ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

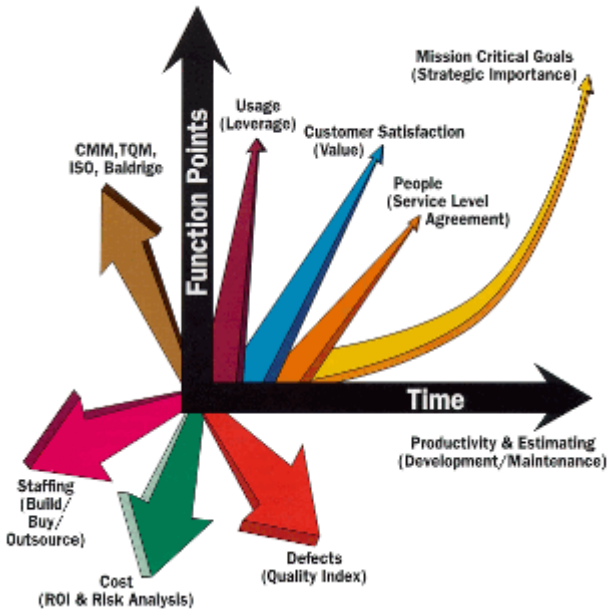
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. "ΔΙΔΑΚΤΙΚΗ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ & ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ "

Κατεύθυνση: ΨΗΦΙΑΚΕΣ ΕΠΙΚΟΙΝΩΝΙΕΣ & ΔΙΚΤΥΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ :

Εκτίμηση κόστους πληροφοριακών συστημάτων. Θεωρία και μελέτη περίπτωσης.



Λάζαρης Θεοφάνης ΜΕ/09087

Επιβλέπουσα : Φλώρα Μαλαματένιου

ΠΕΙΡΑΙΑΣ , Απρίλιος 2013

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο του ΠΜΣ «Διδακτική της Τεχνολογίας και Ψηφιακά Συστήματα» στο τμήμα Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς.

Υπεύθυνη κατά την εκπόνηση της διπλωματικής ήταν η αναπληρώτρια Καθηγήτρια καθηγήτρια κ. Φ. Μαλαματένιου, στην οποίο οφείλω ιδιαίτερες ευχαριστίες για την ανάθεση αυτής και την δυνατότητα που μου δόθηκε να ασχοληθώ με ένα τόσο ενδιαφέρον και σύγχρονο θέμα. Την ευχαριστώ για το ενδιαφέρον της αλλά και για την συνεχή βοήθεια και καθοδήγησή της.

Τέλος ευχαριστώ την οικογένεια μου και όλους τους φίλους μου, που με τον τρόπο τους , με στήριξαν όλοι μαζί και ο καθένας ξεχωριστά σε αυτήν μου την προσπάθεια.

Λάζαρης Θεοφάνης

ΠΕΙΡΑΙΑΣ , Απρίλιος 2013

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία επικεντρώνεται στην παρουσίαση και ανάλυση των μεθόδων εκτίμησης κόστους πληροφοριακών συστημάτων. Επιπλέον περιγράφεται η μετρική Λειτουργικών Σημείων (Function Point Analysis) κατά την οποία πραγματοποιείται κατακερματισμός για τον έλεγχο του μεγέθους αλλά και της κατανόησης της πολυπλοκότητας ενός λογισμικού. Επίσης, περιγράφεται το Δομικό Μοντέλο Κόστους COCOMO και παρουσιάζεται μια μελέτη περίπτωσης που αξιολογεί ένα λογισμικό με βάση το μοντέλο αυτό.

Αναλυτικότερα η διπλωματική εργασία αποτελείται από πέντε επιμέρους κεφάλαια.

Το πρώτο κεφάλαιο αναφέρεται στη αξιολόγηση πληροφοριακών συστημάτων και πιο συγκεκριμένα στις στρατηγικές αξιολόγησης που καταγράφονται στην διεθνή βιβλιογραφία.

Στο δεύτερο κεφάλαιο περιγράφονται οι μετρικές αξιολόγησης και παρουσιάζεται η αξιολόγηση λογισμικού ανάλογα με τις γραμμές κώδικα που περιέχει (LoC). Στην συνέχεια γίνεται μία εκτενής ανάλυση του τρόπου εκτίμησης του κόστους λογισμικών βήμα προς βήμα.

Στο τρίτο κεφάλαιο παρουσιάζονται οι τεχνικές εκτίμησης κόστους, περιγράφονται τα υπάρχουσα μοντέλα και αναλύεται το Δομικό Μοντέλο Κόστους COCOMO το οποίο αποτελεί έναν από τους πιο διαδεδομένους και αξιόπιστους τρόπους αξιολόγησης λογισμικού.

Στο τέταρτο κεφάλαιο αναλύεται εκτενώς η μετρική Λειτουργικών Σημείων. Εξηγούνται οι όροι που χρησιμοποιούνται και περιγράφεται ο τρόπος λειτουργίας της μετρικής , ο τρόπος μέτρησης των λειτουργικών σημείων (function points) αλλά και η μέθοδος που ακολουθείται για την εύρεση του Συντελεστή Προσαρμογής Λειτουργικών Σημείων (Adjusted Function Points - AFP).

Τέλος, στο πέμπτο κεφάλαιο παρουσιάζεται μια μελέτη περίπτωσης που αφορά στη διαδικασία εύρεσης του μεγέθους του λογισμικού Debian μετρώντας τις γραμμές κώδικα που αποτελείται ενώ ακολουθεί η αξιολόγηση του κόστους του με χρήση του Δομικού Μοντέλου Κόστους COCOMO.

ABSTRACT

The current dissertation presents and analyzes the methods that are used in order to evaluate the cost of an information system. Furthermore, a method that fragments software with purpose to evaluate its size and complexity is presented. It describes the COCOMO metric and presents a case study which is evaluated with this method.

More specifically, this study consists of five subchapters. In the first chapter current strategies that are used to evaluate an information system are presented.

In the second chapter the LoC metric evaluation, a most used method for evaluating software, is described .Then, an estimation method of a software cost is extensively analyzed

In the third chapter the techniques of cost estimation and the current models are presented. Furthermore, the Constructive Cost Model (COCOMO) the most widespread and reliable method to evaluate software is analyzed.

In the fourth chapter, the Function Point Analysis metric method is presented and a thorough analysis consists of the terms, the operations and the way that Function Points are calculated, as well as the method that applied to find the Adjusted Function Points (AFP).

In the fifth chapter a case study is presented which outlines the process of finding the size of Debian software by counting its lines of code. The COCOMO metric is further used to evaluate the cost of the software.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1
ΑΞΙΟΛΟΓΗΣΗ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

1.1 ΕΙΣΑΓΩΓΗ

Τα τελευταία χρόνια όλο και περισσότερα χρήματα δαπανώνται σε όλο τον κόσμο για την ανάπτυξη πληροφοριακών συστημάτων. Λόγω της αλματώδους αυτής ανάπτυξης το ενδιαφέρον έχει στραφεί στην αξιολόγησή τους, τα πιθανά οφέλη και την αποτελεσματικότητά τους.
Προκύπτει έτσι, ολοένα και περισσότερο έντονα η ανάγκη για τη δημιουργία κατάλληλων παραμέτρων, οι οποίες θα έχουν στόχο την εκ βάθους αξιολόγηση και μέτρηση της επιτυχούς λειτουργίας ενός πληροφοριακού συστήματος. Η αξιολόγηση δεν είναι ένας εύκολος στόχος και συνεπώς υπάρχουν πολλές προτάσεις για το πώς να αξιολογήσει κανείς ένα πληροφοριακό σύστημα.

1.2 ΑΝΑΓΚΗ ΑΞΙΟΛΟΓΗΣΗΣ

Ολόκληρη η διαδικασία της αξιολόγησης συμβάλλει στο να λαμβάνονται σωστές αποφάσεις, να διασφαλίζεται η ποιότητα του συστήματος και να εκτιμάται η έκταση στην οποία μπορούν να λυθούν τα προβλήματα, που τυχόν το σύστημα αντιμετωπίζει.

Όταν αξιολογούμε ένα πληροφοριακό σύστημα, ασχολούμαστε κατά κύριο λόγο με την οργάνωση και τη διοίκηση ενός οργανισμού, καθώς και με τους χρήστες του. Οι παράμετροι, που καλούμαστε να χρησιμοποιούμε για να αξιολογήσουμε ένα πληροφοριακό σύστημα, είναι [1]:

- τα συστατικά στοιχεία του συστήματος
- οι πληροφοριακές διεργασίες
- οι υπηρεσίες και τα προϊόντα
- οι πληροφοριακές λειτουργίες
- ολόκληρο το πληροφοριακό σύστημα, καθώς και
- το περιβάλλον του

Η χρησιμότητα αυτής της διαδικασίας είναι ιδιαιτέρως σημαντική, γιατί με την εφαρμογή της το σύστημα παρέχει καλύτερα προϊόντα και υπηρεσίες, δικαιολογείται η υπάρχουσα κατάστασή του και γίνεται κατανοητή η λειτουργία του.

1.3 ΤΡΟΠΟΙ ΑΞΙΟΛΟΓΗΣΗΣ

Η αξιολόγηση πληροφοριακών συστημάτων δεν ήταν ποτέ ένας εύκολος στόχος και συνεπώς υπάρχουν πολλές προτάσεις για το πώς μπορεί να αξιολογηθεί επαρκώς ένα τέτοιο σύστημα [1]. Μεγάλο μέρος της βιβλιογραφίας καταλαμβάνεται από ορθολογικές προσεγγίσεις όπου αξιολογείται η ποσοτική διαδικασία υπολογισμού του κέρδους και του κόστους ενός συστήματος με προκαθορισμένα κριτήρια. Υπάρχουν ακόμα και ερμηνευτικές προσεγγίσεις οι οποίες αντιμετωπίζουν το σύστημα ως κοινωνικό σύνολο το οποίο περιέχει τεχνικές πληροφορίες [2]. Υπάρχουν ακόμα αθροιστικές προσεγγίσεις που περιλαμβάνουν μέτρα ή κριτήρια, κάποιες που επικεντρώνονται σε οικονομικά κριτήρια και ορισμένες που εστιάζουν σε κριτήρια προσανατολισμένα στον χρήστη.

Όλες αυτές οι προσεγγίσεις αν και διαφέρουν μεταξύ τους έχουν κοινό στόχο το πώς ο αξιολογητής θα πρέπει να δράσει προκειμένου να έχει τα βέλτιστα αποτελέσματα. Εκτός από το “πως” όμως είναι πολύ σημαντικό το να αποφασιστεί και το “τι” θα αξιολογηθεί. Τα πληροφοριακά συστήματα κατά την αξιολόγησή τους μπορούμε να τα σκεφτούμε σε τουλάχιστον δύο καταστάσεις. Μπορούμε να σκεφτούμε το σύστημα όπως είναι ή κατά την χρήση του.

- **Το πληροφοριακό σύστημα όπως είναι (system as such).**

Αξιολογούμε το πληροφοριακό σύστημα χωρίς καμία συμμετοχή από τους χρήστες [2] . Τα αποτελέσματα της αξιολόγησης βασίζονται στην εκτίμηση του αξιολογητή για το πώς το πληροφοριακό σύστημα υποστηρίζει την εκάστοτε οργάνωση. Αυτή η στρατηγική είναι απαλλαγμένη από τις εκτιμήσεις των χρηστών για το πώς το πληροφοριακό σύστημα ωφελεί την εργασία τους. Το αντικείμενο της αξιολόγησης είναι το πληροφοριακό σύστημα αυτό καθ' εαυτό. Δεν υπάρχει καμία μελέτη πραγματικής κατάστασης χρήσης του συστήματος. Ο αξιολογητής εξερευνά τι είναι δυνατό να κάνει με το σύστημα.

- **Το πληροφοριακό σύστημα σε χρήση (system in use)**.

Μελετάμε μία κατάσταση χρήσης όπου ένας χρήστης αλληλεπιδρά με το σύστημα. Αυτή η περίπτωση είναι πιο πολύπλοκη από την άλλη γιατί συμπεριλαμβάνει και έναν χρήστη και δίνει μια πιο ολοκληρωμένη εικόνα. Τα δεδομένα για αυτή την αξιολόγηση μπορούν να προκύψουν από συνεντεύξεις των χρηστών και τις εκτιμήσεις τους για την ποιότητα του συστήματος, από παρατηρήσεις της αλληλεπίδρασης των χρηστών με το πληροφοριακό σύστημα και από το ίδιο το πληροφοριακό σύστημα.

1.4 ΣΤΡΑΤΗΓΙΚΕΣ ΑΞΙΟΛΟΓΗΣΗΣ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Στην αξιολόγηση συστημάτων μπορούμε να διακρίνουμε 3 βασικές στρατηγικές [3].

1. **Αξιολόγηση με βάση τους στόχους (Goal-based evaluation).**

Όπου ρητοί στόχοι από το οργανωτικό πλαίσιο οδηγούν την αξιολόγηση. Η εστίαση γίνεται πάνω στα επιθυμητά αποτελέσματα του συστήματος, τους στόχους. Οι στόχοι που χρησιμοποιούνται για την αξιολόγηση προέρχονται από ένα συγκεκριμένο οργανωτικό πλαίσιο. Αυτό σημαίνει ότι ισχύουν περιστασιακά. Η βασική στρατηγική αυτής της προσέγγισης είναι να μετρήσουμε εάν συγκεκριμένοι στόχοι εκπληρώνονται ή όχι, σε πιο βαθμό και με πιο τρόπο. Η προσέγγιση είναι συμπερασματική. Τι είναι αυτό που μετριέται εξαρτάται από τον χαρακτήρα των στόχων και γι' αυτό μπορούν να χρησιμοποιηθούν και ποιοτικές και ποσοτικές μέθοδοι.

2. **Ανεξάρτητη των στόχων αξιολόγηση (Goal-free evaluation).**

Εδώ κανένας ρητός στόχος δεν χρησιμοποιείται, είναι μία επαγωγική και κατά περίπτωση οδηγημένη στρατηγική. Είναι μια πιο ερμηνευτική προσέγγιση που βλέπει το πληροφοριακό σύστημα σαν κοινωνικό σύστημα που σε αυτό
έχει εισχωρήσει η τεχνολογία. Η ανεξάρτητη των στόχων αξιολόγηση γίνεται συλλέγοντας στοιχεία όσον αφορά μια ευρεία περιοχή πραγματικών αποτελεσμάτων και αξιολογώντας τη σημασία αυτών των αποτελεσμάτων. Μόνο τα αποτελέσματα του συστήματος μετρούνται . Η βασική στρατηγική αυτής της προσέγγισης είναι η επαγωγική αξιολόγηση. Η προσέγγιση έχει σκοπό να ανακαλύψει ποιότητες του αντικειμένου της μελέτης. Κάποιος μπορεί να πει ότι ο αξιολογητής ψάχνει για πιθανά προβλήματα και ότι η γνώση του αντικειμένου της μελέτης προκύπτει κατά την διάρκεια της αξιολόγησης [4,[6] .

Οι κύριοι στόχοι της αξιολόγησης αυτής είναι:

- Να αποφευχθεί ο κίνδυνος της μονοδιάστατης και στενής παρακολούθησης ενός συστήματος με πιθανό αποτέλεσμα την μη πρόβλεψη εξωτερικών κινδύνων.
- Να αφαιρεθούν οι αρνητικές αντιδράσεις από ένα μη αναμενόμενο αποτέλεσμα.
- Να εξαλειφτούν οι αρνητικές προκαταλήψεις σε αξιολογήσεις ανεξάρτητες των στόχων.
- Να διατηρήσει την ανεξαρτησία και την αντικειμενικότητα του αξιολογητή.

3. Αξιολόγηση βασισμένη σε κριτήρια (Criteria-based evaluation).

Ορισμένα γενικά κριτήρια χρησιμοποιούνται ως κριτήρια αξιολόγησης – η διαφορά με την αξιολόγηση με βάση τους στόχους είναι ότι τα κριτήρια είναι γενικά και μη περιορισμένα σε ένα συγκεκριμένο οργανωτικό πλαίσιο. Υπάρχουν πολλές αξιολογήσεις βάση κριτηρίων όπως οι πίνακες ελέγχου ή τα heuristics. Αυτό που είναι χαρακτηριστικό σε αυτές τις προσεγγίσεις είναι ότι το πληροφοριακό σύστημα και η αλληλεπίδραση του μεταξύ των χρηστών λειτουργούν σαν την βάση για την αξιολόγηση μαζί με ένα σύνολο προκαθορισμένων κριτηρίων. Τα κριτήρια που χρησιμοποιούνται σε αντίθεση με την αξιολόγηση στόχων δεν προέρχονται από ένα συγκεκριμένο οργανωτικό πλαίσιο έχοντας έτσι μια πιο γενική εφαρμογή.

1.5 ΣΥΝΔΥΑΣΜΟΣ ΠΡΟΣΕΓΓΙΣΕΩΝ

Συνδυάζοντας τις τρεις προσεγγίσεις για το πώς αξιολογούμε ένα σύστημα με τις δύο για το τι ακριβώς αξιολογούμε δημιουργούνται έξι νέοι τύποι αξιολόγησης [4] που παρουσιάζονται στον Πίνακα 1.

Αυτοί είναι :

Συνδυασμοί αξιολόγησης πληροφοριακών συστημάτων	**Το πληροφοριακό σύστημα όπως είναι**	**Το πληροφοριακό σύστημα σε χρήση**
Αξιολόγηση με βάση τους στόχους	Τύπος 1	Τύπος 4
Ανεξάρτητη των στόχων αξιολόγηση	Τύπος 2	Τύπος 5
Αξιολόγηση βασισμένη σε κριτήρια	Τύπος 3	Τύπος 6

Πίνακας 1. Τύποι αξιολόγησης

- **Ανεξάρτητη των στόχων αξιολόγηση** - **Το πληροφοριακό σύστημα όπως είναι** .(Goal-free evaluation-System as such)

Σε αυτόν τον συνδυασμό η αξιολόγηση γίνεται με μια πιο ελεύθερη προσέγγιση. Η στρατηγική του συστήματος είναι επαγωγική και οι πηγές δεδομένων είναι το ίδιο το σύστημα και οι περιγραφές του. Αυτή η προσέγγιση έχει με μικρό βαθμό συμμετοχής και χρησιμοποιείται σε περιπτώσεις όπου χρειαζόμαστε μια εισαγωγική μελέτη για το σύστημα. Ένα από τα κύρια πλεονεκτήματα είναι ότι δίνει μία γρήγορη γενική εικόνα του συστήματος στον αξιολογητή.

Για να επιτευχθεί αυτός ο τύπος αξιολόγησης θα πρέπει ο αξιολογητής να έχει μία βαθιά γνώση του συστήματος. Η αξιολόγηση γίνεται υπολογίζοντας τα πλεονεκτήματα και τα μειονεκτήματα του συστήματος. Για καλύτερα αποτελέσματα αυτός ο τύπος αξιολόγησης συνδυάζεται με κάποιο άλλο τύπο ενώ τις περισσότερες φορές αποτελεί είσοδο για άλλο τύπο.

- **Αξιολόγηση με βάση τους στόχους-Το πληροφοριακό σύστημα όπως είναι.** (Goal-based evaluation- System as such)

Αυτή η αξιολόγηση στηρίζεται σε προκαθορισμένους επιχειρηματικούς στόχους ενώ το αντικείμενο της αξιολόγησης είναι το ίδιο το σύστημα. Η μέθοδος αυτή βοηθά στο να γίνει ξεκάθαρο εάν οι στόχοι έχουν επιτευχθεί. Η μέθοδος αυτή μπορεί να θεωρηθεί αφαιρετική δεδομένου ότι ο αξιολογητής υποθέτει ότι οι στόχοι έχουν επιτευχθεί ώστε να γίνει η αξιολόγηση της εργασίας.

Είσοδοι αποτελούν εκτός από το ίδιο το σύστημα οι περιγραφές των στόχων, οι προδιαγραφές και η περιγραφή τα ου συστήματος. Αυτό το είδος αξιολόγησης είναι αρκετά απλό και μπορεί να επιλεγεί εάν το σύστημα συμβαδίζει με τους στόχους που έχουν τεθεί είτε εάν υπάρχουν περιορισμένοι πόροι.

Για να επιτευχθεί μια αξιολόγηση με βάση τους στόχους πρέπει να περιγραφούν οι στόχοι της επιχείρησης και η λειτουργικότητα του συστήματος. Αυτό επιτυγχάνεται με την μελέτη του ίδιου του συστήματος ή και με την βοήθεια του δημιουργού του. Όταν οι στόχοι αυτοί επιτευχθούν τότε μπορεί να ξεκινήσει η αξιολόγηση. Αυτό σημαίνει ότι ο αξιολογητής αποφασίζει για το εάν η λειτουργικότητα έχει αγγίξει τους επιχειρησιακούς στόχους.

- **Αξιολόγηση βασισμένη σε κριτήρια-Το πληροφοριακό σύστημα όπως είναι** .(Criteria-based evaluation -System as such)

Αυτός ο συνδυασμός αξιολόγησης πραγματοποιείται όταν υπάρχουν κάποια προκαθορισμένα γενικά κριτήρια και το αντικείμενο αξιολόγησης είναι το ίδιο το σύστημα. Η προοπτική της αξιολόγησης εξαρτάται από τον χαρακτήρα των κριτηρίων. Η μέθοδος που χρησιμοποιείται είναι αφαιρετική και το ζητούμενο από τον αξιολογητή είναι να κριθεί εάν το σύστημα καλύπτει τα δεδομένα κριτήρια που έχουν τεθεί.

Το σημαντικότερο σημείο είναι ο ορισμός των κατάλληλων κριτηρίων. Η επιλογή τους γίνεται ανάλογα με την προοπτική της αξιολόγησης.

- **Ανεξάρτητη των στόχων αξιολόγηση** - **Το πληροφοριακό σύστημα σε χρήση.** (Goal-free evaluation- System in use)

Αυτός ο τύπος αξιολόγησης δημιουργείται όταν συνδυαστεί μία πιο ελεύθερη προσέγγιση (ανεξάρτητη των στόχων αξιολόγηση) ενώ ταυτόχρονα αξιολογούμε το σύστημα σε χρήση. Έχοντας μία προοπτική πια, που δεν περιορίζεται στα στενά πλαίσια των στόχων χρησιμοποιείται επαγωγική προσέγγιση του συστήματος ώστε να υπάρξει πιο ελεύθερη κρίση που δεν τερματίζει μόνο στην επίτευξη των στόχων.

Μία σημαντική διαφορά αυτής της αξιολόγησης σε σύγκριση με τον πρώτο τύπο είναι ότι υπάρχουν περισσότερες πηγές δεδομένων. Αυτός ο τύπος θα μπορούσε να επιλεχθεί όταν επιθυμείται μια ευρεία και βαθιά εκτίμηση και επιπλέον διατίθενται περισσότεροι πόροι.

Προκειμένου να εκπληρωθούν οι στόχοι αυτής της αξιολόγησης χρειάζεται να γίνει περιγραφή της λειτουργικότητας του συστήματος, περιγραφή των χρηστών (γνώσεις που έχουν, θέσεις που κατέχουν, αρμοδιότητες) αλλά και της αλληλεπίδρασης που υπάρχει μεταξύ και συστήματος. Η αλληλεπίδραση [6] του χρήστη μπορεί να περιγραφή με μία τεχνική που ονομάζεται 'think aloud'. Οι παρατήρηση των χρηστών θα μπορούσε να συνδυαστεί επίσης με συνεντεύξεις τους ώστε να επιτευχθεί ο διερευνητικός χαρακτήρας του μοντέλου.

- **Αξιολόγηση με βάση τους στόχους- Το πληροφοριακό σύστημα σε χρήση.** (Goal-based evaluation- System in use)

Σε αυτό τον συνδυασμό οριοθετούνται οι στόχοι που θα αξιολογηθεί το σύστημα το οποίο αποτελεί και το αντικείμενο αξιολόγησης. Η μέθοδος είναι παραγωγική και η μελέτη γίνεται στο εάν το σύστημα έχει εκπληρώσει τους στόχους του ή όχι. Εκτός από τους στόχους και τις προδιαγραφές του συστήματος χρειάζεται να μελετηθεί και ο ρόλος των χρηστών και η αλληλεπίδρασή τους. Ο τύπος αυτός επιλέγεται όταν υπάρχουν αρκετοί πόροι ή όταν χρειάζεται μία στοχευόμενη αξιολόγηση προσανατολισμένη στους στόχους.

Προκειμένου να επιτευχτεί η ανάλυση του συστήματος ο αξιολογητής ξεκινά την διερεύνηση των στόχων και μελετά την γενικότερη λειτουργικότητα. Για καλύτερα αποτελέσματα μελετά τις λεπτομέρειες του συστήματος , αναλύει την επιρροή των χρηστών και πραγματοποιεί συνεντεύξεις με τους επιχειρησιακούς διευθυντές. Ο αξιολογητής μελετά επίσης την πρότερη γνώση των χρηστών, τις αρμοδιότητες και τον ρόλο τους στο σύστημα. Όταν οι στόχοι, η αξιολόγηση , η αλληλεπίδραση και η οι γνώσεις των χρηστών έχουν αναλυθεί τότε μια αξιολόγηση με βάση τους στόχους συνδυασμένη με το σύστημα σε χρήση μπορεί να επιτευχθεί [5].

- **Αξιολόγηση βασισμένη σε κριτήρια- Το πληροφοριακό σύστημα σε χρήση**. (Criteria-based evaluation - System in use)

Σε αυτόν τον τύπο συνδυάζεται μία αξιολόγηση βασισμένη σε κάποια γενικά κριτήρια ενώ ταυτόχρονα το αντικείμενο της αξιολόγησης είναι το ίδιο το σύστημα. Η μέθοδος που ακολουθείται είναι αφαιρετική και ο αξιολογητής καλείται να αποφασίσει εάν το σύστημα ικανοποιεί τα κριτήρια αυτά ή όχι. Η αλληλεπίδραση των χρηστών με το σύστημα απαιτεί παρατήρηση και αξιολόγηση παρόμοια με αυτήν των δύο προηγούμενων τύπων. Απαιτείται κατανόηση της στάσης των χρηστών σε σχέση με το σύστημα και χρησιμοποιείται όταν είναι απαραίτητη μία εστιασμένη αξιολόγηση αλλά όχι προσανατολισμένη στους στόχους.

Το πρωταρχικό στάδιο αυτής της αξιολόγησης είναι να τεθούν τα κατάλληλα κριτήρια και έπειτα να κριθεί η λειτουργικότητα του συστήματος. Γίνεται πάλι μελέτη του συστήματος και των χρηστών και στο τέλος αναλύεται εάν υπάρχει τήρηση αυτών των πρωταρχικών κριτήριων. Ο αξιολογητής δηλαδή κρίνει εάν αυτά ικανοποιούνται ή όχι.

ΚΕΦΑΛΑΙΟ 2
ΕΚΤΙΜΗΣΗ ΚΟΣΤΟΥΣ ΛΟΓΙΣΜΙΚΟΥ

2.1 ΕΙΣΑΓΩΓΗ

Η βελτίωση αποτελεί την κινητήρια δύναμη στον τομέα της ερευνάς της τεχνολογίας λογισμικού και απευθύνεται στη βελτίωση των διεργασιών, των πόρων και των μεθόδων ώστε να παράγουμε και να συντηρούμε καλυτέρα τα λογισμικά προϊόντα.

Η μέτρηση αποσκοπεί στην κατανόηση, στον έλεγχο και στη βελτίωση του λογισμικού. Κάποιες από τις μετρήσεις αποσκοπούν στην κατανόηση των δραστηριοτήτων της ανάπτυξης και της συντήρησης του λογισμικού. Με τις μετρήσεις αυτές καθίστανται ορατές οι σχέσεις μεταξύ των δραστηριοτήτων και των οντοτήτων, βοηθώντας στην αξιολόγηση της τρέχουσας κατάστασης και στην θέσπιση στόχων για τη μελλοντική εξέλιξη.

2.2 ΜΕΤΡΙΚΕΣ ΛΟΓΙΣΜΙΚΟΥ

Οι μετρικές λογισμικού χωρίζονται σε κατηγορίες ανάλογα με τη φάση ανάπτυξης του λογισμικού που εφαρμόζονται [7],[8]:

- Μετρικές για το μοντέλο της ανάλυσης απαιτήσεων, που εξετάζουν την ορθότητα και την πληρότητα της ανάλυσης των απαιτήσεων, προκειμένου να γίνει ο σχεδιασμός του τελικού συστήματος.
- Μετρικές για το μοντέλο του σχεδιασμού, που αναπτύσσονται για τον έλεγχο του σχεδιασμού του τελικού συστήματος. Εστιάζουν τόσο στα τμήματα (components) του συστήματος, όσο και στις διεπαφές (interface).
- Μετρικές για τον έλεγχο (testing), που αφορούν στον έλεγχο του λογισμικού. Οι περισσότερες μετρικές εστιάζουν στη διαδικασία του ελέγχου και όχι στα τεχνικά χαρακτηριστικά. Οι μετρικές του ελέγχου προκύπτουν ουσιαστικά από τις μετρικές των τριών προηγούμενων φάσεων.
- Μετρικές για τη συντήρηση, που μετρούν το βαθμό στον οποίο είναι συντηρήσιμο το λογισμικό.
- Μετρικές για τον πηγαίο κώδικα, που αφορούν μετρήσεις πάνω στον κώδικα.

2.3 ΓΡΑΜΜΕΣ ΚΩΔΙΚΑ (LOC)

Η μετρική 'Γραμμών Κώδικα' υπολογίζει το μέγεθος ενός προγράμματος λογισμικού μετρώντας τις γραμμές του κώδικα χρησιμοποιούνται στον πηγαίο κώδικα του προγράμματος (SLOC-source lines of code) [9,23].

Χρησιμοποιείται για να προβλέψει το ποσό προσπάθειας που θα απαιτηθεί για να αναπτυχθεί ένα πρόγραμμα, καθώς επίσης και για να υπολογίσει την παραγωγικότητα προγραμματισμού ή της προσπάθειας αφού παραχθεί το λογισμικό.

Ένα αποδοτικό σχέδιο παρέχει λειτουργικότητα με χαμηλότερη προσπάθεια εφαρμογής και λιγότερες γραμμές κώδικα. Αποκλίσεις ενός μεγέθους μπορεί να είναι σαφείς δείκτες της πολυπλοκότητας λογισμικού ή των ανθρωποωρών.

Οι γραμμές κώδικα είναι ενδεχομένως η πιο ευρύτατα χρησιμοποιημένη μετρική για την μέτρηση του μεγέθους ενός προγράμματος. Φαίνεται να είναι εύκολος και ακριβής ο προσδιορισμός, εντούτοις, υπάρχουν διαφορετικοί ορισμοί για τον αριθμό γραμμών κώδικα σε ένα συγκεκριμένο πρόγραμμα.

Αυτές οι διαφορές περιλαμβάνουν την μέτρηση ή μη των κενών γραμμών και γραμμών σχολιασμού, των μη-εκτελέσιμων γραμμών, τις πολλαπλάσιες δηλώσεις ανά γραμμή, και τις πολλαπλάσιες γραμμές ανά δήλωση, καθώς επίσης και το πώς να μετρηθούν οι επαναχρησιμοποιημένες γραμμές κώδικα. Ο πιο κοινός καθορισμός των γραμμών κώδικα φαίνεται να μετρά οποιαδήποτε γραμμή που δεν είναι κενή ή γραμμή σχολίου, ανεξάρτητα από τον αριθμό δηλώσεων ανά γραμμή.

Υπάρχουν δύο σημαντικοί τύποι μέτρησης των πηγαίων γραμμών κώδικα: φυσικές πηγαίες γραμμές κώδικα και λογικές. Ο ορισμός των φυσικών γραμμών κώδικα είναι η μέτρηση των γραμμών κώδικα χωρίς να υπολογίζονται οι κενές γραμμές και οι γραμμές με σχόλια.

Οι λογικές, προσπαθούν να μετρήσουν τον αριθμό των δηλώσεων , αλλά ο ορισμός τους είναι άμεσα συνδεδεμένος με την εκάστοτε χρησιμοποιούμενη γλώσσα προγραμματισμού υπολογιστών Είναι πολύ ευκολότερο να δημιουργηθούν εργαλεία που μετρούν τις φυσικές γραμμές κώδικα, και οι ερμηνείες των φυσικών γραμμών είναι ευκολότερες στην εξήγηση. Εντούτοις, η μέτρηση τους είναι ευαίσθητη στις μορφοποιήσεις, ενώ οι λογικές γραμμές κώδικα είναι λιγότερο ευαίσθητες. Δυστυχώς, οι μετρήσεις των γραμμών κώδικα δηλώνονται συχνά χωρίς να δίνεται ο ορισμός τους δημιουργώντας σύγχυση καθώς πολλές φορές λογικές και φυσικές γραμμές κώδικα διαφέρουν σημαντικά σε αριθμό.

2.4 ΧΡΗΣΗ ΤΗΣ ΜΕΤΡΙΚΗΣ ΓΡΑΜΜΩΝ ΚΩΔΙΚΑ (Source Lines of Code)

Οι μετρήσεις των γραμμών κώδικα είναι κάπως αμφισβητούμενες, ιδιαίτερα λόγω του λάθος τρόπου που μερικές φορές χρησιμοποιούνται. Τα πειράματα έχουν επιβεβαιώσει επανειλημμένα ότι η προσπάθεια συσχετίζεται ιδιαίτερα με τις γραμμές, δηλαδή προγράμματα με μεγαλύτερο αριθμό γραμμών χρειάζονται περισσότερο χρόνο για να αναπτυχθούν. Κατά συνέπεια, οι μετρήσεις γραμμών κώδικα μπορούν να είναι πολύ αποτελεσματικές στον υπολογισμό της προσπάθειας. Εντούτοις, η λειτουργικότητα συσχετίζεται λιγότερο καλά με τον αριθμό των γραμμών: ειδικευμένοι υπεύθυνοι για την ανάπτυξη μπορούν να είναι σε θέση να αναπτύξουν την ίδια λειτουργία με πολύ λιγότερο κώδικα, έτσι ένα πρόγραμμα με λιγότερες γραμμές μπορεί να παρουσιάζει περισσότερη λειτουργικότητα από ένα άλλο παρόμοιο πρόγραμμα. Ειδικότερα, η τεχνική μέτρησης γραμμών κώδικα μπορεί να είναι μία φτωχή μετρική παραγωγικότητας του προσωπικού, δεδομένου ότι ένας υπεύθυνος για την ανάπτυξη μπορεί να αναπτύξει μόνο μερικές γραμμές και όμως να είναι πολύ παραγωγικότερος από την άποψη της λειτουργίας από έναν υπεύθυνο για την ανάπτυξη που φτιάχνει περισσότερες γραμμές (ξοδεύοντας γενικά περισσότερη προσπάθεια). Καλοί προγραμματιστές μπορούν να συγχωνέψουν πολλαπλούς κώδικες σε μία ενιαία ενότητα, βελτιώνοντας το σύστημα αν και φαίνεται να έχουν αρνητική απόδοση επειδή αφαιρούν κώδικα.

2.5 ΤΕΧΝΙΚΕΣ ΜΕΤΡΗΣΗΣ ΚΩΔΙΚΑ

Οι μετρικές κώδικα χωρίζονται σε έξι κατηγορίες ανάλογα με το πώς γίνει η μέτρηση των γραμμών κώδικα σε ένα πρόγραμμα.

Αυτές είναι:

- LOC (Lines of Code): Συνολικός αριθμός γραμμών στο αρχείο πηγαίου κώδικα

- NCLOC (Non-Commented Lines of Code): Συνολικός αριθμός γραμμών στο αρχείο του πηγαίου κώδικα που δεν είναι τεκμηρίωση (δηλ. είναι κείμενο του προγράμματος – εφαρμογής)

- CLOC (Commented Lines of Code): Συνολικός αριθμός γραμμών στο αρχείο του πηγαίου κώδικα που έχουν να κάνουν με σχόλια – τεκμηρίωση

- CD (Comment Density): Ο λόγος που ποσοτικοποιεί τον όγκο των σχολίων σε σχέση με το μέγεθος του αρχείου του πηγαίου κώδικα

- ES (Executable Statements): Αριθμός εκτελέσιμων εντολών στο αρχείο του πηγαίου κώδικα – εξαιρούνται οι headers, και τα οι ορισμοί δομών δεδομένων

- DSI (Delivered Source Instructions): Ο αριθμός των εντολών που τελικά θα παραδοθούν / παραδόθηκαν στον πελάτη / χρήστη (π.χ. δεν περιλαμβάνει κώδικα που γράφτηκε για έλεγχο, πρωτότυπα κλπ.)

2.6 ΜΕΘΟΔΟΛΟΓΙΑ ΕΚΤΙΜΗΣΗ ΚΟΣΤΟΥΣ ΛΟΓΙΣΜΙΚΟΥ

Η εκτίμηση κόστους λογισμικού είναι μία σύνθετη διεργασία που απαιτεί πολύ καλή γνώση ορισμένων χαρακτηριστικών του έργου που βρίσκεται υπό εκτίμηση. Η εκτίμηση του κόστους καλείται και ‘’παραμετρική εκτίμηση’’ γιατί είναι απαραίτητη η ακρίβεια και κατανόηση των σχέσεων μεταξύ των διακριτών παραμέτρων που μπορεί να επηρεάσουν τα αποτελέσματα του έργου [9,24].

Για να υπάρξουν ακριβής εκτιμήσεις χρειάζεται πολύ καλή γνώση παραμέτρων όπως :

- Τα μεγέθη των κύριων παραδοτέων όπως ,προδιαγραφές πηγαίος κώδικας και εγχειρίδια.
- Η πιθανότητα οι απαιτήσεις να αλλάξουν κατά την διάρκεια της ανάπτυξης του έργου.
- Ο πιθανός αριθμός σφαλμάτων ή προβλημάτων που μπορούν να εμφανιστούν.
- Οι δυνατότητες της ομάδας ανάπτυξης
- Οι μισθοί και τα έξοδα που συνδέονται με την ομάδα ανάπτυξης
- Η μεθοδολογία που πρόκειται να χρησιμοποιηθεί
- Τα εργαλεία που θα χρησιμοποιηθούν στο έργο.

Η διαδικασία για την εκτίμηση του κόστους ενός λογισμικού περιλαμβάνει μια σειρά από επαναλαμβανόμενα βήματα. Τα βήματα αυτά συχνά πραγματοποιούνται σε διαφορετική σειρά .Η ιδανικότερη όμως ακολουθία παρατίθεται παρακάτω.

2.6.1 Συγκέντρωση και ανάλυση της λειτουργικότητας του λογισμικού καθώς και των προγραμματιστικών απαιτήσεων.

Σκοπός αυτού του βήματος είναι να αναλυθούν και να βελτιωθούν οι λειτουργικές απαιτήσεις του λογισμικού και να προσδιοριστούν οι προγραμματιστικές τεχνικές, οι δεσμεύσεις και οι απαιτήσεις που θα χρησιμοποιηθούν για την εκτίμηση του κόστους του.

Χρειάζεται:

Ανάλυση των λειτουργικών απαιτήσεων του λογισμικού . Αποσαφήνιση απαιτήσεων που δεν έχουν διαλευκανθεί επαρκώς , προκειμένου να υπάρξουν οι κατάλληλες προσαρμογές κινδύνου. Οι ασαφείς απαιτήσεις είναι ένα στοιχείο κινδύνου που θα πρέπει να μεταφράζεται σε μεγαλύτερη αβεβαιότητα κατά την εκτίμηση του κόστους του λογισμικού.

Ανάλυση της φυσικής αρχιτεκτονικής του λογισμικού με βάση τις λειτουργικές απαιτήσεις . Ορισμός της αρχιτεκτονικής από την άποψη των τμημάτων λογισμικού που θα αναπτυχθούν. Η ανάλυση κάθε τμήματος γίνεται στο χαμηλότερο επίπεδο.

Ανάλυση του έργου για εντοπισμό προγραμματιστικών δεσμεύσεων συμπεριλαμβανομένων των απαιτήσεων που επιβάλλονται από τον προϋπολογισμό και τα χρονοδιαγράμματα.

Τα στοιχεία που αναμένονται να προκύψουν από αυτήν την διαδικασία είναι:

- Τεχνικές και προγραμματικές δεσμεύσεις αλλά και απαιτήσεις.
- Παραδοχές σχετικά με τους περιορισμούς και τις απαιτήσεις.
- Οι μέθοδοι που θα χρησιμοποιηθούν για την τελειοποίηση των λειτουργικών απαιτήσεων.
- Αρχιτεκτονική ιεράρχηση των τμημάτων και των σχετικών λειτουργιών του λογισμικού.

2.6.2 Προσδιορισμός των στοιχείων εργασίας και των προμηθειών.

Ο σκοπός αυτού του βήματος είναι να αναλυθούν όλα τα στοιχεία εργασίας και οι προμήθειες που θα συμμετέχουν στο έργο του λογισμικού ώστε να χρησιμοποιηθούν στην εκτίμηση του [9].

Οι συνήθεις κατηγορίες Δομικής Ανάλυσης Έργου (WBS) για ένα έργο λογισμικού είναι:

- Διαχείριση Λογισμικού (Software Management)
- Μηχανική Συστήματος Λογισμικού (Software System Engineering)
- Μηχανική Λογισμικού (Software Engineering)
- Μηχανική Ελέγχου Λογισμικού (Software Test Engineering)
- Ανάπτυξη Περιβάλλοντος Λογισμικού (Software Development Environment)

Οι κατηγορίες αυτές περιλαμβάνουν δραστηριότητες από ολόκληρο τον κύκλο ζωής του λογισμικού, από την ανάλυση απαιτήσεων έως την ολοκλήρωση και τις δοκιμές του συστήματος.

Στην συνέχεια ακολουθεί ο προσδιορισμός των χαρακτηριστικών των στοιχείων εργασίας που θα προσδιορίσουν το μέγεθος του έργου αλλά και τους κινδύνους υλοποίησής του.

Οι κίνδυνοι που προκύπτουν κατά την υλοποίηση του έργου προέρχονται από διάφορες πηγές όπως:

- Χαμηλό επίπεδο όσων αφορά την εμπειρία και την ικανότητα της ομάδας εργασίας.
- Οτιδήποτε καινούργιο όπως ο κώδικας, η γλώσσα, ή η μέθοδος σχεδιασμού.
- Υψηλά στοιχεία πολυπλοκότητας.
- Ασαφείς ή ελλιπείς απαιτήσεις.
- Η ταυτόχρονη ανάπτυξη του υλικού.
- Χαμηλά επίπεδα τεχνολογικής ετοιμότητας.
- Γεωγραφική κατανομή πολλών οργανισμών ανάπτυξης

2.6.3 Εκτίμηση για το μέγεθος του λογισμικού

Ο σκοπός αυτού του βήματος είναι να εκτιμηθεί το μέγεθος του λογισμικού .Συχνά αυτή η διαδικασία είναι δύσκολη και απαιτητική.

Η απλούστερη μετρική είναι ο αριθμός των γραμμών κώδικα (Lines of Code ή LOC) ή χιλιάδων γραμμών κώδικα (KLOC). Η μέτρηση των γραμμών κώδικα (χωρίς τα σχόλια και κενές γραμμές) γίνεται σε επίπεδο μονάδας (μεθόδου ή κλάσης) ή ακόμα και στο σύνολο του λογισμικού.

Σε επίπεδο μεθόδου είναι μία ένδειξη για την πολυπλοκότητα και την κατανόηση της μεθόδου. Σε επίπεδο του συνόλου του λογισμικού είναι ένα κριτήριο για να κατατάξουμε την πολυπλοκότητα και το μέγεθος του λογισμικού που αναπτύσσουμε.

Αν και οι γραμμές κώδικα δεν μετρούν την πολυπλοκότητα του λογισμικού, είναι προφανές ότι σχετίζονται με αυτή. Όσο αυξάνονται οι γραμμές του κώδικα, τόσο αυξάνει και η πολυπλοκότητα του λογισμικού [8,9]. [8] [9]

Τα αποτελέσματα αυτού του βήματος είναι τα εξής:

- Παραδοχές που έγιναν, προκειμένου να εκτιμηθεί το μέγεθος του λογισμικού.
- Οι μέθοδοι και οι τεχνικές που θα χρησιμοποιηθούν για την εκτίμηση του μεγέθους του.
- Συνολική εκτίμηση του μεγέθους του λογισμικού σε γραμμές κώδικα (SLOC).
- Υπολογισμός της προσπάθειας που απαιτείται για την υλοποίηση του λογισμικού.

2.6.4 Εκτίμηση για την προσπάθεια που θα απαιτηθεί η υλοποίηση του λογισμικού.

Μετατροπή της εκτίμησης στην αντίστοιχη προσπάθεια.

Ο σκοπός αυτού του βήματος είναι η μετατροπή του μέγεθος του λογισμικού στην προσπάθεια που απαιτείται για την υλοποίηση του.

Οι εκτιμήσεις που έχουν γίνει για το μέγεθος του λογισμικού χρησιμοποιούνται για να βρεθούν η κατά μήνα εργασία (WM) που χρειάζεται σύμφωνα με την Δομική Ανάλυση Έργου. Η Ανάπτυξη Λογισμικού (Software Development) περιλαμβάνει την Μηχανική Συστημάτων Λογισμικού (Software System Engineering) , την Μηχανική Συστημάτων (System Engineering) και την Μηχανική Δοκιμών Λογισμικού (Software Test Engineering) .Η μετατροπή για να υπολογίσουμε την Ανάπτυξη Λογισμικού γίνεται ως εξής:

$$Προσπάθεια\ ανάπτυξης\ λογισμικού = \frac{Εκτίμηση\ Μεγέθους}{Παραγωγικότητα\ Ανάπτυξης\ Λογισμικού}$$

Όπου,
• Στην Προσπάθεια Ανάπτυξης Λογισμικού η μέτρηση γίνεται σε εργασία ανά μήνα (WM).
• Την Παραγωγικότητα Ανάπτυξης Λογισμικού οι μονάδες είναι σε γραμμές κώδικα/μήνα .
• Η Εκτίμηση Μεγέθους μετρούμενη σε γραμμές κώδικα.

Για να βρεθεί η κατάλληλη τιμή σε γραμμές κώδικα/μήνα της παραγωγικότητας χρησιμοποιούμε στοιχεία από παλαιότερα ή παρόμοια έργα. Εάν δεν υπάρχουν τέτοια τότε μπορούμε να δεχτούμε ενδεικτικές τιμές από τον παρακάτω πίνακα.

Χαρακτηριστικά	Παραγωγικότητα Ανάπτυξης Λογισμικού (γραμμές κώδικα / μήνα)
Κλασικές τιμές	130-195
Προσεγγίσεις εξέλιξης	244-325
Νέα ενσωμάτωση λογισμικού	17-105

Πίνακας 2

Στην συνέχεια χρησιμοποιούμε ένα παράγοντα που θα προσαρμόσει τις μετρήσεις μας ανάλογα με το πόσο οικείος ή νέος είναι ο κώδικας στην ομάδα εργασίας. Αυτός ο παράγοντας ονομάζεται Δείκτης Προσπάθειας (Effort Multiplier) και τιμές του δίνονται στον πίνακα που ακολουθεί.

Κληρονομικότητα Λογισμικού	Δείκτης Προσπάθειας
Νέος σχεδιασμός και νέος κώδικας	1.2
Παρόμοιος σχεδιασμός και νέος κώδικας	1.0
Παρόμοιος σχεδιασμός και μερική επαναχρησιμοποίηση κώδικα	0.8
Παρόμοιος σχεδιασμός και εκτενής χρησιμοποίηση κώδικα	0.6

Μια από τις σημαντικότερες αιτίες της αύξησης του κόστους είναι οι αισιόδοξες προβλέψεις που μπορεί να γίνουν όσων αφορά την γνώση του κώδικα. Επομένως, οποιαδήποτε μείωση του Δείκτη Προσπάθειας θα πρέπει να γίνεται με σιγουριά.

Εάν ένα νέο έργο απαιτεί εντελώς νέο σχεδιασμό (όχι νέα τεχνολογία) και νέο κώδικα προς ανάπτυξη , τότε θα απαιτηθεί κατά μέσο όρο 20% περισσότερο προσπάθεια για την υλοποίηση του. Εάν κάποια κομμάτια του κώδικα μπορούν να επαναχρησιμοποιηθούν τότε η προσπάθεια που θα καταβάλουμε θα μειωθεί αντίθετα μία νέα τεχνολογία μπορεί να έχει ως αποτέλεσμα μία αύξηση από 50 έως 200 τις εκατό.

Τα αποτελέσματα που θα λάβουμε από αυτήν την διαδικασία είναι τα εξής:

- Οι υποθέσεις που έγιναν με σκοπό την εκτίμηση του παράγοντα Προσπάθειας Ανάπτυξης Λογισμικού.
- Οι μέθοδοι που χρησιμοποιούνται για την εκτίμηση της Προσπάθειας Ανάπτυξης Λογισμικού
- Συνολική προσπάθεια ανάπτυξης λογισμικού σε εργασία/μήνα.

Προέκταση και ολοκλήρωση του υπολογισμού του συντελεστή προσπάθειας.

Ο σκοπός αυτού του βήματος είναι να επεκτείνει τις εκτιμήσεις για να καλύψει όλα τα στοιχεία εργασίας του έργου δομικής ανάλυσης. Μέχρι και αυτό το βήμα, οι εκτιμήσεις έχουν καλύψει μόνο την Ανάπτυξη Λογισμικού (Μηχανική Συστήματος Λογισμικού ,Μηχανική Λογισμικού, Μηχανική Ελέγχου Λογισμικού).

Επιπλέον όμως στον παράγοντα Προσπάθειας Ανάπτυξης Λογισμικού θα πρέπει να συνυπολογιστεί και η προσπάθειας Διαχείρισης Λογισμικού καθώς και ο παράγοντας Εξασφάλισης Ποιότητας Λογισμικού.

Χρησιμοποιώντας τον παρακάτω πίνακα μαζί με ΔΑΕ μπορούμε να βρούμε προστιθέμενο ποσοστό στον Παράγοντα Ανάπτυξης.

Κατηγορία ΔΑΕ	Ποσοστό του παράγοντα Προσπάθειας Ανάπτυξης Λογισμικού %
Διαχείριση Λογισμικού	Επιπλέον 6-27%
System-level Test Support (includes SW Development Test-bed, SW System-level Test Support, ATLO Support)	Επιπλέον 34 - 112%
Εξασφάλιση Ποιότητας Λογισμικού	Επιπλέον 6 - 11 %
Επιπρόσθετες δραστηριότητες	
Διόρθωση Διαχείρισης Έργου	Επιπλέον 3 - 6 %
Διαχείριση Έργου	Επιπλέον 8 - 11 %
Απόκτηση διαχείρισης	Επιπλέον 11 - 22 %
Επανάληψη εργασίας	Επιπλέον 17 - 22 %
Συντήρηση πέντε πρώτων χρόνων	Επιπλέον 22% στον παράγοντα Ανάπτυξης Λογισμικού για κάθε χρόνο συντήρησης.

Συνυπολογίζοντας τώρα τις προσαυξήσεις από κάθε κατηγορία της ΔΑΕ στην Συνολική Προσπάθεια Ανάπτυξης Λογισμικού βρίσκουμε την Συνολική Προσπάθεια Λογισμικού.

Τα αποτελέσματα αυτού του βήματος είναι:

- Οι παραδοχές που έγιναν για να γίνει η εκτίμηση της Συνολικής Προσπάθειας Λογισμικού.
- Οι μέθοδοι που χρησιμοποιούνται για να γίνει η συνολική εκτίμηση της Συνολικής Προσπάθειας Λογισμικού.
- Πλήρης υπολογισμός για όλα τα στοιχεία εργασίας της ΔΑΕ (σε εργασία/μήνα).
- Εκτίμηση της Συνολικής Προσπάθειας Λογισμικού.

2.6.5 Χρονοπρογραμματισμός του έργου.

Ο σκοπός αυτού του βήματος είναι να καθοριστεί το χρονικό διάστημα που απαιτείται για την ολοκλήρωση του έργου και να καθοριστούν οι περίοδοι του χρόνου για κάθε συμβάν εργασίας της ΔΑΕ.

Ο χρονοπρογραμματισμός γίνεται ως εξής :

- Δίνεται ο αιτούμενος χρόνος για κάθε στοιχείο εργασίας της ΔΑΕ και αποφασίζεται ο φόρτος εργασίας .Προτεινόμενη τακτική είναι να αφήνεται ένας μήνα περιθώριο για κάθε έτος υλοποίησης του έργου.

- Καθορισμός της σειράς των στοιχείων εργασίας. Ορισμός ποιων εργασιών μπορούν να γίνουν παράλληλα, καθώς και των εξαρτημένων εργασιών του χρονοδιαγράμματος.

- Στηριζόμενοι στον γενικότερο σχεδιασμό του έργου προσπαθούμε να προγραμματίσουμε τις εργασίες ξεκινώντας από τα δύο άκρα. Ξεκινώντας από την αρχή δημιουργούμε μία δραστηριότητα που να μας φανερώνει την συσχέτιση των στοιχείων εργασίας.

- Έπειτα στην καταληκτική ημερομηνία δημιουργούμε παρόμοια εργασία και δουλεύοντας από το τέλος προς την αρχή βλέπουμε αν οι εργασίες μπορούν να ολοκληρωθούν.

- Προσδιορισμός της κρίσιμης διαδρομής.

- Διόρθωση του φόρτου εργασίας στα επίπεδα του κρίσιμης διαδρομής.

Μπορούν να εμφανιστούν ασυνέπειες και τρύπες στον προγραμματισμό προσπαθώντας να ταιριάξουν τα work elements στους καθορισμένους πόρους. Ως εκ τούτου, μπορεί να χρειαστεί να επαναληφθούν οι εκτιμήσεις των άλλων βημάτων αρκετές φορές ώστε να μειωθεί η προσπάθεια ή να γίνουν νέες εκτιμήσεις λαμβάνοντας υπόψη μεγαλύτερο ρίσκο.

Οι έξοδοι που θα προκύψουν μετά την ολοκλήρωση αυτού του βήματος θα είναι:

- Οι παραδοχές που έγιναν για την εκτίμηση χρονοδιαγράμματος.
- Ημερομηνίες ορόσημα, σχόλια και ο γενικότερος χρονοπρογραμματισμός όλων των στοιχείων εργασίας της ΔΑΕ.
- Οι νέες παραδοχές που έγιναν ώστε να αναθεωρηθούν οι εκτιμήσεις.

2.6.6 Διαδικασία υπολογισμού συνολικού κόστους

Ο σκοπός αυτού του βήματος είναι να υπολογιστεί το συνολικό κόστος του έργου σύμφωνα με τις ανάγκες των στοιχείων εργασίας και των προμηθειών που έχουν οριστεί στην Δομική Ανάλυση Έργου.

- Υπολογισμός του κόστους προμηθειών.
- Προσδιορισμός του κόστους υπηρεσιών, όπως οι σταθμοί εργασίας, δοκιμές , εξομοιωτές, δίκτυα και τηλεφωνικές χρεώσεις.
- Προσδιορισμός του κόστους όσων αφορά τα λογισμικά μέρη όπως τα λειτουργικά συστήματα, μεταγλωττιστές, άδειες, καθώς και εργαλεία ανάπτυξης.

- Προσδιορισμός του κόστους ταξιδιών που σχετίζονται με κριτικές πελατών και διασυνδέσεις καθώς και συμμετοχές σε σχετικά με το έργο συνέδρια.
- Υπολογισμός του κόστους για την εκπαίδευση των ατόμων που σχετίζονται με το έργο.
- Καθορισμός του μισθού και του επιπέδου εξειδίκευσης του εργατικού δυναμικού.

Ενώ υπολογίζουμε το κόστος αυτό είναι πολύ πιθανό να εμφανιστούν προβλήματα και τρύπες στον προϋπολογισμό μας. Ως αποτέλεσμα, μπορεί να είναι απαραίτητο να επαναληφθούν οι εκτιμήσεις των άλλων βημάτων αρκετές φορές, ώστε να μειωθεί η προσπάθεια ή οι προμήθειες. Ακόμα για την μείωση του κόστους είναι δυνατό να αυξήσουμε το ποσοστό του κινδύνου του έργου.

Τα αποτελέσματα αυτού του βήματος είναι τα εξής:

- Οι παραδοχές που έγιναν για την εκτίμηση του κόστους.
- Οι μέθοδοι που χρησιμοποιούνται για τον υπολογισμό του κόστους.
- Το κόστος των προμηθειών.
- Η συνολική εκτίμηση του κόστους (σε χρηματικές μονάδες).

2.6.7 Προσδιορισμός των επιπτώσεων των κινδύνων.

Ο σκοπός αυτού του βήματος είναι να εντοπιστούν οι κίνδυνοι του έργου, και να αξιολογηθούν οι επιπτώσεις τους στο συνολικό κόστος του έργου.

Η αξιολόγηση των κινδύνων γίνεται ως εξής:

- Από προηγούμενα βήματα προσδιορίζουμε τις εργασίες που παρουσιάζουν το μεγαλύτερο ρίσκο, τις μεγαλύτερες επιπτώσεις ή την πιο μεγάλη αβεβαιότητα ως προς τις εκτιμήσεις για την υλοποίηση του λογισμικού.

- Γίνεται εκτίμηση του επιπλέον κόστους που θα χρειαστεί για την εξάλειψη αυτών των κινδύνων.

- Εκτίμηση του προσαρμοστικού παράγοντα ρίσκου.

- Προσαρμογή κάθε άλλης εκτίμησης που βασίζεται στην εκτίμηση των κινδύνων.

- Ανανέωση της εκτίμησης επικινδυνότητας κάθε φορά που ενημερώνονται οι εκτιμήσεις για το έργο.

Τα αποτελέσματα αυτού του βήματος είναι:

- Η αναλυτική λίστα κινδύνων του έργου.
- Οι παραδοχές που έγιναν για την αναθεώρηση των εκτιμήσεων.
- Οι μέθοδοι που χρησιμοποιούνται για την αναθεώρηση εκτιμήσεων
- Αναθεώρηση του μεγέθους, της προσπάθειας, του χρονοδιαγράμματος, και των γενικότερων εκτιμήσεων του κόστους.

2.6.8 Επικύρωση των εκτιμήσεων.

Εκτός από την κύρια εκτίμηση που αναπτύχθηκε στα προηγούμενα βήματα, γίνεται και μία δευτερεύουσα εκτίμηση, χρησιμοποιώντας ένα από τα ακόλουθα:

α. Εναλλακτική εκτίμηση.

Χρησιμοποίηση ενός δεύτερου ατόμου ή ομάδας, με παρόμοιες εμπειρίες σε υλοποίηση λογισμικού, ώστε να παράγει ανεξάρτητες εκτιμήσεις.

β. Ιστορικές αναλογίες

Χρησιμοποιώντας τα ιστορικά δεδομένα γίνεται σύγκριση των εκτιμήσεων. Τέτοιες συγκρίσεις μπορεί να είναι:

Το μέγεθος και η προσπάθεια, παρόμοιων λογισμικών, το μέγεθος σε σχέση με τις λειτουργίες αλλά και η προσπάθεια συγκρινόμενη με το κόστος .

γ. Εκτιμήσεις στηριζόμενες σε μοντέλα.

Κάνοντας αυτό το βήμα επιτυγχάνουμε να έχουμε ένα νέο και αναθεωρημένο size του project νέες εκτιμήσεις για την προσπάθεια , το χρονοδιάγραμμα αλλά και βελτιωμένη ακρίβεια του συνολικού κόστους.

2.6.9 Εναρμόνιση των εκτιμήσεων του προϋπολογισμού και του χρονοπρογράμματος.

Ο σκοπός αυτού του βήματος είναι η επανεξέταση και επικύρωση των εκτιμήσεων σε συσχέτιση με τον προϋπολογισμό και το χρονοδιάγραμμα.

Υπολογισμός του περιθωρίου του προϋπολογισμού. Γίνεται αφαίρεση του εκτιμώμενου κόστους από το προϋπολογισθέν κόστος. Στη συνέχεια διαιρώντας με το προϋπολογισμένο βρίσκουμε το περιθώριο κόστους. Με τον ίδιο τρόπο βρίσκουμε τα περιθώρια στον χρονοπρογραμματισμό.

Συγκρίνουμε το εκτιμώμενο κόστος ,τον προγραμματισμό που έχουμε κάνει και τα περιθώρια που έχουμε δώσει με τις απαιτήσεις του έργου.

Εάν οι αποκλίσεις είναι πολύ μεγάλες ξεκινάμε προσπάθεια για εντοπισμό και επίλυση των διαφορών.

Προσπάθεια για εξάλειψη προμηθειών που δεν είναι απολύτως απαραίτητες.

Προσπάθεια για αναθεώρηση του χρονοδιαγράμματος, των εκτιμήσεων του κόστους, και των κινδύνων.
Επανάληψη της διαδικασίας έως ότου να υπάρξει ταίριασμα του budget σε σχέση με τον προϋπολογισμό και το χρονοδιάγραμμα.

Καθώς το έργο προχωρά είναι δυνατό να συμπεριληφθούν νέες συμβάσεις ή λειτουργίες που αρχικά δεν είχαν συμπεριληφθεί ή δεν ήταν οικονομικά προσιτές σύμφωνα με τις αρχικές εκτιμήσεις.

2.6.10 Εξέταση και έγκριση των εκτιμήσεων.

Σκοπός αυτού του βήματος είναι να ελεγχθούν οι εκτιμήσεις του έργου και να αναθεωρηθούν ώστε να εγκριθούν από την διοίκηση.

Αξιολόγηση εάν έχουν επιτευχθεί οι στόχοι:

- Επιβεβαίωση της δομικής ανάλυσης έργου και της αρχιτεκτονικής λογισμικού.
- Επιβεβαίωση για την ορθότητα των μεθόδων που χρησιμοποιούνται για την εξαγωγή του μεγέθους του έργου, της προσπάθειας, του χρονοδιαγράμματος και του κόστους.
- Επαλήθευση για τα δεδομένα εισόδου που χρησιμοποιούνται στην ανάπτυξη.
- Επίσημη επιβεβαίωση και καταγραφή των εκτιμήσεων.

Ακολουθεί η έγκριση του διαχειριστή λογισμικού, των εκτιμητών λογισμικού, και της διαχείρισης του έργου για την έγκριση του έργου.

2.6.11 Καταγραφή των εκτιμήσεων

Γίνεται προσδιορισμός για το πόσο και εάν το έργο προπορεύεται ή έπεται των εκτιμήσεων. Σύγκριση των δεδομένων με παλαιότερες εκτιμήσεις ώστε να προσδιοριστή η αλλαγή τους με την πάροδο του χρόνου.

Αρχειοθέτηση δεδομένων όπως:

- Στοιχεία έργου (όνομα, πλατφόρμα, μέθοδοι εκτιμήσεων, ημερομηνίες ορόσημα)
- Αναμενόμενα και πραγματικά μεγέθη, του κόστους και της προσπάθειας των εργασιών της δομικής ανάλυσης έργου.
- Προβλεπόμενες και πραγματικές ημερομηνίες του χρονοδιαγράμματος.
- Οι εκτιμώμενοι κίνδυνοι και οι πραγματικές επιδράσεις που είχαν στο έργο.

Έτσι με το τελικό αυτό βήμα επιτυγχάνεται να ελεγχθεί η ακρίβεια του λογισμικού με την πάροδο του χρόνου και να καταγραφούν οι εκτιμήσεις για χρήση σε παρόμοια έργα .

ΚΕΦΑΛΑΙΟ 3
ΤΕΧΝΙΚΕΣ ΕΚΤΙΜΗΣΗΣ ΚΟΣΤΟΥΣ ΛΟΓΙΣΜΙΚΟΥ

3.1 ΕΙΣΑΓΩΓΗ

Η εκτίμηση κόστους ενός λογισμικού (Software Cost Estimation) είναι μια ιδιαίτερα κρίσιμη δραστηριότητα που πρέπει να ολοκληρωθεί στα πρώτα στάδια της διαδικασίας παραγωγής λογισμικού. Το αντικείμενό της είναι η πρόβλεψη είτε της ανθρώπινης προσπάθειας που καταβάλλεται (effort) είτε του χρηματικού κόστους που απαιτείται για την ολοκλήρωση του έργου . Σε πολλές περιπτώσεις υπάρχει ενδιαφέρον και για την πρόβλεψη της παραγωγικότητας (productivity) , δηλαδή του λόγου του μεγέθους του παραγόμενου κώδικα προς την προσπάθεια που καταβάλλεται (μέγεθος/ προσπάθεια).

3.2 ΜΟΝΤΕΛΑ ΕΚΤΙΜΗΣΗΣ ΚΟΣΤΟΥΣ

Η αυξανόμενη ζήτηση για λογισμικό εντός των προκαθορισμένων χρονοδιαγραμμάτων αλλά και η ραγδαία εξάπλωση της ποικιλομορφίας και της πολυπλοκότητας των εφαρμογών οδηγούν στην αναζήτηση νέων και πιο αποτελεσματικών τεχνικών εκτίμησης του κόστους.

3.2.1 Εμπειροτεχνικά (Expert judgment)

Η εμπειροτεχνική μέθοδος περιλαμβάνει την ανάμειξη ειδικών οι οποίοι χρησιμοποιούν την εμπειρία τους και τις γνώσεις τους πάνω σε παρόμοια έργα ώστε να καταλήξουν σε μία εκτίμηση του κόστους.
Κάθε εμπειρογνώμον εκφράζει την άποψη του για το κόστος του έργου. Η διαδικασία επαναλαμβάνεται έως ότου βρεθεί μία κοινή λύση από την οποία να ικανοποιούνται οι περισσότεροι από αυτούς.
Το πλεονέκτημα αυτής της μεθόδου είναι ότι έχει πολύ χαμηλό κόστος το οποίο μπορεί να συνδυαστεί και με μεγάλη ακρίβεια πρόβλεψης στην περίπτωση που οι εμπειρογνώμονες έχουν μελετήσει παρόμοια συστήματα.
Αντίθετα η μέθοδος κρίνεται ως αναξιόπιστη εάν δεν υπάρχουν οι κατάλληλοι εμπειρογνώμονες ενώ είναι δύσκολο να ποσοτικοποιηθούν και να επικυρωθούν οι εκτιμήσεις.[10]

3.2.2 Κατά αναλογία (Estimation by analogy)

Κατά την τεχνική αυτή γίνεται εκτίμηση του κόστους κατά αναλογία με προηγούμενα παρόμοιου μεγέθους έργα στο ίδιο πεδίο εφαρμογής.
Ο υπολογισμός κατά αναλογία μπορεί να γίνει είτε στο συνολικό επίπεδο του έργου είτε σε επίπεδο υποσυστήματος. Το κύριο πλεονέκτημα της μεθόδου κατά αναλογία είναι ότι η εκτίμηση βασίζεται σε πραγματικά δεδομένα και εάν το συγκρινόμενο σύστημα έχει αρκετές ομοιότητες με το πραγματικό μπορεί να επιτευχθεί μεγάλη ακρίβεια στις εκτιμήσεις. Για την υλοποίηση αυτής της

μεθόδου χρειάζεται συστηματική καταγραφή των δεδομένων κόστους για κάθε έργο που τελειώνει και δεν μπορεί να εφαρμοστεί εάν δεν υπάρχουν ιστορικά στοιχεία .[11] [20].

3.2.3 Εκτίμηση κόστους με τον νόμο του Parkinson (Parkinson's Low)

Κατά την μέθοδο αυτή το κόστος δεν καθορίζεται από αντικειμενικές εκτιμήσεις αλλά καθορίζεται από τους διαθέσιμους πόρους [11]. Σε αρκετές περιπτώσεις εκτιμήσεις με την μέθοδο Parkinson έχουν καταλήξει εξαιρετικά ακριβείς Αυτές είναι γενικές περιπτώσεις στις οποίες συνήθως υπήρχαν αρκετοί πόροι και το χρονοδιάγραμμα ήταν ελαστικό. Ακόμα και σε αυτές τις περιπτώσεις όμως δεν είναι σαφές εάν έγινε η καλύτερη αξιοποίηση του ανθρώπινου δυναμικού. Αντίθετα σε περιορισμένου χρόνου και προϋπολογισμού έργα πολλές φορές η εφαρμογή δεν παραδίδεται ολόκληρη ή δεν ολοκληρώνεται ποτέ. Για αυτούς τους λόγους η Εκτίμηση κόστους με τον νόμο του Parkinson μέθοδος θεωρείται μη αποτελεσματική.

3.2.4 Με σκοπό την κατοχύρωση του έργου (Pricing to win)

Η pricing to win είναι μια μέθοδος με κύριο προσανατολισμό την κατοχύρωση του έργου. Σε πολλές περιπτώσεις οι πόροι εξαντλούνται ή το χρονοδιάγραμμα φτάνει στο τέλος του πριν το έργο πάρει την τελική του μορφή. Τότε ξεκινούν αμοιβαίοι συμβιβασμοί ή πολύωρες εργασίες από τους προγραμματιστές προκειμένου να έχουν παραδοτέο το έργο. Έτσι, η πιθανότητα ο πελάτης να παραλάβει την εφαρμογή όπως την ήθελε είναι χαμηλή. Ο κυριότερος λόγος για τον οποίο χρησιμοποιούνται ακόμα τεχνικές όπως η κατοχύρωση είναι ότι η τεχνολογία εκτίμησης κόστους λογισμικού δεν έχει καλύψει και πείσει απόλυτα τους ακόμα τους διαχειριστές έργων.

3.2.5 Εκτίμηση με τη χρήση αλγοριθμικών μοντέλων κόστους – (Algorithmic cost Modeling)

Τα αλγοριθμικά μοντέλα εκτίμησης κόστους βασίζονται σε ιστορικές πληροφορίες κόστους και συσχετίζουν κάποια μετρική λογισμικού με το κόστος του έργου. Μία άλλη διάκριση που μπορεί να γίνει ανάμεσα στα μοντέλα εκτίμησης κόστους είναι σε αυτά που χρησιμοποιούν μέτρηση γραμμών κώδικα (SLIM, Δομικό Μοντέλο) σαν πρωταρχική είσοδο και σε αυτά που δεν χρησιμοποιούν (ESTIMACS, Λειτουργικά Σημεία).

3.2.6 Η μετρική SLIM

Η μέθοδος εκτίμησης SLIM αναπτύχθηκε στα τέλη της δεκαετίας του 1970 από τον Larry Putnam. Η εκτιμήσεις του μεγέθους του λογισμικού στην SLIM εξαρτώνται από μετρήσεις σε γραμμές κώδικα, έπειτα με χρήση του μοντέλου Rayleigh γίνονται τροποποιήσεις ώστε να εκτιμηθεί η παραγόμενη

προσπάθεια. Ο χρήστης μπορεί να επηρεάσει το σχήμα της καμπύλης μέσα από δύο βασικές παραμέτρους: τον δείκτη συσσώρευσης εργατικού δυναμικού και την σταθερά τεχνολογίας. Η επιλογή αυτών τιμών μπορεί να γίνει είτε επιλέγοντας εισαγωγή δεδομένων από παλαιότερα ολοκληρωμένα έργα είτε απαντώντας μία σειρά ερωτήσεων από όπου η SLIM θα επιλέξει τις καταλληλότερες τιμές. [12]

3.2.7 Η μετρική ESTIMACS

Το μοντέλο ESTIMACS αναπτύχθηκε από Howard Rubin [12] του Hunter College.
Μόλις ολοκληρώθηκε πωλήθηκε αμέσως στην Management and Computer Services (MACS) η οποία και κατέχει τα αποκλειστικά δικαιώματά του. Το μοντέλο αυτό δεν απαιτεί γραμμές κώδικα για τιμές εισόδου καθώς λειτουργεί περισσότερο ως μετρική λειτουργικών σημείων.

3.3 ΔΟΜΙΚΟ ΜΟΝΤΕΛΟ ΚΟΣΤΟΥΣ (COCOMO-Constructive Cost Model)

Το Δομικό Μοντέλο Κόστους είναι ευρέως γνωστό και αποτελεί τον κυριότερο εκφραστή των αλγοριθμικών μοντέλων. Είναι ένα μοντέλο εκτίμησης κόστους λογισμικού το οποίο στηρίζεται στην μελέτη πολλών έργων πληροφορικής και χρησιμοποιείται από πολλούς διαχειριστές έργων λογισμικού σε όλο τον κόσμο [13]. Η αρχή γίνεται το 1981 από τον Dr. Barry Boehm με το Δομικό Μοντέλο 81. Ακολουθούν βελτιώσεις του με το Ada Δομικό Μοντέλο και το Δομικό Μοντέλο II.
Στο μοντέλο αυτό γίνεται εκτίμηση της προσπάθειας και της διάρκειας ενός έργου στηριζόμενοι σε εισόδους σχετικές με το μέγεθος των συστημάτων που θα δημιουργηθούν αλλά και σε ορισμένους συντελεστές κόστους που επηρεάζουν την παραγωγικότητα.
Ο βασικότερος υπολογισμός του Δομικού Μοντέλου είναι η χρήση της Συνάρτησης Προσπάθειας (Effort Equation) για την εκτίμηση του αριθμού των Ανθρωπομηνών που απαιτούνται για να αναπτυχθεί ένα έργο.
Κύριο χαρακτηριστικό του αποτελεί το γεγονός ότι είναι ένα ανοιχτό μοντέλο εκτίμησης κόστους με συνέπεια όλοι οι αλγόριθμοι του να είναι δημοσιευμένοι αλλά και οι επαφές του να είναι καλά προσδιορισμένες και παραμετροποιημένες.

3.3.1 ΠΡΟΤΥΠΑ

Το Δομικό Μοντέλο Κόστους χωρίζεται σε τρία πρότυπα [13,22]. Αυτά είναι:

1) Βασικό Δομικό Μοντέλο: Είναι ένα στατικό πρότυπο που υπολογίζει την προσπάθεια ανάπτυξης λογισμικού (και το κόστος) ως λειτουργία του μεγέθους προγράμματος που εκφράζεται στις κατ' εκτίμηση γραμμές κώδικα (LOC).

2) Μεσαίο ή Ενδιάμεσο Δομικό Μοντέλο: Υπολογίζει την προσπάθεια ανάπτυξης λογισμικού ως λειτουργία του μεγέθους προγράμματος και σύνολο "οδηγών δαπανών" που περιλαμβάνουν τις υποκειμενικές αξιολογήσεις του προϊόντος, του υλικού, του προσωπικού, και των ιδιοτήτων του έργου.

3) Προηγμένο Δομικό Μοντέλο: Ενσωματώνει όλα τα χαρακτηριστικά του μεσαίου με μια αξιολόγηση του αντίκτυπου του οδηγού δαπανών σε κάθε βήμα (ανάλυση, σχέδιο, κλπ.).

3.3.2 Βασικό Δομικό Μοντέλο

Το Βασικό Δομικό Μοντέλο μπορεί να εφαρμοστεί σε τρεις κατηγορίες προγραμμάτων λογισμικού.

• Οργανικά προγράμματα : είναι σχετικά μικρά, απλά προγράμματα λογισμικού στα οποία οι μικρές ομάδες με εμπειρία εφαρμογής εργάζονται σε ένα σύνολο λιγότερο άκαμπτων απαιτήσεων.
• Ημί-αποσπασμένα προγράμματα : είναι ενδιάμεσα (στο μέγεθος και την πολυπλοκότητα) προγράμματα λογισμικού στα οποία οι ομάδες με τα μικτά επίπεδα εμπειρίας πρέπει να ικανοποιήσουν ένα μίγμα άκαμπτων και λιγότερο άκαμπτων απαιτήσεων.
• Ενσωματωμένα προγράμματα : είναι προγράμματα λογισμικού που πρέπει να αναπτυχθούν μέσα σε ένα σφιχτό σύνολο, πλαίσιο υλικών, λογισμικών, και λειτουργικών περιορισμών.

Εξισώσεις και παράμετροι που χρησιμοποιούνται στο Βασικό Δομικό Μοντέλο

MM=α*KDSIb	KDSI: χιλιάδες γραμμές κώδικα
TDEV=2.5*MMc	TDEV: ο εκτιμώμενος χρόνος ανάπτυξης

Η βασική εξίσωση έχει τη μορφή

E = ab * (KLOC) bb
D= cb * (Effort) db
P=E/D

Όπου

E	Η προσπάθεια σε άνθρωπο-μήνες
D	Ο χρόνος ανάπτυξης σε μήνες
P	Ο αριθμός των ανθρώπων που απαιτούνται
KLOC	Ο κατ' εκτίμηση αριθμός παραδοτέων γραμμών κώδικα για το πρόγραμμα (εκφρασμένος σε χιλιάδες)

Οι συντελεστές δίνονται από τον παρακάτω πίνακα:

	ΣΥΝΤΕΛΕΣΤΕΣ			
Πρόγραμμα λογισμικού	ab	bb	cb	db
Οργανικό	2.4	1.05	2.5	0.38
Ημί–αποσυνδεμένο	2.5	1.12	2.5	0.35
Ενσωματωμένο	3.6	20	2.5	0.32

Εξίσωση υπολογισμού απαιτούμενης ονομαστικής προσπάθειας	MMNOM = C (KDSI)K	MMNOM: ονομαστική προσπάθεια σε ανθρωπομήνες. C, K: παράμετροι,
Εξίσωση υπολογισμού προσπάθειας ανάπτυξης	MMDEV = q *MMNOM	q: παράμετρος
Εξίσωση υπολογισμού κόστους	Ct = p *MMDEV	p: αξία σε χρήμα ενός ανθρωπομήνα
Εξίσωση υπολογισμού διάρκειας του έργου	TDEV = R *(MMDEV)*m	R, m: παράμετροι

Το Βασικό Δομικό Μοντέλο είναι καλό για γρήγορες εκτιμήσεις αλλά η ακρίβειά του είναι αναγκαστικά περιορισμένη λόγω της έλλειψης παραγόντων που αφορούν στους περιορισμούς υλικού, την ποιότητα και εμπειρία προσωπικού.

3.3.3 Μεσαίο Δομικό Μοντέλο

Το Μεσαίο Δομικό Μοντέλο είναι μια επέκταση του βασικού προτύπου δομικών μοντέλων , και χρησιμοποιείται για να υπολογίσει το χρόνο των προγραμματιστών να αναπτύξουν ένα προϊόν λογισμικού. Προσπαθώντας να γίνει βελτίωση της ακρίβειας του μοντέλου έγινε προσθήκη 15 συντελεστών κόστους (cost drivers).
Αυτή η επέκταση εξετάζει ένα σύνολο "ιδιοτήτων οδηγών δαπανών" που μπορεί να ομαδοποιηθεί σε τέσσερις σημαντικές κατηγορίες, κάθε μια με διαφορετικές υποκατηγορίες:

Οδηγοί Δαπανών (cost drivers)												
Ιδιότητες προσωπικού			**Ιδιότητες υλικού**			**Ιδιότητες προϊόντων**			**Ιδιότητες προγράμματος**			
Ικανότητα αναλυτών	Πολύ Χαμηλό	1.16	*Περιορισμοί χρόνου εκτέλεσης*	Πολύ Χαμηλό	x	*Απαραίτητη αξιοπιστία λογισμικού*	Πολύ Χαμηλό	0.75	*Χρήση εργαλείων λογισμικού*	Πολύ Χαμηλό	1.24
	Χαμηλό	1.19		Χαμηλό	x		Χαμηλό	0.88		Χαμηλό	1.10
	Μέσο	1.00		Μέσο	1.00		Μέσο	1.00		Μέσο	1.00
	Υψηλό	0.86		Υψηλό	1.11		Υψηλό	1.15		Υψηλό	0.91
	Πολύ Υψηλό	0.71		Πολύ Υψηλό	1.30		Πολύ Υψηλό	1.40		Πολύ Υψηλό	0.82
	Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	1.66		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x
Ικανότητα μηχανικών λογισμικού	Πολύ Χαμηλό	1.29	*Περιορισμοί μνήμης*	Πολύ Χαμηλό	x	*Μέγεθος της βάσης δεδομένων εφαρμογής*	Πολύ Χαμηλό		*Εφαρμογή των μεθόδων τεχνολογίας λογισμικού*	Πολύ Χαμηλό	1.24
	Χαμηλό	1.13		Χαμηλό	x		Χαμηλό	0.94		Χαμηλό	1.10
	Μέσο	1.00		Μέσο	1.00		Μέσο	1.00		Μέσο	1.00
	Υψηλό	0.91		Υψηλό	1.06		Υψηλό	1.08		Υψηλό	0.91
	Πολύ Υψηλό	0.82		Πολύ Υψηλό	1.21		Πολύ Υψηλό	1.16		Πολύ Υψηλό	0.82
	Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	1.56		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x
Εμπειρία εφαρμογών	Πολύ Χαμηλό	1.42	*Αστάθεια του εικονικού περιβάλλοντος*	Πολύ Χαμηλό		*Πολυπλοκότητα του προϊόντος*	Πολύ Χαμηλό	0.70	*Απαραίτητο πρόγραμμα ανάπτυξης*	Πολύ Χαμηλό	1.23
	Χαμηλό	1.17		Χαμηλό	0.87		Χαμηλό	0.88		Χαμηλό	1.08
	Μέσο	1.00		Μέσο	1.00		Μέσο	1.00		Μέσο	1.00
	Υψηλό	0.91		Υψηλό	1.15		Υψηλό	1.15		Υψηλό	1.04
	Πολύ Υψηλό	0.82		Πολύ Υψηλό	1.30		Πολύ Υψηλό	1.40		Πολύ Υψηλό	1.10
	Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x
Εικονική εμπειρία μηχανών	Πολύ Χαμηλό	1.21	*Απαραίτητος turnabout χρόνος*	Πολύ Χαμηλό			Πολύ Χαμηλό	x		Πολύ Χαμηλό	x
	Χαμηλό	1.10		Χαμηλό	0.87		Χαμηλό	x		Χαμηλό	x
	Μέσο	1.00		Μέσο	1.00		Μέσο	x		Μέσο	x
	Υψηλό	0.90		Υψηλό	1.07		Υψηλό			Υψηλό	
	Πολύ Υψηλό	x		Πολύ Υψηλό	1.15		Πολύ Υψηλό	x		Πολύ Υψηλό	x
	Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x
Γλωσσική εμπειρία προγραμματισμού	Πολύ Χαμηλό	1.14		Πολύ Χαμηλό	x		Πολύ Χαμηλό	x		Πολύ Χαμηλό	x
	Χαμηλό	1.07		Χαμηλό	x		Χαμηλό	x		Χαμηλό	x
	Μέσο	1.00		Μέσο	x		Μέσο	x		Μέσο	x
	Υψηλό	0.95		Υψηλό	x		Υψηλό	x		Υψηλό	x
	Πολύ Υψηλό	x		Πολύ Υψηλό	x		Πολύ Υψηλό	x		Πολύ Υψηλό	x
	Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x		Πάρα πολύ Υψηλό	x

Με βάση την εκτίμηση, ένας πολλαπλασιαστής προσπάθειας καθορίζεται από έναν καθορισμένο πίνακα. Το προϊόν όλων των πολλαπλασιαστών προσπάθειας οδηγεί σε έναν "παράγοντα ρύθμισης προσπάθειας (EAF). Οι χαρακτηριστικές τιμές για το EAF κυμαίνονται από 0,9 έως 1,4.

Έτσι ο τύπος για το Μεσαίο Δομικό Μοντέλο γίνεται:
E=ai(KLOC)bi*EAF με τους συντελεστές ai και bi να δίνονται:

Πρόγραμμα λογισμικού	ai	bi
Οργανικό	3.2	1.05
Ημί–αποσυνδεμένο	3.0	1.12
Ενσωματωμένο	2.8	1.20

3.3.4 Προχωρημένο Δομικό Μοντέλο

Το Λεπτομερειακό Δομικό Μοντέλο ενσωματώνει όλα τα χαρακτηριστικά του ενδιάμεσου Δομικού Μοντέλου με μια αξιολόγηση του αντίκτυπου του οδηγού δαπανών σε κάθε βήμα (ανάλυση, σχέδιο, κ.λπ.) από τη διαδικασία ανάπτυξης λογισμικού. Το Λεπτομερειακό Δομικό Μοντέλο ονομάζεται επίσης Ada Δομικό Μοντέλο, Boehm, και Royce (1989).
Το Λεπτομερειακό μοντέλο διαφέρει από το Ενδιάμεσο μοντέλο σε ένα βασικό χαρακτηριστικό: Το Λεπτομερειακό μοντέλο χρησιμοποιεί διαφορετικούς Πολλαπλασιαστές Προσπάθειας για κάθε φάση του έργου. Οι εξαρτώμενοι από τη φάση πολλαπλασιαστές προσπάθειας παράγουν καλύτερες εκτιμήσεις από ότι το Ενδιάμεσο μοντέλο.

Το λεπτομερειακό μοντέλο έχει έξι φάσεις υλοποίησής του. Αυτές είναι:

- Απαιτήσεις Χρήστη
- Σχεδιασμό του έργου
- Λεπτομερειακό Σχεδιασμό
- Ανάπτυξη του έργου και Έλεγχος εγκυρότητας
- Ενοποίηση και τελικός έλεγχος.
- Συντήρηση

Οι φάσεις από τον σχεδιασμό του έργου μέχρι την Ενοποίηση και τον τελικό έλεγχο αποκαλούνται και φάσεις Ανάπτυξης του έργου.

3.4 ΔΟΜΙΚΟ ΜΟΝΤΕΛΟ II

Το Δομικό Μοντέλο II αποτελεί μία βελτιωμένη έκδοση του Δομικού Μοντέλου 81 που δημοσιεύθηκε το 1981 .Σε αυτό το μοντέλο ενσωματώθηκε στο είδη υπάρχον και προαναφερθέν Δομικό Μοντέλο I δυνατότητες και βελτιστοποιήσεις που παρέχονται από τα Λειτουργικά Σημεία (FP).
Οι πρωταρχικοί στόχοι του Δομικό Μοντέλο II είναι:

- Να καθιερώσει ένα μοντέλο εκτίμησης κόστους και χρονοπρογραμματισμού.
- Να αναπτύξει μία βάση δεδομένων εκτίμησης κόστους και ένα εργαλείο που θα υποστηρίζει δυνατότητες για συνεχή βελτίωση του μοντέλου.
- Να παρέχει ένα πλαίσιο ποσοτικής ανάλυσης καθώς κα ένα σύνολο εργαλείων και τεχνικών για την εκτίμηση του κόστους λογισμικού.

Ουσιαστικά προτείνει την εισαγωγή του Δομικό Μοντέλο II στις διάφορες φάσεις του κύκλου ζωής του λογισμικού.

Το Δομικό Μοντέλο II αποτελείται ουσιαστικά από τρία ξεχωριστά στάδια:

- Το πρώτο στάδιο περιλαμβάνει την εκτίμηση προσπάθειας σύνθεσης του πρωτότυπου ή της εφαρμογής που πρόκειται να αναπτυχθεί.

- Το δεύτερο στάδιο περιλαμβάνει την εκτίμηση του μοντέλου του Early Design σταδίου του έργου όταν λίγα είναι γνωστά για τους συντελεστές κόστους του έργου.

- Το τρίτο στάδιο περιλαμβάνει το μοντέλο του Post Architecture σταδίου του έργου. Στο στάδιο αυτό έχουν αναπτυχθεί τύποι οι οποίοι υπολογίζουν την προσπάθεια, τον χρονοπρογραμματισμό και το κόστος που απαιτείται για την ανάπτυξη ενός προϊόντος λογισμικού. Επίσης περιλαμβάνεται μία κατάτμηση της προσπάθειας και του χρονοπρογραμματισμού του στις διάφορες φάσεις του κύκλου ζωής του έργου και προσαρμόζεται κατάλληλα σε έργα λογισμικού που αναπτύσσονται με βάση το μοντέλο του «καταρράκτη» (waterfall model).

Στο Δομικό Μοντέλο II η προσπάθεια εκφράζεται σε άνθρωπο-μήνες (PM). Οι άνθρωπο-μήνες είναι το χρονικό διάστημα που ένα άτομο ξοδεύει εργαζόμενο στην ανάπτυξη ενός έργου λογισμικού για έναν μήνα. Αυτός ο αριθμός δεν περιλαμβάνει τις διακοπές και τις αργίες αλλά μετρά τις αργίες των σαββατοκύριακων. Ο αριθμός των άνθρωπο-μηνών είναι διαφορετικός από το χρόνο που θα πάρει το πρόγραμμα για να ολοκληρωθεί, αυτό καλείται πρόγραμμα ανάπτυξης.

Το Δομικό Μοντέλο II αποτελείται από 3 επιμέρους μοντέλα:

3.5 ΔΟΜΙΚΑ ΜΟΝΤΕΛΑ

❖ Μοντέλο Application Composition

Το μοντέλο Application Composition το οποίο αφορά την προσπάθεια που καταβάλλεται για την κατασκευή ενός προτύπου της εφαρμογής ,προκειμένου να αναδειχθούν και να αναλυθούν τα ποσοστά υψηλής επικινδυνότητας και ρίσκου. Θέματα όπως είναι το περιβάλλον ανάπτυξης ,θέματα απόδοσης και ωριμότητας τεχνολογιών. Το συγκεκριμένο μοντέλο χρησιμοποιεί αντικείμενα σημείων (object points) που αφορούν μετρήσεις των οθονών ,των αναφορών και των modules γλωσσών τρίτης γενιάς που αναπτύχθηκαν στην εφαρμογή, κάθε ένα σταθμισμένο με βάση μια κλίμακα (απλό ,μέσο ,δύσκολο) συντελεστών πολυπλοκότητας.
Η εκτίμηση των object points (σημεία αντικειμένων) είναι σχετικά μία νέα προσέγγιση για την εκτίμηση του μεγέθους της εφαρμογής .Στηρίζεται στην χρήση του rapid-composition integrated computer Aided Software Environment (ICASE) που περιέχει builders για διεπαφή χρήστη , εργαλεία ανάπτυξης λογισμικού καθώς και μεγάλα και σύνθετα components εφαρμογών. Σε συγκριτικές μελέτες που έχουν γίνει object points απαιτούν περίπου το 47% του μέσου χρόνου που απαιτείται για να παραχθεί μία αντίστοιχη FP εκτίμηση.

❖ Μοντέλο Early Design

Το μοντέλο Early Design το οποίο αφορά στην αναζήτηση των εναλλακτικών αρχιτεκτονικών συστημάτων καθώς και εννοιών λειτουργικότητας . Χρησιμοποιείται στα αρχικά στάδια του έργου όπου λίγα είναι γνωστά για το Μετρικές Λογισμικού μέγεθος του έργου που θ' αναπτυχθεί. Σε αυτό το στάδιο δεν είναι γνωστή αρκετή πληροφορία ώστε να υποστηριχθεί μία λεπτομερής εκτίμηση κόστους λογισμικού. Το συγκεκριμένο μοντέλο χρησιμοποιεί FPS και ένα μικρό αριθμό συντελεστών κόστους για τις εκτιμήσεις.

❖ Μοντέλο Post Architecture

Το Post Architecture αναφέρεται στην πραγματική ανάπτυξη και συντήρηση του προϊόντος λογισμικού. Το μοντέλο αυτό χρησιμοποιεί πηγαίες γραμμές κώδικα και Λειτουργικά σημεία (FPS) για την μέτρηση του μεγέθους της εφαρμογής καθώς και συντελεστές για την επαναχρησιμοποίηση κώδικα και την τμηματοποίηση του λογισμικού. Ένα σύνολο από 17 πολλαπλασιαστικούς συντελεστές κόστους.
Αυτό το μοντέλο χρησιμοποιεί Λειτουργικά Σημεία για μια χονδρική εκτίμηση του μεγέθους της εφαρμογής καθώς επίσης και την χρήση 5 συντελεστών κόστους. (πχ 2 συντελεστές κόστους για ικανότητα προσωπικού (personal capability) και εμπειρίας προσωπικού (personnel experience) σε σχέση με τους 6 που υπάρχουν στο Post Architecture. Τελικά, το συγκεκριμένο επίπεδο λεπτομέρειας συσχετίζεται με το γενικότερο επίπεδο διαθέσιμης πληροφορίας και το γενικότερο επίπεδο ακρίβειας της εκτίμησης που χρειάζεται σε αυτό το στάδιο.

ΚΕΦΑΛΑΙΟ 4
ΑΝΑΛΥΣΗ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ (FUNCTION POINTS ANALYSIS)

4.1 ΕΙΣΑΓΩΓΗ

Τα λογισμικά συστήματα συνεχίζουν να αναπτύσσονται σε μέγεθος και πολυπλοκότητα κάνοντας πολύ δύσκολη την κατανόηση τους. Οι συνεχείς βελτιώσεις στα εργαλεία κωδικοποίησης επιτρέπουν στους κατασκευαστές λογισμικού να παράγουν όλο και πιο πολλά προγράμματα προσπαθώντας να ικανοποιήσουν τις απαιτήσεις των χρηστών. Τα Λειτουργικά Σημεία (Function Points) αποτελούν μία μέθοδο για την κατανόηση και τον έλεγχο του μεγέθους ενός λογισμικού. Η μέθοδος που χρησιμοποιείται είναι το σπάσιμο του συστήματος σε μικρότερα κομμάτια ώστε να είναι ευκολότερη η ανάλυση και κατανόηση του. Πιο απλά τα Λειτουργικά Σημεία είναι μία παρουσίαση του μεγέθους μίας εφαρμογής. Το μέγεθος ενός σπιτιού μπορεί να μετρηθεί σε τετραγωνικά μέτρα , έτσι με τον ίδιο τρόπο μπορούμε να αποδώσουμε το μέγεθος μια εφαρμογής σε Λειτουργικά Σημεία.

4.2 Η ΑΝΑΠΤΥΞΗ ΤΩΝ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ

Η ανάπτυξή της τεχνικής αυτής έγινε στα μέσα της δεκαετίας του 1979 από τον Allan J. Albrecht σε μία προσπάθεια να ξεπεραστούν τα προβλήματα που εμφανίζονταν στην μέτρηση μεγέθους λογισμικών με την τεχνική μέτρησης γραμμών (Lines of Code) [14].

Τα Λειτουργικά Σημεία αντιμετωπίζουν το μετρούμενο σύστημα από την λειτουργική σκοπιά κάνοντάς τα έτσι ανεξάρτητα τεχνολογίας. Επομένως η μέθοδος που χρησιμοποιείται είναι ανεξάρτητη γλώσσας και τεχνικής ανάπτυξης. Η μόνη μεταβλητή που χρησιμοποιείται είναι το ποσό της προσπάθειας που χρειάζεται για να δοθεί ένα δεδομένο σύνολο λειτουργικών σημείων.

Τα Λειτουργικά Σημεία δηλαδή χρησιμοποιούνται για να καθορίσουν εάν μία γλώσσα ανάπτυξης ή ένα εργαλείο είναι πιο παραγωγικό σε σχέση με κάποιο αντίστοιχο.

4.3 ΣΥΝΤΕΛΕΣΤΗΣ ΠΡΟΣΑΡΜΟΣΜΕΝΩΝ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ (ADJUSTED FUNCTION POINT COUNT - AFP)

Στην αξιολόγηση Λειτουργικών Σημείων πρωταρχικός στόχος είναι ο προσδιορισμός του συντελεστή προσαρμογής λειτουργικών σημείων. Η διαδικασία αυτή γίνεται στα παρακάτω 4 βήματα:

- Αναγνώριση των συναλλαγών (transactional function types) και της συμβολής τους στον υπολογισμό του unadjusted function point count.
- Αναγνώριση των δεδομένων (data function types) για τον υπολογισμό του Μη Προσαρμοσμένου Συντελεστή Λειτουργικών Σημείων (unadjusted function point count).
- Προσδιορισμός της τιμής του Παράγοντα Προσαρμογής (Value Adjustment Factor -VAF).
- Υπολογισμός του Προσαρμοσμένου Συντελεστή Λειτουργικών Σημείων.

4.4 ΣΥΣΤΑΤΙΚΑ ΛΟΓΙΣΜΙΚΟΥ (COMPONENTS)

Υπάρχουν 5 κατηγορίες λειτουργίας χρηστών ή αλλιώς 5 βασικά συστατικά (components) του υπό αξιολόγηση λογισμικού. Αυτά είναι:

α) Είσοδοι (External Inputs)
β) Έξοδοι (External Outputs)
γ) Επερωτήσεις (External Inquiries)
δ) Εσωτερικά Λογικά Αρχεία (Internal Logical Files)
ε) Εξωτερικά Αρχεία Διεπαφής (External Interface Files)

Τα συστατικά στοιχεία κάθε λογισμικού (components) κατηγοριοποιούνται ανάλογα με το εάν είναι Τύποι Εγγραφών (record element types – RET's), Τύποι Δεδομένων (data element types – DET's) ή τύποι αρχείων που ενημερώνονται ή λαμβάνονται δεδομένα από αυτά –Αρχεία Αναφοράς (file types referenced - FTR's). Όταν τα βασικά συστατικά στοιχεία του υπό αξιολόγηση λογισμικού κατηγοριοποιηθούν τότε γίνεται μία ταξινόμηση ανάλογα με την πολυπλοκότητα τους. Έτσι μπορούν να χαρακτηριστούν ως α) χαμηλής, β)μέσης ή γ) υψηλής πολυπλοκότητας.

Όλα τα συστατικά λογισμικού αξιολογούνται με βάση τους Τύπους Δεδομένων (DET's). Ένα πεδίο Τύπων Δεδομένων είναι ένα μοναδικό και αναγνωρίσιμο στον χρήστη πεδίο με μη αναδρομικά συμπεριφορά. Οι DET πληροφορίες είναι πληροφορίες που είναι δυναμικές και όχι στατικές. Είναι ένα δυναμικό πεδίο που διαβάζεται από ένα αρχείο ή δημιουργούνται από άλλο πεδίο DET που περιέχονται σε ένα αρχείο αναφοράς. Εάν ένα DET έχει επαναληπτικό χαρακτήρα τότε υπολογίζεται μόνο η πρώτη του εμφάνιση .

Οι Τύποι Δεδομένων μπορεί να είναι είτε ποσοτικοί είτε ποιοτικοί. Ένα ποσοτικό στοιχείο δεδομένων είναι τα δεδομένα σε αριθμητική μορφή. Ποιοτική μορφή είναι τα δεδομένα που δεν είναι σε αριθμητική μορφή, αλλά εμφανίζονται ως κείμενο, φωτογραφία ή ήχος.

Χωρίζοντας λοιπόν τα στοιχεία ενός λογισμικού σε Αρχεία αναφοράς και Τύπους Εγγράφων και μετρώντας σε αυτά τους Τύπους Δεδομένων που υπάρχουν μπορούμε να μετρήσουμε τα Λειτουργικά Σημεία του συστήματος.

Αυτό γίνεται ως εξής: Στις εισόδους, εξόδους και στις επερωτήσεις μετρώνται τα Αρχεία αναφοράς και οι Τύποι Εγγραφών ενώ στα Εσωτερικά Λογικά Αρχεία και στα Εξωτερικά Αρχεία Διεπαφής οι Τύποι Εγγραφών και οι Τύποι Δεδομένων.

Έτσι μπορεί να σχηματιστεί ο παρακάτω πίνακας:

Είσοδοι	αρχεία αναφοράς	τύποι δεδομένων
Έξοδοι	αρχεία αναφοράς	τύποι δεδομένων
Επερωτήσεις	αρχεία αναφοράς	τύποι δεδομένων
Εσωτερικά Λογικά Αρχεία	τύποι εγγραφών	τύποι δεδομένων
Εξωτερικά Αρχεία Διεπαφής	τύποι εγγραφών	τύποι δεδομένων

4.4.1 Είσοδοι (External Inputs)

Αφορούν διαδικασίες στις οποίες τα δεδομένα εισέρχονται στο σύστημα από εξωτερικές πηγές. Τα δεδομένα βρίσκονται εκτός λογισμικού και η εισαγωγή τους μπορεί να γίνει είτε από οθόνες εισαγωγής δεδομένων είτε από άλλες εφαρμογές. Μπορούν να είναι πεδία εισαγωγής δεδομένων, μηνύματα λάθους, υπολογιζόμενες τιμές, ή απλά buttons.

Η αξιολόγηση των εισόδων γίνεται με βάση τον αριθμό των Τύποι Δεδομένων που εμφανίζονται και των συσχετιζόμενων αρχείων αναφοράς. Αυτά πολλαπλασιάζονται με τον ανάλογο συντελεστή σύμφωνα τον αριθμό τους δηλώνοντας έτσι την πολυπλοκότητα του συστήματος (χαμηλή ,μέση ή υψηλή) .Οπότε σχηματίζεται ο ακόλουθος πίνακας:

Αρχεία αναφοράς (FTR)	**Τύποι Δεδομένων (DET's)**		
	1-4	5-15	Περισσότερα από 15
Λιγότερα από 2	Χαμηλή (επί 3)	Χαμηλή (επί 3)	Μέση (επί 4)
Ίσα με 2	Χαμηλή (επί 3)	Μέση (επί 4)	Υψηλή (επί 5)
Περισσότερα από 2	Μέση (επί 4)	Υψηλή (επί 5)	Υψηλή (επί 5)

4.4.2 Έξοδοι (External Outputs)

Αφορούν διαδικασίες στις οποίες τα δεδομένα εξέρχονται από το σύστημα. Μπορούν να είναι πεδία αναφορών ή μηνύματα λάθους .Τα δεδομένα αυτά συνθέτουν αναφορές ή αρχεία εξόδου που αποστέλλονται σε άλλες εφαρμογές. Επιπλέον οι έξοδοι ενημερώνουν τα Εσωτερικά Λογικά Αρχεία.

Τα δεδομένα δημιουργούν αναφορές ή αρχεία εξόδου που αποστέλλονται σε άλλες εφαρμογές. Αυτά τα αρχεία και αναφορές περιέχονται σε ένα ή περισσότερα Εσωτερικά Λογικά Αρχεία ή Εισόδους.

Όπως όλα τα συστατικά στοιχεία, οι Έξοδοι πρέπει να αξιολογηθούν. Η βαθμολογία είναι με βάση τον αριθμό των Τύποι Δεδομένων και των Αρχείων Αναφοράς που σχετίζονται με τα Αρχεία Εξόδου.

Αρχεία αναφοράς (FTR)	**Τύποι Δεδομένων (DET's)**		
	1-5	6-19	Περισσότερα από 19
Λιγότερα από 2	Χαμηλή (επί 4)	Χαμηλή (επί 4)	Μέση (επί 5)
2 ή 3	Χαμηλή (επί 4)	Μέση (επί 5)	Υψηλή (επί 7)
Περισσότερα από 3	Μέση (επί 5)	Υψηλή (επί 7)	Υψηλή (επί 7)

4.4.3 Επερωτήσεις (External Inquiries)

Αφορούν διαδικασίες εισόδων και εξόδων οι οποίες έχουν ως αποτέλεσμα την λήψη δεδομένων από ένα ή περισσότερα λογικά αρχεία. Μπορεί να αναλυθεί σε πλευρά εισόδου όπου είναι πεδία που μπορούμε να κάνουμε αναζητήσεις κλικάροντας με το ποντίκι και σε πλευρά εξόδου όπου βρίσκουμε πεδία που μπορούν να εμφανιστούν σε μία οθόνη.

Οι διαδικασίες εισόδου δεν ενημερώνουν και δεν διατηρούν Αρχεία Αναφοράς (Internal Logical Files ή External Interface Files) και η Έξοδος δεν αποστέλλει δεδομένα.

Οι Επερωτήσεις βαθμολογούνται και αξιολογούνται ανάλογα του αριθμού των Αρχείων Αναφοράς.

Αρχεία αναφοράς (FTR)	**Τύποι Δεδομένων (DET's)**		
	1-5	6-19	Περισσότερα από 19
Λιγότερα από 2	Χαμηλή (επί 3)	Χαμηλή (επί 3)	Μέση (επί 4)
2 ή 3	Χαμηλή (επί 3)	Μέση (επί 4)	Υψηλή (επί 6)
Περισσότερα από 3	Μέση (επί 4)	Υψηλή (επί 6)	Υψηλή (επί 6)

4.4.4 Εσωτερικά Λογικά Αρχεία (Internal Logical Files)

Αποτελούν αρχεία όπως τα αντιλαμβάνεται ο χρήστης και όχι τις φυσικές τους αναπαραστάσεις. Τα αρχεία αυτά περιλαμβάνουν σύνολο λογικά συσχετιζόμενων δεδομένων τα οποία εμπεριέχονται στα όρια των εφαρμογών και ενημερώνονται από εξωτερικές εισόδους. Ένα Εσωτερικό Λογικό Αρχείο διατηρείται εσωτερικά της εφαρμογής έχει λογική δομή και αποθηκεύεται με την μορφή αρχείου. Δεν αποτελεί κανόνα αλλά συνήθως ένα αρχείο Εξόδου ή ένα αρχείο Επερώτησης θα πρέπει να έχει τουλάχιστον ένα Εσωτερικό Λογικό Αρχείο ως αρχείο αναφοράς.

Η αξιολόγηση των Εσωτερικών Λογικών Αρχείων γίνεται ως εξής:

Τύποι Εγγραφών (RET)	**Τύποι Δεδομένων (DET's)**		
	1-19	20-50	Περισσότερα από 51
1 RET	Χαμηλή (επί 7)	Χαμηλή (επί 7)	Μέση (επί 10)
2 έως 5 RET	Χαμηλή (επί 7)	Μέση (επί 10)	Υψηλή (επί 15)
Περισσότερα από 6	Μέση (επί 10)	Υψηλή (επί 15)	Υψηλή (επί 15)

4.4.5 Εξωτερικά Αρχεία Διεπαφής (External Interface Files)

Τα στοιχεία που περιλαμβάνουν ένα σύνολο λογικά συσχετιζόμενων δεδομένων που χρησιμοποιούνται ως αρχεία αναφοράς. Το λογισμικό χρησιμοποιεί αυτού του είδους τα αρχεία αλλά δεν ενημερώνεται από αυτά. Είναι αρχεία αυτά βρίσκονται εκτός των ορίων της εφαρμογής και συντηρούνται από άλλες εφαρμογές πχ. αρχεία διαμόρφωσης (configuration files), αρχεία αρχικοποίησης (initialization files).

Τα Εξωτερικά Αρχεία Διεπαφής είναι ένα Εσωτερικό Λογικό Αρχείο για μια άλλη εφαρμογή. Μία εφαρμογή μπορεί να μετρά ένα αρχείο είτε ως Εξωτερικό Αρχείο Διεπαφής είτε ως Εσωτερικό Λογικό Αρχείο. Ποτέ και τα δύο.
Ο πίνακας που δημιουργείται για την αξιολόγηση των Εξωτερικών Αρχείων Διεπαφής είναι ο ακόλουθος:

Τύποι Εγγραφών (RET)	**Τύποι Δεδομένων (DET's)**		
	1-19	20-50	Περισσότερα από 51
1 RET	Χαμηλή (επί 5)	Χαμηλή (επί 5)	Μέση (επί 7)
2 έως 5 RET	Χαμηλή (επί 5)	Μέση (επί 7)	Υψηλή (επί 10)
Περισσότερα από 6	Μέση (επί 7)	Υψηλή (επί 10)	Υψηλή (επί 10)

4.5 ΣΥΝΤΕΛΕΣΤΗΣ ΜΗ ΠΡΟΣΑΡΜΟΣΜΕΝΩΝ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ (Unadjusted Function Point-UFP).

Ο τελικός αριθμός των UFP δίνεται από τον πίνακα :

	Πολυπλοκότητα Συστατικών Στοιχείων			
Τύπος Συστατικών Στοιχείων	Χαμηλή	Μέση	Υψηλή	Σύνολο
Είσοδοι	Εις1 (x3)	Εις 2 (x4)	Εις 3 (x6)	Τελ.Εις=∑Εις
Έξοδοι	Εξ.1 (x4)	Εξ.2 (x5)	Εξ.3 (x7)	Τελ.Εξ=∑Εξ
Επερωτήσεις	Επ.1 (x3)	Επ.2 (x4)	Επ.3 (x6)	Τελ.Επ=∑Επ
Εσωτερικά Λογικά Αρχεία	Ε.Λ.Α1 (x7)	Ε.Λ.Α2 (x10)	Ε.Λ.Α3 (x15)	Τελ.Ε.Λ.Α=∑Ε.Λ.Α
Εξωτερικά Αρχεία Διεπαφής	Ε.Α.Δ.1 (x5)	Ε.Α.Δ.2 (x7)	Ε.Α.Δ.3 (x7)	Τελ.Ε.Α.Δ=∑Ε.Α.Δ
Τελ.Εις+Τελ.Εξ+Τελ.Επ+Τελ.Ε.Λ.Α+Τελ.Ε.Α.Δ ⟹ **Συντελεστής Μη Προσαρμοσμένων Λειτουργικών Σημείων**				

4.6 ΣΥΝΤΕΛΕΣΤΗΣ ΠΡΟΣΑΡΜΟΓΗΣ (Value Adjustment Factor-VAF)

Ο συντελεστής προσαρμογής είναι μία αξιολόγηση των γενικών χαρακτηριστικών του συστήματος. Βασίζεται σε 14 χαρακτηριστικά τα οποία μετρούν την γενικότερη λειτουργικότητα της εφαρμογής. Κάθε χαρακτηριστικό ακολουθείται από μία περιγραφή έξι βαθμίδων (από μηδέν έως πέντε) ανάλογα με τον βαθμό επίδρασης που έχει στο σύστημα.

Βαθμός Επίδρασης	Επίδραση στο σύστημα
0	Καμία επιρροή
1	Τυχαία επιρροή
2	Μέτρια επιρροή
3	Μέση επιρροή
4	Σημαντική επιρροή
5	Πολύ μεγάλη επιρροή

Γενικά χαρακτηριστικά του συστήματος. Βαθμοί επιρροής στο σύστημα. (Degrees of Influence-DI) :

DI1) Επικοινωνία δεδομένων (Data Communication): Πόσες επικοινωνιακές συνδέσεις υπάρχουν για να βοηθήσουν την μεταφορά ή την ανταλλαγή πληροφοριών με την εφαρμογή ή το σύστημα;
DI2) Κατανεμημένη επεξεργασία δεδομένων (Distributed data processing): Πώς γίνεται ο χειρισμός της κατανομής των δεδομένων και της εκτέλεσης των διαφόρων λειτουργιών;
DI3) Απόδοση (Performance): Οι χρόνοι απόκρισης ήταν οι απαιτούμενοι από τον χρήστη;
DI4) Υπερφορτωμένο Σύστημα (Heavily used configuration): Πόσο πολυχρησιμοποιημένη είναι η τρέχουσα hardware πλατφόρμα όπου θα εγκατασταθεί και θα λειτουργεί η εφαρμογή;
DI5) Ρυθμός Δοσοληψιών (Transaction Rate): Πόσο συχνά εκτελούνται δοσοληψίες (ημερήσια, εβδομαδιαία, μμηνιαία κλπ);
DI6) On-Line καταχωρήσεις δεδομένων (On-Line data entry): Τι ποσοστό της πληροφορίας εισάγεται On-Line;
DI7) Αποδοτικότητα Τελικού Χρήστη (End-user efficiency): Το σύστημα σχεδιάστηκε με σκοπό να αυξήσει την αποδοτικότητα του τελικού χρήστη;
DI8) On-line ενημέρωση (On-line update): Πόσα Εσωτερικά Αρχεία ενημερώνονται από On-line καταχωρήσεις;
DI9) Πολύπλοκη Επεξεργασία (Complex processing): Η εφαρμογή έχει αυξημένη λογική και μαθηματική επεξεργασία;
DI10) Επαναχρησιμοποίηση (Reusability): Η εφαρμογή αναπτύχθηκε για έναν πελάτη ή θα χρησιμοποιηθεί και για άλλες εγκαταστάσειςq
DI11) Ευκολία εγκατάστασης (Installation ease): Πόσο δύσκολη είναι η μετάβαση και η εγκατάσταση;
DI12) Ευκολία Χρήσης (Operational easy): Πόσο αποτελεσματικές και αυτοματοποιημένες είναι οι διαδικασίες εκκίνησης, λήψης εφεδρικών αντιγράφων (backup) και αποκατάστασης από αποτυχίες (recovery);
DI13) Πολλαπλοί χώροι εγκατάστασης (Multiple sites): Η εφαρμογή σχεδιάστηκε, αναπτύχθηκε και υποστηρίζει πολλαπλές εγκαταστάσεις σε οργανισμούς;
DI14) Διευκόλυνση Αλλαγών (Facilitate change): Η εφαρμογή σχεδιάστηκε, αναπτύχθηκε και υποστηρίχθηκε με τρόπο που προήγαγε την αλλαγή των διαδικασιών στον οργανισμό;

Στην συνέχεια για να υπολογιστεί ο Συντελεστής Προσαρμογής (VAF) βρίσκουμε το άθροισμα όλων των Degrees of Influence (DI), δηλαδή το άθροισμα από τα 14 γενικά χαρακτηριστικά το κάθε ένα από τα οποία παίρνει τιμές από 0 έως 5 ανάλογα με την επίδραση που έχει στο σύστημα.
Έτσι έχουμε: $VAF=0.65+0.01* \sum DI\ i$

Υπολογισμός λειτουργικών σημείων

Ο τελικός υπολογισμός των Λ.Σ του συστήματος δίνεται από το γινόμενο των Μη Προσαρμοσμένων Λειτουργικών σημείων (UAF) με το Σύνολο Προσαρμοσμένων Λειτουργικών Σημείων (VAF) δηλαδή από την εξίσωση :
FP = UAF * VAF

Έχοντας βρει τα ΛΣ του συστήματος μπορούμε πλέον να εκτιμήσουμε την ανάπτυξη του έργου με βάση την παραγωγικότητα των προγραμματιστών που έχουμε στην διάθεσή μας ή ακόμα μπορούμε να κάνουμε απευθείας μετατροπή των ΛΣ σε γραμμές κώδικα (LoC).

4.7 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ

Ακολουθεί η ανάλυση ενός παραδείγματος για την βαθύτερη κατανόηση των Λειτουργικών Σημείων [15,26].

Πρόβλημα

Μία εταιρία σχεδιάζει να φτιάξει μία απλή εφαρμογή για να διαχειρίζεται πληροφορίες σχετικά με εταιρίες που ενδιαφέρονται για μαθήματα σχετικά με τα Λειτουργικά Σημεία.

Μία λογική μορφή της φόρμας συμπλήρωσης των στοιχείων της εταιρία θα μπορούσε να είναι:

Company:	**Name of Contact:**
Job Title:	**Date of Initial Contact:**
Street Address:	**City:**
State:	**Zip Code:**
Phone Number:	**Fax Number:**

Αυτά τα δεδομένα θα πρέπει να συμπληρώνονται όταν δηλώνεται ένα ενδιαφέρον για την παρακολούθηση των μαθημάτων. Οι εργαζόμενοι θα πρέπει να έχουν την δυνατότητα να διορθώνον να αλλάζουν και να σβήνουν οποιαδήποτε πληροφορία μέσω ενός on-line συστήματος χρησιμοποιώντας τις εντολές create, update και delete.

Επιπλέον πεδία περιλαμβάνονται στο Company contact data αλλά ενημερώνονται με ξεχωριστές διεπαφές όπως τα Data packet sent, Data of Phone contact και Notes.

Επιπλέον ένα menu οδηγός απαιτείται για την πλοήγηση στο σύστημα. Αυτό περιλαμβάνει:

Create company contact:	**Retrieve company contact:**
Update company contact:	**Delete company contact:**
Packet sent:	**Phone contact completed:**

Σε αυτή την μελέτη περίπτωσης το ζητούμενο είναι να γίνει προσδιορισμός των λειτουργιών και της πολυπλοκότητας χρησιμοποιώντας των παρακάτω πίνακα:

Function Point Analysis				
Description	**Type**	**DETs**	**RETs/FTRs**	**Complexity**

Στο ίδιο υπό μελέτη σύστημα χρησιμοποιώντας τους βαθμούς επιρροής τους συστήματος (general systems characteristics) που δίνονται στον παρακάτω πίνακα ζητείται να υπολογιστεί ο adjusted function point count.

Function Point Calculation Worksheet				
IFPUG's Unadjusted Function Point Table				
Components	**Low**	**Average**	**High**	**Total**
External inputs	X3	X4	X6	
External outputs	X4	X5	X6	
External Inquiries	X3	X4	X6	
Internal logical files	X7	X10	X15	
External Interface files	X5	X7	X10	
Total unadjusted Function Point (FPA) =				
General System Characteristics				
Characteristics	**Degree of influence**	**Characteristics**	**Degree of influence**	
1.Data communications	4	**8.Online update**	3	
2.Distributed data processing	0	**9.Complex processing**	1	
3.Performance	0	**10.Reusabiulity**	3	
4.Heavily use configuration	0	**11.Instalation ease**	1	
5.Transaction rate	0	**12.Operation ease**	3	
6.Online data entry	5	**12.Multiple sites**	1	
7.End user efficiency	3	**14.Facilitate change**	2	
Total degree of influence (TDI) =				

VAF Value adjustment factor = (TDI X 0.01) +0. 65 =
FP Adjusted Function point count = UFP X VAF =

Απάντηση στην μελέτη περίπτωσης

Παρακάτω δίνονται οι υπολογισμοί [15][15] και οι πίνακες συμπληρωμένοι για τον υπολογισμό των συνολικών Λειτουργικών Σημείων του παραδείγματος.

Function Point Analysis				
Description	**Type**	**DETs**	**RETs/FTRs**	**Complexity**
Company contact data	ILF	13	2	L
Error file	EIF	4	1	L
Create company contact	EI	12	2	A
Retrieve company contact	EQ	15	2	A
Update company contact	EI	12	2	A
Delete company contact	EI	4	2	L
Packet sent	EI	5	2	A
Phone contact	EI	6	2	A

Function Point Calculation Worksheet				
IFPUG's Unadjusted Function Point Table				
Components	**Low**	**Average**	**High**	**Total**
External inputs	X3	X4	X6	**19**
External outputs	X4	X5	X6	**0**
External Inquiries	X3	X4	X6	**4**
Internal logical files	X7	X10	X15	**7**

External Interface files	X5	X7	X10	**5**
Total unadjusted Function Point (FPA) = 35				
General System Characteristics				
Characteristics	**Degree of influence**	**Characteristics**	**Degree of influence**	
1.Data communications	4	**8.Online update**	3	
2.Distributed data processing	0	**9.Complex processing**	1	
3.Performance	0	**10.Reusabiulity**	3	
4.Heavily use configuration	0	**11.Instalation ease**	1	
5.Transaction rate	0	**12.Operation ease**	3	
6.Online data entry	5	**12.Multiple sites**	1	
7.End user efficiency	3	**14.Facilitate change**	2	
Total degree of influence (TDI) = 26				
VAF Value adjustment factor = (TDI X 0.01) + 0. 65 = 0.91				
FP Adjusted Function point count = UFP X VAF = 31.85				

Άρα το μέγεθος του συγκεκριμένου συστήματος αξιολογείται σε 31.85 Λειτουργικά Σημεία .

4.8 ΛΕΙΤΟΥΡΓΙΚΑ ΣΗΜΕΙΑ ΤΡΙΩΝ ΔΙΑΣΤΑΣΕΩΝ - 3-D FUNCTION POINTS

Ανάμεσα στο 1989 και το 1992 η Boeing Company αποφάσισε να χρησιμοποιήσει την μέθοδο των λειτουργικών σημείων για την μέτρηση της παραγωγικότητας .Το 1992 έγιναν δημοσιοποιήσεις κατά την διάρκεια του συνεδρίου Pacific Northwest Software Quality όπου η νέα μέθοδος ονομάστηκε Λειτουργικά Σημεία Τριών Διαστάσεων (3-D Function Points).

Η νέα τεχνική σχεδιάστηκε για να αντιμετωπίσει δύο κλασικά προβλήματα που συνδέονται με την προσέγγιση Albrecht [16]. Αυτή η προσέγγιση αναπτύχθηκε από τον Allan Albrecht για να βοηθήσει στην εύρεση του

μεγέθους μηχανογραφημένων πληροφοριακών συστημάτων αλλά παρουσίαζε αρκετές αδυναμίες με τον συγγραφέα να προτείνει μια μερική λύση. Στα προβλήματα που έδινε λύσει η 3-D μέθοδος είναι η μεγάλη δυσκολία εφαρμογής που υπήρχε στην μέθοδο του Albrecht αλλά και τα προβλήματα που παρουσιάζονταν για την εκτίμηση συστημάτων πραγματικού χρόνου [17].

Οι τρεις διαστάσεις στα 3-D Λειτουργικά Σημεία είναι τα σημεία της λειτουργίας των δεδομένων και του έλεγχου . Η διάσταση των δεδομένων είναι παρόμοια με τα function points στην προσέγγιση Albrecht ενώ η διάσταση της λειτουργίας προσθέτει μετασχηματισμούς παρόμοιους με τους αλγορίθμους . Τέλος, η διάσταση ελέγχου περιλαμβάνει και απαριθμεί τις μεταβάσεις στις αλλαγές καταστάσεις τις εφαρμογής [18].

Είναι συζητήσιμο το κατά πόσον η τεχνική των 3-D Λειτουργικών Σημείων παρέχει περισσότερες ευκολίες κατά την καταμέτρηση ενώ ταυτόχρονα δεν είναι ξεκάθαρο εάν προσφέρει μεγαλύτερη ακρίβεια σε συστηματοποιημένες έρευνες. Γεγονός είναι ότι η τεχνική των 3D Λειτουργικών Σημείων δεν χρησιμοποιείται ευρέως αυτή την στιγμή.

4.9 Η IFPUG ΚΑΙ Η ΧΡΗΣΙΜΟΤΗΤΑ ΤΗΣ ΑΝΑΛΥΣΗΣ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ

Η μετρική των Λειτουργικών Σημείων αναπτύχθηκε από την IBM και έγινε δημόσια γνωστή τον Οκτώβριο του 1978. Το 1984 δημιουργήθηκε ο μη κερδοσκοπικός οργανισμός IFPUG (International Function Point Users Group) και ανέλαβε την ευθύνη για την αποσαφήνιση λεπτομερών αλλά και την δημιουργία κανόνων. Το 2008 η IFPUG αριθμεί πάνω από 3000 μέλη [20].

Στόχος της IFPUG είναι να αποτελέσει ένα αναγνωρισμένο ηγέτη στην προώθηση και την αποτελεσματική διαχείρισης της ανάπτυξης λογισμικού και να συντηρεί δραστηριότητες που χρησιμοποιούν Λειτουργικά Σημεία ή άλλες τεχνικές αξιολόγησης λογισμικού.

Η IFPUG διευκολύνει την ανταλλαγή γνώσεων και ιδεών για να βελτιώσει τις ήδη υπάρχουσες τεχνικές αξιολόγησης και επιδιώκει να παρέχει το κατάλληλο περιβάλλον για την προσωπική αλλά και επαγγελματική ανάπτυξη των μελών της. Κάθε χρόνο πραγματοποιούνται δύο συνέδρια , ένα την άνοιξη και ένα το φθινόπωρο. Το φθινοπωρινό συνέδριο περιλαμβάνει εκπαιδευτικά προγράμματα και συνελεύσεις ενώ στο συνέδριο που γίνεται την άνοιξη περιλαμβάνεται και μία διήμερη αγορά όπου λαμβάνουν χώρα και παρουσιάσεις ιδιωτικών προγραμμάτων .

Οι επιτροπές εργασίας είναι ο πυρήνας της IFPUG. Οι σημαντικότερες είναι η Counting Practices Committee και η Certification Committee. Η Counting

Practices Committee είναι υπεύθυνη για την διαχείριση των εργασιών που είναι σε εξέλιξη ενώ οι εργασίες της Certification Committee περιλαμβάνουν αποκλειστικά θέματα πιστοποίησης. Άλλες γνωστές κατευθυντήριες επιτροπές είναι οι New Environments, η IT Performance και η Management Reporting.

Διεθνώς Η επέκταση της IFPUG δεν περιορίζεται μόνο στις Η.Π.Α αλλά έχει επεκταθεί και σε αρκετές χώρες διεθνώς όπου απαριθμεί πολλά σημαντικά μέλη όπως :

- Australian Software Metrics Association (ASMA)
- Austria Function Point Users Group (FPUGA)
- Brazilian Function Point Users Group (BFPUG)
- Denmark Function Point Users Group (DANMET)
- Software-Metrik und Aufwandschatzung (DASMA)
- Italian User Group on Function points (GUFPI)
- Japan Function Point User Group (JFPUG)
- Netherlands Software Metrics Users Association (NESMA)
- South Africa Software Metrics Association (SAMA)
- UK Software Metrics Association (UKSMA)

Η μέθοδος της Ανάλυσης Λειτουργικών Σημείων είναι μία μέθοδος αξιολόγηση που δεν είχε κατανοηθεί πλήρως και δεν είχε χρησιμοποιηθεί ορθά στο παρελθόν. Η σωστή της χρήση μπορεί να παρέχει πολύ σημαντικές πληροφορίες που μπορούν να βελτιώσουν την αποτελεσματικότητα στην ανάπτυξη και στην σχεδίαση ενός λογισμικού [21,28]..

Εκτός από την συνηθισμένη μορφή των Λειτουργικών σημείων της IFPUG υπάρχουν πάνω από 24 παραλλαγές που έχουν αναπτυχθεί όπως οι COSMIC function points, οι Finish function points, οι Engineering function points, οι Netherlands function points και άλλες. Μεταξύ τους έχουν πολύ μικρές διαφορές ενώ το 85% των έργων χρησιμοποιούν το μοντέλο της IFPUG.

Η μετρική των Λειτουργικών Σημείων μπορεί να έχει πολλές χρήσεις . Οι κυριότερες αυτών μπορεί να είναι :

- Μετρική του ρυθμού βελτίωσης της παραγωγικότητας ,των χρονοδιαγραμμάτων αλλά και του κόστους.
- Σημείο αναφοράς για την παραγωγικότητα λογισμικού.
- Σημείο αναφοράς για την ποιότητα λογισμικού.
- Εκτίμηση νέων εφαρμογών πριν από την ανάπτυξη τους.

Χρησιμοποιώντας την τεχνική των Λειτουργικών Σημείων χρειάζεται να υπάρχει μια ολοκληρωμένη εικόνα των απαιτήσεων του μετρούμενου

λογισμικού. Η αξιολόγηση του λογισμικού πρέπει να γίνεται σε ένα διάστημα μεταξύ του ενός και των έξι μηνών μετά την έναρξη του . Πρόσθετες πληροφορίες όπως οι προδιαγραφές λειτουργίας προσθέτουν ακρίβεια στις μετρήσεις.

4.10 ΚΟΣΤΟΣ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ ΚΑΙ ΟΙ ΠΕΡΙΟΡΙΣΜΟΙ ΣΤΗΝ ΧΡΗΣΗ ΤΟΥΣ

Κατά την μέτρηση με Λειτουργικά Σημεία συνήθως παρατηρείται ένα πλήθος 400 με 600 μετρούμενων σημείων ανά ημέρα ενώ ένα σύνηθες κόστος ανά λειτουργικό σημείο ανέρχεται στα 4 με 6 δολάρια.

Πρόκειται για ένα αρκετά σημαντικό κόστος το οποίο επιβραδύνει την ανάπτυξη των Λ.Σ για εφαρμογές με περισσότερα των 15,000 λειτουργικών σημείων.

Για παράδειγμα σε μεγάλες εφαρμογές (π. χ ERP) το πλήθος των λειτουργικών σημείων μπορεί να φτάσει τις 300,000 και ο χρόνος ολοκλήρωσης του έργου τις 750 ημέρες. Σε αυτήν την περίπτωση το κόστος ίσως ξεπεράσει το 1,800,000 $. Κόστος μεγάλο τόσο σε χρόνο όσο και σε πόρους όπου καμία εταιρία δεν θα ήταν διατιθεμένη να καταβάλει.

Ο συνδυασμός της αναποτελεσματικότητας σε μικρά έργα (κάτω των 15 Λ.Σ.) με το μεγάλο κόστος ανά λειτουργικό σημείο έχει ως αποτέλεσμα μόνο το 10% των λογισμικών να αξιολογούνται με Λ.Σ

Συνήθως οι εταιρίες αφιερώνουν το 10% του φόρτου εργασίας τους σε τέτοια μικρά έργα. Αυτό γίνεται γιατί η διόρθωση των λαθών αλλά και οι μικρές αλλαγές σε ένα λογισμικό δεν ξεπερνούν τα 15 Λειτουργικά Σημεία.

Σε μεγάλες εταιρίες όπως η IBM, Η Microsoft και η EDS είναι σύνηθες να γίνονται τέτοιες μικρές αλλαγές σε λογισμικά. Αυτές μπορεί να ξεπεράσουν τις 30,000 το χρόνο. Ενώ αυτές οι αλλαγές σε μέγεθος μπορούν να ξεκινούν από το 1/50 ενός Λ.Σ. και να φτάνουν τα 15 Λ.Σ. το σύνολό τους περίπου σε 100,000 Λ.Σ.. τον χρόνο.

Για μεγάλες εφαρμογές , άνω των 15,000 Λειτουργικών Σημείων δαπανάται το 40% του φόρτου εργασίας των εταιριών. Παρόλα αυτά οι μεγάλες εφαρμογές είναι λίγες σε αριθμό ενώ χωρίζονται σε ομάδες ανάπτυξης μεταξύ των 500 και 5000 ατόμων. Αυτές οι εφαρμογές όμως παρουσιάζουν μεγάλο ρίσκο και κινδυνεύουν να αποτύχουν ή να παρουσιάσουν χρονικές καθυστερήσεις και υπερβάσεις του budget.

Τα μεγάλα έργα είναι οι καλύτεροι υποψήφιοι για να αξιολογηθούν και να αναλυθούν με την μέθοδο των Λειτουργικών Σημείων. Εκεί υπάρχουν οι μεγαλύτερες πιθανότητες να εμφανιστούν υπερβάσεις του προϋπολογισμού και καθυστερήσεις οπότε η έγκαιρη χρήση των Λ.Σ. θα βοηθούσε στην αποφυγή τους. Δυστυχώς το υψηλό κόστος ανά λειτουργικό σημείο και η υψηλή απαίτηση σε ανθρώπινο δυναμικό μειώνει την χρήση τους ακριβώς στις εφαρμογές εκείνες όπου υπήρχε η μεγαλύτερη ανάγκη [29].

Οι μεσαίου μεγέθους εφαρμογές ,μεγέθους μεταξύ 15 και 15,000 Λ.Σ. αποτελούν το 35% του φόρτου εργασίας των εταιριών ανάπτυξης λογισμικού. Κατά μέσο όρο υπολογίζεται ότι χρησιμοποιούνται 1,500 Λ.Σ. ανά εφαρμογή οι οποίες λόγο του υψηλού κόστους είναι και οι πιο σημαντικές . Εάν μία εταιρία παράγει 100 λογισμικά κάθε χρόνο μεγέθους από 100 έως 15,000 λειτουργικών σημείων μόνο το 25% των εφαρμογών αυτόν θα αξιολογηθεί με Function Points .Αυτά θα είναι έργα με ιδιαίτερη βαρύτητα ή με ερευνητική σημασία.

Επιπλέον όλες οι μεγάλες εταιρίες χρησιμοποιούν εφαρμογές COTS (commercial-off-the-shelf) όπως Microsoft Vista, SAP, Oracle, Symantec anti-virus και χιλιάδες άλλα. Έρευνες δείχνουν ότι οι εταιρίες χρησιμοποιούν περισσότερο COTS εφαρμογές παρά in-house.

Έτσι το τοπίο των εφαρμογών γεμίζει με μικρές εφαρμογές οι οποίες δεν αξιολογούνται με function points , με πολύ μεγάλες εφαρμογές όπου η αξιολόγηση τους είναι πολύ ακριβή , με μεσαίου τύπου εφαρμογές όπου δεν επιλέχθηκαν να αξιολογηθούν αλλά και με τις εφαρμογές τύπου COTS για τις οποίες δεν υπάρχουν αρκετές πληροφορίες για να αναλυθούν . Από τις εφαρμογές που εναπομείναν μόνο το 10% ανήκει σε εταιρίες οι οποίες έχουν την τεχνογνωσία της Ανάλυσης Λειτουργικών Σημείων . Με δεδομένη την ισχύ και την χρησιμότητα των Λ.Σ. σε οικονομικές μελέτες και σε αξιολογήσεις ποιότητας το ποσοστό των λογισμικών που τελικά θα αξιολογηθεί είναι εξαιρετικά μικρό.

Συνοψίζοντας βλέπουμε τους λόγους για την περιορισμένη χρήση της Ανάλυσης Λειτουργικών Σημείων:

Έργα με λιγότερα από 15 Λ.Σ.. δεν αξιολογούνται.
Έργα με περισσότερους από 15.000 Λ.Σ. κρίνονται ως πολύ ακριβά για αξιολόγηση.
Έργα μεταξύ 15 και 15.000 Λ.Σ. αξιολογείται μόνο το 25%.
Λογισμικά τύπου COTS δεν αξιολογούνται.

Ας υποθέσουμε ότι μία εταιρία διαχειρίζεται λογισμικά μεγέθους 10,000,000 function points . Από αυτά τα 5,000,000 αφορούν 3,000 in-house εφαρμογές

και τα άλλα 5,000,000 αναφέρονται σε 2,500 λογισμικά τύπου COTS. Η εταιρία αυτή είναι πιθανό να χρησιμοποιήσει Α.Λ.Σ σε 75 από αυτά τα λογισμικά με μία αντιστοιχία περίπου 100,000 Λ.Σ. Οι 100,000 όμως είναι ένα πολύ μικρό ποσοστό σε σύγκριση με το αρχικό μέγεθος των 10,000,000. Το πιθανό κόστος για αυτές τις 75 εφαρμογές θα ανέρχεται στα 600,000$. Το κόστος είναι αρκετά υψηλό και εξηγεί την περιορισμένη χρήση της μετρικής. Για να γίνει ανάλυση σε όλες τις in-house εφαρμογές (μεγέθους 5,000,000 function points) θα χρειαστεί ένα πόσο γύρω στα 30,000,000$. Ένα εξαιρετικά υψηλό κόστος όπου καμία εταιρία δεν θα ήθελε να δαπανήσει. Έτσι είναι εμφανές ότι η μετρική των Λειτουργικών Σημείων παρουσιάζει περιορισμούς στην χρήση της καθώς μπορεί να γίνει εξαιρετικά δαπανηρή και χρονοβόρα εάν θελήσουμε να χρησιμοποιηθεί στο 100% το πλήθους των εφαρμογών που αναπτύσσει μία μεγάλη εταιρία. Για αυτούς του λόγους η χρήση των Λ.Σ περιορίζεται παρά τα εξαιρετικά αποτελέσματα , την ακριβής ανάλυση αλλά και τα πλεονεκτήματα που παρουσιάζει σε οικονομικές μελέτες .

4.11 ΕΠΕΚΤΑΣΗ ΚΑΙ ΤΡΟΠΟΙ ΠΡΟΩΘΗΣΗΣ ΤΗΣ ΜΕΤΡΙΚΗΣ

Η Ανάλυση Λειτουργικών Σημείων αναμφίβολα είναι η πιο αποτελεσματική μετρική για εφαρμογές λογισμικού που έχει αναπτυχθεί έως τώρα. Δεν υπάρχουν άλλες μετρικές που να αποδίδουν τόσο καλά σε οικονομικές μελέτες , στην ανάλυση ποιότητας αλλά και στην εκτίμηση κόστους. Αλλά για νε βελτιωθεί περισσότερο πρέπει να γίνουν τροποποιήσεις σε μία σειρά από σημαντικούς παράγοντες όπως:

1. Το σημείο εκκίνησης της ανάλυσης θα πρέπει να ξεκινά έξι μήνες νωρίτερα.
2. Το κόστος ανά function point χρειάζεται να πέσει κάτω από το ένα ¢.
3. Η ταχύτητα μέτρησης των Λ.Σ. θα πρέπει να ξεπεράσει τα 10,000 την ημέρα.
4. Δεν θα πρέπει να υπάρχει κατώτερο όριο στην αξιολόγηση των λογισμικών.
5. Θα πρέπει να υπάρχει δυνατότητα αξιολόγησης των COTS εφαρμογών.
6. Έργα που ακυρώθηκαν θα πρέπει να αξιολογούνται.
7. Αλλαγές στις απαιτήσεις θα πρέπει να υπολογίζονται σε πραγματικό χρόνο.
8. Χαρακτηριστικά που διαγράφονται θα πρέπει να υπολογίζονται σε πραγματικό χρόνο.
9. Κληροδοτούμενες εφαρμογές θα πρέπει να προσμετρούνται.
10. Ενημερώσεις για κληροδοτούμενες εφαρμογές πρέπει να υπολογίζονται.

11. Η ανάλυση function point θα πρέπει σαν υπηρεσία ,χωρίς on-site επισκέψεις.
12. Η ανάλυση function point θα πρέπει να υπολογίζει τις αλλαγές στις απαιτήσεις.
13. Χρειάζεται αξιολόγηση στις απαιτήσεις που προκύπτουν από κάθε νέα έκδοση του λογισμικού.

Ήδη όμως , από το 2008, πολλές από αυτές τις τροποποιήσεις βρίσκονται σε εφαρμογή και χρησιμοποιούνται ενώ κάποιες άλλες είναι σε πειραματικό στάδιο και δεν έχουν ακόμα εμπορικά διαθέσιμες.

Η Relative Technologies ανακοίνωσε ένα εργαλείο αξιολόγησης εφαρμογών με την ονομασία “Function Point Analyzer”. Αυτό το λογισμικό λειτουργεί αναλύοντας τον πηγαίο κώδικα και δημιουργώντας κανόνες οι οποίοι μπορούν να χρησιμοποιηθούν με την μετρική Λειτουργικών Σημείων. Η μέτρηση γίνεται με ακρίβεια αλλά και μεγάλη ταχύτητα (μέσα σε λίγα λεπτά και όχι σε μέρες ή εβδομάδες). Έτσι το εργαλείο αυτό μπορεί να παρέχει μια υψηλής ταχύτητας και χαμηλού κόστους προσέγγιση για την αξιολόγηση λογισμικού με FPA.

Μία άλλη εταιρία στον χώρο της ανάπτυξης λογισμικού είναι η Software Productivity Research LLC η οποία δημιούργησε τρία λογισμικά τα οποία όμως , δεν κατάφεραν να καθιερωθούν. Αυτά ήταν το SPQR/20 ,το Checkpoint και το KnowledgePlan.

Τέλος ,η SPR, σε συνεργασία με την εταιρία Bechman ,ανέπτυξε ένα ακόμα λογισμικό για αξιολόγηση αλλά το κλείσιμο της τελευταίας απέτρεψε την ευρεία διάδοση του.

Έτσι το 2008, περισσότερα από 15,000 λογισμικά αξιολογήθηκαν με Α.Λ.Σ στα περισσότερα εκ των οποίων χρησιμοποιήθηκε η συνηθισμένη μέθοδος της IFPUG. Πολλά από αυτά τα λογισμικά είναι εμπορικά διαθέσιμα από την International Software Benchmark Standards Group(ISBSG). Ακόμα πολλά από αυτά μπορούν να χρησιμοποιηθούν ως συμβουλευτικό λογισμικό όπως τα SPR, Gather Group ,SDavid Consulting Group , QPMG πλήθος άλλων.

4.12 Η ΑΝΑΠΤΥΞΗ ΤΩΝ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ ΣΤΙΣ Η.Π.Α

Από το 1978, όταν η IBM κυκλοφόρησε την πρώτη έκδοση του Λ.Σ ήμερα υπάρχει συνεχείς βελτίωση και έρευνα της μετρικής. Στις Ηνωμένες Πολιτείες υπολογίζεται ότι υπάρχουν περίπου 1000 αναλυτές Λειτουργικών Σημείων εκ των οποίων οι 600 περίπου εργάζονται σε μεγάλες επιχειρήσεις ή κυβερνητικούς οργανισμούς ενώ οι υπόλοιποι 400 έχουν συμβουλευτικό χαρακτήρα. Μόνο ένα μικρό ποσοστό από αυτούς έχει πιστοποίηση από την IFPUG ή άλλες ομάδες πιστοποίησης όπως η COSMIC.

Στις Η.Π.Α. υπολογίζεται ότι κάθε αναλυτής αξιολογεί περίπου 25 Λ.Σ.. και συνολικά ολοκληρώνονται 25,000 έργα ανά έτος . Ωστόσο η πλειοψηφία των έργων προορίζεται για εσωτερική χρήση και μόνο ένα μικρό ποσοστό της τάξης των 200 έργα ανά έτος προστίθεται σε δημόσιες βάσεις δεδομένων όπως η ISBSG.

Υποθέτοντας μία μέση τιμή γύρω στα 1500 Λ.Σ.. ανά έργο κάθε χρόνο στις Η.Π.Α αξιολογούνται περίπου 3,750,000 Λ.Σ.. Βάζοντας ένα τυπικό κόστος 6$ ανά λειτουργικό σημείο υπολογίζεται ότι ετησίως δαπανούνται 22,500,000$ για την αξιολόγηση λογισμικού με την μετρική των Λειτουργικών Σημείων.

Τα Λειτουργικά Σημεία χρησιμοποιούνται ως σημείο αναφοράς σε συγκρίσεις και μελέτες λογισμικών. Κάθε χρόνο στις Η.Π.Α πραγματοποιούνται 500 από αυτές τις μελέτες με κόστος περίπου 50,000$ η κάθε μία ενώ τα έσοδα που προκύπτουν από αυτές τις αναφορές φτάνουν τα 25,000,000$. Τα περισσότερα από αυτά τα δεδομένα είναι εμπιστευτικά αλλά υπάρχει και ένα ποσοστό που υποβάλλεται στην International Benchmark Standards Group (ISBSG) . Η ISBSG έως το 2008 διέθετε περίπου 5,000 έργα από όλο τον κόσμο ενώ κάθε χρόνο ο αριθμός αυτός μεγαλώνει κατά 500.

Τα Λειτουργικά Σημεία χρησιμοποιούνται και για εμπορικούς σκοπούς. Πολλά εμπορικά λογισμικά χρησιμοποιούν την μετρική για να αξιολογήσουν το μέγεθος ενός λογισμικού. Επίσης χρησιμοποιούνται και σε δικαστικές αποφάσεις όπως οι παραβιάσεις συμβάσεων που αφορούν το κόστος και την αξία εφαρμογών.

4.13 Η ΜΕΤΡΙΚΗ ΤΩΝ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΗΜΕΙΩΝ ΣΕ ΕΥΡΩΠΗ-ΑΣΙΑ

Το ευρωπαϊκό επιχειρηματικό μοντέλο είναι παρόμοιο με αυτό των Η.Π.Α. σε μέγεθος με έσοδα από συγκριτικές αναλύσεις και μελέτες περίπου 50,000,000$ ανά έτος. Ωστόσο , η Ευρωπαϊκή αγορά είναι κατακερματισμένη και δύσκολη να χαρακτηριστεί εξαιτίας των πολλών παραλλαγών των Function Points που χρησιμοποιούνται. Αν και η IFPUG χρησιμοποιείται περισσότερο υπάρχει ένα πλήθος εναλλακτικών που επίσης χρησιμοποιείται όπως η Βρετανική Mark II,η COSMIC, η Finnish, η NESMA και η Netherlands. Αντίθετα η χρήση των Λειτουργικών Σημείων στην Ρωσία δεν έχει εξαπλωθεί ακόμη όπως στην Δυτική Ευρώπη.

Στην Ασία η αγορά είναι παρόμοια με αυτή των Η.Π.Α και της Ευρώπης . Η ανάπτυξη των Λ.Σ γίνεται κυρίως στην Ινδία ,την Ιαπωνία, την Νότια Κορέα την Κίνα και την Σιγκαπούρη με το μέγεθος των εσόδων να φτάνει τα 75,000,000$ ανά έτος. Ο ρυθμός ανάπτυξης της Ασιατικής αγοράς είναι μεγάλος λόγω της ταχείας ανάπτυξης των εφαρμογών λογισμικού. Το τοπικό κόστος στην Ινδία και στην Κίνα είναι μικρό σε σύγκριση με αυτό των Η.Π.Α

και της Ευρώπης και έχει ως αποτέλεσμα μεγαλύτερο κέρδος ανά function point. Παρόλα αυτά ο πληθωρισμός στην Ασία παραμένει υψηλός με αποτέλεσμα η διαφορές στο κόστος να μπορούν να αλλάξουν ανά πάσα στιγμή.

Σε άλλα μέρη της Ασίας όπως η Βόρεια Κορέα ,το Λάος και η Καμπότζη η ανάπτυξη των Λ.Σ. αφορά κυρίως αμυντικά συστήματα και όπλα ενώ στην Μέση Ανατολή δεν είναι εύκολο να γίνει ποσοτικοποίηση με την μεγαλύτερη ανάπτυξη να εμφανίζεται στο Ισραήλ.

ΚΕΦΑΛΑΙΟ 5

ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ

5.1 Εισαγωγή

Η μελέτη περίπτωσης αφορά την αξιολόγηση του λογισμικού Debian. Το Debian αποτελεί ίσως ένα από τα μεγαλύτερα ελεύθερα λογισμικά [31]. Η έκδοση 3.0 (Debian 3.0) με την κωδική ονομασία “woody” περιλαμβάνει πάνω από 4.500 πακέτα κώδικα. Στο παρών κεφάλαιο χρησιμοποιείται η μέθοδος του David A. Wheeler’s (SLOCCount) για να προσδιοριστεί ο αριθμός των φυσικών γραμμών κώδικα του Debian. Επίσης με χρήση της μετρικής Δομικού Μοντέλου υπολογίζεται το κόστος ανάπτυξης ενός τέτοιου λογισμικού [30].

5.2 Το λογισμικό Debian

Το Debian είναι ένα δωρεάν λογισμικό το οποίο χρησιμοποιεί τον πυρήνα του Linux . Η ανάπτυξη του Debian στηρίζεται στην συνεργασία εθελοντών από όλο τον κόσμο. Πάνω από 900 εθελοντές δουλεύουν παράλληλα και αναπτύσσουν το Debian κάνοντας το διαθέσιμο σε όλους μέσω του διαδικτύου. Το έργο των προγραμματιστών περιλαμβάνει την ανάπτυξη του λογισμικού, την διανομή του , την μετάφραση του σε αρκετές γλώσσες την υποστήριξη διάφορων διαδικτυακών υπηρεσιών (ιστοσελίδα , σύστημα εντοπισμού σφαλμάτων, λίστες ηλεκτρονικού ταχυδρομείου) και την ανάπτυξη εργαλείων ειδικά για το Debian [32].
Για να εξασφαλιστεί ότι το λογισμικό που αναπτύσσεται λειτουργεί χωρίς προβλήματα με το υπόλοιπο των προγραμμάτων υπάρχει ένα σύνολο κανόνων που ονομάζεται Εγχειρίδιο Πολιτικής του Debian (Debian Policy Manual). Το μεγαλύτερο μέρος της εργασίας για να δημιουργηθεί ένα πακέτο προγραμμάτων βρίσκεται στην εναρμόνιση του λογισμικού με όλους αυτούς τους κανόνες λειτουργίας. Οι προγραμματιστές λαμβάνουν αναφορές σφαλμάτων που καλούνται να διορθώσουν. Τα σφάλματα και οι τρύπες ασφαλείας συζητούνται ανοιχτά ενώ συνεχείς ενημερώσεις γίνονται διαθέσιμες ώστε να επιτευχθεί η μέγιστη σταθερότητα και ασφάλεια του λογισμικού [31]..
Ο μη κερδοσκοπικός χαρακτήρας του Debian , το ανοιχτό μοντέλο ανάπτυξης αλλά και το γεγονός ότι διανέμεται δωρεάν είναι κάποιοι από τους λόγους που κάνουν το Debian τόσο ξεχωριστό.

5.3 Η έκδοση Debian 3.0

Το Debian 3.0 θεωρήθηκε μία αρκετά σταθερή έκδοση. Κυκλοφόρησε τον Ιούλιο του 2002 και περιλάμβανε τα κυριότερα πακέτα ελεύθερου λογισμικού. Στην κύρια έκδοση του περιείχε περισσότερα από 4,500 πακέτα κώδικα, ενώ συνολικά περιελάμβανε 10,000 πακέτα τα οποία εύκολα μπορούσε κάποιος να βρει στο Διαδίκτυο.

Η έκδοση 3.0 χωρίζεται σε δύο κατηγορίες . Την ‘’κανονική’’ και την ‘’εκτός Ηνωμένων Πολιτειών’’. Η διανομή ‘’εκτός Ηνωμένων Πολιτειών’’ παρουσιάζει νομικά κωλύματα λόγω της αυστηρής νομοθεσίας των Η.Π.Α σχετικά με την κρυπτογράφηση του κώδικα. Για αυτό το λόγο θεωρείται ξεπερασμένη και δεν περιλαμβάνεται στην έκδοση Debian 3.1 . Στην παρούσα ανάλυση δεν γίνεται αξιολόγηση της ‘’εκτός Ηνωμένων Πολιτειών’’ διανομής παρά μόνο της ‘’κανονικής’’.

5.4 Η συλλογή των δεδομένων

Η προσέγγιση που χρησιμοποιείται για την συλλογή των δεδομένων συνοπτικά είναι:

1. Ποια κομμάτια κώδικα αποτελούν την έκδοση 3.0

Η πρόσβαση στον κώδικα του λογισμικού γίνεται εύκολα μέσω του διαδικτύου. Το μόνο πρόβλημα είναι να προσδιοριστεί ο κώδικας της συγκεκριμένης έκδοσης.

2. Λήψη και συλλογή των δεδομένων

Γνωρίζοντας ποια κομμάτια κώδικα αποτελούν την έκδοση 3.0 αυτό που μένει είναι η λήψη τους. Τα δεδομένα συλλέγονται ανά πακέτα ώστε να μην υπάρχει περίπτωση κομμάτια κώδικα να αγνοηθούν ή άλλα να μετρηθούν δύο φορές.

3. Τελική ανάλυση

Ανάλυση των στοιχείων που έχουν συλλεχθεί και λήψη στατιστικών στοιχείων σχετικά με τον αριθμό των γραμμών κώδικα κάθε πακέτου για κάθε μία από τις γλώσσες προγραμματισμού που χρησιμοποιήθηκαν.

5.4.1 Ποια κομμάτια κώδικα αποτελούν την έκδοση Debian 3.0

Στις εκδόσεις του Debian υπάρχουν δύο είδη πακέτων δεδομένων: πηγαίος και δυαδικός κώδικας. Ένα ή περισσότερα δυαδικά πακέτα μπορούν να δημιουργηθούν αυτόματα από κάθε πηγαίο κώδικα. Για παράδειγμα , από ένα συγκεκριμένο πηγαίο κώδικα μπορούν να δημιουργηθούν τρία διαφορετικά δυαδικά πακέτα: ένα με μία βιβλιοθήκη , ένα με το εκτελέσιμο πρόγραμμα και ένα τρίτο με την τεκμηρίωση του πακέτου. Σε αυτήν την ανάλυση εξετάζονται μόνο τα πακέτα πηγαίου κώδικα.

Όταν δημιουργείται ένα πακέτο κώδικα οι προγραμματιστές του Debian ξεκινούν με το πρωτότυπο κατάλογο (directory) του λογισμικού. Αυτό το

κομμάτι ονομάζεται upstream. Εάν χρειαστεί δημιουργούνται αναθεωρήσεις σε αυτόν τον κατάλογο (patches) και κατασκευάζεται ένα αρχείο με όλες τις ρυθμίσεις του λογισμικού. Όταν το πακέτο κώδικα είναι έτοιμο συνήθως αποτελείται από τρία αρχεία. Το upstream (tar.gz αρχείο) , τις ενημερώσεις (diff.gz αρχείο) και ένα αρχείο με τις τεκμηριώσεις του κώδικα με κατάληξη dsc. Στο Διαδίκτυο υπάρχουν πολλές σελίδες που περιέχουν τα πακέτα κώδικα για κάθε επίσημε έκδοση του Debian. Από την έκδοση 2.0 και σε κάθε έκδοση υπάρχει ένα αρχείο Sources.gz στον αρχικό κατάλογο του λογισμικού που περιέχει πληροφορίες σχετικά με τα πακέτα κώδικα που περιέχει αλλά και τα αρχεία που συνθέτουν κάθε πακέτο. Αυτές είναι οι πληροφορίες που χρειάζονται για να προσδιοριστεί πια πακέτα κώδικα αποτελούν την έκδοση 3.0 . Ωστόσο , χρειάζεται να γίνει ανάλυση των γραμμών κώδικα που θα προσμετρηθούν διότι είναι δυνατόν να υπάρχουν πολλές εκδόσεις από το ίδιο κομμάτι του λογισμικού. Για παράδειγμα στο Debian 3.0 μπορούμε να βρούμε τα πακέτα gcc272, gcc2.95, gcc2.96 και gcc3.0. Μετρώντας όλα αυτά τα πακέτα συνεπάγεται η καταμέτρηση του GNU C μεταφραστή τέσσερις φορές. Επομένως ένα εγχειρίδιο με τις λίστες των πακέτων είναι απαραίτητο ώστε να ανιχνευθούν ποιες αποτελούν εκδόσεις του ίδιου λογισμικού και να γίνει η επιλογή του κατάλληλου αντιπροσώπου τους.

Το αποτέλεσμα αυτής της διαδικασίας είναι η λίστα των πακέτων κώδικα που θα αναλυθούν για την αξιολόγηση του λογισμικού.

5.4.2 Λήψη και συλλογή δεδομένων

Μόλις προσδιοριστούν ποια πακέτα και ποια αρχεία αποτελούν την έκδοση 3.0 μένει να γίνει η λήψη τους από κάποιον server του Debian. Μερικά μικρά προγράμματα σε γλώσσα Perl χρησιμοποιούνται για να αυτοματοποιήσουν αυτή την διαδικασία όπου για κάθε πακέτο χρειάζεται να γίνει:

- Λήψη των αρχείων που συνθέτουν το πακέτων
- Εξαγωγή και συγκέντρωση των δεδομένων από τον κατάλογο προέλευσης
- Ενημέρωση του καταλόγου (**diff.gz** αρχείο)
- Διαγραφή του καταλόγου ώστε να αποφευχθεί να γίνει διπλή καταμέτρηση αρχείων

Η αποθήκευση των γραμμών κώδικα κάθε αρχείου γίνεται με την χρήση προγραμμάτων SLOCCount.

Το τελικό αποτέλεσμα αυτού του βήματος είναι η συλλογή των αρχείων, η οργάνωσή τους σε πακέτα και η προετοιμασία τους για ανάλυση

5.4.3 Ανάλυση των δεδομένων

Το τελευταίο βήμα είναι η στατιστική ανάλυση των δεδομένων και η δημιουργία αναφορών με την βοήθεια του SLOCCount. Σε αυτό το στάδιο τα δεδομένα είναι στην απλούστερή τους μορφή και χωρισμένα σε πακέτα οπότε η ανάλυση τους μπορεί να προχωρήσει με ταχύτητα.

5.6 Τα αποτελέσματα της ανάλυσης του Debian 3.0

Τα κυριότερα αποτελέσματα της ανάλυσης μπορούν να οργανωθούν στις ακόλουθες κατηγορίες:

- Το μέγεθος του λογισμικού Debian.
- Ποιες γλώσσες προγραμματισμού χρησιμοποιήθηκαν και σε τι ποσοστό.
- Το μέγεθος των μεγαλύτερων πακέτων.
- Εκτίμηση προσπάθειας.

5.6.1 Το μέγεθος του Debian

Ο αριθμός των γραμμών κώδικα μπορεί να εκφραστεί με τρεις διαφορετικούς τρόπους. Τα αποτελέσματα τους δίνονται παρακάτω:

- Ο αριθμός των γραμμών κώδικα από τον πρότυπο κατάλογο του λογισμικού: 98,400,000 SLOC.
- Ο αριθμός των γραμμών κώδικα του Debian: 105,500,000 SLOC.
- Ο αριθμός των γραμμών κώδικα χωρίς τον κύριο κατάλογο του Debian : 105,000,000 SLOC.

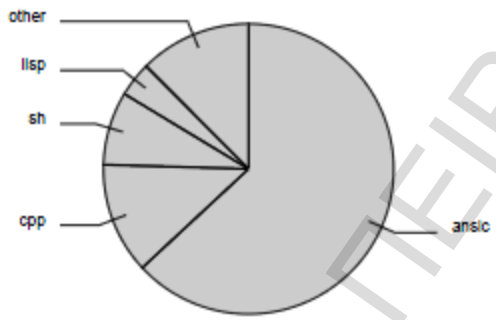
5.6.2 Γλώσσες προγραμματισμού

Ο αριθμός των φυσικών SLOC, ταξινομώντας τους ανά γλώσσα προγραμματισμού (με στρογγυλοποίηση κατά προσέγγιση) είναι ως εξής:

• C: 66.550.000 SLOC (63%)
• C + +: 13.000.000 SLOC (12,4%)
• Shell: 8.630.000 SLOC (8,1%)
• LISP: 4.080.000 SLOC (3,9%)
• Perl: 3.200.000 SLOC (3%)
• FORTRAN: 1.900.000 SLOC (1,8%)
• Python: 1.450.000 SLOC (1,4%)
• Assembler: 1.300.000 SLOC (1,3%)
• Tcl: 1.080.000 SLOC (1%)

• PHP: 648.000 SLOC (0,62%)
• Ada: 576.000 SLOC (0,55%)
• Modula3: 571.000 SLOC (0,55%)

Κάτω από το ποσοστό του 0,5% βρούμε κάποιες άλλες γλώσσες όπως Java (0,5%), Objective C (0,4%), Yacc (0,37%), ML (0,27%),και Lex (0,13%), ενώ διάφορες άλλες σχηματίζουν το υπολειπόμενο 0,1%.



Κατανομή των γλωσσών προγραμματισμού που χρησιμοποιούνται στους κώδικες του Debian 3.0.

Εικόνα 1 Κατανομή των γλωσσών προγραμματισμού.

Στο παραπάνω σχήμα μπορούμε να δούμε τη σημασία της κύριας γλώσσας σε αντιπαράθεση με τις υπόλοιπες. Αυτό συνάδει με το γεγονός ότι τα περισσότερα πακέτα είναι γραμμένα σε C. Η C++ είναι μια άλλη γλώσσα που συναντάμε σε αρκετά πακέτα, και είναι κύρια γλώσσα σε ορισμένα από αυτά (όπως στον Mozilla). Το ίδιο συμβαίνει και με τη Lisp, η οποία είναι η κύρια γλώσσα σε διάφορα πακέτα (όπως στα Emacs), αλλά επίσης χρησιμοποιείται και σε αρκετά άλλα.

Εάν μετρηθούν οι γραμμές κώδικα του Debian χωρίς να μετρηθούν αυτές του καταλόγου του οι αριθμοί είναι παρόμοιοι. Αυτό σημαίνει ότι ο κατάλογος που περιέχει τα σενάρια συντήρησης και ρύθμισης αρχείων δεν αποτελεί ένα σημαντικό μέρος συνεισφοράς σε κώδικα. Οι κυριότερες διαφορές παρατηρούνται σε σενάρια τα οποία έχουν γραφεί σε Perl η Shell.

Όμως , εάν μετρηθεί ο πρωτότυπος κώδικας (upstream) υπάρχουν αξιοσημείωτες διαφορές: περίπου 2.000.000 γραμμές κώδικα C, 300.000 γραμμές της LISP, 200.000 γραμμές της FORTRAN, και μικρές παραλλαγές και σε άλλες γλώσσες. Αυτή η διαφορά μπορεί συνήθως να ανέρχονται σε πακέτα αναβάθμισης τα οποία γίνονται από τους προγραμματιστές του Debian. Ως εκ τούτου, κοιτάζοντας αυτούς τους αριθμούς, μπορούμε να

γνωρίζουμε σε ποιες γλώσσες είναι γραμμένα τα περισσότερα πακέτα τα οποία έχουν υποστεί παραλλαγές.

5.6.3 Τα μεγαλύτερα πακέτα κώδικα.

Η συγκεκριμένη έκδοση του Debian αποτελείται ως επί των πλείστων από τα ακόλουθα πακέτα κώδικα:

Πυρήνας Linux (2.4.18): 2,574,000 SLOC. Αυτή είναι η τελευταία έκδοση του πυρήνα που περιλαμβάνεται στο Debian και είναι σχεδόν εξολοκλήρου γραμμένο σε C (2,558,000 SLOC) .

Mozilla (1.0): 2,362,000 SLOC. Η γνωστή εφαρμογή ανοιχτού κώδικα για περιήγηση στο διαδίκτυο περιλαμβάνει 1,464,000 SLOC γραμμένες σε C++ και 809,000 SLOC γραμμένες σε C.

XFree86 (4.1.0): 1,928,000 SLOC. Η XFree86 είναι μία X Window εφαρμογή που περιλαμβάνει γραφικά του διακομιστή και αποτελείται από C κώδικα(1,833,000 SLOC).

PM3 (1.1.15): 1,501,000 SLOC. Μία ανοιχτού κώδικα διανομή του μεταφραστή Modula-3 αποτελούμενο από 891,000 SLOC σε C και 545,000 σε Modula3.

MingW32 (2.95): 1,500,000 SLOC. Μία μικρογραφία του GNU Win32 μεταφραστή κυρίως γραμμένο σε C (1,137,000 SLOC).

BigLoo (2.4): 1,065,000 SLOC. Μεταφραστής της γλώσσας προγραμματισμού Scheme αποτελούμενος από 948,000 SLOC σε C και 96,000 SLOC σε LISP.

GDB (5.2): 986,000 SLOC. Διορθωτής του λειτουργικού GNU με 904,000 SLOC γραμμένες σε C.

crash (3.3): 968,000 SLOC. Εργαλείο διόρθωσης σφαλμάτων του πυρήνα γραμμένο κυρίως σε C (890,000 SLOC σε C).

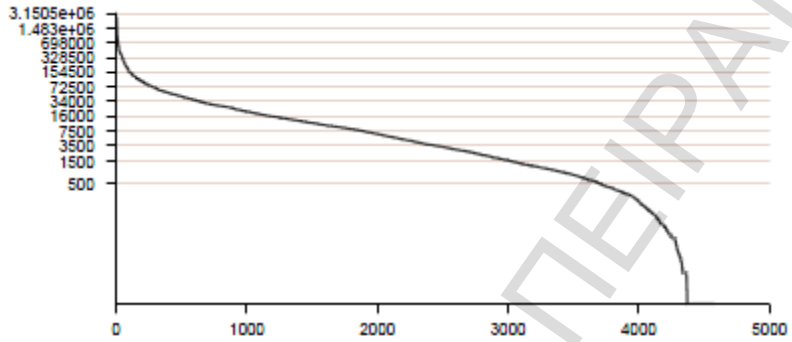
OSKit (0.97): 921,000 SLOC. Πακέτο εργαλείων για την διευκόλυνση ανάπτυξης του πυρήνα. (912,000 SLOC σε C).

NCBI libraries (6.1): Πακέτο από βιβλιοθήκες σχετικές με την ανάπτυξη εφαρμογών βιολογίας (828,000 SLOC γραμμένες σε C).

Οι αριθμοί των SLOC είναι κατά προσέγγιση και η ταξινόμηση τους θα μπορούσε να είναι λίγο διαφορετική ανάλογα με το πώς έχουν δημιουργήσει τα πακέτα οι προγραμματιστές. Για παράδειγμα εάν όλες οι καταλήξεις Emacs

ήταν στα πακέτα Emacs , αυτά θα ήταν κατά πολύ μεγαλύτερα και βρίσκονταν μέσα στα στην λίστα με τα 10 μεγαλύτερα πακέτα. Όμως η κατανομή σε πακέτα κώδικα προσαρμόζεται σε αυτό που οι προγραμματιστές εννοούν πακέτο κάθε φορά καθώς και στην γενικότερη ιδέα του πακέτου.

Η κυριότερη γλώσσα προγραμματισμού φαίνεται να είναι η C ενώ στα πακέτα Emacs υπάρχει πολύς κώδικας γραμμένος σε LISP και αρκετός κώδικα σ σε Ada στο μεταφραστή Gnat [3333].



Στην εικόνα φαίνεται το μέγεθος των πακέτων του Debian. Στον οριζόντιο άξονα είναι το μέγεθος των πακέτων ενώ στον κάθετο παρουσιάζεται λογαριθμικά ο αριθμός των SLOC.

Εικόνα 2 . Κατανομή πακέτων Debian

5.7 Προσπάθεια και εκτίμηση κόστους

Χρησιμοποιώντας το βασικό μοντέλο COCOMO μπορούμε να προσδιορίσουμε το μέγεθος της προσπάθειας που απαιτείται για να φτιαχτεί ένα λογισμικό σαν το Debian. Λόγω του ιδιαίτερου τρόπου δημιουργίας του Debian δεν είναι σίγουρο εάν το ποσοστό της προσπάθειας, είναι αυτό που πράγματι έχει δαπανηθεί αλλά παρόλα αυτά δίνεται μία πολύ καλή εικόνα της τάξης του μεγέθους της προσπάθειας που χρειάζεται για την δημιουργία ενός παρόμοιου μεγέθους έργου [33].

Χρησιμοποιώντας την μέτρηση σε SLOC για τα πακέτα κώδικα το Δομικό Μοντέλο δίνει τα εξής αποτελέσματα:

- Συνολικός αριθμός SLOC : 104,679,026
- Εκτιμούμενη προσπάθεια: 322,031.16 άνθρωπο-μήνες ή 26,835.93 άνθρωπο-χρόνια.
- Εκτίμηση περάτωσης έργου : 81.72 μήνες ή 6.81 χρόνια.
- Εκτιμούμενο κόστος έργου : 3,625,000,000 USD

Για αυτά τα στοιχεία κάθε έργο εκτιμήθηκε σαν να ήταν ξεχωριστό και ανεξάρτητο από τα άλλα, που σε γενικές γραμμές είναι αληθές. Για τον

υπολογισμό του κόστους χρησιμοποιήθηκε ο βασικός μισθός ενός προγραμματιστή κατά την διάρκεια του 2000, σύμφωνα με το Computer World , ο οποίος ανερχόταν στα 56,286 USD το χρόνο.

Οι αριθμοί της προηγούμενης ενότητας δεν είναι τίποτα παραπάνω από εκτιμήσεις. Δίνουν την τάξη μεγέθους του λογισμικού και επιτρέπουν συγκρίσεις αλλά δεν θα πρέπει να λαμβάνονται ως ακριβή δεδομένα.

5.8 Η μέθοδος SLOCCount

Η SLOCCount είναι μία σουίτα προγραμμάτων για την καταμέτρηση φυσικών γραμμών κώδικα (SLOC) μεγάλων λογισμικών [34]. Η SLOCCount αποτελεί ένα εργαλείο μέτρησης λογισμικού ανεπτυγμένο από τον David A. Wheele. Αρχικά χρησιμοποιήθηκε σε διανομές GNU/Linux αλλά εφαρμόζεται και σε άλλα ανεξάρτητα λογισμικά. Έχει εφαρμοστεί στις εκδόσεις 6.2, 7 και 7.1 του Red Hat ενώ με τις κατάλληλες προσαρμογές μπορεί να χρησιμοποιηθεί και σε συστήματα Windows. Η SLOCCount είναι συμβατή με αρκετές γλώσσες προγραμματισμού. Αλφαβητικά αυτές είναι η Ada, Assembly awk ,Bourne shell, C, C++, C-sharp, C shell, COBOL, Expect, Fortran, Haskell, Java, lex , LISP, Modula3, Objective-C, Pascal, Perl, PHP, Python, Ruby, sed, SQL, TCL, και Yacc. Μπορεί εύκολα να διαχειριστεί δύσκολες καταστάσεις και να διαχειριστεί τις συντακτικές ιδιαιτερότητες της κάθε γλώσσας ώστε να τις προσαρμόσει κατάλληλα. Η SLOCCount μπορεί να αναγνωρίσει ένα μεγάλο πλήθος αρχείων και να το κατατάξει ανάλογα, σε όποια γλώσσα και αν είναι γραμμένο. Χρησιμοποιεί διάφορα εργαλεία συλλογής δεδομένων και έχει την δυνατότητα να τα παρουσιάσει σε αρκετές μορφές.

5.8.1 Πηγές ανακρίβειας των SLOC μετρήσεων

Οι εκτιμήσεις όσων αφορά τις SLOC δεν είναι δυνατό να είναι απόλυτα ακριβής . Ειδικότερα όσων αφορά τα πακέτα δεδομένων ανακρίβειες αριθμών ή ανακρίβειες επιλογής πακέτων είναι πιθανές. Οι παράγοντες που μπορεί να επηρεάσουν την ακρίβεια των μετρήσεων είναι:

- Ορισμένα αρχεία μπορεί να μην μετρηθούν με ακρίβεια.

Αν και η τεχνική καταμέτρησης γραμμών κώδικα (SLOCCount) είναι σχεδιασμένη με τέτοιο τρόπο ώστε να μπορεί να ξεχωρίσει τις γραμμές κώδικα από τα σχόλια σε πολλές περιπτώσεις σε πολλές περιπτώσεις μπορεί να μην συμβεί η αναμενόμενη λειτουργία. Επίσης σε πολλές περιπτώσεις είναι δύσκολο να γίνει η διάκριση στα αρχεία που δημιουργούνται από άλλα υπό προγράμματα (και δεν θα έπρεπε να μετρηθούν) και στα μετρούμενα αρχεία.

- Δεν αναγνωρίζονται όλες οι γλώσσες προγραμματισμού

Η τελευταία έκδοση του SLOCCount που χρησιμοποιήθηκε στην αξιολόγηση του Debian μπορεί να αναγνωρίσει περισσότερες από 20 γλώσσες προγραμματισμού. Ωστόσο υπάρχουν ορισμένες που δεν υποστηρίζονται από την μετρική , όπως η Erlang, δημιουργώντας έτσι ανακρίβειες στα πακέτα που είναι γραμμένα σε αυτές τις γλώσσες.

- Διαφορετικές επιλογές στην ομαδοποίηση των πακέτων

Υπάρχουν πολλά κριτήρια για το εάν ένα πακέτο κώδικα θα πρέπει να προσμετρηθεί ή όχι. Περιπτώσεις όπου κάποιο πακέτο κώδικα περιλαμβάνεται σε περισσότερα από ένα πακέτα ή περιπτώσεις διαφορετικών εκδόσεων του ίδιου κώδικα δημιουργούν αστοχίες στην καταμέτρηση.

5.9 Εκτίμηση της προσπάθεια και του κόστους

Τα κλασικά μοντέλα εκτίμησης κόστους και συγκεκριμένα το Δομικό Μοντέλο, εξετάζουν τα λογισμικά από την κλασική ιδιόκτητη σκοπιά. Ωστόσο τα ελεύθερα μοντέλα ανάπτυξης λογισμικού όπως το Debian είναι μάλλον διαφορετικά και ως εκ τούτου στα μοντέλα αυτά δεν μπορεί να είναι άμεσα εφαρμόσιμα. Με αυτές τις τεχνικές υπολογίζεται το κόστος ενός λογισμικού σε προσπάθεια ή και σε χρήμα εάν αυτό είχε αναπτυχθεί με τα κλασικά πρότυπα ανάπτυξης χωρίς όμως αυτό να είναι το πραγματικό.

Μερικές από τις διαφορές που καθιστούν αδύνατη τη χρήση αυτών των μοντέλων εκτίμησης είναι:

Η συνεχής παραγωγή νέων εκδόσεων : Τα ελεύθερα λογισμικά έχουν το χαρακτηριστικό να εμφανίζουν νέες εκδόσεις με γρήγορο ρυθμό. Τα προγράμματα αυτά τροποποιούν το περιεχόμενο τους σχεδόν την ίδια στιγμή που είναι παραδοτέα στους χρήστες.

Προβλήματα και διορθώσεις: Ενώ τα ιδιόκτητα λογισμικά απαιτούν ακριβές διαδικασίες εντοπισμού και διόρθωσης σφαλμάτων τα ελεύθερα λογισμικά στηρίζονται στις αναφορές ή ακόμα και στις διορθώσεις ανθρώπων εκτός έργου.

Επαναχρησιμοποίηση και εξέλιξη του κώδικα: Είναι κοινό στα ελεύθερα λογισμικά η επαναχρησιμοποίηση του κώδικα και αποτελεί αναπόσπαστο κομμάτι της εξέλιξης τους. Σε πολλά τέτοια προγράμματα χρησιμοποιείται συχνά ο κώδικας άλλων ελεύθερων λογισμικών γεγονός σπάνιο σε ιδιόκτητα έργα.

Κατανεμημένο μοντέλο ανάπτυξης: Παρά το γεγονός ότι ορισμένα ιδιόκτητα έργα αναπτύσσονται σε ομάδες διασκορπισμένες γεωγραφικά ο βαθμός κατάτμησης των ελεύθερων λογισμικών είναι αρκετές τάξεις μεγαλύτερος. Υπάρχουν εξαιρέσεις αλλά είναι σύνηθες τα έργα ελεύθερων λογισμικών να εκτελούνται από ανθρώπους διαφορετικών χωρών που δεν εργάζονται στην ίδια εταιρία, που αφιερώνουν διαφορετικό ποσό προσπάθειας και που αλληλεπιδρούν κυρίως από το Διαδίκτυο.

Μερικοί από αυτούς τους παράγοντες αυξάνουν την προσπάθεια που απαιτείται για την κατασκευή του λογισμικού ενώ κάποιοι άλλοι την μειώνουν. Χωρίς να αναλύονται λεπτομερώς οι επιπτώσεις αυτών των παραγόντων , τα μοντέλα εκτίμησης εν γένει, και ειδικότερα το Δομικό Μοντέλο, μπορεί να μην είναι άμεσα εφαρμόσιμα στα έργα ελεύθερου λογισμικού.

5.10 Σύγκριση με άλλα λειτουργικά

Για να γίνει ευκολότερα η σύγκριση του μεγέθους του Debian ακολουθεί το εκτιμώμενο μέγεθος γραμμών κώδικα μερικών γνωστών λειτουργικών.

• Microsoft Windows 3.1: 3,000,000

• Sun Solaris: 7,500,000

• Microsoft Windows 95: 15,000,000

• Red Hat Linux 6.2: 17,000,000

• Microsoft Windows 2000: 29,000,000

• Red Hat Linux 7.1: 30,000,000

• Microsoft XP: 40,000,000

• Red Hat Linux 8.0: 50,000,000

• Debian 2.2: 55,000,000

• Debian 3.0: 105,000,000

Οι περισσότερες από αυτές τις εκτιμήσεις δεν είναι αρκετά λεπτομερείς ωστόσο γίνονται με παρόμοιες μεθόδους της SLOC δίνοντας έτσι αποτελέσματα κατάλληλα για σύγκριση. Ωστόσο η ακρίβεια τους είναι ικανοποιητική και είναι δυνατόν να εξαχθούν χρήσιμα συμπεράσματα και να γίνουν συγκρίσεις με άλλα συστήματα.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Ο υπολογισμός του κόστους ανάπτυξης ενός έργου λογισμικού στον τομέα της Τεχνολογίας Λογισμικού, είναι μια δραστηριότητα δύσκολη αλλά και πολύ σημαντική στο επίπεδο της διαχείρισης ενός οργανισμού ή μιας επιχείρησης. Ειδικότερα, ο υπολογισμός εντός χρόνου και εντός χρηματικού κόστους είναι κρίσιμη για την βιωσιμότητα οργανισμών που αναλαμβάνουν την διεκπεραίωση ενός έργου.

Υπάρχουν πολλά μοντέλα αξιολόγησης λογισμικού. Η αξιολόγηση κατ 'αναλογία είναι ένα από αυτά όπου η εκτίμηση του κόστους γίνεται συγκρίνοντας προηγούμενα παρόμοιου μεγέθους έργα στο ίδιο πεδίο εφαρμογής. Το κύριο πλεονέκτημα είναι ότι η εκτίμηση βασίζεται σε πραγματικά χαρακτηριστικά ανάλογου έργου.

Η εμπειροτεχνική μέθοδος είναι ένα άλλο μοντέλο που πραγματοποιείται με την ανάμειξη ειδικών οι οποίοι χρησιμοποιούν την εμπειρία τους και τις γνώσεις τους πάνω σε παρόμοια έργα ώστε να καταλήξουν σε μία εκτίμηση κόστους.

Άλλη τεχνική εκτίμησης ενός έργου είναι η ανάλυση “από πάνω προς τα κάτω” (top down) η οποία απαιτεί ελάχιστες λεπτομέρειες για το έργο και είναι συνήθως αρκετά γρήγορη και εύκολη στην εφαρμογή. Η μέθοδος με την ακριβώς αντίθετη προσέγγιση είναι η “από κάτω προς τα πάνω” (bottom up) η οποία παρουσιάζει μεγαλύτερη σταθερότητα.

Τα αλγοριθμικά μοντέλα αξιολόγησης όπως το Δομικό Μοντέλο Κόστους και η Ανάλυση Λειτουργικών Σημείων χρησιμοποιούν μαθηματικές εξισώσεις για τις εκτιμήσεις τους. Αυτές στηρίζονται σε εμπειρικά δεδομένα και γνώσεις σε παρόμοια έργα ενώ έχουν την δυνατότητα να συνυπολογίζουν και άλλους παράγοντες όπως η γλώσσα σχεδιασμού, η μεθοδολογία αλλά και παράγοντες κινδύνου.

Σε αυτά τα μοντέλα παρουσιάστηκαν αρκετές δυνατότητες. Κάποιες από αυτές είναι η αντικειμενικότητά τους, η μεγάλη αποτελεσματικότητα τους, το γεγονός ότι μπορούν να βελτιώσουν τις αποδόσεις τους όταν αυξάνεται η εμπειρία ενώ ταυτόχρονα μπορούν να παρέχουν τυποποιημένες αναλύσεις.

Η Ανάλυση Λειτουργικών Σημείων έχει ξεπεράσει κατά πολύ παλαιότερες μετρικές όπως αυτή των γραμμών κώδικα . Παρουσιάζει πολλές χρησιμότητες καθώς σε ένα λογισμικό μπορεί να κάνει μετρήσεις κόστους ανά λάθος μπορεί να μετρήσει μη κωδικοποιημένες δραστηριότητες όπως η διαχείριση και ο σχεδιασμός του έργου αλλά κυρίως μένει ανεξάρτητη της γλώσσας προγραμματισμού που χρησιμοποιείται.

Η Ανάλυση Λειτουργικών Σημείων είναι αποτελεσματικότερη στα μεγάλα έργα πληροφορικής. Εκεί παρουσιάζεται μεγαλύτερο ρίσκο, υψηλότερος κίνδυνος αποτυχίας και μεγαλύτερες χρονικές καθυστερήσεις. Η έγκυρη χρήση των Λειτουργικών Σημείων θα βοηθήσει στην αποφυγή τέτοιων προβλημάτων και θα αποτρέψει υπερβάσεις στον προϋπολογισμό του έργου.

Έργα μικρότερα των 15 Λειτουργικών Σημείων δεν αξιολογούνται συνήθως με την ανάλυση Λειτουργικών σημείων καθώς η χρήση τους κρίνεται ασύμφορη οικονομικά. Αντίθετα σε περιπτώσεις έργων έως 15.000 Λειτουργικών Σημείων η χρήση τους κρίνεται εξαιρετικά αποδοτική.

Η αξιολόγηση με ανάλυση Λειτουργικών Σημείων είναι εφικτή σχεδόν σε κάθε έργο πληροφορικής .Η κυριότερη χρήση τους παρατηρείται σε έργα που σχετίζονται με χρηματοοικονομικούς οργανισμούς ή μεγάλες τράπεζες. Εφαρμόζεται επίσης σε έργα σχετικά με την ασφάλεια λογισμικού ή διαδικτύου αλλά και σε πολλά έργα πληροφορικής που αφορούν τηλεπικοινωνίες και δίκτυα επικοινωνιών .

Το Δομικό Μοντέλο Κόστους είναι ένα μοντέλο αξιολόγησης λογισμικού σχεδιασμένο ώστε να δώσει μια εκτίμηση του αριθμού των μηνών εργασίας που χρειάζεται να καταβάλει ένας άνθρωπος ώστε να υλοποιηθεί ένα προϊόν λογισμικού. Βασίζεται σε μια μελέτη αρκετών έργων. Τα έργα πληροφορικής που αξιολογούνται με αυτό το μοντέλο κυμαίνονται συνήθως από 2000 με 100,000 γραμμές κώδικα, χρησιμοποιώντας αρκετές γλώσσες προγραμματισμού. Το COCOMO αποτελείται από μια ιεραρχία τριών όλο και περισσότερο λεπτομερών και ακριβών μορφών. Το πρώτο επίπεδο, βασικό COCOMO είναι καλό για τις γρήγορες, πρόωρες, κατά προσέγγιση εκτιμήσεις μεγέθους του κόστους λογισμικού, αλλά η ακρίβειά του είναι περιορισμένη λόγω της έλλειψης συντελεστών του (οδηγοί δαπανών, Cost Drivers). Το Ενδιάμεσο COCOMO λαμβάνει αυτούς τους οδηγούς δαπανών υπόψη και το Λεπτομερές COCOMO επιπρόσθετα λαμβάνει υπόψη την επιρροή μεμονωμένων φάσεων του προγράμματος.

Το Δομικό Μοντέλο Κόστους έχει ιδιαίτερη απήχηση και αποδοχή. Είναι σχετικά εύκολο στην χρήση και η διαδικασία εκτίμησης μπορεί να προσαρμοστεί, σε διαφορετικές συνθήκες κάθε φορά, χρησιμοποιώντας τους συντελεστές προσαρμογής. Παρά τα όποια μειονεκτήματα εξακολουθεί να αποτελεί ένα πανίσχυρο εργαλείο για την πρόβλεψη κόστους ενός λογισμικού.

Η ακριβής εκτίμηση για το κόστος υλοποίησης ενός λογισμικού είναι μια κρίσιμη διαδικασία και η επιτυχία της εξαρτάται από πολλούς παράγοντες. Καμία μέθοδος δεν είναι κατ 'ανάγκην καλύτερη ή χειρότερη από την άλλη, στην πραγματικότητα, τα πλεονεκτήματα και τα μειονεκτήματα τους αλληλοσυμπληρώνονται.

Στην αξιολόγηση του λογισμικού Debian 3.0 που αναλύθηκε στη μελέτη περίπτωσης παρουσιάστηκαν οι εκτιμήσεις για το μέγεθος του. Το λογισμικό εκτιμήθηκε ότι αποτελούταν από 105,000,000 SLOC περίπου. Χρησιμοποιώντας την μετρική COCOMO εκτιμήθηκε ότι ένα τέτοιου μεγέθους λογισμικό εάν ήταν ιδιόκτητο θα κόστιζε περίπου 3,625 εκατομμύρια δολάρια ενώ θα απαιτούνταν περισσότερο από 26,800 άνθρωπο-έτη για να περατωθεί. Παρουσιάστηκαν στοιχεία για τα μεγαλύτερα πακέτα κώδικα του λογισμικού αλλά και τις γλώσσες προγραμματισμού που τα αποτελούσαν. Στα στοιχεία αυτά δεν περιλαμβάνονται προγράμματα όπως το OpenOffice.org το οποίο δεν περιλαμβάνεται στην έκδοση που έγινε το καλοκαίρι του 2002 κάτι το οποίο έγινε στην επόμενη έκδοση (Debian 3.1). Το μέγεθος του Debian είναι αρκετά μεγάλο. Αυτό γίνεται αντιληπτό συγκρίνοντας το με άλλες διανομές βασισμένες σε Linux , όπως το Red Hat 8, το οποίο εκδόθηκε 2 μήνες αργότερα και έχει το μισό περίπου μέγεθος ενώ ξεπερνά σε μέγεθος και άλλα γνωστά λειτουργικά όπως αυτά της Microsoft.

Περνώντας σε λεπτομέρειες γίνεται φανερό ότι η πιο δημοφιλής γλώσσα προγραμματισμού είναι η C (περισσότερο από 60%), ακολουθούμενη από C++ (κοντά στο 12%), η Shell (περίπου 8%), LISP (περίπου 4%), Perl (περίπου 3%) και FORTRAN και Python (περίπου 2%).

Τα μεγαλύτερα πακέτα κώδικα αποτελούν ο πυρήνας του Linux (περίπου 2,574,000 SLOC), το Mozilla (περίπου 2,362,000 SLOC), το XFree86 (περισσότερα από 1,900,000), και το PM3 με περίπου 1,501,000 SLOC.

Από αυτούς τους αριθμούς, μπορούμε να δούμε ότι οι διανομές Linux και ειδικότερα το Debian 3.0 αποτελούν μερικά από τα μεγαλύτερα κομμάτια λογισμικού που έχουν δημιουργηθεί ποτέ από κάποια ομάδα προγραμματιστών.

ΑΝΑΦΟΡΕΣ

1. Walsam G. Interpreting Information Systems in Organizations. Wiley, Chichester; 1993.
2. Stefan Cronholm and Göran Goldkuhl , Actable Information Systems – Quality Ideals Put Into Practice; 2002.
3. Olegas Vasilecas, Algis Saulis, Saulius Dereškevičius, Evaluation of Information Systems Procurement: Goal and Task -Driven Approaches
4. Michael Quinn Patton. Qualitative Evaluation and Research Methods; 1990.
5. Stefan Cronholm and Göran Goldkuhl. Strategies for Information Systems Evaluation- Six Generic Types; 2003.
6. Monk A, Wright P, Haber J, Davenport L. Improving your Human-Computer Interface. New York: Prentice-Hall; 1993.
7. Karen Lum. Handbook for Software Cost Estimation; 2003.
8. Steve McConnell, Software Estimation; 2012.
9. Albrecht, A. Software Function, Source Lines of Code, and Development Effort Estimation - A Software Science Validation; 1983.
10. Christopher Rush & Rajkumar Roy. Expert judgement in cost estimating: Modelling the reasoning process; 2001.
11. Boehm, Barry W. Software Engineering Economics; 1981.
12. Chris F. Kemerer. An Empirical Validation of Cost Estimation Models; 1987.
13. Barry W. Boehm , Chris Abts, Software Cost Estimation with Cocomo II; 2000.
14. International Function Point Users Group (IFPUG). Function Point Counting Practice Manual Release 4.1.1
15. David Garmus and David Herron. Function Point Analysis. Measurement Practices for Succesful Software Projects; 2000
16. Albrecht Allan. Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation; 1983.
17. Symons Charles. Function Points Analysis: Difficulties and Improvements; 1988.

18. Whitmire, S. A 3-D Function Points: Scientific and Real-Time Extensions to Function Points´, Proceedings of the 1992 Pacific Northwest Software Quality Conference; 1992.

19. Capers Jones. A New Business Model For Function Point Metrics; 2008.

20. IFPUG Counting Practices Manual, Release 4, International Function Point Users Group, Westerville, Ohio; 1995.

21. Abran A. Analysis of the measurement process of Function Point Analysis; 1994.

22. Roger E. Software Metrics: An Analysis of the Evolution of COCOMO and Function Points; 1997.

23. Roger S. Pressman Software Engineering: A Practitioner's Approach, 7/e; 1996.

24. VanSuetendael. N Software Quality Metrics; 1991.

25. StoneHedge Partners, INC www.stonehenge.org

26. International Software Benchmarking Standards Group. Διαθέσιμο από:www.isbsg.or

27. International Function Point Users Group. Διαθέσιμο από: www.ifpug.org

28. Software Productivity Research. Διαθέσιμο από:www.spr.com

29. Symons C. Software sizing and estimating: Mk II FPA .Function Point Analysis. John Wiley & Sons, Inc.: New York , NY, USA; 1991.

30. Hemmstra F. and Kusters R. Function point analysis: evaluation of a software cost estimation model; 1991.

31. Juan Jose Amor, Gregorio Robles, and Jesus M. Gonzalez. Barahona: Measuring Woody: The Size of Debian 3.0; December 2004.

32. Debian GNU/Linux 3.0 released, Debian Project, Διαθέσιμο από: www.nl.debian.org

33. Gregorio R. and Jesus M. Gonzalez-Barahona. Toy Story: an analysis of the evolution of Debian GNU/Linux ; 2004

34. David A. Wheeler. More Than a Gigabuck: Estimating GNU/Linux's Size; 2001