Testare functionala

Testare functionala

- Datele de test sunt generate pe baza specificatiei (cerintelor) programului, structura programului nejucand nici un rol
- Tipul de specificatie ideal pentru testarea functionala este alcatuit din pre-conditii si post-conditii
- Majoritatea metodelor functionale se bazeaza pe o partitionare a datelor de intrare astfel incat datele apartinand unei aceeasi partitii vor avea proprietati similare (identice) in raport cu comportamentul specificat

Partitionare de echivalenta (equivalence partitioning)

- Ideea de baza este de a partitiona domeniul problemei (datele de intrare) in *partitii de echivalenta* sau *clase de echivalenta* astfel incat, din punctul de vedere al specificatiei datele dintr-o clasa sunt tratate in mod identic.
- Cum toate valorile dintr-o clasa au specificat acelasi comportament, se poate presupune ca toate valorile dintr-o clasa vor fi procesate in acelasi fel, fiind deci suficient sa se aleaga cate o valoare din fiecare clasa
- In plus, domeniul de iesire va fi tratat in acelasi fel, iar clasele rezultate vor fi transformate in sens invers (reverse engineering) in clase ale domeniului de intrare

Partitionare de echivalenta (equivalence partitioning) (2)

- Clasele de echivalenta nu trebuie sa se suprapuna, deci orice clase care s-ar suprapune trebuie descompuse in clase separate
- Dupa ce clasele au fost identificate, se alege o valoare din fiecare clasa. In plus, pot fi alese si date invalide (care sunt in afara claselor si nu sunt procesate de nici o clasa)
- Alegerea valorilor din fiecare clasa este arbitrara deoarece se presupune ca toate valorile vor fi procesate intr-un mod identic

Partitionare de echivalenta - exemplu

- Se testeaza un program care verifica daca un caracter se afla intr-un sir de cel mult 20 de caractere. Mai precis, pentru un intreg *n* aflat intre 1 si 20, se introduc caractere, iar apoi un caracter *c*, care este apoi cautat printre cele *n* caractere introduse anterior. Programul va produce o iesire care va indica prima pozitie din sir unde a fost gasit caracterul *c* sau un mesaj indicand ca acesta nu a fost gasit. Utilizatorul are optiunea sa caute un alt caracter tastand *y* (yes) sau sa termine procesul tastand *n* (no).

Intrari

- un intreg pozitiv *n*
- un sir de caratere *x*
- caracterul care se cauta *c*
- optiune de a cauta sau nu an alt caracter *s*

Domeniul de intrari:

- *n* trebuie sa fie intre 1 si 20, deci se disting 3 clase de echivalenta:
  - N_1 = 1..20
  - N_2 = { n | n < 1}
  - N_3 = {n | n > 20}
- intregul *n* determina lungimea sirului de caractere si nu se precizeaza nimic despre tratarea diferita a sirurilor de lungime diferita deci a doua intrare nu determina clase de echivalenta suplimentare
- *c* nu determina clase de echivalenta suplimentare
- optiunea de a cauta un nou caracter este binara, deci se disting 2 clase de echivalenta
  - S_1 = {y}
  - S_2 = {n}

Iesiri

- Pozitia la care caracterul se gaseste in sir
- Un mesaj care arata ca nu a fost gasit

- Acestea sunt folosite pentru a imparti domeniul de intrare in 2 clase: una pentru cazul in care caracterul se afla in sirul de caractere si una pentru cazul in care acesta lipseste
  - C_1(x) = { c | c se afla in x}
  - C_2(x) = { c | c nu se afla in x}

Clase de echivalenta

- Clasele de echivalenta pentru intregul program se pot obtine ca o combinatie a claselor individuale:
  - C_111 = { (n, x, c, s) | n \in N_1, |x| = n, c \in C_1(x), s \in S_1}
  - C_112 = { (n, x, c, s) | n \in N_1, |x| = n, c \in C_1(x), s \in S_2}
  - C_121 = { (n, x, c, s) | n \in N_1, |x| = n, c \in C_2(x), s \in S_1}
  - C_122 = { (n, x, c, s) | n \in N_1, |x| = n, c \in C_2(x), s \in S_2}
  - C_2 = { (n, x, c, s) | n \in N_2}
  - C_3 = { (n, x, c, s) | n \in N_3}

Date de test

- Setul de date de test se alcatuieste alegandu-se o valoare a intrarilor pentru fiecare clasa de echivalenta. De exemplu:
  - C_111 : (3, abc, a, y)
  - C_112 : (3, abc, a, n)
  - C_121 : (3, abc, d, y)
  - C_122 : (3, abc, d, n)
  - C_2 : (0, _, _, _)
  - C_3 : (25, _, _, _)

Avantaje

- Reduce drastic numarul de date de test doar pe baza specificatiei
- Potrivita pentru aplicatii de tipul procesarii datelor, in care intrarile si iesirile sunt usor de identificat si iau valori distincte

Dezavantaje

- Modul de definire al claselor nu este evident (depinde de experienta testerului).
- In unele cazuri, desi specificatia ar putea sugera ca un grup de valori sunt procesate identic, acest lucru nu este adevarat.
- Mai putin aplicabile pentru situatii cand intrarile si iesirile sunt simple, dar procesarea este complexa.

Analiza valorilor de frontiera (boundary value analysis)

- Analiza valorilor de frontiera este folosita de obicei impreuna cu partitionarea de echivalenta.
- Se concentreaza pe examinarea valorilor de frontiera ale claselor, care deobicei sunt o sursa importanta de erori.

Exemplu

- N_1 = 1..20
  - 1, 20 (valori de frontiera) si o valoare din interior
- N_2 = { n | n < 1}
  - 0 (valoare de frontiera) si o valoare din interior
- N_3 = {n | n > 20}
  - 20 (valoare de frontiera) si o valoare din interior
- C_1(x) = { c | c se afla in x}
  - pe prima pozitie in x, pe ultima pozitie in x, in interiorul lui x
- C_2(x) = { c | c nu se afla in x}
  - nu exista frontiere clare, deci nu apar valori suplimentare

Observatie

- Modul de alegere al frontierelor nu este evident si depinde de experienta testerului.
- Exemplu: pentru alegerea frontierei poate fi considerat numarul de aparitii ale lui c in sirul x.
  - 0 aparitii, 1 aparitie, mai mult de o aparitie

Partitionarea in categorii (category-partition)

- Se bazeaza pe cele doua anterioare.
- Cauta sa genereze date de test care "acopera" functionalitatea sistemului si maximizeaza posibilitatea de gasire a erorilor.

Partitionarea in categorii - pasi

- Descompune specificatia functionala in unitati (programe, functii, etc.) care pot fi testate separat
- Pentru fiecare unitate, identifica parametrii si conditiile de mediu (ex. starea sistemului la momentul executiei) de care depinde comportamentul acesteia.
- Gaseste categoriile (proprietati sau caracteristici importante) fiecarui parametru sau conditii de mediu.
- Partitioneaza fiecare categorie in alternative. O alternativa reprezinta o multime de valori similare pentru o categorie.
- Scrie specificatia de testare. Aceasta consta in lista categoriilor si lista alternativelor pentru fiecare categorie.
- Creeaza cazuri de testare prin alegerea unei combinatii de alternative din specificatia de testare (fiecare categorie contribuie cu zero sau o alternativa).
- Creeaza date de test alegand o singura valoare pentru fiecare alternativa.

Partitionarea in categorii - exemplu

- Descompune specificatia in unitati: avem o singura unitate
- Identifica parametrii: n, x, c, s
- Gaseste categorii:
  - n : daca este in intervalul valid 1..20
  - x : daca este de lungime minima, maxima sau intermediara
  - c : daca ocupa prima sau ultima pozitie sau o pozitie in interiorul lui x sau nu apare in x
  - s : daca este pozitiv sau negativ
- *Partitioneaza fiecare categorie in alternative*:
  - *n* : <0, 0, 1, 2..19, 20, 21, >21
  - *x* : lungime minima, maxima sau intermediara
  - *c* : pozitia este prima, in interior, sau ultima sau *c* nu apare in *x*
  - *s* : y, n

Partitionarea in categorii – exemplu (2)

- *Scrie specificatia de testare*
  - *n*
    - 1) {n | n < 0}
    - 2) 0
    - 3) 1
    - 4) 2..19
    - 5) 20
    - 6) 21
    - 7) {n | n > 21}
  - *x*
    - 1) {x | |x| = 1}
    - 2) {x | 1 < |x| < 20}
    - 3) {x | |x| = 20}
  - *c*
    - 1) { c | c se afla pe prima pozitie in x}
    - 2) { c | c se afla in interiorul lui x}
    - 3) { c | c se afla pe ultima pozitie in x}
    - 4) { c | c nu se afla in x}
  - *s*
    - 1) y
    - 2) n

Cazuri de testare

- Din specificatia de testare ar trebui sa rezulte 7 * 3 * 4 * 2 = 168 de cazuri de testare.
- Unele combinatii de alternative nu au sens si pot fi eliminate.
- Alternativele vor fi combinate doar daca conditiile de selectie sunt satisfacute.
- In exemplu, numarul cazurilor de testare se reduce la 24.

Cazurile de testare - exemplu

- n1
- n2
- n3x1c1s1
- n3x1c1s2
- n3x1c4s1
- n3x1c4s2
- n4x2c1s1
- n4x2c1s2
- n4x2c2s1
- n4x2c2s2
- n4x2c3s1
- n4x2c3s2
- n4x2c4s1
- n4x2c4s2
- n5x3c1s1
- n5x3c1s2
- n5x3c2s1
- n5x3c2s2
- n5x3c3s1
- n5x3c3s2
- n5x3c4s1
- n5x3c4s2
- n6
- n7

Avantaje si dezavantaje

- Pasii de inceput (identificarea parametrilor si a conditiilor de mediu precum si a categoriilor) nu sunt bine definiti si se bazeaza pe experienta celui care face testarea.
- Pe de alta parte, odata ce acesti pasi au fost trecuti, aplicarea metodei este foarte clara.
- Este mai clar definita decat metodele functionale anterioare si poate produce date de testare mai cuprinzatoare, care testeaza functionalitati suplimentare.
- Pe de alta parte, datorita exploziei combinatorice, pot rezulta date de test de foarte mare dimensiune