Universitatea Politehnica Bucuresti

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Testarea produselor SW, verificare si validare de produse SW

Gheorghe Corina 443 A

Petre Radu 443 A

Paraschiv Radu 443 A

Negrei Alexandru 443 A

Teodorascu Paula 443 A

Miu Madalina 441 A

Herea Cristian 441 A

Bucuresti 2013

Cuprins

5.2 Testarea componentelor
5.3 Testarea integritatii
5.4 Testarea sistemului
5.5 Testarea de acceptare
5.6 Testarea functional
5.7 Testarea nefunctionala
5.8 Re-testarea

Negrei Alexandru 443A:

6. Validarea produselor software
    6.1 Notiuni generale
    6.2 Tipuri de validari
    6.3 Aspectele Validarii
    6.4 Principiile validarii
    6.5 Viziuni filozofice asupra validarii
    6.6 Concluzii

Teodorascu Paula 443A:

7. Analiza statistica si analiza fluxului de date
    7.1 Introducere In Analiza Statistica
        7.1.1 Scopul Analizei Statistice
        7.1.2 Descriere - Metode - Termeni cheie
    7.2 Analiza Fluxului de date
        7.2.1 Introducere
        7.2.2 Principiu
        7.2.3 Analiza Inainte - Analiza Inapoi

8. Bibliografie

1. Implementarea unui produs software

Gheorghe Corina 443 A

Atunci cand ne gandim la un produs software, ne gandim in primul rand la ce avem de facut pentru a-l implementa. Astfel, putem spune ca anumite etape trebuie parcurse:

1. Analiza preliminara
2. Dezvoltare si proiectare
3. Instalare si evaluare
4. Mentenanta si support
5. Verificarea produselor software

Sursa imagine: http://depositphotos.com/11799523/stock-photo-The-five-steps-of-Lean-implementation.html

1.1. Analiza preliminara

Analiza preliminara este etapa in care sunt identificate nevoile si asteptarile pe care o persoana le are cu privire la produsul software. Trebuie stabilit ce implica realizarea unui asemenea produs, cum va putea fi utilizat si de ce resurse va fi nevoie. Dupa care se realizeaza un plan de implementare bazat pe diagnosticul stabilit.

1.2. Dezvoltare si proiectare

E un lucru stiut ca, desi unele produse software pleaca de la premise déjà existente, acestea trebuie adaptate conform unor cerinte specifice.

In aceasta etapa se va crea un document cu specificatiile functionale ale produsului care va contine functionalitatea propriu-zisa, importanta, impartirea pe categorii, interfata, utilizabilitatea.

„Functionalitatea propriu-zisa se refera la operativitate (web, mobil, BackOffice, FrontOffice, interfata ERP, End User), rapoarte precum si la configurare.”

In aceasta etapa se vor defasura activitati de programare si testare pe baza documentului cu specificatii.

Activitatile de programare presupun cunoasterea unor limbaje de programare web sau de nivel inalt in functie de produsul software. Inainte de desfasurarea efectiva a acestor activitati de programare se vor stabili cunostintele necesare realizarii si se va conveni ce limbaje vor fi utilizate.

1.3. Instalare si evaluare

In aceasta etapa se desfasoara activitati care sa pemrita produsului software sa devina operational. Aceste activitati pot presupune:

- Instalarea unui mediu de lucru
- Instalarea unei aplicatii mobile
- Instalarea unei aplicatii BackOffice
- Instalarea unor component de integrare si interfatare
- Instalarea unei baze de date
- Migrarea de fisiere, date si informatii
- Traininguri pentru utilizarea anumitor componente

Pentru a putea fi evaluata functionalitatea se vor face teste cu utilizatorii cheie ai produsului software sau cu potentiali utilizatori. Acestia vor avea posibilitatea sa raporteze disfunctionalitatile, erorile si eventualele sectiuni carora li se mai pot aduce imbunatatiri.

1.4. Mentenanta si suport

Aceasta etapa finala presupune finalizarea dezvoltarii produsului software prin predarea documentatiei, solutionarea disfunctionalitatilor sesizate in etapa anterioara, sesiuni de training cu viitori utilizatori sau realizarea de tutoriale pentru a creste gradul de accesibilitate al produsului.

De asemenea, dezvoltatorii unui produs software vor oferi suport celor care il utilizeaza de-a lungul timpului pentru a asigura buna sa functionare si a repara eventualele erori aparute. In plus, este posibila imbunatatirea unui produs software, caz in care documentatia originala va fi necesara.

1.5. Verificarea produselor software

Verificarea presupune actul de a stabili si documenta faptul ca articolele, procesele, serviciile sau documentele sunt in conformitate cu cerintele specificate.
Activitatile de verificare includ:

- Revizii (Reviews)
- Urmarire (tracing)
- Demonstratii formale
- Testare

Activitati de verificare pe parcursul ciclului de viata

Documentul de cerinte software este verificat fata de documentul de cerinte utilizator, conform sectiunii SR din planul de verificare si validare.

„ADD – SRD”

„DDD –ADD”

„Code –DDD”

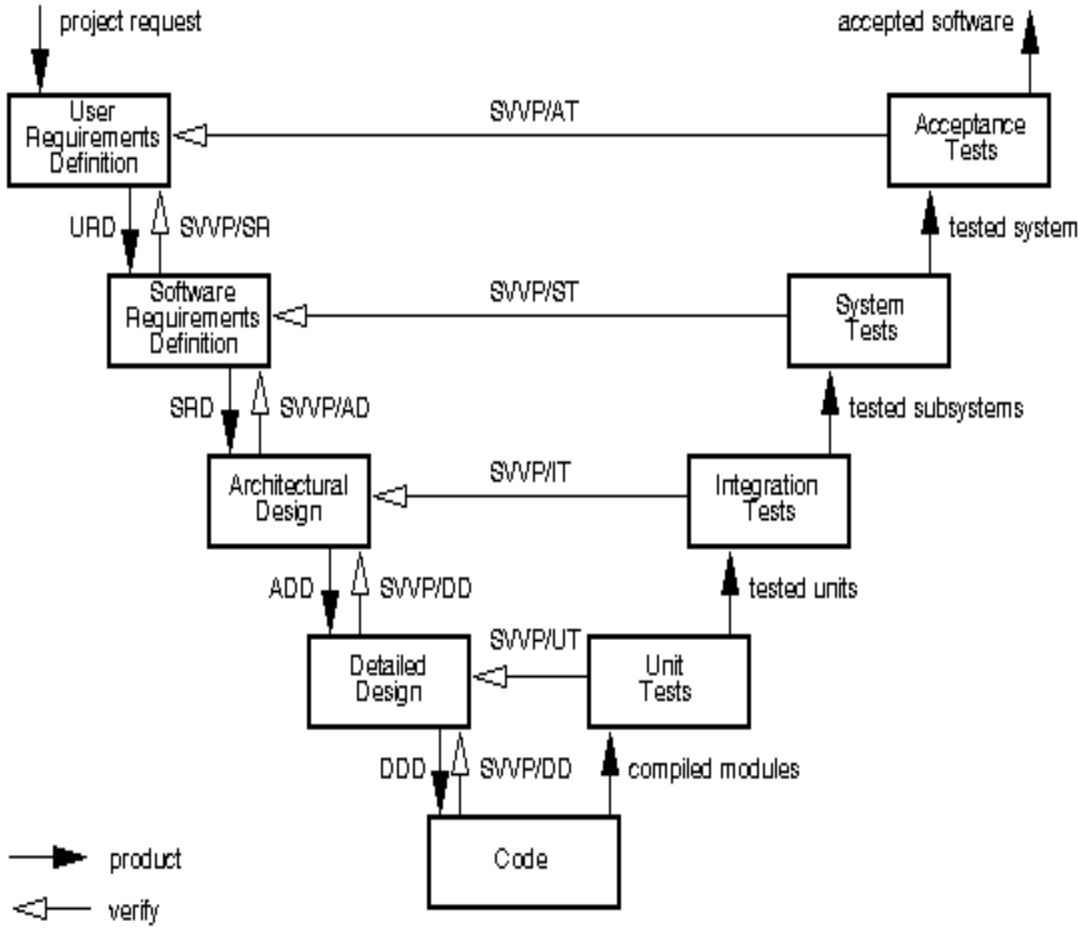
Unit tests „verifica faptul ca modulele software functioneaza corect ca entitati independente, conform specificatiei din DDD si sectiunii UT din SVVP.”

„Integration tests”

„System tests”

„Acceptance tests”

Acestea sunt reprezentate in schema de mai jos.



Sursa foto: http://software-document.blogspot.ro/2011/06/software-verificatio-introduction.html

2. Concepte fundamentale ale testarii

Herea Cristian 441 A

2.1 Definitia testarii

Ce este “testul” ?

Este un set ce contine unul sau mai multe cazuri de test.

Ce este “testul de caz” ?

Este un set de valori de intrare, preconditii de executie, rezultate asteptate si postconditii de executie, dezvoltate pentru un obiectiv particular sau conditie de test cum ar fi repetarea unei secvente a unui program sau pentru a verifica conformitatea cu anumite specificatii.

Ce este “Depanarea – Debugging” ?

Este o activitate de dezvoltare care identifica cauza unui defect, repara codul si verifica daca defectul a fost fixat corect.
Debugging-ul si testarea sunt 2 lucruri diferite.

Ce este “eroarea” ?

Este o actiune umana care produce un rezultat incorrect.

Ce este “defectul = greseala = bug = problema” ?

Este un defect intr-un sistem sau o component care cauzeaza ca sistemul sau component respective sa nu reuseasca sa isi indeplineasca functia. Un defect, daca este intalnit la executie, poate avaria intregul sistem.

Ce este “anomalia” ?

Reprezinta orice conditie care determina deviatia de la asteptari, bazate pe anumite specificatii, documente de design, standarde etc. sau de la perceptia sau experienta cuiva

Ce este “mascarea defectului” ?

Este o modalitate prin care un defect impiedica detectia altuia.

Testare si Calitate

Calitatea – Totalitatea caracteristicilor unei entitati care are abilitatea de a satisfice anumite nevoie implicite sau explicite. Conform DIN-ISO 9126calitatea software-ului include: siguranta, uzabilitate, eficienti, mentenabilitate si portabilitate/aplicabilitate.
Siguranta – Abilitatea produsului software de a exercitate functiile cerute in conditii specificate pentru o anumita perioada de timp sau pentru un anumit numar de ciclii.
Uzabilitate – Capacitatea produsului software de a fi inteles, invatat, utilizat si atractiv pentru utilizator atunci cand este folosit in anumite conditii.
Eficienta – Capacitatea produsului software de a avea performante corespunzatoare, relative la resursele si conditiile date.
Mentenabilitate – Usurinta prin care un produs software poate fi modificat pentru a se corecta anumite defecte, pentru a avea noi cerinte, pentru a face ca in viitor mentenata sa fie mai usoara sau pentru a se adapta la un mediu schimbat.
Portabilitatea – Usurinta prin care un produs software poate fi transferat de pe un echipament pe altul sau dintr-un mediu in altul.
Asigurareacalitatii – Parte a managementul calitatii concentrata pe furnizarea increderii ca cerintele de calitate vor fi respectate.
Controlul calitatii – O activitate princare se verifica faptul ca un produs este potrivit pentru scopul cerut.

Obiectivele principale in diferite tipuri de testare

Testarea in cadrul dezvoltarii- se refera la cauzarea a cat mai multor esecuri pe cat posibili pentru ca acestea sa fie identificate si fixate.
Testarea de acceptanta – confirmarea faptului ca sistemul functioneaza conform asteptarilor, pentru a castiga incredere ca acesta este conform cerintelor.
Testarea mentenantei – include de obicei teste prin care se verifica daca s-au introdus noi defecte in cadrul anumitor modificari (fixari de bug-uri, introducere de noi functii, etc.)
Testarea Operationala – evaluarea caracteristicilor sistemul precum siguranta sau disponibiliatea.

2.2 Necesitatea testarii

Costul defectelor

Un singur defect poate implica costuri extrem de mari. De exemplu in 1989 un mic defect software de pe calculatoarele de la bordul navei Magellan ce urma sa inspecteze planeta venus ar fi putut cauza pierderea in spatiu a probei de 532 milioane de dolari [1] .
In cazuri extreme un defect poate cauza pierderi de vieti omenesti.
De obicei sistemele critice de siguranta sunt testate in mod exhaustive (avioane, trenuri etc.). Un exemplu negativ poate fi Criza Serviciului de Ambulanta din Londra [2].
Costul unui defect creste proportional cu avansarea in stagiile procesului de dezvoltare pana la momentul incare este detectat.

Stagiul in care defectul este identificat	Costul comparativ
Realizarea Cerintelor	1$
Codarea propriu-zisa	10$
Testarea programului	100$
Testarea sistemului	1000$
Testarea de acceptanta	10 000$
Rularea propriu-zisa	100 000$

Cauze ce pot duce la aparitia defectelor software:

- Nimeni nu este perfect, oricine poate face greseli/omisiuni
- Presiunea crescuta poate duce la greseli
- Restrictiile de timp si buget
- Pregatirea slaba, comunicarea ineficienta
- Cerinte slabe
- Specificatii incomplete

Cat de multa testare este necesara?
Daca se opreste testarea prea devreme, se risca aparitia de erori la rularea propriu-zisa a sistemului. Daca testarea continua pentru o perioada prea lunga se poate amana lansarea produsului, fapt ce poate duce la degradarea imaginii companiei sau chiar pierderi de venituri.
Foarte rar este posibila testarea tuturor posibilitatilor (testare exhaustiva), fapt ce duce la concentrarea atentiei asupra anumitor factori :

- Mangementul si reducerea riscurilor
- Analiza continua a riscurilor produsului
- Testarea categoriilor mari de risc
- Intelegerea efectului pe care l-ar putea avea un produs care nu functioneaza correct

2.3 Principii ale testarii

Principiul 1 – Testarea arata prezenta defectelor
Testarea poate arata ca exista defecte, dar nu poate demonstra nu exista defecte.
Testarea reduce probabilitatea ca produsul sa aiba defecte neidentificate, dar chiar dacanu sunt gasite defecte, nu e un mod de a demonstra corectiudinea.

Principiul 2 – Testarea exhaustive este imposibila

Testarea tuturor combinatiilor posibile de valori de intrare si preconditii nu este fezabila, exceptie facand cazurile critice. In locul testarii exhaustive se folosesc riscul si prioritatile in distribuirea eforturilor de testare.

Principiul 3 – Testarea premature

Activitatile de testare ar trebui sa inceapa cat mai devreme posibil in cadrul ciclului de dezvolatare software si ar trebui concentrate pe obiectivele predefinite.

Principiul 4 – Gruparea defectelor

Doar un numar mic de module contin cele mai multe defecte descoperite inaintea lanasarii produsului, sau arata cele mai multe erori de functionare. Este valabila regula 80/20 – 80% din defecte provin din 20% caracteristici (functionalitati).

Principiul 5 – Paradoxul testarii

Daca aceleasi seturi teste sunt repetate de foarte multe ori, acestea nu vor mai gasi noi defecte. Pentru a se putea continua identificarea defectelor persoanele care se ocupa cu testarea trebuie sa:

- Revizuiasca testele existente
- Adauge noi teste pentru diferite parti ale sistemului

O extensie a paradoxului este ca programatorii invata in mod continuu ce defecte apar si incearca sa nu le mai repete. In cazul acesta persoanele care se ocupa cu testarea trebuie sa:

- Invete noi metode si sa inventeze noi modalitati de testare
- Sa se faca o rotatie intre persoane pentru a se evita ca un tester sa lucreze mereu cu acelasi programator

Principiul 6 – Testarea este dependenta de context

Testarea este facuta diferit in functie de context. Contextul include tipul produsului, telurile sale, riscurile associate, tool-uri disponibile, timp si resurse, expertiza echipei samd. .

Principiul 7 - Absurditatea absentei erorilor

Gasirea si fixarea erorilor nu ajuta daca sistemul construit nu este uzabil si nu indeplineste toate cerintele si nevoile utilizatorului

Nivelele maturitatii testarii(Boris Beizer)

Nivelul 0 – Nu este nici o diferenta intre testare si depanare (debugging). In afara de suport pentru depanare, testarea nu are nici un alt rol.

Nivelul 1 – Scopul testarii este sa arate ca produsul software functioneaza.

Nivelul 2 – Scopul testarii este sa arate ca produsul software nu functioneaza.

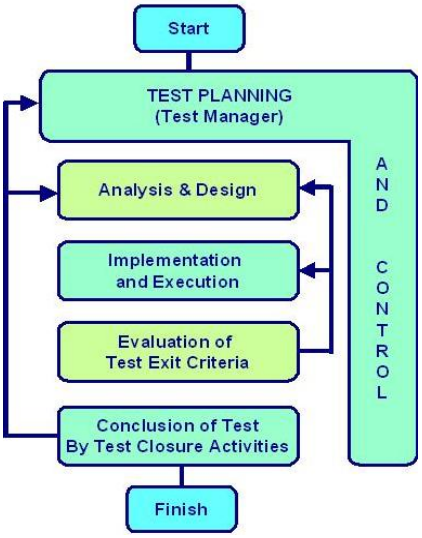
Nivelul 3 – Scopul testarii este nu este sa demonstreze ceva, ci sa reduca riscul perceput ca produsul sa nu mearga la un nivel acceptat.

Nivelul 4 – Testarea nu este o actiune, este o disciplina mental din care rezulta un produse software cu risc minim fara un efort de testare deosebit.

2.4 Procesul testarii

Fazele procesului testarii:

1. Planificarea si controlul testelor
2. Analiza si design-ul testelor
3. Implementarea si executia testelor
4. Evaluarea criteriilor de terminare si raportare
5. Activitati de finalizare a testelor



5 pasi generici implicati in testare [3]

Planificarea testarii

Descrie cum este implementata strategia testarii

Atributii importante:

- Definirea strategiei de testare
- Determinarea scopului si riscurilor
- Determinarea abordarii testarii (tehnici, acoperirea testelor, echipe implicate)
- Programarea implementarii, executiei si evaluarea testelor
- Definirea sistemul de testare (hardware si software)
- Estimarea efortului de testare

Controlul testarii

Este o activitate continua de comparare a progresului actual fata de plan si raportarea starii actuale incluzand si deviatiile de la plan.

Implica luarea actiunilor necesare pentru a se indeplini misiunea si obiectivul proiectului.

Analiza si design-ul testarii

Este un process din 3 pasi:

1. Pregatire si analiza
   - Analizarea aplicatiei
   - Identificarea conditiilor de test
   - Indetificarea cazurilor de test
   - Documentare
2. Construirea cazurilor de test
   - Identificarea datelor de intrare
   - Actiuni necesare
   - Rezultate asteptate
3. Definirea rezultatelor asteptate
   - Rezultatul fiecarei actiuni
   - Starea aplicatiei in timpul si dupa terminarea testului
   - Starea datelor in timpul si dupa terminarea testului

Implementarea si executia testelor
Se bazeaza pe documentele de design-ul si specificatii ale testelor.
In aceasta etapa se dezvolta, implementeaza si prioritizeaza cazuri si proceduri de test, optional se scriu si scripturi de testare automata.

Criterii de prioritizare

Probabilitatea – Daca este o probabilitate mare ca o functie sa nu functioneze corect atunci ar trebui sa ca un test sa fie creat pentru acea parte a sistemului.
Visibilitatea erorilor – Daca o eroare nu are un impact major asupra unui proces, dar este foarte vizibil clientului si/sau apare frecventa, atunci clientul iti va presupuna ca produsul este de calitate slaba
Cerintelor clientului – Prioritatea in cadrul testarii o au cerintele clientului, ceea ce isi doreste clientul ca produsul sa faca sau sa aiba.
Frecventa schimbarilor – Cu cat in cadrul specificatiilor si in codul corespunzator acestora sunt realizate mai multe schimbari cu atat creste riscul de producere a defectelor.
Vulnerabilitatea la erori – Trebuie testate mai intai partile care au cea mai mare vulnerabilitate la erori.
Complexitatea – Codul care are o complexitate mare la creare sau mentenanta va avea aceeasi complexitate la testare.
Timpul si resursele necesare – Definitia prioritatii presupune o limitare in includerea doar acelor parti critice. Cantitatea de resurse disponibile influenteaza scopul si prioritatile testarii – mai multi testeri inseamna ca mai multe teste pot fi create si rulate, si deci mai multe parti ale sistemului pot fi testate.

Evaluarea criteriilor de terminare si raportare
Jurnalul de test ar trebui sa contina:

- Versiunile software-ului si testelor
- Specificatiile folosite ca baza a testelor
- Cronometrarea testelor
- Rezultatele testelor
    - Rezultatele propriu-zise
    - Rezultatele asteptate
- Detalii defectelor (daca se poate)

Din revizuirea jurnalului de test trebuie sa rezulte daca sunt necesare si alte teste sau daca criteriile de terminare ale testelor trebuiesc modificate

Activitati de finalizare a testelor
Aceasta etapa consta in mai multi pasi:

- Verificarea faptului ca produsul rezultat este conform planurilor
- Finalizarea si arhivarea testelor si mediilor de testare

- Daca testarea este finalizata atunci proiectul se poate inchide
- Documentele trebuie sa fie actualizate si inregistrate intr-un sistem de control al versiunii

2.5 Psihologia testarii

Ce trebuie sa faca un tester ?
Sa gaseasca greseli/defecte - cea mai importanta calitate a testarii este de a gasi greseli/defecte:

Testarea:

- Poate fi considerata ca “distructiva"
- Dezvoltatarea este “constructiva”
- Testarea ridica intrebari

Cautarea defectelor intr-un sistem presupune:

- Curiozitate
- Pessimism professional
- Un ochi critic bun, atent la detalii
- O buna comunicare cu programatorii

Ce trebuie sa stie un tester?

- Cum functioneaza proiectul
- Tehnologia folosita
- Anumite aspecte comericale
- Tehnici de testare

Ce abilitati trebuie sa aiba un tester ?

- Sa inteleaga sistemul
- Sa poate citi specificatiile sistemului
- Sa extraga functionalitati ce pot fi testabile
- Sa lucreze efficient
- Sa isi concentreze atentia pe lucruri esentiale
- Sa aiba abilitatea de a gandi si in afara specificatiilor sistemului

Independenta testarii

Exista mai multe nivele ale independentei testarii:

- Nivel scazut – programatorii isi scriu propriile teste
- Nivel mediu – testele sunt scrise de catre alt programator
- Nivel ridicat – testele sunt scrise de catre un alt departament/ alta companie
- Utopie – testele sunt generate automat

3. Testarea produselor software

Petre Radu-Mihai 443 A

3.1 Introducere

Testarea produselor software este un proces sau o serie de procese, menite să asigure ca atunci când se execută un cod de program să facă ceea ce trebuie să facă și nimic altceva. Produsul software trebuie să fie predictibil și consistent, să nu ofere nici o supriză unui utilizator.

Testarea software este ușoară, în anumite metode, datorită faptului că multitudinea de produse software și sisteme de operare sunt mult mai sofisticate decât niciodată, furnizând rutine intrinsece bine testate care pot fi incorporate în aplicație fără ajutorului unui programator care să le construiască de la început. Interfețele grafice (GUI), de exemplu, pot fi construite dintr-o librărie a limbajului de dezvoltare și, dacă aceastea sunt obiecte preprogramate care au fost testate și corectate anterior, nevoia de a testa această parte a unei aplicații personalizate este mult mai redusă.

Cu cât complexitatea dezvoltării programelelor software a crescut, cu atât cererile de inginerie software, tehnologie informațională și calitate profesonială a unui software au crescut. Se așteaptă să se verifice dacă programul software operează în concordanță cu ceea a fost proiectat să facă și să se descopere potențialele probleme care nu au fost anticipate în planul proiectării software-ului. Grupurile de test sunt așteptate să ofere o continuă evaluare a stării curente al unui proiect aflat sub dezvoltare. La orice moment, ei trebuie să fie pregătiți să raporteze detalii explicite despre starea și acoperirea testului, precum și erori nerezolvate. Pe lângă asta, testerii sunt de așteptat să acționeze în calitate de înlocuitori ai utilizatorilor. Aceasta implică să anticipeze probleme de utilizare când mai curând posibil în procesul de dezvoltare pentru ca problemele să fie rezolvate în timp util.

În anii anterior, pe sisteme mainframe, mulți utilizatori erau conectați la un sistem central. Rezolvarea de erori implica updatarea programului stocat central. Aceasta rezolvare singură urma să servească la sute de mii de utilizatori ai sistemului.

Pe măsură ce calculatoarele au devenit mai descentralizate, minicomputerele și microcomputerele erau utilizate ca sisteme de sine stătător sau pe rețea mai mică. Erau multe calculatoare independente sau rețele locale și o rezolvare la un cod de program pe unul din acele calculatoare nu era benefic decât pentru câțiva utilizatori. Companiile software de consum cheltuiau câteodată milioane de dolari trimițând dischete la clienții înregistrați pentru a rezolva un defect serios.

Pe măsură ce piața a devenit mai accesibilă, multă lume utilizează calculatorul pentru multe lucruri, se bazează mult pe un calculator, și consecințele de defecte software cresc an de an. Este imposibil să se găsească toate problemele posibile doar testând programul, dar cu cât costul unei greșeli a crescut, a devenit mai esențial să se facă un test pe bază de risc. Într-o abordare bazată pe risc, se pun următoarele întrebări:

- Care zone ale produsului sunt semnificative pentru client sau care sunt predispuse la un eșec serios ca acestea să fie testate cu mare grijă?
- Cât de multă testare este nevoie pentru anumite secțiuni de program și pentru tot programul?
- Care este riscul implicat în lăsarea unei anumite erori nerezolvate?
- Sunt anumite componente atât de neimportante în cât să nu merite testate?
- În ce punct un produs poate fi considerat testat adecvat și pregătit pentru lansare?
- Cât de mult un produs poate fi amânat pentru testare și rezolvarea erorilor înainte ca viabilitatea pieței să diminueze întoarcerea investiției?

Urmărirea erorilor și asignarea importanței lor sunt prioritare. Echipa de management așteaptă ca echipele de IT și dezvoltare, de testare și asigurarea calității să ofere date cantitative despre testele executate, starea defectelor nerezolvate și potențialul impact asupra amânarii anumitor defecte. Pentru a îndeplini aceste condiții, cei care testează trebuie să înțeleagă produsul și tehnologia folosită. Ei au nevoie de modele pentru a comunica evaluări de cât de multe testări au fost făcute pentru un anumit produs, pâna la ce punct va ajunge testarea și în ce moment produsul va fi considerat testat adecvat. Având informațiile din teste, se vor face predicții despre riscul calității.

În era internetului, conectivitatea între calculatoare a crescut. Calculatoarele personale sunt conectate la internet. Rezolvări de erori și pachete de actualizări sunt disponibile, pe întreaga durata a unei zile, pentru descărcarea imediată din Internet. Caracteristicile care nu sunt disponibile în momentul lansării unui produs software sunt făcute disponibile mai târziu sub forma de pachet de serviciu (service packs).

Cu toate că Internetul oferă conectivitate pentru calculatoare, acesta nu oferă și controlul asupra mediului clientului care era disponibil în modelul mainframe. Provocările dezvoltării și testării cu interfața grafică (GUI) și procesele pe bază de evenimente sunt enorme deoarece clientul încearcă comenzi complexe remarcabile pe sistemele de operare diferite. Există o mulțime nenumărabilă de combinații de procesoare, periferice și aplicații software. Adițional, testarea unui sistem client-server pentru întrepindere poate necesita o mulțime de mii de combinații de sisteme de operare, modemuri, routere și pachete software pentru servere.

Testarea software are rol proeminent în procesul de dezvoltare al unui produs software mai mult decât acum mulți ani. Companiile alocă mai mulți bani și resurse pentru testare pentru că ele consideră că reputația lor constă în calitatea produselor lor.

Fiecare proiect trebuie să conţină un plan amplu de testare care să cuprindă toate funcţionalităţile aplicaţiei şi să asigure funcţionarea corectă a întregului produs. Strategiile de testare se concentrează pe funcţionalitatea şi utilizabilitatea produsului.

Există câteva reguli care sunt considerate ca obiective ale testării:

- testarea este un proces de execuţie a unui program cu intenţia 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 execuţie, independent de datele de intrare;
- metode dinamice de testare: constau în execuţia programului; această metodă este cunoscută şi sub numele de testare structurală.

Dintre cele mai familiare metode statice se amintesc: testarea specificaţiei şi examinarea codului.

La examinarea codului este inclusă şi compilarea. Un compilator modern verifică tot felul de proprietăţi ale programului, şi refuză programele care nu respectă criteriile de corectitudine. Alteori compilatorul dă avertismente asupra unor construcţii care generează probleme la execuţie, cum ar fi de pildă variabile neiniţializate.

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

În cazul testării unui modul ce este dependent de informaţiile 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 fişiere de log sau în baza de date.

Pentru un grup de module ce formează un subsistem se va descompune acest subsistem în funcţii si module, pentru a se putea realiza testarea prezentă. Se testează mai întâi funcţiile apoi modulele şi din aproape în aproape se agregă modulele şi se obţine testarea produsului finit.

3.2. Unelte de testare și managementul configurați ilor [5]

Definiția IEEE despre managementul configurațiilor spune că este o disciplină care aplică direcții tehnice și administrare și se referă la:

- Identificarea și documentarea caracteristicilor funcționale și fizice a configurațiilor
- Schimbarea controlată a acelor caracteristici
- Înregistrarea și raportarea schimbărilor de stări ale procesărilor și implementărilor
- Verificarea conformităților cu cerințele specificate

Scopul este de a stabili și menține integritarea produselor software (componente, date și documentație pe toata durata a proiectului și a vieții produsului. Pentru ingerii de testare software, procedurile și infrastructura(tools) managementului de configurații ar trebui ales și implementat.

Uneltele de testare suportă diferite aspecte ale testării și pot fi clasificate dupa activitățiile de testare care le suportă. Pot fi automatizate cu sarcini cuare sunt repetive sau consumatoare de timp. Pot îmbunătăți eficiența activităților de testare și sunt capabile să facă lucruri care persoanele responsabile de testare nu o pot face ușor. Pot îmbunătăți siguranța testărilor (de exemplu prin automatizarea comparațiilor de date mari sau simularea comportamentului) și pot fi invazivi în ceea ce privește că uneala de testare poate singur să afecteze rezultatul testului (spre exemplu – sincronizarea actuală poate fi diferită depinzând de cum se măsoară cu un test de performanță diferită.

Multe grupuri de programatori se bazează din ce în ce mai mult pe testarea automată, mai ales grupuri care utilizează dezvoltarea bazată pe testare (dezvoltarea bazată pe testare este un proces software de dezvoltare care se bazează pe cicluri de dezvoltare foarte scurte:

prima dată un programator scrie un caz de test automat, inițial să fie picat, care definește îmbunătățiri dorite sau noi funcții, iar apoi face schimbarea minimală a codului ca să treacă testul și în final refactorizează codul nou la standarde acceptabile[29]). Sunt multe limbaje de programare în care testele sunt scrise și programale cu integrare continuă vor rula testele automat de fiecare dată când timpul codului este verificat într-un sistem de control al versiunii.

Uneltele de testare/depanare includ caracteristici precum:

- Monitoare de program, permițând monitorizarea parțială sau totală a codului de program:
  a) Simulator de seturi de instrucțiuni, permite monitorizarea completă la nivel de instrucțiune
  b) Animarea programului, permite executarea pas cu pas și punct de întrerupere condițional la nivelul de sursă sau cod mașină
  c) Rapoarte de acoperire a codului
- Depanarea simbolică, unealtă ce permite inspecția variabilelor de program după eroare sau un punct ales.
- Unealta de testare funcțională și automată cu intefață grafică este utilizată la repetarea testelor la nivel de sistem prin intermediul interfeței
- Benchmark, permite să fie făcute comparații de performanță
- Analiză de performanță (unealtă de profilare) care ajută la identificarea hot spoturilor și utilizarea resurselor

Unele din aceste caracteristici pot fi incorporate într-un mediu integrat de dezvoltare (Integrated Development Environment – IDE).

Simulatorul de seturi de instrucțiuni [26]

Un simulator de seturi de instrucțiuni este un model de simular, programat de obicei într-un limbaj de programare de nivel înalt, care imită comportamentul unui mainframe sau microprocesor prin citirea instrucțiunilor și menținând variabile interne ce reprezintă registrele procesorului

Simularea instrucțiunilor este metodă utilizată pentru una dintre următoarele motive posibile:

- Pentru simularea codului mașină la un alt dispozitiv hardware sau întregului calculator pentru creșterea compatibilității (un simulator de sistem întreg de obicei include un simulator de seturi de instruțiuni)
- Pentru monitorizarea și executarea instrucțiunile codului mașină pe aceeași configurarație hardware pentru scopuri de testare și depanare, de exemplu cu protecția memorii (protejarea împotriva depășirii accidentale sau intenționate de scală a bufferului)
- Pentru a îmbunătăți viteza simulăriilor implicând un nucleu de procesor atunci când procesorul singur nu este un element verificat.
-

Animarea programului[33]

Animarea programului se referă la o metodă comună de depanare, ceea de executarea codului linie cu linie. Programatorul poate examina starea programului, mașinii și a datelor înainte și după executarea a unei linii particulare a codului. Aceasta permite evaluare efectelor a celor declarării sau instrucțiuni în izolare și care câștigă în comportarea bună sau rea în executarea programului. Aproape toate depanatoarele și IDE-urile moderne suportă acest mod de execuție. Câteva unelte de testare permite programului să se execute opțional pas cu pas ori la nivelul de cod sursă sau cod mașină în funcție de disponibilitatea datelelor colectate la compilare.

Acoperirea codului[27][28]

Acoperirea codului (code coverage) este măsură utilizată în testarea software. Descrie gradul de testare a codului sursă a unui program. A fost printre primele metode inventate pentru testarea software sistematică.

În general, uneltele și librăriile de acoperirea codului arătă ce costuri de performanțe sau/și memorie sunt inacceptabile pentru operațiuni normale ale programului. De aceea, ele sunt utilizate doar în laborator. Așa cum era de așteptat, există clase de programe software care nu pot fi subiecte posibile pentru acest tip de test, totuși un grad de acoperire al mapării poate fi exprimat prin analiză de printr-un test direct.

Depanarea simbolică[30]

Un depanator sau o unealtă de depanare este un program care este folosit pentru a testa și depana alte programe.

Un „crash” („căderea„ programului) se întâmplă atunci când un program nu poate normal să continue să ruleze din cauza unei erori de programare. Spre exemplu, programul poate încearcă să utilizeze o instrucțiune care nu este disponibilă pe versiunea curentă a procesorului sau încearcă să acceseze o zonă de memorie indisponibilă sau protejată. Când un program face crash sau ajunge la condiție presetată, depanatorul arată de obicei locația în codul original dacă este depanare la nivel de sursă sau depanare simbolică.

Unealta de testare funcțională și automată cu intefață grafică[32]

Multe unelte de automatizare a testărilor oferă caracteristici de înregistrare și redare care permite utilizatorilor să înregistreze interactiv acțiuni și apoi să le repete de un număr de ori, comparând actualele rezultate cu cele așteptate. Avantajul acestei metode este că necesită puțină sau chiar deloc dezvoltate software. Această metodă se poate aplica pe orice aplicație care o interfață grafică.

Benchmark[31]

Benchmarkul este o formă a rulării unui program, un set de program sau alte operațiuni, cu scopul de estima performanța relativă a unui obiect, normal prin rularea unui număr de teste standarde.

Benchmarkurile sunt făcute să mimeze un tip particular de volum de muncă a unui componente sau sistem. Benchmarkurile sintetice fac acest lucru prin programe speciale create care impune un volum de muncă asupra unei componente.

3.3 Tehnici de testare software

Testarea arhitecturii Client/Server [6]

Aplicaţiile bazate pe arhitectura „Client/Sever" sunt considerate aplicaţii complexe datorită numărului şi diversităţii modulelor de prelucrare. Aceasta presupune o aplicaţie distribuită. În general o aplicaţie distribuită cuprinde un sistem central, mai multe subsisteme conectate la sistemul central şi mai mulţi clienţi conectaţi la un subsistem. Complexitatea arhitecturii este reflectată şi în testare.

Testarea acestei arhitecturi presupune trei niveluri diferite:

- client individual, caz în care aplicaţia client este testată individual, în mod deconectat şi are ca efect acceptarea sau respingerea modulelor;
- client şi server, caz în care acestea sunt testate împreună dar nu se ia în considerare reţeaua şi are ca efect acceptarea sau respingerea interacţiunii client-server;
- client, server şi reţeaua, caz în care se testează tot ansamblul împreună şi se verifică dacă sistemul este corect, complet şi funcţionează în mediu real.

Sunt mai multe tipuri de teste care duc la detalierea acestor niveluri şi anume:

- testarea funcţionalităţilor;
- teste de server în care se verifică funcţiile de management al datelor şi performanţele serverului;
- teste de baza de date în care se probează acurateţea şi integritatea datelor salvate de server cât şi metodele de arhivare;
- testarea tranzacţiilor în care se verifică clasele din punct de vedere al tranzacţiilor în conformitate cu cerinţele; testul se focalizează pe corectitudinea procesului şi pe performanţele acestuia.
- testarea comunicaţiei în care se verifică traficul pe reţea, volumul datelor şi corectitudinea lor; testele de securitate trebuie să facă parte din aceste teste.

Test de compatibilitate şi configurare [6]

Testarea compatibilităţii presupune verificarea interacţiunii software cu celelalte componente software cu care va coexista şi va interacţiona.

Testarea produsului pe mai multe platforme este o muncă foarte costisitoare, atât din punct de vedere al testării cât şi din punct de vedere al rezolvării problemelor care sunt descoperite.

Testarea compatibilităţii, realizată în „beta testing", reprezintă o testare externă cu un grup selectat ca potenţiali clienţi. Selectarea grupului se efectuează după criterii precise

întrucât rezultatele testării sunt cu atât mai concludente cu cât există garanţia că aceşti clienţii selectaţi utilizează o diversitate cât mai mare de module.

O aplicaţie complexă are în componenţă multiple configurări de parametrii, de variabile de environment, setări dependente de sistemul de operare, de tipul bazei de date, de diferitele configurări pe care le permite aplicaţie în funcţie de potenţiali clienţii. Foarte multe din aceste configurări se centralizează pe un singur calculator al clientului şi se distribuie automat pe celelalte calculatoare.

Testarea configurării presupune verificarea îmbinării dintre setările posibile şi ambientul software şi hardware existent. Dacă aplicaţia este proiectată să folosească un scanner trebuie să fie compatibilă cu hardwareul existent. Trebuie făcute setările necesare pentru a putea utiliza un anumit tip de hardware.

Testarea utilizabilităţii [6]

Multe companii de software cheltuiesc foarte mult timp şi bani pe găsirea celei mai bune căi de proiectare a interfeţei cu utilizatorul, „user interface" - UI.

Pentru realizarea unei bune interfeţe cu utilizatorul s-au desprins următoarele caracteristici importante:

- urmărirea utilizării unor standarde - De obicei standardele sunt dependente de sistemele de operare. Dacă o platformă nu are un standard sau se doreşte adaptarea unui standard nou, atunci echipa de proiectare va crea un nou standard cât mai eficient;
- intuitiv - alegerea unui standard care să fie cât mai uşor de înţeles;
- consistenţa - folosirea consecventă a standardului ales în toate aplicaţiile sau modulele dezvoltate: „shortcut key", „meniu", „terminologie", „numire", „consistenţă" în mesaje de erori, plasament al butoanelor fără a fi create confuzii, utilizând secvenţe acumulate anterior;
- flexibilitate - să permită setarea diferitelor caracteristici ale aplicaţiei dorite de clienţi culori, valori default, diferite căi de introducere de date pentru a se potrivi cerinţelor utilizatorilor şi obiceiurilor lor;
- concret - să fie bine precizat şi bine definit;
- utilitate - să fie folositor şi să dea un randament bun, astfel duratele de prelucrare să fie reduse
- accesibilitate - prietenos şi uşor de utilizat prin mesaje comune cu alte produse şi având grupe de taste de comandă cunoscute;

Ca parte a testării utilizabilităţii este şi testarea documentaţiei. Aceasta este de multe ori considerată ca fiind o componentă non-software. Dar o documentaţie bună duce automat la creşterea utilizabilităţii produsului şi implicit la creşterea operaţionalităţii.

4. Tipuri de teste software

Paraschiv Radu 443A

Testarea reprezinta o etapa importanta in procesul de realizare a produselor software. Tehnicile si metodele moderne de dezvoltare a produselor software au ca scop inlaturarea erorilor de analiza, proiectarea , programarea si obtinerea unei aplicatii software.

In literatura exista multe definitii a termenului de software testing, insa toate se rezuma la aceeasi idee: software testing inseamna operatia de a rula un produs software intr-o maniera controlata, pentru a vedea daca acesta se comporta conform specificatiilor. Acest termen se foloseste in general alaturi de alti doi termeni : verificare si validare. Verificarea este procesul de testare a unui obiect, inclusiv un produs software, pentru a ii determina conformitatea si consistenta in corelatie cu specificatiile sale. Testarea software este de fapt un fel de verificare. Validarea este procesul de confirmare a faptului ca produsul indeplineste cererile clientului.

"Termenul de bug este adesea folosit pentru a descrie o problema intr-un calculator. Exista doua tipuri de bug-uri : cele de software si cele de hardware. In aceasta privinta testarea software nu ar trebui asociata cu debugging. Debugging-ul este procesul de analiza si localizare a bugului atunci cand produsul nu se comporta conform asteptarilor. Desi identificarea unor bug-uri vor fi evidente atunci cand evaluam softwar-ul , o abordare metodica a acestor probleme este necesara pentru a le rezolva."[2]

4.1. Elaborarea metodelor de testare

Elaborarea metodelor de testare este supusa acelorasi principii ca si elaborarea produselor software. Un design bun are mai multi pasi ce evolueaza in mod progresiv designul testului incepand cu un nivel initial ridicat, de baza si descinde catre un nivel detaliat. Aceste etape sunt: strategia de test, planificarea testului si designul procedurii de testare.

Designul testului trebuie sa fie organizat conform specificatiilor software-ului ce trebuie testat. La un nivel inalt acest lucru inseamna ca testul va avea ca scop verificarea functionarii corecte si conform specificatiilor a produsului softeare.

Strategia de testare. Aceasta este prima etapa in formularea unei strategii de testare. O strategie de testare este de fapt modul de abordare a testului, identificarea nivelului la care este nevoie de realizarea testului si metodele si tehnicile folosite in realizarea acestuia. Aceasta strategie ar trebuie , in mod ideal, sa fie aplicabila tuturor produselor software din aceeasi gama. Elaborarea unei strategi de testare ce indepliceste necesitatile utlizatorului este critica pentru dezvoltatorul produsului.

Planificarea testului. Urmatorul pas in dezvoltarea testului este acela de realizarea a planului de testare. Un astfel de plan specifica ce rezultate va returna programul, la ce nivel se va aplica acest test, in ce ordice se va realiza testarea, cum se va aplica testul fiecarei componente a produsului si descrierea mediului de testare.

"Un plan de test poate fi proiectat pe larg , sau poate fi o ierarhie de planuri in raport cu nivelurile de specificare si testare:

• Un plan de test a acceptantei, descrie planul testului de acceptanta a produsului software. Acesta este publicat de obicei ca si documente separate, dar poate fi publicat impreuna cu testul in sine.

• Un plan de test a sistemului, descrie planul de integrare cu sistemul si testare. La fel cu testul anterior modul de publicatie poate fi separat sau impreuna cu produsul in sine.

• Un plan de interograre software, descrie planul de interogare a testului.

• Un plan de test pe unitati, descrie planul de testare a unitatilor individuale din cadrul unui produs software.

Scopul fiecarui test este de a oferi un plan de verificare prin testare individuala a componentelor unui produs software."[1]

Designul procedurii de testare. O data ce planul de test a fost intocmit, urmatorul pas pentru crearea produsului este specificare caii urmarite de test pentru ficare componenta in parte. Ficare caz va determina cat de importatnata este calea respectia in raport cu celelalte cazuri. Este important sa se creeze aceste cazuri atat din perspectiva testarea pozitive cat si negative. Testarea pozitiva verifica faptul ca softwar-ul face ceea ce trebuie, iar testarea negativa verifica daca produsul nu indeplineste functia dorinta.

4.2 Tipuri de testare

Pentru a determina adevarata functionare a aplicatiei ce este testata, o multitudine de teste au fost elaborate in acest scop. Cele mai importatne sunt:

4.2.1. Testarea white-box. "Acest tip de testare verifica structura interna a programului, ce este de obicei transparenta utilizatorului. In acest tip de testare sunt folosite atat o perspectiva interna cat si metodele de programare pentru a determina cazurile de testare. Testul alege date de intrare pentru a verifica liniile de cod si pentru a determina datele de iesire. Aceasta metoda este asemanatoare cu verificarea nodurilor intr-un circuit."[3]

Cu toate ca acest tip de testare poate fi aplicat la nivel de unitate, integrare si sistem, este de obicei folosit la nivel de unitate. Poate verifica liniile dintr-o unitate, legaturile dintre unitati si legaturile dintre subsisteme atunci cand este vorba de verificarea intregului sistem. Prin aceasta metoda de testare se pot descoperii multe erori si probleme insa este posibila nedescoperirea unor parti lipsa ce ar fi trebuit implementate in urma cerintelor clientului.

"Tehnicile folosite in aceasta metoda pot include:

• Testare prin API(application programming interface) - testarea aplicatiei prin API-uri publice sau private.

• Acoperirea codului - creearea testelor pentru a satisface unele criterii ale codului (de exemplu testul poate sa creeze un caz in care toate liniile de cod ale unui program trebuie sa se execute cel putin o data).

• Introducerea erorilor - crearea si introducerea intentionata a erorilor pentru a masura efectul lor asupra rezultatului final."[1]

Ustensilele folosite in tehnica acoperirii codului pot fi folosite pentru a completa testul creeat anterior. Acest lucru permite echipei de dezvoltare sa verifice parti ale sistemului care sunt rar folosite si sa se asigure ca cele mai importate puncte ale acestora au fost testate. O acoperire de 100% asigura ca toate liniile de cod au fost executate cel putin o data. Acest lucru este folositor la determinarea functionarii unui program, dar nu este suficienta deoarece o linie de cod poate sa proceseze unele date de intrare corect, iar altele incorect.

4.2.2. Testarea Black-box. Acest mod de testare trateaza software-ul ca o cutie neagra, examinand-ui functionalitatea fara a avea vreo cunostinta asupra ceea ce se intampla in interiorul programului. Testul stie doar ce ar trebui sa se intample la introducerea unor date si nu cum se obtin aceste date.

Metodele de testare de tip black-box includ:

•Analiza valorilor limita: aceasta metoda testeaza valorile limita ce pot aparea in derularea unui program.

•Testarea bazata pe specificatii: aceasta are ca scop testarea functionalitatii software-lui conform cerintelor aplicatiei. Acest nivel de testare necesita cazuri de testare amnuntite, care pe urma pot fi verificate prin aplicarea lor la intrarea programului si verificarea rezultatelor de la iesire. Aceste date de intrare sunt modelate pe baza cerintelor si specificatiilor, adica ce ar trebui ca aplicatiea sa returneze.

Testarea bazata pe specificatii este suficienta pentru a determina daca un program functioneaza corect, dar este insuficienta pentru a prevenii sitoatiile complexe.

Un avantaji al testarii black-box este acela ca nu este nevoie o cunostinta avansata in domeniul programarii.Pentru orice cazuri exceptionale ale programului cel care testeaza are un set de date de intrare si iesire specifice pentru acestea. Un mare desavantaji al acestei metode de testare este acela ca nu verifica liniile de cod , de unde erorile provin, ci mai de graba verifica daca programul functioneaza sau nu.

4.2.3. Testarea gray-box. "Aceasta metoda implica o cunostinta a structurilor de datelor interna si a algoritmilor in scopul elaborarii testelor, ce vor fi executate la nivel de teste black-box. Cel care testeaza nu trebuie neaparat sa aiba acces la codul sursa pentru a realiza un astfle de test." [3]

Manipuland datele de intrare si determinarea valorilor de iesire nu este considerata o testare gray-box, deoarece intrarile si iesirile sunt incadrate in testarea black-box. Aceasta deosebire intre cele doua tipuri de testare este importanta in mod special atunci cand se realizeaza testarea prin interogare a doua module de test scrise de doua echipe diferite. Cu toate aste modificarea datelor de test este considerata o testare gray-box , deoarece utilizatorul nu ar putea face acest lucru in mod normal. Testarea gray-box poate include si ingineria inversa pentru a determina valori limita sau mesaje deeroare.

Stiind conceptul pe care se bazeaza software-ul, persoana care testeaza poate lua decizii mai bune cand vine vorba de datele de la intrare. In mod tipic , testarea grey- box permite realizarea testarii intr-o zona izolata cum ar fi o baza de date. Utilizatorul poate sa

observe starea produsului ce este testat dupa realizarea unor activitaii anume, cum ar fi extragerea din baza de date respectiva a unei valori. Testarea grey-box implementeaza scenarii inteligente , bazate pe informatiile ce ii sunt oferite.

4.2.4. Testarea vizuala. "Scopul testarii vizuale este acela de a oferii dezvoltatorilor o imagine asupra ceea ce s-a intamplat in momentul in care o eroare s-a produs. In acest fel problema se poate remedia mult mai usor, deoarece informatia este prezentata direct , fara a mai fi nevoie de o rulare mai amanuntita a programului."[4]

Testarea vizuala se bazeaza pe ideea ca indicarea directa a problemei este mult mai eficienta decat descrierea acesteia. Asatar aceasta metoda necesita inregistrarea intregului proces de testare, identificand tot ceea ce se petrece in sistemul de test sub format video.

Testarea vizuala oferea un numar mare de avantaje. Calitatea comunicarii este marita deoarece testarea poate identifica problema (si evenimentele ce conduc la aceasta) ofera o imagine dezvoltatorului in loc sa ii descrie problema. In acest mod programatorul va avea tot ceea ce are nevoie pentru a determina sursa exacta a erorii.

"Testarea ad hoc si testarea prin explorare sunt metode importante pentru verificarea integritatii software, deoarece acestea nu au nevoie de timp pentru a fi implementate, in timp ce bug-urile sunt detectate usor de catre acestea. Pentru testatea ad hoc , unde testul are loc intr-un mediu improvizat, abilitatea unui program de testatre de a inregistra tot ceea ce se intampla este foarte importanta."[1]

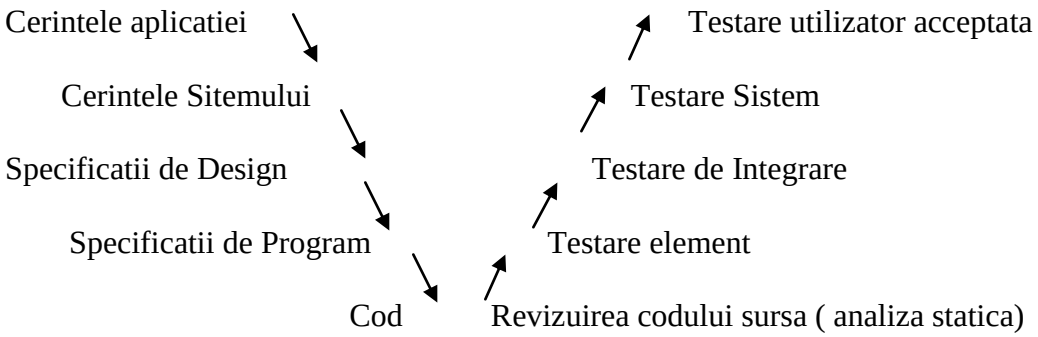
Testarea vizuala este din ce in ce mai folosita deoare ea aduna informatii ce pot fi folosite in procesul de dezvoltare ce pot fi accesate si de catre alte persoane ce nu au lucrat pana in acel moment la proiect.

5. Diferite metodologii de testare: nivele si tipuri de testare

Miu Madalina 441A

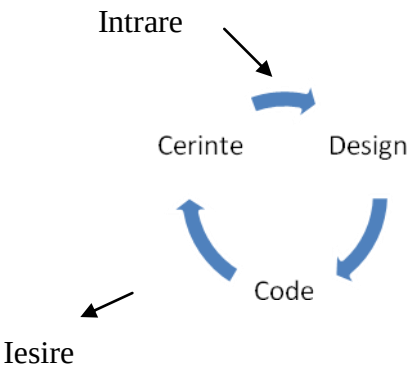
5.1 Modele software de dezvoltare

- *Modelul V*



Modelul V arata diferitele stagii in dezvoltare si testare, precum si relatiile dintre acestea.Se uitilizeaza patru nivele de testare pentru cele patru nivele de dezvoltare. In practica, se pot intalni mai putine nivele de testare si dezvoltare, depinzand de proiect.

Modelul de Dezvolatre Iterativ-Incremental



Acest model preupune impartirea proiectului intr-o serie de subproiecte mici, iar fiecare subproiect urmareste un model V. Modelul de Dezvoltare Iterativ-Incremental are urmatoarele avantaje: analiza timpurie a riscurilor, revizitarea planurilor, calitate pe nivele si o mai mare motivatie.

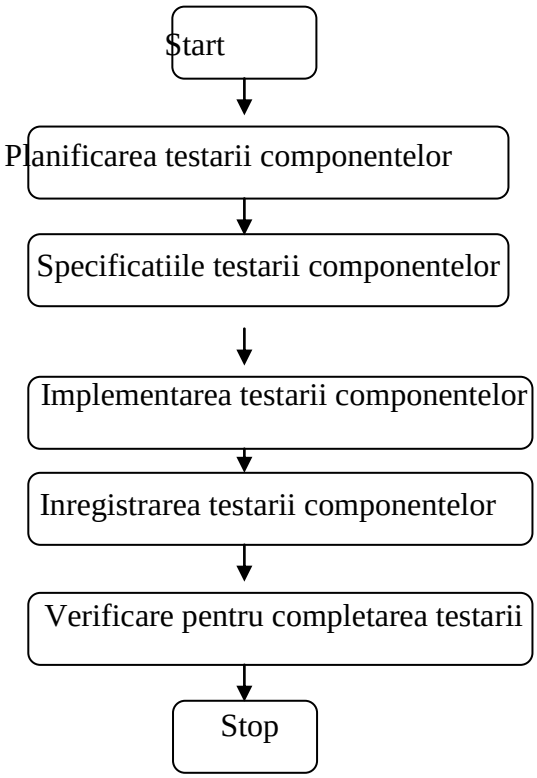
Nu conteaza ce model se foloseste, este necesara o testare buna, intrucat pentru fiecare activitate de dezvoltare corespunde o activitate de testare, fiecare nivel de testare are obiective specifice pentru acel nivel, analiza si designul testarilor pentru un anumit nivel de testare ar trebui sa inceapa corespunzator activitatii de dezvoltare.

5.2 Testarea Componentelor

Ce este o componenta? O componeta este un obiect software minimal care poate fi testat independent.

- La nivelul inferior al modelului de testare V problemele sunt rezolvate atunci cand apar, fara a se inregistra incidente.
- Fiecare componenta este testata separat, izolata de restul sistemului
- Implica testarea codului in cele mai mici detalii (facuta de obicei de dezvoltatorul de componente).

Procesul de testare a componentelor :



5.3 Testarea integrarii

Testarea integrarii este efectuata pentru a evidentia defectele de la nivelul interfetelor si interactiunea dintre componentele integrate ale sistemului.

Aceasta testare se bazeaza pe sistemul de design si software, arhitectura, fluxuri de lucru si cazuri de utilizare. Cele mai utilizate obiecte de testare sunt implementarile de baze de date a sub-sistemelor, infrastructura, interfetele si configurarea de date.

Toate componentele unei aplicatii lucreaza cu un singur lucru: date. Datele sunt pasate de la un calculator la altul sau de la un sistem la altul. Aceste sisteme pot rula aceleasi sau diferite tipuri de sisteme de operare si se pot afla in locatii diferite si folosesc protocoalele de comnicare pentru schimbul de date.

Componentele de sistem trebuie sa adauge, modifice si sa stearga informatia, sa o valideze si sa o raporteze.

O data ce componentele au fost testate, le putem lega inpreuna sa vedem daca ele pot conlucra. Acest lucru presupune legarea in mod gradual a componentelor, dupa care trebuie aratat ca datele sunt transmise intre componente cum este cerut. In mod constant se creste numarul de componente si se creaza si testeaza subsisteme pana se ajunge la sistemul complet.

Testarea integrarii poate fi incrementala (de sus in jos, de jos in sus sau functionala) sau neincrementala de tip „big-bang”.

5.4 Testarea sistemului

Testarea sistemului reprezinta procesul testarii unui sistem integrat pentru a verifica daca acesta indeplineste cerintele specifice.

Testearea unui sistem complet poate fi considerat ultimul pas in testarea intergrarii si , totodata, poate fi prima data cand o parte suficienta din sistem a fost unita in vederea unui sistem functional.

Acest tip de testare este deseori realizata de o echipa de testare independenta si se bazeaza pe specificatiile necesare ale sistemului si sofware-ului, cazuri de utilizare, aplicatii functionale si rapoarte de analiza a riscurilor si poate fi functionala sau nefunctionala.

Testarea functionala este orientata pentru a verifica functionalitatea sistemului in fata specificatiilor.

Testarea bazata pe cerinte specifica anumite functii pe care un sistem sau o componenta de sistem trebuie sa le indeplineasca. Specificatiile sunt utilizate pentru a indeplini anumite cazuri de testatare. Sistemul este testat pentru a se verifica daca aceste cerinte sunt indeplinite.

Testarea nefunctionala implica testarea atributelor componentelor sau a sistemului care nu se leaga de functonalitate, de exmplu: relabiliatatea, eficienta, mentabilitatea, portabilitatea, utilitatea.

De obicei, utilizatorii se axeaza pe functionalitatea sistemului. Testarea functionalitatii arata daca sistemele functioneaza conform cu cerintele, dar exista si alti factori, in afara de functionalitate care sunt importanti in utilizarea sistemului.

Testarea nefunctionala prezinta teste asupra capacitatii de memorie a sistemului, de instalare, de documentare, de recuperare, de incarcare, de stres, de performanta si de volum al datelor.

5.5 Testarea de acceptare

Testarea de acceptare este o testare care are in vedere nevoile si cerintele utilizatorilor. Este o metoda menita sa determine daca un sistem indeplineste sau nu criteriul de acceptare. In acelasi timp, utilizatorul, clientul sau orice alta entitate asociata determina daca accepta sau nu sistemul.

In cazul testarii de acceptare gasirea de defecte nu este scopul principal si nu este neaparat ultimul nivel al testarii.

Testarea de acceptare se poate realiza pe mai multe nivele.

Un produs software poate fi testat din acest punct de vedere cand este instalat sau integrat. Testarea de acceptare a utilizarii unei componente poate fi facuta in timpul testarii componentelor, iar testarea de acceptare a unei noi functionalitati imbunatatite se poate realiza inainte de testarea sistemului.

Forme obisnuite de testare de acceptare:

1. *Testare de acceptare* a utilizatorului: utilizatorului unui produs final efectueaza testele. Setarile lor vor reprezenta un mediu de lucru si acopera toate ariile poriectului, nu numai ale sistemului.

2. *Testarea de lucru* presupune testarea unei rezerve(backup), refacerea in caz de dezastru, mentenanta task-urilor si verificari periodice ale vulnerabilitaii securitatii.
3. *Testarea privind acordul si reglementarile* reprezinta o demnostratie a criteriului de acceptare care a fost definit in contract. Inainte ca software-ul sa fie acceptat, este necesar sa se arata ca el indeplineste conditiile asa cum sunt specificate in contract. Acest tip de testare este efectuat contrar oricarei reglementari la care produsul trebuie sa adere cum ar fi guvernamentale, legale sau de siguranta.
4. *Testarea Alpha* permite testarii de catre client. Oamenii care reprezinta tinta folosesc produsul in acelasi fel ca si cum ar fi cumparat versiunea finala. Aceasta testare se realizeaza in casa (pe site-ul dezvoltatorilor).
5. *Testarea Beta* se aseamana cu testarea Alpha, numai ca utilizatorii efectueaza testarea pe propriile site-uri.

5.6 Testarea functionala

Testarea functionala poate fi efectuata la toate nivele de testare si considera comportamentul extern al software-ului (testarea black-box).

Tipui de testare functionala:

1. *Testarea de siguranta*: investigheaza functii (ex. Firewall) care au in vedere detectia de amenintari cum ar fi virusi si factori daunatori.
2. *Testarea de inter-operabilitate*: evalueaza capabilitatea produsului software de a interactiona cu o sau mai multe componente specifice ale sistemului.

O specifictie functionala este o descriere a comportamentului care se asteapta de la program si este prima sursa de informatie in case unui caz test de specificare.

Testarea functionala poate fi aplicata la orice nivel unde exista o forma de specificare disponibil, de la tot sistemul pana la unitati independente.

Acest mod de testare este impartit in 5 pasi:

- Identificarea functiilor pe care software-ul trebuie sa le indeplineasca
- Crearea unei date de intrare bazate pe specificatiile functiilor
- Determinarea unei iesiri bazate pe specificatiile functiilor
- Executarea cazului de test
- Compararea software-uluii cu iesirile asteptate de la acesta

5.7 Testarea nefunctionala

Testarea caracteristicilor nefunctionale sunt menite sa masoare caracteristicile sistemului si software-ului care pot fi cuantificate la o scara larga (ca timpul de raspuns al unui test de performanta).

Este un test despre cum merge sistemul si poate fi realizat la toate nivelele.

1. *Testarea utilizabilitatii*: determina masura in care produsul software este inteles, usor de invatat, usor de operat si atractiv catre utilizatorii supusi unor conditii specifice.

Un test Beta este o metoda usoara de a realize testul de uzabilitate.

Testarea usurintei de invatare difera de testarea usurintei de utilizare.

2. *Testarea stresului*: evalueaza sistemul sau o componenta la sau dincolo de limitele specificate.
3. *Testarea capacitatii de memorare* (utilizarea resurselor) reprezinta procesul de testare pentru determinarea utilizarii resurselor unui produs software. Este analizata memoria ocupata de produs si se prezice atunci cand este nevoie de mai multa capacitate.
4. *Testarea performantei*: determina performanta produsului software si este asociata cu testarea eficientei.
5. *Testarea recuperarii* (recuperabilitatii sistemului) reprezinta procesul de testare in care se determine recuperabilitatea sistemului. Poate include recuperarea de pe sistemul de rezerva (backup) si este asociata cu increderea.
6. *Testarea volumui de date*: apare in cazul in care sistemul este supus unui volum mare de date si este asociata cu testarea utilizarii resurselor.
7. *Testarea instalarii unui produs software*: raspunde la intrebari ca: Afecteaza instalarea acestui produs alte sisteme software sau sistemul software gazda? Poate fi instalat in medii documentare? Se dezinstaleaza correct?
8. *Testarea documentatiei*: testarea calitatii documentatiei, de exemplu ghidul instalarii, ghidul utilizatorilor sau alte manuale. Aceasta testare trebuie sa verifice daca documentatia este completa, corecta si disponibila.
9. *Testarea de incarcare*: aceasta testare este orientate catre evaluarea abilitatii aplicatiei de a manevra datele si utilizatorii care se asteapta a fi tranzitate. Acest test analizeaza comportomentul unei componente sau a sistemului in conditii de

incarcare sporita, de exemplu: numarul de utilizatori paraleli sau numarul de tranzactii pentru a determina ce incarcatura poate manevra componenta sau sistemul.

5.8 Re-testarea

Re-testarea include testarea de confirmare si testarea de regresie. Atunci cand este corectat un defect, este lansata o noua versiune a software-ului. Apoi, este nevoie de inca o testare pentru a se dovedi ca defectul a fost corectat.

*Testarea de confirmare*este o testare care ruleaza cazuri de test care au esuat ultima oara cand au fost rulate, in vederea verificarii succesului actiunilor corectoare.

Confirmare – dupa ce un defect este detectat, software-ul trebuie re-testat pentru a confirma ca defectul initial a fost indepartat cu success.

Depanare – activitate de dezvoltare (este diferita de testarea de activitate).

Abilitatea pentru re-testare include:

- Necesitatea unor planificatoare care includ re-testarea
- Testele trebuie sa fie usor de re-rulat
- Datele trebuie sa fie usor de re-utilizat
- Mediul trebuie sa fie usor de re-actualizat

Testarea regresiei reprezinta testarea unui program testat anterior care a suferit modificari pentru a se asigura ca nu au fost introduse sau ascunse defecte ca rezultat a modificarilor facute. Acest tip de testare este realizat atunci cand este schimbat software-ul sau mediul.

Masura in care se realizeaza testarea regresiei este bazata pe riscul negasirii defectelor in software-ul care a functionat anterior. Acest tip de testare la toate nivele de testare.

Testarile sunt deaseamnea necesar de reluat atunci cand se verfica actualizarile software-ului.Aceste teste trebuie reluate atunci cand se semnaleaza o modificare in software sau mediu si sunt utilizate pentru a se dovedi ca nu au fost schimbate aspecte ale sistemului sau componentelor.

Selectia cazurilor de test pentru testarea regresiei cuprinde

- Teste care acopera functii critice de securitate
- Teste pentru arii care se schimba in mod regulat

- Teste ale functiilor care au nivel mare de defecte

Testarea regresiei reprezinta un candidat ideal pentru Automatizare.

6. Validarea produselor software

Negrei Alexandru 443A

6.1. Notiuni generale [7],[10]

"Verificarea si validarea reprezinta procesele prin care se stabileste daca un produs software indeplineste cerintele initiale si indeplineste scopul pentru care a fost dezvoltat." Impreuna aceste doua procese se regasesc sub denumirea controlul calitatii software. Este important de stiut ca verificarea si validarea nu sunt acelasi lucru, cu toate ca adese ori sunt confundate. Pe scurt, verificarea asigura ca produsul a fost dezvoltat corect , pe cand validarea confirma daca produsul indeplineste sau nu scopul pentru care a fost dezvoltat.

Un produs este considerat validat daca satisface nevoile consumatorului. Validarea presupune asigurarea ca procesele de dezvoltare si verificare duc la realizarea unui produs ce respecta necesitatile si specificatiile initiale.

In cazul aparitiei unei probleme in cadrul procesului de dezvoltare sau verificarea, va fi necesara modelarea problemei folosind procedurile de validare (ce presupun si simulari) pentru a evita procesele invalide sau incomplete de dezvoltare/verificare. „Exista si proceduri de validare auxiliare care asigura ca o data ce a fost modelata problema initiala, produsul finit va respecta cerintele initiale."

Pe scurt , validarea poate fi definita prin intrebarea "Construiesc produsul potrivit ?".

6.2. Tipuri de validari

1) validare in perspectiva = activitatile efectuate inainte de lansarea de noi produse pentru a se asigura respectarea conditiilor initiale (legislative/propuse/etc) de catre caracteristicile produsului

2)validarea retrospectiva = proces utilizat la produse aflate deja pe piata ce presupune compararea specificatiilor scrise cu cele asteptate in cadrul etapei de dezvoltare. „Este necesar daca lipseste validarea in perspectiva sau au aparut modificari in legislatie ce fac referire la specificatiile produsului."

3)validarea la scara larga

4)validare partiala = procedeu folosit atunci cand timpul este o problema

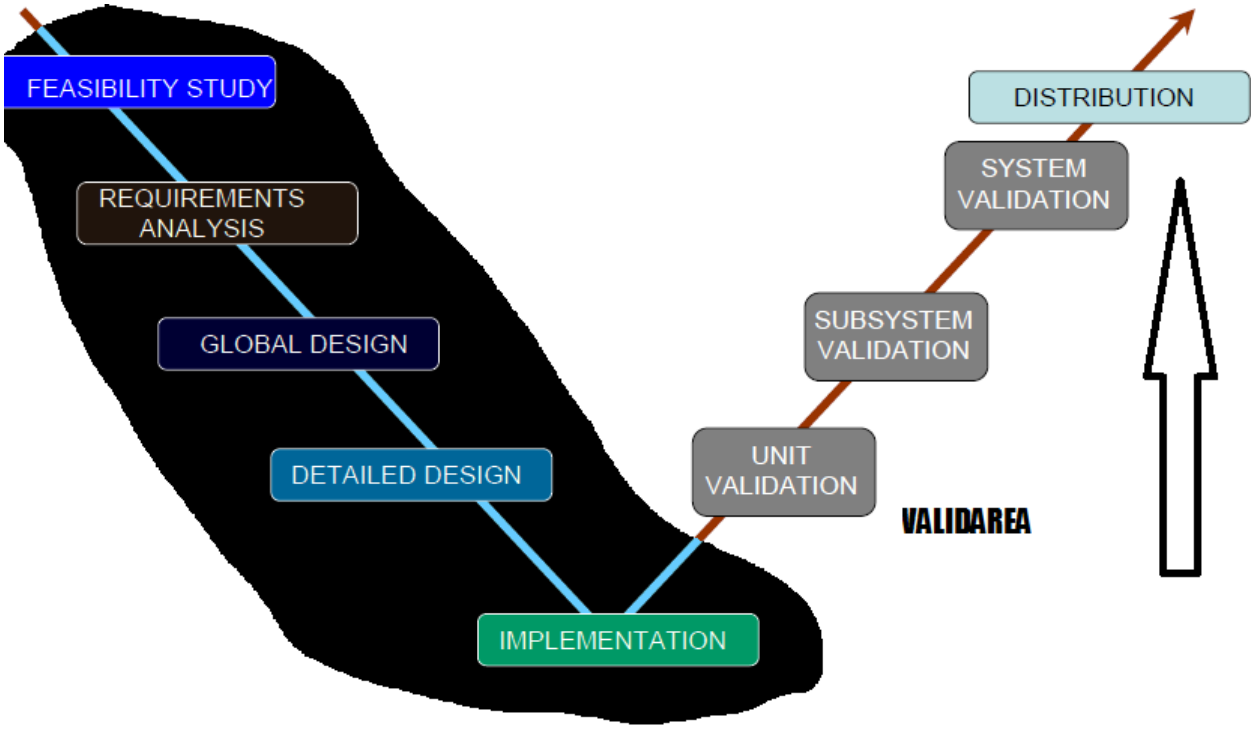
5)validare incrucisata

6) revalidare = necesar la produsele ce trebuiesc revizuite in timp

7)validare curenta = proces de validare ce are loc in acelasi timp cu procesul de fabricatie

6.3. Aspectele validarii

Procesul de validare include testatare pe unitate, testarea pe modul, testatarea pe componenta, testarea subsistemului, testarea integrabilitatii, testarea sistemului si testarea acceptarii.



[Software Engineering for Outsourcing and Offshoring-Bertrand Meyer,Peter Kolb ; 2006-2007]

Cele mai frecvent testate atribute in cadrul procesului de validare sunt :

1)selectivitatea

2)precizia

3)repetabilitatea

4)posibilitatea de reproducere

7)integrabilitatea

6.4.Principiile validarii [8],[9]

Cerintele specificate trebuie sa fie si sub forma de document scris. Fara acesta din urma procesul de validare nu poate fi finalizat.

Testele folosite pentru validare trebuie sa fie efectuate si preventiv, nu numai dupa ce a fost deja scris codul. Este important sa se previna introducerea de greseli in sursa.

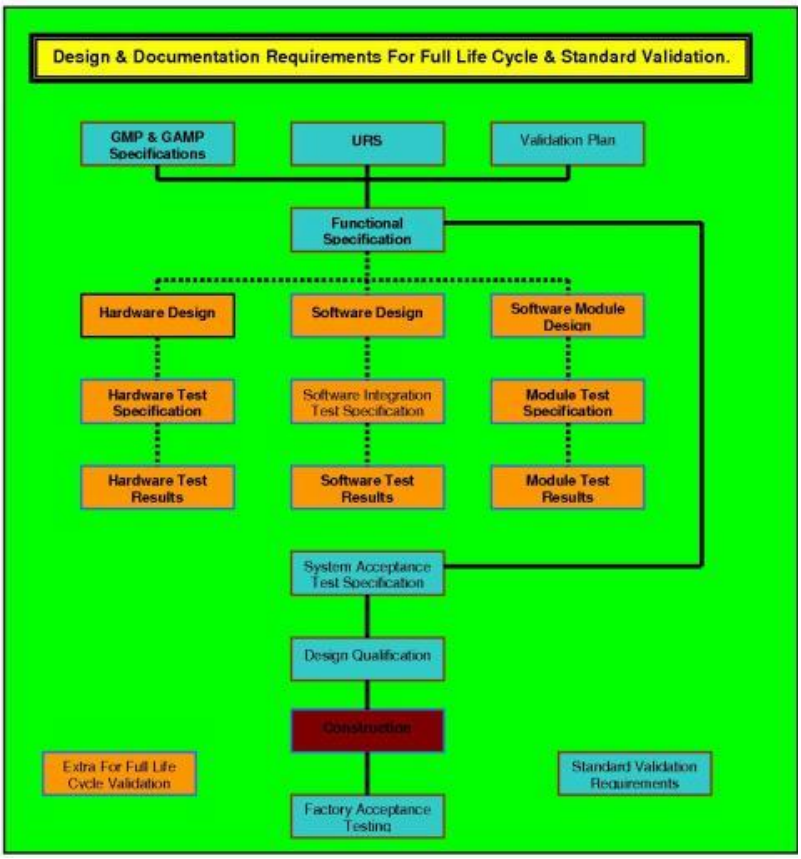
Validarea necesita timp si efort considerabil , astfel este preferabil ca pregatirile necesare acestui proces sa inceapa chiar din etapa de dezvoltare. „Decizia finala de validare trebuie sa poata fi luata in functie de date/statistice obtinute prin eforturi planificate.”

Cand un produs software isi atinge ciclul de viata, procesul de validare trebuie sa fie reluat deoarece specificatiile care trebuiesc respectate se pot modifica in timp.

Folosirea unui plan este esentiala. Acesta defineste ce anume se va realiza prin efortul de validare a softului. In plan sunt specificate scopul, abordarea, resursele ,etc.

Procesul propriu-zis de validare este realizat prin proceduri. Acestea stabilesc cum anume se va desfasura efortul de validare a softului. Procedurile ar trebui sa identifice actiunile specifice sau secventele de actiuni specifice care trebuies efectuate pentru a indeplini activitatile de validare.

[http://www.validation-online.net/21-CFR-Part-11-requirements.html]



Validarea pentru intregul ciclul de viata al produsului

Rezultatul estimarii riscurilor unei validari determina daca produsul software necesita sau nu o validare pentru intregul ciclu de viata. In acest efortul pentru validare este unul foarte mare deoarece trebuie evaluat modul in care softul a fost dezvoltat pentru a determina daca odata finalizat indeplineste cerintele specificatiilor initiale pentru care a fost gandit.

Cand un produs software isi atinge ciclul de viata, procesul de validare trebuie sa fie reluata deoarece specificatiile care trebuiesc respectate se pot modifica in timp.

Folosirea unui plan este esentiala. Acesta defineste ce anume se va realiza prin efortul de validare a softului. In plan sunt specificate scopul, abordarea, resursele ,etc.

Procesul propriu-zis de validare este realizat prin proceduri. Acestea stabilesc cum anume se va desfasura efortul de validare a softului. Procedurile ar trebui sa identifice actiunile specifice sau secventele de actiuni specifice care trebuies efectuate pentru a indeplini activitatile de validare.

„Pana si cea mai mica modificare a unui soft poate avea un impact global asupra produsului. Astfel este necesara o o revalidare nu doar pentru parte modificata ci pentru noul asamblu software .”

Validarea este destul de greu de realizat chiar de firma care a dezvoltat programul. De obicei se apeleaza la firme specializate pe validari de produse software.

Cu toate ca nu participa la validarea propriu-zisa , este prezenta si o echipa QA (quality assurance), care este prezenta pentru a se asigura ca in cadrul procesului de validare sunt folosite cele mai indicate practici in industrie. Datoria QA este de a aduce clientului nu doar un produs validat ci cel mai bun produs validat posibil.

[11]

6.5.Viziuni filozofice asupra validarii

Din punct de vedere logic viziunea pozitiva este:”exista o lume obiectiva care poate fi modelata construind o entitate de cunostinte bazate pe observatii empirice”. In cazul validarii software se refera la exista o problema obiectiva . Aceasta problema se rezolva : construind un model consistent facand suficiente observatii empirice pentru a verifica validitatea,folosirea “uneltelor ” pentru a testa consistenta si cat de complet este modelul, prin folosirea de recenzii,prototipuri pentru a demonstra ca modelul este valid.

Popper a adus urmatoarea modificarea la viziunea de mai sus : “teoriile nu pot fi dovedite corect, pot fi doar respinse gasind exceptii”. La validare trebuie sa gasim dovezi ca modelul este gresit prin colectarea de scenarii , verificand ca modelul le suporta.

Viziunea post modernista este : “nu exista un punct de vedere privilegiat;toate observatiile sunt incarcate cu valoare;investigatia stiintifica este integrata din punct de vedere cultural”. In cazul validarii putem trage concluzia ca este intotdeauna subiectiva si bazata pe context. Astfel trebuiesc implicate partile interesate , ca modelul cerintelor sa le apartina . De asemenea trebuiesc folosite tehnici etnografice pentru a intelege impactul asupra lumii.

6.6.Concluzie

Validarea software este un proces critic, folosit pentru a asigura calitatea produsului software. Prin validare se imbunatateste aplicabilitatea practica si fiabilitatea software-ului, rezultatul fiind rata mica de erori , mai putine actiuni de corecate a codului sursa.resulting. Validarea software poate reduc costurile pe termen lung, facand posibil ca modificarile fiabile software si eventualulele revalidari sa se execute cu costuri scazute. In conditiile in care validarea initiala lipseste costurile de intretinere al produsului software sunt considerabil mai ridicate. O validare initiala bine planificata si amanuntita ajuta la reducerea costurilor pe termen lung , prin reducerea costurilor necesare revalidarilor pentru versiunile urmatoare ale produsului software.

7. Analiza statistica si analiza fluxului de date

Teodorascu Paula 443A

7.1 Introducere In Analiza Statistica

„Analiza statistica studiaza colectarea, organizarea si analizarea unui grup de date, in scopul obtinerii unor rezultate asupra unor produse.”[25] Analiza statistica pentru produse software presupune analizarea impactului pe care un produs software il are asupra utilizatorilor lui.

„Unii considerea statistica, ca pe o componenta a matematicii, o stiinta care colectioneaza, analizeaza, interpreteaza si explica un grup de date si comportatea acestora. „ [25] Dar altii, deoarece are radacini empirice si este concentrata pe aplicatii, considera statisticile a fi o stiinta matematica separata decat o ramura a matematicii.

7.1.1 Scopul Analizei Statistice

Statistica se ocupă de obținerea de informații relevante din datele disponibile într-un volum suficient de mare. Informațiile pot fi folosite pentru a înțelege datele disponibile (statistică descriptivă) sau pentru a descoperi noi informații despre evenimente și relațile dintre ele (statistică inferențială).

Procesul de obținere a informației din date se numește inferență statistică referitoare la unii parametri statistici, sau chiar întregii distribuții probabilistice. Acesta este punctul de vedere mai general adoptat de teoria neparametrică în statistică. În statistica aplicată clasică este preferată ideea de a construi un model statistic cu care se pot face inferențe; în majoritatea cazurilor acest model nu este verificat, ceea ce poate conduce la concluzii eronate. Statistica aplicată modernă analizează însă date mult prea complexe, cum ar fi imagini sau structura proteinelor, pentru a se putea mărgini la ideea de modelare.

Conceptele de bază ale statisticii matematice sunt frecvența, absolută sau relativă, mărimea statistică, legea de repartiție sau de distribuție a unei mărimi, variabilitatea sau stabilitatea unei mărimi, corelațiile sau conexiunile între două caracteristici sau două mărimi statistice, indicii statistici. De asemenea, se folosesc conceptele de variabilă aleatoare, câmp de probabilitate sau câmp statistic, valoare medie, abatere, abatere pătratică.

„Teoriile folosite în statistică au la bază teoria probabilităților.”[25] Statistica matematică folosește diferite teorii, ca teoria selecției (cu teoria estimației, a testelor de semnificație), teoria controlului statistic (mai ales în industrie), teoria deciziilor cu analiza secvențială și teoria predicției, legată de teoria proceselor stohastice.

7.1.2 Descriere - Metode - Termeni cheie

Atunci cand vrem sa analizam statistic o aplicatie sau un produs software trebuie sa dispunem de o colectie de date, ce privesc direct aplicatia, date de intrare, de iesire, date obtinute in timpul rularii produsului software, si folosite pe post de date de intrare pentru o alta ramura a aceluiasi produs, etc. Aceasta colectie de date se analizeaza din punct de vedere al timpului de rulare a produsului software pana la obtinerea datelor de iesire dorite, din punctul de vedere a resurselor folosite de catre produsul software, din punct de vedere a eficientei cu care se obtine setul de date de iesire.

Activitatile uzuale pentru realizarea unei analize statistice sunt:

- introducerea datelor de lucru, si anume definirea variabilelor, introducerea datelor statistice optime, importarea datelor din surse externe.
- operarii de rearanjare a dateleor in vederea prelucrarii acestora de catre produsul software
- executia functiilor de prelucrare ale produsului software
- interpretarea rezultatelor obtinute si realizarea statisticilor

Metode de analiza statistica:

- studii experimentale si observationale: obiectivul principal al analizei statistice este investicarea cauzalitatii si in particulat tragerea unor concluzii pe baza schimbarilor survenite asupra variabilelor. „Exista 2 tipuri majore de metode analitice: studii experimentale si sutdii observationale, in cazul ambelor tipuri de studii, este subliniat efectul produsului software asupra setului de date pe care il ale de prelucrat.”[20] Etapele unui experiment sunt: planificarea studiului, inclunzand alegerea seturilor de date pe care se va face experimentul, pasul urmator presupune alegerea tipul de test care va fi efectuat asupra produsului software, urmeaza efectuarea testului pe seturile de date alese anterior, examinarea rezultatelor obtinute, iar in final crearea documentatiei necesare pentru prezentarea rezultatelor studiului.
- nivele de masura: exista 4 nivele principale de masura utilizare in statistica: nominal, ordinal, interval si ratie. „Acestea au grade diferite de utilizare in analiza unui prosus.” [24]

Termeni cheie utilizati in statistica:

- ipoteza nula: se presupune ca orice etape de prelucrare prin care trec datele de intrare nu au nici un efect asupra acestora.
- eroare: atunci cand datele prelucrare nu au valori cuprinse in normele prevazute de produsul software, sau atunci cand ipoteza nula este rejectata din cauza unui „fals pozitiv”.

7.2 Analiza Fluxului de date

7.2.1 Introducere

Analiza fluxului de date este o tehnica de culegere de informatii despre un posibil set de valori calculate intr-un punct anume de timp cu ajutorul unui produs software. Graful fluxului de control(control flow graph- CFG) al unui program este utilizat pentru a determina acele parti ale programului care sunt folosite pentru atribuirea unor variabile. Informatiile astfel culese sunt de cele mai multe ori utilizate de catre compilatoare pentru optiizarea produsului software. Unexemplu canonic pentru analiza fluxului de date sunt definitiile posibile pentru o anumita valoare(reaching definition).

O cale simpla de a face „analiza a unui flux de date este de a atribuii ecuatii fiecarui nod de control din graf”[20] si de a le rezolva calculand repetat iesirea de la intrarea locala pentru fiecare nod pana cand tot sistemul se stabilizeaza. Aceasta abordare a fost dezvoltata de catre Gary Kildall.

Graful de fux de control al programului este o reprezentare in care:

- nodurile sunt instructiuni
- muchiile indica secventierea instructiunilor (inclusiv salturi)
- putem avea: noduri cu:
    - un singur succesor (ex. atribuiri)
    - mai multi succesori (instructiuni de ramifcatie)
    - mai multi predecesori (reunirea dupa ramifcatie)

7.2.2 Principiu

Analiza fluxului de date este procesul de colectare a informatiilor despre cum sunt folosite variabilele definite intr-un program/produs software. Aceasta analiza incearca sa obtina informatii particulare despre fiecare punct al procedurii. Cel mai usor mod este de a obtine informatii de la granitele blocurilor de baza ale produsului, deoarece de acolo este cel mai usor de calculat informatiile. In analiza inainte, starea de iesire a unui bloc este o functie de starea de intrare a acelui bloc. Aceasta ipoteza da nastere unor ecuatii:

„$out_b = trans_b(in_b)$
$in_b = join_{p€pred_b}(out_p)$” [21]

Unde: $trans_b$ este functia de transfer a blocului b ce actioneaza asupra starii de intrare in_b si scoate la ierisrea blocului starea out_b. Operatia de join combina la iesirea blocului toti predecesorii p ce apartin $pred_b$ ai blocului b.

Dupa rezolvarea acestui set de ecuatii, starile de intrare sau iesire ale blocului pot fi folosite pentru a extrage proprietatile granitelor blocurilor programului. Aceasta functie de transfer poate fi aplicata pentru a extrage informatii in fiecare punct al unui bloc de baza.

Proprietati analizate (dataflow facts):

Concret: se analizeaza diferite proprietati:

- valoarea unei variabile intr-un punct de program
- sau intervalul de valori pentru o variabila

- sau multimi de variabile (live), expresii (available, very busy), definitii posibile pentru o valoare (reaching definitions), etc.

Abstract: o multime D de valori pentru o proprietate (dataflow facts). Restrictie: D este o multime fnita.

Clasifcare a analizelor fluxurilor de date:

- inainte sau inapoi
- must sau may
- dependente sau independente de fuxul de control (fow (in)sensitive):
  - Trebuie luata in considerare ordinea instructiunilor in program
  - nu: ce variabile sunt folosite/modifcate, functii apelate, etc.
  - da: proprietati legate de valorile calculate efective de program
- dependente sau independente de context: in cazul programelor ce contin proceduri: e specializata analiza fecarei proceduri in functie de locul de apel, sau se poate face un sumar (o analiza comuna)
- dependente sau nu de cale (path-(in)sensitive): tine cont de corelarile pe caile individuale de executie

7.2.3 Analiza Inainte - Analiza Inapoi

Tipurile de analiza a fluxului de date ce urmeaza a fi prezentate sunt exemple de proprietati ale programelor ce pot fi calculate cu ajutorul analizei. Aceste proprietati calculate cu ajutorul analizei fluxului de date sunt de cele mai multe ori aproximari ale proprietatilor adevarate. Acest lucru este datorat faptului ca analiza fluxului de date opereaza asupra structurii sintactice ale CFG fara a simula exact controlul fluxului programului.

Analiza Inainte:

Analiza definitiilor posibile pentru o valoare calculata pentru fiecare punct din program indica setul de definitii care pot aparea in acel punct al programului.

Exemplu:

```
„1: if b==4 then
2:    a = 5;
3: else
4:    a = 3;
5: endif
6:
7: if  a < 4 then
8:    ...” [21]
```

Definitia posibila pentru variabila a de la linia 7 este setul de atribuiri a=5 de la linia 2 si a=3 de la linia 4.

Analiza Inapoi:

Analiza variabilei „In Viata” calculeaza pentru fiecare punct din program variabila care poate fi citita inainte de urmatorul update. Rezultatul este de obicei utilizat de

eliminarea codului „mort” pentru a sterge declaratiile a caror valoare nu este folosita mai departe.

Starea de iesirea a unui bloc este setata de variabilele care sunt „in viata” la finalul fiecarui bloc. Starea de intrare este data de variabilele care sunt „in viata” la momentul acela., iar starea de iesire este data de uniunea de stari de intrare de la blocurile succesoare.

Ecemplu:

Codul initial:

```
„b1: a = 3;
        b = 5;
        d = 4;
        if a > b then
b2:    c = a + b;
        d = 2;
b3: endif
        c = 4;
        return b * d + c;” [21]
```

Analiza Inapoi:

```
„// in: {}
b1: a = 3;
    b = 5;
    d = 4;
    x = 100; //x is never being used later thus not in the out set
{a,b,d}
    if a > b then
// out: {a,b,d}    //union of all (in) successors of b1 => b2:
{a,b}, and b3:{b,d}

// in: {a,b}
b2: c = a + b;
    d = 2;
// out: {b,d}

// in: {b,d}
b3: endif
    c = 4;
    return b * d + c;
// out:{} „ [21]
```

Stare de iesire a b3 contine b si d, din moment ce c a fost scris. Starea de intrare a b1 este uniunea lui b2 cu b3. Definitia lui c in b2 poate fi stearsa, din moment ce acesta nu este „in viata” imediat dupa declaratie.

Rezolvarea fluxului de date a ecuatiilor incepe cu initializare tuturor starilor de intrare si a tuturor celor de iesire cu cate un set gol. Lista de lucru este initializata prin introducerea punctului de iesire b3. Este calculata starea de iesire ce difera de cea anterioara, astfel incat predecesorii b1 si b2 sunt inserati si procesul continua. Procesul este sumarizat in tabelul de mai jos:

processing	**in-state**	**old out-state**	**new out-state**	**work list**
b3	{}	{}	{b,d}	(b1,b2)
b1	{b,d}	{}	{}	(b2)
b2	{b,d}	{}	{a,b}	(b1)
b1	{a,b,d}	{}	{}	()

[21]

Se observa ca b1 a fost introdus in lista inaintea lui b2, ceea ce forteaza procesarea lui b1 de 2 ori. Inserarea lui b2 inaintea lui b1 ar fi permis finalizarea mai devreme.

8. Bibliografie

[1] http://en.wikipedia.org/wiki/Boundary_value_analysis

[2] http://www.aptest.com/testtypes.html

[3] http://www.softwaretestinghelp.com/types-of-software-testing/

[4] http://www.buzzle.com/articles/types-of-software-testing.html

[5] IEEE Std 829™-2008 Standard for Software and System Test Documentation

[6] Matt Heusser – Fundamental Strategies in Software Testing

[7] http://en.wikipedia.org/wiki/Verification_and_validation

[8]http://www.fda.gov/RegulatoryInformation/Guidances/ucm126954.htm#_Toc517237945

[9] http://www.fda.gov/RegulatoryInformation/Guidances/ucm126954.htm

[10] http://en.wikipedia.org/wiki/Verification_and_Validation_ (software)

[11] http://www.cs.toronto.edu/~campbell/340/05w/utm/lectures/19-VandV4-up.pdf

[12] Ian Sommerville - *Software Engineering*

[14] http://en.wikipedia.org/wiki/Software_testing

[15] http://www.softwaretestinghelp.com/application-testing-%E2%80%93-into-the-basics-of-software-testing/

[16] http://www2.sas.com/proceedings/sugi30/141-30.pdf

[17] http://www.softwaretestinggenius.com/articalDetails.php?qry=970

[18] http://www.universalexams.com/category/general-software-testing

[19] http://www.softwaretestinggenius.com/articalDetails?qry=826

[20] http://bigfoot.cs.upt.ro/~marius/curs/vf/curs9.pdf
[21] http://en.wikipedia.org/wiki/Data-flow_analysis

[22] http://www.cis.upenn.edu/~cis570/slides/lecture04.pdf

[23] http://www.itc.nl/~rossiter/teach/stats/sintro.pdf

[24] Wiley, J. – Software Testong and QA , eBook, 2008

[25] http://en.wikipedia.org/wiki/Statistics

[26]http://www.xsim.com/bib/index1.d/Index.html

[27]Branch Coverage for Arbitrary Languages Made Easy,Ira. D. Baxter, CTO, Semantic Designs, Inc.

[28]http://www.bullseye.com/coverage.html

[29]Beck, K. Test-Driven Development by Example, Addison Wesley - Vaseem, 2003

[30]Jonathan B. Rosenberg, How Debuggers Work: Algorithms, Data Structures, and Architecture, John Wiley & Sons

[31]Castor, Kevin (2006). "Hardware Testing and Benchmarking Methodology". Retrieved 2008-02-24.

[32]http://www.appperfect.com/products/application-testing/app-test-gui-testing.html

[33]http://en.wikipedia.org/wiki/Program_animation

Referinte Capitolul 2

1 - http://news.google.com/newspapers?nid=1948&dat=19890425&id=PtAqAAAAIBAJ&sjid=SNYFAAAAIBAJ&pg=1028,3615284
2 - http://www.independent.co.uk/news/uk/software-failure-may-be-behind-ambulance-crisis-1560323.html
3 – http://www.softwaretestinggenius.com/articalDetails?qry=826