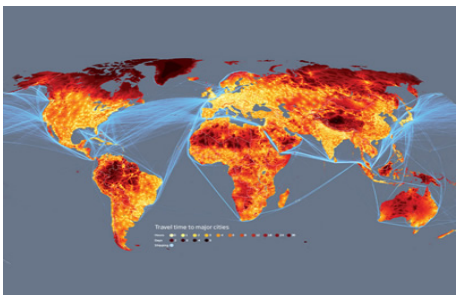

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ


ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

# ΑΝΑΠΤΥΞΗ ΥΠΟΣΤΗΡΙΚΤΙΚΟΥ ΕΡΓΑΛΕΙΟΥ ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΩΝ ΠΟΛΛΑΠΛΩΝ ΣΥΝΔΕΣΕΩΝ ΜΕ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ




**ΑΝΑΣΤΑΣΙΟΣ Π. ΣΤΑΜΕΛΟΣ**

**ΕΠΙΒΛΕΠΩΝ : ΔΗΜΗΤΡΙΟΣ ΑΣΚΟΥΝΗΣ**

**ΑΝ. ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.**



**Αθήνα, Μάρτιος 2015**


ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ


# ΑΝΑΠΤΥΞΗ ΥΠΟΣΤΗΡΙΚΤΙΚΟΥ ΕΡΓΑΛΕΙΟΥ ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΩΝ ΠΟΛΛΑΠΛΩΝ ΣΥΝΔΕΣΕΩΝ ΜΕ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ


**ΑΝΑΣΤΑΣΙΟΣ Π. ΣΤΑΜΕΛΟΣ**

**ΕΠΙΒΛΕΠΩΝ : ΔΗΜΗΤΡΙΟΣ ΑΣΚΟΥΝΗΣ**

**ΕΠ. ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.**


Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26η Μαρτίου 2015.

| ..... | ..... | ..... |
|---|---|---|
| **ΔΗΜΗΤΡΙΟΣ ΑΣΚΟΥΝΗΣ** | **ΙΩΑΝΝΗΣ ΨΑΡΡΑΣ** | **ΒΑΣΙΛΕΙΟΣ ΑΣΗΜΑΚΟΠΟΥΛΟΣ** |
| **ΑΝ. ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.** | **ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.** | **ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.** |

.....

Αναστάσιος Π. Σταμέλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

# ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα αποτελεί τη διπλωματική μου εργασία στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσοβίου Πολυτεχνείου, η οποία εκπονήθηκε κατά τη διάρκεια του ακαδημαϊκού έτους 2014–2015. Πρόκειται για μία εκτενή προσπάθεια εξερεύνησης του πεδίου της λογισμικής μηχανικής.

Θέλω να εκφράσω τις θερμές ευχαριστίες μου στον Καθηγητή μου κ. Δημήτρη Ασκούνη για την τιμή που μου έκανε να με συμπεριλάβει υπό την καθοδήγηση του και για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα.

Θα ήθελα, επίσης, ιδιαίτερα να ευχαριστήσω τον Διδάκτορα Ηλεκτρολόγο Μηχανικό και Μηχανικό Ηλεκτρονικών Υπολογιστών κ. Χρήστο Ντάνο, για την πολύτιμη και εμψυχωτική καθοδήγηση και συνεχή υποστήριξή του σε όλα τα στάδια και επίπεδα διεκπεραίωσης της εργασίας αυτής.

Ακόμα, θέλω να ευχαριστήσω τον Διδάκτορα Ηλεκτρολόγο Μηχανικό και Μηχανικό Ηλεκτρονικών Υπολογιστών κ. Γιώργο Κουρλιμπίνη για τη σημαντική συμβολή του μέσω της πολύτιμης εμπειρίας του σχετικά με το σύγχρονο περιβάλλον ανάπτυξης πληροφοριακών συστημάτων.

Τέλος, ευχαριστώ θερμά την οικογένειά μου και τους φίλους μου, για την συμπαράσταση τους καθ’ όλη τη διάρκεια των σπουδών μου.

Η εργασία αυτή αφιερώνεται σε όλους τους ανθρώπους χάρη στους οποίους κατέστη δυνατή η ανάπτυξή της.

# Πρόλογος

Στη σύγχρονη εποχή ο άνθρωπος χρησιμοποιεί και βασίζεται ολοένα και περισσότερο στην σύγχρονη τεχνολογία πληροφοριακών και υπολογιστικών συστημάτων, τα οποία εξελίσσονται ραγδαία. Βιώνουμε την κοινωνία της πληροφορίας, ενώ παράλληλα συνδιαλλασσόμαστε στην ψηφιακή οικονομία. Επίσης, η διείσδυση της προηγμένης τεχνολογίας αποτελεί αυτονόητη έννοια για τον σύγχρονο επιχειρηματικό κόσμο, και τείνει να γίνει πραγματικότητα ακόμα και στο οικιακό περιβάλλον (Διαδίκτυο των αντικειμένων).

Η πρόκληση, σήμερα, είναι αφενός πως από την κοινωνία της πληροφορίας θα μεταβούμε στην κοινωνία της Γνώσης (web v2.0, τεχνητή νοημοσύνη, κλπ) και αφετέρου πώς θα σχεδιάσουμε συστήματα υψηλής αξιοπιστίας και εγγυημένης λειτουργικότητας, τα οποία θα λειτουργούν ικανοποιητικά παρά τα όποια προβλήματα ενδεχομένως προκύψουν.

Οι σύγχρονες προκλήσεις συνοψίζονται στην διασύνδεση του φυσικού κόσμου με την επιστήμη της πληροφορικής (επαυξημένη πραγματικότητα), στην ενιαιοποίηση και τη σύνθεση των συστημάτων με θεμελιώδεις θεωρητικούς κανόνες και τέλος στην ενσωμάτωση ευφυΐας στα συστήματα αυτά.

Κοινός τόπος όλων παραμένει η πληροφορία, η επεξεργασία της και οι αλληλεπιδράσεις της, και κατ' επέκταση σημείο εστίασης αποτελεί η συνεχής βελτίωση των πληροφοριακών συστημάτων και η ολοκλήρωση τους με τα υπολογιστικά συστήματα.

Η παρούσα εργασία συγκροτείται από δύο θεματικές ενότητες. Τη βιβλιογραφική ανασκόπηση σχετικά με την λογισμική μηχανική και την ανάπτυξη πληροφοριακών συστημάτων αφενός και αφετέρου την εφαρμογή των αρχών τους για τη δημιουργία ενός υποστηρικτικού πρωτότυπου εργαλείου για την ανάπτυξη εφαρμογών με πολλαπλές συνδέσεις με βάσεις δεδομένων.

Η θεωρητική ενότητα περιλαμβάνει τα τέσσερα πρώτα κεφάλαια, στα οποία περιγράφονται και αναλύονται οι βασικές αρχές και έννοιες της αξιοποίησης της λογισμικής μηχανικής στην ανάπτυξη πληροφοριακών συστημάτων. Αναλύονται οι μεθοδολογίες ανάπτυξης πληροφοριακών συστημάτων από την οπτική της διαχείρισης, με έμφαση στις ευέλικτες μεθόδους.

Στην πρακτική ενότητα, που αποτελεί το πέμπτο κεφάλαιο, παρουσιάζεται η δημιουργία ενός νέου καινοτόμου υποστηρικτικού λογισμικού εργαλείου, το ABC, από τα αρχικά των λέξεων **A**pplication & Data**B**ase **C**orrelation, και περιγράφεται ο τρόπος με τον οποίο προδιαγράφηκε, σχεδιάστηκε, υλοποιήθηκε, ελέγχθηκε, καθώς και η έκδοση και λειτουργία του σε πραγματικό έργο.

Τέλος, παρατίθενται τα συμπεράσματα σχετικά με τις μεθοδολογίες και οι προτάσεις για τη χρήση του εργαλείου ABC και τον εμπλουτισμό του με περισσότερες δυνατότητες.

# Abstract

Today, people increasingly use and depend on advanced information and computing technology. We are experiencing information society, while transacting in the digitalized economy. Moreover, the penetration of advanced technology is a fact for the modern business world, and tends to become a reality for the house environment (Internet of Things).

The challenge now is one hand the transition from an information society to a Knowledge Society (web v2.0, artificial intelligence, etc.) and the ability to design highly reliable and guaranteed functional systems, which will be satisfactorily operational despite any problems that may arise.

The current challenges are summarized in the following: connecting the natural world with computer science (Augmented Reality), unifying and synthesizing systems through fundamental theoretical rules and finally, integrating intelligence in these systems.

Considering all the above, the remaining common ground is information, its processing and interactions, and hence the focal point is the aim for continuous improvement of the development of information systems.

In this context, the present dissertation consists of two modules: On one hand, the literature review on software engineering in the development of information systems, and on the other one, their implementation in creating a prototype supporting software tool for the development of applications with multiple database connections.

The theoretical section is laid out on the first four chapters, which describe and analyze the basic principles and concepts of software-engineering in the development of information systems. Information systems development methodologies are analyzed from a managerial perspective, with emphasis on agile methods.

The second section -which consists of chapter five of this dissertation-, is a practical application which relates to the aforementioned analysis. It details the creation of a supporting software tool, named “ABC” after the **A**pplication & Data**B**ase **C**orrelation, and the involving stages of its development process regarding the problem and requirements recognition, the design, the deployment and testing concluding with its first operational edition.

Finally, the last chapter summarizes the conclusions regarding software development methodologies and suggests recommendation for further enrichment of the ABC tool with more features in order to provide broader support to system developers.

*Λέξεις κλειδιά:*

Λογισμική Μηχανική, Ανάπτυξη πληροφοριακών συστημάτων, Προδιαγραφές, Σχεδιασμός, Υλοποίηση, Παραδοσιακές και Ευέλικτες μεθοδολογίες, Scrum, Γλώσσα Προγραμματισμού C#, .NET, Visual Studio 2013, SQL Server 2014 Management Studio: σύνδεση με SQL Server, ανάκτηση, επεξεργασία και παρουσίαση πληροφοριών, Υποστηρικτικό εργαλείο ABC

*Key Words:*

Software engineering, Software Development, Requirements, Design, Deployment, Waterfall, Traditional and Agile Methods, Scrum, Programming Language C#, Visual Studio 2013, SQL Server 2014 Management Studio: retrieving, processing and presenting data, ABC supportive tool.

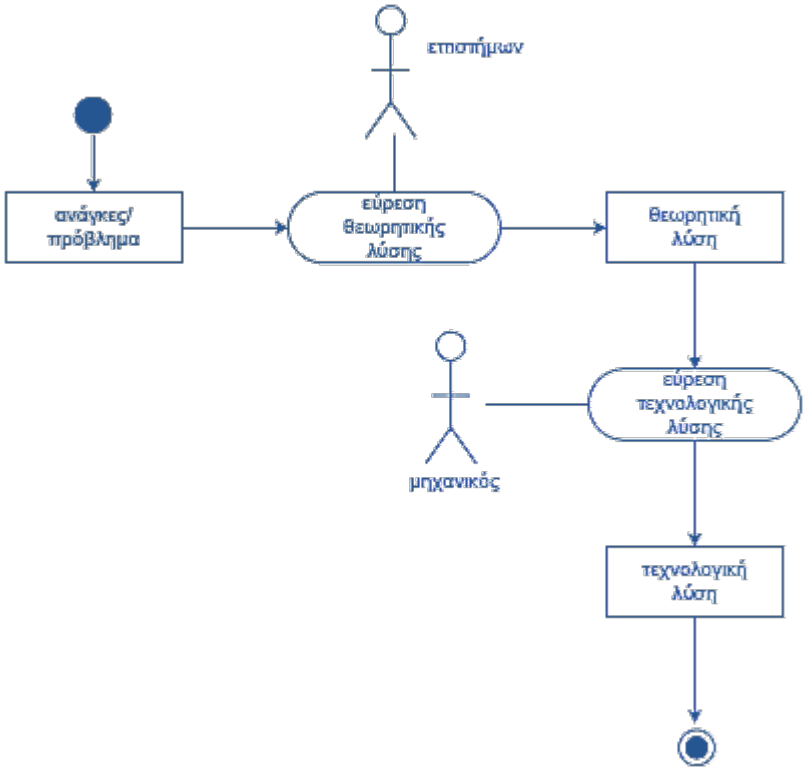
# ΠΕΡΙΕΧΟΜΕΝΑ

## ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

# ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ

# Εισαγωγή

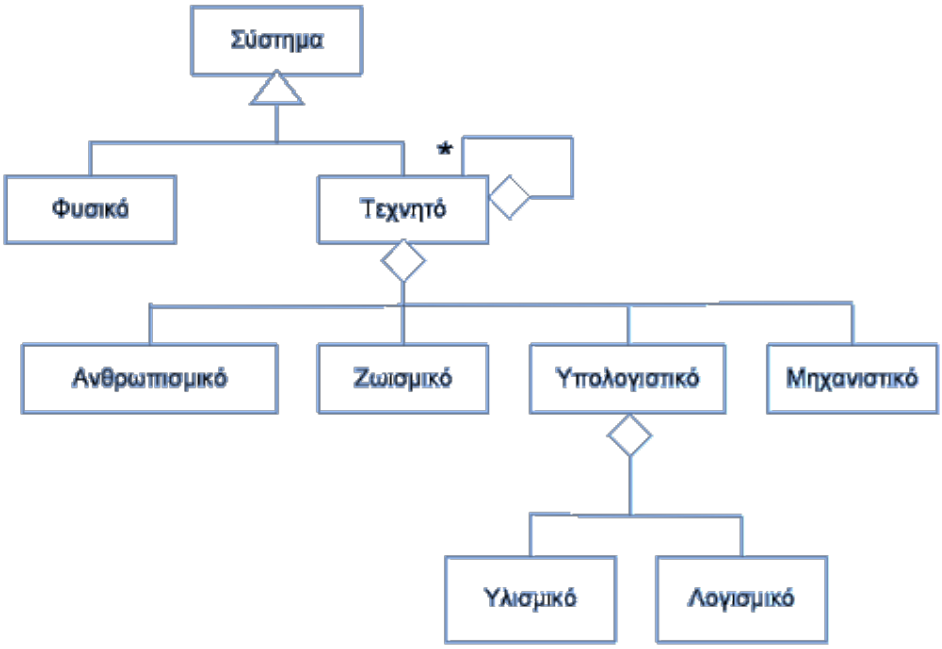
Όσο υπάρχουν άνθρωποι θα υπάρχουν και ανάγκες. Τα τεχνητά συστήματα καλύπτουν τις περισσότερες από αυτές τις ανάγκες αρκεί οι άνθρωποι να έχουν τη δυνατότητα να τα κατασκευάζουν, να τα συντηρούν και να τα λειτουργούν. Η αναγκαιότητα, λοιπόν, των τεχνητών συστημάτων είναι δεδομένη και συνεπώς η επιβίωση και η ευημερία του ανθρώπου εξαρτάται σε μεγάλο βαθμό από αυτά. (Εμμανουήλ Σκορδαλάκης, 2007)




**ΔΙΑΓΡΑΜΜΑ 1 Από την Ανάγκη/Πρόβλημα στη Λύση (Εμμανουήλ Σκορδαλάκης, 2007)**

Τα σημερινά τεχνητά συστήματα έχουν απαραιτήτως υποσυστήματα τα οποία είναι λογισμικά. Τα λογισμικά συστήματα είναι αυτά τα οποία οδήγησαν στην κατασκευή τελειότερων τεχνητών συστημάτων. Σε συνδυασμό, μάλιστα, με τα συστήματα επικοινωνιών έφεραν την τρίτη βιομηχανική επανάσταση, τον προηγμένο αυτοματισμό. (Εμμανουήλ Σκορδαλάκης, 2007)

Τα λογισμικά συστήματα είναι γλωσσολογικά προϊόντα που κατασκευάζονται από εξειδικευμένους, οι οποίοι χρειάζονται να ακολουθήσουν προς τούτο μια Μεθοδολογία. Η Μεθοδολογία καθοδηγεί στο πώς να χρησιμοποιούν την υπάρχουσα Λογισμική Τεχνολογία για να κατασκευάζουν λογισμικά συστήματα υψηλής ποιότητας εντός προβλέψιμης χρονικής περιόδου κατασκευής καθώς και προβλέψιμου κόστους. Αυτές οι μεθοδολογίες ανάπτυξης λογισμικών συστημάτων εμπεριέχονται στο αντικείμενο της Λογισμικής Μηχανικής, ενός νέου σχετικά τεχνολογικού κλάδου που δημιουργήθηκε προς αυτό το σκοπό. (Εμμανουήλ Σκορδαλάκης, 2007)




**ΔΙΑΓΡΑΜΜΑ 2 Κατηγοριοποίηση των συστημάτων (Εμμανουήλ Σκορδαλάκης, 2007)**

Τα πρακτικά προβλήματα αναλύονται σε υποπροβλήματα, μια κατηγορία εκ των οποίων έχει λύσεις που στην εκτέλεσή τους απαιτούνται δεδομένα και προκύπτουν ως αποτελέσματα πάλι νέα δεδομένα. Τα υποπροβλήματα αυτής της κατηγορίας ονομάζονται δεδομενικά προβλήματα, οι λύσεις και οι εργασίες, που αντιστοιχούν στην εκτέλεσή αυτών των λύσεων, ονομάζονται δεδομενικές λύσεις και δεδομενικές εργασίες. Έτσι, τα δεδομενικά προβλήματα είναι μια ειδική κατηγορία των πρακτικών προβλημάτων. Είναι αυτά που η εκτέλεση της λύσης τους απαιτεί δεδομένα, αλλά παράγει και δεδομένα. (Εμμανουήλ Σκορδαλάκης, 2007)

Η λογισμική μηχανική έχει ως αντικείμενο την υλοποίηση λογισμικών συστημάτων (software systems), τα οποία εκτελούν τις δεδομενικές εργασίες με τη βοήθεια των

ηλεκτρονικών υπολογιστών. Σε αυτά τα συστήματα το κυριότερο στοιχείο είναι ο κώδικας. Ο κώδικας περιγράφει στον Η-Υ, που παίζει το ρόλο ενός εργάτη, πως να εκτελεί τις δεδομενικές εργασίες. Ο κώδικας αυτός ετοιμάζεται χωριστά για κάθε δεδομενική εργασία που την εκτέλεσή της επιθυμούμε να αυτοματοποιήσουμε.

Για τις δεδομενικές εργασίες η πρόοδος σε βάθος χρόνου ήταν πολύ αργή, γιατί αποδείχθηκε εξαιρετικά δύσκολη η κατασκευή μηχανών κατάλληλων γι' αυτές τις εργασίες. Με την εφεύρεση του Ηλεκτρονικού Υπολογιστή (Η-Υ) στη δεκαετία του 1950, άνοιξε επιτέλους ο δρόμος για την τόσο επιθυμητή αυτοματοποίηση της εκτέλεσης των δεδομενικών εργασιών, αφού οι εργασίες αυτές είναι ανιαρές, μονότονες, κουραστικές και επιρρεπείς σε λάθη. (Εμμανουήλ Σκορδαλάκης, 2007)

Τα στοιχεία κάθε δεδομενικής εργασίας είναι δύο: λειτουργίες (operations) και δεδομένα (data). Οι λειτουργίες επεξεργάζονται τα δεδομένα και δίνουν ως αποτέλεσμα πάλι δεδομένα. Ανάμεσα στις απλές λειτουργίες διακρίνονται εκείνες που χρειάζονται για επεξεργασία (processing), διαχείριση των δεδομένων (data management), διάδραση (interaction) και καθορισμό της σειράς εκτέλεσης/ συγχρονισμού των λειτουργιών (sequencing/synchronizationn). (Εμμανουήλ Σκορδαλάκης, 2007)

# Κεφάλαιο 1

## 1. Ορισμοί: Από την πληροφορική στην ανάπτυξη πληροφοριακών συστημάτων

### 1.1. Πληροφορική (Computer Science)

Πληροφορική ή επιστήμη των υπολογιστών (Computer Science) ονομάζεται η θετική και εφαρμοσμένη επιστήμη η οποία ερευνά τα θεωρητικά θεμέλια και τη φύση των πληροφοριών, των αλγορίθμων και των υπολογισμών, καθώς και τις τεχνολογικές εφαρμογές τους σε αυτοματοποιημένα υπολογιστικά συστήματα, από τη σκοπιά της σχεδίασης, της ανάπτυξης, της υλοποίησης, της διερεύνησης, της ανάλυσης και της προδιαγραφής τους. ("The Philosophy of Computer Science (Stanford Encyclopedia of Philosophy)," n.d.)

### 1.2. Πληροφοριακά Συστήματα (Information Systems)

Τα πληροφοριακά συστήματα (Information Systems ή IS) είναι ένα σύνολο διαδικασιών, ανθρώπινου δυναμικού και αυτοματοποιημένων υπολογιστικών συστημάτων, που προορίζονται για τη συλλογή, εγγραφή, ανάκτηση, επεξεργασία, αποθήκευση και ανάλυση πληροφοριών. Τα συστήματα αυτά μπορούν να περιλαμβάνουν λογισμικό και τηλεπικοινωνιακό υλικό. (Jessup, 2008)

Τα πληροφοριακά συστήματα αποτελούν το μέσο για την αρμονική συνεργασία ανθρώπινου δυναμικού, δεδομένων, διαδικασιών και τεχνολογιών πληροφορίας και επικοινωνιών. Προέκυψαν ως γέφυρα μεταξύ των πρακτικών εφαρμογών της επιστήμης υπολογιστών και του επιχειρηματικού κόσμου.

Κάθε ειδικό πληροφοριακό σύστημα έχει ως στόχο την υποστήριξη οργανισμών και επιχειρήσεων, στον τομέα της διαχείρισης και της λήψης αποφάσεων. Με την ευρεία έννοια, ο όρος χρησιμοποιείται για να αναφερθεί όχι μόνο στην τεχνολογία της πληροφορίας και της επικοινωνίας (ΤΠΕ), που ένας οργανισμός ή επιχείρηση

χρησιμοποιεί, αλλά και στον τρόπο με τον οποίο οι άνθρωποι αλληλεπιδρούν με αυτή την τεχνολογία για την υποστήριξη των επιχειρηματικών διαδικασιών. (Valacich, 2014)

Ως εκ τούτου, τα πληροφοριακά συστήματα σχετίζονται αφενός με τα συστήματα διαχείρισης βάσης δεδομένων και αφετέρου με τα συστήματα δραστηριοτήτων και εφαρμογών.

## 1.3. Λογισμικό

Το λογισμικό είναι ένας γενικός όρος για οργανωμένες συλλογές από προγράμματα υπολογιστών, διαδικασίες και οδηγίες που εκτελούν ορισμένες εργασίες σε ένα υπολογιστικό σύστημα. Συχνά χωρίζεται σε δύο μεγάλες κατηγορίες: το λογισμικό συστήματος, που παρέχει τις βασικές γενικές λειτουργίες του υπολογιστή και το λογισμικό εφαρμογών που χρησιμοποιείται από τους χρήστες για να διεκπεραιώσει συγκεκριμένες εργασίες. (“software - WordReference.com Dictionary of English,” n.d.)

Το λογισμικό συστήματος είναι υπεύθυνο για τον έλεγχο, την ενσωμάτωση και διαχείριση των επιμέρους στοιχείων του υλικού του υπολογιστικού συστήματος, έτσι ώστε άλλα λογισμικά και οι χρήστες του συστήματος να συνεργάζονται σε αυτό ως μια λειτουργική μονάδα, χωρίς να χρειάζεται να ασχολούνται με τις χαμηλού επιπέδου λεπτομέρειες, όπως η μεταφορά δεδομένων από τη μνήμη στο δίσκο ή την εμφάνιση κειμένου σε μια οθόνη. Σε γενικές γραμμές, το λογισμικό συστήματος αποτελείται από ένα λειτουργικό σύστημα και ορισμένα θεμελιώδη βοηθητικά προγράμματα, όπως διαμορφωτές δίσκου, διαχειριστές αρχείων, διαχειριστές οθόνης, επεξεργαστές κειμένου, ταυτοποίηση χρήστη (login) και διάφορα άλλα εργαλεία διαχείρισης, καθώς και τη δικτύωση και τον λογισμικό έλεγχο συσκευών.

Το λογισμικό εφαρμογών, από την άλλη πλευρά, χρησιμοποιείται για την εκτέλεση ειδικών καθηκόντων, εκτός ακριβώς από τη λειτουργία του συστήματος του υπολογιστή. Μερικά παραδείγματα εφαρμογών λογισμικού περιλαμβάνουν:

- ένα ενιαίο πρόγραμμα, όπως ένα πρόγραμμα προβολής εικόνων,
- μια μικρή συλλογή των προγραμμάτων (που συχνά αποκαλείται ένα πακέτο λογισμικού) που συνεργάζονται στενά για να ολοκληρώσουν μια εργασία, όπως ένα σύστημα υπολογιστικών φύλλων ή επεξεργασίας κειμένου,
- μια μεγαλύτερη συλλογή (που συχνά αποκαλείται μια σουίτα λογισμικού), στην οποία σχετίζονται αλλά ανεξάρτητα προγράμματα και πακέτα, που έχουν μια κοινή διεπαφή χρήστη ή από κοινού χρήση των δεδομένων, όπως το Microsoft Office, το οποίο αποτελείται από ενσωματωμένο επεξεργαστή κειμένου, λογιστικά φύλλα, βάσεις δεδομένων, κλπ .,

- ένα σύστημα λογισμικού, όπως ένα σύστημα διαχείρισης βάσεων δεδομένων, η οποία είναι μια συλλογή από βασικά προγράμματα που μπορούν να παρέχουν κάποια υπηρεσία σε μια ποικιλία άλλων ανεξάρτητων εφαρμογών.

Το λογισμικό εφαρμογών (Application Software), αποτελείται από προγράμματα που έχουν σχεδιαστεί προκειμένου να βοηθήσουν τους χρήστες στην ολοκλήρωση των εργασιών τους, κατά τρόπο ταχύτερο, ευκολότερο και περισσότερο αποδοτικό. Το λογισμικό εφαρμογών μπορεί να διακριθεί στις εξής κατηγορίες:

- το τυποποιημένο λογισμικό (packed software), που κυκλοφορεί έτοιμο στο εμπόριο και καλύπτει το μεγαλύτερο ποσοστό αναγκών της αγοράς, και
- το κατά παραγγελία λογισμικό (custom software), που αναπτύσσεται κατόπιν παραγγελίας για μεμονωμένες εφαρμογές, με εξειδικευμένες απαιτήσεις.

  Το λογισμικό που χρησιμοποιείται στους μικροϋπολογιστές ανήκει ως επί το πλείστον στην πρώτη κατηγορία, ενώ για τα μεγάλα συστήματα απαιτείται συνήθως λογισμικό της δεύτερης κατηγορίας. (Cowan, 2012)

Το λογισμικό αναπτύσσεται με τις γλώσσες προγραμματισμού και των συναφών βοηθητικών εργαλείων, όπως: ενιαία προγράμματα, μεταφραστές δέσμης ενεργειών (script interpreters), πακέτα που περιέχουν ένα μεταγλωττιστή (compiler), συνδέσμους (linker) και άλλα εργαλεία, επίσης μεγάλες σουίτες (που συχνά αποκαλούνται Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης (IDE - Integrated Development Environments) που περιλαμβάνουν συντάκτες (editors), εργαλεία εντοπισμού σφαλμάτων (debuggers), καθώς και πλήθος άλλων εργαλείων για πολλές γλώσσες. (“Computer Software Definition,” n.d.)

## 1.4. Λογισμική Μηχανική - Software Engineering

### 1.4.1. Ορισμός

Μηχανική λογισμικού ή Λογισμική Μηχανική ή τεχνολογία λογισμικού (software engineering), ονομάζεται η τυποποιημένη και συστηματική προσέγγιση στην ανάλυση, σχεδίαση, υλοποίηση και συντήρηση λογισμικού ηλεκτρονικών υπολογιστικών συστημάτων.

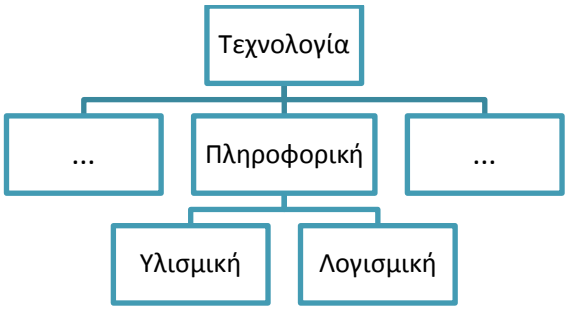
Η τεχνολογία λογισμικού είναι η μελέτη και η εφαρμογή της μηχανικής για το σχεδιασμό, την ανάπτυξη και συντήρηση του λογισμικού. (Bourque et al., 2014)

Τυπικοί ορισμοί της μηχανικής λογισμικού είναι:

η εφαρμογή μιας συστηματικής, πειθαρχημένης, μετρήσιμης προσέγγισης για την ανάπτυξη, λειτουργία και συντήρηση του λογισμικού, δηλαδή η εφαρμογή της μηχανικής στο λογισμικό. (Bourque et al., 2014), [ISO / IEC / IEEE Λεξιλόγιο Συστημάτων και Τεχνολογίας Λογισμικού (SEVOCAB)], και

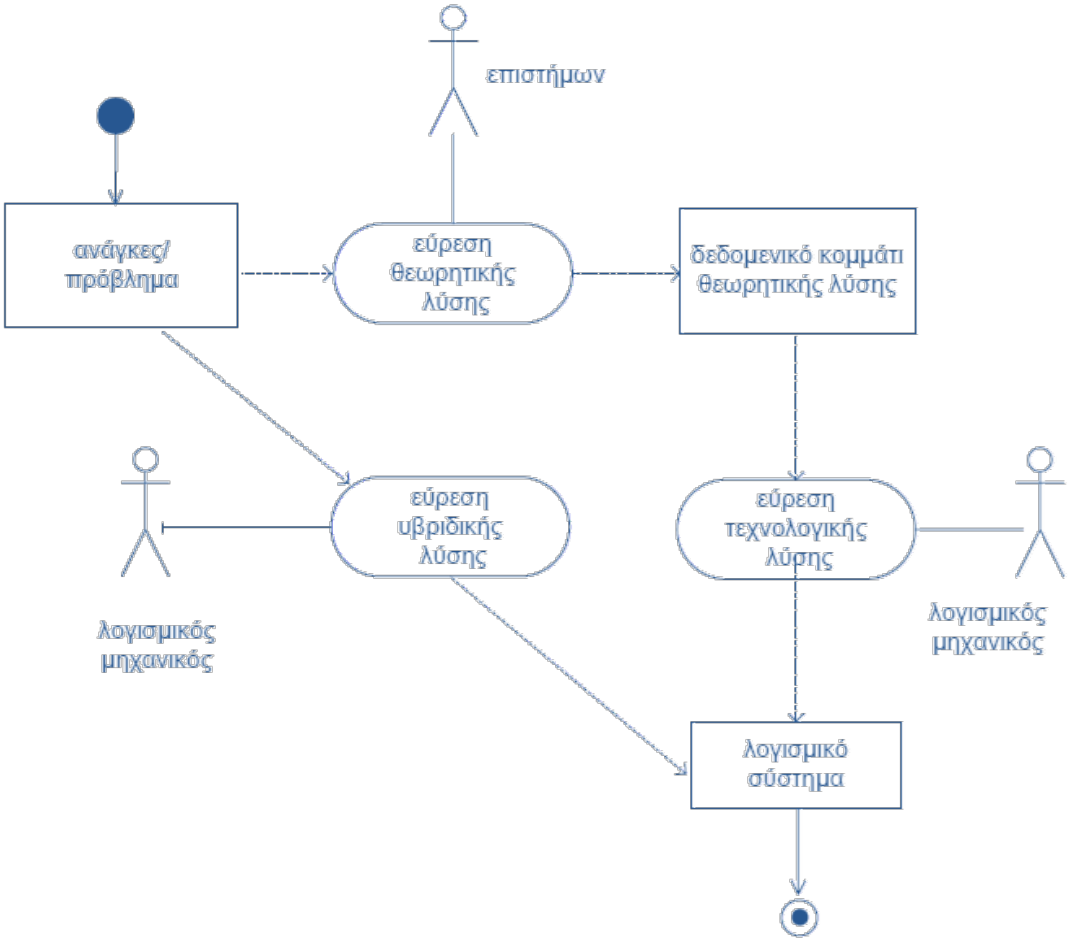
η δημιουργία και η χρήση των αρχών της χρηστής μηχανικής προκειμένου να παραχθεί με οικονομικό τρόπο λογισμικό που να είναι αξιόπιστο και να λειτουργεί αποτελεσματικά σε πραγματικές υπολογιστικές μηχανές. (Sommerville, 2007)

Η Λογισμική Τεχνολογία είναι αλληλένδετη με την Υλισμική Τεχνολογία (Hardware Technology) και οι δυο μαζί συνιστούν την Πληροφορική Τεχνολογία (Information Technology) που είναι εκείνη η οποία χαρακτηρίζει τη σημερινή εποχή. (Εμμανουήλ Σκορδαλάκης, 2007)




**ΔΙΑΓΡΑΜΜΑ 3 Η Λογισμική Τεχνολογία και η θέση της σε μία κατηγοριοποίηση (Εμμανουήλ Σκορδαλάκης, 2007)**

Η κατασκευή των λογισμικών συστημάτων γίνεται σε διάφορα στάδια και η εργασία που απαιτείται γι' αυτό ονομάζεται Λογισμική Εργασία (Software Process), οι δε μηχανικοί που την υλοποιούν ονομάζονται Λογισμικοί Μηχανικοί (Software Engineers). Ο αντίστοιχος τεχνολογικός κλάδος είναι η Λογισμική Μηχανική (Software Engineering), που είναι ένας κατασκευαστικός κλάδος, ο οποίος διαφέρει από τους άλλους κλάδους αυτής της κατηγορίας, στο ότι σ' αυτόν το προϊόν της κατασκευής είναι γλωσσολογικό, δηλαδή αποτελείται από γλωσσολογικά στοιχεία συγκεκριμένων γλωσσών σε γραπτή μορφή (γλωσσολογικό υλικό), ενώ στους άλλους κλάδους το προϊόν αποτελείται και από φυσικά υλικά. (Εμμανουήλ Σκορδαλάκης, 2007)




**ΔΙΑΓΡΑΜΜΑ 4 Διαδρομές Κατασκευής Λογισμικών Συστημάτων (Εμμανουήλ Σκορδαλάκης, 2007)**

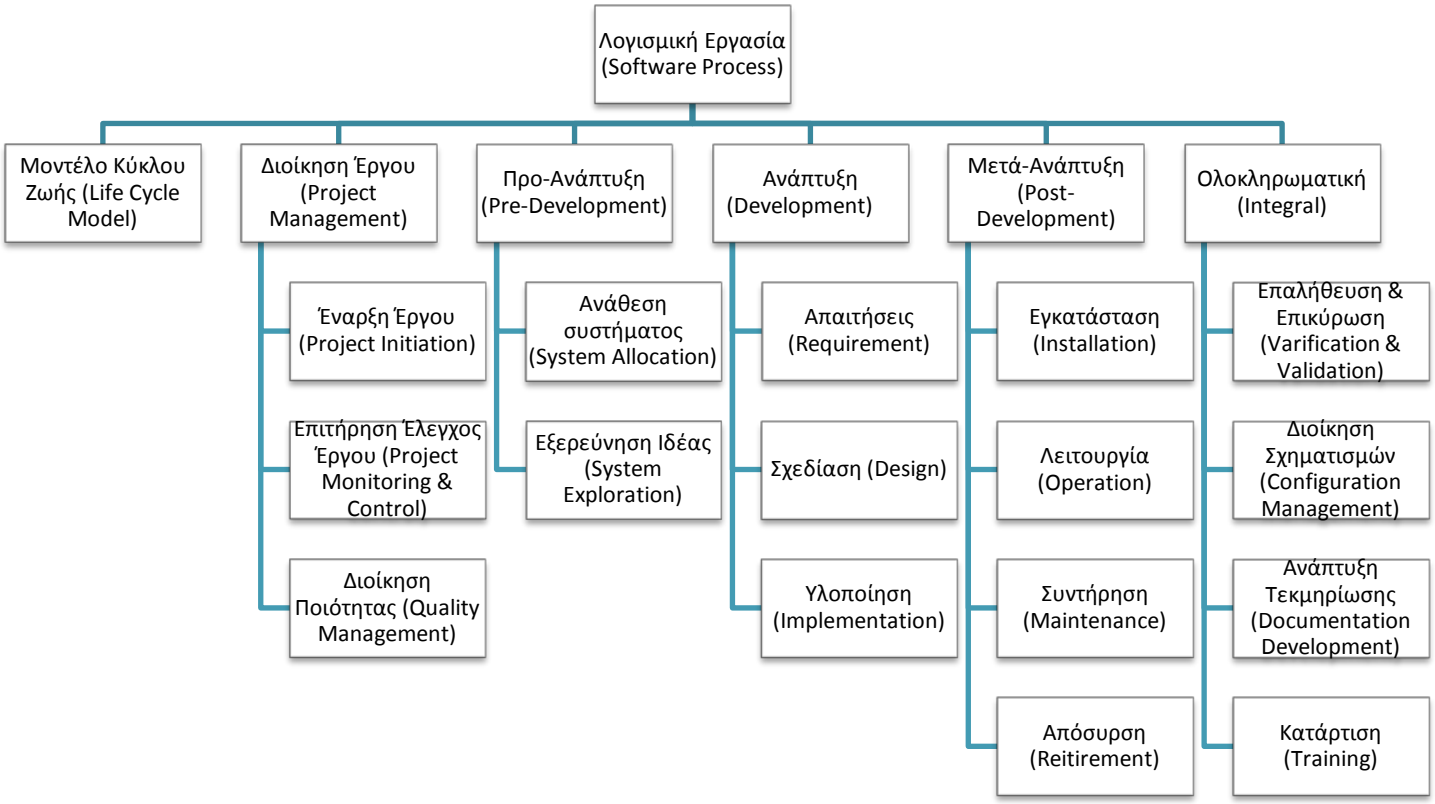
Η εργασία που γίνεται κατά την κατασκευή των λογισμικών συστημάτων ονομάζεται, όπως αναφέρθηκε παραπάνω, λογισμική εργασία (software process). Η λογισμική εργασία είναι μια σύνθετη εργασία που μπορεί να χωρισθεί σε άλλες απλούστερες, κάθε μία από αυτές με τη σειρά της σε άλλες απλούστερες, κοκ. Αυτή η κατάτμηση είναι φανερό πως μπορεί να γίνει με πολλούς τρόπους. Ακόμη, κάθε μία από αυτές τις εργασίες μπορεί να εκτελείται με διάφορους τρόπους. Αυτό οδηγεί σε μια μεγάλη ποικιλία λογισμικών εργασιών. Ουσιαστικά, το αντικείμενο της Λογισμικής Μηχανικής είναι η λογισμική εργασία, αφού τα λογισμικά συστήματα κατασκευάζονται μέσω αυτής. Η Λογισμική Μηχανική (Software Engineering) είναι ένας τεχνολογικός κλάδος ο οποίος χρησιμοποιεί καταλλήλως την Λογισμική Τεχνολογία για την κατασκευή Λογισμικών Συστημάτων, τα οποία βέβαια γίνονται μέρος αυτής της Τεχνολογίας μετά την κατασκευή τους.

Ο όρος μεθοδολογία (methodology) χρησιμοποιείται συχνά αντί του όρου λογισμική εργασία. Η εφαρμογή μιας Μεθοδολογίας στην ανάπτυξη ενός Λογισμικού

Συστήματος οδηγεί σε μια Λογισμική Εργασία (Software Process). Δηλαδή, η Μεθοδολογία και η Λογισμική Εργασία είναι δύο όψεις του ίδιου νομίσματος. Η μια είναι η περιγραφή (Μεθοδολογία) και η άλλη είναι η εκτέλεση (Λογισμική Εργασία) του ίδιου πράγματος. (Εμμανουήλ Σκορδαλάκης, 2007)

Οι Μεθοδολογίες αυτές χρειάζονται για την κατασκευή των λογισμικών συστημάτων και αποτελούν το αντικείμενο της Λογισμικής Μηχανικής. Συγκεκριμένα, απαντούν στο ερώτημα πως πρέπει να χρησιμοποιούνται τα στοιχεία της Βασικής Λογισμικής Τεχνολογίας για την κατασκευή λογισμικών συστημάτων που να είναι λειτουργικά και καλής ποιότητας και ακόμη να είναι προβλέψιμο το απαιτούμενο κόστος και προβλέψιμη η απαιτούμενη χρονική περίοδος της κατασκευής τους.

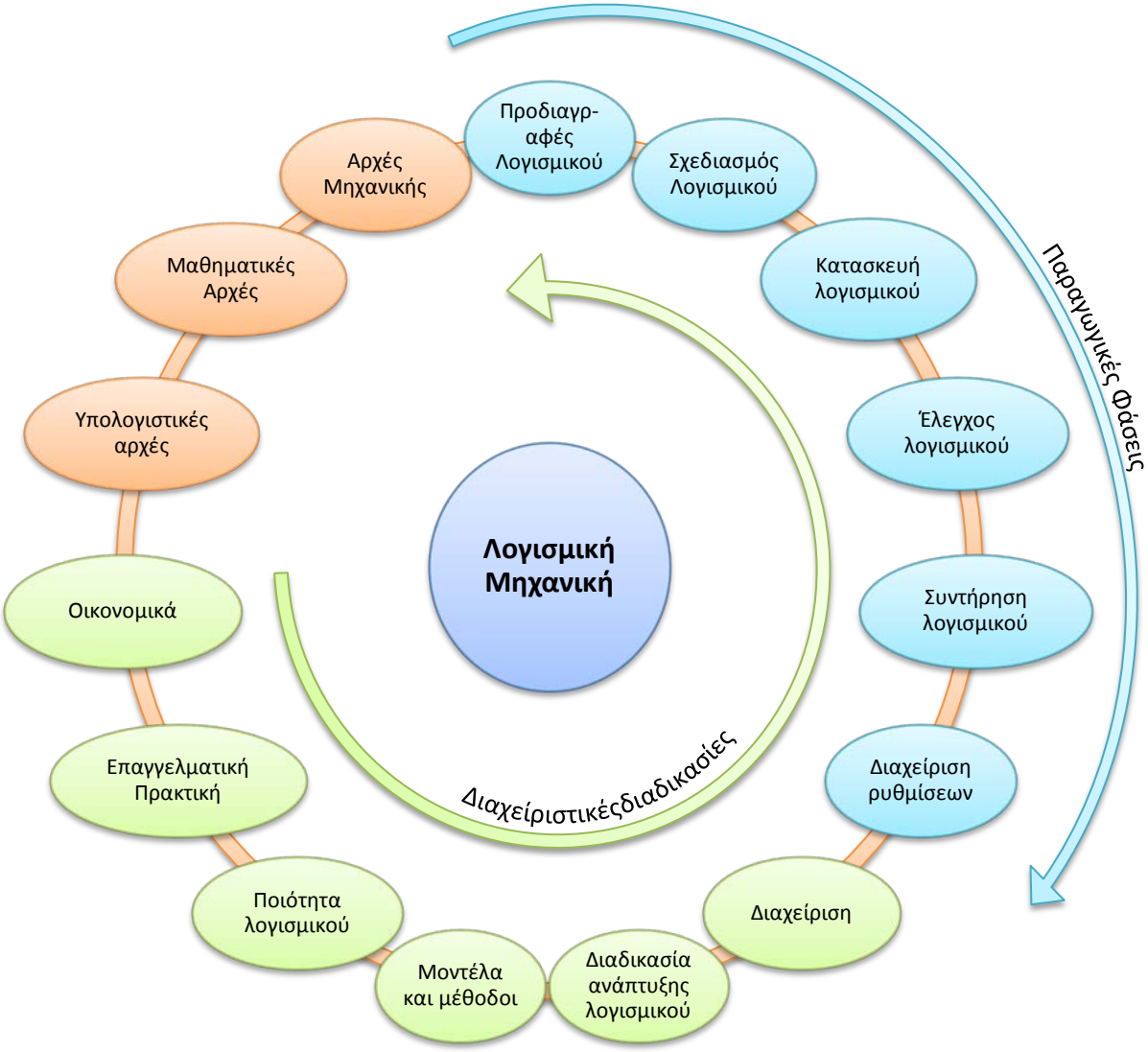
Αντικείμενο, λοιπόν, της Λογισμικής Μηχανικής είναι η επινόηση κατάλληλων Μεθοδολογιών, αφού αποδείχτηκε ότι μια και μοναδική Μεθοδολογία δεν είναι αρκετή για την κατασκευή όλων των ειδών των Λογισμικών Συστημάτων, τη συντήρησή τους καθώς και τη διάθεσή τους για να μπορούν να χρησιμοποιηθούν επιτυχώς στην πράξη.




**ΔΙΑΓΡΑΜΜΑ 5 Οι λογισμικές εργασίες κατά το IEEE (Εμμανουήλ Σκορδαλάκης, 2007), ("IEEE Standard For Developing Software Life Cycle Processes - IEEE Std 1074-1997 - IEEE1074.pdf," n.d.)**

### 1.4.2. Επιμέρους Κλάδοι

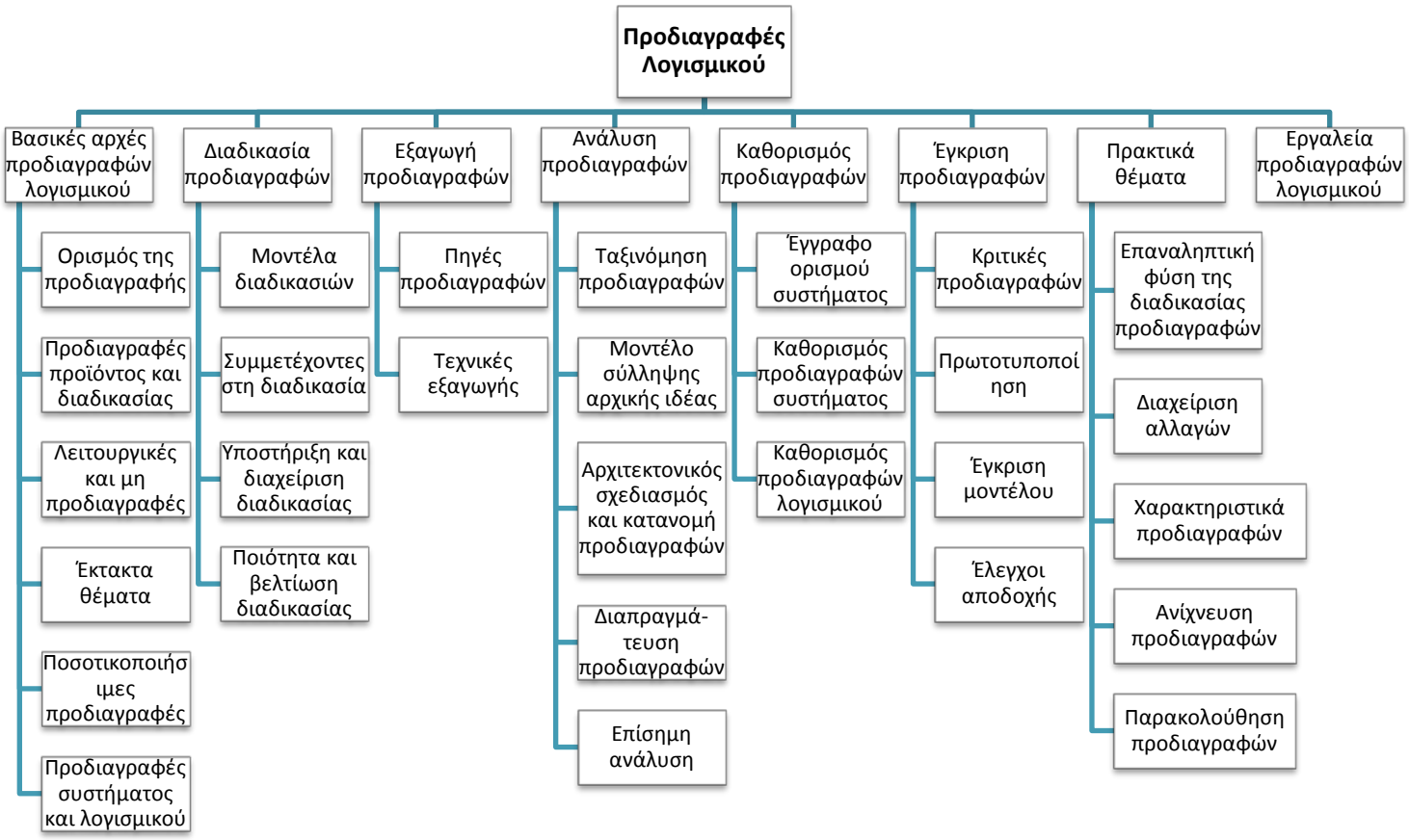
Σύμφωνα με το SWEBOK υπάρχουν 15 θεματικές περιοχές που σχετίζονται με την τεχνολογία λογισμικού.




**ΔΙΑΓΡΑΜΜΑ 6 Οι 15 θεματικές ενότητες του SWEBOK (Bourque et al., 2014)**

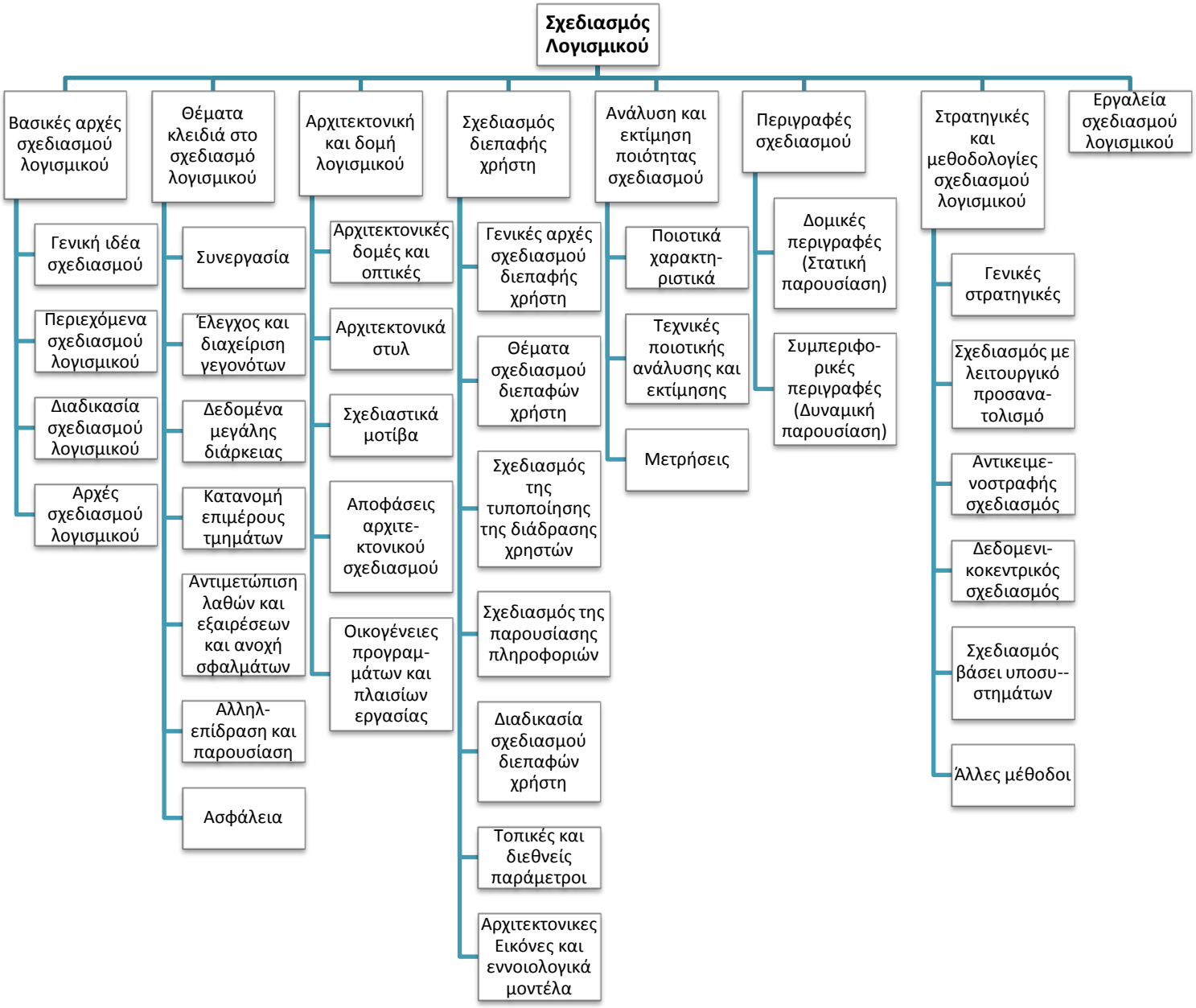
Από αυτές τις θεματικές περιοχές οι δέκα επιμέρους μπορεί να θεωρηθεί ότι εμπλέκονται αμεσότερα με το αντικείμενο της Λογισμικής Μηχανικής (Bourque et al., 2014), (Sommerville, 2007):

1.4.2.1. Καθορισμός Προδιαγραφών (Software Requirements): Η εξαγωγή, η ανάλυση, ο προσδιορισμός, και η επικύρωση των απαιτήσεων και προδιαγραφών για το λογισμικό.



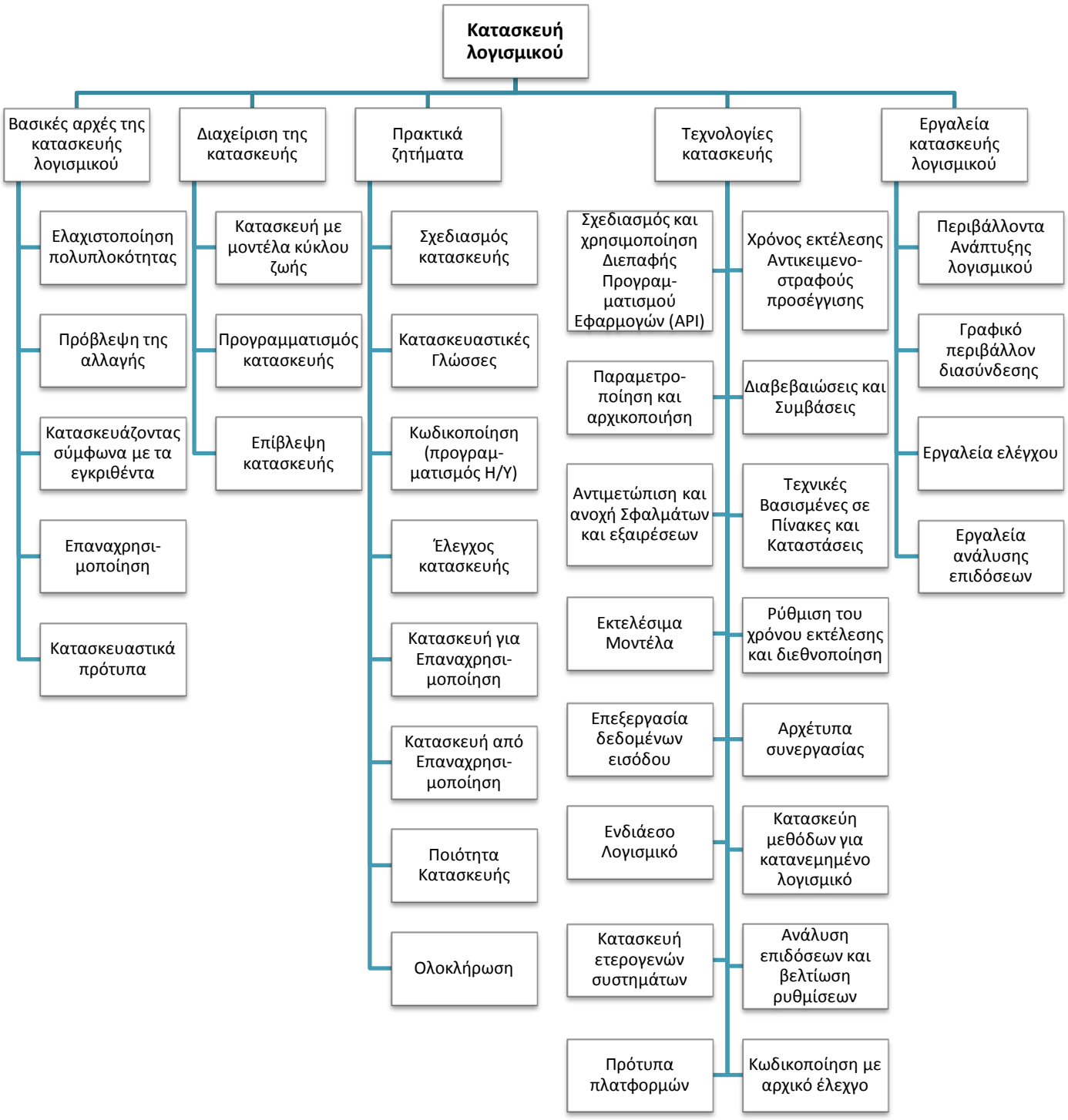

**ΔΙΑΓΡΑΜΜΑ 7 Δραστηριότητες Καθορισμού Προδιαγραφών (Bourque et al., 2014)**

1.4.2.2. Σχεδιασμός λογισμικού (Software Design): Η διαδικασία καθορισμού της αρχιτεκτονικής, των αντικειμένων, των διασυνδέσεων, και άλλων χαρακτηριστικών ενός συστήματος ή τμήματός του. Επίσης, ορίζεται ως και το αποτέλεσμα της διαδικασίας αυτής.



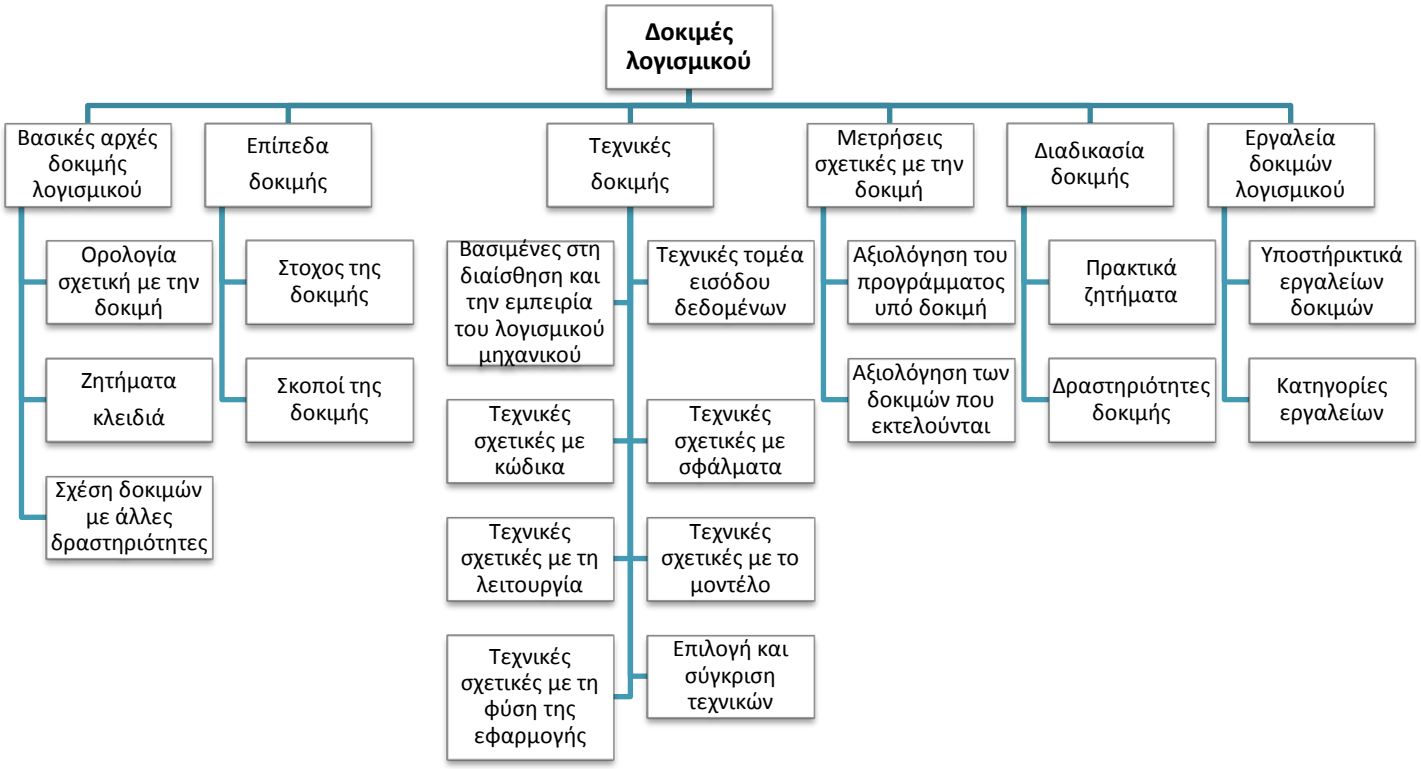

**ΔΙΑΓΡΑΜΜΑ 8 Δραστηριότητες σχεδιασμού λογισμικού (Bourque et al., 2014)**

1.4.2.3. Κατασκευή ή Υλοποίηση λογισμικού(Software Construction): Η λεπτομερής δημιουργία ενός λειτουργικού και αποτελεσματικού λογισμικού μέσω ενός συνδυασμού γραφής κώδικα, επαλήθευσης, ελέγχου, δοκιμών ολοκλήρωσης, καθώς και τον εντοπισμό σφαλμάτων.



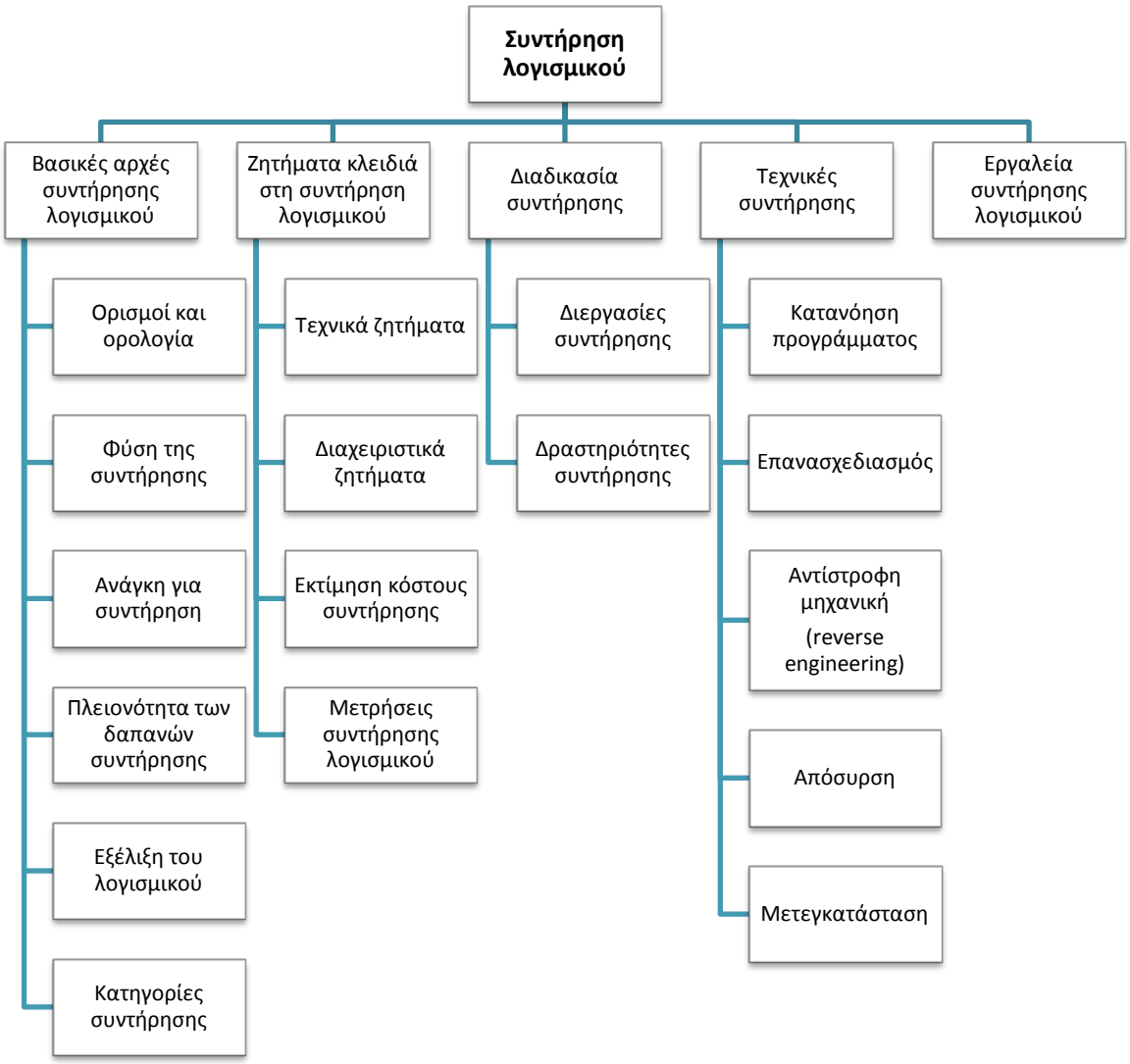

**ΔΙΑΓΡΑΜΜΑ 9 Δραστηριότητες κατασκευής λογισμικού (Bourque et al., 2014)**

1.4.2.4. Δοκιμή Λογισμικού (Software Testing): Η διεξαγωγή μιας εμπειρικής, τεχνικής έρευνας για την παροχή πληροφοριών σχετικά με την ποιότητα του προϊόντος ή της υπηρεσίας υπό δοκιμή.



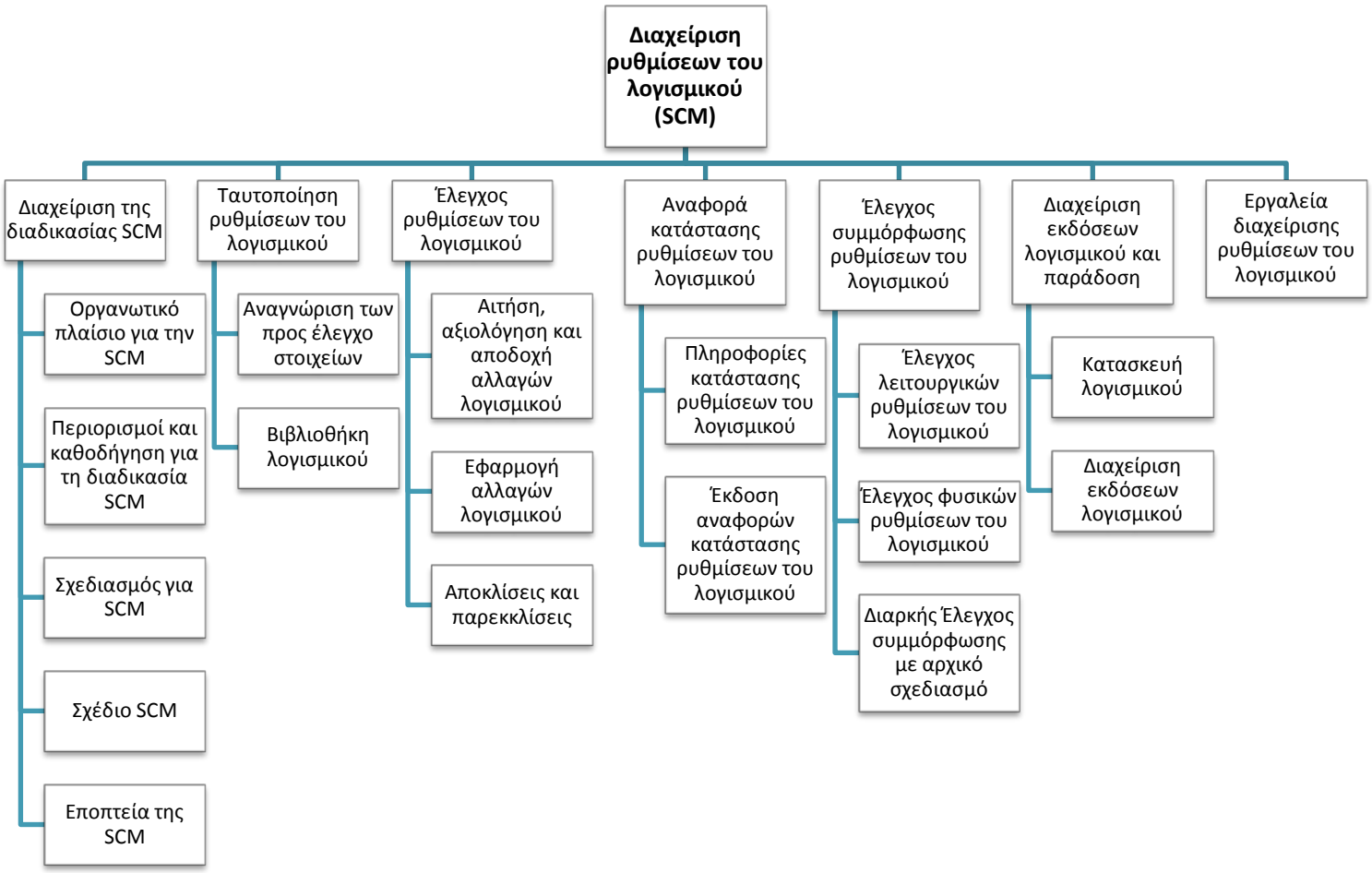

**ΔΙΑΓΡΑΜΜΑ 10 Δραστηριότητες Δοκιμών λογισμικού (Bourque et al., 2014)**

1.4.2.5. Συντήρηση λογισμικού (Software Maintenance): Το σύνολο των δραστηριοτήτων που απαιτούνται για την παροχή οικονομικά αποτελεσματικής υποστήριξης για το λογισμικό.



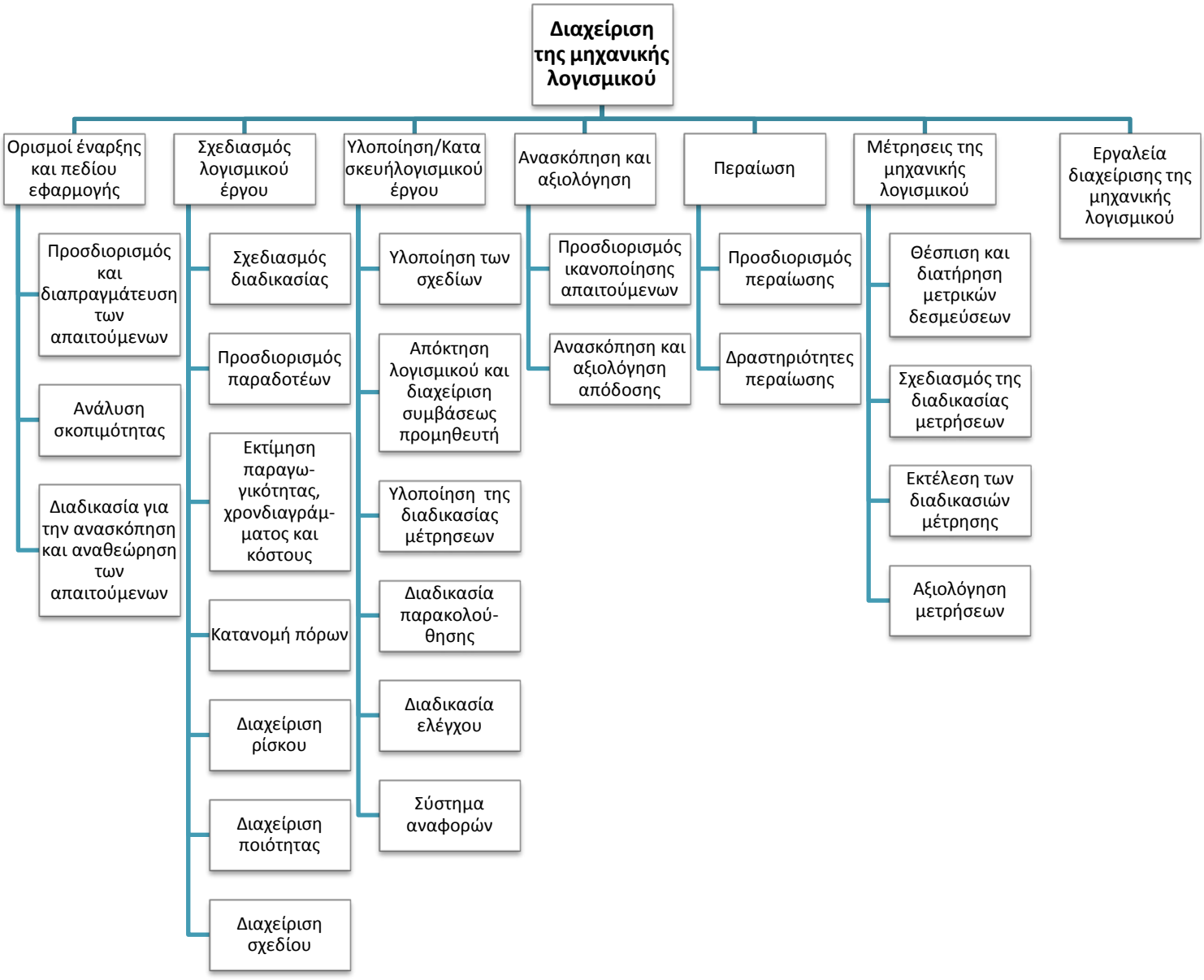

**ΔΙΑΓΡΑΜΜΑ 11 Δραστηριότητες συντήρησης λογισμικού (Bourque et al., 2014)**

1.4.2.6. Διαχείριση ρυθμίσεων του λογισμικού (Software Configuration Management): Η αναγνώριση των ρυθμίσεων ενός συστήματος σε διακριτά χρονικά σημεία με σκοπό τον συστηματικό έλεγχο των αλλαγών στις ρυθμίσεις και τη διατήρηση της ακεραιότητας και της ιχνηλασιμότητας των ρυθμίσεων καθ΄όλη τη διάρκεια του κύκλου ζωής του συστήματος.



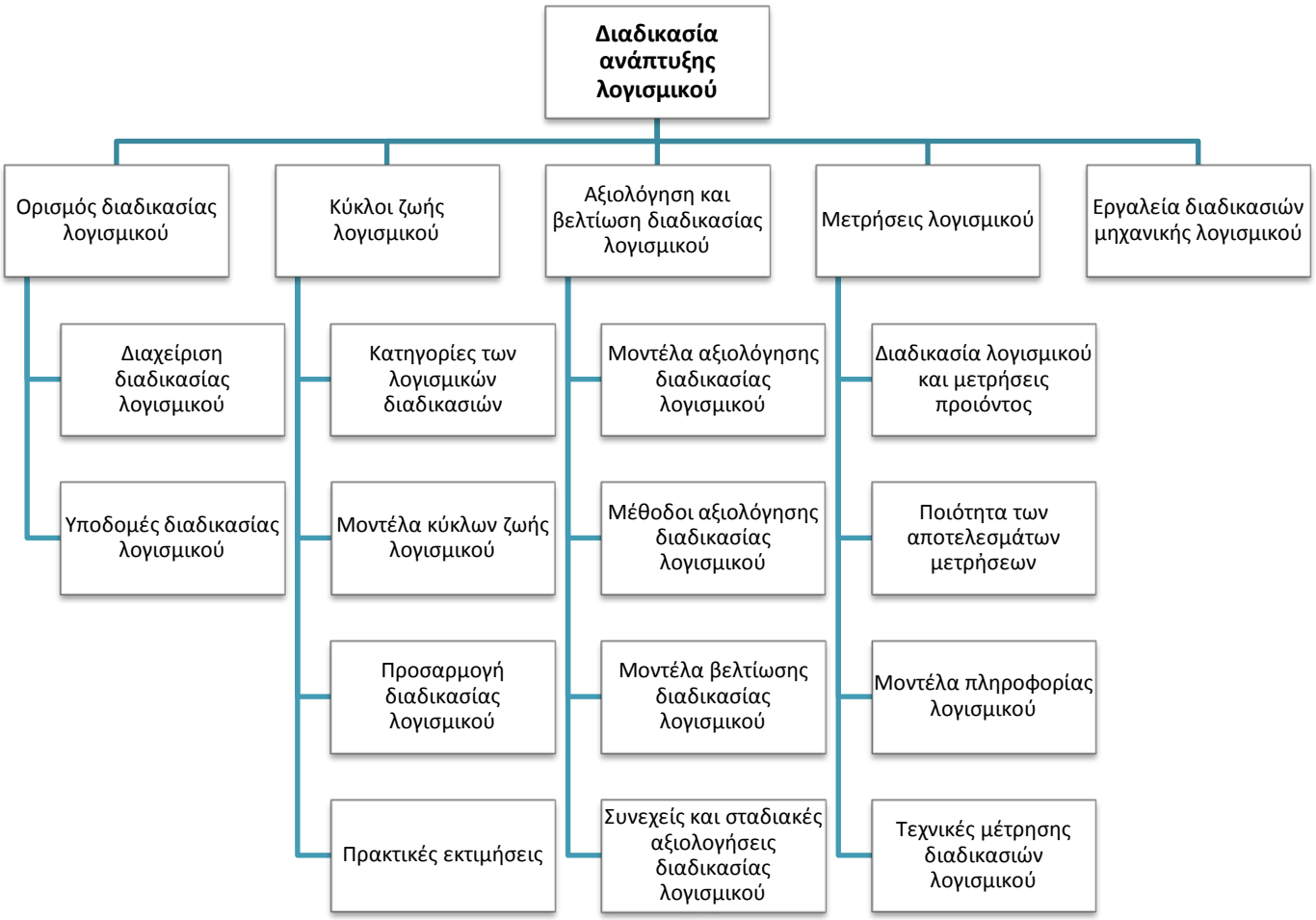

**ΔΙΑΓΡΑΜΜΑ 12 Δραστηριότητες διαχείρισης ρυθμίσεων λογισμικού (Bourque et al., 2014)**

1.4.2.7. Διαχείριση της Μηχανικής Λογισμικού (Software Engineering Management): Η εφαρμογή της διαχείρισης των δραστηριοτήτων, τον σχεδιασμό, τον συντονισμό, τη μέτρηση, την παρακολούθηση, τον έλεγχο και την υποβολή αναφορών, ώστε να εξασφαλιστεί ότι η ανάπτυξη και συντήρηση του λογισμικού γίνεται με τρόπο συστηματικό, πειθαρχημένο και ποσοτικοποιημένο.




**ΔΙΑΓΡΑΜΜΑ 13 Δραστηριότητες διαχείρισης μηχανικής λογισμικού (Bourque et al., 2014)**

1.4.2.8. Διαδικασία ανάπτυξης λογισμικού (Software Engineering Process): Ο καθορισμός, η εφαρμογή, η αξιολόγηση, η μέτρηση, η διαχείριση, η αλλαγή, και η βελτίωση της διαδικασίας του κύκλου ζωής του ίδιου του λογισμικού.

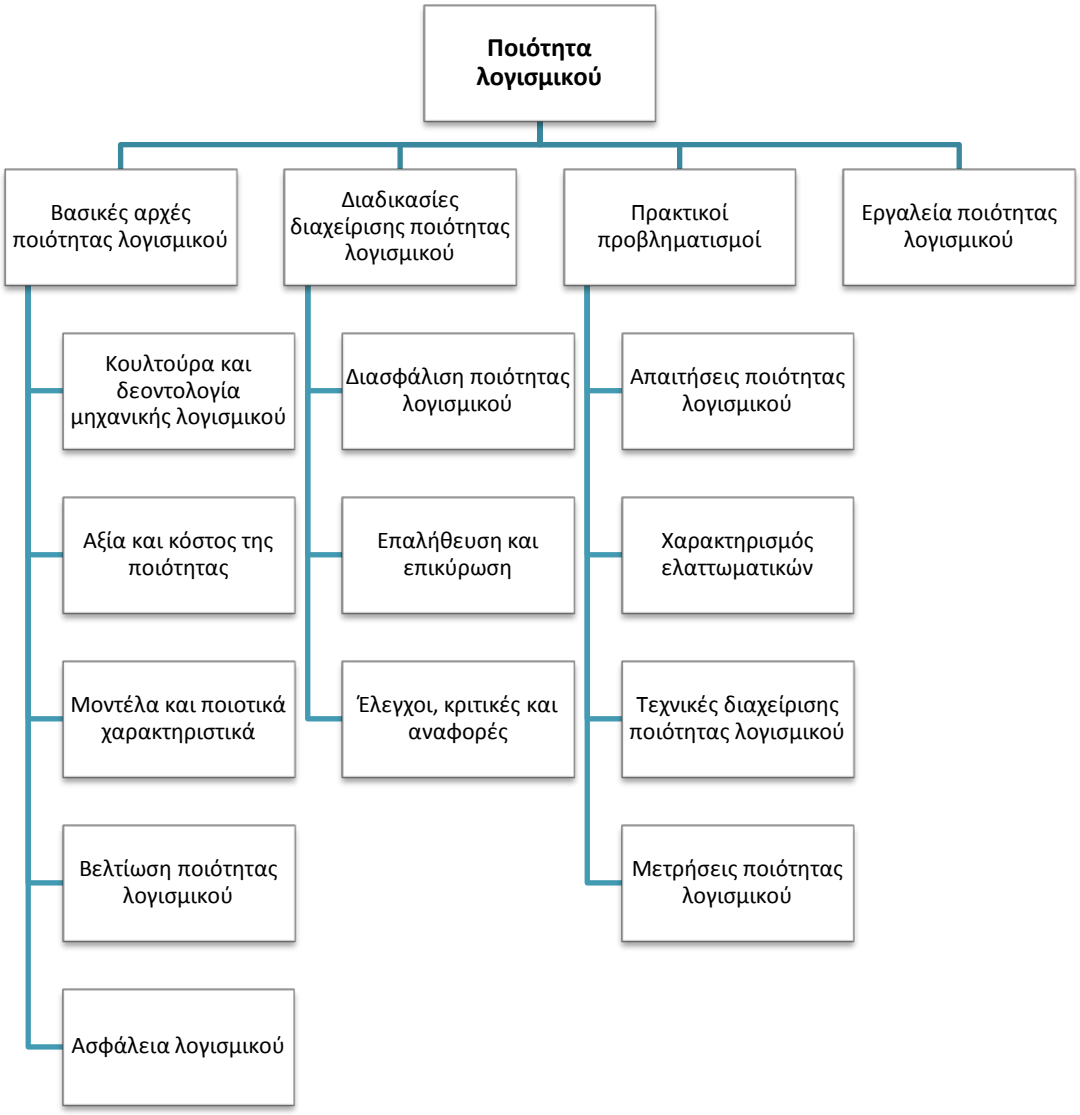



**ΔΙΑΓΡΑΜΜΑ 14 Δραστηριότητες διαχείρισης ανάπτυξης λογισμικού (Bourque et al., 2014)**

1.4.2.9. Μέθοδοι και μοντέλα μηχανικής λογισμικού (Software Engineering Models & Methods): Τα υπολογιστικά εργαλεία που προορίζονται να βοηθήσουν τη διαδικασία του κύκλου ζωής του λογισμικού και οι μέθοδοι που επιβάλλουν οργανωτική δομή σχετικά με τη δραστηριότητα της μηχανικής λογισμικού, με στόχο την συστηματοποίηση της όλης δραστηριότητας και κατ' επέκταση την επιτυχία της.


**ΔΙΑΓΡΑΜΜΑ 15 Μοντέλα και μέθοδοι μηχανικής λογισμικού (Bourque et al., 2014)**

1.4.2.10. Διαχείριση της ποιότητας του λογισμικού (Software Quality): Ο βαθμός στον οποίο ένα σύνολο εγγενών χαρακτηριστικών πληροί τις απαιτήσεις.




**ΔΙΑΓΡΑΜΜΑ 16 Δραστηριότητες διαχείρισης ποιότητας λογισμικού (Bourque et al., 2014)**

## 1.5. Ανάπτυξη πληροφοριακών συστημάτων

Η Ανάπτυξη πληροφοριακών συστημάτων είναι ένας τομέας, ο οποίος πραγματεύεται τεχνικές, μεθοδολογίες, πρακτικές και εργαλεία για την συστηματική, μεθοδική και ποσοτικοποιημένη προδιαγραφή, σχεδίαση, υλοποίηση, έλεγχο, και συντήρηση συστημάτων λογισμικού υψηλής ποιότητας και εντός δεδομένου προϋπολογισμού και χρόνου εκτέλεσης. [IEEE Standard 610.12]

Μια σειρά από μεθόδους και διαδικασίες μπορούν να χρησιμοποιηθούν για την ανάπτυξη και τη χρήση ενός συστήματος πληροφοριών. Πολλοί προγραμματιστές χρησιμοποιούν διαφορετικές προσεγγίσεις, μια περισσότερο διαδεδομένη είναι η μεθοδολογία του Κύκλου Ζωής της Ανάπτυξης Πληροφοριακού Συστήματος (SDLC), η οποία είναι μια συστηματική διαδικασία για την ανάπτυξη ενός συστήματος πληροφοριών μέσα από τα στάδια που εμφανίζονται στη σειρά. (Valacich, 2014)

Η Ανάπτυξη του συστήματος γίνεται επί τη βάσει του κύκλου ζωής του:

- Διερευνητική Μελέτη: αναγνώριση του προβλήματος και προδιαγραφές
- Μελέτη Σκοπιμότητας: συλλογή πληροφοριών
- Ανάλυση Απαιτήσεων: προδιαγραφές και απαιτήσεις για το νέο σύστημα
- Σχεδιασμός: σχεδιασμός του συστήματος
- Υλοποίηση: κατασκευή του συστήματος
- Εγκατάσταση: εφαρμογή του συστήματος
- Συντήρηση: αξιολόγηση και συντήρηση

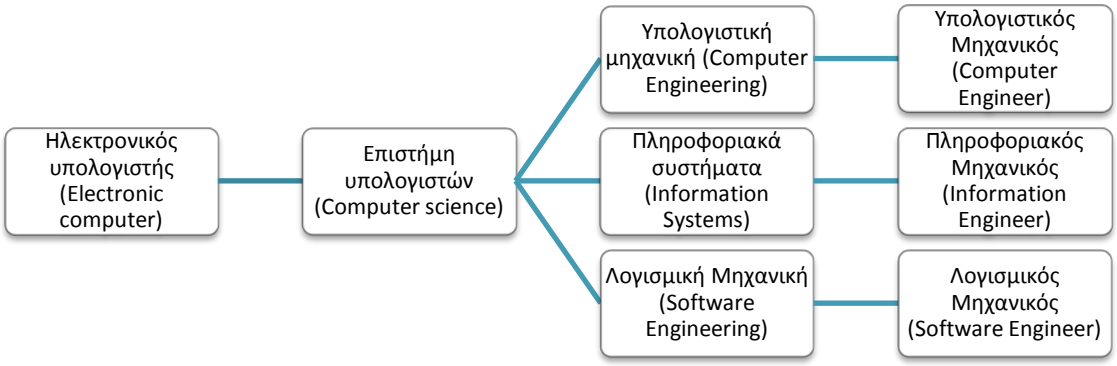
## 1.6. Ανάπτυξη πληροφοριακών συστημάτων και Λογισμική Μηχανική

Η Ανάπτυξη λογισμικού και η λογισμική μηχανική είναι αλληλένδετοι όροι, αλλά δεν σημαίνουν ακριβώς το ίδιο πράγμα. Ένας μηχανικός λογισμικού ασχολείται με την ανάπτυξη λογισμικού. Ωστόσο, όλοι οι προγραμματιστές λογισμικού, δεν είναι μηχανικοί. Ο όρος Λογισμική Μηχανική σημαίνει την εφαρμογή των αρχών της μηχανικής για τη δημιουργία λογισμικού. Μπορεί να φαίνεται παράξενο να μιλάμε για μηχανική για κάτι που δεν έχει μάζα ή δεν καταλαμβάνει χώρο, αλλά το λογισμικό είναι ενσωματωμένο σε αντικείμενα τα οποία έχουν υλική υπόσταση. Το λογισμικό κάνει τα πάντα, όπως από τη διανομή φαρμάκων έως και τον έλεγχο μεγάλου εξοπλισμού. Πολλοί άνθρωποι βασίζονται επίσης στο λογισμικό για την εκτέλεση των καθηκόντων της εργασίας τους, εργαζόμενοι είτε σε ένα γραφείο είτε συνδεόμενοι απομακρυσμένα. Όπως όλοι γνωρίζουμε, στις εφαρμογές λογισμικού μπορεί να προκληθεί δυσλειτουργία. Δεν είναι μόνο γέφυρες που «καταρρέουν» (crash) και δεν είναι μόνο οι γέφυρες που χρειάζονται γερά θεμέλια. Οι μηχανικοί λογισμικού

αρχίζουν με μια λεπτομερή μελέτη των απαιτήσεων. Εργάζονται καθ' όλη τη διαδικασία της ανάπτυξης με συστηματικό τρόπο και σε αυτό αναφέρεται ο όρος κύκλος ζωής της ανάπτυξης λογισμικού. «Τεχνολογία Λογισμικού σημαίνει την εφαρμογή των αρχών της μηχανικής για τη δημιουργία λογισμικού» ~ IEEE

Επειδή, λοιπόν, οι όροι αυτοί τόσο συχνά συγχέονται, είναι δύσκολος ο διαχωρισμός του μηχανικού λογισμικού και του ειδικευμένου σε μόνο ένα μέρος της διαδικασίας ανάπτυξης λογισμικού - για παράδειγμα της κωδικοποίησης. Σε ορισμένα κράτη είναι αυστηρός ο ορισμός για τον τίτλο του μηχανικού λογισμικού, αλλά στα περισσότερα δεν είναι. Ακόμα, σε πολλές εταιρείες γίνεται η επιλογή των τίτλων των θέσεων εργασίας σύμφωνα με το άκουσμά τους. ("Software Development Versus Software Engineering," n.d.)

Οι μηχανικοί λογισμικού επιδιώκουν να εφαρμόζουν τις αρχές της μηχανικής σε όλα τα στάδια της διαδικασίας ανάπτυξης λογισμικού, από την ανάλυση απαιτήσεων μέχρι την έκδοση και συντήρησή του. Οι μηχανικοί λογισμικού γνωρίζουν πολλά τόσο για τους υπολογιστές, όσο και για την ομαδική εργασία. Στις θέσεις έξω στον πραγματικό κόσμο, θα ειδικεύονται συχνά, εστιάζοντας σε ένα συγκεκριμένο στάδιο της ανάπτυξης, αλλά αναμένεται, ωστόσο, να είναι σε θέση να ανταπεξέλθουν σε οποιονδήποτε από πολλαπλούς ρόλους. Ένας προγραμματιστής, από την άλλη πλευρά, θα μάθει πρωτίστως την κωδικοποίηση υπολογιστή, που αποτελεί ένα μέρος του κύκλου ζωής της ανάπτυξης λογισμικού. ("Terminology - What's the difference between programmer and software engineer? - Stack Overflow," n.d.), ("Programmer, Developer, Engineer: What's in a name?," n.d.)




**ΔΙΑΓΡΑΜΜΑ 17 Επιστήμη υπολογιστών (Εμμανουήλ Σκορδαλάκης, 2007)**

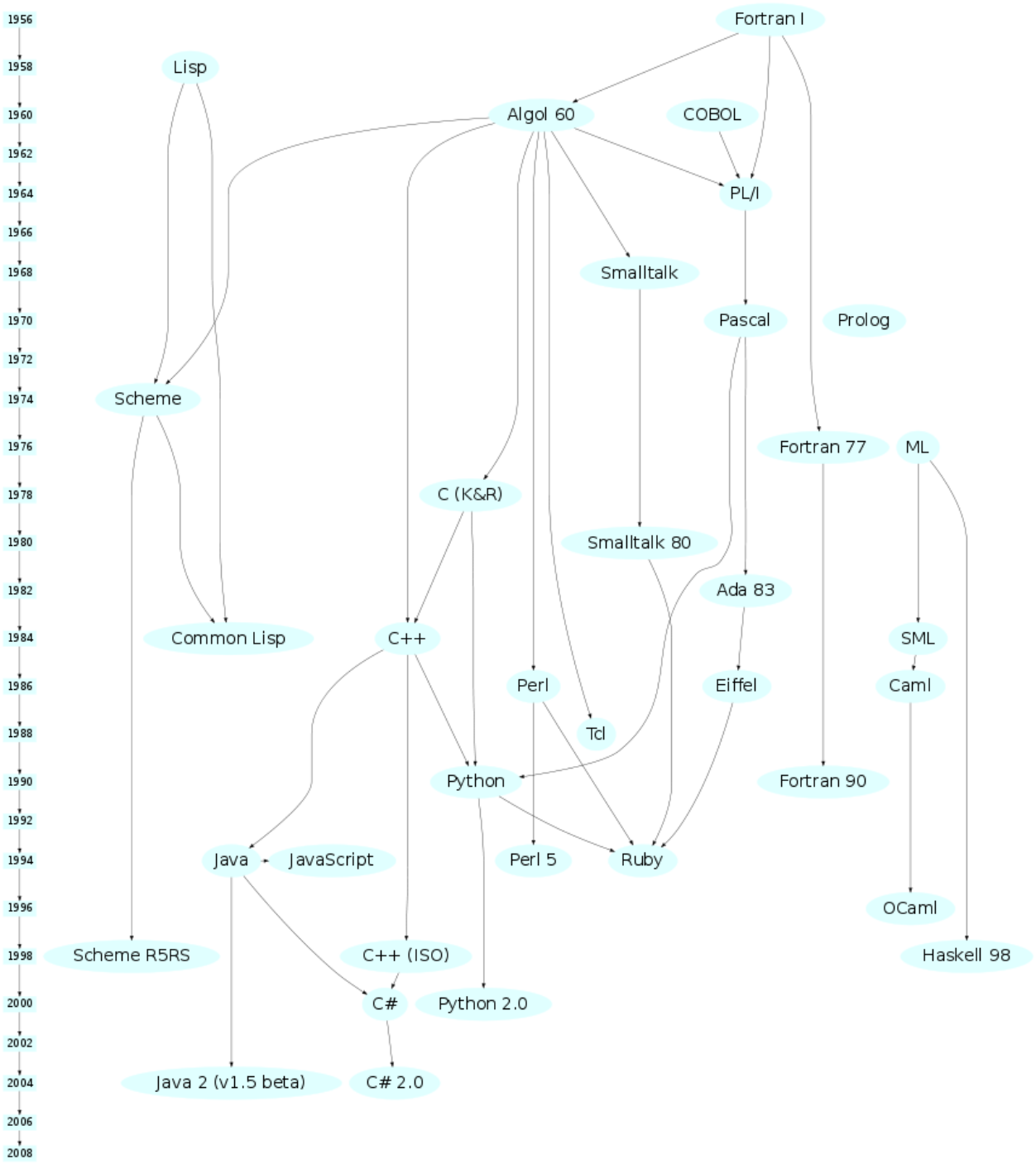
## 1.7. Γλώσσες Προγραμματισμού (ΓΠ)

Οι Γλώσσες Προγραμματισμού παρέχουν την (πρώτη) ύλη για την κατασκευή των λογισμικών συστημάτων, δηλαδή τον κώδικα από τον οποίο αυτά αποτελούνται. Ο κώδικας αυτός οργανώνεται σε ενότητες στις οποίες αναφερόμαστε με τον όρο προγράμματα. Τα προγράμματα είναι περιγραφές σε μια ΓΠ δεδομενικών εργασιών που την εκτέλεσή τους θέλουμε να αυτοματοποιήσουμε με Η-Υ. (Εμμανουήλ Σκορδαλάκης, 2007)

Οι γλώσσες προγραμματισμού (όπως άλλωστε και οι ανθρώπινες γλώσσες) ορίζονται από ένα σύνολο συντακτικών και εννοιολογικών κανόνων, που ορίζουν τη δομή και το νόημα, αντίστοιχα, των προτάσεων της γλώσσας. Οι γλώσσες προγραμματισμού χρησιμοποιούνται για να διευκολύνουν την οργάνωση και διαχείριση πληροφοριών, αλλά και για την ακριβή διατύπωση αλγορίθμων. Ορισμένοι ειδικοί χρησιμοποιούν τον όρο γλώσσα προγραμματισμού μόνο για τυπικές γλώσσες που μπορούν να εκφράσουν όλους τους πιθανούς αλγορίθμους. Μη-υπολογιστικές γλώσσες όπως η HTML ή τυπικές γραμματικές όπως η BNF δεν λέγονται συνήθως γλώσσες προγραμματισμού. (Scott, 2009)

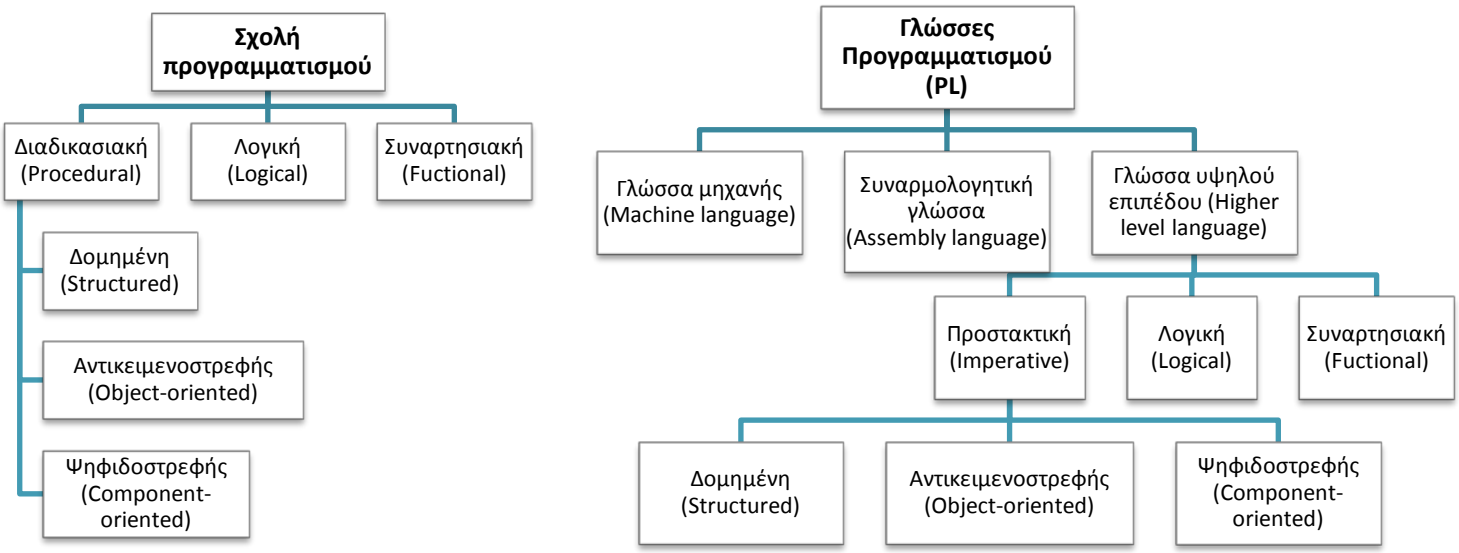
Στη Λογισμική Μηχανική χρησιμοποιούνται οι προστακτικές γλώσσες και μάλιστα σήμερα χρησιμοποιείται περισσότερο η δεύτερη υποκατηγορία τους, δηλαδή η υποκατηγορία των αντικειμενοστρεφών ΓΠ.

Δεν υπάρχει απλός τρόπος να κατηγοριοποιηθούν οι γλώσσες προγραμματισμού. Αυτό συμβαίνει γιατί συνήθως κάθε γλώσσα προγραμματισμού περιέχει επιρροές από πολλές προηγούμενες γλώσσες, συνδυάζοντας θετικά στοιχεία και προσθέτοντας νέα. (Scott, 2009)




**ΔΙΑΓΡΑΜΜΑ 18 Απλοποιημένο Διάγραμμα Ιστορίας Γλωσσών Προγραμματισμού ("Images For > History Of Programming Languages," n.d.)**

Η κατηγοριοποίηση είναι ακόμα πιο περίπλοκη για το λόγο ότι πολλές γλώσσες συνήθως ανήκουν σε παραπάνω από μία κατηγορίες. Για παράδειγμα, η Java είναι τόσο αντικειμενοστρεφής όσο και παράλληλη γλώσσα, δεδομένου ότι υποστηρίζει την οργάνωση των δεδομένων και υπολογισμών σε αντικείμενα, αλλά επιτρέπει επίσης και τη δημιουργία προγραμμάτων με ταυτόχρονα νήματα (threads) που εκτελούνται παράλληλα.




**ΔΙΑΓΡΑΜΜΑ 19 Κατηγοριοποίηση Γλωσσών Προγραμματισμού (Scott, 2009)**

Δεδομένης της δυσκολίας στην κατηγοριοποίηση, μπορούμε να κατηγοριοποιήσουμε τις γλώσσες προγραμματισμού με διάφορους τρόπους. Οι συνηθέστεροι τρόποι είναι:

- με βάση τον τρόπο οργάνωσης του προγράμματος
- με βάση τον στόχο που έχει η γλώσσα
- με βάση τον τρόπο που περιγράφουν το ζητούμενο αποτέλεσμα

Στην πρώτη περίπτωση προκύπτουν κατηγορίες όπως:

- Διαδικαστικές γλώσσες (procedural) όπου το πρόγραμμα είναι οργανωμένο σε διαδικασίες, που αποτελούνται από σειρές εντολών που περιγράφουν αλγορίθμους. Παραδείγματα γλωσσών που ανήκουν σε αυτή την κατηγορία είναι η Pascal ή η C.
- Αντικειμενοστρεφείς γλώσσες (object-oriented) όπου το πρόγραμμα είναι οργανωμένο σε αντικείμενα. Ένα αντικείμενο είναι μια μονάδα που αποτελείται από την περιγραφή κάποιων δεδομένων και την περιγραφή των αλγορίθμων που τα επεξεργάζονται. Ένα αντικειμενοστρεφές πρόγραμμα αποτελείται από διάφορα αντικείμενα που αλληλεπιδρούν μεταξύ τους. Παραδείγματα αντικειμενοστρεφών γλωσσών είναι η Java ή η C++.
- Συναρτησιακές γλώσσες (functional) όπου οι υπολογισμοί εκφράζονται ως εφαρμογές μαθηματικών συναρτήσεων, σε αντίθεση με τα άλλα είδη προγραμματισμού όπου οι υπολογισμοί εκφράζονται ως σειρές εντολών, όπου η κάθε μία αλλάζει με κάποιο τρόπο την κατάσταση του συστήματος. Θεωρητικό τους υπόβαθρο είναι ο λ-λογισμός. Χαρακτηριστικές συναρτησιακές γλώσσες είναι η Lisp, η Haskell (μόνο για εκπαιδευτικούς σκοπούς) και η OCaml.

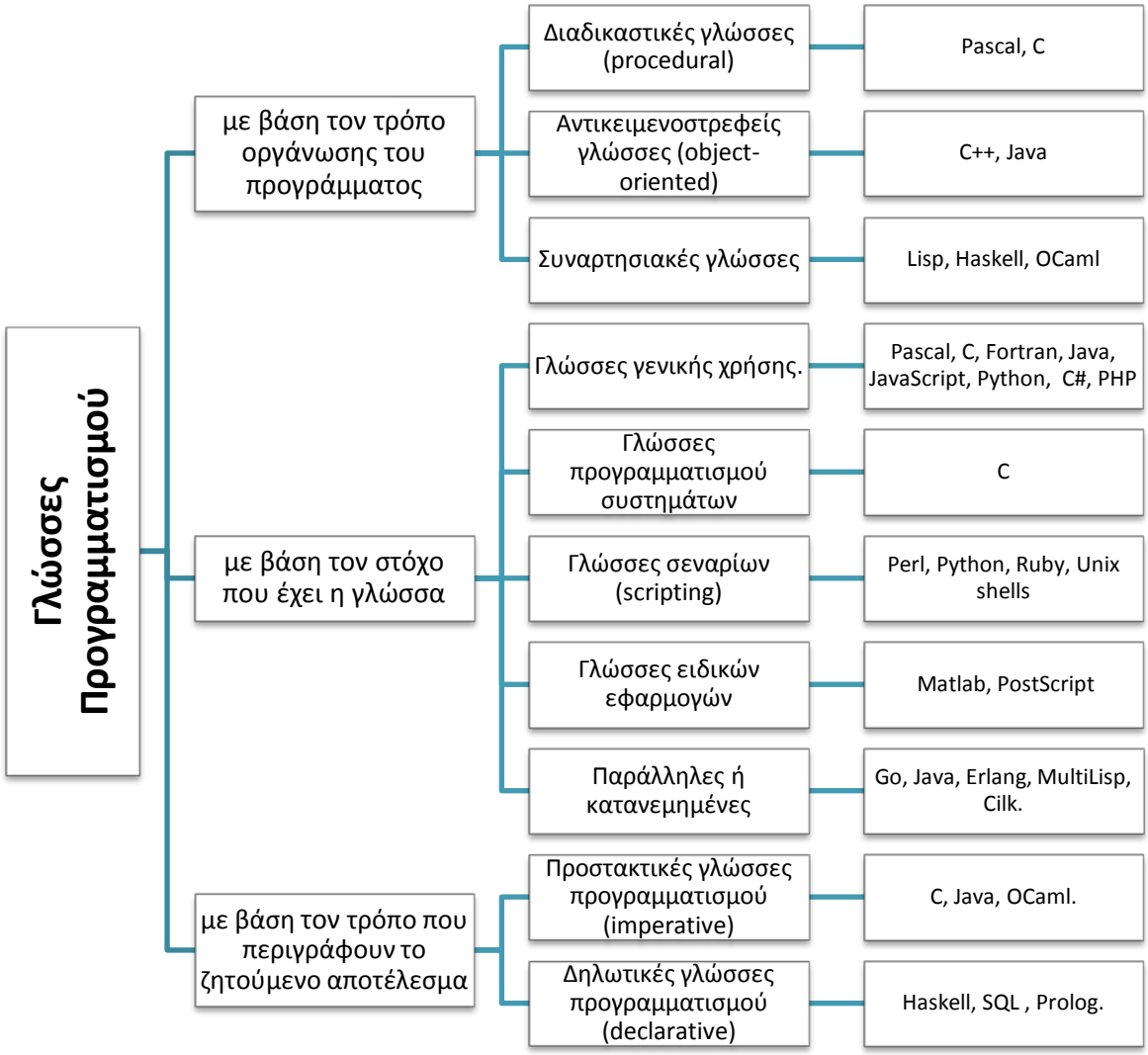
Στην περίπτωση που η κατηγοριοποίηση των γλωσσών προγραμματισμού γίνει με βάση το στόχο που έχει η γλώσσα, υπάρχουν οι παρακάτω κατηγορίες:

- Γλώσσες γενικής χρήσης. Σε αυτήν την κατηγορία ταξινομούνται γλώσσες που δημιουργήθηκαν για τον προγραμματισμό γενικών εφαρμογών, καθώς και πολλές εκπαιδευτικές γλώσσες που αποδείχτηκαν χρήσιμες για την ανάπτυξη γενικών εφαρμογών, όπως η Pascal.
- Γλώσσες προγραμματισμού συστημάτων, που χρησιμοποιούνται συνήθως για τον προγραμματισμό λειτουργικών συστημάτων ή οδηγών (drivers) υλικού, όπου χρειάζεται πολλές φορές ο προγραμματιστής να έχει έλεγχο και γνώση του πως λειτουργεί το υλικό. Η πιο συχνά χρησιμοποιούμενη γλώσσα προγραμματισμού συστημάτων είναι η C.
- Γλώσσες σεναρίων (scripting). Αυτές οι γλώσσες χρησιμοποιούνται συνήθως για τη γρήγορη ανάπτυξη μικρών προγραμμάτων, σε περιπτώσεις που ο χρόνος του προγραμματιστή είναι πιο πολύτιμος από την ταχύτητα εκτέλεσης του προγράμματος, όπως για παράδειγμα συμβαίνει όταν το πρόγραμμα απλά αυτοματοποιεί απλές λειτουργίες. Παραδείγματα γλωσσών σεναρίων (scripting) είναι η Perl, η Python, η Ruby ή τα κελύφη του λειτουργικού συστήματος Unix (shells).
- Γλώσσες ειδικών εφαρμογών. Σε αυτή την κατηγορία ανήκουν γλώσσες που αναπτύχθηκαν ειδικά για μια συγκεκριμένη εφαρμογή. Για παράδειγμα, η γλώσσα PostScript είναι σχεδιασμένη ειδικά για να περιγράφονται με λεπτομέρεια κείμενα προς εκτύπωση, ενώ η γλώσσα Matlab είναι σχεδιασμένη για την επεξεργασία πινάκων από αριθμητικά δεδομένα.
- Παράλληλες ή κατανεμημένες γλώσσες. Στη συγκεκριμένη κατηγορία ταξινομούνται γλώσσες που επιτρέπουν την ανάπτυξη παράλληλων προγραμμάτων, όπου πολλές εντολές εκτελούνται ταυτόχρονα σε πολλούς υπολογιστές, έτσι ώστε το τελικό αποτέλεσμα να προκύψει γρηγορότερα. Οι παράλληλες γλώσσες προσφέρουν συνήθως εύκολους τρόπους επικοινωνίας μεταξύ των νημάτων που εκτελούνται παράλληλα, καθώς και τρόπους ώστε να δημιουργούνται καινούριες παράλληλες εκτελέσεις. Παραδείγματα γλωσσών που ανήκουν (και) σε αυτή την κατηγορία είναι η Go, η Java, η Erlang, η MultiLisp ή η Cilk.

Τέλος, στην περίπτωση που η κατηγοριοποίηση γίνεται με βάση τον τρόπο που περιγράφεται το ζητούμενο, υπάρχουν οι παρακάτω κατηγορίες:

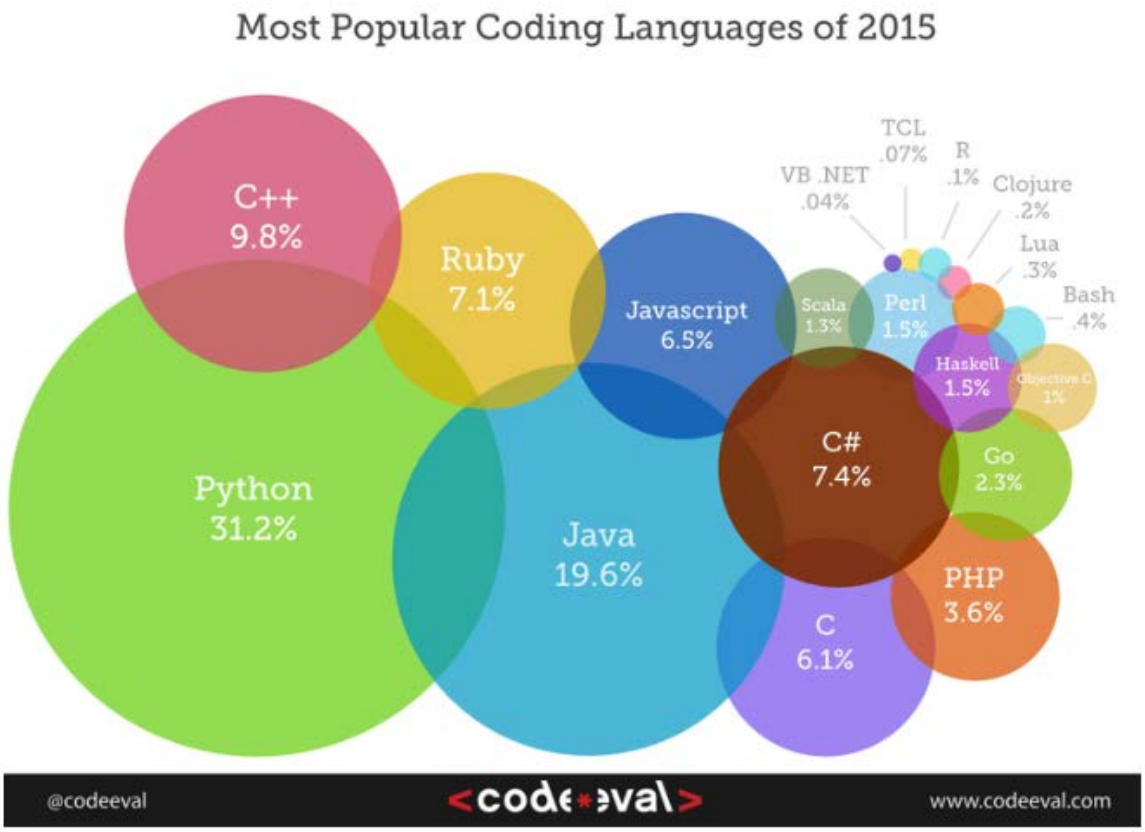
- Προστακτικές γλώσσες προγραμματισμού (imperative) είναι οι γλώσσες που περιγράφουν το ζητούμενο αποτέλεσμα κατασκευαστικά, δίνοντας μια σειρά εντολών που όταν εκτελεστούν παράγουν το ζητούμενο αποτέλεσμα. Τέτοιες γλώσσες είναι η C, η Java, αλλά και η OCaml.
- Δηλωτικές γλώσσες προγραμματισμού (declarative) είναι οι γλώσσες που περιγράφουν το ζητούμενο αποτέλεσμα χρησιμοποιώντας τις ιδιότητες που

έχει, και όχι τον τρόπο με τον οποίο υπολογίζεται. Παραδείγματα δηλωτικών γλωσσών είναι η Haskell, η SQL και η Prolog.

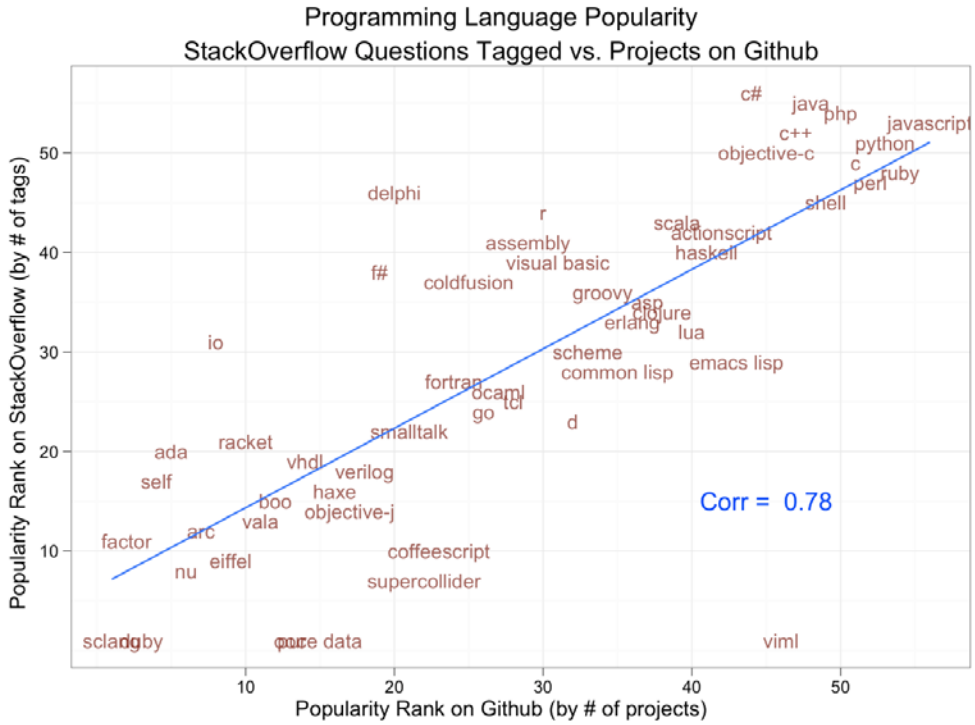



**ΔΙΑΓΡΑΜΜΑ 20 Κατηγοριοποίηση Γλωσσών Προγραμματισμού ("Γλώσσες Προγραμματισμού - Κοινότητα Ελεύθερου Λογισμικού ΕΜΠ," n.d.)**

Ενδιαφέρον παρουσιάζουν τα δημοσιευμένα στοιχεία για τις «πιο δημοφιλείς γλώσσες Προγραμματισμού» ("Most Popular Coding Languages") βασιζόμενα σε εκατοντάδες χιλιάδες σημεία δεδομένων που έχουν συλλεχτεί από επεξεργασία πάνω από 600,000+ δοκιμές και προσκλήσεις κωδικοποίησης πάνω από 2.000+ εργοδοτών. ("Most Popular Coding Languages of 2015 — CodeEval," n.d.)



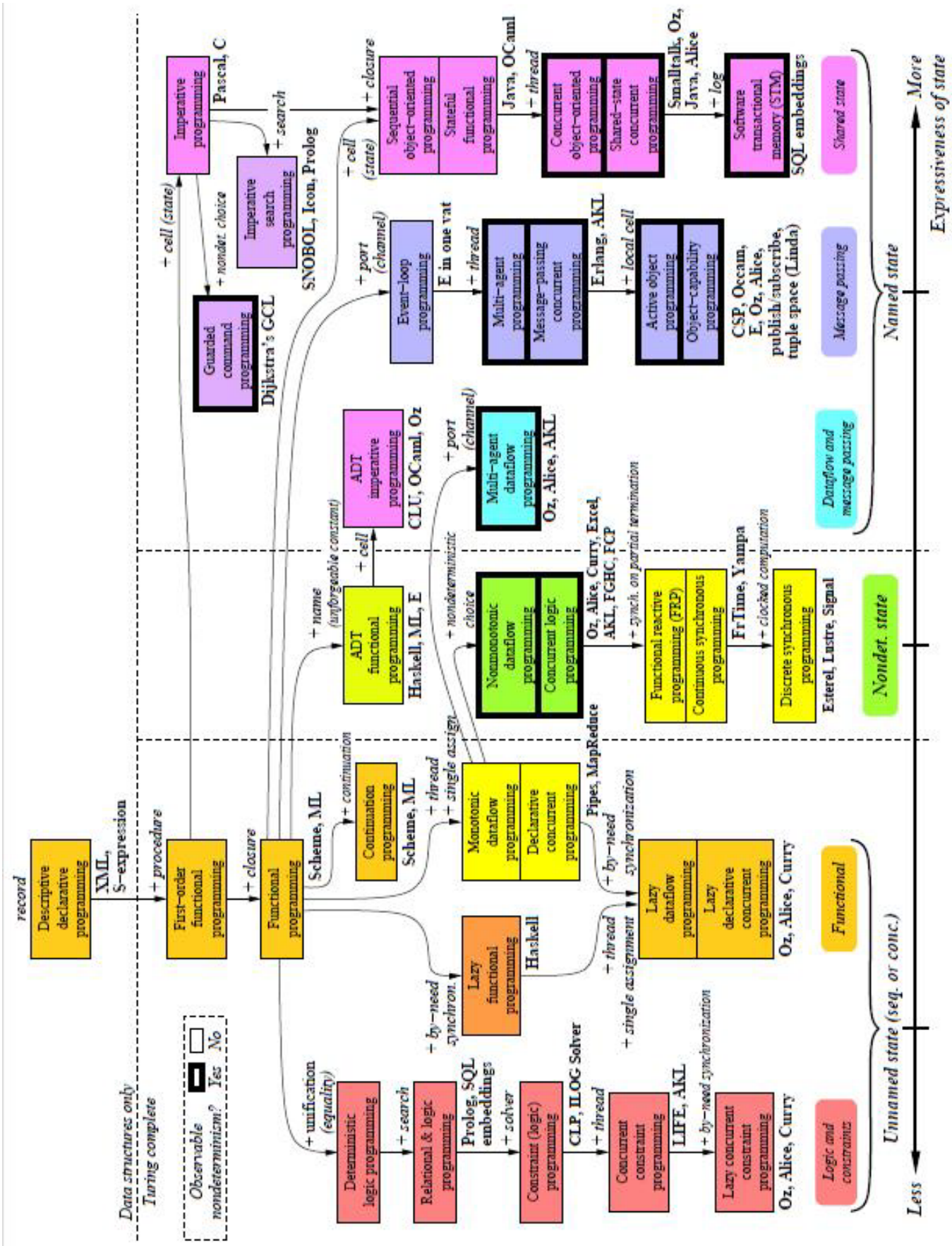

**ΔΙΑΓΡΑΜΜΑ 21 Χρήση Γλωσσών Προγραμματισμού ("Most Popular Coding Languages of 2015 — CodeEval," n.d.)**




**ΔΙΑΓΡΑΜΜΑ 22 Δημοτικότητα Ερωτημάτων ανά Γλώσσα Προγραμματισμού στο StackOverflow ("dataists » Ranking the popularity of programming languages," n.d.)**

Προκύπτει εύλογα το ερώτημα, γιατί υπάρχει τόσο μεγάλη ποικιλία γλωσσών προγραμματισμού, δεδομένου μάλιστα ότι (σχεδόν) όλες είναι θεωρητικά ισοδύναμες. Η απάντηση είναι πολύ απλή, γιατί δεν υπάρχει μία και μόνο γλώσσα κατάλληλη για όλες τις δουλειές. Ενδεχομένως να μην έχει βρεθεί ακόμα, ενδεχομένως να μην υπάρξει ποτέ. Κάθε γλώσσα ενδείκνυται περισσότερο για ορισμένα είδη προγραμμάτων και λιγότερο για άλλα. Επιπλέον, ένα πρότυπο προγραμματισμού είναι μια προσέγγιση για τον προγραμματισμό που βασίζεται σε μια μαθηματική θεωρία ή ένα συνεκτικό σύνολο αρχών. Κάθε πρότυπο υποστηρίζει ένα σύνολο εννοιών που το κάνει καλύτερο για ένα ορισμένο είδος προβλήματος.

Για παράδειγμα, ο αντικειμενοστραφής προγραμματισμός είναι καλύτερος για προβλήματα με έναν μεγάλο αριθμό σχετικών δεδομένων, οργανωμένων σε μια ιεραρχία. Ο λογικός προγραμματισμός είναι προτιμότερος για το μετασχηματισμό ή την πλοήγηση πολύπλοκων συμβολικών δομών σύμφωνα με λογικούς κανόνες. Ο σύγχρονος διακριτός προγραμματισμός είναι προτιμότερος για διαδραστικά προβλήματα, δηλαδή τα προβλήματα που αντιδρούν στις ακολουθίες των εξωτερικών γεγονότων. Κάθε γλώσσα σχετίζεται με ένα ή περισσότερα πρότυπα, και κάθε πρότυπο αποτελείται από ένα σύνολο των εννοιών. (“Languages | Updates from a continuous developer,” n.d.)




**ΔΙΑΓΡΑΜΜΑ 23 Χαρακτηριστική Γλωσσών Προγραμματισμού ("Ideal programming language learning sequence? - Programmers Stack Exchange," n.d.)**

# Κεφάλαιο 2

## 2. Ανάπτυξη πληροφοριακών συστημάτων

Η Ανάπτυξη πληροφοριακών συστημάτων είναι ένας τομέας που πραγματεύεται τεχνικές, μεθοδολογίες, πρακτικές και εργαλεία για την συστηματική, μεθοδική και ποσοτικοποιημένη προδιαγραφή, σχεδίαση, υλοποίηση, έλεγχο, και συντήρηση συστημάτων λογισμικού υψηλής ποιότητας και εντός δεδομένου προϋπολογισμού και χρόνου εκτέλεσης. [IEEE Standard 610.12]

Η ανάπτυξη λογισμικού είναι ο προγραμματισμός ηλεκτρονικών υπολογιστών, η καταγραφή, ο έλεγχος και η διόρθωση σφαλμάτων που αφορούν στη δημιουργία και στη συντήρηση εφαρμογών και πλαισίων που εμπλέκονται στον κύκλο ζωής της εκάστοτε έκδοσης λογισμικού και καταλήγουν σε ένα ολοκληρωμένο προϊόν λογισμικού. Ο όρος αναφέρεται σε μια διαδικασία γραφής και συντήρησης του πηγαίου κώδικα, αλλά με την ευρύτερη έννοια περιλαμβάνει όλα εκείνα που εμπλέκονται ανάμεσα στη σύλληψη της ιδέας μέχρι την τελική έκδοση του λογισμικού, ιδανικά σε μια σχεδιασμένη και δομημένη διαδικασία. Ως εκ τούτου, η ανάπτυξη λογισμικού ενδέχεται να περιλαμβάνει την έρευνα, την καινοτόμα ανάπτυξη, την προτυποποίηση, την τροποποίηση, την επαναχρησιμοποίηση, την αναδιοργάνωση, την συντήρηση, ή και άλλες δραστηριότητες που καταλήγουν σε προϊόντα λογισμικού. (“NEW PRODUCT DEVELOPMENT GLOSSARY,” n.d.)

Η ανάπτυξη λογισμικού συνήθως γίνεται για διάφορους σκοπούς, τρεις συνηθέστεροι των οποίων είναι να καλύψουν τις συγκεκριμένες ανάγκες ενός πελάτη / επιχείρησης (η περίπτωση με του επί παραγγελία λογισμικού (custom software)), μια ανάγκη που γίνεται αντιληπτή από κάποιο σύνολο δυνητικών χρηστών (η περίπτωση των εμπορικών και λογισμικών ανοιχτού κώδικα), είτε τέλος για προσωπική χρήση (π.χ. ένας επιστήμονας μπορεί να γράψει το λογισμικό για την αυτοματοποίηση μιας δραστηριότητας).

Η ανάγκη για καλύτερο έλεγχο της ποιότητας της διαδικασίας ανάπτυξης λογισμικού έχει δώσει ώθηση στον κλάδο της μηχανικής λογισμικού, η οποία έχει ως στόχο να εφαρμόσει τη συστηματική προσέγγιση αφομοιώνοντας τα πρότυπα της μηχανικής στη διαδικασία της ανάπτυξης λογισμικού.

Υπάρχουν πολλές προσεγγίσεις για τη διαχείριση του λογισμικού έργου, γνωστές ως μοντέλα κύκλου ζωής ανάπτυξης λογισμικού, μεθοδολογίες, διαδικασίες, ή μοντέλα. Το μοντέλο καταρράκτη (waterfall model) είναι μια παραδοσιακή εκδοχή, σε αντίθεση με την πιο πρόσφατη καινοτομία της ευέλικτης ανάπτυξης λογισμικού (agile software development). (Klimczak, 2013)

## 2.1. Δραστηριότητες ανάπτυξης λογισμικού

### 2.1.1. Αναγνώριση της ανάγκης και καθορισμός προδιαγραφών

Οι πηγές των ιδεών για τα προϊόντα λογισμικού είναι πολυπληθείς. Αυτές οι ιδέες μπορούν να προέλθουν από την έρευνα της αγοράς, συμπεριλαμβανομένων των δημογραφικών στοιχείων των πιθανών νέων πελατών, των υπαρχόντων πελατών ή χρηστών, του τμήματος πωλήσεων, ή του προσωπικού ανάπτυξης λογισμικού, ή δημιουργικά τρίτα πρόσωπα. Οι ιδέες για προϊόντα λογισμικού συνήθως αξιολογούνται πρώτα από το τμήμα μάρκετινγκ για την οικονομική σκοπιμότητα, την εναρμόνιση με τα υπάρχοντα προϊόντα και διαδικασίες, τις πιθανές επιπτώσεις στις υπάρχουσες συνθήκες παραγωγής και αγοράς, τα απαιτούμενα χαρακτηριστικά, καθώς και για την συνοχή με τους στόχους μάρκετινγκ της εταιρείας. Σε μια φάση αξιολόγησης μάρκετινγκ, οι παραδοχές κόστους και χρόνου αξιολογούνται. Η απόφαση για το αν το έργο πρέπει να υλοποιηθεί λαμβάνεται σε πρώτη φάση, με βάση τις λεπτομερείς πληροφορίες που προέρχονται από το προσωπικό μάρκετινγκ και ανάπτυξης. (Bhandari and Sehgal, 2014)

Οι απαιτήσεις και οι προδιαγραφές για τη δημιουργία ενός λογισμικού είναι ένας βασικός τομέας της μηχανικής λογισμικού και αφορά τον σαφή καθορισμό των αναγκών των ενδιαφερομένων, οι οποίες ζητούνται να καλυφθούν από το λογισμικό. Επειδή, η ανάπτυξη λογισμικού ενδέχεται να περιλαμβάνει συμβιβασμούς ή και επεκτάσεις πέρα από τις απαιτήσεις του πελάτη, ένα έργο ανάπτυξης λογισμικού μπορεί να παρεκτραπεί και σε λιγότερο τεχνικά ζητήματα, όπως το ανθρώπινο δυναμικό, η διαχείριση των κινδύνων, η πνευματική ιδιοκτησία, η κατάρτιση του προϋπολογισμού, η διαχείριση κρίσεων, κλπ. Αυτές οι διαδικασίες μπορούν επίσης να προκαλέσουν την επικάλυψη του ρόλου της ανάπτυξης της επιχείρησης με την ανάπτυξη λογισμικού. (Bhandari and Sehgal, 2014)

### 2.1.2. Προγραμματισμός Λογισμικού Έργου

Ο σκοπός του προγραμματισμού της διαδικασίας της ανάπτυξης πληροφοριακών συστημάτων είναι η δημιουργία λογικών σχεδίων για την εφαρμογή της μηχανικής λογισμικού και τη διαχείριση του Λογισμικού Έργου. Ο προγραμματισμός του Λογισμικού Έργου περιλαμβάνει την ανάπτυξη εκτιμήσεων για το έργο που πρόκειται να εκτελεστεί, για τη θέσπιση των αναγκαίων δεσμεύσεων, καθώς και τον καθορισμό του σχεδίου για την εκτέλεση των εργασιών.

Η διαδικασία του προγραμματισμού του Λογισμικού Έργου αρχίζει με μια αναφορά των εργασιών που πρέπει να εκτελεστούν, τον προσδιορισμό των περιορισμών και των στόχων, που καθορίζουν και δεσμεύουν τη διαδικασία ανάπτυξης λογισμικού. Επίσης, περιλαμβάνει όλες εκείνες τις ενέργειες για να εκτιμηθεί ο όγκος των εργασιών για να ολοκληρωθεί το προϊόν λογισμικού και κατά αντιστοιχία οι πόροι που απαιτούνται. Επιπλέον, παράγει ένα χρονοδιάγραμμα, περιλαμβάνει τον εντοπισμό και την αξιολόγηση των κινδύνων λογισμικού, και διαπραγματεύεται τις αναλήψεις υποχρεώσεων. Ενδεχομένως να απαιτούνται επαναλήψεις των παραπάνω βημάτων για να προκύψει το πρόγραμμα για το έργο του λογισμικού (δηλαδή, το σχέδιο ανάπτυξης λογισμικού). Αυτό το πρόγραμμα αποτελεί τη βάση για την εκτέλεση και τη διαχείριση των δραστηριοτήτων του λογισμικού έργου και αντιμετωπίζει τις δεσμεύσεις προς τον πελάτη, σύμφωνα με τους πόρους, τους περιορισμούς και τις δυνατότητες του σχεδιασμού. (Paulk, 1995)

Ο προγραμματισμός του λογισμικού έργου είναι συνυφασμένος με το αντικείμενο της επιχειρησιακής ανάλυσης, που είναι το σύνολο των εργασιών και τεχνικών που συνδέουν τα ενδιαφερόμενα μέρη μεταξύ τους, προκειμένου να κατανοήσουν τη δομή, τις πολιτικές, και τις λειτουργίες του οργανισμού, και να προτείνουν λύσεις για την επίτευξη των στόχων. (“BABOK Guide Online - IIBA | International Institute of Business Analysis,” n.d.)

### 2.1.3. Μελέτη και σχεδιασμός λογισμικού έργου

Τα περισσότερα συστήματα, ακόμα και τα μικρότερα, πρέπει να σχεδιαστούν πριν υλοποιηθούν, προκειμένου να αποφευχθεί η δαπανηρή επανάληψη των εργασιών λόγω παραλείψεων στο σχεδιασμό. Ο σχεδιασμός λογισμικού έργου είναι η διαδικασία κατά την οποία ορίζονται οι προδιαγραφές των τεχνημάτων (artifacts) λογισμικού, που προορίζεται για την επίτευξη των στόχων, χρησιμοποιώντας ένα σύνολο πρωτογενών στοιχείων, ενώ ταυτόχρονα υπόκειται σε περιορισμούς. Ο σχεδιασμός λογισμικού έργου αναφέρεται σε όλες τις δραστηριότητες που αφορούν την σύλληψη, τη διαμόρφωση, την υλοποίηση, την λειτουργία, και την τροποποίηση πολύπλοκων συστημάτων. Περιλαμβάνει την επίλυση προβλημάτων κατά το

σχεδιασμό μιας λύσης λογισμικού. Αυτό περιλαμβάνει τόσο θέματα χαμηλού επιπέδου και σχεδιασμό αλγόριθμων όσο και θέματα υψηλού επιπέδου, τον αρχιτεκτονικό σχεδιασμό. (Klimczak, 2013)

Ο σχεδιασμός λογισμικού μπορεί να θεωρηθεί ως μια λύση σε ένα δεδομένο πρόβλημα με την αξιοποίηση των διαθέσιμων δυνατοτήτων. Ο σχεδιασμός επικεντρώνεται στις υπάρχουσες δυνατότητες, και μπορεί να υπάρχουν πολλαπλές λύσεις για το ίδιο πρόβλημα, ανάλογα με το περιβάλλον το οποίο θα φιλοξενήσει την εκάστοτε λύση, όπως για παράδειγμα λειτουργικά συστήματα, ιστοσελίδες, εφαρμογές κινητών συσκευών ή νέες τεχνολογίες νέφους (cloud computing). Επίσης, ο σχεδιασμός εξαρτάται και από το περιβάλλον για το οποίο αρχικά δημιουργήθηκε. Κατά το σχεδιασμό του λογισμικού, δύο σημαντικοί παράγοντες που οπωσδήποτε πρέπει να αντιμετωπιστούν είναι η ασφάλεια και η ευχρηστία του. (“Software Design Methodology,” n.d.)

Ο σχεδιασμός του λογισμικού αναφέρεται τόσο σε μια διαδικασία, όσο και σε ένα μοντέλο. Η διαδικασία σχεδιασμού είναι μια σειρά από βήματα που επιτρέπουν στο σχεδιαστή να περιγράψει όλες τις πτυχές του λογισμικού που πρόκειται να κατασκευαστεί. Είναι σημαντικό να σημειωθεί, ωστόσο, ότι η διαδικασία σχεδιασμού δεν είναι απλά ένα βιβλίο σαφών οδηγιών. Έννοιες όπως η δημιουργική ικανότητα, η εμπειρία του παρελθόντος, μια αίσθηση των στοιχείων ενός «καλού» λογισμικού, και μια συνολική δέσμευση για την ποιότητα αποτελούν κρίσιμους παράγοντες επιτυχίας για έναν αποδοτικό σχεδιασμό. Από την άλλη, το μοντέλο σχεδιασμού είναι το ισοδύναμο των σχεδίων ενός αρχιτέκτονα για ένα σπίτι. Ξεκινά από τη γενική αναπαράσταση του λογισμικού έργου που θα κατασκευαστεί (π.χ., κατά αναλογία μια τρισδιάστατη αρχιτεκτονική απόδοση του κτιρίου) και σιγά-σιγά εισάγονται οι λεπτομέρειες που θα παρέχουν την απαραίτητη καθοδήγηση για την κατασκευή (π.χ., η μελέτη Η/Μ). Ομοίως, το μοντέλο σχεδιασμού που έχει δημιουργηθεί για το λογισμικό παρέχει μια ποικιλία λεπτομερών σχεδίων για τους επιμέρους τομείς του έργου. Υπάρχουν διάφορες ημι-τυποποιημένες μέθοδοι για τη βοήθεια του σχεδιασμού, όπως η Ενοποιημένη Γλώσσα Μοντελοποίησης (Unified Modeling Language) και οι θεμελιώδεις έννοιες μοντελοποίησης (Fundamental Modeling Concepts), γενικές μέθοδοι όπως η Systems Modeling Language (SysML) ή η παραδοσιακή Top-down and bottom-up design, Enhanced Function Flow Block Diagram (EFFBD), και η Icam (Integrated Computer Aided Manufacturing) DEFinition for Function Modeling IDEF0. (Buede, 2009)

### 2.1.4. Κατασκευή, έλεγχος και τεκμηρίωση

Η κατασκευή ή υλοποίηση είναι το μέρος της διαδικασίας όπου οι μηχανικοί λογισμικού συντάσσουν τον κώδικα για το λογισμικό έργο. Είναι η λεπτομερής δημιουργία λειτουργικού κατανοητού λογισμικού μέσω ενός συνδυασμού της

κωδικοποίησης, ελέγχου, δοκιμών μονάδων, δοκιμών ολοκλήρωσης, καθώς και τον εντοπισμό και διόρθωση σφαλμάτων. Συνδέεται με όλες τις άλλες ειδικότητες μηχανικών λογισμικού, εντονότερα με το σχεδιασμό και τη δοκιμή του λογισμικού. (McConnell, 2004)

Πολυάριθμα μοντέλα έχουν δημιουργηθεί για την ανάπτυξη λογισμικού, μερικά από τα οποία δίνουν έμφαση στην κατασκευή περισσότερο από άλλα. Μερικά μοντέλα είναι πιο γραμμικά από την οπτική της κατασκευής, όπως του καταρράκτη και των μοντέλων κύκλου ζωής σταδιακής παράδοσης (staged-delivery life cycle models). Τα μοντέλα αυτά αντιμετωπίζουν την κατασκευή ως μια δραστηριότητα που λαμβάνει χώρα μόνο μετά την ολοκλήρωση σημαντικών εργασιών, συμπεριλαμβανομένων των λεπτομερών προδιαγραφών έργου, και του εκτεταμένου προγραμματισμού και λεπτομερούς σχεδιασμού. Άλλα μοντέλα είναι πιο επαναλαμβανόμενα, όπως εξελικτική προτυποποίηση (evolutionary prototyping), ο Ακραίος Προγραμματισμός (Extreme Programming), και η ολιστική προσέγγιση Scrum. Αυτές οι προσεγγίσεις έχουν την τάση να αντιμετωπίζουν την κατασκευή ως μια δραστηριότητα που λαμβάνει χώρα ταυτόχρονα και επικαλύπτει άλλες δραστηριότητες ανάπτυξης λογισμικού, συμπεριλαμβανομένων των προδιαγραφών, του σχεδιασμού και του προγραμματισμού. (Bourque et al., 2014)

Ο έλεγχος λογισμικού αποτελεί αναπόσπαστο και σημαντικό στάδιο της διαδικασίας ανάπτυξης λογισμικού. Αυτό το μέρος της διαδικασίας διασφαλίζει ότι τα ελαττώματα αναγνωρίζονται και αντιμετωπίζονται όσο το δυνατόν συντομότερα. Σε ορισμένες διαδικασίες οι δοκιμές μπορούν να αναπτυχθούν μόλις πριν από την υλοποίηση του λογισμικού και να χρησιμεύσουν ως οδηγός για την ορθότητα της υλοποίηση.(McConnell, 2004)

Η τεκμηρίωση συνοδεύει την εσωτερική σχεδίαση του λογισμικού για το σκοπό της μελλοντικής συντήρησης και επεκτασιμότητας. Αυτό μπορεί επίσης να περιλαμβάνει τη συγγραφή μιας Διεπαφής Προγραμματισμού Εφαρμογών (Application Programming Interface API). Η ομάδα ανάπτυξης θα κρίνει την αναγκαιότητα και θα καθορίσει την παραγωγή κατάλληλης τεκμηρίωσης κατά τη διαδικασία ανάπτυξης του λογισμικού. Τα μοντέλα βασισμένα σε σχεδιασμό (Plan-driven models) (π.χ. Καταρράκτη - Waterfall) γενικά παράγουν περισσότερη τεκμηρίωση από τα ευέλικτα (Agile) μοντέλα.

### 2.1.5. Κυκλοφορία, λειτουργία και συντήρηση

Η κυκλοφορία του λογισμικού έργου ξεκινά αμέσως αφού ολοκληρωθούν οι δοκιμές ελέγχου του κώδικα, και εν συνεχεία εγκριθεί η έκδοση του και ξεκινήσει η πώληση ή η διανομή του. Αυτό μπορεί να περιλαμβάνει την εγκατάσταση, την παραμετροποίηση, τον έλεγχο, και, ενδεχομένως, μια εκτεταμένη περίοδο αξιολόγησης της λειτουργίας του. Επιπλέον, η εκπαίδευση των χρηστών και η

υποστήριξη του λογισμικού είναι σημαντικές διαδικασίες, καθώς το λογισμικό είναι αποτελεσματικό μόνο αν χρησιμοποιηθεί σωστά. (Chiang, 2010)

Η κυκλοφορία του λογισμικού είναι το σύνολο των δραστηριοτήτων που καθιστούν ένα σύστημα λογισμικού να είναι διαθέσιμο για χρήση. Η γενική διαδικασία της κυκλοφορίας αποτελείται από πολλές αλληλένδετες δραστηριότητες με πιθανές μεταβάσεις μεταξύ τους. Αυτές οι δραστηριότητες μπορούν να εμφανιστούν στην πλευρά του παραγωγού είτε στην πλευρά του καταναλωτή ή αμφότερα. Επειδή, κάθε σύστημα λογισμικού είναι μοναδικό, οι ακριβείς διαδικασίες στο πλαίσιο κάθε δραστηριότητας δύσκολα μπορούν να οριστούν. Ως εκ τούτου, η κυκλοφορία θα πρέπει να ερμηνευθεί ως μια γενική διαδικασία που πρέπει να προσαρμοστεί ανάλογα με τις συγκεκριμένες προδιαγραφές και χαρακτηριστικά. (Chiang, 2010)

Η συντήρηση και η επέκταση του λογισμικού για την αντιμετώπιση σφαλμάτων που δεν εμφανίστηκαν στο παρελθόν ή την κάλυψη νέων απαιτήσεων, μπορεί να χρειάζεται σημαντικό χρόνο και προσπάθεια, καθώς ενδεχομένως να είναι απαραίτητος ο επανασχεδιασμός του λογισμικού. Υπάρχει μια εσφαλμένη εντύπωση ότι ο περισσότερος όγκος εργασίας όσον αφορά στη συντήρηση είναι στη διόρθωση σφαλμάτων, ενώ στην πραγματικότητα πρόκειται για λειτουργικές βελτιώσεις ή επεκτάσεις που απαντούν στις αναφορές προβλημάτων των χρηστών. (Finkelstein and International Conference on Software Engineering, 2000)

## 2.2. Μεθοδολογίες

Η μεθοδολογία ανάπτυξης λογισμικού (επίσης γνωστή ως διαδικασία, μοντέλο, ή κύκλος ζωής ανάπτυξης λογισμικού) είναι ένα πλαίσιο που χρησιμοποιείται στο σχεδιασμό, τη δομή και τον έλεγχο της διαδικασίας ανάπτυξης συστημάτων πληροφοριών. Μια ευρεία ποικιλία τέτοιων μεθοδολογιών έχουν αναπτυχθεί με την πάροδο των ετών, καθεμία με τα δικά της αναγνωρισμένα πλεονεκτήματα και αδυναμίες. Υπάρχουν πολλές διαφορετικές προσεγγίσεις για την ανάπτυξη λογισμικού: μερικές είναι περισσότερο δομημένες και βασισμένες στη μηχανική προσέγγιση για την ανάπτυξη επιχειρηματικών λύσεων, ενώ κάποιες μπορεί να χρησιμοποιούν μια πιο σταδιακή προσέγγιση, όπου το λογισμικό εξελίσσεται καθώς αναπτύσσεται κάθε επιμέρους τμήμα του. Δεν υπάρχει μια μεθοδολογία ανάπτυξης συστήματος κατάλληλη για χρήση από όλα τα έργα, κατά αντιστοιχία με το ότι δεν υπάρχει και μια γλώσσα προγραμματισμού που να είναι κατάλληλη για όλες τις εφαρμογές. Κάθε μία από τις διαθέσιμες μεθόδους είναι καταλληλότερη για συγκεκριμένα είδη έργων, με βάση διάφορα ζητήματα τεχνικά, οργανωτικά, σχετικά με το εκάστοτε έργο και την εκάστοτε ομάδα. (“Delivering large-scale IT projects on time, on budget, and on value | McKinsey & Company,” n.d.)

Οι περισσότερες μεθοδολογίες μοιράζονται κάποιο συνδυασμό από τα ακόλουθα

στάδια της ανάπτυξης λογισμικού: (Wiegers, 2013)

- Ανάλυση του προβλήματος
- Έρευνα αγοράς
- Καθορισμός απαιτήσεων για την προτεινόμενη επιχειρησιακή λύση
- Σχεδιασμός της επιχειρησιακής λύσης σε λύση λογισμικού
- Κατασκευή (κωδικοποίηση) του λογισμικού
- Έλεγχος του λογισμικού
- Εγκατάσταση και λειτουργία του λογισμικού
- Συντήρηση του λογισμικού και διόρθωση σφαλμάτων

Αυτά τα στάδια συχνά αναφέρονται ως κύκλος ζωής της ανάπτυξης λογισμικού (software development lifecycle, or SDLC). Υπάρχουν διαφορετικές προσεγγίσεις για την ανάπτυξη λογισμικού που μπορεί να περιλαμβάνουν τα στάδια αυτά σε διαφορετική σειρά, ή να αφιερώνουν περισσότερο ή λιγότερο χρόνο σε διαφορετικά στάδια. Ο βαθμός λεπτομέρειας του κάθε σταδίου επίσης μπορεί να ποικίλλει. Τα στάδια δύνανται να διεξάγονται με τη σειρά (μια προσέγγιση που βασίζεται στο μοντέλο «καταρράκτη», “waterfall”), ή να επαναλαμβάνονται επί διαφόρων κύκλων ή επαναλήψεων (μια πιο ευέλικτη προσέγγιση, agile). Μια ευέλικτη προσέγγιση περιλαμβάνει συνήθως λιγότερο χρόνο για το σχεδιασμό και την τεκμηρίωση, και περισσότερο χρόνο που δαπανάται για την κωδικοποίηση και την ανάπτυξη των αυτοματοποιημένων ελέγχων. Οι ακόμα περισσότερο ακραίες προσεγγίσεις προωθούν, επίσης, τον συνεχή έλεγχο καθ’ όλη τη διάρκεια του κύκλου ζωής της ανάπτυξης, καθώς επίσης διασφαλίζουν ένα λειτουργικό προϊόν (ή bug-free) ανά πάσα στιγμή. Αντίθετα, οι περισσότερο δομημένες προσεγγίσεις ή οι τύπου "καταρράκτη" επιχειρούν να αξιολογήσουν και να αντιμετωπίσουν την πλειοψηφία των κινδύνων και στη συνέχεια να αναπτύξουν ένα λεπτομερές σχέδιο για το λογισμικό πριν από την κατασκευή του λογισμικού (κωδικοποίηση), με σκοπό να αποφεύγονται σημαντικές αλλαγές στο σχεδιασμό και την εκ νέου κωδικοποίηση σε μεταγενέστερα στάδια του κύκλου ζωής της ανάπτυξης λογισμικού. (“Choosing an Appropriate System Development Methodology - electingDevelopmentApproach.pdf,” n.d.)

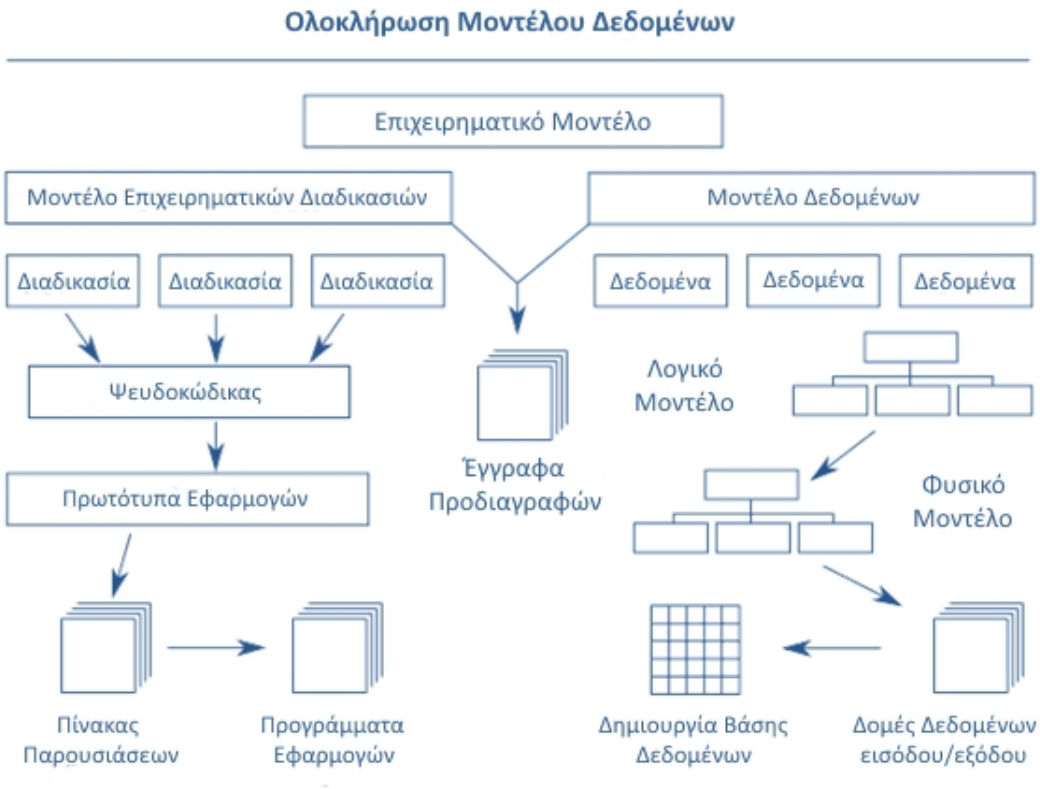
Υπάρχουν σημαντικά πλεονεκτήματα και μειονεκτήματα των διαφόρων μεθοδολογιών, και η επιλογή της καλύτερης προσέγγισης για την επίλυση ενός προβλήματος εξαρτάται συνήθως και από το είδος του προβλήματος. Γενικώς, αν το πρόβλημα είναι καλά κατανοητό και η λύση μπορεί να προγραμματιστεί αποτελεσματικά στο μέλλον, η προσέγγιση τύπου καταρράκτη είναι προτιμητέα. Αν, από την άλλη πλευρά, το πρόβλημα είναι μοναδικό (τουλάχιστον στην ομάδα ανάπτυξης) και η δομή της λύσης δεν αντιμετωπίζεται εύκολα, τότε μια ακραία προσέγγιση μπορεί να λειτουργήσει καλύτερα. (Whitten, 1998)

Οι μεθοδολογίες θα αναλυθούν περισσότερο στο επόμενο κεφάλαιο.

## 2.3.Βοηθητικά Εργαλεία Ανάπτυξης Λογισμικού

### 2.3.1. Επιχειρησιακή Μοντελοποίηση και Μοντελοποίηση Δεδομένων

Η Γραφική αναπαράσταση της τρέχουσας κατάστασης των πληροφοριών παρέχει ένα πολύ υποστηρικτικό μέσο για την παρουσίαση πληροφοριών για τους χρήστες και τους προγραμματιστές του συστήματος.




**ΔΙΑΓΡΑΜΜΑ 24 Αλληλεπίδραση μεταξύ των επιχειρηματικών διαδικασιών και μοντελοποίησης δεδομένων (Paulk, 1995)**

Στο παραπάνω διάγραμμα:

- Το επιχειρηματικό μοντέλο απεικονίζει τις λειτουργίες που σχετίζονται με την επιχειρηματική διαδικασία και τις οργανωτικές δομές που εκτελούν αυτές τις λειτουργίες. Με την απεικόνιση των δραστηριοτήτων και της ροής πληροφοριών, δημιουργείται μια βάση που απεικονίζει, ορίζει, καθιστά κατανοητή και επικυρώνει τη διαδικασία. Η διαδικασία αυτή αποτελεί αντικείμενο της επιχειρησιακής ανάλυσης.
- Το μοντέλο δεδομένων παρέχει τις λεπτομέρειες των πληροφοριών που πρέπει να εισάγονται, να επεξεργάζονται και να αποθηκεύονται, και είναι πρωταρχικής σημασίας για την εφαρμογή ή την προετοιμασία των

λειτουργικών προδιαγραφών για την υποβοήθηση της απόφασης για την αγορά ή την δημιουργία του λογισμικού. (Paulk, 1995)

### 2.3.2. Μηχανική λογισμικού με τη βοήθεια υπολογιστών (CASE)

Η μηχανική λογισμικού με τη βοήθεια υπολογιστών (Computer-aided software engineering Computer-aided, CASE), είναι η επιστημονική εφαρμογή μιας σειράς εργαλείων και μεθόδων ανάπτυξης ενός λογισμικού που οδηγεί σε υψηλής ποιότητας, χωρίς ελαττώματα και διατηρήσιμα προϊόντα λογισμικού. Αναφέρεται, επίσης, σε μεθόδους για την υλοποίηση των πληροφοριακών συστημάτων μαζί με αυτοματοποιημένα εργαλεία που μπορούν να χρησιμοποιηθούν στη διαδικασία ανάπτυξης λογισμικού. Ο όρος μηχανική λογισμικού με τη βοήθεια υπολογιστή (CASE) μπορεί να αναφέρεται στο λογισμικό που χρησιμοποιείται για την αυτοματοποιημένη ανάπτυξη συστημάτων λογισμικού. Οι λειτουργίες CASE περιλαμβάνουν την ανάλυση, το σχεδιασμό και τον προγραμματισμό. Τα εργαλεία CASE αυτοματοποιούν τις μεθόδους για το σχεδιασμό, την τεκμηρίωση και την παραγωγή δομημένου κώδικα στην επιθυμητή γλώσσα προγραμματισμού.

### 2.3.3. Ολοκληρωμένο περιβάλλον ανάπτυξης (IDE)

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (integrated development environment, IDE) είναι μία σουίτα λογισμικού που βοηθάει στην ανάπτυξη προγραμμάτων υπολογιστή. Συνήθως ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) περιλαμβάνει κάποιον επεξεργαστή πηγαίου κώδικα, έναν μεταγλωττιστή, εργαλεία αυτόματης παραγωγής κώδικα, αποσφαλματωτή, συνδέτη, σύστημα ελέγχου εκδόσεων και εργαλεία κατασκευής γραφικών διεπαφών χρήστη για τις υπό ανάπτυξη εφαρμογές. (Myatt, 2007)

Ένας από τους στόχους του IDE είναι να μειωθεί η απαραίτητη παραμετροποίηση για τη σύνδεση των διαφόρων βοηθητικών προγραμμάτων ανάπτυξης, αντί να παρέχει το σύνολο των δυνατοτήτων ως συνεκτική μονάδα. Η αντίστοιχη μείωση του χρόνου συνεπάγεται την αύξηση της παραγωγικότητας του έργου, σε περιπτώσεις όπου η εκμάθηση της χρήσης του IDE είναι ταχύτερη από ότι η με το χέρι ενσωμάτωση όλων των επιμέρους εργαλείων. Η στενότερη ολοκλήρωση όλων των εργασιών ανάπτυξης έχει τη δυνατότητα να βελτιώσει τη συνολική παραγωγικότητα πέρα από την απλή παροχή βοήθειας με τις εργασίες παραμετροποίησης. Για παράδειγμα, ο κώδικας μπορεί να αναλύεται συνεχώς ενώ γίνεται επεξεργασία, παρέχοντας άμεση πληροφόρηση όταν εισάγονται συντακτικά λάθη. Αυτό μπορεί να επιταχύνει την εκμάθηση μιας νέας γλώσσας προγραμματισμού και των συναφών βιβλιοθηκών.

Κάποια IDEs είναι αφιερωμένα σε μια συγκεκριμένη γλώσσα προγραμματισμού, επιτρέποντας ένα σύνολο χαρακτηριστικών που ταιριάζει περισσότερο στα πρότυπα προγραμματισμού της συγκεκριμένης γλώσσας. Ωστόσο, υπάρχουν πολλά ολοκληρωμένα περιβάλλοντα ανάπτυξης σε πολλές γλώσσες, όπως το Eclipse, ActiveState Komodo, IntelliJ IDEA, MyEclipse, Oracle JDeveloper, NetBeans, Codenvy και το Microsoft Visual Studio, Xcode, Xojo και το Delphi, που είναι αφιερωμένα σε μία μεμονωμένη γλώσσα ή ένα συγκεκριμένο σύνολο γλωσσών προγραμματισμού. (Kline and Seffah, 2005)

### 2.3.4. Γλώσσες μοντελοποίησης (Modeling Languages)

Μια γλώσσα μοντελοποίησης είναι οποιαδήποτε τεχνητή γλώσσα που μπορεί να χρησιμοποιηθεί για να εκφράσει πληροφορία ή γνώση ή δομημένα συστήματα, η οποία ορίζεται από ένα συνεκτικό σύνολο κανόνων, που χρησιμοποιούνται για την ερμηνεία της έννοιας των συστατικών της δομής. Μια γλώσσα μοντελοποίησης μπορεί να είναι γραφικών ή κειμένου. Οι γραφικές γλώσσες μοντελοποίησης χρησιμοποιούν διαγραμματικές τεχνικές με σύμβολα που αντιπροσωπεύουν έννοιες και γραμμές που συνδέουν τα σύμβολα και αντιπροσωπεύουν τις μεταξύ τους σχέσεις και διάφορα άλλα γραφικά σχόλια που αναπαριστούν τους εκάστοτε περιορισμούς. Οι γλώσσες μοντελοποίησης κειμένου χρησιμοποιούν συνήθως τυποποιημένες λέξεις-κλειδιά που συνοδεύονται από παραμέτρους ώστε οι εκφράσεις να είναι ερμηνεύσιμες από τον υπολογιστή. (Booch, 2005)

Στο γνωστικό πεδίο της μηχανικής λογισμικού, οι γλώσσες μοντελοποίησης λογισμικού ονομάζονται τυποποιημένες γλώσσες οπτικής συμβολικής αναπαράστασης με τις οποίες κατασκευάζονται αφαιρετικά μοντέλα της δομής και της λειτουργίας ενός συστήματος λογισμικού. Η ευρύτερα διαδεδομένη τέτοια γλώσσα είναι η Unified Modeling Language (UML).

Μετά την ευρεία εξάπλωση του αντικειμενοστρεφούς προγραμματισμού κατά τη δεκαετία του '90, το αντικειμενοστρεφές μοντέλο σχεδίασης (με κλάσεις, κληρονομικότητα, αντικείμενα και τυποποιημένες αλληλεπιδράσεις μεταξύ τους) επικράτησε ακόμη και για μοντελοποίηση που δεν περιελάμβανε προγραμματισμό (π.χ. σχήματα βάσεων δεδομένων). Έτσι αναπτύχθηκαν διάφορες πρότυπες γλώσσες μοντελοποίησης λογισμικού οι οποίες τυποποιούσαν οπτικά σύμβολα και συμπεριφορές με στόχο την αφαιρετική περιγραφή της λειτουργίας και της δομής ενός υπολογιστικού συστήματος. Οι γλώσσες αυτές είχαν εξαρχής έναν εμφανή αντικειμενοστρεφή προσανατολισμό. Τελικά, οι πιο δημοφιλείς από αυτές ενοποιήθηκαν στο κοινό πρότυπο UML που η πρώτη του έκδοση οριστικοποιήθηκε το 1997. (Booch, 2005)

Παραδείγματα γραφικών γλωσσών μοντελοποίησης στον τομέα της μηχανικής λογισμικού είναι:

- Business Process Modeling Notation (BPMN, και η XML μορφή της BPML) είναι ένα παράδειγμα μιας γλώσσας μοντελοποίησης διαδικασιών.
- EXPRESS και EXPRESS-G (ISO 10303 -11) είναι ένα διεθνές πρότυπο γλώσσας γενικής χρήσης για μοντελοποίηση δεδομένων.
- Extended Enterprise Modeling Language (EEML) χρησιμοποιείται συνήθως για μοντελοποίηση επιχειρησιακών διαδικασιών διαφορετικών επιπέδων.
- Το Διάγραμμα ροής είναι μία σχηματική αναπαράσταση ενός αλγορίθμου ή μιας βαθμιδωτής μεθόδου.
- Fundamental Modeling Concepts (FMC) είναι μια γλώσσα μοντελοποίησης για συστήματα λογισμικού.
- IDEF είναι μια οικογένεια γλωσσών μοντελοποίησης, οι πλέον αξιοσημείωτες εκ των οποίων είναι η IDEF0 για τη λειτουργική μοντελοποίηση, η IDEF1X για μοντελοποίηση της πληροφορίας, και η IDEF5 για μοντελοποίηση οντολογιών.
- LePUS3 είναι μια αντικειμενοστραφής οπτικού σχεδιασμού περιγραφική γλώσσα και μια γλώσσα επίσημων προδιαγραφών, κατάλληλη κυρίως για τη μοντελοποίηση μεγάλων αντικειμενοστραφών (Java, C++, C#) προγραμμάτων και σχεδιαστικών προτύπων.
- Specification and Description Language (SDL) είναι μια γλώσσα προδιαγραφών που απευθύνεται στο σαφή προσδιορισμό και στην περιγραφή της συμπεριφοράς των διαδραστικών και κατανεμημένων συστημάτων.
- Unified Modeling Language (UML) είναι μια γλώσσα μοντελοποίησης γενικού σκοπού και αποτελεί ένα βιομηχανικό πρότυπο για τον καθορισμό των συστημάτων λογισμικού. Υποστηρίζει διαφορετικές διαγραμματικές τεχνικές, και έχει ευρεία υποστήριξη.

Δεν είναι όλες οι γλώσσες μοντελοποίησης εκτελέσιμες, ενώ εκείνες που είναι δεν σημαίνει ότι απαραίτητα αντικαθιστούν τους προγραμματιστές. Αντιθέτως, οι εκτελέσιμες γλώσσες μοντελοποίησης προορίζονται για την ενίσχυση της παραγωγικότητας των ειδικευμένων προγραμματιστών, έτσι ώστε να μπορούν να αντιμετωπίσουν πιο δύσκολα προβλήματα, όπως παράλληλα και κατανεμημένα συστήματα.

#### 2.3.4.1. Unified Modeling Language UML

Η Unified Modeling Language (UML, αδόκιμη απόδοση στην Ελληνική γλώσσα: Ενοποιημένη Γλώσσα Μοντελοποίησης) είναι πλέον η πρότυπη γλώσσα μοντελοποίησης στη μηχανική λογισμικού. Χρησιμοποιείται για τη γραφική απεικόνιση, προσδιορισμό, κατασκευή και τεκμηρίωση των στοιχείων ενός συστήματος λογισμικού. Μπορεί να χρησιμοποιηθεί σε διάφορες φάσεις ανάπτυξης, από την ανάλυση απαιτήσεων ως τον έλεγχο ενός ολοκληρωμένου συστήματος.

Αποτελείται από ένα σύνολο προσυμφωνημένων όρων, συμβόλων και διαγραμμάτων που επιτρέπουν:

- την εμφάνιση των ορίων ενός συστήματος και των βασικών λειτουργιών του, χρησιμοποιώντας περιπτώσεις χρήσης (use-cases) και εμπλεκόμενους (actors).
- την επεξήγηση της πραγματοποίησης των περιπτώσεων χρήσης με διαγράμματα αλληλεπίδρασης.
- την αναπαράσταση μιας στατικής δομής ενός συστήματος χρησιμοποιώντας διαγράμματα κλάσεων.
- τη μοντελοποίηση της συμπεριφοράς των αντικειμένων με διαγράμματα καταστάσεων.
- τη μοντελοποίηση της εργασιακής ροής με διαγράμματα δραστηριοτήτων.
- την αποκάλυψη της υλοποίησης της αρχιτεκτονικής με διαγράμματα συστατικών και ανάπτυξης.
- την επέκταση της λειτουργικότητας με στερεότυπα. (Booch, 2005)

### 2.3.5. Πρότυπο προγραμματισμού

Το πρότυπο προγραμματισμού είναι μια θεμελιώδης ύφος-τεχνική του προγραμματισμού ηλεκτρονικών υπολογιστών, με το οποίο ένα υπολογιστικό πρόβλημα και η αλγοριθμική λύση του προσεγγίζονται με συγκεκριμένες μεθόδους. Είναι δηλαδή ένα σύνολο εννοιών, οι οποίες εκφράζουν έναν συγκεκριμένο τρόπο σκέψης, και κατά συνέπεια έκφρασης της υλοποίησης, και διαμορφώνουν τον τρόπο σχεδιασμού ενός προγράμματος, ο οποίος γενικά δεν υπαγορεύεται από τη μεθοδολογία διαχείρισης του έργου όπως οι waterfall ή agile. (Wells, 2008)

Ένα πρότυπο περιγράφει ένα επαναλαμβανόμενο πρόβλημα, σε ένα συγκεκριμένο περιβάλλον, και μια καλά τεκμηριωμένη λύση του, με όφελος την επαναχρησιμοποίηση σχεδίων, την επαναχρησιμοποίηση κώδικα και τη διευκόλυνση της διαχείρισης των αλλαγών.

Τα πρότυπα διαφέρουν μεταξύ τους ως προς τις έννοιες και τις αφαιρέσεις που χρησιμοποιούνται για την αναπαράσταση των στοιχείων του προγράμματος (όπως τα αντικείμενα, τις λειτουργίες, τις μεταβλητές, τους περιορισμούς), καθώς και τα βήματα που περιλαμβάνει ένας υπολογισμός (όπως τις αναθέσεις, την αξιολόγηση, τις συνέχειες, τις ροές δεδομένων). Συχνά, οι έννοιες του προτύπου χρησιμοποιούνται συνεργατικά σε υψηλού επιπέδου σχεδιασμό αρχιτεκτονικής του συστήματος, ενώ σε άλλες περιπτώσεις, το πεδίο εφαρμογής του πρότυπου περιορίζεται στην εσωτερική δομή ενός συγκεκριμένου προγράμματος ή τμήματος.

Το πρότυπο ορίζεται ως μία αποδεδειγμένα καλή λύση που έχει εφαρμοστεί με επιτυχία στην επίλυση ενός επαναλαμβανόμενου προβλήματος σχεδίασης συστημάτων λογισμικού. Τα πρότυπα σχεδίασης ορίζονται τόσο σε επίπεδο μακροσκοπικής σχεδίασης όσο και σε επίπεδο υλοποίησης, ενώ με τη χρήση τους ένας προγραμματιστής αντικαθιστά πρακτικώς μεγάλα τμήματα του απαιτούμενου κώδικα με μαύρα κουτιά. Πρόκειται για αφαιρέσεις υψηλού επιπέδου που αποτελούν πλήρη υποσυστήματα, καταλλήλως ρυθμισμένα για την επίλυση συγκεκριμένων προβλημάτων σχεδίασης λογισμικού και έτοιμα για χρήση. (Gamma, 1995)

Μια γλώσσα προγραμματισμού μπορεί να υποστηρίξει πολλαπλά πρότυπα. Συνήθως τα σχεδιαστικά πρότυπα κατηγοριοποιούνται σε κατασκευαστικά (creational), δομικά (structural) και συμπεριφορικά (behavioral). Ένας άλλος διαχωρισμός των προτύπων που συναντιέται συχνά αναφέρεται σε προστακτικό (imperative), δηλωτικό (declarative), λειτουργικό (functional), αντικειμενοστραφή (object-oriented), διαδικαστικό (procedural), λογικό (logic) και συμβολικό (symbolic) προγραμματισμό. Υπάρχουν και άλλου τύπου διαχωρισμοί και επίπεδα περαιτέρω ανάλυσης.

Ακριβώς όπως η τεχνολογία λογισμικού (ως διαδικασία) ορίζεται από τις διαφορετικές μεθοδολογίες, έτσι οι γλώσσες προγραμματισμού (ως υπολογιστικά μοντέλα) ορίζονται από διαφορετικά πρότυπα. Μερικές γλώσσες έχουν σχεδιαστεί για να υποστηρίζουν ένα συγκεκριμένο πρότυπο (π.χ. η Smalltalk υποστηρίζει αντικειμενοστραφή προγραμματισμό, η Haskell υποστηρίζει λειτουργικό προγραμματισμό), ενώ άλλες γλώσσες προγραμματισμού υποστηρίζουν πολλαπλά πρότυπα (όπως οι Object Pascal, C++, Java, C#, Scala, Visual Basic, Common Lisp, Scheme, Perl, Python, Ruby, Oz και F#). (Scott, 2009)

### 2.3.6. Πλαίσιο Λογισμικού

Το πλαίσιο λογισμικού (software framework) είναι ένα επαναχρησιμοποιήσιμο σχέδιο για ένα λογισμικό σύστημα ή υποσύστημα. Το πλαίσιο λογισμικού μπορεί να περιλαμβάνει προγράμματα στήριξης, βιβλιοθήκες κώδικα, βοηθητικά εργαλεία, διεπαφές προγραμματισμού εφαρμογών (application programming interfaces (APIs)), γλώσσες δέσμης ενεργειών (scripting language), ή άλλο λογισμικό για να βοηθήσουν στην ανάπτυξη και στη σύνδεση των διαφόρων συνιστωσών ενός λογισμικού έργου. (Greenfield, 2004)

Το πλαίσιο λογισμικού είναι μια αφηρημένη έννοια στην οποία υπάρχον λογισμικό, που παρέχει γενικές λειτουργίες, τροποποιείται επιλεκτικά με την εγγραφή ενός πρόσθετου κώδικα, παρέχοντας έτσι νέο κατάλληλα τροποποιημένο λογισμικό για την υπό κατασκευή εφαρμογή. Το πλαίσιο λογισμικού είναι ένα καθολικό και επαναχρησιμοποιήσιμο περιβάλλον λογισμικού που παρέχει ιδιαίτερη

λειτουργικότητα ως μέρος μιας μεγαλύτερης πλατφόρμας λογισμικού για να διευκολύνει την ανάπτυξη του λογισμικού εφαρμογών, προϊόντων και λύσεων. (Riehle, 2000)

# Κεφάλαιο 3

## 3. Μεθοδολογίες Ανάπτυξης Λογισμικού

Η μεθοδολογία ανάπτυξης λογισμικού (επίσης γνωστή ως κύκλος ζωής της ανάπτυξης λογισμικού ή ως διαδικασία ανάπτυξης λογισμικού) είναι μια κατηγοριοποίηση του συνολικού έργου της ανάπτυξης λογισμικού σε διακριτές φάσεις (ή στάδια) που περιέχουν δραστηριότητες, με σκοπό τον καλύτερο προγραμματισμό και την βελτιστοποίηση της διαχείρισης. Συχνά, θεωρείται υποσύνολο του κύκλου ζωής των συστημάτων ανάπτυξης. ("Choosing An Appropriate System Development Methodology - SelectingDevelopmentApproach.pdf," n.d.)

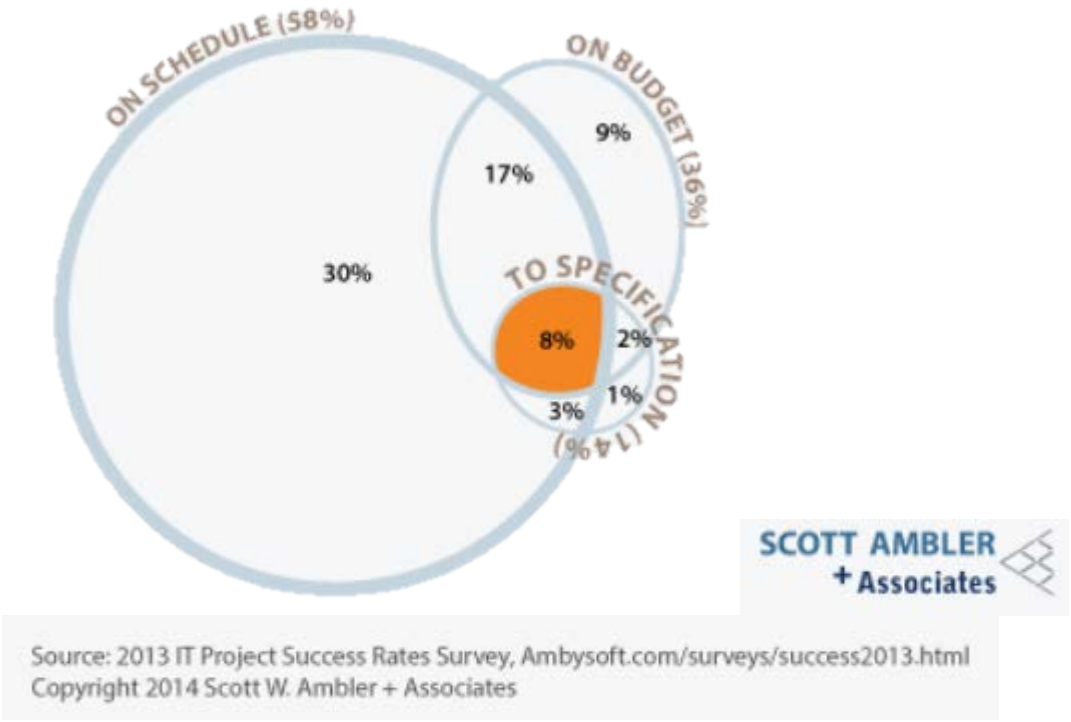
Μερικές από τις πιο γνωστές μεθοδολογίες αποτελούν οι ακόλουθες:

- καταρράκτη (waterfall),
- προτυποποίηση ( prototyping) ,
- επαναληπτική (iterative) και αυξητική (incremental) ανάπτυξη,
- ανάπτυξη σπείρας (spiral development),
- η ταχεία ανάπτυξη εφαρμογών (rapid application development),
- ακραία προγραμματισμού (extreme programming) και
- ευέλικτη μεθοδολογία (agile methodology).

Ο όρος «μοντέλο» του κύκλου ζωής θεωρείται πιο γενικός όρος για τις περισσότερες κατηγορίες των μεθοδολογιών, ενώ ο όρος «διαδικασία» ανάπτυξης λογισμικού ένας πιο εξειδικευμένος όρος. ("Choosing an Appropriate System Development Methodology - SelectingDevelopmentApproach.pdf," n.d.)

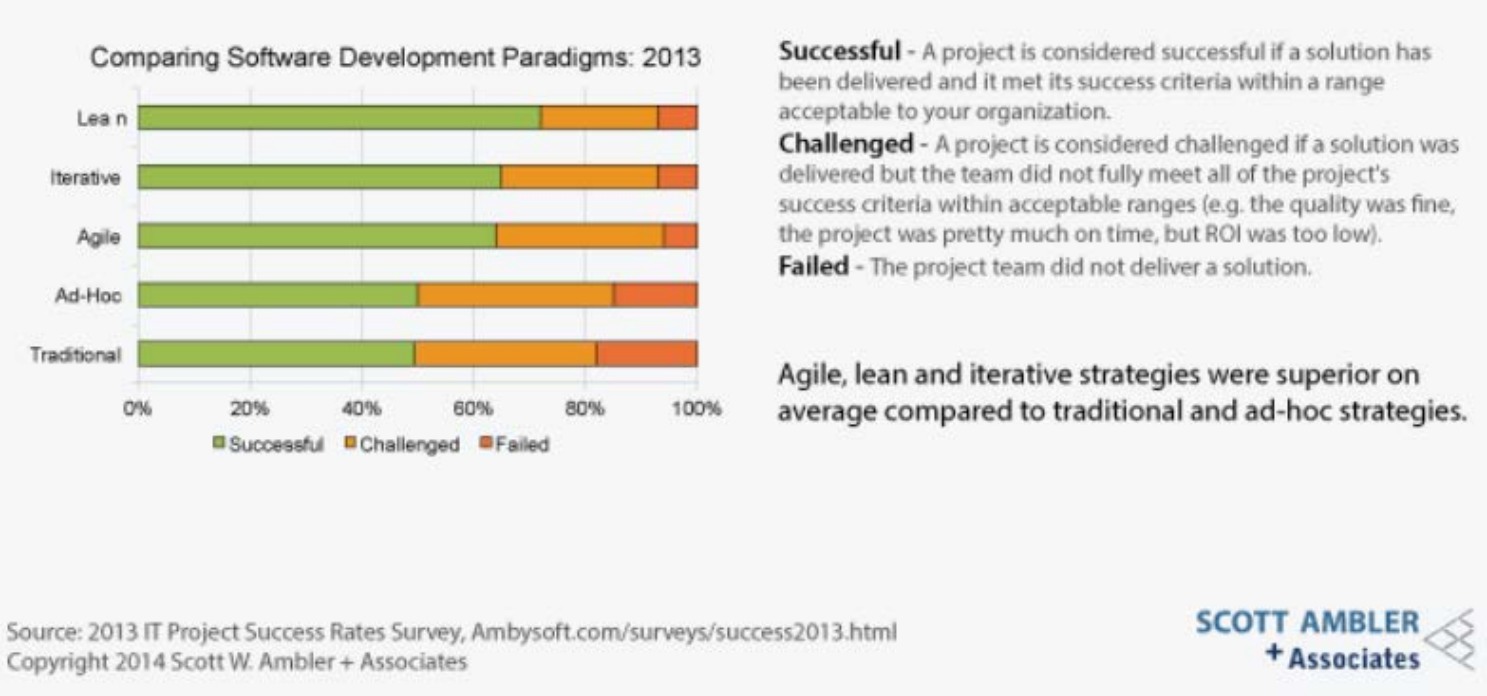
Οι μεθοδολογίες έχουν εξελιχθεί με την πάροδο των ετών, η κάθε μια με τα δικά της αναγνωρισμένα πλεονεκτήματα και αδυναμίες. Δεν υπάρχει μια μεθοδολογία που να είναι κατάλληλη για να χρησιμοποιηθεί σε όλα τα έργα. Κάθε μια από τις διαθέσιμες μεθοδολογίες είναι καταλληλότερη για κάποια συγκεκριμένα είδη των έργων, με γνώμονα ζητήματα τεχνικά, οργανωτικά, σε σχέση με το έργο και την ομάδα. ("SDLC Quick Guide," n.d.)

Οι μεθοδολογίες αυτές έχουν να κάνουν με τεχνικές διαχείρισης έργων για το σχεδιασμό του λογισμικού. Χωρίς την αποτελεσματική διαχείριση του έργου, τα έργα λογισμικού μπορεί εύκολα να ολοκληρωθούν αργότερα από την προκαθορισμένη ημερομηνία παράδοσης ή να ξεφύγουν πέρα από τον αρχικό προϋπολογισμό. Ο ορισμός της επιτυχίας ανάπτυξης ενός λογισμικού έργου αφορά την παράδοσή του εντός προγραμματισμένου χρόνου, προγραμματισμένου κόστους και τήρησης των προδιαγραφών. Λόγω του εξαιρετικά πολύπλοκου περιβάλλοντος των λογισμικών έργων, με συνεχείς αλλαγές στις ανάγκες των πελατών και των τεχνολογικών εξελίξεων, η αποδοχή της επιτυχίας της ανάπτυξής τους, πολλές φορές γίνεται ελαστικότερη, καλύπτοντας ένα συνδυασμό των ανωτέρω, όπως φαίνεται στο ακόλουθο διάγραμμα.




**ΔΙΑΓΡΑΜΜΑ 25 ΠΑΡΑΜΕΤΡΟΙ ΕΠΙΤΥΧΙΑΣ ΕΝΟΣ ΛΟΓΙΣΜΙΚΟΥ ΕΡΓΟΥ ("2013 IT Project Success Rates Survey Results," n.d.)**

Ως προς την επιτυχία των μεθοδολογιών , σε σχέση με τον παραπάνω ορισμό της επιτυχίας τους, σχηματικά δίνεται στο ακόλουθο διάγραμμα, στο οποίο απεικονίζονται τα αντίστοιχα ποσοστά των υπό έρευνα μεθοδολογιών.




**ΔΙΑΓΡΑΜΜΑ 26 ΑΠΟΔΟΤΙΚΟΤΗΤΑ ΔΙΑΦΟΡΩΝ ΜΕΘΟΔΟΛΟΓΙΩΝ ("2013 IT Project Success Rates Survey Results," n.d.)**

Επειδή ένας μεγάλος αριθμός έργων λογισμικού δεν ανταποκρίνεται στις αρχικές προσδοκίες όσον αφορά τη λειτουργικότητα, το κόστος ή το χρονοδιάγραμμα παράδοσης, γίνεται ακόμα πιο επιτακτική η ανάγκη για αποτελεσματική διαχείριση του έργου.

## 3.1. Σύντομη Ιστορική Αναδρομή

Οι πρώτες μεθοδολογίες ανάπτυξης λογισμικού εμφανίστηκαν τη δεκαετία του 1960. Η μεθοδολογία του κύκλου ζωής της ανάπτυξης συστημάτων (SDLC) μπορεί να θεωρηθεί ότι είναι η παλαιότερη επίσημη μεθοδολογία για τη δημιουργία πληροφοριακών συστημάτων. Η κύρια ιδέα της SDLC είναι η συνεχής επιδίωξη της ανάπτυξης των πληροφοριακών συστημάτων με ένα σχεδιασμένο, δομημένο και μεθοδικό τρόπο, επιβάλλοντας ότι όλα τα στάδια του κύκλου ζωής από την σύλληψη της ιδέας μέχρι την παράδοση του τελικού συστήματος, να διενεργούνται με τρόπο συμπαγή και διαδοχικό, εντός της μεθοδολογίας που εφαρμόζεται. Ο κύριος στόχος αυτής της προσέγγισης στη δεκαετία του 1960 ήταν η ανάπτυξη μεγάλης κλίμακας λειτουργικών επιχειρησιακών συστημάτων, σε μια εποχή ομίλων επιχειρήσεων μεγάλης κλίμακας. Οι δραστηριότητες των πληροφοριακών συστημάτων περιστράφηκαν γύρω από βαριά επεξεργασία των δεδομένων και πλήθος διαφορετικών ρουτινών. (Elliott, 2004)

Ένας από τους βασικούς στόχους δεκαετιών ήταν να βρεθούν επαναλαμβανόμενες και προβλέψιμες διαδικασίες που να βελτιώνουν την παραγωγικότητα και την ποιότητα. Σημαντικές προσπάθειες έχουν γίνει για τη συστηματοποίηση και επισημοποίηση του φαινομενικά απείθαρχου έργου του σχεδιασμού λογισμικού, με τη μορφή μεθοδολογιών, σημαντικότερες από τις οποίες παρατίθενται παρακάτω με χρονική σειρά. (“Paper Title (use style: paper title) - V3I3-0198.pdf,” n.d.)

1970s

- Structured programming since 1969
- Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology

1980s

- Structured systems analysis and design method (SSADM) from 1980 onwards
- Information Requirement Analysis/Soft systems methodology

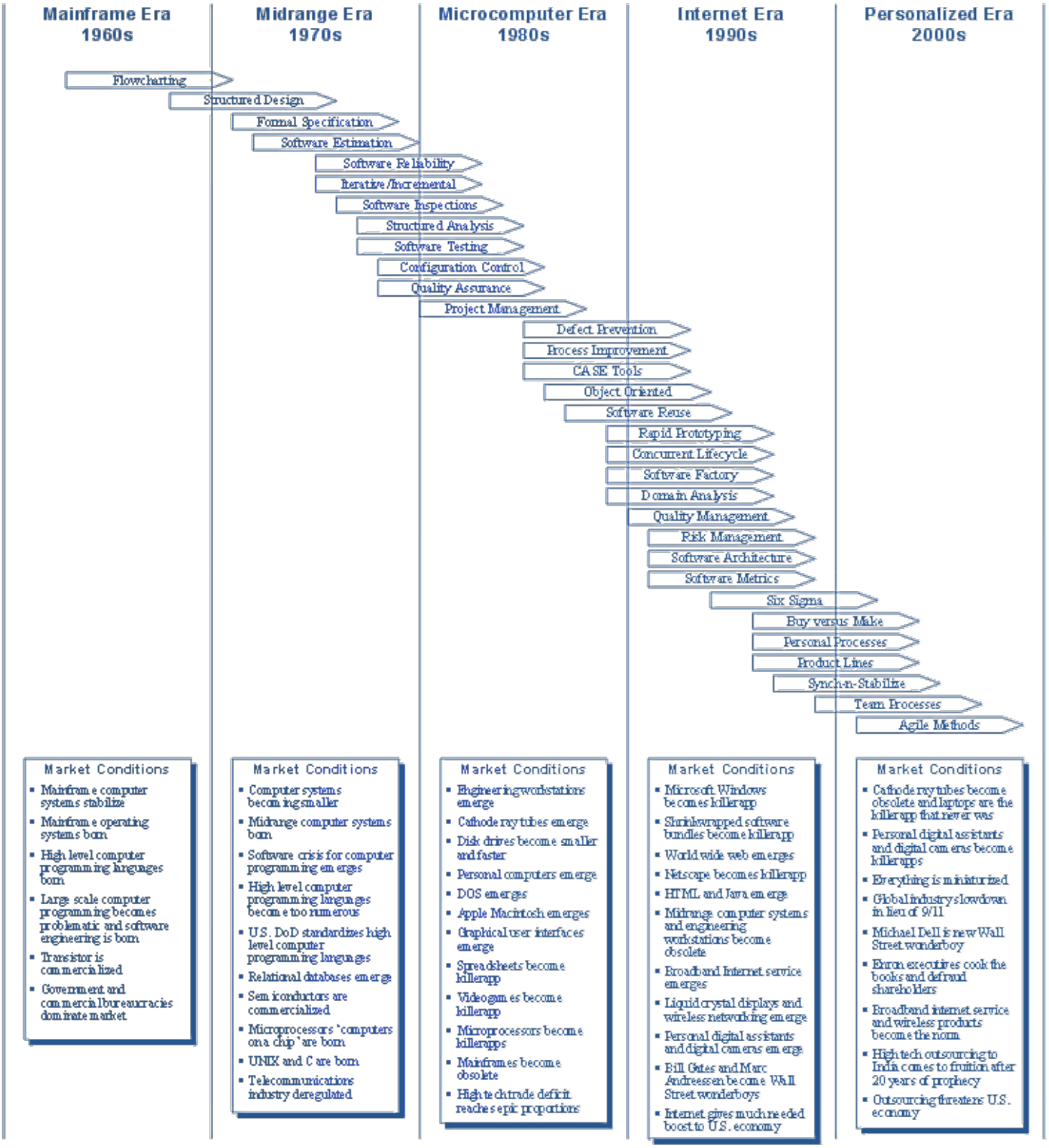
1990s

- Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s
- Rapid application development (RAD), since 1991
- Dynamic systems development method (DSDM), since 1994
- Scrum, since 1995
- Team software process, since 1998
- Rational Unified Process (RUP), maintained by IBM since 1998
- Extreme programming, since 1999

2000s

- Agile Unified Process (AUP) maintained since 2005 by Scott Ambler
- Integrated Methodology (QAIassist-IM) since 2007

Ακολουθεί ένα αναλυτικότερο διάγραμμα σχετικά με την εξελικτική πορεία των μεθοδολογιών, αλλά και των ευρύτερων συνθηκών αγοράς που επικρατούσαν ανά δεκαετία.



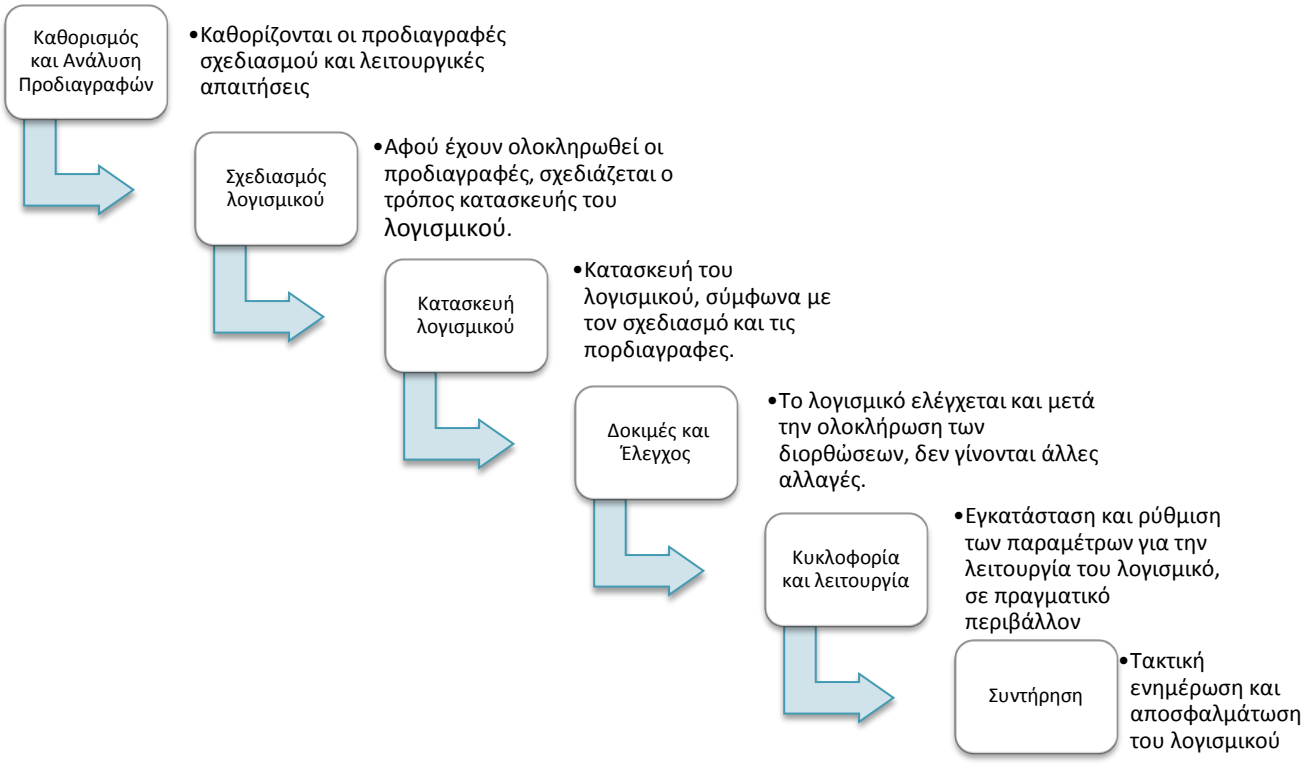

**ΔΙΑΓΡΑΜΜΑ 27 Χρονοδιάγραμμα των Μεθοδολογιών ανάπτυξης πληροφοριακών συστημάτων και των συνθηκών της αγοράς (Rico, 2010)**

Η κάθε μεθοδολογία ανάπτυξης λογισμικού έχει ως ένα βαθμό τη δική της προσέγγιση και διαφοροποίηση. Γενικώς οι μεθοδολογίες αυτές μπορούν να ταξινομηθούν σε δύο κύριες κατηγορίες, στις σειριακές ή γραμμικές, όπου η υλοποίηση του έργου γίνεται σε σειριακά διακριτά διαδοχικά βήματα, και στις επαναλαμβανόμενες ή κυκλικές, όπου η υλοποίηση γίνεται σε επάλληλα βήματα με επαναληπτικό ή/και παράλληλο τρόπο. Στην πράξη η ομάδα διαχείρισης ή ανάπτυξης του έργου επιλέγει μια μεθοδολογία ή έναν συνδυασμό αυτών.

Στη συνέχεια, θα παρουσιαστούν μερικές από τις κυριότερες μεθοδολογίες ανάπτυξης λογισμικού έργου.

## 3.2. Μεθοδολογία Καταρράκτη (Waterfall model)

Το μοντέλο καταρράκτη αποτελεί μια προσέγγιση σειριακών διαδοχικών φάσεων ανάπτυξης λογισμικού, στο οποίο η ανάπτυξη εκλαμβάνεται ως μια διαδικασία η οποία ρέει σταθερά (σαν καταρράκτης) διαμέσου επιμέρους δραστηριοτήτων. Κατά κανόνα περιλαμβάνει τις δραστηριότητες που φαίνονται στο παρακάτω διάγραμμα:




**ΔΙΑΓΡΑΜΜΑ 28 Μοντέλο καταρράκτη ("Waterfall Model | Definition and Concept | IT & Systems | MBA Skool-Study.Learn.Share.," n.d.)**

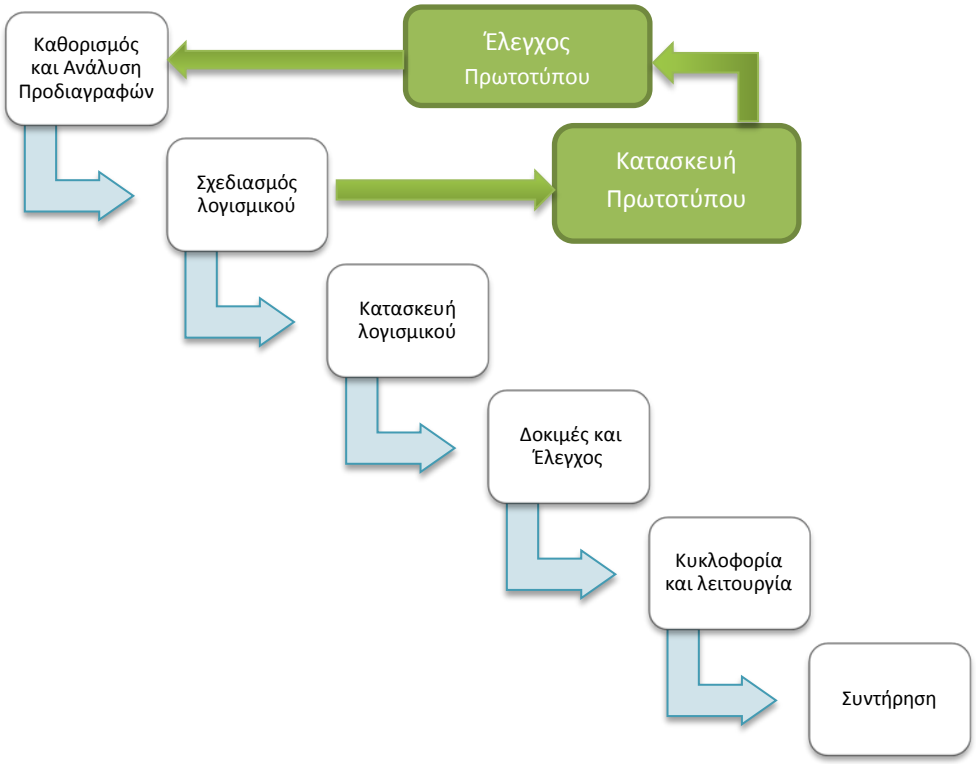
Η πρώτη επίσημη περιγραφή της μεθόδου συχνά αναφέρεται σε ένα άρθρο που δημοσιεύθηκε από τον W. W. Royce το 1970. Οι βασικές αρχές είναι οι εξής:

- Το έργο χωρίζεται σε διαδοχικές φάσεις, ενώ είναι αποδεκτή κάποια επικάλυψη και ανάδραση μεταξύ των διαδοχικών φάσεων.
- Έμφαση δίνεται σε θέματα σχεδιασμού, χρονοδιαγράμματα, ημερομηνίες-στόχους, προϋπολογισμούς και στην άπαξ υλοποίηση ολόκληρου του συστήματος.
- Ο αυστηρός έλεγχος διατηρείται κατά τη διάρκεια ζωής του έργου μέσω εκτεταμένης έγγραφης τεκμηρίωσης, επίσημων αξιολογήσεων, εγκρίσεων και προσυπογραφών από τους χρήστες, και με τον έλεγχο και παραλαβή του αποτελέσματος της κάθε φάσης πριν από την έναρξη της επόμενης.

Το μοντέλο καταρράκτη είναι μια παραδοσιακή προσέγγιση μηχανικής που εφαρμόζεται και στην τεχνολογία λογισμικού. Η αυστηρή προσέγγιση καταρράκτη αποθαρρύνει την επανεξέταση και αναθεώρηση κάθε προηγούμενης φάσης αφού αυτή έχει ολοκληρωθεί. Αυτή η ακαμψία σε ένα καθαρό μοντέλο καταρράκτη υπήρξε πηγή της κριτικής από τους οπαδούς πιο ευέλικτων μεθοδολογιών. Επίσης, ως μεθοδολογία έχει την τάση να ξεπερνά τον αρχικό προϋπολογισμό, μερικές φορές δεν καταφέρνει να καλύψει τις πραγματικές απαιτήσεις λόγω της μη αναθεώρησης των αρχικών προδιαγραφών. Εκτός από λίγες περιπτώσεις, όπου συμβατικά απαιτείται, το μοντέλο καταρράκτη έχει σε μεγάλο βαθμό αντικατασταθεί από πιο ευέλικτες και ευπροσάρμοστες μεθοδολογίες που έχουν αναπτυχθεί ειδικά για την ανάπτυξη λογισμικού.

## 3.3. Πρωτοτυποποίηση (Prototyping)

Η Πρωτοτυποποίηση λογισμικού είναι μια προσέγγιση σχετικά με την εξέλιξη των δραστηριοτήτων κατά τη διάρκεια της ανάπτυξης του λογισμικού, με τη δημιουργία πρωτοτύπων εφαρμογών λογισμικού, όπως πχ. ελλιπών εκδόσεων του λογισμικού που αναπτύσσεται.



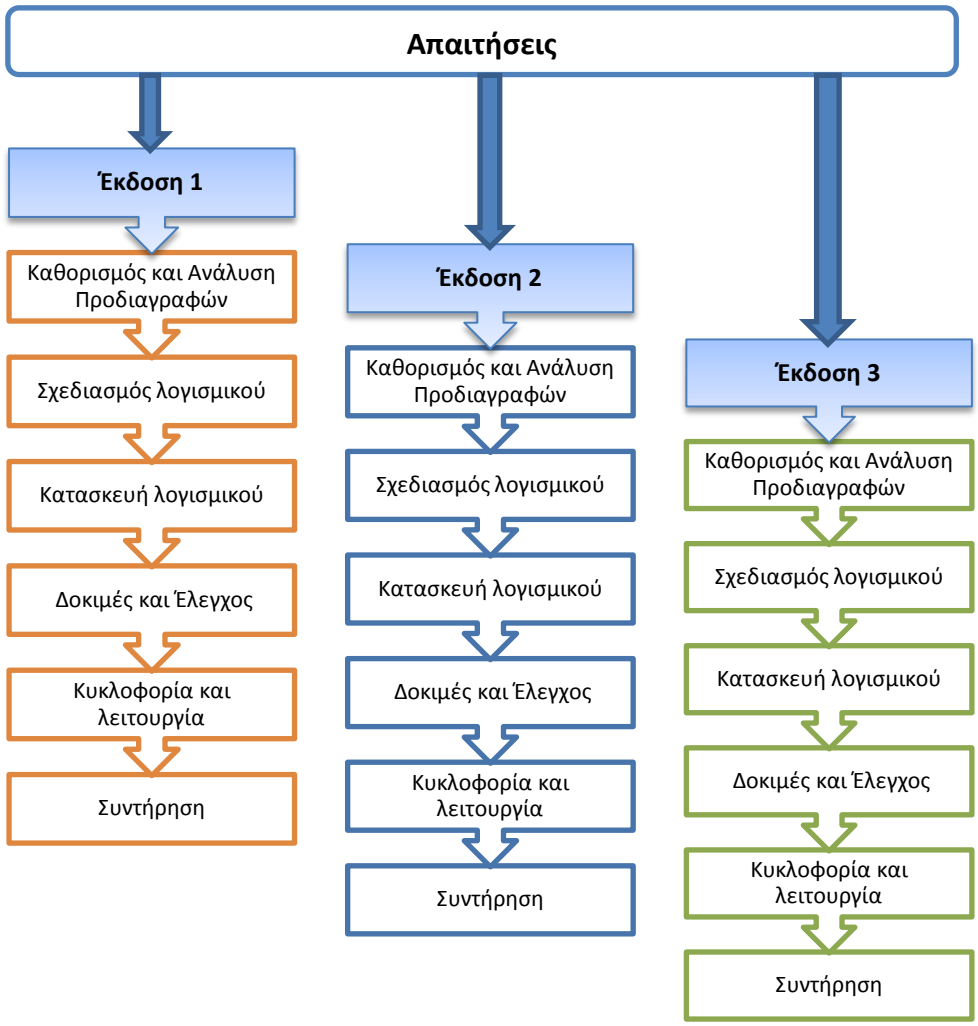

**ΔΙΑΓΡΑΜΜΑ 29 Μεθοδολογία πρωτοτυποποίησης ("Software Process Model — Prototyping Process Model," n.d.) , ("ΔΟΜΗΜΕΝΟΣ - DP Pangalos.pdf," n.d.)**

Οι βασικές αρχές είναι οι εξής:

- Δεν είναι μια αυτόνομη, πλήρης μεθοδολογία ανάπτυξης, αλλά περισσότερο μια προσέγγιση για τη διαχείριση συγκεκριμένων τμημάτων μιας ευρύτερης παραδοσιακής μεθοδολογίας ανάπτυξης.
- Στοχεύει στη μείωση των εγγενών κινδύνων του έργου με το διαχωρισμό ενός έργου σε μικρότερα τμήματα και την παροχή περισσότερων ευκολιών αλλαγής κατά τη διάρκεια της διαδικασίας ανάπτυξης.
- Οι χρήστες συμμετέχουν σε όλη τη διαδικασία ανάπτυξης, αυξάνοντας την πιθανότητα της αποδοχής της τελικής εφαρμογής από τους χρήστες.
- Κατασκευάζονται μικρής κλίμακας μακέτες του συστήματος μετά από μια επαναληπτική διαδικασία τροποποίησης μέχρι το πρωτότυπο να εξελιχθεί τόσο, ώστε να ανταποκρίνεται στις προδιαγραφές και τις απαιτήσεις των χρηστών.
- Είναι ένα σειριακό μοντέλο, που βασίζεται στη σχεδίαση πρωτότυπου, ακολουθούμενου από το μοντέλο Καταρράκτη (το πρωτότυπο δεν είναι το προϊόν και η πρωτοτυποποίηση δεν πρέπει να είναι μέρος της σχεδίασης, αλλά μόνο της συλλογής απαιτήσεων).

## 3.4. Αυξητικό μοντέλο (Incremental development)

Αρκετές μέθοδοι χρησιμοποιούν συνδυασμό γραμμικών και επαναληπτικών μεθοδολογιών ανάπτυξης συστημάτων, με πρωταρχικό στόχο τη μείωση των κινδύνων του έργου με το διαχωρισμό ενός έργου σε μικρότερα τμήματα και την διευκόλυνση πιθανών αλλαγών κατά τη διάρκεια της διαδικασίας ανάπτυξης.




**ΔΙΑΓΡΑΜΜΑ 30 Αυξητικό μοντέλο ("ΔΟΜΗΜΕΝΟΣ - DP Pangalos.pdf," n.d.)**

Οι βασικές αρχές είναι οι εξής:

- Κατ' ουσία εκτελείται μια σειρά από μικρούς καταρράκτες, όπου όλες οι φάσεις του καθενός ολοκληρώνονται για ένα μικρό μέρος του συστήματος και έπειτα ξεκινά ένας επόμενος επαυξημένος.

- Οι γενικές απαιτήσεις καθορίζονται πριν την έναρξη της εξελικτικής διαδικασίας των επιμέρους βημάτων μικρών-Καταρρακτών.
- Η αρχική ιδέα του λογισμικού, η ανάλυση απαιτήσεων, και ο σχεδιασμός της αρχιτεκτονικής και του πυρήνα του συστήματος καθορίζονται μέσω Καταρράκτη, που ακολουθείται από επαναληπτική μεθοδολογία Πρωτότυπων, η οποία κορυφώνεται με την κατασκευή του τελικού πρωτότυπου, ενός λειτουργικού λογισμικού.
- Η ανάπτυξη του λογισμικού γίνεται σε επάλληλες (διαδοχικές) εκδόσεις, όπου σε κάθε έκδοση του λογισμικού γίνεται προσθήκη νέων λειτουργιών και ποιοτικών χαρακτηριστικών, από ένα προκαθορισμένο σύνολο απαιτήσεων.

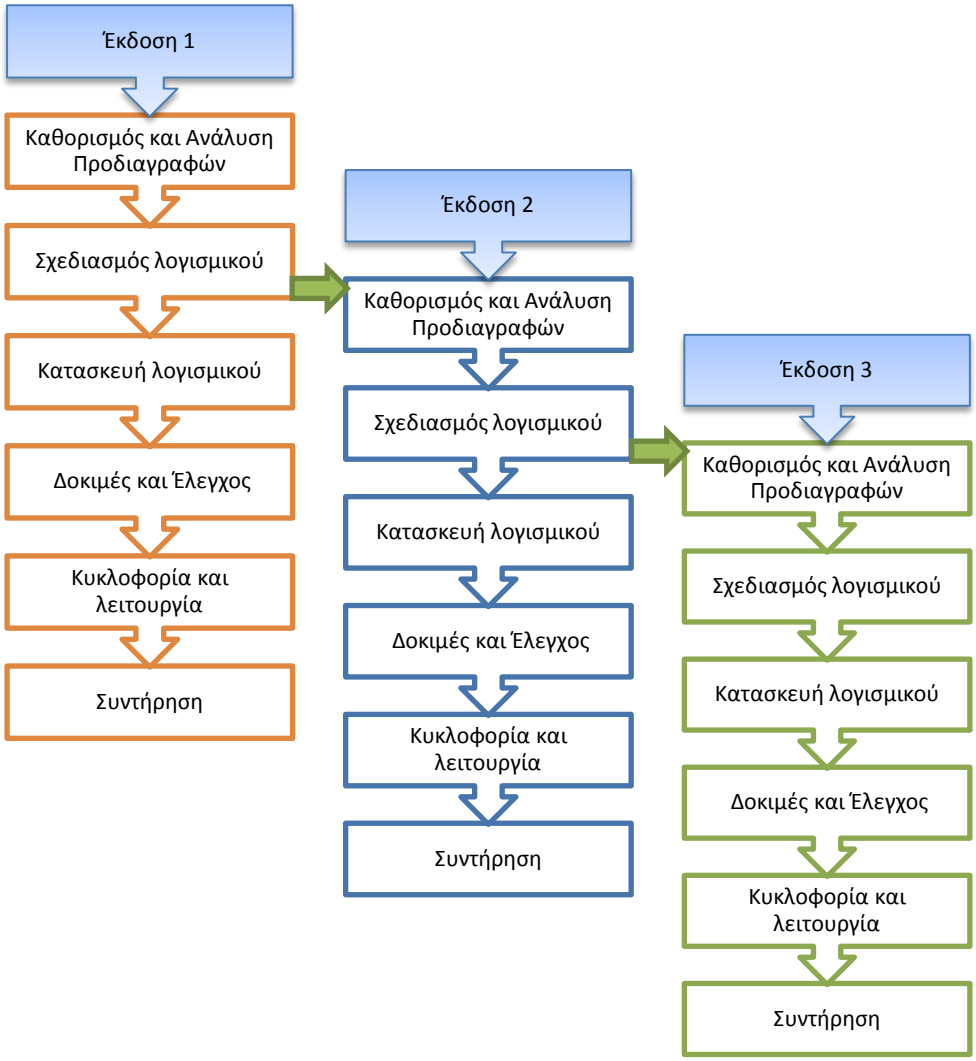
## 3.5. Εξελικτική μεθοδολογία ή Επαναληπτική και αυξητική (Iterative and incremental development)

Η επαναληπτική και αυξητική μεθοδολογία ανάπτυξης περιλαμβάνει οποιοδήποτε συνδυασμό επαναληπτικής μεθόδου και αυξητικού μοντέλου κατασκευής για την ανάπτυξη λογισμικού. Οι μέθοδοι αυτοί συχνά αναφέρονται και ως εξελικτικές.

Γενικώς, οι εν λόγω μεθοδολογίες αποτελούν τη βάση της Rational Unified Process, του Extreme Programming και γενικά των διάφορων ευέλικτων (agile) μεθοδολογιών ανάπτυξης λογισμικού.

Η επαναληπτική ανάπτυξη προβλέπει την κατασκευή αρχικά μικρών, αλλά σε κάθε επανάληψη μεγαλύτερων τμημάτων ενός έργου λογισμικού, για να βοηθήσει όλους τους εμπλεκόμενους να ανακαλύψουν εκ των προτέρων σημαντικά θέματα, πριν τα προβλήματα ή λανθασμένες υποθέσεις οδηγήσουν σε δυσμενείς συνθήκες. (Larman and Basili, 2003)

Η διαφορά της μεθοδολογίας αυτής σε σχέση με την απλή αυξητική έγκειται στο ότι η ανάπτυξη του λογισμικού γίνεται σε επάλληλες (διαδοχικές) εκδόσεις, όπου νέες εκδόσεις λογισμικού υλοποιούν νέες απαιτήσεις που εξελίσσονται όσο το σύστημα υλοποιείται.




**ΔΙΑΓΡΑΜΜΑ 31 Εξελικτική Μεθοδολογία ή επαναληπτική και αυξητική ("ΔΟΜΗΜΕΝΟΣ - DP Pangalos.pdf," n.d.)**

Οι βασικές αρχές είναι οι εξής: (Larman and Basili, 2003)

- Κάθε δυσκολία στο σχεδιασμό, την κωδικοποίηση και τη δοκιμή μιας τροποποίησης σημαίνει την ανάγκη για επανασχεδιασμό ή την εκ νέου κωδικοποίηση.
- Οι τροποποιήσεις πρέπει να εντάσσονται απλά σε απομονωμένες και με εύκολη πρόσβαση ενότητες. Αν δεν επιτυγχάνεται αυτό, πιθανόν να είναι απαραίτητος επανασχεδιασμός.
- Οι τροποποιήσεις πρέπει να γίνονται με ευκολότερο τρόπο με την πρόοδο των επαναλήψεων. Εάν δεν συμβαίνει αυτό, τότε υπάρχει βασική υποψία ύπαρξης προβλήματος, όπως ένα ελάττωμα σχεδιασμού.
- Η υφιστάμενη υλοποίηση θα πρέπει να αναλύεται συχνά για να καθορίζεται κατά πόσο πληροί τους στόχους του έργου.

- Οι αντιδράσεις των χρηστών θα πρέπει να λαμβάνονται υπόψη και να αναλύονται για ενδείξεις ελλείψεων στην τρέχουσα έκδοση.

## 3.6. Σπειροειδής μεθοδολογία (Spiral model)

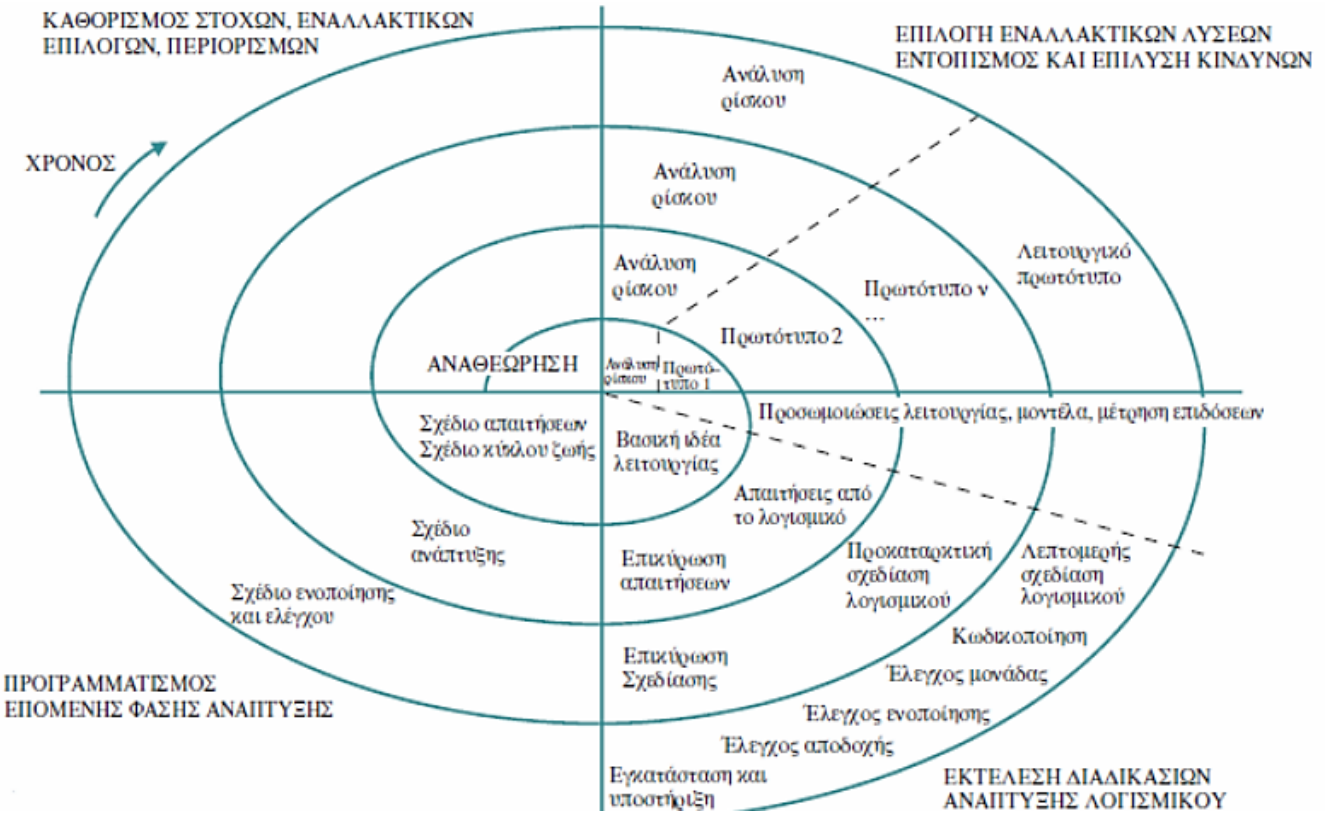
Το 1988, ο Barry Boehm δημοσίευσε μια επίσημη μεθοδολογία ανάπτυξης λογισμικού συστήματος την σπειροειδή, η οποία συνδυάζει κάποιες βασικές πτυχές του μοντέλου καταρράκτη με τις ταχείες μεθοδολογίες πρωτοτυποποίησης, σε μια προσπάθεια να συνδυάσει τα πλεονεκτήματά τους. Έδινε έμφαση σε ένα βασικό τομέα που είχε παραμεληθεί από άλλες μεθοδολογίες, την σκόπιμη επαναληπτική ανάλυση των κινδύνων, που είναι ιδιαίτερα κατάλληλη για πολύπλοκα συστήματα μεγάλης κλίμακας.

Οι βασικές αρχές είναι οι εξής:
(“http://www.icsd.aegean.gr/kkot/softTech06Week1L2.ppt,” n.d.)

- Η μεθοδολογία εστιάζει στην αξιολόγηση και την ελαχιστοποίηση των κινδύνων του έργου, με τον διαχωρισμό του σε μικρότερα τμήματα παρέχοντας διευκόλυνση στην αντιμετώπιση αλλαγών κατά τη διάρκεια της διαδικασίας ανάπτυξης, καθώς και τη δυνατότητα της αξιολόγησης των κινδύνων και της βιωσιμότητας του έργου καθ’ όλη τη διάρκεια του κύκλου ζωής.
- Κάθε κύκλος περιλαμβάνει μια εξέλιξη μέσω της ίδιας ακολουθίας των βημάτων, για κάθε τμήμα του προϊόντος και για κάθε ένα από τα επίπεδα της διαδικασίας.
- Οι φάσεις και οι διαδικασίες ανάπτυξης λογισμικού δεν είναι προκαθορισμένες από το μοντέλο, αλλά εξειδικεύονται στο χώρο της εφαρμογής του.
- Η ανάπτυξη ολόκληρου του συστήματος χωρίζεται σε πολλούς κύκλους, σε καθέναν από τους οποίους προστίθενται νέα λειτουργικά χαρακτηριστικά στο σύστημα.
- Πριν από την έναρξη κάθε κύκλου γίνεται μια μελέτη σκοπιμότητας και ανάλυση κινδύνων, από την οποία προκύπτουν, αφενός , οι συγκεκριμένες εργασίες που θα εκτελεστούν μέσα στον κύκλο, και αφετέρου, η ίδια η εφικτότητα εκτέλεσης του κύκλου αυτού.

Στα τέσσερα τεταρτημόρια κάθε έλικας της σπειροειδούς μεθοδολογίας πραγματοποιούνται τα εξής:

- ο προσδιορισμός των στόχων, των εναλλακτικών λύσεων, και των περιορισμών της επόμενης επανάληψης,
- η αξιολόγηση των εναλλακτικών δυνατοτήτων, ο εντοπισμός και η αντιμετώπιση κινδύνων
- η ανάπτυξη και η επαλήθευση των παραδοτέων της κάθε επανάληψης και
- ο σχεδιασμός της επόμενης επανάληψης.




**ΔΙΑΓΡΑΜΜΑ 32 Σπειροειδής μεθοδολογία ("http://www.icsd.aegean.gr/kkot/softTech06Week1L2.ppt," n.d.)**

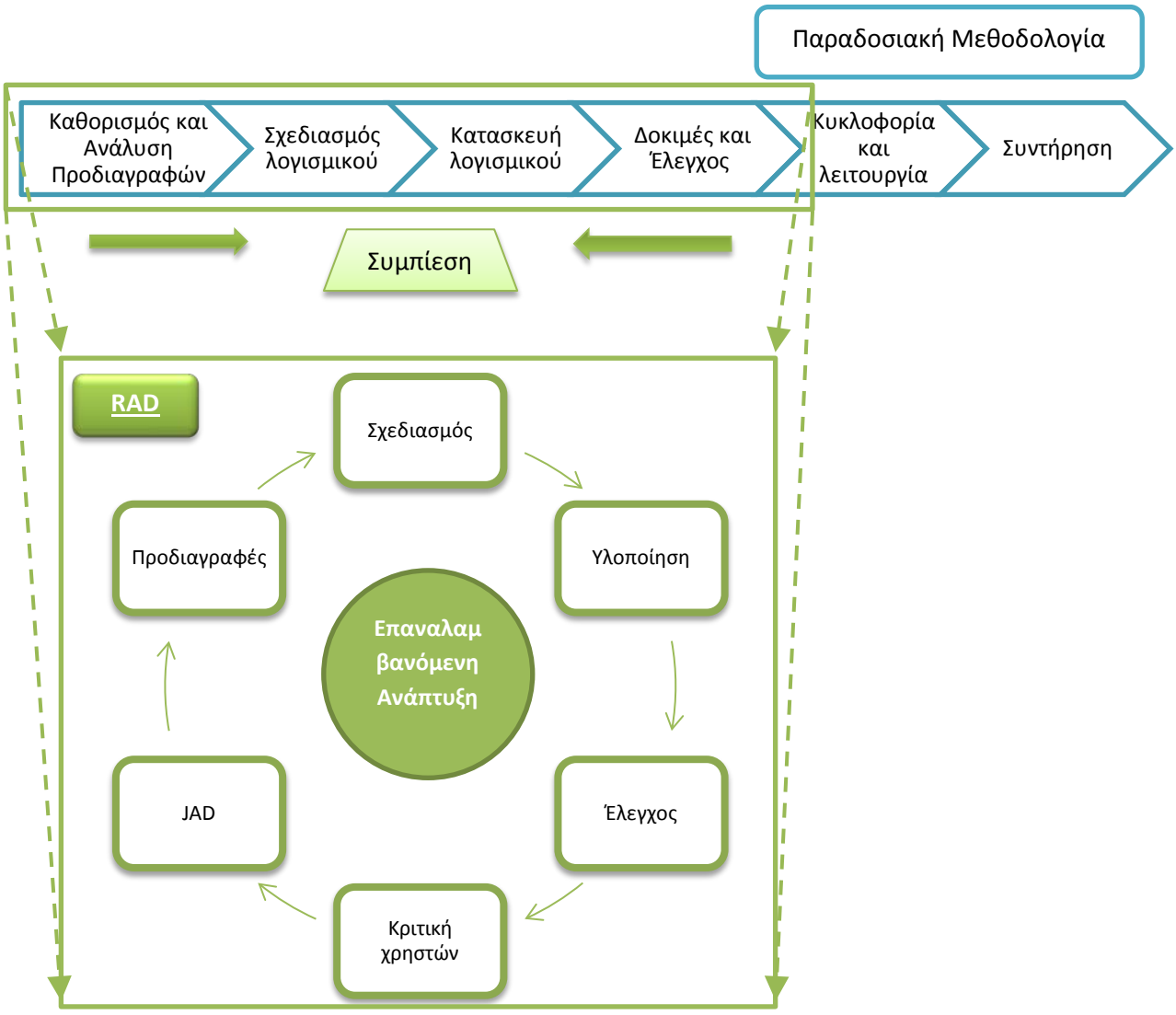
## 3.7. Μεθοδολογία ταχείας ανάπτυξης εφαρμογών (Rapid application development)

Το 1991 εμφανίστηκε ο όρος Ταχεία Ανάπτυξη Εφαρμογών (Rapid Application Development, RAD) για να περιγράψει μία σύνθεση της επαναληπτικής ανάπτυξης με τη χρήση πρωτοτύπων («πρωτοτυποποίηση λογισμικού») και βιβλιοθηκών που παρέχουν έναν έτοιμο προγραμματιστικό σκελετό για την κατασκευή νέων εφαρμογών, προκειμένου να μειωθεί σημαντικά ο χρόνος ανάπτυξης λογισμικού.

Η ταχεία (Rapid) είναι μια μεθοδολογία ανάπτυξης λογισμικού, η οποία τάσσεται υπέρ της επαναληπτικής ροής και της ταχείας κατασκευής πρωτοτύπων αντί του μεγάλου όγκου αρχικού σχεδιασμού. Ο σχεδιασμός του λογισμικού, που πραγματοποιήθηκε με τη μέθοδο ταχείας ανάπτυξης, είναι αλληλένδετος με την κατασκευή του. Η έλλειψη εκτεταμένου προ-σχεδιασμού γενικά επιτρέπει την πολύ πιο γρήγορη κατασκευή και κωδικοποίηση του λογισμικού και καθιστά ευκολότερη την αντιμετώπιση αλλαγής των απαιτήσεων.

Με τον τρόπο αυτό, επιχειρήθηκε να επιλυθεί το βασικό πρόβλημα των αυστηρών μοντέλων όπως ο Καταρράκτης ή η Σπείρα: το ζητούμενο προϊόν απαιτούσε τόσο μεγάλο χρόνο κατασκευής που οι απαιτήσεις ίσως είχαν αλλάξει πριν την παράδοσή του.

Η διαδικασία ταχείας ανάπτυξης ξεκινά με την ανάπτυξη των προκαταρκτικών μοντέλων δεδομένων και των μοντέλων επιχειρηματικής διαδικασίας με τη χρήση δομημένων τεχνικών. Στο επόμενο στάδιο, οι απαιτήσεις επαληθεύονται με τη χρήση των πρωτοτύπων, με στόχο τη βελτίωση των μοντέλων δεδομένων και διαδικασιών. Αυτά τα στάδια επαναλαμβάνονται συνεχώς, ενώ η περαιτέρω ανάπτυξη οδηγεί σε ένα συνδυασμένο αποτέλεσμα επιχειρησιακών απαιτήσεων και τεχνικών σχεδιασμού που θα μπορεί να χρησιμοποιηθεί για την κατασκευή νέων συστημάτων. (Whitten, 1998)




**ΔΙΑΓΡΑΜΜΑ 33 Μεθοδολογία ταχείας ανάπτυξης ("Rapid application development : Software methodologies | Rivulets Technologies," n.d.)**

Οι βασικές αρχές της ταχείας ανάπτυξης εφαρμογών είναι οι εξής:

- Βασικός στόχος είναι η ταχεία ανάπτυξη και παράδοση ενός συστήματος υψηλής ποιότητας με σχετικά χαμηλό κόστος επένδυσης.
- Εστιάζει στη μείωση των εγγενών κινδύνων ενός έργου με το διαχωρισμό του σε μικρότερα τμήματα και την διευκόλυνση της αντιμετώπισης των αλλαγών κατά τη διάρκεια της διαδικασίας ανάπτυξης.
- Αποσκοπεί στην παραγωγή συστημάτων υψηλής ποιότητας γρήγορα, κυρίως μέσω της επαναληπτικής μεθοδολογίας Πρωτοτυποποίησης (σε οποιοδήποτε στάδιο της ανάπτυξης), την ενεργό συμμετοχή των τελικών χρηστών, καθώς και της χρήσης υπολογιστικών εργαλείων ανάπτυξης. Τα εργαλεία αυτά μπορούν να περιλαμβάνουν πακέτα γραφικού περιβάλλοντος χρήστη (Graphical User Interface (GUI)), εργαλεία Computer Aided Software Engineering (CASE), συστήματα διαχείρισης βάσεων δεδομένων (DBMS), γλώσσες προγραμματισμού, γεννήτριες κώδικα, και αντικειμενοστραφείς τεχνικές.
- Βασική έμφαση δίνεται στην εκπλήρωση των αναγκών των επιχειρήσεων, ενώ η τεχνολογική ή μηχανική τελειότητα είναι μικρότερης σημασίας.
- Σε γενικές γραμμές περιλαμβάνει τον από κοινού σχεδιασμό της εφαρμογής (joint application design, JAD), όπου οι χρήστες εμπλέκονται έντονα στο σχεδιασμό του συστήματος, με την επίτευξη συναίνεσης είτε μέσω οργανωμένων συνεργασιών είτε ηλεκτρονικής αλληλεπίδρασης. Η ενεργός συμμετοχή των χρηστών είναι επιτακτική ανάγκη.
- Παράγει τεκμηρίωση ενδεχομένως απαραίτητη για τη διευκόλυνση της μελλοντικής ανάπτυξης και τη συντήρησης.

## 3.8. Ευέλικτη μεθοδολογία (Agile software development)

Οι ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού αποτελούν μια ομάδα μεθόδων στην οποία οι απαιτήσεις και οι λύσεις εξελίσσονται μέσω της συνεργασίας μεταξύ αυτο-οργανωνόμενων διατμηματικών ομάδων εργασίας. Ενσωματώνει τον προσαρμοστικό σχεδιασμό, την εξελικτική ανάπτυξη, την έγκαιρη παράδοση, τη συνεχή βελτίωση, και ενθαρρύνει την ταχεία και ευέλικτη ανταπόκριση στις αλλαγές. (Abrahamsson et al., 2002)

Το Μανιφέστο για την ευέλικτη ανάπτυξη λογισμικού, επίσης γνωστό ως Agile Manifesto, για πρώτη φορά καθόρισε τις βασικές έννοιες της ευέλικτης ανάπτυξης, εισήγαγε τον όρο το 2001, έχοντας ήδη εξελιχθεί με διάφορους τρόπους από το 1995.

Οι βασικές αρχές της ευέλικτης ανάπτυξης εφαρμογών είναι οι εξής: (Cockburn, 2007)

- Είναι περισσότερο ένα σύνολο από κεντρικές αρχές, στηριγμένες στο μοντέλο ανάπτυξης σε φάσεις και στην ευρύτερη φιλοσοφία του, παρά μία αυστηρά ορισμένη διεργασία ανάπτυξης λογισμικού.
- Το Ευέλικτο Μοντέλο διακηρύσσει πως, αντί να προσπαθεί να προβλέψει το μέλλον, προσαρμόζεται σε αυτό. Έτσι έρχεται σε αντιπαράθεση με δύσκαμπτα, αυστηρά μοντέλα όπως ο Καταρράκτης και η Σπείρα, τα οποία επιχειρούν να καταπολεμήσουν την αβεβαιότητα καταρτίζοντας λεπτομερειακά πλάνα εκ των προτέρων. Κατ' αντιδιαστολή, στο ευέλικτο μοντέλο μόνο η αρχή και ο στόχος είναι πλήρως γνωστά από την πρώτη στιγμή, ενώ τα μεσαία στάδια καθορίζονται στην πορεία αναλόγως με τις συνθήκες.
- Σε σχέση με τον Καταρράκτη και τη Σπείρα προσπαθεί να είναι πιο ανθρωποκεντρικό, πιο άτυπο, πιο δεκτικό στις αλλαγές και περισσότερο προσανατολισμένο στη συνεργασία και αλληλεπίδραση μεταξύ των ατόμων που σχετίζονται με το ζητούμενο λογισμικό, παρά στη διαρκή παραγωγή εγγράφων τεκμηρίωσης μέσω μηχανικών κανόνων.
- Ο μακροχρόνιος, συνολικός σχεδιασμός αποφεύγεται και προτιμάται ο βραχυπρόθεσμος σχεδιασμός για μικρά τμήματα του ζητούμενου προϊόντος, τα οποία μπορούν ανεξάρτητα να υλοποιηθούν, ελεγχθούν και παραδοθούν σύντομα προς χρήση και αξιολόγηση.
- Η ευέλικτη μεθοδολογία γίνεται στη βάση της αυξητικής και επαναληπτικής ανάπτυξης από συνεργαζόμενες ομάδες ατόμων, οι οποίες έρχονται σε άμεση και τακτική επαφή μεταξύ τους, με τους χρήστες και με τον πελάτη. Οι ομάδες αυτές είναι μικρές σε μέγεθος, δεν ιεραρχούνται αυστηρά και συνήθως αυτοοργανώνονται ως προς την ανάθεση καθηκόντων. Αυτό δεν σημαίνει πως δεν τηρούνται τυπικές διαδικασίες κατά την ανάπτυξη (π.χ. χρονικά ορόσημα για την ολοκλήρωση κάποιας εργασίας, καταμερισμός εργασιών κλπ), αλλά ότι η έμφαση δίνεται στην προσαρμοστικότητα και στην ευελιξία ώστε να αντιμετωπίζονται εύκολα απρόβλεπτες αλλαγές, χωρίς να συνοδεύονται από καταστροφικές επιπτώσεις στην ανάπτυξη του προϊόντος.
- Ως προβλήματα του Ευέλικτου Μοντέλου έχουν επισημανθεί η ελλιπής γραπτή τεκμηρίωση, η αδυναμία λειτουργίας με μεγάλες ομάδες εργαζομένων και οι καθυστερήσεις στις τελευταίες επαναλήψεις της διεργασίας ανάπτυξης, όταν επιχειρείται η μαζική προσθήκη στο λογισμικό πολλαπλών ζητούμενων χαρακτηριστικών που παραλείπονταν κατά τις προηγούμενες επαναλήψεις.

Μερικές από τις πιο γνωστές ευέλικτες μεθόδους ανάπτυξης λογισμικού αναφέρονται παρακάτω: (Abrahamsson et al., 2002)

- Adaptive software development (ASD)
- Agile modeling
- Agile Unified Process (AUP)
- Crystal Clear Methods (Crystal Clear)

- Disciplined agile delivery
- Dynamic systems development method (DSDM)
- Extreme programming (XP)
- Feature-driven development (FDD)
- Lean software development
- Kanban (development)
- Scrum
- Scrum ban

Οι ευέλικτες μέθοδοι επικεντρώνονται σε διαφορετικές πτυχές του κύκλου ζωής της ανάπτυξης λογισμικού. Κάποιες δίνουν περισσότερη έμφαση στις πρακτικές (π.χ., Extreme programming, ρεαλιστικού προγραμματισμού (pragmatic programming), Agile modeling), ενώ άλλες εστιάζουν στη διαχείριση των έργων λογισμικού (π.χ. Scrum). Ωστόσο, υπάρχουν προσεγγίσεις που παρέχουν πλήρη κάλυψη στο σύνολο του κύκλου ζωής της ανάπτυξης (π.χ. DSDM, IBM RUP), ενώ οι περισσότερες από αυτές είναι κατάλληλες από τη φάση της προδιαγραφής απαιτήσεων και έπειτα (για παράδειγμα η FDD). Έτσι, υπάρχει μια σαφής διαφοροποίηση μεταξύ των διαφόρων ευέλικτων μεθόδων. (Abrahamsson et al., 2002)

## 3.9. Λοιπές μεθοδολογίες ανάπτυξης λογισμικού

Επιπλέον σημαντικές μεθοδολογίες ανάπτυξης λογισμικού έργου υψηλού επιπέδου είναι: (“The Complete List of Software Development Frameworks, Process’s, Methods, or Philosophies | Software Development in the Real World,” n.d.)

- Το Χαοτικό μοντέλο (Chaos model) - Ο βασικός κανόνας είναι ότι πρέπει πάντα πρώτα να επιλύεται το πιο σημαντικό θέμα.
- Η Incremental funding methodology - μια επαναληπτική ππροσέγγιση.
- Η Ανάλυση Δομημένων συστημάτων και μεθόδων σχεδιασμού (Structured systems analysis and design method) - μια ειδική έκδοση της μεθοδολογίας καταρράκτη.
- Ο αργός προγραμματισμός (Slow programming), εστιάζει στην προσεκτική και βαθμιαία εργασία χωρίς (ή με ελάχιστη) πίεση χρόνου. Ο αργός προγραμματισμός στοχεύει στην αποφυγή σφαλμάτων και στην απελευθέρωση από τα υπερβολικά γρήγορα χρονοδιαγράμματα.
- Το V-Μοντέλο (V-Model) - μια επέκταση του μοντέλου καταρράκτη.
- Οι Ενοποιημένες Διαδικασίες (Unified Process, UP), είναι επαναληπτικές μεθοδολογίες ανάπτυξης λογισμικού, με βάση την Ενοποιημένη Γλώσσα

Μοντελοποίησης (Unified Modeling Language, UML). Η ενοποιημένη διαδικασία οργανώνει την ανάπτυξη του λογισμικού σε τέσσερις φάσεις, καθεμιά από τις οποίες αποτελείται από μια ή περισσότερες εκτελέσιμες επαναλήψεις του λογισμικού σε συγκεκριμένο στάδιο της ανάπτυξης: την σύλληψη, την επεξεργασία, την κατασκευή, και τις κατευθυντήριες γραμμές. Πολλά εργαλεία και προϊόντα υπάρχουν για τη διευκόλυνση της ενοποιημένης διαδικασίας. Μία από τις πιο δημοφιλείς εκδόσεις της ενοποιημένης διαδικασίας είναι η Λογική ενοποιημένη διαδικασία (Rational Unified Process, RUP).

# Κεφάλαιο 4

## 4. Οι ευέλικτες (agile) μεθοδολογίες έως τη scrum

### 4.1. Απαιτήσεις

Έχει παρατηρηθεί πως οι επαγγελματίες ανάπτυξης λογισμικού προσπαθούν συνεχώς να αυξήσουν την παραγωγικότητα και παράλληλα να διατηρήσουν την ποιότητα σε ψηλά επίπεδα. Η παραπάνω διαπίστωση είναι ένας από τους σημαντικότερους λόγους που οδήγησε τις επιχειρήσεις ανάπτυξης λογισμικού στη συνεχή αναζήτηση νέων μεθόδων που θα τις βοηθήσουν στην υλοποίηση των έργων τους. Μια ακόμη ανάγκη που θεωρείται ισχυρό κίνητρο, είναι τα προβλήματα που προκύπτουν σχετικά με ανικανοποίητες απαιτήσεις των χρηστών και τα ελλειπή λειτουργικά συστήματα.

Οι ευέλικτες μέθοδοι (Agile methodologies) εμφανίστηκαν πρόσφατα στην ανάπτυξη λογισμικού για να συμπληρώσουν όλες αυτές τις απαιτήσεις των σύγχρονων επιχειρήσεων. Ο όρος «ευέλικτες» χρησιμοποιήθηκε για να ομαδοποιήσει και να χαρακτηρίσει ένα σύνολο από μεθόδους οι οποίες διέπονται από κοινούς «κανόνες». Αυτές οι μέθοδοι δεν είναι θεωρητικές, αλλά υπαρκτά εργαλεία που μπορούν να εξυπηρετήσουν τις ανάγκες που προκύπτουν κατά τη διάρκεια της διαδικασίας ανάπτυξης λογισμικού.

Οι ευέλικτες μέθοδοι συνεισφέρουν στην έγκαιρη παράδοση ενός λογισμικού στον πελάτη και ανταποκρίνονται πιο άμεσα στις περιβαλλοντικές αλλαγές και στις διαφοροποιήσεις των απαιτήσεων και προδιαγραφών του λογισμικού. Επιπλέον, βοηθούν στην αποφυγή περιττών διαδικασιών οι οποίες επιβαρύνουν το έργο, είτε χρηματικά είτε χρονικά, αυξάνοντας παράλληλα την παραγωγικότητα. Οι ιδιότητες αυτές των ευέλικτων πρακτικών, τις έχουν καταστήσει ιδιαίτερα δημοφιλείς.

## 4.2. Το μανιφέστο των ευέλικτων μεθόδων (Agile Manifesto)

Οι ευέλικτες μέθοδοι εμφανίστηκαν στα μέσα της δεκαετίας του 1990, ως μια εναλλακτική λύση στις αυστηρές μεθόδους της παραδοσιακής ανάπτυξης λογισμικού. Στις αρχές του 2001, 17 σημαντικές προσωπικότητες στον τομέα της ευέλικτης ανάπτυξης λογισμικού συναντήθηκαν στο Snowbird της Utah (ΗΠΑ) για να συζητήσουν για ελαφριές μεθόδους ανάπτυξης λογισμικού. Αποτέλεσμα της συνάντησης ήταν το μανιφέστο των ευέλικτων μεθόδων (Agile Manifesto). ("Μανιφέστο για την ευέλικτη ανάπτυξη λογισμικού," n.d.)

Αυτές οι μέθοδοι υπόσχονται ανταπόκριση στις αλλαγές, παραγωγικότερες πρακτικές και λιγότερη γραφειοκρατία. Το όνομα ευέλικτες αναφέρεται κυρίως στη συνολική ικανότητα να προσαρμόζουν κατάλληλα τη διαδικασία ανάπτυξης, όταν προκύπτουν αλλαγές στην πορεία του έργου.

Σύμφωνα με το μανιφέστο υπάρχουν κάποιοι «κανόνες» που καθιστούν μια μέθοδο ευέλικτη. Οι τέσσερις βασικές αρχές που καθορίζουν τις ευέλικτες μεθόδους είναι: ("Μανιφέστο για την ευέλικτη ανάπτυξη λογισμικού," n.d.)

- Τα άτομα και οι αλληλεπιδράσεις πάνω από τις διαδικασίες και τα εργαλεία
- Το λογισμικό που λειτουργεί πάνω από την εκτενή τεκμηρίωση
- Η συνεργασία με τον πελάτη πάνω από τις συμβατικές διαπραγματεύσεις
- Η ανταπόκριση στην αλλαγή πάνω από την τήρηση ενός προδιαγεγραμμένου σχεδίου

Πρώτον, τονίζεται η ανάγκη για συνεργασία μεταξύ των συμμετεχόντων σε ένα έργο λογισμικού, γεγονός το οποίο αποτελεί τον σημαντικότερο παράγοντα κατά την ανάπτυξη λογισμικού. Οι διαδικασίες και τα εργαλεία δεν πρόκειται να αποτελέσουν σημαντική βοήθεια, αν η συνεργασία αυτή δεν είναι επιτυχής.

Δεύτερον, το μανιφέστο υπογραμμίζει τον πρωταρχικό στόχο του έργου, ο οποίος είναι η παραγωγή λειτουργικού λογισμικού. Οι προγραμματιστές προτρέπονται να διατηρήσουν τον κώδικα όσο πιο απλό γίνεται, με αποτέλεσμα το ίδιο το λογισμικό να είναι περισσότερο κατανοητό σε σχέση με μακροσκελή έγγραφα και διαγράμματα τεκμηρίωσης. Η ολοκληρωμένη τεκμηρίωση είναι σημαντική για να εξηγεί το στόχο της εφαρμογής, αλλά ο ρόλος της θα είναι πάντα δευτερεύων και συμπληρωματικός.

Τρίτον, η συνεργασία και η επικοινωνία της ομάδας ανάπτυξης του λογισμικού με τον πελάτη προτιμάται σε σύγκριση με αυστηρά συμβόλαια και συμβάσεις. Η συνεργασία αυτή είναι συνήθως δύσκολη, αφού ο πελάτης αλλάζει συχνά γνώμη σχετικά με τις απαιτήσεις της εφαρμογής. Η ύπαρξη ενός συμβολαίου είναι σημαντική, ωστόσο δεν μπορεί να υποκαταστήσει την επικοινωνία.

Τέταρτον, οι συμμετέχοντες στο έργο πρέπει να είναι καλά πληροφορημένοι, έτοιμοι να εξετάσουν και να αντιμετωπίσουν τις ανάγκες που θα εμφανιστούν κατά τη διάρκεια του έργου. Οι απαιτήσεις των πελατών, το επιχειρηματικό περιβάλλον ακόμα και η ίδια η τεχνολογία, μεταβάλλονται συνεχώς και δεν αφήνουν ανεπηρέαστη την ανάπτυξη. Για αυτό θα πρέπει η ανάπτυξη να ικανοποιεί άμεσα την ανάγκη για αλλαγές. Αυτό δεν σημαίνει ότι απορρίπτεται το σχέδιο χρονοπρογραμματισμού, αλλά τα σχέδια θα πρέπει να είναι περισσότερο ευέλικτα. (Stamelos and Sfetsos, 2007)

## 4.3. Οι 12 αρχές του ευέλικτου μανιφέστο

Στο μανιφέστο των ευέλικτων μεθόδων καθορίστηκαν επιπλέον δώδεκα αρχές που καθοδηγούν τον τρόπο ανάπτυξης. Παρουσιάζονται όπως διατυπώθηκαν: (“Μανιφέστο για την ευέλικτη ανάπτυξη λογισμικού,” n.d.)

1. Βασική προτεραιότητα αποτελεί η ικανοποίηση του πελάτη με τη συνεχή παράδοση σημαντικού τμήματος του λογισμικού από τα πρώτα κιόλας στάδια της παραγωγής.

2. Οι μεταβαλλόμενες απαιτήσεις είναι καλοδεχούμενες ακόμα και σε προχωρημένο στάδιο ανάπτυξης. Οι διαδικασίες, που εφαρμόζονται έχουν τη δυνατότητα να περιλαμβάνουν την αλλαγή προς όφελος του πελάτη.

3. Η παράδοση τμημάτων λογισμικού γίνεται ανά δύο εβδομάδες έως και ανά δύο μήνες. Σημαντική είναι η όσο το δυνατόν συχνότερη παράδοση.

4. Οι πελάτες και οι υπεύθυνοι ανάπτυξης του συστήματος πρέπει να συνεργάζονται καθημερινά μέχρι την παραγωγή του τελικού προϊόντος.

5. Σημαντική είναι η ανάθεση των τμημάτων του συστήματος σε ικανό και αποτελεσματικό προσωπικό και η εξασφάλιση ευνοϊκού περιβάλλοντος εμπιστοσύνης και υποστήριξης.

6. Ο καλύτερος τρόπος μεταβίβασης και ανταλλαγής πληροφοριών με την ομάδα παραγωγής είναι η διαπροσωπική συζήτηση και οι συνομιλίες.

7. Το καλύτερο και πιο αξιόπιστο μέτρο, που επιβεβαιώνει την πρόοδο, είναι το να λειτουργούν σωστά τα τμήματα λογισμικού τα οποία κατασκευάζονται.

8. Οι ευέλικτες διαδικασίες προωθούν τον σταθερό ρυθμό ανάπτυξης του συστήματος, ο οποίος πρέπει να ακολουθείται τόσο από τον πελάτη όσο και από τους υπεύθυνους παραγωγής.

9. Η ευελιξία ενισχύεται από την συνεχή προσπάθεια για τεχνική αρτιότητα και καλό σχεδιασμό.

10. Η απλοποίηση, με την έννοια της υλοποίησης στόχων με σύντομο αλλά αποτελεσματικό τρόπο, είναι ουσιαστική.

11. Οι καλύτερες αρχιτεκτονικές, απαιτήσεις και σχέδια προκύπτουν από ομάδες, που αυτό-οργανώνονται

12. Σε τακτά χρονικά διαστήματα, η ομάδα συζητά τρόπους, που την βοηθούν να γίνει περισσότερο αποτελεσματική και επαναπροσδιορίζει τη συμπεριφορά της.

Η τήρηση των βασικών αυτών αρχών του μανιφέστο μειώνει την πιθανότητα να εμφανιστούν κίνδυνοι και λάθη κατά την διαδικασία ανάπτυξης του λογισμικού. Συνοψίζοντας, λοιπόν, οι ευέλικτες μέθοδοι ενσωμάτωσαν μια ευρεία συλλογή από καλές και δοκιμασμένες αξίες και πρακτικές που βοηθούν στην ανάπτυξη ποιοτικού λογισμικού. Η ανάπτυξη του λογισμικού γίνεται σε σύντομους επαναληπτικούς (iterative) και αυξητικούς (incremental) κύκλους, παρέχοντας τη δυνατότητα της προσαρμογής και αντίδρασης στις αλλαγές που τίθενται από το διαρκώς μεταβαλλόμενο επιχειρησιακό περιβάλλον. (Cockburn, 2007)

Με σαφή προτίμηση στην επικοινωνία για τον περιορισμό των παραγομένων εγγράφων και σε στενή συνεργασία με τον πελάτη, οι κατασκευαστές και οι διαχειριστές του έργου επικεντρώνονται στην ανάπτυξη λογισμικού, που θεωρείται το πρωταρχικό μέτρο προόδου.

## 4.4. Τα χαρακτηριστικά των ευέλικτων μεθόδων

Η καινοτομία για τις ευέλικτες μεθόδους δεν είναι οι πρακτικές που χρησιμοποιούν, αλλά η αναγνώρισή τους από τους ανθρώπους ως τους κύριους συντελεστές της επιτυχίας του έργου, σε συνδυασμό με την έμφαση στην αποτελεσματικότητα και την ευελιξία. (Cockburn, 2007)

Μια μέθοδος μπορεί να θεωρηθεί ως ευέλικτη όταν είναι επαυξητική (παραδίδονται μικρά τμήματα του λογισμικού, σε τακτά χρονικά διαστήματα), συνεταιριστική (υπάρχει επαρκής επικοινωνία μεταξύ της ομάδας ανάπτυξης και των πελατών), απλή (είναι εύκολη, κατανοητή και τροποποιήσιμη) και προσαρμοστική (μπορεί εύκολα να προσαρμόζεται σε απρόβλεπτες αλλαγές).

Οι ακόλουθες αρχές των ευέλικτων πρακτικών θεωρούνται ως οι κύριες διαφορές μεταξύ ευέλικτων και παραδοσιακών μεθόδων. (“Agile Alliance : Home,” n.d.)

Προσανατολισμένες στον άνθρωπο: Οι ευέλικτες μέθοδοι θεωρούν τους ανθρώπους – πελάτες, προγραμματιστές και γενικά τους συμμετέχοντες – ως το σημαντικότερο παράγοντα στην διαδικασία ανάπτυξης λογισμικού.

Προσαρμοστικότητα: Στις ευέλικτες μεθόδους, σκοπός δεν είναι να περιοριστούν οι αλλαγές που παρουσιάζονται, αλλά η εύρεση τρόπων που θα επιτρέψει τον καλύτερο χειρισμό τους. Οι εξωτερικές αλλαγές δεν μπορούν να περιοριστούν, ενώ μια βιώσιμη στρατηγική, είναι η αποτελεσματική αντιμετώπισή τους με το λιγότερο δυνατό κόστος. Οι αλλαγές στις απαιτήσεις αντιμετωπίζονται θετικά, επειδή η ομάδα έχει κατανοήσει καλύτερα τις ανάγκες του πελάτη και συνεπώς γνωρίζει πώς να τις ικανοποιήσει και να καταλήξει σε ένα ολοκληρωμένο προϊόν. (Cockburn, 2007)

Έμφαση στην πραγματικότητα: Οι ευέλικτες μέθοδοι δίνουν έμφαση στα δεδομένα σε κάθε φάση της ανάπτυξης και όχι στην τήρηση ενός καλά ορισμένου σχεδίου. Κάθε επανάληψη προσθέτει αξία στο συνεχώς αναπτυσσόμενο προϊόν. Η απόφαση για το αν η επιχειρηματική αξία έχει προστεθεί ή όχι δεν θα δοθεί από τους προγραμματιστές, αλλά από τους τελικούς χρήστες και τους πελάτες.

Εξισορρόπηση ευελιξίας και σχεδιασμού: Ο χρονικός προγραμματισμός είναι σημαντικός, αλλά το πρόβλημα είναι ότι τα έργα λογισμικού δεν μπορούν να προβλεφθούν με ακρίβεια γιατί επηρεάζονται από πολλούς παράγοντες που πρέπει να ληφθούν υπόψη. Μια καλύτερη στρατηγική είναι να γίνονται λεπτομερή σχέδια για το άμεσο μέλλον, για τις επόμενες εβδομάδες, λιγότερο αναλυτικά σχέδια για τους επόμενους μήνες, και λιγότερα σχέδια πέρα από αυτό το χρονικό διάστημα. Η ευελιξία στην αλλαγή των αποφάσεων είναι σημαντική για ένα έργο λογισμικού. Μια απόφαση δεν πρέπει να παρθεί άμεσα αλλά να αναζητηθεί ένας τρόπος ώστε να αναβληθεί για αργότερα ή να παρθεί με τρόπο ώστε να εξασφαλιστεί το ενδεχόμενο αλλαγής της αργότερα χωρίς δυσκολία.(Dingsøyr et al., 2012)

Εμπειρική διαδικασία: Οι ευέλικτες μέθοδοι ανάπτυξης λογισμικού είναι εμπειρικές διαδικασίες. Η ανάπτυξη λογισμικού δεν μπορεί να θεωρηθεί καθορισμένη διαδικασία γιατί πολλές αλλαγές προκύπτουν κατά τη διάρκεια της ανάπτυξης. Είναι εξαιρετικά απίθανο οποιοδήποτε σύνολο προκαθορισμένων βημάτων να οδηγήσουν σε ένα επιθυμητό, προβλέψιμο αποτέλεσμα, επειδή οι απαιτήσεις αλλάζουν, παρουσιάζονται αλλαγές στην χρησιμοποιούμενη τεχνολογία, άνθρωποι επεμβαίνουν και απομακρύνονται από την ομάδα ανάπτυξης, και ούτω καθεξής. (Abrahamsson et al., 2002)

Αποκεντρωμένη προσέγγιση (decentralized approach): Η αποκεντρωμένη διαχείριση επηρεάζει σημαντικά ένα σύστημα λογισμικού μέσω της εξοικονόμησης περισσότερου χρόνου από ότι σε μια διαδικασία όπου οι λήψεις αποφάσεων περιορίζονται σε ένα πρόσωπο. Η ευέλικτη ανάπτυξη λογισμικού ενθαρρύνει την λήψη αποφάσεων από τους προγραμματιστές, χωρίς αυτό να σημαίνει ότι οι προγραμματιστές θα αναλάβουν το ρόλο της διοίκησης. Η διοίκηση παραμένει

αναγκαία για την αντιμετώπιση των προβλημάτων που παρουσιάζονται κατά το διάστημα της ανάπτυξης αλλά θα πρέπει να αναγνωρίζει την ικανότητα των μελών της ομάδας να λαμβάνουν από μόνοι τους αποφάσεις.

Απλότητα: Μια ομάδα ανάπτυξης λογισμικού, για να είναι ευέλικτη, θα πρέπει να λαμβάνει πάντα την πιο απλή απόφαση που μπορεί να εξυπηρετήσει τους στόχους της. Δεν πρέπει να προβλέπει μελλοντικά προβλήματα και να προσπαθεί να τα επιλύσει σήμερα. Η απλότητα είναι αναγκαία ώστε να είναι εύκολο να αλλάξει το σύστημα. Ποτέ δεν παράγεται περισσότερο από ό,τι είναι απαραίτητο.

Συνεργασία: Η συνεχής συνεργασία μεταξύ των μελών της ομάδας είναι απαραίτητη. Οι ευέλικτες ομάδες δεν μπορούν να υπάρξουν μόνο με την περιστασιακή επικοινωνία. Χρειάζεται συνεχής πρόσβαση και εμπλοκή στις διαδικασίες της επιχείρησης για την οποία αναπτύσσεται το σύστημα. Επίσης, ο πελάτης πρέπει να συνεργάζεται στενά με την ομάδα ανάπτυξης, παρέχοντας συχνή ανατροφοδότηση για το υπό ανάπτυξη σύστημα. (Shore, 2008)

Μικρές αυτό-οργανωμένες ομάδες: Οι ευέλικτες μέθοδοι ανάπτυξης λογισμικού λειτουργούν καλύτερα με μικρές ομάδες οι οποίες πρέπει να οργανώνονται από μόνες τους. Είναι συνηθισμένη η πρακτική να κοινοποιούνται οι ευθύνες στην ομάδα ως σύνολο, και έπειτα η ομάδα να καθορίζει τον καλύτερο τρόπο για την επίτευξή τους μέσω συζητήσεων και συχνής επικοινωνίας μεταξύ των μελών της για όλες τις παραμέτρους του έργου. (“Ευέλικτη Ανάπτυξη Λογισμικού | Sprint SMEs,” n.d.)

## 4.5. Ευέλικτες μέθοδοι ανάπτυξης λογισμικού (Agile methodologies)

Με βάση τα παραπάνω χαρακτηριστικά, παρουσιάζονται περιληπτικά τέσσερις από τις πιο διαδεδομένες ευέλικτες μεθόδους. (“Ευέλικτη Ανάπτυξη Λογισμικού | Sprint SMEs,” n.d.), (Abrahamsson et al., 2002)

### 4.5.1. Ακραίος προγραμματισμός (Extreme Programming)

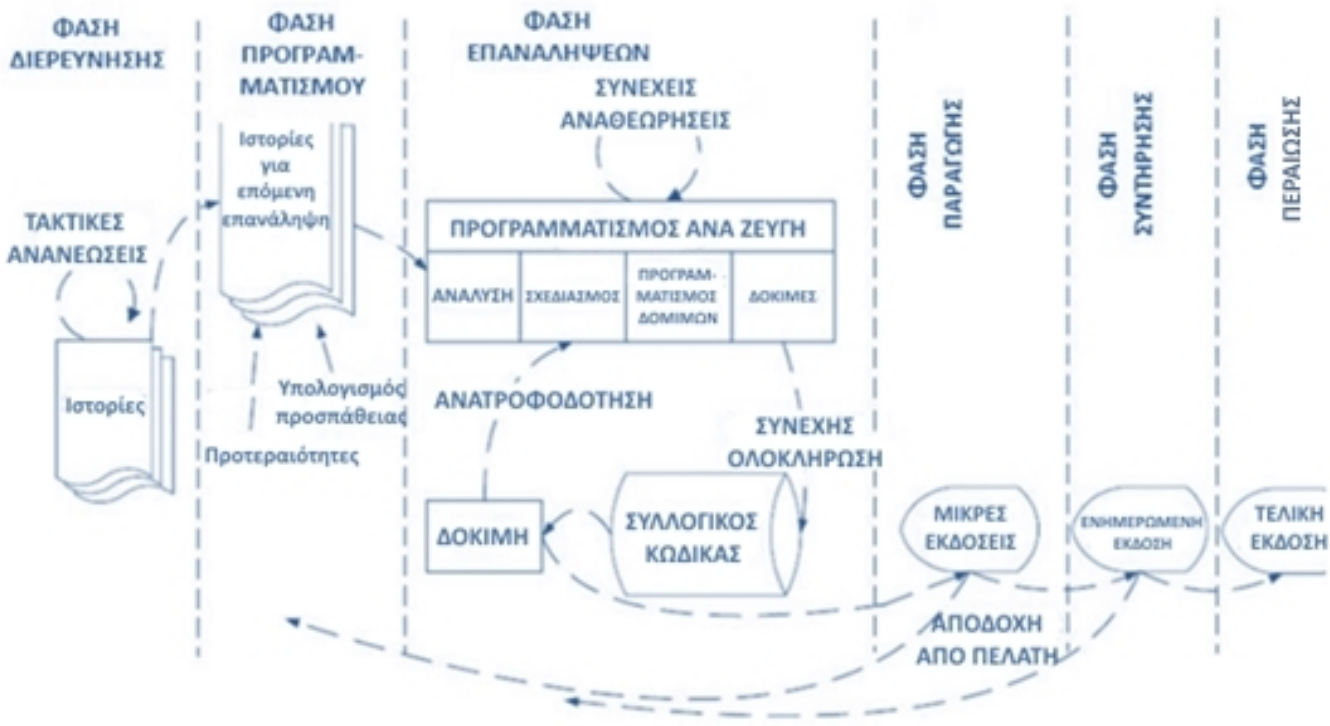
Ο ακραίος προγραμματισμός (Extreme Programming, XP), εντάσσεται στις ευέλικτες μεθόδους αφού προσπαθεί να βελτιώσει την ποιότητα του λογισμικού και επιτρέπει τις συχνές αλλαγές στις απαιτήσεις των χρηστών. Επινοήθηκε από τον Kent Beck και καταγράφηκε για πρώτη φορά, περιγράφοντας τα βασικά χαρακτηριστικά και πρακτικές της μεθόδου. (Beck, 2005)

Η μέθοδος XP χαρακτηρίζεται από σύντομους κύκλους ανάπτυξης, επαυξητικές εκδόσεις, συνεχή ανατροφοδότηση, συνεχή επικοινωνία και επαναστατικό σχεδιασμό. Ο όρος «ακραίος» προέρχεται από τη λήψη των παραπάνω αρχών και

πρακτικών κοινής λογικής σε ακραία επίπεδα. Η μέθοδος βασίζεται στην πεποίθηση ότι τα δυνατά στοιχεία των παραδοσιακών προσεγγίσεων (π.χ. η ανάμειξη του πελάτη), λαμβάνονται και εφαρμόζονται στα ακραία επίπεδά τους, βασισμένη στην ιδέα ότι αν κάτι είναι καλό, τότε ο μέγιστος βαθμός του θα επιφέρει ακόμη καλύτερα αποτελέσματα. (Beck, 2005)

Ο κύκλος ζωής του XP περιλαμβάνει τις εξής φάσεις όπως αυτές παρουσιάζονται στο παρακάτω διάγραμμα: (Beck, 2005)

- Διερεύνηση (Exploration)
- Προγραμματισμός (Planning)
- Επαναλήψεις (Iterations to release phase)
- Παραγωγή (Productionizing)
- Συντήρηση (Maintenance)
- Περαίωση (Retirement)




**ΔΙΑΓΡΑΜΜΑ 34 Κύκλος ζωής της μεθοδολογίας Ακραίου Προγραμματισμού (Beck, 2005)**

Στη φάση της διερεύνησης πραγματοποιείται η καταγραφή των χαρακτηριστικών και των λειτουργιών που θα περιλαμβάνει το σύστημα. Οι πελάτες καλούνται να καταγράψουν σε «κάρτες ιστορίας» (story cards) τα χαρακτηριστικά γνωρίσματα που θα προστεθούν στο σύστημα ενώ η ομάδα ανάπτυξης εξοικειώνεται με τα εργαλεία, τις πρακτικές και την τεχνολογία που θα χρησιμοποιηθούν στο έργο. Μετά τον καθορισμό των τεχνολογιών που θα χρησιμοποιηθούν αναπτύσσεται ένα πρωτότυπο του συστήματος.

Στη φάση του προγραμματισμού καθορίζονται οι ιστορίες που θα έχουν προτεραιότητα και τα περιεχόμενα της πρώτης έκδοσης του συστήματος. Η ομάδα ανάπτυξης αξιολογεί τις ιστορίες, εκτιμά την απαιτούμενη προσπάθεια και εργασία για την υλοποίηση κάθε ιστορίας και στη συνέχεια σχεδιάζει ένα χρονοδιάγραμμα (πρόγραμμα) εργασίας.

Στη φάση των επαναλήψεων γίνεται ένας διαχωρισμός του συστήματος που έχει καθοριστεί στην προηγούμενη φάση σε διάφορες επαναλήψεις, κάθε μια από τις οποίες διαρκεί από μια έως τέσσερις εβδομάδες. Η πρώτη επανάληψη περιλαμβάνει τον καθορισμό της αρχιτεκτονικής του συστήματος και στη συνέχεια ακολουθεί η αναβάθμιση του συστήματος με την ενσωμάτωση των ιστοριών. Στο τέλος κάθε επανάληψης δοκιμάζεται το προϊόν από τον πελάτη. Κάθε επανάληψη ολοκληρώνεται με την παραγωγή της έκδοσης που αναπτύχθηκε κατά την επανάληψη.

Στη φάση της παραγωγής διεξάγονται περισσότεροι έλεγχοι και δοκιμές ώστε να εξασφαλιστεί η απόδοση του συστήματος πριν αυτό παραδοθεί στον πελάτη. Κατά τη διάρκεια της φάσης αυτής μπορεί να εντοπιστούν αλλαγές που πρέπει να γίνουν και αποφασίζεται αν θα πραγματοποιηθούν στην παρούσα επανάληψη.

Στη φάση της συντήρησης καταγράφονται καινούριες ιδέες και εισηγήσεις ώστε να είναι δυνατή η υλοποίηση τους σε νεότερες εκδόσεις.

Στη φάση της περαίωσης ο πελάτης δεν έχει άλλες ιστορίες για υλοποίηση. Αυτό συνεπάγεται πως το σύστημα ικανοποιεί όλες τις απαιτήσεις και καταγράφεται η απαραίτητη τεκμηρίωση (documentation) του λογισμικού, ενώ δεν είναι αποδεκτές αλλαγές στην αρχιτεκτονική, στο σχεδιασμό ή στον κώδικα. Επίσης, η περαίωση μπορεί να επέλθει πριν ολοκληρωθεί το σύστημα, εάν παρατηρηθεί πως δεν μπορεί να επιφέρει τα επιθυμητά αποτελέσματα.

Ο ακραίος προγραμματισμός είναι ένα μίγμα ιδεών και πρακτικών επηρεασμένων από υπάρχουσες μεθόδους. Στοχεύει στην ανάπτυξη ενός επιτυχημένου προϊόντος λογισμικού παρά την ασάφεια στον καθορισμό των προδιαγραφών και τις αστάθειες στις απαιτήσεις.

Παρακάτω αναφέρονται βασικές πρακτικές και χαρακτηριστικά του XP:

Χρόνο-Προγραμματισμός του έργου: Εκτιμάται η προσπάθεια που απαιτείται για την υλοποίηση των ιστοριών του πελάτη και σε συνεργασία με τον πελάτη αποφασίζεται το χρονοδιάγραμμα υλοποίησης των ιστοριών, βάσει εκτιμήσεων.

Μικρές εκδόσεις: Η εφαρμογή αναπτύσσεται με την εκτέλεση μιας σειράς μικρών, επαυξητικών εκδόσεων. Οι εκδόσεις μπορεί να αναπτύσσονται από καθημερινά μέχρι και μηνιαία.

Περιγραφή του συστήματος με μεταφορές: Μια μεταφορά (metaphor) αποτελεί μια κωδικοποίηση ή ονοματοθεσία για ένα τμήμα του συστήματος ή μιας λειτουργίας. Το σύστημα ολόκληρο χαρακτηρίζεται από ένα σύνολο μεταφορών μεταξύ πελατών και προγραμματιστών και με τον τρόπο αυτό είναι ευκολότερη η επικοινωνία για το πώς λειτουργεί το σύστημα.

Απλή σχεδίαση: Δίνεται έμφαση κατά τον σχεδιασμό στην απλούστερη δυνατή λύση η οποία και εφαρμόζεται με την αφαίρεση περιττών πολυπλοκοτήτων και πλεονάζοντος κώδικα.

Αναδιάρθρωση του συστήματος (refactoring): Η αλλαγή σε ένα τμήμα κώδικα μπορεί αναπόφευκτα να επιφέρει επιμέρους αλλαγές σε ένα άλλο τμήμα του συστήματος. Το γεγονός αυτό αντιμετωπίζεται με την αναδιάρθρωση του συστήματος, τη βελτίωση της επικοινωνίας, την απλούστευση και την προσθήκη ευελιξίας, χωρίς να αλλάζει η λειτουργικότητα του λογισμικού συστήματος.

Προγραμματισμός σε ζεύγη (Pair programming): Όλος ο κώδικας είναι γραμμένος από δύο προγραμματιστές οι οποίοι δουλεύουν μαζί και ταυτόχρονα σε έναν υπολογιστή. Ο ένας εκ των δύο γράφει κώδικα καθώς ο δεύτερος αναθεωρεί άμεσα τον κώδικα. Ο προγραμματιστής που δακτυλογραφεί φέρει τον ρόλο του οδηγού ενώ αυτός που ελέγχει τον κώδικα ονομάζεται παρατηρητής ή πλοηγός.

Συλλογική ιδιοκτησία: Δεν υπάρχει ένα μοναδικό πρόσωπο που κατέχει ή είναι υπεύθυνο για μεμονωμένους τομείς κώδικα. Ο καθένας μπορεί να αλλάξει οποιοδήποτε μέρος του κώδικα, ανά πάσα στιγμή. Ένα σημαντικό πλεονέκτημα αυτού είναι το ότι επιταχύνεται η όλη διαδικασία ανάπτυξης γιατί εάν ένας προγραμματιστής εντοπίσει ένα λάθος ή ασάφεια στον κώδικα τότε μπορεί άμεσα να προχωρήσει στην διόρθωση του τμήματος αυτού.

Συνεχής ολοκλήρωση: Ένα νέο κομμάτι του κώδικα ενσωματώνεται στο ισχύον σύστημα μόλις είναι έτοιμο. Όταν γίνει η αναβάθμιση του συστήματος με την προσθήκη ενός νέου τμήματος, τότε το σύστημα θα πρέπει να επανελεγχθεί, ώστε οι αλλαγές να γίνουν αποδεκτές.

40 ώρες την εβδομάδα: Ο μέγιστος ρυθμός εργασίας είναι 40 ώρες την εβδομάδα διαφορετικά αντιμετωπίζεται ως πρόβλημα ανάμεσα στην ομάδα ανάπτυξης. Το

νόημα είναι ότι, αν ο άνθρωπος εργάζεται ξεκούραστος τότε δουλεύει πιο δημιουργικά και αποδοτικά.

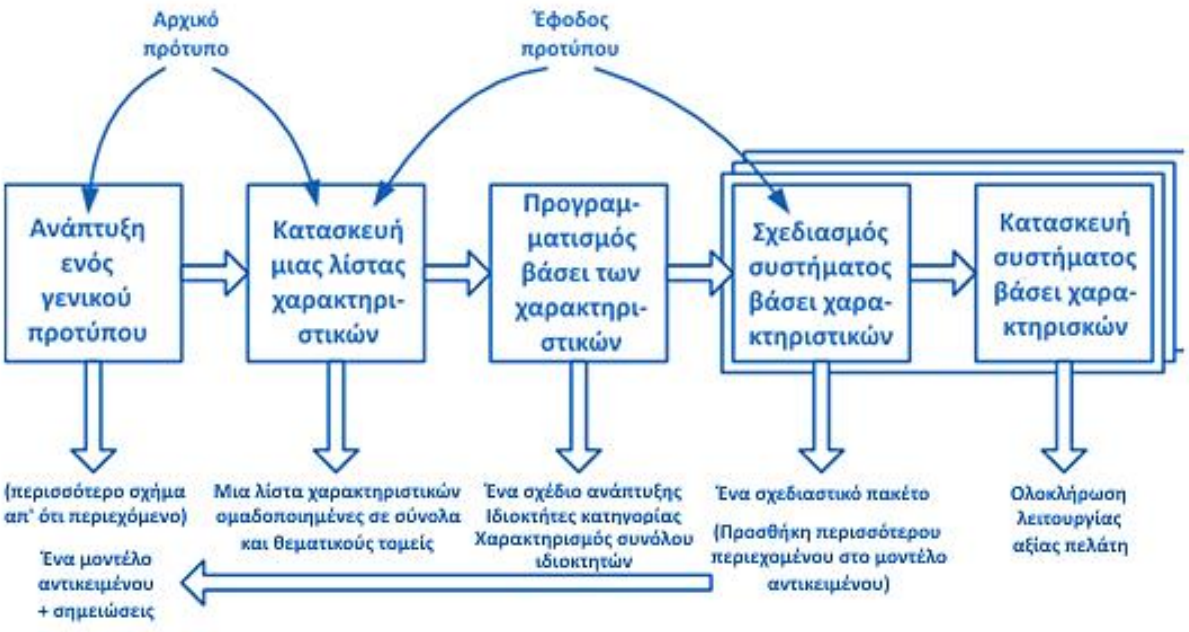
Εμπλοκή του πελάτη: Ο πελάτης κατέχει σημαντικό ρόλο στην ανάπτυξη του λογισμικού. Δεν αντιμετωπίζεται απλά σαν χρηματοδότης αλλά σαν τελικός χρήστης. Ο πελάτης οφείλει να είναι διαθέσιμος ανά πάσα στιγμή για την ομάδα ανάπτυξης, έτοιμος να απαντήσει σε κάθε απορία.

Πρότυπα κωδικοποίησης: Υπάρχουν κανόνες κωδικοποίησης και ακολουθούνται από τους προγραμματιστές έτσι ώστε να υπάρχει συνέπεια και επαρκή επικοινωνία μεταξύ των μελών της ομάδας ανάπτυξης. Τα πρότυπα καθορίζονται συναινετικά.

### 4.5.2. Ανάπτυξη με βάση τα χαρακτηριστικά (Feature Driven Development)

Η ανάπτυξη βάσει χαρακτηριστικών (Feature Driven Development, FDD) περιλαμβάνει πέντε φάσεις από τις οποίες διέρχεται η ανάπτυξη του συστήματος. Οι τρεις πρώτες φάσεις εκτελούνται στην αρχή του έργου ενώ οι δύο τελευταίες φάσεις αποτελούν το επαναληπτικό μέρος της διαδικασίας όπου και παρουσιάζονται οι αρχές της ευέλικτης ανάπτυξης με γρήγορη προσαρμογή σε τυχόν αλλαγές όσον αφορά τις απαιτήσεις. Η προσέγγιση FDD περιλαμβάνει συχνές παραδόσεις και λεπτομερή παρακολούθηση της προόδου του έργου.

Η προσέγγιση αυτή χρησιμοποιήθηκε για πρώτη φορά στην ανάπτυξη μιας πολύπλοκης τραπεζικής εφαρμογής στα τέλη της δεκαετίας του 1990. Σε αντίθεση με τις υπόλοιπες μεθόδους δεν ακολουθεί ολόκληρη την διαδικασία ανάπτυξης λογισμικού αλλά επικεντρώνεται στη φάση του σχεδιασμού και της υλοποίησης. (Palmer, 2002)




**ΔΙΑΓΡΑΜΜΑ 35 Διαδικασία ανάπτυξης με βάση τα χαρακτηριστικά (Palmer, 2002)**

Παρακάτω παρουσιάζονται τα βήματα της διαδικασίας FDD:

Ανάπτυξη ενός γενικού προτύπου (Develop an Overall Model): Με την έναρξη της φάσης αυτής γίνεται μια πρώτη γνωριμία με το πεδίο, το πλαίσιο και τις απαιτήσεις του συστήματος. Ενδέχεται να κατασκευαστούν περαιτέρω τεκμηριωμένες απαιτήσεις όπως οι περιπτώσεις χρήσης και οι λειτουργικές προδιαγραφές. Πραγματοποιείται ένα λεπτομερές «πέρασμα» (walkthrough) του συστήματος και ο επικεφαλής αρχιτέκτονας (software architect) ενημερώνεται αναλυτικά σχετικά με το σύστημα. Το σύστημα διαιρείται σε περαιτέρω υποσυστήματα και διενεργούνται επιμέρους «περάσματα».

Κατασκευή μιας λίστα χαρακτηριστικών (Build a Features List): Παράγεται μια κατηγοριοποιημένη λίστα με τα χαρακτηριστικά για να υποστηρίξει τις απαιτήσεις.

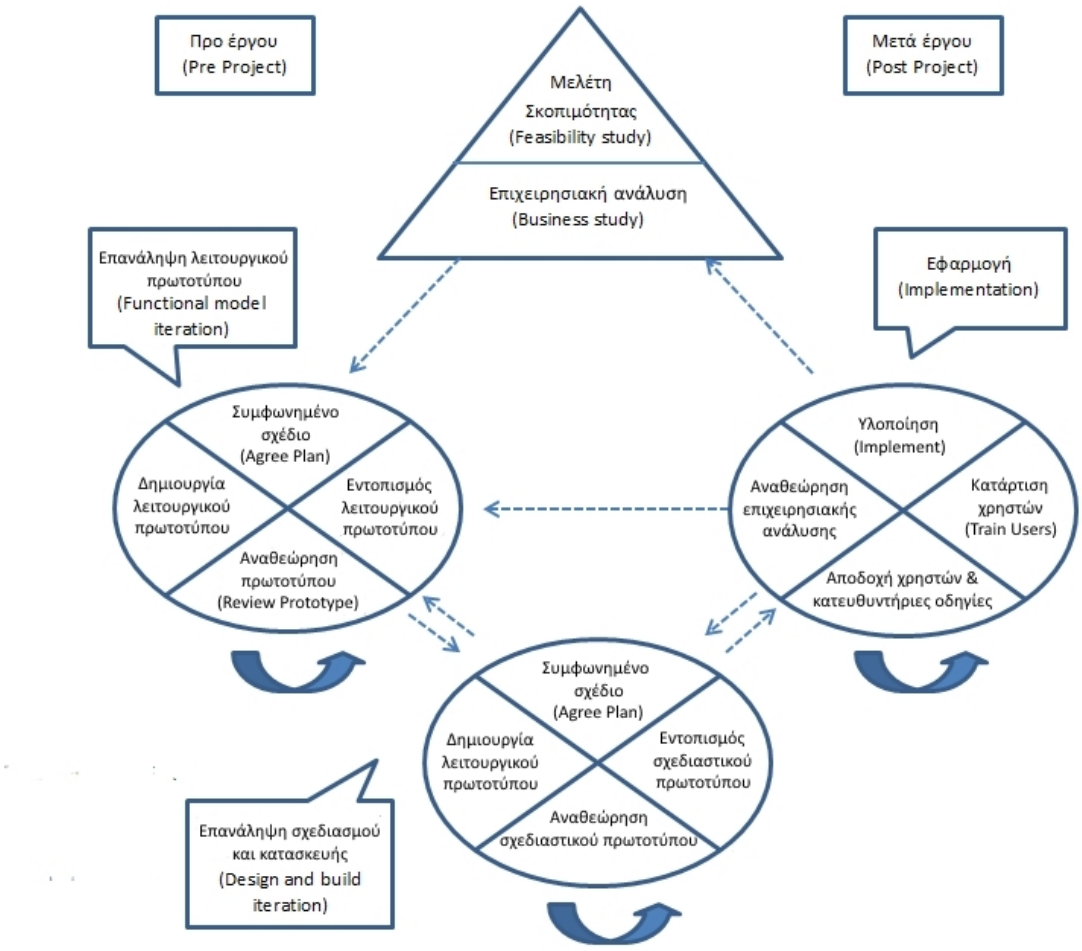
Προγραμματισμός βάσει των χαρακτηριστικών (Plan by Feature): Η ομάδα ανάπτυξης ταξινομεί τα χαρακτηριστικά στοιχεία του συστήματος ανάλογα με τις προτεραιότητες και τις μεταξύ τους εξαρτήσεις και αυτά ανατίθενται σε επικεφαλής προγραμματιστές. Επιπλέον, τα τμήματα που προσδιορίστηκαν στην πρώτη φάση ανατίθενται στους ιδιοκτήτες κατηγορίας (class owners) (μεμονωμένους προγραμματιστές). Επίσης, το χρονοδιάγραμμα και τα ορόσημα τίθενται βάσει των χαρακτηριστικών στοιχείων.

Σχεδιασμός και Κατασκευή του συστήματος βάσει των χαρακτηριστικών (Design & Build by Feauture): Στο στάδιο αυτό επιλέγονται συγκεκριμένα χαρακτηριστικά στοιχεία για υλοποίηση. Οι φάσεις αυτές είναι επαναληπτικές διαδικασίες

αποτέλεσμα των οποίων είναι η υλοποίηση των χαρακτηριστικών που επιλέγονται κάθε φορά. Μια ομάδα αναλαμβάνει να αναπτύξει μέσα από μια επανάληψη κάποιο χαρακτηριστικό στοιχείο. Μπορεί να υπάρχουν πολλές ομάδες που δουλεύουν ταυτόχρονα στην υλοποίηση διαφορετικών στοιχείων. Μια επανάληψη διαρκεί από ορισμένες μέρες μέχρι το μέγιστο δύο βδομάδες. Όταν ολοκληρωθεί μια επανάληψη τότε το σύστημα αναβαθμίζεται με τις καινούριες λειτουργίες και τότε προκύπτει μια νέα έκδοση.

### 4.5.3. Μέθοδος ανάπτυξης δυναμικών συστημάτων (Dynamic Systems Development)

Η μέθοδος ανάπτυξης δυναμικών συστημάτων (Dynamic Systems Development, DSD), εμφανίστηκε για πρώτη φορά στο Ηνωμένο Βασίλειο στα μέσα της δεκαετίας του 1990. Είναι ένα μείγμα αλλά παράλληλα και επέκταση της γρήγορης προτυποποίησης και πρακτικών επαναληπτικής ανάπτυξης. Βασική επιδίωξη στη DSD είναι να καθοριστεί ο χρόνος, οι πόροι, και στη συνέχεια αναλόγως να προσαρμοστεί το επίπεδο της λειτουργικότητας και όχι το αντίστροφο. Το DSDM χωρίζεται σε πέντε φάσεις. (Abrahamsson et al., 2002)




**ΔΙΑΓΡΑΜΜΑ 36 Διαδικασία ανάπτυξης δυναμικών συστημάτων (Stapleton, 1997)**

Οι δύο πρώτες φάσεις εκτελούνται σειριακά και μόνο μια φορά. Οι υπόλοιπες, όπου και αναπτύσσεται το σύστημα, είναι επαναληπτικές και αυξητικές. Ένα σημαντικό χαρακτηριστικό της μεθόδου αυτής είναι ο χρόνος. Οι επαναλήψεις αντιμετωπίζονται σαν χρονικά παράθυρα. Ένα χρονικό παράθυρο έχει συγκεκριμένα όρια και η επανάληψη επιβάλλεται να ολοκληρωθεί μέσα στα χρονικά αυτά όρια. Ένα χρονικό παράθυρο μπορεί να διαρκέσει από μερικές μέρες έως κάποιες εβδομάδες.

1. Μελέτη σκοπιμότητας (Feasibility study): Λαμβάνεται η απόφαση για το αν θα χρησιμοποιηθεί ή όχι η μέθοδος DSD. Αυτό καθορίζεται από το είδος του έργου και, οργανωτικά θέματα και κυρίως τον ανθρώπινο παράγοντα. Στη φάση αυτή παράγονται δύο προϊόντα εργασίας, μια έκθεση σκοπιμότητας και ένα γενικό σχέδιο για την ανάπτυξη.

2. Επιχειρησιακή ανάλυση (Business study): Η συνιστώμενη προσέγγιση σε αυτή τη φάση είναι να κατανοηθεί το επιχειρηματικό πεδίο του έργου. Τα βασικά αποτελέσματα αυτής της ενότητας είναι ο ορισμός της αρχιτεκτονικής του συστήματος και ένα πρωτότυπο σχέδιο αυτού.

3. Επανάληψη λειτουργικού πρωτοτύπου (Functional model iteration): Είναι η πρώτη επαναληπτική φάση. Περιλαμβάνει την ανάλυση, την κωδικοποίηση και την πρωτοτυποποίηση. Τα αποτελέσματα που συλλέγονται χρησιμοποιούνται ώστε να βελτιωθούν τα πρωτότυπα ανάλυσης. Κύριο αποτέλεσμα της φάσης αυτής είναι ένα λειτουργικό πρωτότυπο.

4. Επανάληψη σχεδιασμού και κατασκευής (Design and build iteration): Το σύστημα κτίζεται ουσιαστικά σε αυτή τη φάση. Ο σχεδιασμός και η λειτουργική αξιολόγηση των πρωτοτύπων γίνονται από τους χρήστες και η επιπλέον ανάπτυξη βασίζεται στα σχόλια των χρηστών.

5. Εφαρμογή (Implementation): Το σύστημα παραδίδεται στους χρήστες. Παρέχεται εκπαίδευση των χρηστών όσο αφορά την χρήση του συστήματος. Εγχειρίδια χρήστη και μια έκθεση ανασκόπησης του έργου είναι διαθέσιμα. Ωστόσο, η επαναληπτική και αυξητική φύση της DSD σημαίνει ότι η συντήρηση μπορεί να θεωρηθεί ως συνεχιζόμενη ανάπτυξη. Αντί να τελειώσει το έργο με ένα κύκλο, μπορεί να επιστρέψει σε κάποια από τις προηγούμενες φάσεις, έτσι ώστε το σύστημα να εμπλουτιστεί με τις απαιτήσεις που τυχόν υπολείπονται.

### 4.5.4. SCRUM

Η Scrum είναι μια επαναληπτική και αυξητική μεθοδολογία ευέλικτης ανάπτυξης λογισμικού για τη διαχείριση της ανάπτυξης του προϊόντος. Καθορίζει μια ευέλικτη, ολιστική στρατηγική για την ανάπτυξη του προϊόντος, όπου μια ομάδα ανάπτυξης λειτουργεί ως μονάδα για την επίτευξη ενός κοινού στόχου. Αμφισβητεί τις παραδοχές των παραδοσιακών, διαδοχικών προσεγγίσεων για την ανάπτυξη του προϊόντος, και επιτρέπει στις ομάδες να αυτοοργανώνονται με την ενθάρρυνση της συνεργασίας, φυσικής ή διαδικτυακής όλων των μελών της ομάδας, καθώς και την καθημερινή πρόσωπο-με-πρόσωπο επικοινωνία μεταξύ όλων των μελών της ομάδας και όλων των κλάδων στο έργο. (XP 2008, 2008)

Μια βασική αρχή της scrum είναι η αναγνώριση του ότι κατά τη διάρκεια ενός έργου, οι πελάτες μπορούν να αλλάξουν τις ανάγκες και τις απαιτήσεις τους για το τι θέλουν και χρειάζονται και ότι οι απρόβλεπτες προκλήσεις δεν μπορούν εύκολα να αντιμετωπιστούν με παραδοσιακούς τρόπους πρόβλεψης ή προγραμματισμού. Ως εκ τούτου, η scrum υιοθετεί μια εμπειρική προσέγγιση που δέχεται ότι το πρόβλημα δεν μπορεί να κατανοηθεί ή να οριστεί πλήρως, εστιάζοντας αντ' αυτού στη μεγιστοποίηση της ικανότητας της ομάδας να παραδίδει με ταχύτητα και να ανταποκρίνεται σε αναδυόμενες ανάγκες. (Schwaber, 2004)

Η ύπαρξη της Scrum οφείλεται σε μια ολιστική προσέγγιση των Takeuchi και Nonaka (1986) για την αύξηση της ταχύτητας και ευελιξίας στην ανάπτυξη εμπορικών προϊόντων. Σύμφωνα με την προσέγγιση αυτή, η διαδικασία ανάπτυξης ενός προϊόντος, σε όλες τις επί μέρους φάσεις, που συνήθως επικαλύπτονται, εκτελείται από μία λειτουργική ομάδα που συγκρίνεται με αυτήν του ράγκμπι (από όπου και το όνομα). Δηλαδή, όπως και στο ράγκμπι, όλη η ομάδα σαν μια οντότητα προσπαθεί να διανύσει την απόσταση (δημιουργία του προϊόντος), περνώντας την μπάλα μπρός-πίσω. Το 1995 οι Sutherland και Schwaber θεμελίωσαν την μεθοδολογία σε μια ανακοίνωση-δημοσίευση του συνεδρίου OOPSLA'95. Οι Schwaber και Beedle, το 2001, σε ένα βιβλίο με τίτλο "Ευέλικτη Ανάπτυξη λογισμικού με την Scrum περιγράφουν λεπτομερώς την μέθοδο. (Schwaber, 2004)

Η επιτυχία της μεθόδου οφείλεται στον τρόπο με τον οποίο προσεγγίζει και διαχειρίζεται την έννοια της διαδικασίας ανάπτυξης του λογισμικού. Προσδίδοντας ευελιξία στην οργάνωση και διαχείριση της διαδικασίας ανάπτυξης λογισμικού, αυξάνεται η παραγωγικότητα της ομάδας αλλά και η ικανότητα προσαρμογής της σύμφωνα με τις εκάστοτε ανάγκες που προκύπτουν. Δίνεται μεγάλη έμφαση στο πώς θα πρέπει να οργανωθεί η ομάδα ώστε να μπορέσει να επιτύχει το μέγιστο βαθμό αποτελεσματικότητας, σε ένα διαρκώς μεταβαλλόμενο περιβάλλον. Η μέθοδος προβλέπει ένα εντελώς διαφορετικό μοτίβο οργάνωσης και διοίκησης του έργου σε σχέση με τις υπόλοιπες μεθόδους. Πιο συγκεκριμένα η μέθοδος αυτή επιδιώκει όσο

το δυνατόν μικρότερο χρονικό διάστημα κύκλου ανάπτυξης και παράδοση τμημάτων κώδικα του συστήματος που είχαν συμφωνηθεί ανά κύκλο. Ένα σημαντικό σημείο που θα πρέπει να αναφερθεί είναι η καθημερινή συνεδρίαση των μελών της ομάδας ανάπτυξης με σκοπό την επίτευξη όσο το δυνατόν καλύτερης συνεργασίας και συντονισμού των εργασιών τους.

#### 4.5.4.1. Φάσεις

Η Μέθοδος Scrum διαχωρίζεται σε 3 σημαντικές φάσεις (Pichler, 2010) (“Scrum,” n.d.):

I. Αρχική Διερεύνηση (pre-game)
II. Σχεδιασμός (game)
III. Ολοκλήρωση (post- game)

I. Φάση αρχικής διερεύνησης
Περιλαμβάνει δύο επί μέρους φάσεις. Τη φάση της ανάλυσης και τη φάση του υψηλού επιπέδου σχεδιασμού.

a. Φάση Ανάλυσης

Στη φάση αυτή γίνεται ο καθορισμός και η πιστοποίηση του συστήματος που πρόκειται να αναπτυχθεί. Πιο συγκεκριμένα γίνεται ο καθορισμός των προδιαγραφών του συστήματος καθώς και των απαιτήσεων του χρήστη από αυτό. Σημαντικό ρόλο στη διαδικασία αυτή διαδραματίζει ο πελάτης ο οποίος καλείται να ορίσει με μεγάλη λεπτομέρεια στις απαιτήσεις του για το σύστημα. Σε συνεργασία με τον πελάτη δημιουργείται ένας κατάλογος ανεκτέλεστων προϊόντων (Product Backlog list) που περιέχει όλες τις γνωστές, μέχρι την παρούσα περίοδο, απαιτήσεις του συστήματος. Οι απαιτήσεις κατατάσσονται ανάλογα με την προτεραιότητα τους και υπολογίζεται η προσπάθεια που απαιτείται για την εφαρμογή τους. Ο κατάλογος αυτός ενημερώνεται συνεχώς κατά τη διάρκεια του κύκλου ζωής με νέα και πιο λεπτομερή στοιχεία, καθώς επίσης και με ακριβέστερες εκτιμήσεις για την αναγκαία προσπάθεια και με νέες προτεραιότητες. Έτσι, η ομάδα ανάπτυξης πρέπει να κάνει τα δικά της πλάνα και να ορίσει τις διαδικασίες ανάπτυξης με όσο το δυνατό μικρότερη πιθανότητα εμφάνισης προβλημάτων. Η ομάδα ανάπτυξης έχει το δύσκολο έργο του καθορισμού προτεραιοτήτων και του βαθμού σημαντικότητας των επιμέρους λειτουργιών του συστήματος που πρόκειται να αναπτυχθεί. Οι επιμέρους λειτουργίες αναλύονται μία προς μία, έτσι ώστε να διαπιστωθεί ο βαθμός δυσκολίας υλοποίησής τους όπως και η τυχόν ενσωμάτωση επιπλέον τεχνογνωσίας στη διαδικασία της υλοποίησής τους. Αυτό βέβαια προσδίδει επιπλέον δυσκολία στην ομάδα ανάπτυξης καθώς η ομάδα θα

σπαταλήσει ένα μεγάλο χρονικό διάστημα στην εκπαίδευση κάποιων μελών της ώστε να μπορέσει να αντεπεξέλθει πλήρως στις ανάγκες του έργου.

b. Φάση υψηλού επιπέδου σχεδιασμού

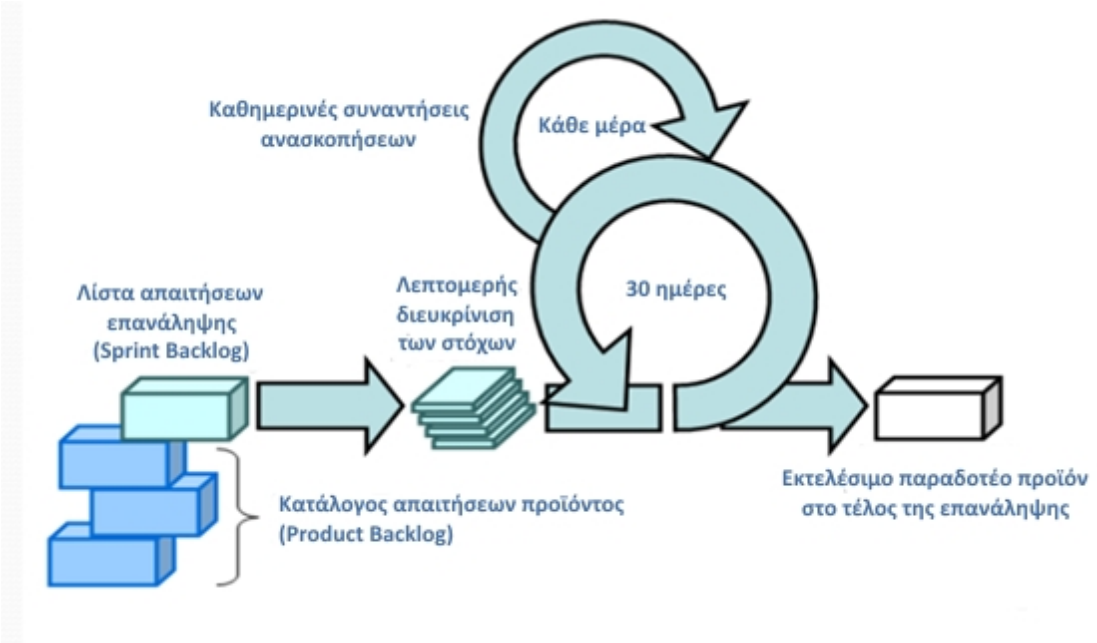
Στη φάση αυτή υλοποιούνται η αρχιτεκτονική και ο σχεδιασμός του συστήματος με βάση τις αρχικές προδιαγραφές του συστήματος οι οποίες έχουν ήδη καθοριστεί στη προηγούμενη φάση.

II. Φάση Ανάπτυξης

Η κρίσιμη αυτή φάση χαρακτηρίζει την ευελιξία της μεθόδου, καθότι η ομάδα ανάπτυξης πρέπει να λάβει υπόψη πολλές παραμέτρους, όπως χρόνος παράδοσης, ποιότητα, απαιτήσεις, πηγές, ενσωμάτωση επιπλέον τεχνογνωσίας κ.α., που ίσως διαφοροποιηθούν κατά τη διάρκεια της ανάπτυξης, αλλά και να είναι ικανή στο να αντιμετωπίσει πολλούς αστάθμητους παράγοντες και κινδύνους. Έτσι, η ομάδα ανάπτυξης θα πρέπει να λάβει τα κατάλληλα μέτρα και να προβεί στους απαραίτητους ελέγχους, ώστε να μπορέσει να αποφύγει τους πιθανούς κινδύνους που παραμονεύουν και να αντιμετωπίσει τα προβλήματα που ίσως προκύψουν.

a. Διαδικασία Ανάπτυξης

Στη φάση αυτή το σύστημα αναπτύσσεται σε πολλές επαναλήψεις (Sprints). Οι επαναλήψεις είναι επαναληπτικοί κύκλοι ανάπτυξης όπου ο κάθε κύκλος συμπεριλαμβάνει όλες τις φάσεις ανάπτυξης, όπως για παράδειγμα τη φάση της ανάλυσης των απαιτήσεων, τη φάση σχεδιασμού, τη φάση υλοποίησης και παράδοσης κ.α. Σε κάθε επανάληψη υλοποιούνται συγκεκριμένες απαιτήσεις, που ονομάζονται απαιτήσεις της επανάληψης (sprint backlog), και έχουν επιλεχθεί από τον κατάλογο των συνολικών απαιτήσεων του προϊόντος, με την βοήθεια του πελάτη. Οι απαιτήσεις αυτές πρέπει θα υλοποιηθούν μέσα στο χρονικό διάστημα της τρέχουσας επανάληψης. Η κάθε επανάληψη διαρκεί ένα συγκεκριμένο χρονικό διάστημα που κυμαίνεται από μία εβδομάδα έως ένα μήνα. Στο τέλος κάθε επανάληψης η ομάδα πρέπει να έχει ένα εκτελέσιμο παραδοτέο προϊόν. Ένα έργο ανάπτυξης λογισμικού συνήθως υλοποιείται και παραδίδεται στον πελάτη σε τρεις έως οκτώ επαναλήψεις.




**ΔΙΑΓΡΑΜΜΑ 37 Κύκλος ζωής της Scrum (Schwaber, 2002)**

III. Φάση Ολοκλήρωσης

Στη φάση της ολοκλήρωσης εκτελούνται όλες εκείνες οι διαδικασίες που απαιτούνται για την περαίωση και την τελική παράδοση του έργου στον πελάτη. Η ομάδα ανάπτυξης αφού επιβεβαιώσει, πάντα σε συνεργασία με τον πελάτη, ότι όλες οι απαιτήσεις έχουν υλοποιηθεί, κινεί τις διαδικασίες παράδοσης του συστήματος.

#### 4.5.4.2. Ρόλοι και ευθύνες

Έξι διακριτοί ρόλοι εμπλέκονται στην ανάπτυξη έργων με την ευέλικτη μεθοδολογία SCRUM. Καθένας από αυτούς τους ρόλους είναι επιφορτισμένος στο να εξυπηρετεί διαφορετικούς σκοπούς και να εκτελεί συγκεκριμένες εργασίες. Έτσι σε ένα έργο Scrum έχουμε ένα διαχειριστή ή ειδικό της Scrum (Scrum Master), τον ιδιοκτήτη ή διαχειριστή του προϊόντος (Product Oowner), την ομάδα Scrum (Scrum team), τον πελάτη (Customer), τον χρήστη (User) και τη διαχείριση (Administration). Στη συνέχεια, παρουσιάζονται αυτοί οι ρόλοι σύμφωνα με τους ορισμούς που έχουν προσδώσει σε αυτούς οι Schwaber και Beedle. (Schwaber, 2004)

Ο διαχειριστής ή ειδικός της Scrum (Scrum Master)

Ο διαχειριστής της SCRUM είναι υπεύθυνος για τη διασφάλιση της ορθής ανάπτυξης του έργου σύμφωνα πάντα με τις πρακτικές, τις αξίες και τους κανόνες που έχουν καθοριστεί κατά το σχεδιασμό του. Για να επιτευχθεί αυτό, αλληλεπιδρά τόσο με την ομάδα ανάπτυξης όσο και με τον πελάτη κατά τη διάρκεια της υλοποίησης του έργου.

Είναι επίσης υπεύθυνος στο να διασφαλίσει ότι τυχόν εμπόδια κατά τη διάρκεια του έργου θα εξαλειφθούν, έτσι ώστε η ομάδα ανάπτυξης να διατηρήσει όσο το δυνατόν ψηλότερα το επίπεδο παραγωγικότητάς της.

Ιδιοκτήτης ή διαχειριστής του προϊόντος (Product Oowner)

Είναι ο υπεύθυνος για το έργο, όσον αφορά τη διαχείριση, τον έλεγχο και την καταγραφή των καταλόγων των απαιτήσεων (backlogs) του έργου. Ο ιδιοκτήτης ή διαχειριστής του προϊόντος επιλέγεται από τον διαχειριστή ή ειδικό της Scrum, τον πελάτη και την διαχείριση. Είναι αυτός που επιλέγει σε συνεργασία με την ομάδα ανάπτυξης ποιες απαιτήσεις θα υλοποιηθούν σε κάθε επανάληψη και με ποια σειρά. Επίσης συμμετέχει ενεργά στην εκτίμηση της προσπάθειας που απαιτείται για την υλοποίηση των απαιτήσεων (χρόνος, εργαλεία κλπ.) και συμβάλει αποφασιστικά στην εξεύρεση λύσεων κατά τη διάρκεια της ανάπτυξης.

Ομάδα ανάπτυξης Scrum (Scrum Team)

Η ομάδα Scrum είναι επιφορτισμένη με την ανάπτυξη του έργου και έχει την εξουσιοδότηση να αποφασίζει και να εκτελεί τις κατάλληλες διαδικασίες και εργασίες που απαιτούνται για την υλοποίησή του. Η ομάδα θα πρέπει να είναι σε θέση να αναδιοργανώνει αποτελεσματικά την λειτουργικότητά της (αυτοελεγχόμενες ομάδες), ώστε να επιτυγχάνονται οι προκαθορισμένοι στόχοι της κάθε επανάληψης. Η ομάδα ανάπτυξης είναι υπεύθυνη για την εκτίμηση του χρόνου μιας επανάληψης, για τα τμήματα που έχουν καθυστερήσει, αλλά και για την επανεξέταση της λίστας καθυστερήσεων του έργου. Επίσης η ομάδα ανάπτυξης συμβάλει στην ανακάλυψη των εμποδίων που θα πρέπει να εξαλειφθούν από το έργο ώστε η διαδικασία ανάπτυξης να συνεχιστεί ομαλά.

Πελάτης (Customer)

Ο πελάτης συμμετέχει ενεργά σε όλη την διαδικασία ανάπτυξης του λογισμικού, ιδιαίτερα στην καταγραφή των απαιτήσεων, στη διαδικασία επιλογής και ανάπτυξής τους, αλλά και στον έλεγχο της καλής λειτουργίας του τελικού προϊόντος. Συνήθως ο πελάτης είναι ταυτόχρονα και ο χρήστης του λογισμικού οπότε εκτός από τις γνώσεις του όσον αφορά το υπό ανάπτυξη προϊόν, μπορεί άμεσα να αναγνωρίσει το παραγόμενο αποτέλεσμα και να ζητήσει την τροποποίησή του όποτε χρειασθεί. Ο πελάτης απαντάει άμεσα στις ερωτήσεις της ομάδας αυξάνοντας έτσι την παραγωγικότητά της.

Διοίκηση

Η διοίκηση είναι υπεύθυνη για την λήψη της τελικής απόφασης καθώς και για τα πρότυπα και τις συμβάσεις που ακολουθούνται στο έργο. Συμμετέχει ενεργά στον καθορισμό των στόχων και των απαιτήσεων του έργου καθώς επίσης και στις

διαδικασίες που έχουν να κάνουν με την επιλογή του κατόχου προϊόντος και τη μείωση των καθυστερήσεων με τη βοήθεια του διαχειριστή της Scrum.

#### 4.5.4.3. Πρακτικές

Η μεθοδολογία Scrum δεν απαιτεί ούτε παρέχει κάποια συγκεκριμένη μέθοδο ανάπτυξης λογισμικού. Όμως απαιτεί ορισμένες πρακτικές διαχείρισης και μερικά εργαλεία στις διάφορες φάσεις της, ώστε να αποφευχθούν οι κίνδυνοι και η πολυπλοκότητα των έργων.

Στη συνέχεια αναλύουμε κάποια από τα χαρακτηριστικά της μεθόδου Scrum σύμφωνα πάντα με τους Schwaber and Beedle.

Κατάλογος ανεκτέλεστων προϊόντων της παραγγελίας (Product Backlog list)

Είναι ένας κατάλογος που περιέχει τις γνωστές, μέχρι την παρούσα περίοδο, απαιτήσεις του συστήματος. Οι απαιτήσεις μπορεί να προέλθουν από τον πελάτη, τις πωλήσεις και το τμήμα μάρκετινγκ, την υποστήριξη πελατών ή τους ίδιους τους υπεύθυνους για την ανάπτυξη λογισμικού. Οι απαιτήσεις κατατάσσονται ανάλογα με την προτεραιότητά τους και υπολογίζεται η προσπάθεια που απαιτείται για την εφαρμογή τους. Ο κατάλογος των απαιτήσεων ενημερώνεται και αναθεωρείται σε κάθε επανάληψη με νέα και πιο λεπτομερή στοιχεία, καθώς επίσης και με ακριβέστερες εκτιμήσεις για την αναγκαία προσπάθεια και τις νέες προτεραιότητες.

Εκτίμηση της προσπάθειας (Effort Estimation)

Η εκτίμηση της προσπάθειας είναι μια επαναληπτική διαδικασία, κατά την οποία τα στοιχεία του καταλόγου των απαιτήσεων εκτιμώνται με μεγαλύτερη ακρίβεια όταν περισσότερες πληροφορίες είναι διαθέσιμες. Υπεύθυνοι για τη διαδικασία αυτή είναι ο διαχειριστής του προϊόντος και η ομάδα ανάπτυξης.

Επανάληψη (Sprint)

Επανάληψη είναι ένας πλήρης κύκλος ανάπτυξης που περιλαμβάνει τον σχεδιασμό, ανάλυση και υλοποίηση συγκεκριμένων απαιτήσεων (sprint backlog). Η ομάδα ανάπτυξης οργανώνει τις εργασίες της έτσι ώστε να παράγει μια νέα εκτελέσιμη και προσαυξητική εφαρμογή σε ένα χρονικό διάστημα που ποικίλει από μία έως τέσσερεις εβδομάδες (30 ημέρες). Στον καθορισμό και υλοποίηση των στόχων μιας επανάληψης συμβάλουν αποφασιστικά οι συνεδριάσεις για το σχεδιασμό των επαναλήψεων (sprint planning meetings), οι καθημερινές συνεδριάσεις Scrum (daily scrum meetings) καθώς και οι συνεδριάσεις για την επανεξέταση της επανάληψης (sprint review meeting).

Συνεδρίαση για το σχεδιασμό της επανάληψης (sprint planning meeting)

Είναι η συνεδρίαση για το σχεδιασμό και την υλοποίηση μιας επανάληψης που οργανώνεται από τον ειδικό της Scrum και υλοποιείται σε δύο φάσεις. Στην πρώτη φάση της συνεδρίασης συμμετέχουν οι πελάτες, οι χρήστες, η διοίκηση, ο ιδιοκτήτης του προϊόντος και η ομάδα ανάπτυξης για να καθορίσουν τους στόχους και την λειτουργικότητα της επόμενης επανάληψης. Στη δεύτερη φάση της συνεδρίασης, όπου συμμετέχουν μόνο ο ειδικός της Scrum και η ομάδα ανάπτυξης, καθορίζεται ο τρόπος με τον οποίο θα υλοποιηθεί η αυξητική ανάπτυξη του προϊόντος κατά την διάρκεια της επανάληψης.

Λίστα απαιτήσεων της επανάληψης (sprint backlog)

Η λίστα αυτή περιέχει απαιτήσεις προς υλοποίηση από τον κατάλογο των ανεκτέλεστων προϊόντων και αποτελεί το σημείο εκκίνησης κάθε επανάληψης. Οι απαιτήσεις αυτές επιλέγονται από την ομάδα ανάπτυξης σε συνεργασία με τον ειδικό της Scrum και τον ιδιοκτήτη του προϊόντος, κατά τη διάρκεια της συνεδρίασης για το σχεδιασμό της επανάληψης, με βάση την προτεραιότητα της κάθε απαίτησης αλλά και τους στόχους που έχουν καθοριστεί για την επανάληψη. Η λίστα παραμένει σταθερή μέχρι την ολοκλήρωση της συγκεκριμένης επανάληψης. Όταν όλα τα στοιχεία της λίστας υλοποιηθούν τότε μια νέα έκδοση του συστήματος παραδίδεται στο πελάτη.

Καθημερινή συνεδρίαση Scrum (daily scrum meeting)

Αυτή η μικρής διάρκειας συνεδρίαση (περίπου δεκαπέντε λεπτών), οργανώνεται σε καθημερινή βάση για να ελέγχεται η πορεία και η απόδοση της ομάδας ανάπτυξης. Η συνεδρίαση αυτή μπορεί να θεωρηθεί και ως συνεδρίαση σχεδιασμού καθώς ελέγχεται τι έχει υλοποιηθεί στο χρονικό διάστημα που μεσολάβησε από την προηγούμενη συνεδρίαση και τι θα πρέπει να υλοποιηθεί στο διάστημα μέχρι την επόμενη. Συζητιούνται όλα τα προβλήματα που έχουν προκύψει καθώς επίσης και τυχόν ελλείψεις ή εμπόδια στη διαδικασία ανάπτυξης. Στη συνεδρίαση αυτή μετέχουν ο ειδικός της Scrum, τα μέλη της ομάδας ανάπτυξης αλλά και μέλη της διοίκησης.

Συνεδρίαση για την επανεξέταση της επανάληψης (sprint review meeting)

Είναι η ανεπίσημη συνεδρίαση της τελευταίας μέρας της επανάληψης, όπου ο ειδικός της Scrum και η ομάδα ανάπτυξης παρουσιάζουν τα αποτελέσματα στην διοίκηση, στους πελάτες, στους χρήστες και στον ιδιοκτήτη του προϊόντος. Οι συμμετέχοντες αξιολογούν την ανάπτυξη του έργου και λαμβάνουν αποφάσεις που αφορούν τις επόμενες λειτουργίες που θα πρέπει να υλοποιηθούν. Αυτή η συγκεκριμένη συνεδρίαση είναι σημαντική για την πορεία του έργου καθώς μπορεί να επιφέρει νέα στοιχεία στον κατάλογο των ανεκτέλεστων προϊόντων ή να σηματοδοτήσει την αλλαγή της κατεύθυνσής του.

Τα χαρακτηριστικά της Scrum σε αντίθεση με τα παραδοσιακά χαρακτηριστικά του μοντέλου του καταρράκτη:

1) Μια δυναμική λίστα των απαιτήσεων του προϊόντος με κατανεμημένες ιεραρχικά εργασίες που πρέπει να γίνουν.

2) Ο σχεδιασμός της υλοποίησης του έργου δια μέσου πολλαπλών επαναλήψεων, με πρώτη αυτήν με την υψηλότερη προτεραιότητα.

3) Μια συνεδρίαση για τον προγραμματισμό της επανάληψης με τον καθορισμό της λίστας των απαιτήσεων που θα υλοποιηθούν.

4) Μια σύντομη καθημερινή συνεδρίαση, όπου ελέγχεται η πρόοδος της ομάδας ανάπτυξης, περιγράφεται και αποδίδεται η επερχόμενη εργασία, και αντιμετωπίζονται τα προβλήματα και πιθανά εμπόδια της ανάπτυξης.

5) Μια συνεδρίαση ανασκόπησης, στην οποία τα μέλη της ομάδας αναλύουν τις περασμένες επαναλήψεις και κάνουν προτάσεις για μελλοντικές βελτιώσεις και αλλαγές.

## 4.6. Συστήματα ελέγχου εκδόσεων (VCS)

Ένα από τα βασικά χαρακτηριστικά της ανάπτυξης λογισμικού είναι ότι ο πηγαίος κώδικας γράφεται σε στάδια. Πρόκειται, όπως αναλυτικά έχει παρουσιαστεί, για μια διαδικασία όπου δοκιμάζονται διάφορες εναλλακτικές, προστίθενται, διορθώνονται και εν τέλει γράφεται περισσότερος κώδικας. Έτσι, δημιουργείται η ανάγκη ύπαρξης ενός συστήματος που να καταγράφει τις διαδοχικές εκδόσεις του κώδικα. Ιδίως όταν η ανάπτυξη περιλαμβάνει πολυπληθείς συνεργαζόμενες ομάδες προγραμματιστών, τότε είναι επιτακτικό ένα σύστημα για τη διαχείριση του κώδικά.

Στο πρόβλημα αυτό έρχονται να απαντήσουν τα συστήματα ελέγχου εκδόσεων (Version Control Systems – VCS) τα οποία είναι λογισμικά συστήματα που διευκολύνουν τη διαχείριση και παρακολούθηση αλλαγών που συμβαίνουν σε οποιοδήποτε έγγραφο, πρόγραμμα, λογισμικό, διαδικτυακές εφαρμογές κ.α. Χαρακτηριστικά παραδείγματα είναι τα Git, Mercurial και Bazaar.

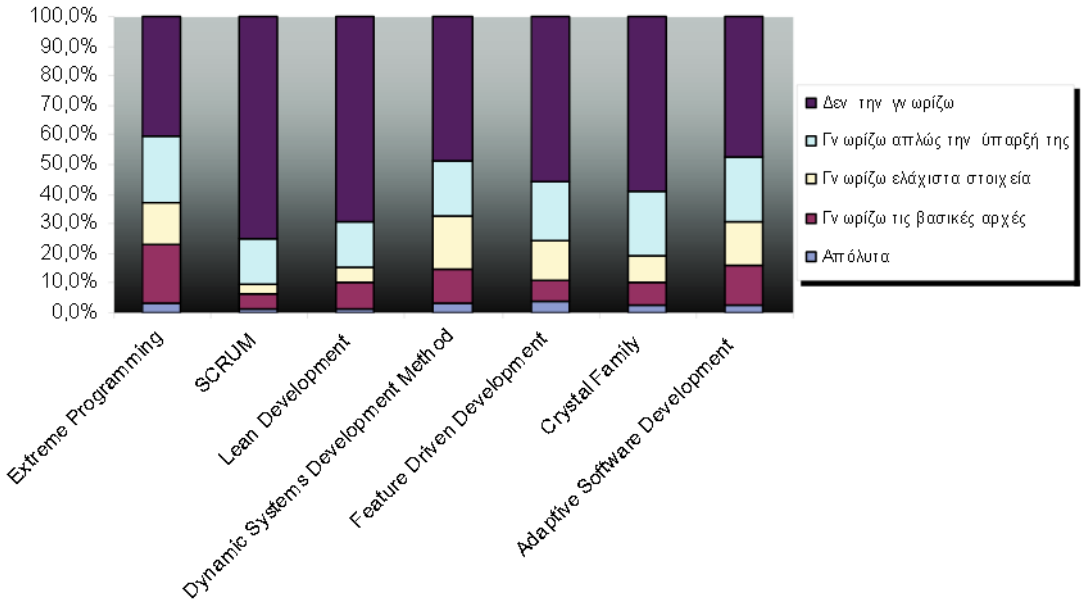
Όσο πολυπλοκότερα είναι τα υπό ανάπτυξη λογισμικά έργα, πολυπληθείς οι εμπλεκόμενες ομάδες προγραμματιστών και ευέλικτη η μεθοδολογία ανάπτυξης τόσο περισσότερο επιτακτική είναι η χρησιμοποίηση των εν λόγω συστημάτων.

Μερικές από τις κύριες ιδιότητες που συνάμα αποτελούν και τα οφέλη από τη χρήση των VCS, είναι:

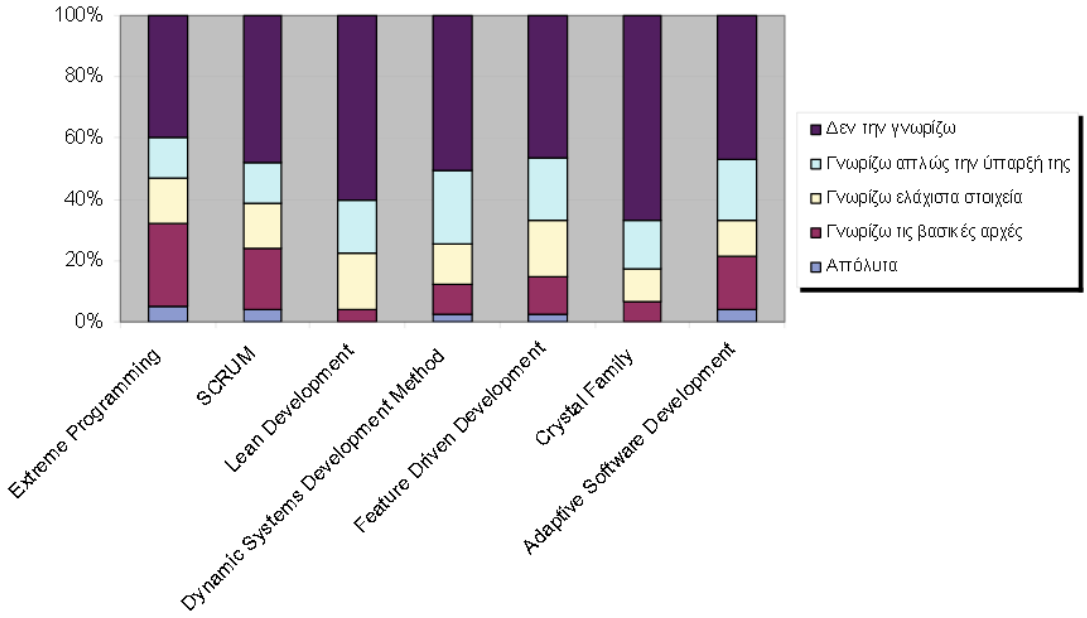
- Η οργάνωση αρχείων που έχουν δημιουργηθεί σε διαφορετικές χρονικές στιγμές (χωρίς συμβάσεις για ονοματοδοσία φακέλων βάσει ημερομηνίας)
- Η ανάκτηση προηγούμενων εκδόσεων και σύγκριση μεταξύ εκδόσεων
- Η διατήρηση αρχείου καταγραφής (log) για τις αλλαγές που έχουν γίνει (εναλλακτικά θα έπρεπε να υπάρχει ένα αρχείο στο οποίο να καταγράφονται τα στοιχεία κάθε αλλαγής που γίνεται)
- Η δυνατότητα εργασίας πολλών ατόμων επί του ιδίου έργου, ανεξαρτήτως της γεωγραφικής τους θέσης ή χρονικής στιγμής
- Η διαχείριση συγκρούσεων (ο χρήστης παρεμβαίνει για να τις επιλύσει)

## 4.7. Ελληνική Πραγματικότητα

Τα παρακάτω διαγράμματα απεικονίζουν τα αποτελέσματα μιας έρευνας σχετικά με την γνώση σχετικά με τις ευέλικτες μεθοδολογίες των ελλήνων εμπλεκόμενων μηχανικών λογισμικού και επιστημόνων πληροφορικής, στην ανάπτυξη πληροφοριακών συστημάτων. Το πρώτο διάγραμμα αφορά έρευνα που διεξήχθη το 2004, ενώ η δεύτερη το 2009.



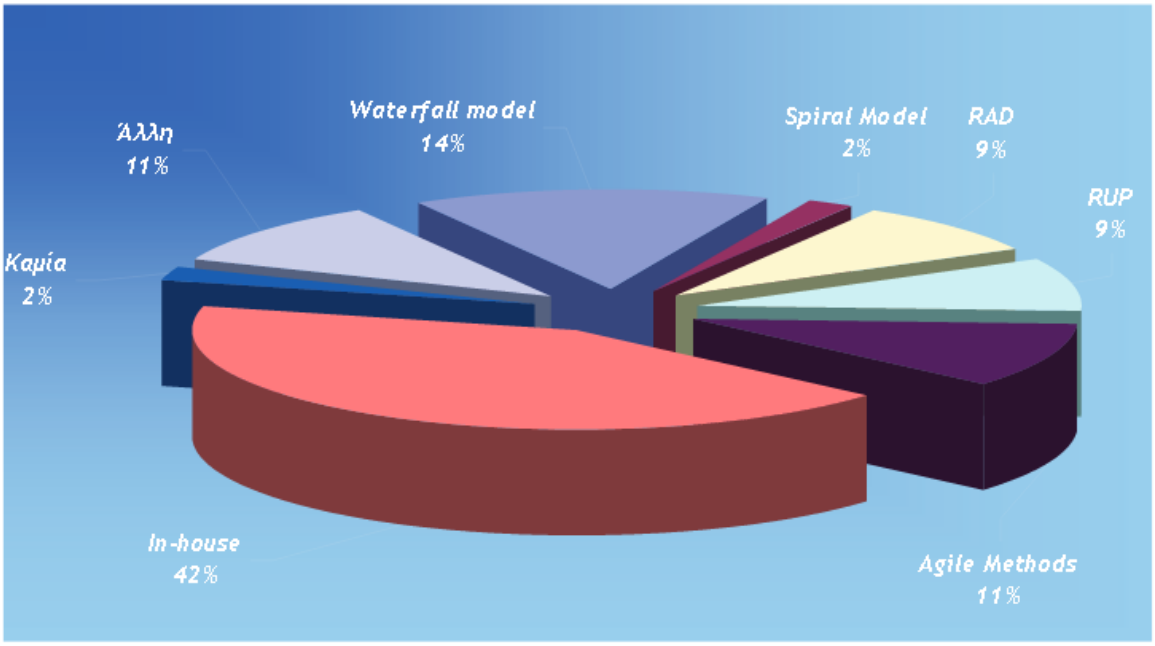

**ΔΙΑΓΡΑΜΜΑ 38 Απεικόνιση αποτελεσμάτων έρευνας του 2004 σχετικά της γνώση ευέλικτων μεθοδολογιών στην Ελλάδα, (" PhD_All Chapters_final_2sided print ready - Monochristou_Phd2011.pdf," n.d.), (σελ. 253)**




**ΔΙΑΓΡΑΜΜΑ 39 Απεικόνιση αποτελεσμάτων έρευνας του 2009 σχετικά της γνώση ευέλικτων μεθοδολογιών στην Ελλάδα ("Microsoft Word - PhD_All Chapters_final_2sided print ready - Monochristou_Phd2011.pdf," n.d.), (σελ 290)**

Είναι ορατή η διείσδυση των μεθοδολογιών αυτών και στην Ελληνική πραγματικότητα ανάπτυξης λογισμικών έργων και ιδιαίτερα της Scrum, ακολουθώντας τις διεθνείς τάσεις.

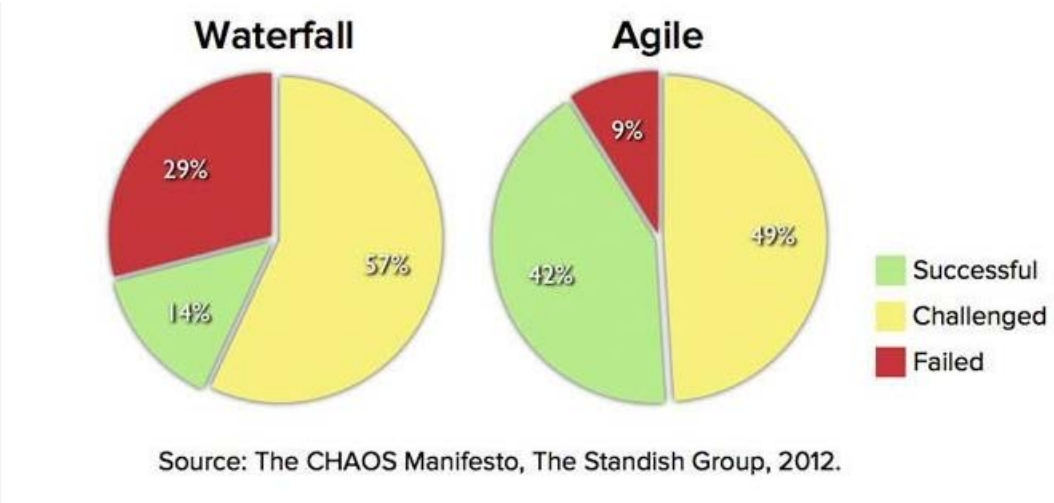
Ακόμα, στα παρακάτω διαγράμματα απεικονίζονται τα ποσοστά χρησιμοποίησης των μεθοδολογιών στον Ελληνικό χώρο, καθώς και το είδος των προς υλοποίηση έργων.




***Διάγραμμα 12** - Ποσοστιαία Κατανομή απαντήσεων στην ερώτηση: «Ως εταιρία ποια μεθοδολογία ανάπτυξης Λογισμικού χρησιμοποιείτε;»*

**ΔΙΑΓΡΑΜΜΑ 40 Ποσοστά έρευνας σχετικά με τη χρησιμοποιούμενη Μεθοδολογίας στην Ελλάδα, 2004 ("Microsoft Word - PhD_All Chapters_final_2sided print ready - Monochristou_Phd2011.pdf," n.d.), (σελ. 272)**

Τέλος, παρατίθενται συγκριτικά τα ποσοστά επιτυχίας στην ανάπτυξη πληροφοριακών συστημάτων με την παραδοσιακή μεθοδολογία Waterfall σε σχέση με την ευέλικτη Agile.




ΔΙΑΓΡΑΜΜΑ 41 Σύγκριση ποσοστών επιτυχίας Waterfall και Agile ("agilevswater.jpg", http://www.codeproject.com/KB/architecture/604417/agilevswater.jpg, n.d.)

# Κεφάλαιο 5

## 5. Ανάπτυξη υποστηρικτικού εργαλείου για εφαρμογές με πολλαπλές συνδέσεις με βάσεις δεδομένων (ABC, Application & dataBase Correlation)

Στο κεφάλαιο αυτό, αναλύεται η προσέγγιση και η διαδικασία σύνθεσης ενός πρωτότυπου υποστηρικτικού εργαλείου για την υποβοήθηση ανάπτυξης ή/και συντήρησης εφαρμογών και πληροφοριακών συστημάτων με πολλαπλές συνδέσεις με βάσεις δεδομένων[1].

Η ιδέα για το λογισμικό, προέκυψε τόσο από θεωρητικά ζητήματα σε σχέση με τις σύγχρονες μεθόδους ανάπτυξης πληροφοριακών συστημάτων όσο και από πρακτική ανάγκη στα πλαίσια έργων του εργαστήριου Συστημάτων Αποφάσεων και Διοίκησης, όπως αναλύεται στη συνέχεια στην αναγνώριση του προβλήματος.

Το λογισμικό ονομάστηκε ABC, από τα αρχικά των λέξεων **A**pplication & data**B**ase **C**orrelation, που περιγράφει τη βασική λειτουργία του, η οποία είναι η συσχέτιση μιας εφαρμογής με τα στοιχεία μιας βάσης δεδομένων.

Η μεθοδολογία που ακολουθήθηκε ήταν περισσότερο κοντά στην παραδοσιακή, ακολουθώντας τα βασικά στάδια της σύλληψης της ιδέας και αναγνώρισης του προβλήματος, του καθορισμού των προδιαγραφών, του σχεδιασμού, της υλοποίησης του εργαλείου, η οποία ήταν και το δυσκολότερο εκ των σταδίων, και τέλος της λειτουργίας του και της δωρεάν διάθεσής του.

[1] Με τον όρο **βάση δεδομένων** ( *database*) εννοείται μία συλλογή από *συστηματικά μορφοποιημένα* σχετιζόμενα δεδομένα στα οποία είναι δυνατή η ανάκτηση δεδομένων μέσω αναζήτησης κατ' απαίτηση. (Elmasri et al., 2007)

## 5.1. Αναγνώριση του προβλήματος

Οι ευέλικτες μέθοδοι και η πιο διαδεδομένη εξ αυτών η scrum, προσπαθούν να αντιμετωπίσουν το δύσκολο και μη προβλέψιμο περιβάλλον της ανάπτυξης πληροφορικών συστημάτων, βασιζόμενες περισσότερο στο ανθρώπινο δυναμικό και τη δημιουργικότητά του και λιγότερο στις διαδικασίες. Συγκεκριμένα για τη Scrum, που αποτελεί μια κοινή πρακτική, το κύριο χαρακτηριστικό της είναι ευέλικτες μικρές ομάδες να ακολουθούν βραχείς επαναληπτικούς κύκλους, σε καθέναν από τους οποίους αποφασίζεται το αντικείμενο του κύκλου, οι κίνδυνοι, η απαιτούμενη προσπάθεια κλπ, ενώ η κάθε επανάληψη διαρκεί περιορισμένο χρονικό διάστημα, συνήθως από μια έως τέσσερις εβδομάδες. Αυτή η επαναληπτική διαδικασία όμως, πέρα από το πλεονέκτημα της άμεσης ανταπόκρισης στις όποιες αλλαγές, είτε προδιαγραφών είτε τεχνολογίας, επιφέρει και ορισμένα μειονεκτήματα. Καθώς το σύστημα αυτό περιέχει ένα σύνολο αλληλεπιδρώντων μεταξύ τους συστατικών στοιχείων, ανθρώπων, υπολογιστών, λογισμικού και διαδικασιών που διέπονται από συγκεκριμένους νόμους, και μάλιστα μερικές φορές όχι σαφώς καθορισμένους, μερικά από τα μειονεκτήματα σχετίζονται με ζητήματα συνεργασίας και διοικητικά θέματα, άλλα είναι τεχνικά και λειτουργικά και άλλα έχουν να κάνουν με εξωτερικές παρεμβάσεις.

Ένα από τα τεχνικά ζητήματα, το οποίο αποτέλεσε την θεωρητική αφορμή για την ανάπτυξη του λογισμικού της διπλωματικής, είναι απόρροια της ίδιας της εφαρμογής της μεθοδολογίας Scrum. Όπως έχει ήδη αναφερθεί, το κεντρικό ερώτημα σε κάθε κύκλο επανάληψης, όταν αποφασίζεται το αντικείμενο εργασίας κατά τη διάρκεια ενός Sprint, είναι η μεγαλύτερη βελτίωση του προϊόντος εκείνη τη συγκεκριμένη στιγμή. Επίσης στη συνέχεια, αφού αυτό έχει αποφασιστεί, θα πρέπει το προϊόν να υλοποιηθεί και ιδανικά να παραδοθεί σε λειτουργική μορφή, γεγονός το οποίο έχει ως αποτέλεσμα σε κάθε επανάληψη μια έκδοση του προϊόντος ή ενός υποσυστήματός του. Η διαδικασία αυτής της επανάληψης, οδηγεί πολλές φορές σε εγγραφή κώδικα με γρήγορο και μη προσχεδιασμένο τρόπο, ο οποίος ενδέχεται στους επόμενους κύκλους λόγω συμμόρφωσης σε κάποια αλλαγή, να μην χρησιμοποιηθεί και να παραμείνει αδρανής, ως πλεονάζον φορτίο, είτε να είναι λανθασμένος και να προκαλεί περαιτέρω αστοχίες, δηλαδή παρεκκλίσεις της συμπεριφοράς του λογισμικού συστήματος από τις αναμενόμενες από τους χρήστες απαιτήσεις. Το γεγονός αυτό κατά την εφαρμογή της εν λόγω μεθοδολογίας, θεωρείται εμπόδιο στην ορθή συνέχιση και ολοκλήρωση του έργου. Επομένως, προκύπτει η ανάγκη ύπαρξης βοηθητικών λογισμικών εργαλείων, τα οποία θα παρέχουν αρωγή στο έργο της εύρεσης των στοιχείων εκείνων που δεν χρησιμοποιούνται, αλλά και διευκόλυνσης της αντιστοίχισης μεταξύ των στοιχείων που χρησιμοποιούνται. Με τον τρόπο αυτό τεκμηριώνεται η θεωρητική ανάγκη δημιουργίας ενός τέτοιου εργαλείου, όπως το ABC.

Το εργαλείο αυτό, σε μια πιο ενισχυμένη μορφή του, θα μπορούσε μέσω της αντιστοίχισης μεταξύ των χρησιμοποιούμενων στοιχείων μιας εφαρμογής να βοηθήσει στον εντοπισμό σφαλμάτων - ένα αναπόφευκτο πρόβλημα καθώς εμπλέκεται ο ανθρώπινος παράγοντας σε κάθε φάση από τη σύλληψη της ιδέας μέχρι την λειτουργία και τη συντήρηση-, που εισάγονται κατά τη διαδικασία ανάπτυξης, και πολλές φορές οδηγούν τα πληροφοριακά συστήματα σε αστοχίες. Το πρόβλημα, μάλιστα, εντείνεται με δεδομένο ότι τα συστήματα αυτά ολοένα και περισσότερο αναπτύσσονται και συντηρούνται από πολλαπλές και συχνά γεωγραφικά διεσπαρμένες ομάδες τεχνικών λογισμικού, οι οποίες σε πολλές περιπτώσεις εργάζονται ταυτόχρονα. Κατά συνέπεια, η ταχεία ανάδραση με βάση τους μηχανισμούς διασφάλισης της ποιότητας και την ανίχνευση σφαλμάτων είναι εξαιρετικά σημαντική, καθώς βοηθά στο να εντοπιστούν και να εξαλειφτούν νωρίς τα λάθη κατά την ανάπτυξη και συντήρηση του λογισμικού, και να προληφθούν περαιτέρω αρνητικές επιπτώσεις.

Στο σημείο αυτό, δίδεται η ευκαιρία για την επεξήγηση της διαφοράς μεταξύ της αξιοπιστίας και ορθότητας ενός πληροφοριακού συστήματος. Ένα σύστημα μπορεί να είναι αξιόπιστο αλλά να μην είναι σωστό. Παράδειγμα το λογισμικό μπορεί να περιέχει σφάλματα, αλλά αν δεν εκτελέσει τα σφάλματα αυτά, τότε θα λειτουργήσει αξιόπιστα. Από την άλλη πλευρά, αν ορίσουμε ως ορθότητα τη συμμόρφωση του κώδικα στις προδιαγραφές, τότε ένα σύστημα μπορεί να είναι σωστό, αλλά να μην είναι αξιόπιστο, επειδή για παράδειγμα ο χρήστης μπορεί να προσπαθήσει να χρησιμοποιήσει το σύστημα με τρόπους που δεν υπήρχαν στις προδιαγραφές, με αποτέλεσμα την ενδεχόμενη κατάρρευση του συστήματος. (“MSWE 609 Syllabus,” n.d.) Σε κάθε περίπτωση τα οφέλη από την ανάλυση των σφαλμάτων και των αστοχιών του λογισμικού έχουν ευρέως αναγνωριστεί. Ωστόσο, είναι σπάνιες λεπτομερείς μελέτες που να βασίζονται σε εμπειρικά δεδομένα.

Αξίζει να σημειωθεί ότι υπάρχει μια σημαντική αδυναμία των τρεχόντων τεχνικών δοκιμών εντοπισμού σφαλμάτων σε λογισμικά συστήματα, η οποία αυξάνεται δυσανάλογα με την πολυπλοκότητα του συστήματος. Ενώ, ταυτόχρονα υπάρχουν περιορισμοί στην ικανότητά μας για την αυτόματη αξιολόγηση του λογισμικού σε σχέση με τα σφάλματα που μπορεί να περιέχει. Αυτό καθιστά την αποτελεσματική ανίχνευση των σφαλμάτων λογισμικού μια σημαντική δραστηριότητα της διαδικασίας ανάπτυξης λογισμικού. Επίσης, δεν υπάρχει μία τεχνική ανίχνευσης βλαβών λογισμικού ικανή να αντιμετωπίσει όλα τα προβλήματα ανίχνευσης σφαλμάτων. Ακόμα, υπάρχει ο κίνδυνος οι επιμέρους αστοχίες συχνά να προκαλούνται από πολλαπλές βλάβες διεσπαρμένες σε όλο το σύστημα. Η παρατήρηση αυτή είναι σημαντική, καθώς η εύρεση και διόρθωση σφαλμάτων που οδηγούν σε τέτοιες αποτυχίες λογισμικού σε μεγάλα και πολύπλοκα συστήματα είναι σαφώς χρονοβόρες, δύσκολες και απαιτητικές εργασίες, παρά τις προόδους στην ανάπτυξη λογισμικού και των υποστηρικτικών εργαλείων. Οι τρεις πιο κοινοί τύποι των σφαλμάτων λογισμικού είναι τα σφάλματα απαιτήσεων, κωδικοποίησης, και δεδομένων. Επιπλέον, σε αντίθεση με τη δημοφιλή πεποίθηση, ένα σημαντικό

ποσοστό των αποτυχιών συνδέονται με τις τελευταίες δραστηριότητες του κύκλου ζωής. ("IEEE Xplore Abstract - Common Trends in Software Fault and Failure Data," n.d.)

Παράλληλα, πρακτική ανάγκη δημιουργίας ενός τέτοιου εργαλείου προέκυψε εντός του πλαισίου έργων και δραστηριοτήτων του εργαστήριου Συστημάτων Αποφάσεων και Διοίκησης. Έργα που αφορούν εκσυγχρονισμό και βελτιστοποίηση μεθόδων πληροφοριακών συστημάτων, με την προσθήκη νέων διαδικασιών, απλοποίηση και βελτιστοποίηση παλαιότερων και αφαίρεση των παρωχημένων. Πρόκειται για πολύπλοκα πληροφοριακά συστήματα με εκτενείς υπάρχουσες βάσεις δεδομένων, γεγονός που καθιστά την όλη διαδικασία εκσυγχρονισμού δυσανάλογα πολύπλοκη, λαμβάνοντας υπόψη και το ότι η ανάγνωση και κατανόηση υπάρχοντος κώδικα είναι πιο δύσκολη από ότι η εκ νέου σύνταξή του.

Ένα αρχικό βήμα της όλης διαδικασίας του εκσυγχρονισμού, του οποίου τα αποτελέσματα είναι κρίσιμα για τη συνέχεια, είναι η συσχέτιση των διαφόρων εφαρμογών του πληροφοριακού συστήματος με τα αντίστοιχα στοιχεία της βάσης δεδομένων. Με άλλα λόγια, αναζητούνται τα σημεία στον κώδικα της εφαρμογής στα οποία εμφανίζονται και χρησιμοποιούνται οι πίνακες της βάσης δεδομένων. Ο ρόλος της συσχέτισης είναι διττός. Από τη μια, η διευκόλυνση του επανελέγχου της εφαρμογής με την παράδοση της πληροφορίας των σημείων αντιστοίχισης του κώδικα με τα στοιχεία της βάσης δεδομένων, και από την άλλη, η άμεση εικόνα των μη χρησιμοποιούμενων στοιχείων της βάσης. Ουσιαστικά, τα αποτελέσματα αποτελούν έναν χάρτη της συσχέτισης μεταξύ μιας εφαρμογής πληροφοριακού συστήματος με τους πίνακες της βάσης δεδομένων της, ο οποίος δυνητικά μπορεί να φανεί χρήσιμος σε όσους έχουν να κάνουν με την ανάπτυξη, τη συντήρηση και τη βελτίωση του συστήματος αυτού.

Η πραγματοποίηση του ελέγχου της συσχέτισης με μη αυτοματοποιημένο τρόπο, δεδομένου ότι οι βάσεις δεδομένων αριθμούν εκατοντάδες πίνακες και οι εφαρμογές μερικές χιλιάδες γραμμές κώδικα, αποτελεί μια ιδιαίτερα χρονοβόρα και κοπιώδη εργασία, επιρρεπή σε ανθρώπινα σφάλματα και παραλείψεις, ενώ η σημασία των αποτελεσμάτων είναι κρίσιμη για την περαιτέρω συνέχιση του εκσυγχρονιστικού έργου. Το γεγονός αυτό, σε συνδυασμό με το ότι η αναζήτηση για δωρεάν έτοιμο εργαλείο απέβη άκαρπη, οδήγησε στην απόφαση της κατασκευής ενός υποστηρικτικού σχετικού εργαλείου, αρχικά στο έργο αυτό, αλλά με στόχο την μελλοντική εφαρμογή του σε οποιοδήποτε αντίστοιχη περίπτωση.

Η ανάπτυξη του εργαλείου ακολούθησε τα στάδια του παραδοσιακού κύκλου ζωής ενός λογισμικού έργου, δηλαδή από τον καθορισμό των προδιαγραφών, το σχεδιασμό και την υλοποίηση, τον έλεγχο και τις δοκιμές, μέχρι και την έκδοση και λειτουργία του, και μελλοντικά την συντήρηση και εξέλιξη του.

## 5.2. Έρευνα αγοράς

Μετά από την αναγνώριση του προβλήματος, η επόμενη κίνηση ήταν η έρευνα αγοράς στο διαδίκτυο για το αν υπάρχει κάποιο έτοιμο λογισμικό εργαλείο που να πληροί τις παραπάνω απαιτήσεις, αρχικά δωρεάν και εκ των υστέρων και με κάποιο κόστος. Από τα αποτελέσματα της έρευνας αγοράς θα κρινόταν και η απόφαση δημιουργίας ή μη του εργαλείου.

Από την έρευνα δεν βρέθηκε κάποιο εργαλείο σχετικά με τις συγκεκριμένες απαιτήσεις, ούτε δωρεάν ούτε επί πληρωμή. Τα πλησιέστερα προϊόντα στο πρόβλημα έρχονται να προσεγγίσουν τμήματα του εν λόγω ζητήματος, αλλά μόνο για εφαρμογές Microsoft access, με τη μορφή πρόσθετων εργαλείων (add-ins, πχ. http://www.databasedev.co.uk/ms-access-products.html). Ακόμα, βρέθηκαν λιγοστές σελίδες υποστήριξης σχετικές με τμήματα του όλου θέματος (www.stackoverflow.com).

Τελικό βήμα ήταν η διερεύνηση των δυνατοτήτων των υπαρχόντων εργαλείων αυτοματοποιημένης επαναδόμησης κώδικα (Automated code refactoring), όπως τα:

- WebStorm (for JavaScript)
- IntelliJ IDEA (for Java)
- Eclipse (for Java, and to a lesser extent, C++, PHP, Ruby and JavaScript)
- NetBeans (for Java)
- JDeveloper (for Java)
- Embarcadero Delphi
- Visual Studio (for .NET and C++)
- ReSharper (addon for Visual Studio)
- CodeRush (addon for Visual Studio)
- Visual Assist (addon for Visual Studio with refactoring support for C# and C++)
- DMS Software Reengineering Toolkit (Implements large-scale refactoring for C, C++, C#, COBOL, Java, PHP and other languages)
- Xcode (for C, Objective-C, and Swift)[citation needed]
- AppCode (for Objective-C, C and C++)
- Smalltalk Refactoring Browser (for Smalltalk)
- Wing IDE (for Python)
- PyDev (for Python)

Κάποια από τα παραπάνω εργαλεία παρέχουν τη δυνατότητα ανεύρεσης μη χρησιμοποιούμενου κώδικα, ωστόσο δεν βρέθηκαν δυνατότητες σχετικά με τη χαρτογράφηση της διεπαφής μεταξύ μιας εφαρμογής και των πινάκων της βάσης δεδομένων της. Αυτό οδήγησε στη λήψη της απόφασης σχετικά με την υλοποίηση ενός τέτοιου υποστηρικτικού εργαλείου, ώστε να διευκολύνεται το έργο των

εμπλεκομένων στην ανάπτυξη και συντήρηση πληροφοριακών συστημάτων με σύνθετες βάσεις δεδομένων.

## 5.3. Καθορισμός των προδιαγραφών

Οι προδιαγραφές προέκυψαν από την ανάγκη για έναν ακριβή έλεγχο για τη συσχέτιση της χρησιμοποίησης ή μη των πινάκων μιας εκτενούς βάσης δεδομένων σε ένα συγκεκριμένο πληροφοριακό σύστημα και της αντιστοίχισης μεταξύ των πινάκων και των αρχείων στα οποία αναφέρονται. Ακολουθώντας την top-down προσέγγιση, αναγνωρίστηκαν πρώτα οι στόχοι, συγκεκριμένα το επιθυμητό παραδοτέο αποτέλεσμα της συσχέτισης των πινάκων της βάσης δεδομένων με τον κώδικα της εφαρμογής, που δεν ήταν άλλο παρά η παρουσίαση με δομημένο τρόπο της εμφάνισης από τη μία του κάθε πίνακα στο αντίστοιχο αρχείο της εφαρμογής, αναγράφοντας εκ παραλλήλου και τον αριθμό των εμφανίσεων εντός αυτών, και από την άλλη σε ξεχωριστή παρουσίαση των αδρανών πινάκων.

Επιπλέον, κρίθηκε απαραίτητο να γίνει με τέτοιο τρόπο η υλοποίηση της εφαρμογής, ώστε να καθίσταται εύκολη η επεκτασιμότητα με πρόσθετα χαρακτηριστικά για μελλοντικές ανάγκες σχετικές με το ίδιο αντικείμενο. Ακόμα, κρίθηκε ωφέλιμο να υπάρχει η δυνατότητα επαρκούς παραμετροποίησης ώστε να μπορεί να φανεί χρήσιμη και σε μελλοντικά έργα.

Η επιλογή του υπάρχοντος έργου στο οποίο θα δοκιμαζόταν η εφαρμογή έπαιξε ρόλο στη διαμόρφωση των προδιαγραφών, καθώς επρόκειτο για έργο σε γλώσσα προγραμματισμού C# με βάση δεδομένων σε SQL Server. Για να υπάρχει μια αίσθηση του όγκου του υπάρχοντος έργου και της βάσης δεδομένων επιγραμματικά αναφέρεται ότι το εγκατεστημένο λογισμικό κομμάτι του καταλαμβάνει χώρο 233MB στον σκληρό δίσκο με 400 επιμέρους αρχεία τύπου “.cs”, ενώ η βάση δεδομένων 2,11GB με περίπου 140 πίνακες.

Συγκεκριμένα, οι προδιαγραφές χωρίστηκαν σε τρεις κατηγορίες:

- Σύνδεση με τη βάση δεδομένων, όπου αποφασίσθηκε ότι θα έπρεπε να δίνει τη δυνατότητα σύνδεσης με βάση δεδομένων σε SQL server
- Επιλογή της υπό εξέταση εφαρμογής, όπου απαραίτητη θα ήταν αφενός η δυνατότητα επιλογής της υπό εξέτασης εφαρμογής, αλλά και επιλογής των επιθυμητών προς εξέταση αρχείων.
- Παρουσίαση των αποτελεσμάτων, όπου η δομημένη παρουσίασή τους σε μορφή πίνακα ήταν η βέλτιστη μεταξύ των επιλογών.

Στη συνέχεια ακολούθησε ο σχεδιασμός του εργαλείου.

## 5.4. Σχεδιασμός του λογισμικού εργαλείου

Ο σχεδιασμός του εργαλείου επικεντρώθηκε σε βασικά θέματα με σκοπό τη λήψη αποφάσεων σχετικών με τη μετέπειτα διευκόλυνση του επόμενου σταδίου της υλοποίησης, δηλαδή της γραφής πηγαίου κώδικα.

Αρχικώς, καθορίστηκε γλώσσα προγραμματισμού η C#, με δεδομένο ότι το εργαλείο προορίζεται να λειτουργεί σε περιβάλλον Windows, λόγω του ότι κρίθηκε πιο κατάλληλη για τις ανάγκες υλοποίησης, από άποψη δυνατοτήτων αλλά και διαθέσιμης στο διαδίκτυο υποστήριξης.

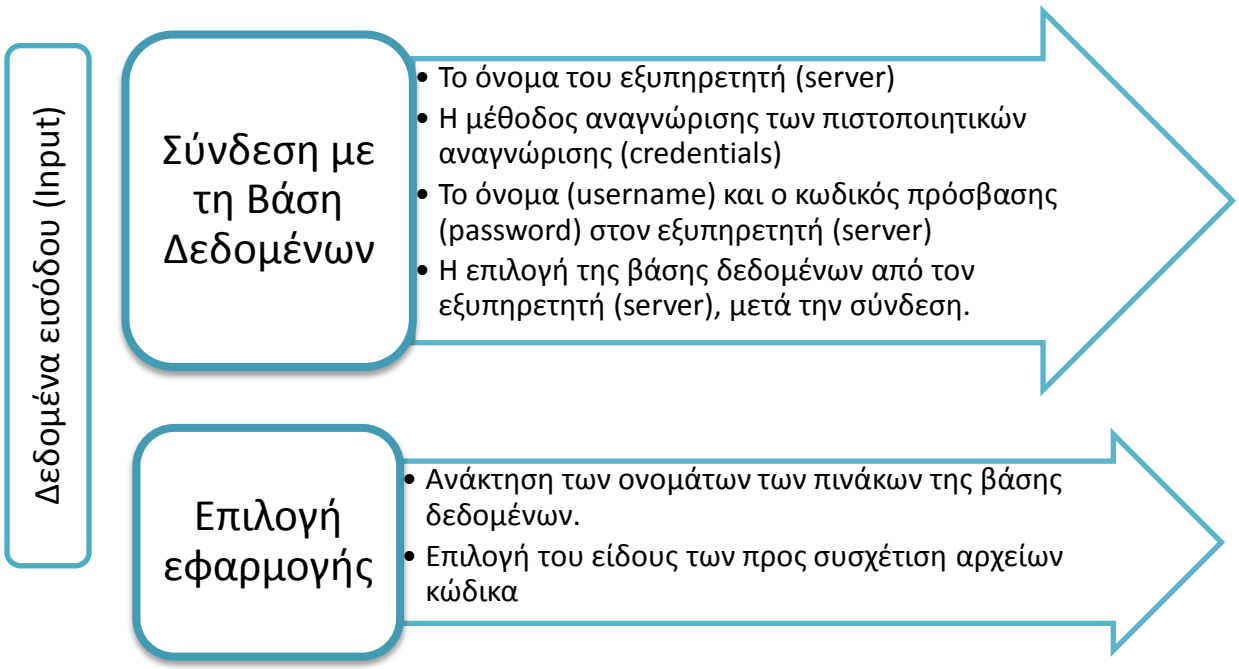
Έπειτα επιλέχθηκε ως περιβάλλον ανάπτυξης (IDE), το Microsoft Visual Studio 2013 με δεδομένο ότι είναι το πιο πλήρες, και διατίθεται δωρεάν μέσω του προγράμματος διανομής λογισμικού σε φοιτητές από τη Microsoft Dreamspark, διαμέσου της συνδρομής του ΕΜΠ και αποτέλεσε μια άριστη ευκαιρία για τον γράφοντα η εξοικείωση με αυτό. Επίσης, για λόγους συμβατότητας κατά την μετεγκατάσταση της βάσης δεδομένων επιλέχθηκε το SQL Server 2014 Management Studio, επίσης διαθέσιμο μέσω του προγράμματος διανομής λογισμικού σε φοιτητές από τη Microsoft Dreamspark.

Εν συνεχεία, ακολούθησε ο γενικός αρχιτεκτονικός σχεδιασμός με γνώμονα την εργονομικότητα της γραφικής διεπαφής χρήστη (GUI), την ευκολία της χρήσης, την απλότητα παρουσίασης των δεδομένων και την ελαχιστοποίηση των απαιτούμενων οδηγιών χρήσης. Ακολουθώντας τις προδιαγραφές, θεωρήθηκε ότι αυτές εξυπηρετούνται αν σε ένα ενιαίο παράθυρο (window) του εργαλείου επιμεριστούν οι λειτουργίες σε αντίστοιχες καρτέλες (tabs). Έτσι, αναλογούν από μια καρτέλα στη σύνδεση με τη βάση δεδομένων, στην επιλογή της εφαρμογής και δύο στην παρουσίαση των αποτελεσμάτων, μία για τον πίνακα που θα παρουσιάζει τα δεδομένα συσχέτισης πινάκων και εφαρμογής, και μια για την εμφάνιση των αδρανών πινάκων.

Το στάδιο του σχεδιασμού ολοκληρώθηκε με το διάγραμμα ροής και τον καθορισμό των δεδομένων εισόδου και δεδομένων εξόδου, καθώς και του αλγορίθμου για τη συσχέτιση, όπως φαίνεται στο διάγραμμα.

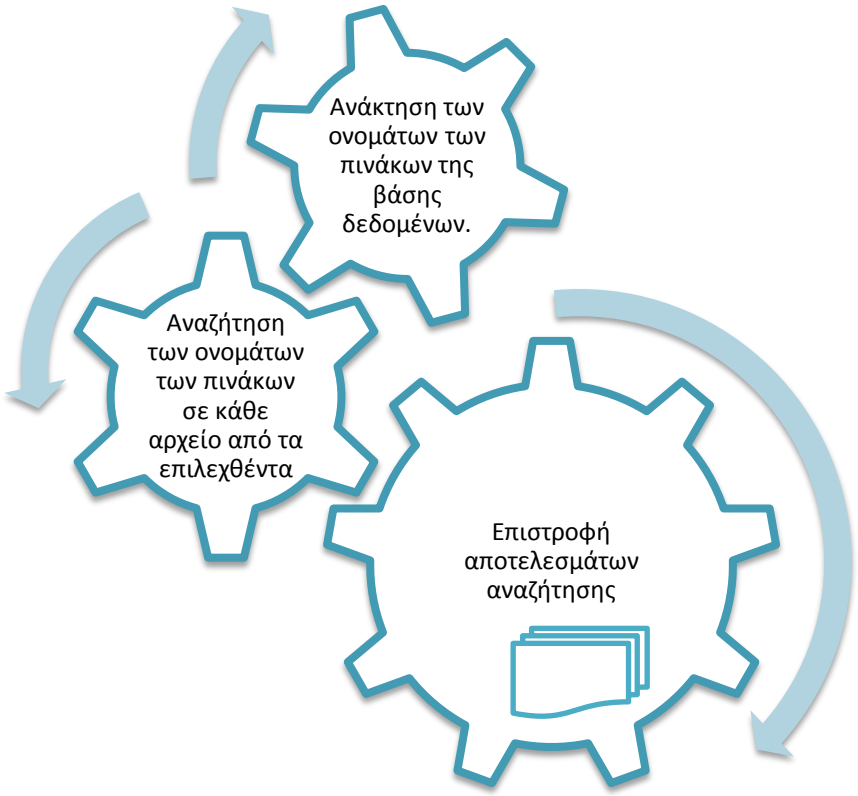

**ΔΙΑΓΡΑΜΜΑ 42 Διάγραμμα ροής υποστηρικτικού εργαλείου**

Αναλυτικά τα δεδομένα εισόδου είναι:



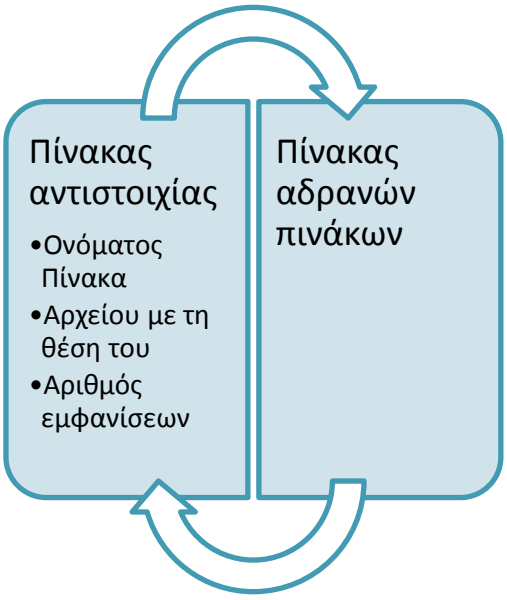

**ΔΙΑΓΡΑΜΜΑ 43 Δεδομένα εισόδου υποστηρικτικού εργαλείου**

Ο αλγόριθμος επεξεργασίας των δεδομένων εισόδου περιλαμβάνει τις εξής λειτουργίες:




**ΔΙΑΓΡΑΜΜΑ 44 Αλγόριθμος επεξεργασίας υποστηρικτικού εργαλείου**

Τέλος τα δεδομένα εξόδου:




**ΔΙΑΓΡΑΜΜΑ 45 Δεδομένα εξόδου υποστηρικτικού εργαλείου**

Με την ολοκλήρωση της φάσης του σχεδιασμού, ξεκίνησε η υλοποίηση του εργαλείου.

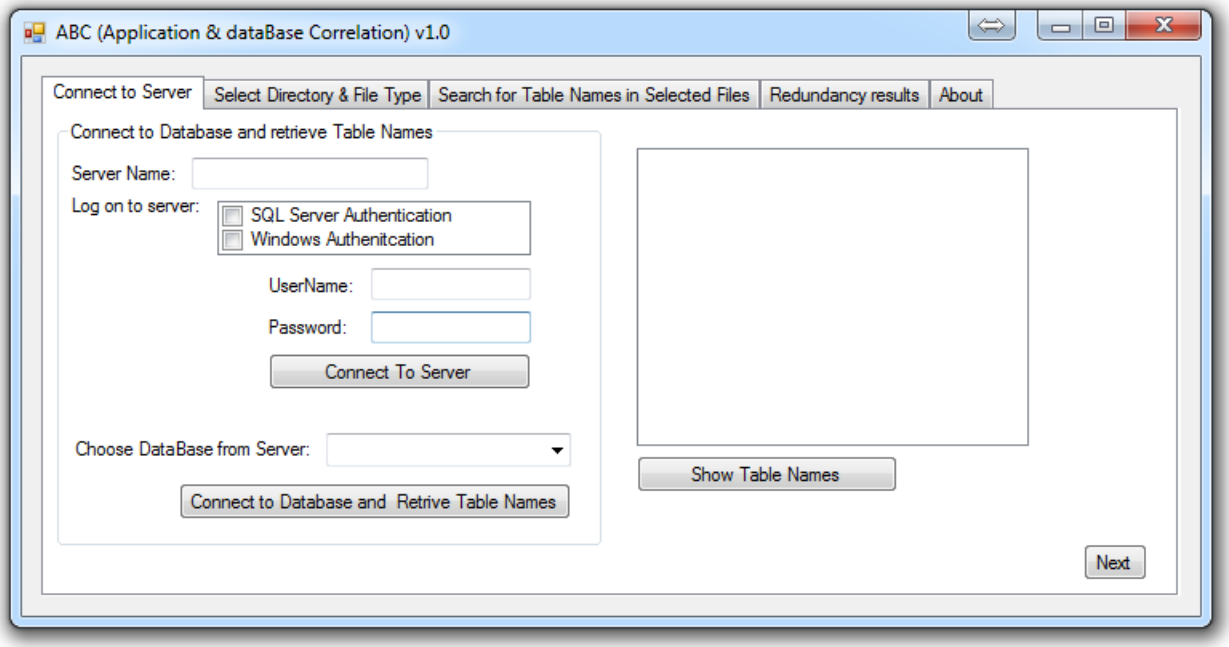
## 5.5. Υλοποίηση του λογισμικού εργαλείου

Η υλοποίηση, δηλαδή η γραφή κώδικα στη γλώσσα προγραμματισμού C#, ακλούθησε τον αρχικό σχεδιασμό της εφαρμογής, με τη μέθοδο code and fix. Στόχος ήταν η ελαχιστοποίηση της πολυπλοκότητας και η πρόβλεψη αλλαγών και επεκτάσεων με την κατάλληλη δομή του κώδικα, αλλά και τον επαρκή σχολιασμό του, ώστε να διευκολύνεται η κατανόηση του από τρίτους και κατ' επέκταση η συμπλήρωση, η διόρθωση και η επαναχρησιμοποίησή του.

### 5.5.1. Γραφικό Περιβάλλον

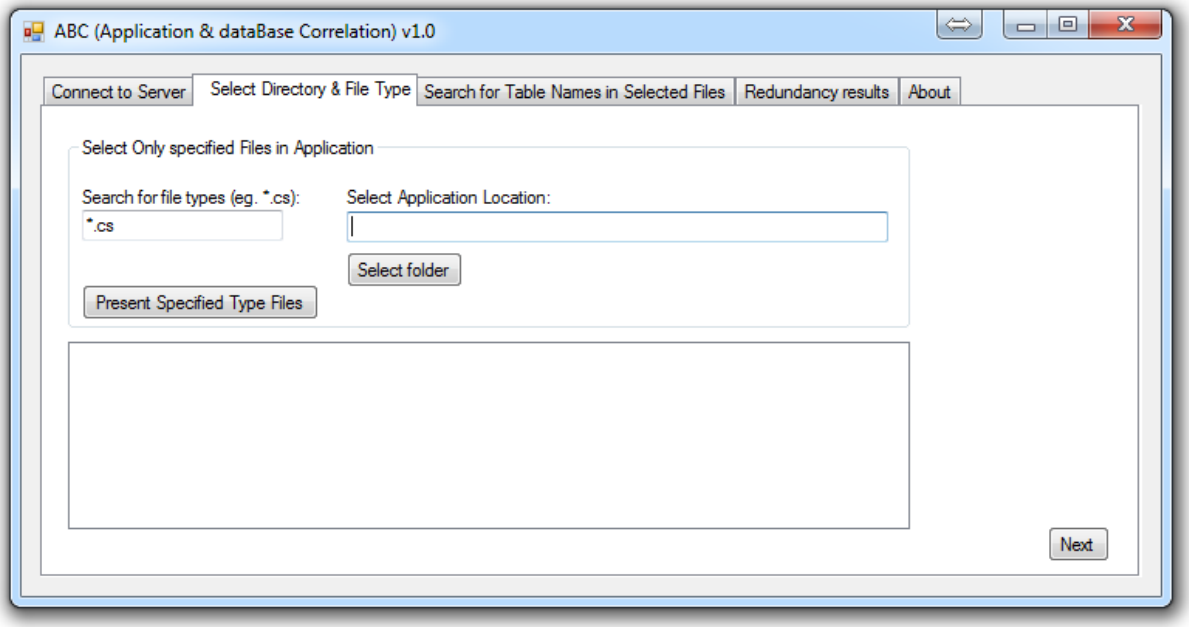
Αρχικά, υλοποιήθηκε η διεπαφή χρήστη (GUI) βάσει του σχεδιασμού, δηλαδή τέσσερις καρτέλες σε ένα παράθυρο, κάθε μια από τις οποίες αντιστοιχεί στα επιμέρους στοιχεία του υπό κατασκευή λογισμικού , όπως αρχικά είχε προβλεφθεί:

Η πρώτη καρτέλα αφορά την ανάκτηση των ονομάτων των πινάκων μιας βάσης δεδομένων αφού πρώτα εξασφαλίζεται η σύνδεση με τον εξυπηρετητή (server), και είναι έτσι σχεδιασμένη ώστε να δίνει με εύκολο τρόπο τη δυνατότητα εισαγωγής των δεδομένων εισόδου από τον τελικό χρήστη. Επίσης, έχει προβλεφθεί στη δεξιά πλευρά για λόγους επαλήθευσης από τον χρήστη, εφόσον το επιθυμεί, η δυνατότητα παρουσίασης των ονομάτων των πινάκων.



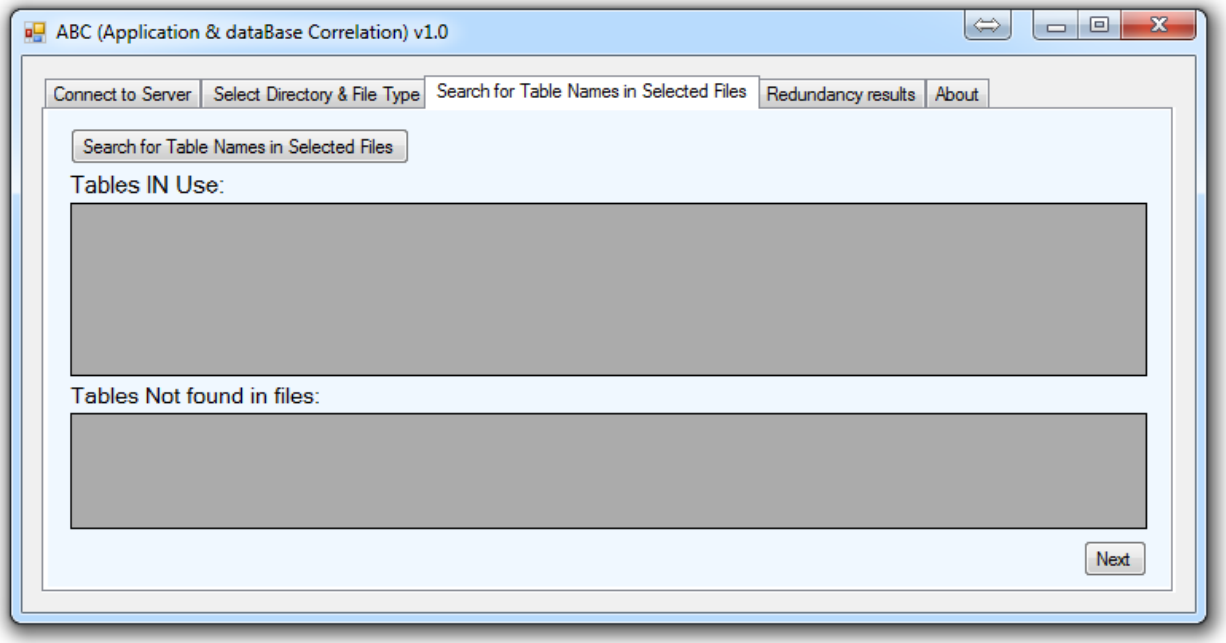

**Εικόνα 1 Υλοποίηση γραφικού περιβάλλοντος εργαλείου –καρτέλα 1η (Σύνδεση με βάση δεδομένων)**

Η δεύτερη καρτέλα καλύπτει τα υπόλοιπα δεδομένα εισόδου από τον τελικό χρήστη, παρέχοντας τη δυνατότητα αφενός της επιλογής της εφαρμογής και αφετέρου του τύπου αρχείων κώδικα που θα ελεγχθούν. Ουσιαστικά, επιλέγεται η τοποθεσία που είναι εγκατεστημένη η εφαρμογή και εντός αυτής, με τον καθορισμό του είδους των υπό εξέταση αρχείων, ανακτώνται τα ονόματα των αρχείων. Η επιλογή του τύπου αρχείων δίνει την πολύτιμη δυνατότητα ελέγχου εφαρμογών ανεξαρτήτως γλώσσας προγραμματισμού. Επιπρόσθετα, εφόσον ο χρήστης επιθυμεί υπάρχει η πρόβλεψη παρουσίασης των ονομάτων των εφαρμογών για λόγους εποπτείας και επιβεβαίωσης.

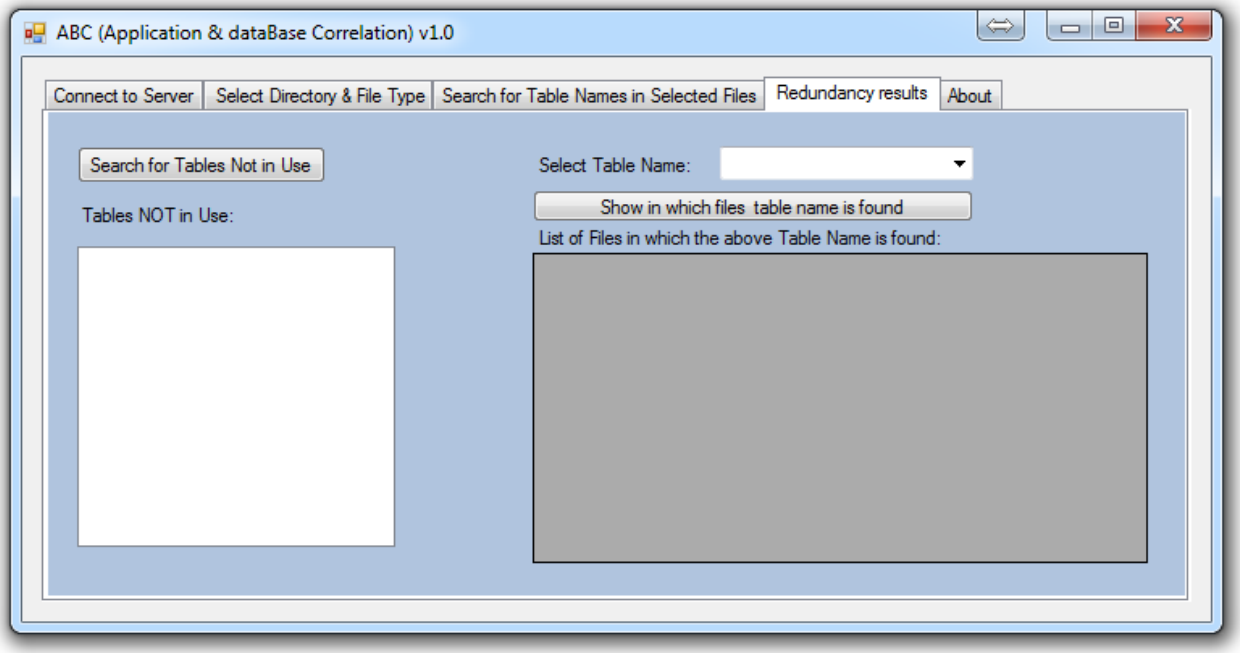



**Εικόνα 2 Υλοποίηση γραφικού περιβάλλοντος εργαλείου –καρτέλα 2η (Επιλογή εφαρμογής και τύπου αρχείων)**

Στις επόμενες δυο καρτέλες παρουσιάζονται τα αποτελέσματα της συσχέτισης μεταξύ των αρχείων της εφαρμογής και της βάσης δεδομένων. Αναλυτικότερα, στην τρίτη καρτέλα εμφανίζονται σε μορφή πίνακα τα ονόματα των πινάκων ανά εμφάνιση τους στα αντίστοιχα αρχεία της εφαρμογής καθώς και σε πρωτογενή μορφή η διασταύρωση της μη χρησιμοποίησης πίνακα ανά αρχείο εφαρμογής. Στην τέταρτη καρτέλα, γίνεται φιλτράρισμα του προαναφερθέντα πρωτογενή πίνακα και παρουσιάζονται οι πίνακες που δεν χρησιμοποιούνται καθόλου από την εφαρμογή. Επίσης, δίνεται η δυνατότητα κατόπιν επιλογής ονόματος πίνακα, να εμφανίζονται τα ονόματα των αρχείων στα οποία εμφανίζονται.




**Εικόνα 3 Υλοποίηση γραφικού περιβάλλοντος εργαλείου –καρτέλα 3η (Αποτελέσματα συσχέτισης εφαρμογής και βάσης δεδομένων)**




**Εικόνα 4 Υλοποίηση γραφικού περιβάλλοντος εργαλείου –καρτέλα 4η (Παρουσίαση μη χρησιμοποιούμενων στοιχείων)**

### 5.5.2. Πηγαίος Κώδικα

Στη συνέχεια ακολούθησε το στάδιο της δημιουργίας πηγαίου κώδικα. Έπειτα από πλήθος δοκιμών, ο κώδικας δομήθηκε σε 5 γενικές ενότητες (regions) σε αντιστοιχία των απαιτούμενων σταδίων βάσει του σχεδιασμού. Επιγραμματικά, αναφέρονται οι ενότητες αυτές όπως εμφανίζονται στο εργαλείο, και αναφέρονται οι κύριες εντολές που αποτελούν και τον κορμό του προγράμματος, χωρίς τις συντακτικές τους λεπτομέρειες. Το εργαλείο υπάρχει στην πλήρη του μορφή στο συνοδευτικό cd του τεύχους της εν λόγω διπλωματικής.

`1. "connect to database and retrieve all table names"`

Η ενότητα αποτελεί τον κώδικα που υποστηρίζει τη λειτουργικότητα της πρώτης καρτέλας. Κύριες εντολές είναι η `SqlConnection` ("SqlConnection Constructor (System.Data.SqlClient)," n.d.) για τη σύνδεση με τον server , η `SqlCommand` ("SqlCommand Class (System.Data.SqlClient)," n.d.) και η `SqlDataReader` ("SqlDataReader Class (System.Data.SqlClient)," n.d.) σε συνδυασμό με άλλες για την ανάκτηση των ονομάτων των πινάκων από την επιλεγμένη βάση δεδομένων.

`2. "select file type and search for file names of this type in the selected directory"`

Η ενότητα αυτή υποστηρίζει τις λειτουργίες της δεύτερης καρτέλας, με κύρια εντολή τη μέθοδο `Directory.GetFiles` ("Directory.GetFiles Method (System.IO)," n.d.) για την ανάκτηση των ονομάτων επιλεγμένου τύπου αρχείων από τη θέση της εφαρμογής.

`3. "search table names in the files of the above file type"`

Στην ενότητα αυτή που αντιστοιχεί στην τρίτη καρτέλα, ουσιαστικά, διαδραματίζεται όλη η επεξεργασία των δεδομένων, κατά την οποία ελέγχεται κάθε όνομα πίνακα με κάθε γραμμή κώδικα στα επιλεγμένα αρχεία της εφαρμογής και στη συνέχεια γίνεται η αντιστοίχιση. Κύριες εντολές είναι η `foreach (var fileName in currentFilesList)` σε συνδυασμό με την `foreach (string tableName in currentTablesList)`, οι οποίες εξασφαλίζουν ότι ο έλεγχος θα γίνει για κάθε όνομα πίνακα σε κάθε αρχείο, ενώ το κλειδί ελέγχου είναι η εντολή η `if (Regex.IsMatch (allRead, tableName))` ("Regex Class (System.Text.RegularExpressions)," n.d.), όπου διαπιστώνει ακριβώς τον συσχετισμό. Τα αποτελέσματα εν συνεχεία εμφανίζονται σε ένα `DataGrid` ("DataGrid Class (System.Windows.Forms)," n.d.), δηλαδή σε μια μορφή οργανωμένου πίνακα.

`4. "Distinguish only the not used tables"`

Στην ενότητα αυτή που αντιστοιχεί στην τέταρτη καρτέλα φιλτράρονται τα προηγούμενα αποτελέσματα και παρουσιάζονται μόνο εκείνοι οι πίνακες που δεν χρησιμοποιούνται καθόλου, με κύρια εντολή την `if (!currentTablesInUseList.Contains(tableName))`.

5. “Show files where the table name is found”

Στην ενότητα αυτή , που επίσης υποστηρίζει την λειτουργία της τέταρτης καρτέλας, αναλαμβάνει την εμφάνιση της αντιστοίχισης του επιλεγμένου εκ των εν χρήσει πινάκων στα αντίστοιχα αρχεία.

Η φάση της υλοποίησης διήρκεσε τον περισσότερο αναλογικά χρόνο σε σχέση με τις υπόλοιπες και είχε επαναληπτικό και αυξητικό χαρακτήρα.

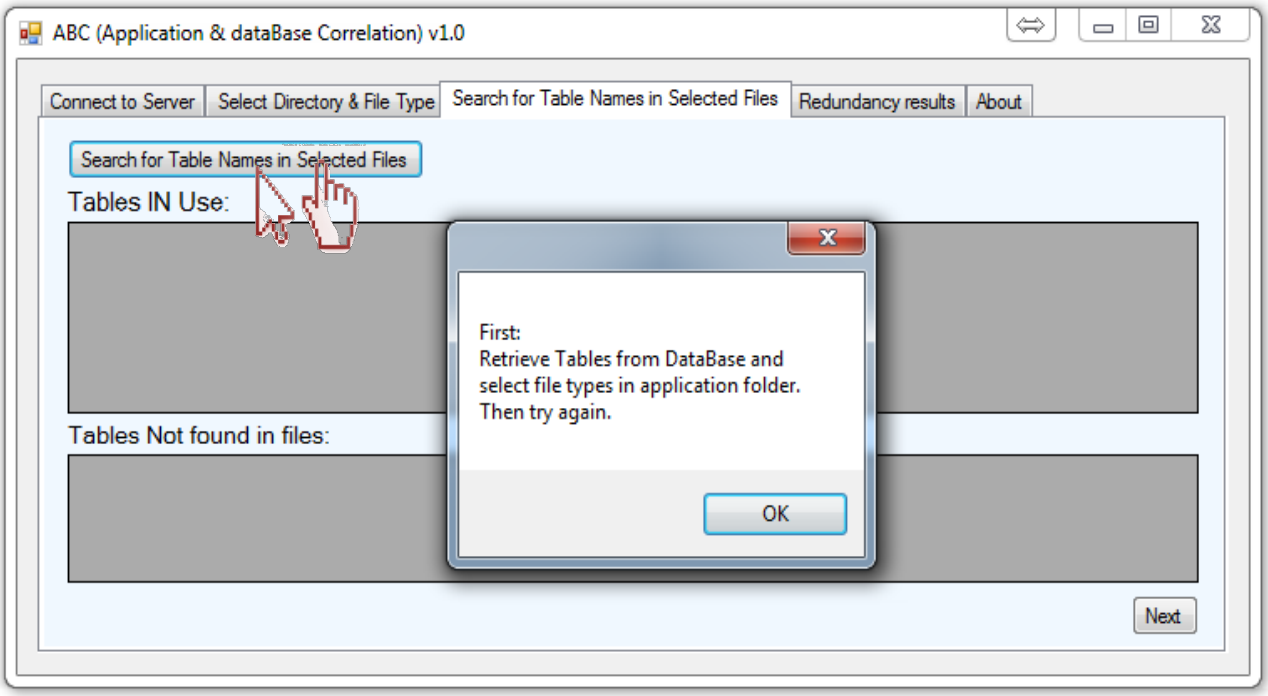
Η εφαρμογή στην πλήρη της μορφή, με ελεύθερο τον κώδικα, υπάρχει διαθέσιμη είτε στο συνοδευτικό cd είτε στην ιστοσελίδα: https://sites.google.com/site/ABC2015tool/

## 5.6. Έλεγχος και διόρθωση σφαλμάτων

Κατ΄ ουσία, η φάση του ελέγχου και των δοκιμών ήταν ενσωματωμένη στους επαναληπτικούς κύκλους της φάσης υλοποίησης, μέσω της πρακτικής «δοκιμής και λάθους» (trial and error). Μετά την προσθήκη του κάθε επιπλέον στοιχείου στον κώδικα, διενεργείτο έλεγχος σε πραγματικές συνθήκες και ακολουθούσε η διαδικασία της αποσφαλμάτωσης (debugging).

Επιπροσθέτως, έμφαση δόθηκε στην πρόβλεψη λανθασμένης σειράς ενεργειών από μεριάς του τελικού χρήστη, με τη βοήθεια αναδυόμενων παραθύρων, με την εκ των προτέρων πρόβλεψη και τοποθέτηση μηνυμάτων με σύντομες σχετικές οδηγίες. Ιδιαίτερα χρήσιμη για τον χειρισμό εξαιρέσεων και λανθασμένης χρήσης του εργαλείου, αποτέλεσε η εντολή `try-catch` (“try-catch (C# Reference),” n.d.).

Αν για παράδειγμα δεν ακολουθηθεί με τη σειρά η ολοκλήρωση των λειτουργιών κάθε καρτέλας και ένας χρήστης μεταβεί κατευθείαν στην τρίτη, κάνοντας κλικ στο κουμπί “Search for Table Names in Selected Files” το εργαλείο δεν θα μπορέσει να ανταποκριθεί, αφού δεν έχουν προηγουμένως καθοριστεί η βάση δεδομένων και η θέση της εφαρμογής. Η αντιμετώπιση περιπτώσεων λάθους χειρισμού γίνεται με την εμφάνιση αναδυόμενων παράθυρών, τα οποία πληροφορούν σχετικά τον χρήστη.




Εικόνα 5 Παράδειγμα αντιμετώπισης λάθος χειρισμού του εργαλείου

## 5.7.Έκδοση και διάθεση λογισμικού εργαλείου

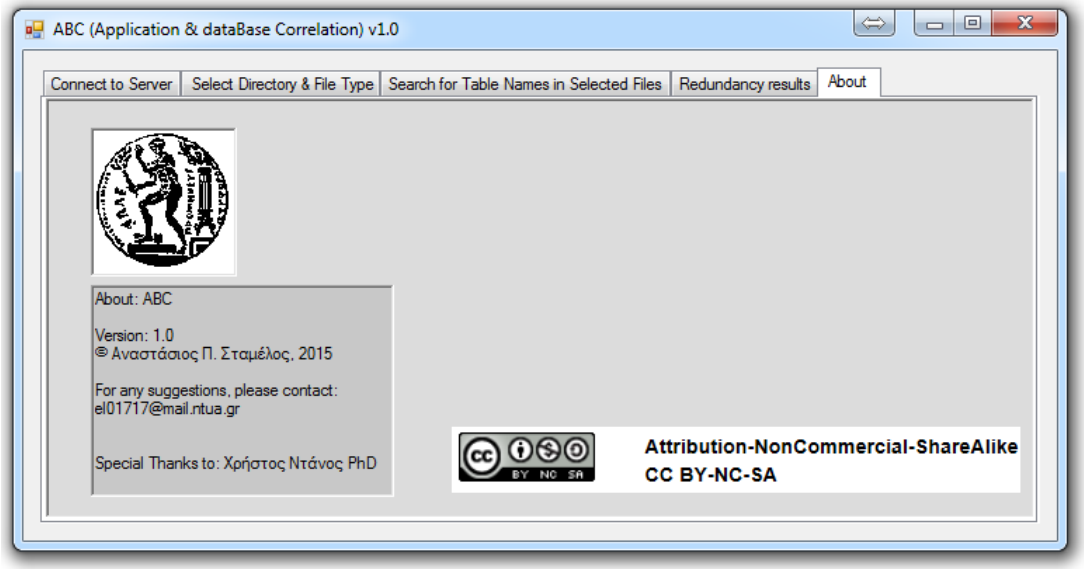
Κατά αρχάς, η έκδοση του εργαλείου γίνεται με την άδεια διάθεσης περιεχομένου CC BY-NC-SA, σύμφωνα με τις άδειες Creative Commons (CC 3.0) σχετικά με τα δικαιώματα πνευματικής ιδιοκτησίας. (“Άδειες Διάθεσης Περιεχομένου - Creative Commons 3.0 | MY-AOC,” n.d.)

Τα αρχικά της άδειας CC BY-NC-SA σημαίνουν Αναφορά - Μη Εμπορική - Παρόμοια Διανομή (Attribution-NonCommercial-ShareAlike CC BY-NC-SA), δηλαδή ο αποδέκτης της άδειας μπορεί να χρησιμοποιήσει το έργο όπως θέλει, εφόσον:

•διατηρήσει τις διατυπώσεις που προβλέπονται στην άδεια σχετικά με την αναφορά στον αρχικό Δικαιούχο

•δεν υπάρχει σκοπός εμπορικής χρήσης

•αδειοδοτήσει οποιοδήποτε παράγωγο έργο με την ίδια άδεια.




**Εικόνα 6 Έκδοση και άδεια διάθεσης εργαλείου**

Η διάθεση του εργαλείου γίνεται δωρεάν διαμέσου του διαδικτυακού ιστοτόπου:

https://sites.google.com/site/ABC2015tool/

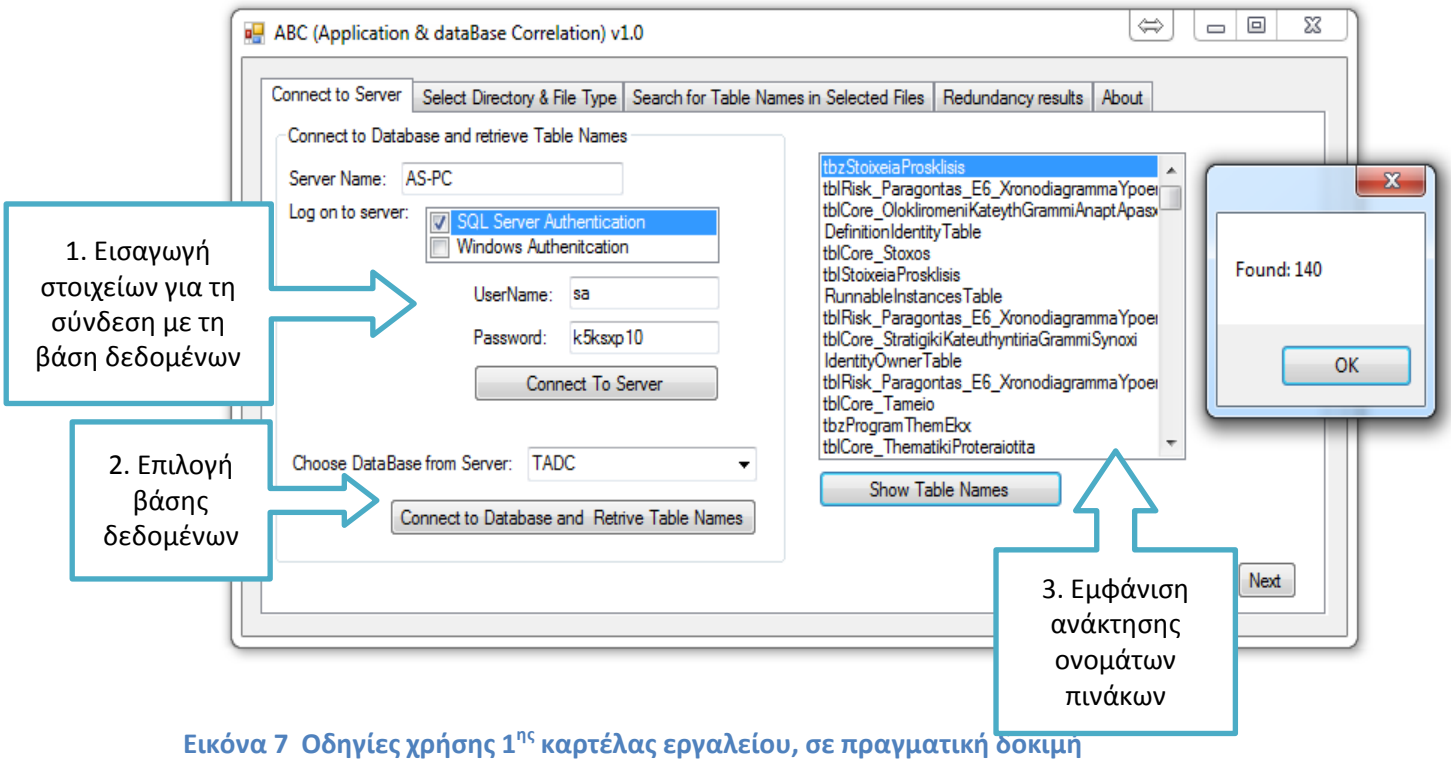
Επίσης, στον ιστότοπο ενθαρρύνονται οι χρήστες να στείλουν τη γνώμη τους (feedback), τις προτάσεις τους για βελτιώσεις και για διορθώσεις σφαλμάτων.

## 5.8.Συντήρηση

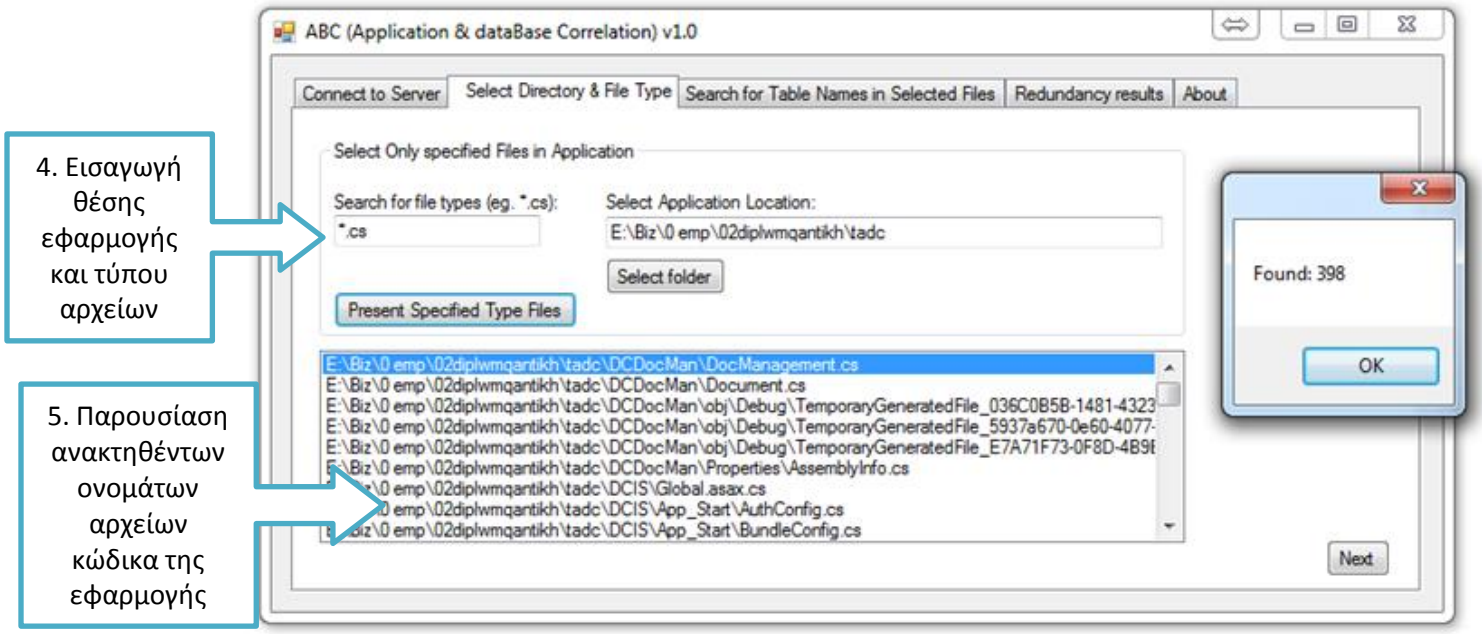
Η διαδικασία της συντήρησης και βελτίωσης του εργαλείου έχει να κάνει με την μελλοντική χρήση του. Στην ιστοσελίδα από όπου διατίθεται προς ανάκτηση το εργαλείο, προτρέπονται οι χρήστες να σχολιάσουν και να υποδείξουν βελτιώσεις και σφάλματα. Τα σχόλια των χρηστών θα λειτουργήσουν ως ανάδραση για την περαιτέρω βελτίωση και επέκταση των δυνατοτήτων του εργαλείου. Επιπλέον, και οι ίδιοι οι χρήστες θα μπορούν να εφαρμόσουν τις δικές τους τροποποιήσεις και βελτιώσεις ανάλογα με τις ανάγκες τους, γεγονός το οποίο με τη σειρά του θα οδηγήσει στον μελλοντικό εμπλουτισμό του εργαλείου.

## 5.9.Οδηγίες Χρήσης

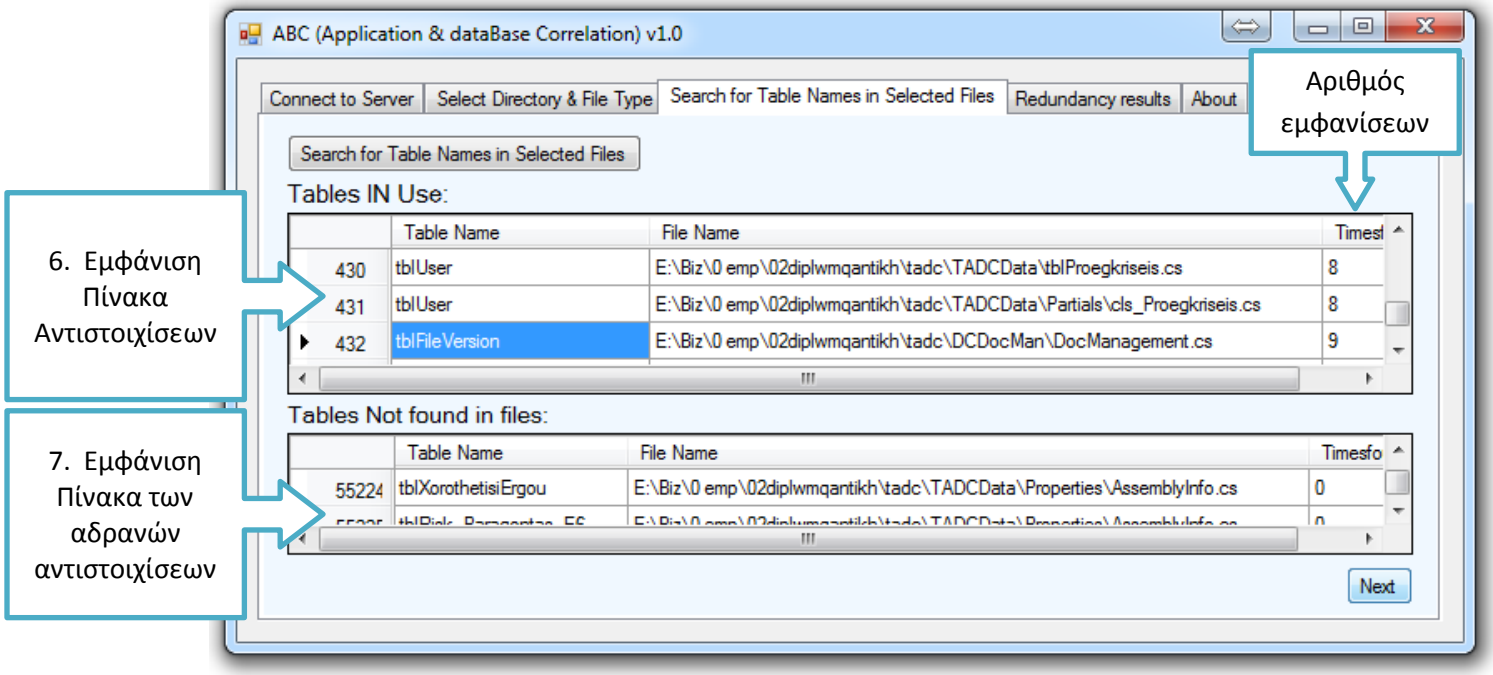
Ακολουθούν σύντομες οδηγίες χρήσης, με παράλληλη παρουσίαση της λειτουργίας του εργαλείου σε πραγματική εφαρμογή και την αντίστοιχη βάση δεδομένων.



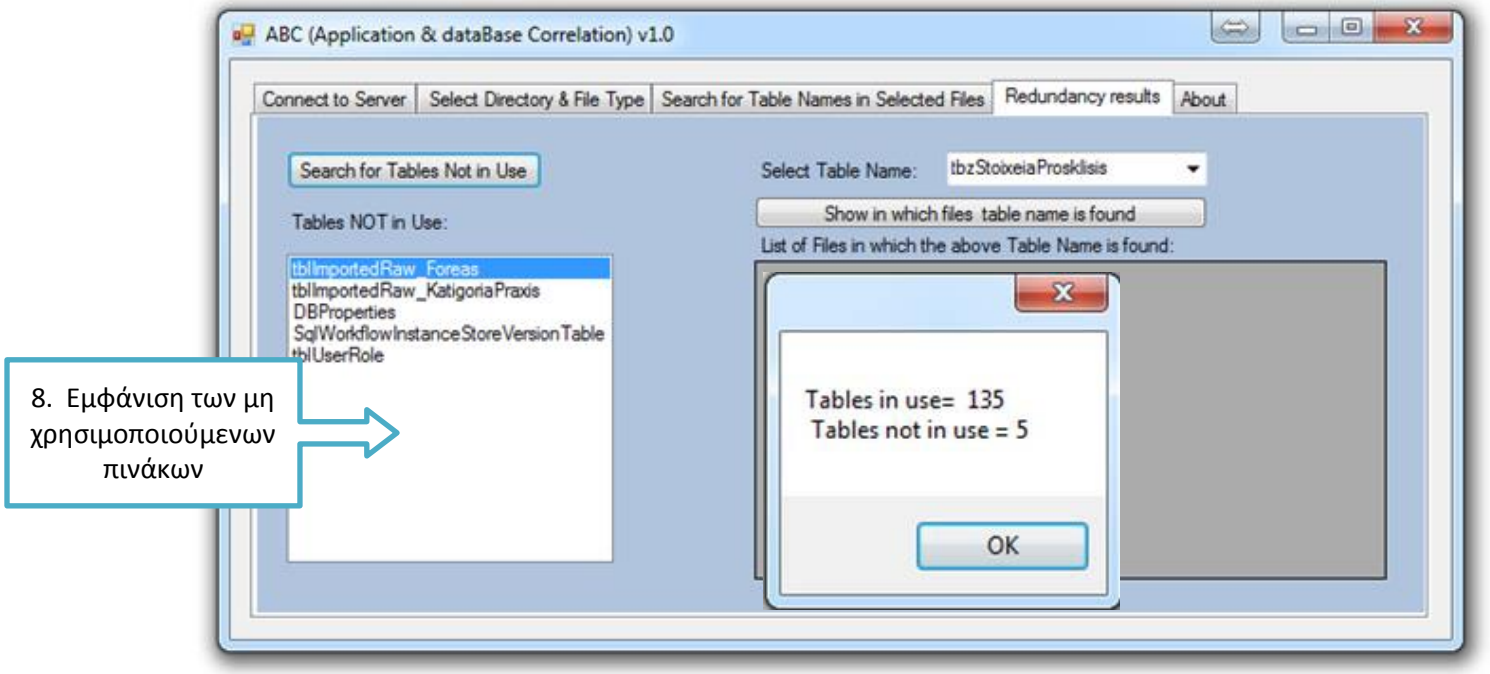

**Εικόνα 7 Οδηγίες χρήσης 1ης καρτέλας εργαλείου, σε πραγματική δοκιμή**




**Εικόνα 8 Οδηγίες χρήσης 2ης καρτέλας εργαλείου, σε πραγματική δοκιμή**




Εικόνα 9 Οδηγίες χρήσης 3ης καρτέλας εργαλείου, σε πραγματική δοκιμή




Εικόνα 10 Οδηγίες χρήσης 4ης καρτέλας εργαλείου, σε πραγματική δοκιμή

# Κεφάλαιο 6

## 6. Συμπεράσματα και Προτάσεις

### 6.1. Θεωρητικά

Καθώς όλο και περισσότεροι οργανισμοί και επιχειρήσεις επιδιώκουν να αποκτήσουν ανταγωνιστικά πλεονεκτήματα μέσω της έγκαιρης ανάπτυξης των εφαρμογών και υπηρεσιών, ιδιαίτερα όσον αφορά τις διαδικτυακές, οι προγραμματιστές βρίσκονται υπό αυξανόμενη πίεση για την ταχύτατη παραγωγή νέων και ενισχυμένων εφαρμογών. Οι ευέλικτες διαδικασίες ανάπτυξης λογισμικού αναπτύχθηκαν κυρίως για την αντιμετώπιση αυτού του προβλήματος, της ανάπτυξης λογισμικού σε εξαιρετικά γρήγορο χρόνο, εν μέσω περιβάλλοντος αλλεπάλληλων αλλαγών. Οι ευέλικτες προσεγγίσεις χρησιμοποιούν τεχνικές και διαχειριστικές διαδικασίες, οι οποίες προσαρμόζονται συνεχώς για να συγχρονιστούν με τις αλλαγές που προέρχονται από τις εμπειρίες κατά τη διάρκεια της ανάπτυξης, τις αλλαγές στις απαιτήσεις λογισμικού και τις αλλαγές στο περιβάλλον ανάπτυξης. Αυτή ακριβώς η ευελιξία τους αποτελεί και την πηγή των πλεονεκτημάτων τους.

Οι ευέλικτες διαδικασίες προορίζονται για τη στήριξη της έγκαιρης και ταχείας παραγωγής λειτουργικού κώδικα. Αυτό επιτυγχάνεται με τη διάρθρωση της αναπτυξιακής διαδικασίας σε επαναλήψεις, όπου μια επανάληψη εστιάζει στην παροχή λειτουργικού κώδικα καθώς και άλλων τεχνημάτων που προσδίδουν αξία για τον πελάτη και, δευτερευόντως, για το έργο. Τόσο οι υποστηρικτές των ευέλικτων διαδικασιών, όσο και οι επικριτές τονίζουν συχνά ότι το επίκεντρο όλων είναι ο κώδικας. (Turk et al., 2014). Το γεγονός, όμως αυτό παρά τα πλεονεκτήματα ως προς την αποτελεσματική αντιμετώπιση των όποιων αλλαγών, δημιουργεί και ορισμένους περιορισμούς στην εφαρμογή αυτών των μεθόδων.

Οι περιορισμοί αυτοί συνοψίζονται: στην έλλειψη μέριμνας για την κατασκευή επαναχρησιμοποιούμενων λογισμικών αντικείμενων, στις προκύπτουσες δυσκολίες από τη συμμετοχή πολυπληθών ομάδων (καθώς στις ομάδες αυτές, ο αριθμός των

διαύλων επικοινωνίας μπορεί να μειώσει αισθητά των αποτελεσματικότητα των ευέλικτων μεθοδολογιών), στην ακαταλληλότητα για ανάπτυξη λογισμικού κρίσιμης ασφάλειας (Κρίσιμης ασφάλειας λογισμικό είναι εκείνο στο οποίο αποτυχία μπορεί να οδηγήσει σε άμεση βλάβη στους ανθρώπους ή να προκαλέσει σοβαρή οικονομική ζημία), καθώς και στην αδυναμία για ανάπτυξη σύνθετων και πολύπλοκων λογισμικών συστημάτων.

Υπάρχει ένα πλήθος υποστηρικτικών εφαρμογών λογισμικού με στόχο την εξουδετέρωση των περιορισμών αυτών, μεταξύ των οποίων κυριότερα αποτελούν τα ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDEs) και τα συστήματα ελέγχου εκδόσεων (VCS), τα οποία διευκολύνουν ζητήματα συνεργασίας και πολυπλοκότητας κατά την ανάπτυξη των πληροφοριακών συστημάτων.

Εν κατακλείδι, σε γενικές γραμμές, ορισμένοι τομείς ενός έργου ανάπτυξης λογισμικού ενδέχεται να επωφελούνται από μια ευέλικτη προσέγγιση, ενώ άλλες από μια λιγότερο ευέλικτη ή πιο προγνωστική προσέγγιση. Από αυτή την άποψη, οι πρακτικές διαδικασίες ανάπτυξης λογισμικού μπορεί να ταξινομηθούν στην έκταση ενός φάσματος ανάλογα με το βαθμό της ευελιξίας τους. Στο ένα άκρο του φάσματος αποτυπώνονται οι καθαρά προβλεπτικές διαδικασίες, υπό την έννοια του αυστηρού καθορισμού προδιαγραφών, προγραμματισμού, σχεδιασμού και τεκμηρίωσης, δηλαδή στις οποίες τα στάδια της διαδικασίας ορίζονται λεπτομερώς στις αρχές του έργου, ενώ οι στόχοι του έργου παραμένουν σχετικά σταθεροί καθ 'όλη την εκτέλεση της διαδικασίας. Στο άλλο άκρο του φάσματος αποτυπώνονται οι καθαρά ευέλικτες διαδικασίες, στις οποίες τα στάδια της διαδικασίας και οι στόχοι του έργου είναι δυναμικοί όροι και προσδιορίζονται βάσει των αναλύσεων από εμπειρίες που αποκομίζονται αφού εκτελεστούν κάποια στάδια της διαδικασίας, είτε από παρόμοιες εμπειρίες που αποκτήθηκαν εκτός του έργου, καθώς επίσης και από μεταβολές στις απαιτήσεις και το περιβάλλον ανάπτυξης. Από αυτή την άποψη, η ευελιξία μιας διαδικασίας καθορίζεται από το βαθμό στον οποίο μία ομάδα ανάπτυξης μπορεί να προσαρμόσει δυναμικά την ακολουθούμενη διαδικασία με βάση τις αλλαγές στο περιβάλλον και τις συλλογικές εμπειρίες από τους εμπλεκομένους στο έργο, με στόχο πάντα την ολοκλήρωση του λογισμικού έργου εντός χρόνου, εντός προϋπολογισμού και στην επιθυμητή ποιότητα.

## 6.2. Σχετικά με το υποστηρικτικό εργαλείο ABC

Στο πλαίσιο της ανάλυσης του θυελλώδους σύγχρονου περιβάλλοντος της ανάπτυξης λογισμικών έργων, αναπτύχθηκε ένα πρωτότυπο υποστηρικτικό εργαλείο σε περιβάλλον Microsoft Windows, με την ονομασία ABC, από τα αρχικά των λέξεων Application & dataBase Correlation, το οποίο έρχεται να δώσει λύση σε ένα συγκεκριμένο πρακτικό ζήτημα που ανακύπτει τόσο στο στάδιο της ανάπτυξης, όσο και της συντήρησης και βελτίωσης των λογισμικών έργων.

Το λογισμικό αυτό εργαλείο στην ουσία παρέχει μια χαρτογράφηση, υπό τη μορφή πινάκων, των συσχετίσεων ενός πληροφοριακού συστήματος με τους πίνακες της βάσης δεδομένων του. Επιπλέον, εκτός από την απεικόνιση της συσχέτισης αυτής, αποκαλύπτει και τους πίνακες της βάσης δεδομένων, οι οποίοι δεν χρησιμοποιούνται από το υπό εξέταση πληροφοριακό σύστημα. Η απεικόνιση των πληροφοριών αυτών είναι ιδιαιτέρως χρήσιμη στο έργο της υλοποίησης, της συντήρησης και της βελτίωσης ενός λογισμικού συστήματος. Η αξία, μάλιστα, του εργαλείου αυτού αυξάνεται ανάλογα με την πολυπλοκότητα του πληροφοριακού συστήματος και της αντίστοιχης βάσης δεδομένων του, με το επίπεδο ευελιξίας της χρησιμοποιούμενης μεθοδολογίας προς την ανάπτυξη και διαχείριση του συστήματος, με το πολυπληθές των εμπλεκομένων ομάδων και των μεταξύ τους αλληλεπιδράσεων, καθώς και με τον ρυθμό αλλαγών τόσο των προδιαγραφών όσο και του εξωτερικού τεχνολογικού περιβάλλοντος.

Το λογισμικό ABC απευθύνεται στους εμπλεκόμενους μηχανικούς λογισμικού και στους προγραμματιστές στα στάδια της υλοποίησης, ελέγχων, συντήρησης, εκσυγχρονισμού και βελτίωσης ενός πληροφοριακού συστήματος. Η χρήση του γίνεται ακόμα πιο απαραίτητη, όταν ο ρυθμός επέμβασης και απομάκρυνσης των εμπλεκομένων στο έργο είναι γρήγορος, ώστε οι νέοι στο έργο να κατατοπίζονται εύκολα όταν πρόκειται για εργασίες σχετικά με την αλληλεπίδραση εφαρμογής και βάσης δεδομένων. Επιπλέον, έχει χρησιμότητα στις περιπτώσεις όπου οι ομάδες προγραμματισμού έχουν να αντιμετωπίσουν μια διαδικασία πολλών επαναληπτικών φάσεων, όπου η γραφή κώδικα γίνεται γρήγορα και χωρίς αρχικό σχεδιασμό, στο να εξυγιάνουν τον μεγάλο όγκο του κώδικα, δεδομένου ότι όσος περισσότερος κώδικας γράφεται τόσο πιο δύσκολο είναι να διακριθούν τα χρήσιμα τμήματα του.

Η πρώτη έκδοση του ABC υλοποιήθηκε σε ένα φιλικό περιβάλλον προς τον χρήστη, με στόχο να είναι άμεσα λειτουργικό, χωρίς την ανάγκη πολλών οδηγιών. Ο πηγαίος κώδικας είναι ελεύθερος προς τροποποίηση και βελτίωση ανάλογα με τις εκάστοτε ανάγκες των χρηστών, και παράλληλα είναι κατάλληλα δομημένος και με επαρκή σχολιασμό για τη διευκόλυνση της κατανόησης, της αλλαγής και συμπλήρωσής του. Προς το παρόν, έχει τη δυνατότητα τοπικής μόνο σύνδεσης αποκλειστικά με συστήματα διαχείρισης βάσης δεδομένων SQL server, αλλά πλεονεκτεί στο ότι μπορεί να ελέγχει εφαρμογές ανεξαρτήτως από την γλώσσα προγραμματισμού αυτών.

### 6.2.1. Μελλοντικές Επεκτάσεις

Από την διαδικασία ανάπτυξης του ABC και την καλύτερη εμπέδωση των εννοιών της ανάπτυξης και συντήρησης πληροφοριακών συστημάτων, έχουν ήδη αναπτυχθεί ιδέες σχετικά με την μελλοντική προσθήκη συμπληρωματικών λειτουργιών, πέρα από τις βελτιώσεις της λειτουργικότητας. Για παράδειγμα προσθήκη νέων δυνατοτήτων: απομακρυσμένης σύνδεσης και με άλλα συστήματα διαχείρισης βάσεων δεδομένων (πχ. MySQL, PostgreSQL, και Oracle), παρουσίασης των αποτελεσμάτων σε δενδροειδή μορφή, εκτύπωσης των αποτελεσμάτων και εξαγωγής τους σε τύπους αρχείων “*.csv” και “*.xls” κατάλληλους για προγράμματα επεξεργασίας λογιστικών φύλλων, κλπ.

Με κατάλληλη υποστήριξη και βελτίωση, θα μπορούσαν να ενσωματωθούν και άλλες κρίσιμες λειτουργίες όπως η χαρτογράφηση αλληλεπιδρώντων αντικειμένων κώδικα και κατ΄ επέκταση ο εντοπισμός αδρανών, με στόχο μια συνολική αναφορά για όλα τα αντικείμενα κώδικα, πλέον της βάσης δεδομένων. Επίσης, μία ιδιαίτερα χρήσιμη πρόταση αφορά τη μετάβαση από την παρουσίαση με μορφή πινάκων σε μία γραφική παρουσίαση δικτυακού διαγράμματος αναπαράστασης των συσχετίσεων. Η εξέλιξη του εργαλείου βασίζεται και στην ανάδραση με τους χρήστες και τη διατύπωση των αναγκών τους, αλλά μια επιπλέον πρόταση θα ήταν η αναβάθμιση του σε διαδικτυακή εφαρμογή, ώστε να είναι απ΄ευθείας διαθέσιμη από οποιαδήποτε συσκευή ανεξαρτήτως της επεξεργαστικής της ισχύος.

Τελικός στόχος, του ABC είναι να διευκολύνει το έργο των εμπλεκομένων, αυξάνοντας την παραγωγικότητά τους, στην αποσαφήνιση και την απλοποίηση υπάρχοντος κώδικα, με σκοπό τη βελτίωση της ποιότητας του λογισμικού συστήματος.

# Βιβλιογραφία


2013 IT Project Success Rates Survey Results [WWW Document], n.d. URL http://www.ambysoft.com/surveys/success2013.html#Results (accessed 2.27.15).

Abrahamsson, P., Salo, O., Ronkainen, J., Valtion teknillinen tutkimuskeskus, 2002. Agile software development methods: review and analysis. VTT, Espoo [Finland].

Agile Alliance :: Home [WWW Document], n.d. URL http://www.agilealliance.org/ (accessed 3.14.15).

agilevswater.jpg (JPEG Image, 600 × 292 pixels) [WWW Document], n.d. URL http://www.codeproject.com/KB/architecture/604417/agilevswater.jpg (accessed 3.16.15).

BABOK Guide Online - IIBA | International Institute of Business Analysis [WWW Document], n.d. URL http://www.iiba.org/babok-guide/babok-guide-online.aspx (accessed 2.23.15).

Beck, K., 2005. Extreme programming explained: embrace change, 2nd ed. ed. Addison-Wesley, Boston, MA.

Bhandari, P., Sehgal, R., 2014. AN EVALUATION OF METHODS TO PRIORITIZE REQUIREMENTS.

Booch, G., 2005. The unified modeling language user guide, 2nd ed. ed. Addison-Wesley, Upper Saddle River, NJ.

Bourque, P., Fairley, R.E., IEEE Computer Society, 2014. Swebok: guide to the software engineering body of knowledge. IEEE Computer Society, [Los Alamitos, CA].

Buede, D.M., 2009. The engineering design of systems: models and methods, 2nd ed. ed, Wiley series in systems engineering and management. John Wiley & Sons, Hoboken, N.J.

Chiang, T., 2010. The lifecycle of software objects. Subterranean Press, Burton, MI.

Choosing An Appropriate System Development Methodology - SelectingDevelopmentApproach.pdf, n.d.

Cockburn, A., 2007. Agile software development: the cooperative game, 2nd ed. ed, The Agile software development series. Addison-Wesley, Upper Saddle River, NJ.

Computer Software Definition [WWW Document], n.d. URL http://www.openprojects.org/software-definition.htm (accessed 2.2.15).

Cowan, A., 2012. Starting a tech business: a practical guide for anyone creating or designing applications or software. Wiley, Hoboken, New Jersey.

DataGrid Class (System.Windows.Forms) [WWW Document], n.d. URL https://msdn.microsoft.com/en-us/library/system.windows.forms.datagrid(v=vs.110).aspx (accessed 3.12.15).

dataists » Ranking the popularity of programming languages [WWW Document], n.d. URL http://www.dataists.com/2010/12/ranking-the-popularity-of-programming-langauges/ (accessed 2.15.15).

Delivering large-scale IT projects on time, on budget, and on value | McKinsey & Company [WWW Document], n.d. URL http://www.mckinsey.com/insights/business_technology/delivering_large-scale_it_projects_on_time_on_budget_and_on_value (accessed 2.27.15).

Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N.B., 2012. A decade of agile methodologies: Towards explaining agile software development. J. Syst. Softw. 85, 1213–1221. doi:10.1016/j.jss.2012.02.033

Directory.GetFiles Method (System.IO) [WWW Document], n.d. URL https://msdn.microsoft.com/en-us/library/system.io.directory.getfiles(v=vs.110).aspx (accessed 3.12.15).

Elliott, G., 2004. Global business information technology: an integrated systems approach. Pearson Addison Wesley, Harlow, England ; New York.
Elmasri, R., Navathe, S.B., Chatzopoulos, M., 2007. Themeliōdeis arches systēmatōn vaseōn dedomenōn. Ekdoseis Diaulos, Athēna.
Finkelstein, A., International Conference on Software Engineering (Eds.), 2000. The future of software engineering 2000: 22nd International Conference on Software Engineering. Association for Computing Machinery, New York.
Gamma, E. (Ed.), 1995. Design patterns: elements of reusable object-oriented software, Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass.
Greenfield, J., 2004. Software factories: assembling applications with patterns, models, frameworks, and tools. Wiley Pub, Indianapolis, IN.
http://www.icsd.aegean.gr/kkot/softTech06Week1L2.ppt [WWW Document], n.d. URL http://www.icsd.aegean.gr/kotis/softTech06/myPresentation/Week1L2.ppt (accessed 3.6.15).
Ideal programming language learning sequence? - Programmers Stack Exchange [WWW Document], n.d. URL http://programmers.stackexchange.com/questions/10675/ideal-programming-language-learning-sequence (accessed 3.18.15).
IEEE Standard For Developing Software Life Cycle Processes - IEEE Std 1074-1997 - IEEE1074.pdf, n.d.
IEEE Xplore Abstract - Common Trends in Software Fault and Failure Data [WWW Document], n.d. URL http://ieeexplore.ieee.org/Xplore/defdeny.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fstamp%2Fstamp.jsp%3Ftp%3D%26arnumber%3D4760152%26userType%3Dinst&denyReason=-134&arnumber=4760152&productsMatched=null&userType=inst (accessed 3.8.15).
Images For > History Of Programming Languages [WWW Document], n.d. URL http://imgkid.com/history-of-programming-languages.shtml (accessed 2.15.15).
Jessup, L.M., 2008. Information systems today: managing in the digital world, 3rd ed. ed. Pearson Prentice Hall, Upper Saddle River, N.J.
Klimczak, E., 2013. Design for software: a playbook for developers. Wiley ; John Wiley [distributor], Hoboken, N.J. : Chichester.
Kline, R.B., Seffah, A., 2005. Evaluation of integrated software development environments: Challenges and results from three empirical studies. Int. J. Hum.-Comput. Stud. 63, 607–627. doi:10.1016/j.ijhcs.2005.05.002
Languages | Updates from a continuous developer [WWW Document], n.d. URL https://continuousdevelopment.wordpress.com/category/languages/ (accessed 2.15.15).
Larman, C., Basili, V.R., 2003. Iterative and incremental developments. a brief history. Computer 36, 47–56. doi:10.1109/MC.2003.1204375
McConnell, S., 2004. Code complete, 2nd ed. ed. Microsoft Press, Redmond, Wash.
Microsoft Word - PhD_All Chapters_final_2sided print ready - Monochristou_Phd2011.pdf, n.d.
Most Popular Coding Languages of 2015 — CodeEval [WWW Document], n.d. URL http://blog.codeeval.com/codeevalblog/2015#.VNsEZy4mmFU= (accessed 2.15.15).
MSWE 609 Syllabus [WWW Document], n.d. URL http://www.cs.umd.edu/~mvz/mswe609/ (accessed 3.12.15).
Myatt, A., 2007. Pro NetBeans IDE 5.5 enterprise edition, Expert's voice in Java technology. Apress ; Distributed to the book trade by Springer-Verlag, Berkeley, CA : New York.

NEW PRODUCT DEVELOPMENT GLOSSARY [WWW Document], n.d. URL http://www.npd-solutions.com/glossary.html (accessed 2.21.15).
Palmer, S.R., 2002. A practical guide to feature-driven development, The Coad series. Prentice Hall PTR, Upper Saddle River, NJ.
Paper Title (use style: paper title) - V3I3-0198.pdf, n.d.
Paulk, M.C. (Ed.), 1995. The capability maturity model: guidelines for improving the software process, The SEI series in software engineering. Addison-Wesley Pub. Co, Reading, Mass.
Pichler, R., 2010. Agile product management with Scrum: creating products that customers love, The Addison-Wesley signature series. Addison-Wesley, Upper Saddle River, NJ.
Programmer, Developer, Engineer: What's in a name? [WWW Document], n.d. URL http://chrislema.com/programmer-developer-engineer/ (accessed 2.1.15).
Rapid application development : Software methodologies | Rivulets Technologies [WWW Document], n.d. URL http://www.rivulets.in/blog/project-management/rapid-application-development-software-methodologies/ (accessed 3.19.15).
Regex Class (System.Text.RegularExpressions) [WWW Document], n.d. URL https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex(v=vs.110).aspx (accessed 3.12.15).
Rico, D.F., 2010. Short History of Software Methods. Downloaded Web August.
Riehle, D., 2000. Framework design. Diss. Technische Wissenschaften ETH Zürich, Nr. 13509, 2000.
Schwaber, K., 2004. Agile project management with Scrum. Microsoft Press, Redmond, Wash.
Schwaber, K., 2002. Agile software development with Scrum, Series in agile software development. Prentice Hall, Upper Saddle River, NJ.
Scott, M.L., 2009. Programming language pragmatics, 3rd ed. ed. Elsevier/Morgan Kaufmann Pub, Amsterdam ; Boston.
Scrum [WWW Document], n.d. URL http://aetos.it.teithe.gr/~sfetsos/Scrum.html (accessed 2.25.15).
SDLC Quick Guide [WWW Document], n.d. URL http://www.tutorialspoint.com/sdlc/sdlc_quick_guide.htm (accessed 2.22.15).
Shore, J., 2008. The art of agile development, Theory in practice. O'Reilly Media, Inc, Beijing : Sebastopol, CA.
Software Design Methodology [WWW Document], n.d. URL http://userpages.umbc.edu/~khoo/survey1.html (accessed 2.25.15).
Software Development Versus Software Engineering [WWW Document], n.d. URL http://www.softwareengineerinsider.com/articles/software-development-software-engineering.html#.VM56nS5LUrM (accessed 2.1.15).
Software development - Wikipedia, the free encyclopedia [WWW Document], n.d. URL http://en.wikipedia.org/wiki/Software_development (accessed 2.21.15).
Software Process Model — Prototyping Process Model [WWW Document], n.d. URL http://lauyin.com/NPD/model/prototype.html (accessed 3.6.15).
software - WordReference.com Dictionary of English [WWW Document], n.d. URL http://www.wordreference.com/definition/software (accessed 3.13.15).
Sommerville, I., 2007. Software engineering, 8th ed. ed, International computer science series. Addison-Wesley, Harlow, England ; New York.
SqlCommand Class (System.Data.SqlClient) [WWW Document], n.d. URL https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand(v=vs.110).aspx (accessed 3.12.15).

SqlConnection Constructor (System.Data.SqlClient) [WWW Document], n.d. URL https://msdn.microsoft.com/en-us/library/9197xfyw(v=vs.110).aspx (accessed 3.12.15).
SqlDataReader Class (System.Data.SqlClient) [WWW Document], n.d. URL https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader(v=vs.110).aspx (accessed 3.12.15).
Stamelos, I.G., Sfetsos, P. (Eds.), 2007. Agile software development quality assurance. Information Science Reference, Hershey, PA.
Stapleton, J., 1997. DSDM, dynamic systems development method: the method in practice. Addison-Wesley, Harlow, England; Reading, Mass.
terminology - What's the difference between programmer and software engineer? - Stack Overflow [WWW Document], n.d. URL http://stackoverflow.com/questions/27516/whats-the-difference-between-programmer-and-software-engineer (accessed 2.1.15).
The Complete List of Software Development Frameworks, Process's, Methods, or Philosophies | Software Development in the Real World [WWW Document], n.d. URL http://www.realsoftwaredevelopment.com/the-complete-list-of-software-development-frameworks-processs-methods-or-philosophies/ (accessed 3.13.15).
The Philosophy of Computer Science (Stanford Encyclopedia of Philosophy) [WWW Document], n.d. URL http://plato.stanford.edu/entries/computer-science/ (accessed 3.11.15).
try-catch (C# Reference) [WWW Document], n.d. URL https://msdn.microsoft.com/en-us/library/0yd65esw.aspx (accessed 3.12.15).
Turk, D., France, R., Rumpe, B., 2014. Limitations of agile software processes. ArXiv Prepr. ArXiv14096600.
Valacich, J.S., 2014. Information systems today: managing in the digital world, Sixth edition. ed. Pearson, Boston.
Waterfall Model | Definition and Concept | IT & Systems | MBA Skool-Study.Learn.Share. [WWW Document], n.d. URL http://www.mbaskool.com/business-concepts/it-and-systems/8658-waterfall-model.html (accessed 3.6.15).
Wells, A.J., 2008. Grid application systems design. Auerbach Publications, Boca Raton, FL.
Whitten, J.L., 1998. Systems analysis and design methods, 4th ed. ed. Irwin/McGraw-Hill, Boston, Mass.
Wiegers, K.E., 2013. Software requirements, Third edition. ed. Microsoft Press, s division of Microsoft Corporation, Redmond, Washington.
XP 2008, 2008. Agile processes in software engineering and extreme programming: 9th international conference, XP 2008, Limerick, Ireland, June 10-14, 2008: proceedings, Lecture notes in business information processing. Springer, Berlin ; New York.
Άδειες Διάθεσης Περιεχομένου - Creative Commons 3.0 | MY-AOC [WWW Document], n.d. URL http://www.aoc.ntua.gr/my/el/article/creative-commons (accessed 3.12.15).
Γλώσσες Προγραμματισμού - Κοινότητα Ελεύθερου Λογισμικού ΕΜΠ [WWW Document], n.d. URL https://foss.ntua.gr/wiki/index.php/ (accessed 2.15.15).
ΔΟΜΗΜΕΝΟΣ - DP Pangalos.pdf, n.d.
Εμμανουήλ Σκορδαλάκης, 2007. Λογισμική Μηχανική, 4ης έκδοση-ηλεκτρονική. ed. Εκδόσεις Ε.Μ.Π., Αθήνα.
Ευέλικτη Ανάπτυξη Λογισμικού | Sprint SMEs [WWW Document], n.d. URL http://sprint.teilar.gr/educational/ (accessed 3.7.15).
Μανιφέστο για την ευέλικτη ανάπτυξη λογισμικού [WWW Document], n.d. URL http://agilemanifesto.org/iso/el/ (accessed 3.8.15).