Testare black-box

3 octombrie 2013

Domenii (tipuri) de testare black-box
Partiționarea în clase de echivalență
Testarea cazurilor limită
Testarea folosind analiza cauză-efect
Testare prin explorare

Testarea “black-box”

Produsul e privit ca un sistem opac
(fără acces la detalii interne, în particular la sursă)

De ce testare black-box ?

- e aplicabilă oricărui produs
- nu necesită efort pentru examinarea / analiza sursei
- aplicabilă de la simplu la complex
- aplicabilă într-o varietate de contexte

Tipuri de testare black-box [Kaner]

Sau: *de unde începem testarea ?*

Testarea funcțiilor (function testing)
- fiecare funcție (caracteristică) separat; funcționalitate de bază
- teste nu neapărat f. "dure", suficient cât să releve erori serioase

Testarea domeniilor de valori (domain testing)
- esența: eșantionarea prin reprezentanți ai claselor de echivalență
- initial: pentru fiecare variabilă separat; evtl. combinații
- valori alese judicios $\Rightarrow$ teste puternice; informative

Testarea bazată pe specificații
- teste pentru fiecare afirmație/proprietate din doc. de specificație
- nu f. puternice (testează că funcționalitatea e satisfăcută)
- mai profund: caută erori/omisiuni/ambiguități/cazuri limită în specificație

Tipuri de testare black-box (cont.)

Testare axată pe riscuri (risk-based testing)
- se imaginează o modalitate de eșec și se generează teste pentru ea
- testele trebuie să fie *puternice, credibile, motivatoare*

Testarea la limită (stress testing): exercită programul
- 1) la efort mare
- 2) la/dincolo de limitele specificate
- 3) pentru a vedea *cum* eșuează

Testarea de regresie
- set de teste proiectate pt. refolosire (la fiecare modificare)
- bine documentate pentru mentenanță

Testarea de către utilizatori
- reali, nu simulați (beta testing);
- pe scenarii specificate dinainte, sau "la liber"
- credibile, motivatoare, nu f. puternice

Tipuri de testare black-box (cont.)

Testarea bazată pe scenarii
- caz specific de utilizare; poate fi bazat pe model (use case)
- credibil, motivator, ușor de evaluat, complex
- mai profund: scenariu de utilizare la limită / ostil

Testarea bazată pe stări (state-model-based testing)
- modelul: automat cu stări finite
- analiza modelului; și a produsului cu teste bazate pe model

Testarea automată de volum ridicat

Testarea prin explorare
- direcționează activ procesul de testare, proiectând teste noi pe baza informațiilor oferite de cele deja executate

Strategii de abordare a testării [Kaner, curs Black-Box Testing]

1. Începe cu teste simple
2. Abordează fiecare funcție pt. a-i înțelege comportamentul
3. Testează întâi prin cuprindere, apoi în adâncime
4. Treci la cazuri de test mai puternice: cazuri limită
5. Gândește-te la situații mai complexe / surpriză
6. Explorează "la liber"

Partiționarea în clase de echivalență [Myers]

Analiza domeniului de valori pentru fiecare variabilă / intrare, identificând mulțimi pt. care presupunem că testele se comportă la fel $\Rightarrow$ folosită pt. a genera un set de condiții "interesante" pt. testare

În plus: un caz de test ar trebui să acopere cât mai multe condiții relevante (să reducă numărul condițiilor de analizat cu > 1)

Pentru fiecare condiție: teste cu cazuri *valide* și *invalide*

Myers sugerează redactarea unui tabel de forma:

Condiție	Clase echiv. valide	Clase echiv. invalide

Partiționarea în clase de echivalență (cont.)

După domeniul de valori al variabilei:

Pentru un interval:
- un caz valid (în interior); două invalide (de ambele părți)
- obs: cu rafinare pt. testarea cazurilor limită

Pentru un număr specificat:
- un caz valid, două cazuri invalide (mai mare, mai mic)

Pentru enumerare: fiecare valoare, plus una invalidă

Generarea cazurilor de test:
- acoperă cât *mai multe clase valide* cu un caz de test
- generează câte un caz de test pentru *fiecare clasă invalidă*
  - (daca s-ar combina, o situație invalidă poate masca alta)

Exemplu

Declararea dimensiunilor tabloului în FORTRAN [Myers]

`DIMENSION` *array-descrp* (, *array-descrp*)*
array-descrp ::= *name* (*dim* (, *dim*)*)
name ::= *letter* (*letter* | *digit*)* (1..6 chars)
dim ::= [*lower-bound* :] *upper-bound*
bound ::= *int-constant* | *name*

-65534 $\leq$ *lower-bound* $\leq$ *upper-bound* $\leq$ 65535
lower-bound e implicit 1

Testarea condițiilor limită

Diferă de/rafinează metoda claselor de echivalență în două aspecte:

1) fiecare limită a unei clase de echivalență acoperită de un test implicit: și valorile de deasupra / sub limită

2) cazuri de test derivate și din domeniul valorilor *de ieșire*, (nu doar al celor de intrare)

Exemplu [Burnstein]: identificatori din 3-15 caractere alfanumerice, din care primele două să fie litere.

Constrangeri (fiecare cu clase de echivalenta/conditii limita)
- caractere alfanumerice
- lungime (minim - 1, minim, intermediar, maxim, maxim + 1)
- primele două caractere

Testarea folosind analiza cauză-efect

Testarea bazată pe clase de echivalență nu permite combinarea condițiilor – într-o combinație de condiții, trebuie acoperit fiecare factor în parte

Etape:

– se descompune specificația în componente de dimensiuni potrivite
– identificarea cauzelor: condiții/clase de echivalență de intrare
– identificarea efectelor: condiție la ieșire/modificare de stare
– se exprimă specificația sub formă de reguli/diagramă booleană
– se generează teste

Testarea folosind analiza cauză-efect

Exemplu [Myers]

> *The character in column 1 must be an A or a B. The character in column 2 must be a digit. In this situation, the file update is made. If the first character is incorrect, message X12 is issued. If the second character is not a digit, message X13 is issued.*

Testele se generează pornind de la ieșire (efect)
setând pe rând cauzele care ar trebui să producă efectul:
- pentru un nod SAU, individual toate cauzele *adevărate*
- pentru un nod ȘI, individual toate cauzele *false*

similar cu acoperirea MC/DC, dar pe *specificație*, nu pe *cod*

Testarea la nivele superioare: exploratory testing

Exploratory testing cf. James Bach:

simultan *învățare*, *proiectare* și *execuție* de teste

dependentă de situație

rezultatele obținute din testele determină explorarea ulterioară

Strategii de găsire a erorilor

[după James Whittaker, How to Break Software]

Perspective de testare:

1. Interfața cu utilizatorul
   black-box: intrări / ieșiri
   open box: accent pe stare, interacțiuni

2. Interfața cu sistemul
   sistemul de fișiere
   sistemul de operare (concurență, memorie, rețea, etc.)

Ce fel de teste încercăm ?

Intrări eronate (tip – ex. obiecte/imagini/fișiere de tip nepotrivit; lungime mare/mică, valori limită)

codul tratează eroarea ? semnalează cu mesaje ?

Forțarea *reinițializării*: se introduc valori nule/invalide pentru opțiuni. Sistemul revine la valorile implicite ?

Intrări cu *caractere invalide* / de control / cu înțeles special

Depășiri de lungimi permise (buffer overflow):

nu doar la introducerea datelor, ci și cum sunt folosite ulterior (limitele pot fi diferite)

Combinații / interacțiuni între intrări

două intrări mari; una mare și una f. mică

Testare *repetitivă* (parcurgerea ciclurilor)

consum de resurse (memorie); probleme de (re)inițializare

Catalog de teste (cont.)

Explorarea *unei* intrări în *contexte diferite*
  răspunsuri diferite: sunt tratate toate situațiile ?

Generare de *ieșiri invalide*
  uneori posibil pe o cale ocolitoare (e.g. 29 feb. 2000 $\rightarrow$ 2001

Atac UI: reîmprospătarea ecranului (se face complet?)

încearcă depășirea limitărilor interne
  ex. creează tablou de dimensiune maximă, adaugă un rând

Calcule cu operatori / operanzi invalizi

Testează incluziuni recursive (cadru în alt cadru; notă de subsol, etc.)