Predavanje 8: Uvod u testiranje softvera i *white box* testiranje

Metode verifikacije (unit testiranje, pregled, inspekcija, prolaz-kroz) koda primjenjuju se u cilju kontrole i poboljšanja koda bez stvarnog pokretanja/izvršavanja koda. Bez obzira što metode verifikacije daju dobre rezultate u identifikaciji defekata koda, one nikada ne mogu zamijeniti testiranje softvera pri njegovom izvršavanju.

Ovo predavanje obrađuje sljedeće tematske jedinice povezane sa testiranjem:

1. Definicija i ciljevi testiranja
2. Strategije testiranja softvera
3. Klasifikacije softverskih testova
4. *White box* testiranje

1.DEFINICIJE I CILJEVI TESTIRANJA

U prvom predavanju je uvedena historija testiranja i razdvajanje testiranja na osiguranje kvaliteta softvera i kontrolu kvaliteta softvera, verifikaciju i validaciju.

Podsjetimo se prve definicije:

Klasična definicija testiranja softvera Myers (1979):

"Testiranje je proces izvršavanja programa u cilju pronalaska grešaka" .

Novija i široko prihvaćena definicija softvera je:

Testiranje softvera je **formalni** proces koji se izvodi sa **specijaliziranim timom** za testiranje kojim se ispituju softverski moduli ili cjelokupni softverski paketi **izvršavanjem programa** na kompjuteru. Svi povezani testovi se izvršavaju u skladu sa **odobrenim test procedurama** i **odobrenim test slučajevima**.

Riječi i fraze naglašene u definiciji:

- **Formalni** proces – Planovi testiranja softvera su dijelovi projekta i plana kvaliteta, sa unaprijed određenim vremenskim rasporedom i aktivnostima.
- **Specijalizirani tim** za testiranje– Specijalizirani tim izvršava profesionalno i objektivno testiranje. Generalno je prihvaćeno da testovi koji se izvršavaju od strane developera koji su i pisali softverski kod vode do siromašnih rezultata.
- **Izvršavanje programa** – Testiranjem pri izvršavanju softvera vrši se validacija softvera.
- **Odobrene test procedure** – Procesi testiranja se izvršavaju u skladu sa testnim planom i procedurama testiranja koje su odobrene i prilagođene od strane softverske kompanije i u skladu sa SQA procedurama.
- **Odobreni test slučajevi** – Slučajevi testiranja (test cases) koji se koriste za ispitivanje funkcionalnosti softvera sastavni su dio testnog plana.

CILJEVI TESTIRANJA

Direktni ciljevi

■ Identifikacija i objavljivanje onoliko grešaka koliko je moguće pri izvršavanju softvera.

■ Postizanje prihvatljivog nivoa kvaliteta softvera (uvijek se može očekivati da će ostati određen broj neidentificiranih bagova u softveru).

Indirektni ciljevi

■ Skupljanje zapisa o softverskim greškama koji se koriste za korektivne i preventivne akcije.

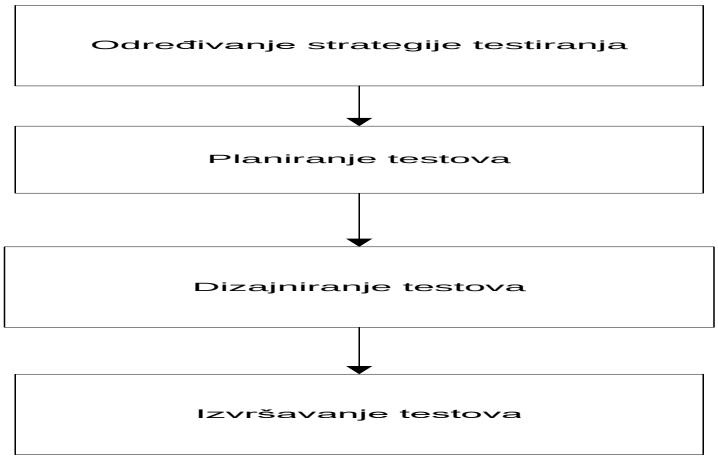
Važno je uočiti da nije slučajan izostanak cilja: potvrda da je softverski paket potpuno spreman. Myers (1979) je ovo jasno sumirao:

"Ako je vaš cilj da se pokaže odsustvo grešaka nećete ih otkriti mnogo."

"Ako je vaš cilj da pokažete postojanje grešaka, otkriti ćete veliki procenat istih".

PROCES TESTIRANJA

Proces testiranja sastoji se od koraka: određivanje strategije testiranja, planiranja testova, dizajniranja testova, izvršavanja testova (slika 1).



Slika 1: Proces testiranja

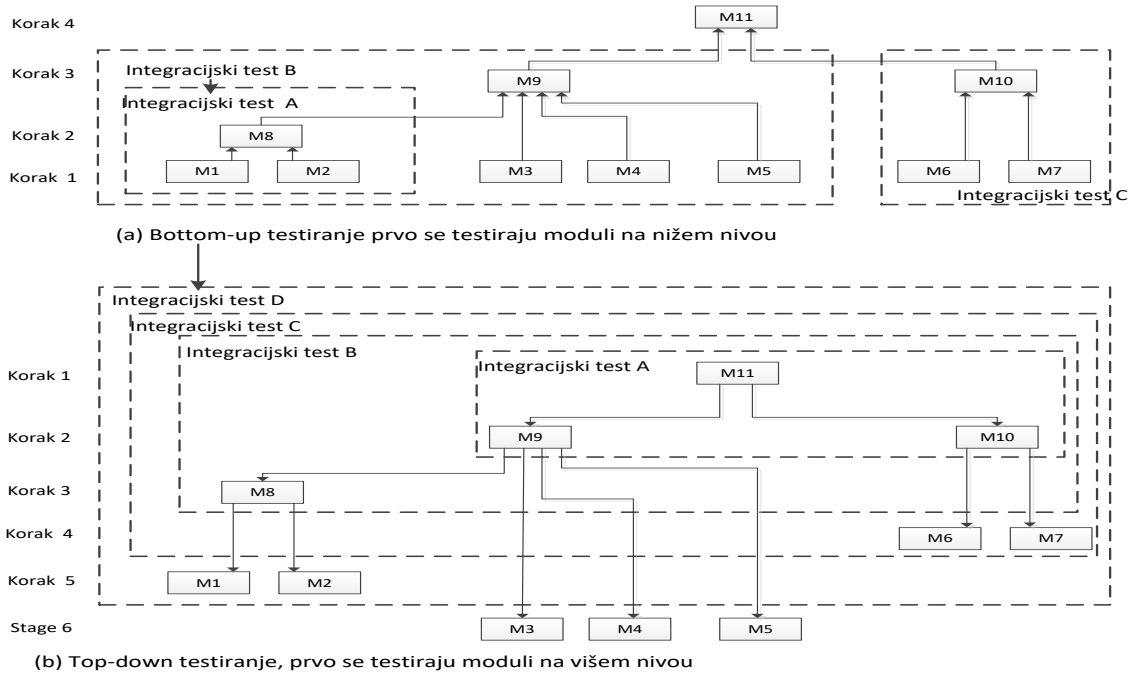
2. STRATEGIJE TESTIRANJA SOFTVERA

Osnovne dvije strategije testiranja softvera su:

■ *Bing bang* testiranje - Testiranje softvera u cijelosti, nakon što je kompletan softverski paket raspoloživ.

■ Inkrementalno testiranje - Testiranje pojedinačnih softverskih modula (unit testovi) poslije njihove implementacije, nakon toga testiranje grupe testiranih modula sa novim završenim modulima (integracijski testovi) i na kraju testiranje cjelokupnog sistema (sistemski testovi). Inkrementalno testiranje se izvršava sa dvije osnovne strategije (slika 2): *bottom-up (odozgo-nadolje)* i *top-down (odozgo-nagore)*. Softverski paket je uglavnom sastavljen od hijerarhije

softverskih modula. U *top-down* testiranju, prvo se testira glavni (main) modul tj. modul na najvišem nivou u softverskoj strukturi; zadnji moduli koji se testiraju su moduli na najnižem nivou. U *bottom-up* strategiji, redoslijed testiranja je obrnuti: prvo se testiraju moduli na najnižem nivou, a glavni (main) modul se testira zadnji.



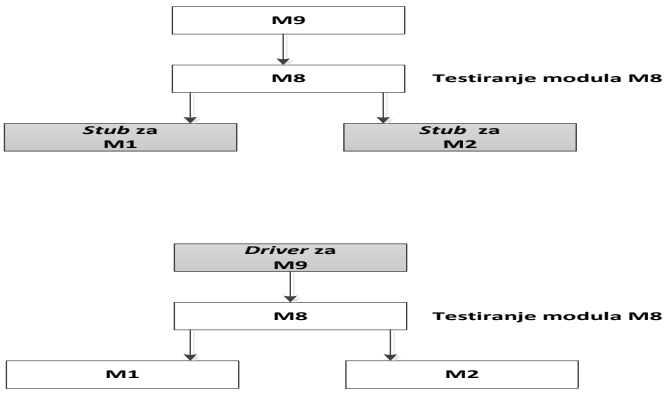
Slika 2: *Bottom-up* (a) i *top-down*(b) testiranje-ilustracija

Inkrementalni putevi testiranja pokazani na slici 2 su uspostavljeni kao "horizontalna sekvenca" ("*breadth first*"), a može se izabrati put koji je "vertikalna sekvenca" ("*depth first*").

Zamjenski programski elementi (STUB I DRIVER) za inkrementalno testiranje

Zamjenski programski elementi (*stubs* i *drivers*) su simulatori za module koji nisu raspoloživi kada se izvršava testiranje.

Stub (često se naziva "*dummy* modul") zamjenjuje neraspoloživi modul nižeg nivoa, koji je podređen modulu koji se testira.



Slika 3: Korištenje zamjenskih modula *stub* i *driver* za inkrementalno testiranje

Driver je zamjenski modul ali višeg nivoa koji koristi modul koji se testira. Na slici 3 prikazan je scenarij korištenja *stub* i *driver* modula.
Glavna prednost inkrementalne *bottom-up* strategije je jednostavno lako izvršavanje, dok glavni nedostatak ove strategije je kasno posmatranje programa kao cjeline.
Glavna prednost *top-down* strategije je uvid u cjelokupne funkcionalnosti programa kratko nakon aktivizacije modula višeg nivoa. U mnogim slučajevima ova karakteristika dozvoljava raniju identifikaciju grešaka nastalih u analizi i dizajnu. Glavni nedostatak *top-down* strategije odnosi se na poteškoće pripreme zahtijevanih *stubova*.
Testeri prilikom odabira strategije testiranja slijede developerski razvojni pristup jer je krucijalno da se testiranja izvršava odmah nakon kodiranja modula.

Big bang strategija nasuprot inkrementanoj strategiji

Big bang strategija testiranja uglavnom je namjenjena za testiranje jednostavnih (malih) projekata. Identifikacija grešaka primjenom *bing bang* strategije postaje sve teža u ovisnosti od kvantiteta softvera. Ispravljanje grešaka je često težak zadatak jer se vrše korekcije više modula u isto vrijeme.

U suprotnosti sa *big bang* testiranjem, inkrementalno testiranje ima više prednosti, neke od važnijih su:
(1) Inkrementalno testiranje se uobičajno izvršava na relativno malim pojedinačnim softverskim modulima. To olakšava lakše postizanje visokog procenta identificiranih grešaka.
(2) Identifikacija i korekcija grešaka je mnogo jednostavnija i zahtjeva manje resursa jer se izvršava na ograničenom dijelu softvera.
(3)veliki broj grešaka se otkriva i popravlja u ranim fazama razvoja i testiranja, što sprječava “migraciju” defekata u kasnije, kompleksnije faze razvoja i testiranja.

Glavni nedostatci inkrementalnog testiranja se odnose na dodatne resurse koji su potrebni za pripremu zamjenskih modula i izvođenje mnogobrojnih testnih operacija za isti program (bing bang testiranje zahtjeva samo jednu operaciju testiranja).

3. KLASIFIKACIJE TESTIRANJA SOFTVERA

Softverski testovi mogu se klasificirati u skladu sa konceptom testiranja ili u skladu sa McCall klasifikacijom softverskih karakteristika.

Klasifikacija prema konceptima testiranja

U ovisnosti da li se testiranje softvera vrši samo na osnovu izlaza ili je uključena i interna struktura softvera testovi se klasificiraju kao IEEE (1990):

■ *Black box* (funkcionalno) testiranje. Otkriva greške (bagove) na osnovu rezultata (izlaza) prilikom izvršavanja programa. *Black box* testiranje zanemaruje interni put procesiranja unutar softverskog koda.

■ *White box* (strukturalno) testiranje. Ispituje strukturu koda u namjeri da identificira bagove. Izraz "*white*" je dat da istakne razliku između ove metode i black box testiranja. Pored ovoga postoje i druga ime za ovu metodu, a to je "*glass box*" testiranje.

U mnogim slučajevima za testiranje kompleksnih softverskih rješenja koriste se oba koncepta. Ponekad se primjenjuje tzv. *gray box* testiranje, neki moduli se testiraju sa *black box*, a neki sa *white box* strategijom.

Klasifikacija u skladu sa karakteristikama kvaliteta softvera

McCall's model (prikazan u predavanju 1) uspostavlja vezu između testova i karakteristika softvera (tabela 1).

Kategorija Faktora kvaliteta	**Faktor kvaliteta**	**Vrste testiranja**	***White box***	***Black box***
Operacijski Faktori	1. Korektnost	1.1. Test korektnosti izlaznih rezultata		+
		1.2 Test korektnosti i kompletnosti dokumentacije		+
		1.3 Test vremena (brzine) reakcije (*reaction time*)		+
		1.4 Test procesiranja podataka i korektnosti računanja	+	
		1.5 Test kvaliteta softvera u skladu sa standardima	+	
	2.Pouzdanost	2. Testovi pouzdanosti		+
	3.Spremnost/Izdržljivost	3. Testovi izdržljivosti (*stress* i *load* testovi)		+
	4. Integritet	4. Testovi sigurnosti		+
	5. Upotrebljivost	5.1 Traning testovi upotrebljivosti		+
		5.2 Operacijski testovi upotrebljivosti		+
Revizija	6. Održavanje	6. Testovi održavanja	+	+
	7. Fleksibilnost	7. Testovi fleksibilnost		+
	8. Lakoća testiranje	8. Testovi mogućnosti testiranja		+
Tranzicija	9. Prenosivost	9. Testovi prenosivosti		+
	10. Ponovno korištenje (*reusability*)	10. Testovi ponovne iskoristljivosti	+	
	11. Interoperabilnost (*Interoperability)*	11.1 Testovi interoperabilnosti sa drugim softverom		+
		11.2 Testovi interoperabilnosti sa drugim uređajima		+

Tabela 1: Kategorije testova za McCall faktore kvaliteta softvera

Primjenjivost *white* ili *black* koncepta testiranja varira u ovisnosti od vrste testiranja. Testiranje ovisi i od vrste aplikacije - web aplikacija, mobilna aplikacija, testiranje igara. Postoje i razna testiranja povezana sa životnim ciklusom razvoja softvera (npr. test prihvatljivosti).

4.WHITE BOX TESTIRANJE

White box testiranje omogućava (vidljivo u tabeli 1) izvršavanje testova procesiranja podataka i korektnosti računanja, testiranje kvaliteta softvera u odnosu na standarde, testova održavanja (koliko je zahtjevno održavanje) i testova ponovne upotrebe koda (testira da li modul predložen za ponovno korištenje (reuse) odgovara standardnima i procedurama za uključivanje u biblioteku komponenti).
Slijedi detaljnije o *white box* testiranju procesiranja podataka i korektonosti kalkulacija.

4.1. Testovi procesiranje podataka i korektnosti računanja

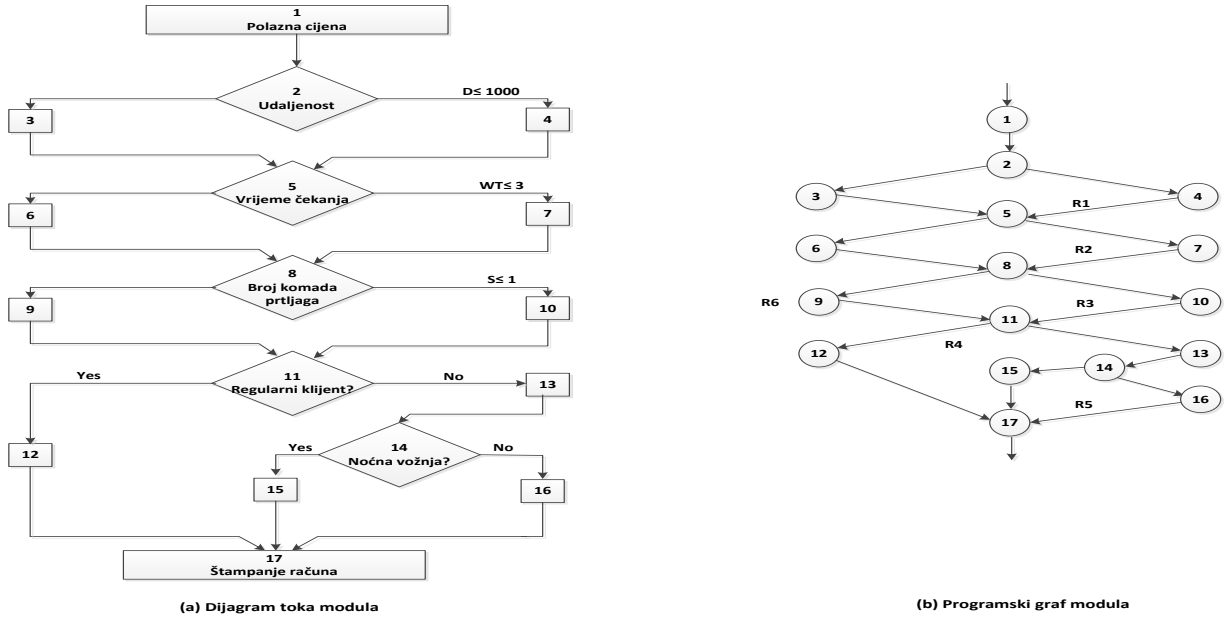
Zadatak testova korektnosti procesiranja podataka i računanja ("*white box correctness test*") je ispitivanje svake operacije računanja u sekvenci operacija. Ispitivanje se vrši sa pripremljenim testnim slučajevima. **Generalno testni slučaj** je skup akcija koji se izvršava da verifikuje pojedinačnu karakteristiku ili funkcionalnost aplikacije. Testni slučaj sadrži ulazne podatke i na osnovu tih podataka se računa izlaz za koji se provjerava da li je očekivani. Upotrebljava se za razne vrste *black box* testiranje (radi se u toku kursa).
U kontekstu *white box* testiranja postoje više alternativnih pristupa planiranja testnih slučajeva za testiranje procesiranja podataka i korektnosti računanja. Neki od njih su:

- Obuhvat puteva (*Path coverage*) – planiranje testova da obuhvate puteve u programskom kodu
- Obuhvat iskaza/linija (*Statement/Line coverage*) – planiranje testova da obuhvate linije u programskog kodu.
- Obuhvat grana (*Branch coverage*)
- Obuhvat uslova (*Conditional coverage*)
- Obuhvat petlji (*Loop coverage*)
- Obuhvat toka podataka (*Data flow coverage*)

***Testovi korektnosti i obuhvat puteva (path coverage) i linija* (statement/*line coverage*)**

Motivacija testiranja puteva je da se postigne kompletan obuhvat programskog koda testirajući sve njegove moguće puteve.
Mogući putevi u softverskim modulima se kreiraju u skladu sa uslovnim iskazima kao što su: IF–THEN–ELSE , DO WHILE, DO UNTIL,….
Ovaj koncept nije praktičan u mnogim slučajevima jer je potrebno mnogo resursa i testnih slučajeva za njegovo izvršavanje.

Obuhvat iskaza/linija (*statement/line coverage*) strategija ima težnju da dizajnira testne slučajeve tako da se svaki iskaz tj. svaka linija koda u programu izvrši barem jednom.
Usložnjavanjem koda je teže dizajnirati testne slučajeve koji će garantovati da se svi putevi i svala linija koda izvrši barem jednom. Za određivanja broja puteva i broja linija od pomoći su dijagram toka (*flow chart*) i graf programskog toka (*flow graph*). U dijagramu toka romb predstavlja opcije uslovnih iskaza (*decisions*), dok pravougaonici predstavljaju softverske sekcije koje su spojene sa ovim uslovima.
Sljedeći primjer demonstrira dijagram toka i programski graf toka za taksimetar – softverski modul koji računa cijenu vožnje: Imperial Taxi Servis (ITS) uslužuje 'one-time' putnike i regularne klijente (identificirane sa taksi karticom). ITS taxi cijenu vožnje za 'one-time' putnike računa:
(1) Početna cijena: 2 KM. Ova cijena uključuje razdaljinu do 1000 m,
i jedno usputno čekanje do 3 minute.
(2) Za svakih dodatnih 250 m plaća se: 25 feninga.
(3) Za svake 2 dvije dodatne minute stajanja ili čekanja: 20 feninga.
(4) Jedan komad prtljaga: bez naplate; svaki dodatni komad: 1 KM.
(5) Noćni dodatak: 25%, za prevoz između 21.00 i 06.00.
Regularni klijenti imaju 10% popusta i ne naplaćuju im se noćni dodatak.
!Napišite programski kod.
Dijagram toka i graf programskog toka za ovaj modul (5 odluka) su prikazani na slici 4.



Slika 4: ITS kalkulacija – (a)dijagram toka (flow chart) i (b)programski graf toka

Na dijagramu toka uočavaju se 24 različita puta što indicira da je potrebno da bi se postigao puni obuhvat puteva ovog softverskog modula pripremiti najmanje 24 testna slučaja (tabela 2).

Broj puta	Put
1	1-2-3-5-6-8-9-11-12-17
2	1-2-3-5-6-8-9-11-13-14-15-17
3	1-2-3-5-6-8-9-11-13-14-16-17
4	1-2-3-5-6-8-10-11-17
5	1-2-3-5-6-8-10-11-13-14-15-17
6	1-2-3-5-6-8-10-11-13-14-16-17
7	1-2-3-5-7-8-9-11-12-17
8	1-2-3-5-7-8-9-11-13-14-15-17
9	1-2-3-5-7-8-9-11-13-14-16-17
10	1-2-3-5-7-8-10-11-12-17
11	1-2-3-5-7-8-10-11-13-14-15-17
12	1-2-3-5-7-8-10-11-13-14-16-17
13	1-2-4-5-6-8-9-11-12-17
14	1-2-4-5-6-8-9-11-13-14-15-17
15	1-2-4-5-6-8-9-11-13-14-16-17
16	1-2-4-5-6-8-10-11-12-17
17	1-2-4-5-6-8-10-11-13-14-15-17
18	1-2-4-5-6-8-10-11-13-14-16-17
19	1-2-4-5-7-8-9-11-12-17
20	1-2-4-5-7-8-9-11-13-14-15-17
21	1-2-4-5-7-8-9-11-13-14-16-17
22	1-2-4-5-7-8-10-11-12-17
23	1-2-4-5-7-8-10-11-13-14-15-17
24	1-2-4-5-7-8-10-11-13-14-16-17

Tabela 2: ITS-**Potpuna lista puteva**

Sa grafa programskog toka može se utvrditi da se puni linijski obuhvat ITS softverskog modula može postići sa ispitivanjem minimalnog broja puteva – ukupno 3 kako je prikazano u tabeli 3.

Broj puta	Put
1	1-2-3-5-6-8-9-11-12-17
23	1-2-4-5-7-8-10-11-13-14-15-17
24	1-2-4-5-7-8-10-11-13-14-16-17

Tabela 3: ITS minimalni broj puteva/puni linijski obuhvat

Omjer test slučajeva koji se zahtijevaju za test sistema sa punim obuhvatom linija (3 testa) nasuprot punog obuhvata puteva (24 testa) je 1:8. Omjer raste rapidno sa kompleksnošću programa.

Neovisni putevi

Primjenom McCabe metrike za određivanje obuhvata neovisnih puteva može se odrediti broj testnih slučajeva (predavanje Metrike). Metrika je bazirana na teoriji grafova.
Neovisni put je: "Svaki put grafa programskog toka koji uključuje najmanje jednu ivicu (tok) koja nije uključena u neki od prethodno formiranih puteva".

Kako je spomenuto, McCabe metrika kompleksnosti- $V(G)$ također određuje maksimalan broj neovisnih puteva koji se mogu indicirati u grafu programskog toka.
$V(G)$ se može izraziti na 3 različita načina, svaki baziran na grafu programskog toka.
1) $V(G) = R$ 2) $V(G) = E - N + 2p$ (p=1)
3) $V(G) = P + 1$

U ovim jednačinama R je broj regiona u grafu programskog toka. Svako okruženo područje u grafu se tretira kao region. Dodatno, područje oko grafa nije okruženo, ali se broji kao jedan dodatni region.
E je broj ivica (tokova) u grafu programskog toka, N je broj čvorova u grafu programskog toka, P je broj odluka koje su sadržane u grafu predstavljene sa čvorom koji ima više od jednog izlaznog toka.
Primjenom gornjih jednačina za ITS modul, na osnovu slike 4:
$R = 6, E = 21, N = 17, P = 5.$
1) $V(G) = R = 6$ 2) $V(G) = E - N + 2 = 21 - 17 + 2 = 6$
3) $V(G) = P + 1 = 5 + 1 = 6$

Rezultati metrike indiciraju da je maksimalan broj neovisnih puteva 6 (npr. putevi prikazani u tabeli 4).

Broj Puta	Put	Dodane grane	Broj dodanih grana
1	1-2-3-5-6-8-9-11-12-17	1-2, 2-3, 3-5, 5-6, 5-8, 8-9, 9-11, 11-12, 12-17	9
2	1-2-4-5-6-8-9-11-12-17	2-4, 4-5	2
3	1-2-3-5-7-8-9-11-12-17	5-7, 7-8	2
4	1-2-3-5-6-8-10-11-12-17	8-10, 10-11	2
5	1-2-3-5-6-8-9-11-13-14-15-17	11-13, 13-14, 14-15, 15-17	4
6	1-2-3-5-6-8-9-11-13-14-16-17	14-16, 16-17	2

Tabela 4: Maksimalan skup neovisnih puteva

Linijski obuhvat (*line coverage*) zahtjeva mnogo manje testova, ali više puteva ostaje ne testirano.

Obuhvat grana (branch coverage)

Obuhvat grana (*branch coverage*) za iskaze grananja dizajnira testne slučajeve tako da se izvrše svi mogući pravci kretanja (grane). Npr. `if` iskaz se testira i sa pozitivno i sa negativno procjenjenim izrazom grananja. `Switch` iskaz se testira tako da se testira svaka `case` i `default` grana. Ako ne postoji `default` grana `switch` se testira i sa vrijednošću koja ne odgovara nijednoj `case` grani. Ova strategija je strožija u odnosu na strategiju obuhvata iskaza.

Obuhvat uslova (condition coverage)

U ovom strukturalnom testiranju testni slučajevi su dizajnirani da svaka komponenta od složenog uslovnog iskaza dobija `true` i `false` vrijednost. Kod ove strategije broj testnih slučajeva raste eksponencijalno sa brojem komponenti uslova.

Testiranje grana je jednostavnije jer se preračunati izrazi tj. izrazi kao cijelina postavljaju na `true` i `false`.

Npr. za `if (!done && (value <100|| c='x'))…` procjenjuju se tri izraza `!done`, `value<100`, `c='x'`. Broj testova neophodnih za obuhvat uslova je $2^3 = 8$. Često se kreiraju i `truth` tabele da bi se razmotrile sve različite kombinacije uslova i postigao potpuni obuhvat iskaza koji se testira.

Ova tehnika pomaže kod otkrivanja
grešaka nastalih upotrebom logičkih i relacijskih operatora, nepravilno postavljenim zagradama.

Obuhvat toka podataka (data flow coverage)

Za dizajn testnih slučajeva biraju se putevi u skladu sa lokacijama definicije i korištenja varijabli. Ova tehnika nije praktična za intezivno korištenje. Pogodna je za module sa ugnježdenim if iskazima i petljama.

Obuhvat petlji (loop coverage)

Petlje su osnova mnogih algoritama i zahtjevaju detaljno testiranje. Potrebno je dizajnirati skup testnih slučajeva da se obuhvate sljedeće situacije:

- Jednostavna petlja sa n prolaza
  - prolazak uslova kojim se omogućava ulazak u petlju
  - jedan prolazak kroz petlju
  - dva prolaska kroz petlju
  - m prolazaka kroz petlju gdje je m<n
  - (n-1), (n), (n+1) prolazaka kroz petlju
- Ugnježdene petlje

  -start sa unutrašnjom petljom; sve ostale petlje se postavljaju na minimalni broj prolaza

  -provesti testiranje kao za jednostavnu petlju sa n prolaza

  - odraditi isto za sve vanjske petlje

- Povezane petlje
  - ako su neovisne petlje koristi se testiranje za jednostavnu petlju sa n prolaza
  - ako su ovisne petlje tretiraju se kao ugnježdene petlje

Očigledno je da ovisno od strukture petlji broj testova je jako velik.

5. Prednosti i nedostaci *white box* testiranja

Glavne prednosti *white box* testiranja su:

■ Direktna iskaz po iskaz provjera koda omogućava određivanje softverske korektnosti. Uključuje detaljne provjere da li su algoritmi korektno postavljeni i kodirani.

■ Dozvoljava izvršavanje linijskog obuhvata i prikaz linija koda koje još nisu izvršene. Tester može poslije inicijalizirati testne slučajeve da obuhvati ove linije koda.

■ Provjerava kvalitet kodiranja i odanost standardima testiranja.

Glavni nedostaci *white box* testiranja su:

■ Potrebno je mnogo resursa, mnogo više nego što je potrebno za black box testiranje istog softverskog paketa.

■ Nemogućnost testiranja softverskih performansi (vrijeme odgovora, pouzdanost, load) i drugih testnih klasa vezanih za operacije, revizije i faktore tranzicije.

Karakteristike *white box* testiranja ograničavaju njegovo korištenje na softverske module veoma velikog rizika.