DOKUMENTIRANJE TESTNIH VZORCEV

Tomaž Dogša
cV&Vs Center za verifikacijo in validacijo sistemov
Fakulteta za elektrotehniko, računalništvo in informatiko
Univerza v Mariboru, Smetanova 17, 2000 Maribor
tdogsa@uni-mb.si

Povzetek

Množica testnih vzorcev je eden izmed najpomembnejših elementov testiranja, saj je v njihovo načrtovanje vloženega ogromno truda. Po količini podatkov so običajno najobsežnejši del dokumentacije. V prispevku bodo opisane lastnosti testnih vzorcev, najpomembnejši atributi, njihova organiziranost in druge posebnosti.

Ključne besede

Testni vzorci, testni primeri, ponovljivost, testiranje, programska oprema.

Abstract

One of the most important elements in testing process are test cases. They are so important because a lot of effort is spent on test case design. In this paper the characteristics of test data will be described together with the most important attributes and organisation of test data, as well as other particularities.

Keywords

Test data, test cases, test-case specification, repeatability, testing, software

1. Uvod

Testiranje je najstarejša in še vedno ena izmed najpogosteje uporabljenih metod preverjanja programske opreme. Celoten proces testiranja je sestavljen iz več posameznih aktivnosti: poganjanje sistema ali pa samo njegove komponente, opazovanje in beleženje rezultatov tega poganjanja ter njihovo vrednotenje glede na določen vidik [IEEE,1990b]. Testni primeri, ki jih pri testiranju potrebujemo, so proizvod obsežnega in zahtevnega procesa, ki zajema analizo specifikacij, obvladovanje raznih testnih strategij in poznavanje objekta, ki ga testiramo. Po količini dokumentacije zavzemajo testni primeri običajno prvo mesto. Testni primer je sestavljen iz več komponent, od katerih sta najbolj pomembna testni vzorec in pričakovano obnašanje oziroma pričakovane izhodne vrednosti.

V prispevku se bomo osredotočili samo na testne vzorce. Ne bomo govorili o načrtovanju testnih vzorcev, ampak predvsem o njihovem opisovanju. Še tako dobro zamišljen testni vzorec postane neuporaben, če ga ne znamo ustrezno dokumentirati. Šele ko osvojimo ustrezen način opisovanja, se lahko posvetimo celotni problematiki načrtovanja. Opisane bodo lastnosti testnih vzorcev, njihova organiziranost in druge posebnosti, ki jih moramo razumeti, če jih želimo specificirati na pravilen način.

Pogosto imamo opravka z velikim številom testnih vzorcev (nekaj tisoč), ki jih shranjujemo v podatkovni bazi. Pri načrtovanju takih baz je potrebno zelo dobro zastaviti entitetno-relacijski model. Ker mnogi avtorji knjig o testiranju [BEIZER,1990], [MYERS,1979], [KANER,1993] posvečajo zelo malo prostora tej problematiki[1], je namen tega prispevka, da zapolni to vrzel.

Opisovanje testnih vzorcev je tesno povezano z načinom, po katerem program sprejema vhodne podatke. V drugem poglavju bomo programe najprej razdelili v tri skupine. Kriterij delitve bo način branja vhodnih podatkov. Osrednji del bo vsebovan v poglavju Struktura testnih vzorcev. Prikazanih bo tudi nekaj krajših zgledov, ki bodo zaradi omejitve prostora vsebovali samo tiste atribute, o katerih bomo razpravljali.

2. Vhodni podatki

Najprej bomo postavili zelo preprosto definicijo vhodnega podatka: vhodni podatki so vsi signali oziroma katerikoli podatki, ki vplivajo na delovanje programa. Zagotovo vsak program potrebuje vhodne podatke. Ker program, ki ne potrebuje nobenega vhodnega podatka, daje vedno enak rezultat, ga v večini primerov

1 *Tudi v standardu ANSI/IEEE 829-1983 (Test documentation, poglavje Test-Case Specification) sta opisu testnih vzorcev namenjena samo dva stavka.*

poženemo samo enkrat. Nas bodo zanimali samo programi, ki sprejemajo vhodne podatke. Vhodne podatke program bere iz različnih virov: tipkovnica, miška, tiskalniki (status tiskalnika), disk, druge vhodno-izhodne enote in iz hitrega pomnilnika (razne nastavitve operacijskega sistema). Pri testiranju nikakor ne smemo pozabiti, da v nekaterih primerih na delovanje programa vplivajo tudi razne nastavitve parametrov operacijskega sistema, vrednosti spremenljivk okolja, vrsta gonilnikov, vrsta računalnika ipd.

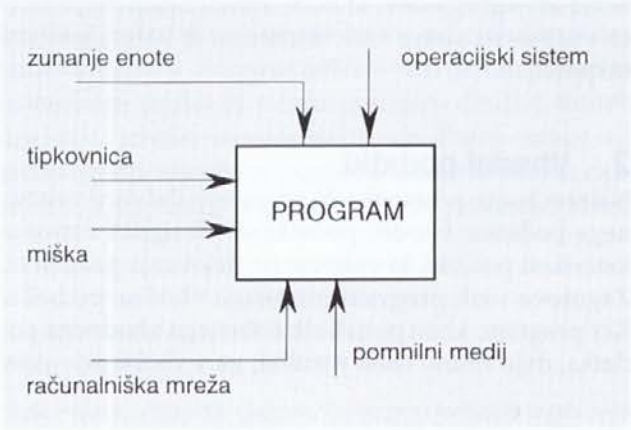
Vsak vhodni podatek ima neko ime in definicijsko območje, iz katerega črpamo konkretne vrednosti. Seznam s temi podatki bomo poimenovali splošen opis vhodnih podatkov. Torej ima entiteta vhodni podatek najmanj štiri atribute: ime, definicijsko območje, medij, na katerem je podatek na razpolago, in konkretna vrednost.

Kljub temu da bomo program obravnavali z vidika črne škatle, je način sprejemanja vhodnih podatkov zelo pomemben. Zato bomo v nadaljevanju postavili preprost klasifikacijski sistem, s katerim bomo lahko klasificirali programe. V literaturi obstaja kar nekaj klasifikacijskih sistemov, ki izhajajo v večini primerov iz implementacijske perspektive. Tukaj nas bodo zanimale predvsem tiste lastnosti, ki jih opazi preverjevalec in ki vplivajo na sistem dokumentiranja testnih vzorcev. Kljub temu da so preverjevalcu v večini primerov implementacijske podrobnosti skrite, lahko iz uporabnikovih navodil sklepa o načinu podajanja vhodnih podatkov. Na klasifikacijski sistem imata največji vpliv način delovanja vhodno-izhodnih enot in razpoložljivost vhodnih podatkov.

2.1. Delovanje vhodno-izhodnih enot

Glede na način delovanja vhodno-izhodnih enot lahko programe razdelimo v naslednje tri skupine:

V prvi skupini so tisti programi, ki čakajo tako dolgo, dokler vhodno-izhodna enota ne dostavi zahtevanega podatka (takšen program vsebuje standardne stavke *read*). Čas, ob katerem posamezni podatek programu dostavimo, nima nobenega pomena. Ker je pomemben samo vrsti red in vsebina podatkov, bomo programe, ki uporabljajo ta način branja, imenovali **časovno neodvisni programi**[2].



Slika 1. Program bere vhodne podatke iz različnih virov

Programe, ki so sposobni prevzeti vhodni podatek kadarkoli znotraj nekega intervala, bomo uvrstili v drugo skupino. Nekateri programski jeziki omogočajo, da se program po nekem času nadaljuje, čeprav nismo dostavili vhodnih podatkov. Na obnašanje programa ne vpliva samo vsebina vhodnih podatkov, ampak tudi **čas**, ob katerem smo jih programu posredovali. Programi iz te skupine so se sposobni odzivati na dogodke (npr. klik z miško). Mnogi programi iz te skupine uporabljajo prekinitveni (interrupt) način branja vhodnih podatkov. Tipični predstavniki so razni gonilniki, operacijski sistemi in objektno orientirane aplikacije. Programe iz te skupine bomo imenovali **časovno odvisni programi.**

V tretji skupini bodo programi, kjer sta prisotna oba načina branja vhodnih podatkov. Ker je obravnava časovno variantnih programov zelo zahtevna, se bomo v tem prispevku omejili predvsem na časovno neodvisne programe

2.2. Razpoložljivost vhodnih podatkov

Na opis testnih vzorcev ima velik pomen tudi **razpoložljivost vhodnih podatkov**. Glede na razpoložljivost vhodnih podatkov delimo programe v tri skupine:

1. **Programi z vedno razpoložljivimi vhodnimi podatki.** Pri tej vrsti programov so vhodni podatki vedno vnaprej pripravljeni. Vhodno-izhodna enota jih lahko, potem ko so pripravljeni za branje, brez kakršnegakoli nadaljnega uporabnikovega posredovanja, kadarkoli prebere. V to skupino uvrščamo programe, ki berejo vhodne podatke iz raznih tračnih enot, diskovnih pogonov, hitrih pomnilnikov (RAM, EPROM, ROM, ..), bralnikov luknjanih kartic, posebnih enote za branje analognih in digitalnih signalov, ki prihajajo iz okolja računalnika ipd. Nekateri starejši operacijski sistemi (paketna obdelava) so delovali samo po tem konceptu. Danes jih pogosto srečamo v raznih računalniško podprtih napravah (npr. mikroprocesorska regulacija motorjev itd.). Tipični predstavnik v domeni osebnih računalnikov je npr. prevajalnik.
2. **Popolnoma interaktivni programi.** To so programi, ki zahtevajo od uporabnika, da po potrebi zagotavlja vhodne podatke - najpogosteje s pomočjo tipkovnice in miške. Programi iz te skupine ne uporabljajo nobenih virov s trajno razpložljivimi podatki. Najpogosteje jih srečamo v raznih računalniško

2 *Ta klasifikacija je v skladu s splošno teorijo sistemov. Glej npr. [SCHWARZ,1965].*

podprtih napravah (npr. računalniške igrice, kalkulatorji itd.). Tudi v okolju Windows bi lahko našli nekaj primerov - to so razni kalkulatorski programi, nekatere igrice itd.

3. **Delno interaktivni programi.** V tretji skupini so programi, ki uporabljajo kombiniran način branja vhodnih podatkov - torej sodelovanje uporabnika in dostopnost virov s trajno razpoložljivimi podatki. Tovrstnih programov je danes največ. Naj naštejemo samo nekaj zgledov: večina aplikacij na današnjih računalnikih (npr. urejevalniki besedil, računalniško podprt sistem rezervacij), mikroprocesorsko krmiljen videorekorder ali televizija itd.

3. Testni vzorec

Testiranje se od drugih preverjevalnih metod razlikuje v tem, da objekt, ki ga preverjamo (v našem primeru je to program), tudi poganjamo. Delovanje programa spoznamo po podatkih, ki jih sporoča, po stanjih, v katerih se nahaja in po načinu prehajanja stanj. Na delovanje pa ne vplivajo samo vhodni podatki, ampak še cela vrsta drugih dejavnikov, kot so npr.: vrsta priključenega tiskalnika, vrsta operacijskega sistema, velikost pomnilnika, razna sporočila in prekinitve itd. Če program poganjamo zaradi odkrivanja prisotnosti napak, potem **niz vseh vhodnih podatkov, ki vplivajo na obnašanje programa,** imenujemo **testni vzorec** (input data, test data, test stimulus). Če program testiramo, potem so testni vzorci sinonim za vhodne podatke.

Število vseh podatkov, ki jih potrebujemo za en test, bomo imenovali **dimenzija testnega vzorca**. Večje je število vhodnih podatkov, bolj kompleksen in zahteven bo testni vzorec. Zato je dimenzija testnega vzorca ena izmed zelo enostavnih metrik, s katerimi merimo **kompleksnost testnega vzorca.** Vsak testni vzorec sproži določeno obnašanje programa in proizvede določene rezultate. S primerjavo med zahtevanim obnašanjem in tistim, ki smo ga opazili pri testiranju, lahko ugotavljamo prisotnost napak[3]. Enako velja za izhodne rezultate, katere primerjamo s pričakovanimi. Če pridružimo testnemu vzorcu še pričakovane izhodne rezultate, dobimo **testni primer**. To je entiteta, ki je rezultat vsakega sistematičnega pristopa k preverjanju.

4. Struktura testnega vzorca

V zahtevah ali specifikacijah smo določili obnašanje programa. Že v teh dveh fazah smo se srečali s problemom opisovanja obnašanja bodočega sistema. Praviloma naj bi bil opis dovolj natančen, da lahko sklepamo, kako je potrebno programu posredovati podatke. Od načina posredovanja je odvisna struktura in način dokumentiranja testnega vzorca. Ne glede na vrsto programa, mora vsak testni vzorec zadostiti naslednjim zahtevam:

1. Testni vzorec mora biti tako razumljivo opisan, da ga lahko uporabi tudi tretja oseba.
2. Testni vzorec mora biti tako natančno opisan, da zagotavlja ponovljivost testa

V nadaljevanju bomo prikazali nekaj tipičnih struktur, ki bodo uporabne le za določeno skupino programov. Izbrali smo dve skupini programov, s katerima se preverjevalci najpogosteje srečujejo. Za vsako vrsto programa bomo določili preprost model, ki bo ponazarjal način izvajanja in krmiljenja programa. Nato bomo določili ustrezno strukturo testnih vzorcev. Ta struktura bo vsebovala samo najpomembnejše atribute, ostalih manj pomembnih (npr. avtor, verzija, datum itd.) ne bomo opisovali. Začeli bomo z najbolj preprostim vhodno-izhodnim modelom, na katerem bomo tudi razložili osnovni koncept. Ta koncept bomo kasneje razširili. Pri opisu modela se bomo naslanjali na metodo, ki se uporablja za opis[4] končnih avtomatov (finite state machine). Če bi program obravnavali popolnoma enako kot digitalne sisteme, pri katerih predstavlja vsaka različna kombinacija bitov v registrih novo stanje, bi imeli opravka z ogromnim številom stanj. To ogromno množico bomo drastično zmanjšali, saj nas bodo zanimala samo tista stanja, v katerih program zahteva vhodne podatke.

4.1. Osnovna struktura testnega vzorca

Glede na način podajanja vhodnih podatkov je najbolj enostaven takšen program, ki je časovno neodvisen in ki prebere vse vhodne podatke iz datoteke in pri tem uporablja programiran način branja. V to skupino spadajo aplikativni programi, pisani za prve večje računalniške sisteme. V fizičnem pogledu so bili podatki bodisi na karticah, trakovih ali pa na disku. Čas, ob

Slika 2. Najbolj preprost model programa

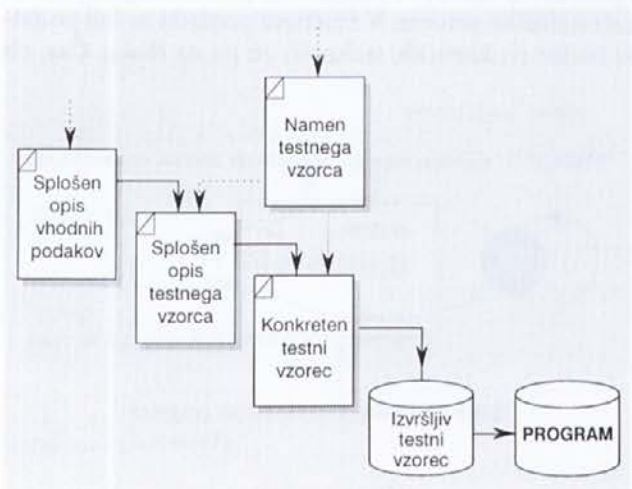
3 *Več o modelu preverjanja je v [DOGŠA,1994].*
4 *Zelo uporabna predvsem v digitalnih sistemih.*

katerem smo posamezni podatek dostavili programu, ni imel nobenega pomena. Pomemben je bil samo vrstni red in vsebina podatkov. Tem programom pogosto pravimo paketni programi. Ko pripravimo vse vhodne podatke, program poženemo. Po končani obdelavi se program konča. Takšen program pozna samo dve stanji (glej sliko 2): začetno stanje in stanje, v katerem program procesira vhodne podatke.

Začnimo z najbolj preprostim programom iz te skupine, ki naj ima več vhodnih stavkov. Vsak vhodni stavek pomeni branje enega ali več podatkov, zapisanih na vhodnem mediju. Predpostavimo, da program za določen test potrebuje n podatkov. Če označimo vsak podatek na mediju z x_i, potem lahko testni vzorec formalno opišem z vektorjem n-te dimenzije: $x = (x_1, x_2, ...x_n)$. Mnogi zato testnim vzorcem pravijo **testni vektorji.** Ker imamo vedno opravka z več testnimi vzorci, je smiselno, da jih prikazujemo kot polje:

$x(1) = (x_1, x_2, ...x_n)$
$x(2) = (x_1, x_2, ...x_n)$
:

Komponente testnega vzorca so lahko: numerični podatki, besedila, grafični objekti, datoteke itd. Zapis *x(3) = (3,4,Miha)* nam ne pove dosti, če ne poznamo splošnega opisa vhodnih podatkov. Šele takrat lahko vemo, kateremu vhodnemu podatku bo prirejena vrednost 3. Opis testnega vzorca torej tvorita dva vektorja: **splošni opis testnega vzorca** in **konkretni testni vzorec.** V prvem je splošni opis posameznih komponent testnega vzorca (ime podatka, položaj na mediju ipd.), v drugem pa konkretne vrednosti. Takoj vidimo, da imata oba enako dimenzijo, in da lahko uporabimo en splošni opis za več konkretnih vektorjev. Ker na delovanje programa vpliva tudi začetno stanje, bomo njegov opis pridružili testnemu vzorcu. Dva testna vzorca sta enaka, če imata enak splošni opis, enako začetno stanje ter enake vrednosti vhodnih podatkov.



Slika 3. Razvojne faze testnega vzorca

Iz splošnega vhodnega opisa vzamemo samo tiste vhodne podatke, ki jih potrebujemo za splošni opis testnega vzorca. Izbor je odvisen od tega, katere podatke bo program pri svojem izvajanju potreboval. Definicija splošnega opisa vhodnih podatkov je prvi korak pri načrtovanju testnih vzorcev. Glede na izbrano testno strategijo nato določimo enega ali več konkretnih testnih vzorcev.

Zgled 1: Testiramo program, ki poišče rešitev kvadratne enačbe: $ax^2 + bx + c = 0$. Program izpiše rešitev na ekran ali pa na tiskalnik. Iz specifikacij ugotovimo, da bo program prebral en zapis na datoteki INPUT.DAT

Takoj vidimo, da bo testni vzorec četrte dimenzije. Splošen opis testnega vzorca bi lahko zgledal takole[5]:

```
S(i) = <začetno stanje>
x(i) = (a, b, c, način izpisa)
a, b, c : množica realnih števil
način izpisa: 1: tiskalnik, 2: ekran
Položaj na mediju (formatno določilo):
                    nnn.n   nnn.n   nnn.n   nnn

Datoteka: INPUT_i.DAT
```

Zapišemo lahko tudi nekaj konkretnih testnih vzorcev:

x(1) = (23.4, 10, 45, 1)	S(1) = DOS prompt in klic programa
x(2) = (4, 0, -45, 1)	S(1) = DOS prompt in klic programa
x(3) = (100, -10, 45, 2)	S(1) = DOS prompt in klic programa

S pomočjo formatnega določila lahko tvorimo tri datoteke INPUT_1.DAT, INPUT_2.DAT in INPUT_3.DAT, ki predstavljajo izvršljivo obliko testnega vzorca[6].

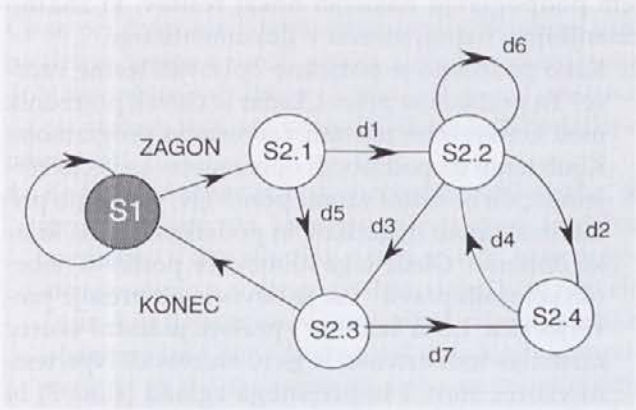
Le v redkih primerih program prebere vhodne podatke iz vhodno-izhodne enote samo enkrat (to se zgodi takrat, ko imamo v programu samo en stavek *read*, ki se ne nahaja znotraj nobene zanke). V večini primerov zahtevamo prenos podatkov iz vhodno-izhodne enote večkrat. To pomeni, da ima večina programov zelo kompleksne testne vzorce. Testni vzorec je običajno kar celotna datoteka z vsemi podatki. Zgled za take vrste program je npr. program za izračun plač. Avtomatizacija testiranja (samo poganjanje programa) je za programe iz te skupine relativno enostavna. Tudi takrat, ko je program v uporabi, imamo pri vsaki odpovedi programa na razpolago tudi stimulus, s katerim lahko v večini primerov dosežemo ponovljivost odpovedi. Ker so vsi vhodni podatki zapisani na datotekah, nimamo nobenih posebnih težav z arhiviranjem testnih vzorcev.

5 *Splošen opis je odvisen od implementacije (formatnega določila). Ker je način opisa vhodnih podatkov na datotekah zelo dobro poznan, se ne bomo spuščali v podrobnosti glede formatnega določila.*

6 *Pojem "izvršljiva oblika testnega vzorca" bo podrobneje obravnavan v nadaljevanju besedila.*

4.2. Struktura testnega vzorca za delno interaktivne programe

Delno interaktivni programi dobivajo vhodne podatke iz vhodnih enot, s katerimi upravlja uporabnik in iz drugih vhodnih enot (npr. disketnik). Najpogosteje se uporabljata tipkovnica in miška. Komunikacija med programom in uporabnikom je izvedena s pomočjo preprostih vnosov, menujev in dialogov. Poiščimo preprost model tovrstnih programov. V modelu iz slike 2 lahko stanje S2 opišemo z množico podrobnejših stanj, kjer vsako stanje ustreza enemu dialogu. Vsako stanje, če je potrebno, lahko zopet prikažemo kot množico podrobnejših stanj.



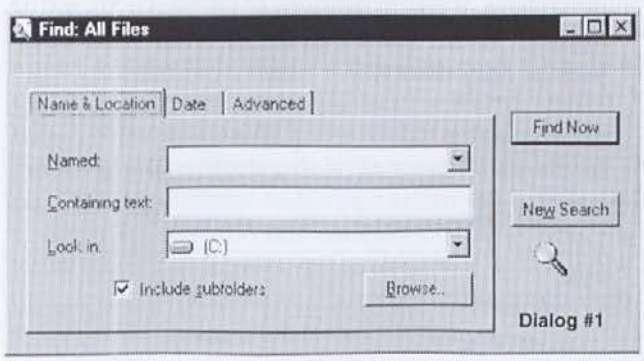
Slika 4 Model sprejemanja vhodnih podatkov nekega programa, ki ima 4 dialoge

Ko program sprejme podatek, se lahko preseli v drugo stanje (dialog) ali pa ostane v istem. Prehod je definiran z določenim pogojem, ki ga bomo označili z d_j. Če je zadoščeno pogoju d_j, se izvedejo predvidene funkcije in prehod v ustrezno stanje. Program, ki je predstavljen z modelom na sliki 4, ima 4 dialoge. Iz modela je razvidno, koliko dialogov imamo in kakšna je njihova dostopnost. Npr. iz dialoga S2.1 lahko s sprožitvijo dogodka d1 pridemo v dialog S2.2

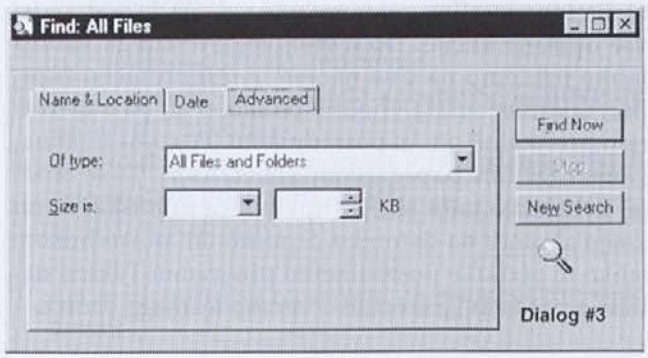
Rezultati procesiranja podatkov se kažejo v načinu, kako se izvede prehod v drugo ali vrnitev v isto stanje in v podatkih, ki se zapišejo na določeno izhodno enoto oziroma v pomnilnik. Redosled prehodov bomo imenovali izvajalna pot ali test. Začetek te poti je začetno stanje. Stanje, v katerem se test konča, bomo poimenovali končno stanje. Pri interaktivnih programih lahko vhodne podatke razdelimo v procesne in krmilne podatke. Procesni so tisti, ki jih običajno vpišemo v določena polja, krmilni pa predstavljajo npr. aktiviranje raznih gumbov. Ločitev na procesne in krmilne ni vedno enostavna, saj lahko isti podatek uporabimo kot krmilnega in procesnega hkrati. Ker mnogokrat ta delitev niti ni pomembna, jo pri dokumentiranju testnega vzorca pogosto izpuščamo.

Pri interaktivnih programih so komponente testnega vzorca akcije in vrednosti, ki jih moramo vnesti. Pri časovno variantnih je potrebno dodati še dogodke, kar dokumentiranje zelo zaplete. Pogoj, da sta dva testna vzorca enaka, ni samo v enakih vrednostih komponent testnega vektorja. Dva testna vzorca sta enaka, če imata enako začetno stanje in ob vsakem času enake vrednosti komponent. Za opis imamo na voljo razne (zahtevne!) mehanizme opisovanja paralelnih sistemov (glej npr. [TILBORG,1991]).

V večini interaktivnih aplikacij so dogodki relativno enostavni (kliki z miško ali pa pritisk na kakšno posebno tipko - npr. <ENTER>). Tudi tukaj moramo najprej definirati splošni opis testnega vzorca. Kadar interaktivno vnašamo vhodne podatke, potem atribut "položaj na mediju" zamenjamo z atributom "položaj v dialogu". Ime vhodnega podatka označimo z besedo, ki se nahaja zraven vpisnega mesta. Podobno velja za dogodke. Zaradi večje ločljivosti jih dodatno opremimo z ostrimi oklepaji. Za zgled poglejmo preprost program z imenom search.exe, ki poišče datoteko (glej sliko 5). Program se nahaja na imeniku c:\test. Vse dialoge najprej oštevilčimo. V "Dialogu #1" lahko sprožimo 7 dogodkov in vpišemo 4 podatke. K testnemi vzorcu je potrebno prišteti še začetno stanje, ki predstavlja inštaliran operacijski sistem 98 in datoteke, ki ležijo na istem imeniku kot program (npr.c:\test). Ponovljivost testa bo zagotovljena, če bomo na imeniku c:\test imeli enake datoteke, če bomo imeli enak operacijski sistem in če bomo izvedli enake akcije in vpisali enake podatke.



Slika 5 Program, ki išče datoteke. Z oštevilčenjem dialogov lahko zelo enostavno povečamo testabilnost.



Slika 6 Dogodek **<Advanced>** v Dialogu #1 sproži prikaz Dialoga #3

Testni vzorec 1
Začetno stanje: Windows95, imenik c:\test

akcija/ime	vhodne spremenljivke	vrednost
<Start>	pritisni	
<Run>	pritisni	
vpiši		c:\test\search.exe
<ENTER>		

Slika 7 Zgled za opis testnega vzorca za interaktivne programe (združena sta splošni in konkretni opis)

Za opise testnih vzorcev lahko uporabimo posebne formularje ali pa jih direktno vnašamo v posebno podatkovno bazo (Test case manager). Slaba stran testiranja interaktivnih programov je v tem, da moramo ročno vnašati podatke. Tega problema pri paketnih programih ni bilo.

4.3. Izvršljiva vrsta testnega vzorca

Razvoj programske opreme se začne z uporabnikovimi zahtevami. Zelo poenostavljeno gledano poteka razvoj nekako v teh korakih: najprej se zahteve pretvorijo v sistemske specifikacije, te v izvorno kodo, katero s prevajalnikom prevedemo in dobimo izvršljiv program. Podobnim korakom sledi načrtovanje in uporaba testnih primerov (glej sliko 3). Začetek predstavlja namen testnega vzorca. Le-ta se mora po več korakih pretvoriti v takšno obliko, da lahko program poženemo. To pomeni, da ko program poganjamo, mu posredujemo prav vse vhodne podatke, ki jih za konkreten test potrebuje in ne samo tiste, ki so za dosego določenega namena potrebni. Za ilustracijo bomo uporabili kar prejšnji zgled. Recimo, da je namen testnega vzorca preveriti, ali bo program pravilno deloval, če bi uporabili eksponentno obliko zapisa števil (npr. 1.23E-3). Glede na to zahtevo moramo najprej izbrati konkretna števila, izbrati način izpisa (tiskalnik ali ekran) in z urejevalnikom besedil vpisati testni vzorec na datoteko. Ta datoteka je testni vzorec v **izvršljivi obliki**, saj jo program lahko sam prebere. Testni vzorec s slike 7 ni direktno izvršljiv, saj potrebuje nekoga, ki bo izvedel vse opisane akcije. Do izvršljivega testnega vzorca lahko pridemo na dva načina: z intepretacijo (npr. preverjevalec vnaša vhodne podatke) ali pa z ustreznim prevajalnikom in posrednikom. Pogosto si pomagamo z raznimi papagajskimi generatorji[7]. To so orodja, ki beležijo naše akcije (tipkovnica in miška) in jih znajo shraniti na datoteko. S posebnim posrednikom lahko te podatke posredujemo programu. Takšna datoteka predstavlja izvršljivo verzijo testnega vzorca.

7 *Capture playback tool.*

Le če imamo na razpolago izvršljivo vrsto testnega vzorca, je možno avtomatsko poganjanje programa, ki znatno pospeši testiranje. Večina operacijskih sistemov (npr. DOS, UNIX, VAX VMS) omogoča preusmerjanje vhodnih podatkov. Na ta način lahko pretvorimo interaktivne programe v programe, ki imajo vedno razpoložljive vhodne podatke.

5. Racionalizacija opisa testnih vzorcev

Zaradi čedalje večje kompleksnosti programov narašča tudi kompleksnost in število testnih vzorcev. Ker je v dokumentiranje vloženega ogromno truda, bomo v tem podpoglavju nakazali nekaj rešitev, ki znatno zmanjšujejo napor, vložen v dokumentiranje.

1. Kako podrobno je potrebno opisovati testne vzorce? Ta problem se pojavi, kadar je človek posrednik med konkretnim testnim vzorcem in programom. Kljub temu da podrobejši opis zagotavlja večjo verjetnost, da bo testni vzorec ponovljiv, v mnogih primerih skupino dogodkov in podatkov kar na kratko opišemo. Glede tega ni mogoče postaviti nobenih splošnih pravil - vse je odvisno od presoje preverjevalca. Le-ta se mora vprašati: je testni vzorec razumljiv tudi tistemu, ki ga ni načrtoval? Npr. testni vzorec štev. 1 iz prejšnega zgleda (slika 7) bi lahko bil krajši (glej sliko 8). Pri krajši različici ni zapisano, kako smo program pognali.

Testni vzorec 2
Začetno stanje: Windows95, imenik c:\test
Poženemo c:\test\search.exe

Slika 8 Zelo skopo opisan testni vzorec s slike 7

2. Ker se z večanjem izvajalne poti veča tudi dimenzija testnega vzorca, je smiselno, da določene segmente izvajalne poti zamenjamo z ustreznim začetnim stanjem. Namesto da bi vsakič opisovali celotno pot, se raje sklicujemo na končno stanje enega izmed prejšnjih testov (glej sliko 9).
3. V večini primerov se določeni deli testnih vzorcev ponavljajo. V takem primeru se lahko sklicujemo na določen testni vzorec in definiramo samo spremembe (glej sliko 10),

Testni vzorec: 2
Začetno stanje: Dialog #1 (končno stanje po testu 1)

akcija/ime	vhodne spremenljivke	vrednost
Vpiši		
	Named:	*.txt
	Containg text:	Maribor
<Find Now>		

Slika 9 Skrajšan opis testnega vzorca za interaktivne programe

Testni vzorec 3
Začetno stanje: Dialog #1 (končno stanje po testu štev. 1)
Odvisnost od testnih vzorcev : 1, 2
Vpliva na testne vzorce : 5,6,8,10

akcija/ime vhodne spremenljivke	vrednost
Isto kot testni vzorec štev. 2	
Razlike:	
Named	c*2.txt

Slika 10 V vse nadaljne testne vzorce, ki so povezani s prvim, vpisujemo samo spremembe

Če se pri določenem testnem vzorcu sklicujemo tudi na druge testne vzorce, potem njihova organizacija dobi hierarhično strukturo. V tem primeru je smotrno, da pri vsakem testnem vzorcu vodimo tudi podatke o odvisnosti.

4. Kadar so dimenzije testnih vzorcev zelo velike, je priporočljivo, da komponente testnega vzorca razdelimo v dve skupini: v prvi so tisti, ki se pogosto spreminjajo, v drugo pa damo take, ki se redkeje (npr. konfiguracija računalnika). Slednje lahko obravnavamo kot združen del določenih komponent, ki ga v dokumentaciji samo na enem mestu opišemo in se kasneje na njega sklicujemo. Ta datoteka je seveda sestavni del testnega vzorca.

6. Pomen dimenzije testnega vzorca

Dimenzijo testnega vzorca in njegove komponente določimo z analizo specifikacij, analizo navodil za uporabo ali z analizo izvorne kode. V mnogih primerih je zelo težko določiti pravilno dimenzijo oziroma vse komponente, ki so potrebne za tvorbo testnega vzorca. Na izvajanje programa lahko imajo vpliv tudi razni gonilniki, velikost pomnilnika, vrsta operacijskega sistema (npr. Windows 3.1 ali Windows 95). Dimenzija je odvisna tudi od izvajalne poti oziroma od vrednosti vhodnih podatkov. Pri uporabi slabo dokumentiranih knjižnic in drugih programskih komponent se lahko zgodi, da program bere nekatere sistemske podatke (npr. maksimalno dovoljeno število odprtih datotek), vendar to ni nikjer dokumentirano. Če tega ne vemo, tega ne moremo vključiti v testni vzorec. V takem primeru imamo kasneje skoraj vedno težave s ponovljivostjo. Ali smo določili pravilno dimenzijo ali ne, lahko ugotovimo tudi s ponavljanjem istih testnih vzorcev. Pri pravilno določeni dimenziji, se mora program pri ponovljenem testnem vzorcu vsakič enako obnašati. Testni vzorec s pravilno dimenzijo bomo poimenovali **kompleten testni vzorec**.

7. Ponovljivost testiranja

Rezultat testiranja je v večini primerov seznam nepravilnosti[8], ki smo jih opazili. Ta seznam potrebujejo popravljalci programa, skupina za zagotavljanje kakovosti in pa vodstvene strukture. Popravljalci najprej ponovijo test in nato s pomočjo analize izvorne kode ter izkušenj, odstranijo napake, za katere predvidevajo, da so povzročile opisane nepravilnosti. Z **istim** testnim vzorcem nato preverijo, ali so odpravili neustrezno obnašanje programa. Vidimo, da popravljalci isti test ponovijo najmanj **dvakrat**.

Vodstvene strukture se na podlagi opaženih nepravilnostih odločajo o nadaljnem poteku razvoja in preverjanja. V skrajnem primeru se lahko celo odločijo o opustitvi projekta ali pa o tožbi zaradi prenizke kakovosti produkta. V mnogih primerih prizadeta stranka želi obremenjujoče teste tudi sama ponoviti. Skratka preverjevalci lahko s svojimi ugotovitvami sprožijo zelo pomembne odločitve. Če se izkaže, da se nepravilnosti pri ponovitvi testov niso pojavile, potem preverjevalci zgubijo zaupanje glede svoje strokovnosti. **Ponovljivost testa je torej ena izmed temeljnih lastnosti, ki bi jo moral imeti vsak test**. V nekaterih primerih je zelo težko zagotavljati ponovljivost.To še posebej velja za večopravilne sisteme, in za programe, ki so vezani na dogajanje na Internetu. Kljub temu da test ni ponovljiv, še to ne pomeni, da nima nobene vrednosti. Neponovljivi test lahko kaže na nestrokovnost preverjevalcev, ali na potencialno neustreznost, ki se bo prej ali slej pojavila. V nekaterih primerih je neponovljivost testa posledica nepravilno inicializiranih spremenljivk. Kolikšna je vrednost neinicializirane spremenljivke, je odvisno od prevajalnika. Nekateri jih postavijo na nič, pri drugih pa lahko zavzamejo naključne vrednosti. Prevajalniki večinoma izpišejo ustrezna opozorila, katere pa programerji večinoma ignorirajo.

8. Idealne lastnosti testnega vzorca

Idealni testni vzorec ima naslednje lastnosti:

1. njegovo generiranje je zelo poceni - zelo enostavno ga generiramo
2. določitev pričakovanega obnašanja oziroma izhodnih vrednosti je zelo enostavno
3. testni vzorec zahteva zelo enostavno odpovedno kriterijsko funkcijo
4. v trenutku pripravi program v tako stanje, v katerem se kaže prisotnost napak
5. je vedno ponovljiv
6. možno ga je enostavno opisati in shraniti
7. ker je hierarhično zasnovan, ga je enostavno vzdrževati

8 *Priloženi morajo biti tudi testni vzorci.*

8. pri uporabi se manifestira prisotnost prav vseh napak
9. pretvorba v izvršilno obliko testnega vzorca je zelo enostavna.

9. Sklep

Opisali smo samo nekatere najpomembnejše značilnosti testnega vzorca, ki je eden izmed najpomembnejših sestavnih delov testnega primera. Na kratko je bil prikazan tudi del življenskega ciklusa testnega vzorca. Testni vzorci se pojavljajo kot splošni, konkretni in izvršilni. Za avtomatsko poganjanje programa potrebujemo prevajalnik, ki pretvori splošni opis v izvršljivo vrsto testnega vzorca in ustrezen posredovalnik. Ne smemo pozabiti, da za testiranje potrebujemo **testni primer**, ki je še bolj kompleksen kot testni vzorec, saj vsebuje še dodatne atribute, od katerih bomo omenili samo pričakovano obnašanje in odpovedno kriterijsko funkcijo.

Ker naj bi vsak testni vzorec zadostil kriterijem **ponovljivosti**, jih je potrebno ustrezno dokumentirati in nato glede na spremembe v novih verzijah programa, tudi vzdrževati. Tudi testni vzorci so podvrženi podobnemu življenskemu ciklusu, kot velja za programsko opremo.

Dokumentiranje testnih vzorcev zahteva veliko truda, ki se povrne šele, ko jih ponovno uporabimo. Ker je izvršljive testne vzorce v večini primerov nemogoče opremiti z dodatnimi komentarji, to storimo v splošnem testnem vzorcu. Ker je način dokumentiranja tesno povezan z vrsto programa, je prva naloga preverjevalca, da določi, v katero skupino spada program, ki ga testira. Šele nato izbere ustrezen način dokumentiranja.

10. Literatura

[BEIZER,1990] B. Beizer:
"Software Testing Techniques", Van Nostrand Reinhold, New York,1990, 2. izdaja.

[DOGŠA,1994] T. Dogša:
"Verifikacija in validacija programske opreme", Tehniška fakulteta, Maribor, 1993.

[IEEE,1990b] "IEEE Standard Glossary of Software Engineering Terminology", IEEE Std. 610.12-1990, Revision and redesignation of IEEE Std. 792-1983, The Institute of Electrical and Electronics Engineers, USA, 1990.

[KANER,1993] Cem Kaner, Jack Falk, Hung Quoc Nguyen:
"Testing Computer Software", Van Nostrand Reinhold, 1993.

[MYERS ,1979] J. G. Myers:
"The Art of Software Testing", John Wiley and Sons, Inc., New York, 1979.

[SCHWARZ,1965] R. J. Schwarz, B. Friedland:
"Linear systems", McGraw-Hill,1965

[TILBORG,1991] "Foundations of real-time computing: Formal Specifications and Methods", zbornik, urednika A. M. Tilborg, G. M. Koob, 1991, Kluwer Academic Publisher

◆

Avtor je docent na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru, kjer predava na dodiplomski in podiplomski stopnji in vodi Center za verifikacijo in validacijo sistemov. Na raziskovalnem področju se ukvarja predvsem z V&V tehnologijo oziroma testirnimi orodji.

◆