Managementul Calității

1 Introducere

Asigurarea calității (QA) este necesară pentru ca întregul Ciclu de Viață de Dezvoltare a Software (SDLC) să fie unul de calitate înaltă și este o etapă integrală și critică a fiecărui proiect de dezvoltare software-ului. Acest proces nu doar anticipează cerințele, dar și asigură un software sigur și stabil garantând urmarea tuturor standardelor și procedurilor din cadrul companiei.

Procesul de asigurare a calității constă în verificarea tuturor soluțiilor software dezvoltate și asigurarea că ele întrunesc specificațiile și așteptările Beneficiarului. Asigurarea calității este un aspect critic începând cu etapa de inițiere și finisând cu etapa de lansare a software-ului.

2 Abordare

Strategiile de asigurare a calității sunt bazate pe următoarele principii:

- Activitățile de asigurare a calității nu sunt doar niște sarcini executate la finele etapei de dezvoltare (cum ar fi testarea), acestea au loc pe parcursul întregului ciclu de viață de dezvoltare a software (SDLC).
- Calitate bună este un rezultat a lucrului de echipă, calitatea fiind responsabilitatea fiecărui membru și este realizată prin munca zilnică a fiecărui membru.
- ”Prevenirea apariției erorilor” în loc de ”Investigare și testare”.

Prin urmare, sistemul de asigurare a calității intern reprezintă următoarele:

Standarde – standardizare (toți programatorii se vor supune standardelor companiei - Standarde de programare, standarde de programare a bazelor de date, standard UI, Definition of Done, etc.)

Metodologii – Agile Software Development (sugerăm clienților noștri să construiască echipe transversale funcționale, care vor adopta o dezvoltare incrementală cu cicluri iterative scurte).

Flux de lucru – producerea calității (programatorii urmează următorul flux pentru a se asigura că sarcina este cu siguranță realizată:

- estimarea costului de dezvoltare;
- crearea listelor auto-testare (self-test);
- testarea codului de unitate (code unit testing);
- programare;
- executarea testării unitare (unit testing);

- code refactoring;
- code peer review;
- executarea auto-testării;
- merge and submit code.

Cele mai bune practici (practici adoptate pe parcurs pentru îmbunătățirea calității):

- Cele mai bune practici în programare (Unit testing, TDD, Code Review, Refactor, Daily Build/Continuous Integration);
- Cele mai bune practici în testare (Self-testing, Cross-testing, Integration Testing, Regression Testing, Test Flow);
- Cele mai bune practici la nivel funcțional (Prototype, Function list, Scrum Review).

Supportul companiei (managementul companiei, de asemenea, va monitoriza și acorda suport echipelor pentru asigurarea acestora de a anticipa satisfacția Beneficiarului)

- Instruirea Echipelor;
- Inspecția Echipelor;
- Îmbunătățirea Proceselor.

Din strategiile de mai sus, am construit mecanisme sistematice pentru asigurarea calității produselor software.

Procesul de asigurare a calității nu este izolat, ci interacționează cu alte aspecte ale proiectului precum Bugetul, Scopul și Planul de Implementare. Prin urmare, întru asigurarea calității și asigurarea interacțiunii între calitate și alte constrângeri s-au adaptat cele mai bune practici pentru de a Controla Graficul de Implementare, Gestiona Scopul și Controla Costul.

3 Standarde – calitatea de bază

Standardele sunt componentele de bază ale oricărui sistem de asigurare a calității. Indrivo a elaborat mai multe standarde pentru a fi urmate de către membrii echipei, inclusiv, standardele de programare în PHP, .Net, etc., standard de programare a bazelor de date, standard UI, standarde de testare, etc. Toate împreună, aceste standarde stabilesc o bază pentru asigurarea calității.

4 Metodologia – Agile Software Development

Este demonstrat că Dezvoltarea Software după metodologia Agile îmbunătățește semnificativ calitatea software-ului. Trei componente de bază au efect sporit asupra calității:

4.1 Lucru de echipă „Cross-funcțional”

O echipă multifuncțională este compusă din persoane din toate departamentele companiei cum ar fi Departamentul QA, Departamentul de Dezvoltare, Management, etc.

Membrii echipei lucrează în aceeași încăpere, în contact direct, colaborează foarte strâns unii cu alții, prin urmare echipa este capabilă să livreze de sine stătător componente și nu necesită o comunicare multi-departamentală.

În rezultat, este exclusă lipsa de comunicare, ciclul de feedback este scurtat și eficiența în lucru este ridicată. Astfel, aspectele legate de calitate pot fi abordate mai devreme și corectate la un cost mai mic.

Roluri și responsabilități:

Departament	Rol	Responsabilități în activitățile de QA
Departament QA	QA/Tester	Urmărește proiectele în derulare, revizuii și evaluează artefacte, asigură progresul proiectului și calitatea artefactului conform standardelor relevante.
Departamentul de Dezvoltare	Manager de proiect	Controlează managementul general al proiectelor, implementează Standardele fiecărei etape și coordonează între membrii echipei QA și membrii echipei de proiect. Înregistrează și integrează informația despre proiect ți feedback-ul clienților pentru îmbunătățiri ulterioare.

4.2 Dezvoltare Incrementală

Dezvoltarea incrementală este o strategie de planificare și de etapizare, în care sistemele sunt dezvoltate pe părți și integrate în ansamblu atunci când sunt finalizate. În comparație cu modelul tradițional Waterfall (”de cascadă”), produsul final crește incremental în cicluri iterative scurte. Astfel, se evită multitudinea modificărilor într-o singură dată și astfel se reduc șansele de apariție a mai multor erori.

Compania Indrivo utilizează metoda dezvoltării incrementale în cadrul implementării unui proiect și separă ciclul de viață a dezvoltării unui software în mai multe faze mici și etape de reper.

4.3 Livrare frecventă

Prin Dezvoltarea Incrementală software-ul este dezvoltat în mai multe cicluri iterative scurte, la sfârșitul fiecărui ciclu, livrabilele sunt transmise către departamentul de testare și apoi către Beneficiar.

Prin urmare, multe defecte pot fi găsite și eliminate mai devreme, iar problemele similare vor fi, de asemenea, evitate în următoarele faze de dezvoltare.

Se recomandă ca echipa să livreze software-ul Beneficiarului la fiecare două săptămâni. În funcție de complexitatea fiecărui proiect, cea mai lungă iterație nu va depăși două luni, iar cea mai scurtă - o săptămână.

5 Cele mai bune practici – îmbunătățirea calității în orice aspect

Pentru asigurarea și îmbunătățirea în mod optim calitatea fiecărui proiect, dezvoltatorii cât și managementul companiei implementează cele mai bune practici în fiecare aspect al proiectelor noastre. Aceste practici se împart în trei categorii:

- Cele mai bune practici la Nivel Funcțional;
- Cele mai bune practici pentru Programare;
- Cele mai bune practici pentru Testare.

5.1 Cele mai bune practici la Nivel Funcțional

Aceste practici sunt executate în mare măsură a nivel funcțional și nu pot fi sortate în activități de dezvoltare sau de testare, dar ajută la asigurarea calității.

Prototiparea

Scopul principal al unui prototip este de a comunica și confirma cerințele față de Beneficiar, asigurând o calitate a produsului care va fi dezvoltat. Prototiparea rezolvă și identifică multe erori de funcționalitate, probleme de neconformitate și probleme de experiență ale utilizatorilor.

Lista Funcționalităților

Lista funcționalităților este o listă organizată care include toate caracteristicile software-ului și care este revizuită atunci când se schimbă cerințele. Deoarece lista funcționalităților poate fi utilizată în practicarea testelor de auto-testare și de regresie, aceasta va reduce la minim șansele de a pierde ceva și, de asemenea, va ajuta echipa de a descoperi probleme inconsecvente.

Scrum review

Scrum Review este o practică provenită din metodologia Scrum, care se practică la sfârșitul fiecărui Scrum Sprint (fiecare iterație). Echipa demonstrează produsul software tuturor părților interesate, odată ce au realizat un livrabil. Această activitate obligă echipa să livreze ceea ce funcționează cu adevărat. De asemenea, revizuirea iterației oferă altor părți interesate șansa de a contribui.

5.2 Cele mai bune practici pentru Programare

Aceste practici se focusează în prim plan pe îmbunătățirea calității codului-sursă, ceea ce duce la obținerea unor produse mai sustenabile. Rezultatul presupune mai puține erori și cost mai mic pentru perioada de mentenanță.

Unit Testing

Unit Testing constă în scrierea unor fragmente de cod pentru a valida corectitudinea funcționalităților (cum ar fi funcții și clase). Acestea formează o rețea de siguranță pentru prevenirea modificărilor neintenționate ale codului. Astfel, odată ce apare o eroare, testele de unitate pot ajuta programatorul să găsească problema și să o repare foarte rapid, având mai puțin efort de testare. Testele de unitate se efectuează cu acordul Beneficiarului.

Dezvoltare pe baza de test – TDD (Test Driven Development)

Test Driven Development cere dezvoltatorilor să scrie teste înainte de implementarea funcționalităților. Astfel, dezvoltatorii vor fi forțați să se gândească din punctul de vedere al utilizatorilor și vor face un design mai circumspect până a fi transpus în cod. Pe măsură ce problema de proiectare este evitată de TDD, complexitatea codurilor va fi păstrată la un nivel minim.

Revizuirea codului (Code Review)

Revizuirea codului este o practică demonstrată, care îmbunătățește considerabil calitatea codului. Prin revizuirea codului, multe problem potențiale vor fi identificate, deviațiile în standarde vor fi corectate, și cel mai important, dezvoltatorii vor face schimb de experiență, iar toate împreună vor contribui la calitatea integrală a software-ului.

Refactoring

Refactoring este o modalitate disciplinată de a curăța codul și de a îmbunătăți designul. Refactoring-ul păstrează codurile ordonate și minimizează introducerea erorilor. Refactoring-ul este parte a fluxului de dezvoltare și toate proiectele necesită un refactoring a codului atunci când sunt efectuate modificări.

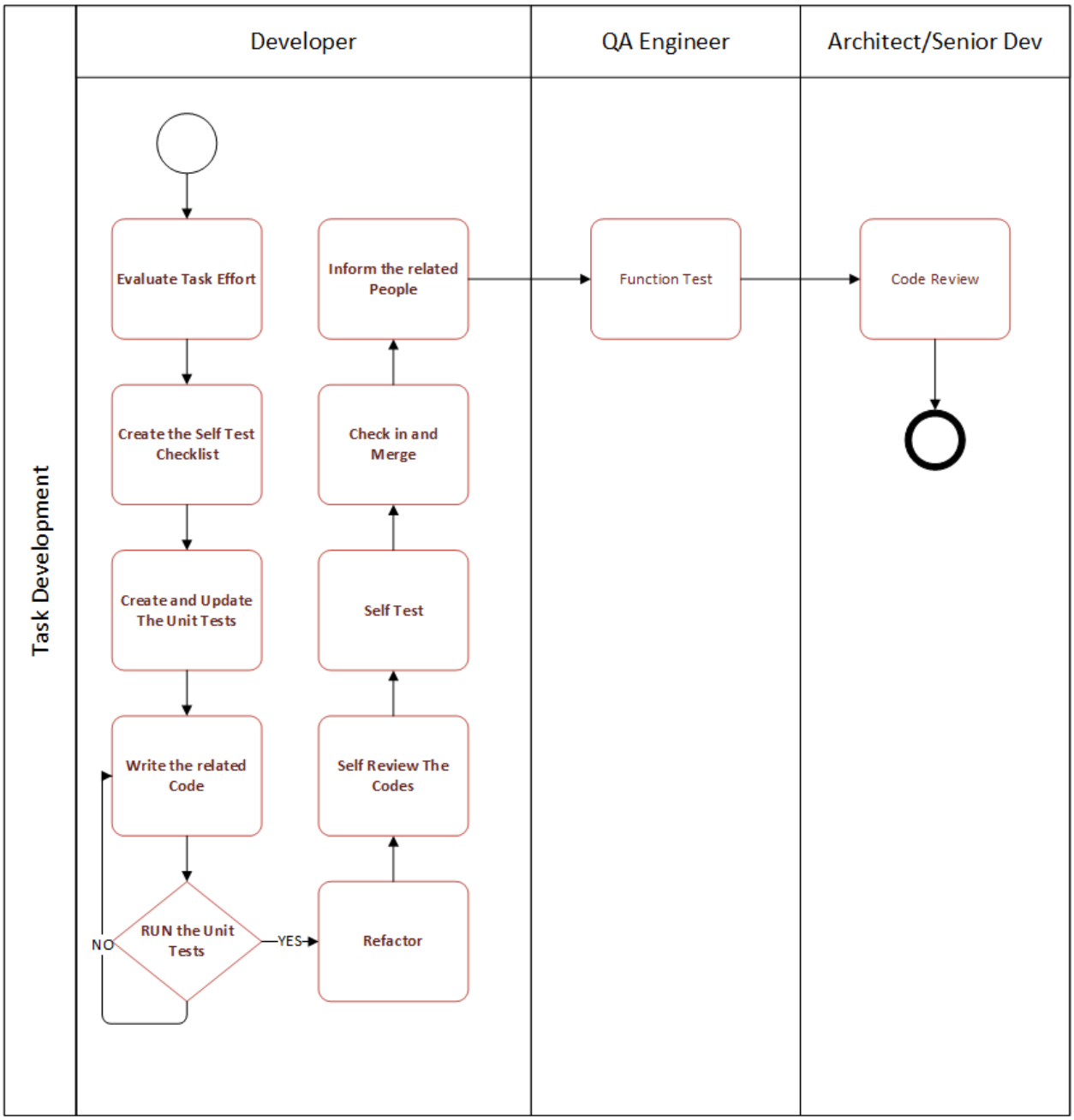
Compilarea zilnică (Daily Build)

Compilarea zilnică este o practică de a compila software-ului în fiecare zi. Astfel este minimizat riscul integrării și sunt evitate multe erori care ar apărea în integrarea pachetelor de software. Compilarea zilnică asigură faptul că dezvoltatorii se sincronizează, astfel încât orice defecte cauzate de neînțelegerea și lipsa de comunicare să poată fi rezolvate ușor și rapid.

Fluxul de lucru – Dezvoltarea Calității

Una dintre componentele esențiale ale procesului de asigurare a calității este executarea fluxului de lucru standardizat, în care sunt integrate strategii și metode de calitate.

Fluxul următor asigură că pașii necesari pentru implementare nu vor fie omiși de către dezvoltatori, iar prin urmare, problemele potențiale vor fi evitate. Multe verificări vor fi executate în primă instanță, ceea ce va asigura în continuare descoperirea și eliminarea erorilor în procesul de lucru.



5.3 Cele mai bune practici în Testare

Fiecare proiect trebuie să conțină un plan amplu de testare care să cuprindă toate funcționalitățile aplicației şi să asigure funcționarea corectă a întregului produs. Strategiile de testare se concentrează pe funcționalitatea şi utilizabilitatea produsului.

Există câteva reguli care sunt considerate ca obiective ale testării:

- testarea este un proces de execuție a unui program cu intenția de a găsi o eroare;
- un caz de test bun este unul care are probabilitate mare de a găsi o eroare nedescoperită încă;
- un test terminat cu succes este un test în care se descoperă o eroare nedescoperită încă.

Metodele de verificare a corectitudinii unui produs se împart în două mari grupuri:

- metode statice de testare: constau în analiza unui program înainte de a fi lansat în execuție, independent de datele de intrare;
- metode dinamice de testare: constau în execuția programului; această metodă este cunoscută şi sub numele de testare structurală.

Dintre cele mai familiare metode statice se amintesc: testarea specificației și examinarea codului.

La examinarea codului este inclusă și compilarea. Un compilator modern verifică tot felul de proprietăți ale programului, și refuză programele care nu respectă criteriile de corectitudine. Alteori compilatorul dă avertismente asupra unor construcții care generează probleme la execuție, cum ar fi de pildă variabile neinițializate.

Testarea sistemelor informatice complexe presupune testarea pe componente şi părţi ale componentelor (funcții sau clase), după un plan de test ce cuprinde toate cazurile de test. În sistemele complexe componentele sunt strâns legate între ele iar funcționalitatea unui modul este dependentă de alte module şi atunci testarea se complică.

În cazul testării unui modul ce este dependent de informațiile din alte module se practică scrierea diferitelor date de test în baza de date pentru a testa toate cazurile posibile şi rezultatul acestor teste se scrie în fișiere de log sau în baza de date.

Pentru un grup de module ce formează un subsistem se va descompune acest subsistem în funcții si module, pentru a se putea realiza testarea prezentă. Se testează mai întâi funcțiile apoi modulele şi se agregă modulele și se obține testarea produsului finit.

Practicile de testare se focusează pe activități de testare. Implementând teste eficiente și efective, calitatea produselor este garantată. Personalul necesar pentru testare trebuie să fie specializat să cunoască tehnicile de analiză, proiectare și programare şi să înțeleagă problema pe care aplicația dorește să o rezolve. Procesul de testare se recomandă a fi independent de producător şi de utilizator pentru a asigura rigurozitatea rezultatelor și a interpretării corecte a acestora.

Etapele testării se derulează astfel:

1. se formează echipa de test în funcție de scopul testului şi de aplicația de testat, cu cât sistemul software este mai complex cu atât crește numărul testărilor și specialiștilor;
2. echipa va fi împărțită pe tipuri de funcții pe care trebuie să le testeze persoanele grupului de test;
3. se construiesc exemplele de test şi se utilizează şi exemplele de test furnizate în specificație;
4. se face un plan de test cuprinzând durata şi numărul de iterații.
5. se alege metoda se testare adecvata în raport cu produsul;

6. se definesc documentele/rapoartele pe care trebuie să le elaboreze membri echipe de test, cât şi documentele care se realizează la nivelul echipei;
7. se colectează erorile, le stabilesc frecvența şi se cuantifică efectele pe care acestea le generează la utilizatori;
8. reproduc condițiile de producere a erorilor;
9. în cadrul programării orientate obiect testarea trebuie să cuprindă în mod special testarea nivelurilor de încapsulare, moștenire şi polimorfism, pentru fiecare existând tehnici de testare adecvate.

Cazurile de test trebuie să țină seama că:

- fiecare caz de test trebuie să fie identificat unic şi asociat explicit cu clasele care vor fi testate;
- să se spună din start scopul testului;
- trebui realizată o lista de pași prin care trebui să treacă testarea ce trebuie să cuprindă:
  - listă cu stările prin care trebuie să treacă obiectul testat;
  - listă de mesaje şi operații care trebuie făcute pentru ca testul să fie consistent;
  - listă de excepții prin care un obiect trebuie testat;
  - listă de condiții externe (exemplu: modificarea unor variabile de environment);
  - informații suplimentare necesare pentru a înțelege sau realiza testul.

Pe baza erorilor şi documentelor colectare în urma testării, șeful de proiect va stabili timpii şi prioritățile în rezolvarea defectelor.

5.3.1 Testarea funcțională

Unit testing

Termenul de 'testare unitara' se refera la testarea individuala a unor unități separate dintr-un sistem software. În timpul proiectării şi codificării se comit erori care sunt grupate în următoarele categorii:

- erori legate de alegerea şi descrierea algoritmului: algoritm incorect, sau corect dar inadecvat problemei; algoritm mai puțin performant ca precizie sau timp necesar rezolvării problemei; omiterea, interpretarea greșită sau incompletă a unor părți ale algoritmului; validarea incorectă şi/sau incompletă a datelor de intrare; inversarea răspunsurilor la un bloc de decizie;
- erori în definirea şi utilizarea datelor ce provin din variabile neinițializate, formate improprii de citire, contoare de capacitate insuficientă, neverificarea datelor de intrare, aliniere/redefinire incorectă a câmpurilor, utilizarea unor cuvinte cheie ca variabile, variabile ilegale (formate prin concatenare sau despărțite între două linii de program);

- erori de calcule care au ca surse: expresii complicate cu posibilități necontrolate de eroare; conversii implicite de tip (cu eroare de conversie, rotunjire, trunchiere); ne interceptarea cazurilor de depășire/subdepăşire a intervalului definit;
- erori produse în tehnica de programare cum sunt variabile şi structuri de date globale, acces necontrolat la zone de memorie partajate, interfețe program - subprogram nerespectate, pasarea constantelor ca parametri transmiși prin adresă, pasarea parametrilor de intrare/ieșire prin valoare, automodificarea programului în timpul execuției, utilizarea necontrolată a mai multor limbaje cu convenții de apel diferite;
- erori produse din neatenție caz în care logica de control e defectuoasă, salt în afara limitelor programului, condiții logice compuse sau incorect negate, neprelucrarea primei sau ultimei înregistrări, neluarea în considerare a posibilității de existență a fișierelor vide, neprelucrarea erorilor de intrare/ieșire, depășirea capacității stivei, adresare incorectă, necontrolarea indecșilor;
- erori în contextul execuției datorate memoriei dinamice insuficiente sau nealocată, periferice neoperaționale, comunicare defectuoasă cu sistemul de operare.

Cea mai mare parte a erorilor enumerate sunt depistate în faza de compilare a programului şi sunt extrase în fișierul de ieșire, într-o formă specifică fiecărui compilator. Tot ca erori de codificare sunt considerate şi cele detectate în faza de editare a legăturilor. În timpul execuției programelor apar erori de genul:

- erori de echipament, care sunt legate de contextul în care se execută un program şi care se împart în: erori în datele de intrare, erori ce decurg din neglijarea specificului unui limbaj sau compilator (aritmetica numerelor în calculator, modul de implementare a tipurilor şi structurilor de date pe un limbaj dat)
- erori de încărcare a programelor şi de apelare incorectă a diferitelor periferice.

Furnizorul va instala o versiune a sistemului în mediul de dezvoltare MCloud. Acoperirea testelor de tip Unit Testing va fi de minim 60%

Auto-testare (Self-testing)

Auto-testarea se efectuează în conformitate cu o anumită listă de verificare înainte de efectuarea modificării. Scopul este de a asigura că rezultatele dezvoltatorilor sunt acceptabile. Cu o auto-testare eficientă, erorile pot fi descoperite și corectate la o etapă prematură.

Testare încrucișată (Cross-testing)

Metoda de testare încrucișată presupune ca ”A” testează produsul lui ”B” în timp ce ”B” testează produsul lui ”A”. Pentru a elimina ”zonele moarte” ale fiecărui dezvoltator/tester, testarea încrucișată va aduce valoare adăugată calității produsului software.

Testarea pentru acceptanță

Se efectuează cu scopul de a valida funcțional produsul din perspectiva utilizatorului final. Obiectivul recomandat este demonstrarea modului în care produsul se va integra în mediul de lucru real al Beneficiarului.

Ca alternativă, se urmărește familiarizarea utilizatorilor finali cu modul de operare a aplicației, caz în care are loc o trecere în revistă a funcțiilor aplicației. Se apreciază că acest tip de testare, care implică şi participarea viitorilor Beneficiari, este pentru dezvoltator o importantă sursă de informații privind contextul în care va fi utilizat sistemul.

Presupune o abordare graduală a modulelor specifice. Odată modulele asamblate şi testate în integration test şi după ce erorile de interfață au fost descoperite şi rezolvate începe seria finală de test şi anume *testul de acceptanță.* Validarea este terminată când aplicația software funcționează într-o manieră rezonabilă acceptată de Beneficiar. Acceptările rezonabile sunt definite în documentul ce cuprinde specificația cerințelor şi care descrie toate atributele cerute de Beneficiar.

Specificațiile conțin o secțiune numită criteriile de validare care reprezintă baza testului de validitate. Pentru realizarea acestui test se pregătește un plan de test împreună cu procedurile prin care se face testul. Planul şi procedurile sunt proiectate astfel încât să se testeze cerințele, performanțele, documentația, compatibilitatea, întreținerea cât şi procedurile de restaurare. În urma testului de validare se creează o documentație cu specificare pentru fiecare element din planul de test ca funcția/caracteristică de performanță:

- este conform cu specificația şi este acceptată;
- nu este conform cu specificația şi este atașată o lista de erori sau abateri de la specificație.

Pentru eliminarea erorilor descoperite în această etapă a proiectului se corectează înainte de predare dar de cele mai multe ori trebuie să se negocieze cu Beneficiarul pentru stabilirea metodelor pentru rezolvarea deficiențelor găsite în această etapă.

Un element important al procesului de validare este revederea configurației - configuration review. Se verifică dacă toate elementele necesare pentru configurare au fost dezvoltate şi funcționează la parametri stabiliți. Este imposibil de imaginat cum Beneficiarul va folosi în realitate produsul realizat de o companie de software, chiar dacă acesta cuprinde un manual de utilizare.

Cei mai mulți producători de software utilizează procese numite Alpha Test şi Beta Test pentru a descoperi erori pe care numai utilizatorii finali le descoperă.

Testul alpha se realizează de către clienți selectați, este condus de către dezvoltatorii de software şi este de obicei într-un mediu controlat. Aplicația este utilizat având în spate dezvoltatorul pentru a înregistra erorile şi problemele apărute.

Testul beta este făcut de unul sau mai mulți clienți finali fără nici un control din partea dezolatorului. Acesta este un test într-un mediu necontrolat (ambient real) în care Beneficiarul

înregistrează toate problemele reale sau imaginare şi vor fi raportate la intervale regulate către dezvoltator.

5.3.2 Testare non-funcțională

Este specific sistemelor complexe care trebuie să fie operaționale. Într-un sistem software complex este obligatoriu să se facă şi testul de sistem.

Testul de sistem este compus dintr-o serie de teste al căror obiectiv este să testeze evoluția produsului software în condiții date de sistemul hardware. Avem următoarele teste care trebuie făcute în testarea sistemului:

- test de recuperare - recovery testing;
- test de securitate - security testing;
- test de stres - stress testing;
- test de performanță - performance testing.

Testarea de integrare

Testarea de integrare este o tehnică sistematică de construire a structurii programului prin gruparea componentelor în paralel cu testarea aspectelor legate de interfața dintre componente.

Pentru acest proiect va fi utilizată metoda de testare ascendentă/bottom-up testing care presupune testarea mai întâi a modulelor de pe cel mai de jos nivel al ierarhiei programului, apoi se continuă în sus. Componentele de pe nivelul de sus, care de obicei sunt critice, sunt testate ultimele. În timp sunt descoperite erori care pot influența multe module care au fost deja testate. După corecția erorilor este necesar ca toate modulele de pe nivelurile de jos să fie testate regresiv.

Testarea de integrare se va realiza într-o manieră incrementală. Furnizorul va instala componentele sistemului în Mediul de Integrare, urmând ca Beneficiarul să testeze aleatoriu 5-10% din cerințele tehnice obligatorii. Livrabilul va fi acceptat dacă în urma testării vor fi constatate mai puțin de o neconformitate critică și 3 neconformități majore.

Testul de recuperare

Este un test de sistem prin care se forțează sistemul să dea o varietate de erori pentru a putea verifica dacă restaurarea se realizează corect. Se verifică: restaurarea (automată sau manuală), reinițializarea, mecanismele de verificare a restaurării şi respectiv timpul necesar pentru restaurare.

Testul de securitate

Presupune verificarea mecanismelor de protecție implementate în sistem, de fapt protecția la intrările neautorizate în sistem. Rolul unui proiect de securitate al unui sistem este

să facă astfel încât costul de spargerea al sistemului să fie mai mare decât beneficiile pe care le obține prin spargerea sistemului.

Furnizorul va efectua teste de securitate conform standardului OWASP Top 10 Vulnerabilități.

Testul de stres

Presupune execuția sistemului într-o manieră anormală. Adică se testează confruntarea software cu situați anormale (multiple tranzacții, memorie insuficientă, spațiu liber mic pe disk, blocarea perifericelor cu care lucrează aplicația, etc.)

Furnizorul va efectua un test privind comportamentul sistemului la un volum mare de cereri (stress testing).

Testul de performanță

Este proiectat să testeze în run-time performanțele sistemului. Acest test se face atât la nivelul modulelor cât şi la nivelul global al întregii aplicații, dar însă pentru verificarea cerințelor de performanță această testare se face după ce integrarea este completă. Testarea de performanță implică atât elemente software cât şi elemente hardware.

Furnizorul va efectua un test privind performanța sistemului (load testing).

Testarea regresivă

Reprezintă o treaptă deosebit de importantă pentru echipele care doresc să dezvolte procese accelerate. Ca modalitate de lucru, prevede repetarea testării cu date de test şi în condiții identice, pentru fiecare nouă versiune internă a unei componente software. Prin compararea rezultatelor testării şi identificarea diferențelor se depistează erorile nou apărute; acest lucru este deosebit de util pentru maniera actuală de dezvoltare a aplicațiilor *RAD - Rapid Application Development*, care implică utilizarea instrumentelor vizuale de programare şi se caracterizează prin apariția unui număr mare de modificări într-un interval scurt de timp.

Pentru executarea cu succes a testării prin regresie, toate cazurile de testare sunt executate și acceptate, fiecare modul ale software-ului este testat pe deplin. Prin această metodă, pot fi găsite cele mai multe probleme (peste 95%) ascunse în software, reducând astfel la minim șansele ca o careva eroare să nu fie întâmpinată de utilizatori.

Testarea de regresie va fi implementată atunci când va fi lansată o versiune. În dependență de mărimea proiectului, testarea și corectarea erorilor ar putea dura două-trei săptămâni.

Testarea prin regresie reprezintă retestarea unui sistem software pentru a confirma că modificările efectuate în câteva părți ale codului nu afectează funcționalitățile existente ale sistemului. Trebuie verificat dacă codul funcționează așa cum a fost înainte de introducerea noilor schimbări.

5.3.3 Flux de testare

Un flux de testare adecvat asigură eficiența și eficacitatea activităților de testare, ceea ce la rândul său îmbunătățește calitatea software-ului. Următorul flux este necesar pentru asigurarea calității:

#	Flux	Descriere
1	Determinarea cerințelor de calitate	Prima etapă este de a identifica cerințele de calitate. QA/Testerii sunt responsabili de analiza cerințelor Beneficiarului și adresarea întrebărilor aferente. Aceștia de asemenea vor confirma înțelegerea cerințelor (cum ar fi cerințe de performanță, cerințe de securitate, etc.) până a întreprinde următorii pași.
2	Crearea Strategiilor și Planurilor de Testare	Odată ce cerințele calității sunt clare, vor fi stabilite strategiile de testare, ce tipuri de teste necesită a fi implementate în proiect, cât de frecvent fiecare tip de testare va fi executat, când și cum va fi măsuraat rezultatul, etc. Apoi, va fi elaborat planul de testare pentru a urma activitățile de testare.
3	Crearea cazurilor de testare	Pentru fiecare funcționalitate (user story sau specificație) a software-ului, QA/Testerii proiectează cazurile de testare, iar aceste cazuri vor indica modul în care fiecare funcționalitate va fi testată și care sunt rezultatele așteptate. Aceste cazuri vor servi în executarea etapelor de testare și testarea de regresie. De obicei, pe parcursul proiectării cazurilor de testare, testerii identifică potențialele erori.
4	Executarea testării	La această etapă, se execută planul de testare care implică mai multe activități cum ar fi pregătirea mediilor necesare, pregătirea instrumentelor și datelor de testare, executarea cazurilor de testare, etc.
5	Raportarea erorilor	În cazul în care testerii identifică erori, aceștia îi loghează în sistemul de management al erorilor (Easy Redmine). Prin urmare, dezvoltatorii le reproduc și le corectează.
6	Verificarea erorilor	Când erorile sunt înlăturate, statutul lor va fi schimbat în "rezolvat" și vor fi transmise înapoi către testeri ca aceștia să le verifice pentru a asigura că aceste erori nu mai persistă.
7	Analiza erorilor	Pentru a identifica statutul calității, se analizează statutul erorilor. Această activitate generează

		informații valoroase, astfel încât se decide dacă planul inițial ar trebui ajustat. Analiza erorilor are loc în timpul și ca urmare a etapei de dezvoltare.
8	Perfecționarea procesului de testare	La sfârșitul fiecărei iterații sau după lansarea unei versiuni majore, testerii analizează dacă procesele și activitățile de testare actuale sunt suficient de eficiente și la necesitate ele sunt îmbunătățite. Procesele și activitățile îmbunătățite vor fi aplicate în faza următoare.

Procesul de testare este asistat de instrumente specifice, care diminuează aspectele de rutină. Se apreciază că utilizarea instrumentelor de testare aduce beneficii comparativ cu efectuarea manuală a testelor, deoarece:

- testarea manuală, chiar în cazul unei planificări riguroase, prezintă riscul neidentificării erorilor din neatenție sau din cauza nerespectării riguroase a cazurilor de test prevăzute;
- testarea manuală solicită un consum intens de resurse umane, care sunt costisitoare şi nu întotdeauna disponibile;
- testarea manuală este înceată comparativ cu testarea automatizată; adesea apare problema dezvoltării unor noi versiuni interne ale componentelor înainte de testarea completă a versiunilor precedente.

Dintre categoriile de instrumente pentru asistarea testării enumerăm:

- instrumente de capturare/redare înregistrează o sesiune de testare într-un fișier script, permițând repetarea acesteia şi sunt efectuate teste multiple în manieră automată cu efectuarea de comparații asupra rezultatelor, aceste instrumente sunt eficiente în testarea regresivă;
- instrumente de execuție automată a testelor asemănătoare cu cele de mai sus, dar cazurile de test sunt specificate de utilizator în fișiere script;
- analizor de acoperire evaluează gradul în care structura codului testat a fost acoperită prin cazurile de test, astfel de instrumente sunt utile pentru identificarea porțiunilor de cod netestate;
- generator de cazuri de test este un instrument care, pe baza unor informații precum cerințe, modele ale datelor, modele obiectuale; generează cazuri de test semnificative, avantajul este eliminarea redundanței în testare, prin determinarea cazurilor de test care asigură acoperirea cât mai mare a codului; această activitate, executată manual, este dificilă;
- generator de date de test este un instrument care folosește la popularea fișierelor şi bazelor de date în vederea testării, popularea se face în general cu date aliatoare, dar

unele instrumente prevăd şi posibilitatea specificării unor condiții; instrumentele sunt utilizate în general pentru popularea cu volume mari de date, necesare testărilor operaționale şi la capacitate maximă;

- analizor logic / de complexitate servește la cuantificarea complexității unor porțiuni de cod; multe astfel de instrumente oferă şi reprezentări grafice ale căilor posibile în structura codului; sunt utile pentru determinarea cazurilor de test necesare pentru atingerea anumitor puncte din cod din rutine complexe.
- instrumente de trasare a erorilor permit gestiunea informațiilor privitoare la erorile detectate şi stadiul corectării lor şi centralizarea acestor informații pentru urmărirea tendințelor acestor defecte; pe baza acestor tendințe se efectuează îmbunătățiri în procesele de dezvoltare şi/sau mentenanță ale organizației;
- instrumente de gestionare a testării au rolul de a asista planificarea şi organizarea elementelor implicate în testare precum fișiere script, cazuri de testare, rezultate;

6 Suportul companiei

Managementul de asemenea joacă un rol foarte important în asigurarea calității. Suportul companiei este foarte important pentru a asigurarea calității.

6.1 Instruire

Instruirea efectuată de companie asigura o mai bună înțelegere a politicilor, standardelor, practicilor de asigurare a calității de către angajați. Instruirea va elimina, de asemenea, unele obstacole întâmpinate de echipă la adoptarea celor mai bune practici. Prin schimbul de experiență între diferite echipe, instruirea va ajuta alte echipe să evite greșelile similare. În plus, formarea tehnică va crește abilitățile programatorilor, ceea ce va duce la mai puține erori și la sporirea calității.

6.2 Îmbunătățirea procesului

Îmbunătățirea procesului este un alt suport important pentru asigurarea calității. Compania îmbunătățește procesele, standardele și practicile încontinuu. Prin colectarea feedback-ului de la echipe, rezumând experiența și standardele de calitate, are loc îmbunătățirea continuă a metodologiei de asigurare a calității existente.