πρώτο στάδιο του κύκλου ζωής ανάπτυξης, όπου πραγματοποιείται η μετατροπή των απαιτήσεων σε προδιαγραφές.

Αυτή ακριβώς η θεώρηση υιοθετείται και από την προσέγγιση cleanroom (cleanroom approach) [Adler99], η οποία περιλαμβάνει διαδικασίες που στοχεύουν στο να μην επιτρέπουν την εισχώρηση λαθών στο πρόγραμμα, προσπαθώντας να τα καταπολεμούν πριν τη κωδικοποίηση ή τον έλεγχο, πράγμα που έχει ως αποτέλεσμα το προϊόν λογισμικού να διαθέτει υψηλότερη ποιότητα και αξιοπιστία.

Εκτός όμως από αυτά τα οφέλη, επιτυγχάνεται επίσης και μεγάλη μείωση του κόστους διαχείρισης και αντιμετώπισης των λαθών, πράγμα που γίνεται φανερό από το σχήμα που ακολουθεί. Το σχήμα παρουσιάζει πόσο αυξάνεται το κόστος κατά τη διάρκεια του κύκλου ζωής ανάπτυξης λογισμικού, όσο τα λάθη δεν αντιμετωπίζονται έγκαιρα.



Σχήμα 1.5: *Πολλαπλασιασμός Κόστους Λαθών Λογισμικού*

1.5 Ευριστική Ανάλυση Επικινδυνότητας (Heuristic Risk Analysis)

Ανάμεσα στις διάφορες μεθόδους ανάλυσης και διαχείρισης της επικινδυνότητας του λογισμικού, υπάρχουν και κάποιες όχι τόσο δομημένες και αυστηρώς καθορισμένες, οι οποίες συνηθίζεται να αποκαλούνται ευριστικές. Συνήθως σκοπεύουν σε πληρέστερη κατανόηση των διαφόρων πτυχών ενός προβλήματος και διεύρυνση του ορίζοντα των δυνατοτήτων επίλυσής του, μέσω διαφόρων ερωτήσεων, προτάσεων και οδηγιών.

Θεωρήθηκε σκόπιμο να παρουσιαστούν δύο προσεγγίσεις που ανήκουν στην κατηγορία των ευριστικών, οι οποίες μπορεί να θεωρηθούν ως συμπληρωματικές, έχοντας κάθε μία τα δικά της δυνατά σημεία [Bach99b].

1.5.1 Inside-Out προσέγγιση

Σύμφωνα με αυτή την προσέγγιση, εξετάζονται λεπτομέρειες της κατάστασης και αναγνωρίζονται οι κίνδυνοι που σχετίζονται με αυτές. Δηλαδή, μελετάται ένα προϊόν

και κάθε φορά επαναλαμβάνεται η ερώτηση: "Τί μπορεί να πάει λάθος σε αυτό το σημείο;" Πιο συγκεκριμένα, για κάθε κομμάτι του προϊόντος πρέπει να θέτονται τα εξής ερωτήματα :

- **Ευπάθειες (Vulnerabilities):** Τι αδυναμίες ή πιθανές αποτυχίες υπάρχουν σε αυτό το κομμάτι του λογισμικού;

- **Απειλές (Threats):** Τι είδους είσοδοι ή καταστάσεις, που πιθανόν να εκμεταλλευτούν μία ευπάθεια ή να προκαλέσουν μια αποτυχία, μπορούν να λάβουν χώρα σε αυτό το τμήμα του λογισμικού;

- **Θύματα (Victims):** Ποιος ή τι θα μπορούσε να επηρεαστεί από πιθανές αποτυχίες και σε τι μέγεθος;

Αυτή η προσέγγιση απαιτεί σημαντική τεχνική οξυδέρκεια, καθώς και πολύ καλή επικοινωνία μεταξύ των εμπλεκομένων στην ανάπτυξη του συστήματος λογισμικού, έτσι ώστε να πραγματοποιείται όσο το δυνατόν καλύτερα η αναγνώριση, η εκτίμηση και η αντιμετώπιση των κινδύνων.

1.5.2 Outside-In προσέγγιση

Ενώ η προσέγγιση Inside-Out έθετε το ερώτημα "Τι κίνδυνοι σχετίζονται με αυτό το πράγμα;", η προσέγγιση Outside-In θέτει το αντίθετο ακριβώς ερώτημα, δηλαδή: "Τι πράγματα σχετίζονται με αυτού του είδους τους κινδύνους;".

Σε αυτή λοιπόν την περίπτωση, έχοντας διαθέσιμο ένα σύνολο κινδύνων, γίνεται προσπάθεια ταιριάσματός τους με τις διάφορες λεπτομέρειες της εν λόγω κατάστασης. Πρόκειται για μια πιο γενική προσέγγιση από ότι η προηγούμενη και ίσως και περισσότερο εύκολη. Αυτό που συμβαίνει είναι ότι βάσει ενός προκαθορισμένου καταλόγου κινδύνων, καθορίζεται το κατά πόσο αυτοί μπορούν να εμφανιστούν στην συγκεκριμένη περίπτωση. Ο προκαθορισμένος αυτός κατάλογος, μπορεί είτε να είναι αποτυπωμένος στο χαρτί, είτε απλά να υπάρχει στο μυαλό κάποιων από τους επιφορτισμένους με την ευθύνη αυτής της διαδικασίας ανθρώπων, βάσει περασμένης εμπειρίας. Συνήθως χρησιμοποιούνται τρία είδη καταλόγων: κατηγορίες κριτηρίων ποιότητας (quality criteria categories), γενικές λίστες κινδύνων (generic risk lists) και κατάλογοι κινδύνων (risk catalogs).

Κατηγορίες Κριτηρίων Ποιότητας: Αυτές οι κατηγορίες σχετίζονται κυρίως με τα διάφορα είδη απαιτήσεων. Θέτουν ερωτήσεις του τύπου : "Τι θα γινόταν εάν οι απαιτήσεις που σχετίζονται με οποιαδήποτε από αυτές τις κατηγορίες δεν μπορούν να ικανοποιηθούν;" ή "Πόση προσπάθεια δικαιολογείται σε έλεγχο για να υπάρξει η

διαβεβαίωση ότι οι απαιτήσεις ικανοποιούνται σε ένα ικανοποιητικό βαθμό;". Πιο κάτω παρουσιάζονται τα διάφορα είδη απαιτήσεων με τα αντίστοιχα τους ερωτήματα:

- Ικανότητα (Capability): Μπορεί να εκτελέσει τις απαιτούμενες λειτουργίες;
- Αξιοπιστία (Reliability): Θα δουλέψει ικανοποιητικά και θα καταφέρει να αποφεύγει την αποτυχία σε όλες τις απαιτούμενες περιπτώσεις;
- Χρησιμοποιησιμότητα (Usability): Πόσο εύκολο είναι για τον πραγματικό χρήστη να χρησιμοποιήσει το προϊόν;
- Επίδοση (Performance): Πόσο γρήγορο και ανταποκρίσιμο είναι;
- Εγκαταστασιμότητα (Installability): Πόσο εύκολα μπορεί να εγκατασταθεί στην πλατφόρμα για την οποία προορίζεται;
- Συμβιβασιμότητα (Compatibility): Πόσο καλά λειτουργεί με εξωτερικά μέρη (components) και διαμορφώσεις;
- Υποστηριξιμότητα (Supportability): Πόσο οικονομικό θα είναι ώστε να παρέχει υποστήριξη σε χρήστες του προϊόντος;
- Ελεγξιμότητα (Testability): Πόσο αποτελεσματικά μπορεί να ελεγχθεί το προϊόν;
- Συντηρισιμότητα (Maintability): Πόσο οικονομική θα είναι η δημιουργία, διόρθωση και καλυτέρευση του προϊόντος;
- Μεταφερσιμότητα (Portability): Πόσο οικονομική θα είναι η μεταφορά ή η επαναχρησιμοποίηση της τεχνολογίας αλλού;

Γενικές Λίστες Κινδύνων: Ως γενικοί κίνδυνοι χαρακτηρίζονται οι κίνδυνοι οι οποίοι μπορεί να παρουσιάζονται σε κάθε σύστημα, όπως για παράδειγμα οι ακόλουθοι :

- Πολύπλοκο (Complex) – οτιδήποτε δυσανάλογα μεγάλο ή περίπλοκο
- Νέο (New) – οτιδήποτε δεν έχει ιστορία στο προϊόν
- Αλλαγμένο (Changed) – οτιδήποτε έχει διαφοροποιήθηκε ή "βελτιώθηκε"
- Upstream Depedency – οτιδήποτε του οποίου η αποτυχία μπορεί να προκαλέσει αποτυχία και στο υπόλοιπο σύστημα

- Downstream Depedency – οτιδήποτε είναι εξαιρετικά ευαίσθητο σε αποτυχίες που συμβαίνουν στο υπόλοιπο σύστημα

- Κρίσιμο (Critical) – οτιδήποτε του οποίου η αποτυχία θα μπορούσε να προκαλέσει σημαντική ζημιά

- Ακριβές (Precise) – οτιδήποτε πρέπει να ικανοποιήσει ακριβώς τις απαιτήσεις του

- Δημοφιλές (Popular) – οτιδήποτε πρόκειται να χρησιμοποιηθεί πολύ

- Στρατηγικό (Strategic) – οτιδήποτε το οποίο έχει ειδική σημασία για την επιχείρηση, όπως ένα χαρακτηριστικό το οποίο σε θέτει εκτός ανταγωνισμού

- Τρίτης-οντότητας (Third-party) – οτιδήποτε χρησιμοποιείται στο προϊόν, αλλά αναπτύσσεται εκτός του έργου

- Κατανεμημένο (Distributed) – οτιδήποτε βρίσκεται διασκορπισμένο στο χρόνο ή το χρόνο, αλλά του οποίου τα στοιχεία πρέπει να δουλέψουν μαζί

- Προβληματικό (Buggy) – οτιδήποτε είναι γνωστό να έχει πολλά προβλήματα

- Πρόσφατη Αποτυχία (Recent Failure) – οτιδήποτε διαθέτει πρόσφατη ιστορία αποτυχίας

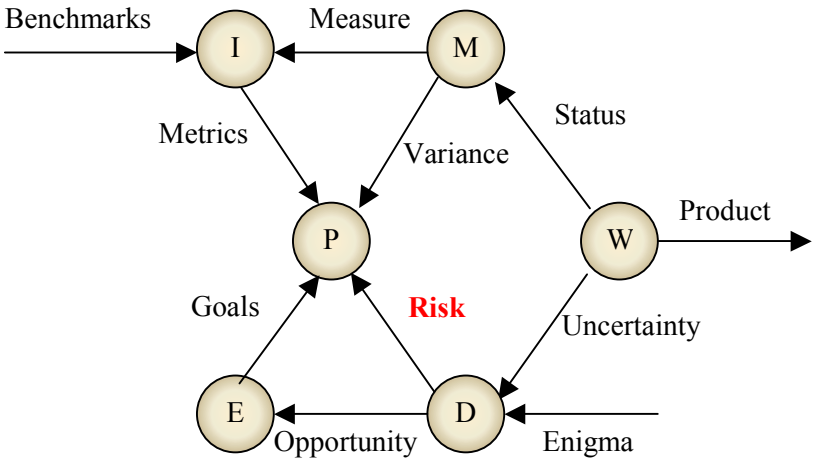
Κατάλογοι Κινδύνων: Ένας κατάλογος κινδύνων είναι μια περίληψη των κινδύνων που ανήκουν σε μία συγκεκριμένη περιοχή. Κάθε γραμμή σε ένα κατάλογο κινδύνων ξεκινά με τη φράση "Είναι πιθανό να αντιμετωπίσουμε το πρόβλημα να…". Οι κατάλογοι κινδύνων παρακινούνται από τον έλεγχο του ίδιου τύπου τεχνολογίας συνέχεια. Υπάρχει η δυνατότητα να δημιουργηθεί ένας κατάλογος κινδύνων, μόνο με την κατηγοριοποίηση του είδους των προβλημάτων που παρατηρούνται κατά την διάρκεια του ελέγχου.

Οι κατάλογοι κινδύνων μπορούν να χρησιμοποιηθούν με πολλούς και ποικίλους τρόπους με σκοπό να προσδιοριστούν οι επικίνδυνες περιοχές και να εκτιμηθεί η σοβαρότητα της κάθε μίας για το υπό εξέταση λογισμικό. Είναι επίσης πολύ πιθανό, στην πορεία να εμφανιστούν και κίνδυνοι οι οποίοι δεν βρίσκονται καταχωρημένοι σε αυτούς τους καταλόγους και οποίοι φυσικά πρέπει να καταγραφούν. Τα συμπεράσματα που θα προκύψουν από αυτή τη διαδικασία, μπορούν να χρησιμοποιηθούν ώστε να ληφθούν κατάλληλες αποφάσεις ελέγχου, όπως το σε ποιες περιοχές πρέπει να εστιαστεί ο έλεγχος.

1.6 Διαχείριση Επικινδυνότητας και Διαχείριση Ανάπτυξης Συστήματος Λογισμικού

Όπως προαναφέρθηκε, η έννοια της επικινδυνότητας και η ανάγκη για διαχείρισή της ενυπάρχει καθ' όλη την διάρκεια του κύκλου ζωής ανάπτυξης λογισμικού. Άλλωστε, από τη στιγμή που οι διάφορες μεθοδολογίες ανάπτυξης λογισμικού που χρησιμοποιούνται έχουν ως έναν από τους κύριους στόχους τους τη μείωση της πιθανότητας αποτυχίας του έργου ανάπτυξης και κατ' επέκταση τη μείωση της πιθανότητας εμφάνισης αρνητικών συνεπειών από τη χρήση του εν λόγω λογισμικού, γίνεται φανερό ότι σε τελική ανάλυση εμπεριέχουν μια διαδικασία διαχείρισης επικινδυνότητας.

Αυτή ακριβώς η εμπλοκή της επικινδυνότητας στην διαχείριση ανάπτυξης ενός συστήματος λογισμικού, γίνεται φανερή από το μοντέλο των "έξι αρχών" (six-discipline model) [Hall98], το οποίο παρουσιάζεται στο παρακάτω σχήμα:



Σχήμα 1.6 : *The Six-Discipline Model*

Το πιο πάνω μοντέλο περιγράφει τη σχέση ανάμεσα στις αρχές που απαιτείται να ληφθούν υπόψη, προκειμένου να διαχειριστεί με επιτυχία το έργο ανάπτυξης ενός συστήματος λογισμικού. Πιο αναλυτικά οι αρχές αυτές έχουν ως εξής :

Envision (E): Είναι η ανάπτυξη ενός οράματος για το προϊόν λογισμικού, ο μετασχηματισμός δηλαδή ορισμένων ιδεών σε στόχους.

Plan (P): Αναφέρεται στην αντιστοίχηση των διαθέσιμων πόρων στις απαιτήσεις οι οποίες έχουν προέλθει από τους αντικειμενικούς σκοπούς του έργου.

Work (W): Πρόκειται για την υλοποίηση του σχεδίου (plan) προκειμένου να παραχθεί το προϊόν (product). Προϊόντα της διαδικασίας αυτής είναι η κατάσταση (status) και η αβεβαιότητα (uncertainty) που αναπτύσσονται καθώς προχωράει η όλη διαδικασία. Συμπεριλαμβάνει την αναγνώριση αγνώστων στοιχείων στα ενδιάμεσα προϊόντα (π.χ. μη ολοκληρωμένες απαιτήσεις) καθώς και της αβεβαιότητας του περιβάλλοντος στο οποίο πραγματοποιείται η ανάπτυξη του λογισμικού (π.χ. μη γνώριμα εργαλεία). Τα προϊόντα παραμένουν σε επεξεργασία έως ότου ικανοποιήσουν τα κριτήρια ποιότητας που έχουν τεθεί.

Measure (M): Αναφέρεται στη σύγκριση αναμενόμενων και πραγματικών αποτελεσμάτων. Δηλαδή, παρακολουθούνται τα αποτελέσματα και εκτιμάται η πρόοδος με σύγκριση της κατάστασης (status) με το σχέδιο (plan).

Improve (I): Αναλύονται τα σημεία αναφοράς μετρήσεων (benchmarks) και οι μετρήσεις (measures) του έργου, προκειμένου να βελτιωθούν οι οργανωσιακές διαδικασίες και μετρικές (metrics). Με άλλα λόγια πρόκειται για μια διαδικασία συνεχούς μάθησης από παλαιότερη εμπειρία.

Discover (D): Η έκτη αρχή αναφέρεται στην προσπάθεια εκτίμησης του μέλλοντος. Γίνεται εκτίμηση της αβεβαιότητας (uncertainty) και της επικινδυνότητας (risk) που υπάρχει και πραγματοποιούνται κατάλληλες αλλαγές στο σχέδιο (plan). Απαιτείται επίσης η ύπαρξη γνώσης γύρω από εξωτερικά "αινίγματα" (enigma) που επηρεάζουν και δυσκολεύουν το έργο. Σε αυτή τη φάση θέτονται υπό αμφισβήτηση διάφορα ζητήματα, ώστε να ανακαλύπτονται ευκαιρίες (opportunities). Η σημαντικότητα της έκτης αυτής αρχής έγκειται στο ότι παρέχει την είσοδο (input) που απαιτείται για να τροποποιηθεί το όραμα.

Σύμφωνα με το μοντέλο που μόλις περιγράφθηκε, επιβεβαιώνεται ο αρχικός ισχυρισμός, το ότι δηλαδή η επικινδυνότητα και η ορθή διαχείρισή της αποτελεί ένα εξαιρετικά σημαντικό κομμάτι στην γενικότερη προσπάθεια διαχείρισης του έργου ανάπτυξης του συστήματος λογισμικού, από τη στιγμή που επηρεάζει τη διαδικασία σχεδιασμού του έργου ανάπτυξης.

ΚΕΦΑΛΑΙΟ 2

2.1 Εισαγωγικά

Ενώ στο κεφάλαιο που προηγήθηκε, παρουσιάστηκε η εφαρμογή της διαδικασίας διαχείρισης επικινδυνότητας σε όλα τα στάδια του κύκλου ζωής ανάπτυξης λογισμικού, σε αυτό το κεφάλαιο υπάρχει επικέντρωση μόνο στη φάση του ελέγχου του.

Έτσι, επιχειρείται ένας προσδιορισμός του τρόπου συσχέτισης του ελέγχου και της διαχείρισης επικινδυνότητας του λογισμικού, και παρατίθενται επιχειρήματα που συνηγορούν υπέρ της άποψης που υποστηρίζει ότι είναι εξαιρετικά σημαντικό για την εξέλιξη ενός έργου λογισμικού, οι αποφάσεις ελέγχου να λαμβάνονται μεταξύ των άλλων και βάσει των εμπλεκόμενων κινδύνων.

2.2 Σχέση Ελέγχου Λογισμικού και Διαχείρισης Επικινδυνότητας

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.

Γι' αυτούς ακριβώς τους λόγους απαιτείται πριν από την παράδοση οποιουδήποτε προϊόντος ή εφαρμογής, 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).

Γιατί βέβαια, μπορεί η αποτυχία του λογισμικού να μην είναι ποτέ ευχάριστη για τον υπεύθυνο ανάπτυξής του, αλλά είναι σαφώς προτιμότερο να λάβει χώρα κατά τη διάρκεια του ελέγχου παρά μετά την παράδοση. Διότι κατά τη διάρκεια του ελέγχου, η εύρεση ενός σφάλματος απαιτεί διόρθωση και επανέλεγχο, ενώ κατά τη διάρκεια της χρήσης, η αποτυχία μπορεί να οδηγήσει σε μη ικανοποίηση, ενόχληση και σε ακραίες περιπτώσεις ακόμα και σε απώλεια ζωής [Voas94].

Το λογισμικό μπορεί να ελεγχθεί σε διάφορα στάδια της ανάπτυξής του και όπως όλες οι δραστηριότητες ανάπτυξης, απαιτεί προσπάθεια και κατ' επέκταση οικονομικούς πόρους. Οι υπεύθυνοι ανάπτυξης πρέπει να υπολογίζουν ότι το 30% έως και 70% της προσπάθειας ανάπτυξης του έργου θα σπαταλάται σε διαδικασίες επαλήθευσης και επικύρωσης [IPL96a].

Βέβαια 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.

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 [Rosen96].

Προκύπτει λοιπόν το συμπέρασμα, ότι ενώ σε οποιαδήποτε από τις προηγούμενες φάσεις του κύκλου ζωής ανάπτυξης λογισμικού υπάρχει η δυνατότητα ελαχιστοποίησης έως και εξάλειψης του μεγέθους του κινδύνου που απομένει μετά την ολοκλήρωσή της, στη φάση του ελέγχου κάτι τέτοιο δεν μπορεί να επιτευχθεί. Πρέπει δηλαδή να γίνει αποδεκτό apriori το γεγονός ότι πάντα θα υπάρχει κάποιο ποσοστό κινδύνου που θα απομένει μετά την ολοκλήρωση της φάσης του ελέγχου.

Γι' αυτό τον λόγο, απαιτείται να δίνεται εξέχουσα σημασία στη διαδικασία διαχείρισης επικινδυνότητας κατά τον έλεγχο του λογισμικού, υπό την παραδοχή βέβαια ότι έχει διαχειριστεί αποτελεσματικά ο κίνδυνος που υπεισέρχεται στο λογισμικό κατά τις προηγούμενες από τον έλεγχο φάσεις.

Υπό αυτή την προοπτική τίθεται και το κρίσιμο ζήτημα καθορισμού των κριτηρίων σταματήματος του 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.

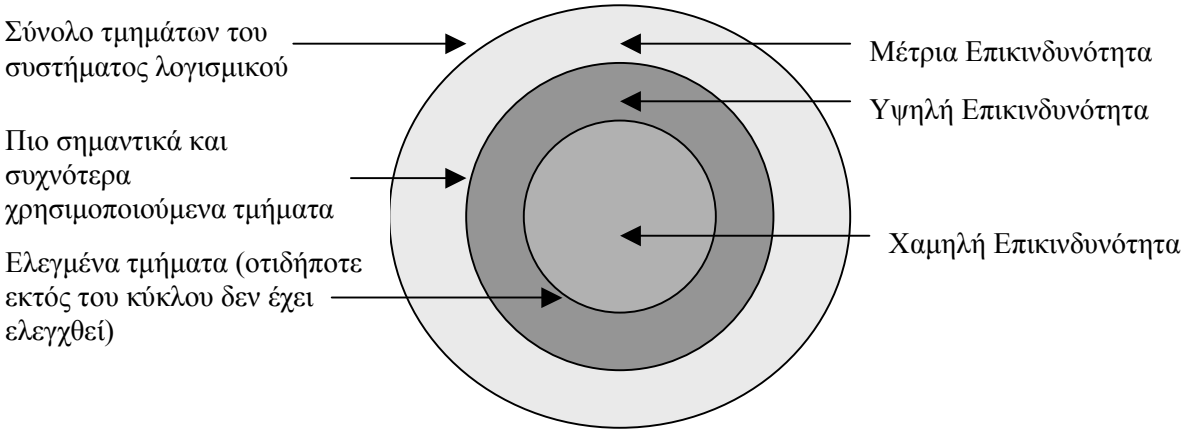
Εκτός όμως από τους παραπάνω παράγοντες, εξίσου σημαντική παράμετρο αποτελεί και η κρισιμότητα του λογισμικού η οποία μπορεί να καθορίζεται από τον σκοπό για τον οποίο πρόκειται να χρησιμοποιηθεί το προϊόν, το ποιος πρόκειται να το χρησιμοποιήσει και τι απαιτήσεις έχει από αυτό, ή το ποιες θα είναι οι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

Αν λοιπόν θεωρήσουμε 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, διαπιστώνουμε 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. Το γεγονός αυτό συνηγορεί υπέρ της απόψεως ότι 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.

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. Ο έλεγχος δηλαδή αποτελεί στην πραγματικότητα μια διαδικασία διαχείρισης επικινδυνότητας, που στοχεύει στην εξασφάλιση εμπιστοσύνης στο λογισμικό. Άλλωστε εάν δεν υπήρχαν κίνδυνοι που να απειλούσαν το λογισμικό δεν θα υπήρχε και λόγος διεξαγωγής του ελέγχου.

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 και κατ' επέκταση να διαφοροποιείται η επικινδυνότητά τους.

Το σχήμα που ακολουθεί, απεικονίζει αυτή ακριβώς τη διαβάθμιση της επικινδυνότητας στις διάφορες περιοχές του λογισμικού και τονίζει το γεγονός ότι οι κίνδυνοι αποτελούν βάση για τις διάφορες επιλογές ελέγχου [Kit95]:



Σχήμα 2.1 : *Η Επικινδυνότητα λογισμικού σαν βάση επιλογών ελέγχου*

Παρ' όλα αυτά, στα περισσότερα έργα ανάπτυξης λογισμικού, το πρόβλημα δεν έγκειται τόσο στη δυσκολία εντοπισμού και αντιμετώπισης των κινδύνων, αλλά στο ότι δεν υπάρχει προσπάθεια να εντοπιστούν και να εκτιμηθούν οι κίνδυνοι που τα απειλούν. Σαν αποτέλεσμα, 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 [Redm99].

Σύμφωνα με τα όσα προαναφέρθηκαν, κρίνεται πολύ σημαντικό να επιχειρείται εξέταση του βαθμού επικινδυνότητας του λογισμικού ανάλογα με το επίπεδο ελέγχου που έχει επιτευχθεί σε αυτό, με κύριο στόχο το επίπεδο κινδύνου που απομένει μετά την ολοκλήρωση του ελέγχου να βρίσκεται μέσα σε αποδεκτά για την κάθε περίπτωση όρια. Βέβαια, ο καθορισμός του επιτρεπτού επιπέδου στο οποίο πρέπει να κυμαίνεται ο κίνδυνος που τελικά απομένει, δεν αποτελεί καθόλου εύκολη υπόθεση, καθότι τις περισσότερες φορές, δεν υπάρχει σαφής γνώση του επιπέδου εμπιστοσύνης που απαιτείται να έχει το σύστημα.

2.3 Έλεγχος Βασιζόμενος στην Επικινδυνότητα και τις 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 [Bach99a].

Έχει διατυπωθεί η άποψη ότι χωρίς δηλωμένες 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 ισχυριστούμε ότι 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 και είναι βέβαια περιοριστικό να τις βλέπουμε ως ασύνδετες ιδέες.

Ένας πιο ευρύς τρόπος αντίληψης της ικανοποίησης των απαιτήσεων είναι να λαμβάνονται σοβαρά υπόψη οι κίνδυνοι που σχετίζονται με την παραβίασή τους.

Πιο συγκεκριμένα, μπορούν να καταγραφούν κάποιες οδηγίες για το πως πρέπει να βλέπουμε τον έλεγχο σε σχέση με τις απαιτήσεις και τους εμπλεκόμενους κινδύνους [Bach99a]:

1. Η ικανότητά μας να αναγνωρίζουμε προβλήματα σε ένα προϊόν είναι περιορισμένη και επηρεάζεται από το τρόπο με τον οποίο κατανοούμε την ύπαρξή τους. Ένα έγγραφο απαιτήσεων αποτελεί μια ενδεχόμενη πηγή πληροφοριών σχετικών με τα προβλήματα, αλλά βέβαια υπάρχουν και άλλες.

2. Προκαλούμε το κίνδυνο εφόσον παραδίδουμε ένα προϊόν το οποίο έχει σοβαρά προβλήματα. Ο πραγματικός σκοπός του ελέγχου είναι να φανερώσει αυτό το κίνδυνο και όχι απλά να αποδείξει την συμμόρφωση με τις απαιτήσεις που έχουν τεθεί.

3. Ιδιαίτερα σε καταστάσεις υψηλής κρισιμότητας, η διαδικασία ελέγχου θα είναι περισσότερο πειστική εάν βρισκόμαστε σε θέση να εκφράσουμε με ευκρίνεια και να δικαιολογήσουμε πως η στρατηγική ελέγχου σχετίζεται με τον ορισμό της ποιότητας.

4. Η διαδικασία ελέγχου θα είναι πιο αποτελεσματική εάν οι απαιτήσεις καθορίζονται με όρους οι οποίοι να συνδυάζουν το επιθυμητό με την ιδέα των κινδύνων, των οφελών και της σημασίας της κάθε απαίτησης.

ΚΕΦΑΛΑΙΟ 3

3.1 Εισαγωγικά

Από τη στιγμή που έχει αποδειχθεί το γεγονός ότι η επικινδυνότητα λογισμικού συνδέεται άμεσα με τον έλεγχο που διεξάγεται σε αυτό, καθώς και το ότι μετά το πέρας του ελέγχου αναπόφευκτα θα απομείνει ένα ποσοστό κινδύνου, αυτό που πρέπει να ακολουθήσει είναι ο προσδιορισμός των παραγόντων που συνιστούν το ποσοστό αυτό της εναπομείνασας επικινδυνότητας.

Ο καθορισμός και η ανάλυση αυτών ακριβώς των παραγόντων αποτελεί το βασικό στόχο του παρόντος κεφαλαίου, περιλαμβάνοντας παράλληλα και μια σύντομη αναφορά σε θέματα ελέγχου αντικειμενοστραφούς λογισμικού.

3.2 Παράγοντες Επικινδυνότητας Λογισμικού

Σε μία προσπάθεια καθορισμού της επικινδυνότητας του λογισμικού, σε σχέση πάντα με τον έλεγχο που πραγματοποιείται σε αυτό, προκύπτει ένα σύνολο παραγόντων που την επηρεάζουν άμεσα και το οποίο παρουσιάζεται στον πίνακα 3.1. Πρέπει να σημειωθεί ότι θεωρείται αναγκαίο να λαμβάνονται υπόψη οι συγκεκριμένοι παράγοντες, χωρίς αυτό να σημαίνει ότι δεν υπάρχει πιθανότητα κάποια στιγμή να εντοπιστούν και ορισμένοι επιπλέον.

α/α	Μνημονικό Όνομα Παράγοντα	Σημασία
1	CRIT	Κρισιμότητα Λογισμικού
2	SIZE	Μέγεθος Λογισμικού
3	TESM	Μέθοδος Ελέγχου Λογισμικού
4	TEXP	Εμπειρία Ομάδας Ελέγχου
5	CPLX	Πολυπλοκότητα Λογισμικού
6	FREQ	Συχνότητα Χρήσης
7	TIME	Χρόνος Ελέγχου
8	RESO	Διαθέσιμοι Πόροι

Πίνακας 3.1 : *Παράγοντες Επικινδυνότητας Λογισμικού*

Πιο αναλυτικά οι παράγοντες έχουν ως εξής :

3.2.1 Κρισιμότητα Λογισμικού (CRIT)

Η κρισιμότητα που χαρακτηρίζει ένα λογισμικό αποτελεί αδιαμφισβήτητα έναν από τους βασικότερους παράγοντες που επηρεάζουν την επικινδυνότητα του. Καθορίζεται συναρτήσει της σοβαρότητας των επιπτώσεων που πιθανό να υπάρξουν, εξαιτίας της εμφάνισης κάποιων μη επιθυμητών αποτελεσμάτων. Κατά συνέπεια, μπορεί να θεωρηθεί ότι ισχύει η σχέση :

$$CRIT = f(L(UO)) \quad (1)$$

όπου :
CRIT: η κρισιμότητα του λογισμικού
L(UO): η απώλεια που θα επέλθει λόγω του μη επιθυμητού αποτελέσματος
(Loss if the outcome is unsatisfactory)

Βάσει των προαναφερθέντων, προκύπτει μια κατηγοριοποίηση της κρισιμότητας του λογισμικού ανάλογα με τις επιπτώσεις που είναι πιθανό να προκληθούν και η οποία φαίνεται στον πίνακα που ακολουθεί :

Επιπτώσεις L(UO)	Κρισιμότητα (CRIT)					
	Πολύ Χαμηλή	Χαμηλή	Ονομαστική	Υψηλή	Πολύ Υψηλή	Εξαιρετικά Υψηλή
Μικρή ενόχληση	✓					
Μικρές, εύκολα ανακτήσιμες απώλειες, όπως απώλειες πληροφοριών ενός μικρού ποσοστού πελατών/χρηστών		✓				
Μέτριες, ανακτήσιμες απώλειες, όπως οικονομικές απώλειες περιορισμένης κλίμακας			✓			
Μεγάλες, δύσκολα ανακτήσιμες απώλειες, όπως μεγάλες οικονομικές απώλειες, ή απώλειες πληροφοριών που αφορούν μεγάλο αριθμό χρηστών/πελατών				✓		
Κοινωνικό χάος, το οποίο μπορεί να προκληθεί για παράδειγμα από δυσλειτουργία ενός αεροπορικού συστήματος κρατήσεων				✓		
Απειλή κατά του περιβάλλοντος					✓	
Απειλή κατά της ανθρώπινης ζωής						✓

Πίνακας 3.2 : *Κατάταξη του παράγοντα "Κρισιμότητα"*

Σύμφωνα με την κρισιμότητα που τα χαρακτηρίζει, τα συστήματα λογισμικού, ομαδοποιούνται σε κάποιες κατηγορίες. Έτσι στην υψηλότερη βαθμίδα βρίσκονται τα λεγόμενα safety-critical συστήματα λογισμικού. Ο όρος αυτός χρησιμοποιείται για να χαρακτηρίσει συστήματα των οποίων η αποτυχία μπορεί να οδηγήσει σε απειλή κατά της ανθρώπινης ζωής ή ακόμα και σε απώλεια αυτής, καθώς και σε άλλες σοβαρές συνέπειες όπως η περιβαντολλογική καταστροφή [McGet93].

Το πεδίο των εφαρμογών που μπορούν να χαρακτηριστούν ως safety-critical είναι ευρύ και καλύπτει ένα μεγάλο αριθμό βιομηχανικών τομέων, συμπεριλαμβανομένης της ιατρικής βιομηχανίας, της αεροναυπηγικής, της βιομηχανίας πετρελαίου (π.χ. συστήματα συναγερμού), της βιομηχανίας άμυνας, της πυρηνικής βιομηχανίας (π.χ. συστήματα ελέγχου), της βιομηχανίας μεταφορών, του αυτοματισμού και της ρομποτικής και άλλων τομέων.

Παραδείγματα safety-critical συστημάτων, είναι τα συστήματα σηματοδότησης σιδηροδρόμων, τα οποία πρέπει να επιτρέπουν στους ελεγκτές να κατευθύνουν σωστά τα τρένα, ώστε να αποφεύγεται οποιαδήποτε περίπτωση σύγκρουσης. Γίνεται λοιπόν φανερό ότι εξαρτώνται ζωές από την ορθή λειτουργία ενός τέτοιου συστήματος.

Ακόμα και κάτι τόσο απλό όπως τα φανάρια κυκλοφορίας στους δρόμους μπορούν να θεωρηθεί ως safety-critical σύστημα, καθότι ένα λάθος, όπως το να υπάρξει πράσινο φως και στις δύο κατευθύνσεις ενός σταυροδρομίου, μπορεί να προκαλέσει ένα αυτοκινητιστικό ατύχημα. Αλλά και μέσα στα αυτοκίνητα το λογισμικό που σχετίζεται με λειτουργίες όπως η διαχείριση της μηχανής, τον έλεγχο κίνησης και το μπλοκάρισμα των φρένων, θα μπορούσε κάποια στιγμή να αποτύχει με αποτέλεσμα να αυξηθεί η πιθανότητα δυστυχήματος [IPL96b].

Σε πολλές περιπτώσεις, για να χαρακτηριστούν συστήματα λογισμικού υψηλής κρισιμότητας και γενικότερα συστήματα που απαιτείται να διατηρούν υψηλά επίπεδα αξιοπιστίας, χρησιμοποιείται και ο όρος high-integrity systems, των οποίων μια πιθανή δυσλειτουργία μπορεί να επιφέρει διαφόρων ειδών σημαντικές απώλειες, όπως απώλεια ασφάλειας ή ιδιωτικότητας, απώλεια πληροφοριών σχετικών με ολόκληρη την επιχείρηση (π.χ. εγγραφές τραπέζης) ή και να προκαλέσει κοινωνικό χάος (π.χ. λόγω δυσλειτουργίας ενός αεροπορικού συστήματος κρατήσεων). Γενικότερα για ένα σύστημα υψηλής αξιοπιστίας ισχύει η σχέση :

High-integrity = secure + safe

Σε περιπτώσεις χειρισμού τέτοιου είδους συστημάτων λογισμικού, στόχος είναι η ελαχιστοποίηση οποιασδήποτε πιθανότητας αποτυχίας που θα μπορούσε να επιφέρει καταστροφικές συνέπειες. Με άλλα λόγια, ο κίνδυνος που απομένει μετά την

ολοκλήρωση του ελέγχου του λογισμικού, θα πρέπει να κυμαίνεται σε όσο το δυνατό χαμηλότερο επίπεδο.

Ακολουθούν τα συστήματα που χαρακτηρίζονται ως semi-critical, των οποίων οι απώλειες που μπορούν να προκληθούν από πιθανή δυσλειτουργία τους, είναι περιορισμένης κλίμακας και ανακτήσιμες. Σε αυτή την περίπτωση, υπάρχουν μεγαλύτερα περιθώρια ως προς το επίπεδο του κινδύνου που θα απομείνει μετά τον έλεγχο, από ότι στην προηγούμενη κατηγορία συστημάτων.

Στη χαμηλότερη βαθμίδα βρίσκονται τα non-critical συστήματα λογισμικού, τα οποία εάν παρουσιάσουν μια αποτυχία μπορούν να προκαλέσουν μόνο απλές ενοχλήσεις ή μικρές, εύκολα ανακτήσιμες απώλειες και συνήθως προορίζονται για έναν περιορισμένο αριθμό χρηστών. Κατά συνέπεια, σε αυτά τα συστήματα το επίπεδο κινδύνου που επιτρέπεται να απομένει κυμαίνεται σε σαφώς υψηλότερα επίπεδα σε σχέση με τις κατηγορίες των safety-critical και semi-critical συστημάτων.

Όμως το λογισμικό αποτελείται από περιοχές οι οποίες μπορεί να παρουσιάζουν και αυτές με τη σειρά τους διαβάθμιση ως προς την κρισιμότητά τους. Στόχος είναι η εστίαση στις πιο κρίσιμες περιοχές του λογισμικού, χωρίς όμως να αγνοούνται εντελώς και οι λιγότερο κρίσιμες, διότι λόγω της αλληλεπίδρασης που υπάρχει μεταξύ των διαφόρων περιοχών του λογισμικού, κάτι τέτοιο θα οδηγούσε σε εσφαλμένες αποφάσεις και δραστηριότητες.

3.2.2 Μέγεθος Λογισμικού (SIZE)

Ο έλεγχος που διεξάγεται σε ένα λογισμικό και κατ' επέκταση και η επικινδυνότητα που απομένει μετά την ολοκλήρωσή του, εξαρτάται άμεσα από το μέγεθός που αυτό έχει. Διότι όσο μεγαλύτερο είναι το μέγεθος του λογισμικού, τόσο μεγαλύτερη είναι και η πιθανότητα εμφάνισης σφαλμάτων και κατά συνέπεια και η πιθανότητα μη εντοπισμού ενός μέρους αυτών.

Γι' αυτόν ακριβώς το λόγο και προκειμένου να εκτιμηθεί η επικινδυνότητα που σχετίζεται με το λογισμικό, θεωρείται απαραίτητο να λαμβάνεται υπόψη και ο παράγοντας "μέγεθος". Θα μπορούσαμε να θεωρήσουμε ότι το μέγεθος του λογισμικού μετριέται σε χιλιάδες εντολές πηγαίου κώδικα (KDSI), αλλά κάτι τέτοιο δεν θα ήταν αρκετά αντιπροσωπευτικό για ορισμένες μορφές λογισμικού, όπως τα αντικειμενοστραφή (object-oriented) λογισμικά. Γι' αυτό τον λόγο θεωρήθηκε πιο ορθό για την μέτρηση του μεγέθους, να χρησιμοποιηθούν τρία διαφορετικά μεγέθη: Object Points, Unadjusted Function Points και γραμμές πηγαίου κώδικα (Source Lines of Code). Πιο συγκεκριμένα:

- **Object Points**:

  Ως τύποι αντικειμένων (object types) θεωρούνται οι οθόνες, τα reports και τα συστατικά μέρη που είναι εκφρασμένα σε γλώσσες τρίτης γενεάς (3GL Components).

- **Function Points**:

  Μετρούν το μέγεθος ενός έργου λογισμικού, με το να ποσοτικοποιούν την λειτουργικότητα της επεξεργασίας πληροφοριών που σχετίζεται με σημαντικά εξωτερικά δεδομένα ή εισόδους ελέγχου, εξόδους ή τύπους αρχείων. Έχουν αναγνωρισθεί πέντε διαφορετικοί τύποι, οι οποίοι παρουσιάζονται πιο κάτω:

Είσοδοι {External Input (Inputs)}	Μετρούν κάθε ξεχωριστό δεδομένο ή τύπο εισόδου ελέγχου χρήστη που (i) εισέρχεται στο εξωτερικό σύνορο του συστήματος λογισμικού και (ii) προσθέτει ή αλλάζει δεδομένα σε ένα εσωτερικό αρχείο.
Έξοδοι {External Output (Outputs)}	Μετρούν κάθε ξεχωριστό δεδομένο ή τύπο εξόδου ελέγχου που αφήνει το εξωτερικό σύνορο του συστήματος λογισμικού.
Εσωτερικά Λογικά Αρχεία {Internal Logical File (Files)}	Μετρούν κάθε λογικό σύνολο από δεδομένα χρήστη ή πληροφορίες ελέγχου του συστήματος λογισμικού, ως ένα λογικό τύπο εσωτερικού αρχείου. Συμπεριλαμβάνει κάθε λογικό αρχείο (π.χ. κάθε λογικό σύνολο δεδομένων) που δημιουργείται, χρησιμοποιείται ή συντηρείται από το σύστημα λογισμικού.
Εξωτερικά αρχεία Διεπαφών {External Interface Files (Interfaces)}	Αρχεία που περνούν ή μοιράζονται μεταξύ συστημάτων λογισμικού πρέπει να μετρούνται ως τύποι εξωτερικών αρχείων διεπαφών μέσα στο κάθε σύστημα.

Ερωτήματα {External Inquiry (Queries)}	Μετρούν κάθε ξεχωριστό συνδυασμό εισόδου-εξόδου, όπου μία είσοδος γεννά ένα άμεσο αποτέλεσμα, ως ένα τύπο ερωτήματος.

Πίνακας 3.3: *User Function Types*

3.2.3 Μέθοδος Ελέγχου (TESM)

Προκειμένου να πραγματοποιηθεί ο έλεγχος του λογισμικού, γίνεται πρώτ' απ' όλα επιλογή μιας κατάλληλης μεθόδου, από ένα σύνολο μεθόδων ελέγχου που έχουν κατά καιρούς αναπτυχθεί και χρησιμοποιούνται. Δεν είναι επίσης σπάνιο το φαινόμενο χρησιμοποίησης περισσοτέρων από μίας μεθόδων, ανάλογα με την περιοχή του λογισμικού που ελέγχεται κάθε φορά.

Ανάλογα με τη μέθοδο ελέγχου ή τον συνδυασμό μεθόδων που επιλέγεται να ακολουθηθεί σε κάθε περίπτωση επηρεάζεται και η αποτελεσματικότητα του ελέγχου ως προς την εύρεση σφαλμάτων στο λογισμικό και κατά συνέπεια και η επικινδυνότητα που απομένει μετά την ολοκλήρωσή του.

3.2.3.1 Κατηγορίες Μεθόδων Ελέγχου

Ένας πρώτος διαχωρισμός των μεθόδων 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)* [Cowa88].

Ένας δεύτερος διαχωρισμός βασίζεται στο αν υπάρχει ενδιαφέρον για την εσωτερική δομή του προγράμματος οδηγώντας σε δύο βασικές κατηγορίες: *White-box testing* και *Black-box testing*.

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.

3.2.3.2 Μέτρα Κάλυψης Κώδικα

Τα μέτρα κάλυψης του κώδικα, μπορεί να θεωρηθεί ότι αποτελούν ένα μέτρο πληρότητας του ελέγχου που διεξήχθη στο λογισμικό. Από την άλλη, η επικινδυνότητα του λογισμικού που απομένει μετά το πέρας της φάσης του ελέγχου, σχετίζεται άμεσα και με το πόσο πλήρης είναι ο έλεγχος στον οποίο αυτό έχει υποβληθεί. Γι’ αυτό το λόγο θεωρήθηκε σκόπιμο να αναφερθούν τα σημαντικότερα εξ' αυτών των μέτρων και να παρουσιαστεί μία σύντομη σύγκρισή τους ως προς την πληρότητα ελέγχου που αυτά προσφέρουν.

Πιο συγκεκριμένα, κάποια από τα κυριότερα μέτρα κάλυψης κώδικα, τα οποία στοχεύουν στο να ποσοτικοποιήσουν την αποτελεσματικότητα του ελέγχου του λογισμικού είναι [Τζαβ98]:

1) Κάλυψη εντολών (Statement Coverage)

2) Κάλυψη κλάδων-αποφάσεων (Branch/Decision Coverage)

3) Κάλυψη γραμμικών ακολουθιών κώδικα και αλμάτων (Linear Code Sequence And Jump Coverage – LCSAJ Coverage)
4) Κάλυψη υποσυνθηκών (Subcondition Coverage – Tsub (C))

5) Κάλυψη κλήσεων διαδικασιών/συναρτήσεων (Procedure/Function Call Coverage – P/F Call Coverage)

6) Κάλυψη συνθηκών διακλάδωσης (Branch Condition Coverage – BCC)

7) Κάλυψη συνδυασμών συνθηκών διακλάδωσης (Branch Condition Combination Coverage – BCCC)

8) Τροποποιημένη κάλυψη συνθηκών-αποφάσεων (Modified Condition/Decision Coverage – MC/DC)

9) Κάλυψη ροής δεδομένων (Dataflow Coverage)

Ακολούθως παρουσιάζεται η πληρότητα που προσφέρει καθ' ένα από τα παραπάνω μέτρα κάλυψης κώδικα :

- Η σχέση για την κάλυψη εντολών είναι : Statement Coverage $= \frac{s}{S}$

  όπου s = ο αριθμός των εντολών που εκτελούνται τουλάχιστον μια φορά
  και S = ο συνολικός αριθμός των εκτελούμενων εντολών.

  Οπότε κάλυψη εντολών 100% (TER1 = 100%) σημαίνει ότι
  - κάθε statement του κώδικα θα έχει εκτελεστεί.

- Για την κάλυψη κλάδων-αποφάσεων ισχύει ο τύπος:

  Decision Coverage $= \frac{d}{D}$

  όπου d = ο αριθμός των κλάδων-αποφάσεων που έχει εκτελεστεί τουλάχιστον μία φορά και D = ο συνολικός αριθμός των κλάδων-αποφάσεων.

  Κατά συνέπεια κάλυψη κλάδων-αποφάσεων 100% (TER2 = 100%) σημαίνει ότι
  - κάθε statement του κώδικα θα έχει εκτελεστεί

- κάθε κλάδος μεταξύ statements (και συνεπώς και κάθε απόφαση) θα έχει εκτελεστεί.

- Όσον αφορά τη κάλυψη LCSAJ, ισχύει η σχέση : LCSAJ Coverage = $\frac{l}{L}$

  όπου l = ο αριθμός των LCSAJs που εκτελούνται τουλάχιστον μία φορά
  και L = ο συνολικός αριθμός των LCSAJs.

  άρα, κάλυψη LCSAJ 100% (TER3 = 100%) σημαίνει ότι
  - κάθε statement του κώδικα θα έχει statement του κώδικα θα έχει εκτελεστεί
  - κάθε κλάδος απόφασης θα έχει εκτελεστεί
  - θα έχουν εκτελεστεί όλοι οι συνδυασμοί των φωλιασμένων (nested) και των εν σειρά ανακυκλώσεων
  - θα έχουν εκτελεστεί όλοι οι συνδυασμοί των αποφάσεων

- Κάλυψη υποσυνθηκών 100% (Tsub = 100%) σημαίνει ότι
  - κάθε λογική υποπαράσταση μιας συνθήκης θα έχει πάρει και την τιμή TRUE και την τιμή FALSE

- Κάλυψη κλήσεων διαδικασιών/συναρτήσεων 100% (P/F CALL = 100%) σημαίνει ότι
  - θα έχει εκτελεστεί κάθε κλήση συνάρτησης/διαδικασίας

- Κάλυψη συνθηκών διακλάδωσης 100% (BCC = 100%) σημαίνει ότι
  - οι λογικοί τελεστές κάθε συνθήκης θα έχουν πάρει και την τιμή TRUE και την τιμή FALSE

- Κάλυψη συνδυασμών συνθηκών διακλάδωσης 100% (BCCC = 100%) σημαίνει ότι
  - θα έχουν εκτελεστεί όλοι οι συνδυασμοί λογικών τιμών των υποσυνθηκών της κάθε συνθήκης

- Τροποποιημένη κάλυψη συνθηκών-αποφάσεων 100% (MD/DC = 100%) σημαίνει ότι
  - θα έχει εκτελεστεί ένα υποσύνολο των συνδυασμών λογικών τιμών των υποσυνθηκών κάθε συνθήκης, τέτοιο ώστε να επιδεικνύεται η επιρροή του κάθε λογικού τελεστή ξεχωριστά στο αποτέλεσμα της συνθήκης

- Κάλυψη p-use (predicate use) και c-use (computational use) 100% (κάλυψη Dataflow) σημαίνει ότι
  - θα έχει εκτελεστεί κάθε δυνατός συνδυασμός ορισμού τιμής και χρήσης για όλες τις μεταβλητές

Από τα παραπάνω γίνεται φανερό πως η κάλυψη LCSAJ εξασφαλίζει τον πληρέστερο έλεγχο του πηγαίου κώδικα και της δομής του, χωρίς βέβαια αυτό να σημαίνει ότι είναι και η πλέον κατάλληλη για όλες τις περιπτώσεις. Μπορεί να χρησιμοποιούνται κάθε φορά ένα ή και περισσότερα μέτρα κάλυψης με απώτερο σκοπό να επιτυγχάνεται αποδοτικότερη ανακάλυψη λαθών και κατά συνέπεια χαμηλότερο επίπεδο επικινδυνότητας.

Αυτό όμως, που πρέπει να γίνει κατανοητό, είναι ότι επειδή δεν είναι πάντα εφικτή η επίτευξη 100% κάλυψης όλου του κώδικα, είτε λόγω χρόνου, είτε λόγω άλλων αιτιών όπως για παράδειγμα η μη προσπελασιμότητα ορισμένων μονοπατιών, ο στόχος που πρέπει να τίθεται και να επιδιώκεται είναι η επίτευξη όσο το δυνατόν υψηλότερου ποσοστού κάλυψης στις περιοχές που παρουσιάζουν και την υψηλότερη επικινδυνότητα.

Βέβαια όλα τα παραπάνω αφορούν συμβατικά συστήματα λογισμικού και δεν συμπεριλαμβάνουν το είδος των αντικειμενοστραφών συστημάτων. Επειδή λοιπόν θα αποτελούσε σίγουρα σημαντική παράληψη, η μη αναφορά σε συστήματα που αναπτύσσονται σύμφωνα με την αντικειμενοστραφή τεχνολογία, σε παράγραφο που ακολουθεί (Παρ. 3.3), πραγματοποιείται μία συνοπτική αναφορά στον έλεγχο που εφαρμόζεται σε τέτοιου είδους συστήματα λογισμικού.

Ως ένα εναλλακτικό μέτρο της πληρότητας του ελέγχου θα μπορούσε να θεωρηθεί η έκταση του ελέγχου (testing effort) με μονάδα μέτρησης το χρόνο διάρκειάς του. όμως, ο χρόνος διάρκειας του ελέγχου δεν μπορεί να αποτελέσει καθοριστικό παράγοντα εκτίμησης της πληρότητας του. Γιατί βέβαια, υπάρχει η πιθανότητα διεξαγωγής εκτενούς ελέγχου, χωρίς να επιτυγχάνεται τελικά ικανοποιητική κάλυψη, λόγω κάποιων λανθασμένων επιλογών που έχουν παρθεί. Με άλλα λόγια, δεν ισχύει σε καμία περίπτωση ο ισχυρισμός ότι όσο μεγαλύτερος είναι ο χρόνος διάρκειας ελέγχου ενός λογισμικού, τόσο μικρότερη είναι και η πιθανότητα αποτυχίας και κατ' επέκταση και η εναπομείνασα επικινδυνότητα του.

3.2.4 Εμπειρία Ομάδας Ελέγχου (TEXP)

Δεν πρέπει να λησμονείται το γεγονός πως η αποδοτικότητα του ελέγχου και κατ' επέκταση και το επίπεδο επικινδυνότητας που απομένει μετά την ολοκλήρωσή του, εξαρτάται και από την ικανότητα και εμπειρία της ομάδας που διεξάγει τον έλεγχο. 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 3.4: *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.

3.2.5 Πολυπλοκότητα Λογισμικού (CPLX)

Η πολυπλοκότητα που έχει το λογισμικό αποτελεί έναν ακόμα παράγοντα που επηρεάζει την επικινδυνότητα που το χαρακτηρίζει. Γιατί βέβαια, όσο υψηλότερη είναι η πολυπλοκότητα του λογισμικού, τόσο μεγαλύτερη είναι και η πιθανότητα εμφάνισης μιας αποτυχίας, λόγω μη εύρεσης κάποιων σφαλμάτων κατά τη διάρκεια του ελέγχου.

Το επίπεδο πολυπλοκότητας του λογισμικού μπορεί να καθοριστεί με βάση ορισμένες κατηγορίες λειτουργιών του και πιο συγκεκριμένα βάσει των λειτουργιών ελέγχου, υπολογισμού, εξαρτώμενων από συσκευές, διαχείρισης δεδομένων και διαχείρισης διεπαφής χρήστη. Η κατάταξη αυτή φαίνεται στον πίνακα που ακολουθεί:

Κατάταξη	Λειτουργίες Λογισμικού				
	Λειτουργίες Ελέγχου	Λειτουργίες Υπολογισμού	Λειτουργίες Εξαρτημένες Από Συσκευές	Λειτουργίες Διαχείρισης Δεδομένων	Λειτουργίες Διαχείρισης Διεπαφής Χρήστη
Πολύ Χαμηλή	Ευθύς κώδικας με λίγους φωλιασμέ-νους τελεστές του δομημένου προγραμματισμού: DO, CASE, IF THEN ELSE, απλά κατηγορήματα.	Αποτίμηση απλών εκφράσεων π.χ. A= B*C+(D-E)	Απλές εντολές READ, WRITE με απλό FORMAT	Απλά ARRAYS στην κύρια μνήμη. Απλές COTS-DB ερωτήσεις, ενημερώσεις.	Απλές φόρμες εισόδου, γεννήτορες αναφορών (reports).
Χαμηλή	Ομαλό φώλιασμα τελεστών του δομημένου προγραμματισμού, κυρίως απλά κατηγορήματα.	Αποτίμηση μετρίου επιπέδου εκφράσεων π.χ. D=SQRT(B**2-4*A*C)	Δεν απαιτείται γνώση των χαρακτηριστικών του ιδιαίτερου επεξεργαστή ή συσκευής εισόδου/ εξόδου. Λειτουργίες εισόδου/εξόδου γίνονται στο επίπεδο GET/PUT. Δεν απαιτείται γνώση υπερκαλύψεων.	Απλά αρχεία χωρίς αλλαγές στις δομές δεδομένων, χωρίς έκδοση, χωρίς ενδιάμεσα αρχεία. Μέτριας πολυπλοκότητας COTS-DB ερωτήσεις, ενημερώσεις.	Χρήση απλών builders γραφικών διεπαφής χρήστη (GUI).
Ονομαστική	Κυρίως απλό φώλιασμα. Λίγος έλεγχος μεταξύ δομικών λίθων. Πίνακες αποφάσεων.	Χρησιμοποίηση προτύπων μαθηματικών και στατιστικών ρουτινών. Βασικές λειτουργίες μητρών.	Επεξεργασία εισόδου/εξόδου συμπεριλαμβάνει επιλογή συσκευής, έλεγχο κατάστασης και λαθών.	Πολλαπλά αρχεία εισόδου και ένα αρχείο εξόδου. Απλές δομικές αλλαγές, απλές εκδόσεις. Πολύπλο-κες COTS-DB ερωτήσεις, ενημερώσεις.	Απλή χρήση widget set.
Υψηλή	Τελεστές δομημένου προγραμματισμού πολύ φωλιασμένοι με πολλά κατηγορήματα έλεγχος ουράς και στοίβας. Σημαντικός έλεγχος μεταξύ μονάδων προγράμματος.	Βασική αριθμη-τική ανάλυση: παρεμβολή, κοινές διαφορικές εξισώσεις. Φρο-ντίδα αποκοπής και στρογγυλο-ποίησης.	Λειτουργίες στο φυσικό επίπεδο εισόδου/εξόδου. Αριστοποίηση επικαλύψεων εισόδου/εξόδου.	Ειδικού σκοπού ρουτίνες δραστηριοποιούμενες από τα περιεχόμενα Καναλιών δεδομένων. Πολύπλοκες λειτουργίες σε επίπεδο εγγραφής.	Ανάπτυξη και επέκταση widget set. Απλά φωνητικά πολυμέσα εισόδου/ εξόδου
Πολύ Υψηλή	Επαναλήψιμος και αναδρομικός κώδικας. Σταθερής προτεραιότητας διακοπές.	Δύσκολες αλλά δομημένες λειτουρ-γίες αριθμητικής ανάλυσης: γραμ-μικές εξισώσεις με μερικές παραγώγους	Ρουτίνες για διάγνωση, εξυπηρέτηση και απομόνωση διακοπών. Εξυπηρέτηση γραμμών επικοινωνίας.	Γενική, διευθυνόμενη από παραμέτρους, ρουτίνα για δόμηση αρχείου. Κτίσιμο αρχείου, επεξεργασία προσταγών.	Μέτριας πολυπλοκό-τητας δυναμ-ικά γραφικά δύο και τριών διαστάσεων και πολυμέσα.

Εξαιρετικά Υψηλή	Χρονοπρογραμ-ματισμός πολλών πόρων με προτεραιότητες που αλλάζουν δυναμικά. έλεγχος σε επίπεδο μικροκώδικα.	Δύσκολες και αδόμητες λειτουργίες αριθμητικής ανάλυσης: πολύ υψηλής ακρίβειας ανάλυση θορύβου, στοχαστικών δεδομένων.	Κώδικας εξαρτημένος από τη συσκευή και από το χρόνο, λειτουργίες μικροπρογραμ-ματιζόμενες.	Πολύ συζευγμένες δυναμικές σχεσιακές δομές. Διαχείριση δεδομένων φυσικής γλώσσας.	Πολύπλοκα πολυμέσα, εικονική πραγματικό-τητα

Πίνακας 3.5*: Κατάταξη του παράγοντα " πολυπλοκότητα" με βάση πέντε κατηγορίες λειτουργιών του λογισμικού*[1].

3.2.6 Συχνότητα Χρήσης (FREQ)

Η συχνότητα χρήσης αποτελεί έναν ακόμα σημαντικό παράγοντα που επηρεάζει την επικινδυνότητα του λογισμικού. Και αυτό διότι, όσο πιο συχνά χρησιμοποιείται μια εφαρμογή ή ένα προϊόν, τόσο περισσότερο αυξάνεται και η πιθανότητα να αναδειχθεί ένα σφάλμα του λογισμικού και κατ' επέκταση να προκύψει ένα μη επιθυμητό αποτέλεσμα.

Έτσι, για παράδειγμα μια εφαρμογή η οποία πρόκειται να χρησιμοποιηθεί μία φορά στη διάρκεια ενός έτους (και φυσικά όχι για κάτι τόσο σημαντικό και κρίσιμο, όσο η εκτόξευση ενός διαστημοπλοίου), χαρακτηρίζεται αναμφισβήτητα από χαμηλότερο επίπεδο επικινδυνότητας σε σχέση με κάποια που χρησιμοποιείται σε καθημερινή βάση (όπως για παράδειγμα ένα σύστημα κρατήσεων αεροπορικών θέσεων).

Όμως δεν υπάρχει διαφοροποίηση ως προς τη συχνότητα χρήσης μόνο ανάμεσα σε διαφορετικά συστήματα λογισμικού, αλλά και μεταξύ των τμημάτων και του ίδιου του λογισμικού. Δηλαδή, υπάρχουν περιοχές οι οποίες είναι πιθανό να χρησιμοποιούνται πιο συχνά από άλλες (πράγμα που αποτυπώνεται και στο σχήμα 2.1 του προηγούμενου κεφαλαίου), όπως υπάρχουν και στον κώδικα μονοπάτια των οποίων η προσπέλαση είναι συχνότερη από κάποιων άλλων.

Σαν παράδειγμα στα όσα προαναφέρθηκαν παρατίθεται το πιο κάτω πρόγραμμα δημιουργίας τριγώνου το οποίο είναι χωρισμένο σε βασικές ενότητες. Εφόσον τα δεδομένα που δίνονται στην είσοδο είναι τυχαία (δηλαδή τυχαία νούμερα), η πιθανότητα δημιουργίας ενός σκαληνού τριγώνου είναι σαφώς μεγαλύτερη από αυτή ενός ισοσκελούς και ακόμη πιο μεγάλη από αυτή ενός ισοπλεύρου. Αυτό έχει ως αποτέλεσμα, κάποιες εντολές που ανήκουν σε ορισμένες βασικές ενότητες (basic blocks) να εκτελούνται λιγότερο συχνά από κάποιες άλλες.

[1] Ο πίνακας της κατάταξης πολυπλοκότητας βασίζεται στην τεχνική COCOMO (Constructive Cost Model), που χρησιμοποιείται για τον υπολογισμό κόστους έργων λογισμικού.

```
 1 PROGRAM TRIANGLE (INPUT,OUTPUT);
 2 VAR I,J,K,MATCH : INTEGER;
 3   BEGIN                                                          1
 4     WHILE
 5         NOT EOF (INPUT)
 6     DO                                                           2
 7      BEGIN
 8        READLN (I,J,K);
 9        WRITELN (I,J,K);
10        IF
11            (I > = J + K) OR (J > = K + I) OR (K > = I + J)
12        THEN                                                     3
13         BEGIN
14            WRITELN ("NOT A TRIANGLE")
15         END
16        ELSE                                                     4
17         BEGIN
18            MATCH : = 0;
19            IF
20                I = J
21            THEN                                                 5
22             BEGIN
23                MATCH : = MATCH + 1
24             END;                                                6
25            IF
26                J = K
27            THEN                                                 7
28             BEGIN
29                MATCH : = MATCH + 1
30             END;                                                8
31            IF
32               K = I
33            THEN                                                 9
34             BEGIN
35                MATCH = MATCH + 1
36             END;                                               10
37            IF
38                MATCH = 0
39            THEN                                                11
40             BEGIN
41                WRITELN ("SCALENE TRIANGLE")
42             END
43            ELSE                                                12
44             BEGIN
45                IF
```

```
46                      MATCH = 1
47                  THEN                                    13
48                  BEGIN
49                      WRITELN ("ISOSCELES TRIANGLE")
50                  END
51                  ELSE                                    14
52                  BEGIN
53                      WRITELN ("EQUILATERAL TRIANGLE")
54                  END                                     15
55              END                                         16
56          END                                             17
57      END                                                 18
58  END.                                                    19
```

Στο παραπάνω πρόγραμμα, για παράδειγμα, υπάρχει μεγαλύτερη πιθανότητα να εκτελεστούν οι εντολές της βασικής ενότητας 12 από ότι αυτές των ενοτήτων 14 και 15, πράγμα που σημαίνει ότι έχουν και μικρότερη πιθανότητα εμφάνισης λάθους και κατ' επέκταση χαμηλότερη επικινδυνότητα.

Όλα όσα προαναφέρθηκαν είναι σημαντικό να λαμβάνονται υπόψη για τη λήψη αποφάσεων σχετικών με τον έλεγχο. Έτσι, σε περίπτωση που δεν επαρκεί για παράδειγμα ο χρόνος, ή δεν υπάρχουν οι απαιτούμενοι πόροι για έλεγχο όλων των τμημάτων του λογισμικού, θα μπορέσει να δοθεί προτεραιότητα στα μέρη εκείνα με τη μεγαλύτερη συχνότητα χρήσης.

3.2.7 Χρόνος Ελέγχου (TIME)

Ως γνωστόν, για κάθε σύστημα λογισμικού θέτονται κάποια χρονικά όρια μέσα στα οποία πρέπει να ολοκληρωθεί η ανάπτυξή του, ώστε να παραδοθεί έγκαιρα το όποιο προϊόν ή εφαρμογή. Ο διαθέσιμος αυτός χρόνος, όπως είναι φυσικό, μοιράζεται κατάλληλα σε όλες τις φάσεις του κύκλου ζωής ανάπτυξης του λογισμικού συμπεριλαμβανομένης φυσικά και της φάσης του ελέγχου.

Αυτοί ακριβώς οι περιορισμοί ως προς τα χρονικά περιθώρια διεξαγωγής του ελέγχου, δεν είναι δυνατό να μην επηρεάζουν και την επικινδυνότητα που απομένει μετά το πέρας του. Από τη μια πλευρά, θα μπορούσε να διατυπωθεί η άποψη ότι όσο πιο πιεστικά είναι τα περιθώρια τα οποία πρέπει να τηρηθούν, τόσο μεγαλύτερη είναι και η πιθανότητα να μην λάβει χώρα ο πλέον κατάλληλος και αποδοτικός έλεγχος, με αποτέλεσμα να αυξάνεται το ποσοστό της εναπομείνασας επικινδυνότητας. Από την άλλη όμως κάτι τέτοιο δεν θα ήταν εντελώς σωστό, καθότι, όπως αναφέρθηκε και προηγουμένως, ο χρόνος που διαρκεί ο έλεγχος δεν μπορεί να αποτελέσει απόδειξη

για το πόσο πλήρης αυτός ήταν. Όπως και να έχει όμως ο περιορισμός του χρόνου του ελέγχου είναι κάτι που πρέπει απαραίτητα να λαμβάνεται υπόψη από την ομάδα ελέγχου και που με τον ένα ή τον άλλο τρόπο επηρεάζει την όλη διαδικασία και κατ' επέκταση και το αποτέλεσμα αυτής.

3.2.8 Διαθέσιμοι Πόροι (RESO)

Ένας άλλος, εξίσου σημαντικός με τον χρόνο παράγοντας είναι και οι πόροι που διατίθενται κάθε φορά προκειμένου να πραγματοποιηθεί η ανάπτυξη ενός συστήματος λογισμικού. Γιατί βέβαια, δεν μπορούν να υπάρχουν αστείρευτες οικονομικές πηγές οι οποίες θα χρηματοδοτούν το έργο ανάπτυξης, με αποτέλεσμα σε κάποια χρονική στιγμή να μην παραχωρούνται πλέον άλλοι πόροι για την συνέχιση του έργου.

Έτσι, εκτός των χρονικών περιορισμών, υφίστανται και οικονομικοί περιορισμοί, οι οποίοι πολλές φορές δεν επιτρέπουν στην ομάδα ελέγχου να ενεργήσει όπως θα επιθυμούσε, με αποτέλεσμα ο έλεγχος που διεξάγεται να μην είναι ο πλέον αποτελεσματικός.

Οι τελευταίοι επτά παράγοντες που παρουσιάστηκαν πιο πάνω, μπορούμε να θεωρήσουμε ότι επηρεάζουν την πιθανότητα του να λάβει χώρα ένα ανεπιθύμητο αποτέλεσμα, δηλαδή μια αποτυχία του συστήματος λογισμικού. Υπό αυτή την προοπτική προκύπτει η σχέση:

$$P(UO) = f(SIZE, TEXP, TESM, CPLX, FREQ, TIME, RESO) \quad (2)$$

όπου :

P(UO) : η πιθανότητα να συμβεί ένα μη επιθυμητό αποτέλεσμα
(probability of an unsatisfactory outcome)

Βάσει των παραγόντων που προαναφέρθηκαν, για την επικινδυνότητα που απομένει μετά την ολοκλήρωση του ελέγχου του λογισμικού θα ισχύει η παρακάτω σχέση:

$$R = P(UO) * CRIT \quad (3)$$

όπου :

R : η επικινδυνότητα του λογισμικού

και

CRIT όπως ορίσθηκε από τη σχέση (1)
P(UO) όπως ορίσθηκε από τη σχέση (2)

3.3 Έλεγχος Αντικειμενοστραφούς Λογισμικού

Είναι σίγουρο πώς υπάρχουν αρκετές ομοιότητες στον έλεγχο των συμβατικών και των αντικειμενοστραφών συστημάτων λογισμικού. Όμως, αδιαμφισβήτητα τα αντικειμενοστραφή συστήματα εισάγουν κάποιες νέες μορφές προβλημάτων, όπως για παράδειγμα:

- Μέχρι ποιου σημείου θα πρέπει να ελέγχονται ξανά τα στοιχεία που κληρονομούνται;
- Πότε μπορούμε να εμπιστευτούμε τα reports για την κατάσταση ενός μη ελεγμένου αντικειμένου;

- Πως μπορούμε να επαληθεύσουμε μια εξαρτώμενη από κατάσταση συμπεριφορά (state-dependent behavior), όταν ο έλεγχος κατάστασης κατανέμεται σε μία ολόκληρη αντικειμενοστραφή εφαρμογή (OOA);

Με σκοπό να δοθεί μια όσο το δυνατό πληρέστερη εικόνα των θεμάτων που αφορούν τον έλεγχο των αντικειμενοστραφών συστημάτων λογισμικού, ακολουθεί μία σύντομη παρουσίαση των κυριότερων εξ' αυτών:

3.3.1 Βασική μονάδα για έλεγχο (Basic unit for testing)

Υπάρχει μια κοινή συμφωνία σχετικά με το ότι η κλάση (class) αποτελεί τη φυσική μονάδα ελέγχου. Άλλες μονάδες για έλεγχο αποτελούν τα σύνολα κλάσεων (π.χ. class clusters). Ανάλογα με τη χρήση της κλάσης ποικίλουν και οι απαιτήσεις ελέγχου.

3.3.2 Κληρονομικότητα (Inheritance)

Είναι γενικώς αποδεκτό πως στις περισσότερες των περιπτώσεων τα χαρακτηριστικά που κληρονομούνται απαιτούν επανεξέταση. Και αυτό διότι προκύπτει ένα νέο πλαίσιο χρήσης όταν κληρονομούνται κάποια χαρακτηριστικά. Επίσης, όπως είναι αναμενόμενο, η πολλαπλή κληρονομικότητα αυξάνει τον αριθμό αυτών που πρέπει να ελεγχθούν.

3.3.3 Ενθυλάκωση (Encapsulation)

Η ενθυλάκωση δεν αποτελεί πηγή λαθών, αλλά μπορεί να γίνει εμπόδιο για την διαδικασία του ελέγχου. Ο έλεγχος απαιτεί αναφορά σε όλες τις καταστάσεις του αντικειμένου (concrete & abstract state) και η ενθυλάκωση, παρόλα τα πλεονεκτήματα που προσφέρει, είναι πολύ πιθανό να μην επιτρέπει κάτι τέτοιο.

Σε αυτή την περίπτωση, μπορούν να βοηθήσουν οι τεχνικές απόδειξης της ορθότητας (proof-of-correctness). Βέβαια, η τυπική απόδειξη της ορθότητας είναι ισοδύναμη με τον εξαντλητικό έλεγχο, με αποτέλεσμα να είναι αρκετά δύσκολο και να επιτευχθεί και υπερβολικά χρονοβόρο.

3.3.4 Πολυμορφισμός (Polymorphism)

Κάθε πιθανή σύνδεση ενός πολυμορφικού τμήματος απαιτεί ξεχωριστό έλεγχο. Είναι πιθανόν πολύ δύσκολο να βρεθούν όλες οι αυτές οι συνδέσεις, πράγμα που αυξάνει την πιθανότητα εμφάνισης λαθών και βάζει εμπόδια στην επίτευξη των στόχων κάλυψης που έχουν τεθεί.

3.3.5 Έλεγχος White-Box

Ο έλεγχος white-box, όπως έχει προαναφερθεί σημαίνει ότι εξετάζεται ο πηγαίος κώδικας, σύμφωνα με κάποιες τεχνικές που επιλέγονται. Επικρατεί η κοινή άποψη ότι οι συμβατικές προσεγγίσεις αυτού του τύπου δεν μπορούν να υιοθετηθούν στην περίπτωση των αντικειμενοστραφών συστημάτων, διότι δεν είναι κατάλληλες για έλεγχο κλάσεων.

Τα προβλήματα της χρήσης white-box μόνο μεθόδων ελέγχου, είναι πολλά και προκύπτουν κυρίως από τη ταυτολογική, κατά βάση, φύση αυτής της προσέγγισης καθώς και την πληροφοριακή ανεπάρκεια του πηγαίου κώδικα για μία κατάλληλη διαδικασία ελέγχου.

3.3.6 Έλεγχος Black-Box

Σε αυτή την περίπτωση ως γνωστόν χρησιμοποιούνται απαιτήσεις ή προδιαγραφές για να αποφασιστεί το κατά πόσο ικανοποιούνται από το λογισμικό. Υπάρχει μία γενική συμφωνία σχετικά με το ότι οι συμβατικές black-box μέθοδοι θα ήταν αρκετά χρήσιμες για αντικειμενοστραφή συστήματα. Παρόλα αυτά, δεν έχουν αναπτυχθεί ιδιαίτερα συστηματικές τεχνικές σε αυτή τη κατεύθυνση.

3.3.7 State-Based Έλεγχος

Αυτός ο τύπος ελέγχου αντλεί test cases μοντελοποιώντας μία κλάση ως μηχανή κατάστασης. Οι μέθοδοι καταλήγουν σε μεταβάσεις καταστάσεων. Το μοντέλο καταστάσεων ορίζει επιτρεπτές ακολουθίες μεταβάσεων. Για παράδειγμα, αυτό σημαίνει ότι ένα στιγμιότυπο πρέπει να δημιουργηθεί προτού μπορεί να ενημερωθεί ή διαγραφεί.

3.3.8 Επάρκεια και Κάλυψη (Adequacy and Coverage)

Ο σκοπός της επανεξέτασης κάτω από κληρονομικότητα είναι κοινώς αποδεκτός. Αυτό ορίζει ένα γενικό σκοπό ελέγχου, αλλά δεν υποστηρίζει μια συγκεκριμένη κάλυψη. Οι περισσότερες προσεγγίσεις απαιτούν λεπτομερή έλεγχο.

3.3.9 Στρατηγικές Ενοποίησης (Integration Strategies)

Η ενοποίηση των κλάσεων για τη δημιουργία ενός συστήματος εφαρμογών πρέπει να συνδέεται στενά με τη συνολική διαδικασία ανάπτυξης. Υπάρχουν δύο βασικές στρατηγικές: thread-based και uses-based. Ένα thread αποτελείται από όλες τις κλάσεις που χρειάζεται να αντιδρούν σε ένα μοναδικό εξωτερικό input. Κάθε κλάση ελέγχεται ως μονάδα και κατόπιν εφαρμόζεται ένα σύνολο από thread. Η uses-based ενοποίηση αρχίζει με τον έλεγχο των κλάσεων που χρησιμοποιούν λίγες ή καθόλου κλάσεις εξυπηρετητή (server classses). Μετά, ελέγχονται οι κλάσεις που χρησιμοποιούν την πρώτη ομάδα κλάσεων, στη συνέχεια οι κλάσεις που χρησιμοποιούν τη δεύτερη ομάδα κ.ο.κ.

3.3.10 Στρατηγική Διαδικασίας Ελέγχου (Test Process Strategy)

Η αντικειμενοστραφής ανάπτυξη τείνει προς μικρότερους κύκλους. Η όλη διαδικασία μπορεί να χαρακτηριστεί ως λίγος σχεδιασμός, λίγη κωδικοποίηση και λίγος έλεγχος. Στα ζητήματα των διαδικασιών περιλαμβάνονται το πότε θα γίνει test case σχεδιασμός, πότε επανεξέταση των χαρακτηριστικών που έχουν κληρονομηθεί και πότε θα ακολουθηθεί ενοποιητική στρατηγική.

Κατά καιρούς έχουν προταθεί διάφορες προσεγγίσεις που αφορούν τον τρόπο διεξαγωγής του ελέγχου στα αντικειμενοστραφή συστήματα λογισμικού, οι οποίες πολλές φορές υποστηρίζουν εντελώς διαφορετικές και αλληλοσυγκρουόμενες οδούς. Το ποια από όλες ή ποιος συνδυασμός αυτών είναι κατάλληλο να χρησιμοποιηθεί εξαρτάται από παράγοντες παρόμοιους με αυτούς που ισχύουν και στις περιπτώσεις των συμβατικών συστημάτων.

3.3.11 Συμπεράσματα για τον Έλεγχο και την Επικινδυνότητα των Αντικειμενοστραφών Συστημάτων Λογισμικού

Από τα όσα αναφέρθηκαν μπορεί να προκύψει το συμπέρασμα ότι ενώ οι αντικειμενοστραφείς γλώσσες προγραμματισμού πιθανόν να μειώνουν την πιθανότητα εμφάνισης ορισμένων τύπων λαθών, αυξάνουν την πιθανότητα εμφάνισης κάποιων άλλων. Οι μέθοδοι τείνουν να είναι μικρές με χαμηλή αλγοριθμική

πολυπλοκότητα, με αποτέλεσμα να είναι λιγότερο πιθανά τα σφάλματα μονοπατιών. Η ενθυλάκωση προλαμβάνει μεν μερικά από τα προβλήματα τα οποία δημιουργούνται, αλλά συγχρόνως δημιουργεί κάποια νέα. Επίσης, λάθη κώδικα έχουν την ίδια πιθανότητα να συμβούν όπως και στα συμβατικά συστήματα λογισμικού, ενώ δυστυχώς μερικά από τα σημαντικά χαρακτηριστικά των αντικειμενοστραφών γλωσσών δημιουργούν νέες μορφές σφαλμάτων.

Βάσει όλων αυτών, γίνεται φανερό ότι ο έλεγχος του αντικειμενοστραφούς λογισμικού είανι ακόμα πιο σημαντικός για την παραγωγή υψηλής ποιότητας συστημάρων σε σύγκριση με τις συμβατικές υλοποιήσεις. Στα συμβατικά συστήματα, η στατική επαλήθευση (static verification) μπορεί να αποδειχθεί εξαιρετικά αποτελεσματική για την εύρεση και καταπολέμηση σφαλμάτων. Κάτι τέτοιο δεν ισχύει για τον πηγαίο κώδικα του αντικειμενοστραφούς λογισμικού, καθότι είναι αρκετά δυσκολότερο να διαβαστεί σε σχέση με τον καλά δομημένο συμβατικό κώδικα, λόγω ύπαρξης όλων αυτών των χαρακτηριστικών που αναφέρθησαν νωρίτερα. Αυτό έχει ως αποτέλεσμα να απαιτείται παρισσότερος έλεγχος ώστε να επιτευχθεί το όποιο επιθυμητό επίπεδο ποιότητας.

ΚΕΦΑΛΑΙΟ 4

4.1 Εισαγωγικά

Ο καθορισμός της εναπομείνασας επικινδυνότητας, ο οποίος γίνεται λαμβάνοντας υπόψη τους παράγοντες που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, δεν θα μπορούσε να αποδώσει χωρίς την ύπαρξη μιας πολιτικής από την πλευρά των υπεύθυνων για την ανάπτυξη του λογισμικού. Γιατί σε μια τέτοια περίπτωση δε θα ήταν δυνατή η ορθή αξιολόγηση του επιπέδου επικινδυνότητας που απομένει μετά το πέρας του ελέγχου, πράγμα που επηρεάζει φυσικά και την απόφαση για διεξαγωγή ή μη διεξαγωγή περαιτέρω ελέγχου.

Στόχος αυτού του κεφαλαίου, είναι να τονίσει αυτήν ακριβώς την ανάγκη ύπαρξης πολιτικής, αλλά και να παρουσιάσει τη γενικότερη διαδικασία διαχείρισης επικινδυνότητας σε σχέση πάντα με τον έλεγχο του λογισμικού.

4.2 Καθορισμός Πολιτικής σε σχέση με το Επιτρεπτό Επίπεδο Επικινδυνότητας

Υπό την παραδοχή ότι μετά την ολοκλήρωση του ελέγχου του λογισμικού, απομένει σίγουρα κάποιο ποσοστό κινδύνου, προβάλλει επιτακτική η ανάγκη καθορισμού εκ των προτέρων μιας πολιτικής η οποία θα θέτει τα επιτρεπτά όρια μέσα στα οποία θα πρέπει να κυμαίνεται η εναπομείνασα επικινδυνότητα.

Με αυτό το τρόπο, θέτονται συγκεκριμένοι στόχοι, πάνω στους οποίους θα μπορέσουν να στηριχθούν και οι διάφορες αποφάσεις ελέγχου, όπως το ποια μέθοδος ή συνδυασμός μεθόδων θα ακολουθηθεί ή το πότε αυτός πρέπει να σταματήσει.

Τα όρια επικινδυνότητας που πρέπει να τηρούνται γενικότερα καθορίζονται λαμβάνοντας υπόψη παραμέτρους όπως ο σκοπός για τον οποίο πρόκειται να χρησιμοποιηθεί το προϊόν λογισμικού, ποιος πρόκειται να το χρησιμοποιήσει, τι απαιτήσεις έχει από αυτό και ποιες θα είναι οι επιπτώσεις που θα επέλθουν εξ' αιτίας μιας πιθανής δυσλειτουργίας του.

Όμως δεν πρέπει να λησμονείται και ο παράγοντας κόστος, ο οποίος παίζει σημαντικό ρόλο στη χάραξη μιας πολιτικής από την πλευρά αυτών που αναπτύσσουν το σύστημα λογισμικού. Γιατί βέβαια, ποτέ δεν υπάρχουν ανεξάντλητοι οικονομικοί πόροι που να επιτρέπουν την διεξαγωγή ελέγχου για την επίτευξη του ελάχιστου δυνατού επιπέδου επικινδυνότητας.

Προκειμένου να παρέχεται η δυνατότητα υπολογισμού του επιπέδου ελάττωσης της επικινδυνότητας που επιτυγχάνεται σε σχέση πάντα με το κόστος αυτής της ελάττωσης, μπορεί να χρησιμοποιείται ο ακόλουθος τύπος:

$$RRL = \frac{R_{Before} - R_{After}}{\text{Risk Reduction Cost}} = \frac{L(UO) \times (P(UO)_{Before} - P(UO)_{After})}{\text{Risk Reduction Cost}} \qquad (4)$$

<u>όπου</u> :
RRL : Επίπεδο Ελάττωσης Επικινδυνότητας (Risk Reduction Leverage)
Risk Reduction Cost : Κόστος Ελάττωσης Επικινδυνότητας
R_{Before} : Η Επικινδυνότητα πριν τον έλεγχο
R_{After} : Η Επικινδυνότητα μετά τον έλεγχο

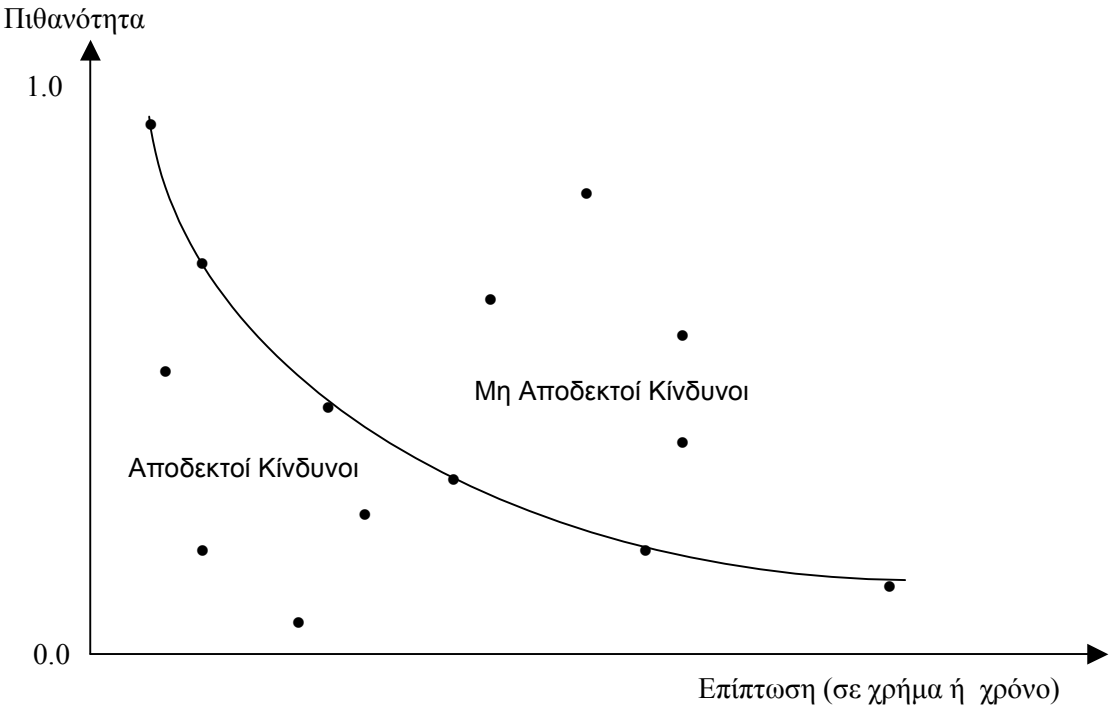
Η σχέση αυτή περιέχει στους υπεύθυνους ένα δείκτη της πορείας της διαδικασίας ελέγχου συναρτήσει του κόστους που σπαταλάται. Έτσι ώστε να βρίσκονται ανά πάσα στιγμή σε θέση να πάρουν απόφαση τερματισμού ή συνέχισης του ελέγχου, βάσει των περιορισμών και των απαιτήσεων που έχουν προκαθοριστεί.

4.3 Isorisk Chart

Από τη στιγμή που κάθε κίνδυνος χαρακτηρίζεται από την πιθανότητα εμφάνισής του και από την επίπτωση που θα έχει εάν εμφανιστεί, μπορεί να δημιουργηθεί ένα διάγραμμα το οποίο να αναφέρεται σε όλους συνολικά τους κινδύνους που έχουν αναγνωρισθεί και το οποίο θα τους διαχωρίζει σε αποδεκτούς και μη αποδεκτούς, σύμφωνα πάντα με το επίπεδο επικινδυνότητας που έχει προκαθοριστεί από την πολιτική που ακολουθείται κάθε φορά.

Πιο συγκεκριμένα, οι κίνδυνοι θα αναπαριστώνται με σημεία στο επίπεδο, των οποίων οι συντεταγμένες θα αντιστοιχούν στην πιθανότητα εμφάνισης και την επίπτωση. Η μεν πιθανότητα που θα βρίσκεται στον ένα άξονα θα υπολογίζεται σε κλίμακα από 0.0 έως 1.0, η δε επίπτωση θα μετράται είτε σε χρόνο είτε σε χρήμα. Επίσης στο διάγραμμα θα υπάρχει και μία καμπύλη η οποία αναπαριστώντας το επιτρεπτό επίπεδο επικινδυνότητας θα κάνει τον διαχωρισμό που προαναφέρθηκε [Phil98].

Ένα παράδειγμα ενός τέτοιου διαγράμματος παρουσιάζεται πιο κάτω :



Σχήμα 4.1: *An Isorisk Chart*

Το διάγραμμα αυτό, μπορεί να αποτελέσει ένα ακόμα εργαλείο στα χέρια της ομάδας ελέγχου, το οποίο θα τους βοηθά να αντιμετωπίζουν ένα υποσύνολο των κινδύνων που έχουν αναγνωρισθεί, βάσει πάντα της πολιτικής που έχει χαραχθεί.

4.4 Συνολική Διαδικασία Διαχείρισης Επικινδυνότητας σε σχέση με τον Έλεγχο Λογισμικού

Σύμφωνα με τα όσα έως τώρα αναφέρθηκαν, προκύπτει μια συνολική εικόνα της όλης διαδικασίας διαχείρισης επικινδυνότητας του λογισμικού σε συσχέτιση πάντα με την φάση του ελέγχου του.

Έτσι, πρώτ' απ' όλα και προκειμένου να παρθούν οι διάφορες αποφάσεις ελέγχου, πρέπει απαραιτήτως να έχει προηγηθεί καθορισμός μιας συγκεκριμένης πολιτικής σχετικά με το ανεκτό επίπεδο επικινδυνότητας, σύμφωνα με όσα προαναφέρθηκαν, περιλαμβάνοντας φυσικά και 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.

Κατόπιν ακολουθεί εκτίμηση του επιπέδου επικινδυνότητας που απομένει μετά την ολοκλήρωση του ελέγχου, συνυπολογίζοντας όλους τους παράγοντες που το επηρεάζουν και οι οποίοι παρουσιάστηκαν προηγουμένως.

Αν το επίπεδο αυτό ικανοποιεί τις όποιες απαιτήσεις που έχουν τεθεί σύμφωνα με την πολιτική που ακολουθείται, τότε ο έλεγχος έχει ικανοποιήσει έναν από τις παραμέτρους που χαρακτηρίζουν την ποιότητα του λογισμικού, δηλαδή την επίτευξη ενός ικανοποιητικού και αποδεκτού ποσοστού επικινδυνότητας.

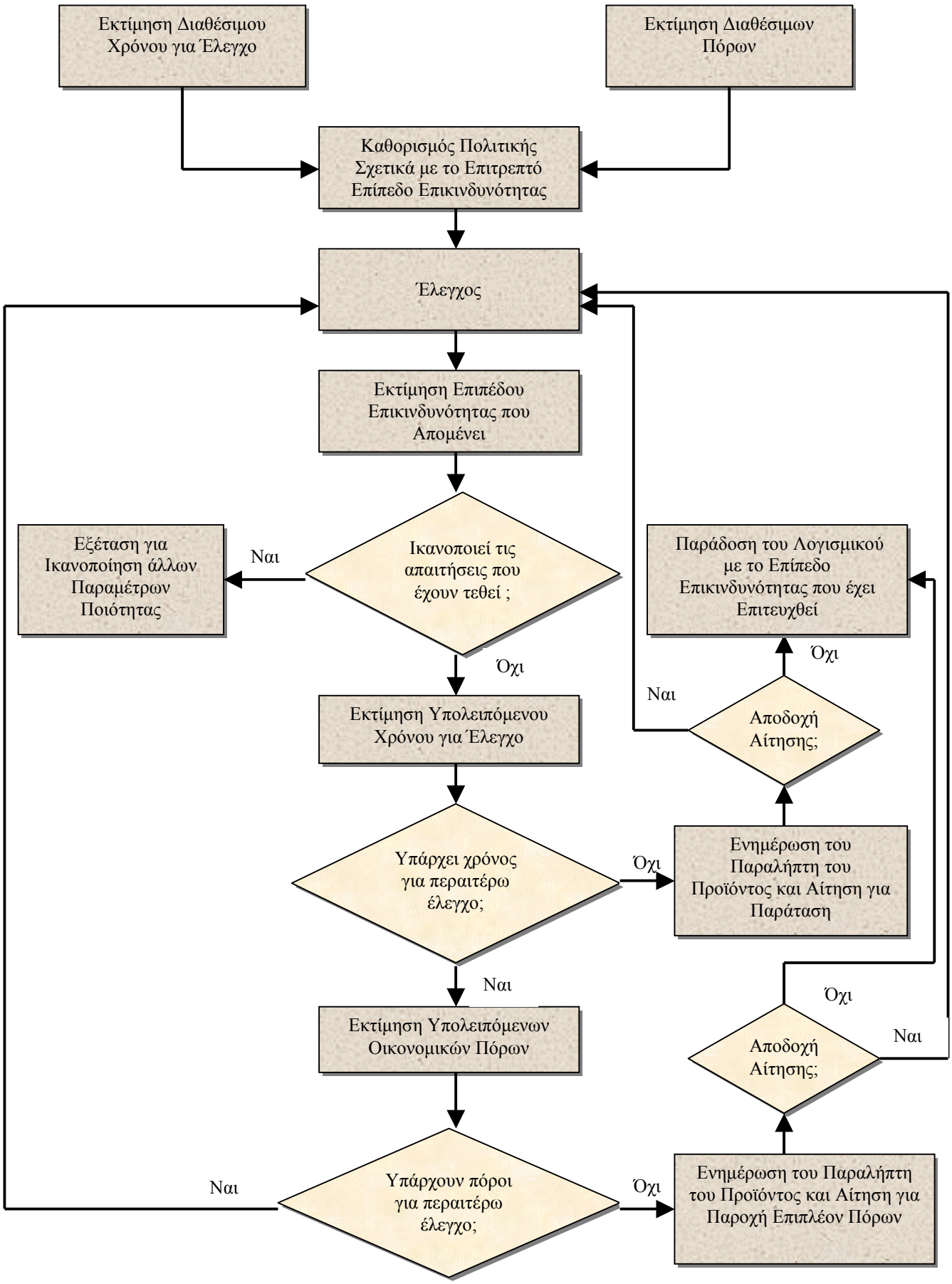
Εφόσον διασφαλιστεί το γεγονός ότι ικανοποιούνται και οι υπόλοιπες παράμετροι ποιότητας (όπως για παράδειγμα: λειτουργικότητα, αξιοπιστία, συντηρησιμότητα, αποδοτικότητα, κ.α.), έπεται ότι το σύστημα λογισμικού μπορεί να εγγυηθεί την ικανοποίηση του πελάτη (user satisfaction) και κατά συνέπεια είναι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

Εάν όμως η επικινδυνότητα που απομένει ξεπερνά τα όρια που έχουν τεθεί, πρέπει να παρθούν νέες αποφάσεις ώστε να διεξαχθεί περαιτέρω έλεγχος έως ότου τελικά αυτά επιτευχθούν, υπό την προϋπόθεση βέβαια ότι υπάρχουν τα χρονικά και οικονομικά περιθώρια που θα επιτρέψουν κάτι τέτοιο.

Σε περίπτωση όμως που δεν υπάρχουν είτε τα χρονικά περιθώρια είτε οι απαιτούμενοι πόροι για τη συνέχιση του ελέγχου, οι υπεύθυνοι για το έργο αιτούνται στον παραλήπτη του προϊόντος για να τους δοθεί παράταση ή να τους παραχωρηθούν επιπλέον πόροι.

Εφόσον η αίτηση αυτή γίνει δεκτή τότε η ομάδα ελέγχου μπορεί να προχωρήσει σε περαιτέρω έλεγχο, διαφορετικά το προϊόν παραδίδεται όπως έχει, εις γνώση όμως των πελατών σχετικά με το επίπεδο επικινδυνότητας που έχει απομείνει.

Η διαδικασία που μόλις περιγράφθηκε παρουσιάζεται διαγραμματικά παρακάτω:



Σχήμα 4.2: *Διαδικασία Διαχείρισης Επικινδυνότητας σε σχέση με τον Έλεγχο του Λογισμικού*

ΚΕΦΑΛΑΙΟ 5

5.1 Συμπεράσματα

Η διαχείριση επικινδυνότητας αποτελεί αδιαμφισβήτητα ένα πολύ σημαντικό κομμάτι οποιουδήποτε έργου λογισμικού και είναι ιδιαίτερα κρίσιμη για μεγάλα έργα, για έργα που χαρακτηρίζονται από υψηλή αβεβαιότητα ή για έργα των οποίων μια πιθανή δυσλειτουργία θα μπορούσε να προκαλέσει μη αναστρέψιμες καταστροφές.

Καθόλη τη διάρκεια ανάπτυξης του λογισμικού, υπεισέρχονται κίνδυνοι οι οποίοι είναι απαραίτητο να αναγνωρισθούν και να αντιμετωπιστούν όσο το δυνατόν πιο άμεσα. Γι' αυτόν ακριβώς το λόγο πρέπει να εφαρμόζεται μια συνεχής διαδικασία διαχείρισης της επικινδυνότητας, από την αρχή έως και το τέλος του κύκλου ζωής του λογισμικού, πράγμα το οποίο υποστηρίζει και η πλειοψηφία των μεθόδων και γενικότερα των προσεγγίσεων που έχουν ασχοληθεί με αυτό το τομέα.

Όμως, πρέπει κανείς να αποδεχθεί το γεγονός, ότι όσο κατάλληλη και αποτελεσματική και να είναι η διαχείριση της επικινδυνότητας, δεν θα μπορέσει ποτέ να επιτευχθεί πλήρης εξάλειψη των κινδύνων που απειλούν το λογισμικό. Και αυτό γίνεται ιδιαίτερα κατανοητό στη φάση του ελέγχου, όπου δεν υπάρχει ποτέ πιθανότητα να εκτελεστεί ένας τόσο εξαντλητικός έλεγχος ο οποίος να μπορέσει να διασφαλίσει ότι όλα θα λειτουργήσουν κατά το προσδοκώμενο και ότι τίποτα δεν θα μπορέσει να προκαλέσει μια αποτυχία του λογισμικού.

Από τη στιγμή λοιπόν που πάντα θα απομένει ένα ποσοστό επικινδυνότητας, ο στόχος που τίθεται είναι το ποσοστό αυτό να κυμαίνεται μέσα σε κάποια αποδεκτά όρια, τα οποία θέτονται ανάλογα με την περίπτωση σύμφωνα με την πολιτική που έχει χαραχθεί.

Εστιάζοντας λοιπόν στη φάση του ελέγχου, και προκειμένου να επιτευχθεί ο στόχος που προαναφέρθηκε, είναι απαραίτητο να ανεβρεθούν τρόποι με χρήση των οποίων να μπορεί να προσδιοριστεί το επίπεδο της εναπομείνασας επικινδυνότητας.

Στα πλαίσια αυτής της επιδίωξης, η εργασία, αφού έχει πρώτα παραθέσει επιχειρήματα που συνηγορούν υπέρ των απόψεων που προαναφέρθηκαν, παρουσιάζει και αναλύει ένα σύνολο παραγόντων οι οποίοι επηρεάζουν την πιθανότητα εμφάνισης ενός μη επιθυμητού αποτελέσματος και την απώλεια που θα έχει αυτή η εμφάνιση, επηρεάζοντας κατά συνέπεια και την επικινδυνότητα που απομένει μετά την ολοκλήρωση του ελέγχου, σύμφωνα με τη σχέση που ορίζεται.

Δεν πρέπει βέβαια σε καμία περίπτωση, να θεωρηθούν δεσμευτικοί οι συγκεκριμένοι παράγοντες. Ενώ δηλαδή θεωρείται ότι είναι απαραίτητο να ληφθούν υπόψη προκειμένου να προκύψει ένα ορθό αποτέλεσμα, δεν αποκλείεται η ύπαρξη και κάποιων άλλων επιπρόσθετων παραγόντων, οι οποίοι να παίζουν έναν εξίσου σημαντικό ρόλο στον προσδιορισμό της υπολειπόμενης επικινδυνότητας.

Πρέπει επίσης να σημειωθεί ότι οι παράγοντες που προέκυψαν στην πραγματικότητα αποτελούν ένα προστάδιο για την εύρεση ενός κατάλληλου μοντέλου επικινδυνότητας σε σχέση με τον έλεγχο λογισμικού, πράγμα που αν και αποτελούσε επιδίωξη της παρούσας εργασίας, τελικά δεν επιτεύχθει.

Αντί αυτού, προτείνεται μία διαδικασία η οποία συνδυάζει τη διαχείριση επικινδυνότητας με τον έλεγχο που διεξάγεται στο λογισμικό συγκεντρώνοντας τις ιδέες και τις προτάσεις που παρουσιάζονται σε προηγούμενα κεφάλαια και η οποία παριστάνεται με τη μορφή διαγράμματος ροής.

Η εφαρμογή της διαδικασίας αυτής προϋποθέτει την ύπαρξη συγκεκριμένης πολιτικής από την πλευρά των υπευθύνων για την ανάπτυξη του λογισμικού, έτσι ώστε να παρέχεται μια βάση για τη λήψη αποφάσεων σχετικών με το επιτρεπτό επίπεδο της μετά τον έλεγχο εναπομείνασας επικινδυνότητας.

5.2 Μελλοντική Έρευνα

Η παρούσα εργασία, αποδεικνύοντας το γεγονός ότι πάντα υπάρχει ένα ποσοστό επικινδυνότητας που απομένει μετά την ολοκλήρωση του ελέγχου του λογισμικού, παρέχει μια καλή βάση πάνω στην οποία μπορεί να στηριχθεί περαιτέρω έρευνα του πεδίου που προκύπτει από τη συσχέτιση των εννοιών της διαχείρισης επικινδυνότητας λογισμικού και του ελέγχου που διεξάγεται σε αυτό.

Πιο συγκεκριμένα, λαμβάνοντας κανείς τους παράγοντες που επηρεάζουν το ποσοστό της εναπομείνασας επικινδυνότητας, έτσι όπως παραθέτονται και αναλύονται στα πλαίσια της εργασίας, θα μπορούσε να κάνει μια προσπάθεια ποσοτικοποίησης τους και ανεύρεσης κατάλληλων μετρικών, ώστε να δοθεί τελικά η δυνατότητα υπολογισμού αυτού του ποσοστού μέσω μαθηματικού τύπου, ο οποίος θα είναι σύμφωνος με τη σχέση που έχει ήδη παρουσιασθεί.

Επίσης, θα μπορούσε να πραγματοποιηθεί μοντελλοποίηση της επικινδυνότητας του λογισμικού σε σχέση πάντα με τον έλεγχο, ενώ παράλληλα ανοίγονται και προοπτικές δημιουργίας ενός αυτοματοποιημένου εργαλείου, το οποίο στηριζόμενο και πάλι

στους ίδιους παράγοντες, αλλά πιθανόν και σε κάποιους επιπλέον, να παρέχει εκτίμηση του ποσοστού επικινδυνότητας που υπολείπεται μετά το πέρας του ελέγχου.

Η δημιουργία ενός τέτοιου εργαλείου, σίγουρα θα προκαλούσε γενικότερο ενδιαφέρον, και ιδιαίτερα αυτό των διαφόρων επιχειρήσεων και οργανισμών που ασχολούνται με την ανάπτυξη λογισμικού, αφού η χρήση του θα μπορούσε να αποτελέσει ένα μέσο μείωσης της αβεβαιότητας που αναπόφευκτα ενυπάρχει σε ένα οποιοδήποτε έργο λογισμικού.

Μια άλλη περιοχή στην οποία θα μπορούσε να έχει εφαρμογή με επιτυχία κάτι τέτοιο, είναι αυτή των εταιριών που παρέχουν ασφάλιση που σχετίζεται με λογισμικό. Γιατί σίγουρα, τόσο η συνειδητοποίηση, από μέρους αυτών που αναλαμβάνουν την ευθύνη της ασφάλισης, των όσων αναφέρονται στην εργασία, αλλά ακόμα περισσότερο η πιθανή δημιουργία ενός αυτοματοποιημένου εργαλείου σαν αυτό που προαναφέρθηκε, θα ελάττωνε σημαντικά το ποσοστό του ρίσκου που διατίθεται να πάρει ο ασφαλιστής προκειμένου να εξασφαλίσει ένα χρήστη λογισμικού από απώλειες που πιθανόν να συμβούν εξαιτίας μιας αποτυχίας του συστήματος (software insurability).

Τέλος, αυτό που είναι σημαντικό να επιτευχθεί είναι ένας κατάλληλος ποιοτικός, αλλά κυρίως ποσοτικός συνδυασμός της επικινδυνότητας του λογισμικού με τα διάφορα κριτήρια ποιότητας που το χαρακτηρίζουν, όπως είναι για παράδειγμα η αξιοπιστία ή η ασφάλεια, ώστε να παρέχεται στον ενδιαφερόμενο μια πληρέστερη και ακριβέστερη εικόνα του.

ΒΑΣΙΚΟΙ ΟΡΙΣΜΟΙ

Σύστημα λογισμικού (Software system)
Ένα αλληλεπιδρόμενο σύνολο από υποσυστήματα λογισμικού, το οποίο είναι ενσωματωμένο σε ένα λειτουργικό περιβάλλον που παρέχει εισροές στο σύστημα λογισμικού και δέχεται υπηρεσίες (εκροές) από το λογισμικό. Ένα υποσύστημα λογισμικού αποτελείται με τη σειρά του από άλλα υποσυστήματα κ.ο.κ., έως ένα επιθυμητό επίπεδο αποσύνθεσης σε μικρότερα σημαντικά στοιχεία.

Υπηρεσία (Service)
Αναμενόμενη υπηρεσία ενός συστήματος λογισμικού είναι μια εξαρτώμενη από το χρόνο σειρά εκροών, η οποία συμφωνεί με τις αρχικές απαιτήσεις στις οποίες στηρίχθηκε η υλοποίηση του λογισμικού (για τον σκοπό της επαλήθευσης), ή να συμφωνεί με το τι οι χρήστες του συστήματος αντιλήφθηκαν ως σωστές τιμές (για τον σκοπό της επικύρωσης).

Αποτυχία (Failure)
Μια αποτυχία συμβαίνει όταν ο χρήστης αντιλαμβάνεται ότι το πρόγραμμα αδυνατεί να εκπληρώσει την αναμενόμενη υπηρεσία.

Σφάλμα (Fault)
Ένα σφάλμα αποκαλύπτεται όταν λαμβάνει χώρα μία αποτυχία του προγράμματος ή ένα εσωτερικό λάθος. (Το σφάλμα συχνά αναφέρεται και ως bug).

Στις περισσότερες περιπτώσεις ένα σφάλμα μπορεί να ανιχνευτεί και να διορθωθεί. Υπάρχουν όμως και περιπτώσεις που παραμένει μία υπόθεση η οποία δεν μπορεί να επιβεβαιωθεί επαρκώς.

Ελάττωμα (Defect)
Ο όρος αυτός μπορεί να χρησιμοποιηθεί γενικότερα για την αναφορά είτε σε μια αποτυχία, είτε σε ένα σφάλμα.

Λάθος (Error)
Μία αντίφαση μεταξύ μιας υπολογισμένης, παρατηρημένης ή μετρημένης τιμής ή συνθήκης και της πραγματικής, καθορισμένης ή θεωρητικά σωστής τιμής ή συνθήκης. Λάθη συμβαίνουν όταν κάποιο κομμάτι του λογισμικού παράγει μια μη επιθυμητή κατάσταση. Μπορεί να είναι εννοιολογικό, συντακτικό ή οφειλόμενο σε παράλειψη (clerical).

Απειλή (Threat)
Μια πιθανή ενέργεια ή ένα γεγονός που μπορεί να προκαλέσει την απώλεια ενός ή περισσότερων χαρακτηριστικών της ασφάλειας του συστήματος λογισμικού.

Ευπάθεια (Vulnerability) συστήματος λογισμικού
Μια αδυναμία του συστήματος, η οποία επιτρέπει, κάτω από ένα σύνολο περιστάσεων, να συμβεί οποιοδήποτε είδος παραβίασης, δηλαδή επιτρέπει σε μία απειλή να είναι επιτυχής.

Επίπτωση (Impact)
Περιλαμβάνει απώλεια αξίας, οικονομικές απώλειες, αποκάλυψη προσωπικών στοιχείων, κοινωνικό χάος, περιβαλλοντική καταστροφή, απώλεια ανθρώπινης ζωής, ή οποιαδήποτε άλλη μορφή απώλειας που θα μπορούσε να προκύψει ως συνέπεια μια συγκεκριμένης απειλής.

ΒΙΒΛΙΟΓΡΑΦΙΑ – ΑΝΑΦΟΡΕΣ

[Adler99] Terry R. Alder, John G. Leonard, Ric K. Nordgren, "*Improving Risk Management: moving from risk elimination to risk avoidance*", Information and Software Technology 41 (1999) 29 – 34.

[Ashr97] Mohammad Ashrafuzzaman, "*Software Risk and Reliability*", CST, November 1997.

[Bach99a] James Bach, "*Risk and Requirements Based Testing*", Computer, June 1999.

[Bach99b] James Bach, "*Risk-Based Testing*", Software Testing & Quality Engineering, November 1999.
http://www.stqemagazine.com/

[Bala98] M. G. Balakrishna Rao, "*Software Risk Management SPINtalk*", Verifone, India Private Ltd, January 1998.
http://stpb.allindia.com/spin/spin-balki/

[Bind95] Robert V. Binder, "*Testing Object-Oriented Systems: A Status Report*", RBSC Corporation, 1995.
http://www.stsc.hill.af.mil/crosstalk/1995/apr/testinoo.asp

[Boehm91] Barry W. Boehm, "*Software Risk Management: Principles and Practices*", IEEE Software, Vol. 8, No. 1, Jan. 1991, pp. 32-41.

[Cowa88] P. David Coward, "*A Review of Software Testing*", Information and Software Technology, Vol. 30, No. 3, April 1998, pp.189-198.

[Fried95] Michael A. Friedman, Jeffrey M. Voas, "*Software Assessment: reliability, safety, testability*", John Wiley & Sons, Inc., 1995.

[Hall98] Elaine M. Hall, "*Managing Risk: Methods for Software Systems Development*", Addison Wesley, 1998.

[IPL96a] Information Processing Limited (IPL), "*An Introduction to Software Testing*", September 1996.
http://www.teleport.com/~qcs/papers/p820.htm

[IPL96b] Information Processing Limited (IPL), "*An Introduction to Safety Critical Systems*", September 1996.
http://www.teleport.com/~qcs/papers/p826.htm

[Kit95] Edward Kit, "*Software Testing In The Real World: improving the process*", Addison Wesley, 1995.

[Kont97] Jyrki Kontio, Victor R. Basili, "*Riskit: Increasing Confidence in Risk Management*", Software Tech News, 1997.
http://www.dacs.com/awareness/newsletters/technews2-2/riskit.html

[Leves87] Nancy G. Leveson, "*Software Safety*", SEI Curriculum Module SEI-CM-6-1.1 (Preliminary), July 1987.

[McGet93] Andrew D. McGettric, "*Software in Safety-Related Systems: An Overview*", IEEE Software, 1993.

[Phil98] Dwyane Phillips, "*The Software Project Manager's Hanbook*", IEEE Computer Society, 1998.

[Redm99] Felix Redmill, "*Why Systems Go Up In Smoke*", The Computer Bulletin, September 1999.

[Rosen96] Linda H. Rosenberg, Lawrence E. Hyatt, "*Software Metrics Program for Risk Assessment*", 47th International Austronautical Congress & Exhibition, 29th Safety and Rescue Symposium, October 1996.
http://satc.gsfc.nasa.gov/support/IAC_OCT96/iaf.html

[Voas94] Jeffrey M. Voas, Keith W. Miller, "*Dynamic Testability Analysis for Assessing Fault Tolerance*", High Integrity Systems Journal, 1(2): 171-178, 1994.

[Wieg98] Karl E. Wiegers, "*Know Your Enemy : Software Risk Management*", Software Development, October 1998.
http://www.processimpact.com/articles/risk_mgmt.html

[Σκορ91] Εμμ. Σ. Σκορδαλάκης, "*Εισαγωγή στην Τεχνολογία Λογισμικού*", Αθήνα, 1991.

[Τζαβ98] Αθανάσιος Ι. Τζαβάρας, "*Συσχέτιση Αξιοπιστίας και Ελέγχου Λογισμικού*", Πτυχιακή Εργασία, Αθήνα, 1998.