ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ Μ/Υ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΝΑΥΤΙΛΙΑΣ ΚΑΙ ΒΙΟΜΗΧΑΝΙΑΣ
ΤΜΗΜΑΤΟΣ ΒΙΟΜΗΧΑΝΙΚΗΣ ΔΙΟΙΚΗΣΗΣ & ΤΕΧΝΟΛΟΓΙΑΣ
ΔΙΑΠΑΝΕΠΙΣΤΗΜΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΤΕΧΝΟ-ΟΙΚΟΝΟΜΙΚΑ ΣΥΣΤΗΜΑΤΑ»

ΔΙΕΠΙΣΤΗΜΟΝΙΚΟ – ΔΙΑΠΑΝΕΠΙΣΤΗΜΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΤΕΧΝΟ-ΟΙΚΟΝΟΜΙΚΑ ΣΥΣΤΗΜΑΤΑ»

Διασφάλιση Ποιότητας σε Agile Ανάπτυξη Λογισμικού - Υλοποίηση Εφαρμογής Αυτοματοποιημένης Δημιουργίας Test Cases

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευαγγελία Ι. Παπαθανασίου

Επιβλέπων: Δημήτριος Ασκούνης, Καθηγητής ΕΜΠ

Συν-Επιβλέπων (ΕΔΙΠ): Δημήτριος Πανόπουλος Διδάκτωρ ΕΜΠ

Αθήνα, Φεβρουάριος 2024

ΔΙΕΠΙΣΤΗΜΟΝΙΚΟ – ΔΙΑΠΑΝΕΠΙΣΤΗΜΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΤΕΧΝΟ-ΟΙΚΟΝΟΜΙΚΑ ΣΥΣΤΗΜΑΤΑ»

Διασφάλιση Ποιότητας σε Agile Ανάπτυξη Λογισμικού - Υλοποίηση Εφαρμογής Αυτοματοποιημένης Δημιουργίας Test Cases

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευαγγελία Ι. Παπαθανασίου

Επιβλέπων: Δημήτριος Ασκούνης, Καθηγητής ΕΜΠ

Συν-Επιβλέπων (ΕΔΙΠ): Δημήτριος Πανόπουλος Διδάκτωρ ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29 Φεβρουαρίου 2024.

.....	.....	.....
Δημήτριος Ασκούνης	Ιωάννης Ψαρράς	Ευάγγελος Μαρινάκης
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Επίκουρος Καθηγητής ΕΜΠ

Αθήνα, Φεβρουάριος 2024

.....
Ευαγγελία, Ι Παπαθανασίου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών ΕΜΠ

Περίληψη

Στα πλαίσια της παρούσας διπλωματικής εργασίας, έχει μελετηθεί η ανάπτυξη λογισμικού και η διασφάλιση ποιότητας του, χρησιμοποιώντας την μεθοδολογία Agile. Πέρα από το θεωρητικό πλαίσιο, έχει υλοποιηθεί η web εφαρμογή **Smart Test Cases tool** για τις ανάγκες της ανάλυσης, ανάπτυξης και ελέγχου ενός λογισμικού (software). Η εφαρμογή απευθύνεται σε κάθε Agile ομάδα ανάπτυξης λογισμικού, η οποία για την υλοποίηση ενός project, βασίζεται στα user stories, όπως προκύπτουν από την εκάστοτε ανάλυση του έργου, και εν συνεχεία στα αντίστοιχα test cases που απαιτούνται ανά περίπτωση για την διασφάλιση της ποιότητας του.

Δεδομένου ότι γίνεται Agile ανάπτυξη εφαρμογής, χρησιμοποιούνται τα user stories που περιγράφουν τι πρέπει να υλοποιηθεί από τους προγραμματιστές και αναφέρονται λεπτομερώς στα στοιχεία που θα υπάρχουν στην εφαρμογή καθώς και στην αναμενόμενη λειτουργικότητα τους. Τα test cases με την σειρά τους καλύπτουν σενάρια τόσο για τους επιχειρησιακούς ελέγχους όσο και στους λογικούς ελέγχους. Όσο αναφορά την φάση του ελέγχου του λογισμικού, η ομάδα διασφάλισης ποιότητας καλείται να κατανοήσει τα user stories και να γράψει τα αντίστοιχα test cases που προκύπτουν ανά πεδίο και acceptance criterion και στην συνέχεια να τα εκτελέσει μέσω manual ή automation testing. Τα test cases που αφορούν τους λογικούς ελέγχους των πεδίων επαναλαμβάνονται σε κάθε user story άρα κάθε φορά θα πρέπει να δημιουργούνται οι ίδιοι έλεγχοι για όμοια πεδία. Η διαδικασία αυτή είναι αρκετά χρονοβόρα και επαναλαμβανόμενη καθώς επίσης έχουν παρατηρηθεί περιπτώσεις που τα βασικά σενάρια έχουν παραληφθεί από λάθος και στο τέλος δημιουργούν σφάλματα κατά την χρήση της εφαρμογής.

Επομένως, η υλοποίηση της εφαρμογής καλύπτει την ανάγκη να δημιουργούνται αυτόματα τα test cases μέσω των user stories τουλάχιστον για τους λογικούς ελέγχους και δίνει την δυνατότητα για πιο παραγωγική εργασία τόσο της ομάδας ελέγχου όσο και της ομάδας ανάπτυξης. Κατά τη χρήση της εφαρμογής, ο αναλυτής μπορεί να εισάγει τα user stories με τα αντίστοιχα πεδία που πρέπει να υλοποιηθούν, καθώς και τους διάφορους περιορισμούς ανά πεδίο. Οι περιορισμοί αυτοί ανάγονται σε test cases από την ομάδα ελέγχου, ωστόσο μέσω της εφαρμογής υπάρχει η δυνατότητα να δημιουργηθούν αυτόματα τα test cases που αφορούν τους λογικούς ελέγχους κάθε πεδίου και προκύπτουν από τα user stories. Τέλος, ακόμη μια βασική λειτουργία της εφαρμογής είναι η διαχείριση των έργων (projects), των user stories και των test cases.

Λέξεις Κλειδιά: έλεγχος ποιότητας λογισμικού, agile ανάπτυξη λογισμικού, user stories, test cases

Abstract

In terms of the respective thesis, software development, and quality assurance have been studied using the Agile methodology. On top of the theoretical part, the web application Smart Test Cases tool has been implemented for the needs of analysis, development and testing of a software product. The web application is oriented toward each Agile development team which is based on the user stories, to implement a software product as they originated from the analysis, and later the relevant test cases which are required per case for the quality assurance.

Given that the software development is based on Agile methodology, user stories are used which describe what should be implemented by the developers and mention thoroughly the elements that exist in the app as well as their expected functionality (acceptance criteria). The test cases cover the scenarios for both business and logical tests. Regarding the testing of the software, the quality assurance team is asked to understand the user stories and to write up the respective test cases per field and acceptance criterion and later on execute them through manual and automation testing. The test cases related to the logical tests of the fields are repeated in each user story, so every time the same tests should be created for the same fields. This process is time-consuming and recurrent, as well as cases are noticed in which the basic scenarios are missed by mistake and finally errors occur during the use of the application.

Consequently, the implementation of the application covers the need of auto-generating the test cases related to the logical tests from the user stories and also it is a tool for the efficient work of both the testing and development teams. Using the application, the analyst can input the user stories with the respective fields that should be implemented, as well as any limitations per field. From these kinds of limitations, the test team can extract the test cases related to the logical tests, however the test cases can be generated automatically through the application since they are coming from the user stories. Last but not least, another basic functionality of the application is the management of the projects, the user stories and the test cases.

Keywords: software quality assurance, agile software development, user stories, test cases

Περιεχόμενα

1. Εισαγωγή

1.1. Ανάπτυξη Λογισμικού και Έλεγχος (Software Development and Testing)

Η Ανάπτυξη Λογισμικού (Software Development) αναφέρεται στις δραστηριότητες της Επιστήμης των Υπολογιστών (Computer Science) που απαιτούνται για την διαδικασία δημιουργίας, σχεδίασης, ανάπτυξης και υποστήριξης λογισμικού. Η διαδικασία της ανάπτυξης λογισμικού αποτελείται από τα στάδια της Καταγραφή και Ανάλυση Απαιτήσεων (Requirements analysis and specification), τον Σχεδιασμό και την Ανάπτυξη (Design and Development), τον Έλεγχο (Testing), την Διάθεση προς χρήση (Deployment) και την Συντήρηση και Υποστήριξη (Maintenance and Support). Τα στάδια αυτά μπορούν να ομαδοποιηθούν ανάλογα με την φάση του κύκλου ζωής του λογισμικού αλλά της μεθοδολογίας που έχει επιλεχθεί [1].

Ανεξάρτητα από το την μεθοδολογία ή το είδος του λογισμικού, το στάδιο της αξιολόγησης και επαλήθευσης είναι σημαντικό και κρίσιμο δεδομένου ότι διασφαλίζει την αξιόπιστη λειτουργία του λογισμικού, βελτιώνει την απόδοση του, προλαμβάνει τυχόν σφάλματα και μεριμνά για την μείωση του κόστους της ανάπτυξης. Ο Έλεγχος Λογισμικού (Testing) είναι το σύνολο των ενεργειών που απαιτούνται για τον εντοπισμό των ελαττωμάτων και την αξιολόγηση της ποιότητας του λογισμικού. Μια κοινή παρανόηση σχετικά με τον έλεγχο είναι ότι αποτελείται μόνο από την εκτέλεση δοκιμών, ενώ στην πραγματικότητα οι δοκιμές αυτές περιλαμβάνουν κι άλλες δραστηριότητες που πρέπει να εκτελούνται σύμφωνα με τον τρέχων κύκλο ζωής του λογισμικού [2].

Ο έλεγχος πρέπει να πραγματοποιείται σε κάθε φάση του κύκλου ζωής του λογισμικού και όσο τον δυνατόν σε πρώιμο στάδιο της φάσης ώστε να εξασφαλιστεί ότι το προϊόν ικανοποιεί τις απαιτήσεις που έχουν τεθεί κατά την ανάλυση και ότι ο τελικός χρήστης δεν θα αντιμετωπίσει κάποια δυσλειτουργία. Ακόμη, στο στάδιο του ελέγχου, μπορεί να επιβεβαιωθεί ότι πληρούνται σχετικοί νομικοί περιορισμοί ή απαιτήσεις και άλλα κανονιστικά πλαίσια. Επομένως, ο ενδελεχής έλεγχος κατά την ανάπτυξη έχει ως αποτέλεσμα την παράδοση λογισμικού υψηλής ποιότητας με ελάχιστα ή καθόλου σφάλματα, το οποίο ανταποκρίνεται στις ανάγκες και τις προσδοκίες των χρηστών.

1.2. Έλεγχος και Διασφάλιση Ποιότητας (Testing and Quality Assurance)

Οι όροι Έλεγχος (Testing) και Διασφάλιση Ποιότητας (Quality Assurance) χρησιμοποιούνται, λανθασμένα, για να περιγράψουν την ίδια διαδικασία, ωστόσο δεν είναι το ίδιο. Ο Έλεγχος είναι η διαδικασία της Ποιοτικού Ελέγχου (Quality Control), ο οποίος γίνεται στοχευμένα στο εκάστοτε λογισμικό με σκοπό να γίνουν οι απαραίτητες διορθωτικές ενέργειες, εάν χρειάζονται, ώστε να επιτευχθεί το επιθυμητό επίπεδο ποιότητας βάσει των απαιτήσεων. Η Διασφάλιση Ποιότητας είναι η ίδια η διαδικασία που εστιάζει στην εφαρμογή και τη βελτίωση των διαδικασιών, κατά την οποία διασφαλίζεται η συνθήκη όταν μια καλή διαδικασία ακολουθηθεί σωστά, τότε θα δημιουργήσει ένα καλό προϊόν. Ωστόσο, η Διασφάλιση Ποιότητας εφαρμόζεται τόσο στις διαδικασίες ανάπτυξης του λογισμικού όσο και στις διαδικασίες ελέγχου, γι' αυτό είναι ευθύνη όλων όσων εμπλέκονται στο έργο. Αντίθετα, για τον Έλεγχο του λογισμικού είναι υπεύθυνη η ομάδα ελέγχου. Τα αποτέλεσμα του ελέγχου χρησιμοποιούνται για να διορθωθούν τυχόν σφάλματα, ενώ τα αποτελέσματα της διασφάλισης ποιότητας παρέχουν την πληροφορία πόσο καλά εφαρμόζονται οι διαδικασίες ανάπτυξης και ελέγχου του λογισμικού. [2]

1.3. Η Αξία του Ελέγχου του Λογισμικού

Οι περισσότερες εταιρίες γνωρίζουν πόσο σημαντικός είναι ο έλεγχος κατά την διάρκεια του κύκλου ζωής του λογισμικού ώστε να εντοπίσουν και να διορθώσουν τυχόν σφάλματα. Έρευνες έχουν δείξει ότι η διαδικασία του ελέγχου εκτιμάται ότι αντιστοιχεί στο 40% του κόστους της ανάπτυξης και με την συνεχώς αυξανόμενη απαίτηση για ποιότητα και αποτελεσματικότητα, ο έλεγχος είναι αναπόφευκτος. Τα διάφορα μοντέλα κύκλου ζωής έχουν καθοδηγητικό ρόλο για το έργο και λειτουργούν ως ένα ευέλικτο μέσο για την προσαρμογή των αλλαγών και την ομαλή εκτέλεση του έργου με στόχο την επίτευξη των στόχων.

Πέρα από τον εντοπισμό των σφαλμάτων και την διασφάλιση ότι έγιναν οι απαραίτητες διορθώσεις, ο έλεγχος δίνει την πληροφορία στους ενδιαφερόμενους (stakeholders) για το στάδιο και την ποιότητα στο οποίο βρίσκεται το λογισμικό. Επιπλέον, με την κατάλληλη εφαρμογή μεθόδων ελέγχου, επισημαίνονται και προλαμβάνονται διάφορες αστοχίες, που μπορούν προκύψουν κατά την χρήση του λογισμικού, οι οποίες όμως δεν αποτελούν σφάλματα, αλλά χρήζουν βελτίωση. Δεδομένου ότι χρησιμοποιούνται τα βέλτιστα εργαλεία και μέθοδοι για την επαλήθευση και την επικύρωση του λογισμικού, μειώνονται σημαντικά οι πιθανότητες οι χρήστες να εντοπίσουν κάποια δυσλειτουργία κατά

την χρήση του λογισμικού και ως αποτέλεσμα ενισχύεται η αξιοπιστία του για την άριστη χρησιμότητα και λειτουργικότητα. [3]

1.4. Μεθοδολογίες και Μοντέλα Κύκλου Ζωής Λογισμικού (Software Development Lifecycle Methodologies and Models)

Στον σύγχρονο κόσμο ανάπτυξης λογισμικού, η συνεχής αλλαγή των απαιτήσεων, η πίεση για γρήγορη παράδοση και η ανάγκη μείωσης κόστους λόγω ανταγωνισμού είναι κάποιες από τις προκλήσεις που έρχονται αντιμέτωπες οι εταιρίες. Για την αντιμετώπιση αυτών των προκλήσεων, έχουν αναπτυχθεί διάφορες μέθοδοι που στοχεύουν σε μια πιο ευέλικτη προσέγγιση για ταχύτερη παραγωγή ποιοτικών προϊόντων λογισμικού. Ο σχεδιασμός και η παρακολούθηση της διαδικασίας ανάπτυξης λογισμικού με τη χρήση των ευέλικτων μεθοδολογιών είναι θεμελιώδη βήματα προς την εξασφάλιση της επιτυχίας και συμπεριλαμβάνουν την αποτελεσματική διαχείριση των πόρων και τη συνεχή παρακολούθηση της εξέλιξης χρησιμοποιώντας συγκεκριμένες μετρικές. [4] [5]

Το Μοντέλο Κύκλου Ζωής Λογισμικού είναι ένα εννοιολογικό πλαίσιο το οποίο καθορίζει τον πλήρη κύκλο ανάπτυξης, δηλαδή όλα τα στάδια που εμπλέκονται για την ανάπτυξη ενός λογισμικού από την αρχική μελέτη σκοπιμότητας μέχρι την τελική ανάπτυξη και συντήρηση της. Υπάρχουν πολλά μοντέλα που περιγράφουν διάφορες προσεγγίσεις για την ανάπτυξη λογισμικού και καθορίζουν τις διαφορετικές φάσεις ανάπτυξης αλλά και τα είδη δραστηριοτήτων. Ωστόσο, τα τέσσερα βασικά στάδια, όπως αναφέρθηκαν παραπάνω, είναι κοινά σε όλα τα μοντέλα και σχετίζονται μεταξύ τους τόσο λογικά όσο και χρονολογικά. [6]

Παραδείγματα μοντέλων κύκλων ζωής είναι:

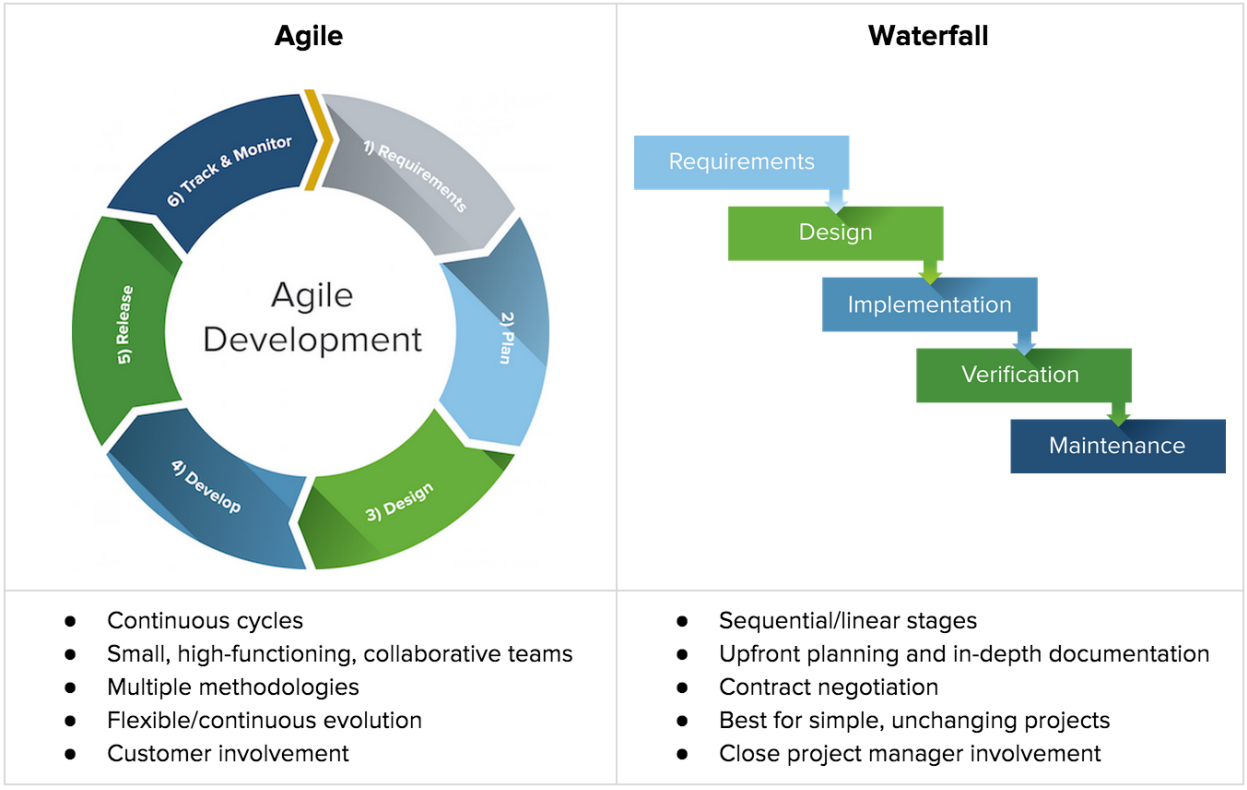
- τα Μοντέλα Διαδοχικής Ανάπτυξης (Sequential Development models)
    - Μοντέλο Καταρράκτη (Waterfall model)
    - Μοντέλο V (V Model)
- τα Μοντέλα Επαναληπτικής Ανάπτυξης (Iterative Development models)
    - Σπειροειδές Μοντέλο (Spiral model)
    - Πρωτότυπο Μοντέλο (Prototyping model)
- τα Μοντέλα Σταδιακής Ανάπτυξης (Incremental Development models)
    - Ενοποιημένη μέθοδος της Rational Software Corp (Rational Unified Process – RUP)
- το Μοντέλο Ανάπτυξης Agile (Agile Software Development model)

- Μεθοδολογία Scrum
- Μεθοδολογία Kanban
- Μεθοδολογία Scrumban

Τα μοντέλα που έχουν επικρατήσει κυρίως στην εποχή μας και χρησιμοποιούνται αποτελεσματικά από τις ομάδες ανάπτυξης λογισμικού είναι το Μοντέλο Καταρράκτη και το Μοντέλο Agile, τα οποία επιλέγονται ανά περίπτωση βάσει των αναγκών κάθε έργου. Κατά την εφαρμογή του Μοντέλου Καταρράκτη στην πράξη όμως, χαρακτηρίστηκε ως μη παραγωγικό και γραφειοκρατικό διότι αποτελείται από πολλές φάσεις, μεγάλης διάρκειας και με αυστηρά προκαθορισμένες διαδικασίες. Καθώς η κάθε φάση πραγματοποιείται μία φορά κατά τον κύκλο ζωής, η ανάλυση ή η υλοποίηση του λογισμικού μπορεί να επαναληφθεί αρκετές φορές εάν αλλάξουν οι απαιτήσεις ή παρατηρηθεί κάποιο

κενό ή πρόβλημα στην αρχική ανάλυση. Οι αδυναμίες αυτές καθιστούν το μοντέλο Καταρράκτη ανελαστικό και γενικότερα με το πέρασμα του χρόνου, παρατηρείται μία συνεχόμενη αποστροφή για τα Μοντέλα Διαδοχικής Ανάπτυξης.

Αντίθετα, οι μεθοδολογίες Agile καλύπτουν την ανάγκη για διαρκής ανάπτυξη και παράδοση λογισμικού λαμβάνοντας υπόψιν διάφορες αλλαγές στις απαιτήσεις. Αυτό συμβαίνει διότι το λογισμικό δεν αναπτύσσεται και παραδίδεται εξ' ολοκλήρου αλλά σε πολλά μέρη (increments), όπου το κάθε μέρος περιλαμβάνει μια νέα λειτουργικότητα/δυνατότητα του συστήματος. Στις μεθόδους σταδιακής ανάπτυξης, το κάθε increment διαρκεί δύο ή τρεις εβδομάδες και ασχολείται με μικρά αντικείμενα εργασίας, τα οποία στο τέλος του increment είναι πλήρως λειτουργικά και παραδίδονται στον πελάτη ως νέα έκδοση. Συχνά, κατά την εφαρμογή του μοντέλου Agile, οι πελάτες εμπλέκονται καθ' όλη την διάρκεια του κύκλου ζωής του λογισμικού έτσι ώστε να εκφέρουν την γνώμη τους, να κατευθύνουν την υλοποίηση βάσει των αναγκών τους και επισημάνουν τυχόν αλλαγές που επιθυμούν όσο το δυνατόν σε πρώιμο στάδιο. [7]



1.5. Ο Έλεγχος στον Κύκλο Ζωής του Λογισμικού

Ο κύριος σκοπός της διαδικασίας του ελέγχου είναι να εντοπιστούν τα σφάλματα και στην συνέχεια να διορθωθούν. Στα πλαίσια του ελέγχου, απαιτείται η εκτέλεση του κώδικα του λογισμικού σε διάφορα περιβάλλοντα και η αξιολόγηση όλων των μερών του ώστε να διασφαλίσει το λογισμικό κάνει αυτό που υποτίθεται ότι πρέπει να κάνει και ότι λειτουργεί βάσει των προδιαγραφών. Ο έλεγχος, ως μια μορφή ποιοτικού ελέγχου, έχει γίνει σημαντικό μέρος του κύκλου ζωής του λογισμικού γι' αυτό συνιστάται να ξεκινά στα αρχικά στάδια. Αυτό όχι μόνο εξυπηρετεί ώστε να αποφευχθούν διορθώσεις σφαλμάτων στα τελικά στάδια, αλλά μειώνει και την πιθανότητα αλλαγές σε ήδη υλοποιημένα μέρη. Από την άλλη, λόγω της σπουδαιότητας του ελέγχου στα διάφορα στάδια του κύκλου ζωής του λογισμικού, η διαδικασία αυτή, πρακτικά, δεν σταματά μέχρι να διασφαλιστεί η μέγιστη δυνατή ποιότητα, και δεδομένου ότι το έργο βρίσκεται εντός χρονικών ορίων και οικονομικού προϋπολογισμού.

Δεδομένου ότι ο κύκλος ζωής του λογισμικού είναι μία αναπαράσταση της διαδικασίας ανάπτυξης του, ο έλεγχος πρέπει να προσαρμόζεται στον κύκλο ζωής για να διεκπεραιωθεί με επιτυχία. Στα μοντέλα διαδοχικής ανάπτυξης, η ομάδα ελέγχου τυπικά συμμετέχει στην ανάλυση απαιτήσεων και την ανάλυση και τον σχεδιασμό του ελέγχου. Ο εκτελέσιμος κώδικας παράγεται σε μετέπειτα στάδια, άρα ο έλεγχος δεν μπορεί να ξεκινήσει σε νωρίτερα στάδια. Αντιθέτως, το μοντέλο Agile προϋποθέτει ότι αλλαγές μπορούν να συμβούν καθ' όλη την διάρκεια του έργου. Επομένως, τόσο η τεκμηρίωση του έργου όσο και η εκτεταμένη αυτοματοποίηση ελέγχων (automation testing) συνιστώνται για την διευκόλυνση του regression testing, καθώς επίσης οι περισσότεροι χειροκίνητοι έλεγχοι (manual testing) συνήθως πραγματοποιούνται με την χρήση ελέγχων που βασίζονται στην εμπειρία και δεν απαιτούν εκτενή ανάλυση και σχεδιασμό. [2] [3]

2. Έλεγχος & Διασφάλιση Ποιότητας

2.1. Τι είναι Έλεγχος;

Η χρήση λογισμικών αποτελεί αναπόσπαστο κομμάτι της καθημερινότητας της σύγχρονης κοινωνίας ωστόσο αρκετές φορές οι άνθρωποι έρχονται αντιμέτωποι με λογισμικά που δεν λειτουργούν όπως αναμένεται. Τα προβλήματα, που προκαλούνται από μερικώς λειτουργικά λογισμικά, μπορεί να είναι η απώλεια χρημάτων, χρόνου, ακόμη και σε ακραίες περιπτώσεις τραυματισμοί ή θάνατοι. Γι' αυτό κρίνεται απαραίτητο να γίνονται έλεγχοι και δοκιμές στο λογισμικό οι οποίοι αξιολογούν την ποιότητα του και συμβάλλουν στην μείωση αστοχιών λογισμικού κατά την λειτουργία.

Ο Έλεγχος λογισμικού είναι ένα σύνολο δραστηριοτήτων για την εύρεση ελαττωμάτων και την αξιολόγηση της ποιότητας του λογισμικού. Μια κοινή παρανόηση σχετικά με τον έλεγχο είναι ότι αποτελείται μόνο από την εκτέλεση δοκιμών (δηλαδή, εκτέλεση του λογισμικού και έλεγχος των αποτελεσμάτων). Ωστόσο, ο έλεγχος λογισμικού περιλαμβάνει και άλλες δραστηριότητες και πρέπει να συμβαδίζει με τον κύκλο ζωής ανάπτυξης λογισμικού.

Επομένως, ο έλεγχος λογισμικού διασφαλίζει εάν κατά την εκτέλεση ενός τμήματος ή ολόκληρου του λογισμικού έχουν υλοποιηθεί οι αναμενόμενες απαιτήσεις και ανάγκες των χρηστών, καθώς και ότι δεν υπάρχουν σφάλματα ή ελαττώματα που να επηρεάζουν την λειτουργικότητα του. Για τον έλεγχο του λογισμικού πραγματοποιούνται δοκιμές είτε χειροκίνητα είτε χρησιμοποιώντας αυτοματοποιημένα εργαλεία. Οι δοκιμές αυτές σχετίζονται με τις έννοιες της επαλήθευσης (verification) και της επικύρωσης (validation). Η επαλήθευση αξιολογεί ότι το λογισμικό πληροί τις απαιτήσεις που έχουν οριστεί κατά την ανάλυση ενώ η επικύρωση αξιολογεί ότι το λογισμικό εξυπηρετεί και καλύπτει τις ανάγκες των χρηστών. [2]

2.2. Τι είναι Διασφάλιση Ποιότητας;

Η διαδικασία της Διασφάλισης Ποιότητας είναι μια προληπτική προσέγγιση που εστιάζει στην εφαρμογή πρακτικών για την βελτίωση του λογισμικού. Η ποιότητα του λογισμικού εξετάζεται από διαφορετικές οπτικές, με βάση τους διαφορετικούς ρόλους και ευθύνες που δημιουργούν διαφορετικές προσδοκίες. Ο καθορισμός της ποιότητας μπορεί να προσδιοριστεί βάσει των παρακάτω προσεγγίσεων:

- Την υπερβατική (transcendent) προσέγγιση όπου η ποιότητα είναι δύσκολο να οριστεί ή να περιγραφεί, αλλά μπορεί να αναγνωριστεί αν υπάρχει. Συνήθως συνδέεται με την ικανοποίηση των χρηστών.
- Την προσέγγιση της ποιότητας με βάση το προϊόν (product-based) όπου επικεντρώνεται στα εσωτερικά και εγγενή χαρακτηριστικά του προϊόντος. Ο έλεγχος αυτών των εσωτερικών δεικτών ποιότητας μπορεί να οδηγήσει σε βελτιωμένη συμπεριφορά του εξωτερικού προϊόντος.
- Την προσέγγιση της ποιότητας με βάση την κατασκευή (manufacturing-based) κατά την οποία η ποιότητα σημαίνει η συμμόρφωση με τα πρότυπα διεργασίας και σχετίζεται με τον καθορισμό των απαιτήσεων, των προδιαγραφών και της τεχνολογίας μέσα στην εταιρία.
- Την προσέγγιση της ποιότητας με βάση τον χρήστη (user-based) η οποία βασίζεται στα χαρακτηριστικά του προϊόντος και αφορά στην ικανοποίηση των αναγκών και προσδοκιών του χρήστη.
- Την προσέγγιση της ποιότητας με βάση την αξία (value-based) που αφορά το κόστος και την τιμή και σχετίζεται με έναν άλλο καθοριστικό ορισμό της ποιότητας που αναφέρει ότι «Η ποιότητα είναι όλα τα χαρακτηριστικά που επιτρέπουν σε ένα προϊόν να ικανοποιεί δηλωμένες ή σιωπηρές ανάγκες με προσιτό κόστος».

Συμπερασματικά, η ποιότητα είναι ο βαθμός στον οποίο το λογισμικό διαθέτει τα επιθυμητά χαρακτηριστικά και πληροί συγκεκριμένες απαιτήσεις. Ωστόσο, η ποιότητα του λογισμικού μπορεί να προσδιοριστεί και ποσοτικά, για παράδειγμα βάσει της user-based προσέγγισης, ένας παράγοντας της ποιότητας είναι η λειτουργικότητα. Ένα λογισμικό έχει τουλάχιστον μία λειτουργικότητα στην οποία αντιστοιχεί τουλάχιστον μία δοκιμαστική περίπτωση (test case). Επομένως ο λόγος του αριθμού των δοκιμαστικών περιπτώσεων που ήταν επιτυχείς προς τον συνολικό αριθμό των δοκιμαστικών περιπτώσεων είναι ένα μέτρο ποιότητας των λειτουργικοτήτων. [8]

2.3. Αρχές Ελέγχου Λογισμικού

Με την πάροδο των χρόνων, έχουν οριστεί και ακολουθούνται μερικές βασικές αρχές, οι οποίες εφαρμόζονται σε κάθε είδους ελέγχου λογισμικού και περιγράφονται παρακάτω [2]:

- **Ο έλεγχος αναδεικνύει την παρουσία σφαλμάτων και όχι την απουσία τους**: Ο έλεγχος μπορεί να δείξει ότι υπάρχουν σφάλματα στο αντικείμενο δοκιμής, αλλά δεν μπορεί να

αποδείξει ότι δεν υπάρχουν (Buxton 1970). Με άλλα λόγια, ο έλεγχος μειώνει την πιθανότητα να παραμείνουν σφάλματα στο λογισμικό, τα οποία δεν έχουν ανακαλυφθεί.

- **Ο εξαντλητικός έλεγχος είναι αδύνατος:** Δεν είναι εφικτό να ελεγχθεί πλήρως το αντικείμενο ελέγχου, εκτός εάν πρόκειται για τετριμμένες περιπτώσεις (Manna 1978). Αντί να επιχειρείται εξαντλητικός έλεγχος, θα πρέπει να χρησιμοποιούνται τεχνικές ελέγχων, ιεράρχηση προτεραιοτήτων δοκιμαστικών περιπτώσεων και δοκιμές βάσει κινδύνου που επικεντρώνονται στις προσπάθειες ελέγχου.
- **Ο πρώιμος έλεγχος εξοικονομεί χρόνο και χρήμα:** Όταν τα σφάλματα διορθώνονται σε πρώιμο στάδιο, εξασφαλίζεται ότι δεν θα προκαλέσουν επικείμενα ελαττώματα σε παράγωγα προϊόντα του λογισμικού. Το κόστος της ποιότητας θα μειωθεί καθώς λιγότερα σφάλματα θα προκύψουν σε επόμενο στάδιο του κύκλου ζωής του λογισμικού (Boehm 1981). Για να εντοπιστούν έγκαιρα τα σφάλματα, θα πρέπει ο έλεγχος να ξεκινήσει όσο το δυνατόν νωρίτερα.
- **Τα σφάλματα βρίσκονται συγκεντρωμένα:** Τα περισσότερα σφάλματα εντοπίζονται σε ένα μικρό αριθμό ενοτήτων του συστήματος (Enders 1975). Το φαινόμενο αυτό είναι η εφαρμογή της αρχής Pareto – περίπου το 80% των προβλημάτων εντοπίζονται στο 20% των ενοτήτων του συστήματος.
- **Οι έλεγχοι φθείρονται:** Εάν οι ίδιοι έλεγχοι επαναλαμβάνονται πολλές φορές, σταματούν να είναι αποτελεσματικοί στην εύρεση νέων σφαλμάτων (Beizer 1990). Γι’ αυτό οι έλεγχοι καθώς και τα test data πρέπει να ενημερώνονται και να εμπλουτίζονται συνεχώς. Ωστόσο σε κάποιες περιπτώσεις, η επανάληψη των ίδιων ελέγχων μπορεί να είναι αποτελεσματική π.χ. κατά το αυτοματοποιημένο regression testing.
- **Ο έλεγχος εξαρτάται από το ευρύτερο πλαίσιο:** Για κάθε τύπος λογισμικού πρέπει να πραγματοποιηθούν διαφορετικοί τύποι ελέγχων (Kaner 2011). Γι’ αυτό κάθε προσέγγιση εξαρτάται από γενικό πλαίσιο του λογισμικού που αναπτύσσεται και δεν υπάρχει μια καθολική εφαρμογή ελέγχων.
- **Η απουσία σφαλμάτων είναι παραπλανητική:** Θα ήταν παρανόηση εάν μόνο με την επαλήθευση του λογισμικού διασφαλιστεί η επιτυχία του. Ο λεπτομερής έλεγχος των καθορισμένων απαιτήσεων και η διόρθωση όλων των σφαλμάτων, δημιουργεί ένα άρτιο λειτουργικά σύστημα, το οποίο όμως πιθανόν να μην ικανοποιηθεί τις ανάγκες και τις προσδοκίες των χρηστών ή να μην βοηθά στην επίτευξη επιχειρηματικών στόχων του πελάτη. Εκτός από την επαλήθευση, θα πρέπει επίσης να πραγματοποιείται και επικύρωση (Boehm 1981).

2.4. Κύκλος Ζωής Διαδικασίας Ελέγχου Λογισμικού

Η διαδικασία που ακολουθείται για τον έλεγχο του λογισμικού εξαρτάται από το είδος του αλλά σε κάθε περίπτωση, εφαρμόζεται μια κοινή διαδικασία που περιλαμβάνει συγκεκριμένες δραστηριότητες, οι οποίες είναι προσαρμοσμένες ανάλογα με τις ανάγκες του λογισμικού. Παρόλο που οι δραστηριότητες φαίνονται ότι ακολουθούν μία λογική σειρά, συχνά πραγματοποιούνται επαναληπτικά ή παράλληλα. Οι δραστηριότητες που απαιτούνται για την επαλήθευση και επικύρωση των απαιτήσεων καθώς και για τον εντοπισμό και την διόρθωση σφαλμάτων, είτε χρησιμοποιώντας αυτοματοποιημένα εργαλεία είτε με χειροκίνητες δοκιμές, αποτελούν τις φάσεις του κύκλου ζωής ελέγχου του λογισμικού. Ο κύκλος ζωής ελέγχου λογισμικού αποτελείται από την φάση της ανάλυσης, του σχεδιασμού, της προετοιμασίας, της εκτέλεσης και της ολοκλήρωσης. Οι κυριότερες μέθοδοι ελέγχου είναι ο Λειτουργικός (Functional) έλεγχος, ο έλεγχος Απόδοσης (Performance) και ο έλεγχος Ασφαλείας (Security) και κάθε φάση του κύκλου ζωής ελέγχου λογισμικού προσαρμόζεται κατάλληλα σε κάθε μέθοδο. [2] [9]

2.4.1. Η φάση του Προγραμματισμού

Ο Προγραμματισμός καθορίζει το πλάνο (test plan) που θα ακολουθηθεί, τους στόχους του ελέγχου και την προσέγγιση που θα χρησιμοποιηθεί έτσι ώστε να επιτευχθούν οι στόχοι εντός των περιορισμών που έχουν τεθεί στα πλαίσια του έργου. Το test plan είναι ουσιαστικά η πρώτη διεργασία που απαιτείται να γίνει δεδομένου ότι περιγράφονται λεπτομερώς το εύρος των δοκιμών, τους στόχους, τα χαρακτηριστικά (features) που πρέπει και αυτά που δεν πρέπει να ελεγχθούν, οι τύποι και οι μέθοδοι ελέγχου που θα εφαρμοστούν, οι ρόλοι των μελών της ομάδας και οι αντίστοιχες αρμοδιότητες και ευθύνες κάθε μέλους, τα entry και exit κριτήρια και διάφορες υποθέσεις που πρέπει να ληφθούν υπόψιν. [2] [9]

2.4.2. Η φάση της Παρακολούθησης και του Ελέγχου

Η φάση της Παρακολούθησης της διαδικασίας περιλαμβάνει συνεχή έλεγχο όλων των δραστηριοτήτων και την σύγκριση της πραγματικής προόδου με το αρχικό πλάνο. Ταυτόχρονα

διεξάγεται και η φάση του Ελέγχου της διαδικασίας, κατά την οποία λαμβάνονται όλες οι αποφάσεις και οι πραγματοποιούνται οι αντίστοιχες ενέργειες για την επίτευξη τω στόχων. [2] [9]

2.4.3. Η φάση της Ανάλυσης

Η ομάδα ελέγχου εμπλέκεται στα πρώτα κιόλας στάδια της ανάπτυξης του λογισμικού όπου γίνεται η ανάλυση των απαιτήσεων και αργότερα κατά το στάδιο του σχεδιασμού και ανάπτυξης του λογισμικού. Η πιο βασική φάση του κύκλου ζωής ελέγχου του λογισμικού είναι η Ανάλυση, η οποία περιλαμβάνει την ανάλυση των λειτουργικών και μη λειτουργικών απαιτήσεων π.χ. τις επιχειρησιακές απαιτήσεις (business requirements), τις λειτουργικές και τεχνικές προδιαγραφές. Οι μη λειτουργικές απαιτήσεις χαρακτηρίζονται από τις έννοιες της χρηστικότητας, τις επεκτασιμότητας, της δυνατότητας ελέγχου και δοκιμών, την δυνατότητα της συντήρησης, της απόδοσης και της ασφάλειας. Ωστόσο υπάρχουν και απαιτήσεις που δεν μπορούν ελεγχθούν λόγω διαφόρων περιορισμών του συστήματος ή του περιβάλλοντος, σε αυτές τις περιπτώσεις το γεγονός αυτό θα πρέπει να επικοινωνηθεί στην αρμόδια ομάδα που είναι υπεύθυνη για τα business requirements.

Επομένως σε αυτή την φάση, η ομάδα προσδιορίζει τα features που πρέπει να ελεγχθούν και προσδιορίζει τις συνθήκες κάτω από τις οποίες θα γίνει ο έλεγχος, καθώς επίσης θέτει τις προτεραιότητες των δοκιμών.. Επιπλέον, λαμβάνει υπόψιν διάφορα ρίσκα που μπορεί να προκύψουν κατά την διαδικασία. Γενικότερα, η ανάλυση απαντά στο ερώτημα «τι πρέπει να ελεγχθεί;» στα πλαίσια του έργου και των κριτηρίων που πρέπει να ικανοποιηθούν. [2] [9]

2.4.4. Η φάση του Σχεδιασμού

Η φάση της Σχεδιασμού ελέγχου εξετάζει και αναλύει τις απαιτήσεις που έχουν οριστεί, περιλαμβάνει την προετοιμασία των δοκιμών (test cases) που πρέπει να εκτελεστούν, των δεδομένων για τις δοκιμές (test data) και του περιβάλλοντος των δοκιμών (test environment).

Ως περίπτωση δοκιμής (test case) ορίζεται μια σειρά ενεργειών που περιγράφει αναλυτικά τα βήματα που χρειάζονται να εκτελεστούν για την δοκιμή μιας λειτουργικότητας η οποία έχει ένα πραγματικό και ένα αναμενόμενο αποτέλεσμα. Το αναμενόμενο αποτέλεσμα είναι αυτό που έχει προσδιοριστεί κατά την ανάλυση απαιτήσεων – αυτό που θα πρέπει να κάνει το λογισμικό – ενώ το πραγματικό είναι αυτό που

κάνει εν τέλει. Για κάθε απαίτηση προετοιμάζονται τουλάχιστον δύο test cases, μία θετική περίπτωση (positive test case) στην οποία επιβεβαιώνεται η ορθή λειτουργικότητα και μία αρνητική περίπτωση (negative test case) στην οποία επιβεβαιώνεται ότι οποιαδήποτε μη έγκυρη ενέργεια δεν προκαλεί σφάλμα στο λογισμικό. Για να εξασφαλιστεί ότι τα test cases έχουν καλύψει όλες τις περιπτώσεις και ότι ο έλεγχος έχει πραγματοποιηθεί για το 100% της λειτουργικότητας, χρησιμοποιείται ο πίνακας ιχνηλασιμότητας απαιτήσεων (Requirement Traceability Matrix – RTM). Ο Requirement Traceability Matrix προσδιορίζει την σχέση μεταξύ των απαιτήσεων και των λειτουργιών του λογισμικού και χρησιμοποιείται για να εξασφαλίσει ότι πληρούνται οι απαιτήσεις και τυπικά τεκμηριώνει τις απαιτήσεις, τα test cases, τα αποτελέσματα ελέγχων και τυχόν σφάλματα.

Όσον αφορά τα περιβάλλοντα (test environments), στα οποία θα εκτελούνται οι δοκιμές του λογισμικού, θα πρέπει να είναι σύμφωνα με τις hardware αλλά και software απαιτήσεις. Η προετοιμασία των test environments είναι, επίσης, μία από τις πιο σημαντικές ενέργειες αυτής της φάσης εφόσον σε αυτά θα εκτελεστούν τα μέρη του κώδικα ανάπτυξης που έχουν ήδη ολοκληρωθεί και είναι έτοιμα προς έλεγχο. [2] [9]

2.4.5. Η φάση της Εφαρμογής

Σε αυτή την φάση, η ομάδα ελέγχου φροντίζει να δημιουργήσει τις κατάλληλες προϋποθέσεις (π.χ. test data) για την εκτέλεση των δοκιμών. Συχνά τα test cases κατηγοριοποιούνται σε διαδικασίες (test procedures) οι οποίες ιεραρχούνται και ταξινομούνται εντός του χρονοδιαγράμματος της διαδικασίας του ελέγχου. Επιπλέον, τα test cases μπορούν να ομαδοποιηθούν και να ταξινομηθούν έτσι ώστε να δημιουργήσουν σύνολα από test cases (test suites), τα οποία εκτελούνται βάσει του test plan. Τέλος, η ομάδα επιβεβαιώνει ότι τα test environments έχουν δημιουργηθεί και ρυθμιστεί σωστά για την διαδικασία του ελέγχου. [2] [9]

2.4.6. Η φάση της Εκτέλεσης

Σε αυτή την φάση, η ομάδα ελέγχου εκτελεί – χειροκίνητα ή με αυτοματοποιημένα εργαλεία – τα test cases σύμφωνα με το χρονοδιάγραμμα, δηλαδή διεξάγονται όλες οι ενέργειες – βήματα που απαιτούνται ώστε να γίνουν οι κατάλληλες δοκιμές στο λογισμικό και καταγράφονται τα αντίστοιχα αποτελέσματα. Στην περίπτωση που το πραγματικό αποτέλεσμα του ελέγχου δεν συμφωνεί με το αναμενόμενο, τότε έχει

εντοπιστεί ένα σφάλμα. Για την διόρθωση των σφαλμάτων, πρέπει να ακολουθηθεί ο πλήρης κύκλος ζωής σφάλματος, όπως να γίνει η σωστή καταγραφή του στο αντίστοιχο σύστημα (tracking tool) με τις απαραίτητες πληροφορίες, να αναλυθεί ώστε να εντοπιστούν οι πιθανές αιτίες κλπ. [2] [9]

2.4.7. Η φάση της Ολοκλήρωσης

Οι δραστηριότητες της φάσης της Ολοκλήρωσης σηματοδοτούν διάφορα ορόσημα του έργου (όπως την παράδοση του συγκεκριμένου μέρους λογισμικού, την ολοκλήρωση του increment, την ολοκλήρωση ελέγχου ενός μέρους του λογισμικού) και περιλαμβάνουν όλες τις αναφορές για τα αποτελέσματα των δοκιμών διασφαλίζοντας ότι όλοι έλεγχοι έχουν διεκπεραιωθεί και δεν υπάρχει κάποιο κρίσιμο σφάλμα που εκκρεμεί για διόρθωση. Οι τελικές αναφορές κοινοποιούνται στους άμεσα stakeholders και αυτοί με την σειρά του λαμβάνουν αποφάσεις βάσει των αποτελεσμάτων. Τέλος, αναλύονται και αξιολογούνται όλες οι φάσεις της διαδικασίας ώστε να εξεταστεί τι πήγε καλά και τι όχι, τι χρήζει βελτίωση, η δραστηριότητα αυτή ονομάζεται Root Cause Analysis (RCA). [2] [9]

2.5. Επίπεδα Ελέγχου (Test Levels)

Τα Επίπεδα Ελέγχου (Test Levels) είναι σετ δραστηριοτήτων, οι οποίες οργανώνονται και διαχειρίζονται από κοινού. Κάθε test level αποτελεί ένα μέρος της διαδικασίας ελέγχου και ανάλογα με το στάδιο ανάπτυξης που βρίσκεται το λογισμικό. Η κύρια ιδέα πίσω από αυτή την λογική είναι ότι κάθε test level στοχεύει σε συγκεκριμένες πτυχές της λειτουργικότητας και έτσι επιτρέπεται η καλύτερη διασφάλιση ποιότητας και λιγότερα πιθανά σφάλματα.

Τα test levels σχετίζονται με άλλες δραστηριότητες του κύκλου ζωής του λογισμικού και στα μοντέλα διαδοχικής ανάπτυξης συχνά ορίζονται με τέτοιο τρόπο έτσι ώστε τα exit κριτήρια ενός επιπέδου να αποτελούν μέρος των entry κριτηρίων για το επόμενο επίπεδο. Αντίθετα, αυτό μπορεί να μην συμβαίνει στα μοντέλα επαναληπτικής ανάπτυξης. Δεδομένου ότι οι δραστηριότητες ανάπτυξης μπορεί να εκτείνονται σε περισσότερα από ένα test level, τα test levels ενδέχεται να επικαλύπτονται χρονικά. Φυσικά, το λογισμικό πρέπει να περάσει από όλα τα test levels, όπως περιγράφονται παρακάτω, για την διασφάλιση της ποιότητας του. [2] [10]

- **Unit / Component Testing:** επικεντρώνεται σε με ένα μεμονωμένο μέρος ή module του λογισμικού. Πιο συγκεκριμένα, γίνεται έλεγχος στο μικρότερο δυνατό τμήμα του λογισμικού για να επαληθευτεί η λειτουργικότητα του σε σχέση με τις προδιαγραφές του. Συνήθως το component testing πραγματοποιείται από τους προγραμματιστές στα δικά του περιβάλλοντα ανάπτυξης (development environments).
- **Integration Testing:** περιλαμβάνει τον έλεγχο δύο ή παραπάνω μερών, τα οποία πρέπει να δουλεύουν μεταξύ τους ομαλά για να διασφαλιστεί ότι δεν υπάρχουν σφάλματα στην διαδικασία διασύνδεσης τους και στον συνολικό σχεδιασμό τους. Ο έλεγχος αυτό πραγματοποιείται από την ομάδα ελέγχου.
- **System Testing:** αφορά τον έλεγχο της εξολοκλήρου λειτουργίας και των δυνατοτήτων του λογισμικού και συχνά περιλαμβάνει functional testing σε end-to-end διεργασίες και non-functional testing των ποιοτικών χαρακτηριστικών. Την διεξαγωγή του system testing μπορεί να την αναλάβει ακόμη και μια ανεξάρτητη ομάδα ελέγχου.
- **Acceptance Testing:** εκτελείται για να επικυρώσει ότι το λογισμικό ικανοποιεί τις απαιτήσεις και τις επιχειρηματικές ανάγκες των χρηστών. Το User Acceptance Testing (UAT), όπως αποκαλείται συνήθως, πραγματοποιείται από μια ομάδα χρηστών του πελάτη που είτε πρόκειται να το χρησιμοποιήσουν οι ίδιοι ως τελικοί χρήστες είτε πρόκειται να χρησιμοποιηθεί από τους δικούς τους τελικούς πελάτες – χρήστες.

2.6. Τύποι Ελέγχου (Test Types)

Οι Τύποι Ελέγχου (test types) είναι σετ δραστηριοτήτων που σχετίζονται με συγκεκριμένα ποιοτικά χαρακτηριστικά και οι δραστηριότητες αυτές μπορούν να εκτελούνται σε κάθε test level. Ωστόσο είναι αδύνατο να εκτελεστούν όλοι οι τύποι ελέγχου σε ένα λογισμικό λόγω των χρονικών περιορισμών. Σύμφωνα με έρευνες, έχει διαπιστωθεί ότι θα πρέπει να εκτελείται ένας σωστός συνδυασμός όλων των τύπων ελέγχων ώστε να διασφαλιστεί ποιοτικό και συνολικά αξιόπιστο λογισμικό. Υπάρχουν αρκετοί τύποι ελέγχων που μπορούν να εφαρμοστούν, και στις παρακάτω ενότητες περιγράφονται αναλυτικά οι τέσσερις πιο διαδεδομένοι. [2] [9]

2.6.1. Functional Testing

Ο κύριος παράγοντας που καθορίζει την ποιότητα του λογισμικού είναι πληρούνται οι προϋποθέσεις της λειτουργικότητας και της συμπεριφοράς του. Το λειτουργικό μέρος του λογισμικού περιλαμβάνει την εξωτερική συμπεριφορά του που κυρίως προσδιορίζει όλες τις απαιτήσεις των χρηστών. Ο σχεδιασμός του πραγματοποιείται έτσι ώστε ο πελάτης να είναι ικανοποιημένος στα πρώιμα στάδια του σχεδιασμού και της ανάπτυξης. Ο σκοπός του Functional Testing είναι να ελέγξει κάθε λειτουργία του λογισμικού τόσο στις βασικές διαδικασίες όσο και στις πιο εναλλακτικές διαδικασίες – δηλαδή αυτές που δεν είναι αναμενόμενες για την καλή χρήση του λογισμικού. Οι διαδικασίες αυτές μπορούν να αναπαρασταθούν από διάφορα use case διαγράμματα, ενώ τα αυτοματοποιημένα test cases δημιουργούνται από μοντέλα UML. Τα test cases βασίζονται από τις προδιαγραφές και τις απαιτήσεις, όπως «τι υποτίθεται πρέπει να κάνει το λογισμικό». Ως εκ τούτου, το Functional Testing εστιάζει στο «Τι» πρέπει να κάνει το λογισμικό και όχι στο «Πως».

Υπάρχουν πολλοί τύποι Functional Testing μεθόδων και τεχνικών, οι οποίοι μπορούν να εκτελεστούν σε διάφορα test levels όπως, Unit testing, Integration testing, System testing, Acceptance testing, White-box testing και Black-box testing. [9]

2.6.2. Non-Functional Testing

Ο σκοπός του Non-Functional Testing είναι να ελέγχονται οι μη-λειτουργικές απαιτήσεις, όπως η αξιοπιστία, η ευχρηστία, ασφάλεια, η απόδοση, η ευελιξία. Η διεξαγωγή του συνιστάται σε πρώιμα στάδια, εφόσον πολλοί μη-λειτουργικές δοκιμές προέρχονται από λειτουργικές δοκιμές που ελέγχουν άλλους περιορισμούς (π.χ. μια λειτουργία ολοκληρώνεται σε συγκεκριμένο χρόνο). Τα μη-λειτουργικά σφάλματα αποτελούν σημαντικό παράγοντα για την επιτυχία ενός έργου, γι' αυτό επιβάλλεται να εντοπιστούν και να διορθωθούν όσο το δυνατόν νωρίτερα. Οι μη-λειτουργικές απαιτήσεις ελέγχονται με την χρήση μη έγκυρων ή μη αναμενόμενων δεδομένων, επομένως αυτού του είδους ο έλεγχος απαντά στο ερώτημα «πόσο καλά συμπεριφέρεται το λογισμικό».

Υπάρχουν ειδικές μέθοδοι και τα αντίστοιχα εργαλεία για να πραγματοποιηθεί Non-Functional Testing, όπως Performance testing, Load testing, Stress testing, Security testing, Stability testing, Usability testing, Compatibility testing. [11]

2.6.3. Black-box Testing

Ο όρος «Black-box testing» υποδηλώνει ότι ο έλεγχος βασίζεται στις προδιαγραφές και αντλεί τα σενάρια ελέγχου από την τεκμηρίωση και τα εγχειρίδια χρήσης του λογισμικού. Ο κύριος σκοπός του Black-box testing είναι ο έλεγχος της συμπεριφοράς του λογισμικού. Σε αυτόν τον τύπο ελέγχου, η ομάδα δεν έχει καμία γνώση σχετικά με τον κώδικα που έχει αναπτυχθεί και τον τρόπο που έχει σχεδιαστεί. Το μόνο, που γνωρίζει η ομάδα ελέγχου, είναι οι απαιτούμενες προδιαγραφές και τα αναμενόμενα αποτελέσματα. Συνεπώς για να επαληθευτεί η απαιτούμενη συμπεριφορά, γίνονται δοκιμές με διάφορα δεδομένα εισόδου (test data) ώστε να ελεγχθεί εάν το σύστημα αποτυγχάνει χρησιμοποιώντας μη έγκυρα δεδομένα.

Υπάρχουν διάφορες τεχνικές που ακολουθούνται για την διεξαγωγή Black-box testing όπως, Decision Making testing, Error guessing, State transition testing, All-pairs testing, Equivalence partitioning, Cause-effect graph, Boundary value analysis, Use case testing. [12]

2.6.4. White-box Testing

Σε αυτόν τον τύπο ελέγχου, ο έλεγχος βασίζεται στην δομή του κώδικα, επομένως οι δοκιμές πραγματοποιούνται από την αντίστοιχη ομάδα έχοντας γνώση του κώδικα, π.χ. λεπτομέρειες για την υλοποίηση ή την δομή του κώδικα. Γι' αυτό, το White-box testing είναι απόλυτα παραγωγικό στον εντοπισμό σφαλμάτων και στην επίλυση προβλημάτων που προκαλούνται από αυτά τα σφάλματα. Δεδομένου ότι τα σενάρια ελέγχου εξαρτώνται από το πως έχει σχεδιαστεί το λογισμικό, η δημιουργία τους μπορεί να γίνει μόνο μετά την υλοποίηση του λογισμικού ή του μέρους του λογισμικού που πρόκειται να ελεγχθεί. Επιπλέον για την διεξαγωγή White-box testing, απαιτείται πλήρης γνώση κώδικα και κατανόηση στο πως συνδέονται και επικοινωνούν τα components μεταξύ τους.

Οι πιο διαδεδομένες τεχνικές White-box testing είναι Decision coverage, Prime path testing, Data flow testing, Statement coverage, Control flow testing, Modified condition/decision coverage, Branch testing, Path testing. Ωστόσο υπάρχουν και πιο αυστηρές τεχνικές που χρησιμοποιούνται για να καλύψουν ενδελεχώς την ποιότητα του κώδικα, κυρίως σε περιβάλλονται όπου η ασφάλεια, ο σκοπός και η ακεραιότητα είναι κρίσιμα. Τέλος, υπάρχουν τεχνικές οι οποίες χρησιμοποιούνται σε υψηλότερα test levels (π.χ. API testing) ή δεν σχετίζονται με κώδικα (π.χ. Neural Network testing). [12]

2.7. Τεχνικές Ελέγχου (Test Techniques)

Οι Τεχνικές Ελέγχου (Test Techniques) βοηθούν την ομάδα ελέγχου στην φάση ανάλυσης (τι να ελέγξει) και του σχεδιασμού (πως να το ελέγξει) της διαδικασίας του ελέγχου του λογισμικού. Οι διάφορες τεχνικές βοηθούν στην ανάπτυξη ενός μικρού αλλά επαρκούς συνόλου test cases καθώς και την ομάδα ελέγχου να καθορίσει τα test conditions και τα test data. Οι τεχνικές ελέγχου κατηγοριοποιούνται ως Black-box, White-box και Experience-based.

2.7.1. Black-box Test Techniques

Οι Black-box τεχνικές ελέγχου που χρησιμοποιούνται ευρέως είναι:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing

2.7.1.1. Equivalence Partitioning

Η Equivalence Partitioning είναι μια τεχνική για τον σχεδιασμό test cases, η οποία διαχωρίζει τα δεδομένα σε ισοδύναμα μέρη (equivalence partitions). Με αυτή την τεχνική, το σύνολο των αποδεκτών τιμών (test inputs) που μπορούν να εισαχθούν στο λογισμικό διαχωρίζονται σε υποσύνολα, καθένα από τα οποία αντιπροσωπεύει τα test inputs με παρόμοια χαρακτηριστικά και προδιαγραφές. Κάθε equivalence partition αποτελείται από έγκυρές ή μη έγκυρες τιμές βάσει των περιορισμών για τις τιμές που επιτρέπονται να εισαχθούν στο λογισμικό κατά τον έλεγχο. Από κάθε partition, τα test cases επιλέγονται έτσι ώστε οι περισσότερες ιδιότητες του partition να ελέγχονται ταυτόχρονα. Η θεωρία πίσω από αυτή την τεχνική είναι ότι αν ένα test case, που ελέγχει μια τιμή από ένα partition, εντοπίσει ένα σφάλμα, τότε αυτό το σφάλμα θα πρέπει να εντοπίζεται από test cases που ελέγχουν οποιαδήποτε άλλη τιμή του ίδιου partition. Αυτό οφείλεται στο γεγονός ότι όλα τα στοιχεία ενός partition επεξεργάζονται με τον ίδιο τρόπο από το λογισμικό. Επομένως, ο αριθμός των test cases μειώνεται σημαντικά αφού ένας μόνο έλεγχος από κάθε partition είναι αρκετός και αποτελεσματικός. [13]

2.7.1.2. Boundary Value Analysis

Η τεχνική Boundary Value Analysis βασίζεται στον έλεγχο των ορίων των equivalence partitions, γι' αυτό και χαρακτηρίζεται ως η επέκταση της τεχνικής Equivalence Partitioning, ωστόσο μπορεί να χρησιμοποιηθεί μόνο όταν τα partitions προσδιορίζονται και αποτελούνται από αριθμητικές ή διαδοχικές τιμές. Η ελάχιστη και η μέγιστη τιμή για ένα partition είναι οι τιμές των ορίων (boundary values). Στην περίπτωση της Boundary Value Analysis, εάν δύο τιμές ανήκουν στο ίδιο partition, τότε όλες οι τιμές μεταξύ αυτών των δύο ανήκουν σε αυτό το partition. Η τεχνική αυτή επικεντρώνεται στις τιμές των ορίων επειδή είναι πολύ συχνό φαινόμενο να υπάρχουν σφάλματα στο λογισμικό που σχετίζονται με τα επιτρεπτά όρια. Συνήθως, τέτοιου είδους σφάλματα βρίσκονται στις περιπτώσεις που υλοποιούνται λειτουργίες με όρια, τα οποία έχουν καθοριστεί εσφαλμένα κατά την ανάπτυξη του λογισμικού. Υπάρχουν δύο εκδοχές της τεχνικής Boundary Value Analysis, η 2-value και η 3-value BVA, όπου διαφέρουν ως προς τας στοιχεία κάλυψης ανά όριο που πρέπει να εκτελεστούν ώστε να επιτευχθεί 100% κάλυψη. [13] [2]

Στην 2-value BVA, για κάθε όριο του εύρους τιμών υπάρχουν δύο στοιχεία κάλυψης, η τιμή του ορίου και ο πλησιέστερος γείτονας του επόμενου partition. Για παράδειγμα, αν η συνθήκη είναι $x \leq 10$, τότε τα test cases θα εκτελεστούν για $x = 10$ και $x = 11$. Για να επιτευχθεί 100% κάλυψη, τα test cases πρέπει να εφαρμόζονται για όλα τα στοιχεία κάλυψης, δηλαδή όλα τα καθορισμένα όρια. Ενώ στην 3-value BVA, για κάθε όριο του εύρους τιμών υπάρχουν τρία στοιχεία κάλυψης, η τιμή του ορίου και οι δύο γείτονες του, δηλαδή αν η συνθήκη είναι $x \leq 10$, τότε τα test cases θα εκτελεστούν για $x = 9$, $x = 10$ και $x = 11$. Σε αυτή την περίπτωση, η κάλυψη θα είναι 100% όταν τα test cases θα εφαρμόζονται για όλα τα όρια αλλά και τις γειτονικές τιμές. Συμπερασματικά, η 3-value BVA τεχνική είναι πιο αυστηρή αλλά και αποδοτική, καθώς μπορεί να εντοπίσει σφάλματα που παραβλέπονται με την 2-value BVA. [2]

2.7.1.3. Decision Table Testing

Οι Decision Tables χρησιμοποιούνται για τον έλεγχο απαιτήσεων ενός συστήματος όπου οι διαφορετικοί συνδυασμοί συνθηκών οδηγούν σε διαφορετικά αποτελέσματα. Αποτελούν το πιο αποτελεσματικό τρόπο για την καταγραφή περίπλοκης λογικής που αφορούν επιχειρησιακούς κανόνες. Κατά την χρήση της τεχνικής Decision Table Testing, τα test cases είναι σχεδιασμένα έτσι ώστε να

εφαρμόζουν όλους του πιθανούς συνδυασμούς που προκύπτουν από τους decision tables. Η δημιουργία των decision tables γίνεται λαμβάνοντας υπόψιν τις συνθήκες και τα αποτελέσματα των ενεργειών του συστήματος. Κάθε στήλη ενός πίνακα αντιστοιχεί σε έναν κανόνα απόφασης που ορίζει έναν μοναδικό συνδυασμό συνθηκών, μαζί με τις σχετικές ενέργειες. Κάθε σειρά του πίνακα αναπαριστά μία συνθήκη, εάν η συνθήκη ικανοποιείται τότε σημειώνεται «T» (True), αλλιώς με «F» (False). Για τις ενέργειες που δεν είναι εφικτές σημειώνεται «N/A», ενώ οποιαδήποτε άλλη σημείωση μπορεί να χρησιμοποιηθεί ανάλογα με τις ανάγκες.

Ένας πίνακας μπορεί να περιλαμβάνει αρκετές στήλες ώστε να καλύψει κάθε συνδυασμό μιας συνθήκης, ωστόσο μπορεί να απλοποιηθεί εάν διαγραφούν οι στήλες που αντιστοιχούν σε μη-εφικτούς συνδυασμούς. Επιπλέον, ο πίνακας μπορεί να ελαχιστοποιηθεί με την συγχώνευση στηλών, οι οποίες συνθήκες δεν επηρεάζουν το αποτέλεσμα, σε μία. Το πλεονέκτημα της τεχνικής αυτής είναι ότι παρέχει μία συστηματική προσέγγιση για τον εντοπισμό όλων των δυνατών συνδυασμών των συνθηκών. Ακόμη, βοηθά να βρεθούν τυχόν κενά ή αντιφάσεις στις απαιτήσεις. [2] [14]

2.7.1.4. State Transition Testing

Ένα State Transition Diagram αναπαριστά την συμπεριφορά ενός συστήματος δείχνοντας όλες τις πιθανές καταστάσεις και τις έγκυρες μεταβάσεις των καταστάσεων. Μία μετάβαση ξεκινά από ένα γεγονός, το οποίο μπορεί να χαρακτηριστεί από μια συνθήκη. Ένας State Table είναι ένα μοντέλο ισοδύναμο με ένα State Transition Diagram, όπου οι σειρές του αντιπροσωπεύουν καταστάσεις και οι στήλες συμβάντα. Οι μεταβάσεις αποτελούν τις εγγραφές του πίνακα και περιέχουν την επιθυμητή κατάσταση καθώς και τις ενέργειες που προκύπτουν (εάν ορίζονται). Σε αντίθεση με το State Transition Diagram, ο State Table περιλαμβάνει και τις μη-έγκυρες μεταβάσεις (που εμφανίζονται ως κενά κελιά). Ένα test case της State Transition τεχνικής παρουσιάζεται ως ακολουθία από γεγονότα, τα οποία καταλήγουν σε μια σειρά αλλαγών κατάστασης (μεταβάσεις). Αξιοσημείωτο είναι ότι ένα test case μπορεί να καλύπτει αρκετές μεταβάσεις μεταξύ των καταστάσεων. [2] [15]

2.7.2. White-box Test Techniques

Υπάρχουν αρκετές και αυστηρές τεχνικές που χρησιμοποιούνται σε περιβάλλοντα που είναι κρίσιμη η ασφάλεια τους και υψηλής ακεραιότητας για να επιτύχουν ενδελεχή κάλυψη του κώδικα. Ωστόσο

υπάρχουν και τεχνικές για υψηλότερα επίπεδα ελέγχου όπως API Testing, ωστόσο οι πιο διαδεδομένες White-box τεχνικές ελέγχου, για κάλυψη κώδικα, είναι:

- Statement testing
- Decision/Branch testing

2.7.2.1. Statement Testing

Ο σκοπός της Statement testing τεχνικής είναι να σχεδιάσει test cases που εκτελούν τα statements (εντολές) του κώδικα, δηλαδή εξετάζεται κάθε κομμάτι εκτελέσιμου κώδικα. Για να επιτευχθεί κάλυψη 100%, θα πρέπει να εκτελεστούν όλα τα statements τουλάχιστον μία φορά, το γεγονός αυτό σημαίνει ότι εάν κάποιο κομμάτι του κώδικα παράγει σφάλμα, είναι πολύ πιθανό να γίνει αντιληπτό. Ωστόσο, αυτό δεν μπορεί να διασφαλίσει ότι θα εντοπιστεί κάθε σφάλμα, διότι μπορεί να εξαρτάται από τα test data ή άλλους παράγοντες. [2] [16]

2.7.2.2. Decision/Branch Testing

Με την τεχνική Decision (ή Branch) Testing, ο έλεγχος ακολουθεί την ροή του κώδικα, κάθε κόμβος, δηλαδή ένα σημείο απόφασης (για παράδειγμα ένα IF statement), δείχνει τα πιθανά statements μπορούν να εκτελεστούν. Η τεχνική αυτή εφαρμόζει τις αποφάσεις σε κάθε κόμβο και με αυτόν τον τρόπο ελέγχεται ο κώδικας που θα εκτελεστεί βάσει του αποτελέσματος της απόφασης. Πιθανώς, όμως, η τεχνική αυτή να μην εντοπίζει όλα τα σφάλματα, εάν για παράδειγμα δεν ακολουθηθεί η συγκεκριμένη ροή κώδικα που παράγει το σφάλμα. [2] [16]

2.7.3. Experience-based Test Techniques

Κατά την χρήση των Experience-based τεχνικών ελέγχου, τα test cases προκύπτουν από τις δεξιότητες του μέλους της ομάδας ελέγχου καθώς και από την εμπειρία του με παρόμοια λογισμικά και τεχνολογίες. Αυτού του είδους οι τεχνικές μπορεί να είναι χρήσιμες για την εύρεση σφαλμάτων που δεν

εντοπίστηκαν από άλλες συστηματικές τεχνικές. Βάσει της εμπειρίας και της προσέγγισης του ατόμου, οι τεχνικές αυτές μπορεί να αποδειχθούν ιδιαίτερα αποτελεσματικές και ευρέως χρησιμοποιούνται οι εξής:

- Error Guessing
- Exploratory Testing
- Checklist-based Testing

2.7.3.1. Error Guessing

Η Error Guessing τεχνική είναι σχετικά απλή και χρησιμοποιείται για την πρόβλεψη της εμφάνισης σφαλμάτων, ελαττωμάτων, αστοχιών με βάση τις γνώσεις και την εμπειρία της ομάδας ελέγχου. Για να επιτύχει η τεχνική χρειάζεται καλή γνώση της υπάρχουσας λειτουργικότητας του λογισμικού, το είδος των σφαλμάτων που τείνουν να προκύπτουν και τις διάφορες αστοχίες που έχουν ήδη παρουσιαστεί σε άλλα λογισμικά. Μία μεθοδική προσέγγιση αυτής της τεχνικής είναι η δημιουργία μιας λίστας πιθανών σφαλμάτων, ελαττωμάτων και αστοχιών καθώς και test cases που θα εκτελεστούν και θα εντοπίσουν αυτά τα σφάλματα. Σε κάθε περίπτωση, συνίσταται η τεχνική Error Guessing να έπεται άλλων πιο συστηματικών τεχνικών. [2] [17]

2.7.3.2. Exploratory Testing

Ακόμη μία experience-based τεχνική ελέγχου είναι η Exploratory Testing, όπου τα test cases σχεδιάζονται, καταγράφονται, εκτελούνται, και αξιολογούνται δυναμικά κατά την εκτέλεση του ελέγχου. Ταυτόχρονα, τα αποτελέσματα συμβάλλουν στην εξοικείωση της ομάδας ελέγχου με την λειτουργικότητα που ελέγχεται, γι' αυτό η τεχνική αυτή χαρακτηρίζεται και ως μια προσέγγιση εκμάθησης του λογισμικού. Επιπλέον, θεωρείται χρήσιμη όταν υπάρχουν λίγες ή ελλιπής προδιαγραφές ή στην περίπτωση που υπάρχει ανεπαρκής χρόνος για έλεγχο και συνήθως χρησιμοποιούνται για να συμπληρώσουν άλλες black-box και white-box τεχνικές. [2][17]

2.7.3.3. Checklist-based Testing

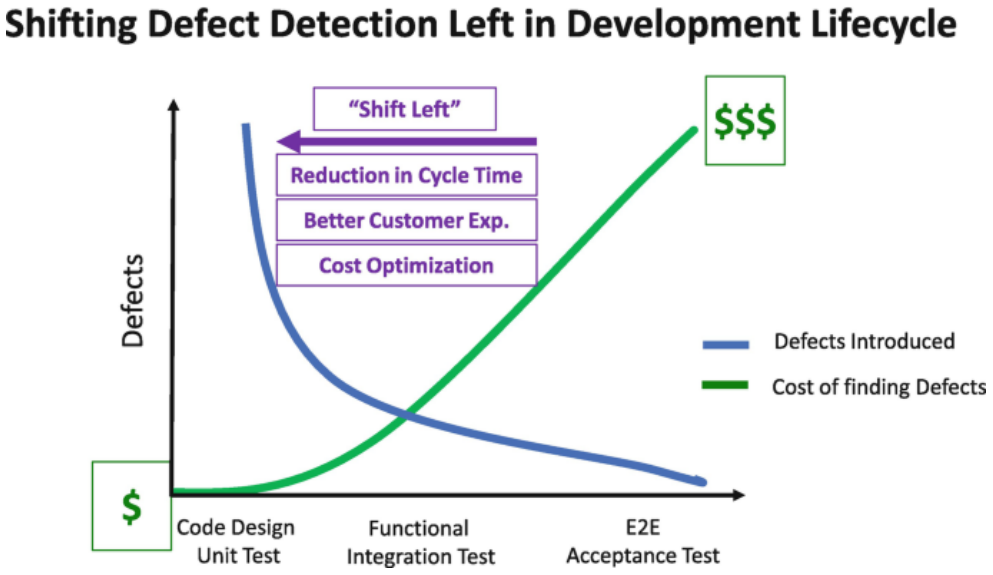
Για την εφαρμογή της Checklist-based Testing τεχνικής, η ομάδα ελέγχου βασίζεται στην εμπειρία και τις γνώσεις των μελών και κατά την διαδικασία ανάλυσης ελέγχου δημιουργούν μία νέα λίστα ή εμπλουτίζουν μια υπάρχουσα η οποία καθοδηγεί την ομάδα σχετικά με τις ενέργειες που πρέπει να πραγματοποιηθούν. Αυτή η λίστα επιτρέπει στην ομάδα ελέγχου να παρακολουθεί την πρόοδο του ελέγχου και να διασφαλίζει ότι έχουν εκτελεστεί όλες οι απαιτούμενες εργασίες. Λίστες τέτοιου είδους μπορούν να δημιουργηθούν για την υποστήριξη διαφόρων τύπου ελέγχου, συμπεριλαμβανομένων των functional και non-functional. Η τεχνική Checklist-based Testing χρησιμεύει όταν πρέπει να πληρούνται συγκεκριμένες απαιτήσεις ή κανονισμοί καθώς μπορεί να προσαρμοστεί κατάλληλα ανά περίπτωση. [2] [17]

2.8. Shift-Left Προσέγγιση

Η έννοια της Shift-Left προσέγγισης δεν είναι καινούρια, αλλά δεν έχει υιοθετηθεί ακόμη από πολλές επιχειρήσεις και ομάδες ανάπτυξης λογισμικού. Η κύρια ιδέα της εύρεσης σφαλμάτων και άλλως προβλημάτων νωρίτερα στον κύκλο ζωής λογισμικού είναι απαραίτητη για την μείωση κόστους και την εξάλειψη κινδύνων ποιότητας. Όταν τα σφάλματα και τα προβλήματα εντοπίζονται αργά στον κύκλο ζωής, η επιδιόρθωσή τους θα μπορούσε να αποσταθεροποιήσει τον κώδικα, να επιφέρει νέα σφάλματα και καθυστερήσεις στην παράδοση του λογισμικού. Το κόστος εύρεσης και επίλυσης προβλημάτων σε αργότερα στάδια του κύκλου ανάπτυξης είναι εκθετικά υψηλότερο από την εύρεση και επίλυση τους νωρίτερα. Η αρχή του έγκαιρου ελέγχου, δηλαδή η Shift-Left προσέγγιση, δεν βοηθά μόνο στην βελτιστοποίηση του κόστους αλλά μειώνει επίσης τον χρόνο του κύκλου ζωής και τελικά δημιουργεί ένα καλύτερο λογισμικό.

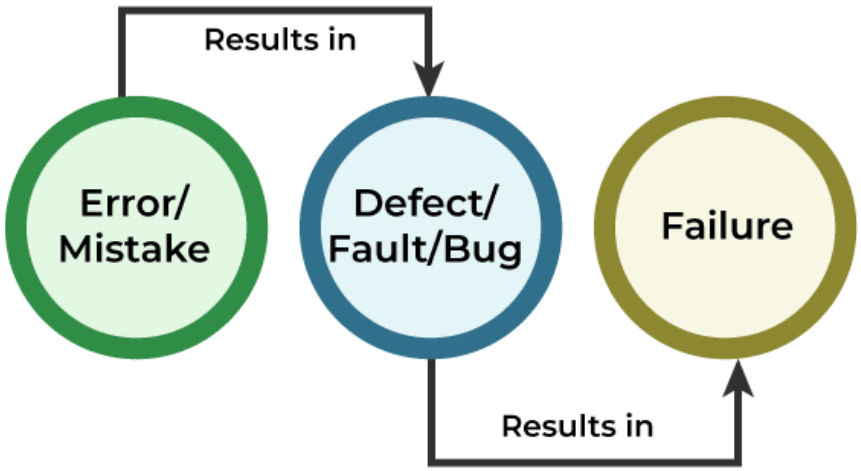
Η σταθερή στρατηγική ελέγχων σε συνδυασμό με τις καλές πρακτικές ανάπτυξης λογισμικού θεωρούνται τα βασικά βήματα για την βελτίωση της εμπειρίας του χρήστη/πελάτη. Ο αυτοματοποιημένος έλεγχος δεν αποτελεί πλέον πολυτέλεια αλλά θεωρείται αναγκαιότητα για την ταχύτερη απόδοση ποιότητας. Για να επιτευχθεί η πρακτική Shift-Left, αρχικά, να γίνεται αξιολόγηση των προδιαγραφών από την πλευρά του ελέγχου και συχνά βρίσκονται πιθανά σφάλματα, όπως ασάφειες, ατέλειες και ασυνέπειες. Στην συνέχεια, χρειάζεται τα test cases να γραφτούν πριν τον κώδικα, ώστε να μπορεί να εκτελεστεί ο έλεγχος όταν τρέξει ο κώδικάς. Επιπλέον, καλό είναι να χρησιμοποιείται Continuous

Integration (CI) και Continuous Delivery (CD) ώστε κάθε φορά που ολοκληρώνεται ένα κομμάτι κώδικα να εκτελούνται οι αυτοματοποιημένοι έλεγχοι και παρέχεται άμεση αξιολόγηση. Τέλος, θα πρέπει να πραγματοποιείται η εκτέλεση μη-λειτουργικών ελέγχων στο επίπεδο του component testing, εφόσον είναι δυνατό, παρόλο που τέτοιου είδους έλεγχοι συνηθίζουν να εκτελούνται σε αργότερα στάδια του κύκλου ζωής του λογισμικού, όταν είναι διαθέσιμο πλήρως το λογισμικό και υπάρχει ένα αντιπροσωπευτικό περιβάλλον ελέγχου. [2] [18]



2.9. Βασικές Έννοιες

Τα ανθρώπινα όντα κάνουν λάθη, τα οποία προκαλούν ελαττώματα/σφάλματα (bugs) και αυτά με την σειρά του ίσως προκαλέσουν αστοχίες/αποτυχίες. Εφόσον, λοιπόν, τα λογισμικά αναπτύσσονται από ανθρώπους μπορεί να έχουν σφάλματα που οδηγούν σε αποτυχημένες διαδικασίες. Στον κόσμο των λογισμικών, τα ελαττώματα (defects) μπορεί να βρεθούν κατά την ανάλυση, όπως κατά τον προσδιορισμό των απαιτήσεων ή ακόμη τον έλεγχο του κώδικα. Στην περίπτωση που τα ελαττώματα δεν εντοπιστούν σε αυτά τα στάδια και το λογισμικό τεθεί σε χρήση, τότε θα προκληθούν διάφορες αστοχίες κατά την λειτουργία του και είτε δεν θα λειτουργήσει όπως προβλέπεται (σφάλμα) είτε αποτύχει εντελώς (failure). [2]



2.9.1. Ελάττωμα (Defect)

Εσφαλμένα οι όροι Ελάττωμα (Defect) και Σφάλμα (Bug) συχνά χρησιμοποιούνται με την ίδια σημασία, δηλαδή ένα λάθος στον κώδικα του λογισμικού, ωστόσο υπάρχουν διαφορές μεταξύ τους. Ένα λάθος, που προκαλεί ελάττωμα στο λογισμικό, γίνεται από τον προγραμματιστή κατά την διάρκεια της ανάπτυξης. Το ελάττωμα, όμως, είναι μια ασυμβατότητα μεταξύ των αναμενόμενων και των πραγματικών αποτελεσμάτων που εντοπίζεται στο περιβάλλον παραγωγής.

Τα είδη των ελαττωμάτων ποικίλουν και προέρχονται από διαφορετικές αιτίες, όπως μη επαρκής έλεγχος στο σύνολο του λογισμικού, παρανόηση των απαιτήσεων μεταξύ της ομάδας ανάπτυξης και του πελάτη/χρήστη, λανθασμένη διαμόρφωση του περιβάλλοντος δοκιμής, ελλιπής οργάνωση κατά την διάρκεια του ελέγχου, κακή διαχείριση των ελαττωμάτων, παράλειψη ελέγχου σε πραγματικές συνθήκες.

Για την αντιμετώπιση και την επίλυση των ελαττωμάτων, ακολουθείται η διαδικασία κατά την οποία αναγνωρίζεται αρχικά το ελάττωμα, προτεραιοποιείται σύμφωνα με τον κίνδυνο που μπορεί να προκαλέσει, προγραμματίζεται και πραγματοποιείται η διόρθωση του και τέλος συντάσσεται η αναφορά επίλυσης. Κατά την προτεραιοποίηση, γίνεται αξιολόγηση του κινδύνου και ανάλογα την περίπτωση μπορεί να ληφθεί η απόφαση να μην επιλυθεί το ελάττωμα, εάν δεν είναι μείζονος σημασίας για την λειτουργία του λογισμικού. [19]

2.9.2. Σφάλμα (Bug)

Το Σφάλμα (Bug) είναι το λάθος που έχει εντοπιστεί στο περιβάλλον ανάπτυξης κατά την διάρκεια του ελέγχου του λογισμικού. Τα σφάλματα δημιουργούνται λόγω έλλειψης χρόνου και πόρων της ομάδας ανάπτυξης ή απλά λόγω εισαγωγής μη έγκυρων τιμών και μη καλής χρήσης της εφαρμογής.

Για την επίλυση κάθε σφάλματος ακολουθείται η ίδια διαδικασία για την αντιμετώπιση των Ελαττωμάτων (Defects) σχετικά με το severity και το priority, όπως περιγράφεται στην προηγούμενη παράγραφο. Αξίζει να τονιστεί ότι η διαφορά των όρων Σφάλμα και Ελάττωμα είναι το περιβάλλον εντοπισμού τους και το στάδιο της ανάπτυξης που βρίσκεται το λογισμικό. Οι δύο αυτοί παράγοντες επηρεάζουν άμεσα την πολυπλοκότητα της επίλυσης τους. [19]

2.9.3. Αστοχία (Failure)

Με τον όρο Αστοχία (Failure) υποδηλώνεται οποιαδήποτε δυσλειτουργία σε επιμέρους λειτουργικότητα ή αδυναμία λειτουργίας του λογισμικού. Μερικά σφάλματα στον κώδικα επιφέρουν αστοχίες (Failures) μόνο αν εκτελεστεί το συγκεκριμένο κομμάτι του κώδικα, αλλά υπάρχουν και σφάλματα τα οποία προκαλούν αστοχίες κάτω από συγκεκριμένες συνθήκες. Ωστόσο τα ελαττώματα και τα σφάλματα δεν είναι οι μόνες αιτίες αστοχιών, αστοχίες μπορεί να προκληθούν από περιβαλλοντικές συνθήκες όπως το ηλεκτρομαγνητικό πεδίο που επηρεάζει το firmware. [19]

2.9.4. Root Cause Analysis

Εφόσον εντοπιστεί κάποιο ελάττωμα ή σφάλμα, η ομάδα ανάπτυξης θα πρέπει να διερευνήσει την βασική αιτία (root cause) του προβλήματος μέσω της αντίστοιχης ανάλυσης (Root Cause Analysis – RCA). Όταν βρεθεί η βασική αιτία και αντιμετωπιστεί κατάλληλα, τότε μπορούν να αποφευχθούν άλλες παρόμοιες αστοχίες και σφάλματα ή να μειωθεί η συχνότητα τους. [2]

2.9.5. Severity & Priority

Τα ελαττώματα και τα σφάλματα κατηγοριοποιούνται ανάλογα τη σοβαρότητα (severity) και την προτεραιότητα (priority). Το Severity δείχνει τον βαθμό επίπτωσης του σφάλματος στην λειτουργία του λογισμικού από την σκοπιά των άμεσων ενδιαφερόμενων (πελατών και χρηστών). Αντιθέτως, το Priority καθορίζει την σειρά με την οποία η ομάδα ανάπτυξης θα διορθώσει τα σφάλματα. Όσο υψηλότερη είναι η επίπτωση του σφάλματος στο τελικό αποτέλεσμα, τόσο μεγαλύτερη είναι η προτεραιότητα που του αποδίδεται. [20]

Οι κατηγορίες του Severity είναι:

- **Critical:** Το σφάλμα ή ελάττωμα, που έχει μπλοκάρει εντελώς την λειτουργία του λογισμικού και ο τελικός χρήστης δεν μπορεί να το χρησιμοποιήσει λόγω αυτού, χαρακτηρίζεται κρίσιμο και πρέπει να διορθωθεί άμεσα. Η επίπτωση ενός κρίσιμου σφάλματος επηρεάζει όλη την λειτουργία του συστήματος και ίσως κάποιες φορές το σύστημα είναι εξ ολοκλήρου απενεργοποιημένο.
- **Major:** Στην περίπτωση που το σφάλμα δεν επηρεάζει όλο το σύστημα αλλά εξακολουθεί να εμποδίζει κάποιες λειτουργίες του, χαρακτηρίζεται μείζον. Στην πραγματικότητα, δεν υπάρχει πλήρης τερματισμός λειτουργίας αλλά επιφέρει δυσκολίες ομαλής χρήσης σε επιμέρους λειτουργίες.

- **Minor:** Τα δευτερεύοντα σφάλματα ενδέχεται να μην μπλοκάρουν την λειτουργία του συστήματος αλλά προκαλούν μια συμπεριφορά της εφαρμογής η οποία είναι διαφορετική από την αναμενόμενη. Η επίλυση τέτοιου είδους σφαλμάτων, που δεν περιορίζουν την λειτουργικότητα, μπορούν να γίνουν σε επόμενες εκδόσεις.
- **Trivial:** Τα ελαττώματα που δεν επηρεάζουν καθόλου την λειτουργικότητα και την χρήση του συστήματος αλλά είναι αισθητικής φύσεως, είναι αφενός έγκυρα αλλά αφετέρου δεν χρήζουν άμεση αντιμετώπιση.

Τα επίπεδα του Priority είναι:

- **High:** Εάν το σφάλμα επηρεάζει άμεσα την εμπειρία του χρήστη, επισημαίνεται ως υψηλής προτεραιότητας. Αυτού του είδους τα σφάλματα ενδέχεται να επηρεάσουν ολόκληρη την εφαρμογή και χρήσουν επίλυση το συντομότερο δυνατό. Ο χαρακτηρισμός High Priority εξασφαλίζει τον λιγότερο δυνατό χρόνο για επίλυση.
- **Medium:** Τα σφάλματα, που δεν επηρεάζουν την επιχείρηση και τους πελάτες, συνήθως χαρακτηρίζονται ως μέσης προτεραιότητας. Δεν είναι τόσο επείγοντα όσο τα υψηλής προτεραιότητας και μπορούν να διορθωθούν όταν η ομάδα ανάπτυξης έχει τον διαθέσιμο χρόνο για να τα αναλάβει, αυτό σημαίνει ότι μπορούν να διορθωθούν στην τρέχουσα ή σε επόμενη έκδοση.
- **Low:** Τα σφάλματα με την μικρότερη προτεραιότητα επιδιορθώνονται εφόσον όλα τα υψηλής και μέσης προτεραιότητας έχουν διορθωθεί. Η επιδιόρθωση τους συνήθως παραδίδεται με άλλα διορθωμένα σφάλματα μεγαλύτερης προτεραιότητας.

3. Agile Ανάπτυξη Λογισμικού

3.1. Μοντέλο Agile

Το μοντέλο Agile είναι η πιο ευρέως κοινή τεχνική ανάπτυξης λογισμικού, η οποία έχει αντικαταστήσει το μοντέλο Waterfall, και προτείνει συχνούς και σύντομους κύκλους (iterations) ανάπτυξης και παράδοσης λογισμικού σε μέρη. Αυτό επιτρέπει στους πελάτες και οποιονδήποτε άλλο stakeholder να έχουν μεγαλύτερη συμμετοχή στην διαδικασία ανάπτυξης, με αποτέλεσμα στην προώθηση της ποιότητας του λογισμικού, αφού ο πελάτης κατανοεί και προσδιορίζει πλήρως τις απαιτήσεις στην φάση του σχεδιασμού και της ανάλυσης. Το Agile δημιουργήθηκε για να βοηθήσει την ομάδα ανάπτυξης να δημιουργήσει ένα προϊόν το οποίο μπορεί να προσαρμοστεί εύκολα και γρήγορα στις αλλαγές των απαιτήσεων. Γι' αυτό εξασφαλίζεται ευελιξία κατά την ανάπτυξη και εξαλείφονται δραστηριότητες, οι οποίες μπορεί να μην είναι κρίσιμες για το συγκριμένο έργο ή την συγκεκριμένη χρονική περίοδο.

Το μοντέλο Agile αποτελεί ένα συνδυασμό επαναληπτικού και σταδιακού μοντέλου. Σε κάθε επανάληψη συμμετέχει μια ομάδα που εργάζεται ταυτόχρονα σε διάφορους τομείς ανάπτυξης του προϊόντος, όπως τον χρονικό προγραμματισμό, την ανάλυση των απαιτήσεων, τον σχεδιασμό, την συγγραφή του κώδικα, τον έλεγχο. Η επαναληπτική προσέγγιση έχει γίνει αρκετά αποτελεσματική βοηθώντας την ομάδα ανάπτυξης να βελτιώσει τις δεξιότητες της στην εκτίμηση και τον προσδιορισμό του χρονοδιαγράμματος για όλες τις απαιτούμενες δραστηριότητες. Ο προσδιορισμός του χρονοδιαγράμματος είναι μια από τις πιο δύσκολες ευθύνες της ομάδας ανάπτυξης, επειδή τα τεχνικά προβλήματα, που μπορεί να προκύψουν και να καθυστερήσουν το χρονοδιάγραμμα, είναι απρόβλεπτα. Το Agile δίνει λύση σε αυτό πρόβλημα και προωθεί την ανάλυση των απαιτήσεις μεγάλης κλίμακας σε μικρότερα μέρη, πιο διαχειρίσιμα, επιτυγχάνοντας καλύτερη και πιο ρεαλιστική εκτίμηση.

Στον σημερινό κόσμο ανάπτυξης λογισμικού, η συμμετοχή των stakeholders είναι απαραίτητη ώστε να είναι ενήμεροι για την συνεχή πρόοδο του έργου και να μπορούν να επέμβουν με τυχόν αλλαγές, νέες προτάσεις και βελτιωμένες ιδέες. Καθώς στο Agile, υπάρχουν σταθεροί κύκλοι εργασιών και παράδοσης λογισμικού, οι stakeholders μπορεί να επηρεάσουν το αποτέλεσμα του λογισμικού, δεδομένου ότι είναι απόλυτα αποδεκτό να εφαρμοστούν αλλαγές κατά την διάρκεια της ανάπτυξης.

Το μοντέλο Agile προβλέπει την ανάγκη για ευελιξία και εφαρμόζει ένα επίπεδο προγραμματισμού στην παράδοση του τελικού προϊόντος. Η ανάπτυξη λογισμικού με το μοντέλο Agile απαιτεί μια πολιτιστική αλλαγή σε πολλές επιχειρήσεις, επειδή εστιάζει στην παράδοση μεμονωμένων μερών ή

τμημάτων του λογισμικού και όχι την παράδοση ολόκληρης της εφαρμογής. Τα πλεονεκτήματα περιλαμβάνουν την ικανότητα του μοντέλου να βοηθά τις ομάδες ανάπτυξης σε ένα εξελισσόμενο πεδίο, διατηρώντας παράλληλα τη εστίαση στην αποτελεσματική απόδοση της επιχειρηματικής αξίας. Η κουλτούρα της συνεργασίας, που διευκολύνεται από το Agile, βελτιώνει επίσης την αποτελεσματικότητα σε ολόκληρη την επιχείρηση, καθώς οι ομάδες συνεργάζονται και κατανοούν τους συγκεκριμένους ρόλους τους στην διαδικασία. Τέλος, οι επιχειρήσεις που αναπτύσσουν προϊόντα λογισμικού Agile διασφαλίζουν ότι το προϊόν είναι υψηλής ποιότητας, επειδή πραγματοποιούνται έλεγχοι καθ' όλη την διάρκεια της ανάπτυξης. Επιπλέον, οι έλεγχοι μπορεί να επιφέρουν απαραίτητες αλλαγές για την αντιμετώπιση τυχόν προβλημάτων, ωστόσο μπορούν να διαχειριστούν και να εφαρμοστούν εύκολα χρησιμοποιώντας το Agile. [21]

3.2. Αξίες Agile

Το μοντέλο Agile βασίζεται στις αξίες και στις αρχές του Agile Manifesto [5], το οποίο δημιουργήθηκε το 2001 από κορυφαίους επαγγελματίες του χώρου. Το Agile Manifesto ορίζει τέσσερις βασικές αξίες του Agile, στις οποίες βασίζονται οι διαδικασίες ανάπτυξης, και είναι οι παρακάτω:

- **Τα άτομα της ομάδας και οι αλληλεπιδράσεις του είναι πιο σημαντικές από τις διαδικασίες και τα εργαλεία:** Οι άνθρωποι είναι αυτοί που καθορίζουν την διαδικασία της ανάπτυξης λογισμικού και ανταποκρίνονται στην επιχειρηματικές ανάγκες, γι' αυτό αποτελούν το πιο σημαντικό μέρος της ανάπτυξης και πρέπει να εκτιμώνται πάνω από τις διαδικασίες και τα εργαλεία. Εάν οι διαδικασίες και τα εργαλεία καθορίζουν την πρόοδο της ανάπτυξης, τότε η ομάδα θα είναι λιγότερο πιθανό να ανταποκριθεί και να προσαρμοστεί επιτυχώς στις αλλαγές και, επομένως, να καλύψει τις ανάγκες των stakeholders
- **Εστίαση στο λειτουργικό λογισμικό παρά στην ενδελεχή τεκμηρίωση:** Πριν το Agile, είχε δαπανηθεί πολύς χρόνος για την τεκμηρίωση του προϊόντος καθ' όλη την διάρκεια της ανάπτυξης. Η λίστα των τεκμηριωμένων απαιτήσεων ήταν μεγάλη και προκαλούσε μεγάλες καθυστερήσεις στην διαδικασία. Αντίθετα, το Agile δεν καταργεί την χρήση της τεκμηρίωσης αλλά την απλοποιεί με τρόπο που παρέχει στην ομάδα μόνο τις πληροφορίες που χρειάζονται για την εκτέλεση των απαραίτητων διαδικασιών – όπως τα user stories. Το Agile Manifesto συνεχίζει να δίνει αξία στη διαδικασία τεκμηρίωσης, αλλά δίνει μεγαλύτερη αξία στο λειτουργικό λογισμικό.

- **Συνεργασία αντί για διαπραγματεύσεις συμβολαίων:** Το Agile εστιάζει στην συνεργασία μεταξύ των stakeholders και του project manager, για να επεξεργαστεί τις λεπτομέρειες της παράδοσης του έργου. Η συνεργασία και η συμμετοχή των stakeholders σημαίνει ότι συμπεριλαμβάνονται σε όλη την διαδικασία ανάπτυξης και όχι μόνο στην αρχή και το τέλος, διευκολύνοντας έτσι τις ομάδες να ανταποκριθούν στις ανάγκες τους. Για παράδειγμα, στο Agile, πραγματοποιούνται μικρές παρουσιάσεις του υλοποιημένου λογισμικού (demo) σε κάθε sprint, δίνοντας έτσι την δυνατότητα να εκφέρει άποψη. Ωστόσο, οι stakeholders μπορούν να είναι παρών καθημερινά, αλληλοεπιδρώντας με την ομάδα και παρακολουθώντας όλες τις συναντήσεις ώστε να διασφαλίσουν ότι το λογισμικό ανταποκρίνεται στις επιθυμίες τους.
- **Εστίαση στην ανταπόκριση στην αλλαγή:** Η παραδοσιακή ανάπτυξη λογισμικού χρησιμοποιήθηκε για να αποφευχθούν οι αλλαγές επειδή θεωρούνταν ανεπιθύμητες δαπάνες. Το Agile εξαλείφει αυτή την ιδέα, αφού οι σύντομες επαναλήψεις των κύκλων εργασιών επιτρέπουν την εύκολη πραγματοποίηση αλλαγών, βοηθώντας την ομάδα να τροποποιήσει την διαδικασία ώστε να ταιριάζει καλύτερα στις ανάγκες της και το αντίστροφο. Κατά το Agile, γενικότερα, πιστεύεται ότι οι αλλαγές εν μέσω της ανάπτυξης λογισμικού είναι ένας τρόπος βελτίωσης του έργου και προσδίδουν αξία

3.3. Ομάδα Ανάπτυξης & Ρόλοι

Η Agile ανάπτυξη εφαρμογών βασίζεται περισσότερο στα άτομα της ομάδας και τις αλληλεπιδράσεις παρά σε προκαθορισμένες διαδικασίες. Οι ομάδες ανάπτυξης αποτελούνται από λίγα άτομα που συγκεντρώνονται, δεσμεύονται να επιτύχουν έναν κοινό στόχο, εργάζονται ταυτόχρονα, αυτοοργανώνονται καθώς και αλληλοεξαρτώνται μεταξύ τους. Κάθε μέθοδος Agile έχει τους δικούς της διαφορετικούς ρόλους, αλλά σε γενικές γραμμές, οι τρεις ρόλοι Agile Manager, Product Owner (PO) και η ομάδα ανάπτυξης είναι οι πιο βασικοί. Στην ομάδα των ανάπτυξης ανήκουν οι Business Analysts (BA), οι Software Developer Engineers (DEVs), οι Quality Assurance Engineers (QA), ο Database Managers (DBA Managers) και τουλάχιστον ένα μέλος Development Operation engineers (DevOps). [22]

Ένα από τα πιο βασικά πρόσωπα μιας Agile ομάδας είναι ο Agile Manager, όπου επιβάλλεται να έχει άριστες γνώσεις των αρχών και των πρακτικών Agile αλλά και αρκετή εμπειρία στην διαχείριση Agile έργων. Είναι κυρίως υπεύθυνος για την επίβλεψη ολόκληρου του κύκλου ζωής του λογισμικού, συμπεριλαμβανομένου του αρχικού καθορισμού του πεδίου εφαρμογής του έργου, του εντοπισμού

πιθανών κινδύνων και της παρακολούθησης του έργου να παραδοθεί με επιτυχία. Τέλος, ο Agile Manager συμμετέχει σε συζητήσεις με τους stakeholders και με κάθε άλλο ενδιαφερόμενο, όπου εκφέρουν την γνώμη τους για το λογισμικό. Μέσω αυτών των συζητήσεων, ο Agile Manager ενημερώνεται για τις τυχόν αλλαγές που προκύπτουν και συνεπώς καθοδηγεί την ομάδα να προσαρμοστεί και να ανταποκριθεί σε αυτές τις ανάγκες. [23]

Ο Product Owner είναι υπεύθυνος για την μεγιστοποίηση της αξίας του λογισμικού που είναι το αποτέλεσμα της εργασίας της ομάδας. Ως μέλος της ομάδας ανάπτυξης, ο Product Owner καθοδηγεί την ομάδα προς την επίτευξη του στόχου, συνεπώς, την επιτυχία έργου. Όλες οι απαραίτητες εργασίες που πρέπει να γίνουν προέρχονται, ιεραρχούνται και αποσαφηνίζονται βάσει τον στόχο που έχει θέσει ο Product Owner προκειμένου να ικανοποιηθούν οι απαιτήσεις των stakeholders. Στις αρμοδιότητες του είναι να καθορίσει τα προβλήματα, προσδιορίσει τις επιχειρηματικές ανάγκες, εκμαιεύσει απαιτήσεις των stakeholders και να συντάξει τα User Stories, που περιγράφουν λεπτομερώς την ζητούμενη λειτουργικότητα βάσει των απαιτήσεων και θα χρησιμοποιηθούν από την ομάδα ανάπτυξης. Επιπλέον, ο Product Owner πρέπει να εντοπίζει, να αξιολογεί και να μεγιστοποιεί την αξία του λογισμικού καθ’ όλη την διάρκεια του κύκλου ζωής του. [24]

Η Ανάλυση Απαιτήσεων αποτελεί τον πυλώνα της ανάπτυξης λογισμικού και εκτιμάται στο 20-30% της συνολικής διάρκειας του έργου. Γι’ αυτό, ο Business Analyst έχει καθοριστικό ρόλο στην ανάπτυξη του λογισμικού εφόσον είναι ο συνδετικός κρίκος μεταξύ των stakeholders, του Product Owner και της ομάδας ανάπτυξης. Κατά την ανάλυση, ο Business Analyst πρέπει επανεξετάσει τα User Stories και να διασφαλίσει ότι οι ανάγκες και οι απαιτήσεις των stakeholders θα ικανοποιηθούν με την υλοποίηση του λογισμικού και ότι η λειτουργικότητα θα είναι σωστή βάσει των προδιαγραφών που έχουν τεθεί. Στην φάση της επανεξέτασης των User Stories, ο Business Analyst μπορεί να βελτιώσει τις απαιτήσεις που περιγράφονται αλλά και να τα εμπλουτίσει με νέες ιδέες προς ανάπτυξη. Επιπλέον, ο Business Analyst είναι υπεύθυνος να οργανώσει τις Agile συναντήσεις με τους εμπλεκόμενους ώστε να αποσαφηνιστούν οι απαιτήσεις και να εξηγηθούν τυχόν λεπτομέρειες των User Stories. [25]

Οι Software Developer Engineers αναλαμβάνουν το σχεδιασμό και την ανάπτυξη αποτελεσματικών λύσεων λογισμικού για την κάλυψη των απαιτήσεων των έργου. Οι περισσότεροι Software Developer Engineers ειδικεύονται σε έναν συγκεκριμένο τομέα και θα μπορούσαν να δουλεύουν σε οτιδήποτε, από Web Εφαρμογές και Mobile Εφαρμογές έως Βάσεις Δεδομένων και Λειτουργικά Συστήματα. [26] Εφόσον η υλοποίηση του λογισμικού (ή ένα μέρους αυτού) έχει ολοκληρωθεί, Quality Assurance Engineers είναι υπεύθυνοι για τον έλεγχο του λογισμικού και την διασφάλιση ότι δεν υπάρχει κάποιο σφάλμα ή πρόβλημα που μπορεί να επηρεάσει την λειτουργικότητα του λογισμικού. Οι Quality Assurance Engineers εκτελούν όλες τις διαδικασίες ελέγχου ποιότητας λογισμικού, γι’ αυτό έχουν

γνώσεις διάφορων εργαλείων και τεχνικών που χρησιμοποιούν για χειροκίνητους ή αυτοματοποιημένους ελέγχους. [27]

Επιπλέον, κάθε ομάδα ανάπτυξης χρειάζεται ανθρώπους, οι οποίοι διαχειρίζονται τις βάσεις δεδομένων, τα δίκτυα και οτιδήποτε αφορά τις υποδομές των συστημάτων. Οι Database Managers καλούνται να αναλάβουν την ανάπτυξη και τη διατήρηση συστημάτων που αποθηκεύουν και οργανώνουν δεδομένα. Οι Database Managers είναι επίσης υπεύθυνοι να εφαρμόζουν τα κατάλληλα προγράμματα ασφαλείας, ώστε να διασφαλίζονται τα αποθηκευμένα δεδομένα, και να επιβλέπουν τις καθημερινές δραστηριότητες της βάσης δεδομένων και την αξιολόγηση των αναγκών σχετικά με την αποθήκευση δεδομένων. [28] Τέλος, διάφορες δραστηριότητες και πρακτικές του Development Operation βοηθούν τις ομάδες ανάπτυξης να επιταχύνουν και να αυτοματοποιήσουν διάφορες διαδικασίες της ανάπτυξης, του ελέγχου, της διάθεσης προς χρήση και της ενημέρωσης του λογισμικού. [29]

Στην πράξη, υπάρχουν Agile ομάδες, στις οποίες ο Product Owner και ο Business Analyst είναι το ίδιο άτομο, δεδομένου ότι έχει τις κατάλληλες γνώσεις και δεξιότητες ώστε να καλύψει τόσο τις επιχειρησιακές όσο και τις τεχνικές πτυχές του λογισμικού. Τέλος, επισημαίνεται ότι όλοι οι ρόλοι συμμετέχουν σε ολόκληρο τον κύκλο ζωής του έργου και τις Agile συναντήσεις, όπως θα επιγραφθούν παρακάτω.

3.4. Κύκλος Ζωής Ανάπτυξης Λογισμικού Agile

Ο κύκλος ζωής ανάπτυξης λογισμικού Agile είναι δομημένα στάδια που περνά ένα λογισμικό καθώς κινείται από την αρχή στο τέλος και περιλαμβάνει έξι στάδια: την ιδέα, την έναρξη, την επανάληψη, την παράδοση, την συντήρηση και την απόσυρση, όπως περιγράφονται παρακάτω. Ο κύκλος ζωής θα ποικίλλει ελαφρώς ανάλογα με τη μεθοδολογία διαχείρισης έργου που επιλέγεται από μια ομάδα. Για παράδειγμα, οι ομάδες Scrum εργάζονται σε σύντομες χρονικές περιόδους γνωστές ως sprint και έχουν επίσης σαφώς καθορισμένους ρόλους, όπως τον Scrum master. Από την άλλη πλευρά, οι ομάδες Kanban εργάζονται χωρίς απαιτούμενους ρόλους. Ένα άλλο παράδειγμα είναι ο Extreme Programming, όπου οι ομάδες τείνουν να εργάζονται σε μικρότερους κύκλους επανάληψης και δίνουν επιπλέον έμφαση στις πρακτικές μηχανικής. Ωστόσο, ο στόχος όλων των ομάδων ανάπτυξης λογισμικού είναι ο ίδιος: να παραδώσουν το λειτουργικό λογισμικό στους χρήστες εγκαίρως. [30] [31]

3.4.1. Στάδιο 1: Η Ιδέα

Το πρώτο στάδιο, η Ιδέα, περιλαμβάνει τον καθορισμό των επιχειρηματικών αναγκών για ένα πιθανό έργο, το εύρος του καθώς και μια εκτίμηση του χρόνου και της εργασίας που απαιτείται. Εάν υπάρχουν πολλά έργα, ο Product Owner θα πρέπει να τα ιεραρχήσει και να διακρίνει ποια από αυτά αξίζει να υλοποιηθούν με βάση την τεχνική και οικονομική σκοπιμότητα. Για να παρθεί αυτή η απόφαση, ο Product Owner θα πρέπει να συζητήσει τις βασικές απαιτήσεις των stakeholders, να προετοιμάσει την τεκμηρίωση τους, συμπεριλαμβανομένων των δυνατοτήτων που θα υποστηριχθούν και των προτεινόμενων τελικών αποτελεσμάτων. Σε αυτό το σημείο, συνιστάται οι απαιτήσεις να περιοριστούν στο ελάχιστο, καθώς μπορούν να προστεθούν σε μεταγενέστερα στάδια. Αυτή η λεπτομερής ανάλυση θα βοηθήσει να αποφασιστεί εάν ένα έργο είναι εφικτό πριν ξεκινήσουν οι απαραίτητες εργασίες.

3.4.2. Στάδιο 2: Η Έναρξη

Κατά το στάδιο της Έναρξης πρέπει να συσταθεί η ομάδα ανάπτυξης, να καθοριστεί η χρηματοδότηση και να ξεκινήσουν οι αρχικές συζητήσεις για τις απαιτήσεις με τους stakeholders για την διαδικασία του σχεδιασμού. Κατά το δεύτερο στάδιο, η ομάδα ανάπτυξης δημιουργεί τα πρώτα σχέδια (mock-ups) του User Interface καθώς και την αρχιτεκτονική του έργου. Το στάδιο της έναρξης περιλαμβάνει επίσης να δοθούν περαιτέρω πληροφορίες από τους stakeholders σχετικά με τις απαιτήσεις και τον προσδιορισμό της λειτουργικότητας του προϊόντος. Θα πρέπει επίσης να δημιουργηθεί ένα χρονοδιάγραμμα που να περιγράφει τις διάφορες ευθύνες της ομάδας ανάπτυξης και να ορίζει με σαφήνεια πότε αναμένεται να ολοκληρωθεί η εργασία για κάθε sprint.

3.4.3. Στάδιο 3: Η Επανάληψη

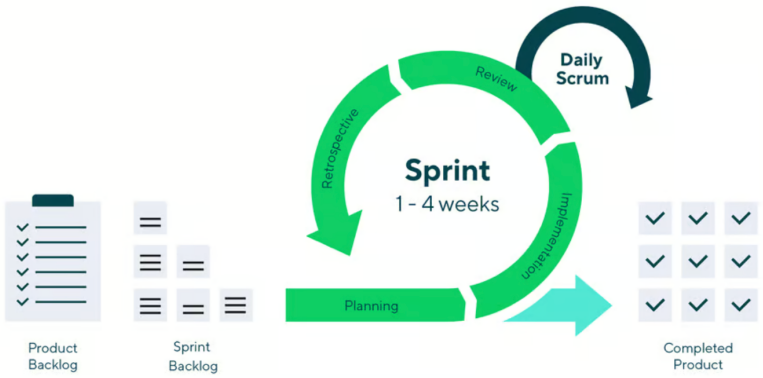
Ακολουθεί το στάδιο της Επανάληψης, όπου τείνει να είναι το μεγαλύτερο στάδιο, καθώς εκτελείται το μεγαλύτερο μέρος της εργασίας. Η ομάδα ανάπτυξης ξεκινά να δημιουργεί το λογισμικό μετατρέποντας τα σχέδια σε κώδικα βάσει των απαιτήσεων και την συνεχή κριτική για καθοδήγηση (feedback) των stakeholders. Ο κύκλος ανάπτυξης λογισμικού Agile βασίζεται σε επαναλήψεις (iterations) ή μεμονωμένους κύκλους ανάπτυξης, που βασίζονται ο ένας στον άλλο και οδηγούν στο

επόμενο βήμα της συνολικής διαδικασίας ανάπτυξης μέχρι να ολοκληρωθεί το έργο. Ο στόχος είναι να δημιουργείται ένα λειτουργικό μέρος του λογισμικού που είναι έτοιμο προς χρήση στο τέλος κάθε sprint, ενώ επιπλέον δυνατότητες και τροποποιήσεις μπορούν να προστεθούν σε επόμενα iterations. Αυτό το στάδιο είναι ο ακρογωνιαίος λίθος της ανάπτυξης λογισμικού Agile, διότι δίνεται η δυνατότητα στους προγραμματιστές να δημιουργήσουν λειτουργικό λογισμικό γρήγορα και να κάνουν βελτιώσεις για να ικανοποιήσουν τους stakeholders.

3.4.3.1. Διαδικασίες Κάθε Επανάληψης (Iteration)

Κάθε στάδιο του κύκλου ζωής Agile περιέχει πολλά iterations και κάθε iteration διαρκεί συνήθως από δύο έως τέσσερις εβδομάδες, με μια καθορισμένη ημερομηνία ολοκλήρωσης. Σε κάθε iteration, όμως, συμβαίνουν οι εξής διαδικασίες:

- **Προσδιορισμός & Προγραμματισμός Απαιτήσεων:** Καθορίζονται οι απαιτήσεις (requirements) σύμφωνα με τις εκκρεμείς απαιτήσεις του λογισμικού (product backlog) και τις μη-υλοποιημένες εργασίες (tasks) προηγούμενων iterations (iteration backlog) και το feedback των stakeholders.
- **Ανάπτυξη Λογισμικού:** Πραγματοποιείται η ανάπτυξη του λογισμικού βάσει των εργασιών (tasks) και απαιτήσεων (requirements) που έχουν τεθεί στο συγκεκριμένο sprint
- **Έλεγχος Λογισμικού:** Διεξάγονται οι απαραίτητοι έλεγχοι, η εκπαίδευση των χρηστών καθώς και η τεκμηρίωση για την λειτουργικότητα και την χρήση του λογισμικού (User Manual)
- **Παράδοση Sprint:** Παραδίδεται προς χρήση το ολοκληρωμένο μέρος του λογισμικού και ενσωματώνεται στο λογισμικό της παραγωγής
- **Αξιολόγηση Λογισμικού:** Μαζεύονται τα σχόλια των stakeholders (feedback), αξιολογούνται και αξιοποιούνται κατάλληλα ώστε να γίνουν απαραίτητες διορθώσεις/βελτιώσεις και να δημιουργηθούν νέες απαιτήσεις για τα επόμενα iterations.



3.4.4. Στάδιο 4: Η Παράδοση

Στο τέταρτο βήμα, το λογισμικό είναι έτοιμο προς παράδοση αλλά πρώτα, η ομάδα διασφάλισης ποιότητας πρέπει να πραγματοποιήσει ορισμένους ελέγχους για να διασφαλίσει ότι το λογισμικό είναι πλήρως λειτουργικό – εάν εντοπιστούν πιθανά σφάλματα ή ελαττώματα, οι προγραμματιστές θα τα διορθώσουν άμεσα. Επιπλέον, κατά τη διάρκεια αυτού του σταδίου, θα πραγματοποιηθεί εκπαίδευση χρηστών, για την οποία χρειάζεται να έχει οριστικοποιηθεί η αντίστοιχη τεκμηρίωση (User Manual). Όταν ολοκληρωθούν όλα αυτά, το τελικό sprint του λογισμικού μπορεί στη συνέχεια να κυκλοφορήσει στην παραγωγή.

3.4.5. Στάδιο 5: Η Συντήρηση

Μετά την Παράδοση, το πέμπτο βήμα είναι η Συντήρηση που εστιάζει στη συνεχή υποστήριξη για την ομαλή λειτουργία του συστήματος και την επίλυση τυχόν νέων σφαλμάτων. Η ομάδα ανάπτυξης προσφέρει πρόσθετη εκπαίδευση στους χρήστες και διασφαλίζουν ότι γνωρίζουν πώς να χρησιμοποιούν το λογισμικό. Με την πάροδο του χρόνου, μπορούν να πραγματοποιηθούν νέα iterations για την ανανέωση του υπάρχοντος λογισμικού με αναβαθμίσεις και πρόσθετες λειτουργίες. Το στάδιο της Συντήρησης συνεχίζεται μέχρις ότου προγραμματιστεί απόσυρση του λογισμικού.

3.4.6. Στάδιο 6: Η Απόσυρση

Το τελευταίο στάδιο, η Απόσυρση, συμβαίνει όταν το λογισμικό είτε αντικαθίσταται με ένα νέο είτε το ίδιο το σύστημα έχει καταστεί απαρχαιωμένο ή ασυμβίβαστο με την πάροδο του χρόνου. Το τελευταίο στάδιο σηματοδοτεί το τέλος του κύκλου ζωής του λογισμικού, γι’ αυτό η ομάδα ανάπτυξης πρέπει να ενημερώσει τους τελικούς χρήστες ότι αφαιρέσουν την υποστήριξη για το συγκεκριμένο λογισμικό.

3.5. Agile Ceremonies

Ένα από τα χαρακτηριστικά του Agile είναι ότι δίνει ιδιαίτερη σημασία και βασίζεται στην επικοινωνία και στην συνεργασία μεταξύ των μελών του έργου. Γι' αυτό, η Agile ανάπτυξη λογισμικού έχει ένα σύνολο πρότυπων Agile Ceremonies (συναντήσεις μεταξύ των μελών της ομάδας), οι οποίες είναι απαραίτητες για την διατήρηση της ευελιξίας της ανάπτυξης λογισμικού. Υπάρχουν τέσσερα ceremonies τα οποία καθορίζουν και «ελέγχουν» την διαδικασία της ανάπτυξης του προϊόντος, το Product Refinement (ή Product Grooming), το Iteration Planning, τα Daily Stand-ups και το Iteration Review & Retrospective. Ο χρόνος που κάθε ομάδα δαπανά για τα Agile Ceremonies και ο χρόνος της ανάπτυξης κατά την διάρκεια του sprint δεν είναι προκαθορισμένοι και σταθεροί από το Agile, αλλά η ίδια ομάδα ανάπτυξης τον ορίζει μόνη της βάσει της ταχύτητας της. Η ομάδα μπορεί να πετύχει την βέλτιστη ισορροπία ανάμεσα στα διάφορα Agile Ceremonies και στις εργασίες που απαιτούνται για την ανάπτυξη. Στον κόσμο της τεχνολογίας, υπάρχει η λανθασμένη εντύπωση ότι τηρώντας και πραγματοποιώντας τα Agile Ceremonies, η ομάδα θεωρείται Agile. Οι Agile ομάδες χαρακτηρίζονται από τις πρακτικές που ακολουθούν και τις τακτικές και τις στρατηγικές που χρησιμοποιούν ώστε να επιτύχουν τον σκοπό τους μέσω της συνεργασίας. Επομένως, τα Agile Ceremonies απλά διευκολύνουν την επικοινωνία μέσα στην ομάδα. [32]

3.5.1. Iteration Planning

Ο προγραμματισμός (Planning) του Iteration είναι μια από τις συναντήσεις της ομάδας στα πλαίσια του Agile, όπου θα καθοριστούν οι εργασίες (tasks) και τα user stories στα οποία θα εργαστεί η ομάδα κατά την διάρκεια αυτού του Iteration. Το Iteration Planning γίνεται την πρώτη μέρα του Iteration και συνιστάται να πραγματοποιείται μετά το Iteration Review/Retrospective, έτσι ώστε τυχόν αποτελέσματα από αυτές τις συζητήσεις να μπορούν να ληφθούν υπόψη κατά τον σχεδιασμό για το νέο iteration. Συνήθως, οι ομάδες θέτουν έναν στόχο του iteration βάσει του οποίου καθορίζονται ποια από τα tasks και user stories του product backlog θα υλοποιηθούν σε αυτό iteration.

Στο planning συμμετέχει όλη ομάδα ανάπτυξης και κάθε ρόλος συμβάλει διαφορετικά στην διαμόρφωση του. Ο Product Owner προσδιορίζει τα product backlog tasks και δίνει τις αντίστοιχες προτεραιότητες τους, καθώς επίσης προτείνει έναν στόχο για το συγκεκριμένο iteration. Τα μέλη της ομάδας καθορίζουν πόσα tasks από το product backlog προβλέπουν ότι θα είναι σε θέση να

ολοκληρώσουν και καθορίζουν τον τρόπο με τον οποίο θα τα παραδώσουν. Τέλος, ο Iteration Manager συνήθως διευκολύνει την διαδικασία για τον προγραμματισμό του iteration προκειμένου να διασφαλίσει ότι η συζήτηση είναι αποτελεσματική, ότι υπάρχει συμφωνία για τον στόχο του και ότι τα κατάλληλα tasks του product backlog περιλαμβάνονται στο iteration.

Το Iteration Planning χωρίζεται σε δύο μέρη, το «Scope» και το «Plan». Στο πρώτο μέρος «Scope», ο Product Owner έχει ετοιμάσει μια λίστα από user stories και τις προτεραιότητες τους και η ομάδα επιλέγει με ποια από αυτά, θα ασχοληθεί δεδομένου ότι μπορούν να ολοκληρωθούν κατά την διάρκεια του iteration. Στο δεύτερο μέρος «Plan», η ομάδα αναλύει λεπτομερώς πως θα υλοποιηθούν τα user stories και εντοπίζει τυχόν tasks που πρέπει να προηγηθούν πριν από άλλα tasks ώστε να ορίσει και την σειρά με την οποία θα εργαστεί κάθε μέλος της ομάδας. [33]

3.5.2. Daily Stand-up

Καθημερινά και συνήθως την ίδια ώρα, η ομάδα πραγματοποιεί μια ολιγόλεπτη συνάντηση με σκοπό να ενημερωθούν όλα τα μέλη σχετικά με πληροφορίες που είναι ζωτικής σημασίας για τον συντονισμό της ομάδας και την πρόοδο του iteration. Κάθε μέλος της ομάδας αναφέρει επιγραμματικά την εξέλιξη του task που έχει αναλάβει και τυχόν δυσκολίες που αντιμετωπίζει σχετικά με την ολοκλήρωση του. Η τρέχουσα κατάσταση των tasks αποτυπώνεται στο Iteration Board, από το οποίο κάθε ενδιαφερόμενος μπορεί να δει και να αξιολογήσει την πορεία του iteration. Το Iteration Board χρησιμοποιείται και κατά την διάρκεια του Daily Stand-up.

Το Daily Stand-up είθισται να έχει μέγιστη διάρκεια 15 λεπτά, ωστόσο μπορεί να προσαρμοστεί κατάλληλα ανάλογα με το μέγεθος της ομάδας. Για να μην υπερβαίνεται η καθορισμένη διάρκεια του Daily Stand-up, δεν αναλύονται λεπτομερώς θέματα που χρήζουν περαιτέρω συζήτηση και αντιμετώπιση, αλλά πραγματοποιείται μια επιπρόσθετη συνάντηση – συνήθως μετά το Daily Stand-up – ώστε τα εμπλεκόμενα μέλη να τα συζητήσουν εκτενέστερα. [34]

3.5.3. Iteration Refinement

Το Iteration Refinement, γνωστό και ως Iteration Grooming, είναι μια απαραίτητη διαδικασία στην οποία ο Product Owner και η υπόλοιπη ομάδα συνεργάζονται με σκοπό να διασφαλίσουν ότι τα user

stories του product backlog είναι πλήρως κατανοητά (shared understanding) και ότι έχει προσδιοριστεί ο εκτιμώμενος χρόνος υλοποίησης (user story estimation) και η αντίστοιχη προτεραιότητα τους (user story priority). Δεδομένου ότι το iterarion refinement είναι μια συνεχή διαδικασία και όχι μεμονωμένη συνάντηση, δεν υπάρχει προκαθορισμένη χρονική στιγμή που πραγματοποιείται κατά την διάρκεια του iteration. Ωστόσο, πολλές ομάδες προτιμούν να προγραμματίζουν την συνάντηση αυτή στην μέση του iteration ώστε να έχουν την δυνατότητα για βελτιώσεις πριν το επόμενο iteration planning.

Για τα νέα user stories, ο Product Owner (ή ο Business Analyst) είναι υπεύθυνος να τα παρουσιάσει λεπτομερώς στην ομάδα ώστε όλοι οι εμπλεκόμενοι να γνωρίζουν και να έχουν κατανοήσει με τον ίδιο τρόπο τις απαιτήσεις, δηλαδή τι θα κάνει και τι δεν θα κάνει το λογισμικό. Στην συνέχεια, η ομάδα ανάπτυξης και η ομάδα ελέγχου, θα πρέπει να λάβουν υπόψιν τους τις ανάγκες του user story και τυχόν τεχνικούς περιορισμούς ώστε να υπολογίσουν τον χρόνο υλοποίησης και ελέγχου βάσει της πολυπλοκότητας του, ενώ ο Product Owner καθορίζει την προτεραιότητα του βάσει της επιχειρηματικής του αξίας. Εάν η εκτίμηση χρόνου υλοποίησης και ελέγχου ενός user story υπερβαίνει τις 3-5 ανθρωποημέρες, τότε θα πρέπει να δημιουργηθούν δύο ή και περισσότερα μικρότερα user stories.

Επιπλέον στο iteration refinement, οι συμμετέχοντες επανεξετάζουν τις προτεραιότητες των user stories και των tasks που περιέχονται στο Product Backlog ώστε όσα συμπεριληφθούν στο επόμενο iteration να είναι αυτά με την υψηλότερη προτεραιότητα. Ακόμα, η ομάδα αξιολογεί εκ νέου user stories που έχουν αναλυθεί σε προηγούμενο refinement, εάν έχουν αλλάξει οι απαιτήσεις ή έχουν δοθεί περαιτέρω πληροφορίες σχετικά με την υλοποίηση. Σε αυτή την περίπτωση, η ομάδα πρέπει να εκτιμήσει ξανά τον αναμενόμενο χρόνο υλοποίησης καθώς και την προτεραιότητα του user story. [35]

3.5.4. Iteration Review & Retrospective

Το Iteration Review είναι η προτελευταία συνάντηση των Agile Ceremonies, στο οποίο συμμετέχουν η ομάδα ανάπτυξης και οι stakeholders. Ουσιαστικά, το Iteration Review είναι η αξιολόγηση του iteration σύμφωνα με το Iteration Planning που είχε γίνει στην αρχή. Πιο συγκεκριμένα, ο Product Owner ξεκινά την αξιολόγηση εξετάζοντας τον αρχικό στόχο του iteration και την τρέχουσα κατάσταση του στόχου και των user stories του συγκεκριμένου iteration. Η ομάδα ανάπτυξης ενημερώνει τους stakeholders για την νέα λειτουργικότητα που έχει προστεθεί στο λογισμικό παρουσιάζοντας την υλοποίηση κάθε ολοκληρωμένου user story. Κατά την παρουσίαση, οι stakeholders αξιολογούν την νέα λειτουργικότητα και με τα σχόλια τους κατευθύνουν κατάλληλα την ομάδα στην περίπτωση που χρειάζονται αλλαγές.

Επιπλέον, η ομάδα αναφέρει ποια user stories δεν κατάφεραν να ολοκληρωθούν καθώς και τους λόγους που δεν το κατέστησαν εφικτό.

Επομένως, αξιολογείται κατά πόσο επιτεύχθηκε ο στόχους του iteration που είχε τεθεί κατά το planning και συζητείται η πρόοδος ως προς τον στόχο του λογισμικού. Αυτή η συζήτηση, συνήθως, αποκαλύπτει εμπόδια ή κινδύνους που δεν είχαν εντοπιστεί νωρίτερα, εσφαλμένες υποθέσεις που έγιναν, ακόμη και προβλήματα που σχετίζονται με την σύσταση της ομάδας. Αυτά τα ευρήματα συχνά χρήζουν περαιτέρω ανάλυση, η οποία γίνεται κατά το Iteration Retrospective, και μπορεί, μελλοντικά, να οδηγήσουν σε βελτιώσεις στον σχεδιασμό και στην υλοποίηση του λογισμικού. [36]

Η τελευταία συνάντηση της ομάδας στα πλαίσια του Agile Ceremonies είναι το Iteration Retrospective στο οποίο συμμετέχει ο Product Owner, ο Agile Manager και η ομάδα ανάπτυξης (Devs & QAs). Το Iteration Retrospective είναι μια συνάντηση για την ανασκόπηση του τι πήγε σωστά και τι πήγε λάθος κατά την διάρκεια του iteration. Πρόκειται για μια συζήτηση στην οποία θα αναφερθούν τόσο οι επιτυχίες όσο και τα προβλήματα του iteration, θα εξεταστούν οι πρακτικές που ακολουθήθηκαν, θα προσδιοριστούν τυχόν προβλήματα και τρόποι επίλυσης τους καθώς και προτεινόμενες βελτιώσεις. Το Iteration Retrospective δεν πραγματοποιείται ώστε τα μέλη να αναφέρουν μόνο τα προβλήματα και τα παράπονα τους, αλλά μέσω της συζήτησης η ομάδα μπορεί να βρει δημιουργικές λύσεις και να αναπτύξει ένα σχέδιο δράσης ώστε να τα ξεπεράσει. Επιπλέον, μέσω του retrospective, η ομάδα αναγνωρίζει τις πρακτικές που χρησιμοποιεί και λειτουργεί σωστά ώστε να συνεχίσει να εστιάζει σε αυτές. Η συνεχής βελτίωση είναι αυτή που συντηρεί μια Agile ομάδα και την οδηγεί στην ανάπτυξη και τα iteration retrospectives αποτελούν βασικό της μέρος. [37]

3.6. User Stories

Για την υλοποίηση ενός έργου, οι stakeholders θα πρέπει να επικοινωνήσουν τις ανάγκες και τις απαιτήσεις τους στην ομάδα ανάπτυξης, ωστόσο η επιτυχία του έργου εξαρτάται από αυτές τις πληροφορίες, οι οποίες πρέπει να είναι ακριβείς και σαφείς. Δεδομένου ότι μεσολαβούν αρκετοί άνθρωποι, όπως stakeholders, αναλυτές, προγραμματιστές στον καθορισμό, την διατύπωση, την διανομή και την υλοποίηση των απαιτήσεων του λογισμικού, η επικοινωνία τους μπορεί να αποτελέσει πρόβλημα. Εάν μία από τις δύο πλευρές των συμβαλλομένων (stakeholders ή ομάδα ανάπτυξης) «κυριαρχεί» στην επικοινωνία, τότε σίγουρα το έργο θα αποτύχει. Όταν η επιχειρηματική πλευρά κυριαρχεί, τότε επιβάλλει ημερομηνίες ή απαιτήσεις από την ομάδα ανάπτυξης χωρίς λαμβάνουν υπόψιν τους εάν η ομάδα μπορεί να ανταπεξέλθει ή κατανοεί ακριβώς τι χρειάζεται. Αντιθέτως όταν η ομάδα ανάπτυξης κυριαρχεί, τότε η

τεχνική ορολογία αντικαθιστά τη γλώσσα της επιχείρησης και οι προγραμματιστές χάνουν την ευκαιρία να μάθουν τι χρειάζεται να υλοποιηθεί. Γι' αυτό χρειάζεται ένας κοινός τρόπος να συνεργάζονται τα δύο μέρη και καμία πλευρά να μην κυριαρχεί, έτσι ξεκινά μια σειρά συζητήσεων, ενίοτε μακροσκελής, κατά την διάρκεια των οποίων καθορίζονται και επικοινωνούνται οι απαιτήσεις του λογισμικού. Σε αυτό το σημείο, δημιουργούνται τα user stories.

Το User Story είναι ένα μικρό μέρος της συνολικής δουλειάς που χρειάζεται να γίνει για την επίτευξη του τελικού στόχου, δηλαδή δεν αποτελεί μια εξ' ολοκλήρου αυτόνομη λειτουργικότητα και συνήθως εκφράζεται από την οπτική του τελικού χρήστη. Ουσιαστικά, είναι μια απλή και ανεπίσημη περιγραφή μέρους της λειτουργικότητας του λογισμικού και ο σκοπός του είναι να διατυπώσει πώς μια δυνατότητα λογισμικού θα προσφέρει αξία στον τελικό χρήστη. Κάθε user story έχει τρία χαρακτηριστικά, τα οποία ονομάζονται τα «3 C's» :

- **Card:** Σύντομη περιγραφή της επιθυμητής λειτουργικότητας
- **Conversation:** Συζητήσεις για την λειτουργικότητα που θα εμπλουτίσουν με λεπτομέρειες το user story
- **Confirmation:** Τα acceptance criteria που τεκμηριώνουν τις παραπάνω λεπτομέρειες και μπορούν να χρησιμοποιηθούν για επιβεβαιώσουν ότι το user story έχει ολοκληρωθεί πλήρως

Τα user stories, αρχικά, γράφονται σε απλή γλώσσα, χωρίς να χρησιμοποιούν τεχνική ορολογία ή τεχνικές προδιαγραφές, για να αναφέρουν το επιθυμητό αποτέλεσμα και ύστερα από τις συζητήσεις με την ομάδα προστίθενται οι επιχειρησιακές λεπτομέρειες και οι τεχνικές απαιτήσεις.

Συνήθως, στα user stories καταγράφονται μόνο τα βασικά στοιχεία όπως «για ποιον είναι;», «τι περιμένει από το σύστημα» και προαιρετικά «γιατί είναι σημαντικό» γι' αυτό και η πιο γνωστή μορφή τους είναι «Ως [χρήστης], θέλω να [επιτευχθεί ο στόχος], ώστε να μπορέσω [προκύπτουσα επιχειρηματική αξία για τον χρήστη]». Ωστόσο καλύπτοντας μόνο αυτές τις γενικές πληροφορίες, η ομάδα ανάπτυξης δεν μπορεί να ξεκινήσει την υλοποίηση, ούτε η ομάδα ελέγχου δεν μπορεί να διασφαλίσει την ποιότητα. Γι΄αυτό πρέπει να προστεθούν οι τεχνικές λεπτομέρειες καθώς και τα acceptance criteria, που θα προκύψουν από τις συζητήσεις και θα καταγραφούν στο user user. Επιπλέον, κάθε user story θα πρέπει να υλοποιείται και να ελέγχεται πλήρως στην διάρκεια ενός iteration, επομένως εάν το story size είναι αρκετά μεγάλο, θα πρέπει να διασπαστεί σε δύο ή περισσότερα μικρότερα user stories.

Εφόσον το user story είναι έτοιμο για υλοποίηση (και βρίσκεται στο product backlog), η ομάδα ανάπτυξης το εντάσσει σε ένα από τα επόμενα iterations, ενώ κατά την διάρκεια του planning, το user story διασπάται σε επιμέρους tasks, ώστε να είναι πιο διαχειρίσιμο κατά την ανάπτυξη του. Κατά το

iteration, οι προγραμματιστές ασχολούνται με την υλοποίηση του user story, ενώ η ομάδα ελέγχου μπορεί να προετοιμάσει τα χειροκίνητα ή αυτοματοποιημένα test cases, τα οποία θα εκτελεστούν όταν ολοκληρωθεί η ανάπτυξη. Για να θεωρηθεί ολοκληρωμένο ένα user story, πρέπει να έχει ολοκληρωθεί τόσο η ανάπτυξη και ο έλεγχος του αλλά και να έχουν διορθωθεί τυχόν σφάλματα που βρέθηκαν κατά την φάση του ελέγχου και να έχουν επανελεγχθεί. Εάν δεν πληρούνται αυτές οι προϋποθέσεις, το user story μεταφέρεται στο επόμενο iteration. [38] [39]

3.7. Test Cases

Στα πλαίσια της Agile ανάπτυξης λογισμικού, το σημείο αναφοράς των Test Cases είναι το εκάστοτε User Story, καθώς προκύπτουν από τα acceptance criteria του. Τα acceptance criteria αποτελούν τις προϋποθέσεις που πρέπει να πληροί η υλοποίηση του user story για να γίνει αποδεκτό από τους stakeholders. Ουσιαστικά, το Test Case είναι ένα σύνολο ενεργειών που πρέπει να πραγματοποιηθούν στο λογισμικό ώστε να διασφαλίσουν ότι ικανοποιούνται οι απαιτήσεις και η λειτουργικότητα είναι η αναμενόμενη. Κάθε test case αναφέρει λεπτομερώς τα βήματα που πρέπει να εκτελεστούν, τα test data που θα χρησιμοποιηθούν, τις προϋποθέσεις καθώς και το αναμενόμενο αποτέλεσμα.

Κατά την δημιουργία του test case, η ομάδα ελέγχου πρέπει να καθορίσει συγκεκριμένες μεταβλητές (test data), ώστε μετά την εκτέλεση του να μπορέσει να συγκρίνει τα πραγματικά (actual) με τα αναμενόμενα (expected) αποτελέσματα και να καταλήξει εάν το feature που είναι υπό έλεγχο ικανοποιεί τα acceptance criteria. Συστήνεται η δημιουργία των test cases να γίνεται όσο πιο νωρίτερα κατά την διάρκεια του iteration – ακόμη και πριν την υλοποίηση του user story – ώστε να εντοπιστούν έγκαιρα τυχόν προβλήματα χρηστικότητας ή κενά στην ανάλυση και το σχεδιασμό του feature. Όσο το προϊόν αναπτύσσεται, τόσο θα αυξάνεται ο αριθμός των test cases, γι’ αυτό είναι απαραίτητη η προτεραιοποίηση τους. Η εκτέλεση όλων των test cases απαιτεί πολύ χρόνο και ο έλεγχος όλης της test suite μετά από κάθε build είναι πρακτικά αδύνατο. Τέτοιου είδους προβλήματα και προκλήσεις μπορεί να ξεπεραστούν εύκολα με την σωστή προτεραιοποίηση. [40]

4. Υλοποίηση Εφαρμογής & Οδηγίες Χρήσης Smart Test Cases Tool

4.1. Σκοπός της Εφαρμογής

Στην εποχή της σύγχρονης τεχνολογίας, οι ομάδες ανάπτυξης λογισμικού, που ακολουθούν Agile μεθοδολογία, επαναλαμβάνουν τον ίδιο κύκλο εργασιών σε κάθε iteration, που βασίζονται στα εκάστοτε user stories, όπως έχουν προκύψει από την ανάλυση του έργου, την ανάπτυξη του κώδικα, και εν συνεχεία στα αντίστοιχα test cases που απαιτούνται ανά περίπτωση για την διασφάλιση της ποιότητας του.

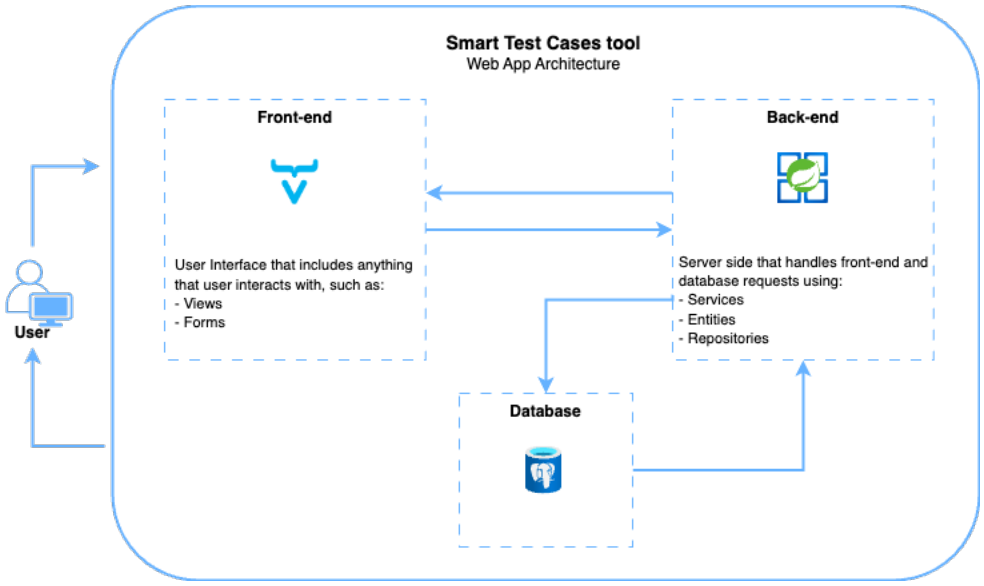
Όπως αναλύθηκε στα προηγούμενα κεφάλαια, τα user stories περιγράφουν τι πρέπει να υλοποιηθεί από τους προγραμματιστές και αναφέρονται λεπτομερώς στα στοιχεία που θα υπάρχουν στην εφαρμογή καθώς και στην αναμενόμενη λειτουργικότητα τους. Τα test cases με την σειρά τους καλύπτουν σενάρια τόσο για τους επιχειρησιακούς ελέγχους όσο και στους λογικούς ελέγχους. Όσο αναφορά την φάση του ελέγχου του λογισμικού, η ομάδα διασφάλισης ποιότητας καλείται να κατανοήσει τα user stories και να γράψει τα αντίστοιχα test cases που προκύπτουν ανά πεδίο και acceptance criterion και στην συνέχεια να τα εκτελέσει μέσω manual ή automation testing. Τα test cases που αφορούν τους λογικούς ελέγχους των πεδίων επαναλαμβάνονται σε κάθε user story άρα κάθε φορά θα πρέπει να δημιουργούνται οι ίδιοι έλεγχοι για όμοια πεδία. Η διαδικασία αυτή είναι αρκετά χρονοβόρα και επαναλαμβανόμενη καθώς επίσης έχουν παρατηρηθεί περιπτώσεις που τα βασικά σενάρια έχουν παραληφθεί από λάθος και στο τέλος δημιουργούν σφάλματα κατά την χρήση της εφαρμογής.

Για το σκοπό αυτό, σχεδιάστηκε και αναπτύχθηκε η διαδικτυακή εφαρμογή “Smart Test Cases tool” όπου δημιουργούνται αυτόματα τα test cases που πρέπει να εκτελεστούν για κάθε user story, τουλάχιστον για τους λογικούς ελέγχους και δίνει την δυνατότητα για πιο παραγωγική εργασία τόσο της ομάδας ελέγχου όσο και της ομάδας ανάπτυξης. Πρόκειται για μία εφαρμογή, όπου ο αναλυτής (BA) έχει την δυνατότητα να προσθέσει projects, να δημιουργήσει τα users stories με τα επιθυμητά πεδία και να προσδιορίσει τους περιορισμούς και τις επιτρεπτές ενέργειες ανά πεδίο. Κατά την φάση του ελέγχου, οι περιορισμοί αυτοί ανάγονται σε test cases, ωστόσο η ομάδα ελέγχου δεν χρειάζεται να καταγράψει τα test cases που αφορούν τους λογικούς ελέγχους κάθε πεδίου για κάθε user story, αλλά να τα δημιουργήσει αυτόματα μέσω της εφαρμογής. Επιπλέον, τα εν λόγω test cases μπορούν να χρησιμοποιηθούν και από τους developers για να επιβεβαιώσουν την σωστή λειτουργικότητα του κώδικα τους αλλά και για να βελτιώσουν την ποιότητα του. Τέλος, ακόμη μια βασική λειτουργία της εφαρμογής είναι η διαχείριση των έργων (projects), των user stories και των test cases.

4.2. Τεχνική Προσέγγιση

Η ανάπτυξη της εφαρμογής έγινε με την γλώσσα προγραμματισμού Java χρησιμοποιώντας την open-source έκδοση του Vaadin framework[1] και το περιβάλλον ανάπτυξης (Integrated Development Environment – IDE) IntelliJ[2]. Το Vaadin framework χρησιμοποιείται για την υλοποίηση διαδικτυακών εφαρμογών, παρέχοντας μια ποικιλία web-components για το front-end μέρος και συνδυάζοντας το Spring Boot[3] για το back-end μέρος. Με το Vaadin δίνεται η δυνατότητα της χρήσης μιας ποικιλίας web-components με τα οποία δημιουργείται το User Interface (UI) μόνο με την γλώσσα Java και χωρίς να χρειάζεται η HTML περαιτέρω. Επιπλέον, το Spring Boot framework παρέχει ένα ολοκληρωμένο μοντέλο προγραμματισμού και διαμόρφωσης για σύγχρονες εφαρμογές για κάθε είδους πλατφόρμα ανάπτυξης υποστηρίζοντας την επιχειρηματική λογική σε επίπεδο εφαρμογής και χωρίς περιορισμούς με τα διάφορα περιβάλλοντα ανάπτυξης (Integrated Development Environment – IDE). Για την αποθήκευση και διαχείριση του πηγαίου κώδικα, όπως επίσης και για τον διαμοιρασμό του, χρησιμοποιήθηκε το GitHub[4] το οποίο παρέχει κατανεμημένο έλεγχο εκδόσεων του λογισμικού (Version Control System – VCS), ο οποίος είναι διαθέσιμος στο link.

Τα βασικά χαρακτηριστικά της αρχιτεκτονικής δομής της εφαρμογής απεικονίζονται στο παρακάτω διάγραμμα:



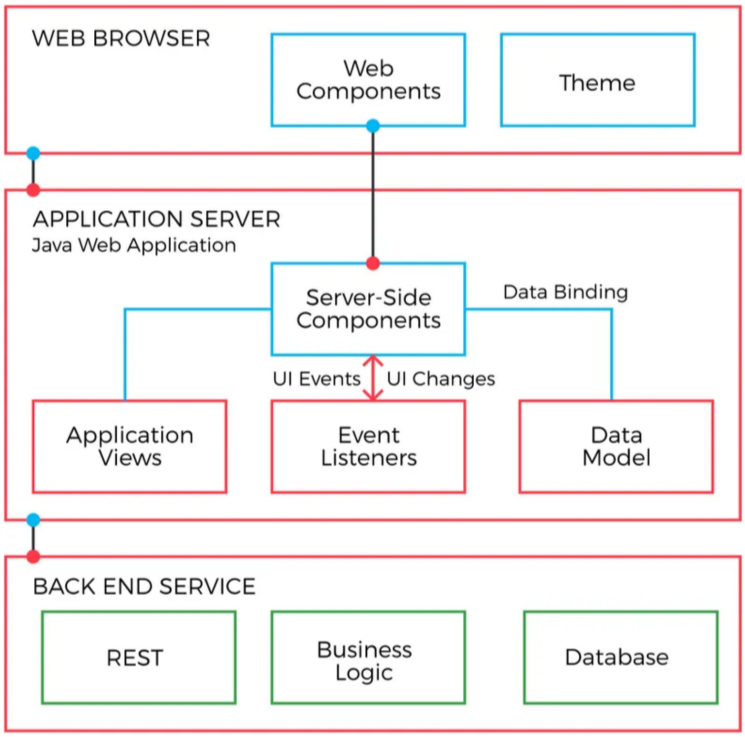
[1] https://vaadin.com/
[2] https://www.jetbrains.com/idea/
[3] https://spring.io/projects/spring-boot
[4] https://github.com/

4.2.1. Vaadin Framework

Το Vaadin framework επιτρέπει μία server-side Java εφαρμογή να δημιουργήσει ένα user interface (UI) από Java components. Αυτά τα components συνδέονται με web-components που εκτελούνται στον browser. Το Vaadin Flow διαχειρίζεται την αναμετάδοση της αλληλεπίδρασης του χρήστη με την server-side εφαρμογή, η οποία με την σειρά της μπορεί να τα χειριστεί με διάφορα event listeners. Οι οθόνες της εφαρμογής και τα components τους χρησιμοποιούνται συνήθως για την εμφάνιση και την αποδοχή της εισαγωγής δεδομένων. Αυτά τα δεδομένα αποθηκεύονται σε ένα back-end service, όπως η βάση δεδομένων. Η λογική της εφαρμογής δημιουργείται χρησιμοποιώντας framework εφαρμογών, όπως το Spring.

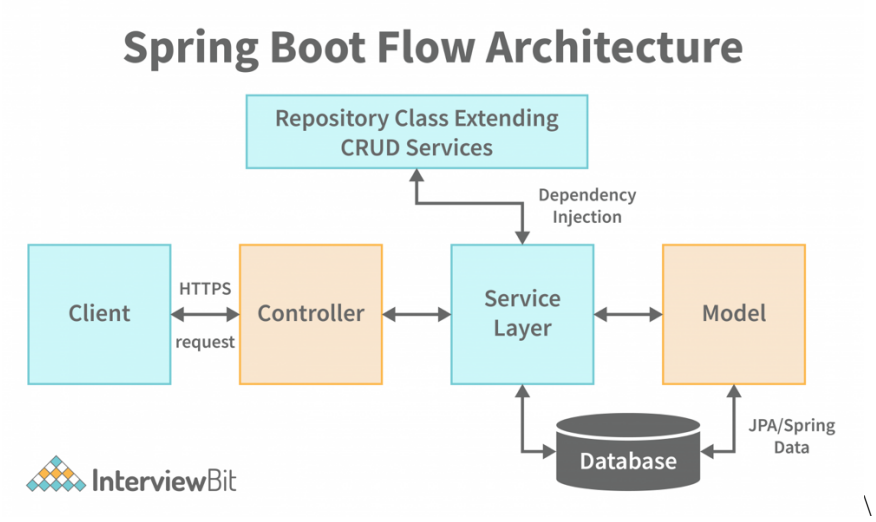
Τα βασικά χαρακτηριστικά του Vaadin Flow είναι:

- Η αρχιτεκτονική που επιτρέπει να την εύκολη επικοινωνία του client-side μέρος με το server-side και με αυτόν τον τρόπο εστιάζει περισσότερο στο UI
- Ένα σύνολο UI components υψηλής ποιότητας που επικεντρώνονται τόσο στην εμπειρία του χρήστη όσο και του προγραμματιστή
- Το Data Binding API για την σύνδεση των components με το back-end μέρος
- Το Router API για την δημιουργία σελίδων με ιεραρχική δομή για την πλοήγηση του χρήστη



4.2.2. Spring Boot Framework

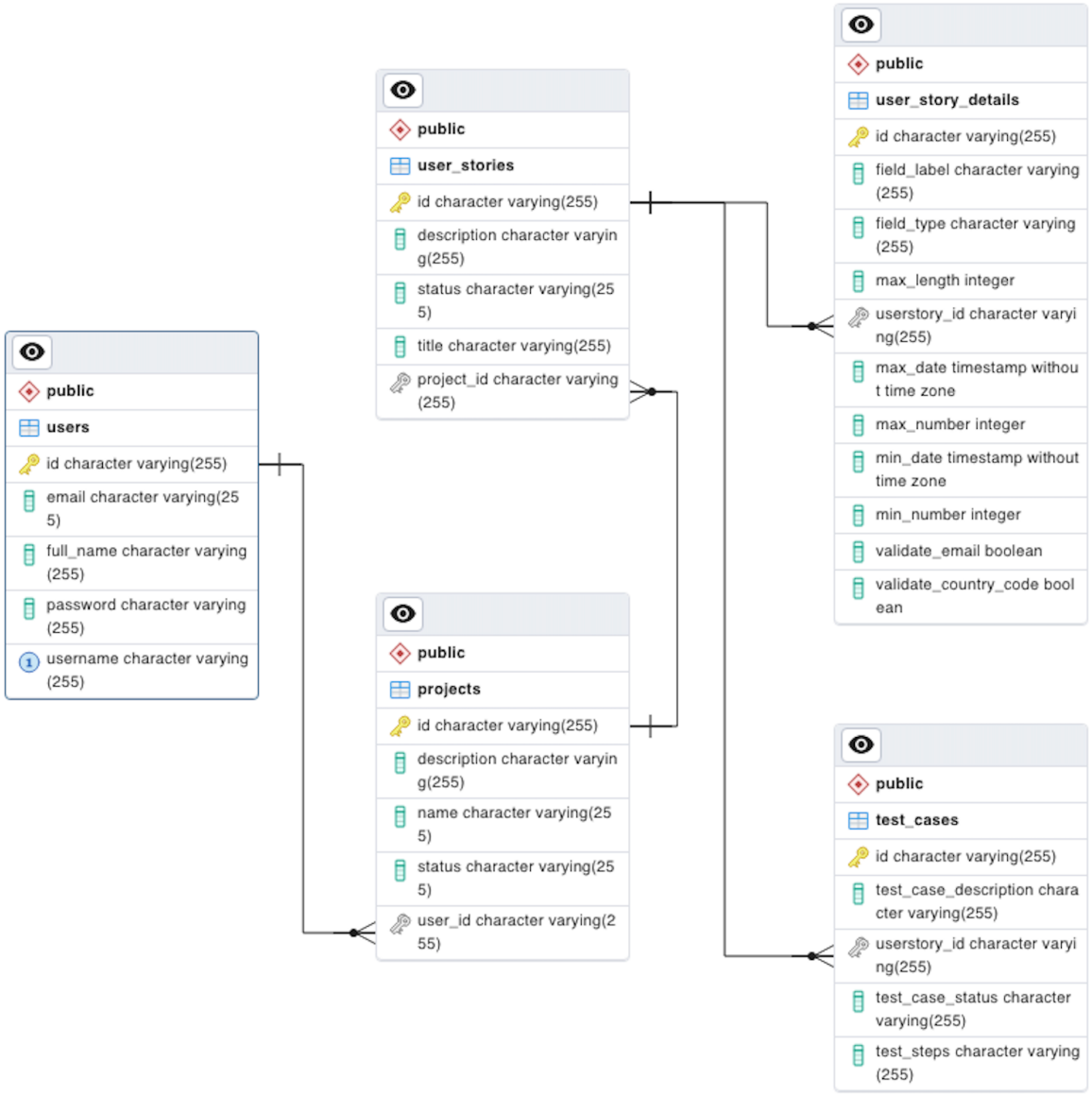
Το Spring είναι ένα module που αναπτύχθηκε στα πλαίσια του Spring Framework και διευκολύνει την δημιουργία μεμονωμένων εφαρμογών που βασίζονται στο Spring και επιπλέον έχει σχεδιαστεί έτσι ώστε να εκτελείται χωρίς να χρειάζονται XML και annoted-based ρυθμίσεις. Το Spring Boot ακολουθεί μία πολυεπίπεδη αρχιτεκτονική στην οποία κάθε επίπεδο επικοινωνεί με τα άλλα επίπεδα (που είναι πάνω και κάτω στην ιεραρχική σειρά). Η βασική δομή αποτελείται από τα Entities (ή Models), τα Repositories, τα Services και τους Controllers, όπως φαίνεται στο παρακάτω διάγραμμα.



- **Controller:** Το επίπεδο του Controller διαχειρίζεται τα αιτήματα HTTP, μεταφράζει τις JSON παραμέτρους σε αντικείμενα και προσδιορίζει τα τελικά API endpoints. Ο κύριος στόχος του Controller είναι να παρέχει responses στον χρήστη.
- **Service:** Το επίπεδο του Service υλοποιεί την επιχειρησιακή λογική, αποτελείται από κλάσεις ενώ λειτουργεί με τα δεδομένα που προέρχονται από την βάση δεδομένων. Επίσης, εκτελεί αυθεντικοποίηση και επικύρωση των δεδομένων του επιπέδου του Model, καθώς και κάθε είδους υπολογισμός. Γι' αυτό, το επίπεδο του Service επικοινωνεί με το επίπεδο του Repository.
- **Repository:** Το επίπεδο του Repository (γνωστό και ως Data Access Object – DAO) έχει την πρόσβαση στα δεδομένα (Query) από την βάσης δεδομένων και τα παρέχει στο επίπεδο του Service. Αυτό το επίπεδο μεταφράζει τα επιχειρησιακά αντικείμενα σε εγγραφές της βάσης δεδομένων.

4.2.3. Βάση Δεδομένων

Η αποθήκευση και η διαχείριση των δεδομένων υλοποιήθηκε με open-source σχεσιακή βάση δεδομένων PostgreSQL[5], η οποία χρησιμοποιείται σε web εφαρμογές και προσφέρει την μέγιστη ευελιξία και λειτουργικότητα. Στο παρακάτω διάγραμμα, απεικονίζονται αναλυτικά οι πίνακες του σχήματος της βάσης δεδομένων όπως διαμορφώνονται τα πεδία, τα primary και τα foreign keys, καθώς και οι σχέσεις που αναπτύσσονται μεταξύ τους.

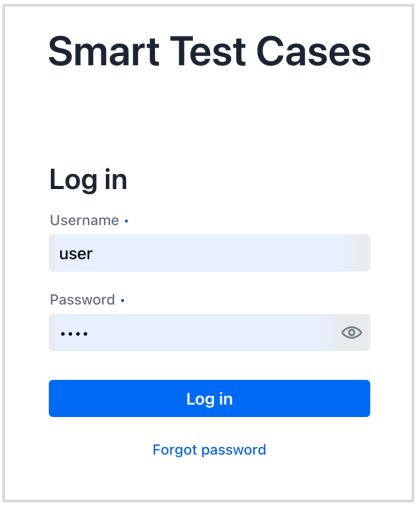


[5] https://www.postgresql.org/

4.3. Οδηγίες Χρήσης

4.3.1. Login

Μόλις ο χρήστης μεταβεί στην σελίδα της εφαρμογής από οποιονδήποτε browser, εμφανίζεται η παρακάτω οθόνη στην οποία θα πρέπει να συνδεθεί χρησιμοποιώντας τα credentials (**Username** και **Password**) που του έχουν δοθεί για την εφαρμογή Smart Test Cases και να πατήσει το κουμπί **Login**. Στην περίπτωση που ο χρήστης αντιμετωπίζει οποιοδήποτε πρόβλημα με τα credentials του, μπορεί να πατήσει το κουμπί **Forgot password** και να ακολουθήσει τις οδηγίες για να ανακτήσει τους κωδικούς του.



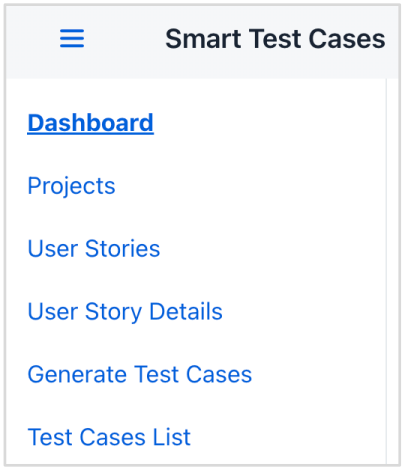
Μετά την επιτυχή σύνδεση του, ο χρήστης μεταφέρεται στην σελίδα του Dashboard. Εάν τα credentials δεν είναι σωστά, η εφαρμογή ενημερώνει τον χρήστη κατάλληλα ώστε να εισάγει τα στοιχεία του εκ νέου.

4.3.1.1. Υλοποίηση Login – Spring Security

Η υλοποίηση της διαδικασίας επαλήθευσης ταυτότητας (Authentication) και εξουσιοδότησης (Authorization) των χρηστών έγινε με το Spring Security[6], που είναι ενσωματωμένο με το Vaadin Flow. Το Vaadin Flow περιέχει διάφορες μεθόδους ασφαλείας που ενεργοποιούν έναν μηχανισμό ελέγχου πρόσβασης βάσει των ρυθμίσεων του Spring Security. Ο μηχανισμός αυτός επιτρέπει ή αποτρέπει την ευέλικτα πρόσβαση στην εφαρμογή.

4.3.2. Menu

Δεδομένου ότι ο χρήστης έχει επιτυχώς εισέλθει στην εφαρμογή, στα αριστερά της σελίδας βρίσκεται το μενού από το οποίο μπορεί να δει όλες τις διαθέσιμες λειτουργίες και να πλοηγηθεί σε αυτές, όπως φαίνονται παρακάτω. Πατώντας το κουμπί με τις **τρεις γραμμές**, το οποίο βρίσκεται πάνω αριστερά, μπορεί να ανοίξει ή να κλείσει το πτυσσόμενο μενού αντίστοιχα.

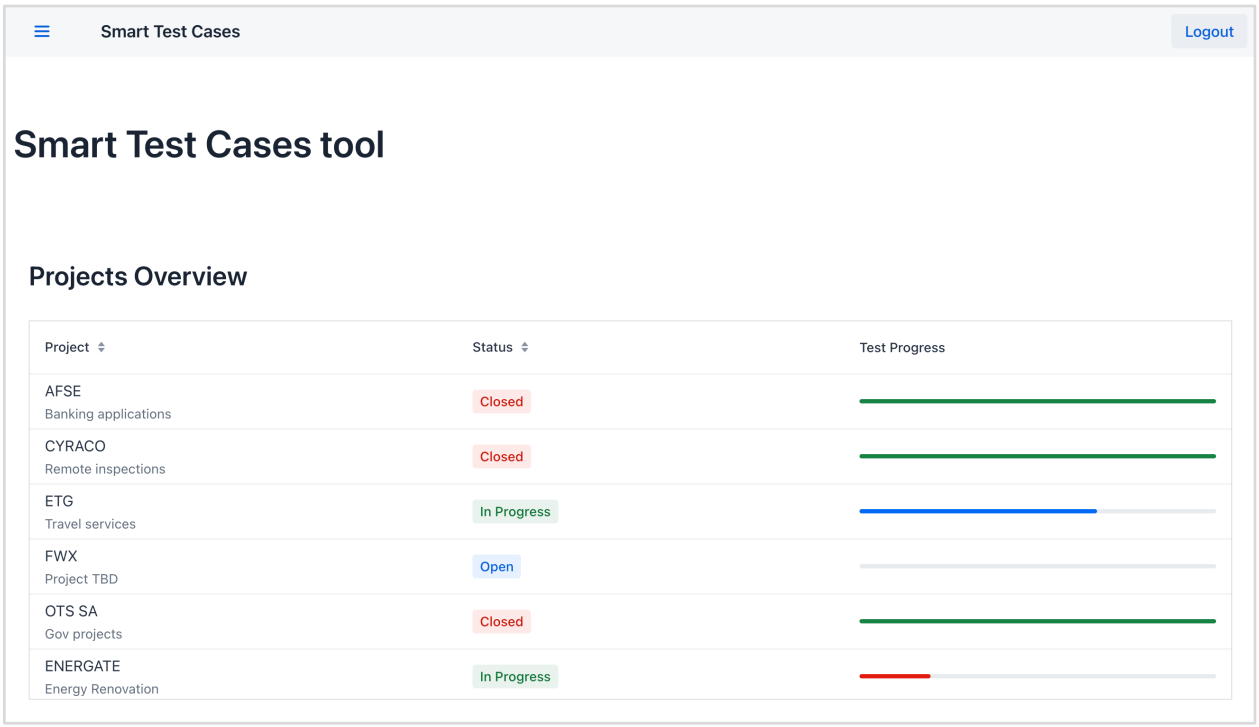


[6] https://spring.io/projects/spring-security

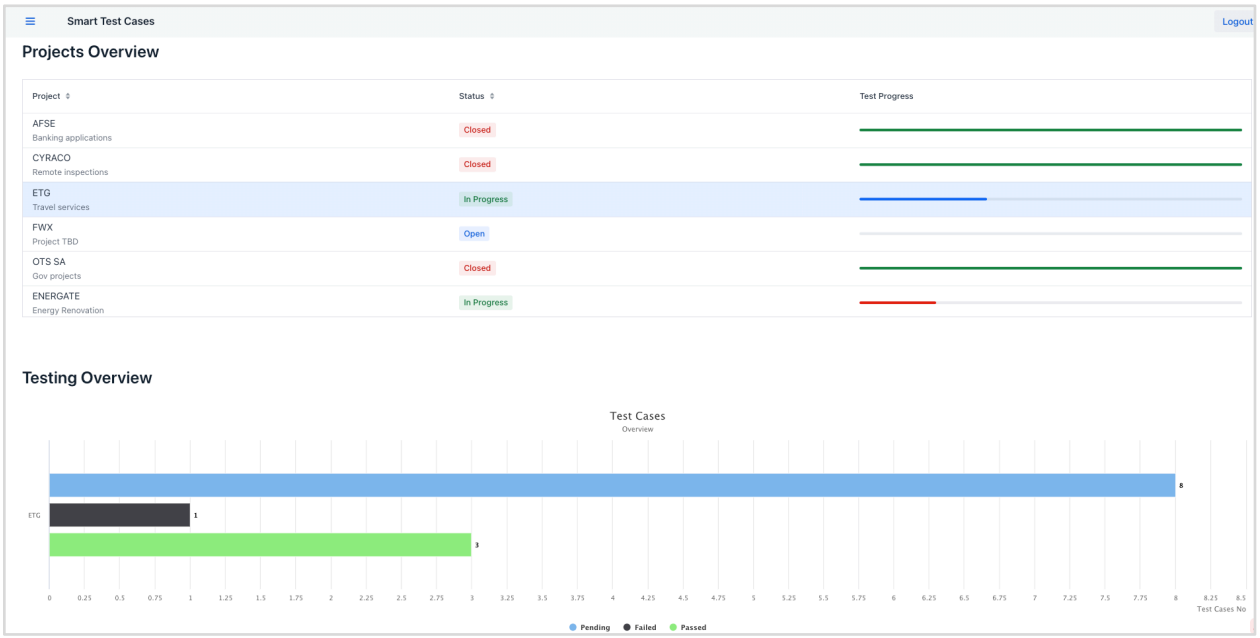
4.3.3. Dashboard

Εφόσον ο χρήστης ανακατευθυνθεί στην σελίδα του **Dashboard**, έχει την δυνατότητα να δει κάποια στατιστικά στοιχεία που αφορούν τα projects του, όπως το **Project Name**, το **Status** καθώς και το **Test Progress**.

Σχετικά με την στήλη **Test Progress**, εμφανίζεται μία μπάρα η οποία δείχνει ποσοστιαία την πρόοδο του ελέγχου, υπολογίζοντας πόσα test cases έχουν εκτελεστεί (δηλαδή το status τους δεν είναι *Pending*). Στην περίπτωση που όλα τα test cases έχουν εκτελεστεί επιτυχώς τότε η μπάρα έχει πράσινο χρώμα, εάν έχει εκτελεστεί το 30% των test cases του project, η μπάρα έχει κόκκινο χρώμα, ενώ εάν τα εκτελεσμένα test cases είναι πάνω από 30% αλλά κάτω από 70% τότε η μπάρα έχει χρώμα μπλε. Επιπλέον, εάν το status του project είναι *Open*, δηλαδή δεν έχει ξεκινήσει ακόμη, τότε η μπάρα δεν έχει κανένα χρώμα.



Επιπλέον, ο χρήστης επιλέγοντας οποιοδήποτε project, εμφανίζεται ένα διάγραμμα το οποίο δείχνει λεπτομερώς το σύνολο των test cases ανά status για το επιλεγμένο project, όπως φαίνεται παρακάτω.

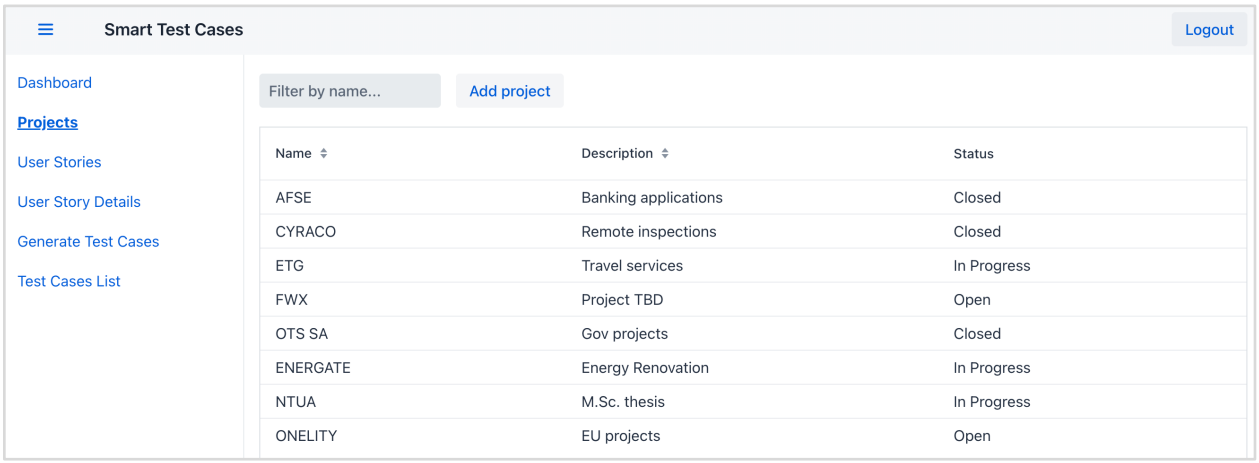


4.3.4. Projects

Στην ενότητα **Projects**, ο χρήστης έχει την δυνατότητα να δει και να διαχειριστεί τα projects που είναι αποθηκευμένα στην βάση δεδομένων, καθώς και να προσθέσει νέα.

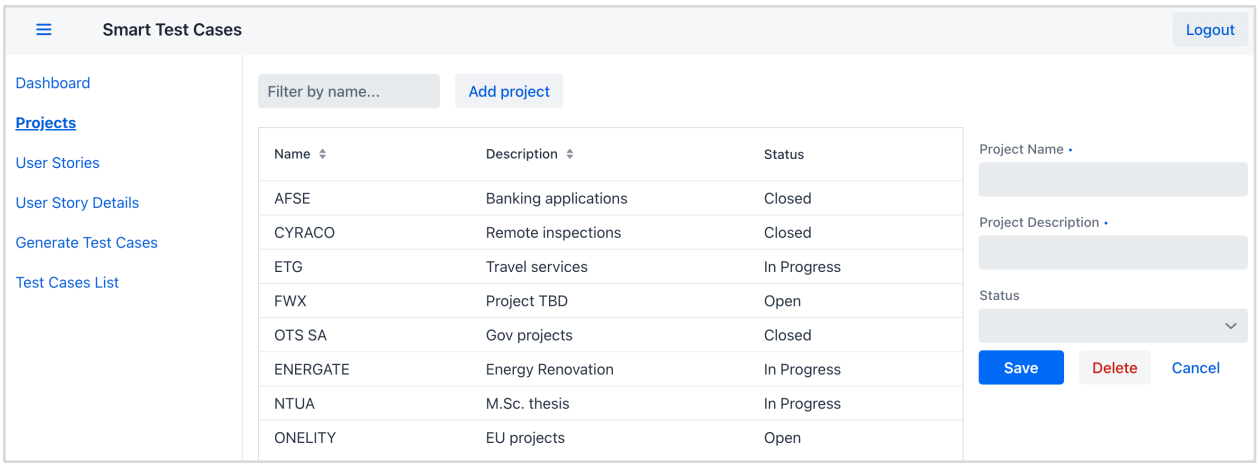
4.3.4.1. List

Πάνω από την λίστα των projects, υπάρχει το πεδίο **Filter by name**, με το οποίο ο χρήστης μπορεί εύκολα να αναζητήσει το επιθυμητό project. Καθώς ο χρήστης πληκτρολογεί στο πεδίο, τότε αυτόματα τα αποτελέσματα φιλτράρονται βάσει της τιμής που έχει δοθεί από τον χρήστη. Επιπλέον, ο χρήστης έχει την δυνατότητα να εμφανίσει τα αποτελέσματα ταξινομημένα βάσει των στηλών **Name** και **Description**.

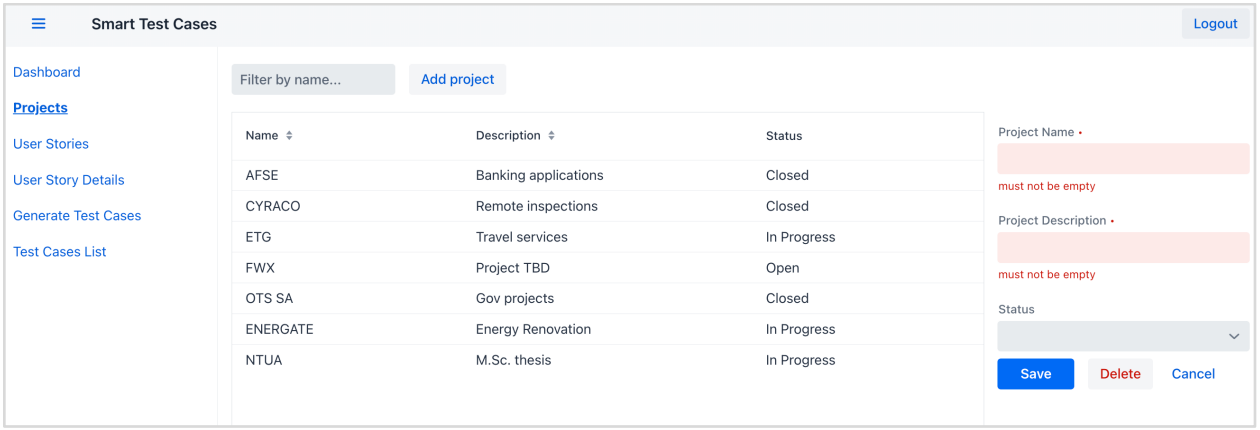


4.3.4.2. Add New Project

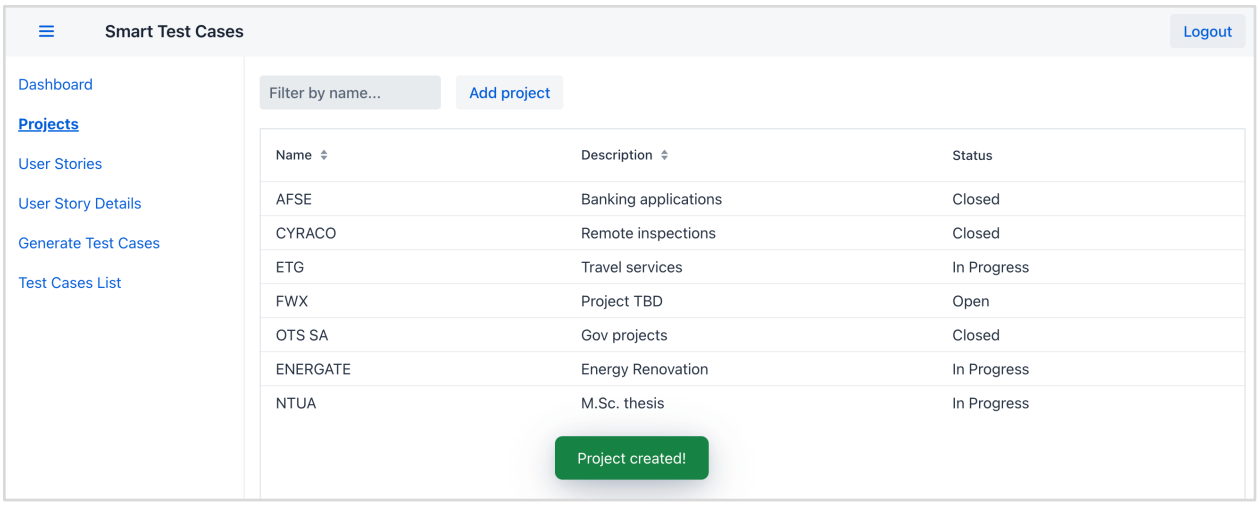
Όταν πατηθεί το κουμπί **Add project**, στα δεξιά της οθόνης εμφανίζεται μία φόρμα, στην οποία ο χρήστης μπορεί συμπληρώσει τα απαραίτητα πεδία (**Project Name**, **Project Description** και **Status**) και με το κουμπί **Save** να αποθηκεύσει το νέο project.



Ο χρήστης πρέπει να συμπληρώσει όλα τα υποχρεωτικά πεδία της φόρμας που εμφανίζονται με αστερίσκο (*) για να μπορέσει να δημιουργήσει ένα νέο project. Στην περίπτωση τα υποχρεωτικά πεδία δεν έχουν συμπληρωθεί και πατηθεί το κουμπί **Save**, τότε τα κατάλληλα μηνύματα ενημερώνουν τον χρήστη, όπως στην παρακάτω εικόνα.

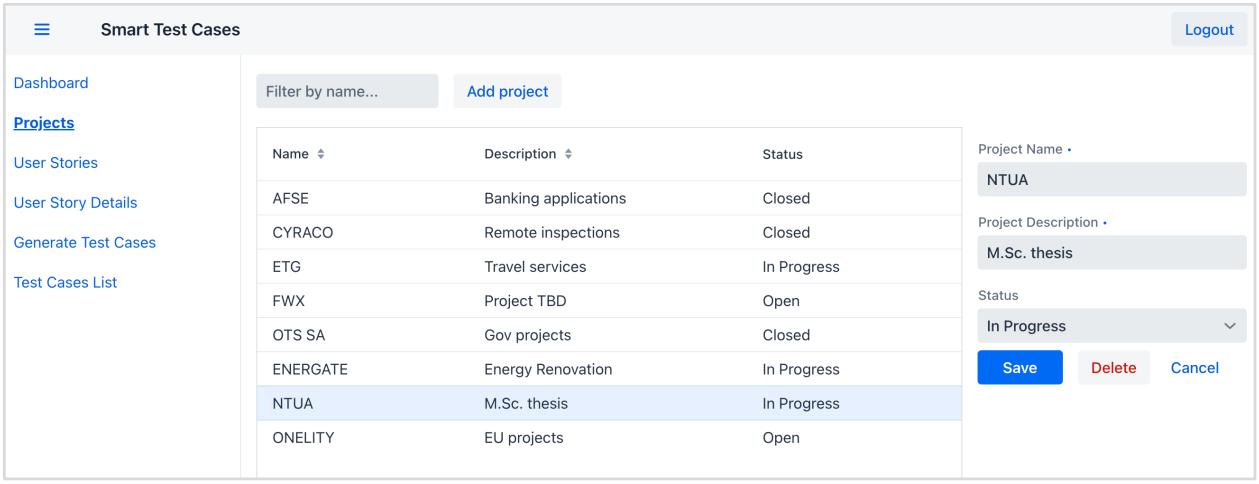


Εφόσον όλα τα υποχρεωτικά πεδία είναι συμπληρωμένα και το νέο project δημιουργηθεί επιτυχώς, ο χρήστης ενημερώνεται με το αντίστοιχο μήνυμα και η λίστα ανανεώνεται αυτόματα, ενώ η φόρμα δεν είναι πλέον ορατή στην σελίδα.

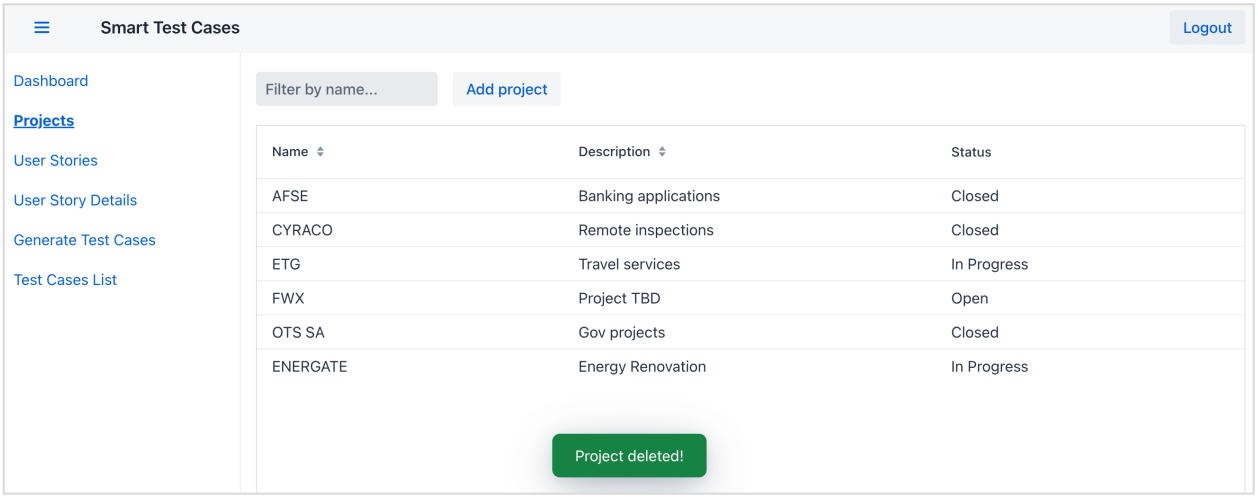


4.3.4.3. Edit & Delete Project

Επιλέγοντας ένα project της λίστας, τα στοιχεία του και οι επιτρεπτές ενέργειες εμφανίζονται στην φόρμα στα δεξιά της οθόνης.



Έτσι, ο χρήστης έχει την δυνατότητα είτε να επεξεργαστεί τα στοιχεία του, και πατώντας το κουμπί **Save** να αποθηκεύσει τις αλλαγές, είτε να διαγράψει το επιλεγμένο project. Σε κάθε περίπτωση ο χρήστης ενημερώνεται με το κατάλληλο ενημερωτικό μήνυμα. Για παράδειγμα, παρακάτω φαίνεται η περίπτωση διαγραφής ενός project.

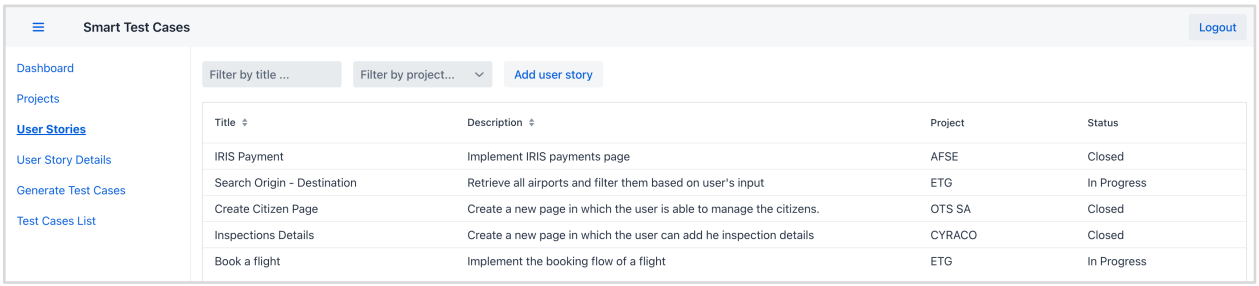


4.3.5. User Stories

Στην ενότητα **User Stories**, ο χρήστης έχει την δυνατότητα να διαχειριστεί τα user stories που είναι ήδη αποθηκευμένα για κάθε project, να τα επεξεργαστεί καθώς και να δημιουργήσει νέα.

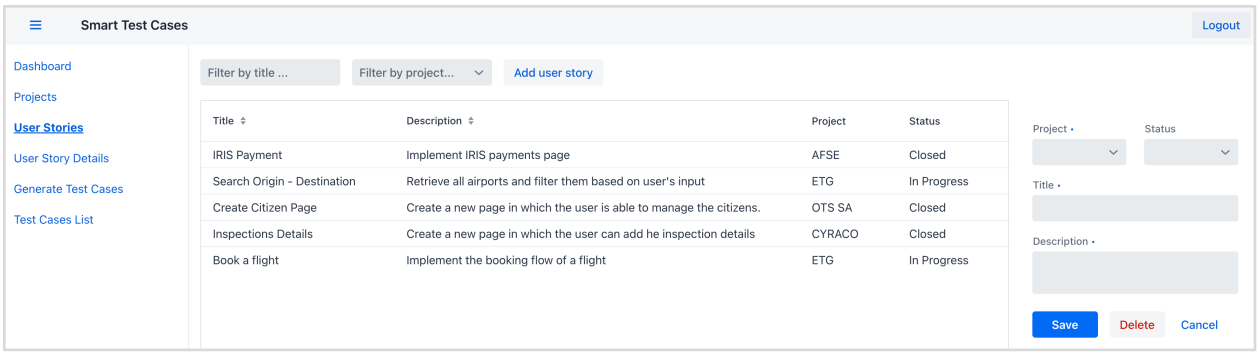
4.3.5.1. List

Πάνω από την λίστα των user stories, υπάρχει το πεδίο **Filter by title** και η λίστα **Filter by project**, με το οποία ο χρήστης μπορεί εύκολα να αναζητήσει το επιθυμητό user story ή να τα φιλτράρει ανά project. Επιπλέον, ο χρήστης έχει την δυνατότητα να εμφανίσει τα αποτελέσματα ταξινομημένα βάσει των στηλών **Title** και **Description**.

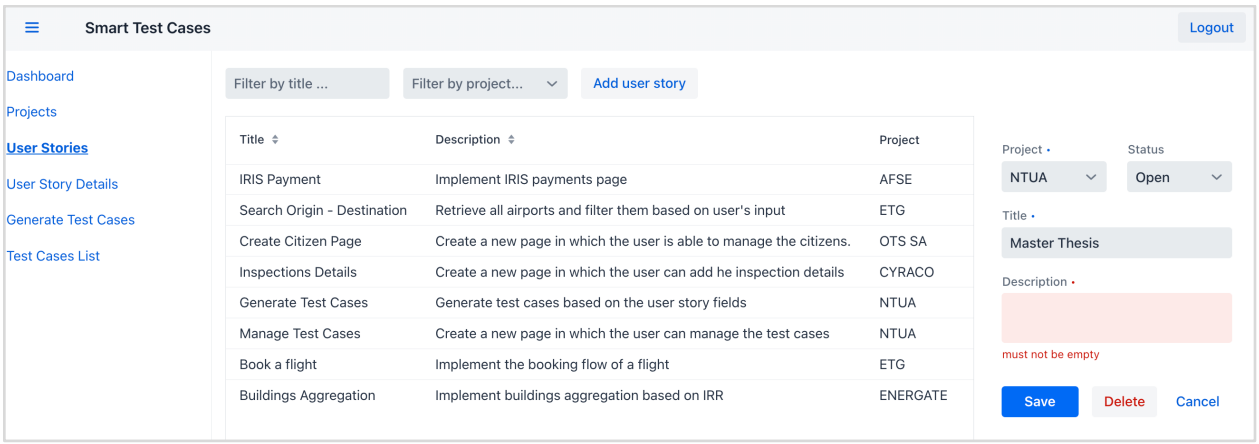


4.3.5.2. Add New User Story

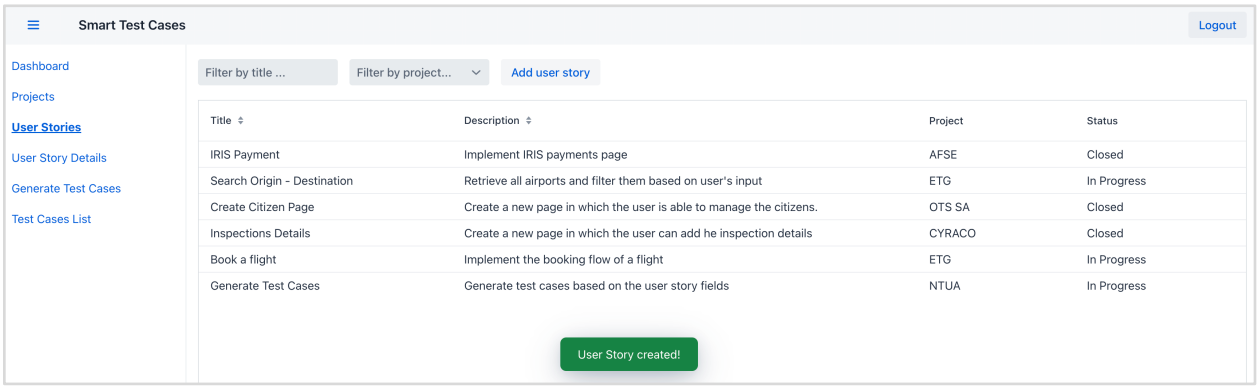
Όταν πατηθεί το κουμπί **Add user story**, στα δεξιά της οθόνης εμφανίζεται μία φόρμα, στην οποία ο χρήστης πρέπει να συμπληρώσει τα απαραίτητα πεδία (**Project**, **Status**, **Title** και **Description**) και με το κουμπί **Save** να αποθηκεύσει το νέο user story του αντίστοιχου project.



Για να δημιουργηθεί επιτυχώς το νέο user story, ο χρήστης θα πρέπει να συμπληρώσει όλα τα υποχρεωτικά πεδία της φόρμας, όπως αυτά εμφανίζονται με αστερίσκο (*). Εάν κάποιο από τα πεδία δεν είναι συμπληρωμένο, τότε το αντίστοιχο μήνυμα εμφανίζεται προς ενημέρωση του χρήστη, όπως παρακάτω.

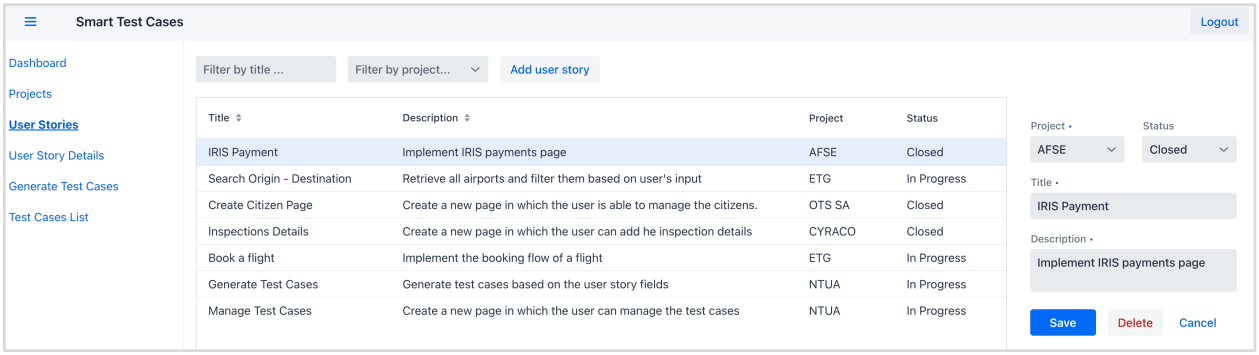


Μετά την δημιουργία του user story, ο χρήστης ενημερώνεται με το αντίστοιχο μήνυμα και η λίστα ανανεώνεται αυτόματα, ενώ η φόρμα δεν είναι πλέον ορατή στην σελίδα.

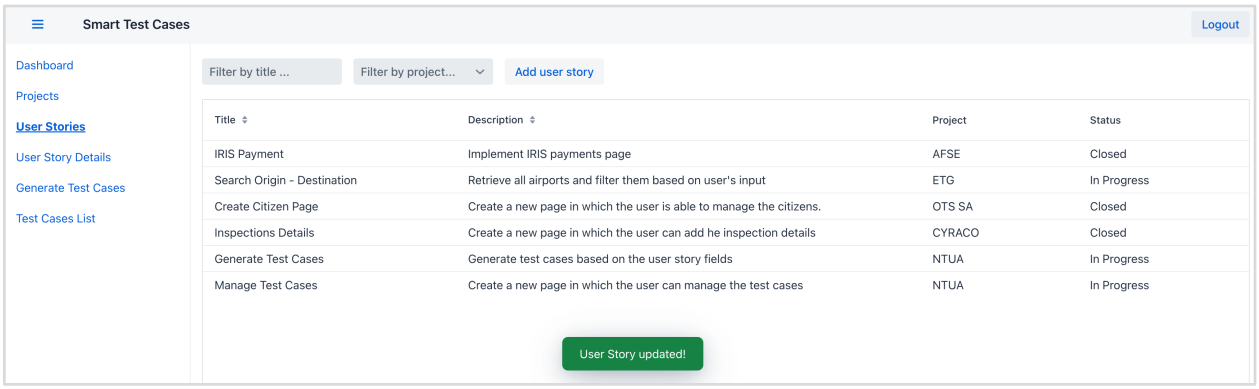


4.3.5.3. Edit & Delete User Story

Επιλέγοντας ένα user story της λίστας, τα στοιχεία του και οι επιτρεπτές ενέργειες εμφανίζονται στην φόρμα στα δεξιά της οθόνης.



Ο χρήστης μπορεί είτε επεξεργαστεί τα στοιχεία του user story, και πατώντας το κουμπί **Save** να αποθηκεύσει τις αλλαγές, είτε να διαγράψει το επιλεγμένο project. Σε κάθε περίπτωση ο χρήστης ενημερώνεται με το κατάλληλο ενημερωτικό μήνυμα. Για παράδειγμα, παρακάτω φαίνεται η περίπτωση ενημέρωσης των στοιχείων ενός user story.

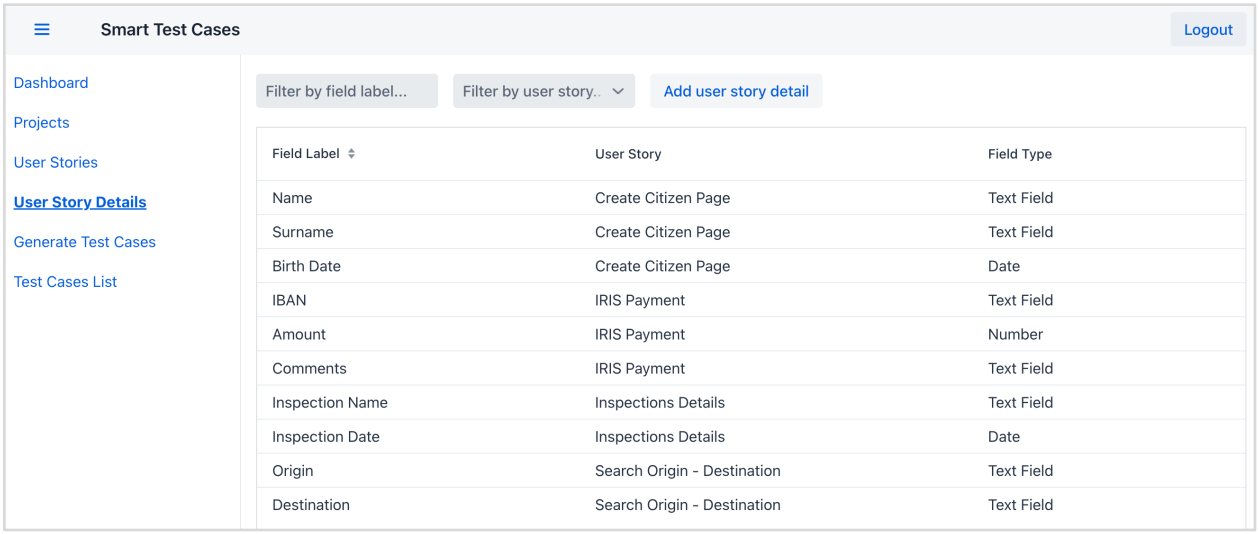


4.3.6. User Story Details

Η ενότητα **User Story Details** αναφέρεται στα πεδία που έχει κάθε user story καθώς και στα χαρακτηριστικά κάθε πεδίου (π.χ. τύπος). Επομένως, ο χρήστης έχει την δυνατότητα να διαχειριστεί τα user story details που είναι ήδη αποθηκευμένα για κάθε user story, να τα επεξεργαστεί καθώς και να δημιουργήσει νέα.

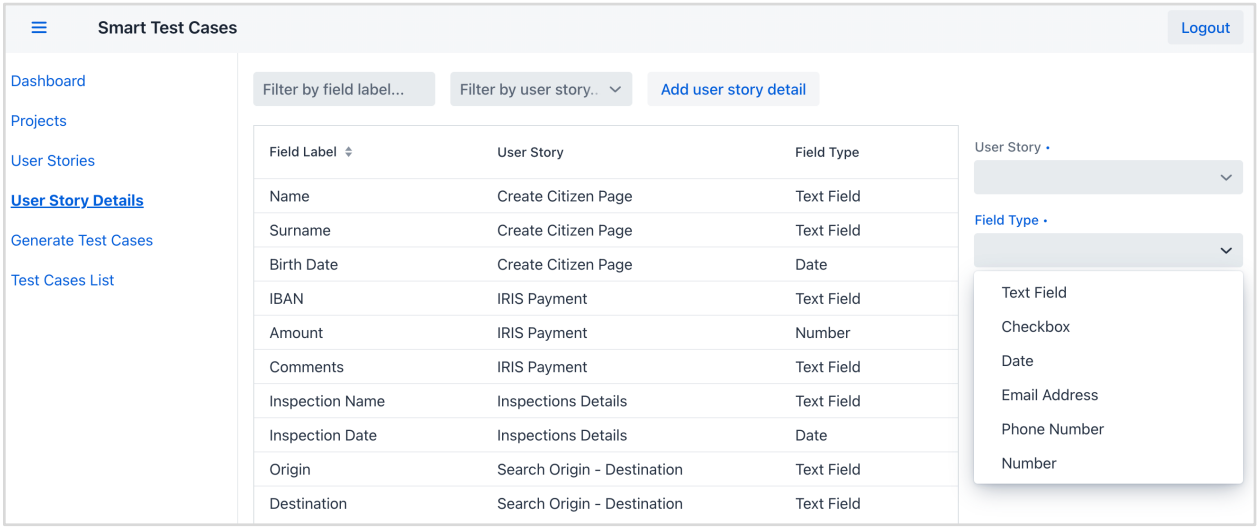
4.3.6.1. List

Πάνω από την λίστα των user stories, υπάρχει το πεδίο **Filter by field label** και η λίστα **Filter by user story**. Τα πεδία αυτά χρησιμοποιούνται ώστε να γίνεται αναζήτηση των πεδίων κάθε user story είτε χρησιμοποιώντας το **Filter by field label Filter by field** δηλαδή με το όνομα του πεδίου, είτε επιλέγοντας κάποιο user story. Τα αποτελέσματα της αναζήτησης μπορούν να ταξινομηθούν βάσει του **Field Label**.



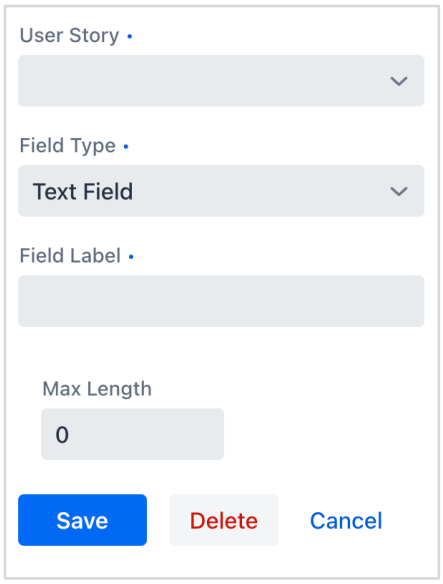
4.3.6.2. Add New User Story Detail

Όταν πατηθεί το κουμπί **Add user story detail**, στα δεξιά της οθόνης εμφανίζεται μία φόρμα, η οποία πρέπει να συμπληρωθεί κατάλληλα ώστε να δημιουργηθεί το πεδίο του user story. Ο χρήστης εκτός από το **Field Label**, δηλαδή το όνομα το πεδίου, θα πρέπει να προσδιορίσει και τον τύπο του από την αντίστοιχη λίστα **Field Type**, όπου οι διαθέσιμες επιλογές είναι *Text Field*, *Checkbox*, *Date*, *Email Address*, *Phone Number* και *Number*, όπως φαίνονται στην παρακάτω εικόνα. Ανάλογα με τον τύπο που έχει επιλεχθεί, εμφανίζονται τα επιπλέον πεδία που αφορούν τα χαρακτηριστικά του πεδίου, καθώς και τα κουμπιά **Save**, **Delete** και **Cancel**. Τα επιπλέον πεδία αφορούν τις επιτρεπτές τιμές που μπορεί να πάρει κάθε πεδίο και συνεπώς, τους ελέγχους εγκυρότητας που πρέπει να πραγματοποιηθούν ανά περίπτωση.



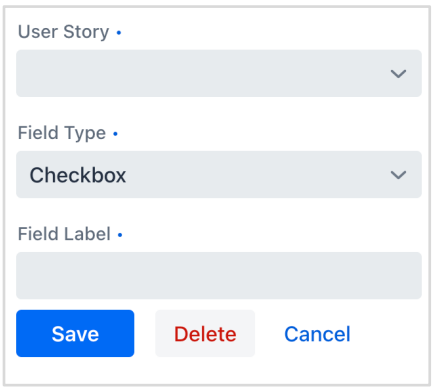
Field Type: Text Field

Όταν επιλεχθεί **Field Type:** *Text Field,* τότε ο χρήστης πρέπει να ορίσει το μέγιστο αριθμό χαρακτήρων που θα επιτρέπονται σε αυτό το πεδίο και στην συνέχεια να προχωρήσει με την αποθήκευση, πατώντας το κουμπί **Save**.



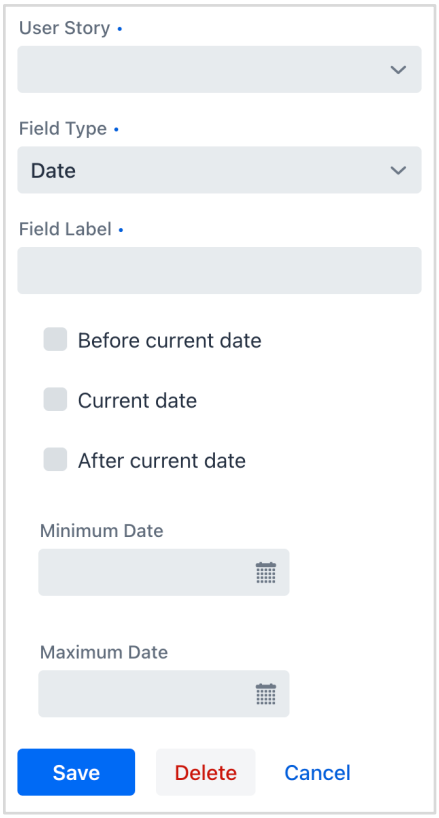
Field Type: Checkbox

Όταν επιλεχθεί **Field Type:** *Checkbox,* τότε ο χρήστης χρειάζεται να συμπληρώσει μόνο το **Field Label**.



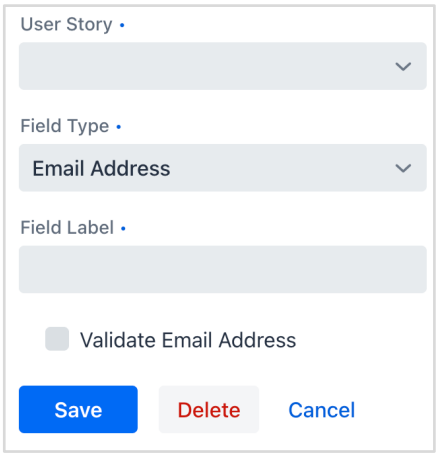
Field Type: Date

Όταν επιλεχθεί **Field Type:** *Date,* τα πεδία **Before current date**, **Current Date**, **After current date**, **Minimum Date** και **Maximum Date** είναι διαθέσιμα προς συμπλήρωση. Με το **Before current date** checkbox, ο χρήστης μπορεί να ορίσει εάν θα επιτρέπονται ημερομηνίες πριν την τρέχουσα, ενώ με το **After current date** εάν θα επιτρέπονται ημερομηνίες μετά την τρέχουσα. Ενώ εάν το **Current Date** είναι επιλεγμένο τότε ορίζεται ότι η τρέχουσα ημερομηνία θα επιτρέπεται από το σύστημα. Ο χρήστης έχει την δυνατότητα, επίσης, να ορίσει ένα συγκεκριμένο επιτρεπτό εύρος ημερομηνιών συμπληρώνοντας τα αντίστοιχα πεδία **Minimum Date** και **Maximum Date**.



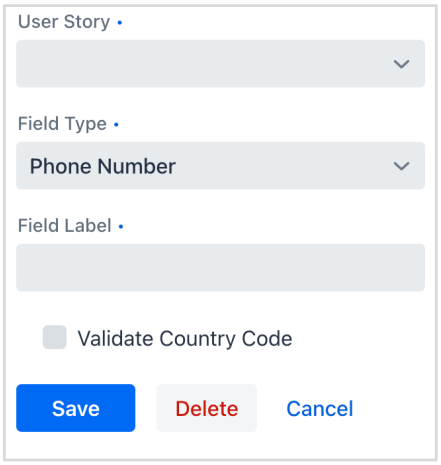
Field Type: Email Address

Όταν επιλεχθεί **Field Type:** *Email Address,* ο χρήστης πρέπει να επιλέξει το **Validate Email Address** checkbox, εάν επιθυμεί να εφαρμοστούν οι κατάλληλοι έλεγχοι, που διασφαλίζουν ότι το Email Address είναι έγκυρης μορφής, και στην συνέχεια να προχωρήσει με την αποθήκευση, πατώντας το κουμπί **Save**.



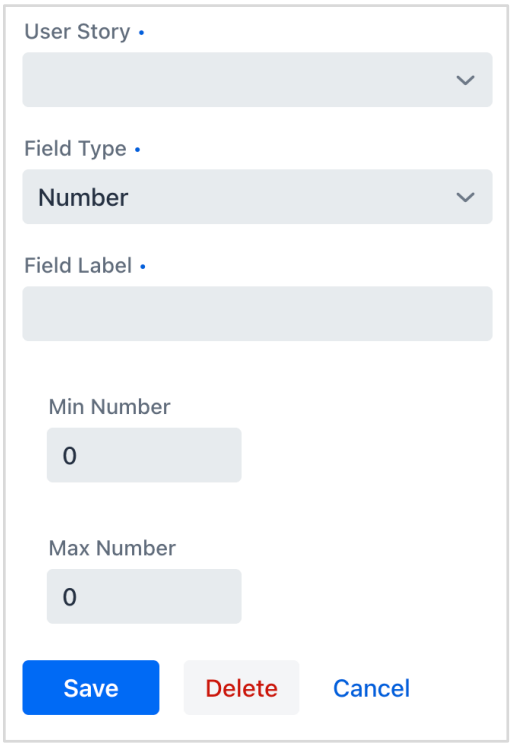
Field Type: Phone Number

Όταν επιλεχθεί **Field Type:** *Phone Number,* ο χρήστης πρέπει να επιλέξει το **Validate Phone Number** checkbox, εάν εκτός από τους ελέγχους εγκυρότητας του τηλεφωνικού αριθμού, επιθυμεί να εφαρμοστούν και οι κατάλληλοι έλεγχοι την εγκυρότητα του τηλεφωνικού κωδικού της εκάστοτε χώρας, και στην συνέχεια να προχωρήσει με την αποθήκευση, πατώντας το κουμπί **Save**.



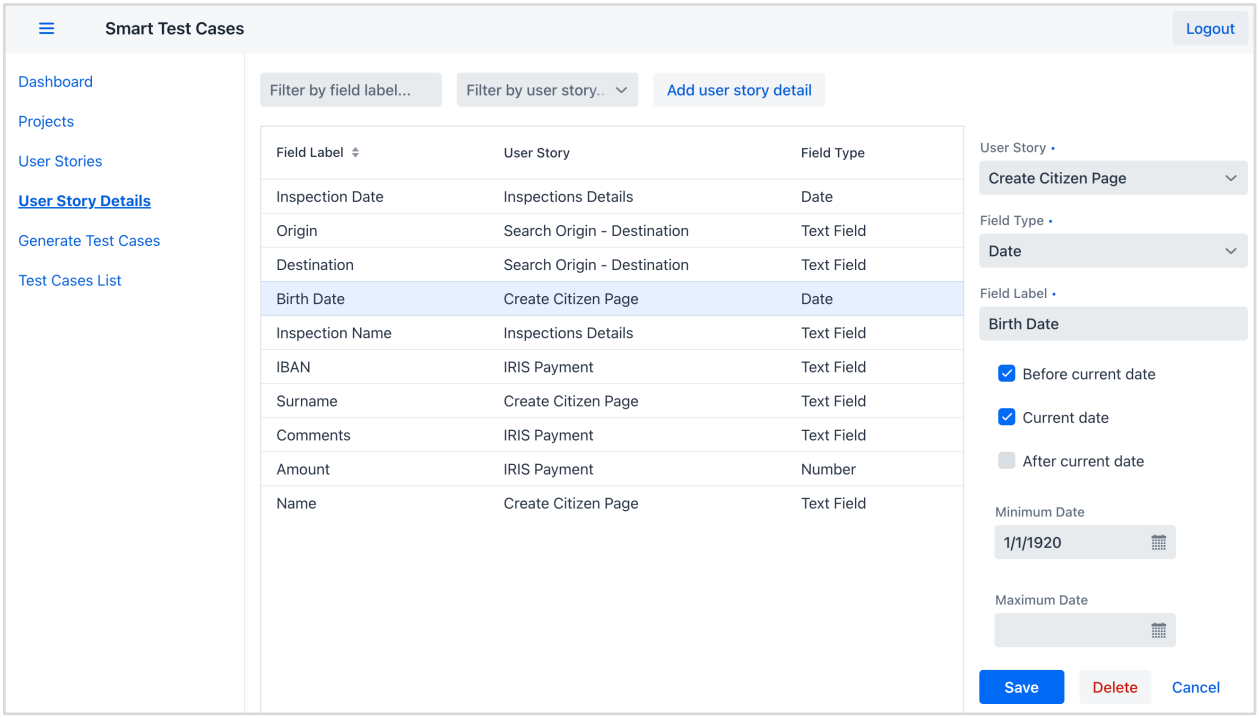
Field Type: Number

Όταν επιλεχθεί **Field Type:** *Number,* τα πεδία **Min Number** και **Max Number** είναι διαθέσιμα, στα οποία ο χρήστης μπορεί να ορίσει, εάν το επιθυμεί, το επιτρεπτό εύρος αριθμών που θα είναι αποδεκτοί στο συγκεκριμένο πεδίο και στην συνέχεια να προχωρήσει με την αποθήκευση, πατώντας το κουμπί **Save**.



4.3.6.3. Edit & Delete User Story Detail

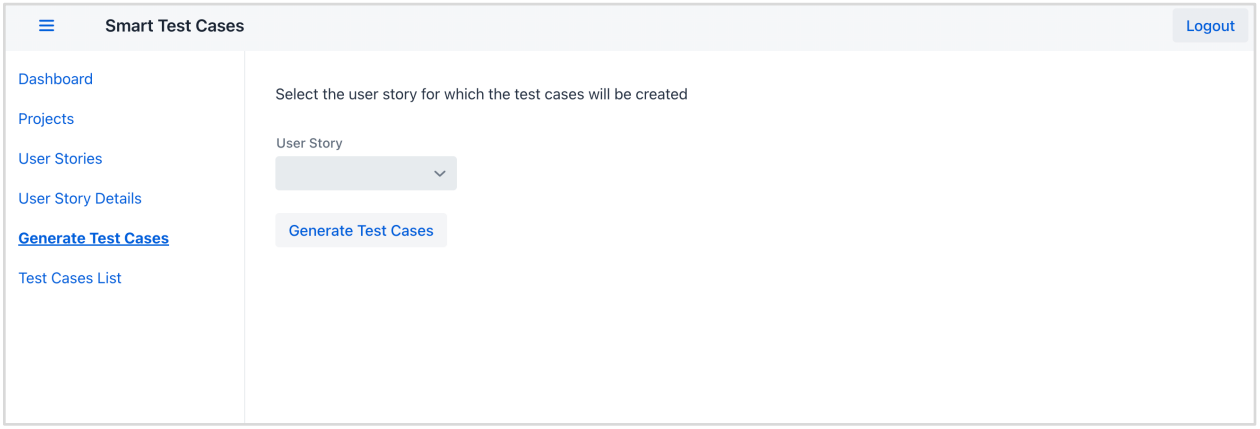
Επιλέγοντας ένα user story detail της λίστας, τα στοιχεία του και οι επιτρεπτές ενέργειες εμφανίζονται στην φόρμα στα δεξιά της οθόνης.



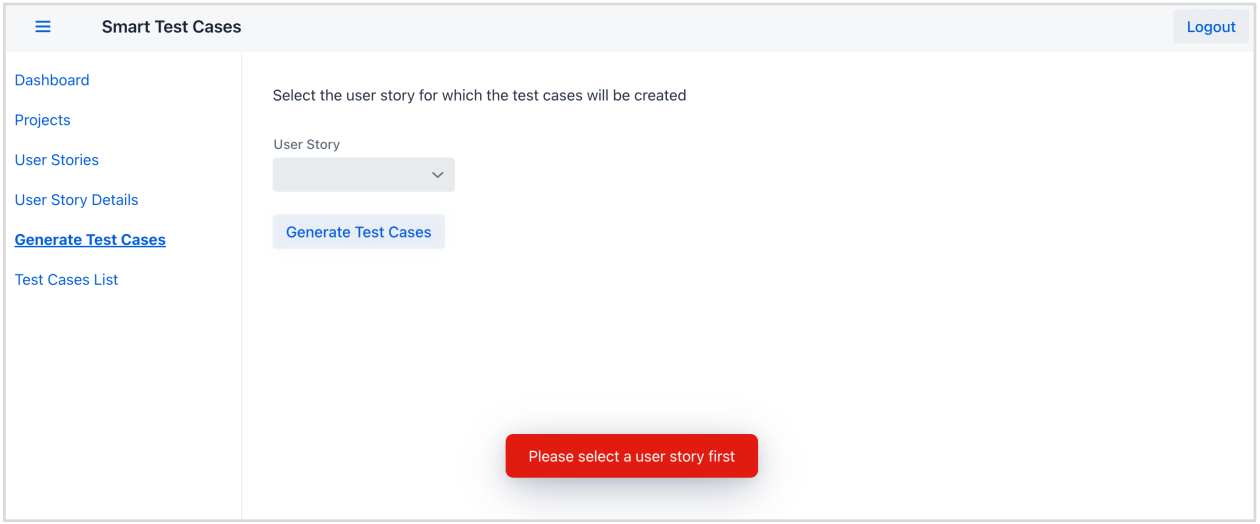
Ο χρήστης μπορεί είτε επεξεργαστεί οποιοδήποτε από τα στοιχεία του user story detail, και πατώντας το κουμπί **Save** να αποθηκεύσει τις αλλαγές, είτε να διαγράψει το επιλεγμένο project. Σε κάθε περίπτωση ο χρήστης ενημερώνεται με το κατάλληλο ενημερωτικό μήνυμα.

4.3.7. Generate Test Cases

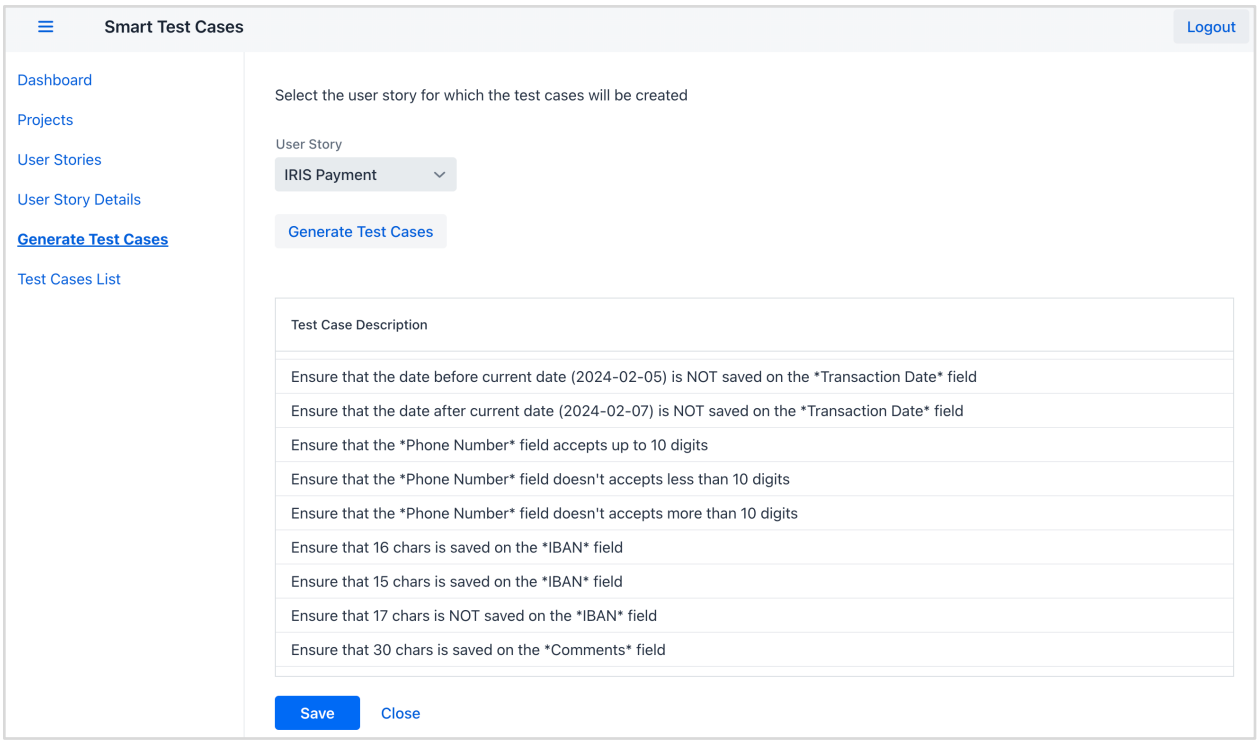
Η ενότητα **Generate Test Cases** αποτελεί την πιο βασική λειτουργία της εφαρμογής καθώς σύμφωνα με την επιλογή του επιθυμητού user story, δημιουργούνται αυτόματα τα test cases που πρέπει να εκτελεστούν ώστε να διασφαλιστεί η ποιότητα και η ορθή υλοποίηση του user story.



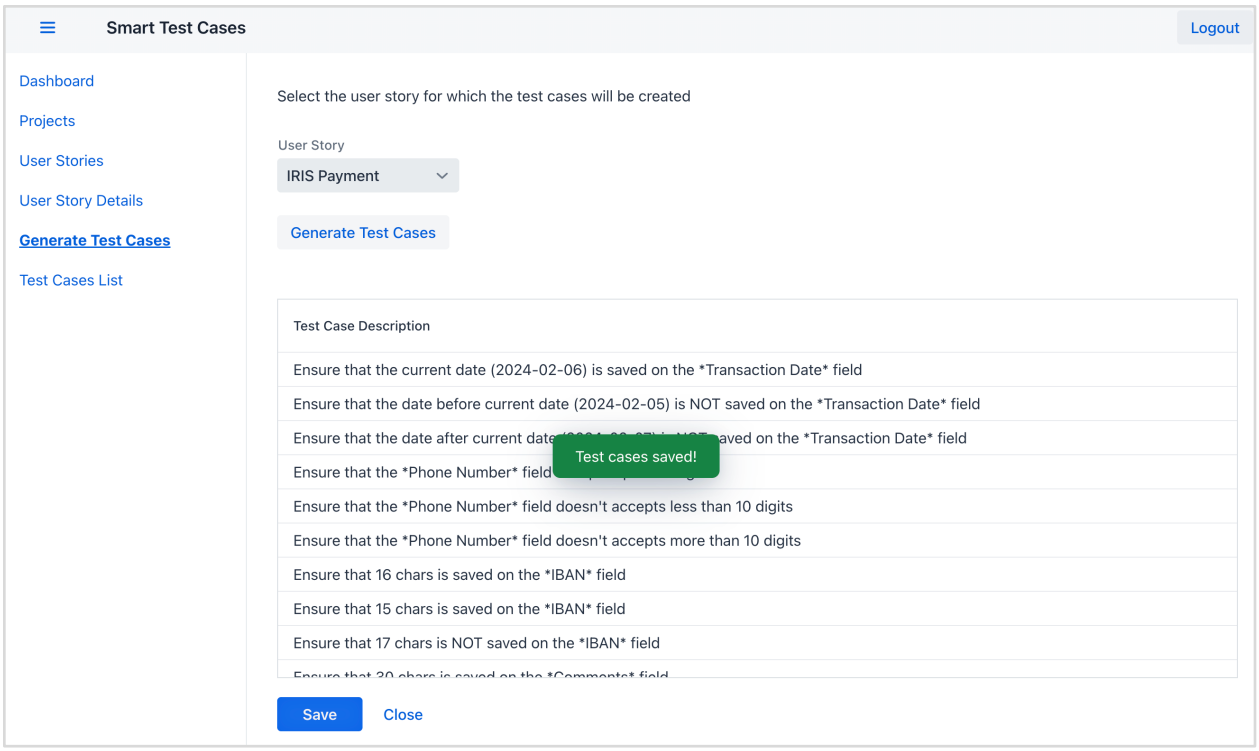
Στην περίπτωση που ο χρήστης δεν έχει επιλέξει user story και πατήσει το κουμπί **Generate Test Cases,** το ενημερωτικό μήνυμα εμφανίζεται στη οθόνη, όπως στην παρακάτω εικόνα.



Όταν ο χρήστης επιλέξει το user story από την αντίστοιχη λίστα **User Story** και στην συνέχεια πατήσει το κουμπί **Generate Test Cases**, αυτόματα δημιουργούνται και εμφανίζονται όλα τα test cases, βάσει των details (πεδίων) και των χαρακτηριστικών τους, του εκάστοτε user story, όπως παρακάτω.



Με το κουμπί **Save**, ο χρήστης μπορεί να αποθηκεύσει τα test cases και να προχωρήσει με επόμενα user stories. Στην περίπτωση που υπάρχουν ήδη αποθηκευμένα test cases για το συγκεκριμένο user story, πραγματοποιείται πρώτα η διαγραφή τους και στην συνέχεια η αποθήκευση των νέων. Αυτό εξασφαλίζει ότι δεν θα υπάρχουν πάνω από μια φορά το ίδιο test case και τα test cases θα είναι πάντα σύμφωνα με τα τρέχοντα user story details. Πιο συγκεκριμένα, εάν μετά την δημιουργία των test cases αλλάξει, προστεθεί ή αφαιρεθεί κάποιο user story detail, τότε ο χρήστης θα πρέπει να δημιουργήσει εκ νέου τα test cases.



4.3.7.1. Τεχνική Δημιουργίας Test Cases

Η εφαρμογή έχει υλοποιηθεί για να δημιουργεί αυτόματα test cases βάσει των πεδίων του αντίστοιχου user story χρησιμοποιώντας την τεχνική ελέγχου Boundary Value Analysis και πιο συγκεκριμένα την 3-value εκδοχή της, όταν τον τύπος του πεδίου είναι αριθμητικός (*Number*) και οι επιτρεπτές τιμές ανήκουν σε ένα εύρος τιμών. Όπως αναλύθηκε στο κεφάλαιο 2.7.1.2, η τεχνική αυτή εφαρμόζεται ελέγχοντας τα όρια των partitions καθώς και τις γειτονικές τιμές τους. Για παράδειγμα, όταν ο τύπος του πεδίου είναι Number, ο χρήστης καλείται να προσδιορίσει το επιτρεπτό εύρος τιμών (Min και Max Number), επομένως τα test cases θα περιέχουν ελέγχους για τις παρακάτω περιπτώσεις:

- Ο προηγούμενος αριθμός του Min Number να μην γίνεται δεκτός
- Ο αριθμός Min Number να γίνεται δεκτός
- Ο επόμενος αριθμός του Min Number να γίνεται δεκτός
- Ο προηγούμενος αριθμός του Max Number να γίνεται δεκτός
- Ο αριθμός Max Number να γίνεται δεκτός
- Ο επόμενος αριθμός του Max Number να μην γίνεται δεκτός

Αντίθετα με τα αριθμητικά πεδία, για την δημιουργία test cases των αλφαριθμητικών πεδίων (*Text Field*) χρησιμοποιείται η εκδοχή 2-value της Boundary Value Analysis, εφόσον λαμβάνεται υπόψιν μόνο ο μέγιστος επιτρεπτός αριθμός (Max Length). Βάσει της τεχνικής, για κάθε όριο του εύρους τιμών υπάρχουν δύο στοιχεία κάλυψης, η τιμή του ορίου και ο πλησιέστερος γείτονας του επόμενου partition, στην περίπτωση των αλφαριθμητικών πεδίων, οι έλεγχοι εγκυρότητας αφορούν μόνο το μέγιστο όριο του εύρους τιμών, γι' αυτό τα test cases είναι οι παρακάτω:

- Το πεδίο να δέχεται ακριβώς *Max Length* χαρακτήρες
- Το πεδίο να μην δέχεται παραπάνω από *Max Length* χαρακτήρες (δηλαδή *Max Length*+1)

Ωστόσο, στις περιπτώσεις που ο τύπος του πεδίου επιτρέπει μόνο διακριτές τιμές, η τεχνική Boundary Value Analysis δεν μπορεί να εφαρμοστεί, γι' αυτό δημιουργούνται τα test cases λαμβάνοντας υπόψιν όλες τις δυνατές τιμές για το πεδίο αυτό. Χαρακτηριστικό παράδειγμα είναι ο τύπος πεδίου Checkbox, όπου οι δυνατές τιμές είναι *True* και *False.* Σε τέτοιου είδους περιπτώσεις, δημιουργούνται τόσα test cases όσες οι δυνατές τιμές, δηλαδή ένα για κάθε δυνατή τιμή.

Όταν ο τύπος του πεδίου είναι ημερομηνία (*Date*), τότε ο χρήστης καλείται να προσδιορίσει εάν θα επιτρέπεται η τρέχουσα ημερομηνία καθώς και οι ημερομηνίες πριν και μετά την τρέχουσα. Επιπλέον, μπορεί να ορίσει και ένα εύρος ημερομηνιών (Minimum – Maximum Date) που θα είναι αποδεκτές. Επομένως, τα test cases που δημιουργούνται περιλαμβάνουν ελέγχους που καλύπτουν όλες τις δυνατές περιπτώσεις ανάλογα με τις επιλογές του χρήστη.

Τέλος για τις περιπτώσεις που ο τύπος των πεδίων είναι τηλεφωνικός αριθμός (*Phone Number*) ή διεύθυνση ηλεκτρονικού ταχυδρομείου (*Email Address*) και ο χρήστης επιθυμεί τον έλεγχο εγκυρότητας τους, τότε τα test cases καλύπτουν όλες τις περιπτώσεις για να εξασφαλίσουν ότι:

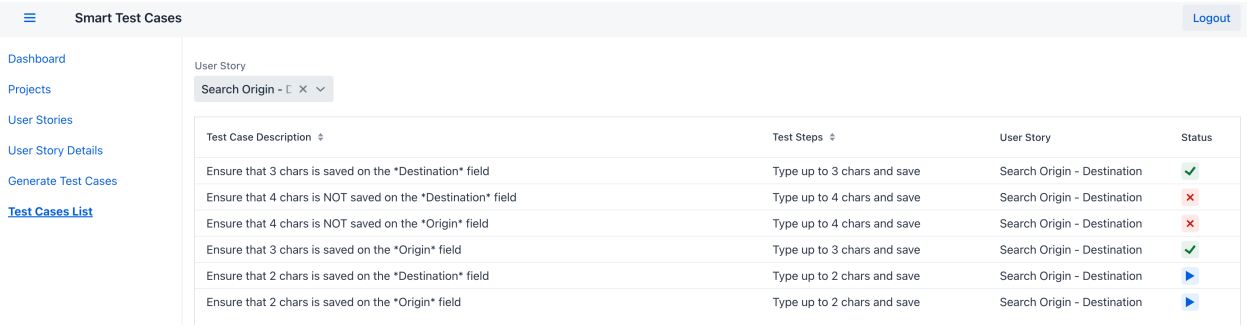
- Ο τηλεφωνικός αριθμός να περιέχει ακριβώς 10 ψηφία και ο κωδικός της χώρας είναι της μορφής (+xxxx)
- Το Email Address να είναι της μορφής test@example.com

4.3.8. Test Cases List

Η ενότητα **Test Cases List** περιέχει όλα τα test cases που έχουν δημιουργηθεί και αποθηκευτεί από τον χρήστη όλων των user stories. Ο χρήστης έχει την δυνατότητα να διαχειριστεί τα test cases όπως να κάνει διάφορες αλλαγές, να αλλάξει το status ή ακόμη και να διαγράψει κάποιο test case.

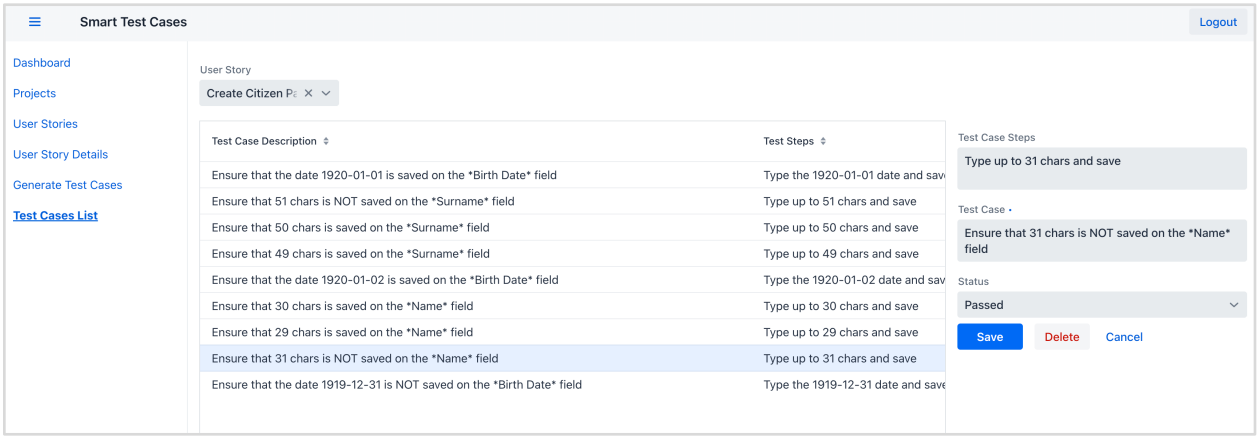
4.3.8.1. List

Πάνω από την λίστα των user stories, υπάρχει το πεδίο **Filter by user story** με την βοήθεια του οποίου γίνεται η αναζήτηση και το φιλτράρισμα των test cases ανά user story. Η λίστα εμφανίζει το **Test Case Description**, τα **Test Steps** καθώς και το **Status** των εγγραφών, όπως φαίνεται παρακάτω. Το status των test cases, που έχουν δημιουργηθεί από την εφαρμογή ορίζεται αυτόματα *Pending*.



4.3.8.2. Edit & Delete Test Case

Επιλέγοντας ένα test case της λίστας, τα στοιχεία του και οι επιτρεπτές ενέργειες εμφανίζονται στην φόρμα στα δεξιά της οθόνης. Με την λειτουργία της επεξεργασίας των test cases, ο χρήστης έχει κυρίως την δυνατότητα να προσδιορίσει την τρέχουσα κατάσταση των test cases, δηλαδή εάν έχει ολοκληρωθεί η εκτέλεση του τότε μπορεί να ορίσει εάν ήταν επιτυχημένη ή αποτυχημένη.



Έτσι, ο χρήστης έχει την δυνατότητα είτε να επεξεργαστεί τα στοιχεία του, και πατώντας το κουμπί **Save** να αποθηκεύσει τις αλλαγές, είτε να διαγράψει το επιλεγμένο test case. Σε κάθε περίπτωση ο χρήστης ενημερώνεται με το κατάλληλο ενημερωτικό μήνυμα.

4.4. Προτάσεις – Βελτιώσεις

Υποθέτοντας ότι η ανάπτυξη της εφαρμογής εξελίσσεται σε τρεις φάσεις, η πρώτη υλοποιεί την βασική λειτουργικότητα, η δεύτερη αφορά διάφορες βελτιώσεις των λειτουργιών και την δημιουργία πιο σύνθετων ελέγχων καθώς και θέματα ασφάλειας, και τέλος η τρίτη φάση θα περιέχει την υλοποίηση για integration με οποιοδήποτε άλλο issue tracking σύστημα.

Για τον σκοπό της παρούσας διπλωματικής εργασίας, αναπτύχθηκε και ολοκληρώθηκε η πρώτη φάση, κατά την οποία έχουν υλοποιηθεί οι βασικές λειτουργίες της εφαρμογής όπου ο business analyst να έχει την δυνατότητα να διαχειριστεί τα projects και τα αντίστοιχα user stories, ενώ ο tester να δημιουργεί αυτόματα τα test cases και να τα διαχειρίζεται κατάλληλα.

Στην δεύτερη φάση, η υλοποίηση θα επικεντρωθεί στον εμπλουτισμό των βασικών λειτουργικών καθώς και στην ασφάλεια της εφαρμογής. Μια σημαντική λειτουργία της εφαρμογής είναι να υποστηρίζεται η διαχείριση των features που πρέπει να υλοποιηθούν για κάθε project. Ουσιαστικά, κάθε feature αποτελεί ένα Business Requirement για το εκάστοτε project, το οποίο με την σειρά του αποτελείται από τα αντίστοιχα user stories. Με αυτόν τον τρόπο, τα user stories θα μπορούν να κατηγοριοποιηθούν ανά Business Requirement και έτσι διευκολύνεται η ευρύτερη διαχείριση τόσο των user stories όσο και των projects. Δεδομένου ότι ένα project έχει πάρα πολλά user stories, οι χρήστες θα έχουν την δυνατότητα να δουν την συνολική πρόοδο του project σε επίπεδο Business Requirement και να επικεντρωθούν μόνο στα user stories που τους ενδιαφέρουν.

Επιπλέον στην δεύτερη φάση, κατά την δημιουργία των test cases, ο συνδυασμός των ελέγχων μπορεί να εμπλουτιστεί λαμβάνοντας υπόψιν τους επιχειρησιακούς ελέγχους και όχι μόνο τους λογικούς. Με την κατάλληλη υλοποίηση, οι χρήστες θα έχουν την δυνατότητα να προσθέτουν τις σχέσεις μεταξύ των πεδίων και διάφορους άλλους περιορισμούς, όπως επιτρεπτές τιμές ενός πεδίου να εξαρτώνται από την τιμή ενός άλλου πεδίου. Για παράδειγμα, ο χρήστης να μπορεί να προσδιορίσει εάν στο πεδίο **Φύλο** επιλεχθεί η τιμή *Άνδρας* τότε θα εμφανιστεί το checkbox **Εκπληρωμένες Στρατιωτικές Υποχρεώσεις** και στην συνέχεια να δημιουργούνται τα απαραίτητα test cases. Ένα άλλο παράδειγμα είναι οι τιμές ή η συμπεριφορά ενός πεδίου να εξαρτάται από την επιλογή μιας τιμής που έχει γίνει σε προηγούμενο πεδίο.

Τέλος στην δεύτερη φάση, θα μελετηθούν και αναπτυχθούν διάφορα θέματα ασφαλείας, καθώς επίσης και οι ρόλοι των χρηστών. Αρχικά, θα υλοποιηθεί η λειτουργία διαχείρισης χρηστών, κατά την οποία, ο admin χρήστης θα έχει την δυνατότητα να δημιουργεί νέους χρήστες, να προσδιορίσει τους ρόλους και τα δικαιώματά τους καθώς επίσης να επεξεργαστεί ή να διαγράψει έναν υπάρχον χρήστη. Ανάλογα με τον ρόλο και τα δικαιώματα κάθε χρήστη, θα προσδιορίζονται και οι επιτρεπτές ενέργειες

του, όπως φαίνονται στον παρακάτω πίνακα. Επίσης, στα πλαίσια αυτής της υλοποίησης, θα συμπεριληφθούν έλεγχοι ώστε ο κάθε χρήστης να μπορεί να έχει πρόσβαση και να επεξεργαστεί μόνο τα δικά του projects, business requirements κλπ.

	Admin	*Product Owner Business Analyst*	*Developer*	*Tester*
Manage Users	✓	✗	✗	✗
Manage Projects	✓	✓	✗	✗
Manage Business Requirements	✓	✓	Read only	Read only
Manage User Stories	✓	✓	Read only	✓
Manage Test Cases	✓	Read only	Read only	✓

Στην τρίτη και τελευταία φάση, θα γίνουν οι απαραίτητες ενέργειες ώστε η εφαρμογή να μπορεί να διασυνδεθεί με οποιοδήποτε άλλο issue tracking tool, όπως το Jira[7]. Οι σύγχρονες ομάδες ανάπτυξης λογισμικού συνήθως χρησιμοποιούν κάποιο issue tracking tool από τα διαθέσιμα της αγοράς, για την καταγραφή των user stories και των bugs, τα οποία είναι ταυτόχρονα διασυνδεδεμένα με κάποιο version control system (VCS) όπως BitBucket[8]. Επομένως, η διασύνδεση της εφαρμογής Smart Test Cases με κάποιο issue tracking tool, θα δώσει την δυνατότητα στην ομάδα να χρησιμοποιεί μόνο ένα κοινό σύστημα στο οποίο θα υπάρχουν όλες οι πληροφορίες διασυνδεδεμένες που αφορούν το project και τα user stories, το versioning του κώδικα, τα test cases όπως δημιουργήθηκαν αυτόματα, τα bugs που βρέθηκαν και σε ποιο test case σχετίζονται καθώς επίσης και την όλη πρόοδο τόσο της ανάπτυξης όσο και του ελέγχου του λογισμικού.

Συμπερασματικά, η εφαρμογή Smart Test Cases, μετά την πλήρη υλοποίηση της, μπορεί να αποτελέσει ένα εργαλείο καθημερινής χρήσης για την Agile ομάδα ανάπτυξης. Μέσα από το εύχρηστο περιβάλλον, οι χρήστες θα έχουν την πλήρη εικόνα προόδου του project, θα διαχειρίζονται τα user stories που πρέπει να υλοποιηθούν, θα δημιουργούν εύκολα και γρήγορα τα test cases καθώς και θα διαχειρίζονται ολόκληρη την διαδικασία ελέγχου.

[7] https://www.atlassian.com/software/jira
[8] https://bitbucket.org/product

References

[1] M. Rouse, "Software," March 2020. [Online]. Available: https://www.techopedia.com/definition/4356/software.

[2] International Software Testing Qualifications Board, "Foundation Level Syllabus," April 2023. [Online]. Available: https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB_CTFL_Syllabus-v4.0.pdf.

[3] T. Devi, "Importance of Testing in Software Development," *International Journal of Scientific & Engineering Research,* 2012.

[4] G. Alleman, "Agile Project Management Methods for IT Projects," in *The Story of Manageming Projects: A Global, Cross-Disciplinary Collection of Perspectives*, Berkeley, Greenwood Press, 2002.

[5] 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, "Manifesto for Agile Software Development," 2001. [Online]. Available: https://ai-learn.it/wp-content/uploads/2019/03/03_ManifestoofAgileSoftwareDevelopment-1.pdf.

[6] N. B. Ruparelia, "Software Development Lifecycle," May 2010. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1764810.1764814.

[7] I. Sommerville, Software Engineering, Pearson, 2010.

[8] D. A. Garvin, "Sloan Management Review: What Does “Product Quality” Really Mean?," 1984. [Online]. Available: https://sloanreview.mit.edu/article/what-does-product-quality-really-mean/.

[9] R. S. C. Itti Hooda, "Software Test Process, Testing Types and Techniques".*International Journal of Computer Applications* .

[10] M. A. Umar, "A Study of Software Testing: Categories, Levels, Techniques, and," School of Computer Science and Technology, Changchun University of Science and Technology, China.

[11] H. Samra, "Study on Non Functional Software Testing," International Journal of Computers & Technology, 2013.

[12] M. G. M. Karuturi Sneha, "Research on Software Testing Techniques and Software Automation Testing Tools," *International Conference on Energy, Communication, Data Analytics and Soft Computing,* 2017.

[13] P. M. Jacob and D. M. Prasanna, "A Comparative analysis on Black Box Testing Strategies," IEEE, 2016.

[14] D. Swain, "Decision Table Testing A Complete Overview," 2023. [Online]. Available: https://testsigma.com/blog/decision-table-testing/.

[15] M. E. Khan, "Different Approaches To Black Box Testing Technique For Finding Errors," *International Journal of Software Engineering & Applications (IJSEA),* vol. 2, 2011.

[16] S. Nidhra and J. Dondeti, "Black Box And White Box Testing Techniques - A Literature Review," *International Journal of Embedded Systems and Applications (IJESA) ,* vol. 2, 2012.

[17] "Experience Based Testing," 2019. [Online]. Available: https://www.professionalqa.com/experience-based-testing.

[18] N. Nader-Rezvani, An Executive's Guide to Software Quality in an Agile Organization.

[19] T. Khomenko, "Bug vs. Defect: Difference With Definition Examples Within Software Testing," 2023. [Online]. Available: https://testomat.io/blog/bug-vs-defect-difference-with-definition-examples-within-software-testing/.

[20] P. Yadav, "The Differences Between Priority and Severity in Testing," 2023. [Online]. Available: https://testsigma.com/blog/difference-between-priority-and-severity/.

[21] C. Edeki, "AGILE SOFTWARE DEVELOPMENT METHODOLOGY," *European Journal of Mathematics and Computer Science,* 2015.

[22] D. A. P. Zainala, R. Razalia and Z. Mansora, "Team Formation for Agile Software Development: A Review," *International Journal on Advanced Science, Engineering and Information Technology,* 2020.

[23] N. Tsonev, "What Is an Agile Project Manager? Role, Responsibilities and Skills," 2022. [Online]. Available: https://businessmap.io/blog/agile-project-manager.

[24] "What is a Product Owner?," [Online]. Available: https://www.scrum.org/resources/what-is-a-product-owner.

[25] Paula, "Where does a Business Analyst fit in a Scrum Team?," [Online]. Available: https://premieragile.com/where-does-a-business-analyst-fit-in-a-scrum-team/#:~:text=A%20Scrum%20Business%20Analyst%20is,members%20of%20the%20Scrum%20Team..

[26] K. Swed, "What Is a Software Engineer?," 2023. [Online]. Available: https://www.computerscience.org/careers/software-engineer/.

[27] M. Rouse, "What Does Software Tester Mean?," 2014. [Online]. Available: https://www.techopedia.com/definition/29845/software-tester.

[28] P. Zandbergen, "What is a Database Management System? - Purpose and Function," [Online]. Available: https://study.com/academy/lesson/what-is-a-database-management-system-purpose-and-function.html.

[29] N. Duggal, "DevOps Engineer Job Description: Skills, Roles and Responsibilities," [Online]. Available: https://www.simplilearn.com/devops-engineer-job-description-article.

[30] Wrike, "The Agile Software Development Life Cycle," [Online]. Available: https://www.wrike.com/agile-guide/agile-development-life-cycle/.

[31] K. Brush and V. Silverthorne, "Agile software development," 2019. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/agile-software-development.

[32] S. Sharma, D. Kumar and M. Fayad, "An Impact Assessment of Agile Ceremonies on Sprint Velocity Under Agile Software Development," in *9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO) Amity University*, Noida, India, 2021.

[33] "Sprint Planning," [Online]. Available: https://www.agilealliance.org/glossary/sprint-planning/.

[34] J. Yip, "Daily Meeting," [Online]. Available: https://www.agilealliance.org/glossary/daily-meeting/.

[35] M. Venema, "Backlog Refinement/ Grooming : What It Is and Why & How You Do It?," 2023. [Online]. Available: https://www.nimblework.com/agile/backlog-refinement/.

[36] "Iteration Review," 2022. [Online]. Available: https://scaledagileframework.com/iteration-review/.

[37] D. RADIGAN, "An agile guide to scrum meetings," [Online]. Available: https://www.atlassian.com/agile/scrum/ceremonies#:~:text=Sprint%20retrospective&text=master%2C%20product%20owner-,When%3A%20At%20the%20end%20of%20a%20sprint.,based%20on%20a%20fixed%20cadence..

[38] M. Cohn, User Stories Applied: For Agile Software Development, The Addison-Wesley Signature Series, 2004.

[39] M. Rehkopf, "User stories with examples and a template," Atlassian, [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories.

[40] S. Bose, "How to write Test Cases in Software Testing?," BrowserStack, 2023. [Online]. Available: https://www.browserstack.com/guide/how-to-write-test-cases.