

TEHNIČKO VELEUČILIŠTE U ZAGREBU

POLITEHNIČKI SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ

Specijalizacija informatika

**Testiranje softvera uporabom PyTest i Hypothesis
radnih okvira**

DIPLOMSKI RAD br. 0246068885-30-2022-1

Zagreb, svibanj, 2023.

POLITEHNIČKI SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ

Specijalizacija informatika

**Testiranje softvera uporabom PyTest i Hypothesis
radnih okvira**

DIPLOMSKI RAD br. 0246068885-30-2022-1

Povjerenstvo:

dr. sc. Željko Kovačević, v. pred., predsjednik _____

dr. sc. Aleksandar Stojanović, v. pred. _____

dr. sc. Danko Ivošević, pred., mentor _____

Zagreb, svibanj, 2023.

Zagreb, 22. svibnja 2023.

Grana: **2.09.06 programsko inženjerstvo**

DIPLOMSKI ZADATAK br. 0246068885-30-2022-1

Pristupnik: **Igor Stanković (0246068885)**

Studij: Stručni diplomski studij Informatika, izvanredna izvedba (smjer računarstvo)

Zadatak: **Testiranje softvera uporabom Pytest i Hypothesis radnih okvira**

Opis zadatka:

Dati pregled metodologije testiranja sustava i softvera.

Opisati mogućnosti PyTest i Hypothesis testnih radnih okvira. Usporediti pristupe "testiranja temeljenog na primjerima" i "testiranja temeljenog na svojstvima" s obzirom na uporabu tih testnih radnih okvira.

Opisati okolinu primjene i vlastitu implementaciju testiranja programske aplikacije u specifičnom industrijskom razvojnom okruženju.

Prikazati uspješnost implementacije na nekoliko testnih primjera uz uporabu izabranih radnih okvira. Iznijeti zaključke rada i komentirati mogućnosti primjene ostvarene implementacije.

Zadatak uručen pristupniku: 3. travnja
2023. Rok za predaju rada: 23. svibnja
2023.

Mentor:

Predsjednik povjerenstva za
diplomski ispit:

dr. sc. Danko Ivošević, pred.

Ognjen Mitrović, struč. spec.
ing.techn. inf., pred.

Zahvala

Zahvaljujem se svojim roditeljima na podršci tijekom studiranja, Xylon tvrtki na uloženom vremenu i financijskoj podršci te se zahvaljujem mentoru koji mi je omogućio pisanja rada na ovu temu.

Sažetak

Ovaj rad bavi se metodologijom testiranja programskog koda kao i važnosti kontinuiranog testiranja programskog sustava kao cjeline. Posebno se fokusira na pitanja „testiranja temeljenog na svojstvima“ (engl. *property-based testing*) gdje se prolazi kroz velik broj testnih slučajeva koji se lako mogu propustiti tijekom uobičajenog testiranja temeljenog na pojedinačnim primjerima (engl. *example-based testing*). Navedeno je koji razvojni okviri i programski jezici to omogućavaju te prednost tog pristupa u odnosu na druge. U praktičnom djelu demonstrirana je provedba automatiziranog testiranja unutar Python okruženja nad sustavom kojeg čini logiRECORDER uređaj i *Dashboard* aplikacija kojom se uređaj upravlja. Testiranje je podijeljeno u dva dijela gdje se najprije demonstrira testiranje temeljeno na primjerima korištenjem Pytest razvojnog okvira, a potom testiranje temeljeno na svojstvima uporabom Pytest okvira i *Hypothesis* biblioteke. Cilj rada je dati uvid u korištenje izabраних alata za automatizirano testiranje, prikazati njihovu primjenu u industrijskom okruženju te testirati doprinos testiranja temeljenog na svojstvima unutar takvog okruženja.

Ključne riječi: test, automatizacija, PyTest, *Hypothesis*, parametar, strategija, CAN

Sadržaj

1. UVOD	1
1.1 METODOLOGIJE TESTIRANJA PROGRAMSKE POTPORE	2
1.1.1 Razine testiranja	2
1.1.2 Metode testiranja	3
1.1.3 Vrste testiranja	4
1.2 AUTOMATIZACIJA TESTIRANJA	5
2. PYTEST RAZVOJNI OKVIR	6
2.1 KAKO I ZAŠTO KORISTITI PYTEST	6
2.2 GRUPACIJA TESTOVA	8
2.3 PYTEST FUNKCIJA UČVRŠĆENJA	9
2.4 PARAMETRIZACIJA TESTOVA	10
2.5 PRESKAKANJE TESTOVA	11
3. TESTIRANJE TEMELJENO NA SVOJSTVIMA	12
3.1 RAZVOJ TEHNOLOGIJE	13
3.2 KORIŠTENJE HIPOTEZE	15
3.1.1 Stvaranje vlastitih strategija	18
3.1.2 Pravila kreiranje vlastitih strategija	19
4. OPIS TESTNOG SUSTAVA	20
4.1 RADNO OKRUŽENJE	24
4.2 OPIS TESTNIH PODATAKA	25
4.2.1 Struktura CAN okvira	26
4.2.2 CAN Modbus baza podataka	28
4.3 TEMELJI TESTNOG PROJEKTA	32
5. PROVEDBA TESTIRANJA	37
5.1 TESTIRANJE TEMELJENO NA PRIMJERIMA	38
5.1.1 Struktura testne funkcije	38

5.1.2	<i>Primjer pokretanja testa</i>	40
5.2	TESTIRANJE TEMELJENO NA SVOJSTVIMA.....	42
5.2.1	<i>Struktura testne funkcije</i>	42
5.2.2	<i>Primjeri izvršavanja testa</i>	48
5.3	PRIMJERI PRONAĐENIH GREŠAKA POMOĆU TESTA PARSIRANJA.....	57
6.	ZAKLJUČAK	60
7.	LITERATURA	61

Popis oznaka, kratica

ADAS - Advanced Driver-Assistance System

API - Application Programming Interface

ARXML - AUTOSAR Extensible Markup Language

AUTOSAR - AUTomotive Open System Architecture

CAN - Controller Area Network

CAN FD - Controller Area Network Flexible Data-Rate

CRC - Cyclic Redundancy Check

ECU – Engine Control Unit

HTTP - Hyper Text Transfer Protocol

HIL –Hardware-in-the-loop

IT – Information Technology

LIN - Local Interconnect Network

Mdf4 –Measurement Data Format version 4

PDU –Protocol Data Unit

URL - Uniform Resource Locator

UTP – Unshielded-Twisted-Pair

XML – Extensible Markup Language

Popis slika

SLIKA 1. LOGIRECORDER 3.2 UREĐAJ (IZVOR: [17])	21
SLIKA 2. DASHBOARD APLIKACIJA - PROZOR ZA SNIMANJE (IZVOR: [20])	22
SLIKA 3. DASHBOARD APLIKACIJA-PROZOR ZA REPRODUKCIJU (IZVOR: [20])	22
SLIKA 4. DIJAGRAM RAZMJEŠTAJA TESTNOG SUSTAVA.....	23
SLIKA 5. CAN OKVIR (IZVOR [22])	26
SLIKA 6. CAN FD OKVIR (IZVOR: [22])	28
SLIKA 7. PDU KONTEJNER (IZVOR: [26])	29
SLIKA 8. CAN OKVIRI MOTBUS BAZE PODATAKA	29
SLIKA 9. SIGNALI ABSDATA OKVIRA	30
SLIKA 10. PARSIRANI OKVIRI UNUTAR DASHBOARD PROZORA	31
SLIKA 11. DIJAGRAM KLASA TESTNIH METODA PROJEKTA.....	34
SLIKA 12. DIJAGRAM KLASA TESTIRANIH SUČELJA	35
SLIKA 13. DIJAGRAM KLASA TESTA	36
SLIKA 14. PRIKAZ INVOKACIJE 32 UNUTAR TABLICE INVOCATIONS.....	41
SLIKA 15. PRIKAZ INVOKACIJE 32 UNUTAR TABLICE INVOCATIONS.....	41
SLIKA 16. ISHOD PRVE INVOKACIJE	48
SLIKA 17. ISHOD PRVE INVOKACIJE	49
SLIKA 18. ISHOD DRUGE I TREĆE INVOKACIJE	49
SLIKA 19. USPOREDBA VRIJEDNOSTI WHEELSPEEDFL SIGNALA	50
SLIKA 20. TABLICA MESSAGE	53
SLIKA 21. ISHOD INVOKACIJE 12	54
SLIKA 22. ISHOD INVOKCIJA	54
SLIKA 23. PRIKAZ TOČNE VRIJEDNOSTI POČETNOG BITA.....	56
SLIKA 24. USPOREDBA VRIJEDNOSTI CAMHEADINGANGLEOBJECT00_ITS SIGNALA	58
SLIKA 25. PRIKAZ GREŠKE UNUTAR DASHBOARD APLIKACIJE.....	59

Popis tablica

TABLICA 1. USPOREDBA DVA NAČINA TESTIRANJA	14
--	----

1. Uvod

Danas je u svijetu sve više IT tvrtki koje razvijaju sve zahtjevnije aplikacije koje nude niz značajki te je njihova struktura vrlo kompleksna. Takve aplikacije potrebno je detaljno testirati kako bi se uklonio što veći broj grešaka u kodu. Testiranje je moguće provesti ručno gdje tester simulira korisnika aplikacije te pokušava proći kroz sve mogućnosti koje nudi neka aplikacija. Svaki korisnik ima neku svoju viziju kako koristiti pojedinu aplikaciju te je gotovo nemoguće da će jedan tester pa čak i desetak njih proći kroz sve mogućnosti koje nudi neka aplikacija. Za takvo nešto potrošilo bi se previše vremena te se zato preporučuje pisanje automatiziranih testova koji puno brže prolaze kroz puno veći broj testnih slučajeva. Nakon isporuke neke aplikacije potrebno je njezino održavanje i nadogradnja novim značajkama po potrebama korisnika. Često kod kreiranja novih značajki nastane greška koja prouzroči nepravilan rad prethodno kreiranih komponenti. Kako bi se to izbjeglo potrebno je stalno testirati sve komponente aplikacije, a ne samo one koje su u procesu nadogradnje. To je vrlo lako postići regresijskim testiranjem odnosno automatizacijom testa koji se po potrebi mogu izvršavati svakoga dana kako bi se odmah uočila potencijalna opasnost u izmjenjivanju postojećeg. Kako bi se testirale sve komponente i njihova međusobna komunikacija predlaže se integracijski način testiranja, a da bi se testiranje izvršilo sa što raznovrsnijim testnim slučajevima preporučuje se koristiti tzv. „testiranje temeljeno na svojstvima“.

U uvodnom poglavlju opisana je važnost testiranja, moguće metode testiranja i koje su prednost koje nudi automatizirano testiranje. U drugom i trećem poglavlju opisani su razvojni okviri i njihove značajke koje omogućuju automatizirano testiranje temeljeno na svojstvima. Takva vrsta testiranja provedena je unutar petog poglavlja nad sustavom kojeg čine logiRECORDER uređaj (sustav za snimanje i reproduciranje podataka video kamera i sabirnica unutar vozila) i *Dashboard* aplikacija koji su prethodno opisani u četvrtom poglavlju.

1.1 Metodologije testiranja softvera

Svaki gotov sustav trebao bi raditi bez grešaka i onako kako je zamišljeno, no često se kod oblikovanja, implementacije i nadograđivanja sustava ne uviđaju moguće pogreške ili nestabilnosti u radu. Testni sustav *Dashboard* aplikacije pruža programsku potporu kontrole i vizualizacije značajki koje pruža logiRECORDER uređaj o kojem se detaljnije govori u nastavku rada. Svaki softver nužno je testirati budući da programi sami za sebe ne mogu pružati jamstvo pravilnog rada pod svim mogućim okolnostima. Testiranje softvera je složen i dugotrajan proces koji započinje testiranjem manjih komponenti u ranim fazama razvoja, a završava testiranjem potpuno izgrađenog sustava prije puštanja u produkciju korisnicima koji odlučuju o konačnoj kvaliteti proizvoda. Kako bi proces testiranja pružao najbolji učinak ka dovođenju softvera do ispunjenja svojih ciljeva potrebno je odrediti najbolje metode i pristupe testiranja. Metodologije testiranja mogu se smatrati skupom tehnika za testiranje koje se provode u životnom ciklusu razvoja softvera s ciljem ubrzavanja njene isporuke bez narušavanja kvalitete. Izbor odgovarajuće metodologije ključan je dio unutar procesa testiranja koji uključuje odabir odgovarajuće razine testiranja, metode testiranja te vrstu testiranja koji se planira provesti nad softverom. U nastavku poglavlja ukratko se opisuju pojedine razine testiranja i što predstavljaju, moguće metode testiranja te vrste testiranja. [1] [2] [3]

1.1.1 Razine testiranja

Razine testiranja softvera predstavljaju različite faze provođenja testiranja tijekom njezina razvoja. Svaka razina ima svoju svrhu, ciljeve ali i nedostatke. Kod testiranja softvera razine testiranja se mogu podijeliti na:

- Jediničnim testiranjem (engl. *unit testing*) testiraju se pojedinačne jedinice odnosno komponente sustava kako bi se ustanovilo da svaka jedinica radi onako kako je zamišljeno.
- Integracijsko testiranje (engl. *integration testing*) je vrsta testiranja koja provjerava ispravan način rada pojedinih jedinica softvera kada se one povežu u jednu cjelinu za razliku od testiranja jedinica kod kojeg se pojedine

jedinice testiraju zasebno neovisno jedna o drugoj. Kako bi se pravilno testirao cijeli programski kôd preporučuje se provesti obje vrste testiranja gdje se prvo provodi testiranje jedinica, a potom integracijsko testiranje.

- Sistemsko testiranje se temelji na testiranju sustava kao cjeline kako bi se provjerilo radi li sustav u skladu s definiranim zahtjevima. Najčešće se kod sistemskog testiranja pristupa metodom crne kutije kod koje tester ne poznaje unutarnju strukturu sustava.
- Testiranje prihvatljivosti je vrsta testiranja koju provodi krajnji korisnik kako bi se utvrdilo radi li softver prema očekivanjima. [4]

1.1.2 Metode testiranja

Postoji mnogo različitih metoda testiranja softvera, no u osnovi ne postoji metoda koja bi se mogla izdvojiti kao najbolja. U praksi se najčešće kombiniraju više različitih metoda ovisno o potrebama testiranja.

Ovisno o stanju sustav koji se testira metode se mogu podijeliti na:

- Statičko testiranje je metoda kod koje se testiranje provodi bez pokretanja koda, bilo ručno ili pomoću alata. Ovakva vrsta testiranja se provodi u ranim fazama razvoje prije provođenja dinamičkog testiranja.
- Dinamičko testiranje se provodi tako da se softver testira u procesu izvođenja kako bi se ispitalo njeno ponašanje unutar dinamičkog okruženja. Takvim testiranjem se nastoje pronaći slabe točke softvera gdje se nakon unošenja ulaznih podataka rezultati uspoređuju s očekivanim rezultatima.

Ovisno o poznavanju ili nepoznavanju unutarnje strukture sustava koji se testira može se napraviti sljedeća podjela:

- Testiranje metodom crne kutije (engl. *black-box testing*) kao što je rečeno testira se softver čija unutarnja struktura nije poznata. Glavna prednost ove

metode je što tester ne mora poznavati strukturu i sadržaj koda niti posjedovati znanje o određenim programskim jezicima.

- Testiranje metodom bijele kutije (engl. *white-box testing*) usredotočuje se na to kako softver radi te se podrazumijeva da tester poznaje unutarnju strukturu koda. Greške koje se ovom metodom mogu pronaći su loša struktura koda, nepredviđeni ulazi, logičke pogreške u kodu i propusti u sigurnosti sustava.
- Testiranje metodom sive kutije (engl. *gray-box testing*) je kombinacija metoda crne i bijele kutije kod koje je struktura koda djelomično poznata. Glavni cilj ove metode je pronaći greške koje nastaju zbog nepravilne strukture koda ili nepravilne uporabe aplikacije. [3] [5]

Isto tako, ovisno o načinu pripreme testnih scenarija razlikuju se:

- Ručno testiranje kao metoda kod koje tester samostalno priprema testne scenarije koje potom provodi kako bi pronašao greške softvera.
- Automatizirano testiranje za razliku od ručnog testiranja predstavlja metodu testiranja softvera kod koje se testiranje automatizira pomoću posebnih alata.

1.1.3 Vrste testiranja

Općenito, testiranje softvera može sadržavati jednu ili više vrsta testiranja ovisno o njezinoj složenosti i proračunu tvrtke koja ulaže u testiranje. Od ostalih vrsta testiranja razlikuju se još:

- Funkcionalno testiranje temelji se na zahtjevima i projektnoj specifikaciji te provjerava ponaša li se softver prema očekivanom.
- Nefunkcionalno testiranje temelji se na ispunjavanju nefunkcionalnih zahtjeva kao što su performanse sustava, pouzdanost, sigurnosti, usklađenost.
- Strukturno testiranje smatra se više tehničkim testiranjem nego funkcionalnim testiranjem kod kojeg se testni slučajevi pokušavaju dizajnirati iz izvornog koda, a ne iz specifikacija.

- Testiranje ovisno o promjenama odnosno regresijsko testiranje osigurava da napravljene promjene unutar softvera ne narušavaju funkcionalnost komponentata koje se ne bi trebale mijenjati. Što znači da se regresijskim testiranjem ponavljaju testovi koji se prethodno izvršeni kako bi se osigurao pravilan rad postojećih komponenta. [4] [6] [7]

1.2 Automatizacija testiranja

Automatizirano testiranje kao što je rečeno predstavlja metodu testiranja kod koje se testiranje provodi izvršavanjem automatiziranih skripti za testiranje softvera. Testne skripte mogu se napraviti u jednom od programskih alata kao što su Python, JavaScript, C#, C++, a implementacija skripti može se uvelike olakšati korištenjem razvojnih okvira za testiranje softvera. Neki od najpoznatijih razvojnih okvira za testiranje su PyTest i Robot za programski jezik Python, Selenium i JUnit za programski jezik Java te MSTest i xUnit.Net za programski jezik C#. Kako bi se automatizirano testiranje izvršavalo kontinuirano testirajući postojeće i nove funkcionalnosti potrebno je koristiti neki od alata kao što su Jenkins, TeamCity, GitLab, Buddy koji omogućuju samostalno pokretanje skripti. Provođenjem kontinuiranih automatiziranih testa dobivaju se različite prednosti kao što su:

- ušteda vremena jer se izvode samostalno bez potrebnog nadzora,
- u dugoročnoj uporabi su isplativiji od ručnog testiranja,
- zbog brzine izvođenja osigurana je veća pokrivenost,
- ako se pravilno sastave manje su sklone greškama,
- isključuju ljudski faktor umora kod ponavljanja testnih slučajeva. [6]

Pristup koji se primjenjuje u ovom radu temelji se na dinamičkom testiranju aplikacije kao crne kutije te se primjenjuje kontinuirano regresijsko testiranje sustava kao cjeline gdje je integrirana softverska aplikacija Dashboard s uređajem logiRECORDER. Uporabom softverskih alata postignuta je automatizacija testiranja.

2. PyTest radni okvir

Jedan od razvojnih okvira za testiranje aplikacija je PyTest [4] koji omogućuje testiranje programskog koda odnosno pisanje jednostavnih i skalabilnih testnih slučajeva za testiranje baza podataka i API korisničkih sučelja. Razvojni okvir PyTest ponajviše se koristi za testiranje programskog sučelja aplikacije u kojem se nudi široki raspon složenosti testa od jednostavnijih testa jedinica do složenijih funkcionalnih testa unutar programskog jezika Python. Python je jedan od najpopularnijih programskih jezika u svijetu te pokazao veliku primjenu u razvoju mrežnih servisa, automatizaciji testiranja kao i u analizi podataka i strojnom učenju. Dizajniran tako da bude lako čitljiv i vrlo proširiv s velikim brojem specijaliziranih biblioteka koje je moguće uključiti unutar vlastitog programskog okruženja. Neke od biblioteka koje se koriste za izradu rada bit će detaljnije opisane u nastavku. [8] [9]

2.1 Kako i zašto koristiti PyTest

PyTest je besplatan razvojni okvir koji nudi varijaciju mogućnosti kod pisanja vlastitih testnih slučajeva. Neke od brojnih mogućnosti uključuju pokretanje specifičnog testa ili skupine testova na temelju zadanog uvjeta, automatsko prepoznavanje testa, preskakanje pojedinih testova, izvršavanje više testnih slučajeva istovremeno čime se smanjuje trajanje izvršenja. Glavni smisao testnih slučajeva su testiranje Python tvrdnji (eng. *assertions*) koje se provjeravaju uporabom naredbe *assert* te vraćaju status istinitosti ili laži. Ako tvrdnja u testnoj metodi pokaže kao lažna tada se izvršenje na tom mjestu zaustavlja, a preostali dio koda u testnoj metodi se ne izvršava. PyTest će tada nastaviti sa sljedećom testnom metodom ako nije drugačije specificirano. Primjer dviju testnih metoda prikazan je programskim kodom 1. [8] [9]


```
import pytest

def test_file1_method1():
    x=5
    y=6
    assert x+1 == y, "test failed"
    assert x == y, "test failed"

def test_file1_method2():
    x=5
    y=6
    assert x+1 == y, "test failed"
```

Programski kôd 1. Primjeri testnih metode

Pokretanjem testnih metoda prikazanih kodom 1 kao rezultat na konzoli dobit će se sljedeći ispis:

```
test_sample1.py F.
===== FAILURES =====
_____ test_sample1 _____
      def test_file1_method1():
          x=5
          y=6
          assert x+1 == y, "test failed"
>       assert x == y, "test failed"
E       AssertionError: test failed
E       assert 5 == 6
test_sample1.py:6: AssertionError
===== 1 failed, 1 passed in 0.04 seconds
```

Na terminalu je vidljivo da su provedena dva testa od kojih je jedan uspješno prošao, a jedan neuspješno. Testna metoda *test_file1_method2()* je zadovoljila sve tvrdnje i ona se smatra uspješnom, dok metoda *test_file1_method1()* nije zadovoljila tvrdnje i ona se smatra neuspješnom. Pretpostavka da je varijabla *x* jednaka varijabli *y* odnosno da je broj 5 jednak broju 6 jest lažna, što uzrokuje izlaz iz testne metode i vraćanjem pogrešne tvrdnje uz dodatnu poruku s opisom pogreške. Vrlo bitan dio kod postavljanja tvrdnji unutar testnih metoda je poruka koja će se ispisati uz lažnu tvrdnju kako bi korisnik testa imao što više informacija o tome zašto neki dio koda nije zadovoljio tvrdnju. [9] [10]

2.2 Grupacija testova

Prema već zadanim postavkama PyTest identificira nazive datoteka koji počinju ili završavaju na ključnom riječi *test* kao testne datoteke. Isto tako, PyTest zahtijeva da nazivi testnih metoda počinju s ključnom riječi *test* jer svi nazivi koji ne počinju tako bit će zanemareni čak i ako je izričito zatraženo pokretanje tih metoda.

Naredbom *pytest* pokrenuti će se svi nazivi datoteka koji počinju ili završavaju ključnom riječi *test* u dotičnoj mapi i podmapama. PyTest nudi opciju grupiranja prema podnizu podudaranja unutar naziva testnih metoda i grupiranje po markerima. Kako bi se pokrenuli testovi prema podnizu podudaranja potrebno je prilikom zadavanja naredbe navesti opciju *-k* pomoću kojeg se predaje željeni niz. Za grupaciju testa po markerima prvo je potrebno postaviti PyTest dekorator iznad testa koji se želi pokrenuti kao što je prikazano programskim kodom 2.

```
import pytest

@pytest.mark.set1
def test_file1_method1():
    x=5
    y=6
    assert x+1 == y, "test failed"
assert x == y, "test failed because x=" + str(x) \
    + " y=" + str(y)

@pytest.mark.set2
def test_file1_method2():
    x=5
    y=6
    assert x+1 == y, "test failed"
```

Programski kôd 2. PyTest marker dekorator

Naredbom *pytest -m set1* pokrenuti će se test *test_file1_method* koji ima dekorator *pytest.mark.set1* dok će test *test_file1_method2* pri testiranju biti zanemaren. [9] [10]

2.3 PyTest funkcija učvršćenja

Unutar PyTest okvira postoje posebne funkcije koje se nazivaju funkcije učvršćenja (eng. *fixture functions*). Funkcije učvršćenja se pokreću prije testnih funkcija na koje se primjenjuju.

One služe za unos podataka unutar testa kao što su URL adrese nad kojima se obavlja testiranje, podaci o povezivanju i opis baze podataka koja je dio aplikacije koja se testira i druge vrste ulaznih podataka.

Tako sprječavamo pokretanje suvišnog koda prilikom ponovljenih testiranja jer jednom kada se funkcija učvršćenja izvrši njezini izlazni podaci dostupni su svim testovima u istom modulu. Primjer funkcije učvršćenja prikazano je kodom 3.

```
import pytest

@pytest.fixture
def input_value():
    input = 39
    return input

def test_divisible_by_3(input_value):
    assert input_value % 3 == 0

def test_divisible_by_6(input_value):
    assert input_value % 6 == 0
```

Programski kôd 3. Pytest fixture dekorator

Funkcija *input_value()* je funkcija učvršćenja koja vraća tip podataka koji je cijeli broj. Testne funkcije *test_divisible_by_3()* i *test_divisible_by_6()* kao ulazni parametar definiraju ime varijable koja odgovara imenu funkcije učvršćenja te tako PyTest prepoznaje da se unutar testne funkcije koristi izlazna vrijednost funkcije učvršćenja. Kako je ranije spomenuto funkcija učvršćenja koja je definirana u određenom modulu ima opseg vidljivosti samo u tom modulu. Kako bi bila vidljiva i u drugim modulima potrebno ju je definirati u posebnom modulu pod nazivom *conftest.py* prema kojoj će pokrenuti testovi tražiti funkcije učvršćenja. [9] [10]

2.4 Parametrizacija testova

Parametrizacija testa je dodatak unutar PyTest okvira koji omogućuje izvođenje pojedinog testa za više skupova ulaznih podataka. Kako bi se testu definirali ulazni parametri potrebno je koristiti dekorator `@pytest.mark.parametrize`. Primjer korištenja parametara prikazano programskim kodom 4.

```
import pytest

@pytest.mark.parametrize("num, output", [(1, 11), (2, 22), (3, 35), (4, 44)])
def test_multiplication_11(num, output):
    assert 11*num == output
```

Programski kôd 4. Parametri testne metode

Programskim kodom je prikazan test koji kao ulazne parametre prima dva broja i provjerava jednakost prvog od tih brojeva pomnoženog s brojem 11 s drugim primljenim brojem. Nakon pokretanja gore prikazanog koda unutar konzole test će generirati sljedeće:

```
test_multiplication.py::test_multiplication_11[1-11] PASSED
test_multiplication.py::test_multiplication_11[2-22] PASSED
test_multiplication.py::test_multiplication_11[3-35] FAILED
test_multiplication.py::test_multiplication_11[4-44] PASSED
===== FAILURES =====
_____ test_multiplication_11[3-35] _____
num = 3, output = 35
    @pytest.mark.parametrize("num, output", [(1, 11), (2, 22), (3, 35), (4, 44)])
    def test_multiplication_11(num, output):
> assert 11*num == output
E assert (11 * 3) == 35
test_multiplication.py:5: AssertionError
===== 1 failed, 3 passed, 8 deselected in 0.08
seconds
```

Unutar konzole vidljivo je kako se isti test definiran testnom funkcijom `test_multiplication_11()` prema očekivanju izvršio četiri puta.

Test je pokrenut za svaki par (engl. *tuple*) u listi parametara od kojih je treći po redu prilikom izvršavanja uzrokovao grešku jer ne ispunjava pretpostavku dane jednakosti.

Također PyTest nudi nizanje više parametara za isti test. Tada će se test izvršiti sve moguće kombinacije predanih parametara. Da bi se dobile sve kombinacije više argumenata, moguće je složiti dekoratore za predaju parametara kako je prikazano programskim kodom 5 .

```
@pytest.mark.parametrize("x", [0, 1])
@pytest.mark.parametrize("y", [2, 3])
def test_foo(x, y):
    pass
```

Programski kôd 4. Kombinacije parametara

U slučaju postavljanja parametara prikazanim kodom 5 test će se izvršiti na sljedećim kombinacijama ulaznih parametara: x=0/y=2, x=1/y=2, x=0/y=3 i x=1/y=3. [9] [10]

2.5 Preskakanje testova

Ponekad pojedine testove koji su napisani potrebno je preskočiti zbog toga što nisu relevantni u nekom određenom razdoblju ili je riječ o značajki u implementaciji, za čije testiranje je napravljen poseban test. U navedenim situacijama postoji opcija dekoratora `@pytest.mark.xfail` koji će omogućiti izvršenje testa, ali ga zanemariti u slučaju neuspjeha te neće ispisivati detalje vezane uz tu neuspjelu pretpostavku. Druga opcija je dekorator `@pytest.mark.skip` koji će u potpunosti preskočiti test koji ima definiran takav dekorator. Primjer korištenja navedenih markera prikazan je programskim kodom 6. [9] [10]

```

import pytest

@pytest.mark.xfail
def test_greater():
    num = 100
    assert num > 100

@pytest.mark.xfail
def test_greater_equal():
    num = 100
    assert num >= 100

@pytest.mark.skip
def test_less():
    num = 100
    assert num < 200

```

Programski kôd 5. Markeri xfail i skip

Nakon pokretanje navedenog koda dobije se rezultat prikazan na konzoli:

```

test_compare.py::test_greater xfail

test_compare.py::test_greater_equal XPASS

test_compare.py::test_less SKIPPED

===== 1 skipped, 1 xfailed, 1 xpassed in 0.06
seconds

```

Za testove *test_greater()* i *test_greater_equal()* prikazan je ishod izvršenja s naznakom da se radi o testu s dekoratorom s *@pytest.mark.xfail* dok je test *test_less()* s dekoratorom *@pytest.mark.skip* preskočen. [9] [10]

3. Testiranje temeljeno na svojstvima

Testiranje temeljeno na svojstvima je naziv za tehniku testiranja kod koje se generira velik broj nasumično generiranih ulaza uporabom specijaliziranih alata koji su namijenjeni da olakšaju provođenje takvog pristupa. U tom smislu upotrebljava se pojam zadovoljenja određenog „svojstva“ umjesto pojedinačnog „testnog slučaja“ s pretpostavkom pokrivenosti testa s nebrojenim brojem ulaza. Alati koji se koriste u tu

svrhu nude i dodatne funkcionalnosti dijagnosticiranja specifičnih testnih slučajeva i naravi greške.

3.1 Razvoj tehnologije

Tvrtka Quviq AB osnovana 2006. godine, jedna je od tvrtki koja primjenjuje testiranjem koda s testovima temeljenim na svojstvima. Njihovi projekti su usmjereni na povećanje učinkovitosti i djelotvornosti u testiranju programskog koda korištenjem automatski generiranih testnih slučajeva. Testiranje temeljeno na svojstvima primijenili su za 5G radio bazne stanice i AUTOSAR programe te pokazali deset puta veću učinkovitost testiranja i za sustave s više od milijun linija koda. Rezultati su objavljeni u znanstvenim časopisima i na konferencijama, kao i predstavljeni na mnogim razvojnim konferencijama i događajima diljem svijeta. [11]

Izvorna biblioteka za testiranje temeljeno na svojstvima je *QuickCheck* napisana za programski jezik Haskell, ali postoje i drugi alati za druge programske jezike kao što su *FsCheck* za .NET, *Jqwik* za programski jezik Java, *QuickCheck* za JavaScript. Unutar programskog jezika Python postoje dvije biblioteke *pytest-quickcheck* i *Hypothesis*. Za izradu testa temeljenih na svojstvima unutar ovog rada koristi se biblioteka *Hypothesis*. Iako se *pytest-quickcheck* isto kao i *Hypothesis* lako može ukomponirati s PyTest razvojnim okvirom *Hypothesis* biblioteka je naprednija biblioteka koja omogućuje lakšu proširivost po vlastitim potrebama testiranja. *Hypothesis* biblioteka omogućuje izradu testa temeljenih na svojstvima koji su po strukturi jednostavniji, ali u mnogim slučajevima detaljniji kod izvođenja. Takav pristup olakšava otkrivanje i tvrdokornijih grešaka pomoću pametnih algoritama i generiranjem nasumičnih parametara te radi tako da testira rubne slučajeve i sve one slučajeve kojeg se autor testa ne bi mogao sjetiti prilikom izrade. Razlika u načlima provođenja standardnih testova i testova temeljnih na svojstvima u usporedbi alata PyTest i Hypothesis prikazan je u tablici 1. [11] [12] [13].

Tablica 1. Usporedba dva načina testiranja

Test temeljen na primjerima (PyTest)	Test temeljen na svojstvima (<i>Hypothesis</i>)
Postavljanje ulaznih parametara ručno proizvoljno	Automatizirano generiranje nasumičnih ulaznih parametra koji odgovaraju nekoj strategiji
Izvršavanje testnih metoda nad parametrima	Izvršavanje testnih metoda nad parametrima uz sužavanje rubnih slučajeva
Provjera rezultata	Provjera rezultata, i jednostavna reprodukcija neuspjelih testa

Testiranje temeljeno na svojstvima alatom Hypothesis radi tako da generira proizvoljne podatke koji odgovaraju postavljenim specifikacijama i provjerava vrijede li pretpostavke postavljene unutar testa za predane podatke. Ako pronađe primjer za koji pretpostavke ne vrijede, uzima ga i pojednostavljuje ga dok ne pronađe najmanji primjer koji i dalje uzrokuje problem. Zatim sprema taj primjer za kasnije, tako da jednom kada pronađe problem u testnom kodu koristi ga kod ponovnog testiranja. [14]

Pisanje testa ovog oblika obično se sastoji od testiranja pretpostavljenih tvrdnji nekog svojstva koje bi uvijek trebale biti istinite za bilo koji ulazni parametar.

Primjeri takvih tvrdnji mogu biti:

- Programski kôd ne bi trebao izbaciti iznimku ili bi trebao izbaciti samo određenu vrstu iznimke (ovo funkcionira posebno dobro ako postoje mnogo internih tvrdnji).
- Ako se izbriše objekt, on više nije vidljiv.
- Ako se serijalizira, a zatim deserijalizira vrijednost, tada će se dobiti ista vrijednost natrag.

Neke od značajki koje obilježavaju testove temeljene na svojstvima su:

- Iako se koriste nasumično generirani podaci u ponovljenim testovima se koriste prethodno generirani podaci koji su prouzročili grešku.
- Dobiva se detaljan odgovor kako i zašto je test pao te nudi opciju reproduciranja ulaznih parametara koji su uzrokovali pad testa.
- Nudi se opciju proširivanja strategija koje se koriste za generiranje ulaznih parametara u obliku kompozitnih funkcija o kojima se detaljnije govori u nastavku. [11] [14]

3.2 Korištenje hipoteze

Hypothesis razvojni okvir pruža korištenje strategija za veliki broj ugrađenih tipova podataka s argumentima za ograničavanje ili prilagođavanje izlaza, kao i složenije strategije koje se mogu sastaviti za generiranje složenijih tipova podataka nad kojima će se provoditi testiranje. Jedan od najkorištenijih tipova u programiranju je niz znakova. Programskim kodom 7 prikazana je testna metoda `test_write_read_csv()`. [14] [15]

```
def test_write_read_csv():
    #Test function
    fields = ["Hello", "World"]
    formatted_row = naive_write_csv_row(fields)
    parsed_row = naive_read_csv_row(formatted_row)
    assert fields == parsed_row
```

Programski kôd 6. Testna metoda

Test kao ulazni parametar prima listu nizova znakova i testira dvije funkcije od kojih jedna kao ulazni parametar prima predanu listu nizova i vraća jedan tekstualni zapis sastavljen od svih elemenata liste odvojenih zarezom. Druga funkcija radi suprotno te vraća listu nizova na temelju predanog tekstualnog zapisa, Programski kod 8.

```
def naive_write_csv_row(fields: List[str]) -> str:
    # The function returns text containing comma-separated strings
    return ",".join(f'"{field}"' for field in fields)

def naive_read_csv_row(row: str) -> List[str]:
    # The function separates the text by a comma in the form of a list
    return [field[1:-1] for field in row.split(",")]
```

Programski kôd 7. Funkcije unutar testne metode

Prema tome, pretpostavka da ulazna lista mora biti jednaka listi koju vraća funkcija *naive_read_csv_row()* bi trebala vrijediti za listu koja je inicijalizirana u primjeru. *Hypothesis* nudi strategiju za kreiranje liste nasumične duljine koja se sastoji od nizova. Ako se test prikazan programskim kodom 7 napiše kao test temeljen na svojstvima onda bi testna funkcija trebala primiti parametre kako je prikazano programskim kodom 9. [14] [15]

```
import hypothesis.strategies as st
from hypothesis import given

@given(fields=st.lists(st.text(), min_size=1, max_size=10))
def test_read_write_csv_hypothesis(fields):
    formatted_row = naive_write_csv_row(fields)
    parsed_row = naive_read_csv_row(formatted_row)
    assert fields == parsed_row
```

Programski kôd 8. Test temeljen na svojstvima

Test kao ulazni parametar prima strategiju koja generira listu nizova koja može biti duljine od 1 do 10 elemenata. Parametar generiran strategijom predaje se testnoj funkciji preko dekoratora *@given*. Pokretanjem programskog koda naredbom *pytest* dobiva se ispis na terminalu.

```
$ pytest test.py::test_read_write_csv_hypothesis
E      AssertionError: assert ['',''] == ['', '']
test.py:44: AssertionError
----- Hypothesis -----
Falsifying example: test_read_write_csv_hypothesis(
    fields=['',''],
)
FAILED test.py::test_read_write_csv_hypothesis - AssertionError: assert ['',''] ==
['', '']
```

Kako testna funkcija kao ulazni parametar prima listu nasumično generiranih nizova znakova jedan od generiranih ulaza prouzrokovao je grešku. Generirana ulazna lista (['','']) ne odgovara listi koju je vratila metoda *naive_read_csv_row()* te postavljena tvrdnja unutar testa nije istinita. [11] [12] Ako se ubuduće izričito želi uključiti testni slučaj s listom ['',''] to se može učiniti navodom dekoratora *@example*.

To bi bio jednostavan primjer načina rada i otkrivanja grešaka *Hypothesis* radnog okvira. Otkriveno je da postoji problem s obzirom na to da zarez kao korišteni graničnik pri spajanju nizova znakova ujedno može biti i znak u unutar pojedinih nizova pri čemu kombinacija korištenih funkcija ne daje očekivani rezultat.

3.1.1 Stvaranje vlastitih strategija

Iako biblioteka *Hypothesis* nudi veliki broj gotovih strategija za razne tipove podataka nudi i opciju kreiranje vlastitih strategija. Dekorator `@composite` omogućuje pisanje kompozitnih funkcija, odnosno imperativnog koda kojim se generiraju parametri. Primjer `@composite` dekoratora prikazan programskim kodom 10. [14] [15]

```
from hypothesis.strategies import composite

def generate_westernized_name(min_size=2):
    return (st.text(alphabet=string.ascii_letters,
                    min_size=min_size)
            .map(lambda name: name.capitalize()))

@composite
def generate_full_name(draw):
    first_name = draw(generate_westernized_name())
    last_name = draw(generate_westernized_name())
    return (last_name, first_name)
```

Programski kôd 9. Kompozitna strategija

Funkciji koja ima dekorator `@composite` potrebno je kao ulazni parametar predati `draw()` funkciju pomoću koje se dobivaju generirane vrijednosti strategija. Kompozitna funkcija `generate_full_name()` inicijalizira varijable `first_name` i `last_name` preko poziva funkcije `generate_westernized_name()` koja vraća strategiju za generiranje teksta minimalne veličine dva znaka. Zatim se dvije generirane strategije spremaju u par koji se potom predaju testnoj metodi preko dekoratora `@given`, Programski kôd 11.

```
@given(first_name=generate_full_name())
def test_create_customers(first_name):
    pass
```

Programski kôd 10. Predaja parametara

Kada se koristi strategija koja ima `@composite` dekorator za predaju parametara nije moguće koristiti `example()` dekorator jer se tada prekida sposobnost pravilnog sintetiziranja testnih slučajeva. Ako strategija generira neki primjer koji ne odgovara potrebama testa moguće je koristiti funkciju `assume()` pomoću koje se definira koji generirani primjeri su validni. Primjer funkcije `assume()` prikazan je programskim kodom 12. [14] [15]

```
from hypothesis import assume

@composite
def generate_full_name(draw):
    first_name = draw(generate_westernized_name())
    last_name = draw(generate_westernized_name())
    assume(first_name != last_name)

    return (last_name, first_name)
```

Programski kôd 11. Funkcija `assume()`

Funkcija `assume()` navodi pretpostavku koju hipoteza mora ispuniti ako se želi generirati primjer iz strategije `generate_full_name()`. [16]

3.1.2 Pravila kreiranje vlastitih strategija

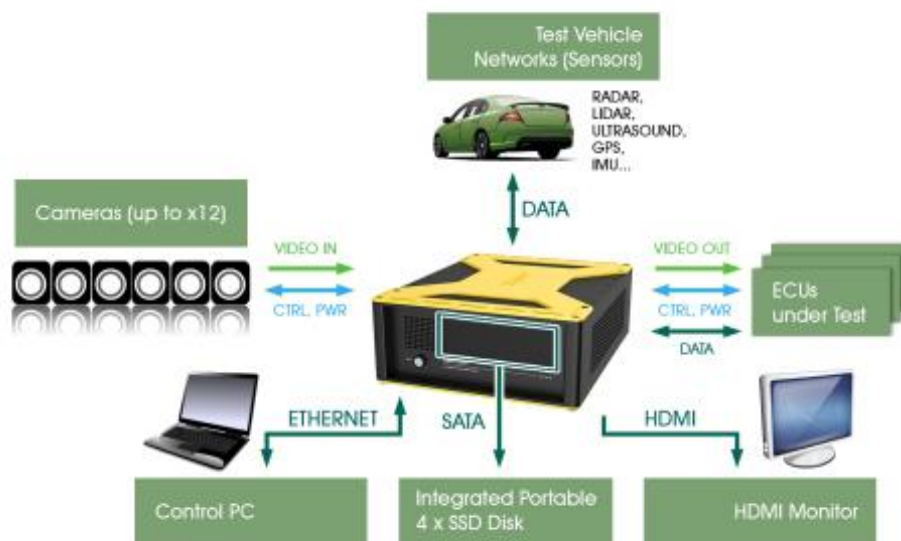
Kod kreiranja vlastitih strategija potrebno se pridržavati određenih pravila kako bi strategije pravilno generirale testne slučajeve. Svi testni slučajevi koji su generirani i predani testu trebali bi proći kroz sve provjere i pretpostavke unutar testa. Zato bi se potrebna filtriranja testnih slučajeva trebala odraditi unutar kompozitnih strategija, a ne unutar testa nakon što su testu već proslijeđeni testni slučajevi.

Kod pisanja složenijih strategija poželjno bi bilo odvojiti funkcionalne strategije koje generiraju tip podatka koji mora zadovoljiti određene kriterije i nefunkcionalne strategije koje generiraju bilo kakav određeni tip podataka. Odvajanje je korisno kada su strategije neuspješne tj. ne generiraju primjere kakvi su očekivani jer se tada strategije mogu postepeno provjeravati te je pronalaženje uzroka greške olakšano.

Važno je napomenuti da se vlastite strategije pišu samo kad je nužno. Imperativnim strategijama koje koristite *assume()* potrebno je više vremena da generiraju željene podatke. [14] [15]

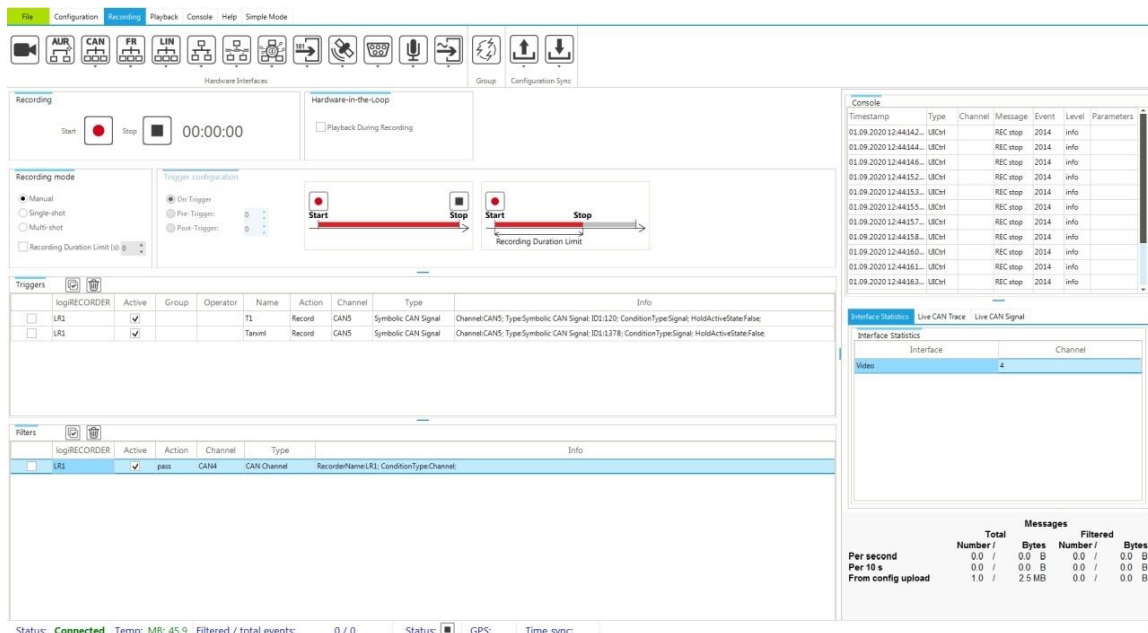
4. Opis testnog sustava

U nastavku rada opisuje se provedba automatiziranog testiranja nad sustavom proizvedenim od strane tvrtke Xylon. Sustav se sastoji od logiRECORDER 3.2 uređaja, pomoćnih komponenti i stolne aplikacije kojom se upravlja uređajem. Razvoj i testiranje vrhunskih sustava autonomne vožnje i ADAS sustava koji se temelje na vizualnom pregledu okoline u rasponu od 360 stupnjeva u blizini vozila (engl. *surround view*) zahtijeva veći broj testiranja različitih situacija vožnje ovisno o uvjetima na cesti. Dugotrajni, skupi i ograničeni testni scenarij prilikom razvoja softvera za stvarni prometni sustav mogu se u velikoj mjeri unaprijediti s logiRECORDER uređajem. LogiRECORDER omogućuje snimanje podataka s niskom latencijom neobrađenih višekanalnih video i mrežnih podataka, analizu podataka i sinkronu reprodukciju zapisanih podataka s pomoću simulacija u stvarnom vremenu (engl. *hardware-in-the-loop*) koje reproduciraju video sustav testnog vozila s više kamera i podataka sučelja u vozilu kako bi se testirali kontroleri unutar automobila u što realističnijim uvjetima. Paralelno je moguće koristiti 20 različitih sučelja ulazno/izlaznih jedinica, uključujući video module (FPD-Link 3, GMSL 2), automobilske mrežne module (CAN, LIN, FlexRay) i Internet module (Smart Ethernet, BroadR-Reach). Snimljene podatke ulazno/izlaznih jedinica moguće je pohraniti na četiri SSD (engl. *solid state drive*) diska koji osiguravaju dovoljno prostora za pohranjivanje video snimaka visoke rezolucije. LogiRECORDER 3.2 uređaj prikazan je na slici 1. [17] [18] [19]

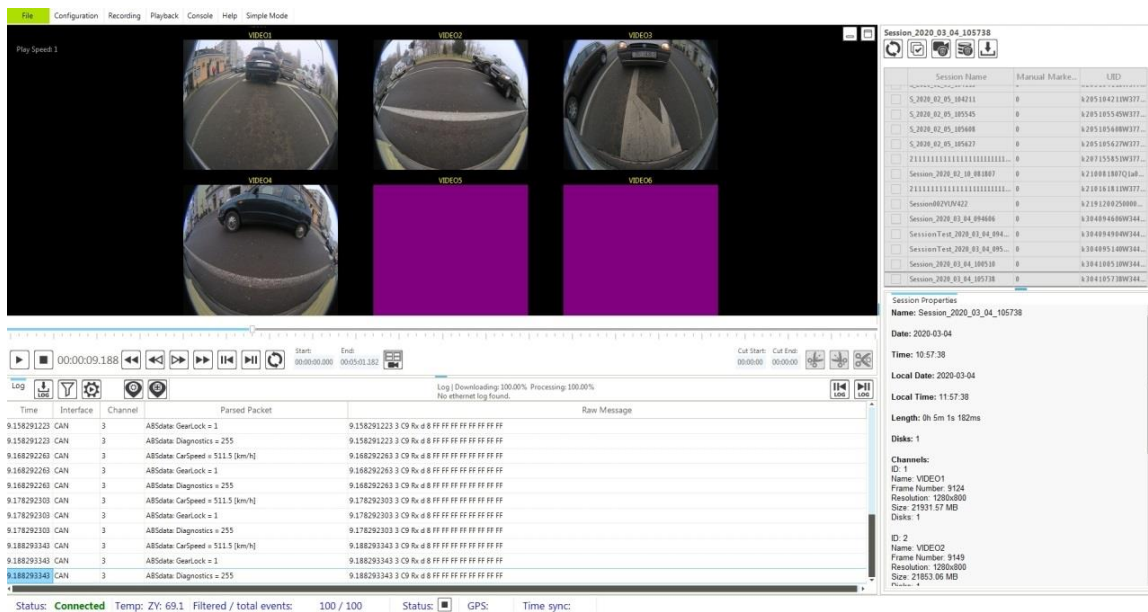


Slika 1. logiRECORDER 3.2 uređaj (izvor: [17])

Kako bi korištenje logiRECORDER uređaja bilo jednostavnije napravljena je *Windows* stolna aplikacija *Dashboard* unutar .NET okruženja kojom je omogućeno upravljanje uređajem kao i vizualizacija cjelokupnog prometa ulazno/izlaznih jedinica. Aplikacija omogućuje potpunu kontrolu nad svim značajkama uređaja. Pruža se mogućnost mrežnog načina rada na osobnom ili tablet računalu povezanom s logiRECORDER uređajem putem kabela ili bežične veze, ili na računalu za izvanmrežnu analizu zapisa testnih podataka bez povezanog logiRECORDER uređaja. Aplikacijski prozor za snimanje prometa sa sabirnica vozila i video kanala prikazan je na slici 2, a prozor za reprodukciju snimaka koje sadržavaju video i CAN promet prikazan je na slici 3. [17] [20]

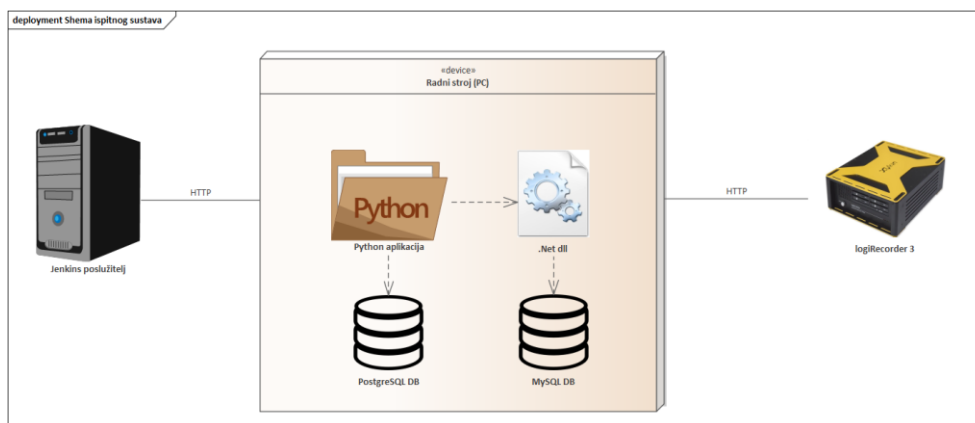


Slika 2. Dashboard aplikacija - prozor za snimanje (izvor: [20])



Slika 3. Dashboard aplikacija-prozor za reprodukciju (izvor: [20])

Svrha projekta je svakodnevno testiranje logiRECORDER sustava i *Dashboard* aplikacije, a to se postiže automatskim pokretanjem testa koji testiraju nove inačice *firmware* osnovice logiRECORDER uređaja i *Dashboard* projekta te njihovu međusobnu komunikaciju. Komunikacija između *Dashboard* aplikacije i logiRECORDER *firmware* osnovice provodi se HTTP protokolom kojim se šalju zahtjevi na logiRECORDER uređaj za izmjenu njegove konfiguracije ili pokretanje pojedinih naredbi kao što su snimanje i reproduciranje prometa ulazno/izlaznih jedinica. Unutar Python aplikacije koja sadrži testove napisane u PyTest okruženju uvode se klase i metode iz prethodno kreiranih .NET dinamičkih biblioteka povezivanja (engl. *DLL* - *dynamic link library*) *Dashboard* aplikacije. Uvođenje dinamičkih biblioteka odvija se pomoću Python.Net biblioteke koja omogućuje interakciju između Python i .NET programskog koda koji sadrži potrebne metode za slanje spomenutih HTTP zahtjeva. Za automatiziranje testa koristi se Jenkins poslužitelj koji pokreće testove u postavljenom vremenu i redoslijedu. Unutar Jenkins poslužitelja postoje izvješća koja nakon završetka zadanih poslova nude pregled ishoda svakog pojedinog testa. Na slici 4. prikazana je shema sustava.



Slika 4. Dijagram razmještaja testnog sustava

Detaljniju analizu izvršenih testa moguće je provesti unutar PostgreSQL baze podataka u kojoj se pohranjuju rezultati testnih koraka kao i druge korisne informacije koja pomažu u otkrivanju grešaka. MySQL baza podataka služi *Dashboard* aplikaciji pisanoj u .NET tehnologiji za pohranu snimljenih logova i njihovih parsiranih vrijednosti.

4.1 Radno okruženje

Kod izrade projekta i prikaza rezultata testiranja potrebna je instalacija određenih alata i paketa koje pružaju podršku kreatoru testa i njihovim korisnicima. Kako bi se osiguralo da se unutar projekta uvezu sve biblioteke određene inačice potrebne za pravilan rad napravljeno je virtualno okruženje kojim se izbjegavaju sukobi s pogrešno instaliranim inačicama biblioteka i nepotrebnim instalacijama na računalu domaćinu te se osigurava da će projekt uvijek imati sve potrebne biblioteke s odgovarajućim inačicama. [21]

Za izradu programskog kôda projekta izabran je programski jezik Python inačice 3.8.10. Python je odabran zbog svoje dinamike i velike količine otvorenih paketa koje olakšavaju izradu vlastitog proizvoda među kojima su i biblioteke *PyTest* i *Hypothesis* koje se koriste za izradu ovoga projekta, već opisane u ranijem poglavlju. *PyTest* alat se smatra čitavim radnim okvirom zbog svoje složenosti i mogućnosti kontrole nad konstrukcijom programa. Taj razvojni okvir podrazumijeva određena pravila i strukturu testnog projekta koji se gradi. *Hypothesis* se također smatra razvojnim okvirom iz istih razloga kao i *PyTest* s kojim ga je moguće ukomponirati. Takva struktura gdje se zajedno koriste *PyTest* i *Hypothesis* radni okvir opisuje se i prikazuje se u ovom radu.

Tu su i ostale važnije biblioteke koje se nalaze unutar virtualnog okruženja namijenjenog za testiranje opisanog testnog sustava:

- ***Python.Net*** paket omogućuje integraciju s .NET radnim okvirom. Pomoću ovog paketa moguće je uvesti već kreirane biblioteke .NET aplikacije ili graditi cijele aplikacije u Python programskom jeziku koristeći .NET komponente napisane u jezicima temeljenim na njezinoj CLR komponenti, kao što su C#, VB.NET, F#, C++. *Dashboard* aplikacija koja se testira u projektu napisana je u C# jeziku korištenjem .NET radnog okvira 4.7.2. Pomoću *Python.Net* paketa uvoze se sve potrebne klase koje je potrebno treba testirati.

- ***Requests*** je standardna Python biblioteka za izradu HTTP zahtjeva. Njome se smanjuje složenost kreiranja zahtjeva korištenjem jednostavnog API sučelja koje osim kreiranja zahtjeva nudi opcije njihove konfiguracije kao i čitanje povratnih informacija. Osim povezanosti preko serijskog porta s logiRECORDER uređajem se povezuje i Ethernet kabelom pa je moguće slati HTTP zahtjeve. Pomoću *Requests* biblioteke automatizira se slanje zahtjeva na logiRECORDER uređaj kada se želi poslati neka određena naredba ili ako se želi provjeriti neko određeno stanje samog uređaja.
- ***PyFunctional*** biblioteka olakšava manipulaciju nad kolekcijama pomoću ulančanih funkcijskih operatora. Kako testovi koriste mnogo podataka koji su pohranjeni jedan unutar drugoga testne funkcije koje se koriste imaju kvadratnu složenost što značajno usporava vrijeme izvođenja. Korištenjem *PyFunctional* biblioteke ubrzava se obrada takvih podataka, a posebice se koristi kod filtriranja prikupljenih podataka po specifičnom uvjetu i mapiranja jedne vrste objekata u druge.

4.2 Opis testnih podataka

S obzirom da je jedna od glavnih funkcija logiRECORDER uređaja snimanje prometa sabirnica vozila i video kamera te njihova reprodukcija, vrlo važna komponenta *Dashboard* aplikacije je vizualizacija podataka. Za vizualan prikaz podataka potrebno je sirove podatke sabirnica parsirati u podatke čitljive ljudskom oku. Za demonstraciju i testiranje komponente parsiranja odabrana je sabirnica CAN ili skraćeno od Controller Area Network za koju postoji velika količina baza podataka u kojima se nalaze potrebne informacije za parsiranje podataka. CAN predstavlja sustav serijske sabirnice visokog integriteta koji za umrežavanje velikog broja elektroničkih upravljačkih jedinica unutar automobila koristi UTP kabel dizajniran za pouzdanost u okruženjima s elektromagnetskim šumom.

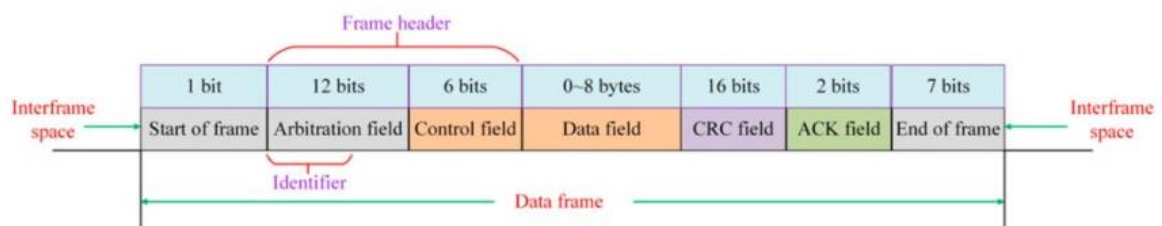
CAN se značajno razvijao od 1986. godine te su od tada razvijene sljedeće implementacije CAN sabirnice:

- CAN male brzine (LS-CAN),
- CAN velike brzine (HS-CAN),
- CAN s mogućnošću fleksibilne brzine prijenosa podataka (CAN FD),
- Treća generacija, poznata kao *extra-long* CAN (CAN XL).

CAN je protokol temeljen na okvirima koji se koristi na multipleksnim ožičenjima gdje podaci putuju slijedno u oba smjera po istoj žici unutar automobilske elektronike kako bi se smanjila količina kablenskog opterećenja i podržale visoke brzine prijenosa podataka. Ako više od jednog čvora odašilje podatke istovremeno, prijenosi podataka su strukturirani tako da osiguraju da čvor s najvećim prioritetom prvi dobije pristup sabirnici, a ostali čvorovi su u stanju čekanja. Primjerice, tijekom vožnje stalno se dobiva informacija o trenutnoj brzini vozila, no u slučaju nalijetanja na drugo vozilo prevelikom brzinom signal naglog kočenja vozila bi trebao dobiti najveći prioritet kako bi reakcija bila što je moguće brža. [21] [22]

4.2.1 Struktura CAN okvira

CAN podatkovni okvir sastoji se od sedam različitih polja bitova: početak okvira, arbitražno polje, kontrolno polje, podatkovno polje, CRC polje, polje znaka potvrde (ACK) i kraj okvira, Slika 5.

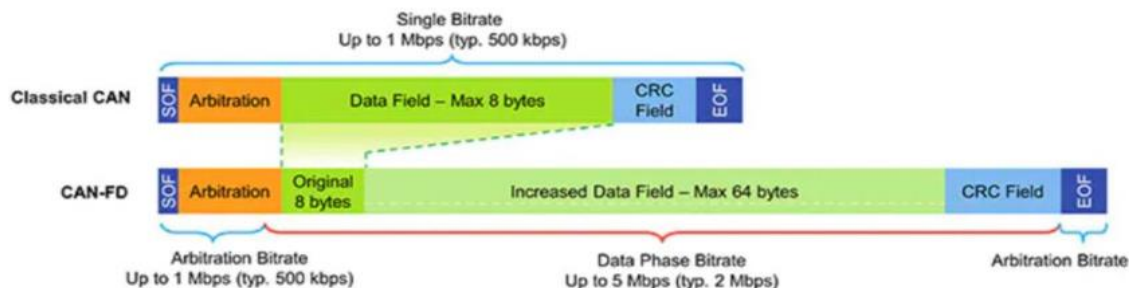


Slika 5. CAN okvir (izvor [22])

Karakteristike elementa takve strukture okvira su sljedeće:

- Početak okvira (engl. *Start of Frame*) sastoji se od jednog dominantnog bita.
- Arbitražno polje uključuje 11-bitni identifikator koji postavlja prioritet poruke i RTR (engl. *Remote Transmission Request*) bita koji su dominantni kada jedan čvor zahtjeva informacije od drugog čvora.
- Kontrolno polje sastoji se od šest bitova gdje prvi bit predstavlja identifikator formata (engl. *Identifier Extension*), drugi bit je rezerviran za buduće namjene, a preostala četiri bita predstavljaju duljinu polja podataka DLC (engl. *Data Length Code*).
- Podatkovno polje (engl. *Data field*) šalje međuspremnik koji sadrži stvarne podatke duljine određene DLC duljinom. Polje CRC (engl. *Cyclic Redundancy Check*) sastoji se od CRC duljine 15 bitova i CRC graničnog znaka koji služe za otkrivanje pogrešaka.
- ACK (engl. *ACKnowledge*) polje sadrži ACK bit potvrde i ACK bit razdvajanja koji se šalju kao recesivan bit. Svi čvorovi koji prime ispravan CRC niz mijenjaju recesivni bit u dominantni dok čvorovi koje prime pogrešnu poruku ostavljaju bit recesivnim i ignoriraju poruku. [21] [22]

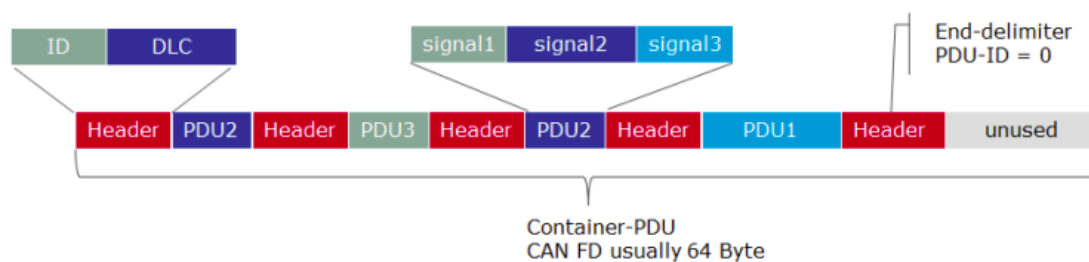
CAN FD skraćeno od CAN (engl. *Flexible Data-rate*) pruža značajno povećanje brzine u usporedbi s klasičnom izvedbom HS-CAN velike brzine, Slika 6. Veća brzina, u kombinaciji s maksimalnim povećanjem veličine korisnog opterećenja s osam do najviše 64 bajta, osmišljena je za značajno poboljšanje propusnosti CAN FD mreža. CAN FD protokol koristi fleksibilnu brzinu prijenosa podataka koja kombinira nominalnu (arbitražna) brzinu prijenosa koja je ograničena na 1 Mbit/s kao u klasičnom CAN protokolu i brzinu prijenosa u podatkovnom polju koja može biti i do 8 Mbit/s. Promjenom brzine upravlja bit BRS (engl. *Bit Rate Switch*) koji je smješten unutar arbitražnog polja CAN FD okvira. [22] [23]



Slika 6. CAN FD okvir (izvor: [22])

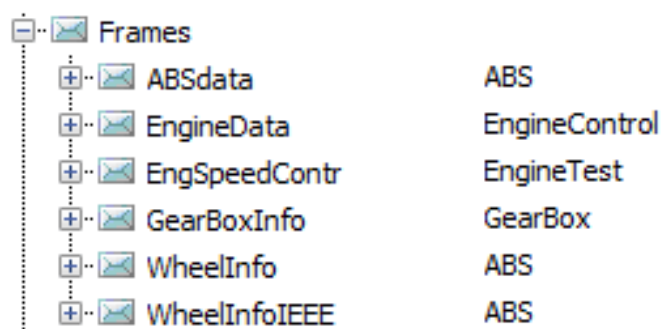
4.2.2 CAN Motbus baza podataka

CAN DBC (engl. *CAN database*) je tekstualna datoteka koja sadrži informacije za tumačenje sirovih podataka CAN sučelja. Važne informacije koje opisuju jedan CAN okvir su ID okvira, naziv okvira, duljina okvira, vrsta formata (CAN ili CAN FD), vrijeme ciklusa odašiljanja okvira, i lista signala koja se nalazi unutar podatkovnog polja. Svaki signal ima svoj naziv, početni bit, broj bitova koje zauzima, vrstu bitova i ostale podatke koje služe za parsiranje podatkovnog polja. Sadržaj CAN DBC baze podataka može se transformirati u XML oblik ili nešto složeniji ARXML oblik. ARXML datoteka je konfiguracijska datoteka spremljena u tzv. AUTOSAR XML formatu kojeg koristi AUTOSAR inicijativa proizvođača i dobavljača automobila. AUTOSAR je osnovan 2003. radi uspostavu softverske arhitekture za automobilske elektroničke upravljačke jedinice skraćenog naziva ECU. ARXML datoteke sadrže podatke o konfiguraciji i specifikacijama u XML formatu za ECU jedinice što se koristi za kontrolu komponenti motora radi postizanja optimalnih performansi. Okviri iz ARXML baze podataka mogu biti samo CAN FD vrste s veličinom do 64 bajta koji se mogu modelirati u manje podatkovne jedinice kao tzv. PDU (engl. *Protocol Data Unit*) kontejneri tako da podatkovno polje sadrži više PDU okvira koji unutar svog podatkovnog polja sadrže signale. ARXML baze mogu sadržavati okvire koji imaju stotine signala pa se raspoređivanjem signala u manje podatkovne jedinice ostvaruje preglednost i lakše pronalaženje signala. Svaki PDU kontejner ima svoje zaglavlje koje se sastoji od 24 bitnog identifikacijskog polja i 8 bitnog DLC polja. [24] [25] [26]



Slika 7. PDU kontejner (izvor: [26])

Motbus baza podataka je manja XML baza koja se sastoji od šest izabranih CAN okvira kreirana od globalne tehnološke tvrtke ZF koja isporučuje sustave za osobna vozila. Na slici 8. prikazani su okviri Motbus baze podatka unutar profesionalnog alata Vector Logger Configurator.



Slika 8. CAN okviri Motbus baze podataka

Svaki od tih okvira može u sebi sadržavati više signala koji mu pripadaju, Slika 9.

Name	/	Tx Node	Channel Mask	Start Bit	Comment
motbus					
+ Signals					
+ Frames					
+ ABSdata		ABS			
+ CarSpeed		ABS		0	
+ Diagnostics		ABS		16	
+ GearLock		ABS		15	
+ EngineData		EngineControl			
+ EngSpeedContr		EngineTest			
+ GearBoxInfo		GearBox			
+ WheelInfo		ABS			
+ WheelInfoIEEE		ABS			

Slika 9. Signali ABSdata okvira

Sljedeći ispis prikazuje sadržaj okvira ABSdata u XML obliku i prvi signal naziva CarSpeed, odnosno brzina vozila.

```
<TxMessage DB_ID="6">
  <Name>ABSdata</Name>
  <ID>0xC9</ID>
  <Frametype>CAN Standard</Frametype>
  <DLC>3</DLC>
  <Comment></Comment>
  <Attribute>
    <Name>CycleTime</Name>
    <Value>50</Value>
    <DefaultUsed>No</DefaultUsed>
  </Attribute>
  <Signal DB_ID="17">
    <Name>CarSpeed</Name>
    <Bitposition>0</Bitposition>
    <Bitsize>10</Bitsize>
    <Byteorder>Intel</Byteorder>
    <Valuetype>Unsigned</Valuetype>
    <Factor>0.5</Factor>
    <Offset>0</Offset>
    <Minimum>0</Minimum>
    <Maximum>300</Maximum>
    <Unit>km/h</Unit>
    <Comment></Comment>
  </Signal>
</TxMessage>
```


Vidi se polje *ID* kao identifikacijski ključ okvira s vrijednosti 0xC9 ili 201, vrsta okvira kao običan CAN, veličina okvira od tri bajta i ciklus poruke od 50 milisekundi. Nadalje, za *CarSpeed* signal vidi se da mu je početna pozicija 0 unutar podatkovnog polja okvira, veličina signala 10 bita kao nepredznačeni cjelobrojni tip podatka *Unsigned* što određuje mogući raspon vrijednosti od 0 do 1023. Ako se ta vrijednost signala pomnoži s faktorom definiranim poljem *Factor* i zbroji s pomakom definiranim poljem *Offset* dobiva se stvarna signala brzine vozila po formuli. [27] [28]

$$\text{signal_value} = \text{decimal_value} * \text{Factor} + \text{Offset}$$

Za signal *CarSpeed* prikazan u ispisu 1 poredak bajtova tipa Intel što znači da je najznačajniji bajt podatka smještan na lokaciju s najvećom adresom. Ako nadolazeći promet kroz CAN kanale sadrži okvire s ID poljem 0xC9 i podatkovnim poljem 0x6c0064 unutar *Dashboard* prozora *LIVE CAN Trace* moguće je vidjeti fizičke vrijednosti signala za taj okvir, Slika 10.

Interface Statistics Live CAN Trace Live CAN Signal XCP Measurement		
Channel:	LR1_CAN3	Database: motbus
	<input type="checkbox"/> Filters	<input type="checkbox"/> II
Timestamp	Parsed Message	Raw Message
16.01.2023 16:12:54.7920	ABSData: CarSpeed=54[km/h]; GearLock=Gear_Lock_Off (0); Diagnostics=100	6C 00 64
16.01.2023 16:12:54.8419	ABSData: CarSpeed=54[km/h]; GearLock=Gear_Lock_Off (0); Diagnostics=100	6C 00 64
16.01.2023 16:12:54.8937	ABSData: CarSpeed=54[km/h]; GearLock=Gear_Lock_Off (0); Diagnostics=100	6C 00 64
16.01.2023 16:12:54.9436	ABSData: CarSpeed=54[km/h]; GearLock=Gear_Lock_Off (0); Diagnostics=100	6C 00 64
16.01.2023 16:12:54.9925	ABSData: CarSpeed=54[km/h]; GearLock=Gear_Lock_Off (0); Diagnostics=100	6C 00 64

Slika 10. Parsirani okviri unutar Dashboard prozora

Bitovi koji predstavljaju *CarSpeed* signal su zadnjih 10 bitova podatkovnog polja 00 0110 1100 što predstavlja vrijednost 108 u cjelobrojnom zapisu. Ako se taj broj pomnoži s faktorom 0.5 i zbroji s pomakom 0 kao rezultat se dobije se vrijednost brzine od 54 km/h.

4.3 Temelji testnog projekta

Arhitektura testnog projekta temeljena je na objektno orijentiranoj paradigmi uz pridržavanje određenih pravila PyTest radnog okvira. Glavna svrha projekta je kontinuirano testiranje postojećih i novih funkcionalnosti *Dashboard* aplikacije kojom se upravlja logRECORDER uređaj. Projekt je izrađen tako da nudi mogućnost testiranja pojedinog sučelja (CAN, Video, Ethernet, LIN, UART) i testiranja svih sučelja zajedno. Ako neko od sučelja nije pokriveno testovima ili nije u potpunosti razvijeno neće sprječavati rad ostalih sučelja i njihovog testiranja kao cjeline. S obzirom da se radi veliki broj testova unutar sustava potrebno je izdvojiti zajedničke korake za pojedine skupine testa unutar baznih klasa kako bi se smanjila količina koda i nepotrebno ponavljanje. Time je refaktoriranje programskog koda olakšano jer se mijenjanjem pojedine metode unutar bazne klase ona automatski mijenja i za sve klase koje ju nasljeđuju, Slika 11. Isto tako, *Dashboard* aplikacija nudi niz istih mogućnosti (konfiguriranje, snimanje prometa, reproduciranje snimaka) za više različitih sučelja pa će testni postupak koji testira pojedino sučelje imati iste testne korake kao i testni postupci za ostala sučelja, ali drugačiju implementaciju. Testni koraci predstavljaju sve metode koje poziva funkcija *run_steps()* unutar testnog postupka. Takve metode sadrže dekorator *@step* koji omogućuje ispis imena metode unutar konzole te njezin ishod. Testni postupak se smatra uspješnim ako su sve provjere unutar svih testnih koraka zadovoljene. U nastavku rada koristiti će se izraz *test* koji predstavlja jedan testni postupak.

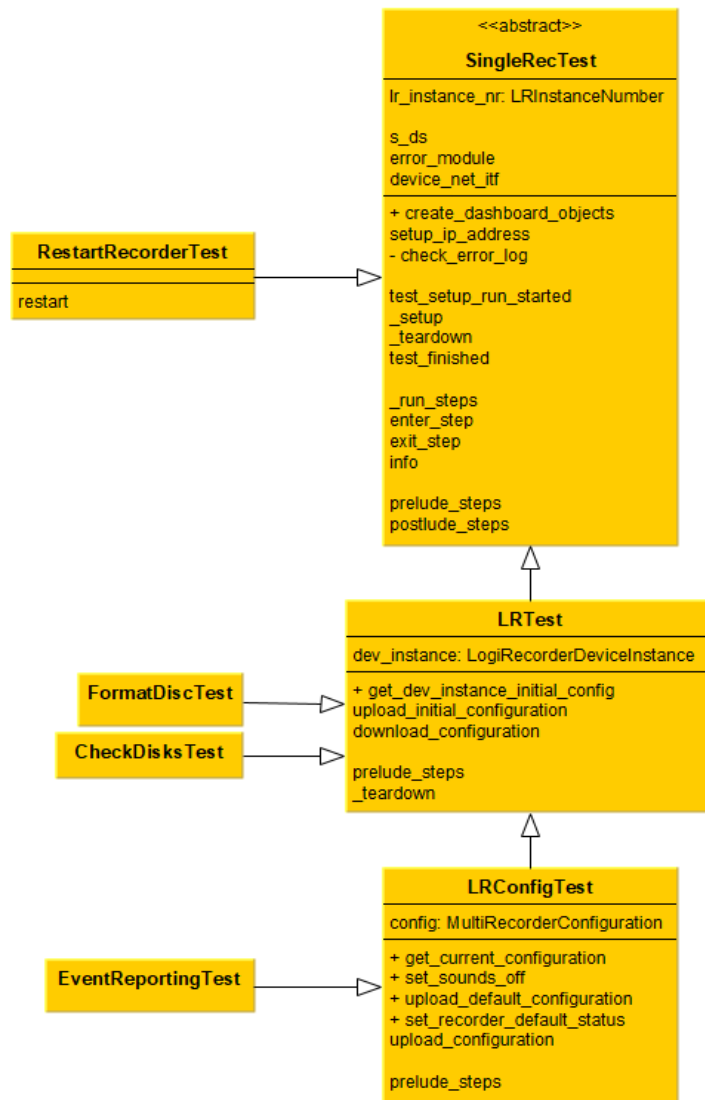
Recimo da se žele kreirati testovi koji će testirati snimljeni promet za LIN i testovi koji će testirati snimljen promet za CAN. Tada će se napraviti bazni test koji će sadržavati zajedničke korake kao što su postavljanje konfiguracije i njezino slanje na *firmware*, pokretanje snimanja, slanje prometa, zaustavljanje snimanja i provjera snimljenog prometa. Testovi za CAN i LIN naslijedit će bazni test sa zasebno specificiranim metodama za konfiguraciju i slanje prometa kako su to jedini koraci koji se međusobno razlikuju za pojedino sučelje .

Svi testovi unutar projekta imaju zajedničku klasu *SingleRecTest* u kojoj su definirani svi potrebni testni koraci za svaki test unutar projekta, Slika 11.

To podrazumijeva kreiranje instanci svih potrebnih klasa iz *Dashboard* projekta, kreiranje potrebnih objekata, pokretanje određenih dretvi kao što su dretva za praćenje serijskog porta, dretva za upisivanje podataka u bazu podataka, dretva za praćenje HTTP prometa i dretva koja prati logiRECORDER događaje.

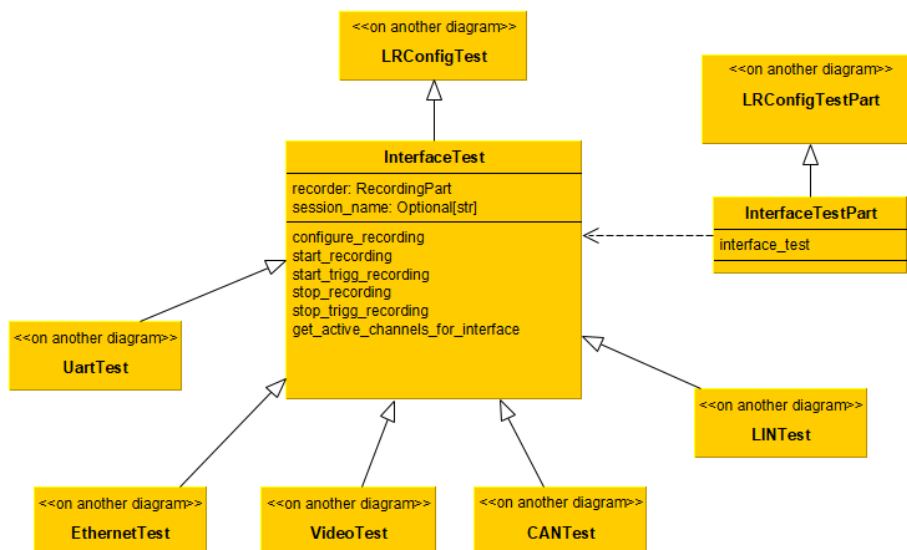
Testovi koji su izvedeni iz *SingleRecTest* klase su vrlo jednostavni i najčešće odrađuju samo jednu određenu provjeru. Ti testovi ne mijenjaju konfiguraciju, njihova uloga je testiranje samo jedne funkcionalnosti kao što je resetiranje logiRECORDER uređaja. Iz *SingleRecTest* klase izvedena je *LRTest* klasa koja nadodaje korake dohvaćanje osnovnog konfiguracijskog dokumenta preko kojeg se stvara inicijalna konfiguracija za logiRECORDER uređaj. Iz *LRTest* klase izvedeni su testovi kao što su nadogradnja *firmwarea* i provjera logiRECORDER diskova. Također postoji pomoćna metoda *teardown()* koja osigurava urednu daljnju provedbu postupka testiranja neovisno o ishodu i uspješnosti pokrenutih testa.

Sljedeća klasa koja je naslijeđena iz *LRTest* klase je *LRConfigTest* koja implementira korake vezane uz slanje konfiguracije na logiRECORDER kao i postavljanje logiRECORDER uređaja u stanje u kojem je spreman za konfiguriranje. Prve tri klase na vrhu arhitekture prikazane su slikom 11.



Slika 11. Dijagram klasa testnih metoda projekta

Klasa koja nasljeđuje klasu *LRConfigTest* je klasa *InterfaceTest* koja predstavlja baznu klasu za testiranje svih sučelja, Slika 12. *InterfaceTest* klasa sadrži testne korake zadužene za konfiguraciju snimanja, pokretanje i zaustavljanje snimanja te dohvaćanje aktivnih kanala s logiRECORDER uređaja za pojedino sučelje ovisno o klasi koja ju nasljeđuje, Slika 12.



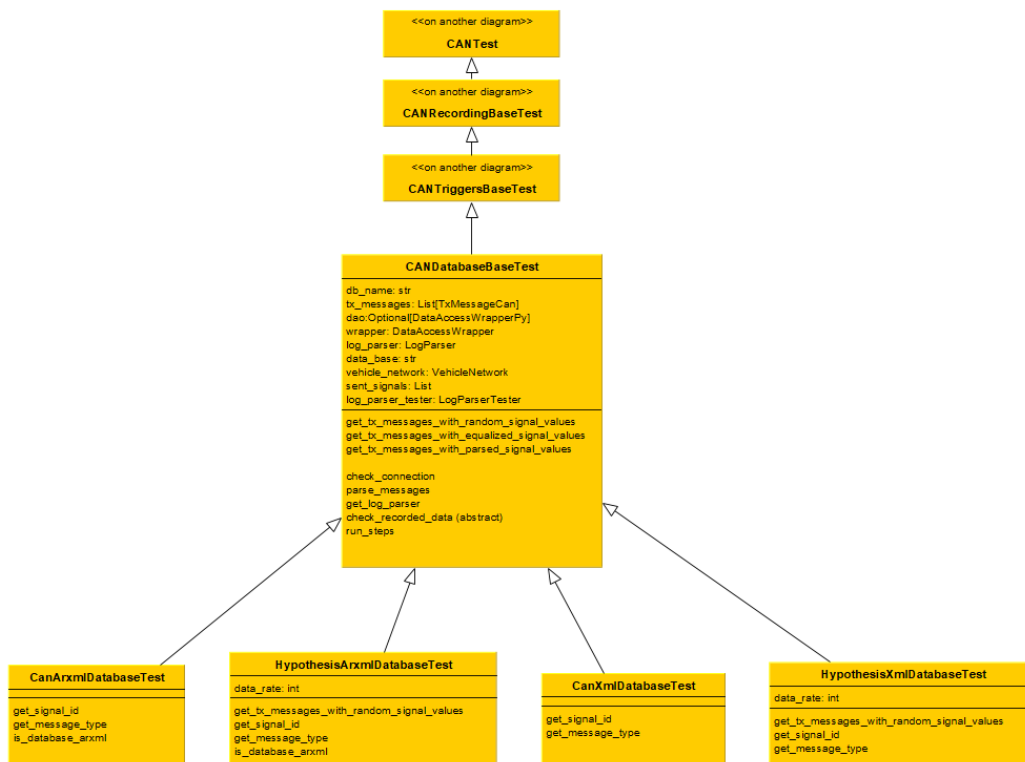
Slika 12. Dijagram klasa testiranih sučelja

Metodom *teardown()* osigurava se izvršavanje pojedinih zadataka nakon što se testovi završe, no isto tako je bitno da se određeni uvjeti ispune i prije izvršavanja samih testa. Za to se koristi PyTest značajka uporabe funkcija učvršćenja koja se izvršava prije pokretanja testa. Kako se testovi pozivaju više puta za drugačije parametre dobro bi bilo numerirati pozive testa odnosno invokacija (engl. *invocation*). Invokacija predstavlja pokretanje testa za kojeg su generirani određeni parametri. Kada se za jedan test generiraju parametri i sve pretpostavke unutar testa budu zadovoljene ponavlja se izvođenje testa s novim generiranim parametrima kao nova invokacija. Tako se omogućava praćenje koliko se test puta izvršio, na kojoj invokaciji je bio neuspješan te se dobiva mogućnost manipulacije nad pozivima testa. Stoga je potrebno prilikom poziva testa funkcijom učvršćenja proslijediti potrebne informacije o tome koje invokacije se žele pokrenuti. Druga funkcija učvršćenja koju je potrebno izvršiti prije pokretanja testa je kreiranje privremene mape u kojeg će se spremati snimke i drugi podaci koji se koriste i unutar testa.

S obzirom da je izabrano CAN sučelje za demonstraciju testiranja nastavlja se testna arhitektura kako je prikazano slikom 13. Klase *CANTest*, *CANRecordingBaseTest* i *CANTriggerBaseTest* implementiraju testne korake zaslužne za konfiguraciju CAN sučelja i slanje CAN prometa. Nakon što je

logiRECORDER uređaj konfiguriran na temelju izvršenih testnih koraka pokreće se snimanje.

Pokretanjem snimanja potrebno je generirati CAN okvire pomoću PCAN USB FD adaptera kojim je moguće upravljati PCAN-View aplikacijom koja ima besplatan API za automatizirano generiranje prometa unutar testa. Kako bi se testirao snimljeni CAN promet i pravilno parsiranje CAN okvira napravljena je klasa *CANDatabaseBaseTest* koja implementira korake potrebne za provjeru parsiranih CAN okvira unutar MySQL tablice. Testovi koji se demonstriraju u poglavlju 5 *CanXMLDatabaseTest* i *HypothesisXmlDatabaseTest* prikazani su na slici 13.



Slika 13. Dijagram klasa testa

Prikazani testovi na dnu testne arhitekture sadržavaju sve testne korake koji su implementirani unutar naslijeđenih klasa te kao takvi moraju biti uspješno izvršeni.

5. Provedba testiranja

U nastavku su prikazani testovi koji pomoću PyTest i *Hypothesis* razvojnih okvira generiraju ulazne parametre za testiranje ispravnosti rada *Dashboard* aplikacije u radu s CAN bazama podataka. To uključuje:

- testiranje snimljenog CAN prometa na svim brzinama prijenosa u tekstualnom formatu za spremanje podataka u *Ros-bag* i binarnom *Mdf4* formatu,
- provjere parsiranja CAN prometa za odabranu bazu podataka gdje se koristi Modbus baza podataka,
- testiranje CAN filtera i CAN okidača o kojima se govori u nastavku rada.

Najprije će se demonstrirati testiranje temeljeno na primjerima gdje se ulazni parametri postavljaju pomoću Pytest parametara. Korištenjem PyTest parametara bez uporabe *Hypothesis* razvojnog okvira testna metoda će se uvijek pozivati s istim zadanim parametrima osim ako se parametri izravno ne promjene unutar koda. Parametar brzina prijenosa odnosi se na brzinu prijenosa CAN okvira, dok se parametrom format snimanja odabire u kojem formatu se žele pohraniti snimljeni CAN okviri. CAN filteri i okidači predstavljaju funkcionalnost unutar *Dashboard* aplikacije kojom je moguće filtrirati nadolazeće CAN okvire po nekom uvjetu ili pokrenuti određenu akciju za specifičan CAN okvir kao što je snimanje ili zvuk zujalice. Generiranjem CAN okvira koji sadrži ID pohranjen unutar izabrane baze trebao bi biti parsiran unutar *Dashboard* aplikacije. To podrazumijeva ispis naziva okvira kao i sve nazive signala i njihove fizičke vrijednosti koji neki okvir sadrži. LogiRECORDER omogućuje snimanje na 16 kanala koji se mogu raspodijeliti na sučelja CAN, Uart i LIN, a unutar ovog testnog scenarija konkretno se radi o kanalima 3 i 4.

5.1 Testiranje temeljeno na primjerima

Testiranje i testni postupak se provodi temeljem testne funkcije koja se opisuje u prvom dijelu poglavlja nakon čega se opisuje primjer pokretanja testa temeljem te iste funkcije.

5.1.1 Struktura testne funkcije

Kako bi se proveli testni koraci koje sadrži klasa *CanXMLDatabaseTest* potrebno je napisati test koja će izvršavati specificirane korake određenim redoslijedom za različite skupove predanih parametara. Parametri se inicijaliziraju unutar varijabli koji se potom predaju testu pomoću PyTest dekoratora *parametrize*. Inicijalizirani parametri i test prikazani su kôdom 13.

Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković

```
# Input parameters-----
-----

input_rates = [1000, 800, 500, 250]

input_filters = [
    [CanLinFilterProps("pass", True, "CAN Channel", "", "Channel", "0")],
    [CanLinFilterProps_1("pass", True, "CAN ID Condition", "3", "Single
        ID", "0", "201")]]
input_triggers = [[],
    [RecCanLinTriggerProps(True, "Marker", "",
        "CAN ID Condition", "Single ID", "0", "201")]]

input_format = [FormatProps("Natural"), FormatProps("MDF4")]

input_data_bases = CanDataBasesPath.get_can_xml_databases_paths()
```



```

# Test function-----
-----

@pytest.mark.can
@pytest.mark.can_database
@pytest.mark.parametrize("flt", input_filters)
@pytest.mark.parametrize("trigg", input_triggers)
@pytest.mark.parametrize("rate", input_rates)
@pytest.mark.parametrize("can_format_param", input_format)
@pytest.mark.parametrize("data_base", input_data_bases)
@pytest.mark.jUnit_test_name("rate", "can_format_param")
def test_parsing_can_xml_database(request, invocations, rate: int,
                                  flt: List[CanLinFilterProps],
                                  can_format_param: FormatProps,
                                  data_base: str,
                                  trigg: List[RecCanLinTriggerProps]):
    """
    Check if can messages from xml document (input database parameters) are
    parsed correctly
    """
    test_suite = test_suite_injector.test_suite_lr()

    test_context = SingleRecTestContext(test_suite, invocations,
                                         "ERROR_TestCAN_Baud1000_",
    __file__)
    runner_context = TestRunnerContext(test_suite, currentframe(),
                                       invocations, request, "SP16001AA-
109")

    if invocations.new_invocation(test_suite.args.invoc_filters, request) \
        == Invoc.ShouldRun:

        with store_info_on_error(test_suite.pg_database, runner_context,
                                invocations):

            test = CanXmlDatabaseTest(test_context, rate, flt, trigg,
                                       can_format_param, data_base)
            runner = SingleTestRunner(test, runner_context)

            runner.setup_run(lambda: test.run_steps())
    else:
        TestRunner(runner_context).skip()
        pytest.skip("Number of test invocations has exceeded the limit")

```

Programski kôd 12. Testna funkcija i ulazni parametri

Testna funkcija `test_parsing_can_xml_database()` prima pet listi ulaznih parametara, a one su:

- `input_rates`: lista koja sadrži četiri brzine prijenosa CAN okvira u kb/s.
- `input_filters`: lista koja sadrži filtre od kojih prva lista sadrži jedan filter koji propušta sve okvire na kanalu i druge liste koja filtrira samo okvire koji imaju ID 201 (=0xC9) predstavljajući okvir ABSData unutar Motbus XML baze.
- `input_triggers`: lista koja sadrži dvije liste od kojih je prva lista prazna što znači da se okidači ne postavljaju unutar konfiguracije i druge liste koja sadrži okidač marker za okvire koji imaju ID 201 ispisujući marker poruku u slučaju detekcije, slika 14.
- `input_format`: lista koja sadrži dva formata snimanja: *Natural(Ros-bag)* i *Mdf4*.
- `input_data_base`: parametar koji sadrži putanju do mape koji sadrži CAN XML baze podataka. Unutar ovog testa mapa sadrži samo Motbus XML bazu.

Pytest poziva test za svaku kombinaciju parametra unutar navedenih listi. Unutar ovog konkretnog slučaja test se poziva 32 puta. Na početku testa kreira se *TestSuiteLR* instanica koji dohvaća argumente postavljene unutar konfiguracijske datoteke projekta. Zatim se kreiraju *SingleRecTestContext* i *TestRunnerContext* instance koje pohranjuju potrebne argumente za svaki test. Nakon kreiranja navedenih instanci preko parametara predanih unutar konfiguracijske datoteke odabire se željena invokacija. Ako invokacija nije odabrana, preskače se i kreće se na novu invokaciju s novim generiranim parametrima. Parametri koji su predani testu predaju se instanci klase *CANXmlDatabaseTest* koji definira svoje korake testiranja i pridružuje ih svim koracima koje je naslijedila. Kada je instanica *CANXmlDatabaseTest* klase kreirana, predaje se *SingleTestRunner* objektu koji pokreće sve korake testiranja pozivom metode `run_steps()`.

5.1.2 Primjer pokretanja testa

PyTest naredbom test se izvršava za sve kombinacije predanih parametara, a nakon završetka svih 32 poziva testa rezultati testa mogu se analizirati unutar

PostgreSQL tablice *invocations*. Tablica *invocations* sadrži sljedeće kolone: broj testa integracije, vrijeme pokretanja invokacije, trajanje invokacije, ime testa, redni broj invokacije, ishod, testni korak na kojemu se dogodila greška, poruka uz grešku te ulazni parametri. Test se izvršio uspješan za sve invokacije, a invokacija broj 32 unutar *invocations* tablice prikazana je slikom 14.

	id [PK] integer	build integer	start [PK] timestamp without time zone	duration time without time zone	test text	invoc integer	outcome text
1	30664	10746	2023-03-25 13:10:57.279841	00:00:18	test_parsing_can_xml_database	32	Passed

Slika 14. Prikaz invokacije 32 unutar tablice *invocations*

Na slici 14 prikazani su prvih sedam stupaca redaka u kojima je prikazana uspješnost izvršavanja invokacije 32. Parametri generirani za prikazanu invokaciju su:

```
flt: [pass, True, CAN ID Condition, 3, Single ID, 0, 201],
rate: 250,
trigg: [Marker, True, CAN ID Condition, Single ID, 0, 201],
data_base: "C:\\j\\workspace\\Projects\\logiRecorder\\Integration_Testing\\
            integration-tests-current-2\\CANFiles\\xml_test_databases\\
            can_databases\\motbusAll.xml",
can_format_param: MDF4
```

Kako je test imao uspješan ishod za sve invokacije pa tako i zadnju invokaciju koja sadrži prikazane parametre *SignalView* tablica unutar MySQL baze za zadnju invokaciju treba sadržavati samo CAN ABSdata okvire na kanalu 3 i markere za taj okvir. Jedan korak unutar testa zadužen za generiranje CAN okvira generira po jedan okvir iz baze podataka. Motbus baza sadrži šest okvira od kojih je jedan okvir ABSdata. Prikaz *SignalView* tablice za invokaciju 32 prikazana je slikom 15.

TimeStamp	MessageType	Channel	MessageName	SignalName	Value	Unit	IDSignal	IDRawmessage	LogString	LogBinary
0.885885821	1	3	ABSdata	CarSpeed	4	km/h	237121044	140864843	NULL	BLOB
0.885885821	1	3	ABSdata	GearLock	Gear_Lock_On (1)		237121045	140864843	NULL	BLOB
0.885885821	1	3	ABSdata	Diagnostics	58		237121046	140864843	NULL	BLOB
0.885885821	9	0	NULL	NULL	NULL	NULL	NULL	140864842	Trigger 201	NULL

Slika 15. Prikaz invokacije 32 unutar tablice *invocations*

Na slici 15 vidljivo je kako se unutar *SignalView* stupca nalaze samo signali iz ABSdata okvira na kanalu 3 i jedan marker okidač za ABSdata ID što potvrđuje uspješnost testa (*test_parsing_can_xml_database*, slika 13) i ispravnost testiranih komponenti.

5.2 Testiranje temeljeno na svojstvima

Kreirani test iz prethodnog poglavlja *test_parsing_can_xml_database* testira pravilnost rada parsiranja CAN okvira u integraciji s ostalim funkcionalnostima kao što je izmjena konfiguracije logiRECORDERA, snimanje prometa, filtriranje prometa i testiranje okidača za CAN okvire. Test odrađuje svoju osnovnu funkcionalnost, ali pokriva premalo ulaznih parametara. *Dashboard* aplikacija uključuje mogućnost kombiniranja više filtera i okidača istovremeno koji bi se trebali ispravno ponašati za bilo koji CAN okvir iz predane baze kao i za sve signale koji se nalaze unutar njega. Testiranje koje uključuje jedan filter i jedan okidač za specifični CAN okvir iz baze pokriva vrlo uzak raspon mogućih ulaza. Za demonstraciju testiranja temeljenom na svojstvima potrebno je izmijeniti kreirani test prikazanim kodom 13 tako da se ulazni parametri kreiraju strategijama korištenjem *Hypothesis* biblioteke. Testni postupak se prema tome najprije prikazuje opisom strukture testnih funkcija nakon čega slijede primjeri izvršavanja testa koji se na temelje na odgovarajućim testnim funkcijama.

5.2.1 Struktura testne funkcije

Kako bi se generirali potrebni parametri testa stvorena je kompozitna funkcija *get_can_database_test_parameters()*, a struktura testnog koda je sljedeća:

- Funkcija *get_can_database_test_parameters* koja poziva nekoliko pomoćnih funkcija:
 - *get_baud_rates* koja generira različite brzine prijenosa,
 - *get_tx_messages* koja dohvaća sve CAN okvire iz baze podataka,
 - *get_signal_with_random_values* koja generira nasumične vrijednosti signala.

- Testna funkcija *test_parsing_xml_hypothesis* poziva testne korake:
 - *run_steps* koja pokreće sve korake jednog testa.

Glavna kompozitna funkcija *get_can_database_test_parameters()* ima sposobnost generiranja željenih parametara iz ostalih strategija pomoću posebne funkcije *draw()*. Kreirani parametri prosljeđuju se testnoj funkciji pomoću dekoratora *@given* u obliku podatkovne klase *DatabaseHypothesisParameters*. Kompozitna funkcija *get_can_database_test_parameters()* prikazana je programskim kodom 14.

Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković

```
@composite
def get_can_database_test_parameters(draw) -> DatabaseHypothesisParameters:

    # CAN FD or CAN
    can_fd: bool = draw(booleans())

    # CAN Baud Rate and Data baud Rate
    data_baud_pair= draw(get_baud_rates(can_fd))

    # CANFD or CAN database from /CANFiles
    data_base = draw(sampled_from(can_fd_xml_databases_paths)) if can_fd \
        else draw(sampled_from(can_xml_databases_paths))

    # Recording format
    rec_format: FormatProps = draw(sampled_from([FormatProps("Natural"),
                                                FormatProps("MDF4")]))

    # All TxMessageCan from database
    tx_messages = seq(XmlExtractor(data_base).get_tx_messages())\
        .distinct_by(lambda m: m.id).to_list()

    # Randomly generated TxMessageCan from database
    sampled_tx_messages = draw((lists(sampled_from(tx_messages),
                                         min_size=int(len(tx_messages) / 2),
                                         max_size=len(tx_messages))))

    # Filter list temporarily set as Channel Filter
    can_filters = [CanLinFilterProps(
        "pass", True, "CAN Channel", 3, "Channel", "0"),
        CanLinFilterProps(
        "pass", True, "CAN Channel", 4, "Channel", "0")]

    # Trigger list temporarily set as empty list!!!
    can_triggers = []
```

```

# Adding random signal values within the message
messages_with_generated_signals = seq(sampled_tx messages)\
    .map(lambda m: m.replace_signals(seq(m.signals)\
    .map(lambda s: s.replace_value(draw
    (get_signal_with_random_value(s))))).to_list()))\
    .to_list()

# Randomly generated filters
return DatabaseHypothesisParameters(data_baud_pair, can_filters,
    rec_format, can_triggers,
    messages_with_generated_signals,
    data_base)

```

Programski kôd 13. Kompozitna funkcija za kreiranje parametara testa

Najprije se generira varijabla *can_fd* kojom se odlučuje hoće li test provesti na CAN ili na CAN FD bazi podataka. Zatim se poziva *get_baud_rates()* funkcija koja vraća kombinaciju nominalne brzine prijenosa i brzine prijenosa podatkovnog polja. Ako se radi o CAN sučelju tada je moguća samo nominalna brzina prijenosa dok je za CAN FD moguće dobiti bilo koju kombinaciju sve dok ona zadovoljava funkciju *assume()* koja ne dozvoljava razliku tih dviju brzina veću od 4500 Kbit/s zbog ograničenja unutar *Dashboard* aplikacije. Kompozitna strategija koja vraća kombinaciju brzina prikazana je sljedećim programskim kodom 15.

Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković

```

@composite
def get_baud_rates(draw, can_fd) -> BaudRatePair:
    """Returns randomly generated CAN rates in kb/s

    if can_fd:
        baud = draw(sampled_from([1000, 800, 500,
                                   250, 125, 100]))
        data_baud = draw(sampled_from([8000, 6000, 4000,
                                         3000, 2000, 1000, 800, 500, 0]))
        assume(data_baud > baud)
        assume(data_baud - baud < 4500)
    else:
        data_baud = 0
        baud = draw(sampled_from([1000, 800, 500,
                                   250, 125, 100, 83, 50, 33]))
    return BaudRatePair(baud, data_baud)

```

Programski kôd 14. Dobivanje brzine prijenosa podataka

LogiRECORDER omogućuje snimanje u *Mdf4* i *Ros-bag* formatu, a koji će se format koristiti nasumično će odlučiti strategija *sampled_from()* koja uzima jedan uzorak iz liste. U nastavku nasumično se dohvaća jedna od CAN ili CAN FD XML baza podataka iz koje se uzimaju svi CAN okviri, a zatim se kreira uzorak nasumično dohvaćenih okvira u rasponu od jedne polovice ukupnog broja okvira do ukupnog broja okvira.

Dashboard aplikacija omogućuje filtriranje CAN prometa po kanalu, ID-u okvira ili rasponu ID okvira, PDU ID kontejnera ili njegovom rasponu. Filtere je primjerice moguće kombinirati tako je moguće postaviti filter koji će propustiti sav promet na jednom kanalu dok na će na drugom propustiti samo okvire s određenim ID-em. Kao i filtere, okidače je isto moguće međusobno kombinirati, a uvjeti okidanja su ID okvira, PDU ID kontejnera i fizička vrijednost signala. Ponekad nije dobro krenuti s testiranjem svih mogućih komponenti na samom početku zbog otežanog i složenijeg pronalaženja potencijalnih grešaka. Kako bi se najprije detaljno testiralo samo parsiranje CAN okvira iz baze podataka bez dodatnih filtera i okidača, lista filtera postavljena je tako da se kroz kanale 3 i 4 propuštaju svi okviri, a lista okidača postavljena je na praznu listu kako je prikazano kodom 14. Kada se provede testiranje parsiranja okvira bez dodatnih filtera i okidača te se dokaže da ta komponenta radi može se krenuti s dodavanjem dodatnih komponenti unutar testa. S obzirom da podatkovno polje CAN okvira sadrži signale i njihove vrijednosti kreirana je kompozitna funkcija *get_signal_with_random_value()* koja nasumično generira moguće vrijednosti signala pojedinog CAN okvira prikazana kôdom 16.

Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković

```
@composite
def get_signal_with_random_value(draw, signal: MessageSignal) -> float:
    """A strategy that returns randomly generated value for given signal"""

    gen_value: float = 0

    if signal.signal.Valuetype == "Unsigned":

        gen_value = draw(integers(
            min_value=0, max_value=pow(2,
int(signal.signal.Bitsize) - 1)))
```

```

elif signal.signal.Valuetype == "Signed":

    min_value = - pow(2, int(signal.signal.Bitsize) - 1)
    max_value = pow(2, int(signal.signal.Bitsize) - 1) - 1
    gen_value = draw(integers(min_value=min_value,
                              max_value=max_value))

elif signal.signal.Valuetype == "IEEE Float":
    if signal.signal.Bitsize == 32:
        gen_value = draw(floats(
            width=32, allow_infinity=False,
allow_nan=False))
    elif signal.signal.Bitsize == 64:
        gen_value = draw(floats(width=64, allow_infinity=False,
allow_nan=False))

return gen_value

```

Programski kôd 15. Generiranje nasumične vrijednosti signala

Ovisno o tome o kojem tipu podatka signala se radi, pomoću strategija *integers()* i *floats()* generira se moguća vrijednost za dotični signal. Iako su sve vrijednosti za signale nasumično generirane neuspjelim ishodom testa *Hypothesis* nakon njegova izvršavanja ispisuje dekorator *@reproduce_failure* kojim je moguće reproducirati sve generirane vrijednosti za koje se test pokazao neuspješnim. Kada su generirani svi parametri potrebni za izvršavanje testa potrebno ih je proslijediti pomoću dekoratora *@given*. Iznad testne funkcije specificirane su neke od postavki *Hypothesis* razvojnog okvira kao i markeri za testnu metodu. Testna funkcija *test_parsing_xml_hypothesis()* prikazana je kodom 17.

Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković

```

@pytest.mark.can_database
@pytest.mark.db_parsing
@settings(deadline=None, max_examples=50, print_blob=True,
          phases=(Phase.explicit, Phase.reuse, Phase.generate,
Phase.target),
          suppress_health_check=HealthCheck.all())
@given(parameters=get_can_database_test_parameters())
def test_parsing_xml_hypothesis(request, invocations,
                                parameters: DatabaseHypothesisParameters):
    """
    Check if can messages from xml document (input_data_bases parameter)
    are parsed correctly

```



```

    The test uses xml databases inside: CANFiles\xml_test_databases
    """
    test_suite = test_suite_injector.test_suite_lr()

    test_context = SingleRecTestContext(test_suite, invocations,
                                         "ERROR_TestCAN_Baud1000_",
    __file__)
    runner_context = TestRunnerContext(test_suite, currentframe(),
                                       invocations, request, "SP16001AA-
109")

    if invocations.new_invocation(test_suite.args.invoc_filters, request)
    == Invoc.ShouldRun:

        with store_info_on_error(test_suite.pg_database, runner_context,
                                invocations):

            test = HypothesisXmlDatabaseTest(test_context,
                                             parameters.baud_rate_pair,
                                             parameters.can_filters,
                                             parameters.can_triggers,
                                             parameters.rec_format,
                                             parameters.data_base,
                                             parameters.\

messages_with_generated_signals)
            runner = SingleTestRunner(test, runner_context)

            runner.setup_run(lambda: test.run_steps())
    else:
        TestRunner(runner_context).skip()
        pytest.skip("Number of test invocations has exceeded the limit")

```

Programski kôd 16 . Testna metoda *test_parsing_xml_hypothesis()*

Iznad testne funkcije postavljeni su PyTest markeri *can_database* i *db_parsing* kao bi se test grupirao u posebne skupine. Pomoću *Hypothesis* dekoratora *@settings* postavljene su postavke koje uključuju broj izvođenja od 50 invokacija, isključen je vremenski rok za generiranje pojedinačnog primjera, omogućeno je reproduciranje neuspješnih testa, odabrane su željene faze koje se žele izvršiti na primjerima te su isključena upozorenja za predugo trajanje generiranja strategija zbog složenosti strategije. Testna funkcija *test_parsing_xml_hypothesis()* vrlo je slična metodi *test_parsing_can_xml_database()* prikazanim kodom 13 samo što se unutar nje kreira instanca klase *HypothesisXmlDatabaseTest* kod koje su svi ulazni parametri nasumično generirani strategijama.

5.2.2 Primjeri izvršavanja testa

Provedena su dva primjera testiranja parsiranja CAN prometa. U prvom nije uključena provjera filtara i okidača, a u drugom se radi provjera parsiranja s postavljenim filtrima i okidačima.

Primjer testiranja parsiranja CAN prometa

Ulazni parametri testa su generirani strategijama prikazanim kôdom 14, gdje se za inicijalizaciju listi filtera i okidača ne koristi strategija već su one postavljene na fiksnu vrijednost kao bi se test pojednostavio i kako bi fokus testiranja bio na parsiranju CAN okvira. Predani parametri za prvu invokaciju su bili:

```
BaudRatePair (nominal=1000, data=0),  
Filters: [pass, True, CAN Channel, 3, Channel, 0, pass, True, CAN Channel,  
4, Channel, 0],  
RecFormat: Natural,  
Triggers: [],  
messages_with_generated_signals: size=6,  
database: motbusAll.xml
```

Kako se radi o standardnom CAN okviru generirana je samo njegova nominalna brzina prijenosa od 1000 kbit/s. Filteri i okidači su za sada fiksni, a broj okvira koji je generiran je jedan okvir iz Motbus XML baze. Prva invokacija je uspješno prošla za generirane parametre što nam pokazuje tablica *invocations* unutar PostgreSQL baze. Prve četiri kolone za prvu invokaciju prikazane su na slici 16.

id [PK] integer	build integer	start [PK] timestamp without time zone	duration time without time zone	test text	invoc integer	outcome text
20022	9578	2023-01-21 13:53:25.829106	00:02:08	test_parsing_xml_hypothesis	1	Passed

Slika 16. Ishod prve invokacije

Okvir koji je generiran i njegovi signali mogu se pogledati u MySQL bazi unutar tablice *SignalView* prikazana na slici 17.

TimeStamp	MessageType	Channel	MessageName	SignalName	Value	Unit	IDSigal	IDRawmessage	LogString
0.893660628	1	3	ABSdata	CarSpeed	0	km/h	237121044	140864842	0.893660628 3 C9 Rx d 3 00 00 00
0.893660628	1	3	ABSdata	GearLock	Gear_Lock_Off (0)		237121045	140864842	0.893660628 3 C9 Rx d 3 00 00 00
0.893660628	1	3	ABSdata	Diagnostics	0		237121046	140864842	0.893660628 3 C9 Rx d 3 00 00 00
0.893660628	1	4	ABSdata	CarSpeed	0	km/h	237121047	140864843	0.893660628 4 C9 Rx d 3 00 00 00
0.893660628	1	4	ABSdata	GearLock	Gear_Lock_Off (0)		237121048	140864843	0.893660628 4 C9 Rx d 3 00 00 00
0.893660628	1	4	ABSdata	Diagnostics	0		237121049	140864843	0.893660628 4 C9 Rx d 3 00 00 00

Slika 17. Ishod prve invokacije

Okvir koji je poslan adapterom na oba kanala je ABSdata s ID-em okvira 0xC9. Veličina podatkovnog polja okvira je tri bajta i oni su 0x00 0x00 0x00. Kada se vrijednosti podatkovnog polja parsiraju dobivaju se vrijednosti prikazane pod stupcem *Value*. Kako su sve provjere prošle za prvu invokaciju preostaje izvršavanje preostalih 49 invokacija ili sve dok jedna provjera bude kriva i test prestaje s izvršavanjem. Novim pozivom testa invokacija broj 2 ne uspijeva zadovoljiti pretpostavku provjere vrijednosti signala unutar *check_recorder_data()* testnih koraka. *Hypothesis* ponovno pokušava izvršiti sljedeću invokaciju s istim parametrima, no njezin ishod je isti kao kod druge invokacije kao što je prikazano unutar posljednja četiri stupca tablicom *invocations*.

invoc integer	outcome text	error_step text	error text
1	Passed		
2	Failed	check_recorded_data	Wrong signal Value, parsed: 1122927903.0, should 119.26 signal name: WheelSpeedFL, session_uid: n121125857q1a184c
3	Failed	check_recorded_data	Wrong signal Value, parsed: 1122927903.0, should 119.26 signal name: WheelSpeedFL, session_uid: n121130154q1a184f

Slika 18. Ishod druge i treće invokacije

Parametri koji su generirani za drugu invokaciju:

```
BaudRatePair (nominal=500, data=0),
Filters: [pass, True, CAN Channel, 3, Channel, 0, pass, True, CAN
Channel, 4, Channel, 0],
RecFormat: MDF4,
Triggers: [],
messages_with_generated_signals: size=3
database: motbusAll.xml
```

Slika 18. prikazuje usporedbu vrijednosti signala WheelSpeedFL izračunata unutar testnog koraka koja iznosi 119.26 1/min i vrijednosti koja je izračunata unutar *Dashboard* aplikacije koja iznosi 1122937903.0 1/min.

Kako bi se utvrdila točna vrijednost koristi se Vectorov alat CANalyzer Pro pomoću kojeg je moguće utvrditi točnu parsiranu vrijednost za pojedini signal. Na slici 19. prikazane su vrijednosti signala WheelSpeedFL unutar tablice *Signal/View* i vrijednosti unutar CANalyzer alata.

1. Parsed Dashboard value:

0.853378168	1	3	WheelInfoIEEE	WheelSpeedFL	1122927903	1/min	237121051	140864844	0.853378168 3 C7 Rx d 8 1F 85 EE 42 66 E6 B0 C2
0.853378168	1	3	WheelInfoIEEE	WheelSpeedFR	3266373222	1/min	237121052	140864844	0.853378168 3 C7 Rx d 8 1F 85 EE 42 66 E6 B0 C2

2. Canalyzer parsed value:

Time	Chn	ID	Name	Event Type	Dir	DLC	Da...	Data
1.054753	CAN 1	C7	WheelInfoIEEE	CAN Frame	Rx	8	8	1F 85 EE 42 66 E6 B0 C2
			WheelSpeedFL	119.2600 1/min				119.2600
			WheelSpeedFR	-88.4500 1/min				-88.4500

Slika 19. Usporedba vrijednosti WheelSpeedFL signala

Vrijednost signala koja je prikazana unutar CANalyzer alata je 119.2600 1/min. Ta vrijednost slaže se s vrijednosti koja je izračunata unutar testa što ukazuje na pogrešku kod izračuna vrijednosti signala unutar *Dashboard* aplikacije.

Primjer testiranja parsiranja s filterima i okidačima

Kada se isprave sve greške koje su otkrivene unutar prethodnog primjera i sve pretpostavke unutar testa su zadovoljene fiksne liste filtera i okidača mogu se zamijeniti strategijama. Kako bi se to postiglo potrebno je dodati dvije nove kompozitne funkcije *get_filter_list()* i *get_trigger_list()* unutar kompozitne funkcije *get_can_database_test_parameters()*, *Programski kod 14*. Prvo se dodaje kompozitna funkcija koja vraća strategiju nasumično generiranih filtera prikazana programskim kodom 18.

```
@composite
def get_filter_list(draw, tx_messages: Optional[List[Message]],
                    channels: List[int]) -> List[CanLinFilterProps]:
    """A strategy that returns randomly generated CAN filter list for given
    List[Message]"""

    if tx_messages:

        id_filters = seq(tx_messages) \
            .flat_map(lambda t: [CanLinFilterProps_1(draw(
                sampled_from(["pass", "stop"])),
                True, "CAN ID Condition",
                draw(sampled_from(channels)),
                "Single ID", "0",
                str(t.id))] if not t.check_pdu() else
            seq(t.pdus).map(lambda p:
                CanLinFilterProps_1
                (draw(sampled_from(["pass", "stop"])),
                 True, "Symbolic CAN Message",
                 draw(sampled_from(channels)),
                 "Single ID", str(p.id),
                 str(t.id)).to_list()).to_list()

    else:
        list_ids = draw(lists(integers(
            min_value=0,
            max_value=536870911),
            min_size=1, max_size=10))

        id_filters = seq(list_ids) \
            .flat_map(lambda id: [CanLinFilterProps_1
                (draw(
                    sampled_from(["pass", "stop"])),
                    True, "CAN ID Condition",
                    draw(sampled_from(channels)),
                    "Single ID", "0",
                    str(id))].to_list())

    channel_filters = [CanLinFilterProps("pass", True, "CAN Channel",
                                         draw(sampled_from(channels)),
                                         "Channel", "0"),
                       CanLinFilterProps("stop", True, "CAN Channel",
                                         draw(sampled_from(channels)),
                                         "Channel", "0")]

    id_range_filters = draw(get_range_filters(channels, tx_messages))

    filters_combination_list = \
        draw(sampled_from([id_filters, channel_filters, id_range_filters,
            id_filters + channel_filters,
            id_filters + id_range_filters,
            channel_filters + id_range_filters,
            id_filters + channel_filters + id_range_filters]))
    return draw(lists(sampled_from(filters_combination_list), min_size=1))
```

Programski kôd 17. Kompozitna funkcija za generiranje filtera

Funkcija *get_filter_list()* na temelju ulaznih parametara liste CAN okvira i liste kanala vraća listu filtera koji filtriraju CAN okvire. Lista može sadržavati filtere po ID-u ili PDU ID-u nekog okvira iz predane liste ili nasumično generiranog ID-a u rasponu od 0 do maksimalne moguće vrijednosti ($2^{29}-1$). Filtri mogu biti akcije propuštanja ili stopiranja za bilo koji ID ili po cijelom kanalu. Uz navedene filtere mogući su i filteri koji filtriraju sve okvire u nekom rasponu ID-eva, a oni se dobivaju dodatnom kompozitnom funkcijom *get_range_filters()*. Nakon dodavanje kompozitne funkcije *get_filter_list()*, postojeći testovi se ponovno se pokreću za svih 50 invokacija. Generirani parametri za prvu invokaciju testa *test_parsing_xml_hypothesis()* su:

```
BaudRatePair (nominal=500, data=0),
Filters: [pass, True, CAN Channel, 3, Channel, 0,
          pass, True, CAN Channel, 4, Channel, 0,
          stop, True, CAN Single ID, 4, Channel, 0, 201],
RecFormat: Natural,
Triggers: [],
messages_with_generated_signals: size=6
database: motbusAll.xml
```

Nominalna brzina slanja je 500 kbit/s, lista filtera sada sadrži filter koji propušta sve okvire na kanalu 3, filter koji propušta sve okvire na kanalu 4 te posljednji filter koji stopira sve okvire s ID-em 201 na kanalu 4. Format snimanja je postavljen na *Natural*, a okidači su postavljeni na praznu listu. Generirani su svih šest okvira iz *Motbus* XML baze, a jedan od okvira *ABSdata* sadrži ID 201 koji bi trebao biti stopiran na četvrtom kanalu. Pretpostavlja se da je broj poruka koji treba biti snimljen 11 iz razloga što kanal 3 treba propustiti svih šest okvira dok kanal 4 treba propustiti pet okvira koji nemaju ID 201. Nakon završetka prve invokacija očekivan broj poruka je snimljen te se ona smatra uspješnom. Snimljene poruke mogu se vidjeti unutar MySQL baze pod tablicom *Message* prikazana na slici 20.

0.881767979	3	0.881767979 3 C9 Rx d 3 01 81 03	NULL	1	0	201	140864842
0.958082939	4	0.958082939 4 C8 Rx d 8 02 01 02 01 07 01 02...	NULL	1	0	200	140864843
0.958082939	3	0.958082939 3 C8 Rx d 8 02 01 02 01 07 01 02...	NULL	1	0	200	140864844
1.026996019	3	1.026996019 3 C7 Rx d 8 00 00 00 00 00 00 00...	NULL	1	0	199	140864845
1.026996019	4	1.026996019 4 C7 Rx d 8 00 00 00 00 00 00 00...	NULL	1	0	199	140864846
1.096769059	3	1.096769059 3 64 Rx d 4 07 1D 85 00	NULL	1	0	100	140864847
1.096769059	4	1.096769059 4 64 Rx d 4 07 1D 85 00	NULL	1	0	100	140864848
1.162712189	4	1.162712189 4 12C Rx d 1 00	NULL	1	0	300	140864849
1.162712189	3	1.162712189 3 12C Rx d 1 00	NULL	1	0	300	140864850
1.230519269	3	1.230519269 3 3FC Rx d 1 00	NULL	1	0	1020	140864851
1.230519269	4	1.230519269 4 3FC Rx d 1 00	NULL	1	0	1020	140864852

Slika 20. Tablica Message

Broj snimljenih okvira je 11, a okvir koji nedostaje je okvir s ID-em 201 na četvrtom kanalu. To se može vidjeti unutar drugog stupca koji prikazuje na kojem kanalu je snimljen okvir i sedmi stupac u kojem je prikazan ID okvira. Ne postoji red koji sadržava vrijednost kanala 4 i ID okvira koji ima vrijednost 201. Nakon daljnjeg izvođenja invokacija 12 sa sljedećim parametrima ne zadovoljava pretpostavku.

```
BaudRatePair (nominal=500, data=0),
Filters: [stop, True, CAN Single ID, 4, Channel, 0, 201 pass, True, CAN
Channel, 4, Channel, 0],
RecFormat: Natural,
Triggers:[],
messages_with_generated_signals: size=6,
database: motbusAll.xml"
```

Kako je prvi element iz liste filtra filter koji stopira okvire s ID-em 201 na kanalu 4, a drugi i posljednji filter unutar liste propušta sve okvire na kanalu 4 svi bi poslani okviri trebali biti propušteni. Filter koji filtrira okvire po kanalu ima prednost po važnosti filtriranja u odnosu na filtere po ID-u te ako korisnik aplikacije takav filter postavi na kraju on bi trebao nadvladati sve ostale filtere manje važnosti te filter koji stopira okvire s ID-em 201 ne bi trebao vrijediti. To se u ovome slučaju nije dogodilo te je pretpostavka koja provjera broj filtriranih okvira nije zadovoljena kao što je prikazano na slici 21.

test text	invoc integer	outcome text	error_step text	error text
test_parsing_xml_hypothesis	12	Failed	check_recorded_data	Filtered messages: 6, Trigger markers: 0, messages in table: 5, uid: n123160807qa184y

Slika 21. Ishod invokacije 12

Filter koji stopira okvire s ID-em 201 stopirao je poruku s ABSdata s ID-em 201 iako je trebao biti poništen postavljenjem filtera po kanalu koji je postavljen nakon njega. Nakon ispravke greške otkrivene na slici 21 i prolaska svih preostalih invokacija testa dodaje se funkcija za generiranje liste okidača. Funkcija *get_trigger_list()* temelji se na istom principu kao i funkcija *get_filter_list()* uz dodatno generiranje nasumičnih vrijednosti signala za koje će se okidač okinuti. U situaciji da je postavljen okidač koji se okida na signal CarSpeed iz okvira ABSdata gdje je brzina 100 km/h te se šalju CAN okviri ABSdata samo za one okvire čije podatkovno polje kada se parsira sadrži vrijednost 100 km/h okinuti će se marker okidač. Pokretanjem *test_parsing_xml_hypothesis()* testa s dodanim funkcijama *get_filter_list()* i *get_trigger_list()* dobivaju se sljedeći rezultati prikazani u PostgreSQL bazi unutar tablice *invocations*, Slika 22.

	id [PK] integer	build integer	start [PK] timestamp w	duration time without time zone	test text	invoc integer	outcome text	error_step text	error text
1	31407	10900	2023-05-03 13:...	00:00:17	test_parsing_xml_hypothesis	9	Passed		
2	31408	10900	2023-05-03 13:...	00:00:17	test_parsing_xml_hypothesis	10	Passed		
3	31409	10900	2023-05-03 13:...	00:00:17	test_parsing_xml_hypothesis	11	Passed		
4	31410	10900	2023-05-03 13:...	00:00:21	test_parsing_xml_hypothesis	12	Passed		
5	31411	10900	2023-05-03 13:...	00:00:22	test_parsing_xml_hypothesis	13	Failed	check_recorded_data	Wrong number number of trigger markers: recorded: 4, Should: 5
6	31412	10900	2023-05-03 13:...	00:00:22	test_parsing_xml_hypothesis	14	Failed	check_recorded_data	Wrong number number of trigger markers: recorded: 4, Should: 5

Slika 22. Ishod invokacija

Na slici 22 je vidljivo kako su prvih 12 invokacija bile uspješne dok trinaesta invokacija nije zadovoljila pretpostavku koja provjera broj markera okidača.

Ulazni parametri invokacije 12 su sljedeći.

```
BaudRatePair (nominal=100, data=2000),
Filters: [pass, True, CAN Channel, 3, Channel, 0, pass,
          True, CAN Channel, 4, Channel, 0],
RecFormat: Natural,
Triggers: [
    Marker, True, CAN ID Condition, 4, Signal ID, 237,
    Marker, True, CAN ID Condition, 4, Signal ID, 146,
    Marker, True, Symbolic CAN Signal, 3, Signal, FilteringCondition: Equal,
    ID: 1125, Signal: GlrFrHiBmSts, Value1: 1,
    Marker, True, CAN ID Condition, 3, Signal ID, 32,
    Marker, True, Symbolic CAN Signal, 3, Signal,
    FilteringCondition: InRange, ID: 41,
    Signal: ARDG1_LtVehAhdPolLatAccel, Value1: -2, Value2: -4],
messages_with_generated_signals: size=49,
database: GB_ASR_FCM_LC_22_22_150_MY220DX_RBS_Modified.arxml
```

Unutar liste okidača nalazi se pet okidača koji su postavljeni tako da se okidaju na jedan od CAN okvira iz ARXML baze. Poslani su svi okviri iz GR_ASR_FCM ARXML baze s takvim podatkovnim poljem da zadovoljavaju uvjet okidanja za postavljene okidače. Okidači koji imaju “CAN ID Condition” okidaju se na ID okvira dok okidači koji imaju “Symbolic CAN signal” kao uvjet okidanja okidaju se na vrijednost signala unutar podatkovnog polja. Kod detaljnijeg testiranja uviđa se kako se okidač za signal ARDG1_LtVehAhdPolLatAccel koji je trebao okinuti marker za vrijednosti signala između $-4 m^2$ i $-2 m^2$ za poslani CAN okvir čije podatkovno polje sadrži vrijednost signala ARDG1_LtVehAhdPolLatAccel $-3.85 m^2$ nije očekivano okinuo. Parsirana je očekivana vrijednost signala i ona kao takva ulazi u raspon mogućih vrijednosti za koje bi se okidač trebao okinuti, no to se nije dogodilo te potrebno pronaći grešku. Zadatak programskog testera je pronaći greške programskog koda te uputiti razvojne programere na pronađe greške sa što korisnijim informacijama ako je to moguće. Tako u ovom slučaju ako su dobro testani koraci parsiranja, a novi koraci koji provjeravaju okidače prouzrokuju greške početna sumnja bi trebala ići prema testiranju komponente koja je dodana naknadno prouzrokujući neuspješan ishod testa koji je prije bio uspješan za svih 50 invokacija. Zato se daljnje testiranje usmjerava prema provjeri konfiguracije okidača. Kod provjere konfiguracije uočena je greška postavljenog početnog bita za signal za koji je potrebno provjeriti vrijednost. Čvor okidača unutar XML konfiguracije pokazan je sljedećim ispisom:

```

<Trigger>
  <ID>1</ID>
  <ParentID xsi:nil="true" />
  <Type>Symbolic CAN Signal</Type>
  <Name>Trigger</Name>
  <Active>true</Active>
  <AllowPcapExpression>>false</AllowPcapExpression>
  <Action>Marker</Action>
  <Channel>CAN3</Channel>
  <ID1>41</ID1>
  <PDU_ID xsi:nil="true" />
  <DeltaTime xsi:nil="true" />
  <ExtendedID>>false</ExtendedID>
  <ConditionType>Signal</ConditionType>
  <SignalCondition>
    <ByteOrder>BigEndian</ByteOrder>
    <FrgIdxValue>3</FrgIdxValue>
    <StartBit>203</StartBit>
    <BitLength>9</BitLength>
    <ValueType />
    <ConversionFactor>0.125</ConversionFactor>
    <ConversionOffset>0</ConversionOffset>
    <Filtering Condition>InRange</Filtering Condition>
    <FilteringValue1>-4</FilteringValue1>
    <FilteringValue2>-2</FilteringValue2>
    <ID>0</ID>
    <Name>ARDG1_LtVehAhdPol_LatAccel</Name>
  </SignalCondition>
</Trigger>

```

Početni bit za signal ARDG1_LtVehAhdPolLatAccel postavljen je na vrijednost 203 dok je njegova prava vrijednost 195 kao što je prikazano na slici 23.

Name	Start Position
ARDG1_HstNxtLnLtCrv	117
ARDG1_HstNxtLnLtCrvDrvt	133
ARDG1_HstNxtLnLtHdg	149
ARDG1_HstNxtLnLtOfst	155
ARDG1_HstNxtLnLtVwRngLngDist	169
ARDG1_HstNxtLnLtTyp	172
ARDG1_HstNxtLnLtQty	174
ARDG1_HstNxtLnLtVwStrtLngDist	178
ARDG1_HstNxtLnLtWdth	187
ARDG1_LtVehAhdPolLatAccel	195

Slika 23. Prikaz točne vrijednosti početnog bita

Vrijednost je uzeta iz Autosar Vector Explorer View alata koji se koristi za prikaz CAN okvira, PDU kontejnera i signala iz ARXML baze podataka. Kako je postavljena kriva startna pozicija za signal unutar konfiguracije okidač će uzimati pogrešne vrijednosti za parsiranje signala što prouzrokuje nepravilan rad aplikacije.

5.3 Primjeri pronađenih grešaka pomoću testa parsiranja

Osim prethodno navedenih primjera, u nastavku će biti prikazane neke od pronađenih grešaka koje su pronađene izvođenjem prethodno opisanih postupaka od kojih je prva greška pronađena pomoću testiranja temeljenom na primjerima dok su druge dvije greške pronađene pomoću testiranja temeljenom na svojstvu:

- U jednoj od verzija aplikacije *Dashboard* parsiranje logova za *Ros-bag* format nikada ne završi s izvođenjem. Testni korak *parse_message* sadrži brojač koji ukoliko vrijeme trajanja parsiranja prijeđe vrijeme trajanja od jedne minute testni korak vraća grešku te se smatra neuspjelim. Kod detaljnijeg debugiranja na .NET strani otkriveno je kako ulazak u granu koja prekida dretvu za parsiranje logova za *Ros-bag* format nije ostvaren.
- Izračunate su pogrešne vrijednosti signala za podatke koji sadrže redoslijed bitova Motorola. Motbus baza podataka ne sadrži takve CAN FD okvire no druge baze ih sadrže te pojedini okviri sadrže signale kojima je poredak bajtova takav da se najznačajniji bajt nalazi na početku. Kod generiranih parametara pomoću *Hypothesis* strategija s takvim signalima otkrivena je greška kod koje su izračunate pogrešne vrijednosti za takve signale. Na slici 37 je prikazan ispis s konzole na kojoj se može vidjeti kako pretpostavka koja uspoređuje vrijednosti signala izračune od strane *Dashboard* aplikacije i vrijednosti izračunate unutar testa nije uspjela.

```
assert parsed_value == value, f"Wrong signal Value, parsed:
{parsed_value},
should {value},"
AssertionError: Wrong signal Value, parsed: -31.22, should -2.42,
signal name: CamHeadingAngleObject00_ITS
```

Na slici 24 je prikazana je parsirana vrijednost unutar SignalView tablice i vrijednosti koju je prikazao CANalyzer alat. [24]

Dashboard parsed:

Timestamp	MessageType	Channel	MessageName	SignalName	Value	Unit	IDSignal	IDRawmessage	LogString
1.29503783	1	3	FrCamera_ITS_RD13	CamHeadingAngleObject00_ITS	-31.22	rad	237121316	140864875	1.295037830 3 291 Rx d 8 26 70 D6 00 14 90 9C 57

CANalyzer parsed:

Time	Chn	ID	Name	Event Type	Dir	DLC	Da...	Data
0.836957	CAN 1	2EE	FrCamera_ITS_A41	CAN Frame	Rx	8	8	80 00 00 00 00 61 70
1.836957	CAN 1	291	FrCamera_ITS_RD13	CAN Frame	Rx	8	8	26 70 D6 00 14 90 9C 57
			CRC_FrCamera_ITS_RD13_ITS	38	26			
			CamAngleRateVARObject00_ITS	53.5000 (deg/s)^2	68			
			CamASILLLevelObject00_ITS	0	0			
			Clock_FrCamera_ITS_RD13_ITS	7	7			
			CamObjectToLineAngleObject00_ITS	-0.3420 rad	292			
			CamHeadingAngleObject00_ITS	-2.4200 rad	4E			
			CamLeftLineEndViewRange_ITS	21.7500 m	57			

Slika 24. Usporedba vrijednosti CamHeadingAngleObject00_ITS signala

- Nemogućnost parsiranja naziva okvira za 64 bitne vrijednosti signala. Kod generiranja vrijednosti za 64 bitne signale strategija generira velike vrijednosti koje su unutar Dashboard aplikacije izazvale iznimku kod produživanja tako velikih vrijednosti unutar varijabli koje su deklarirane kao vrijednosti tipa *decimal* umjesto *double*. Za takve slučajeve iznimka bi se uhvatila s ispisanom porukom „Unable to set message Name.“ koja bi se postavila kao vrijednost za naziv okvira te bi pretpostavka unutar *check_recorder_data()* (Programski kod 17) testnog koraka koja provjerava naziv okvira završila neuspješnom.


Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković

```
assert table_signal['MessageName'] == signal.message_name, \
    f"Wrong name, message: {table_signal['MessageName']}"
```

Programski kôd 17. Provjera imena parsiranog CAN okvira

Ispis greške uzrokovan iznimkom ispisan je unutar stupca Parsed Message koji se nalazi u prozoru za prikazivanje parsiranih vrijednosti nadolazećeg CAN prometa unutar *Dashboard* aplikacije.

Dashboard result:

Channel: LR1_CAN3 Database: CADM_P Filters 

Timestamp	Parsed Message
06.12.2022 16:37:24.6345	MRR_RL_Status_Radar: RadarAlignmentNotStarted_MRRRL=Radar align not started (1): RadarAlignmentIncomplete_MRRRL=...
06.12.2022 16:37:25.3063	MRR_RL_Status_Radar: RadarAlignmentNotStarted_MRRRL=Radar align not started (1): RadarAlignmentIncomplete_MRRRL=...
06.12.2022 16:38:01.8177	MRR_RL_Status_Radar: RadarAlignmentNotStarted_MRRRL=Radar align not started (1): RadarAlignmentIncomplete_MRRRL=...
06.12.2022 16:40:06.0725	Unable to set message name, CANFD network is not defined or error has occurred: RadarAlignmentNotStarted_MRRRL=Ra...
06.12.2022 16:41:14.0718	Unable to set message name, CANFD network is not defined or error has occurred: RadarAlignmentNotStarted_MRRRL=Ra...

Slika 25. Prikaz greške unutar Dashboard aplikacije

Navedeni slučajevi su neki od tipičnih primjera pogrešaka kojim se opisuju neke od karakteristika testnog okruženja za provjeru parsiranja CAN okvira kao dijela razvojnog okruženja u kojem se provodi testiranje sustava. Korištenjem testa napisanih u PyTest okruženju otkrivaju su greške koje uzrokuju nepravilan rad unutar šireg okvira funkcionalnosti *Dashboard* aplikacije. Dodavanjem *Hypothesis* biblioteke unutar takvog okruženja testovi postaju temeljitiji te se otkrivaju greške koje uzrokuju nepravilan rad aplikacije za specifične slučajeve.

6. Zaključak

Kada se uz valjanu strategiju i uspješan plan testiranja ukomponiraju razvojni okviri kao što su PyTest i *Hypothesis* koji služe za izradu automatiziranog testiranja sustava moguće je napraviti proizvod koji će kontinuirano testirati svojstva softvera. Automatizacijom testa pomoću skripti dobiva se na brzini izvođenja, uštedi vremena gdje računalno odrađuje većinu posla, učestalosti i smanjenom utjecaju ljudskog faktora na greške. Kako se testovi koji se koriste za automatsko testiranje napisani od strane testera ne bi uvijek izvršavali testiranje s istim ulaznim parametrima i postali predvidljivi i neučinkoviti poželjno je pisati testove koji su utemeljeni na svojstvima. Takvo testiranje pokazalo je veću uspješnost testiranja programskog koda gdje su testovi podložni ulaznim parametrima koji su generirani izvan okvira razmišljanja autora testa. Testiranje koda jednako je bitno kao i izrada same programske aplikacije. Ukoliko se požuruje s isporukom aplikacija dok nisu testirane sve komponente i svi mogući ulazni parametri koje korisnik može unijeti potencijalno se može isporučiti nepravilna aplikacija. To dovodi do općeg nezadovoljstva, a proizvod kao takav smatra se nepouzdanim te se gubi povjerenje korisnika. Testovima temeljenim na svojstvima koji testiraju ispravnost parsiranja CAN okvira prikazanih u ovome radu nađeno je mnogo grešaka koje bi bilo teško detektirati ručnim testiranjem zbog velikog broja dostupnih CAN baza podataka koje sadrže velike količine CAN okvira od kojih je u radu prikazano samo nekoliko njih. Svakodnevnim izvođenjem testa, pogreške se lakše mogu detektirati te se time skraćuje vrijeme razvoja novih inačica aplikacije za daljnja testiranja. *Hypothesis* biblioteka pokazala se kao dobar alat za kreiranje testova temeljenih na svojstvima zbog svoje proširivosti i mogućnosti prilagodbe prema vlastitim potrebama testiranja. Testiranje temeljeno na svojstvima ponajviše je doprinijelo temeljitosti u testiranju *Dashboard* komponenti te se za daljnje unaprjeđenje projekta planira ukomponirati takva vrstu testiranja za ostala sučelja komponente *Dashboard* aplikacije.

7. Literatura

- [1] Glenford J. Myers: The Art of Software Testing, Third Edition, Wiley, 2004, ISBN 978-1-118-03196-4
- [2] Alan Jović, Nikolina Frid, "Procesi programskog inženjerstva", https://www.fer.unizg.hr/download/repository/Procesi_programskog_inzenjerstva_3_izdanje%5b1%5d.pdf, pristupano 23.3.2023.
- [3] Robert Manger, Softversko inženjerstvo, Prirodoslovno-matematički fakultet Sveučilišta u Zagrebu, Predavanja 2022/2023, Poglavlje 4: Verifikacija i validacija, <http://web.studenti.math.pmf.unizg.hr/~manger/si/>, pristupano 18.5.2023
- [4] Software Testing Help (2021). Software Development And Testing Methodologies (With Pros And Cons) [Online]. Software Testing Help, <https://www.softwaretestinghelp.com/software-development-testing-methodologies>, pristupano 17.5.2023.
- [5] Acharya, S., Pandya, V. (2012). Bridge between Black Box and White Box–Gray Box Testing Technique, International Journal of Electronics and Computer Science Engineering, ISSN: 2277-1956
- [6] Alka Bamotra i Amanjot Kaur Randhawa (2017), International Journal of Innovative Computer Science & Engineering, "Software testing techniques", ISSN: 2393-8528
- [7] R. M. Sharma (2014). Quantitative analysis of automation and manual testing. International Journal of Engineering and Innovative Technology , ISSN: 2277-3754
- [8] Allen B. Downey, "Think Python", ISBN: 9781449330729
- [9] Brian Okken, "Python Testing with pytest, Second Edition", Pragmatic Bookshelf, 2022, ISBN: 9781680508604

- [10] Holger Krekel, "Full pytest documentation",
<https://buildmedia.readthedocs.org/media/pdf/pytest/latest/pytest.pdf>,
pristupano 10.8.2022
- [11] ITEA 4, Quviq AB, <https://itea4.org/organisation/11130-quviq-ab.html>,
pristupano 19.5.2023
- [12] John Hughes, "Experiences with QuickCheck: Testing the Hard Stuff and Staying Sane", [Online]. Available:
<https://www.cs.tufts.edu/~nr/cs257/archive/john-hughes/quviq-testing.pdf>
- [13] David R. MacIver, Welcome to Hypothesis! — Hypothesis 6.75.3 documentation, <https://hypothesis.readthedocs.io/en/latest/>, pristupano 29.8.2022
- [14] Mickey Petersen, "Testing your Python Code with Hypothesis",
<https://www.inspiredpython.com/course/testing-with-hypothesis/testing-your-python-code-with-hypothesis>, pristupano 29.8.2022
- [15] Xylon company, logiRecorder 3.2 AUTOMOTIVE HIL VIDEO LOGGER,
<https://xylon-lab.com/product/logirecorder/>, pristupano 3.1.2023
- [16] Fred Hebert, "Property-Based Testing with PropEr, Erlang, and Elixir", 2019, ISBN: 9781680506211, [Online-free version]. Available:
<https://www.propertesting.com/>
- [17] Steve Campbell, "PyTest Tutorial: What is, How to Install, Framework, Assertions", <https://www.guru99.com/pytest-tutorial.html>, pristupano 9.8.2022.
- [18] Grant Maloy Smith, "What is ADAS (Advanced Driver Assistance Systems)?",
<https://dewesoft.com/blog/what-is-adas>, pristupano 17.5.2022
- [19] MathWorks, Basics of Hardware-in-the-Loop simulation,
<https://www.mathworks.com/help/simscape/ug/what-is-hardware-in-the-loop-simulation.html>, pristupano 17.5.2023
- [20] Xylon company, logiRecorder DASHBOARD, <https://xylon-lab.com/product/logirecorder-dashboard-2/>, pristupano 3.1.2023.

- [21] Wilfried Voss, A Comprehensible Guide to Controller Area Network, Copperhill Media Corporation, 2005, ISBN-10: 0976511606
- [22] Jeff Shepard, What is the controller area network (CAN) bus?, <https://www.microcontrollertips.com/what-is-the-controller-area-network-can-bus-faq/>
- [23] CSS Electronics, CAN FD Explained - A Simple Intro, <https://www.csselectronics.com/pages/can-fd-flexible-data-rate-intro>
pristupano 19.2.2023
- [24] Samir Bhagwat, "Understanding AUTOSAR ARXML for Communication Networks, Interpid Control Sytems, 2019, https://www.intrepidcs.net.cn/wp-content/uploads/2019/05/202_Understanding_ARXML_EEA_COM_TD_USA_2019.pdf, pristupano 9.1.2023
- [25] CSS Electronics, CAN DBC File Explained – A Simle Intro, <https://www.csselectronics.com/pages/can-dbc-file-database-intro>, pristupano 25.3.2023
- [26] Oliver Garnatz / Peter Decker, "CAN FD with dynamic multi-PDU-to-frame mapping", Vector Informatik GmbH, 2015
- [27] Vector Informatik GmbH, Vector Logger Suite Manual, <https://www.vector.com/int/en/download/manual-vector-logger-suite/>,
Pristupano 10.1.2023
- [28] Vector Informatik GmbH, CANalyzer Quickstart, https://cdn.vector.com/cms/content/products/VectorCAST/Events/TechNights/CANalyzer_QuickStart.pdf, 10.1.2023

Summary

This masters' thesis paper deals with a methodology of testing the program code as well as the importance of continuous testing the program system. It is focused on the topic of detailed property-based testing, where a large number of test cases that can easily be missed during usual example-based testing are passed through. It is stated which development frameworks and programming languages enable this type of testing and its advantage over others. In the practical work, the implementation of automated testing within the Python environment was demonstrated over the system consisting of the logiRECORDER device and the Dashboard application that manages the device. The testing is divided into two parts, first demonstrating example-based testing using the Pytest development framework, and then property-based testing using the Pytest framework and the Hypothesis library. The aim of the paper is to provide insight into the use of selected tools for automated testing, to show their application in an industrial environment, and to test the contribution of property-based testing within such an environment.

Keywords: test, automatization, PyTest, Hypothesis, parameter, strategy, CAN