Sisteme de Încredere
- Fiabilitatea -

Ciprian Dobre
ciprian.dobre@cs.pub.ro

Fiabilitatea

- Laprie
  - Fiabilitatea reprezintă continuitatea unui serviciu (un serviciu este corect atunci când implementează funcția corectă)
- Mai pragmatic
  - Într-o perioadă de timp, pentru un model de folosire presupus, care e probabilitatea ca sistemul să se defecteze?
    - Defect = devierea de la specificația sistemului
      - Pentru unele sisteme Defectul poate însemna devierea de la așteptările utilizatorului

Calificatori pentru evaluare

- Evaluarea fiabilității depinde de:
  - Folosirea sistemului (intenția)
  - Profilul operațional (intenția)
  - Context și mediul de folosire
  - Timpul și perioada de folosire
  - Încărcare și intensitatea folosirii
- Fiabilitatea este o funcție a tuturor acestor factori
- În cazul unor modificări, trebuie re-evaluată

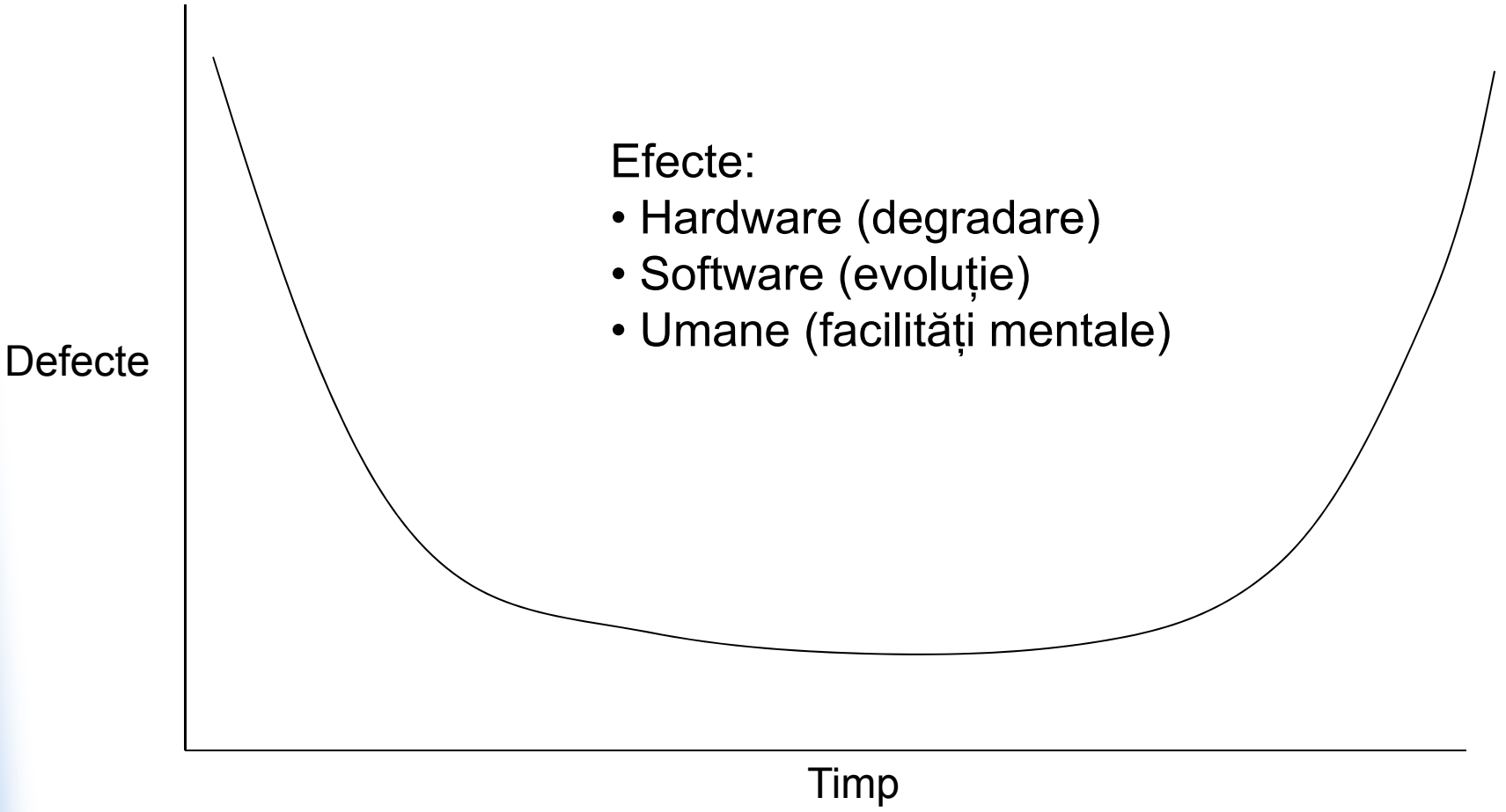
Măsură de Fiabilitate

- Fiecare este adecvată pentru diverse sisteme
  - POFOD - Prob. de apariție a defectelor la cerere (în experimente statistice)
  - ROCOF – Rata de apariție a unui defect
  - MTTF - Mean time to failure
- Trebuie alese unități de măsură adecvate
  - Perspectiva fizică vs. logică

Unități de măsură???

- Determinarea unor unități de măsură pentru:
  - Retrageri de la ATM — Nr. de tranzacții
  - Editarea folosind un procesor word — Minute
  - Un Web server ce furnizează pagini — Cereri
  - Diagnosticul pacienților de către medic — Boli diagnosticate
  - Oprirea unui reactor nuclear — Alerte

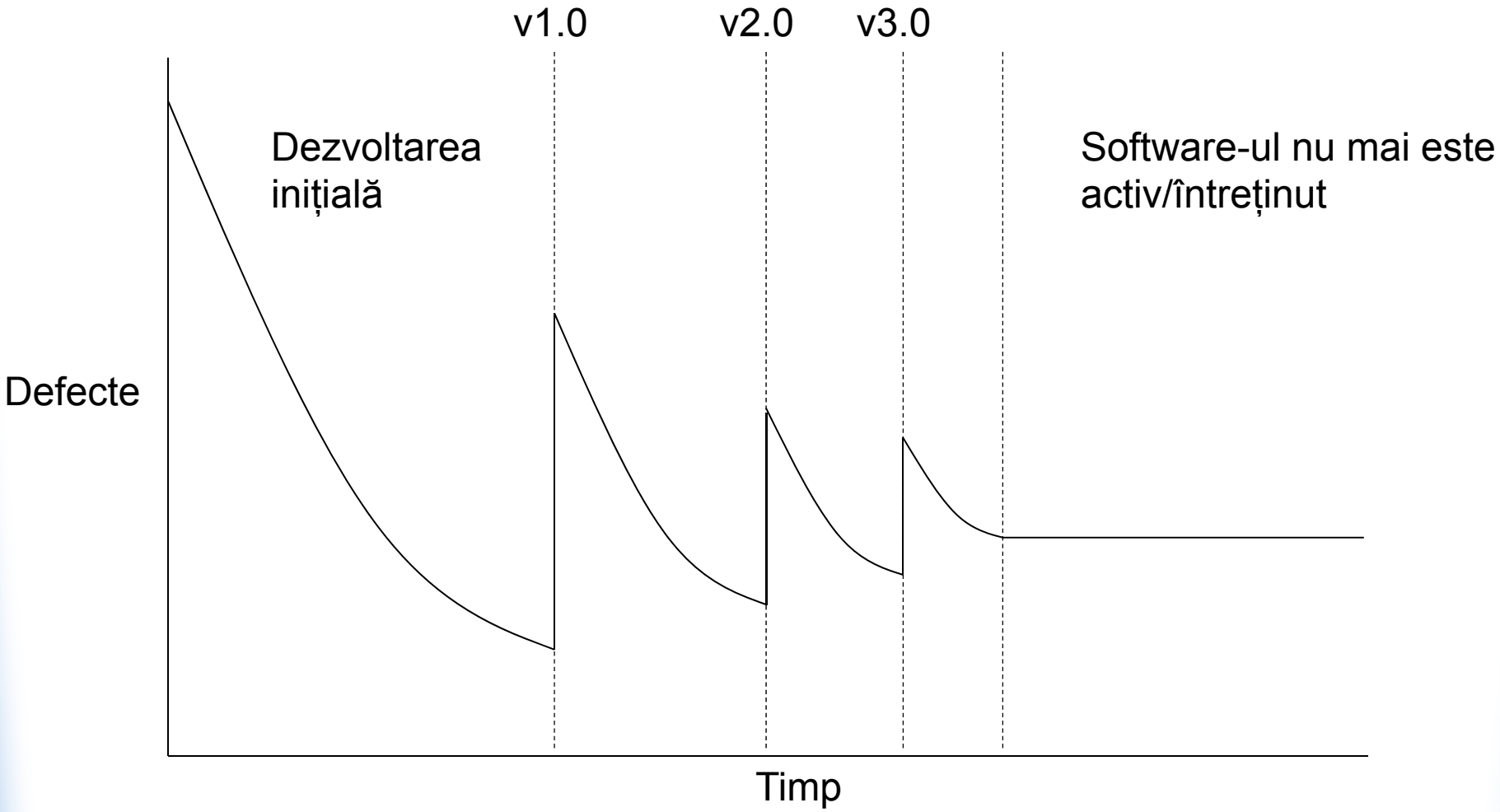
Bath tub curve



Software & curba defectelor…

- Curba anterioară reprezintă o schemă a probabilității apariției defectelor hardware
- Diferită în cazul aplicațiilor software
  - Pe perioade de timp trendul este similar
  - Însă operații de upgrade conduc la scăderea fiabilității
    - De obicei urmate de alte perioade de scădere conform curbei
  - În mod ideal, efectele cauzate de upgrade scad în timp
  - Însă, de îndată ce software-ul nu mai este întreținut curba de fiabilitate tinde să rămână constantă

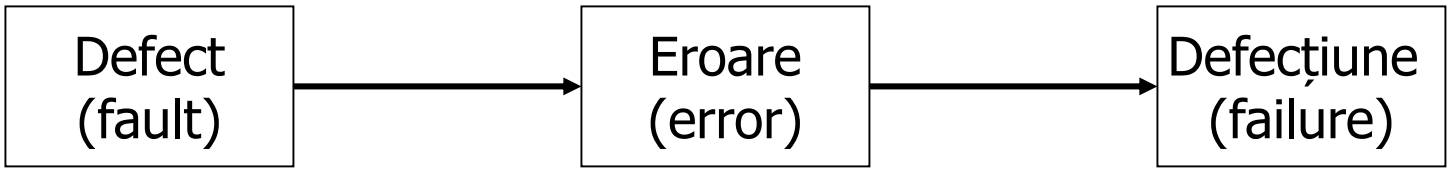
Curba disponibilității software



Însă toate acestea sunt doar modele

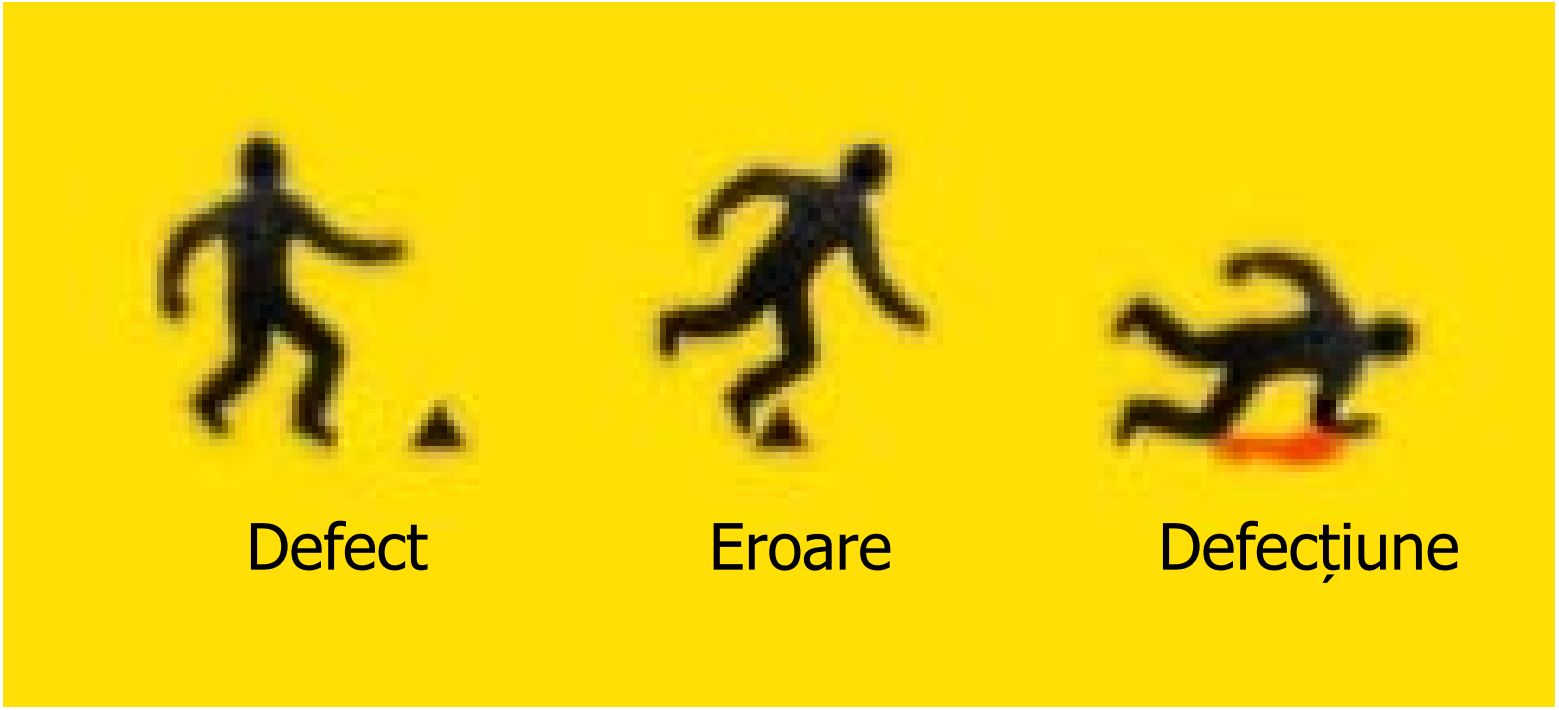
- Sistemele reale sunt construite atât din software, cât și hardware…
- …și mai intră și factorul uman?
- Cum arată curbele de apariție a defectelor în cazul unor sisteme reale?
  - Scădere a fiabilității în perioade = training/familiarizare
  - Poate căpăta scăderi bruște în timp
  - Modificările organizaționale au efecte
    - Ex: încărcare/stress ridicat afectează capacitatatea mentală
  - Modificările personale au efecte

Manifestările defectelor



- Defect
  - Cauza adjudecată sau ipotetică a erorii. De obicei constă într-o greșeală sau lipsa pregătirii corecte a unei componente.
- Eroare
  - Deviația inițială de la starea sistemului care conduce la apariția defecțiunii. De obicei constă într-un comportament neintenționat/neașteptat.
- Defecțiune
  - Deviația de la funcționarea corectă a serviciului (ex., de la specificații sau funcția sistemului).

Cu alte cuvinte…



Exemple

- Defect
  - Erori de programare
  - Training de slabă calitate
  - Pini îndoiți pe procesor
- Eroare
  - Calcularea incorectă a unui operații în virgulă mobilă
  - Completarea incorectă a unor documente
  - Dispariția unui semnal pe o placă hardware
- Defecțiune
  - Abatarea de la traseul de navigație corect
  - Lipsa unui tratament corect aplicat unui pacient
  - Alarma pentru hoți nu se declanșează

Clasificarea defectelor

- Se pot clasifica din mai multe perspective:
  - Faze ale creației/apariției
    - Defecte de dezvoltare / Defecte operaționale
  - Limitele sistemului
    - Defecte interne / externe
  - Cauze fenomenologice
    - Defecte naturale / Defecte având cauză umană
  - Dimensiuni
    - Defecte hardware / software
  - Obiective
    - Defecte malițioase / ne-malițioase

Clasificarea defectelor (2)

- Se pot clasifica din mai multe perspective:
  - Intenție
    - Defecte deliberate / ne-deliberate
  - Capabilitate
    - Defecte accidentale / cauzate de incompetență
  - Persistență
    - Defecte permanente / tranziente

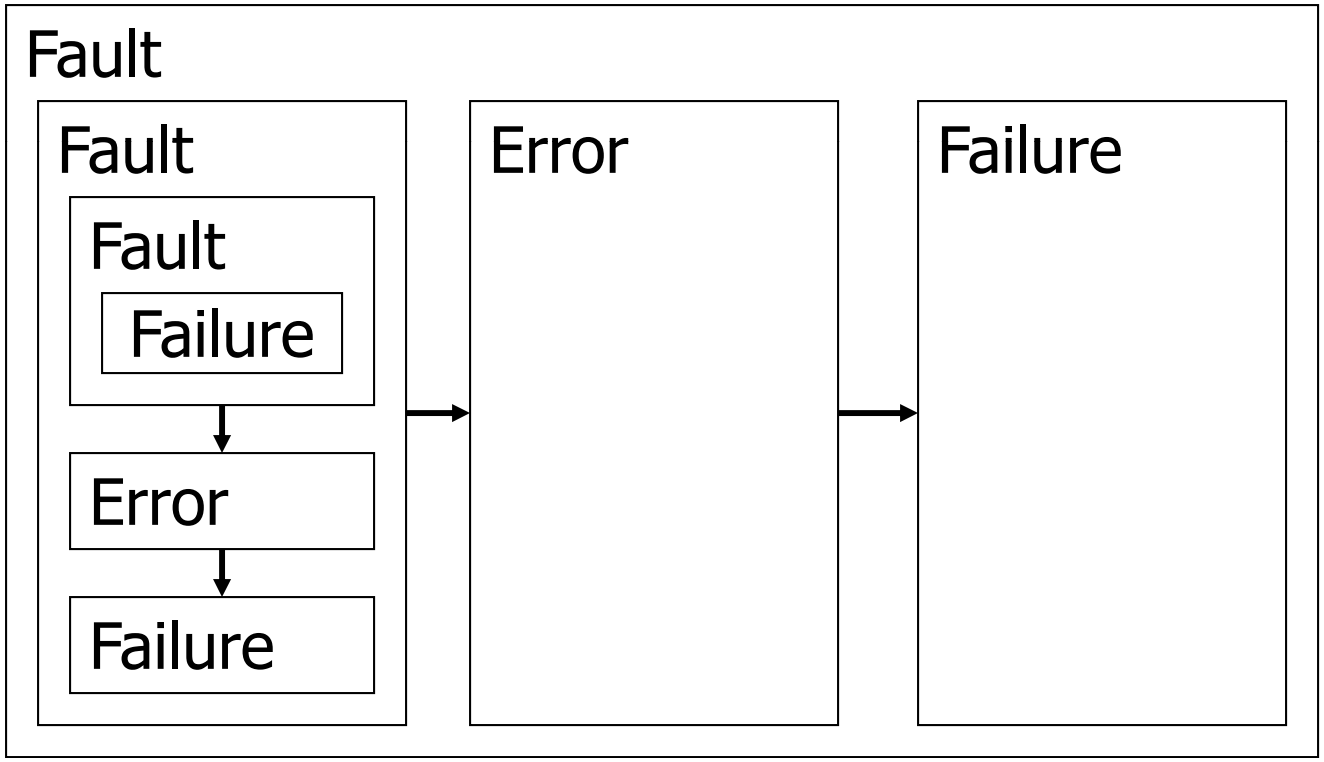
Clasificarea defecțiunilor

- Din nou, apar mai multe perspective:
  - Domeniul defecțiunii
    - Content failure / Timing failure
  - Detectibilitate
    - Defecțiuni semnalate / nesemnalate
  - Consistență
    - Defecțiuni consistente / inconsistene (bizantine)
  - Consecințe
    - Defecțiuni minore / catastrofice

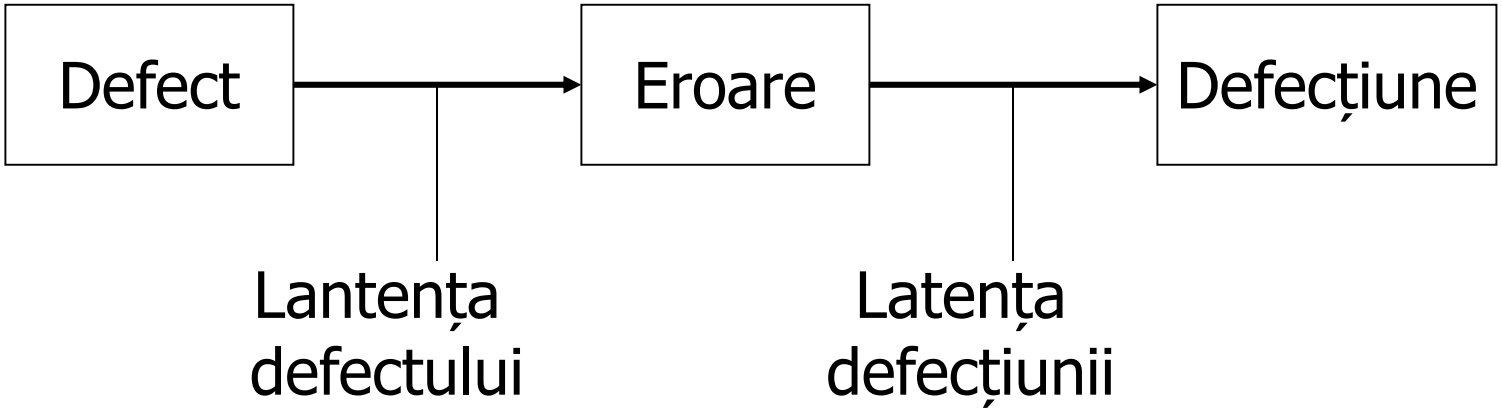
Defect – Eroare – Defecțiune

- O cauză comună de confuzie apare ca rezultat al perspectivei asupra sistemului:
  - Defecțiune cauzată de mintea umană ⇒
  - Eroare de programare ⇒
  - Defect software ⇒
  - Eroare software ⇒
  - Defecțiune a sistemului

Cazul general

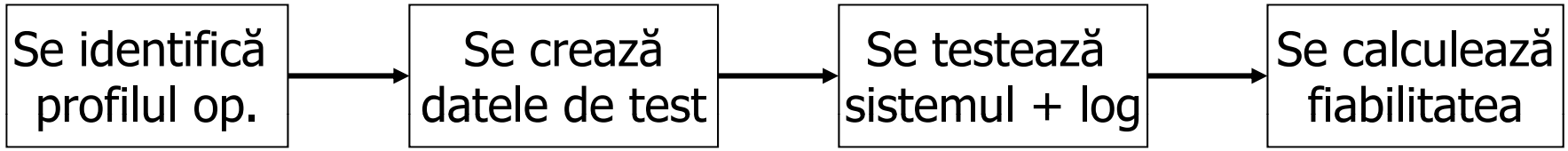


Latențe



- Defectele pot fi nedetectate o lungă perioadă de timp
- Defectele pot fi mascate (ex, nu ajung niciodată să declanșeze o eroare) sau active
- Erorile interne se poate să nu afecteze niciodată starea externă a sistemului

Teste statistice de fiabilitate

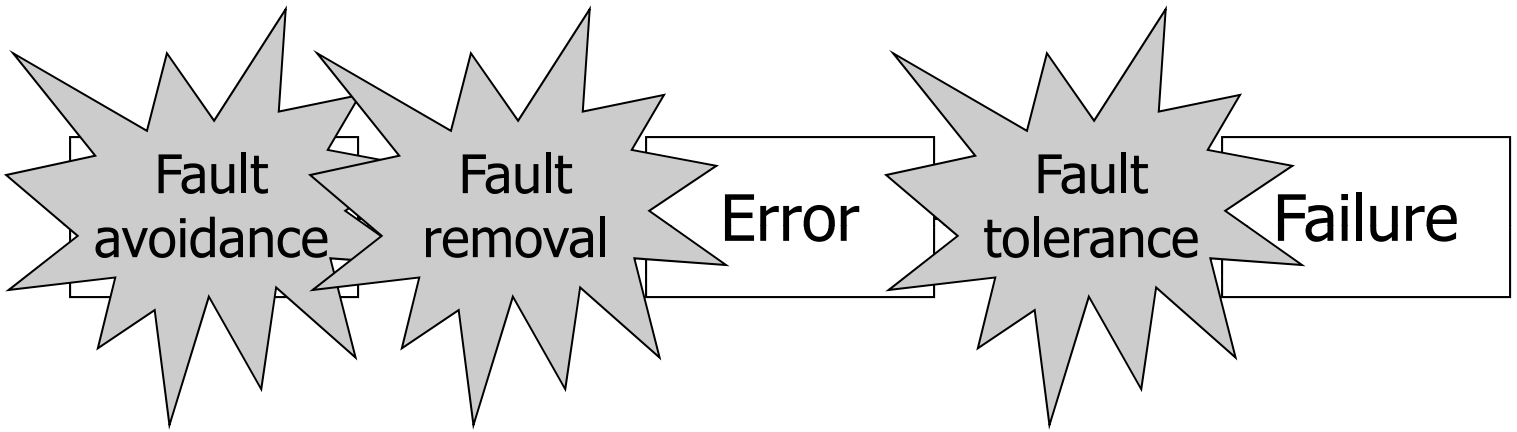


- Greu de identificat profile operaționale:
  - Nu există un pattern de folosire “standard”
  - Modele particulare de folosire și utilizatori “Maverick”
  - Modelele de folosire pot fi dinamice în timp
- Greu de efectuat un număr semnificativ de teste pentru o încredere sporită în fiabilitatea astfel calculată

Testare statistică automatizată

- Conduc la captarea profilelor operaționale
- Auto-generarea de seturi de test minimale
- Folosite pentru evaluarea fiabilității sistemului
- Încă ne-realiste/insuficiente pentru anumite sisteme

Prevenirea defectelor



Evitarea defectelor
Înlăturarea defectelor
Tolerarea defectelor – ne împiedicăm, dar ne recăpătăm echilibrul

Este de preferat să evităm defectele decât să le tolerăm
(prevenirea este mai bună decât tratamentul)
S-ar putea să nu reușim să ne recâștigăm echilibrul !

Evitarea defectelor

- Prevenirea introducerii de noi defecte
  - Dezvoltare controlată
  - Metode formale
  - Cultura calității la nivel organizațional

Dezvoltare controlată

- Folosirea unui proces de dezvoltare matur (Proces Certificat?)
- Gestiunea și controlul:
  - Cerințelor și proiectului
  - Evoluției
  - Testării
  - Configurării
  - Documentării
- Auditare
- Review-uri

Metode formale

- Specificarea sistemului folosind un limbaj formal
- Vocabular, sintaxă, semantică bine definite
- Bazate pe elemente împrumutate din matematică, teoria mulțimilor, logică, etc.
- Spec. pot fi procesate formal

Beneficii ale metodelor formale

- Reducerea ambiguităților & neînțelegerilor
- Analiza automată a:
  - Consistenței
  - Corectitudinii
  - Completitudinii
- Specificațiile pot fi emulate sau simulate
- Verificarea conduce la dezvoltarea de sisteme folosind demonstrații
- Verificarea diverselor proprietăți ale sistemului
- Transformarea în scopul construirii sistemului

Calcul propozițional

- Bazat pe “fraze” și “propoziții”
- Permite operatori “implică”, “negare”, “și”, “sau”
- Expresii și demonstrații

Notational conventions: Let *G* be a variable ranging over sets of sentences. Let *A*, *B*, and *C* range over sentences. For "*G* syntactically entails *A*" we write "*G* proves *A*". For "*G* semantically entails *A*" we write "*G* implies *A*".
We want to show: (*A*)(*G*)(if *G* proves *A*, then *G* implies *A*).
We note that "*G* proves *A*" has an inductive definition, and that gives us the immediate resources for demonstrating claims of the form "If *G* proves *A*, then ...". So our proof proceeds by induction.

Basis. Show: If *A* is a member of *G*, then *G* implies *A*.

Basis. Show: If *A* is an axiom, then *G* implies *A*.

Inductive step (induction on *n*, the length of the proof):

Assume for arbitrary *G* and *A* that if *G* proves *A* in *n* or fewer steps, then *G* implies *A*.

For each possible application of a rule of inference at step *n* + 1, leading to a new theorem *B*, show that *G* implies *B*.

Logică bazată pe predicate

- Extensii ale propositional calculus
- Logică mai “puternică”
- Permite folosirea de variabile
- Cuantificatori “pentru toți” (universali)
- Cuantificator “dacă există” (existențial)

Metode “populare”

- OBJ – Orientat obiectual, limbaj executabil
- VDM – Bazat pe stări și operații
- Z – Teoria mulțimilor și scheme grafice
- Lotos – Sisteme paralele & concurente
- CCS – Sisteme paralele & concurente

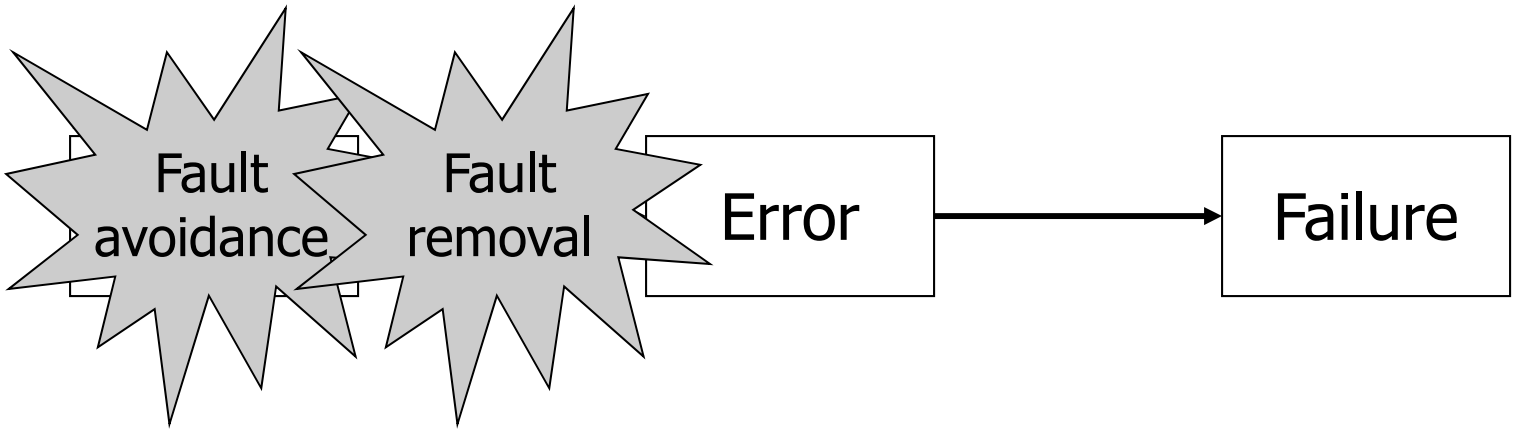
Probleme ale metodelor formale

- Consumatoare de timp și costisitoare
- Greu de înțeles (specialiști, nu e “fun”)
- Experții pe domenii au dificultăți în folosire
- Problemele pot fi chiar acoperite de formalism
- Transformarea sistemului poate fi greoaie și dificilă
- Suportul instrumental poate fi problematic
- Cum putem să dovedim că însăși specificația este corectă?
- Nu există un singur limbaj care să acopere toate cazurile

Aplicarea metodelor formale

- Sunt utile pentru sub-sisteme particulare
- Sunt utile pentru sub-probleme specifice (ex., verificarea siguranței)
- Balansează costurile de producție doar dacă sunt folosite corespunzător
- Folosire limitată în industrie
- Încă nu sunt disponibile pe scară largă

Înlăturarea defectelor



Înlăturarea defectelor

- Detecția și înlăturarea defectelor existente
  - Testare și depanare
  - Review și inspecții
  - Analiză statică a codului
    - Nu implică execuția codului
    - Instrumente, metrici software

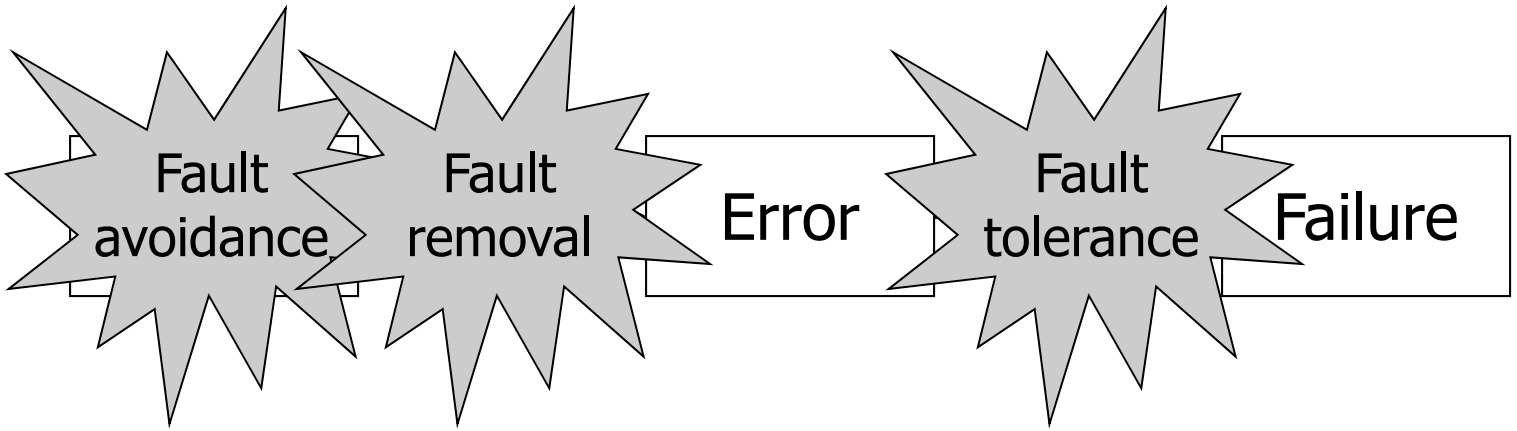
Testare

- Alpha/Beta – Cod operațional de acceptare / de producție
- Black/White box – Componente opace / transparente
- Testare funcțională / structurală
- Teste pentru defecte / statistice – căutare explicită / folosire normală
- Teste de unitate / de integrare – testare la nivelul unei componente / întregului sistem
- Teste de regresie – set de teste repetitive după fiecare reparație
- Stress test – testarea sistemului “la limită”, încercarea de forțare a sistemului la granițele specificației inițiale

Review-uri și inspecții

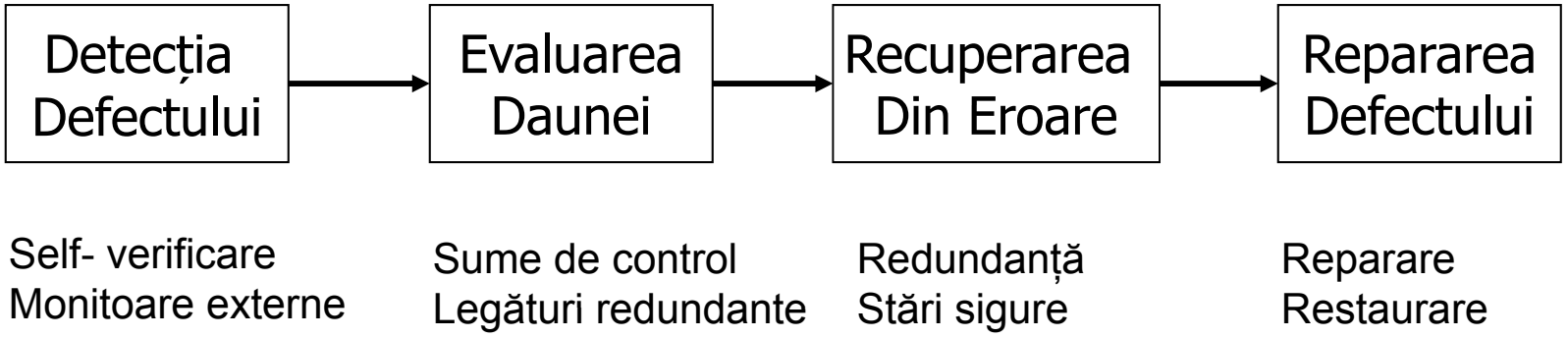
- Focus pe artefacturile produse
- Nu necesită un sistem operațional
- Examinarea & criticarea artefacturilor produse
- Bazate pe revizii din partea unor experți – respectiv, revizii cu implicarea de specialiști din mai multe discipline
- Necesită înțelegerea artefacturilor și a domeniului
- Mai puțin costisitoare decât testarea
- Nu toate problemele pot fi identificate
- Folosite pentru evaluarea atributelor non-testabile

Toleranța la defecte



Toleranța la defecte

- Gestiunea defectelor și a erorilor rezultate
- Prevenția propagării defecțiunilor

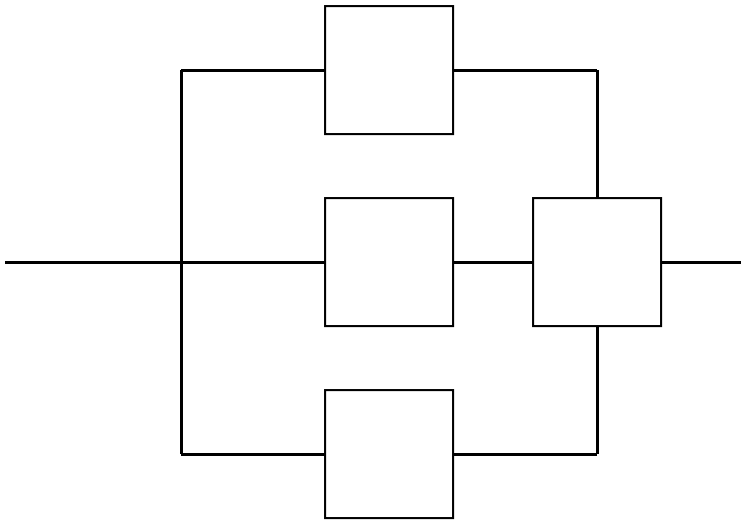


Aserțiuni

- Verificări făcute la run-time
- Efectuate periodic
- Verificare asupra siguranței stării curente a sistemului
  - Suntem siguri asupra stării înainte de continuare?
- Cotate manual / sau / auto-generate

Redundanța modulară

- Redundanță modulară triplă (TMR)
- 3 componente efectuează același task în același timp
- Comparator la ieșire
- Decizie luată prin vot

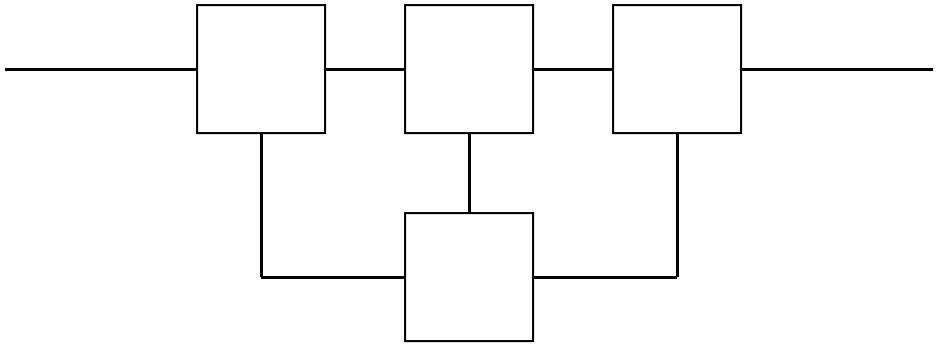


Probabilitatea de apariție a defectelor

- Probabilitate mică ca toate modulele să se defecteze simultan
- Cu condiția ca funcționarea modulelor să fie independentă !!!
- Soluția poate scala până la sisteme “N-version”
- Completată cu soluții de reparare/înlocuire automată a modulelor

Blocuri de recuperare (Recovery Blocks)

- Componente redundante folosite în serie
- Teste de acceptanță folosite pentru evaluarea rezultatelor
- Dacă o componentă se defectează, se încearcă următoarea
- Se încearcă reluarea stării anterioare înainte de încercarea următoare
- Testele continuă până când ajungem la succes sau nu mai avem componente



Abordări bazate pe comparație

- Redundanța modulară nu e foarte eficientă
- Deoarece TOATE modulele trebuie executate
- Blocurile de recuperare sunt mai buna (presupunând că nu avem defecte)
- Dar cum putem scrie un test de acceptanță ?

Diversificarea componentelor

- Diversitatea este esențială pentru aceste metode
- Atât în faza de proiectare, cât și în faza de implementare
- Fiecare componentă trebuie să folosească alte:
  - Specificații de sistem
  - Paradigme de proiectare
  - Limbaje de programare
  - Medii de dezvoltare
  - Algoritmi
  - Culturi organizaționale

Probleme cu redundanța

- Defectele duplicate pot încă exista !!!
- Oamenii încă fac aceleași greșeli
- E greu de imaginat diverse moduri de a efectua aceleași operații
- Complexitatea adăugată poate ascunde la rândul ei noi defecte
- Nu putem face teste de acceptare pentru orice
- Ce se întâmplă dacă componentele nu ajung deloc la un consens?
- Problemă legată de eficiență (importantă pentru sisteme RT)
- Poate fi costisitoare (de trei ori costul inițial al operării sistemului)

? Care e gradul de fiabilitate al acestui curs ?

Ce înseamnă evitarea erorilor în cazul acestui curs:

- Folosirea de lucrări de calitate — Evitarea defectelor (aproape metode formale;o)
- Folosirea de surse alternative — Evitarea defectelor Redundanță, diversitate
- Spell checker pentru slide-uri — Înlăturarea defectelor
- Al doilea an în care țin acest curs — Înlăturarea defectelor Testare
- Acoperim materia împreună — Redundanță
- Ne verificăm reciproc prezentările — Înlăturarea defectelor - review
- Comentariile voastre asupra cursului — Toleranța la defecte

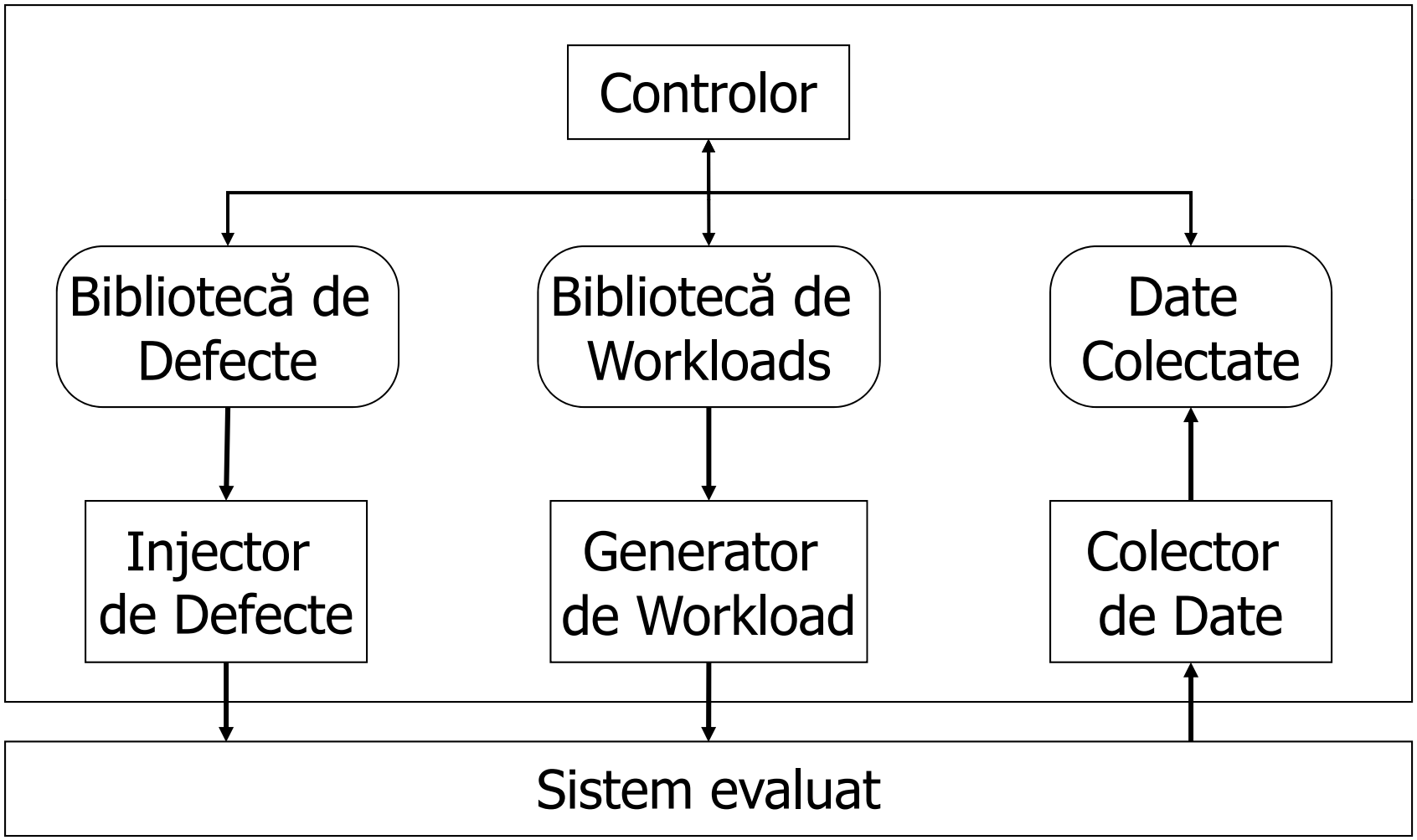
Injectarea defectelor (Fault injection)

- Evaluarea sistemului sau a sub-componentelor acestuia
- Suită de teste pentru evaluarea toleranței la defecte
- Create artificial prin introducerea de defecte
- Pot fi efectuate asupra:
  - Simulării unei componente
  - Componentei reale supusă unui load de test
  - Componentei reale în condiții reale de folosire
- Fiecare abordare are pros și cons

Folosirea injecției de defecte

- Identificarea problemelor de fiabilitate / încredere
- Studierea comportamentului sistemului în prezența defectelor
- Evaluarea mecanismelor de detecție a erorilor
- Evaluarea mecanismelor de recuperare
- Evaluarea mecanismelor de reparare din eroare

Injectarea defectelor



Tipuri de injectare

- Injectare la compilare
- Injectare la run-time
- Injectare interactivă

- Adăugarea de entități noi
- Alterarea celor existente
- Înlăturarea unor entități vechi

Probleme cu injectarea defectelor

- Profile operaționale nerealiste
- Consumatoare de timp pentru sisteme complexe
- Instrumentele pot interfera cu operațiile
- Limitare în cazul componentelor Software și Hardware

Discuție de grup

Marius lucrează la birou. El folosește un PC cu un sistem de operare obișnuit. Majoritatea activităților le realizează folosind o suită Office standard (word, etc.). Atunci când intervine plictiseala, Marius navighează pe Web și schimbă e-mail-uri cu prietenii. Câinele lui Marius se numește Azorel.

Computerul lui Marius nu e foarte fiabil și adesea acesta își pierde munca. Care sunt posibilele cauze ale lipsei de fiabilitate? Ce ar putea fi făcut pentru îmbunătățirea fiabilității?

Se vor considera atât perspectivele sociale, cât și cele tehnice.

Câteva soluții

- Este sistemul nefiabil ? Din perspectiva lui Marius – desigur
- Hardware vechi sau defect (ex, memoria)
- Software de slabă calitate
- Lipsa unor update-uri periodice
- Marius a avut parte de un training neadecvat
- Fire de câțel în interiorul unității CD
- Interfața cu utilizatorul neadecvată (OS și suita)
- Folosire neanticipată a suitei Office (ex, jocuri Excel)
- Infecție cu virus cauzată de e-mail-uri
- Entități auxiliare s/w slab calitative (ex, browserul)

Câteva soluții

- Verificarea hardware-ului și repararea acestuia
- Instalarea ultimelor update-uri s/w
- Un comportament pro-activ din partea lui Marius (salvări regulate)
- Proceduri de back-up
- Evitarea folosirii funcțiilor cu problemă (ex., tabele)
- Redundanță – replicarea muncii lui Marius
- Mai bună testare, review-uri, inspecții, etc.
- Modelarea formală a biroului lui Marius (?)
- Studii etnografice ale biroului lui Marius (?)