Metodologie de testare a erorilor fizice şi umane pentru un produs software

Exemplele de eşecuri/defectari software sunt mai multe, printre ele se numără:

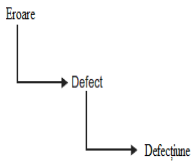
• Prima lansare a rachetei Agenţiei Spaţiale Europene din iunie 1996, care a eşuat după 37.5 sec. O eroare software a cauzat devierea rachetei de la ascendența sa verticală şi capacităţile de auto-distrugere au fost activate ducând la o problemă şi mai mare.

În noiembrie 2005, în Statele Unite un site web a dezvăluit informaţii despre top 10 cei mai căutaţi criminali iar această poveste a fost redată la ştiri şi la radio, oferind astfel publicitate site-ului web, ducând la un număr impresionant de accesări care a atras după site performanțe disfuncţionale, întrucât site-ul nu fusese testat la o rată atât de mare a traficului.

Toate aceste exemple au în comun disfuncţionalitatea datorată lipsei rigurozității de testare

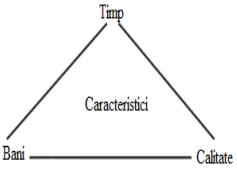
În figura de mai jos sunt prezentate cele 3 resurse care trebuiesc luate în consideratie la realizarea unui produs software.

Figura 1.1 Efectul unei erori



Defectele software se manifestă ca rezultat al următorului proces: un programator comite o **eroare** (greșeală), care la rândul ei rezultă într-un defect (bug) la nivel de codul sursă al programului; dacă acest defect este executat, în anumite condiţii sistemul va produce un rezultat greşit, ceea ce ulterior va duce la o eşuare a programului.[1]

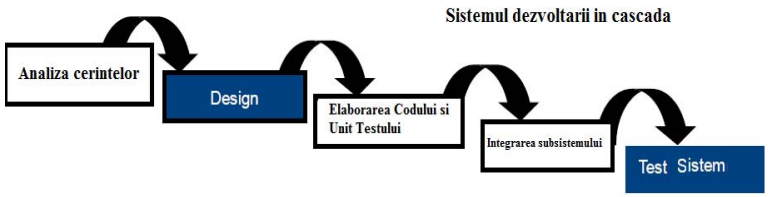
Figura 1.2 Triunghiul resurselor



Paşii realizării proiectelor software

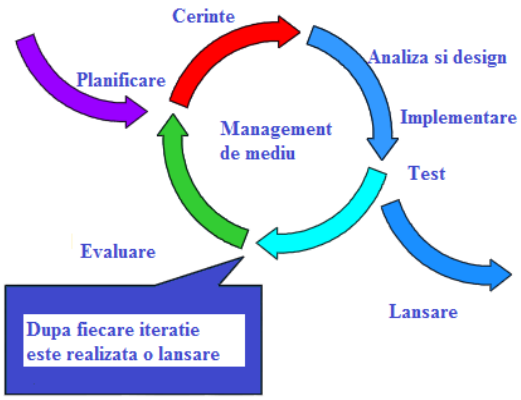
Rolul procesului de testare este de a asigura examinarea funcţionalităţilor înainte ca sistemul să fie folosit, şi orice defectare să fie raportată echipei de dezvoltare pentru a fi rezolvată. Testarea nu poate înlătura direct defectele, şi nu poate nici îmbunătăţi calitatea, dar prin raportarea defectelor descoperite, acestea pot fi înlăturate crescand astfel fiabilitatea sistemului.

Practici în dezvoltarea software Dezvoltarea iterativă



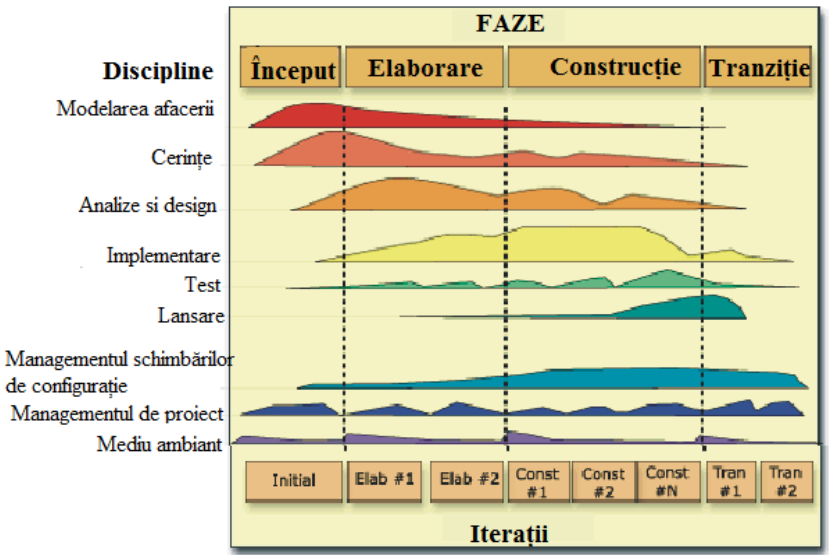
3

Dezvoltarea in cascada poate fi şi mai bine prezentată în figură următoare care pune în evidenţă etapele executabile ale dezvoltării proiectului.



Sistemul executabil al dezvoltării iterative

4



Fazele dezvoltării unui proiect software

5

Realizarea testării unui produs software – principii generale

- Un test consta in executia programului pentru un set de date de intrare convenabil ales, pentru a verifica daca rezultatul obtinut este corect
- Un caz de test este un set de date de intrare impreuna cu datele de iesire pe care programul ar trebui sa le produca.
- Testarea este activitatea de conceptie a cazurilor de test, de executie a testelor si de evaluare a rezultatelor testelor, in diferite etape ale ciclului de viata al programelor.
- Scopul primordial pentru procesul de testare este (1) identificarea erorilor de software, (2) izolarea și fixarea/corectarea defectelor care au cauzat aceste erori. Deoarece testarea nu poate demonstra cu certitudine de 100% ca produsul funționează corect în orice condiții; testarea doar poate demonstra că produsul nu funcționează corect în anumite condiții. În scopul testării deseori se include atât examinarea statică a codului sursă, cât și examinarea codului în execuție în diferite împrejurări și sub diferite condiții. Aspectele de cod ce rămân sub ochiul vigilent al test/software inginerului în timpul acestui exercițiu sunt (1) codul face lucrul care este presupus sa-l facă și (2) codul face lucrul care trebuie sa-l facă. Informația obținută în urma procesului de testare poate fi folosită pentru corectarea și îmbunătățirea procesului conform căruia se dezvoltă produsul software.[1]

6

EVOLUTIE

1945 - 1956 - Teste orientate pe depanare-(Debugging oriented)

Primul cercetator care s-a ocupat de noțiunea de testare a unui program este Alan Turing care publică în 1949 un articol despre principiile teoretice ale verificării corectitudinii funcționării unui program. În 1950, Turing publică un alt articol, în care ridică problema inteligenței artificiale, și definește un test pe care sistemul informatic trebuie să îl treacă, și anume răspunsurile oferite de către acesta la întrebările adresate de un operator (testerul) să nu poată fi diferențiate de către răspunsurile oferite de un om (etalonul). Această lucrare poate fi considerată a fi prima care se ocupă de conceptul de testare, considerat distinct de activitățile de elaborare a codului respectiv de depanare.

1957–1978 – Teste orientate pe demonstrare functionalitatii (Demonstration oriented)

Testarea pe scară largă a devenit necesară datorită creșterea impactului economic pe care defectele nedetectate le puteau avea. De asemenea, persoanele implicate în dezvoltarea de programe informatice au devenit mai conștiente de riscurile asociate cu defectele din programe și au început să pună mai mult accent pe testarea și remedierea defectelor înainte ca acestea să afecteze produsul livrat. În această perioadă, termenii de testare și depanare se suprapuneau și se refereau la eforturile făcute în scopul descoperirii, identificării și remedierii defectelor din sistemele informatice. Scopul procesului de testare era demonstrarea corectitudinii funcționării programului, adică absența erorilor

1979–1982 - Orientare spre defectare (Destruction oriented)

In 1979. Glenford J. Myers face distincția dintre depanare care este o activitate care ține de dezvoltarea programului și testarea care este activitatea de rulare a unui program cu scopul declarat de a descoperi erori. El susține că în testarea bazată pe demonstrație există pericolul ca operatorul să aleagă în mod inconștient acel set de parametri care ar asigura funcționarea corectă a sistemului, ceea ce creează pericolul ca multe defecte să treacă neobservate. Myers propune în abordarea sa și o serie de activități de analiză și control care împreună cu procesul de testare să crească nivelul de calitate a sistemelor informatice.

1983 - 1987 - Orientarea spre evaluare (Evaluation oriented)

instituțiile americane ANSI și IEEE încep elaborarea unor standarde care să formalizeze procesul de testare, efort concretizat în standarde precum **ANSI IEEE STD 829**, în 1983, care stabilea anumite formate care să fie utilizate pentru crearea documentației de testare.

1988 - în prezent - Orientarea spre prevenire– Prevention oriented

Testarea trebuie făcută în toate fazele de lucru, în paralel cu programarea și are următoarele activități principale: planificare, analiză, proiectare, implementare, execuție și întreținere. Respectarea acestei metodologii duce la scăderea costurilor de dezvoltare și de întreținere a unui sistem prin scăderea numărului de defecte care ajung nedetectate în produsul final

7

Metode de Testare

Functional Testing	Testarea in vederea verificarii programului folose□te date de test alese de participantii la procesul de dezvoltare a programului. Ea este efectuata la mai multe nivele: la nivel de unitate functionala, de modul, de subsistem, de sistem.
Black box Testing	Nu se bazează pe cunoașterea interna a design-ului sau a codului. Testele sunt bazate pe cerințe și funcționalitate
White box Testing	Se bazează pe cunoașterea logicii interne a codului aplicației. Testele sunt bazate pe acoperirea sintaxei de cod, ramuri, cai, condiții

Testarea in vederea validarii programului, numita si testare de acceptare, are drept scop stabilirea faptului ca programul satisface cerin□ele viitorilor utilizatori. Ea se efectueaza în mediul in care urmeaza sa functioneze programul, deci folosindu-se date reale. Prin testarea de acceptare pot fi descoperite si erori, deci se efectueaza si o verificare a programului

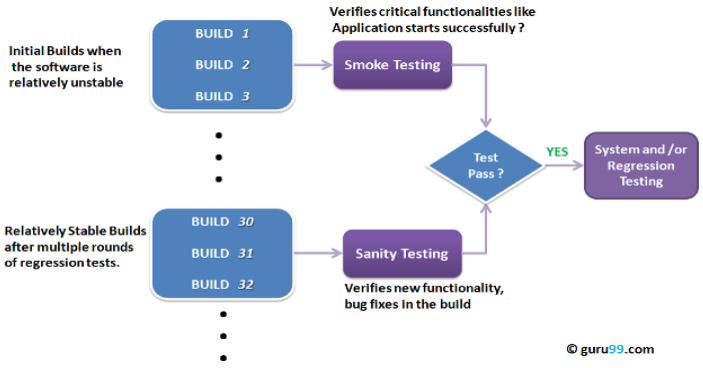
8

Testarea Funcțională

- **Unit Testing** — Prima treapta a testării; se testează funcțiile sau module de cod. De obicei sunt făcute de programatori deoarece presupun cunoștințe avansate a design-ului intern al aplicației și codului.
- **Integration Testing** — Testare combinată a diferitelor părți dintr-o aplicație pentru a vedea dacă funcționează corect. Aceste "parți" pot fi module de cod, aplicații individuale, aplicații client-server într-o rețea etc.
- **System Testing** — Testare de tip Black-box bazata pe specificații care acoperă toate părțile sistemului
- **User Acceptance Testing** — Testare prin care se determina dacă software-ul este satisfăcător pentru un utilizator final sau client.
- **Usability Testing** — Testare prin care se determină dacă un software este "user-friendly" (prietenos pentru utilizatorul final. Evident aceasta testare este subiectivă și va depinde de utilizatorul sau clientul vizat. Interviurile cu utilizatorii, sondajele, monitorizarea sesiunilor utilizatorilor sunt metode care pot fi folosite.

9

- **Install/Uninstall Testing** — Testare parțială sau integrală a procesului de instalare sau upgrade
- **End-to-end Testing** — Este similară cu System testing; este o testare de nivel 'macro'; presupune testarea aplicației în mediul în care va fi folosită
- **Sanity Testing sau Smoke Testing**



10

- **Regression Testing** — Reprezintă acțiunea de re-testare după corecții sau modificări ale software-ului sau a mediului său de dezvoltare. Poate fi dificil de a determina cât de mult este nevoie de re-testare, în special, aproape de sfârșitul ciclului de dezvoltare. Instrumentele automate de testare pot fi utile, în special, pentru acest tip de testare
- **Compatability Testing** — Testare finală pe baza specificațiilor utilizatorul final sau clientului, sau pe baza utilizării de către aceștia pentru o perioadă limitată de timp
- **Acceptance Testing** — Se testează cât de bine software-ul funcționează într-o anumită configurație de hardware / software / sistem de operare / rețea / etc

11

Testarea Non-Funcțională

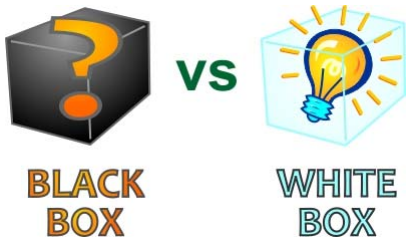
- **Performance Testing** — Acest tip este adesea folosit în mod alternativ cu testarea de "stres" sau testarea de "încărcare".
- **Stress Testing** — Folosit pentru a descrie testarea funcțională a sistemului în timp, sub încarcări neobișnuit de mari, repetarea în mai multe cicluri a unor anumite acțiuni sau intrări, inserarea de valori numerice mari, interogări complexe și mari la un sistem de baze de date, etc
- **Load Testing** — Testare a unei aplicații în sarcini mari, cum ar fi testarea unui site web sub o serie de sarcini pentru a determina în ce moment timpul de răspuns a sistemului se degradează sau nu
- **Recovery Testing** — Testare pentru a se vedea cât de bine un sistem se recuperează de la accidente, erori de hardware, sau alte probleme catastrofice
- **Failover Testing** — Acest tip de testare validează capacitatea unui sistem de a fi în măsură să aloce resurse suplimentare și să mute operațiunile pe sisteme back-up
- **Security Testing** — Testare pentru a se evidenția cât de bine sistemul este protejat împotriva accesului intern sau extern neautorizat, deteriorarea intenționată, etc.; poate necesita tehnici de testare sofisticate

12

Alte metode de testare

- **Exploratory Testing** — O testare informală, care nu se bazează pe planuri de testare oficiale sau cazuri de testare. În această situație specialiștii în testare pot învăța software-ul în timp ce-l testează
- **Ad-hoc Testing** — Testare similară cu cea exploratoare, dar specialiștii în testare nu cunosc produsul software în amănunt
- **Context-driven testing** — Testare efectuată din nevoia de a înțelege mediul, cultura, și destinația de utilizare a software-ului. De exemplu, abordarea de testare pentru software-ul pentru echipamentele medicale de urgență ar fi cu totul diferită de cea pentru un joc pe calculator cu buget redus
- **Comparison Testing** — Testare ce constă în compararea punctele slabe și punctele forte ale software-ului de la produsele concurente
- **Alpha Testing** — Testare a unei aplicații atunci când dezvoltare sa se apropie de finalizare, modificări minore de design pot fi încă făcute ca urmare a unei astfel de testare. Se efectuează de obicei de către utilizatorii finali sau alte persoane, și nu de programatori sau specialiști în testare.
- **Beta Testing** — Testare când dezvoltare și testare sunt, în general, complete și bug-uri finale și problemele trebuie găsite înainte de produsul final. Se efectuează de obicei de către utilizatorii finali sau alte persoane, și nu de programatori sau specialiști în testare.
- **Mutation Testing** — Este o metodă pentru a determina dacă un set de date de testare sau de cazuri de testare este util, prin introducerea în mod deliberat diferite modificări de cod ("bug-uri") și retestarea cu date de test / cazuri originale, pentru a determina dacă sunt detectate de "bug-uri".

- În prezent, testarea se realizează utilizând fie strategia black box (când se urmareşte numai comportamentul funcţional al produsului), fie strategia white box (se analizează comportamentul programului având la dispozitie codul sursă al acestuia).



- Iniţial, testarea se facea integral manual, fiind considerată singura soluţie de a descoperi eventualele defecte.
- Testele automate execută practic o secvenţă de acţiuni cu intervenţie umană minimă şi pot simula utilizarea unei aplicaţii în condiţii reale, de simultaneitate ridicată.

14

Testarea Manuală vs Testarea Automată

AVANTAJE	
Testare Automată	Testare Manuală
• Daca trebuiesc repetate aceleași teste de multe ori automatizarea prezintă un mare avantaj	• Daca Test Case-urile trebuiesc rulate de un număr mic de ori e mult mai probabil sa se prefere testarea manuala
• Ajuta la executarea de teste de compatibilitate a programelor create pe mai multe configurații	• Permite tester-ului să facă mai multe teste ad-hoc
• Permite executarea scenariilor de testare ajutând la testele de regresie	• Costurile pe termen scurt sunt reduse
• Permite rularea mai multor teste de regresie pe un cod care se schimba des	• Cu cat un tester petrece mai mult timp verificând un modul cu atât cresc șansele de a găsi mai multe defecte și posibile greșeli de utilizare
• Pot fi rulate simultan pe mai multe mașini astfel scăzând timpul de testare	
• Costurile pe termen lung sunt reduse	

Testarea automată - prezentare

- Cele mai importante beneficii sunt:
  - acoperirea tuturor etapelor de testare de la concepţie până la lansare
  - posibilitatea simulării testării cu mai mulţi utilizatori.
  - posibilitatea repetării testelor
  - creşterea siguranţei în produs
- Un sistem de testare automată trebuie să aibă la bază două caracteristici principale:
  - module reutilizabile
  - întreţinerea şi urmărirea activităţii dintr-un singur punct de control

16

Testarea automată

- Într-o abordare mai detaliată testarea automată înseamnă:
  - 1. planificare
    - identificarea cerinţelor şi a funcţionalităţilor.
    - gruparea acestora în condiţii de test.
    - crearea cazurilor de test pentru aceste condiţii.
  - 2. design
    - construcţia scripturilor de test.
    - generarea testelor de rulare.
  - 3. execuţie
    - crearea scenariului de rulare a scripturilor.
    - rularea uneltelor monitor pentru înregistrarea datelor.
    - înregistrarea rezultatelor pentru fiecare rulare.
    - raportarea şi gestionarea bug-urilor.
  - 4. management
    - generarea rapoartelor şi graficelor.
    - controlul dintr-un singur punct de comandă.
    - documentarea permanentă a stadiului curent al proiectului.

17

Tipurile de testare automată

- 1. structurală (white-box testing) - se verifică structura software-ului şi necesită acces complet la codul sursă.
- 2. funcţională (black-box testing) – se definesc aşteptările clientului de la aplicaţie şi se verifică automat dacă software-ul se comportă conform acestor aşteptări
- 3. regresivă (regression testing) - se verifică dacă s-a modificat neaşteptat comportamentul aplicaţiei în urma implementării unor noi cerinţe/schimbări (change requests).
- 4. negativă (negative testing) - se solicită aplicaţia, producând deliberat cazuri complicate,neobişnuite sau particulare pentru a forţa apariţia erorilor.
- 5. de solicitare (stress testing) - se determină capabilităţile absolute ale aplicaţiei şi ale infrastructurii pe care este implementată
- 6. de performanţă (performance testing) - în urma acestui tip de testare se verifică dacă performanţa aplicaţiei este adecvată pentru cerinţele stabilite, în termeni de viteză de acces, resurse de sistem utilizate şi procesarea cererilor de acces.
- 7. de încărcare (load testing) - se determină punctele slabe ale aplicaţiei şi dacă sunt necesare îmbunătăţiri ale infrastructurii hardware sau software prin măsurarea caracteristicilor de performanţă şi scalabilitate a principalelor componente ale aplicaţiei web; de regulă aceasta se realizează prin creşterea numărului de sesiuni utilizator sau a conexiunilor TCP/IP.

18

Unelte pentru sistemele de testare automată

- GUI , prin folosirea metodei "înregistrare/redare".
- analizoare de cod - analizează complexitatea codului scris, respectarea unor standarde de scriere a codului.
- analizoare de memorie - detectează depăşirea memoriei alocate, suprascrieri în zone nealocate şi zone rămase nedealocate.
- testare de solicitare/performanţă - pentru testarea aplicaţiilor web şi client/server în diferite cenarii de solicitare.
- testare servere web - verifică validitatea şi integritatea link-urilor, a codului html, programe client-side şi server-side, securitatea transmiterii datelor.
- alte unelte - pentru managementul documentaţiei, raportării bug-urilor, configuraţiei, etc.

19

Avantajele utilizării testării automate

- 1. avantaje privind eficienţa şi costurile
  - prevenirea erorilor prin abordarea structurată a procesului de dezvoltare a proiectului.
  - detecţia erorilor care au ajuns până în faza de producţie (prin teste de regresie automată).
  - reutilizarea informaţiei acumulate (condiţii de test, scripturi).
  - reducerea creării de noi scripturi (refolosindu-le şi adaptându-le pe cele vechi).
  - execuţia automată a testelor de performanţă în fazele de început ale proiectului poate evita eforturile de redesign în fazele ulterioare.
  - odată ce scripturile de testare automată sunt implementate, o parte din personal poate fi redirecţionat către alte necesităţi.
- 2. avantaje privind economia de timp
  - analiză rapidă şi exactă în cazul schimbării parametrilor sistemului.
  - durată scurtă a ciclurilor de testare.
  - estimări mai exacte pentru procesul de planificare a testului.
  - posibilitatea efectuării mai multor teste (scripturile de testare pot fi rulate şi după orele de program economisind astfel timp).
  - generarea rapidă a condiţiilor de testare.
- 3. avantaje privind calitatea
  - o mai bună înţelegere a scopului testării.
  - o acoperire mai mare a elementelor de testat.
  - rezultate mai consistente datorită repetabilităţii testelor.
  - compararea automată a rezultatelor.

20

Test Driven Development - prezentare

- TDD (Test Driven Development) este o tehnică de dezvoltare software bazată pe iteraţii mici care se desfasoară astfel: se scrie codul de test pentru functionalităţile noi care trebuie introduse în aplicaţie, apoi se scrie codul care implementează funcţionalitatea.

21

Test Driven Development

- Test Driven Developement înseamnă să se scrie mai intai unit test-uri şi apoi să se scrie codul, trecând prin cele 3 stării specifice:
- RED, când mai întâi există numai testul scris, fără nici un pic de cod, deci nu se compilează. Apoi când se "mimează" o funcţie pentru a compila codul şi a pica testul.
- GREEN, când se modifică codul în cel mai simplu mod pentru a satisface exact acel test (de exemplu, dacă exită o funcţie de căutare şi returnează true/false şi verifică să gasească un element se scrie doar return true).
- REFACTOR, unde se elimină codul duplicat, pentru a simplifica şi generaliza cât mai mult codul.

22

Concluzii

- Testarea automată nu va putea înlocui în întregime testarea manuală şi nici nu trebuie.
- Tester-ii pot să observe cum un utilizator poate interacţiona cu produsul, în timp ce un sistem de testare automată nu poate întotdeauna să prevadă aceste acţiuni sau să găsească cea mai bună cale de a le testa.
- Dacă sunt bine folosite, programele de testare automată măresc considerabil productivitatea QA, economisesc costuri, măresc semnificativ consistenţa şi calitatea produsului şi ajută la optimizarea şi accelerarea procesului de dezvoltare al unei aplicaţii.

23