

Testarea Produselor Soft

Lect. Dr. Sergiu Jecan

sergiu_jecan@yahoo.com

George Boitor - Senior QA Engineer

george.boitor@gmail.com

Skype: [georgeboitor](#)

Bogdan Dinga - Senior QA Engineer

bogdan.dinga.paul@gmail.com

Skype: [dinga.bogdan](#)

“Quality is not an act, it is a habit” – Aristotle

Curs 1

- Termeni Cheie
- Ce este Testarea?
- Obiectivele Testării
- Beneficiile Testării
- Psihologia Testării
- 7 principii ale Testării
- Procesul de Testare

Termeni Cheie

- Feature – funcționalitate
- Bug – eroare
- Fixing bugs – “rezolvare” de erori
- Debugging – depanare, depistarea erori
- Design – proiectare
- Specifications, specs – specificații
- User requirements – cerințe bazate pe nevoile utilizatorilor produsului
- Unit testing – testare (de regulă, automata) a unui modul
- Build – o versiune executabilă a unui program
- Deliverable – produs intermediar livrabil (de regulă documente sau module de aplicație, buildul în sine)
- Software Development Life Cycle - Procesul de dezvoltare al unei aplicații/produs
- Software Testing Life Cycle - Procesul de testare al unei aplicații

Ce este Testarea?

Testarea produselor software este o activitate în procesul de dezvoltare al unei aplicații/produs. Aceasta activitate cuprinde un set de metode, strategii, cunoștințe, abilitați care se folosesc în certificarea calității/funcționalității unui produs, validarea lui în funcție de specificații.

Testarea se face cu un anumit scop:

- Sa ne asiguram ca dezvoltam produsul corect
- Prevenirea defectelor
- Gasirea defectelor
- Creșterea încrederii în nivelul de calitate al produsului dezvoltat
- Input de informatie legat de calitatea unui produs

Obiectivele Testării

Obiectivele testării diferă în funcție de tipurile de testare care se fac.

Development Testing - În timpul procesului de dezvoltare al unui produs (Testarea Componentelor, Testarea de Integrare, Testarea Sistemului) obiectivul principal este găsirea a cât mai multe defecte astfel încât acestea să fie reperate cât mai repede, la un cost cât mai mic.

Acceptance Testing - obiectivul principal este confirmarea faptului că produsul dezvoltat funcționează cum trebuie (conform specificațiilor) și creșterea încrederii în produsul dezvoltat.

Maintenance Testing - obiectivul principal este de confirmare că nu au fost introduse noi defecte.

Beneficiile Testării?

- Crește gradul de încredere în produsul dezvoltat;
- Crește calitatea produsului;
- Scad costurile de mentenanță;
- Crește performanța produsului;
- Crește gradul de satisfacție al persoanei/persoanelor care au un interes în produs;
- Creșterea profitului.

Psihologia testarii

1. Este indicat ca testarea unei funcționalități/produs sa fie făcuta de o persoana diferita de cea care a scris codul; în acest fel se diminuează riscul ca sentimentele personale sa intervină în calitate.
2. Raportarea problemelor/greșelilor poate fi interpretata ca și critica pentru cel/cei care au dezvoltat codul; Comunicarea și empatia sunt esențiale în a preveni acest lucru.
3. Un tester trebuie sa fie curios, cu mintea deschisa, ochi critic bine dezvoltat, atent la detalii.
4. Expunerea problemelor găsite trebuie făcuta într-un mod constructiv, fără atac la persoana.

7 Principii ale Testarii(1)

- Testarea arata prezenta defectelor:
 - Testarea nu poate dovedi ca o aplicație/produs nu are nici un defect; poate doar reduce probabilitatea existentei unor defecte nedescoperite.
- Testarea exhaustiva e imposibila:
 - Testarea tuturor scenariilor posibile este imposibila/nefezabila. Testări trebuie sa își prioritizeze activitatea de testare în funcție de anumiți factori.
- Early Testing - Testarea facuta cat mai devreme in SDLC
 - Testare ar trebui făcuta cat mai devreme în procesul de dezvoltare al unei aplicații astfel încât problemele apărute sa fie reparate cat mai repede.
- Defect Clustering - Gruparea defectelor
 - Testarea trebuie concentrata în zonele care prezintă cele mai multe riscuri, cele mai multe defecte(20% din cod conține de obicei 80% din defecte)

7 Principii ale Testarii(2)

- Pesticide Paradox (Paradoxul Pesticidului)
 - Rularea în continuu al aceluiași set de teste o sa valideze doar o anumita parte a aplicației/produsului; testele ar trebuie revizuite constant astfel încât sa acopere o parte mai mare a aplicației/produsului;
- Testarea in functie de context
 - Testarea se face în funcție de zona de funcționalitate care este dezvoltata, în funcție de aplicația creata;
- Absence-of-errors fallacy
 - Găsirea și fixarea defectelor este fără sens dacă aplicația/produsul nu e construit specificațiilor și așteptărilor.
 - O aplicație oricât de buna ar fi (și fără erori), va fi respinsa de utilizatori dacă e dificila și greu de utilizat

Procesul de Testare(1)

1. Planificare si Control

- a. Obiectivele testării sunt definite și sunt alese activitățile de testare care se pliază cel mai bine pe context. Acest proces e continuu pe toată durata dezvoltării unei aplicații.

2. Design si Analiza

- a. Re-verificarea bazei de testare - specificații, design-uri, raport-urilor de risc, ...
- b. Evaluarea testabilitatii unui Feature/Aplicații
- c. Crearea si prioritizarea scenariilor de testare initiale
- d. Identificarea datelor de test de care este nevoie
- e. Crearea/Validarea mediului de testare

Procesul de Testare(2)

3. Implementare si Executie

- a. Finalizarea, implementare și execuția scenariilor de testare
- b. Crearea și prioritizarea procedurilor de testare, crearea datelor de test.
- c. Crearea suitelor de testare
- d. Verificarea mediului de testare
- e. Logarea rezultatelor
- f. Compararea rezultatelor cu specificațiile
- g. Raportarea discrepantelor
- h. Raportarea activitatilor de testare
- i. Repetarea activităților de testare ori de cate ori e necesar


Procesul de Testare(3)

4. Verificarea activitatilor precedente si raportarea

- a. Compararea rezultatelor de testare cu specificațiile
- b. Verificare cantitativa si calitativa a activitatii de testare
- c. Crearea unui Raport de Testare

5. Activitati Finale

- a. Documentarea tuturor aspectelor de testare
- b. Închiderea incidentelor
- c. Arhivarea artefactelor de testare
- d. Retrospectiva la tot procesul de testare
- e. Feedback si Imbunatatirea activitatilor viitoare de testare



Testarea produselor software

Lect. Dr. Sergiu Jecan
sergiu_jecan@yahoo.com

ISTQB
International Software Testing Qualification Board
www.istqb.org

■ CURS 1

- Eficiență, efectivitate, eficacitate
- Termeni cheie
- Exemple istorice
- Definirea erorii
- Costul calității

[Reguli]

- 30% Examen scris
- 20% Teme
- 50% Proiect seminarii
 - Dezvoltare aplicatie + Internationalizare (cu cod sursă accesibil) (10%);
 - Planificare Gantt, Specificatii, (Proiectare), Produse auxiliare (10%);
 - Dosar de testare (30%):
 - Plan testare
 - Cazuri de testare
 - Raport de testare (cu baza de date si grafice)
 - Instrumente de testare automată.

Echipe: 4 persoane (recomandat: 1 manager + 3 muncitori: programator, tester, localizator samd)

Evaluare: 1 notă pe echipa (calitatea proiectului afectează toți membrii)



[Termeni cheie

- Feature – funcționalitate
- Bug – eroare
- Fixing bugs – “rezolvare” erori
- Debugging – depanare, corectare erori
- Design – proiectare
- Specifications, specs – specificații
- User requirements – cerințele beneficiarilor
- Unit testing – testare modulara
- Build – statut executabil al unui program
- Deliverable – produs intermediar livrabil (de regulă documente sau module de aplicație)

[Eficiență, efectivitate, eficacitate]

Postulat: nu exista programe perfecte

Testarea PS – proces ce trebuie integrat în ciclul de viață software pentru asigurarea unei balanțe optime între **eficiență** și **efectivitate**.

Eficiență – doing things right

Efectivitate – doing the right things

Eficacitate – getting things done

Eficient – accent pe minimizarea costurilor (atingând “suficient de bine” obiectivele, obiective de fațadă):

- TENDINȚA GENERALĂ de CONSERVARE a RESURSELOR
- simptom al rentabilității

Efectiv – accent pe calitatea rezultatelor (indiferent de cost)

- IDEAL (de regulă resursa timp se consumă inevitabil)
- simptom al calității

Eficace – punct optim între ideal și real: să se atingă cât mai multe din obiective în condiții de resurse limitate (timp, oameni, capital)

[Eficiență, efectivitate, eficacitate]

Cazuri extreme:

- Orientare spre eficiență (e posibil ca rezultatele să fie atât de slabe încât să provoace costuri ulterioare, anulând eficiența) – tendință naturală sub aspect economic!
- Orientare spre efectivitate (e posibil ca rezultatele să nu se obțină niciodată datorită consumării resurselor, anulând efectivitatea)

Punctul de eficacitate variază după obiectivele propuse:

- Efectivitate prioritară – tendință în domeniile de care depind vieți umane (obiectivele au un prag sau o marjă de abatere mai strictă decât resursele);
- Eficiență prioritară – tendință în situațiile de monopol sau domenii în care se tolerează marje mari de abatere de la calitate dar există limitări mai stricte asupra resurselor.

Management – trebuie să asigure identificarea punctului de eficacitate și urmărirea acestuia.

În domeniul software:

- Efectivitate prioritară => minimizarea erorilor (software medical, militar, economic);
- Eficiență prioritară => ascunderea erorilor (software domestic)

Tendință actuală în producția software:

apropierea de efectivitate = măsură de asigurare a calității

Robert Buchmann, Ph.D. = măsură de prevenire a costurilor de despăgubire, întreținere
Babes Bolyai University

[Eficiență, efectivitate, eficacitate]

Teoria informației:

- Oportunitatea informației (o formă a eficienței)
- Acuratețea informației (o formă a efectivității)

Informația oportună este cel mai probabil inexactă și invers.

Implicații în management:

- Deciziile pe bază de informație imediată (uneori neverificată) sunt orientate spre eficiență dar pot fi eronate
- Deciziile pe bază de informații verificate și exacte sunt orientate spre efectivitate dar consumă resurse (cel puțin timp).

Testarea, ca activitate creatoare de costuri, poate fi la rândul său:

- Eficientă – se desfășoară rapid și superficial în vederea lansării imediate a produsului (e mai degrabă o confirmare a funcționalității generale);
- Efectivă – e un proces costisitor orientat spre detectarea erorilor, cu costurile:
 - Echipă specială de testare
 - Echipamente de testare
 - Activități de testare planificate (=consumatoare de timp) în cadrul procesului de producție software.

Măsuri de îmbunătățire a eficacității testării (atât eficiență cât și efectivitate) :

- Automatizarea testelor
- Distribuirea testelor (Testarea colectivă)

Exemple istorice

Calculatorul = instrument **domestic**, nu mai este apanajul specialiștilor

Informațiile stocate electronic = **extensie a memoriei umane**

Erorile software = afectează integritatea informației posedate de om, propagă diverse costuri, pot afecta vieți omenești

Exemple:

- 1994 – eroare la împărțire în procesorul Pentium s-a propagat în numeroase aplicații software (utilizatorul nu își pune problema ca PC-ul să greșească la calcule!);
 - Eroarea se manifesta doar în anumite situații, deci nu afecta pe toată lumea;
 - Eroarea a fost semnalată de testerii, dar ignorată de management pentru a nu crea costuri suplimentare de corectare;
 - Retragerea de pe piață a creat costuri mult mai mari, anulând criteriul eficienței invocat inițial!
 - În prima fază, managementul Intel a retras de pe piață doar produsele acelor utilizatori care puteau dovedi că au fost afectați negativ de eroare=> scandal, noi costuri;
 - De atunci, Intel are un site de semnalare a erorilor pe baza feed-backului public iar procesoarele Pentium III 1.13 Ghz au fost retrase de pe piață înainte ca problema să capete anvergură;
- 1999 – un dispozitiv NASA a eșuat pe planeta Marte datorită “reducerii costurilor” la mecanismul de aterizare și la desfășurarea testelor;
- 1991 – o marjă de eroare acumulată prin funcționarea continuă pe termen lung a produs moartea a numeroși soldați în războiul din Golf (marjă de eroare prea mare la detectarea rachetelor inamice);
- 2000 – eroarea Y2K, tolerată încă din anii 70 pe principiul eficienței. Costurile de prevenire în preajma anului 2000 au fost comparabile cu ale marilor catastrofe naturale.

Eroarea

- Terminologie:
 - **Problemă, eroare, bug** – termeni generali;
 - Anomalie, incident – termeni cu conotație negativă slabă, când sursa erorii nu este estimată (poate fi o cauză externă) sau funcționalitatea programului nu e afectată (culori alese greșit);
 - Pană, cădere, defect – termeni cu conotație negativă, indică faptul că funcționarea programului a fost întreruptă.
- Definiția erorii este raportată la specificații
- **Specificații (specs)** = un pseudocontract care definește produsul, scopul său, interacțiunile cu utilizatorul și, uneori, indică ce NU trebuie să facă produsul.
- Eroare = 5 condiții:
 1. Programul nu realizează ceva ce era prevăzut în specs;
 2. Programul realizează ceva ce specs nu prevede;
 3. Programul realizează ceva ce specs prevede explicit că NU va realiza;
 4. Programul nu realizează ceva ce specs nu prevede, dar ar trebui (specs implicite!)
 5. Programul nu satisface utilizatorul deși asigură celelalte 4 condiții (lent, dificil de utilizat și înțeles, insuficient documentat).



[Example – Calculator]

Specs: indică realizarea de adunări, scăderi, înmulțiri, împărțiri
indică faptul că programul rezistă la tastare aleatoare

Eroare tip 1: o adunare dă un rezultat eronat sau nu are efect

Eroare tip 2: apăsarea aleatoare a tastelor blochează programul

Eroare tip 3: programul calculează radicali (extra-feature – atrage costuri suplimentare)

Eroare tip 4: tastele nu sunt ordonate crescător, butoanele nu au forma standard Windows (standarde asumate implicit, nu sunt explicitate de specs)

Eroare tip 5: tastele sunt prea mici, cu riscul de apăsare pe lângă ele,
simbolurile de pe taste sunt neclare, programul e lent
+orice alte nemulumiri potentiale ale clientului (**subiectivismul calitatii**)

[Eroarea]

Subiectivismul erorii:

- o eroare e definita de un observator, nu de existenta sa (nedetectata)
- observatori diferiti vor atribui nivele de calitate diferite
- **testerul** e cel mai important observator, erorile care trec de el vor fi costisitoare
- **depanatorul** va corecta acea parte din erorile observate considerate relevante de **manager**

Sursele erorilor (in ordinea descrescătoare a relevanței si gravității):

1. **Specificatiile** – lipsuri, ambiguitate, modificări din mers, comunicare eronată (se impune documentarea amănunțită a specificațiilor, ca un contract între membrii echipei)
2. **Proiectarea** – motive similare cu erorile de la specificații
3. **Programarea**
 1. erorile cele mai ușor de urmărit și corectat (depanatoare automate)
 2. Adesea erorile de programare sunt efecte secundare ale erorilor de specificatii si proiectare
4. **Depanarea** – erori produse de procesul de corectare
5. **Testarea** – erorile false

[Costurile efectivității]

Calitatea e gratuita, lipsa sa costa!

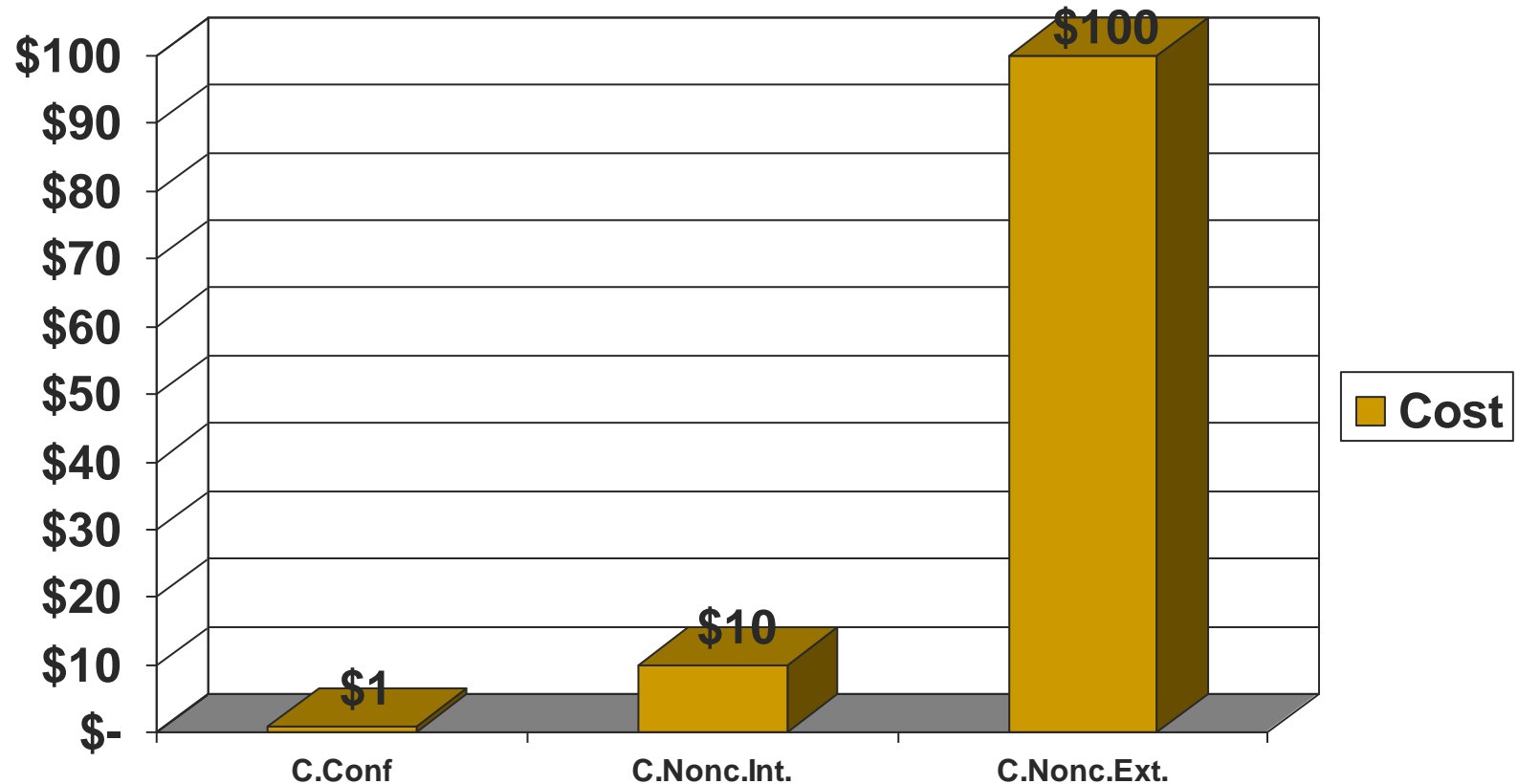
Tipuri de costuri:

- **Costuri de conformitate** (efortul depus în urmărirea unui prag de calitate)
 - Au caracter preventiv față de fazele târzii ale procesului de producție: impunerea unei discipline, planificări, reguli de programare (comentarii, nume de variabile, claritatea codului sursă), testările superficiale de confirmare a funcționalității generale.
- **Costuri interne de nonconformitate** (efortul depus în detectarea abaterilor față de pragul de calitate urmărit și eliminarea lor)
 - Au caracter preventiv față de manifestarea erorii la beneficiar: testarea și corectarea erorilor
- **Costuri externe de nonconformitate** (efortul depus în eliminarea erorilor manifestate la beneficiar – despăgubiri, consultanță, retragerea produsului de pe piață etc.)

În practică:

- **$C_{ex} > C_c + C_{int}$**
- **Pe măsură ce cresc C_c și C_{int} , scad C_{ex}**
- **Costul unei erori se propagă exponențial**
- **=> investiții masive în asigurarea calității în detrimentul costurilor propagate**

[Propagarea costurilor unei erori]



■ CURS 2

- Cerințe proiect
- Componentele produsului software
- Modele de procese de producție software
- Axiomele testării



[Referințe Web

- **www.bugnet.com** publică erori detectate în produse software comerciale și soluții de corectare;
- **www.io.com** referă o colecție de articole legate de testare;
- **www.mtsu.edu/~storm** – colecție de hiperlegături relevante;
- **www.qaforums.com** – grupuri de discuții;
- **www.stickyminds.com** – revista on-line;
- **www.securityfocus.com** – sursă de informații legată de vulnerabilități de securitate;

[Proiect]

- **Cerințele beneficiarului:**
 - **Profil client: casă de schimb valutar**
 - **Nevoi:**
 - **Înregistrarea operațiilor de cumpărare și vânzare valută**
 - **Obținerea de rapoarte cu evoluția cursului leu/valută**
 - **Obținerea de rapoarte cu istoricul operațiilor valutare (diverse totaluri – pe zile, pe persoane, pe valute etc.)**
 - **Rulare Windows XP**
 - **Interfață grafică (formulare, meniuri)**

Criterii evaluare la deadline 1

- Aplicația nu trebuie să fie completă și testată la deadline 1 pentru nota 5!
- Aplicația va fi prezentată de manager + cei implicați în programare
- Criterii deadline 1:
 - Calitatea diagramei Gantt, compararea diagramei Gantt cu stadiul proiectului
 - Calitatea specificațiilor
 - Calitatea codului sursă (comentarii!) și stăpânirea sa de către programatori
 - Funcționarea generală a aplicației (testare pozitivă)
 - Nivelul de completitudine a aplicației (baze de date, rapoarte, formulare, meniuri, etc.)
 - Proiectele copiate de la o echipă la alta vor fi notate cu 1
- Autoritatea șefului de echipă în cazuri de nesubordonare:
 - Șeful are dreptul să elimine din echipă membri, dar nu va primi membri înlocuitori din alte echipe
 - Membri eliminați din echipe trebuie să își construiască propriul proiect, singuri

Criterii evaluare la deadline 2

- Aplicația localizată și documentația de localizare (aspectele localizate, metoda folosită, complicațiile întâmpinate exemplificate)
- Documentația testelor auxiliare (testare hardware, probleme detectate de compatibilitate, de utilizabilitate)
- Plan de testare (Test Plan Outline)
- Documentația cazurilor de testare (justificarea claselor de echivalențe)
- Baza de date cu rapoarte de eroare (Excel sau PHP)
- Metrice și grafice Excel cu evoluția ratei de detectare și reparare a erorilor
- Instrumente de testare automată:
 - Un monkeytester creat în Autolt
 - Macro-uri Autolt pentru toate testele realizate
- Produse auxiliare:
 - Site de prezentare a proiectului (membrii, sarcini, diagrama Gantt, info de contact)
 - Help (poate fi on-line, inclus în pagina de prezentare)

Elemente dosar proiect

- Gantt
- Specificatii
- Aplicatie in limba romana (cu cod sursa)
- Aplicatie in limba engleza
- Documentatie localizare (aspecte localizate, metoda, complicatii intalnite)
- Plan de testare
- Documentatia cazurilor de testare (justificarea claselor de echilvalente)
- Documentatia testelor auxiliare (hardware, software, utilizabilitate)
- Baza de date cu rapoarte de eroare + grafice +metrici
- Scripturi Autolt:
 - Monkeytester
 - Script Autolt
- Produse auxiliare:
 - Pagina Web de prezentare a proiectului (membri, sarcini, Gantt, date contact)
 - Help (poate fi inclus in pagina Web)
- Reguli:
 - Nota se acorda tuturor membrilor unei echipe, indiferent cat au lucrat
 - Managerul poate decide sa excluda membri ai echipei dar nu va primi inlocuitori
 - Cei exclusi din echipe vor trebui sa realizeze proiectul pe cont propriu

[Testerul]

- Detectează (observa) erori **cat mai devreme**, pentru a opri propagarea
- Confirma un **nivel de calitate a produsului**
- Lucrează împotriva programatorilor (succesul testerului = insuccesul progr.)
- Semnalează erorile într-un mod convingător și sistematic
- Confirmă rezultatele depanării

Rezolvare erori (Fixing, resolving) <> Corectarea erorilor (depanare, debugging)

Metode de **rezolvare** care nu presupun **corectare**:

- Completarea specificațiilor cu restricții de utilizare;
- Definirea unor cerințe de hardware și compatibilitate;
- Acorduri cu beneficiarii pentru tratarea problemelor create de erori.
- Anuntarea unui patch (programe corectoare) sau upgrade (amanarea publicării unor module);

[Testerul]

- Calitățile testerului:
 - Caracter explorator, pentru detectarea neprevăzutului
 - Insistență (și răbdare) în derularea de operații repetitive
 - Creativitate, pentru formularea de situații neprevăzute de programator
 - Perfecționism, pentru a detecta chiar și erori care nu vor corectate sau eliminate
 - Diplomatie în semnalarea erorilor (sistematic și impersonal), proces care frustrează programatorul
 - Persuasiune pentru a indica gravitatea precisă a efectelor erorii (programatorii subestimează erorile)
 - Cunoștințe de programare pentru a surprinde șabloanele comportamentale ale programului sau pentru a construi instrumente automate de testare
 - Cunoștințe privind beneficiarul și domeniul de aplicație (informatica economică!)

Procesul de producție software (PPS)

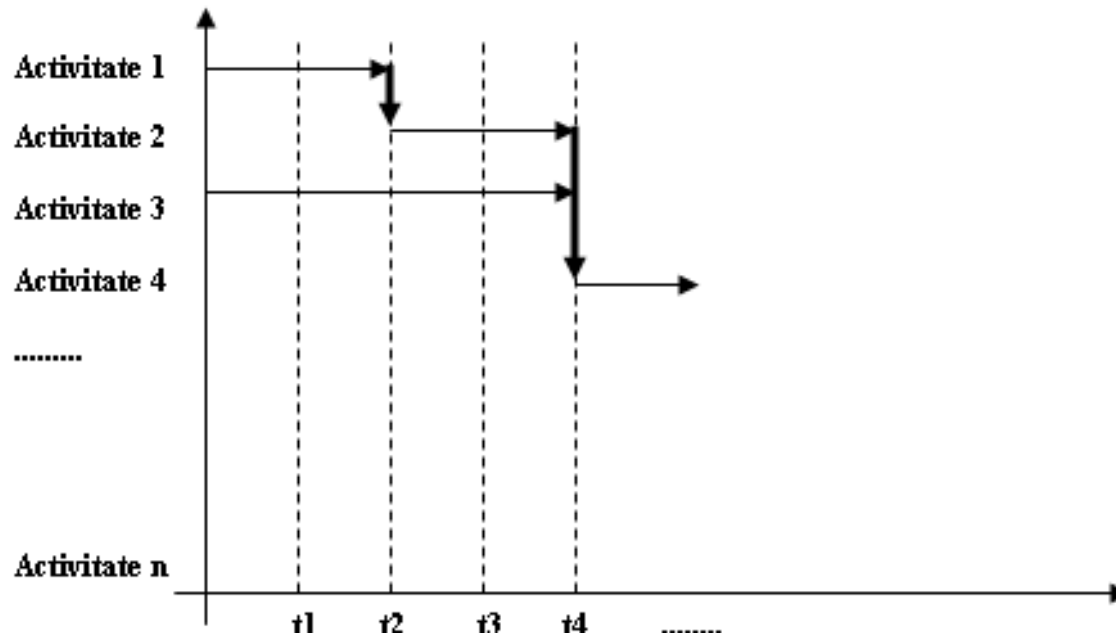
- Obiectul testării : **produsul software (PS)**, rezultat al **procesului de producție software (PPS)**
- Produsul software format din:
 - **Aplicație (produs principal)**
 - **Produse auxiliare:**
 - Help, fișiere Read Me, manual de utilizare
 - Pictograme, fonturi
 - Tutoriale și exemple
 - Informații pentru service și suport
 - Kit de instalare, wizarduri de configurare
 - Etichete, ambalaj (informații tehnice, cod serial etc.)
 - Materiale promo
 - **Produse intermediare**, “consumate” în cadrul PPS (**documente livrabile, deliverables**):
 - Cerințele beneficiarilor (user requirements)
 - Specificațiile (specs)
 - Planificările
 - Documente de proiectare
 - Documente de testare

Procesul de producție software (PPS)

- Cerințele
 - Descriu nevoia pe care o va satisface produsul
 - Metode de creare: chestionar, interviu, metodologii ca Persona sau Volere
 - Sursa: grupurile focus (beneficiari potențiali selectați ca eșantion de respondenți), pot fi implicate și în testarea finală
- Specificațiile
 - Indică precis cum se va comporta programul, asigură o terminologie și obiective comune la nivelul echipei
 - Pot fi formalizate și rigide în domenii riguroase, orientate pe efectivitate (medical, militar etc.)
 - Pot fi flexibile și adaptabile în alte domenii
 - Sursa: cerințele

Procesul de producție software (PPS)

- Planificările
 - Definesc sarcini, responsabilități și gestionează resursa timp în mod eficient (paralelismul activităților)
 - Diagrame Gantt:



Procesul de producție software (PPS)

■ Documente de proiectare

- Detaliază structura și comportamentul aplicației, pe baza specificațiilor
- Exemple:
 - Arhitectura aplicației (modulele și interacțiunile)
 - Diagrama fluxurilor de date (descrie transferurile de date)
 - Diagrama tranzițiilor (descrie succesiunea stărilor prin care poate trece programul)
 - Scheme logice (descriu algoritmi)
 - Comentariile asociate modulelor aplicației, necesare ca mecanism de comunicare cu testerul și depanatorii

■ Documente de testare

- Planificarea testării (metodologia, obiectivele de calitate, resursele necesare, responsabilii)
- Cazurile de testare (elemente testate, datele de test, procedura de testare pe fiecare caz)
- Rapoartele de testare (descriu problemele detectate pe cazuri de testare și sunt agregate într-o bază de date)

○ Documentația instrumentelor de testare automată, dacă e cazul

○ Metrice și statistici

Procesul de producție software (PPS)

- Persoane implicate în PPS:
 - Managerul de proiect – definește specificații și răspunde de ele, planificări și ia deciziile de risc (ignorarea de erori, alocare de resurse)
 - Inginerii de sistem – realizează documentele de proiectar, elimină erori de proiectare
 - Programatorii – realizează codul sursă, depanează sau elimină erori de programare
 - Testerii – detectează erori și le semnalează
 - Asistența tehnică – redactează Help, tutoriale, instrumente de înregistrare, suport și alte produse auxiliare
 - Managerul de implementare – alcătuiește produsul final, livrabil beneficiarului

Modele PPS

■ Big Bang:

- organizare haotică,
- managerul alocă resurse global, unui grup neorganizat
- planificările sunt înlocuite cu deadline-uri succinte
- lipsește strategia și structurarea PPS
- accentul se pune pe programare;
- testarea este secundară, cu rol strict de confirmare și cu resurse minimale
- rezultatele testării (erorile detectate) sunt receptate negativ de manager (tratate ca o întârziere a produsului, datorită neintegării testării în PPS).

■ Code-Fix:

- Ciclu programare-testare-depanare iterativ;
- Se realizează un prototip, acesta se testează;
- Se corectează prototipul, se testează din nou, etc.
- Ciclul se încheie când se termină resursele (resursa de timp);
- Planificarea și documentația sunt slabe
- Testarea capătă importanță.

■ Cascadă – model pe faze (se trece la faza următoare doar după definitivarea fazei curente):

- Preliminarii și planificare;
- Analiza cerințelor potențialilor beneficiari (definirea de specs);
- Proiectarea aplicației;
- Dezvoltarea (programare, configurare, implementare);
- Testarea;
- Obținerea produsului final.

Modelul spirala

- Definit în 1986 – A Spiral Model of Software Development and Enhancement (Barry Boehm)
 - Îmbinare între cascadă și code-fix;
 - Se definește un ciclu code-fix, dar care conține toate fazele modelului Cascadă;
 - Produsul crește în iterații succesive, testarea apare la fiecare iterație.
- Fazele unei iterații:
 - Specificarea obiectivelor, alternativelor și constrângerilor iterației curente
 - Identificarea și tratarea riscurilor
 - Evaluarea alternativelor
 - Dezvoltarea și testarea produsului în starea asociată iterației curente, conform modelului Cascadă
 - Planificarea următoarei iterații
 - Decizii de abordare a următoarei iterații pe baza riscurilor și nerealizărilor iterației curente
- Testerul intervine în fiecare iterație și influențează calitatea produsului devreme (în primele iterații)
- www.agilemanifesto.org – modelul PPS agil, varianta a spiralei, orientat spre **schimbarea frecvență a specificațiilor pentru avantaj competitiv**

Axiomele testării

1. Testarea completă e imposibilă. Motive:

1. Numărul mare de date de intrare (și combinații)
2. Numărul mare de ieșiri posibile
3. Numărul mare de ramificații ale algoritmilor
4. Subiectivismul specificațiilor și erorilor de tip 5 (moștenit din subiectivismul cerințelor)

Exemplu: teste pentru Windows Calculator:

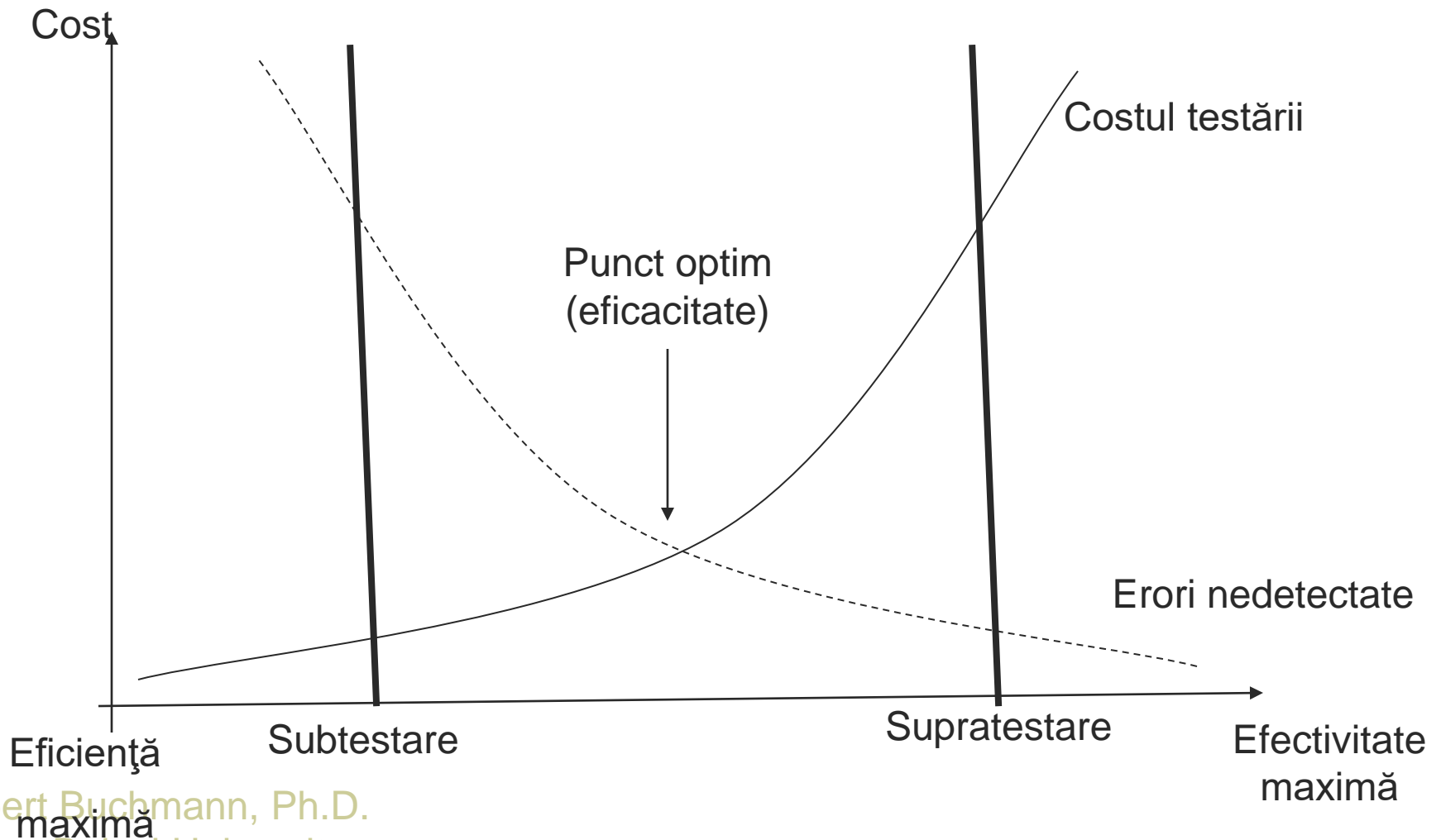
- Toate adunările posibile, toate scăderile posibile etc. cu numere de până la 32 cifre
- Toate adunările invalide, scăderile invalide etc. (cu șiruri de caractere)
- Toate operațiile cu valori shortcut ($p+p=2*\pi$)
- Toate operațiile cu valori editate în caseta de intrare (înlocuire cifre)

Concluzii:

- Efectivitate totală = imposibilă
- Testarea = activitate de management (urmărirea eficacității)
- Testarea = activitate de management a riscului (definirea cazurilor de testare relevante, capabile să detecteze majoritatea erorilor)
- Testarea trebuie să se oprească atunci când costul testării depășește costul erorilor remanente (latente)



[Axiomele testării]



[Axiomele testării]

2. Testarea nu poate dovedi că erorile nu există

3. Pe măsură ce se detectează erori, apar mai multe erori. Motive:

1. Erorile sunt înrudite (erori-consecințe, determinate de alte erori)
2. Erorile pot fi consecințe ramificate ale erorilor timpurii (din specificații)
3. Programatorul care a produs o eroare va produce și altele în porțiunea sa de program
4. Reutilizabilitatea codului și algoritmilor sursă implică reutilizarea erorilor (cod sursă multiplicat prin copieri ale unor proceduri similare)

Consecință:

- Testarea are rezultate neconstante: perioade în care nu se detectează nimic alternează cu perioade în care se detectează erori multe

4. Paradoxul pesticid: imunizarea erorilor – erorile se detectează tot mai dificil dacă metoda și cazurile de testare rămân constante

Axiomele testării

5. O eroare e definită doar prin observație și nu prin existența ei. Consecințe:

- o eroare care nu se manifestă la beneficiar nu e o eroare (**subiectivismul calității software**)
- rezolvarea erorii <> corectarea erorii:
 - Rezolvare (fixing) = îndepărtarea manifestării erorii sau daunelor sale (să nu mai poată fi observată sau să fie acceptată de beneficiar)
 - Corectare (debugging) = îndepărtarea erorii propriu-zise (să nu mai existe eroarea și nici o șansă de manifestare a sa)

6. Nu toate erorile detectate vor fi corectate. Motive:

- Raportarea deficitară a erorilor de către tester
- Termenele de timp (limite neflexibile ale resursei timp)
- Erorile interpretabile (programatorii au tendința de a subestima erorile sau a le prezenta ca funcționalități intenționate)
- Corectarea poate fi riscantă (să producă noi erori)
- Corectarea poate fi prea costisitoare în raport cu daunele erorii
- 3 perspective diferite asupra erorilor: manager, programator, tester, fiecare cu motivele lui și nivelul de risc asumat.

Axiomele testării

7. Specificațiile sunt variabile. Motive:

- Calitatea scăzută a specificațiilor inițiale
- Adaptarea din mers a specificațiilor pentru a dobândi avantaj competitiv și corelare cu factori externi (progres pe piața software și IT)

8. Testerii lucrează împotriva programatorilor = sursă conflictuală

- succesul unui tester = insuccesul programatorilor

Metode de ameliorare a conflictului:

- Detectarea timpurie a erorilor pentru minimizarea propagării și efortului de depanare (corectare)
- Diplomație în promovarea succesului testelor (comunicarea impersonală a erorilor detectate, prin raport formal)
- Diplomație prin aprecierea succesului depanărilor (evitarea specializării în “vești proaste”)

9. Testarea necesită disiciplină și profesionalism.

- Modelul Big Bang anulează această cerință, impunând testarea ad-hoc intuitivă, cu efectivitate scăzută (managerul Big Bang nu vrea să audă că există erori și speră că nu va exista un observator care să le detecteze)
- Celelalte modele văd în testare un proces cu intrări livrabile (cod sursă, executabile, plan de testare) și ieșiri livrabile (raport de testare, metrice)

■ CURS 3

- Terminologie
- Structuri de programare
- Abordări în testarea software
- Testarea SBB

[Terminologie]

Terminologia variază dar trebuie să fie fixată (constantă, consistentă) în cadrul unei echipe pentru a elimina problemele de comunicare.

Termeni fundamentali:

Eroarea (vezi definiția). Modificările aduse la definiție (nerecomandate) trebuie cunoscute de toți membrii echipei (ex: ignorarea erorilor subiective)

Specificațiile – contract cu scop de unificare a scopurilor echipei și terminologiei

Precizie vs. Acuratețe – attribute ale ieșirilor aplicației:

- **Precizie** – ține de metodă, cât de bune sunt mijloacele prin care se urmărește un rezultat (ex: numărul de zecimale din rezultatul unei împărțiri)
- **Acuratețe** – ține de scop, cât de aproape e rezultatul obținut față de cel dorit (ex: corectitudinea rezultatului unei împărțiri)

[Terminologie]

Verificare vs. Validare:

- **Verificare** = confirmarea față de specs
- **Validare** = confirmarea față de cerințele utilizatorului
- Apar diferențe între cele două dacă trecerea de la cerințe la specs s-a făcut eronat.
- Karl Wieggers - *More About Software Requirements: Thorny Issues and Practical Advice* – Microsoft Press 2006)

Calitate vs. Fiabilitate:

- **Calitate** = nivelul de satisfacție a beneficiarului (subiectiv)
- **Fiabilitate** = probabilitatea programului de a funcționa într-un timp dat fără a manifesta erori (obiectiv, influențează calitatea)

Testare vs. Asigurarea calității (QA)

- **Testare** = detectarea timpurie a erorilor și semnalarea lor (previne manifestarea erorilor la beneficiar)
- **QA** = garantarea unor standarde de calitate pe tot parcursul PPS, inclusiv în derularea testării (previne manifestarea erorilor la testare și implicit la beneficiar)

[Exemplu specs]

- *Meniul Edit va avea două opțiuni, de sus în jos în ordinea: Copy și Paste*
- *Metodele de activare a opțiunilor vor fi:*
 - *Clic*
 - *combinația Alt-E urmată de C, respectiv P*
 - *combinațiile de taste implicite din Windows (Ctrl-C, Ctrl-V).*
- *Opțiunea Copy va avea ca efect copierea în Clipboard a afișajului Windows Calculator.*
- *Opțiunea Paste va avea ca efect copierea conținutului Clipboard în câmpul de afișare din Windows Calculator.*
- *Recomandare: la specificație se pot adauga și imagini dacă e cazul (în exemplul de față, notiunile de *meniu*, *opțiune* sunt suficient de clare)*
- *Specificația e sistemul de referință al testerului ! (vezi definiția erorii)*

Recapitulare structuri de programare

IF Statements

Each language offers equivalent IF functionality. The THEN clause is allowed only on the IF statement in Visual FoxPro.

Visual FoxPro	BASIC
<pre>IF nCnt < nMax nTot = nTot * nCnt nCnt = nCnt + 1 ENDIF</pre>	<pre>If nCnt < nMax Then nTot = nTot * nCnt nCnt = nCnt + 1 End If</pre>
Pascal	C/C++
<pre>if nCnt < nMax then begin nTot:=nTot * nCnt; nCnt:=nCnt + 1; end</pre>	<pre>if(nCnt < nMax) { nTot *= nCnt; nCnt++; }</pre>

Recapitulare structuri de programare

CASE Statements

Only Pascal does not offer defaults in CASE statements.

Visual FoxPro	BASIC
<pre>DO CASE CASE n = 0 ? 'Zero' CASE n > 0 ? 'Pos' OTHERWISE ? 'Neg' ENDCASE</pre>	<pre>Select Case n Case 0 Print 'Zero' Case Is > 0 Print 'Pos' Case Else Print 'Neg' End Select</pre>
Pascal	C/C++
<pre>case n of 0: writeln("Zero"); 1: writeln("One"); end</pre>	<pre>switch(n) { case 0: printf("Zero\n"); break; case 1: printf("One\n"); break; default: printf("? \n"); }</pre>

Recapitulare structuri de programare

FOR Loops

Each language offers a FOR statement; C/C++ has the most flexibility for expressions.

Visual FoxPro	BASIC
<pre>FOR n = 1 TO 10 ? n ENDFOR</pre>	<pre>For n = 1 to 10 Print n Next n</pre>
Pascal	C/C++
<pre>for n := 1 to 10 do writeln(n);</pre>	<pre>for(n=1; n<=10; n++) printf("%d\n",n);</pre>

Recapitulare structuri de programare

 *Visual FoxPro Language Reference*

WHILE Loops

Each language offers equivalent WHILE loop functionality.

Visual FoxPro	BASIC
<pre>DO WHILE n < 100 n = n + n ENDDO</pre>	<pre>Do While n < 100 n = n + n Loop</pre>
Pascal	C/C++
<pre>while n < 100 do n := n + n;</pre>	<pre>while(n < 100) n += n;</pre>

Recapitulare structuri de programare

Diferențe IF-CASE:

- IF oferă două alternative (chiar dacă ELSE lipsește)
- CASE oferă mai multe alternative și o variantă implicită (otherwise, default)

Diferențe FOR-WHILE:

- Condiția FOR are loc întotdeauna asupra variabilei contor
- Incrementarea contorului e implicită
- variabila de contorizare FOR indică precis numărul de iterații
- Condiția WHILE e mai generală
- Modificarea variabilei din condiția WHILE trebuie explicitată

Unele limbaje (nu si VFox) oferă două tipuri de WHILE:

- WHILE conditie DO instructiuni
- DO instructiuni WHILE conditie – asigură executarea a cel puțin unei iterații

Erori while clasice: ciclu infinit din motivele:

- Nu se face modificarea variabilei din conditie
- Se pune o condiție care va fi întotdeauna adevărată

Abordări în testarea software

■ Black box:

- Testerul stie CE ar trebui sa facă programul, nu și CUM!
- Testerul nu are acces la codul sursă, doar la ieșiri
- Se mai numește **testare comportamentală**: se testează comportamentul programului (ieșirile) corespundente unor stimuli (intrările)
- Se mai numește **testare funcțională**: programul e văzut ca o funcție definită pe domeniul tuturor intrărilor posibile, cu codomeniul în mulțimea tuturor ieșirilor posibile.

■ White box:

- Testerul are acces la codul sursă pentru a obține indicii privind logica după care se comportă programul
- Dezavantaj: **testerii pierd viziunea utilizatorului**, au tendința să testeze precizia programului, nu și acuratețea sa.

■ Ordinea recomandată:

- Mai întâi black box, pentru a asimila viziunea utilizatorului și a verifica dacă programul face CE trebuie (acuratețe)
- Apoi white box, pentru rafinarea testării și a verifica dacă programul face CUM trebuie (precizie)

Abordări în testarea software

- **Testare statică:**

- Testarea care nu implică executarea programului

- **Testare dinamică:**

- Testarea care implică executarea programului

- **Testare pozitivă (optimistă, test-to-pass):**

- Confirmarea funcționalității generale (ex Windows Calculator: dacă adunările între două numere dau rezultate corecte)
- Pune accent pe cazurile de utilizare uzuale, ignorând cazurile excepționale

- **Testare negativă (pesimistă, test-to-fail):**

- Forțarea limitelor programului, căutarea defectelor
- Pune accent pe cazurile de utilizare excepționale

- **Ordinea firească:**

- Mai întâi testare pozitivă – încurajează programatorii, dectează erorile de care se va lovi ORICE utilizator
- Apoi testare negativă – poate descuraja programatorii, se ignoră faptul că, **ÎN GENERAL**, programul funcționează

Abordări în testarea software

■ Testarea error-forcing (forțarea erorilor):

- Menită să declanșeze erori
- E simultan pozitivă și negativă
- Error-forcing pozitiv: se verifică dacă programul tratează excepțiile și erorile de utilizare (dacă s-au definit mesaje de eroare, nu se pierd date și programul nu se blochează)
- Error-forcing negativ: se testează limitele programului, se caută erorile de utilizare și excepțiile netratate de program

Lipsa unui mesaj de eroare este o **eroare**! Utilizatorul trebuie să știe ce s-a întâmplat atunci când operația dorită nu se desfășoară conform așteptărilor.

Atenție la confuzie:

- **eroare de utilizare, excepție** – situații invalide/de excepție provocate prin utilizare incorectă care sunt prevăzute și gestionate de aplicație (mesaje de eroare, prevenirea blocării aplicației, reluarea funcționării etc.)

- **eroare** – termenul general care indică defecte ale aplicației

[Testare statică black box]

- **Testare SBB = testarea specificațiilor**
- **Statică** pt că nu necesită rularea programului
- **Black box** pt că testerii nu au acces la metoda prin care specs au fost obținute din reqs (cerințe)
- **Rol SBB:** detectarea erorilor înainte de faza de programare, evită propagarea!
- Testarea SBB = activitate de cercetare
- Faze SBB:
 - Testerul se pune în locul utilizatorului
 - Testerul consultă specificațiile de securitate
 - Testerul consultă standardele curente
 - Testerul consultă produse similare sau recenzii on-line ale produselor similare (cunoscute de dpt. marketing)
 - Testerul consultă recenzii din presă asupra versiunilor anterioare ale produsului

[Testare statică black box]

- **Testerul în locul utilizatorului** – consultarea cerințelor sau a departamentului de marketing
- **Specificațiile de securitate** – considerate implicite, utilizatorul nu le va indica în cerințe (decât în cazuri excepționale: - soft bancar)
- **Standardele existente**
 - Testerul nu definește standarde, le consultă și află pe care dintre ele trebuie să le considere **specificații implicite**
 - Aspecte afectate de standarde:
 - Terminologia clientului trebuie reflectată în mesajele interfeței
 - Domeniul de aplicație (contabilitatea are reglementări ce trebuie reflectate de software)
 - Legislația impune reguli asupra unor produse software
 - GUI
 - Securitatea, implicită dar poate fi și solicitată explicit de beneficiar
 - Ex: standarde GUI impuse de Windows și Macintosh
 - Standardele GUI sunt definite de psihologia utilizatorului și ergonomie
 - Standardele GUI ameliorează curba de învățare a utilizatorului
 - Standardele GUI asigură reutilizabilitatea GUI



[Testare statică black box]

■ Produse similare (sau versiuni anterioare):

- Cunoscute de dpt. de marketing
- Se estimează diferența de complexitate față de produsul comparat
- Se estimează resursele necesare testării și se pot chiar defini cazuri de testare
- Se pot folosi calitatea, fiabilitatea sau securitatea unui produs similar ca prag de referință urmărit
- Recenziile din presă scad efortul de testare și atrag atenția asupra anumitor erori.

■ Testarea SBB se finalizează cu un checklist:

- Completitudinea specs - se indică omisiunile, ambiguitățile
- Acuratețea – se indică abaterile de la scopul aplicației
- Precizia – se indică abaterile de la corectitudinea metodei
- Consistența – se arată contradicțiile
- Relevanța – se indică părțile redundante sau irelevante față de cerințe
- Fezabilitatea – se indică elementele pentru care nu există resurse
- Dependența de codul sursă – specs definesc un produs, nu trebuie să conțină cod sursă, date de configurare sau implementare (deformație profesională a programatorilor care scriu specs)
- Testabilitatea – se arată unde lipsesc informații necesare testării.

[Testare statică black box]

■ Termeni cheie în specs:

- **Niciodată, întotdeauna, fiecare, niciunul, totul, nimic** – afirmații absolute, a căror valoare de adevăr absolut trebuie verificate
- **Cu siguranță, deci, evident** – termeni persuasivi care nu au ce căuta în specs (specs nu trebuie să convingă pe nimeni)
- **Uneori, adesea, în general, în principiu** – termeni vagi care nu au ce căuta în specs
- **Etc., asemănător, șamd** – nu trebuie să apară, enumerările în specs vor fi complete pentru a indica precis toate funcționalitățile testabile
- **Bun, rapid, ieftin, stabil** – termeni necuantificabili, deci netestabili (dacă apar, vor fi însoțiți de pragul cuantificat și unitatea de măsură)
- **Procesat, respins, eliminat, ignorat** – termeni cu încărcătură puternică, maschează o funcționalitate mai detaliată (dacă apar, trebuie definită funcționalitatea asociată lor)
- **Dacă ... Atunci...** (fără **Altfel**) – evită specificarea ramurei negative a deciziei (aceasta trebuie explicitată clar)

[Testare statică black box]

- Modelul Big Bang ignoră testarea SBB. Motive:
 - Lipsa specificațiilor sau ambiguitatea lor
 - Testarea inclusă ca ultimă fază a PPS
 - Testerul nu are sistem de referință
 - Recomandare: testerul trebuie **să provoace crearea de specificații**:
 - Contactează persoanele care au cea mai precisă idee despre scopul proiectului (marketing, manager, poate chiar programatori)
 - Testerul creează propria specificație pe baza informațiilor obținute
 - Testerul supune aprobării propria specificație, care sigur va fi incompletă
 - Managerul va fi obligat să consulte specificația, să o completeze și eventual va organiza o recenzie în ședință publică
 - În urma acestui proces vor ieși la iveală detaliile concrete ale proiectului (și chiar managerul va căpăta involuntar o idee mai clară asupra sa)
- www.mfagan.com – metodologie formală de inspectare software (inclusiv a specificațiilor)

■ CURS 4

- Testarea DBB
- Clase de echivalențe
- Testarea intrărilor DBB
- Testarea stărilor și tranzițiilor DBB
- Recomandare: site-ul
www.softwaretestinghelp.com

[Testare dinamică black box]

- Specificațiile trebuie să indice corelația între stimuli și comportament (intrări și ieșiri)
- Testarea completă e imposibilă: testerul definește **cazuri de testare** relevante, pentru o balanță optimă între costul testării și succesul testării
 - Mai întâi cazurile de testare pozitivă
 - Apoi cazurile de testare negativă
- Modelul Big Bang – în lipsa specificațiilor, avem variantele:
 - Testerul provoacă specificațiile prin metoda amintită la SBB
 - Testerul apelează la testarea exploratorie, intuitivă, considerând că funcționalitatea generală este chiar specificația (nu se vor sesiza anumite categorii de erori conform definiției erorii)

[Testare dinamică black box]

- Cazuri de testare relevante = **clase de echivalențe** ale intrărilor (CE)
- Clasificarea echivalențelor
 - Gruparea intrărilor care produc un comportament similar al programului și testarea unui singur caz pe clasă
 - Reducerea combinațiilor teoretic infinite de intrări
- **O CE va conține un set de intrări care în mod potențial vor detecta aceeași eroare**
- Ex. Windows Calculator:
 - Avem cazul $1+1$, care se executa corect.
 - Se impune finalizarea testării: mai sunt resurse –timp- pt un singur test)
 - Care se va alege dintre $1+10$ și $1+p$?
- Clase Windows Calculator:
 - Clasa adunărilor cu nr.
 - Clasa adunărilor invalide
 - Clasa adunărilor cu shortcuturi ($p=\pi$)
 - Clasa adunărilor cu intrări editate
 - Aceleași clase pentru restul operațiilor

[Testare dinamică black box]

- Rafinarea CE:
 - Divizarea unei clase în subclase pentru surprinderea a cât mai multe erori potențiale;
 - Subclase ale clasei adunărilor între numere care pot teoretic detecta erori diferite:
 - Adunările cu numere întregi
 - Adunările cu numere reale
 - Adunările cu numere în format exponențial
 - Adunările cu numere de limită (cel mai mare și cel mai mic număr posibil)
- Rafinarea continuă a subclaselor va duce, teoretic, la obținerea unui număr de clase cu o singură intrare, deci la numărul tuturor intrărilor posibile. Așadar:
 - Rafinare excesivă – separarea în subclase a unor intrări care vor fi tratate identic de program (ex: e posibil ca întregii și realii să fie tratați identic)
 - Rafinarea superficială – ignorarea unor situații de excepție prin înglobarea lor în clase de intrări uzuale
 - Procesul de rafinare trebuie făcut top-down (clase-subclase-subsubclase)

[Testare dinamică black box]

- Definirea claselor inițiale și rafinarea lor definește **managementul riscului de testare** – cu siguranță nu se vor testa toate intrările dar e bine să se testeze cât mai multe din cele relevante.
- O clasă de echivalențe = o clasă de riscuri (se testează o singură intrare, restul rămân netestate în baza presupunerii de comportament identic)
- Calitatea rafinării dă eficacitatea testării
- Rafinarea se poate realiza:
 - Printr-o clasificare intuitivă a datelor de intrare
 - Pe baza cunoștințelor de programare ale testerului (care poate intui unde apar structuri IF sau CASE)
 - Prin testare white box (inspectarea codului sursă), care dă cele mai precise informații de clasificare a intrărilor pe baza structurilor ramificate

[Testare DBB a datelor]

- Criterii cheie în rafinarea CE:
 - **Limite**
 - **Limite interne**
 - **Valori nule sau implicite**
 - **Valori invalide**
- **Testarea limitelor** = testarea programului cu valori de intrare de la limitele claselor de echivalențe. Majoritatea erorilor au loc la limite.
- Tipuri de limite:
 - **Numerice: marginile unui interval**
 - **De tip caracter: capetele unui șir de caractere**
 - **De poziție: indicii inițiali și finali la poziționare în vectori, matrici, tabele BD, cicluri FOR**
 - **De cantitate: valori maxime și minime acceptate**
 - **Tehnice: de viteză (limitele unui modem), de stocare (limitele memoriei), de procesare (limitele unui procesor) etc.**
 - **De localizare: prima și ultima înregistrare a unui tabel, primul și ultimul element dintr-un vector**

[Testare DBB a datelor]

- Termeni cheie care desemnează limite:
 - prim—ultim (element, iterație, înregistrare)
 - minim-maxim (valoare, cantitate, lungime string)
 - început-sfârșit (de fișier, tabel, vector)
 - plin-gol (tabel, fișier, stivă)
 - cel mai lent-cel mai rapid (transfer, operație)
 - cel mai apropiat-cel mai îndepărtat (caracter, înregistrare)
- **Fiecare limită=2 sau 3 teste**
 - **Testare spre interior (pre-limita)**— nu trebuie să producă erori
 - **Testarea spre exterior (depășirea de limită)** – trebuie să producă erori de utilizare sau excepții tratate de program (mesaj, prevenirea blocării sau pierderii de date)
 - **Testarea pe limită** – nu trebuie să producă erori

Testare DBB a datelor

■ Ex: fie MAX cel mai mare număr reprezentat de Windows Calculator

■ Teste la limita superioară:

- **$x+1=MAX$, $MAX+1$, $MAX-1$**
- **$MAX+1$** – excepție care nu provoacă eroare de utilizare: **$1.e+32$**

■ Similar, se definesc teste pt limita inferioară

■ Exemple:

- O casetă acceptă 1-10 caractere
 - teste de limită: 1, 10, 2, 9, 0, 11 caractere
- Un program salvează fișiere. Se vor încerca salvări pt:
 - un fișier cu o înregistrare
 - Un fișier gol
 - Un fișier de dimensiune maximă (impusă de SO sau capacitatea discului)
 - Un fișier care depășește dimensiunea maximă
- Simulator auto:
 - Se vor testa ieșirile în decor, la marginile mediului virtual
- Un program imprimă documente. Se vor imprima:
 - Un document gol
 - Maximul de pagini permise
 - Un document care depășește numărul de pagini permise
 - Un document cu 0 pagini (dacă e posibil)

○ Buffer overflow – depășirea de buffer (eroare celebră de alocare a memoriei insuficiente față de valoarea maximă permisă a datelor)

Testare DBB a datelor

■ Limite interne

- limite nonintuitive, impuse de metoda de programare, platformă, limbaj etc.
- trebuie indicate de programatori, nu apar în specificații

■ Exemple:

- Puterile lui 2
 - fixează dimensiunile spațiilor de memorare (bit, octet, cuvânt)
 - Importante în protocoale (HTTP, TCP, IP) care împachetează datele în pachete de dimensiuni fixe segmentate în octeți cu diferite semnificații (antet, cifre de control, adrese). Se va testa ce se întâmplă dacă datele stocate depășesc segmentele alocate.
- Tabelul ASCII
 - Tabel folosit în identificarea caracterelor prin coduri numerice și stabilirea unei relații de ordine între ele
 - Limite convenționale pentru codurile caracterelor: majusculele au coduri între 65-90, minusculele între 97-122, cifrele între 48-57.
 - Un câmp în care se acceptă doar majuscule va face verificarea intervalului de coduri ASCII, deci caracterele din afara limitelor vor fi cele cu cod 64 și 91
 - Un câmp care acceptă doar cifre, va fi testat și cu caracterele 47 și 58
- Codurile Unicode, similare cu ASCII, pentru setul de caractere universal (extins cu caractere din toate limbile).

Testare DBB a datelor

■ Intrări nule

- = zero, șir vid, dată calendaristică vidă
- Comportament uzual al utilizatorului:
 - Ignorarea completării câmpurilor și apăsarea lui Enter
 - Efect uzual: câmpul va memora în variabile:
 - fie valori nule
 - fie o valoare atribuită în mod invizibil (prin program): valoare implicită, valoare inițială
- Se impun teste pt acest comportament de ignorare a completării intrărilor

■ Intrări invalide (garbage data)

- Apar la testarea negativă, pornind de la premisa comportamentului aberant al utilizatorului
- Programul trebuie
 - să constrângă potențialul distructiv al utilizatorului (validarea formularelor)
 - Să lămurească utilizatorul asupra erorilor de utilizare prin dialog
- CE invalide sunt dificil de rafinat, se pune accent pe imaginația testerului și pe tipurile de date, pornind de la identificarea corectă a domeniului de intrări valide (care să fie ocolite)

Testare DBB a stărilor

■ În timpul execuției, un program trece prin diferite **stări**.

■ **Ex. Paint:**

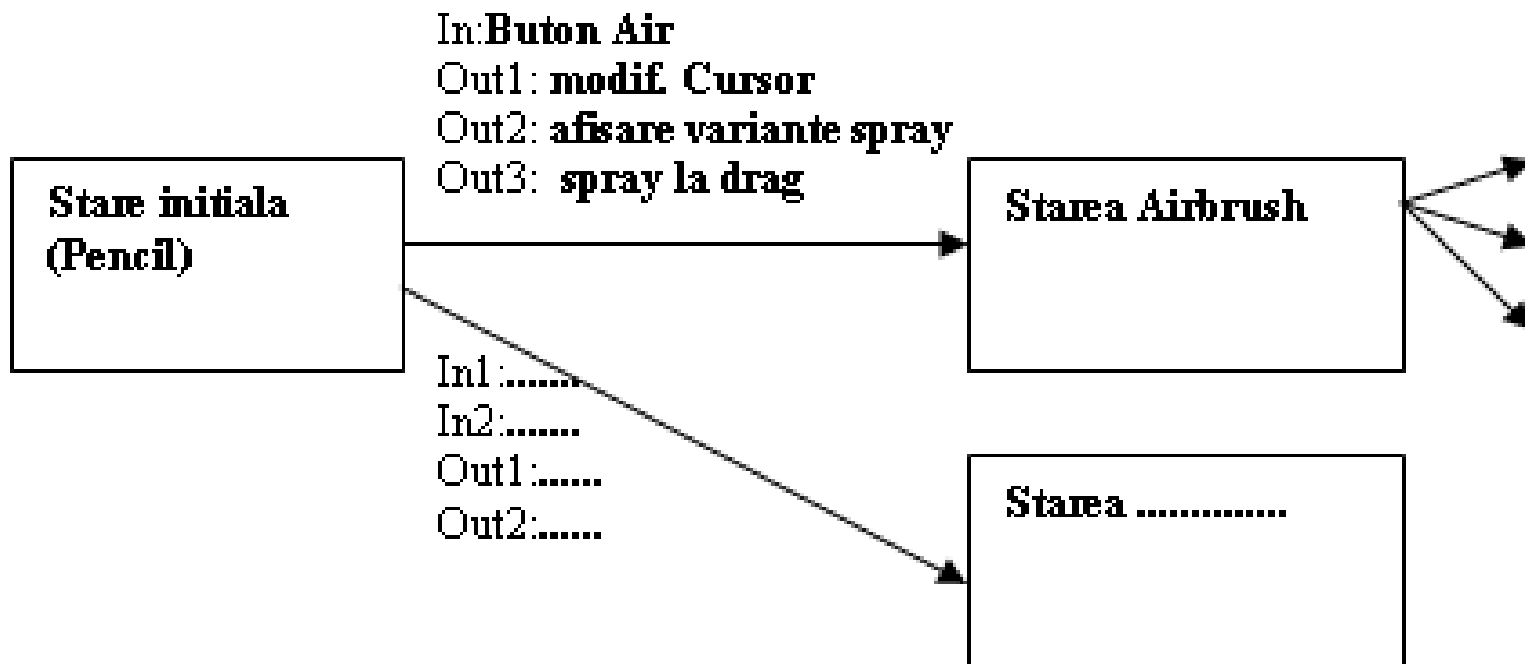
- **Starea inițială – pencil**
 - **Atributele stării:**
 - Pencil activ
 - Cursorul = creion
 - Reacția la drag = trasarea unei linii
 - Pagina albă
 - Culoarea activă negru (culoare fundal alb)
 - Numele documentului e untitled
 - Dimensiunile ferestrei sunt cele rămase la închiderea aplicației
 - **Stări posibile a fi declanșate: airbrush, radieră etc.**
 - **Trecerea (tranziția) în starea airbrush: clic pe butonul Airbrush**
 - **Atributele stării:**
 - Airbrush activ
 - Apar variantele de spray
 - Cursorul=spray
 - Reacția la drag = spray
 - **Fiecare instrument Paint are propria stare asociată**
- Stările posibile pot fi testate complet
- Tranzițiile posibile NU pot fi testate complet
- O aceeași tranziție poate fi declanșată pe mai multe căi
 - Tranziția între două stări neconsecutive poate urma mai multe drumuri
 - Problemă similară cu testarea datelor de intrare



[Testare DBB a stărilor]

■ Selectarea tranzițiilor se face pe baza unei **hărți de tranziții** care indică:

- Toate stările posibile
- Declanșatorii fiecărei stări (elemente de interfață, condiții de declanșare)
- Efectele fiecărei stări (condiții create, modificări în interfață)



Testare DBB a stărilor

■ Recomandări:

- Nici o stare nu va fi ignorată
- Se încearcă gruparea tranzițiilor similare (cu declanșatori și efecte similare) în CE
- Programatorii pot face recomandări privind tranzițiile similare
- Se testează mai întâi scenariile uzuale (testare pozitivă)
- Apoi se testează tranzițiile ce produc mesaje de eroare
- La final se fac teste aleatoare (prin tragere la sorți a două stări și testarea drumului dintre ele)

■ Un caz de testare a stărilor trebuie să verifice toate atributele stării.

- Unele atribute sunt invizibile, trebuie sugerate de programatori.
- Ex: atributul dirty document controlează apariția mesajului *Do you want to save the changes?*
- Atributul DD e activat la operații de modificare a stării inițiale a documentului, e inactiv dacă se fac doar operații de scroll, zoom, configurări care nu modifică documentul etc.)
- Unele programe resetează DD dacă modificările se anulează (Undo), altele consideră documentul modificat chiar dacă anularea a readus documentul în starea inițială.

Testare DBB negativă a stărilor

- Teste multitasking (concurențiale)
 - Verifică modul în care programul gestionează întreruperile și reluarea funcționării
 - Se creează scenarii de întrerupere pt fiecare stare și se urmărește:
 - Ce se întâmplă dacă intrările devin indisponibile (Winamp la scoaterea CD-ului sau ștergerea fișierului rulat)
 - Ce se întâmplă cu ieșirile dacă acestea sunt solicitate în timpul creării sau modificării (modificarea unui text Word în timpul imprimării)
 - Exemple:
 - Salvarea sau deschiderea unui fișier de către programul testat și simultan de alte programe
 - Accesul simultan la o imprimantă, port sau alt periferic
 - Accesul simultan la o bază de date
- Funcționarea simultană a mai multor ferestre cu programul testat

[Testare DBB negativă a stărilor]

■ Teste de stress (suprasolicitarea, worst-case testing)

- Repetarea unei operații pentru detectarea eșecului în eliberarea memoriei RAM
- Executarea programului cu resurse minimale (memorie, procesor slab, modem slab) – stă la baza formulării cerințelor hardware
- Suprasolicitarea programului cu cantități mari de date de intrare (fișiere mari, periferice multe conectate simultan, conexiuni multiple la un server)
- Executarea continuă a programului fără resetare pentru detectarea acumulărilor de marje de eroare.

■ Dificultăți:

- necesită resurse hardware
- managerii nu cred în realismul condițiilor de stres (acestea se creează totuși, adesea în mod accidental)

■ Testele de stress sunt tipuri particulare de teste asupra limitelor!

Alte teste DBB

■ Simularea comportamentului aberant al utilizatorului

- Implicarea în teste a unor persoane fără pregătire și înregistrarea tendințelor psihologice
- Manipularea anormală a interfeței (taste apăsate simultan, clicuri în afara suprafețelor interactive)
- Întreruperea de operații și revenirea irațională la stări anterioare
- Ignorarea mesajelor de eroare și a casetelor de dialog

■ Teste insistente pe baza cazurilor de testare care au detectat deja erori (axioma erorilor grupate)

- Unele erori se “reutilizează” prin copierea în codul sursă a condiției de eroare
- O eroare detectată cu un caz de testare poate fi propagată în toată CE din care s-a extras cazul

■ Simularea comportamentului hackerilor

- Exploatarea vulnerabilităților de securitate

■ Orice tester își construiește de-a lungul carierei o bază de date cu:

- Cazuri de testare care au avut succes
- Erori detectate în trecut de beneficiari
- Erori descrise de recenziiile din presă

Această bază de date e **portofoliul testerului** și pe baza ei va realiza teste suplimentare dacă mai este disponibilă resursa de timp.



- CURS 5
 - Testarea SWB
 - Tipologia erorilor de cod
 - Testarea DWB



[Testarea statică white box]

- Similară cu testarea SBB, doar că în loc să aibă loc asupra specificațiilor, are loc asupra:
 - Documentelor de proiectare (detectează erori timpurii)
 - Codului sursă listat și comentariile (oferă indicii de rafinare a CE pentru testarea DBB)
- Intră în sarcina unor echipe care conțin testeri, programatori, proiectanți și chiar manageri
- Este WB pentru că are acces la codul sursă și detaliile de creare a documentelor testate
- Se realizează în ședințe de recenzare (**recenzii interne**)
- Modelul Big Bang ignoră SWB (ca și SBB) deoarece
 - e considerată costisitoare și neproductivă (deși are un impact major asupra calității)
 - productivitatea e măsurată cantitativ în linii de cod programate și nu prin calitate

Testarea statică white box

■ Recenzii interne (RI)

- Ședințe de dialog în cadrul echipei SWB, cu inspecții amănunțite asupra documentelor de proiectare și a codului sursă
- Comparativ cu SBB, implică o echipă complexă și lucrează cu documente complexe
- Scopurile definitorii ale recenziei interne:
 - Testarea propriu-zisă (Identificarea omisiunilor sau erorilor)
 - Organizarea recenzării:
 - Definirea unui set de reguli pentru structurarea efortului de recenzare (se alocă anumite documente sau linii de cod sursă fiecărei ședințe, fiecărui participant)
 - Definirea de roluri pentru participanți (tester, manager, client simulat etc.)
 - Scrierea unui raport de recenzare
- Absența unuia din cele 4 scopuri face recenzia invalidă.
- **Rezultate subtile ale “organizării recenzării”:**
 - Generarea involuntară, prin dialog, de indicii pentru testerii DBB
 - Asigurarea unui cadru de comunicare între membrii echipei (descurajarea izolaționismului – sindromul pseudoautist la programatori)
 - Calitatea codului sursă recenzat crește dacă programatorul e conștient că va fi supus recenzării
 - Propunerea de soluții și raportul de recenzare e comunicat unitar membrilor echipei (nu apare pericolul desincronizării)

Testarea statică white box

■ Metode RI:

○ P2P (PeerToPeer)

- informale, dialog direct între creatorii programului și recenzori (tester, manager și chiar alți programatori)
- Apare pericolul diluării dialogului așa că trebuie urmărite clar scopurile: identificarea de probleme, organizarea prin reguli și scrierea unui raport

○ Prezentările (Powerpoint, HTML, etc.)

- Programatorul își prezintă în fața echipei de recenzare codul sursă
- Recenzorii îl chestionează asupra elementelor suspecte

○ Inspecțiile formale

- Prezentatorul e altcineva decât creatorul programului recenzat
- Prezentatorul studiază documentul din propria perspectivă neinfluențată, apoi îl prezintă echipei de inspectori
- Inspectorii sunt practic echipa de recenzare organizată după anumite roluri: moderator, acuzator (tester, manager), acuzat (programatorul)
- Se încurajează diversitatea interpretărilor pe marginea erorilor
- Programatorul, în rol de “acuzat” are un rol pasiv, poate corecta interpretările eronate cauzate de prezentator sau inspectori

Testarea statică white box

■ Consultarea standardelor

- ca și în cazul SBB, testerul are posibilitatea ca în cadrul unei recenzii interne să invoce standardele existente la nivel de cod sursă (mai degrabă ghiduri de programare disciplinată)
- Standardele și ghidurile de disciplină a programării oferă:
 - Garanții de fiabilitate
 - Lizibilitatea codului sursă
 - Portabilitatea asigurată de unele standarde de editare a codului sursă
- Exemplu din C++ Programming Guidelines:
 - Instrucțiunea Go To trebuie evitată
 - Justificare prin dificultatea depanării salturilor necondiționate
 - While are prioritate față de Do while (mai puțin atunci când e obligatorie minim o iterație)
 - Justificare prin posibilitatea de a evita procesarea blocului While atunci când nu are loc nici o iterație (optimizarea execuției)
 - Se vor evita elementele limbajului C conflictuale cu C++:
 - Folosirea lui 0 în locul macroului NULL
 - Folosirea lui stdio.h, iostream.h și stream.h în același program

Testarea statică white box

- Standardele și ghidurile de disciplină a codului sursă s-au creat pentru

- Uniformizarea stilului de programare în echipe de programatori
- Facilitarea colaborării (programatori care modifică sau corectează programele altui programator)
- Scăderea efortului în recenzii interne
- Scăderea efortului de depanare

- Alte aspecte afectate de standardele privind codul sursă:

- Stilul comentariilor
- Denumirea variabilelor
- Indentarea instrucțiunilor în editorul de cod sursă:

```
<html><head>xxx</head><body>xxx</body></html>
```

<html>	If then
<head>	xxxxxx
xxx	else
</head>	xxxxxx
<body>	endif
xxx	
</body>	
</html>	

Testarea statică white box

- Organizații implicate în standardizarea programării:

- **American National Standards Institute (ANSI)**, www.ansi.org
- **International Engineering Consortium (IEC)**, www.iec.org
- **International Organization for Standardization (ISO)**, www.iso.ch
- **National Committee for Information Technology Standards (NCITS)**, www.ncits.org

- Ghiduri de disciplină a programării, oferite de:

- **Association for Computing Machinery (ACM)**, www.acm.org
- **Institute of Electrical and Electronics Engineers, Inc (IEEE)**, www.ieee.org

- Producătorii mediilor de programare anexează propriile ghiduri și creează editoare de cod sursă care formatează sintaxa, detectează automat erori de sintaxă, oferă sugestii de scriere corectă a instrucțiunilor și chiar convertesc codul sursă la anumite standarde

Tipuri de erori în codul sursă

■ Erori de referire: - principala cauză a depășirilor de buffer

- Referirea de variabile neinițializate
- Indici de matrici sau șiruri de caractere în afara intervalului care definește dimensiunea
- Erori datorate indexării față de zero în matrici sau șiruri de caractere
- Folosirea de variabile în locul constantelor (ex: la declararea dimensiunilor matricilor)
- Atribuire type mismatch
- Memorie nealocată unui pointer
- Structuri de date referite de funcții și proceduri, procesate eronat față de definiția structurii

■ Erori de declarare:

- Variabile cu dimensiune sau tip declarat eronate
- Variabile inițializate la declarare prin valori invalide
- Conflicte de nume
- Variabile declarate și neutilizate
- Probleme date de domeniul de vizibilitate a variabilelor

Tipuri de erori în codul sursă

■ Erori de calcul

- Operații invalide față de tipul și dimensiunea operanzilor
- Operații type mismatch, nerezolvabile prin conversiile implicite
- Operații ale căror rezultate sunt atribuite unei variabile invalide (ca dimensiune, tip, vizibilitate, domeniu de valori)
- Operații cu depășire aritmetică față de spațiul de reprezentare alocat
- Diviziune cu 0, Modulo 0
- Pierderi de precizie la trunchieri (implicite sau explicite)
- Confuzii privind precedența operatorilor în expresii

■ Erori de comparare

- Confuzii între comparații exclusive (mai mic, sau exclusiv) și inclusive (mai mic sau egal, sau)
- Comparații afectate de precizia datelor în virgulă mobilă (float)
- Confuzii privind precedența operatorilor logici
- Operații logice între operanzi non-booleeni

Tipuri de erori în codul sursă

■ Erori de flux

- Erori de imbricare
- Cicluri infinite (test eronat la ieșirea din ciclu)
- Cicluri ignorate sau întrerupte prematur
- Cicluri contorizate eronat (datorită indexului cu baza zero de obicei)
- Structuri CASE cu domeniul variabile de control segmentat greșit

■ Erori de parametrizare

- Parametri transferați eronat între sursă și subrutină (ca tip, dimensiune)
- Constante transferate ca parametri și modificate de subrutină
- Parametri strict de intrare modificați de subrutină
- Parametru transferați în ordine greșită
- Variabile globale referite eronat în subrutine

Tipuri de erori în codul sursă

■ Erori de intrare-ieșire

- Utilizarea intrărilor și ieșirilor eronat față de formatul lor
- Programul nu tratează excepția de indisponibilitate a perifericului de intrare-ieșire
- Programul nu tratează întreruperea funcționării perifericului
- Erori în mesajele de eroare, casetele de dialog sau lipsa mesajelor de eroare

■ Alte erori:

- Erori de localizare
- Erori de portabilitate
- Erori de compatibilitate hardware și software
- Pseudoerori (avertismente sau casete de dialog nejustificate, redundante, irelevante).

Testarea dinamică white box

- Numită și **testare structurală** – se bazează pe structurile de programare folosite (și pe cunoștințele de programare ale testerului)

- Studiul codului sursă are un impact major asupra **rafinării CE** (se vor cunoaște precis și nu doar intuitiv clasele de intrări tratate diferit de program)

- Avantaje:

- Posibilitatea de a izola și testa module și subrutine (funcții, proceduri)
- Rafinarea CE definite la testarea DBB
- Acces direct la atributele stărilor programului
- Acces direct la variabilele care condiționează tranzițiile
- Observare precisă a ramificațiilor de program antrenate în executarea unui caz de testare

- Testare DWB<>Depanare:

- Scopul DWB nu este corectarea erorilor, ci izolarea și localizarea lor cât mai precisă (la nivel de subrutină sau chiar linie de cod)
- Rezultatul DWB va fi cel mai simplu caz de testare care semnalează eroarea. Pe baza sa, programatorul va face depanarea.

- Instrumentele DWB sunt comune cu cele de depanare: debuggerul compilatorului, analizorul de cod sursă

Testarea dinamică white box

■ Pericolul testării DWB:

- Ignorarea specificațiilor și realizarea testării în raport cu calitatea codului sursă, nu cu scopul produsului
- Apare riscul de a realiza un program foarte precis și corect, care însă face altceva decât ce ar trebui (lipsa de acuratețe)
- Din acest motiv DWB se realizează abia după DBB, iar CE DWB rafinează CE definite prin DBB

■ **Testarea modulară (unit test)** – testează module de program și subrutine, imposibil de izolat dpdv al utilizatorului (deci inaccesibile direct prin testare DBB)

■ **Testarea sistem (system test)** – testează aplicația în ansamblu, felul în care modulele comunică între ele

Testarea dinamică white box

- Instrumente de testare modulară (create de tester):

- **Driver** – programe care lansează în execuție un modul de testat
 - Ex: o funcție preia argumente dintr-o bază de date și returnează un rezultat (altei funcții). Un driver va înlocui funcția apelantă cu o casetă de dialog la care testerul oferă intrări, va conține apelul funcției și va afișa rezultatele returnate de funcție.
 - Asigură executarea unor module care nu pot fi executate izolat de restul aplicației
- **Stuburi** – programe rudimentare care simulează condițiile de mediu oferind intrări și receptând ieșiri de la aplicație
 - Ex: o aplicație preia date de intrare de la un termometru. Termometrul poate fi înlocuit cu un stub care generează date de testare fără a crea condițiile de mediu necesare termometrului
 - De obicei stuburile înlocuiesc periferice (imprimante, scannere, diverși senzori)

Testarea dinamică white box

- Exemplu: funcția C `atoi()` – convertește șiruri de caractere în numere

- Specificațiile funcției:

- **Returnează valoarea numerică obținută din interpretarea numerică a argumentului**
- **Returnează zero dacă argumentul nu poate fi convertit**
- **Returnează valoare nedefinită dacă argumentul provoacă o depășire**
- **Argumentul este de tip string**
- **Structura argumentului trebuie să fie [spații][semn] șir**
- **Spațiile pot fi formate din combinații de Space și Tab și vor fi ignorate**
- **Semnul poate fi + sau –**
- **Șirul va fi un șir de caractere-cifre**
- **Funcția nu interpretează virgula zecimală și nici un alt caracter decât cele indicate până aici**
- **Funcția se oprește din interpretarea argumentului la primul caracter neinterpretabil. Acesta poate fi NULL, care încheie orice șir de caractere C**

Testarea dinamică white box

■ Un driver de test va fi un program improvizat cu funcționalitatea:

- Asigură o casetă de dialog ce permite introducerea de intrări (argumentele funcției) de către tester sau preia date de test dintr-o bază de date
- Apelează funcția
- Afișează sau stochează rezultatele returnate de funcție pentru fiecare caz
- Permite testerului să definească CE pentru argumentele funcției: șiruri de cifre, șiruri cu semn, șiruri cu spații, șiruri neinterpretabile, șiruri care încep cu cifre și conțin caractere neinterpretabile

■ Driverul este necesar deoarece dpdv al utilizatorului e imposibil accesul direct la intrările și ieșirile funcției

■ În final, driverul asigură o testare DBB raportată la specificațiile funcției!

- Raportat la specificații, un tester ar fi tentat să facă teste de limite ASCII "a123", "z123", "A123", "Z123"
- Dacă există acces la codul sursă al funcției, se va observa că funcția testează doar apariția cifrelor, semnelor și spațiilor, celelalte caractere fiind tratate identic în ramura OTHERWISE (DEFAULT) a unei structuri CASE, deci o singură CE pt acestea va fi suficientă



■ CURS 6

- Testarea datelor și algoritmilor prin DWB
- Teste auxiliare
 - configurații, compatibilitate, localizare

Testarea dinamică white box

- **Testarea datelor prin DWB** se realizează prin **urmărirea variabilelor** (instrumentul trace sau watch din debugger)
 - permite observarea tuturor valorilor intermediare pe care le ia o variabilă de la inițializare până la final
 - permite observarea variabilelor intermediare, nu doar a celor de intrare și ieșire
 - permite observarea și rafinarea CE de risc (limite, limite interne) pentru variabile și valori intermediare
 - permite identificarea limitelor interne precum:
 - Pragul de memorie RAM sub care programul se comportă diferit;
 - Modelul matematic folosit de un program pe baza unei condiții interne, inaccesibile utilizatorului
 - Pragurile de delimitare a variantelor structurii CASE
 - Tabelele interne de mapare (ASCII, indecși)
 - permite examinarea formulelor de calcul și formularea de CE care să declanșeze operații riscante (împărțiri cu zero pe baza cărora se definesc CE pentru variabilele de la numitor)
 - permite rafinarea testelor error-forcing prin observarea condițiilor erorilor de utilizare

[Testarea dinamică white box]

- Debuggerele permit și **forțarea valorilor variabilelor**, astfel că testarea DWB nu e limitată de necesitatea de a concepe date de intrare care în mod algoritmic să provoace o anumită situație sau eroare
 - Există riscul ca forțarea variabilelor să se realizeze cu valori nerealiste, pentru care nu există nici o șansă să fie stocate în variabila testată
 - Unele medii de programare întrețin **variabile de eroare** ale căror valori sunt coduri de eroare iar denumirile sunt sugestive privind tipul erorii
 - Forțarea erorilor se poate realiza prin manipularea **codurilor de eroare**, fără a fi necesară provocarea **condițiilor de eroare** ce pot fi costisitoare (ex: erori de funcționare a unor periferice)

Testarea dinamică white box

- Testarea algoritmilor se realizează prin **acoperire de cod** (AC, cod coverage)
 - AC se realizează cu analizoare de cod (incluse în debugger) care măsoară cât anume din algoritm a fost antrenat în fiecare caz de testare
 - AC permite semnalarea porțiunilor de cod cu execuție foarte rară, care e posibil să nu fie surprinse de rafinarea CE anterioară
 - VFox: SET COVERAGE to fisier creează un jurnal care înregistrează AC pentru fiecare caz de testare
 - Tipuri de AC:
 - Acoperirea instrucțiunilor (statement coverage)
 - Acoperirea căilor de execuție (path coverage)
 - Acoperirea condițiilor (condition coverage)

Testarea dinamică white box

Ex: 1: PRINT "Hello World"
2: IF Date\$ = "01-01-2000" THEN
3: PRINT "Happy New Year"
4: END IF
5: PRINT "The date is: "; Date\$
6: PRINT "The time is: "; Time\$
7: END

Caz de **acoperire a instrucțiunilor**: 01-01-2000 (antrenează liniile 1-7)

Acoperirea căilor: sunt necesare două cazuri pentru cele două căi, ramificate de IF: 1-7 (THEN) și 1-2,5-7 (ELSE)

Acoperirea condițiilor: dacă în condiția IF apare o operație SAU, vor fi necesare 4 cazuri pentru testarea tuturor situațiilor: F-F, T-F, F-T, T-T

■ Cele 3 tipuri de AC nu sunt echivalente:

- E posibil un caz de testare care să antreneze toate instrucțiunile dar să nu antreneze toate căile de execuție

■ Cele 3 tipuri de AC se includ una pe alta:

- Cele 4 cazuri de testare care **acoperă condițiile**, asigură și **acoperirea căilor** și **acoperirea instrucțiunilor** (invers nu e adevărat)

[Testarea configurațiilor hardware]

TCH = verificarea compatibilității cu diverse platforme hardware

Portabilitatea – calitatea unei aplicații de a rula pe diferite platforme (hardware sau software) fără efort de conversie din partea utilizatorului

Eterogenitatea tehnologică

- Diversitatea arhitecturilor de calculatoare
- Diversitatea modelelor de procesoare
- Diversitatea performanțelor
- Diversitatea memoriilor
- Diversitatea tuturor componentelor calculatorului (modele, performanțe)
- Diversitatea conectorilor la fiecare componentă (AGP, PCI, USB, SATA, IDE)
- Diversitatea driverelor de instalare
- Diversitatea BIOS-urilor

■ Testerul trebuie să identifice care din domeniile de mai sus vor afecta funcționarea aplicației:

- O aplicație multimedia – afectată de RAM, procesor, modelul și driverele plăcii video, monitoare
- Un procesor de texte – afectat de imprimante, bufferul de imprimantă

○ Un program de rețea – afectat de modemuri, plăci și configurații de rețea

○ Orice aplicație modernă – conexiunea de rețea pentru modulul de on-line registering,

[Testarea configurațiilor hardware]

Standarde hardware = referințe teoretice de la care există abateri majore pentru dobândirea de avantaj competitiv (funcții noi, overclocking, ieftinirea componentelor)

Erorile de configurație hardware

- **Nu țin neapărat de aplicație, ci de modul în care aplicația exploatează hardwareul**
- **Sunt foarte costisitoare – beneficiarul va prefera să schimbe aplicația decât să schimbe hardwareul, consultanța și întreținerea devin neputincioase**
- **Tipuri de erori CH:**
 - **Eroare a aplicației manifestată într-o clasă largă de configurații**
 - **Eroarea va trebui corectată**
 - **Eroare a aplicației manifestată în configurații de excepție**
 - **Eroarea va fi tratată prin ocolire sau prin specificarea sa în cerințele de sistem hardware**
 - **Eroare a platformei hardware revelată de aplicație, cu două situații**
 - **Echipa va încerca să contacteze producătorul hardware pentru a solicita corectarea erorii**
 - **Echipa va încerca să ocolească eroarea dacă popularitatea producătorului e suficientă încât acesta să ignore eroarea sau să o tergiverseze (beneficiarul va atribui vina aplicației care a revelat eroarea, nu platformei)**

[Testarea configurațiilor hardware]

Organizarea TCH:

- Cunoașterea beneficiarului și a platformei hardware pe care o folosește
- Cunoașterea resurselor hardware exploatate intens de aplicație
- Cunoașterea limitelor minime necesare a resurselor exploatate
- Crearea unui tabel cu variabilele:
 - Componente hardware vizate
 - Modele și mărci – selectate după topurile de popularitate publicate anual de revistele cu recenzii hardware (se vor ignora modelele clonate)
 - Drivele – e necesar un test pe driverele generice din SO și unul pe driverele de pe site-ul producătorului
 - Opțiuni și moduri de utilizare (rezoluții la monitoare, culori la imprimante, moduri de transmisie la modemuri) – se va fixa un prag minim de cerințe privind opțiunile (rezoluție minimă, modem minim etc.)
- Pentru 3 din aceste variabile (fără drivere) și pentru fiecare variantă se va defini **un indice de popularitate** și **un indice de actualitate** – managerul va fixa un prag minimal pentru acești indici pentru a reduce gama de configurații hardware la dimensiuni fezabile
- Cunoașterea aspectelor din aplicație executate diferit în funcție de configurație (praguri de memorie RAM la care se schimbă execuția, simbolurile și culorile imprimate etc.)
- Crearea și aplicarea cazurilor de testare
- Cunoașterea erorilor făcute publice de producătorii hardware
- Inițierea și negocierea unui protocol de corectare a erorilor detectate cu producătorii, dacă e posibil

[Testarea configurațiilor hardware]

Cum se procură hardwareul?

- Achiziția de platforme hardware diferite pentru testerii și programatorii sau chiar a unor platforme diferențiate de la tester la tester (managerii au obiceiul de a achiziționa configurații uniforme pt toată echipa de lucru)
- Parteneriate cu producători sau vânzători de hardware, în schimbul reclamei sau certificărilor
- Contractarea unui laborator profesional specializat în TCH
- Închirierea de configurații de testare de la magazine
- Implicarea prietenilor și beneficiarilor în TCH

■ Informații privind TCH:

- <http://developer.apple.com/testing> conține recomandări și legături la laboratoare de testare pentru echipamente Apple;
- <http://www.microsoft.com/wdhc/system/platform> oferă instrumente și recomandări importante pentru testarea sub Windows;
- diverse companii oferă pe site-urile lor certificări ale unor componente hardware sau drivere (ex: certificările Whql de la Microsoft).

[Testarea compatibilității software]

Similar cu TCH, urmărește detectarea problemelor pe variate platforme software (sistem de operare, browser, aplicații comunicante)

TCS e mai puțin costisitor, eterogenitatea sistemelor de operare fiind mai redusă decât cea a hardwareului

Aspecte vizate:

- Cunoașterea platformei software a beneficiarului
- Consultarea standardelor care asigură compatibilitatea software
- Stabilirea versiunilor SO țintă și aplicațiilor
- Testarea interacțiunilor cu alte aplicații:
 - Operații cu Clipboardul (cut, copy, paste, paste special, office clipboard)
 - Operații de import-export
 - Compatibilitate cu SGF și software-ul de rețea (browser, client email)
 - Compatibilitate cu alte versiuni ale aplicației

[Testarea compatibilității software]

Retrocompatibilitate – compatibilitatea cu **versiuni anterioare** ale aplicației (ex: Notepad și formatul txt sunt retrocompatibile până la MS-DOS 1.0)

Compatibilitate în avans – compatibilitate cu **versiuni viitoare!** (se poate asigura prin standardizare – aplicații viitoare vor urmări să respecte standardul)

Deși mai ieftin ca TCH, TCS rămâne un efort costisitor. Se aplică din nou metoda tabelului cu criterii:

- Popularitatea programelor cu care se va asigura compatibilitatea
- Actualitatea programelor
- Tipologia programelor, defalcată pe categorii (editoare text, imagini, SGBD etc.)
- Producătorii programelor

Se atribuie indici de prioritate fiecărui criteriu: pentru un program poate se pune accent pe compatibilitatea cu orice editor de texte indiferent de producător și SO sau cu orice SGBD Microsoft de la versiunea 2000 încoace, etc.

[Testarea compatibilității software]

Standarde și certificări – pot garanta compatibilitatea software

- Compatibilitate de nivel înalt
 - legată de sistemul de operare și soft de platformă (browser, SGF), de regulă nu constrânge funcționarea aplicației dar o garantează
 - testată în laboratoare de certificare
- Compatibilitatea de nivel scăzut
 - afectează funcționarea aplicației
 - legată de extensiile de fișiere, protocoale, formate de import-export
 - testată de tester

Exemplu:

<http://msdn.microsoft.com/certification> oferă detalii privind certificarea compatibilității Windows (indicată de aplicarea logo-ului Windows pe produse soft). Exemple de cerințe:

- suport pt mouse cu 3 butoane
- suport pentru instalarea de pe alte discuri decât C:
- suport pentru nume lungi de fișiere

[Testarea compatibilității software]

Căi de schimb a datelor cu alte aplicații:

- Prin SGF (Save, Open) – se va asigura aderarea la standardele indicate de extensii (txt, jpg, gif, avi etc.);
- Prin conversii de format (Import, Export, Open As, Save As, wizarduri) – se va testa conversia față de versiuni anterioare și față de alte aplicații pe baza unor **documente eșantion**
- Prin tehnologii Windows: Dynamic Data Exchange, Component Object Model, Object Linking and Embedding
 - Spre deosebire de Clipboard, DDE și OLE asigură transfer în timp real (Insert-Object)
 - COM asigură interoperabilitatea obiectelor

[Testarea internaționalizării]

Justificare: pt un sistem de operare, peste jumătate din piața țintă nu sunt vorbitori de engleză

- Internaționalizare parțială – doar produse auxiliare (Help, manual, ambalaj)
- Internaționalizare completă – inclusiv GUI
- Internaționalizare mot-a-mot – nerecomandată, produce interpretări grave
- **Localizare** = internaționalizare prin adaptarea la un **specific geografic local** (lingvistic, cultural etc.), poate include opțiuni noi care nu există în versiunea originală
- Provocare: atât localizatorii cât și testerii de localizare trebuie să fie familiari cu specificul local => se subcontractează în laboratoare sau filiale locale
- Există totuși aspecte ce pot fi testate fără a cunoaște limba locală



[Testarea internaționalizării]

Expandarea textului

- Engleza e una din cele mai concise limbi!
- Expandarea prin traducere are ca efect necesitatea redimensionării obiectelor GUI
- Marjă de siguranță recomandată pentru engleză-română:
 - Dublarea dimensiunii textului de pe butoane și opțiuni
 - Creșterea cu 50% a textului sub formă de propoziții complete
- Testerul trebuie să identifice zonele afectate: trunchieri forțate, concatenări incorecte, rearanjarea automată a obiectelor GUI, alocarea insuficientă a memoriei pentru stringuri



[Testarea internaționalizării]

Setul de caractere

- ASCII reprezintă 256 caractere
- Metode de depășire a limitei:
 - DBCS alocă 2 octeți (65536 caractere) – probleme de compatibilitate între diferite sisteme
 - Paginile ASCII – redefinirea locală a tabelului ASCII cu un set alternativ de 256 caractere (limita de 256 rămâne, se înlocuiesc doar caracterele)
 - Sistemul Unicode (www.unicode.org) – asociază un cod unic oricărui caracter din orice limbă și nu numai (vezi Insert-Symbol în Word)
- Dacă softwareul va fi localizat, se recomandă folosirea de la bun început a setului Unicode, ceea ce va scădea efortul de testare a localizării
- Testerul va crea cazuri de testare pentru caractere extinse: transferul prin rețea a caracterelor extinse, apariția în numele fișierelor a caracterelor extinse, imprimarea caracterelor extinse, transferul prin Clipboard a caracterelor extinse

[Testarea internaționalizării]

Consecințe ale caracterelor extinse

- Operațiile cu stringuri sunt afectate:
 - Sortarea textelor localizate conform regulilor limbii (ordinea literelor cu diacritice: A,Ă,Â,B...)
 - Conversii între majuscule și minuscule – truc frecvent în ASCII: diferența între codul unei majuscule și a minusculei corespondente e 32 (trebuie semnalate cazurile în care a fost folosit trucul pentru adaptarea sa la Unicode)
 - Instrumente spell-checking în editoare de text
 - Concatenările – ordinea componentelor propozițiilor nu e aceeași în toate limbile (ex: în engleză adjectivul apare în fața substantivului)
 - Sensul de afișare a textului (în ebraică se transformă în oglindă toate mesajele)

[Testarea internaționalizării]

Acces GUI

- Taste calde
 - Alt-H : deschide meniul Help
 - În urma traducerii Help – Asistență, trebuie schimbată și combinația de taste!
 - Se vor crea taste calde pentru noua limbă
 - Se vor dezactiva tastele calde din versiunea originală!
- Text grafic – zone de text create ca imagini, nu ca stringuri (ex: B de pe butonul Bold, textele din sigle)
 - În urma traducerii, butoanele B și U își pierd expresivitatea asociată termenilor Bold și Underline
- Recomandare: **înlocuirea valorilor string din codul sursă cu variabile**
 - Mesajele vor fi stocate într-o bază de date
 - Localizatorii vor lucra pe baza de date și nu pe codul sursă
 - O parte din testare se va realiza direct pe baza de date



[Testarea internaționalizării]

Conceptualizarea locală

- exemple: volanul pe dreapta în Anglia, sensul termenului football în SUA, formatul Letter în SUA, unități de măsură, formatul datei calendaristice, forma virgulei zecimale, simboluri financiare, prima zi din săptămână, formatul numerelor de telefon
- conceptele locale pot fi
 - personalizate la nivelul SO (Control Panel – Regional Settings)
 - personalizate la nivelul aplicației (Tools-Options-Measurement Units în Word)
- Ex: Fine Artist, lansat de Microsoft în 1993 nu s-a vândut în unele zone datorită culorii pielii și formei nasului pe care îl avea animația wizard

[Testarea internaționalizării]

Recomandări privind localizarea:

- Specificații de localizare vor indica aspectele de localizat și aspectele locale ignorate;
- **Problemele de TCH** sunt amplificate la localizare – un editor de texte va suporta tastaturi localizate, un program cu opțiuni de imprimare va suporta diferite unități de măsură și formate de hârtie, o aplicație client-server se va adapta la necesitățile tehnologiilor de comunicație adoptate în diferite țări
- **Problemele de TCS** sunt amplificate de localizare – o operație copy-paste va putea transfera o imagine măsurată cu anumite unități într-o aplicație cu alte unități, formatul localizat necesită un efort suplimentar la conversia datelor sau un dialog suplimentar cu utilizatorul
- Se va realiza un **test de localizabilitate (fezabilitatea localizării)** – pentru a estima dacă efortul de localizare e fezabil și scalabil
 - E vorba de testare SWB cu căutarea stringurilor în codul sursă, a textelor grafice, identificarea setului de caractere folosit, a unităților de măsură
 - <http://www.microsoft.com/globaldev>
- Testarea localizării , este afectată de specificațiile de localizare
 - Dacă specs au prevăzut localizarea, aplicația folosește Unicode și baze de date cu stringuri, testarea e facilă
 - Dacă prin localizare s-au redefinit specificațiile și s-a modificat codul sursă, versiunea localizată va fi tratată ca un produs nou (se or relua toate tipurile de teste)



- CURS 7

- Teste auxiliare

- Utilizabilitate, componente auxiliare, securitate

[Testarea utilizabilității]

Scopul unui produs software – utilizarea (oricât de bun ar fi un program, acesta va fi respins dacă necesită efort excesiv în interacțiunea cu utilizatorul)

Utilizabilitatea (usability)

- eficacitatea interacțiunii între GUI și utilizator
- potențialul unui program de a fi utilizat de o gamă cât mai largă de utilizatori cu o curbă de învățare minimă
- pune accent pe erorile de tip 5 (nemulțumiri subiective) – utilizabilitatea scăzută afectează eficiența utilizatorului
- testerul e prima persoană care își asumă rolul utilizatorului și trebuie să construiască simptomatologia GUI

Testarea utilizabilității

■ GUI

- totalitatea elementelor grafice prin care se realizează interacțiunea cu utilizatorul
- supusă unor standarde consacrate de Apple și Microsoft, bazate pe principiul WIMP (Window, Icon, Menu, Pointer)
- evoluează spre interfețe verbale și biometrice
- face subiectul unor cercetări de psihologia utilizatorului și ergonomie, în laboratoare de utilizabilitate:
 - camere video care urmăresc comportamentul userului)
 - software care urmărește traiectoria cursorului
 - sisteme eye-tracking care urmăresc traiectoria privirii prin scanarea globului ocular

■ Parametrii utilizabilității

- Alinierea la standardele GUI privind componenta senzorială (look and feel)
- Caracterul intuitiv
- Consistența
- Flexibilitatea
- Confortul
- Corectitudinea
- Utilitatea

[Testarea utilizabilității]

- **Standardele GUI** – eforturi din partea lui Apple și Macintosh pentru garantarea calității componentei look-and-feel a GUI
- Macintosh Human Interface
 - [http:// developer.apple.com/documentation/mac/HIGuidelines/HIGuidelines-2.html](http://developer.apple.com/documentation/mac/HIGuidelines/HIGuidelines-2.html)
- Windows User Experience
 - [msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/ welcome.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/welcome.asp).
- Aceste ghiduri stabilesc modul de utilizare a obiectelor GUI, clasificarea și designul mesajelor de eroare
- E posibil ca aplicația în lucru să definească și să propună propriile standarde, caz în care acestea vor fi explicitate de specs

Testarea utilizabilității

■ Caracterul intuitiv

- se referă la curba de învățare
- factor fundamental în adoptarea domestică a noilor tehnologii (telefoane, TV, software etc.)
- O GUI intuitivă necesită efort minim de învățare
- O GUI intuitivă e utilizată prin mecanisme psihologice automate (apăsarea unui buton GUI), spre deosebire de o interfață cu linie de comandă
- O GUI intuitivă e bazată pe mecanisme invizibile, cu timpi de așteptare minimi din partea utilizatorului
- O GUI intuitivă nu creează stări de incertitudine pentru utilizator
- Obiective în asigurarea caracterului intuitiv:
 - Minimizarea traiectoriei cursorului (și a privirii) pentru operații uzuale
 - Balans între
 - Numărul de obiecte GUI vizibile simultan pe ecran (pericol de supraîncărcare)
 - Numărul de nivele din arborele opțiunilor (pericol: clicuri numeroase pentru a ajunge la o opțiune)
 - Separarea opțiunilor de utilizare uzuală față de cele excepționale (eventual dezactivarea implicită a celor din urmă)
 - Asigurarea unui sistem Help relevant

Testarea utilizabilității

■ Consistența

- o aceeași operație **să poată** fi realizată de fiecare dată **în același mod** (chiar dacă există mai multe metode de realizare)
- Ex. de inconsistență: până la Windows Me, operația Find se realiza din meniuri diferite în Notepad și în Wordpad
- Ex. De consistență – opțiunile Cut-Copy-Paste apar în orice aplicație în meniul Edit
- Se vor testa:
 - Consistența tastelor calde (F1, Ctrl-C, Ctrl-V etc.)
 - Terminologia (denumirea opțiunilor și meniurilor vor fi aceleași – Open, Save etc.)
 - Consistența limbajului – nu trebuie ca unele casete de dialog să aibă limbaj tehnic iar altele limbaj trivial
 - Consistența plasării obiectelor GUI – butonul OK apare întotdeauna la stânga lui Cancel, forma butoanelor e aceeași

Testarea utilizabilității

■ Flexibilitatea

- o aceeași operație **să poată** fi realizată **în mai multe moduri** (meniul, taste calde).
- mediul de lucru să poată fi personalizat – ascunderea sau activarea elementelor GUI după nevoie (meniul View e folosit în acest scop – vezi MS Office)
- Flexibilitatea GUI are efecte asupra testării DBB
 - Apar tranziții și stări redundante
 - Apar stări opționale peste care utilizatorii avansați trebuie să poată sări (Skip, Next – vezi wizardurile)
 - Apar metode de intrare multiple (mouse, tastatură, drag and drop)
 - Apar metode de ieșire multiple (viziuni multiple asupra documentelor, configurare de grafice etc.)

■ Confortul – dificil de cuantificat, afectat de:

- Impactul GUI să nu fie prea agresiv vizual sau sonor
- Posibilitatea de revenire și recuperare din erori de utilizare (Undo-Redo)
- Notificări asupra operațiilor lente și locației utilizatorului în aplicație (preloader, ferestre de progres, Status Bar)

Testarea utilizabilității

■ Corectitudinea = acuratețea GUI

- Lipsa de acuratețe
 - elemente GUI lipsă (ex: lipsa butonului Cancel), elemente GUI active atunci când ar trebui să fie inactive
 - cauzată de reutilizarea subrutinelor
- Aspecte susceptibile:
 - Discrepanțe între aplicație și materialele de antrenare a utilizatorului (tutoriale, wizarduri)
 - Mesaje de dialog incorecte sau incoerente
 - Discrepanțe WYSIWYG – GUI nu reflectă starea reală (discrepanță între Print Preview și documentul imprimat)
 - Discrepanțe în elementele multimedia (pictograme și sunete incorecte, inconsistente față de operațiile asociate lor)

■ Utilitatea – nu e vorba de utilitatea aplicației, ci de utilitatea GUI în a garanta accesul la aplicație

- Aspecte vizate:
 - Eliminarea elementelor GUI inutile, redundante sau inaccesibile
 - Pericol – semnalarea unor elemente GUI a căror utilitate e subtilă
 - Prin studiul specificațiilor, testerul va identifica elementele care apar în plus și va înțelege exact rostul fiecărui element GUI

Testarea utilizabilității

■ **Accesibilitatea** – calitatea de a oferi persoanelor cu dizabilități metode de acces prin GUI

- 20% din piața unui SO e formată din persoane cu dizabilități
- Unele persoane cu handicap sunt chiar nevoite să folosească calculatorul
- SUA impune prin lege suportul pentru persoane cu dizabilități
- SO moderne (și unele browsere) oferă opțiuni de accesibilitate: Control Panel – Accessibility Options
 - Sticky Keys – permite apăsarea consecutivă în loc de simultană a tastelor Ctrl, Shift, Alt
 - Filter Keys – previne tastarea repetată accidentală
 - Toggle Keys – declanșează semnale sonore la apăsarea tastelor Lock
 - SoundSentry – creează un semnal vizual la fiecare sunet
 - ShowSounds – componentă reutilizabilă în programe pentru a afișa textul vorbit
 - High Contrast – culori contrastante
 - MouseKeys – manevrarea cursorului prin taste
 - SerialKeys – înlocuirea tastaturii cu un dispozitiv de intrare alternativ pentru simularea tastării

www.microsoft.com/enable - detalii privind accesibilitatea Windows

Testarea componentelor auxiliare

■ **Componente auxiliare** ale produsului software, vizate de tester

- Ambalaj – conține capturi de ecran, cerințe de sistem, listă de funcționalități, informații de copyright
- Materiale de marketing
- Documente de garanție și registration, on-line sau off-line
- Documentul EULA (End User License Agreement) – document avizat juridic, inclus în kitul de instalare, prin care utilizatorul se poate angaja să nu intenteze procese legate de eventuale erori
- Etichete cu numere de serie, coduri de licență și activare
- Instrucțiuni de instalare, configurare, manual, fișier Read Me
- Documentația Help interogabilă
- Tutoriale și wizarduri conectate la Help
- Exemple și șabloane de documente
- Mesaje de eroare documentate

■ **Rolul componentelor auxiliare**

- Îmbunătățesc utilizabilitatea (antrenarea utilizatorului)
- Sincronizează acuratețea aplicației cu așteptările utilizatorului
- Permit ignorarea unor tipuri de erori (fixează cerințe de sistem, antrenează utilizatorul în evitarea erorilor, impunerea juridică prin citirea EULA la instalare)



Testarea componentelor auxiliare

- **E posibil ca testarea componentelor auxiliare să releve chiar erori ale aplicației**
- **Lista de verificare la testarea auxiliară:**
 - Nivelul audienței – limbajul tutorialurilor, help-ului, manualului de utilizare trebuie să fie la nivelul beneficiarului
 - Terminologia și abrevierile să fie adaptate domeniului beneficiarului
 - Se vor detecta lipsurile și ambiguitățile
 - Se vor detecta dezacorduri între aplicație și componentele auxiliare
 - Se va studia acuratețea documentației și a datelor de contact
 - Se vor testa hiperlegăturile (cuprins, Help)
 - Se vor detecta lipsurile în etapizarea wizardurilor
 - Se vor detecta erori de interogare a Helpului
 - Se vor detecta erori în capturile de ecran și în textul grafic
 - Se va verifica acuratețea exemplurilor
 - Se va realiza spell-checking pe documentație

Testarea componentelor auxiliare

■ Factori care împiedică testarea auxiliară

- I se alocă resurse minimale
- Persoanele care creează componentele auxiliare nu sunt experți nici în aplicație, nici în domeniul său
- Componentele auxiliare off-line sunt costisitoare și întârzie lansarea produsului

Testarea securității

- **Produs securizat** = produs care protejează confidențialitatea, integritatea și disponibilitatea informațiilor
- **Vulnerabilitate** = caracteristica unui produs care permite altora decât proprietarului acces la nivelul de confidențialitate, integritate și disponibilitate
- Motivele hackerilor
 - Prestigiul - hacking benign orientat spre încălcarea protecțiilor și nu spre accesarea de resurse protejate
 - util în testarea securității;
 - Curiozitatea – implică accesarea de resurse protejate dar nu și exploatarea lor
 - read-only hacking
 - Utilizarea – implică exploatarea resurselor protejate în interesul hackerului, fără a bloca accesul proprietarului la resurse
 - spyware, viruși de e-mail
 - Vandalizarea – implică blocarea accesului proprietarului la resurse fără beneficii directe pentru hacker
 - modificarea GUI (frecvent la site-uri Web), distrugerea datelor sau blocarea funcționalității (denial-of-service)
 - Furtul – exploatarea resurselor în interesul hackerului și blocarea accesului proprietarului la resurse
 - furtul de numere de card, bunuri și servicii, date personale

Testarea securității

■ Threat modelling

- modelarea amenințărilor, proces similar cu recenzarea internă
- Echipa de recenzare identifică zonele aplicației predispuse la amenințări
- Etape:
 - Asamblarea echipei, cu specialiști în securitate, testeri și programatori care să elimine vulnerabilitățile
 - Identificarea țintelor – baze de date cu numere de card, resurse de distribuire spam, imaginea organizației (prin site branding)
 - Modificarea arhitecturii aplicației (document de proiectare) pentru evidențierea nodurilor de comunicare între module
 - Stabilirea limitelor de autorizare pe baza nodurilor dintre module
 - Studiul diagramelor de tranziție pentru identificarea căilor de acces la date
 - Identificarea datelor și modulelor care necesită criptare și parole
 - Identificarea amenințărilor, pe baza țintelor vizate, a limitelor de autorizare și a căilor de acces
 - Documentarea amenințărilor
 - Clasificarea amenințărilor după modelul DREAD
(msdn.microsoft.com/library)

[Testarea securității]

■ Modelul DREAD

- Atribuește note de la 1 la 3 celor 5 atribute ale unei amenințări:
 - Dauna potențială (estimată financiar)
 - Reproductibilitatea amenințării prin exploatare repetată
 - Exploatabilitatea (dificultatea tehnică de exploatare a vulnerabilității)
 - Afectarea (numărul de utilizatori afectați)
 - Descoperirea (dificultatea descoperirii vulnerabilității)
- Notele se însumează pentru a obține indicele DREAD
- Se fixează un prag de relevanță sub care amenințările vor fi ignorate de manager



[Testarea securității]

■ Pt. Tester

- vulnerabilitate = eroare
- securitatea = funcționalitate implicită sau explicitată în specificații
- testarea securității = testare negativă (limitele aplicației)
- www.securityfocus.com – detalii la zi privind clasificarea vulnerabilităților



[Testarea securității]

■ Buffer overflow:

```
1: void copiereBuffer(char * sursa) {  
2:   char dest[100];  
3:   int a = 123;  
4:   int b = 456;  
5:   strcpy(dest, sursa);  
...  
6: }  
7: void validparola()  
8: {  
9: /*  
10:  cod de validare parola  
11: /*  
12: }
```

- Functia copiaza un string în altul
- Destinatia are dimensiunea fixată
- Sursa nu are dimensiunea fixată
- Dacă sursa e mai mare decât memoria alocată destinației, surplusul va fi stocat la adresele adiacente (depășirea de buffer!)
- La adresele adiacente însă, s-au stocat variabilele a, b și funcția de validare a parolei
- Surplusul va suprascrie aceste adrese adiacente
- Surplusul poate ajunge la adrese de memorie care sunt executate automat
- Dacă hackerul are acces la stringul sursă, poate să introducă în el un surplus de caractere care să fie interpretate și executate ca instrucțiuni de asamblare care vor înlocui informațiile adiacente (datele funcției de validare)



[Testarea securității

- Buffer overflow – virușii ascunși în date:
 - Se consideră că virușii apar în programe executabile
 - În 2004 a apărut primul virus în fișiere JPG, prin exploatarea unei depășiri de buffer
 - JPG memorează, în plus față de imagine, comentarii care încep cu valoarea FFFE, urmată de lungimea comentariului minus 2, presupus a fi număr pozitiv
 - Dacă lungimea e zero, rezultatul -2 e interpretat ca fiind 4 GB, deci fișierul JPG poate fi însoțit de 4GB de comentarii care suprascriu zone masive din memoria internă
 - Comentariul poate conține instrucțiuni în asamblare distructive care să suprascrie zone de memorie executate automat



[Testarea securității]

- 2002 – Microsoft a lansat o inițiativă de identificare și înlocuire a funcțiilor C și C++ susceptibile la buff.ovf.
- S-au creat funcții securizate (safe string functions) cu beneficiile:
 - Fiecare funcție primește ca argument dimensiunea bufferului destinație
 - Fiecare funcție finalizează stringurile cu caracterul NULL, chiar și la trunchiere forțată
 - Fiecare funcție returnează un status pentru diagnosticarea succesului operației
 - Fiecare funcție are două versiuni, ASCII și Unicode, diferențiate prin memoria alocată unui caracter
- Exemple:
 - Funcții de cocatenare: Strcat, wcscat, Strncat, wcsncat;
 - Funcții de copiere: Strcpy, wcsncpy, Strncpy, wcsncpy;
 - Funcții de determinare a lungimii: Strlen, wcslen;
 - Funcții de creare a șirurilor formate: Sprintf, swprintf, Vsprintf, vswprintf.



[Testarea securității

- **Vulnerabilități implicite** – nu necesită atac din partea hackerilor, sunt rezultate ale neglijenței utilizatorilor
 - Pachetele cookie
 - Parolele memorate pe PCuri publice
 - Istoricul navigărilor
- Toate acestea sunt **date latente** care ar trebui șterse cu regularitate de utilizator dar sunt de regulă ignorate
- Rolul datelor latente este să îmbunătățească utilizabilitatea (completarea automată a formularelor, memorarea parolei, funcționarea butonului Back în browser)
- Un browser creator de date latente trebuie să se achite de obligația de a oferi căi de ștergere sau criptare a acestora, răspunderea revenind astfel utilizatorului



[Testarea securității]

- **Datele latente** pot fi dovezi juridice.
 - Ștergerea de fișiere și formatarea logică a discului nu șterg efectiv informația, ci o marchează în vederea suprascrierii
 - La crearea de noi fișiere, o parte din aceste zone sunt suprascrise, dar o parte rămân libere și informația “ștearsă” constituie **date latente** ce pot fi recuperate
 - Un fișier se stochează într-un număr întreg de clustere și de regulă ultimul cluster incomplet acoperit păstrează în continuare date latente care nu vor fi suprascrise de alte fișiere
 - Datele latente pot fi fișiere temporare, parole și alte date utile în investigarea juridică
 - Pirații software care își formatează discurile sunt verificați prin extragere de date latente cu instrumente software specializate
 - Modul de tratare a latenței datelor dpdv al testerului va trebui indicată în specificații



- CURS 7
 - Teste auxiliare
 - Utilizabilitate, componente auxiliare, securitate

[Testarea utilizabilității]

Scopul unui produs software – utilizarea (oricât de bun ar fi un program, acesta va fi respins dacă necesită efort excesiv în interacțiunea cu utilizatorul)

Utilizabilitatea (usability)

- eficacitatea interacțiunii între GUI și utilizator
- potențialul unui program de a fi utilizat de o gamă cât mai largă de utilizatori cu o curbă de învățare minimă
- pune accent pe erorile de tip 5 (nemulțumiri subiective) – utilizabilitatea scăzută afectează eficiența utilizatorului
- testerul e prima persoană care își asumă rolul utilizatorului și trebuie să construiască simptomatologia GUI

Testarea utilizabilității

■ GUI

- totalitatea elementelor grafice prin care se realizează interacțiunea cu utilizatorul
- supusă unor standarde consacrate de Apple și Microsoft, bazate pe principiul WIMP (Window, Icon, Menu, Pointer)
- evoluează spre interfețe verbale și biometrice
- face subiectul unor cercetări de psihologia utilizatorului și ergonomie, în laboratoare de utilizabilitate:
 - camere video care urmăresc comportamentul userului)
 - software care urmărește traiectoria cursorului
 - sisteme eye-tracking care urmăresc traiectoria privirii prin scanarea globului ocular

■ Parametrii utilizabilității

- Alinierea la standardele GUI privind componenta senzorială (look and feel)
- Caracterul intuitiv
- Consistența
- Flexibilitatea
- Confortul
- Corectitudinea
- Utilitatea

[Testarea utilizabilității]

- **Standardele GUI** – eforturi din partea lui Apple și Macintosh pentru garantarea calității componentei look-and-feel a GUI
 - Macintosh Human Interface
 - [http:// developer.apple.com/documentation/mac/HIGuidelines/HIGuidelines-2.html](http://developer.apple.com/documentation/mac/HIGuidelines/HIGuidelines-2.html)
 - Windows User Experience
 - msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/welcome.asp.
 - Aceste ghiduri stabilesc modul de utilizare a obiectelor GUI, clasificarea și designul mesajelor de eroare
 - E posibil ca aplicația în lucru să definească și să propună propriile standarde, caz în care acestea vor fi explicitate de specs

Testarea utilizabilității

■ Caracterul intuitiv

- se referă la curba de învățare
- factor fundamental în adoptarea domestică a noilor tehnologii (telefoane, TV, software etc.)
- O GUI intuitivă necesită efort minim de învățare
- O GUI intuitivă e utilizată prin mecanisme psihologice automate (apăsarea unui buton GUI), spre deosebire de o interfață cu linie de comandă
- O GUI intuitivă e bazată pe mecanisme invizibile, cu timpi de așteptare minimi din partea utilizatorului
- O GUI intuitivă nu creează stări de incertitudine pentru utilizator
- Obiective în asigurarea caracterului intuitiv:
 - Minimizarea traiectoriei cursorului (și a privirii) pentru operații uzuale
 - Balans între
 - Numărul de obiecte GUI vizibile simultan pe ecran (pericol de supraîncărcare)
 - Numărul de nivele din arborele opțiunilor (pericol: clicuri numeroase pentru a ajunge la o opțiune)
 - Separarea opțiunilor de utilizare uzuală față de cele excepționale (eventual dezactivarea implicită a celor din urmă)
 - Asigurarea unui sistem Help relevant

Testarea utilizabilității

■ Consistența

- o aceeași operație **să poată** fi realizată de fiecare dată **în același mod** (chiar dacă există mai multe metode de realizare)
- Ex. de inconsistență: până la Windows Me, operația Find se realiza din meniuri diferite în Notepad și în Wordpad
- Ex. De consistență – opțiunile Cut-Copy-Paste apar în orice aplicație în meniul Edit
- Se vor testa:
 - Consistența tastelor calde (F1, Ctrl-C, Ctrl-V etc.)
 - Terminologia (denumirea opțiunilor și meniurilor vor fi aceleași – Open, Save etc.)
 - Consistența limbajului – nu trebuie ca unele casete de dialog să aibă limbaj tehnic iar altele limbaj trivial
 - Consistența plasării obiectelor GUI – butonul OK apare întotdeauna la stânga lui Cancel, forma butoanelor e aceeași

Testarea utilizabilității

■ Flexibilitatea

- o aceeași operație **să poată** fi realizată **în mai multe moduri** (meniul, taste calde).
- mediul de lucru să poată fi personalizat – ascunderea sau activarea elementelor GUI după nevoie (meniul View e folosit în acest scop – vezi MS Office)
- Flexibilitatea GUI are efecte asupra testării DBB
 - Apar tranziții și stări redundante
 - Apar stări opționale peste care utilizatorii avansați trebuie să poată sări (Skip, Next – vezi wizardurile)
 - Apar metode de intrare multiple (mouse, tastatură, drag and drop)
 - Apar metode de ieșire multiple (viziuni multiple asupra documentelor, configurare de grafice etc.)

■ Confortul – dificil de cuantificat, afectat de:

- Impactul GUI să nu fie prea agresiv vizual sau sonor
- Posibilitatea de revenire și recuperare din erori de utilizare (Undo-Redo)
- Notificări asupra operațiilor lente și locației utilizatorului în aplicație (preloader, ferestre de progres, Status Bar)

Testarea utilizabilității

■ Corectitudinea = acuratețea GUI

- Lipsa de acuratețe
 - elemente GUI lipsă (ex: lipsa butonului Cancel), elemente GUI active atunci când ar trebui să fie inactive
 - cauzată de reutilizarea subrutinelor
- Aspecte susceptibile:
 - Discrepanțe între aplicație și materialele de antrenare a utilizatorului (tutoriale, wizarduri)
 - Mesaje de dialog incorecte sau incoerente
 - Discrepanțe WYSIWYG – GUI nu reflectă starea reală (discrepanță între Print Preview și documentul imprimat)
 - Discrepanțe în elementele multimedia (pictograme și sunete incorecte, inconsistente față de operațiile asociate lor)

■ Utilitatea – nu e vorba de utilitatea aplicației, ci de utilitatea GUI în a garanta accesul la aplicație

- Aspecte vizate:
 - Eliminarea elementelor GUI inutile, redundante sau inaccesibile
 - Pericol – semnalarea unor elemente GUI a căror utilitate e subtilă
 - Prin studiul specificațiilor, testerul va identifica elementele care apar în plus și va înțelege exact rostul fiecărui element GUI

Testarea utilizabilității

■ **Accesibilitatea** – calitatea de a oferi persoanelor cu dizabilități metode de acces prin GUI

- 20% din piața unui SO e formată din persoane cu dizabilități
- Unele persoane cu handicap sunt chiar nevoite să folosească calculatorul
- SUA impune prin lege suportul pentru persoane cu dizabilități
- SO moderne (și unele browsere) oferă opțiuni de accesibilitate: Control Panel – Accessibility Options
 - Sticky Keys – permite apăsarea consecutivă în loc de simultană a tastelor Ctrl, Shift, Alt
 - Filter Keys – previne tastarea repetată accidentală
 - Toggle Keys – declanșează semnale sonore la apăsarea tastelor Lock
 - SoundSentry – creează un semnal vizual la fiecare sunet
 - ShowSounds – componentă reutilizabilă în programe pentru a afișa textul vorbit
 - High Contrast – culori contrastante
 - MouseKeys – manevrarea cursorului prin taste
 - SerialKeys – înlocuirea tastaturii cu un dispozitiv de intrare alternativ pentru simularea tastării

www.microsoft.com/enable - detalii privind accesibilitatea Windows

Testarea componentelor auxiliare

■ **Componente auxiliare** ale produsului software, vizate de tester

- Ambalaj – conține capturi de ecran, cerințe de sistem, listă de funcționalități, informații de copyright
- Materiale de marketing
- Documente de garanție și registration, on-line sau off-line
- Documentul EULA (End User License Agreement) – document avizat juridic, inclus în kitul de instalare, prin care utilizatorul se poate angaja să nu intenteze procese legate de eventuale erori
- Etichete cu numere de serie, coduri de licență și activare
- Instrucțiuni de instalare, configurare, manual, fișier Read Me
- Documentația Help interogabilă
- Tutoriale și wizarduri conectate la Help
- Exemple și șabloane de documente
- Mesaje de eroare documentate

■ **Rolul componentelor auxiliare**

- Îmbunătățesc utilizabilitatea (antrenarea utilizatorului)
- Sincronizează acuratețea aplicației cu așteptările utilizatorului
- Permit ignorarea unor tipuri de erori (fixează cerințe de sistem, antrenează utilizatorul în evitarea erorilor, impunerea juridică prin citirea EULA la instalare)



Testarea componentelor auxiliare

- **E posibil ca testarea componentelor auxiliare să releve chiar erori ale aplicației**
- **Lista de verificare la testarea auxiliară:**
 - Nivelul audienței – limbajul tutorialurilor, help-ului, manualului de utilizare trebuie să fie la nivelul beneficiarului
 - Terminologia și abrevierile să fie adaptate domeniului beneficiarului
 - Se vor detecta lipsurile și ambiguitățile
 - Se vor detecta dezacorduri între aplicație și componentele auxiliare
 - Se va studia acuratețea documentației și a datelor de contact
 - Se vor testa hiperlegăturile (cuprins, Help)
 - Se vor detecta lipsurile în etapizarea wizardurilor
 - Se vor detecta erori de interogare a Helpului
 - Se vor detecta erori în capturile de ecran și în textul grafic
 - Se va verifica acuratețea exemplurilor
 - Se va realiza spell-checking pe documentație

Testarea componentelor auxiliare

■ Factori care împiedică testarea auxiliară

- I se alocă resurse minimale
- Persoanele care creează componentele auxiliare nu sunt experți nici în aplicație, nici în domeniul său
- Componentele auxiliare off-line sunt costisitoare și întârzie lansarea produsului



[Testarea securității

- **Produs securizat** = produs care protejează confidențialitatea, integritatea și disponibilitatea informațiilor
- **Vulnerabilitate** = caracteristica unui produs care permite altora decât proprietarului acces la nivelul de confidențialitate, integritate și disponibilitate

■ **Confidențialitatea**

Este definită conform ISO (organizația mondială pentru standardizare) ca fiind asigurarea accesabilității informației doar de către persoanele autorizate în accesarea și folosirea acestor informații.

Cea mai eficientă metodă de asigurare a confidențialității informațiilor este criptografia. Astfel, datele cu caracter privat și sensibile sunt encriptate de către sursă și decriptate de către persoana autorizată.

Asigurarea confidențialității este critică în aplicațiile care folosesc tranzacții bancare online. De asemenea confidențialitatea este necesară și în menținerea caracterului privat al datelor cu caracter personal.

■ **Integritatea datelor și a informațiilor**

Are ca obiectiv asigurarea că informațiile ajung la persoanele autorizate nealterate, în formă identică cu informațiile de la sursă, iar modificările asupra datelor se fac doar de către persoanele care au autorizație.

Integritatea informațiilor poate fi compromisă de către persoane în mod accidental sau în mod voit. De asemenea alterarea datelor în mod accidental se poate întâmpla și din cauza disfuncționalității sistemelor informatice.

De aceea se impun algoritmi de verificare a integrității datelor – HASH.

■ Disponibilitatea

Se referă la asigurarea accesului la informație, atunci când este cerută și implică în principiu disponibilitatea sistemelor informatice de a oferi informația. Internetul a perfecționat acest principiu prin introducerea conceptului de disponibilitate permanentă (High Availability), fiind o necesitate absolută a traseului informție-utilizator. Sistemele informatice care oferă informație în internet trebuie să asigure disponibilitatea permanentă, să prevină prin soluții tehnice întreruperile de serviciu din cauze diverse (căderi de tensiune în rețeaua electrică, disfuncționalități hardware, etc) și prevenirea atacurilor de tip Denial of Service.



- Pe lângă cele 3 principii enumerate, confidențialitate, integritate, disponibilitate, în 2002, Don Parker a propus adițional alte 3: posesia, autenticitatea și utilitatea.
- **Non-repudierea**
În termeni juridici non-repudierea înseamnă imposibilitatea ca persoanele angajate în schimbul de informații să nege trimiterea, respectiv recepționarea informației.
- **Autentificare vs. Autorizare**

Testarea securității

■ Threat modelling

- modelarea amenințărilor, proces similar cu recenzarea internă
- Echipa de recenzare identifică zonele aplicației predispuse la amenințări
- Etape:
 - Asamblarea echipei, cu specialiști în securitate, testeri și programatori care să elimine vulnerabilitățile
 - Identificarea țintelor – baze de date cu numere de card, resurse de distribuire spam, imaginea organizației (prin site branding)
 - Modificarea arhitecturii aplicației (document de proiectare) pentru evidențierea nodurilor de comunicare între module
 - Stabilirea limitelor de autorizare pe baza nodurilor dintre module
 - Studiul diagramelor de tranziție pentru identificarea căilor de acces la date
 - Identificarea datelor și modulelor care necesită criptare și parole
 - Identificarea amenințărilor, pe baza țintelor vizate, a limitelor de autorizare și a căilor de acces
 - Documentarea amenințărilor
 - Clasificarea amenințărilor după modelul DREAD
(msdn.microsoft.com/library)

[Testarea securității]

■ Modelul DREAD

- Atribuește note de la 1 la 3 celor 5 atribute ale unei amenințări:
 - Dauna potențială (estimată financiar)
 - Reproductibilitatea amenințării prin exploatare repetată
 - Exploatabilitatea (dificultatea tehnică de exploatare a vulnerabilității)
 - Afectarea (numărul de utilizatori afectați)
 - Descoperirea (dificultatea descoperirii vulnerabilității)
- Notele se însumează pentru a obține indicele DREAD
- Se fixează un prag de relevanță sub care amenințările vor fi ignorate de manager



[Testarea securității]

■ Pt. Tester

- vulnerabilitate = eroare
- securitatea = funcționalitate implicită sau explicitată în specificații
- testarea securității = testare negativă (limitele aplicației)
- www.securityfocus.com – detalii la zi privind clasificarea vulnerabilităților
- PenTesting – Kali Linux



[Testarea securității]

■ Buffer overflow:

```
1: void copiereBuffer(char * sursa) {  
2:   char dest[100];  
3:   int a = 123;  
4:   int b = 456;  
5:   strcpy(dest, sursa);  
...  
6: }  
7: void validparola()  
8: {  
9: /*  
10:  cod de validare parola  
11: /*  
12: }
```

- Functia copiaza un string în altul
- Destinatia are dimensiunea fixată
- Sursa nu are dimensiunea fixată
- Dacă sursa e mai mare decât memoria alocată destinației, surplusul va fi stocat la adresele adiacente (depășirea de buffer!)
- La adresele adiacente însă, s-au stocat variabilele a, b și funcția de validare a parolei
- Surplusul va suprascrie aceste adrese adiacente
- Surplusul poate ajunge la adrese de memorie care sunt executate automat
- Dacă hackerul are acces la stringul sursă, poate să introducă în el un surplus de caractere care să fie interpretate și executate ca instrucțiuni de asamblare care vor înlocui informațiile adiacente (datele funcției de validare)



[Testarea securității

- Buffer overflow – virușii ascunși în date:
 - Se consideră că virușii apar în programe executabile
 - În 2004 a apărut primul virus în fișiere JPG, prin exploatarea unei depășiri de buffer
 - JPG memorează, în plus față de imagine, comentarii care încep cu valoarea FFFE, urmată de lungimea comentariului minus 2, presupus a fi număr pozitiv
 - Dacă lungimea e zero, rezultatul -2 e interpretat ca fiind 4 GB, deci fișierul JPG poate fi însoțit de 4GB de comentarii care suprascriu zone masive din memoria internă
 - Comentariul poate conține instrucțiuni în asamblare distructive care să suprascrie zone de memorie executate automat



[Testarea securității]

- 2002 – Microsoft a lansat o inițiativă de identificare și înlocuire a funcțiilor C și C++ susceptibile la buff.ovf.
- S-au creat funcții securizate (safe string functions) cu beneficiile:
 - Fiecare funcție primește ca argument dimensiunea bufferului destinație
 - Fiecare funcție finalizează stringurile cu caracterul NULL, chiar și la trunchiere forțată
 - Fiecare funcție returnează un status pentru diagnosticarea succesului operației
 - Fiecare funcție are două versiuni, ASCII și Unicode, diferențiate prin memoria alocată unui caracter
- Exemple:
 - Funcții de cocatenare: Strcat, wcscat, Strncat, wcsncat;
 - Funcții de copiere: Strcpy, wcsncpy, Strncpy, wcsncpy;
 - Funcții de determinare a lungimii: Strlen, wcslen;
 - Funcții de creare a șirurilor formate: Sprintf, swprintf, Vsprintf, vswprintf.



[Testarea securității]

- **Vulnerabilități implicite** – nu necesită atac din partea hackerilor, sunt rezultate ale neglijenței utilizatorilor
 - Pachetele cookie
 - Parolele memorate pe PCuri publice
 - Istoricul navigărilor
- Toate acestea sunt **date latente** care ar trebui șterse cu regularitate de utilizator dar sunt de regulă ignorate
- Rolul datelor latente este să îmbunătățească utilizabilitatea (completarea automată a formularelor, memorarea parolei, funcționarea butonului Back în browser)
- Un browser creator de date latente trebuie să se achite de obligația de a oferi căi de ștergere sau criptare a acestora, răspunderea revenind astfel utilizatorului



[Testarea securității

- **Datele latente** pot fi dovezi juridice.
 - Ștergerea de fișiere și formatarea logică a discului nu șterg efectiv informația, ci o marchează în vederea suprascrierii
 - La crearea de noi fișiere, o parte din aceste zone sunt suprascrise, dar o parte rămân libere și informația “ștearsă” constituie **date latente** ce pot fi recuperate
 - Un fișier se stochează într-un număr întreg de clustere și de regulă ultimul cluster incomplet acoperit păstrează în continuare date latente care nu vor fi suprascrise de alte fișiere
 - Datele latente pot fi fișiere temporare, parole și alte date utile în investigarea juridică
 - Pirații software care își formatează discurile sunt verificați prin extragere de date latente cu instrumente software specializate
 - Modul de tratare a latenței datelor dpdv al testerului va trebui indicată în specificații

■ Motivele hackerilor

- Prestigiul - hacking benign orientat spre încălcarea protecțiilor și nu spre accesarea de resurse protejate
 - util în testarea securității;
- Curiozitatea – implică accesarea de resurse protejate dar nu și exploatarea lor
 - read-only hacking
- Utilizarea – implică exploatarea resurselor protejate în interesul hackerului, fără a bloca accesul proprietarului la resurse
 - spyware, viruși de e-mail
- Vandalizarea – implică blocarea accesului proprietarului la resurse fără beneficii directe pentru hacker
 - modificarea GUI (frecvent la site-uri Web), distrugerea datelor sau blocarea funcționalității (denial-of-service)
- Furtul – exploatarea resurselor în interesul hackerului și blocarea accesului proprietarului la resurse
 - furtul de numere de card, bunuri și servicii, date personale

“Portret robot” al atacatorului [cybernetic - tipul "criminalității gulerelor albe"]

- - bărbat cu vârsta cuprinsă între 15 și 45 de ani, având un statut social bun, fără antecedente penale, inteligent și motivat. În multe cazuri, autorul este chiar salariat al întreprinderii atacate, sau cunoaște modul de funcționare a sistemului atacat.

[

]

- Parole
- Distributia initiala a parolelor
- Ingineria sociala



- CURS 8
 - Testarea Web
 - Testarea automată



[Testarea Web

■ Particularități ale aplicațiilor Web

- Caracter distribuit – beneficiarii sunt toți vizitatorii potențiali ai site-ului
- Relația cu beneficiarii nu e contractuală
- Procesul de producție Web e de regulă mai simplu
- Mediul de execuție Web este mai complex, cu eterogenitate hardware și software puternică (client, server, limbaje multiple, plug-inuri)
- Stările sunt paginile aplicației iar tranzițiile sunt date de hiperlegături și schimbul de date cu serverul (formulare)
- Accentul cade pe utilizabilitate

■ Testarea Web conține toate abordările prezentate până aici, la care se adaugă testarea grey-box



[Testarea Web black box]

Textul se testează similar cu documentația auxiliară

- Nivelul audienței
- Terminologia
- Spell-checking

....la care se adaugă:

- Actualitatea informație conformă cu ritmul de actualizare a site-ului
- Corectitudinea paginii de contact
- Corectitudinea titlului ferestrei (preluat de semnele de carte)
- Corectitudinea etichetele Tooltips
 - definite prin marcatorul ALT, care constituie totodată și o funcție de accesibilitate – textul ALT e citit de browserele audio
- Comportamentul formatărilor la redimensionarea ferestrei browserului



[Testarea Web black box]

Hiperlegăturile – referințe text sau grafice. Aspecte vizate:

- Corectitudinea ancorelor destinație și a cadrelor țintă
- Formatarea hiperlegăturilor pentru evidențierea celor vizitate, celor active etc.
- Funcționarea hiperlegăturilor de tip mailto
- Identificarea paginilor orfan pe baza hărții site-ului

Elementele grafice. Aspecte vizate:

- Erori de afișare, dimensionare
- Lipsa elementelor grafice datorită erorilor din căi
- Erori de încadrare a textului la redimensionarea ferestrei browserului
- Erori de optimizare prin comprimarea insuficientă a elementelor grafice (testare dial-up)

Formulare. Aspecte vizate:

- Dimensionarea și ordinea câmpurilor
- Modul de validare
- Funcționarea lui Enter la confirmare
- Probleme specifice testării datelor black box (limite, limite interne, valori nule și valori garbage)

Elementele programate (contorul vizitelor, motor de căutare intern, efecte marquee) vor fi testate ca programe de sine stătătoare

Testarea Web white box și grey box

WB implică studiul scripturilor. Aspecte vizate

- Conținutul dinamic asigurat de scripturi client (roll-over, validare)
- Paginile generate dinamic și accesul la baza de date prin scripturi server
- Performanța serverului prin simularea unui număr mare de conexiuni simultane
- Securitatea serverului în conformitate cu recomandările făcute deja

GB implică studiul codului HTML

- Practic e vorba de metoda BB suplimentată cu studiul codului din browser (View-Source)
- Nu e white box pentru că nu e vorba de studiul algoritmilor (HTML e un limbaj de formatare)



[Testarea Web CH și CS]

- Sunt mai complexe având în vedere complexitatea platformelor beneficiarilor (browsere, versiuni de browsere, SO).
 - Versiunile de browsere impun două strategii de dezvoltare Web:
 - Folosirea codului sursă comun, suportat de toate browserele;
 - Folosirea codului sursă degradat – cod sursă redundant oferă variante diferite ale unei pagini Web în funcție de versiunea și tipul de browser,
- Plug-in-urile din browser măresc complexitatea platformelor și asigură compatibilitate cu elemente multimedia
- Opțiunile de personalizare ale browserului afectează funcționarea, specificațiile de securitate, modul de gestionare a textului ALT, modificarea fonturilor
- Rezoluțiile terminalelor afectează modul de afișare a site-urilor (accent puternic pe site-uri afișabile în terminale mobile – telefon, PDA, etc.)
- Diversitatea modemurilor asigură diversitatea performanțelor
- Testarea CH și CS va porni de la sondajele de popularitate tehnologică (www.websidestory.com)



[Testarea utilizabilității Web]

Utilizabilitate Web = foarte slabă.

Diferențe mari între fluiditatea experienței de utilizare față de aplicațiile desktop.

Motive:

- Aplicațiile Web au evoluat din documente Web prin adoptarea treptată a unor tehnologii eterogene în implementarea interactivității;
- În aplicațiile desktop, apelurile de procedură sunt locale – apelatul și apelatorul sunt în aceeași memorie:
- În aplicațiile Web, unele apeluri de procedură sunt la distanță
 - Browserul apelează procesări executate la server, deci intervine latența rețelei
 - Evenimente din browser sunt tratate de server prin comunicări HTTP
- **Refresh redundant** (cea mai mare parte a unei pagini e regenerată dinamic atunci când de fapt sunt necesare doar anumite date din baza de date)
- Crearea aplicațiilor Web este la îndemâna oricui, majoritatea nefiind create de profesioniști

[Testarea utilizabilității Web]

Optimizarea utilizabilității Web: trecerea de la Thin Client la Rich Client

Thin Client – clasic (HTML-PHP) :

- Accentul pus pe scripturi server (procesări intense la server, slabe la client)
- Browserul are doar rol de formatare a conținutului livrat de server
- Accesarea unei hiperlegături sau buton are ca efect o nouă comunicare cu serverul
- Fiecare transfer de la server conține părți redundante (care există deja la client dar sunt regenerate prin Refresh redundant)

Rich Client – nou (AJAX, Flash, XML):

- Accentul pus pe scripturi client (procesări intense la client, slabe la server)
- Browserul are rol de a executa o aplicație, nu de a formata conținut
- Unele hiperlegături sunt procesate direct la client, fără a contacta serverul
- Are loc un transfer masiv inițial la client, apoi se transferă doar date necesare de la server (fără conținut redundant)
- Latența rețelei are efecte mai scăzute (se reduce comunicarea cu serverul)



[Testarea utilizabilității Web]

Listă de erori Web frecvente (www.useit.com):

- Utilizare nejustificată a tehnologiilor de ultimă oră (imature)
 - Utilizatorul e atras de utilizabilitate și conținut, nu de spectacolul aplicației
- Inconsistența stilistică (formatări eterogene)
 - Poate fi prevenită prin foile de stiluri CSS externe
- Agresivitatea vizuală
 - Excesul de mișcare solicită vederea periferică – animații, text derulant
- Derularea orizontală a paginilor
- Mecanismele de navigare trebuie grupate în partea de sus
 - Pentru minimizarea derulărilor verticale
- Hiperlegăturile non-standard (neevidențiate)
 - Efect negativ asupra timpului de reacție a utilizatorului
- Informațiile neactualizate
 - Întreținerea unui site este un efort constant față de întreținerea aplicațiilor desktop
- Actualizările frecvente sunt susceptibile la erori

[Testarea utilizabilității Web]

Listă de erori Web frecvente - continuare:

- Performanța slabă, coroborată cu latența de rețea
 - 0.1 s e timpul de reacție imperceptibil, 10 s e marja de toleranță medie la așteptare
- Lipsa mecanismelor de navigare în site – nu sunt suficiente butoanele browserului
 - Implică o hartă a site-ului și un motor de căutare intern
- Toate paginile trebuie să conțină un link la pagina inițială și o metodă de a sugera locul paginii curente în site
 - Garantează simțul locației la utilizator
- URL-uri complicate sau cu caractere dificil de tastat (sedila)
 - Unii utilizatori memorează URL-uri ale paginilor relevante din site pentru acces direct
- Utilizarea cadrelor
 - Se recomandă evitarea lor, deoarece afectează utilizabilitatea, crearea semnelor de carte, simțul locației
 - Se recomandă structurarea paginii prin tabele invizibile, deschiderea de noi pagini în ferestre noi (în loc de cadru țintă) și interacțiune asigurată prin scripturi client



[Testarea automată]

- crește eficiența - teste mai rapide, paralelisme în testare
- crește efectivitatea (acuratețea și precizia) – se elimină oboseala și rutina
- asigură fezabilitatea unor teste nefezabile în mod manual
- reduce necesitatea unor cunoștințe privind implementarea

TA nu înlocuiește testerul!

Abordări

- TA invazivă – afectează codul sursă al aplicației și poate induce erori
- TA noninvazivă – monitorizează și măsoară

[Instrumente de testare automată]

■ **Monitoare**

- analizoare de coverage, debuggere, programe de monitorizare a traficului, analizoare de protocol etc.
- de regulă sunt invazive, introduc “bruiaje” în codul sursă sau sunt compilate odată cu codul sursă
- sniffere – noninvazive, monitorizează starea datelor în nodurile unei rețele

■ **Drive-re**

- Fișiere BAT, task scheduler, unele macrouri
- Programe ce controlează lansarea în execuție a programului (modulului) testat

■ **Stuburi**

- Accentul cade pe generarea de intrări și receptarea de ieșiri, nu pe controlul execuției programului
- Simulează condiții de mediu nefezabile (înlocuiește periferice), oferind intrări pe căi alternative

■ **Emulatoare**

- Sunt stuburi cu GUI, prin care testerul poate monitoriza activitatea

[Instrumente de testare automată]

■ Instrumente de stress

- Folosesc drivere, stuburi sau agenți software pentru a executa în mod repetat aplicația și a-i livra cantități mari de intrări
- Măsoară performanța obținută (benchmarking)
- ex. Microsoft Stress Utility permite limitarea resurselor hardware (memorie, procesor) pentru simularea condițiilor de stres
- **Cod degradat** = programe capabile să se adapteze la criza de resurse, oferind căi alternative de executare în funcție de constrângerile de platformă

■ Generatoare de bruiaje

- Similare cu instrumentele de stress, dar simulează un mediu instabil
- MSU are și opțiuni de variație a limitelor impuse asupra resurselor
- Există programe care injectează interferențe în comunicații pentru a testa gestionarea bruiajelor

[Instrumente de testare automată]

■ Instrumente de uz general

- Editoare de text (pentru spell-checking)
- Spreadsheeturi (pentru grafice și măsurători)
- Baze de date (raportarea erorilor)
- Programe de file comparison și gestiunea capturilor de ecran
- Camere video, cronometru etc.



[Macrouri

■ Macro

- tip particular de scripturi orientate pe simularea interacțiunilor între utilizator și GUI
- Înregistrează o succesiune de acțiuni GUI (tastare, clicuri)
- Se obțin prin
 - Programare (Limbeje: AutoIT, VisualTest, VBA în MS Office)
 - Înregistrare (MacroExpress, MacroMachine, MacroRecorder, macrouri MS Office, Test Harness în VFox)
- Proprietăți:
 - Nume macro
 - Declanșator (combinație de taste)
 - Setul de acțiuni înregistrate (numai mouse, numai taste etc.)
 - Viteza de rulare a macroului (accelerată, decelerată) – trebuie să se sincronizeze cu performanța aplicației
 - Poziția coordonatelor – relativ la tot ecranul, relativ la fereastra curentă
- Macrourele nu realizează verificări ale rezultatelor!



[Macrouri

Exemplu Macro Magic:

1: Calculator Test #2

2: <<EXECUTE:C:\WINDOWS\SYSTEM32\Calc.exe~~~~>>

3: <<LOOKFOR:Calculator~~SECS:5~~>>

4: 123-100=

5: <<PROMPT:The answer should be 23>>

6: <<CLOSE:Calculator>>

- Execută programul
- Așteaptă 5 secunde apariția ferestrei cu titlul Calculator
- Tastează 123-100
- Oferă caseta de dialog PROMPT
- Închide aplicația

Avantajul programării – se rezolvă problemele de sincronizare cu performanța aplicației (vezi așteptarea ferestrei!)

[Macrouri]

Exemplu Visual Test (inclus în Rational Development Studio):

FOR i=1 TO 100

PLAY "sir de caractere"

NEXT i

- Tastează stringul de 100 de ori

PLAY "{MOVETO 10,10}"

PLAY "{DBLCLICK}"

- Deplasează cursorul și acționează dublu clic

wButtonClick ("OK")

- Acționează un clic pe butonul OK, indiferent de poziția sa în fereastră

[Macrouri]

Verificarea rezultatelor testelor:

- Prin capturi automate de ecran la momente cheie ale execuției macroului
 - Permit folosirea instrumentelor de file comparison la nivel de bit sau pixel
 - Compararea capturilor trebuie să țină cont de orice bruijaj posibil (modificarea orei pe desktop, modificarea culorilor pixelilor)
- Prin variabile de control conectate la starea butoanelor, la valoarea casetelor de editare etc.
- Prin baze de date în care se salvează rezultatele testelor, supuse apoi la file comparison

Produse comerciale de TA:

- www.sdtcorp.com
- www.mercury.com



[Monkey testing]

- Postulat nedemonstrabil: Dacă 1000 de maimuțe tastează într-un editor de texte 1000 de ani, există o probabilitate mare să rescrie din întâmplare o piesă de Shakespeare
- Consecință: Dacă 1000 de maimuțe interacționează cu GUI timp de 1000 de ani, există o probabilitate mare să se detecteze toate erorile aplicației
- Monkey testing
 - testare arbitrară prin simularea comportamentului aberant al utilizatorului cu o viteză și repetabilitate accelerată
 - realizează rapid teste aleatoare nefezabile prin metode manuale
 - se bazează pe drivere și macrouri

[Monkey testing]

■ Dumb monkeys

- Programe ce realizează clicuri și tastări aleatoare cu o probabilitate dorită, pe suprafața de ecran dorită
- Surprind accidental erori nedetectabile prin clase de echivalențe și procese raționale
- Surprind erori de acumulare și de stress prin executarea repetată a unei aplicații
- Pot surprinde elementele care reacționează la clic când nu ar trebui să reacționeze (neglijate adesea de testeri umani concentrați asupra ferestrei active)
- Pot înregistra condiții de eroare într-un fișier jurnal
- Pot conține un mecanism crash recognition, de resetare a calculatorului și reluare a testării după apariția unei erori



[Monkey testing

■ Dumb monkeys

○ Exem plu Visual Test:

1: RANDOMIZE TIMER

2: FOR i=1 TO 10000

3: PLAY "{CLICK "+STR\$(INT(RND*800))+", "+STR\$(INT(RND*600))+"}"

4: PLAY CHR\$(RND*256)

5: NEXT I

Într-un ciclu repetat de 10000 ori, apasă un clic la o poziție aleatoare în rezoluție 800x600 și introduce aleator un caracter ASCII (indicat prin cod)

Monkey testing

■ Smart monkeys

- Programe sensibile la condițiile de mediu
- Pot exploata harta de tranziții pentru a ști ce obiecte GUI sunt accesibile în fiecare stare
- Pot folosi indici de prioritate sau probabilitate a accesării obiectelor GUI
- Pot lua decizii pe baza condițiilor de eroare

■ Factori care influențează TA:

- Flexibilitatea specificațiilor implică flexibilitatea aplicației și afectează reutilizabilitatea TA
- TA nu poate înlocui calitățile testerului
- TA oferă posibilități reduse de verificare a rezultatelor, necesită intervenție umană
- Apare pericolul devierii interesului testerului spre instrumentele TA în locul aplicației de testat
- Instrumentele TA trebuie la rândul lor testate
- Instrumentele invazive pot crea erori (erorile TA trebuie confirmate în absența TA)



- CURS 9
 - Testarea colectivă
 - Planificarea testării



[Testarea colectivă]

- Testeri diferiți vor surprinde erori diferite
- Testarea colectivă elimină monotonia, distribuie efortul și creează un **efect de rețea** în detectarea erorilor
- Tipuri de testare colectivă:
 - **Bug-bashing**
 - exploatează independența punctelor de vedere
 - poate include testeri profesioniști și neprofesioniști
 - profesioniștii vor avea rol de recenzori și confirmatori
 - neprofesioniștii pot fi familiarizați cu anumite tipuri de erori (de utilizabilitate), fiind mai apropiați de perspectiva clientului
 - **Subcontractarea**
 - Se aplică atunci când nu există resursele de testare necesare
 - Domenii predispuse: TCH, TCS, utilizabilitate, localizare
 - Testerii locali vor colabora cu laboratorul subcontractat pentru confirmarea testelor



[Testarea colectivă]

■ Beta testing

- se realizează de către grupuri focus (beneficiari potențiali sau testerii neprofesioniști)
- e un proces de validare și nu unul de verificare (se testează față de beneficiar, nu față de specificații)
- poate avea diverse scopuri: recenzarea în presă, teste de utilizabilitate, TCH, TCS
- Necesită o planificare BT:
 - Grupul focus va fi selectat conform cu scopul testării (recenzie în presă, utilizabilitate, TCH, TCS)
 - Testerii beta vor fi superficiali, e necesar un protocol formal cu deadlineuri și raportări formale
 - Testerii beta se vor concentra pe anumite tipuri de erori, nu pot înlocui testarea profesionistă
 - Testerii profesioniști vor izola erorile semnalate de testerii beta prin cazuri de testare precise
 - Testarea beta e târzie, constrânsă de timp și cu posibilități de corectare reduse

Planificarea testării

- **Standardul de test planning: IEEE 829-1998**

(<http://standards.ieee.org>)

- **Planul de testare**

- prescrie acoperirea, metoda, resursele și programul activităților de testare
- identifică elementele testate, funcționalitatea testată, sarcinile de testare, personalul alocat și riscurile
- nu calitatea documentului PT contează, ci utilitatea lui
- nu formalizarea contează (IEEE recomandă un șablon) ci realismul planificării și felul în care planul comunică sarcinile celor implicați

Rubricile PT, conform IEEE

- **Generalități** – aspecte ce par evidente (trebuie explicitate deoarece evidențele teoretice sunt cele care creează probleme de comunicare)

- Scopul PT
- Produsul testat și scopul produsului
- Versiunea testată și statutul său (produs nou, upgrade, etc.)
- Stadiul produsului (modul, aplicație integrată, build alfa)
- Nivelul de calitate urmărit – **punct sensibil!**
 - Marketerii vor urmări un nivel de utilizabilitate
 - Programatorii vor invoca nivelul de actualitate tehnologică sau complexitate
 - Managerii vor urmări reducerea costurilor
 - Nivelul de calitate trebuie stabilit la nivelul PT, nu ulterior și va fi impus pe toată durata testării
 - Nivelul de calitate va fi precizat prin cuantificare: număr de erori detectate în 24 ore, un prag de fiabilitate, un prag de performanță, un indice de utilizabilitate etc.



[Rubricile PT, conform IEEE]

■ **Identificatori** – necesari referințelor care se realizează în textul PT

- Oamenii implicați (date de contact, posibilități de comunicare)
- Instrumentele software implicate (atât produsul testat cât și alte instrumente, locul în care se vor stoca documentele de testare)
- Locațiile (clădiri, laboratoare, săli, locul în care se stochează documentația de testare)
- Echipamentele hardware (și modul lor de obținere, sursele – subcontracte, închirieri etc.)

Rubricile PT, conform IEEE

■ **Termeni și informații cheie** – necesari uniformizării semantice și fixării unor deadlineuri comune

- **Definiția erorii**
- **Build** – cod compilat livrabil de către programatori (executabil)
- **TRD (Test Release Document)** – documentul care însoțește un build, pe care se indică ce s-a modificat, ce s-a corectat și alte date despre build
- **Alpha** – un build timpuriu, în scop demonstrativ, pentru care se fixează un nivel de calitate urmărit mai scăzut decât cel final;
- **Beta** – un build ajuns în faza de testare beta (ajuns la unii beneficiari), pentru care trebuie indicat scopul testării beta (recenzie, utilizabilitate etc.)
- **Spec complete** – deadline după care specificațiile nu vor mai fi modificate
- **Feature complete** – deadline după care aplicației nu i se vor mai adăuga funcționalități
- **Comitetul de gestiune a erorilor** – comitet format din managerul de testare, managerul de proiect, managerul de dezvoltare, managerul relației cu clienții care trebuie să decidă ce erori vor fi corectate, tratate sau ignorate.

Rubricile PT, conform IEEE

- **Responsabilități generale** – componentele livrabile (deliverables) și sarcinile care afectează testarea, prezentate în mod tabelar pe oameni implicați:



Livrabile	Manager Proiect	Programatori	Marketing	...
Cerințele utilizatorilor	X		X	
Specificații	X			
Cod sursă		X		
.....				
Raportul comitetului de gestiune a erorilor	X			



- **Acoperirea** – aspectele testate și aspectele netestate (lăsate pentru subcontractare sau testare beta).

[Rubricile PT, conform IEEE]

■ Fazele testării (reflectate eventual într-o diagramă Gantt)

- Prima fază e chiar PT
- Se stabilește ordinea tipurilor de teste realizate (a specificațiilor, a codului sursă, a variate module, a integrării modulelor, stress, utilizabilitate, TCH, TCS etc.)
- Pentru fiecare fază se arată criteriul de intrare (condiția ce trebuie îndeplinită pt începerea fazei)
- Pentru fiecare fază se arată criteriul de ieșire din fază (de obicei publicarea unui raport sau o recenzie internă)
- În absența fazelor, avem o testare haotică (Big Bang)

■ Strategia de testare – BB, WB, manual, automat, pozitiv, negativ, error forcing

[Rubricile PT, conform IEEE]

■ Necesarul de resurse

- Personal – oameni, nivel expertiză, program de lucru, mod de contractare
- Echipamente
- Spațiu de lucru – locații, organizare
- Software
- Subcontractări – criteriu de selecție, cost
- Consumabile și obiecte de inventar – manuale, discuri, telefoane, acces Internet sau alte căi de comunicare între membrii

Rubricile PT, conform IEEE

- **Responsabilități de testare** – detaliate pe testeri, pe faze și strategii

Aspecte	A	B	C	...
Formatare și ortografie	X		X	
Utilizabilitate	X			
Help și tutoriale		X		
Cod sursă				
Teste de stres	X			

[Rubricile PT, conform IEEE]

- **Programarea testelor** – detalierea fazelor și reflectarea efortului de testare în timp (neproportional cu efortul de programare!)

- Poate afecta data de livrare a produsului
- Poate avea ca efect amânarea unor componente ale aplicației
- Programul testelor e influențat de efortul de programare
- Pentru a evita responsabilitatea întârzierilor, programul testelor se exprimă în date relative:
 - În loc de: testul X se finalizează la data de ...
 - Se va folosi: textul X va dura două zile de la livrarea modului

- **Cazurile de testare** – prezentate la modul general (tipuri de cazuri de testare – limite, valori nule, stress), alături de o referință spre dosarul cazurilor de testare și responsabilul cu acesta (care se întocmește în cursul testării și detaliază precis cazurile de test)



[Rubricile PT, conform IEEE]

- **Rapoartele de testare** – vor fi indicate tipurile de rapoarte întocmite și modul de urmărire a erorilor (manual, printr-o bază de date on-line sau o foaie Excel)
- **Metricile** – vor fi indicate modurile de cuantificare a testării și modul de colectare a datelor pentru cuantificare (de obicei prin interogarea sistemului de urmărire a rapoartelor de eroare). Exemple de metrici:
 - Erori detectate pe zi
 - Lista de erori vizate
 - Clasificarea erorilor vizate
 - Erori detectate de un tester
 - Erori detectate pe modul
- **Riscurile** – se identifică impactul celorlalte activități asupra echipei, riscurile de întârziere, riscurile de expertiză, riscurile de buget

Planificarea cazurilor de testare (PCT)

- **PCT** – Dosar care detaliază rubrica privind cazurile de testare din PT, se realizează un dosar PCT pentru fiecare componentă testată

- Scopuri PCT:

- Organizarea și clasificarea cazurilor de test (teste de portofoliu, teste specifice situației)
- Recenzarea cazurilor de test
- Reutilizabilitatea testelor
- Urmărirea facilă a rezultatelor pe cazuri de test
- Crearea unor dovezi de testare, solicitate în unele domenii ca măsură de asigurare a calității

- Dosarul PCT conține 3 documente:

- Proiectarea CT
- Specificarea CT
- Procedeul de aplicare CT

- La Dosarul PCT e important formalismul, precizia sa, deoarece scopul de bază este **reutilizarea PCT** și înțelegerea sa de către diferiți testeri (procedeul CT e descris chiar algoritmic)

Planificarea cazurilor de testare (PCT)

■ **Proiectarea CT** – detaliere a strategiei de testare **pentru o componentă a aplicației**. Conține

- Identificator al proiectului CT și referințe spre identificatori ai altor documente (PT, specs, proceduri CT)
- Descrierea componentei testate
- Precizarea componentelor testate auxiliar datorită interacțiunilor
- Precizarea componentelor netestate (au fost testate deja sau sunt înlocuite cu stuburi și drivere)
- Tehnicile de testare folosite (BB, WB, automat, manual, monkey etc.)
- Modul de verificare a rezultatelor pe fiecare tehnică de testare
- Lista identificatorilor cazurilor de test, asociați cu clasa de echivalență pe care o reprezintă și referințe spre specificarea și procedeul CT aferente)
- Criteriul de admisie-respingere a componentei testate

Planificarea cazurilor de testare (PCT)

- **Specificarea CT** – e nucleul dosarului CT, indică exact intrările testate, ieșirile așteptate, condițiile de mediu **pentru un CT!**. Documentul conține:
 - Identificator unic al cazului (prin care să poată fi referit de proiectarea CT și de procedeul CT)
 - Descrierea componentei testate mai detaliat decât la proiectarea CT
 - cu referințe spre specs
 - cu detalierea tipului de test (ex: test DBB de depășire a limitei superioare la adunarea Windows Calculator);
 - Specificarea intrărilor și condițiilor necesare executării CT;
 - Specificarea ieșirilor și condițiilor așteptate în urma executării CT;
 - Condițiile de mediu necesare executării CT (hardware, software, personal);
 - Dependențele față de alte specificații CT.
- Crearea unui astfel de document pentru fiecare CT poate fi costisitor dar poate fi impus de beneficiar și domeniul aplicației. Acolo unde nu se impune detalierea de tipul un document pt un CT, se poate construi un format tabelar în care fiecare rând corespunde unui CT și fiecare coloană uneia din rubricile indicate.

Planificarea cazurilor de testare (PCT)

■ **Procedeul CT (Case script)** – descrie algoritmul de executare **pentru un CT** (implementează specificarea CT). Relația *spec CT- case script* este de tip m-n, adică același caz (spec CT) poate referi mai multe proceduri, același procedeu poate fi aplicat pe mai multe cazuri (spec CT). Documentul conține:

- Identificator unic al procedeiului (referit de PT, Proiectul CT și Specs CT)
- Scopul procedeiului – cu referințe la specs CT pe care le deservește
- Condiții de mediu pentru executarea procedeiului
- Algoritmul procedeiului:
 - Metoda de stocare a rezultatelor testului
 - Metoda de pregătire a testului
 - Metoda de lansare în execuție (linie de comandă, driver, clic)
 - Pașii testului
 - Metoda de măsurare a rezultatelor (cronometru, observare, file comparison)
 - Metoda de suspendare și reluare a testului, dacă e posibil
 - Metoda de finalizare a testului
 - Metoda de restaurare a condițiilor precedente testului
 - Modul de tratare a situațiilor de excepție

Planificarea cazurilor de testare (PCT)

■ Exemplu de procedeu CT:

- Identificator: ProcWinCalc001
- Scop: testarea cazurilor cu specificațiile CazWC001 până la CazWC55 (acești identificatori indică în specs CT că sunt cazuri de testare a operațiilor din Windows Calculator)
- Condiții speciale: doar cele prevăzute de specs CT
- Algoritm:
 - Înregistrarea rezultatelor – foaia teste.xls
 - Pregătirea testului: formatarea discului și instalarea unei copii Win XP SP2 (evitarea interferențelor)
 - Executarea testului: Start-Programs-Accessories-Calculator
 - Pași:
 - Tastarea intrărilor prevăzute de specs CT
 - Compararea cu rezultatele așteptate (din specs CT)
 - Repetarea testului folosind mouse-ul
 - Compararea cu rezultatele așteptate
 - Metodă de observare: vizual
 - Metodă de suspendare sau oprire: închiderea ferestrei WC
 - Metodă de reluire: testele întrerupte se vor considera oprite
 - Modul de restaurare a condițiilor: formatarea discului
 - Excepții: în caz de resetare se notează condițiile erorii și se relansează testul

Planificarea cazurilor de testare (PCT)

- Pentru înregistrarea rezultatelor testării se va folosi un **jurnal de testare**:
 - Foaie Excel
 - SGBD
 - O aplicație Web specializată: Bugzilla, Mantis
- Jurnalul trebuie să grupeze testele în suite de testare (seturi interdependente de CT)
- Rubricile tabelului-jurnal vor indica numele și identificatorul cazului (din specs CT), rezultatul cazului (admis-respins), data de executare a procedurii și eventual o referință la un tabel de descriere a erorilor



■ CURS 10

- Raportarea testării
- Măsurarea rezultatelor testării



[Raportarea testării]

- De calitate raportării depinde gradul de înțelegere a erorilor, efortul și disponibilitatea de corectare a lor din partea altor membri ai echipei
- Există o tendință implicită de a subestima sau ignora erorile și consecințele lor
- Unul din motivele ignorării erorilor – **raportarea deficitară**
- **Raport de eroare** – documentul pe baza căruia comitetul de gestiune a erorilor decide asupra modului de tratare a lor
- Calitatea raportării depinde **izolarea și reproducerea erorilor**

Raportarea testării

■ Principiile raportării:

- Erorile trebuie raportate devreme pentru a împiedica propagarea și a aloca timp de corectare
- Erorile trebuie descrise precis, prin informații care vor sta la baza deciziei comitetului:
 - Cu detalii stricte, fără redundanțe, cu exemplificarea datelor de intrare
 - Cu indicii de reproductibilitate (dacă s-a detectat comportamentul regulat al erorii, deci dacă poate fi confirmată prin repetarea CT)
 - Cu delimitare clară între erori (se recomandă un raport pe eroare) din perspectiva simptomelor (efectelor asupra utilizatorului) și nu din perspectiva sursei de eroare
 - Cu precizarea procedurii care a detectat eroarea și frecvența de manifestare a erorii (de fiecare dată când..., o dată pe oră... etc,)
- Erorile trebuie raportate obiectiv, impersonal
- Erorile raportate trebuie urmărite de către tester
 - Pentru a confirma corectarea lor
 - Pentru a evita raportarea aceleiași erori de mai multe ori, mai ales dacă s-a decis ignorarea sa

Raportarea testării

- Izolarea și reproducerea erorilor - urmărește obținerea de indicii privind sursa erorii și reproductibilitatea sa
- Eroare izolată = eroare pe care testerul e capabil să o producă oricând (deoarece cunoaște precis condițiile de eroare)
- O eroare trebuie raportată chiar dacă nu e izolată
 - Se vor detalia eforturile de izolare care au eșuat, care pot oferi indicii programatorilor în scopul izolării prin depanare
- Factori care ajută la izolarea erorilor:
 - Testarea colectivă
 - Înregistrarea video a testării
 - Înregistrarea de macrouri
 - Studiarea condițiilor concurențiale (activitatea în rețea, activitatea discurilor și perifericelor)
 - Erorile care nu mai apar după resetarea calculatorului se datorează alocării memoriei, caz în care se aplică teste repetate
 - Erorile care nu depind de condiții concurențiale sunt de regulă erori a căror manifestare depinde de tranziția între stări
 - Testele de stress pot izola erori de acces la resurse hardware

TCH pot izola erorile cauzate de hardware

Raportarea testării

- Atributele erorii - se vor atribui de către tester
 - **Severitatea** – impactul negativ potențial. Nivele de severitate:
 - 1. Pierdere de date, Resetare
 - 2. Pierdere de funcționalitate, Erori operaționale, Rezultate eronate
 - 3. Pierdere de utilizabilitate, probleme ortografice, erori cu manifestare rară
 - 4. Avertismente și sugestii
 - **Prioritatea** – urgența (dacă ține în loc proiectul)
 - 1. Corectare imediată, eroare evidentă, testarea nu poate continua
 - 2. Corectare necesară înainte de lansarea produsului
 - 3. Corectare în limita timpului disponibil
 - 4. Corectare neglijabilă, se recomandă alte măsuri de tratare a erorii
- Vulnerabilitățile de securitate sunt dificil de clasificat – poate fi folosit modelul DREAD
- Prioritatea unei erori se poate schimba în timp, pe măsură ce se apropie deadlineul

Raportarea testării

- Ciclul de viață al erorii = fazele prin care trece raportul unei erori (încă un motiv pentru a crea un raport pe eroare)

- Fazele ciclului:

- Detectat (se finalizează prin crearea raportului)
- Deschis (se finalizează prin corectarea erorii)
- Rezolvat (se finalizează prin confirmarea de către tester)
- Închis (se finalizează prin închiderea erorii).

- Ciclu extins:

- Între fazele Deschis și Rezolvat poate să apară faza Recenzare, în care intervine comitetul pentru stabilirea relevanței erorii
- În urma fazei de Recenzare se poate reveni la faza Deschis (programatorul corectează) sau se sare la faza Închis (s-au găsit alte metode de tratare și managerul decide închiderea forțată) sau se creează faza temporară Amânat (On hold).
- În faza Rezolvat
 - testerul poate fi de acord cu modul de tratare a erorii și trece la faza Închis
 - testerul poate să găsească noi informații în sprijinul corectării erorii (izolează eroarea mai bine) și revine la faza Detectat
 - testerul poate, prin testare regresivă, să arate că eroarea nu a fost corectată și revine la faza Detectat

Raportarea testării

- Sistem de urmărire a rapoartelor de eroare (Bug tracking) – necesar pentru urmărirea ciclului de viață al fiecărei erori
- Se recomandă ca sistemul bug tracking să conțină și jurnalul de testare
- Rubricile unui raport de testare, conform standardului IEEE:
 - Identificator unic al raportului (deci al erorii)
 - Descrierea sumară a erorii cu referință la componenta testată, procedeul CT și specs CT care au descoperit-o
 - Descrierea detaliată a erorii:
 - Data și ora
 - Testerul
 - Platforma
 - Datele de intrare
 - Procedeul
 - Rezultatele reale și cele așteptate
 - Metoda de reproducere a erorii și succesul său
 - Indicii destinate programatorilor
 - Impactul erorii:
 - Severitatea
 - Prioritatea
 - Impactul asupra planificării testării



[Raportarea testării]

- În practică, aceste rubrici corespund fazei Deschis și la acestea se adaugă câte o rubrică pentru celelalte faze ale ciclului de viață: Deschis, Recenzat, Rezolvat, Închis, Amânat. Fiecare fază va indica rubricile:
 - Identitatea și funcția persoanei implicate în faza respectivă (cu semnătură)
 - Data intrării în fază
 - Rezultatul sau decizia fazei, cu comentariu justificativ (metoda de corectare la Deschis, concluzii la Recenzat, concluziile testării regresive la Rezolvat, parafa la Închis sau Amânat)
- Sistemele bug tracker se bazează pe formulare off-line sau on-line
- Sistemele on-line sunt puternic automatizate: Bugzilla, Mantis

Mantis – fereastră de editare

Additional edit fields

Mantis - Editing bug 3238

Title: **Mantis: Convert common wild cards to TSQL in queries**

Severity: 3-Minor Error Product: Mantis

Priority: 3-Should fix if time Version: 0

Assigned To: DavBal Area: Data Retrieval

Related Bug: Percent Complete: 40

Projected Completion Date: 7/14/00 Projected Work Days: 2

Comments:

----- Open DAVBAL 06/22/2000 12:32 pm -----

Objective: Return wild-card queries as expected.

Reproduction steps:

1) Use a common wild card - * or ?

Expected results:

The * should return records which match (any characters) missing
The ? should return records which match (one character) missing

Actual results:

Currently TSQL supports the underscore _ in place of a ? and % in place of *. Mantis should perform a replacement of * and ? to enable wild cards to work as expected.

Environment (OS, RAM, video resolution):

- Assigned to DavBal on 06/22/2000 at 12:35 pm

----- Edited DAVBAL 07/14/2000 3:24 pm -----

Life cycle
tracking
information

Mantis – fereastră fazei Inchis

Mantis - Closing bug 3238

Title: **Mantis: Convert common wild cards to TSQL in queries**

Severity: 3-Minor Error Assigned To: QA Save

Priority: 3-Should fix if time Build: 0715 Cancel

Comments:

----- **Open DAVBAL** 06/22/2000 12:32 pm -----
Objective: Return wild-card queries as expected.

Reproduction steps:
1) Use a common wild card - * or ?

Expected results:
The * should return records which match (any characters) missing
The ? should return records which match (one character) missing

Actual results:
Currently TSQL supports the underscore _ in place of a ? and % in place of *. Mantis should perform a replacement of * and ? to enable wild cards to work as expected.

Environment (OS, RAM, video resolution):

- Assigned to DavBal on 06/22/2000 at 12:35 pm

----- **Resolved DAVBAL** 07/14/2000 3:28 pm -----
This will be fixed in tomorrow's build.

- Assigned to QA on 07/14/2000 at 03:28 pm

----- **Closed DAVBAL** 07/14/2000 3:28 pm -----
Confirmed as fixed within build 0715.]

Măsurarea rezultatelor testării

■ Măsurarea = interogări asupra bazei de date din bug-tracker

Mantis Query Builder

Load Query Delete Query Save Query Run Query Cancel

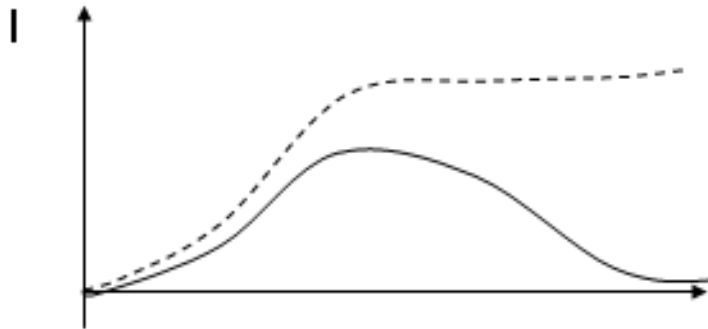
	Field Name	Operator	Argument	Boolean
<input checked="" type="checkbox"/>	(Product	Equals	Mantis) <input type="checkbox"/> Or
<input type="checkbox"/>	(Product	Equals	Mantis Web) <input checked="" type="checkbox"/> And
<input checked="" type="checkbox"/>	(Opened_By	Equals	IraCol) <input type="checkbox"/> Or
<input type="checkbox"/>	(Opened_By	Equals	JosNar) <input checked="" type="checkbox"/> And
<input type="checkbox"/>	(Status	Not Equals	Closed) <input type="checkbox"/> <END>

■ Rezultatele interogării se exportă în Excel pentru grafice și totalizări

Măsurarea rezultatelor testării

■ Metrice uzuale:

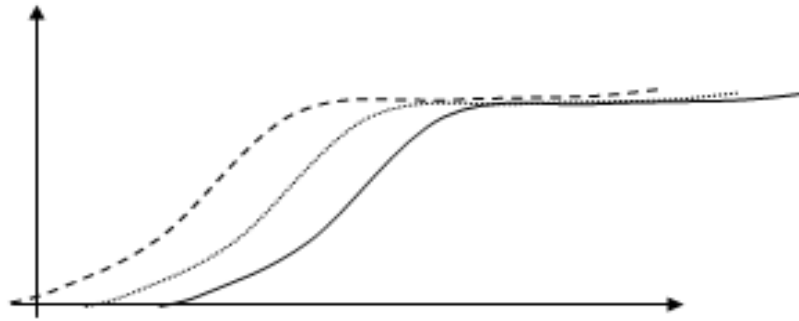
- Distribuția statistică a erorilor după diverse criterii: severitate, prioritate, timp, tester, mod de rezolvare, la nivel de componentă
- Metrice la nivel de proiect (sintetice):
 - Distribuția erorilor după tipologie sau aria în care s-au detectat (utilizabilitate, erori de calcul, erori de referire etc.) – un grafic Pie va indica zonele cu erori mai multe, deci zonele susceptibile (axioma grupării erorilor)
 - Cronogramele – evoluția în timp a tipurilor de erori. Evoluție naturală și cumulată



- Cronograma presupune un efort de testare constant (nu se trec zilele libere)!
- Rata de detecție începe să scadă după un punct critic (dacă rata de detecție crește, nu s-a atins punctul critic)
- Decizia de oprire a testării se poate face după rata de detecție

Măsurarea rezultatelor testării

■ Cronograma erorilor după fazele ciclului de viață: Deschis, Rezolvat, Închis



■ Pe axa X, cele trei faze sunt decalate de intervalul de timp între acestea

■ Ideal, cele 3 curbe, vor atinge același punct (dacă tot ce s-a Deschis s-a Rezolvat și s-a Închis)

■ În practică

- curba din mijloc (Rezolvat) variază între curbele Deschis și Închis (indicând eficacitatea la corectare a programatorilor)
- curba Închis poate să nu urce până la nivelul Deschis (dacă există erori Amânate)
- dacă se practică Închiderea forțată de către manager, e posibil ca Închis să atingă același nivel cu Deschis

■ CURS 11

- Clasificarea erorilor de programare
cf. *Find the Bug* – Adam Barr, Addison Wesley,
2004

Clasificarea erorilor de programare

Clasificarea manifestărilor erorilor de programare

- eșec, pană (failure) – dinamică, manifestare vizibilă utilizatorului
 - disfuncție, defect (fault) – dinamică, stare defectuoasă a programului în timpul execuției sale, care conduce la o pană
 - eroare (error) – statică, existența propriu-zisă a erorii observabilă în codul sursă
-
- O pană e produsă de una sau mai multe disfuncții
 - O disfuncție e produsă de una sau mai multe erori

Clasificarea erorilor de programare

Erori sintactice – încalcă regulile limbajului de programare, sunt ușor detectate prin metode automate (editoare de cod sursă avansate)

Erori semantice

erori de logică – au ca efect rezultate eronate ale programului

erori dinamice – au ca efect oprirea funcționării programului

Clasificarea erorilor semantice după D. Knuth:

- Anomalii algoritmice – algoritmul e precis, dar lipsit de acuratețe
- Anomalii de implementare – algoritmul e corect, dar codul sursă îl implementează prost
- Anomalii ale datelor – datele au fost supuse unor transformări incorecte
- Omisiuni – lipsa de completitudine a algoritmului
- Anomalii de limbaj – utilizarea incorectă a regulilor limbajului nedetectată de compilator (ex: precedența operatorilor)
- Nepotriviri – erori de tip sau parametrizare nedetectate de compilator (ex: datorită conversiilor de tip implicite)
- Anomalii de robustețe – lipsa de robustețe la garbage data, excepții netratate, mesaje de eroare insuficiente
- Erori surpriză – interacțiuni neprevăzute între secțiunile programului, pot fi încadrate în celelalte categorii dar distanța între eroare, disfuncție și pană e suficient de mare încât să fie dificil de conectat între ele.
- Erori de tastare care nu produc erori de sintaxă (confuzie între numele unor variabile de tip similar etc.)

Clasificarea erorilor de programare

Clasificarea erorilor după Barr (restrânsă după clasificarea Knuth)

- tip A – erori algoritmice, pot include anomalii algoritmice și erorile surpriză care de fapt pot fi același lucru, dar și erorile de robustețe.
- tip D – erori de date, când codul sursă citește sau scrie date incorecte, inclusiv erorile provocate de rotunjiri
- tip F – omisiuni și locații eronate ale unor părți din structurile de programare
- tip B – anomalii de implementare, erori de nepotrivire, anomalii de limbaj și erori de tastare (cod sursă eronat față de algoritm)

[Erorile de tip A]

A.off-by-one

- Erori de decalare cu unu (1 bit, 1 iterație, 1 element de matrice, 1 înregistrare etc.), adică omiterea unei valori dintr-o mulțime de valori procesate
- Ex: eroarea fencepost: **`nrdepagini = nr.ultima pagina – nr.prima pagina`**
- Ex: **folosirea comparației `>` în loc de `>=`**
- Ex: **contor For începând de la 1 în limbaje care indexează vectorii începând de la 0**

A.logic

- Logică incorectă în obținerea unor rezultate, de obicei din cauza unei presupunerii false
- Ex: ciclurile FOR și WHILE infinite (limbaje ca Java, C permit cicluri FOR infinite, deoarece valoarea finală a contorului e indicată printr-o condiție)



[Erorile de tip A

A.validation

- Date de intrare incorecte față de domeniul așteptat al datelor
- Poate fi vorba de date introduse de utilizator (validarea formularelor, validarea Excel, validare la citirea datelor)
- Poate fi vorba de date transferate ca parametrii spre o subrutină (funcție)
- Aceste erori intră în clasificarea Knuth la robustețe:
 - orice funcție sau program trebuie să verifice dacă datele primite (de la utilizator sau parametru) se încadrează în domeniul de valori acceptat.
 - dacă valoarea e invalidă, avem de a face cu o excepție pentru care funcția sau programul trebuie să creeze un mesaj de eroare
- Cel mai frecvent mod de validare:
 - validarea împotriva valorilor nule: IF EMPTY(a)

[Erorile de tip A]

A.performance

- După Knuth, acestea nu sunt erori, ci intră în domeniul optimizării algoritmilor
- Utilizarea inutilă a unor resurse în timpul execuției
 - Resursa poate fi timpul, memoria, încărcarea rețelei etc.
 - Economisirea resurselor e contradictorie:
 - Un algoritm mai rapid e posibil să fie mai mare (ca memorie ocupată)
 - Se pune problema perspectivei (criteriului) după care se dorește optimizarea

[Erorile de tip D]

D.Index

- indice invalid la parcurgerea unei structuri (matrice, șir de caractere, contor FOR)
- apare uneori cuplată cu erorile A.off-by-one, la limbaje care numerotează indicii începând de la zero: un indice care trebuie să parcurgă n elemente, va lua valori de la 0 la $n-1$
- tot indice invalid avem și dacă se depășește domeniul indicelui
- Erorile A.off-by-one se referă la neglijarea unei valori în timp ce D.index se referă la valoarea invalidă a indicelui – chiar dacă apar adesea simultan, sunt două tipuri de erori diferite, putând apare și separat

D.Limit

- Eroare de limită – se procesează eronat primul și ultimul element dintr-o structură
- Adesea e cauzată de
 - o eroare D.index (s-au pierdut valori de la marginile domeniului indexului, s-au prelucrat valori invalide datorită depășirii domeniului indexului)
 - o eroare de robustețe (nu s-au tratat limitele ca și cazuri de excepție)

[Erorile de tip D]

D.Number

- Erori cauzate de modul de reprezentare a numerelor
 - trunchierile neprevăzute: rotunjirea automată la împărțirea întreagă, conversiile de tip implicite
 - depășirile de buffer (stocarea unui număr într-un spațiu de memorie insuficient), în unele limbaje provoacă erori, în altele trunchieri forțate, în altele suprascrierea memoriei adiacente
 - Ex: $a=b$, în condițiile în care a e de tip short (16 b), b e de tip long (32 b)
 - Ex: $a=b*b$, dacă rezultatul depășește valoarea maximă admisă pentru tipul lui a
 - programarea la nivel de bit (C) trebuie să țină cont de tipul procesorului:
 - big-endian – procesoare ce memorează cel mai semnificativ octet la început
 - little-endian- procesoare ce memorează cel mai semnificativ octet la sfârșit
 - Ex: numărul hexa 1234 va fi memorat
 - De un procesor BE în doi octeți cu valorile 12 și 34
 - de un procesor LE în doi octeți cu valorile 34 și 12, în această ordine (cifrele 12 sunt cele mai semnificative)

[Erorile de tip D]

D.memory

- Erori cauzate de utilizarea defectuoasă a memoriei, în cazul limbajelor ce permite programarea la nivel de pointeri
 - Accesarea unei zone de memorie inaccesibile (acces la o variabilă care nu mai are valoare)
 - Folosirea a două variabile diferite cu același nume (suprascrierea memoriei și pierderea de date) – apare frecvent la copierea unor părți din codul sursă
 - Memorie neeliberată
 - Memorie eliberată prea repede

Erorile de tip F

F.init

- Utilizarea de variabile neinițializate.
 - Efecte: unele limbaje alocă o valoare implicită, altele alocă valoarea reziduu care există la momentul respectiv la adresa noii variabile, altele provoacă o eroare de tip D.memory.
- Apare frecvent în structuri IF în care o variabilă e inițializată pe o ramură și neglijată pe cealaltă ramură
- Apare frecvent în structuri WHILE, care necesită inițializarea explicită a contorului înainte de începerea ciclului

F.missing

- omiterea unei instrucțiuni, e de obicei cauza ciclurilor infinite care uită să își modifice variabila pe care s-a creat condiția

F.location

- Instrucțiune plasată eronat:
 - inițializarea unui contor în interiorul ciclului
 - folosirea unei variabile înainte atribuirii unei valori variabilei
 - Imbricări incorecte
 - Instrucțiuni redundante

Erorile de tip B

B.variable

- utilizarea eronată a unui nume de variabilă (erori de tastare, ordine incorectă a parametrilor unei funcții)
- cauzată adesea de copierea codului sursă

B.expression

- utilizarea eronată a unor expresii (operatori eronați, precedența eronată a operatorilor, confundarea operatorilor logici and și or)
- poate cauza erori de tip A.logic

B.language

- utilizarea eronată a regulilor sintactice ale limbajului, fără a provoca erori de sintaxă
- ex: în C:

```
if (i == 5);{  
    i = 0;  
}
```



- CURS 12
 - Limbajul Autolt

[Tipizare]

Un singur tip de date – variant = o variabila isi poate schimba tipul, intre 4 tipuri de valori:

- Numeric (zecimal, exponential, hexazecimal)
- String
- Boolean
- Coduri hexazecimale:
 - Coduri obtinute prin conversia sirurilor de caractere (ASCII, Unicode)
 - Coduri de culoare (RGB)
 - Identificatori GUI (Handles)

Conversii implicite = declansate de operatori (elimina eroarea type mismatch):

- “2”+”2”=4
- 2 & 2 = “22”
- “ab”+2=2 (conversia esuata are rezultat 0)
- ((“Tr”&”ue”) OR False) + 1 = (“True” OR False) + 1 = True + 1 = 2
- “False” AND “False” = True

Conversii explicite – cu functii predefinite!

[Variabile]

Valori posibile a fi atribuite:

- Valori elementare
- Valori masive eterogene, indexate de la 0
 - `$A[2][2]=[[1,2],["a","b"]]`
- Referinte COM
 - `$V=ObjCreate("excel.application")`
- Identificatori GUI
 - `$ID=GUICreate("Fereastra Mea")`
 - `$ID2=WinGetHandle("Untitled – Notepad")`
- Structuri de date C
 - `$Struc=DllStructCreate("int v1; char v2[10]")`

Structuri de programare

Conditionale:

1. IF THEN ELSE
2. SWITCH **variabila**
 CASE **valoare1**
 bloc 1
 CASE **valoare2**
 bloc 2
 CASE ELSE
 bloc 3
ENDSWITCH
3. SELECT
 CASE **conditie1**
 bloc 1
 CASE **conditie2**
 bloc 2
 CASE ELSE
 bloc 3
ENDSELECT

Repetitive:

1. Cu contor:
 FOR ...TO ... STEP ...
 bloc
 NEXT
2. Parcurgere de obiecte
 FOR var IN colectie
 bloc
 NEXT
3. Parcurgere de proprietati calificate
 WITH obiect
 prop1=...
 prop2=...
 prop3=...
 ENDWITH

Structuri de programare

Repetitive:

4. Cu preconditie de continuare

WHILE preconditie

 bloc

WEND

5. Cu postconditie de oprire

DO

 bloc

UNTIL postconditie

Instructiuni de fortare:

1. **CONTINUECASE**

 Forteaza executia urmatoarei ramuri
 CASE, chiar daca e invalida

2. **CONTINUELOOP**

 Forteaza terminarea iteratiei curente

3. **EXITLOOP**

 Forteaza terminarea structurii repetitive
 curente

4. **EXIT**

 Forteaza terminarea macroului

Functia OnAutoItExit()

- Handler pentru iesirea din program
 (gestioneaza codurile de terminare)



[Functiile utilizatorului

```
Func functiaMea ($p1, $p2=val)    nume (para obligatoriu, para optional)
    ; blocul de instructiuni al functiei
    Return $v                    oprire si returnare
Endfunc
```

```
functiaMea(5,"a")                ; apel (argumente)
$a=functiaMea(2)                  ;apel fara argument optional (i se atribuie val)
```

Un macro care contine doar definitii de functii, fara apeluri si program principal =
biblioteca de functii

Parametrii – read only (declarati cu CONST, nu pot fi modificati de functie)
- read write (implicit, pot fi modificati de functie)



[Functiile utilizatorului]

Argumente transferate prin valoare

- argumentul si parametrul sunt variabile diferite chiar daca au acelasi nume
- parametrul nu influenteaza argumentul)

```
Func F($p)
```

```
$p=2*$p
```

```
;p devine 10
```

```
Endfunc
```

```
$a=5
```

```
F($a)
```

```
; a ramane la valoarea 5
```

[Functiile utilizatorului]

Argumente transferate prin referinta

- argumentul si parametrul sunt aceeași variabilă chiar dacă au nume diferite
- parametrul influențează argumentul

```
Func F(ByRef $p)
```

```
$p=2*$p
```

```
;p devine 10
```

```
Endfunc
```

```
$a=5
```

```
F($a)
```

```
; a devine 10
```

[Functiile utilizatorului]

Vizibilitatea variabilelor (testata poate fi testata cu IsDeclared)

```
Func F()  
    Global $c=10  
    Local $b  
    $b=3*$a  
    ; b devine 15  
    ; (variabilele din programul principal sunt vizibile in toate functiile!)  
Endfunc
```

```
$a=5  
F()  
$a=$c  
;a devine 10,  
;variabilele globale din functii sunt vizibile in tot programul  
$a=$b
```

;eroare, variabilele locale din functii sunt vizibile doar in functia care le-a creat!

[Functiile utilizatorului]

Vizibilitatea e controlata de pozitia in codul sursa si de modul de declarare:

- **Local** – variabile vizibile local (doar in functia care le-a creat), ignora variabilele globale cu acelasi nume!
- **Global** – variabile vizibile global (in toate functiile si programul principal)
- **Dim** (implicit, poate lipsi) – variabile vizibile local, create cu conditia sa nu existe deja o variabila globala cu acelasi nume
- Variabilele create in programul principal sunt globale!
- Parametrii functiilor sunt locali!
- Variabilele nu sunt vizibile inainte de crearea lor (implicatii asupra ordinii apelurilor de functii multiple)!



[Directive

#cs

bloc de comentarii

#ce

Rolul comentariilor in depanare = portiuni de cod sursa pot fi activate sau dezactivate daca se comenteaza sau decommenteaza linia #cs!

#include fisier.au3

Folosit in includerea bibliotecilor de functii și a bibliotecilor de variabile predefinite (directorul Include)



[Tipuri de functii

Functii

- Ale utilizatorului
- Predefinite ale utilizatorului (publicate si documentate in pachetul Autolt) – necesita #include pentru a fi folosite
- Predefinite – recunoscute de Autolt
 - OPT (configurarea interpretorului Autolt)
 - Manipularea ferestrelor (pe baza de identificatori GUI)
 - Programare GUI (varianta vizuala: Tools - Koda Form Designer)
 - Manipularea timpului (asteptare, sincronizare)
 - Manipularea tipurilor
 - Teste de tip
 - Conversie explicita
 - Manipulare de date
 - Manipularea registrilor si obiectelor COM

■ Succesul functiilor predefinite e testat cu **macrovariabila @error** – valoarea 1 indica esecul ultimului apel de functie!



[Programare GUI

GUI = o multime de ferestre

Fereastra = un grup de obiecte GUI

Identificator GUI = cod hexazecimal atribuit de Windows fiecarei ferestre si obiect create

Etapele construirii GUI:

- #include GUIconstants.au3
- GUICreate – crearea ferestrelor implicit invizibile (genereaza Window ID)
- GUICtrlCreate... – crearea obiectelor (genereaza Control ID)
- GUISwitch – comutare intre ferestre
- GUISetState – schimbarea starii unei ferestre (ex: afisarea lor initiala)
- Programare evenimente prin cicluri infinite cu iesire fortata
 - metoda MessageLoop (implicita)
 - metoda OnEvent (se activeaza cu functia OPT)

Programarea evenimentelor prin MessageLoop

Fiecare eveniment genereaza un mesaj:

- 0 – lipsa de eveniment
- Control ID – eveniment al unui obiect
- Variate constante – evenimente ale ferestrei (inchidere, maximizare, etc.)

Continutul ciclului infinit:

- Ascultarea functiei GUIGetMsg
- Compararea valorii returnate cu valorile asteptate intr-o structura CASE
- Una din ramurile CASE va contine iesirea fortata (asociata unui buton de inchidere)



Programarea evenimentelor prin MessageLoop

- #include <GUIConstants.au3>

- GUICreate("Fereastra mea", 200,200)

- \$IDButon=GUICtrlCreateButton("Butonul meu",50,50)

- GUISetState(@SW_SHOW)

- **While 1** ; **ciclu de ascultare infinit**
- \$mesaj=**GUIGetMsg()** ; ascultare mesaje
- Switch \$mesaj ; testarea mesajului
- Case \$IDButon ; **mesajul butonului**
- MsgBox(0,"","S-a apasat butonul!")
- Case \$**GUI_EVENT_CLOSE** ; **mesajul ferestrei**
- MsgBox(0,"","Se inchide fereastra!")
- **ExitLoop** ; **oprirea ciclului de ascultare**
- EndSwitch
- **Wend**



Programarea evenimentelor prin OnEvent

Fiecare eveniment:

- Apeleaza implicit o functie handler
- Genereaza macrovariabilele
 - @GUI_CTRLID – Control ID al obiectului
 - @GUI_WINHANDLE – ID al ferestrei

Etape:

- Se activeaza metoda cu functia OPT(“GUIOnEventMode”,1)
- Se programeaza functiile handler urmarite (1 pe eveniment sau 1 pe mai multe evenimente separate intr-un CASE al functiei)
- Se construiesc GUI si obiectele
- Se leaga obiectele la functiile handler (GUICtrlSetOnEvent)
- Se leaga ferestrele la functiile handler (GUISetOnEvent)
- Se construiesc un ciclu infinit de asteptare (Sleep) – una din functiile handler va trebui sa contina iesirea fortata (Exit,ExitLoop)

Programarea evenimentelor prin OnEvent

```
#include <GUIConstants.au3>
Func GestiuneButon()           ;handler pt buton
    MsgBox(0,"","S-a apasat butonul!")
Endfunc
Func GestiuneInchidere()      ;handler pt inchiderea ferestrei
    MsgBox(0,"","Se inchide fereastra!")
    Exit                      ; terminarea executiei
Endfunc

Opt("GUIOnEventMode",1)      ; activare Event mode
$IDXfer=GUICreate("Fereastra mea",200,200)
$IDbuton=GUICtrlCreateButton("Apasa-ma",20,20)
GUISetOnEvent($GUI_EVENT_CLOSE,"GestiuneInchidere") ; legare
GUICtrlSetOnevent($IDbuton,"GestiuneButon")         ; legare
GUISetState(@SW_SHOW)

While 1                      ; ciclu infinit
    Sleep(1000)
Wend
```



[Manipulare GUI

SEND si variante - simularea tastarii

MOUSECLICK si variante – simularea utilizarii mouseului

Functiile Window Management (vezi Help) – manipularea ferestrelor si
obiectelor GUI (ex: CONTROLCLICK)

De multe ori MOUSECLICK si functiile de acces la butoane sunt inlocuite
cu SEND – se simuleaza shortcuturile de acces la butoane si meniuri
prin tastatura! (se elimina unele probleme de sincronizare cu pozitia
butonului sau cu textul de pe el, care se pot modifica dinamic)

■ CURS 13

- Managementul calității software
- Standarde de calitate software
- Tipuri de testeri
- Metode de acumulare a experienței pentru angajare

[QA <> testare]

QA – activitate (departament) care încearcă să impună un nivel de calitate:

- Instituire de metodologii și standarde;
- Definirea de soluții la probleme;
- Integrarea testării în procesul de producție;
- Definirea de garanții adresate beneficiarilor și asigurarea lor.

Testarea – activitate (și dpt.) care oferă un diagnostic privind abaterea de la nivelul de calitate impus.

QA – preventivă față de erorile detectate la testare;

Testarea – preventivă față de erorile manifestate la beneficiar.

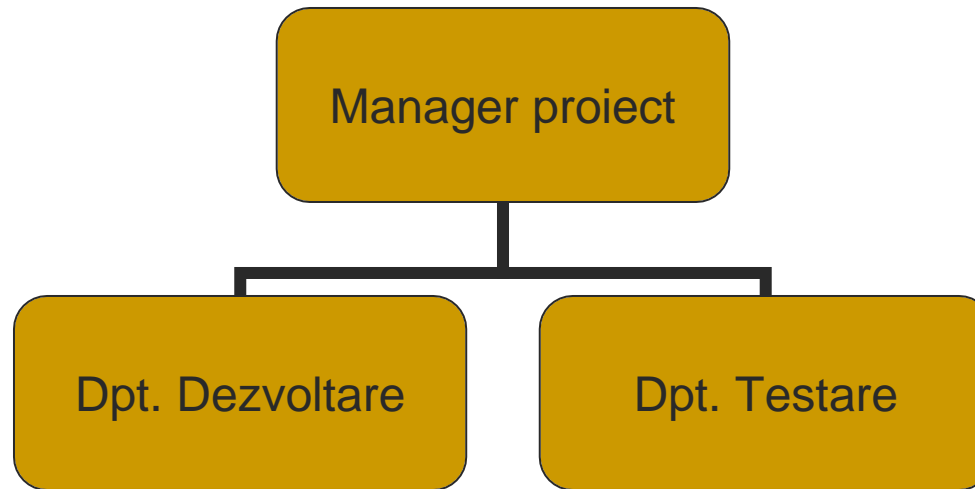
Dependențe între QA și testare:

- QA asigură și calitatea (efectivitatea) testării prin măsuri de organizare a procesului de testare;
- QA scade efortul de testare prin prevenirea de erori (anulează unele costuri de nonconformitate internă);
- QA folosește concluziile testării pentru a-și adapta strategia QA;

Concepte înrudite:

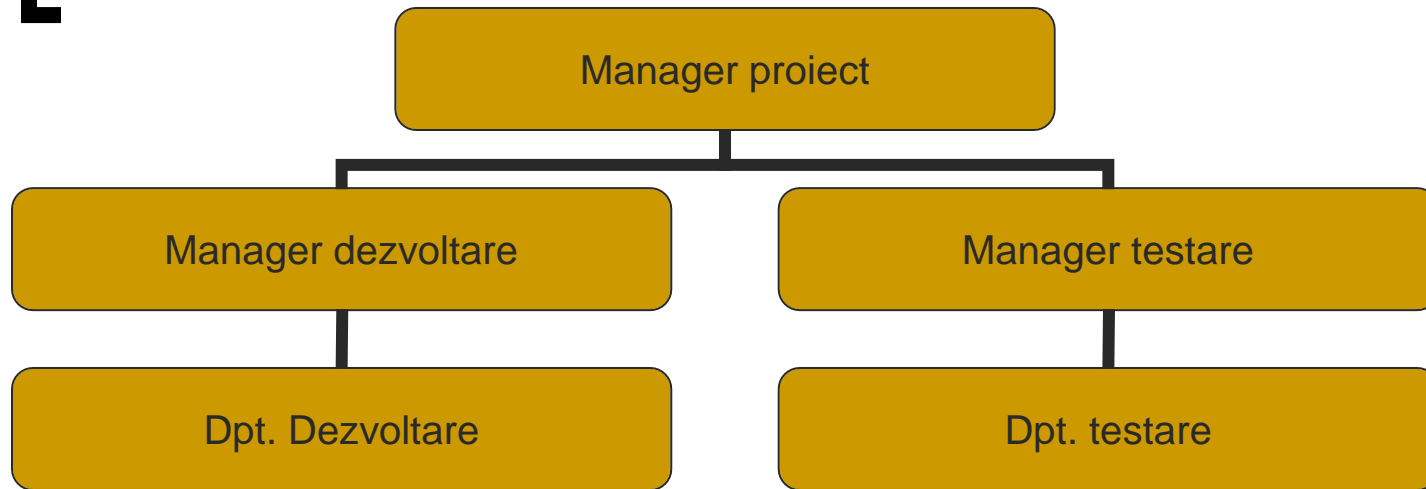
- **Controlul calității** – terminologie industrială, bazată pe **inspectori de calitate** (testeri externi cu autoritatea de a solicita certificări de calitate, a verifica QA și chiar a bloca procesul de producție);
- **Verificarea și validarea software** – termen atribuit departamentelor de testare divizate în:
 - **verificare** (testare față de specificații);
 - **validare** (testare față de cerințele beneficiarului);
- **Integrare și testare, management și testare, Dezvoltare și testare** – termeni vagi ce indică o organizare slabă, cu divizare neclară a răspunderilor între producție și testare;
- **Total Quality Management** – abordarea prin care QA nu este un departament cu răspunderi centralizate, ci fiecare membru al echipei are propriile răspunderi QA (asigurarea calității este distribuită).

Modele de integrare a managementului QA



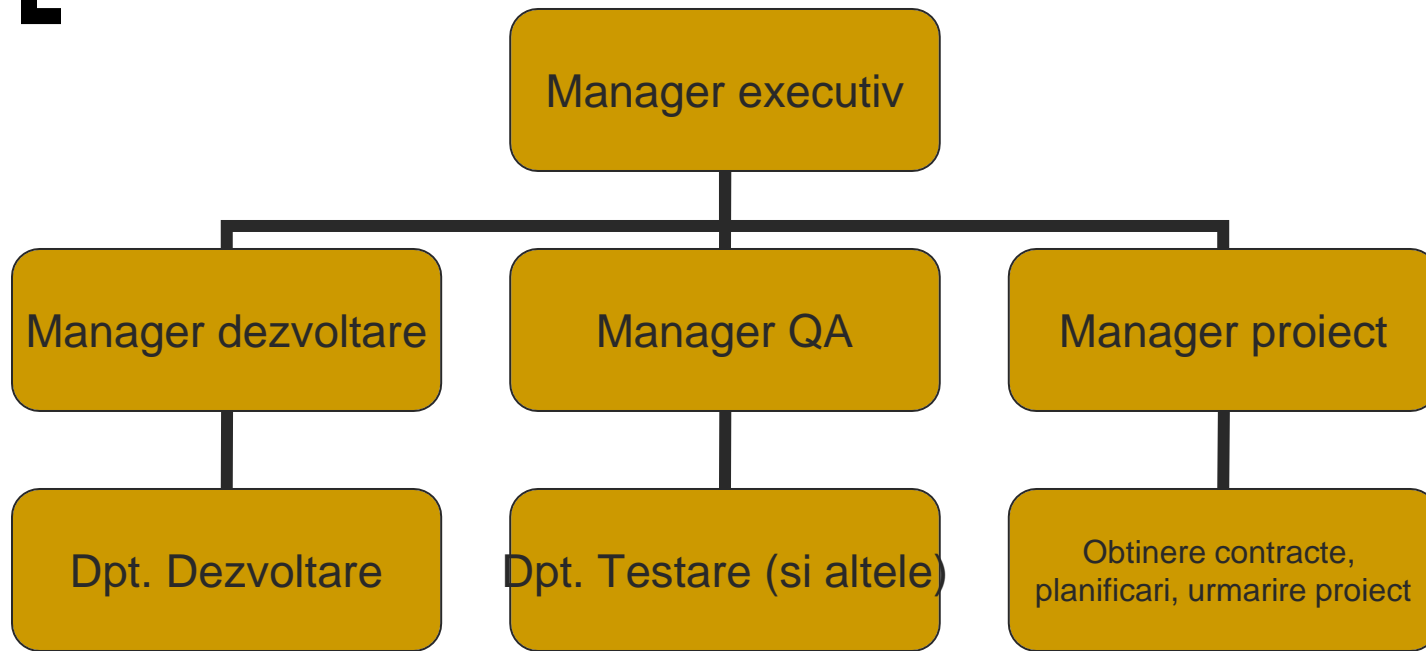
- Frecvent în companii mici, care nu lucrează la mai multe proiecte simultan
- Managerul de proiect are tendința să favorizeze unul din departamente, sau să considere testarea un departament auxiliar
- Managerul de proiect poartă atât răspunderea producției cât și a calității producției (aspecte conflictuale dpdv al rezultatelor și resurselor alocate)

Modele de integrare a managementului QA



- Managerul de proiect e neutru;
- Managerii de dezvoltare și testare promovează rezultatele celor două departamente și solicită resurse;
- Departamentele sunt independente și sunt reprezentate echitabil la nivelul managerului de proiect.
- Managerul de testare este subordonat proiectului, deci mai multe proiecte derulate simultan necesită mai mulți manageri de testare

Modele de integrare a managementului QA



- Acest model se pretează la companiile care derulează mai multe proiecte simultan
- Managerul de proiect e la același nivel de autoritate decizională cu managerii calității și producției
- Funcționarea QA este transparentă față de proiect și afectează toate proiectele
- Calitatea e asigurată la nivelul companiei (standarde, metodologii, testare) și nu la nivelul proiectului;

Standarde de calitate

- **Capability Maturity Model for Integration** (creat de DoD) – măsoară maturitatea organizațională:
 - **Nivel de bază:**
 - Organizare pe bază de improvizații;
 - Resurse alocate haotic și intuitiv;
 - Succesul depinde de sacrificiul individual al unor persoane cheie;
 - Planificarea e nonrealistă și costurile imprevizibile;
 - Proiectele sunt desfășurate după modelul Big Bang;
 - Testarea e ad hoc, auxiliară și irelevantă.
 - **Nivelul modelelor repetabile** (majoritatea instituțiilor din România se află la acest nivel):
 - Costurile, procesele și calitatea pot fi urmărite la nivel de proiect (dar nu și la nivel de organizație);
 - Apar situații și decizii reutilizabile în proiecte similare;
 - Testarea e parțial documentată.
 - **Nivelul modelelor definite:**
 - Costurile, procesele și calitatea pot fi urmărite la nivelul organizației;
 - Apar situații și decizii reutilizabile la nivel de organizație;
 - Nu se acceptă abateri de la planificări și standarde;
 - Testarea e documentată și recenzată;
 - Managementul QA e la același nivel de autoritate cu managementul de proiect.
 - **Nivelul modelelor controlate:**
 - Calitatea e precis cuantificată;
 - Procesele sunt monitorizate prin modele statistice precise.
 - Celelalte trăsături ale nivelului anterior;
 - **Nivelul optimizărilor:**
 - Procesele și metodele sunt supuse în continuu îmbunătățirii;
 - Calitatea evoluează incremental, prin ridicarea continuă a pragurilor de calitate și realocare flexibilă de resurse în vederea urmăririi noilor praguri

■ Obs: apar probleme dacă nivelul testerilor e diferit de nivelul organizației!

Standarde de calitate

■ ISO 9000

- oferă o certificare (cu logo) care atestă **nivelul de calitate al procesului de producție** (deci nu al produsului);
- ține cont de subiectivismul calității produsului, insistând asupra aspectelor obiective ale procesului de producție;
- este un punct forte în competițiile de proiecte din toată lumea.

■ ISO 9000 conține prevederi legate de testare:

- Să se creeze planuri de calitate și proceduri de desfășurare a testării, de evaluare a nonconformității și de derulare a acțiunilor corective;
- Să se supună aprobării un plan de dezvoltare software care să conțină definirea proiectului, o listă a obiectivelor, o planificare în timp, specificații, o descriere a structurii organizaționale, o justificare a riscurilor și strategii de control a riscurilor;
- Să se comunice specificațiile în termeni accesibili beneficiarilor potențiali;
- Să se definească proceduri de recenzare la nivelul proiectării software;
- Să se definească măsuri de control a modificărilor de la nivelul proiectării;
- Să se realizeze documentarea completă a testării și stocarea pe termen lung a rezultatelor;
- Să se definească și aplice metode de testare relative la cerințele beneficiarilor (validare);
- Să se controleze modul de investigare și rezolvare a erorilor (ciclul de viață al erorii);
- Să se asigure dovezi privind calitatea produsului;
- Să se definească proceduri care să controleze lansarea produsului (integrarea cu produsele auxiliare);
- Să se definească metrici de analiză a calității și de analiză a procesului de producție software.

[Referințe Web]

- www.sei.cmu.edu/cmmi - modelul maturității organizaționale CMMI
- International Organization for Standardization (ISO), www.iso.ch
- American Society for Quality (ASQ), www.asq.org
- American National Standards Institute (ANSI), www.ansi.org

[Despre tester]

- Testerul nu este o poziție provizorie, alocată programatorilor fără experiență (prejuducată întreținută de organizațiile imature pe scara CMMI, care adoptă metoda Big Bang și consideră testarea o activitate secundară);
- **Roluri uzuale ale testerilor:**
 - **Tehnician de testare** – începător, responsabil de obicei cu teste beta și teste de configurare;
 - **Tester profesionist (Test engineer)** – capabil să deruleze teste profesionale black box, să își definească propriile cazuri de testare, documentația de testare, participă la recenzii interne și colaborează cu programatorii;
 - **Tester-dezvoltator** – are aptitudini de programare, își automatizează testele sau evaluează instrumente de testare automată, creează drivere și stuburi, e specialist white box;
 - **Responsabil testare** – conduce echipa de testeri alocată unei porțiuni de produs, planifică, monitorizează și colectează metrice;
 - **Manager testare** – conduce departamentul de testare, poate fi și manager QA, alocă resurse de testare și definește strategia de testare în acord cu politica QA.

[Despre tester]

- **Metode de acumulare a experienței pentru începători:**
 - Exersarea testării pe produse domestice, considerând Help-ul ca specificație și înregistrând erorile cu Excel (erorile pot fi comunicate pe site-ul producătorilor);
 - Participarea la sesiuni beta testing, promovate în mod public, chiar și on-line;
 - Participarea ca voluntar în teste de utilizabilitate, promovate public de către laboratoarele de utilizabilitate;
 - Participarea în programe publice de bug-bashing, promovate on-line (frecvent destinate hackerilor);
 - Implicarea în comunitatea www.opensourcetesting.org care publică articole și promovează instrumente de testare automată.

Cerințele beneficiarului:

- **Profil client: casă de schimb valutar**
- **Nevoi:**
 - **Înregistrarea operațiilor de cumpărare și vânzare valută**
 - **Obținerea de rapoarte cu evoluția cursului leu/valută**
 - **Obținerea de rapoarte cu istoricul operațiilor valutare (diverse totaluri – pe zile, pe persoane, pe valute etc.)**
 - **Rulare Windows 7**
 - **Interfață grafică (formulare, meniuri)**

User Requirement(s) Document (URD)

- Cerintele utilizatorului este un document utilizat în mod obișnuit în ingineria software care specifică ce așteptări are utilizatorul de la produsul software (ce poate face).

Exemplu specs

- *Meniul Edit va avea două opțiuni, de sus în jos în ordinea: Copy și Paste*
- *Metodele de activare a opțiunilor vor fi:*
 - *Clic*
 - *combinația Alt-E urmată de C, respectiv P*
 - *combinațiile de taste implicite din Windows (Ctrl-C, Ctrl-V).*
- *Opțiunea Copy va avea ca efect copierea în Clipboard a afișajului Windows Calculator.*
- *Opțiunea Paste va avea ca efect copierea conținutului Clipboard în câmpul de afișare din Windows Calculator.*
- **Recomandare:** la specificație se pot adăuga și imagini dacă e cazul (în exemplul de față, notiunile de *meniu*, *opțiune* sunt suficient de clare)
- **Specificația e sistemul de referință al testerului ! (vezi definiția erorii)**

Ce este o cerință?

- Poate varia de la o descriere abstractă de nivel înalt a unui serviciu sau a unei constrângeri a sistemului până la o specificație funcțională precizată în detaliu în termeni matematici.
- Acest lucru este inevitabil deoarece cerințele pot servi unei funcții duale
 - Pot fi baza unei licitații pentru un contract – trebuie să fie deschise către interpretare;
 - Pot fi baza contractului însuși – trebuie definite în detaliu;
 - Ambele declarații (abstractă și detaliată) pot fi numite cerințe.

Software Requirement Specification (SRS)

- Specificațiile cerințelor software reprezintă o descriere detaliată a unui produs software care urmează să fie dezvoltat cu cerințele sale funcționale și nefuncționale.
- Este dezvoltat pe baza acordului dintre client și contractori.
- Documentul cu specificațiile cerințelor software cuprinde toate cerințelor necesare pentru dezvoltarea proiectului.
- Pentru a dezvolta sistemul software, ar trebui să avem o înțelegere clară a sistemului software. Pentru a realiza acest lucru avem nevoie de o comunicare continuă cu clienții pentru a aduna toate cerințele.

- **Standardul IEEE 830** (IEEE recommended practice for software requirements specifications) descrie continutul, calitatile si avantajele unei bune specificatii a cerintelor software

Calitatile Specificatiile cerintelor software trebuie sa fie:

- Corecte
- Neambigue – fiecare cerinta definita are o singura interpretare
- Complete – ar trebui sa contina tot ceea ce este necesar pentru realizarea software-ului
- Consistente – intre ele si cu documentele pe care le refera
- Clasificate dupa importanta si/sau stabilitate
- Verificabile. Trebuie evitate cerinte ca :”va furniza un raspuns rapid”, “sistemul nu va cadea niciodata”, etc.
- Modificabile. Atunci cand o aceeaasi cerinta apare in mai multe parti, actualizarile documentului sunt mai greu de facut
- Usor de corelat cu cerinte formulate in alte documente, de ex. URD

Avantajele

- Sta la baza contractului dintre clienti si furnizori.
- Reduce efortul de dezvoltare.
- Sta la baza estimarii costurilor si a planificarii
- Permite planificarea testelor de verificare si validare
- Usureaza transferul produsului la noi utilizatori sau pe platforme noi.
- Serveste ca baza pentru viitoarele imbunatatiri sau modificari ale produsului.

Tipurile de cerințe

- Cerințe utilizator

- Expuneri în limbaj natural plus diagrame ale serviciilor pe care le furnizează sistemul și constrângerile sale operaționale. Scrise pentru clienți.

- Cerințe sistem

- Un document structurat care să conțină descrieri ale funcționalității sistemului, serviciile și operațiile pe care le permite. Se definește ce va fi implementat, deci poate fi văzut ca o parte a contractului dintre client și contractor (producator).

Cerințe funcționale și non-funcționale

- Cerințe funcționale
 - Descriu funcțiile pe care trebuie să le realizeze sistemul, într-un mod independent de implementare.
 - Ce transformări trebuie efectuate asupra intrărilor și ce ieșiri trebuie să se obțină pentru fiecare tip de intrare.
 - Precizarea serviciilor pe care trebuie să le ofere sistemul, cum trebuie să reacționeze sistemul la intrări particulare și cum trebuie să se comporte sistemul în situații particulare.
- Cerințe non-funcționale
 - Constrângeri asupra serviciilor sau funcțiilor oferite de sistem, cum ar fi constrângeri de timp, constrângeri asupra procesului de dezvoltare, standarde, etc.
 - Cerințe de: performanță, interfață, de operare, de verificare, de portabilitate, de întreținere, de fiabilitate
- Cerințe de domeniu
 - Cerințe care vin din partea domeniului de aplicație a sistemului și care reflectă caracteristicile acelui domeniu.

Cerințe funcționale

- Descriu funcționalitatea sau serviciile sistem.
- Depind de tipul de software, de utilizatorii preconizați și de tipul sistemului în care este utilizat software-ul.
- *Cerințele utilizator* funcționale pot fi expuneri de nivel înalt despre ceea ce trebuie să facă sistemul.
- *Cerințele sistem* funcționale trebuie să descrie serviciile sistemului în detaliu.

Imprecizia cerințelor

- Dacă cerințele nu sunt exprimate precis pot să apară probleme.
- Cerințe ambigue pot fi interpretate în moduri diferite de către dezvoltatori și utilizatori.
- Considerăm termenul ‘instrumente de vizualizare corespunzătoare’
 - Intenția utilizatorului – instrumente de vizualizare specifice pentru fiecare tip de document;
 - Interpretarea dezvoltatorului – Oferirea unui instrument simplu de vizualizare text care să arate conținutul documentului.

Completitudinea și consistența cerințelor

- În principiu, cerințele trebuie să fie atât complete cât și consistente.
- Complete
 - Trebuie să includă descrieri ale tuturor facilităților necesare.
- Consistente
 - Nu trebuie să existe conflicte sau contradicții în descrierile facilităților sistemului.
- În practică, este imposibil să se producă un document al cerințelor și complet și consistent.

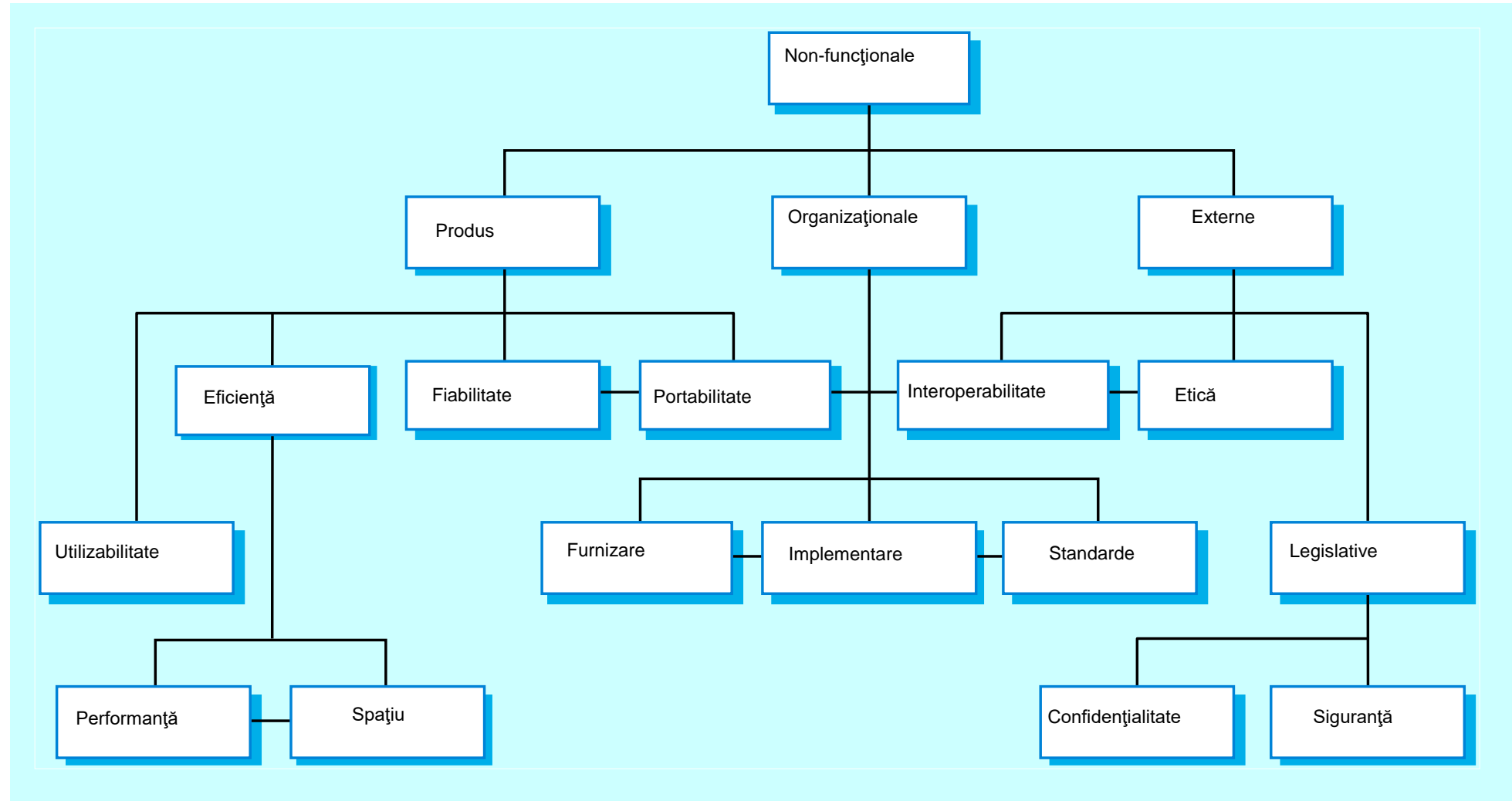
Cerințe non-funcționale

- Definesc proprietățile (ex. fiabilitate, timp de răspuns, necesar de memorie) și constrângerile (ex. capacitățile dispozitivelor de I/E, reprezentările sistemului, etc.)
- Cerințele procesului pot fi, de asemenea, specificate impunând un limbaj de programare sau o metodă de dezvoltare.
- Pot fi mai critice decât cerințele funcționale: dacă nu sunt îndeplinite atunci sistemul este nefolositor.

Clasificarea cerințelor non-funcționale

- Cerințe produs
 - Cerințe care specifică modul particular în care trebuie să se comporte produsul furnizat (ex. viteză de execuție, fiabilitate, etc.)
- Cerințe de organizație
 - Cerințe ce sunt consecință a politicilor și procedurilor organizaționale (ex. standarde de proces utilizate, cerințe de implementare, etc.)
- Cerințe externe
 - Cerințe care provin de la factorii externi sistemului și procesului dezvoltării sale (ex. cerințe de interoperabilitate, cerințe legislative etc.)

Tipurile cerințelor non-funcționale



Cerinte de performanta

- Valori numerice atasate unor parametri masurabili cum ar fi: viteza, capacitatea, precizia, frecventa.
- Temperatura este masurata cu o precizie de 1 grad Celsius

Cerinte pentru testarea de acceptare

Cerinte de portabilitate

- "Nici o parte a software-ului nu trebuie scrisa in assembler."

Cerinte de intretinere

- "Timpul de reparare a unei erori nu va depasi niciodata o saptamana"

Cerinte de fiabilitate

- "Timpul minim intre doua caderi severe va fi mai mare de o luna"

Cerinte de securitate

- Cum sa fie securizat sistemul impotriva pericolelor:
 - Erori utilizator (distrugerea accidentala a software-ului sau datelor)
 - Access ne-autorizat
 - Virusi

Cerinte de siguranta

- Protectia impotriva distrugerilor cauzate de caderile software

Măsuri ale cerințelor

Proprietate	Măsură
Viteză	Numărul de tranzacții procesate pe secundă. Timpul de răspuns la utilizator/eveniment. Timpul de refresh al ecranului.
Mărime	M Bytes Numărul de chip-uri ROM.
Ușurință în utilizare	Timpul de instruire (antrenament). Numărul de cadre (frames) al help-ului.
Fiabilitate	Timpul mediu între defectări. Probabilitatea indisponibilității. Rata de apariție a defectelor. Disponibilitate.
Robustețe	Timpul de relansare după defect. Procentul evenimentelor care produc defecte. Probabilitatea coruperii datelor la apariția unui defect.
Portabilitate	Procentul de instrucțiuni dependente de țintă. Numărul de sisteme țintă.

- Tema:
- Realizati User Requirements si Specificatiile pentru butonul de start al windowsului.

Testerul

- Detectează (observa) erori **cat mai devreme**, pentru a opri propagarea
- Confirma un **nivel de calitate a produsului**
- Lucrează împotriva programatorilor (succesul testerului = insuccesul progr.)
- Semnalează erorile într-un mod convingător și sistematic
- Confirmă rezultatele depanării

Rezolvare erori (Fixing, resolving) <> Corectarea erorilor (depanare, debugging)

Metode de **rezolvare** care nu presupun **corectare**:

- Completarea specificațiilor cu restricții de utilizare;
- Definirea unor cerințe de hardware și compatibilitate;
- Acorduri cu beneficiarii pentru tratarea problemelor create de erori.
- Anuntarea unui patch (programe corectoare) sau upgrade (amanarea publicării unor module);

Testerul

- Calitățile testerului:
 - Caracter explorator, pentru detectarea neprevăzutului
 - Insistență (și răbdare) în derularea de operații repetitive
 - Creativitate, pentru formularea de situații neprevăzute de programator
 - Perfecționism, pentru a detecta chiar și erori care nu vor corectate sau eliminate
 - Diplomatie în semnalarea erorilor (sistematic și impersonal), proces care frustrează programatorul
 - Persuasiune pentru a indica gravitatea precisă a efectelor erorii (programatorii subestimează erorile)
 - Cunoștințe de programare pentru a surprinde șabloanele comportamentale ale programului sau pentru a construi instrumente automate de testare
 - Cunoștințe privind beneficiarul și domeniul de aplicație (informatica economică!)

De ce este necesara testarea?

- Sistemele software fac parte din viata noastra, fie ca este vorba de aplicatii business(ex: banca) pana la produse pentru consumatori (ex: masini)
- Majoritatea persoanelor au avut cel putin o experienta neplacuta cu un program care nu a functionat asa cum se asteptau
- Faptul ca un program nu functioneaza asa cum ne asteptam poate duce la provocarea multor probleme
- Aceste probleme pot sa fie de genul:
 - Pierderea banilor
 - Timp sau reputatia business-ului
 - Ranirea persoanelor sau moartea acestora

De ce este necesara testarea?

- O persoana umana poate sa faca o eroare(o greseala) care poate sa produca un defect(o problema, un bug) in codul programului ori in documentatia acestuia.
- Daca defectul care este produs in cod este rulat(executat) atunci programul s-ar putea sa functioneze gresit si nu o sa mai faca ceea ce ar trebui sa faca.Acest lucru va duce la creearea unui bug.
- Motive pentru care apar buguri in aplicatie:
 - Presiunea timpului
 - Complexitatea codului
 - Complexitatea infrastructurii aplicatiei
 - Schimbarea tehnologiilor
 - Interactiunea cu alte sisteme de operare

De ce este necesara testarea?

- Cu ajutorul testarii ne putem da seama de calitatea un program
- In functie de ce erori descoperim in momentul in care testam ne putem da seama de calitatea acestuia, indiferent daca folosim testarea functionala sau non-functionala pe baza specificatiilor si a documentatiei programului.
- Cu cat erorile sunt gasite mai repede cu atat calitatea programului creste si va oferi mult mai multa incredere clientului inainte de-al scoate pe piata
- Erorile care sunt gasite in timp ce programul este pe piata si trebuie fixate sunt foarte costisitoare, de aceea cu cat gasim erorile mai repede cu atat sunt mai ieftin de fixat.
- Ca testerii avem datoria de a ne asigura de calitatea unui program si de a ne asigura ca functioneaza conform specificatiilor venite de la client

De ce este necesara testarea?

- Firma de masini Nissan a trebuit sa solicite la 1 milion de cumparatori sa aduca masinile inapoi pentru ca s-a descoperit o problema la senzorii de la airbag. Pana sa isi dea seama de acest defect au fost inregistrare 5 accidente.
- Starbucks au fost fortati sa inchida aproape 60 % din magazinele din U.S si Canada din cauza unui bug in sistemul lor de POS.

Ce este testarea?

- “The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.” – **ISTQB**
- “Testing is an infinite process of comparing the invisible to the ambiguous in order to avoid the unthinkable happening to the anonymous.” - **James Bach**
- "A technical investigation of the product under test conducted to provide stakeholders with quality-related information"- **Cem Kaner**

Ce este testarea?

- Un raspuns comun oferit de majoritatea persoanelor este acela ca testarea reprezinta rulara unor teste prin executarea programului in sine. Acest lucru este doar o activitate din procesul de testare.
- Activitatile de testare exista inainte si dupa executarea testelor.
- Aceste activitati includ urmatoarele lucruri:
 - Revizuirea documentatiei programului si analizarea acesteia
 - Planificarea informatiilor
 - Alegerea conditiilor de test
 - Scrierea si executarea testelor din fiecare test design
 - Verificarea rezultatelor obtinute
 - Evaluarea rezultatelor obtinute pe baza documentatiei
 - Raportarea rezultatelor obtinute pe baza activitatilor de mai sus

Ce este testarea?

Testarea se face cu un anumit scop:

- Sa ne asiguram ca dezvoltam produsul corect
- Prevenirea defectelor
- Gasirea defectelor
- Creșterea încrederii în nivelul de calitate al produsului dezvoltat
- Input de informatie legat de calitatea unui produs