ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Επαλήθευση, Επικύρωση και Συντήρηση Λογισμικού

ΘΕΜΑ ΕΡΓΑΣΙΑΣ:

Functional Testing

ΔΙΔΑΣΚΩΝ ΚΑΘΗΓΗΤΗΣ:
ΜΑΛΕΥΡΗΣ ΝΙΚΟΛΑΟΣ

ΟΜΑΔΑ:
ΝΤΟΓΚΑ ΜΑΡΙΝΑ (3160119)
ΜΠΟΥΚΑΣ ΔΗΜΗΤΡΙΟΣ (3150120)

ΑΘΗΝΑ
ΜΑΙΟΣ 2019

ΠΕΡΙΕΧΌΜΕΝΑ

1. **Software Testing**
   - 1.1. Τι είναι
   - 1.2. Σχέση μεταξύ software testing και προγραμματιστών
   - 1.3. Κατηγορίες
2. **Functional Testing**
   - 2.1. Τι είναι
   - 2.2. Τι ελέγχει
   - 2.3. Πως υλοποιείται
   - 2.4. Κατηγορίες
3. **Unit Testing**
   - 3.1. Tasks
   - 3.2. Ποιος το εκτελεί
   - 3.3. Tools
4. **Smoke Testing**
   - 4.1. Πως υλοποιείται
5. **Regression Testing**
   - 5.1. Παράδειγμα
   - 5.2. Πότε εκτελείται
   - 5.3. Τύποι
   - 5.4. Μπορεί να εκτελεστεί manually;
   - 5.5. Tools
6. **Sanity Testing**
   - 6.1. Παράδειγμα
   - 6.2. Χαρακτηριστικά
   - 6.3. Πλεονεκτήματα
7. **Integration Testing**
   - 7.1. Κάποιοι χρήσιμοι ορισμοί αναφορικά
   - 7.2. Αναλογίες
   - 7.3. Μέθοδος
   - 7.4. Tasks
   - 7.5. Πότε εκτελείται
   - 7.6. Ποιος το πραγματοποιεί
   - 7.7. Προσεγγίσεις
8. **Black Box Testing**
   - 8.1. Ορισμοί
   - 8.2. Παράδειγμα
   - 8.3. Που εκτελείται
   - 8.4. Τεχνικές
   - 8.5. Πλεονεκτήματα
   - 8.6. Tools
9. **White Box Testing**
   - 9.1. Ορισμός
   - 9.2. Παράδειγμα
   - 9.3. Που εφαρμόζεται
10. **Διαφορές Black Box - White Box Testing**

10.1. Πλεονεκτήματα
11. User Acceptance Testing
11.1. Γιατί γίνεται
11.2. Ποιος εκτελεί
11.3. Αναγκαιότητα δοκιμής
11.4. V-Model
11.5. Προϋποθέσεις
11.6. Πως γίνεται
11.7. Κριτήρια Ετοιμότητας
11.8. Βέλτιστες Τεχνικές
11.9. Tools
12. Non-Functional Testing
12.1. Τι είναι
12.2. Παράδειγμα
12.3. Χαρακτηριστικά
12.4. Παράμετροι
13. Functional vs Non-Functional Testing
14. Βιβλιογραφία

1. Software Testing

1.1. Τι είναι

Το Software testing είναι μία έρευνα που έχει ως σκοπό να παρέχει πληροφορίες σχετικά με την ποιότητα του υπό δοκιμή προϊόντος ή υπηρεσίας στους ενδιαφερόμενους. Οπότε το testing παρέχει μια αντικειμενική και ανεξάρτητή άποψη του λογισμικού ώστε να μπορεί ο ενδιαφερόμενος (η επιχείρηση) να εκτιμήσει και να κατανοήσει τους κινδύνους από την εφαρμογή του λογισμικού. Οι τεχνικές test περιλαμβάνουν τη διαδικασία εκτέλεσης ενός προγράμματος ή μιας εφαρμογής με στόχο την εύρεση σφαλμάτων λογισμικού (ή γενικά ελαττωμάτων) και την επαλήθευση ότι το προϊόν λογισμικού είναι κατάλληλο για χρήση.

Το Software testing περιλαμβάνει την εκτέλεση ενός στοιχείου λογισμικού ή ενός στοιχείου συστήματος για την αξιολόγηση μιας ή περισσοτέρων ενδιαφερόντων ιδιοτήτων. Γενικά, αυτές οι ιδιότητες υποδεικνύουν την έκταση στην οποία το στοιχείο λογισμικού ή συστήματος υπό δοκιμή: a) πληροί τις απαιτήσεις που οδήγησαν τον σχεδιασμό και την ανάπτυξή του, b) ανταποκρίνεται σωστά σε όλα τα είδη εισόδων, c) εκτελεί τις λειτουργίες του εντός αποδεκτού χρόνου, d) είναι επαρκώς χρησιμοποιήσιμο, e) μπορεί να εγκατασταθεί και να εκτελεστεί στα προβλεπόμενα περιβάλλοντα του και f) επιτυγχάνει το γενικό αποτέλεσμα που επιθυμούν οι ενδιαφερόμενοι.

Όλα τα Software testing χρησιμοποιούν κάποια στρατηγική για να επιλέξουν tests που είναι εφικτά για το διαθέσιμο χρόνο και πόρο, δεδομένου ότι ο αριθμός των πιθανών tests (για ακόμα και απλά στοιχεία λογισμικού) είναι πρακτικά άπειρος. Η εργασία των tests είναι μία επαναληπτική διαδικασία επειδή όταν ένα σφάλμα διορθώνεται, μπορεί να εμφανίσει άλλα σφάλματα, βαθύτερα, ή ακόμα και να δημιουργήσει νέα. Δυστυχώς δε γίνεται ποτέ να βρεθούν όλα τα σφάλματα ή ελαττώματα σε ένα κομμάτι λογισμικού και ποτέ δε γίνεται να δοκιμαστεί κάθε δυνατή εισαγωγή στο λογισμικό.

Το Software testing μπορεί να εκτελεστεί μόλις υπάρξει εκτελέσιμο λογισμικού (έστω και μερικώς πλήρες). Η γενική προσέγγιση στην ανάπτυξη λογισμικού συχνά καθορίζει πότε και πώς διεξάγονται τα testing. Για παράδειγμα, σε μια σταδιακή διαδικασία, τα περισσότερα testing συμβαίνουν αφού καθοριστούν οι απαιτήσεις του συστήματος και στη συνέχεια εφαρμοστούν σε δοκιμαστικά προγράμματα.

Η κύρια ιδέα πίσω από το Software testing είναι να μειωθεί ο κίνδυνος ότι ο πελάτης επηρεάζεται πολύ αρνητικά κατά τη χρήση του λογισμικού. Τυπικά αυτό επιτυγχάνεται με την πρώτη προτεραιότητα σε ποιες περιοχές

του λογισμικού είναι πιθανό να έχει το μεγαλύτερο αντίκτυπο και στη συνέχεια να αποφασίσει ένα σύνολο tests για να τρέξει το οποίο επαληθεύει την επιθυμητή λειτουργικότητα σε αυτή τη περιοχή. Οπότε παρέχει μια κριτική ή μια σύγκριση που συγκρίνει την κατάσταση και τη συμπεριφορά του προϊόντος έναντι test oracles (αρχές ή μηχανισμούς με τους οποίους κάποιος μπορεί να αναγνωρίσει ένα πρόβλημα). Αυτά τα oracles μπορούν να περιλαμβάνουν (αλλά δεν περιορίζονται σε) προδιαγραφές, συμβάσεις , συγκρίσιμα προϊόντα, προηγούμενες εκδόσεις του ίδιου προϊόντος, συμπεράσματα σχετικά με τον επιδιωκόμενο ή αναμενόμενο σκοπό, τις προσδοκίες των χρηστών ή των πελατών, τα σχετικά πρότυπα, τους εφαρμοστέους νόμους ή άλλα κριτήρια.

Ο σκοπός του testing δεν είναι μόνο η ανίχνευση αποτυχιών του λογισμικού, έτσι ώστε τα ελαττώματα να μπορούν να ανακαλυφθούν και να διορθωθούν. Το testing δεν μπορεί να αποδείξει ότι ένα προϊόν λειτουργεί σωστά υπό όλες τις συνθήκες, αλλά μόνο ότι δεν λειτουργεί σωστά υπό συγκεκριμένες συνθήκες. Το πεδίο εφαρμογής των δοκιμών λογισμικού περιλαμβάνει συχνά την εξέταση του κώδικα καθώς και την εκτέλεση αυτού του κώδικα σε διάφορα περιβάλλοντα και συνθήκες καθώς και την εξέταση των πτυχών του κώδικα: κάνει αυτό που πρέπει να κάνει και κάνει ό, τι χρειάζεται να κάνει.

1.2. Σχέση μεταξύ Software Testing και προγραμματιστών.

Συνήθως οι νέοι προγραμματιστές δε καταλαβαίνουν την αξία του testing, ή δε το θεωρούν απαραίτητο. Ο κύριος λόγος είναι γιατί νομίζουν ότι ο στόχος είναι μόνο να βρει σφάλματα και να κάνει το λογισμικό καλύτερο. Στη πραγματικότητα όμως είναι να μειωθεί ο κίνδυνος προωθώντας την εξερεύνηση και την εξάλειψη των προβλημάτων που θα έχουν τον μεγαλύτερο αντίκτυπο στον πελάτη χρησιμοποιώντας το λογισμικό.

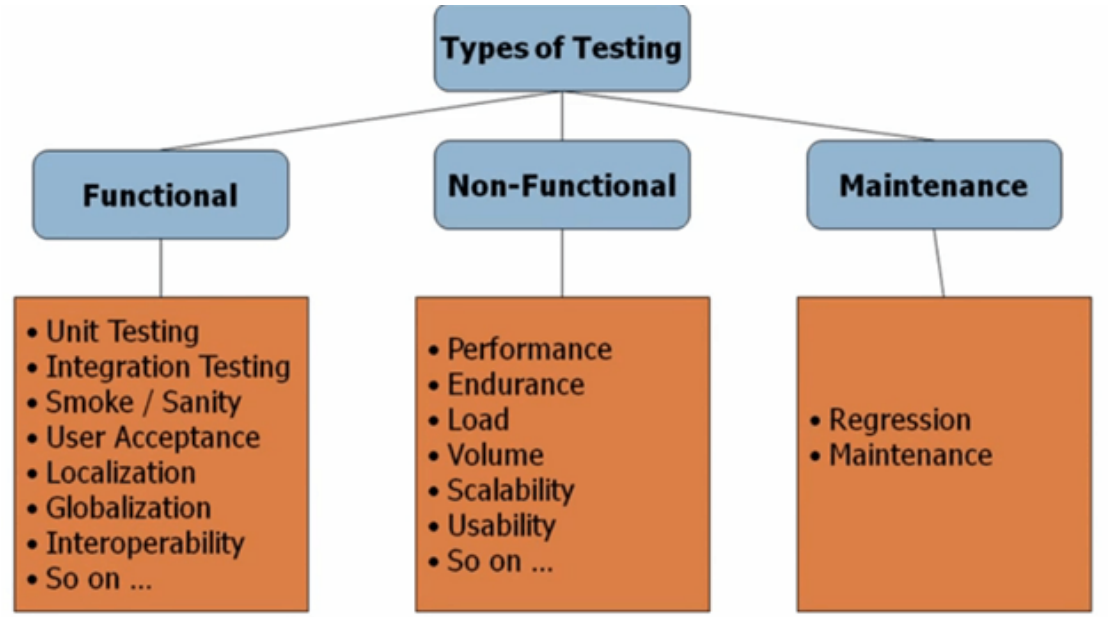
Οι προγραμματιστές τείνουν να μην αναγνωρίζουν εύκολα την αξία της διασφάλισης ποιότητας του λογισμικού. Πολλοί προγραμματιστές βλέπουν το Software testing ως κάτι που επιβαρύνει τη διαδικασία παραγωγής. Χωρίς το testing οι πελάτες θα έβλεπαν αναπόφευκτα ότι οι προσπάθειές τους για ανάπτυξη λογισμικού τελειώνουν σε αποτυχία καθώς εκδίδουν πολύ ελαττωματικά προγράμματα. Η καθυστέρηση στην επιστροφή δοκιμασμένων δομικών στοιχείων στους προγραμματιστές είναι μια σημαντική πηγή τριβής μεταξύ προγραμματιστών και testers. Τυχόν επιβράδυνση στον κύκλο παραγωγής αποδίδεται στο τμήμα των testers και πιθανότατα θα θεωρηθεί από τους προγραμματιστές ως περαιτέρω απόδειξη ότι αυτή η μονάδα εμποδίζει τις αυστηρές ημερομηνίες κυκλοφορίας.

Είναι σημαντικό οι testers να κάνουν ό, τι μπορούν για να επιταχύνουν τη διαδικασία του testing χωρίς να θυσιάζουν την επιμέλεια και την προσοχή στη λεπτομέρεια. Ένας τρόπος για να επιταχυνθούν οι δοκιμές είναι η αυτοματοποίηση εργαλείων. Τα αυτοματοποιημένα δοκιμαστικά σενάρια μπορούν να αναλύσουν πολύ περισσότερο κώδικα από έναν χειριστή που θα μπορούσε ποτέ, με αποτέλεσμα μικρότερους χρόνους επιστροφής στα αποτελέσματα των δοκιμών ή ένας άλλος τρόπος για να βελτιωθεί τη διαδικασία του testing είναι να χρησιμοποιηθούν ενημερώσεις σε πραγματικό χρόνο με όλες τις σχετικές αναφορές.

1.3. Κατηγορίες

Γενικά, στο Software Testing υπάρχουν 3 κατηγορίες:

- *Functional*
- *Non - Functional*
- *Maintenance*



Εμείς για τους σκοπούς της εργασίας μελετάμε το Functional Testing.

Παρόλα αυτά υπάρχουν συνοπτικά περιγραφές του Non-functional Testing, ώστε να μπορούμε να εντοπίσουμε τις διαφορές με το Functional Testing.

2. Functional Testing

2.1. Τι είναι

Το functional testing είναι ένας τρόπος ελέγχου του λογισμικού, που επικυρώνει ένα σύστημα λογισμικού βάσει των λειτουργικών απαιτήσεων/προσδιορισμών.

Σκοπός του Functional Testing είναι ο έλεγχος κάθε λειτουργίας μιας εφαρμογής, παρέχοντας κατάλληλες εισόδους και επαληθεύοντας την απόδοση του λογισμικού σύμφωνα με τις λειτουργικές απαιτήσεις.

Το Functional Testing εμπλέκεται με black box testing και όχι με τον κώδικα της εφαρμογής. Τι σημαίνει πρακτικά αυτό; Αυτό σημαίνει ότι αυτός ο τύπος testing ασχολείται κυρίως με τη διεπαφή του χρήστη (User Interface), APIs, βάσεις δεδομένων (database), ασφάλεια (security), επικοινωνία που σχετίζεται με sockets και ports (Client- Server communication) και άλλες λειτουργικότητες της εφαρμογής που βρίσκονται υπό έλεγχο.

Σε αυτή την περίπτωση το testing μπορεί να γίνει με 2 τρόπους: είτε manually είτε χρησιμοποιώντας automation.

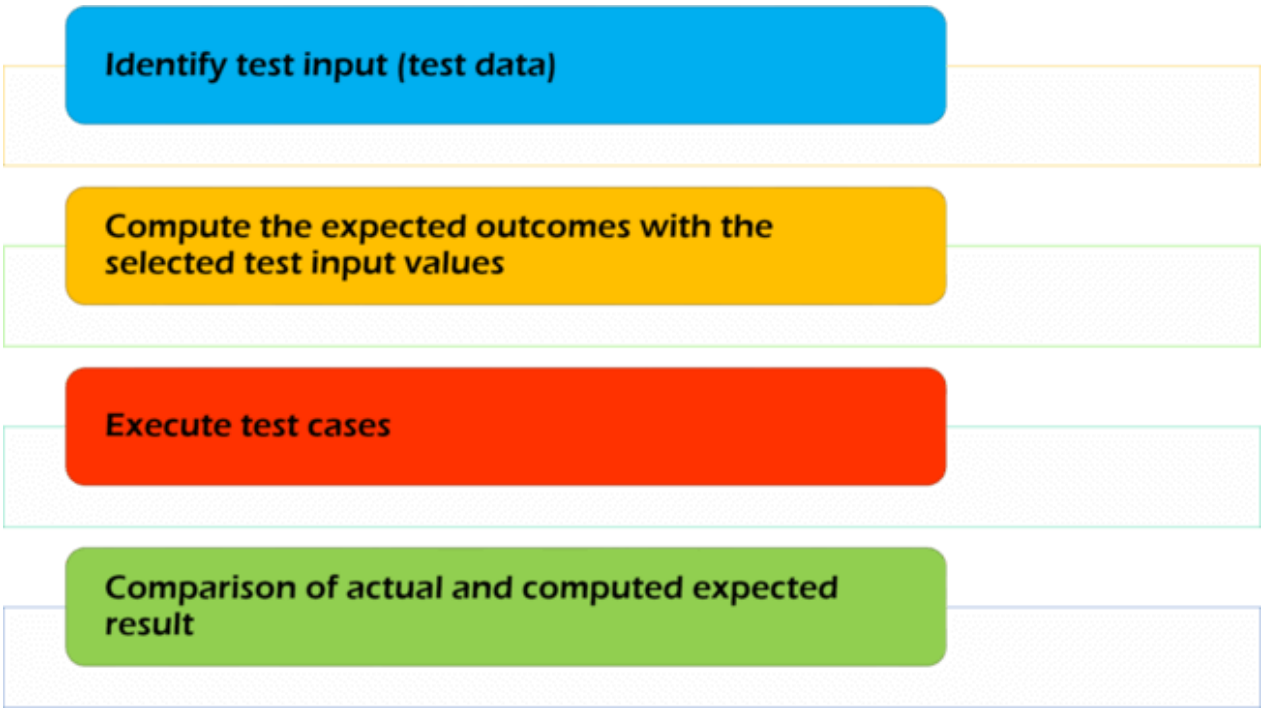
2.2. Τι ελέγχει

Το κύριο αντικείμενο που μας απασχολεί όταν μιλάμε για Functional Testing είναι οι λειτουργικότητες ενός συστήματος λογισμικού. Αυτό σημαίνει ότι επικεντρώνεται σε:

1) *Mainline functions*
   Έλεγχος των βασικών λειτουργιών της εφαρμογής
2) *Basic Usability*
   Έλεγχος της ευχρηστίας του συστήματος. Αυτός ο βασικός έλεγχος αφορά στην δυνατότητα του χρήστη να ελίσσεται από το ένα screen στο επόμενο χωρίς να αντιμετωπίζει δυσκολίες.
3) *Accessibility*
   Έλεγχος της ευκολίας πρόσβασης του χρήστη στο σύστημα.
4) *Errors Conditions*
   Χρήση διαφόρων τεχνικών testing για να ελεγχθεί η καταλληλόλητα των μηνυμάτων λάθους (είτε αφορά σε errors του συστήματος είτε σε exceptions). Εν ολίγοις, ελέγχεται αν η εφαρμογή “πετάει” τα σωστά μηνύματα λάθους στο χρήστη.

2.3. Πως υλοποιείται

Υπάρχει μια σειρά βημάτων που πρέπει να ακολουθήσουμε προκειμένου να εκτελέσουμε σωστά τη διαδικασία του testing. Αυτά τα βήματα περιγράφονται στο παρακάτω σχήμα:



Βήμα 0: *Software Engineering Requirements*

Πριν αρχίσει η διαδικασία του testing, ο Tester πρέπει να είναι σε θέση να κατανοεί τις απαιτήσεις παραγωγής του λογισμικού.

Βήμα 1: *Identify test input*

Καθορισμός δεδομένων εισόδου. Είναι πολύ σημαντικό να έχει επιβεβαιωθεί η ορθότητα / εγκυρότητα των δεδομένων εισόδου.

Βήμα 2 : *Compute the expected outcomes with the selected test input values*

Υπολογισμός των αναμενόμενων τιμών εξόδου βάσει των δεδομένων εισόδου.

Βήμα 3: *Execute test cases*

Εκτέλεση πιθανών σεναρίων και εύρεση ακριβών τιμών / αποτελεσμάτων.

Βήμα 4 : *Comparison of actual and computed expected result*

Στο τελευταίο βήμα συγκρίνουμε την πραγματική τιμή με την αναμενόμενη που υπολογίσαμε σε προηγούμενο βήμα.

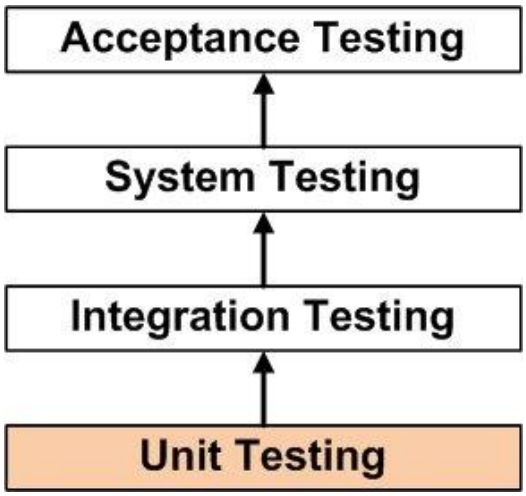
2.4. Κατηγορίες

Οι κύριες κατηγορίες του Functional Testing είναι οι εξής:

- Unit Testing
- Smoke Testing
- Sanity Testing
- Integration Testing
- White box testing
- Black Box testing
- User Acceptance testing
- Regression Testing

3. Unit Testing

Το Unit Testing είναι ένα επίπεδο του software testing, όπου ελέγχεται ξεχωριστά κάθε unit / component του κώδικα. Ο σκοπός του Unit Testing είναι η επικύρωση ότι κάθε μονάδα του λογισμικού λειτουργεί ακριβώς όπως σχεδιάστηκε να λειτουργεί.



Ως unit θεωρούμε τη μικρότερη δυνατή μονάδα λογισμικού που μπορεί να ελεγχθεί.

Το Unit Testing συνήθως έχει ένα ή περισσότερα inputs και ένα output.

Στο διαδικασιακό προγραμματισμό, ένα unit μπορεί να είναι ένα πρόγραμμα, ένα function, ένα procedure κλπ.

Στον αντικειμενοστραφή προγραμματισμό, η μικρότερη μονάδα είναι η μέθοδος, η οποία μπορεί να ανήκει σε μια κλάση ή σε μια υπερκλάση, ή και σε abstract class. Πολλοί θεωρούν ως unit και το module,

το οποίο όμως είναι μη αποδεκτό, καθώς ένα module μπορεί να περιέχει άνω του ενός units μέσα σε αυτό.

Στο Unit Testing χρησιμοποιούνται frameworks, drivers και fake objects (mocks).

3.1. Tasks

Τα tasks του Unit Testing είναι:

- Unit Test Plan
  - Prepare
  - Review
  - Rework
  - Baseline
- Unit Test Cases/Scripts
  - Prepare
  - Review
  - Rework
  - Baseline
- Unit Test
  - Perform

3.2. Ποιος το εκτελεί

Συνήθως είναι ευθύνη του ίδιου του προγραμματιστή να εκτελέσει Unit Testing. Σε κάποιες περιπτώσεις όμως, το τρέχουν ξεχωριστά οι software testers.

3.3. Tools

Το Unit Testing μπορεί να γίνεται manually από τον προγραμματιστή, παρόλα αυτά ο ίδιος χρησιμοποιεί πακέτα ή εργαλεία που τον διευκολύνουν.

Τα βασικότερα εργαλεία για Unit testing είναι τα παρακάτω:

1)NUnit

Είναι ένα Unit testing framework βασισμένο σε .NET platform.Επιτρέπει την εκτέλεση test scripts. Λειτουργεί όπως το

JUnit της Java, αλλά χρησιμοποιεί Console Runner για να φορτώνει και να εκτελεί τα tests.

2)JMockit

Είναι ένα open source Unit Testing tool, που διαθέτει συλλογή από tools και APIs. Οι προγραμματιστές χρησιμοποιούν τα tools και τα APIs για να γράψουν tests που τρέχουν με TestNG ή JUnit. Παρέχει 3 τύπους code coverage: Line Coverage, Path Coverage και Data Coverage.

3)Emma

Είναι open source toolkit που μετράει το Java Code Coverage. Ενεργοποιείται για κάθε προγραμματιστή της ομάδας. Υποστηρίζει class, line, method και basic block coverage και report types όπως txt, html, xml κλπ..

4)Quilt HTTP

Βασίζεται σε free cross platform και java Software development Tools. Χωρίς να χρησιμοποιεί τον βασικό κώδικα αναπαριστά κλάσεις και κώδικα μηχανής του JVM (Java Virtual Machine).

5)HtmlUnit

Είναι ένα Open Source Java Library (χωρίς γραφική διεπαφή) . Υποστηρίζει JavaScript και παρέχει διάφορα GUI features όπως forms, links, tables κλπ. Είναι java unit testing framework για testing σε web applications. Χρησιμοποιεί ένα Javascript Engine που ονομάζεται Mozilla Rhino. Είναι επίσης, συμβατό με πρωτόκολλα όπως το HTTP, HTTPS, submit methods όπως GET, POST και proxy servers.

6)Embunit

Είναι συντομία για το embedded Unit. Είναι free unit testing framework και έχει σχεδιαστεί για testing είτε από την πλευρά των developers είτε από την πλευρά των testers. Τρέχει για εφαρμογές γραμμένες σε C ή C++. Το σχέδιο του ακολουθεί τα patterns του JUnit.

7)SimpleTest

Ένα ακόμα open source framework. Σε αντίθεση με τα προηγούμενα, αυτό υποστηρίζει τη γλώσσα PHP και πρωτόκολλα όπως SSL, forms, proxies και authentication.

8)ABAP Unit

Είναι Unit testing tool που χρησιμοποιείται και στο manual testing και στο automation. Τα test γράφονται στο ABAP. Επιτρέπει ομαδοποιήσεις των test cases από διαφορετικά ABAP προγράμματα.

9)Typemock

Το Typemock Isolator είναι open source framework. Είναι κυρίως για έλεγχο κώδικα συστήματος. Μειώνει το χρόνο εύρεσης bugs. Διαθέτει APIs και έτοιμες μεθόδους και υποστηρίζει C/C++ σε περιβάλλον Windows. Ιδιαίτερα εύχρηστο και παρέχει μεγάλο code coverage.

10)LRDA

Πρόκειται για ένα εργαλείο επί πληρωμή για στατική και δυναμική ανάλυση και έλεγχο συστήματος λογισμικού. Παρέχει statement, decision και branch coverage, και επίσης καλύπτει όλο το εύρος της κατηγορίας του (requirement analysis έως και deployment) .

11)Microsoft unit testing Framework

Εργαλείο επι πληρωμή που βοηθάει στο testing στο περιβάλλον του Visual Studio. Συμπεριλαμβάνει VisualStudio TestTools - Unit Testing namespace (εκεί βρίσκεται το unit testing).

12)Unity Test Tools

Είναι ένα δωρεάν framework για τη δημιουργία και την εκτέλεση αυτόματων tests. Υποστηρίζει 3 components: Unit Tests, Integration Tests, Assertion Tests (για hard debugging).

13)Cantata

Είναι commercial framework που παρέχει test development environment. Χρησιμοποιείται για testing σε C και C++. Είναι ένα πλήρως αυτοματοποιημένο tool που υποστηρίζει μεγάλα datasets. Τα test scripts γράφονται σε C/C++ και διαθέτει test Script Manager. Επίσης, υποστηρίζει στατική ανάλυση και requirement testing.

14)Karma

Είναι Open Source testing framework. Τρέχει tests για JavaScript σε πραγματικές μηχανές. Αρκετά εύκολο στο debugging.

15)Jasmine

Unit Testing framework για JavaScript που επικεντρώνεται σε behavior-driven testing (δηλαδή testing που βασίζεται πιο πολύ στη συμπεριφορά του χρήστη).

16)Mocha

Open Source Testing Framework για JavaScript που τρέχει σε Node.js. Βρίσκεται διαθέσιμο στο github. Έχει επιπλέον features όπως test coverage report, browser support, report test duration κλπ..

17)Parasoft

Είναι επι πληρωμή εργαλείο για unit testing σε C/C++ και παρέχει στατική ανάλυση για κάθε μια από αυτές τις γλώσσες. Εντοπίζει προβλήματα στη λειτουργικότητα της εφαρμογής και προβλήματα που προκαλούν crash. Βοηθάει στην εκτέλεση functional tests,

παρέχοντας runtime error detection requirement traceability, debugger integration και detailed reporting.

18)JUnit

Είναι open source unit testing framework για εφαρμογές σε Java. Βασίζεται στην ιδέα "First testing than coding". Τα test data ελέγχονται πρώτα πριν ενσωματωθούν στη ροή του κώδικα.

19)TestNG

Είναι κι αυτό open source automation testing framework για εφαρμογές σε Java. Έχει επηρεαστεί από το JUnit και το NUnit.

20) JTest

Αποτελεσματικό framework για testing εφαρμογών σε Java, ενώ παρέχει και στατική ανάλυση του κώδικα. Περιλαμβάνει έλεγχο ροής , metrics analysis, runtime error detection κλπ. Επίσης, είναι χρήσιμο σε Regression Testing το οποίο αναλύεται παρακάτω.

4. Smoke Testing

Το Smoke Testing είναι ένα είδος software testing που αποφασίζει κατά πόσο είναι ευσταθές το build. Τα Smoke Tests τρέχουν σε κάθε build της εφαρμογής.

Το Smoke Testing είναι η διαδικασία, όπου το build αναπτύσσεται σε QA (Quality Assurance) environment και επιβεβαιώνεται το stability της εφαρμογής.

Ονομάζεται επίσης και Build Verification Testing / Confidence Testing.

4.1. Πως υλοποιείται

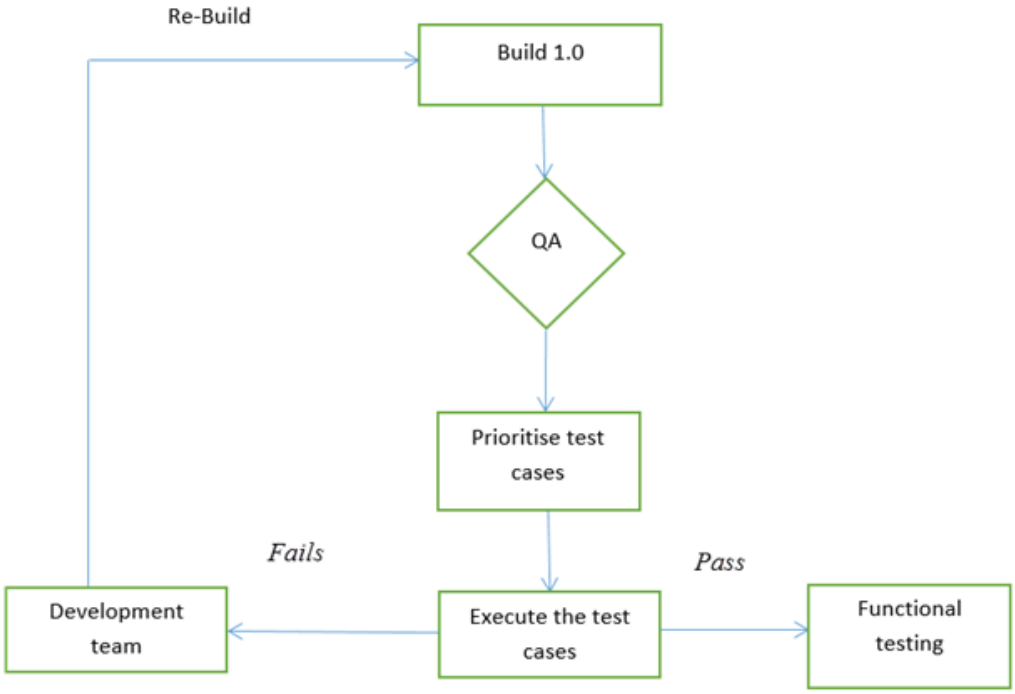
Το Smoke Testing υλοποιείται είτε Manually είτε με τη χρήση automation.

Το πιο συνηθισμένο είναι να γίνεται χειροκίνητα. Με κάθε release του build στο QA περιβάλλον, δίνεται υψηλή προτεραιότητα στα test cases των functionalities. Αν το testing είναι επιτυχές, συνεχίζεται το functional testing. Αν το test αποτύχει, το build απορρίπτεται και στέλνεται στο development team για διόρθωση.

Στην περίπτωση του automation, αντί να το κάνουμε κάθε φορά manually σε κάθε νέο build, τρέχουμε πάνω στο build κάποια δεδομένα smoke test cases. Αυτό μας επιβεβαιώνει αν τα βασικά functionalities λειτουργούν όπως πρέπει. Αν το test αποτύχει, διορθώνεται και

ξαναγίνεται deploy . Αυτή η διαδικασία εξοικονομεί χρόνο και εξασφαλίζει ποιότητα στο QA.

Η αναλυτική διαδικασία του Smoke Testing φαίνεται παρακάτω στο σχήμα:



5. Regression Testing

Το Regression Testing είναι ένα είδος testing που επικυρώνει ότι οι αλλαγές που έχουν γίνει στον κώδικα δεν επηρεάζουν την υπάρχουσα λειτουργικότητα του προϊόντος.

Η πραγματική του χρησιμότητα είναι η επιβεβαίωση ότι το προϊόν λειτουργεί σωστά με μια νέα λειτουργικότητα, διορθωμένα bugs που ενδεχομένως να είχαν προκύψει ή οποιαδήποτε αλλαγή είχε γίνει στη λειτουργία της εφαρμογής.

Σε αυτή την κατηγορία testing, τα test cases που είχαν εκτελεστεί σε προηγούμενες φάσεις πρέπει να εκτελεστούν ξανά.

Το test μπορεί να γίνει σε νέο build όταν υπάρχουν σημαντικές αλλαγές στην αρχική λειτουργία της εφαρμογής.

Το Regression Testing είναι μια μέθοδος επαλήθευσης. Τα test cases είναι αυτοματοποιημένα καθώς εκτελούνται επαναληπτικά. Υπάρχει πάντα η επιλογή να τρέξουν τα test cases χωρίς τη χρήση automation (manually). Σε αυτή την περίπτωση θα χαθεί άσκοπα χρόνος.

Αξίζει να σημειωθεί ότι αυτό το είδος testing δεν είναι εξαρτώμενο από τη γλώσσα προγραμματισμού, δηλαδή μπορεί να εφαρμοστεί σε αρκετές διαφορετικές γλώσσες (Java,C++,C# κλπ.).

5.1. Παράδειγμα

Θεωρούμε ένα προϊόν X το οποίο τρέχει τις εξής λειτουργίες: Confirmation, Acceptance, Dispatch όταν πατιέται το κουμπί Confirm, Accept, Dispatch αντίστοιχα.

Έστω ότι προκύπτουν κάποια issues στο Confirmation. Γίνονται προφανώς, κάποιες αλλαγές στον κώδικα. Μετά τις αλλαγές, δεν ελέγχεται μόνο το Confirmation, αλλά ελέγχεται και η ομαλή λειτουργία του Acceptance και του Dispatch, ώστε να επιβεβαιωθεί ότι οι αλλαγές στο Confirmation δεν επηρεάζουν άλλες λειτουργίες.

5.2. Πότε εκτελείται

Συνήθως, το Regression Testing εκτελείται μετά από εισαγωγή νέων functionalities ή αλλαγές στα υπάρχοντα, χωρίς όμως να είναι απόλυτο.

5.3. Τύποι

Οι τύπου του Regression Testing είναι:

1) *Unit Regression*
   Γίνεται κατά το Unit Testing και ο κώδικας ελέγχεται μεμονωμένα. Πχ. Εξαρτήσεις με το unit που ελέγχεται μπλοκάρονται ώστε να μπορεί να ελεγχθεί ξεχωριστά.
2) *Partial Regression*
   Επιβεβαιώνει ότι ο κώδικας λειτουργεί σωστά μετά από αλλαγές
3) *Complete Regression*
   Γίνεται όταν οι αλλαγές στον κώδικα γίνονται σε μεγάλο αριθμό από modules ή η επιρροή σε άλλο module από κάποια αλλαγή δεν είναι σίγουρη.

5.4. Μπορεί να εκτελεστεί manually;

Αρχικά, η εκτέλεση των test είναι μια απλή διαδικασία εκτέλεσης των test cases στο AUT (application under test) , παρέχοντας test data ως είσοδο και συγκρίνοντας το αποτέλεσμα με το αναμενόμενο αποτέλεσμα.

Παρόλα αυτά, όπως προαναφέρθηκε η διαδικασία λόγω της επαναληπτικότητας της είναι πολύ χρονοβόρα. Συνεπώς, η βέλτιστη λύση είναι χρήση automation.

5.5. Tools

Το προτεινόμενο Automated Regression Testing Tool είναι το Ranorex Studio.

Παρόλα αυτά, υπάρχουν πολλά εργαλεία όπως:

- Selenium
- Katalon Studio
- AdventNet QEngine
- Regression Tester
- vTest
- Watir
- actiWate
- Rational Functional Tester
- SilkTest
- TimeShiftX

Σε οποιοδήποτε εργαλείο πραγματοποιηθεί το testing, θα ήταν συνετό το τακτικό update των test cases.

6. Sanity Testing

Πρόκειται για μια υποκατηγορία του Regression Testing. Εκτελείται για να επιβεβαιώσει ότι παρά τις αλλαγές ο κώδικας παραμένει λειτουργικός.

Στην ουσία είναι ένα break για έλεγχο. Αυτός ο έλεγχος αφορά το build. Επιβεβαιώνει αν μπορεί ο tester να προχωρήσει στο testing του build ή όχι.

Κατά τη διάρκεια του Sanity Testing, εκεί όπου εστιάζουν οι testers είναι η λειτουργικότητα της εφαρμογής και όχι το detailed testing.

6.1. Παράδειγμα

Ας πάρουμε ως παράδειγμα ένα project όπου οι βασικές λειτουργίες είναι το login, display profile, user registration κλπ. και όπου υπάρχει η απαίτηση στο login page ο κωδικός πρόσβασης να αποτελείται από λιγότερο από 4 αλφαριθμητικά και από τα requirements ο κωδικός πρόσβασης δεν πρέπει να αποτελείται από λιγότερους από 8 χαρακτήρες. Αυτό πρέπει να αναφερθεί από το testing team στο development team για να επιλυθεί. Το development το διορθώνει και το στέλνει πίσω για επανέλεγχο. Ελέγχεται, επίσης αν υπάρχει σχέση με άλλο συσχετισμένο functionality.

Σε συνέχεια του παραδείγματος, υπάρχει functionality που κάνει update τον κωδικό στο user profile page. Ως μέρος του sanity testing, το login page επιβεβαιώνεται μαζί με το profile page για να βεβαιωθεί ότι οι έλεγχοι λειτουργούν σωστά και στα 2 μέρη.

6.2. Χαρακτηριστικά

Τα χαρακτηριστικά του Sanity Testing είναι:

1. *Υποκατηγορία του Regression Testing*
   Το Sanity Testing είναι υποκατηγορία του regression testing και επικεντρώνεται σε μικρότερα μέρη της εφαρμογής.
2. *Unscripted:*
   Τις περισσότερες φορές γίνεται χωρίς υπάρχον script.
3. *Not documented:*
   Δεν υποστηρίζεται από documentation.
4. *Narrow and deep:*
   Αφορά σε βαθύτερες προσεγγίσεις του testing.
5. *Performed by testers:*
   Τις περισσότερες φορές εκτελείται από testers

6.3. Πλεονεκτήματα

Τα πλεονεκτήματα του Sanity Testing είναι:

- Βρίσκει λάθη / ελαττώματα του κώδικα στη βασική λειτουργικότητα
- Γίνεται σε λιγότερο χρόνο αφού δεν απαιτείται documentation
- Αν βρεθούν λάθη, το project απορρίπτεται αμέσως, το οποίο είναι χρήσιμο στην εξοικονόμηση χρόνου για εκτέλεση άλλων regression tests.

7. Integration Testing

Το Integration Testing είναι ένα επίπεδο ελέγχου λογισμικού όπου μεμονωμένες μονάδες συνδυάζονται και δοκιμάζονται ως ομάδα. Ο σκοπός αυτού του επιπέδου δοκιμών είναι να αποκαλυφθούν σφάλματα στην αλληλεπίδραση μεταξύ ολοκληρωμένων μονάδων. Τα test drivers και τα test stubs χρησιμοποιούνται για να βοηθήσουν στη δοκιμή ενοποίησης.

7.1. Κάποιοι χρήσιμοι ορισμοί αναφορικά

Δοκιμή ολοκλήρωσης:

Έλεγχος που πραγματοποιήθηκε για την έκθεση ελαττωμάτων στις διεπαφές και στις αλληλεπιδράσεις μεταξύ ολοκληρωμένων εξαρτημάτων ή συστημάτων.

Δοκιμή ενοποίησης στοιχείων:

Έλεγχος που πραγματοποιήθηκε για την έκθεση ελαττωμάτων στις διεπαφές και αλληλεπίδραση μεταξύ ολοκληρωμένων συστατικών

Δοκιμή ολοκλήρωσης συστήματος:

Δοκιμή της ολοκλήρωσης συστημάτων και πακέτων.

7.2. Αναλογίες

Κατά τη διαδικασία κατασκευής ενός στυλό, το καπάκι, το σώμα, η ουρά και το κλιπ, το φυσίγγιο μελανιού και το σημείο σφαίρας παράγονται ξεχωριστά και η μονάδα δοκιμάζεται ξεχωριστά. Όταν δύο ή περισσότερες μονάδες είναι έτοιμες, συναρμολογούνται και πραγματοποιείται έλεγχος ενοποίησης. Για παράδειγμα, αν το καπάκι ταιριάζει στο σώμα ή όχι.

7.3. Μέθοδος

Μπορεί να χρησιμοποιηθεί οποιαδήποτε μέθοδος δοκιμής: Black Box, White Box, Gray Box. Κανονικά, η μέθοδος εξαρτάται από τον ορισμό της «μονάδας» (unit) .

7.4. Tasks

Τα tasks του Integration Testing είναι τα εξής:

- Integration Test Plan

    - Prepare
    - Review
    - Rework
    - Baseline
- Integration Test Cases/Scripts
    - Prepare
    - Review
    - Rework
    - Baseline
- Integration Test
    - Perform

7.5. Πότε εκτελείται

Είναι το δεύτερο επίπεδο δοκιμών που πραγματοποιείται μετά το unit testing και πριν από το System Testing.

7.6. Ποιος πραγματοποιεί

Οι ίδιοι οι προγραμματιστές ή οι testers εκτελούν Integration Testing.

7.7. Προσεγγίσεις

-Το Big Bang είναι μια προσέγγιση στο Integration Testing όπου όλες ή οι περισσότερες μονάδες συνδυάζονται και δοκιμάζονται ταυτόχρονα. Αυτή η προσέγγιση ακολουθείται όταν η ομάδα των testers λαμβάνει ολόκληρο το λογισμικό σε ένα πακέτο. Ποια είναι λοιπόν η διαφορά μεταξύ του Big Bang Integration Testing και του System Testing; Λοιπόν, η πρώτη δοκιμάζει μόνο τις αλληλεπιδράσεις μεταξύ των μονάδων, ενώ η δεύτερη δοκιμάζει ολόκληρο το σύστημα.

-Το Top Down είναι μια προσέγγιση στο Integration Testing όπου οι μονάδες ανώτερου επιπέδου δοκιμάζονται πρώτα και οι μονάδες χαμηλότερου επιπέδου δοκιμάζονται βήμα προς βήμα μετά από αυτό. Αυτή η προσέγγιση ακολουθείται όταν ακολουθείται η προσέγγιση ανάπτυξης από πάνω προς τα κάτω. Απαιτούνται Test Stubs για την προσομοίωση μονάδων χαμηλότερου επιπέδου που ενδέχεται να μην είναι διαθέσιμες κατά τις αρχικές φάσεις.

-Το Bottom Up είναι μια προσέγγιση στο Integration Testing όπου οι μονάδες κατώτατου επιπέδου δοκιμάζονται πρώτα και οι μονάδες ανώτερου επιπέδου βήμα προς βήμα μετά από αυτήν. Αυτή η προσέγγιση ακολουθείται όταν ακολουθείται η προσέγγιση ανάπτυξης από κάτω προς τα πάνω. Απαιτούνται δοκιμαστικά προγράμματα οδήγησης για την προσομοίωση μονάδων υψηλότερου επιπέδου που ενδέχεται να μην είναι διαθέσιμες κατά τις αρχικές φάσεις.

-Το Sandwich / Hybrid είναι μια προσέγγιση στο Integration Testing που αποτελεί συνδυασμό προσεγγίσεων Top Down και Bottom Up.

8. Black Box Testing

Το Black Box Testing, γνωστό και ως Behavioral Testing ή Specification Based Techniques, είναι μια μέθοδος ελέγχου λογισμικού στην οποία η εσωτερική δομή / σχεδιασμός / υλοποίηση του αντικειμένου που δοκιμάζεται δεν είναι γνωστή στον tester. Αυτές οι δοκιμές μπορεί να είναι λειτουργικές ή μη λειτουργικές, αν και συνήθως λειτουργικές.

Αυτή η μέθοδος ονομάζεται έτσι επειδή το πρόγραμμα λογισμικού, στα μάτια του ελεγκτή, είναι σαν ένα μαύρο κουτί, μέσα στο οποίο δεν μπορεί να δει κανείς. Αυτή η μέθοδος προσπαθεί να εντοπίσει σφάλματα στις ακόλουθες κατηγορίες:

-Λανθασμένες ή μη υπάρχουσες λειτουργίες

-Σφάλματα διασύνδεσης

-Σφάλματα σε δομές δεδομένων ή πρόσβαση σε εξωτερική βάση δεδομένων

-Σφάλματα συμπεριφοράς ή απόδοσης

-Σφάλματα αρχικοποίησης και τερματισμού

8.1. Ορισμοί

Black Box Testing: Δοκιμές, λειτουργικές ή μη λειτουργικές, χωρίς αναφορά στην εσωτερική δομή του component ή του συστήματος.

Τεχνική σχεδίασης Black Box testing:

Διαδικασία για την εξαγωγή ή / και την επιλογή περιπτώσεων δοκιμής βάσει ανάλυσης των προδιαγραφών, λειτουργικών ή μη λειτουργικών, ενός component ή συστήματος χωρίς αναφορά στην εσωτερική δομή του.

8.2. Παράδειγμα

Ένας tester, χωρίς γνώση των εσωτερικών δομών ενός ιστότοπου, δοκιμάζει τις ιστοσελίδες χρησιμοποιώντας ένα πρόγραμμα περιήγησης , πχ παροχή εισόδων (κλικ, πλήκτρα) και επαλήθευση των εξόδων σε σχέση με το αναμενόμενο αποτέλεσμα.

8.3. Πού εκτελείται

Το Black Box Testing εφαρμόζεται στα ακόλουθα επίπεδα:

-Δοκιμή ολοκλήρωσης

-Δοκιμή συστήματος

-Δοκιμή αποδοχής

Όσο υψηλότερο είναι το επίπεδο, και ως εκ τούτου όσο μεγαλύτερο και πιο περίπλοκο το κουτί, τόσο περισσότερο χρησιμοποιείται η μέθοδος.

8.4. Τεχνικές

Equivalence Partitioning: Πρόκειται για μια τεχνική σχεδιασμού δοκιμής λογισμικού που περιλαμβάνει τη διαίρεση των τιμών εισόδου σε έγκυρα και μη έγκυρα διαμερίσματα και την επιλογή αντιπροσωπευτικών τιμών από κάθε διαμέρισμα ως δεδομένα δοκιμής.

Ανάλυση οριακής τιμής: Πρόκειται για μια τεχνική σχεδιασμού testing που περιλαμβάνει τον προσδιορισμό των ορίων για τις τιμές εισόδου και την επιλογή τιμών που βρίσκονται στα όρια και ακριβώς μέσα / έξω από τα όρια ως test data.

Cause-Effect Graphing: Πρόκειται για μια τεχνική σχεδιασμού δοκιμής λογισμικού που περιλαμβάνει τον εντοπισμό των περιπτώσεων (συνθήκες εισόδου) και των αποτελεσμάτων (συνθήκες εξόδου), την παραγωγή γραφήματος Cause-Effect και τη δημιουργία αντίστοιχων περιπτώσεων δοκιμής.

8.5. Πλεονεκτήματα

Οι δοκιμές γίνονται από την πλευρά του χρήστη και θα βοηθήσουν στον εντοπισμό αποκλίσεων στις προδιαγραφές.

Ο tester δεν χρειάζεται να γνωρίζει γλώσσες προγραμματισμού ή πώς έχει υλοποιηθεί το λογισμικό.

Οι δοκιμές μπορούν να διεξαχθούν από έναν οργανισμό ανεξάρτητο από τους προγραμματιστές, επιτρέποντας μια αντικειμενική προοπτική.

8.6. Tool

Το testing μπορεί να γίνει manually, όμως παρόλα αυτά υπάρχει η δυνατότητα χρήσης automated tool. Το προτεινόμενο automated tool είναι το Black Box Automatation Tool για desktop,Web & Mobile Testing της Ranorex με τα εξής χαρακτηριστικά:

-Edge Cases

Τα edge cases επικυρώνουν την ελάχιστη και τη μέγιστη αποδεκτή τιμή ενός μόνο πεδίου, όπως για μια ημερομηνία ή τη διάρκεια ενός κωδικού πρόσβασης. Επομένως, εάν ένα πεδίο πρέπει να επιτρέπει καταχωρήσεις στην περιοχή από 1 έως 100, δημιουργήστε ένα test case για την τιμή 1 και ενα για την τιμή 100. Δεν είναι απαραίτητο να δοκιμαστούν όλες τις τιμές στο μεταξύ.

-Icon corner case testing

Τα corner cases δοκιμάζουν συνδυασμούς ακρών. Η ιδέα εδώ είναι ότι ενώ τα μεμονωμένα πεδία μπορεί να λειτουργούν σωστά, ο συνδυασμός μπορεί να μην λειτουργεί. Για παράδειγμα, εάν η αίτησή σας αποδεχτεί μια ημερομηνία έναρξης και μια ημερομηνία λήξης, μια γωνιακή περίπτωση θα ελέγξει έναν συνδυασμό, όπως η νωρίτερα δυνατή ημερομηνία έναρξης με την νωρίτερα δυνατή ημερομηνία λήξης.

-Icon boundary value analysis

Οι οριακές περιπτώσεις δοκιμάζουν τιμές ακριβώς έξω και μέσα από τα edge cases. Για να ελέγξετε την τιμή που ισχύει μεταξύ 1 και 100, δημιουργήστε περιθώρια για 0, 1 και 2. συν 99, 100 και 101. Εάν η εφαρμογή χειριστεί σωστά τις οριακές τιμές, τότε θα πρέπει να χειριστείτε σωστά όλες τις άλλες τιμές. Αυτή η τεχνική αναφέρεται επίσης ως «ανάλυση οριακής τιμής»

-Icon user scenario testing

Τα σενάρια περιγράφουν τον στόχο που θέλει να επιτύχει ένας χρήστης, όπως τον έλεγχο του υπολοίπου του λογαριασμού του ή την αλλαγή του κωδικού πρόσβασης. Στην ανάπτυξη βάσει συμπεριφοράς (BDD), τα σενάρια ακολουθούν μια συγκεκριμένη σύνταξη "Given-When-Then" που διευκολύνει τη δημιουργία test cases.

-Icon decision table testing

Σε αυτήν την τυπική τεχνική δοκιμών, οι αιτίες / είσοδοι και τα αποτελέσματα / αποτελέσματα στην εφαρμογή είναι διατεταγμένες σε έναν πίνακα, με κάθε κελί να περιέχει έναν μοναδικό συνδυασμό αιτίας /

αποτελέσματος. Στη συνέχεια, δημιουργείται test case για κάθε τιμή κελιού.

-Icon error guessing technique

Κατά την εικασία σφάλματος, οι υπεύθυνοι δοκιμών χρησιμοποιούν τις γνώσεις τους για το σύστημα και την εμπειρία τους για να μαντέψουν τους τύπους ελαττωμάτων που ενδέχεται να υπάρχουν και να δημιουργήσουν test cases για αυτά. Αυτή η τεχνική είναι παρόμοια με τις εξερευνητικές δοκιμές, καθώς και οι δύο βασίζονται σε μεγάλο βαθμό στην ικανότητα και τη γνώση του tester.

9. White Box Testing

White Box Testing (επίσης γνωστή ως Clear Box Testing, Open Box Test, Glass Box Testing, Transparent Box Testing, Code-Based Test ή Structural Testing) είναι μια μέθοδος testing στην οποία η εσωτερική δομή / σχεδιασμός / υλοποίηση του αντικειμένου που δοκιμάζεται είναι γνωστή στον tester. Ο tester επιλέγει τις εισόδους μέσω του κώδικα και καθορίζει τις κατάλληλες εξόδους. Η τεχνογνωσία προγραμματισμού και η γνώση εφαρμογής είναι απαραίτητες.

9.1. Ορισμός

Δοκιμή βασισμένη σε ανάλυση της εσωτερικής δομής του component ή του συστήματος.

Τεχνική σχεδιασμού white box testing:

Διαδικασία για την εξαγωγή ή / και την επιλογή των test cases βάσει ανάλυσης της εσωτερικής δομής ενός component ή συστήματος.

9.2. Παράδειγμα

Ένας υπεύθυνος δοκιμών, συνήθως ένας προγραμματιστής, μελετά τον κώδικα εφαρμογής ενός συγκεκριμένου πεδίου σε μια ιστοσελίδα, καθορίζει όλες τις νόμιμες (έγκυρες και μη έγκυρες) ΚΑΙ παράνομες εισόδους και επαληθεύει τα αποτελέσματα έναντι των αναμενόμενων αποτελεσμάτων.

Αυτή η μέθοδος ονομάζεται έτσι επειδή το πρόγραμμα λογισμικού, στα μάτια του tester, είναι σαν ένα λευκό / διαφανές πλαίσιο, μέσα στο οποίο βλέπει κανείς καθαρά.

Το White Box Testing είναι σαν το έργο ενός μηχανικού που εξετάζει τον κινητήρα για να δει γιατί το αυτοκίνητο δεν κινείται.

9.3. Που εφαρμόζεται

Η μέθοδος White Box Testing εφαρμόζεται στα ακόλουθα επίπεδα δοκιμών λογισμικού:

-*Unit Testing*:

Για δοκιμές μέσα σε μια μονάδα.

-*Integration Testing*:

Για δοκιμές μεταξύ μονάδων.

-*System Testing*:

Για δοκιμές διαδρομών μεταξύ υποσυστημάτων.

Ωστόσο, εφαρμόζεται κυρίως στο Unit Testing.

10. Διαφορές Black Box-White Box Testing

Παρακάτω, κάποιες ενδεικτικές διαφορές Black Box και White Box Testing:

1. Ο κύριος στόχος του Black Box testing είναι να ελέγξουμε τη λειτουργικότητα / συμπεριφορά της εφαρμογής, ενώ στο White Box Testing ο κύριος στόχος είναι να δοκιμαστεί η υποδομή της εφαρμογής.

2. Το Black Box Testing μπορεί να εκτελεστεί από έναν tester χωρίς καμία γνώση του κώδικα του AUT (Application Under Test), εν αντιθέσει με το White Box Testing όπου ο tester πρέπει να έχει τη γνώση της εσωτερικής δομής και του τρόπου λειτουργίας του.

3. Στην περίπτωση του Black Box το testing μπορεί να πραγματοποιηθεί μόνο με χρήση του GUI. Στο White Box το testing μπορεί να γίνει σε πρώιμο στάδιο πριν το GUI ετοιμαστεί.

4. Το Black Box testing δεν μπορεί να καλύψει όλες τις πιθανές εισόδους, η δοκιμή με White Box είναι πιο διεξοδική καθώς μπορεί να δοκιμάσει κάθε περίπτωση.

5. Ορισμένες τεχνικές δοκιμών Black Box περιλαμβάνουν ανάλυση οριακής τιμής, κατανομή ισοδυναμίας, εκτίμηση σφαλμάτων κ.λπ. Ορισμένες τεχνικές δοκιμής White Box περιλαμβάνουν δοκιμή υπό όρους, δοκιμή ροής δεδομένων, δοκιμή βρόχου κ.λπ.

6. Test Cases πρέπει να γράφονται με βάση την προδιαγραφή απαιτήσεων στο Black Box. Στο White Box test cases θα πρέπει να γράφονται με βάση το Αναλυτικό Σχεδιαστικό Έγγραφο.

7. Το Black Box πραγματοποιείται από testers, ενώ στο White Box είναι η ευθύνη των προγραμματιστών λογισμικού.

8. Δεν απαιτείται γνώση προγραμματισμού και εφαρμογής στον έλεγχο με Black Box. Απαιτείται γνώση προγραμματισμού και εφαρμογής στον έλεγχο με White Box.

9. Το Black Box είναι λιγότερο χρονοβόρο και εξαντλητικό.

10.Μικρότερη κάλυψη δοκιμών στο Black Box Testing.

11. Στο White Box Testing υπάρχει η δυνατότητα έγκαιρου εντοπισμού σφαλμάτων. Στο Black Box πρέπει να αναπτυχθεί ο κώδικας.

10.1. Πλεονεκτήματα

Η δοκιμή μπορεί να ξεκινήσει σε προγενέστερο στάδιο. Δεν χρειάζεται να περιμένετε να είναι διαθέσιμο το GUI.

Η δοκιμή είναι πιο διεξοδική, με δυνατότητα κάλυψης των περισσότερων διαδρομών.

11. User Acceptance Testing

User Acceptance Testing (UAT) είναι ένας τύπος testing που πραγματοποιείται από τον τελικό χρήστη ή τον πελάτη για την επαλήθευση / αποδοχή του συστήματος λογισμικού πριν από τη μεταφορά της εφαρμογής λογισμικού στο production environment. Το UAT γίνεται στην τελική φάση των δοκιμών μετά τη λειτουργία, την ολοκλήρωση και τον έλεγχο του συστήματος.

11.1. Γιατί γίνεται

Ο κύριος σκοπός της UAT είναι να επικυρώσει την επιχειρηματική ροή από άκρο σε άκρο ΔΕΝ επικεντρώνεται σε λάθη «αισθητικά», ορθογραφικά λάθη ή δοκιμές συστήματος. Ο Έλεγχος Αποδοχής Χρήστη πραγματοποιείται σε ξεχωριστό περιβάλλον δοκιμών με ρύθμιση δεδομένων παραγωγής. Είναι ένα είδος black box testing όπου θα συμμετέχουν δύο ή περισσότεροι τελικοί χρήστες.

11.2. Ποιος το εκτελεί

Το UAT το εκτελούν οι:

-Πελάτης

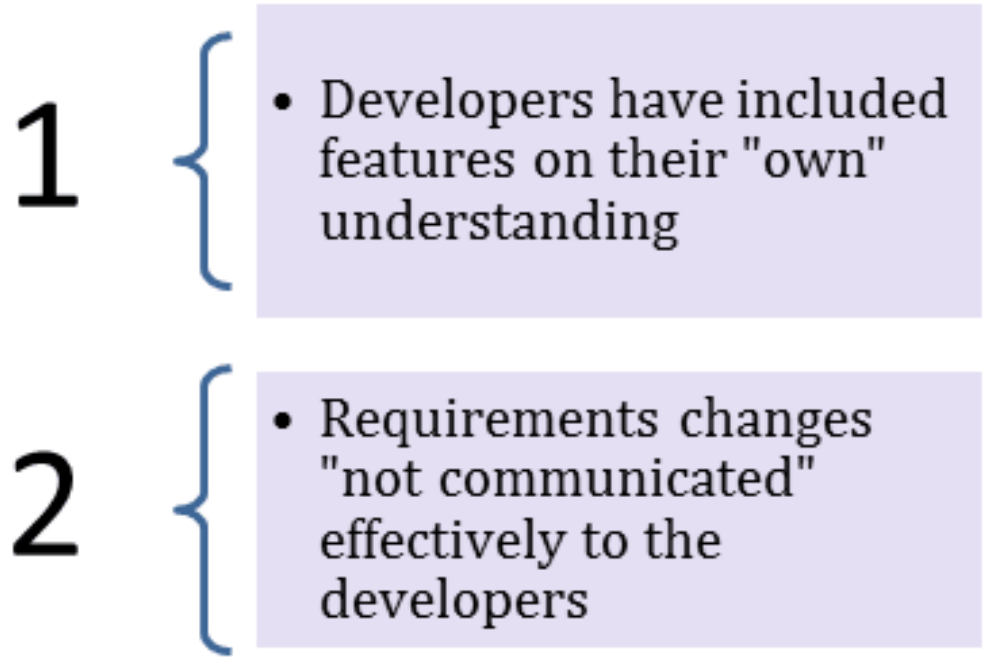
-Τελικοί χρήστες

11.3. Αναγκαιότητα Δοκιμής

Όταν το λογισμικό έχει υποβληθεί σε Unit Testing, Integration Testing και System Testing, η ανάγκη για User Acceptance Testing μπορεί να φαίνεται περιττή. Ωστόσο, απαιτείται έλεγχος αποδοχής επειδή:

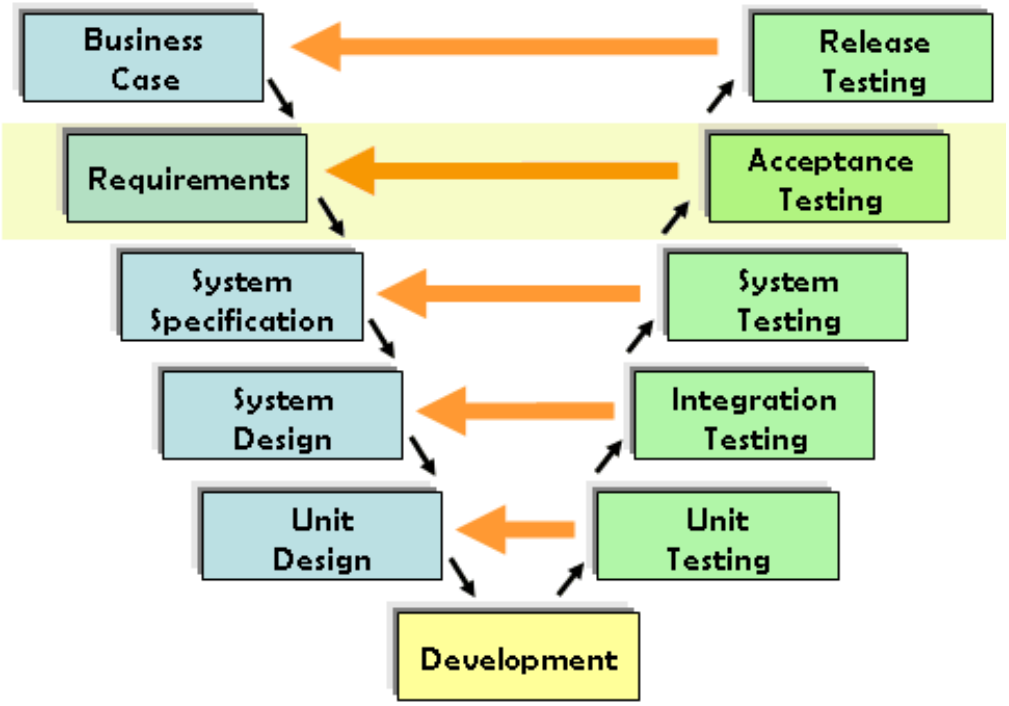
-Οι προγραμματιστές προγραμματίζουν το λογισμικό βάσει εγγράφων απαιτήσεων που είναι η «δική τους» κατανόηση των απαιτήσεων και ενδέχεται να μην είναι πραγματικά αυτό που χρειάζεται ο πελάτης από το λογισμικό.

-Οι αλλαγές στις απαιτήσεις κατά τη διάρκεια του έργου ενδέχεται να μην κοινοποιούνται αποτελεσματικά στους προγραμματιστές.



11.4. V-Model

Στο V-Model, το User Acceptance Testing αντιστοιχεί στη φάση απαίτησης του κύκλου ζωής ανάπτυξης λογισμικού (SDLC).



11.5. Προϋποθέσεις

-Οι επιχειρηματικές απαιτήσεις πρέπει να είναι διαθέσιμες.

-Ο κώδικας πρέπει να έχει αναπτυχθεί πλήρως

-Το Unit Testing, System Testing και Integration Testing πρέπει να έχουν ολοκληρωθεί.

-Δεν πρέπει να υπάρχουν Showstoppers, μεγάλα ή μεσαία ελαττώματα στη φάση Integration Testing

-Μόνο το «αισθητικό» σφάλμα είναι αποδεκτό πριν από το UAT

- Το Regression Testing πρέπει να ολοκληρωθεί χωρίς σοβαρά ελαττώματα

-Όλα τα αναφερόμενα ελαττώματα πρέπει να διορθωθούν και να δοκιμαστούν πριν από το UAT

-Ο πίνακας ανιχνευσιμότητας για όλες τις δοκιμές πρέπει να συμπληρωθεί

-Το περιβάλλον UAT πρέπει να είναι έτοιμο

11.6. Πώς γίνεται

Το UAT γίνεται από τους προοριζόμενους χρήστες του συστήματος ή του λογισμικού. Αυτός ο τύπος testing συμβαίνει συνήθως στην τοποθεσία του πελάτη που είναι γνωστή ως δοκιμή Beta. Μόλις ικανοποιηθούν τα κριτήρια εισαγωγής για το UAT, ακολουθούν οι εργασίες που πρέπει να εκτελέσουν οι υπεύθυνοι δοκιμών:

-Ανάλυση επιχειρηματικών απαιτήσεων

-Δημιουργία προγράμματος δοκιμών UAT

-Προσδιορισμός σεναρίων χρήσης

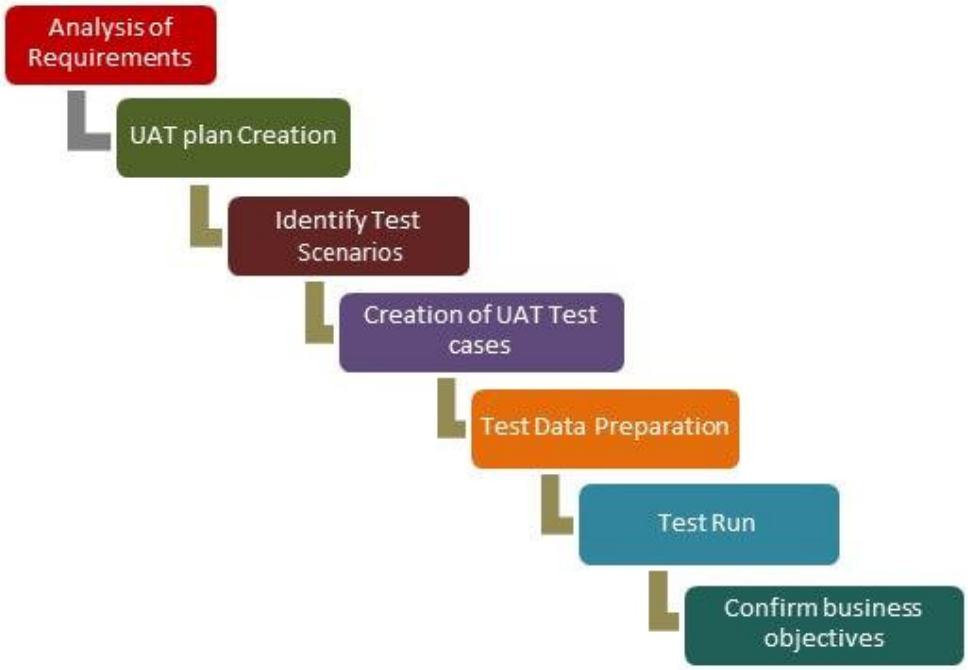
-Δημιουργία UAT Test Case

-Προετοιμασία δεδομένων δοκιμής

-Εκτέλεση δοκιμών

-Καταγραφή αποτελεσμάτων

-Επιβεβαίωση επιχειρηματικών στόχων



Βήμα 1) *Ανάλυση επιχειρηματικών απαιτήσεων*

Μία από τις πιο σημαντικές δραστηριότητες στο UAT είναι ο εντοπισμός και η ανάπτυξη σεναρίων δοκιμών. Αυτά τα σενάρια δοκιμής προέρχονται από τα ακόλουθα έγγραφα:

-Project Charter

-Business Use Cases

-Process Flow Diagrams

-Business Requirements Document (BRD)

-System Requirements Specification (SRS)

Βήμα 2) *Δημιουργία προγράμματος UAT*

Το σχέδιο δοκιμών UAT περιγράφει τη στρατηγική που θα χρησιμοποιηθεί για την επαλήθευση και τη διασφάλιση ότι μια εφαρμογή πληροί τις επιχειρηματικές της απαιτήσεις. Τεκμηριώνει τα κριτήρια εισόδου και εξόδου για UAT, σενάρια δοκιμών και προσέγγιση test cases.

Βήμα 3) *Προσδιορίστε τα σενάρια δοκιμής και τις περιπτώσεις δοκιμής*

Προσδιορισμός σεναρίων δοκιμής σε σχέση με την επιχειρηματική διαδικασία υψηλού επιπέδου και δημιουργία test cases με σαφή βήματα δοκιμής. Τα test cases πρέπει να καλύπτουν επαρκώς τα περισσότερα σενάρια UAT. Οι περιπτώσεις επιχειρηματικής χρήσης αποτελούν στοιχεία για τη δημιουργία των δοκιμαστικών υποθέσεων

Βήμα 4) *Προετοιμασία δεδομένων δοκιμής*

Συνιστάται η χρήση live data για UAT. Τα δεδομένα πρέπει να ανακαλυφθούν για λόγους απορρήτου και ασφάλειας. Ο tester πρέπει να είναι εξοικειωμένος με τη ροή της βάσης δεδομένων.

Βήμα 5) *Εκτέλεση και καταγραφή αποτελεσμάτων*

Εκτέλεση test cases και αναφορά σφαλμάτων αν υπάρχουν. Έπειτα, επανεξέταση σφαλμάτων αφού διορθωθούν. Εργαλεία διαχείρισης δοκιμών μπορούν να χρησιμοποιηθούν για εκτέλεση.

Βήμα 6) *Επιβεβαίωση επιχειρησιακών στόχων που πληρούνται*

Οι αναλυτές ή οι υπεύθυνοι δοκιμών UAT πρέπει να στείλουν ένα μήνυμα αποσύνδεσης μετά τον έλεγχο UAT. Μετά την αποσύνδεση, το προϊόν είναι έτοιμο για παραγωγή. Παραδοτέα για δοκιμή UAT είναι το Σχέδιο δοκιμών, τα σενάρια UAT και test cases, τα αποτελέσματα των δοκιμών και το αρχείο με τα logs.

11.7. Κριτήρια ετοιμότητας

Πριν προχωρήσουμε στην παραγωγή, πρέπει να ισχύουν τα ακόλουθα:

-Δεν βρέθηκαν κρίσιμα σφάλματα

-Η επιχειρηματική διαδικασία λειτουργεί ικανοποιητικά

-Έγινε επιτυχής αποσύνδεση από όλα τα μέρη

11.8. Βέλτιστες Τεχνικές

Τα ακόλουθα σημεία πρέπει να ληφθούν υπόψη για την επιτυχία του UAT:

• Προετοιμασία σχεδίου UAT στις αρχές του κύκλου ζωής του έργου

• Προετοιμασία λίστας ελέγχου πριν από την έναρξη του UAT

• Διεξαγωγή συνεδρίας πριν από το UAT κατά τη διάρκεια του System Testing

• Σαφήνεια στο πεδίο εφαρμογής του UAT

• Δοκιμή επιχειρηματικής ροής end to end

• Εφαρμογή με πραγματικά σενάρια και δεδομένα

• Συμπεριφορά του tester ως άγνωστος χρήστης του συστήματος

• Εκτέλεση δοκιμής ευχρηστίας

11.9. Tools

Υπάρχουν πολλά εργαλεία στην αγορά που χρησιμοποιούνται για το User Acceptance Testing, για παράδειγμα το παρακάτω αναφερθέν εργαλείο:

Fitness Tool: Είναι ένα εργαλείο java που χρησιμοποιείται ως testing engine. Είναι εύκολο να γίνουν δοκιμές και να καταγραφούν αποτελέσματα σε έναν πίνακα. Οι χρήστες του εργαλείου εισάγουν τη μορφοποιημένη είσοδο και οι δοκιμές δημιουργούνται αυτόματα. Στη συνέχεια, οι δοκιμές εκτελούνται και η έξοδος επιστρέφεται στον χρήστη.

12. Non- functional Testing

Σκοπός της εργασίας είναι η μελέτη του Functional Testing. Παρόλα αυτά, θα ήταν χρήσιμη η σύγκριση και η μελέτη των διαφορών μεταξύ Functional και Non-functional testing.

12.1. Τι είναι Non-Functional Testing

Το Non-Functional Testing χαρακτηρίζεται ως ένας τύπος ελέγχου λογισμικού που ως σκοπό έχει τον έλεγχο των non-functional μερών.

Ο σκοπός του είναι η μελέτη της αναγνωσιμότητας του συστήματος ως συνδυασμός non-functional παραμέτρων, οι οποίες παράμετροι δεν έχουν αναφερθεί στο functional testing.

12.2. Παράδειγμα

Ένα παράδειγμα non-functional testing είναι να ελέγξουμε πόσα άτομα μπορούν να συνδεθούν ταυτόχρονα σε ένα σύστημα.

Το non-functional testing είναι εξίσου σημαντικό με το functional testing και συνεισφέρει στην ικανοποίηση του πελάτη στον ίδιο βαθμό.

12.3. Χαρακτηριστικά non-functional testing

Τα αποτελέσματα πρέπει να είναι ακριβή, επομένως δεν υπάρχει η δυνατότητα χαρακτηρισμού του testing ως καλό, καλύτερο, άριστο κλπ.

Οι ακριβείς αριθμοί δεν είναι δυνατό να είναι γνωστοί στην αρχή της διαδικασίας.

Επίσης στο non-functional testing, είναι σημαντικό να δίνεται προτεραιότητα στις απαιτήσεις.

12.4. Παράμετροι:

Non Functional Testing Parameters

Σύντομη Περιγραφή Παραμέτρων

1) Security: Καθορίζει πως προστατεύεται το σύστημα από εσωτερικές και εξωτερικές επιθέσεις. Αυτό είναι γνωστό ως Security Testing.

2) Reliability: Το σύστημα πρέπει να εκτελεί τις λειτουργίες χωρίς λάθη. Αυτό είναι γνωστό ως Reliability Testing.
3) Survivability: Ελέγχει αν το σύστημα συνεχίζει τις λειτουργίες και επιστρέφει στην αρχική του κατάσταση μετά από μήνυμα λάθους. Γνωστό ως Recovery Testing.
4) Availability: Καθορίζει τον βαθμό στον οποίο ο χρήστης μπορεί να βασιστεί στο σύστημα κατά τη διάρκεια της λειτουργίας του. Γνωστό ως Stability Testing.
5) Usability: Η ευχρηστία ή αλλιώς η δυνατότητα γρήγορης εκμάθησης των λειτουργιών από το χρήστη. Γνωστό ως Usability Testing.
6) Scalability: Ο όρος αναφέρεται στον βαθμό στον οποίο κάθε εφαρμογή λογισμικού μπορεί να επεκταθεί και να αποκτήσει μεγαλύτερη κλίμακα. Γνωστό ως Scalability Testing.
7) Interoperability: Έλεγχος του συστήματος με αλληλεπίδραση άλλων συστημάτων. Γνωστό ως Interoperability Testing.
8) Efficiency: Διαχείριση χώρου, ποιότητας και ταχύτητας ανταπόκρισης της εφαρμογής (μας ενδιαφέρει η χρονική διάρκεια).
9) Flexibility: Δυνατότητα της εφαρμογής να λειτουργήσει σε διαφορετικό hardware ή ρυθμίσεις συστήματος.
10) Portability: Δυνατότητα μεταφοράς σε άλλο hardware.
11) Reusability: Δυνατότητα χρήσης του λογισμικού σε άλλες εφαρμογές.

13. Functional vs Non-Functional Testing

Αφού είδαμε μια σύντομη περιγραφή του Non-functional Testing, καθώς και των χαρακτηριστικών του, μπορούμε να εντοπίσουμε τις διαφορές μεταξύ Functional Testing και Non-functional testing.

Μια βασική διαφορά, είναι ότι το functional testing πραγματοποιείται χρησιμοποιώντας specification που παρέχεται από τον πελάτη και επικυρώνει το σύστημα με βάση τις λειτουργικές απαιτήσεις, ενώ το non-functional testing ασχολείται με το Performance, το Reliability, το Scalability και οι παραπάνω λειτουργίες, όπως αυτές αναφέρθηκαν και ορίστηκαν στην προηγούμενη σελίδα.

Επίσης, αξίζει να αναφερθεί ότι απαιτούνται και τα 2 είδη testing και συνεισφέρουν στην ποιότητα του λογισμικού. Όμως, μια ακόμα διαφορά τους είναι η σειρά με την οποία εκτελείται το κάθε testing. Το functional testing εκτελείται πρώτο. Το non-functional ακολουθεί μετά το functional.

Ως προς την υλοποίηση τους, στο functional testing έχουμε τη δυνατότητα να επιλέξουμε αν θα το κάνουμε manually ή αν θα χρησιμοποιήσουμε automation tools, ενώ στο non-functional κρίνονται απαραίτητα. Σε αντίθεση με το functional, το manual testing είναι αρκετά επίπονο στο non-functional testing.

Ως τελευταία διαφορά, το functional testing παίρνει ως είσοδο επιχειρησιακές απαιτήσεις και περιγράφει το ΤΙ κάνει το προϊόν, σε αντίθεση με το non-functional testing που δέχεται παραμέτρους όπως η ταχύτητα και η κλιμάκωση και περιγράφει πόσο καλά λειτουργεί το παραγόμενο προϊόν λογισμικού.

14.Βιβλιογραφία

Testing basics https://en.wikipedia.org/wiki/Software_testing
https://usersnap.com/blog/software-testing-basics/

Functional Testing https://www.guru99.com/functional-testing.html
https://www.simform.com/functional-testing/

Automation Test https://en.wikipedia.org/wiki/Test_automation
https://www.guru99.com/automated-testing-tools.html
https://www.ranorex.com/black-box-testing-tools/

Non Functional Testing https://en.wikipedia.org/wiki/Non-functional_testing
https://www.guru99.com/non-functional-testing.html

Practical Approach https://www.simform.com/functional-testing/#bestpractices

Unit Testing http://softwaretestingfundamentals.com/unit-testing/
https://www.softwaretestinghelp.com/unit-testing-tools/

Smoke Testing https://www.guru99.com/smoke-testing.html

Sanity Testing https://www.geeksforgeeks.org/sanity-testing-software-testing/

Regression Testing https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/

Integration testing http://softwaretestingfundamentals.com/integration-testing/

Black Box Testing http://softwaretestingfundamentals.com/black-box-testing/

User Acceptance Testing https://www.guru99.com/user-acceptance-testing.html