Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Igor Rožanc

Uvod v testiranje

Študijsko gradivo za interno uporabo pri predmetu Testiranje in kakovost (TiK)

Ljubljana, 2017/18

Kazalo

Izhodišča 1

Testiranje (PO) postaja vse pomembnejše:

veliko * => več stroškov/težav zaradi napak

•= sistemi, uporabnikov, zahtevnost, roki, konkurenca, agilnost,spletna dostopnost, pričakovanja, varnostne zahteve, polomi v preteklosti ..

Industrija nujno potrebuje več testiranja:

–varnostno zahtevni sistemi,

–večje aplikacije,

–vgrajeni sistemi,

–spletni programi,

–odprtokodni programi

Napoved 1: najmanj polovica časa razvoja je (bo) testiranje

Napoved 2: strošek testiranja bo vedno višji

Izhodišča 2

- MOTIVACIJA: obvladovanje
- STROŠEK: pravočasno testiranje : (pre)pozno testiranje
- NAMEN : dokumentirani cilji, kdaj dovolj...
- IZVEDBA: pravočasna zasnova testa (ob zahtevi)
- NAČIN: testne ekipe, znanstvena podlaga
- PRAKSA: vloga testiranja, netestiranje

Šele ko se bo učinkovito uporabljalo v praksi, bo zares dobilo svoje mesto

To je naša misija!

Definicija testiranja

a) Testiranje PO je praktična inženirska aktivnost, ki je ključna za izdelavo visoko kakovostnih (programskih) izdelkov
(P. Ammann, 2008)

b) Testiranje PO je faza v razvoju programske opreme, ki služi učinkovitemu izločanju napak iz programske kode.

c) Testiranje tipično povezujemo s:

- **programskimi izdelki** (diagrami, programsko kodo, dokumentacijo) ali
- **delom razvojnega procesa** (test zahtev, načrta ali kode)

Definicija testiranja

d) Testiranje PO daje naročnikom odgovor o kakovosti izdelka ali storitve testiranja in kot taka daje objektiven vpogled v tveganja uporabe le–te.

Tipične prakse testiranja slonijo na izvajanju programov s ciljem odkrivanja napak.

Testiranje je lahko validacija ali verifikacija izdelka za:

- doseganje poslovnih in tehničnih ciljev iz zahtev,
- delovanje po pričakovanjih in
- preverjanje pričakovanih lastnosti.

Testiranje se lahko izvaja kadarkoli v procesu razvoja (z ustreznim pristopom), a je tradicionalno faza za kodiranjem.

Novejši pristopi določajo testiranje kot aktivnost vseh razvijalcev, tradicionalni določajo posebno skupino testerjev za to. (Wikipedia, 2018)

Definicija testiranja

5

e) Testiranje je dokazovanje pravilnosti programa.

Problem:

Teoretično se (običajno) ne da dokazati pravilnosti delovanja programa.

Posledično je tudi v praksi nemogoče pokazati pravilnost.

Vzrok:

Testiranje zelo raznolikih naprav z različnimi nameni vnaša zmedo

Obstaja veliko (=preveč) programskih poti.

Treba bi bilo testirati vse možne vhodne podatke, kar (običajno) ni mogoče.

In še zlasti: nemogoče je testirati vse programske poti za vse kombinacije vhodnih podatkov.

Aktivnosti testiranja

6

Upravljanje testiranja

Načrtovanje testiranja:

- na podlagi kriterijev
- na podlagi domene

Avtomatizacija testiranja

Izvajanje testov

Analiza rezultatov testiranja

Dokumentiranje testiranja

Vzdrževanje testiranja

Izvajalci testiranja

Testno ekipo sestavljajo testni vodja in testni inženirji

Vodja testiranja:

- upravlja in nadzira izvedbo testiranja
- se dogovarja in poroča vodstvu

Testni inženirji izvajajo raznovrstne testne aktivnosti:

- načrtujejo testiranje
- generirajo testne podatke
- avtomatizirajo testiranje
- poganjajo testne skripte
- zbirajo ali analizirajo testne rezultate ter o njih poroča vodstvu
- vzdržujejo in dokumentirajo testiranje

Različna testna opravila zahtevajo različne spretnosti, znanja, izobrazbo in pripravo: smiselni so različni izvajalci

Izvajalci testiranja – potrebna znanja

Vodenje testiranja:

- Znanja s področja upravljanja, testiranja, komunikacije, psihologije

Načrtovanje testiranja na podlagi kriterijev:

- znanje matematike, programiranja, testiranja
- zahteva velik intelektualni napor
- zadostuje malo takih načrtovalcev

Načrtovanje testiranja na podlagi domene:

- zahteva dobro poznavanje domene, testiranja, UV – brez računalniških znanj
- vseeno intelektualno zahtevno
- nujno vsaj en načrtovalec

Avtomatizacija testiranja:

- samo programerska znanja
- sicer nezahtevno, ni zanimivo za netehnike

Izvajalci testiranja – potrebna znanja

Izvajanje testov:

- le osnovna programerska znanja, natančnost
- nezahtevno, veliko ročnega dela, dolgočasno za marsikoga

Analiza rezultatov testiranja:

- zahteva dobro poznavanje domene, testiranja, okolja
- zelo intelektualno zahtevno, a ni za računalničarje

Dokumentiranje testiranja:

- konkretno: specifična programerska znanja, nezahtevno
- vsi sodelujejo, odgovor na ZAKAJ

Vzdrževanje testiranja:

- znanje matematike, programiranja, testiranja
- zelo zahtevno, lahko navezava na načrtovalca

KLJUČNO: prave izvajalce na prava mesta

Testiranje in življenski cikel

Konceptualni model za življenjski cikel programske opreme (SDLC):

- Analiza zahtev in specifikacija sistema
- Načrtovanje sistema in komponent
- Implementacija
- Integracija sistema
- Prenos sistema v ciljno okolje
- Vzdrževanje

Različni modeli (slapovni, iterativni, inkrementalni, spiralni model) in metodologije upoštevajo te faze na različne načine in v različnih zaporedjih.

Testiranje lahko implementiramo v vsaki fazi življenjskega cikla.

Dobra praksa: teste načrtujemo in pripravimo med vsako fazo, četudi se bodo izvajali predvsem v fazi implementacije.

Testiranje in življenski cikel

Testiranje ima različen namen v različnih fazah = iščemo različne vrste napak

Analiza zahtev in specifikacija sistema

• Glavni izdelek: zahteve naročnika.

• Preverimo, če so zahteve pravilne, izvedljive ter celovite.

Načrtovanje sistema in komponent

• Glavni izdelek: načrt sistema, sestava posameznih komponent.

• Preverjamo usklajenost načrta z zahtevami, izvedljivost posameznih komponent, možnost integracije komponent.

Implementacija

• Glavni izdelek: programska koda.

• Cilj testiranja v tej fazi je uspešno testiranje enot in modulov.

Testiranje in življenski cikel

Integracija sistema

• Cilj testiranja je preveriti sestavo sistema, komunikacijo med posameznimi deli.

Prenos sistema v ciljno okolje

• V tej fazi izvedemo sistemsko ter sprejemno testiranje.

Vzdrževanje

• Vključuje odpravo napak, ki se pokažejo med uporabo programske opreme.

• Ob spremembah izvajamo regresijsko testiranje.

Klasične ravni testiranja glede na aktivnosti razvoja

13

Vrsta testiranja	Nivo	Faza
1) Sprejemno testiranje	zahteve	analiza zahtev
2) Sistemsko testiranje	načrt sistema	načrtovanje
3) Integracijsko testiranje	sestava podsistema	načrtovanje
4) Testiranje modilov	modul	kodiranje
5) Testiranje enot	enota	kodiranje

Vrste testiranja

14

5) Testiranje enot (unit testing)

- najnižji nivo testiranja (najmanjše enote - v Javi metode/objekti)
- vezan na specifikacijo posameznih enot
- enote testiramo v izolaciji
- testiramo z vnaprej pripravljenimi vhodnimi podatki
- dobljene rezultate preverjamo s predvidenimi rezultati
- izvaja hkrati z razvojem vsak razvijalec zase
- sam napiše teste enot, ki jih razvija
- dosežemo pravilno samostojno delovanje enot ne glede na kasnejšo uporabo
- dve vrsti testov: potrditveni (enota deluje) in nepričakovani vhodi
- obvezna podlaga za efektivno testiranje na ostalih nivojih

Vrste testiranja

4) Testiranje modulov (module testing)

- test je vezan na specifikacijo modulov
- modul sestavlja več enot (recimo razred v Javi)
- običajno nivo podatkovne strukture
- testiramo usklajenost enot v modulu, popolnost funkcionalnosti
- običajno izvajajo razvijalci sami
- temeljijo na uspešnem testiranju enot
- smiselna nadgradnja prejšnjega koraka, izvaja se podobno

Vrste testiranja

3) Integracijsko testiranje (integration testing)

- test vezan na sestavo enot in modulov v celoto
- testiranje komunikacije med moduli
- iskanje nekompatibilnosti med vmesniki
- iskanje nepričakovanega delovanja med njihovo interakcijo
- osnova je uspešno testiranje enot (in modulov)
- zahteva sodelovanje razvijalcev
- lahko izvaja testna ekipa ali ne

Vrste testiranja

2) Sistemsko testiranje (system testing)

- preveri, če sistem kot celota deluje
- delovanje sistema definirano v načrtu
- išče napake v specifikaciji in načrtovanju programske opreme
- cilj predvsem identifikacija napak
- predpogoj je uspešno izvedeno integracijsko testiranje
- ponavadi ga izvajajo ločene testne ekipe (neodvisnost)
- običajno več delov testiranja ločeno: funkcionalni, nefunkcionalni del
- ponavadi veliko vrst (nefunkcionalnih) testiranj z različnimi cilji: stresno, obremenitveno, varnostno, ...

Vrste testiranja

1) Sprejemno testiranje (acceptance testing)

- ključno: ali programska oprema zadostuje zahtevam naročnika
- testiranje izvaja (ali vsaj opredeli) sam naročnik (ali nekdo v njegovem imenu) , laho izvaja tudi testna ekipa
- lahko zelo obsežno, odvisno od zahtev
- obstaja lahko več vrst sprejemnih testov: uporabnost, namestitev
- lahko na lokaciji ali ne (namestitveno testiranje)
- sem spada tudi regresijsko testiranje, ki ga izvajamo ob spremembah programske opreme ali odpravi napak

Dodatno: usposabljanje uporabnikov in dokumentacija (ob namestitvi)

Posebne oblike sistemskega testiranja

Testiranje funkcionalnosti

• Ali programska oprema izpolnjuje vse funkcionalne zahteve (KAJ)

• Orodja: HP Quick Test Professional, AppPerfect, DejaGnu.

Testiranje performans

• Ali programska oprema dosega performančne zahteve (KAKO)

• Orodja: jMeter, sistemska orodja za spremljanje performans

Testiranje uporabnosti

• Ali je aplikacija enostavna, učinkovita, intuitivna za uporabo

• Orodja: Chalkmark, Google web site optimizer, ClickHeat.

Posebne oblike sistemskega testiranja

Testiranje grafičnih uporabniških vmesnikov

• Ali je uporabniški vmesnik estetsko,nedvoumno in enostavno zasnovan

• Orodja: Clubic Test, Selenium, Watir.

Testiranje kompatibilnosti

• Ali je aplikacija kompatibilna z drugimi orodji programerja

• Orodja: Browser photo, Browsershots, Netrenderer...

Varnostno testiranje

• Ali je aplikacija varna pred vdori in izgubo podatkov

• Orodja: Netsparker, Websecurify, Webscarab

Obremenitveno (stresno) testiranje

- Preverjamo mejne vhodne vrednosti (napačni vhodi, preveč zahtev …)

Posebne oblike sistemskega testiranja

Količinsko testiranje

- Ali program ustrezno prenese predvidene količine podatkov

Konfiguracijsko testiranje

- Analiza vseh predvidenih konfiguracij strojne in programske opreme

Časovni test

- Test ustreznosti odzivnih časov

Okoljski test

- Delovanje v okolju namestitve

Test kakovosti

- Ustreza specifičnim zahtevam kakovosti

Posebne oblike sistemskega testiranja

Test obnovitve

- Ali se sistem ustrezno ponovno vzpostavi po nepričakovani zaustavitvi (izguba podatkov, odziv)

Test vzdrževanja

- Ali je sistem prilagojen za vzdrževanje, orodja

Test dokumentacije

- Ali dokumentacija ustreza predpisanim zahtevam

Test dostopnosti

- Je sistem dostopen v skladu s specifičnimi zahtevami (WCAG)

Test obravnave izjem

- Ali ustezno obravnava izjeme

Starejše delitve testiranja

Metoda bele škatle (strukturno testiranje)

- Testi so definirani na podlagi notranje strukture aplikacije
- Izvaja se s pregledom in analizo kode programa (notranje strukture).
- Najpogosteje se uporablja na nivoju testiranja enot
- Uporaba različnih tehnik (tok kontrole, vejitve, ...)
- Ker poznamo delovanje posameznih enot, izberemo testne primere, ki bodo preverili čim več notranjih struktur.
- Prednost: nujen način testiranje, pokritost kode
- Slabost: zahtevna izvedba, nemogoče popolno testirati

Starejše delitve testiranja

Metoda črne škatle (funkcionalno testiranje)

- Testiramo zunanje obnašanje programa na podlagi izbranih vhodnih podatkov.
- Teste določamo na podlagi začetne specifikacije (opisa) programske rešitve.
- Ponavadi na višjih nivojih testiranja (funkcijski testi, integracijsko testiranje)
- Različni pristopi: grupiranje vhodnih podatkov
- Prednosti: brez omejitev glede not. Strukture
- Slabosti: kdaj je testiranje popolno?

Metoda sive škatle:

- kombinacija obeh pristopov

Modelno-vodeno testiranje (MDT):

- teste določamo z modeli (UML) – ne MDTD

Starejše delitve testiranja

Testiranje od zgoraj navzdol (ang. top-down)

- Testiranje od splošnega proti podrobnemu
- začnemo z glavno proceduro in postopoma preverjamo proti manjšim enotam
- manj primerno za OO, ker ni jasen vrstni red
- potrebujemo nastavke (stub) – navidezen klican modul

Testiranje od spodaj navzgor (ang. bottom-up)

- Testiranje od podrobnega k splošnemu
- začnemo s testiranjem posameznih enot, te sestavimo v module …
- dejansko običajno testiranje
- potrebujemo (navidezen) gonilnik komponente (component driver)

Starejše delitve testiranja

Statično testiranje (ang. static)

- Testiranje brez poganjanja programa
- vključuje analize in preglede kode, strukture …
- literatura ta del pogosto imenuje “preverjanje” (ang. verificaton)

Dinamično testiranje (ang. dinamic)

- Testiranje s poganjanjem programa
- ključen del – ustrezno planiranje testnih primerov, priprava testnih podatkov in pričakovanih rezultatov
- literatura to vrsto pogosto imenuje “testiranje”

Regresijsko testiranje

27

- Najpogostejša oblika testiranja
- Obsega vse skupine testiranj
- Vnovično izvajanje že obstoječih testov po vsaki spremembi
- Izvaja se na vseh nivojih testiranja, najbolj pomembno pa je na nivoju testiranja enot
- Do izraza pride avtomatizacija testiranja (množica testov, časovna dinamika)
- Zelo pomembno za vzdrževanje

Namestitvena testiranja

28

- **Primerjalni test:** primerjava delovanja nameščenih sistemov po performansah (tipične uporabe)
- **Pilotni test:** testna manestitev, normalna uporaba, analiza
- **Alfa test:** oblika pilotnega testa pri razvijalcih med razvojem
- **Beta test:** oblika pilotnega testa pri uporabnikih nad deloma razvitim izdelkom (specifičen izdelek, zahtevno testiranje)

Beizer-ovi nivoji testiranja glede na zrelost testiranja 29

Nivo 0: ne loči med nepravilnim delovanjem programa ter napako v programu. Posledično tudi ni razlike med testiranjem in razhroščevanjem.

Nivo 1: dokazuje pravilnost programa. Ker iščemo potrditev pravilnega delovanja, je težnja izbirati testne primere, ki najverjetneje ne bodo našli napak. Če testi ne najdejo napak, ne vemo ali program nima napak ali imamo le slabe teste.

Nivo 2: dokazuje obstoj napak, saj se zavedamo, da vsak program vsebuje napake. Ko ne najdemo napak, ne vemo ali je program brez napak ali le nimamo dovolj dobrih testov.

Nivo 3: ne dokazuje nič specifičnega, temveč skuša le zmanjšati tveganje uporabe programske opreme.

Nivo 4: kultura organizacije - namen testiranja je v tem, da izboljšujemo kvaliteto programske opreme.

Modelno vodeno načrtovanje testiranja - MDTD 30

Model-Driven Test Design (MDTD)

Osnova za načrtovanje testiranja je MODEL

Testiranje obsega več aktivnosti, ki jih delimo na štiri splošne korake:

a) Načrtovanje testiranja

- prvi in najpomembnejši del procesa testiranja
- določimo cilje testiranja
- določimo strategijo testiranja
- ugotovimo testne zahteve
- modeliramo objekt testiranja
- formalno določimo kriterije pokritja

Modelno vodeno načrtovanje testiranja - MDTD

b) Priprava testiranja

- kreiramo testne primere in pričakovane rezultate
- določimo časovne okvire testiranja

c) Izvajanje testiranja

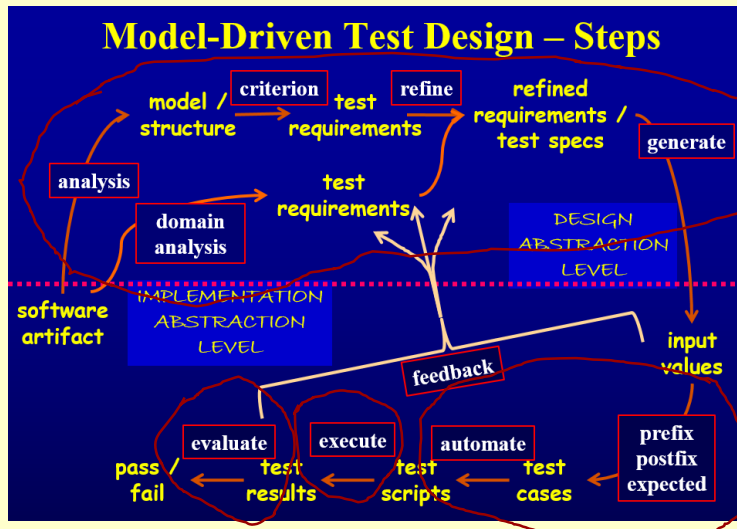
- izvajanje pripravljenih testnih primerov
- preverjanje uspešnosti na podlagi primerjave dobljenih rezultatov s pričakovanimi rezultati

d) Analiza rezultatov in ocenjevanje

- preverimo, če je testiranje odkrilo napake
- če so napake, poskušamo najti vzroke napak ter jih odpraviti

Pogosta napaka: (celotno) testiranje = izvajanje testiranja

Modelno vodeno načrtovanje testiranja - MDTD



Testno voden razvoj - TDD

Test-Driven Develipment (TDD)

Agilni pristop pri testiranju (XP)

Ideja:

- pred pisanjem rešitve (kode) napišemo teste enot
- te uporabimo pri testiranju enot

Prednosti:

- pred kodiranjem dobro poznamo specifikacije – hitro napišemo še teste
- isto bi morali narediti ob kasnejšem testiranju
- tako dejansko moramo delati pri višjenivojskih testih
- navezanost na regresijsko testiranje

Slabosti:

- ni v skladu s pogostim (kaskadnim) načinom razvoja
- problem različnih izvajalcev razvoja in testiranja

Avtomatizacija testiranja

- Testiranje PO lahko terja do 50% časa razvoja
- Avtomatizacija poveča učinkovitost in zmanjša stroške.
- Uporablja posebna programska orodja
- Izvaja ponavljajoča, dolgotrajna testiranja, kar je težko izvajati ročno (izničimo možnost napake človeškega faktorja)
- Običajno lahko avtomatiziramo le določen del testiranja.
- Najprimernejša aktivnost je izvajanje ponavljajočih testiranj – regresijsko testiranje z izvrševanjem testnih primerov.
- Aktivnost priprave in načrtovanja testiranja ni primerna za avtomatizacijo.
- Tudi aktivnost analize v veliki meri ni primerna za avtomatizacijo.

Testna terminologija

35

V nadaljevanju bomo spoznali množico standardnih terminov, ki jasno definirajo posamezne pojme iz področja testiranja**.**

1. Validacija *(ang. validation)* je proces preverjanja PO na koncu razvoja z namenom, da se ugotovi skladnost z namenom uporabe.

2. Verifikacija *(ang. verification)* je postopek ugotavljanja, če izdelki določene faze izpolnjujejo zahteve predhodne faze.

Validacija zahteva poznavanje določenega področja

Verifikacija je bolj tehnična aktivnost, ki uporablja tehnično znanje, programska orodja in postopke.

Obe se (lahko) izvajata s testiranjem.

Pojem IV&V – neodvisno ocenjevanje …

Testna terminologija

36

Pomembno je ločevanje pojmov **napaka, okvara** ter **odpoved PO.**

3. Napaka *(ang. fault)* je pomanjkljivost (pomota) v programu. Napake običajno vnesejo razvijalci z napačnim razumevanjem zahtev.

4. Okvara *(ang. error)* je neveljavno stanje, ki je posledica napake. Do okvare pride, ko se izvede del programa, ki vsebuje napako.

5. Odpoved *(ang. failure)* odstopanje od zahtevanega delovanja. Okvara preide v odpoved v primeru, ko ne dobimo pričakovanega rezultata. Vsaka okvara ne povzroči odpovedi.

Napaka:

- zaradi nerazumevanja zahtev s strani razvijalca
- zaradi napake razvijalca-koderja pri implementaciji

Ključna ustrezna izbira testnih vhodnih podatkov, da se sprožijo (vidne) okvare in tako najdemo napake.

Testna terminologija

37

Pomembno je tudi ločevanje med pojmoma **testiranje** in **razhroščevanje**.

6. Testiranje (*ang. testing*) je vrednotenje programske opreme na podlagi opazovanja izvajanja le-te.

7. Testna odpoved (*ang. test failure)* je izvajanje programa zaradi testiranja, ki se konča z odpovedjo.

8. Razhroščevanje *(ang. debugging)* je iskanje napake na podlagi odpovedi programa. Z razhroščevanjem odpravljamo napake, ki jih odkrijemo s testiranjem.

3 pogoji za razhroščevanje:

- **R**eachability – mesto v programu mora biti dosegljivo.
- **I**nfection – po izvedbi kode na tem mestu mora program prikazati odpoved
- **P**ropagation – odpoved se mora prikazati z nekakšnim odzivom (izpisom)

Testna terminologija

38

Pri izvajanju testiranja poznamo:

9. Vrednosti testnega primera (*ang. test case values*) je skupek vhodnih vrednosti za celotno izvedbo testiranega programa.

10. Pričakovani rezultati (*ang. expected results*) so rezultati, ki jih bo vrnil popolnoma pravilno delujoč program (v skladu z zahtevami).

11. Možnost vpogleda PO (*ang. software observability*) določa, kako težko je opazovati delovanje programa z vidika izhodov ter učinka delovanja le-tega.

12. Možnost nadzora (*ang. software controllability*) določa, kako zahtevno je zagotoviti programu ustrezne vhode za njegovo delovanje.

Določeno vrsto PO (recimo vgrajene sisteme – embeded systems) ni mogoče opazovati niti nadzorovati, ker kot vhode in izhode uporabljajo naprave

Testna terminologija

Formalizem za izvajanje tovrstnih testiranj.

13. Vrednosti predpone (*ang. prefix values*) so vrednosti, ki jih je treba najprej vnesti v program, da ta lahko začne brati vrednosti testnega primera.

14. Vrednosti končnice (*ang. postfix values*) so vrednosti, ki jih je treba poslati v program po vnesenih vrednostih testnega primera.

15. Vrednosti za preverjanje (*ang. verification values*) so del vrednosti končnic, ki poskrbijo, da vidimo rezultate vnesenih vrednosti testnega primera.

16. Ukaz za izhod (*ang. exit commands*) vrednosti za zaključek izvajanja testa in vrnitev v stabilno stanje.

17. Testni primer (*ang. test case*) je sestavljen iz zbirke vhodnih podatkov, pričakovanih izhodnih rezultatov ter vrednosti predpon in končnic, ki so potrebne za izvedbo in oceno testirane PO.

18. Testna množica (*ang. test set*) je množica testnih primerov.

Kriterij pokritja

19. Izvršljiva testna skripta (*ang. executable test script*) je testni primer v obliki za avtomatsko izvedbo na testnih orodjih, ki tvorijo poročilo.

Nemogoče je testirati čisto vse možne vhode, zato uvedemo formalne kriterije pokritja (ang. coverage criterion) na naslednji način:

20. Testna zahteva (*ang. test requirement*) je specifična formalna zahteva, ki jo mora testni primer zadovoljiti.

Testne zahteve lahko napišemo na podlagi specifikacije, izvorne kode, itd.

Ponavadi se testne zahteve uporablja v **množicah testnih zahtev** (*ang. set of test requirements - TR*)

Primer:

Testiranje vrečice sadnih bombonov; pokriti je treba bombone z okusom banane

Kriterij pokritja

21. Kriterij pokritja (*ang. coverage criterion*) je pravilo ali množica pravil, ki določa testne zahteve za testno množico.

Pomen:

• Kriteriji pokritja nam na podlagi formalnih modelov pomagajo izbrati vhodne podatke za testiranje

• S pravilno uporabo kriterijev pokritja je večja verjetnost, da bomo odkrili napake v programski opremi

• Kriterij lahko uporabimo kot pokazatelj kvalitete programa.

• Testne zahteve so s kriterijem pokritja popolno in jasno določene.

Kriterij pokritja

22. Pokritje (*ang. coverage*): za množico testnih zahtev *TR* in kriterij pokritja *C* velja, da testna množica *T* zadošča *C* takrat in samo takrat, ko za vsako testno zahtevo *tr* iz *TR* obstaja vsaj en test *t* iz *T* tako, da *t* zadošča *tr*.

Primer: v vrečici imamo bombone 3 okusov: Ananas, Banana in Papaja.

C: preveriti vse tri okuse

TR= {okus=Ananas, okus=Banana, okus=Papaja}

T vsebuje 5 bombonov: 2 Ananasa, 1 Banano in 2 Papaji

To zadošča kriteriju pokritja.

23. Nivo pokritja (*ang. coverage level*) je razmerje med številom z elementi testne množice T pokritih zahtev in vsemi testnimi zahtevami iz TR.

Problem neizvedljivih zahtev – običajno jih izločimo jih iz TR.

V praksi je običajno težko doseči 100% nivo pokritja.

Kriterij pokritja

Dva načina uporabe kriterija pokritja:

- znanstven pristop – služi za generiranje testne množice (problem: ročno delo)

- indstrijski pristop – testno množico pripravimo izven in primerjamo nivo pokritja z TR (problem: kaj pomeni 10% odstopanja)

Teorija: odločitev, če T ustreza C je odločljiv (ni neizvedljivih zahtev)
Praksa: skoraj vedno so neizvedljive zahteve – ni odločljivo

Medsebojen vpliv kriterijev – podobno (a ne enako) kot množice

24. Vsebovanost kriterijev (*ang. criteria subsumption*) kriterij pokritja C1 je vsebovan v kriteriju C2 takrat in natanko takrat, če vsaka testna množica, ki zadošča C2 zadošča tudi C1.

Primer: C1 = preverjanje bombonov na črko A, C2 je preverjanje vseh vrst

Teoretično gledano na vsebovanost vpliva neizvedljivost, v praksi neizvedljive zahteve izločimo

Kriterij pokritja

Merila za dober kriterij pokritja:

a) Zahtevnost za pripravo testnih zahtev (vpliv izdelka in kriterija)

b) Zahtevnost za generiranje testov (povezanost z c)

c) Kako dobro testi odkrivajo napake

Premislek: če C2 vsebuje C1, ali bo odkril več napak?

- Da (več zahtev) in
- Ne (neizvršljivi testi)

Uravnotežene rešitve

Pristop pri obravnavi testiranja

"Testiranje je enostavno – treba je le določiti graf in ga pokriti s testnimi primeri" (Beizer)

Staro dojemanje testiranja:

- množica različnih vrst testov,
- vsak svoje specifike
- nejasne definicije, delitve ...

Novo dojemanje testiranja:

- testiramo (vedno enako) na podlagi abstraktnega modela
- Uporabimo pristop s pokritji
- V praksi potrebujemo dve preslikavi:
  - predmet testiranja -> model
  - pokritje –> dejanski testni primeri

Pristop pri obravnavi testiranja

V nadaljevanju bo obravnava različnih testnih tehnik z vidika kriterijev pokritja

Štirje abstraktni modeli pokrivajo vse možne tehnike**:**

1. **pokritje z grafi (ang. Graph Coverage)**
2. **pokritje z logični izrazi (ang. Logic Coverage)**
3. **delitev prostora vhodnih podatkov (ang. Input Space Partitioning)**
4. **sintaksno testiranje (ang. Syntax-Based Testing)**

Prihodnost testiranja

47

Kako izboljšati testiranje?

- Več in boljša testna orodja
- Boljši prijemi in tehnike za testerje
- Več tehničnih znanj
- Bolj specializirana orodja

Trenutne težave:

- Pomanjkanje izobraževanja o testiranju
- Nadgradnje obstoječih procesov z več testiranja
- Bolj močna, učinkovita in uporabna testna orodja

Prihodnost testiranja

48

Z vidika raziskav testiranja moramo:

- razviti boljše tehnike in orodja,
- preliti teorijo v prakso in
- eksperimentalno dokazati, da se to splača

Z vidika izobraževanja testiranja moramo:

- spremeniti predmetnike,
- učiti več teorije testiranja in
- priučiti - motivirati praktično uporabo le-tega v praksi

Literatura

1. Paul Ammann, Jeff Offutt: Introduction to software testing, Cambridge University Press, 2008.
2. Wikipedia