

**UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET**

Katarina Antešević

**IMPLEMENTACIJA RADNOG OKVIRA ZA
SOFTVERSKO TESTIRANJE NAMJENSKIH
SISTEMA NA LINUX PLATFORMI**

diplomski rad

Banja Luka, april 2024.

**Tema: IMPLEMENTACIJA RADNOG OKVIRA ZA
SOFTVERSKO TESTIRANJE NAMJENSKIH
SISTEMA NA LINUX PLATFORMI**

**Ključne riječi: Radni okvir za testiranje
Robot framework
TTE Switch Space 3U cPCI**

**Komisija: doc. dr Željko Ivanović, predsjednik
doc. dr Mihajlo Savić, mentor
Srđan Popić, ma, član**

**Kandidat:
Katarina Antešević**

UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET
KATEDRA ZA RAČUNARSTVO I INFORMATIKU

Predmet: Razvoj radnog okvira za testiranje uređaja TTE Switch Space
3U cPCI

Tema: Implementacija radnog okvira za softversko testiranje namjenskih
sistema na Linux platformi

Zadatak: Dati uvod u oblast testiranja softvera. Obraditi nivoe, tipove i tehnike
testiranja. Detaljno opisati radne okvire za testiranje namjenskih
sistema. Analizirati i uporediti dostupna komercijalna i nekomercijalna
rješenja.

Praktični dio treba da obuhvati jedno rješenje implementacije testnog
radnog okvira u Linux okruženju i demonstrira upotrebu
implementiranog rješenja na primjeru izvršavanja softverskih testova.

Mentor: doc. dr Mihajlo Savić

Kandidat: Katarina Antešević (1187/15)

Banja Luka, april 2024.

SADRŽAJ

1. UVOD.....	1
2. POJAM TESTIRANJA SOFTVERA	2
2.1 Osnovni pojmovi	2
2.2 Principi testiranja	4
2.3 Ciljevi testiranja.....	5
2.4 Nivoi testiranja	6
2.4.1 Testiranje komponenti (Unit testiranje)	6
2.4.2 Integraciono testiranje.....	7
2.4.3 Testiranje sistema	7
2.4.4 Testiranje prihvatljivosti.....	8
2.5 Tehnike testiranja	9
2.5.1 Statičke tehnike testiranja.....	10
2.5.2 Dinamičke tehnike testiranja	10
2.6 Tipovi testiranja	12
2.6.1 Funkcionalno testiranje (testiranje crne kutije).....	12
2.6.2 Nefunkcionalno testiranje	13
2.6.3 Strukturalno testiranje (testiranje bijele kutije).....	14
2.6.4 Testiranje koje se tiče promjena načinjenih u softveru.....	14
3. VEZA IZMEĐU MODELA RAZVOJA SOFTVERA I TESTIRANJA SOFTVERA.....	15
3.1 Model vodopada	15
3.2 V model	16
3.3 Iterativni model.....	17
3.4 Agilni model.....	18
3.5 Spiralni model	18
4. RADNI OKVIRI ZA TESTIRANJE	20
4.1 Tipovi radnih okvira za testiranje.....	20
4.1.1 Radni okviri bazirani na modulima	20
4.1.2 Radni okviri koji se baziraju na arhitekturi biblioteka.....	21
4.1.3 Radni okviri koji se baziraju na podacima.....	21
4.1.4 Radni okviri koji se baziraju na ključnim riječima	22
4.1.5 Radni okviri koji se baziraju na ponašanju.....	22
4.1.6 Hibridni radni okviri	23
4.2 Primjeri komercijalnih radnih okvira	23

4.2.1	Unified Functional Testing (UTF)	23
4.2.2	VectorCAST	24
4.3	Primjeri nekomercijalnih radnih okvira	25
4.3.1	GaugeFramework	25
4.3.2	Robot framework	26
5.	OPIS PRAKTIČNOG DIJELA DIPLOMSKOG RADA.....	27
5.1	Vremenski osjetljiv Ethernet (eng. Time Triggered Ethernet).....	27
5.2	Vremenski osjetljiv switch (eng. Time Triggered Switch).....	28
5.2.1	Mrežne funkcije.....	31
5.2.2	TTE-Switch Space 3U cPCI Firmware	32
5.2.3	Flash memorija.....	33
5.2.4	Primjena uređaja TTE-Switch Space 3U cPCI.....	35
5.3	Robot framework	36
5.3.1	Testne biblioteke	37
5.3.2	Varijable	39
5.3.3	Ključne riječi	40
5.3.4	Testni paketi	41
5.3.5	Testni slučajevi	42
5.3.6	Tagovi.....	43
5.3.7	Pokretanje procesa testiranja	45
5.4	Testiranje TTE-Switch Space 3U cPCI Firmware-a	46
5.4.1	report_data_save.....	48
5.4.2	icmp_request_check.....	48
5.4.3	tftp_download.....	50
5.4.4	safety_relevant_function	51
5.4.5	tftp_upload_default_files.....	51
5.4.6	tftp_upload_user_files	52
5.4.8	snmp_dsus_pin_state_check	55
5.4.9	memory_content_check	56
5.4.10	Izvještaji procesa testiranja	57
6.	ZAKLJUČAK	62
	Literatura.....	64

Uz rad je priložen CD.

1. UVOD

Softverski sistemi su postali sastavni dio svakodnevnog života svih nas, počevši od računarskih igara, preko softvera koji se koristi u obrazovne, medicinske svrhe, te softvera koji se koriste za različite poslovne svrhe. Većina korisnika se u nekom trenutku susrela sa softverom koji nije radio na očekivani način ili nije ispunio određena očekivanja. Korištenje takvog softvera može dovesti do različitih problema kao što su gubitak vremena, novca, reputacije, klijenata, može izazvati tjelesne povrede korisnika, pa dovesti i do smrtnih ishoda.

Zbog svega navedenog, testiranje softvera je veoma bitan postupak i to je način da se utvrdi da li softver ispunjava sva očekivanja korisnika i da li je softver siguran za korištenje.

Pogrešno je pretpostaviti da testiranje obuhvata samo izvršavanje testova, tačnije korištenje softvera i provjeru da li radi na odgovarajući način. Testiranje softvera je proces koji uključuje različite aktivnosti, a izvršavanje testova je samo jedna od njih. Proces testiranja softvera uključuje aktivnosti kao što su planiranje testiranja, analizu softvera koji se testira, dizajn i implementaciju testova, prikupljanje rezultata testova i evaluaciju kvaliteta testnih objekata [1].

Neki testovi uključuju izvršavanje komponente sistema ili sistema koji se testira i to je dinamički tip testiranja. Postoji i drugi tip testiranja koji ne uključuje takvo izvršavanje i to je statički tip testiranja. Testiranje uključuje i pregled zahtjeva koje softver treba da ispuni, različite svrhe za koje se može iskoristiti, ali i testiranje samog koda. Takođe, testiranjem se treba utvrditi da li softver ispunjava očekivanja krajnjih korisnika koji će ga koristiti, ali i očekivanja naručioca softvera.

U drugom poglavlju ovog rada obrađena su sljedeća pitanja: koji su osnovni pojmovi koji se tiču procesa testiranja, kao i koji su principi i ciljevi testiranja. Zatim se upoznajemo sa nivoima, tehnikama i tipovima testiranja.

U trećem poglavlju predstavljeno je kako je u stvari proces testiranja povezan sa drugim procesima koji su sastavni dio razvoja softvera, te kakva je veza između modela razvoja softvera koji je odabran i testiranja. Takođe, upoznajemo se sa parametrima koje treba uzeti u obzir kada tim developera bira model razvoja softvera.

Četvrta glava nudi pregled različitih okvira za testiranje i pregled različitih tipova radnih okvira koje tim testera može da odabere. Predstavljene su prednosti i mane različitih tipova, koje tim testera treba da sagleda i prouči prije nego se odluči za specifični radni okvir koji će se koristiti i koji će testnom timu biti najviše od koristi. Takođe su opisana i postojeća komercijalna i nekomercijalna rješenja koja se mogu iskoristiti za automatizaciju procesa testiranja, što je od posebnog interesa za ovaj diplomski rad.

Peta glava sadrži pregled praktičnog dijela diplomskog rada. Opisano je koji uređaj se testira, te koja je njegova svrha, koje su mu mogućnosti i na koji način je realizovana automatizacija procesa testiranja.

Na kraju rada je dat zaključak.

2. POJAM TESTIRANJA SOFTVERA

2.1 Osnovni pojmovi

Postoje različiti razlozi zašto softver ne radi na ispravan način. Ako korisnik ne koristi softver na očekivani način, logično je očekivati da se i sam softver neće ponašati onako kako je očekivano. Međutim, osobe koje dizajniraju i učestvuju u implementaciji softvera mogu napraviti greške. Ako dođe do toga, to znači da će softver imati neke nedostatke. Ti nedostaci se nazivaju greške, defekti ili bagovi [2]:

- Greška (eng. error) – kreira je čovjek u toku pravljenja specifikacije ili realizacije softvera.
- Defekt (eng. defect) – defekt predstavlja situaciju u kojoj softver ne radi u skladu sa očekivanim zahtjevima i rezultat izvršavanja softvera ne odgovara očekivanom rezultatu. Posljedica je greške.
- Bag (eng. bug) – bag predstavlja grešku u sistemu koja označava da softver nije implementiran na osnovu unaprijed određenih softverskih zahtjeva.
- Testni scenario – opisuje neku funkcionalnost sistema koja se može testirati.

Testni slučaj predstavlja skup određenih aktivnosti koje se izvršavaju i čije izvršavanje ima cilj da utvrdi da li softver sadrži bilo kakve greške, da li ispunjava sve softverske zahtjeve i da li se može koristiti za svrhu za koju je namijenjen. Testni slučaj se može posmatrati kao uređena trojka (Ulaz, Stanje, Izlaz) [3]:

- Ulaz – skup ulaznih podataka
- Stanje – stanje sistema kojem se predaje skup ulaznih podataka
- Izlaz – očekivani izlaz sistema

Testni paket (eng. test suite) predstavlja logički grupisanu kolekciju testnih slučajeva čija je svrha da ispita neku funkcionalnost softvera [1].

Primjer jednog test paketa i testnih slučajeva koje sadrži:

Testni paket koji predstavlja provjeru mogućnosti kupovine proizvoda na nekoj e-commerce web aplikaciji¹ [4]:

1. Prvi testni slučaj: Prijava na aplikaciju
2. Drugi testni slučaj: Odabir željenog proizvoda i dodavanje u korpu
3. Treći testni slučaj: Kupovina proizvoda
4. Četvrti testni slučaj: Odjava

Pri pokretanju pomenutog testnog paketa, testni slučajevi će se izvršavati jedan za drugim, redoslijedom koji je prethodno određen, kao što je prikazano.

¹ E-commerce web aplikacija – web aplikacija koja omogućava korisnicima da kupuju, ali i nude usluge ili bilo koji tip robe i uključuje online transakciju novca.

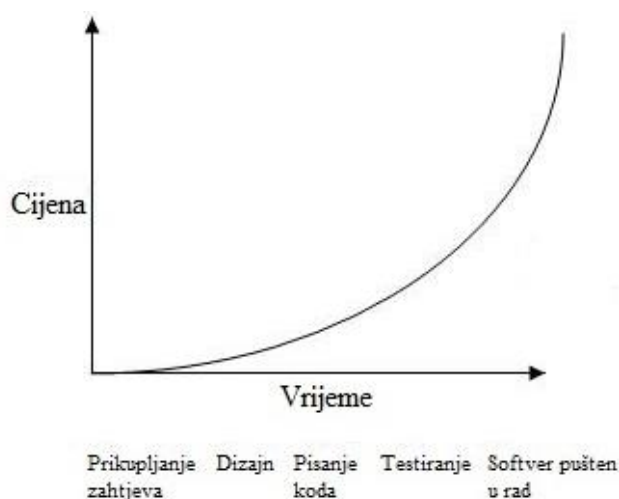
Kada razvijamo softver, uvijek postoji šansa da postoje greške, a sa porastom kompleksnosti softvera, povećava se i vjerovatnoća da greške zaista postoje. Treba imati na umu da je nemoguće kreirati softver koji ne sadrži ni jednu grešku, zato i koristimo različite tehnike testiranja koje nam omogućavaju da otkrijemo i otklonimo postojeće greške. Pored kvaliteta i pouzdanosti, koje želimo osigurati, još jedan od bitnih razloga zbog kojeg vršimo testiranje je smanjenje troškova koji bi nastali kada bi se greške otkrile tek kasnije, nakon puštanja softvera u rad.

Često se desi da softver bude pušten u rad, ali zbog prisustva grešaka ne radi na ispravan način ili ne ispuni očekivanja zainteresovanih strana. Greške se mogu desiti zbog različitih razloga [1]:

- Kratkih vremenskih rokova koji su postavljeni za isporuku softvera
- Slučajne greške koje napravi developer
- Neiskustvo i neodgovarajuće vještine učesnika koji rade na projektu
- Manjak komunikacije ili neadekvatna komunikacija između učesnika u izradi softvera
- Korištenje novih, nepoznatih tehnologija
- Kompleksnost koda, dizajna ili arhitekture softvera
- Uslova okoline u kojima se izvršava softver. Npr: radijacija ili elektromagnetno polje može proizvesti različite defekte na firmware-u² [5]

Greške pri izvršavanju softvera se mogu desiti zbog nepravilnog korištenja softvera od strane krajnjih korisnika: mogu unijeti pogrešnu ulaznu vrijednost ili pogrešno razumjeti rezultat izvršavanja nekog dijela softvera. Takođe, neočekivano ponašanje softvera se može desiti jer neko namjerno pokušava izazvati takvo ponašanje, tačnije, ima neke maliciozne namjere.

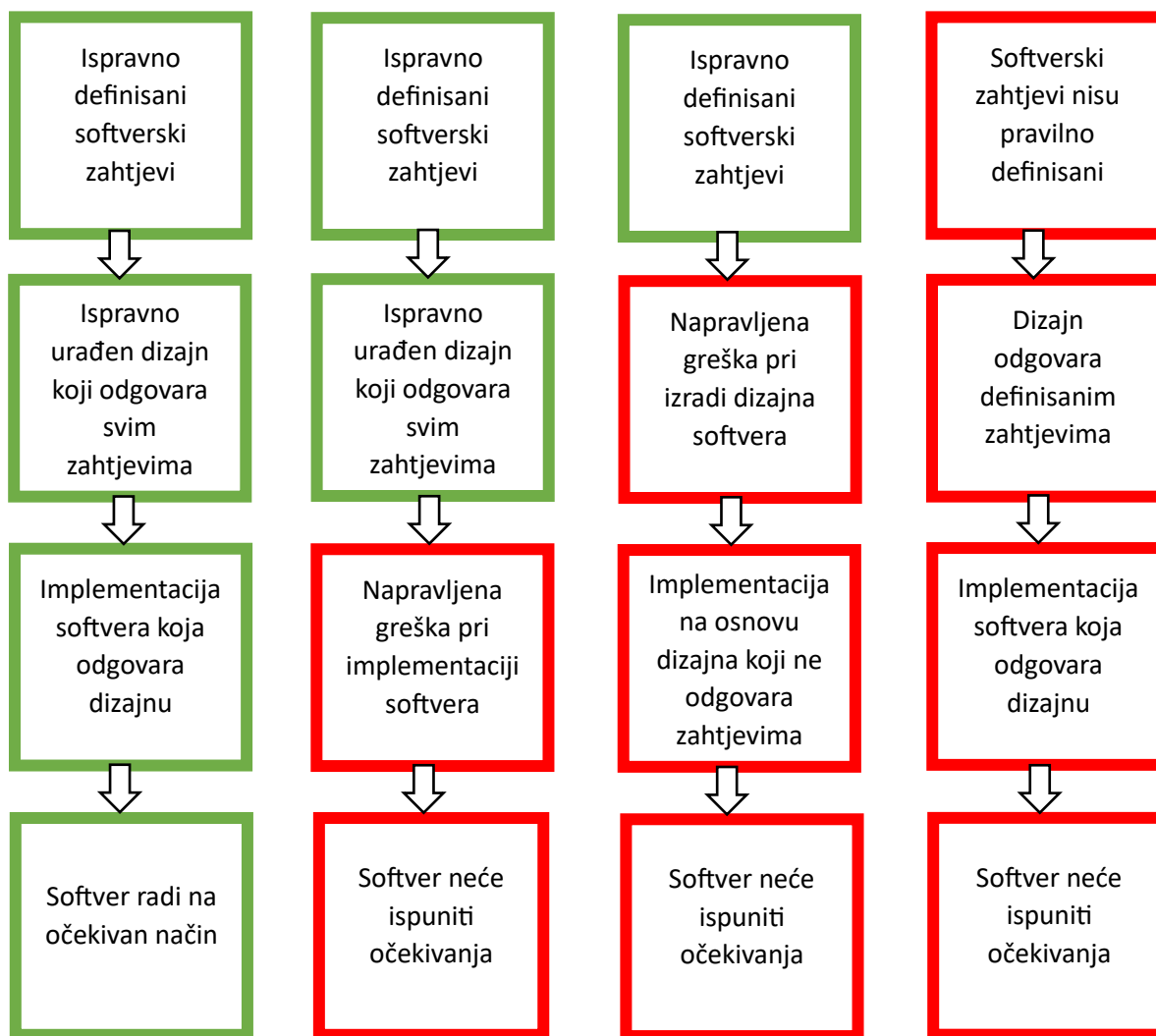
Važno je uzeti u obzir posljedice koje imaju pomenute greške. Cijena ispravljanja grešaka vrtoglavo raste što se nalazimo dalje u razvoju softvera, što znaci da je bolje pronaći greške u ranijim fazama razvoja softvera (Slika 1).



Slika 1. Odnos cijene otklanjanja grešaka i faze razvoja softvera

² Firmware – tip softvera koji omogućava kontrolisanje određenog hardvera. Upravlja osnovnim funkcijama uređaja kao što su pokretanje hardvera, konfiguracija sistema i upravljanje napajanjem.

Greške koje dovode do defekata se mogu desiti u različitim fazama razvoja softvera, pa imamo sljedeće situacije [2]:



2.2 Principi testiranja

Osnovni principi testiranja [1]:

- Testiranje pokazuje prisustvo grešaka, a ne njihovo odsustvo. Testiranjem se smanjuje vjerovatnoća da postoje neotkrivene greške, ali čak i ako testiranje ne pokazuje nikakve greške, to ne znači da one nisu prisutne.
- Testiranje svih mogućih kombinacija ulaza i preduslova nije izvodljivo, osim u nekim trivijalnim slučajevima. Zato je potrebno koristiti odgovarajuće tehnike testiranja i utvrditi šta su prioriteti za testiranje.
- Testiranje u ranim fazama može uštediti dosta novca i vremena. Da bismo ovako pronašli greške, i dinamičke i statičke testne aktivnosti trebalo bi izvršavati od najranijih faza razvoja softvera. Takođe, ranim testiranjem možemo izbjeći potrebu za skupim izmjenama u nekim kasnijim fazama razvoja.

- Greške obično nisu same, tj. mali broj modula najčešće sadrži većinu grešaka.
- Ako iste testove izvršavamo mnogo puta, nakon nekog vremena oni uopšte neće pokazivati postojanje grešaka. Da bi se otkrile nove greške, postojeći testovi i testni podaci se trebaju mijenjati, ali i kreirati novi.
- Testiranje se izvršava na različite načine, kada su u pitanju različiti softverski proizvodi. Npr. neki industrijski kontrolni sistem³ visokog rizika se testira drugačije nego neka e-commerce mobilna aplikacija.
- Odsustvo grešaka je samo prividno. Nije realno očekivati da tester i mogu izvršiti sve moguće testove i naći sve moguće greške. Takođe, zabluda je se pronalaskom i ispravljanjem grešaka osigurava da sistem radi tačno onako kako treba. Npr. temeljno testiranje svih zahtjeva i ispravljanje grešaka bi ipak moglo kreirati sistem koji ipak ne ispunjava sve potrebe i očekivanja, ili nije dorastao sličnim sistemima.

2.3 Ciljevi testiranja

Različiti su ciljevi koji se trebaju ostvariti testiranjem softvera [1]:

- Evaluacija softverskih zahtjeva, različitih načina na koji se softver može koristiti, dizajna softvera i samog koda
- Verifikacija ispunjenja nekih specifičnih zahtjeva
- Validacija da li testni objekat, tačnije komponenta sistema ili sam sistem, radi u skladu sa očekivanjima krajnjih korisnika ili naručioca softvera
- Pronalazak defekata
- Slanje korisnih informacija naručiocima softvera, tako da oni mogu donositi informisane odluke
- Pregled ugovora, legalnih i različitih regulatornih zahtjeva i provjera da li je softver u skladu sa njima
- Dokumentovanje toga šta je testirano, na koji način je testirano i kakav je rezultat testiranja

Proces testiranja omogućava da se utvrdi kvalitet softvera u smislu broja defekata koji su pronađeni, broja testova koji su pokrenuti i pokrivenosti testova [2].

Da bi softver bio kvalitetno testiran, testovi moraju biti dobro dizajnirani. Tačnije, nekorektno dizajnirani testovi možda neće otkriti neke prisutne defekte, a korektno dizajnirani mogu otkriti različite defekte koji su prisutni i omogućavaju programerima da ih isprave i da softver bude siguran za korištenje. Nakon detektovanja i otklanjanja defekta, kvalitet softvera raste, u slučaju da su defekti pravilno otklonjeni [2].

Cilj programera je da dostave naručiocu softver koji odgovara dogovorenoj specifikaciji zahtjeva. A da bi to bilo moguće, neophodno je da ta specifikacija bude ispravna, to jest, sistem mora ispuniti sve zahtjeve da bi se očekivanja naručioca ispunila [2]. Da bude utvrđeno da li sistem ispunjava zahtjeve koristi se validacija i verifikacija.

³ Industrijski kontrolni sistem – tip sistema koji prati i kontroliše industrijske procese. Koristi se u različitim industrijama kao što je naftna, hemijska, farmaceutska, autoindustrija, industrija hrane i pića i mnogim drugim. Svaki sistem posluje drugačije u skladu sa potrebama date industrije.

IEEE-STD-2012⁴ standard definiše verifikaciju na sljedeći način:
“Verifikacija predstavlja testiranje sistema u cilju utvrđivanja da li sistem ispunjava sve potrebne zahtjeve u nekoj fazi razvoja.” [6]

Nakon što je proces specifikacije gotov, slijedi proces verifikacije. Verifikacija odgovara na pitanje: “Da li kreiramo proizvod na pravi način?” [6]. Predstavlja statičko testiranje softvera i ne podrazumijeva izvršavanje koda. Izvršava se prije validacije i podrazumijeva pregled i provjeru različitih dokumenata i fajlova, dizajna softvera, koda i ovaj tip testiranja izvršavaju ljudi. Važno je dodati da je verifikacija bazirana na subjektivnom mišljenju osobe koja izvršava verifikaciju i sam rezultat verifikacije zavisi od osobe do osobe [2].

IEEE-STD-2012 standard definiše validaciju na sljedeći način:
“Validacija je aktivnost koja osigurava da softver ispunjava sva očekivanja i potrebe naručioca” [6].

Validacija odgovara na pitanje: “Da li kreiramo pravi proizvod?” [6] i odvija se nakon što je softver kreiran i osigurava da naručioci dobiju ono što su naručili. Validacija predstavlja dinamički tip testiranja i uključuje izvršavanje koda i izvršava se nakon verifikacije [2].

2.4 Nivoi testiranja

Nivoi testiranja predstavljaju testne aktivnosti koje su organizovane zajedno i kojima se upravlja kao cjelinom. Svaki nivo testiranja predstavlja instancu testnog procesa i sastoji se od sljedećih aktivnosti (planiranje, monitoring i kontrole, analiza, dizajn, implementacija, izvršavanje) [1].

Nivoi testiranja [1]:

- Testiranja komponenti (Unit testiranje)
- Integraciono testiranje
- Testiranje sistema
- Testiranje prihvatljivosti

2.4.1 Testiranje komponenti (Unit testiranje)

Testiranje komponenti ili Unit testiranje se fokusira na komponente koje se mogu testirati pojedinačno, kao što su moduli, klase, objekti i može se odvijati nezavisno od testiranja cjelokupnog sistema. Može se odvijati i paralelno sa testiranjem sistema ili nakon toga.

Testiranje komponenti može obuhvatiti testiranje funkcionalnih ili nefunkcionalnih karakteristika (memory leak, performanse) [1]. Kada se vrši testiranje komponenti omogućen je pristup kodu i to uz pomoć razvojnog okruženja (npr. Unit testni framework⁵). Testiranje vrši programer koji je napisao kod, ali može vršiti neki drugi programer [7]. Najčešći je slučaj da programer napiše kod za određenu komponentu, a nakon toga kreira test za istu i izvrši ga.

⁴ The Institute of Electrical and Electronics Engineers Standard 2012 — standard koji je razvijen 2012. godine od strane Instituta električnih i elektronskih inženjera. IEEE Standard predstavlja dokument kojim se utvrđuju tehničke specifikacije i procedure osmišljene tako da maksimalno povećaju pouzdanost materijala, proizvoda, metoda i usluga koje se svakodnevno koriste.

⁵ Unit testni framework – skup alata i biblioteka dizajniranih da omoguće pisanje i izvršavanje unit testova. Nude testerima okruženje za kreiranje, upravljanje i pokretanje testova, da bi se utvrdilo da li svaka komponenta sistema radi kako je i predviđeno.

Cilj testiranja komponenti jeste da se pronade što veći broj defekata u komponentama, tako da se ne nailazi na njih na višim nivoima testiranja.

2.4.2 Integraciono testiranje

Integraciono testiranje testira interakcije između različitih komponenti i između različitih dijelova sistema kao što su operativni sistem, file sistem, hardver ili interface-i između sistema⁶. Kada govorimo o integracionom testiranju, možemo razlikovati dva nivoa [1]:

- Integraciono testiranje komponenti - fokusira se na interakcije i interface-e između integriranih komponenti. U ovom slučaju se testiranje izvršava nakon testiranja komponenti i uglavnom je automatizovano.
- Integraciono testiranje sistema - fokusira se na interakcije između sistema ili mikroservisa⁷. Ovaj tip testiranja se može izvršavati nakon testiranja sistema ili paralelno s njim

Fokus integracionog testiranja komponenti i integracionog testiranja sistema treba biti sama integracija. Ukoliko se testira integracija dva modula, fokus treba biti na komunikaciji između ta dva modula, a ne na funkcionalnosti svakog od njih, jer bi se to trebalo testirati prilikom testiranja komponenti (poglavlje 2.4.1). Ukoliko se testira integracija dva sistema, fokus treba biti na njihovoj komunikaciji , a ne na funkcionalnostima sistema, jer se njihove funkcionalnosti testiraju prilikom testiranja sistema (poglavlje 2.4.3) [1].

Integraciono testiranje komponenti obično vrše osobe zadužene za razvoj komponenti koje se testiraju, a integraciono testiranje sistema tester. U idealnom slučaju, tester koji vrše integraciono testiranje sistema trebaju biti upoznati sa sistemom i imati znanja o samoj arhitekturi sistema koji se testira.

Dobra je praksa planirati integraciono testiranje i kreirati testove prije kreiranja komponenti sistema, jer na taj način je moguće implementirati sistem redoslijedom koji će omogućiti najefikasnije testiranje.

2.4.3 Testiranje sistema

Testiranje sistema se bavi mogućnostima i ponašanjima sistema ili proizvoda. Testiranje sistema najčešće predstavlja finalni test koji razvojni tim izvršava da bi utvrdio da li sistem ispunjava sve potrebne zahtjeve i da li radi na očekivani način, a cilj je da pronade što više defekata. Ovaj tip testiranja najčešće izvršava poseban, nezavisan tim testera koji menadžeru projekta javljaju rezultate izvršavanja testova [8].

Testiranje sistema utvrđuje i funkcionalne i nefunkcionalne zahtjeve koje ispunjava sistem.

⁶ Interface između različitih dijelova sistema – veza između dva odvojena dijela sistema koja im omogućava da razmjene informacije. Razmjena informacija se može vršiti između softvera, hardvera, perifernih uređaja, čovjeka ili kombinacije navedenog.

⁷ Mikroservis ili mikroservisna arhitektura – arhitektonski pristup u kojem se jedna aplikacija sastoji od manjih nezavisnih komponenti, tačnije servisa, koji imaju svoje odgovornosti.

Tipični nefunkcionalni testovi provjeravaju performanse sistema i pouzdanost. Testeri tokom testiranja mogu naići na nepotpune i nedokumentovane zahtjeve [7]. Za verifikaciju ispunjenih funkcionalnih zahtjeva može se kreirati tabela u kojoj će biti prikazano šta je ispunjeno, a šta ne.

Greške u specifikaciji zahtjeva mogu dovesti do nerazumijevanja onoga sto se od sistema očekuje i zato je veoma bitno da testeri budu prisutni u najranijim fazama razvoja proizvoda [1].

2.4.4 Testiranje prihvatljivosti

Kada testni tim izvrši testiranje sistema i koriguje greške, ako je naišao na njih, sistem će biti dostavljen krajnjem korisniku ili naručiocu sistema za testiranje prihvatljivosti [1].

Testiranje prihvatljivosti se fokusira na ispitivanje da li sistem odgovara svojoj svrsi. Može se desiti situacija u kojoj su testeri ispitali sistem na različitim nivoima, tj. izvršeno je testiranje komponenti, integraciono testiranje i testiranje sistema i svi detektovani defekti su otklonjeni, ali neki korisnički zahtjevi nisu ispunjeni ili neki dio sistema ne radi na očekivan način. Zato je ovaj nivo testiranja veoma bitan korak.

Najbitnije je utvrditi da li korisnici mogu koristiti sistem za ispunjavanje svojih potreba i zahtjeva i da li mogu izvršavati poslovne procese sa minimalnim rizikom i cijenom.

Pitanja na koje ovaj tip testiranja treba dati odgovore su [9]:

- Da li sistem može biti pušten u rad?
- Da li je razvojni tim ispunio sva očekivanja?

Tokom ovog testiranja moguće je naići na greške, ali postojanje većeg broja grešaka smatra se velikim rizikom projekta. Testiranje prihvatljivosti se uglavnom fokusira na validaciju, tj. ispituje se da li sistem odgovara svojoj svrsi [9].

Forme ovog tipa testiranja [1]:

- Testiranje u simuliranom okruženju
- Ugovorno testiranje
- Regulatorno testiranje
- Alfa testiranje
- Beta testiranje

2.4.4.1 Testiranje u simuliranom okruženju

Ovakvo testiranje se izvodi u okruženju koje simulira stvarno okruženje u kojem će se koristiti proizvod. Operacije na koje se fokusira ovo testiranje su [1]:

- Instaliranje, deinstaliranje, ažuriranje
- Održavanje
- Testiranje performansi
- Učitavanje podataka
- Testiranje sigurnosnih propusta

2.4.4.2 Ugovorno testiranje

Ugovorno testiranje se izvodi na osnovu ugovornog kriterijuma koji je kreiran kada zainteresovane strane potpišu ugovor. Ovaj tip testiranja najčešće izvršavaju krajnji korisnici ili nezavisni testeri [1].

2.4.4.3 Regulatorno testiranje

Regulatorno testiranje se izvodi na osnovu bilo kakvih postojećih regulativa kao što su zakoni, mjere zaštite [1]...

2.4.4.4 Alfa testiranje

Alfa testiranje je testiranje koje izvode potencijalni korisnici, naručioci ili tim testera koji nisu radili na razvoju softvera koji se testira [10]. Testiranje se izvodi tako što se vrši simulacija scenarija koji se mogu realizovati korištenjem softvera, da bi se ispitala sve funkcionalnosti softvera, performanse i sama upotrebljivost [11].

Cilj je da se dobije povratna informacija od osoba koje vrše testiranje, prije nego što je proizvod pušten u rad. Ukoliko testeri prijave grešku, prelazi se na proces rješavanja greške, pa počinje novi ciklus testiranja [11]. Zbog toga je proces alfa testiranja veoma bitan, jer omogućava razvojnom timu da dobije informaciju o kvalitetu i stabilnosti samog softvera.

Nakon otklanjanja svih grešaka i kada se alfa testiranje završi bez da testni tim prijavi nove greške, prelazi se na proces beta testiranja.

2.4.4.5 Beta testiranje

Beta testiranje je testiranje koje se vrši nakon završetka alfa testiranja. U ovom slučaju se ograničenom broju korisnika isporučuje takozvana beta verzija softvera, da bi se testiralo da li softver sadrži greške koje nisu detektovane prethodno izvršenim testiranjem [10]. Često se desi da se beta verzija softvera učini javno dostupnom da bi se dobila povratna informacija od što većeg broja korisnika.

Kada je u pitanju beta testiranje, fokus nije na pronalasku grešaka i problema sa funkcionalnostima, već je cilj utvrditi koliko je softver siguran, pouzdan i upotrebljiv [11]. Beta testiranje u stvari predstavlja testiranje crne kutije, što je objašnjeno u poglavlju 2.5.2.1.

2.5 Tehnike testiranja

Postoje različite tehnike testiranja i svaka od njih ima prednosti i mane. Cilj neke tehnike testiranja jeste da pomogne pri identifikaciji uslova testiranja, testnih slučajeva i testnih podataka [1]. Tehnike testiranja se mogu podijeliti u dvije grupe [12]:

- Statičke tehnike testiranja
- Dinamičke tehnike testiranja

2.5.1 Statičke tehnike testiranja

Statičke tehnike testiranja ne izvršavaju kod koji se ispituje i generišu se prije nego što se izvrše bilo kakvi testovi u softveru.

Statičkim tehnikama se može testirati [1]:

- Funkcionalni zahtjevi, sigurnosni zahtjevi, specifikacija zahtjeva
- Kod
- Specifikacije koje se tiču dizajna
- Ugovori, projektni planovi, budžet
- Testni slučajevi
- Dijagrami aktivnosti

Da bi vršio ovu tehniku testiranja, tester ne mora poznavati programski jezik koji je korišten za implementaciju softvera, jer ova faza ne uključuje izvršavanje koda. Glavni cilj statičkih tehnika testiranja jeste poboljšanje kvaliteta softvera na način da se pomogne developerima da identifikuju i riješe greške što ranije u procesu razvoja softvera, jer je otklanjanje grešaka u ranijim fazama dosta jeftinije i brže [12].

Identifikacija i otklanjanje defekata korištenjem tehnika statičkog testiranja je skoro uvijek jeftinije nego korištenjem dinamičkih tehnika, pogotovo zbog dodatnih troškova koji nastaju zbog potrebe za ažuriranjem i korištenjem potvrdnog (poglavlje 2.6.4.1) i regresivnog testiranja (poglavlje 2.6.4.2), koje je neophodno odraditi ukoliko se defekt identifikuje i otkloni korištenjem neke tehnike dinamičkog testiranja [1].

2.5.2 Dinamičke tehnike testiranja

Dinamičko testiranje predstavlja tehniku testiranja koja uključuje izvršavanje koda. Da bi tester mogao da izvršava ovu tehniku testiranja, mora imati određeno znanje o programskom jeziku, kao i okruženju koje je korišteno za implementaciju softvera. Uključuje odabir određenih ulaznih vrijednosti i analiziranje izlaza koje softver generiše na osnovu njih [12].

Dinamičke tehnike testiranja dijelimo na:

- Testiranje crne kutije
- Testiranje bijele kutije
- Testiranje bazirano na iskustvu

2.5.2.1 Testiranje crne kutije

Ova tehnika testiranja posmatra softver kao crnu kutiju sa ulazima i izlazima, budući da testeri ne znaju kakva je struktura sistema i njegovih komponenti (Slika 2). Testere zanima šta sistem radi, a ne kako nešto radi. Ovu tehniku testiranja možemo koristiti za funkcionalno testiranje (poglavlje 2.6.1), jer se funkcionalno testiranje bavi tim šta sistem radi, koje su njegove mogućnosti i funkcije [13].



Slika 2. Primjer testiranja crne kutije

Testni slučajevi i testni podaci mogu sadržati softverske zahtjeve, specifikacije i slučajeve upotrebe, a testni slučajevi se mogu iskoristiti da pronađu nedostatke u implementaciji zahtjeva, kao i odstupanja od zahtjeva [13]. Testeri u ovom slučaju samo znaju ulazne podatke i očekivane izlazne podatke, ali ne znaju način na koji softver kreira izlazne podatke i nikada ne pristupaju kodu. Iz ovog razloga tester i developer su nezavisni jedan od drugog.

Ova tehnika testiranja se može koristiti na svim nivoima testiranja: prilikom testiranja komponenti, prilikom integracionog testiranja, testiranja sistema i testiranja prihvatljivosti [13].

Testiranje crne kutije je veoma efikasna tehnika za testiranje velikih segmenata koda. Ono što je nedostatak ove tehnike je to što nisu svi mogući testni scenariji izvršeni, što rezultira time da pokrivenost testova nije najbolja [7]. Budući da se testiranje provodi prema specifikaciji, nije lako kreirati kvalitetne testne slučajeve.

Testiranje koje se može vezati za testiranje crne kutije je vizualno testiranje, koje se još naziva i vizualno testiranje korisničkog interface-a. To je testiranje čiji je cilj da utvrdi kakav je izgled i ponašanje korisničkog interface-a [14]. Vizualno testiranje se fokusira na vizuelne elemente aplikacije kao što su slike, boje, fontovi, raspored elemenata na ekranu i da li je izgled i raspored tih elemenata konzistentan kroz čitavu aplikaciju i da li su konzistentni na različitim uređajima, web čitačima i operativnim sistemima. Tačnije, provjerava se da li svaki prikazani element ima odgovarajući oblik, da li je odgovarajuće boje i na odgovarajućoj poziciji na ekranu [15].

Vizualno testiranje omogućava testerima da detektuju defekte i manjak konzistentnosti kada je u pitanju izgled softvera, što može negativno uticati na iskustvo krajnjih korisnika.

2.5.2.2 Testiranje bijele kutije

Ova tehnika testiranja se naziva još i testiranje staklene kutije ili providne kutije, s obzirom na to da je za razliku od tehnike crne kutije, omogućen pristup kodu i da se zahtjeva znanje o tome kako je softver implementiran tj. na koji način radi. Ova tehnika testiranja se bavi analizom arhitekture, dizajna, interne strukture ili koda objekata koji se testiraju [13].

Testiranje bijele kutije je tehnika koja se može koristiti za testiranje komponenti, integracije i testiranje sistema i neophodno je da tester ima pristup izvornom kodu da bi tester znao koji dio koda ne radi na očekivani način [13].

Testni slučajevi i testni podaci mogu sadržati kod, arhitekturu softvera⁸, dizajn softvera⁹ ili neke druge informacije koje se tiču strukture softvera [1]. Specifikacije se takođe mogu iskoristiti kao dodatni izvor informacija da bi se odredio predviđeni rezultat testnih slučajeva [16].

2.5.2.3 Testiranje bazirano na iskustvu

Ove tehnike koriste znanje i iskustvo developera, testera, zainteresovanih strana i krajnjih korisnika da bi se dizajnirali, implementirali i izvršavali testni slučajevi [1]. Učešće svih nabrojanih je veoma važno, jer zahvaljujući njihovim različitim znanjima i iskustvima, veća je vjerovatnoća da će neko od njih moći predvidjeti do kakvih grešaka može doći, što je veoma važno za testiranje.

2.6 Tipovi testiranja

Tip testiranja predstavlja grupu određenih testnih aktivnosti koje imaju za cilj testiranje željene karakteristike softverskog sistema ili nekog njegovog dijela [1]. Različiti tipovi testiranja se koriste da bi se postigli različiti ciljevi [1]:

- Provjera da li je softver u potpunosti implementiran i da li je implementiran na odgovarajući način
- Provjera nefunkcionalnih karakteristika, kao što su pouzdanost, sigurnost, upotrebljivost, performanse
- Provjera da li je arhitektura i struktura softvera ispravna, potpuna i da li odgovara softverskim zahtjevima
- Provjera da li je otklanjanje grešaka u softveru uticalo na otklanjanje defekata i da li se softver ponaša na odgovarajući način nakon načinjenih promjena

Na osnovu prethodno pomenutih ciljeva, kreirani su sljedeći tipovi testiranja [1]:

- Funkcionalno testiranje (testiranje crne kutije)
- Nefunkcionalno testiranje
- Strukturalno testiranje (testiranje bijele kutije)
- Testiranje koje se tiče promjena načinjenih u softveru

2.6.1 Funkcionalno testiranje (testiranje crne kutije)

Funkcionalno testiranje u stvari predstavlja testiranje crne kutije, koje podrazumijeva izvršavanje testova koji ispituju funkcije koje sistem treba da izvršava, a prethodno je detaljno opisano u poglavlju 2.5.2.1 .

Temeljnost funkcionalnog testiranja je osobina funkcionalnog testiranja koja se može izmjeriti kroz funkcionalnu pokrivenost. Funkcionalna pokrivenost predstavlja u kojoj mjeri je neki funkcionalni element pokriven testovima i izražena je u procentima [1].

⁸ Arhitektura softvera – definiše kako se komponente softvera sastavljaju, definiše njihov odnos i komunikaciju između njih. Služi kao polazni nacrt i koristi tim ga developera za dalji razvoj.

⁹ Dizajn softvera – proces definisanja softverskih metoda, funkcija, objekata i cjelokupne strukture koda, tako da rezultujuće funkcionalnosti zadovolje sve zahtjeve krajnjih korisnika, ali i naručioca softvera.

2.6.2 Nefunkcionalno testiranje

Nefunkcionalno testiranje vrši evaluaciju nefunkcionalnih karakteristika softvera. Kod ovog tipa testiranja fokus je na tome koliko dobro je nešto urađeno ili koliko je brzo nešto urađeno. Nefunkcionalno testiranje se, kao i funkcionalno testiranje, može vršiti na svim nivoima [12]. Uključuje testiranje performansi, iskoristivosti, mogućnosti održavanja, pouzdanost i prenosivost. Tačnije, testira se koliko dobro sistem radi [1].

Internacionalna organizacija za standardizaciju¹⁰ je definisala skup karakteristika koje se odnose na kvalitet softvera. ISO-9126 standard definiše šest karakteristika, koje je dalje dijele na potkarakteristike. Pomenute karakteristike su [17]:

- Funkcionalnost – mogućnost softvera da obezbijedi funkcije koje zadovoljavaju navedene i podrazumijevane potrebe kada se softver nalazi pod određenim uslovima. Te funkcije trebaju da zadovolje navedene ili podrazumijevane potrebe. Dalje se dijeli na:
 - prikladnost
 - preciznost
 - sigurnost
 - interoperabilnost (sposobnost dva ili više uređaja da neometano rade zajedno)
- Pouzdanost - predstavlja mogućnost softvera da održi svoj nivo performansi, pod određenim uslovima, duži vremenski period. Dalje se dijeli na:
 - robustnost (sposobnost softvera da neometano, čak i kada su prisutne greške)
 - mogućnost oporavka
 - otpornost na greške
- Upotrebljivost – mogućnost softvera da bude razumljiv, proučen i korišten od strane krajnjih korisnika, kad se softver koristi pod određenim uslovima. Dalje se dijeli na:
 - razumljivost
 - mogućnost učenja
 - učinkovitost
 - atraktivnost
- Efikasnost – mogućnost softvera da obezbijedi odgovarajuće performanse u odnosu na količinu korištenih resursa, kada se softver koristi pod određenim uslovima. Dalje se dijeli na:
 - performanse
 - mogućnost iskorištavanja resursa
- Mogućnost održavanja – mogućnost softvera da se mijenja. Modifikacije mogu uključivati korekcije, poboljšanja ili adaptaciju softvera na promjene nastale u okruženju ili u zahtjevima koje treba da ispuni. Dalje se dijeli na:
 - promjenjivost,
 - stabilnost

¹⁰ The International Organization for Standardization (ISO) – međunarodna organizacija za standardizaciju osnovana 1946. godine i njen cilj je promovisanje razvoja međunarodnih proizvodnih, trgovinskih i komunikacionih standarda. Sastavljena je od 147 članova iz različitih zemalja.

- mogućnost testiranja
- mogućnost analize
- Prenosivost – mogućnost softvera da se prenosi iz jednog okruženja u drugo. Dalje se dijeli na:
 - prilagodljivost
 - mogućnost instalacije
 - zamjenljivost
 - mogućnost suživota sa drugim softverom, u istom okruženju, koristeći zajedničke resurse

2.6.3 Strukturalno testiranje (testiranje bijele kutije)

Strukturalno testiranje u stvari predstavlja testiranje bijele kutije, koje podrazumijeva testiranje sistema ili neke njegova komponente i bavi se testiranjem samog koda [9]. Npr. Testiranje da li je grananje u kodu implementirano na ispravan način [9]. Ovaj tip testiranja je detaljnije opisan u poglavlju 2.5.2.2 .

2.6.4 Testiranje koje se tiče promjena načinjenih u softveru

Testiranje koje se tiče promjena načinjenih u softveru se vrši kada je u sistemu ispravljena detektovana greška ili kada je dodata ili promjenjena funkcionalnost [1]. U takvim slučajevima, neophodno je ponovo izvršiti testiranje koje će potvrditi da nema grešaka ili da je funkcionalnost pravilno implementirana. Prema tome, imamo 2 tipa testiranja [1]:

- Potvrдно testiranje
- Regresivno testiranje

I potvrđno i regresivno testiranje se vrše na svim nivoima.

2.6.4.1 Potvrđno testiranje

Potvrđno testiranje se vrši nakon što je detektovana greška otklonjena. Softver se mora testirati uz pomoć testnih slučajeva koji su pokazali postojanje greške, ali se može dodatno testirati i novim testnim slučajevima [1]. Cilj ovog testiranja je da se potvrdi da je greška uspješno otklonjena.

2.6.4.2 Regresivno testiranje

Regresivno testiranje se vrši jer je moguće da promjena koja je nastala u jednom dijelu koda, može nenamjerno uticati na ponašanje drugih dijelova koda, bilo da je taj dio koda u sklopu iste komponente, ili u sklopu druge komponente, u istom sistemu ili čak u drugom sistemu [1].

Kao kod potvrđnog testiranja, softver se mora testirati uz pomoć testnih slučajeva koji su već korišteni. Za razliku od potvrđnog testiranja, prethodno testiranje je bilo uspješno. Svi testni slučajevi u regresivnom testnom paketu se izvršavaju svaki put kada se kreira nova verzija softvera [1]. Veoma je često da organizacije posjeduju nešto što se zove regresivni testni paket. To predstavlja skup testnih slučajeva koji se koriste za regresivno testiranje. Testovi su dizajnirani tako da testiraju većinu funkcionalnosti u sistemu, ali da ne testiraju detalje te funkcionalnosti [18].

3. VEZA IZMEĐU MODELA RAZVOJA SOFTVERA I TESTIRANJA SOFTVERA

Tip modela koji se odabere za razvoj softvera ima veliki uticaj na to kako će se vršiti testiranje samog softvera, jer testiranje nije samostalan proces i umnogome zavisi od odabranog modela razvoja softvera, tačnije, model razvoja definiše kada će se testiranje vršiti i šta će se testirati [2].

Zato je je veoma važno da tester i budu upoznati sa tipom odabranog modela, da bi na odgovarajući način izvršili testiranje, jer način na koje se izvodi testiranje mora odgovarati odabranom modelu testiranja.

Koji model razvoja softvera je odabran zavisi od ciljeva koji se žele postići. Postoji nekoliko tipova razvoja softvera koji su razvijeni upravo iz tog razloga [19]:

- Model vodopada
- V model
- Iterativni model
- Agilni model
- Spiralni model

3.1 Model vodopada

Model vodopada je najstariji i najpoznatiji model razvoja softvera. Ono što je specifično kod ovog modela jeste da se koraci izvršavaju sekvencijalno (Slika 3). Ovaj model je pogodan za koristan za projekte kod kojih je kontrola kvaliteta od velikog značaja zbog obimne dokumentacije i intenzivnog planiranja [20].



Slika 3. Model vodopada

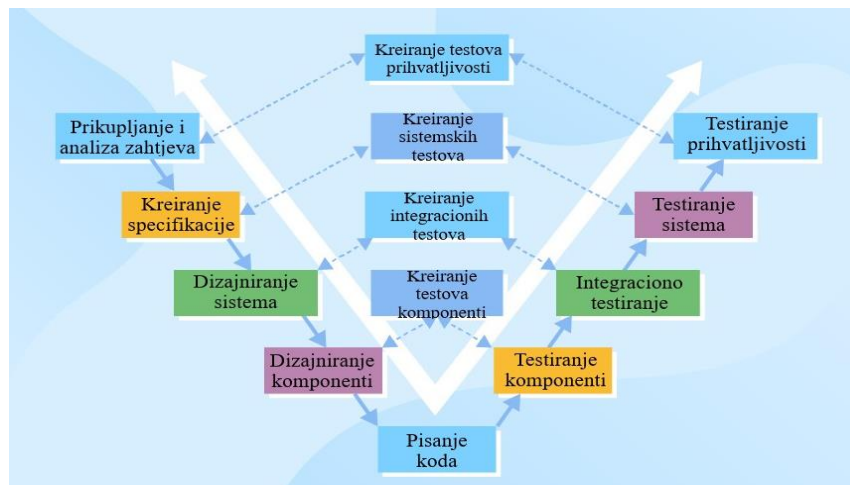
Faze razvoja se u ovom modelu ne preklapaju, što znači da naredna faza ne počinje dok se prethodna ne završi [77]. Testiranje se u ovom tipu modela dešava na kraju, tako da su svi defekti otkriveni u trenutku kada je rok za predaju projekta blizu, što znači da nije lako ispraviti greške koje su se desile u kasnijim fazama razvoja. Faze modela vodopada su [20]:

1. Prikupljanje zahtjeva – faza u kojoj se određuje kako se treba ponašati sistem koji će se kreirati. Zato u ovoj fazi dolazi do dogovora između naručioca sistema i developera, tj. prikupljaju se zahtjevi koje imaju naručioci.
2. Analiza zahtjeva – faza u kojoj se vrši analiza zahtjeva prikupljenih u prethodnoj fazi, kreiraju se softverski zahtjevi i oni se nakon toga dokumentuju.
3. Dizajn sistema – faza u kojoj se prikupljaju informacije iz prethodne faze i analiziraju, te se na osnovu tih informacija bira odgovarajući arhitekturni dizajn, kreira se šema baze podataka, te se određuje na koji način će se podaci organizovati, čuvati i obrađivati.
4. Pisanje koda – u fazi kodiranja programeri kreiraju kod.
5. Testiranje – u fazi testiranja se provjerava da li je kreirani softver u saglasnosti da prikupljenim softverskim zahtjevima. Takođe, u ovoj fazi se vrši detaljno testiranje da li softver sadrži bilo kakve greške koje mogu dovesti kasnije do nekih defekata.
6. Održavanje - faza u kojoj je softver pušten u rad. Cilj je da softver nastavi da radi bez bilo kakvih problema, a ako dođe do bilo kakve greške, nastoji se da se ona otkloni.

3.2 V model

V model razvoja softvera je nastao kao odgovor na nedostatke koje sadrži model vodopada [2]. Budući da se defekti kod modela vodopada otkrivaju prekasno u životnom ciklusu softvera i testiranje se izvršava tek na kraju razvoja, često se desi da faza testiranja utiče na datum isporuke softvera naručiocu [2].

V model, sa druge strane zahtjeva da se testiranje odvija od najranijih faza razvoja i u ovom slučaju nakon svake faze, a može se desiti i da se vrši paralelno sa nekom fazom. Vrlo je važno da tester rade zajedno sa ostalim učesnicima u razvoju softvera da bi se kreirali što efikasniji testovi [21]. Budući da se testiranje vrši od najranijih faza, moguće je pronaći greške i u dokumentaciji, što će spriječiti da se greške propagiraju u fazu kodiranja (Slika 4). Validacija se vrši u ranijim fazama, na primjer u analizi korisničkih zahtjeva, ali i u kasnim fazama, prilikom testiranja prihvatljivosti [2].



Slika 4. V model

Faze razvoja [2]:

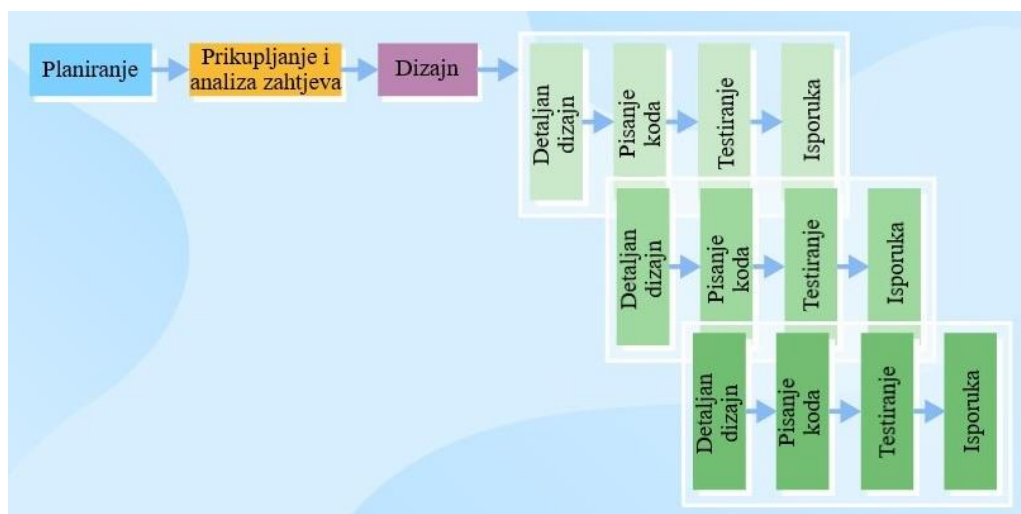
1. Prikupljanje i analiza zahtjeva – Prva faza u V-modelu predstavlja fazu prikupljanja i analize zahtjeva korisnika. Tokom ove faze se takođe kreiraju **testovi prihvatljivosti** (poglavlje 2.4.4).
2. Kreiranje specifikacije – Nakon definisanja zahtjeva, sljedeća faza je dokumentovanje specifikacije. U toku ove faze se kreiraju i **sistemski testovi** (poglavlje 2.4.3).
3. Dizajniranje sistema – U toku ove faze, na osnovu dokumentovane specifikacije se detaljno određuje kako su različite komponente sistema povezane jedne sa drugima. Takođe, u toku ove faze pisu se i **integracioni testovi** (poglavlje 2.4.2).
4. Dizajniranje komponenti – U toku ove faze dizajniraju se sve pojedinačne komponente sistema i kreiraju se **testovi komponenti** (poglavlje 2.4.1).
5. Pisanje koda – U ovoj fazi se piše sav kod sistema.

Nakon faze implementacije, slijedi faza validacije gdje se sistem testira različitim testovima da bi se osiguralo da sistem radi sve ono što treba [2]. Takođe, ukoliko neki test nije prošao, vrši se modifikacija samog softvera, nakon čega bi softver opet trebao da se testira na odgovarajući način, da bi se utvrdilo da je pronađena greška otklonjena i tada koristimo potvrdno i regresivno testiranje (poglavlja 2.6.4.1 i 2.6.4.2).

Treba imati na umu da je cijena otklanjanja greške manja, što je greška pronađena ranije u projektu.

3.3 Iterativni model

Da bi se započeo razvoj softvera pomoću iterativnog modela, lista korisničkih zahtjeva ne mora biti kompletna. Razvojni proces može da počne kada se prikupi određen broj zahtjeva, a konačna lista zahtjeva se naknadno može proširiti. Kada se doda neki novi zahtjev, proces se ponavlja i tako se dobija nova verzija proizvoda nakon završetka svakog ciklusa [20]. Svaka iteracija u ovom slučaju predstavlja razvoj zasebne komponente softvera, koja se dodaje na ranije kreiranu verziju (Slika 5).



Slika 5. Iterativni model

Prednost ovog tipa modela je što je lako provjeriti kako napreduje razvoj softvera, a problemi koji su uočeni u okviru jedne iteracije se mogu riješiti prije početka sljedeće i tako je onemogućeno njihovo propagiranje u kasnije faze razvoja softvera [2].

Ono što je najveća mana je to što neki problemi će se javiti tek u kasnijim fazama razvoja softvera i mogu uticati na rok za isporuku gotovog proizvoda.

3.4 Agilni model

Agilni model razvoja softvera se fokusira na adaptivnost prilikom razvoja [22]. U ovom slučaju se umjesto tačno definisanog rasporeda koraka u razvoju vrši više iteracija svih koraka i nastoji se da nakon svake iteracije dobijemo bolji krajnji proizvod. Svaka iteracija se sastoji od sljedećih koraka [22]:

1. Planiranje
2. Dizajn
3. Kodiranje
4. Testiranje
5. Evaluacija od strane naručioca sistema

Ovaj model omogućava da se vrše promjene u dizajnu u zadnjem trenutku, što znači da je dozvoljeno da se dizajn razvija i unapređuje i nove ideje se uzimaju u obzir tokom procesa razvoja [23] (Slika 6). Takođe, greške detektovane u toku testiranja, biće ispravljene u narednoj iteraciji. Dokumentacija je u ovom slučaju manje bitna, a naglasak je na brzini kojom će se isporučiti krajnji proizvod [23].



Slika 6. Agilni model

Nakon kraja svake iteracije je moguće pokazati naručiocu trenutno stanje softvera, tako da je moguće uzeti u obzir povratne informacije od naručioca i primijeniti prijedloge u sljedećoj iteraciji [23].

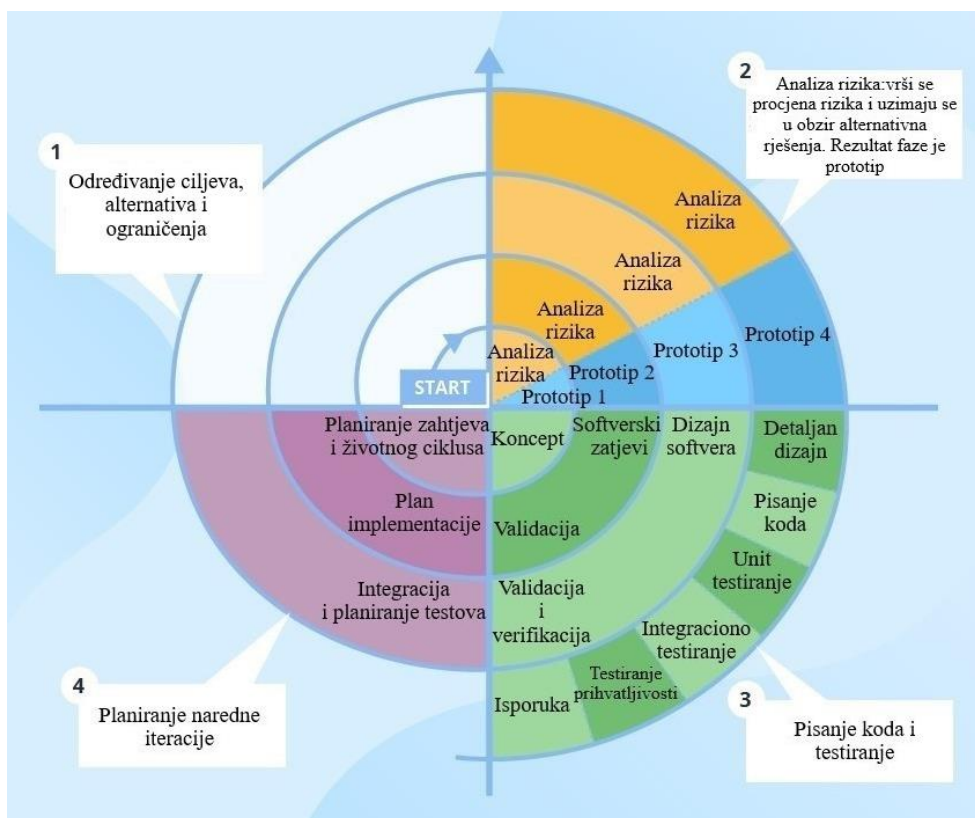
3.5 Spiralni model

Spiralni model je dosta sličan agilnom modelu, s tim da se više značaja pridaje analizi rizika. Projekat se dijeli na više segmenata, što omogućava da se izmjene lakše implementiraju

[24]. Ovaj tip modela omogućava da razvoj softvera počne na osnovu malog dijela softverskih zahtjeva, koji će se tokom razvoja upotpuniti. Tim će tako dodavati funkcionalnosti za dodatno skupljene zahtjeve prilikom svake „spirale“, sve dok softver ne bude spreman za isporuku i fazu održavanja (Slika 7) [20].

Faze razvoja [24]:

1. Određivanje ciljeva, alternativa i ograničenja
2. Analiza rizika: vrši se procjena rizika i uzimaju se u obzir potencijalna alternativna rješenja. Rezultat ove faze je prototip.
3. Pisanje koda i testiranje
4. Planiranje naredne iteracije



Slika 7. Spiralni model

Ovaj tip modela se koristi pri razvoju kompleksnih sistema, a cijena razvoja softvera na osnovu ovog tipa modela je nešto veća nego kod ostalih modela, upravo zbog analize rizika [20].

4. RADNI OKVIRI ZA TESTIRANJE

Svrha radnih okvira jeste automatizacija procesa testiranja, tako da tester ne mora ručno pokretati svaki test.

Radni okvir predstavlja skup pravila i smjernica koji omogućavaju testeru da na efikasan način razvija, izvršava testne skripte i kreira izvještaje na osnovu rezultata izvršenih testova. Sadrže različite alate koji su dizajnirani da pomognu testeru da kreira efikasne testne slučajeve [25].

Za razliku od komercijalnih radnih okvira, čije korištenje korisnici mogu plaćati i nekoliko hiljada maraka na godišnjem nivou, nekomercijalna rješenja su besplatna za korištenje i imaju velike i aktivne zajednice developera i korisnika koji doprinose njihovom poboljšanju, kao i dokumentaciji [26]. U većini slučajeva podržavaju veći broj jezika i platformi. Međutim, mogu imati manji kvalitet i sigurnosne standarde od komercijalnih rješenja, budući da ne podliježu rigoroznim testiranjima i procesima verifikacije. Takođe, vrlo je moguće da neće imati adekvatnu i ažurnu korisničku podršku i redovne update-e. Moguće je i da ne podliježu određenim industrijskim standardima ili regulacijama [25].

4.1 Tipovi radnih okvira za testiranje

Postoji nekoliko tipova radnih okvira za testiranje [27]:

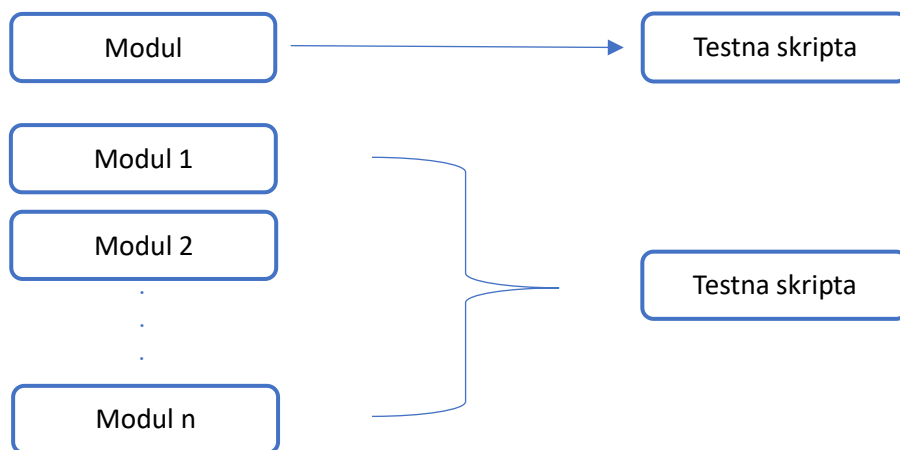
- Radni okviri bazirani na modulima
- Radni okviri koji se baziraju na arhitekturi biblioteka
- Radni okviri koji se baziraju na podacima
- Radni okviri koji se baziraju na ključnim riječima
- Radni okviri koji se baziraju na ponašanju
- Hibridni radni okviri

4.1.1 Radni okviri bazirani na modulima

Ovaj tip radnog okvira se bazira na jednom od najpopularnijih koncepata u objektivno orijentisanom programiranju – apstrakciji. Ovaj tip radnog okvira dijeli softver koji se testira na određeni broj modula. Za svaki modul postoji posebna testna skripta, a može postojati i testna skripta koja služi za testiranje grupe modula (Slika 8) [27].

Prednosti ovog tipa radnog okvira su to što omogućava visok nivo modularizacije, tj. podjelu softvera na veći broj modula, te omogućava lakše i jeftinije održavanje. Još jedna od prednosti je ta što promjene nastale u jednom modulu neće uticati na druge module [95].

Mana ovog tipa radnog okvira je to što se testni podaci nalaze unutar same testne skripte i ukoliko je potrebno testirati modul drugačijim setom podataka, potrebno je mijenjati samu testnu skriptu [25].



Slika 8.

4.1.2 Radni okviri koji se baziraju na arhitekturi biblioteka

Radni okviri koji se baziraju na arhitekturi biblioteka koriste sličan pristup kao i radni okviri bazirani na modulima. U ovom slučaju se softver koji se testira, umjesto da se dijeli na module, dijeli na funkcionalnosti i te funkcionalnosti će činiti biblioteku tog softvera. Pri testiranju, testne skripte imaju mogućnost pozivanja određenih biblioteka [25].

Prednost ovog tipa radnog okvira je ta što omogućava visok stepen modularizacije softvera i lako i jednostavno održavanje, te mogućnost ponovne upotrebe, jer se kreirana biblioteka može koristiti od strane različitih testnih skripti [27].

Mana ovog pristupa je, kao i u slučaju radnih okvira baziranih na modulima, to što se testni podaci nalaze u samoj skripti i ukoliko je potrebno vršiti testiranje drugačijim setom podataka, mora se napraviti promjena u samoj testnoj skripti [27].

4.1.3 Radni okviri koji se baziraju na podacima

U ovom slučaju, radni okvir razdvaja logiku i testne podatke, te čuva testne podatke u nekoj eksternoj bazi (npr. XML, Excel, CSV fajlovi). Podaci se čuvaju u parovima “ključ-vrijednost” i da bi se pristupilo podacima ili da bi se unijeli podaci, koristi se ključ [25].

Glavna prednost ovog pristupa je to što je smanjen broj testnih skripti koje su potrebne da bi se pokrili svi mogući scenariji, jer jedna testna skripta može koristiti različite testne podatke. Promjena podataka u bazi neće uticati na testne skripte [25]. Ovaj pristup omogućava povećanje fleksibilnosti i lakše održavanje i svaki testni scenario se može testirati sa različitim testnim podacima.

Mana je to što se povećava kompleksnost, jer su potrebni novi elementi koji će omogućiti čitanje podataka iz određene baze.

4.1.4 Radni okviri koji se baziraju na ključnim riječima

Slično radnim okvirima koji se baziraju na podacima, ovaj pristup razdvaja testne podatke i testne skripte, a pored toga, čuva testne skripte u eksternom fajlu [27]. Kod u ovom fajlu se zove ključna riječ i organizovan je pomoću tabele (Tabela 1).

Npr. Kada glavnoj testnoj skripti zatreba određeni kod koji se nalazi u fajlu sa ključnim riječima, alat za automatizaciju poziva odgovarajuću ključnu riječ i na taj način glavna testna skripta dobija odgovarajući argument.

Jedna od glavnih prednosti ovog pristupa je što tester ne mora znati ništa o skriptama. U većini slučajeva ključna riječ odgovara svojoj funkcionalnosti. Još jedna prednost je što se jedna ključna riječ može koristiti u više skripti [28].

Mana je što je kompleksnost izrade ovakvog radnog okvira visoka i potrebno je dosta vremena za izradu. Takođe, kompleksnost raste sa porastom broja ključnih riječi [28].

Broj koraka	Opis	Ključna riječ	Lokator
Prvi korak	Pritisnite link na početnoj strani	clickLink	@id = 'homepage'
Drugi korak	Unesite korisničko ime	inputdata	@id = 'username'
Treći korak	Unesite lozinku	inputdata	@id = 'password'
Četvrti korak	Verifikuj korisnikove informacije	verifyUser	@id = 'User'
Peti korak	Prijava na aplikaciju	login	@id = 'login'

Tabela 1. Organizacija ključnih riječi

4.1.5 Radni okviri koji se baziraju na ponašanju

Radni okviri koji se baziraju na ponašanju omogućavaju testeru da kreira testne slučajeve na jeziku kojim on sam govori, npr. na engleskom ili srpskom jeziku. Korištenje takvog jezika u testnim scenarijima omogućava testerima koji nemaju obimno tehničko znanje da razumiju sam softver koji se razvija [27]. Takođe, omogućava da se poboljša komunikacija između članova tima koji imaju različit stepen znanja i iskustva, kao i njihova komunikacija sa menadžerima projekta i naručiocima samog sistema [27].

Ideja je da se omogućí da se različiti testni podaci mogu koristiti nad istim dijelom koda, što povećava mogućnost ponovne upotrebe koda. Testovi se pišu u posebnim tekstualnim fajlovima, na način da se prati kako bi krajnji korisnik upotrebljavao softver i kako bi se u tom slučaju softver ponašao i pišu se laičkim jezikom. Na osnovu tekstualnih fajlova se piše kod potreban za implementaciju testa u nekom programskom jeziku, npr. Javi ili Python-u [25]. Cucumber je alat koji se često koristi da bi se pisali testni slučajevi kod ovog tipa radnog okvira [27].

Za kreiranje testova koristi se Given-When-Then pristup, tj. navodi se pretpostavka (Given), zatim akcija (When), pa rezultat (Then) [29]. Primjer testa:

Naziv testne skripte: Implementacija logovanja na Gmail nalog pomoću Cucumber alata

Scenario: Logovanje na Gmail nalog od strane krajnjeg korisnika

Pretpostavka: Korisnik je kliknuo link koji ga vodi do stranice za logovanje na Gmail nalog

Akcija: Korisnik treba da unese svoje korisničko ime u polje „Korisničko ime“ i lozinku i polje „Lozinka“

Rezultat: Korisnik se uspješno prijavio na svoj Gmail nalog i redirektovan je na odgovarajuću stranicu

```
1 import cucumber.api.java.en.Given;
2 import cucumber.api.java.en.Then;
3 import cucumber.api.java.en.When;
4
5 public class Sample {
6     @Given("^Korisnik je kliknuo link koji ga vodi do stranice za logovanje na Gmail nalog$")
7     public void user_is_navigating_to_Gmail_Login_Page() throws Throwable {
8         // Napisati odgovarajući kod
9     }
10
11     @When("^Korisnik unosi korisničko ime i lozinku u odgovarajuća polja$")
12     public void user_need_to_enter_username_as_and_password_as(String arg1, String arg2) throws
13         Throwable {
14         // Napisati odgovarajući kod
15     }
16
17     @Then("^Korisnik se ulogovao na Gmail nalog i redirektovan je na odgovarajuću stranicu$")
18     public void user_is_successfully_navigated_to_the_Gmail_Mail_Box() throws Throwable {
19         // Napisati odgovarajući kod
20     }
21 }
```

Slika 9. Kod koji nastaje na osnovu prethodno kreiranog testa

4.1.6 Hibridni radni okviri

Kao što im samo ime govori, hibridni radni okviri predstavljaju kombinaciju dva ili više prethodno pomenuta tipa radnih okvira. Ono što je dobra strana ovog tipa radnih okvira je to što iskorištavaju dobre strane kombinovanih radnih okvira.

4.2 Primjeri komercijalnih radnih okvira

4.2.1 Unified Functional Testing (UTF)

Unified Functional Testing (UTF) predstavlja alat koji omogućava automatizaciju procesa funkcionalnog i regresivnog testiranja i jedan je od najpopularnijih na tržištu. Razvijen je i održavan od strane kompanije Micro Focus¹¹ [30].

¹¹ Micro Focus – Britanska multinacionalna kompanija koja se bavi informacionim tehnologijama i stacionirana je u Newbury-u, Engleska. Osnovana je 1976. godine od strane Brian-a Reynolds-a. Kompanija nudi različita softverska rješenja. U januaru 2023. godine je preuzeta od strane kanadske softverske kompanije OpenText.

UTF se fokusira na testiranje GUI-ja¹² i API-ja¹³. Budući da pruža mogućnost kombinovanja prethodno pomenutih testiranja, postoji mogućnost testiranja funkcionalnosti različitih slojeva aplikacije, kao npr. front-end GUI sloj, ali i back-end servisni sloj. U slučaju ugrađenih sistema, budući da oni nemaju GUI, od interesa je API testiranje, jer ugrađeni sistemi imaju interface-e. Budući da je cilj testiranja integracija različitih uređaja, API testiranje je od značaja, jer se testiraju funkcionalnosti putem interface-a, što je od suštinskog značaja [25]. Međutim, da bi UTF mogao komunicirati sa ugrađenim uređajima, potrebno je razviti dodatne komponente.

UTF koristi pristup zasnovan na ključnim riječima i koristi Visual Basic Scripting Edition (VBScript), JavaScript, Visual Basic, Visual C++ ili Visual Studio .NET jezik za kreiranje testnih skripti [31]. Takođe, omogućava testiranje bazirano na podacima, tako što omogućava postojanje baze podataka u kojoj će se nalaziti podaci koji će se koristiti u testnim slučajevima. Omogućava i kreiranje testnih biblioteka [25].

Cijena korištenja ovog alata za automatizaciju procesa testiranja, na godišnjem nivou iznosi oko 3200 dolara, a nudi besplatan probni period od 60 dana [32].

4.2.2 VectorCAST

VectorCAST predstavlja porodicu proizvoda koji omogućavaju automatizaciju procesa testiranja tokom cijelog životnog ciklusa softverskog proizvoda. Razvijeni su i održavani od strane kompanije Vector Software¹⁴ i fokusiraju se na ugrađene sisteme [33].

U okviru ove porodice proizvoda postoje [25]:

- **VectorCAST/C++** – automatski generiše radni okvir za testiranje jedne ili više jedinica izvornog koda aplikacije koja je napisana u programskom jeziku C ili C++. Takođe, generiše i izvršava testne slučajeve i generiše izvještaje na osnovu rezultata testiranja.
- **VectorCAST/RSP** – nadograđuje VectorCAST/C++ tako što mu omogućava izvršavanje u ugrađenom okruženju.

Kombinacijom ova dva rješenja izvorni kod se parsira i generiše radni okvir za testiranje uz pomoć VectorCAST/C++, a VectorCAST/RSP automatizuje komunikaciju između radnog okvira i ugrađenog uređaja [25].

¹² GUI (Graphical User Interface) – korisnički interface koji uključuje grafičke elemente, kao što su prozori, ikonice i dugmad. Termin je nastao sedamdesetih godina prošlog vijeka da bi se grafički interface-i razlikovali od tekstualnih, kao što su interface-i komandne linije

¹³ API (Application Programming Interface) – mehanizam koji omogućava dvjema komponentama računarskog sistema da međusobno komuniciraju, koristeći različite protokole.

¹⁴ Vector Software – Njemačka kompanija osnovana 1988. godine od strane Eberhard-a Hinderer-a, Martin-a Litschel-a, and Dr. Helmut-a Schelling-a. Cilj kompanije je pojednostavljivanje razvoja automobilske elektronike, a bavi se i kreiranjem alata koji otkrivaju greške i olakšavaju proces testiranja.

4.3 Primjeri nekomercijalnih radnih okvira

4.3.1 GaugeFramework

Gauge Framework predstavlja nekomercijalni radni okvir čiji je cilj pisanje testova koji će biti razumljivi svim učesnicima u razvoju softverskog proizvoda [34]. Da bi se to postiglo, razvijena je Gauge terminologija koja uključuje [25]:

- Specifikaciju - u ovom dokumentu će biti opisane određene osobine softvera koji se testira. Specifikacije se zapisuju korištenjem Markdown sintakse.
- Scenarije - predstavljaju jedan od mogućih tokova specifikacije
- Korake - predstavljaju komponente specifikacije koje su mogu izvršavati. Svaki korak je implementiran u nekom programskom jeziku (Java, C# ili Ruby) i izvršava se kada se koraci unutar specifikacije izvršavaju. Mogu biti i parametrizovani, pomoću eksternih fajlova (Excel ili CSV fajlovi)
- Koncepte - omogućavaju da se koraci ponovo koriste sa drugačijim parametrima i kombinuju u posebnu cjelinu (Slika 11)

# Search the Internet	→zaglavlje	step("Goto Google's home page", () => { goto("google.com") });
specifikacije		
## Look for something	→scenario	
* Goto Google's home page	→korak	

Slika 10. Primjer specifikacije napisane pomoću Markdown sintakse i implementacija koraka

```
# Search the Internet
## Look for cakes
* Goto Google's home page
* Search for "Cup Cakes"

## Look for movies
* Goto Google's home page
* Search for "Star wars"
```

Slika 11.

Takođe podržava testiranje bazirano na podacima koristeći Markdown tabele (Slika 12) i eksterne CSV fajlove [34].

```
# Search the internet

|query |
|-----|
|Cup Cakes|
|Star wars|
|Pies |

## Look for things
* Search Google for <query>
```

Slika 12.

4.3.2 Robot framework

Robot framework predstavlja nekomercijalni radni okvir koji se koristi za testiranje prihvatljivosti. Ovaj radni okvir teži tome da kod bude lako razumljiv i da čak i osobe koje nisu tester i mogu razumjeti šta kod tačno radi [25].

Koristi programski jezik Python i pristup koji se bazira na ključnim riječima. Ključne riječi predstavljaju operacije koje radni okvir izvršava da bi se postigao određeni cilj, i predstavljaju osnovu svake skripte ovog testnog okvira [35]. Nije kreiran za rad sa nekim određenim sistemom i nema strogo definisanu listu funkcionalnosti. Naprotiv, kreiran je tako da se može prilagoditi za rad na različitim sistemima. Prilagođavanje određenom sistemu se vrši pomoću biblioteka. Sam radni okvir sadrži neke standardne biblioteke, koje se mogu iskoristiti za izvršavanje nekih čestih slučajeva [35].

Trenutne mogućnosti ovog radnog okvira se proširuju dodavanjem novih biblioteka koje trebaju biti implementirane u programskom jeziku Java ili Python ili korisnici mogu kreirati nove ključne riječi na osnovu postojećih [25].

Više riječi o Robot Frameworku biće u poglavlju 5.3.

5. OPIS PRAKTIČNOG DIJELA DIPLOMSKOG RADA

5.1 Vremenski osjetljiv Ethernet (eng. Time Triggered Ethernet)

Vremenski osjetljiv Ethernet¹⁵ se može koristiti za implementaciju kritičnih aplikacija¹⁶ koje omogućavaju distribuiranu kontrolu¹⁷ gdje je deterministička komunikacija i fiksno kašnjenje u komunikaciji od značaja [42].

Vremenski osjetljiv Ethernet, ili kraće rečeno TTEthernet, implementira komunikacioni mehanizam koji obezbjeđuje servis za determinističku komunikaciju¹⁸ putem Etherneteta [39]. U determinističkim mrežama se posebna pažnja pridaje sinhronizaciji. Budući da se poruke u TTEthernet mreži razmjenjuju u tačno definisanim vremenskim trenucima, neophodno je da svi uređaji u mreži budu sinhronizovani [39].

Ovaj mehanizam uspostavlja i održava globalno vrijeme, koje sinhronizacija lokalnih satova svih TTEthernet uređaja prepoznaje. Globalno vrijeme se koristi kao osnova za implementaciju vremenskog particionisanja, kao i učinkovitog korištenja resursa [43].

Sinhronizacija u TTEthernet mreži ima 2 svrhe [43]:

- Mogućnost zakazivanja i nadziranja prenosa podataka kroz pomenutu mrežu
- Mogućnost sinhronizacije računara koji su povezani u mreži, tako da koriste globalno vrijeme

Krajnji sistemi i switch predstavljaju fizičke komponente u ovakvoj mreži. Postoje tri različite uloge kada je u pitanju sinhronizacija u TTEthernet mreži [43]:

- SM (eng. Synchronization master) doprinosi uspostavljanju sinhronizovanog mrežnog vremena, tako što prenosi informaciju o njegovom lokalnom vremenu mreži slanjem i primanjem frejmova¹⁹, takozvanih PCF-ova (eng. Protocol Control Frames). Ovi frejmovi definišu vrijeme i prioritet slanja poruke CM-u.
- CM (eng. Compression master) doprinosi uspostavljanju sinhronizovanog mrežnog vremena, tako što kompresuje primljene PCF-ove i dodjeljuje im globalno vrijeme koje se koristi u mreži, te šalje ovu informaciju nazad SM-u i SC-u preko PFC-ova
- SC (eng. Synchronization client) ne doprinosi uspostavljanju sinhronizovanog mrežnog vremena i ne generiše nikakav saobraćaj, nago samo sinhronizuje svoje sopstveno vrijeme kada primi PFC.

¹⁵ Ethernet – mrežna tehnologija za LAN (eng. Local Area Network) mreže. Sastoji se od fizičkog medijuma preko koga informacije putuju u računarskoj mreži (npr. UTP kabl), protokola koji predstavlja skup pravila za kontrolu pristupa medijumu i Ethernet paketa u kojima se prenose podaci.

¹⁶ Kritične aplikacije – aplikacije nekorektno izvršavanje može imati katastrofalne posljedice po organizaciju u kojoj se koristi.

¹⁷ Distribuirana kontrola – tip kontrole u kojem ne postoji centralni kontroler na određenoj lokaciji koji ima ulogu kontrolisanja određenih funkcionalnosti, nego postoji više kontrolera na više lokacija koji su zaduženi za kontrolisanje određenih funkcionalnosti.

¹⁸ Deterministička komunikacija – mogućnost mreže da garantuje da će se neki događaj desiti (npr. poruka će biti dostavljena) u tačno određenom i predvidivom vremenskom roku.

¹⁹ Frejm (eng. frame) – jedinica transporta podataka u računarskim mrežama i predstavlja kontejner za paket koji se prenosi preko mreže.

Obično se uloga CM-a dodjeljuje switch-u, a SM-a krajnjem sistemu. Ulogu SC-a može imati i krajnji sistem i switch [43].

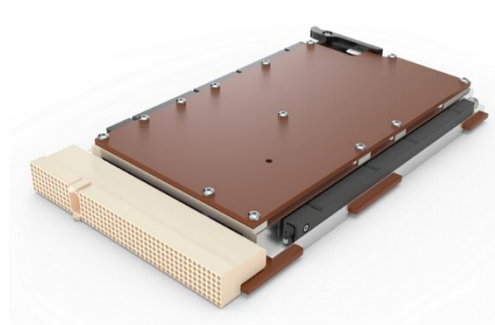
Budući da su u okviru TTEthernet mreže precizno određeni trenuci slanja i primanja paketa, kao i vremenski intervali u kojima se može očekivati njihov dolazak na prijemnoj strani, ovakve mreže su idealne za korištenje u real-time sistemima [39]. Kod ovog tipa mreže ne smije doći do gubljenja određene količine informacija koje se razmjenjuju, pogotovo ukoliko se upotrebljavaju u bezbjednosno-kritičnim sistemima, npr. ukoliko nađu primjenu u avio ili auto industriji. Kod ovog tipa mreža jednostavno je odrediti maksimalna kašnjenja poruka, a važno je napomenuti da ne smije doći do odbacivanja paketa jer se njima prenose veoma bitne informacije koje su neophodne za funkcionisanje real-time sistema od interesa, kao što su informacije o iniciranju određenih operacija, informacije o njihovom trajanju, ali i preuzimanje rezultata neke operacije [39].

5.2 Vremenski osjetljiv switch (eng. Time Triggered Switch)

Kada je riječ o vremenski osjetljivom switch-u, treba znati da postoje dvije verzije pomenutog switch-a, EDU i FLIGHT verzija (Slika 13.) [43]. EDU verzija se koristi za testiranje u laboratorijskim uslovima, a FLIGHT verzija je namijenjena za stvarnu upotrebu, u uslovima letenja. U ovom poglavlju, kada se pominje switch, misli se na EDU verziju switcha.



Slika 13.a) EDU verzija switch-a



Slika 13.b) FLIGHT verzija switch-a

TTE-Switch Space 3U cPCI je 12-portni TTEthernet switch. 3U u nazivu se odnosi na fizičku veličinu switch-a. Budući da se switch nalazi u laboratoriji, potrebno je znati koliko prostora će on zauzeti. 1U označava jednu jedinicu veličine u šasiji, pa će 3U označavati tri jedinice veličine, a njegova standardna veličina je 100mm x 160mm [45]. cPCI (Compact Peripheral Component Interface) se odnosi na šasiju u okviru koje se nalazi, tačnije 3U CompactPCI šasiju (eng. chassis) (Slika 14).



Slika 14. Primjer 3U Compact PCI šasija

Količina podataka koju ovaj switch može da obradi u jedinici vremena [43]:

- 6 x 100/1000 Mbit/s
- 6 x 100 Mbit/s

Podržava tri klase TTEthernet saobraćaja[43]:

- **RC** (eng. Rate Constrained) saobraćaj – RC saobraćaj definiše maksimalno dozvoljeni propusni opseg za određenu poruku da bi se osiguralo ograničenje kašnjenja poruke u kompleksnim mrežama, taj parametar se naziva BAG (eng. Bandwidth Allocation Gap) Uređaji ne moraju biti međusobno sinhronizovani da bi se saobraćaj mogao odvijati. Ukoliko postoji mogućnost za preopterećenje saobraćajnog linka, određeni broj RC paketa će se odbaciti da do toga ne dođe.
- **TT** (eng. Time Triggered) saobraćaj – TT saobraćaj se najčešće upotrebljava za komunikaciju u distribuiranim²⁰ real-time sistemima²¹. Kašnjenje paketa ili nekonzistentnost pri kašnjenju, kao i potreba za determinizmom predstavljaju neke od glavnih razloga zašto se pribjegava korištenju ove klase saobraćaja. Sve što se prenosi ovom klasom saobraćaja, prenosi se u prethodno definisanom vremenskom trenutku. Uloga TT saobraćaja jeste da učini komunikaciju u mreži determinističkom, tj. da se tačno zna u kojem trenutku je poruka poslata i kada primalac može da je očekuje. TT komunikacije se odvija tek nakon što se svi uređaji u mreži koji pripadaju istom sinhronizacionom domeni sinhronizuju. Ova klasa saobraćaja je primarna u TTEthernet mrežama i ima viši prioritet u odnosu na RC i BE saobraćaj.
- **BE** (eng. Best Effort) saobraćaj – Za razliku od TT saobraćaja, nisu definisani trenuci slanja i prijema paketa. Brzina slanja paketa, kašnjenje njihove isporuke i gubitak paketa zavisi od trenutne količine saobraćaja u mreži, kao i mogućnosti hardverskih komponenti mreže. Kada je opterećenje mreže veće, može doći do gubitka paketa, kašnjenja isporuke paketa, kao i varijacija u vremenima kašnjenja, a može doći i do prekida sesije. Ovaj klasa saobraćaja ima najniži prioritet u TTEthernet mreži i odvijaće se samo u slučaju da komunikacioni link nije zauzet od strane neke druge klase saobraćaja.

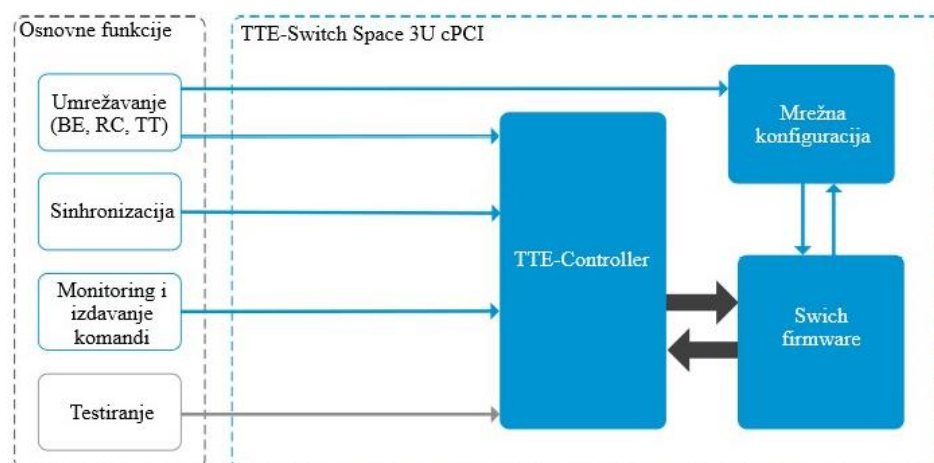
Fizička identifikacija TTEthernet-Switch Space 3U cPCI switch-a se može izvršiti pomoću naljepnice na kojoj se nalaze sljedeće oznake [43]:

- PNR – Broj proizvoda (eng. Product Number)
- SER – Serijski broj (eng. Serial Number)
- DMF – Datum proizvodnje (eng. Date of Manufacturing)

²⁰ Distribuirani sistemi – sistemi čije su komponente fizički razdvojene, ali mogu komunicirati i koordinisati svoje aktivnosti tako što će prosljeđivati poruke jedni drugima i dijeliti resurse.

²¹ Sistemi u realnom vremenu (eng. Real-Time systems) – pojam se odnosi na bilo koji sistem za procesiranje informacija, koji ima hardverske i softverske komponente, i koji može na odgovori na određeni događaj u okviru predviđenog i specificiranog vremenskog okvira. Tipični predstavnici ovog tipa sistema su sistemi za kontrolu avio saobraćaja, sistemi za rezervaciju avionskih karata, srčani pejsmejkari...

Fizički blokovi TTE-Switch Space 3U cPCI su prikazani na slici 15.



Slika 15. Fizički blokovi TTE-Switch Space 3U cPCI-a

- TTE-Controller²² – omogućava pristup osnovnim funkcijama koje obezbeđuje TTE-Controller Space.
- Firmware – omogućava funkcije TTE-Controllera za monitoring i izdavanje komandi
- Mrežna konfiguracija – neophodna je za različite mrežne funkcije koje se primjenjuju putem firmware-a.

TTE-Switch ima sljedeće funkcionalne karakteristike [43]:

- Šest 100/1000 Mbit/s Ethernet portova
- Šest 100 Mbit/s Ethernet portova
- Podržava Multicast²³ i Unicast²⁴ saobraćaj
- MAC Lookup tabela²⁵ sa do 256 Multicast/Unicast adresa
- Dinamičko učenje MAC adresa²⁶
- Podržava 3 klase Ethernet saobraćaja, kao što je prethodno pomenuto
- Udaljeni (eng. Remote) menadžment pomoću ICMP, TFTP i SNMP protokola

²² Kontroler (eng. Controller) – hardverski uređaj ili softverski program koji upravlja ili usmjerava prenos podataka između dva entiteta. Ukoliko se radi o hardverskim uređajima, to mogu biti kartice, mikročipovi ili odvojeni hardverski uređaji koji kontrolišu uređaje koji izvršavaju niz različitih funkcija. Uopšteno govoreći, kontrolerom se može smatrati neko ili nešto što povezuje dva sistema i upravlja komunikacijom među njima

²³ Multicast saobraćaj – saobraćaj u kome se poruke šalju odabranoj grupi primalaca od strane pošiljaoca. To znači da će poruku dobiti klijenti koji su zainteresovani da prime poruke. Ukoliko se ne nalaze unutar multicast grupe, neće primiti poruku.

²⁴ Unicast saobraćaj – najčešći tip komunikacije. U ovom slučaju se poruku pošiljalac šalje samo jednom primaocu. Ovaj tip saobraćaja se koristi pri pretrazi interneta ili transferu fajlova pomoću FTP protokola.

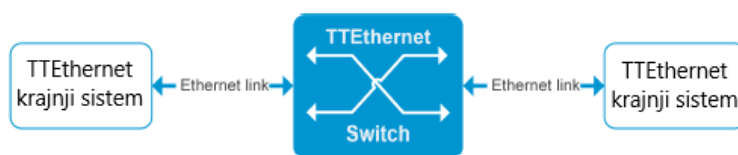
²⁵ MAC (Media Access Control) Lookup tabela – MAC adresa predstavlja fizičku adresu uređaja i ona jedinstveno identifikuje uređaj i proizvođača uređaja. MAC Lookup tabela je tabela koju održava switch. Sastoji se od redova od kojih svaki ima dvije vrijednosti: IP adresu i odgovarajuću MAC adresu uređaja na mreži.

²⁶ Dinamičko učenje MAC adresa – predstavlja automatsko dodavanje unosa u MAC Lookup tabelu kada switch primi frame. U novi red tabele upisuje IP i MAC adresu pošiljaoca. Unosi će biti obrisani ako su neaktivni određeno vrijeme.

5.2.1 Mrežne funkcije

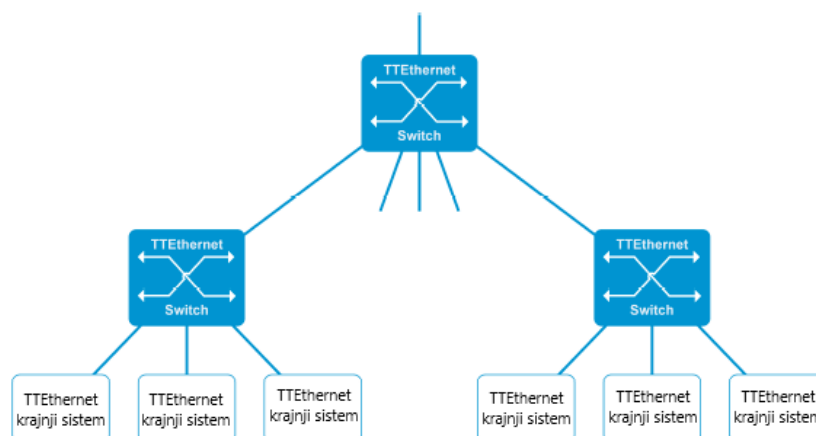
TTEthernet mreža je Ethernet bazirana mreža koja se sastoji od krajnjih sistema, switch-eva i linkova [41]. Paketi se mogu prenositi između krajnjih sistema pomoću jednog ili više switch-eva. Krajnji sistemi predstavljaju početnu ili krajnju destinaciju Ethernet paketa, dok switch-evi usmjeravaju pakete prema prethodno definisanoj TTEthernet konfiguraciji, što znači da TTE-Switch 3U cPCI omogućava računarima da se povežu sa različitim čvorovima u TTEthernet mreži.

Na slici 16 prikazana je mreža koja se sastoji od dva krajnja sistema i jednog switch-a. Krajnji sistemi posjeduju jedan full-duplex komunikacioni port, koji je putem linka povezan sa komunikacionim portom switch-a. U ovom slučaju, switch ima dva komunikaciona porta [43].



Slika 16. Mrežni komunikacioni kanal

Na slici 17 je prikazan primjer mreže koja se sastoji od nekoliko krajnjih sistema, ali i switch-eva. Switch-evi u ovom slučaju povezuju veći broj krajnjih sistema i tako omogućavaju da se paketi prenose od jednog krajnjeg sistema do jednog ili više drugih krajnjih sistema.



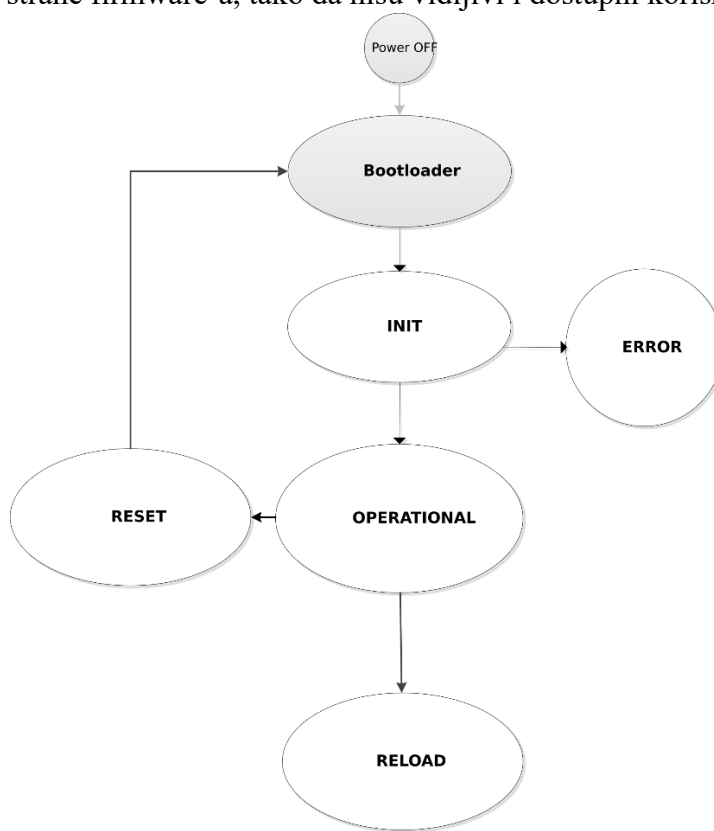
Slika 17. Primjer TTEthernet mreže

Vremenski osjetljiv (eng. Time Triggered) saobraćaj zahtjeva da se zakaže slanje svih poruka koje se šalju u mreži [41]. To znači da će se, za svaku poruku, na odredišnom sistemu definisati period prenošenja poruke, a na switch-u prihvatljiv period koji će garantovati da će se ispoštovati prethodno zakazano vrijeme slanja i da će se izbjeći kašnjenje poruke na krajnjoj destinaciji. Podaci se prenose u paketima, a poruke u payload sekciji paleta. Switch će sve primljene pakete prosljeđivati na osnovu prethodno definisanih vremena slanja, upravo da bi se ispoštovalo prethodno zakazano vrijeme slanja paketa, kao i da bi se izbjeglo kašnjenje paketa (tzv. jitter) [42].

5.2.2 TTE-Switch Space 3U cPCI Firmware

Firmware predstavlja software koji je ugrađen u NVM memoriju (eng. non-volatile memory) switch-a, tj. memoriju koja će nastaviti da čuva podatke, čak i kad je napajanje prekinuto. Koristi se za pokretanje komandi na uređaju i može se smatrati software-om koji omogućava pokretanje hardware-a [38].

Firmware ima nekoliko modova, što je prikazano na slici 18. Modovi obojeni sivom bojom nisu kontrolisani od strane firmware-a, tako da nisu vidljivi i dostupni korisnicima firmware-a.



Slika 18. Modovi firmware-a

Mod firmware-a	Svrha
Power Off	Nikakav software nije pokrenut
Bootloader	Bootloader je pokrenut i izvršava učitavanje default-nog firmware-a iz Flash memorije.
INIT	Firmware je pokrenut i vrši inicijalizaciju hardware-a
OPERATIONAL	Firmware može izvršavati ugrađene testove ili komande
RESET	Firmware se priprema za reset firmware-a.
ERROR	Firmware je zaustavljen jer se desila greška od koje se ne može oporaviti.
RELOAD	Vrši se učitavanje nove firmware slike i izvršava se

Tabela 2. Modovi firmware-a

5.2.3 Flash memorija

Firmware podržava postojanje Flash uređaja koji koristi NOR tehnologiju za čuvanje podataka. NOR predstavlja tip NVM memorije [44]. Podaci u flash memoriji se čuvaju u fajlovima. Imena fajlova su nabrojana u tabeli 3 i to su imena koja prepoznaje uređaj [40]. Ovim fajlovima se može pristupiti, tačnije mogu se pročitati ili upisati, pomoću TFTP protokola, ali se fajlovi mogu flash-ovati, tj. pisati putem ft232 interface-a. Fajl može imati manju veličinu od naznačene u tabeli. Maksimalna veličina se odnosi na veličinu fajla, uključujući header fajla, ECC podatak, i hash vrijednost, jer su ovi podaci fizički zapisani u Flash memoriju [40]. RO/RW kolona tabele se odnosi na dio NVM memorije gdje se fajl nalazi, RO – read-only, RW – read-write.

Ime fajla	Maksimalna veličina fajla	RO/RW	Svrha fajla
TTC_FirmwareDft.bin	384 kiB	RO	Binarni fajl format Default Firmware-a
TTC_FWparamDft.bin	64 kiB	RO	Hardware-specifična konfiguracija default-nog konfiguracija
TTC_NetworkCfgDft.bin	64 kiB	RO	Default-ni mrežni parametar fajl
TTC_EScfgDft.bin	192 kiB	RO	Default-na End-System konfiguracija. Maksimalna veličina koju podaci zauzimaju u ovom fajlu je 100 kiByte
TTC_SWEcfgDft.bin	640 kiB	RO	Default-na konfiguracija switch-a. Maksimalna veličina koju podaci zauzimaju u ovom fajlu je 360 kiByte
TTC_BITcfgDft.bin	64 kiB	RO	Default BIT konfiguracioni fajl. Omogućava konfiguraciju ponašanja ugrađenih testova
TTC_Fwloader.bin	64 kiB	RO	Firmware binarni loader. Maksimalna veličina koju podaci zauzimaju u ovom fajlu je 56 kiByte
TTC_ProductData.bin	64 kiB	RO	Podaci o proizvodnji hardware-a. Maksimalna veličina koju podaci zauzimaju u ovom fajlu je 48 kiByte
TTC_Firmware.bin	384 kiB	RW	Binarni fajl format korisničkog Firmware-a
TTC_Fwparam.bin	64 kiB	RW	Korisnička hardware-specifična konfiguracija default-nog konfiguracija. Napomena: Ovaj fajl mora biti prisutan prije nego default-ni firmware inicira tranziciju u RELOAD mode, da bi učitao korisnički firmware
TTC_NetworkCfg.bin	64 kiB	RW	Korisnički mrežni parametar fajl. Napomena: Ovaj fajl mora biti prisutan prije nego default-ni firmware inicira tranziciju u RELOAD mode, da bi učitao korisnički firmware
TTC_Escfg.bin	192 kiB	RW	Mrežna konfiguracija krajnjeg sistema. Napomena: Ovaj fajl mora biti prisutan prije nego default-ni firmware inicira tranziciju u

			RELOAD mode, da bi učitao korisnički firmware
TTC_SWEcfg.bin	640 kiB	RW	Mrežna konfiguracija switch-a. Napomena: Ovaj fajl mora biti prisutan prije nego default-ni firmware inicira tranziciju u RELOAD mode, da bi učitao korisnički firmware
TTC_BITcfg.bin	64 kiB	RW	Korisnički BIT konfiguracioni fajl Napomena: Ovaj fajl mora biti prisutan prije nego default-ni firmware inicira tranziciju u RELOAD mode, da bi učitao korisnički firmware

Tabela 3: Pregled Default i User fajlova u NVM

Svaki fajl koji se nalazi u NVM memoriji, sadrži 16-bajtni header, sam sadržaj fajla, te SHA2-256 hash vrijednost. U tabeli 4 je pokazano kakav format ima svaki fajl.

Byte offset	Veličina u bajtovima	Parametar	Opis
Header fajla			
0x00	4	TAG	ASCII kod, može biti "BOOT", za fajlove koje učitava bootloader ili firmware, ili "DATA", za fajlove koje učitava isključivo firmware
0x04	4	Data size	N, koje odgovara veličini podataka, bez veličine header-a
0x08	4	Data CRC	Cyclic Redundancy Check podataka, koji se koristi za detekciju grešaka u podacima
0x0C	3	RSV	Rezervisano za kasniju upotrebu. Inicijalna vrijednost je 0x0000 0000
0x0F	1	Header CRC	CRC 15 byte-ova headera.
Sadržaj fajla			
0x10	N	Data content	Sadržaj fajla. N mora osigurati da se uzima u obzir maksimalna veličina fajla
0x10 + N	32	Data SHA256	SHA2-256 hash vrijednost sadržaja

Tabela 4. Format fajla u NVM memoriji

Fajlovi u NVM memoriji su zaštićeni na sljedeći način [40]:

- U svakom trenutku postoje dvije kopije fajla. Kada se vrši upload fajla u NVM, upload-ovani fajl se upisuje u prvu kopiju i provjerava se njegova ispravnost. Ukoliko je ispravan, upisuje se i u drugu kopiju i nakon toga su obe kopije iste i predstavljaju novi, upload-ovani fajl. Ukoliko je upload-ovani fajl neispravan, druga, stara kopija se ne mijenja, što osigurava da u NVM memoriji uvijek postoji jedna ispravna kopija.
- Svi fajlovi s ECC-zaštićeni. Elliptic Curve Cryptography algoritam dekodira po 4 bloka od 8 bitova. Ovaj algoritam koristi 72/64 Hamming kod, što znači da je 64-bitnom bloku podataka dodat 8-bitni kontrolni kod. Ukoliko je fajl ECC zaštićen, binarni podaci će biti dekodovani prije preuzimanja.

5.2.4 Primjena uređaja TTE-Switch Space 3U cPCI

Vremenski osjetljiv Ethernet može biti iskorišten kao jedinstveno mrežno rješenje u avionima. U ovom slučaju elektronska navigacija²⁷, sistem za navođenje²⁸ i pristup internetu na sjedištima putnika se mogu integrisati u jednu mrežu, i samim tim se može uštediti na cijeni i samoj težini pojedinačnih žica i mrežnih komponenti koje bi bile iskorištene [46]. Da bi se to omogućilo, vremenski osjetljiv Ethernet mora da garantuje particionisanje na nivou mreže, tačnije da kritične aplikacije dobiju potrebnu mrežnu uslugu, da bi obezbjedile određenu funkcionalnost i da različite aplikacije ne ometaju jedna drugu na bilo koji način [39]. Takođe, mehanizmi za rukovanje kvarovima trebaju da detektuju kvarove na vrijeme i spriječe njihovo propagiranje, te uticanje na druge mrežne uređaje, ali i da spriječe neovlaštene osobe da pristupaju mrežnih resursima za koja nemaju ovlaštenja.

Globalno vrijeme, opisano u poglavlju 5.1, doprinosi efikasnom iskorištavanju resursa na više načina. Vremenski osjetljiva komunikacija dopušta implementaciju mrežnih komponenti sa minimalnim memorijskim buffer-ima²⁹, budući da raspored vremenski osjetljive komunikacije oslobođen neusaglašenosti. Zbog toga, vremenski osjetljivi switch-evi ne moraju imati velike buffer prostore koji bi se iskoristili u slučaju da se desi da velika količina poruka mora biti isporučena preko istog fizičkog linka [39].

Koristeći koncept globalnog vremena, vremenski osjetljiv switch može da blokira saobraćaj koji generiše neispravna komponenta, ali i da spriječi poruke koje nisu poslate u odgovarajućem vremenskom trenutku. Ono što ide u prilog vremenski osjetljivom switch-u je i to da rade na tzv. fail-silent način, tj. ili funkcionišu na ispravan način, ili ne nude nikakvu funkcionalnost.

TTE-Switch-evi se mogu koristiti u sistemima za kontrolu leta za pokretanje određenih naredbi u naprijed određeno vrijeme tokom leta. Npr. mogu se koristiti za aktiviranje površina za kontrolu leta (nalaze se na krilima i repu aviona), podešavanje određenih parametara motora, izvršavanje unaprijed definisanih manevara leta ili podešavanje orijentacije svemirske letjelice.

U avio i svemirskoj industriji, sistemi za prikupljanje podataka se često oslanjaju na vremenski osjetljive switch-eve, za prikupljanje podataka putem senzora u preciznim vremenskim intervalima. Ti podaci su ključni za praćenje performansi različitih podsistema, izvođenje naučnih eksperimenata ili analiziranje uslova leta. Vremenski osjetljivi switch-evi osiguravaju da će podaci biti prikupljeni i preneseni do naučnika na Zemlji, u skladu sa zahtjevima same misije.

Vremenski osjetljivi switch-evi se koriste za detekciju grešaka i praćenje ispravnosti i statusa ugrađenih komponenti i podsistema. Oni periodično provjeravaju systemske parametre i izvršavaju dijagnostičke rutine, tako da mogu detektovati anomalije ili otkaze u ranim fazama i

²⁷ Elektronska navigacije – proces praćenja i kontrole kretanja letjelice koji se oslanja na tehnologiju koja se napaja električnom energijom

²⁸ Sistem za navođenje – virtuelni ili fizički uređaj ili grupa uređaja koji sprovode kontrolu kretanja broda, aviona, rakete, satelita ili bilo kog drugog pokretnog objekta. Navođenje je proces izračunavanja promjena položaja, brzine, visine i/ili brzine rotacije pokretnog objekta potrebnog za praćenje određene putanje i/ili visine profila na osnovu informacija o stanju kretanja objekta.

²⁹ Memorijski buffer – dio memorije koji se koristi za privremeno čuvanje podataka, u trenutku kada se podaci prenose sa jedne lokacije na drugu

olakšati izolaciju neispravnih komponenti, tako da se osigura sigurnost i pouzdanost letjelica.

Budući da otkazi mogu imati ozbiljne posljedice, redundancija je ključna. Vremenski osjetljivi switch-evi se često koriste u redundantnim sistemima radi pružanja rezervne funkcionalnosti u slučaju kvara primarnog sistema. Aktiviranjem redundantnih sistema u određeno vrijeme, svemirske letjelice mogu da nastave s radom, čak i ako su neke komponente u kvaru [46].

Efikasno upravljanje energijom je za svemirske letjelice od ključnog značaja, jer izvori energije, kao što su solarni paneli ili baterije, mogu biti ograničeni. U ovom slučaju se vremenski osjetljivi switch-evi koriste za kontrolu aktivacije i deaktivacije sistema i komponenti koje koriste veću količinu energije, i na taj način pomažu optimizaciju potrošnje energije i produžuju vrijeme trajanje neke misije.

Vremenski osjetljiv switch-evi mogu doprinjeti sigurnosti i pouzdanosti svemirskih misija izvršavanjem unaprijed definisanih naredbi u predefinisanim trenucima, i tako smanjuju rizik od ljudske greške i osiguravaju pravilno funkcionisanje sistema svemirskih letjelica.

5.3 Robot framework

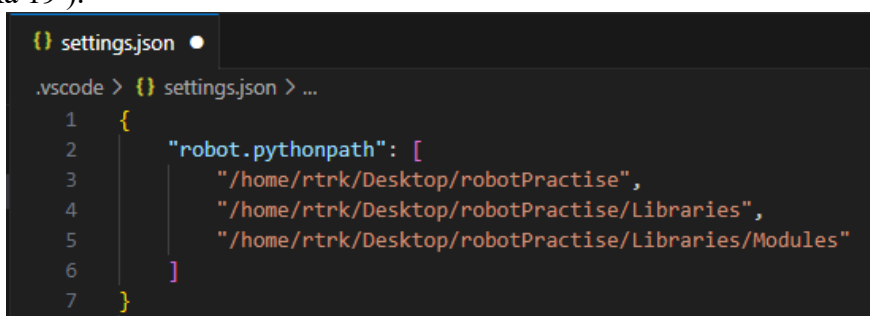
Kao što je pomenuto u poglavlju 4.3.2 Robot framework predstavlja nekomercijalni radni okvir koji je baziran ključnim riječima i koristi Python ili Java programski jezik. Razlog za odabir upravo Robot framework-a kao rješenja koje je prilagođeno za rad na projektu za diplomski rad upravo leži u tome što je Robot framework kao prvo, nekomercijalno, tj. besplatno rješenje, a kao drugo, nudi širok spektar mogućnosti kada je u pitanju automatizovanje procesa testiranja vremenski osjetljivog switch-a, čije testiranje je od interesa. Takođe, Robot framework nudi developerima mogućnost da kreiraju izvještaje koji će biti razumljivi i ljudima koji nemaju veliko tehničko znanje. Zbog svih ovih karakteristika, Robot framework je odabran kao rješenje koje će biti proučeno i upotrebljeno u praktičnom dijelu ovog diplomskog rada i u ovom poglavlju će biti opisane njegove karakteristike, mogućnosti i način na koji je implementiran praktički dio diplomskog rada.

Razlozi za korištenje Robot framework-a su zaista brojni [36]:

- Jednostavna tabelarna sintaksa pomoću koje se kreiraju testni slučajevi
- Omogućava rad sa ključnim riječima koje se mogu koristiti više puta, a nove se mogu kreirati na osnovu postojećih
- Može se koristiti na različitim platformama, može se koristiti i na Linux i na Windows operativnom sistemu
- Omogućava kreiranje testnih biblioteka koje proširuju njegove mogućnosti
- Omogućava testiranje web aplikacija, rest API-ja³⁰, mobilnih aplikacija, a moguće je koristiti Robot framework udaljeno pomoću SSH protokola³¹, što je urađeno u projektu koji je dio diplomskog rada
- Omogućen je rad sa varijablama, što daje mogućnost da se lako koristi u različitim okruženjima
- Podržava rad baziran na podacima (poglavlje 4.1.3) i modulima (poglavlje 4.1.1)

Iako Robot framework može biti baziran i na Java i na Python programskom jeziku, u ovom slučaju korišten je Python programski jezik i zato je neophodno instalirati isti na mašini na kojoj se framework koristi, te podesiti PATH i PYTHONPATH environment varijable³² tako da se u njima nalazi i informacija o source folderu projekta. Takođe, potrebno je instalirati Python Pip³³, standardni Python paket menadžer, kroz terminal, pomoću sljedeće komande: `sudo apt install python-pip`. Kao okruženje korišten je Visual Studio Code³⁴ i korišten je udaljen pristup mašini sa Linux operativnim sistemom pomoću SSH protokola. Instaliranje samog Robot frameworka je veoma jednostavno i vrši se kroz terminal na sljedeći način: `pip install robotframework`.

Da bi se Robot framework mogao koristiti kroz Visual Studio Code okruženju, neophodno je u projektom source folderu još kreirati i folder naziva `.vscode`, koji sadrži fajl naziva `settings.json` u kojem je potrebno u json formatu navesti putanju do source foldera samog projekta, ali i putanje do foldera svih modula koje je developer kreirao i koji se pozivaju u bibliotekama projekta (Slika 19).



```

1  {
2      "robot.pythonpath": [
3          "/home/rtrk/Desktop/robotPractise",
4          "/home/rtrk/Desktop/robotPractise/Libraries",
5          "/home/rtrk/Desktop/robotPractise/Libraries/Modules"
6      ]
7  }

```

Slika 19.

5.3.1 Testne biblioteke

Testne biblioteke nude Robot framework-u različite testne mogućnosti. Iako sam Robot framework ima ugrađene testne biblioteke kao što su [37]:

- BuiltIn – Automatski se importuje i sadrži ključne riječi koje se često koriste kao npr.: *Shoud Be Equal, Should Contain, Convert To Integer, Sleep ...*
- Collections – Sadrži ključne riječi koje se koriste za liste i kolekcije: *Append To List, Get*

³⁰ Rest API (takođe poznat pod nazivom RESTful API) – API ili web API koji je u skladu sa ograničenjima REST (Representational State Transfer) arhitektonskog stila i omogućava interakciju sa RESTful web servisima. Da bi se API smatrao kao RESTful mora postojati klijent-server arhitektura koju čine klijenti, serveri i resursi, kao i zahtjevi koji se odvijaju putem HTTP stateless protokola (stateless znači da je svaki zahtjev zaseban i ne sadrži informaciju o prethodnom zahtjevu).

³¹ Secure Shell (SSH) protokol – kriptografski mrežni protokol za bezbjedno slanje komandi nekom računaru putem nezaštićene mreže, kao što je internet. SSH koristi kriptografiju da autentifikuje i enkriptuje konekciju između dva uređaja. Često se koristi za udaljenu (remote) kontrolu servera i za prenos fajlova između dva računara.

³² Environment varijabla – vrijednost koju operativni sistem ili neki drugi softver može iskoristiti da bi došao do informacije koja je specifična za dati računar. Drugim riječima, to je vrijednost koja se odnosi na nešto, npr. lokacija nečega na računaru, broj verzije nekog softvera instaliranog na računaru, lista objekata...

³³ Python Pip – paket menadžer za Python pakete. Pip se koristi da bi se instalirali paketi (distribucija Python koda koja uključuje jedan ili više modula ili biblioteka. Mogu da sadrže module, podpakete (subpakete) i dodatne resurse kao što su dokumentacija i datoteke sa podacima) koji ne dođu instaliranjem Python-a.

³⁴ Visual Studio Code – besplatni editor koda koji je kreiran tako da se može koristiti na Windows operativnom sistemu, ali i na Linux-u, MacOS-u i Raspberry Pi operativnom sistemima. Dolazi sa ugrađenom podrškom za JavaScript, TypeScript i Node.js, ali nudi mogućnost ekstenzije za različite programske jezike (C++, C#, Java, Python, PHP), radne okvire (.Net i Unity), okruženja (Docker i Kubernetes) i oblake eng. clouds (Amazon Web Services, Microsoft Azure i Google Cloud Platform).

From Dictionary, List Should Be Equal ...

- **DateTime** – Omogućava kreiranje i verifikaciju datuma i vremena, kao i kalkulacije s njima. Često korištene ključne riječi su: *Get Current Date, Convert Time, Add Time to Time ...*
- **OperatingSystem** – Omogućava izvršavanje zadataka koji su vezani za operativni sistem. Često korištene ključne riječi ove biblioteke su: *Run, Create File, Remove Directorz, File Should Exist ...*
- **Process** – Podržava izvršavanje procesa i sistemu. Često korištene riječi su: *Run Process, Start Process, Wait For Process, Terminate Process ...*
- **Screenshot** – Obezbjeđuje ključne riječi za kreiranje snimka ekrana i čuvanje istog: *Set Screenshot Directory, Take Screenshot*
- **String** – Biblioteka koja omogućava manipulaciju stringovima i njihovu verifikaciju. Često korištene ključne riječi: *Should Be String, Split To Lines ...*
- **XML** – Biblioteka za verifikaciju i modifikaciju XML dokumenata. Često korištene ključne riječi: *Parse XML, Get Element, Save XML, Add Element ...*

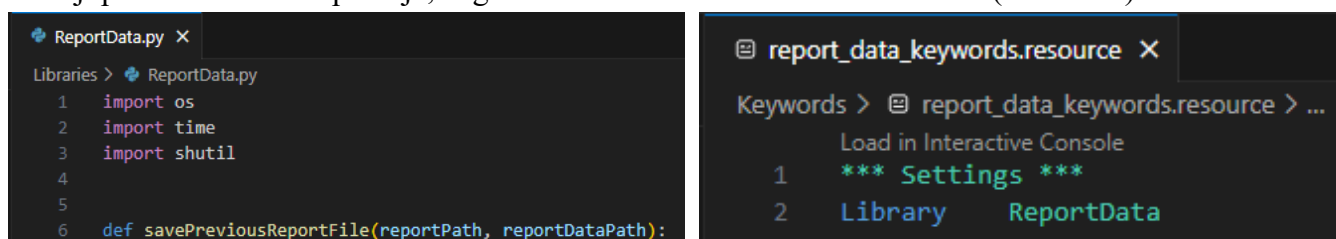
postoji potreba za kreiranjem novih biblioteka, kao u slučaju praktičnog dijela diplomskog rada.

Robot framework omogućava kreiranje tri tipa API-ja testnih biblioteka [36]:

- **Statički API** – ovaj pristup podrazumijeva kreiranje modula koji u sebi sadrže funkcije koje se kasnije mapiraju u ključne riječi ili kreiranje klasa sa metodama koje se mapiraju u ključne riječi koje se pozivaju u testnim slučajevima. Ukoliko funkcije ili metode imaju argumente, to znači da će ključnim riječima morati biti proslijeđeni isti argumenti.
- **Dinamički API** – prilikom korištenja ovog pristupa, neophodno je kreirati klase sa metodom koja može dobiti sve ključne riječi koje su implementirane unutar klase (npr. `get_keyword_names()`), ali i metodu koja će izvršiti željenu ključnu riječ sa odgovarajućim argumentima (npr. `run_keyword(keyword_name, arguments)`).
- **Hibridni API** – predstavlja hibrid prethodno opisanih tipova. Testna biblioteka je klasa sa metodom koja govori koje su ključne riječi implementirane unutar klase (`get_keywords()`), ali ne sadrži metodu koja izvršava neku ključnu riječ.

U praktičnom dijelu diplomskog rada odabran je statički pristup gdje su testne biblioteke implementirane kao moduli koji sadrže odgovarajuće funkcije, koje se zatim mapiraju u ključne riječi u odgovarajućim fajlovima.

Budući da je odabran pristup gdje se testne biblioteke implementiraju kao moduli, kada se kreira python modul `ReportData.py`, automatski se kreira testna biblioteka imena `ReportData` i može se importovati kao bilo koja druga ugrađena biblioteka. Tj. nije potrebno importovati sam fajl pomoću relativne putanje, nego samo navesti ime kreirane biblioteke (Slika 20).

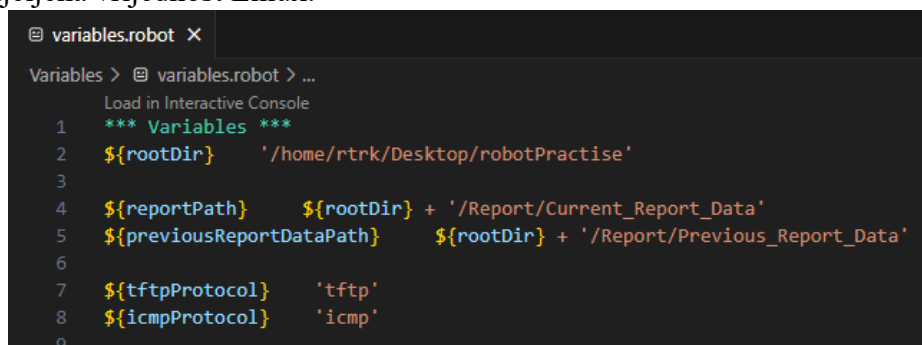


Slika 20. Primjer kreiranja modula `ReportData` sa metodom `savePreviousReportFile` i način importa kreirane biblioteke

5.3.2 Varijable

Varijable predstavljaju opciju u Robot framework-u koja se može koristiti na više mjesta, ali najčešće se koristi prilikom kreiranja ključnih riječi i testnih slučajeva. Najčešće se kreiraju u posebnom fajlu, sa ekstenzijom `.robot`, u formatu $\{naziv_varijable\} = vrijednost$ (Slika 21) i korisne su u sljedećim slučajevima [36] :

- Kada se određene vrijednosti koriste na više mjesta u projektu, varijable nude mogućnost da ukoliko dođe do promjene vrijednosti, ona se treba promijeniti na samo jednom mjestu
- Kada su vrijednosti u testnim slučajevima ili ključnim riječima dugačke, kao npr. `http://long.domain.name:8080/path/to/service?foo=1&bar=2&zap=42`, lakše je napisati $\{URL\}$
- Omogućeno je i dodavanje vrijednosti jedne varijable na drugu varijablu, kao npr.
 $\{rootDir\}$ `'/home/rtrk/Desktop/robotPractise'`
 $\{reportPath\}$ `$\{rootDir\} + '/Report/Current_Report_Data'$`
- Omogućeno je umjesto da se vrijednosti varijabli hard-koduju³⁵, da se vrijednosti varijabli postavljaju kroz terminal prilikom pokretanja samog procesa automatizovanog testiranja na sljedeći način: `robot --variable OS:Linux`. Na ovaj način je varijabli naziva `OS` dodijeljena vrijednost `Linux`.

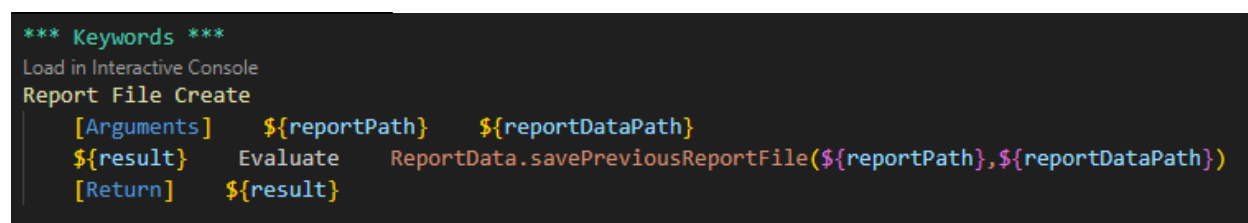


```

@ variables.robot X
Variables > @ variables.robot > ...
Load in Interactive Console
1 *** Variables ***
2 ${rootDir}      '/home/rtrk/Desktop/robotPractise'
3
4 ${reportPath}    ${rootDir} + '/Report/Current_Report_Data'
5 ${previousReportDataPath}  ${rootDir} + '/Report/Previous_Report_Data'
6
7 ${tftpProtocol}  'tftp'
8 ${icmpProtocol}  'icmp'
9

```

Slika 21. Primjer fajla koji sadrži varijable



```

*** Keywords ***
Load in Interactive Console
Report File Create
    [Arguments]    ${reportPath}    ${reportDataPath}
    ${result}      Evaluate    ReportData.savePreviousReportFile(${reportPath},${reportDataPath})
    [Return]       ${result}

```

Slika 22. Primjer upotrebe varijabli `reportPath` i `reportDataPath` u ključnoj riječi `Report File Create`

³⁵ Hard-kodovanje – način razvijanja softvera gdje se vrijednosti promjenljivih postavljaju direktno u source kodu, pa se te vrijednosti ne mogu promijeniti bez mijenjanja samog koda. Ovaj način razvoja čini kod manje fleksibilnim i teže ga je ponovo iskoristiti, ali ga je teže i održavati.

```

*** Test Cases ***
Run | Debug | Run in Interactive Console
Saving Previous Report Data
[Documentation]    Save report.html and log.html file created in directory Current_Report_Data,
...    when previous test was run, in directory Previous_Report_Data
${result}    Report File Create    ${reportPath}    ${previousReportDataPath}
Set Test Message    ${result}

```

Slika 23. Primjer upotrebe varijabli reportPath i reportDataPath u testnom slučaju Saving Previous Report Data

5.3.3 Ključne riječi

Ključne riječi se obično kreiraju u posebnim fajlovima sa ekstenzijom .resource. Sintaksa za kreiranje ključnih riječi je veoma jednostavna i sastoji se od sljedećeg: Neophodno je navesti u sekciji Settings odgovarajuće testne biblioteke iz kojih se pozivaju potrebne funkcije (Slika 24).

```

report_data_keywords.resource X
Keywords > report_data_keywords.resource > ...
Load in Interactive Console
1  *** Settings ***
2  Library    ReportData
3

```

Slika 24.

Zatim slijedi sekcija u kojoj se navode ključne riječi (Slika 25).

```

report_data_keywords.resource X
Keywords > report_data_keywords.resource > ...
Load in Interactive Console
1  *** Settings ***
2  Library    ReportData
3
4  *** Keywords ***

```

Slika 25.

Da bi se kreirala ključna riječ, neophodno je u Keywords sekciji navesti u prvom redu naziv ključne riječi. npr.

Report File Create

U drugom redu se navodi dokumentacija, koja nije obavezna. Ukoliko je dokumentacija dugačka, radi preglednosti, mogu se iskoristiti tri tačke (...) i prelazak u novi red, te nastaviti sa pisanjem dokumentacije:

```

[Documentation]    My documentation is too long and i am going to split it in two rows
...                like this

```

U narednom redu se navode Argumenti ukoliko ih ključna riječ, tačnije funkcija testne biblioteke koja se mapira u ključnu riječ, ima. Važno je naglasiti da nazivi argumenata navedenih u ključnim riječima ne moraju odgovarati nazivima varijabli, ali nazivi argumenata u ovom redu trebaju biti isti kao argumenti u narednom redu . Navodi se na sljedeći način:

[Arguments] argument1 argument2 ... argumentN

i moguće je umjesto vrijednosti argumenata koristiti varijable. Ukoliko funkcija testne biblioteke koja je mapirana u ključnu riječ nema argumente, dovoljno je navesti sljedeće:

[Arguments]

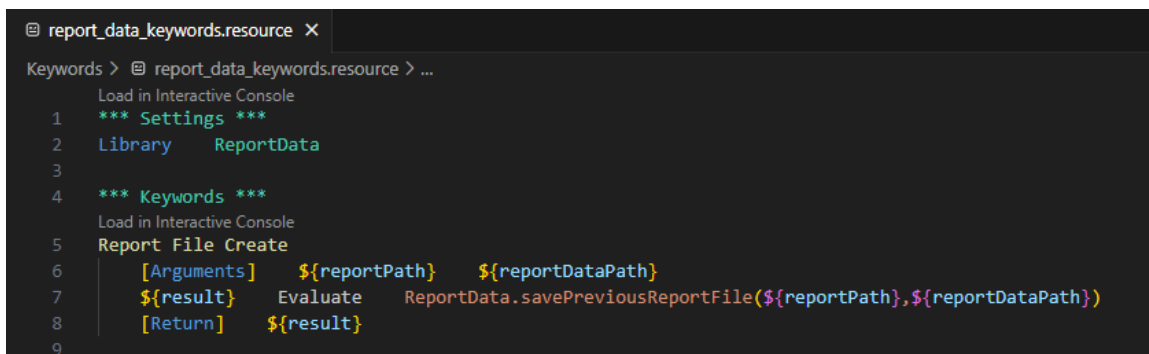
Zatim se prelazi u sljedeći red. U ovom redu se poziva funkcija odgovarajuće testne metode sa potrebnim argumentima. Ukoliko se želi ispisati rezultat funkcije u izvještaju, potrebno je sam rezultat izvršavanja funkcije dodijeliti *result* vrijednosti. To se vrši na sljedeći način:

\${result} Evaluate ReportData.savePreviousReportFile(\${reportPath},\${reportDataPath})

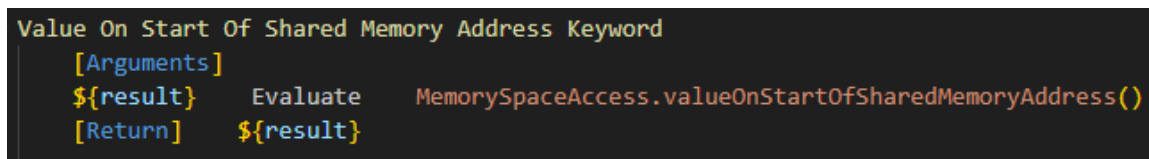
Evaluate je ključna riječ koja omogućava izvršavanje funkcije savePreviousReportFile testne biblioteke ReportData i vraćanje njenog rezultata.

Da bi se ispisao rezultat izvršavanja funkcije savePreviousReportFile, koji je dodijeljen vrijednosti result, potrebno je u narednom redu navesti sljedeće:

[Return] \${result}



Slika 26. Primjer ključne riječi Report File Create koja ima 2 argumenta



Slika 27. Primjer ključne riječi Value On Start Of Shared Memory Address Keyword koja nema argumente

5.3.4 Testni paketi

Testni paketi (eng. test suites) u Robot framework-u predstavljaju datoteke sa ekstenzijom .robot, koje mogu da sadrže jedan ili više testnih slučajeva. Testni slučajevi će prilikom pokretanja testnog paketa, izvršavati redoslijedom kojim su navedeni.

Ono što dodatno povećava mogućnost automatizacije procesa testiranja jeste to što Robot framework omogućava kreiranje većeg broja testnih paketa koji se nalaze u jednom folderu i sadrže

određen broj testnih slučajeva. Ukoliko postoji više testnih paketa u jednom folderu, dovoljno je u terminalu navesti naziv foldera i svi testni paketi će se izvršavati u alfabetskom redu.

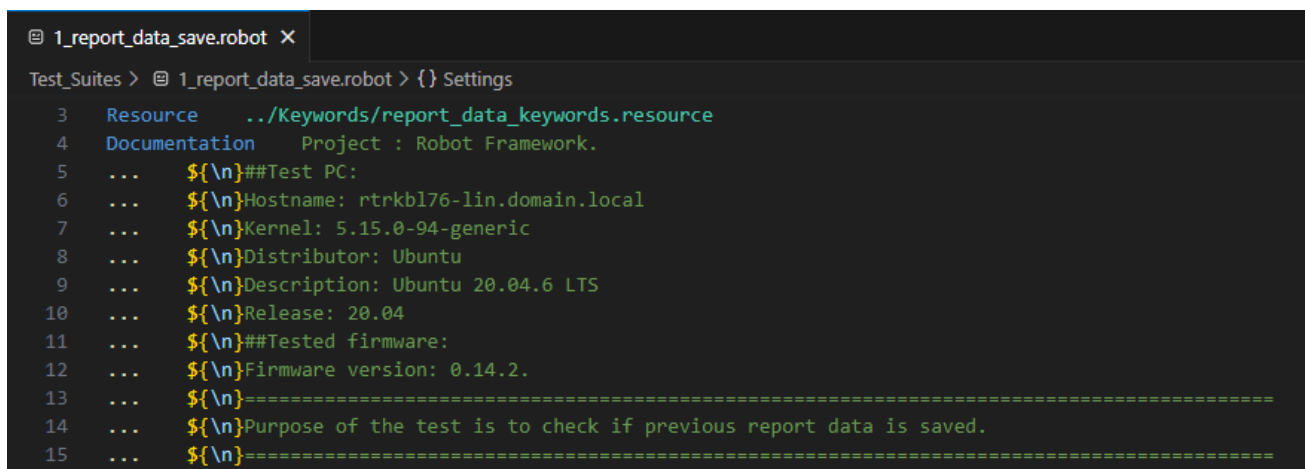
U fajlu koji predstavlja testni paket, neophodno je prvo navesti resurse, tačnije, relativne putanje do fajlova koji sadrže varijable i ključne riječi koje će se koristiti u testnim slučajevima unutar datog testnog paketa. Pomenuti resursi se navode na početku fajla, u sekciji Settings (Slika 28). Nakon resursa, u sekciji Settings, navodi se dokumentacija koja je relevantna za dati testni paket. Omogućeno je i pisanje određenih informacija u dokumentaciji u novom redu, pomoću sljedeće oznake: `${\n}` (Slika 28).

U sekciji Settings moguće je iskoristiti ugrađene ključne riječi Test Setup i Test Teardown, koje imaju svoje argumente, i predstavljaju aktivnosti koje se izvršavaju prije početka procesa testiranja, u slučaju Test Setup-a ili nakon procesa testiranja, u slučaju Test Teardown-a. Kao argument ovih ugrađenih ključnih riječi obično se proslijedi neka druga ključna riječ, ugrađena ili kreirana od strane developera. Ove dvije ključne riječi se u testnom paketu navode u sekciji Settings, na sljedeći način:

*** Settings ***

Test Setup Open Application App A

Test Teardown Close Application



```

1  Test Suites > @ 1_report_data_save.robot > {} Settings
2
3  Resource      ../Keywords/report_data_keywords.resource
4  Documentation  Project : Robot Framework.
5  ...           ${\n}##Test PC:
6  ...           ${\n}Hostname: rtrkbl76-lin.domain.local
7  ...           ${\n}Kernel: 5.15.0-94-generic
8  ...           ${\n}Distributor: Ubuntu
9  ...           ${\n}Description: Ubuntu 20.04.6 LTS
10 ...           ${\n}Release: 20.04
11 ...           ${\n}##Tested firmware:
12 ...           ${\n}Firmware version: 0.14.2.
13 ...           ${\n}=====
14 ...           ${\n}Purpose of the test is to check if previous report data is saved.
15 ...           ${\n}=====

```

Slika 28. Primjer navođenja resursa i dokumentacije u fajlu koji predstavlja testni paket

5.3.5 Testni slučajevi

Testni slučajevi se u Robot framework-u navode u okviru odgovarajućeg testnog paketa, jedan za drugim, redoslijedom kojim se trebaju izvršavati. Da bi se kreirao testni slučaj, neophodno je u sekciji Test Cases navesti u prvom redu naziv testnog slučaja, npr:

Saving Previous Report Data

U drugom redu se navodu dokumentacija vezana da dati testni slučaj, tačnije može se opisati šta se testnim slučajem tacno testira, šta se dešava prilikom testiranja ili šta je očekivani rezultat. To se navodi na sljedeći način. Ukoliko je dokumentacija dugačka, kao i prilikom kreiranja dokumentacije ključnih riječi, mogu se iskoristiti tri tačke (...) i prelazak u novi red, te se može nastaviti pisati dokumentacija:

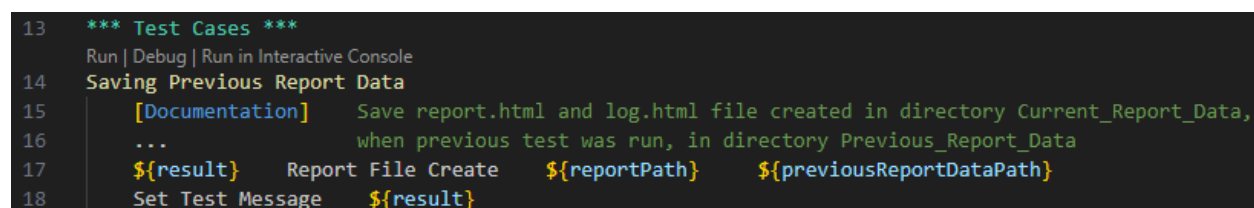
[Documentation] Save report.html and log.html file created in directory Current_Report_Data, when previous test was run, in directory Previous_Report_Data
...

Nakon dokumentacije, poziva se odgovarajuća ključna riječ. Ukoliko ključna riječ ima argumente, neophodno joj je iste proslijediti. Moguće je kao vrijednost argumenta proslijediti prethodno definisanu varijablu. Da bismo dobijeni rezultat mogli ispisati kao testnu poruku, potrebno je rezultat proslijediti *result* vrijednosti.

`\${result}` Report File Create `\${reportPath}` `\${previousReportDataPath}`

Moguće je result vrijednost dodijeliti vrijednosti testne poruke, na sljedeći način:

Set Test Message `\${result}`



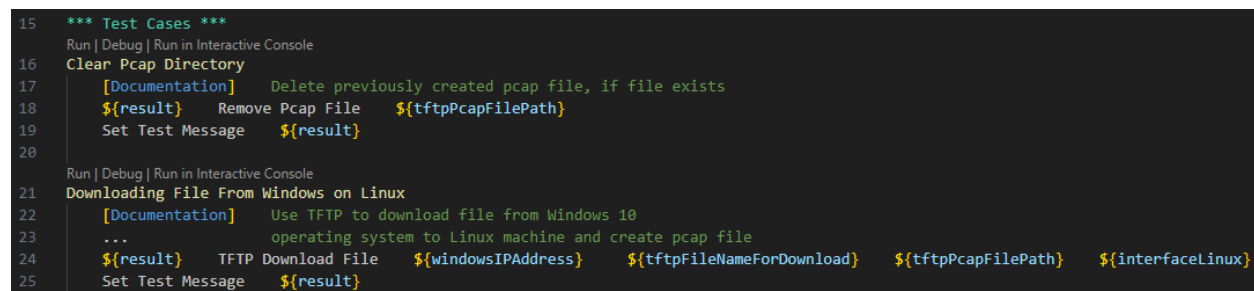
```

13  *** Test Cases ***
    Run | Debug | Run in Interactive Console
14  Saving Previous Report Data
15  [Documentation] Save report.html and log.html file created in directory Current_Report_Data,
16  ...           when previous test was run, in directory Previous_Report_Data
17  `${result}` Report File Create `${reportPath}` `${previousReportDataPath}`
18  Set Test Message `${result}`

```

Slika 29. Primjer izgleda jednog kreiranog testnog slučaja u sekciji Test Cases

Kreiranje sljedećeg testnog slučaja se vrši ispod posljednjeg kreiranog testnog slučaja, s tim da je potrebno ostaviti jedan prazan red između njih, kao na slici 30.



```

15  *** Test Cases ***
    Run | Debug | Run in Interactive Console
16  Clear Pcap Directory
17  [Documentation] Delete previously created pcap file, if file exists
18  `${result}` Remove Pcap File `${tftpPcapFilePath}`
19  Set Test Message `${result}`
20
    Run | Debug | Run in Interactive Console
21  Downloading File From Windows on Linux
22  [Documentation] Use TFTP to download file from Windows 10
23  ...           operating system to Linux machine and create pcap file
24  `${result}` TFTP Download File `${windowsIPAddress}` `${tftpFileNameForDownload}` `${tftpPcapFilePath}` `${interfaceLinux}`
25  Set Test Message `${result}`

```

Slika 30.

5.3.6 Tagovi

Tagovi predstavljaju opciju koja se može koristiti u ključnim riječima ili testnim slučajevima i prikazuju se u izvještajima koji su kreirani kada se proces testiranja završi.

Ukoliko je riječ o tagovima koji se koriste u ključnim riječima, mogu se koristiti da se ključne riječi, u izvještaju testnog procesa, pretražuju na osnovu njih i kreiraju se na jedan od tri načina [36]:

- Unutar Settings sekcije na sljedeći način:

**** Settings ****

Keywords Tags gui html

Ukoliko se na ovaj način navedu tagovi, oni će biti uključeni u sve ključne riječi.

- Unutar Keywords sekcije prilikom kreiranja ključne riječi, nakon dokumentacije na sljedeći način:

[Tags] gui html

Ukoliko se koristi onaj način navođenja tagova, oni će biti uključeni samo u datu ključnu riječ. Koristeći ovaj pristup moguće je i odstraniti neki tag iz željene ključne riječi, ukoliko su tagovi navedeni u sekciji Settings. To se radi na sljedeći način:

[Tags] -html

U ovom slučaju, tag html koji je naveden u sekciji Settings, neće biti uključen u željenu ključnu riječ.

- Ukoliko postoji dokumentacija unutar ključne riječi, tagovi se mogu navesti u posljednjem redu dokumentacije, na sljedeći način:

**** Keywords ****

My New Keyword

[Documentation] This is documentation

... Tags: gui, html

Ukoliko je riječ o tagovima koji se koriste u testnim slučajevima, njihova uloga može biti [36]:

- Izvor dodatnih informacija o testnom slučaju
- Prikazivanje statistike o testnim slučajevima, tj. ukupan broj testnih slučajeva, broj testnih slučajeva koji su prošli, pali i koji su testni slučajevi preskočeni.
- Isključivanje (exclude) ili uključivanje (include) testnih slučajeva, kao i preskakanje istih (skip) .

Tagovi koji se koriste u testnim slučajevima se mogu kreirati na sljedeće načine [36]:

- Unutar sekcije Settings na sljedeći način:

**** Settings ****

Test Tags myTag

U ovom slučaju će svi testni slučajevi unutar testnog paketa imati tag myTag

- Unutar sekcije Test Cases, prilikom kreiranja testnog slučaja, nakon naziva testnog slučaja i dokumentacije se navodi sljedeće:

[Tags] myTag

Na ovaj način je tag dodat samo određenom testnom slučaju. Moguće je na ovaj način i ukloniti tag koji je kreiran u sekciji Settings i primjenjen na sve testne slučaje, na sljedeći način:

[Tags] -myTag

- Pomoću komande *--settag* koja se navodi u terminalu. Na ovaj način se navedeni tag dodjeljuje svim testnim slučajevima u svim testnim paketima.
- Pomoću ugrađenih ključnih riječi *Set Tags*, *Remove Tags*, *Fail* i *Pass Executing*, koje omogućavaju manipulaciju tagovima prilikom izvršavanja testova, npr:

**** Test Cases ****

Set Tags And Remove Tags Keyword

[Documentation] This test already has tag 'smoke'

Set Tags example another

Remove Tag smoke

Korisnici mogu kreirati tagove bilo kakvog naziva i koji odgovaraju njihovim testnim slučajevima, međutim, u Robot framework-u postoje određeni tagovi koji imaju predefinisano

značenje i njihovo korištenje na način koji nije predviđen može izazvati neočekivane rezultate [36]. Svi ti specijalni tagovi koji postoje u Robot framework-u i koji će se u budućnosti kreirati, imaju prefiks *robot:*, tako da nije preporučljivo kreirati sopstvene tagove koji imaju ovaj prefiks. Neki od ovih tagova su [36]:

- *robot:continue-on-failure* i *robot:recursive-continue-on-failure* – Ukoliko je testni slučaj kreiran na način da ima više koraka i ukoliko jedan od tih koraka padne, ovaj tag omogućava da se i ostali koraci izvrše, nezavisno od toga što jedan korak nije prošao.
- *robot:stop-on-failure* i *robot:recursive-stop-on-failure* – Ovi tagovi se koriste da onemoguće prethodno pomenute tagove, ukoliko su oni iskorišteni.
- *robot:skip-on-failure* – Testni slučajevi će biti označeni kao preskočeni, ukoliko se desi da padnu.
- *robot:skip* – Testni slučajevi koji imaju ovaj tag, će biti preskočeni, ali će biti prisutni u izvještajima i označeni kao preskočeni.
- *robot:exclude* – Testni slučajevi koji imaju ovaj tag, neće biti izvršeni i neće biti prikazani u izvještajima i logovima.
- *robot:include* – Samo testni slučajevi koji imaju ovaj tag će biti izvršeni.
- *robot:exit* – U nekim slučajevima je potrebno da se proces testiranja zaustavi prije nego što su svi testovi izvršeni, ali je neophodno da izvještaj bude kreiran, koristi se ovaj tag. U ovom slučaju, za sve testove koji nisu izvršeni, smatra se da su pali.

5.3.7 Pokretanje procesa testiranja

Budući da je Robot framework radni okvir za automatizaciju procesa testiranja, on omogućava da se testni slučajevi, ali i testni paketi ne moraju pokretati jedan po jedan. Ukoliko postoji potreba da se veći broj testnih paketa pokrene jedan za drugim, ovaj radni okvir nudi tu opciju. To znači da tester neće morati čekati da se izvršavanje procesa testiranja jednog testnog paketa završi, da bi on mogao pokrenuti proces testiranja sljedećeg testnog paketa. Dovoljno je sve testne pakete smjestiti u jedan folder i pokrenuti proces testiranja. Na ovaj način će se testiranje testnih paketa vršiti sekvencijalno, po alfabetskom redu. Ukoliko testerima ne odgovara da se testiranje vrši alfabetskim redom, dovoljno je numerisati sve testne pakete po željenom redoslijedu.

Važno je još napomenuti da izvršavanje nekog dijela testnog slučaja nekada zahtijeva root privilegije³⁶ i da se to može desiti na više mjesta u različitim testnim paketima. Da bi se izbjeglo hard-kodovanje lozinke i njeno otkrivanje u kodu, Robot framework nudi rješenje. Dovoljno je instalirati SSH biblioteku³⁷ i tester može započeti automatizovani proces testiranja iz terminala sa root privilegijama. Instaliranje se vrši na sljedeći način: *pip install robotframework-SSHLibrary*.

³⁶ Root privilegije - privilegije koje ima root korisnik na Linux operativnom sistemu. Root korisnici mogu da urade sve što pozele, bez ograničenja. Ukoliko root korisnik napravi neku grešku, ona može biti katastrofalna i zato se nalog root korisnika koristi za neke administrativne svrhe, a većinu vremena se koristi korisnički nalog. Ukoliko korisnički nalog ima potrebu da izvrši radnju kojoj su neophodne root privilegije, potrebno je iskoristiti komandu *sudo*, nakon koje se zahtijeva navođenje lozinke root naloga.

³⁷ SSH biblioteka – biblioteka koja se u Robot framework-u koristi ukoliko je potrebno koristiti protokole kao što su SSH ili SFTP (SSH File Transfer Protocol).

Za pokretanje automatizovanog procesa testiranja potrebno se pozicionirati u source folder projekta i u terminalu ukucati sljedeće:

```
sudo robot -d Report/Current_Report_Data/ Test_Suites/
```

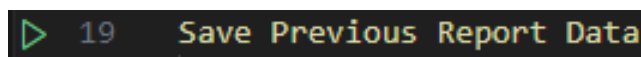
Gdje je -d komanda koja omogućava odabir specifičnog foldera za smještanje izvještaja koji se kreira nakon što se proces testiranja završi. Nakon toga se navodi relativna putanja do željenog foldera. Nakon toga se navodi relativna putanja do foldera u kojem se nalaze svi testni paketi koje tester želi da pokrene.

Dok teče proces testiranja, na terminalu će se ispisivati, u realnom vremenu, rezultati testiranja pojedinih testnih slučajeva iz odgovarajućih testnih paketa. Kada se proces testiranja završi, to će biti ispisano na terminalu i tester može da, u prethodno navedenom folderu za smještanje izvještaja procesa testiranja, pogleda detaljnije rezultate procesa testiranja.

Ukoliko tester ima potrebu da pokrene specifični testni paket, to može da uradi tako što će, umjesto da navede relativnu putanju do foldera sa svim test suite-ovima, navesti relativnu putanju do željenog testnog paketa. To može izvršiti na sljedeći način:

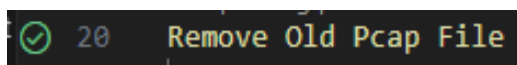
```
sudo robot -d Report/Current_Report_Data/ Test_Suites/1_report_data_save.robot
```

Pokretanje pojedinačnog testnog slučaja može se izvršiti klikom na zelenu oznaku koja se nalazi lijevo od naziva samog testnog slučaja (Slika 31), koji se nalazi u odgovarajućem testnom paketu.

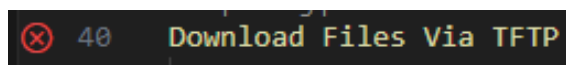


Slika 31.

U terminalu tester može da prati proces izvršavanja testnog slučaja, kao i krajnji rezultat njegovog izvršavanja, da li je testni slučaj pao ili prošao. Pored toga što rezultat izvršavanja testnog slučaja tester može da vidi u terminalu, rezultat izvršavanja testa se može vidjeti i pored naziva samog testnog slučaja, kao na slici 32. Važno je napomenuti da ukoliko tester na ovaj način pokreće testni slučaj, neće biti kreiran izvještaj za pokrenuti testni slučaj.



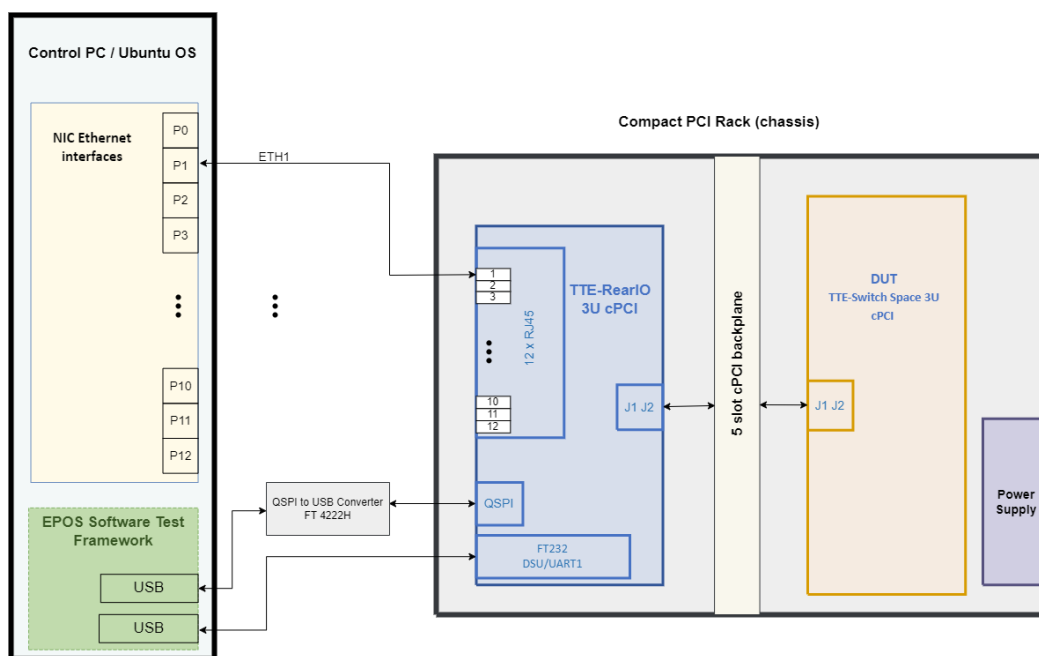
Slika 32.a) Testni slučaj prošao



Slika 32.b) Testni slučaj pao

5.4 Testiranje TTE-Switch Space 3U cPCI Firmware-a

U ovom poglavlju biće detaljno opisani kreirani testni paketi (eng. test suites) koji se koriste za testiranje TTE-Switch Space 3U cPCI Firmware-a. Testiranje je zasnovano na validaciji softverskih zahtjeva bez poznavanja detalja implementacije Firmware-a. Testno okruženje je prikazano na slici 33.



Slika 33. Testno okruženje

Hardware/Oprema	Svrha
Control PC	Računar sa instaliranim Linux operativnim sistemom koji je povezan sa uređajem koji se testira. Sa računara se pokreće framework koji omogućava kontrolu i monitoring svih funkcija koje su potrebne da bi se izvršile testne procedure
QSPI to USB Converter (FT4222H)	Konvertor USB interface-a u više kanalni SPI (sa 4 linije za prenos podataka)
UART to USB Converter (FT232R)	Konvertor USB interface-a u serijski UART interface
Compact PCI Rack (chassis)	CompactPCI šasija koja ima slobodan cPCI slot za TTE-Switch 3U cPCI uređaj i jedan slobodan cPCI slot za napajanje
TTE-RearIO 3U cPCI	Ploča koja obezbeđuje nekoliko fizičkih interface-a za korisnike. Posjeduje rJ2 konektor koji se može montirati u šasiju da bi se povezao sa J2 konektorom TTE-Switch 3U cPCI uređaja
cPCI Backplane	TTE-Switch Space 3U cPCI sadrži konektore J1 i J2 koji se koriste za povezivanje TTE-RearIO 3U cPCI ploče i cPCI Backplane-a da bi se omogućio pristup i kontrola svim interface-ima TTE-RearIO 3U cPCI ploče

Testni paketi se nalaze u folderu Test_Suites, njihova svrha i svi njihovi koraci, te očekivani ishodi.

Testni paketi kreirani u okviru ovog projekta su sljedeći:

- report_data_save
- icmp_report_check
- tftp_download
- safety_relevant_function
- tftp_upload_user_files
- tftp_upload_default_files
- snmp_firmware_version_check
- snmp_dsus_pin_state_check
- memory_content_check,

5.4.1 report_data_save

Cilj ovog testa je da provjeri da li je uspješno izvršeno čuvanje prethodno kreiranih fajlova report.html i log.html iz foldera Current_Report_Data, u folder Previuos_Report_Data.

Za potrebe testiranja ovog zahtjeva, kreiran je u folderu Libraries modul *ReportData.py*. U pomenutom modulu je kreirana funkcija *savePreviousReportFile*, u kojoj je implementirana sva logika vezana za kopiranje fajlova report.html i log.html iz foldera Current_Report_Data u odgovarajući folder, čiji naziv predstavlja datum i vrijeme kreiranja datih fajlova, unutar foldera Previuos_Report_Data.

Takođe, kreiran je i fajl *report_data.resource* u folderu Keywords. On sadrži ključnu riječ **Report File Create**, u okviru koje se poziva funkcija *savePreviousReportFile*, koja će biti iskorištena za kreiranje testnog paketa.

U okviru testnog paketa kreiran je testni slučaj **Save Previous Report Data** koji će pozvati ključnu riječ Report File Create i pomenute fajlove sačuvati u poseban folder u okviru foldera Previuos_Report_Data.

Npr. fajlovi report.html i log.html koji su kreirani 1. marta 2024. godine u 10:30:38 časova, moraju biti sačuvani u folderu Mar_1_2024_10:30:38, koji se nalazi u folderu Previuos_Report_Data.

5.4.2 icmp_request_check

ICMP Ping zahtjev se može iskoristiti za provjeru da li je neki uređaj sa specifičnom MAC/IP adresnom kombinacijom na specifičnoj lokaciji na mreži, tako što će se provjeriti da li je uređaj odgovorio na zahtjev.

Cilj ovog testa je da provjeri da li testirani hardware, tj. vremenski osjetljiv switch, odgovara na ICMP Ping zahtjev koji mu je upućen sa računara na kojem se izvršavaju testovi, za manje od 2 sekunde. Ovo se provjerava hvatanjem saobraćaja između računara i switch-a, kreiranjem pcap fajla na osnovu tog saobraćaja, u folderu Pcap_ICMP_File, te parsiranjem kreiranog fajla.

Za potrebe testiranja ovog zahtjeva, kreiran je u folderu Librarires modul *IcmpReplyCheck.py*, te fajl *icmp_reply_check.resource* u folderu Keywords.

U modulu *IcmpReplyCheck.py* su kreirane sljedeće funkcije:

Funkcija *deletePcapFile*, koja briše prethodno kreirani pcap fajl iz foldera Pcap_ICMP_File, ukoliko takav fajl postoji. Ukoliko u folderu ne postoji nikakav fajl, neće se desiti nikakvo ni brisanje, a odgovarajuća poruka će biti ispisana. U fajlu *icmp_reply_check.resource* kreirana je ključna riječ **Remove Pcap File**, koja poziva ovu funkciju.

Funkcija *performingPreInitializationStepsOFTheTestEnvironment*, koja vrši podešavanje ARP entry-a testne mašine tako da ima podatak o hardware-u koji se testira (dodavanje mrežnog interface-a, ip i mac adrese hardware-a). U fajlu *icmp_reply_check.resource* je kreirana ključna riječ **Performing Pre-Initialization**, koja poziva ovu funkciju.

Funkcija *getFirmwareIntoOperationalMode*, koja šalje reset komandu putem ft232 interface-a³⁸. Nakon toga potrebno je sačekati 10 sekundi, da bi firmware ušao u Operacioni mod. To se radi pomoću funkcije *sleep()* kojoj se proslijedi odgovarajući parametar koji naznačuje koliki je period *sleep-a*, iz modula *time*. Nakon što 10 sekundi istekne, firmware je u Operacionom modu. U fajlu *icmp_reply_check.resource* je kreirana ključna riječ ***Get Firmware Into Operational Mode***, koja poziva ovu funkciju.

Funkcija *createPcapFileBasedOnICMPRequest*, koja šalje 5 ICMP ping zahtjeva koji su upućeni hardware-u. Istovremeno se saobraćaj prisluškuje, hvata, i na osnovu njega se kreira fajl *icmpPcap.pcap* fajl u folderu *Pcap_ICMP_File*. U fajlu *icmp_reply_check.resource* je kreirana ključna riječ ***Create ICMP Requests***, koja poziva ovu funkciju.

Funkcija *createJsonFileFromPcapFile*, koja na osnovu fajla *icmpPcap.pcap* se kreira json fajl, *icmpJson.json*, koji će u sljedećem koraku služiti za parsiranje, tj. za utvrđivanje vremena koje je potrebno da hardware odgovori na ICMP ping zahtjev. U fajlu *icmp_reply_check.resource* je kreirana ključna riječ ***Json***, koja poziva ovu funkciju.

Funkcija *parseJsonFileIcmp*, koja vrši parsiranje kreiranog *icmpJson.json* fajla, tj. čitanje vremena odgovora za svih 5 poslatih ICMP ping zahtjeva, i njihovo ispisivanje. Ukoliko je vrijeme nekog odgovora veće od 2 sekunde, smatra se da test pada. U fajlu *icmp_reply_check.resource* je kreirana ključna riječ ***Parse Json File***, koja poziva ovu funkciju.

Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Remove Old Pcap File – testni slučaj poziva ključnu riječ Remove Pcap File, koja vrši brisanje prethodno kreiranog pcap fajla iz datog foldera.
2. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi Performing Pre-Initialization
3. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod pozivanjem ključne riječi Get Firmware Into Operational Mode
4. Create ICMP Requests – slanje 5 ICMP ping zahtjeva koji su upućeni hardware-u i istovremeno prisluškivanje saobraćaja, njegovo hvatanje, te kreiranje fajla *icmpPcap.pcap* u folderu *Pcap_ICMP_File*, pozivanjem ključne riječi Create ICMP Request.
5. Create Json file – na osnovu fajla *icmpPcap.pcap* kreiranje json fajla, *icmpJson.json*, pozivanjem ključne riječi Json.
6. Parse Json File – u posljednjem testnom slučaju se parsira kreirani *icmpJson.json* fajl, tj. čitaju vremena odgovora za svih 5 poslatih ICMP ping zahtjeva, i ispisuju se pozivanjem ključne riječi Parse Json File. Ukoliko je vrijeme nekog odgovora veće od 2 sekunde, smatra se da test pada.

³⁸ FT232 interface – Asinhroni serijski interface koji se koristi za razmjenu serijskih podataka između testiranog hardware-a i testnog računara. Preko njega se vrši flash-ovanje firmware-a, uključivanje/isključivanje DSU-a, čitanje statusnih informacija, slanje komandi za reset hardware-a...

5.4.3 tftp_download

TFTP je skraćenica koja se odnosi na Trivial File Transfer Protocol. TFTP je protokol koji omogućava transfer fajlova između uređaja u mreži, na sljedeći način:

- Uređaj koji ima namjeru započeti proces preuzimanja ili slanja fajla upotrebljava TFTP GET ili PUT komandu, tj. šalje Read ili Write zahtjev, u zavisnosti od toga da li se fajl preuzima ili šalje.
- Ukoliko je u pitanju TFTP Read zahtjev, uređaj kojem je upućen zahtjev treba da pošalje prvi blok podataka koji sadrži prvih 512 byte-ova fajla koji se preuzima od strane uređaja koji je poslao TFTP Read zahtjev. Nakon što je primio 512 byte-ova, neophodno je da uređaj pošalje potvrdu (eng. acknowledgment), to jest da dâ do znanja uređaju koji mu šalje podatke, da je primio prethodno poslate podatke. Ova razmjena podataka od 512 byte-ova, osim posljednjeg bloka podataka koji može sadržati manje od 512 byte-ova, i acknowledgment-a se odvija sve dok se posljednji byte fajla ne pošalje odgovarajućem uređaju.
- Ukoliko je u pitanju TFTP Write zahtjev, uređaj kojem je upućen zahtjev treba da pošalje potvrdu, tj. acknowledgment, da je primio zahtjev, prije nego što mu se paketi od 512 byte-ova počnu slati. Onog trenutka kada uređaj koji je poslao TFTP Write zahtjev dobije acknowledgment, on počinje slati po 512 byte-ova datog fajla, sve dok fajl u potpunosti ne bude poslat

Cilj ovog testa je da provjeri da li firmware dozvoljava download, tj. preuzimanje tzv. User i Default fajlova koji se nalaze u NVM (Non-Volatile Memory) pomoću TFTP protokola.

Za potrebe testiranja ovog zahtjeva kreiran je folder TFTP_Files, u koji će se smještati preuzeti Default-ni i User fajlovi, modul *TftpDownload.py* u folderu Libraries, te fajl *tftp_download.resource* u folderu Keywords.

U modulu TftpDownload.py su kreirane sljedeće funkcije:

Funkcija *clearMyDirectory*, koja vrši brisanje svih prethodno preuzetih Default-nih i User fajlova iz foldera TFTP_Files. Ukoliko u folderu ne postoji nikakav fajl, neće se desiti nikakvo ni brisanje, a odgovarajuća poruka treba biti ispisana. U fajlu *tftp_download.resource* je kreirana ključna riječ **Clear Directory**, koja poziva ovu funkciju.

Pomoćna funkcija *downloadFileViaTftp*, koja vrši download određenog fajla, putem TFTP protokola, u folder TFTP_Files.

Funkcija *downloadFilesViaTftp*, koja poziva pomoćnu funkciju *downloadFileViaTftp* za svaki Default-ni i User fajl, te provjerava da li su svi fajlovi preuzeti. U fajlu *tftp_download.resource* je kreirana ključna riječ **TFTP Download Files**, koja poziva ovu funkciju.

Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Clear Directory – testni slučaj briše sve prethodno preuzete Default-ne i User fajlove iz foldera TFTP_Files, pozivanjem ključne riječi Clear Directory.
2. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi Performing Pre-Initialization, kreirane u fajlu *icmp_reply_check.resource*.

3. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod pozivanjem ključne riječi Get Firmware Into Operational Mode, kreirane u fajlu `icmp_reply_check.resource`.
4. Download Files Via TFTP – preuzimanje Default-nih i User fajlova iz NVM, putem TFTP protokola, i njihovo smještanje u folder `TFTP_Files`, te provjera da li su svi fajlovi preuzeti, pozivanjem ključne riječi TFTP Download Files.

5.4.4 safety_relevant_function

Cilj ovog testa je da utvrdi da firmware dopušta izvršavanje sigurnosne funkcije kao što je TFTP upload, samo u intervalu od 10 sekundi, nakon što se izvrši `CMD_ARM` komanda.

U okviru testa izvršen je pokušaj TFTP upload-a User fajla `TTC_Firmware.bin` u NVM memoriju, ali nakon što je prošlo 10 sekundi od izvršavanja `CMD_ARM` komande i očekivano je da fajl neće biti upload-ovan. `CMD_ARM` komanda omogućava izvršavanje drugih komandi i upload-a. To je komanda koja mora biti izvršena prije neke sigurnosne funkcije, kao što je TFTP upload. Kada se izvrši `CMD_ARM` komanda i ukoliko prođe više od 10 sekundi nakon zadnje izdate komande nakon `CMD_ARM`, biće potrebno ponovo izvršiti `CMD_ARM` komandu.

Za potrebe testiranja ovog zahtjeva kreiran je modul *SafetyFunctionCheck.py* u folderu Library, te fajl *safety_function_check.resource* u folderu Keywords.

U modulu *SafetyFunctionCheck.py* su kreirane sljedeće funkcije:

Funkcija *uploadUserFileViaTFTP*, u okviru koje je izvršena `CMD_ARM` komanda, a zatim čekanje 10 sekundi pomoću funkcije `sleep`, iz modula `time`. Nakon isteka 10 sekundi, pokušao je TFTP upload-a User fajla `TTC_Firmware.bin` u NVM memoriju, za koji je očekivano da će biti neuspješan. U fajlu *safety_function_check.resource* je kreirana ključna riječ **Check Safety Relevant Function**, koja poziva ovu funkciju.

Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi Performing Pre-Initialization, kreirane u fajlu `icmp_reply_check.resource`.
2. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod pozivanjem ključne riječi Get Firmware Into Operational Mode, kreirane u fajlu `icmp_reply_check.resource`.
3. Check Safety Relevant Function – testni slučaj poziva ključnu riječ Check Safety Relevant Function, koja izvršava `CMD_ARM` komandu, a zatim čeka 10 sekundi. Nakon isteka 10 sekundi, pokušava TFTP upload User fajla `TTC_Firmware.bin` u NVM memoriju, za koji je očekivano da će biti neuspješan.

5.4.5 tftp_upload_default_files

Cilj ovog testa je da provjeri da li firmware dozvoljava upload Default-nih fajlova u NVM memoriju, koristeći TFTP protokol. Očekivani ishod je da firmware ne dozvoli upload ni jednog od datih Default-nih fajlova.

Razlog za očekivanje ovakvog ishoda testa je to što se Default-ni fajlovi nalaze u Read Only dijelu memorije i obični korisnik ne bi smio imati mogućnost da na isti način vrši upload Default-nih i User fajlova u NVM memoriju.

Za potrebe testiranja ovog zahtjeva kreiran je modul *TFTPUploadDefaultFiles.py* u folderu Libraries, te fajl *tftp_upload_default_files.resource* u folderu Keywords.

Da bi fajl mogao biti upload-ovan, on treba da bude odgovarajućeg formata, što je opisano u poglavlju 5.2.3. Svaki fajl koji će biti upload-ovani u ovom testnom paketu, se nalazi u posebnom Valid folderu, čija se putanja dobija pomoću funkcije *get_valid_file*, kojoj se proslijedi naziv željenog Default fajla, te ekstenzija fajla, koja je u ovom slučaju .tftp. Validni Default fajlovi koji se koriste za ovaj test su prethodno već kreirani i imaju odgovarajući format.

U modulu TFTPUploadDefaultFiles.py kreirana je funkcija *uploadDefaultFilesViaTFTP* koja za svaki Default-ni fajl treba da uradi sljedeće:

- Pronaći putanju do validnog fajla, pomoću funkcije *get_valid_file*
- Izvršiti CMD_ARM komandu. Ova komanda omogućava izvršavanje drugih komandi i komadne za upload, koja ja od interesa. Komanda se mora izvršiti prije nego upload započne. Kada se izvrši CMD_ARM komanda i ukoliko prođe više od 10 sekundi nakon zadnje izdate komande nakon CMD_ARM, biće potrebno ponovo izvršiti CMD_ARM komandu. Zato se ona izvršava prije upload-a svakog Default fajla, jer upload nekih fajlova traje više od 10 sekundi.
- Pokušaj TFTP upload-a Default-nog fajla. Očekivani ishod je da TFTP upload Default-nog fajla bude neuspješan.

U fajlu *tftp_upload_default_files.resource* je kreirana ključna riječ ***TFTP Upload Default Files***, koja poziva ovu funkciju.

Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi Performing Pre-Initialization, kreirane u fajlu *icmp_reply_check.resource*.
2. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod, pozivanjem ključne riječi Get Firmware Into Operational Mode, kreirane u fajlu *icmp_reply_check.resource*.
3. Upload Default Files Via TFTP – testni slučaj vrši pokušaj upload-a Default-nih fajlova, pozivanjem ključne riječi TFTP Upload Default Files. Očekivano je da upload svih Default-nih fajlova bude neuspješan.

5.4.6 tftp_upload_user_files

Cilj ovog testa je da provjeri da li firmware dozvoljava upload User fajlova u NVM memoriju, koristeći TFTP protokol.

Da bi fajl mogao biti upload-ovan, on treba da bude odgovarajućeg formata, što je opisano u poglavlju 5.2.3. Svaki fajl koji će biti upload-ovani u ovom testnom paketu, se nalazi u posebnom Valid folderu, čija se putanja dobija pomoću funkcije *get_valid_file()*, kojoj se proslijedi naziv željenog User fajla, te ekstenzija fajla, koja je u ovom slučaju .tftp. Validni User fajlovi koji se koriste za ovaj test su prethodno već kreirani i imaju odgovarajući format.

Za potrebe testiranja ovog zahtjeva kreiran je modul *TFTPUploadUserFiles.py* u folderu Libraries, te fajl *tftp_upload_user_files.resource* u folderu Keywords.

U modulu *TFTPUploadUserFiles.py* su kreirane sljedeće funkcije:

Pomoćna funkcija *cmd_arm*, koja izvršava *CMD_ARM* komandu.

Pomoćna funkcija *check_file_validity*, koja vrši validaciju ispravnosti upload-ovanog fajla. Validacija fajla se vrši nakon što se izvrši upload fajla u NVM memoriju. Razlog za to je mogućnost postojanja grešaka koje neće biti prijavljene prilikom samog procesa upload-a. Ovom prilikom se provjerava *fileTable.fileCopyValid* vrijednost MIB-a za obe kopije fajla pomoću SNMP protokola. Ukoliko je vrijednost 0, upload-ovani fajl je validan, ukoliko je 1, upload-ovani fajl nije validan. Firmware ne dozvoljava kopiranje neispravnog fajla u drugu kopiju, tako da će samo prva kopija biti neispravna, ukoliko je User fajl neispravan.

Funkcija *uploadUserFilesViaTFTP*, koja za svaki User fajl vrši sljedeće:

- Pronalazak putanje do validnog fajla, pomoću funkcije *get_valid_file* (poglavlje 5.4.5)
- Izvršavanje *CMD_ARM* komande (poglavlje 5.4.5), pozivanjem pomoćne funkcije *cmd_arm*
- TFTP upload User fajla
- Validacija upload-ovanog fajla, pozivanjem pomoćne funkcije *check_file_validity*

U fajlu *tftp_upload_user_files.resource* je kreirana ključna riječ ***TFTP Upload User Files***, koja poziva ovu funkciju.

Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi *Performing Pre-Initialization*, kreirane u fajlu *icmp_reply_check.resource*.
2. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod pozivanjem ključne riječi *Get Firmware Into Operational Mode*, kreirane u fajlu *icmp_reply_check.resource*.
3. Upload User Files Via Tftp – testni slučaj poziva ključnu riječ *TFTP Upload User Files*, koja vrši TFTP upload User fajlova i njihovu validaciju

5.4.7 snmp_firmware_version_check

SNMP je skraćenica koja se odnosi na Simple Network Management Protocol i predstavlja protokol koji prikuplja i organizuje informacije o uređajima na mreži, ali i mijenja te informacije da bi se određeno ponašanje promijenilo.

SNMP protokol omogućava korištenje SNMP GET i SNMP SET zahtjeva. Ovi zahtjevi se koriste u kombinaciji sa MIB (eng. Management Information Database) fajlom koja predstavlja kolekciju objekata kojima se može upravljati. Ovi objekti predstavljaju logičku reprezentaciju mrežnih komponenti kojima se može upravljati putem SNMP-a, kao što su računari, hub-ovi³⁹, router-i⁴⁰, switch-evi ili neki mrežni softver. MIB-ovi sadrže informacije o konfiguraciji ovih

³⁹ Hub – hardverski mrežni uređaj koji se koristi za konekciju više uređaja u jednoj mreži. Za razliku od switch-a nema mogućnost filtriranja saobraćaja i ne može odrediti destinaciju paketa, tako da paket šalje svim uređajima koji su povezani na njega.

⁴⁰ Router – hardverski mrežni uređaj koji se koristi za prosljeđivanje paketa između računarskih mreža

mrežnih komponenata, kao što je verzija softvera koji je pokrenut na komponenti, IP adresa ili broj porta. MIB-ovi su tekstualni fajlovi koji sadrže logička imena mrežnih resursa, a njihovim konfiguracionim parametrima se upravlja pomoću SNMP-a. Svaki MIB fajl i svaki unos u MIB fajlu imaju sopstveni objektni identifikator, tzv. OID (eng. Object Identifier).

Monitoring se obavlja pomoću SNMP GET zahtjeva i na taj način se dobijaju podaci o određenom OID-u ili MIB fajlu. Izdavanje komande se vrši pomoću SNMP SET zahtjeva za određeni OID u MIB tabeli.

Važno je napomenuti da se uređaj TTE-Switch Space 3U cPCI isporučuje sa MIB tekstualnim fajlovima koji sadrže OID-e, koji predstavljaju objekte kojima se može pristupati.

Cilj ovog testa je da provjeri da li firmware, dok je u operacionom stanju, može da vrati informaciju o svojoj verziji. Verziju vraća putem SNMP protokola, čitajući vrijednost iz odgovarajućeg OID-a. Da bi se utvrdilo da li je verzija zaista ispravna, potrebno je uporediti verziju koja je dobijena putem SNMP protokola i pročitati verziju iz konfiguracionog `environment_config.ini` fajla, koja sadrži informaciju o trenutnoj verziji firmware-a.

Za potrebe testiranja ovog zahtjeva kreiran je modul *FirmwareVersionCheck.py* u folderu Libraries, te fajl *snmp_firmware_version_check.resource* u folderu Keywords.

U modulu *FirmwareVersionCheck.py* kreirane su sljedeće funkcije:

Funkcija *retrieveFirmwareVersionViaSNMP*, koja vrši čitanje verzije firmware-a putem SNMP protokola iz odgovarajućeg OID-a. U fajlu *snmp_firmware_version_check.resource* je kreirana ključna riječ ***Get Firmware Version***, koja poziva ovu funkciju.

Funkcija *compareTwoFirmwareVersions*, koja vrši čitanje verzije firmware-a iz `environment_config.ini` fajla i upoređivanje verzije dobijene ovim putem i verzije firmware-a dobijene putem *retrieveFirmwareVersionViaSNMP* funkcije. U fajlu *snmp_firmware_version_check.resource* je kreirana ključna riječ ***Compare Firmware Versions***, koja poziva ovu funkciju.

Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi Performing Pre-Initialization, kreirane u fajlu *icmp_reply_check.resource*.
2. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod pozivanjem ključne riječi Get Firmware Into Operational Mode, kreirane u fajlu *icmp_reply_check.resource*.
3. Get Firmware Version Via SNMP – čitanje verzije firmware-a putem SNMP protokola iz odgovarajućeg OID-a, pozivajući ključnu riječ Get Firmware Version.
4. Compare Firmware Versions – čitanje verzije firmware-a iz `environment_config.ini` fajla i upoređivanje verzije dobijene ovim putem i verzije firmware-a dobijene u koraku broj 3, pozivajući ključnu riječ Compare Firmware Versions . Da bi se smatralo da je test prošao, obe verzije moraju biti iste.

5.4.8 snmp_dsu_pin_state_check

DSU (eng. Debug Support Unit) predstavlja hardware-sku komponentu, kojoj se pristupa preko ft232 interface-a, koju tester i mogu koristiti isključivo u svrhu debug-ovanja određenog hardware-a⁴¹. Preko DSU-a je moguće staviti procesor u debug mod i postavljati breakpoint-e, čitati sadržaj registara itd.

Debug mod se postiže tako što se DSU_EN pin postavi na visok nivo. Ukoliko se želi zaustaviti procesor, čitati neku vrijednost datog registra, pa zatim nastaviti izvršavanje, onda se mora prethodno uraditi DSU enable (DSU_EN = 1). Ako je DSU isključen, onda se preko UART-a mogu dobiti statusne informacije firmware-a, a kad je uključen, tada je procesor doveden u debug mode i nema ispisa statusnih informacija.

Cilj ovog testa je da provjeri da li firmware dok je u operacionom stanju, može provjeriti putem SNMP protokola da li je DSU (eng. Debug Support Unit) omogućen ili ne.

Za potrebe testiranja ovog zahtjeva kreiran je modul *SnmpDsuPinStateCheck.py* u folderu Libraries, te fajl *snmp_dsu_pin_state_check.resource* u folderu Keywords.

U modulu *SnmpDsuPinStateCheck.py* kreirane su sljedeće funkcije:

Funkcija *dsuPinToLow*, koja vrši podešavanje stanja DSU_EN pina na vrijednost nula, te provjera stanja DSU_EN pina putem ft232 interface-a i utvrđivanje da li je vrijednost jednaka nuli. U fajlu *snmp_dsu_pin_state_check.resource* kreirana je ključna riječ **Set Dsu Pin To Low**, koja poziva ovu funkciju.

Funkcija *checkDsuPinSnmpLow*, koja vrši provjeru stanja DSU_EN pina putem SNMP protokola i utvrđivanje da li je vrijednost nula, koristeći SNMP Get zahtjev, te odgovarajući OID. U fajlu *snmp_dsu_pin_state_check.resource* kreirana je ključna riječ **SNMP Dsu Pin State Check Low**, koja poziva ovu funkciju.

Funkcija *dsuPinToHigh*, koja vrši podešavanje stanja DSU_EN pina na vrijednost jedan, te provjera stanja DSU_EN pina putem ft232 interface-a i utvrđivanje da li je vrijednost jednaka jedan. U fajlu *snmp_dsu_pin_state_check.resource* kreirana je ključna riječ **Set Dsu Pin To High**, koja poziva ovu funkciju.

Funkcija *checkDsuPinSnmpHigh*, koja vrši provjeru stanja DSU_EN pina putem SNMP protokola i utvrđivanje da li je vrijednost jedan, koristeći SNMP Get zahtjev, te odgovarajući OID. U fajlu *snmp_dsu_pin_state_check.resource* kreirana je ključna riječ **SNMP Dsu Pin State Check High**, koja poziva ovu funkciju.

Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi Performing Pre-Initialization, kreirane u fajlu *icmp_reply_check.resource*.
2. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod pozivanjem ključne riječi Get Firmware Into Operational Mode, kreirane u fajlu *icmp_reply_check.resource*.

⁴¹ Debug-ovanje hardware-a – postupak prepoznavanja i rješavanja problema ili grešaka u hardware-skim komponentama. U okviru debug-vanja hardware-a traže se hardware-ske komponente koje nisu ispravno instalirane ili konfigurisane

3. Set Dsu Pin To Low – podešavanje stanja DSU_EN pina na vrijednost nula, te provjera stanja DSU_EN pina, pozivajući ključnu riječ Set Dsu Pin To Low
4. SNMP Dsu Pin State Check Low – provjera stanja DSU_EN pina putem SNMP protokola i utvrđivanje da li je vrijednost nula, pozivajući ključnu riječ SNMP Dsu Pin State Check Low
5. Set Dsu Pin To High – podešavanje stanja DSU_EN pina na vrijednost jedan, te provjera stanja DSU_EN pina, pozivajući ključnu riječ Set Dsu Pin To High
6. SNMP Dsu Pin State Check High – provjera stanja DSU_EN pina putem SNMP protokola i utvrđivanje da li je vrijednost jedan, pozivajući ključnu riječ SNMP Dsu Pin State Check High

5.4.9 memory_content_check

Cilj ovog testa je da provjeri sadržaj specifičnih memorijskih lokacija. Adresa početka dijeljene memorije⁴² (eng. Shared memory) se upisuje na adresu 0x2000 1000 u RAM memoriji i ona se ne mijenja bez obzira na to da li se mijenja verzija firmware-a.

Za potrebe testiranja ovog zahtjeva kreiran je modul *MemorySpaceAccess.py* u folderu Libraries, te fajl *memory_space_access.resource* u folderu Keywords.

U modulu *MemorySpaceAccess.py* kreirane su sljedeće funkcije:

Funkcija *testStartOfSharedMemory*, koja vrši provjeru sadržaja na adresi 0x2000 1000, pomoću ft232 interface-a. Sadržaj na pomenutoj memorijskoj adresi treba da predstavlja početak dijeljene memorije, tačnije na toj adresi treba da bude vrijednost 0x2005 F400. U fajlu *memory_space_access.resource* kreirana je ključna riječ ***Start Of Shared Memory Keyword***, koja poziva ovu funkciju.

Funkcija *valueOnStartOfSharedMemoryAddress*, koja vrši provjeru sadržaja na adresi 0x2005 F400 pomoću ft232 interface-a, koja predstavlja početak dijeljene memorije. Vrijednost na toj adresi treba da bude 0x534D 0300, a to predstavlja tzv. Interface Version Number. Ova vrijednost predstavlja neku vrstu identifikatora fajla koji se definiše da bi se mogao provjeriti tip ili namjena fajla. U fajlu *memory_space_access.resource* kreirana je ključna riječ ***Value On Start Of Shared Memory Address Keyword***, koja poziva ovu funkciju.

Funkcija *valueForOffset0x0004*, koja vrši 0x0004 offset-a na početak dijeljene memorije i provjera sadržaja pomoću ft232 interface-a. Vrijednost na toj adresi treba da bude 0x8765 4321, a to predstavlja konstantnu vrijednost. U fajlu *memory_space_access.resource* kreirana je ključna riječ ***Value For Offset 0x0004 Keyword***, koja poziva ovu funkciju.

Funkcija *valueForOffset0x0008*, koja vrši dodavanje 0x0008 offset-a na početak dijeljene memorije i provjera sadržaja pomoću ft232 interface-a. Vrijednost na toj adresi treba da bude 0x4550 0101, a to predstavlja tzv. Firmware Type Identifier. U fajlu *memory_space_access.resource* kreirana je ključna riječ ***Value For Offset 0x0008 Keyword***, koja poziva ovu funkciju.

⁴² Dijeljena memorija – memorija kojoj istovremeno može pristupiti više procesa s namjerom da se omogući komunikacija među njima i da se spriječe suvišne kopije. Dijeljena memorija predstavlja efikasan način za prenos podataka između različitih programa. Zavisno od konteksta, programi se mogu izvršavati na istim ili različitim uređajima.

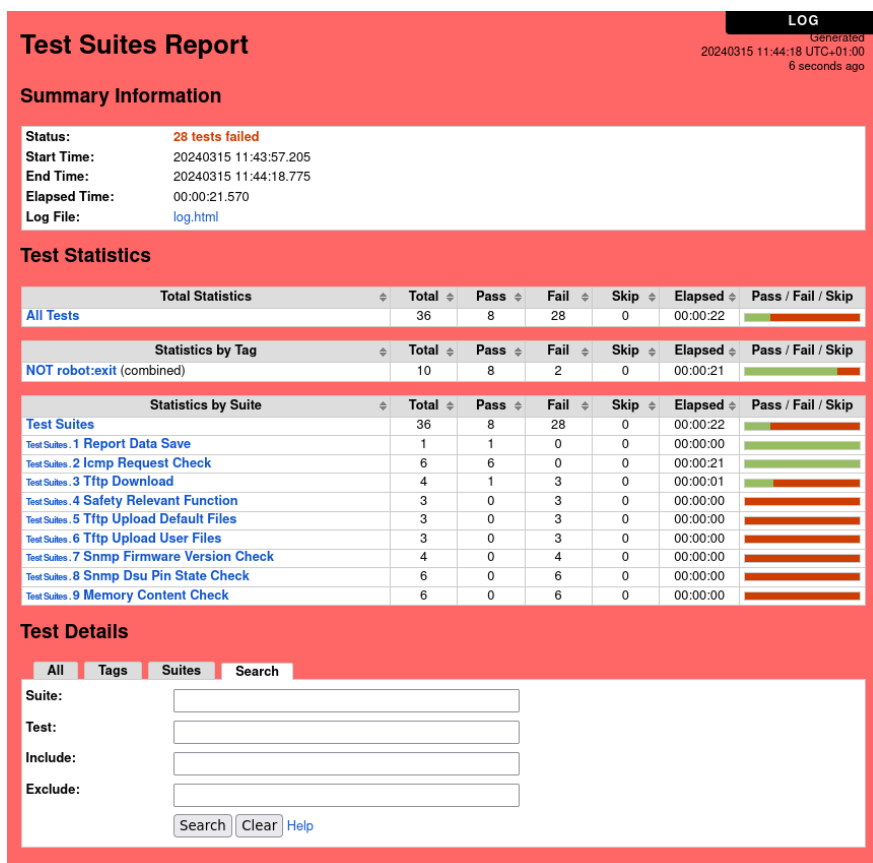
Testni slučajevi koji su sastavni dio ovog testnog paketa:

1. Pre-Initialization – izvršavanje preinicijalizacije testnog okruženja, pozivanjem ključne riječi Performing Pre-Initialization, kreirane u fajlu icmp_reply_check.resource.
2. Get Firmware Into Operational Mode – dovođenje firmware-a u Operacioni mod pozivanjem ključne riječi Get Firmware Into Operational Mode, kreirane u fajlu icmp_reply_check.resource.
3. Start Of Shared Memory Test – provjera sadržaja na adresi 0x2000 1000 pozivanjem ključne riječi Start Of Shared Memory Keyword . Sadržaj na pomenutoj memorijskoj adresi treba da predstavlja početak dijeljene memorije, tačnije na toj adresi treba da bude vrijednost 0x2005 F400.
4. Value On Start Of Shared Memory Address Test – provjera sadržaja na adresi 0x2005 F400 pozivanjem ključne riječi Value On Start Of Shared Memory Address Keyword, koja predstavlja početak dijeljene memorije. Vrijednost na toj adresi treba da bude 0x534D 0300, a to predstavlja tzv. Interface Version Number.
5. Value For Offset 0x0004 Test – dodavanje 0x0004 offset-a na početak dijeljene memorije i provjera sadržaja pozivanjem ključne riječi Value For Offset 0x0004 Keyword. Vrijednost na toj adresi treba da bude 0x8765 4321, a to predstavlja konstantnu vrijednost.
6. Value For Offset 0x0008 Test – dodavanje 0x0008 offset-a na početak dijeljene memorije i provjera sadržaja pozivanjem ključne riječi Value For Offset 0x0008 Keyword. Vrijednost na toj adresi treba da bude 0x4550 0101, a to predstavlja tzv. Firmware Type Identifier.

5.4.10 Izvještaji procesa testiranja

Nakon što se automatizovani proces testiranja završi, izvještaji će biti kreirani u folderu Current_Report_Data. Robot framework kreira u pomenutom folderu report.html i log.html fajlove. Ovi fajlovi nude detaljan uvid u rezultate testiranja i poprilično su jednostavni za razumijevanje, tako da i osoba koja nema obimno znanje iz oblasti koja je obuhvaćena procesom testiranja, može da razumije šta se testira, šta se očekuje da bude rezultat testiranja, šta je stvarni rezultat testiranja, zatim koji je procenat testova koji su pali, a i na jednostavan način može da pretražuje testne pakete i testne slučajeve unutar njih.

Ukoliko tester namjerno zaustavi proces testiranja, prije nego što je on završen, i report.html i log.html fajlovi će biti kreirani, s tim što će se smatrati da su pali svi testovi koji nisu stigli da se izvrše (Slika 34). Fajl će izgledati slično i ako samo jedan testni slučaj padne.



Slika 34. Report.html fajl ukoliko neki od testnih slučajeva nije prošao



Slika 35. Report.html fajl kada su svi testni slučajevi svih testnih paket prošli

Na slici 35 prikazano je kako izgleda fajl report.html kada su svi testni slučajevi prošli.

U sekciji Summary Information je prikazano da li su svi testovi prošli, ili u slučaju da je neki testni slučaj pao, biće prikazano koliko je testnih slučajeva palo. Prikazano je kad je proces testiranja počeo i kada je završen, te koliko je ukupno vremena proces testiranja trajao.

Test Statistics sekcija daje više informacija o samim testnim paketima. Može se vidjeti koliko je testnih slučajeva prošlo, koliko palo, te su prikazani nazivi testnih paketa, koji predstavljaju linkove ka opširnijim informacijama.

U sekciji Test Details moguće je vršiti pretragu testnih slučajeva i testnih paketa po određenim parametrima.

Klikom na link iz sekcije Statistics by Suite, koji predstavlja naziv nekog testnog paketa, dobijaju se detaljne informacije, tj sekcija Test Details (Slika 36), kao što su: broj testnih slučajeva unutar paketa koji su prošli i koji su pali, ukupan broj testnih slučajeva, dokumentacija testnog paketa, vrijeme izvršavanja svakog testnog slučaja (Elapsed sekcija), te dokumentacija svakog testnog slučaja, tj. šta se testnim slučajem treba testirati i koji su očekivani rezultati, a kakav je rezultat testiranja svakog testnog slučaja (Message sekcija).

Test Details					
<div> <div>AllTagsSuitesSearch</div> <div> <div>Suite:</div> <div>Test Suites.3 Tftp Download</div> </div> <div> <div>Status:</div> <div>4 tests total, 4 passed, 0 failed, 0 skipped</div> </div> <div> <div>Documentation:</div> <div> Project : Robot Framework. ##Test PC: Hostname: rtrkb176-lin.domain.local Kernel: 5.15.0-94-generic Distributor: Ubuntu Description: Ubuntu 20.04.6 LTS Release: 20.04 ##Tested firmware: Firmware version: 0.14.2. Purpose of the test is to check if firmware allows download of Default and User files from NVM (Non-Volatile Memory) using TFTP protocol. Default files that need to be downloaded: TTC_FirmwareDft.bin, TTC_FWparamDft.bin, TTC_NetworkCfgDft.bin, TTC_EScglDft.bin, TTC_SWEcglDft.bin, TTC_BITClgDft.bin, TTC_FWloader.bin, TTC_ProductData.bin User files that need to be downloaded: TTC_Firmware.bin, TTC_FWparam.bin, TTC_NetworkCfg.bin, TTC_EScgl.bin, TTC_SWEcgl.bin, TTC_BITClg.bin Start / End Time: 20240326 09:14:30.815 / 20240326 09:14:55.055 Elapsed Time: 00:00:24.240 Log File: log.html#s1-s3 </div> </div> </div>					
Name	Documentation	Status	Message	Elapsed	Start / End
Test Suites.3 Tftp Download. Clear Directory	Delete all files previously downloaded in directory TFTP_Files	PASS	Directory for storing default and user files is emptied	00:00:00.002	20240326 09:14:30.821 20240326 09:14:30.823
Test Suites.3 Tftp Download. Pre-Initialization	Perform pre-initialization steps of the test environment	PASS	- EPOS interface enx7cc2c64758ab has IP address 10.0.1.37 Clearing following ip addresses: Clearing EPOS IP address. Setting following ip addresses: Setting EPOS IP address. Setting following ARP entries: Setting arp entry for EPOS	00:00:01.017	20240326 09:14:30.823 20240326 09:14:31.840
Test Suites.3 Tftp Download. Get Firmware Into Operational Mode	Get Firmware into Operational mode	PASS	Firmware is in OPERATIONAL mode	00:00:11.569	20240326 09:14:31.840 20240326 09:14:43.409
Test Suites.3 Tftp Download. Download Files Via TFTP	Download all necessary default and user files	PASS	TFTP download of file TTC_FirmwareDft.bin successful TFTP download of file TTC_FWparamDft.bin successful TFTP download of file TTC_NetworkCfgDft.bin successful TFTP download of file TTC_EScglDft.bin successful TFTP download of file TTC_SWEcglDft.bin successful TFTP download of file TTC_BITClgDft.bin successful TFTP download of file TTC_FWloader.bin successful TFTP download of file TTC_ProductData.bin successful TFTP download of file TTC_Firmware.bin successful TFTP download of file TTC_FWparam.bin successful TFTP download of file TTC_NetworkCfg.bin successful TFTP download of file TTC_EScgl.bin successful TFTP download of file TTC_SWEcgl.bin successful TFTP download of file TTC_BITClg.bin successful All default and user files downloaded successfully	00:00:11.643	20240326 09:14:43.411 20240326 09:14:55.054

Slika 36. Report.html fajl

Imena testnih slučajeva u sekciji Name sa lijeve strane predstavljaju linkove, koji preusmjeravaju korisnike na fajl log.html, i to na dio tog fajla koji će pružiti više informacija o samom testnom slučaju čiji link je odabran, kao na slici 37.

SUITE 3 Tftp Download 00:00:00.002 **REPORT**

Full Name: Test Suites.3 Tftp Download
Documentation: Project : Robot Framework.
 ##Test PC:
 Hostname: rtrkbl76-lin.domain.local
 Kernel: 5.15.0-94-generic
 Distributor: Ubuntu
 Description: Ubuntu 20.04.6 LTS
 Release: 20.04
 ##Tested firmware:
 Firmware version: 0.14.2.

=====

Purpose of the test is to check if firmware allows download of Default and User files from NVM (Non-Volatile Memory) using TFTP protocol.
 Default files that need to be downloaded: TTC_FirmwareDft.bin, TTC_FWparamDft.bin, TTC_NetworkCfgDft.bin, TTC_ESCfgDft.bin, TTC_SWECfgDft.bin, TTC_BITCfgDft.bin, TTC_FWloader.bin, TTC_ProductData.bin
 User files that need to be downloaded: TTC_Firmware.bin, TTC_FWparam.bin, TTC_NetworkCfg.bin, TTC_ESCfg.bin, TTC_SWECfg.bin, TTC_BITCfg.bin

=====

Source: /home/rtrk/Desktop/robotPractise/Test_Suites/3_tftp_download.robot
Start / End / Elapsed: 20240326 09:14:30.815 / 20240326 09:14:55.055 / 00:00:24.240
Status: 4 tests total, 4 passed, 0 failed, 0 skipped

Test Name	Time
TEST Clear Directory	00:00:00.002
TEST Pre-Initialization	00:00:01.017
TEST Get Firmware Into Operational Mode	00:00:11.569
TEST Download Files Via TFTP	00:00:11.643

Full Name: Test Suites.3 Tftp Download.Download Files Via TFTP
Documentation: Download all necessary default and user files
Start / End / Elapsed: 20240326 09:14:43.411 / 20240326 09:14:55.054 / 00:00:11.643
Status: **PASS**
Message: TFTP download of file TTC_FirmwareDft.bin successful
 TFTP download of file TTC_FWparamDft.bin successful
 TFTP download of file TTC_NetworkCfgDft.bin successful
 TFTP download of file TTC_ESCfgDft.bin successful
 TFTP download of file TTC_SWECfgDft.bin successful
 TFTP download of file TTC_BITCfgDft.bin successful
 TFTP download of file TTC_FWloader.bin successful
 TFTP download of file TTC_ProductData.bin successful
 TFTP download of file TTC_Firmware.bin successful
 TFTP download of file TTC_FWparam.bin successful
 TFTP download of file TTC_NetworkCfg.bin successful
 TFTP download of file TTC_ESCfg.bin successful
 TFTP download of file TTC_SWECfg.bin successful
 TFTP download of file TTC_BITCfg.bin successful

All default and user files downloaded successfully

KEYWORD \${result} = tftp_download... **TFTP Download Files** \${tftpFilePath} 00:00:11.639

Slika 37. Log.html fajl

TEST Start Of Shared Memory Test 00:00:02.607

Full Name: Test Suites.9 Memory Content Check.Start Of Shared Memory Test
Documentation: Read value from address 0x20001000. Value represents start of shared memory.
Start / End / Elapsed: 20240328 09:30:45.906 / 20240328 09:30:48.513 / 00:00:02.607
Status: **PASS**
Message: Value on address 0x20001000: 0x2005f400, i.e. Start of shared memory

KEYWORD \${result} = memory_space_access... **Start Of Shared Memory Keyword** \${configFile} 00:00:02.603

Start / End / Elapsed: 20240328 09:30:45.909 / 20240328 09:30:48.512 / 00:00:02.603

KEYWORD \${result} = BuiltIn... **Evaluate** MemorySpaceAccess.testStartOfSharedMemory(\${configFile}) 00:00:02.601

Documentation: Evaluates the given expression in Python and returns the result.
Start / End / Elapsed: 20240328 09:30:45.910 / 20240328 09:30:48.511 / 00:00:02.601
 09:30:48.511 **INFO** \${result} = Value on address 0x20001000: 0x2005f400, i.e. Start of shared memory
 09:30:48.512 **INFO** \${result} = Value on address 0x20001000: 0x2005f400, i.e. Start of shared memory

KEYWORD BuiltIn... **Set Test Message** \${result} 00:00:00.001

Documentation: Sets message for the current test case.
Start / End / Elapsed: 20240328 09:30:48.512 / 20240328 09:30:48.513 / 00:00:00.001
 09:30:48.513 **INFO** Set test message to:
 Value on address 0x20001000: 0x2005f400, i.e. Start of shared memory

Slika 38. Prikaz detalja testnog slučaja Start Of Shared Memory Test u fajlu log.html

Na slici 38. je prikazan izgled sekcije log.html fajla, za testni slučaj Start Of Shared Memory Test, iz testnog paketa memory_content_check.

Ispod samog naziva testnog slučaja nalazi se sekcija Full Name. U ovom slučaju vidi se naziv foldera u kojem se taj testni slučaj nalazi (Test Suites), zatim naziv testnog paketa (9 Memory Content Check), te naziv testnog slučaja (Start Of Shared Memory Test). U folderu Test Suites testni slučajevi su numerisani, pa se iz tog razloga vidi broj 9.

Documentation sekcija ispod predstavlja dokumentaciju testnog slučaja.

Start/End/Elapsed sekcija pruža informacije o tome kada je testni slučaj započeo izvršavanje (Start), kada je izvršavanje završeno (End), te koliko je trajao proces izvršavanja ovog testnog slučaja (Elapsed).

Status sekcija daje informaciju o tome da li je testni slučaj prošao ili je pao (pass/fail).

Message sekcija predstavlja poruku koja je dobijena u trenutku kada je testni slučaj izvršen, tj. predstavlja rezultat izvršavanja testnog slučaja.

Na slici se potom vidi sekcija u kojoj su prikazane ključne riječi koje su pozivane u okviru testnog slučaja (oznake Keyword).

Prva ključna riječ koja je pozvana u testnom slučaju je Start Of Shared Memory Keyword iz fajla memory_space_access.resource. Može se vidjeti da ova ključna riječ ima argument, te da je kao argument proslijeđena varijabla \${configFile}.

Zatim slijedi sekcija Start/End/Elapsed, koja je prethodno opisana.

Unutar ključne riječi Start Of Shared Memory može se primjetiti da je pozvana BuiltIn ključna riječ Evaluate, opisana u poglavlju 5.3.3, a ona ima svoju dokumentaciju koja je ispisana. Rezultat izvršavanja ove ključne riječi je upisan u vrijednost result, što je prikazano u sekciji Info. Ispod informacije o ključnoj riječi Evaluate, nalazi se još jedna Info sekcija, koja predstavlja rezultat izvršavanja ključne riječi Start Of Shared Memory.

Druga ključna riječ pozvana u testnom slučaju Start Of Shared Memory Test je još jedna BuiltIn ključna riječ, Set Test Message, koja kao argument prihvata vrijednost result. Vrijednost result je postavljena u prethodnoj ključnoj riječi. Korištenje ove ključne riječi omogućava kreiranje sekcije Message testnog slučaja na osnovu vrijednosti result. Ključna riječ ima i svoju dokumentaciju, ispisanu ispod njenog imena, te sekcije Start/End/Elapsed i Info, prethodno opisane.

Izveštaji procesa testiranja i detalji implementacije biće priloženi uz rad.

6. ZAKLJUČAK

Softversko testiranje ima ključnu ulogu kada je u pitanju osiguravanje kvaliteta, sigurnosti i pouzdanosti kako softverskih aplikacija, tako i različitih namjenskih uređaja. Kroz njihovo testiranje, testni tim je u prilici da detektuje ali i otkloni postojeće nedostatke ili greške, ali i uoči mogućnost za unapređenje određenih aspekata softverskih, ali i hardverskih proizvoda. Prilikom testiranja primjenjuju se različite testne tehnike i različiti tipovi testiranja, a proizvodi se testiraju na različitim nivoima, kao što je detaljnije objašnjeno u poglavlju broj dva.

Veoma je važno razumjeti da testiranje nije aktivnost koja se jednom izvrši, već proces koji traje i koji je neophodno ponavljati kako se softverski i hardverski proizvodi razvijaju i unapređuju, da bi se osiguralo da proizvodi dostignu svoj maksimalan potencijal.

Zbog toga što je proces testiranja veoma važan i često veoma opširan i detaljan, automatizacija procesa testiranja predstavlja njegov ključni aspekt i upravo zato se pribjegava korištenju različitih komercijalnih ili nekomercijalnih testnih okvira. Međutim, prije odabira određenog testnog okvira koji će se koristiti za automatizaciju procesa testiranja, testni tim se mora upoznati sa svim detaljima i karakteristikama istih, i utvrditi koji radni okvir za testiranje će za određeni proizvod dati najbolje rezultate.

Korištenje Robot framework-a u cilju automatizacije procesa testiranja uređaja TTE-Switch Space 3U cPCI se pokazalo kao odlično rješenje, upravo zbog njegovih karakteristika koje omogućavaju da se testni slučajevi grupišu u logičke cjeline, te testiraju kao kakvi, a omogućava i testiranje samo pojedinačnih testnih slučajeva. Tester ima mogućnost da pokrene veliki broj testnih paketa, čije izvršavanje može trajati i nekoliko sati, te nakon što se proces testiranja završi, može analizirati detaljan i veoma razumljiv izvještaj koji je kreiran u trenutku završetka procesa testiranja.

Robot framework omogućava testnom timu da detaljno dokumentuje svaki testni slučaj pojedinačno, ali i testni paket koji je sastavljen od testnih slučajeva, da bi se kasnije analizirajući izvještaj, moglo utvrditi koja je svrha kreiranja testnog slučaja unutar testnog paketa, tj. razlog zbog kojeg određeni testni slučajevi čine jednu logičku cjelinu, ali i svrhu kreiranja samog testnog paketa, tj. šta tačno logička cjelina treba da testira. Pomenuti izvještaji testnom timu, ili osobama iz tima koje analiziraju izvještaje nude čitljiv prikaz rezultata testiranja, ali i različite mogućnosti za pretraživanje testnih paketa i testnih slučajeva, da bi se evaluiralo njihovo izvršavanje.

Prilikom izrade praktičnog dijela kreiran je radni okvir za testiranje firmware-a uređaja TTE-Switch Space 3U cPCI, prilikom čega je upotrebljen upravo Robot framework, koji je iskorišten za automatizaciju procesa testiranja sljedećeg:

- Provjera da li su prethodno kreirani izvještaji procesa testiranja, tačnije fajlovi log.html i report.html, sačuvani u odgovarajućem folderu, unutar foldera Previous_Report_Data.
- Provjera vremena ICMP odgovora (eng. ICMP response), nakon što je pomenutom switch-u upućen određen broj ICMP Ping zahtjeva (eng. ICMP Ping request).
- TFTP download User i Default-nih fajlova iz NVM memorije u specifični folder.

- Provjera da li firmware dopušta izvršavanje sigurnosne funkcije TFTP upload samo u intervalu od 10 sekundi, nakon što se izvrši CMD_ARM komanda.
- TFTP upload User fajlova u NVM memoriju i provjera validnosti obe kopije upload-ovanih fajlova.
- Pokušaj TFTP upload-a Default-nih fajlova u NVM memoriju. U ovom slučaju je očekivano da upload svakog Default-nog fajla bude neuspješan.
- Dobijanje verzije firmware-a iz odgovarajućeg OID-a, putem SNMP protokola.
- Postavljanje stanja DSU_EN pina na nulu i provjera novog postavljenog stanja pomoću ft232 interface-a. Zatim provjera stanja DSU_EN pina putem SNMP protokola, čitanjem vrijednosti odgovarajućeg OID-a. Ponavljanje istog postupka, ali stanje DSU_EN pina se postavlja na jedan, te izvršavanje prethodno pomenutih provjera putem ft232 interface-a i SNMP protokola.
- Ispitivanje sadržaja na adresi koja treba da sadrži informaciju o početku dijeljene memorije (eng. Shared Memory) i provjeravanje vrijednosti dijeljene memorije nakon dodavanja određenog pomaka (eng. offset) na početnu adresu dijeljene memorije .

Nakon što se proces testiranja završio, kreirani su izvještaji, koji su detaljno opisani u poglavlju 5.4.10.

Kreiranje ključnih riječi na osnovu funkcija iz modula, te njihovo korištenje u testnim slučajevima, omogućava ponovnu upotrebu koda, kao i njegovu preglednost i pokazalo se kao proces koji nije teško savladati.

Pregledom izvještaja koji su kreirani kao rezultat izvršavanja testnih paketa, u okviru foldera Test_Suites, možemo zaključiti da Robot framework zaista predstavlja radni okvir koji može da zadovolji sve zahtjeve testnog tima koji se bavi testiranjem firmware-a uređaja TTE-Switch Space 3U cPCI. Upravo sam izgled izvještaja i jednostavnost pokretanja procesa testiranja su glavni aspekti ovog radnog okvira.

Literatura

- [1] K. Olsen, T. Parveen, R. Black, D. Friedenberg, J. McKay, M. Posthuma, H. Schaefer, R. Smilgin, M. Smith, S. Toms, S. Ulrich, M. Walsh, E. Zakaria, *Certified Tester Foundation Level Syllabus*, International Software Testing Qualifications Board, 2018.
- [2] D. Graham, E. van Veenendaal, R. Black, *Foundations of Software Testing*, International Software Testing Qualifications Board, 2021.
- [3] M. Savić, *Testiranje i kvalitet softvera*. studija, Elektrotehnički fakultet, Banja Luka, 2021.
- [4] BrowserStack, *What is a Test Suite & Test Case?*, , posjećeno: 21.12.2023. godine
- [5] S.M.K. Quadri, S.U. Farooq, "Software Testing – Goals, Principles, and Limitations", *International Journal of Computer Applications*, br. 9, str. 7-11, Septembar 2010.
- [6] IEEE Std 1012™-2017, IEEE Standard for System, Software and Hardware Verification and Validation
- [7] S. R. Jan, S. T. U. Shah, Z. U. Johar, Y. Shah, F. Khan, "An Innovative Approach to Investigate Various Software Testing Techniques and Strategies", *International Journal of Scientific Research in Science, Engineering and Technology*, br. 2, str. 682-689, mart 2016.
- [8] I. Burnstein, *Practical software testing*, Springer, New York, 2002
- [9] M. E. Khan, "Different Forms of Software Testing Techniques for Finding Errors", *International Journal of Computer Science Issue*, br. 3, str. 11-16, maj 2010.
- [10] M. Kumar, S. K. Singh, Dr. R. K. Dwivedi, "A Comparative Study of Black Box Testing and White Box Testing Techniques", *International Journal of Advance Research in Computer Science and Management Studies*, br. 10, str. 32-44, oktobar 2015.
- [11] Testlio, *Comparing Alpha vs. Beta Testing (and how to use both)*, <https://testlio.com/blog/alpha-vs-beta-testing/>, posjećeno: 27.12.2023. godine
- [12] A. Sethi, "A review paper on levels, types & techniques in software testing", *International Journal of Advanced Research in Computer Science*, br. 7, str 269-271, avgust 2017.
- [13] M. A. Umar. , *Comprehensive Study of Software Testing: Categories, Levels, Techniques, and Types*. studija, Changchun University of Science and Technology, Jilin, Kina, jun 2020.
- [14] E. Borjesson, R. Feldt, *Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry*. studija, Chalmers University, Gothenburg, Sweden, maj 2012.
- [15] Lambdatest, *Visual Testing: Ultimate Guide With Examples*, <https://www.lambdatest.com/learning-hub/visual-testing>, posjećeno: 04.01.2024. godine
- [16] M. E. Khan, F. Khan, "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques", *International Journal of Advanced Computer Science and Applications*, br. 6, str. 12-15, jun 2012.
- [17] International Organization for Standardization, Information technology—Software product quality (ISO Standard No.9126)
- [18] Testingchat, *Regression Testing for Software – Your FREE Complete Guide to becoming a better tester*, <https://www.testingchat.com/fundamentals/types-of-testing/regression-testing/>, posjećeno: 08.01.2024. godine

- [19] Edureka!, *What are the Types of Software Testing Models*, <https://www.edureka.co/blog/software-testing-models/>, posjećeno: 20.12.2023. godine
- [20] A. Alshamrani, A. Bahattab, "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model", *International Journal of Computer Science Issue*, br. 1, str 106-111, januar 2015.
- [21] Builtin, *What Is the V-Model in Software Development?*, <https://builtin.com/software-engineering-perspectives/v-model>, posjećeno: 20.12.2023. godine
- [22] R. Black, M. Walsh, G. Coleman, B. Cornanguer, I. Forgacs, K. Kakkonen, J. Sabak, *Agile Testing Foundations*, BCS. jun 2017.
- [23] L. Crispin, J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*, Addison-Wesley, oktobar 2014.
- [24] Edureka!, *What are the Types of Software Testing Models*, <https://www.edureka.co/blog/software-testing-models/>, posjećeno: 22.12.2023. godine
- [25] [Jorge Sena Figueira, *Test Automation Framework for Embedded Systems*, magistrski rad, Univerzitet za nauku i tehnologiju Nova, Lisabon, 2018.
- [26] BrowserStack, *Best Open Source Test Management Tools*, <https://www.browserstack.com/guide/best-open-source-test-management-tools>, posjećeno: 28.12.2023. godine
- [27] Software Testing Help, *Most Popular Test Automation Frameworks with Pros and Cons*, <https://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/>, posjećeno: 28.12.2023. godine
- [28] BMC, *Test Automation Frameworks: The Ultimate Guide*, <https://www.bmc.com/blogs/test-automation-frameworks/>, posjećeno: 28.12.2023. godine
- [29] docs.robotframework.org, *BDD (Behavior Driven Development)*, https://docs.robotframework.org/docs/testcase_styles/bdd#:~:text=The%20Given-When-Then%20syntax%20is%20a%20commonly%20used%20structure,system%20in%20a%20clear%2C%20concise%2C%20and%20consistent%20manner., posjećeno: 28.12.2023. godine
- [30] Lambdatest, *What is UFT? What are the benefits of UFT?*, <https://www.lambdatest.com/software-testing-questions/what-is-uft>, posjećeno 02.01.2024. godine
- [31] UTF One Automation, *How to Create a UFT One Automation Script*, https://admhelp.microfocus.com/uft/en/all/AutomationObjectModel/Content/AutomationIntro/Output/Create_Automation_Script.htm, posjećeno: 05.01.2024. godine
- [32] SaaSwothy, *Micro Focus UFT One Pricing*, <https://www.saaswothy.com/product/micro-focus-uft-one/pricing>, posjećeno: 05.01.2024. godine
- [33] Vector, *Automating Software Testing with VectorCAST*, <https://www.vector.com/int/en/products/products-a-z/software/vectorcast/>, posjećeno: 03.01.2024. godine
- [34] Gauge, *Overview*, <https://docs.gauge.org/overview?os=windows&language=javascript&ide=vscode>, posjećeno: 03.01.2024. godine
- [35] S. Bisht, *Robot Framework Test Automation*, Packt publishing, Birmingham, 2013.
- [36] Robotframework, *Robot Framework User Guide*, <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html>, posjećeno: 16.01.2024. godine

- [37] docs.robotframework.org, *Standard Library*, https://docs.robotframework.org/docs/different_libraries/standard, posjećeno: 16.01.2024. godine
- [38] TechTarget, *What is firmware?*, <https://www.techtarget.com/whatis/definition/firmware>, posjećeno: 15.02.2024.
- [39] A. Ademaj, *Time-Triggered Ethernet – A Powerful Network Solution for Multiple Purpose*, studija, TTTech Computertechnik AG, Beč, 2015.
- [40] M. Pisecky, *TT-EPOS Switch Firmware 0.14.1*, TTTech Computertechnik AG, Beč, Februar 2024.
- [41] M. Sandić, I. Velikić, A. Bilbija, M. Milošević, *Analiza principa komunikacije u TTEthernet mrežama*, studija, Institut RT-RK, Jun 2017.
- [42] H. Kopetz, A. Ademaj, P. Grillinger, K. Steinhammer, *The Time-Triggered Ethernet (TTE) Design*, studija, Vienna University of Technology, Beč, maj 2005.
- [43] *TTE-Switch Space 3U cPCI User Manual*, TTTech Computertechnik AG, Beč, Okrobar 2023.
- [44] TectTarger, *NOR flash memory*, <https://www.techtarget.com/searchstorage/definition/NOR-flash-memory>, posjećeno: 28.02.2024.
- [45] Interfacebus, *cPCI and PXI card size*, http://www.interfacebus.com/Design_Connector_CPCI.html, posjećeno: 28.02.2024.
- [46] FlightGlobal, *TTTech Aerospace: certifiable solutions for safety-critical aviation systems*, <https://www.flightglobal.com/paid-content/tttech-aerospace-certifiable-solutions-for-safety-critical-aviation-systems/153549.article>, posjećeno: 29.02.2024.