ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ

**ΤΙΤΛΟΣ**

# ΛΟΓΙΣΜΙΚΟ ΕΛΕΓΧΟΥ ΑΥΤΟΚΙΝΟΥΜΕΝΟΥ ΟΧΗΜΑΤΟΣ

# SOFTWARE OF CONTROL AUTONOMOUS VEHICLE

**ΠΛΑΚΑΣ ΗΛΙΑΣ**

**ΑΜ 45001**

**Αιγάλεω, Ιούλιος 2021**

## ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΡΟΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/η κάτωθι υπογεγραμμένος Πλάκας Ηλίας με αριθμό μητρώου 45001 φοιτητής/τρια του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Βιομηχανικής Σχεδίασης και Παραγωγής, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της προπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Πλάκας Ηλίας

Ο/Η Δηλών/ούσα

**ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ**

1. ΠΑΠΑΠΟΥΤΣΙΔΑΚΗΣ ΜΙΧΑΛΗΣ

MICHAIL PAPOUTSIDAKIS
Digitally signed by MICHAIL PAPOUTSIDAKIS
Date: 2021.07.23 16:25:35 +03'00'

2. ΧΑΤΖΟΠΟΥΛΟΣ ΑΒΡΑΑΜ

Avraam Chatzopoulos
Digitally signed by Avraam Chatzopoulos
Date: 2021.07.22 13:31:49 +03'00'

3. ΔΡΟΣΟΣ ΧΡΗΣΤΟΣ

Christos Drosos
Digitally signed by Christos Drosos
Date: 2021.07.22 09:34:31 +03'00'

**ΠΕΡΙΕΧΟΜΕΝΑ**

**Περίληψη**

Τα *αυτόνομα οχήματα (autonomous vehicle*) που είναι ικανά να κινούνται και να δρουν στο χώρο, χωρίς να απαιτείται καθοδήγηση ή τηλεχειρισμός - παρουσιάζουν σημαντικό ενδιαφέρον. Σκοπός της παρούσας εργασίας είναι ο σχεδιασμός και η υλοποίηση λογισμικού ελέγχου (software) ενός αυτοκινούμενου οχήματος, χειριζόμενου μέσω ασύρματης επικοινωνίας, με δυνατότητα ιχνηλάτησης διαδρομής, αξιοποιώντας τις δυνατότητες ενός μικροελεγκτή. Η σχεδίαση του λογισμικό για τον έλεγχο του οχήματος, καθώς και η υλοποίηση του είναι εύχρηστη όπως και ο πειραματισμός ταχύς. Όσο αφορά τους αλγόριθμους linetracking, ο PID έλεγχος φάνηκε να είναι αξιόπιστος και πιο γρήγορος. Κύριος κανόνας σε αυτούς τους αλγορίθμους είναι το όχημα να μην βγει εκτός διαδρομής. Για κάθε όχημα και κάθε διαδρομή, υπάρχει ένα κατώφλι ταχύτητας, στην οποία το ρομπότ μπορεί να ολοκληρώσει επιτυχώς την διαδρομή. Επομένως, οι απλοί αλγόριθμοι .μπορούν να λειτουργήσουν ικανοποιητικά σε οχήματα μικρής ταχύτητας ή σε εύκολες διαδρομές, χωρίς την απαίτηση  πιο σύνθετου αλγόριθμου.

**Λέξεις-κλειδιά** - αυτόνομα οχήματα, σχεδιασμός λογισμικού, τεχνολογία λογισμικού, bluetooth, python

**Abstract**

Autonomous vehicles that are able to move and operate in space, without the need for guidance or remote control - are of great interest. The purpose of this work is the design and implementation of control software (software) of a motor vehicle, operated via wireless communication, with the possibility of tracking, utilizing the capabilities of a microcontroller. The design of the software for the control of the vehicle, as well as its implementation is easy to use as well as the fast experimentation. In terms of linetracking algorithms, PID control seemed to be more reliable and faster. The main rule in these algorithms is that the vehicle does not go off-road. For each vehicle and each route, there is a speed threshold at which the robot can successfully complete the route. Therefore, simple algorithms can work well on low speed vehicles or easy routes without requiring a more sophisticated algorithm.

**Keywords** - Autonomous vehicles, python, software design, bluetooth, software engineering

# ΕΙΣΑΓΩΓΗ

Τα *αυτόνομα οχήματα (autonomous vehicle*) που είναι ικανά να κινούνται και να δρουν στο χώρο, χωρίς να απαιτείται καθοδήγηση ή τηλεχειρισμός - παρουσιάζουν σημαντικό ενδιαφέρον. Οι πιθανές εφαρμογές τους είναι ποικίλες, όπως χειρισμός εργαλείων/υλικών, απομακρυσμένες επισκευές και έργα συντήρησης, υπηρεσίες-διευκολύνσεις σε καθημερινές ανάγκες, αναγνώριση και διερεύνηση σε δυσπρόσιτα ή επικίνδυνα περιβάλλοντα (θάλασσα, αέρας, διάστημα κ.α.).

Σκοπός της παρούσας εργασίας είναι ο σχεδιασμός και η υλοποίηση λογισμικού ελέγχου (software) ενός αυτοκινούμενου οχήματος, χειριζόμενου μέσω ασύρματης επικοινωνίας, με δυνατότητα ιχνηλάτησης διαδρομής, αξιοποιώντας τις δυνατότητες ενός μικροελεγκτή.

Η διάθρωση της εργασίας αφορά τα εξής κεφάλαια:

- *Κεφάλαιο 1*. Παρουσιάζεται η θεωρία σχετικά με την τεχνολογία λογισμικού.
- *Κεφάλαιο 2*. Παρουσιάζεται η γλώσσα προγραμματισμού PYTHON.
- *Κεφάλαιο 3*. Σχεδίαση και ανάπτυξη λογισμικού.

Η εργασία βασίστηκε στην πτυχιακή εργασία του Βαφειάδη, Ι. στο Πανεπιστήμιο Πατρών (2017), που ανέπτυξε ένα σχετικό λογισμικό.

# ΚΕΦΑΛΑΙΟ ΠΡΩΤΟ

# ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

## 1.1 Γενικά

Ένα από τα σπουδαιότερα γεγονότα που καθόρισαν τον 20ο αιώνα, ήταν η εφεύρεση του *ηλεκτρονικού υπολογιστή (Η/Υ, computer*). Με τη ανακάλυψη του έγινε δυνατή η αυτοματοποίηση της εκτέλεσης πολλών κουραστικών, ανιαρών και επιρρεπών σε λάθη εργασιών, καθώς και η εκτέλεση εργασιών που μέχρι τότε ήταν πρακτικά αδύνατη. Ακολούθησε, στη δεκαετία του '50, η εμφάνιση της έννοιας του *λογισμικού (software),* όπου μέχρι τότε, κανείς δεν προέβλεψε τη σημασία που θα έχει στο μέλλον αυτή ανακάλυψη. Το λογισμικό έδωσε μεγάλες δυνατότητες σε υπάρχουσες τεχνολογίες (π.χ. ρομποτική, τηλεπικοινωνίες), δημιούργησε νέες επιστήμες (π.χ. γενετική βιολογία), έφερε πιο κοντά την τεχνολογία στο χρήστη (π.χ. επεξεργαστές κειμένου), βελτίωσε την ποιότητα ζωής των πολιτών (π.χ. τηλεϋπηρεσίες εκπαίδευσης, υγείας, πολιτισμού, ενημέρωσης) και βοήθησε στη μεγιστοποίηση της κοινωνικής ευημερίας με την αποδοτικότερη αξιοποίηση των οικονομικών πόρων (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015; Fox and Patterson, 2017).

Το λογισμικό αναπτύσσεται αλλά και αξιοποιείται από τους *μηχανικούς λογισμικού (software engineers*). 'Ομως τι είναι λογισμικό (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015; Fox and Patterson, 2017):

- Προγράμματα ηλεκτρονικού υπολογιστή τα οποία όταν εκτελούνται επιτυγχάνουν επιθυμητά αποτελέσματα και επιδόσεις.
- Δομές δεδομένων που επιτρέπουν σε προγράμματα να διαχειριστούν με επάρκεια κάθε λογής δεδομένα.
- Κείμενα, διαγράμματα, μορφές παράστασης κ.ά., που περιγράφουν τη δομή, λειτουργία και χρήση των προγραμμάτων.

Από την άλλη, *τεχνολογία λογισμικού (software engineering*) είναι ο κλάδος της πληροφορικής που ασχολείται με τη μελέτη και την εφαρμογή συστηματικών, μεθοδικών και ποσοτικοποιημένων προσεγγίσεων για την ανάπτυξη, λειτουργία και συντήρηση του λογισμικού. Η τεχνολογία λογισμικού δεν έχει ακόμα όλα εκείνα τα χαρακτηριστικά που θα την καθιστούσαν επιστήμη και θα της επέτρεπαν να αποδεικνύει την ύπαρξη βέλτιστης, ορθής, πλήρους και αξιόπιστης λύσης. Παρ' όλα αυτά, είναι ένας σημαντικός τομέας της επιστήμης των υπολογιστών και αποτελεί

βασικό αντικείμενο σπουδών για τα πανεπιστημιακά τμήματα πληροφορικής σε όλο τον κόσμο (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015; Fox and Patterson, 2017).

## 1.2 Ο Ρόλος και η Σημασία του Λογισμικού

Το λογισμικό κυρίως εκτελεί δυο κατηγορίες λειτουργιών. Είτε λειτουργεί ως "*μετασχηματιστής πληροφορίας*" είτε ως "*ελεγκτής-συντονιστής*" της μηχανής. Η πρώτη κατηγορία συλλέγει, επεξεργάζεται, διαχειρίζεται, μεταδίδει, παρουσιάζει, μορφοποιεί και τροποποιεί πληροφορία. Η δεύτερη κατηγορία διαχειρίζεται τους πόρους της μηχανής για να εξυπηρετήσει τα αιτήματα των χρηστών, μεριμνά για την επικοινωνία δυο μηχανών (Η/Υ) και συντονίζει τις λειτουργίες δύο μηχανών ή δύο λογισμικών με σκοπό τη ολοκλήρωση μίας εργασίας (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015; Fox and Patterson, 2017).

Στην ορολογία της επιστήμης των υπολογιστών θα διακρίνουμε τις παρακάτω κατηγορίες λογισμικού (όχι πάντα διακριτές μεταξύ τους)( Γιακουμάκης και Διαμαντίδης, 2009; Fox and Patterson, 2017):

- *Λογισμικό Συστήματος (system software):* λειτουργικά συστήματα, μεταγλωττιστές (μεταφραστές), υπηρεσίες διαχείρισης αρχείων, οδηγοί περιφερειακών συσκευών, λογισμικό δικτύου και κάθε λογισμικό που σκοπό έχει να παρέχει υπηρεσίες σε άλλο λογισμικό αξιοποιώντας τους πόρους του υλικού του υπολογιστή.
- *Λογισμικό Εφαρμογής (application software*): κάθε λογισμικό που σκοπό έχει την επίλυση ειδικών προβλημάτων του χρήστη στα πλαίσια κάποιων επιχειρησιακών λειτουργιών.
- *Επιστημονικό Λογισμικό / Λογισμικό για μηχανικούς (engineering / scientific software*): λογισμικό που χρησιμοποιείται από διάφορους επιστημονικούς κλάδους (αστρονομία, αεροδιαστημική, αρχιτέκτονες, βιολόγους, μηχανικούς λογισμικού κ.α.) για τη διαχείριση προβλημάτων τους και που κατά κανόνα χαρακτηρίζεται από ειδικές απαιτήσεις σε αλγορίθμους και διαχείριση πολυπλοκότητας.
- *Ενσωματωμένο Λογισμικό (embedded software*): λογισμικό που έχει ενσωματωθεί σε μηχανές ειδικού σκοπού (π.χ. φούρνος μικροκυμάτων,

σύστημα φρένων αυτοκίνητου κ.τ.λ.) με σκοπό την εκτέλεση εσωτερικών λειτουργιών.

- *Λογισμικό γραμμής παραγωγής (product line software*): λογισμικό που χρησιμοποιείται από ευρύ φάσμα χρηστών για την εκτέλεση κάποιων λειτουργιών ευρείας χρήσης (π.χ. επεξεργαστές κειμένου, εφαρμογές διαχείρισης οικονομικών στοιχείων, προγράμματα γραφικών κ.τ.λ.).
- *Λογισμικό Διαδικτυακών Εφαρμογών (web applications):* λογισμικό που αξιοποιεί τις τεχνολογίες του διαδικτύου. Παραδείγματα διαδικτυακού λογισμικού είναι οι εφαρμογές ηλεκτρονικού εμπορίου.
- *Λογισμικό Τεχνητής Νοημοσύνης (artificial intelligence software*): λογισμικό για ρομπότ, νευρωνικά δίκτυα, συστήματα συμπερασμού και γενικά λογισμικό εφαρμογών Τεχνητής Νοημοσύνης.
- *Λογισμικό για διάχυση πληροφορίας και υπολογισμού (ubiquitous computing*): λογισμικό που αξιοποιεί τα ασύρματα δίκτυα και την κατανεμημένη σε διάφορες μηχανές υπολογιστική ισχύ και παρέχει υπηρεσίες στο χρήστη μέσω κινητών συσκευών διεπαφής.
- *Λογισμικό δικτυακό (net sourcing*): λογισμικό εφαρμογών που αξιοποιεί τις δυνατότητες του διαδικτύου και παρέχει υπηρεσίες στον τελικό χρήστη του διαδικτύου.
- *Λογισμικό Ανοικτού κώδικα (Open source*): λογισμικό που επιτρέπει τόσο στο χρήστη του όσο και στην ομάδα που το ανέπτυξε να γνωρίζει τις αλλαγές που έχουν γίνει και τα αποτελέσματα που αυτές δημιουργούν.

Ανεξάρτητα από ρόλο, το λογισμικό είναι απαραίτητο μέρος κάθε μηχανής (Η/Υ). Επίσης, οι λειτουργικές δυνατότητές του και οι επιδόσεις του είναι συνάρτηση των χαρακτηριστικών της μηχανής. Όπως με την πάροδο του χρόνου οι δυνατότητες της μηχανής (Η/Υ) πολλαπλασιάζονται, παράλληλα διευρύνονται και οι δυνατότητες του λογισμικού. Επιπλεόν, εκτός από τα τεχνολογικά χαρακτηριστικά, σημαντική είναι η διείσδυση του λογισμικού στην οργάνωση και λειτουργία της οικονομίας, του κράτους και της κοινωνίας σε διεθνή κλίμακα. Όλα αυτά δείχνουν μια μεγάλη διεθνή βιομηχανία, τη βιομηχανία του λογισμικού. Σε όλες τις ανεπτυγμένες χώρες η βιομηχανία λογισμικού κατέχει σημαντική θέση στην οικονομία τους. Στη βιομηχανία Λογισμικού περιλαμβάνονται τόσο η ανάπτυξη λογισμικού όσο και οι υπηρεσίες αξιοποίησης και βελτίωσης του υπάρχοντος λογισμικού. Η προσπάθεια για

την αναζήτηση μεθόδων και τεχνικών για την παραγωγή αξιόπιστου & ποιοτικού λογισμικού συνεχίζεται, με στόχο την παράδοση του στην ώρα του και εντός των πλαισίων του προϋπολογισμού του έργου ανάπτυξης. Επίσης, η προσπάθεια αυτή αποκτά μεγαλύτερες διαστάσεις λόγω των εξελίξεων που αναδεικνύουν το σπουδαίο ρόλο και σημασία του λογισμικού. Παράγοντες που ενισχύουν τη αξία του λογισμικού χρόνο είναι οι εξής (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015):

- Οι αλλαγές στο λόγο κόστους υλικού προς κόστος λογισμικού στα εγκατεστημένα υπολογιστικά συστήματα. Η τιμή του παραπάνω λόγου συνεχώς μειώνεται, τόσο διότι το κόστος του υλικού μειώνεται, όσο και γιατί το κόστος του λογισμικού αυξάνεται.
- Η αυξανόμενη σημασία της συντήρησης του λογισμικού. Στην προσπάθειά μας να παρατείνουμε το χρόνο ζωής ενός λογισμικού, επενδύουμε στη συντήρησή του.
- Οι εξελίξεις στο υλικό. Οι χωρητικότητες και οι ταχύτητες του υλικού αυξάνονται εκθετικά, όταν την ίδια στιγμή τα κόστη του υλικού μειώνονται σταθερά. Το υλικό απαιτεί απαιτητικότερο σε υπολογιστική ισχύ λογισμικό για να αξιοποιηθεί.
- Οι εξελίξεις στις τεχνικές λογισμικού. Θέλουμε πολυμεσικό, δικτυακό, αλληλοδραστικό, πολλών χρηστών, on-line λογισμικό.
- Οι αυξανόμενες απαιτήσεις για λογισμικό. Οι υπολογιστές και το λογισμικό έχουν θέση σε όλες τις δραστηριότητες της δημόσιας ζωής, στη διοίκηση, στην εκπαίδευση, στην οικονομία, στην υγεία, στην ενημέρωση, στον πολιτισμό.
- Η απαίτηση για μεγαλύτερα και πολυπλοκότερα συστήματα λογισμικού. Θέλουμε λειτουργικότερα, ταχύτερα, διαδικτυωμένα, με υψηλό βαθμό ολοκλήρωσης λογισμικά συστήματα.

## 1.3 Η Κρίση του Λογισμικού

Το σύνολο αυτών των χρόνιων προβλημάτων ονομάζονται "*κρίση λογισμικού*". Ενδεχομένως, η χρήση όρων όπως "*κρίση*" ή "*χρόνια προβλήματα*" μπορεί να χαρακτηριστεί υπερβολική, αυτό όμως δεν αναιρεί ούτε τη σοβαρότητα ούτε την παρατεταμένη διάρκεια εκδήλωσης των προβλημάτων που έχουν καταγραφεί και

καθημερινά επιβεβαιώνονται στην ανάπτυξη του λογισμικού. Είναι χαρακτηριστικό ότι το λογισμικό είναι ένα από τα ελάχιστα ανθρώπινα κατασκευάσματα που πωλείται ως έχει, χωρίς *καμία απολύτως* εγγύηση για τις ζημιές που μπορεί να προκαλέσει η χρήση του, όσο σημαντικές και αν είναι αυτές. Στον επόμενο Πίνακα φαίνονται τα σπουδαιότερα από τα προβλήματα αυτά (Βεσκούκης, 2015; Fox and Patterson, 2017).

**Πίνακας 1.** Βασικά προβλήματα κρίσης του Λογισμικού

| | |
|---|---|
| • Εξαιρετικά δύσκολη διαδικασία κατασκευής | Δεν είναι πάντα σαφές ποια βήματα πρέπει να γίνουν, με ποια σειρά, με ποια ενδιάμεσα προϊόντα κ.λπ. |
| • Ανεπαρκής ή και κακή ποιότητα τελικού προϊόντος | Λάθη στην κατασκευή, μη ικανοποίηση του σκοπού. |
| • Μη τήρηση χρονοδιαγραμμάτων | Υπερβολικές και αδικαιολόγητες καθυστερήσεις. |
| • Υπερβάσεις προϋπολογισμών | Κακές αρχικές εκτιμήσεις κόστους. Τελικά προϊόντα με πολλαπλάσιο κόστος από το αρχικά προϋπολογισθέν. |
| • Μεγάλη δυσκολία και συνεπαγόμενο κόστος συντήρησης | Παρενέργειες μεταβολών σε στοιχεία που πριν λειτουργούσαν, πρόχειρες λύσεις. |
| • Δύσκολη κατανόηση εγγράφων, σχεδίων κ.λπ. από διαφορετικούς κατασκευαστές | Στην πράξη, η κατανόηση ενός συστήματος λογισμικού από τρίτους, πλην των κατασκευαστών του, είναι συχνά αδύνατη ή ιδιαίτερα ασύμφορη. |

Προβλήματα  σαν αυτά του προηγούμενου πίνακα, έχουν βρεθεί εδώ και δεκαετίες από την κοινότητα κατασκευαστών και επιστημόνων/ερευνητών στο πεδίου του λογισμικού και έχουν αναφερθεί με πολλούς τρόπους και σε πολλές ευκαιρίες. Συχνά, νέες τεχνολογίες ή προϊόντα που προτείνονται για την ανάπτυξη του λογισμικού

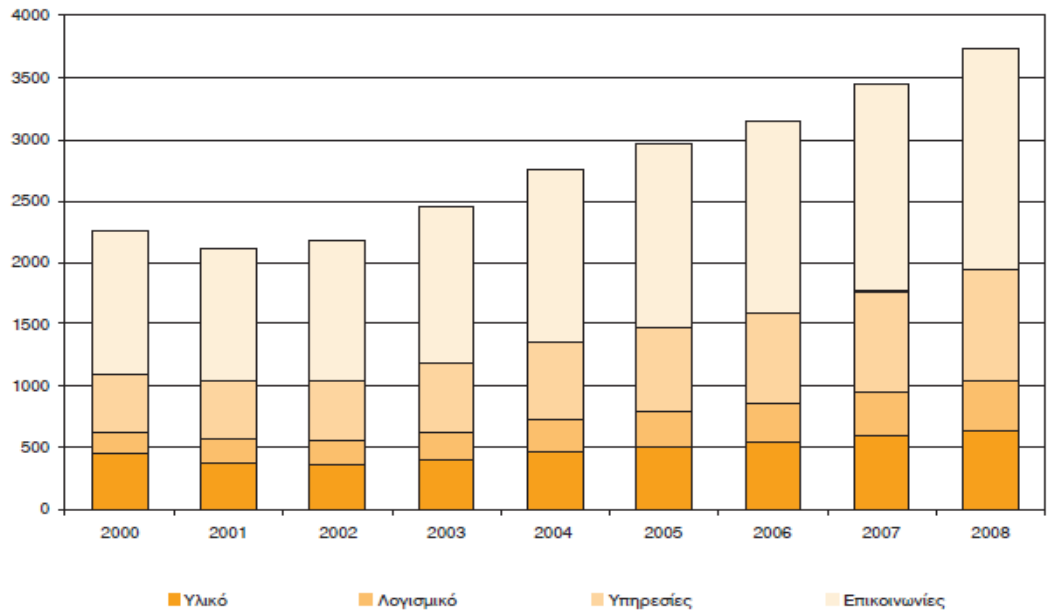
κάνουν επίκληση των προβλημάτων αυτών, ισχυριζόμενα ότι μπορούν να διαθέσουν νέες ικανοποιητικές λύσεις (Βεσκούκης, 2015; Fox and Patterson, 2017).

Ως συνέπεια όλων αυτών, αναπτύχθηκε ένας ειδικός κλάδος της επιστήμης της πληροφορικής που ονομάστηκε "*Τεχνολογία Λογισμικού" (Software Engineering*). Πρόσφατα προτάθηκε η Τεχνολογία Λογισμικού να αποτελέσει εξειδίκευση της επιστήμης του μηχανικού.

## 1.4 Οικονομική σημασία του Λογισμικού

Στις οικονομίες των αναπτυγμένων και υπο-ανάπτυξη χωρών ξεχωρίζει για το δυναμισμό του και τη σημασία του ο τομέας της Πληροφορικής και Επικοινωνιών (ICT). Στον τομέα αυτό περιλαμβάνονται οι οικονομικές δραστηριότητές που αφορούν την πληροφορική και τις ηλεκτρονικές επικοινωνίες. Ο τομέας της πληροφορικής και των επικοινωνιών είναι ένας ταχέως αναπτυσσόμενος τομέας από τα τέλη της δεκαετίας του '70. Παρά το γεγονός ότι δεν αναπτύσσεται πλέον με τους εκρηκτικούς ρυθμούς που αναπτυσσόταν τις δεκαετίες του '80 και '90, όμως εξακολουθεί να αναπτύσσεται παγκοσμίως με ετήσιους ρυθμούς της τάξης του 6%, ενώ υπάρχουν χώρες όπως η Ρωσία, η Ινδία και η Κίνα που ο ετήσιος ρυθμός ανάπτυξης υπερβαίνει το 20% (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015).

Διεθνώς σχεδόν το 50% της αγοράς πληροφορικής και επικοινωνιών αφορά τις υπηρεσίες επικοινωνίας (φωνή και δεδομένα) και τον εξοπλισμό επικοινωνιών, το 23% τις υπηρεσίες πληροφορικής, το 17% το υλικό υπολογιστών και το 10% το λογισμικό. Όσον αφορά τους ρυθμούς ανάπτυξης των επιμέρους αγορών, η αγορά του λογισμικού αυξάνεται με ετήσιους ρυθμούς της τάξης του 10%, των υπηρεσιών πληροφορικής με ρυθμούς της τάξης του 7,4%, των επικοινωνιών με 5,2% και του υλικού υπολογιστών με 2,3%. Ένα άλλο χαρακτηριστικό αυτών των αγορών είναι ότι στο επίπεδο των τιμών, οι τιμές των διάφορων προϊόντων συνεχώς μειώνονται με χαρακτηριστικότερο παράδειγμα τις τιμές των προϊόντων υλικού (Σχ.1)( Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015; Fox and Patterson, 2017).




**Σχήμα 1.** Η παγκόσμια αγορά της πληροφορικής και επικοινωνιών σε δισεκατομμύρια δολάρια.

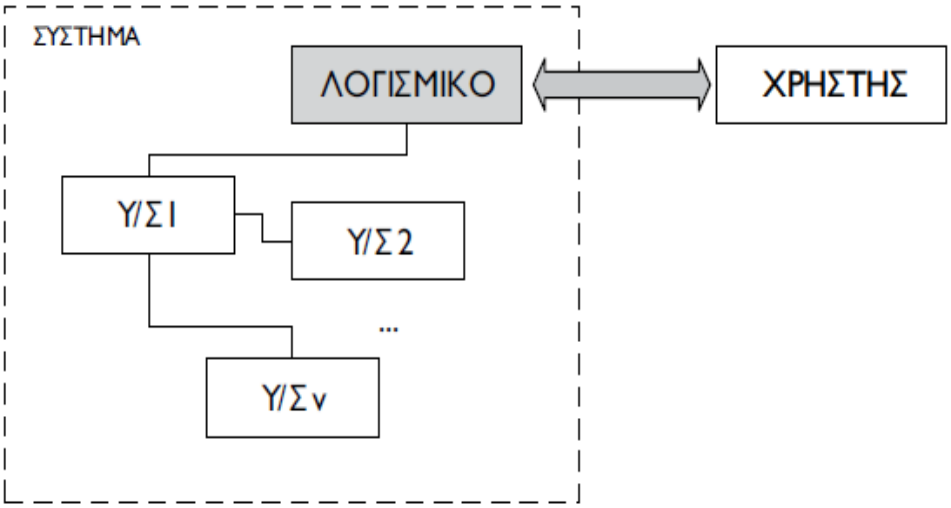
Συνολικά εκτιμάται ότι στις χώρες του ΟΟΣΑ περίπου 15 εκατομμύρια άνθρωποι εργάζονται στον κλάδο της πληροφορικής και των επικοινωνιών. Η δε σύγκλιση νανοτεχνολογίας, βιοτεχνολογίας και τεχνολογιών πληροφορικής υπόσχεται μεγάλες εξελίξεις και προόδους στην ιατρική αλλά και νέες σημαντικές εξελίξεις στα οικονομικά μεγέθη του κλάδου πληροφορικής και επικοινωνιών (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015; Boehm, 1984, 1988).

## 1.5 Το Λογισμικό ως Μέρος Συστημάτων

Συχνά, το λογισμικό αντιμετωπίζεται με λανθασμένο τρόπο όχι ως μέρος ενός ευρύτερου συστήματος με το οποίο αλληλεπιδρά με διάφορους τρόπους αλλά ως αυθύπαρκτη οντότητα. Στην Τεχνολογία Λογισμικού συχνά γίνεται ξεχωριστά λόγος για το *σύστημα* και ξεχωριστά για το *λογισμικό*, και δεν είναι λίγες οι περιπτώσεις όπου μπορεί να δημιουργηθεί σύγχυση σχετικά με τους όρους και την προσέγγιση πολλών οντοτήτων του πραγματικού κόσμου κατά την ανάπτυξη λογισμικού. Υπάρχουν κυρίως δύο περιπτώσεις (Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015; Fox and Patterson, 2017):

- Το λογισμικό αποτελεί εσωτερικό συστατικό ενός τεχνητού, μη υπολογιστικού συστήματος.
- Το λογισμικό λειτουργεί αυτοτελώς (αυτόνομα) σε ένα υπολογιστικό σύστημα.

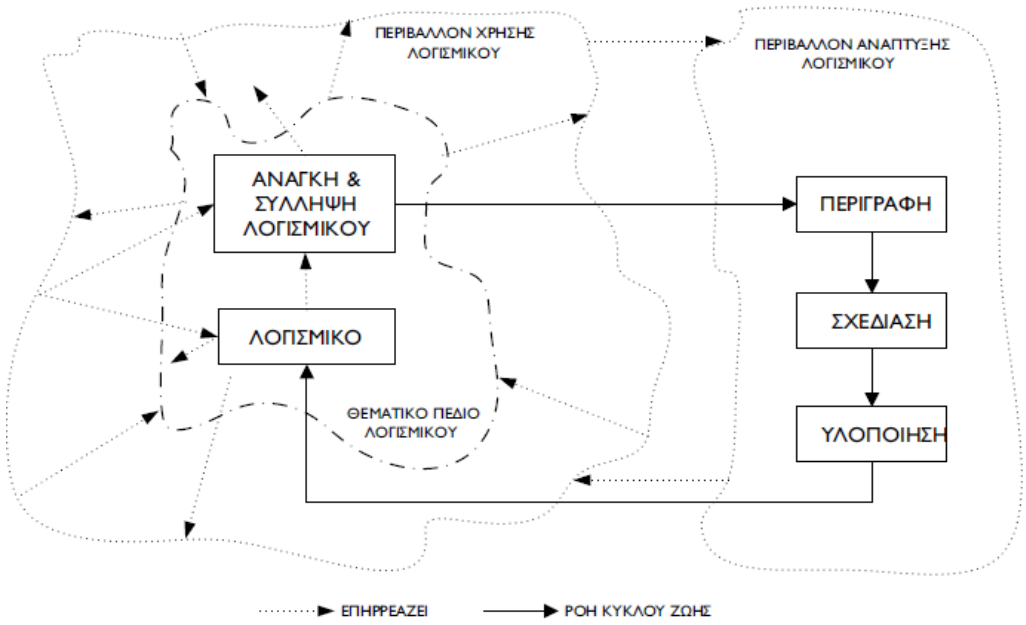
Για παράδειγμα, για το πρώτο μπορούμε να αναφέρουμε όλες τις περιπτώσεις όπου μια συσκευή λειτουργεί χρησιμοποιώντας λογισμικό, όπως οι μηχανές αυτόματης πώλησης, οι ψηφιακοί αυτοματισμοί και, σύντομα, αρκετές οικιακές συσκευές. Επίσης, στην πρώτη κατηγορία ανήκουν πολύπλοκα συστήματα όπου το λογισμικό ή η υπολογιστική μονάδα στην οποία αυτό εκτελείται λειτουργεί συνδεδεμένη με άλλες συσκευές, όπως τα ιατρικά μηχανήματα ανάλυσης και απεικόνισης (MRI, fMRI κοκ.), τα συστήματα ελέγχου εναέριας κυκλοφορίας κ.ά. (Σχ.2). Το λογισμικό έχει τη δυνατότητα να είναι μέρος πολλών συστημάτων. Έτσι, ο χρήστης, αλληλεπιδρώντας με τα συστήματα, μπορεί να χρησιμοποιεί λογισμικό χωρίς να έχει άμεση αντίληψη αυτού του γεγονότος (Βεσκούκης, 2015; Boehm, 1988; Faitly, 1985; Williams, 1984).




**Σχήμα 2.** Το Λογισμικό ως μέρος Συστημάτων

Στις περιπτώσεις αυτές, κατά την ανάπτυξη του λογισμικού οφείλουν να λαμβάνονται υπόψη τα ειδικά χαρακτηριστικά των συσκευών που αποτελούν τα υπόλοιπα μέρη του συστήματος. Τα χαρακτηριστικά τέτοιων συσκευών, χωρίς να αποτελούν αυτά καθαυτά χαρακτηριστικά του λογισμικού, καθορίζουν σε μεγάλο βαθμό τη δομή και τη συμπεριφορά του (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988).

Στη δεύτερη περίπτωση, όπου το λογισμικό στεγάζεται απλώς σε ένα υπολογιστικό σύστημα, δεν αποτελεί με τη δομική αλλά με τη λειτουργική έννοια μέρος ενός ευρύτερου οργανισμού από τον οποίο καθορίζεται και τον οποίο με τη σειρά του καθορίζει (Σχ.3)(Γιακουμάκης και Διαμαντίδης, 2009; Βεσκούκης, 2015).




**Σχήμα 3.** Αλληλεπιδράσεις στην ανάπτυξη του λογισμικού

## 1.6 Το Λογισμικό ως Βιομηχανικό Προϊόν

Το Λογισμικό απασχολεί σπουδαίο ποσοστό της όλης οικονομικής δραστηριότητας. Μπορεί να θεωρηθεί ότι είναι ένα βιομηχανικό προϊόν με αντίστοιχη βιομηχανία. Όπως κάθε βιομηχανικός κλάδος, πρέπει και η βιομηχανία του λογισμικού να υιοθετήσει εκείνες τις διαδικασίες και τεχνικές παραγωγής που θα εξασφαλίζουν μεγαλύτερη απόδοση και καλύτερα προϊόντα. Αυτός ακριβώς είναι ο στόχος της τεχνολογίας λογισμικού. Για αν επιτευχθεί αυτός θα πρέπει να ισχύουν τα εξής στοιχεία (Βεσκούκης, 2015; Davis et al., 1988; Faitly, 1985; Williams, 1984):

- *διάρκεια ζωής:* το λογισμικό ως επένδυση αποσβένεται μόνο εάν παραμείνει σε χρήση και αξιοποίηση μακρύ χρονικό διάστημα.
- *αξιοπιστία:* το λογισμικό πρέπει να είναι αξιόπιστο σε όλες τις πιθανές συνθήκες λειτουργίας.
- *αποδοτικότητα:* το λογισμικό πρέπει να έχει την αίσθηση της οικονομίας των πόρων που χρησιμοποιεί (κυρίως μνήμη και υπολογιστική ισχύς).
- *λειτουργικότητα:* το λογισμικό πρέπει να εξυπηρετεί τον εργαζόμενο και να του δίνει τη δυνατότητα να δει με νέο και δημιουργικό τρόπο την εργασία του. Επίσης πρέπει να ανταποκρίνεται στις δυνατότητες του χρήστη με φιλικότητα.
- *ποιότητα:* η ποιότητα είναι ένα ζητούμενο όλων των βιομηχανικών προϊόντων και επομένως και του λογισμικού. Το λογισμικό έχει τα δικά του στοιχεία ποιότητας που το χαρακτηρίζουν.

## 1.7 Η τεχνολογία Λογισμικού

Η απάντηση στην κρίση λογισμικού ήταν η δημιουργία της τεχνολογίας του λογισμικού. Με τον όρο αυτό, η επιστημονική κοινότητα εννοούσε ότι η εφαρμογή αρχών των υπαρχόντων κλάδων των μηχανικών στη διαδικασία ανάπτυξης λογισμικού θα επέτρεπε την υπέρβαση της κρίσης λογισμικού. Τα χρόνια που ακολούθησαν η επαγγελματική & επιστημονική κοινότητα επιχείρησαν να αντιμετωπίσουν τα προβλήματα με την κρίση λογισμικού, μέσω ανάπτυξης τεχνικών διαχείρισης της πολυπλοκότητας των μεγάλων συστημάτων λογισμικού, διαχείρισης της συνεργασίας ομάδων προγραμματιστών και μέτρησης της ποιότητας λογισμικού Όλα αυτά αποτέλεσαν λόγους για την αποδοχή της τεχνολογίας λογισμικού από τον κόσμο της πράξης, αφού συνεπάγεται εξοικονόμηση χρημάτων και "β*έλτιστα*" συστήματα λογισμικού (Βεσκούκης, 2015; Boehm, 1984, 1988; Faitly, 1985; Williams, 1984).

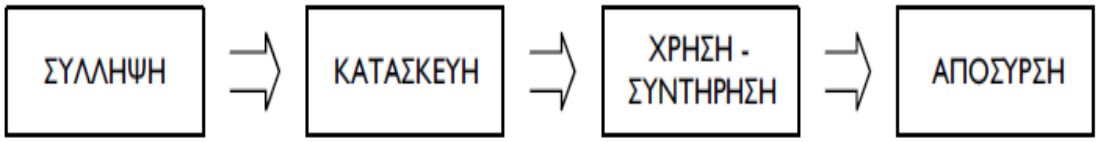
Επιθυμητά χαρακτηριστικά του λογισμικού και της διαδικασίας κατασκευής του είναι η ποιότητα, η μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα και το ελάχιστο δυνατό κόστος παραγωγής και συντήρησης. Οι έννοιες ποιότητα, αυτοματοποίηση, παραγωγικότητα και κόστος είναι σε πολλές περιπτώσεις αντίθετες. Εντός του πεδίου της Τεχνολογίας Λογισμικού είναι ο καθορισμός των ενεργειών και της αλληλουχίας με την οποία αυτές πρέπει να γίνονται *(software process*), καθώς και η περιγραφή με σαφή και κατανοητό τρόπο όλων των προϊόντων που παράγονται κατά την εκτέλεση αυτών των ενεργειών. Το τελικό παραδοτέο προϊόν κάθε ενέργειας ανάπτυξης λογισμικού είναι ο εκτελέσιμος κώδικας, δηλαδή ένα σύνολο εντολών άμεσα εκτελέσιμων από έναν Η/Υ κάτω από συγκεκριμένες και γνωστές εκ των προτέρων βασικές προϋποθέσεις. Το σύνολο αυτών των εντολών αποτελεί μια περιγραφή του τρόπου εκτέλεσης των εργασιών που αυτοματοποιούνται με τη χρήση μιας εφαρμογής λογισμικού (Βεσκούκης, 2015; Boehm, 1988; Davis et al., 1988; Faitly, 1985; Williams, 1984).

Δεν είναι δυνατό η κατασκευή του λογισμικού να οδηγήσει κατευθείαν στον εκτελέσιμο κώδικα, όπως άλλωστε καμία απολύτως τεχνική κατασκευή δεν μπορεί να γίνει κατευθείαν, χωρίς να έχουν προηγηθεί αναλύσεις, μελέτες και σχέδια. Ωστόσο, ένα στοιχείο που διαφοροποιεί σημαντικά το λογισμικό από τις παραδοσιακές τεχνικές κατασκευές είναι ότι η κατασκευή του δεν είναι μια σειριακά ακολουθούμενη διαδικασία που ολοκληρώνεται με την κατασκευή του παραδοτέου

προϊόντος, αλλά το αρχικό αυτό παραδοτέο (1η έκδοση εκτελέσιμου κώδικα και αντίστοιχο υλικό τεκμηρίωσης) υπόκειται συχνά σε πολλές τροποποιήσεις και επαναδιαμορφώσεις. Συνήθεις αιτίες για αυτές στο λογισμικό είναι τα εξής (Βεσκούκης, 2015; Boehm, 1984, 1988):

- η διόρθωση σφαλμάτων,
- η βελτιστοποίηση της απόδοσης,
- η αυτοματοποίηση της εκτέλεσης νέων εργασιών και
- η ενσωμάτωση μεταβολών που οφείλονται σε αλλαγές που συμβαίνουν στον πραγματικό κόσμο.

Η πραγματοποίηση μεταβολών/διορθώσεων στις εφαρμογές λογισμικού αναφέρεται με τον όρο "*συντήρηση λογισμικού" (software maintenance*). Όλες οι φάσεις από τις οποίες διέρχεται το λογισμικό αναφέρονται ως "*κύκλος ζωής λογισμικού" (software life cycle*)(Σχ.4). Γίνεται σαφές ότι *η Τεχνολογία Λογισμικού δεν ασχολείται μόνο με την κατασκευή, αλλά με ολόκληρο τον κύκλο ζωής του λογισμικού.* Με βάση το χρόνο, πρόκειται για το διάστημα από η σύλληψη της ιδέας της κατασκευής μιας εφαρμογής λογισμικού μέχρι την απόσυρση αυτής από τη χρήση (Βεσκούκης, 2015).




**Σχήμα 4.** Γενικές φάσεις του κύκλου ζωής του λογισμικού

## 1.8 Παράγοντες Επιτυχίας Λογισμικού

Εξειδικευμένες έρευνες έχουν δείξει ότι για την επιτυχή ολοκλήρωση ενός έργου ανάπτυξης λογισμικού οι σημαντικότεροι παράγοντες που επιδρούν θετικά είναι οι ακόλουθοι (Βεσκούκης, 2015; Boehm, 1984; Davis et al., 1988; Faitly, 1985; Williams, 1984):

- *Επιχειρησιακή στήριξη*. Η στάση της διοίκησης του οργανισμού που αναπτύσσει το λογισμικό επηρεάζει την πρόοδο και το αποτέλεσμα της διαδικασίας του έργου ανάπτυξης. Ένα έργο ανάπτυξης θα πρέπει να έχει την προσοχή της επιτελικής διοίκησης. Η επιτελική διοίκηση καθορίζει τους στόχους και την εμβέλεια του έργου, συμμετέχει στις διαπραγματεύσεις για

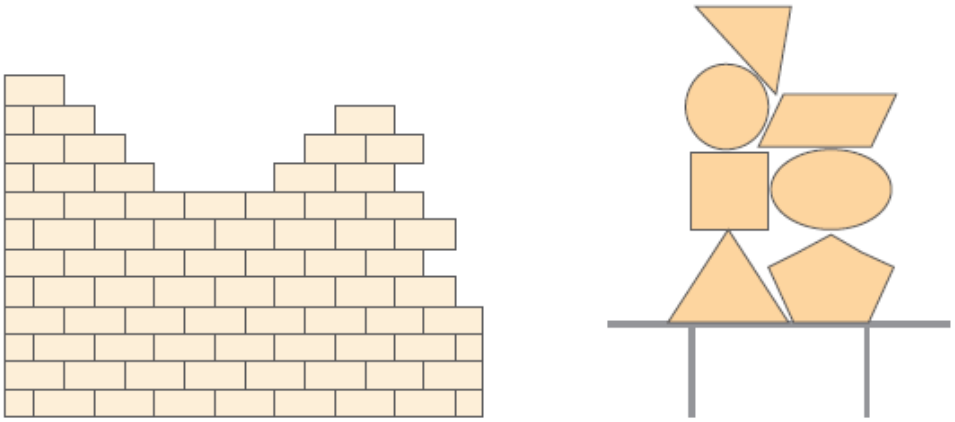
τη στρατηγική του, έχει την υψηλού επιπέδου εποπτεία του και λαμβάνει κεντρικές αποφάσεις για την εξέλιξή του. Απουσία επιχειρησιακής στήριξης δημιουργεί προϋποθέσεις καθυστερήσεων και εμποδίων στην εξέλιξη του έργου.

- *Εμπλοκή χρηστών*. Οι χρήστες που θα χρησιμοποιήσουν το προϊόν του λογισμικού είναι και οι τελικοί κριτές για την επιτυχία ή την αποτυχία του λογισμικού. Έτσι, είναι λογικό να υπάρχει επιθυμία για τη συμμετοχή των χρηστών σε όλη την πορεία ενός έργου ανάπτυξης, για να μειώσουμε δυσάρεστες εκπλήξεις, όταν αυτοί κληθούν να το χρησιμοποιήσουν. Η ανάπτυξη λογισμικού χωρίς την ενεργή συμμετοχή των χρηστών είναι ένας σημαντικός κίνδυνος αποτυχίας ενός έργου.
- *Ικανός διοικητής έργου*. Η διοίκηση ενός έργου ανάπτυξης είναι μία πολύπλοκη διαδικασία. Ένας έμπειρος διοικητής έργου, ο οποίος συνδυάζει τεχνικές γνώσεις, με ένα σύνολο διοικητικών, επικοινωνιακών και άλλων δεξιοτήτων συμβάλλει σημαντικά στην πετυχημένη ολοκλήρωση του έργου.
- *Σαφείς επιχειρησιακοί στόχοι*. Ένας οργανισμός επιδιώκει με τη χρήση του λογισμικού να λύσει κάποια προβλήματα. Δεν αρκεί το λογισμικό να χαρακτηριστεί ως μοντέρνο ή σύγχρονο για να γίνει αποδεκτό. Θα πρέπει να υπάρξουν σαφείς επιχειρησιακοί στόχοι, πριν ξεκινήσει η ανάπτυξη του λογισμικού. Εμπλεκόμενοι θα πρέπει να είναι όλοι οι εμπλεκόμενοι σε ένα έργο ανάπτυξης, όπως ο πελάτης, οι χρήστες, ο διοικητής του έργου, οι μηχανικού λογισμικού, οι συγγραφείς των εγχειριδίων χρήσης κ.ά.
- *Εστιασμένο πεδίο εφαρμογής του προϊόντος*. Μετά τους επιχειρησιακούς στόχους, υπάρχει την κατάλληλη οριοθέτηση του πεδίου εφαρμογής του προϊόντος. Το προϊόν πρέπει να στοχεύει στην υποστήριξη εκείνων των λειτουργιών που αφορούν τους επιχειρησιακούς στόχους και μόνο. Είναι συνήθης πρακτική το προϊόν να έχει στόχους υπέρμετρα φιλόδοξους και καμιά φορά και μη ρεαλιστικούς. Η επιδίωξη ανέφικτων στόχων οδηγεί τελικά στην αποτυχία επίτευξης ακόμα και των βασικών. Επομένως, θα πρέπει να υπάρχει μία κοινή συμφωνία οριοθέτησης της λειτουργικότητας του τελικού προϊόντος σε συνδυασμό με την εφικτότητα των χρονοδιαγραμμάτων και του προϋπολογισμένου κόστους.

- *Πρότυπες υποδομές ανάπτυξης*. Οι έρευνες που μελετούν έργα λογισμικού δείχνουν ότι ένα πολύ υψηλό ποσοστό της προσπάθειας σε ένα έργο ανάπτυξης λογισμικού δεν εστιάζουν αποκλειστικά στην ανάπτυξη λογισμικού για την επίλυση του συγκεκριμένου επιχειρησιακού προβλήματος. Σημαντική προσπάθεια της ανάπτυξης αναλώνεται στην ανάπτυξη "*κώδικα υποδομής*" για την υποστήριξη της ανάπτυξης ειδικών τεχνικών ζητημάτων του λογισμικού. Ένας λοιπόν παράγοντας επιτυχίας στην τεχνική διάσταση ενός έργου ανάπτυξης του λογισμικού είναι η χρήση έτοιμων, διαδεδομένων ώριμων και δοκιμασμένων υποδομών, τεχνολογιών και εργαλείων που θα επιτρέψουν στους μηχανικούς λογισμικού να επικεντρωθούν στη λύση του επιχειρησιακού προβλήματος με αποδοτικό τρόπο.

## 1.9 Η Ιδιαιτερότητα του Λογισμικού

Υπάρχουν χαρακτηριστικά του λογισμικού τα οποία εμποδίζουν την επινόηση μεθόδων, τεχνικών και εργαλείων που να εξασφαλίζουν την επιτυχημένη ανάπτυξη. Τα χαρακτηριστικ αυτά είναι η εγγενής πολυπλοκότητα του λογισμικού, η προσαρμοστικότητά του, η άυλη φύση του και η συμμόρφωση με το περιβάλλον του. Το σημαντικότερο χαρακτηριστικό του λογισμικού είναι η εγγενής πολυπλοκότητά του. Τα δομικά στοιχεία του λογισμικού παρουσιάζουν μεγάλη μεταβλητότητα σε αντίθεση με άλλους κλάδους, όπως οι κατασκευές, που χρησιμοποιούν ένα σχετικά περιορισμένο σύνολο πρωτογενών υλικών. Η πολυπλοκότητα του λογισμικού αυξάνεται, όταν αυτά τα δομικά στοιχεία επικοινωνούν μεταξύ τους (Σχ.5). Οι διεπαφές των δομικών στοιχείων αυξάνουν μη-γραμμικά την πολυπλοκότητά του. Η πολυπλοκότητα είναι ένα ουσιώδες χαρακτηριστικό του λογισμικού το οποίο επιχειρούμε να διαχειριστούμε κατασκευάζοντας αφαιρέσεις, δηλαδή μοντέλα που αφαιρούν κάποιες λεπτομέρειες του λογισμικού. Πολλές φορές όμως οι λεπτομέρειες αυτές είναι εξ ορισμού σημαντικές για την περιγραφή του λογισμικού. Το δεύτερο ουσιώδες χαρακτηριστικό του λογισμικού είναι η προσαρμοστικότητά του. Το λογισμικό είναι αντικείμενο συνεχών αλλαγών. Σε αντίθεση με άλλα βιομηχανικά προϊόντα, το λογισμικό αλλάζει ακόμη και μετά την παραγωγή του και επίσης, η απαίτηση για αλλαγές είναι ίσως πιο έντονες στα πετυχημένα προϊόντα, παρά στα αποτυχημένα (Σχ.6)(Βεσκούκης, 2015).



**Σχήμα 5.** Το λογισμικό κατασκευάζεται από ανομοιογενή υλικά με σύνθετες διεπαφές σε αντίθεση με άλλες κατασκευές

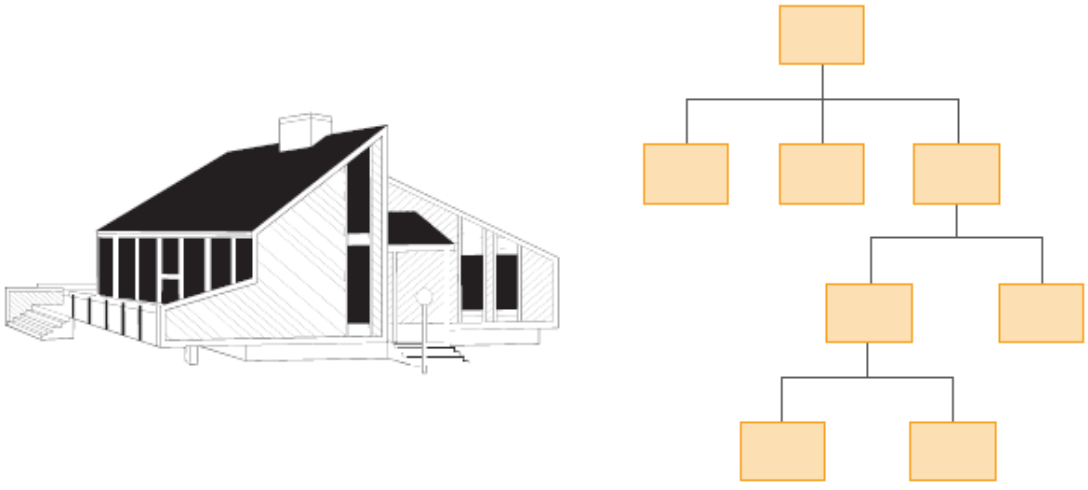

**Σχήμα 6.** Το λογισμικό αλλάζει και μετά την κατασκευή του

Η άυλη φύση του λογισμικού εμποδίζει τον ανθρώπινο νου να το συλλάβει στην ολότητά του, οπτικοποιώντας τα χαρακτηριστικά του. Τα σχέδια ενός κτηρίου μάς δείχνουν την πραγματική του γεωμετρία. Ο πολιτικός μηχανικός και ο αρχιτέκτονας μπορούν να αξιολογήσουν με μεγάλη ακρίβεια τα χαρακτηριστικά του κτηρίου μελετώντας μόνο τα σχέδια. Τα μοντέλα που κατασκευάζουμε για το λογισμικό, όπως ήδη έχουμε αναφέρει, αδυνατούν να οπτικοποιήσουν πλήρως τη συμπεριφορά του (Σχ.7). Οι προσπάθειες δημιουργίας ενός μοντέλου αφαιρούν ουσιώδη χαρακτηριστικά του λογισμικού, με αποτέλεσμα να αδυνατούμε να κατανοήσουμε πλήρως τη συμπεριφορά του (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Faitly, 1985; Williams, 1984).

Η κατασκευή των κτηρίων διέπεται από νόμους ή παραδοχές που αποτελούν μία διαχρονική κατάκτηση του κατασκευαστικού κλάδου. Η εργονομία των κτηρίων, η

αντισεισμικότητά τους, η ηχομόνωσή τους, αποτελούν ένα σταθερό γνωσιακό υπόβαθρο για τους μηχανικούς. Κάτι τέτοιο όμως δε συμβαίνει με το λογισμικό. Το λογισμικό καλείται να συμμορφωθεί με το υλικό, με υφιστάμενα συστήματα λογισμικού και με κανόνες και παραδοχές των οργανισμών που το χρησιμοποι- ούν, οι οποίοι με τη σειρά τους λειτουργούν με κανόνες που δεν είναι σταθεροί, αλλάζουν συνεχώς με την πάροδο του χρόνου και εμφανίζουν πολυπλοκότητα που δεν μπορεί να προβλεφθεί. Σε αυτό το παιγνίδι των συνεχών αλλαγών το λογισμικό είναι που θα προσαρμοστεί στο περιβάλλον του και όχι το περιβάλλον στις ανάγκες του λογισμικού (Βεσκούκης, 2015; Boehm, 1984, 1988).

Τα παραπάνω ιδιαίτερα χαρακτηριστικά του λογισμικού προσθέτουν πολυπλοκότητα και στη διαδικασία ανάπτυξής του συγκριτικά με τη διαδικασία παραγωγής άλλων προϊόντων. Αυτό όμως που μπορούμε να δανειστούμε από την παραγωγή άλλων προϊόντων είναι η τεχνογνωσία που αποκτάται από την εφαρμογή καλά ορισμένων μεθόδων και διαδικασιών στη βιομηχανική παραγωγή αγαθών ή υπηρεσιών και οι οποίες έχουν αποδειχθεί αποτελεσματικές στην πράξη. Ανάλογη τεχνογνωσία συσσωρεύει και η παραγωγή του λογισμικού. Η τεχνογνωσία αυτή μπορεί να αξιοποιηθεί, ώστε η ανάπτυξη του λογισμικού ως διαδικασία να οργανωθεί με τρόπο που να εξασφαλίζει αξιόπιστα προϊόντα μέσω αξιόπιστων διαδικασιών παραγωγής που επιτυγχάνουν τους στόχους ποιότητας, κόστους και χρόνου που εκ των προτέρων έχουν τεθεί. Η προσπάθεια μας αυτή θα εξελιχθεί έχοντας πάντα υπόψη μας ότι παρ' όλα αυτά η ανάπτυξη λογισμικού είναι μια διαδικασία με σημαντικές διαφορές από αντίστοιχες παραγωγής άλλων προϊόντων (Βεσκούκης, 2015).



**Σχήμα 7.** Τα μοντέλα του λογισμικού δεν οπτικοποιούν πλήρως τη συμπεριφορά του

## 1.10 Λογισμικό και Ποιότητα

Η τεχνολογία λογισμικού είναι μία στρατηγική για την παραγωγή ποιοτικού λογισμικού. Το σημαντικό λοιπόν είναι για να προσδιοριστεί ο όρος ποιοτικό λογισμικό. Τα χαρακτηριστικά του ποιοτικού λογισμικού εξαρτώνται από το ποιος εξετάζει το λογισμικό. Οι χρήστες κρίνουν το λογισμικό ως υψηλής ποιότητας, εάν αυτό κάνει ό,τι αυτοί θέλουν να κάνει με έναν τρόπο εύκολο στη μάθηση και εύκολο στη χρήση. Το λογισμικό πρέπει επίσης να κρίνεται από αυτούς που σχεδιάζουν και γράφουν τον κώδικα καθώς επίσης και από αυτούς που θα συντηρούν τα προγράμματα μετά την ολοκλήρωση της ανάπτυξής τους. Επομένως, το υψηλής ποιότητας λογισμικό έχει χαρακτηριστικά που ανταποκρίνονται στις ανάγκες των χρηστών, των ανθρώπων ανάπτυξης του λογισμικού και των συντηρητών του λογισμικού (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988).

Τα ποιοτικά χαρακτηριστικά του λογισμικού αποτελούν σημαντικό κριτήριο για την ικανοποίηση των χρηστών του λογισμικού. Ένας τελικός χρήστης μπορεί να εργαστεί με λογισμικό που θα του παρέχει λιγότερη λειτουργικότητα από αυτή που θα επιθυμούσε, αλλά δεν ανέχεται λογισμικό που καθυστερεί υπερβολικά να ανταποκριθεί στους χειρισμούς του. Μπορεί επομένως να είναι απαιτητικός σε κάποιο ποιοτικό χαρακτηριστικό του λογισμικού περισσότερο από τη λειτουργικότητα που αυτό παρέχει (Boehm, 1984, 1988; Faitly, 1985; Williams, 1984).

Δεν υπάρχει ένας κοινά αποδεκτός ορισμός των ποιοτικών χαρακτηριστικών του λογισμικού. Ακολουθούν ορισμένα από τα σημαντικότερα ποιοτικά χαρακτηριστικά του λογισμικού (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988):

- *Διαθεσιμότητα (Availability).* Η διαθεσιμότητα εκφράζει το χρονικό διάστημα λειτουργίας του συστήματος στη διάρκεια του χρόνου. Η διαθεσιμότητα αντανακλά την ταχύτητα διαχείρισης κάποιας αστοχίας του συστήματος. Η διαθεσιμότητα θέτει ερωτήματα όπως, πώς εντοπίζεται μία αστοχία, η εκτίμηση της συχνότητας των αστοχιών, η απόκριση σε περίπτωση αστοχίας, το χρονικό διάστημα για το οποίο είναι επιτρεπτή η διακοπή της λειτουργίας του συστήματος κα.
- *Απόδοση (Performance, Efficiency).* Η απόδοση αναφέρεται στην ταχύτητ απόκρισης του συστήματος, όταν ζητείται να παρέχει τις υπηρεσίες του. Η

απόδοση μπορεί να επεκταθεί και στην αποδοτικότητα η οποία εκφράζει την αξιοποίηση των διαθεσίμων πόρων του υλικού.

- *Ευελιξία ή επεκτασιμότητα (Flexibility, Extensibility*). Η ευελιξία σχετίζεται με την ευκολία προσαύξησης της λειτουργικότητας του λογισμικού.
- *Ακεραιότητα (Integrity*). Η ακεραιότητα συμπεριλαμβάνει την ασφάλεια, δηλ. την ικανότητα του συστήματος να αντιστέκεται σε πιθανές απειλές από μη εξουσιοδοτημένους χρήστες. Επεκτείνει την ασφάλεια και σε άλλες απειλές όπως οι *ιοί υπολογιστών (computer virus*) και οι *επιθέσεις άρνησης υπηρεσίας (denial of service)*.
- *Διαλειτουργικότητα (Interoperability*). Η διαλειτουργικότητα εκφράζει την ευκολία με την οποία το υπό ανάπτυξη σύστημα μπορεί να ανταλλάξει δεδομένα ή υπηρεσίες με άλλα συστήματα. Η διαλειτουργικότητα επηρεάζει σε μεγάλο βαθμό τις διεπαφές του συστήματος με το περιβάλλον του.
- *Συντηρησιμότητα (Maintainability*). Η συντηρησιμότητα έχει στενή σχέση με την ευελιξία και την ελεγξιμότητα του λογισμικού. Εκφράζει το βαθμό ευκολίας τροποποίησης του λογισμικού και της διόρθωσης των σφαλμάτων.
- *Μεταφερσιμότητα (Portability*). Η μεταφερσιμότητα εκφράζει τη δυνατότητα λειτουργίας του λογισμικού σε διαφορετικά υπολογιστικά περιβάλλοντα, όπως διαφορετικές διατάξεις υλικού, λειτουργικών συστημάτων κ.τ.λ. Η μεταφερσιμότητα μπορεί να αφορά το σύνολο του λογισμικού ή κάποια κρίσιμα τμήματά του. Η μεταφερσιμότητα περιλαμβάνει και τις διαδικασίες *διεθνοποίησης (internationalization*) του λογισμικού.
- *Αξιοπιστία (Reliability*). Είναι η ιδιότητα του λογισμικού να συνεχίζει να λειτουργεί για κάποιο χρονικό διάστημα χωρίς αποτυχία και να εκτελεί τις αναμενόμενες λειτουργίες με την απαιτούμενη ακρίβεια.
- *Επαναχρησιμότητα (Reusability*). Η επαναχρησιμότητα του λογισμικού σχετίζεται με την ευκολία με την οποία κάποιες μονάδες του λογισμικού μπορούν να επαναχρησιμοποιθούν σε άλλα προϊόντα λογισμικού.
- *Ευρωστία (Robustness*). Η ευρωστία εκφράζει την ανοχή του λογισμικού σε σφάλματα κάτω από ιδιάζουσες συνθήκες, όπως συστημικά σφάλματα υλικού ή λειτουργικού συστήματος. Εύρωστο λογισμικό δεν είναι μόνο αυτό που μπορεί να λειτουργήσει και μετά από εσφαλμένη χρήση αλλά και το λογισμικό που μπορεί να τερματίσει προβλέψιμα μετά από αυτή, καθώς επίσης και ότι μπορεί να ανακάμψει κατόπιν ακραίων συνθηκών.

- *Ελεγξιμότητα (testability*). Η ελεγξιμότητα εκφράζει το βαθμό της ευκολίας με την οποία μπορεί να ελεγχθεί το λογισμικό και την ευκολία με την οποία μπορεί να εντοπιστεί η πηγή ενός σφάλματος.
- *Ευχρηστία (Usability*). Η ευχρηστία είναι η ευκολία με την οποία οι τελικοί χρήστες χρησιμοποιούν το λογισμικό, την ταχύτητα εκμάθησης των λειτουργιών του, η προσαρμοστικότητα του λογισμικού στις ανάγκες συγκεκριμένων χρηστών και άλλα θέματα που αφορούν κυρίως στη διεπαφή χρήστη.

Επομένως το ποιοτικό λογισμικό είναι ένα λογισμικό που ικανοποιεί τις ανάγκες των χρηστών και προγραμματιστών που ενδιαφέρονται γι' αυτό. Μπορούμε να θεωρήσουμε ένα λογισμικό ως υψηλής ποιότητας, εφόσον (Βεσκούκης, 2015):

- κάνει ό,τι οι χρήστες θέλουν να κάνει,
- χρησιμοποιεί τους πόρους του υπολογιστή σωστά και αποδοτικά,
- είναι εύκολο για το χρήστη να το μάθει και να το χρησιμοποιήσει,
- οι ειδικοί μπορούν να σχεδιάσουν, να κωδικοποιήσουν, να ελέγξουν και να συντηρήσουν το σύστημα σχετικά εύκολα.

## 1.11 Η Ανάπτυξη Λογισμικού

Κατά αναλογία με την παραγωγή των βιομηχανικών προϊόντων η διαδικασία της ανάπτυξης του λογισμικού μπορεί να αναπαρασταθεί σύμφωνα με το Σχ.8. Η βιομηχανία ανάπτυξης λογισμικού θα πρέπει να συλλέξει τις απαιτήσεις των πελατών της για ένα προϊόν και με τις κατάλληλες ενέργειες να κατασκευάσει το προϊόν για τους πελάτες της. Με μια πρώτη ματιά η βιομηχανία αυτή θα πρέπει να οργανώσει την εργασία της σε τρεις διακριτές φάσεις. Την ανάλυση, την κατασκευή και τον έλεγχο. Σκοπός της ανάλυσης είναι να συμφωνηθεί με τους χρήστες του προϊόντος ποιες ακριβώς λειτουργίες αυτό θα επιτελεί, ποιοι περιορισμοί τίθενται στη λειτουργία του και ποιες επιδόσεις αυτό πρέπει να επιτυγχάνει. Στην κατασκευή θα αναπτύξουμε το προϊόν που ικανοποιεί τις απαιτήσεις της ανάλυσης και τέλος στον έλεγχο θα επιβεβαιώσουμε ότι όντως το προϊόν που αναπτύχθηκε ικανοποιεί τις απαιτήσεις των χρηστών του (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Faitly, 1985; Williams, 1984).


**Σχήμα 8.** Η βιομηχανική ανάπτυξη του λογισμικού

## 1.12 Ο Μηχανικός Λογισμικού

Παράλληλα με τον όρο "*Τεχνολογία Λογισμικού*" προέκυψε και ο όρος "*Μηχανικός Λογισμικού*". Είναι ο ειδικός που θα εφαρμόσει τις αρχές της Τεχνολογίας Λογισμικού. Μεταξύ των αρχών είναι και η εφαρμογή διεθνών ή έγκυρων προτύπων στη διαδικασία ανάπτυξης που θα εγγυώνται εμπιστοσύνη στο παραγόμενο προϊόν (Βεσκούκης, 2015; Boehm, 1984, 1988).

Ο Μηχανικός Λογισμικού διαφέρει από έναν προγραμματιστή. Η διαφορά γίνεται αντιληπτή, κατ' αναλογία, από το χώρο των κατασκευών κτηρίων. Η κατασκευή ενός μεγάλου οικοδομήματος απαιτεί επιστημονικές ειδικότητες γεωλόγων, πολιτικών μηχανικών, αρχιτεκτόνων, ηλεκτρολόγων μηχανικών, project manager, ομάδες τεχνικών, όταν για την κατασκευή ενός μικρού κτίσματος ολίγων τετραγωνικών μέτρων αρκεί ένας μόνο ειδικός, χωρίς σχέδια αρχιτεκτονικά ή στατικά, για να ολοκληρωθεί το έργο. Η πολυπλοκότητα ενός μεγάλου οικοδομήματος προαπαιτεί μέθοδο, τεχνικές, εργαλεία και οργάνωση, απαιτήσεις που δεν αφορούν έργα του ενός ατόμου. Οι επιστήμονες πληροφορικής έχουν ως βασικό εφόδιο τις γνώσεις που σχετίζονται με την επιστήμη των υπολογιστών. Ορισμένοι μάλιστα διά μέσου της έρευνας φθάνουν σε υψηλά σημεία εξειδίκευσης σε κάποιο στενό ερευνητικό πεδίο. Οι μηχανικοί λογισμικού χρησιμοποιούν τις γνώσεις της επιστήμης των υπολογιστών για την επίλυση πρακτικών προβλημάτων. Για παράδειγμα, για να λύσουν ένα πρόβλημα δεν αρκούν μόνο οι γνώσεις κάποιου επιστημονικού κλάδου. Μάλιστα μόνο με αυτές συνήθως δε λύνεται κανένα πρόβλημα (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Faitly, 1985; Williams, 1984).

Στο επόμενο σχήμα φαίνεται οπτικά ότι στο πεδίο της γνώσης της τεχνολογίας λογισμικού εκτός από την επιστήμη των υπολογιστών συμμετέχουν και

επιστημονικοί κλάδοι και κλάδοι της μηχανικής, ακόμα και κλάδοι της ψυχολογίας και της κοινωνιολογίας (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Williams, 1984):


**Σχήμα 9.** Το πεδίο γνώσης της Τεχνολογίας Λογισμικού

Οι μηχανικοί λογισμικού ως επιστήμονες πληροφορικής χρησιμοποιούν επιπλέον εργαλεία, μεθόδους-τεχνικές, διαδικασίες, πρότυπα και αρχιτεκτονικές προσεγγίσεις για να επαυξήσουν την ποιότητα του λογισμικού που παράγουν. Ο στόχος τους είναι η χρήση αποδοτικών και παραγωγικών πρακτικών, ώστε να καταστεί ο υπολογιστής αποτελεσματικός λύτης προβλημάτων (Davis et al., 1988; Faitly, 1985; Williams, 1984).

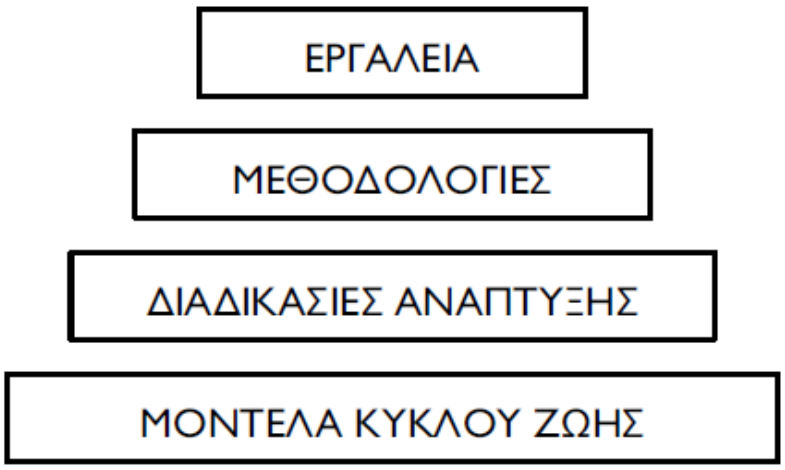
Η τεχνολογία λογισμικού αποτελεί έναν ξεχωριστό κλάδο της πληροφορικής και ο ρόλος του μηχανικού λογισμικού μια ξεχωριστή επαγγελματική ειδικότητα. Λόγω της σημασίας που έχει αποκτήσει και συνεχίζει να αποκτά το λογισμικό, υπάρχουν διεθνώς κινήσεις που αποσκοπούν στον προσδιορισμό των γνώσεων και δεξιοτήτων που θα πρέπει να έχει ο μηχανικός λογισμικού. Μεταξύ αυτών των προσπαθειών συμπεριλαμβάνονται οι παρακάτω (Βεσκούκης, 2015):

- Έχει καταρτιστεί το σώμα της γνώσης της τεχνολογίας λογισμικού.
- Οργανισμοί όπως η IEEE και η ACM έχουν καταρτίσει το περιεχόμενο προγραμμάτων σπουδών για την τεχνολογία λογισμικού.

- Η IEEE έχει συντάξει κώδικα δεοντολογίας για την άσκηση του επαγγέλματος (www.computer.org).
- Η IEEE έχει διαδικασίες πιστοποίησης ειδικά για την ανάπτυξη λογισμικού (www.computer.org).

## 1.13 Μοντέλα Κύκλου Ζωής Λογισμικού

Ένα μοντέλο κύκλου ζωής λογισμικού είναι μια περιγραφή των δραστηριοτήτων και των επιμέρους φάσεων από τις οποίες διέρχεται μια εφαρμογή λογισμικού από τη σύλληψη μέχρι την απόσυρσή της, καθώς και των εργασιών που λαμβάνουν χώρα σε καθεμία από τις φάσεις αυτές. Στο επόμενο σχήμα φαίνεται η σχέση μεταξύ των εννοιών "*μοντέλο κύκλου ζωής*", "*διαδικασία ανάπτυξης*", "μ*εθοδολογία*", καθώς και "*εργαλείο*", οι οποίες ορίστηκαν προηγουμένως. Μια έννοια που βρίσκεται χαμηλότερα στην πυραμίδα αποτελεί το υπόβαθρο πάνω στο οποίο βασίζεται η έννοια που βρίσκεται στο αμέσως ψηλότερο σημείο κ.ο.κ. (Boehm, 1984, 1988; Williams, 1984).




**Σχήμα 10.** Σχέσεις εννοιών στην ανάπτυξη του λογισμικού

Τα μοντέλα κύκλου ζωής λογισμικού προσδιορίζουν τις διαδικασίες ανάπτυξης οι οποίες λαμβάνουν χώρα κατά τις γενικές φάσεις "*κατασκευή*" και "*χρήση – συντήρηση*" , προσδιορίζοντας τις επιμέρους φάσεις στις οποίες αυτές αναλύονται, τα προϊόντα που παράγονται σε καθεμία από αυτές, καθώς και τη σειρά εκτέλεσής τους. Σε κάθε διαδικασία ανάπτυξης μπορούμε να διακρίνουμε περισσότερες από μία επιμέρους φάσεις, ενώ σε κάθε επιμέρους φάση μπορούμε να διακρίνουμε περισσότερες από μία εργασίες. Οι διαδικασίες ανάπτυξης λογισμικού μπορούν να

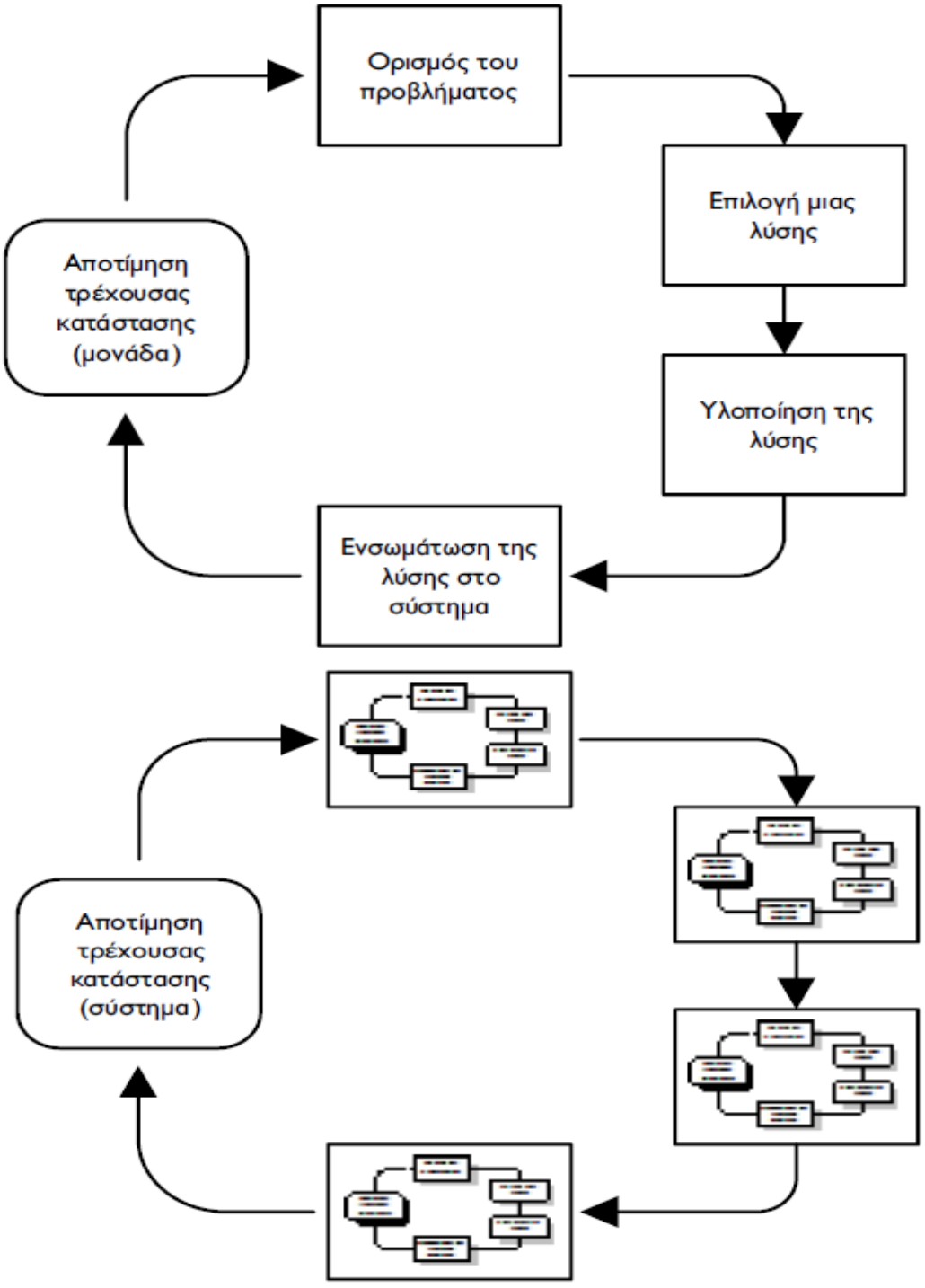
ταξινομηθούν ως ακολούθως (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Faitly, 1985; Williams, 1984):

- *Προδιαγραφή,* δηλαδή καθορισμός των εργασιών που θα επιτελεί το λογισμικό, καθώς και των περιορισμών και των παραδοχών που ισχύουν.
- *Ανάπτυξη,* δηλαδή κατασκευή του λογισμικού. Εδώ, σε όλα τα μοντέλα κύκλου ζωής μπορούμε να διακρίνουμε τρεις επιμέρους φάσεις: την *ανάλυση*, τη *σχεδίαση* και τη *συγγραφή του πηγαίου κώδικα* (source code), την οποία στη συνέχεια θα ονομάζουμε και *κωδικοποίηση*.
- *Επαλήθευση,* δηλαδή επιβεβαίωση της ικανοποίησης των προδιαγραφών και της μη ύπαρξης σφαλμάτων.
- *Εξέλιξη*, δηλαδή επαύξηση των λειτουργικών χαρακτηριστικών του λογισμικού ή τροποποίηση υπαρχουσών, προκειμένου να ικανοποιούνται οι μεταβαλλόμενες ανάγκες.

Ένα μοντέλο κύκλου ζωής λογισμικού στοχεύει στην καθοδήγηση του κατασκευαστή, προκειμένου αυτός να επιτύχει την καλύτερη δυνατή υλοποίηση των διαδικασιών ανάπτυξης λογισμικού. Όταν λέμε "*καλύτερη δυνατή*", εννοούμε περισσότερο παραγωγική, με τα λιγότερα δυνατά σφάλματα και το μικρότερο δυνατό ρίσκο στις εκάστοτε συνθήκες. Τα παραπάνω μπορούν να διαφοροποιούνται ανάλογα με το μέγεθος και το θεματικό πεδίο κάθε εφαρμογής λογισμικού, με την εμπειρία και τα ιδιαίτερα χαρακτηριστικά του κάθε κατασκευαστή και ασφαλώς με το εκάστοτε περιβάλλον ανάπτυξης (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Faitly, 1985; Williams, 1984).

Υπάρχουν αρκετά μοντέλα κύκλου ζωής τα οποία διαφοροποιούνται ως προς τη σύλληψη της ιδέας του τρόπου κατασκευής αλλά και ως προς τις επιμέρους φάσεις που προτείνουν, την επαναληπτικότητα και την εμβέλεια των εργασιών αυτών, τα ενδιάμεσα προϊόντα – συστατικά λογισμικού και την περιγραφή τους, τις οικονομικές και επιχειρηματικές πλευρές της χρήσης τους κ.ά. Καθεμία από τις ενέργειες που περιγράφεται σε ένα μοντέλο κύκλου ζωής είναι μια *διαδικασία επίλυσης προβλημάτων (problem solving process*). Τα βήματα κάθε τέτοιας διαδικασίας φαίνονται στο επόμενο Σχήμα. Ο μηχανικός λογισμικού εκτελεί συνεχώς τέτοιες διαδικασίες επίλυσης προβλημάτων τόσο σε μικροσκοπικό επίπεδο, δηλαδή στις μονάδες του υπό κατασκευή λογισμικού (αριστερό τμήμα του σχήματος), όσο και σε

μακροσκοπικό, δηλαδή για ολόκληρο το σύστημα (δεξί τμήμα του σχήματος)( Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Williams, 1984).




**Σχήμα 11.** Επίλυση προβλημάτων στο Λογισμικό

Τα μοντέλα κύκλου ζωής λογισμικού που έχουν παρουσιαστεί διακρίνονται σε ακολουθιακά και σε επαναληπτικά. Στα ακολουθιακά μοντέλα η ανάπτυξη γίνεται σε διαδοχικές διακριτές φάσεις και για ολόκληρο το σύστημα λογισμικού, ενώ στα

επαναληπτικά η ανάπτυξη του λογισμικού γίνεται σε τμήματα. Χαρακτηριστικότερο ακολουθιακό μοντέλο είναι αυτό του καταρράκτη, ενώ το γενικότερο από τα επαναληπτικά είναι το σπειροειδές. Πρακτικά, χρησιμότερα στην πράξη είναι τα μοντέλα κύκλου ζωής που αφήνουν ελευθερία εξειδίκευσης στις εκάστοτε συνθήκες και δεν προσδιορίζουν με αυστηρότητα τις ενέργειες που πρέπει να γίνουν, τα προϊόντα κ.λπ. Δεν υπάρχει ένα "κ*αλύτερο*" μοντέλο κύκλου ζωής αλλά ένα καταλληλότερο στις εκάστοτε συνθήκες τόσο του κατασκευαστή, όσο και του θεματικού πεδίου της εφαρμογής λογισμικού (Βεσκούκης, 2015).

Στον επόμενο Πίνακα φαίνονται συνοπτικά ορισμένα χαρακτηριστικά των μοντέλων κύκλου ζωής λογισμικού στα πιο σημαντικά ακολουθεί περιγραφή.

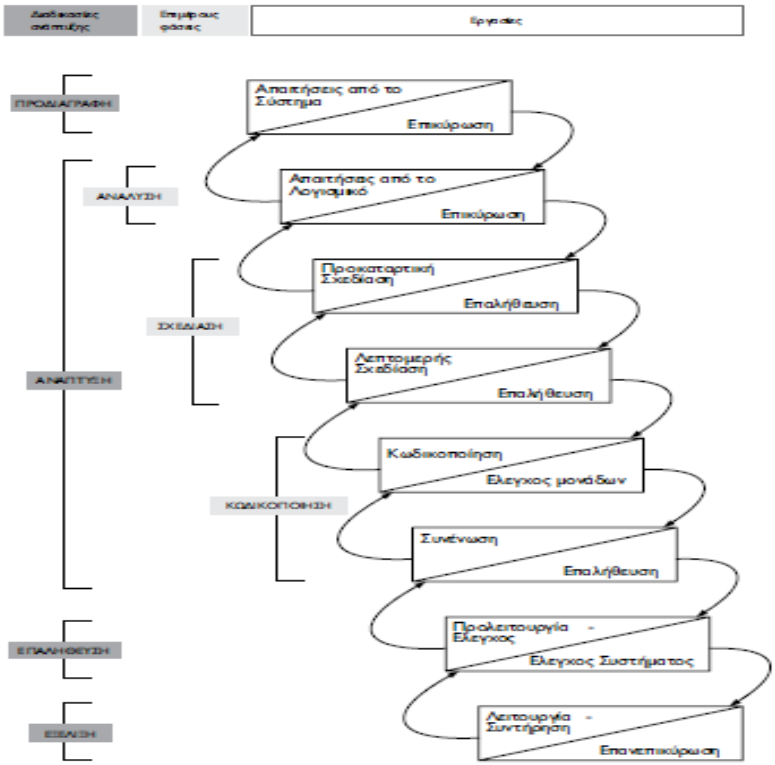
**Πίνακας 2.** Μοντέλα Κύκλου Ζωής Λογισμικού

| Μοντέλο | Μέγεθος εφαρμογών | Μεταβολές στις απαιτήσεις | Προσαρμοστικότητα στον κατασκευαστή | Διάδοση |
|---|---|---|---|---|
| Καταρράκτη | Μικρό έως μεσαίο | Ανεπιθύμητες | Καμία | Μεγάλη με τάση μείωσης |
| Πρωτοτυποποίησης | Μικρό ως μεσαίο | Δεκτές | Μικρή | Μικρή με τάση αύξησης |
| Λειτουργικής επαύξησης | Μεσαίο ως μεγάλο | Ανεπιθύμητες | Καμία | Μικρή με τάση μείωσης |
| Σπειροειδές | Μεσαίο ως μεγάλο | Δεκτές | Αρκετή | Μικρή με τάση μείωσης |
| Πίδακα | Οποιοδήποτε | Δεκτές | Αρκετή | Μικρή |
| Γενικό | Οποιοδήποτε | Δεκτές | Μεγάλη | Μικρή με ισχυρές τάσεις αύξησης |

Τα σημαντικότερα μοντέλα είναι τα εξής (Βεσκούκης, 2015; Boehm, 1984, 1988; Davis et al., 1988; Faitly, 1985; Williams, 1984):

- *μοντέλο του καταρράκτη* (*waterfall model*).Είναι το παλαιότερο μοντέλο κύκλου ζωής λογισμικού. Με το μοντέλο του καταρράκτη οι δραστηριότητες ανάπτυξης οργανώνονται σειριακά σε καλά ορισμένες φάσεις (Σχ.12). Η πρώτη φάση είναι ο προσδιορισμός και η ανάλυση των απαιτήσεων. Η δεύτερη φάση είναι η αρχιτεκτονική σχεδίαση, η οποία χρησιμοποιεί τις απαιτήσεις για να προσδιορίσει την αρχιτεκτονική του λογισμικού. Μετά την ολοκλήρωση της αρχιτεκτονικής σχεδίασης πραγματοποιείται η λεπτομερής σχεδίαση του λογισμικού, η οποία ακολουθείται από την κωδικοποίηση, τον

έλεγχο και την παράδοση του λογισμικού. Τέλος, έχουμε τη λειτουργία και συντήρηση του λογισμικού. Με το μοντέλο αυτό κάθε φάση παράγει ένα ενδιάμεσο προϊόν που χρησιμοποιείται στην επόμενη. Κάθε ενδιάμεσο προϊόν αξιολογείται από την ομάδα ανάπτυξης και τον πελάτη. Σε περίπτωση που κρίνεται ικανοποιητικό η ανάπτυξη προχωρά στην επόμενη φάση. Σφάλματα που εντοπίζονται σε κάθε φάση πολλές φορές οδηγούν στην αναθεώρηση του προϊόντος κάποιας προηγούμενης φάσης. Αν για παράδειγμα στη φάση της λεπτομερούς σχεδίασης η ομάδα ανακαλύψει ότι υπάρχουν σφάλματα και ελλείψεις στις απαιτήσεις, τότε επιστρέφει και διορθώνει τα ενδιάμεσα προϊόντα των απαιτήσεων και πιθανόν της αρχιτεκτονικής σχεδίασης. Διαπιστώνεται λοιπόν ότι, για να λειτουργήσει καλά το μοντέλο του καταρράκτη, προϋποθέτει το πάγωμα των ενδιάμεσων προϊόντων και τη δέσμευση όλων των μερών σε αυτά, κάτι που πολλές φορές είναι εκ των πραγμάτων ανέφικτο. Πολλές εταιρείες λογισμικού υιοθέτησαν (και πολλές ακόμη υιοθετούν) το μοντέλο του καταρράκτη για την παραγωγή προϊόντων λογισμικού.



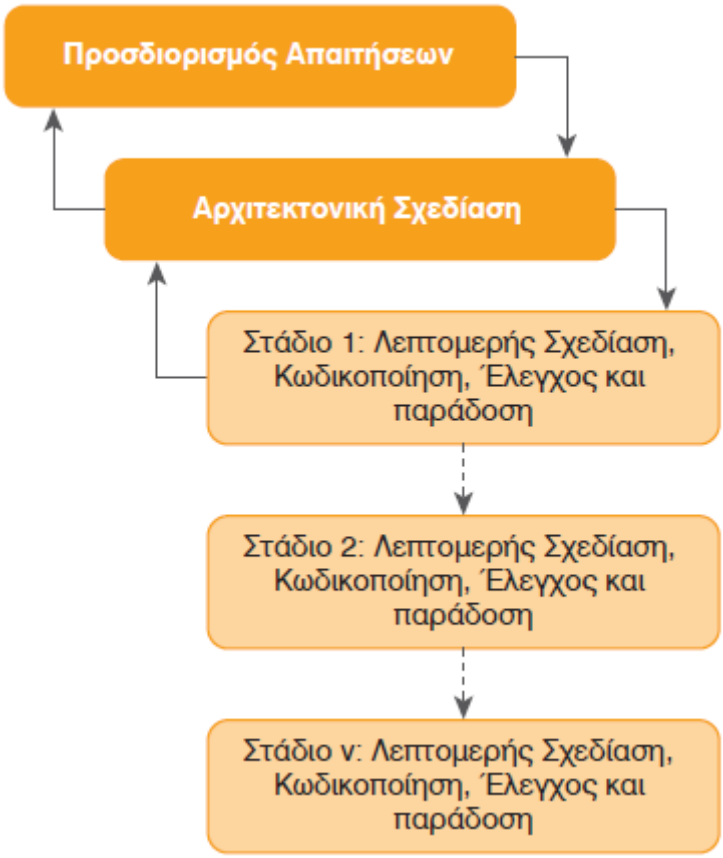

**Σχήμα 12.** Μοντέλο καταρράκτη

Συνοψίζοντας τα παραπάνω, το μοντέλο του καταρράκτη είναι η πρώτη προσπάθεια να προσεγγίσουμε μεθοδικά την ανάπτυξη του λογισμικού. Παρ'

όλες τις αδυναμίες του, που αναφέραμε παραπάνω, το μοντέλο του καταρράκτη μπορεί να εφαρμοστεί, όταν είμαστε βέβαιοι για τις απαιτήσεις και την αρχιτεκτονική του λογισμικού και όταν η συμμετοχή των χρηστών στο έργο είναι εκ των πραγμάτων περιορισμένη.

- *Επαυξητικό μοντέλο (incremental model*). Αυτό επιχειρεί να καλύψει κάποιες από τις αδυναμίες του μοντέλου του καταρράκτη. Με το επαυξητικό μοντέλο το λογισμικό παραδίδεται σταδιακά (Σχ.13). Νέες εκδόσεις που λέγονται *προσαυξήσεις (increments*) εμπλουτίζουν τη λειτουργικότητα του λογισμικού. Επιδιώκεται η σταδιακή κάλυψη της συνολικής λειτουργικότητας για να ελαχιστοποιηθεί το ενδεχόμενο το τελικό προϊόν να απέχει από τις επιδιώξεις του πελάτη. Με το επαυξητικό μοντέλο οι δραστηριότητες του προσδιορισμού απαιτήσεων και της αρχιτεκτονικής σχεδίασης γίνονται όμοια με το μοντέλο του καταρράκτη. Επιχειρείται η πλήρης και λεπτομερής καταγραφή των απαιτήσεων και ο προσδιορισμός της αρχιτεκτονικής. Οι υπόλοιπες δραστηριότητες πραγματοποιούνται σε στάδια, το αποτέλεσμα των οποίων είναι μία προσαύξηση του λογισμικού. Στην πρώτη προσαύξηση προτεραιοποιούνται οι απαιτήσεις και καθορίζεται η εμβέλεια της έκδοσης του πρώτου σταδίου. Ακολουθεί η λεπτομερής σχεδίαση, η κωδικοποίηση, ο έλεγχος και η παράδοση του λογισμικού. Επόμενα στάδια συμπληρώνουν τη λειτουργικότητα του λογισμικού βάσει των απαιτήσεων που δεν έχουν καλυφθεί. Η βασική αρχή οργάνωσης των σταδίων είναι ότι οι πρώτες εκδόσεις παρέχουν την πιο σημαντική λειτουργικότητα του λογισμικού, ενώ μεταγενέστερες εκδόσεις του λογισμικού στοχεύουν στο να συγκλίνει η λειτουργικότητα σε αυτό που επιθυμεί ο πελάτης. Το επαυξητικό μοντέλο έχει τις ίδιες αδυναμίες με το μοντέλο του καταρράκτη σε ό,τι αφορά στις απαιτήσεις και στην αρχιτεκτονική του λογισμικού. Θα πρέπει οι απαιτήσεις και η αρχιτεκτονική να είναι σχετικά σταθερές και να μην υπόκεινται σε συνεχείς αλλαγές. Σε αυτό που πλεονεκτεί, είναι η διαχείριση των κινδύνων και η αξιολόγηση της προόδου του έργου. Συμπερασματικά, το επαυξητικό μοντέλο μπορεί να χρησιμοποιηθεί, όταν έχουμε σχετικά σταθερές απαιτήσεις και αρχιτεκτονική. Μπορεί να θεραπεύσει κάποιες από τις αδυναμίες του καταρράκτη, όπως η διαχείριση των κινδύνων, η αξιολόγηση της προόδου του έργου και η συμμετοχή των χρηστών. Μπορεί να χρησιμοποιηθεί στην ανάπτυξη εμπορικού-τυποποιημένου λογισμικού, όπου αξιολογείται συνεχώς

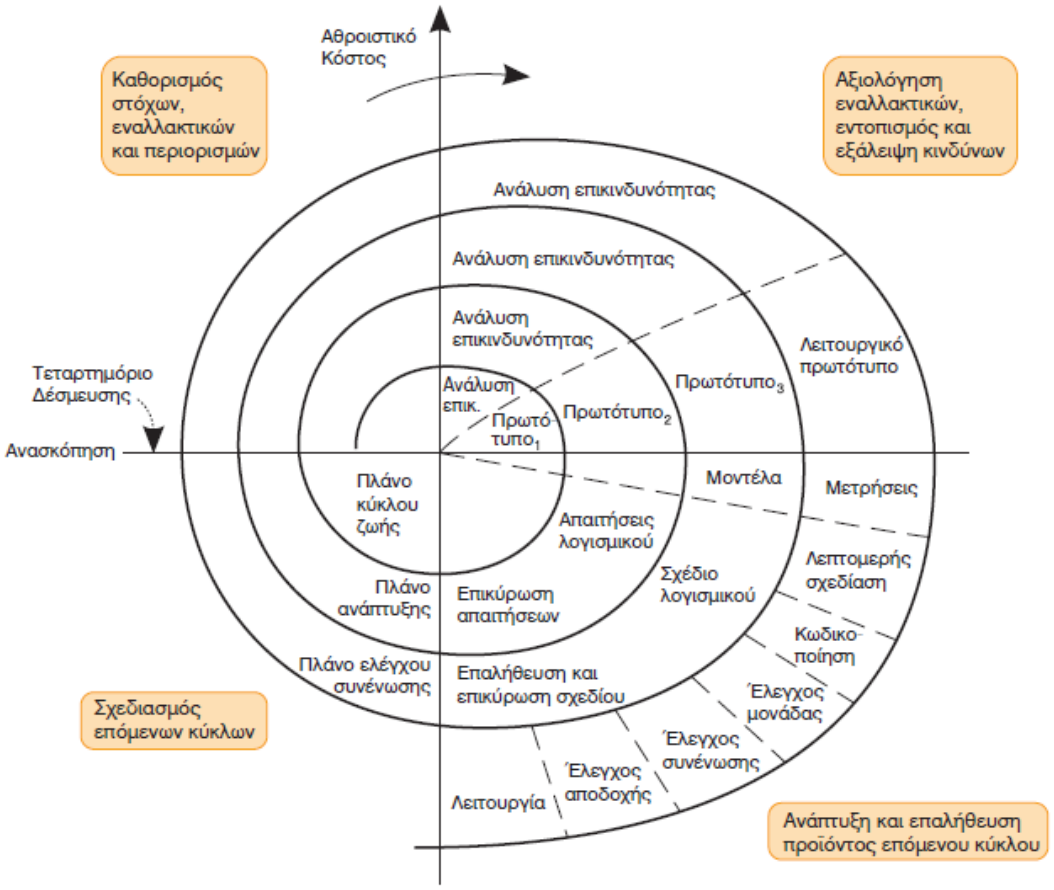
η αποδοχή του προϊόντος και περιορίζεται ο κίνδυνος της απόρριψης του προϊόντος από τους χρήστες.




**Σχήμα 13.** Το επαυξητικό μοντέλο

- *Σπειροειδές μοντέλο (spiral model*). Αυτό το μοντέλο εισάγει την ιδέα της μη γραμμικής διαδοχής των δραστηριοτήτων της ανάπτυξης. Η ανάπτυξη του λογισμικού πραγματοποιείται σε κύκλους, κατά τους οποίους μπορεί να πραγματοποιούνται ταυτόχρονα πολλές δραστηριότητες ανάπτυξης. Το τι θα πραγματοποιηθεί σε κάποιο κύκλο του σπειροειδούς μοντέλου καθορίζεται από τους κινδύνους που αντιμετωπίζει το έργο σε κάθε φάση του. Αν, για παράδειγμα, οι ανάγκες του πελάτη μας είναι εντελώς άγνωστες, τότε δίνουμε έμφαση στην αρχική καταγραφή των απαιτήσεων. Αν υπάρχουν ιδιαίτερες απαιτήσεις απόδοσης σε κάποιες λειτουργίες του λογισμικού, τότε μπορεί να αναπτυχθεί ένα *πρωτότυπο (prototype*), για να εξεταστεί το κατά πόσο μπορούν να ικανοποιηθούν τέτοιου είδους απαιτήσεις. Κάθε κύκλος στο σπειροειδές μοντέλο επιχειρεί να προσδιορίσει όσο το δυνατό καλύτερα το τελικό προϊόν που περιμένει ο πελάτης, μειώνοντας ταυτόχρονα τους κινδύνους του έργου. Η επιλογή της φάσης που θα ακολουθήσει βασίζεται στο

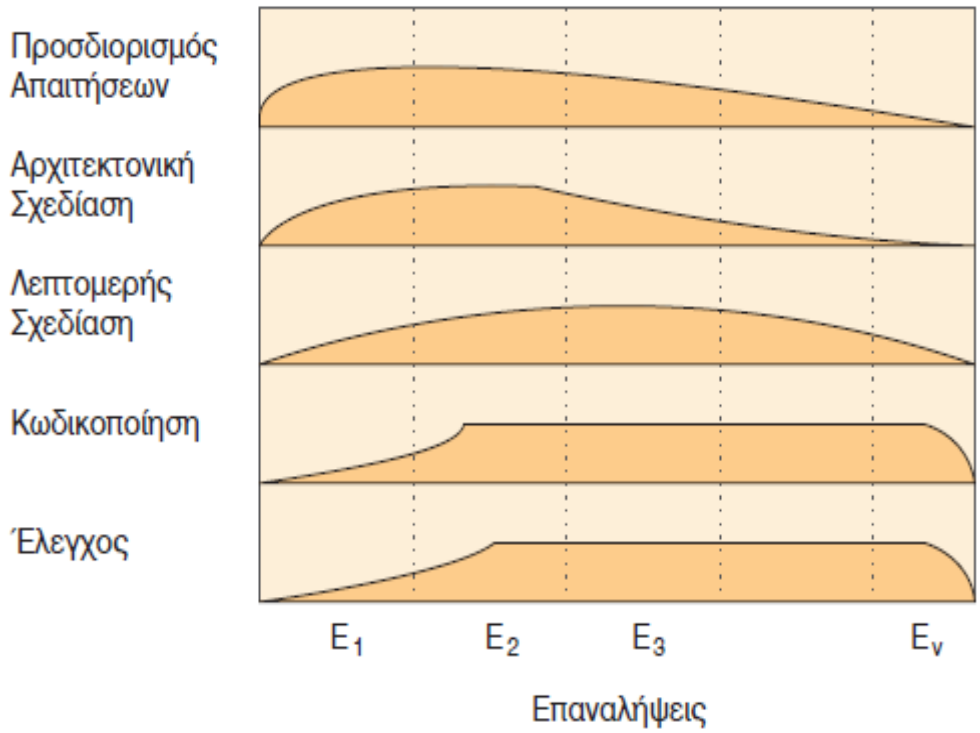
βαθμό ρίσκου που ενυπάρχει στην πρόοδο των εργασιών. Στο επόμενο σχήμα παρουσιάζεται η οργάνωση των παραπάνω δραστηριοτήτων σε κύκλους ανάπτυξης, ορίζοντας τέσσερα βασικά βήματα που εκτελούνται σε κάθε κύκλο. Τα βήματα αυτά εμφανίζονται στα τέσσερα τεταρτημόρια του σχήματος. Η πολυπλοκότητα που εισάγει το σπειροειδές μοντέλο στη διοίκηση του έργου αντισταθμίζεται από τη συνεχή μείωση των κίνδυνων. Επομένως, το σπειροειδές μοντέλο είναι κατάλληλο για την ανάπτυξη μεγάλων και πολύπλοκων συστημάτων, για τα οποία δεν υπάρχει αρκετή τεχνογνωσία και υπάρχουν πολλές εστίες σοβαρών κινδύνων.



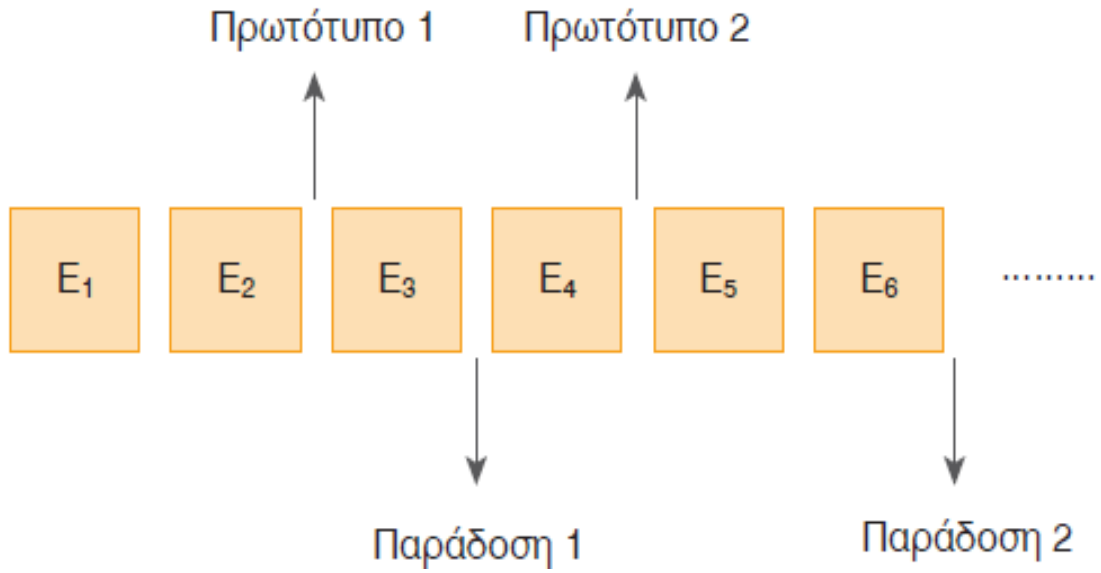

**Σχήμα 14.** Το σπειροειδές μοντέλο

- *Επαναληπτικό μοντέλο (incremental model).* Το μοντέλο αυτό όπως και το σπειροειδές απορρίπτει την ακολουθιακή οργάνωση των δραστηριοτήτων ανάπτυξης. Αντί η ανάπτυξη του λογισμικού να οργανώνεται σε σειριακές φάσεις, οργανώνεται σε *επαναλήψεις (iterations*), όπου σε κάθε επανάληψη εκτελούνται όλες οι δραστηριότητες της ανάπτυξης, ξεκινώντας από τον προσδιορισμό απαιτήσεων και καταλήγοντας στην κωδικοποίηση και στον

έλεγχο.Κάθε επανάληψη μπορεί να θεωρηθεί ως μία μικρογραφία ενός έργου ανάπτυξης που οδηγεί στην παραγωγή ημιτελών εκτελέσιμων προγραμμάτων, τα οποία όμως έχουν ελεγχθεί με επιτυχία. Το επαναληπτικό μοντέλο είναι εξ ορισμού και *επαυξητικό (incremental*), επειδή η λειτουργικότητα προστίθεται σταδιακά στο λογισμικό. Όπως φαίνεται στο επόμενο σχήμα, σε κάθε επανάληψη πραγματοποιούνται όλες οι δραστηριότητες αλλά με διαφορετική βαρύτητα. Στις πρώτες επαναλήψεις εστιάζουμε την προσοχή μας στον προσδιορισμό των απαιτήσεων και στην αρχιτεκτονική σχεδίαση. Όσο προχωρά η ανάπτυξη, το βάρος μετατοπίζεται στη λεπτομερή σχεδίαση, στην κωδικοποίηση και τον έλεγχο. Εκτός από τη λογική της εξελικτικής πρωτοτυποποίσης και της σταδιακής παράδοσης, το επαναληπτικό μοντέλο μπορεί να υποστηρίξει και μία τρίτη ενδιαφέρουσα παραλλαγή, αυτή της *εξελικτικής παράδοσης (evolutionary delivery*) (Σχ.16). Η λεπτή διαφορά της εξελικτικής παράδοσης και της σταδιακής παράδοσης του επαυξητικού μοντέλου είναι ότι η εξελικτική παράδοση καθοδηγείται περισσότερο από το σχολιασμό των χρηστών, ενώ στη σταδιακή παράδοση κάθε έκδοση είναι αποτέλεσμα του σχεδιασμού που προκύπτει από το λεπτομερή προσδιορισμό των απαιτήσεων. Στην πράξη βέβαια και στις δύο παραλλαγές η λειτουργικότητα που προστίθεται είναι αποτέλεσμα συνδυασμού του σχεδιασμού και της ανατροφοδότησης των χρηστών. Η διαφορά είναι περισσότερο στην έμφαση. Η μεγαλύτερη παγίδα στη διοίκηση ενός έργου είναι να αγνοείται το βασικό σκεπτικό του επαναληπτικού μοντέλου και η οργάνωση του έργου να "*εκφυλίζεται*" στο μοντέλο του καταρράκτη. Για παράδειγμα, ας υποθέσουμε ότι η ομάδα ανάπτυξης επιλέγει να μην ελέγξει τις ενδιάμεσες εκδόσεις του λογισμικού αλλά να αναβάλει τον έλεγχο προς το τέλος του έργου. Η προσέγγιση αυτή αποκλίνει σημαντικά από το σκεπτικό του επαναληπτικού μοντέλου, το οποίο απαιτεί τη συνεχή εκτέλεση των ελέγχων για τη διαρκή αξιολόγηση της ποιότητας του λογισμικού.



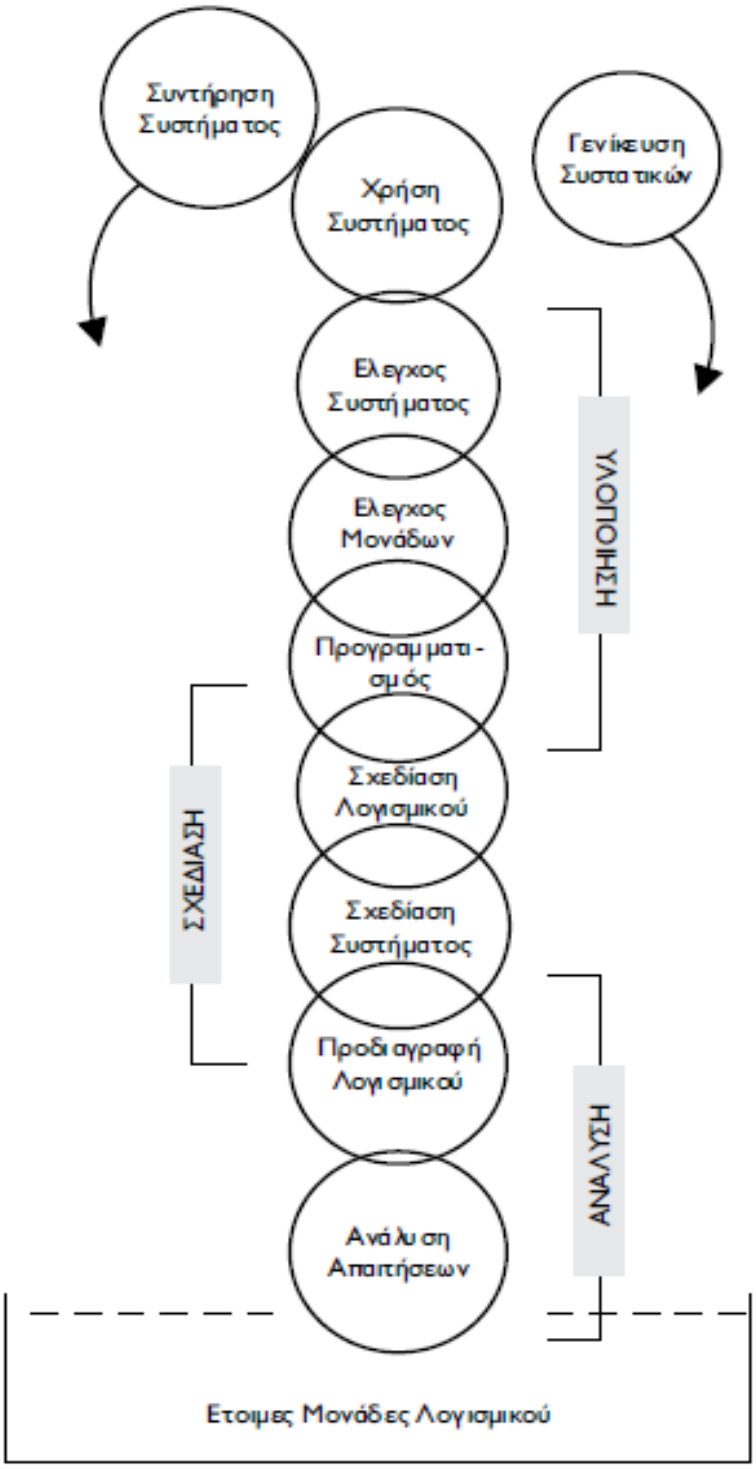

**Σχήμα 15.** Το επαναληπτικό μοντέλο




**Σχήμα 16.** Εξελικτική πρωτοτυποποίηση και παράδοση στο επαναληπτικό μοντέλο

- *Μοντέλο του Πίδακα*. Η ιδέα αυτού του μοντέλου τονίζει περισσότερο τα επιθυμητά χαρακτηριστικά της μεθοδολογίας κατασκευής του λογισμικού σύμφωνα με την αντικειμενοστρεφή λογική, ήταν δε αρκετά επίκαιρη κατά την έκρηξη ενδιαφέροντος για την αντικειμενοστρεφή τεχνολογία στα τέλη της δεκαετίας του ’80 και στις αρχές της δεκαετίας του’90. Κατά την ανάπτυξη παρατηρούνται επικαλύψεις των φάσεων "*ανάλυση*", "*σχεδίαση*",

"*κωδικοποίηση*", οι οποίες φαίνονται με την επικάλυψη των κύκλων στο επόμενο σχήμα. Κατά το τέλος της ανάπτυξης ορισμένα από τα συστατικά του λογισμικού που έχουν παραχθεί ενσωματώνονται σε μια "*δεξαμενή*" συστατικών και διατίθενται για να χρησιμοποιηθούν στην ανάπτυξη και νέων συστημάτων




**Σχήμα 17.** Το μοντέλο κύκλου ζωής του πίδακα

# ΚΕΦΑΛΑΙΟ ΔΕΥΤΕΡΟ

# ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PYTHON

## 2.1 Γενικά

Το 1991 ο Guido van Rossman δημιούργησε την Python ως μια ανοιχτού λογισμικού ανεξάρτητη πλατφόρμας γενικού σκοπού γλώσσα προγραμματισμού (open-source platformindependent). Είναι βασικά μια πολύ απλή γλώσσα που περιστοιχίζεται από μια τεράστια βιβλιοθήκη προσθηκών (add-on modules), οι οποίες περικλείουν πλήρη πρόσβαση στα υποκείμενα λειτουργικά συστήματα. Αυτό σημαίνει ότι μπορεί να διαχειριστεί και επεξεργαστεί προγράμματα από άλλα πλήρη ακόμη και μεταγλωττισμένα πακέτα, είναι δηλαδή μια *scripting* γλώσσα. Αυτή η ευκολία επέφερε την υιοθέτησή της από ισχυρούς χρήστες, όπως η Google και η NASA, και μια στρατιά από κατασκευαστές λογισμικού (Μαγκούτης και Νικολάου, 2015; Κάππος και Μπούρας, 2017).

Οι δυο κυριότερες εκδόσεις της είναι η 2.x και η 3.x. Η έκδοση 2.x παρουσιάστηκε στις 16 Οκτωβρίου 2000 και πολύ γρήγορα ξεκίνησε να χρησιμοποιείται σε πολλούς τομείς και να αναπτύσσεται μια μεγάλη ποικιλία εφαρμογών σε Python. Η έκδοση αυτή, που δεν είναι συμβατή με την τελευταία έκδοση 3.x, χρησιμοποιείται ακόμη για την ανάπτυξη εφαρμογών. Η έκδοση 3.x παρουσιάστηκε στις 3 Δεκεμβρίου 2008, με το όνομα Python 3000 ή Py3K. Τη στιγμή που γραφόταν το βιβλίο η τελευταία έκδοση της γλώσσας ήταν η 3.7.3. Οι δοκιμές του κώδικα που περιέχει το βιβλίο έγιναν με διάφορες εκδόσεις της Python (μέχρι και την έκδοση 3.7). Το όνομα της γλώσσας δεν είναι σχετικό με το γνωστό ερπετό, παρόλο που το λογότυπο που χρησιμοποιείται από το *Ίδρυμα της Python (Python Software Foundation)*, που συντηρεί την κύρια έκδοση της γλώσσας, παραπέμπει σε αυτό. Το όνομά της προέρχεται από την ομάδα κωμικών της δεκαετίας του 1970 με το όνομα Monty Python (*Flying Circus*), της βρετανικής τηλεόρασης (BBC)(Μαγκούτης και Νικολάου, 2015; Κάππος και Μπούρας, 2017).

## 2.2 Βασικά Στοιχεία PYTHON

Η γλώσσα προγραμματισμού Python, παρόλο που χρησιμοποιείται εδώ και δυο δεκαετίες περίπου, έχει αποκτήσει, μόλις τα τελευταία χρόνια στην Ελλάδα, τεράστια απήχηση και αποδοχή, όχι μόνο στον χώρο της *επιστήμης των υπολογιστών (Computer Science)*, αλλά και σε άλλους εμπορικούς και επιστημονικούς τομείς. Χαρακτηριστική είναι η ταχύτατη διείσδυση της γλώσσας τόσο σε πρακτικές εφαρμογές (σε επιχειρήσεις και οργανισμούς), όσο και στον τομέα της διδασκαλίας και εκμάθησης προγραμματισμού σε διάφορα επίπεδα εκπαίδευσης (από τη δευτεροβάθμια μέχρι την τριτοβάθμια εκπαίδευση).

Δεν είναι τυχαίο το ότι χρησιμοποιείται σε πάρα πολλούς και ασυνήθιστα, θα λέγαμε, διαφορετικούς τομείς δραστηριοτήτων και αυτό είναι κάτι που την κάνει μοναδική στο είδος της. Η ιδιαίτερη ευκολία εκμάθησης είναι ένα από τα χαρακτηριστικά της. Για αυτό τον λόγο πλέον αποτελεί, την πρώτη γλώσσα προγραμματισμού για εκμάθηση.Υπάρχουν όμως και άλλοι ακόμα πολλοί λόγοι που θα οδηγήσουν κάποιον να στραφεί στην εκμάθηση της γλώσσας και να επιλέξει την Python ως τη γλώσσα προγραμματισμού που θα χρησιμοποιεί για τη δημιουργία προγραμμάτων. Στη συνέχεια παρατίθενται οι κυριότεροι λόγοι από αυτούς(Μαγκούτης και Νικολάου, 2015; Κάππος και Μπούρας, 2017):

- *Δυναμική (Dynamic)*. Κάθε *δομή δεδομένων (data structure)* (Σχ.18) που υποστηρίζει μπορεί να δημιουργηθεί και να αλλάξει δυναμικά κατά την εκτέλεση του προγράμματος. Δεν χρειάζεται να είναι προκαθορισμένη, κατασκευάζεται και τροποποιείται πλήρως σύμφωνα με τις ανάγκες του προγράμματος, την ώρα που αυτό τρέχει. Τα δεδομένα που υπάρχουν στη μνήμη του υπολογιστή υφίστανται δυναμική διαχείριση από την ίδια τη γλώσσα.
- *Δωρεάν*. Είναι δωρεάν για κάθε χρήση (προσωπική και εμπορική) και διανέμεται ελεύθερα. Είναι *ανοιχτού κώδικα (open-source)*.
- *Υψηλού επιπέδου (High-level)*. Αυτό σημαίνει ότι παρέχει απλούς και εύκολα κατανοητούς κανόνες (συντακτικούς και γραμματικούς) για τη σύνταξη προγραμμάτων που είναι πολύ κοντά στην ανθρώπινη γλώσσα.
- *Διερμηνευόμενη (Interpreted)*. Η εκτέλεση ενός προγράμματος γραμμένου στη γλώσσα Python γίνεται με τη βοήθεια ενός ειδικού προγράμματος, του

*διερμηνευτή (interpreter)*, που αναλαμβάνει να ελέγξει την ορθότητά του και να πραγματοποιήσει την εκτέλεσή του.

Σε κάθε πρόγραμμα, οποιασδήποτε γλώσσας προγραμματισμού (από την Assembly και τη C, μέχρι τη C#, την Python ή την Java), πρέπει να γραφτεί ο κώδικας – οι εντολές ενός προγράμματος, σε κάποιο (συνήθως απλό, χωρίς μορφοποιήσεις) αρχείο κειμένου. Αυτό το αρχείο ονομάζεται αρχείο *πηγαίου κώδικα (source code)* και περιλαμβάνει όλες τις οδηγίες προς τον υπολογιστή για την επίλυση ενός προβλήματος. Για να μπορέσει όμως να εκτελέσει ο υπολογιστής αυτές τις εντολές, θα πρέπει να μετατραπούν, να μεταφραστούν δηλαδή, σε γλώσσα κατάλληλη για τον υπολογιστή, σε *γλώσσα μηχανής (machine language, code)*. Αυτή τη μετάφραση αναλαμβάνει να την κάνει ένα ειδικό πρόγραμμα που καλείται *μεταγλωττιστής (compiler)* ή *διερμηνευτής (interpreter)*.

Κάθε γλώσσα προγραμματισμού συνοδεύεται από ένα τέτοιο πρόγραμμα, διαθέσιμο είτε μέσα από κάποιο Ολοκληρωμένο Περιβάλλον Ανάπτυξης Λογισμικού (IDE, Integrated Development Environment), είτε μέσα από τη *γραμμή εντολών (command line)* του λειτουργικού συστήματος (Windows, Linux, MacOSx). Ο μεταγλωττιστής αναλαμβάνει να ελέγξει όλες τις εντολές του πηγαίου κώδικα για την ύπαρξη *συντακτικών σφαλμάτων (syntax errors)*. Αν εντοπίσει λάθη, θα σταματήσει και θα εμφανίσει, συνήθως κωδικοποιημένα, μια περιγραφή του σφάλματος και τη γραμμή όπου εντοπίστηκε αυτό. Αν το πρόγραμμα δεν έχει λάθη, η διαδικασία της μεταγλώττισης θα ολοκληρωθεί. Το αποτέλεσμα της μεταγλώττισης είναι η δημιουργία ενός αρχείου με *δυαδικό κώδικα (binary code), εκτελέσιμου (executable)* από τον υπολογιστή.

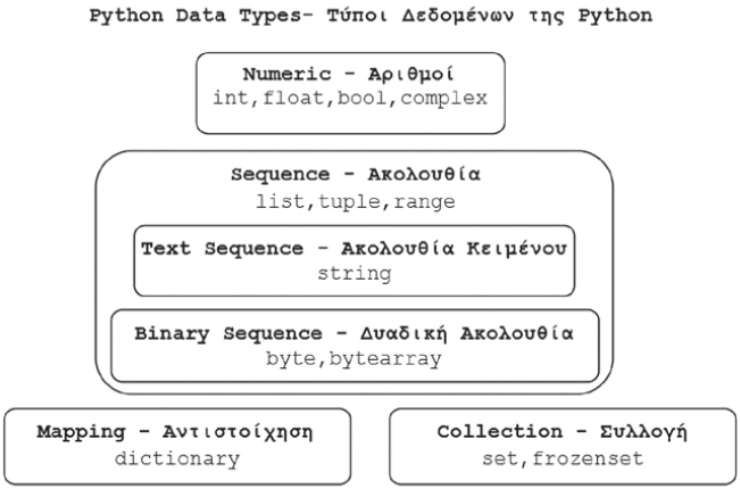
Σε αρκετές περιπτώσεις η διαδικασία αυτή περιλαμβάνει και τη *σύνδεση (link)* του πηγαίου κώδικα και τη *φόρτωση (load)* άλλων προγραμμάτων και βιβλιοθηκών της γλώσσας, που είναι απαραίτητα για τη λειτουργία του προγράμματος. Ο διερμηνευτής, από την άλλη, λειτουργεί αρκετά διαφορετικά. Τις περισσότερες φορές αναλαμβάνει να ελέγξει μια εντολή, να τη μεταφράσει σε κώδικα μηχανής και να την εκτελέσει. Αμέσως μετά θα πάει στην επόμενη εντολή για να επαναλάβει την ίδια διαδικασία. Αν εντοπίσει κάποιο σφάλμα σε μια από τις εντολές του προγράμματος, θα σταματήσει εκεί, εμφανίζοντας ένα κατάλληλο μήνυμα.

Η εκτέλεση ενός προγράμματος γραμμένου σε Python γίνεται από τον διερμηνευτή της γλώσσας. Κατά την εκτέλεση του προγράμματος δημιουργείται, στον ίδιο φάκελο όπου βρίσκεται το αρχείο πηγαίου κώδικα, ένα αρχείο με *δυαδικό κώδικα (byte code)*, με την επέκταση .pyc στο όνομα του αρχείου. Αυτό το αρχείο χρησιμοποιεί ο διερμηνευτής για την εκτέλεση του προγράμματος.

- *Αντικειμενοστραφής (Object-oriented)*. Η Python είναι μια γλώσσα προσανατολισμένη στα *αντικείμενα (objects)*. Έχει τα χαρακτηριστικά μιας αντικειμενοστραφούς γλώσσας, ωστόσο, υπάρχουν αρκετές διαφωνίες9 για το αν υλοποιεί πλήρως αυτό το μοντέλο προγραμματισμού. Αυτό οφείλεται στο γεγονός ότι εφαρμόζει μια χαλαρή πολιτική σε ένα από τα χαρακτηριστικά του συγκεκριμένου μοντέλου, το χαρακτηριστικό της *ενθυλάκωσης (encapsulation)*. Παρέχει έναν υποτυπώδη και έμμεσο τρόπο προστασίας της ιδιωτικότητας των αντικειμένων.
- *Γενικού σκοπού (General-purpose)*. Αυτό σημαίνει ότι μπορεί να χρησιμοποιηθεί σχεδόν για κάθε πρόβλημα και να καλύψει τις απαιτήσεις και τις ανάγκες για μεγάλη ποικιλία εφαρμογών. Αυτός άλλωστε είναι και ένας από τους βασικούς λόγους που χρησιμοποιείται σε τόσο πολλές και διαφορετικές περιπτώσεις.
- *Διαθέσιμη για διάφορα συστήματα*. Ένα πρόγραμμα σε Python, που έχει γραφτεί σε περιβάλλον Windows, μπορεί, τις περισσότερες φορές, να χρησιμοποιηθεί χωρίς αλλαγές και σε άλλα λειτουργικά συστήματα. Εξαιρούνται ορισμένες περιπτώσεις όπου κάποιες βιβλιοθήκες κώδικα λειτουργούν με διαφορετικό τρόπο από το ένα σύστημα στο άλλο.
- *Κατάλληλη για πολλαπλά προγραμματιστικά υποδείγματα (Multi-Paradigm)*. Ενσωματώνει διαφορετικές λειτουργίες και δίνει απόλυτη ελευθερία στον δημιουργό ενός προγράμματος για το ποια χαρακτηριστικά θα χρησιμοποιήσει κάθε φορά.
- *Ικανή να ενσωματώνει υπηρεσίες (Frameworks)*. Αρκετά διαδεδομένη είναι η χρήση έτοιμων συστημάτων υπηρεσιών, που υλοποιούνται σε Python. Κυρίως αφορούν την εγκατάσταση και λειτουργία διαδικτυακών υπηρεσιών, όπως *διακομιστές ιστοσελίδων (web servers)*, *υπηρεσίες νέφους (cloud services)* κ.λπ. Τα πιο γνωστά από αυτά είναι το Flask και το Django.

- *Υλοποιήσιμη σε πολλές εκδοχές*. Υπάρχουν αρκετές διανομές της Python, με διαφορετικές *υλοποιήσεις (implementations)*. Η επίσημη έκδοση είναι γνωστή με το όνομα CPython, αφού χρησιμοποιεί τη γλώσσα C σε διάφορες λειτουργίες και σε ενσωματωμένες βιβλιοθήκες της. Εκτός όμως από αυτήν υπάρχουν και άλλες εκδόσεις της γλώσσας, η καθεμία με τα δικά της χαρακτηριστικά και ιδιότητες, που κυρίως αφορούν τη συνεργασία και την επικοινωνία της με άλλα συστήματα γλωσσών προγραμματισμού. Οι πιο γνωστές από αυτές είναι οι ακόλουθες:
    - *Cython*. Είναι ένας μεταγλωττιστής τόσο για κώδικα γραμμένο σε απλή Python, όσο και σε ένα υπερσύνολο της γλώσσας που ονομάζεται και Cython. Χρησιμοποιείται για τη συγγραφή *επεκτάσεων κώδικα (extensions of code)* σε C και τη σύνδεσή τους με προγράμματα σε Python. Είναι δυνατή η δημιουργία εφαρμογών για την κλήση προγραμμάτων σε C/C++ ή την ενσωμάτωσή τους σε αυτά.
    - *Jython.* Κάνει δυναμική μεταγλώττιση σε *δυαδικό κώδικα (byte code)* της Java και μπορεί να *επεκτείνει (extend)* τις κλάσεις της.
    - *Iron Python*. Είναι μια υλοποίηση της γλώσσας στενά συνδεδεμένη με το Net Framework, το οποίο μπορεί και χρησιμοποιεί. Άλλες γλώσσες του Net Framework μπορούν να χρησιμοποιούν προγράμματα φτιαγμένα σε Iron Python.
    - *Brython.* Είναι έκδοση της γλώσσας που σκοπό έχει να αντικαταστήσει την Javascript στην κατασκευή *κώδικα που εκτελείται σε ιστοσελίδες (web programming)*. Κάνει χρήση το περιβάλλον της HTML 5.
    - *Pyjs*. Κάνει εύκολη τη δημιουργία εφαρμογών για ιστοσελίδες γραμμένες εξ ολοκλήρου σε Python.
- *Επεκτάσιμη (Extendable)*. Μπορεί εύκολα να συνεργαστεί με άλλες γλώσσες προγραμματισμού. Σε αυτή τη λειτουργία βοηθούν και οι διάφορες υλοποιήσεις της Python.

Η Python χρησιμοποιείται μαζί με άλλες γλώσσες προγραμματισμού από επιχειρήσεις και οργανισμούς όπως οι Dropbox, Facebook, Spotify, Netflix, Quora, Reddit, IBM, Nokia, Disney, Yahoo!, NASA, JP Morgan, Instagram, PayPal, eBay κλπ.




**Σχήμα 18.** Τύποι Δεδομένων στην Python

## 2.3 Προγραμματιστικές Τεχνικές

Η έννοια του *προγραμματιστικού υποδείγματος (programming paradigm)* συνδέεται με τις τεχνικές, τις λειτουργίες και τους μηχανισμούς που χρησιμοποιεί μια γλώσσα προγραμματισμού. Η γλώσσα Python διαθέτει χαρακτηριστικά και λειτουργίες *πολλαπλών προγραμματιστικών υποδειγμάτων (multi-programming paradigms)*. Ο τρόπος χρήσης της και το μοντέλο που θα χρησιμοποιηθεί σε κάθε πρόγραμμα εξαρτάται κυρίως από τον χρήστη και τον σχεδιασμό που αυτός θα κάνει σε κάθε περίπτωση, καθώς και ανάλογα με το πρόβλημα που πρέπει να αντιμετωπίσει. Σε κάθε υλοποίηση ενός προγράμματος, υπάρχουν τα κατάλληλα εργαλεία, με τις αντίστοιχες τεχνικές, που μπορεί κάποιος να επιλέξει, ώστε να επιτύχει με τον καλύτερο τρόπο τον στόχο του.

Οι προγραμματιστικές τεχνικές που μπορεί να ακολουθεί ένα πρόγραμμα σε Python είναι οι εξής (Μαγκούτης και Νικολάου, 2015; Κάππος και Μπούρας, 2017):

- *Διαδικαστικός προγραμματισμός (Procedural Programming)*. Η υλοποίηση περιλαμβάνει την ιεραρχική σχεδίαση του προγράμματος και τη δημιουργία μιας σειράς από αυτόνομα τμήματα κώδικα, με τη μορφή υποπρογραμμάτων. Για αυτό τον λόγο, συνήθως ακολουθείται ο σχεδιασμός *από πάνω προς τα κάτω (top to bottom)*, από το γενικό προς το ειδικό. Ένα πρόβλημα, επομένως, σπάει, ή χωρίζεται, σε ανεξάρτητα τμήματα, τα οποία στη συνέχεια θα αποτελέσουν αντίστοιχες ενότητες υποπρογραμμάτων, με τη μορφή συναρτήσεων ή βιβλιοθηκών κώδικα. Τα χαρακτηριστικά του *Δομημένου Προγραμματισμού (Structured Programming)* είναι διαθέσιμα και

χρησιμοποιούνται για αυτό τον σκοπό. Ειδικότερα είναι: οι δομές της ακολουθίας, της επιλογής και της επανάληψης.

- *Προστακτικός προγραμματισμός (Imperative Programming)*. Δίνεται έμφαση στο πώς θα λυθεί ένα πρόβλημα. Το πρόγραμμα δημιουργείται με σκοπό να αλλάζει την *κατάσταση (state)* σε στοιχεία του υπολογιστή (μνήμη, επεξεργαστή κ.λπ.), ώστε τελικά να μπορέσει να λύσει το δοσμένο πρόβλημα.
- *Αντικειμενοστραφής προγραμματισμός (Object-oriented Programming).* Βασικό συστατικό αυτού του μοντέλου είναι οι κλάσεις αντικειμένων. Κάθε οντότητα μέσα σε ένα πρόγραμμα παίρνει τη μορφή ενός *αντικειμένου (object)*, που ενσωματώνει μέσα του κώδικα και δεδομένα. Έτσι, κάθε αντικείμενο διαθέτει όχι μόνο γνωρίσματα και ιδιότητες, αλλά και κώδικα που προσδιορίζει τη συμπεριφορά του και χειρίζεται τα δεδομένα του.
- *Συναρτησιακός προγραμματισμός (Functional Programming)*. Αν και η Python δεν ακολουθεί πιστά αυτό το μοντέλο (όπως άλλες γλώσσες σαν τη γλώσσα Haskell), ωστόσο διαθέτει αρκετά από τα χαρακτηριστικά του. Ένα από αυτά είναι η δυνατότητα μιας εντολής να παίρνει τη μορφή μαθηματικής συνάρτησης, όπου το αποτέλεσμα παράγεται μέσα από δηλώσεις και εκφράσεις, και όχι από εντολές ή οδηγίες.

## 2.4 Εγκατάσταση

Η εγκατάσταση της Python σχετίζεται με το λειτουργικό σύστημα που έχει ο υπολογιστής. Ανάλογα με το *λειτουργικό σύστημα (Operating System)* που διαθέτει, ακολουθείται η διαδικασία ένταξης της γλώσσας στον κάθε υπολογιστή. Για την Python, η *εγκατάσταση (installation – setup)* της πραγματοποιείται κυρίως σε συστήματα Windows και Linux.

Για παράδειγμα, στα *Windows* η διαδικασία αφορά τα εξής (Σχ.19,20)(Μαγκούτης και Νικολάου, 2015; Κάππος και Μπούρας, 2017):

- Τα Windows δεν έχουν εγκατεστημένη τη γλώσσα Python.
- Να εγκαταστήσει το περιβάλλον διερμηνευτή της επίσημης έκδοσης της γλώσσας. Αυτό του δίνει τη δυνατότητα να έχει πρόσβαση στην κονσόλα του διερμηνευτή (με το σήμα >>>) αλλά και στο (πολύ απλό, που δεν απαιτεί πολλούς υπολογιστικούς πόρους) IDE που διαθέτει. Μπορεί να εγκατασταθεί παράλληλα και κάποιο πιο σύνθετο και με περισσότερες δυνατότητες IDE,

όπως το Pycharm, που δίνει τη δυνατότητα για εγκατάσταση πακέτων βιβλιοθηκών και τη βηματική *εκσφαλμάτωση (debugging)* των προγραμμάτων.

- Να εγκαταστήσει ένα ολοκληρωμένο περιβάλλον (IDE), όπως το Spyder, μέσα από το σύστημα Anaconda ή το Canopy Enthought. Αν γίνει αυτό, δεν χρειάζεται να εγκατασταθεί και ο διερμηνευτής της Python. Επίσης, και τα δυο ενσωματώνουν έναν πολύ μεγάλο αριθμό *προεγκατεστημένων (preinstalled)* βιβλιοθηκών και διαθέτουν δικό τους σύστημα αναζήτησης και διαχείρισης βιβλιοθηκών.


**Σχήμα 19.** Εγκατάσταση της Python (A)


**Σχήμα 20.** Εγκατάσταση της Python (B)

## 2.5 microPython

Για τον προγραμματισμό μικροελεγκτών χρησιμοποιείται η *microPytho*n, μια έκδοση της γλώσσας Python σχεδιασμένη να ελαχιστοποιεί την χρήση μνήμης RAM (περιλαμβάνει αποθήκευση στη μνήμη ROM ο,τιδήποτε μπορεί να αποθηκευτεί, μικρούς ακέραιους σε έναν δείκτη, βελτιστοποιημένες μέθοδοι κλήσης και εύρος αντικειμένων κ.α.).

Η *microPython* διαθέτει δυο βασικά εργαλεία για γρήγορη υλοποίηση, πειραματισμό και βελτίωση του προγραμματισμού του μικροελεγκτή (Schwartz and Christiansen, 1999):

- *συσκευή USB*. Το discovery board αναγνωρίζεται ως μια απλή συσκευή USB (usb-stick). Συνδέοντας το σε μια θύρα USB του υπολογιστή (ακροδέκτης micro-USB του discovery board) εμφανίζονται τα αρχεία του μικροελεγκτή (με κώδικα σε Python). Έτσι, μπορούν να διαβαστούν, να επεξεργαστούν και να αποθηκευτούν απευθείας, με την χρήση ακόμα και ενός απλού επεξεργαστή κειμένου (notepad) και έπειτα να εκτελεστούν.
- *PERL*. Υποστηρίζει διαδραστικό προγραμματισμό μέσω Python promt (read – eval – print loop). Συνδέοντας και πάλι το discovery board (micro-USB) σε μια θύρα USB του υπολογιστή και με την χρήση ενός προγράμματος σειριακού τερματικού εξομοιωτή (π.χ. PuTTY - http://www.putty.org/) υπάρχει η δυνατότητα να γράφονται και να διαβάζονται απευθείας οι εντολές προς τον μικροελεγκτή, να εκτελούνται και να εμφανίζονται τα αποτελέσματα τους. Μπορούν έτσι να γίνουν απευθείας δοκιμές εντολών ή τμημάτων κώδικα, εργαλείο χρήσιμο κατά την διόρθωση του προγραμματισμού. Το όνομα *PERL* προκύπτει από τα αρχικά των αγγλικών λέξεων "*Practical Extraction and Report Language*" (Γλώσσα πρακτικής εξαγωγής και αναφοράς)[1]. Η *Perl* είναι μία πολύ δημοφιλής *αντικειμενοστραφής γλώσσα προγραμματισμού (object oriented programming)*. Συνήθως ένα πρόγραμμα σε *Perl* εκτελείται χρησιμοποιώντας άμεσα ή έμμεσα το *διερμηνέα (interpreter)* της γλώσσας. Η γλώσσα σχεδιάστηκε από τον *L. Wall* και ο πηγαίος κώδικάς της διατίθεται βάση της αδείας ανοικτού κώδικα *GPL*. Η πρώτη έκδοση της

[1]Schwartz, R. L., Christiansen, T. (1999). *Μάθετε την Perl, Δεύτερη Αμερικανική Έκδοση*, Εκδόσεις Κλειδάριθμος, σελ. 35.

γλώσσας εμφανίστηκε το 1987 ενώ μέχρι σήμερα βγαίνουν συνεχώς νέες εκδόσεις. Ως γλώσσα προγραμματισμού η *Perl* έχει ένα *δυναμικό σύστημα τύπων*, δηλαδή μία μεταβλητή αποκτά τύπο μόνο μετά από την ανάθεση μιας τιμής σε αυτή. Η *Perl* μπορεί να διαχειριστεί αριθμούς (δεκαδικούς και ακεραίους) και συμβολοσειρές. Επιπλέον, μπορούμε να έχουμε πίνακες απλούς (arrays) και συσχετιστικούς (associative arrays ή απλά hash tables), οι οποίοι είναι *μηχανισμοί οργάνωσης δεδομένων*. Μάλιστα με τους συσχετιστικούς πίνακες μπορούμε να δημιουργήσουμε δυναμικές δομές όπως δένδρα, στοίβες, ουρές κ.λπ. Υπάρχει πρόνοια για τον διαχωρισμό μεταβλητών στις οποίες αναθέτουμε απλές τιμές ή σύνθετες τιμές[2]. Στην *Perl* τα προγράμματα μπορούν να γραφτούν σε μία γραμμή κειμένου. Παρόλα αυτά, σχεδόν όλα τα προγράμματα της *Perl* χρησιμοποιούν εσοχές όπως και τα προγράμματα *C*. Η *Perl* χρησιμοποιείται από ένα ευρύ φάσμα χρηστών και σε πολλές εφαρμογές. Για παράδειγμα χρησιμοποιείται στη βιοπληροφορική, στη διαχείριση συστημάτων, στη διαχείριση ιστοτόπων, στην ανάκτηση πληροφοριών κ.λ.π.

[2] Perl, Wikipedia, βλέπε 2.

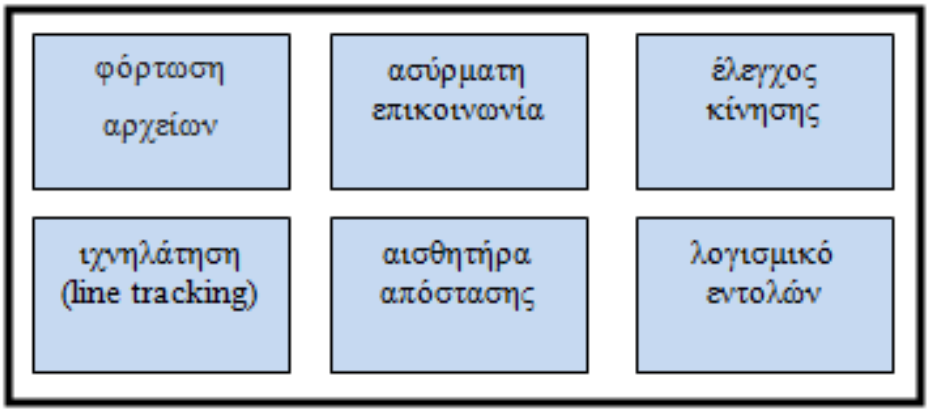
# ΚΕΦΑΛΑΙΟ ΤΡΙΤΟ

# ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΛΟΓΙΣΜΙΚΟΥ

## 3.1 Σχεδιασμός Λογισμικού

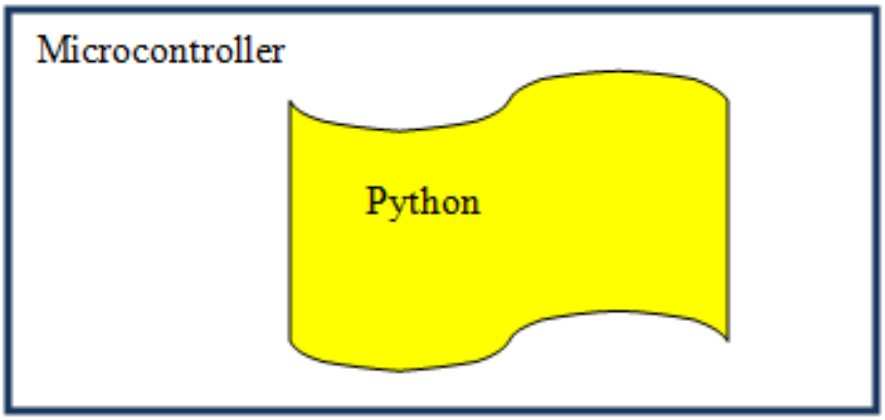
Το λογισμικό για την ανάπτυξη της εφαρμογής περιλμβάνει την εξής δομή (Σχ.21):

- λογισμικό φόρτωσης αρχείων
- λογισμικό ασύρματης επικοινωνίας
- λογισμικό ελέγχου κίνησης
- λογισμικό ιχνηλάτησης (line tracking)
- λογισμικού ασθητήρα απόστασης
- λογισμικό εντολών




**Σχήμα 21.** Συστατικά Λογισμικού Εφαρμογής

Το λογισμικό αναπτύχθηκε σε PYTHON σε περιβάλλον μικροελεγκτή:




**Σχήμα 22.** Περιβάλλον λογισμικού-μικροελεγκτή

Η σχεδίαση του προγράμματος ακολούθησε τις εξής αρχές:

- *αντικειμενοστραφής προγραμματισμός* ,
- χρήση *εσοχών* κατά τη συγγραφή του κώδικα (καλύτερη ευκρίνεια),
- και *τεκμηρίωση* ανά συστατικό λογισμικού (component).

## 3.2 Κώδικας Λογισμικού

Ακολουθούν οι κώδικες των επιμέρους συστατικών του λογισμικού της εφρμογής:

- *φόρτωση αρχείων*

boot.py

```
# boot.py -- run on boot-up

import pyb
import micropython
micropython.alloc_emergency_exception_buf(100)
import ultrasonic
import urandom

execfile('/flash/bluetooth.py')
execfile('/flash/anticrash.py')
execfile('/flash/movement.py')
execfile('/flash/linechaser.py')
execfile('/flash/modes.py')
execfile('/flash/commands.py')
```

- *Λογισμικό ασύρματης επικοινωνίας Bluetooth*

αρχική διαμόρφωση με τη χρήση AT Bluetooth

```
import pyb
from pyb import UART

uart = UART(4, 38400)
uart.init(38400, bits=8, stop=1, parity=None, timeout=1000, timeout_char=1000, read_buf_len=64)

pyb.repl_uart(uart)
uart.write('AT+UART=9600,1,0\r\n');print(uart.read())
uart.write('AT+NAME=pyRO\r\n');print(uart.read())
uart.write('AT+PSWD=1234\r\n');print(uart.read())
```

Bluetooth.py

```
from pyb import UART
BT = UART(4, 9600)
BT.init(9600, bits=8, parity=None, stop=1)
```

- Λογισμικό ελέγχου κίνησης

movement.py

```
    motor_A_pin1 = pyb.Pin.board.PE11
    motor_A_pin2 = pyb.Pin.board.PE12
    motor_B_pin1 = pyb.Pin.board.PE13
    motor_B_pin2 = pyb.Pin.board.PE14
    motor_A_pin1.init(motor_A_pin1.OUT_PP,pull=motor_A_pin1.PULL_NONE,af=-1)
    motor_A_pin2.init(motor_A_pin2.OUT_PP,pull=motor_A_pin2.PULL_NONE,af=-1)
    motor_B_pin1.init(motor_B_pin1.OUT_PP,pull=motor_B_pin1.PULL_NONE,af=-1)
    motor_B_pin2.init(motor_B_pin2.OUT_PP,pull=motor_B_pin2.PULL_NONE,af=-1)
pulses = pyb.Timer(2, freq=3000)
RightMotorSpeed = pulses.channel(3, pyb.Timer.PWM, pin=pyb.Pin.board.PA2, pulse_width_percent=0)
LeftMotorSpeed = pulses.channel(4, pyb.Timer.PWM, pin=pyb.Pin.board.PA3, pulse_width_percent=0)

# Assembly instructions
right_motor = [motor_A_pin1,motor_A_pin2]
left_motor = [motor_B_pin1,motor_B_pin2]

def motor_spin(motor,direction):
    if direction == 1:
        motor[0].value(True)
        motor[1].value(False)
    elif direction == -1:
        motor[0].value(False)
        motor[1].value(True)
    else:
        motor[0].value(False)
        motor[1].value(False)

def speed_adj(R_mot_percent,L_mot_percent):
    RightMotorSpeed.pulse_width_percent(R_mot_percent)
    LeftMotorSpeed.pulse_width_percent(L_mot_percent)

def speed(speed_percent):
    RightMotorSpeed.pulse_width_percent(speed_percent)
    LeftMotorSpeed.pulse_width_percent(speed_percent)

def stop():
    motor_spin(right_motor,0)
    motor_spin(left_motor,0)

def forward():
    motor_spin(right_motor,1)
    motor_spin(left_motor,1)

def backward():
    motor_spin(right_motor,-1)
    motor_spin(left_motor,-1)

def tilt_right():
    motor_spin(right_motor,-1)
    motor_spin(left_motor,1)

def tilt_left():
    motor_spin(right_motor,1)
    motor_spin(left_motor,-1)

def right_forward():
    motor_spin(right_motor,0)
    motor_spin(left_motor,1)

def left_forward():
    motor_spin(right_motor,1)
    motor_spin(left_motor,0)

def right_backward():
    motor_spin(right_motor,0)
    motor_spin(left_motor,-1)

def left_backward():
    motor_spin(right_motor,-1)
    motor_spin(left_motor,0)

# Same layout as any 8-Way Direction Pad:
# [1][2][3]  \ | /
# [4][0][5] -  0  -
# [6][7][8]  / | \
movement = [stop,
            left_forward,forward,right_forward,
            tilt_left,tilt_right,
            left_backward,backward,right_backward]

def move(direction):
    movement[direction]()
```

- *Λογισμικό ιχνηλάτησης*

linechaser.py

```
road_ir = pyb.Pin.board.PD0
road_ir.init(road_ir.OUT_PP,pull=road_ir.PULL_NONE,af=-1)
wheel = pyb.ADC(pyb.Pin.board.PC1)
line = pyb.ADC(pyb.Pin.board.PC2)
brightness = pyb.ADC(pyb.Pin.board.PC0)

def LuminosityCheck():
    if brightness.read() < 30:
      road_ir.high()
    if brightness.read() > 300:
      road_ir.low()

################## P Controler #################
# R_min-- # +chance to go out of road ########
R_min = 330
# Center values # +/- oscillation -/+ speed ##
R_max = 600
L_min = 2200
# L_max++ # +chance to go out of road ########
L_max = 3500

a_R = 100/(R_max-R_min)
b_R = -100*R_min/(R_max-R_min)
a_L = -100/(L_max-L_min)
b_L = 100*L_max/(L_max-L_min)
################## P Controler #################

################ PID Controler ################
center = 1700
lasterror = 0
integral = 0
derivative = 0
Lboost = 0
Rboost = 0
LineBoost = 0
Speed = 27
Kp = 30
Ki = 0.04
Kd = 90
```

```
Cbrakes = 0
Clasterror = 0
Cintegral = 0
Cderivative = 0
Cp = 30
Ci = 0
Cd = 100
################ PID Controler ################

def Acontrol():
    global integral
    global derivative
    global lasterror
    global Lboost
    global Rboost
    error = (center - wheel.read())/4000
    integral = error + integral
    derivative = error - lasterror
    correction = Kp*error + Ki*integral + Kd*derivative
    Rboost = -correction
    Lboost = +correction
```

```
def Bcontrol():
    global LineBoost
    inline = line.read()
    LineBoost = inline/3600

def Ccontrol():
    global Cintegral
    global Cderivative
    global Clasterror
    global Cbrakes
    error = (3700 - line.read())/4000
    Cintegral = error + Cintegral
    Cderivative = error - Clasterror
    Cbrakes = Cp*error + Ci*Cintegral + Cd*Cderivative

def chase1():
    Bcontrol()
    Acontrol()
    speed_adj(Speed + Rboost,Speed + Lboost)
    move(2)
    pyb.delay(4)
    move(0)
    pyb.delay(5)
```

```
def chase3():
    Ccontrol()
    Acontrol()
    speed_adj(Speed - Cbrakes + Rboost,Speed - Cbrakes + Lboost)
    move(2)
    pyb.delay(4)
    move(0)
    pyb.delay(5)
```

```
# Black - White

road = pyb.ADC(pyb.Pin.board.PC1)

def chase(margin,step):
    line = road.read()
    if line > margin:
        move(1)
    elif line < margin:
        move(3)
    pyb.delay(step)
    move(0)
    pyb.delay(step)

while True:
    chase(300,30)
```

---

```
# Black - White | Many resistors test

road = pyb.ADC(pyb.Pin.board.PC1)

eye1 = pyb.Pin.board.PD1
eye2 = pyb.Pin.board.PD2
eye3 = pyb.Pin.board.PD3
eye1.init(eye1.OUT_PP,pull=eye1.PULL_NONE,af=-1)
eye2.init(eye2.OUT_PP,pull=eye2.PULL_NONE,af=-1)
eye3.init(eye3.OUT_PP,pull=eye3.PULL_NONE,af=-1)

eye = [eye1,eye2,eye3]

# Start-Up resistance (starting with low intensity)
# eye1: R0=10K eye2: R1=100K eye3: R2=500K
resistance = 0
eye1.value(True)


def eye_intensity(resistance,scale):
    if scale=="up" and eye3.value() == False:
        eye[resistance].value(False)
        resistance = resistance + 1
        eye[resistance].value(True)
    elif scale=="down" and eye1.value() == False:
        eye[resistance].value(False)
        resistance = resistance - 1
        eye[resistance].value(True)
    return resistance

def chase(margin,step,resistance):
    white = road.read()
    if white > 2500:
        resistance = eye_intensity(resistance,scale="up")
        pyb.delay(2000)
    white = road.read()
    way = road.read()
    while way < round(white*(1+margin)):
        move(3)
        pyb.delay(step)
        move(0)
        pyb.delay(2*step)
        way = road.read()
    black = road.read()
    if black < 500:
        resistance = eye_intensity(resistance,scale="down")
        pyb.delay(2000)
    black = road.read()
    way = road.read()
    while way > round(black*(1-margin)):
        move(1)
        pyb.delay(step)
        move(0)
        pyb.delay(2*step)
        way = road.read()

while True:
    if BT.any() == True:
        for eyes in eye:
            eyes.value(False)
        move(0)
    else:
        chase(0.4,50,resistance)
```

```
# Left - Right

road = pyb.ADC(pyb.Pin.board.PC1)
road_eyes = pyb.Pin.board.PD0
road_eyes.init(road_eyes.OUT_PP,pull=road_eyes.PULL_NONE,af=-1)

# min value = 600, max value = 3800 -> center = 2200
left_margin = 1500
right_margin = 3000

def chase(min,max):
    if min < road.read() < max:
        move(2)
    if road.read() >= max:
        move(1)
    if road.read() <= min:
        move(3)
```

- λογισμικό αισθητήρα απόστασης

ultrasonic.py

```
# Ultrasonic library for MicroPython's pyboard.
# Compatible with HC-SR04 and SRF04.
#
# Copyright 2014 - Sergio Conde Gomez <skgsergio@gmail.com>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
##

import pyb

# Pin configuration.
# WARNING: Do not use PA4-X5 or PA5-X6 as the echo pin without a 1k resistor.

class Ultrasonic:
    def __init__(self, tPin, ePin):
        self.triggerPin = tPin
        self.echoPin = ePin

        # Init trigger pin (out)
        self.trigger = pyb.Pin(self.triggerPin)
        self.trigger.init(pyb.Pin.OUT_PP, pyb.Pin.PULL_NONE)
        self.trigger.low()

        # Init echo pin (in)
        self.echo = pyb.Pin(self.echoPin)
        self.echo.init(pyb.Pin.IN, pyb.Pin.PULL_NONE)

    def distance_in_inches(self):
        return (self.distance_in_cm() * 0.3937)

    def distance_in_cm(self):
        start = 0
        end = 0

        # Create a microseconds counter.
        micros = pyb.Timer(8, prescaler=83, period=0x3fffffff)
        micros.counter(0)

        # Send a 10us pulse.
        self.trigger.high()
        pyb.udelay(10)
        self.trigger.low()

        # Wait 'till whe pulse starts.
        while self.echo.value() == 0:
            start = micros.counter()

        # Wait 'till the pulse is gone.
        while self.echo.value() == 1:
            end = micros.counter()
```

```
63
64          # Deinit the microseconds counter
65          micros.deinit()
66
67          # Calc the duration of the recieved pulse, divide the result by
68          # 2 (round-trip) and divide it by 29 (the speed of sound is
69          # 340 m/s and that is 29 us/cm).
70          dist_in_cm = ((end - start) / 2) / 29
71
72          return dist_in_cm
```

anticrash.py

```
# Anticrash System - Ultrasonic Sensor (HC-SR04)

ranger = pyb.Pin.board.PD2
ranger.init(ranger.OUT_PP,pull=ranger.PULL_NONE,af=-1)

US_trigger = pyb.Pin.board.PB0
US_echo = pyb.Pin.board.PB1
US_sensor = ultrasonic.Ultrasonic(US_trigger, US_echo)

# Must first enable ranger to use range
def range():
      if ranger.value() == False:
            return 999
      else:
            return US_sensor.distance_in_cm()

def AvoidManual():
      if range() < 30:
            stop()
            pyb.delay(500)

def AvoidAuto():
        if range() < 30:
              stop()
              move(7)
              pyb.delay(250)
              if urandom.getrandbits(1) = True:
                    move(4)
                    pyb.delay(200)
                    stop()
              else:
                    move(5)
                    pyb.delay(200)
                    stop()
```

- *λογισμικό εντολών*

main.py

```
# pyRO - main code running

def BTread():
     if BT.any() == True:
         command(BT.readchar())

BTtimer = pyb.Timer(4)
BTtimer.init(freq=100)
BTtimer.callback(lambda t:BTread())
```

nodes.py

```
##  pyRO - Modes ##
#####  Lasers #####
lasers = pyb.Pin.board.PD1
lasers.init(lasers.OUT_PP,pull=lasers.PULL_NONE,af=-1)

def laser():
    if lasers.value() == False:
        lasers.value(True)
    else:
        lasers.value(False)
```

```
#####  Possition Lights #####
lights_pos = pyb.Pin.board.PD3
lights_pos.init(lights_pos.OUT_PP,pull=lights_pos.PULL_NONE,af=-1)
def lights():
    if lights_pos.value() == False:
        lights_pos.value(True)
    else:
        lights_pos.value(False)

#####  Beam Lights #####
lights_beam = pyb.Pin.board.PD4
lights_beam.init(lights_beam.OUT_PP,pull=lights_beam.PULL_NONE,af=-1)
def beam():
    if lights_beam.value() == False:
        lights_beam.value(True)
    else:
        lights_beam.value(False)

##### Linechaser (mode 1 PID) #####
def LT1():
    global lasterror
    global integral
    global derivative
    global BaseSpeed
    if road_ir.value() == False:
        road_ir.value(True)
        while BT.any() == False:
            LuminosityCheck()
            AvoidAuto()
            chase1()
    else:
        road_ir.value(False)
        stop()
        lasterror = 0
        integral = 0
        derivative = 0

#####  Linechaser (mode 2 P) #####
def LT2():
    if road_ir.value() == False:
        road_ir.value(True)
        while BT.any() == False:
            LuminosityCheck()
            AvoidAuto()
            chase2(a_R,b_R,a_L,b_L)
    else:
        road_ir.value(False)
        stop()

##### Linechaser (mode 3 2xPID) #####
def LT3():
    if road_ir.value() == False:
        road_ir.value(True)
        while BT.any() == False:
            LuminosityCheck()
            AvoidAuto()
            chase3()
    else:
        road_ir.value(False)
        stop()
        lasterror = 0
        integral = 0
        derivative = 0
        Clasterror = 0
        Cintegral = 0
        Cderivative = 0

##### Anticrash #####
def anticrash():
    if ranger.value() == False:
        ranger.value(True)
    else:
        ranger.value(False)

modes = [lights,beam,laser,anticrash,LT1,LT2,LT3]
```

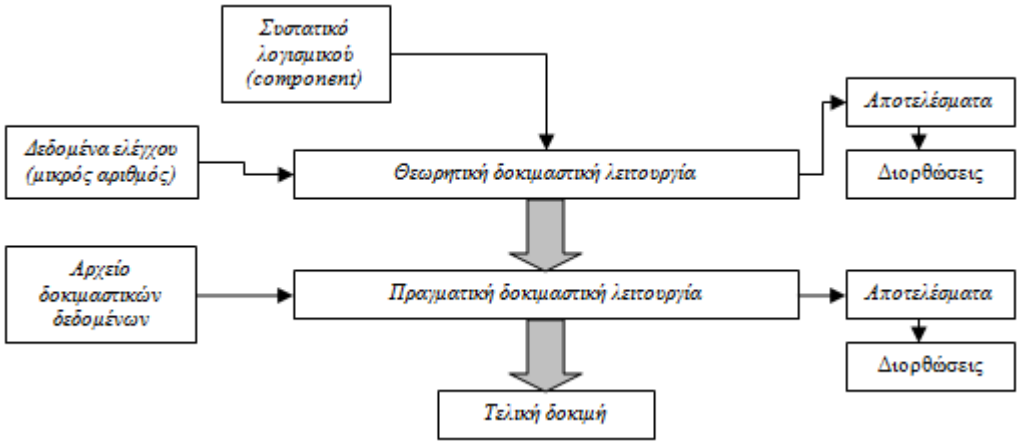
commands.py

```
# pyRO - Commands

def idle():
    move(0)

def manual():
    direction = BT.readchar()
    if direction == 2:
        while BT.any() == False:
            forward()
            AvoidManual()
    else:
        move(direction)

def speedbar():
    BT.readchar()
    BT.readchar()
    speed((BT.readchar()-47)*10)

def mode():
    modes[BT.readchar()]()

commands = [idle,manual,speedbar,mode]

def command(id):
    commands[id]()
```

## 3.3 Δοκιμή Λογισμικού

Κάθε τμήμα του λογισμικού δοκιμάστηκε σε δύο επίπεδα (Σχ23):

- *θεωρητική δοκιμαστική λειτουργία* (εκτέλεση του αλγόριθμου στο χαρτί για τυχόν λάθη) και
- *πραγματική δοκιμαστική λειτουργία* (εκτέλεση συστατικών λογισμικού για τυχόν λάθη).



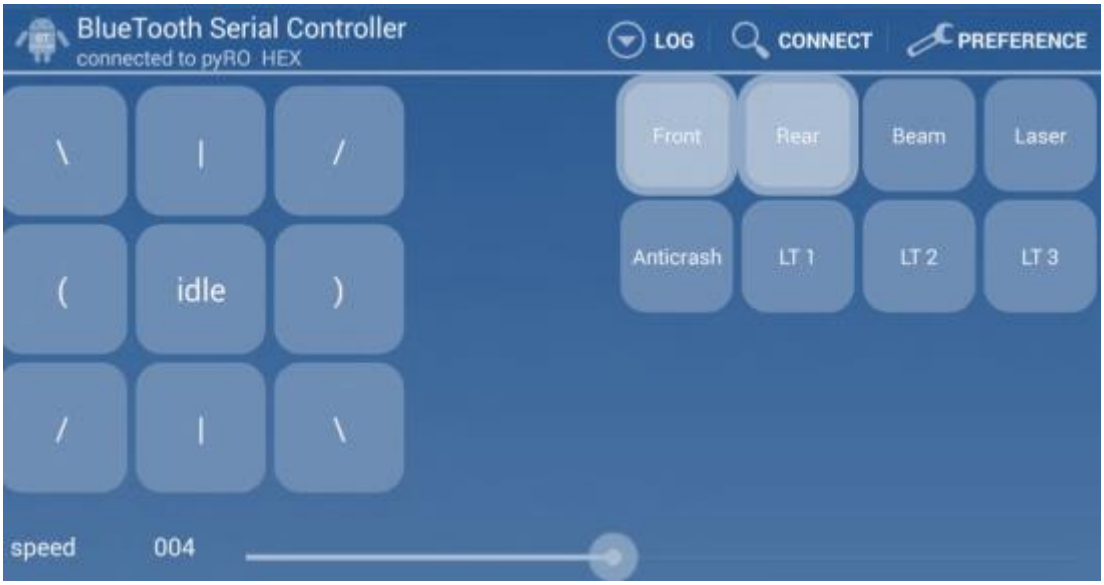

**Σχήμα 23.** Διαδικασία δοκιμών συστατικών του λογισμικού

Με την ολοκλήρωση των δοκιμών των επιμέρους συστατικών του λογισμικού ακολουθεί η τελική δοκιμή. Αυτή περιλαμβάνει ένα αριθμό δεδομένων που εισάγονται στο λογισμικό για έλεγχο της λειτουργίας του. Τυχόν προβλήματα οδηγούν σε διορθώσεις στον κώδικα.

### 3.4 Εφαρμογή σε Έξυπνο Κινητό Τηλέφωνο

Η ασύρματη δικτύωση του ρομπότ με το κινητό τηλέφωνο γίνεται με χρήση της τεχνολογίας *Bluetooth*. Είναι βασική η αξιοποίηση της εφαρμογής σε έξυπνο κινητό τηλέφωνο (smart phone) που προσφέρει σύνδεση και ανταλλαγή δεδομένων με το σύστημα Bluetooth του μικροελεγκτή. Στην παρούσα εφαρμογή χρησιμοποιήθηκε ένα κινητό τηλέφωνο με λειτουργικό σύστημα *android 4.1* και η εφαρμογή "*BT controller*" (*NEXT PROTOTYPES),* μια εφαρμογή *Bluetooth serial controller*, που προέκυψε από το Google Play Store. Η εφαρμογή αυτή προσφέρει την εύρεση και ζεύξη του ρομπότ με το κινητό τηλέφωνο, όπως και την εύκολη δημιουργία κουμπιών–συντομεύσεων (shortcut) στο κινητό τηλέφωνο, με όλες τις εντολές που είναι επιθυμητές να στέλνονται στο ρομπότ, κατά την λειτουργία του (Σχ.24).

Η σύνδεση του κινητού τηλεφώνου με το ρομπότ γίνεται με την επιλογή "*connect*". Στην αναζήτηση συσκευών που εμφανίζεται, επιλέγεται η συσκευή με όνομα "pyRO" και χρησιμοποιώντας τον κωδικό "*1234"* επιτυγχάνεται η σύνδεση των δυο συσκευών. Στο αριστερό τμήμα της εφαρμογής υπάρχει το χειριστήριο κίνησης του ρομπότ (χειροκίνητη λειτουργία). Δεξιά, στην πρώτη γραμμή υπάρχουν τα κουμπιά για τον φωτισμό του ρομπότ, ενώ στην δεύτερη, τα κουμπιά ενεργοποίησης διαφόρων τρόπων λειτουργίας του ρομπότ (anticrash, linetracking modes). Τέλος, στο κάτω μέρος της εφαρμογής υπάρχει μπάρα για χειροκίνητο καθορισμό της ταχύτητας του ρομπότ.




**Σχήμα 24.** Η εφαρμογή του BT controller στο έξυπνο τηλέφωνο (smart phone)

# ΣΥΜΠΕΡΑΣΜΑΤΑ

Η σχεδίαση του λογισμικό για τον έλεγχο του οχήματος, καθώς και η υλοποίηση του είναι εύχρηστη όπως και ο πειραματισμός ταχύς. Όσο αφορά τους αλγόριθμους linetracking, ο PID έλεγχος φάνηκε να είναι αξιόπιστος και πιο γρήγορος. Κύριος κανόνας σε αυτούς τους αλγορίθμους είναι το όχημα να μην βγει εκτός διαδρομής. Για κάθε όχημα και κάθε διαδρομή, υπάρχει ένα κατώφλι ταχύτητας, στην οποία το ρομπότ μπορεί να ολοκληρώσει επιτυχώς την διαδρομή. Επομένως, οι απλοί αλγόριθμοι .μπορούν να λειτουργήσουν ικανοποιητικά σε οχήματα μικρής ταχύτητας ή σε εύκολες διαδρομές, χωρίς την απαίτηση πιο σύνθετου αλγόριθμου.

Τέλος, η γλώσσα Python συνήθως δεν επιλέγεται για χρήση σε ενσωματωμένα συστήματα, σλλά στην παρούσα εφαρμογή αναδείχθηκε αρκετά πρακτική και εύχρηστη. Βοήθησε να δοκιμαστούν αρκετά τμήματα κώδικα και ολοκληρωμένοι αλγόριθμοι απευθείας από τον υπολογιστή στο ρομπότ, γεγονός σημαντικό κατά την διαδικασία υλοποίησης του λογισμικού. Βέβαια κατά την λειτουργία του, ο μικροελεγκτής επιβαρύνεται από επιπλέον κύκλους εργασιών, έτσι σε περίπτωση περιορισμένων πόρων του συστήματος ή επιθυμίας για βέλτιστη αξιοποίηση του ελεγκτή, προτείνεται μετά την πλήρη ολοκλήρωση του λογισμικού σε Python, να γίνει αντικατάσταση του με γλώσσα μηχανής, έτσι ώστε να παρακαμφθεί η άσκοπη και συνεχής για τον μικροελεγκτή διαδικασία της μετάφρασης.

# ΒΙΒΛΙΟΓΡΑΦΙΑ

ΕΛΛΗΝΙΚΗ

Γιακουμάκης, Μ. Διαμαντίδης, Ν. (2009). *Τεχνολογία Λογισμικού*. Εκδόσεις Σταμούλης, Αθήνα.

Βεσκούκης, Β. (2015). *Στοιχεία Τεχνολογία Λογισμικού*. ΣΕΑΒ, www.kallipos.gr.

Fox, A. Patterson, D.A. (2017). Τεχνολογία Ανάπτυξης Λογισμικού ως Υπηρεσίας. Εκδόσεις Κλειδάριθμος, Αθήνα.

Κάππος, Ι.Θ. και Μπούρας, Α.Σ. (2017). Αλγοριθμική και Προγραμματισμός Υπολογιστών σε Python. Εκδ΄σοεις Κλειδάριθμος, Αθήνα.

Μαγκούτης, Κ. Νικολάου, Χ. (2015). Εισαγωγή στον αντικειμενοστραφή προγραμματισμό με PYTHON. ΣΕΑΒ, www.kallipos.gr.

Schwartz, R. L., Christiansen, T. (1999). *Μάθετε την Perl, Δεύτερη Αμερικανική Έκδοση*, Εκδόσεις Κλειδάριθμος, Αθήνα.

ΞΕΝΗ

Boehm, B. W., A *Spiral Model of Software Development and Enhancement,* IEEE Computer, May 1988, pp. 61-72, 1988.

Boehm, B. W., *Software life cycle factors, Handbook of Software Engineering,* edited by C. Vick and C. V. Ramamoorthy, Van Nostrand Reinhold, New York, pp. 494-518, 1984.

Davis, A. M., E. H. Bersoff and E. R. Comer, *A Strategy for Comparing Aternative Software Development Life Cycle Models,* IEEE Trans. on Soft. Eng., Vol.14, No.10, pp. 1453-1461, 1988.

Fairley, R. E., *Software Engineering Concepts,* McGraw-Hill, 1985.

Khoshafian S., Abnous R., *Object Orientation: Concepts, Languages, Databases, User Interfaces,* Wiley.

Pressman, R. S., *Software Engineering-A Practitioners Approach,* McGraw-Hill.

Scacchi, W., *Models of Software Evolution: Life Cycle and Process,* SEI Curriculum Module SEI-CM-10-1.0, Carnegie Mellon University, Software Engineering Institute, 1987.

Sommerville, *I. Software Engineering,* London: Addison-Wesley.

Williams, R. D., *Management of Software Development, Handbook of Software Engineering,* Edited by C. R. Vick and C. V. Ramamoorthy, Van Nostrand Reinhold, 1984.