Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.31
DOI: https://doi.org/10.24867/07BE04Adamovic

AUTOMATIZACIJA TESTIRANJA PRORAČUNA KRATKIH SPOJEVA

AUTOMATION OF SHORT CIRCUIT CALCULATION TESTING

Milovan Adamović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je izvršena automatizacija testova koji vrše proračun kratkih spojeva koristeći softversku funkcionalnost za proračun. Najpre su objašnjeni osnovni principi testiranja softvera, osnovni principi automatskog testiranja i na kraju sama realizacija automatskih testova. Predstavljen je algoritam koji se koristi za postavku šema u željeno stanje pre samog proračuna kratkih spojeva kao i rezultati izvršenih testova.*

Ključne reči: *kratki spojevi, testiranje softvera, automatsko testiranje.*

Abstract *–In this paper, the automation of tests that calculate short circuit using software function for calculating. First, the basic principles of software testing, the basic principles of automatic testing, and finally the realization of automatic tests, are explained. The algorithm used to set schematics to the desired state before calculate short circuit is presented, as well as the results of the performed tests.*

Keywords: *short circuit, software testing, automatic testing.*

1. UVOD

Testiranje softvera je proces koji se koristi da bi se utvrdila ispravnost, potpunost i kvalitet razvijenog softvera. Nepravilno kreiran softver može imati velike posledice. Ukoliko je sam softver namenjen za komercijalnu upotrebu i ukoliko se ne ponaša očekivano može izazvati novčane gubitke i silne penale od strane korisnika.

Testiranje softvera je način da se obezbedi manji broj grešaka, manji trošak održavanja i sveukupne cene softvera. Testiranje je aktivnost koja se sprovodi radi evaluacije kvaliteta proizvodnje softvera i njegovog poboljšanja.

Ono nije aktivnost koja počinje samo nakon kompletiranja faze kodiranja. Softversko testiranje se danas vidi kao aktivnost koja obuhvata ceo proces razvoja i održavanja, i predstavlja važan deo kompletne konstrukcije softvera. Planiranje testiranja treba da počne sa ranom fazom izrade zahteva za kvalitet softverskog proizvoda i procesa njegovog projektovanja, i test planovi i procedure moraju biti sistematski i kontinualno razvijani i po potrebi redefinisani.

Pravi stav prema kvalitetu je prevencija, mnogo je bolje izbeći probleme nego ih ispravljati, kao i za sve ostale životne situacije fraza je neizbežna da je bolje sprečiti, nego lečiti [2].

U radu je predstavljena problematika i pronalaženja najoptimalnijeg rešenja za pripremu šema od interesa za proračun kratkih spojeva pomoću automatskih testova. Algoritam rešenja prikazan je na blok dijagramu i opisno je objašnjen princip realizacije.

Rad je organizovan tako da su u drugom delu objašnjeni modeli procesa razvoja softvera, dok su u trećem delu objašnjeni nivoi testiranja i test dizajn tehnike. U četvrtom delu objašnjen je postupak za proračun kratkih spojeva. U petom delu objašnjen je način realizacije automatskih testova za potrebe rada i algoritma primenjenog u testovima koji vrše postavku šema sistema. Na kraju su dati predlozi za proširenje rada kroz opis u zaključku.

2. MODELI PROCESA RAZVOJA SOFTVERA

Modeli procesa razvoja softvera su različiti procesi i metodologije, koji se biraju i koriste u razvoju projekta, u zavisnosti od ciljeva i zahteva projekta. Postoji više modela koji su razvijeni tokom godina, svaki sa drugačijim ciljevima. Uopšteno gledano modeli specificiraju različite faze u procesu razvoja, kao i redosled kojim se te faze izvršavaju. Odabir modela koji se koristi u razvoju softvera ima ogroman uticaj na testiranje koje će se sprovesti. Odabrani model definiše šta, kada i kako će se testirati i utiče na odabir tehnika koje će se koristiti. Navedeni su i opisani samo neki od postojećih modela [2].

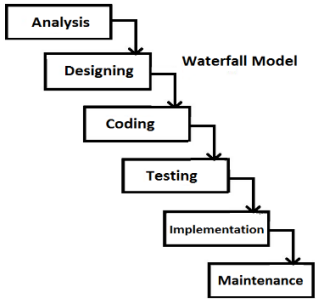
2.1. Waterfall model (model vodopada)

Waterfall model (model vodopada) je prvi model koji je uveden u razvoj softvera. Često se naziva još linearni sekvencijalni model. Veoma je lak za razumevanje i upotrebu. Prema ovom modelu svaka faza mora u potpunosti biti kompletirana pre nego što počne sledeća, odnosno faze se ne mogu preklapati [2].

Na slici 2.1 prikazan je blok dijagram waterfall modela razvoja softvera.

Nakon završetka jedne faze započinje druga, sa testiranjem se kreće nakon sto je razvoj kompletno završen. Prednosti modela su da je lak za razumevanje i upotrebu, nema preklapanja faza, pogodan je za manje projekte gde su zahtevi jasni. Mane modela su te da se operativni softver dobija kasno, veliki rizik i neizvesnost, ukoliko je aplikacija u fazi testiranja teško se vratiti nazad i nešto promeniti u fazi razvoja, loš u slučaju rizika od čestih promena zahteva.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Duško Bekut, red. prof.

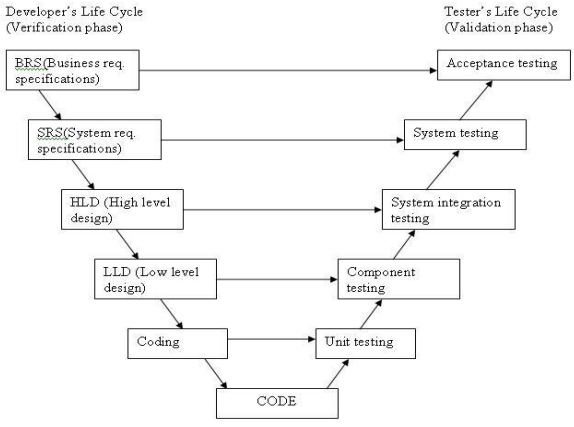


Slika 2.1 *Waterfall model razvoja softvera*

Navedeni model se koristi kada su zahtevi poznati jasni, kada je tehnologija u kojoj se razvija poznata i kada je projekat kratak. U ovakvom načinu razvoja nema interakcije sa klijentima tj. krajnjim korisnicima, tek kada je projekat potpuno gotov može se pokazati korisnicima. U slučaju da tada dođe do ozbiljnih grešaka cena ispravke je obično velika [2].

2.2. V model

Ovaj model je poznat i pod nazivom verifikacija i validacija. Svaka faza mora biti kompletirana pre nego sto počne sledeća faza u razvoju modela. Testiranje se planira paralelno sa razvojem.
Sam model se prikazuje u obliku karakterističnog slova V. Zahtevi se analiziraju na početku modela. Pre nego što se krene sa razvojem kreira se plan sistemskog testiranja. Ovaj plan testiranja se fokusira na specificirane funkcionalnosti koje su definisane u fazi prikupljanja zahteva. Faza dizajna visokog nivoa se fokusira na arhitekturu i dizajn sistema. Pruža pregled rešenja platformi sistema, servisa koje je potrebno implementirati. Paralelno se kreira plan integracionog testiranja kako delovi softvera rade zajedno. Faza dizajna niskog nivoa se fokusira na dizajn softverskih komponenti. Definiše logiku za svaku komponentu sistema, i kreira se klasni dijagram sa svim metodama i relacijama između klasa. Paralelno se kreira plan testiranja komponenti [2].
Na slici 2.2 prikazan je V model razvoja softvera.



Slika 2.2 *V model razvoja softvera*

Faza implementacije je faza gde se odigrava kompletno pisanje koda. Kada je kodiranje završeno putanja izvršavanja kreće desnom stranom V modela, gde se kreirani test planovi stavljaju u upotrebu. Kodiranje se nalazi na dnu V modela, gde se dizajn komponenti pretvara u kod. Jedinično testiranje izvršavaju programeri koji rade na kodu paralelno sa pisanjem koda. Prednosti modela su da je jednostavan i lak za upotrebu, određene aktivnosti testiranja se dešavaju pre kodiranja na primer planiranje i dizajn testova sto štedi dosta vremena, defekti se mogu pronaći rano, dobar za projekte sa jasnim zahtevima. Mane modela su te da je veoma rigidan i nefleksibilan, softver se razvija unutar faze implementacije pa ne postoje rani prototipovi koji se mogu pokazati klijentu, ukoliko se desi promena na pola puta u modelu svi test dokumenti se moraju ažurirati zajedno sa zahtevima.

3. AUTOMATSKO TESTIRANJE

U ovom poglavlju su predstavljeni nivoi testiranja i test dizajn tehnike, objašnjene su samo neke od navedenih.

3.1. Nivoi testiranja

Komponentno testiranje može da se primeni i izolovano od ostatka softvera u zavisnosti od faze procesa razvoja softvera. Zbog nedostatka ostalih komponenti softvera, vrše se različite simulacije razmene podataka [4].

Predmet testiranja mogu da budu funkcionalne karakteristike komponente, ali i nefunkcionalne karakteristike kao što je: ponašanje hardvera (curenje memorije), performanse.
Integraciono testiranje testira prenos informacija između različitih komponenti softvera, interakciju između različitih delova sistema kao što su: operativni sistem, baza podataka i hardver. Integraciono testiranje treba da bude sprovedeno od strane integracionog inženjera ili test inženjera integracije [4].
Sistemsko testiranje uzima u obzir ponašanje krajnjeg produkta/softvera u skladu sa definisanim ciljem projekta koji je bio predmet razvoja. Sistemsko testiranje može da obuhvata testove vezane sa specifične zahteve koje je softver trebao da ispuni, biznis proces, specifično korišćenje softvera, interakciju sa operativnim sistemom ili bazom podataka. Sistemsko testiranje je najčešće završna faza u procesu testiranja koje treba da potvrdi da softver ispunjava sve zahteve pre nego što ser isporuči klijentu, svrha sistemskog testiranja takođe može da bude i pronalaženje što je moguće više grešaka.
Prihvatljivo testiranje treba da odgovori na nekoliko pitanja: „Da li softver može biti isporučen?“, „Šta su, i da li ima rizika?“, „Da li je razvojni tim ispunio sve svoje obaveze?“. Prihvatljivo testiranje najčešće izvršavaju sami klijenti ili korisnici, neretko u saradnji sa test inženjerima. Prihvatljivo testiranje, isto kao i sistemsko testiranje, zahteva okruženje za testiranje koje preslikava okruženje u kojem će softver raditi u produkciji. Glavni cilj prihvatljivog testiranja je da pruži sigurnost i pouzdanost u softver, deo softvera ili neke nefunkcionalne karakteristike kao što su performanse [1].

3.2. Test dizajn tehnike

Statičke tehnike za dizajn testova – Većina statičkih tehnika se koristi prilikom testiranja bilo kakve dokumentacije vezane za softver ili izvorni kod softvera. Predmet testiranja statičkih tehnika mogu biti dizajn dokumenti i dizajn model softvera, funkcionalne

specifikacije i specifikacije zahteva koje softver treba da ispuni.

Tehnike „crne kutije“ – Prva dinamička tehnika koja će biti predstavljena je tehnika „crne kutije“ ili tehnika „unos/rezultat“. Naziv „crna kutija“ potiče od samog načina pristupa softveru, softveru se pristupa kao crnoj kutiji koji ima ulaz i izlaz(rezultat) i nije poznato kako su komponente softvera struktuirane unutar softvera, takođe nije poznato kako softver izračunava krajnji rezultat. Prilikom testiranja softvera tehnikom „crne kutije“, test inženjer se fokusira na čemu softver zaista radi, a ne na koji način softver radi [1].

Tehnike „bele kutije“ – Tehnike „bele kutije“ koriste unutrašnju strukturu softvera za izvršavanje testova. Naziv „bela kutija“ ili „staklena kutija“ potiče od zahteva za poznavanjem načina implementacije softvera i načinom rada softvera. Na primer, tehnika „bele kutije“ testira izvršavanje petlji unutar izvornog koda softvera. Različiti testovi mogu biti dizajnirani da pojedine petlje, unutar softvera izvornog koda, budu izvršene jednom, dva puta ili više puta zaredom. Testiranje tehnikom „bele kutije“ može biti izvršeno bez obzira na funkcionalnost softvera [1].

Tehnike koje se zasnivaju na iskustvu test inženjera – Iskustvo, znanje, veštine test inženjera najviše doprinose testiranju ovom tehnikom. Tehničko i poslovno iskustvo test inženjera je veoma bitno, jer donose različite perspektive u analizi testova i procesu dizajna. Na osnovu prethodnog iskustva sa sličnim softverima, test inženjer može da predvidi kritičan deo softvera koji je veoma bitan za testiranje.

4. Funkcionalnost proračuna kratkih spojeva

Funkcija za proračun struja kratkih spojeva je jedna od jako bitnih funkcija, na osnovu čijih rezultata se bira relejna oprema setuju vrednosti struje zaštite i mnoge druge operacije u mreži.

4.1 Algoritam za proračun kratkih spojeva

Sama funkcija za proračun struja kratkog spoja realizovana je pomoću algoritma koji se sastoji od sledećih koraka:
Svaki proračun kratkog spoja počinje sa dekompozicijom mreže pod kvarom na deo pre kvara i fiktivno takozvano delta kolo. Ovo je primer tj metoda superpozicije. Fiktivno delta kolo je iste topologije kao i originalno kolo osim što je pasivno u celini osim na mestu kratkog spoja gde postoje strujni i naponski izvori, u ostatku kola su izvori pasivni s tim da naponski izvori predstavljaju kratak spoj a strujni izvori predstavljaju prekid [3].
Određivanje matrice ekvivalentnih Tevenenovih impedansi viđenih sa svakog mesta kvara kao i uzajamnih ekvivalentnih Tevenenovih impedansi između mesta kvara. Potrebno je računati ove impedanse za sve tri faze ukoliko se radi sa trofaznim balansiranim mrežama ili se računaju samo za jednu fazu ukoliko mreža nije balansirana. Zbog ovoga se svi elementi na mreži (transformatori, generatori, prekidači i ostali) zamenjuju sa ekvivalentnim impedansama. Impedansa transformatora sa regulacionom sklopkom se računa uzimajući u obzir stvarni položaj regulacione sklopke [3].
Definišu se opcije za selektovani tip kratkog spoja, pri tome se setuje period proračuna, minimalna i maksimalna struja kratkog spoja, snaga mreže, otpor impedanse na mestu kvara, tip kvara [3].
Računanje struje kratkog spoja i kompenzacija struje koristeći kanonički model [3].
Računanje stanja delta kola. Za ovu svrhu se koristi modifikovani Shirmohammadi -ev algoritam za proračun struje. U ovom algoritmu svi elementi mreže su zamenjeni sa svojim impedansama. Potrošači su izostavljeni i mesto kvara je modelovano kao idealni strujni izvor, i predstavlja jedini generator u kolu. Posmatrano kolo je tada linearno i može se rešiti direktno bez primene iteracija. Takvim postupkom je potrebna samo jedna iteracija tj. proračun kako bi se rešilo delta kolo [3].

Nakon proračuna za delta kolo, celokupno stanje mreže u kvaru se dobija superpozicijom kola pre kratkog spoja i fiktivnog delta kola [3].

5. AUTOMATIZACIJA PRORAČUNA KRATKIH SPOJEVA

Razvoj testova za potrebe automatizacije realizovan je u okviru okruženja PyCharm. Koristi se za programiranje u pajtonu. Razvojno okruženje pruža analizu koda, grafički debager, integrisanu jedinicu tastera i podržava veb razvoj. Dobra osobina okruženja jeste ta da podržava rad sa json fajlovima. Samo razvojno okruženje ima instalirane pakete koji mu omogućavaju rad sa json fajlovima. Json je tip formata fajla koji se koristi za čuvanje i razmenu podataka. Jednostavan je za rad jer je korišćenje veoma jednostavno, lak je upis i čitanje. U okviru same realizacije rada json fajl pored standradnih instalacionih putanja, za potrebe rada sadrži i ključeve koji su od interesa. Razlog zbog čega se koristi u realizaciji rada je taj da ukoliko se desi neka promena u konfiguraciji, ključevi elemenata se promene izmena se veoma lako ostvaruje menjajući ključ u json-u umesto izmene automatskih testova, ostvaruje se ušteda na vremenu potrebnom za održavanje testova.
Na slici 5.1 prikazana je struktura json fajla.

```
{
    "Rail Booking": {
        "reservation": {
            "ref_no": 1234567,
            "time_stamp": "2016-06-24T14:26:59.125",
            "confirmed": true
        },
        "train": {
            "date": "07/04/2016",
            "time": "09:30",
            "from": "New York",
            "to": "Chicago",
            "seat": "57B"
        },
        "passenger": {
            "name": "John Smith"
        },
        "price": 1234.25,
        "comments": ["Lunch & dinner incl.", "\"Have a nice day!\""]
    }
}
```

Slika 5.1 *Prikaz strukture json fajla*

Kako bi problematika automatizacije proračuna kratkih spojeva bila rešena, sam koncept realizacije osmišljen je tako da su automatski testovi podeljeni u dve celine: testovi zaduženi za postavku seme za stanice od interesa u kojima se simuliraju kratki spojevi, test koji je zadužen za proračun kratkih spojeva nad već pripremljenim šemama i upis vrednosti u željeni fajl.

U okviru testova zaduženih za postavku šema neophodno je ispuniti par zahteva: postaviti sve regulacione transformatore u neutralni položaj, razvezati sve potrošače i kondezatore sa sabirnica. Da bi realizovali postavku sistema potrebno je sve otvorene elemente na postojećoj šemi vratiti u zatvoreni položaj.

To se čini zbog toga što su na taj način zatvoreni strujni krugovi od izvora napajanja do potrošača i jednostavnim manipulacijama, mogu se raskačiti svi neželjeni elementi. Kako bi izbegli pristupanju svim elementima jer je cilj imati sve elemente u stanju zatvoren na datoj stanici iskorišćena je pomoć trace funkcionalnosti koja se nalazi u sklopu samog softvera.

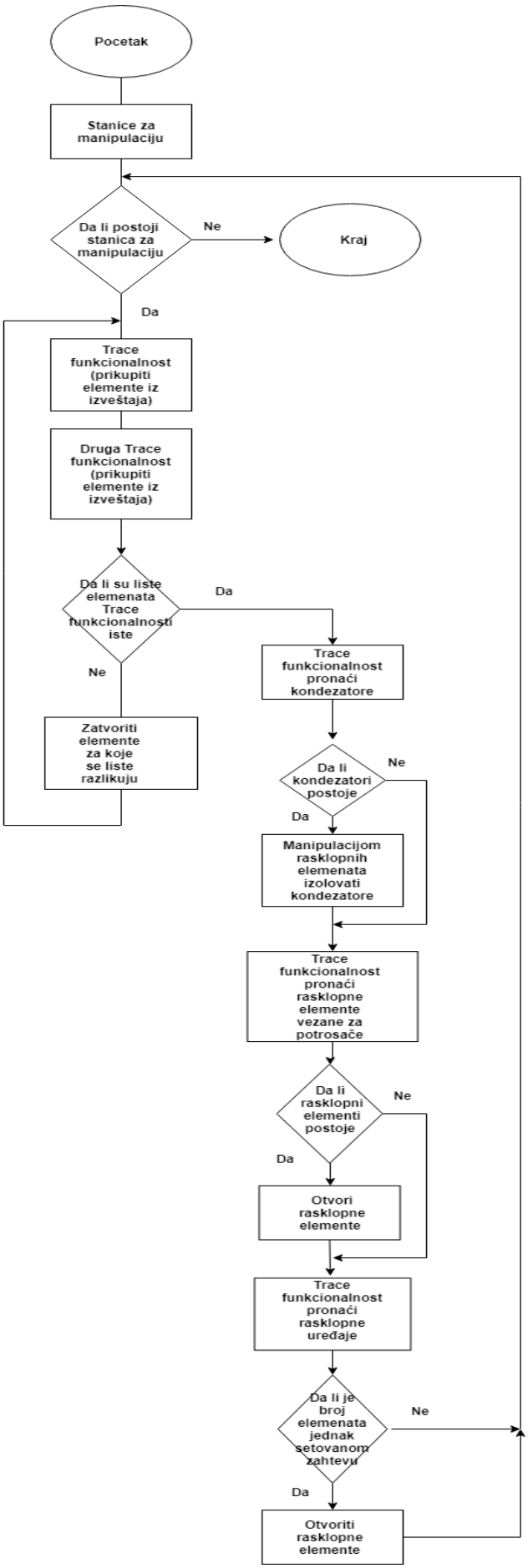
Navedena problematika rešena je korišćenjem dve trace funkcionalnosti. Trace funkcionalnosti kao startnu lokaciju koriste izvore napajanja i prikupljaju sve elemente nadole od izvora napajanja, za izabrani smer funkcionalnosti nadole.

Korišćenjem funkcionalnosti i upoređivanjem elemenata prikupljenih ih izveštaja funkcionalnosti nakon filtriranja sadržaja izveštaja kako bi bilo izbegnuto manipulisanje sa nepotrebnim elementima prikupljeni su elementi od interesa.

Razlika sadržaja dve funkcionalnosti predstavlja listu elemenata koji se nalaze u stanju otvoren i nad njima je neophodno manipulisati. Samo stanje tih elemenata neophodno je promeniti. Ukoliko je zateknuto stanje takvo da su svi elementi u stanju zatvoren pri prvom prolasku algoritma ne postoji potreba za manipulacijom nad elementima, ukoliko postoje elementi za manipulaciju menja im se stanje i ponovo se ponavlja isti postupak sa pokretanjem dve trace funkcionalnosti ako tada ne postoji razlika elemenata postupak se prekida ako naprotiv i dalje postoje elemenati za manipulaciju njima će se manipulisati i tako dok se svi elementi ne postave u stanje zatvoren.

Nakon ovakve postavke željenog stanja primenom trace funkcionalnosti pronalaze se svi kondezatore koji postoje i razvezuju se od ostatka šeme, isto se izvršava i za potrošače koji se odvajaju od ostatka šeme. Ovakvim pristupom pripremljena je šema od interesa za proračun kratkog spoja. Na slici 5.2 prikazan je blok dijagram algoritma za pripremu šema od interesa za proračun kratkih spojeva.

Test kojim se realizuje proračun kratkih spojeva koristi postavljeno stanja sistema ostvareno testom za postavku stanja. Stanje sistema se učitava i pronalaze se lokacije na kojima se simulira proračun kratkog spoja. Te lokacije se setuju u opcijama funkcionalnosti kratkog spoja, podešavaju se parametri funkcionalnosti koji su prikupljeni iz unapred definisanog csv fajla, računaju se vrednost struja kratkog spoja, date vrednosti se prikupljaju iz izveštaja funkcije. Sve prikupljene vrednosti se upisuju u dati csv fajl na tačno definisanu lokaciju, za dati tip kratkog spoja, željenu stanicu i postavku sistema. Isti proces se ponavlja za svaku stanicu ponaosob. Na slici 5.3 prikazana je topološka struktura šema stanica od interesa.

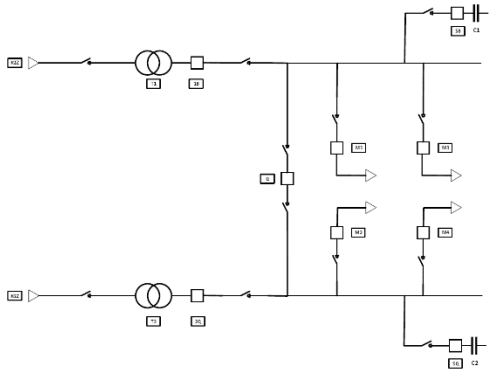


Slika 5.2 *Blok dijagram algoritma za postavku šeme za stanice od interesa*

5.1. Prednosti i mane automatskog testiranja vezane za opisani rad

Prednosti ostvarene automatizacijom proračuna: izbegnuto manuelno izvršavanje testova za proračun kratkih spojeva, izbegnuto popunjavanje elemenata kroz json, auto-

matizovana postavka šeme i priprema sistema za proračun bez obzira na zatečeno stanje sistema, jednostavnom manipulacijom se broj stanica za proračun može proširiti, izbegnute izmene na šemama korišćenjem metoda za generisanje pomeraja, testiranje performansi odziva softvera.



Slika 5.3 *Topološka struktura šema stanica od interesa*

Mane ispoljene automatizacijom testova: testovi za pripremu stanja sa porastom broja stanica se vremenski produžavaju, mogu trajati i nekoliko desetina sati u zavisnosti od broja stanica za pripremu, usporavanje testnog sistema.
Glavna svrha napisanih testova jeste održavanje. Zbog velikog broja manipulacija nad stanicama od interesa vreme izvršavanja testova vremenski će oscilirati.

6. ZAKLJUČAK

Proračun kratkih spojeva realizovan je podelom automatskih testova na dve grupe. Prva grupa je skup od dva testa koji priprema šemu na stanicama od interesa u okviru posmatranog sistema. Drugu grupu predstavlja test koji radi proračun struja kratkih spojeva za stanice od interesa koristeći sačuvane postavke sistema kreiranih testovima zaduženim za pripremu šema za proračun. Test vrši i upis proračunatih vrednosti u željeni fajl. Postoje dva dodatna testa koji rade verifikaciju fajla, ukoliko neki podaci nisu adekvatno popunjeni ili su popunjeni pogrešno, test skreće pažnju kako bi ispravili pogrešno, kao i test za proveru upisanih proračunatih struja spram definisanog kriterijuma.

Ukoliko vrednosti odstupaju podaci koji su nevalidni biće obeleženi. Vremenska zavisnost testova za pripremu šema veća je ukoliko se proširi broj stanica za manipulaciju. Mogući nastavci rada su primena postojećih testova za postavku šema sa manjim ili većim izmenama kako bi se automatizovale i ostale funkcionalnosti vezane za proračune, optimizacija datog rešenja ili pronalaženje boljeg kako bi se izvršavanje postavke šema ubrzalo i samim tim i testovi.

7. LITERATURA

[1] R.Black, Erik Van Veenendaal, D.Graham: *Foundations of Software Testing 3rd Edition*, South – Western, 2009.

[2] Miodrag Živković: Testiranje Softvera, Beograd 2018.

[3] Dragan Popović, Duško Bekut, Valentina Treskanica: Specijalizovani DMS Algoritmi, Novi Sad 2004.

[4] Miloš Kajkut: Automatsko Testiranje CVC DMS funkcije implementirane u aDms softver, Novi Sad 2018.

Kratka biografija:

Milovan Adamović rođen je u Šapcu 1993. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva - Energetska elektronika i Električne mašine odbranio je 2017. god. Iste godine upisao se na master studije na istom smeru.