Verificare și validare software. Introducere

Marius Minea

26 septembrie 2013

Tehnici ce vor fi discutate

Testare black-box (fără acces la sursă)
Testare white-box/glass-box (cu acces la sursă)
  – Generarea de teste la nivel de modul
  – Metrici de acoperire cu teste
Analiză statică a codului sursă
Analiză dinamică și profilare
Testarea programelor orientate pe obiecte
Testarea programelor concurente
Verificare (formală) a modelelor
Generarea automată de teste
  – bazată pe specificație / model
Testarea securității software
Conceperea unui plan de test

Erori au fost, erori sunt încă ...

Erori grave: Therac-25

- aparat medical pentru terapie cu radiație
- 6 accidente cu morți și răni grave (1985-87, SUA, Canada)
- cauza directă: erori in programul de control

Erori grave: Therac-25

- aparat medical pentru terapie cu radiație
- 6 accidente cu morți și răni grave (1985-87, SUA, Canada)
- cauza directă: erori in programul de control

Analiză retrospectivă [Leveson 1995]:
încredere excesivă în software (în analiza produsului)
fiabilitate $\neq$ siguranță
lipsa măsurilor de siguranță hardware
lipsa practicilor de ingineria programării (proiectare defensivă, specificare, documentație, simplitate, analiză formală, testare)
corectarea unei erori nu face sistemul mai sigur !!

Erori: racheta Ariane 5

- Autodistrugere după o defecțiune la 40 s de la lansare (1996)
- Cauza: conversia 64-bit float $\rightarrow$ 16-bit int generează o excepție de depășire netratată în programul ADA
- Cost: 500 milioane dolari (racheta), 7 miliarde dolari (proiectul)

Erori: racheta Ariane 5

- Autodistrugere după o defecțiune la 40 s de la lansare (1996)
- Cauza: conversia 64-bit float $\rightarrow$ 16-bit int generează o excepție de depășire netratată în programul ADA
- Cost: 500 milioane dolari (racheta), 7 miliarde dolari (proiectul)

Analiză retrospectivă

principala cauză: reutilizarea nejudicioasă de software

cod preluat de la Ariane 4, fără reanalizare corespunzătoare:

- execuția nu mai era necesară in timpul erorii
- analiza absenței depășirii pentru variabilele neprotejate

$\Rightarrow$ necesitatea specificării și respectării unei interfețe

proiectarea greșită a redundanței: sistemul de referință inerțial și cel de rezervă scoase din funcțiune de aceeași eroare

Eroarea din procesorul Pentium

Eroare în unitatea de împățire cu virgulă mobilă, 1994
algoritm de împărțire cu refacere, în baza 4
determină următoarea cifră din cât dintr-un tabel
câteva intrări marcate greșit ca "don't care"
cost: cca. 500 milioane dolari

Eroarea din procesorul Pentium

Eroare în unitatea de împățire cu virgulă mobilă, 1994
algoritm de împărțire cu refacere, în baza 4
determină următoarea cifră din cât dintr-un tabel
câteva intrări marcate greșit ca "don't care"
cost: cca. 500 milioane dolari

Analiză retrospectivă
Circuitul putea fi verificat formal
- demonstrare automată de teoreme
- cu structuri speciale de date pentru reprezentarea înmulțirii / împărțirii
dar accentul a fost pus pe componente mai complexe
(unitatea de execuție, coerența memoriei cache)

Erori: probele trimise pe Marte

Mars Pathfinder, 1997
ajunsă pe Marte, proba spațială se reseta frecvent
cauza: inversiune de prioritate între procese cu resurse comune
fenomenul și soluția: cunoscute în literatura de specialitate !
[Sha, Rajkumar, Lehoczky. Priority Inheritance Protocols, 1990]

1. procesul A de prioritate mică obține resursa R
2. A întrerupt de C (prioritate mare)
3. C așteaptă eliberarea lui R; A revine în execuție
4. A întrerupt de B (prioritate medie, $A < B < C$)

$\Rightarrow$ C așteaptă după B, deși nu depinde de el și are prioritate mai mare!
Soluția: ridicarea priorității unui proces care obține o resursă (A)
la nivelul celui mai prioritar proces care poate solicita resursa (C)

Erori: probele trimise pe Marte

Mars Climate Orbiter, 1998
dezintegrare la intrarea în atmosferă
eroarea tehnică: discrepanța între unități de măsură în sistemele anglo-american și metric
erori multiple de proces: lipsa unor interfețe formale

Mars Polar Lander, 1998
trenul de aterizare e activat prematur la intrarea în atmosferă
șocul e interpretat ca aterizare, motoarele sunt oprite
eroarea: lipsa testării de integrare

Erorile sunt multe, și consecințele foarte grave

Citiți:
Forum on Risks to the Public in Computers and Related Systems
http://catless.ncl.ac.uk/Risks

Verificare și validare: terminologie

Software Engineering Institute – Capability Maturity Model Integration:

Verificarea asigură că produsul e construit în concordanță cu cerințele, specificațiile și standardele.

Sunt îndeplinite cerințele specificate ?
Produsul e construit corect (*cum trebuie*) ?
(*are we building the product right?*)

Validarea asigură a produsul va fi utilizabil pe piață.

Produsul acoperă nevoile operaționale ?
Produsul poate fi utilizat în mediul intenționat ?
Se construiește produsul *care trebuie* ?
(*are we building the right product?*)

V & V: terminologie

NASA Software Assurance Guidebook and Standard:

V&V: Procesul de a asigura că produsul software:
– va satisface cerințele (funcționale și altele) – *validare*
– și fiecare pas în construirea sa rezultă un produs corect – *verificare*

"Diferența dintre verificare și validare e importantă doar pentru teoretician; practicienii folosesc V&V referindu-se la toate activitățile care asigură că software-ul va funcționa conform cerințelor".

V&V: proces și tehnologii

Tehnologii

– inspecție
  review (analiză de către un terț)
  inspection (ca mai sus, dar cu proces si roluri riguros definite)
  walkthroughs (prezentare solicitând opinii)
– testare
– analiză și verificare formală

Proces

– faza de conceptie: stabilirea planului de testare
– analiza cerințelor: scenarii de test pe baza celor de utilizare
– design: verificarea modelului în raport cu specificarea
– implementare: inspecție + testare la nivel de modul
– integrare: testare de integrare, rapoarte de test

Ce e testarea ?

"Testarea e procesul prin care se execută un program cu intenția de a găsi erori" (Myers, The Art of Software Testing).

Ce e testarea ?

"Testarea e procesul prin care se execută un program cu intenția de a găsi erori" (Myers, The Art of Software Testing).

Aparent la fel, dar de fapt invers (și fals)
– "Testarea e procesul de a demonstra că programul nu are erori".
(imposibil doar prin testare)

Dijkstra: "Testing can be used very effectively to show the presence of bugs but never to show their absence."

$\Rightarrow$ un test *de succes* e acela care descoperă (și localizează) o eroare.

Ce e testorul ?

Rolul unui testor e *să găsească erori*

– cât mai devreme cu putință
  (costul corectării crește odată cu timpul)

– și să asigure corectarea lor
  (rapoarte, depanare, mentenanță)

(Patton, Software Testing)

Erori, cauze și cost

Cauza / sursa erorilor

– cele mai multe, cauzate de deficiențe în *specificație*
  – apoi cele originând în faza de proiectare
  – doar relativ puține (uneori sub 15%) erori directe de programare

Costul erorilor

crește, chiar exponențial, avansând în procesul de producție
  O(\$1) la specificare, O(\$1000+) după livrare

Cauzele și costul erorilor (cont.)

Dinamica erorilor în software [după John Rushby, SRI]
20-50 erori/kloc înainte de testare, 2-4 după
examinarea formală a codului: 10x reducere de erori înainte de testare

Cauzele și costul erorilor (cont.)

Dinamica erorilor în software [după John Rushby, SRI]
20-50 erori/kloc înainte de testare, 2-4 după
examinarea formală a codului: 10x reducere de erori înainte de testare

Studiu de caz pe 10kloc timp real, distribuit:
verificare și validare: 52% cost (57% timp)
din acesta, 27% cost în examinare, 73% în testare
21% pt. 4 defecte în testarea finala, din care 1 de proiectare
eliminarea prin examinarea codului: 160x mai eficientă decât la testare

Cauzele și costul erorilor (cont.)

Dinamica erorilor în software [după John Rushby, SRI]
20-50 erori/kloc înainte de testare, 2-4 după
examinarea formală a codului: 10x reducere de erori înainte de testare

Studiu de caz pe 10kloc timp real, distribuit:
verificare și validare: 52% cost (57% timp)
din acesta, 27% cost în examinare, 73% în testare
21% pt. 4 defecte în testarea finala, din care 1 de proiectare
eliminarea prin examinarea codului: 160x mai eficientă decât la testare

Studiu după NASA JPL (sondele Voyager, Galileo)
majoritatea: deficiențe în specificarea cerințelor și a interfețelor
1 eroare la 3 pagini de cerințe și 21 pagini de cod
doar 3 din 197 erau erori de programare
2/3 din erorile funcționale: omisiuni în specificarea cerințelor
majoritatea erorilor de interfață: datorate proastei comunicări

Calitățile testorului software

What Makes a Good Software Tester (Patton)

They are explorers.
They are troubleshooters
They are relentless
They are creative
They are (mellowed) perfectionists
They exercise good judgment
They are tactful and diplomatic (?)
They are persuasive (!)

Principii ale testării (Myers)

Un caz de test trebuie să definească ieșirea sau rezultatul dorit.
– pare evident dar ... daca nu, vom vedea ceea ce dorim să vedem

Un programator ar trebui să evite să-și testeze propriul program.
– psihologic, nu dorește să găsească erori
– excepție: dezvoltarea simultană cu testarea unitară (TDD)
Corolar: grupul de testori ar trebui să fie altul decât cel de dezvoltare

Sunt necesare cazuri de test pentru condiții de intrare valide și invalide.
Trebuie testat că produsul face ce trebuie și nu face ce nu trebuie !

Păstrați și refolosiți cazurile de test!

Nu planificați procesul de testare presupunând că nu vor fi găsite erori!

Probabilitatea de a găsi erori într-un fragment de cod e proporțională cu numărul de erori deja găsite.

“Axiome” ale testării (Patton)

Testarea software e un exercițiu de apreciere a riscurilor

Cu cât mai multe erori găsești, cu atât mai multe sunt

Paradoxul pesticidelor (Beizer): erorile devin reziliente la teste (pentru a găsi erori noi e nevoie de teste noi)

Nu toate erorile găsite vor fi corectate

E greu de spus când o eroare e o eroare ...

Specificațiile produselor nu sunt niciodată definitive

Testorii nu sunt cei mai populari membri ai echipei de proiect :)

Testarea de software e o profesie tehnică guvernată de o disciplină

Disciplina testării: organizare și metrici

Exemplu de raport succint de testare
(Marnie Hutcheson, Software Testing Fundamentals)

"As per our agreement, we have tested 67 percent of the test inventory [...] the most important tests in the inventory as determined by our joint risk analysis.

The bug find rates and the severity composition of the bugs we found were within the expected range. Our bug fix rate is 85 percent.

It has been three weeks since we found a Severity 1 issue. There are currently no known Severity 1 issues open. Fixes for the last Severity 2 issues were regression-tested and approved a week ago. Overall, the system seems to be stable.

The load testing has been concluded. The system failed at 90 percent of the design load. The system engineers [...] will need 3 months to implement the fix.

Our recommendation is to ship on schedule, with the understanding that we have an exposure if the system utilization exceeds the projections before we have a chance to install the previously noted fix."

Testarea – întrebări fundamentale

[Cem Kaner, Black-box software testing course, Florida Inst. of Tech]

Ce testăm ? Ce vrem să aflăm din asta ?

Care e misiunea testării ?

Cum organizăm lucrul pentru a îndeplini misiunea ?

Problema strategiei testării

Când am testat destul ?

Problema măsurării în testare

Testarea – o viziune mai generală

"o investigație tehnică a produsului de testat efectuată pentru a oferi persoanelor implicate informație legată de calitate" [Kaner]

investigație: căutare *activă, organizată* de informații

tehnică: experimente, logică, modele, algoritmi, unelte

produsul testat: ce primește clientul, în totalitate
(software, hardware, baze de date, documentație, etc.)

persoane implicate: în succesul produsului, și al testării

Scopuri ale testării

[Kaner 2003 – What is a good test case ?]
– găsirea de defecte: mai ales în părțile "interesante" (acoperire bună)
– găsirea cât mai multor defecte: în timp limitat
– oprirea livrării premature / suport pentru decizie: livrare sau nu
– minimizarea costului de suport tehnic
– evaluarea conformanței la specificații / reguli / standarde
– minimizarea riscurilor de accidente (și costurilor legale ...)
– găsirea de scenarii de utilizare sigură (funcționare corectă)
– evaluarea calității produsului (dar nu doar prin testare)

Ce nu poate face testarea ?
– verificarea produsului (absența de erori)
– asigurarea calității produsului (problemă de proces)

Calitățile unui bun caz de test

puternic: șansă mare de a descoperi o anumită eroare dacă există

convingător (problemă importantă) și credibil (e realist să apară)

reprezentativ / probabil pentru client

ușor de evaluat (e o eroare sau nu?) / ușor de depanat / informativ

de complexitate potrivită (progresivă)

relevant privind funcționarea / performanța produsului
(de ex. detectează modificări în comportament / performanță)

Câteva utilitare de încercat

Klee `http://klee.llvm.org/`
testare C/C++ prin execuție simbolică

Pex `http://research.microsoft.com/en-us/projects/Pex/`
testare C# prin execuție simbolică

CHESS testarea programelor concurente
`http://research.microsoft.com/en-us/projects/CHESS/`

RoadRunner analiză dinamică pentru programe Java concurente
`hhttp://www.cs.williams.edu/~freund/rr/`

Java Pathfinder `http://babelfish.arc.nasa.gov/trac/jpf`
model checking și testare / execuție simbolică

Randoop `http://people.csail.mit.edu/cpacheco/randoop/`
testare prin generare aleatoare + feedback, pentru Java

Blast (verificator pentru C), CREST (execuție concretă + simbolică), Daikon (generare de invarianți), FramaC, ESCJava2 (analiză statică) ...