Ατομική Διπλωματική Εργασία

ΜΕΛΕΤΗ ΑΓΟΡΑΣ ΓΙΑ ΤΗΝ ΧΡΗΣΗ ΕΥΕΛΙΚΤΩΝ ΜΕΘΟΔΩΝ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ ΜΕ ΕΜΦΑΣΗ ΣΤΟ SCRUM

Μάριος Χρίστου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΛΕΤΗ ΑΓΟΡΑΣ ΓΙΑ ΤΗΝ ΧΡΗΣΗ ΕΥΕΛΙΚΤΩΝ ΜΕΘΟΔΩΝ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ ΜΕ ΕΜΦΑΣΗ ΣΤΟ SCRUM

Μάριος Χρίστου

Επιβλέπων Καθηγήτρια
Γεωργία Μ. Καπιτσάκη

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2012

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω την επιβλέποντα καθηγήτρια μου κα. Γεωργία Μ. Καπιτσάκη για την υποστήριξη, την καθοδήγηση και τις συμβουλές που μου προσέφερε καθ' όλη τη διάρκεια εκπόνησης αυτής της Διπλωματικής Εργασίας.

Θα ήθελα επίσης να ευχαριστήσω όλους όσους έλαβαν μέρος στην έρευνα (ανεξάρτητα πρόσωπα και εταιρίες), η οποία αποτελεί ένα σημαντικό τμήμα αυτής της διπλωματικής εργασίας. Παρόλο που οι ευχαριστίες δεν δύναται να μεταφερθούν προσωπικά στον κάθε ένα από τους συμμετέχοντες, νιώθω την ανάγκη να δηλώσω την ευγνωμοσύνη μου προς αυτούς, αφού η συμμετοχή τους στην έρευνα ήταν καίριας σημασίας για την ολοκλήρωση της εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω θερμά το φιλικό και κυρίως το οικογενειακό μου περιβάλλον που στάθηκε δίπλα μου καθ' όλη τη διάρκεια της εκπόνησης της εργασίας αυτής, αλλά και γενικότερα σε όλη την διάρκεια των προπτυχιακών μου σπουδών, γιατί χωρίς τη στήριξη και την ηθική συμπαράστασή τους τίποτα δε θα ήταν κατορθωτό.

Περίληψη

Ο τομέας της ανάπτυξης λογισμικού δεν υστερεί στην εισαγωγή νέων μεθοδολογιών. Τα τελευταία χρόνια, ένας μεγάλος αριθμός διαφορετικών προσεγγίσεων στην ανάπτυξη λογισμικού έχει προταθεί, αλλά μόνο λίγες έχουν επιζήσει για να χρησιμοποιηθούν σήμερα. Οι βιομηχανικοί υπεύθυνοι για την ανάπτυξη λογισμικού έχουν γίνει δύσπιστοι για τις «νέες» λύσεις που είναι δύσκολο να κατανοηθούν και παραμένουν έτσι αχρησιμοποίητες. Αυτό ήταν το κλίμα πριν από την εμφάνιση των ευέλικτων μεθόδων ανάπτυξης λογισμικού (Agile methodologies).

Οι ευέλικτες μέθοδοι ενσωμάτωσαν μια ευρεία συλλογή από καλές και δοκιμασμένες αξίες και πρακτικές που βοηθούν στην ανάπτυξη ποιοτικού λογισμικού. Η ανάπτυξη του λογισμικού γίνεται σε σύντομους επαναληπτικούς και αυξητικούς κύκλους, παρέχοντας την δυνατότητα της προσαρμογής και αντίδρασης στις αλλαγές που τίθενται από το διαρκώς μεταβαλλόμενο επιχειρησιακό περιβάλλον. Οι ευέλικτες μέθοδοι είναι προσανατολισμένες στους ανθρώπους και όχι στις διαδικασίες, όπως οι παραδοσιακές μέθοδοι, και βασίζονται πολύ στην ομαδική εργασία. Υπόσχονται ποιοτικότερο λογισμικό και ταχύτερη ανάπτυξη.

Μια από αυτές τις μεθόδους είναι και η Scrum. Δημιουργήθηκε για να αντιμετωπίσει τόσο τα προβλήματα, που προκαλούνται από τις μεταβολές των απαιτήσεων, όσο και εκείνα που προέρχονται από τους κινδύνους στα έργα λογισμικού. Βασίζονται σε ένα σύνολο από αρχές, που υποστηρίζουν την ολοκλήρωση του έργου μέσα στα προβλεπόμενα χρονικά όρια, ενώ παράλληλα επιτρέπουν στους υπεύθυνους ανάπτυξης του λογισμικού να ανταποκρίνονται στις μεταβαλλόμενες απαιτήσεις του πελάτη καθ' όλη τη διάρκεια του κύκλου παραγωγής.

Στόχος της παρούσας διπλωματικής εργασίας είναι η μελέτη της μεθοδολογίας Scrum στα πλαίσια χρήσης της από διάφορες εταιρίες. Αρχικά θα παρουσιαστούν κάποιες παραδοσιακές μεθοδολογίες στις οποίες βασίζονται οι νέες ευέλικτες μέθοδοι. Στη συνέχεια θα ακολουθήσει η ένταξη στην ευέλικτη ανάπτυξη λογισμικού. Οι περισσότερες από τις υπόλοιπες ευέλικτες μεθοδολογίες θα καθοριστούν

περιγράφοντας για καθεμίαν τις διαδικασίες και τις πρακτικές που τις χαρακτηρίζουν. Ακολούθως θα περιγραφεί εκτενώς η μεθοδολογία Scrum. Έχει οργανωθεί και πραγματοποιηθεί μια μελέτη πεδίου με σκοπό τη διερεύνηση της έκτασης και της χρήσης της μεθοδολογίας Scrum στην παγκόσμια βιομηχανία λογισμικού. Η έρευνα έχει διεξαχθεί σε εταιρίες παραγωγής λογισμικού. Για τον λόγο αυτό χρησιμοποιήθηκε ερωτηματολόγιο, σκοπός του οποίου είναι η εξαγωγή συμπερασμάτων σχετικά με την αποτελεσματικότητα των ευέλικτων μεθόδων και ειδικότερα της μεθοδολογίας Scrum. Τα δεδομένα, έχουν αναλυθεί κατάλληλα και στη συνέχεια παρουσιάζονται τα ευρήματα των εταιριών που εφάρμοσαν την Scrum. Τέλος πραγματοποιείται μια σύγκριση με την παραδοσιακή ανάπτυξη λογισμικού παρουσιάζοντας έτσι τα πλεονεκτήματα και τα μειονεκτήματα της μεθοδολογίας Scrum.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό καθορίζονται μέσα από βιβλιογραφική αναζήτηση, ορισμένοι ορισμοί και έννοιες σχετικά με την διαχείριση –πληροφοριακών- έργων, αντικείμενο το οποίο εξετάζουμε, ώστε να είναι πιο εύκολη η παρακολούθηση της εργασίας. Η διαχείριση έργων είναι μια προγραμματισμένη προσέγγιση με βάση την οποία γίνεται δυνατή η διαχείριση, εκτέλεση και ολοκλήρωση ενός έργου. Έχοντας ως δεδομένο το γεγονός ότι οι απαιτήσεις αυξάνονται και ως λογικό επακόλουθο αυξάνεται και η πολυπλοκότητα των έργων, τότε η ανάγκη για προγραμματισμό, έλεγχο κα διαχείριση των έργων γίνεται επιτακτική.

1.1.1 Τι είναι έργο

Έργο (Project) καλείται η μοναδική διαδικασία που αποτελείται από ένα σύνολο συντονισμένων και ελεγχόμενων δραστηριοτήτων με προκαθορισμένες τις ημερομηνίες έναρξης και λήξης τους, και που αναλαμβάνεται με σκοπό την επίτευξη ενός στόχου σύμφωνου με συγκεκριμένες απαιτήσεις και περιορισμούς χρόνου, κόστους και πόρων [BSI, 2002]. Συμφώνα με το εγχειρίδιο του ινστιτούτου διαχείρισης έργων (Project Management Institute, PMI), έργο είναι ένα προσωρινό εγχείρημα που στοχεύει στην δημιουργία ενός μοναδικού προϊόντος ή υπηρεσίας [PMBOK, 2004].

Τα έργα διαδραματίζουν σημαντικό ρόλο και μπορούν να καθορίσουν την επιτυχία μιας εταιρίας ή επιχείρησης. Αποτελούν διαδικασίες των οποίων το αποτέλεσμα είναι ένα καινούριο ή τροποποιημένο προϊόν ή υπηρεσία. Τα έργα συμβάλλουν στην αύξηση της παραγωγικότητας και συνεπώς των κερδών της εταιρίας, στην καλύτερη ποιότητα και την ικανοποίηση των πελατών και τον εμπλουτισμό του περιβάλλοντος εργασίας [Kerzner 2001]. Κάθε έργο διαφέρει ως προς το μέγεθος, το κόστος και τον χρόνο που χρειάζεται ώστε να ολοκληρωθεί. Μερικά παραδείγματα έργων είναι τα ακόλουθα:

- Ο σχεδιασμός και η οικοδόμηση ενός σπιτιού ή μεγάλου κτηρίου
- Η ανάπτυξη ενός πληροφοριακού συστήματος (λογισμικού)

Κάθε έργο, όπως αναφέραμε πιο πάνω, είναι προσωρινό, δηλαδή κάθε έργο έχει προκαθορισμένη αρχή και τέλος, καθώς και μοναδικό αφού η ανάπτυξη ενός έργου προϋποθέτει πώς δεν υπάρχει κάτι παρόμοιο του και έτσι παρουσιάζεται η ανάγκη για κάλυψη καινούριων αναγκών. Επίσης ένα έργο χαρακτηρίζεται από αβεβαιότητα καθώς δεν δύναται να κατανοηθεί απόλυτα από την αρχή ανάπτυξης του. Γι’ αυτό τον λόγο η ανάπτυξη χωρίζεται σε φάσεις ή βήματα. Συνοπτικά μπορούμε να συγκεντρώσουμε τα κυριότερα εμπόδια ή κινδύνους που μπορούν να εμφανιστούν κατά την ανάπτυξη ενός έργου [Δημητριάδης, 1999]:

- Υπέρβαση προϋπολογισμού.
- Υπέρβαση χρονικών ορίων.
- Μη καλά καθορισμένοι στόχοι και απαιτήσεις.
- Ελλιπές πληροφόρηση.
- Έλλειψη τυποποίησης .

1.1.2 Συμμετέχοντες σε ένα έργο

Οι συμμετέχοντες σε ένα έργο (Project stakeholders) μπορεί να είναι άνθρωποι ή και οργανισμοί που συμμετέχουν ενεργά στην ανάπτυξη του έργου ή επηρεάζονται –άμεσα ή έμμεσα- από την ολοκλήρωση του έργου. Είναι, επίσης, πιθανό οι ίδιοι να επηρεάζουν το έργο και τα αποτελέσματα του. Έτσι, όσοι εμπλέκονται με το έργο, η καταγραφή των απαιτήσεων και οι προσδοκίες τους όπως επίσης και ο καταμερισμός του ελέγχου και ο βαθμός επίδρασης του κάθε ενός καθορίζουν την επιτυχία ενός έργου.

Πιο κάτω αναφέρονται οι βασικοί συμμετέχοντες σε ένα έργο [PMBOK, 2004]:

- Διαχειριστής: το άτομο που διοικεί το έργο.
- Πελάτες: τα άτομα ή ο οργανισμός για τα οποία προορίζεται το προϊόν και που πρόκειται να το χρησιμοποιήσουν.
- Ο φορέας που αναλαμβάνει την υλοποίηση: ο οργανισμός που αναλαμβάνει να αναπτύξει το έργο και να το παραδώσει.
- Η ομάδα του έργου: άτομα που ανήκουν στον φορέα και αναλαμβάνουν να εκτελέσουν με επιτυχία τις φάσεις που απαρτίζουν το έργο.
- Διοίκηση: τα μέλη που ανήκουν στην ομάδα του έργου και είναι άμεσα εμπλεκόμενοι με την διοίκηση του έργου.
- Χορηγός: άτομα ή οργανισμός που χρηματοδοτεί την ανάπτυξη του έργου.
- Υπόλοιποι συμμετέχοντες: όσοι δεν έχουν άμεση σχέση με το έργο αλλά μπορεί να επηρεάζονται από την χρήση του ή τα αποτελέσματα που επιφέρει η χρήση του.

1.1.3 Διαχείριση έργων

Τα έργα, κάθε μορφής, ανέκαθεν υπήρχαν στην ζωή του ανθρώπου, από την αρχαιότητα μέχρι και το σήμερα. Ωστόσο το κεφάλαιο διοίκησης τους άρχισε να απασχολεί τον άνθρωπο και απέκτησε σημασία στα τέλη του προηγούμενου αιώνα. Το θέμα της διαχείρισης ή αλλιώς διοίκησης έργων (Project Management), έκανε την εμφάνιση του στην βιβλιογραφία του Management τις τελευταίες δύο δεκαετίες περίπου. Ένας από τους κύριους λόγους για τον οποίο σήμερα η διαχείριση έργων τυγχάνει εξαιρετικής σημασίας, είναι το γεγονός ότι έχει να κάνει με την διαχείριση

πόρων, μεταξύ των οποίων και του πιο σημαντικού, του ανθρώπινου πόρου. Το έργο εξαρτάται από τον χρόνο κατά τον οποίο θα πραγματοποιηθούν οι στόχοι που έχουν θεσπιστεί, καθώς επίσης και κατά πόσο το τελικό αποτέλεσμα θα είναι εντός του προϋπολογισμού. Εκτός αυτού, παραμονεύει και ο ανταγωνισμός. Είναι δεδομένο πως εάν κάποιος δεν καταφέρει να φέρει εις πέρας τις υποχρεώσεις που του έχουν ανατεθεί, τότε σίγουρα κάποιος άλλος θα τον αντικαταστάσει στο επόμενο έργο. Η διοίκηση έργων μπορεί να διατυπωθεί ως η εκπλήρωση των προκαθορισμένων στόχων του έργου χρησιμοποιώντας το σύνολο των διαθέσιμων πόρων (χρόνος, χρήματα, άνθρωποι, υλικά, μηχανήματα, ενέργεια, χώρος κ.α.). Η [PMBOK, 2004], ορίζει το ίδιο ως «τη διαδικασία κατά την οποία εφαρμόζουμε γνώσεις, δεξιότητες, εργαλεία και τεχνικές κατά την εκτέλεση των δραστηριοτήτων του έργου έτσι ώστε να ικανοποιηθούν οι απαιτήσεις και οι προσδοκίες των συμμετεχόντων.»

Τα κύρια χαρακτηριστικά ενός έργου είναι τα εξής :

1. Έχει προκαθορισμένες ημερομηνίες έναρξης και ολοκλήρωσης
2. Με τη λήξη του πρέπει να ικανοποιείται ένας συγκεκριμένος στόχος.
3. Υπόκειται σε περιορισμούς διαφόρων ειδών (χρόνου, κόστους ποιότητας κ.α.)
4. Οι διαθέσιμοι πόροι είναι περιορισμένοι.

Η διαχείριση έργων περιλαμβάνει τον σχεδιασμό, την οργάνωση, την παρακολούθηση της εκτέλεσης του έργου και τον έλεγχο των πόρων. Για να θεωρηθεί ένα έργο επιτυχημένο τότε θα πρέπει αυτό να έχει ολοκληρωθεί μέσα στα προκαθορισμένα χρονικά περιθώρια, να μην ξεφεύγει του προϋπολογισμού και να έχει γίνει κατάλληλη και σωστή διαχείριση των διαθέσιμων πόρων. Επιπλέον, την επιτυχία καθορίζουν σε μεγάλο βαθμό η αποδοχή του έργου από τον εκάστοτε πελάτη-χρήστη καθώς επίσης και οι αλλαγές που ενδεχομένως να χρειαστεί να γίνουν στο έργο μετά την παράδοσή του στον οργανισμό (ή εταιρία) που υλοποιεί το έργο αφού οι αλλαγές επιφέρουν αλλαγή στην ροή εργασίας του οργανισμού. Η σωστή διαχείριση έρχου επιφέρει σημαντικά οφέλη.

Το πεδίο της διαχείρισης έργων περιγράφεται από τις παρακάτω γνωστικές περιοχές [PMBOK, 2004]:

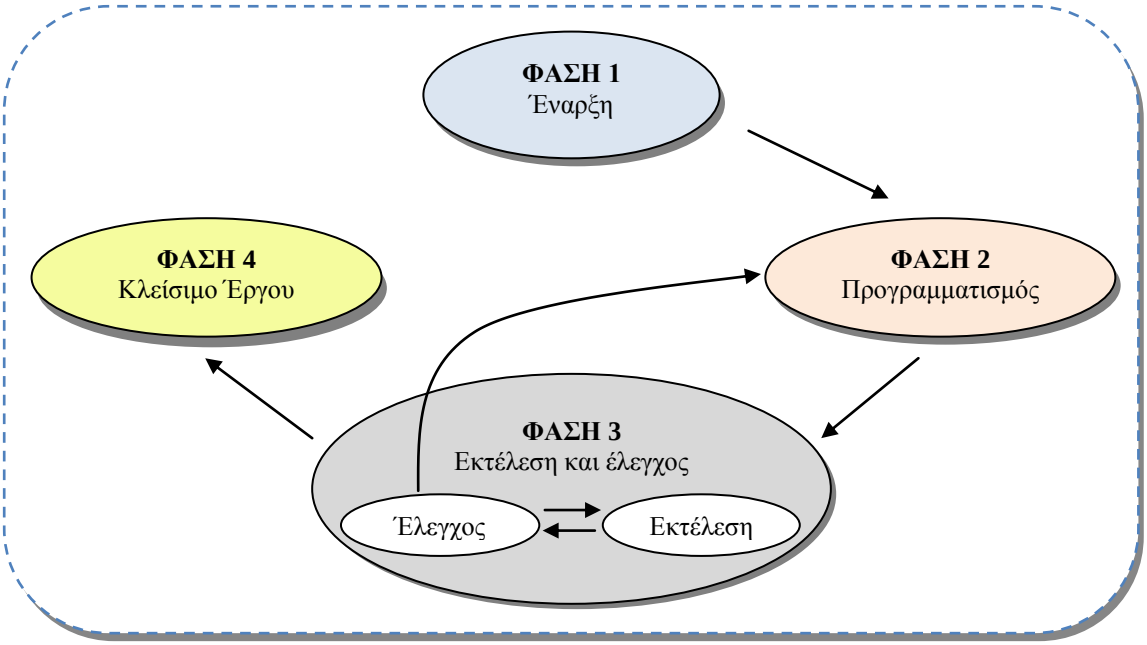
- Ενοποίηση του έργου (Project integration management): ενοποιεί τις βασικές διαδικασίες που περιγράψαμε πιο πάνω: τον σχεδιασμό, την οργάνωση, την παρακολούθηση της εκτέλεσης του έργου και τον έλεγχο των πόρων).
- Διαχείριση του αντικειμένου εργασιών (Project scope management): Είναι ο τρόπος με τον οποίο διασφαλίζεται ότι ένα έργο θα είναι σε θέση να ικανοποιεί όλες τις απαιτούμενες εργασίες ή λειτουργίες.
- Διαχείριση χρόνου (Project time management): Είναι ο τρόπος με τον οποίο διασφαλίζεται ότι ένα έργο θα ολοκληρωθεί έγκαιρα, μέσα στα προκαθορισμένη χρονική διορία.
- Διαχείριση κόστους (Project cost management): Είναι ο τρόπος με τον οποίο διασφαλίζεται ότι ένα έργο θα ολοκληρωθεί εντός των πλαισίων του προϋπολογισμού.
- Διαχείριση ποιότητας (Project quality management): Είναι ο τρόπος με τον οποίο διασφαλίζεται ότι τα παραδοτέα ενός έργου πληρούν τις ποιοτικές ανάγκες του πελάτη.
- Διαχείριση ανθρώπινων πόρων (Project human resource management): Είναι ο τρόπος με τον οποίο διασφαλίζεται η καλύτερη δυνατή εργασία των ανθρώπων που συμμετέχουν άμεσα σε ένα έργο.
- Διαχείριση επικοινωνίας (Project communication management): Είναι ο τρόπος με τον οποίο προγραμματίζεται η συλλογή και η διάδοση των απαραίτητων πληροφοριών σχετικά με το έργο.
- Διαχείριση κινδύνου (Project risk management): Είναι ο τρόπος με τον οποίο προσδιορίζεται οι κίνδυνοι κατά την ανάπτυξη ενός έργου, και εφευρίσκονται τρόποι για την διαχείριση αυτού.

1.1.3.1 Ο κύκλος ζωής ενός έργου

Καθώς το έργο είναι μοναδικό και ενέχει κάποιο βαθμό κινδύνου, οι εταιρίες που αναλαμβάνουν την εκτέλεση έργων συνήθως τα υποδιαιρούν σε φάσεις για να υπάρχει καλύτερος διοικητικός έλεγχος. Συλλογικά όλες μαζί οι φάσεις αυτές συνιστούν τον κύκλο ζωής του έργου. Ο κύκλος ζωής ενός έργου περιλαμβάνει μια λογική ακολουθία λειτουργιών για την επίτευξη των στόχων του έργου. Ανεξάρτητα από το είδος και την πολυπλοκότητα του, κάθε έργο διέρχεται από μια σειρά φάσεων κατά την διάρκεια της

ζωής του. Ο κύκλος ζωής ενός έργου (Project life cycle) γενικά περιλαμβάνει τέσσερις φάσεις όπως αυτές παρουσιάζονται στο Σχήμα 1.1. Κάθε φάση περιλαμβάνει μια σειρά από λογικά συνδεδεμένες δραστηριότητες η εκτέλεση των οποίων επιφέρει ένα αποτέλεσμα (βλ. Πίνακα 1.1). Οι φάσεις, κατά κύριο λόγο, εκτελούνται σειριακά αλλά κάποιες φορές μπορεί να επικαλύπτονται. Δεν υπάρχει ένας τυποποιημένος τρόπος ώστε να καθορίζονται οι φάσεις κάθε έργου. Ο κύκλος ζωής ενός έργου ορίζει:

- Ποιες εργασίες εκτελούνται σε κάθε μια από τις φάσεις
- Τα παραδοτέα που δημιουργούνται σε κάθε φάση
- Τα άτομα που εμπλέκονται σε κάθε φάση
- Ο τρόπος με τον οποίο ελέγχεται και εγκρίνεται κάθε φάση



Σχήμα 1.1 Κύκλος ζωής έργου

1.1.3.2 Φάσεις κύκλου ζωής έργου

<u>Φάση 1. Έναρξη</u>: Η έναρξη έργου είναι η πρώτη φάση του κύκλου ζωής του έργου και ουσιαστικά περιλαμβάνει την έναρξη λειτουργίας του έργου. Κατά την φάση αυτή εντοπίζεται το επιχειρησιακό πρόβλημα ή μια ευκαιρία και κατανοείται το επιχειρησιακό περιβάλλον. Πραγματοποιείται η έκθεση επιχειρησιακής σκοπιμότητας έργου (feasibility study) και ως αποτέλεσμα αυτής γίνεται ξεκάθαρος ο σκοπός και η λύση που πρόκειται να κατασκευαστεί. Η έκθεση επιχειρησιακής σκοπιμότητας ή

μελέτη σκοπιμότητας, είναι ένα έγγραφο στο οποίο καθορίζονται οι απαιτήσεις, οι περιορισμοί και τα αποτελέσματα που αναμένονται. Ένα σημαντικό μέρος της έκθεσης επιχειρησιακής σκοπιμότητας είναι ο έλεγχος βιωσιμότητας του έργου. Επίσης θα πρέπει να συνταχτεί μια ομάδα από κατάλληλα εξειδικευμένα άτομα που θα αναλάβει το έργο. Το στάδιο αυτό καθορίζει σημαντικά την επιτυχία του έργου αφού αν κάτι αποτύχει σε αυτό το στάδιο τότε δύσκολα θα ικανοποιούνται οι ανάγκες της επιχείρησης με το πέρας του έργου.

Φάση 2. Προγραμματισμός ή Σχεδιασμός: Μετά τον ορισμό του έργου και το διορισμό της ομάδας έργου, είμαστε έτοιμοι να εισέλθουμε στην αναλυτική φάση προγραμματισμού. Η φάση αυτή περιλαμβάνει τον προγραμματισμό/σχεδιασμο όλων των στοιχείων και παραμέτρων του έργου. Για τον λόγο αυτό εκπονούνται κατάλληλα σχέδια όπως Χρονοδιάγραμμα δραστηριοτήτων (ουσιαστικά προγραμματίζονται οι εργασίες και γίνεται ένας χρονικός προγραμματισμός), Σχέδιο Διαχείρισης πόρων κτλ. Κύριος στόχος της φάσης αυτής είναι να προγραμματίσει η διαχείριση του χρόνου, του κόστους και των πόρων κατάλληλα ώστε να εκτιμηθούν οι απαραίτητες εργασίες και να διαχειριστούν κατάλληλα τυχόν κίνδυνοι κατά την διάρκεια εκτέλεσης του έργου. Γίνεται αναλυτική περιγραφή και ορίζονται τα παραδοτέα του έργου καθώς διασπάται το έργο σε διάφορα τυχόν υποέργα (ανάλογα με το μέγεθος του έργου).

Φάση 3. Εκτέλεση & Έλεγχος Έργου: Αφού ορίσαμε το έργο και μια σειρά από λεπτομερή σχέδια, είμαστε τώρα έτοιμοι να εισέλθουμε στο στάδιο της εκτέλεσης του έργου. Η φάση αυτή περιλαμβάνει την εκτέλεση κάθε δραστηριότητας και εργασίας που ορίζεται στο Χρονοδιάγραμμα του Έργου. Κατά την υλοποίηση των δραστηριοτήτων και των εργασιών εκτελείται επίσης μία σειρά από διαχειριστικές διαδικασίες για την παρακολούθηση και τον έλεγχο των εξής: χρόνου, πόρων, κόστους, κινδύνων, ποιότητας, ζητημάτων, αλλαγών, διαδικασίας αποδοχής παραδοτέων, επικοινωνίας, κλπ. Η παρακολούθηση και ο έλεγχος αποτελείται από αυτές τις διαδικασίες που εκτελούνται για να παρατηρούν την εκτέλεση του έργου, έτσι ώστε πιθανά προβλήματα να μπορούν να εντοπιστούν εγκαίρως και διορθωτικά μέτρα να μπορούν να ληφθούν, όταν είναι απαραίτητο. Το βασικό πλεονέκτημα είναι ότι η απόδοση του έργου παρατηρείται και μετριέται τακτικά για να προσδιοριστούν αποκλίσεις από το αρχικό σχέδιο διαχείρισης του έργου. Όταν όλα τα παραδοτέα έχουν

παραχθεί και ο πελάτης έχει αποδεχθεί την τελική λύση, το έργο είναι έτοιμο για το κλείσιμο.

Φάση 4. Κλείσιμο Έργου: Η φάση αυτή περιλαμβάνει όλες τις απαραίτητες ενέργειες που διασφαλίζουν την ολοκλήρωση του έργου και το κλείσιμο του. Επίσης περιλαμβάνει την αξιολόγηση των διαδικασιών που χρησιμοποιήθηκαν στο έργο και των αποτελεσμάτων που επιτεύχθηκαν. [Tinnirello, 2001]

	Σύλληψη	**Σχεδιασμός**	**Υλοποίηση**	**Κλείσιμο**
Δεδομένα εισόδου	Πρόβλημα ή ευκαιρία, εντολή έργου, καταστατικό έργου.	Έγκριση για συνέχιση και σχεδιασμό το προϊόντος.	Έγκριση για υλοποίηση του έργου.	Σχέδιο θέσης σε λειτουργία, κοινοποίηση της ολοκλήρωσης του έργου.
Ακολουθούμενη διαδικασία	Πρόταση έργου, μελέτη σκοπιμότητας, προσδιορισμός συμμέτοχων, ανάλυση κόστους -ωφέλειας	Σχεδιασμός προϊόντος, λεπτομερής προγραμματισμός και κατάρτιση προϋπολογισμού.	Ανάθεση συμβάσεων και έκδοση οδηγιών, αγορά εξοπλισμού και υπηρεσιών, κατασκευή προϊόντος ή επίλυση προβλήματος, έναρξη λειτουργίας και έλεγχος του προϊόντος.	Δημιουργία τελικών σχεδίων που απεικονίζουν το πως κατασκευάστηκε και σύνταξη εγχειριδίων λειτουργίας.
Παραδοτέο	Μελέτη Σκοπιμότητας.	Βασικό πλάνο.	Πιστοποίηση Ολοκλήρωσης.	Έκθεση ολοκλήρωσης του έργου.
Έγκριση	Απόφαση ναι ή όχι σχετικά με το αν θα προχωρήσει το έργο στη φάση Σχεδιασμού.	Το έργο να εφαρμοστεί.	Να γίνει η προετοιμασία για να τεθεί σε λειτουργία.	Το έργο γίνεται αποδεκτό από τον πελάτη.

Πίνακας 1.1 Κύκλος ζωής έργου

Τα παραδοτέα (deliverables) αποτελούν έγκυρα προϊόντα εργασίας. Μπορεί να είναι μια μελέτη, ένα χρονοδιάγραμμα ή ένα πρωτότυπο. Καθώς προγραμματίζεται ένα έργο, επιβάλλεται να προσδιορίζονται κάποια ορόσημα (milestones). Ένα ορόσημο αποτελεί ένα χρονικό σημείο που ορίζει το τέλος κάποιας φάσης.

1.1.4 Διαχείριση έργων Πληροφορικής

Η διαχείριση έργων λογισμικού (Software project management) αναφέρεται στην γνώση, τις μεθόδους και τα μέσα που είναι απαραίτητα για την διαχείριση ενός έργου πληροφορικής, δηλαδή για την ανάπτυξη ενός λογισμικού συστήματος.

Τη δεκαετία του ΄60 καθώς και αρχές της δεκαετίας του ΄70 πολλά έργα λογισμικού αποτύγχαναν μαζικά. Το γεγονός αυτό έφερε στην επιφάνεια τα προβλήματα της διοίκησης έργων Πληροφορικής. Η αποτυχία των έργων αυτών δεν ήταν αποτέλεσμα της ανικανότητας των συμμετεχόντων ή των προγραμματιστών. Η αποτυχία οφειλόταν στον λάθος προγραμματισμό, σχεδιασμό και μεθόδους που χρησιμοποιούνταν. Το σύνολο αυτών των χρόνιων προβλημάτων ονομάζεται «κρίση λογισμικού». Είναι αξιοσημείωτο ότι το λογισμικό είναι ένα από τα λίγα κατασκευάσματα του ανθρώπου που πωλείται «ως έχει», χωρίς εγγύηση για τις ζημιές που μπορεί να επιφέρει η χρήση του [Βεσκούκης, 2000]. Στον πίνακα 1.2 φαίνονται τα πιο σημαντικά προβλήματα της κρίσης λογισμικού.

Εξαιρετικά δύσκολη διαδικασία κατασκευής.	Δεν είναι πάντα σαφές ποια βήματα πρέπει να γίνουν, με ποια σειρά, με ποια ενδιάμεσα προϊόντα κτλ.
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 κτλ. από διαφορετικούς κατασκευαστές.	Παρενέργειες μεταβολών σε στοιχεία που πριν λειτουργούσαν, πρόχειρες λύσεις.

Πίνακας 1.2 Ορισμένα σημαντικά χαρακτηριστικά της κρίσης λογισμικού [Βεσκούκης, 2000]

Κατά καιρούς εμφανίζονταν πολλά προϊόντα ή μέθοδοι που υποστήριζαν ότι μπορούν να επιλύσουν τα προβλήματα αυτά. Κάτι τέτοιο δεν έγινε, και για τον λόγο αυτό αναπτύχθηκε ένας ειδικός κλάδος της Πληροφορικής, η «Τεχνολογία Λογισμικού» (Software Engineering).

1.1.4.1 Τεχνολογία λογισμικού

Αναμφίβολα, ένα από τα σημαντικότερα κατασκευάσματα του ανθρώπου είναι ο ηλεκτρονικός υπολογιστής. Οι δυνατότητες του, χρόνο με το χρόνο αυξάνονται ολοένα και περισσότερο ενώ είναι σήμερα προσιτοί στους πλειστούς ανθρώπους και αποτελούν αναπόσπαστο μέρος της επαγγελματικής και προσωπικής τους ζωής. Σημαντικό είναι το γεγονός ότι η εποχή που διανύουμε χαρακτηρίζεται ως «εποχή της πληροφορίας», με το διαδίκτυο (Internet) και τις διαδικτυακές εφαρμογές να κατέχουν πρωταγωνιστικό ρόλο. Οι συνεχόμενες εξελίξεις γίνονται εφικτές χάρη στην ύπαρξη και λειτουργία ενός συνόλου πολύπλοκων εφαρμογών λογισμικού. Ο Η/Υ ως υλικό δεν θα είχε κάποια λειτουργικότητα χωρίς την ύπαρξη και χρήση συνοδευτικού λογισμικού. Οι υπολογιστές μπορούν να εκτελούν απλές εντολές με μεγάλη ταχύτητα, όμως με τρόπο που ένας κοινός άνθρωπος δεν μπορεί να κατανοήσει, καθώς απευθύνονται σε μηχανές. Το λογισμικό είναι ο «μεσολαβητής» ώστε ο άνθρωπος να μπορεί μέσω αυτού να εκμεταλλευτεί τις δυνατότητες ενός υπολογιστή. Θα μπορούσαμε να ορίσουμε το Λογισμικό ως τη συλλογή από προγράμματα υπολογιστών, διαδικασίες και οδηγίες χρήσης που εκτελούν ορισμένες εργασίες σε ένα υπολογιστικό σύστημα [http://el.wikipedia.org/wiki, 2007].

Μέρα με τη μέρα η υποκατάσταση των ανθρώπινων εργασιών από τους ηλεκτρονικούς υπολογιστές αυξάνεται με εκθετικούς ρυθμούς. Οι υπολογιστές ενδέχεται να αυτοματοποιούν ολικώς ή μερικώς ανθρώπινες δραστηριότητες καθώς δεν υπάρχει κανένας τομέας της ανθρώπινης εργασίας που να μην εξαρτάται από κατάλληλα προγραμματισμένους υπολογιστές. Καθημερινές δραστηριότητες, όπως έκδοση αποδείξεων είσπραξης σε μια υπεραγορά αλλά και πιο πολύπλοκες, όπως τραπεζικές συναλλαγές βασίζονται σε υπολογιστές που υποστηρίζουν κατάλληλα και εξειδικευμένα συστήματα λογισμικού, απαιτώντας από αυτά αυξημένη αξιοπιστία. Τα συστήματα λογισμικού λαμβάνουν, πλέον, καίρια θέση στην εξέλιξη της

ανθρωπότητας. Οι απαιτήσεις του ανθρώπου αυξάνονται συνεχώς και το γεγονός αυτό οδηγεί σε αυξημένες απαιτήσεις από τα συστήματα, με αποτέλεσμα το μέγεθος και η πολυπλοκότητα τους να πολλαπλασιάζονται με την πάροδο του χρόνου. Φυσικό επακόλουθο των αυξημένων απαιτήσεων από τα συστήματα, είναι οι αυξημένες απαιτήσεις από την διαδικασία ανάπτυξής τους.

Θα μπορούσαμε να παραλληλίσουμε την διαδικασία ανάπτυξης ενός συστήματος λογισμικού, με την αντίστοιχη διαδικασία ανέγερσης ενός κτηρίου. Για να αναγερθεί ένα μικρό κτίσμα δεν είναι απαραίτητο να ακολουθηθούν όλες οι προβλεπόμενες διαδικασίες και να εμπλακούν όλοι οι απαραίτητοι μηχανικοί. Σε μια τέτοια όμως περίπτωση το τελικό αποτέλεσμα καθώς και η μελλοντική του κατάσταση θα είναι αμφίβολα. Εάν όμως πρόκειται για μεγαλύτερο κτήριο, τότε η διαδικασία ανέγερσης του επιτάσσει την ύπαρξη και εμπλοκή μηχανικών. Η έναρξη των εργασιών ανέγερσης και υλοποίησης ενός σύγχρονου έργου, προϋποθέτει την ύπαρξη όλων των απαραίτητων μελετών, σχεδίων και τον προγραμματισμό των εργασιών. Όλα αυτά διαδραματίζουν σημαντικό ρόλο στην όλη φάση της ανέγερσης ενός οικοδομήματος. Αντίστοιχα, για την ανάπτυξη ενός σύγχρονου συστήματος λογισμικού, προαπαιτείται η ύπαρξη ενός σχεδίου, δηλαδή ο προγραμματισμός κάποιων βημάτων κατά τα οποία θα πληρείται η πληθώρα των απαιτήσεων που έχουν τα σύγχρονα συστήματα. Το έργο για την δημιουργία ενός τέτοιου σχεδίου είναι εξαιρετικά δύσκολο και στο στάδιο αυτό εμπλέκεται η επιστήμη της Τεχνολογίας Λογισμικού (Software Engineering). Τεχνολογία λογισμικού ορίζεται «η συστηματική προσέγγιση για την ανάλυση, τη σχεδίαση, την αξιολόγηση, την υλοποίηση, τον έλεγχο, τη συντήρηση και τον επανασχεδιασμό λογισμικού, δηλ. την εφαρμογή της μηχανικής στο λογισμικό» [Laplante, 2007]. Η διαδικασία ανάπτυξης λογισμικού αποτελείται από ορισμένα μέρη το κάθε ένα από τα οποία είναι σημαντικό ώστε να εξασφαλιστεί η επιτυχία ενός έργου. Τεχνολογία λογισμικού είναι η διαδικασία με την οποία οι ανάγκες των χρηστών μεταφράζονται σε ένα προϊόν λογισμικού. Η διαδικασία περιλαμβάνει τη μετατροπή των αναγκών των χρηστών σε απαιτήσεις λογισμικού, μετατροπή των απαιτήσεων του λογισμικού στο σχεδιασμό του συστήματος, την υλοποίηση του σχεδιασμού σε κώδικα, τον έλεγχο του κώδικα, και μερικές φορές, την εγκατάσταση και τον έλεγχο του λογισμικού για επιχειρησιακή χρήση. Οι δραστηριότητες αυτές μπορούν να επικαλύπτονται ή να εκτελούνται επαναληπτικά.

Αναμενόμενα χαρακτηριστικά του λογισμικού και της διαδικασίας κατασκευής του, είναι οπωσδήποτε η ποιότητα, η μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα με το ελάχιστο δυνατό κόστος παραγωγής και συντήρησης. Η Τεχνολογία λογισμικού περιλαμβάνει τον καθορισμό των βημάτων και την σειρά με την οποία πρέπει να γίνονται, καθώς και την περιγραφή με σαφή και κατανοητό τρόπο όλων των προϊόντων που παράγονται μετά την εκτέλεση αυτών των ενεργειών.

Το Λογισμικό, όπως αναφέραμε, έχει αποκτήσει κρίσιμη σημασία για την πρόοδο σε όλους σχεδόν τους τομείς της ανθρώπινης ζωής. Οι καλές γνώσεις προγραμματισμού και αλγορίθμων δεν επαρκούν πλέον για την κατασκευή προγραμμάτων. Όπως είδαμε, υπάρχουν αρκετά προβλήματα όσο αφορά το κόστος, την προθεσμία παράδοσης, της συντήρηση και την ποιότητα πολλών προϊόντων λογισμικού.

Η Τεχνολογία Λογισμικού, έχει ως στόχο την επίλυση αυτών των προβλημάτων με την παραγωγή καλής ποιότητας προϊόντων λογισμικού, έτοιμα στον προκαθορισμένο χρόνο και πάντα μέσα στα πλαίσια του προϋπολογισμού. Για την επίτευξη αυτού του στόχου, πρέπει να εστιάσουμε με ένα πειθαρχημένο τρόπο τόσο στην ποιότητα του προϊόντος όσο και στη διαδικασία που χρησιμοποιείται για να την ανάπτυξη του προϊόντος.

Κατά την πρώτη διάσκεψη για την ανάπτυξη λογισμικού το 1968, ο Fritz Bauer [Bauer, 1968] όρισε την Τεχνολογία λογισμικού ως , «Η καθιέρωση και χρήση ορθών αρχών μηχανικής, ώστε να κατασκευάσουμε οικονομικά λογισμικό που είναι αξιόπιστο και λειτουργεί αποτελεσματικά σε πραγματικά μηχανήματα». Ο Stephen Schach [Schach, 1990] όρισε το ίδιο με το "Μια διαδικασία που έχει ως στόχο την παραγωγή ποιοτικού λογισμικού, που παραδίδεται στην ώρα του, στα πλαίσια του προϋπολογισμού, και το οποίο πληροί τις απαιτήσεις του ». Και οι δύο ορισμοί είναι δημοφιλής και αποδεκτοί από την πλειοψηφία. Ωστόσο, λόγω της αύξησης των εξόδων συντήρησης του λογισμικού, στόχος τώρα είναι η στροφή προς την παραγωγή ποιοτικού λογισμικού που είναι διατηρήσιμο, παραδίδεται εγκαίρως, εντός προϋπολογισμού, αλλά και ικανοποιεί τις απαιτήσεις του. Αποτελεί ένα σημαντικό λόγο που μας «αναγκάζει» να αναζητήσουμε καινούργιες μεθοδολογίες ανάπτυξης συστημάτων λογισμικού.

1.1.4.2 Σύντομη ιστορική αναδρομή στην τεχνολογία λογισμικού

Τα πρώτα χρόνια που οι υπολογιστές έκαναν την εμφάνισή τους, οι δημιουργοί λογισμικού ακολουθούσαν την διαδικασία code and fix [Βεσκούκης, 2000]. Το μοντέλο αυτό ουσιαστικά χαρακτηρίζεται από δύο βήματα, όπως άλλωστε δηλώνει και το όνομα του: συγγραφή κώδικα και διόρθωση τυχόν προβλημάτων. Δεν γινόταν κανένας προγραμματισμός, έρευνα για απαιτήσεις, σχεδιασμός ή έλεγχος, με αποτέλεσμα πολύ σύντομα το μοντέλο αυτό να αποδειχτεί λανθασμένο αφού αρκετά προβλήματα έκαναν την εμφάνισή τους. Το σημαντικότερο πρόβλημα ήταν ότι κάθε παρεμβατική αλλαγή βελτίωσης στον κώδικα είχε μεγάλα κόστη συντήρησης. Αποτελούσε τον πιο γρήγορο αλλά καθόλου αποδοτικό τρόπο ανάπτυξης λογισμικού [Charvat, 2003]. Το γεγονός αυτό οδήγησε στην θέσπιση μιας φάσης πριν την συγγραφή κώδικα, η οποία ονομάζεται φάση σχεδιασμού. Με το πέρασμα του καιρού διαπιστώθηκε ότι παρά το γεγονός ότι τα συστήματα διέρχονται από την φάση αυτή πριν ξεκινήσει η κωδικοποίηση, ήταν πολλές φορές ανίκανα να καλύψουν τις πραγματικές ανάγκες του χρήστη. Και με τον τρόπο αυτό υιοθετήθηκε ακόμα μια φάση πριν τον σχεδιασμό, η φάση της ανάλυσης. Το αποτέλεσμα ήταν γύρω στο 1959 να διαμορφωθεί το κατά στάδια μοντέλο (stage-wise), από το οποίο και προήλθε το γνωστό μοντέλο του καταρράκτη (waterfall model). Παράλληλα είχε τονισθεί ότι το κόστος διόρθωσης του κώδικα ήταν πολύ μεγάλο, και έτσι η ανάγκη της προετοιμασίας για έλεγχο και τροποποιήσεις πριν την συγγραφή κώδικα έγινε επιτακτική.

Η φύση και η πολυπλοκότητα του λογισμικού έχουν αλλάξει σημαντικά τα τελευταία 30 χρόνια. Το 1970, οι εφαρμογές έτρεχαν σε ένα μόνο επεξεργαστή. Οι σημερινές εφαρμογές είναι πολύ πιο πολύπλοκες και σύνθετες. Συνήθως έχουν γραφική διεπαφή και αρχιτεκτονική τύπου πελάτη-εξυπηρετητή (client-server). Συχνά τρέχουν σε ένα ή δύο επεξεργαστές, κάτω από διαφορετικά λειτουργικά συστήματα, και σε γεωγραφικά κατανεμημένες μηχανές. Σημαντικό είναι, λοιπόν, η τεχνολογία λογισμικού να ανανεώνεται και να εφευρίσκει τρόπους ώστε να μπορεί να στηρίξει την παραγωγή σύγχρονων απαιτητικών λογισμικών.

1.1.4.3 Μοντέλα κύκλου ζωής Λογισμικού

Όπως είδη αναφέραμε, η Τεχνολογία Λογισμικού ασχολείται με την εξεύρεση και θεμελίωση τρόπων για να περιγράφεται, να κατασκευάζεται και να συντηρείται Λογισμικό καλής ποιότητας, με τη μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα και τα ελάχιστο δυνατό κόστος. Για να το επιτύχει αυτό η Τεχνολογία Λογισμικού θα πρέπει να ορίσει μια σειρά από ενέργειες. Πρέπει να οριστούν κάποιες γενικές φάσεις από τις οποίες θα περάσει η κατασκευή του λογισμικού και επιπλέον να οριστούν τα επιμέρους βήματα που πρέπει να ακολουθηθούν σε κάθε μια από αυτές τις φάσεις. Υπάρχουν πολλοί διαφορετικοί τρόποι για την κατασκευή και συντήρηση ενός λογισμικού, οι οποίοι ονομάζονται μοντέλα κύκλου ζωής (Software lifecycles). Δεν υπάρχει καθορισμένος αριθμός φάσεων που απαρτίζουν τον κύκλο ζωής ανάπτυξης λογισμικού.

1.1.4.4 Η έννοια του μοντέλου κύκλου ζωής

Οποιαδήποτε εφαρμογή λογισμικού, από την σύλληψη της ιδέας μέχρι και την απόσυρση της, περνάει διάφορες προκαθορισμένες φάσεις, σε κάθε μια από τις οποίες γίνονται ορισμένες εργασίες. Γενικά οι φάσεις από τις οποίες ενδέχεται να διέλθει ένα σύστημα λογισμικού είναι οι εξής:

Σχήμα 1.2 Γενικές φάσεις κύκλου ζωής λογισμικού

Μια δραστηριότητα ή διαδικασία ανάπτυξης λογισμικού καθορίζει ποιες ενέργειες πρέπει να γίνουν ώστε να πετύχουμε το επιθυμητό αποτέλεσμα σε οποιαδήποτε από τις φάσεις του κύκλου ζωής. Μια μεθοδολογία ανάπτυξης καθορίζει το πώς θα πρέπει να εκτελούνται οι δραστηριότητες (ποιές επιμέρους ενέργειες περιλαμβάνουν, τι γίνεται σε κάθε μια, ποια προϊόντα παράγονται καθώς και πότε αυτές θεωρούνται ολοκληρωμένες). Ένα Μοντέλο κύκλου ζωής Λογισμικού είναι μια περιγραφή των δραστηριοτήτων και των επιμέρους φάσεων από τις οποίες διέρχεται μια εφαρμογή λογισμικού από την σύλληψη μέχρι την απόσυρση της, καθώς και των εργασιών που λαμβάνουν χώρα σε κάθε μια από τις φάσεις αυτές.

Στόχος ενός μοντέλου κύκλου ζωής, είναι να καθοδηγεί σωστά τον κατασκευαστή ώστε αυτός να πετύχει το καλύτερο δυνατό αποτέλεσμα, δηλαδή την καλύτερη δυνατή υλοποίηση των διαδικασιών ανάπτυξης λογισμικού. Μια καλή υλοποίηση είναι αυτή που είναι περισσότερο παραγωγική, με λιγότερα σφάλματα και το ελάχιστο δυνατό κόστος παραγωγής. Βέβαια το μοντέλο που επιλέγεται σε κάθε περίπτωση εξαρτάται από το θέμα της εφαρμογής, τις προτιμήσεις του κατασκευαστή και το εκάστοτε περιβάλλον ανάπτυξης.

Ένας σημαντικός παράγοντας που επιτάσσει την χρήση των μοντέλων κύκλου ζωής, είναι το κόστος. Το κόστος με την ευρύτερη σημασία του. Το κόστος διόρθωσης σφαλμάτων και αλλαγών επιφέρει οικονομικές επιβαρύνσεις και χρονικές καθυστερήσεις.

Υπάρχουν αρκετά μοντέλα κύκλου ζωής, τα οποία διαφέρουν ως προς τον τρόπο κατασκευής του προϊόντος, αλλά και ως τις επιμέρους φάσεις που προτείνουν και την επαναληπτικότητα. Στο επόμενο κεφάλαιο θα εξετάσουμε ορισμένα παραδοσιακά μοντέλα κύκλου ζωής λογισμικού με σκοπό να κατανοήσουμε τις βασικές αρχές τους ώστε να προκύψει μια ομαλή μετάβαση στην έννοια των ευέλικτων πρακτικών.

1.1.5 Σημαντικοί παράγοντες διαχείρισης έργων

Κατά την ανάπτυξη ενός έργου (λογισμικού ή μη) θα πρέπει να δίνεται έμφαση στους πιο κάτω παράγοντες κατά την διάρκεια όλου του κύκλου ζωής του έργου:

- Ποιότητα: Η ποιότητα του έργου μπορεί να προέλθει μέσα από την ακριβή τήρηση των προτύπων, την εφαρμογή αποτελεσματικών μεθόδων ανάπτυξης και τον τακτικό έλεγχο σε όλες τις φάσεις ανάπτυξης. Οι υπεύθυνοι για την ανάπτυξη του έργου θα πρέπει να συνεργάζονται άψογα με τους εκάστοτε πελάτες.
- Κόστος: Το κόστος ανάπτυξης ενός έργου αποτελεί ίσως τον βασικότερο από τους παράγοντες που θα ορίσουν την επιτυχία ενός έργου. Εάν το έργο ανταποκρίνεται απόλυτα στις απαιτήσεις του πελάτη και είναι ποιοτικό, αλλά έχει ξεπεράσει σε μεγάλο βαθμό τον προϋπολογισμό τότε το έργο μπορεί να θεωρηθεί «αποτυχημένο».

- Παραγωγικότητα: Η επαρκής παραγωγικότητα μπορεί να οδηγήσει σε αισθητή μείωση του κόστους ενός έργου.
- Κίνδυνοι: Θα πρέπει να καταβάλλεται προσπάθεια για αποτελεσματική διαχείριση και αντιμετώπιση των κινδύνων που θα προκύψουν.
- Χρόνος: Ο χρόνος έχει σχεδόν ίσο βαθμό σημαντικότητας με τον κόστος παραγωγής. Ένα έργο για να θεωρείται επιτυχημένο, εκτός των άλλων, θα πρέπει να ολοκληρωθεί εντός του χρονοδιαγράμματος.

Κεφάλαιο 2

Παραδοσιακή Ανάπτυξη Λογισμικού

2.1 Εισαγωγή

Όπως αναφέραμε στο προηγούμενο κεφάλαιο, η «μεθοδολογία ανάπτυξης λογισμικού» στην επιστήμη της Πληροφορικής, αποτελεί το μοντέλο που χρησιμοποιείται για να δομηθεί, να προγραμματιστεί και να ελεγχθεί η διαδικασία της παραγωγής λογισμικού.

Κατά καιρούς έχουν αναπτυχθεί αρκετά τέτοια μοντέλα, το κάθε ένα από αυτά με αποδεδειγμένες δυνάμεις και αδυναμίες. Για κάθε έργο πληροφορικής δύναται να είναι κατάλληλη διαφορετική μεθοδολογία ανάπτυξης λογισμικού. Κάθε μια από τις υπάρχουσες μεθοδολογίες απευθύνεται σε ένα συγκεκριμένο είδος έργου, ανάλογα με τα τεχνικά και οργανωτικά χαρακτηριστικά του. Κάθε μοντέλο έχει τη δική του προσέγγιση και φιλοσοφία στην ανάπτυξη λογισμικού. Συχνά τα μοντέλα είναι

συνυφασμένα με ένα οργανισμό ο οποίος υιοθετεί το εκάστοτε μοντέλο, το αναπτύσσει και προωθεί την χρήση του.

Πριν την εμφάνιση των μοντέλων ανάπτυξης λογισμικού, υπήρχε ένα εργαλείο το οποίο θα μπορούσε να αποτελέσει και μοντέλο: τα λογικά διαγράμματα ή διαγράμματα ροής (Flowchart) τα οποία αναπτύχθηκαν στις αρχές της δεκαετίας του ΄20. Η έννοια της μεθοδολογίας ανάπτυξης λογισμικού έκανε την εμφάνισή της τη δεκαετία του ΄60. Η πιο παλιά τυποποιημένη μέθοδος για την ανάπτυξη λογισμικού είναι ο κύκλος ζωής ανάπτυξης συστημάτων (Software Development Life Cycle). Η βασική ιδέα ήταν να ακολουθείται ένας προσχεδιασμένος, δομημένος και τακτικός τρόπος ανάπτυξης πληροφοριακών συστημάτων. Το αρχικό αυτό μοντέλο -ο κύκλος ανάπτυξης συστημάτων- είχε δημιουργηθεί για να καλύψει τις ανάγκες της ανάπτυξης των λειτουργικών επιχειρησιακών συστημάτων ευρείας κλίμακας.

Έκτοτε έχουν προταθεί διάφορες μεθοδολογίες ανάπτυξης λογισμικού. Όπως αναφέραμε, κάθε μια από αυτές διαφέρει ως προς την προσέγγιση ανάπτυξης του λογισμικού. Ωστόσο υπάρχει μια ομάδα γενικότερων προσεγγίσεων οι οποίες συναντώνται σε διάφορες μεθοδολογίες και αποτελούν ίσως τη βάση αυτών. Οι προσεγγίσεις αυτές είναι οι εξής:

- Μοντέλο καταρράκτη (Waterfall), το οποίο είναι ουσιαστικά ένα γραμμικό μοντέλο.
- Μοντέλο της δημιουργίας πρωτοτύπου (Prototyping), αποτελεί ένα επαναληπτικό μοντέλο.
- Επαυξητικό -ή αυξητικό- μοντέλο (Incremental), συνδυάζει επαναληπτικότητα και γραμμικότητα.
- Σπειροειδές μοντέλο (Spiral), επαναληπτικό αλλά και γραμμικό μοντέλο.
- Γρήγορη πρωτοτυποποίηση εφαρμογής (Rapid Application Development RAD), επαναληπτικό μοντέλο.

Για κατανοήσουμε καλύτερα τις ευέλικτες μεθοδολογίες και κατ’ επέκταση την μεθοδολογία Scrum (επαναληπτική και αυξητική μεθοδολογία ανάπτυξης έργων), αντικείμενο το οποίο εξετάζουμε, περιγράφονται σύντομα οι πέντε αυτές βασικές προσεγγίσεις οι οποίες χρησιμοποιούνται ως πυλώνας των ευέλικτων πρακτικών. Οι

πλείστες από τις ευέλικτες μεθοδολογίες επιτρέπουν ή και επιβάλλουν την ανάπτυξη πρωτοτύπου σε κάποιο στάδιό τους καθώς επίσης μια μεθοδολογία για να ονομάζεται «ευέλικτη», πρέπει να ακολουθεί μέχρι κάποιο ορισμένο σημείο το επαυξητικό μοντέλο. Επίσης, καταγράφονται οι δυνάμεις και αδυναμίες της κάθε προσέγγισης ώστε να είναι ευκολότερο στη συνέχεια να υπάρξει η σύγκριση μεταξύ αυτών των παραδοσιακών προσεγγίσεων και της μεθοδολογίας Scrum. Εκτός αυτού, οι ευέλικτες προσεγγίσεις σκοπεύουν να εκμεταλλευτούν τα δυνατά στοιχεία των παραδοσιακών, και να καλύψουν τις αδυναμίες τους.

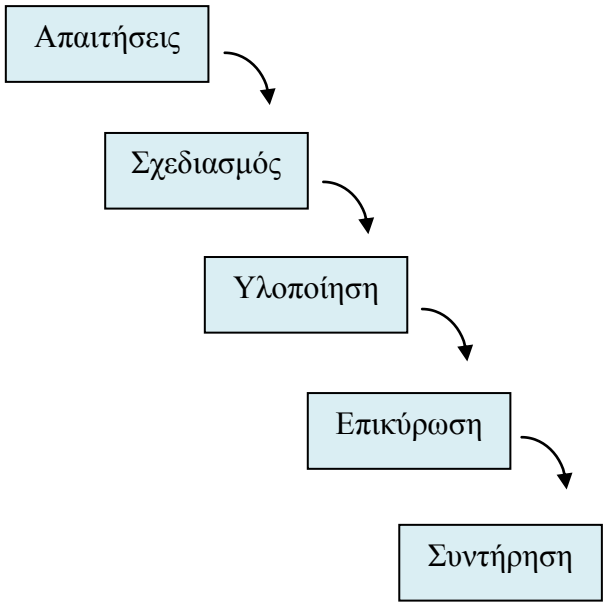
2.2 Το μοντέλο του καταρράκτη

Η πρώτη επίσημη αναφορά στο μοντέλο του καταρράκτη έγινε το 1970 από τον Winston W. Royce σε ένα άρθρο του να και στο άρθρο αυτό δεν αναφέρεται πουθενά ο όρος «καταρράκτης». Η κρίση λογισμικού το 1960, έφερε στην επιφάνεια την διαπίστωση πώς η διαδικασία παραγωγής λογισμικού, μοιάζει με άλλες τεχνολογικές διαδικασίες. Για τον λόγο αυτό το μοντέλο του καταρράκτη μοιάζει με άλλες τεχνολογικές διαδικασίες [Sommerville, 2011]. Το μοντέλο του καταρράκτη (Waterfall model) αποτελεί μια προσέγγιση που ακολουθεί διαδοχικά τα βήματα που την αποτελούν και κατά την οποία η ανάπτυξη γίνεται προς τα κάτω (όπως και σε ένα καταρράκτη, εξού και η ονομασία). Όπως παρουσιάζονται στο σχήμα 2.1, οι φάσεις από τις οποίες διέρχεται το μοντέλο αυτό είναι οι εξείς:

1. **Ανάλυση απαιτήσεων (Requirements):** Οι υπεύθυνοι ανάπτυξης του συστήματος μέσα από συνεντεύξεις και άλλες μεθόδους, και σε συνεργασία με τους μελλοντικούς χρήστες του συστήματος, προχωρούν στην καταγραφή των προδιαγραφών και των απαιτήσεων. Έπειτα ορίζονται με τέτοιο τρόπο ώστε οι πελάτες να κατανοήσουν πλήρως τι πρόκειται θα αναπτυχθεί.
2. **Σχεδιασμός (Design):** Σχεδιάζεται μια γενική αρχιτεκτονική του συστήματος.
3. **Υλοποίηση (Implementation):** Στη φάση αυτή υλοποιείται το λογισμικό περιλαμβανομένων των επιμέρους κομματιών στα οποία τυχόν να έχει διαχωριστεί το έργο.
4. **Επικύρωση (Verification):** Στο στάδιο αυτό γίνεται η επικύρωση του λογισμικού η οποία περιλαμβάνει τον έλεγχο ολόκληρου του συστήματος. Εξετάζεται κατά πόσο το σύστημα που αναπτύχθηκε καλύπτει όλες τις

προδιαγεγραμμένες προδιαγραφές και επίσης εξετάζεται αν έχει παραχθεί ένα σωστό σύστημα.

5. **Συντήρηση (Maintenance):** Η φάση αυτή εφαρμόζεται εφόσον το σύστημα έχει ολοκληρωθεί και παραδοθεί στους χρήστες. Περιλαμβάνει αλλαγές στο σύστημα που μπορεί να εντοπισθούν μετέπειτα κατά την χρήση του λογισμικού, εντοπισμό και διόρθωση λαθών, βελτίωση λειτουργιών ή ακόμα και πρόσθεση καινούριων αφού εντοπιστούν καινούριες απαιτήσεις. [Sommerville, 2011]



Σχήμα 2.1 Το μοντέλο του καταρράκτη

Οι βασικές αρχές του εν λόγω μοντέλου, ορίζουν ότι κάθε έργο διαχωρίζεται σε διαδοχικές φάσεις. Ωστόσο τυχόν επικάλυψη και αναθεωρήσεις ανάμεσα στις φάσεις, είναι αποδεκτές, συνήθως επιτρέπεται ανάδραση ανάμεσα σε δύο γειτονικά βήματα. Βασίζεται στην δημιουργία προδιαγραφών σε κάθε βήμα και δίνεται έμφαση στα στον προγραμματισμό, στα χρονικά όρια που έχουν προκαθοριστεί καθώς και τις ημερομηνίες, τον προκαθορισμένο προϋπολογισμό και ταυτόχρονα στην δημιουργία ενός ολοκληρωμένου συστήματος. Διεξάγεται αυστηρός έλεγχος καθ' όλη τη διάρκεια του έργου ο οποίος πραγματοποιείται με τη χρήση λεπτομερούς γραπτής τεκμηρίωσης, καθώς επίσης και μέσα από επίσημες αναθεωρήσεις που εγκρίνονται από τον πελάτη του προϊόντος.

2.2.1 Δυνάμεις/Αδυναμίες

Η μεθοδολογία αυτή είναι κατάλληλη για να υποστηρίξει ομάδες που δεν διαθέτουν ιδιαίτερη πείρα στην ανάπτυξη λογισμικού, ή ομάδες που αναδιοργανώνουν συνεχώς την σύνθεση τους. Η σειριακή ανάπτυξη, οι αυστηροί έλεγχοι ώστε να εξασφαλισθεί η επαρκής τεκμηρίωση και οι αναθεωρήσεις του σχεδίου, κατοχυρώνουν την ποιότητα, της αξιοπιστία και την μελλοντική προοπτική συντήρησης του λογισμικού. Επιπλέον, το μοντέλο αυτό, εξασφαλίζει την συνεχής ανάπτυξη των πληροφοριακών συστημάτων αφού η πρόοδος μετριέται. Τέλος, ο λεπτομερές προγραμματισμός και προϋπολογισμός που προηγείται, συμβάλλει στην καλύτερη δυνατή εκμετάλλευση των πόρων.

Από την άλλη κάποια στοιχεία που κατατάσσονται στα θετικά του μοντέλου αυτού, λειτουργούν παράλληλα και ως αδυναμίες. Το μοντέλο του καταρράκτη είναι άκαμπτο, κοστίζει αρκετά καθώς η ανάπτυξη του λογισμικού προχωρά με αργούς ρυθμούς ακριβώς εξαιτίας του εκτεταμένου προγραμματισμού και των αυστηρών ελέγχων. Η ροή των εργασιών προχωρά προς τα μπροστά με μόνη εξαίρεση την οπισθοδρόμηση σε προηγούμενο βήμα και γι' αυτό τον λόγο δεν υπάρχουν περιθώρια για επαναληπτική χρήση και επαναχρησιμοποίηση του συστήματος. Είναι γνωστό πώς η επιτυχία ενός συστήματος εξαρτάται σε μεγάλο βαθμό από την φάση της καταγραφής των απαιτήσεων και πώς οι απαιτήσεις αλλάζουν συνεχώς μέχρι να οριστεί η τελική μορφή τους. Συχνά όμως οι πελάτες δεν μπορούν να καθορίσουν σαφώς τις απαιτήσεις του συστήματος από την αρχή της εξέλιξης του συστήματος. Συνεπώς στο πρότυπο αυτό οι ασάφειες στις προδιαγραφές και οι απροσδόκητες ανάγκες σε υποδομή και υλικό, εντοπίζονται καθυστερημένα κατά την φάση της υλοποίησης. Οι ελλείψεις και ασάφειες δύναται να εντοπιστούν ακόμα και κατά την φάση του ελέγχου και δοκιμής του λογισμικού, γεγονός που καθιστά την διόρθωση του συστήματος δύσκολη έως και ακατόρθωτη πολλές φορές. Η απόπειρα διόρθωσης τέτοιων προβλημάτων, που εμφανίζονται δηλαδή αργά στον κύκλο ανάπτυξης του έργου, συνεπάγεται μεγάλο κόστος καθώς η ενσωμάτωση των αλλαγών στο σύστημα είναι εξαιρετικά δύσκολη. Επίσης λόγω του ότι παράγεται πολλή τεκμηρίωση, δαπανείται χρόνος για την ενημέρωσή της. Τέλος, οι γραπτές αναφορές είναι συχνά ακατανόητες για τους χρήστες, γεγονός που οδηγεί σε μη επαρκή αξιολόγηση των. Με τον τρόπο αυτό επικρατεί ένα χάσμα μεταξύ μηχανικών λογισμικού και χρηστών καθώς οι χρήστες

συμμετέχουν μόνο στην αρχή της διαδικασίας και βλέπουν το τελικό προϊόν πολύ αργά στη διάρκεια του έργου.

2.3 Το μοντέλο της δημιουργίας πρωτοτύπου

Η μεθοδολογία της δημιουργίας πρωτοτύπου (Prototyping model) είναι ένα μοντέλο ανάπτυξης λογισμικού, κατά το οποίο κατασκευάζονται ημιτελής εκδόσεις του πληροφοριακού συστήματος που πρόκειται να αναπτυχθεί. Κάθε μια από αυτές τις εκδόσεις ονομάζεται πρωτότυπο, και περιέχει μερικά χαρακτηριστικά και λειτουργίες του λογισμικού και ενδέχεται να είναι πολύ διαφορετικό από το τελικό πρόγραμμα.

Ο λόγος για τον οποίο εφαρμόζεται η μέθοδος αυτή είναι να επιτρέπει στους χρήστες του συστήματος να αξιολογήσουν τις προτάσεις των μηχανικών λογισμικού, δοκιμάζοντας έμπρακτα ένα πρωτότυπο του προϊόντος αντί του να πρέπει να κατανοήσουν και να αξιολογήσουν προδιαγραφές και περιγραφές του συστήματος. Η διαμόρφωση ενός πρωτοτύπου επιτρέπει έτσι στους χρήστες να εντοπίσουν τυχόν ασάφειες στις απαιτήσεις του λογισμικού και με τον τρόπο αυτό η «δοκιμασία του πρωτότυπου» μπορεί να αποτελέσει έναυσμα για μια καλή επικοινωνία μεταξύ υπευθύνων ανάπτυξης και πελατών.

Το πρωτότυπο που αναπτύσσεται είναι μια λειτουργική έκδοση του πληροφοριακού συστήματος και θεωρείται προκαταρτικό μοντέλο. Το πρωτότυπο θα αλλάζει συνεχώς και θα βελτιώνεται μέχρι να τύχει πλήρης αποδοχής από τους πελάτες και συνεπώς να ικανοποιούνται οι απαιτήσεις τους. Όταν συμβεί αυτό τότε το πρωτότυπο μπορεί να μετατραπεί σε ένα τελικό παραγωγικό σύστημα.

Η μέθοδος δημιουργίας πρωτοτύπου είναι μια επαναληπτική διαδικασία και όπως όλα τα μοντέλα, περιέχει κάποιες βασικές φάσεις ανάπτυξης του λογισμικού. Η δημιουργία πρωτοτύπου είναι μια προγραμματισμένη επαναληπτική διαδικασία όπου, κάθε νέα έκδοση συναντά ακριβέστερα τις απαιτήσεις των χρηστών. Τα βήματα της διαδικασίας φαίνονται στο σχήμα 2.2 και αποτελείται από τα εξής:

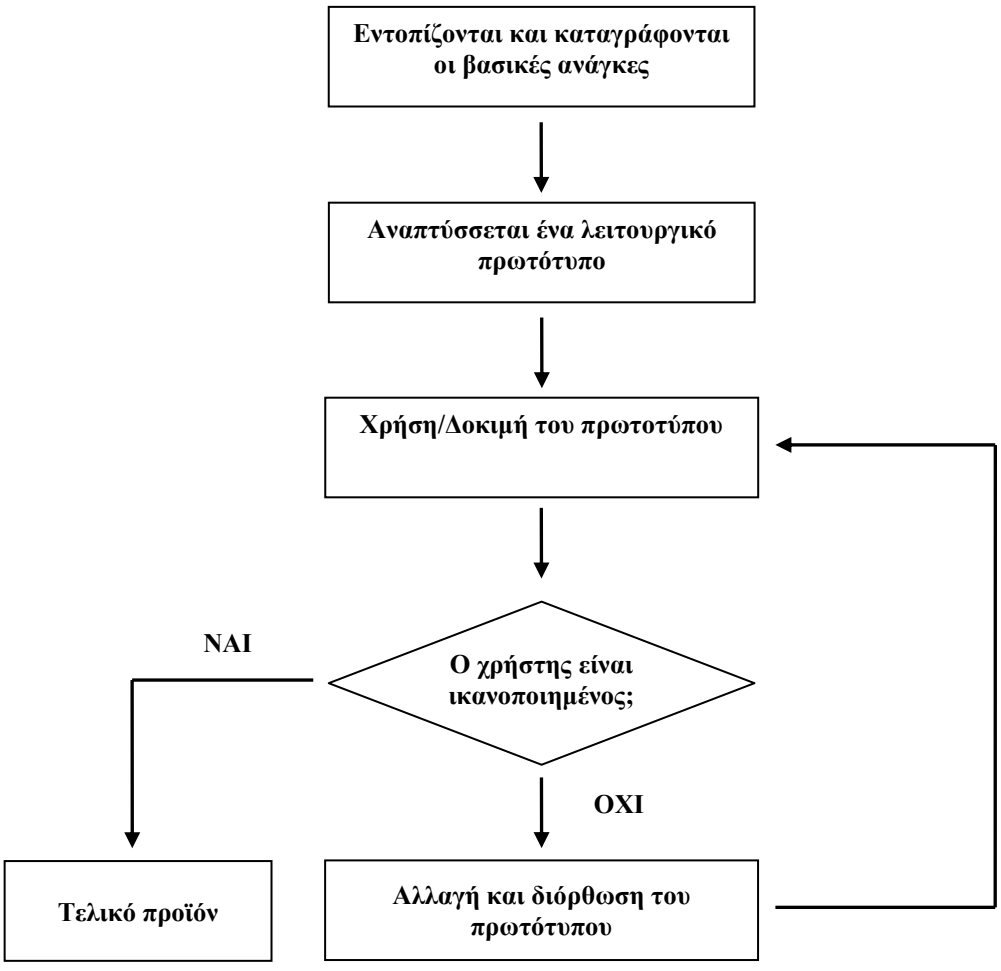
Βήμα 1: Εντοπίζονται και καταγράφονται οι βασικές ανάγκες του χρήστη.

Βήμα 2: Αναπτύσσεται γρήγορα ένα λειτουργικό πρωτότυπο.

Βήμα 3: Δοκιμή του πρωτοτύπου. Οι χρήστες ενθαρρύνονται να χρησιμοποιήσουν το σύστημα προκειμένου να εντοπίσουν ασάφειες και να καθορίζουν ποιες ανάγκες τους δεν ικανοποιούνται απόλυτα.

Βήμα 4: Διορθώνεται και βελτιώνεται το πρωτότυπο. Οι σχεδιαστές αλλάζουν και βελτιώνουν το πρωτότυπο βασισμένοι στις υποδείξεις των χρηστών. Μετά την διόρθωση η διαδικασία επιστρέφει στο βήμα 3 και πραγματοποιείται συνεχώς αυτή η επανάληψη μέχρι ο χρήστης να ικανοποιηθεί πλήρως.

Όταν πια δεν θα χρειάζονται άλλες αλλαγές και ολοκληρώνονται έτσι οι επαναλήψεις, τότε το πρωτότυπο μετατρέπεται στο τελικό προϊόν περιέχοντας όλες τις προδιαγραφές του συστήματος.



Σχήμα 2.2 Μοντέλο δημιουργίας πρωτοτύπου

Το μοντέλο δημιουργίας πρωτοτύπων, θεωρείται από πολλούς σαν μια ελλιπές διαδικασία ανάπτυξης. Αποτελεί μια προσέγγιση χειρισμού ορισμένων τμημάτων μιας πιο παραδοσιακής μεθοδολογίας ανάπτυξης, όπως είναι το επαυξητικό ή το σπειροειδές μοντέλο. Στην τεχνολογία λογισμικού, με την χρήση της μεθόδου αυτής, καταβάλλεται προσπάθεια ώστε να μειωθεί ο κίνδυνος ένα έργο να αποτύχει. Αυτό επιτυγχάνεται με το σπάσιμο του έργου σε μικρότερα επιμέρους έργα-κομμάτια καθώς παρέχεται μεγαλύτερη προσαρμογή στην αλλαγή κατά την ανάπτυξη του έργου. Επίσης, ο χρήστης συμμετέχει σε ολόκληρη την διαδικασία και με τον τρόπο αυτό υπάρχουν περισσότερες πιθανότητες αποδοχής του τελικού παραδοτέου. Τα πρωτότυπα παραμένουν μικρά σε μέγεθος ανάπτυξης, ώστε να εντοπίζονται αδυναμίες και να μπορούν εύκολα και επαναληπτικά να τροποποιηθούν για να καλύψουν τις ανάγκες των χρηστών. Στις πλείστες περιπτώσεις τα πρωτότυπα που αναπτύσσονται, αναμένεται να τύχουν απόρριψης από τους χρήστες. Ωστόσο σε ορισμένες περιπτώσεις ένα πρωτότυπο πιθανό να εξελιχθεί και να αποτελέσει ολόκληρο το σύστημα. Η μεθοδολογία αυτή προϋποθέτει την κατανόηση του εξαιρετικής σημασίας επιχειρησιακού προβλήματος του εκάστοτε οργανισμού για το οποίο αναπτύσσεται η λύση.

2.3.1 Δυνάμεις/Αδυναμίες

Το μοντέλο αυτό παρέχει την δυνατότητα σχηματισμού άποψης για το σύστημα νωρίς κατά την διάρκεια του έργου. Το μοντέλο δημιουργίας πρωτοτύπων χρησιμοποιείται ιδιαίτερα στην ανάπτυξη συστημάτων όπου οι προδιαγραφές δεν είναι πλήρως προδιαγεγραμμένες και υπάρχουν ασάφειες ως προς τις απαιτήσεις των χρηστών. Παράλληλα υπάρχει η δυνατότητα να εκμεταλλευτεί η γνώση που συλλέχτηκε από το πρωτότυπο ώστε να αναπτυχθεί ένα «πλήρες» τελικό προϊόν. Επιπλέον, είναι δυνατό κατά τις πρώτες φάσεις του κύκλου ζωής του λογισμικού να προσδιοριστούν οι ασάφειες και τα στοιχεία εκείνα που προκαλούν σύγχυση. Το συγκεκριμένο μοντέλο θα μπορούσαμε να πούμε ότι είναι συγκριτικά καινοτόμο.

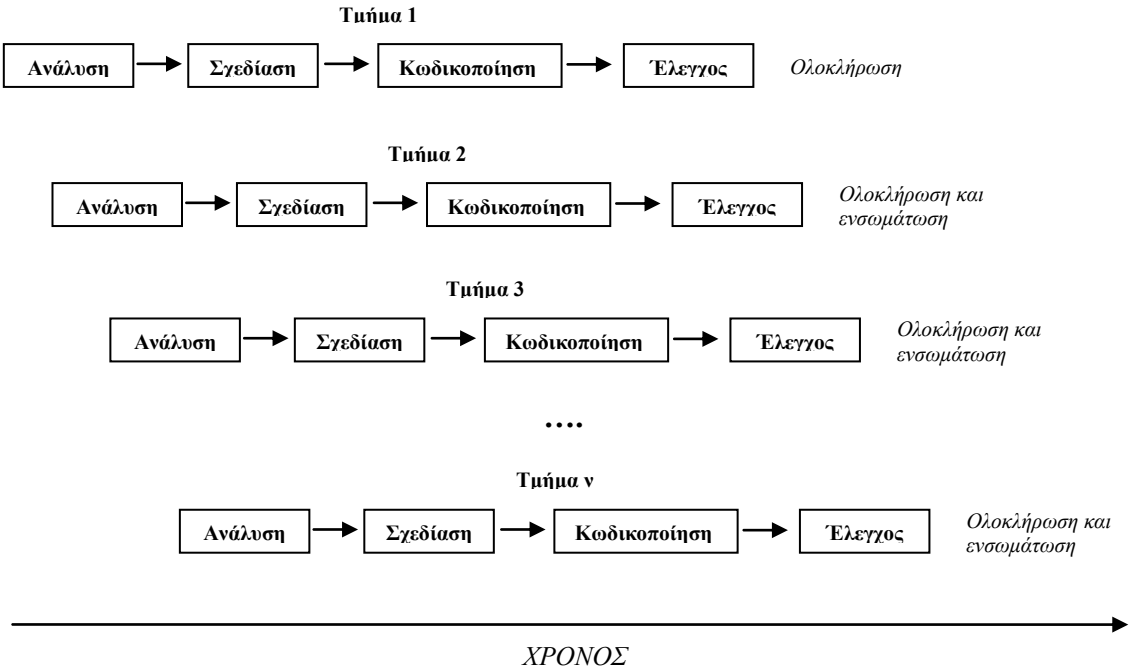
Εντούτοις, η διαδικασία ανάπτυξης του πρωτοτύπου καθώς και η τελική μορφή που θα πάρει, δεν είναι αυστηρώς καθορισμένη διαδικασία με αποτέλεσμα να οδηγεί σε μια σειρά προβλημάτων. Η τυχόν ανεπάρκεια στην ανάλυση του υφιστάμενου

προβλήματος ενδέχεται να οδηγήσει σε κάλυψη μόνο ορισμένων επιφανειακών αναγκών του συστήματος. Στο μοντέλο αυτό, οι προδιαγραφές δύναται να αλλάζουν σε τακτά χρονικά διαστήματα και σε μεγάλο βαθμό. Οι υπεύθυνοι για την ανάπτυξη του συστήματος μπορούν να προχωρούν στην παραγωγή του πρωτοτύπου σχετικά γρήγορα, χωρίς όμως να αναλύουν ουσιαστικά τις ανάγκες των χρηστών με αποτέλεσμα να μην καταλήγουν ποτέ σε ένα άκαμπτο προϊόν γεγονός που περιορίζει το σύστημα ώστε να μπορεί να επεκταθεί στον μέλλον. Οι σχεδιαστές μπορεί να αγνοήσουν την τεκμηρίωση και έτσι το τελικό προϊόν αδυνατεί να αιτιολογηθεί πλήρως. Η ανάπτυξη ενός συστήματος χρησιμοποιώντας το μοντέλο πρωτοτύπου, εύκολα μπορεί να οδηγήσει σε ένα ημιτελές και όχι εύχρηστο σύστημα, γιατί η έλλειψη εμπειρίας εκ μέρους των σχεδιαστών μπορεί να οδηγήσει στην αντικατάσταση ενός κακώς σχεδιασμένου πρωτοτύπου με το τελικό προϊόν. Με τον τρόπο αυτό δημιουργείται ένα σύστημα που υστερεί λειτουργικότητας και χωρίς να εκτιμώνται όλες οι προδιαγραφές. Το αρχικό πρωτότυπο συχνά αναπτύσσεται ώστε να μπορεί να εφαρμοστεί στο περιβάλλον εργασίας του πελάτη, ώστε δηλαδή να μπορεί να χρησιμοποιηθεί. Η προσπάθεια να αναπτύσσεται αναδρομικά ένα προϊόν μπορεί να αποδειχτεί προβληματική. Ένα πρωτότυπο μπορεί να δώσει προσδοκίες που δεν ανταποκρίνονται στην πραγματικότητα για τον λόγο ότι οι χρήστες λανθασμένα πολλές φορές θεωρούν ότι το σύστημα είναι ολοκληρωμένο καθώς δεν ισχύει αυτό ή λανθασμένα αντιλαμβάνονται την λειτουργία του συστήματος με αποτέλεσμα να μην μπορούν περιγράψουν επαρκώς τις ανάγκες τους. Οι επαναλήψεις που προηγούνται της τελικής μορφής του συστήματος, νοείται ότι κοστίζουν και συνεπώς ο προϋπολογισμός του έργου δεν δύναται να καθοριστεί επακριβώς εξ’ αρχής. Η μέθοδος αυτή ενδείκνυται για μεγάλα προγράμματα που εμπερικλείουν μηδενική ανοχή σε σφάλματα και οι απαιτήσεις είναι πολύ καλά δομημένες και προσδιορισμένες (όπως συστήματα τραπεζικών συναλλαγών), καθώς δεν συνίσταται για μικρά συστήματα γιατί δεν θα είναι σε θέση να δικαιολογήσουν τον προστιθέμενο χρόνο και κόστος. Η επιτυχημένη δημιουργία πρωτοτύπων προϋποθέτει την ύπαρξη διαδικασιών και μηχανισμών για τον καθορισμό των προσδοκιών, την εκχώρηση πόρων, τον εντοπισμό προβλημάτων, και την αξιολόγηση της προόδου.

2.4 Το επαυξητικό μοντέλο

Έχουν προταθεί διάφορα μοντέλα που συνδυάζουν γραμμικές και επαναληπτικές μεθοδολογίες ανάπτυξης λογισμικού με αρχικό στόχο κάθε ενός από αυτά να μειωθεί όσο το δυνατό περισσότερο ο υπαρκτός κίνδυνος του συστήματος με το σπάσιμο του σε μικρότερα μέρη και την παροχή ευελιξίας στην αλλαγή κατά τον κύκλο ανάπτυξης του έργου [Pfleeger, 2003].

Το επαυξητικό μοντέλο ή μοντέλο λειτουργικής επαύξησης (Incremental model) συνδυάζει την σειριακή ανάπτυξη του μοντέλου του καταρράκτη και την επαναληπτική ανάπτυξη της πρωτοτυποποίησης. Κεντρική ιδέα του μοντέλου αυτού είναι η κατάτμηση του λογισμικού που πρόκειται να αναπτυχθεί, σε τμήματα τα οποία αναπτύσσονται ανεξάρτητα. Το κάθε κομμάτι ακολουθεί τη μέθοδο του καταρράκτη όπως παρουσιάζεται στο σχήμα 2.3. Κατά το αρχικό στάδιο της ανάλυσης και σχεδίασης προσδιορίζονται τα τμήματα όπου θα χωριστεί η εφαρμογή, η παραγωγή των οποίων γίνεται ανεξάρτητα και παράλληλα. Όταν ολοκληρωθεί η ανάπτυξη ενός τμήματος τότε γίνεται η ενσωμάτωσή του στο ευρύτερο σύστημα, διαδικασία που δικαιολογεί την ονομασία «επαυξητικό μοντέλο».



Σχήμα 2.3 Το επαυξητικό μοντέλο

2.4.1 Δυνάμεις/Αδυναμίες

Η δυνατότητα εκμετάλλευσης της γνώσης που συλλέγεται σε μια επαύξηση στις επόμενες, συγκαταλέγεται στα δυνατά σημεία του μοντέλου αυτού. Επίσης γίνεται συνεχής έλεγχος κατά την διάρκεια της ανάπτυξης του συστήματος μέσα από την γραπτή τεκμηρίωση, την επίσημη αναθεώρηση και την έγκριση του χρήστη. Τα δεδομένα αυτά μπορούν να δοθούν στους σχεδιαστές και σε όσους εμπλέκονται στην ανάπτυξη του συστήματος. Στο μοντέλο αυτό περιορίζεται ο κίνδυνος να παρουσιαστούν σφάλματα από την αρχή κιόλας του κύκλου ζωής. Ένα ακόμη στοιχείο το οποίο περιλαμβάνεται στα πλεονεκτήματα του μοντέλου αυτού, είναι το γεγονός ότι παράγονται και παραδίδονται λειτουργικά τμήματα του συστήματος τα οποία είναι σταδιακά πληρέστερα. Τέλος, η συνεχής και σταδιακή χρήση των επαυξητικών τμημάτων του λογισμικού, παρέχει την δυνατότητα να εντοπιστούν πιο εύκολα τυχόν προβλήματα και να γίνουν οι απαραίτητες προσαρμογές ώστε να μην προλάβει να επηρεαστεί αρνητικά η λειτουργία του οργανισμού που χρησιμοποιεί το προϊόν. Πέρα αυτών, κύριο πλεονέκτημα της ιδέας αυτής είναι ότι η ανάπτυξη γίνεται σε μικρότερο χρονικό διάστημα λόγω της παράλληλης ανάπτυξης.

Κατά την εφαρμογή της μεθόδου καταρράκτη σε μια επαύξηση του συστήματος, ενδέχεται να παρατηρηθεί ασάφεια στην κατανόηση του επιχειρησιακού προβλήματος και των προδιαγραφών της εφαρμογής. Επίσης, όπως αναφέραμε, το σύστημα σπάει σε επιμέρους κομμάτια τα οποία αναπτύσσονται ανεξάρτητα και ίσως παράλληλα. Για τον λόγο αυτό θα πρέπει να οριστούν σαφώς οι αλληλεξαρτήσεις της εφαρμογής και η σειρά ανάπτυξης πολύ νωρίς πριν την διαδικασία της κατάτμησης καθώς υπάρχει ο κίνδυνος ένα τμήμα του συστήματος να απαιτεί την ύπαρξη ενός άλλου ώστε να μπορεί να λειτουργήσει. Τέλος σε περίπτωση αλλαγής των λειτουργικών απαιτήσεων της εφαρμογής κατά την δοκιμή ενός επαυξητικού τμήματος, τότε ενδέχεται οι αλλαγές να επηρεάζουν όλα τα τμήματα που έχουν αναπτυχθεί μέχρι στιγμής.

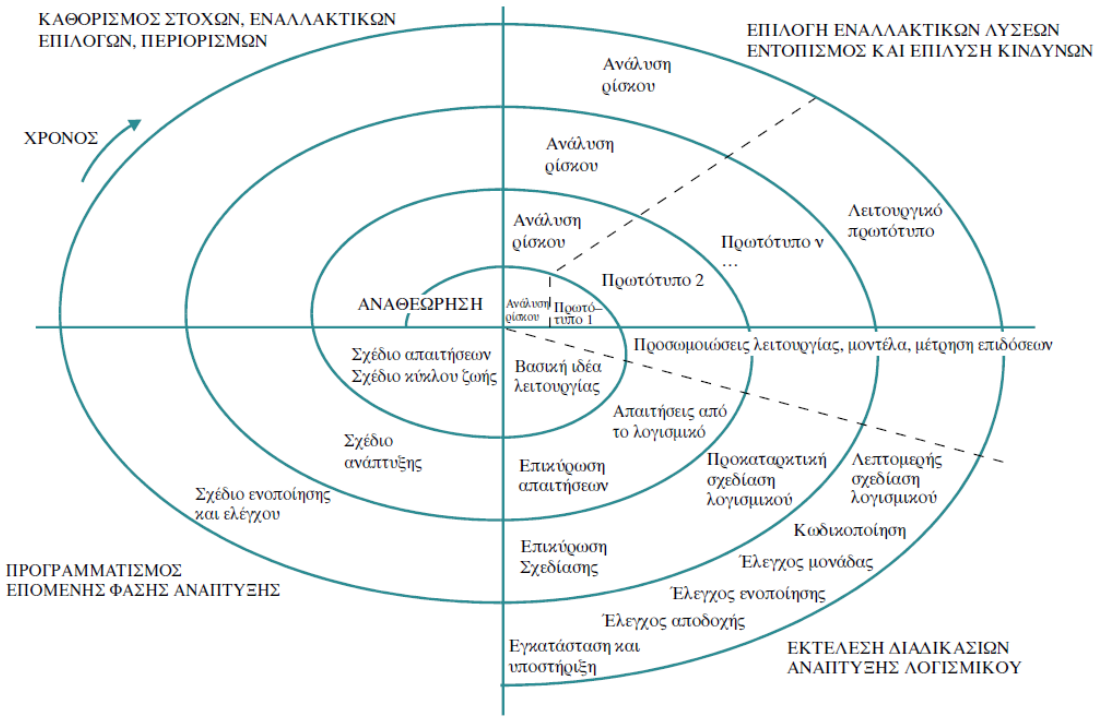
2.5 Το σπειροειδές μοντέλο

Το σπειροειδές μοντέλο θεσπίστηκε από τον Barry Boehm το 1988. Η μεθοδολογία αυτή δεν ήταν η πρώτη που εφάρμοζε την επαναληπτική ανάπτυξη αλλά ήταν η πρώτη που εξηγούσε γιατί η επανάληψη είναι σημαντική.

Κάθε στάδιο ξεκινούσε με ένα στόχο και ολοκληρωνόταν με την συμμετοχή του χρήστη ο οποίος αναθεωρούσε την εξέλιξη του έργου. Η ανάλυση και η εφαρμογή παρουσιάζονται σε κάθε φάση του κύκλου ζωής. Το σπειροειδές μοντέλο πήρε το όνομά του από την γραφική απεικόνισή του, όπως παρουσιάζεται στο Σχήμα 2.4. Στο μοντέλο αυτό γενικεύονται οι μέθοδοι της πρωτοτυποποίησης και της επαύξησης, με σημαντικά ωστόσο νέα στοιχεία. Τα βασικά χαρακτηριστικά του σπειροειδούς μοντέλου είναι τα εξής (Boehm, 1998):

- Δίνεται έμφαση στην αξιολόγηση του κινδύνου και στην προσπάθεια να ελαχιστοποιηθεί. Αυτό προκύπτει με την κατάτμηση του έργου σε μικρότερα υποέργα και έτσι υπάρχει καλύτερη προοπτική αλλαγής ενός τμήματος κατά την ανάπτυξη του λογισμικού. Για τον λόγο αυτό η ανάπτυξη του λογισμικού χωρίζεται σε πολλούς κύκλους, σε κάθε ένα από τους οποίους προστίθενται καινούρια λειτουργικά χαρακτηριστικά στο σύστημα.
- Σε κάθε κύκλο το έργο εξελίσσεται ακολουθώντας τα βήματα, για κάθε κομμάτι του έργου και για κάθε ένα από τα επίπεδα επεξεργασίας του.
- Κάθε πέρασμα πάνω στον κύκλο, διέρχεται από τέσσερις βασικές φάσεις (όπως φαίνεται στο Σχήμα 2.4):
    1. Καθορίζονται οι στόχοι, οι εναλλακτικές λύσεις και οι περιορισμοί της επανάληψης (Determine objectives): Καθορίζονται οι εργασίες κάθε επανάληψης καθώς και οι περιορισμοί που προκύπτουν. Επίσης εντοπίζονται οι κίνδυνοι που υποκρύπτει η διαδικασία και προσδιορίζονται εναλλακτικές λύσεις εφόσον αυτό είναι εφικτό.
    2. Αξιολόγηση των εναλλακτικών λύσεων, εντοπίζονται και επιλύονται τυχόν κίνδυνοι (Identify and resolve the risks): Γίνεται ανάλυση των κινδύνων που καταγράφηκαν και εξετάζεται κάθε εναλλακτική λύση. Στο σημείο αυτό αποφασίζεται αν θα συνεχίσει ή όχι η διαδικασία ανάπτυξης, η μεθοδολογία που θα εφαρμοστεί για την επανάληψη αυτή, αν θα παραχθεί ή όχι πρωτότυπο κτλ.
    3. Αναπτύσσονται και ελέγχονται τα κομμάτια λογισμικού που προέκυψαν από την επανάληψη (Development and test): Εφαρμόζεται η μέθοδος που επιλέγηκε προηγουμένως.

4. Προγραμμάτισε την επόμενη επανάληψη (Plan the next iteration): Αφού γίνει η επικύρωση των αποτελεσμάτων, του τμήματος λογισμικού που αναπτύχθηκε, τότε προγραμματίζεται η συνέχιση της ανάπτυξης.



Σχήμα 2.4 Το σπειροειδές μοντέλο [Βεσκούκης, 2000]

2.5.1 Δυνάμεις/Αδυναμίες

Η εφαρμογή της μεθοδολογίας αυτής αυξάνει τις πιθανότητες ώστε να μην παρουσιαστούν σημαντικοί κίνδυνοι. Σε κάθε επανάληψη μπορεί να επιλεγεί το κατάλληλο μοντέλο για την ανάπτυξη ενός τμήματος του λογισμικού. Έτσι, μπορεί να εφαρμόσει την μέθοδο του καταρράκτη, την πρωτοτυποποίηση, το επαυξητικό μοντέλο ή ακόμα και συνδυασμό αυτών ως μεμονωμένες περιπτώσεις επαναλήψεων και βάση του βαθμού του κινδύνου που αντιμετωπίζει το σύστημα την δεδομένη χρονική στιγμή. Για παράδειγμα σε ένα έργο όπου οι απαιτήσεις των πελατών είναι ξεκάθαρες αλλά υπάρχει αβεβαιότητα ως προς τον προϋπολογισμό ή τα χρονικά περιθώρια που θα τεθούν, τότε θα εφαρμοζόταν το μοντέλο του καταρράκτη κατά τις επαναλήψεις του. Αν οι κίνδυνοι αυτοί αντιστραφούν, τότε θα εφαρμοζόταν μια επαναληπτική μεθοδολογία.

Ωστόσο, η σπειροειδής μεθοδολογία είναι δύσκολο να ορίσει το μοντέλο ή τον συνδυασμό των μοντέλων που θα εφαρμοστούν κατά την διάρκεια μιας επανάληψης πάνω στη σπείρα. Η επιλογή αυτή είναι προσαρμοσμένη σε κάθε επανάληψη ξεχωριστά με βάση τις απαιτήσεις και τις δυσκολίες που θα παρουσιαστούν στο τμήμα αυτό του συστήματος. Έτσι δεν υπάρχει μεγάλη προοπτική επαναχρησιμοποίησης της ίδιας διαδικασίας. Συνήθως απαιτείται ένας επαγγελματίας με μεγάλη πείρα ώστε να δώσει λύση στο πώς θα εφαρμοστεί το πρότυπο αυτό σε κάθε περίπτωση. Επίσης, στα μειονεκτήματα του μοντέλου αυτού κατατάσσεται το γεγονός ότι δεν υπάρχουν όρια. Δεν υπάρχουν, δηλαδή, προθεσμίες και έτσι οι επαναλήψεις μπορεί να συνεχίζονται με ορατό τον κίνδυνο να ξεπεραστούν ο προϋπολογισμός και τα χρονικά όρια. Τέλος, η ομάδα θα πρέπει να είναι εξαιρετικά ικανή στην ανάλυση ρίσκου ώστε το σπειροειδές μοντέλο να μπορεί να επιφέρει θετικά αποτελέσματα.

2.6 Η γρήγορη ανάπτυξη εφαρμογής

Το μοντέλο της γρήγορης ανάπτυξης εφαρμογής (Rapid Application Development, RAD) αποτελεί μια προσέγγιση ανάπτυξης λογισμικού η οποία εφαρμόζει ουσιαστικά το επαυξητικό μοντέλο και την πρωτοτυποποίηση. Η ορολογία της γρήγορης ανάπτυξης εφαρμογής αρχικά εμφανίστηκε για να προσδιορίσει μια μέθοδο ανάπτυξης συστημάτων που επινόησε την δεκαετία του ΄80 ο James Martin στην IBM. Αργότερα η μέθοδος αυτή επισημοποιήθηκε μέσα από την έκδοση ενός βιβλίου του την δεκαετία του ΄90.

Τα βασικά χαρακτηριστικά του μοντέλου αυτού είναι τα εξής:

- Βασική αρχή είναι η γρήγορη κατασκευή και παράδοση ενός λογισμικού υψηλής ποιότητας με όσο το δυνατό χαμηλότερη οικονομική επιβάρυνση.
- Μειώνεται ο έμφυτος κίνδυνος αποτυχίας του έργου, με την κατάτμηση του σε μικρότερα υποέργα. Έτσι παρέχεται μεγαλύτερη ευελιξία για αλλαγές ενός τμήματος κατά τον κύκλο ζωής του προϊόντος.
- Το υψηλής ποιότητας προϊόν προκύπτει μέσα από την χρησιμοποίηση της επαναληπτικής μεθόδου της πρωτοτυποποίησης (σε όλα τα στάδια της

ανάπτυξης), της εμπλοκής των χρηστών και της χρήσης αυτοματοποιημένων εργαλείων ανάπτυξης.

- Η διαδικασία επικεντρώνεται στην εκπλήρωση των επιχειρησιακών αναγκών.
- Δίνεται έμφαση στην ανάπτυξη, την τήρηση του προϋπολογισμού και των χρονικών ορίων που έχουν προκαθοριστεί. Αν σε κάποια φάση παρατηρηθεί πώς η διαδικασία ανάπτυξης εξέρχεται των πιο πάνω ορίων, τότε οι απαιτήσεις αναπροσαρμόζονται κατάλληλα ώστε να μην παραβιαστεί το χρονικό περιθώριο καθώς δεν επιτρέπεται η αναπροσαρμογή του.
- Το λογισμικό αναπτύσσεται επαναληπτικά καθώς παράγεται επαρκής τεκμηρίωση, γεγονός που ευκολύνει την φάση της συντήρησης και την μελλοντική επέκταση.
- Στη μεθοδολογία αυτή είναι δυνατό να χρησιμοποιηθούν τεχνικές ανάλυσης και σχεδίασης συστημάτων.

2.6.1 Δυνάμεις/Αδυναμίες

Συγκριτικά με τα υπόλοιπα μοντέλα, η γρήγορη ανάπτυξη εφαρμογής πλεονεκτεί στο γεγονός ότι μπορεί να παράγει πιο γρήγορα ένα σύστημα που να είναι λειτουργικά άρτιο, δίνοντας έμφαση στις επιχειρησιακές ανάγκες. Η χρήση αυτής της μεθόδου συνήθως έχει χαμηλότερο χρηματικό αντίκτυπο και επιπλέον η συμμετοχή των χρηστών στην όλη διαδικασία ανάπτυξης είναι εντονότερη από τα προηγούμενα μοντέλα που περιγράφηκαν. Για τον λόγο αυτό οι χρήστες νιώθουν ότι το προϊόν τους ανήκει ενώ οι υπεύθυνοι για την ανάπτυξη του συστήματος ικανοποιούνται περισσότερο αφού μπορούν να αναπτύξουν γρήγορα ένα λειτουργικό προϊόν. Το μοντέλο εστιάζει στις πιο σημαντικές απαιτήσεις των χρηστών και υπάρχει η δυνατότητα να αναπροσαρμοστούν γρήγορα οι απαιτήσεις από τους ίδιους τους χρήστες. Γίνεται προσπάθεια ώστε να μην ξοδεύεται περιττός χρόνος και χρήμα.

Βέβαια, δεν μπορείς να έχεις μεγάλες απαιτήσεις από ένα σύστημα που αναπτύσσεται γρήγορα με το δυνατό χαμηλότερο κόστος. Εξαιτίας του αυστηρού ελέγχου που διεξάγεται στα χρονικά περιθώρια, πολλές φορές ορισμένα χαρακτηριστικά αναβάλλονται για επόμενες επαναλήψεις. Γι’ αυτό ενδέχεται το σύστημα, τελικά, να

μην μπορεί να ενταχθεί στον οργανισμό για τον οποίο προορίζεται λόγω της παράκαμψης ορισμένων σημαντικών απαιτήσεων.

2.7 Χαρακτηριστικά παραδοσιακών μεθοδολογιών

Οι παραδοσιακές μεθοδολογίες υπάρχουν και χρησιμοποιούνται εδώ και πολύ καιρό. Επιβάλλουν μια πειθαρχημένη διαδικασία κατά την ανάπτυξη λογισμικού με στόχο να καταστήσουν την ανάπτυξη λογισμικού πιο αποτελεσματική. Δεν παρουσιάζονται ως πετυχημένες προσεγγίσεις παρόλο που κάποιες από αυτές είναι δημοφιλής. Η πιο γνωστή παραδοσιακή μέθοδος είναι το μοντέλο του καταρράκτη, το οποίο χρησιμοποιείται ακόμα από μικρές και μεγάλες εταιρίες. Αν και οι ευέλικτες μεθοδολογίες (Agile methodologies) τείνουν να χρησιμοποιούνται περισσότερο, μια πρόσφατη έρευνα ανάμεσα σε Ευρωπαϊκές εταιρίες ανάπτυξης λογισμικού δηλώνει ότι το παραδοσιακό μοντέλο του καταρράκτη θα διατηρεί ακόμη ψηλή θέση ανάμεσα στις κυρίαρχες αναπτυξιακές στρατηγικές [CA Technologies, 2010]. Κατά καιρούς έχουν προταθεί πολλές παραλλαγές του μοντέλου αυτού. Ο Fowler επικρίνει αυτές τις μεθοδολογίες ως προς την γραφειοκρατία που τις χαρακτηρίζει, επισημαίνοντας πώς υπάρχουν αρκετά στοιχεία που πρέπει να εφαρμοστούν και στο τέλος χάνεται το νόημα της ανάπτυξης του ίδιου του συστήματος [Fowler, 2005].

Οι παραδοσιακές μεθοδολογίες έχουν την τάση να γίνεται πρώτα ένας καλός λεπτομερής προγραμματισμός και αυτό διαρκεί για πολύ καιρό. Όπως είδαμε στις προσεγγίσεις που μελετήσαμε στο κεφάλαιο αυτό, δίνεται έμφαση στον σχεδιασμό του συστήματος, στις ανάγκες αυτού και στον τρόπο που θα καλυφθούν οι ανάγκες αυτές αποτελεσματικά. Ο σχεδιασμός αυτός γίνεται συνήθως από μια ομάδα ατόμων και στη συνέχεια τα αποτελέσματα παραδίδονται σε μια άλλη ομάδα που είναι υπεύθυνη για την ανάπτυξη του συστήματος. Προβλέπεται ότι η διαδικασία ανάπτυξης του συστήματος θα ακολουθήσει τα σχέδια αυτά. Ο σχεδιασμός καθορίζει τον προϋπολογισμό, τις ανάγκες που θα πρέπει να πληρεί το σύστημα καθώς επίσης ορίζει το χρονοδιάγραμμα που πρέπει να ακολουθηθεί.

Τα παραδοσιακά μοντέλα ανάπτυξης λογισμικού, αντιμετωπίζουν τις απαιτήσεις ως το βασικό κομμάτι της τεκμηρίωσης (documentation). Η κύρια διαδικασία στις

παραδοσιακές μεθοδολογίες είναι η συλλογή των απαιτήσεων του πελάτη πριν από την σύνταξη οποιουδήποτε κώδικα. Αυτό έχει σαν αποτέλεσμα να μην υπάρχει ευελιξία ως προς το ενδεχόμενο της αλλαγής του συστήματος εάν αλλάξουν ή επαναξιολογηθούν οι απαιτήσεις. Την αδυναμία αυτή προσπαθεί να καλύψει η ευέλικτη ανάπτυξη λογισμικού. Η ευέλικτη οικογένεια μεθόδων ανάπτυξης λογισμικού, γεννήθηκαν με την πεποίθηση ότι μια προσέγγιση είναι περισσότερο προσαρμοσμένη στην ανθρώπινη πραγματικότητα. Οι ευέλικτες μέθοδοι εισάγουν μία διαφορετική προσέγγιση στον τρόπο ανάπτυξης του λογισμικού. Η προσέγγιση αυτή αφορά την στροφή από τον παραδοσιακό τρόπο, τον προσανατολισμένο στις διαδικασίες και την πλήρη τεκμηρίωση, στον ευέλικτο τρόπο, που είναι προσανατολισμένος στον άνθρωπο και την λιγότερη τεκμηρίωση. Στο επόμενο κεφάλαιο θα μελετήσουμε την ευέλικτη ανάπτυξη λογισμικού.

Κεφάλαιο 3

Ευέλικτη Ανάπτυξη Λογισμικού

3.1 Εισαγωγή

Οι ευέλικτες μεθοδολογίες (Agile methodologies) αποτελούν ένα σχετικά πρόσφατο κεφάλαιο στην ανάπτυξη λογισμικού. Ο όρος «ευέλικτες» χρησιμοποιείται για να συγκεντρώσει και να περιγράψει ένα αριθμό από μεθοδολογίες στις οποίες εφαρμόζονται κάποιοι κοινοί «κανόνες». Αυτές οι μέθοδοι δεν αναπτύχθηκαν σε θεωρητικό επίπεδο αλλά βγήκαν στην επιφάνεια για να εξυπηρετήσουν εταιρικές ανάγκες.

Οι επιχειρήσεις δείχνουν προτίμηση στις ευέλικτες μεθοδολογίες έναντι των παραδοσιακών μεθόδων για τον λόγο ότι τις θεωρούν πιο «ευκίνητες». Παρατηρήθηκε πώς οι επαγγελματίες ανάπτυξης λογισμικού προσπαθούν συνεχώς να αυξήσουν την παραγωγικότητα ενώ παράλληλα καταβάλλεται προσπάθεια η ποιότητα να διατηρείται

σε ψηλά επίπεδα [Bowers, 2002]. Αυτή η διαπίστωση ίσως αποτελεί τον σημαντικότερο λόγο για τον οποίο οι εταιρίες οδηγούνται στην αναζήτηση νέων μεθοδολογιών και τρόπων ώστε να υλοποιήσουν τα έργα λογισμικού.

Τα προβλήματα σχετικά με τις απαιτήσεις των χρηστών που δεν ικανοποιούνταν και τα λειτουργικά ελλιπή συστήματα, αποτέλεσαν μια ακόμη μια ανάγκη και στελέχη από μεγάλες εταιρίες τα θεώρησαν ως ισχυρό κίνητρο [Domingo, 2004]. Για παράδειγμα, το προκαθορισμένο χρονοδιάγραμμα ορίζει πότε θα παραδοθούν συγκεκριμένα κομμάτια του λογισμικού στο κάθε ένα από τα οποία θα ικανοποιούνται συγκεκριμένες απαιτήσεις. Ωστόσο, όπως αναφέραμε στα προηγούμενα κεφάλαια, παρατηρείται πώς στην πράξη οι απαιτήσεις δεν μπορούν να προδιαγραφούν πλήρως από την έναρξη του έργου. Ακριβώς για αυτό τον λόγο οι εταιρίες ανάπτυξης λογισμικού έπρεπε να εντοπίσουν τρόπους ώστε να μπορούν να διαχειριστούν τα τμήματα εκείνα του λογισμικού για τα οποία οι απαιτήσεις δεν έχουν ξεκαθαριστεί πλήρως. Επιπλέον, ένας ακόμη σημαντικός κίνδυνος προκύπτει όταν οι απαιτήσεις ενδέχεται να τροποποιηθούν ή να αλλάξουν ριζικά, γεγονός που οδηγεί στον επαναπρογραμματισμό του έργου. Αυτό συνεπάγεται, βέβαια, περισσότερο κόστος και χρόνο. Ο κίνδυνος αυτός καθιστά επιτακτική την ανάγκη για εύρεση και θεμελίωση μεθόδων ώστε να μπορούν να εντοπιστούν οι πραγματικές ανάγκες του πελάτη εξ' αρχής. Η διαδικασία για την καταγραφή των προδιαγραφών και των απαιτήσεων είναι αρκετά χρονοβόρα και συχνά οι προδιαγραφές καταλήγουν να μην ισχύουν πλέον όταν πια οριστικοποιηθούν.

Πέρα των δυσκολιών που αφορούν την ασάφεια στον καθορισμό των προδιαγραφών, οι γρήγοροι ρυθμοί με τους οποίους αλλάζουν οι απαιτήσεις καθώς και άλλοι παράγοντες που αφορούν στο ευρύτερο περιβάλλον, δημιουργούν την ανάγκη ώστε οι επιχειρήσεις να μπορούν να προσαρμόζονται εύκολα στις συνεχώς μεταβαλλόμενες και αναπτυσσόμενες εξελίξεις και τεχνολογίες. Έτσι, οι ομάδες ανάπτυξης λογισμικού τείνουν να αναζητήσουν μια πιο ευέλικτη μεθοδολογία η οποία να επιτρέπει τις αστάθειες στις απαιτήσεις.

Οι εταιρείες που αναπτύσσουν λογισμικό, μπορούν να αντιμετωπίσουν μέχρι ένα βαθμό τα προβλήματα αυτά. Σε μια έρευνα που διεξήχθη το 2004 [Domingo, 2004], μερικοί συμμετέχοντες εξέφρασαν την δυσαρέσκειά τους ως προς τις δύσκολες μεθοδολογίες

ανάπτυξης. Οι μεγαλύτερες εταιρίες είναι πιο πιθανό να ακολουθούν καθορισμένες μεθοδολογίες ανάπτυξης ώστε να διασφαλίζουν την ποιότητα κατά την διάρκεια της ανάπτυξης ενός συστήματος. Η επιλογή της μεθοδολογίας πολλές φορές εξαρτάται, αν και όχι αποκλειστικά, από το ίδιο το σύστημα που πρόκειται να αναπτυχθεί. Έτσι η ομάδα ανάπτυξης περιορίζεται σημαντικά ως προς το ποιες μέθοδοι είναι κατάλληλες, γεγονός που μπορεί να επηρεάσει τον χρόνο ανάπτυξης.

Μια ομάδα ανάπτυξης λογισμικού προσπαθεί να μην ξεφύγει από το χρονοδιάγραμμα ώστε να παραδώσει έγκαιρα το λογισμικό, καθώς επίσης πρέπει να ικανοποιήσει όλες τις απαιτήσεις του προϊόντος. Η εύρεση τρόπον ώστε να αναπτυχθεί ένα ποιοτικό σύστημα όσο το δυνατό συντομότερα, είναι πλέον ουσιαστική.

Οι ευέλικτες μεθοδολογίες εμφανίστηκαν για να συμπληρώσουν όλες αυτές τις απαιτήσεις των σύγχρονων εταιριών. Συνεισφέρουν στην έγκαιρη παράδοση ενός συστήματος στον πελάτη και ανταποκρίνονται πιο άμεσα στις περιβαλλοντικές αλλαγές και στις διαφοροποιήσεις των απαιτήσεων και προδιαγραφών του συστήματος. Επιπλέον οι ευέλικτες μεθοδολογίες καθιστούν δυνατή την αποφυγή περιττών διαδικασιών οι οποίες επιβαρύνουν το έργο, είτε χρηματικά είτε χρονικά, αυξάνοντας παράλληλα την παραγωγικότητα. Οι ιδιότητες αυτές των ευέλικτων πρακτικών, τις έχουν καταστήσει ιδιαίτερα δημοφιλείς στον κύκλο της ανάπτυξης λογισμικού κερδίζοντας την προσοχή της βιομηχανίας λογισμικού.

Προηγουμένως αναφερθήκαμε σε «κανόνες» που καθιστούν μια μέθοδο ευέλικτη. Οι κανόνες αυτοί αναφέρονται στο μανιφέστο των ευέλικτων μεθόδων. Στην συνέχεια θα εξετάσουμε τις βασικές αρχές (μανιφέστο) που χαρακτηρίζουν τις ευέλικτες μεθοδολογίες.

3.2 Το μανιφέστο των ευέλικτων μεθόδων

Οι ευέλικτες μεθοδολογίες αποτελούν ένα γενικότερο πλαίσιο στην τεχνολογία λογισμικού. Εμφανίστηκαν κυρίως στα μέσα της δεκαετίας του 1990, ως μια εναλλακτική λύση στις «βαριές μεθοδολογίες» (heavyweight) της παραδοσιακής ανάπτυξης λογισμικού. Τον Φεβρουάριο του 2001, 17 σημαντικές προσωπικότητες

στον τομέα της ευέλικτης ανάπτυξης λογισμικού, μεταξύ των οποίων εμπειρογνώμονες ανάπτυξης λογισμικού και σύμβουλοι εταιριών παραγωγής λογισμικού, συναντήθηκαν στο Snowbird της Utah (Ηνωμένες Πολιτείες Αμερικής) για να συζητήσουν για καινούριους τρόπους και ελαφριές (lightweight) μεθοδολογίες ανάπτυξης λογισμικού. Αποτέλεσμα αυτής της συνάντησης ήταν το μανιφέστο [Agile Alliance, 2001] των ευέλικτων μεθοδολογιών (Agile Manifesto), το οποίο θεωρείται ως «Βίβλος» για τις ευέλικτες μεθόδους και τις αρχές τους. Οι μεθοδολογίες αυτές υπόσχονται προσαρμοστικότητα και ανταπόκριση στις αλλαγές, παραγωγικότερες πρακτικές και λιγότερη γραφειοκρατία. Το όνομα ευέλικτες αναφέρεται κυρίως στην συνολική ικανότητα να προσαρμόζουν κατάλληλα τη διαδικασία ανάπτυξης, όταν προκύπτουν αλλαγές στην πορεία του έργου.

Οι ευέλικτες μεθοδολογίες ορίζονται από τέσσερις βασικές αρχές [Beck et al., 2002]:

- **Άτομα και αλληλεπιδράσεις αντί διαδικασίες και εργαλεία**
- **Καθαρός κώδικας αντί γραπτής τεκμηρίωσης**
- **Συνεργασία με τον πελάτη αντί αυστηρών συμβολαίων**
- **Ανταπόκριση σε αλλαγές αντί ακολουθούμενου σχεδίου**

Αρχικά οι ευέλικτες μεθοδολογίες τονίζουν την ανάγκη για συνεργασία μεταξύ των συμμετεχόντων σε ένα έργο λογισμικού. Ο σημαντικότερος παράγοντας κατά την ανάπτυξη λογισμικού είναι τα εμπλεκόμενα πρόσωπα (προγραμματιστές, ελεγκτές, διευθυντές έργων, υπεύθυνοι μοντελοποίησης, πελάτες) και πως αυτά συνεργάζονται και πράττουν μεταξύ τους. Αν η συνεργασία αυτή δεν είναι επιτυχής οι διαδικασίες και τα εργαλεία δεν πρόκειται να αποτελέσουν σημαντική βοήθεια. Αν η συνεργασία αυτή δεν είναι επιτυχής, οι διαδικασίες και τα εργαλεία δεν πρόκειται να αποτελέσουν σημαντική βοήθεια. Στις ευέλικτες μεθοδολογίες αυτό παρατηρείται στις στενές σχέσεις μεταξύ των συμμετεχόντων, και στα χαρακτηριστικά που προωθούν την ομαδικότητα.

Έπειτα, το μανιφέστο υπογραμμίζει τον πρωταρχικό στόχο του έργου ο οποίος είναι η παραγωγή ποιοτικού, λειτουργικού λογισμικού. Οι υπεύθυνοι για την ανάπτυξη του λογισμικού προτρέπονται να διατηρήσουν τον κώδικα όσο πιο απλό γίνεται. Έτσι το ίδιο το λογισμικό είναι πιο κατανοητό σε σχέση με μακροσκελή έγγραφα και διαγράμματα τεκμηρίωσης που περιγράφουν πως δουλεύει ένα πρόγραμμα. Η

ολοκληρωμένη τεκμηρίωση είναι σημαντική για να εξηγεί ποιος είναι ο στόχος του προγράμματος και πως μπορούμε να το χειριστούμε, αλλά ο ρόλος της θα είναι πάντα δευτερεύων και συμπληρωματικός, αφού το ίδιο το πρόγραμμα είναι το βασικό προϊόν που πρέπει να παραχθεί.

Τρίτον, η συνεργασία και η επικοινωνία της αρμόδιας ομάδας ανάπτυξης του λογισμικού με του πελάτες και συνεπώς μελλοντικούς χρήστες του προϊόντος προτιμάται σε σύγκριση με αυστηρά συμβόλαια και συμβάσεις. Η στενή συνεργασία με τον πελάτη, ο οποίος προσδιορίζει τις απαιτήσεις, είναι απαραίτητη. Η συνεργασία αυτή είναι συνήθως δύσκολη, αφού ο πελάτης αλλάζει συχνά γνώμη σχετικά με τις απαιτήσεις του προγράμματος αλλά αναπόφευκτη. Το να υπάρχει ένα συμβόλαιο με τον πελάτη, που θα καθορίζει τις ευθύνες και τα δικαιώματα του καθενός και θα περιγράφει την μεταξύ τους σχέση, είναι κάτι το σημαντικό. Ωστόσο, το συμβόλαιο δεν μπορεί να αποτελέσει υποκατάστατο της επικοινωνίας.

Τέλος, οι συμμετέχοντες στην ανάπτυξη του λογισμικού θα πρέπει να είναι όλοι καλά πληροφορημένοι, έτοιμοι να εξετάσουν και να αντιμετωπίσουν ικανά ενδεχόμενες καινούριες ανάγκες που θα εμφανιστούν κατά την πορεία του έργου. Όπως αναφέραμε ξανά, κατά τη διάρκεια ανάπτυξης του λογισμικού αλλάζουν οι απαιτήσεις των πελατών, το επιχειρηματικό περιβάλλον ακόμα και η ίδια η τεχνολογία. Αυτές οι αλλαγές δεν αφήνουν ανεπηρέαστη την ανάπτυξη. Για αυτό το λόγο θα πρέπει η ανάπτυξη να μπορεί να ικανοποιεί άμεσα την ανάγκη για αλλαγές. Αυτό δεν σημαίνει ότι απορρίπτεται το σχέδιο χρονοπρογραμματισμού αλλά τα σχέδια θα πρέπει να είναι περισσότερο ευέλικτα, λαμβάνοντας υπόψη τους τις περιπτώσεις αλλαγών, που μπορεί να προκύψουν ξαφνικά [Σφέτσος, 2007].

3.3 Οι αρχές του ευέλικτου μανιφέστο

Στη διακήρυξη των ευέλικτων μεθόδων καθορίστηκαν επιπλέον δώδεκα αρχές που καθοδηγούν τον τρόπο ανάπτυξης. Οι αρχές αυτές αποτελούν το μανιφέστο (Agile Manifesto). Παρουσιάζονται όπως αυτές διατυπώθηκαν [Beck et al., 2002]:

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

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

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

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

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

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

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

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

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

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

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

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

Η τήρηση των βασικών αυτών αρχών του μανιφέστο μειώνουν την πιθανότητα να εμφανιστούν κίνδυνοι και λάθη κατά την διαδικασία ανάπτυξης του λογισμικού. Συνοψίζοντας, λοιπόν, οι ευέλικτες μέθοδοι ενσωμάτωσαν μια ευρεία συλλογή από καλές και δοκιμασμένες αξίες και πρακτικές που βοηθούν στην ανάπτυξη ποιοτικού

λογισμικού. Η ανάπτυξη του λογισμικού γίνεται σε σύντομους επαναληπτικούς (iterative) και αυξητικούς (incremental) κύκλους, παρέχοντας την δυνατότητα της προσαρμογής και αντίδρασης στις αλλαγές που τίθενται από το διαρκώς μεταβαλλόμενο επιχειρησιακό περιβάλλον [Cockburn, 2002]. Με σαφή προτίμηση στην πρόσωπο με πρόσωπο επικοινωνία για τον περιορισμό των παραγομένων εγγράφων και σε στενή συνεργασία με τον πελάτη οι κατασκευαστές και οι διαχειριστές του έργου επικεντρώνονται στην ανάπτυξη λογισμικού, που θεωρείται το πρωταρχικό μέτρο προόδου. Σαν προσαρμόσιμες (adaptive) μέθοδοι και όχι προφητικές, όπως οι παραδοσιακές μέθοδοι, καλωσορίζουν τις αλλαγές ιδιαίτερα στις απαιτήσεις των πελατών, που είναι αβέβαιες και μεταβάλλονται εύκολα.

3.4 Ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού

Τα βασικά χαρακτηριστικά των ευέλικτων μεθόδων είναι η απλότητα και η ταχύτητα. Μια μεθοδολογία μπορεί να θεωρηθεί ως ευέλικτη όταν αυτή είναι επαυξητική (παραδίδονται μικρά τμήματα του λογισμικού, σε τακτά χρονικά διαστήματα), συνεταιριστική (υπάρχει επαρκής επικοινωνία μεταξύ της ομάδας ανάπτυξης και των πελατών), απλή (η μεθοδολογία είναι εύκολη, κατανοητή και τροποποιήσιμη) και προσαρμοστική (μπορεί εύκολα να προσαρμόζεται σε απρόβλεπτες αλλαγές έστω και την τελευταία στιγμή).

Έχοντας ως βάση τα πιο πάνω χαρακτηριστικά, στη συνέχεια αυτού του κεφαλαίου παρουσιάζονται περιληπτικά οι πιο διαδεδομένες ευέλικτες μεθοδολογίες εξαιρουμένης της μεθοδολογίας Scrum, η οποία περιγράφεται εκτενώς στο κεφάλαιο 4:

- «Ακραίος Προγραμματισμός» - Extreme Programming (Beck 1999)
- «Ανάπτυξη με βάση τα χαρακτηριστικά» - Feature Driven Development (Palmer and Felsing 2002)
- «Μέθοδος ανάπτυξης δυναμικών συστημάτων» - Dynamic Systems Development Method (Stapleton 1997)
- «Εναρμονισμένη Ανάπτυξη Λογισμικού» - Adaptive Software Development (Highsmith 2000)

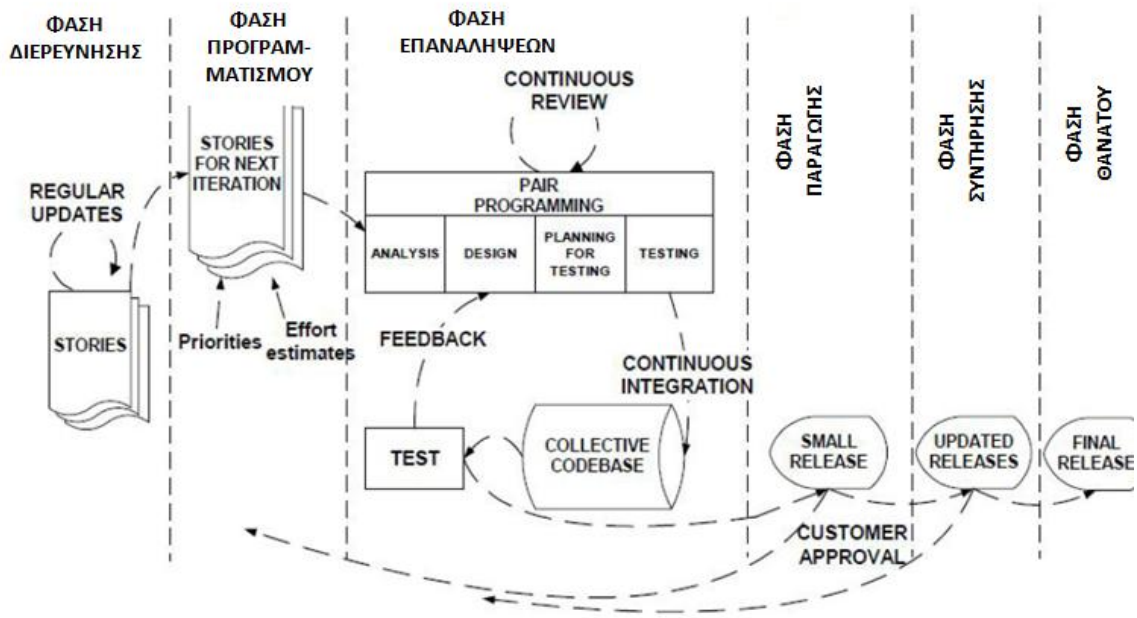
3.4.1 Ακραίος προγραμματισμός (Extreme Programming)

Ο ακραίος προγραμματισμός (Extreme Programming, XP), εντάσσεται στις ευέλικτες μεθοδολογίες αφού προσπαθεί να βελτιώσει την ποιότητα του λογισμικού και επιτρέπει τις συχνές αλλαγές στις απαιτήσεις των χρηστών. Έχει εξελιχθεί από τα προβλήματα που προκαλούνται από τους μεγάλους κύκλους ανάπτυξης των παραδοσιακών μοντέλων ανάπτυξης [Beck, 1999]. Η μεθοδολογία XP χαρακτηρίζεται από μικρούς κύκλους ανάπτυξης, επαυξητικές εκδόσεις, τη συνεχή ανατροφοδότηση, την συνεχή επικοινωνία και τον επαναστατικό σχεδιασμό [Beck, 2004]. Με όλες τις παραπάνω ιδιότητες, οι προγραμματιστές ανταποκρίνονται στο μεταβαλλόμενο περιβάλλον με περισσότερη ευκολία. Ο όρος «ακραίος» προέρχεται από τη λήψη αυτών των αρχών και πρακτικών κοινής λογικής σε ακραία επίπεδα. Η μέθοδος αυτή βασίζεται στην πεποίθηση ότι τα δυνατά στοιχεία των παραδοσιακών προσεγγίσεων (π.χ. η ανάμειξη του πελάτη), λαμβάνονται και εφαρμόζονται στα ακραία επίπεδα τους, βασισμένο στην ιδέα ότι αν κάτι είναι καλό, τότε ο μέγιστος βαθμός του θα επιφέρει ακόμη καλύτερα αποτελέσματα.

Ο ακραίος προγραμματισμός επινοήθηκε από τον Kent Beck στα τέλη της δεκαετίας του 1990 και αρχές της δεκαετίας του 2000. Ωστόσο μετά από δοκιμές στην πράξη [Anderon et al., 1998], ο ακραίος προγραμματισμός καταγράφηκε για πρώτη φορά από τον Beck, περιγράφοντας τα βασικά χαρακτηριστικά και πρακτικές της μεθόδου, στο βιβλίο «Extreme Programming Explained» [Beck and Anders, 2005].

Ο κύκλος ζωής του XP περιλαμβάνει τις εξείς φάσεις όπως αυτές παρουσιάζονται στο σχήμα 3.1 [Beck, 2004][Beck and Anders, 2005]:

1. Διερεύνηση (Exploration)
2. Προγραμματισμός (Planning)
3. Επαναλήψεις (Iterations to release phase)
4. Παραγωγή (Productionizing)
5. Συντήρηση (Maintenance)
6. Θάνατο (Death)



Σχήμα 3.1 Κύκλος ζωής της μεθοδολογίας Ακραίου Προγραμματισμού [Beck, 2004]

Η φάση της **διερεύνησης** ή εξερεύνησης, περιλαμβάνει στην ουσία την καταγραφή των χαρακτηριστικών-λειτουργιών που πρόκειται να περιλαμβάνει το σύστημα. Οι πελάτες καλούνται να καταγράψουν σε «κάρτες ιστορίας» (story cards) τα χαρακτηριστικά γνωρίσματα που θα προστεθούν στο σύστημα. Ταυτόχρονα η ομάδα ανάπτυξης εξοικειώνεται με τα εργαλεία, τις πρακτικές και την τεχνολογία που θα χρησιμοποιηθούν στο έργο. Αφού καθοριστούν οι τεχνολογίες που πρόκειται να χρησιμοποιηθούν, τότε αναπτύσσεται ένα πρωτότυπο του συστήματος.

Στη φάση του **προγραμματισμού** καθορίζονται οι ιστορίες που θα τύχουν προτεραιότητας και το τι πρόκειται να περιέχει η πρώτη έκδοση του συστήματος. Η ομάδα ανάπτυξης αξιολογεί τις ιστορίες, υπολογίζεται η προσπάθεια και ο φόρτος εργασίας που χρειάζεται για κάθε μια από αυτές και τότε κατασκευάζεται ένα χρονοδιάγραμμα (πρόγραμμα) εργασίας.

Στη φάση των **επαναλήψεων** γίνεται ένας διαχωρισμός του προγράμματος που έχει καθοριστεί στην προηγούμενη φάση σε διάφορες επαναλήψεις, κάθε μια από τις οποίες διαρκεί από μια έως τέσσερις εβδομάδες. Η πρώτη επανάληψη περιλαμβάνει τον καθορισμό της αρχιτεκτονικής του συστήματος και στη συνέχεια ακολουθεί η αναβάθμιση του συστήματος με την ενσωμάτωση των ιστοριών. Στο τέλος κάθε

επανάληψης δοκιμάζεται το προϊόν από τον πελάτη. Κάθε επανάληψη ολοκληρώνεται με την παραγωγή της έκδοσης που αναπτύχθηκε κατά την επανάληψη.

Στη φάση της **παραγωγής** το προϊόν δοκιμάζεται περεταίρω. Διεξάγονται περισσότεροι έλεγχοι και δοκιμές ώστε να εξασφαλιστεί η απόδοση του συστήματος πριν αυτό παραδοθεί στον πελάτη. Κατά την φάση αυτή μπορεί να εντοπιστούν αλλαγές που θα πρέπει να γίνουν και αποφασίζεται κατά πόσο θα πραγματοποιηθούν στην παρούσα επανάληψη.

Επιπλέον καταγράφονται καινούριες ιδέες και οι εισηγήσεις ώστε να είναι δυνατή η υλοποίηση τους σε νεότερες εκδόσεις που πραγματοποιούνται στη φάση της **συντήρησης.**

Η φάση του **θανάτου** επέρχεται όταν πλέον ο πελάτης δεν έχει οποιεσδήποτε ιστορίες για υλοποίηση. Αυτό συνεπάγεται πώς το σύστημα ικανοποιεί όλες τις απαιτήσεις του χρήστη. Στο στάδιο αυτό καταγράφεται η απαραίτητη τεκμηρίωση (documentation) του λογισμικού και δεν είναι αποδεκτές οποιεσδήποτε αλλαγές στην αρχιτεκτονική, στο σχεδιασμό ή στον κώδικα του συστήματος. Επίσης, ο θάνατος μπορεί να επέλθει πριν ακόμα το σύστημα προλάβει να ολοκληρωθεί, εάν παρατηρηθεί πως δεν μπορεί να επιφέρει τα επιθυμητά αποτελέσματα .

Ο ακραίος προγραμματισμός είναι ένα συνονθύλευμα ιδεών και πρακτικών επηρεασμένες από είδη υπάρχουσες μεθοδολογίες. Στοχεύει στην ανάπτυξη ενός επιτυχημένου προϊόντος λογισμικού παρά την ασάφεια στον καθορισμό των προδιαγραφών και τις αστάθειες στις απαιτήσεις. Οι σύντομες επαναλήψεις με μικρές επαυξητικές εκδόσεις και η γρήγορη ανατροφοδότηση, η συμμετοχή των πελατών, η επικοινωνία και ο συντονισμός, η συνεχής ενσωμάτωση λειτουργιών και η δοκιμή, η συλλογική ιδιοκτησία του κώδικα, η περιορισμένη τεκμηρίωση και ο προγραμματισμός ανά ζεύγη είναι ανάμεσα στα κύρια χαρακτηριστικά του ΧΡ. Πιο κάτω αναφέρονται οι πιο βασικές πρακτικές και χαρακτηριστικά του ΧΡ:

- Προγραμματισμός: Ο προγραμματιστής εκτιμά την προσπάθεια που απαιτείται για την υλοποίηση των ιστοριών του πελάτη, σε συνεργασία με τον πελάτη αποφασίζεται το χρονοδιάγραμμα υλοποίησης των ιστοριών, βάσει εκτιμήσεων.

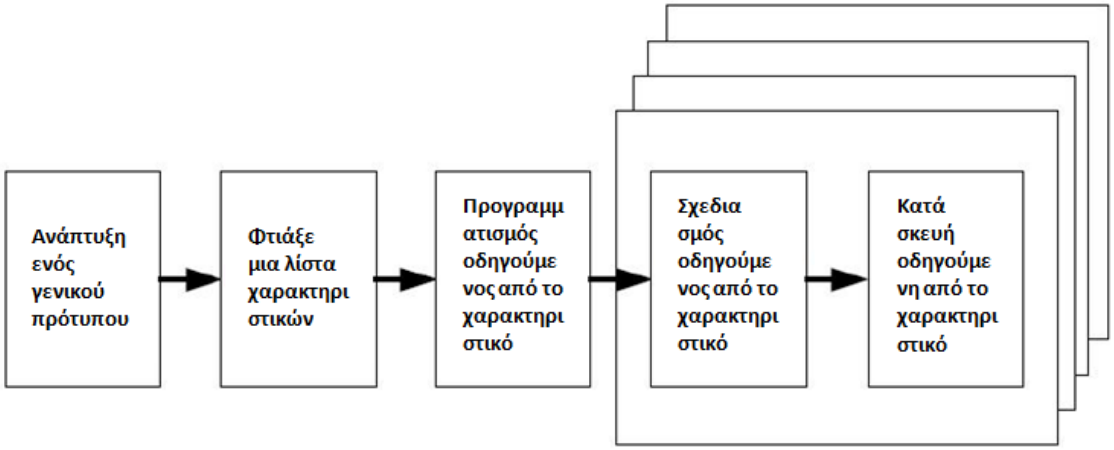
- Μικρές εκδόσεις: Η εφαρμογή αναπτύσσεται με την εκτέλεση μιας σειράς μικρών, επαυξητικών εκδόσεων. Οι εκδόσεις μπορεί να αναπτύσσονται από καθημερινά μέχρι και μηνιαία.
- Περιγραφή του συστήματος με μεταφορές: Μια μεταφορά (metaphor) αποτελεί μια κωδικοποίηση ή ονοματοθεσία για ένα τμήμα του συστήματος ή μιας λειτουργίας. Το σύστημα ολόκληρο χαρακτηρίζεται από ένα σύνολο μεταφορών μεταξύ πελατών και προγραμματιστών και με τον τρόπο αυτό είναι ευκολότερη η επικοινωνία για το πώς λειτουργεί το σύστημα.
- Απλή σχεδίαση: Δίνεται έμφαση στον σχεδιασμό, την απλούστερη δυνατή λύση η οποία και εφαρμόζεται καθώς περιττή πολυπλοκότητα και επιπλέον κώδικας αφαιρούνται αμέσως.
- Αναδιάρθρωση του συστήματος (refactoring): Η αλλαγή σε ένα τμήμα κώδικα μπορεί αναπόφευκτα να επιφέρει επιμέρους αλλαγές σε ένα άλλο τμήμα του συστήματος. Το γεγονός αυτό αντιμετωπίζεται με την αναδιάρθρωση του συστήματος, τη βελτίωση της επικοινωνίας, την απλούστευση και την προσθήκη ευελιξίας, χωρίς να αλλάζει η λειτουργικότητα του λογισμικού συστήματος.
- Προγραμματισμός σε ζεύγη (Pair programming): Όλος ο κώδικας είναι γραμμένος από δύο προγραμματιστές οι οποίοι δουλεύουν μαζί και ταυτόχρονα σε ένα υπολογιστή. Ο ένας εκ των δύο γράφει κώδικα καθώς ο δεύτερος αναθεωρεί άμεσα τον κώδικα. Ο προγραμματιστής που δακτυλογραφεί φέρει τον ρόλο του οδηγού ενώ αυτός που ελέγχει τον κώδικα ονομάζεται παρατηρητής ή πλοηγός.
- Συλλογική ιδιοκτησία: Δεν υπάρχει ένα μοναδικό πρόσωπο που κατέχει ή είναι υπεύθυνο για μεμονωμένους τομείς κώδικα. Ο καθένας μπορεί να αλλάξει οποιοδήποτε μέρος του κώδικα, ανά πάσα στιγμή. Ένα σημαντικό πλεονέκτημα αυτού είναι το ότι επιταχύνεται η όλη διαδικασία ανάπτυξης γιατί εάν ένας προγραμματιστής εντοπίσει ένα λάθος ή ασάφεια στον κώδικα τότε μπορεί άμεσα να προσχωρήσει με την διόρθωση του τμήματος αυτού.
- Συνεχής ολοκλήρωση: Ένα νέο κομμάτι του κώδικα ενσωματώνεται στο ισχύον σύστημα μόλις είναι έτοιμο. Όταν γίνει η αναβάθμιση του συστήματος με την προσθήκη ενός νέου τμήματος, τότε το σύστημα θα πρέπει ξανά να «περάσει» την διαδικασία ελέγχου και τότε οι αλλαγές γίνονται αποδεκτές.

- 40 ώρες την εβδομάδα: Κανείς δεν μπορεί να εργαστεί υπερωρίες δύο εβδομάδες στη σειρά. Ο μέγιστος ρυθμός εργασίας είναι 40 ώρες την εβδομάδα διαφορετικά αντιμετωπίζεται ως πρόβλημα ανάμεσα στην ομάδα ανάπτυξης. Το νόημα είναι ότι, αν ο άνθρωπος εργάζεται ξεκούραστος τότε δουλεύει πιο δημιουργικά και αποδοτικά.
- Εμπλοκή του πελάτη: Ο πελάτης κατέχει σημαντικό ρόλο στην ανάπτυξη του λογισμικού. Δεν αντιμετωπίζεται σαν αυτός που θα δώσει τα χρήματα για την ανάπτυξη αλλά σαν αυτός που θα χρησιμοποιήσει το τελικό προϊόν. Ο πελάτης πρέπει να είναι διαθέσιμος ανά πάσα στιγμή για την ομάδα ανάπτυξης, έτοιμος να απαντήσει σε κάθε απορία.
- Πρότυπα κωδικοποίησης: Υπάρχουν κανόνες κωδικοποίησης και ακολουθούνται από τους προγραμματιστές έτσι ώστε να υπάρχει συνέπεια και επαρκή επικοινωνία μεταξύ των μελών της ομάδας ανάπτυξης. Τα πρότυπα καθορίζονται συναινετικά από κοινού.

3.4.2 Ανάπτυξη με βάση τα χαρακτηριστικά (Feature Driven Development)

Η ανάπτυξη βάσει χαρακτηριστικών (Feature Driven Development, FDD), χρησιμοποιήθηκε για πρώτη φορά για την ανάπτυξη μιας πολύπλοκης τραπεζικής εφαρμογής στα τέλη της δεκαετίας του 1990 [Palmer and Felsing, 2002]. Σε αντίθεση με τις υπόλοιπες μεθοδολογίες δεν ακολουθεί ολόκληρη την διαδικασία ανάπτυξης λογισμικού αλλά επικεντρώνεται στη φάση του σχεδιασμού και της υλοποίησης [Palmer and Felsing, 2002]. Η FDD μεθοδολογία ακολουθεί επαναληπτική και αυξητική ανάπτυξη με τακτικές παραδόσεις τμημάτων του συστήματος.

Η προσέγγιση αυτή περιλαμβάνει πέντε φάσεις από τις οποίες διέρχεται η ανάπτυξη του συστήματος (Σχήμα 3.2). Οι τρεις πρώτες φάσεις εκτελούνται στην αρχή του έργου. Οι δύο τελευταίες φάσεις αποτελούν το επαναληπτικό μέρος της διαδικασίας όπου και παρουσιάζονται οι αρχές την ευέλικτη ανάπτυξης με γρήγορες προσαρμογές σε καθυστερημένες αλλαγές όσον αφορά τις απαιτήσεις και τις επιχειρησιακές ανάγκες. Η προσέγγιση FDD περιλαμβάνει συχνές παραδόσεις και λεπτομερής παρακολούθηση της προόδου του έργου.



Σχήμα 3.2 Διαδικασία ανάπτυξη με βάση τα χαρακτηριστικά [Palmer and Felsing, 2002]

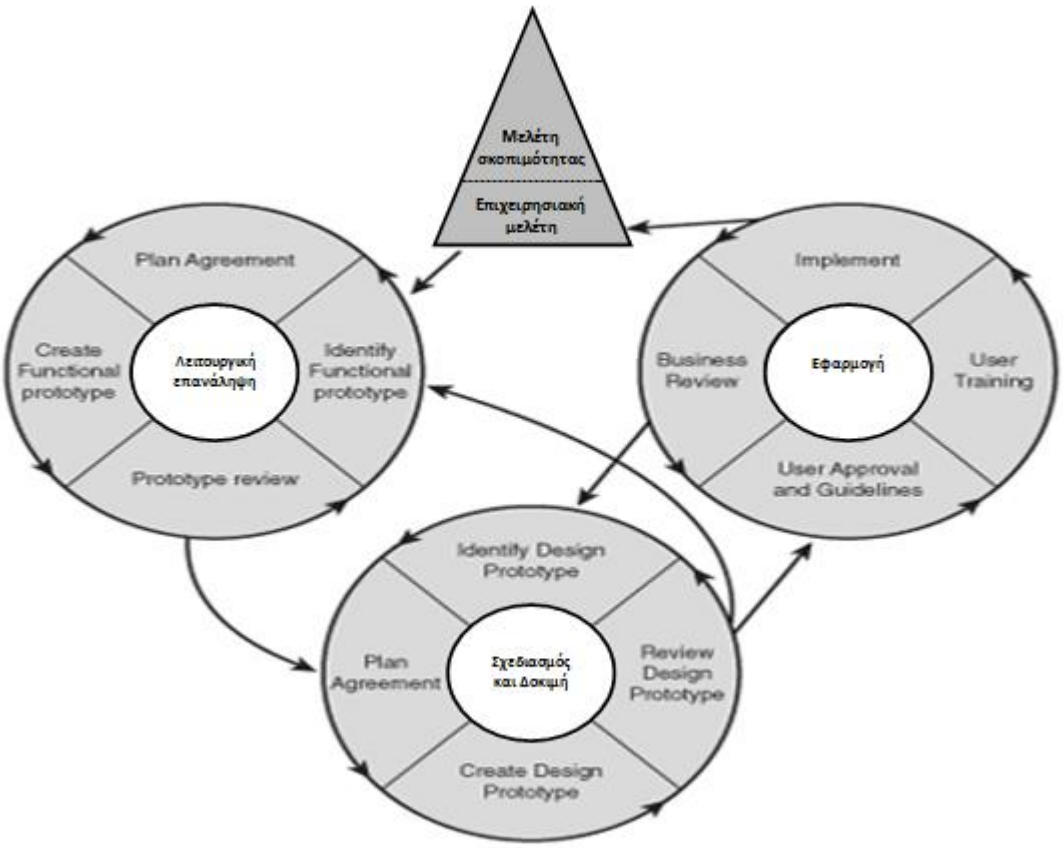
Πιο κάτω παρουσιάζονται τα διαδοχικά βήματα της διαδικασίας αυτής:

- **Ανάπτυξη ενός γενικού προτύπου (Develop an Overall Model)**: Με την έναρξη τα φάσης αυτής γίνεται μια πρώτη γνωριμία με το πεδίο, το πλαίσιο και τις απαιτήσεις του συστήματος. Ενδέχεται να κατασκευαστούν περαιτέρω τεκμηριωμένες απαιτήσεις όπως οι περιπτώσεις χρήσης και οι λειτουργικές προδιαγραφές. Πραγματοποιείται ένα λεπτομερές «πέρασμα» (walkthrough) του συστήματος και ο επικεφαλής αρχιτέκτονας (software architect) ενημερώνεται αναλυτικά σχετικά με το σύστημα. Το σύστημα διαιρείται σε περαιτέρω υποσυστήματα και διενεργούνται επιμέρους «περάσματα»
- **Φτιάξτε μια λίστα χαρακτηριστικών (Build a Features List)**: Παράγεται μια κατηγοριοποιημένη λίστα με τα χαρακτηριστικά για να υποστηρίξει τις απαιτήσεις
- **Προγραμματισμός οδηγούμενος από το χαρακτηριστικό (Plan by Feature)**: Η ομάδα ανάπτυξης ταξινομεί τα χαρακτηριστικά γνωρίσματα του συστήματος ανάλογα με τις προτεραιότητες και τις μεταξύ τους εξαρτήσεις και αυτά ανατίθενται σε επικεφαλής προγραμματιστές. Επιπλέον, τα τμήματα που προσδιορίστηκαν στην πρώτη φάση ανατίθενται στους ιδιοκτήτες κατηγορίας (class owners) (μεμονωμένους προγραμματιστές). Επίσης, το χρονοδιάγραμμα και τα ορόσημα τίθενται για τα χαρακτηριστικά γνωρίσματα.
- **Σχεδιασμός και Κατασκευή του συστήματος οδηγούμενα από το χαρακτηριστικό (Design & Build by Feauture)**: Στο στάδιο αυτό επιλέγονται συγκεκριμένα χαρακτηριστικά γνωρίσματα για υλοποίηση. Οι φάσεις αυτές

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

3.4.3 Μέθοδος ανάπτυξης δυναμικών συστημάτων (Dynamic Systems Development)

Η μεθοδολογία ανάπτυξης δυναμικών συστημάτων (Dynamic Systems Development, DSD), πρωτοεμφανίστηκε στο Ηνωμένο Βασίλειο στα μέσα της δεκαετίας του 1990. Αποτελεί ένα μείγμα αλλά παράλληλα και επέκταση της γρήγορης προτυποποίησης και πρακτικών επαναληπτικής ανάπτυξης [Stapleton, 1997]. Ο Martin Fowler, ένας από τους συγγραφείς του Μανιφέστου των ευέλικτων μεθόδων, πιστεύει πώς «Η DSD μεθοδολογία είναι αξιοσημείωτη για το λόγο ότι ακολουθεί την υποδομή των παραδοσιακών μεθοδολογιών ανάπτυξης λογισμικού ενώ παράλληλα ακολουθεί τις αρχές του ευέλικτου τρόπου ανάπτυξης» [Fowler, 2005]. Η βασική ιδέα πίσω από την DSD είναι να καθορίσει το χρόνο και τους πόρους, και στη συνέχεια να προσαρμόσει το ποσό της λειτουργικότητας ανάλογα και όχι να καθορίσει το ποσό της λειτουργικότητας σε ένα προϊόν, και κατόπιν να προσαρμοστεί ανάλογα ο χρόνος και οι πόροι για την επίτευξη αυτής της λειτουργικότητας [DSDM Consortium, 2012]. Το DSDM αποτελείται από πέντε φάσεις (Σχήμα 3.3). Οι δύο πρώτες φάσεις εκτελούνται σειριακά μόνο μια φορά. Οι υπόλοιπες τρείς φάσεις, όπου και αναπτύσσεται το σύστημα, είναι, επαναληπτικές και αυξητικές. Ένα σημαντικό χαρακτηριστικό της μεθόδου αυτής είναι ο χρόνος. Οι επαναλήψεις αντιμετωπίζονται σαν χρονικά παράθυρα. Ένα χρονικό παράθυρο έχει συγκεκριμένα όρια και η επανάληψη επιβάλλεται να ολοκληρωθεί μέσα στα χρονικά αυτά όρια. Όπως και στην διαδικασία FDD έτσι και στο DSD μια επανάληψη και συνεπώς ένα χρονικό παράθυρο μπορεί να διαρκέσει από μερικές μέρες και κάποιες εβδομάδες το μέγιστο.



Σχήμα 3.3 Διαδικασία ανάπτυξης δυναμικών συστημάτων [Stapleton, 1997]

1. **Μελέτη σκοπιμότητας (Feasibility study):** Στη φάση αυτή λαμβάνεται η απόφαση για το αν θα χρησιμοποιηθεί ή όχι η μεθοδολογία DSD. Αυτό καθορίζεται από το είδος του έργου και, οργανωτικά θέματα και κυρίως τον ανθρώπινο παράγοντα. Επιπλέον, παράγονται δύο προϊόντα εργασίας, μια έκθεση σκοπιμότητας και ένα γενικό σχέδιο για την ανάπτυξη.
2. **Επιχειρησιακή μελέτη (Business study):** Η συνιστώμενη προσέγγιση σε αυτή τη φάση είναι να κατανοηθεί το επιχειρηματικό πεδίο του έργου. Τα βασικά αποτελέσματα αυτής της ενότητας είναι ο ορισμός της αρχιτεκτονικής του συστήματος και ένα πρωτότυπο σχέδιο αυτού.
3. **Λειτουργική πρότυπη επανάληψη (Functional model iteration):** Είναι η πρώτη επαναληπτική φάση. Η φάση αυτή περιλαμβάνει την ανάλυση, την κωδικοποίηση και την πρωτοτυποποίηση. Τα αποτελέσματα που συλλέγονται από την φάση αυτή χρησιμοποιούνται ώστε να βελτιωθούν τα πρωτότυπα ανάλυσης. Το κύριο αποτέλεσμα της φάσης αυτής είναι ένα λειτουργικό πρωτότυπο.

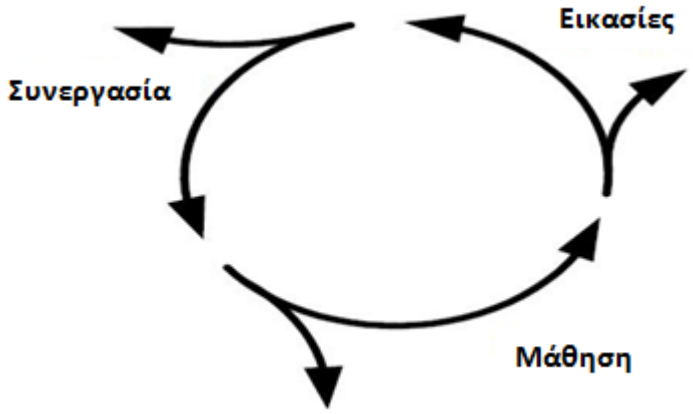
4. **Σχεδιασμός και δοκιμή της επανάληψης (Design and build iteration):** Το σύστημα κτίζεται ουσιαστικά σε αυτή τη φάση. Ο σχεδιασμός και η λειτουργική αξιολόγηση των πρωτοτύπων γίνονται από τους χρήστες και η επιπλέον ανάπτυξη βασίζεται στα σχόλια των χρηστών.
5. **Εφαρμογή (Implementation):** Σε αυτή την τελική φάση, το σύστημα παραδίδεται στους χρήστες. Η κατάρτιση των χρηστών όσο αφορά την χρήση του συστήματος, παρέχεται. Εγχειρίδια χρήστη και μια έκθεση ανασκόπησης του έργου είναι διαθέσιμα. Ωστόσο, η επαναληπτική και αυξητική φύση της DSD σημαίνει ότι η συντήρηση μπορεί να θεωρηθεί ως συνεχιζόμενη ανάπτυξη. Αντί να τελειώσει το έργο με ένα κύκλο, μπορεί να επιστρέψει σε κάποια από τις προηγούμενες φάσεις, έτσι ώστε το σύστημα να εμπλουτιστεί με τις απαιτήσεις που τυχόν υπολείπονται και να τελειοποιηθεί.

3.4.4 Προσαρμόσιμη Ανάπτυξη Λογισμικού (Adaptive Software Development)

Η προσαρμόσιμη ανάπτυξη λογισμικού (Adaptive Software Development, ASD) που αναπτύχθηκε από τον James A. Highsmith, προσφέρει μια ευέλικτη και προσαρμοστική προσέγγιση σε έργα λογισμικού [Highsmith, 2000]. Ένα σύστημα δεν είναι δυνατόν να προγραμματίσει πλήρως με επιτυχία σε ένα μεταβαλλόμενο και απρόβλεπτο επιχειρησιακό περιβάλλον. Η μεθοδολογία αυτή προωθεί την αυξητική και επαναληπτική ανάπτυξη με την συνεχή κατασκευή πρωτότυπων.

Η ASD επικεντρώνεται σε τρεις μη-γραμμικές και επικαλυπτόμενες φάσεις (Σχήμα 3.4):

- **Εικασίες (Speculate):** Ορίζεται η αποστολή του έργου και προσδιορίζονται οι ασάφειες.
- **Συνεργασία (Collaborate):** Υπογραμμίζει τη σημασία της ομαδικής εργασίας για την ανάπτυξη συστημάτων τα οποία εύκολα μεταβάλλονται.
- **Μάθηση (Learn):** Αυτή η φάση υπογραμμίζει την ανάγκη για την εκτίμηση των κινδύνων και την αντίδραση σε λάθη, και ότι οι απαιτήσεις μπορεί να αλλάξουν κατά τη διάρκεια της ανάπτυξης.



Σχήμα 3.4 Κύκλος εναρμονισμένης ανάπτυξη λογισμικού [Highsmith, 2000]

Δεδομένου ότι τα αποτελέσματα είναι απρόβλεπτα, ο Highsmith βλέπει παράδοξα τον προγραμματισμό μέσα σε ένα προσαρμοστικό περιβάλλον. Στην παραδοσιακή ανάπτυξη λογισμικού όταν τα πράγματα δεν βαδίζουν βάσει σχεδίου τότε υπάρχει κάποιο πρόβλημα και θα πρέπει να διορθωθεί. Ωστόσο, σε ένα προσαρμοστικό περιβάλλον οι αποκλίσεις μας καθοδηγούν προς την σωστή λύση.

Η ASD επικεντρώνεται περισσότερο στα αποτελέσματα και την ποιότητά τους παρά στα καθήκοντα ή τη διαδικασία που χρησιμοποιείται για την παραγωγή των αποτελεσμάτων. Σε ένα απρόβλεπτο περιβάλλον θα πρέπει οι άνθρωποι να συνεργάζονται με έναν καθορισμένο τρόπο για την αντιμετώπιση της αβεβαιότητας. Η διοίκηση προωθεί την επικοινωνία καθώς αποφεύγει να ορίζει στους ανθρώπους τι να κάνουν, έτσι ώστε να παράγονται πιο δημιουργικά και αποδοτικά αποτελέσματα.

Η μεθοδολογία αυτή δεν διαθέτει λεπτομερείς αρχές όπως ο XP, αλλά παρέχει ένα πλαίσιο για το πώς να ενθαρρύνεται η συνεργασία και μάθηση στην διαδικασία ανάπτυξης του έργου. Η ASD δεν παρουσιάζεται ως μια μεθοδολογία για να κατασκευάζονται έργα λογισμικού, αλλά πρόκειται για μια προσέγγιση ή μια στάση που πρέπει να υιοθετηθεί από μια οργάνωση όταν εφαρμόζει ευέλικτες διαδικασίες [Highsmith, 2000].

3.5 Ανάπτυξη με βάση τις δοκιμές (Test Driven Development)

Η ανάπτυξη με βάση τις δοκιμές (Test Driven Development, TDD) είναι μια διαδικασία ανάπτυξης λογισμικού που βασίζεται στην επανάληψη ενός πολύ σύντομου κύκλου ανάπτυξης. Αποτελείται από επαναλήψεις με βάση αυτοματοποιημένες δοκιμές (Automated unit tests) που καθορίζουν μια επιθυμητή βελτίωση ή μια νέα λειτουργία του συστήματος. Αντί της γραφής κώδικα πρώτα και τον έλεγχο εκ των υστέρων, η δοκιμή γίνεται πριν την γραφή του λειτουργικού κώδικα. Το TDD προέρχεται από τις ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού και η ανακάλυψη του οφείλεται στον Kent Beck, όταν το 2003 δημοσίευσε ένα βιβλίο με τίτλο «Test Driven Development: By Example» [Beck, 2003].

Η τεχνική αυτή εστιάζει την προσοχή της στις δοκιμές των υλοποιημένων χαρακτηριστικών ενός συστήματος, στην ενσωμάτωση και αποδοχή τους. Μια δοκιμή που δεν μπορεί εφαρμοστεί, αναδεικνύει μια απαίτηση του συστήματος που δεν είναι καλά καθορισμένη. Οι προγραμματιστές γράφουν τον απαιτούμενο κώδικα ώστε μια δοκιμή νε τελειώσει με επιτυχία. Τα χαρακτηριστικά του λογισμικού, κωδικοποιούνται ανεξάρτητα, και έπειτα ακολουθεί η δοκιμή για να εξακριβωθεί εάν ένα χαρακτηριστικό έχει υλοποιηθεί σωστά. Εάν η δοκιμή αποτύχει τότε ο κώδικας αλλάζει, προσαρμόζεται ανάλογα και η δοκιμή επαναλαμβάνεται. Εάν η δοκιμή επιτύχει τότε ο προγραμματιστής προχωράει στην φάση του «Refactoring». Πιο αναλυτικά η διαδικασία του TDD φαίνεται στο Σχήμα 3.5 και περιγράφεται πιο κάτω με βάση τον [Beck, 2003]:

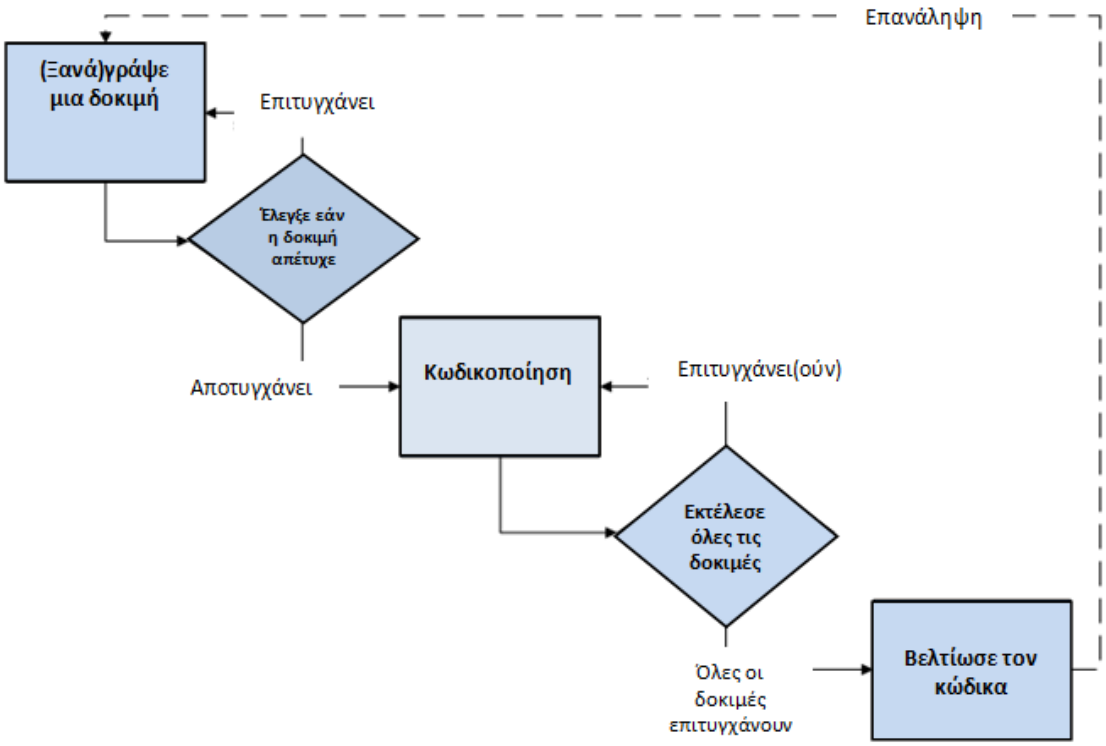
1. **Πρόσθεσε μια καινούρια δοκιμή (Add a test):** Στο TDD η υλοποίηση κάθε νέου χαρακτηριστικού ξεκινά με μια δοκιμή. Η δοκιμή αυτή αρχικά θα αποτύχει, αφού δεν έχει υλοποιηθεί οποιοσδήποτε κώδικας. Για να γραφτεί μια δοκιμή, οι προγραμματιστής θα πρέπει να έχει κατανοήσει πλήρως τις προδιαγραφές και τις απαιτήσεις του εν λόγω χαρακτηριστικού. Αυτό επιτυγχάνεται με την χρήση σεναρίων Use cases ή User stories που καλύπτουν τις απαιτήσεις του συστήματος. Μια δοκιμή, όπως αναφέραμε, μπορεί επίσης να περιλαμβάνει βελτιώσεις ή τροποπαύσεις μια υφιστάμενης δοκιμής. Μια λεπτή αλλά ουσιαστική διαφορά του TDD και του παραδοσιακού τρόπου όπου ο κώδικας γράφεται κε έπειτα δοκιμάζεται, είναι το γεγονός ότι αναγκάζει τους

κατασκευαστές του συστήματος να επικεντρώνονται στις απαιτήσεις πριν την γραφή κώδικα.

2. **Εκτέλεσε όλες τι δοκιμές και έλεγξε αν κάποια αποτυγχάνει (Run all tests and see if the new one fails):** Στη φάση αυτή διασφαλίζεται ότι η διαδικασία της αυτοματοποιημένης δοκιμής δουλεύει σωστά και ως εκ τούτου η πρώτη δοκιμή δεν ολοκληρώνεται με επιτυχία για τον λόγο που αναφέραμε πιο πάνω. Έτσι διαφαίνεται κατά πόσο η διαδικασία της δοκιμής είναι έμπιστη (αν και αυτό δεν ισχύει πάντα) ως προς το γεγονός ότι εξετάζει πάντα το σωστό και επιτυγχάνει μόνο στις περιπτώσεις που πρέπει.
3. **Κωδικοποίηση (Write some code):** Το επόμενο βήμα είναι η συγγραφή του απαιτούμενου κώδικα ώστε η δοκιμή να ολοκληρωθεί με επιτυχία. Ο νέος κώδικας που γράφεται σε αυτό το στάδιο δεν θα είναι πάντα τέλειος και μπορεί, για παράδειγμα, να περάσει τη δοκιμή με «ύπουλο» τρόπο. Το γεγονός αυτό είναι αποδεκτό, διότι στα επόμενα βήματα ο κώδικας θα βελτιωθεί. Είναι σημαντικό το ότι ο κώδικας έχει σχεδιαστεί μόνο για να περάσει το την δοκιμή με επιτυχία. Καμία περαιτέρω λειτουργικότητα θα πρέπει να προβλεφθεί, να υλοποιηθεί και να επιτραπεί σε οποιοδήποτε στάδιο.
4. **Εκτέλεσε τις αυτοματοποιημένες δοκιμές και έλεγξε αν επιτυγχάνουν (Run the automated tests and see them succeed):** Εφόσον όλες οι δοκιμές ολοκληρωθούν με επιτυχία, ο προγραμματιστής μπορεί να είναι σίγουρος ότι ο κώδικας που έχει υλοποιηθεί καλύπτει όλες τις απαιτούμενες προδιαγραφές. Η φάση αυτή είναι το πιο κρίσιμο σημείο του κύκλου ζωής αφού καθορίζει εάν μια επανάληψη θα ολοκληρωθεί ή όχι.
5. **Βελτίωση του κώδικα (Refactor code):** Το Refactoring αναφέρεται στις βελτιώσεις που μπορούν να γίνουν στον κώδικα (Clean up Code). Είναι μια πειθαρχημένη διαδικασία για βελτίωση του κώδικα που υλοποιήθηκε στο προηγούμενο βήμα, χωρίς όμως να αλλάζει τη συμπεριφορά του [Shore and Warden, 2008]. Ο κύριος στόχος της φάσης αυτής είναι η εξάλειψη των επικαλύψεων (Duplications), ωστόσο οι προγραμματιστές μπορούν επίσης να: βελτιώσουν την αναγνωσιμότητα του κώδικα, απλοποίηση της δομής κώδικα, αλλαγή κώδικα ώστε να συμμορφώνονται με τις πρακτικές κωδικοποίησης, βελτίωση της απόδοσης, βελτίωση της προοπτικής για συντήρηση (Maintainability), βελτίωση επεκτασιμότητας. Για παράδειγμα η διάσπαση

μεγάλων συναρτήσεων σε πιο μικρές, όχι μόνο βελτιώνει την αναγνωσιμότητα αλλά επιτρέπει να επαναχρησιμοποιηθεί. Κατά το Refactoring γίνονται μόνο μικρές αλλαγές στον κώδικα, αλλά αυτές οι μικρές αλλαγές μπορούν να έχουν σημαντικό αντίκτυπο κατά την μετέπειτα κωδικοποίηση που πολλές φορές γίνεται πολύπλοκη. Η διαδικασία είναι παρόμοια με το XP, όπου ο σχεδιασμός του συστήματος προκύπτει καθώς το έργο προχωράει. Το Refactoring υποστηρίζει τη συλλογική ιδιοκτησία κώδικα, όπως και στο XP. Για τον λόγο αυτό ακολουθούνται πρότυπα κωδικοποίησης και πρακτικές, ώστε ο κώδικας που γράφτηκε από άλλους προγραμματιστές να γίνεται ευκολότερα κατανοητός. Έτσι, δίνει στους προγραμματιστές τη δυνατότητα να αλλάξουν ανά πάσα στιγμή οποιοδήποτε κώδικα, όταν είναι απαραίτητο, αντί να εξαρτώνται από άλλους προγραμματιστές.

6. **Επανάληψη (Repeat):** Οι επαναλήψεις συνεχίζονται με τον ίδιο τρόπο και έτσι σε κάθε νέο κύκλο προστίθεται λειτουργικότητα στο σύστημα. Αν ο νέος κώδικας δεν ανταποκρίνεται γρήγορα σε μια νέα δοκιμή, ή κάποιες δοκιμές απροσδόκητα αποτυγχάνουν, τότε ο προγραμματιστής θα πρέπει καταφύγει σε περαιτέρω αποσφαλμάτωση (Debugging).



Σχήμα 3.5 Κύκλος ζωής της Ανάπτυξη με βάση τις δοκιμές [Excirial, 2009]

3.6 Χαρακτηριστικά ευέλικτων μεθοδολογιών

Σύμφωνα με τους [Highsmith and Cockburn, 2004], *«η καινοτομία για τις ευέλικτες μεθοδολογίες δεν είναι οι πρακτικές που χρησιμοποιούν, αλλά η αναγνώριση τους από τους ανθρώπους ως τους κύριους συντελεστές της επιτυχίας του έργου, σε συνδυασμό με την έμφαση στην αποτελεσματικότητα και την ευελιξία. Αυτό δημιουργεί ένα νέο συνδυασμό αξιών και αρχών που ορίζουν μια ευέλικτη κοσμοθεωρία»*. Οι ακόλουθες αρχές των ευέλικτων πρακτικών θεωρούνται ως οι κύριες διαφορές μεταξύ ευέλικτων και παραδοσιακών μεθοδολογιών.

Προσανατολισμένες στον άνθρωπο: Οι ευέλικτες μεθοδολογίες θεωρούν τους ανθρώπους - πελάτες, προγραμματιστές, τους τελικούς χρήστες και γενικά τους συμμετέχοντες - ως το σημαντικότερο παράγοντα στην διαδικασία ανάπτυξης του λογισμικού.

Προσαρμοστικότητα: Κατά τη διάρκεια μιας επανάληψης, νέοι κίνδυνοι ενδέχεται να εμφανιστούν οι οποίοι απαιτούν ορισμένες δραστηριότητες που δεν είχαν σχεδιαστεί. Οι συμμετέχοντες σε μια ευέλικτη διαδικασία δεν φοβούνται τις αλλαγές. Καλωσορίζουν τις αλλαγές σε όλα τα στάδια του έργου. Αντιμετωπίζουν τις αλλαγές στις απαιτήσεις θετικά, γιατί σημαίνει ότι η ομάδα έχει κατανοήσει καλύτερα τις ανάγκες του πελάτη και συνεπώς γνωρίζει πώς να τις ικανοποιήσει και να καταλήξει σε ένα ολοκληρωμένο προϊόν [Fowler, 2005]. Στις ευέλικτες μεθοδολογίες, η πρόκληση δεν είναι να περιοριστούν οι αλλαγές που παρουσιάζονται, αλλά ο προσδιορισμός τρόπων ώστε να τύχουν καλύτερου χειρισμού. *«Οι εξωτερικές περιβαλλοντικές αλλαγές προκαλούν κρίσιμες παραλλαγές. Επειδή δεν μπορούμε να περιορίσουμε αυτές τις αλλαγές, η προσπάθεια ώστε να αντιμετωπιστούν με το λιγότερο δυνατό κόστος, είναι η μόνη βιώσιμη στρατηγική»* [Highsmith and Cockburn, 2004].

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

Εξισορρόπηση ευελιξίας και σχεδιασμού: Τα σχέδια είναι σημαντικά, αλλά το πρόβλημα είναι ότι τα έργα λογισμικού δεν μπορούν να προβλεφθούν με ακρίβεια, καθώς υπάρχουν πολλοί παράγοντες που πρέπει να ληφθούν υπόψη. Μια καλύτερη στρατηγική σχεδιασμού είναι να γίνονται λεπτομερή σχέδια για το άμεσο μέλλον, για τις επόμενες εβδομάδες, λιγότερο αναλυτικά σχέδια για τους επόμενους μήνες, και λιγότερα σχέδια πέρα από αυτό το χρονικό διάστημα. Μια από τις κύριες πηγές της πολυπλοκότητας των συστημάτων είναι η μη αντιστρεψιμότητα των αποφάσεων. Είναι σημαντικότερη η ευελιξία στην αλλαγή των αποφάσεων. Αντί να παρθεί μια σωστή απόφαση άμεσα, αναζητείται ένας τρόπος ώστε να αναβληθεί η απόφαση για αργότερα ή να παρθεί η απόφαση με τέτοιο τρόπο ώστε να εξασφαλιστεί το ενδεχόμενο αλλαγής της αργότερα χωρίς δυσκολία [Fowler, 2005].

Εμπειρική διαδικασία: Οι ευέλικτες μέθοδοι ανάπτυξης λογισμικού είναι εμπειρικές (ή μη γραμμικές) διαδικασίες. Οι διαδικασίες ορίζονται σαν ορισμένα σειριακά βήματα και παράγουν τα ίδια αποτελέσματα κάθε φορά που εκτελούνται. Η ανάπτυξη λογισμικού δεν μπορεί να θεωρηθεί σαν μια καθορισμένη διαδικασία γιατί πάρα πολλές αλλαγές προκύπτουν κατά τη διάρκεια ανάπτυξης του προϊόντος. Είναι εξαιρετικά απίθανο ότι οποιοδήποτε σύνολο προκαθορισμένων βημάτων θα οδηγήσουν σε ένα επιθυμητό, προβλέψιμο αποτέλεσμα, επειδή οι απαιτήσεις αλλάζουν, παρουσιάζονται αλλαγές στην χρησιμοποιούμενη τεχνολογία, άνθρωποι προστίθενται και αποσύρονται από την ομάδα ανάπτυξης, και ούτω καθεξής [Williams and Cockburn, 2003].

Αποκεντρωμένη προσέγγιση (decentralized approach): Η ενσωμάτωση ενός αποκεντρωμένου στυλ διαχείρισης μπορεί να επηρεάσει σοβαρά ένα σύστημα λογισμικού, διότι θα μπορούσε να εξοικονομηθεί πολύς χρόνος από ότι σε μια διαδικασία όπου οι λήψεις αποφάσεων θα περιορίζονταν σε ένα πρόσωπο. Η ευέλικτη ανάπτυξη λογισμικού επιτρέπει την λήψη αποφάσεων από τους προγραμματιστές. Αυτό δεν σημαίνει ότι οι προγραμματιστές θα αναλάβουν το ρόλο της διοίκησης. Η διοίκηση εξακολουθεί να είναι αναγκαία για την αντιμετώπιση των εμποδίων που παρουσιάζονται κατά την πορεία της ανάπτυξης. Ωστόσο, η διαχείριση αναγνωρίζει την ικανότητα των μελών της ομάδας να παίρνουν αποφάσεις χωρίς την άδειά τους.

Απλότητα: Οι ευέλικτες ομάδες λαμβάνουν πάντα την πιο απλή απόφαση που μπορεί να εξυπηρετήσει τους στόχους τους. Οι ευέλικτες ομάδες δεν προβλέπουν προβλήματα του αύριο και προσπαθούν να τα επιλύσουν σήμερα. Η ανάγκη για απλότητα, παρουσιάζεται ώστε να είναι εύκολο να αλλάξει το σύστημα, αν χρειαστεί σε μια μεταγενέστερη φάση. Ποτέ δεν παράγεται περισσότερο από ό,τι είναι απαραίτητο.

Συνεργασία: Οι ευέλικτες μέθοδοι περιλαμβάνουν τα σχόλια των πελατών σε τακτά χρονικά διαστήματα. Ο πελάτης του λογισμικού συνεργάζεται στενά με την ομάδα ανάπτυξης, παρέχοντας τη συχνή ανατροφοδότηση για το αναπτυσσόμενο προϊόν. Επίσης, η συνεχής συνεργασία μεταξύ των μελών της ομάδας είναι απαραίτητη. Λόγω της αποκεντρωμένης προσέγγισης, που αναφέραμε πιο πάνω, των ευέλικτων πρακτικών, η συνεργασία ενθαρρύνει τη συζήτηση. Όπως περιγράφει ο Fowler, «οι ευέλικτες ομάδες δεν μπορούν να υπάρξουν μόνο με την περιστασιακή επικοινωνία. Χρειάζεται συνεχής πρόσβαση και εμπλοκή στις διαδικασίες της επιχείρησης για την οποία αναπτύσσεται το σύστημα» [Fowler, 2005].

Μικρές αυτό-οργανώμενες ομάδες: Μια ευέλικτη ομάδα είναι μια ομάδα που οργανώνεται μόνη της. Οι ευθύνες κοινοποιούνται στην ομάδα ως σύνολο, και έπειτα η ομάδα καθορίζει τον καλύτερο τρόπο για την επίτευξή τους. Συζητούν και επικοινωνούν μεταξύ τους για όλες τις πτυχές του έργου. Αυτός, ίσως, να είναι ο λόγος που οι ευέλικτες μεθοδολογίες λειτουργούν καλύτερα με μικρές ομάδες.

Κεφάλαιο 4

Μεθοδολογία Scrum

4.1 Εισαγωγή

Όπως και οι υπόλοιπες ευέλικτες μεθοδολογίες που εξετάσαμε στο προηγούμενο κεφάλαιο, έτσι και η μεθοδολογία Scrum είναι μια επαναληπτική και αυξητική διαδικασία ανάπτυξης λογισμικού. Η προσέγγιση αυτή αρχικά κατασκευάστηκε με στόχο την ανάπτυξη έργων λογισμικού. Ωστόσο, αργότερα αποδείχθηκε ότι μπορεί να χρησιμοποιηθεί για την γενικότερη διαχείριση έργων.

Το 1986 οι Hirotaka Takeuchi και Ikujiro Nonaka περιέγραψαν μια καινούρια μεθοδολογία η οποία βελτιώνει αισθητά την ταχύτητα ανάπτυξης και τον βαθμό ευελιξίας στην διαδικασία οικοδόμησης ενός εμπορικού προϊόντος [Takeuchi and

Nonaka, 1986]. Την αποκάλεσαν ολιστική προσέγγιση (holistic approach). Αργότερα, το 1990 οι DeGrace και Stahl έκαναν αναφορά σε αυτή την προσέγγιση ως Scrum. Ένας όρος που επίσης αναφέρουν οι Takeuchi και Nonaka στο άρθρο τους. Η διαδικασία ανάπτυξης του προϊόντος, σε όλες τις επί μέρους φάσεις, που συνήθως επικαλύπτονται, εκτελείται από μία λειτουργική «ομάδα» που συγκρίνεται με αυτήν του ράγκμπι (από όπου και το όνομα). Δηλαδή, όπως και στο ράγκμπι, όλη η ομάδα σαν μια οντότητα προσπαθεί να διανύσει την απόσταση (δημιουργία του προϊόντος), περνώντας την «μπάλα» μπρός-πίσω [Σφέτσος, 2007]. Στις αρχές της δεκαετίας του 1990, ο Ken Schwaber χρησιμοποίησε στην εταιρία του την διαδικασία αυτή που ονομαζόταν Scrum. Ο Jeff Sutherland, με τον John Jeff Scumniotales και τον McKenna ανέπτυξαν μια παρόμοια προσέγγιση στο Easel Corporation, και ήταν οι πρώτοι που αναφέρθηκαν επίσημα σε αυτή την μεθοδολογία χρησιμοποιώντας τον όρο Scrum [Sutherland, 2004]. Το 2001 ο Sutherland εργάστηκε μαζί με τον Mike Beedle για να καταγράψουν την μεθοδολογία αυτή στο βιβλίο «Agile Software Development with Scrum» [Schwaber and Beedle, 2002].

Μέχρι στιγμής το Scrum, είναι η πιο δημοφιλής μέθοδος από τις ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού. Χρησιμοποιείται ευρέως από μεγάλες και μικρές εταιρίες, μεταξύ των οποίων οι Yahoo, Microsoft, Google, Lockheed Martin, Motorola, SAP, Cisco, GE, CapitalOne και US Federal Reserve. Πολλές ομάδες που χρησιμοποιούν την μέθοδο αυτή, αναφέρουν σημαντικά οφέλη και σε ορισμένες περιπτώσεις ολοκληρωτική βελτίωση της παραγωγικότητας.

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

κώδικα του συστήματος που είχαν συμφωνηθεί ανά κύκλο. Ένα σημαντικό σημείο που θα πρέπει να αναφερθεί είναι η καθημερινή συνεδρίαση των μελών της ομάδας ανάπτυξης με σκοπό την επίτευξη όσο το δυνατόν καλύτερης συνεργασίας και καλύτερου συντονισμού των εργασιών τους [Σφέτσος, 2007].

4.2 Ανασκόπηση της μεθοδολογίας

Όπως αναφέραμε το Scrum είναι μια επαναληπτική και αυξητική προσέγγιση για την ανάπτυξη έργων. Ο σκοπός είναι να ελέγχεται ο κίνδυνος καθώς η μέθοδος επιτρέπει την προσαρμογή στις αλλαγές που προκύπτουν. Στα παραδοσιακά μοντέλα ανάπτυξης λογισμικού, οι απαιτήσεις μπορεί να είναι προκαθορισμένες. Αντίθετα, το Scrum, τονίζει τη μεταβαλλόμενη φύση των απαιτήσεων των πελατών.

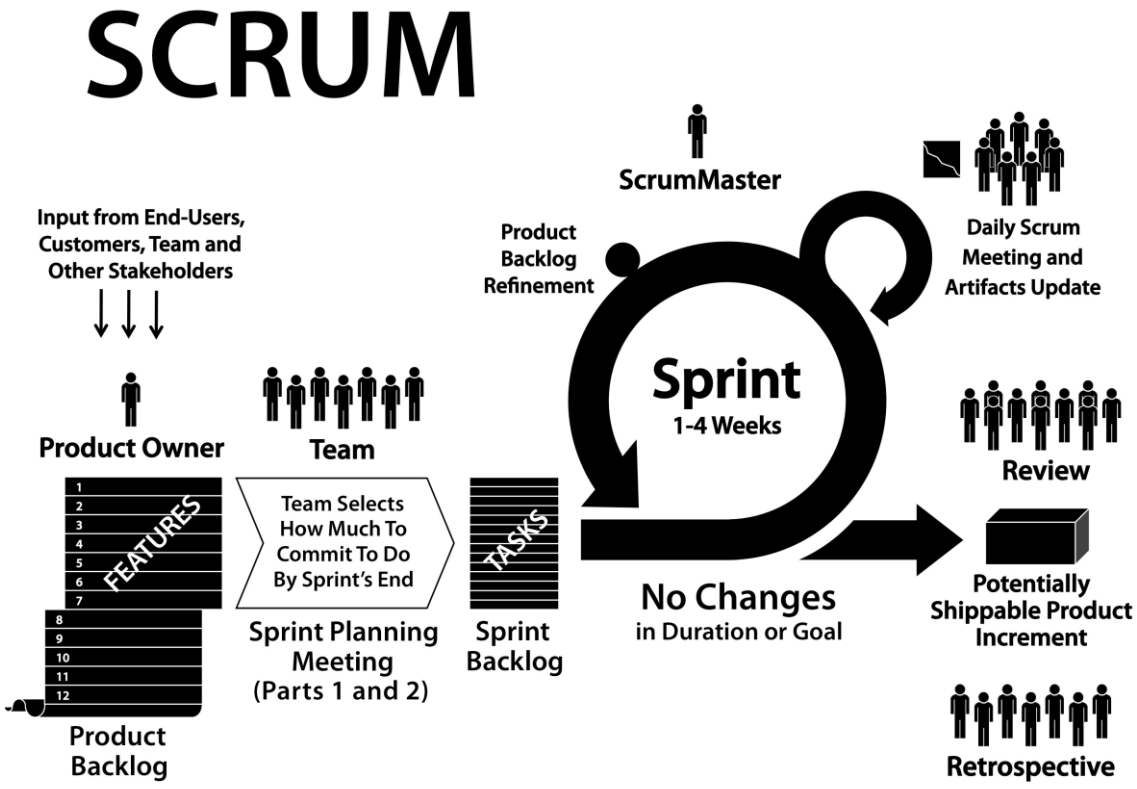
Η ανάπτυξη δομείται και χωρίζεται σε επαναληπτικούς κύκλους εργασίας, οι οποίοι ονομάζονται **Sprints**. Οι επαναλήψεις δεν διαρκούν περισσότερο από ένα μήνα και εκτελούνται σειριακά χωρίς οποιεσδήποτε χρονικές καθυστερήσεις μεταξύ τους. Τα Sprints έχουν συγκεκριμένο χρονοδιάγραμμα το οποίο ποτέ δεν παρακάμπτεται, έστω και αν η καθορισμένη εργασία δεν έχει ολοκληρωθεί. Στην αρχή κάθε Sprint η ομάδα επιλέγει τα στοιχεία, τα οποία αποτελούν τις απαιτήσεις των πελατών, από μια λίστα στην οποία οι απαιτήσεις είναι ταξινομημένες βάση προτεραιότητας (Backlog).

Η ομάδα δεσμεύεται να υλοποιήσει τα στοιχεία που επιλέγηκαν μέχρι το τέλος του Sprint. Κατά τη διάρκεια του Sprint, τα επιλεγόμενα στοιχεία δεν αλλάζουν. Κάθε μέρα η ομάδα οργανώνει σύντομες συγκεντρώσεις για να επιθεωρήσει την πρόοδο της, και να προσδιοριστούν τα επόμενα βήματα που χρειάζονται για να ολοκληρώσουν την υπόλοιπη εργασία. Στο τέλος του Sprint, η ομάδα παρουσιάζει το παραγόμενο αποτέλεσμα στον πελάτη. Με τον τρόπο αυτό η ομάδα ανάπτυξης του συστήματος, συγκεντρώνει σχόλια, διορθώσεις και παραλήψεις ώστε να ενσωματωθούν στο επόμενο Sprint. Η μεθοδολογία Scrum δίνει έμφαση στο προϊόν που είναι πραγματικά λειτουργικό στο τέλος κάθε επανάληψης. Στην περίπτωση του λογισμικού αυτό σημαίνει κώδικας που ενσωματώνεται πλήρως και έχει δοκιμαστεί. Για τον λόγο αυτό τα μέλη της ομάδας θα πρέπει να έχουν μια κοινή κατανόηση για το πότε μια εργασία θεωρείται ολοκληρωμένη (Done). Για την ομάδα αυτό περιγράφεται με τον όρο

«**Definition of Done**» και χρησιμοποιείται για να αξιολογείται πότε μια εργασία έχει ολοκληρωθεί και μπορεί να ενσωματωθεί στο προσαυξημένο προϊόν. Με τον τρόπο εξασφαλίζεται η ποιότητα για το τελικό προϊόν [Schwaber and Sutherland, 2011]. Το Definition of Done είναι μια συμφωνία που γίνεται μεταξύ της ομάδας Scrum και του Product Owner.

Το Definition of Done είναι ένας απλός κατάλογος δραστηριοτήτων (σύνταξη κώδικα, σχόλια κωδικοποίησης, δοκιμές ολοκλήρωσης, σημειώσεις έκδοσης, τα έγγραφα σχεδιασμού, κ.λπ.) που προσθέτουν αποδεδειγμένη αξία στο προϊόν. Επιτρέπει στην ομάδα να επικεντρωθεί σε αυτά που πρέπει να ολοκληρωθούν για την δημιουργία του λογισμικού, καθώς παράλληλα αποτρέπει την υλοποίηση περιττών δραστηριοτήτων που περιπλέκουν την διαδικασία ανάπτυξης λογισμικού.

Οι διαδικασίες, τα χαρακτηριστικά και οι ρόλοι που εφαρμόζονται στο Scrum, συνοψίζονται στο Σχήμα 4.1 το οποίο εξηγείται και περιγράφεται στην πορεία.



Σχήμα 4.1 Διαδικασία Scrum [Deemer et al., 2010]

Ένα σημαντικό θέμα στο Scrum είναι ο έλεγχος και η προσαρμοστικότητα. Δεδομένου ότι η ανάπτυξη συνεπάγεται αναπόφευκτα, μάθηση, καινοτομία και απρόσμενες αλλαγές, το Scrum δίνει έμφαση στην ανάπτυξη ενός μικρού μέρους του όλου συστήματος, την επιθεώρηση του παραγόμενου προϊόντος και την κατάλληλη αναπροσαρμογή του. Η διαδικασία αυτή επαναλαμβάνεται συνεχώς μέχρι το σύστημα να ολοκληρωθεί. Με τον τρόπο αυτό η ευελιξία προσαρμογής στις μεταβαλλόμενες απαιτήσεις των πελατών ενισχύεται.

4.3 Ρόλοι και ευθύνες

Στο Scrum παρουσιάζονται οι εξής τρείς βασικοί ρόλοι, όπως αυτοί περιγράφονται στη συνέχεια:

- **Ο ιδιοκτήτης του προϊόντος (The Product Owner)**
- **Η ομάδα (The Team)**
- **Ο διαχειριστής ή ειδικός του Scrum (The Scrum Master)**

Ο **Product Owner** είναι υπεύθυνος για την οικονομική αποδοτικότητα (The profitability of the project, ROI) της ανάπτυξης του προϊόντος [Hundermark, 2009], εντοπίζοντας τα χαρακτηριστικά του προϊόντος, μεταφράζοντας τα σε μια λίστα απαιτήσεων, αποφασίζοντας ποια από αυτά τα χαρακτηριστικά θα πρέπει να είναι στην κορυφή της λίστας για το επόμενο Sprint και συνεχώς αναπροσαρμόζει και καθορίζει την προτεραιότητα της λίστας αυτής. Ο Product Owner κατέχει καθοριστικό ρόλο στην ανάπτυξη του συστήματος. Σε ορισμένες περιπτώσεις ο Product Owner και ο πελάτης είναι το ίδιο πρόσωπο. Ο ρόλος που κατέχει ένας Product Owner θα μπορούσε να παραλληλιστεί με τον ρόλο ενός Product Manager, ωστόσο ο ρόλος ενός παραδοσιακού Product Manager διαφέρει. Είναι σημαντικό να σημειωθεί ότι στο Scrum δεν υπάρχει ένα και μόνο ένα άτομο που λειτουργεί ως Product Owner, ο οποίος επίσης είναι υπεύθυνος για την αποδοτικότητα του συστήματος.

Η **ομάδα** (The team), χτίζει το προϊόν που σε κάθε επανάληψη υποδεικνύει ο Product Owner. Η ομάδα ανάπτυξης απαρτίζεται από άτομα με διαφορετικές λειτουργικές γνώσεις που ωστόσο εργάζονται για έναν κοινό στόχο (Cross-functional). Κατέχουν όλες τις απαραίτητες γνώσεις ώστε να είναι σε θέση να παραδίδουν κάθε φορά το

προϊόν το οποίο είναι αποτέλεσμα ενός Sprint. Επιπλέον, είναι αυτό-οργανώμενες ομάδες (Self-organizing) με πολύ υψηλό βαθμό αυτονομίας και υπευθυνότητας. Η ομάδα αποφασίζει το πώς θα αναπτύξει ένα προϊόν χωρίς οποιεσδήποτε υποδείξεις από τρίτους. Στην βιομηχανία λογισμικού, τα μέλη της ομάδας είναι γνωστά ως «γουρούνια» και οι υπόλοιποι αποκαλούνται «κότες». Οι χαρακτηρισμοί αυτοί προέρχονται από ένα αστείο που αφορά ένα γουρούνι και μια κότα που επρόκειτο να ανοίξουν ένα εστιατόριο το οποίο θα ονομαζόταν «Ham and eggs». Το γουρούνι, έκανε δεύτερες σκέψεις γιατί θα «δεσμευόταν» ολοκληρωτικά. Η κότα, αντίθετα, απλά θα εμπλεκόταν.

Μια ομάδα Scrum συνήθως απαρτίζεται από εφτά άτομα και για την ανάπτυξη ενός έργου λογισμικού η ομάδα μπορεί να περιλαμβάνει άτομα με δεξιότητες στην ανάλυση, ανάπτυξη, δοκιμή, το σχεδιασμό διεπαφής, τις βάσεις δεδομένων, την αρχιτεκτονική, και ούτω καθεξής. Η ομάδα αναπτύσσει το προϊόν και δίνει ιδέες στον Product Owner για το πώς να βελτιώσει την ανάπτυξη. Οι ομάδες είναι περισσότερο παραγωγικές και αποτελεσματικές εάν όλα τα μέλη είναι αφοσιωμένα στην παραγωγή του προϊόντος κατά την διάρκεια ενός Sprint και αποφεύγονται ανακατατάξεις στην σύνθεση της ομάδας. Οι ομάδες με πολλά άτομα συνήθως οργανώνονται σε πολλαπλές μικρότερες ομάδες, η κάθε μια από τις οποίες θα αναλάβει την ανάπτυξη διαφορετικών χαρακτηριστικών του συστήματος ενώ πάντα θα υπάρχει επικοινωνία και συντονισμός μεταξύ των ομάδων. Οι ομάδες αναλαμβάνουν, ώστε ένα προϊόν να περάσει από όλες τις φάσεις ανάπτυξης (σχεδιασμός, ανάλυση, προγραμματισμό και έλεγχο) όπως και στις περισσότερες μεθοδολογίες ανάπτυξης.

Ο **Scrum Master** βοηθά τα μέλη της ομάδας ανάπτυξης ώστε να μάθουν και να εφαρμόσουν καλύτερα τις πρακτικές που ακολουθούνται στην διαδικασία του Scrum. Κάνει οτιδήποτε μπορεί ώστε να βοηθήσει την ομάδα και τον Product Owner. Είναι υπεύθυνος για τη διασφάλιση της ορθής ανάπτυξης του έργου σύμφωνα πάντα με τις πρακτικές, τις αξίες και τους κανόνες που έχουν καθοριστεί κατά το σχεδιασμό του. Ο Scrum Master κατευθύνει τους υπόλοιπους συμμετέχοντες, προ την σωστή εφαρμογή του Scrum και παράλληλα διασφαλίζει ότι όλοι έχουν κατανοήσει και ακολουθούν τις πρακτικές του Scrum. Από την στιγμή που τα εμπόδια και οι δυσκολίες είναι αναπόφευκτα, είναι σημαντικό να υπάρχει κάποιος που να μπορεί να δώσει ουσιαστικές

λύσεις σε αυτά έτσι ώστε η ομάδα ανάπτυξης να διατηρήσει όσο το δυνατόν ψηλότερα το επίπεδο παραγωγικότητάς της. Συν των άλλων, αυτό τον ρόλο καλείται να καλύψει ο Scrum Master. Κανονικά, θα πρέπει να υπάρχει ένας Scrum Master, ο οποίος θα έχει αναλάβει αποκλειστικά τον ρόλο αυτό. Ωστόσο σε μικρές ομάδες ενδέχεται κάποιο μέλος της ομάδας να αναλάβει το ρόλο αυτό.

Αξιοσημείωτο, είναι το γεγονός πώς δεν υπάρχει ο ρόλος του Project Manager στην διαδικασία του Scrum, για τον λόγο ότι δεν χρειάζεται αφού οι παραδοσιακές υπευθυνότητες ενός Project Manager έχουν διαμοιραστεί ανάμεσα στους τρείς ρόλους που περιγράφηκαν πιο πάνω.

4.4 Διαδικασία

Αρχικά ο Product Owner θα πρέπει να προγραμματίσει την δομή ανάπτυξης. Αυτό σημαίνει πώς θα πρέπει να ορίσει μια ιεραρχική λίστα με τα χαρακτηριστικά που πρόκειται να αναπτυχθούν. Η λίστα αυτή στην ουσία περιλαμβάνει τις απαιτήσεις των χρηστών και ονομάζεται **Product Backlog** (κατάλογος των ανεκτέλεστων εργασιών του προϊόντος). Η λίστα υπάρχει και εξελίσσεται καθ’ όλη την διάρκεια ανάπτυξης του συστήματος. Είναι ένας κατάλογος που περιέχει τις γνωστές, μέχρι την παρούσα περίοδο, απαιτήσεις του συστήματος. Οι απαιτήσεις μπορεί να προέλθουν από τον πελάτη, τις πωλήσεις και το τμήμα μάρκετινγκ, την υποστήριξη πελατών ή τους ίδιους τους υπεύθυνους για την ανάπτυξη του λογισμικού. Οι απαιτήσεις κατατάσσονται ανάλογα με την προτεραιότητα τους και υπολογίζεται η προσπάθεια που απαιτείται για την υλοποίηση τους. Ο κατάλογος των απαιτήσεων ενημερώνεται και αναθεωρείται σε κάθε επανάληψη με νέα και πιο λεπτομερή στοιχεία, καθώς επίσης και με ακριβέστερες εκτιμήσεις για την αναγκαία προσπάθεια και τις νέες προτεραιότητες που προκύπτουν. Στο Σχήμα 4.2 παρουσιάζεται ένα παράδειγμα μιας τέτοιας λίστας.

					New Estimates of Effort Remaining as of Sprint...					
Item	Details (wiki URL)	Priority	Estimate of Value	Initial Estimate of Effort	1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5						
As a buyer, I want to remove a book in a shopping cart	...	2	6	2						
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13						
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20						
Upgrade all servers to Apache 2.2.3	...	5	5	13						
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3						
As a shopper, I want to create and save a wish list	...	7	7	40						
As a shopper, I want to to add or delete items on my wish list	...	8	4	20						

Σχήμα 4.2 Ενδεικτικό παράδειγμα ενός Product Backlog [Deemer et al., 2010]

Το Product Backlog περιλαμβάνει κυρίως τα χαρακτηριστικά του συστήματος όπως αυτά ορίζονται από τον πελάτη. Ωστόσο, όπως φαίνεται και στο Σχήμα 4.2, μπορεί να περιλαμβάνει και βελτιώσεις ή ελαττώματα (π.χ «rework the transaction processing module to make it scalable») που θα πρέπει να τύχουν προσοχής στο σύστημα. Οι αλλαγές αυτές ορίζονται από την ομάδα ανάπτυξης του προϊόντος. Τα χαρακτηριστικά που περιλαμβάνονται στη λίστα, μπορούν να διατυπωθούν με οποιονδήποτε σαφή τρόπο, όπως για παράδειγμα μέσω σεναρίων Use cases ή User stories, όπως και στον ακραίο προγραμματισμό.

Το υποσύνολο των εργασιών ή χαρακτηριστικών του Product Backlog που επιλέγεται για μια επανάληψη, ονομάζεται **Release Backlog.** Όπως αναφέραμε η σειρά με την οποία επιλέγονται τα χαρακτηριστικά προς υλοποίηση εξαρτάται από την προτεραιότητα που δίνεται σε αυτά. Αυτή είναι και η κύρια ευθύνη του Product Owner.

Το Product Backlog ενημερώνεται συνεχώς ώστε να περιλαμβάνει τις αλλαγές στις απαιτήσεις και ανάγκες του πελάτη. Η ομάδα Scrum ενημερώνει τον Product Owner όσο αφορά την εκτιμώμενη προσπάθεια που απαιτείται για την υλοποίηση κάθε χαρακτηριστικού που περιλαμβάνεται στην λίστα. Επιπλέον, ο Product Owner είναι υπεύθυνος για την ανάθεση επιχειρηματικής αξίας σε κάθε μεμονωμένο χαρακτηριστικό. Στο σημείο αυτό μπορεί να εμπλακεί και ο Scrum Master. Έχοντας υπόψη του τις δύο αυτές εκτιμήσεις (προσπάθεια και αξία), και επιπρόσθετα δίνοντας έμφαση σε επικείμενους κινδύνους, ο Product Owner καθορίζει την ιεράρχηση του Product Backlog. Το σημαντικότερο κριτήριο στην διαδικασία ανάθεσης

προτεραιότητας, είναι να επιλέγονται πρώτα χαρακτηριστικά με μεγάλη αξία και τα οποία απαιτούν λιγότερη προσπάθεια για να υλοποιηθούν.

Το Scrum δεν καθορίζει τυποποιημένες τεχνικές για το πώς θα γίνει η ιεράρχηση του Product Backlog. Μια κοινή τεχνική είναι εκτιμηθούν οι πιο σημαντικοί παράγοντες (απαιτούμενη προσπάθεια, πολυπλοκότητα, αβεβαιότητα) και να αποδοθεί μια τιμή-βάρος σε κάθε ένα από τους παράγοντες αυτούς. Αυτό συμβαίνει για κάθε χαρακτηριστικό ξεχωριστά και οι τιμές που ανατίθενται ονομάζονται Βαθμοί «Points». Ωστόσο, υπάρχουν διάφορες τεχνικές που καθορίζουν πώς γίνεται η ιεράρχηση. Ο Cohn [Cohn, 2008] προσδιόρισε ορισμένες τεχνικές που μπορούν να χρησιμοποιηθούν. Πρώτον, η ανάλυση «Kano», η οποία περιλαμβάνει την συμμετοχή των χρηστών και που θα αποκαλύψει ποια χαρακτηριστικά είναι σημαντικότερα και τα οποία αυξάνουν την ικανοποίηση των χρηστών, και ποια είναι τα χαρακτηριστικά που οι χρήστες δεν γνωρίζουν, ακόμη, ότι χρειάζονται. Δεύτερον, η προτεραιότητα μπορεί να αποδοθεί με κριτήρια επιλογής κάποιων εμπειρογνωμόνων. Τρίτον, μπορεί να ανατεθούν βάρη στα χαρακτηριστικά με ορισμένα κριτήρια που θα επιλεγούν (αυτό είναι η πιο κοινή τεχνική και η οποία περιγράφηκε πιο πάνω).

Με την εμπειρία που αποκτά μια ομάδα, μπορεί να εκτιμήσει πόση δουλειά απαιτείται για κάθε Sprint. Καθορίζεται ένας μέσος όρος από points για κάθε Sprint. Για παράδειγμα αν ο μέσος όρος αυτός οριστεί στα 30, τότε αυτό σημαίνει πώς το άθροισμα των points των χαρακτηριστικών που θα επιλεγούν για ένα Release Backlog, δεν θα πρέπει να υπερβαίνει τα 30. Έτσι μπορεί να εκτιμηθεί ο χρόνος που χρειάζεται για την ολοκλήρωση ενός Sprint, ή να εκτιμηθεί πόσα χαρακτηριστικά μπορούν να υλοποιηθούν σε ένα ορισμένο χρονικό διάστημα. Ο μέσος όρος αυτός ονομάζεται **Velocity** (ταχύτητα της ομάδας) και εκφράζεται με τον ίδιο τρόπο όπως και τα εκτιμώμενα χαρακτηριστικά του Product Backlog. Το Velocity εμπλέκεται στον αριθμό των χαρακτηριστικών που θα επιλεγούν για το Product Backlog. Η πληροφορία αυτή είναι σημαντική για την ομάδα γιατί εάν το Velocity είναι γνωστό, τότε μπορεί να βοηθήσει σημαντικά την ομάδα στον προγραμματισμό της. Ο Product Owner στη συνέχεια να χρησιμοποιεί το Velocity για να προβλέψει το ρυθμό με τον οποίο η ομάδα είναι σε θέση να παραδίδει τμήματα του συστήματος.

Τα χαρακτηριστικά που περιλαμβάνονται στο Product Backlog μπορεί να διαφέρουν αρκετά ως προς το μέγεθος και την απαιτούμενη προσπάθεια. Τα μεγαλύτερα από αυτά, σπάζουν σε επιμέρους μικρότερα χαρακτηριστικά κατά την διάρκεια της συνεδρίασης προγραμματισμού του Sprint (Sprint Planning Meeting). Παρόμοια μικρότερα χαρακτηριστικά μπορούν να ενοποιηθούν. Τα χαρακτηριστικά που θα επιλεγούν για τα άμεσα επόμενα Sprints, θα πρέπει να είναι λεπτομερώς καθορισμένα χωρίς οποιεσδήποτε ασάφειες ώστε να είναι κατανοητά από την ομάδα.

4.4.1 Προγραμματισμός ενός Sprint (Sprint Planning)

Με την έναρξη κάθε Sprint, πραγματοποιείται το **Sprint Planning Meeting** (συνεδρίαση προγραμματισμού του Sprint). Η συνεδρίαση αυτή χωρίζεται σε δύο μέρη. Τα δύο μέρη του Sprint Planning Meeting απαντούν στις ακόλουθες ερωτήσεις, αντίστοιχα [Schwaber and Sutherland, 2011]:

- Τι θα παραδοθεί στον πελάτη με το τέλος του Sprint;
- Πώς θα προγραμματιστεί η δουλειά ώστε να επιτευχθεί η παράδοση;

Το πρώτο μέρος εστιάζει στο τι θα υλοποιηθεί κατά το Sprint. Ο Product Owner μαζί με την ομάδα Scrum και πάντοτε με την συμμετοχή του Scrum Master, επανεξετάζουν τα υψηλής προτεραιότητας χαρακτηριστικά που βρίσκονται στο Product Backlog. Συζητούν για τους στόχους και την λειτουργία αυτών των απαιτήσεων που πρόκειται να υλοποιηθούν. Το πρώτο μέρος του Sprint Planning Meeting επικεντρώνεται στην κατανόηση του τι ακριβώς ζητά και αναμένει ο Product Owner.

Το δεύτερο μέρος της συνεδρίασης επικεντρώνεται στον λεπτομερή προγραμματισμό των εργασιών που πρέπει να γίνουν ώστε η ομάδα να υλοποιήσει τα χαρακτηριστικά του Product Backlog. Οι εργασίες αυτές ονομάζονται **Tasks.** Η ομάδα επιλέγει τα χαρακτηριστικά από το Product Backlog και δεσμεύεται να τα ολοκληρώσει έως το τέλος του Sprint, ξεκινώντας από την κορυφή της λίστας, δηλαδή, ξεκινώντας με την υλοποίηση των χαρακτηριστικών με μεγαλύτερη προτεραιότητα. Η ομάδα, όπως αναφέραμε, αυτό-οργανώνεται. Τα μέλη αποφασίζουν μεταξύ τους το ποσοστό της εργασίας που πρόκειται να ολοκληρώσουν αντί να τους ανατεθεί από τον Product Owner. Αυτή είναι μια βασική πρακτική στο Scrum. Επιπλέον η ομάδα έχει την

δυνατότητα να αλλάξει την σειρά των χαρακτηριστικών στο Product Backlog, εφόσον κρίνει πως κάτι χαμηλότερης προτεραιότητας ταιριάζει με κάτι υψηλότερης προτεραιότητας και έτσι είναι προτιμότερο η υλοποίηση των στοιχείων αυτών να γίνει διαδοχικά.

Το Sprint Planning Meeting συνήθως διαρκεί μερικές ώρες. Κατά το δεύτερο μέρος της συνεδρίας, καθορίζονται πόσες εργασιακές ώρες διαθέτει το κάθε μέλος της ομάδας. Συνήθως κάθε μέλος δουλεύει περίπου από 4 μέχρι 6 ώρες την ημέρα. Έπειτα συναθροίζονται οι ώρες κάθε μέλους ξεχωριστά και προκύπτει ο συνολικός χρόνος σε ώρες κατά τον οποίο θα πρέπει να ολοκληρωθεί το Sprint. Εφόσον ο χρόνος έχει καθοριστεί, τότε η ομάδα υπολογίζει πόσα χαρακτηριστικά από το Product Backlog μπορεί να υλοποιήσει μέσα στα χρονικά αυτά πλαίσια και πώς θα υλοποιηθούν αυτά. Η ομάδα ξεκινά με την υλοποίηση των χαρακτηριστικών που έχουν μεγαλύτερη προτεραιότητα (όπως αυτά καθορίστηκαν από τον Product Owner), και αποσυνθέτει το κάθε χαρακτηριστικό σε επιμέρους εργασίες (Tasks) οι οποίες συγκεντρώνονται και καταγράφονται σε μια λίστα η οποία ονομάζεται **Sprint Backlog** (Σχήμα 4.3). Στο τέλος κάθε συνεδρίας, θα πρέπει να προκύψει ένας τέτοιος κατάλογος με όλες τις εργασίες που πρέπει να γίνουν και τις εκτιμώμενες ώρες εργασίας που απαιτούνται για κάθε μια από αυτές.

				New Estimates of Effort Remaining at end of Day...					
Product Backlog Item	**Sprint Task**	**Volunteer**	**Initial Estimate of Effort**						
As a buyer, I want to place a book in a shopping cart	modify database	Sanjay	5						
	create webpage (UI)	Jing	3						
	create webpage (Javascript logic)	Tracy & Sam	2						
	write automated acceptance tests	Sarah	5						
	update buyer help webpage	Sanjay & Jing	3						
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5						
	complete machine order for pRank		3						
	change DCP and reader to use pRank http API		5						
	. . .	. . .	. . .						
		Total (person hours)	50						

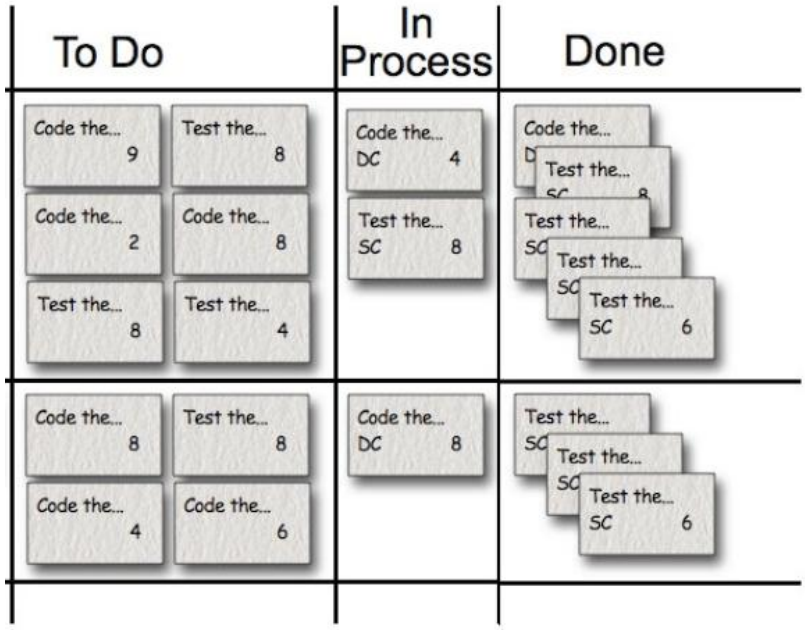
Σχήμα 4.3 Ενδεικτικό παράδειγμα ενός Sprint Backlog [Deemer et al., 2010]

Στο Σχήμα 4.3 παρουσιάζεται ένα παράδειγμα ενός Sprint Backlog. Στην πρώτη στήλη φαίνονται τα χαρακτηριστικά το οποία επιλέγηκαν από το Product Backlog και τα οποία διασπάστηκαν σε επιμέρους Tasks, όπως αυτά φαίνονται στην δεύτερη στήλη.

Τα Tasks ανατίθενται σε μέλη της ομάδας. Επιπλέον παρουσιάζεται η εκτίμηση που απαιτείται για την προσπάθεια ολοκλήρωσης του κάθε Task. Στο συγκεκριμένο παράδειγμα, η συνολική εκτίμηση για την προσπάθεια που απαιτείται είναι 50 ώρες [Deemer et al., 2010].

Στο Scrum, οι συμμετέχοντες συνεργάζονται ανεξαρτήτως της ειδικότητας που κατέχει ο κάθε ένας. Με άλλα λόγια τα μέλη της ομάδας βοηθά ο ένας τον άλλο. Συνεπώς, κατά το Sprint Planning, δεν είναι απαραίτητο αλλά ούτε και σκόπιμο για τα μέλη της ομάδας να επιλέξουν να υλοποιήσουν ο κάθε ένας τα Tasks που μπορούν να κάνουν καλύτερα ή που έγκεινται στην ειδικότητα τους. Με τον τρόπο αυτό τα άτομα της ομάδας έχουν την δυνατότητα να διευρύνουν τις γνώσεις και τις ικανότητες τους σε διάφορους τομείς.

Ακόμη και αν η ομάδα χρησιμοποιεί ειδικό λογισμικό για τη διαχείριση των Sprint, είναι επωφελής να χρησιμοποιηθεί ένας πιο φυσικός τρόπο για την διαχείριση των Tasks. Αυτό επιτυγχάνεται με την χρήση ενός πίνακα εργασιών **(Task board)** στον οποίο καταγράφονται οι εργασίες ενός Sprint σε μορφή σημειώσεων (Σχήμα 4.4). Αποτελεί μια φυσική αναπαράσταση του καταλόγου των εργασιών που η ομάδα έχει δεσμευθεί να υλοποιήσει κατά τη διάρκεια του τρέχοντος Sprint. Κάθε λευκό χαρτί στο ταμπλό αποτελεί μια ευρύτερη περιοχή των εργασιών. Κάθε Task είναι γραμμένο σε ένα αυτοκόλλητο, συμπεριλαμβανομένης της εκτιμώμενης προσπάθειας που απαιτείται, π.χ., σε ώρες. Η πρώτη στήλη στον πίνακα περιλαμβάνει τις εργασίας που δεν έχουν ελεγχθεί ακόμη (Not checked out ή To do), δηλαδή τις εργασίες για τις οποίες δεν εργάζεται ακόμη κανένας. Η δεύτερη στήλη περιλαμβάνει τις εργασίες για τις οποίες δουλεύει είδη η ομάδα και συνεπώς βρίσκονται σε εξέλιξη (Checked out ή In process). Τέλος, η τρίτη στήλη είναι για τις εργασίες που έχουν ολοκληρωθεί (Done ή Completed). Έτσι, τα αυτοκόλλητα μετακινούνται από τα αριστερά προς τα δεξιά [Kniber, 2007].



Σχήμα 4.4 Sprint Task Board [Hundermark, 2009]

Ο πίνακας αναπαριστά την δουλειά που έχει προγραμματιστεί για ένα Sprint και την τρέχουσα κατάστασή της προόδου.

Ένας από τους βασικούς κανόνες του Scrum είναι ότι όταν η ομάδα έχει είδη ξεκινήσει με την υλοποίηση ενός Sprint, τότε τυχόν προσθήκες ή αλλαγές θα πρέπει να αναβληθούν μέχρι το επόμενο Sprint. Αυτό σημαίνει ότι εάν κατά την διάρκεια ενός Sprint ο Product Owner αποφασίσει ότι υπάρχει ένα νέο στοιχείο που θα ήθελε η ομάδα να υλοποιήσει, τότε δεν μπορεί να κάνει την προσθήκη ή την αλλαγή μέχρι την έναρξη του επόμενου Sprint. Ωστόσο, ενδέχεται ένα Sprint να τερματιστεί εφόσον προκύψουν νέα δεδομένα τα οποία αλλάζουν σημαντικά τις προτεραιότητες και σημαίνει ότι η ομάδα σπαταλάει πολύτιμο χρόνο εάν συνεχίσει να εργάζεται στο τρέχων Sprint. Τότε πραγματοποιείται ξανά ένα Sprint Planning Meeting και καθορίζεται ένα καινούριο Sprint. Το ενδεχόμενο να συμβεί αυτό είναι βέβαια μικρό γιατί το Scrum είναι καλά δομημένο ώστε να προστατεύει την ομάδα από την αλλαγή των στόχων κατά την διάρκεια ενός Sprint. Η ομάδα πρώτου ξεκινήσει να εργάζεται, γνωρίζει με απόλυτη βεβαιότητα ότι οι υποχρεώσεις που ανέλαβε για τον τρέχον Sprint δεν πρόκειται να αλλάξουν. Επιπλέον, ο Product Owner καταλήγει στην ιεράρχηση του Product Backlog λαμβάνοντας πάντα υπόψη του το ενδεχόμενο της μελλοντικής αλλαγής των απαιτήσεων των πελατών.

4.4.2 Καθημερινή συνεδρίαση Scrum (Daily Scrum)

Όταν πλέον το Sprint έχει αρχίσει τότε η ομάδα συμμετέχει σε μια ακόμη από τις βασικές πρακτικές του Scrum: **The Daily Scrum Meeting** (καθημερινή συνεδρίαση Scrum). Είναι μια σύντομη συνεδρίαση, η οποία συνήθως διαρκεί 15 περίπου λεπτά, και η οποία πραγματοποιείται, όπως δηλώνει και το όνομά της, καθημερινά όπου παρίσταται ολόκληρη η ομάδα. Είναι μια ευκαιρία για να παρακολουθείται η πρόοδος της ομάδας, ώστε τα μέλη της ομάδας να συγχρονιστούν μεταξύ τους και να γίνουν ευρέως γνωστά τυχόν προβλήματα και δυσκολίες που υπάρχουν. Κατά την συνάθροιση αυτή, ένα προς ένα, κάθε μέλος της ομάδας πρέπει να κοινοποιήσει στα υπόλοιπα μέλη τρία βασικά πράγματα:

1. **Τι καινούριο έχει κάνει στο διάστημα αυτό που έχει μεσολαβήσει από την τελευταία συνεδρίαση.**
2. **Τι εμπόδια και δυσκολίες αντιμετώπισε κατά το διάστημα αυτό.**
3. **Τι σκοπεύει να ολοκληρώσει στο διάστημα μέχρι την επόμενη συνεδρίαση.**

Η συνάντηση πραγματοποιείται την ίδια ώρα και στο ίδιο μέρος κάθε μέρα. Ο σκοπός είναι να βελτιωθεί η επικοινωνία και η λήψη αποφάσεων παράλληλα με την επίλυση προβλημάτων. Μόνο τα μέλη της ομάδας Scrum και ο Scrum Master επιτρέπεται να παρευρίσκονται. Ο Scrum Master οφείλει να διασφαλίσει ότι όλα τα προβλήματα που υπάρχουν έχουν επιλυθεί ομαλά. Η καθημερινή συνάντηση επιθεωρεί την πρόοδο, προκειμένου να επιτευχθούν οι στόχοι του Sprint. Η συνάντηση είναι σημαντική όσον αφορά την αυτό-οργάνωση της ομάδα [Deemer et al., 2010] αφού αν το Daily Scrum Meeting δεν πραγματοποιηθεί με επιτυχία, η ομάδα είναι πιθανό να αγνοεί την τρέχουσα κατάσταση της ανάπτυξης του συστήματος, και επομένως πιθανό να αντιμετωπίσουν μελλοντικές, σημαντικές, δυσκολίες όσο αφορά την επίτευξη των στόχων τους [Schwaber, 2008].

Κατά την διάρκεια της συνάντησης αυτής σημαντική είναι η ανανέωση του Task Board με τα καινούρια στοιχεία που έχουν προκύψει. Με τον τρόπο αυτό ενημερώνεται η τρέχουσα κατάσταση της προόδου και η ομάδα προγραμματίζει τις αμέσως επόμενες ενέργειές της [Kniber, 2007].

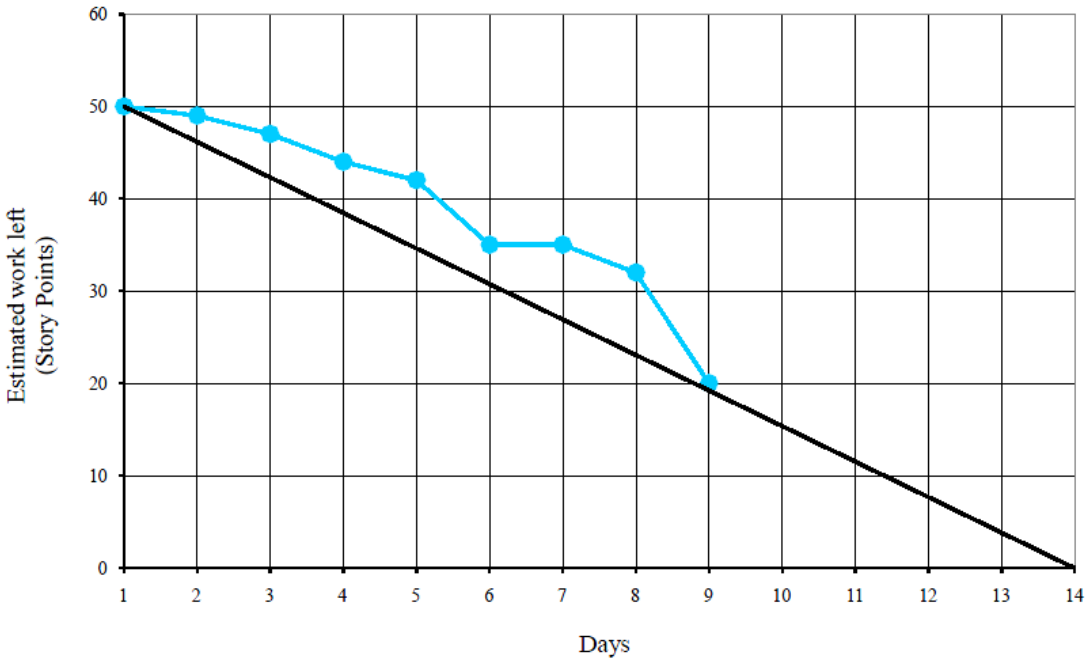
4.4.3 Ενημέρωση των Sprint Backlog & Sprint Burndown Chart

Η ομάδα Scrum αυτό-οργανώνεται και για να μπορεί να το κάνει αυτό θα πρέπει να γνωρίζει την κατάσταση σχετικά με ην πρόοδο της ανάπτυξης του συστήματος. Καθημερινά τα μέλη της ομάδας ενημερώνουν το εκτιμώμενο υπόλοιπο του χρόνου που χρειάζεται για ολοκληρώσουν την εργασία τους. Μετά από αυτή την ενημέρωση, προκύπτει μια συνολική ανανέωση του χρόνου που απομένει ώστε να ολοκληρωθεί ολόκληρο το sprint (Σχήμα 4.6).

				New Estimates of Effort Remaining at end of Day...					
Product Backlog Item	**Sprint Task**	**Volunteer**	**Initial Estimate of Effort**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart	modify database	Sanjay	5	4	3	0	0	0	
	create webpage (UI)	Jing	3	3	3	2	0	0	
	create webpage (Javascript logic)	Tracy & Sam	2	2	2	2	1	0	
	write automated acceptance tests	Sarah	5	5	5	5	5	0	
	update buyer help webpage	Sanjay & Jing	3	3	3	3	3	0	
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5	5	5	5	5	5	
	complete machine order for pRank		3	3	8	8	8	8	
	change DCP and reader to use pRank http API		5	5	5	5	5	5	
	. . .	. . .	. . .						
		Total (person hours)	50	49	48	44	43	34	

Σχήμα 4.6 Ανανέωση της εργασίας που απομένει σε ένα Sprint Backlog [Deemer et al., 2010]

Αποτέλεσμα αυτής της ενημέρωσης είναι ένα γράφημα το οποίο ονομάζεται **Sprint Burndown Chart.** Το γράφημα αυτό αναπαριστά την υπόλοιπη εργασία που απομένει για ένα Sprint Backlog. Ενημερώνεται καθημερινά και παρουσιάζει με ένα απλό τρόπο την πρόοδο κάθε Sprint. Το διάγραμμα αυτό παρουσιάζει, κάθε μέρα, μια νέα εκτίμηση για το πόση δουλειά απομένει μέχρι η ομάδα να ολοκληρώσει όλα τα Tasks σε ένα Sprint. Ονομάζεται «Burndown» αφού αποκαλύπτει τον φόρτο εργασίας που απομένει σαν συνάρτηση του χρόνου σε μια καμπύλη που έχει κλίση προς τα κάτω. Η γραφική αυτή παράσταση ενημερώνεται κατά την διάρκεια του Daily Scrum Meeting [Kniber, 2007]. Στο Σχήμα 4.7 απεικονίζεται ένα παράδειγμα ενός τέτοιου γραφήματος.



Σχήμα 4.7 Παράδειγμα ενός Sprint Burndown Chart [Deemer et al., 2010]

Το διάγραμμα στο Σχήμα 4.7 παρουσιάζει ένα Burndown Chart για ένα Sprint που διαρκεί 14 ημέρες. Στον άξονα X παρουσιάζονται ο υπολειπόμενος χρόνος εργασίας (σε μέρες) και στον άξονα Y παρουσιάζεται το ποσοστό της υπολειπόμενης εργασίας. Κάθε μέρα η τρέχουσα κατάσταση σημειώνεται στο γράφημα με μια τελεία, έπειτα οι τελείες ενώνονται μεταξύ τους με μια γραμμή και με τον τρόπο αυτό προκύπτει μια καμπύλη που κλίνει προς τα κάτω (στο σχήμα παρουσιάζεται με μπλε χρώμα). Επίσης υπάρχει μια γραμμή η οποία ονομάζεται «ιδανική γραμμή» (Idealized Line) και η οποία αναπαριστά την ιδανική πρόοδο που η ομάδα θα έπρεπε να πετύχει (στο σχήμα παρουσιάζεται με μαύρη γραμμή). Βρίσκεται πάντοτε πάνω στο γράφημα ώστε να μπορεί να υπάρξει μια σύγκριση της παρούσας κατάστασης της προόδου με την ιδανική. Η γραμμή αντιπροσωπεύει μια γραμμική πρόοδο από την πρώτη μέρα του Sprint μέχρι και την τελευταία μέρα. Έτσι, η γραμμή ξεκινά από το σημείο όπου καθόλου δουλειά δεν έχει γίνει και τελειώνει στο σημείο στο οποίο δεν έχει απομείνει δουλειά κατά την τελευταία ημέρα.

Το σημαντικό είναι ότι το Burndown Chart υπενθυμίζει στην ομάδα την πρόοδο τους προς τον στόχο, όχι από την άποψη του τι έχουν κάνει μέχρι τώρα αλλά όσο αφορά την δουλειά που απομένει για τον μέλλον και συνεπώς για την επίτευξη του στόχου τους, ο

οποίος είναι η επιτυχής ολοκλήρωση του Sprint. Εάν η καμπύλη δεν κλίνει προς τα κάτω τότε αυτό σημαίνει πως η ομάδα θα πρέπει να ανασυνταχτεί και να εντοπίσει τρόπους ώστε να δουλεύει πιο αποτελεσματικά διατηρώντας παράλληλα ένα σταθερό ρυθμό.

Ενώ το Burndown Chart μπορεί να δημιουργηθεί και να προβληθεί χρησιμοποιώντας ένα ειδικό λογισμικό (π.χ Excel), εντούτοις πολλές ομάδες διαπίστωσαν ότι είναι πιο αποτελεσματικό να χρησιμοποιούν ένα πιο φυσικό τρόπο αναπαράστασης του γραφήματος, όπως πάνω σε ένα πίνακα στον χώρο εργασίας τους. Η λύση αυτή είναι γρήγορη, απλή, και συχνά πιο ορατή από ένα διάγραμμα στον υπολογιστή. Συνήθως το Burndown Chart τοποθετείται κοντά στο Task Board ώστε να είναι προσβάσιμο και ορατό από όλους ανά πάσα στιγμή [Schwaber, 2008].

4.4.4 Αναθεώρηση του Product Backlog

Μια από τις λιγότερο χρησιμοποιούμενες πρακτικές στο Scrum είναι η αναθεώρηση του Product Backlog στο τέλος κάθε Sprint. Αυτό περιλαμβάνει λεπτομερή ανάλυση των απαιτήσεων, διάσπαση των μεγαλύτερων χαρακτηριστικών του συστήματος σε μικρότερα, εκτίμηση των νέων στοιχείων που ενδέχεται να προκύψουν, και την εκ νέου εκτίμησης των υφιστάμενων απαιτήσεων. Η εργασία αυτή γίνεται προς το τέλος κάθε Sprint με την συμμετοχή της ομάδας Scrum και του Product Owner. Για ένα Sprint που διαρκεί 2 βδομάδες, το 5% (περίπου μισή εργάσιμη μέρα) της διάρκειας αυτής αφιερώνεται στην αναθεώρηση του Product Backlog. Η αναθεώρηση αυτή δεν αφορά τα χαρακτηριστικά που επιλέγηκαν για το τρέχον Sprint, αλλά αυτά που πρόκειται να συμπεριληφθούν στα αμέσως επόμενα Sprints. Με τον τρόπο αυτό ο προγραμματισμός ενός Sprint γίνεται σχετικά απλός για τον λόγο ότι ο Product Owner και η ομάδα έχουν στην διάθεση τους ένα σύνολο από καλά ορισμένες και αναλυμένες απαιτήσεις των πελατών.

4.4.5 Ολοκλήρωση ενός Sprint

Όπως είδη αναφέραμε, η χρονική διάρκεια ενός Sprint ποτέ δεν επεκτείνεται και για κανένα λόγο. Ολοκληρώνεται πάντα στο προκαθορισμένο χρονικό όριο ανεξάρτητα

από το αν η ομάδα έχει ολοκληρώσει ή όχι το έργο που δεσμεύτηκε να ολοκληρώσει. Μια ομάδα συνήθως δεν κατορθώνει να ολοκληρώσει όλη την εργασία που της έχει ανατεθεί κατά τα πρώτα Sprints. Με την πάροδο όμως του χρόνου, η ομάδα εντοπίζει τον ρυθμό της και μπορεί πιο εύκολα να προσδιορίσει το φόρτο εργασίας που μπορεί να ολοκληρώσει με επιτυχία κατά την διάρκεια ενός Sprint. Είναι καλύτερο για τις ομάδες να επιλέξουν μια σταθερή διάρκεια που θα έχουν τα Sprints (π.χ. δύο εβδομάδες). Αυτό βοηθάει την ομάδα ώστε να πετύχει έναν σταθερό ρυθμό εργασίας, δηλαδή να ξέρει πόση δουλειά μπορεί να υλοποιήσει μέσα σε αυτό το προκαθορισμένο χρονικό περιθώριο. Ο ρυθμός εργασίας της ομάδας Scrum είναι γνωστός ως «Heartbeat». Έχοντας ως δεδομένο τον παράγοντα αυτό, γίνεται ένας πιο σωστός προγραμματισμός των Sprints.

4.4.5.1 Επανεξέταση του Sprint (Sprint Review)

Όταν ένα Sprint έχει ολοκληρωθεί, ο Product Owner μαζί με την ομάδα Scrum επανεξετάζουν την επανάληψη που μόλις τελείωσε. Η διαδικασία αυτή ονομάζεται **Sprint Review.** Η ομάδα παρουσιάζει τα αποτελέσματα του Sprint και εξετάζεται κατά πόσο οι στόχοι που είχαν τεθεί στο Sprint Planning έχουν ολοκληρωθεί με επιτυχία. Μια βασική αρχή στο Scrum είναι ο έλεγχος και η προσαρμοστικότητα (Inspect and adapt). Αυτό σημαίνει πώς αφού εξεταστεί το αποτέλεσμα που έχει προκύψει τότε εντοπίζονται οι αλλαγές ή προσαρμογές, εφόσον υπάρχουν, που θα πρέπει να γίνουν στο σύστημα κατά τις επόμενες επαναλήψεις. Γίνεται μια λεπτομερής συζήτηση μεταξύ του Product Owner και της ομάδας ώστε να προσδιοριστεί η κατάσταση στην οποία βρίσκεται η όλη ανάπτυξη του συστήματος.

Πρόκειται για μια ανεπίσημη συνεδρίαση της τελευταίας μέρας της επανάληψης. Παρών μπορεί επίσης να είναι ο Scrum Master και επιπλέον οι πελάτες και οποιοσδήποτε άλλος εμπλέκεται στην ανάπτυξη. Γίνεται μια παρουσίαση των αποτελεσμάτων στους χρήστες του προϊόντος. Οι συμμετέχοντες αξιολογούν την ανάπτυξη του έργου και λαμβάνουν αποφάσεις που αφορούν τις επόμενες λειτουργίες που θα πρέπει να υλοποιηθούν. Αυτή η συνεδρίαση είναι σημαντική για την πορεία του έργου καθώς μπορεί να επιφέρει νέα στοιχεία στον κατάλογο του Product Backlog αλλά

και αναθεώρηση υπαρχόντων. Επίσης καθορίζονται οι στόχοι για το αμέσως επόμενο Sprint [Schwaber and Sutherland, 2011].

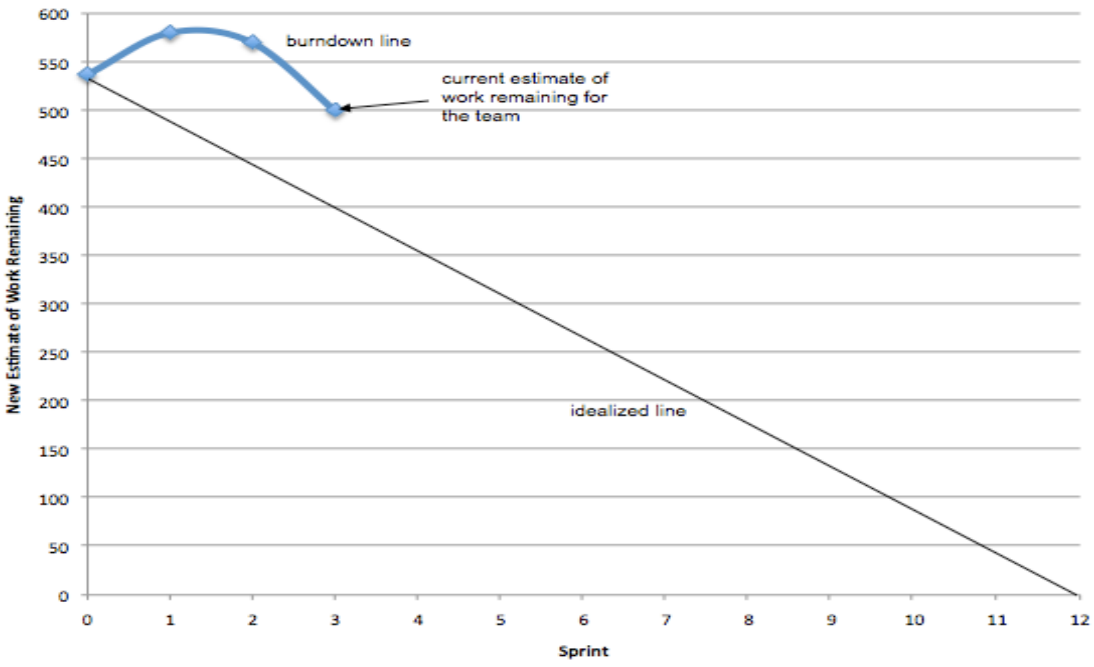
4.4.5.2 Sprint Retrospective

Το **Sprint Retrospective** είναι μια διαδικασία που ακολουθεί της επανεξέτασης του Sprint (Sprint Review). Το Sprint Review αφορά την εξέταση του παραγομένου αποτελέσματος, δηλαδή του προϊόντος. Το Sprint Retrospective αφορά την εξέταση και αναθεώρηση της διαδικασίας που ακολουθήθηκε κατά το τελευταίο Sprint. Αποτελεί τον βασικό μηχανισμό ώστε να εντοπιστούν καθοριστικοί παράγοντες επιτυχίας και πιθανοί τομείς βελτίωσης της διαδικασίας. Κατά το στάδιο αυτό η ομάδα έχει την δυνατότητα να συζητήσει τι δουλεύει και τι όχι και με τον τρόπο αυτό, τα μέλη της ομάδας μπορούν να καταλήξουν σε κάποιες αλλαγές τις οποίες μπορούν να εφαρμόσουν κατά το επόμενο Sprint.

Παρόν στην διαδικασία αυτή είναι ο Scrum Master και η ομάδα Scrum. Η παρουσία του Product Owner δεν είναι υποχρεωτική. Μια επιλογή για την οργάνωση της συνάντησης, είναι να σχηματιστούν δύο στήλες σε ένα πίνακα. Στη μια στήλη θα παρουσιάζονται οι παράγοντες επιτυχίας (What is working well) και στην δεύτερη οι πιθανοί τομείς βελτίωσης (What could work better). Τα μέλη της ομάδας, ένα προς ένα, προσθέτουν στον πίνακα ένα ή περισσότερα στοιχεία σε οποιαδήποτε από τις στήλες. Στο τέλος, γίνεται μια καταμέτρηση και προκύπτουν τα δημοφιλέστερα προβλήματα ή παράγοντες επιτυχίας αντίστοιχα. Τα ζητήματα, που αφορούν βελτιώσεις στην διαδικασία, τα οποία προκύπτουν μέσα από αυτό το στάδιο, κατατάσσονται σε μια σειρά προτεραιότητας και εφαρμόζονται στα επόμενα Sprints. Ο [Kniber, 2007] προτείνει η ομάδα να επικεντρώνεται μόνο σε μια περιοχή βελτίωσης ανά Sprint. Είναι σημαντικό να διαχωρίζονται τα στοιχεία που αφορούν την διαδικασία του Scrum από τους παράγοντες που δεν επηρεάζονται από τις πρακτικές και τεχνικές που ορίζει το Scrum.

4.4.5.3 Ενημέρωση των Release Backlog & Burndown Chart

Στο σημείο αυτό ορισμένες εργασίες έχουν ολοκληρωθεί, κάποιες άλλες έχουν προστεθεί στο Product Backlog ενώ μερικές έχουν αναθεωρηθεί. Ο Product Owner είναι υπεύθυνος για την ενημέρωση του Release Backlog (κατάλογος των ανεκτέλεστων εργασιών του προϊόντος για ένα Sprint) και κατ' επέκταση του Product Backlog. Επιπλέον μερικές ομάδες ενδέχεται να ενημερώνουν το **Release Burndown Chart**, το οποίο αποτελεί μια προέκταση του Sprint Burndown Chart με την διαφορά ότι επικεντρώνεται στις απατήσεις και όχι στα Tasks. Είναι μια γραφική παράσταση στην οποία αναπαριστάται η συνολική πρόοδος της ομάδας. Παρουσιάζει μια συνάρτηση της αναθεωρημένης υπολειπόμενης συνολικής ανεκτέλεστης εργασίας σε σχέση με τον αριθμός των Sprints που απομένουν (Σχήμα 4.8).



Σχήμα 4.8 Release Burndown Chart [Deemer et al., 2010]

Το γράφημα αυτό είναι γνωστό και σαν Product Burndown Chart [Hundermark, 2009] και απεικονίζει τον ρυθμό παράδοσης τμημάτων του συστήματος. Χρησιμοποιώντας αυτό το διάγραμμα ο Product Owner είναι σε θέση να προσδιορίσει την πρόοδο και να καθορίσει τις επόμενες ημερομηνίες παράδοσης.

4.4.6 Ολοκλήρωση της διαδικασίας

Ο απώτερος σκοπός του Scrum, είναι να παραδίδει λειτουργικά τμήματα του τελικού συστήματος στο τέλος κάθε Sprint. Αυτό σημαίνει πως το αποτέλεσμα κάθε επανάληψης, είναι μια επαύξηση στο τελικό προϊόν. Στο τέλος του Sprint, η νέα επαύξηση (άθροισμα όλων των εργασιών που περιλαμβάνονται στο Sprint Backlog) θα πρέπει να είναι σε κατάσταση που να μπορεί να δοθεί στον πελάτη **(Shippable Product Increment)** και να ανταποκρίνεται στον ορισμό του «ολοκληρωμένου» (Definition of Done).

Αφού, λοιπόν, ολοκληρωθεί ένα Sprint, το επόμενο βήμα είναι να ξεκινήσει ξανά η διαδικασία από την αρχή. Οι επαναλήψεις συνεχίζονται μέχρι οι απαιτήσεις των πελατών να ικανοποιηθούν πλήρως και να προκύψει ένα ολοκληρωμένο προϊόν.

Κεφάλαιο 5

Έρευνα Εφαρμογής των Ευέλικτων Μεθόδων με Έμφαση στο Scrum

5.1 Εισαγωγή

Οι ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού έχουν τύχει ιδιαίτερης προσοχής τα τελευταία δέκα χρόνια. Το Scrum και ο Ακραίος Προγραμματισμός (XP) είναι οι πλέον δημοφιλείς μέθοδοι μεταξύ αυτών. Αν και οι ερευνητές έχουν δήξει ενδιαφέρον προς αυτό τον «νέο» τρόπο ανάπτυξης, πολλοί έχουν επικρίνει τις ευέλικτες μεθοδολογίες και αμφισβητούν την καταλληλότητα τους. Στη διάρκεια των τελευταίων χρόνων, ανεπίσημα αλλά και επίσημα στοιχεία, καταγράφουν, ως επί το πλείστον, θετικές εμπειρίες από την εφαρμογή της ευέλικτης ανάπτυξης λογισμικού. Ωστόσο, θα μπορούσαμε να πούμε ότι λίγα είναι τα ευρήματα της αποτελεσματικότητας των ευέλικτων πρακτικών στο παγκόσμιο εξελισσόμενο και πολύπλοκο περιβάλλον ανάπτυξης λογισμικού. Για τον λόγο αυτό έχει οργανωθεί και πραγματοποιηθεί, στα

πλαίσια της παρούσας διπλωματική εργασίας, μια μελέτη πεδίου με σκοπό τη διερεύνηση της έκτασης και της χρήσης των ευέλικτων μεθόδων και ιδιαίτερα της μεθοδολογίας Scrum στην παγκόσμια βιομηχανία λογισμικού.

Όπως παρουσιάστηκε στο κεφάλαιο 3, το μανιφέστο των ευέλικτων πρακτικών (Agile Manifesto) τονίζει την σημασία των εξής στοιχείων: (α) άτομα και αλληλεπιδράσεις αντί διαδικασίες και εργαλεία, (β) καθαρός κώδικας αντί γραπτής τεκμηρίωσης, (γ) συνεργασία με τον πελάτη αντί αυστηρών συμβολαίων και (δ) ανταπόκριση σε αλλαγές αντί ακολουθούμενου σχεδίου. Οι ευέλικτες μεθοδολογίες ανάπτυξης, συμπεριλαμβανομένου και του Scrum, βασίζονται στις αρχές του Agile Manifesto και προσανατολίζονται προς την επίτευξη των στόχων του.

Σε γενικές γραμμές, η ανταπόκριση από τις οργανώσεις που έχουν εφαρμόσει ευέλικτη ανάπτυξη είναι θετική. Μερικά από τα πλεονεκτήματα που αποδίδονται στην ευέλικτη ανάπτυξη, είναι η αύξηση της παραγωγικότητας, η βελτίωση της ποιότητας, λιγότερα ελαττώματα, μείωση χρόνου και κόστους, διατηρήσιμος και επεκτάσιμος κώδικας, καλύτερη συνεργασία καθώς και μεγαλύτερη ικανοποίηση του πελάτη. Η υιοθέτηση της ευέλικτης ανάπτυξης αποκάλυψε επίσης κάποιες προκλήσεις, όπως η έλλειψη λεπτομερούς αξιολόγησης του κόστους, η ανάγκη για τακτική συμμετοχή του πελάτη, η υποστήριξη της διαχείρισης και η ανάγκη για εκτεταμένη εκπαίδευση και κατάρτιση για χρήση των ευέλικτων μεθόδων.

Τα ευρήματα αυτά, τα οποία συγκεντρώθηκαν από μελέτες περιπτώσεων (Case Studies) και έρευνες όπως αυτές των [Drobka et al., 2004] (η εμπειρία της Motorola με το XP), [Schatz and Abdelshafi, 2005] (Primavera Systems' adoption of Scrum), παρέχουν λεπτομερείς γνώσεις σχετικά με την εφαρμογή της ευέλικτης ανάπτυξη λογισμικού για συγκεκριμένα έργα.

Στόχος της μελέτης αυτής, είναι να καταγραφεί μια γενικότερη εικόνα όσο αφορά την εφαρμογή των ευέλικτων μεθόδων και ειδικότερα της μεθοδολογίας Scrum. Η έρευνα αυτή επιτρέπει τη συγκέντρωση στοιχείων, από ένα σχετικά μεγάλο δείγμα οργανισμών, που αφορούν το ποιές ευέλικτες μέθοδοι χρησιμοποιούνται συνήθως και την εξαγωγή συμπερασμάτων που αφορούν την αποτελεσματικότητα από την

εφαρμογή αυτών των μεθόδων, συμπεριλαμβανομένων των πλεονεκτημάτων που έχουν επιτευχθεί και των προκλήσεων που αντιμετωπίζουν. Επιπλέον καταγράφονται απόψεις σχετικά με την ευέλικτη έναντι παραδοσιακής ανάπτυξης λογισμικού. Η ικανότητα να ανταποκρίνονται στις συνεχώς μεταβαλλόμενες ανάγκες των πελατών και η παράδοση ποιοτικών προϊόντων λογισμικού στην ώρα τους είναι μερικά από τα σημαντικά οφέλη της ευέλικτης ανάπτυξης. Παράλληλα αποδεικνύεται η ακαταλληλότητά της χρήση τέτοιων πρακτικών για έργα που χαρακτηρίζονται από κατανεμημένα περιβάλλοντα. Επίσης, οι μεγάλες ομάδες ανάπτυξης αναγνωρίζονται ως ένα ουσιαστικής σημασίας πρόβλημα. Τα αποτελέσματα δείχνουν την τρέχουσα τάση της βιομηχανίας λογισμικού που κινείται προς την υιοθέτηση ευέλικτων μεθόδων στη διαδικασία ανάπτυξής λογισμικού. Στο κεφάλαιο αυτό συγκεντρώνονται αυτά αλλά και περαιτέρω ευρήματα τα οποία συνοδεύονται από σχετική ερμηνευτική ανάλυση.

5.2 Προηγούμενες έρευνες

Η ανάπτυξη λογισμικού ήταν πάντα μια κρίσιμη διαδικασία, και η ιστορία της βιομηχανίας ανάπτυξης λογισμικού είναι γεμάτη λάθη, σφάλματα, μη έγκαιρη παράδοση των προϊόντων κτλ. (Βλέπε Κρίση Λογισμικού, Κεφάλαιο 1). Μια έρευνα από το Conference Board [Cooke et al., 2001] το 2001 αποκάλυψε ότι μόνο το 34% των πελατών ήταν ικανοποιημένοι, το 58% ήταν κάπως ικανοποιημένοι, το 8% ήταν δυσαρεστημένοι και 40% των έργων λογισμικού απέτυχαν να ολοκληρωθούν. Το Agile Alliance είχε τότε σχηματιστεί κατά τη διάρκεια αυτής της περιόδου, και οι ευέλικτες μεθοδολογίες γίνονταν όλο και πιο δημοφιλείς. Οι ευέλικτες μέθοδοι, όπως το Scrum και ο Ακραίος Προγραμματισμός, μπορεί να εκληφθούν ως μια αντίδραση στα εκτενής διαρκείας πλάνα που χαρακτηρίζουν τα παραδοσιακά μοντέλα και ως μια απάντηση στον άκαμπτο παραδοσιακό τρόπο ανάπτυξης λογισμικού. Η επίδραση των ευέλικτων μεθόδων φαίνεται από τις πιο κάτω έρευνες οι οποίες παρατίθενται με χρονολογική σειρά ώστε να σκιαγραφηθεί η τάση και οι προτιμήσεις προς τις ευέλικτες μεθοδολογίες.

Το 2003, οι Shine Technologies [Shine Technologies, 2003], μια Αυστραλιανή εταιρία Πληροφοριακών Συστημάτων (Informational Technology, IT), διεξήγαγε μια διαδικτυακή (web-based) έρευνα για να διαπιστωθεί το ενδιαφέρον των οργανισμών

προς τις ευέλικτες μεθόδους. Έλαβαν 131 απαντήσεις από όλο τον κόσμο, η πλειοψηφία των οποίων (84,7%) ανέφεραν ότι ήταν γνώστες των ευέλικτων μεθόδων. Τα πορίσματα της έρευνας δείχνουν ότι το XP ήταν η πιο δημοφιλής ευέλικτη μέθοδος με το 59% των ερωτηθέντων να την χρησιμοποιούν. Η συντριπτική πλειοψηφία των ερωτηθέντων (80% περίπου) ανέφεραν ότι οι ευέλικτες διαδικασίες είχαν βελτιώσει την παραγωγικότητα της ομάδας, την ποιότητα των εφαρμογών και την ικανοποίηση των επιχειρήσεων (πελατών). Περαιτέρω, περίπου το 50% των ερωτηθέντων πίστευαν ότι το κόστος παραγωγής είχε μειωθεί με την εισαγωγή ευέλικτων μεθόδων, ενώ η ανταπόκριση στις αλλαγές και η έμφαση στους ανθρώπους έναντι των διαδικασιών προέκυψαν ως θετικά στοιχεία της ευέλικτης ανάπτυξης. Τέλος, δεν υπήρχε ομόφωνη πρόθεση μεταξύ των ερωτηθέντων για το εάν θα συνεχίσουν να χρησιμοποιούν ευέλικτη ανάπτυξη.

Η Digital Focus [Digital Focus, 2006], μια άλλη μεγάλη εταιρεία Πληροφορικής, κατασκεύασε μια on-line έρευνα το 2006 αποσπώντας απαντήσεις από 136 άτομα που αντιπροσώπευαν 128 οργανισμούς από 17 διαφορετικές χώρες. Περίπου το 90% των ερωτηθέντων της παρούσας έρευνας είχαν γνώριζαν την ευέλικτη ανάπτυξη λογισμικού και το 81% αυτών είτε χρησιμοποιούσαν είτε επρόκειτο να χρησιμοποιήσουν ευέλικτες μεθόδους στις εταιρίες τους. Δημοφιλή κίνητρα για την υιοθέτηση των ευέλικτων πρακτικών, αποδείχτηκαν η ανάγκη για την αντιμετώπιση έργων με αμφιλεγόμενες ή/και εξελισσόμενες απαιτήσεις, εμπνέουν σταθερότητα στην αναπτυξιακή διαδικασία και επιταχύνεται η παράδοση του λογισμικού. Η πλειοψηφία των συμμετεχόντων ξεχώρισε την ικανότητα των μεθόδων αυτών να ανταποκρίνονται στις αλλαγές ως βασική αξία που προσφέρει η ευέλικτη ανάπτυξη, και την έλλειψη οργανωτικών γνώσεων και δεξιοτήτων ως τα πιο σημαντικά μειονεκτήματα της εφαρμογής των μεθόδων αυτών.

Μια, σχετικά, πρόσφατη έρευνα από την Forrester που διεξήχθη τον Μάιο του 2009, δείχνει ότι το 35% των οργανισμών που έχουν ήδη εφαρμόσει ευέλικτη ανάπτυξη, το 33% ήταν τις εφάρμοζαν κατά την περίοδο της έρευνας, το 17% μόλις άρχισε να τις εφαρμόζει, ένα άλλο 17% εξακολουθούσε να αξιολογεί τις ευέλικτες μεθόδους και δεν τις είχαν υιοθετήσει και μόνο το 4% δεν κατάφεραν να τις εφαρμόσουν. Στην έρευνα απάντησαν 229 επαγγελματίες του κλάδου ανάπτυξης λογισμικού από εταιρείες όπως η

IBM, ACS, Protegra κτλ. Τα αποτελέσματα αναφέρθηκαν, επίσης, στο ότι οι ευέλικτες μέθοδοι επιδέχονται αλλαγές ακόμη και σε προχωρημένα στάδια του κύκλου ζωής του λογισμικού.

Σήμερα, πολλοί από τους γίγαντες της βιομηχανίας λογισμικού χρησιμοποιούν ευέλικτες μεθόδους ανάπτυξης λογισμικού, συμπεριλαμβανομένων των Yahoo, Microsoft, Oracle, Sun Microsystems, HP, APL, IBM, Motorola, Xerox, Federal Reserve Bank και Capital One [Bhardwaj, 2011].

5.3 Μεθοδολογία έρευνας

Τα δεδομένα για την παρούσα έρευνα συλλέχθηκαν μέσω ενός ανώνυμου online ερωτηματολογίου βασιζόμενο στο SurveyMonkey (πάροχος web-based λύσεων έρευνας) και το οποίο προσφέρθηκε για μια περίοδο δύο μηνών από τον Μάρτιο μέχρι και τον Απρίλη του 2012. Το ερωτηματολόγιο αποστάλθηκε σε επαγγελματίες ανάπτυξης λογισμικού. Εντοπίστηκαν αρκετές εταιρίες που παρέχουν υπηρεσίες Πληροφορικής και πιο συγκεκριμένα που ειδικεύονται στην ανάπτυξη λογισμικού. Επιπλέον, μέσω διάφορων κοινωνικών δικτύων (π.χ LinkedIn, Facebook) εντοπίστηκαν ομάδες, που επικεντρώνονται στην ευέλικτη ανάπτυξη λογισμικού και ειδικότερα στο Scrum, όπου δημοσιεύτηκε ένα μήνυμα πρόσκλησης καλώντας τα μέλη που είχαν εμπειρία στη χρήση ευέλικτων προσεγγίσεων να συμπληρώσουν το ερωτηματολόγιο. Το ερωτηματολόγιο καθώς και το μήνυμα πρόσκλησης που στάλθηκε προς τις εταιρίες και τους επαγγελματίες ανάπτυξης λογισμικού παρουσιάζονται στο Παράρτημα Α.

5.4 Μορφή ερωτηματολογίου

Το ερωτηματολόγιο δημιουργήθηκε με συγκεκριμένη κατάλληλη δομή για να μπορεί να απαντηθεί εύκολα και γρήγορα από τους ερωτώμενους. Οι ερωτήσεις ήταν στην πλειοψηφία τους κλειστού τύπου. Αυτό σημαίνει πώς το ερωτηματολόγιο είχε προκαθορισμένη μορφή και περιελάμβανε ερωτήσεις για τις οποίες ο ανταποκρινόμενος είχε να επιλέξει μεταξύ συγκεκριμένων απαντήσεων που του παρουσιάζονταν. Ωστόσο σε μεμονωμένες ορισμένες περιπτώσεις υπήρχαν ερωτήσεις και επιλογές ανοικτού τύπου δίνοντας έτσι στους ερωτώμενους την ευχέρεια απάντησης

μέσα από ένα χώρο που υπήρχε στο ερωτηματολόγιο. Οι ερωτήσεις ανήκαν σε μία από τις εξής 3 κατηγορίες-ενότητες:

1. **Χαρακτηριστικά οργανισμού:** Σε αυτήν την ενότητα περιέχονται ερωτήσεις σχετικά με τον τύπο, την τοποθεσία και το μέγεθος του οργανισμού που εργάζονται οι ερωτώμενοι. Επίσης, διερευνήθηκε η αποφασιστικότητα και θέληση των οργανισμών να υιοθετήσουν νέες τεχνολογίες και μεθόδους.
2. **Ερωτήσεις σχετικά με τις ευέλικτες και παραδοσιακές μεθοδολογίες:** Σε αυτήν την ενότητα υπάρχουν ερωτήσεις για τις μεθοδολογίες, τόσο ευέλικτες όσο και παραδοσιακές, που χρησιμοποιούν οι ερωτώμενοι. Επίσης, υπάρχουν ερωτήσεις που διερευνούν τα δυνατά και αδύναμα σημεία των ευέλικτων μεθοδολογιών, το πλήθος των έργων που υλοποιήθηκαν με ευέλικτες μεθοδολογίες και τις επιδράσεις σε διάφορους τομείς που είχε η εφαρμογή των ευέλικτων μεθοδολογιών. Τέλος, οι συμμετέχοντες ρωτήθηκαν αν είναι ευχαριστημένοι από τις ευέλικτες μεθοδολογίες και διερευνήθηκε το ενδεχόμενο επαναχρησιμοποίησης τους στο μέλλον.
3. **Δημογραφικές ερωτήσεις:** Αυτές οι ερωτήσεις σκοπεύουν στο να σκιαγραφήσουν το προφίλ των ερωτώμενων, δηλαδή περιέχονται ερωτήσεις σχετικά με την ηλικία, το φύλο, το ακαδημαϊκό επίπεδο, τη θέση, την εργασιακή εμπειρία γενικά αλλά και ειδικά την εμπειρία με ευέλικτες μεθοδολογίες.

Η μεταφορά από τη μια ενότητα στην άλλη ήταν αισθητή, συνοδευόταν από επικεφαλίδες που διευκόλυναν την μετάβαση και διατηρούσαν την ομαλή ροή των ερωτήσεων. Το λεξιλόγιο που χρησιμοποιήθηκε στην διατύπωση των ερωτήσεων ήταν απλό, συγκεκριμένο και κατανοητό. Στην $2^{η}$ ενότητα προηγήθηκε μια σύντομη εισαγωγική περιγραφή στην οποία δίνονταν διευκρινήσεις για κάποιους ορισμούς που θα ακολουθούσαν. Η κάθε ερώτηση δεν αναφερόταν σε περισσότερα του ενός θέματα, ήταν σύντομη, η διατύπωσή της δεν προέτρεπε προς μία συγκεκριμένη απάντηση και δεν προϋπόθετε γνώσεις πέρα του αντικειμένου.

Επίσης, στο τέλος του ερωτηματολογίου υπήρχε μια ερώτηση ανοικτού τύπου στην οποία δινόταν η δυνατότητα στους ερωτώμενους, να εκφράσουν οποιαδήποτε σχόλια

και απόψεις ήθελαν. Οι πληροφορίες αυτές συλλέχτηκαν και συγκεντρώνονται στο Παράρτημα Γ.

5.5 Παρουσίαση/Ανάλυση αποτελεσμάτων

Στο υποκεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα που προέκυψαν από την έρευνα που διεξήχθη με το ερωτηματολόγιο. Στις περισσότερες ερωτήσεις του ερωτηματολογίου, οι ερωτήσεις αναφέρονται σε Agile μεθοδολογίες και όχι απευθείας στη μέθοδο Scrum, για να μπορέσουν να απαντήσουν όσο το δυνατόν περισσότεροι ερωτώμενοι. Στο μήνυμα συμμετοχής (Παράρτημα Α) όμως που στάλθηκε προς στις εταιρίες και τους μηχανικούς λογισμικού, υπήρχε η υποσημείωση ότι αν έχουν συμμετέχει σε διάφορες ομάδες που χρησιμοποίησαν Agile μεθοδολογίες, τότε να απαντήσουν στις ερωτήσεις του ερωτηματολογίου έχοντας ως βάση τη μέθοδο Scrum. Για να διαπιστωθεί πόσοι είναι οι ερωτώμενοι που χρησιμοποιούν τη μέθοδο Scrum υπάρχει μία ερώτηση στο ερωτηματολόγιο (Ερώτηση 7), που όπως θα αναλυθεί σε επόμενη ενότητα προκύπτει ότι η συντριπτική πλειοψηφία των ερωτώμενων (76,9%) χρησιμοποιούν κύρια τη μέθοδο Scrum από τις Agile μεθοδολογίες.

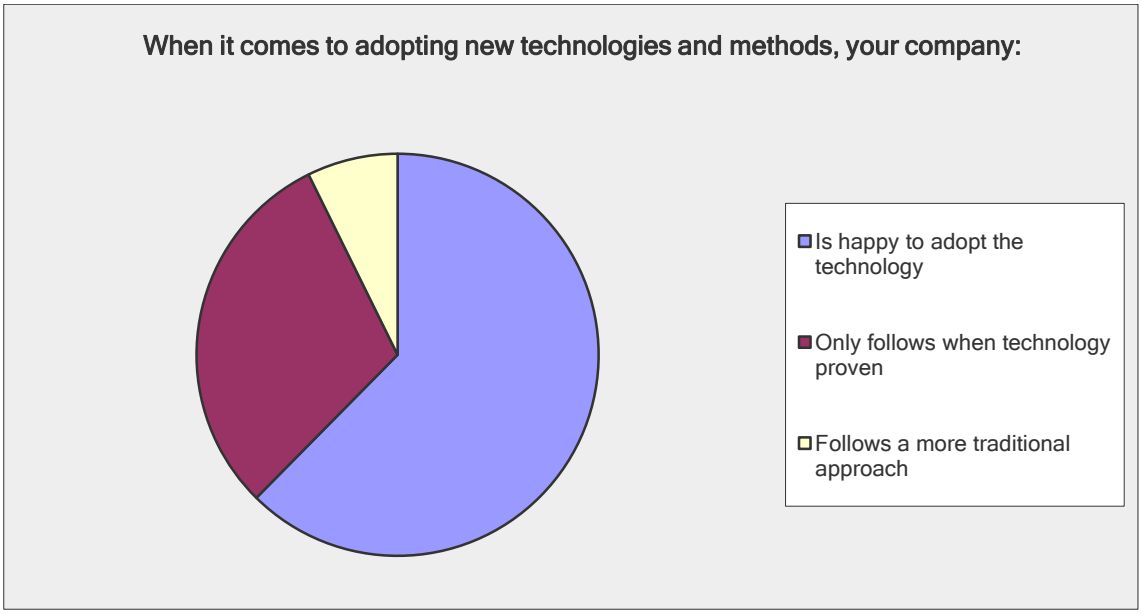
Η ανάλυση και τα αποτελέσματα του ερωτηματολογίου παρουσιάζονται παρακάτω και όλα τα ευρήματα συνοψίζονται και παρουσιάζονται στο Παράρτημα Β.

5.5.1 Προφίλ οργανισμών

Σε αυτήν την ενότητα θα αναλύσουμε το προφίλ των επιχειρήσεων/οργανισμών από τις οποίες προέρχονται οι συμμετέχοντες. Το προφίλ των ερωτώμενων που έλαβαν μέρος στην έρευνα παρουσιάζεται στην επόμενη ενότητα.

Συλλέχτηκαν συνολικά 335 απαντήσεις από επαγγελματίες λογισμικού, από τις οποίες οι 233 είναι ολοκληρωμένες, δηλαδή το 70% περίπου των συμμετεχόντων έδωσαν απάντηση σε όλες τις ερωτήσεις του ερωτηματολογίου. Οι συμμετέχοντες στην έρευνα προέρχονται από περισσότερες από 126 εταιρίες Πληροφορικής η βάση των οποίων βρίσκεται σε 44 διαφορετικές χώρες ανά τον κόσμο.

Το 71,7% των επιχειρήσεων έχει ως κύρια απασχόληση την ανάπτυξη λογισμικού, ενώ το 28,3% των επιχειρήσεων ασχολείται δευτερευόντως με την ανάπτυξη λογισμικού. Το 30,6% των επιχειρήσεων έχει πάνω από 1000 εργαζομένους, το 25,5% έχει 101 – 1000 υπαλλήλους, το 24,2% έχει 11 – 100 εργαζομένους και το 19,7% έχει 1 – 10 υπαλλήλους. Όταν πρόκειται να υιοθετήσουν νέες τεχνολογίες και μεθόδους (Σχήμα 5.1), το 62,4% των ερωτηθέντων απάντησαν ότι οι εταιρίες τους δεν διστάζουν να ενσωματώσουν μια νέα τεχνολογία αμέσως, το 30,3% περιγράφονται ως συντηρητικοί εφόσον δήλωσαν ότι περιμένουν να δοκιμαστεί πρώτα μια τεχνολογία για να την χρησιμοποιήσουν, ενώ το 7,3% προτιμά πάντα να ακολουθεί μια δοκιμασμένη παραδοσιακή τεχνολογία. Έχοντας ως βάση το ποσοστό των συμμετεχόντων που δηλώνουν ότι εύκολα ακολουθούν μια νέα μεθοδολογία (ποσοστό 62.4%), παρατηρήθηκε πώς η πλειοψηφία αυτών δεν χρησιμοποιεί παραδοσιακές μεθοδολογίες παρά μόνο ευέλικτες πρακτικές. Αντίστοιχα η συντριπτική πλειοψηφία (ποσοστό 91,7%) μεταξύ αυτών που ακολουθούν μια δοκιμασμένη πρακτική, παρατηρήθηκε πώς χρησιμοποιούν μόνο παραδοσιακές τεχνολογίες με προτιμότερη αυτή του καταρράκτη. Αυτή η παρατήρηση υποστηρίζει το συμπέρασμα ότι η πλειοψηφία των ευέλικτων μεθοδολογιών γίνονται αποδεκτές μόνο από καινοτόμους ανθρώπους και όσους τις υιοθέτησαν νωρίς, σύμφωνα με το νόμο του Moore Geoffrey [Moore, 2002].



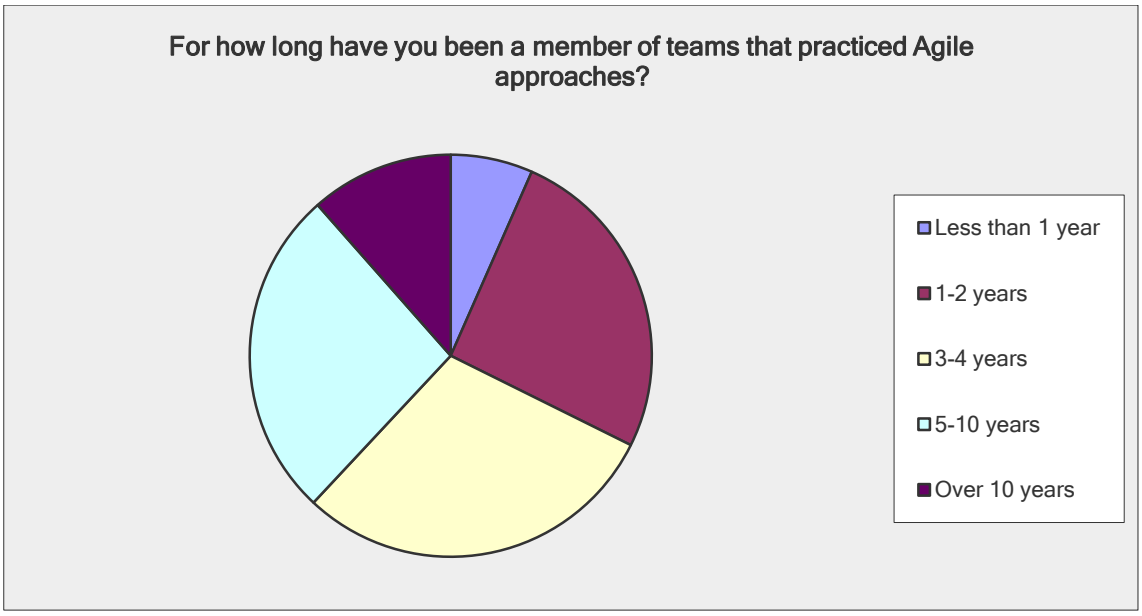
Σχήμα 5.1 Τάση υιοθέτησης νέων τεχνολογιών

Συμπερασματικά, το προφίλ των επιχειρήσεων που έλαβαν μέρος στην έρευνα ήταν μεσαίου και μεγάλου μεγέθους επιχειρήσεις, που έχουν ως κύρια απασχόληση την ανάπτυξη λογισμικού, και επιθυμούν πάντα την ενσωμάτωση μιας νέας τεχνολογίας.

5.5.2 Δημογραφικά στοιχεία ερωτώμενων

Σχετικά με την ηλικία των συμμετεχόντων, το 41,1% των ερωτώμενων είναι μεταξύ 30 – 40 ετών, το 28,1% μεταξύ 40 – 50 ετών, το 14,3% μεταξύ 50 – 60 ετών, το 12,1% μεταξύ 18 – 30 ετών και το 4,5% πάνω από 60 ετών. Επίσης, το μεγαλύτερο ποσοστό των ερωτώμενων (90,5%) είναι άνδρες και το 9,5% είναι γυναίκες. Αναφορικά με το ακαδημαϊκό επίπεδο των ερωτώμενων, το 42,5% αυτών είναι κάτοχοι μεταπτυχιακού τίτλου σπουδών, το 38,9% έχουν πτυχίο πανεπιστημίου, το 8% έχουν δίπλωμα τεχνικής εκπαίδευσης, το 4,4% είναι κάτοχοι διδακτορικού, το 4% έχουν απολυτήριο λυκείου και μόλις το 2,2% έχουν άλλους τίτλους σπουδών (με τη συντριπτική πλειοψηφία αυτών να είναι κάτοχοι πτυχίου από κάποιο κολλέγιο). Οι συμμετέχοντες κατέχουν διάφορες θέσεις στον οργανισμό όπου εργάζονται. Το 27% είναι μηχανικοί λογισμικού, το 25,2% υπεύθυνοι του τμήματος πληροφορικής, το 23% υπεύθυνοι διαχείρισης έργου (Project Managers), ενώ το υπόλοιπο 23,8% δουλεύουν σε άλλες διάφορες ειδικότητες όπως υπεύθυνοι ποιότητας λογισμικού (Quality Assurance/Tester), επαγγελματίες ανάλυσης δεδομένων, υπεύθυνοι μοντελοποίησης κτλ.

Όσο αφορά την επαγγελματική εμπειρία που έχουν οι συμμετέχοντες σχετικά με την ανάπτυξη λογισμικού, το 44,1% των ερωτηθέντων έχουν εμπειρία πάνω από 15 έτη, το 31,7% μεταξύ 11 – 15 ετών, το 18,1% μεταξύ 5 – 10 ετών, ενώ μόλις το 6,2% μεταξύ 1 – 4 ετών. Πιο συγκεκριμένα, όσο αφορά την εμπειρία των ερωτώμενων σχετικά με την συμμετοχή του σε ομάδες που εφαρμόζουν Agile μεθοδολογίες (Σχήμα 5.2), το 29,6% των ερωτώμενων έχουν εμπειρία μεταξύ 3 – 4 ετών, το 26,5% μεταξύ 5 – 10 ετών, το 25,7% μεταξύ 1 – 2 ετών, το 11,5% πάνω από 10 έτη, ενώ μόλις το 6,6% έχει εμπειρία λιγότερη από 1 χρόνο.

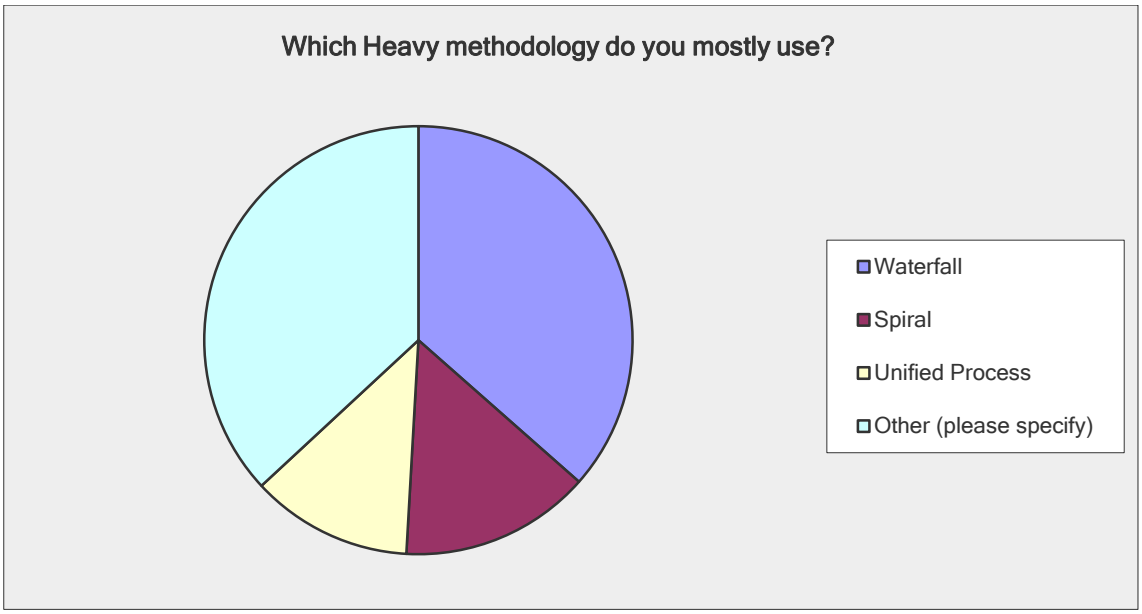


Σχήμα 5.2 Εμπειρία δείγματος σχετικά με τις Agile μεθοδολογίες

Συμπερασματικά, το προφίλ των ερωτώμενων που έλαβαν μέρος στην έρευνα ήταν άνδρες μεταξύ 30 και 60 ετών, που έχουν ανώτερο ή ανώτατο τίτλο σπουδών, εργάζονται είτε ως υπεύθυνοι διαχείρισης έργου ή ως μηχανικοί λογισμικού είτε γενικά στο τμήμα πληροφορικής, έχουν επαγγελματική εμπειρία άνω το 10 ετών και προηγούμενη εμπειρία με τις Agile μεθοδολογίες πάνω από 1 χρόνο.

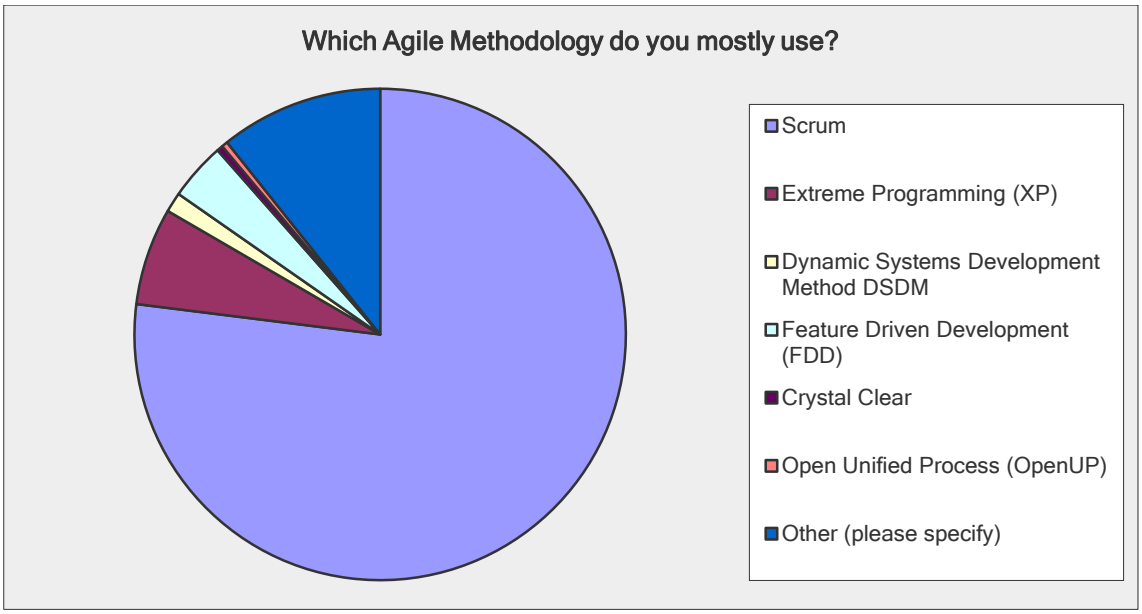
5.5.3 Χρησιμοποιούμενες μεθοδολογίες

Στην ερώτηση για το ποιά παραδοσιακή μέθοδο χρησιμοποιούν περισσότερο οι οργανισμοί στους οποίους ανήκουν οι ερωτώμενοι (Σχήμα 5.3), το 36,5% αυτών χρησιμοποιούν το μοντέλο καταρράκτη, το 14,4% το σπειροειδές μοντέλο, το 12,2% την ενοποιημένη μέθοδο, ενώ τέλος το 36,9% χρησιμοποιεί άλλες μεθόδους, με κυριότερες απαντήσεις είτε κάποια υβριδική μέθοδο μεταξύ των παραπάνω μεθοδολογιών είτε δεν χρησιμοποιούν καμία παραδοσιακή μεθοδολογία, αλλά χρησιμοποιούν μόνο Agile μεθοδολογίες. Τα ευρήματα της έρευνας βρίσκονται σε συμφωνία με την έρευνα [Cocco et al., 2011], όπου επίσης βρέθηκε ότι το 55% των ερωτώμενων προτιμούν το μοντέλο του καταρράκτη από τις παραδοσιακές μεθοδολογίες.



Σχήμα 5.3 Εφαρμογή παραδοσιακών μεθοδολογιών

Η ίδια έρευνα έδειξε πώς η πιο δημοφιλής χρησιμοποιούμενη ευέλικτη μεθοδολογία ήταν ο Ακραίος Προγραμματισμός (Extreme Programming, XP). Τα ευρήματα της έρευνας αυτής βρίσκονται σε διαφωνία με την παρούσα έρευνα, αφού η προτιμότερη Agile μέθοδος φαίνεται πως είναι η μέθοδος Scrum (Σχήμα 5.4). Πιο συγκεκριμένα, το 76,9% των ερωτώμενων απάντησαν ότι χρησιμοποιούν περισσότερο τη μέθοδο Scrum, το 6,4% τη μέθοδο Extreme Programming, το 3,8% τη μέθοδο Feature Driven Development, το 1,3% τη μέθοδο Dynamic Systems Development Method, το 0,4% τη μέθοδος Crystal Clear, άλλο ένα 0,45% τη μέθοδο Open Unified Process, ενώ το 10,7% απάντησαν κάποια άλλη μέθοδο, με κυριότερη απάντηση τη χρησιμοποίηση συνδυασμού ή την παραλλαγή κάποιων μεθόδων από τις παραπάνω.



Σχήμα 5.4 Εφαρμογή ευέλικτων μεθοδολογιών

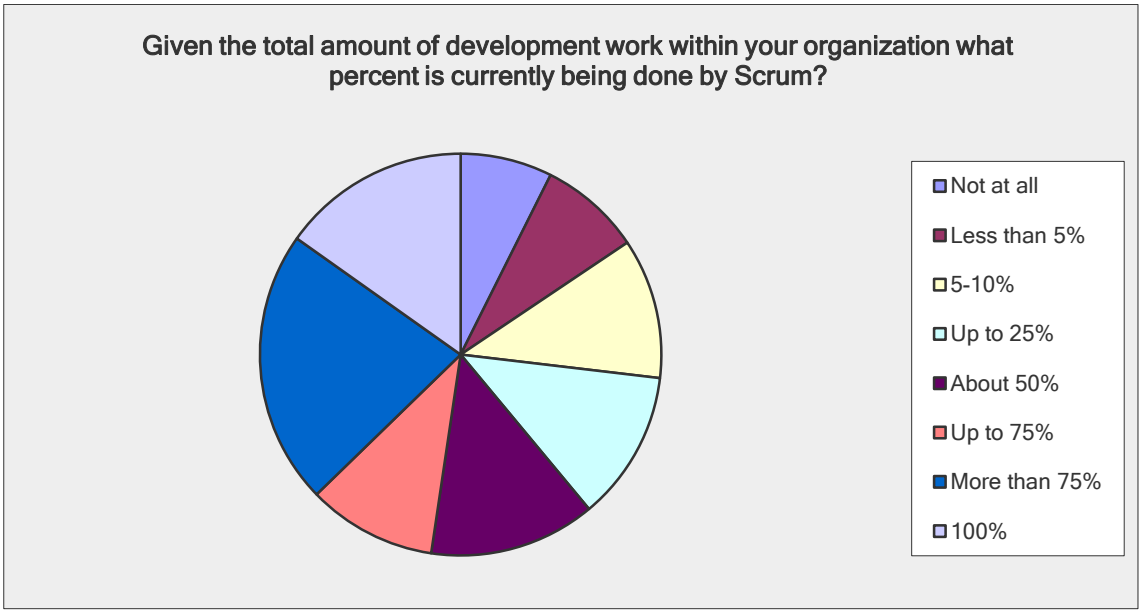
Το γεγονός αυτό συγκεντρώνει αρκετό ενδιαφέρον, αφού φαίνεται πώς η τάση χρησιμοποίησης των ευέλικτων πρακτικών κινείται από την εφαρμογή του Ακραίου Προγραμματισμό προς την υιοθέτηση της μεθοδολογίας Scrum, η οποία κερδίζει ολοένα και περισσότερους χρήστες.

5.5.4 Εφαρμογή της μεθοδολογίας Scrum

Σχετικά με τη χρησιμοποίηση της μεθόδου Scrum, το 5,2% των ερωτώμενων απάντησε ότι δεν έχουν χρησιμοποιήσει καθόλου τη μέθοδο Scrum, το 9,5% απάντησαν ότι δοκίμασαν πιλοτικά τη μέθοδο Scrum, κατά το χρονικό διάστημα που έλαβε χώρα η έρευνα, το 10,8% έχουν δοκιμάσει πιλοτικά τη μέθοδο Scrum αλλά δεν έχουν αποφασίσει ακόμα την υιοθέτησή της, το 14,7% έχει μόλις υιοθετήσει τη μέθοδο Scrum, το 27,3% απάντησαν ότι η μέθοδος Scrum είναι ένας από τους βασικούς τρόπους που χρησιμοποιούν για την ανάπτυξη λογισμικού, ενώ το μεγαλύτερο ποσοστό (32,5%) απάντησαν ότι η μέθοδος Scrum είναι ο μόνο τρόπος που χρησιμοποιούν για την ανάπτυξη λογισμικού.

Εξετάζοντας τη συχνότητα χρησιμοποίηση της μεθοδολογίας Scrum, οι συμμετέχοντες ρωτήθηκαν για το ποσοστό των έργων που αναπτύσσονται με τη μεθοδολογία Scrum (Σχήμα 5.5). Το 7,4% απάντησαν ότι δε χρησιμοποιείται καθόλου, το 8,2%

χρησιμοποίησαν τη μέθοδο σε λιγότερο από το 5% των έργων, το 11,3% μεταξύ 5 – 10% των έργων, το 12,1% μεταξύ 11 – 25%, το 13,4% μεταξύ 26 – 50%, το 10,4% μεταξύ 51 – 75%, το 22,1% εφάρμοσαν το Scrum σε πάνω από το 76% των έργων, ενώ τέλος το 15,2% των ερωτώμενων εφάρμοσε τη μεθοδολογία Scrum για όλα τα έργα λογισμικού που έχει αναλάβει η εταιρία τους.



Σχήμα 5.5 Ποσοστό έργων που αναπτύσσονται με τη μεθοδολογία Scrum

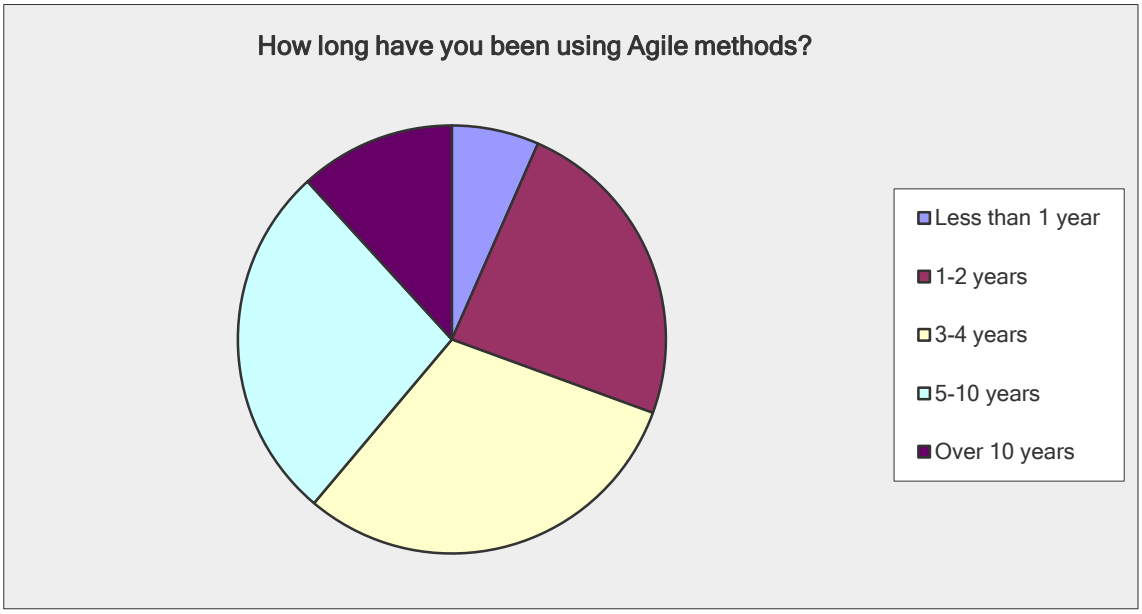
Συνοπτικά παρατηρούμε πώς το 61,1% των ανταποκρινόμενων, χρησιμοποίησαν την μέθοδο Scrum σε περισσότερα από τα μισά έργου λογισμικού που ανέλαβε η εταιρία τους. Επίσης, παρατηρείται πως ένα σχετικά μεγάλο ποσοστό 37,3% εφαρμόζουν σχεδόν αποκλειστικά την μέθοδο Scrum για την ανάπτυξη λογισμικού.

Μεταξύ των ερωτώμενων οι οποίοι υπήρξαν κάποτε Scrum Master σε κάποια ομάδα που εφάρμοσε το Scrum, το 73,7% απάντησαν πώς ο ρόλος αυτός είχε σημαντική επιρροή στην ανάπτυξη του έργου, το 19,2% απάντησαν πώς ο ρόλος αυτός ήταν μάλλον βοηθητικός, το 2,8% απάντησαν πως ο ρόλος του Scrum Master δεν είναι βοηθητικός και τέλος το 4,2% πιστεύει πως ο ρόλος αυτός είναι αχρείαστος. Το μεγαλύτερο ποσοστό 92,9%, θεωρεί τον ρόλο του Scrum Master βοηθητικό κατά την διάρκεια ανάπτυξης του λογισμικού γεγονός που αποδεικνύει πώς ο ρόλος αυτός είναι πράγματι καίριας σημασίας αφού, όπως αναφέρθηκε στο Κεφάλαιο 4, ο Scrum Master είναι υπεύθυνος για την αντιμετώπιση οποιονδήποτε εμποδίων εμφανίζονται και για τον

συντονισμό της ομάδας. Επίσης, σχετικά με την γενικότερη ικανοποίηση των ερωτώμενων όσο αφορά τη μεθοδολογία Scrum, το 86,9% έχουν θετική άποψη, το 8% αρνητική και το 5,1% έχουν ουδέτερη άποψη.

5.5.5 Εφαρμογή των ευέλικτων μεθοδολογιών

Αναφορικά με το χρονικό διάστημα που οι συμμετέχοντες χρησιμοποιούν τις ευέλικτες μεθοδολογίες (Σχήμα 5.6), το 6,6% χρησιμοποιεί τις μεθόδους αυτές για λιγότερο από 1 χρόνο, το 24% μεταξύ 1 – 2 ετών, το 30,6% μεταξύ 3 – 4 ετών, το 27,1% μεταξύ 5 – 10 ετών και το 11,8% χρησιμοποιούν τις Agile μεθόδους για πάνω από 10 χρόνια.

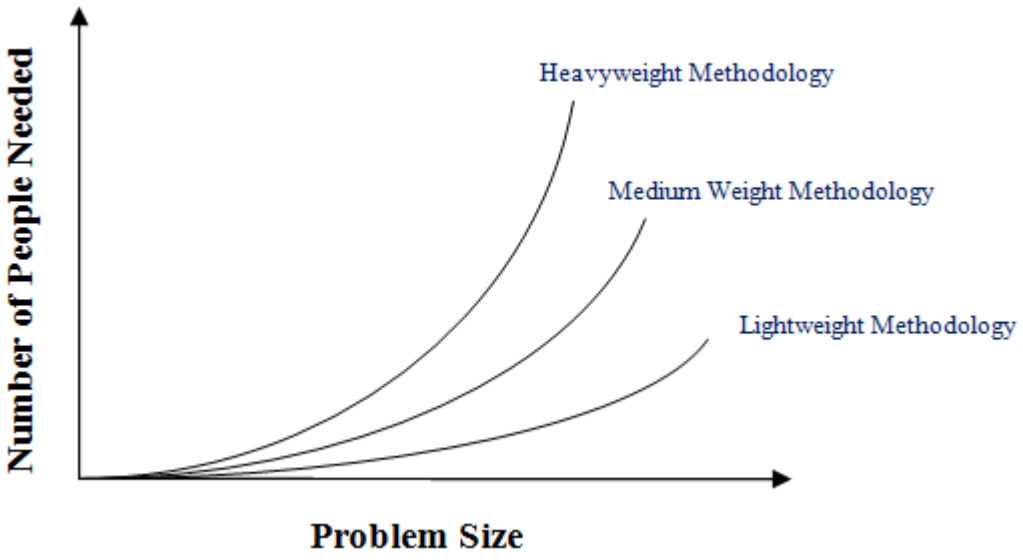


Σχήμα 5.6 Εμπειρία δείγματος σχετικά με τις ευέλικτες μεθοδολογίες

Κατά μέσο όρο, η πλειοψηφία των ερωτηθέντων χρησιμοποιεί τις ευέλικτες πρακτικές για περίπου 5 χρόνια. Οι περισσότερες από τις ευέλικτες μεθοδολογίες εμφανίστηκαν στην δεκαετία του 2000. Επομένως παρατηρούμε πώς δεν υπήρξε ιδιαίτερη καθυστέρηση ώστε να γίνουν ευρέως γνωστές και να χρησιμοποιηθούν. Επίσης, δεδομένου ότι η πλειοψηφία των ερωτηθέντων έχει μεγάλη εμπειρία στην εφαρμογή ευέλικτων μεθόδων, οι απαντήσεις στην παρούσα έρευνα μπορούν να θεωρηθούν ως έγκυρες.

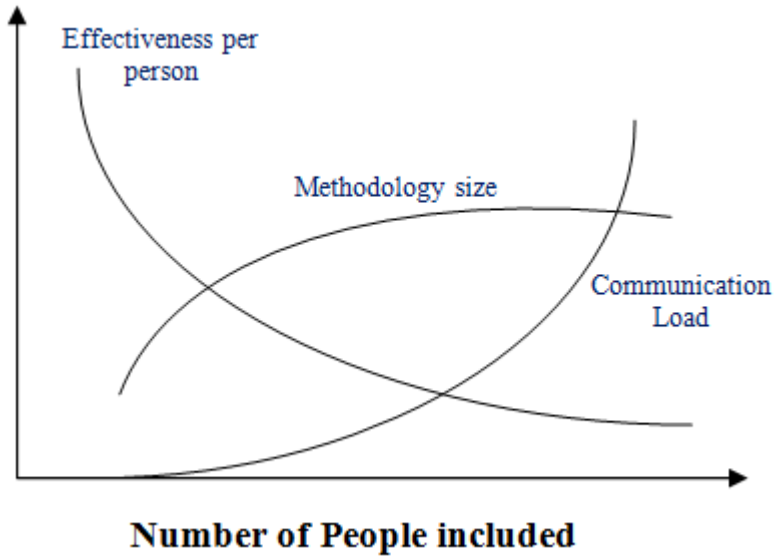
Σχετικά με το μέγεθος των ομάδων που έχουν δουλέψει ποτέ σε έργο λογισμικού εφαρμόζοντας κάποια ευέλικτη μεθοδολογία, το 14,3% των εταιριών έχει χρησιμοποιήσει μεταξύ 1 – 5 άτομα, το 35,2% μεταξύ 6 – 10 άτομα, το 25,2% μεταξύ 11 – 20 άτομα, το 12,2% μεταξύ 21 – 50 άτομα, το 7% μεταξύ 51 – 100 άτομα, ενώ μόλις το 6,1% χρησιμοποίησαν πάνω από 100 άτομα σε κάποιο έργο.

Ένας από τους περιορισμούς των ευέλικτων μεθόδων είναι το μέγεθος του έργου. Τα βασικά στοιχεία ενός έργου λογισμικού είναι το μέγεθος του προϋπολογισμού του έργου, η χρονική διάρκεια του έργου και η οργάνωση της ομάδας. Όσο μεγαλύτερη είναι η ομάδα ή όσο μεγαλύτερος είναι ο προϋπολογισμός, τότε τόσο μεγαλύτερο θα είναι και το έργο. Έτσι υπάρχουν σίγουρα περισσότερες απαιτήσεις, και ως εκ τούτου απαιτούνται περισσότερα άτομα και καλύτερος συντονισμός. Οι παραδοσιακές μεθοδολογίες υποστηρίζουν μεγάλης έκτασης έργα λογισμικού, παρέχοντας σχέδια, έγγραφα και διαδικασίες για την καλύτερη επικοινωνία και συντονισμό σε μεγάλες ομάδες. Όπως φαίνεται στο Σχήμα 5.7 παρακάτω, ο Alistair Cockburn, ένας από τους ιδρυτές του Agile Alliance, ισχυρίζεται ότι για ένα δεδομένο μέγεθος προβλήματος, *«Χρειάζονται λιγότεροι άνθρωποι, αν θα εφαρμοστεί μια ευέλικτη (Lightweight) μέθοδος, και περισσότεροι άνθρωποι εάν θα χρησιμοποιηθεί μια παραδοσιακή (Heavyweight) προσέγγιση»* και υποστηρίζει ότι, *«Υπάρχει ένα όριο στο μέγεθος του προβλήματος που μπορεί να λυθεί με ένα συγκεκριμένο αριθμό ατόμων»* [*Cockburn, 1999*].



Σχήμα 5.7 Πως το μέγεθος του έργου επηρεάζει το μέγεθος της ομάδας για διαφορετικούς τύπους μεθοδολογιών [*Cockburn, 1999*]

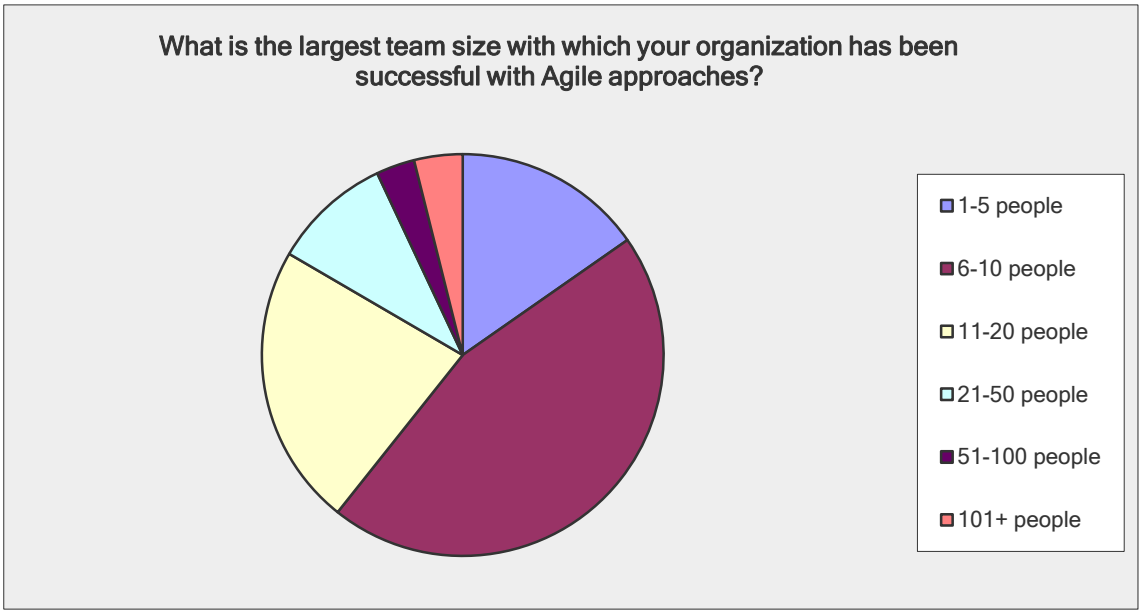
Όσο μεγαλύτερη είναι η ομάδα τόσο επηρεάζεται η επικοινωνία σε ένα έργο και η αποτελεσματικότητα των μελών της ομάδας. Το Σχήμα 5.8 δείχνει πως όσο το φορτίο της επικοινωνίας αυξάνεται, τόσο μειώνεται η αποτελεσματικότητα των ατόμων που συμμετέχουν σε μια ομάδα. Ο Cockburn αναφέρει ότι η μεθοδολογία είναι ένα θέμα για το συντονισμό και τη διαχείριση της επικοινωνίας των ανθρώπων, και ως εκ τούτου το επίπεδο της μεθοδολογίας θα πρέπει να αυξάνεται καθώς ο αριθμός των ανθρώπων μεγαλώνει. Αυτό καθιστά πιο δύσκολη τη χρήση μεθόδων με ευέλικτες ομάδες άνω των 40 ατόμων. Οι παραδοσιακές μεθοδολογίες αποτελούν μια προτιμότερη επιλογή για τις μεγάλες ομάδες. Ωστόσο, ο Ken Schwaber, ένας από τους προταθέντες του Scrum, διαφωνεί με αυτό που δηλώνει, *«...μεγάλες ομάδες μπορεί να διασπαστούν σε μικρότερες ομάδες που χρησιμοποιούν Scrums of Scrums»* [Schwaber, 2001].



Σχήμα 5.8 Επίδραση του μεγέθους ενός έργου [*Cockburn, 1999*]

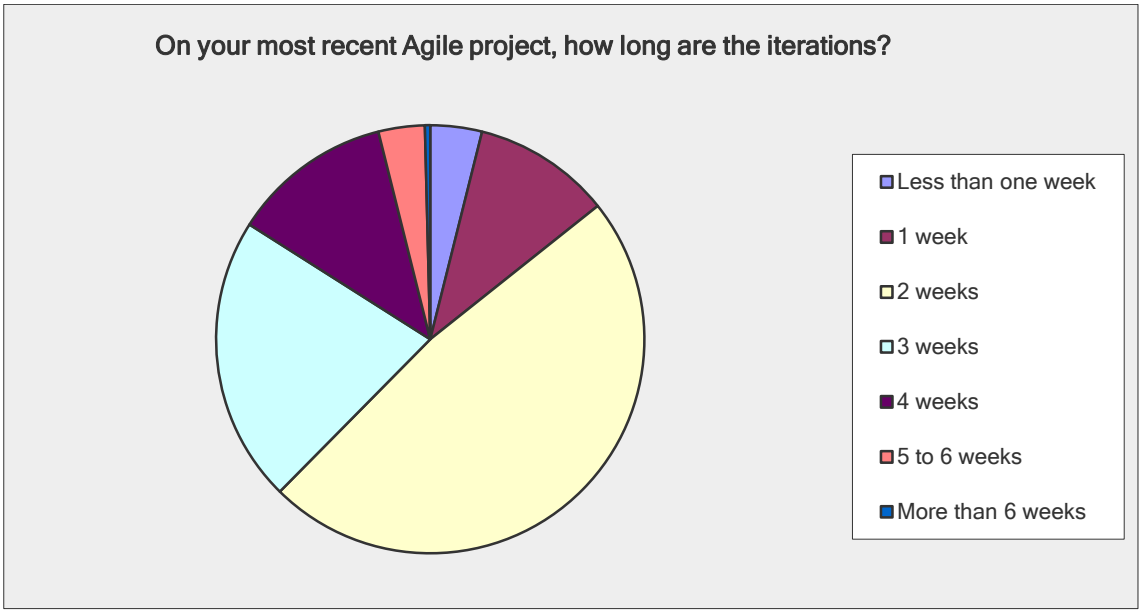
Περαιτέρω διερευνήθηκε το ιδανικό μέγεθος ομάδας που έχει δουλέψει ποτέ σε έργα που εφάρμοσαν κάποια ευέλικτη μεθοδολογία (Σχήμα 5.9). Το 15,3% των εταιριών χρησιμοποίησε μεταξύ 1 – 5 άτομα, το 45,4% μεταξύ 6 – 10 άτομα, το 22,7% μεταξύ 11 – 20 άτομα, το 9,6% μεταξύ 21 – 50 άτομα, το 3,1% μεταξύ 51 – 100 άτομα, ενώ μόλις το 3,9% χρησιμοποίησαν πάνω από 100 άτομα σε κάποιο επιτυχημένο έργο. Συνοψίζοντας, βλέπουμε πώς οι περισσότερες ομάδες (ποσοστό 83,4%) είχαν θετικά αποτελέσματα σε έργα λογισμικού όπου η ομάδα αποτελούταν από 1-20 άτομα, δηλαδή σχετικά μικρές ομάδες. Τα ευρήματα της έρευνας συμφωνούν με αυτά της

βιβλιογραφίας [Cockburn, 2005], που αναφέρουν ότι το μέγεθος μιας ομάδας ιδανικά θα πρέπει να είναι μικρό. Ενώ 8-10 άτομα είναι το βιομηχανικό πρότυπο για μια ευέλικτη ομάδα, και πολλοί σύμβουλοι ευέλικτης ανάπτυξης υποστηρίζουν ότι υπάρχει ένα ιδανικό μέγεθος για μια ομάδα, η αλήθεια είναι ότι οι παράμετροι που μπορούν να καθορίσουν το μέγεθος μιας ομάδας μεταβάλλονται ανάλογα με το έργο. Ο Cockburn συνιστά ένα εύρος μεταξύ 5 και 15 μελών για μια ομάδα, ανάλογα με το μέγεθος και τις ανάγκες του οργανισμού.



Σχήμα 5.9 Ιδανικό μέγεθος ομάδας που εφαρμόζει ευέλικτες προσεγγίσεις

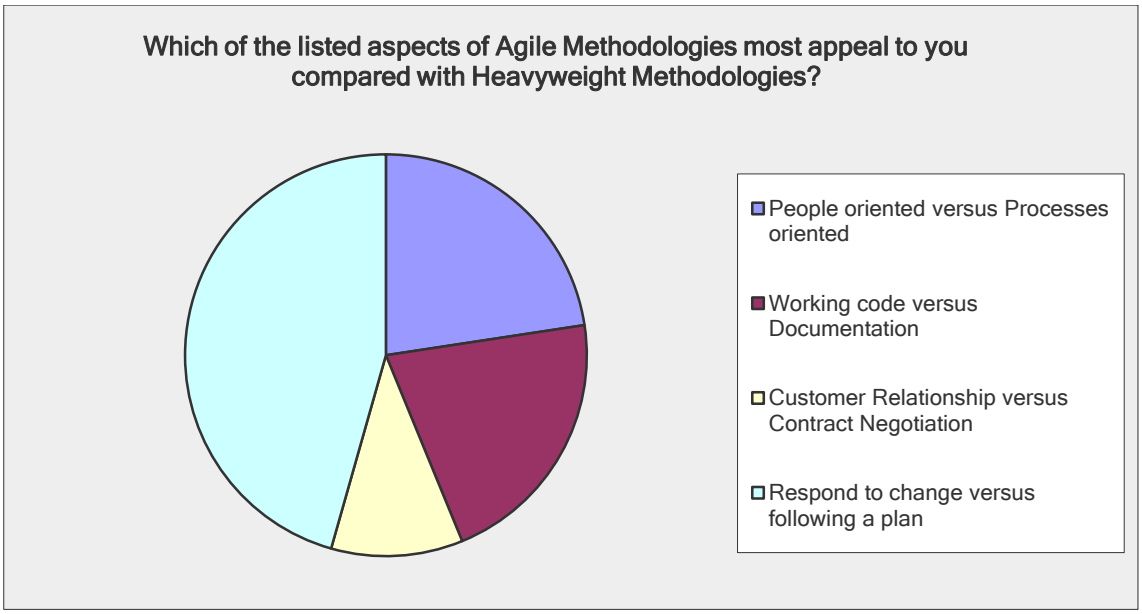
Πολλοί είναι και οι παράγοντες που επηρεάζουν την ποσότητα του χρόνου που μια ευέλικτη ομάδα θα πρέπει να έχει ώστε να επιτύχει τους στόχους μιας επανάληψης (Iteration). Μερικοί από αυτούς τους παράγοντες είναι η γνώμη των πελατών, η εμπειρία της ομάδας, ο χρόνος που αφιερώνεται στις αξιολογήσεις και στον προγραμματισμό και η δυνατότητα να προγραμματιστεί μια επανάληψη. Επιπλέον το μήκος μιας επανάληψης εξαρτάται από το χρονικό διάστημα που ο πελάτης ορίζει ώστε να εξετάσει την πρόοδο του συστήματος. Σχετικά με το χρονικό διάστημα που διήρκησε κάθε επανάληψη στο τελευταίο έργο που υλοποιήθηκε με κάποια Agile μέθοδο (Σχήμα 5.9), το 3,9% των ερωτηθέντων απάντησε πώς διήρκησε λιγότερο από μία βδομάδα, το 10,4% 1 εβδομάδα, το 48,1% 2 εβδομάδες, το 21,6% 3 εβδομάδες, το 12,1% 4 εβδομάδες, το 3,5% μεταξύ 5 – 6 εβδομάδων, ενώ μόλις το 0,6% διήρκησε πάνω από 6 εβδομάδες.



Σχήμα 5.10 Χρονικό διάστημα επαναλήψεων σε ευέλικτα έργα

Πολλοί υποστηρίζουν πώς το ιδανικότερο μήκος για μια επανάληψη είναι ένα διάστημα 2 εβδομάδων. Από τα αποτελέσματα της έρευνας παρατηρούμε πως η πλειοψηφία των ανταποκρινόμενων εφαρμόζει το διάστημα των 2 εβδομάδων ενώ αρκετοί (ποσοστό 33,7%) εφαρμόζουν επαναλήψεις που διαρκούν 3-4 εβδομάδες. Σε κάθε περίπτωση ελάχιστοι είναι αυτοί που υπερβαίνουν τον ένα μήνα.

Οι ερωτώμενοι κλήθηκαν να επιλέξουν την οπτική με την οποία αντιμετωπίζουν τις ευέλικτες μεθοδολογίες συγκριτικά με τις παραδοσιακές μεθόδους. Το 22,6% των συμμετεχόντων απάντησαν την ανθρωποκεντρική έναντι της προσανατολισμένης σε διαδικασίες οπτική, το 21,2% απάντησαν τη συγγραφή κώδικα έναντι της τεκμηρίωση, το 10,6% απάντησαν τις σχέσεις των πελατών έναντι τη διαπραγμάτευσης συμφωνιών, ενώ το 45,6% απάντησαν την ευελιξία στην αλλαγή έναντι στην προσκόλληση στο πλάνο. Τα αποτελέσματα παρουσιάζονται παρακάτω γραφικά στο Σχήμα 5.11.



Σχήμα 5.11 Προτιμότερη οπτική των ευέλικτων μεθόδων έναντι των παραδοσιακών

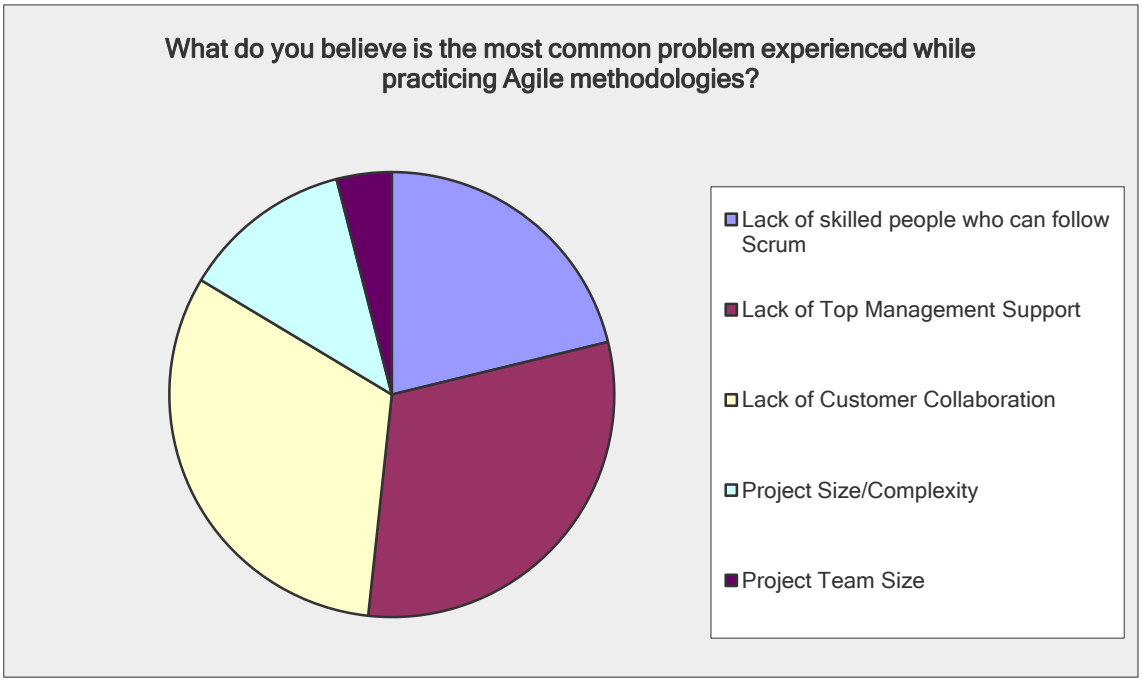
Τα αποτελέσματα αυτά συμφωνούν με τη βιβλιογραφία που, όπως θα μελετηθεί εκτενεστέρα στο επόμενο κεφάλαιο, εμφανίζει ως μερικά από τα κυριότερα πλεονεκτήματα τα παραπάνω, με πρώτο και σημαντικότερο αυτό της ευελιξίας στις αλλαγές [Aguanno, 2004; Biju, 2010; Ionel, 2008].

Στην περίπτωση της οπτικής των ευέλικτων μεθόδων, που οι ερωτώμενοι δεν προτιμούν, η λιγοστή τεκμηρίωση του έργου αποδόθηκε ως ένα μειονέκτημα. Ομοίως η έλλειψη δομής σε ένα έργο είναι ένα άλλο μείζον θέμα. Συγκεκριμένα, το 35,2% των ερωτώμενων απάντησαν τη λιγοστή τεκμηρίωση του έργου, το 16,6% τον μη επαρκή προγραμματισμό, το 10,1% τον περιορισμένο έλεγχο, ενώ το 38,2% την έλλειψη δομής του έργου.

Παράλληλα συγκεντρώθηκαν πληροφορίες από τους ερωτηθέντες ως προς το ποια ήταν τα κυριότερα εμπόδια στην εφαρμογή ευέλικτων μεθόδων. Το Σχήμα 5.12 δείχνει ότι η έλλειψη ειδικευμένων ατόμων που μπορούν να ακολουθήσουν ευέλικτες μεθοδολογίες, ήταν ένας κύριος παράγοντας. Όπως αναφέρει ο Dan Mark, *«Χρειάζονται καλοί άνθρωποι με κίνητρα. Οι Agile μεθοδολογίες είναι μια σκληρή δουλειά και απαιτούν πολύ υψηλό βαθμό πειθαρχίας για να επιτύχουν»* [Marks, 2002]. Ορισμένες εταιρίες αναλαμβάνουν οι ίδιες να παρέχουν την απαραίτητη εκπαίδευση στους υπαλλήλους τους. Για παράδειγμα, η IBM εισήγαγε ένα ευέλικτο πρόγραμμα σε νυχτερινό σχολείο

που εκπαιδεύει τα μέλη του προσωπικού της σχετικά με τη χρήση των ευέλικτων μεθόδων, ενώ ταυτόχρονα εργάζονται σε άλλα έργα λογισμικού [West, 2010].

Το 12,4% των ερωτηθέντων συμφώνησε ότι το μεγαλύτερο εμπόδιο στη χρήση των ευέλικτων μεθόδων είναι το μέγεθος και η πολυπλοκότητα του έργου. Αυτό υποστηρίζει το επιχείρημα που αναφέρθηκε προηγουμένως, πως το μέγεθος του έργου αυξάνει τον αριθμό των ατόμων, αυξάνοντας έτσι και την ανάγκη για επικοινωνία.

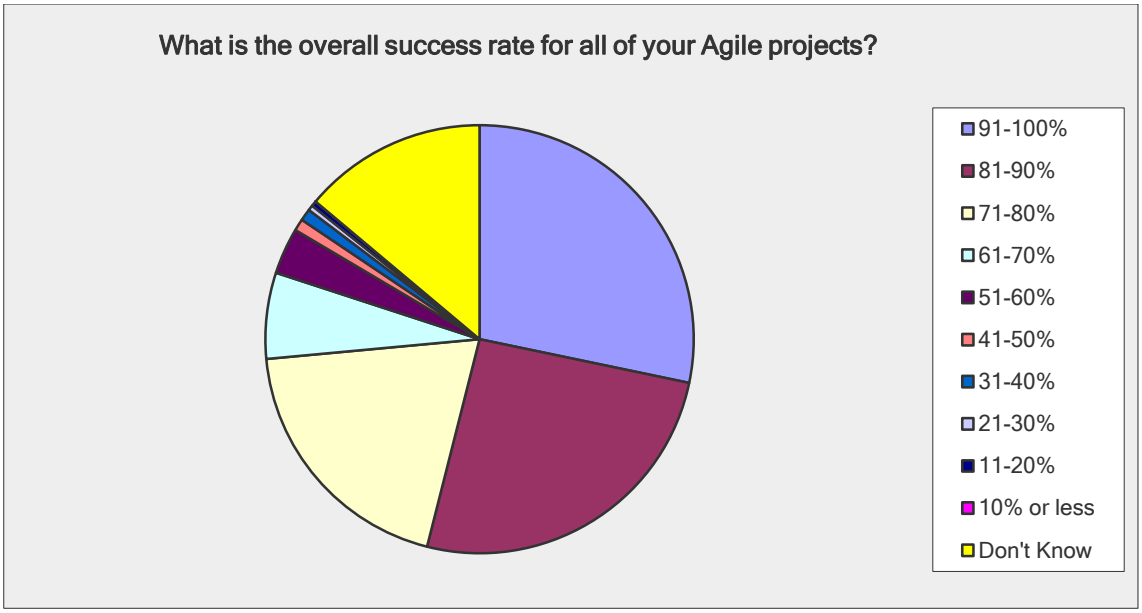


Σχήμα 5.12 Κυριότερα εμπόδια στην εφαρμογή ευέλικτων μεθοδολογιών

Το 31,9% απάντησαν ότι το μεγαλύτερο πρόβλημα είναι η έλλειψη συνεργασίας με τους πελάτες. Η ενεργός συμμετοχή των χρηστών και η στενή συνεργασία είναι απαραίτητη σε όλη τη διάρκεια του κύκλου ανάπτυξης. Αυτός είναι ίσως ο πιο έγκυρος τρόπος, ώστε να εξασφαλιστεί η παράδοση του σωστού προϊόντος. Είναι η θεμελιώδης αρχή που εξασφαλίζει ότι οι προσδοκίες και οι απαιτήσεις του πελάτη θα τύχουν υλοποίησης. Η έλλειψη, λοιπόν, συνεργασίας με τον πελάτη θα αποτελούσε σίγουρα ένα ουσιαστικό πρόβλημα. Τέλος μόλις το 4% υποδεικνύουν το μέγεθος της ομάδας σαν πρόβλημα προφανώς γιατί οι περισσότερες ομάδες, όπως είδαμε σε προηγούμενη ερώτηση, αποτελούνταν από μικρό αριθμό ατόμων γεγονός που επηρεάζει σημαντικά την επικοινωνία και τον προγραμματισμό των μελών της ομάδας. Οι ευέλικτες

μεθοδολογίες βασίζονται σε μεγάλο βαθμό στην επικοινωνία, γι' αυτό και μεγάλες ομάδες είναι δύσκολο να χρησιμοποιήσουν ευέλικτες μεθόδους.

Είναι δύσκολο να συλλέξουμε μια καλή εκτίμηση των ποσοστών επιτυχίας των έργων που εφάρμοσαν ευέλικτη ανάπτυξη, επειδή δεν υπάρχει ένας κοινός τυποποιημένος ορισμός της επιτυχίας. Έτσι, αν ορίσουμε την επιτυχία, για παράδειγμα ως «έγκαιρη παράδοση λογισμικού, εντός προϋπολογισμού, σύμφωνα με τις προδιαγραφές», ο ορισμός θα ισχύει για ορισμένα έργα, αλλά όχι για όλα. Έτσι, θα μπορούσαμε να πάρουμε πιθανώς μια ακριβή εκτίμηση με βάση αυτά τα κριτήρια, αλλά δεν θα είναι το πραγματικό ποσοστό επιτυχίας που αντιπροσωπεύει την βιομηχανία λογισμικού. Για τον λόγο αυτό οι συμμετέχοντες στην έρευνα κλήθηκαν να επιλέξουν οι ίδιοι το συνολικό ποσοστό επιτυχίας των έργων που διεξήχθησαν με κάποια ευέλικτη μέθοδο (Σχήμα 5.13). Το 28,3% δήλωσαν ότι το ποσοστό επιτυχίας είναι μεταξύ 91 – 100%, το 25,7% είχαν ποσοστό επιτυχίας μεταξύ 81 – 90%, το 19,6% μεταξύ 71 – 80%, το 6,5% μεταξύ 61 – 70%, το 3,5% μεταξύ 51 – 60%, το 13,9% δε γνωρίζει το ακριβές ποσοστό, ενώ ένα πολύ μικρό ποσοστό 1,6% απάντησε ότι υπήρχε ποσοστό επιτυχίας μεταξύ 1 – 50%.

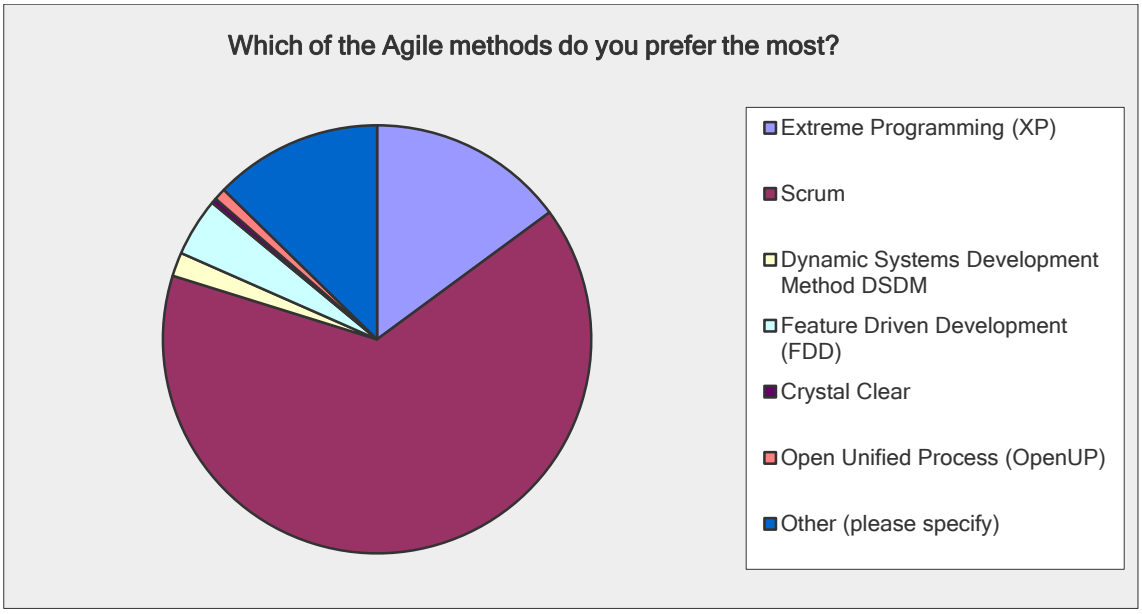


Σχήμα 5.13 Ποσοστό επιτυχίας των έργων που έγιναν με κάποια ευέλικτη μέθοδο

Αποδεικνύεται ότι είναι δύσκολο να διαπιστώσετε στατιστικά την επιτυχία των ευέλικτων μεθόδων. Μία από τις πιο έγκυρες έρευνες το 2010 [Dr. Dobbs IT Project

Success Rate survey], έδειξε πώς τα ποσοστά επιτυχίας για ευέλικτες/επαναληπτικές μεθοδολογίες ήταν περίπου 60%, και τα παραδοσιακά έργα καταρράκτη στο 47%. Στην παρούσα έρευνα συνολικά ποσοστό 73,6% των συμμετεχόντων δήλωσαν πώς οι ευέλικτες μεθοδολογίες είχαν ποσοστό επιτυχίας 70-100%. Αυτή είναι μια βελτίωση, γεγονός που αποδεικνύει την αποτελεσματικότητα των μεθόδων αυτών.

Εξετάζοντας ποιά Agile μεθοδολογία προτιμούν περισσότερο οι συμμετέχοντες, το 64,9% αυτών δήλωσαν πως προτιμούν τη μέθοδο Scrum (Σχήμα 5.14), το 14,9% τη μέθοδο Extreme Programming, το 4,4% τη Feature Driven Development, το 1,8% τη Dynamic Systems Development Method, το 0,9% την Open Unified Process, το 0,4% την Crystal Clear, ενώ το 12,7% προτιμούν άλλες μεθοδολογίες με κυριότερη απάντηση τη χρησιμοποίηση συνδυασμού δύο μεθοδολογιών από τις παραπάνω. Η μεθοδολογία Scrum παρατηρούμε ότι συγκεντρώνει τις περισσότερες προτιμήσεις.



Σχήμα 5.14 Προτιμότερη ευέλικτη μεθοδολογία

Τέλος, το 90,4% των ανταποκρινόμενων, πιστεύουν ότι οι εταιρίες τους θα ξαναχρησιμοποιήσουν στο μέλλον ευέλικτες πρακτικές σε κάποιο έργο λογισμικού, γεγονός που αποδεικνύει ότι οι οργανώσιμοι που υιοθετούν ευέλικτες μεθοδολογίες έχουν σημαντικά οφέλη. Το 7,8% δε γνωρίζουν το ενδεχόμενο επαναχρησιμοποίησης κάποιας ευέλικτης μεθόδου στο μέλλον, ενώ μόλις το 1,7% δίνουν αρνητικά απάντηση.

Είναι αρκετά σαφές ότι οι ευέλικτες μέθοδοι ανάπτυξης λογισμικού κερδίζουν αποδοχής εντός της βιομηχανίας λογισμικού.

5.5.6 Κατανεμημένη ευέλικτη ανάπτυξη λογισμικού (Offshore development)

Μία από τις θεμελιώδεις αρχές της ευέλικτης ανάπτυξης λογισμικού είναι η σημασία της επικοινωνίας μεταξύ των διαφόρων ατόμων που εμπλέκονται στην ανάπτυξη. Επιπλέον, οι ευέλικτες μέθοδοι προωθούν την βελτίωση της επικοινωνίας μέσα από την επικοινωνία πρόσωπο με πρόσωπο. Όπως το ευέλικτο μανιφέστο αναφέρει «Ο καλύτερος τρόπος μεταβίβασης και ανταλλαγής πληροφοριών με την ομάδα παραγωγής είναι η διαπροσωπική συζήτηση και οι συνομιλίες.» Ο Ακραίος Προγραμματισμός τονίζει αυτή την αρχή με την πρακτική του η ομάδα να εργάζεται σε έναν ενιαίο ανοιχτό χώρο, όπου οι άνθρωποι μπορούν να συνεργαστούν στενά.

Μια τάση που πρόσφατα άρχισε να παρουσιάζεται στην παγκόσμια ανάπτυξη λογισμικού, είναι η κατανεμημένη ανάπτυξη (Offshore development) όπου μεγάλο μέρος της εργασίας γίνεται από χαμηλότερο αμειβόμενο προσωπικό. Η Offshore ανάπτυξη έρχεται σε αντίθεση με την ευέλικτη ανάπτυξη αφού δεν μπορεί να εφαρμοστεί η έννοια της φυσικής επικοινωνίας, εφόσον οι προγραμματιστές είναι πολύ μακριά [Fowler, 2006].

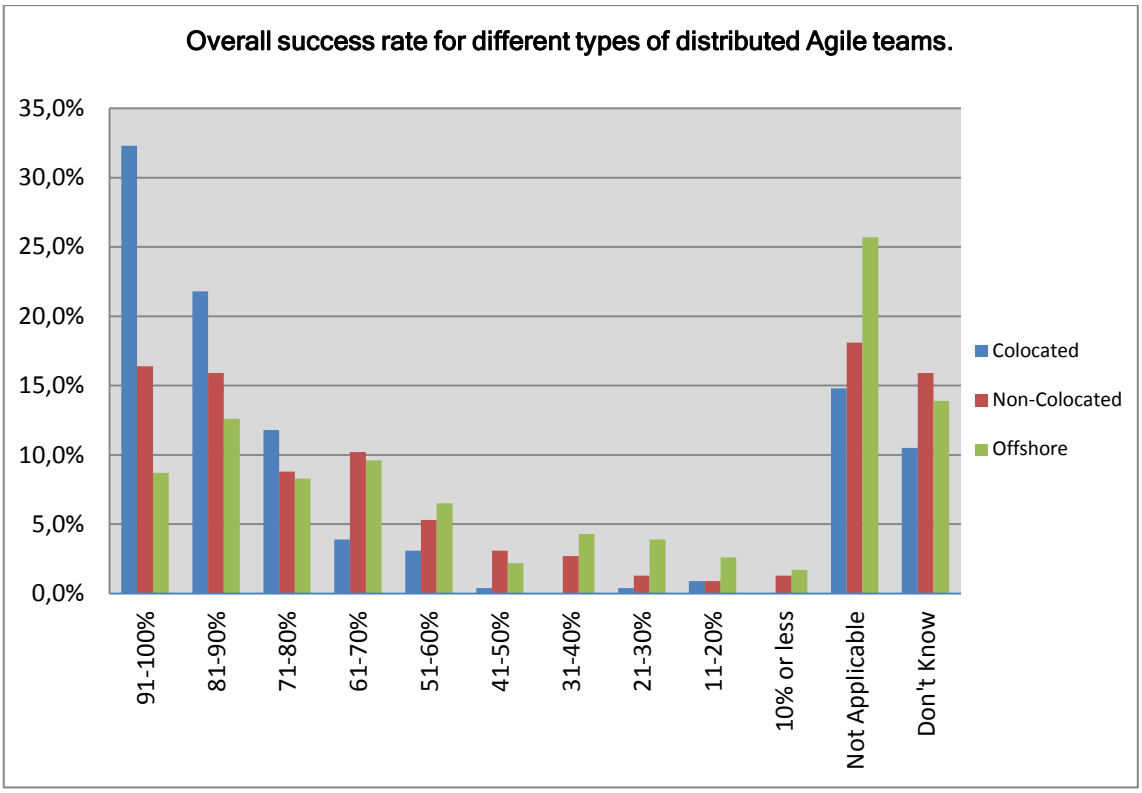
Ωστόσο η ευέλικτη ανάπτυξη μπορεί να πραγματοποιηθεί και να φέρει καλά αποτελέσματα σε κατανεμημένες ομάδες εφόσον γίνουν ορισμένες τροποποιήσεις για να μπορεί η ομάδα να λειτουργήσει αποδοτικά. Μερικά από τα πράγματα που θα μπορούσαν να εφαρμοστούν είναι τα εξής:

- Περισσότερες λεπτομέρειες και καλύτερη δυνατή κατανοητή τεκμηρίωση λειτουργικών προδιαγραφών από τον πελάτη.
- Χρήση συνεργατικών εργαλείων για να έχουν πιο αποτελεσματική επικοινωνία σε πραγματικό χρόνο.
- Χρήση ειδικής πλατφόρμες διαχείρισης της εργασίας, έτσι ώστε ο προγραμματισμός της ομάδας να μπορεί να γίνει αποτελεσματικά.
- Χρήση ειδικών εργαλείων διαχείρισης που επιτρέπουν τη συνεχή αλληλεπίδραση με τους χρήστες, παρέχοντας οργανωμένη πρόσβαση σε αυτούς.

Το 2008, ο Dr.Dobb διεξήγαγε μια έρευνα με 642 συμμετέχοντες ώστε εξερευνήσει την υιοθέτηση και τα ποσοστά επιτυχίας της ευέλικτης ανάπτυξης λογισμικού. Ο Scott Ambler παρουσίασε μια ανάλυση της έρευνας και διαπιστώθηκαν πολύ ενδιαφέροντα στατιστικά στοιχεία σχετικά με τα ποσοστά επιτυχίας των Agile ομάδων. Οι Colocated ομάδες, όπου ο καθένας, συμπεριλαμβανομένων των πελατών, βρίσκονταν στον ίδιο ή κοντινό χώρο εργασίας, σημείωσαν ποσοστό επιτυχίας 83% και οι Non-Colocated ομάδες, όπου οι άνθρωποι δε στεγάζονταν στο ίδιο γραφείο αλλά μπορούσαν να βρεθούν σχετικά εύκολα (δηλαδή βρισκόταν στο ίδιο κτίριο ή σε κοντινές γεωγραφικές θέσεις), είχαν κατά μέσο όρο 72% ποσοστό επιτυχίας. Έργα όπου μέλη της ομάδας βρίσκονταν σε διαφορετικές γεωγραφικές θέσεις, Offshore περιοχές, είχαν 60% ποσοστό επιτυχίας κατά μέσο όρο [Dr.Dobb, 2008]. Η έρευνα αυτή του Dr. Dobb απομυθοποιεί την διαπίστωση ότι η ανάπτυξη ευέλικτων έργων μπορεί να εφαρμοστεί μόνο σε μικρές Co-located ομάδες.

Στην παρούσα έρευνα, διερευνήθηκε το ποσοστό επιτυχίας των έργων για τους διαφορετικούς τύπους ομάδων κατανεμημένης ευέλικτης ανάπτυξης (Σχήμα 5.15), όπως περιγράφηκαν πιο πάνω.

Σχετικά με το ποσοστό επιτυχίας των έργων που διεξήχθησαν με κάποια Agile μεθοδολογία από ομάδα (Co-located Team) που βρισκόταν στο ίδιο γραφείο, το 32,3% δήλωσαν ότι το ποσοστό επιτυχίας είναι μεταξύ 91 – 100%, το 21,8% είχαν ποσοστό επιτυχίας μεταξύ 81 – 90%, το 11,8% μεταξύ 71 – 80%, το 3,9% μεταξύ 61 – 70%, το 3,1% μεταξύ 51 – 60%, το 10,5% δε γνωρίζει το ακριβές ποσοστό, το 14,8% απάντησε ότι δεν έχει διεξαχθεί τέτοιο έργο στην εταιρία τους, ενώ ένα πολύ μικρό ποσοστό απάντησε ότι υπήρχε ποσοστό επιτυχίας μεταξύ 1 – 50%.



Σχήμα 5.15 Συνολικό ποσοστό επιτυχίας για κατανεμημένες ομάδες ευέλικτης ανάπτυξης

Όσο αφορά το ποσοστό επιτυχίας των έργων που διεξήχθησαν με κάποια Agile μεθοδολογία από Non-Colocated ομάδες, το 16,4% δήλωσαν ότι το ποσοστό επιτυχίας είναι μεταξύ 91 – 100%, το 15,9% είχαν ποσοστό επιτυχίας μεταξύ 81 – 90%, το 8,8% μεταξύ 71 – 80%, το 10,2% μεταξύ 61 – 70%, το 5,3% μεταξύ 51 – 60%, το 15,9% δε γνωρίζει το ακριβές ποσοστό, το 18,1% απάντησε ότι δεν έχει διεξαχθεί τέτοιο έργο στην εταιρία τους, ενώ ένα πολύ μικρό ποσοστό απάντησε ότι υπήρχε ποσοστό επιτυχίας μεταξύ 1 – 50% (Σχήμα 5.25). Παρατηρούμε ότι το ποσοστό επιτυχίας των έργων που αναπτύχθηκαν από ομάδα που δε στεγαζόταν στο ίδιο γραφείο αλλά μπορούσε να βρεθεί σχετικά εύκολα έχει μειωθεί αισθητά σε σχέση με τις ομάδες που βρίσκονται στο ίδιο γραφείο.

Επιπλέον, το ποσοστό επιτυχίας των ευέλικτων έργων που διεξήχθησαν από ομάδα που δε στεγαζόταν στο ίδιο γραφείο και δε μπορούσαν να βρεθούν εύκολα έχει ως εξής, το 8,7% δήλωσαν ότι το ποσοστό επιτυχίας είναι μεταξύ 91 – 100%, το 12,6% είχαν ποσοστό επιτυχίας μεταξύ 81 – 90%, το 8,3% μεταξύ 71 – 80%, το 9,6% μεταξύ 61 – 70%, το 6,5% μεταξύ 51 – 60%, το 13,9% δε γνωρίζει το ακριβές ποσοστό, το 25,7% απάντησε ότι δεν έχει διεξαχθεί τέτοιο έργο στην εταιρία τους, ενώ ελάχιστοι

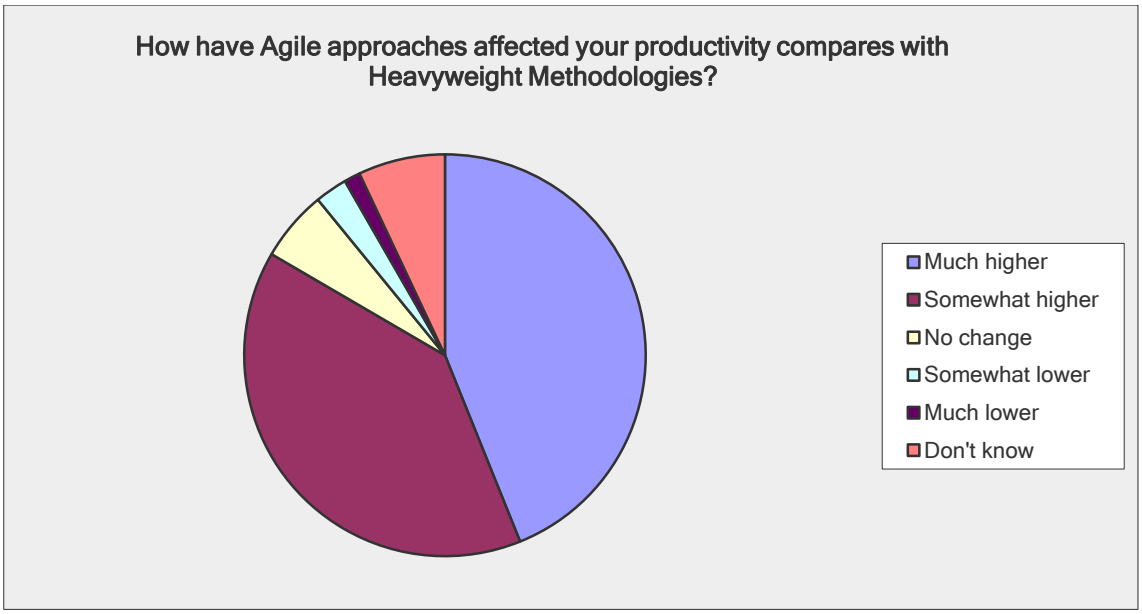
συμμετέχοντες απάντησαν ότι υπήρχε ποσοστό επιτυχίας μεταξύ 1 – 50%. Επίσης, παρατηρούμε ότι το ποσοστό επιτυχίας των έργων που αναπτύχθηκαν από ομάδα που δε στεγαζόταν στο ίδιο γραφείο και δε μπορούσε να βρεθεί σχετικά εύκολα έχει μειωθεί αισθητά σε σχέση με τις ομάδες που βρίσκονται στο ίδιο γραφείο ή τις ομάδες που δε στεγαζόταν στο ίδιο γραφείο αλλά μπορούσαν να βρεθούν σχετικά εύκολα.

Παρατηρούμε επίσης πως σημαντικό είναι και το ποσοστό που δεν εφαρμόζει κατανεμημένη ανάπτυξη. Υπάρχουν ακόμη πολλές διαφορές στις απόψεις για το κόστος και τα οφέλη της χρήσης Offshore ανάπτυξης. Θα μπορούσε να αποτελέσει ένα μεμονωμένο θέμα έρευνας. Το μεγαλύτερο πρόβλημα που αντιμετωπίζει αυτού του είδους η ανάπτυξη, είναι η απόσταση και η αδυναμία φυσικής επικοινωνίας. Είναι δύσκολο να αποδειχθεί ότι μια προσέγγιση είναι καλύτερη από την άλλη. Αυτό που παρατηρείται είναι ότι παρόλο που είναι δύσκολο να εφαρμοστεί, τα οφέλη από την ευέλικτη Offshore ανάπτυξη λογισμικού αυξάνονται σημαντικά (όπως για παράδειγμα η σημαντική μείωση του κόστους).

5.5.7 Επιδράσεις των ευέλικτων μεθοδολογιών σε σύγκριση με τις παραδοσιακές

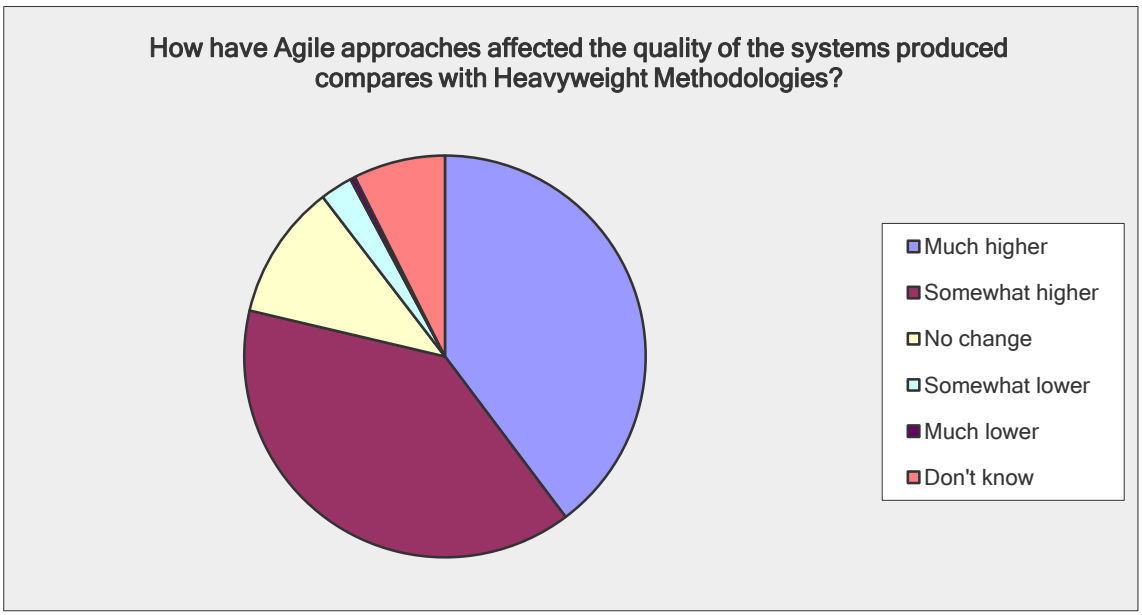
Η ευέλικτη προσέγγιση στην ανάπτυξη λογισμικού φαίνεται πως λειτουργεί πολύ καλά στην πράξη. Στην ενότητα αυτή ο ερωτώμενος έπρεπε να απαντήσει αν πιστεύει ότι η εφαρμογή μιας ευέλικτης μεθοδολογίας αντί μιας παραδοσιακής θα επηρεάσει το κόστος, την ποιότητα του λογισμικού, την παραγωγικότητα και την ικανοποίηση του πελάτη. Τα αποτελέσματα δείχνουν ότι η υιοθέτηση των ευέλικτων μεθόδων σαφώς επιφέρουν καλύτερα αποτελέσματα, συγκριτικά με τις παραδοσιακές.

Η επίδραση που οι ευέλικτες προσεγγίσεις έχουν στην παραγωγικότητα συνοψίζεται στο Σχήμα 5.16. Συγκριτικά με τις παραδοσιακές μεθοδολογίες, το 43,9% των συμμετεχόντων απάντησαν ότι η παραγωγικότητα αυξήθηκε κατά πολύ, το 39,5% δήλωσαν ότι απλά αυξήθηκε, το 5,7% δήλωσαν ότι δεν επηρεάστηκε, το 3,9% αναφέρουν ότι η παραγωγικότητα μειώθηκε, ενώ το 7% δεν γνωρίζουν. Τα στατιστικά αυτά στοιχεία δεν είναι καθόλου κακά εξετάζοντας το ενδεχόμενο της αποτυχίας από κάποιες ομάδες, ιδιαίτερα στην πρώτη προσπάθειά τους.



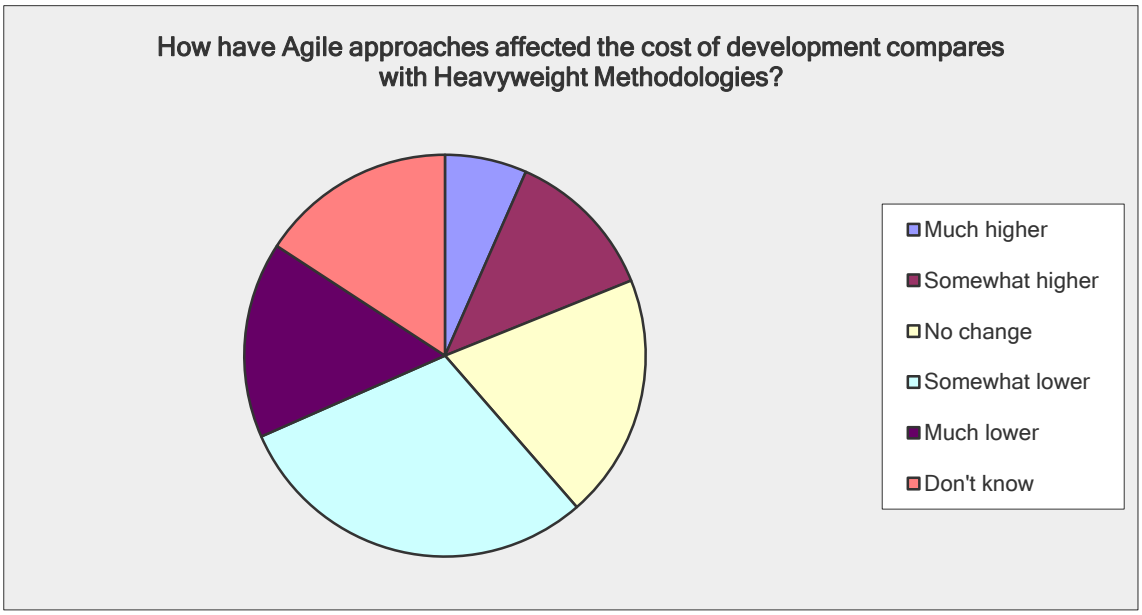
Σχήμα 5.16 Παραγωγικότητα των έργων ευέλικτης συγκριτικά με έργα παραδοσιακής ανάπτυξης

Όταν το ζήτημα της ποιότητας του λογισμικού παρουσιάστηκε στους ερωτηθέντες, τα αποτελέσματα είναι ακόμη καλύτερα. Η πλειοψηφία των ερωτηθέντων (Σχήμα 5.17), ποσοστό 78,6% πιστεύουν πώς τα έργα λογισμικού όπου εφαρμοζόταν ευέλικτη ανάπτυξη έγιναν πιο ποιοτικά συγκριτικά με τα έργα που ακολουθούσαν παραδοσιακή μεθοδολογία. Το 10,9% πιστεύει πώς δεν επηρεάστηκε η ποιότητά τους, το 3% αναφέρει ότι τα ευέλικτα έργα έγιναν λιγότερο ποιοτικά ενώ το 7,4% δε γνωρίζουν.



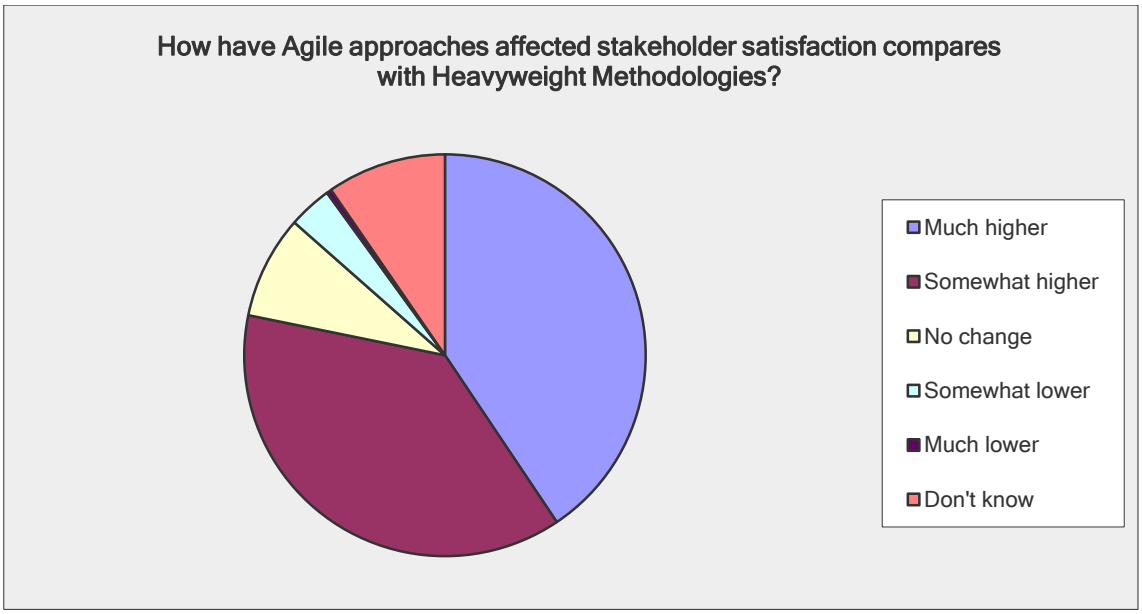
Σχήμα 5.17 Ποιότητα των έργων ευέλικτης συγκριτικά με έργα παραδοσιακής ανάπτυξης

Όσον αφορά το κόστος του λογισμικού (Σχήμα 5.18), οι περισσότεροι συμμετέχοντες (ποσοστό 45,6%) συμφώνησαν ότι η υιοθέτηση μιας ευέλικτης διαδικασία σε σύγκριση με μια παραδοσιακή, έχει ως αποτέλεσμα τη μείωση του κόστους. Περίπου το 18% των ερωτηθέντων ψηφίσαν υπέρ της αύξησης των δαπανών στα έργα με ευέλικτη ανάπτυξη, το 19,7% των ερωτηθέντων πιστεύουν ότι τόσο η παραδοσιακή όσο και η ευέλικτη ανάπτυξη λογισμικού έχουν το ίδιο αντίκτυπο στο κόστος, ενώ 15,8% δε γνωρίζουν.



Σχήμα 5.18 Κόστος των έργων ευέλικτης συγκριτικά με έργα παραδοσιακής ανάπτυξης

Φαίνεται, επίσης, πώς η έμφαση που η ευέλικτη ανάπτυξη λογισμικού δίνει στις συνεργατικές τεχνικές, όπως η ενεργή συμμετοχή των πελατών, συνέβαλαν στη βελτίωση της ικανοποίησης των ενδιαφερόμενων (Stakeholders). Όπως παρουσιάζεται στο Σχήμα 5.19, το 40,6% των ανταποκρινόμενων απάντησαν ότι οι πελάτες έμειναν πολύ πιο ικανοποιημένοι -συγκριτικά πάντα με τις παραδοσιακές μεθοδολογίες-, το 37,6% πιο ικανοποιημένοι, το 8,3% δεν επηρεάστηκε η ικανοποίησή τους, το 3,9% λιγότερο ικανοποιημένοι, ενώ το 9,6% δε γνωρίζουν.



Σχήμα 5.19 Ικανοποίηση των πελατών σε έργα ευέλικτης συγκριτικά με έργα παραδοσιακής ανάπτυξης

Ο Πίνακας 5.1 συγκεντρώνει όλα τα πιο πάνω ευρήματα. Το δείγμα των απαντήσεων περιορίστηκε μόνο στους συμμετέχοντες που έδωσαν σαφή απάντηση, αγνοώντας το δείγμα των απαντήσεων που δεν γνώριζαν.

	Αυξήθηκε	**Καμία αλλαγή**	**Μειώθηκε**
Παραγωγικότητα	**89,6%**	6,12%	4,1%
Ποιότητα	**84,8%**	11,7%	3,2%
Κόστος	22,4%	23,3%	**54,2%**
Ικανοποίηση πελάτη	**86,5%**	9,2%	4,3%

Πίνακας 5.1 Συγκεντρωτικός πίνακας με συνολικά οφέλη των ευέλικτων μεθόδων έναντι των παραδοσιακών προσεγγίσεων

Όπως προκύπτει από την έρευνα, μπορούμε να συμπεράνουμε ότι οι ευέλικτες μέθοδοι παρέχουν ουσιαστικές βελτιώσεις στην διαδικασία ανάπτυξης λογισμικού.

Κεφάλαιο 6

Σύγκριση Scrum με Παραδοσιακή Ανάπτυξη Λογισμικού

6.1 Εισαγωγή

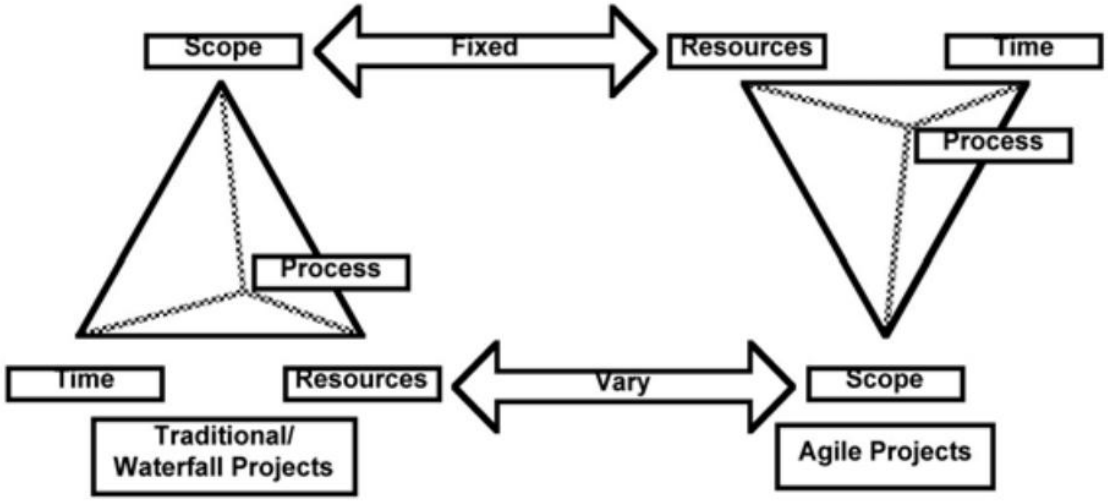
Υπάρχει μια έντονη διαφωνία μεταξύ υποστηρικτών τόσο της παραδοσιακής όσο και της ευέλικτης ανάπτυξης λογισμικού. Ενώ κάποιοι μελετητές θεωρούν τις δύο προσεγγίσεις ανταγωνιστικές μια σημαντική μερίδα από αυτούς υποστηρίζουν ότι οι δύο προσεγγίσεις αλληλοαποκλείονται, αλλά παράλληλα αλληλοσυμπληρώνονται. Πρόκειται για δύο διαφορετικούς τρόπους διαχείρισης έργων, ο κάθε ένας από τους οποίους είναι κατάλληλος για διαφορετικά έργα. Αυτό υποστηρίζεται σε μια μελέτη του [Sauser et al, 2009] ο οποίος κατέληξε στο συμπέρασμα ότι το κάθε έργο είναι μοναδικό και απαιτεί εξειδικευμένο τρόπο διαχείρισης. Για να κατανοήσουμε πλήρως τις διαφορές μεταξύ παραδοσιακής και ευέλικτης ανάπτυξης, στη συνέχεια θα εξετάσουμε και πάλι τις βασικές αρχές τους.

Να σημειωθεί πως γίνεται αναφορά σε στατιστικά στοιχεία που προέκυψαν από την έρευνα που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Το ερωτηματολόγιο απαντήθηκε από άτομα τα οποία είναι γνώστες ή/και έχουν χρησιμοποιήσει ευέλικτη ανάπτυξη λογισμικού. Ως εκ τούτου, το δείγμα των ερωτηθέντων, για τη συγκεκριμένη περίπτωση, είναι πιο αντιπροσωπευτικό των συμμετεχόντων που εφαρμόζουν την μέθοδο Scrum και όχι όλων των συμμετεχόντων.

6.2 Παραδοσιακή έναντι ευέλικτης διαχείρισης έργων

Οι αρχές από τη μηχανική (π.χ. κατασκευές) διαδραμάτισαν καίριο ρόλο στην εμφάνιση και εξέλιξη των παραδοσιακών μεθοδολογιών διαχείριση έργων και κατ' επέκταση ανάπτυξης λογισμικού. Οι παραδοσιακές μεθοδολογίες χρησιμοποιούνταν ευρέως για μεγάλο χρονικό διάστημα και η επιτυχία τους στη βιομηχανία λογισμικού έχει επισημανθεί από διάφορους μελετητές [Papke-Shields et al., 2009]. Η αυξανόμενη πίεση για να παραδοθούν γρήγορα ποιοτικά προϊόντα λογισμικού σε μια δυναμική και συνεχώς μεταβαλλόμενη παγκόσμια αγορά, έφερε στην επιφάνεια τις ευέλικτες μεθόδους οι οποίες προέκυψαν από τις προκλήσεις των έργων που χαρακτηρίζονταν από αβεβαιότητα και αστάθεια.. Σε αντίθεση με τις παραδοσιακές προσεγγίσεις, η ευέλικτη ανάπτυξη δίνει μεγαλύτερη έμφαση στην επικοινωνία και όχι σε μια διαδικασία ή σχέδιο και ως εκ τούτου οι δύο διαφέρουν ως προς πεδίο εφαρμογής τους.

Το Σχήμα 6.1 πιο κάτω απεικονίζει την εννοιολογική διαφορά μεταξύ παραδοσιακών και ευέλικτων πρακτικών [Owen et al., 2006]. Σε αντίθεση με την παραδοσιακή διαχείριση έργου, η οποία επικεντρώνεται στον προγραμματισμό ακολουθώντας ένα σταθερό σχέδιο (Scope), οι ευέλικτες μέθοδοι θεωρούν ότι η λειτουργικότητα του έργου επηρεάζει τον προγραμματισμό και γι' αυτό το ακολουθούμενο σχέδιο μεταβάλλεται συνεχώς, ενώ οι πόροι για το έργο (χρόνος και άνθρωποι) είναι σταθεροί.



Σχήμα 6.1 Διαφορές μεταξύ παραδοσιακών και ευέλικτων μεθοδολογιών

Προφανώς, η ευέλικτη προσέγγιση διαφέρει σημαντικά από τον παραδοσιακό τρόπο διαχείρισης έργων, όπου οι τεχνικές που εφαρμόζονται δεν είναι κατάλληλες να διαχειριστούν τις αλλαγές που προκύπτουν ενόσω το σχέδιο ανάπτυξης βρίσκεται σε λειτουργία. Σύμφωνα με τους [Augustine and Woodcock, 2008], οι ευέλικτες μεθοδολογίες διαφέρουν από τις παραδοσιακές και αυτό οφείλεται σε διάφορους παράγοντες σχετικά με την διαχείριση των αλλαγών των απαιτήσεων ενός έργου, τον έλεγχο, την οργάνωση καθώς και τον τρόπο αντιμετώπισης των προβλημάτων. Μια ανασκόπηση της βιβλιογραφίας δείχνει ότι οι δύο προσεγγίσεις διαφέρουν κυρίως με βάση τα ακόλουθα χαρακτηριστικά (Πίνακας 6.1) [Augustine and Woodcock, 2008; Hoda et al, 2008].

Παραδοσιακή διαχείριση έργων	**Ευέλικτη διαχείριση έργων**
Έμφαση στις διαδικασίες και το σχέδιο.	Έμφαση στους ανθρώπους.
Έμφαση στην ανάπτυξη όλων των τμημάτων πρώτα όπως ορίζει το σχέδιο.	Έμφαση σε ένα τμήμα, υλοποίηση του και έπειτα ακολουθεί το επόμενο.
Οι αλλαγές δύσκολα γίνονται κατορθωτές και επιφέρουν σημαντικό κόστος.	Ο κανονισμός των αλλαγών γίνεται με ευέλικτες και προσαρμόσιμες διαδικασίες.
Τα μέλη της ομάδας δουλεύουν ανεξάρτητα, λιγότερη συνεργασία.	Τα μέλη της ομάδας συνεργάζονται σε όλα τα στάδια της ανάπτυξης.
Το προϊόν αναπτύσσεται σειριακά με βάση ιεραρχικές οργανωτικές δομές.	Το προϊόν προκύπτει σαν αποτέλεσμα επαναλήψεων που παράγουν τμηματικές επαυξήσεις του τελικού προϊόντος.
Αυξημένος έλεγχος ανάμεσα στα μέλη της ομάδας. Ελεγχόμενος τύπος διοίκησης.	Η ομάδα αυτοδιοικείται και αυτό-οργανώνεται. Ο ρόλος της διοίκησης είναι να διευκολύνει και να παρέχει υποστήριξη.
Ο πελάτης εμπλέκεται μόνο κατά την φάση συλλογής των απαιτήσεων και κατά την παράδοση του προϊόντος.	Ο πελάτης εμπλέκεται ενεργά και συνεχώς καθ 'όλη τη διάρκεια ζωής του έργου.
Οι εργασίες αναλύονται και προγραμματίζονται μαζικά πριν την έναρξη της υλοποίησης.	Οι επαναληπτικές προσεγγίσεις σε επιλεγμένες εργασίες με συνεχή ανατροφοδότηση από τα μέλη της ομάδας

	και τους πελάτες, οδηγούν σε σταδιακή πρόοδο σε σύντομο χρονικό διάστημα.
Τα έργα και οι κίνδυνοι είναι επαρκώς προβλέψιμα και είναι δυνατόν να διαχειρίζονται μέσω λεπτομερών και περίπλοκων σχεδίων εκ των προτέρων.	Είναι αδύνατο να προβλεφτεί το μέλλον, διότι τα έργα και οι κίνδυνοι είναι απροσδιόριστα εξαιτίας της αβεβαιότητας και ως εκ τούτου δεν υπάρχει ανάγκη για λεπτομερή σχεδιασμό εκ των προτέρων.
Οι δοκιμές γίνονται στο τέλος του έργου.	Οι δοκιμές γίνονται συνεχώς στο τέλος κάθε επανάληψης.
Η τεκμηρίωση είναι λεπτομερής.	Η τεκμηρίωση γίνεται μόνο όταν χρειάζεται.

Πίνακας 6.1 Διαφορές παραδοσιακής και ευέλικτης διαχείρισης έργων λογισμικού [Augustine and Woodcock, 2008; Hoda et al, 2008]

6.3 Scrum έναντι παραδοσιακών μεθοδολογιών ανάπτυξης λογισμικού

Τα αποτελέσματα από την έρευνα που παρουσιάστηκε στο προηγούμενο κεφάλαιο, έδειξαν πώς προτιμότερη παραδοσιακή μεθοδολογία είναι αυτή του καταρράκτη. Για δεκαετίες, τα έργα ανάπτυξης λογισμικού ακολουθούσαν το κλασικό μοντέλο του καταρράκτη με το οποίο οι προδιαγραφές του συστήματος εξετάζονταν προσεκτικά, τεκμηριώνονταν, κωδικοποιούνταν, δοκιμάζονταν, και τελικά παραδιδόταν στον πελάτη το προϊόν, μερικές φορές πολύ καθυστερημένα. Στην συνέχεια του κεφαλαίου αυτού θα συγκριθεί η μεθοδολογία Scrum με το μοντέλο του καταρράκτη, ενώ τέλος θα παρουσιαστούν τα πλεονεκτήματα και τα μειονεκτήματα της μεθοδολογίας Scrum έναντι των παραδοσιακών μεθόδων τόσο με βάση τα αποτελέσματα της έρευνας, που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, όσο και σε θεωρητικό επίπεδο

Η μεθοδολογία Scrum μπορεί να μην είναι δυνατό να εφαρμοστεί σε όλα τα έργα λογισμικού, παρέχει όμως σημαντικά πλεονεκτήματα όταν εφαρμόζεται σε συγκεκριμένα επιχειρησιακά προβλήματα. Από όλες τις ευέλικτες μεθοδολογίες, το Scrum ξεχωρίζει, γιατί εισάγαγε την ιδέα του εμπειρικού ελέγχου των διαδικασιών.

Αυτό σημαίνει ότι η μεθοδολογία Scrum χρησιμοποιεί την πραγματική εξέλιξη του έργου για το σχεδιασμό και τον προγραμματισμό των εκδόσεων του λογισμικού.

Παραδοσιακά, ο κύκλος ζωής λογισμικού απαιτεί την προδιαγραφή των απαιτήσεων, την ανάλυση, το σχεδιασμό και τον έλεγχο με το κάθε ένα από αυτά τα βήματα να ολοκληρώνεται πριν μεταβούμε στο επόμενο (βλέπε μοντέλο του καταρράκτη, Κεφάλαιο 2). Ο κύριος εκπρόσωπος των παραδοσιακών μεθόδων είναι το μοντέλο του καταρράκτη. Αν και το μοντέλο του καταρράκτη είναι χρήσιμο ορισμένες φορές, μειονεκτεί. Ένα από τα μειονεκτήματα του είναι ότι δημιουργείται πολύ νωρίς τεκμηρίωση για το σύστημα, ενώ δεν έχει δημιουργηθεί ακόμα κάτι απτό. Αυτό γίνεται γιατί η υλοποίηση αρχίζει αρκετά μετά την έναρξη του έργου. Ένα άλλο μειονέκτημα είναι ότι οι πελάτες πρέπει να εγγυηθούν ότι οι απαιτήσεις που προδιέγραψαν είναι σωστές σε πολύ αρχικά στάδια του προγραμματισμού του έργου, οπότε τυχόν λάθη στην προδιαγραφή των απαιτήσεων επιφέρουν κινδύνους και καθορίζουν σημαντικά την πορεία του έργου. Ωστόσο, ο ορισμός των απαιτήσεων σε τόσο πρώιμο στάδιο του έργου είναι συχνά πολύ δύσκολος και κατά συνέπεια πολλές απαιτήσεις αλλάζουν στην πορεία. Μελέτες έχουν δείξει ότι στα μεγαλύτερα και σύνθετα έργα το 60% από τις αρχικές απαιτήσεις έχουν αλλάξει στην πορεία του έργου. Άλλες απαιτήσεις υλοποιούνται όπως αρχικά ορίζονται, αλλά στην πραγματικότητα υλοποιείται κάτι που δεν χρειάζεται ο πελάτης και καταναλώνεται πολύτιμος χρόνος.

Το μοντέλο του καταρράκτη είναι γνωστό και σαν «βαρύ» (Heavyweight) μοντέλο εξαιτίας της λεπτομέρειας και του προγραμματισμού που το χαρακτηρίζει. Το Scrum είναι πιο προσαρμοστικό και ευέλικτο στις αλλαγές. Αντί της καταγραφής όλων των απαιτήσεων και προδιαγραφών στην αρχή έναρξης του έργου, οι χρήστες συμμετέχουν ενεργά και οι απαιτήσεις συγκεντρώνονται ή προσαρμόζονται σε τακτά χρονικά διαστήματα. Καθώς στο Scrum εφαρμόζεται σταδιακή επαυξητική ανάπτυξη, δεν υπάρχει η ανάγκη για μια πλήρη ανάλυση και λεπτομερή σχεδιασμό.

Η μέθοδος του καταρράκτη είναι μια σειριακή προσέγγιση ανάπτυξης λογισμικού όπου εργασίες υλοποιούνται με συγκεκριμένη σειρά, πράγμα που σημαίνει ότι μία ομάδα θα πρέπει να περιμένει μια άλλη ομάδα να ολοκληρώσει το έργο της ώστε να μπορεί να ξεκινήσει. Ο διαχωρισμός του έργου σε διαφορετικές φάσεις επιβάλλει στους χρήστες

να αντιμετωπίζουν κάθε φάση ξεχωριστά. Το πρόβλημα είναι ότι οι περισσότερες από αυτές τις φάσεις συνήθως δεν είναι ξεχωριστές αλλά αλληλοεξαρτούνται. Αυτό έχει ως αποτέλεσμα την κατανάλωση περισσότερου χρόνου και πόρων. Το Scrum λειτουργεί επαναληπτικά και πολλές εργασίες μπορούν να επικαλύπτονται. Ως εκ τούτου απαιτείται λιγότερος χρόνος για την ολοκλήρωση των εργασιών.

Με τη μεθοδολογία Scrum υπάρχει πολύ μικρότερο ρίσκο έναντι του μοντέλου του καταρράκτη, γιατί εστιάζεται στην παράδοση πλήρως ελεγμένων, ανεξάρτητων και μικρών λειτουργιών σε κάθε επανάληψη. Οπότε, με τη διάσπαση αυτή, αν προκύψει κάποιο λάθος σε μία λειτουργία, δε θα επηρεάσει τις υπόλοιπες. Το κύριο πλεονέκτημα της Scrum είναι ότι μπορούν να δημιουργηθούν πολλές λειτουργικές εκδόσεις του λογισμικού, η κάθε μια από τις οποίες αποτελεί ένα μικρό μέρος ολόκληρου του συστήματος ενώ στο μοντέλο του καταρράκτη το προϊόν παραδίδεται μόνο στο τέλος του χρονοδιαγράμματος του έργου.

Στον παρακάτω πίνακα υπάρχει μία σύγκριση των δύο μεθοδολογιών για όλες τις λειτουργίες και διαδικασίες τους [Cocco et al., 2011; Loeser, 2006]:

Κριτήριο	**Μοντέλο Καταρράκτη**	**Scrum**
Διαδικασίες		
Προγραμματισμός Έργου (Σκοπός, πρόγραμμα, επικοινωνίες και ανθρώπινοι πόροι)	1. Δεν υπάρχει αντίστοιχη αναφορά στο μοντέλο του καταρράκτη. 2. Δεν υπάρχει αντίστοιχη αναφορά στο μοντέλο του καταρράκτη.	1. Vision: Μία σύντομη αναφορά για την κατεύθυνση του έργου (πραγματοποιείται από τον Product Owner) 2. Roadmap: Μία σύντομη αναφορά για τις υψηλού προτεραιότητας λειτουργίες που πρέπει να υλοποιηθούν (γράφεται από Product

		Owner).
	3. Μόλις ένα έργο έχει οριστικοποιηθεί, ο Project Manager (PM) ηγείται της προδιαγραφής απαιτήσεων και του υπολογισμού της διάρκειας των εργασιών.	3. Release Plan: Προγραμματισμός επαναλήψεων κατά τις οποίες παραδίδονται τμήματα του προϊόντος.
	4. Ο PM δημιουργεί το Project Plan και το αναθεωρεί με την ομάδα διαχείρισης του έργου.	4. Sprint Plan: Προσδιορίζονται όλες τις εργασίες που πρέπει να γίνουν και προβλέπεται ο χρόνος για κάθε εργασία (εμπλέκεται ο Product Owner και η ομάδα ανάπτυξης και ενημερώνεται καθημερινά από την ομάδα στο Daily Scrum meeting).
	5. Ο PM συναντιέται με την ομάδα έργου περιοδικά (σύμφωνα με το πλάνο επικοινωνίας) για την ανανέωση του Project Plan με πραγματικές ώρες εργασίας και αναθεωρεί το έργο. Καταγράφονται κίνδυνοι και προβλήματα.	5. Daily Scrum Meeting: συνάντηση που διαρκεί 15 λεπτά και την συντονίζει ο Scrum Master. Είναι το κύριο μέσο για την εσωτερική επικοινωνία της ομάδας. Ο Scrum Master ρωτάει το κάθε μέλος της ομάδας τι έγινε χθες, τι πρέπει να γίνει σήμερα και τυχόν προβλήματα.

		Ενημερώνεται το Sprint Plan.
	6. Line Management: Ο PM διαπραγματεύεται με τη διοίκηση της εταιρίας σε κάθε στιγμή για τους πόρους και μπορεί να τους αλλάξει.	6. Line Management: Η ομάδα του Scrum δεν αλλάζει κατά τη διάρκεια ενός Sprint.
	7. Το έργο οδηγείται από την ημερομηνία παράδοσης και περιλαμβάνει όσο το δυνατόν περισσότερη τεκμηρίωση.	7. Ο προγραμματισμός του έργου εξάγεται από το Release Plan. Το κάθε Sprint έχει συγκεκριμένη διάρκεια και οι εργασίες με την υψηλότερη προτεραιότητα πρέπει να γίνονται στα αρχικά Sprint.
Παρακολούθηση και έλεγχος των αλλαγών	8. Οι αλλαγές στις απαιτήσεις χειρίζονται μέσω μιας διαδικασίας αλλαγών που επιβλέπει ο PM.	8. Μόνο η ομάδα Scrum μπορεί να αλλάξει τις λειτουργίες και τις εργασίες σε ένα Sprint και μόνο αν συμφωνούν όλα τα μέλη της. Αλλιώς η επιθυμητή αλλαγή προστίθεται στο Product Backlog σαν καινούρια εργασία.
	9. Ο PM είναι υπεύθυνος για την ανάλυση του ρίσκου και συνήθως διατηρεί ένα αντίστοιχο έγγραφο (Risk Document).	9. Η ομάδα διαχειρίζεται την ανάλυση του ρίσκου του έργου πριν ξεκινήσει να το αναπτύσσει.
Χρονοπρογραμματι	10. Ο PM δημιουργεί ένα	10. Τα μέλη της ομάδας

σμός και ανθρώπινοι πόροι	Project Plan με εργασίες, αναθέσεις εργασιών και εκτιμήσεις για το χρόνο διεκπεραίωσής τους. Το Project Plan αναθεωρείται από την ομάδα έργου. Για τις αποκλίσεις από το Project Plan είναι υπεύθυνος ο PM. 11. Ο PM ενημερώνει περιοδικά το Project Plan με τις πραγματικές ώρες και νέες εκτιμήσεις. 12. Ο PM λειτουργεί ως αρχηγός για την ομάδα έργου και βοηθάει την ομάδα έργου να ξεπερνάει τα εμπόδια.	δημιουργούν τις εργασίες και προβλέπουν το χρόνο διεκπεραίωσής τους. 11. Τα μέλη της ομάδας ενημερώνουν τις εκτιμήσεις των εργασιών και προσθέτουν/τροποποιούν/διαγράφουν άλλες. 12. Ο Scrum Master είναι υπεύθυνος για την ομάδα, αλλά τα μέλη της ομάδας λειτουργούν μόνα τους.
Ανέφικτη ημερομηνία ολοκλήρωσής του έργου	13. Αν η ημερομηνία ολοκλήρωσής του έργου είναι ανέφικτη, τότε ο PM διαπραγματεύεται τα εξής: • Το σκοπό του έργου • Το χρονικό ορίζοντα ολοκλήρωσης • Το κόστος • Τους πόρους • Την ποιότητα	13. Αν κάποιο παραδοτέο φαίνεται ότι θα καθυστερήσει, τότε οι εργασίες προστίθενται στο Product Backlog και μετατίθενται στο επόμενο Sprint. Δε μπορεί το Sprint να καθυστερήσει και ο Scrum Master δε μπορεί να διαπραγματευτεί άλλους πόρους ή χρόνο.
Κόστος έργου	14. Ο PM συνήθως διαχειρίζεται το budget του έργου, που περιλαμβάνει κόστη	14. Στην αρχή του κάθε Sprint, ανατίθεται ένα συγκεκριμένο κόστος στο

	για ανθρώπινους πόρους και άλλα έξοδα. Πιθανές υπερβάσεις του κόστους πρέπει να τεκμηριωθούν και να επιβεβαιωθούν από τη διοίκηση.	Sprint. Στη συνέχεια τα μέλη της ομάδας μπορούν να χρησιμοποιήσουν το budget αυτό όπως κρίνουν.
Έλεγχος ποιότητας έργου	15. Από την πλευρά του πελάτη, η ποιότητα ορίζεται ως η έγκαιρη παράδοση αυτών που είχαν συμφωνηθεί με το διαθέσιμο budget. 16. Το προσωπικό που ελέγχει τον κώδικα ελέγχει ολόκληρο τον κώδικα στο τέλος του έργου.	15. Στο Scrum, ποιότητα σημαίνει δημιουργία επαυξημένων προϊόντων μετά από κάθε Sprint που να συμφωνούν με τις λειτουργίες που αναμένονταν. 16. Το προσωπικό που ελέγχει τον κώδικα ελέγχει κάθε Sprint ξεχωριστά.
Έγγραφα		
Εξέταση του κόστους	17. Cost Benefit Analysis: το έγγραφο κατατίθεται από τον PM και εγκρίνεται από τον πελάτη.	17. Δεν υπάρχει αντίστοιχο έγγραφο.
Απαιτήσεις	18. Τα έγγραφα που παράγονται από τις απαιτήσεις είναι τα εξής: • Feasibility Study • Statement of Work • Project Scope • Requirements Document • Functional Design	18. Τα έγγραφα που παράγονται από τις απαιτήσεις είναι τα εξής: • Product Backlog • Release Plan • Sprint Backlog/Plan

	• Detailed Design • Test Plans και Test Cases 19. Όλα τα έγγραφα εγκρίνονται από τον PM.	19. Όλα τα έγγραφα αναθεωρούνται από Product Owner και την ομάδα ανάπτυξης.
Προγραμματισμός επικοινωνίας	20. Ο PM δημιουργεί ένα πρόγραμμα επικοινωνίας (Communication Plan) και καθορίζει τις συναντήσεις του έργου, τα παραδοτέα και ορίζει ποιοι θα συμμετέχουν σε κάθε συνάντηση.	20. Δεν υπάρχει αντίστοιχο Communication Plan στο Scrum αφού η ομάδα αυτό-οργανώνεται.
Διαχείριση προϋπολογισμού	21. Ο PM είναι υπεύθυνος για την επίβλεψη και αναθεώρηση του προϋπολογισμού του έργου.	21. Δεν υπάρχει αντίστοιχη διαδικασία. Εναπόκειται στην ομάδα η διαχείριση του προϋπολογισμού.
Διαχείριση των κινδύνων	22. Ο PM είναι υπεύθυνος για την επίβλεψη των κινδύνων που προκύπτουν και οφείλει να συμβουλεύει την ομάδα ώστε να ξεπερνάει εμπόδια και δυσκολίες, ώστε το προϊόν να παραδοθεί με επιτυχία.	22. Οι κίνδυνοι και τα προβλήματα που προκύπτουν διαχειρίζονται από τον Product Owner και την ομάδα ανάπτυξης κατά το Release Plan. Η ομάδα είναι υπεύθυνη για την επίλυση των προβλημάτων.
Προγραμματισμός του έργου	23. Ο PM δημιουργεί το συγκεκριμένο έγγραφο (Project Plan) στην αρχή του έργου με εκτιμήσεις για τις εργασίες του έργου. Στη συνέχεια, το	23. Ο προγραμματισμός κάθε επανάληψης γίνεται κατά το Sprint Planning Meeting από τον Product Owner και την ομάδα

	χρονοδιάγραμμα αυτό ενημερώνεται τακτικά με τις πραγματικές ώρες και τις αναθεωρημένες εκτιμήσεις/καθήκοντα.	ανάπτυξης και ενημερώνεται καθημερινά.
Αναφορές προόδου	24. Ο PM δημιουργεί σε τακτά χρονικά διαστήματα αναφορές για την πρόοδο του έργου.	24. Δεν υπάρχει αντίστοιχο έγγραφο. Το Scrum επικεντρώνεται στην προσωπική επικοινωνία και έτσι η ομάδα είναι συνεχώς ενήμερη για την πρόοδο της ανάπτυξης.
Συναντήσεις ομάδας		
Συναντήσεις/Συντονισμός ομάδας	25. Σύμφωνα με το Communication Plan, ο PM συνήθως προγραμματίζει συναντήσεις για την επίβλεψη του έργου. Οι συναντήσεις αυτές μπορεί να διαρκέσουν μία ώρα ή και περισσότερο.	25. Στο Scrum υπάρχει μια καθημερινή 15-λεπτη συνάντηση για τη συζήτηση της πορείας του έργου. Στο τέλος του κάθε Sprint, ένα demo πραγματοποιείται κατά το οποίο παρουσιάζεται στον πελάτη του προϊόντος ότι καινούριο δημιουργήθηκε κατά τη διάρκεια του Sprint.

Αναθεωρήσεις μετά την υλοποίηση του έργου	26. Ο PM πραγματοποιεί μια αναθεώρηση μετά την ολοκλήρωση του έργου, προκειμένου να καταγράψει ότι πήγε καλά και αυτά που θα μπορούσαν να βελτιωθούν.	26. Κατά το Sprint Retrospective, το οποίο πραγματοποιείται μετά το τέλος κάθε Sprint, τα μέλη της ομάδας αναθεωρούν το παρελθόν, διατυπώνουν συστάσεις για μελλοντικές βελτιώσεις ή αλλαγές.

Πίνακας 6.2 Σύγκριση Scrum με το μοντέλο του καταρράκτη

Από τα παραπάνω, παρατηρούμε πώς το Scrum υποστηρίζει την αυτό-διαχείριση των ομάδων στις οποίες ο Scrum Master ενεργεί κυρίως ως σύμβουλος, βοηθώντας την ομάδα να ξεπερνάει εμπόδια που ενδέχεται να έχουν αρνητικές επιπτώσεις στην παραγωγικότητα της ομάδας. Οι ομάδες που αυτό-διοικούνται χρειάζονται χρόνο για να εξελιχθούν. Από τη στιγμή που η τεκμηρίωση είναι σχετικά λιγοστή, θα πρέπει η ομάδα να προετοιμαστεί για να μπορεί να επικοινωνεί αποτελεσματικά.

Ο παραδοσιακός Project Manager μπορεί να παραλληλιστεί με τον καπετάνιο ενός πλοίου, ο οποίος είναι υπεύθυνος για την πρόβλεψη και την αντιμετώπιση των δυσκολιών και τελικά την μεταφορά με ασφάλεια του φορτίου και των επιβατών σύμφωνα με το χρονοδιάγραμμα. Σε αντίθεση, ο Scrum Master ενεργεί κυρίως ως διαμεσολαβητής για την ομάδα και είναι υπεύθυνος να βοηθήσει την ομάδα να ολοκληρώσει με επιτυχία το κάθε Sprint. Τα καθήκοντα ενός Scrum Master αποτελούν ένα υποσύνολο του συνόλου των δεξιοτήτων που απαιτούνται για να είναι επιτυχής ένας Project Manager. Η εμπειρία και η γνώση σχετικά με τον καθορισμό των απαιτήσεων, τη διαχείριση του χρόνου, την εκτίμηση, τη διαπραγμάτευση, την εποπτεία του προϋπολογισμού, και πρόβλεψη των κινδύνων, όλα αναμένονται από τον Project Manager.

6.3.1 Πλεονεκτήματα μεθόδου Scrum έναντι των παραδοσιακών μεθόδων

Το Scrum έχει τα δικά του πλεονεκτήματα και μειονεκτήματά, όπως και οποιαδήποτε άλλη μεθοδολογία ή διαδικασία παραγωγής λογισμικού. Στη συγκεκριμένη υποενότητα παρουσιάζονται τα πλεονεκτήματα της μεθοδολογίας Scrum και σχολιάζονται σύμφωνα με τα ευρήματα της έρευνας.

Ευκολία αλλαγών έναντι προσκόλληση στο πλάνο: Η παραγωγή λογισμικού δεν βασίζεται πάντοτε στην τελειότητα. Αλλαγές προκύπτουν σε όλη τη διάρκεια του κύκλου ζωής του λογισμικού. Επίσης, δεν μπορούμε να αγνοήσουμε ότι ο πελάτης είναι (συνήθως) μόνο ένας χρήστης και απλώς θέλει το λογισμικό να πληροί τις ανάγκες του, χωρίς ενδιαφέρον για το πώς αυτό συμβεί. Το Scrum είναι πολύ ευέλικτη μεθοδολογία και μπορεί να αντιμετωπίσει με ευκολία τις αλλαγές που θα προκύψουν στις απαιτήσεις του πελάτη. Αντίθετα, στις παραδοσιακές μεθοδολογίες πρέπει να ακολουθείται το πλάνο που έχει τεθεί εξαρχής και έτσι οι αλλαγές στις απαιτήσεις μπορεί αν αποδειχθούν καταστροφικές. Σύμφωνα με τα αποτελέσματα της έρευνας, το 47,1% των ερωτώμενων πιστεύουν ότι αυτό είναι και το σημαντικότερο πλεονέκτημα του Scrum. Το Scrum παρέχει μια συνεχή, ταχεία αναδιοργάνωση, επιτρέποντας ξαφνικές μαζικές αλλαγές στο έργο, χωρίς μεγάλες ζημιές από άποψη χρόνου και κόστους. Οι αλλαγές είναι εφικτές, αφού το έργο διαχωρίζεται σε μια σειρά από μικρά προβλήματα, ή «κομμάτια» ενός μεγαλύτερου προβλήματος, και έτσι είναι πολύ πιο εύκολο να διαχειριστούν οι αλλαγές από το αν προσπαθούσαμε να προσεγγίσουμε το έργο σε ολόκληρη τη μορφή του. Αν και η προθεσμία και ο προϋπολογισμός είναι σταθερές μεταβλητές, οι απαιτήσεις του έργου δεν είναι. Στην πραγματικότητα, οι πελάτες και οι συμμετέχοντες θα προτείνουν αλλαγές στην πορεία. Η συμμετοχή του ιδιοκτήτη προϊόντος κατά τη διαδικασία διαχείρισης του έργου διευκολύνει αυτές τις αλλαγές.

Ανθρωποκεντρικός προσανατολισμός έναντι προσανατολισμού στις διαδικασίες: Κανονικά, σε σύνθετα έργα, η διασύνδεση μεταξύ των διαφόρων τμημάτων του λογισμικού είναι υπερβολικά περίπλοκη, λόγω του ότι τα εμπλεκόμενα μέλη δουλεύουν ανεξάρτητα. Στο Scrum, οι ομάδες συνεργάζονται και επικοινωνούν καθιστώντας την ολοκλήρωση των εργασιών πολύ πιο εύκολη, αφού ο καθένας ξέρει τι κάνει ο κάθε άλλος. Η Scrum είναι μία ανθρωποκεντρική μεθοδολογία έναντι του προσανατολισμού

στις διαδικασίες που παρουσιάζεται στο μοντέλο του καταρράκτη. Το 22,4% των ερωτώμενων πιστεύουν ότι αυτό είναι ένα σημαντικό πλεονέκτημα του Scrum.

Συγγραφή κώδικα έναντι τεκμηρίωσης: Το 21,3% των ερωτώμενων κατέδειξαν την συγγραφή κώδικα έναντι τεκμηρίωσης ως το σημαντικότερο πλεονέκτημα της μεθοδολογίας Scrum. Η μέθοδος Scrum βασίζεται σε επαναλήψεις αποτέλεσμα των οποίων είναι ένα μικρό τμήμα του λογισμικού κάθε φορά. Η τεκμηρίωση κάθε επανάληψης είναι ο ίδιος ο κώδικας που παράγεται. Αντίθετα, στις παραδοσιακές μεθοδολογίες πρέπει πρώτα να υπάρχει πλήρη τεκμηρίωση (έγγραφο απαιτήσεων κτλ.) και στη συνέχεια συγγράφεται ο κώδικας που στην ουσία υλοποιεί το σύστημα.

Διαπροσωπικές σχέσεις με τον πελάτη έναντι τυπικών διαδικασιών:. Ένα άλλο πλεονέκτημα είναι τα μικρά χρονικά διαστήματα μεταξύ των παρουσιάσεων του έργου στον πελάτη και η γνώση για την εξέλιξη του έργου καθώς και η συχνή ανατροφοδότηση σχετικά με τις λειτουργίες του προϊόντος. Αναπτύσσεται μια σχέση εμπιστοσύνης με τον πελάτη του προϊόντος η οποία βελτιώνει σημαντικά όλες τις διαδικασίες ανάπτυξης και δίνει ιδιαίτερα κίνητρα στην ομάδα ώστε να προσπαθήσει για το καλύτερο δυνατό αποτέλεσμα. Λόγω της συνεχής ολοκλήρωσης των Sprint, συνήθως κάθε 2-4 εβδομάδες, ο πελάτης μπορεί να βλέπει νέες εκδόσεις του λογισμικού σε τακτά χρονικά διαστήματα

Μεγαλύτερη παραγωγικότητα της ομάδας ανάπτυξης: Σύμφωνα με τα αποτελέσματα της έρευνας, το 87.5% των ερωτώμενων πιστεύουν ότι η παραγωγικότητα βελτιώθηκε εφαρμόζοντας τη μεθοδολογία Scrum συγκριτικά με τις παραδοσιακές μεθόδους ανάπτυξης λογισμικού. Το αποτέλεσμα αυτό είναι αναμενόμενο, καθώς οι πρακτικές και οι διαδικασίες που εφαρμόζονται στο Scrum επιτρέπουν, όπως αναφέραμε, την συχνή παραγωγή πλήρως λειτουργικών τμημάτων λογισμικού.

Πιο ποιοτικά έργα λογισμικού: Το Scrum παρέχει το μηχανισμό για να βοηθήσει ώστε η ανάπτυξη να επικεντρωθεί στα υψηλότερης προτεραιότητας και συνεπώς επιχειρηματικής αξίας χαρακτηριστικά σε ένα σύστημα. Επιτρέπει έτσι να αυξηθεί η παραγωγικότητα και μακροπρόθεσμα η ποιότητα του παραγόμενου προϊόντος. Οι

παράγοντες αυτοί οδηγούν επίσης στην μείωση του συνολικού κόστους υλοποίησης (ανάπτυξης και συντήρησης). Οι επαναλήψεις ή Sprints, επιτρέπουν την παραγωγή ποιοτικών τμημάτων του προϊόντος καθώς το έργο είναι σε εξέλιξη. Επιπλέον, εξετάζοντας κάθε Sprint, κατά την διάρκεια του Sprint Retrospective Meeting, πριν προχωρήσουμε στο επόμενο σημαίνει ότι οι διαδικασίες που διεξάγονται σε όλη τη διάρκεια της ανάπτυξης βελτιώνονται συνεχώς μέσα από εμπειρικές διαδικασίες, με αποτέλεσμα και την βελτίωση του παραγόμενου αποτελέσματος. Το γεγονός αυτό αποδεικνύεται από την έρευνα που πραγματοποιήθηκε: η συντριπτική πλειοψηφία των συμμετεχόντων (ποσοστό 84,3%) παρατήρησαν αυξημένη ποιότητα στα συστήματα λογισμικού που αναπτύχθηκαν με την μεθοδολογία Scrum.

Μείωση κόστους του έργου: Σύμφωνα με τα αποτελέσματα της έρευνας, μόλις το 48,5% των ερωτώμενων πιστεύουν ότι στα έργα λογισμικού με τη μεθοδολογία Scrum έχει μειωθεί το κόστος. Αντίστοιχα, 26% πιστεύουν ότι αυξήθηκε το κόστος, το 25,4% πιστεύουν ότι δεν επηρεάστηκε το κόστος των έργων. Άρα, σύμφωνα με τα αποτελέσματα της έρευνας το πλεονέκτημα αυτό που αναφέρεται στη βιβλιογραφία δεν επιβεβαιώνεται πλήρως.

Μεγαλύτερη ικανοποίηση πελατών: Η μεγιστοποίηση της ικανοποίησης των πελατών σημαίνει υλοποίηση και παράδοση των απαιτήσεων τους με τρόπο που να συνάδει με τις προτεραιότητες των πελατών. Παραδίδονται πρώτα οι λειτουργίες με υψηλότερη προτεραιότητα. Ο μόνος τρόπος για να παραδίδεται το προϊόν είναι μέσω των επαναληπτικών επαυξητικών κύκλων ανάπτυξης, γι’ αυτό ακριβώς τον λόγο η διάρκεια μιας επανάληψης κυμαίνεται από 2-4 εβδομάδες. Με τον τρόπο αυτό η παραγωγικότητα και η ικανοποίηση των πελατών είναι πάντα δεδομένη. Επιπλέον υπάρχει διαφάνεια (Transparency) στην όλη διαδικασία ανάπτυξης. Αυτό σημαίνει πως ο πελάτης, με την ενεργή συμμετοχή του, είναι συνεχώς ενήμερος για την κατάσταση της προόδου του συστήματος. Σύμφωνα με τα αποτελέσματα της έρευνας, το 85,4% των ερωτώμενων πιστεύουν ότι στα έργα λογισμικού με τη μεθοδολογία Scrum έχει αυξηθεί η ικανοποίηση των πελατών σε σχέση με έργα στα οποία εφαρμόζονται παραδοσιακές προσεγγίσεις, γεγονός που αποδείκνυε πως το Scrum επιτυγχάνει στην πράξη.

Τα σημαντικότερα από τα πιο πάνω πλεονεκτήματα συγκεντρώνονται και παρουσιάζονται στον Πίνακα 6.3.

	Αυξήθηκε	**Καμία αλλαγή**	**Μειώθηκε**
Παραγωγικότητα	**87.5%**	6,8%	5.5%
Ποιότητα	**84,3%**	13,1%	2.5%
Κόστος	26%	25.4%	**48.5%**
Ικανοποίηση πελάτη	**85.4%**	9,5%	5.15%

Πίνακας 6.3 Συγκεντρωτικός πίνακας με συνολικά οφέλη της μεθοδολογίας Scrum έναντι των παραδοσιακών προσεγγίσεων

Μπορούμε να συμπεράνουμε λοιπόν ότι η μεθοδολογία Scrum παρέχει ουσιαστικά οφέλη στην διαδικασία ανάπτυξης λογισμικού τόσο για την ομάδα ανάπτυξης και τους πελάτες όσο και για την διαχείριση. Το Scrum έχει σχεδιαστεί για να βελτιστοποιήσει την ικανοποίηση τόσο της ομάδας όσο και του πελάτη, την παραγωγικότητα, την ποιότητα των προϊόντων, την ανταπόκριση στις μεταβαλλόμενες απαιτήσεις του πελάτη, και την διαφάνεια για τους συμμετέχοντες. Οι βασικές πρακτικές που επιτρέπουν αυτά τα οφέλη περιλαμβάνουν την έμφαση στις εργασίες με μεγαλύτερη προτεραιότητα, τις σύντομες επαναλήψεις, την συχνή παράδοση λειτουργικών τμημάτων του προϊόντος και τον εμπειρικό έλεγχο των διαδικασιών και πρακτικών.

6.3.2 Μειονεκτήματα μεθόδου Scrum έναντι των παραδοσιακών μεθόδων

Δυστυχώς, όπως σε οποιαδήποτε άλλη μέθοδο, το Scrum παρουσιάζει κάποια προβλήματα που πολλές φορές είναι δύσκολο και ίσως αδύνατο να ξεπεραστούν, λόγω της φύσεώς τους.

Έλλειψη συνεργασίας με τον πελάτη: Η συμμετοχή των πελατών στα παραδοσιακά έργα ανάπτυξης λογισμικού είναι συνήθως περιορισμένη. Γίνεται μόνο κατά την καταγραφή των απαιτήσεων στην αρχή και την ανατροφοδότηση προς το τέλος, με

περιορισμένες συναντήσεις μεταξύ του πελάτη και την ομάδα ανάπτυξης. Σε αντίθεση, η συνεργασία του πελάτη είναι ένας σημαντικός παράγοντας επιτυχίας για το Scrum. Στη μέθοδο Scrum πελάτης εμπλέκεται ενεργά μέσα σε όλη τη διαδικασία ανάπτυξης με τη συμμετοχή του στον καθορισμό των user stories, συζητώντας τα χαρακτηριστικά του προϊόντος, την ιεράρχηση του Product Backlog, και την παροχή ανατροφοδότησης στην ομάδα ανάπτυξης σε τακτική βάση. Στο Scrum εκ φύσεως η συμμετοχή του πελάτη είναι απαραίτητη. Ο πελάτης είναι το μέσο για να διασφαλιστεί η πρόοδος του συστήματος και η επικύρωση των παραγόμενων τμημάτων σε κάθε επανάληψη. Η συνεργασία των πελατών είναι ζωτικής σημασίας γενικότερα για τις ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού. Μέσα από μια μελέτη στη Νέας Ζηλανδία και της Ινδία [Hoda et al., 2010], διαπιστώθηκε πως σε ομάδες που εφάρμοζαν ευέλικτες προσεγγίσεις, η έλλειψη συμμετοχής του πελάτη είχε ως αποτέλεσμα προβλήματα στην συλλογή και την αποσαφήνιση των απαιτήσεων, μείωση της παραγωγικότητας και της ποιότητας του προϊόντος. Στην πράξη, φαίνεται πως το πρόβλημα της έλλειψης συνεργασίας εκ μέρους του πελάτη είναι πράγματι σοβαρό. Αυτό προκύπτει από το γεγονός πως η πλειοψηφία των ανταποκρινόμενων στην έρευνα μας, ποσοστό 31,4%, απάντησε πώς το σημαντικότερο πρόβλημα στην εφαρμογή της μεθοδολογίας Scrum είναι η έλλειψη συνεργασίας με τον πελάτη.

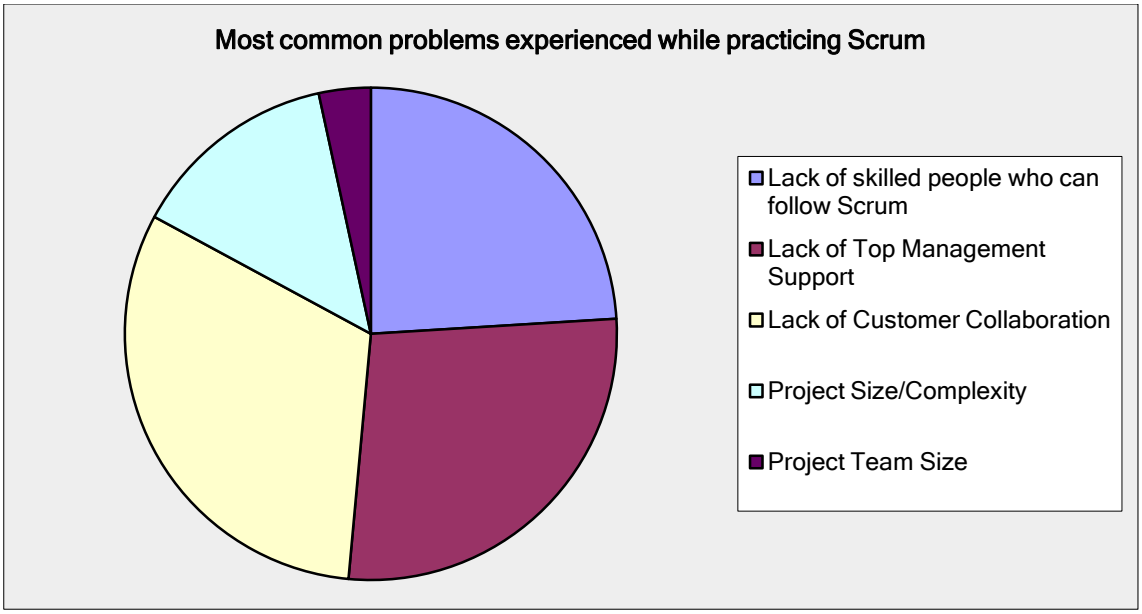
Έλλειψη βοήθειας από τα ανώτερα στελέχη: Η διαχείριση πρέπει να είναι διαρκώς πρόθυμη να προβεί σε τροποποιήσεις, προκειμένου να βοηθά την ομάδα ανάπτυξης να ξεπερνάει εμπόδια και δυσκολίες. Σύμφωνα με τα αποτελέσματα της έρευνας, το 27,4% των ερωτώμενων πιστεύουν ότι δεν παρέχεται βοήθεια από ανώτερα στελέχη των οργανισμών (Lack of top management support). Αν και στο Scrum η ομάδα αυτό-διοικήται και μπορεί να πάρει αποφάσεις, σε γενικές γραμμές στερείται άμεσης διαχείρισης, καθώς κάποιες αποφάσεις είναι ζωτικής σημασίας. Με τη μεθοδολογία Scrum ουσιαστικά η ομάδα κατευθύνεται μόνη της και πάντα με τη βοήθεια του Scrum Master. Ωστόσο, τα ανώτερα στελέχη της επιχείρησης δεν υποστηρίζουν πολύ την ομάδα έργου λόγω έλλειψης εμπειρίας. Η έλλειψη σαφούς, τελικής απόφασης σχετικά με θέματα που μπορεί να καθορίζουν σημαντικά την πρόοδο ενός έργου, μπορεί να οδηγήσει στην μειωμένη παραγωγικότητα και την αποτυχία.

Έλλειψη ικανών ατόμων που μπορούν να ακολουθήσουν τη μεθοδολογία Scrum: Η μεθοδολογία αυτή χρειάζεται έμπειρα μέλη να απαρτίζουν την ομάδα Scrum. Η ομάδα Scrum αποτελείται από εξειδικευμένα και αφοσιωμένα άτομα που εργάζονται για ένα έργο ανάπτυξης λογισμικού. Η ομάδα αυτό-διοικήται και τα μέλη της έχουν εμπειρία στην παροχή βασικών λειτουργιών μέσα σε σύντομο χρονικό διάστημα. Το 24% των ερωτώμενων πιστεύουν ότι η έλλειψη ικανών ατόμων που να μπορούν αν ακολουθήσουν τη μεθοδολογία Scrum είναι το τρίτο σε σειρά σημαντικότερο μειονέκτημα της μεθοδολογίας Scrum. Για τον λόγο αυτό πολλοί οργανισμοί (π.χ Scrum Alliance) παρέχουν ειδικά πρόγραμμα εκπαίδευσης σχετικά με τις πρακτικές και τις διαδικασίες του Scrum.

Μη υποστήριξη για πολύ μεγάλα ή πολύπλοκα έργα: Όπως αναλύθηκε στο προηγούμενο κεφάλαιο, ένας από τους περιορισμούς των ευέλικτων μεθόδων είναι το μέγεθος του έργου. Σύμφωνα με τα αποτελέσματα της έρευνας, το 13.7% των ερωτώμενων πιστεύουν ότι το σημαντικότερο μειονέκτημα της Scrum είναι ότι δε μπορεί να εφαρμοστεί αποδοτικά σε πολύ μεγάλα ή πολύπλοκα έργα. Αυτό έγκειται στο γεγονός ότι η μεθοδολογία Scrum βασίζεται στην επικοινωνία και την διαφάνεια. Μεγάλα έργα συνεπάγονται πολλούς συμμετέχοντες. Καθώς αυξάνεται ο αριθμός των μελών μιας ομάδας τόσο μειώνεται η δυνατότητα για επαρκή επικοινωνία και συντονισμό ανάμεσα στα μέλη της ομάδας.

Μέγεθος της ομάδας: Όπως εξηγήσαμε προηγουμένως, όσο μεγαλύτερη είναι η ομάδα τόσο επηρεάζεται η επικοινωνία σε ένα έργο και η αποτελεσματικότητα των μελών της ομάδας. Το μέγεθος της ομάδας υποδείχθηκε από το 3,4% των ερωτώμενων ως το σημαντικότερο μειονέκτημα του Scrum. Στην βιομηχανία λογισμικού υπάρχει ένας άγραφος κανόνας κατά τον οποίο το μέγεθος μιας ομάδας Scrum θα πρέπει να κυμαίνεται μεταξύ 7±2 ατόμων. Όταν το μέγεθος της ομάδας είναι μικρό, τότε ο κάθε προγραμματιστής γνωρίζει σχεδόν κάθε τμήμα του συστήματος. Ο κώδικας ή ο σχεδιασμός /αρχιτεκτονική του συστήματος είναι γνωστά σε όλους τους προγραμματιστές. Αυτό οδηγεί σε καλύτερες λύσεις και ιδέες που δημιουργούνται από τους προγραμματιστές και συνεπώς σε πιο επαρκή παραγωγικότητα.

Στο Σχήμα 6.2 συνοψίζονται τα μειονεκτήματα που παρουσιάστηκαν πιο πάνω, σύμφωνα με τα ποσοστά κατά τα οποία οι συμμετέχοντες στην έρευνα υπέδειξαν τα προβλήματα αυτά.



Σχήμα 6.2 Μειονεκτήματα της μεθοδολογίας Scrum

Κεφάλαιο 7

Συμπεράσματα

7.1 Σύνοψη και γενικά συμπεράσματα

Τη δεκαετία του ΄60 καθώς και αρχές της δεκαετίας του ΄70 πολλά έργα λογισμικού αποτύγχαναν μαζικά. Το γεγονός αυτό έφερε στην επιφάνεια τα προβλήματα της διοίκησης έργων Πληροφορικής και αναπτύχθηκε μία πληθώρα μοντέλων κύκλων ζωής λογισμικού τα οποία θεμελίωσαν τους τρόπους για να περιγράφεται, να κατασκευάζεται και να συντηρείται λογισμικό καλής ποιότητας, με στόχο τη μεγαλύτερη δυνατή ποιότητα και παραγωγικότητα και τα ελάχιστο δυνατό κόστος.

Από τότε που ξεκίνησε η ανάπτυξη λογισμικού, πολλές διαφορετικές μέθοδοι και μοντέλα διαδικασιών έχουν αναπτυχθεί και χρησιμοποιηθεί για την παραγωγή ποιοτικού λογισμικού. Οι πιο πολλές από αυτές που ονομάζονται παραδοσιακές μέθοδοι και είναι προσανατολισμένες στη βαριά τεκμηρίωση και στις αυστηρά προκαθορισμένες διαδικασίες [Boehm, 2002]. Για δεκαετίες, τα έργα ανάπτυξης λογισμικού ακολουθούσαν το κλασικό μοντέλο του καταρράκτη με το οποίο οι προδιαγραφές του συστήματος εξετάζονταν προσεκτικά, τεκμηριώνονταν, κωδικοποιούνταν, δοκιμάζονταν, και τελικά παραδιδόταν στον πελάτη το προϊόν, μερικές φορές πολύ καθυστερημένα. Οι μέθοδοι αυτοί, που χρησιμοποιούν συνήθως το μοντέλο του καταρράκτη για την ανάπτυξη του λογισμικού, έχουν κατηγορηθεί ως ανελαστικές, μη παραγωγικές, και γραφειοκρατικές. Θεωρούνται μη παραγωγικές διότι οι φάσεις ανάπτυξης του λογισμικού είναι πολλές και μεγάλης διάρκειας επιβραδύνοντας τον συνολικό χρόνο ανάπτυξης. Ωστόσο, το κυριότερο από τα προβλήματα αυτών των μεθόδων είναι ότι προσχεδιάζουν το μεγαλύτερο κομμάτι της

διαδικασίας ανάπτυξης του λογισμικού με μεγάλη λεπτομέρεια και σε βάθος χρόνου, προσπαθώντας να μαντέψουν και να προλάβουν κάθε πρόβλημα που μπορεί να προκύψει. Αυτή είναι μία σημαντική αδυναμία η οποία τις καθιστά ανελαστικές και ανίκανες να αντιδράσουν στις αλλαγές που προκύπτουν, όπως για παράδειγμα στις αλλαγές των απαιτήσεων των πελατών που είναι ένα συνηθισμένο γεγονός.

Με την αλλαγή της χιλιετίας εξελίχθηκε μία νέα ομάδα μεθόδων η οποία με την ιδρυτική της διακήρυξη «Ευέλικτο Μανιφέστο» (Agile Manifesto), και με την ονομασία ευέλικτες μέθοδοι (Agile methods) υπόσχεται προσαρμοστικότητα και ανταπόκριση στις αλλαγές, παραγωγικότερες πρακτικές και λιγότερη γραφειοκρατία. Ουσιαστικά, ο όρος ευέλικτη ανάπτυξη εφευρέθηκε το Φεβρουάριο του 2001, όταν 17 άτομα μεταξύ των οποίων εμπειρογνώμονες της ανάπτυξης λογισμικού και σύμβουλοι επιχειρήσεων παράγωγης λογισμικού, μαζεύτηκαν για να συζητήσουν το μέλλον της ανάπτυξης λογισμικού. Αποτέλεσμα αυτών των συζητήσεων ήταν η διατύπωση του ευέλικτου μανιφέστο το όποιο παρέχει μια εννοιολογική «ομπρέλα» για έναν αριθμό μεθοδολογιών. Το όνομα ευέλικτες αναφέρεται κυρίως στην συνολική ικανότητα να προσαρμόζουν κατάλληλα τη διαδικασία ανάπτυξης, όταν προκύπτουν αλλαγές στην πορεία του έργου.

Το μανιφέστο των ευέλικτων πρακτικών τονίζει την σημασία των (α) άτομα και αλληλεπιδράσεις αντί διαδικασίες και εργαλεία, (β) καθαρός κώδικας αντί γραπτής τεκμηρίωσης, (γ) συνεργασία με τον πελάτη αντί αυστηρών συμβολαίων και (δ) ανταπόκριση σε αλλαγές αντί ακολουθούμενου σχεδίου. Οι ευέλικτες μεθοδολογίες ανάπτυξης, βασίζονται στις αρχές του Agile Manifesto και προσανατολίζονται προς την επίτευξη των στόχων του.

Οι μεθοδολογίες αυτές αναδύθηκαν μέσα από τον πειραματισμό των εταιριών ώστε να καλύψουν κάποιες εταιρικές ανάγκες όταν αναζητούσαν εναλλακτικές λύσεις στις παραδοσιακές μεθοδολογίες ανάπτυξης λογισμικού, τις οποίες έβρισκαν πάρα πολύ δυσκίνητες, γραφειοκρατικές, και άκαμπτες. Τα προβλήματα σχετικά με την ασάφεια των απαιτήσεων καθώς επίσης και οι γρήγοροι ρυθμοί αλλαγών των συνεχώς εξελισσόμενων απαιτήσεων αλλά και του επιχειρηματικού περιβάλλοντος, παρείχαν μια ουσιαστική ανάγκη για την δημιουργία νέων μοντέλων ανάπτυξης λογισμικού. Οι

ευέλικτες μέθοδοι ενσωμάτωσαν μια ευρεία συλλογή από καλές και δοκιμασμένες αξίες και πρακτικές που βοηθούν στην ανάπτυξη ποιοτικού λογισμικού.

Ανάμεσα στις ευέλικτες μεθοδολογίες συγκαταλέγεται και το Scrum, μια επαναληπτική και αυξητική μεθοδολογία ανάπτυξης έργων. Χρησιμοποιείται σαν μια μεθοδολογία ανάπτυξης λογισμικού, όσον αφορά την οργάνωση και λειτουργία των ομάδων ανάπτυξης αλλά και στον σχεδιασμό και διαχείριση των έργων λογισμικού. Στόχος της παρούσας διπλωματικής ήταν η μελέτη της μεθοδολογίας Scrum στα πλαίσια χρήσης της από διάφορες εταιρίες παραγωγής λογισμικού. Για τον σκοπό αυτό έχει οργανωθεί και πραγματοποιηθεί, στα πλαίσια της παρούσας διπλωματική εργασίας, μια μελέτη με σκοπό τη διερεύνηση της έκτασης και της χρήσης των ευέλικτων μεθόδων και ιδιαίτερα της μεθοδολογίας Scrum στην παγκόσμια βιομηχανία λογισμικού.

Τα κύρια ευρήματα της έρευνας ήταν τα πλεονεκτήματα και αντίστοιχα τα μειονεκτήματα των ευέλικτων μεθόδων, συμπεριλαμβανομένου της μεθοδολογίας Scrum. Τα αποτελέσματα δείχνουν ότι η μέθοδος Scrum συγκεντρώνει τις περισσότερες προτιμήσεις καθώς επίσης αποδεικνύεται αποτελεσματική για μικρές ομάδες. Η επιτυχία της μεθόδου οφείλεται στον τρόπο με τον οποίο προσεγγίζει και διαχειρίζεται τη διαδικασία ανάπτυξης του λογισμικού, προσδίδοντάς της μια ευελιξία στην οργάνωση, αυξάνοντας με αυτόν τον τρόπο την παραγωγικότητα της ομάδας αλλά και τη ικανότητα προσαρμογής της σύμφωνα με τις εκάστοτε ανάγκες που προκύπτουν. Κατά την εφαρμογή της μεθόδου δίνεται μεγάλη έμφαση στον τρόπο οργάνωσης της ομάδας, ώστε να μπορέσει να επιτύχει το μέγιστο βαθμό αποτελεσματικότητας.

Όπως προέκυψε από την έρευνα, μπορούμε να καταλήξουμε στο συμπέρασμα ότι οι ευέλικτες μέθοδοι παρέχουν σαφείς βελτιώσεις στην διαδικασία ανάπτυξης λογισμικού. Η ικανότητα να ανταποκρίνονται στις ανάγκες των πελατών, η αυξημένη παραγωγικότητα και η παράδοση ποιοτικών προϊόντων λογισμικού είναι μερικά από τα σημαντικά οφέλη της ευέλικτης ανάπτυξης. Επιπλέον αποδεικνύεται η ακαταλληλότητά της χρήση τέτοιων πρακτικών για έργα που χαρακτηρίζονται από κατανεμημένα περιβάλλοντα. Επίσης, οι μεγάλες ομάδες ανάπτυξης αναγνωρίζονται ως ένα ουσιαστικής σημασίας πρόβλημα. Η υιοθέτηση της ευέλικτης ανάπτυξης αποκάλυψε επίσης περαιτέρω προκλήσεις, όπως η έλλειψη λεπτομερούς αξιολόγησης του κόστους,

η ανάγκη για τακτή συμμετοχή του πελάτη, η υποστήριξη της διαχείρισης και η ανάγκη για εκτεταμένη εκπαίδευση και κατάρτιση για χρήση των μεθόδων αυτών.

Συμπερασματικά, είναι μερικές πλευρές της ανάπτυξης λογισμικού που μπορούν να εκμεταλλευτούν καλύτερα οι παραδοσιακές μεθοδολογίες και άλλες πλευρές που εκμεταλλεύονται καλύτερα οι ευέλικτες μεθοδολογίες, όπως η μέθοδος Scrum. Στην ερώτηση της καταλληλότερης μεθοδολογίας για την ανάπτυξη έργων λογισμικών, η απάντηση δε μπορεί να είναι μόνο μία, γιατί το κάθε έργο λογισμικού είναι μοναδικό.

7.2 Μελλοντική εργασία

Όπως ανέφερα νωρίτερα, η ανάγκη των επιχειρήσεων να ανταποκρίνονται γρήγορα στο περιβάλλον με ένα καινοτόμο, οικονομικά αποδοτικό και αποτελεσματικό τρόπο, καθιστά επιτακτική την χρήση μεθόδων για την ευέλικτη ανάπτυξη λογισμικού. Οι ευέλικτες μέθοδοι απαιτούν αλλαγή στη νοοτροπία και τη σκέψη των συμμετεχόντων σε ένα ευέλικτο έργο. Αν και αυτές οι μέθοδοι, όπως παρουσιάστηκε στην παρούσα εργασία, έχουν δείξει ενθαρρυντικά αποτελέσματα στη βιομηχανία λογισμικού και η υιοθέτηση τους γίνεται με γρήγορους ρυθμούς, υπάρχουν ορισμένες προκλήσεις οι οποίες στερούνται ερευνητικής μελέτης.

Ένα ενδιαφέρον εύρημα που προέκυψε από την έρευνα, είναι η ανάμειξη (hybridisation), μιας παραδοσιακής μεθοδολογίας με μια ευέλικτη ή πολλές ευέλικτες πρακτικές ή/και διαδικασίες. Τα μοντέλα αυτά που υιοθετούν πρακτικές και διαδικασίες από διάφορες προσεγγίσεις, ονομάζονται υβριδικά (Hybrid). Αρκετοί είναι αυτοί που έχουν αρχίσει να χρησιμοποιούν υβριδικά μοντέλα, όπως για παράδειγμα το Scrum μαζί με την Kanban (μέθοδος για την ανάπτυξη προϊόντων λογισμικού με έμφαση στην έγκαιρη παράδοση), που ήταν η κυρίαρχη απάντηση στην επιλογή «Άλλο» στην ερώτηση «Ποια ευέλικτη μεθοδολογία χρησιμοποιείται περισσότερο;» στην έρευνα που διεξήγαμε. Το μοντέλο αυτό ονομάζεται Scrum-Ban και φαίνεται πως συγκεντρώνει την προσοχή αρκετών αφού πολλοί συμμετέχοντες ανέφεραν το μοντέλο αυτό μέσα από την ανοικτή ερώτηση (Ερώτηση 36) για παράθεση σχολίων.

Αυτό οφείλεται στο ότι στην πραγματικότητα δεν είναι εφικτό ποτέ να εφαρμοστεί αυστηρά μια συγκεκριμένη μεθοδολογία γιατί σε ένα έργο λογισμικού ίσως χρειαστεί να προσαρμοστούν διαφορετικές προσεγγίσεις ώστε να αλληλεπικαλύπτονται. Ακόμα κατά την ανάπτυξη ενός έργου, καθώς η ανάπτυξη διέρχεται μέσα από διάφορες φάσεις, ίσως χρειαστεί να προσαρμοστεί ανάλογα η μεθοδολογία σύμφωνα με το τι πρέπει να γίνει ώστε να επιτευχθεί το δυνατό καλύτερο αποτέλεσμα. Το στοιχείο αυτό του υβριδισμού, προκαλεί ενδιαφέρον και μπορεί να αποτελέσει αντικείμενο για περαιτέρω έρευνα για την εφαρμογή τέτοιων μοντέλων στη βιομηχανία λογισμικού.

Σύμφωνα με τον [Leybourne, 2009], μερικές φορές η αυξημένη ελευθερία που προσφέρει η ευέλικτη ανάπτυξη είναι ένα μειονέκτημα και επομένως χρησιμοποιείται ένα μίγμα με παραδοσιακές μεθόδους ώστε να εισαχθεί το στοιχείο του ελέγχου. Από την άλλη πλευρά, ο [Fernandez, 2009] προτείνει όπως η προσαρμοστική και πρακτικής φύσης ευέλικτη ανάπτυξη θα πρέπει να εισάγεται σε παραδοσιακές μεθόδους. Ως εκ τούτου, η δυνατότητα εφαρμογής νέων προσεγγίσεων ή ενός υβριδικού μοντέλου διαχείρισης έργων λογισμικού χρήζει περαιτέρω διερεύνησης, δεδομένου ότι προσφέρει ουσιαστικά οφέλη και λύσεις στα σενάρια αβεβαιότητας, ασάφειας και της πολυπλοκότητας που χαρακτηρίζουν τα σύγχρονα έργα λογισμικού.

Επιπλέον, υπάρχει ένα αυξανόμενο ενδιαφέρον και παράλληλα έλλειψη βιβλιογραφικών αναφορών όσο αφορά την κατανόηση της εφαρμογής της μεθοδολογίας Scrum σε παγκόσμιο κατανεμημένο περιβάλλον (Distributed projects). Είναι ακόμα αμφιλεγόμενο κατά πόσο ή όχι το Scrum μπορεί να χρησιμοποιηθεί με επιτυχία σε κατανεμημένες ομάδες ανάπτυξης λογισμικού. Ο κλάδος αυτός απαιτεί περισσότερη εμπειρική μελέτη για την κατανόηση της χρήσης –γενικότερα- των ευέλικτων πρακτικών σε παγκόσμιο επίπεδο και την εξέταση των περιορισμών που προκύπτουν από ένα τέτοιο εγχείρημα, όπως παράγοντες που ενδέχεται να επηρεάσουν την επικοινωνία, τη συνεργασία και το συντονισμό των διαδικασιών.

Βιβλιογραφία

Agile Alliance (2001), Manifesto for Agile Software Development, http://www.agilemanifesto.org.

Aguanno, K. (2004), Managing Agile Projects, Canada: Multi-media Publications Inc. Augustine, S. and Woodcock, S. (2008), Agile Project Management.

Bauer, F. et al. (1968), Software Engineering: A Report on a Conference Sponsored by NATO Science Committee, NATO.

Beck, K. (1999), Embracing change with Extreme Programming. IEEE Computer, Vol. 32, Issue 10.

Beck, K. (2003), Test Driven Development: By Example, Addison-Wesley.

Beck, K. (2004), Extreme Programming explained: Embrace change. Reading, Mass., Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W.,Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J.,Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2002), Manifesto for Agile Software Development.

Bhardwaj, D. (2011), Scrumming it up: A Survey on Current Software Industry Practices, University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi.

Biju, S. M. (2010), Agile Software Development Methods and its Advantages, Technological Developments in Networking, Education and Automation, pp. 603 – 607.

Boehm B. (1998), A Spiral Model of Software Development and Enhancement, Computer, IEEE, May 1988, pp 61-72.

Bowers J. et al. (2002), Tailoring XP for Large System Mission Critical Software Development,, Proc. 2nd XP Universe and 1st Agile Universe Conf. on Extreme Programming and Agile Methods, Springer, 2002, pp. 100-111

BSI (2002), Project Management-Part 1: Guide to project management, London: BSI.

Burke, R. (2002), Διαχείριση Έργου - Τεχνικές Σχεδιασμού και Έλεγχου. Εκδόσεις Κριτική.

CA Technologies (2010), Balancing agility with governance: A survey of portfolio management for agile IT.

Cocco, L., Mannaro, K., Concas, G. and Marchesi, M. (2001), Simulating Kanban and Scrum vs. Waterfall with System Dynamics, Agile Processed in Software Engineering and Extreme Programming, Lecture Notes in Business Information Processing, vol. 77(1), pp. 117-131.

Cockburn, A. (2001), Agile software development. In: Cockburn A, Highsmith J (eds) The Agile Software Development Series, Addison-Wesley Longman, Reading, Massachusetts.

Cockburn, A. (2005), The Methodology Space.

Cockburn, A. (1999), Methodology per project, Humans and Technology Technical Report HaT TR.

Cohn, M. (2008), Prioritizing Product Backlog, Presentation at Chicago Scrum Gathering.

Cooke, D., Gelman, L., William J., Peterson (2001), ERP Trends, Conference Board.

Deemer et al. (2010), The Scrum Primer, Scrum Training Institute, Version 1.2.

Digital Focus (2006), Agile 2006 Survey: Results and Analysis, Digital Focus was bought by Command Information.

Domingo A. et al (2004), Agile Software Development in Large Organizations, IEEE Computer, pp. 26-34.

Dr. Dobb, J. (2008), Agile Adoption Rate Survey Results: February 2008, Ambysoft

Drobka, J., Noftz, D., and Raghu, R. (2004), Piloting XP on Four Mission-Critical Projects, IEEE Software, Vol. 21, No. 6.

Dynamic System Development Method Consortium, DSDM Tour.

Excirial (2009), A quick overview of the Test-driven development lifecycle, Wikipedia

Fernandez, D.J. (2009), Agile Project Management – Agilism versus Traditional Approaches, Journal of Computer Information Systems, Vol. 49(2), pp. 10-17.

Forrester Research (2009), From Agile Development to Agile Engagement, Vendor Strategy Professionals.

Fowler, M. (2005), Is Design Dead? Software Development.

Fowler, M. (2006), Using an Agile Software Process with Offshore Development.

Fowler, M. (2005), The New Methodology.

Highsmith, J. A. (2000), Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York, NY, Dorset House Publishing

Highsmith, J. and Cockburn, A. (2004), Agile Software Development: The Business of Innovation, IEEE Computer.

Hoda, R., Noble, J. and Marshall, S. (2008), Agile Project Management, New Zealand Computer Science Research Student Conference, Christchurch, New Zealand, pp. 218-221.

Hoda, R., Noble, J. and Marshall, S. (2010), Agile Undercover: When Customers Don't Collaborate, Victoria University of Wellington.

Hundermark, P. (2009), Do Better Scrun An unofficial set of tips and insights into how implement scrum well, Scrum Sense, Second edition.

Inglish, R. (2010), Agile-Scrum vs. Waterfall-Cycle Methodology.

Ionel, N. (2008), Critical Analysis of the Scrum Project Management Methodology, Annals of the University of Oradea Economic Science Series, vol. 7(4), pp. 435 – 441.

Kanth, S. K. (2009), Scrum Methodology in Product Testing: A Practical Approach, Infosys Technologies Limited.

Kerzner, H. (2001), Strategic planning for project management using a project management maturity model, New York: John Wiley & Sons, Inc.

Kniber, H. (2007), Scrum and XP from the Trenches - How we do Scrum, C4 Media Inc., InfoQ Enterprise Software Development Series.

Laplante, P. (2007), What every engineer should know about Software Engineering, CRC Press: Taylor & Francis Group.

Leybourne, S.A. (2009), Improvisation and agile project management: a comparative Consideration, International Journal of Managing Projects in Business, Vol. 2(4), pp. 519-535.

Loeser, A. (2006), Project Management and Scrum – A Side by Side Comparison.

Marks, D. (2002), Development Methodologies Compared, N CYCLES software solutions.

Moore, G. (2002), Crossing the Chasm. HarperBusiness, Revised Edition

Moore, S. and Barnett, L. (2004), Offshore Outsourcing And Agile Development, Forrester Best Practices.

Owen, R., Koskela, L.J., Henrich, G., and Codinhoto, R. (2006), Is Agile Project Management Applicable To Construction?, Proceedings 14th Annual

Conference of the International Group for Lean Construction, Ponteficia Universidad Catolica de Chile, Santiago, Chile.

Palmer , S.R. and Felsing J.M. (2002), J.M., A Practical Guide to Feature-Driven Development. Upper Saddle River, NJ, Prentice-Hall.

Papke-Shields, Beise, K.E., Quan, J. (2009), Do project managers practice what they preach, and does it matter to project success?, International Journal of Project Management, In Press, Corrected Proof.

Pfleeger, S. L. (2003), Τεχνολογία Λογισμικού, Θεωρία και Πράξη (2 ed.). (I. Σταμέλος, Trans.), Εκδόσεις Κλειδάριθμος.

PMBOK. (2004), A Guide to the Project Management Body of Knowledge, 3.Four Campus Boulevard Newton Square, USA: project management institute.

Pressman, S. R. (2005), Software Engineering - A practitioner's approach, Sixth Edition Mc Graw Hill - Higher Education.

Schach, S. (1990), Software Engineering, Vanderbilt University, Aksen Association.

Schatz, B. and Abdelshafi, I. (2005), Primavera Gets Agile: A Successful Transition to Agile Development, IEEE Software, Vol. 22, No. 3

Schwaber, K. and Beedle, M. (2001), Agile Software Development with Scrum, Upper Saddle River, NJ, Prentice – Hall, 1st Edition.

Schwaber, K. and Sutherland J. (2011), The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game, Scrum.org.

Schwaber, K. (2008), It's Not Scrum If..., Presentation at Stockholm Scrum Gathering .

Shine Technologies (2003), Agile Methodologies Survey.

Shore, J. and Warden, S. (2008), The Art of Agile , O'Reilly, O'Reilly Media.

Sommerville, I. (2011), Software Engineering, Addison-Wesley Publishing Company, 9th ed.

Stamelos, I. G. and Sfetsos, P. D. (2007), Agile Software Development Quality Assurance, Idea Group Publishing.

Stapleton, J. (1997), Dynamic systems development method – The method in practice. Addison Wesley.

Sutherland, J. (2004), Agile Development: Lessons learned from the first Scrum.

Takeuchi, H. and Nonaka, I. (1986), The New Product Development Game, Harvard Business Review.

Tinnirello, P. (2001), New Directions in Project Management, Auerbach Publications.

Voas, J. (2007), A Baker's Dozen: 13 Software Enginnering Challenges, IT Professional, 2 (9).

West, D. (2010), Agile Systems Integrators: Plausible Or Paradoxical?, for Application Development & Delivery Professionals.

Williams, L. and Cockburn, A. (2003), Agile Software Development: It's about Feedback and Change, IEEE Computer, pp. 39-43.

Wysocki, K. R. (2006), Effective Software Project Management, Wiley.

Βεσκούκης, Β. (2000), Τεχνολογία Λογισμικού Ι, Ελληνικό Ανοικτό Πανεπιστήμιο.

Δημητριάδης, Α. (1999), Διοικηση-Διαχείριση Έργου (Project Management Μεθοδολογία και Τεχνικές Εφαρμογές με το MS-Project 98, Αθήνα : Εκδόσεις Νέων Τεχνολογιών.

Εμμανουήλ Στ. Σ. (2003), Λογισμική Μηχανική (Software Engineering), Draft-Version 1, Αθηνά.

Παντουβάκης, Π. (2003), Θεωρία και Πράξη στη διαχείριση Έργου, Τεχνικά Χρονικά, Νοέμβριος – Δεκέμβριος 2003.

Σφέτσος, Π. Δ. (2007), Πειραματικές μέθοδοι στην Αντικειμενοστρεφή Τεχνολογία και στις Ευέλικτες Προγραμματιστικές Πρακτικές, Διδακτορική Διατριβή.

Παράρτημα Α

Σε αυτό το παράρτημα παρατίθεται το μήνυμα συμμετοχής που στάλθηκε προς τις διάφορες εταιρίες και μηχανικούς λογισμικού καθώς επίσης και το ερωτηματολόγιο όπως αυτό παρουσιάστηκε στους συμμετέχοντες της έρευνας και στο οποίο κλήθηκαν να απαντήσουν.

Title: Questionnaire about Software Development using Agile methodologies within organizations

Author: Marios Christou

Supervisor: Dr.-Ing. Georgia M. Kapitsaki

Greetings,

I am an undergraduate student of the Computer Science department at the University of Cyprus. I am doing a research for my diploma thesis about Software Development using Agile methodologies. My diploma thesis focuses on Scrum approach but I also would like to examine any other Agile approach. If you work or have worked on a software project using Scrum or any other Agile methodology, I would like to have your input on a questionnaire which is part of my research.

This questionnaire aims to study the effectiveness of Agile approaches compared to traditional approaches. The data from the survey will only be used for academic research. Your responses will be kept strictly confidential. It is guaranteed that the information provided by you will not be disclosed to any third party.

In return for your participation, if you would like, I will send you a draft version of the survey results and keep you informed of any publications based on the survey. To take the survey, go to https://www.surveymonkey.com/s/AgileMethodologiesSurvey. The survey would take less than 10 minutes and will close on **27/04/2012**. If you have

experience on various Agile methodologies – including Scrum – please fill out the questionnaire based on your experience with Scrum.

If you have any suggestions, or would like clarification about any of the questions or how the data will be used, please do not hesitate to contact me or my diploma thesis advisor (contact details are below).

Thank you very much in advance for your time and cooperation.
Sincerely yours,

Marios Christou
Undergraduate student
Department of Computer Science
University of Cyprus
Contact email: marios_2c<AT>hotmail.com

Advisor:
Dr.-Ing. Georgia M. Kapitsaki
gkapi<AT>cs.ucy.ac.cy
http://www.cs.ucy.ac.cy/~gkapi/

(The questionnaire should be distributed among software engineers. Please forward this message to the appropriate professionals on your company or any other company/organization that you think could help.)

Section 1 : Company Information	
1	**Please provide the name of your company/organization (optional):** ☐
2	**Is your organization involved in software development as its primary business activity?** Yes No
3	**How many people are employed at your organization?** 1 - 10 11 - 100 101 - 1000 Over 1000
4	**Where are you based?** Please Specify (Country) : ☐
5	**When it comes to adopting new technologies and methods, your company :** Is happy to adopt the technology Only follows when technology proven Follows a more traditional approach
Section 2 : Agile Methodologies related information *Heavyweight Methodologies are considered the traditional way to develop software using a requirement-design-build paradigm with standard, well- defined processes e.g. Waterfall.* *Agile Methodologies: employ short iterative cycles, and rely on tacit knowledge within a team .e.g. XP programming, SCRUM.*	
5	**Which Heavy methodology do you mostly use?** Waterfall Spiral Unified Process Other (Specify): ☐
6	**Which Agile Methodology do you mostly use?** Scrum

	Extreme Programming (XP) Dynamic Systems Development Method DSDM Feature Driven Development (FDD) Crystal Clear Open Unified Process (OpenUP) Other (Please specify) : ☐
7	**Usage of Scrum by the organization can be described as:** We have never used Scrum before We are currently piloting Scrum We have piloted Scrum but no decision about adoption has been made We are currently deploying Scrum across the organization Scrum is one of the standard methods we use Scrum is the normal way we build software
8	**Given the total amount of development work within your organization what percent is currently being done by Scrum?** Not at all Less than 5% 5-10% Up to 25% About 50% Up to 75% More than 75% 100%
9	**If you have ever been a Scrum Master, do you think that this role was helpful and had a significant contribution towards the project?** Yes No Helpful to some extent I believe it's unnecessary
10	**How satisfied are you with Scrum?** Very pleased with Scrum Scrum exceeds my expectations Scrum is adequate for my needs Disappointing outcome Not at all pleased with Scrum I don't know yet Not applicable
11	**How long have you been using Agile methods?** Less than 1 year

	1-2 years 3-4 years 5-10 years Over 10 years
12	**What is the largest team size your organization has attempted for Agile projects?** 1-5 people 6-10 people 11-20 people 21-50 people 51-100 people 101+ people
13	**On your most recent Agile project, how long are the iterations?** Less than one week 1 week 2 weeks 3 weeks 4 weeks 5 to 6 weeks More than 6 weeks
14	**What is the largest team size with which your organization has been successful with Agile approaches?** 1-5 people 6-10 people 11-20 people 21-50 people 51-100 people 101+ people
15	**How many Agile projects has your organization run?** 1-5 6-10 11-20 21-50 51+
16	**Which of the listed aspects of Agile methodologies most appeal to you compared with Heavyweight Methodologies?** People oriented versus Processes oriented Working code versus Documentation Customer Relationship versus Contract Negotiation

	Respond to change versus following a plan
17	**Which aspect of Agile methodologies, do you dislike the most?** Low Documentation Low planning Less Management Control Lack of Project Structure
18	**What is the overall success rate for all of your Agile projects?** 91-100% 81-90% 71-80% 61-70% 51-60% 41-50% 31-40% 21-30% 11-20% 10% or less Don't Know
19	**What is the success rate for co-located Agile teams? (Co-location means that everyone is in the same room)** 91-100% 81-90% 71-80% 61-70% 51-60% 41-50% 31-40% 21-30% 11-20% 10% or less Not Applicable Don't Know
20	**What is the success rate for Agile teams that are not co-located? (Not co-located means that some team members are in different cubicals/offices, on different floors, or in different buildings, but could still get together easily if required)** 91-100% 81-90% 71-80% 61-70%

	51-60% 41-50% 31-40% 21-30% 11-20% 10% or less Not Applicable Don't Know
21	**What is the success rate for Agile teams with an offshoring/outsourcing element? (Some team members are at a different location which would require significant travel to get to)** 91-100% 81-90% 71-80% 61-70% 51-60% 41-50% 31-40% 21-30% 11-20% 10% or less Not Applicable Don't Know
22	**How have Agile approaches affected your productivity compares with Heavyweight Methodologies?** Much higher Somewhat higher No change Somewhat lower Much lower Don't know
23	**How have Agile approaches affected the quality of the systems produced compares with Heavyweight Methodologies?** Much higher Somewhat higher No change Somewhat lower Much lower Don't know
24	**How have Agile approaches affected the cost of development compares with Heavyweight Methodologies?**

	Much higher Somewhat higher No change Somewhat lower Much lower Don't know
25	**How have Agile approaches affected stakeholder satisfaction compares with Heavyweight Methodologies?** Much higher Somewhat higher No change Somewhat lower Much lower Don't know
26	**What do you believe is the most common problem experienced while practicing Agile methodologies?** Lack of skilled people who can follow Scrum Lack of Top Management Support Lack of Customer Collaboration Project Size/Complexity Project Team Size
27	**Which of the Agile methods do you prefer the most?** Extreme Programming (XP) Scrum Dynamic Systems Development Method DSDM Feature Driven Development (FDD) Crystal Clear Open Unified Process (OpenUP) Other (Please specify) : ☐
28	**Do you think that your company will again adopt any of Agile methods in the future?** Yes No Don’t know
Section 3 : Demographic Information	
29	**What is your age?**

	18-30 30-40 40-50 50-60 Above 60
30	**What is your Gender?** Male Female
31	**What is your highest educational qualification?** High school diploma Technical degree University degree Master PhD Other (Specify):
32	**Which best describes your current position?** Business Stakeholder Data Professional Software Engineer/Developer IT Management Modeler Operations/Support Staff Project Manager Quality Assurance/Tester Other
33	**How many years of work experience do you have in software development?** None 1-4 5-10 11-15 Over 15
34	**For how long have you been a member of teams that practiced Agile approaches?** Less than 1 year 1-2 years 3-4 years 5-10 years Over 10 years

35	**Do you have any additional comments?**

Παράρτημα Β

Στο παράρτημα αυτό παρουσιάζονται οι απαντήσεις στις ερωτήσεις της έρευνας που διεξήχθη με ερωτηματολόγιο.

Section 1 : Company Information

1. Please provide the name of your company/organization (optional):

Answer Options	Response Count
	139
answered question	139
skipped question	196

2. Is your organization involved in software development as its primary business activity?

Answer Options	Response Percent	Response Count
Yes	71,7%	236
No	28,3%	93
answered question		329
skipped question		6

3. How many people are employed at your organization?

Answer Options	Response Percent	Response Count
1 - 10	19,7%	65
11 - 100	24,2%	80
101 - 1000	25,5%	84
Over 1000	30,6%	101
answered question		330
skipped question		5

4. Where are you based?

Answer Options	Response Count
	313
answered question	313
skipped question	22

5. When it comes to adopting new technologies and methods, your company :

Answer Options	Response Percent	Response Count
Is happy to adopt the technology	62,4%	204
Only follows when technology proven	30,3%	99
Follows a more traditional approach	7,3%	24
answered question		327
skipped question		8

Section 2 : Agile Methodologies related information

6. Which Heavy methodology do you mostly use?

Answer Options	Response Percent	Response Count
Waterfall	36,5%	81
Spiral	14,4%	32
Unified Process	12,2%	27
Other (please specify)	36,9%	82
answered question		222
skipped question		113

7. Which Agile Methodology do you mostly use?

Answer Options	Response Percent	Response Count
Scrum	76,9%	180
Extreme Programming (XP)	6,4%	15
Dynamic Systems Development Method DSDM	1,3%	3
Feature Driven Development (FDD)	3,8%	9
Crystal Clear	0,4%	1
Open Unified Process (OpenUP)	0,4%	1
Other (please specify)	10,7%	25
answered question		234
skipped question		101

8. Usage of Scrum by the organization can be described as:

Answer Options	Response Percent	Response Count
We have never used Scrum before	5,2%	12
We are currently piloting Scrum	9,5%	22
We have piloted Scrum but no decision about adoption has been made	10,8%	25
We are currently deploying Scrum across the organization	14,7%	34
Scrum is one of the standard methods we use	27,3%	63

Scrum is the normal way we build software	32,5%	75
	answered question	231
	skipped question	104

9. Given the total amount of development work within your organization what percent is currently being done by Scrum?

Answer Options	Response Percent	Response Count
Not at all	7,4%	17
Less than 5%	8,2%	19
5-10%	11,3%	26
Up to 25%	12,1%	28
About 50%	13,4%	31
Up to 75%	10,4%	24
More than 75%	22,1%	51
100%	15,2%	35
	answered question	231
	skipped question	104

10. If you have ever been a Scrum Master, do you think that this role was helpful and had a significant contribution towards the project?

Answer Options	Response Percent	Response Count
Yes	73,7%	157
No	2,8%	6
Helpful to some extent	19,2%	41
I believe it's unnecessary	4,2%	9
	answered question	213
	skipped question	122

11. How satisfied are you with Scrum?

Answer Options	Response Percent	Response Count
Very pleased with Scrum	38,0%	87
Scrum exceeds my expectations	10,9%	25
Scrum is adequate for my needs	38,0%	87
Disappointing outcome	3,1%	7
Not at all pleased with Scrum	3,9%	9
I don't know yet	2,6%	6
Not applicable	3,5%	8
	answered question	229
	skipped question	106

12. How long have you been using Agile methods?

Answer Options	Response Percent	Response Count

Less than 1 year	6,6%	15
1-2 years	24,0%	55
3-4 years	30,6%	70
5-10 years	27,1%	62
Over 10 years	11,8%	27
	answered question	229
	skipped question	106

13. What is the largest team size your organization has attempted for Agile projects?

Answer Options	Response Percent	Response Count
1-5 people	14,3%	33
6-10 people	35,2%	81
11-20 people	25,2%	58
21-50 people	12,2%	28
51-100 people	7,0%	16
101+ people	6,1%	14
	answered question	230
	skipped question	105

14. On your most recent Agile project, how long are the iterations?

Answer Options	Response Percent	Response Count
Less than one week	3,9%	9
1 week	10,4%	24
2 weeks	48,1%	111
3 weeks	21,6%	50
4 weeks	12,1%	28
5 to 6 weeks	3,5%	8
More than 6 weeks	0,4%	1
	answered question	231
	skipped question	104

15. What is the largest team size with which your organization has been successful with Agile approaches?

Answer Options	Response Percent	Response Count
1-5 people	15,3%	35
6-10 people	45,4%	104
11-20 people	22,7%	52
21-50 people	9,6%	22
51-100 people	3,1%	7
101+ people	3,9%	9
	answered question	229
	skipped question	106

16. How many Agile projects has your organization run?

Answer Options	Response Percent	Response Count
1-5	40,1%	91
6-10	19,4%	44
11-20	14,5%	33
21-50	10,1%	23
51+	15,4%	35
answered question		227
skipped question		108

17. Which of the listed aspects of Agile Methodologies most appeal to you compared with Heavyweight Methodologies?

Answer Options	Response Percent	Response Count
People oriented versus Processes oriented	22,6%	51
Working code versus Documentation	21,2%	48
Customer Relationship versus Contract Negotiation	10,6%	24
Respond to change versus following a plan	45,6%	103
answered question		226
skipped question		109

18. Which aspect of Agile methodologies, do you dislike the most?

Answer Options	Response Percent	Response Count
Low Documentation	35,2%	70
Low planning	16,6%	33
Less Management Control	10,1%	20
Lack of Project Structure	38,2%	76
answered question		199
skipped question		136

19. What is the overall success rate for all of your Agile projects?

Answer Options	Response Percent	Response Count
91-100%	28,3%	65
81-90%	25,7%	59
71-80%	19,6%	45
61-70%	6,5%	15
51-60%	3,5%	8
41-50%	0,9%	2
31-40%	0,9%	2
21-30%	0,4%	1
11-20%	0,4%	1
10% or less	0,0%	0
Don't Know	13,9%	32

answered question	230
skipped question	105

20. What is the success rate for co-located Agile teams? (Co-location means that everyone is in the same room)

Answer Options	Response Percent	Response Count
91-100%	32,3%	74
81-90%	21,8%	50
71-80%	11,8%	27
61-70%	3,9%	9
51-60%	3,1%	7
41-50%	0,4%	1
31-40%	0,0%	0
21-30%	0,4%	1
11-20%	0,9%	2
10% or less	0,0%	0
Not Applicable	14,8%	34
Don't Know	10,5%	24
answered question		229
skipped question		106

21. What is the success rate for Agile teams that are not co-located? (Not co-located means that some team members are in different cubicals/offices, on different floors, or in different buildings, but could still get together easily if required)

Answer Options	Response Percent	Response Count
91-100%	16,4%	37
81-90%	15,9%	36
71-80%	8,8%	20
61-70%	10,2%	23
51-60%	5,3%	12
41-50%	3,1%	7
31-40%	2,7%	6
21-30%	1,3%	3
11-20%	0,9%	2
10% or less	1,3%	3
Not Applicable	18,1%	41
Don't Know	15,9%	36
answered question		226
skipped question		109

22. What is the success rate for Agile teams with an offshoring/outsourcing element? (Some team members are at a different location which would require significant travel to get to)

Answer Options	Response Percent	Response Count
91-100%	8,7%	20
81-90%	12,6%	29
71-80%	8,3%	19

61-70%	9,6%	22
51-60%	6,5%	15
41-50%	2,2%	5
31-40%	4,3%	10
21-30%	3,9%	9
11-20%	2,6%	6
10% or less	1,7%	4
Not Applicable	25,7%	59
Don't Know	13,9%	32
	answered question	230
	skipped question	105

23. How have Agile approaches affected your productivity compares with Heavyweight Methodologies?

Answer Options	Response Percent	Response Count
Much higher	43,9%	100
Somewhat higher	39,5%	90
No change	5,7%	13
Somewhat lower	2,6%	6
Much lower	1,3%	3
Don't know	7,0%	16
	answered question	228
	skipped question	107

24. How have Agile approaches affected the quality of the systems produced compares with Heavyweight Methodologies?

Answer Options	Response Percent	Response Count
Much higher	39,7%	91
Somewhat higher	38,9%	89
No change	10,9%	25
Somewhat lower	2,6%	6
Much lower	0,4%	1
Don't know	7,4%	17
	answered question	229
	skipped question	106

25. How have Agile approaches affected the cost of development compares with Heavyweight Methodologies?

Answer Options	Response Percent	Response Count
Much higher	6,6%	15
Somewhat higher	12,3%	28
No change	19,7%	45
Somewhat lower	29,8%	68
Much lower	15,8%	36
Don't know	15,8%	36
	answered question	228

skipped question	107

26. How have Agile approaches affected stakeholder satisfaction compares with Heavyweight Methodologies?

Answer Options	Response Percent	Response Count
Much higher	40,6%	93
Somewhat higher	37,6%	86
No change	8,3%	19
Somewhat lower	3,5%	8
Much lower	0,4%	1
Don't know	9,6%	22
answered question		229
skipped question		106

27. What do you believe is the most common problem experienced while practicing Agile methodologies?

Answer Options	Response Percent	Response Count
Lack of skilled people who can follow Scrum	21,2%	48
Lack of Top Management Support	30,5%	69
Lack of Customer Collaboration	31,9%	72
Project Size/Complexity	12,4%	28
Project Team Size	4,0%	9
answered question		226
skipped question		109

28. Which of the Agile methods do you prefer the most?

Answer Options	Response Percent	Response Count
Extreme Programming (XP)	14,9%	34
Scrum	64,9%	148
Dynamic Systems Development Method DSDM	1,8%	4
Feature Driven Development (FDD)	4,4%	10
Crystal Clear	0,4%	1
Open Unified Process (OpenUP)	0,9%	2
Other (please specify)	12,7%	29
answered question		228
skipped question		107

29. Do you think that your company will again adopt any of Agile methods in the future?

Answer Options	Response Percent	Response Count
Yes	90,4%	208
No	1,7%	4
Don’t know	7,8%	18
answered question		230

skipped question	105

Section 3 : Demographic Information

30. What is your age?

Answer Options	Response Percent	Response Count
18-30	12,1%	27
30-40	41,1%	92
40-50	28,1%	63
50-60	14,3%	32
Above 60	4,5%	10
answered question		224
skipped question		111

31. What is your Gender?

Answer Options	Response Percent	Response Count
Male	90,5%	200
Female	9,5%	21
answered question		221
skipped question		114

32. What is your highest educational qualification?

Answer Options	Response Percent	Response Count
High school diploma	4,0%	9
Technical degree	8,0%	18
University degree	38,9%	88
Master	42,5%	96
PhD	4,4%	10
Other (please specify)	2,2%	5
answered question		226
skipped question		109

33. Which best describes your current position?

Answer Options	Response Percent	Response Count
Business Stakeholder	8,4%	19
Data Professional	1,3%	3
Software Engineer/Developer	27,0%	61
IT Management	25,2%	57
Modeler	0,4%	1
Operations/Support Staff	0,9%	2

Project Manager	23,0%	52
Quality Assurance/Tester	2,2%	5
Other	11,5%	26
answered question		226
skipped question		109

34. How many years of work experience do you have in software development?

Answer Options	Response Percent	Response Count
None	0,0%	0
1-4	6,2%	14
5-10	18,1%	41
11-15	31,7%	72
Over 15	44,1%	100
answered question		227
skipped question		108

35. For how long have you been a member of teams that practiced Agile approaches?

Answer Options	Response Percent	Response Count
Less than 1 year	6,6%	15
1-2 years	25,7%	58
3-4 years	29,6%	67
5-10 years	26,5%	60
Over 10 years	11,5%	26
answered question		226
skipped question		109

36. Do you have any additional comments?

Answer Options	Response Count
	51
answered question	51
skipped question	284

Παράρτημα Γ

Στο τρίτο αυτό παράρτημα παρουσιάζονται τα σχόλια που συλλέχτηκαν από τους συμμετέχοντες στην έρευνα (Ερώτηση 36).

	Response Date	Response Text
1.	Mar 23, 2012 12:49 PM	Scrum is a framework. Other processes fit within it. XP is programming discipline. Others range between. So this has been confusing.
2.	Mar 24, 2012 1:50 AM	One of the questions asks how satisfied I am with Scrum. I see a pattern in all agile "methods"/frameworks, they all allow to make a process improvement machine in your organization. This makes me very satisfied. About what I dislike most, considering what I wrote above, we could use this process improvement machine to make improvements so we do not have lack of documentation (we have just enough) and other problems listed there. About rate of success, you should define it better. Is is stakeholder's satisfaction? Or is it meeting milestones dates? What about developer's satisfaction etc? I considered we had 100% success rate, but we did miss some important dates, and several other problems, but we were always able to overcome these. Hope I could help.
3.	Mar 24, 2012 5:12 PM	You asked about the # of projects, we don't look at it that way. We are a Scrum shop in an Agile company. We use Scrum primarily, with a lot of XP, Kanban, BDD/TDD and Lean principles included where and when appropriate.
4.	Mar 25, 2012 7:30 PM	The biggest thing to overcome when adopting agile is doubtful team members. You need a team that is open minded with a strong scrum master who does not over-manage.
5.	Mar 25, 2012 9:36 PM	Biggest obstacle that we have faced is adapting to change. Especially when you are transitioning from waterfall to agile.
6.	Mar 26, 2012 12:19 AM	To make scrum success we added some elements in the beginning (pre-game), that includes some level of establishing architecture choices, budget and release schedule, UX and Quality Objectives. For Maintenance of software we use Kanban, iterative approach to handle daily work as per priority and capacity. Please check our SaaS tools for consulting (includes scrum based project management) at http://www.xamun.com
7.	Mar 26, 2012 8:02 PM	At my current employer, I am piloting Scrum. So consider that factor while reviewing my feedback. But at my previous employer, I

		have done a number of agile projects.
8.	Mar 27, 2012 2:50 AM	Largest implementation I did with Agile practices is 400 people at Sprint PCS. I find Scrum a step in the right direction but generally abused and often unable to complete Sprints successfully. Kanban has been the best approach for an organization to launch features in production in a 1 to 2 week cycle. This continuous delivery gives the highest flexibility to the company to change priorities. It does require a high degree of engineering practices to be successfully with fully automated feature, functional and unit tests along with environment suites and continuous integration. Best practices with software configuration management are also key to success.
9.	Mar 28, 2012 2:34 PM	In our pilot: 1. Agile lowered overall defect rate. 2. Brought forward architectural issues early on. 3. Build a stronger relationship with the business. 4. Provided better communication with a distributed team. 5. Served as a very effective method to remove roadblocks. (our retrospective were really cumulative, with a list of actions items/ people responsible/and dates.
10.	Mar 28, 2012 3:03 PM	We purposely don't use Scrum. The Scrum-related questions didn't all have a "Not Applicable" answer, so I had to pick one of the available answers, even though they were not correct.
11.	Mar 30, 2012 5:17 AM	good luck
12.	Mar 30, 2012 12:56 PM	Improving Enterprises is a consulting organization that provides Software Development (usually using Agile) and Agile Coaching to assist corporate adoption of Agile.
13.	Apr 4, 2012 5:05 PM	Agile is very effective, although for large enterprise efforts with 10+ tracks for the product (6-8 members per track) it gets complex quickly and the interaction of stories betweek tracks are where the artistry is...
14.	Apr 4, 2012 5:59 PM	I think this survey lost a lot of quality by having single choice questions which were missing quite popular choices or "other (specify)". If my team is currently using kanban we are very agile but none of the alternatives would fit. Same thing with the "heavy weight" alternatives. "None of the above" or "none at all" would probably have been the most selected options for most people answering (with an agile background)
15.	Apr 4, 2012 7:27 PM	I'm a Technical Publications Manager. In our company, technical writers are one of three key skill sets in R&D, the others being developers and testers. In our experience, Agile Scrum is particularly bad at incorporating technical writers and performance testers into the Scrum team. Tools aren't set up to accommodate their needs; iterations are developer-focused. Agile Scrum assumes you have the bandwidth to address at least one complete "chunk" of work within the iteration. However, writers often are trying to document the output of many different developers on their team

		while getting the developers and testers to verify the documentation. (Note: the writers do verify their own documentation as well, but it is a best practice for dev & test to validate it.) So, doc and perf testing iterations are often at least one if not more iterations behind development. There are workarounds, such as hiring more staff, but they are not done in this economy.
16.	Apr 4, 2012 8:01 PM	Team size of agile teams varies with project and project dependencies. For example, sprint 1 (no dependencies) may start small, sprint 5 (more dependencies) could have more resources. One issue is that not all resources are fully utilised all the time in a Scrum. Some key roles like a Business Analyst or Solutions Architect don't have a clear role in Agile, though they play a key role for the Scrum team too. Running multiple Scrums, which would depend on a central testing approach could cause a lot of delays for Scrums and affect the performance. In my view Agile has to be seen in context of Project Method, IT operating model and Test Method too. Most organisation have a mix and Scrum needs to align with other methods. Finally, conflicting methods of use cases, user stories, requirements (detailed/high-level) and process diagrams too.
17.	Apr 4, 2012 10:38 PM	Agile is a journey - not a destination
18.	Apr 5, 2012 7:23 AM	Questions that ask for a percentage success rate. It depends how you measure success. Make sure that is defined. Scrum has several characteristics that in turn bring success (success = return on investment because, fit for purpose, in budget, adopted and by business customer, bringing the key benefits).
19.	Apr 5, 2012 9:17 AM	The emphasis on scrum / agile on some of the multiple choice questions might give biased results, because now a company that's "been there done that" and is NO LONGER using agile but a more lightweight approach is grouped with companies that are doing waterfall and no agile at all.
20.	Apr 5, 2012 12:56 PM	You are missing Scrumban and Kanban, and forcing to pick a "least favourite" side of agile is a bit iffy. Also you're missing director/VP-level from the position list.
21.	Apr 7, 2012 8:37 AM	Be adaptive to the company culture.
22.	Apr 8, 2012 10:19 PM	I tried to get our current project done via Agile, and they wimped out. It failed.
23.	Apr 8, 2012 10:47 PM	UK government has recently mandated Agile for all its new IT projects. Currently working on largest Agile project in the world! (Universal Credit)
24.	Apr 8, 2012 11:23 PM	The T.R.A.M. is really the way to go. Especially since it can be used with SCRUM.
25.	Apr 9, 2012 12:02 AM	Your survey question about SCRUM assumes it was or will be the choice. We tried it, it was not very good and we moved to XP... it was not an option in your survey.
26.	Apr 9, 2012 12:16 AM	I first started adopting Agile approaches in 1983, then called 4GL. With current tools (eg .NET,SQLServer) and 40+ years IT experience I can do end-to-end projects from client's mouth to

		implemented system. The trick is that each delivery is 30mh of effort :)
27.	Apr 9, 2012 1:35 AM	We were using agile practices without knowing their names, but now we are more aware about agile, their different methods and their practices since they fit more with the current rapid environment.
28.	Apr 9, 2012 1:49 AM	Scrum is a project management method, not an Agile Approach. It is only coincidentally "Agile Friendly"
29.	Apr 9, 2012 2:14 AM	There are no questions about how well the agile practices were implemented. In both companies I've had an agile structure, it was supported by many, but turned out to be crippled by confusion and corporate resistance.
30.	Apr 9, 2012 2:19 AM	I am Scrum Master now. However Scrum Master is not listed in the position.... I believe it should be there... Together with PO.
31.	Apr 9, 2012 6:39 AM	Have a Working software is the key success factor for agile, & clear progress is the key success factor for scrum project
32.	Apr 9, 2012 6:43 AM	Overall I like working in scrum teams, however I still fail to see any benefit for the sake of the project. All projects I have worked on as a tester have all been run according to the "book" and that simply is wrong in my opinion and the main reason for all of these methods (including the heavyweight methods) to fail. You should only pick those parts of any method that seem to work for your team and not do something for the sake of following the method.
33.	Apr 9, 2012 9:31 AM	Senior management has a low understanding of agile. Many middle managers are threatened by agile given that agile teams tend to be self-directed. The root of the lack of understanding and fear is loss of control. Once one understands the role of a leader for an agile team the lack of understanding and fear disappears. Management 3.0 is an excellent book that describes the role of a leader for an agile organization.
34.	Apr 9, 2012 9:36 AM	Success!!
35.	Apr 9, 2012 10:55 AM	The current position is Product Manager (read Product Owner) who not only looks after detailing, scheduling & delivery, but also pre-sales/sales, product marketing, customer satisfaction & legal/IP aspects. It's a pretty vast role. I would like to have a copy of your conclusions about this survey to put us in a better position with our Agile strategy.
36.	Apr 9, 2012 11:44 AM	Very long survey
37.	Apr 9, 2012 12:19 PM	Question 18 is a bad question. This implies that the right side of the manifesto is not produced. The manifesto is about values, you should read the lead in. The selections in question 18 are all valued and are produced as required.
38.	Apr 9, 2012 12:28 PM	Questions 12 and 35 are tricky to answer. Iterative methodologies that precipitated the Agile Manifesto have been around for more than 10 years. But the word "Agile" was used to consolidate them into a category of methodologies was only established 10 years ago. I been using Iterative and Incremental techniques with lightweight processes and tight customer interaction for almost 30 years. See the

		IEEE paper presented by Winston Royce in 1970 where he describes waterfall and then explains why it is a bad idea followed by a description of iterative techniques. "Agile" is 10 years old. Iterative with close interaction has been used successfully for more than 40 years.
39.	Apr 9, 2012 1:06 PM	We use MAnGve for implementing Agile Governance
40.	Apr 9, 2012 1:41 PM	Good luck with your research! :)
41.	Apr 9, 2012 2:04 PM	Pretty good questionnaire, however I have to disagree with the fixed options in Question 18: Which aspect of Agile methodologies, do you dislike the most? Reason: Agile methodologies do not prescribe low documentation or low planning - you can do them as much as you want or need. Also, you actually get MORE management control, as management can clearly see every 1-4 weeks what is happening, and change or adjust the plan if needed. Also, Scrum actually brings MORE project structure than an average waterfall project, by applying strict time-boxed iterations (instead of following seemingly strict waterfall phases that in practice go "old" a few days after the project plan is written, and things are then done mostly ad-hoc). What I personally dislike is the fact that many people want to believe in unrealistic plans instead of understanding and embracing change: Projects are actually learning experiences, and it is relatively easy to build the final product when we truly understand the underlying questions and problems in the project. Agile methods allow us to learn as we go, instead of assuming that everything is known beforehand, when we plan the project. Good luck with your research and studies! Lare Lekman Professional Scrum Trainer & Certified Coach
42.	Apr 9, 2012 3:03 PM	We use DSDM for all of our projects here, it and others. I was trained in 1999 in DSDM and have watched true collaborative aspects of the agile movement lessen, not improve over the years. Sad! DSDM was the birthplace of the international certification for professional facilitators in 1998, the International Association of Facilitator's CPF. As collaboration in business has grown by leaps and bounds, collaboration on the IT Agile side has been misunderstood and picked apart, much like Deming's total quality was dismantled for expediency. In both cases, gurus saw the chance to take shortcuts and make a lot of money. All to the detriment of excellence in Agile projects. Agile as we know it today is a sad reflection of Jim Highsmith's and others' original vision. It was so

		much more. Read history of the movement. Understand what could be. Get beyond drinking the SCRUM Koolaid.
43.	Apr 9, 2012 3:50 PM	Agile / Scrum is the best practice for the quickest show of project development to a client. I would never go back to standard practice.
44.	Apr 9, 2012 5:02 PM	Agile is the best process and secret key for a success project... Cheer, DANYAL(danyal.sahu@gmail.com)
45.	Apr 9, 2012 8:32 PM	Thank you.
46.	Apr 9, 2012 10:56 PM	In my team we work with a mix of Scrum and Kanban. And we've been very successful over the last year. But Kanban alone didn't work at all.
47.	Apr 10, 2012 7:45 AM	I wrote a PhD thesis about the relation between project management, agile software development methods and organisational structures. I am convinced, there are some aspects you could use in our thesis. If you are interested in some details, send me a note to wolfgang@richter.jipp.it .
48.	Apr 10, 2012 8:36 AM	Not sure why you've excluded the Scrum roles from the Positions list - the Project Manager is now equal parts Scrum Master and Product Owner, and should be seen separately. I feel in this survey you have limited the options too much to have representative feedback. I really don't understand why you ask for a 'most valuable' option of the principles of the agile manifesto, but I see it as damaging to have a short and by no means comprehensive list of "typical" problems (the next question) and ask "which do you least like about agile". The question could be "which have affected you" - as a multiple choice question - but I feel you are in danger of misrepresenting agile when the results indicate "70% of people least liked the low level of planning" when in fact the low level of planning is only relevant in highly exploratory work, where no amount of gant charts will help anyone. The same for the rest of the list, so your data is likely to misrepresent 'hot issues' that are in fact non-issues.
49.	Apr 11, 2012 3:19 AM	You assume a person that works inside an organization will follow the organization approaches. Remember that Agile is a new methodology and some organizations hesitate into adopt new methodologies, you should make questions regarding the "official" approach of an organization and questions regarding the "individual" experience of members of the organization that are pushing Agile.
50.	Apr 12, 2012 12:05 AM	Scrum is not the only Agile practice. Its actually one that fails a lot. You are also missing TDD
51.	Apr 12, 2012 4:17 AM	We use kanban for some teams instead of scrum. Perhaps 20-30% of our workers are on kanbans. The successful scrum project was an MMORPG game that used over 100 people at its peak.