Testare software

- *functionala* (*black box*): bazata pe cerinte (specificatie)
- *structurala* (*white box*): bazata pe program(implemantare)

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

1. 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
- 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

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).

1. Domeniul de intrari:

Exista 4 intrari:

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

- *n* trebuie sa fie intre 1 si 20, deci se disting 3 clase de echivalenta:

$N\_1 = 1..20$
$N\_2 = \{ n \mid n < 1\}$
$N\_3 = \{n \mid n > 20\}$

- 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}

2. Domeniul de iesiri

Consta din urmatoarele 2 raspunsuri:

- 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}

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}

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, _, _, _)

6 clase

Intrari				Rezultat afisat
n	**x**	**c**	**s**	
0				Cere introducerea unui intreg intre 1 si 20
25				Cere introducerea unui intreg intre 1 si 20
3	abc	a	y	Afiseaza pozitia 1; se cere introducerea unui nou caracter
3	abc	a	n	Afiseaza pozitia 1
3	abc	d	y	Caracterul nu apare; se cere introducerea unui nou caracter
3	abc	d	n	Caracterul nu apare

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 (nu exista nici o modalitate riguroasa sau macar niste indicatii clare pentru identificarea acestora).
- In unele cazuri, desi specificatia ar putea sugera ca un grup de valori sunt procesate identic, acest lucru nu este adevarat. Acest lucru intareste ideea ca metodele functionale trebuie aplicate impreuna cu cele structurale.
- Mai putin aplicabile pentru situatii cand intrarile si iesirile sunt simple, dar procesarea este complexa.

2. Analiza valorilor de frontiera (boundary value analysis)

Analiza valorilor de frontiera este folosita de obicei impreuna cu partitionarea de echivalenta. Ea se concentreaza pe examinarea

valorilor de frontiera ale claselor, care deobicei sunt o sursa importanta de erori.

Pentru exemplul nostru, odata ce au fost identificate clasele, valorile de frontiera sunt usor de identificat:

- valorile 0, 1, 20, 21 pentru *n*
- caracterul c poate sa se gaseasca in sirul *x* pe prima sau pe ultima pozitie

Deci se vor testa urmatoarele valori:

- N_1 : 1, 20
- N_2 : 0
- N_3 : 21

- C_1: = c_11 se afla pe prima pozitie in x, c_12 se afla pe ultima pozitie in x

- Pentru restul claselor se ia cate o valoare (arbitrara)

Deci, pentru C_111 si C_112 vom alege cate 3 date de test (x de 1 caracter si x de 20 de caractere in care c se gaseste pe pozitia 1 si pe pozitia 20), iar pentru C_121 si C_122 cate 2 date de test (x de 1 caracter si x de 20 de caractere). In total vom avea 12 date de test.

C_111 : (1, a, a, y), (20, abcdefghijklmnoprstu, a, y),
(20, abcdefghijklmnoprstu, u, y)
C_112 : (1, a, a, n), (20, abcdefghijklmnoprstu, a, n),
(20, abcdefghijklmnoprstu, u, n)
C_121: (1, a, b, y), (20, abcdefghijklmnoprstu, z, y)
C_122: (1, a, b, n), (20, abcdefghijklmnoprstu, z, n)
C_2 : (0, _, _, _)
C_3 : (21, _, _, _)

Intrari			
n	**x**	**c**	**s**
0			
21			
1	a	a	y
		a	n
1	a	b	y
		b	n
20	abcdefghijklmnoprstu	a	y
		a	n
20	abcdefghijklmnoprstu	u	y
		u	n
20	abcdefghijklmnoprstu	z	y
		z	n

Avantaje si dezavantaje:
Aceleasi ca metoda anterioara. In plus, aceasta metoda adauga informatii suplimentare pentru generarea setului de date de test si se concentreaza asupra unei arii (frontierele) unde de regula apar multe erori.

3. Partitionarea in categorii (category-partition)

Aceasta metoda se bazeaza pe cele doua anterioare. Ea cauta sa genereze date de test care "acopera" functionalitatea sistemului si maximizeaza posibilitatea de gasire a erorilor.

Cuprinde urmatorii pasi:
1. Descompune specificatia functionala in unitati (programe, functii, etc.) care pot fi testate separat

2. Pentru fiecare unitate, identifica parametrii si conditiile de mediu (ex. starea sistemului la momentul executiei) de care depinde comportamentul acesteia.
3. Gaseste categoriile (proprietati sau caracteristici importante) fiecarui parametru sau conditii de mediu.
4. Partitioneaza fiecare categorie in alternative. O alternativa reprezinta o multime de valori similare pentru o categorie.
5. Scrie specificatia de testare. Aceasta consta in lista categoriilor si lista alternativelor pentru fiecare categorie.
6. Creeaza cazuri de testare prin alegerea unei combinatii de alternative din specificatia de testare (fiecare categorie contribuie cu zero sau o alternativa).
7. Creeaza date de test alegand o singura valoare pentru fiecare alternativa.

Pentru exemplul nostru:

1. *Descompune specificatia in unitati*: avem o singura unitate
2. *Identifica parametrii*: *n, x, c, s*
3. *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
4. *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
5. *Scrie specificatia de testare*
   - *n*

     1) {n | n < 0}

2) 0
3) 1 [ok, lungime1]
4) 2..19 [ok, lungime_medie]
5) 20 [ok, lungime20]
6) 21
7) {n | n > 21}

- *x*
  1) {x | |x| = 1} [if ok and lungime1]
  2) {x | 1 < |x| < 20} [if ok and lungime_medie]
  3) {x | |x| = 20} [if ok and lungime20]
- *c*
  1) { c | c se afla pe prima pozitie in x} [if ok]
  2) { c | c se afla in interiorul lui x} [if ok and not lungime1]
  3) { c | c se afla pe ultima pozitie in x} [if ok and not lungime1]
  4) { c | c nu se afla in x} [if ok]
- *s*
  1) y [if ok]
  2) n [if ok]

Din specificatia de testare ar trebui sa rezulte 7 * 3 * 4 * 2 = 168 de cazuri de testare. Pe de alta parte, unele combinatii de alternative nu au sens si pot fi eliminate. Acest lucru se poate face adaugand constrangeri acestor alternative. Constrangerile pot fi fie proprietati ale alternativelor fie conditii de selectie bazate pe aceste proprietati. In acest caz, alternativele vor fi combinate doar daca conditiile de selectie sunt satisfacute. Folosind acest procedeu, in exemplul nostru vom reduce numarul cazurilor de testare la 24.

6. Creeaza cazuri de testare

n1
n2
n3x1c1s1

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

24 cazuri de testare

7. Creeaza date de test

Intrari			
n	**x**	**c**	**s**
-5			
0			
21			
25			
1	a	a	y

1	a	a	n
1	a	b	y
1	a	b	n
5	abcde	a	y
5	abcde	a	n
5	abcde	c	y
5	abcde	c	n
5	abcde	e	y
5	abcde	e	n
5	abcde	f	y
5	abcde	f	n
20	abcdefghijklmnoprstu	a	y
20	abcdefghijklmnoprstu	a	n
20	abcdefghijklmnoprstu	c	y
20	abcdefghijklmnoprstu	c	n
20	abcdefghijklmnoprstu	u	y
20	abcdefghijklmnoprstu	u	n
20	abcdefghijklmnoprstu	z	y
20	abcdefghijklmnoprstu	z	n

Nota: O alta categorie poate fi considerata numarul de aparitii a lui c in sirul x. Aceasta categorie poate fi adaugata celor existente.

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